Midterm 1 exam

In-class midterm. 2:00-2:15pm on Monday Feb 25.

- No makeups.

Rules.

- Closed book, closed note.
- Covers all material thru Lecture 8.
- No computers or other computational devices.
- 8.5-by-11 cheatsheet (one side, in your own handwriting).

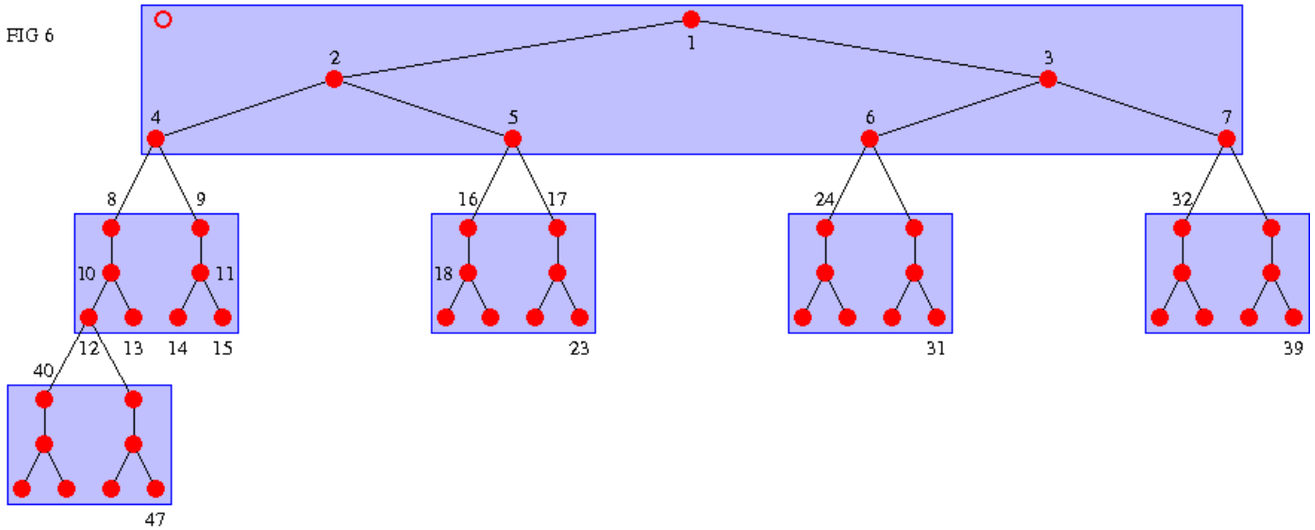
including associated assignments (but no serious Java programming)

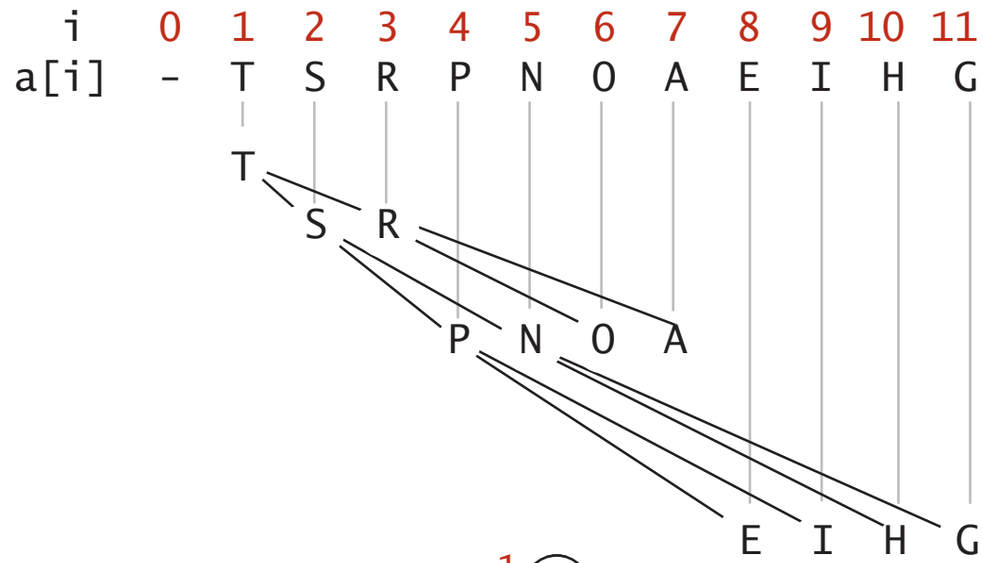
Midterm preparation.

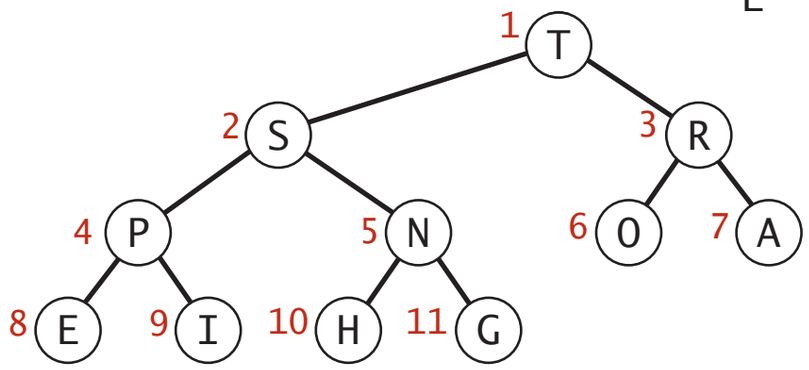
- Review session: Go to recitations.

Binary heap: practical improvements

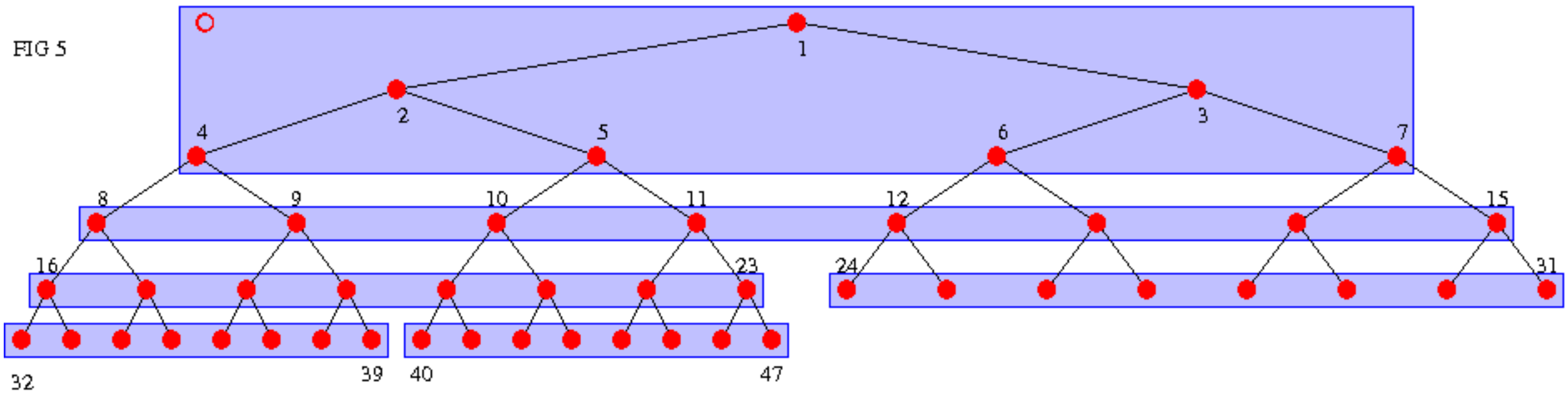
Caching. Binary heap is not cache friendly.







Heap representations



2-3 TREES

2-3 tree

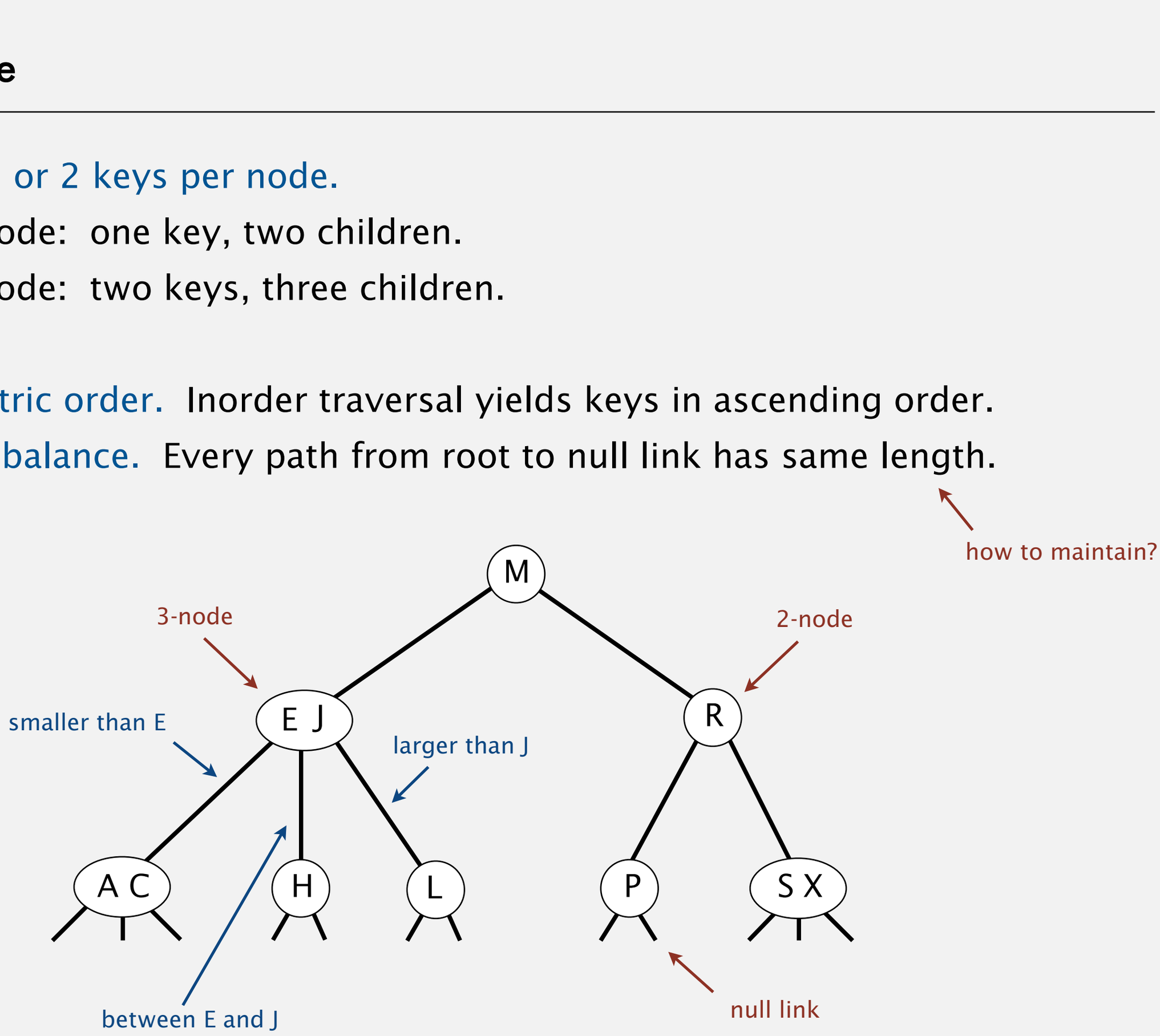
Allow 1 or 2 keys per node.

- 2-node: one key, two children.
- 3-node: two keys, three children.

Symmetric order. Inorder traversal yields keys in ascending order.

Perfect balance. Every path from root to null link has same length.

how to maintain?

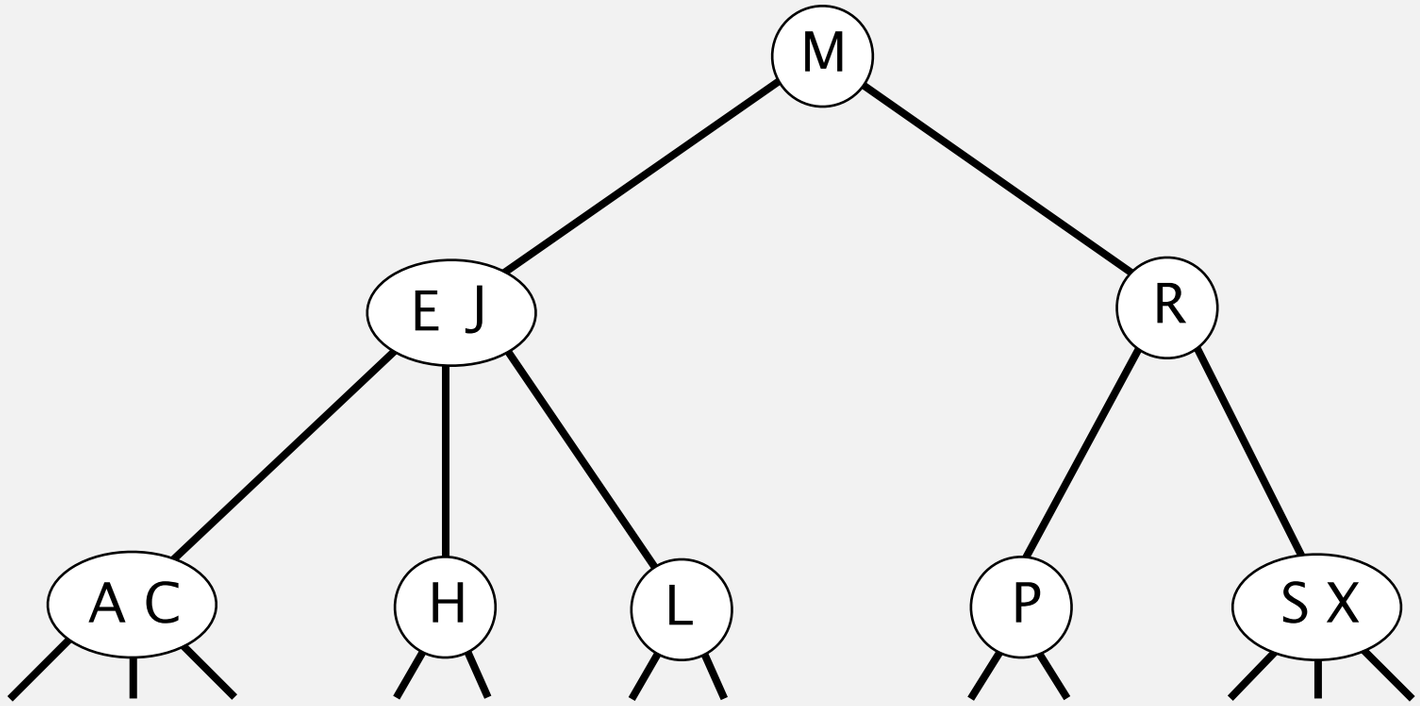


2-3 tree demo

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

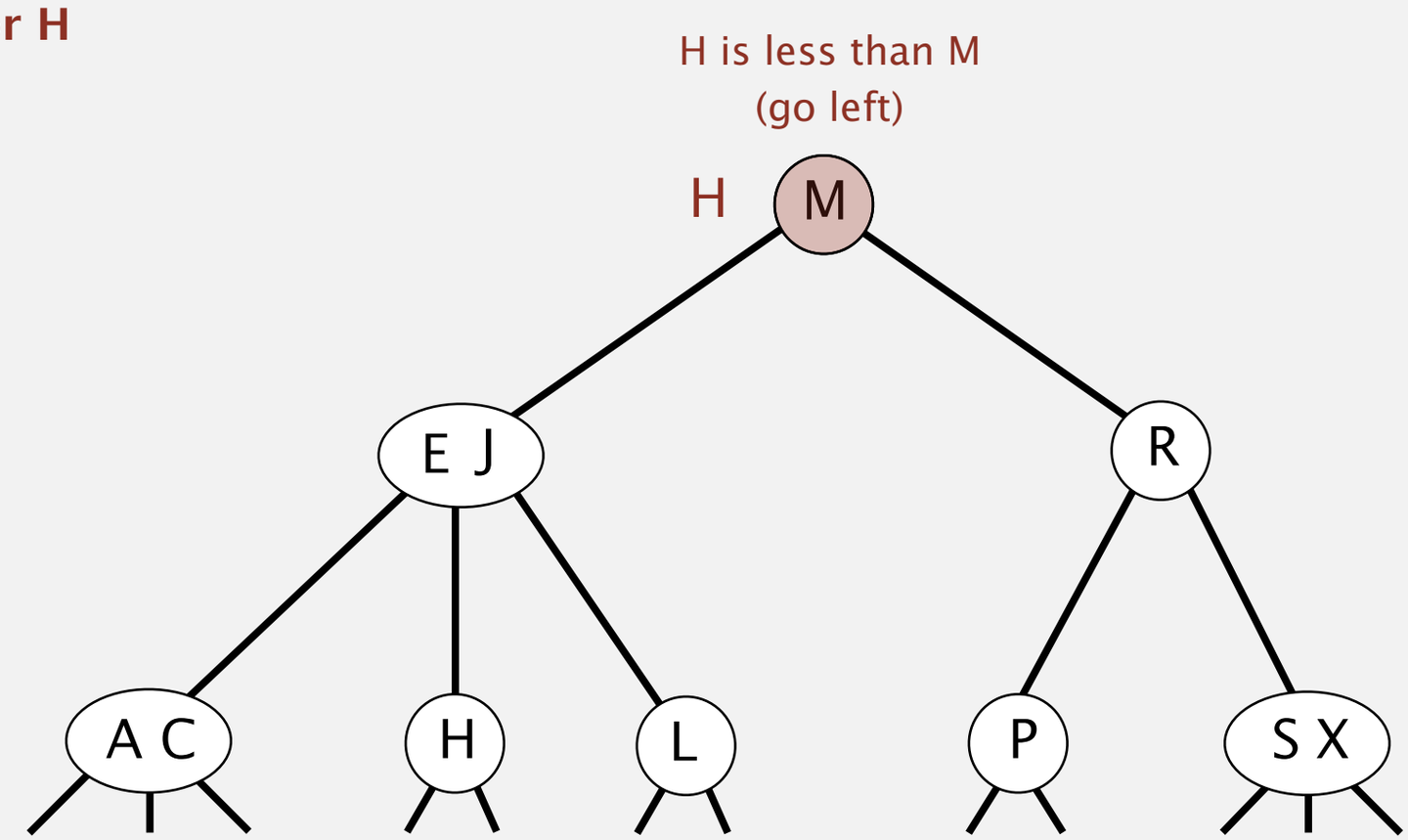
search for H



2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

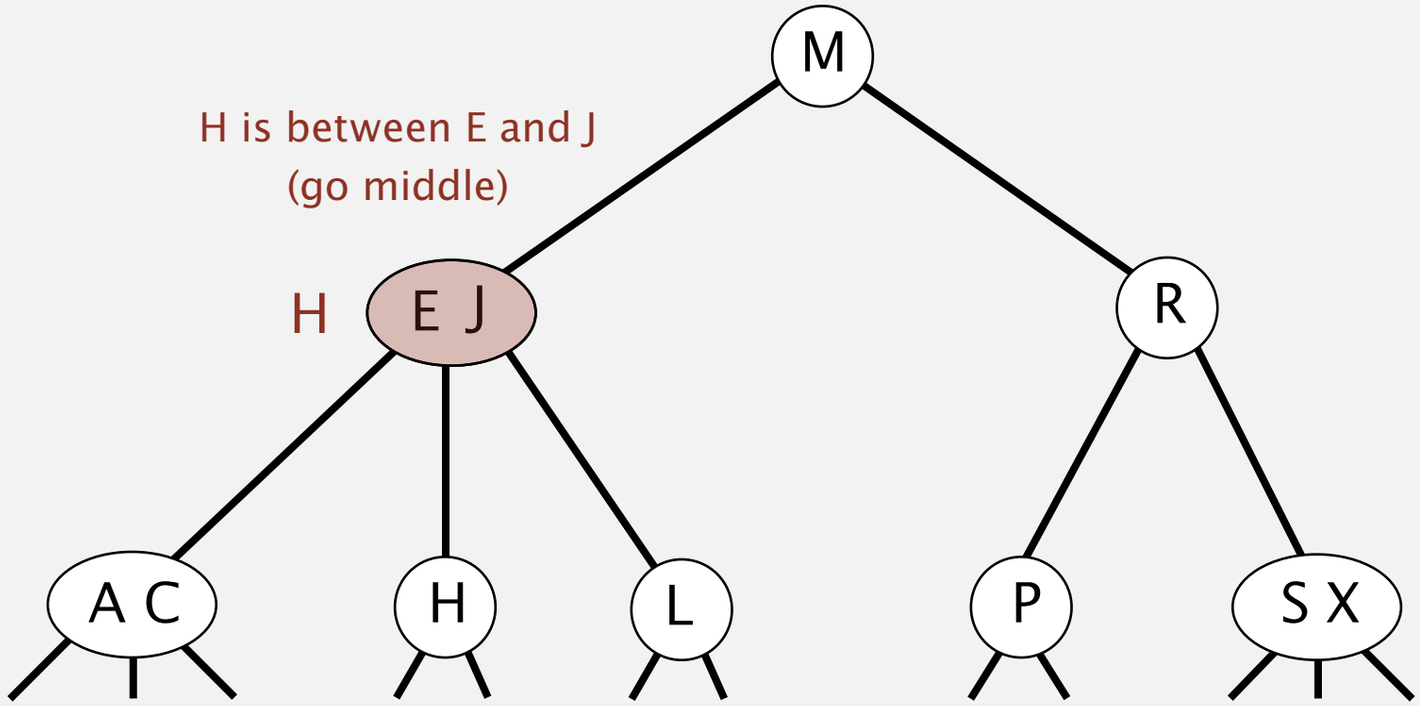


2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for H

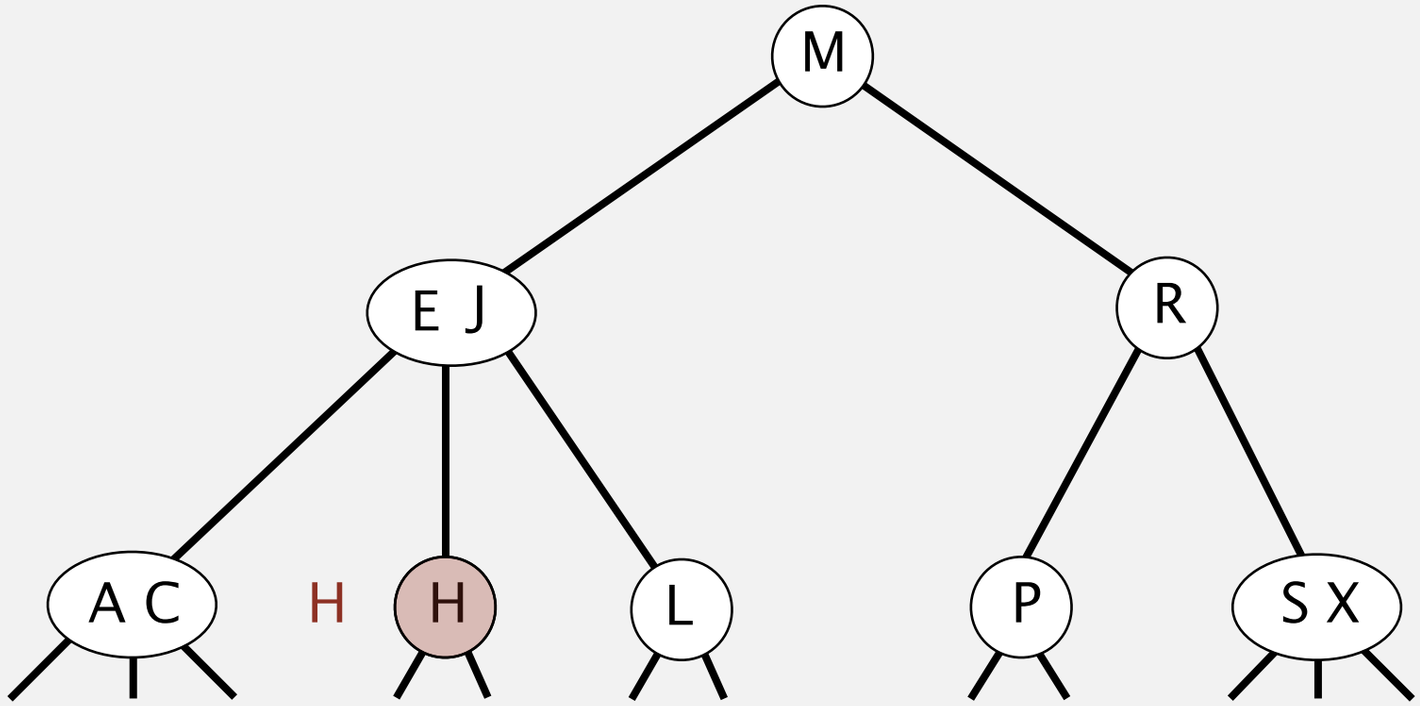


2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for H

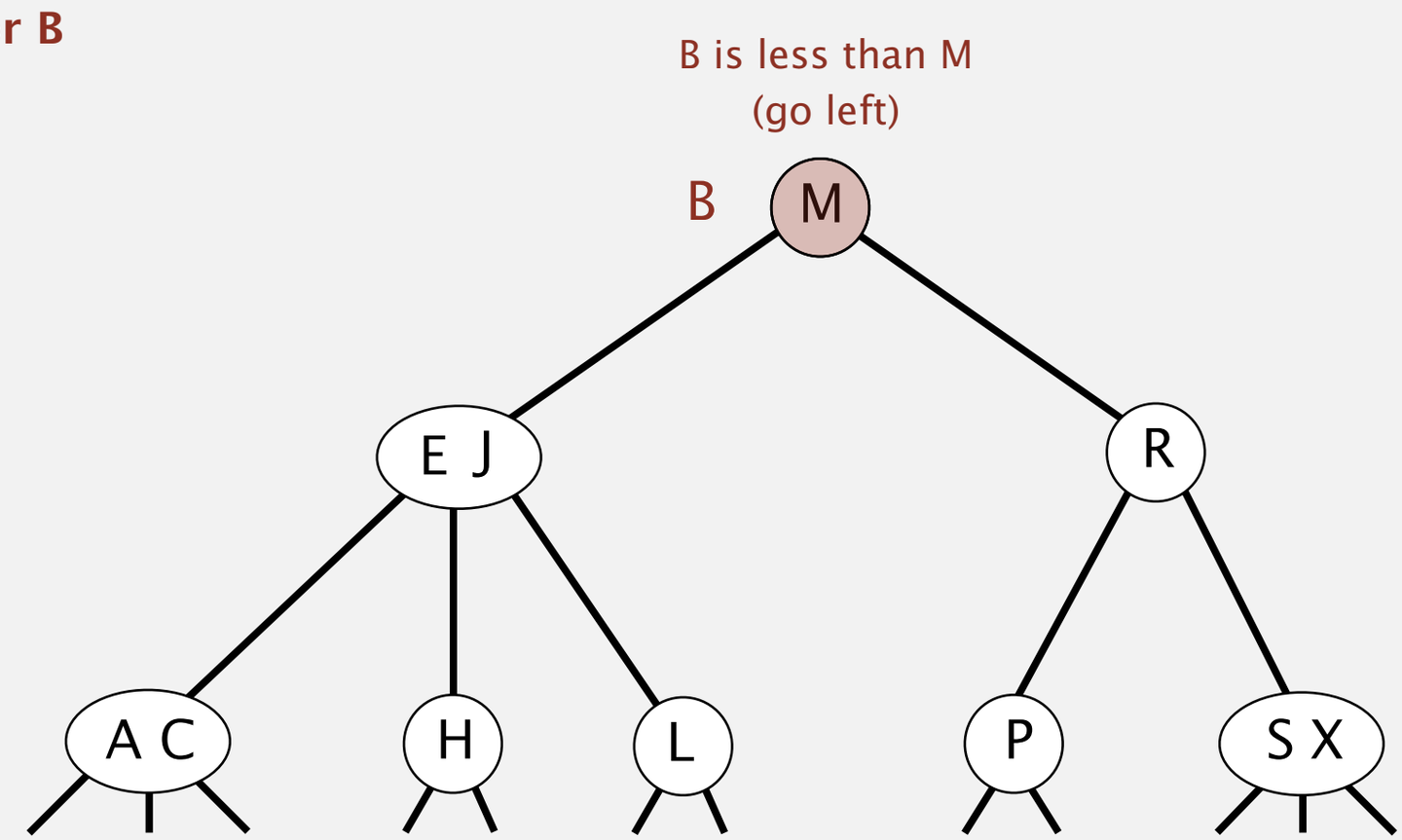


found H
(search hit)

2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

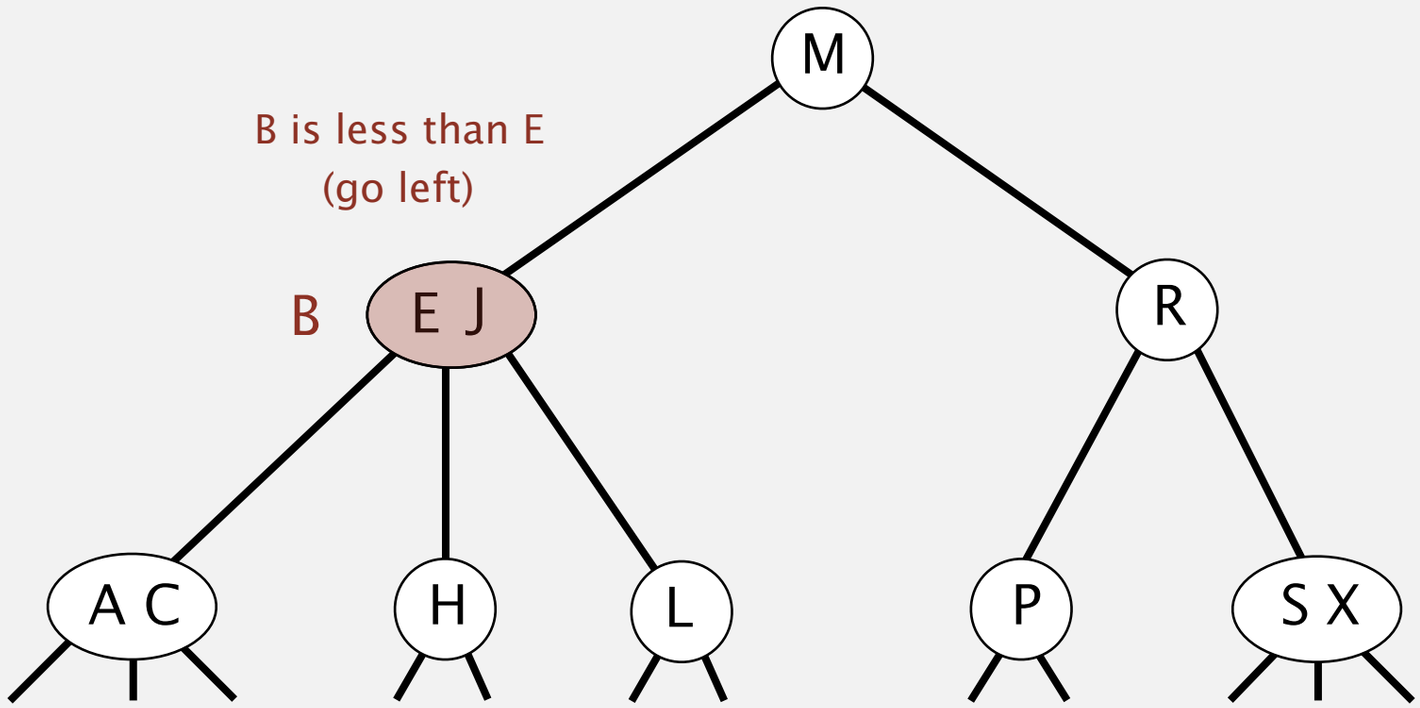


2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for B

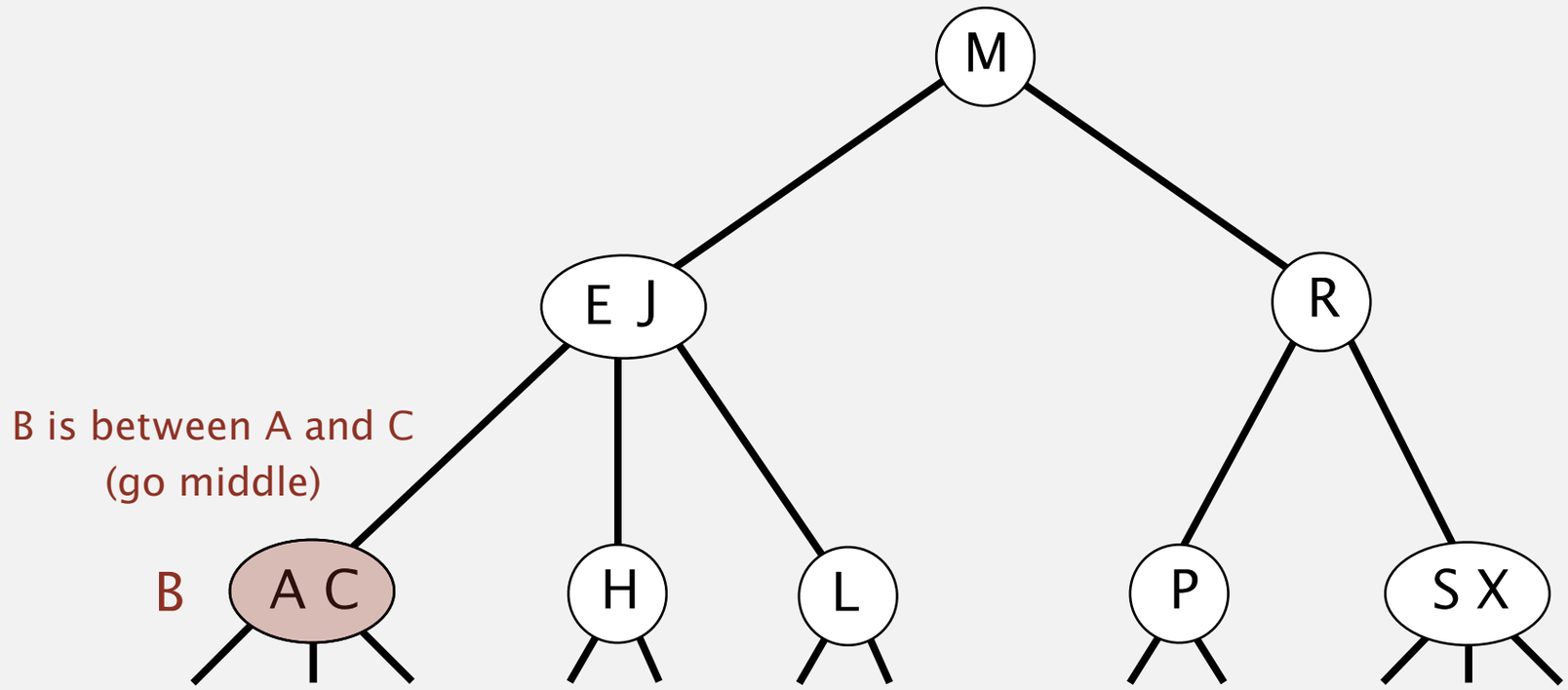


2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for B

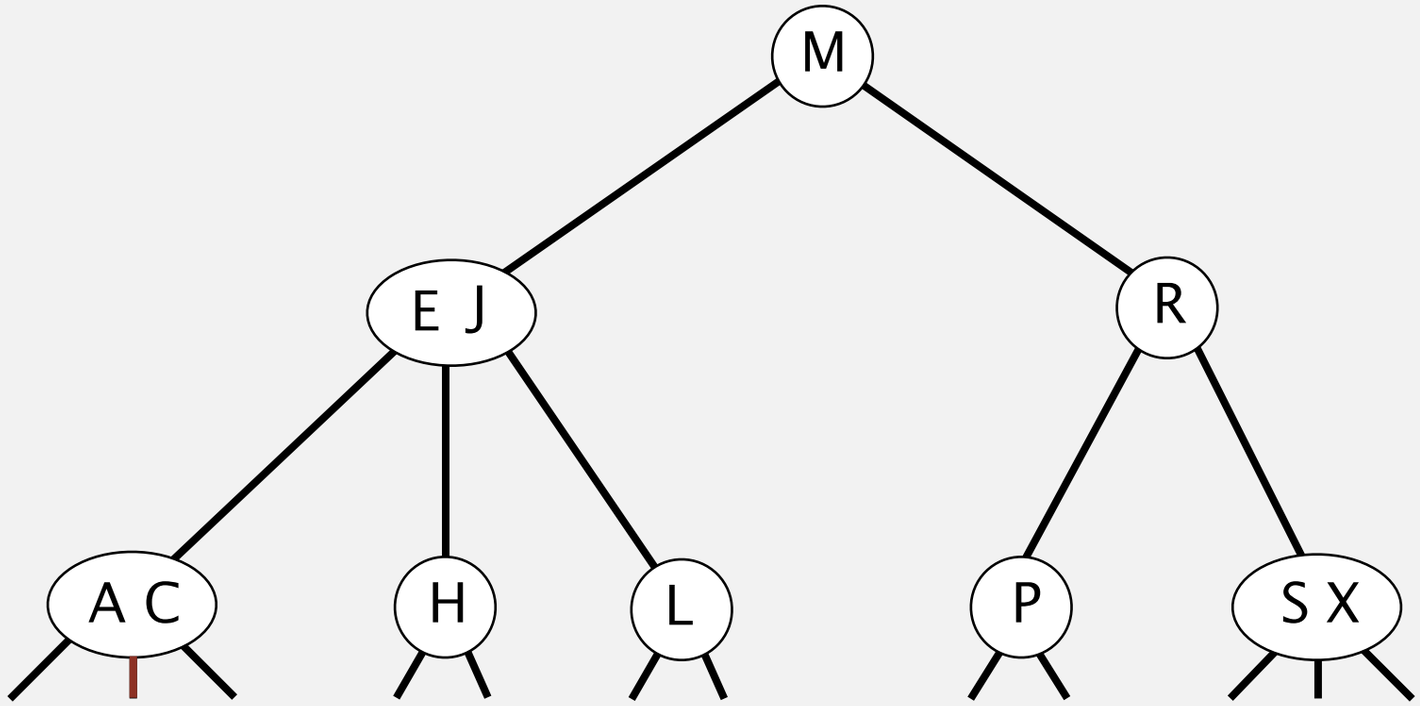


2-3 tree demo: search

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for B



B

link is null
(search miss)

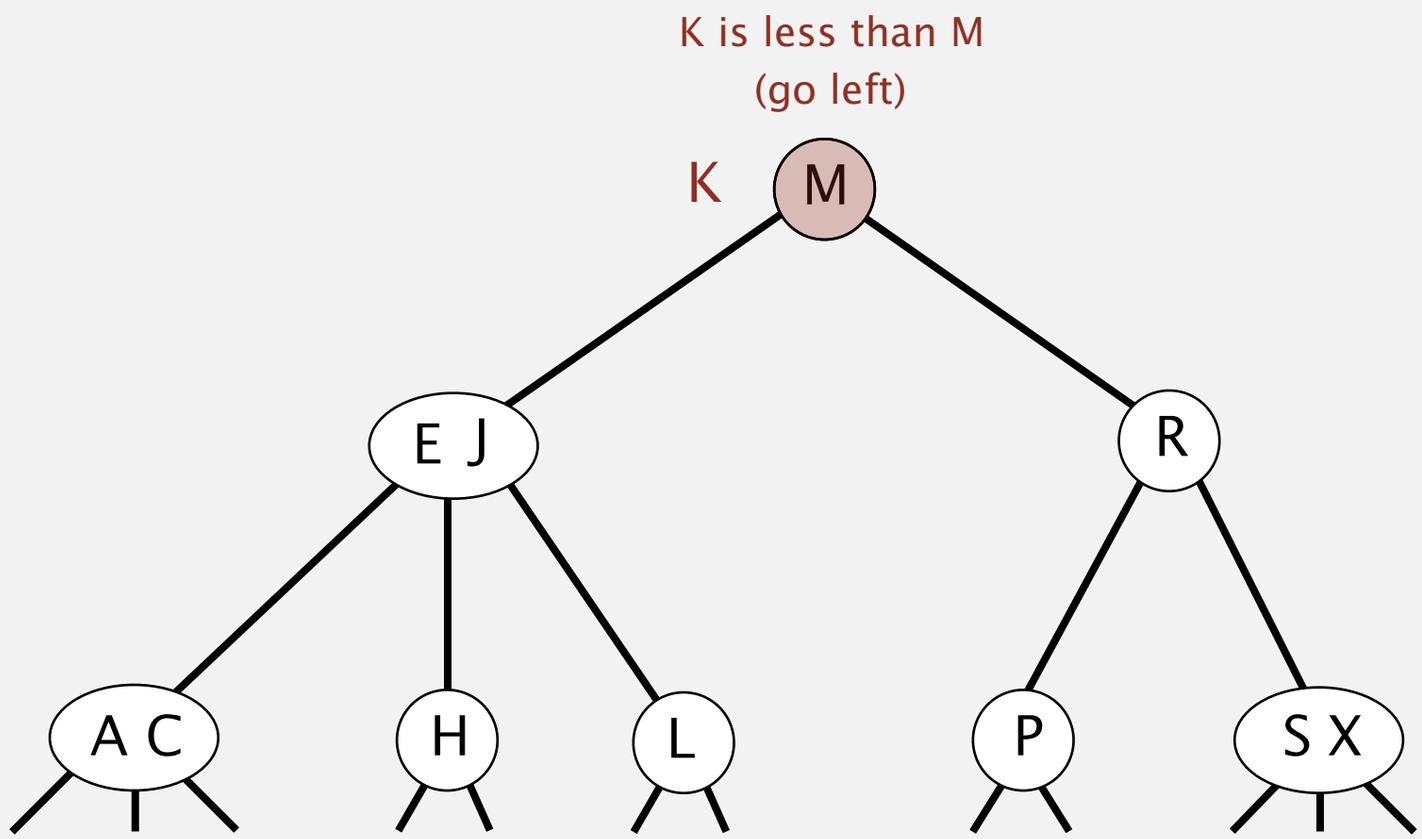
2-3 TREES

Inserting Values

2-3 tree demo: insertion

Insert into a 2-node at bottom.

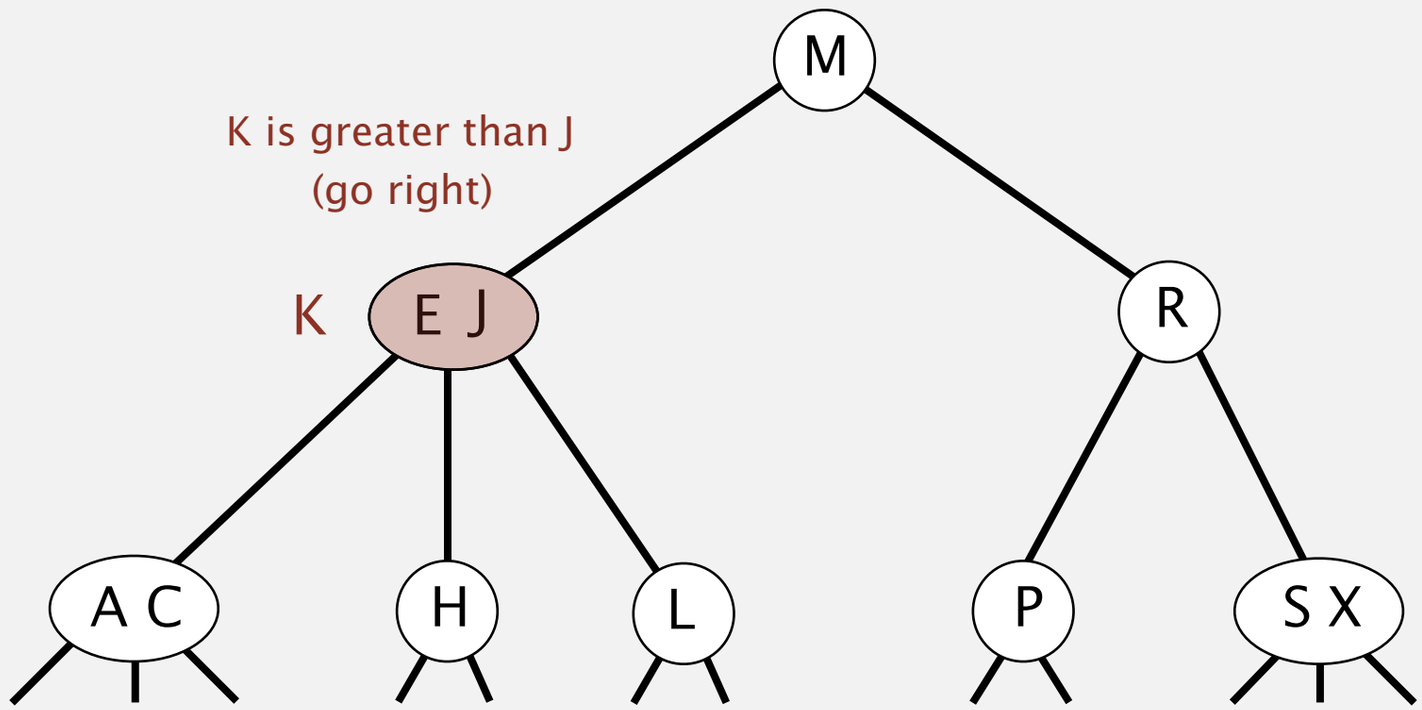
- Search for key, as usual.
- Replace 2-node with 3-node.



2-3 tree demo: insertion

Insert into a 2-node at bottom.

- Search for key, as usual.
- Replace 2-node with 3-node.

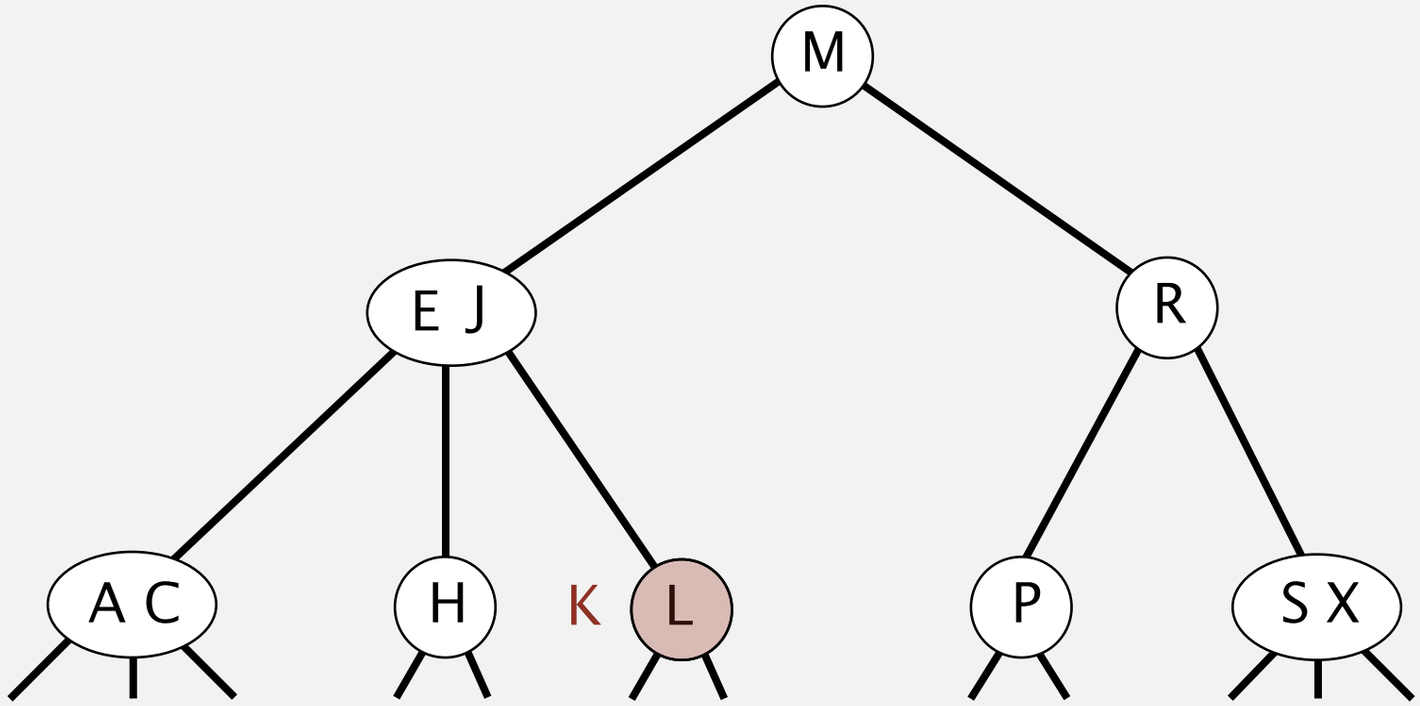


2-3 tree demo: insertion

Insert into a 2-node at bottom.

- Search for key, as usual.
- Replace 2-node with 3-node.

insert K



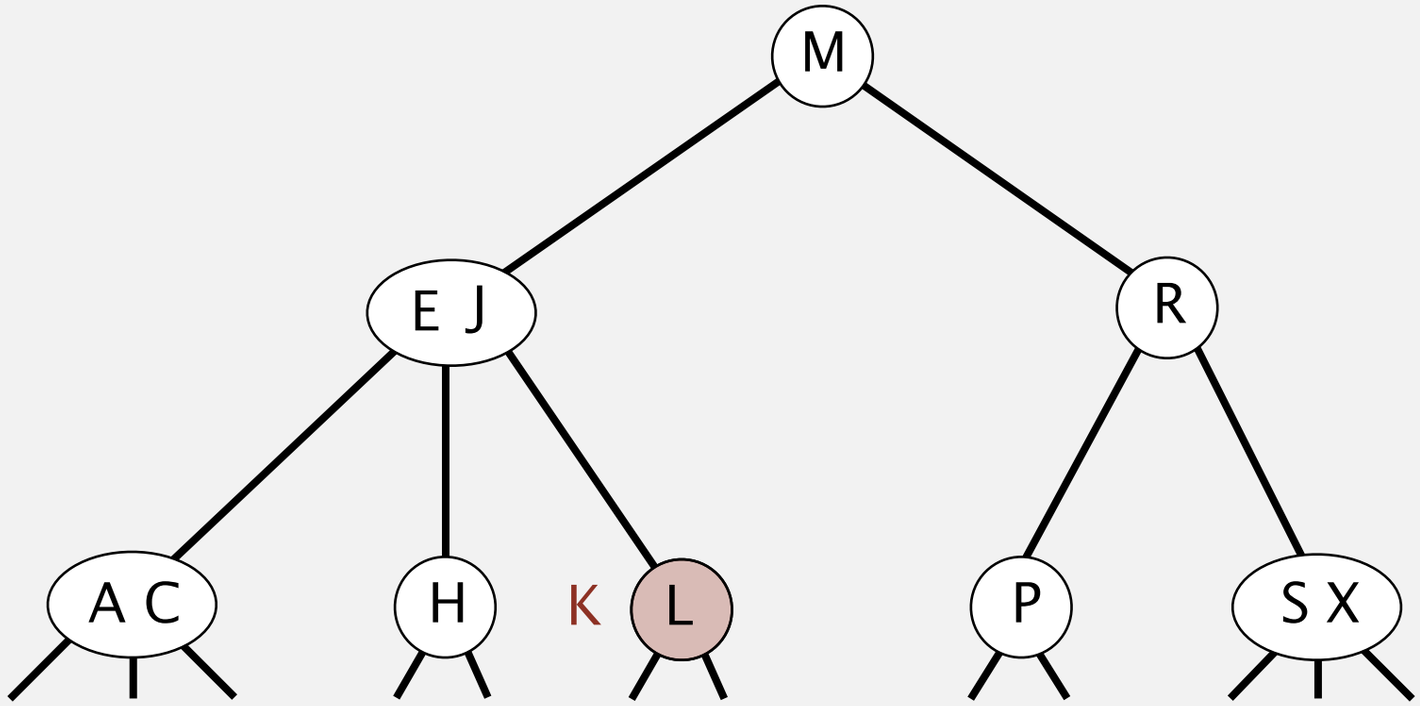
search ends here

2-3 tree demo: insertion

Insert into a 2-node at bottom.

- Search for key, as usual.
- Replace 2-node with 3-node.

insert K



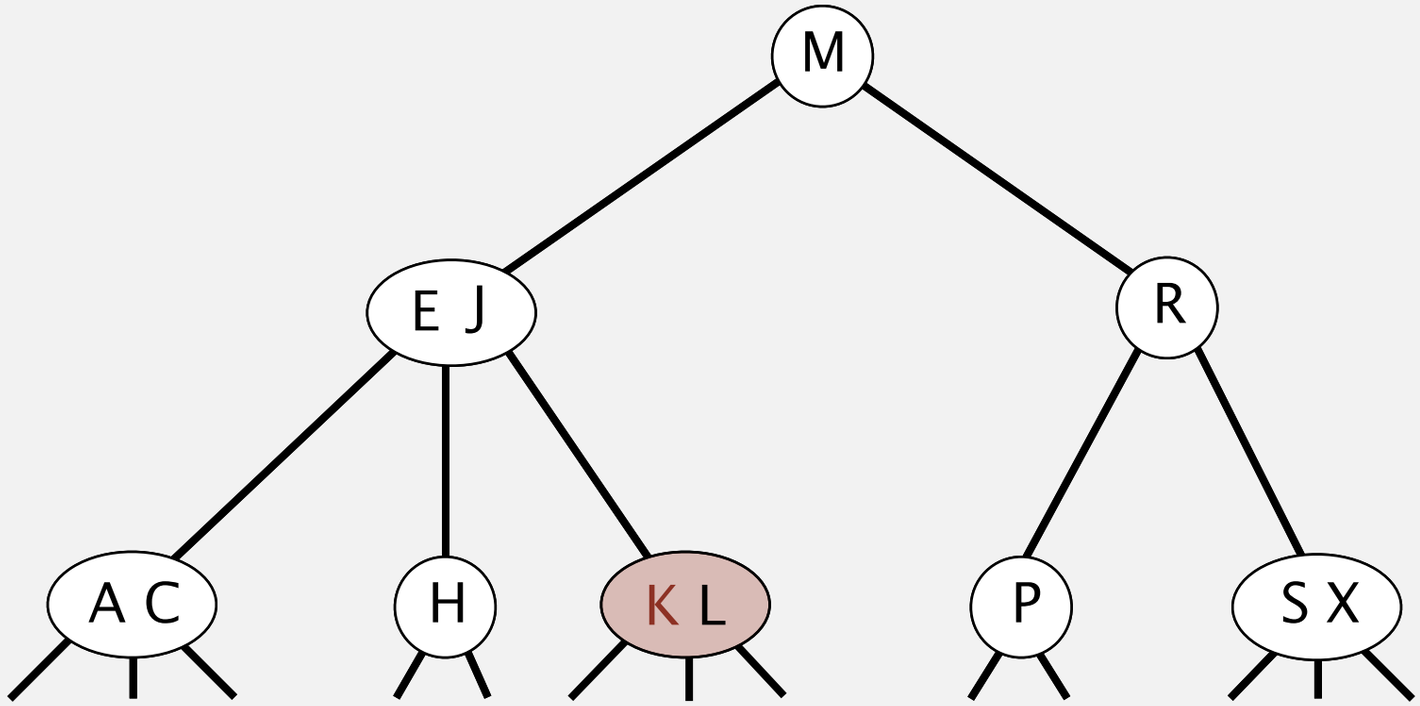
replace 2-node with
3-node containing K

2-3 tree demo: insertion

Insert into a 2-node at bottom.

- Search for key, as usual.
- Replace 2-node with 3-node.

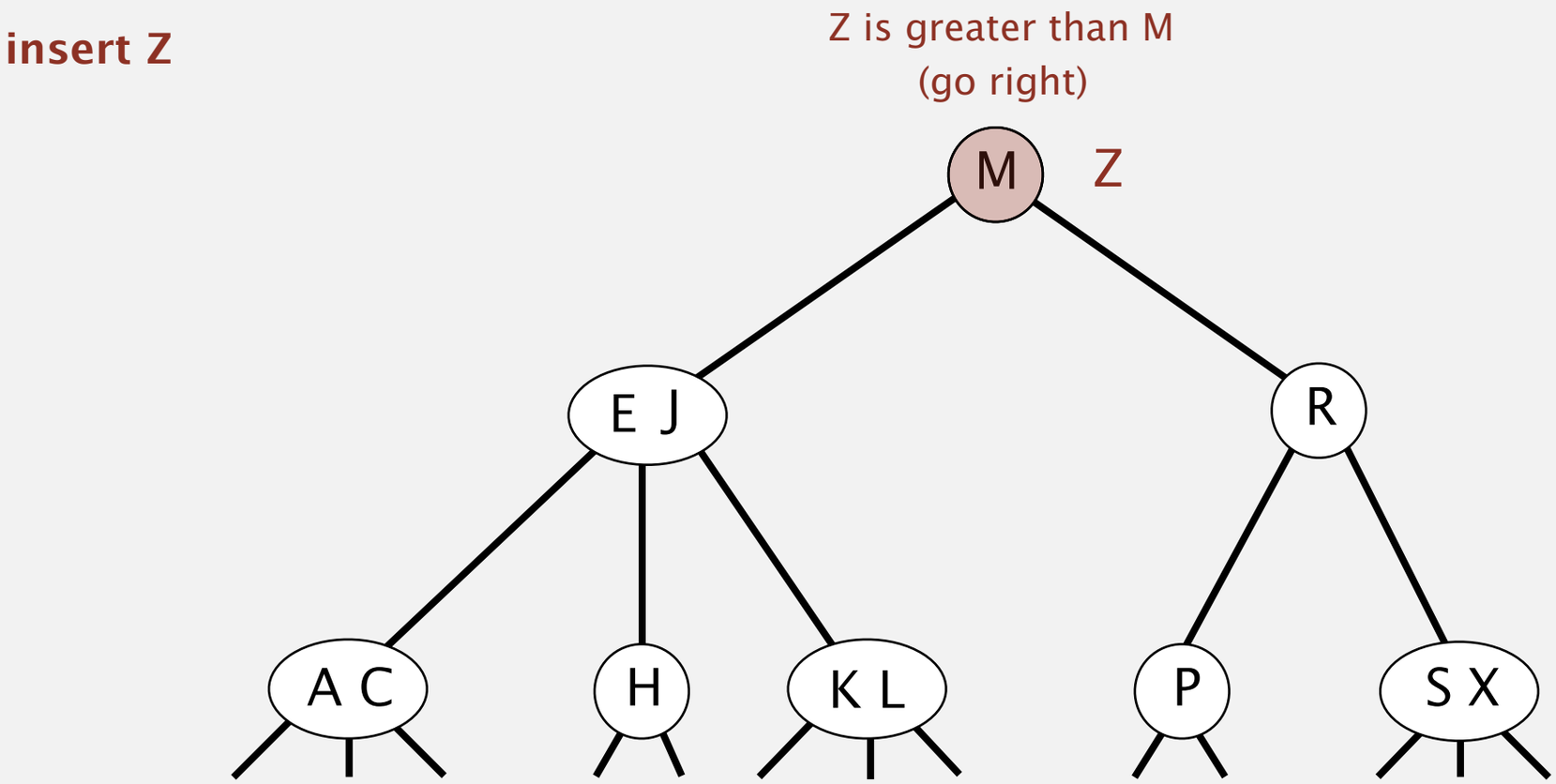
insert K



2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

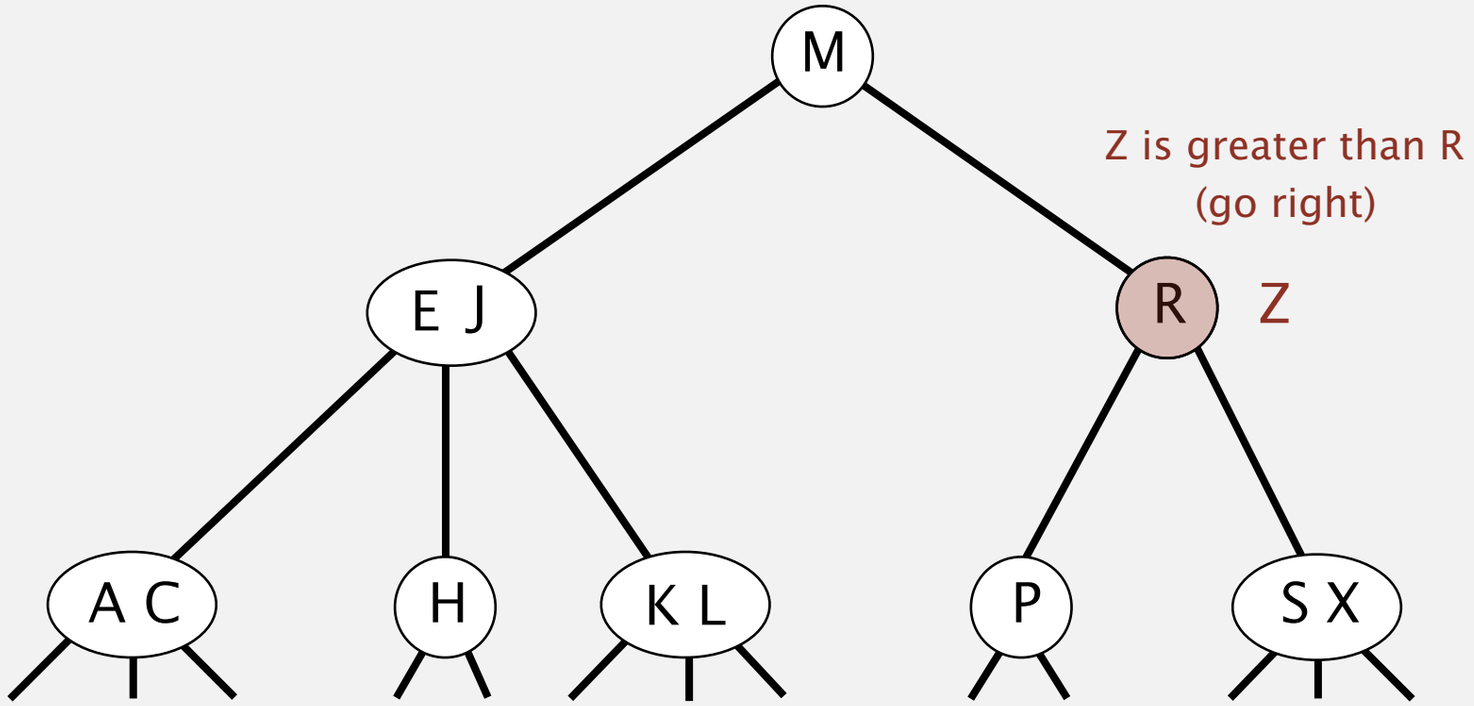


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z

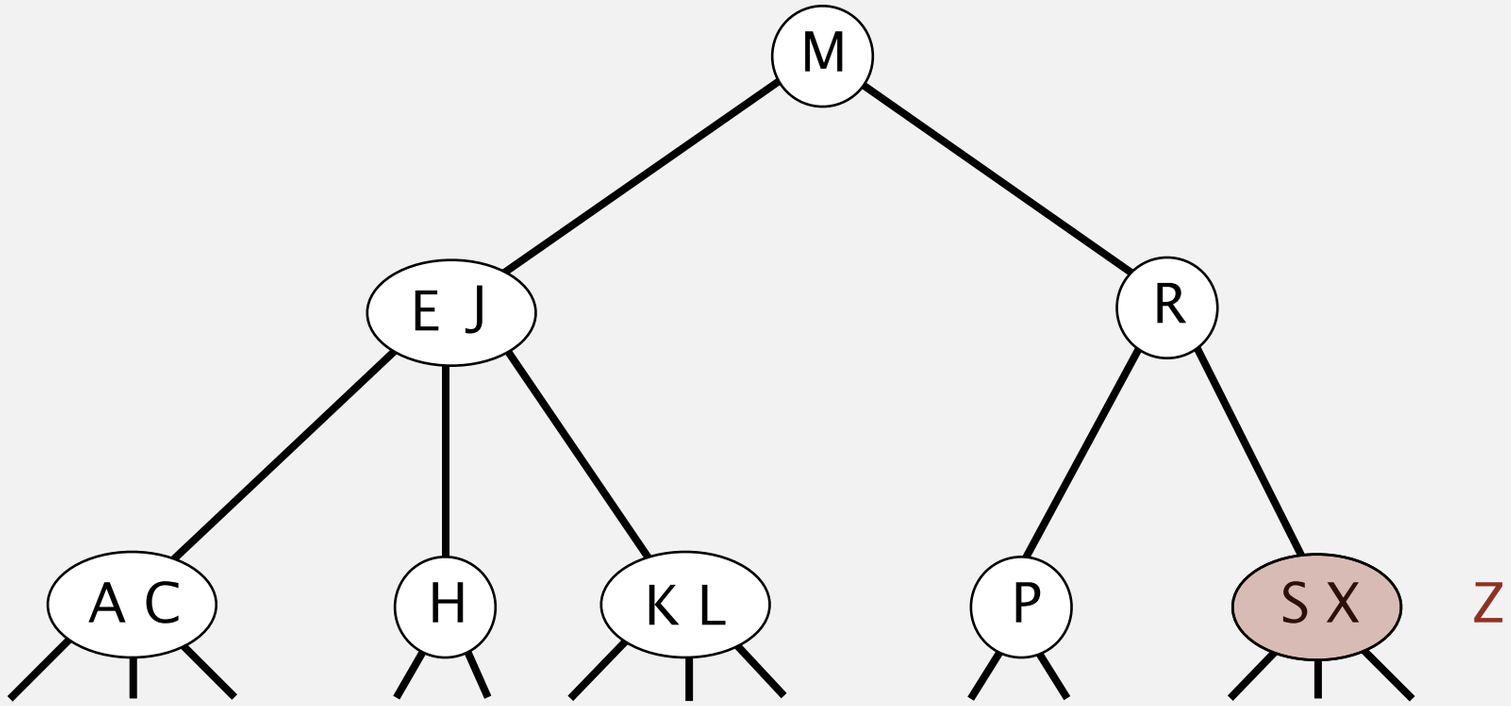


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z

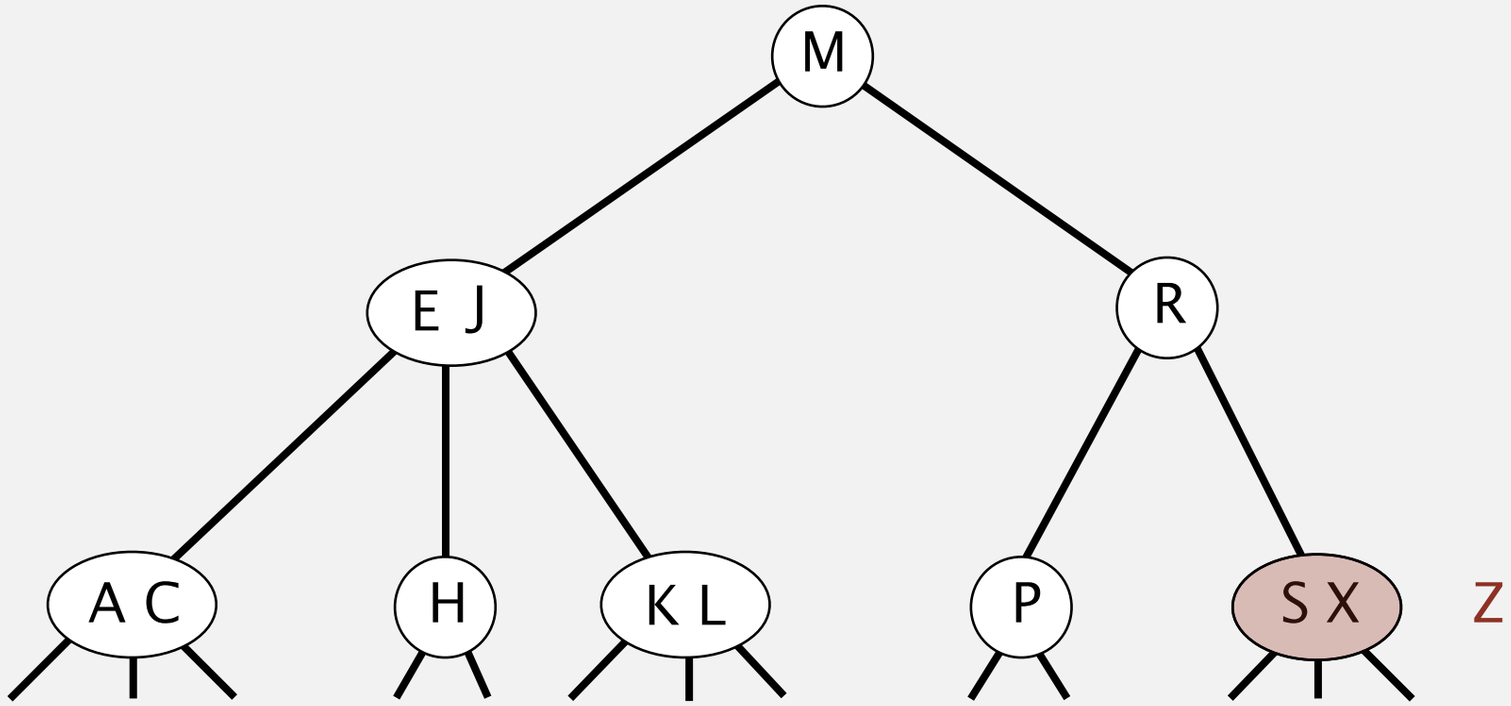


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z

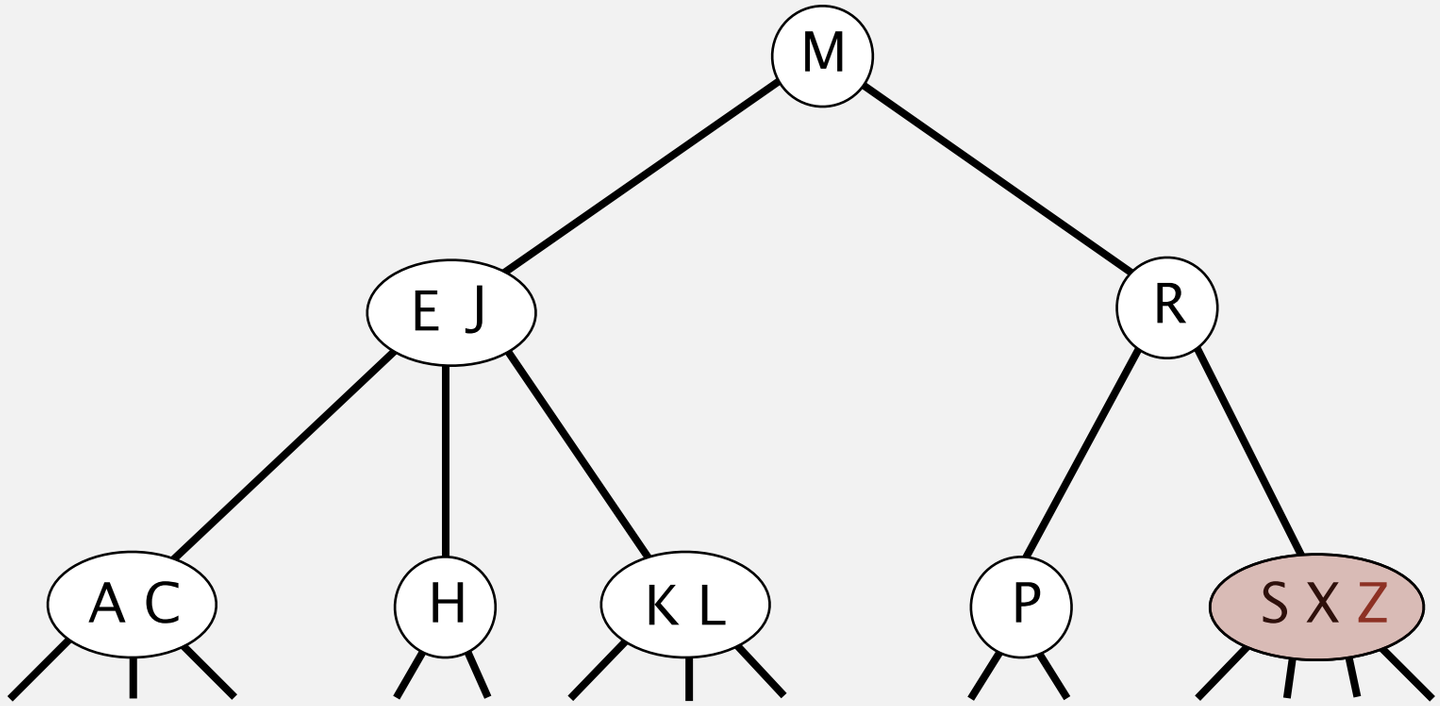


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z

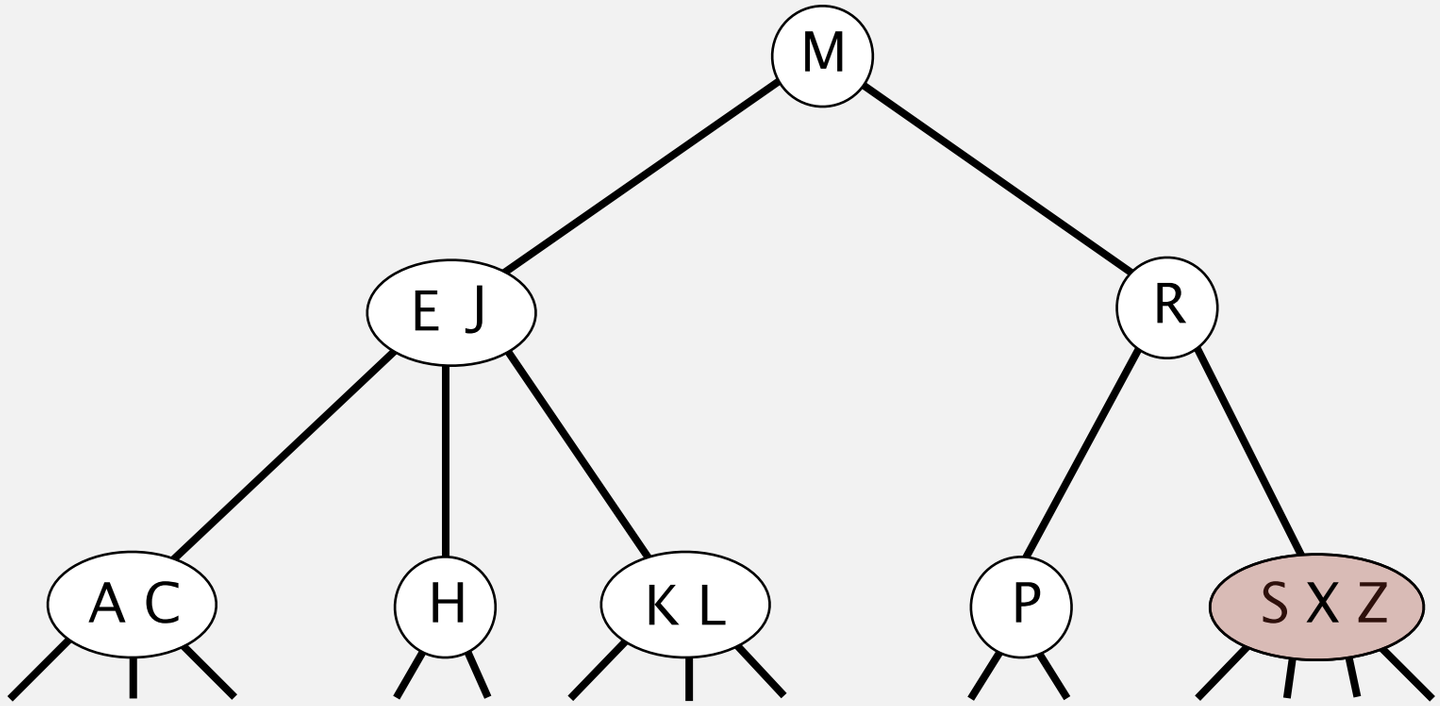


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z



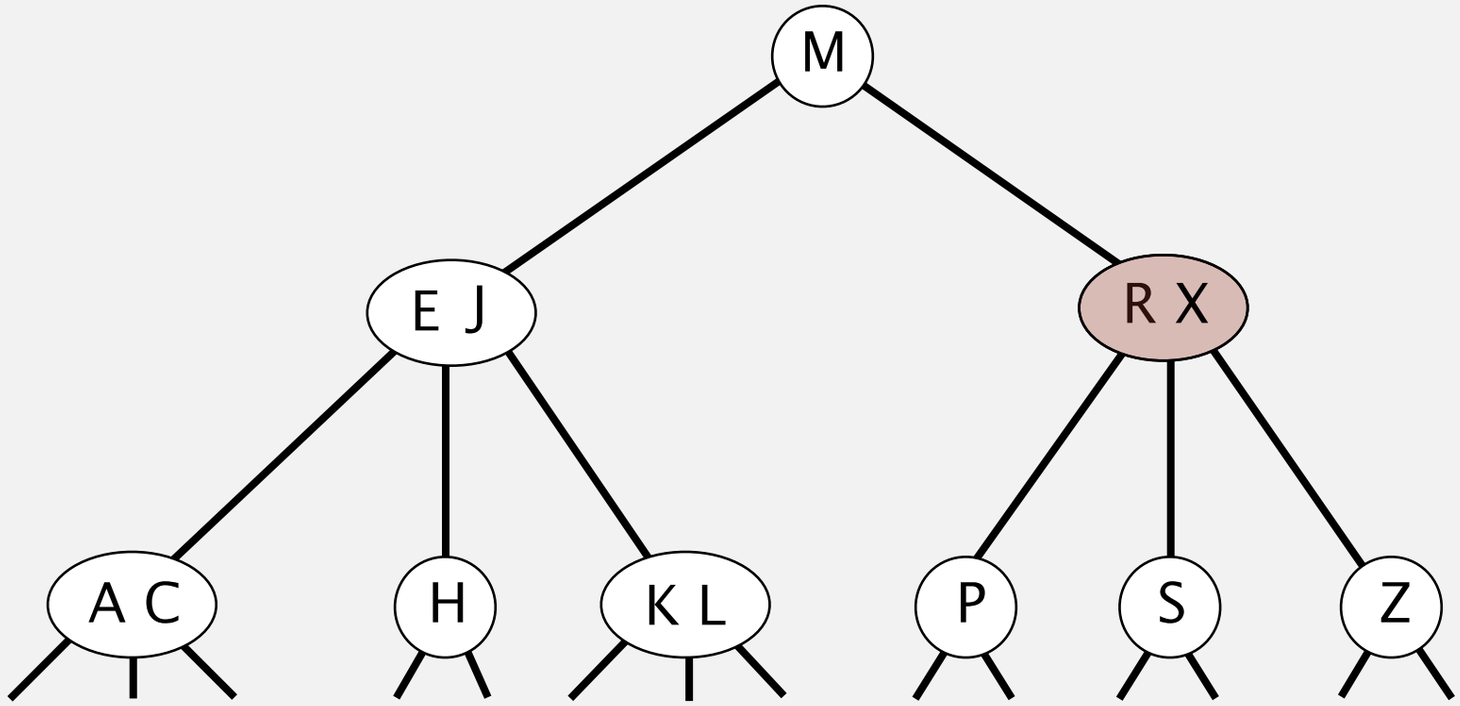
split 4-node into two 2-nodes
(pass middle key to parent)

2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z

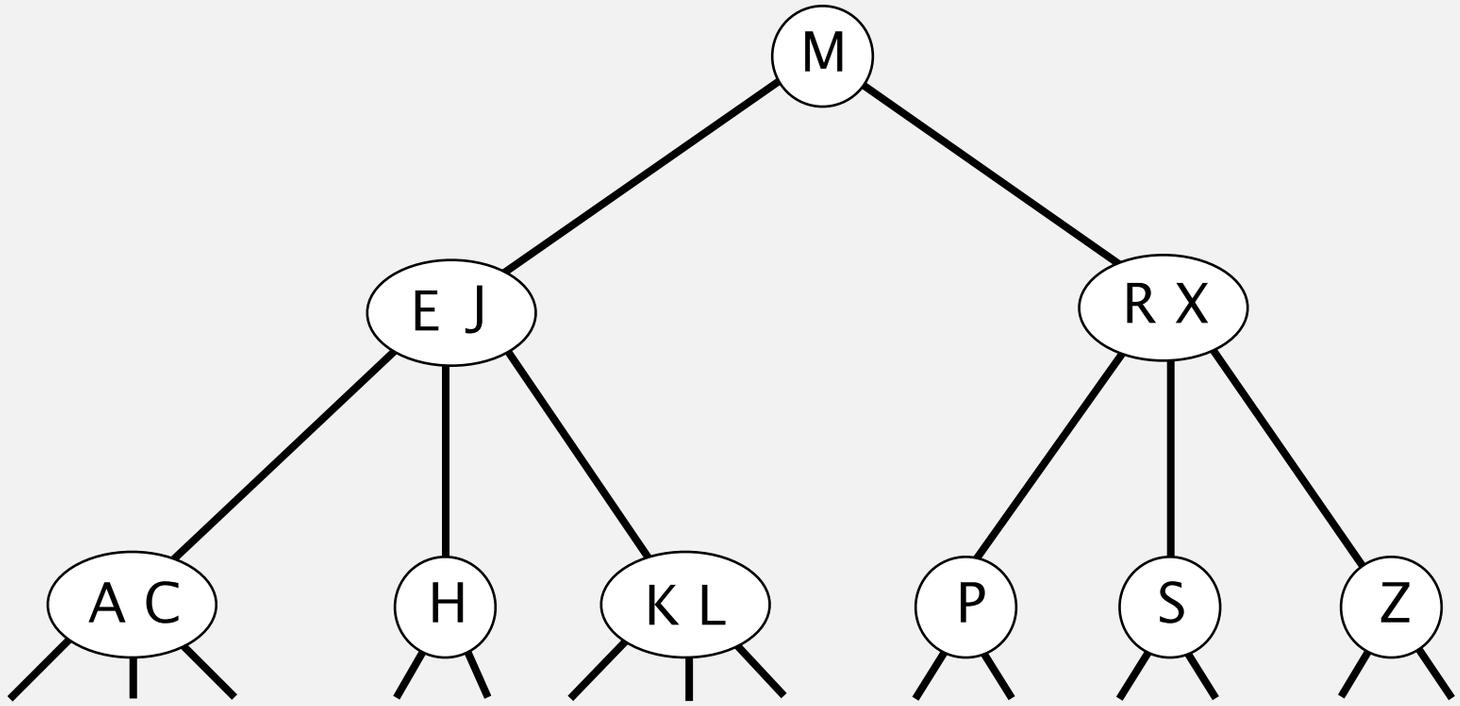


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.

insert Z

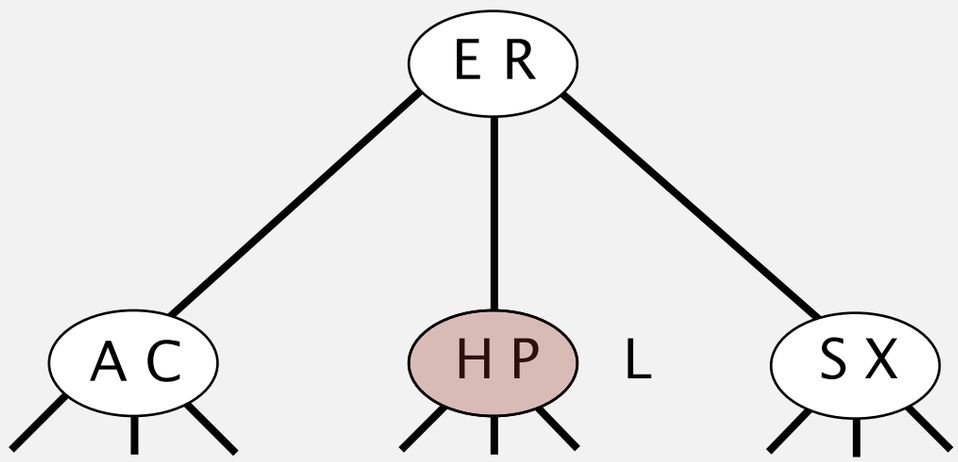


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert L



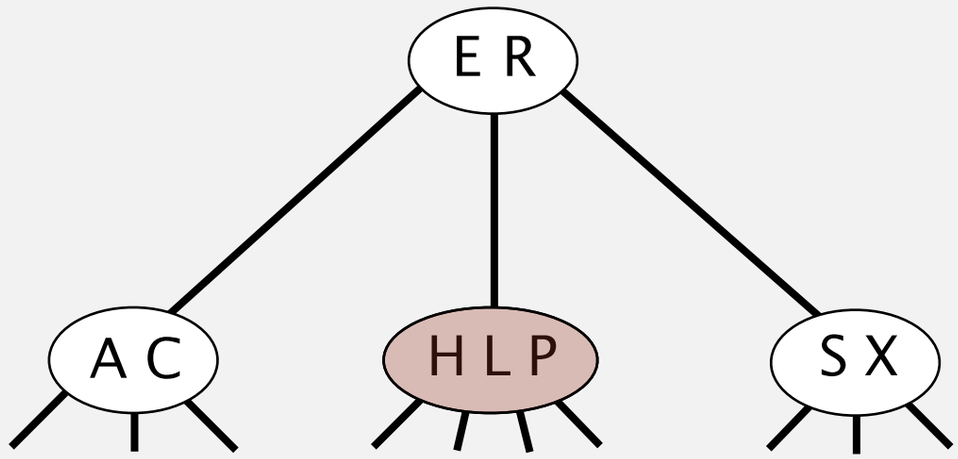
convert 3-node into 4-node

2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert L

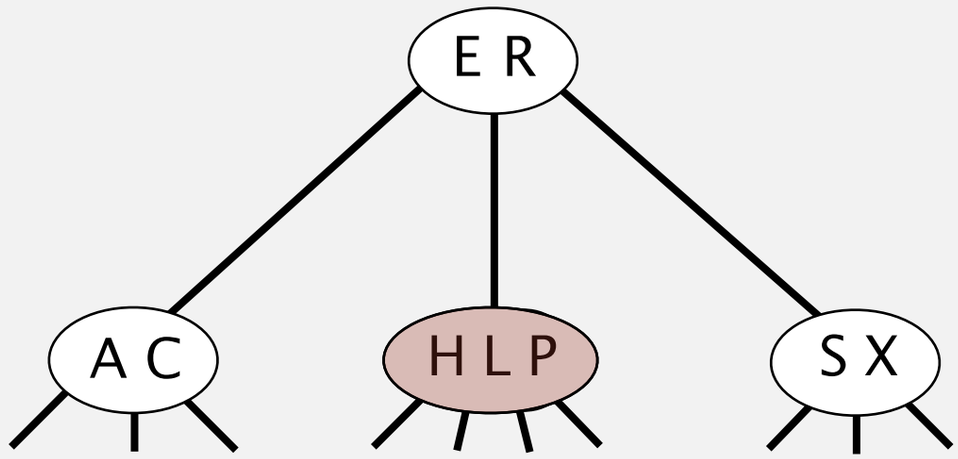


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert L



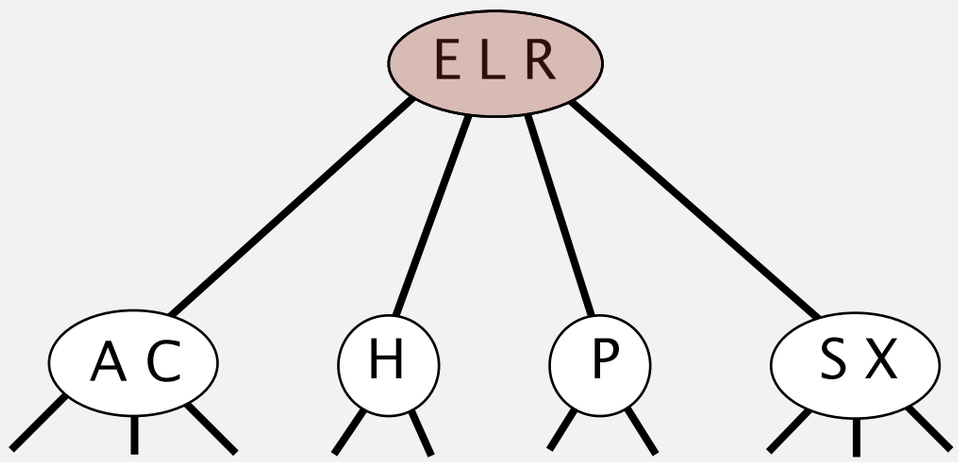
split 4-node
(move L to parent)

2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert L

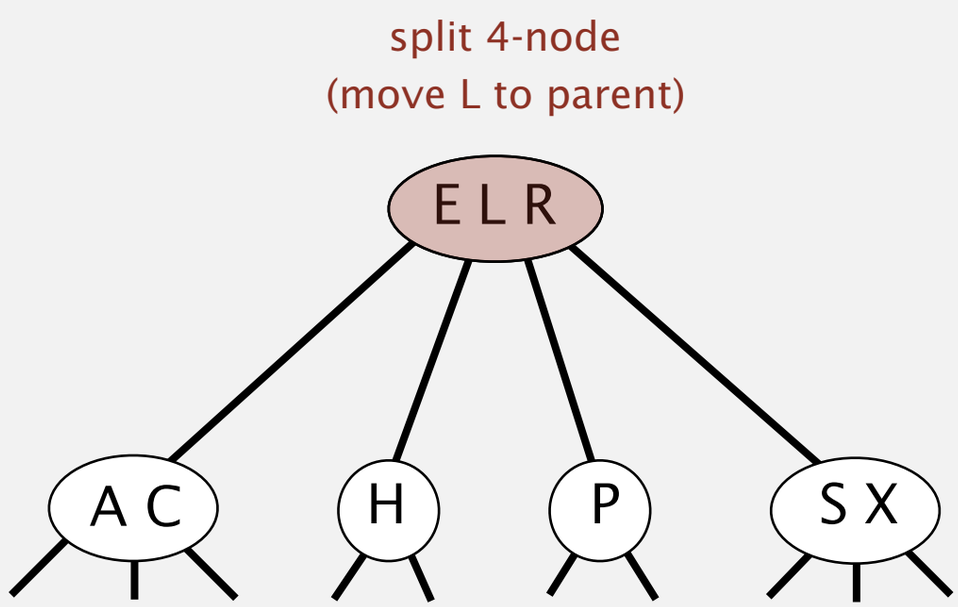


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert L



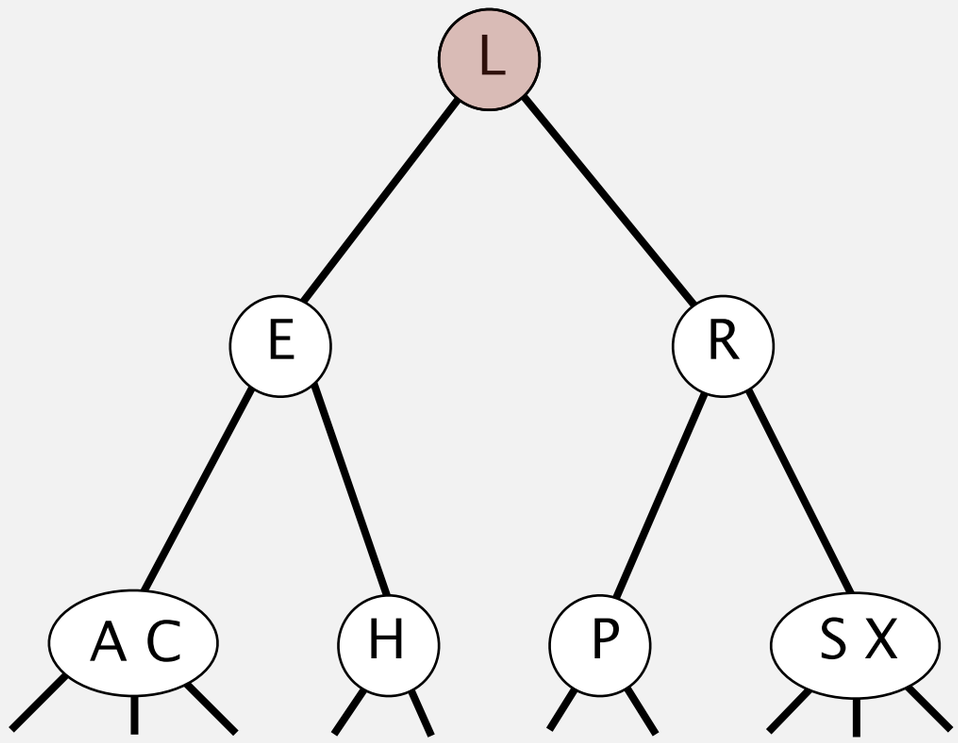
2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

height of tree increases by 1

insert L

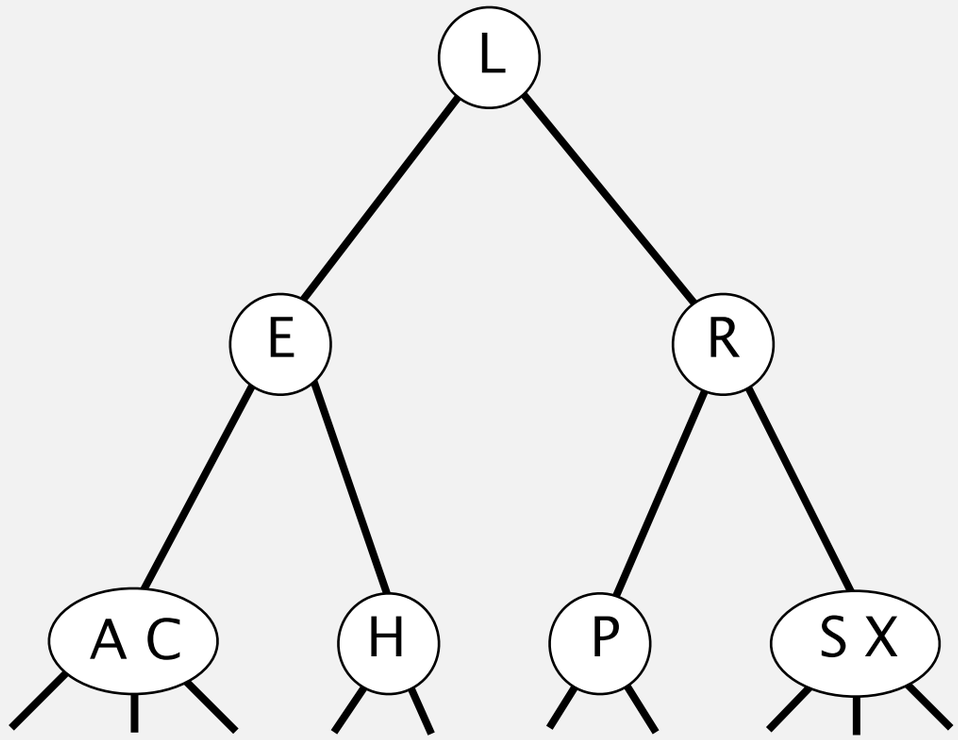


2-3 tree demo: insertion

Insert into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

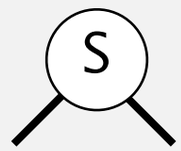
insert L



CONSTRUCTING A 2-3 TREE

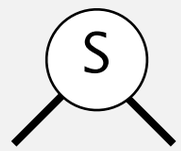
2-3 tree demo: construction

insert S



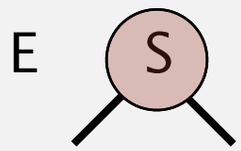
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

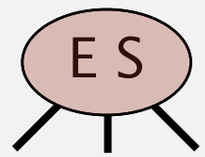
insert E



convert 2-node into 3-node

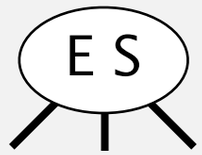
2-3 tree demo: construction

insert E



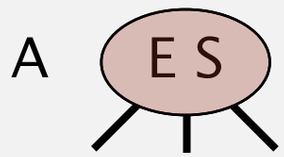
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

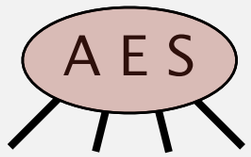
insert A



convert 3-node into 4-node

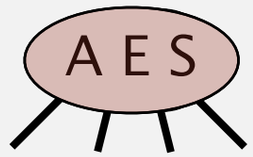
2-3 tree demo: construction

insert A



2-3 tree demo: construction

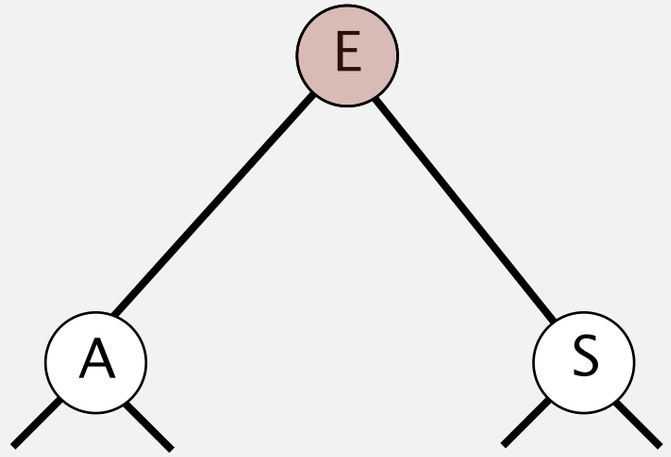
insert A



split 4-node
(move E to parent)

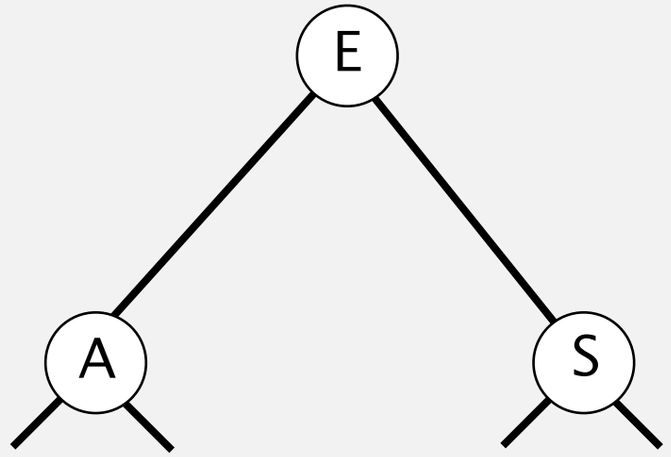
2-3 tree demo: construction

insert A



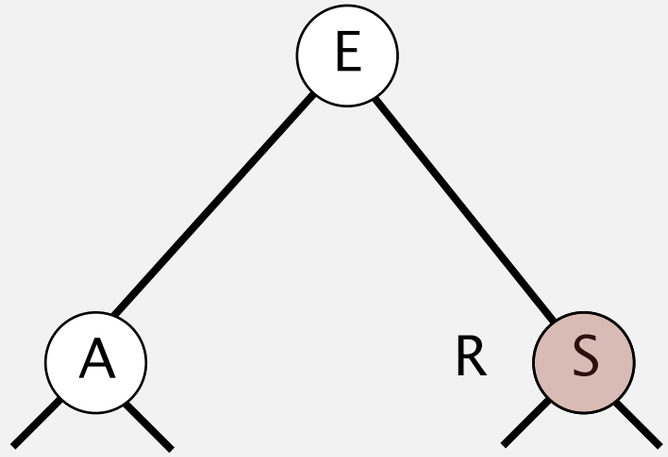
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

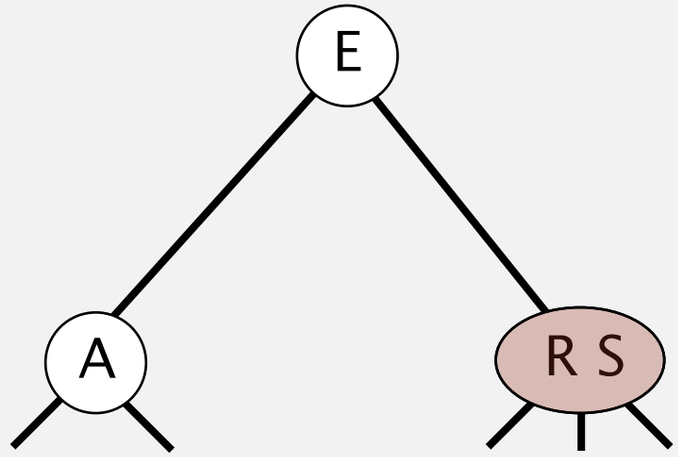
insert R



convert 2-node into 3-node

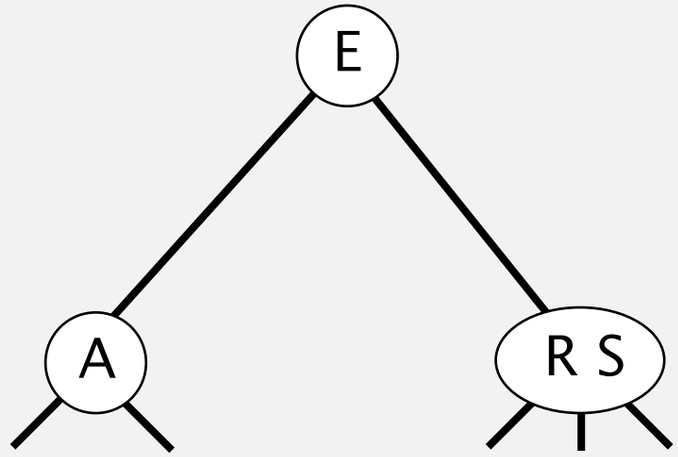
2-3 tree demo: construction

insert R



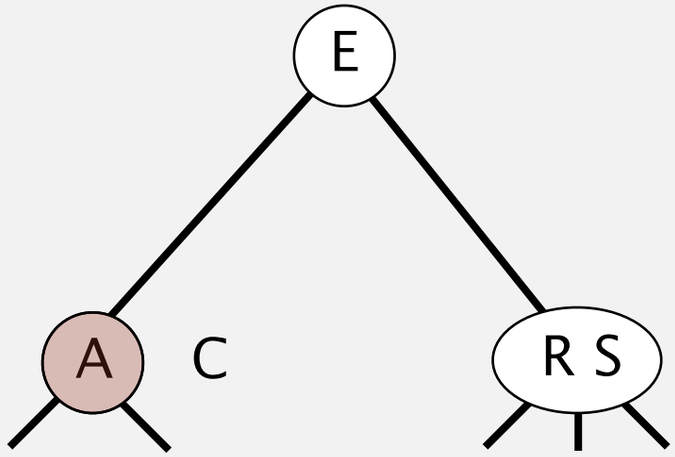
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

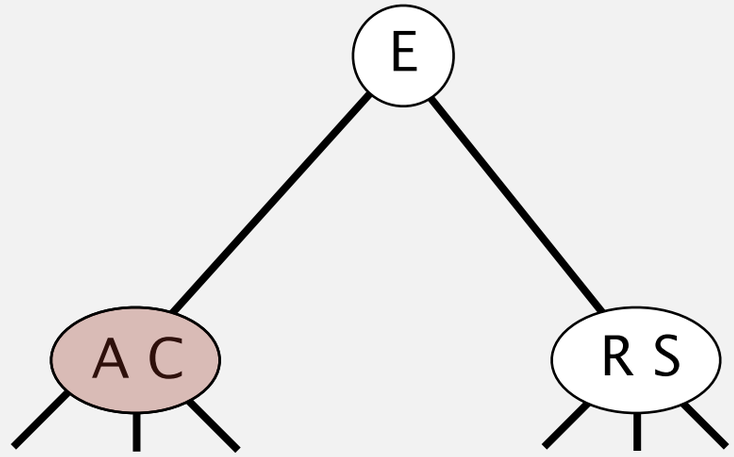
insert C



convert 2-node into 3-node

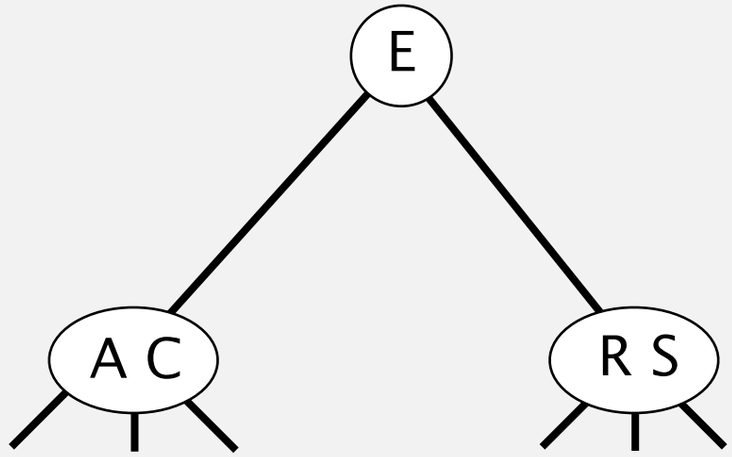
2-3 tree demo: construction

insert C



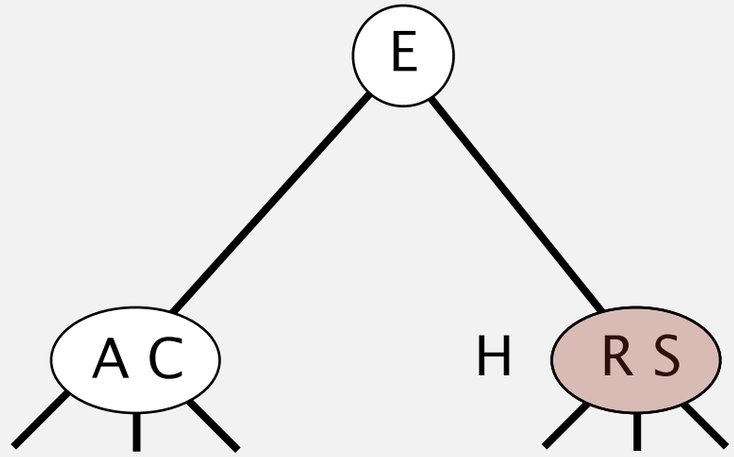
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

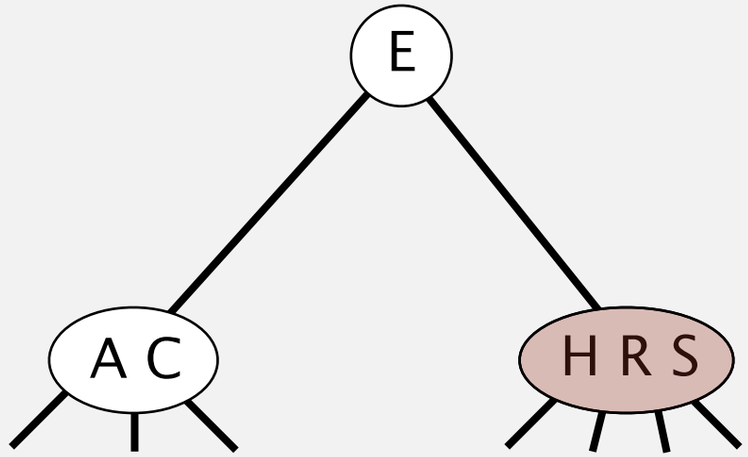
insert H



convert 3-node into 4-node

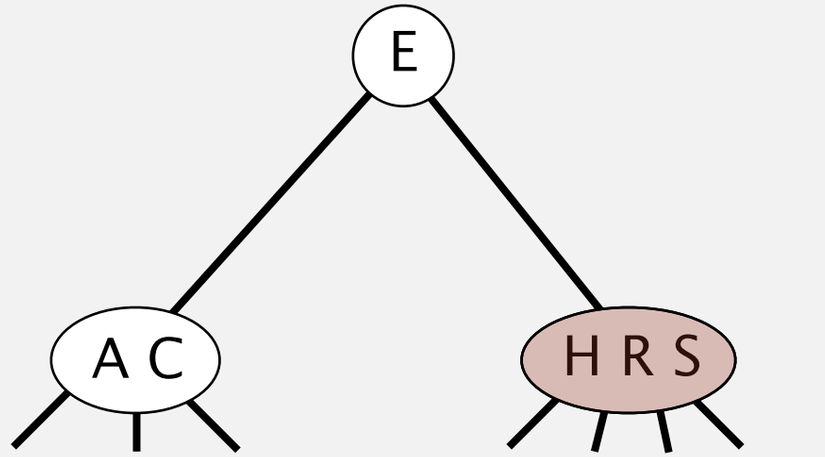
2-3 tree demo: construction

insert H



2-3 tree demo: construction

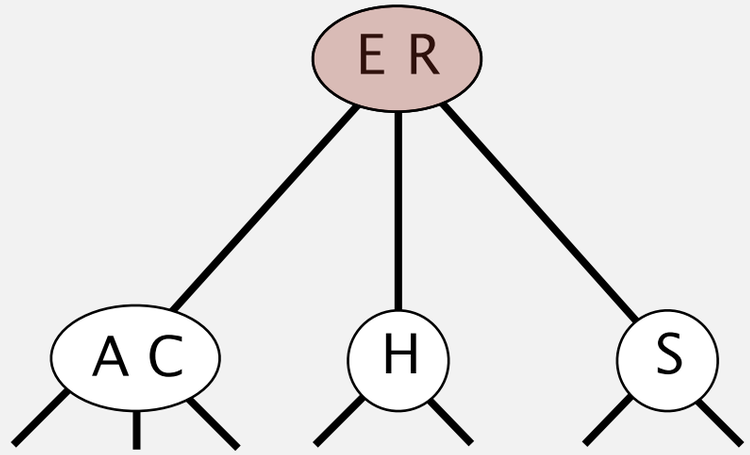
insert H



split 4-node
(move R to parent)

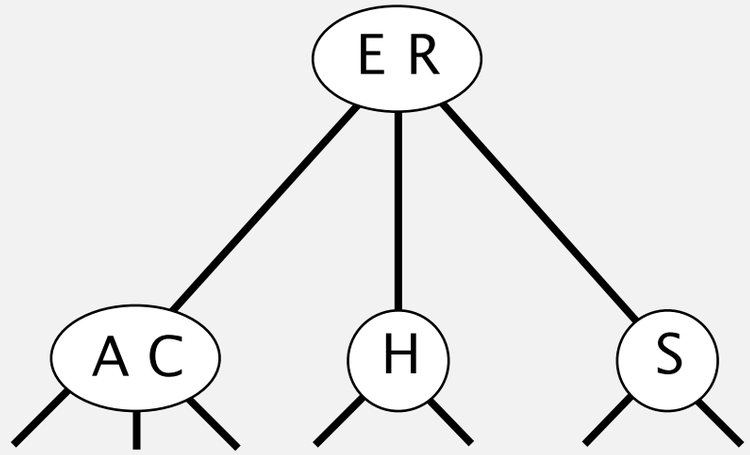
2-3 tree demo: construction

insert H



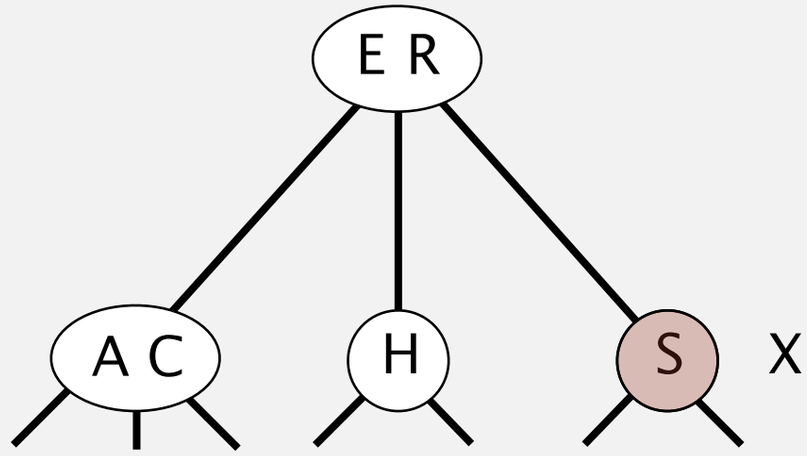
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

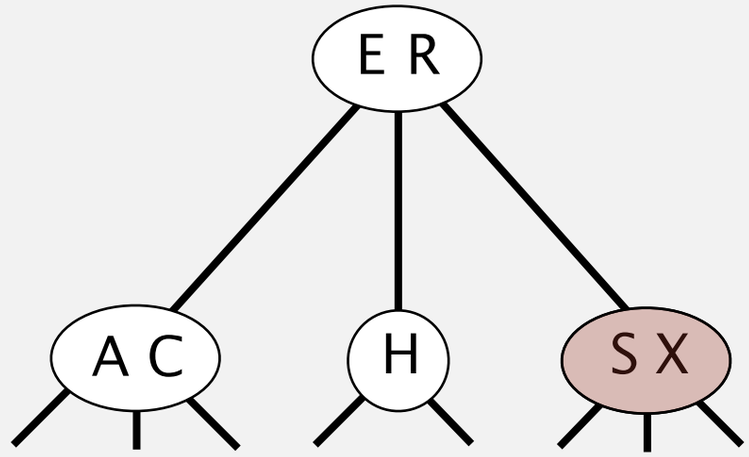
insert X



convert 2-node into 3-node

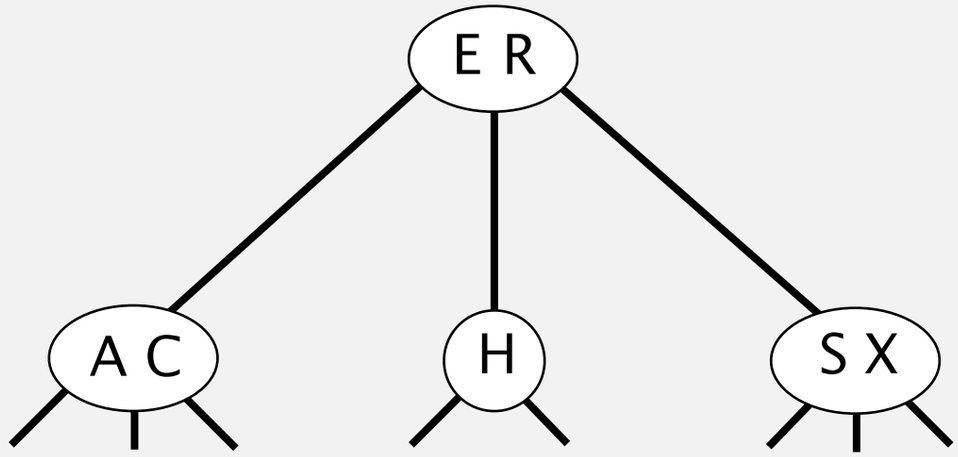
2-3 tree demo: construction

insert X



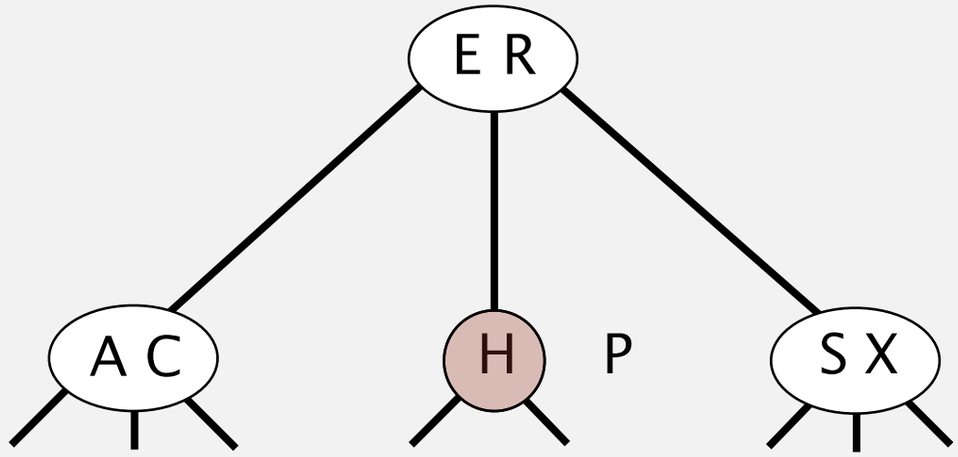
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

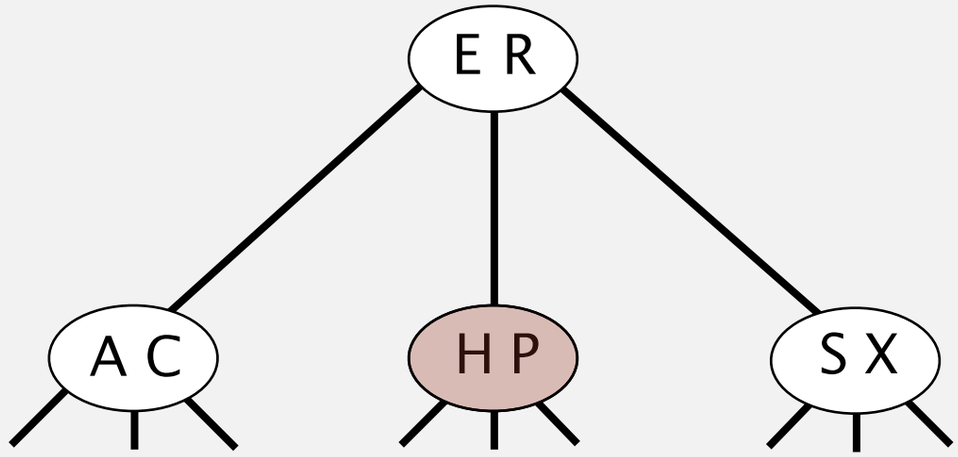
insert P



convert 2-node into 3-node

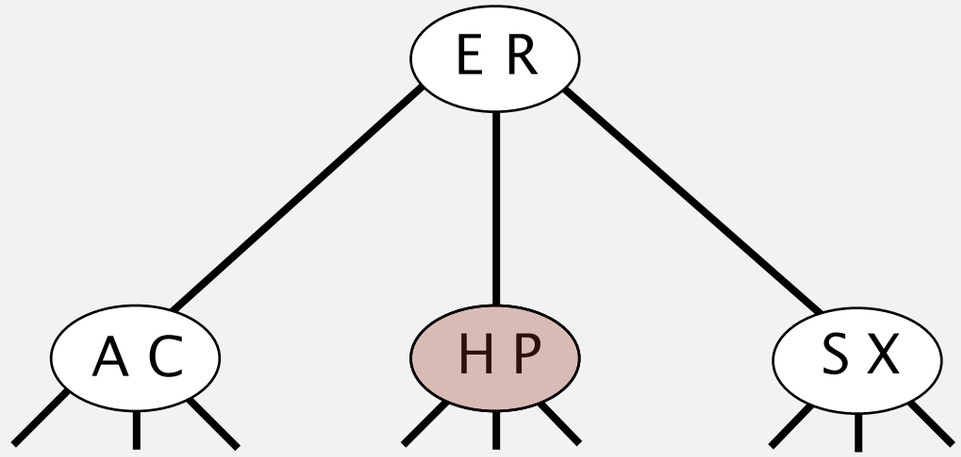
2-3 tree demo: construction

insert P



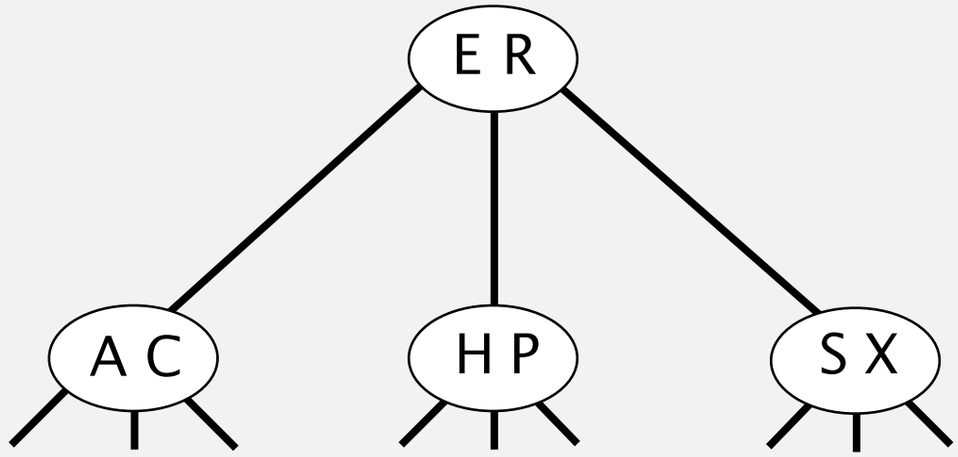
Quiz

- What is the result of inserting L into the following tree?



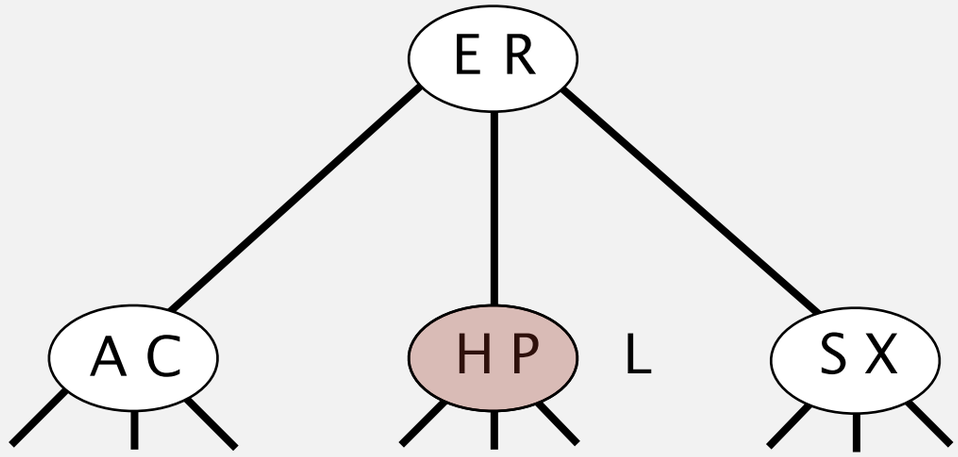
2-3 tree demo: construction

2-3 tree



2-3 tree demo: construction

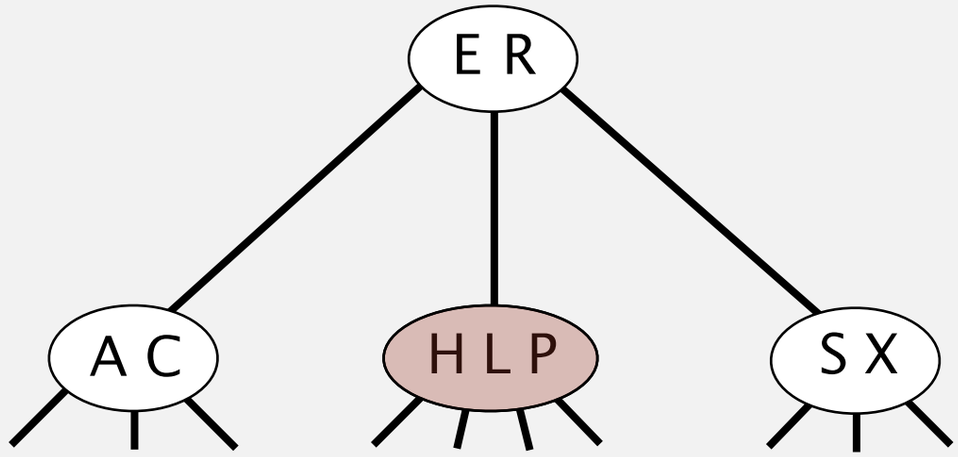
insert L



convert 3-node into 4-node

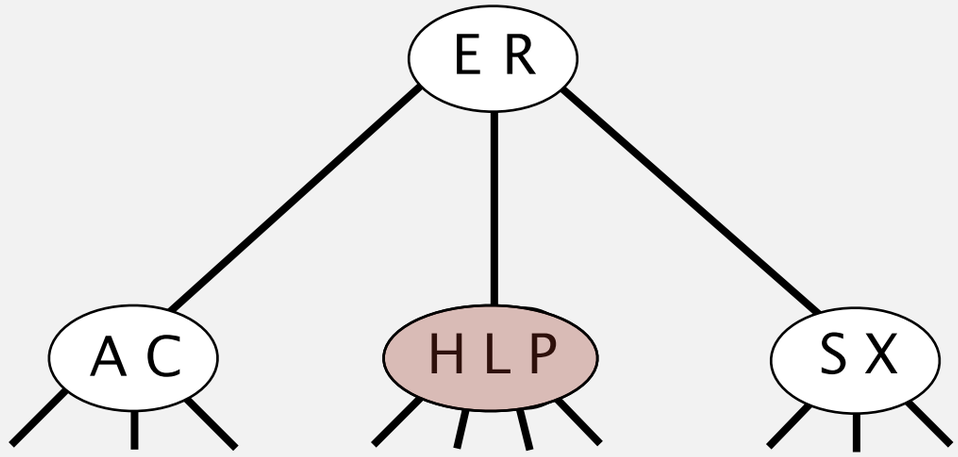
2-3 tree demo: construction

insert L



2-3 tree demo: construction

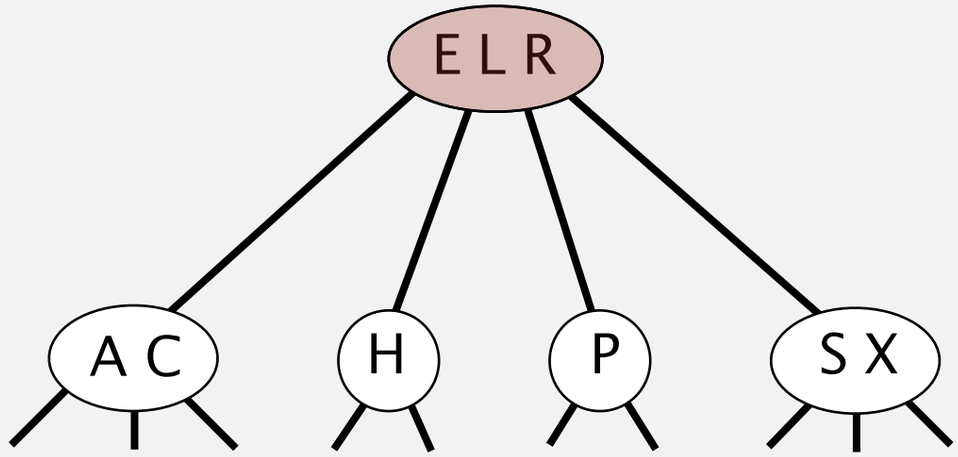
insert L



split 4-node
(move L to parent)

2-3 tree demo: construction

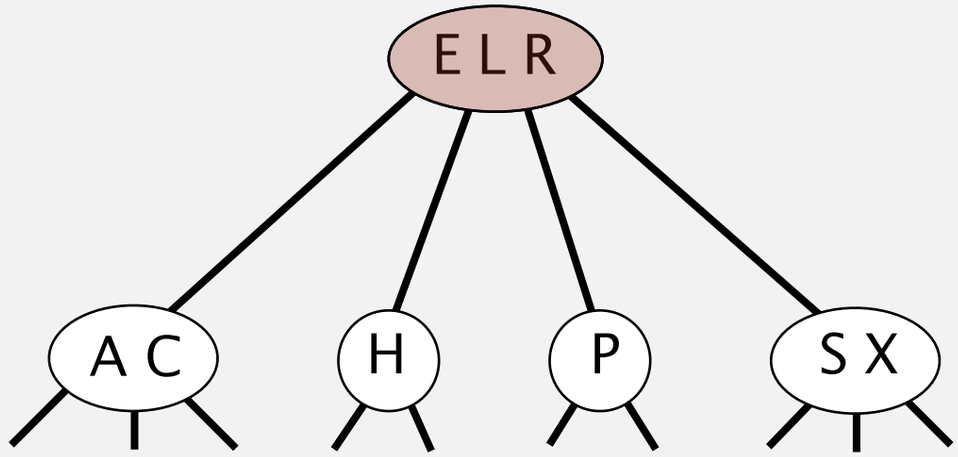
insert L



2-3 tree demo: construction

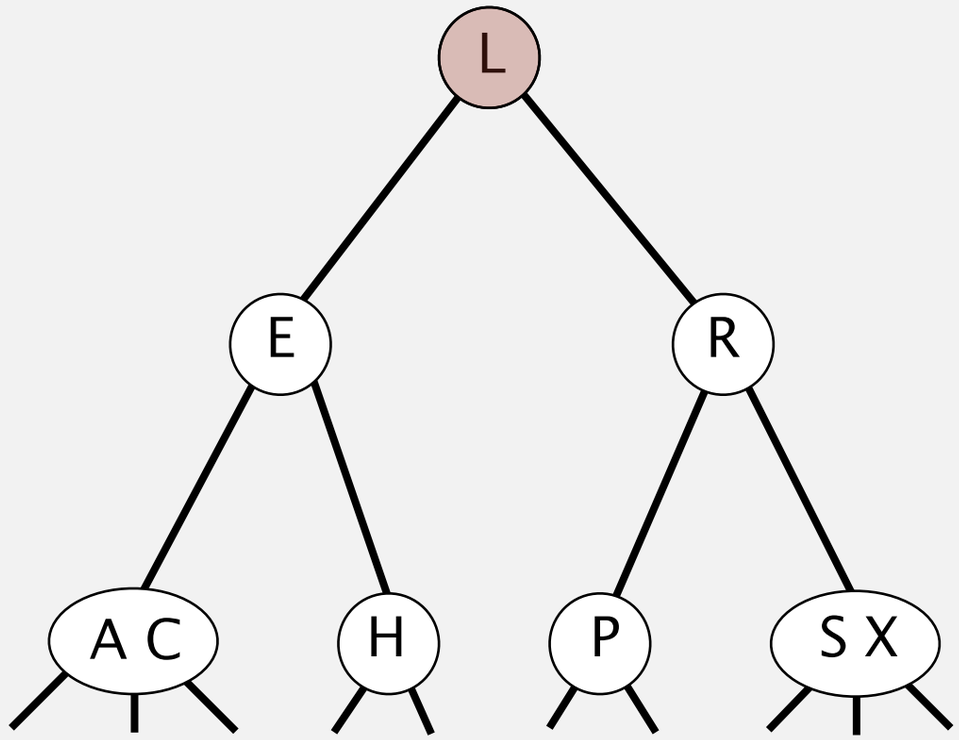
insert L

split 4-node
(move L to parent)



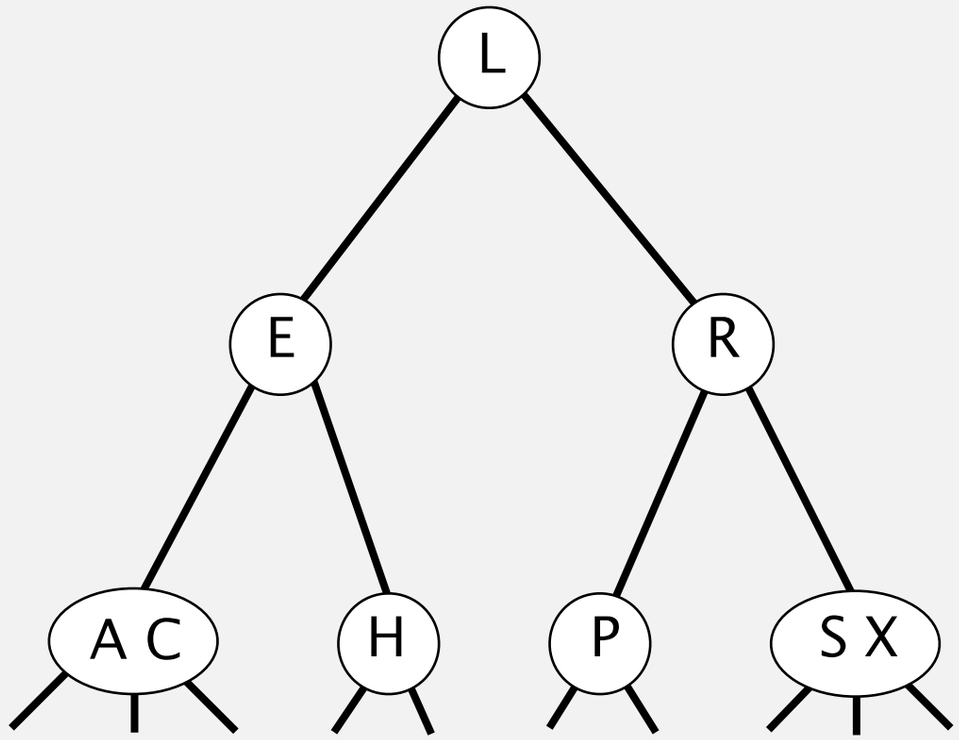
2-3 tree demo: construction

insert L



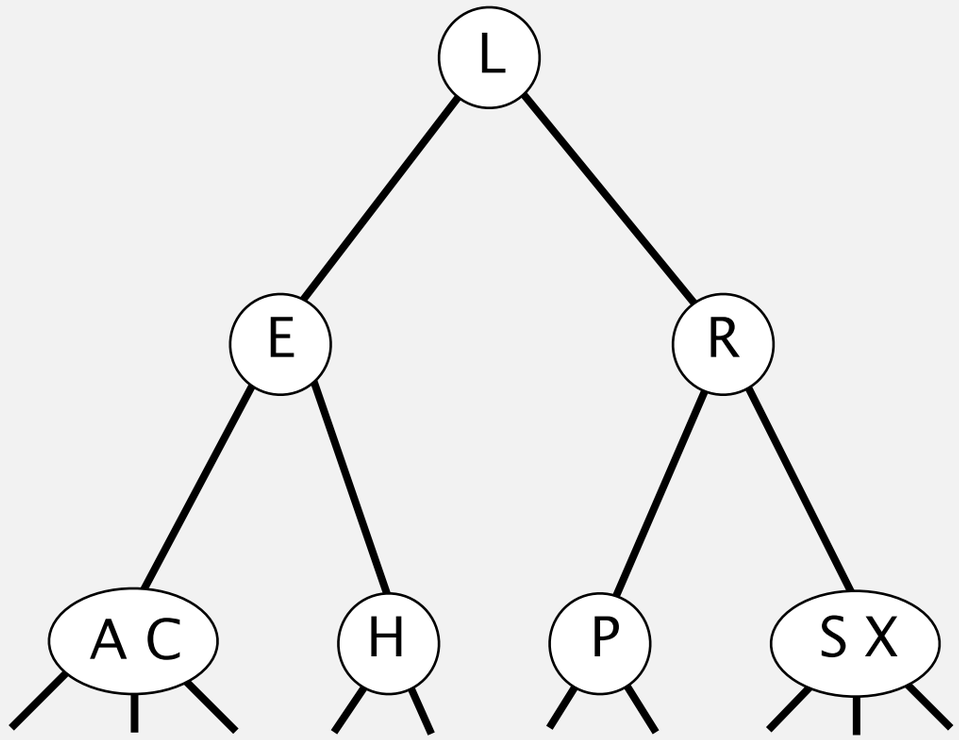
2-3 tree demo: construction

2-3 tree



Quiz

What is the result of inserting G into the following tree

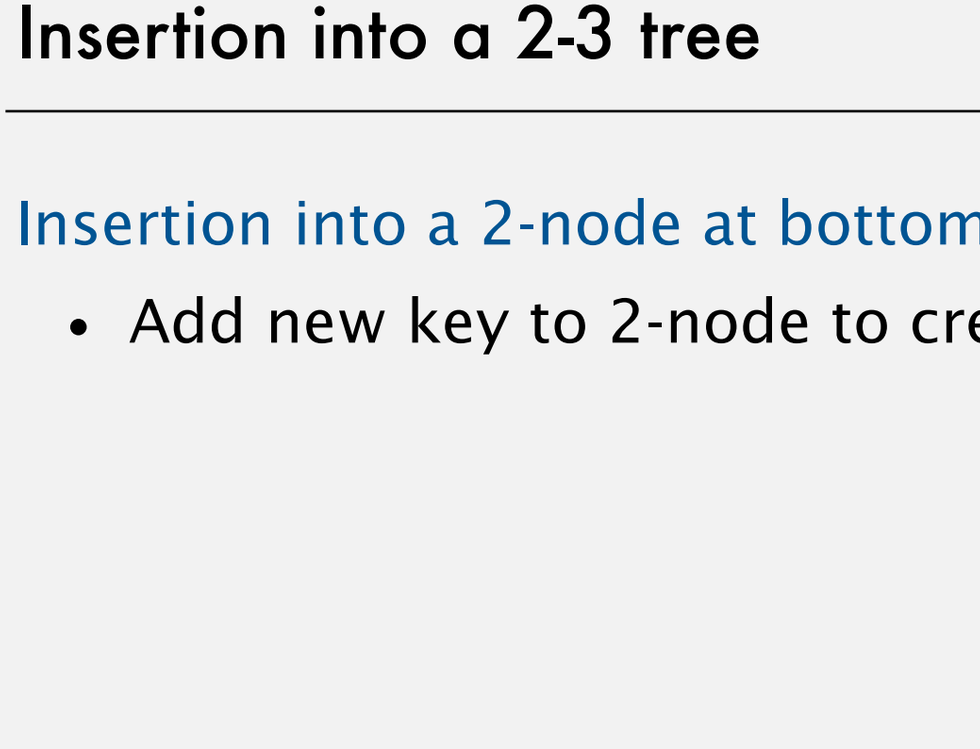


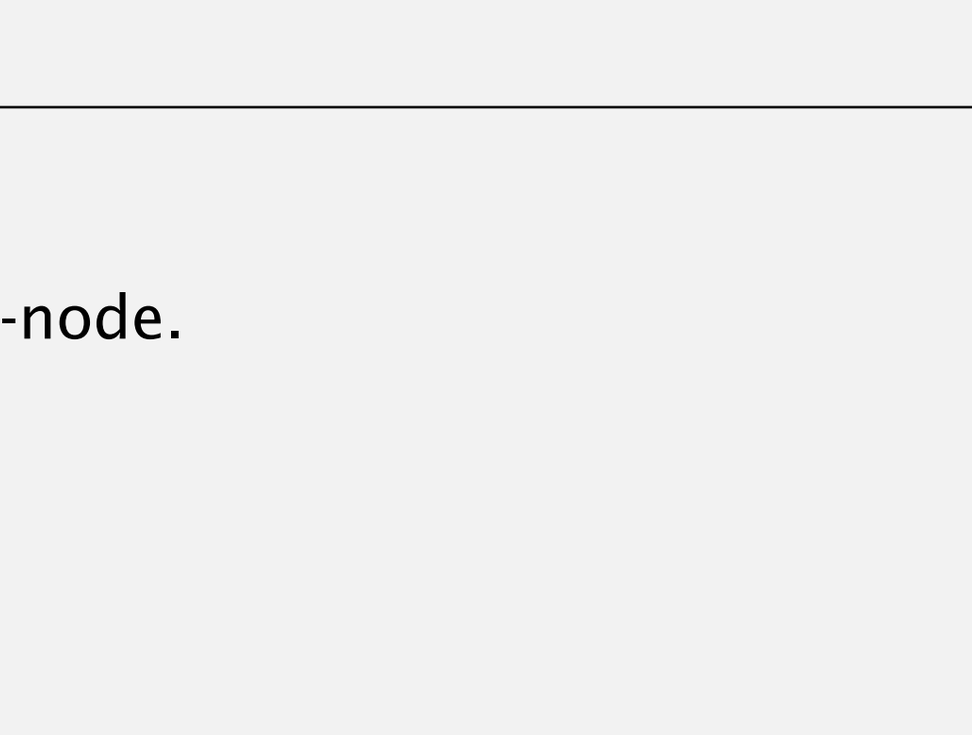
Insertion into a 2-3 tree

Insertion into a 2-node at bottom.

- Add new key to 2-node to create a 3-node.

insert G

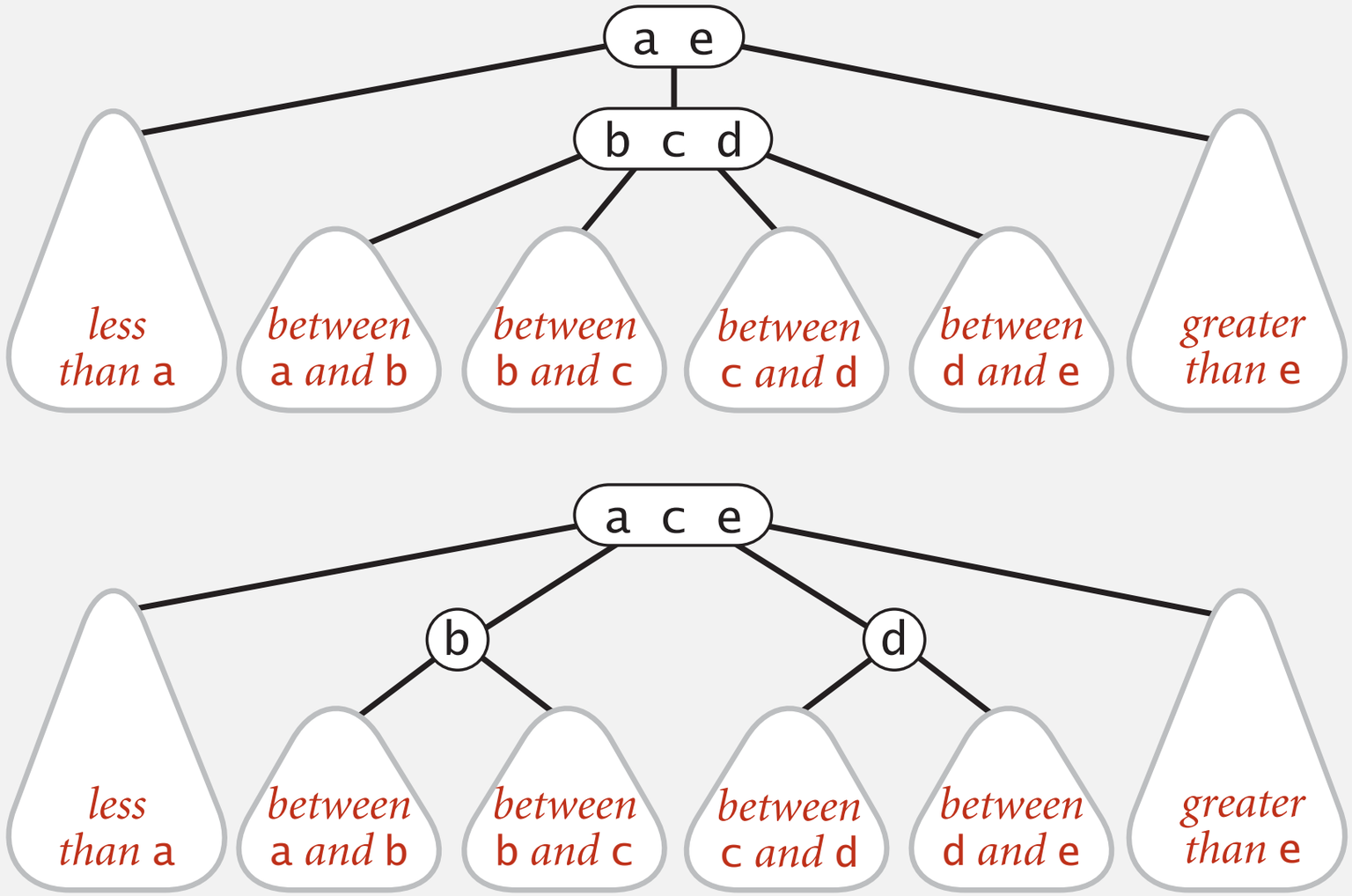




Local transformations in a 2-3 tree

Splitting a 4-node is a local transformation: constant number of operations.

Splitting does not change the balance of the tree



Global properties in a 2-3 tree

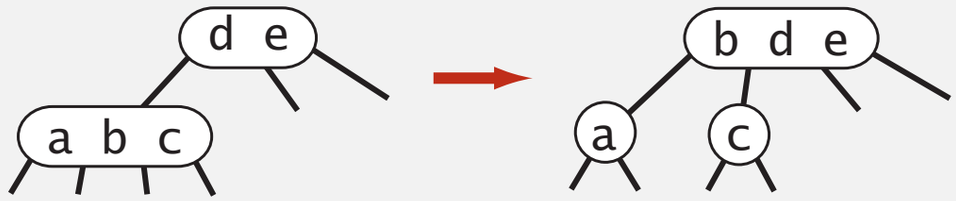
Invariants. Maintains symmetric order and perfect balance.

Pf. Each transformation maintains symmetric order and perfect balance.

root

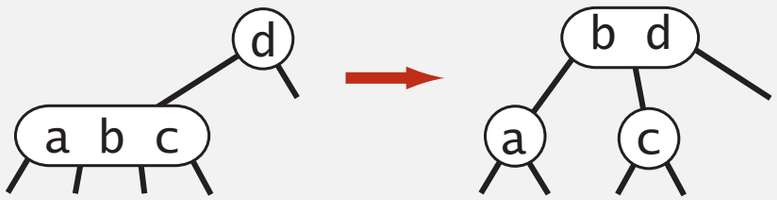
parent is a 3-node

left

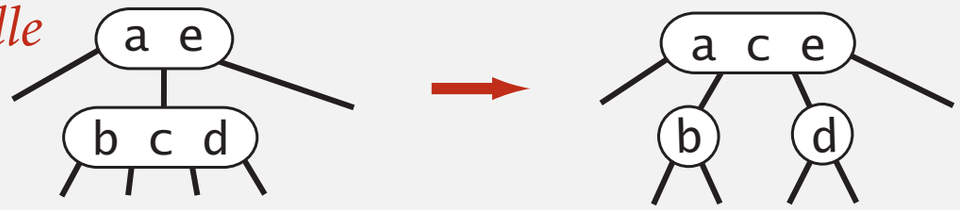


parent is a 2-node

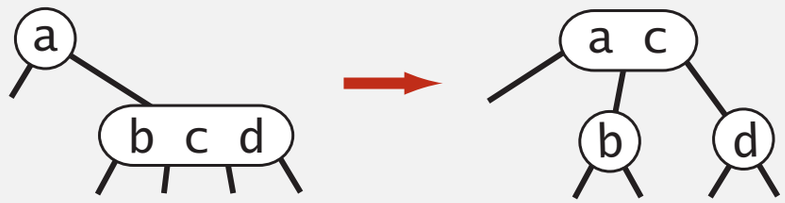
left



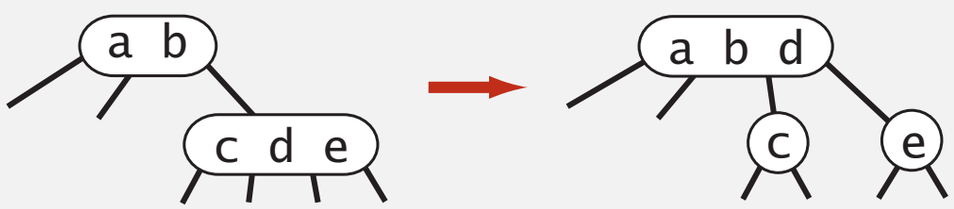
middle



right

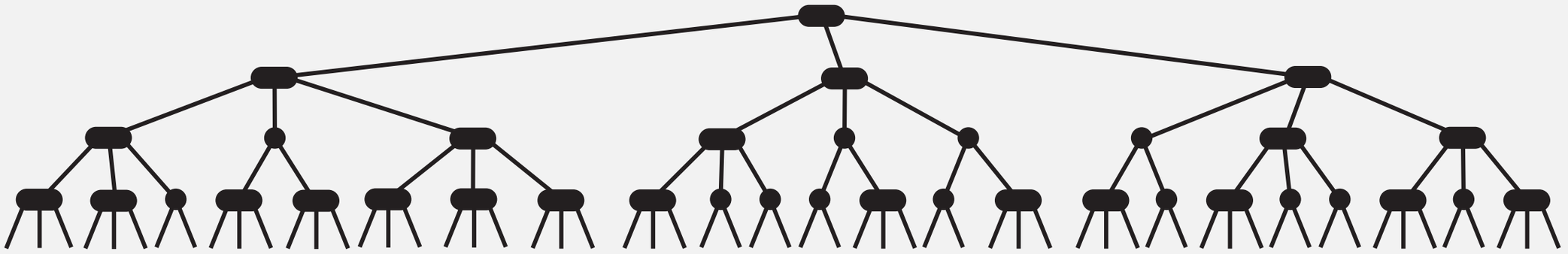


right



2-3 tree: performance

Perfect balance. Every path from root to null link has same length.

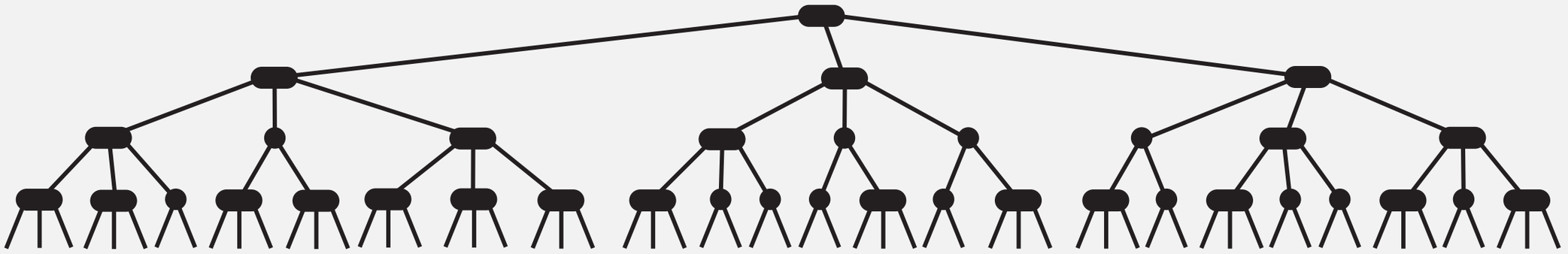


Tree height.

- Worst case:
- Best case:

2-3 tree: performance

Perfect balance. Every path from root to null link has same length.



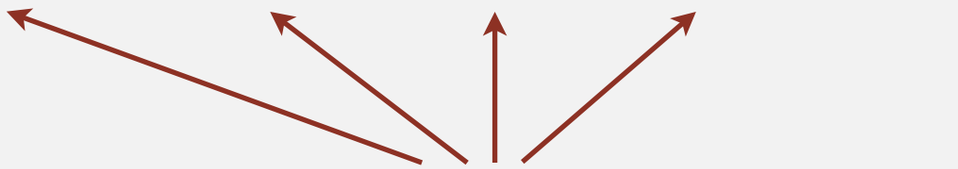
Tree height.

- Worst case: $\log_2 N$. [all 2-nodes]
- Best case: $\log_3 N \approx .631 \log_2 N$. [all 3-nodes]
- Between 12 and 20 for a million nodes.
- Between 18 and 30 for a billion nodes.

Bottom line. Guaranteed logarithmic performance for search and insert.

ST implementations: summary

implementation	guarantee			average case		
	search	insert	delete	search hit	insert	delete
BST	N	N	N	C lg N	c lg N	c lg N
2-3 tree	c lg N	c lg N	c lg N	c lg N	c lg N	c lg N



2-3 TREES

Implementation

2-3 tree: implementation?

Direct implementation is complicated, because:

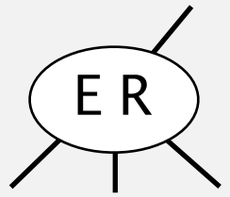
- Maintaining multiple node types is cumbersome.
- Need multiple compares to move down tree.
- Need to move back up the tree to split 4-nodes.
- Large number of cases for splitting.

> *" Beautiful algorithms are not always the most useful. "*
>
> — *Donald Knuth*

Bottom line. Could do it, but there's a better way.

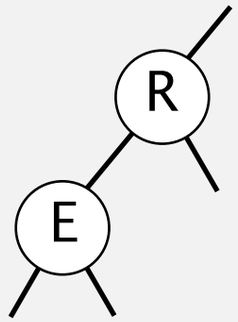
How to implement 2-3 trees with binary trees?

Challenge. How to represent a 3 node?



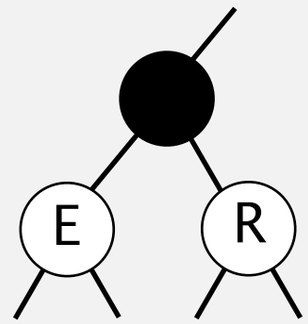
Approach 1: regular BST.

- No way to tell a 3-node from a 2-node.
- Cannot map from BST back to 2-3 tree.



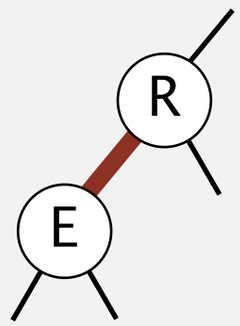
Approach 2: regular BST with "glue" nodes.

- Wastes space, wasted link.
- Code probably messy.



Approach 3: regular BST with red "glue" links.

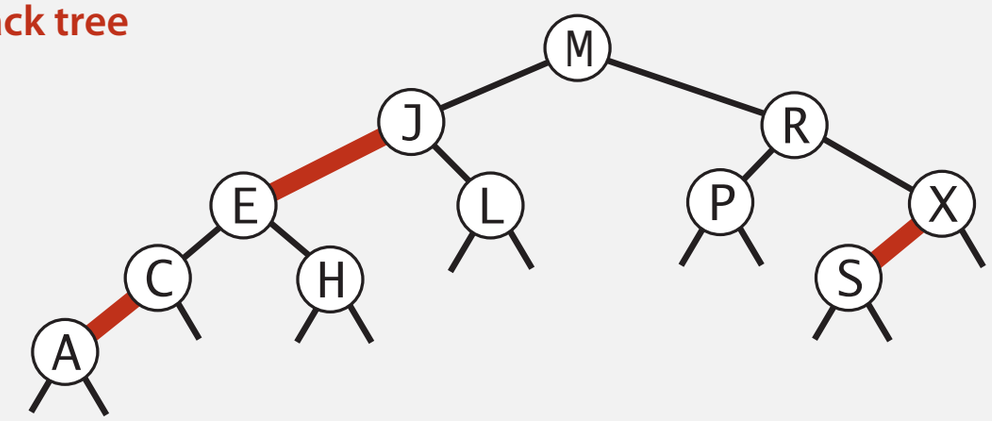
- Widely used in practice.
- Arbitrary restriction: red links lean left.



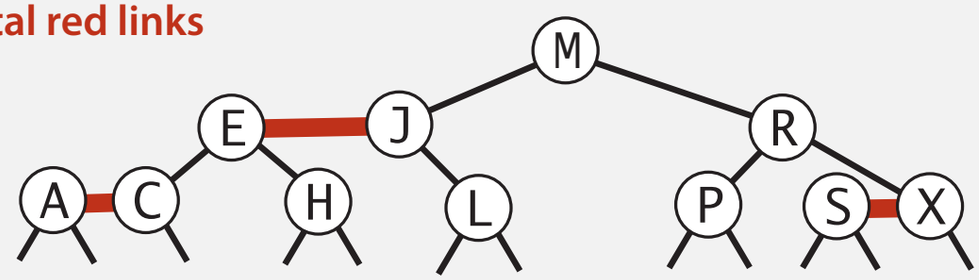
Left-leaning red-black BSTs: 1-1 correspondence with 2-3 trees

Key property. 1–1 correspondence between 2–3 and LLRB.

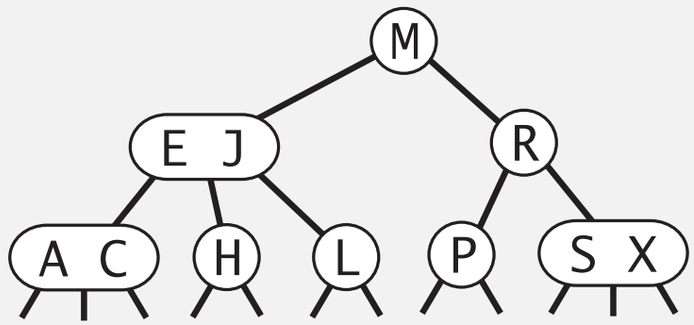
red–black tree



horizontal red links



2-3 tree

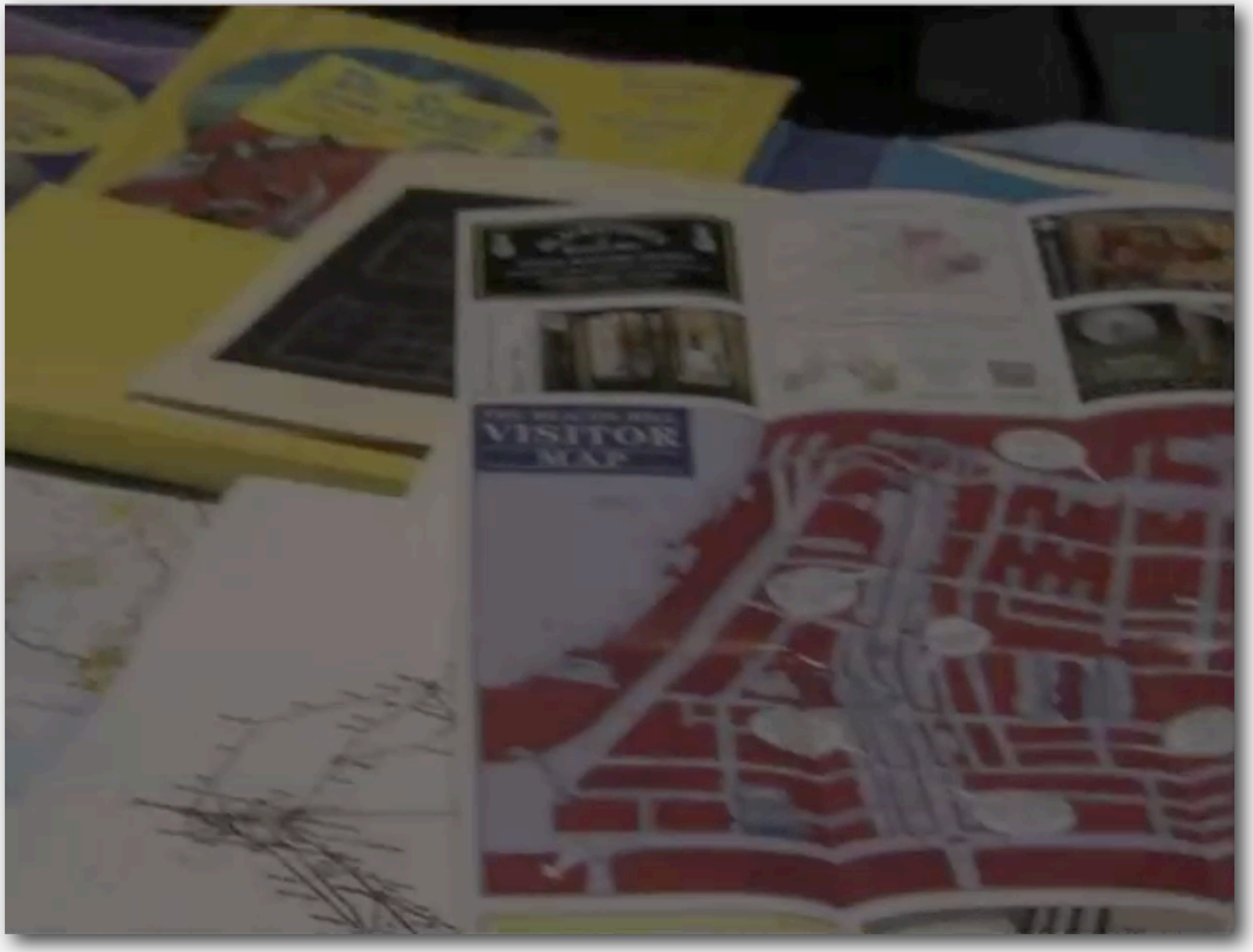


Balanced trees in the wild

Red-black trees are widely used:

- Java: `java.util.TreeMap`, `java.util.TreeSet`.
- Linux kernel: completely fair scheduler, `linux/rbtree.h`.

Red-black BSTs in the wild



Red-black BSTs in the wild

ACT FOUR

FADE IN:

48 INT. FBI HQ - NIGHT 48

Antonio is at THE COMPUTER as Jess explains herself to Nicole and Pollock. The CONFERENCE TABLE is covered with OPEN REFERENCE BOOKS, TOURIST GUIDES, MAPS and REAMS OF PRINTOUTS.

JESS
It was the red door again.

POLLOCK
I thought the red door was the storage container.

JESS
But it wasn't red anymore. It was black.

ANTONIO
So red turning to black means... what?

POLLOCK
Budget deficits? Red ink, black ink?

NICOLE
Yes. I'm sure that's what it is. But maybe we should come up with a couple other options, just in case.

Antonio refers to his COMPUTER SCREEN, which is filled with mathematical equations.

ANTONIO
It could be an algorithm from a binary search tree. A red-black tree tracks every simple path from a node to a descendant leaf with the same number of black nodes.

JESS
Does that help you with girls?

War story: why red-black?

Xerox PARC innovations. [1970s]

- Alto.
- GUI.
- Ethernet.
- Smalltalk.
- InterPress.
- Laser printing.
- Bitmapped display.
- WYSIWYG text editor.
- ...

XEROX PARC

Xerox Alto

A DICHROMATIC FRAMEWORK FOR BALANCED TREES

Leo J. Guibas
Xerox Palo Alto Research Center,
Palo Alto, California, and
Carnegie-Mellon University

and

Robert Sedgewick*
Program in Computer Science
Brown University
Providence, R. I.

ABSTRACT

In this paper we present a uniform framework for the implementation and study of balanced tree algorithms. We show how to imbed in this the way down towards a leaf. As we will see, this has a number of significant advantages over the older methods. We shall examine a number of variations on a common theme and exhibit full implementations which are notable for their brevity. One implementation is examined carefully, and some properties about its

War story: red-black BSTs

Telephone company contracted with database provider to build real-time database to store customer information.

Database implementation.

- Red-black BST search and insert; (no rebalancing on deletion).
- Exceeding height limit of 80 triggered error-recovery process.

allows for up to 2^{40} keys well over a trillion keys

Extended telephone service outage.

- Main cause = height bounded exceeded!
- Telephone company sues database provider.
- Legal testimony:

> *" If implemented properly, the height of a red-black BST with N keys is at most 2 lg N. "* — *expert witness*

B- TREES

B-tree variants.

B-tree, Nodes contains keys and data.

B+ tree, - Data only stored in leafs (Deletion/Insertion simple just leaves)

B-trees (and variants) are widely used for file systems and databases.

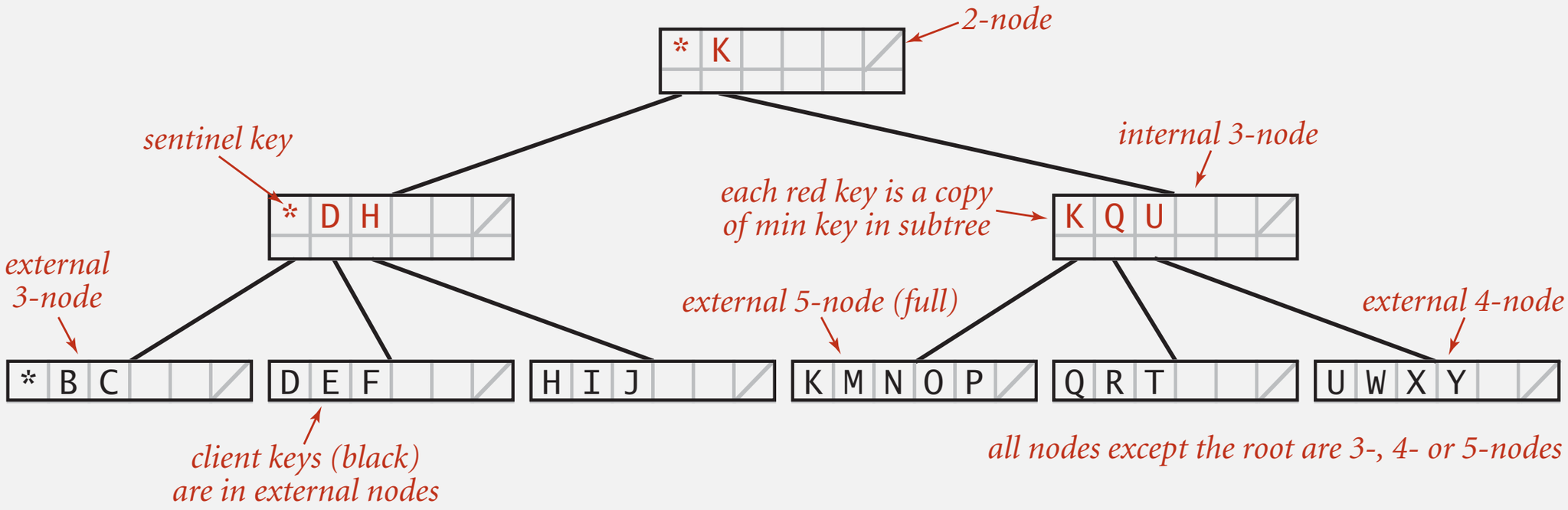
- Windows: NTFS.
- Mac: HFS, HFS+.
- Linux: ReiserFS, XFS, Ext3FS, JFS.
- Databases: ORACLE, DB2, INGRES, SQL, PostgreSQL.

B-trees (Bayer-McCreight, 1972)

B-tree. Generalize 2-3 trees by allowing up to $M - 1$ key per node.

- At least 2 key-link pairs at root.
- At least $M / 2$ key-link pairs in other nodes.
- External nodes contain client keys.
- Internal nodes contain copies of keys to guide search.

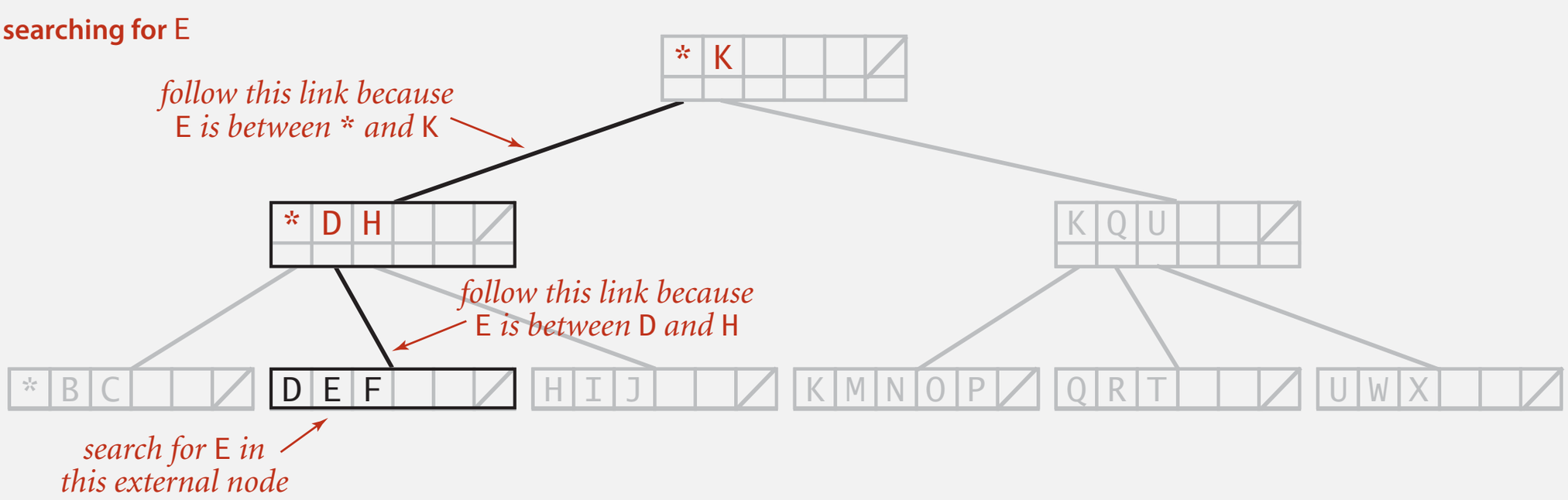
choose M as large as possible so that M links fit in a page, e.g., M = 1024



Anatomy of a B-tree set (M = 6)

Searching in a B-tree

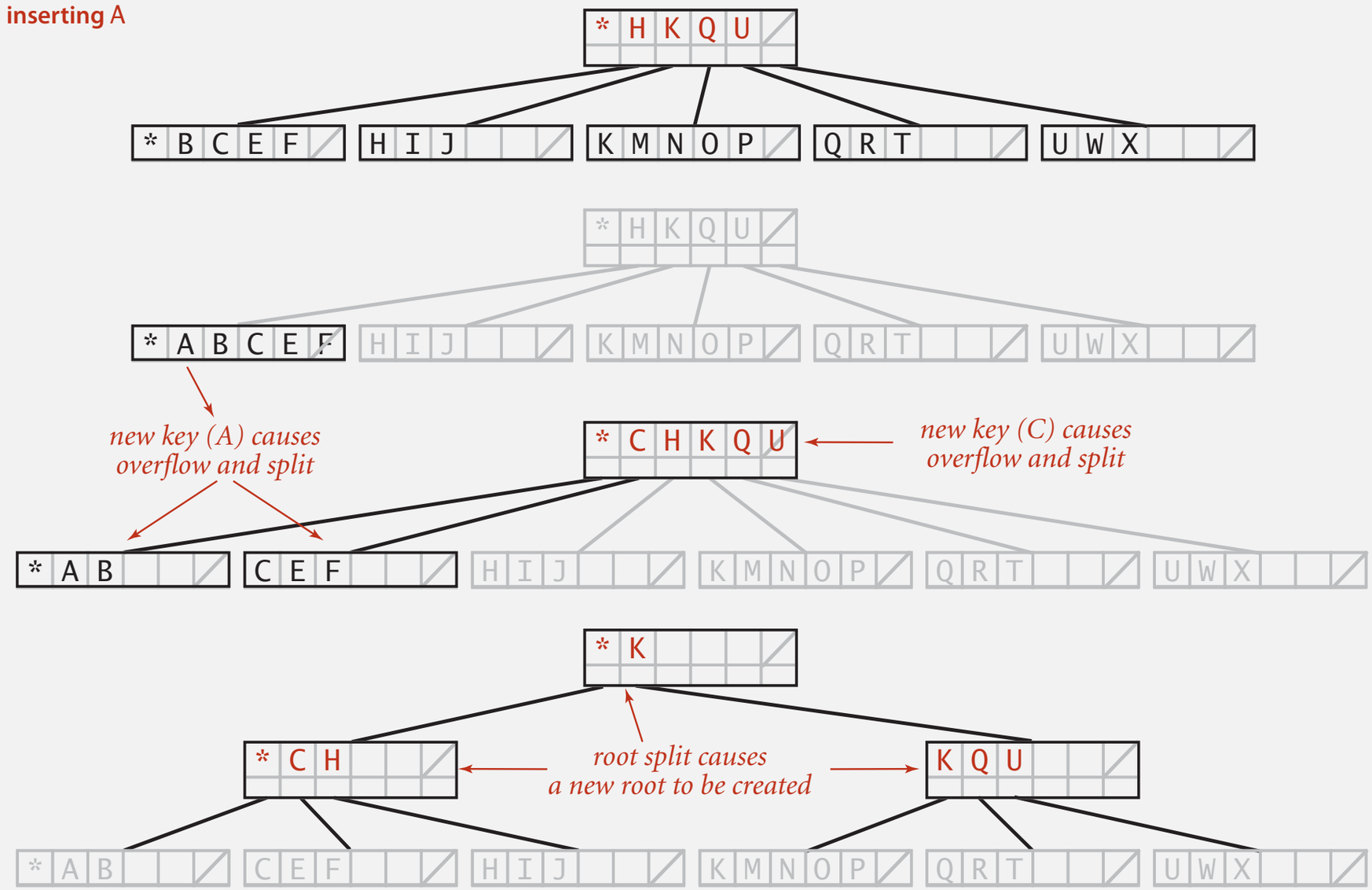
- Start at root.
- Find interval for search key and take corresponding link.
- Search terminates in external node.



Searching in a B-tree set (M = 6)

Insertion in a B-tree

- Search for new key.
- Insert at bottom.
- Split nodes with M key-link pairs on the way up the tree.



Inserting a new key into a B-tree set

http://algs4.cs.princeton.edu

3.3 BALANCED SEARCH

- *2-3 search trees*
- *red-black BSTs*
- *B-trees*