

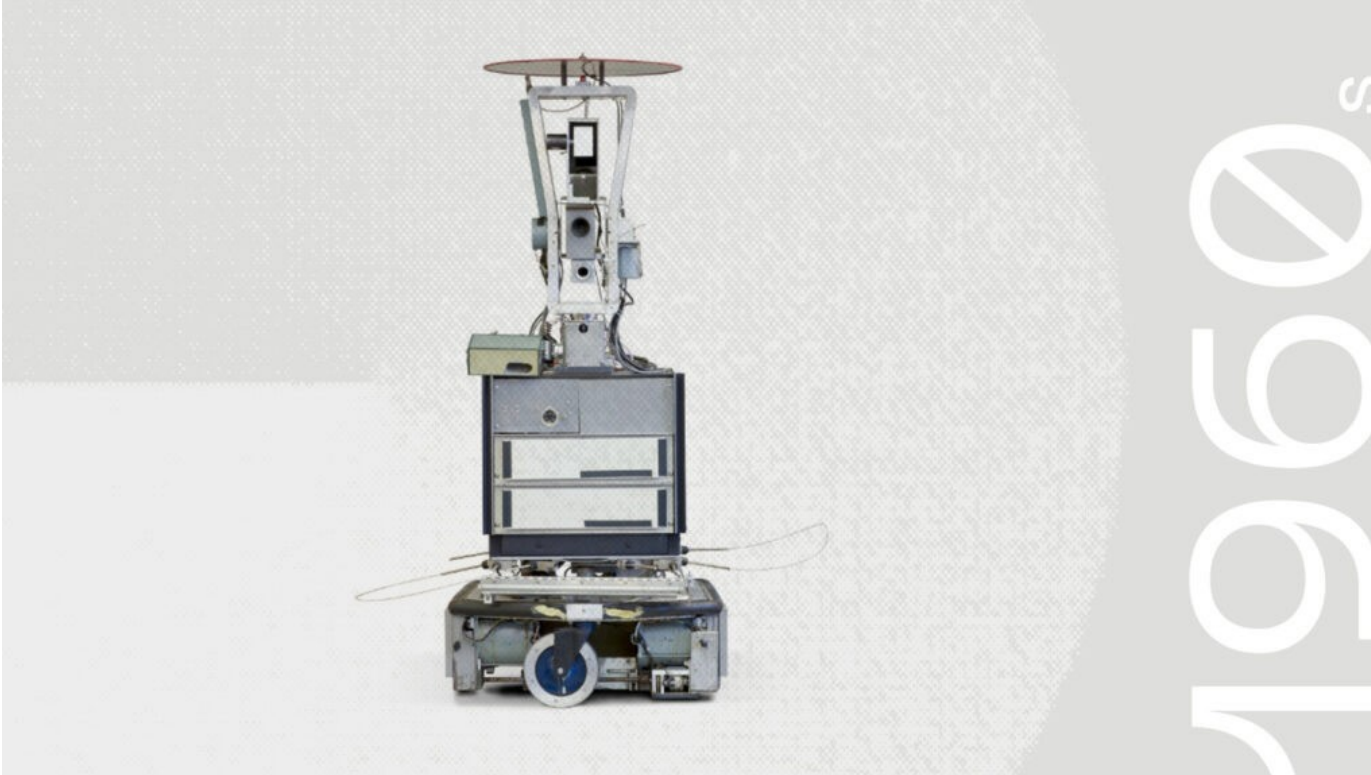
Artificial intelligence and data structures

Rules, search and learning

Linked lists, stacks, queues and trees

Prof. Dr. Mehmet Göktürk · Gebze Technical University

A robot that could plan



A camera observes the room.

A planner chooses actions.

The robot moves and checks the result.

Shakey at SRI, 1966–1972. Photograph: SRI International.

What counts as an intelligent system?

Perception turns observations into a useful representation.

Reasoning or learning helps choose an action under constraints.

Success depends on the task, environment and evidence used to evaluate it.

AI, machine learning and deep learning

Term	Main idea	Example
Artificial intelligence	Systems that perform tasks requiring intelligent behavior	Planning a route through a room
Machine learning	Fit a model using data or experience	Learn to recognize faulty parts
Deep learning	Learn using neural networks with many layers	Image or language models

Machine learning is one family of AI methods. Deep learning is one family of machine learning.

Selected milestones

When	Example	Central question
1950 / 1956	Turing's imitation game / Dartmouth project	How could machine intelligence be studied?
1966–1972	Shakey	Can perception support planning and action?
1997 / 2016	Deep Blue / AlphaGo	How can search and learned evaluation work together?
2017 onward	Transformers	How can models learn relationships in sequences?

These approaches overlap. New methods do not make every earlier method obsolete.

Turing's imitation game evaluates behavior through conversation.

Strong performance on one task does not establish general intelligence or consciousness.

Useful questions remain concrete: What task? Which test cases? What kinds of failure?

Explicit rules and learned patterns

A rule system uses conditions and conclusions that people specify.

A learning system adjusts model parameters using examples and an objective.

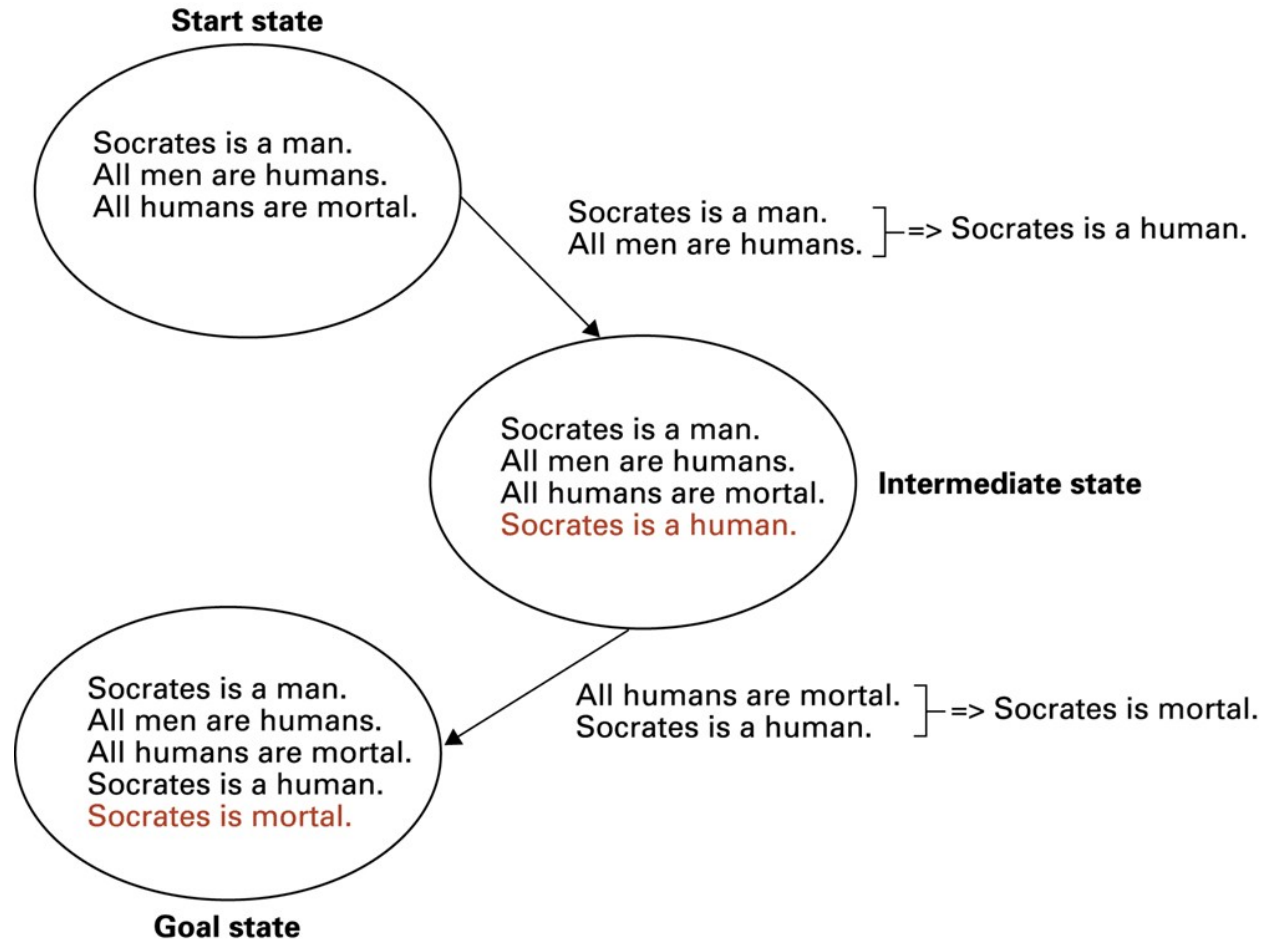
A complete application can combine learned predictions, explicit rules and search.

Expert systems: diagnostic rules

Knowledge	Rule or fact
Facts	Motor commanded ON. Current is zero. Supply voltage is present.
Rule 1	IF current is zero AND supply is present, THEN inspect the motor circuit.
Rule 2	IF a connector is loose, THEN reconnect it before retesting.

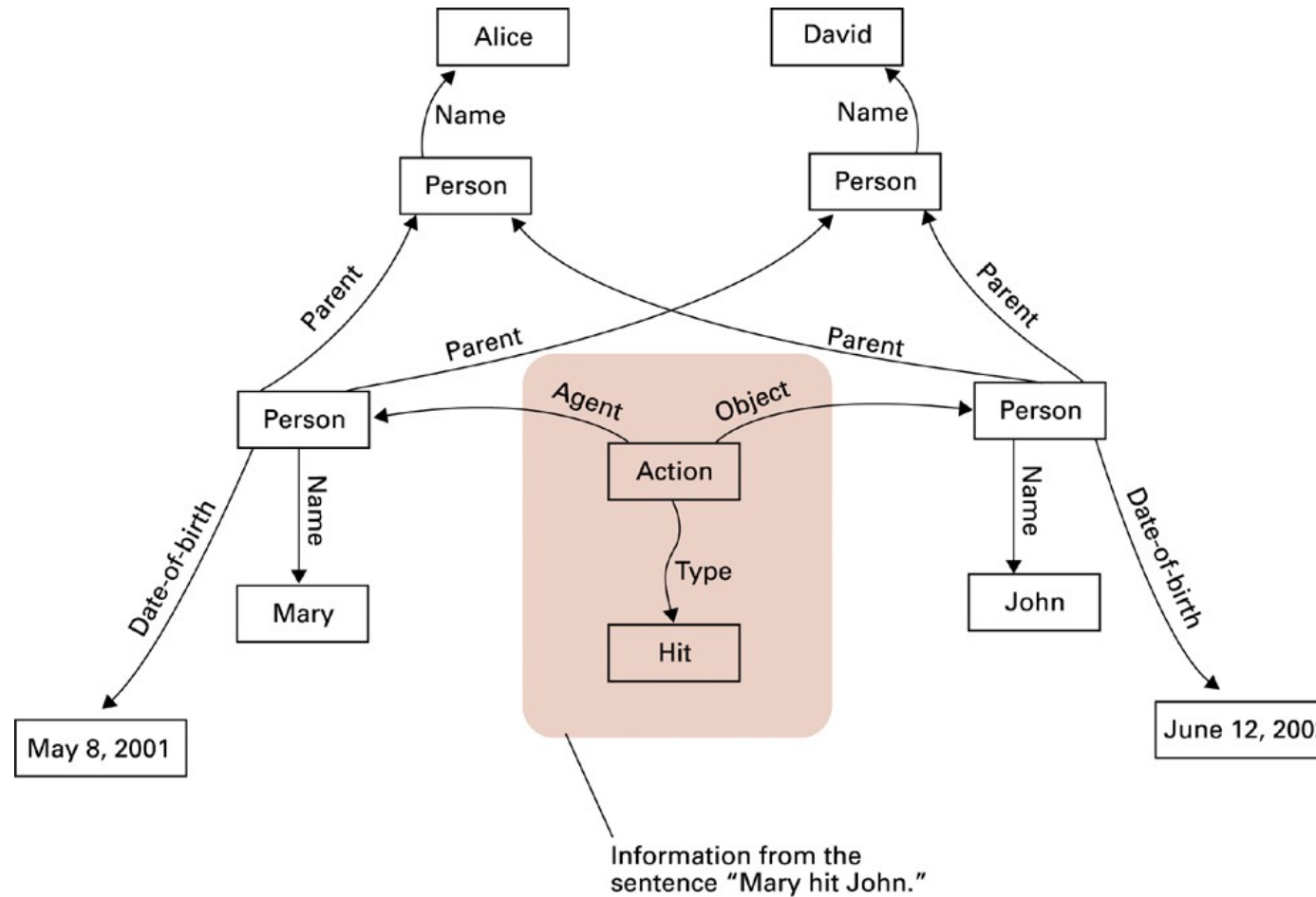
The system explains which rule fired. Missing or incorrect rules still produce poor advice.

Deduction with explicit knowledge



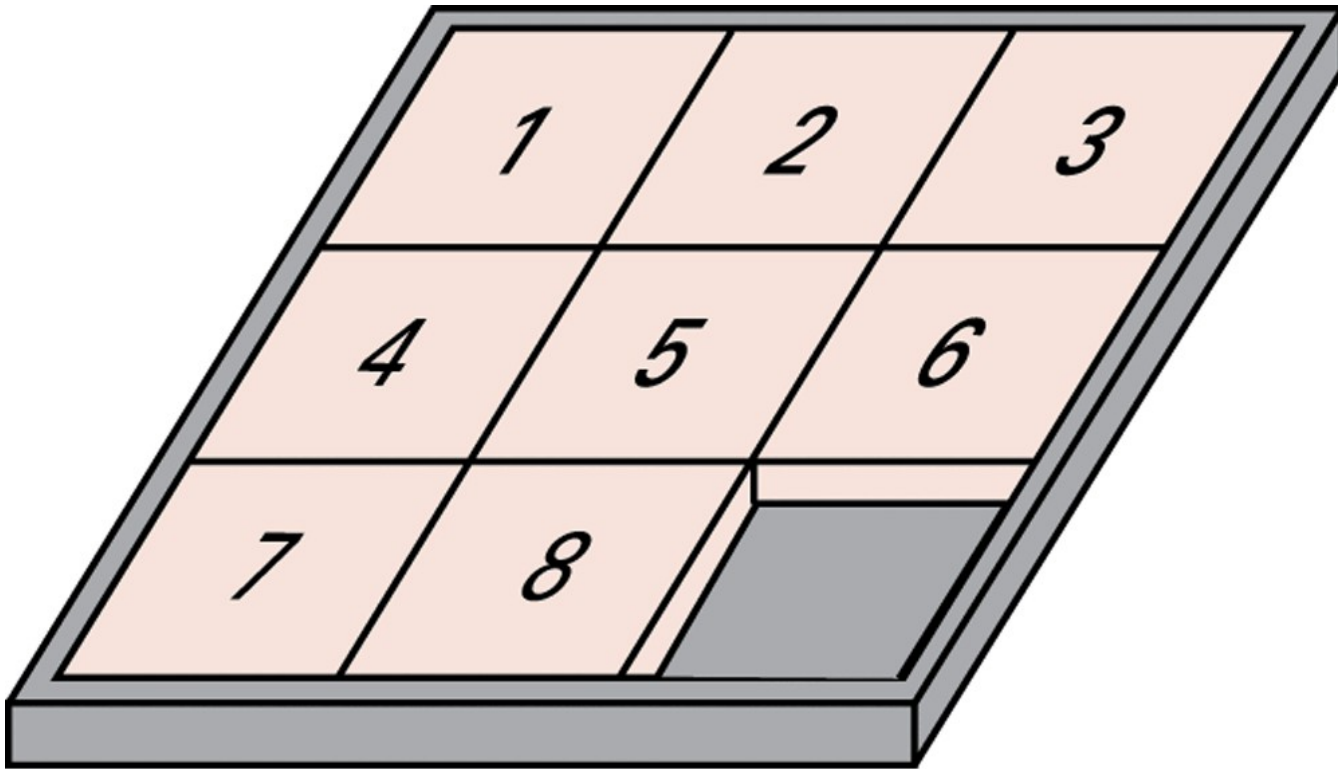
Facts and rules produce a new conclusion. The conclusion depends on the premises.

Knowledge represented as relationships



A semantic network records concepts and relationships. Its links have explicit meanings.

A puzzle as a search problem



State: the arrangement of tiles.

Action: slide a neighbor into the empty position.

Goal test: compare with this arrangement.

Goal state: every numbered tile in its target position.

A starting state

1	3	5
4	2	
7	8	6

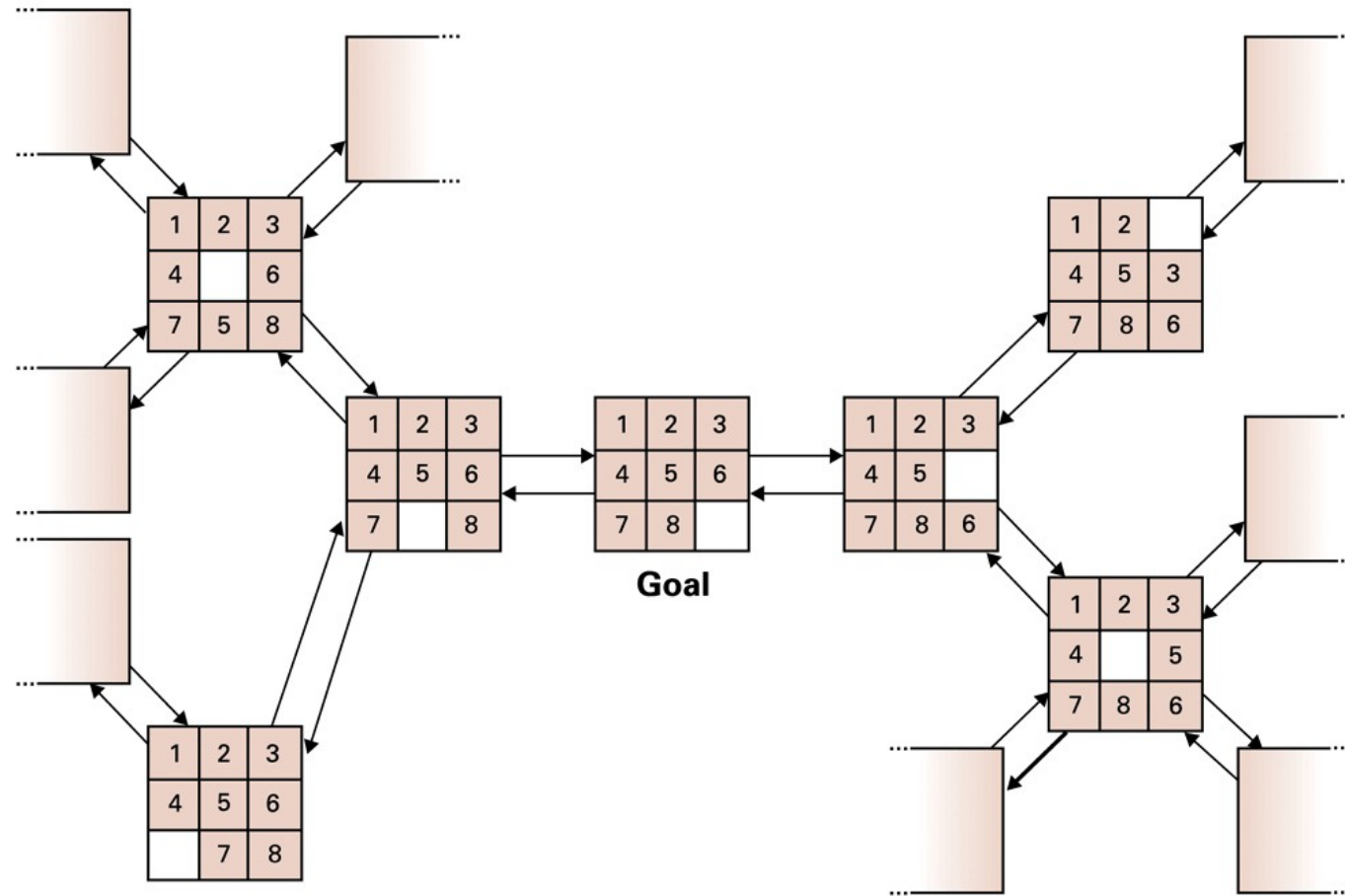
Which actions are legal here?

Several choices can lead away from the goal.

Trying moves creates a search tree.

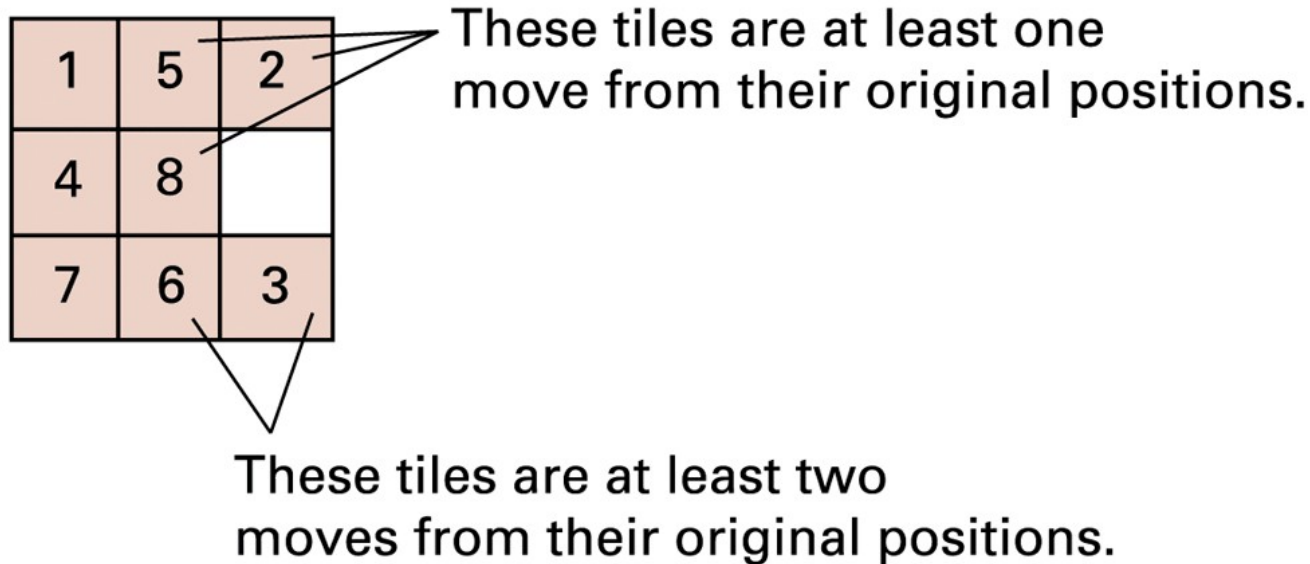
A solver needs a sequence of legal moves, not only the final arrangement.

States and possible transitions



A state graph describes possible transitions. A search tree records the exploration of alternatives.

A heuristic estimates remaining work



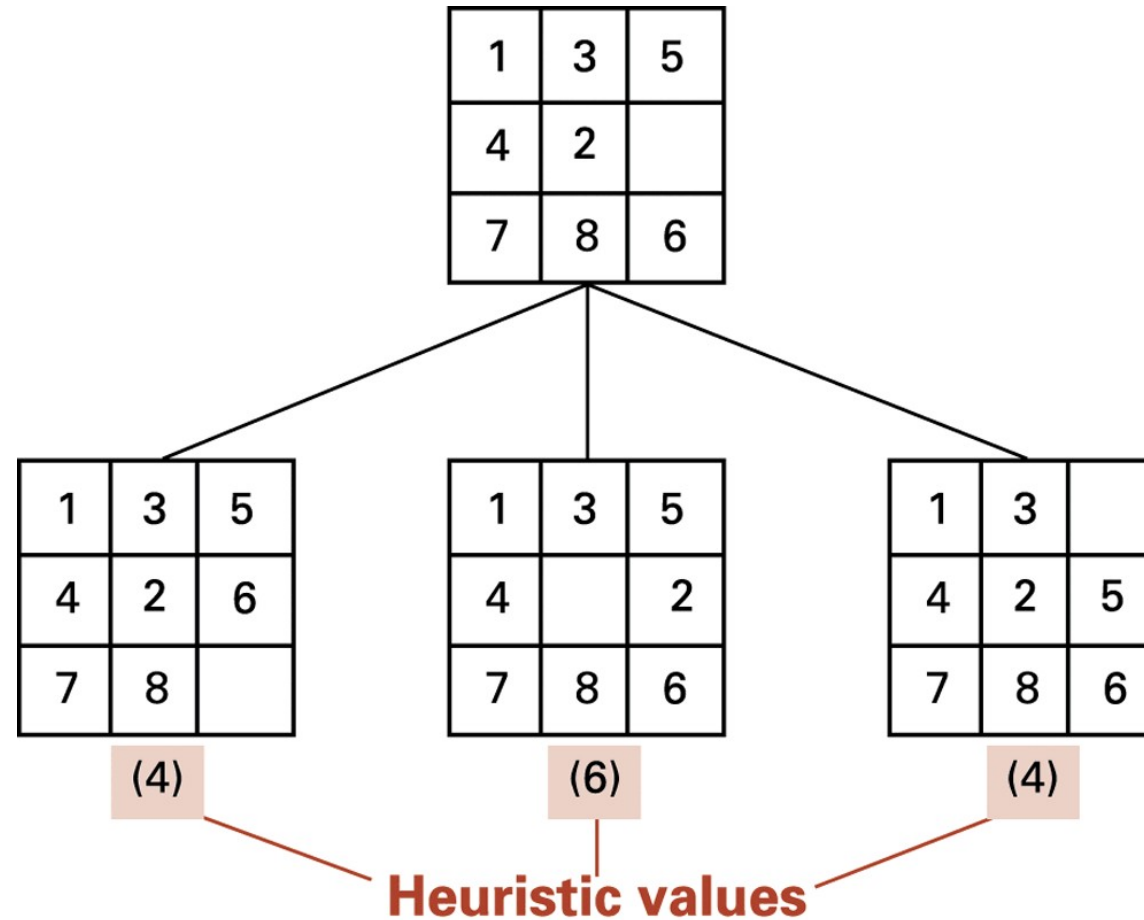
Tiles 5, 2 and 8: 1 move each.

Tiles 6 and 3: 2 moves each.

Total estimate $h = 7$.

Manhattan distance adds each numbered tile's row and column distance from its goal. The blank is excluded.

Choosing which state to explore



This example ranks candidates by estimated distance h . Smaller h looks more promising.

Search still requires judgment

A heuristic guides exploration. It does not automatically guarantee the best route.

Greedy best-first search ranks by h . A^* ranks by work already done plus estimated work remaining: $g + h$.

Repeated states need handling. Otherwise a solver can keep revisiting the same positions.

Deep Blue and chess



Deep Blue defeated Kasparov in their 1997 match.

Search examines possible continuations.

Evaluation estimates the strength of positions.

1997 rematch: Garry Kasparov and IBM operator Joe Hoane. Photograph: IBM.

AlphaGo combines learning and search



DeepMind team watching the match. Photograph: Google DeepMind.

Neural networks help select moves and assess positions.

Search explores continuations.

Training used expert games and reinforcement learning.

Training examples provide observations and, in supervised learning, target answers.

A model produces predictions. A loss measures how far they are from the targets.

An optimization procedure adjusts parameters to reduce loss on the training data.

Several ways to learn

Learning setting	Feedback	Example
Supervised	An answer accompanies each example	Image with a defect/no-defect label
Unsupervised	No target answer for each example	Group similar sensor patterns
Self-supervised	The data supplies the prediction target	Predict a missing or next token
Reinforcement	Actions lead to rewards over time	Learn choices in a game

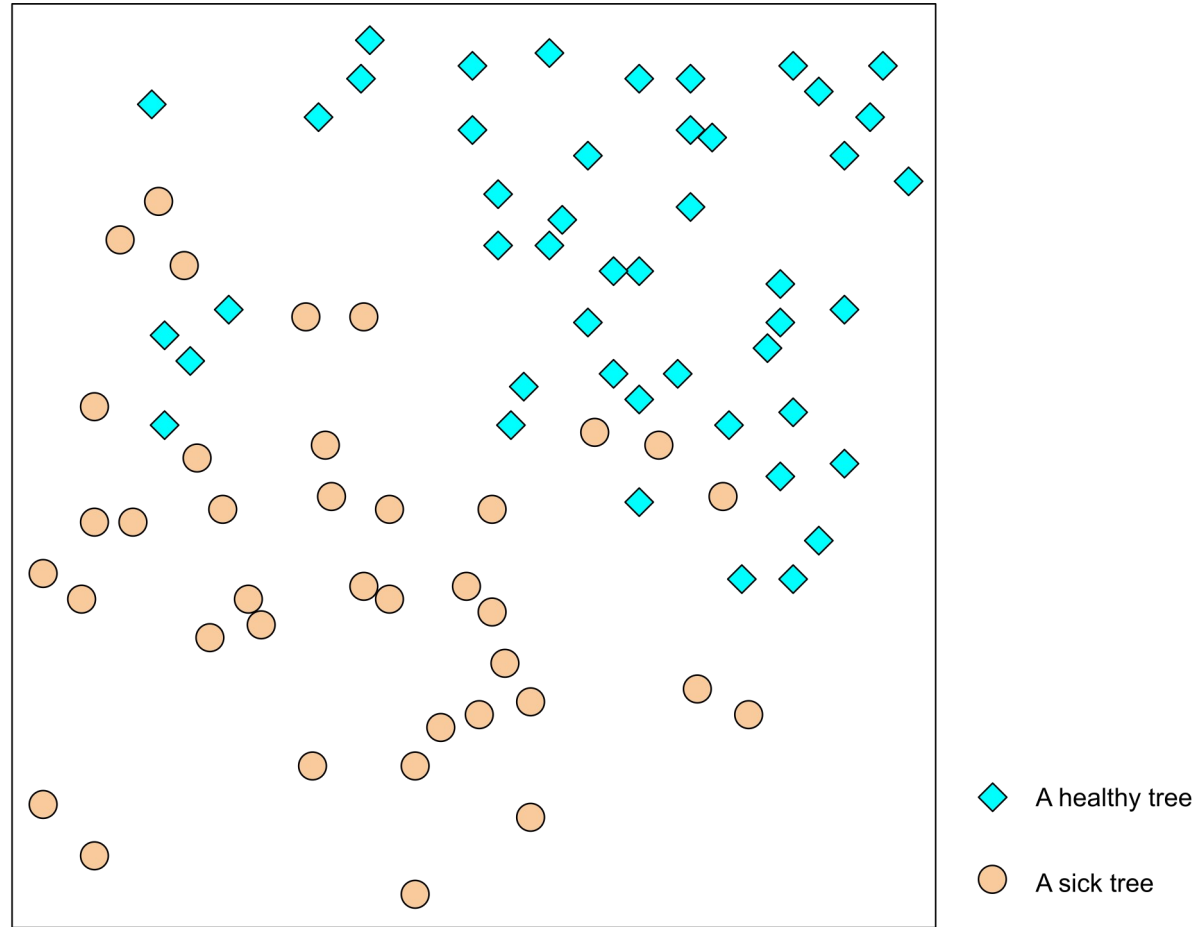
The available feedback determines the learning problem.

Training and inference

Stage	What happens	What changes?
Training	Process examples and reduce a loss	Model parameters
Inference	Apply the trained model to a new input	Input, activations and output
Further training	Update using additional examples	Parameters change again

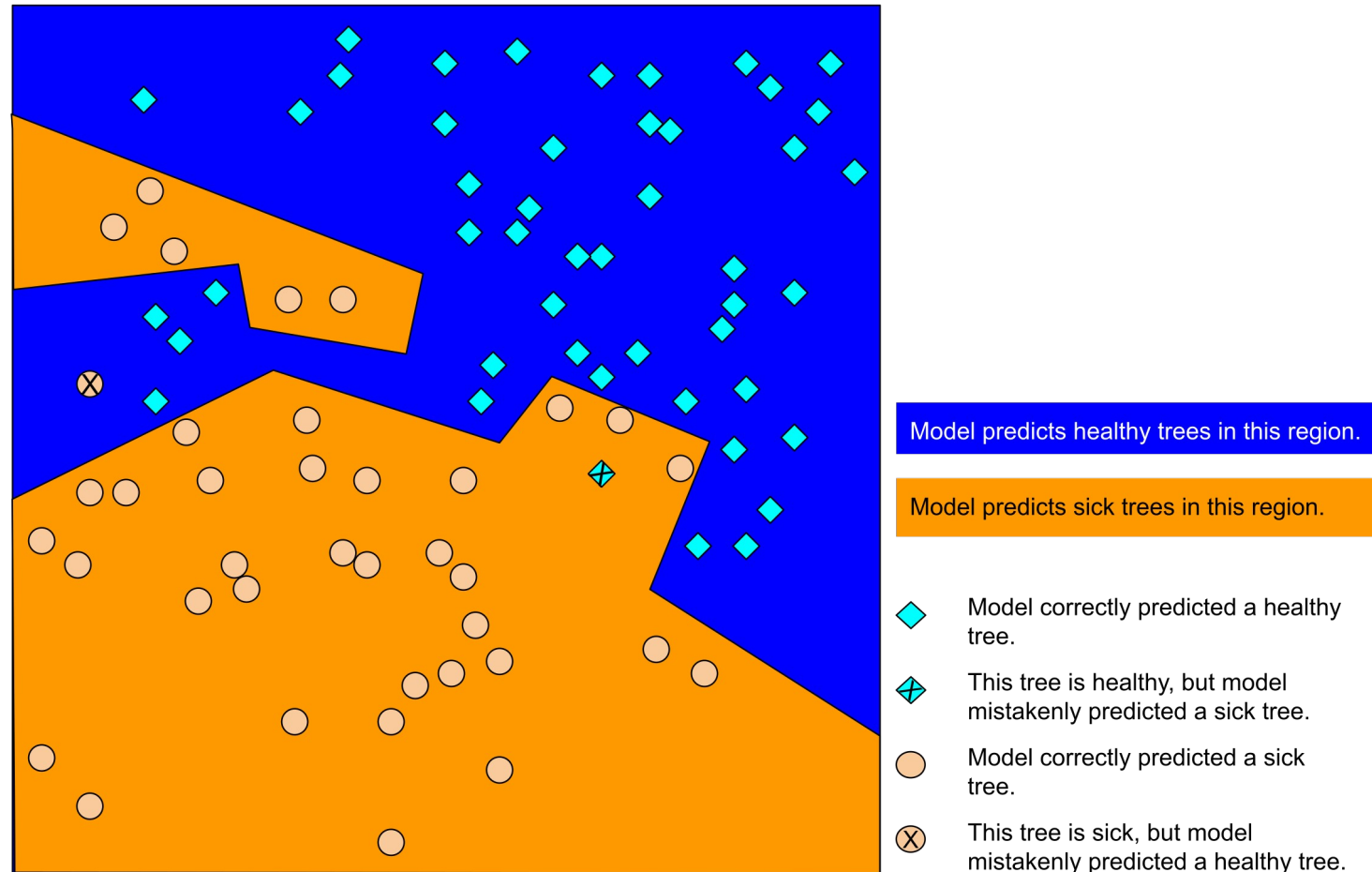
A new prompt changes the context. It does not by itself mean that the model's weights were retrained.

Examples used for training



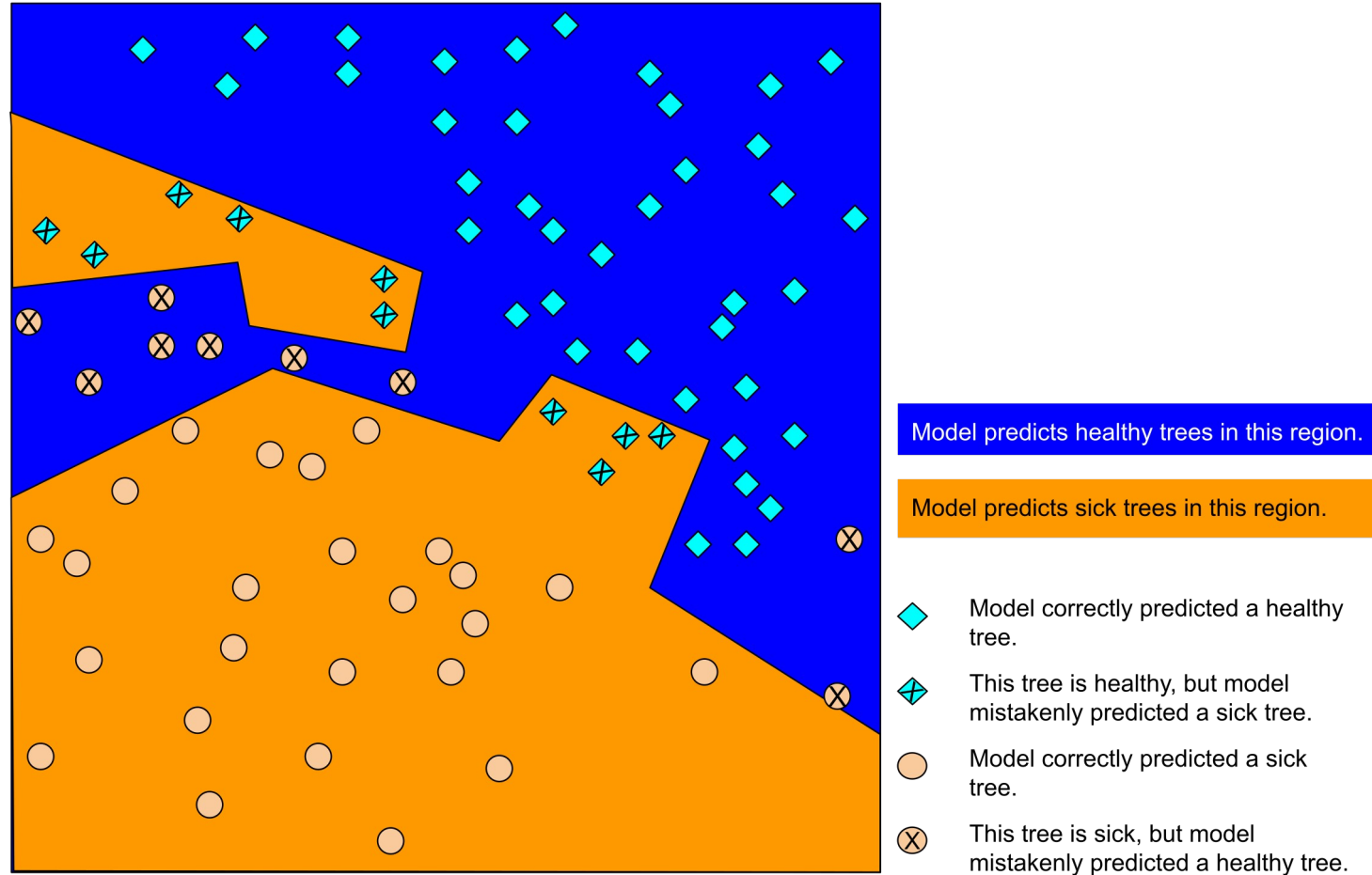
Diamonds and circles mark two classes of trees. Diagram: Google ML Crash Course, CC BY 4.0.

A boundary fitted to the training examples



A complicated boundary can fit many training examples. Diagram: Google ML Crash Course, CC BY 4.0.

Performance on new examples

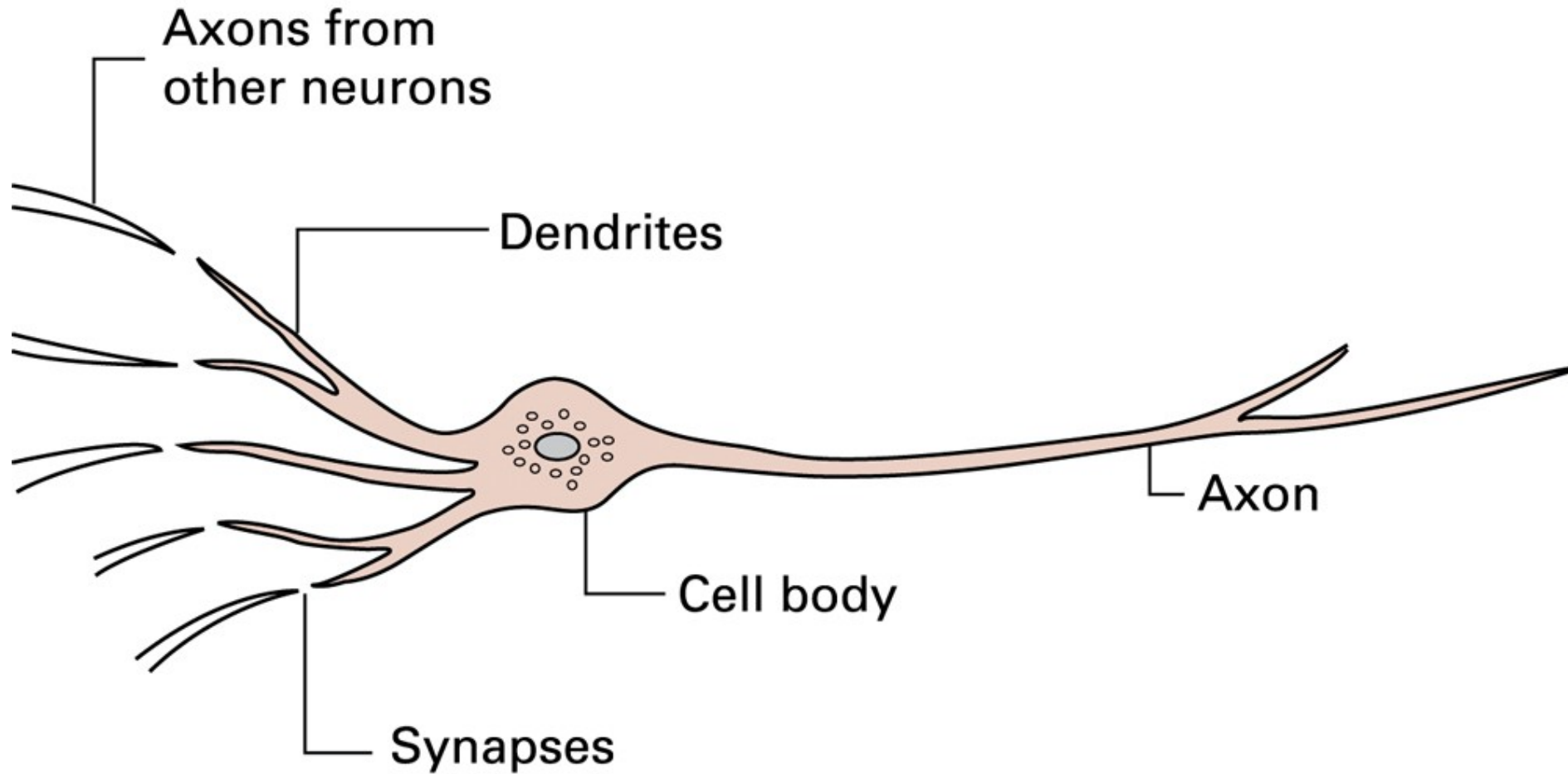


Crossed symbols show errors on test examples. Diagram: Google ML Crash Course, CC BY 4.0.

Training data fits the parameters. Validation data helps select the model and its settings.

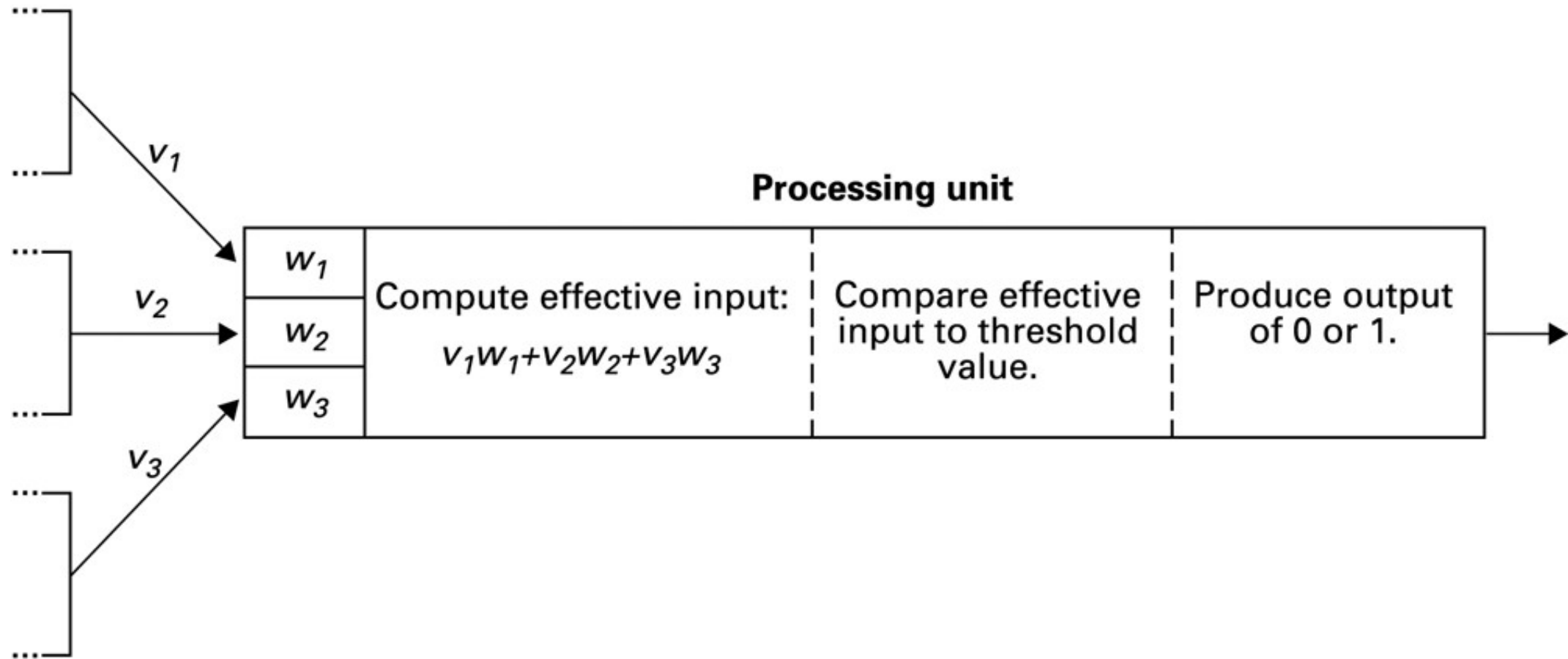
An untouched test set estimates performance after those choices.

Similar test conditions may hide weaknesses. New cameras, lighting or users can change performance.



Artificial neural networks borrow a simplified idea of connected units. They are not complete models of a brain.

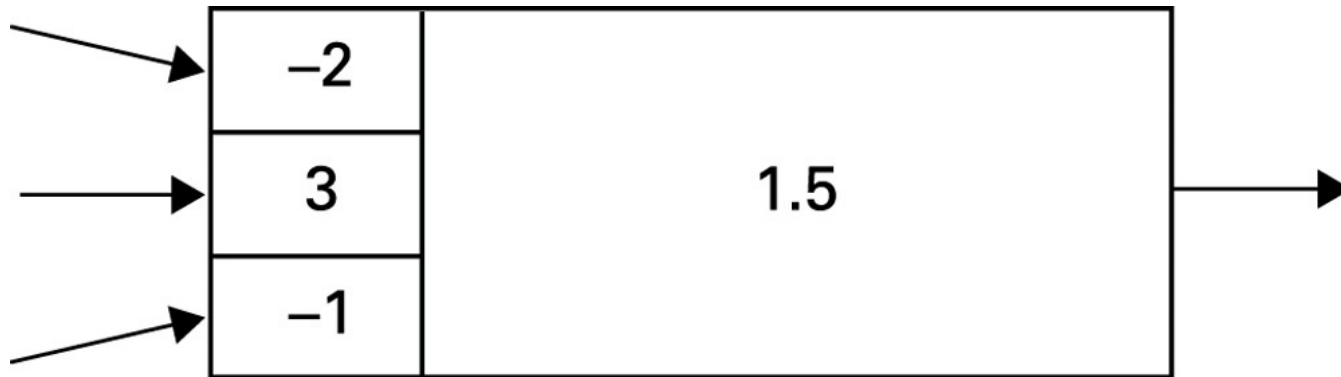
An artificial processing unit



Multiply inputs by weights, add the results, then apply an activation rule.

A small numerical example

Inputs (0, 1, 1):

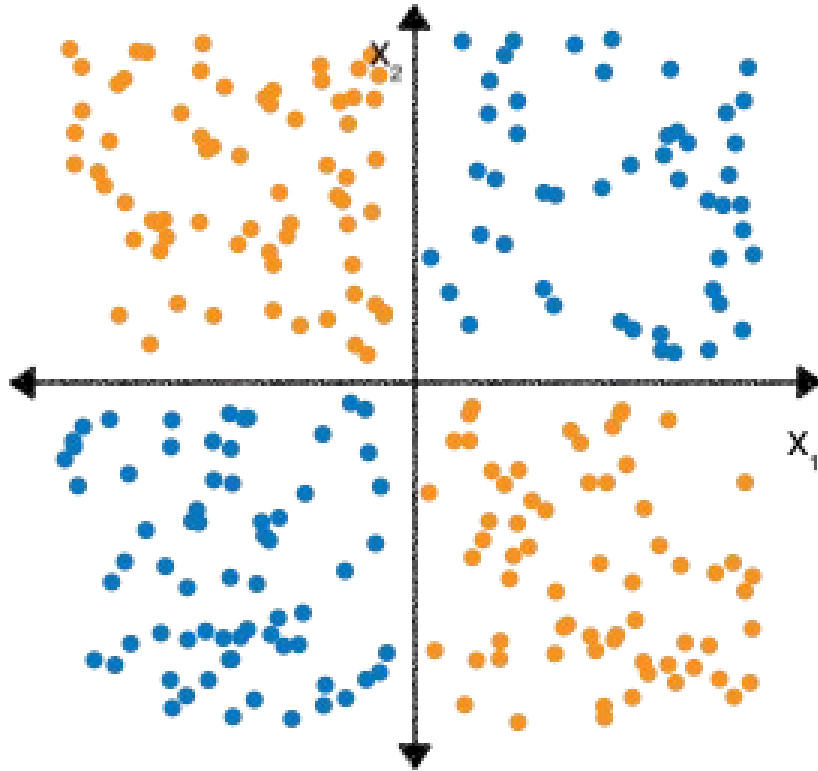


$$-2 \times 0 + 3 \times 1 - 1 \times 1 = 2$$

$2 > 1.5$, so output = 1.

Threshold rule in this example: output 1 when the weighted sum exceeds 1.5, otherwise output 0.

When one straight boundary is insufficient



A single straight line cannot separate these groups.

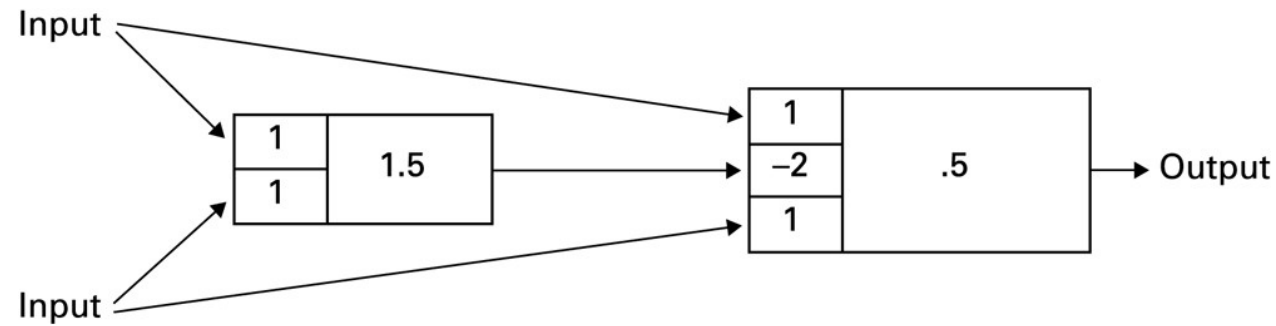
Hidden units and nonlinear activations can represent more complex boundaries.

Deep learning uses many layers of learned representations.

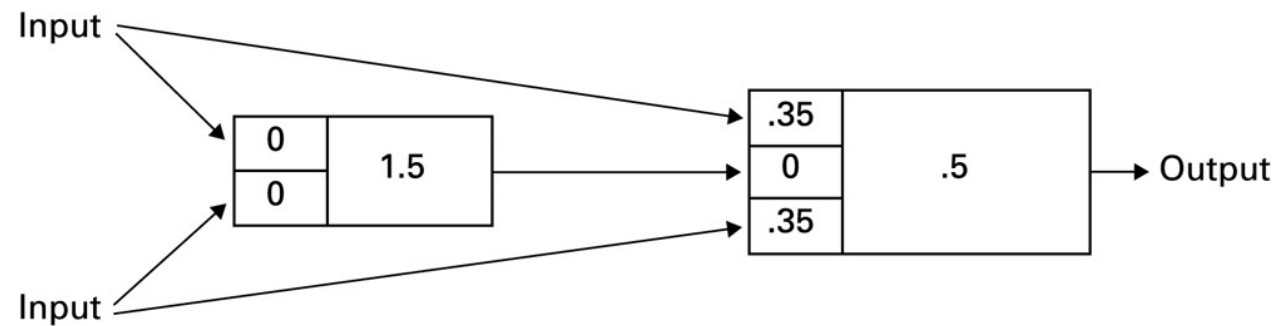
Two classes occupy alternating regions. Diagram: Google ML Crash Course, CC BY 4.0.

Connections alone do not define behavior

a.

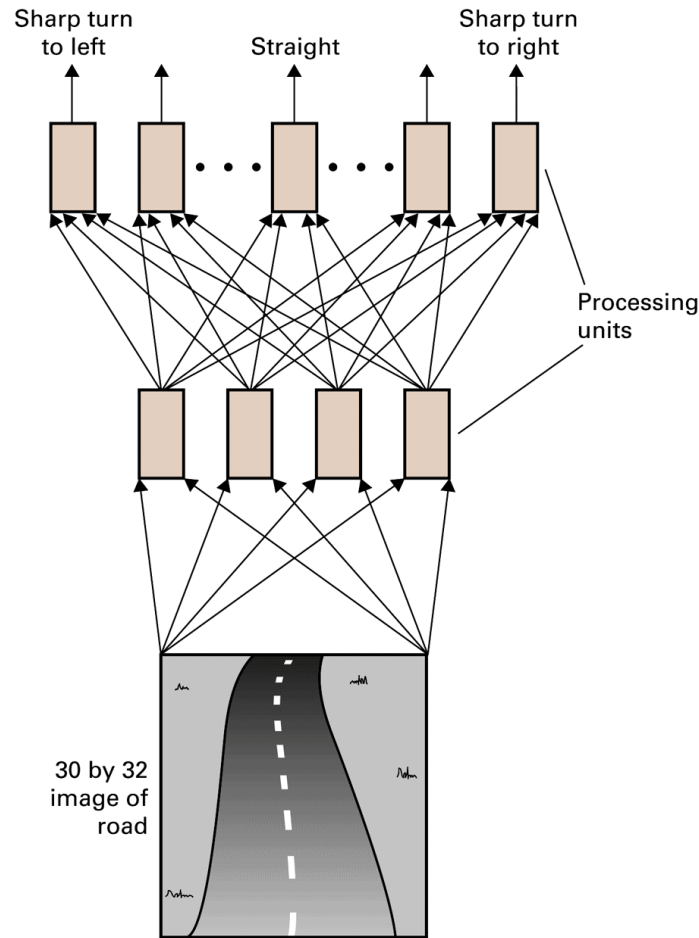


b.



The upper network computes XOR. The lower network has different weights and behaves differently.

Learning from camera images: ALVINN



Input: camera measurements.

Hidden units: intermediate computations.

Output: a steering choice.

Historical neural-network driving example. Image inputs connect through a hidden layer to steering outputs.

How neural-network training works

Run the input forward to obtain a prediction and a loss.

Backpropagation computes how the loss changes with each parameter. An optimizer uses these gradients to update weights.

Many examples and repeated updates are needed. A lower training loss alone does not establish useful performance.

A classifier predicts a category. A generative model produces a new sample, such as text or an image.

An autoregressive language model predicts the next token using earlier tokens as context.

A diffusion image model learns a denoising process and generates by repeatedly refining noise.

A next-token prediction

Context	Possible next token	Illustrative probability
The kettle contains hot	water	0.60
Same context	tea	0.25
Same context	coffee	0.10
Same context	all other tokens combined	0.05

Invented distribution for explanation. Actual tokens can be words, word parts or other text units.

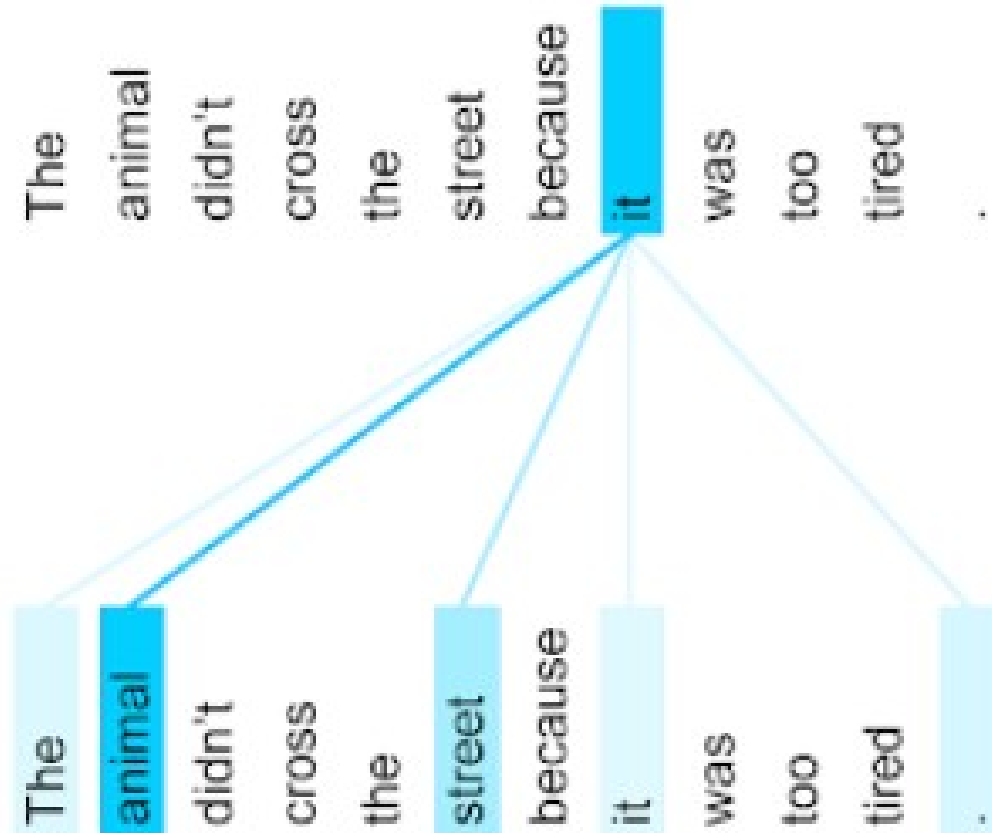
Generating a sequence

Select a token from the model's predicted distribution, according to the decoding method.

Append that token to the context and predict again. Repeat until a stopping condition.

The result depends on the prompt, model parameters and decoding choices. Fluent output can still contain false statements.

Attention uses relationships in context



Words become numerical representations.

Attention mixes information from relevant positions in a sequence.

Here “it” relates to “animal”. The picture illustrates a relationship, not proof of human understanding.

“The animal didn’t cross the street because it was too tired.” Diagram: Google ML Crash Course, CC BY 4.0.

A language model inside an application

A base model generates from its context. An application may also retrieve documents or call tools.

Retrieved information must still be checked for relevance and reliability. A tool result can also be wrong or misused.

Generating plausible text and verifying a claim are separate operations.

Evaluation beyond a single score

100 tested parts	Actually defective	Actually sound
Predicted defective	16	3
Predicted sound	4	77

Illustrative data: 93% accuracy, but 4 of 20 defective parts were missed. The cost of each error matters.

Biased or unrepresentative examples can produce unequal errors across groups or conditions.

A model can exploit a misleading clue, such as a background that happens to match a label.

Use evidence matched to the intended setting. Record uncertainty and provide a way to detect or correct failures.

Check: rules, learning and evidence

A system predicts a defective part from a photograph. Where did the decision rule come from?

It scores 99% on the images used for training. What evidence is missing?

A chatbot gives a confident answer with a citation. What should you verify?

Check: discussion

The decision may come from explicit human rules or parameters fitted to examples. Inspect how the system was built.

We need separate evaluation data and a test that resembles the actual use conditions.

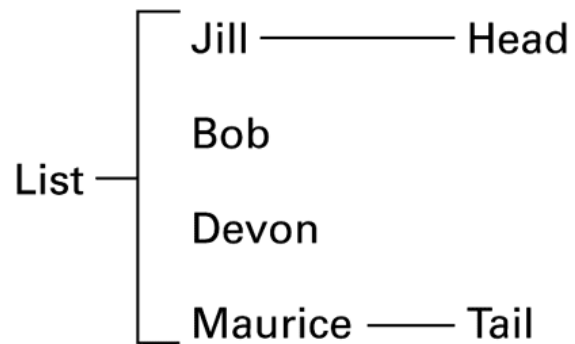
Check that the cited source exists, supports the claim and applies to the question.

Linked lists

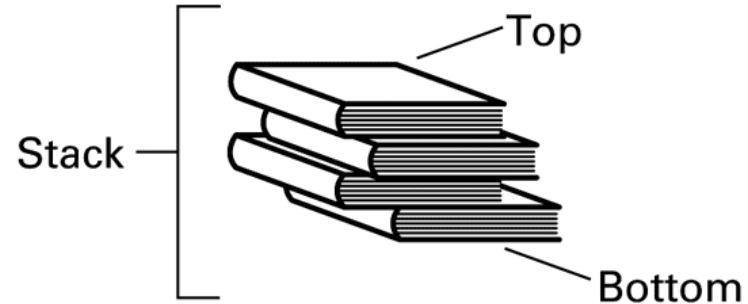
Linked lists, stacks and queues

Tree representation, traversal and insertion

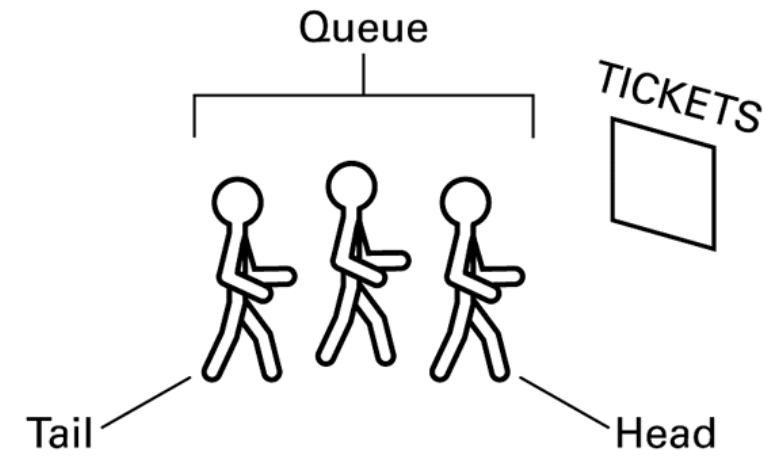
Lists, stacks and queues



a. A list of names



b. A stack of books

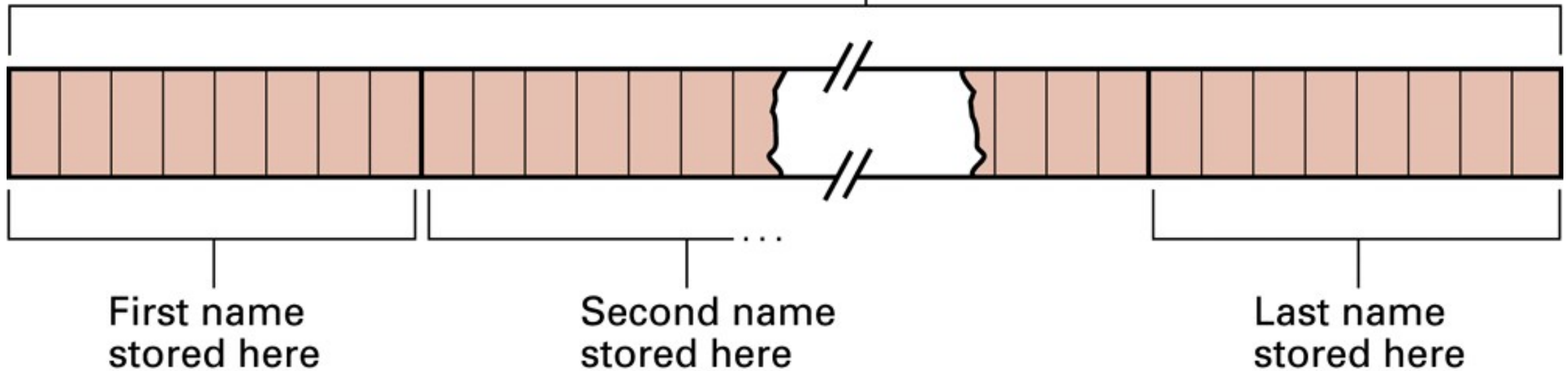


c. A queue of people

These are different access rules. Their implementation can use arrays or linked nodes.

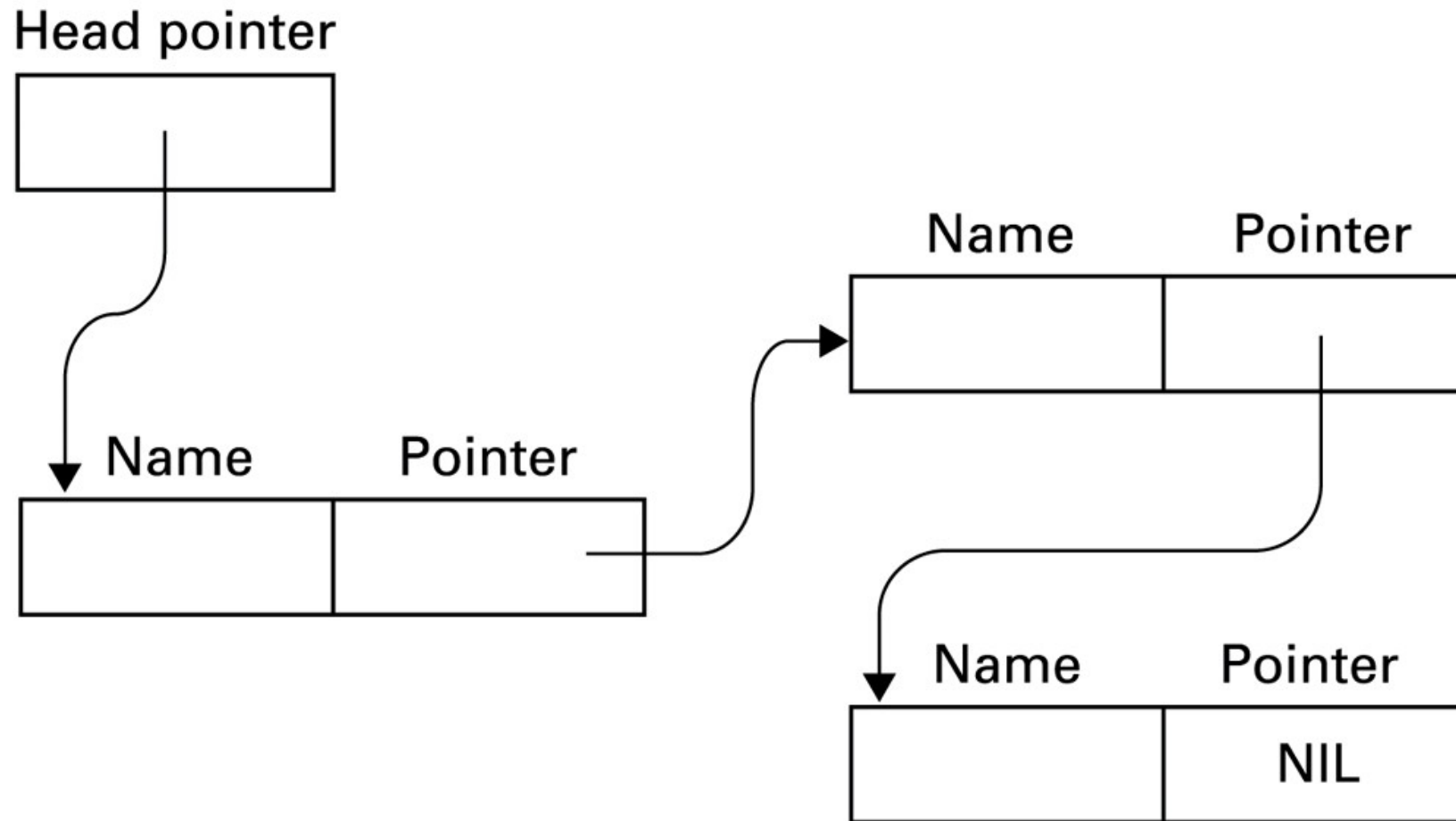
Contiguous storage

Contiguous block of memory cells



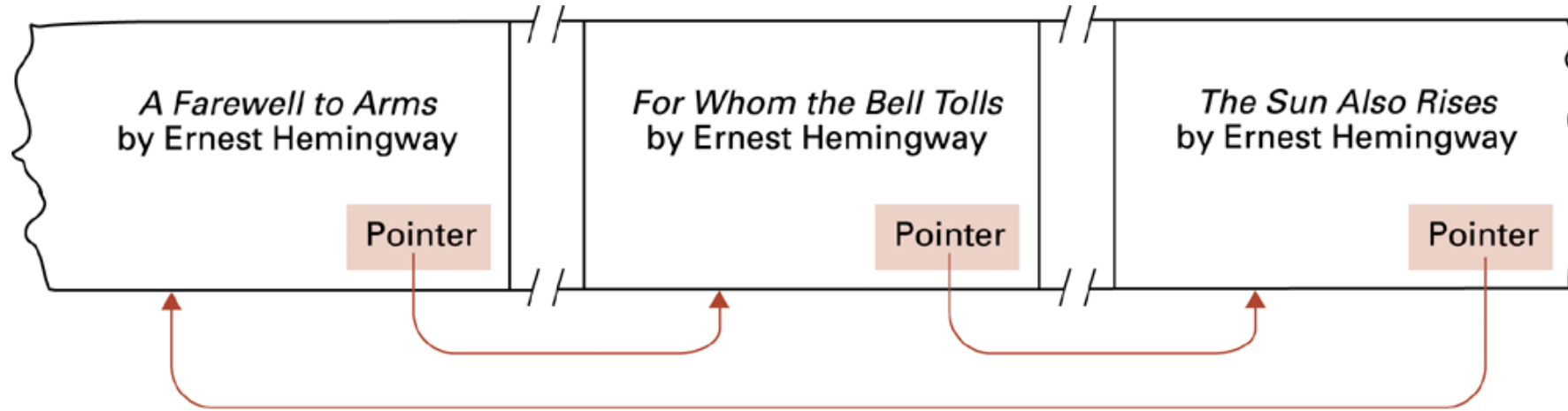
An array keeps successive elements together. Inserting in the middle may require moving later elements.

Linked storage



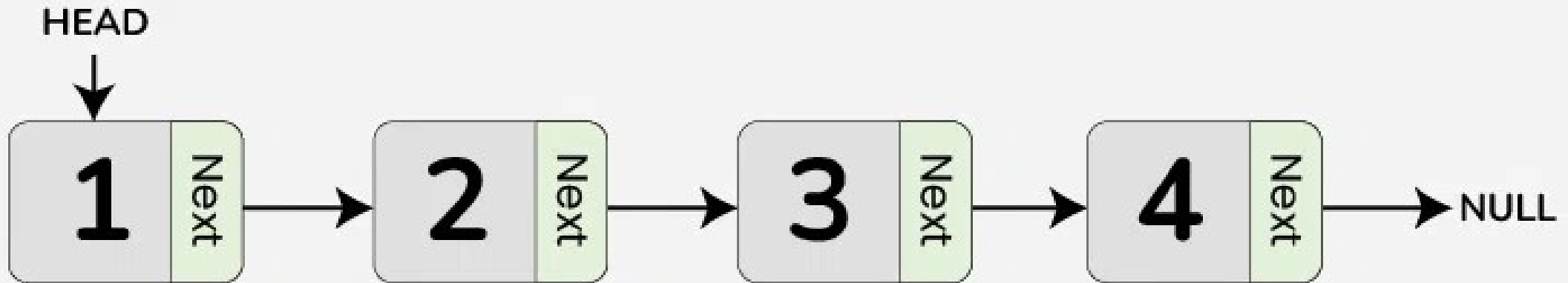
Each node stores data and a link. Physical location and logical sequence can differ.

Two orders for the same collection



Books remain arranged by title while links provide an authorship order. Original instructor analogy and photograph.

Following one link at a time



Singly Linked List

Start at head, visit each node and follow next until NULL.

An event record inside a node

```
typedef struct Node {  
    int event_id;  
    int value;  
    struct Node *next;  
} Node;
```

The record fields describe an event.

next holds the address of another Node.

A pointer can refer to its own structure type.

Three nodes with known lifetimes

```
Node c = {103, 30, NULL};  
Node b = {102, 20, &c};  
Node a = {101, 10, &b};  
Node *head = &a;
```

Logical order: 101, 102, 103.

NULL marks the end.

These local objects remain
alive throughout the trace.

```
void print_list(const Node *p)
{
    while (p != NULL) {
        printf("%d: %d\n",
               p->event_id, p->value);
        p = p->next;
    }
}
```

Test p before dereferencing it.

Advance to the next node.

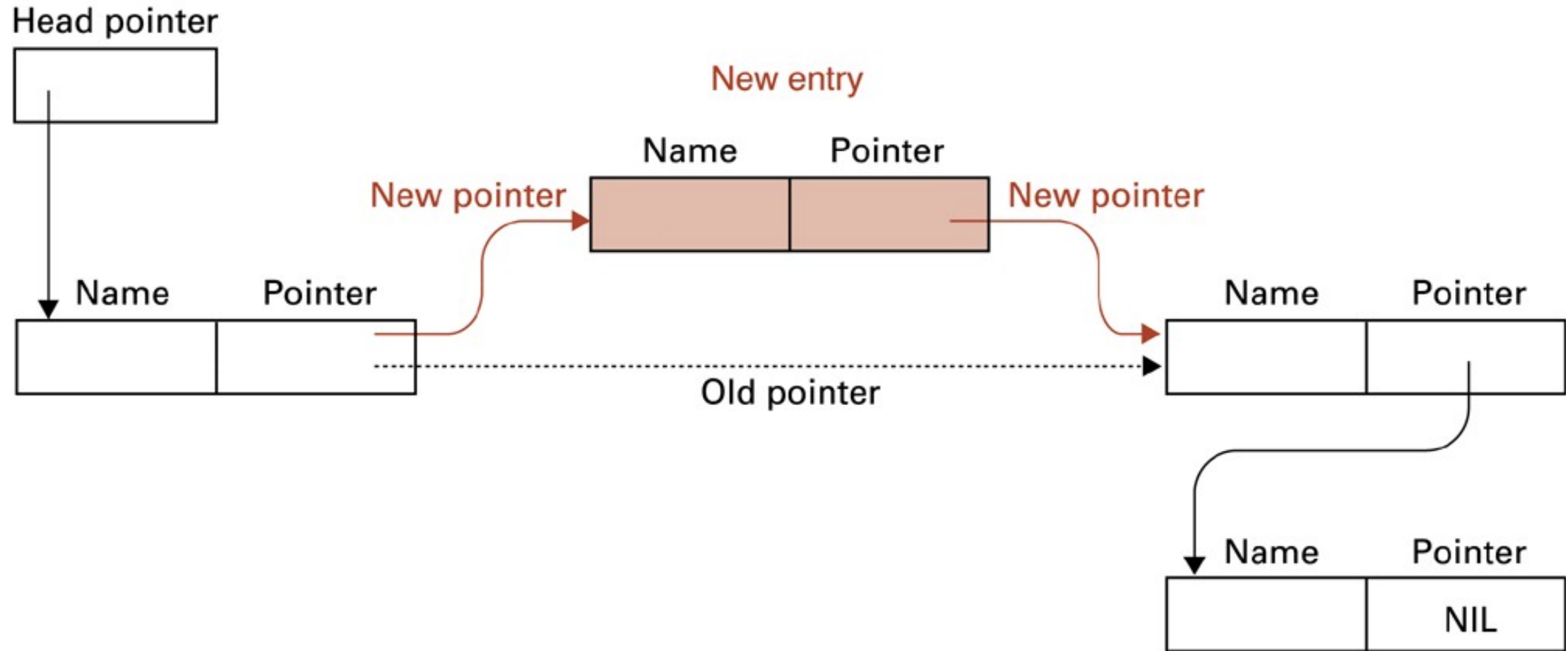
The caller's head pointer is unchanged.

Traversal trace

Iteration	p refers to	Printed	Next p
1	a	101: 10	&b
2	b	102: 20	&c
3	c	103: 30	NULL
Stop	NULL	Nothing	Loop ends

An empty list has head == NULL and prints no records.

Insertion changes links



Preserve the old successor before redirecting the predecessor. No other node needs to move.

Inserting after a known node

```
Node x = {104, 40, NULL};
```

```
x.next = a.next;
```

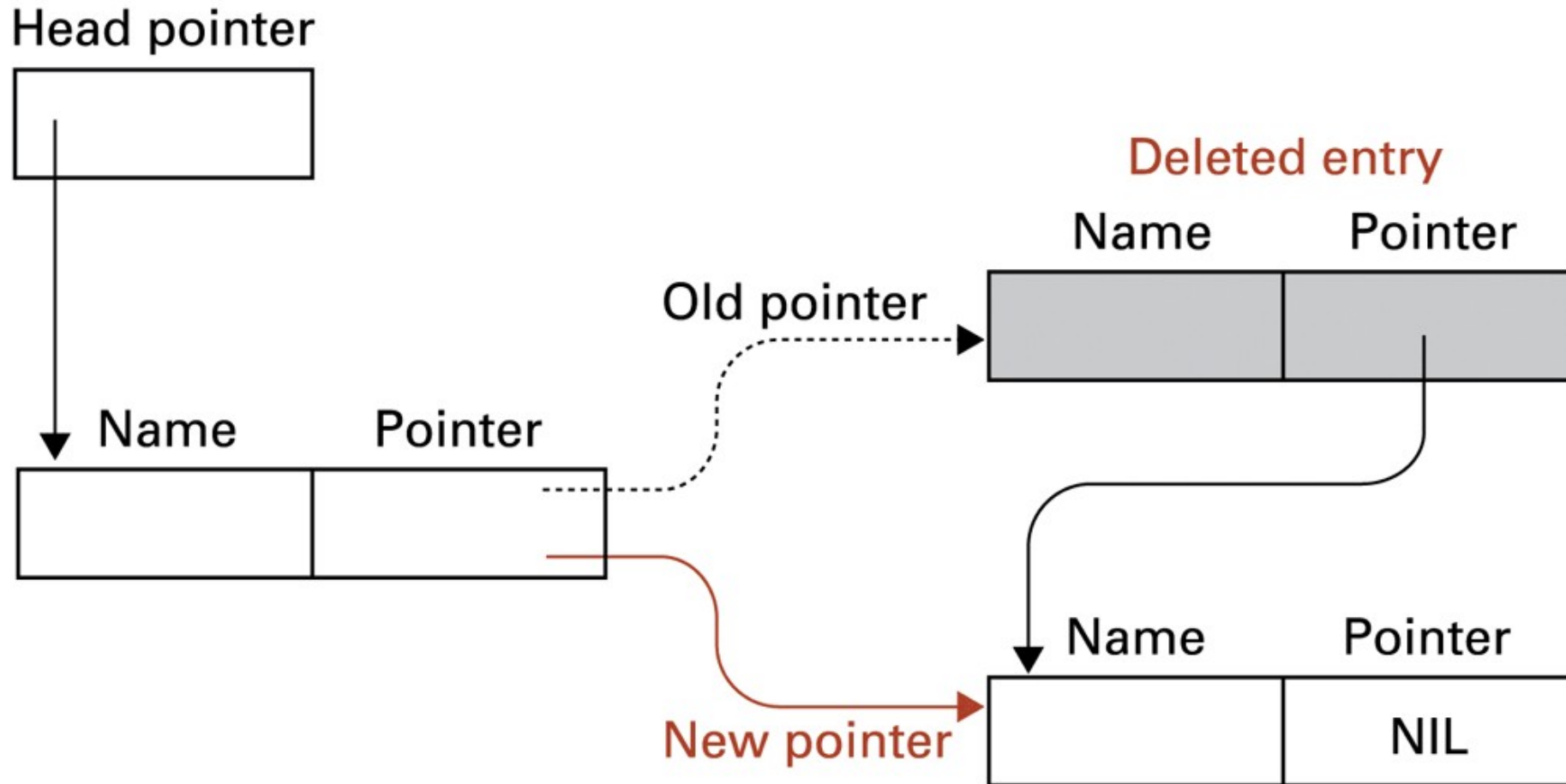
```
a.next = &x;
```

Before: 101, 102, 103.

After: 101, 104, 102, 103.

Reversing these two assignments would make x point to itself.

Removal changes reachability



Bypassing a node removes it from this list. It does not automatically release its storage.

Unlinking after a known predecessor

```
Node *victim = x.next;  
if (victim != NULL) {  
    x.next = victim->next;  
    victim->next = NULL;  
}
```

The removed node is b.

List order becomes 101, 104,
103.

No free here: b is a local
object, not a malloc
allocation.

Removing the first node

```
Node *pop_front(Node *head)
{
    if (head == NULL)
        return NULL;
    Node *rest = head->next;
    head->next = NULL;
    return rest;
}
```

```
head = pop_front(head);
```

An empty list remains empty.

A one-node list becomes empty.

The caller must store the returned head. This function only unlinks.

Common pointer failures

Mistake	Consequence	Useful check
Dereference NULL	Invalid access	Test before reading a field
Lose the old successor	Remaining nodes become unreachable	Save the link before replacing it
Make a cycle accidentally	Traversal may never finish	Trace the links after each change
Use a dead or freed node	Dangling pointer	Track who owns the storage

Drawing the links can expose an error before executing the code.

Array or linked list?

Operation	Array	Singly linked list
Find position i	Direct indexing	Follow links from the head
Insert after a known node	May shift later elements	Change a few links
Storage	Contiguous elements	Data plus pointer per node
Find the insertion position	Depends on the search	May require a full traversal

Fast relinking does not make the search for the right position free.

Check: a broken insertion

A list contains A followed by B. We want to insert X between them.

```
A.next = &X;  
X.next = A.next;
```

What does X.next point to? Which original link has been lost?

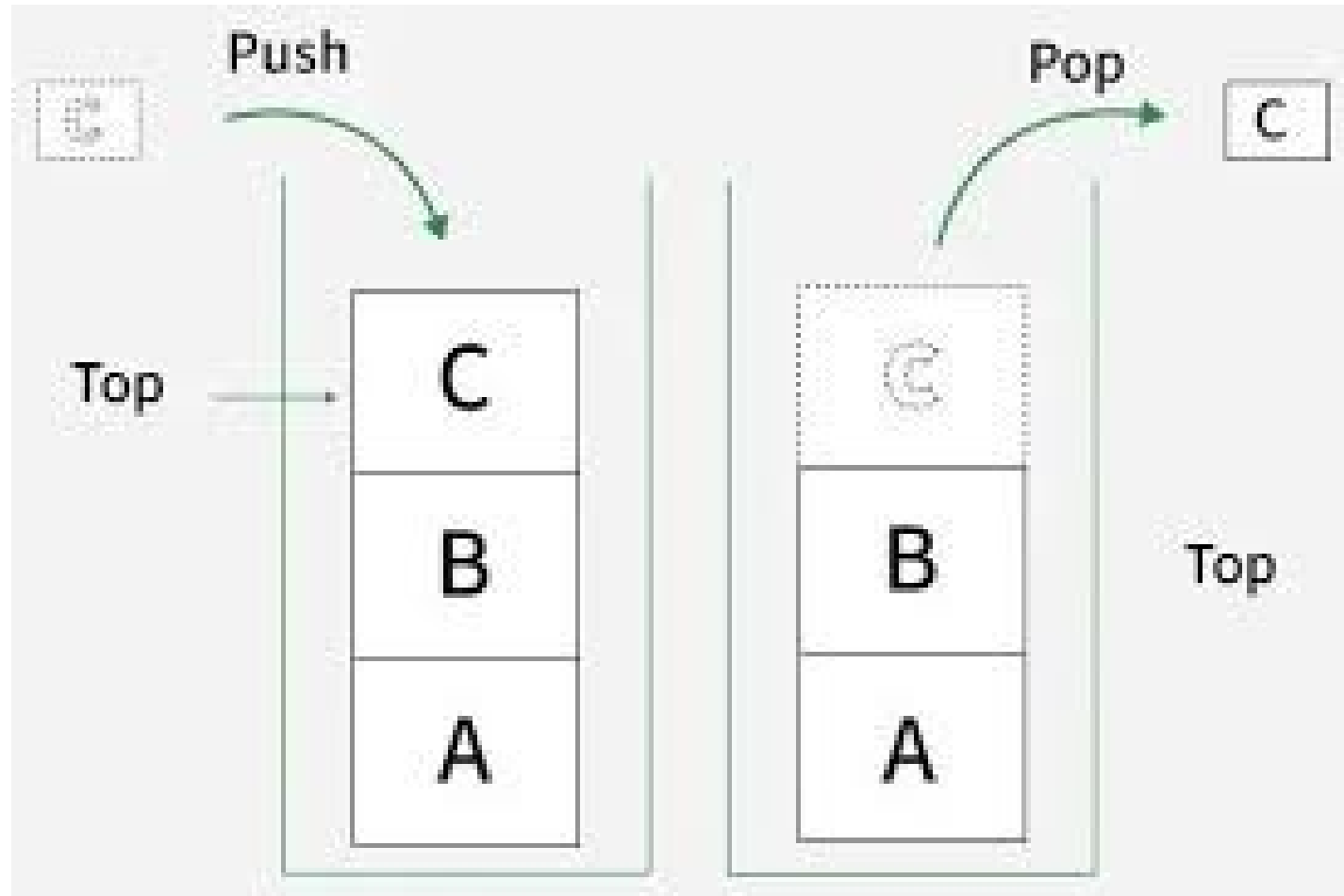
Check: explanation

After the first statement, A.next already points to X. The second statement therefore makes X.next point to X.

The original link from A to B has been overwritten. A traversal starting at A reaches a cycle.

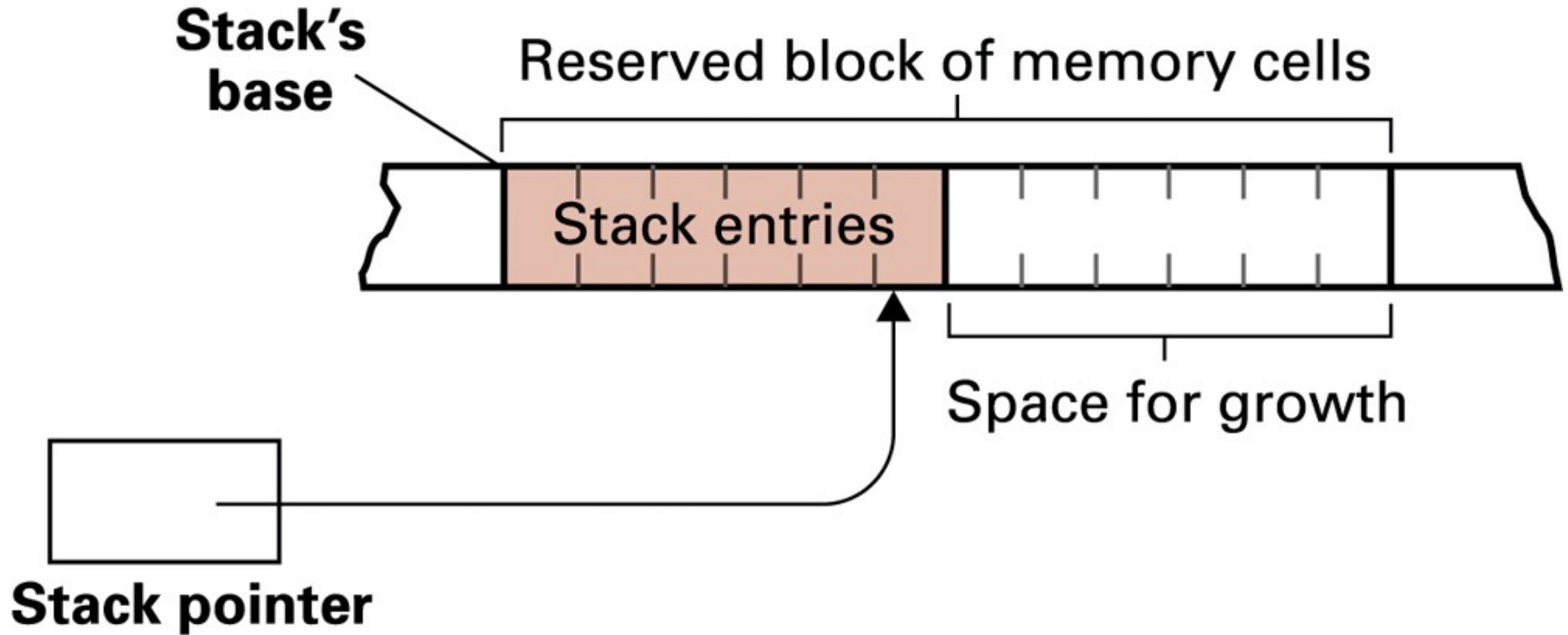
Correct order: first $X.next = A.next$, then $A.next = \&X$.

A stack with push and pop



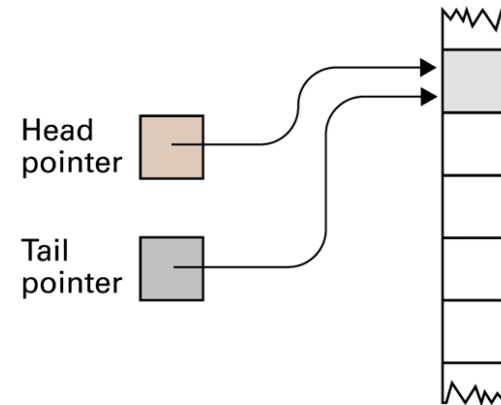
Push and pop act on the same end. This rule does not require one particular storage representation.

An array implementation of a stack

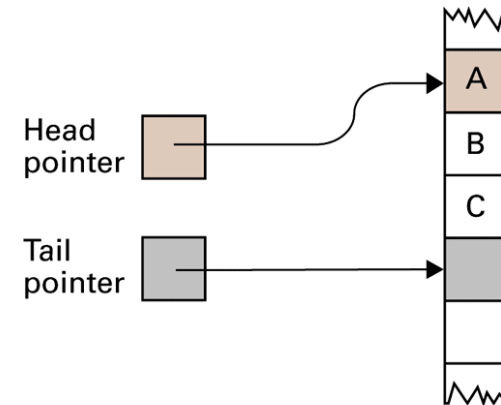


A stack can use an array. Its defining access rule is last in, first out.

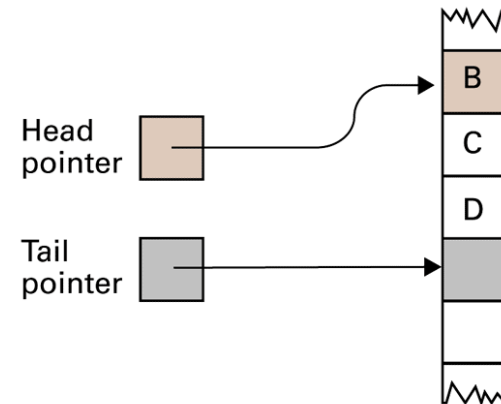
Queue head and tail



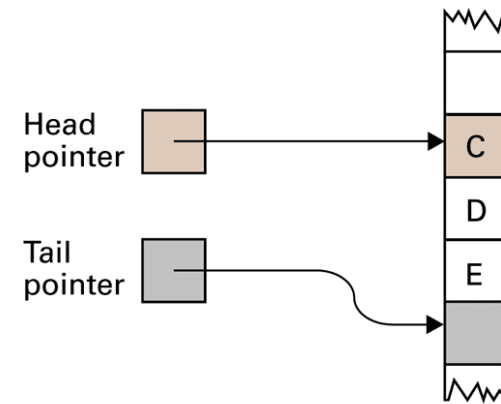
a. Empty queue



b. After inserting entries A, B, and C



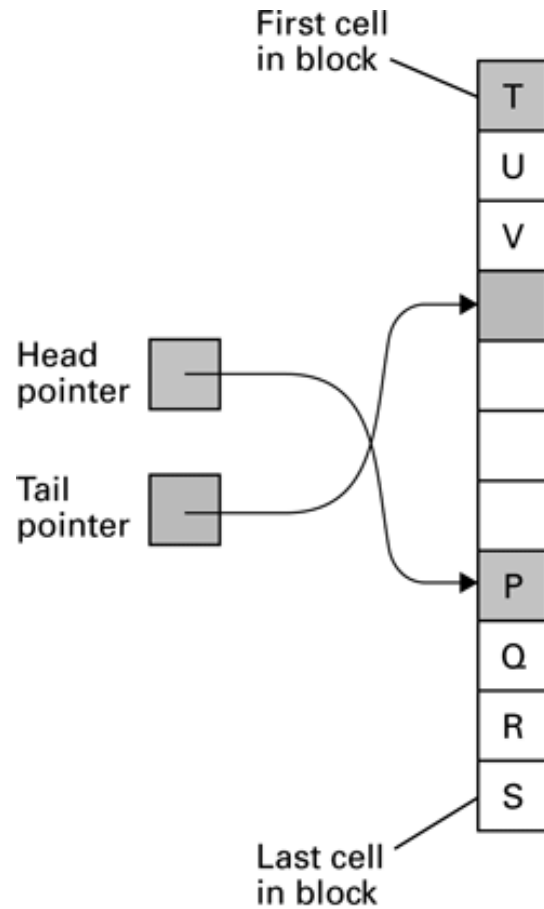
c. After removing A and inserting D



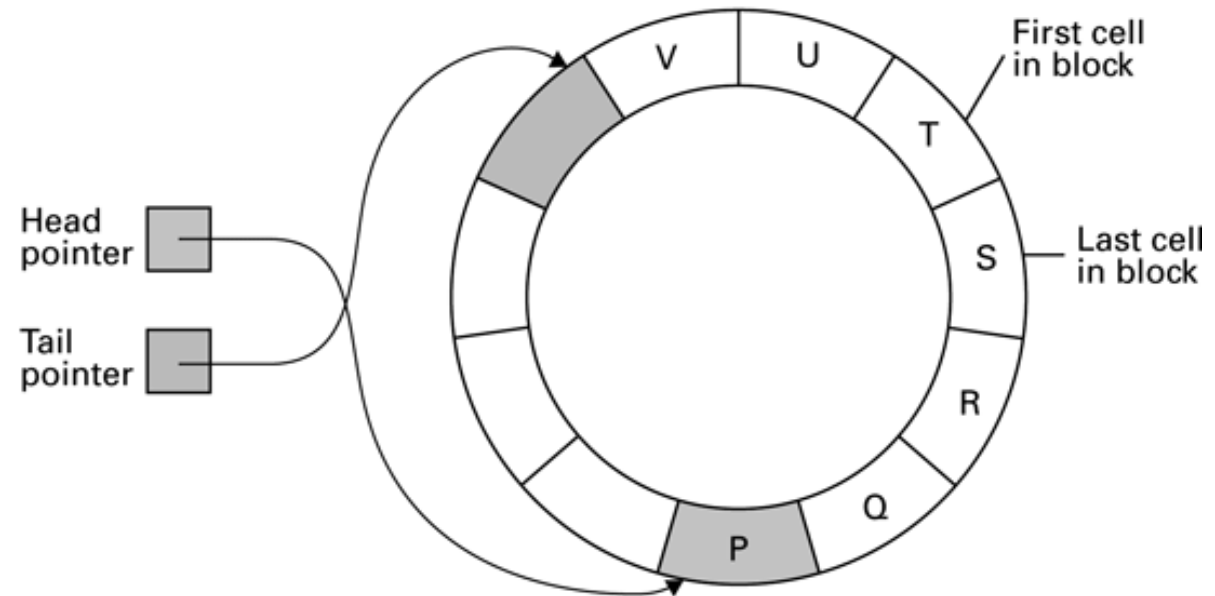
d. After removing B and inserting E

A queue's access rule is first in, first out. Trace which positions change.

A circular queue



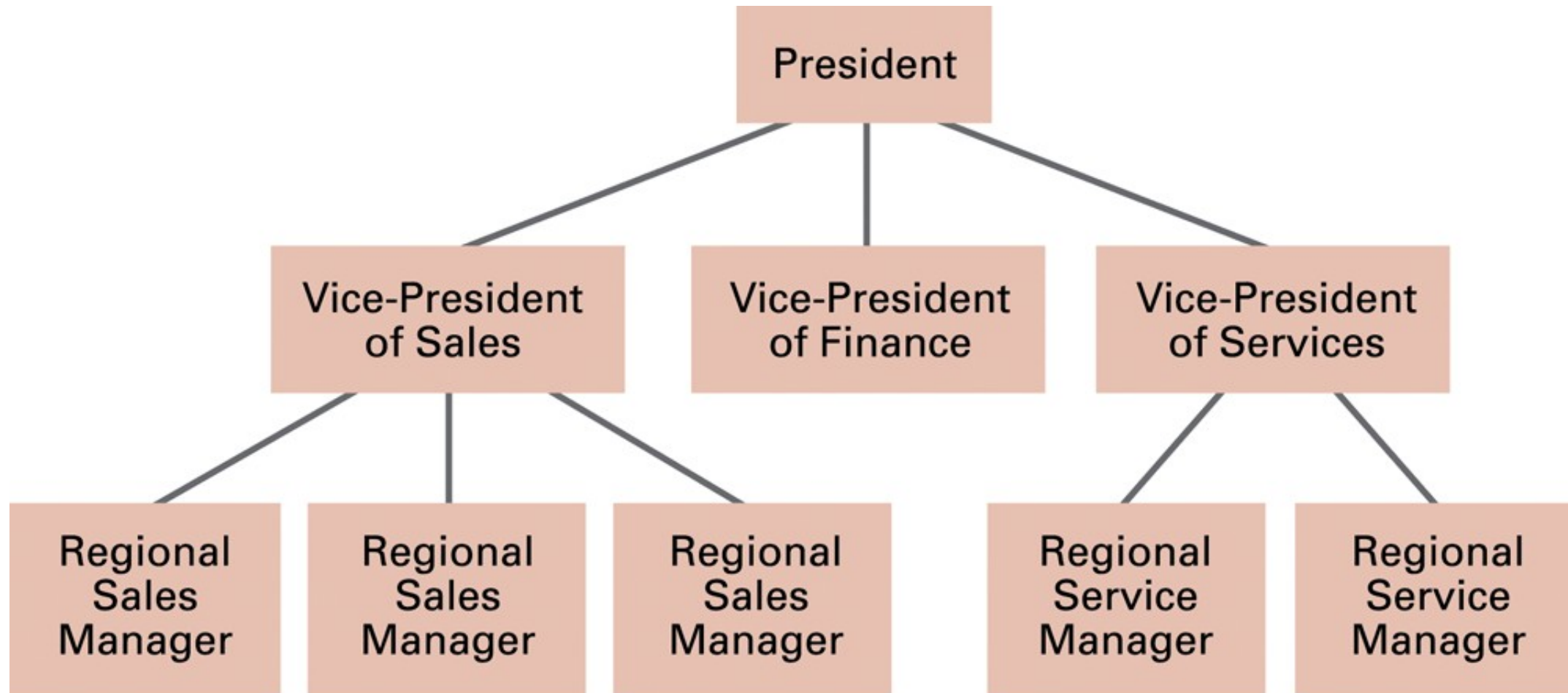
a. Queue as actually stored



b. Conceptual storage with last cell "adjacent" to first cell

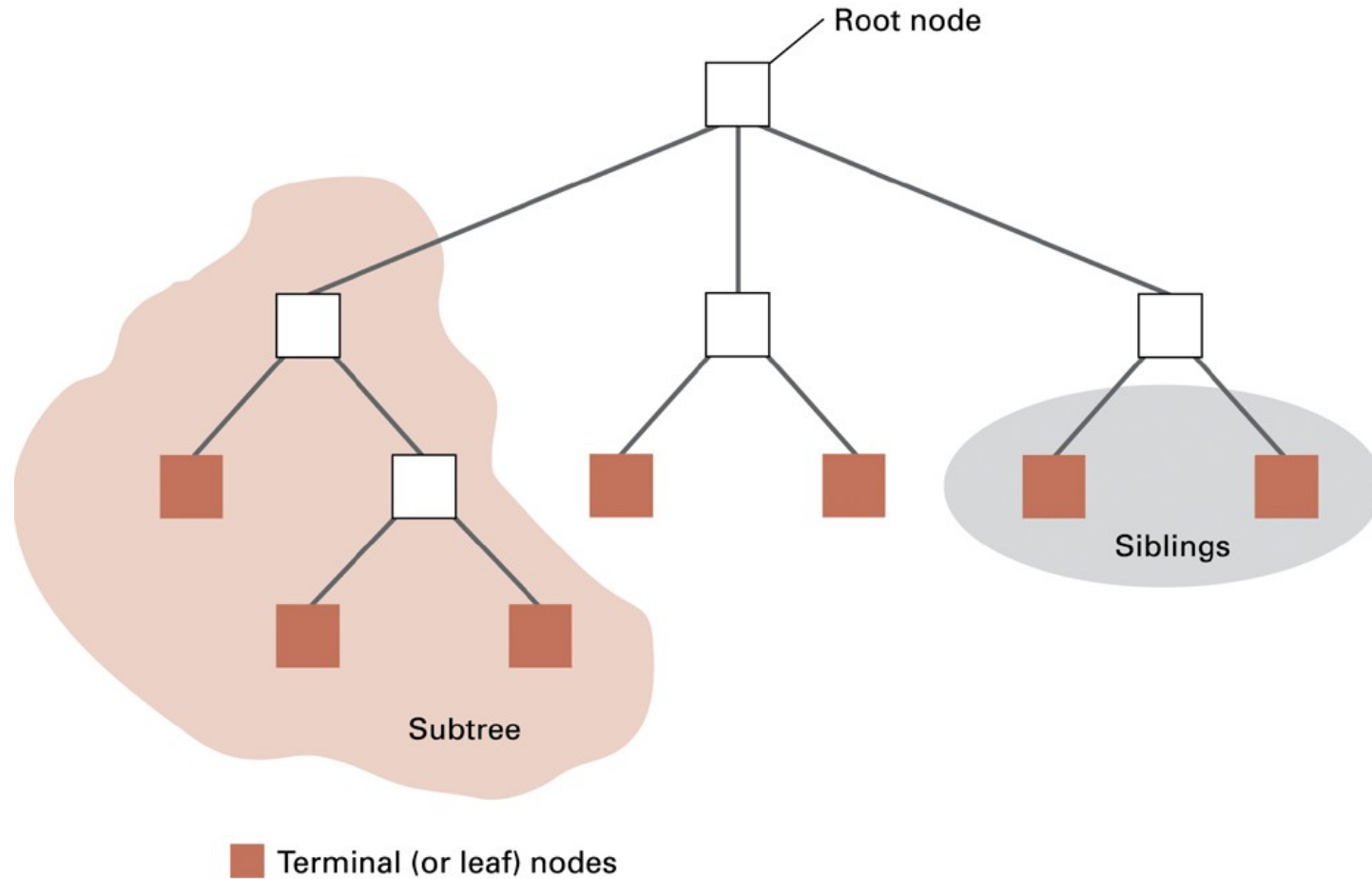
Wrapping indices reuses space in a bounded array.

Hierarchy in an organization



A tree represents relationships rather than only a linear order.

Tree terminology



Root, leaves, parent and child describe positions in the hierarchy.

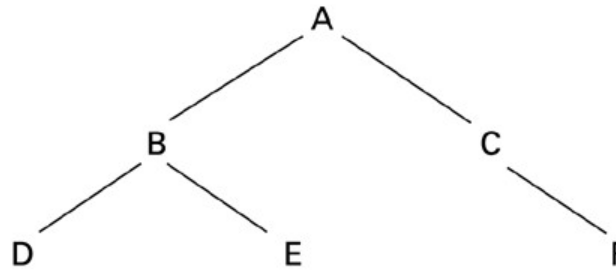
A binary-tree node



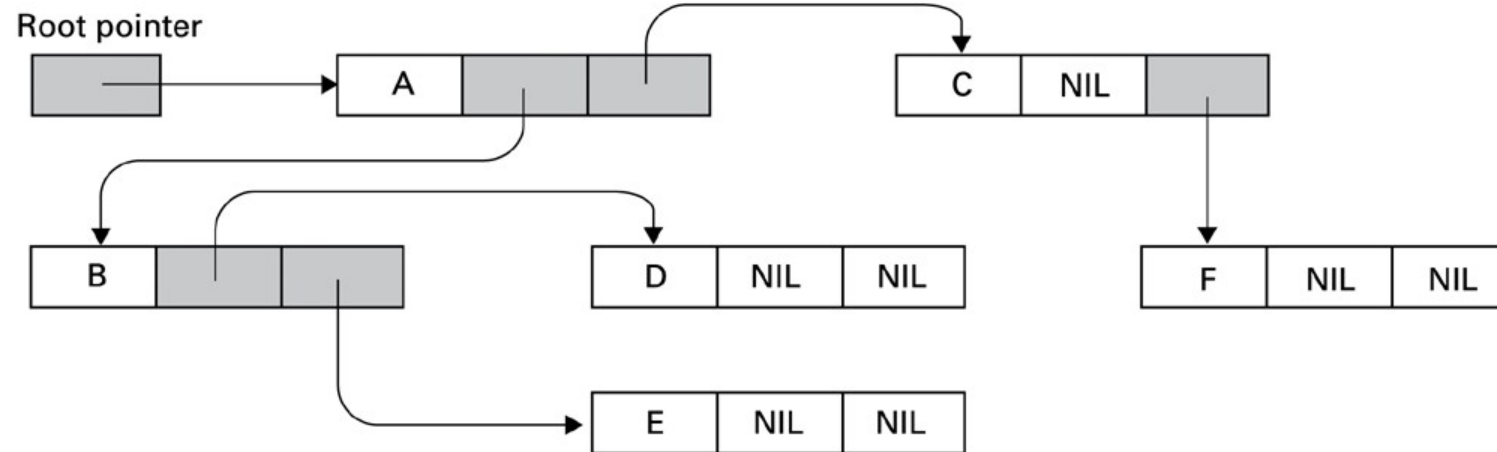
Each node stores data and up to two child links.

A tree uses more than one link

Conceptual tree



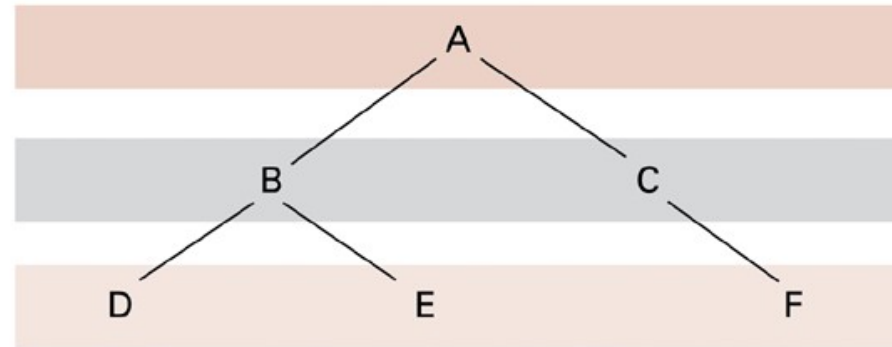
Actual storage organization



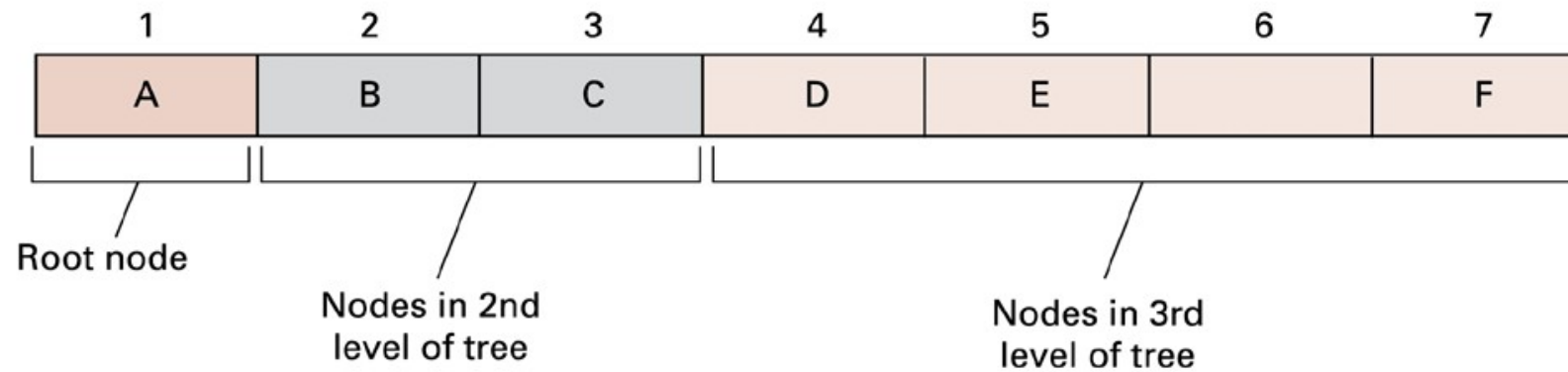
A binary-tree node can have left and right links. A linked list has one successor link.

A tree stored in an array

Conceptual tree



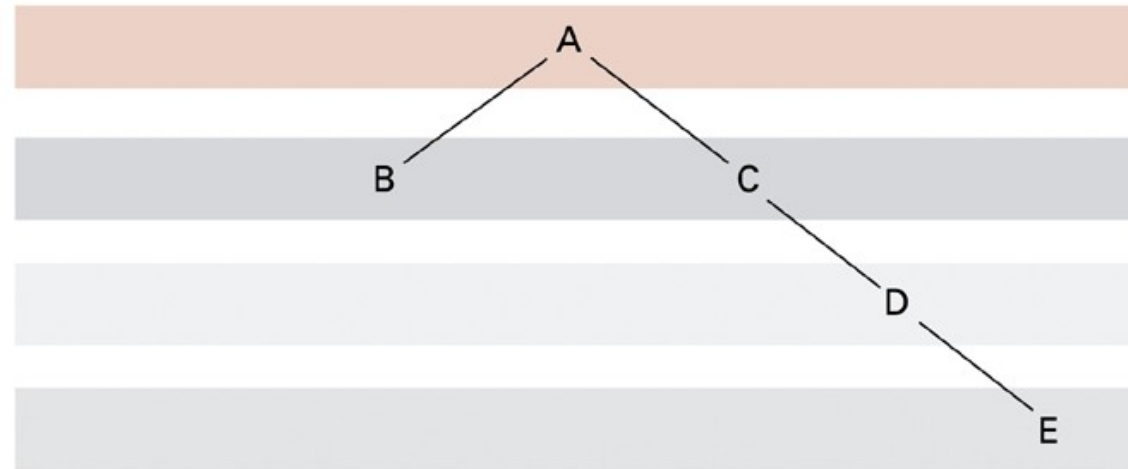
Actual storage organization



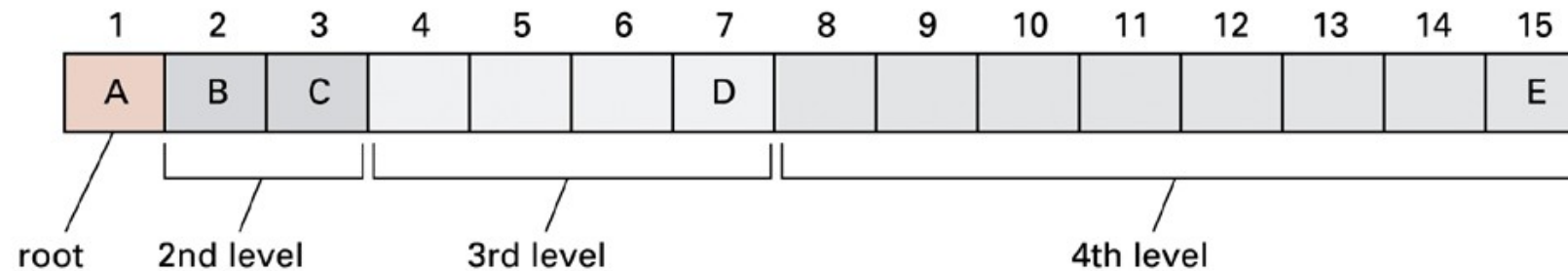
Index rules can encode child relationships without explicit pointers.

Sparse trees in array storage

Conceptual tree

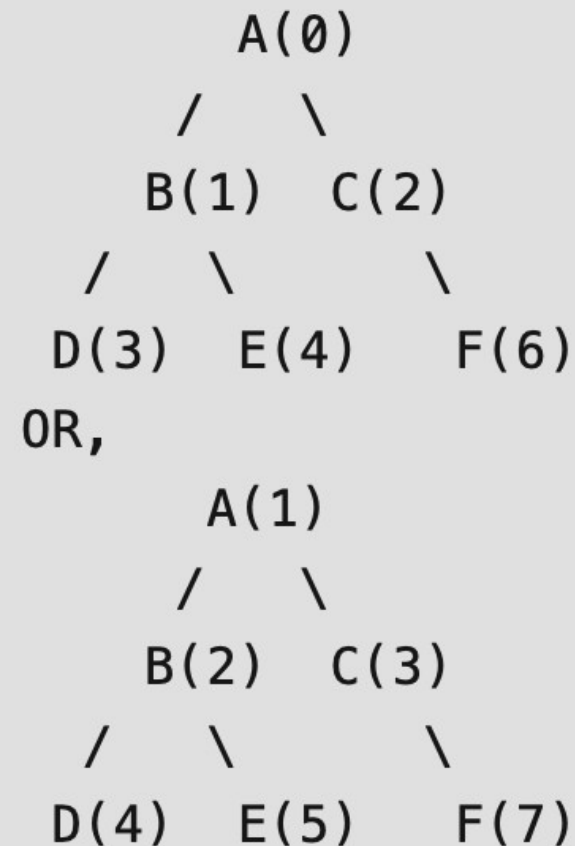


Actual storage organization



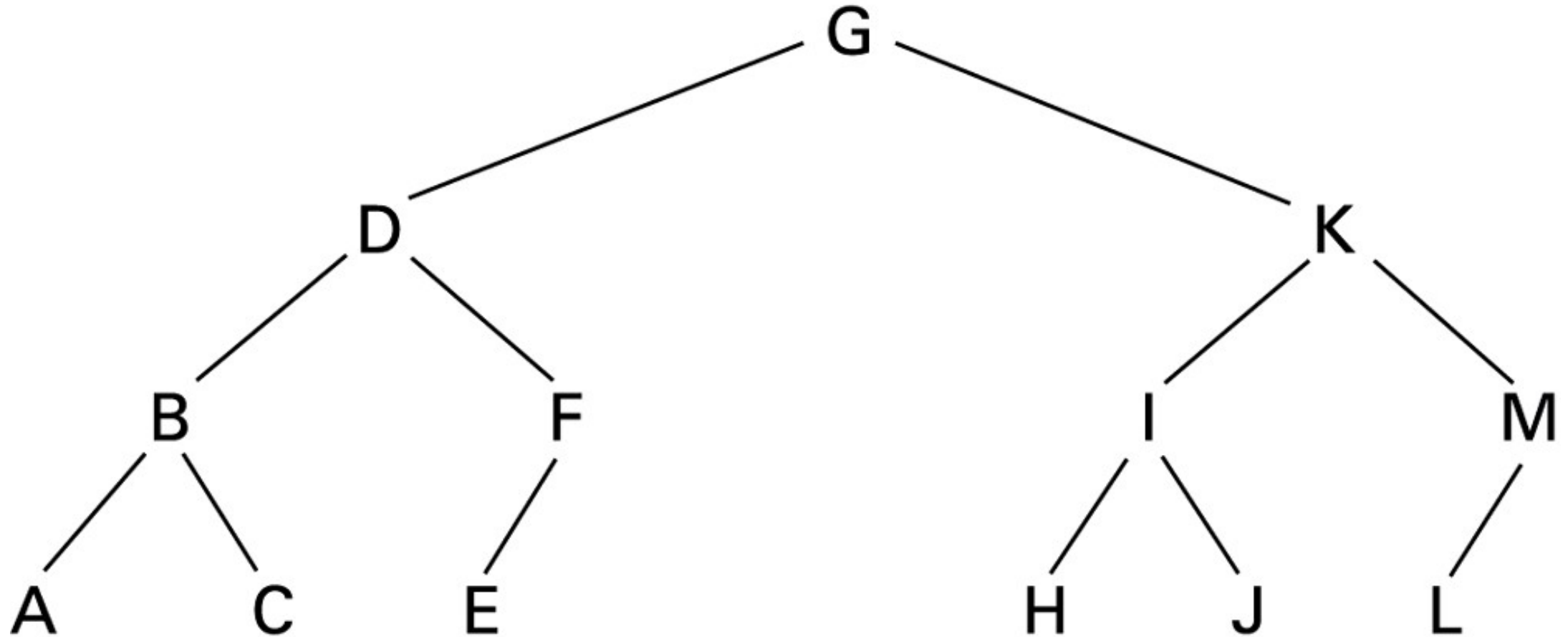
A sparse tree can leave many unused array positions.

Array indexing for a tree



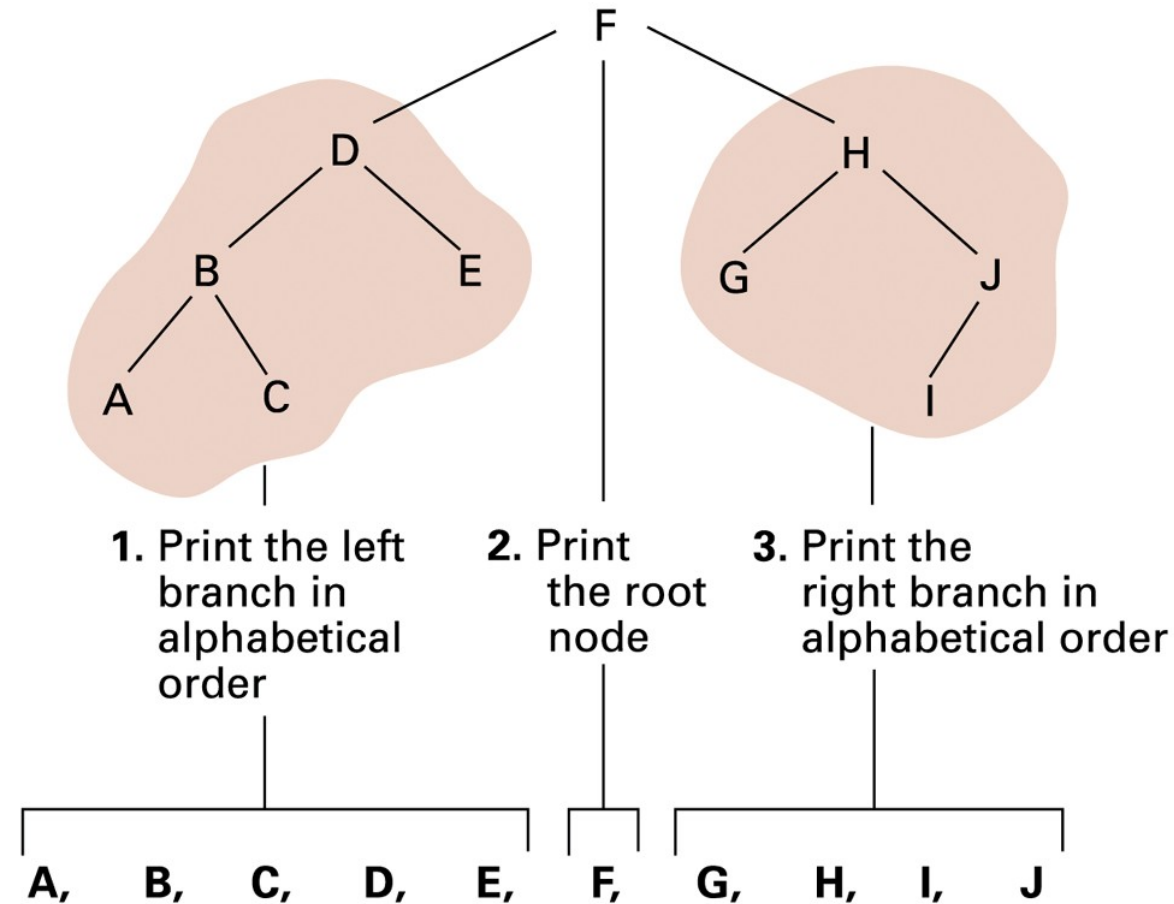
Compare the original index convention before applying a child-index formula.

Ordered-tree search: Week 8 recap



Week 8 explained the search advantage. Here the focus is the stored nodes and their left/right links.

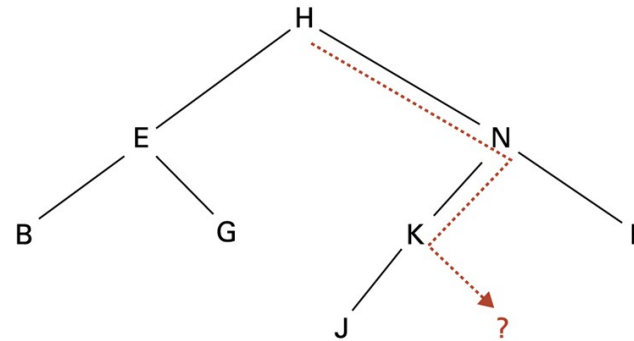
An in-order traversal



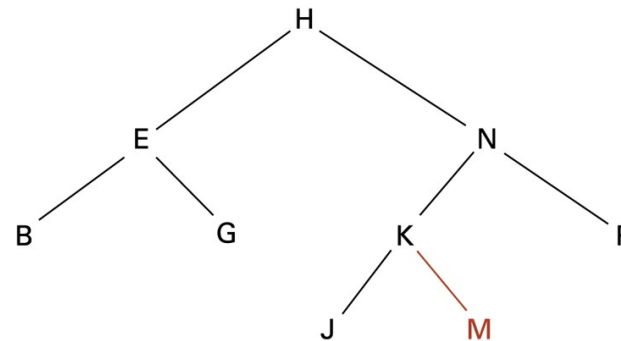
For a binary search tree, left subtree, node, right subtree gives sorted order.

Inserting M into an ordered tree

a. Search for the new entry until its absence is detected



b. This is the position in which the new entry should be attached



Follow comparisons to the location of the new leaf.

Recognizing related structures

Structure	Connections	Typical use
Singly linked list	One next link	Sequential records
Doubly linked list	Next and previous links	Traversal in both directions
Tree	Parent–child hierarchy	Ordered lookup or organization
Graph	General connections, possibly cycles	Roads, networks or search states

These structures organize access and relationships. Tree and graph implementation are outside this week's lab scope.

A node combines record fields with links. Traversal follows those links while the nodes remain valid.

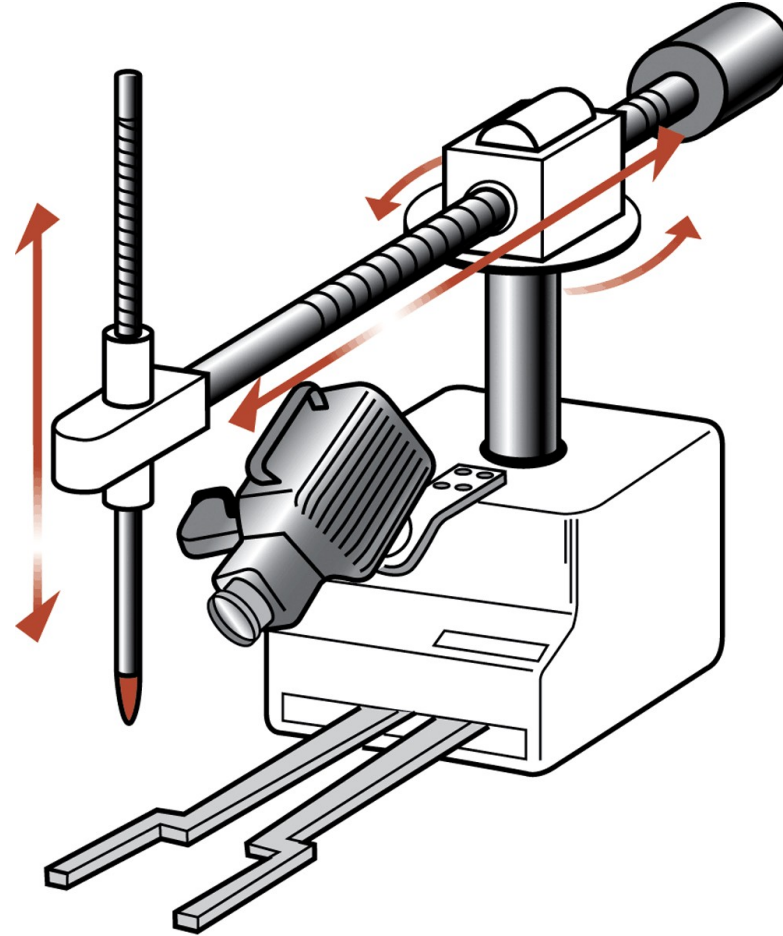
For the guided event-list exercise, explain each link change before writing the statement.

PC C is enough for the list exercise. Supplied allocation and cleanup support can keep the focus on structure.

Additional traces and structures

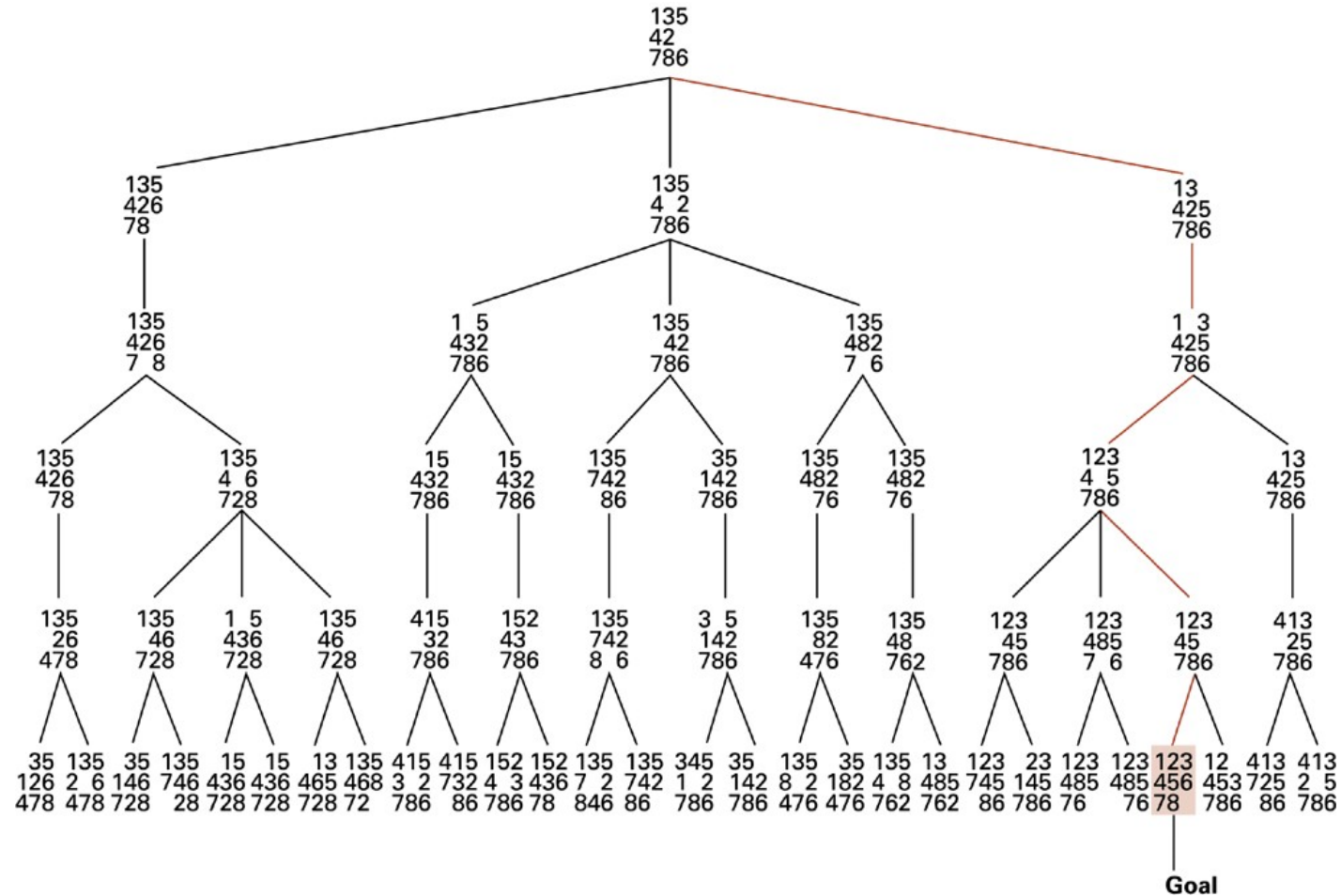
Search, neural-network updates and original data-structure examples

A machine for the puzzle



A historical conceptual machine links perception, planning and physical action.

A complete search-tree example



Different paths can revisit a state. A practical search procedure must handle repetitions.

A solution stored as actions

Top of stack —

- Move the 5 tile down.
- Move the 3 tile right.
- Move the 2 tile up.
- Move the 5 tile left.
- Move the 6 tile up.

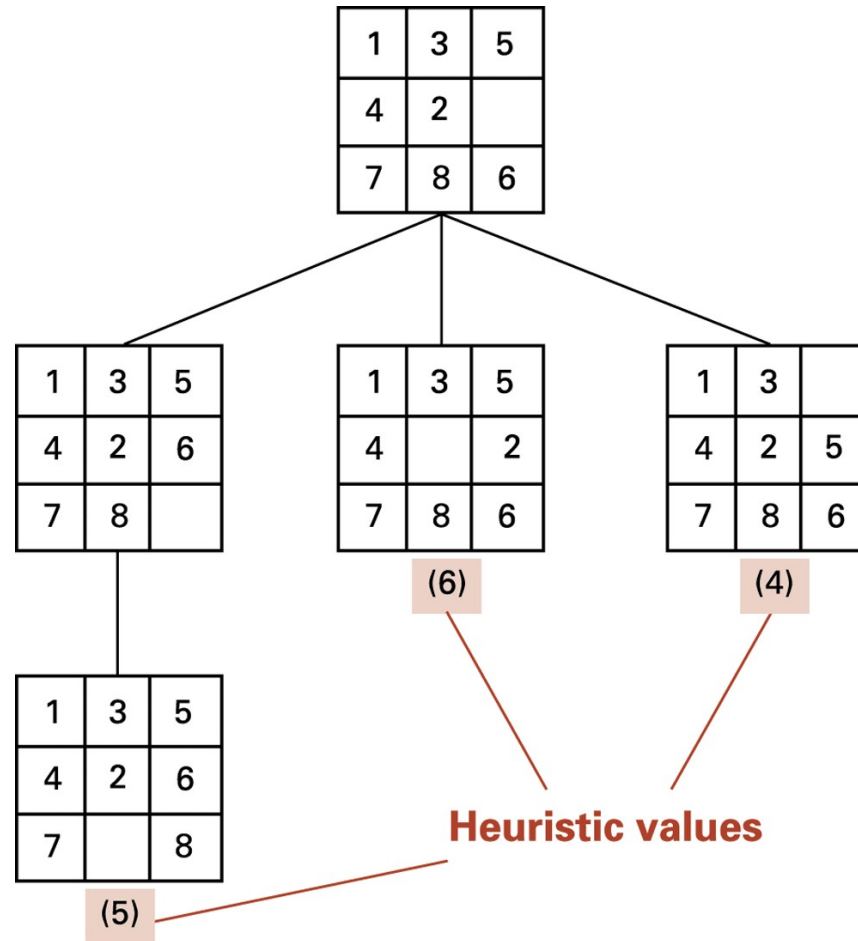
The stack records actions for later execution. Read the indicated top first.

The original heuristic search procedure

```
Establish the start node of the state graph as the root of the
search tree and record its heuristic value.
while (the goal node has not been reached) do
    [Select the leftmost leaf node with the smallest heuristic
     value of all leaf nodes.
     To this selected node attach as children those nodes that
     can be reached by a single production.
     Record the heuristic of each of these new nodes next
     to the node in the search tree
    ]
Traverse the search tree from the goal node up to the root,
pushing the production associated with each arc traversed
onto a stack.
Solve the original problem by executing the productions as they
are popped off the stack.
```

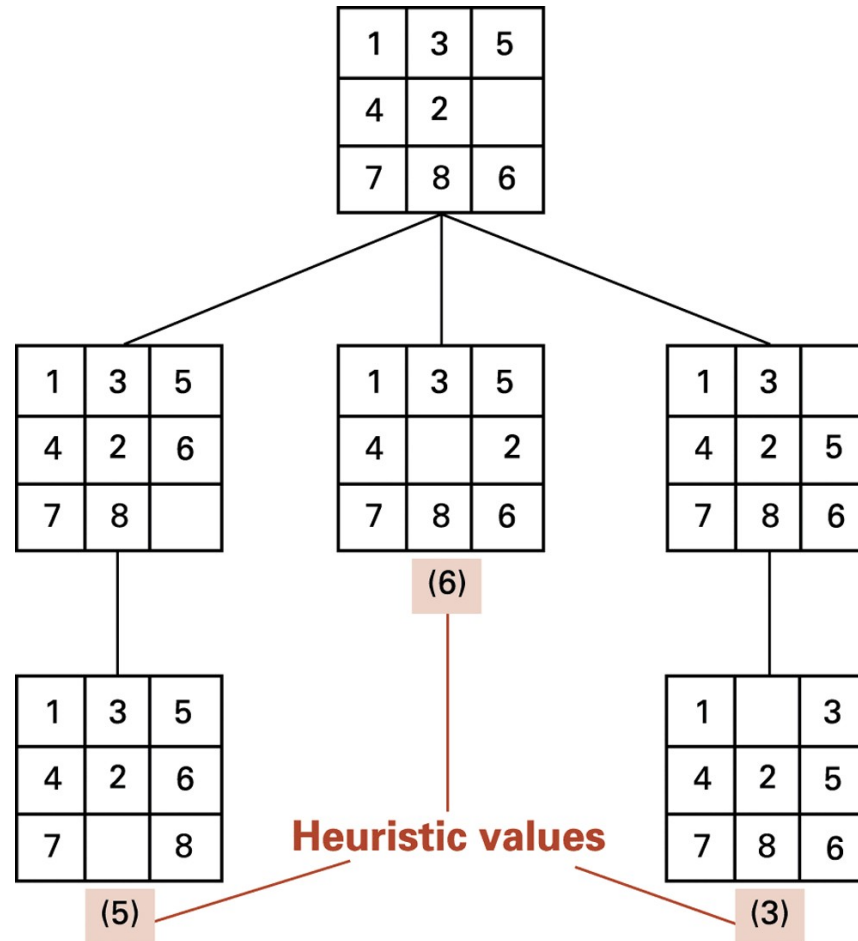
This procedure chooses by h alone: greedy best-first search. It does not use the A^* score $g + h$.

Heuristic search: two passes



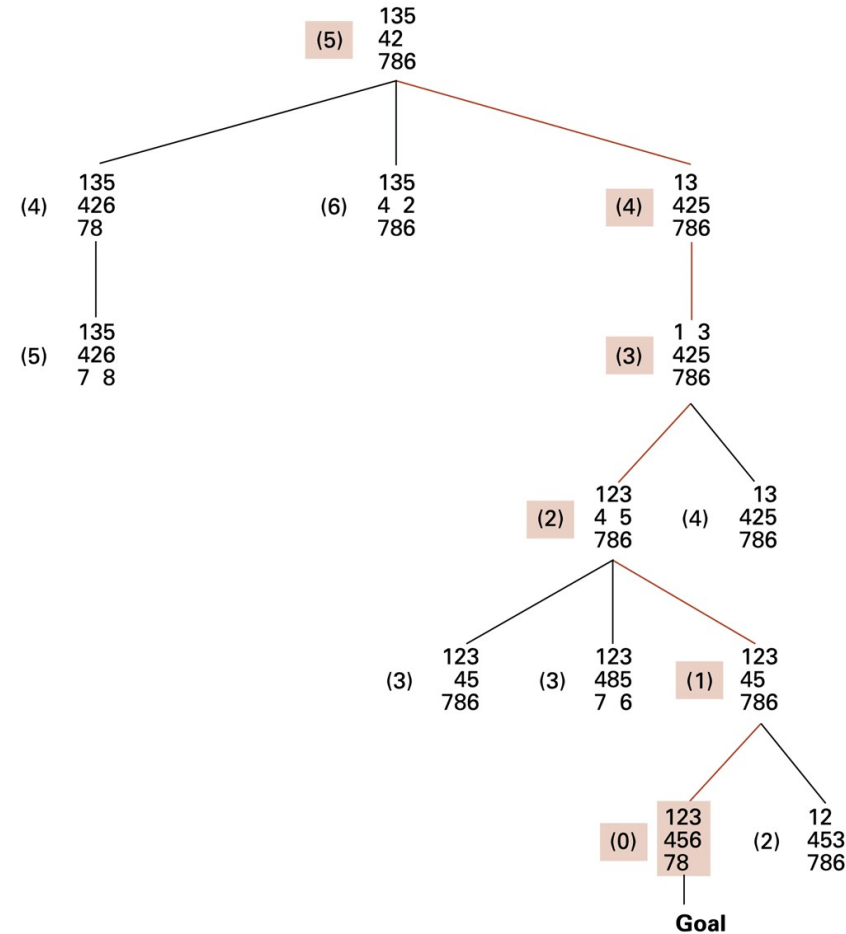
The original intermediate tree retains the alternative candidates.

Heuristic search: three passes



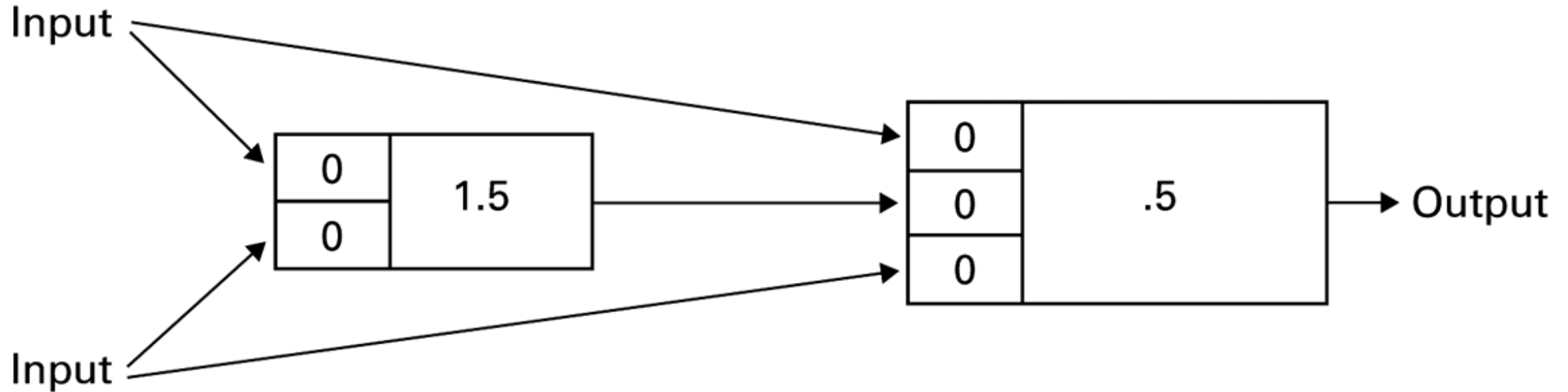
Compare the newly generated candidates with those still waiting.

Heuristic search: the complete example



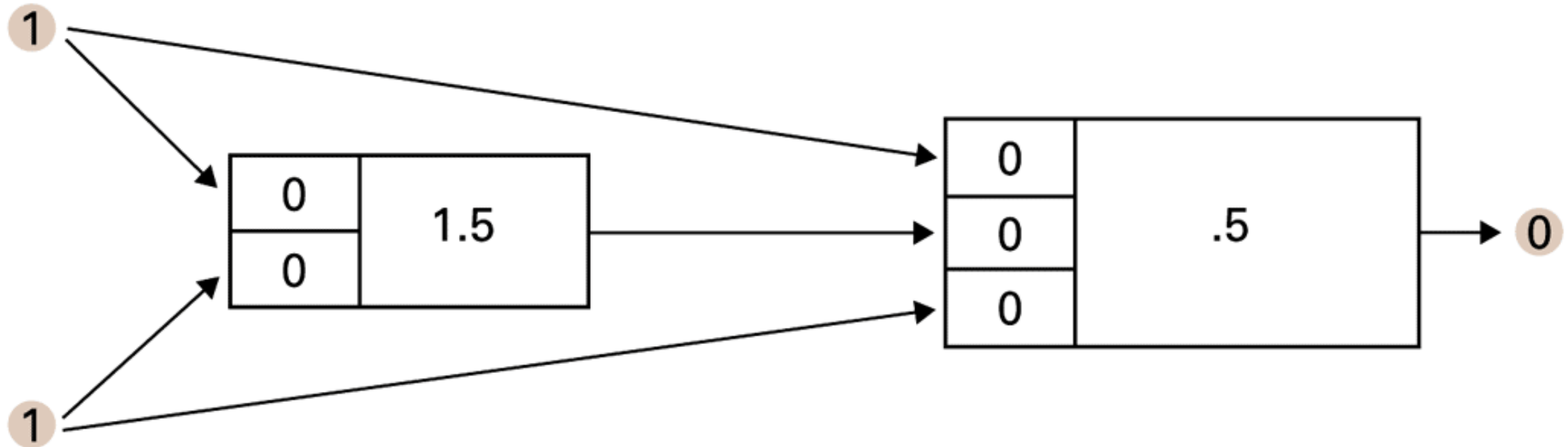
The figure shows this example's exploration, not a guarantee of optimality for every problem.

A network before weight adjustment



The architecture is fixed in this example. The numerical weights can change.

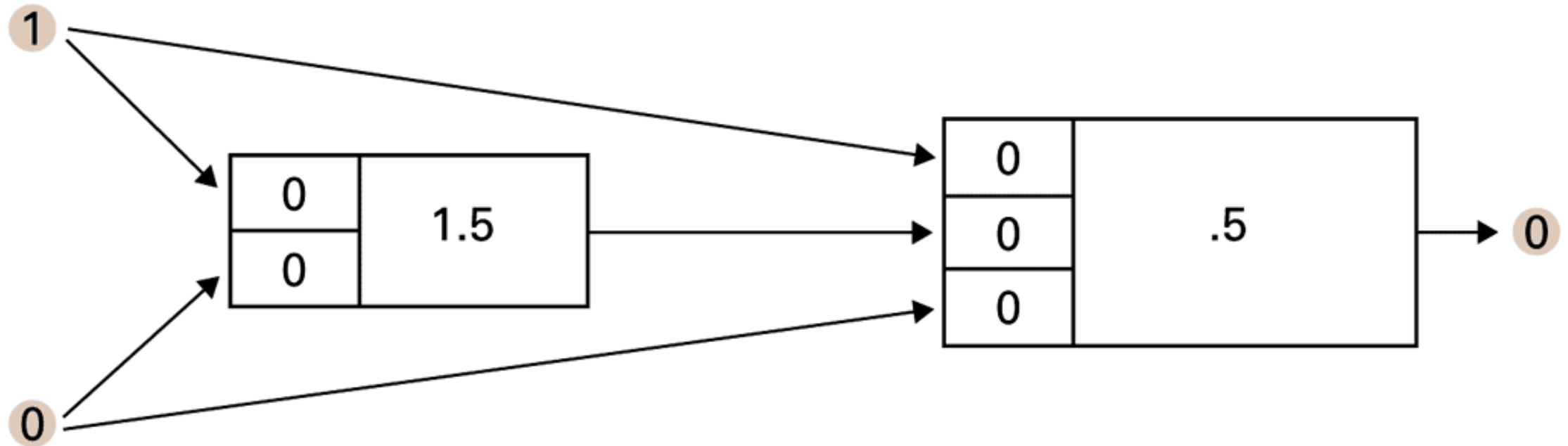
Training trace: a correct output



a. The network performs correctly for the input pattern 1, 1.

For XOR, equal binary inputs should produce 0. This example currently does so.

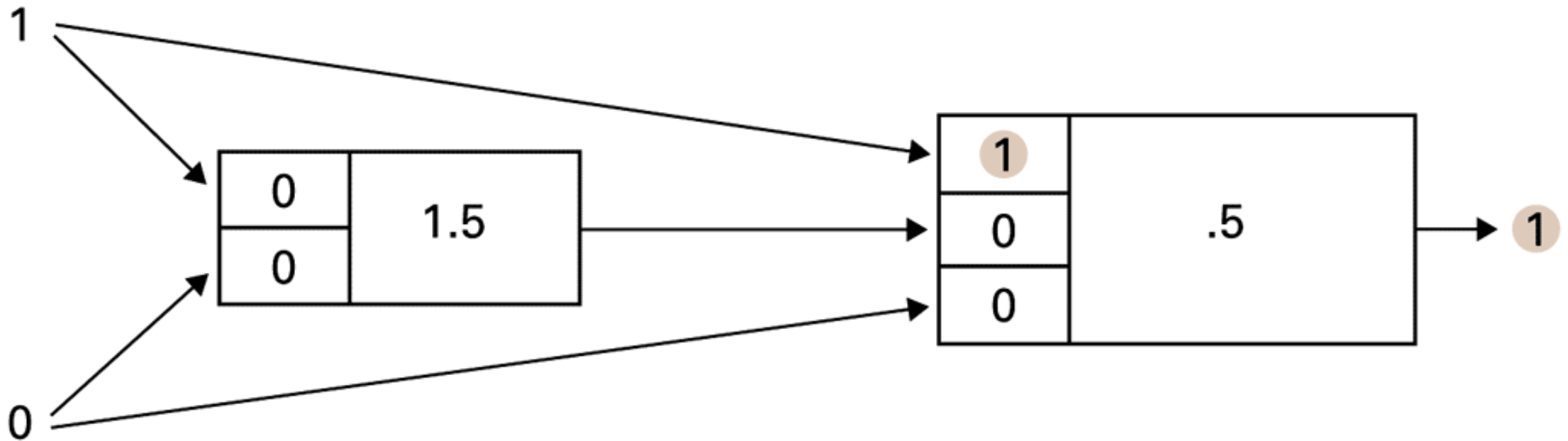
Training trace: an incorrect output



b. The network performs incorrectly for the input pattern 1, 0.

For XOR, unequal inputs should produce 1. The current network produces 0.

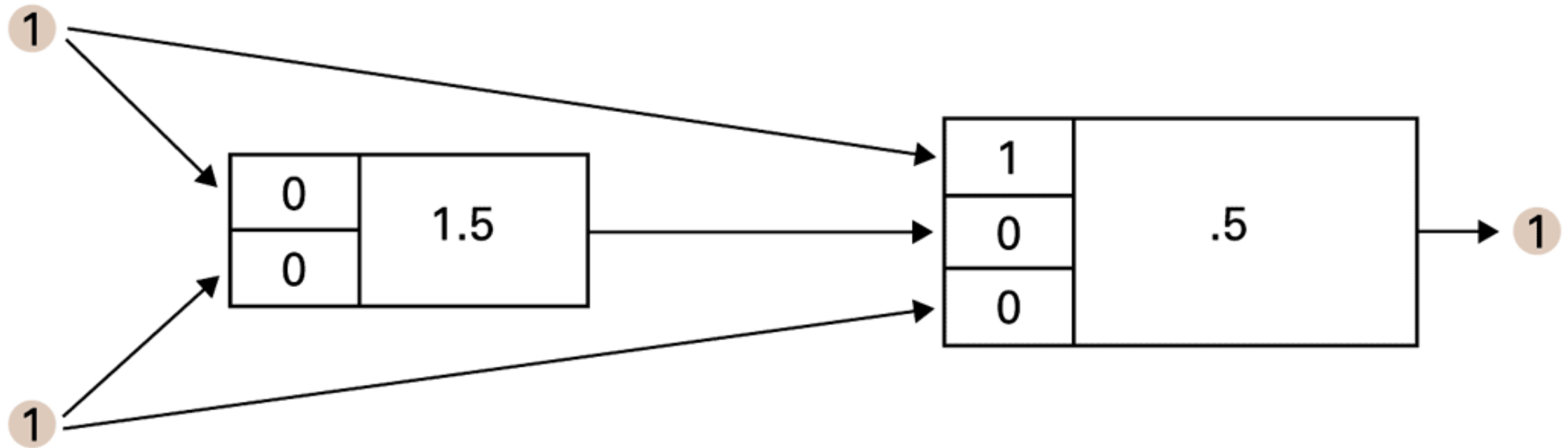
Training trace: a weight change



c. The upper weight in the second processing unit is adjusted.

The changed weight corrects this one example.

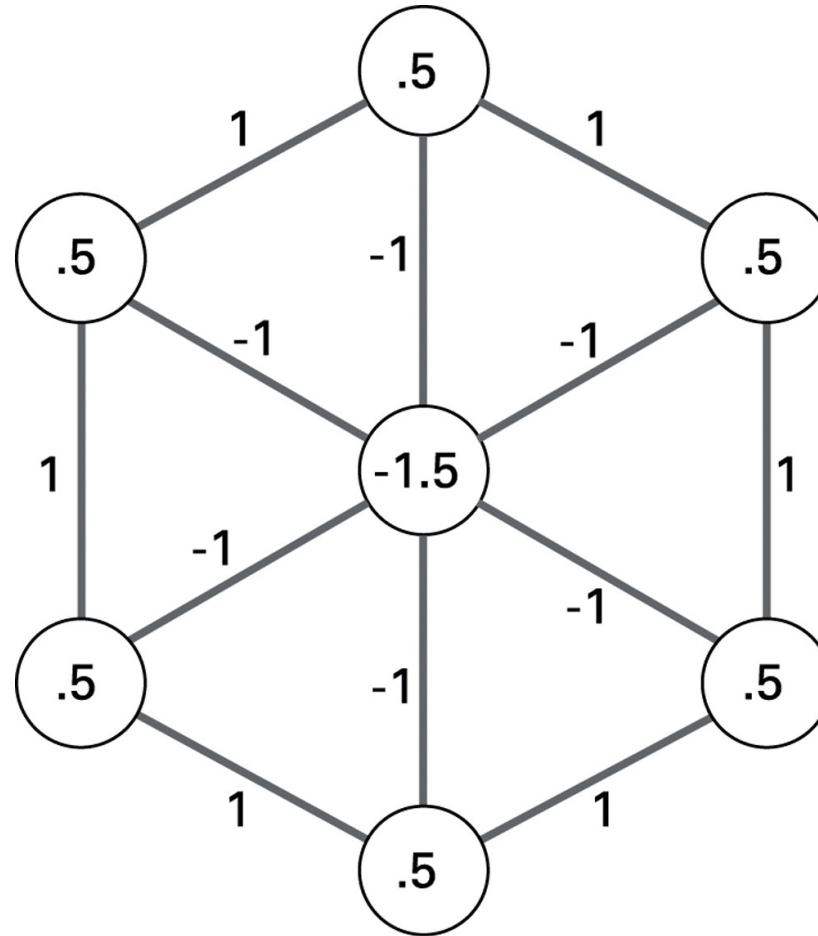
Training trace: another example fails



d. However, the network no longer performs correctly for the input pattern 1, 1.

A change that helps one example can harm another. Training must account for multiple examples.

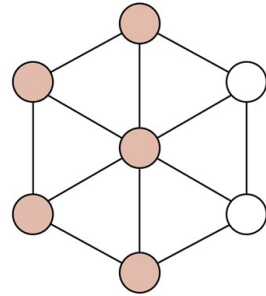
Associative memory



A network can reconstruct a stored pattern from a related input. This is a historical architecture.

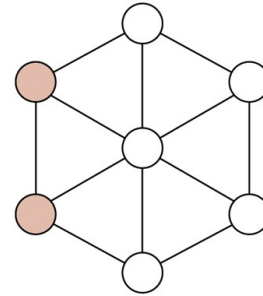
Settling toward a stable configuration

a.



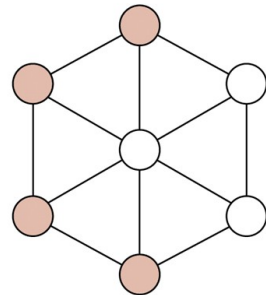
Start: All but the rightmost units are excited

b.



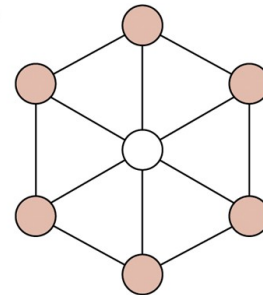
Step1: Only the leftmost units remain excited

c.



Step 2: The top and bottom units become excited

d.



Final: All the units on the perimeter are excited

Follow the successive states. Associative memory here differs from next-token generation.

procedure PrintList (List)

CurrentPointer $\leftarrow$ head pointer of List.

while (CurrentPointer is not NIL) **do**

(Print the name in the entry pointed to by CurrentPointer;
Observe the value in the pointer cell of the List entry
pointed to by CurrentPointer, and reassign CurrentPointer
to be that value.)

The source uses pseudocode. The C version checks the pointer before accessing the node.

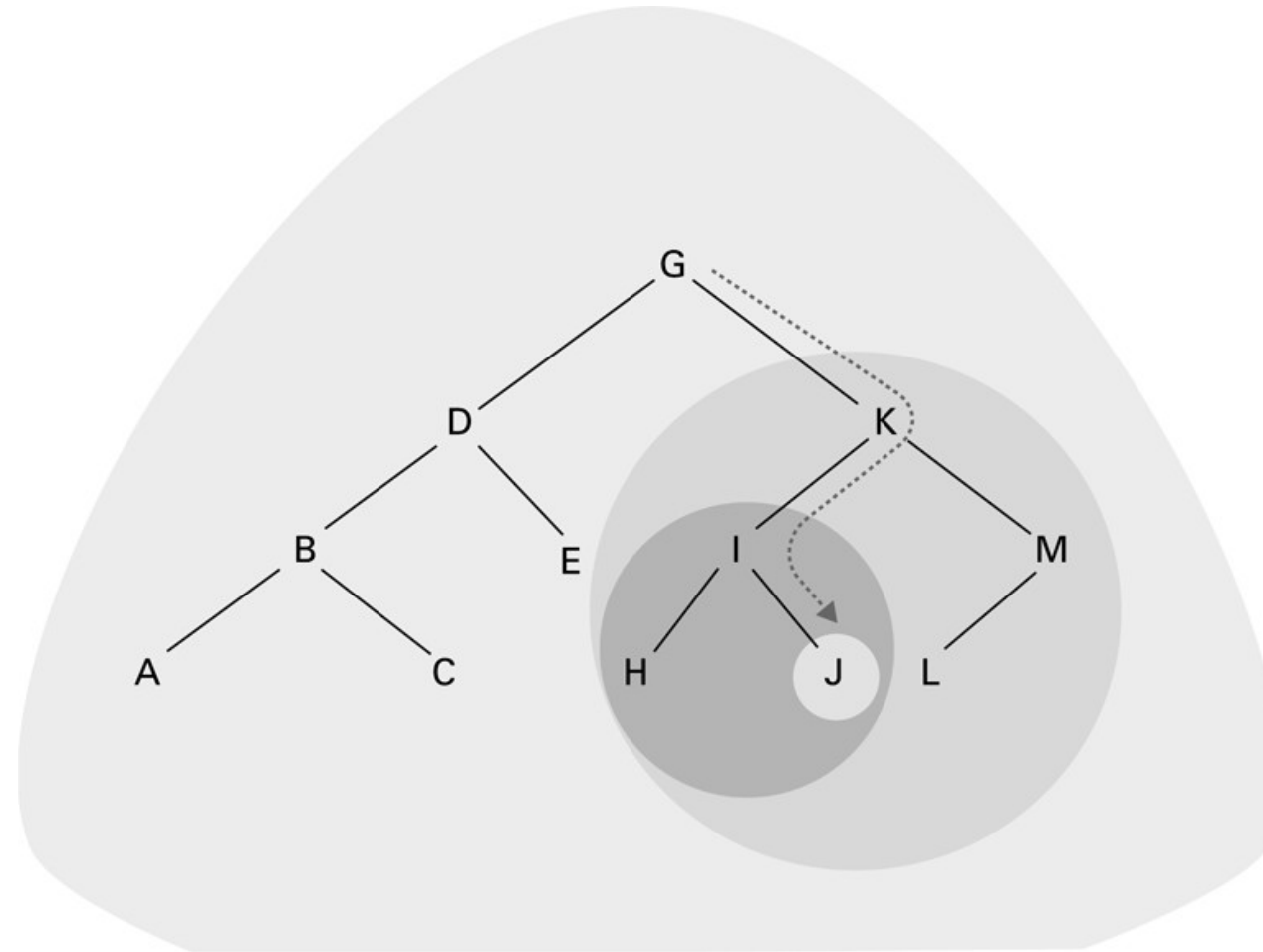
Search in an ordered tree

```
procedure Search(Tree, TargetValue)

if (root pointer of Tree = NIL)
  then
    (declare the search a failure)
  else
    (execute the block of instructions below that is
     associated with the appropriate case)
    case 1: TargetValue = value of root node
      (Report that the search succeeded)
    case 2: TargetValue < value of root node
      (Apply the procedure Search to see if
       TargetValue is in the subtree identified
       by the root's left child pointer and
       report the result of that search)
    case 3: TargetValue > value of root node
      (Apply the procedure Search to see if
       TargetValue is in the subtree identified
       by the root's right child pointer and
       report the result of that search)
  ) end if
```

Each comparison chooses a subtree. This is a reference algorithm.

The search path for J



Trace the progressively smaller subtrees in the original example.

procedure PrintTree (Tree)

if (Tree is not empty)

then (Apply the procedure PrintTree to the tree that
appears as the left branch in Tree;

Print the root node of Tree;

Apply the procedure PrintTree to the tree that
appears as the right branch in Tree)

Each call handles one subtree. The empty subtree supplies the stopping case.

Original tree insertion procedure

```
procedure Insert(Tree, NewValue)

if (root pointer of Tree = NIL)
    (set the root pointer to point to a new leaf
     containing NewValue)
else (execute the block of instructions below that is
       associated with the appropriate case)
    case 1: NewValue = value of root node
        (Do nothing)
    case 2: NewValue < value of root node
        (if (left child pointer of root node = NIL)
            then (set that pointer to point to a new
                 leaf node containing NewValue)
            else (apply the procedure Insert to insert
                 NewValue into the subtree identified
                 by the left child pointer)
        case 3: NewValue > value of root node
            (if (right child pointer of root node = NIL)
                then (set that pointer to point to a new
                     leaf node containing NewValue)
                else (apply the procedure Insert to insert
                     NewValue into the subtree identified
                     by the right child pointer)
            ) end if
    ) end if
```

Advanced reference. Tree implementation is not required for this week's list exercise.

A stack class in other languages

```
class StackOfIntegers
{private int[] StackEntries = new int[20];
  private int StackPointer = 0;

  public void push(int NewEntry)
  {if (StackPointer < 20)
    StackEntries[StackPointer++] = NewEntry;
  }

  public int pop()
  {if (StackPointer > 0) return StackEntries[--StackPointer];
   else return 0;
  }
}
```

Original Java/C# illustration for comparison. The C lecture does not require class syntax.

Brookshear & Brylow, Computer Science: An Overview, 12th Global edition: Chapter 11, Artificial Intelligence.

Original illustrations: supplied Brookshear 10th-edition AI slides and instructor Set 8.

Google Machine Learning Crash Course: neural networks, overfitting and language models. Diagrams: CC BY 4.0.

SRI: Shakey. IBM: Deep Blue. Google DeepMind: AlphaGo. Primary histories and photographs.

Vaswani et al. (2017), Attention Is All You Need. Brown et al. (2020), Language Models are Few-Shot Learners.

Ho et al. (2020), Denoising Diffusion Probabilistic Models. 3Blue1Brown: Large Language Models Explained Briefly.