Recursive Exploration

Chris Piech

CS 106B
Lecture 8
Jan 25, 2016

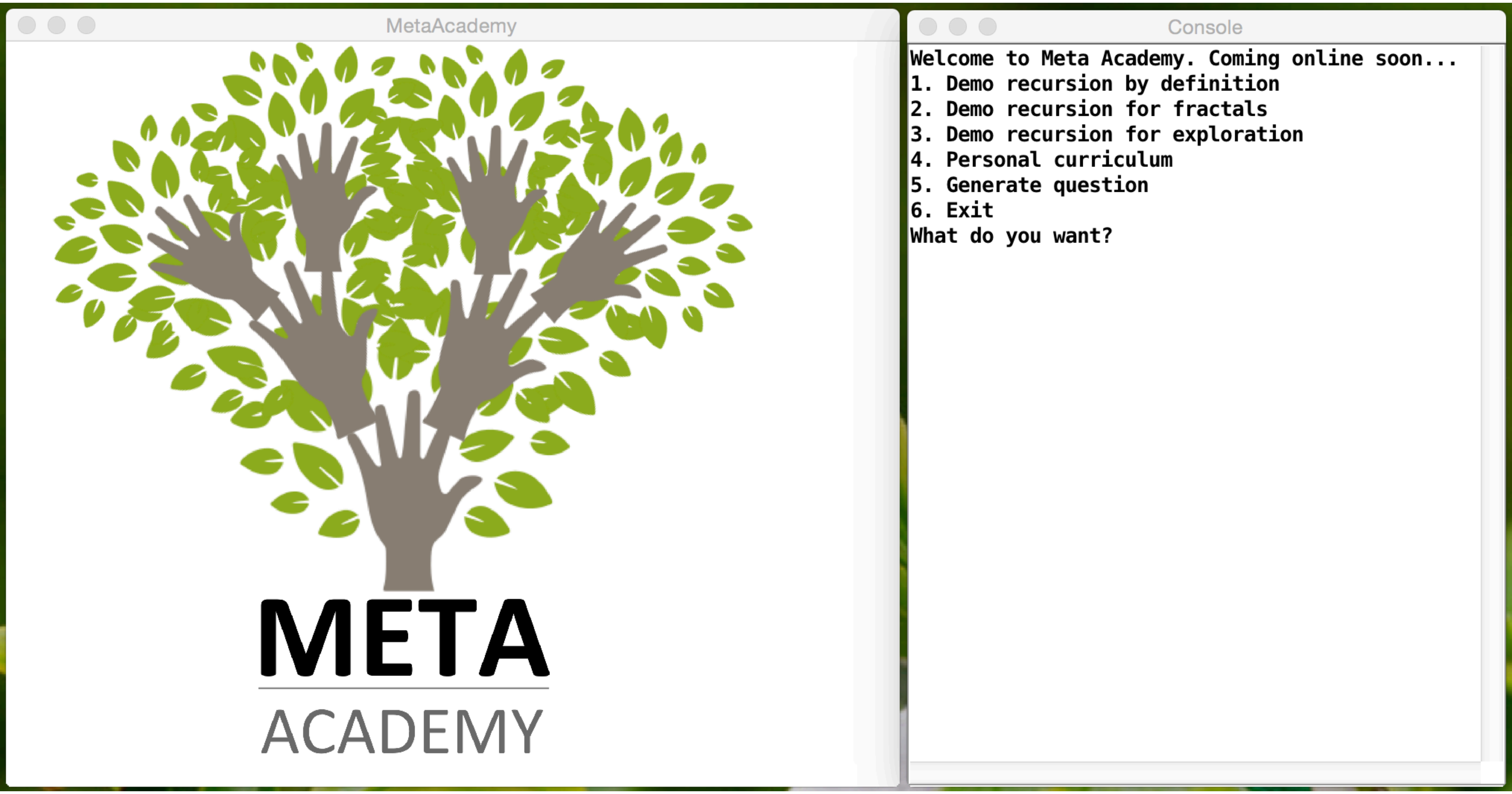
Assignment 3

META

ACADEMY

Due: Feb 3rd

Assignment 3

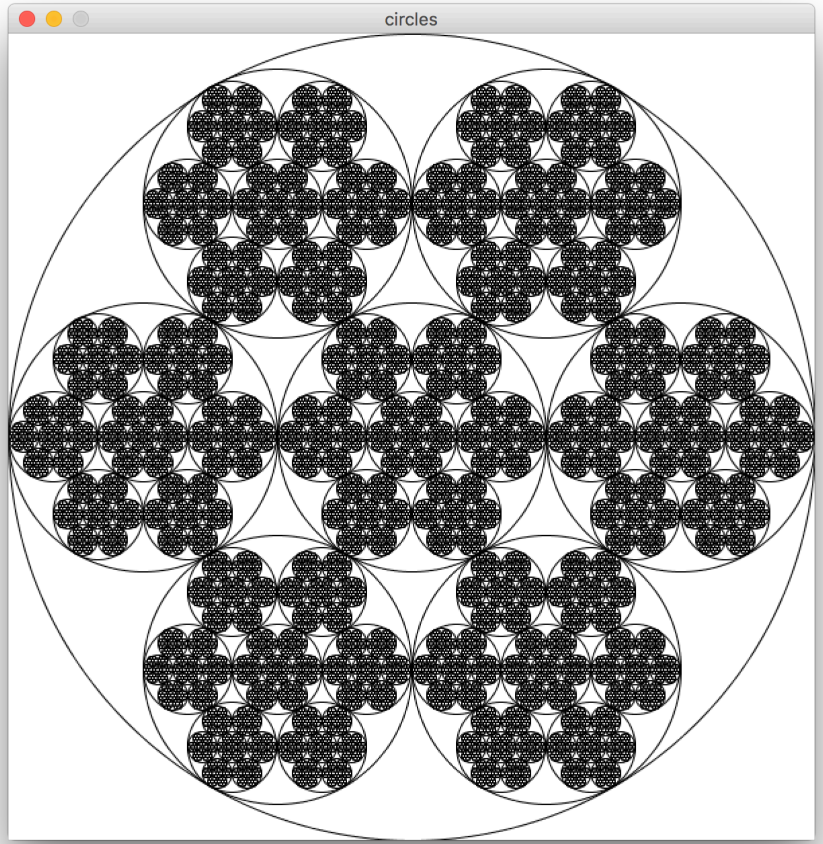


```
Welcome to Meta Academy. Coming online soon...
1. Demo recursion by definition
2. Demo recursion for fractals
3. Demo recursion for exploration
4. Personal curriculum
5. Generate question
6. Exit
What do you want?
```

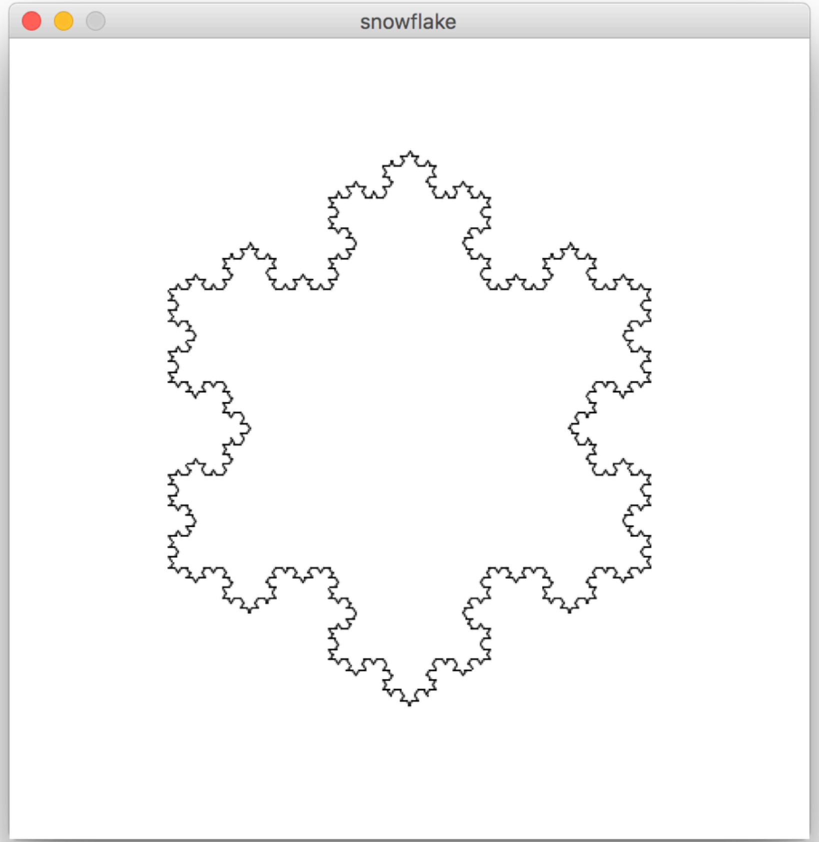
In order to learn recursion, you will teach recursion.
In order to teach recursion, you must first learn recursion.

Extra Fractal Examples

circles

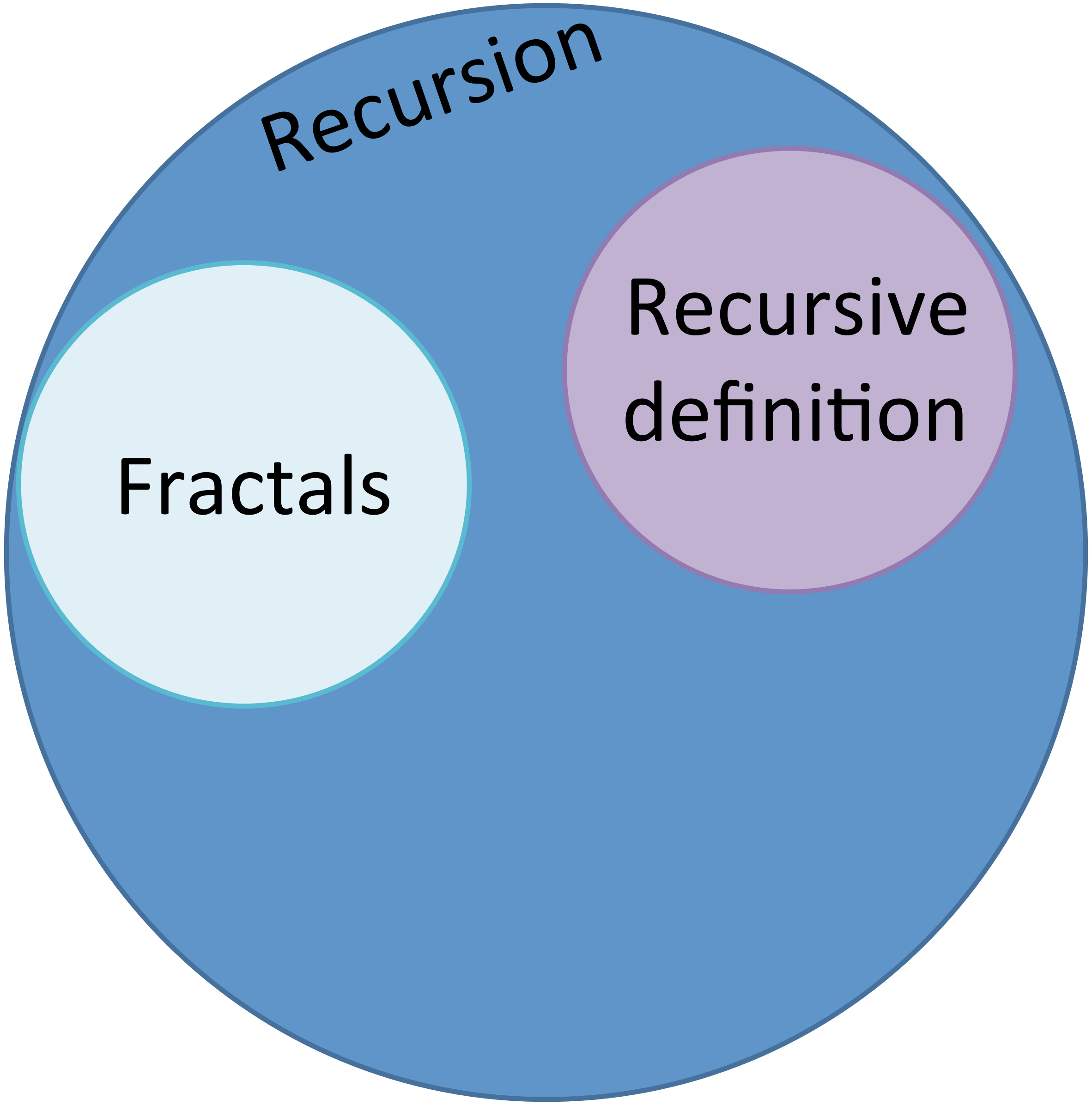


koch

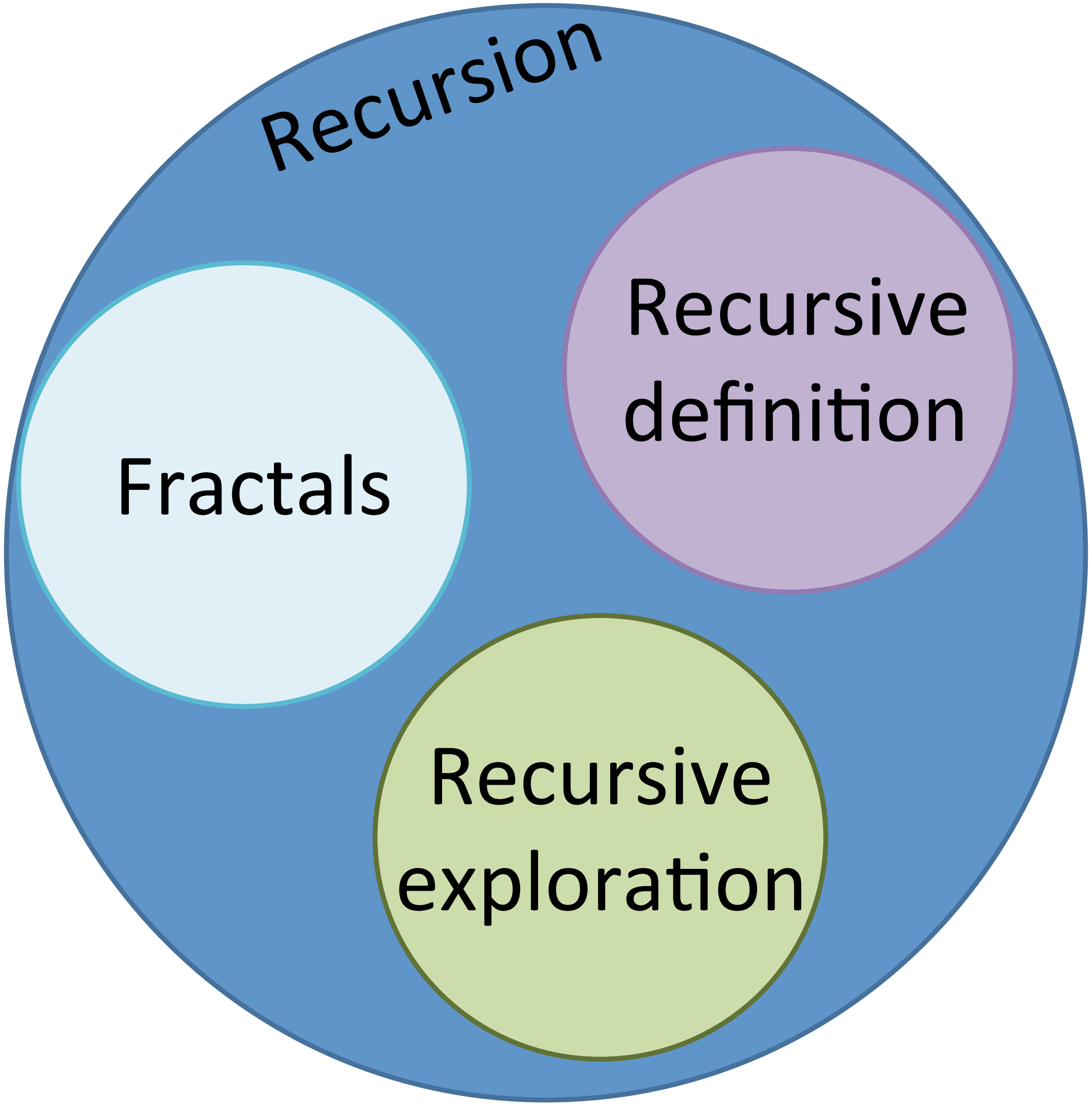


Recursive Ray Tracing

Flavors of Recursion



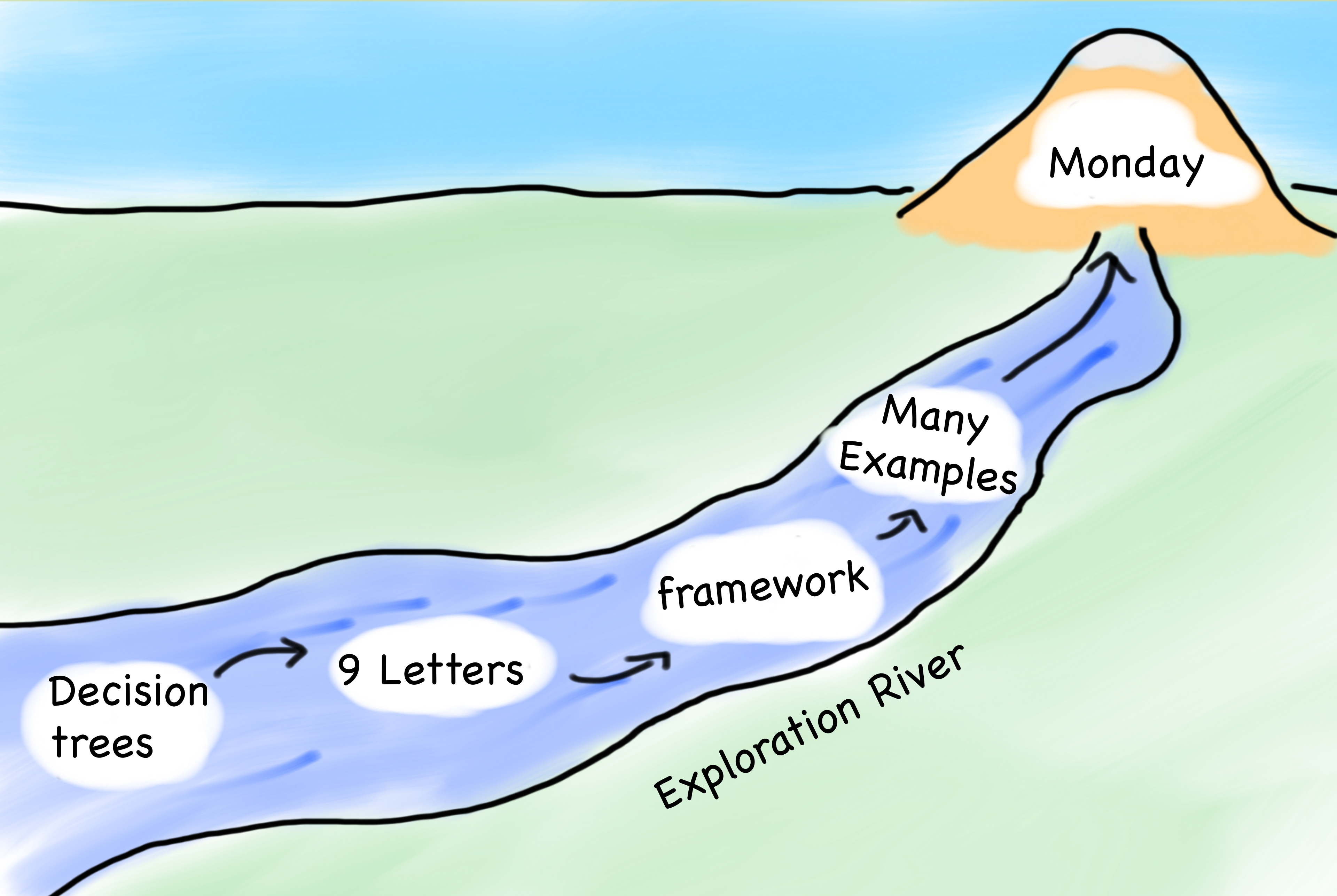
Flavors of Recursion



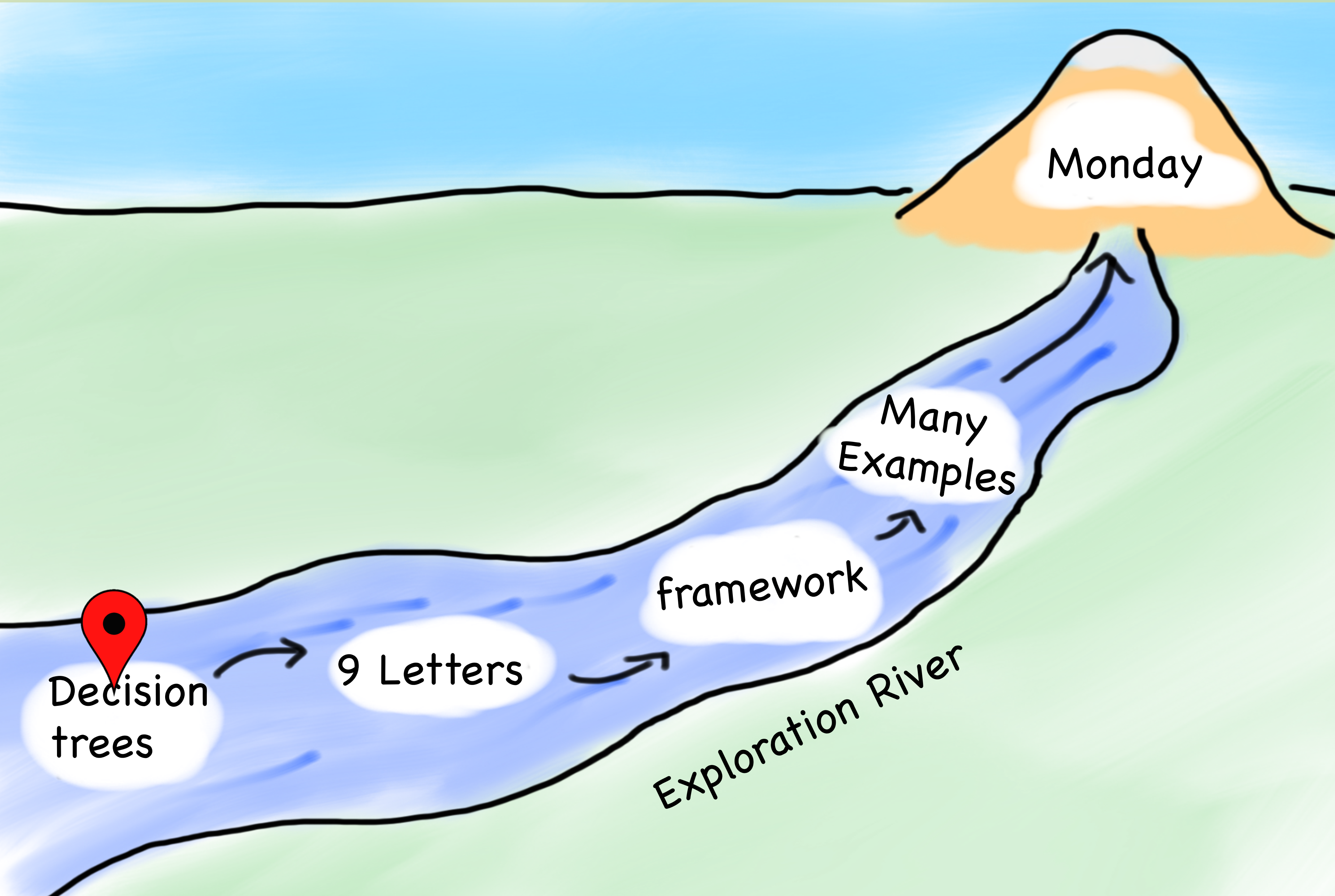
Today's Goal

1. Introduction to decision trees.

Today's Route



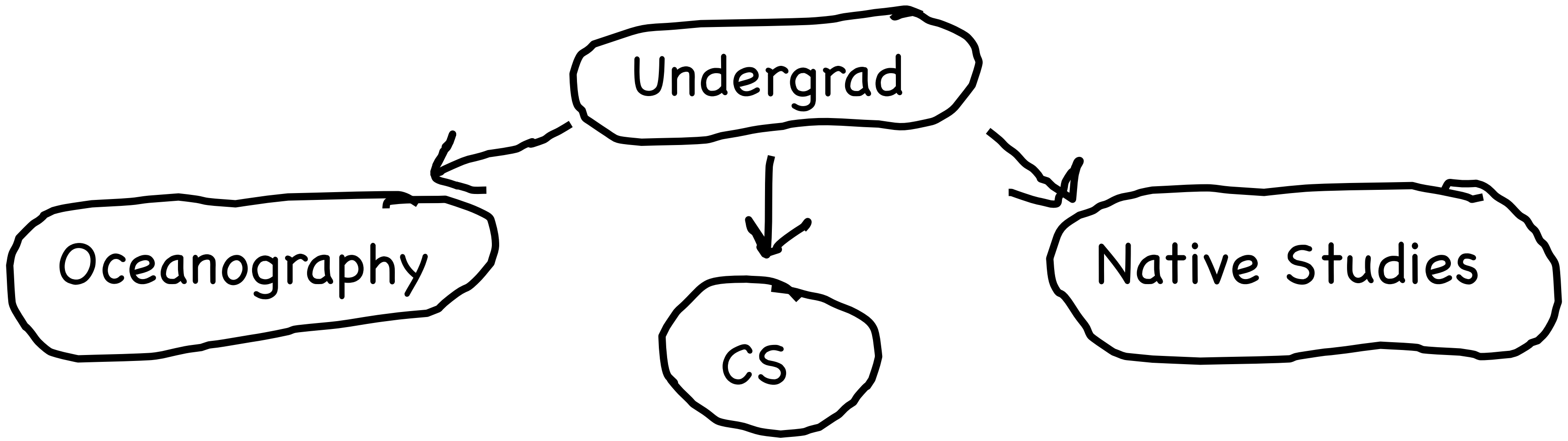
Today's Route



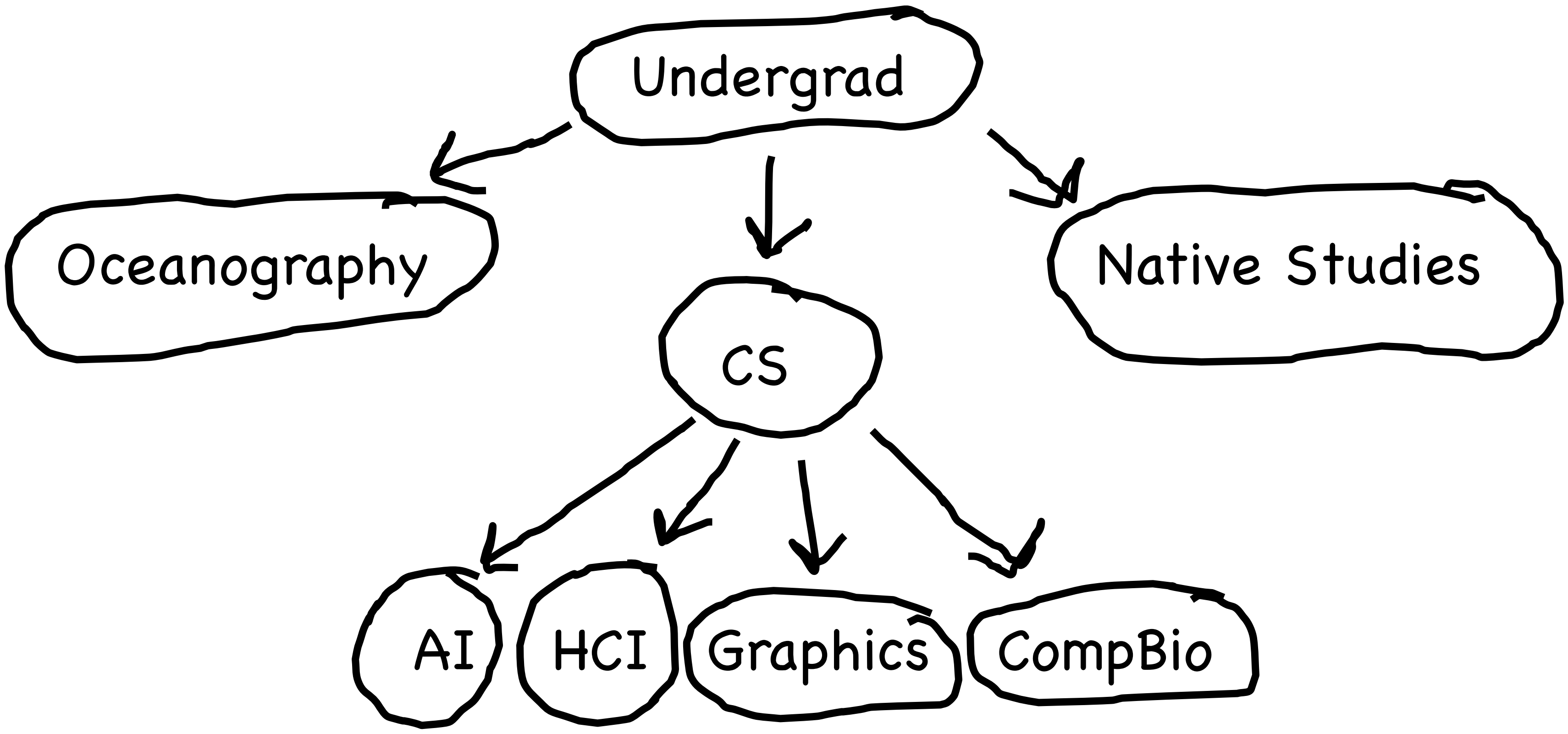
Lets talk about your life decisions

Decision Tree

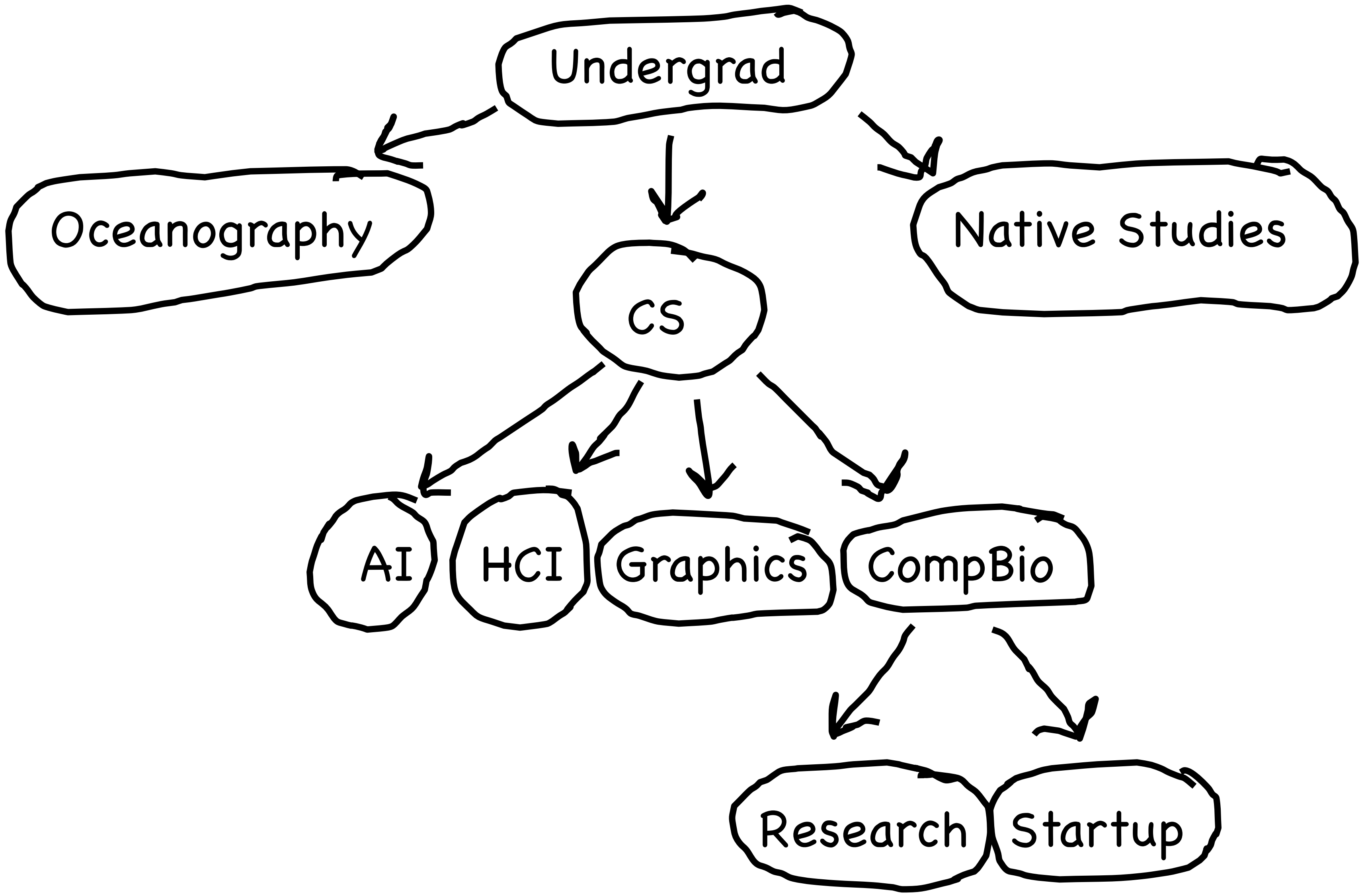
Decision Tree



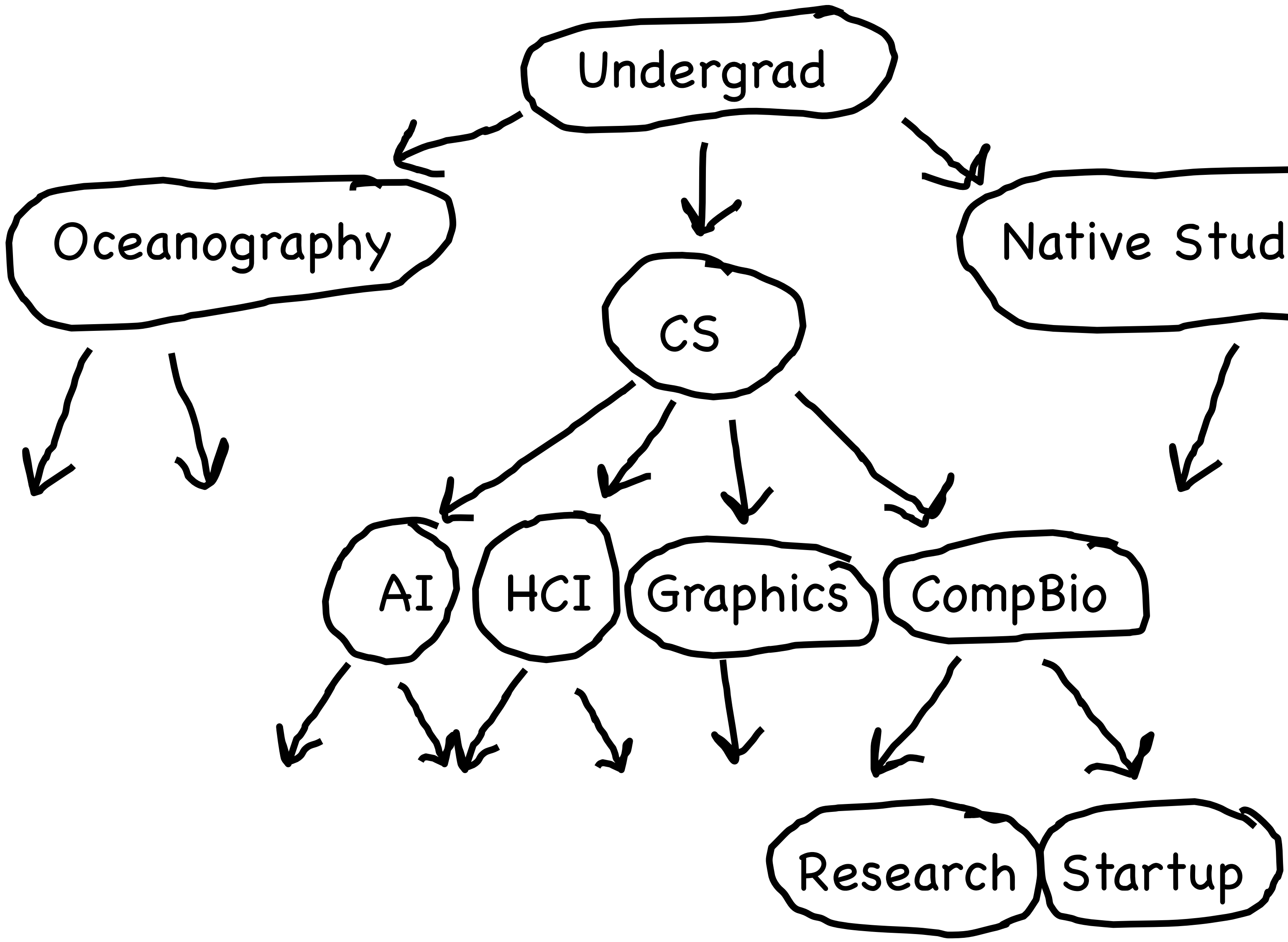
Decision Tree



Decision Tree



Decision Tree



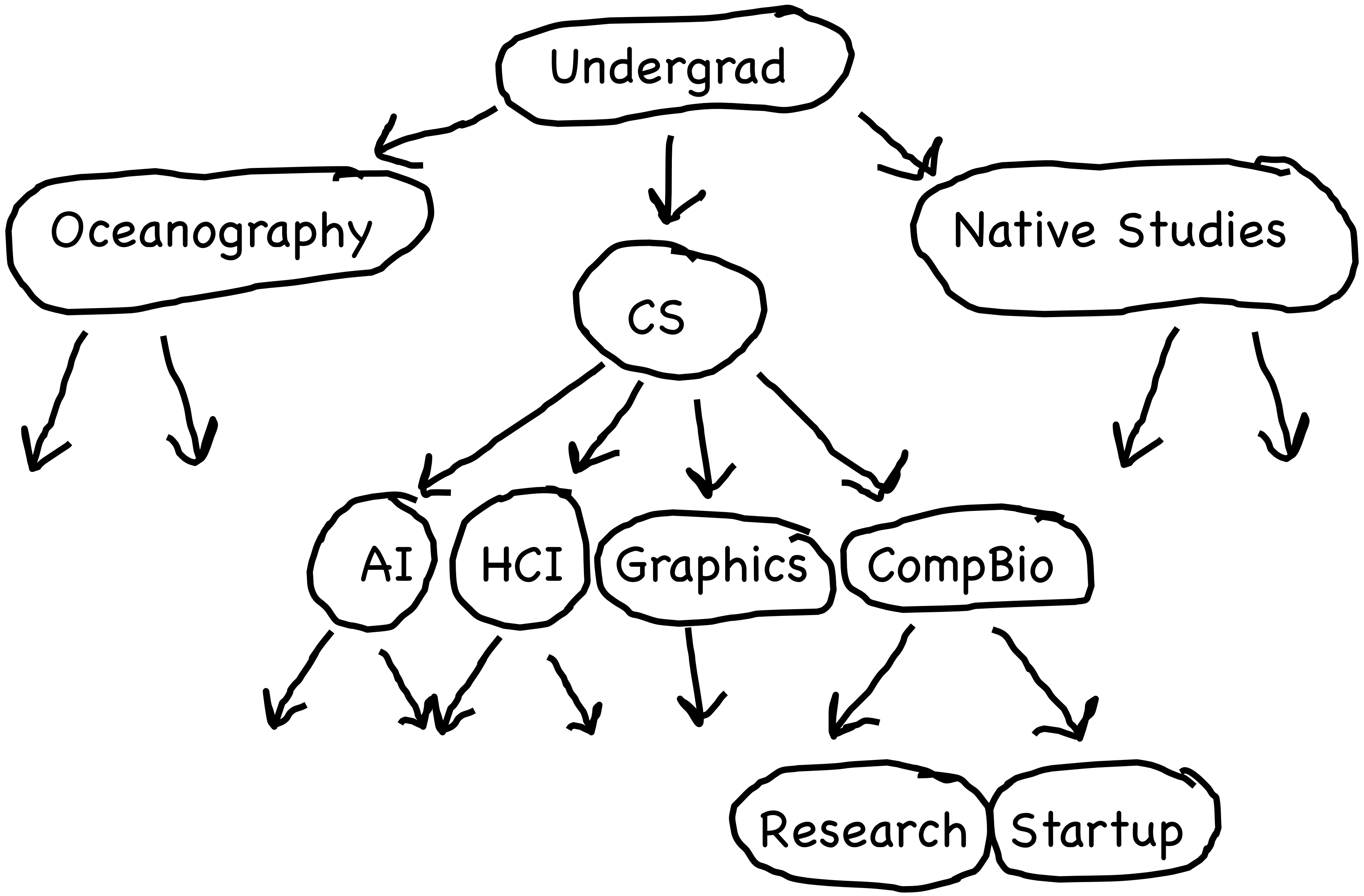
Aside

Less important then say love

But the love decision tree is more complex

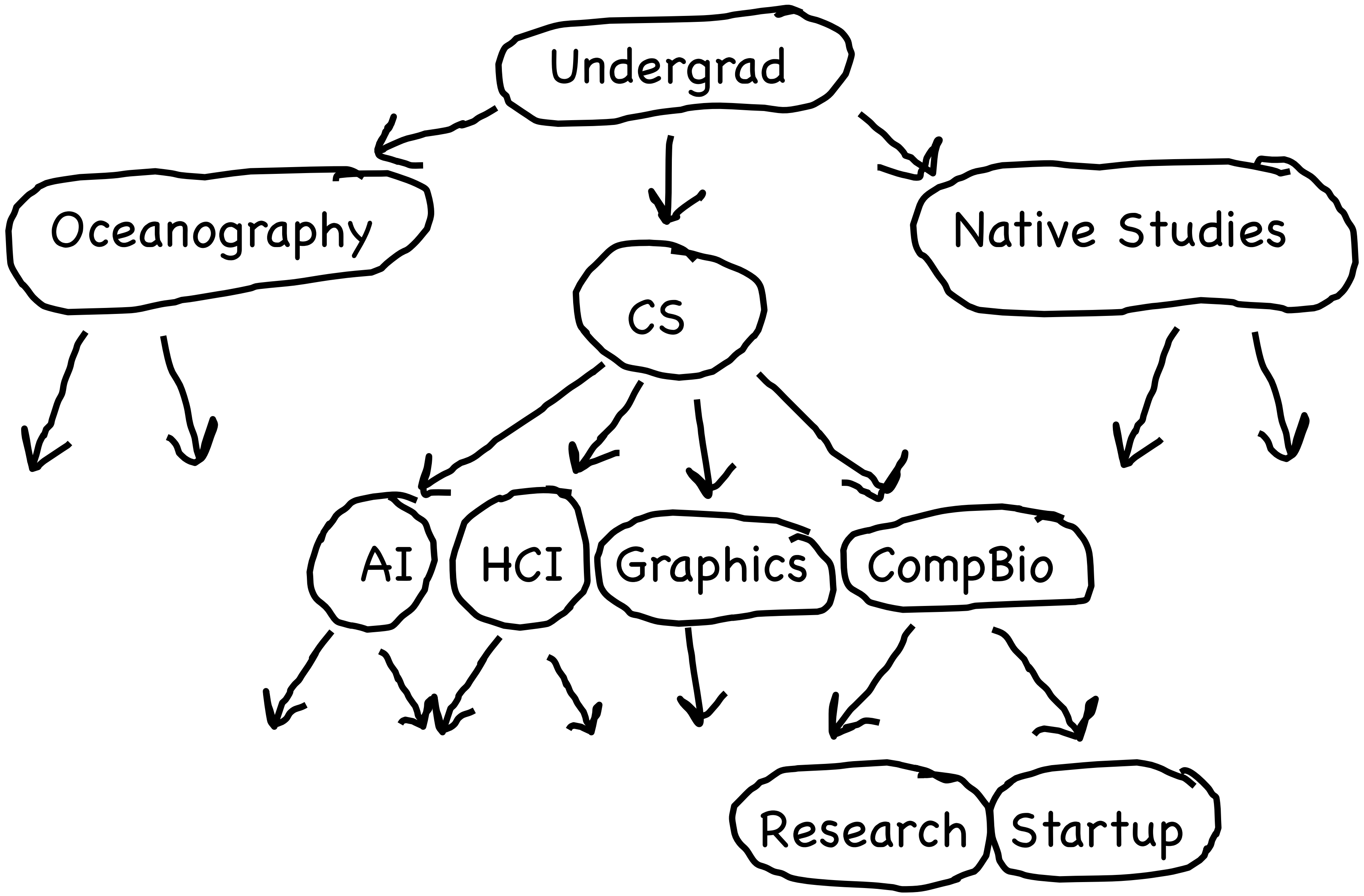
End aside

Decision Tree

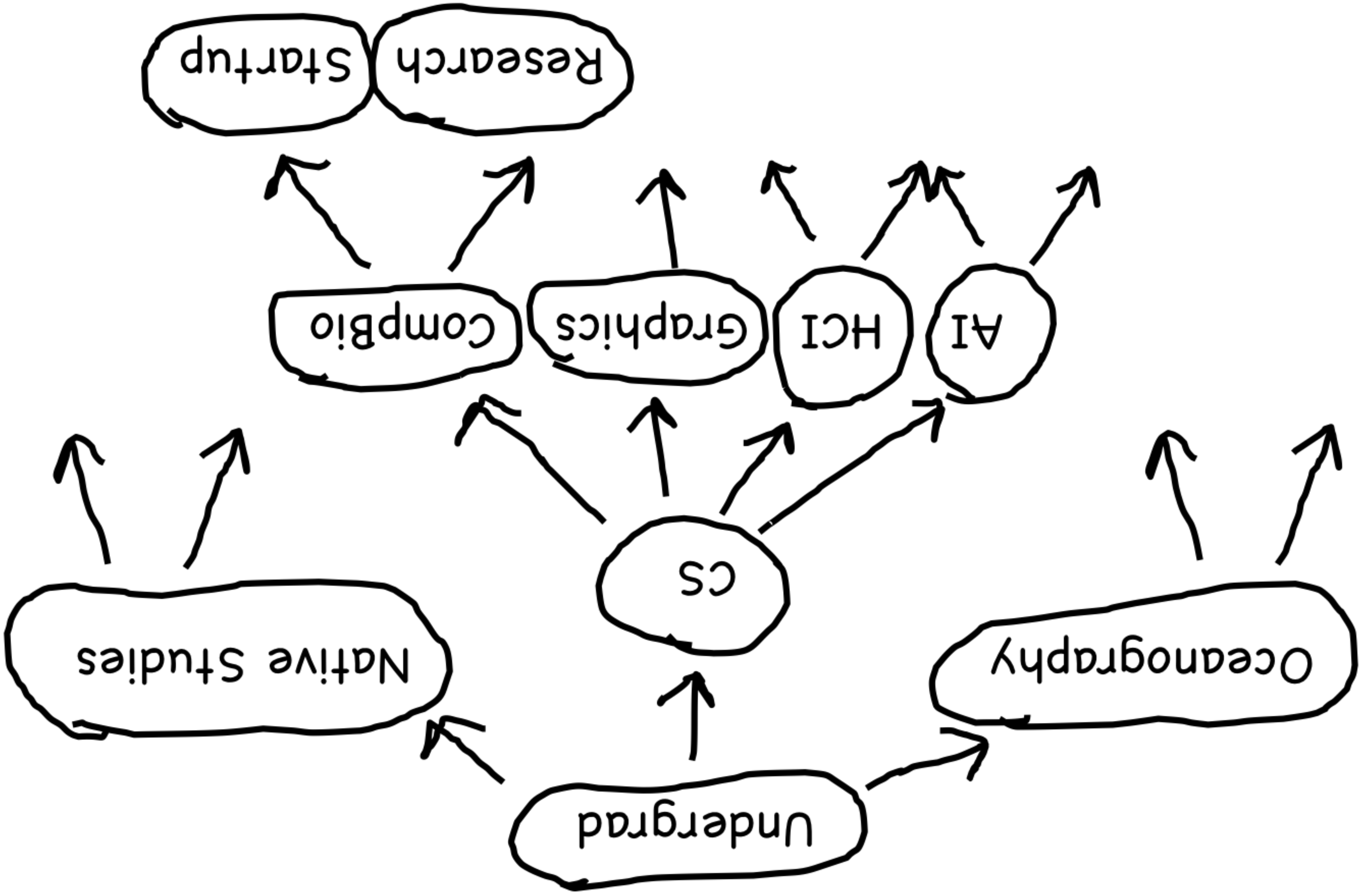


Why Decision Tree?

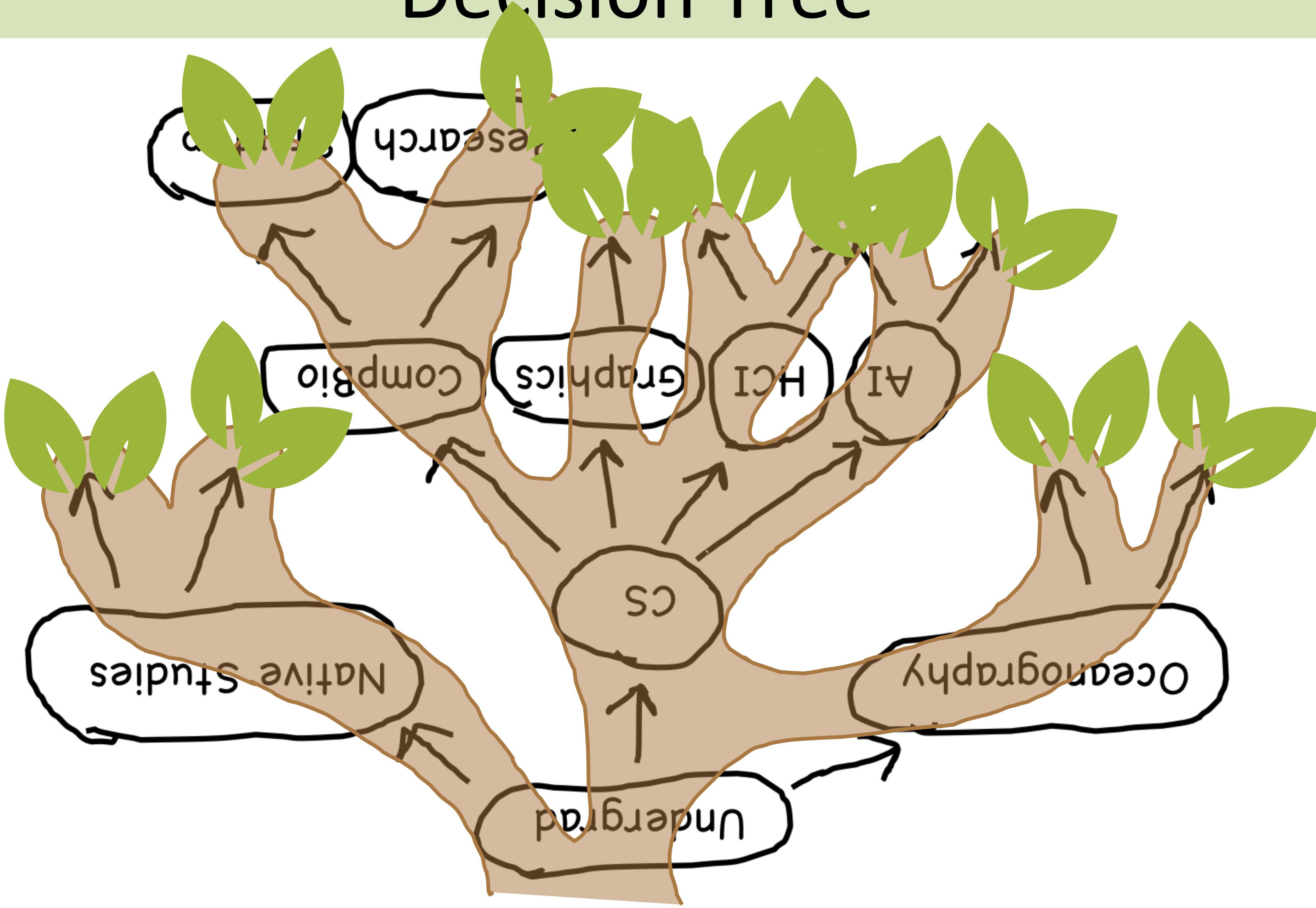
Decision Tree



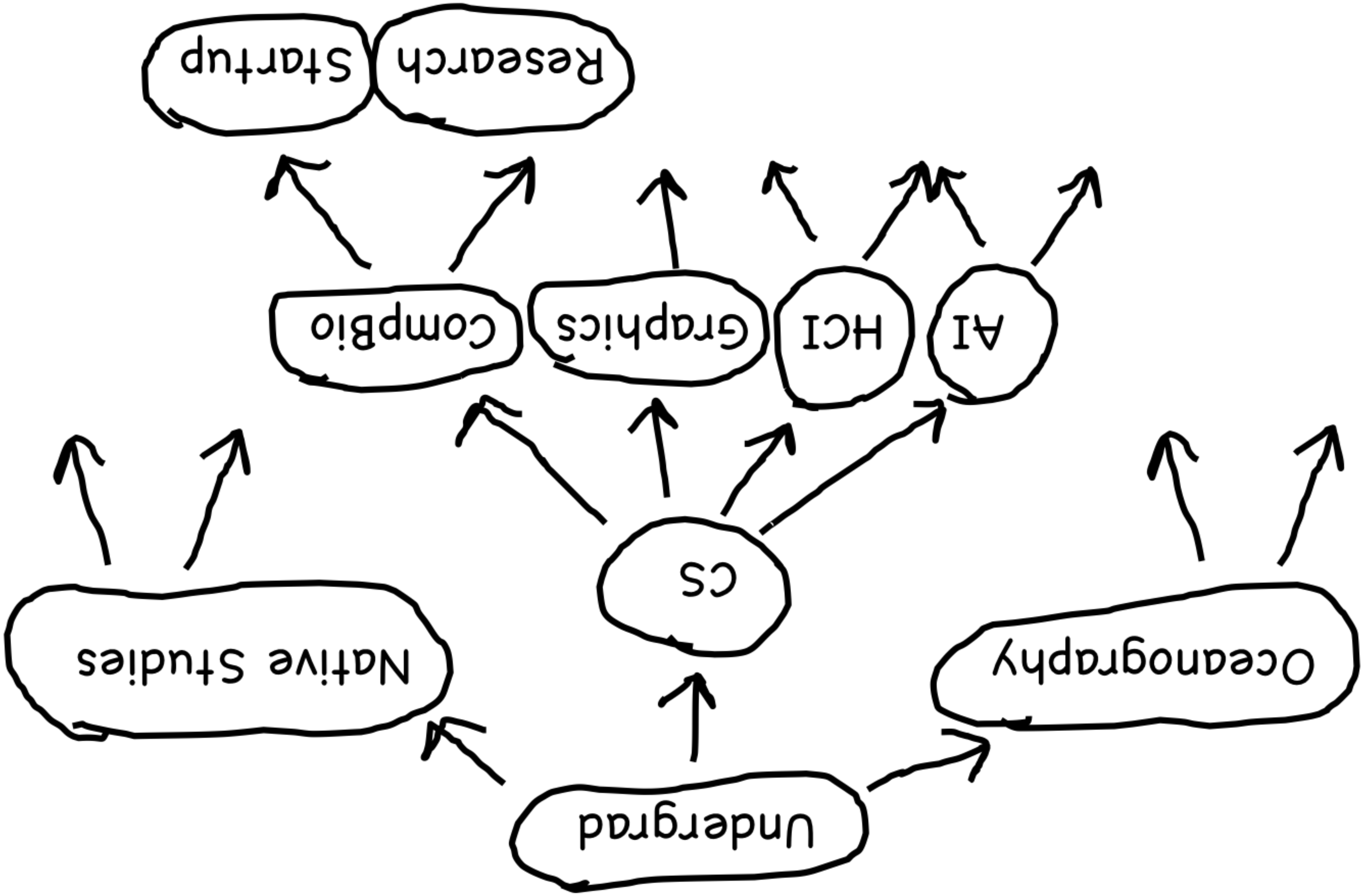
Decision Tree



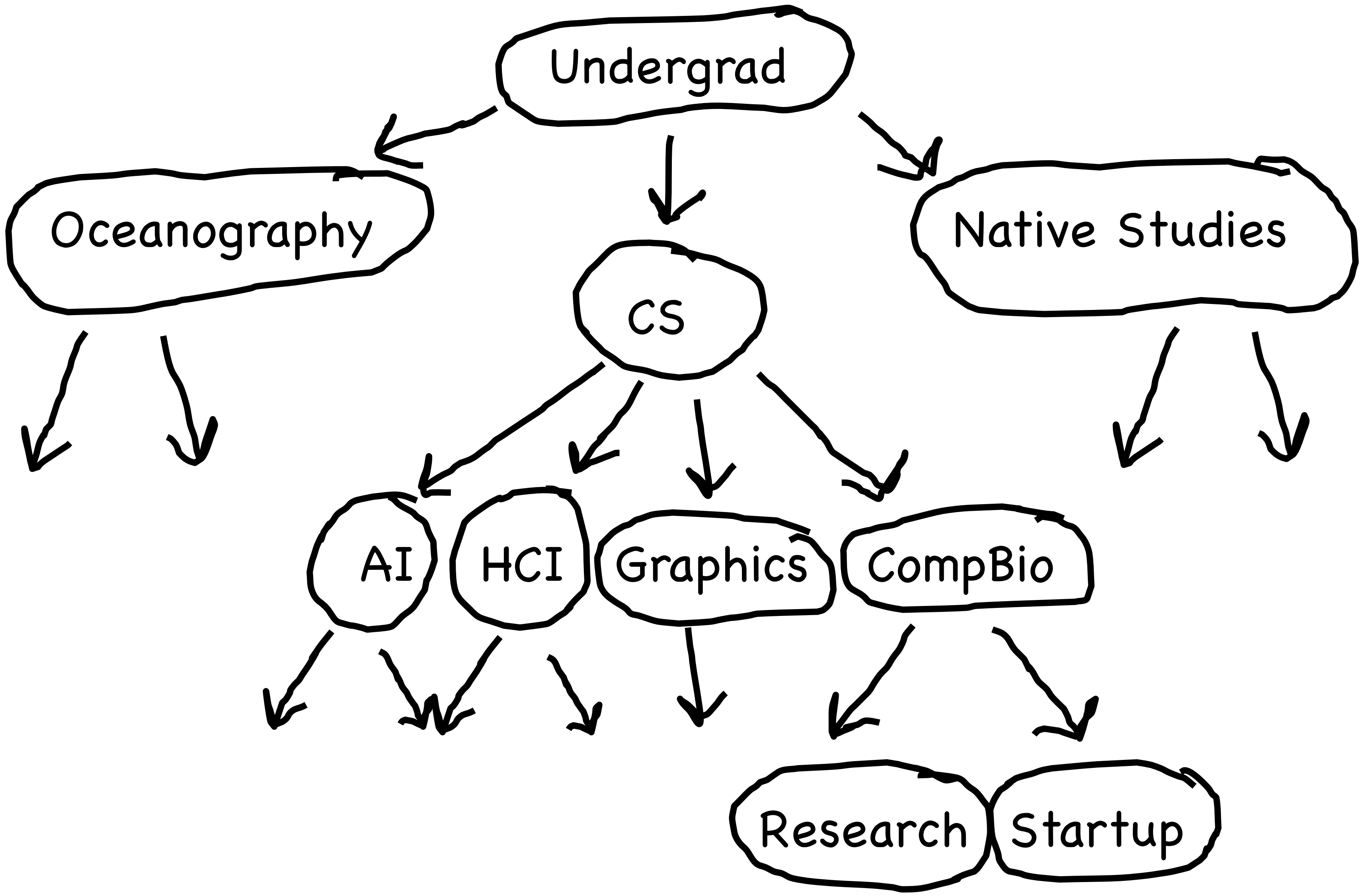
Decision Tree



Decision Tree

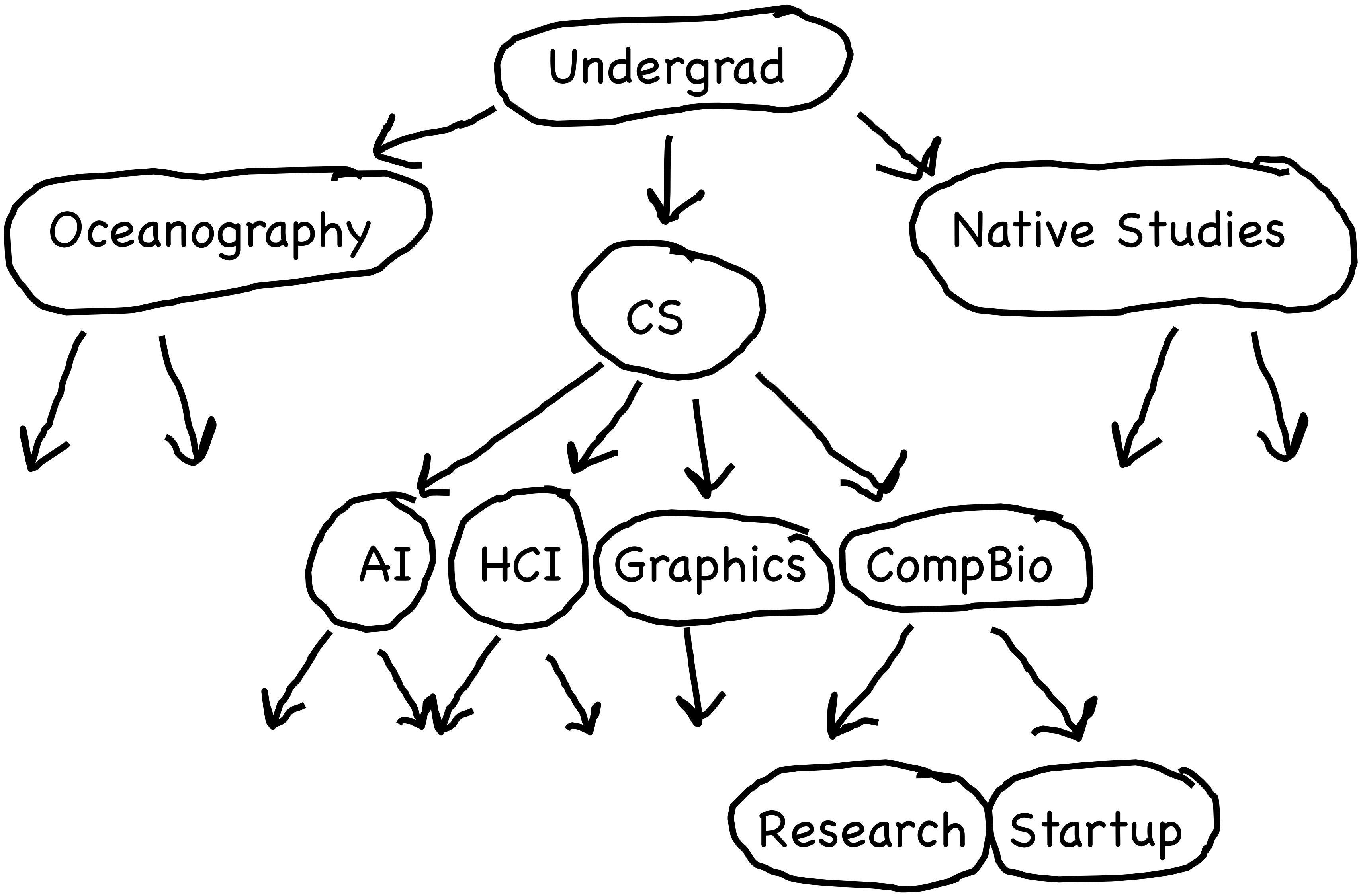


Decision Tree

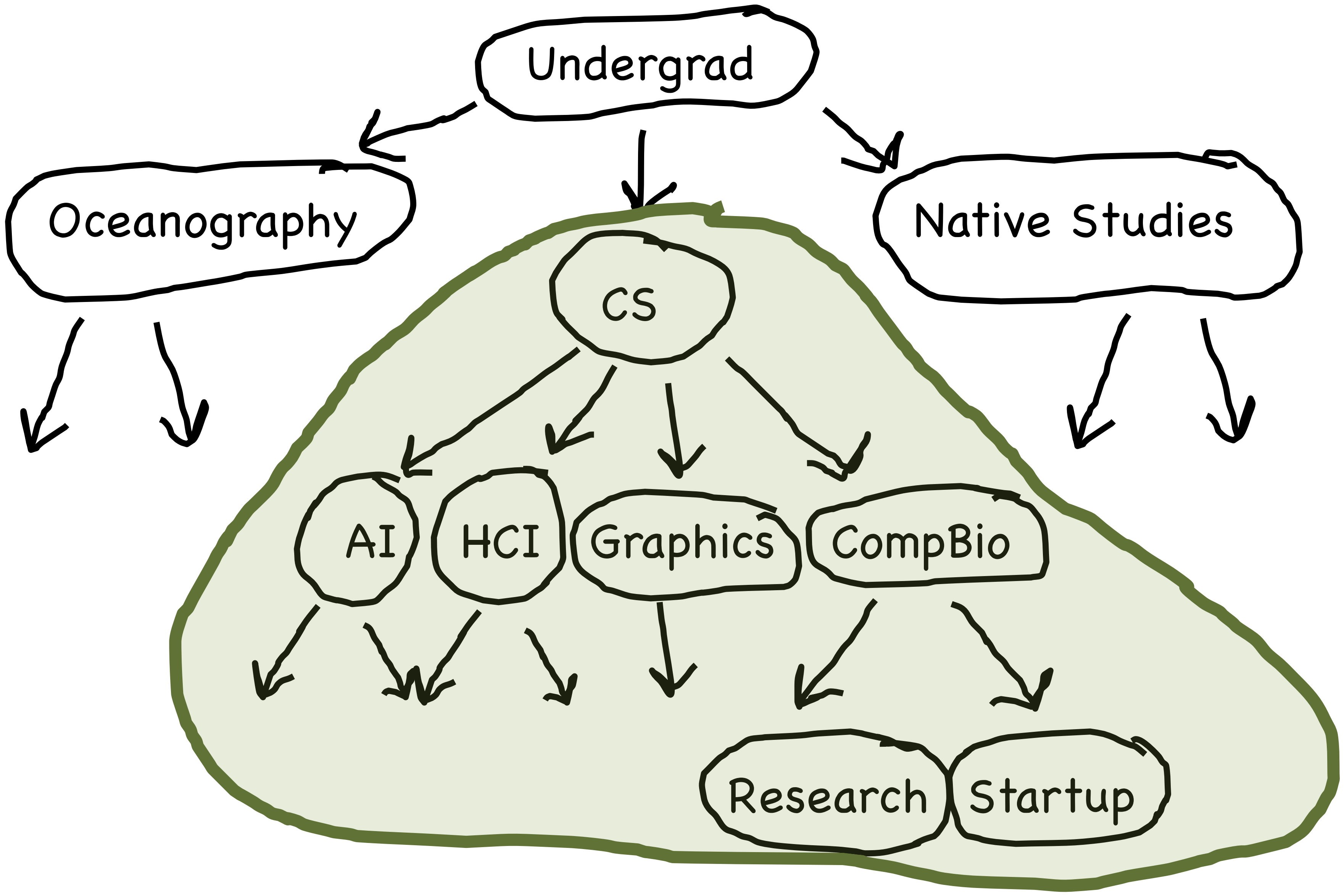


Decision Trees are Recursive

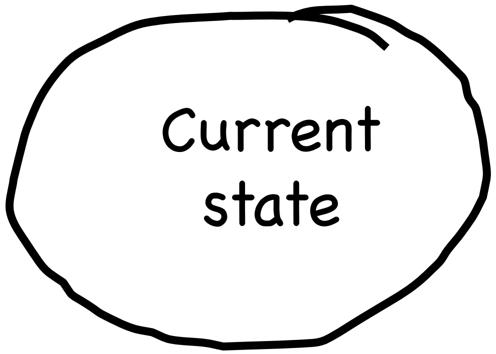
Decision Tree



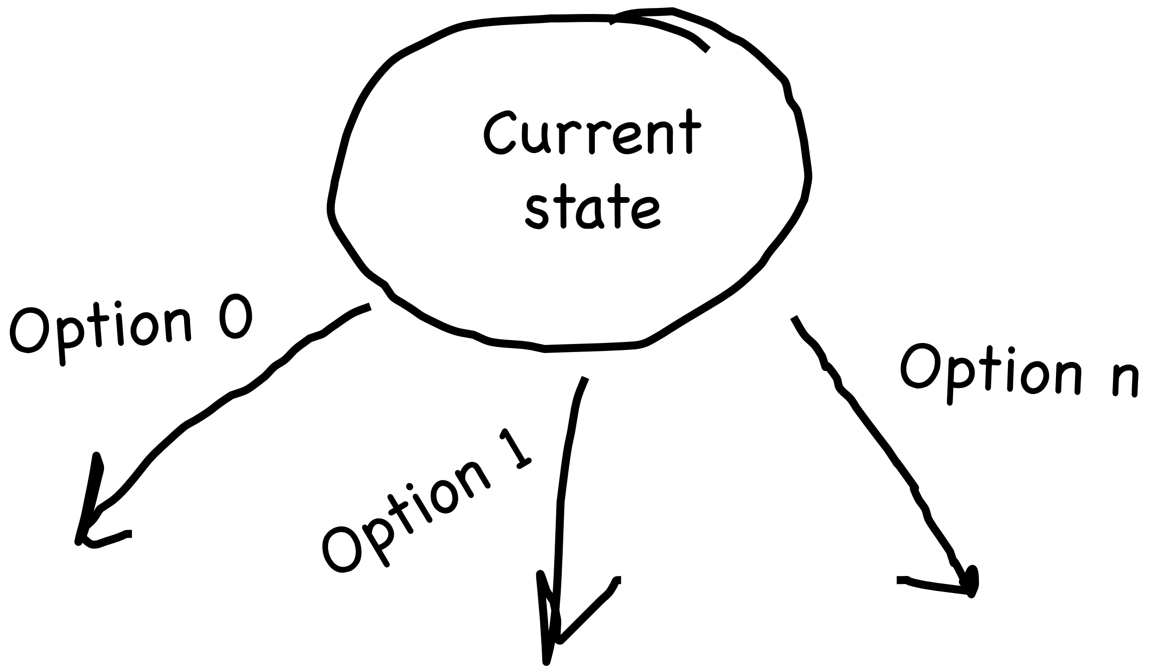
Decision Tree



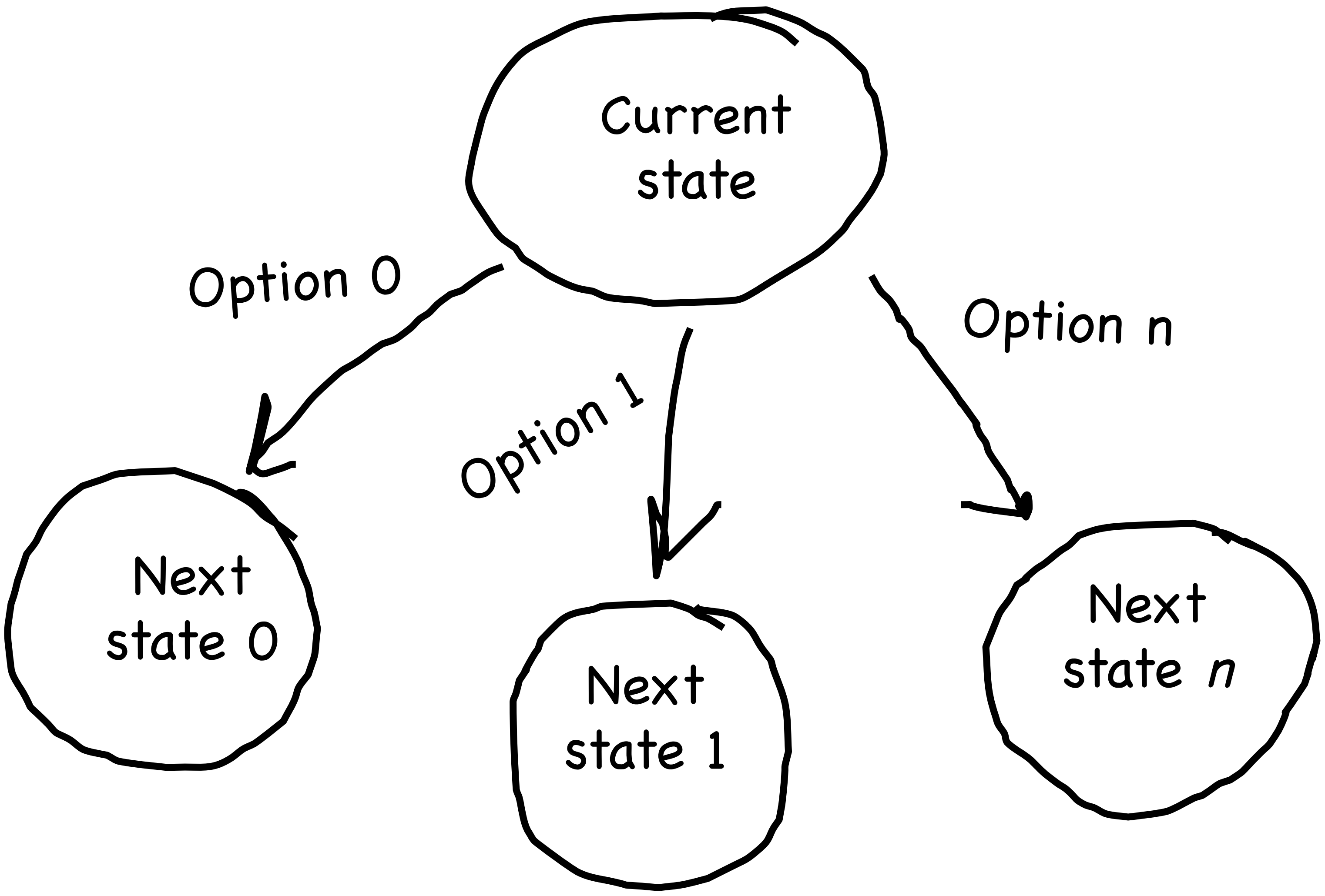
Decision Tree



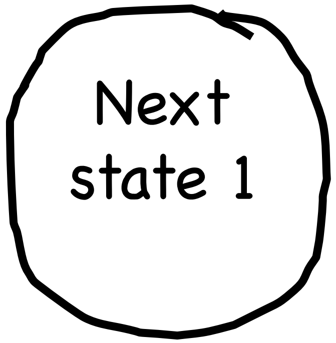
Decision Tree



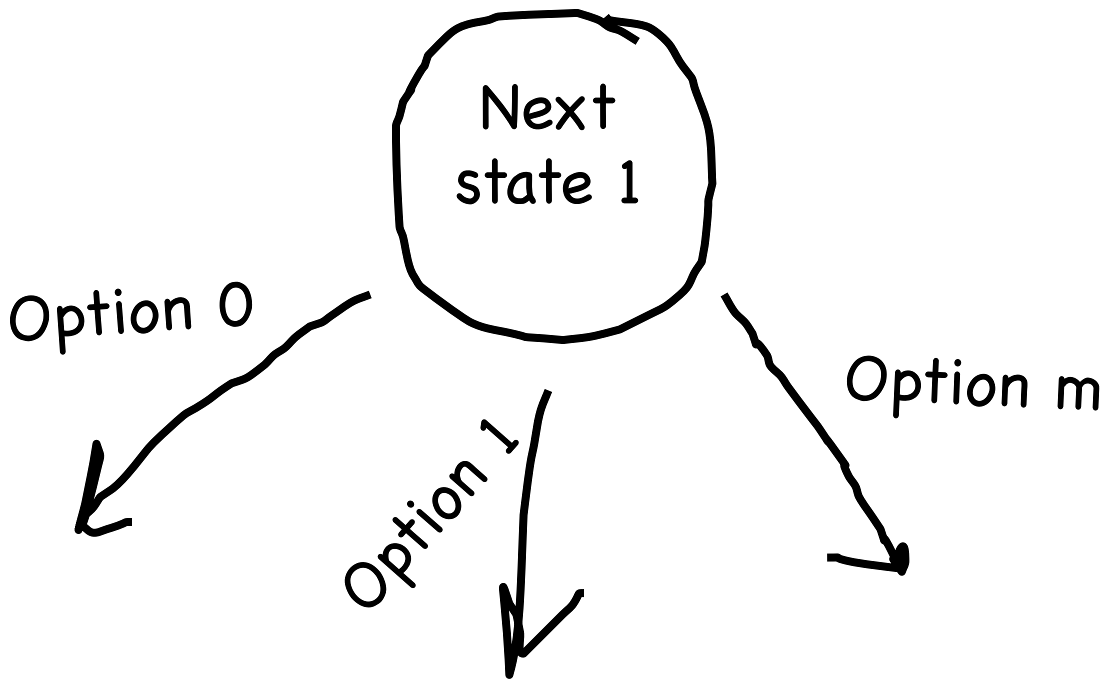
Decision Tree



Decision Tree



Decision Tree



Recursion is a great tool for exploring decisions

Output all paths

Output all Paths

```
void outputAllPaths(Vector<string> soFar) {
  if(endOfUndergrad(soFar)) {
    cout << soFar << endl;
  } else {
    Set<string> options = getOptions(soFar);
    for(string option : options) {
      Vector<string> next = soFar;
      next.add(option);
      outputAllPaths(next);
    }
  }
}
```

Output all Paths

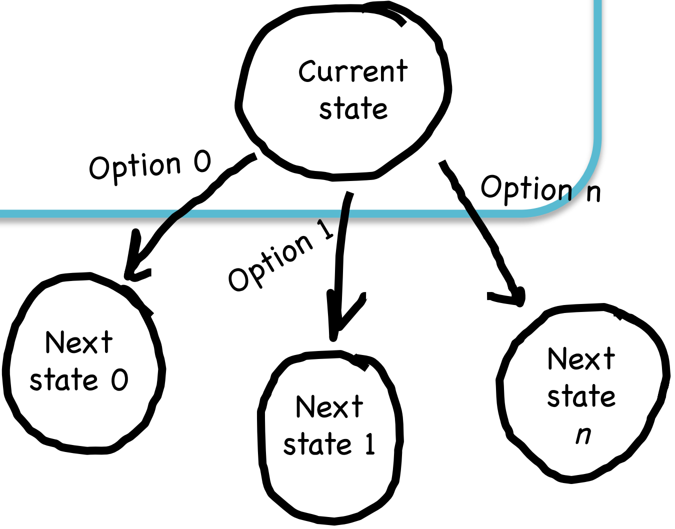
```
void outputAllPaths(Vector<string> soFar) {
  if(endOfUndergrad(soFar)) {
    cout << soFar << endl;
  } else {
    Set<string> options = getOptions(soFar);
    for(string option : options) {
      Vector<string> next = soFar;
      next.add(option);
      outputAllPaths(next);
    }
  }
}
```

Output all Paths

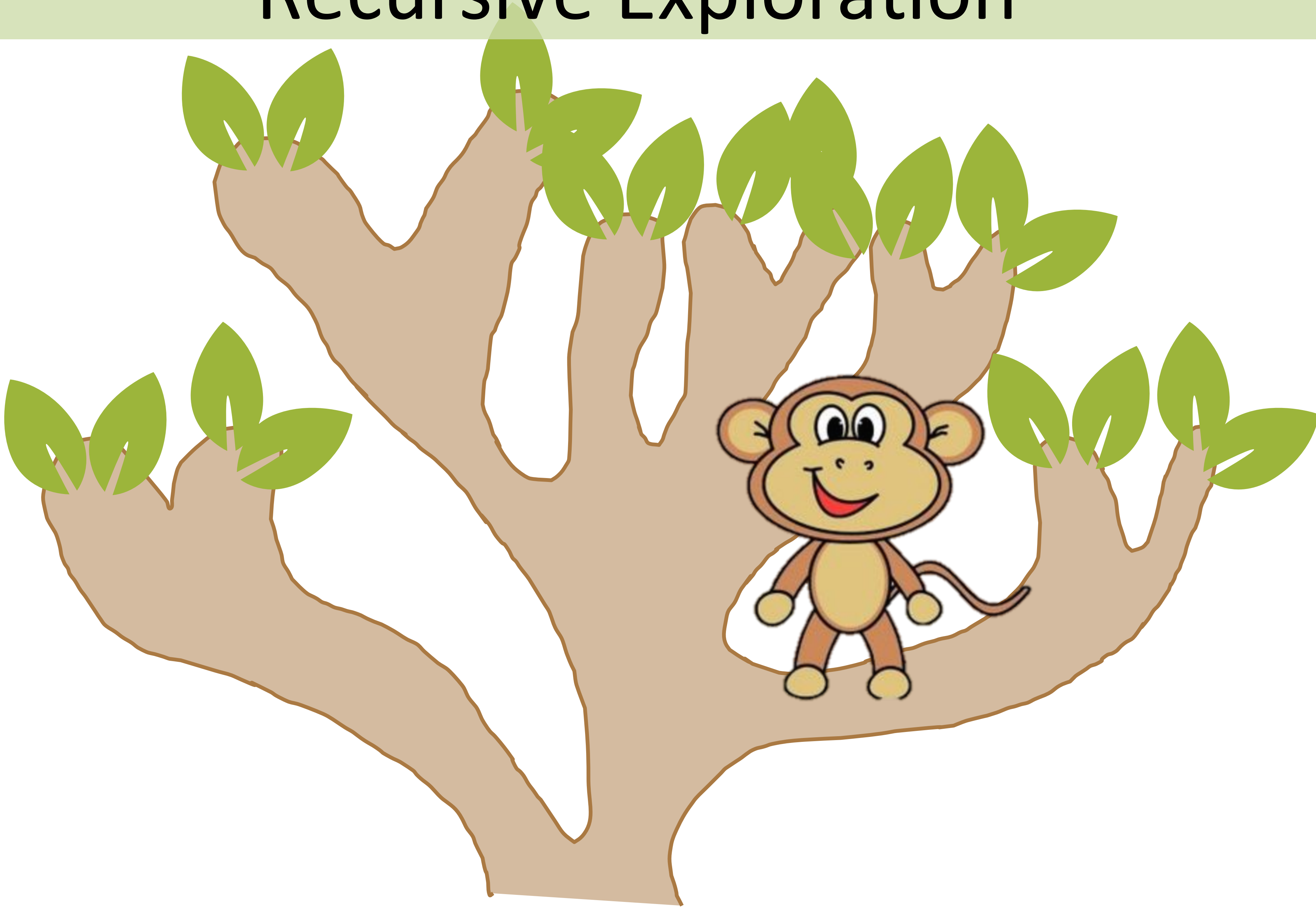
```
void outputAllPaths(Vector<string> soFar) {
  if(endOfUndergrad(soFar)) {
    cout << soFar << endl;
  } else {
    Set<string> options = getOptions(soFar);
    for(string option : options) {
      Vector<string> next = soFar;
      next.add(option);
      outputAllPaths(next);
    }
  }
}
```

Output all Paths

```
void outputAllPaths(Vector<string> soFar) {
  if(endOfUndergrad(soFar)) {
    cout << soFar << endl;
  } else {
    Set<string> options = getOptions(soFar);
    for(string option : options) {
      Vector<string> next = soFar;
      next.add(option);
      outputAllPaths(next);
    }
  }
}
```



Recursive Exploration

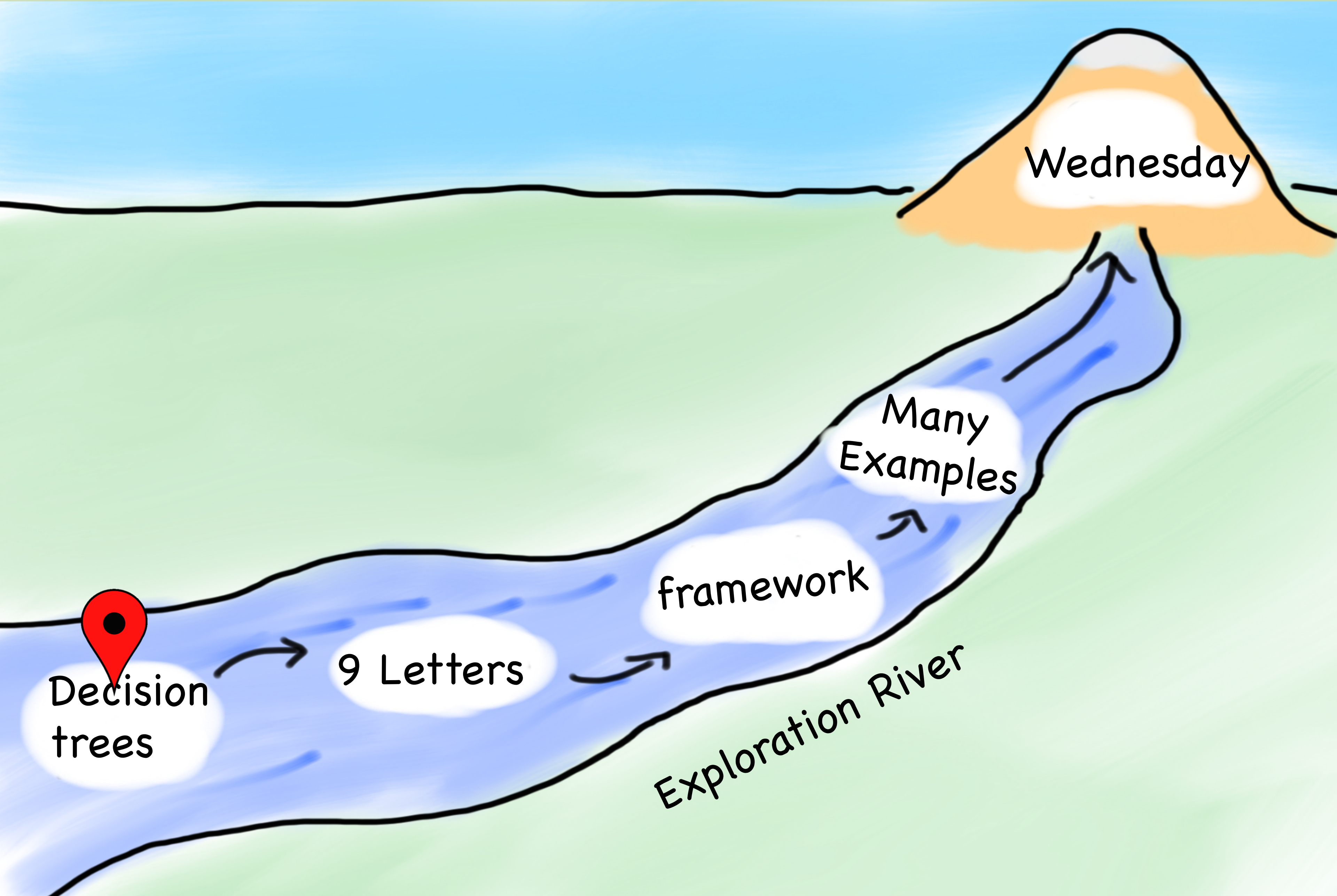


Recursive Exploration

Has other names

- Recursive Depth First Search
- Try Everything Search
- Recursive Backtracking (subset)

Today's Route



Today's Route

Wednesday
Many Examples
framework
9 Letters
Decision trees
Exploration River

A Little Word Puzzle

“What nine-letter word can be reduced to a single-letter word one letter at a time by removing letters, leaving it a legal word at each step?”

The Startling Truth

S	T	A	R	T	L	I	N	G

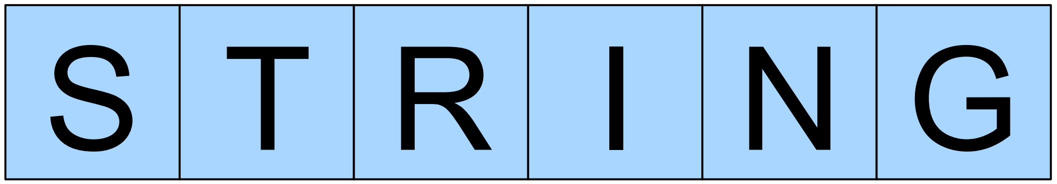
The Startling Truth

S	T	A	R	T	I	N	G

The Startling Truth

S	T	A	R	I	N	G

The Startling Truth



The Startling Truth

S	T	I	N	G

The Startling Truth

S	I	N	G

The Startling Truth

S	I	N

The Startling Truth

I	N

The Startling Truth

I

Is there **really** just one nine-letter word with this property?

Iterative?

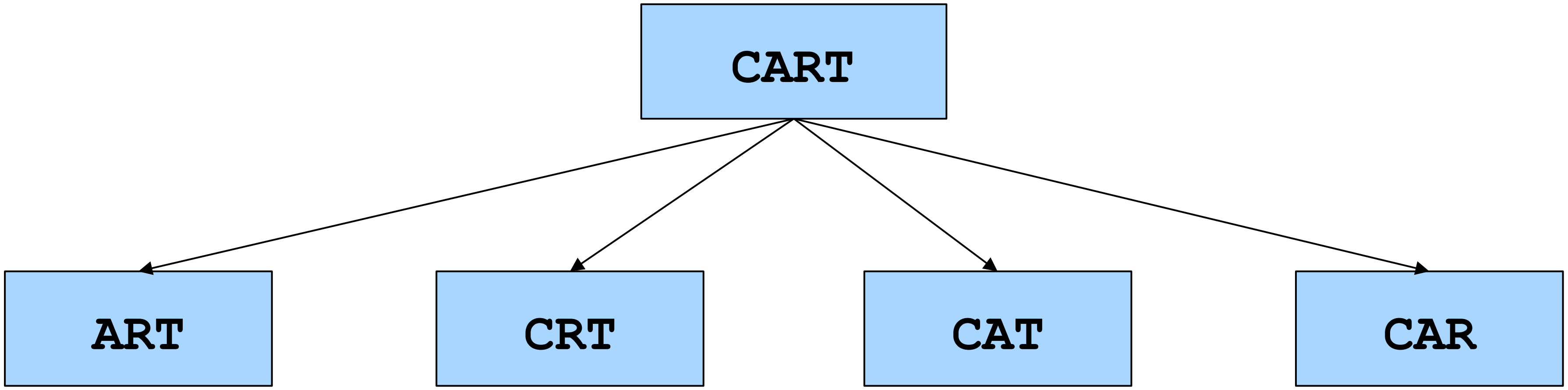
Recursive?

Decision Tree?

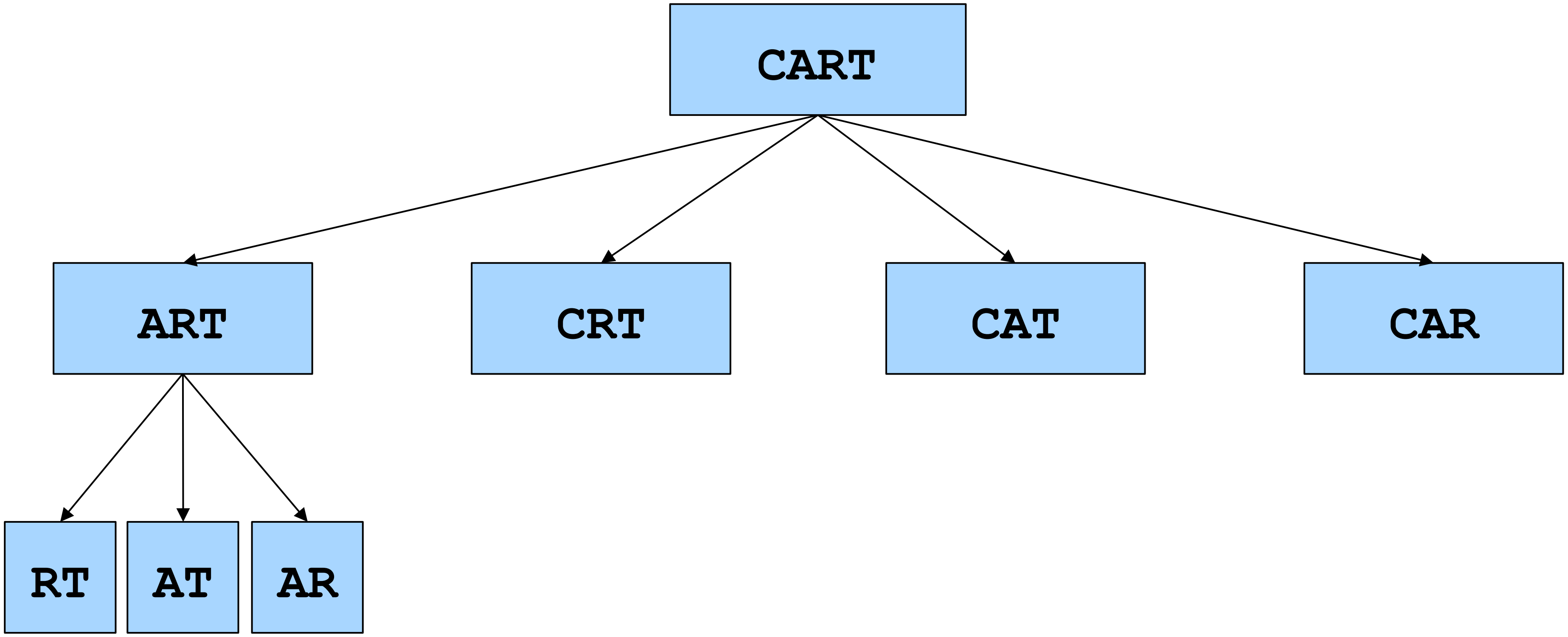
Reduce Decision Tree

CART

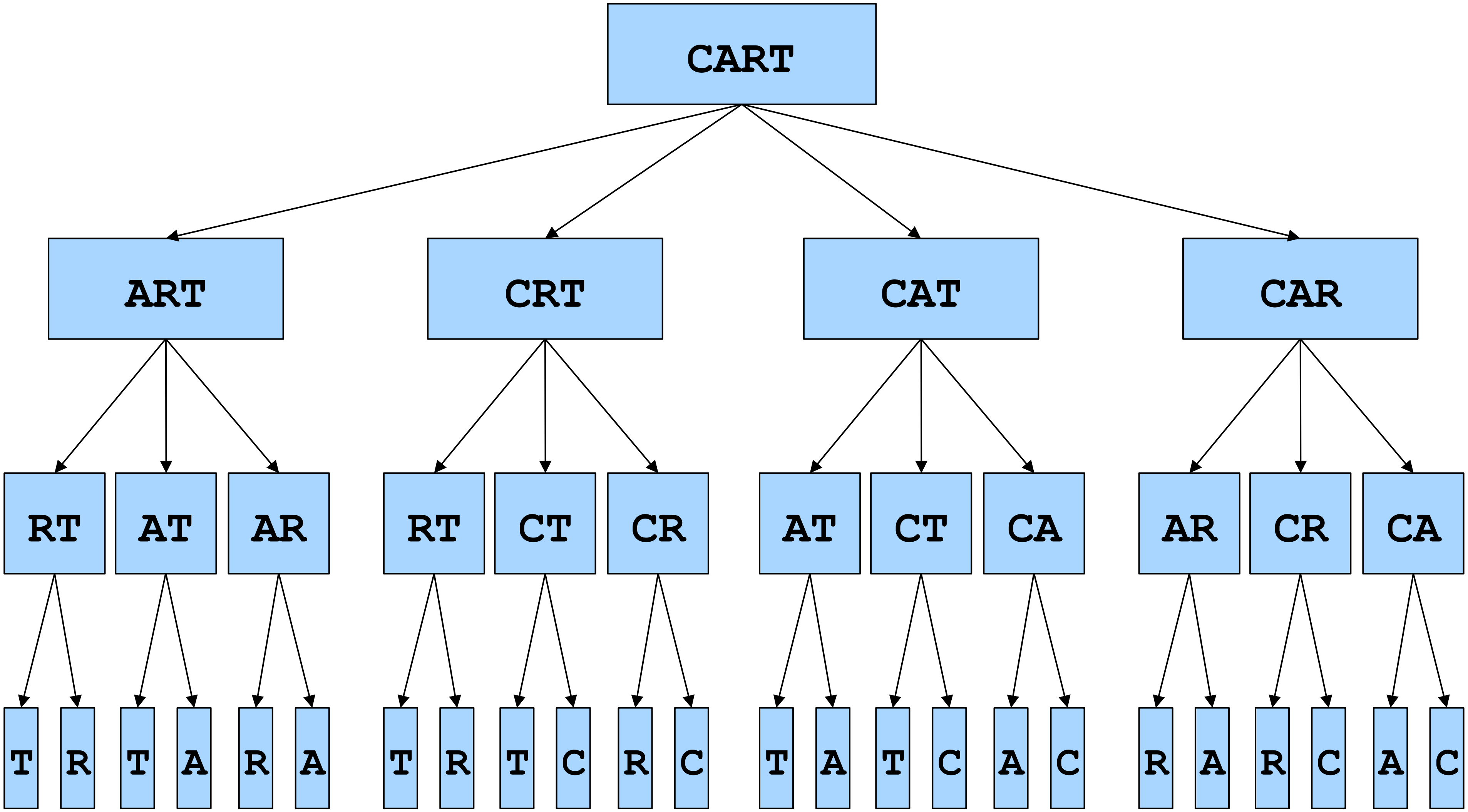
Reduce Decision Tree



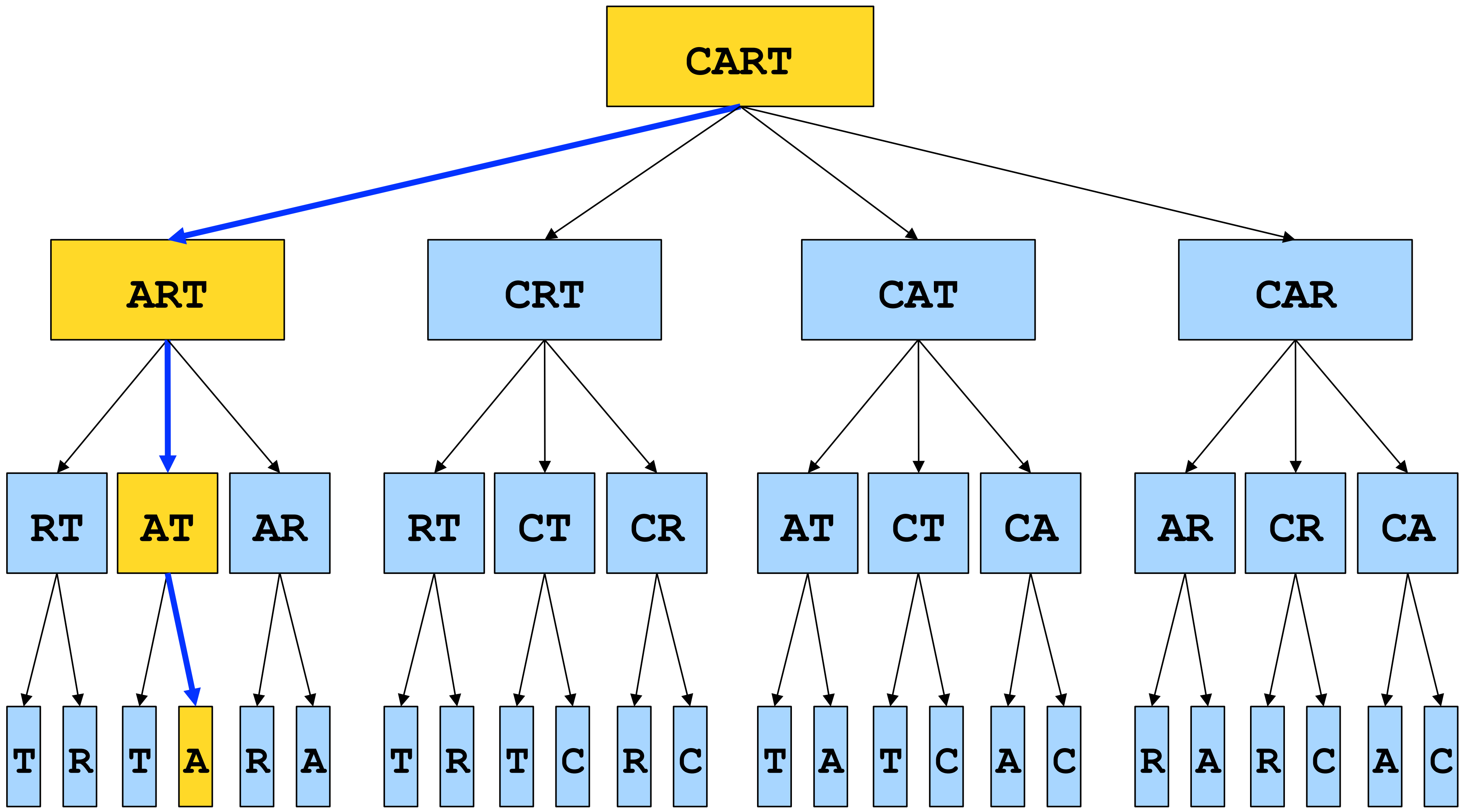
Reduce Decision Tree



Reduce Decision Tree



Reduce Decision Tree



Decision Tree Search

```
bool search(currentState) {
  if(isSolution(currentState)) {
    return true;
  } else {
    for(option : moves from currentState) {
      nextState = takeOption(curr, option);
      if(search(nextState)){
        return true;
      }
    }
    return false;
  }
}
```

Reducible Word

Let's define a **reducible word** as a word that can be reduced down to one letter by removing one character at a time, leaving a word at each step.

- **Base Cases:**
  - The empty string
- **Recursive Step:**
  - Any multi-letter word is reducible if you can remove a letter (legal move) to form a shrinkable word.

Decision Tree Search

```
bool reducible(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(reducible(lex, copy)){
               return true;
            }
         }
      }
      return false;
   }
}
```

Decision Tree Search

```
bool reducible(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(reducible(lex, copy)){
               return true;
            }
         }
      }
      return false;
   }
}
```

Get all legal moves, and corresponding next states

Decision Tree Search

```
bool reducible(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(reducible(lex, copy)){
               return true;
            }
         }
      }
      return false;
   }
}
```

Get all legal moves, and corresponding next states

Decision Tree Search

```
bool reducable(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(reducable(lex, copy)){
               return true;
            }
         }
      }
      return false;
   }
}
```

If any decision is reducible return true

Decision Tree Search

```
bool reducable(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(reducable(lex, copy)){
               return true;
            }
         }
      }
      return false;
   }
}
```

Only return false if every single option failed.

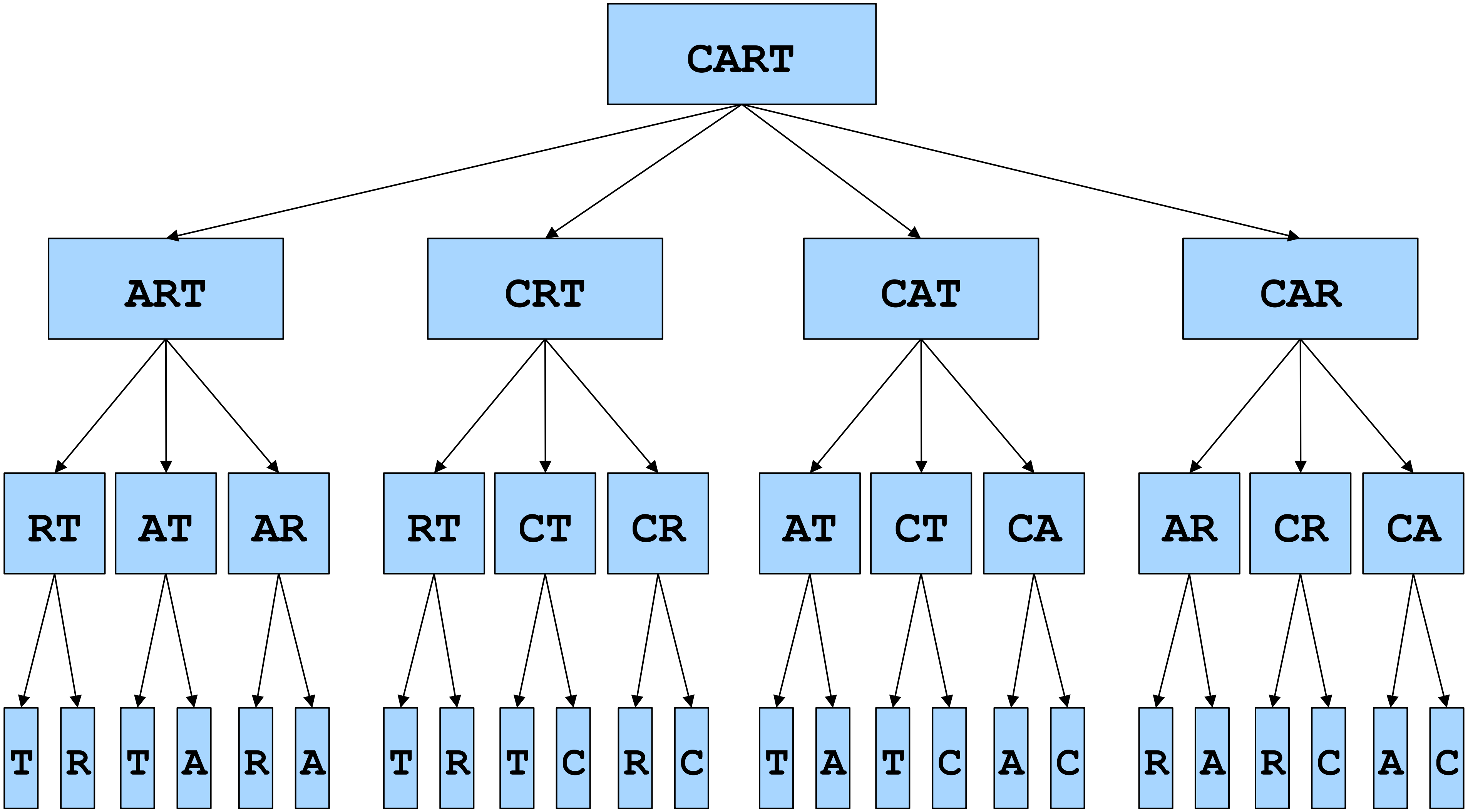
Is there **really** just one nine-letter word with this property?

Recursive Exploration

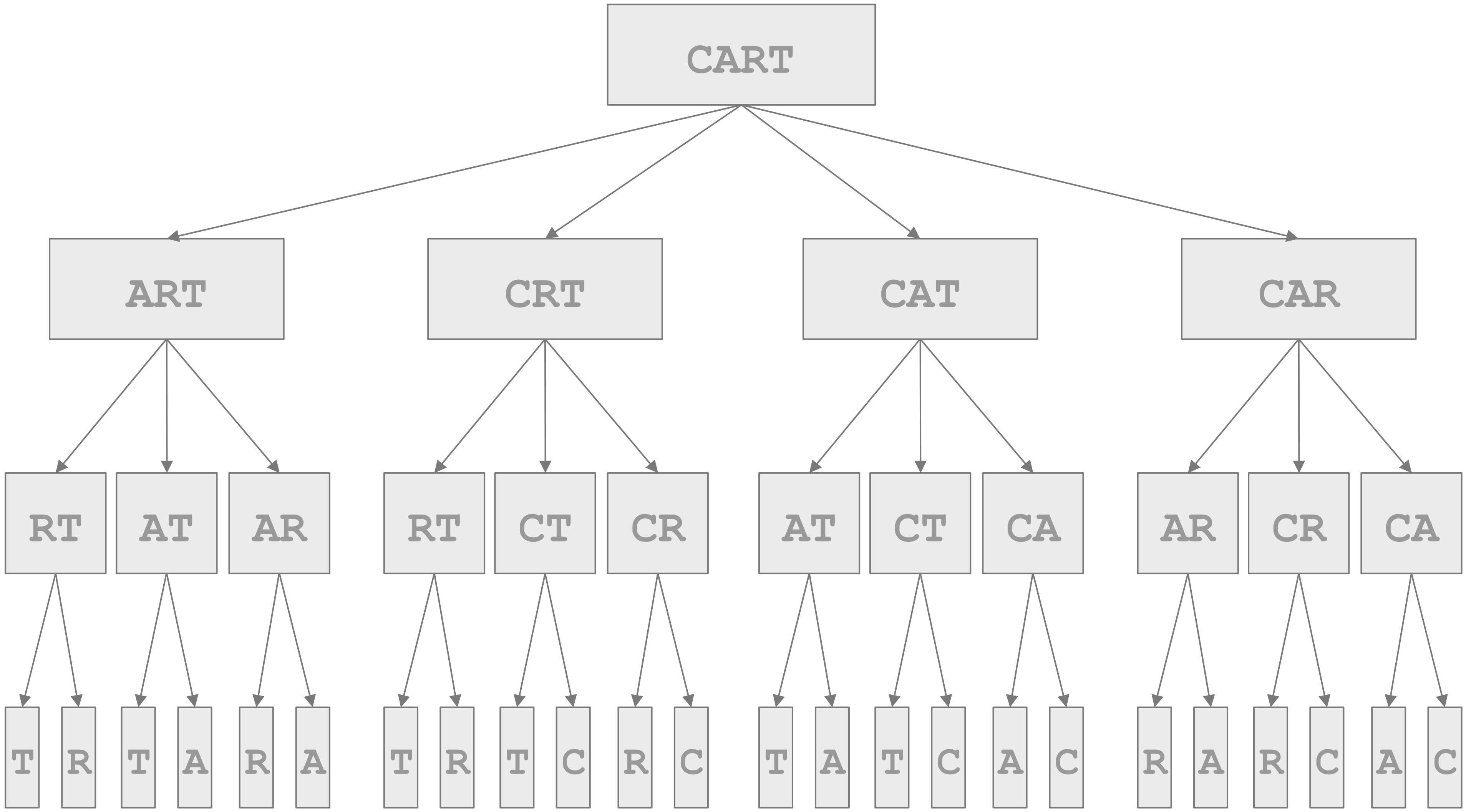
The function we have just defined is an example of **recursive exploration**. In this case we are looking for any path through the decision tree. For a given state:

- If *any* option leads to succeeds, that's great! We're done.
- If *none* of the options succeed, then this particular problem can't be solved from the state.

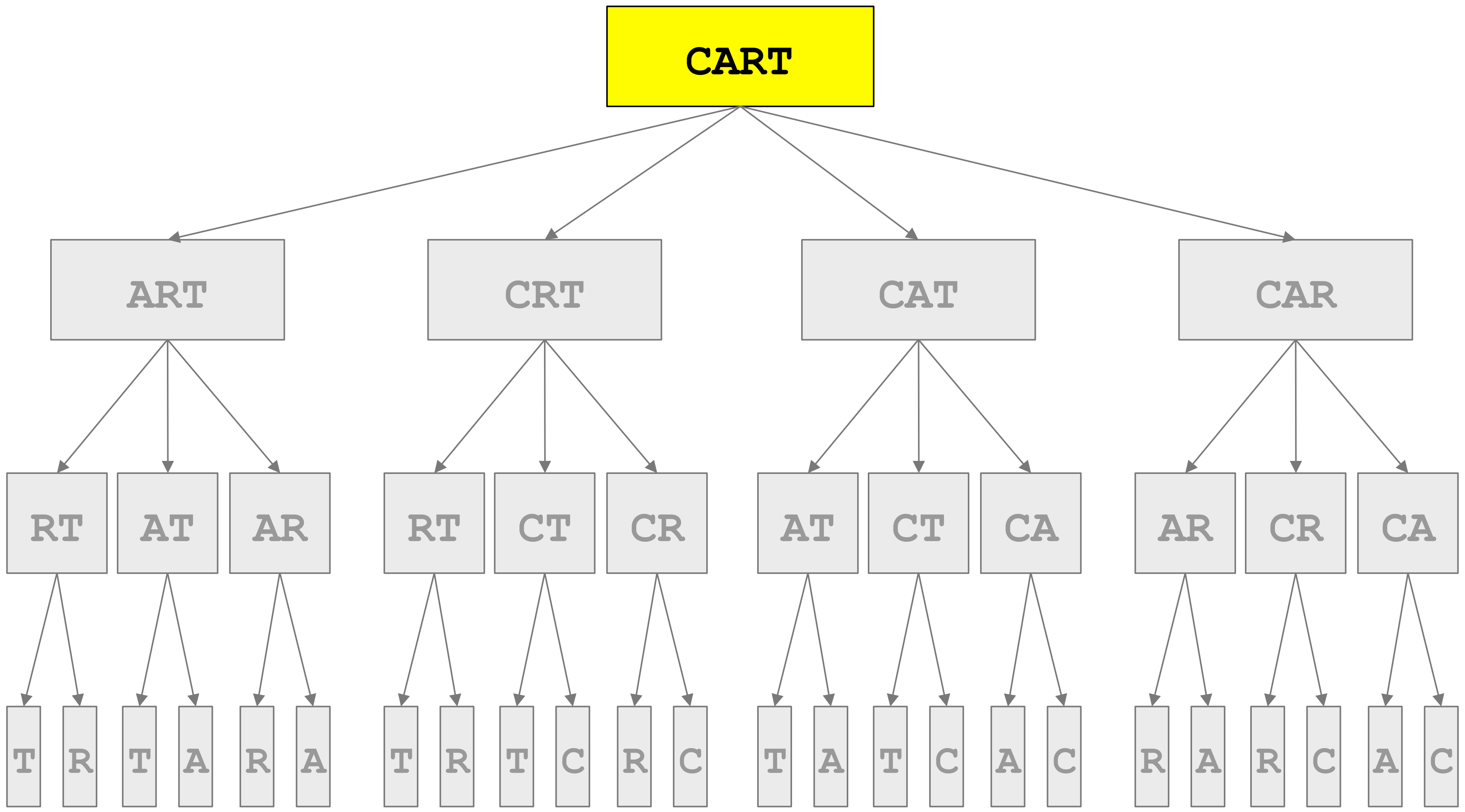
Recursive Exploration



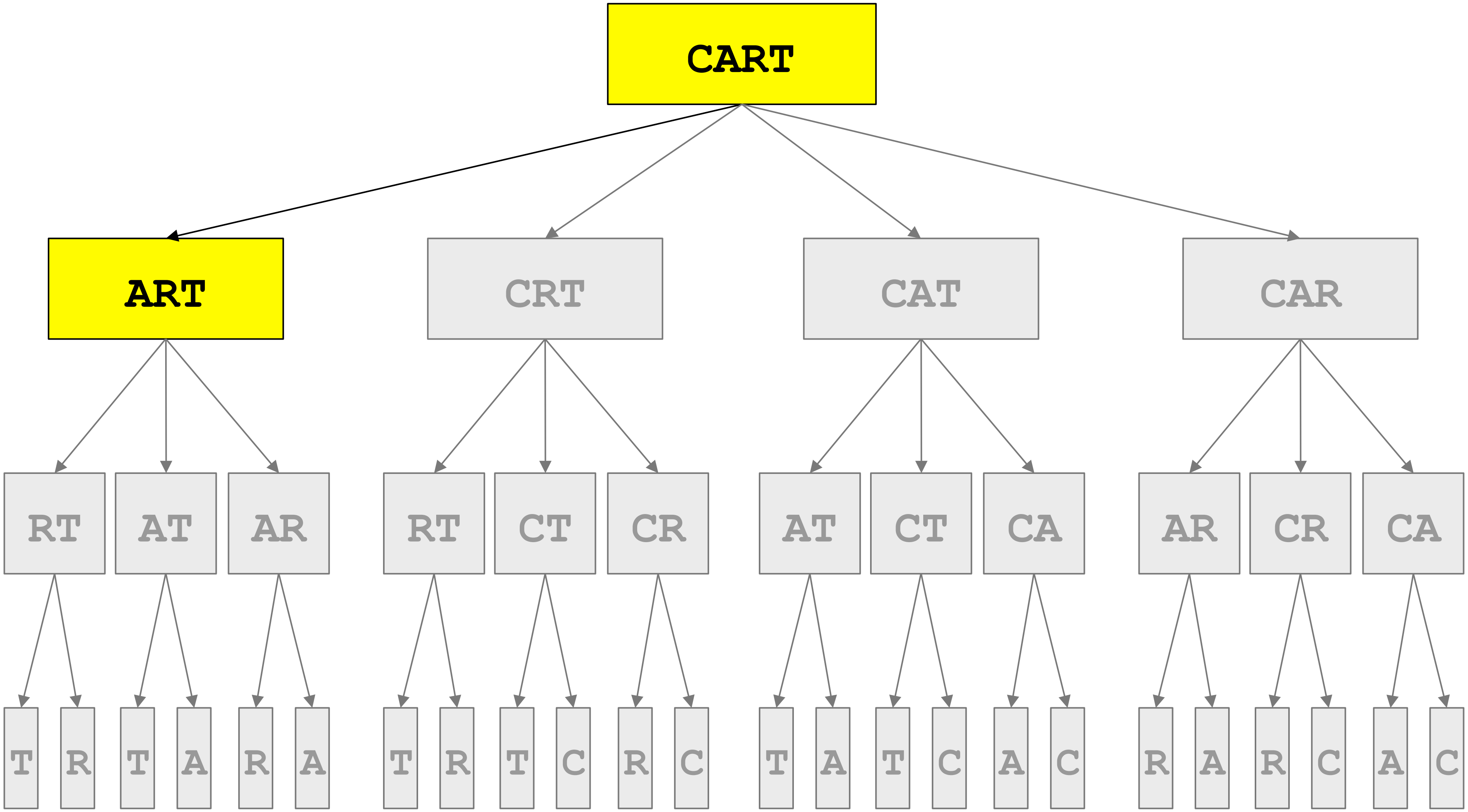
Recursive Exploration



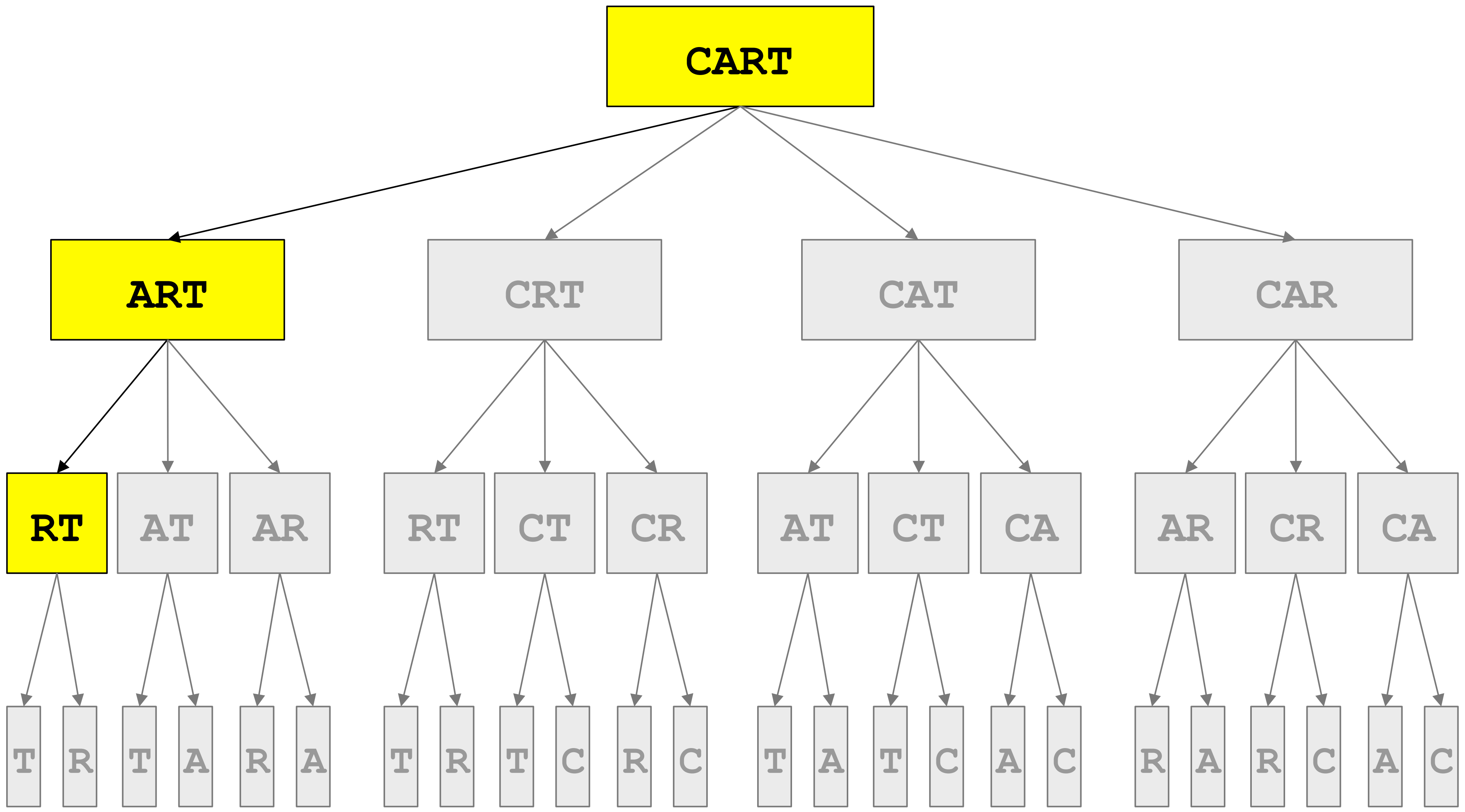
Recursive Exploration



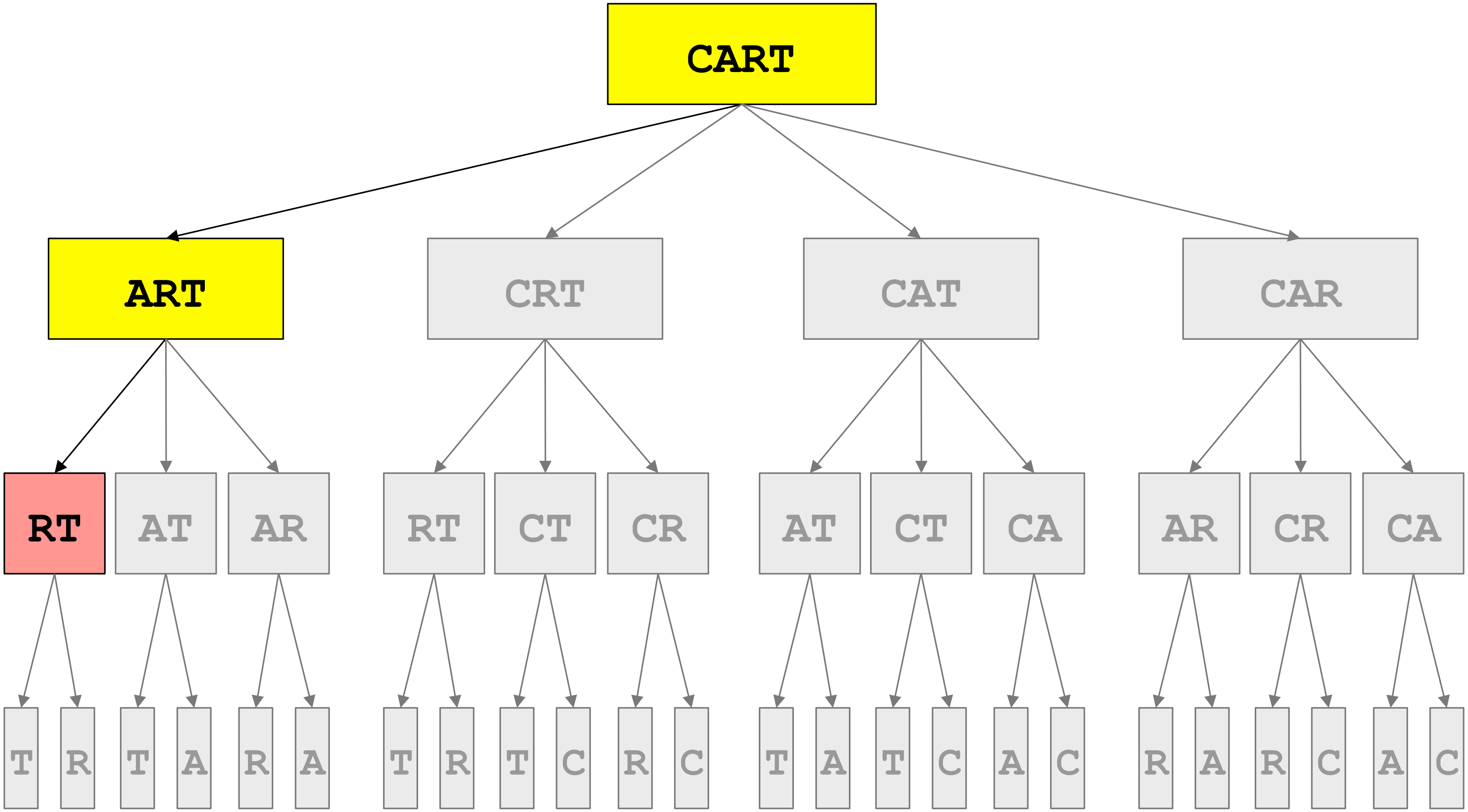
Recursive Exploration



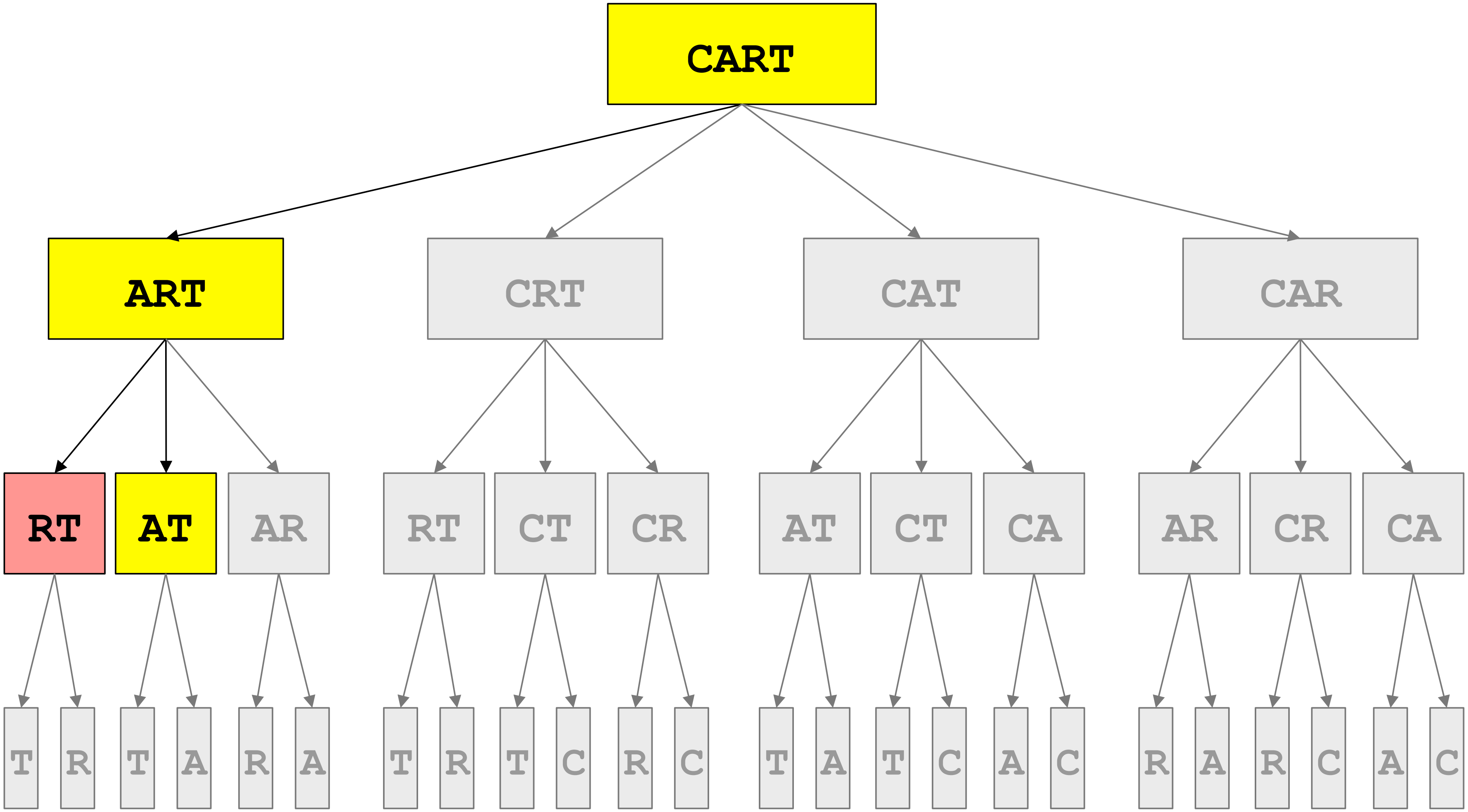
Recursive Exploration



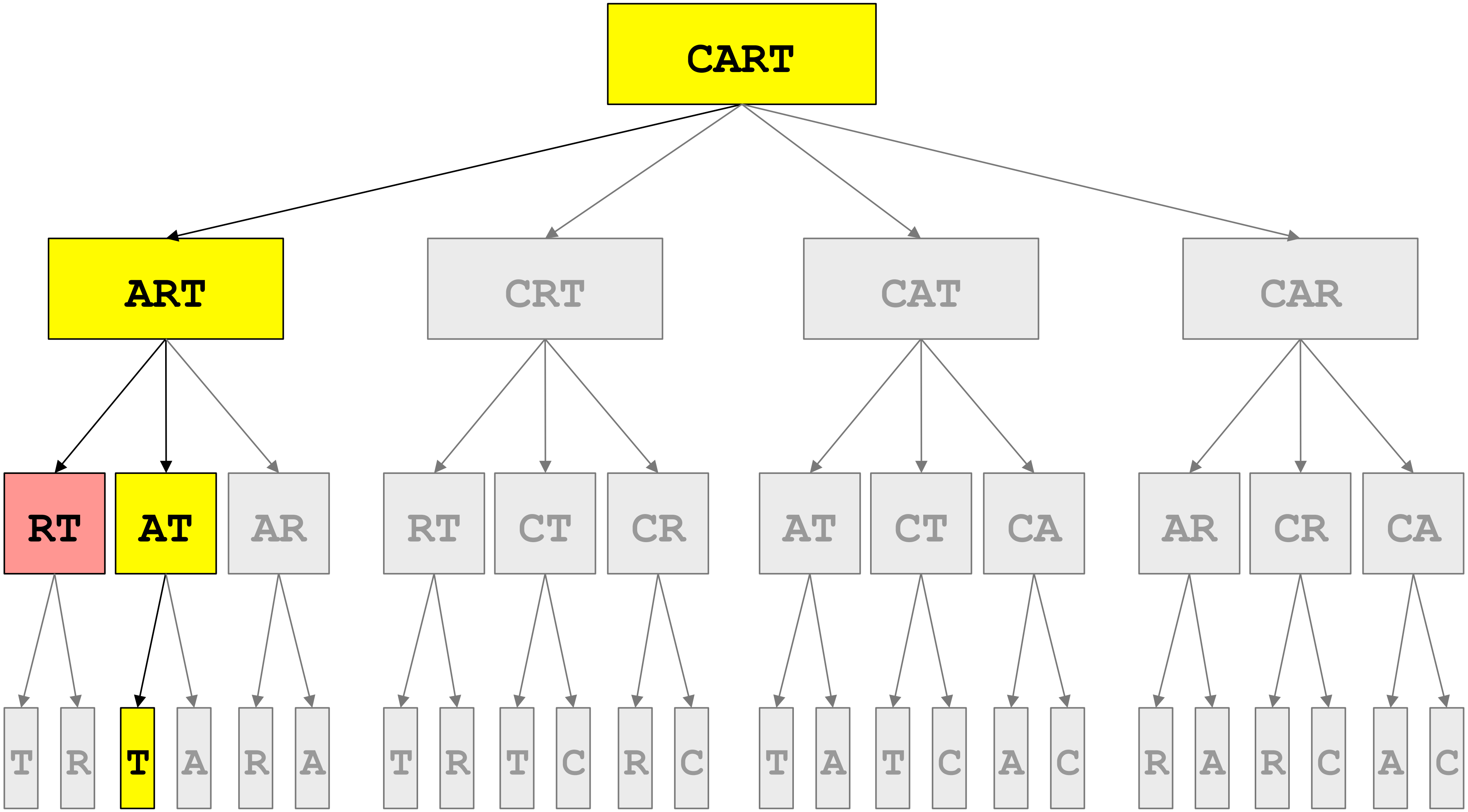
Recursive Exploration



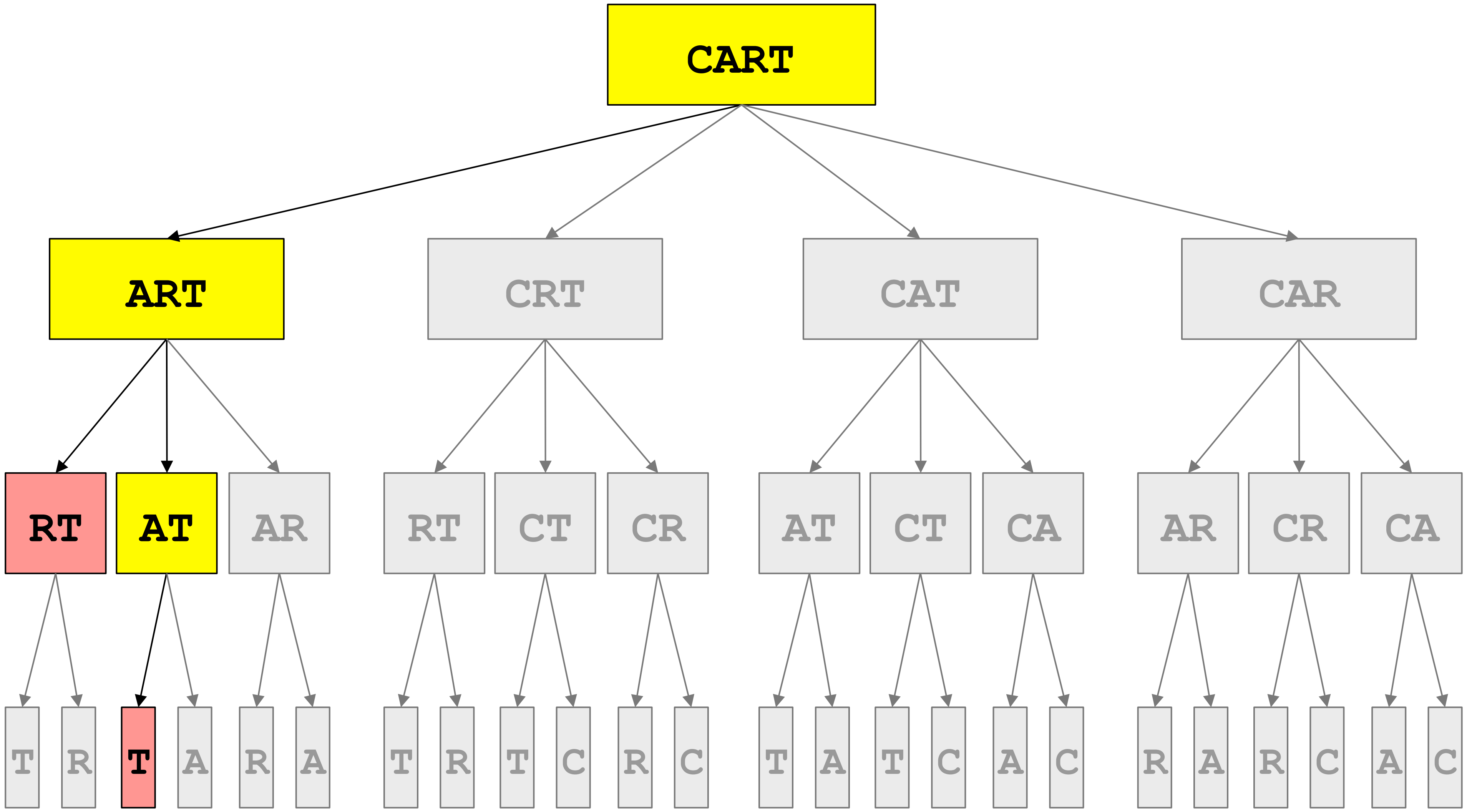
Recursive Exploration



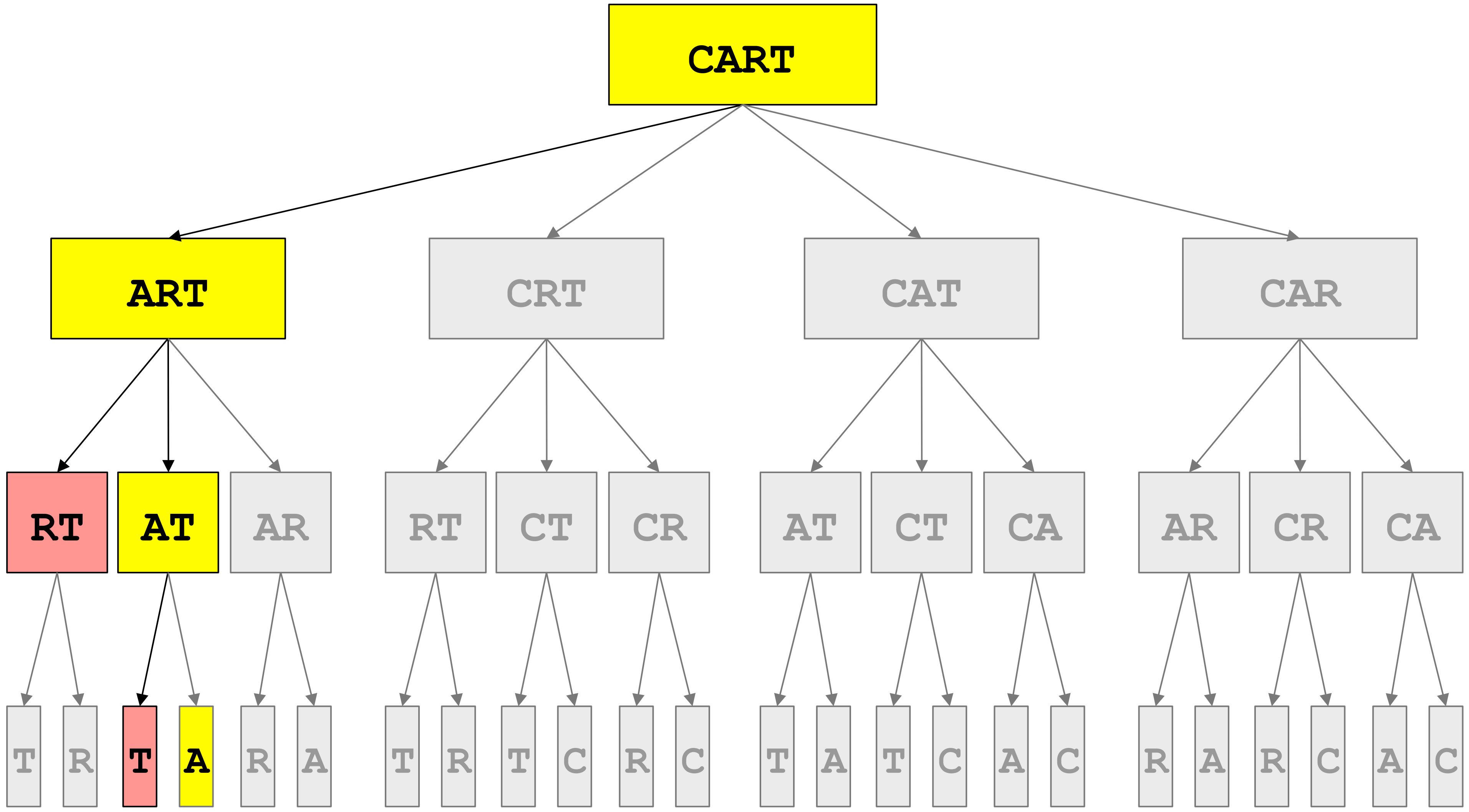
Recursive Exploration



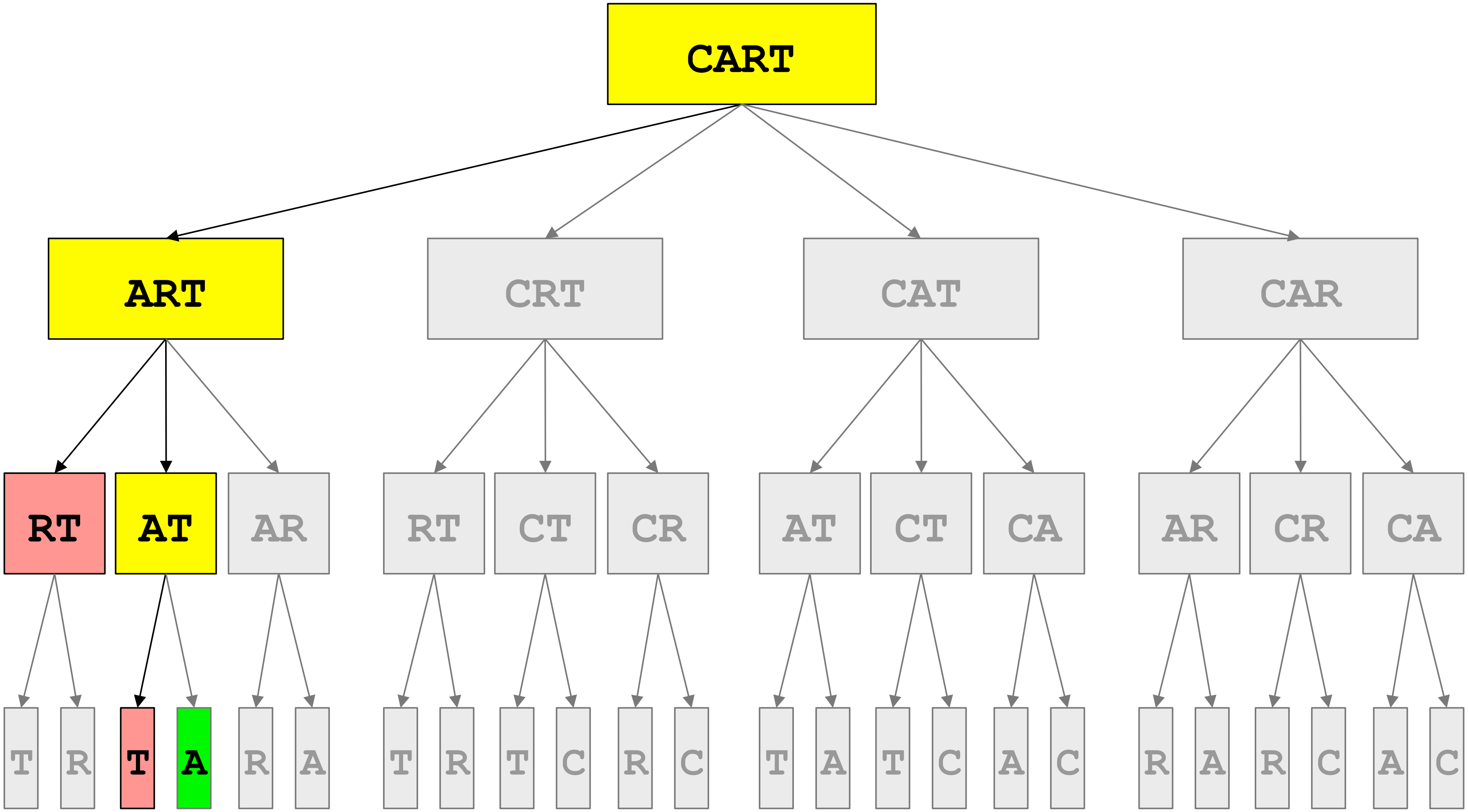
Recursive Exploration



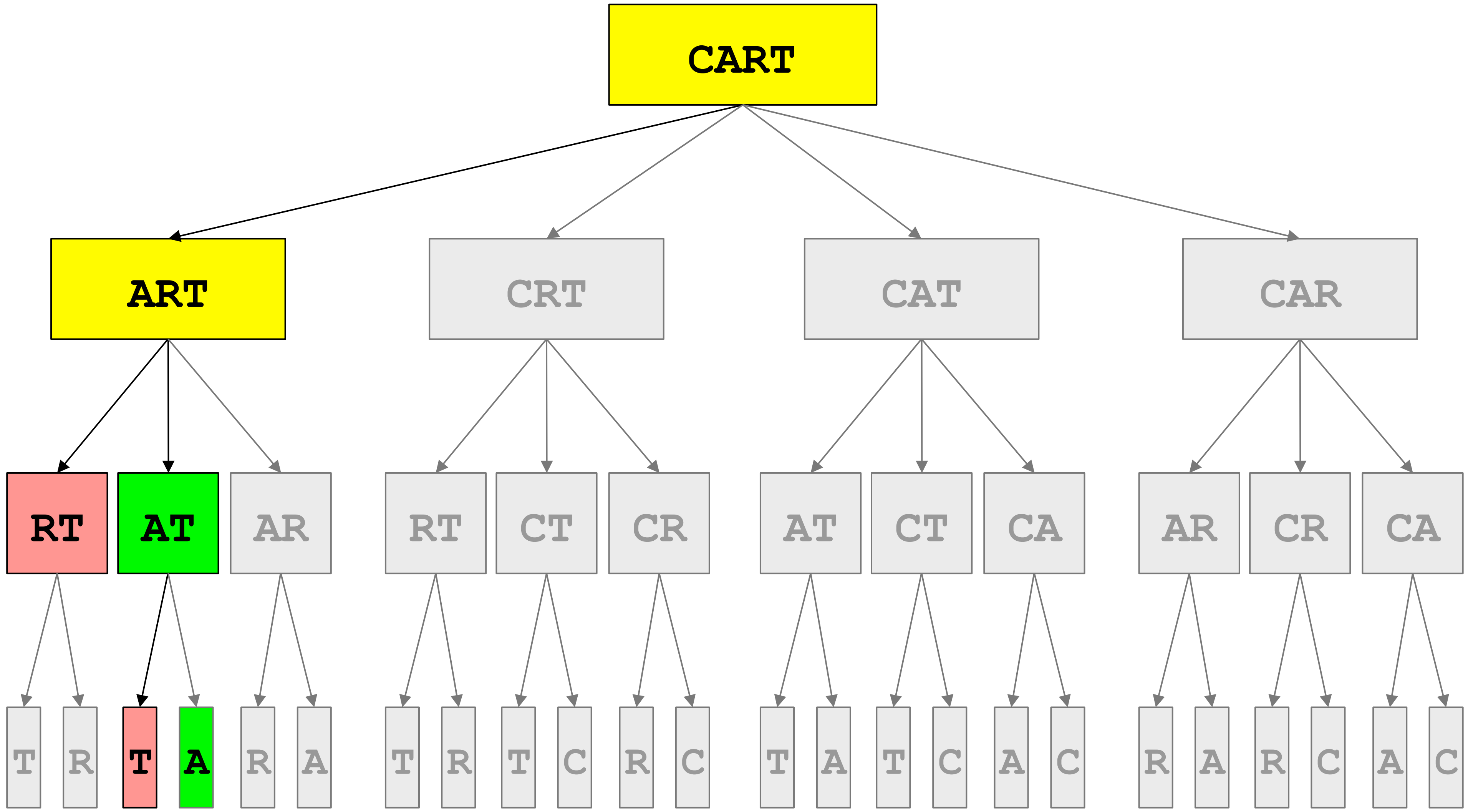
Recursive Exploration



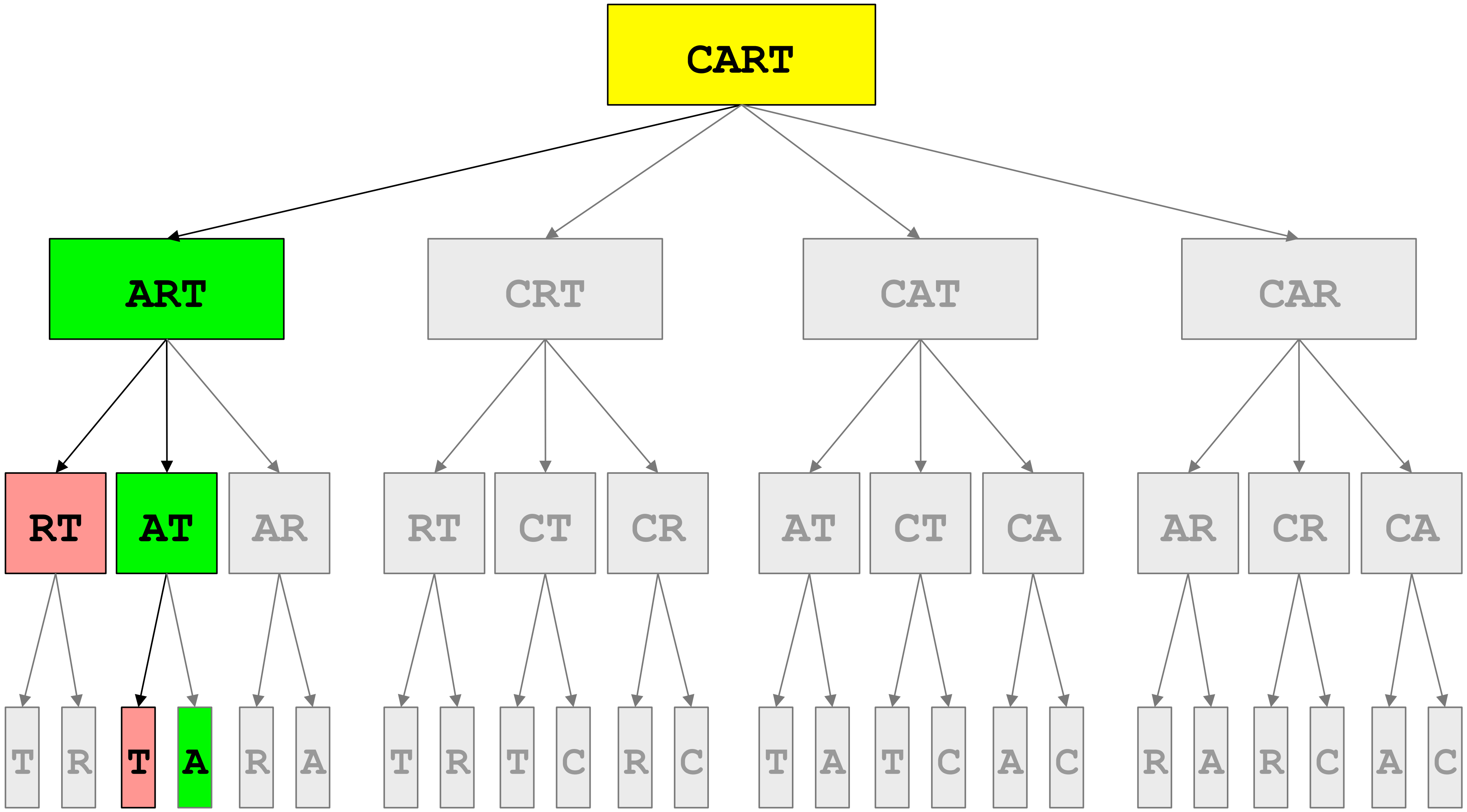
Recursive Exploration



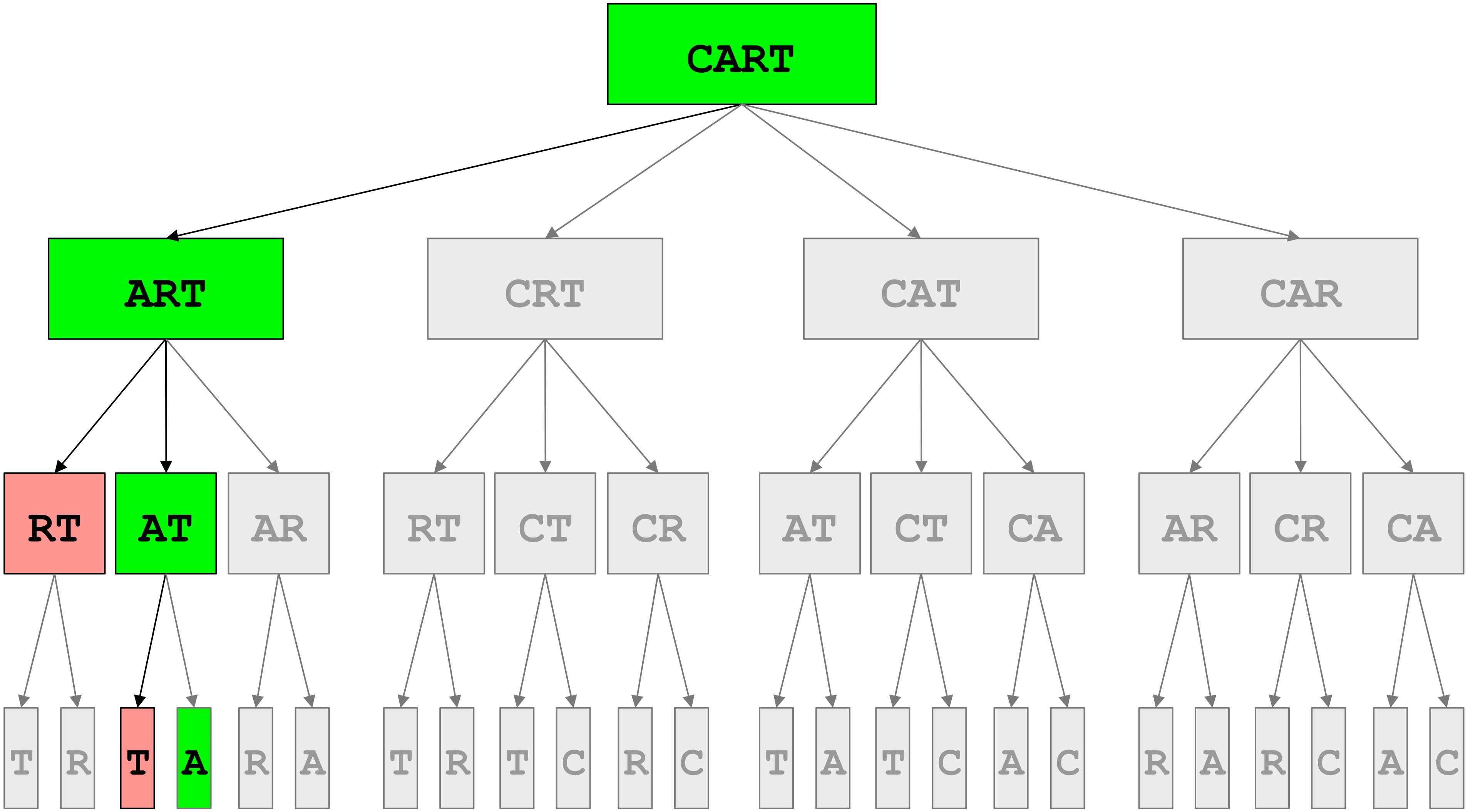
Recursive Exploration



Recursive Exploration



Recursive Exploration



Only Need One Path

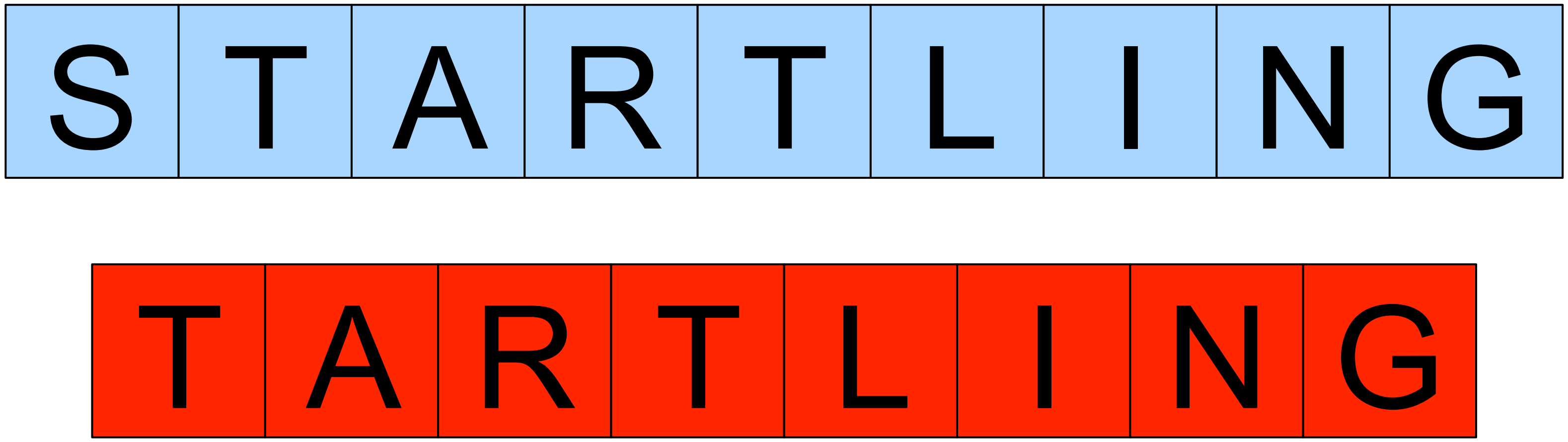
S	T	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

T	A	R	T	L	I	N	G

Only Need One Path



Only Need One Path

S	T	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	A	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	A	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	A	R	L	I	N	G

Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	A	R	L	I	N	G

Sources: http://www.rubyinside.com/wp-content/uploads/2008/07/starling.jpg an
http://2.bp.blogspot.com/-hCzVof1xqWo/TvaFXmpJAmI/AAAAAAAAIdk/QzUg1sKHDnA/s1600/1+Common_starling_in_london+paul+lomax+wiki+commons.jpg

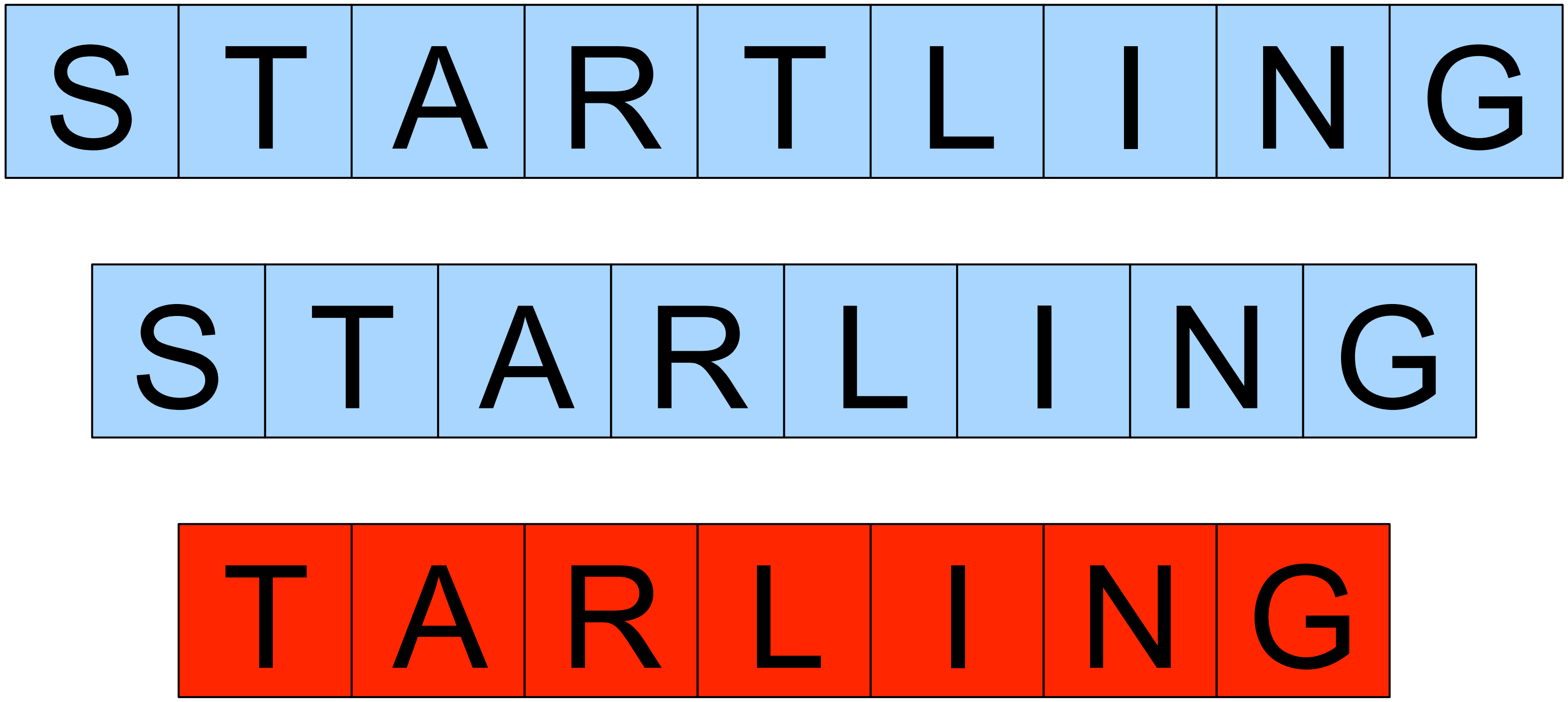
Only Need One Path

S	T	A	R	T	L	I	N	G

S	T	A	R	L	I	N	G

T	A	R	L	I	N	G

Only Need One Path

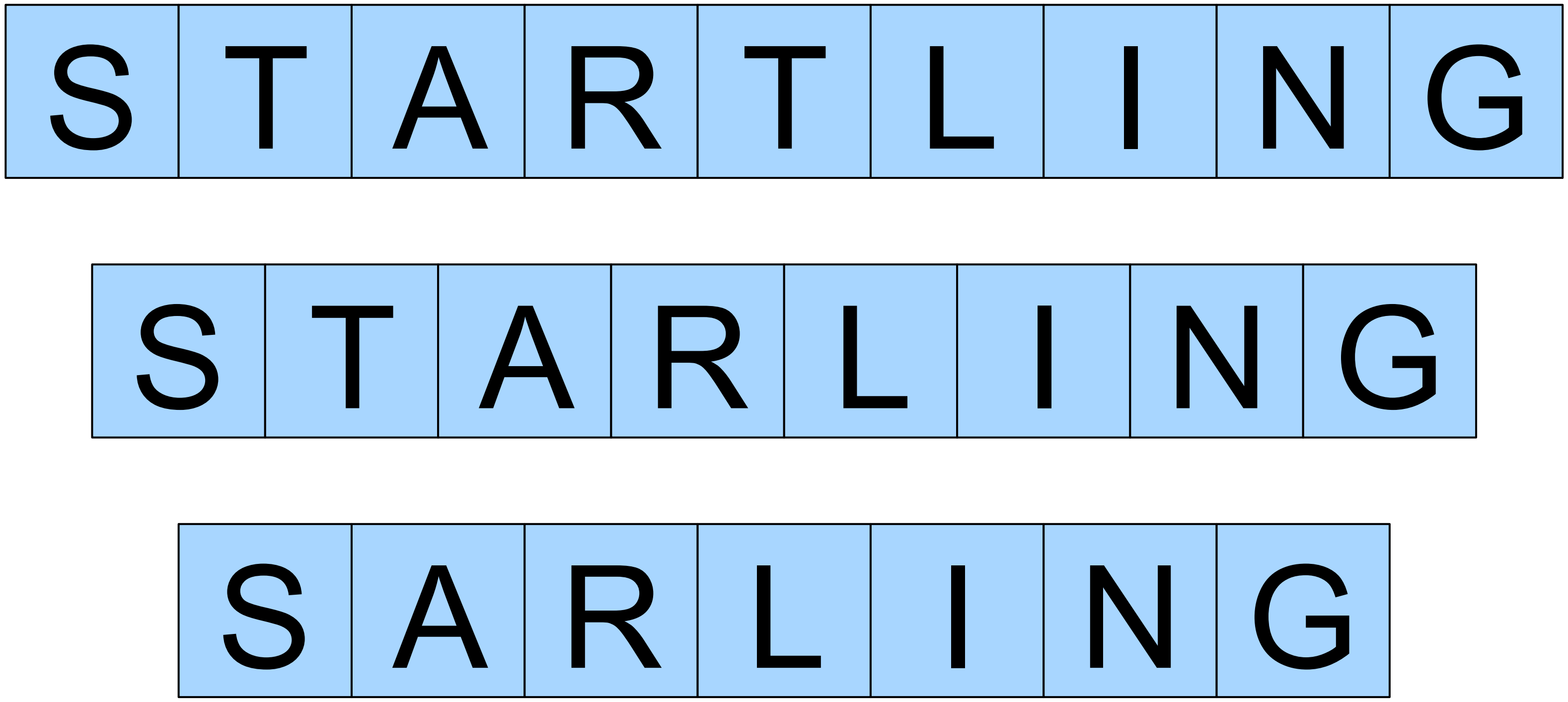


Only Need One Path

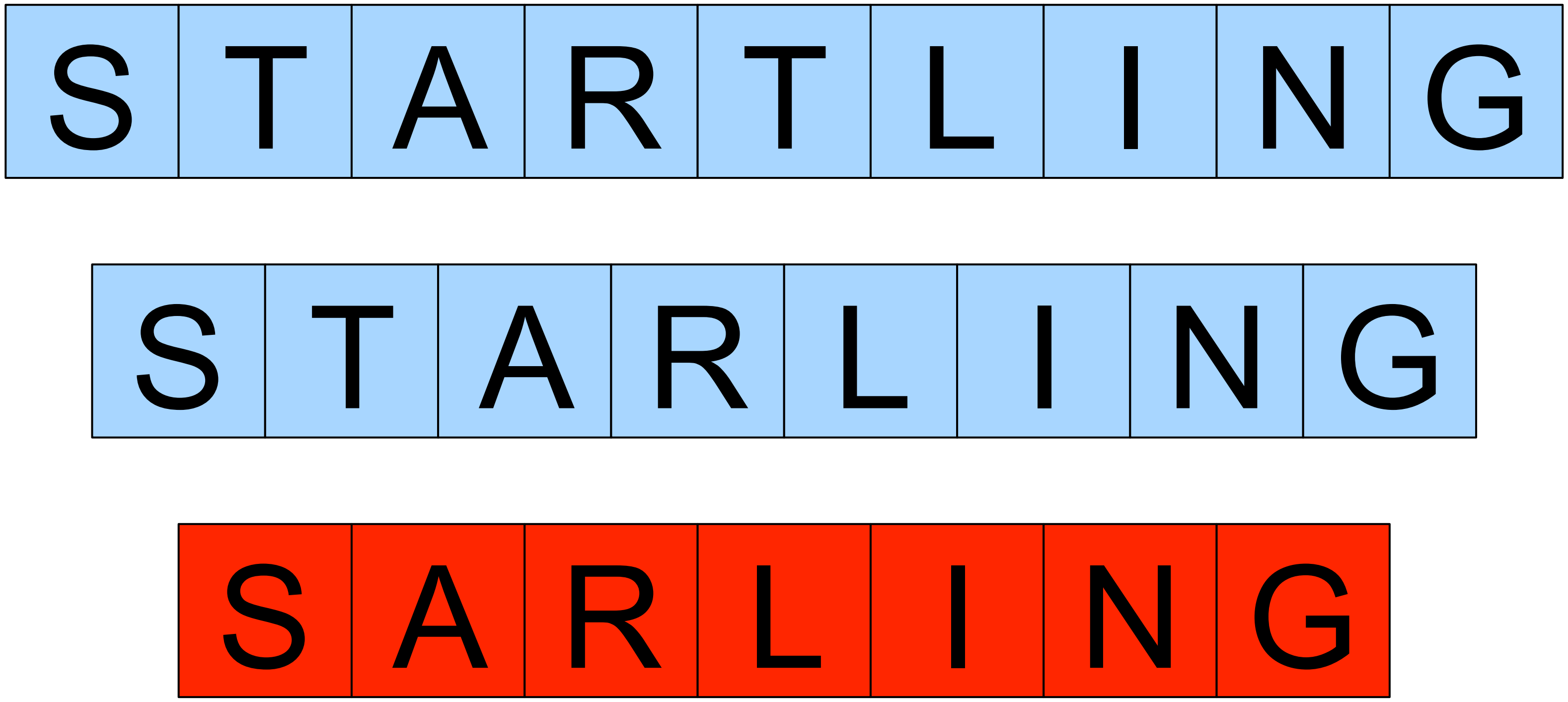
S	T	A	R	T	L	I	N	G

S	T	A	R	L	I	N	G

Only Need One Path



Only Need One Path



Decision Tree Search

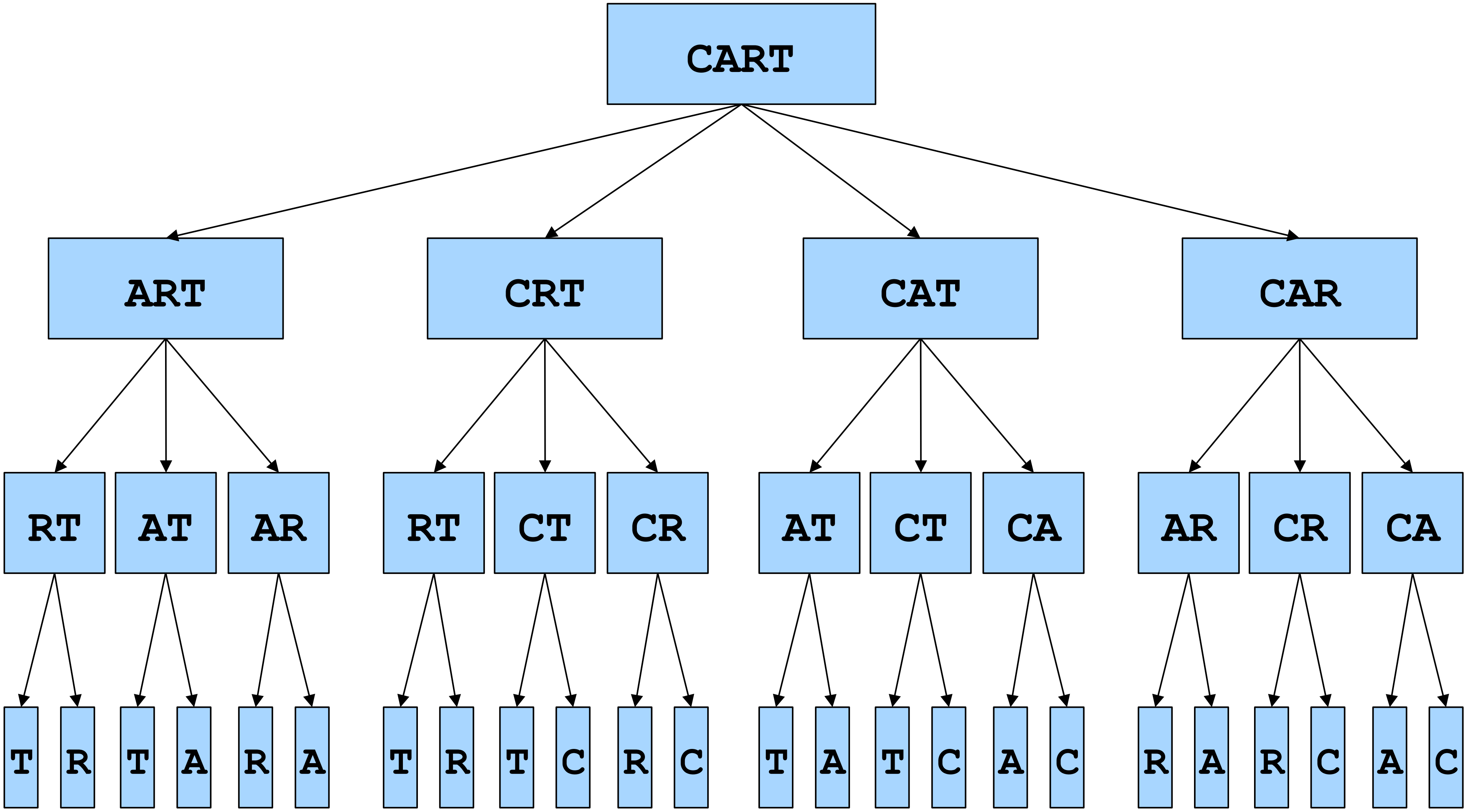
```
bool reducible(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(reducible(lex, copy)){
               return true;
            }
         }
      }
      return false;
   }
}
```

Ur Doin it Rong!

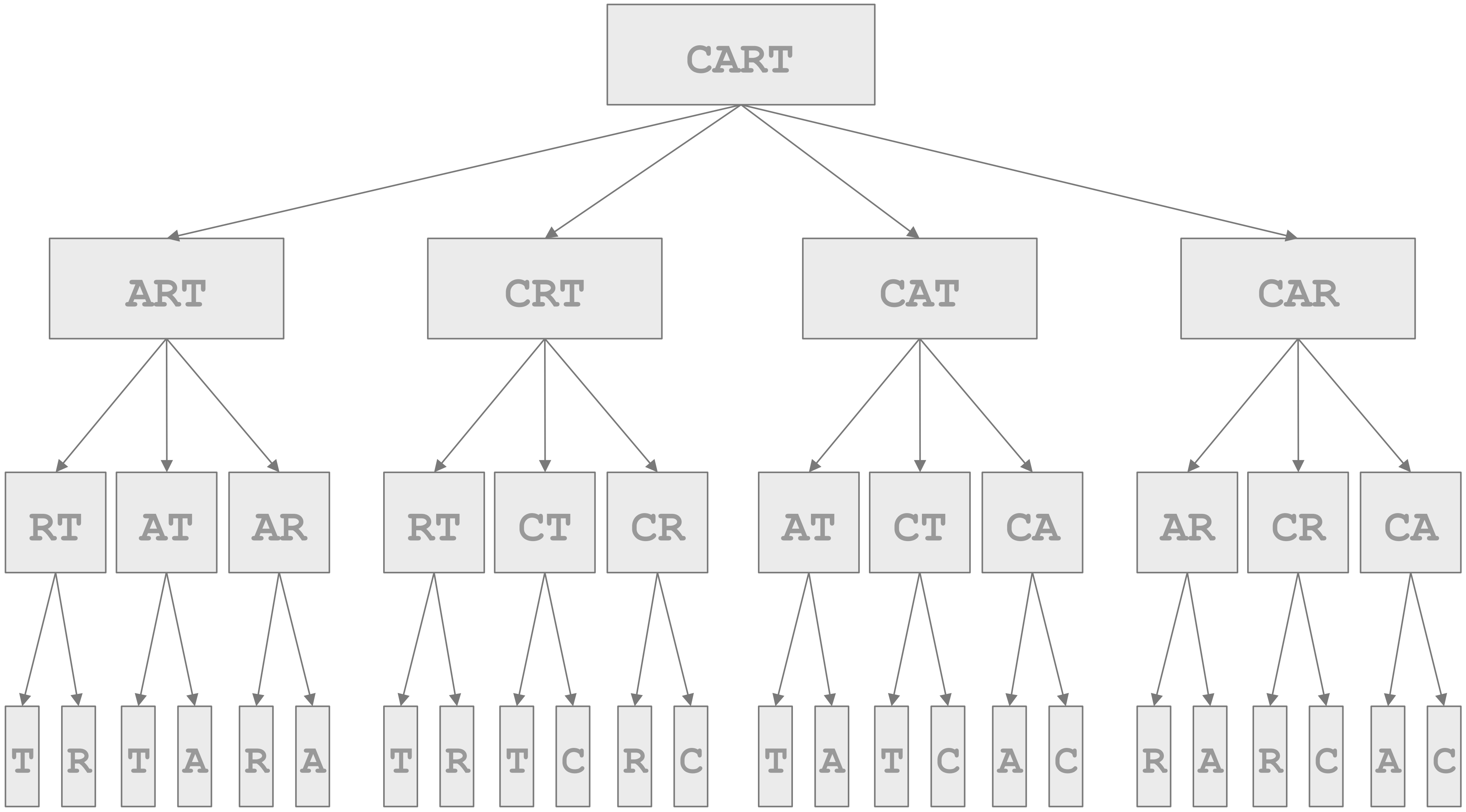
```
bool reducible(Lexicon & lex, string word) {
   if(word.length()==1 && lex.contains(word)){
      return true;
   } else {
      for(int i=0; i < word.length(); i++) {
         string copy = word;
         copy.erase(i, 1);
         if(lex.contains(copy)){
            if(!reducible(lex, copy)){
               return false;
            }
         }
      }
      return true;
   }
}
```

Note how the true became a false

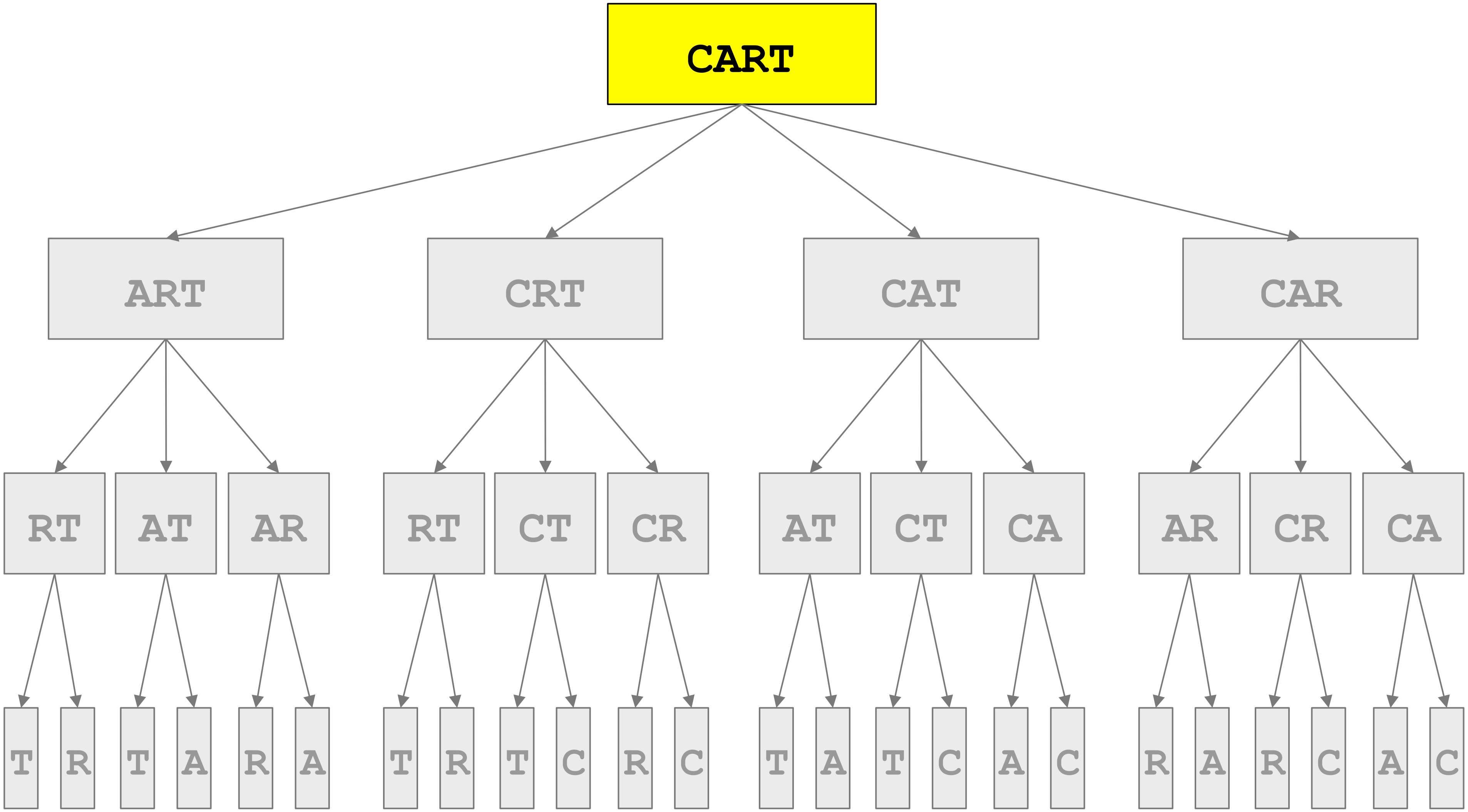
Ur Doin It Rong!



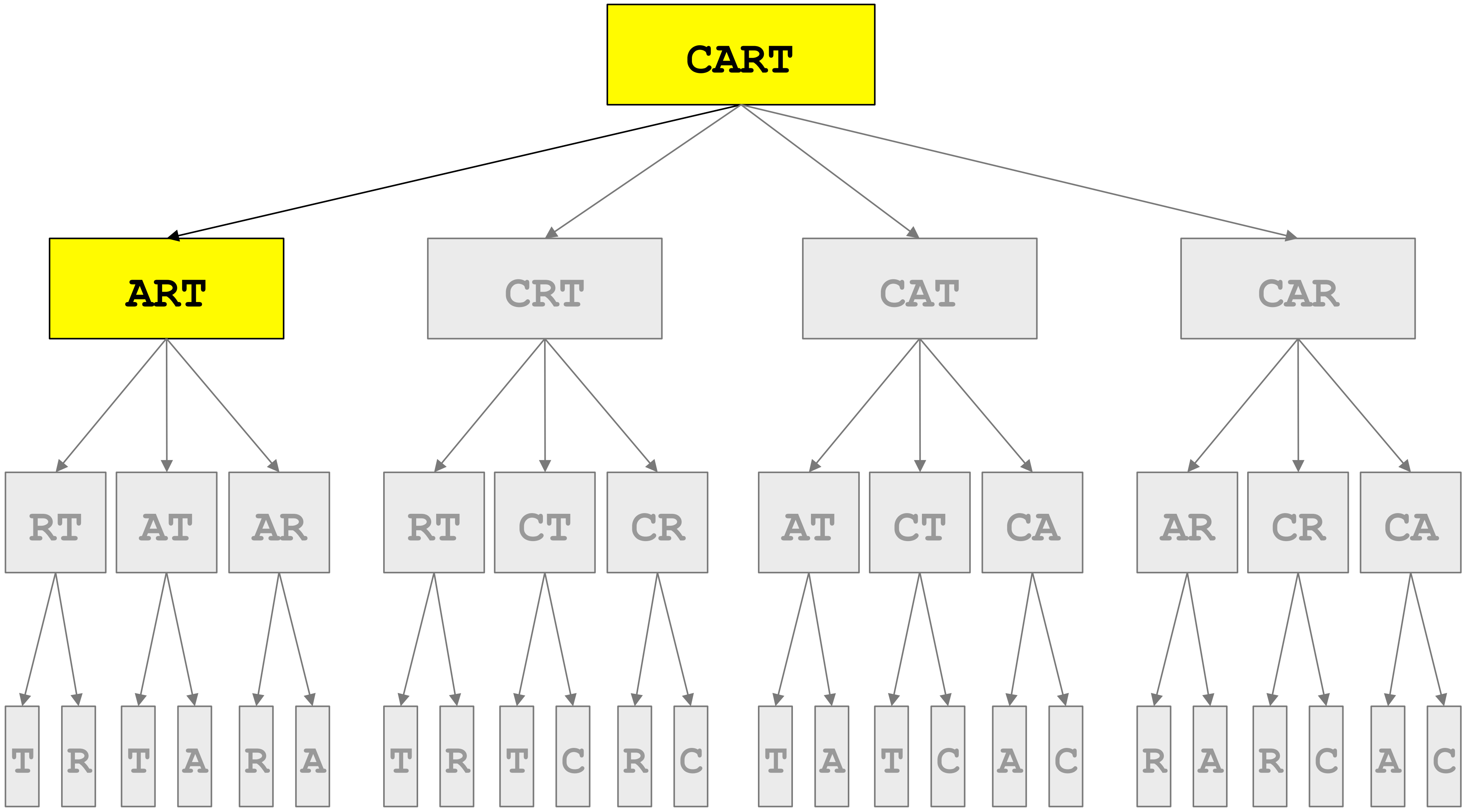
Ur Doin It Rong!



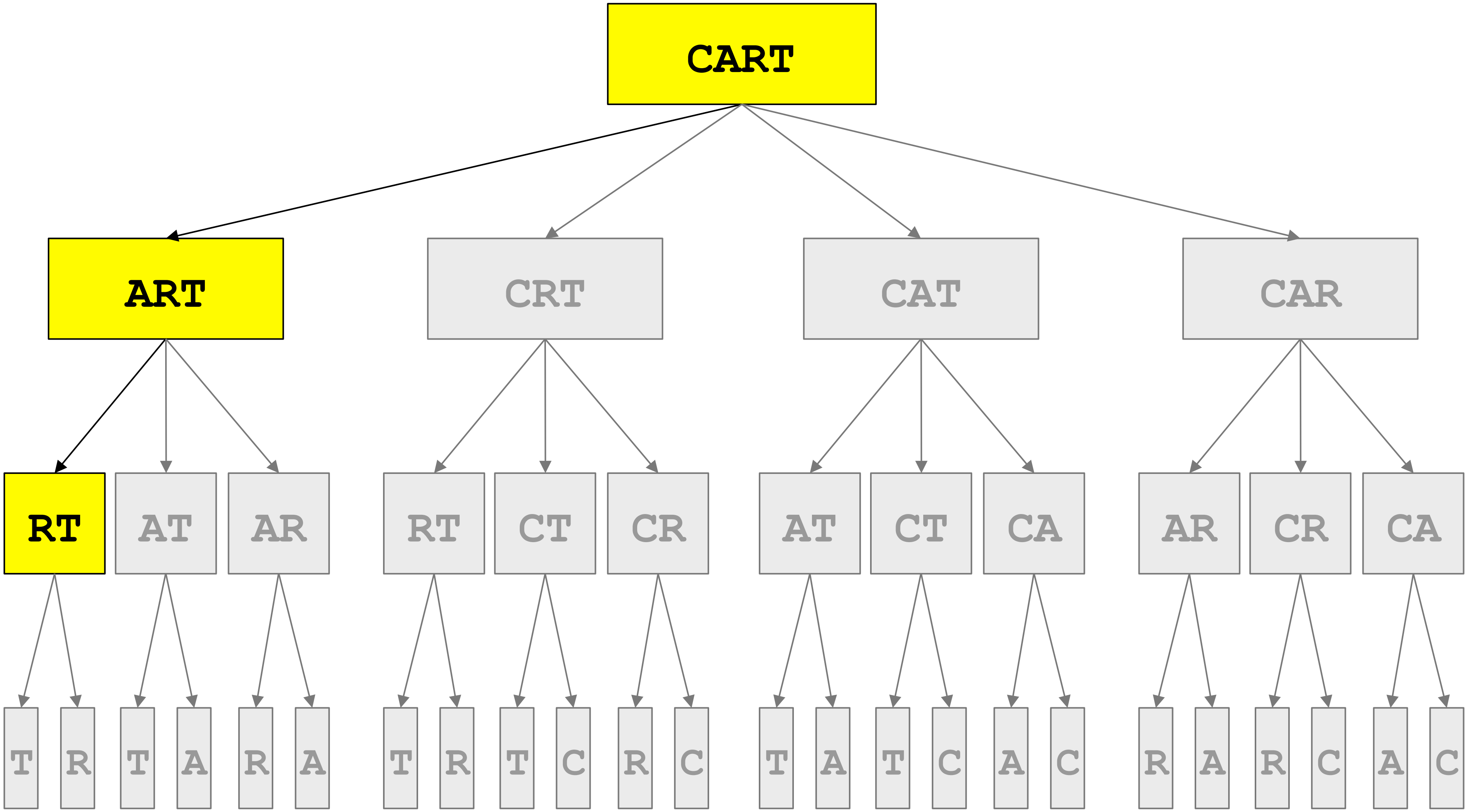
Ur Doin It Rong!



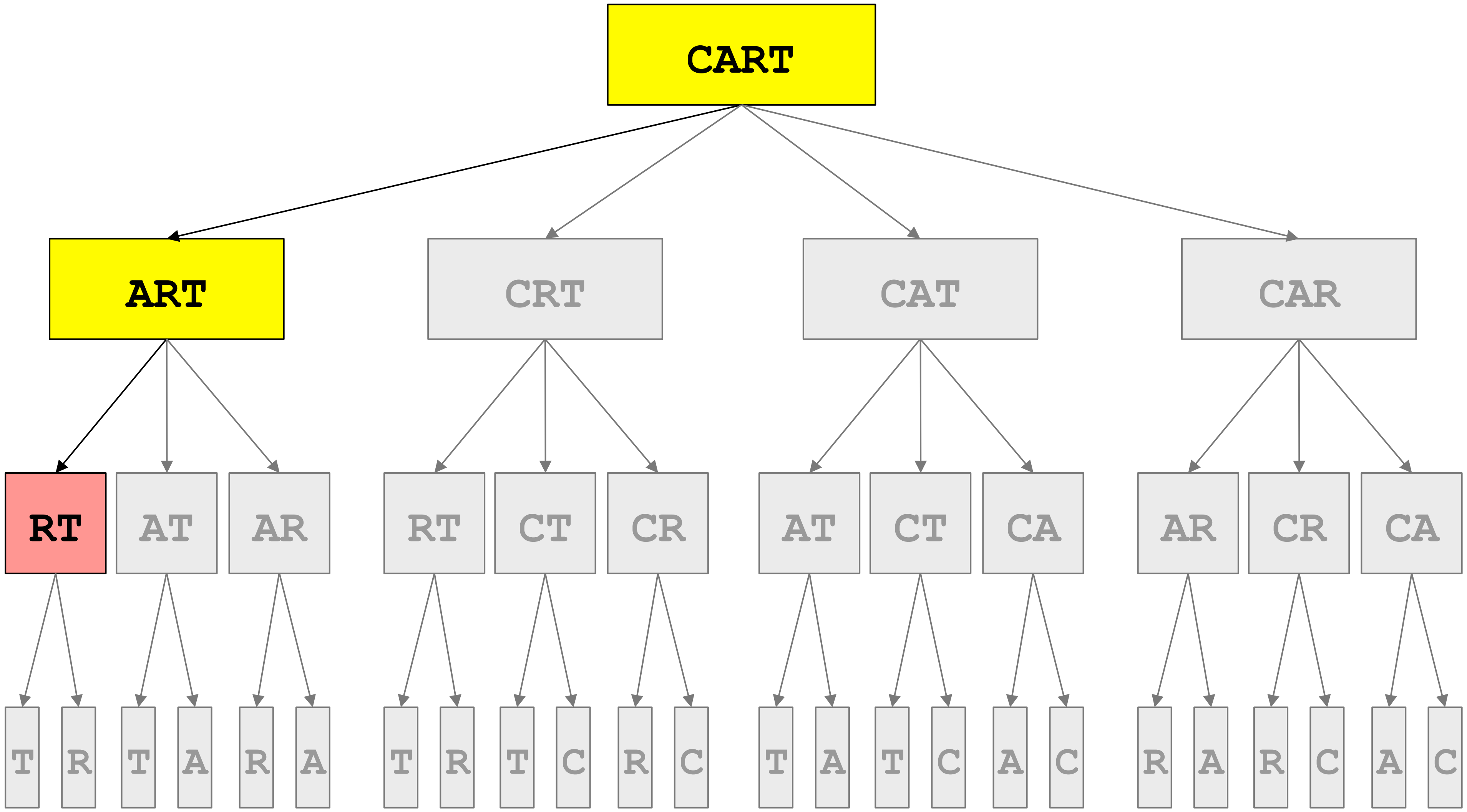
Ur Doin It Rong!



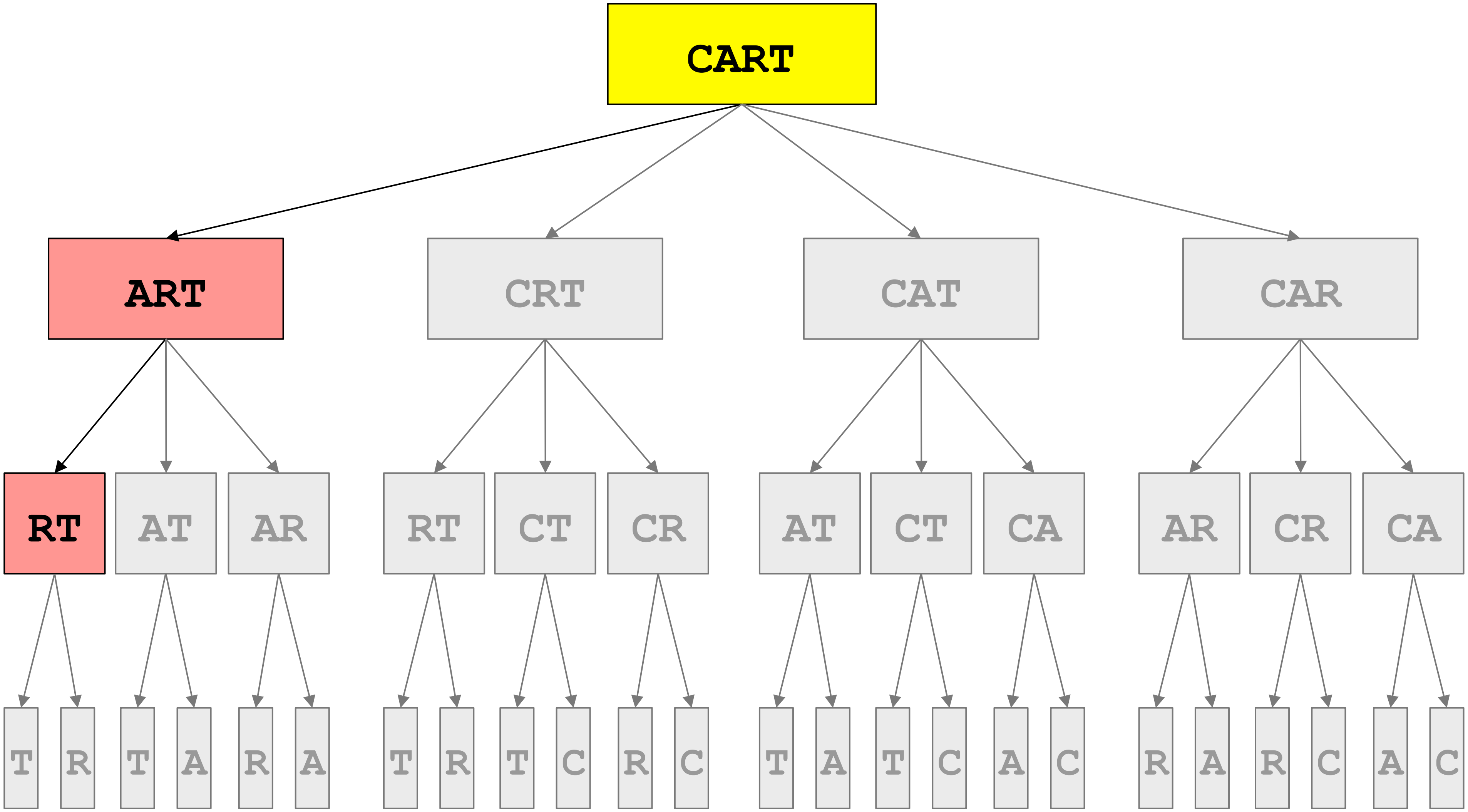
Ur Doin It Rong!



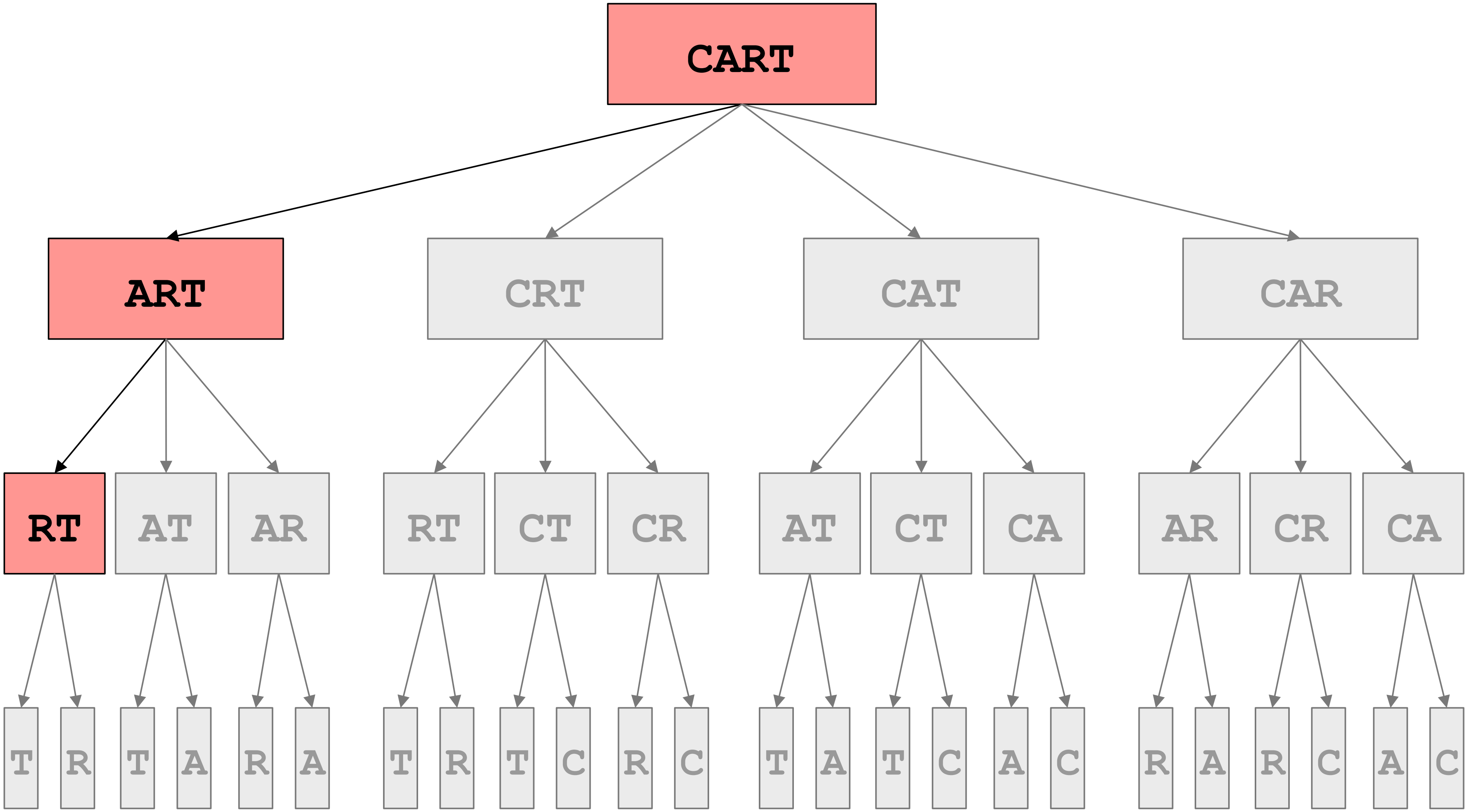
Ur Doin It Rong!



Ur Doin It Rong!



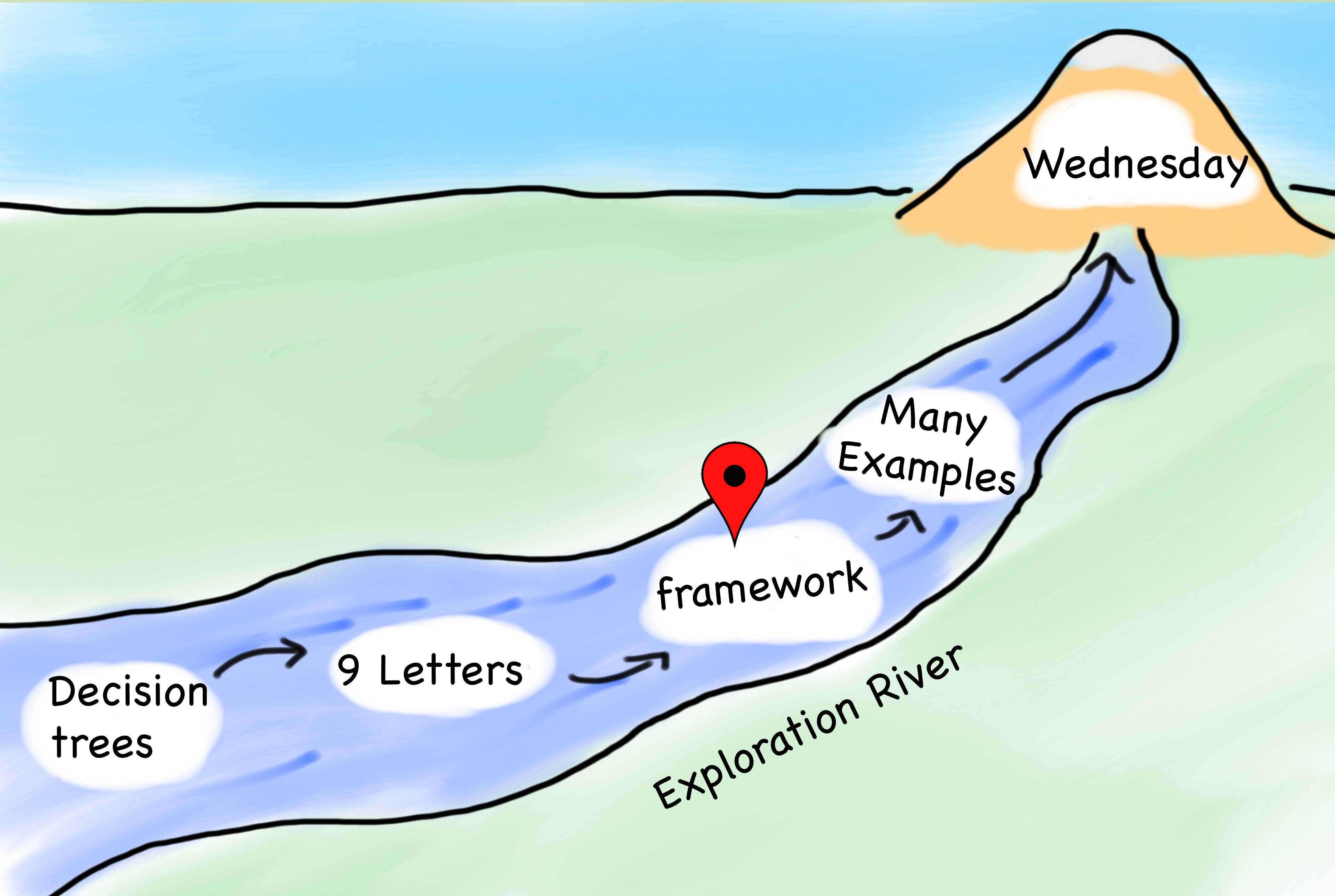
Ur Doin It Rong!



Today's Route

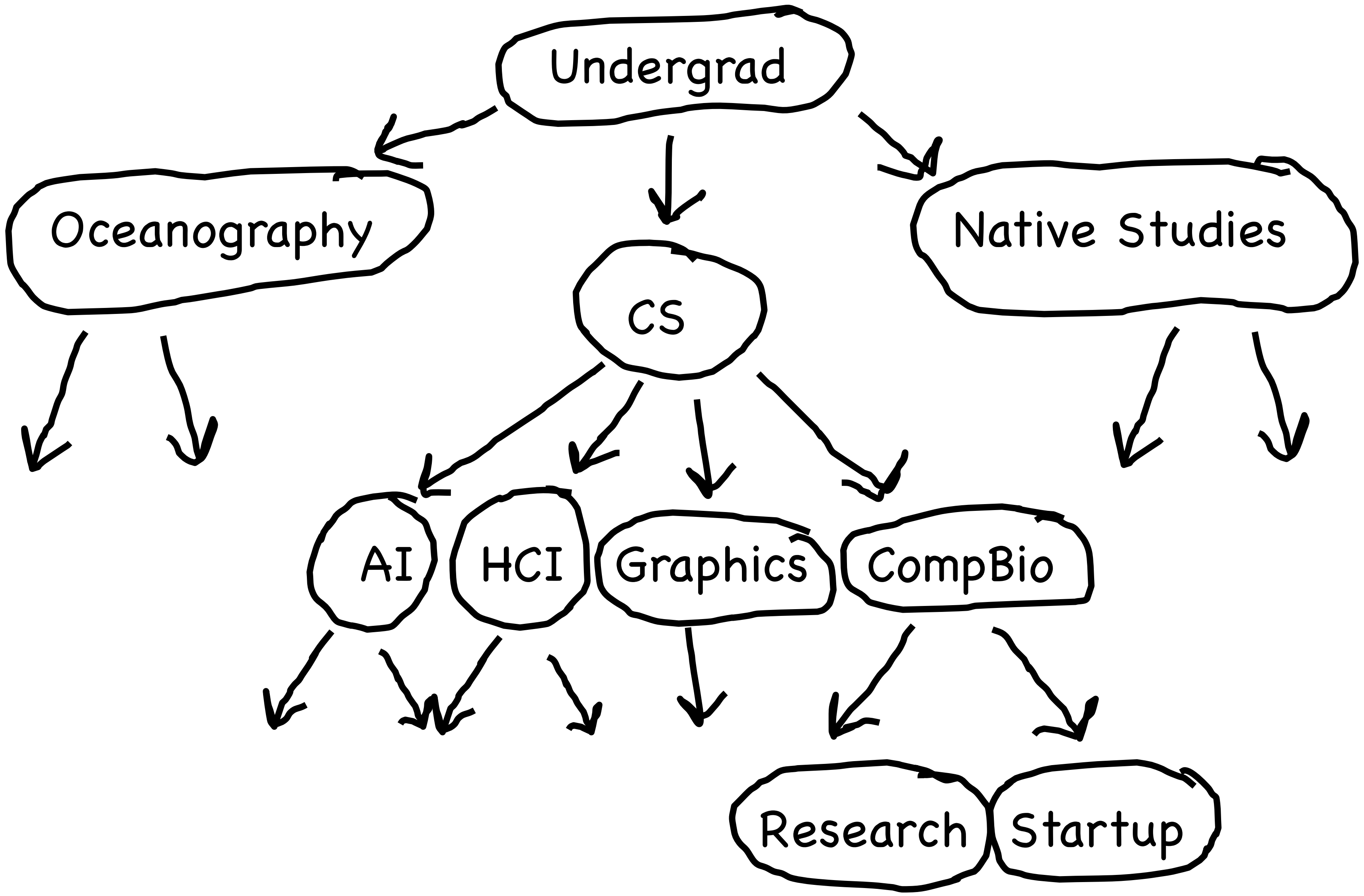
Wednesday
Many Examples
framework
9 Letters
Decision trees
Exploration River

Today's Route



Many problems can be seen as decision trees

Decision Tree



Art of seeing decision trees

Templates for working with a decision tree.

How to Formulate as Decision Tree

Four questions!

How to Formulate as Decision Tree

1. How do you represent a current state?

2. How do you calculate legal moves?

3. How do you generate next states given move?

4. How do you know if you should stop recursing?

Generating Permutations

All permutations of **`"abcd"`**

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Generating Permutations

x_1	x_2	x_3	x_4

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

How to Formulate as Decision Tree

1. How do you represent a current state?

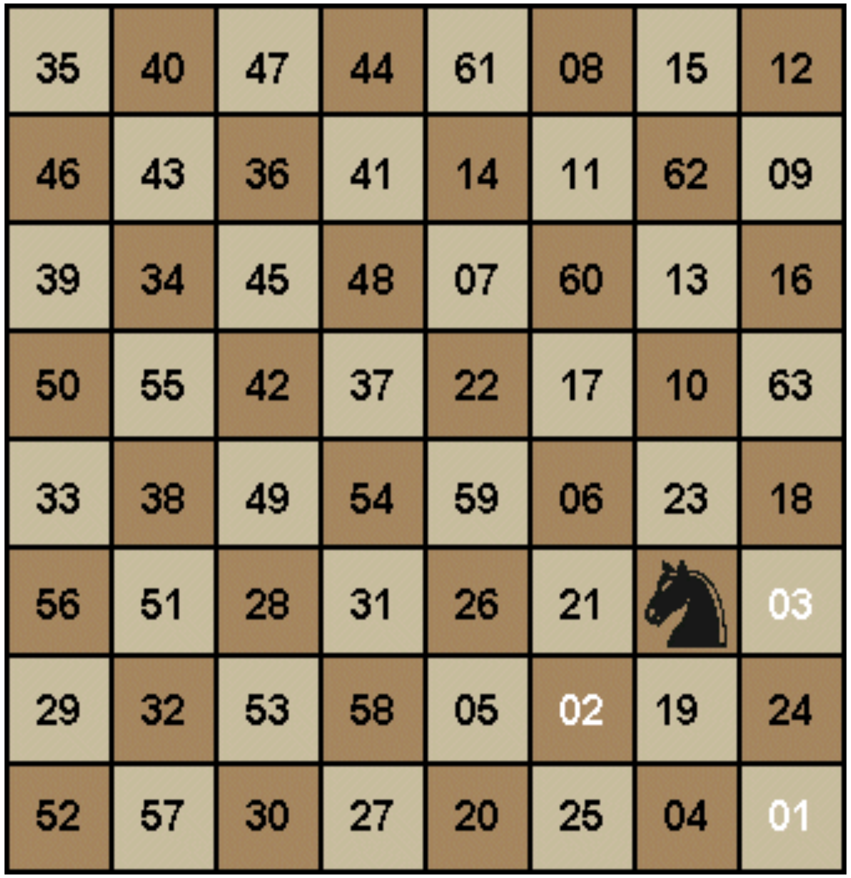
2. How do you calculate legal moves?

3. How do you generate next states given move?

4. How do you know if you should stop recursing?

Generating Permutations Tree

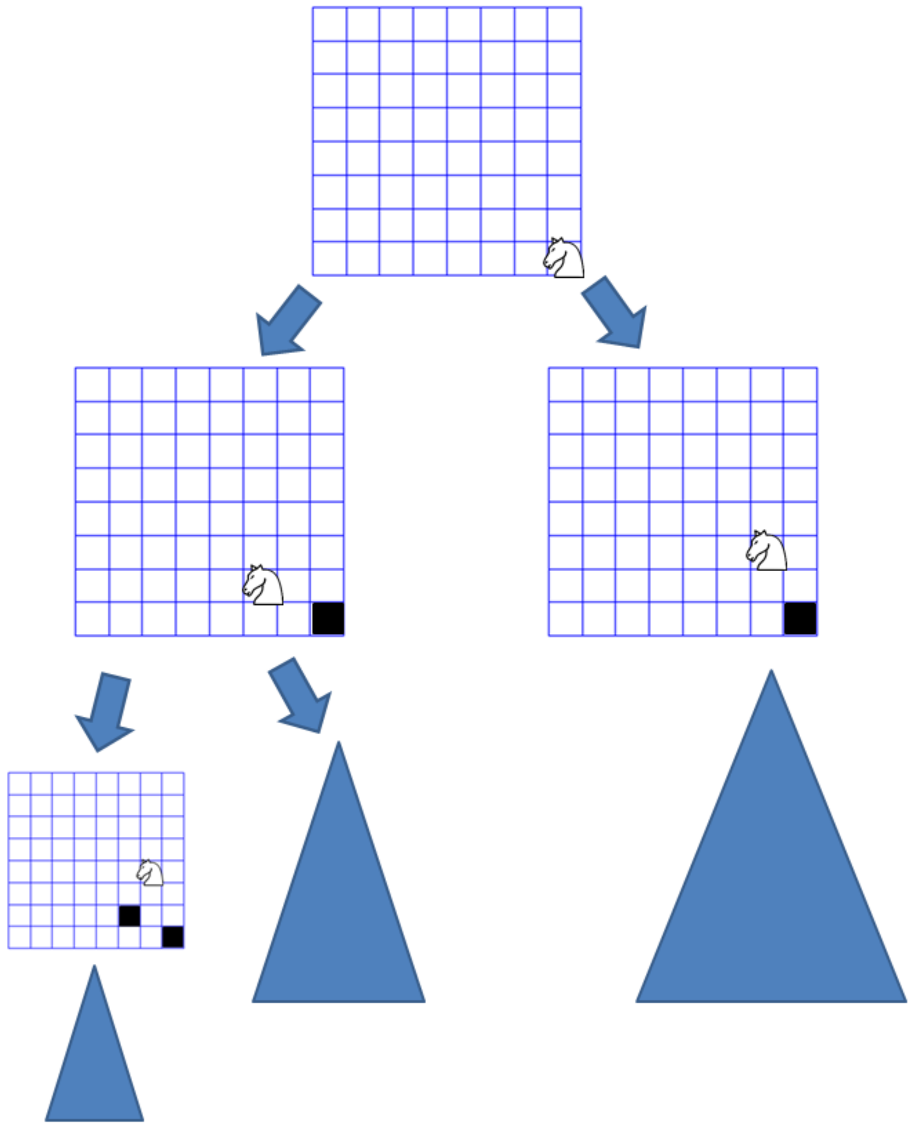
Knights Tour



35	40	47	44	61	08	15	12
46	43	36	41	14	11	62	09
39	34	45	48	07	60	13	16
50	55	42	37	22	17	10	63
33	38	49	54	59	06	23	18
56	51	28	31	26	21	♞	03
29	32	53	58	05	02	19	24
52	57	30	27	20	25	04	01

[Knight's Tour Demo]

Knights Tour



DNA Alignment

Today's Goal

1. Introduction to decision trees.