Thinking Recursively

Part IV

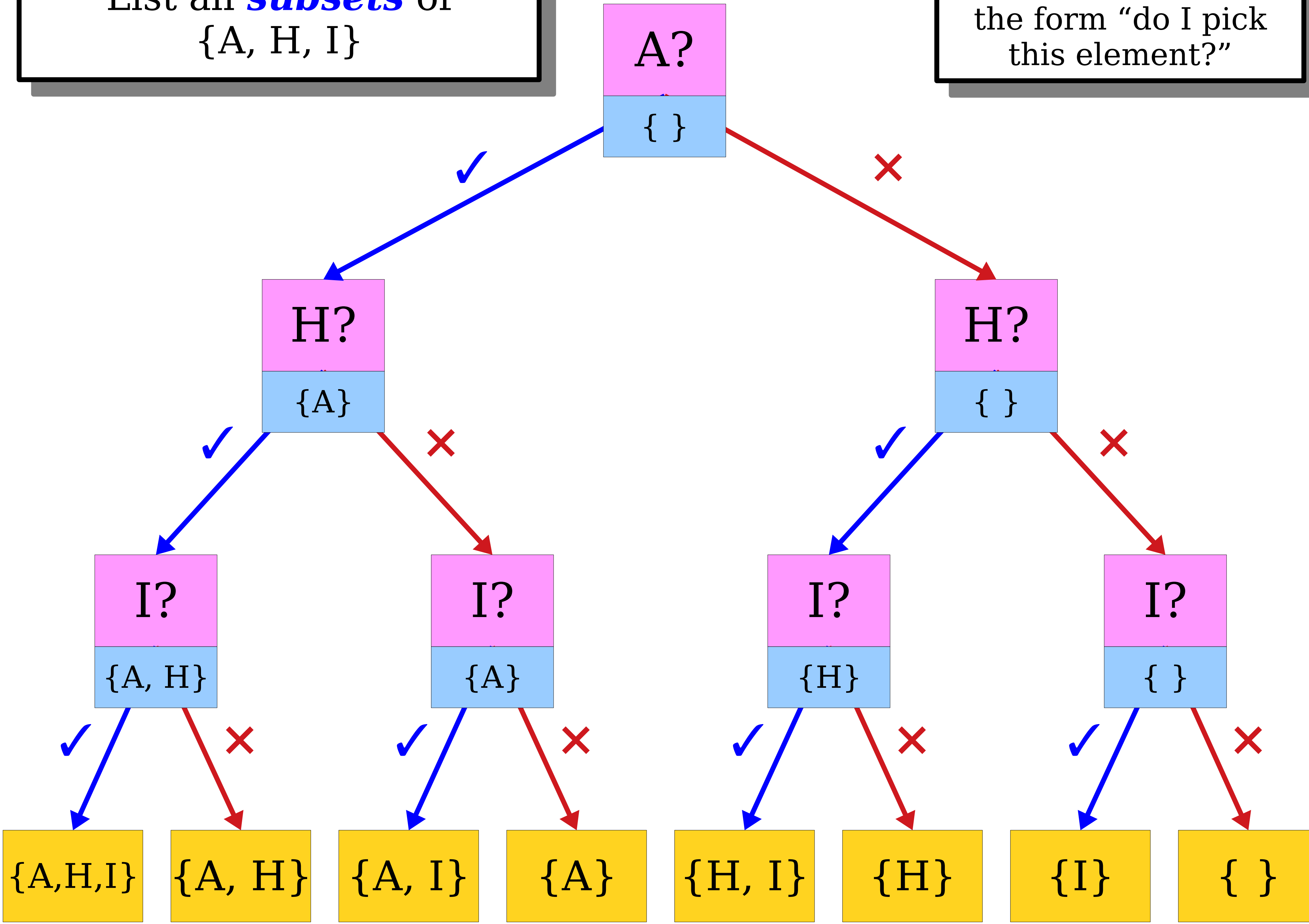
Outline for Today

- ***Recap From Last Time***
  - Where are we, again?
- ***Enumerating Combinations***
  - Addressing some points from last time.
- ***Shrinkable Words***
  - A little word puzzle!

Recap from Last Time

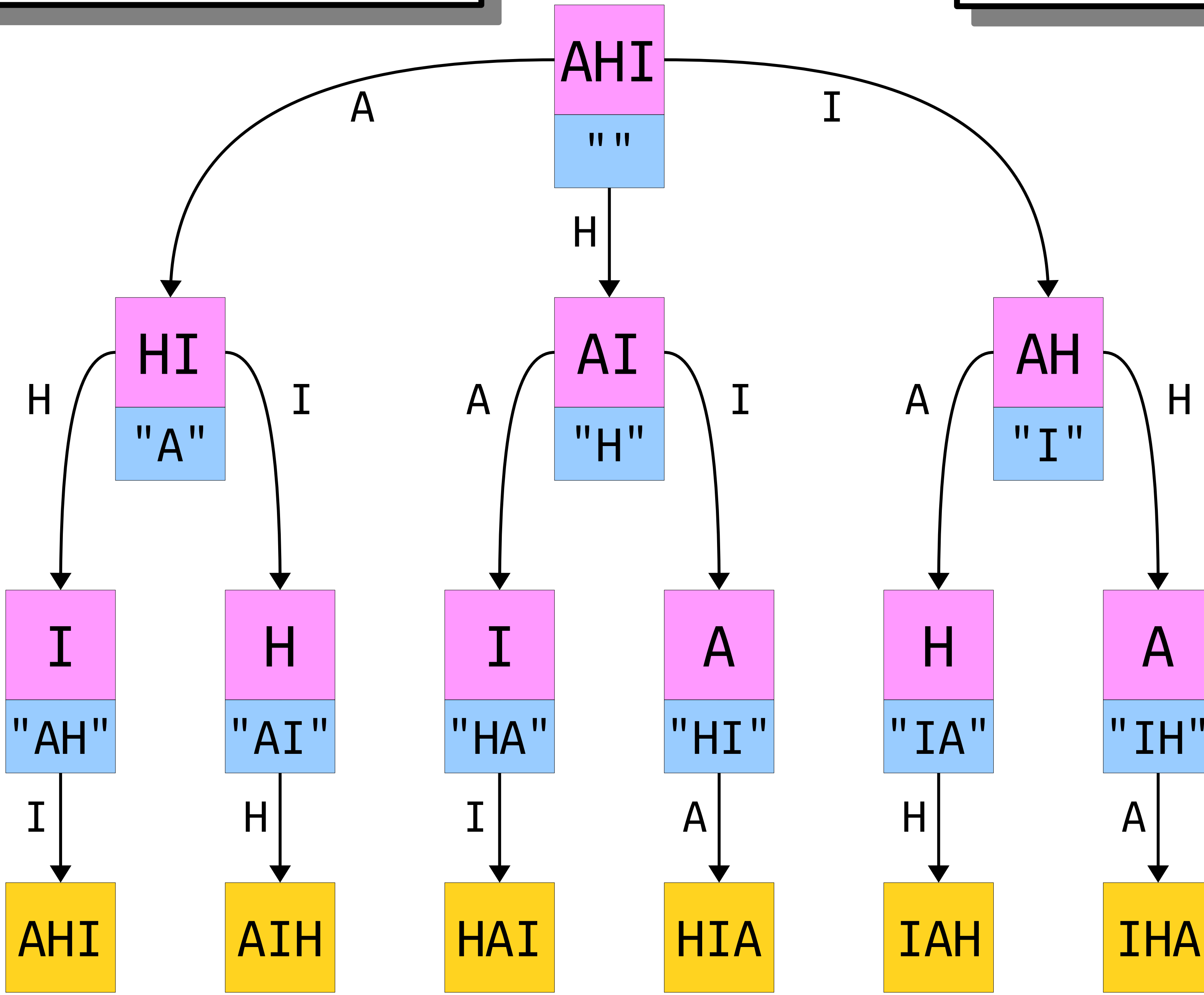
List all ***subsets*** of {A, H, I}

Each decision is of the form “do I pick this element?”



List all ***permutations*** of {A, H, I}

Each decision is of the form “what do I pick next?”



Base Case: No decisions remain.

Decisions yet to be made

Decisions already made

```
void exploreRec(decisions remaining,
                decisions already made) {
  if (no decisions remain) {
    process decisions made;
  } else {
    for (each possible next choice) {
      exploreRec(all remaining decisions,
                 decisions made + that choice);
    }
  }
}
```

Recursive Case: Try all options for the next decision.

```
void exploreAllTheThings(initial state) {
  exploreRec(initial state, no decisions made);
}
```

New Stuff!

Enumerating Combinations

You need at least five US Supreme Court justices to agree to set a precedent.

What are all the ways you can pick five justices off the US Supreme Court?

Generating Combinations

- Suppose that we want to find every way to choose exactly ***one*** element from a set.
- We could do something like this:

```
for (int x: mySet) {
    cout << x << endl;
}
```

Generating Combinations

- Suppose that we want to find every way to choose exactly ***two*** elements from a set.
- We could do something like this:

```
for (int x: mySet) {
  for (int y: mySet) {
    if (x != y) {
      cout << x << ", " << y << endl;
    }
  }
}
```

Generating Combinations

- Suppose that we want to find every way to choose exactly ***three*** elements from a set.
- We could do something like this:

```
for (int x: mySet) {
  for (int y: mySet) {
    for (int z: mySet) {
      if (x != y && x != z && y != z) {
         cout << x << ", " << y << ", " << z << endl;
      }
    }
  }
}
```

Generating Combinations

- If we know how many elements we want in advance, we can always just nest a whole bunch of loops.
- But what if we don't know in advance?
- Or we *do* know in advance, but it’s a large number and we don’t want to type until our fingers bleed?

```
combinationsOf(const Set<string>& elems,
               int numToPick);
```

Given this set of elements to pick from…

```
combinationsOf(const Set<string>& elems,
               int numToPick);
```

Given this set of elements to pick from…

```
combinationsOf(const Set<string>& elems,
               int numToPick);
```

… return all the ways to pick this many of them.

Given this set of elements to pick from…

```
??? combinationsOf(const Set<string>& elems,
                   int numToPick);
```

… return all the ways to pick this many of them.

Given this set of elements to pick from...

```
??? combinationsOf(const Set<string>& elems,
                   int numToPick);
```

... return all the ways to pick this many of them.

What should this function's return type be?

Take thirty seconds to formulate a hypothesis, but **don't post it in chat yet!**

Given this set of elements to pick from…

```
??? combinationsOf(const Set<string>& elems,
                   int numToPick);
```

What should this function's return type be?

Okay, now type your hypothesis into the chat.

… return all the ways to pick this many of them.

Implementing Combinations

Our Base Case

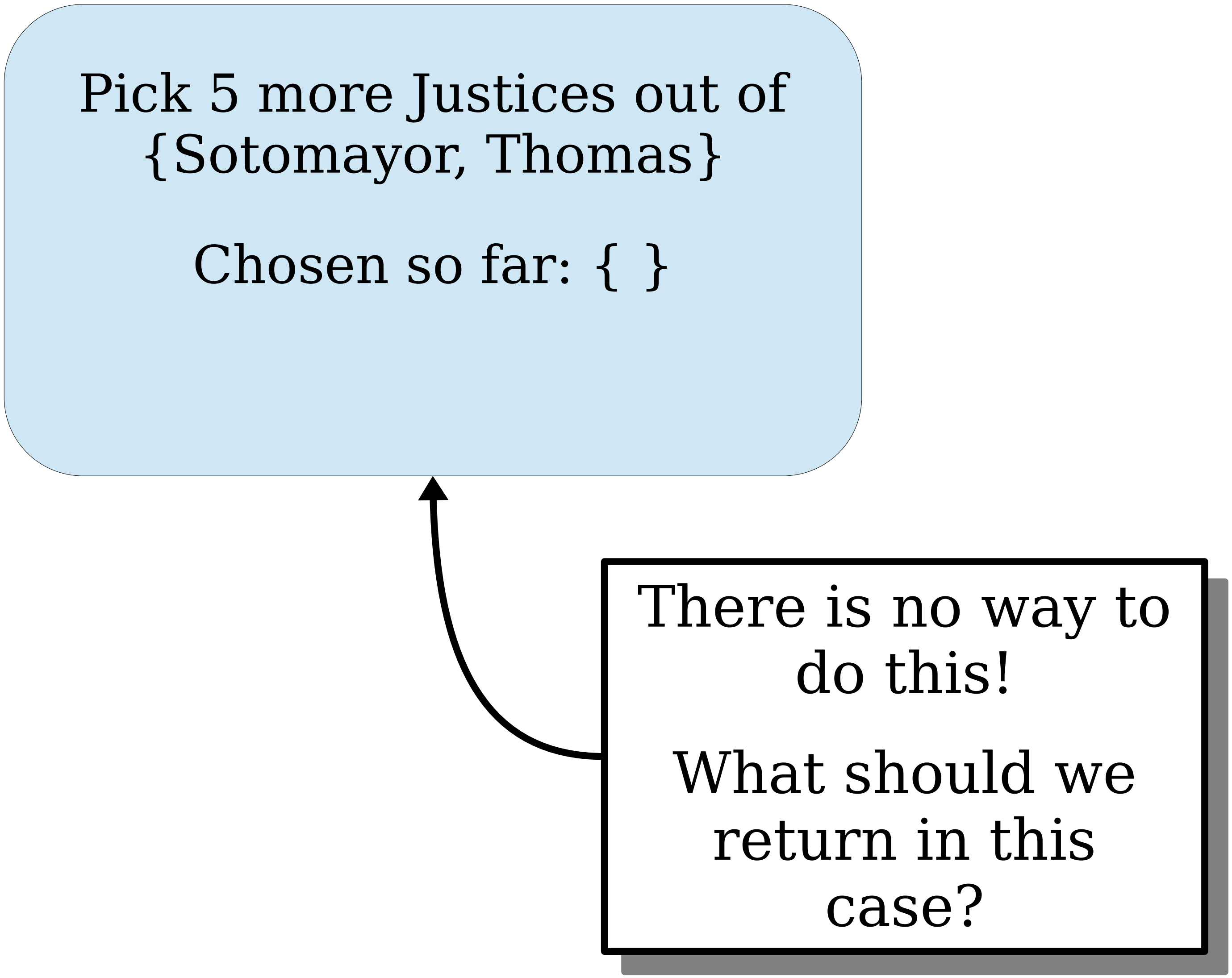
Pick 0 more Justices out of
{Kagan, Breyer}

Chosen so far:
{Alito, Roberts, Gorsuch,
Thomas, Sotomayor}

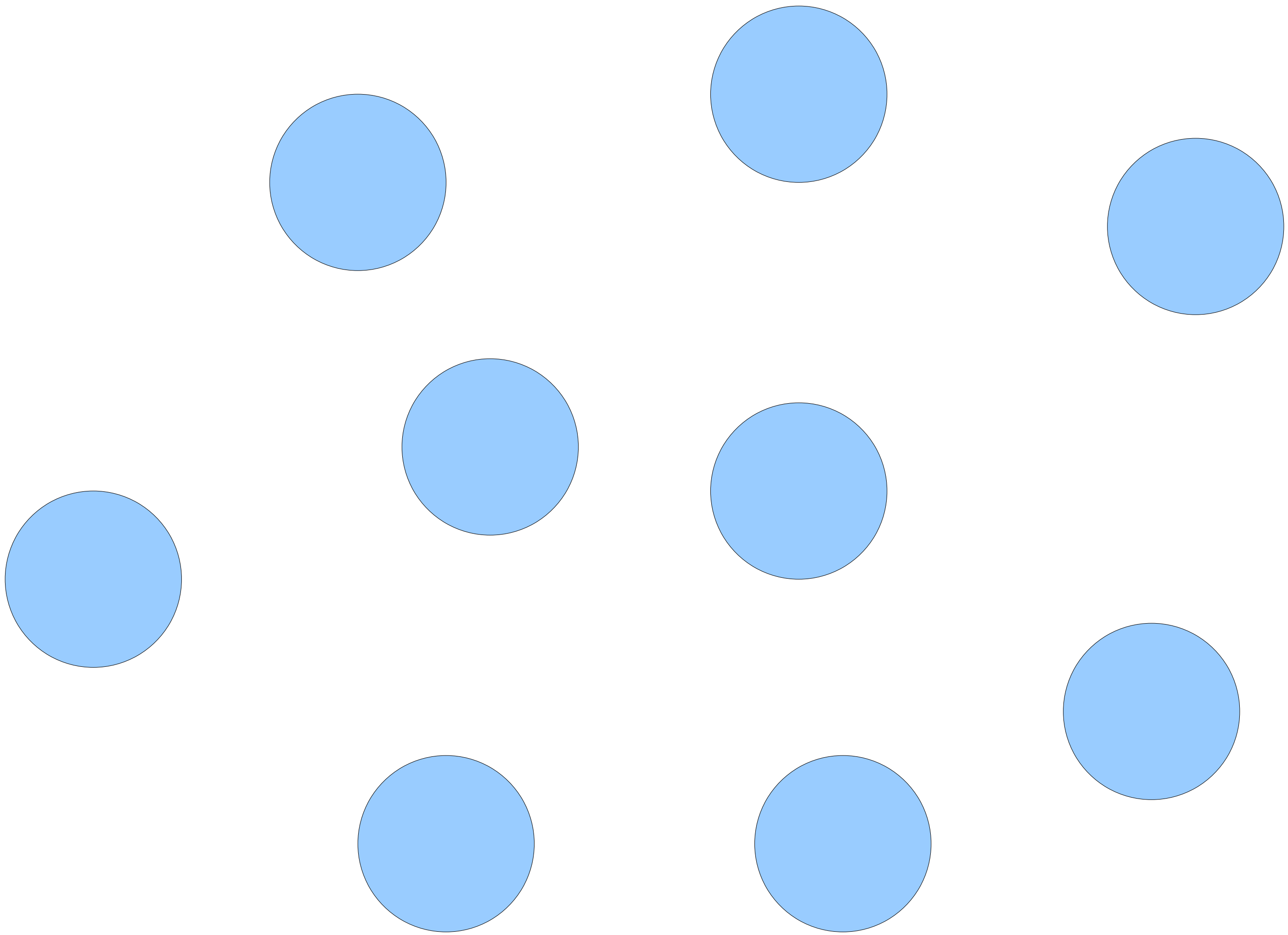
There's no need to keep looking.

What should we return in this case?

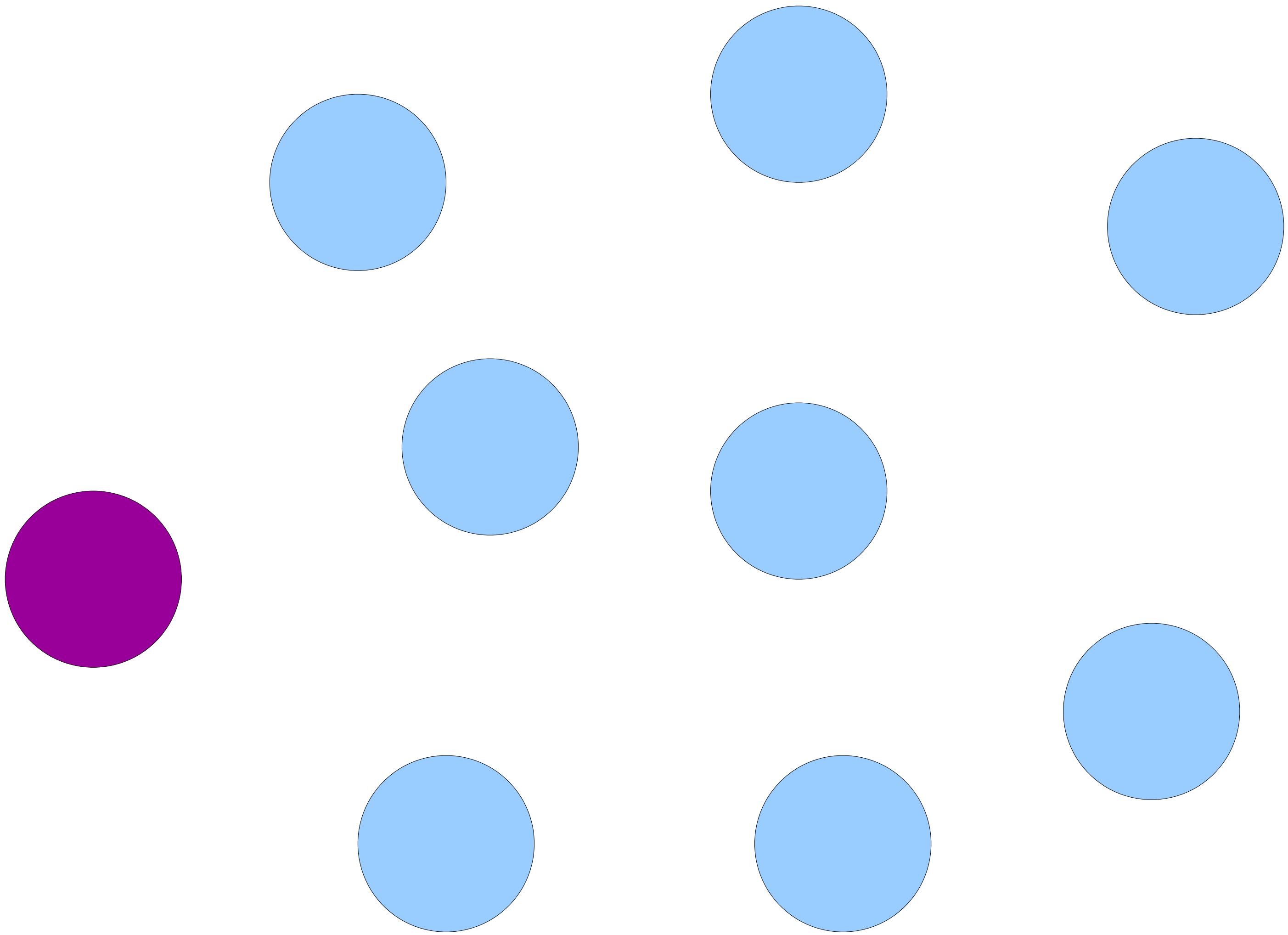
Our Base Case, Part II



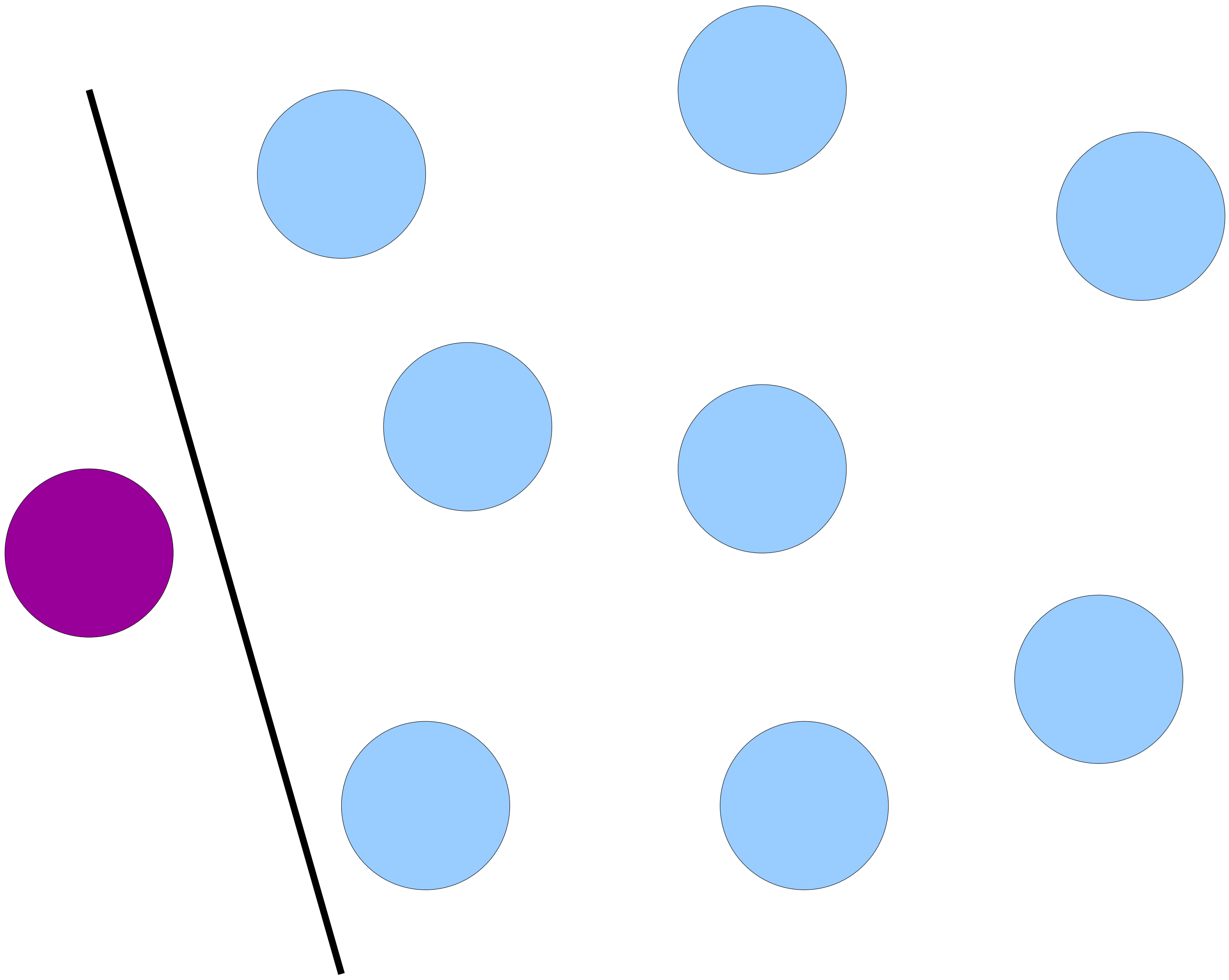
Generating Combinations



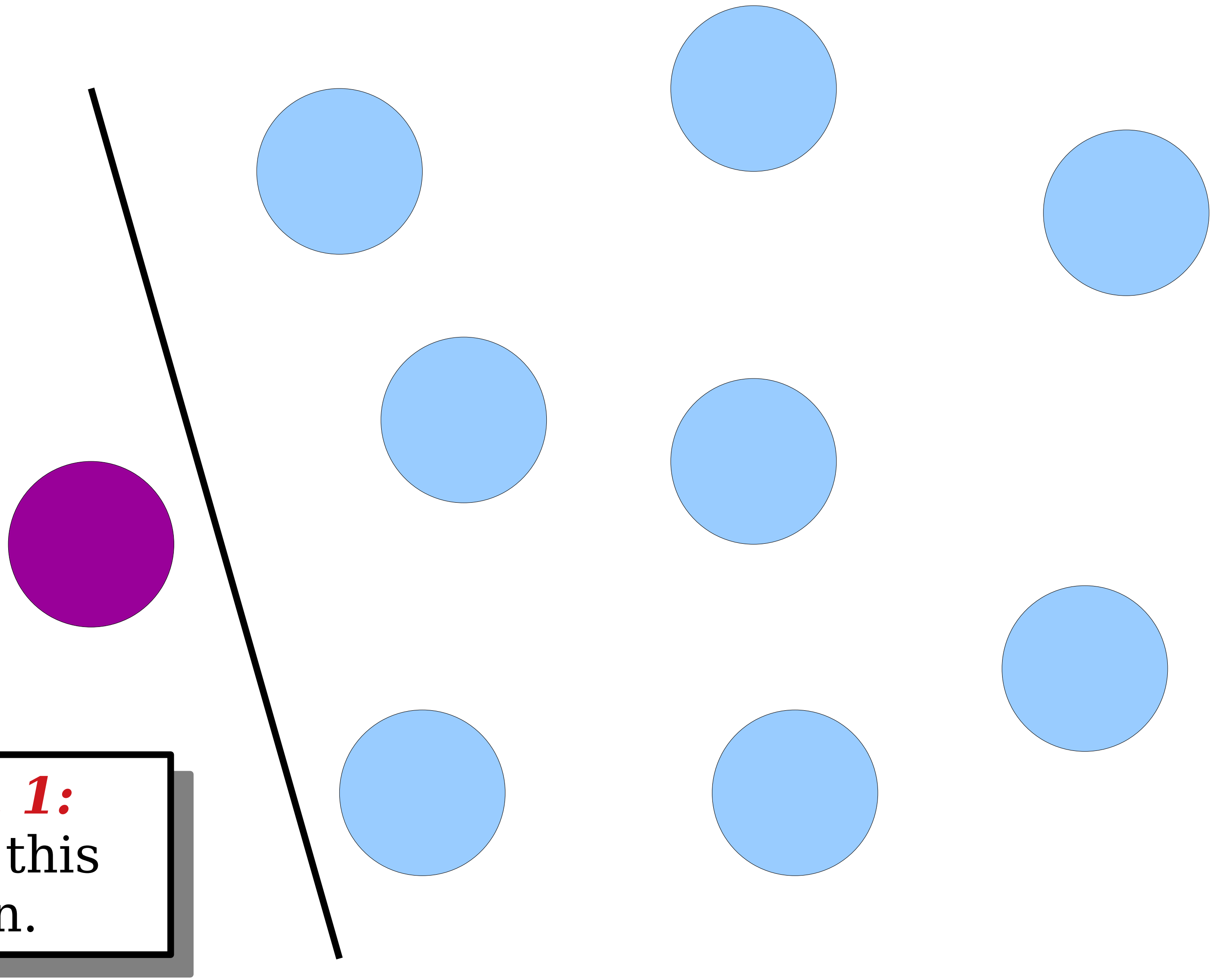
Generating Combinations



Generating Combinations

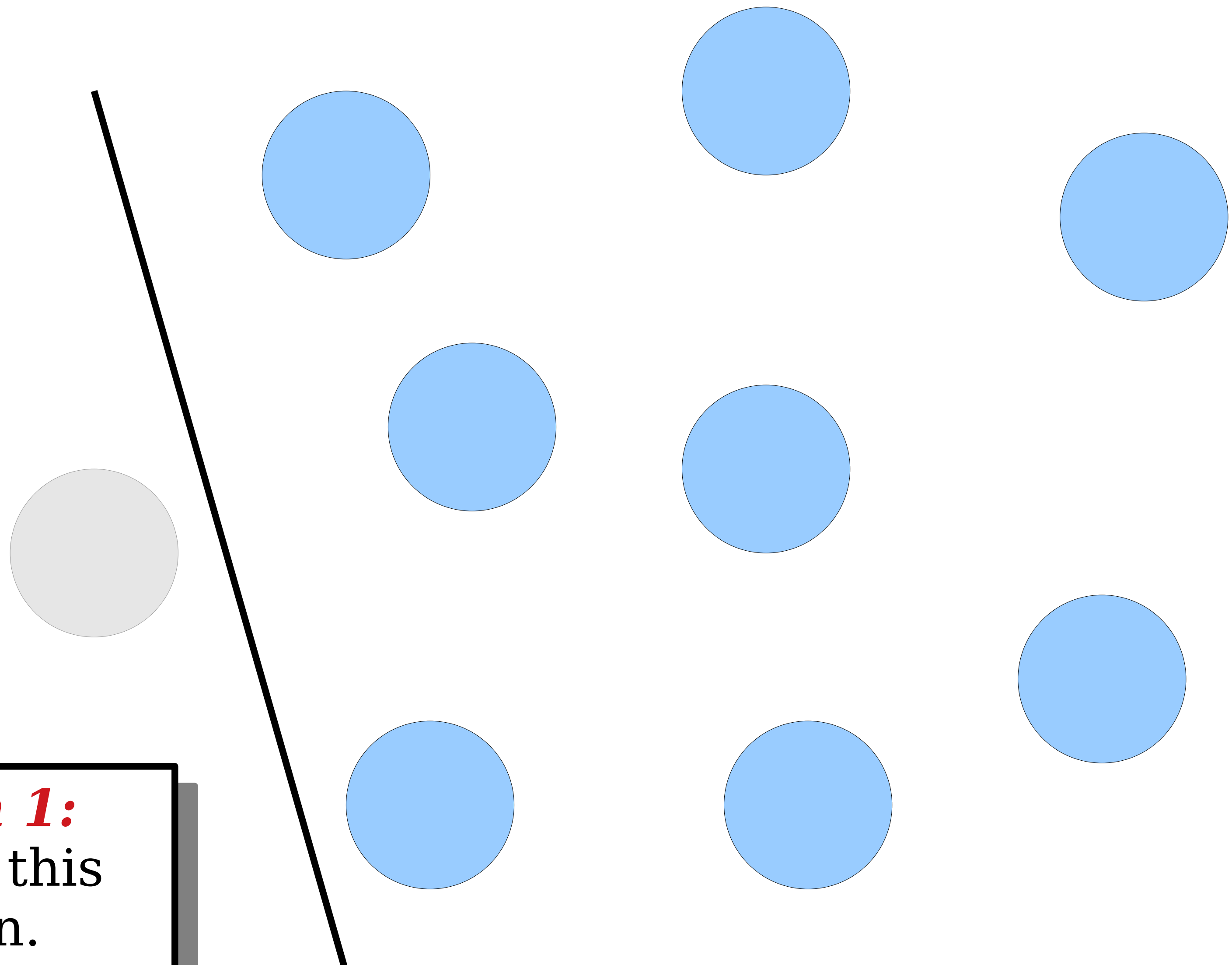


Generating Combinations



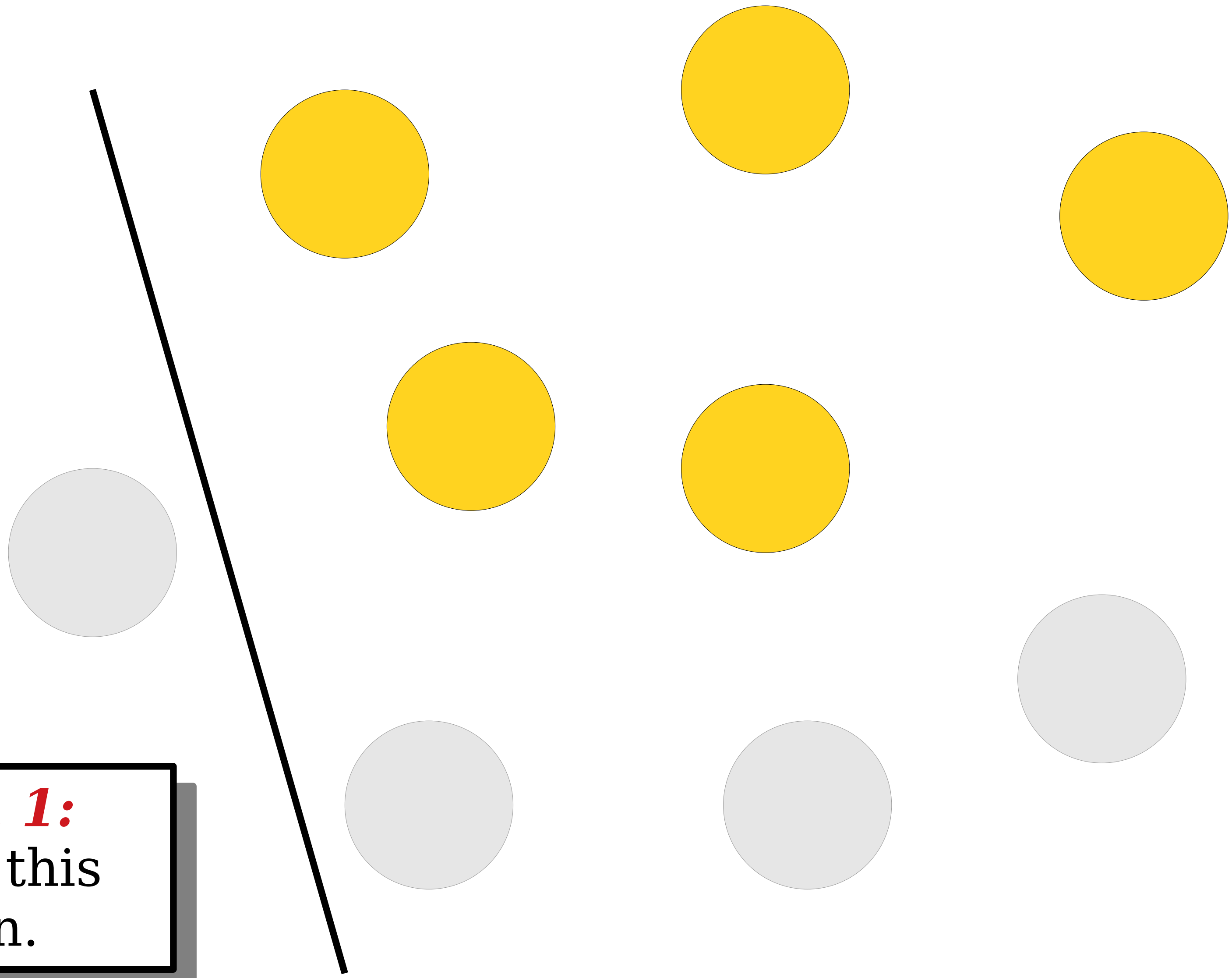
Option 1:
Exclude this
person.

Generating Combinations



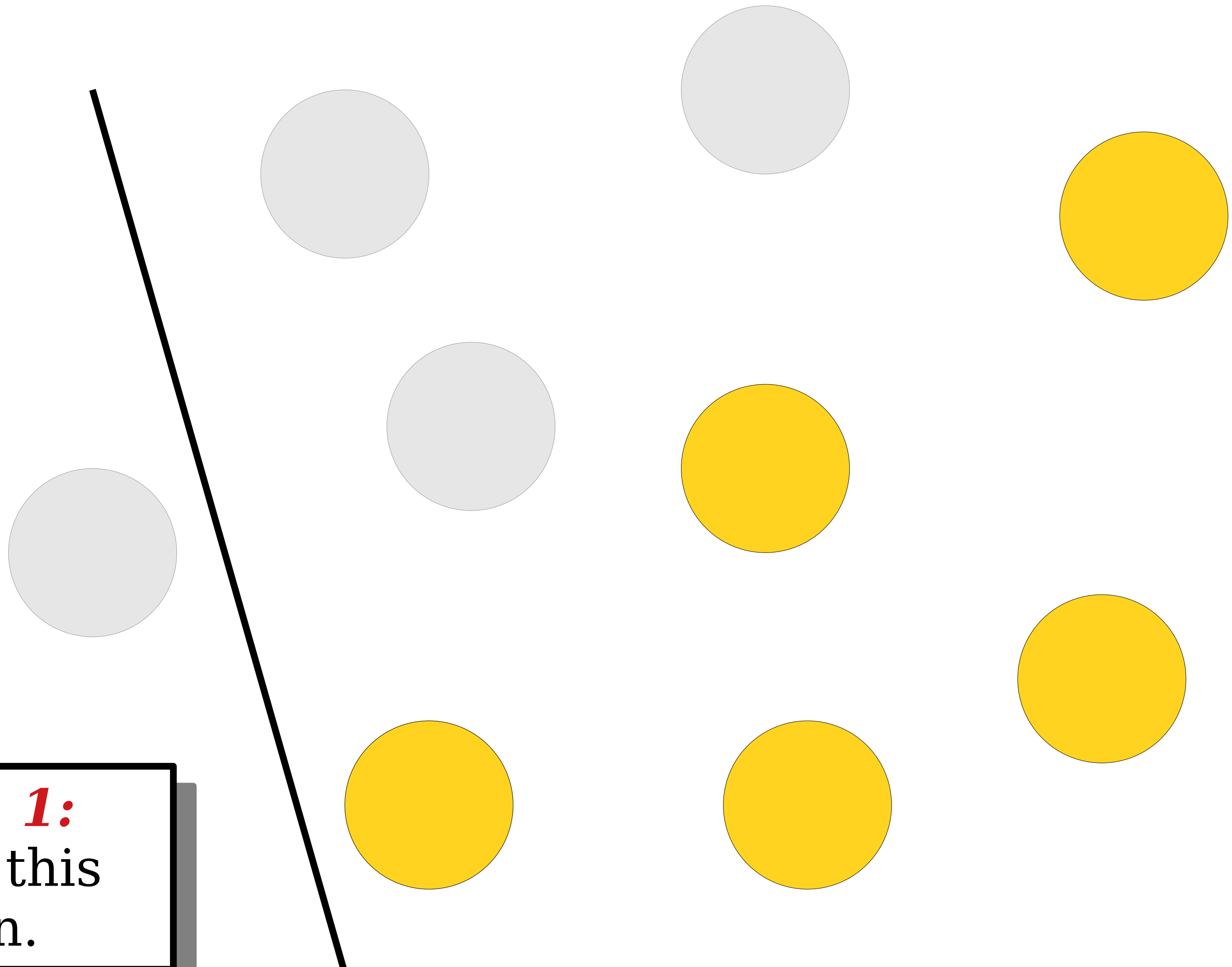
Option 1:
Exclude this person.

Generating Combinations



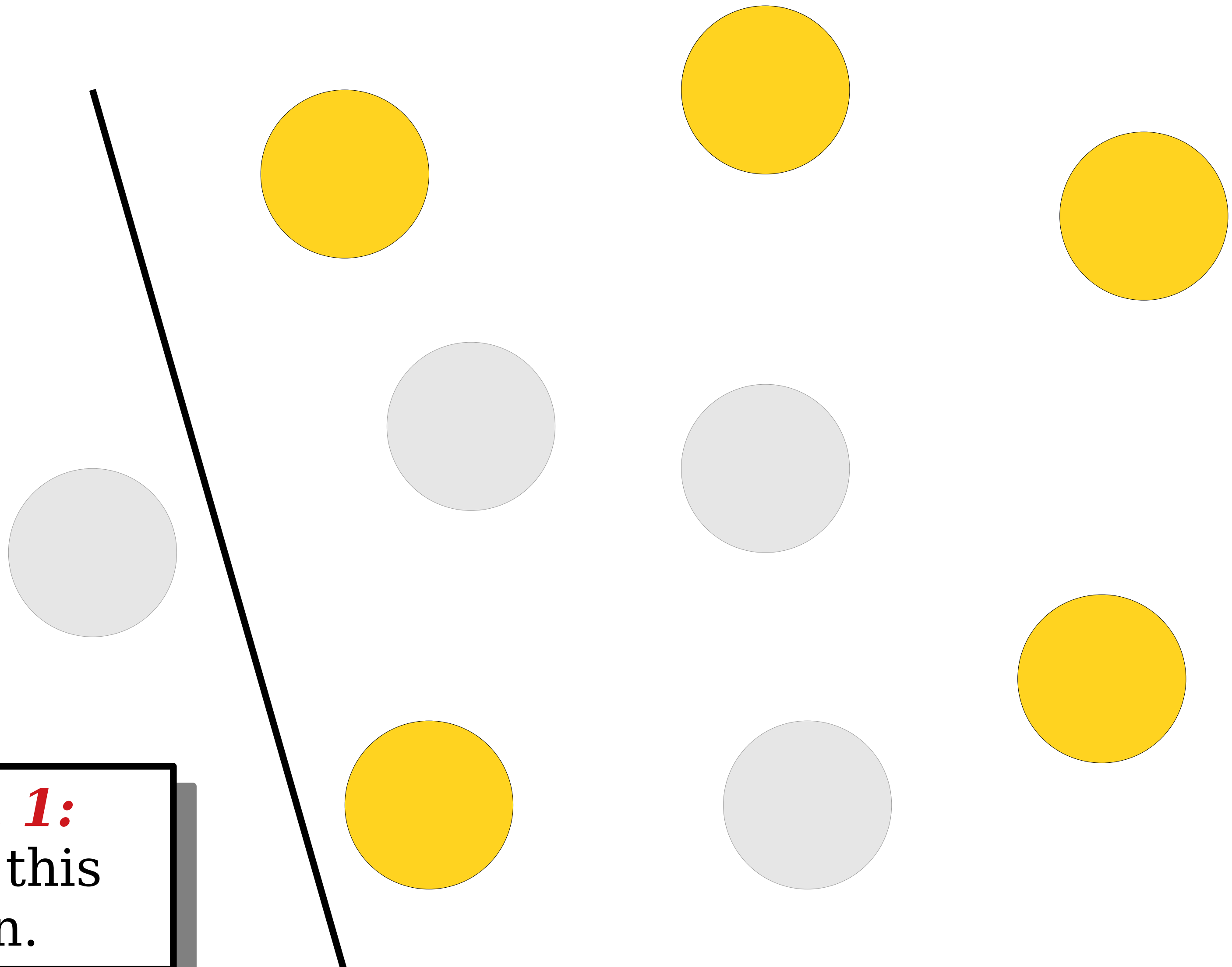
Option 1:
Exclude this person.

Generating Combinations



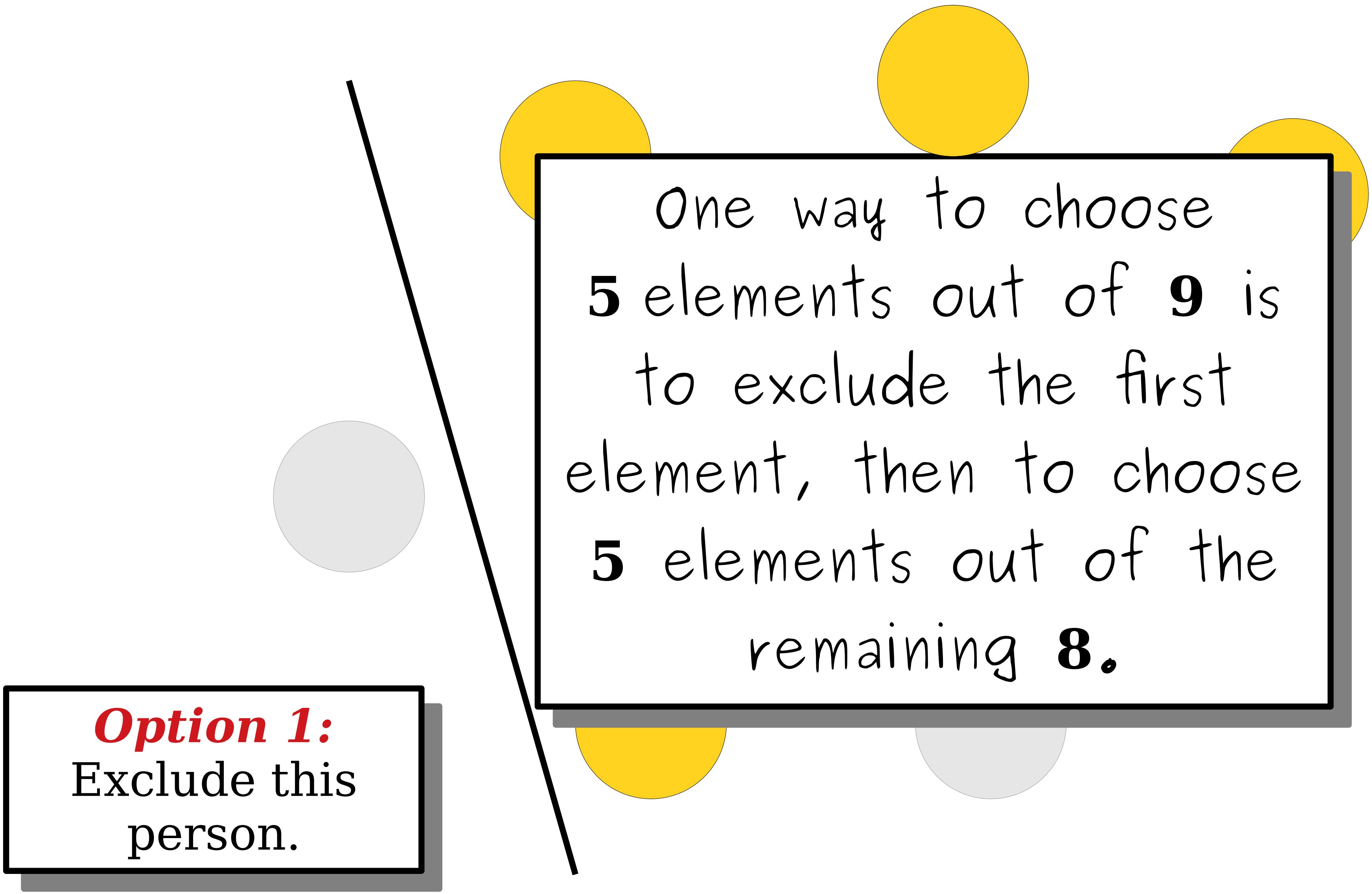
Option 1:
Exclude this
person.

Generating Combinations

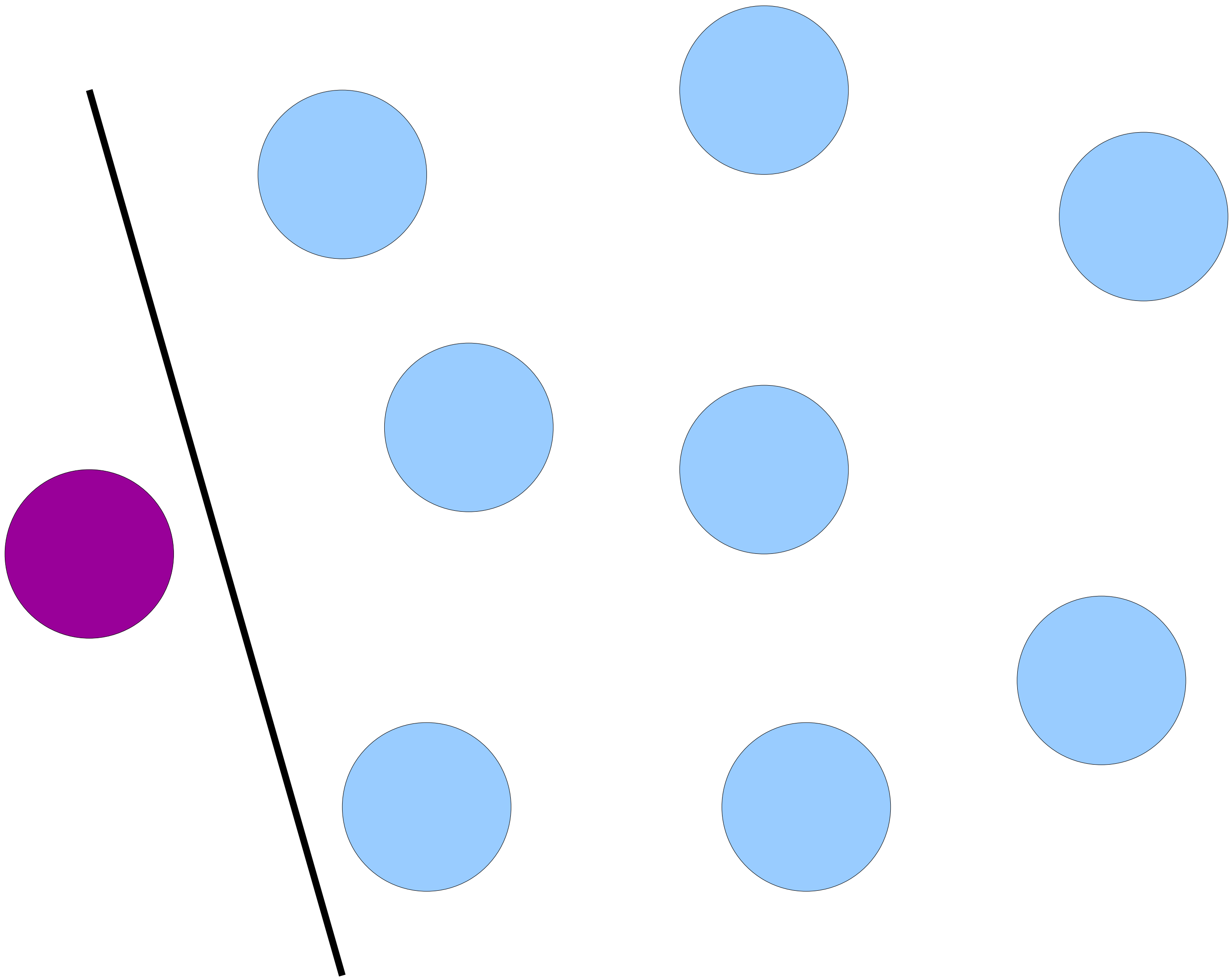


Option 1:
Exclude this person.

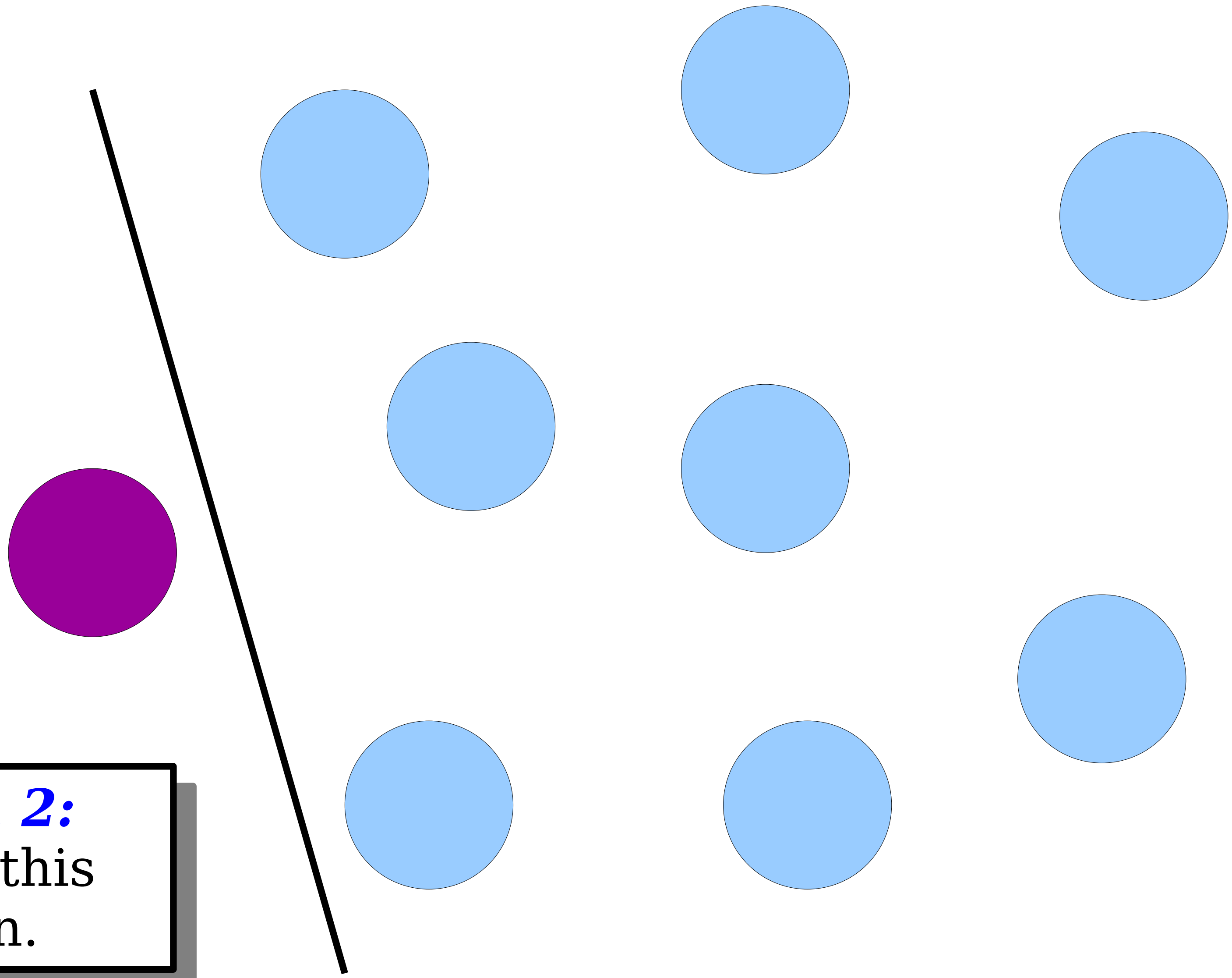
Generating Combinations



Generating Combinations

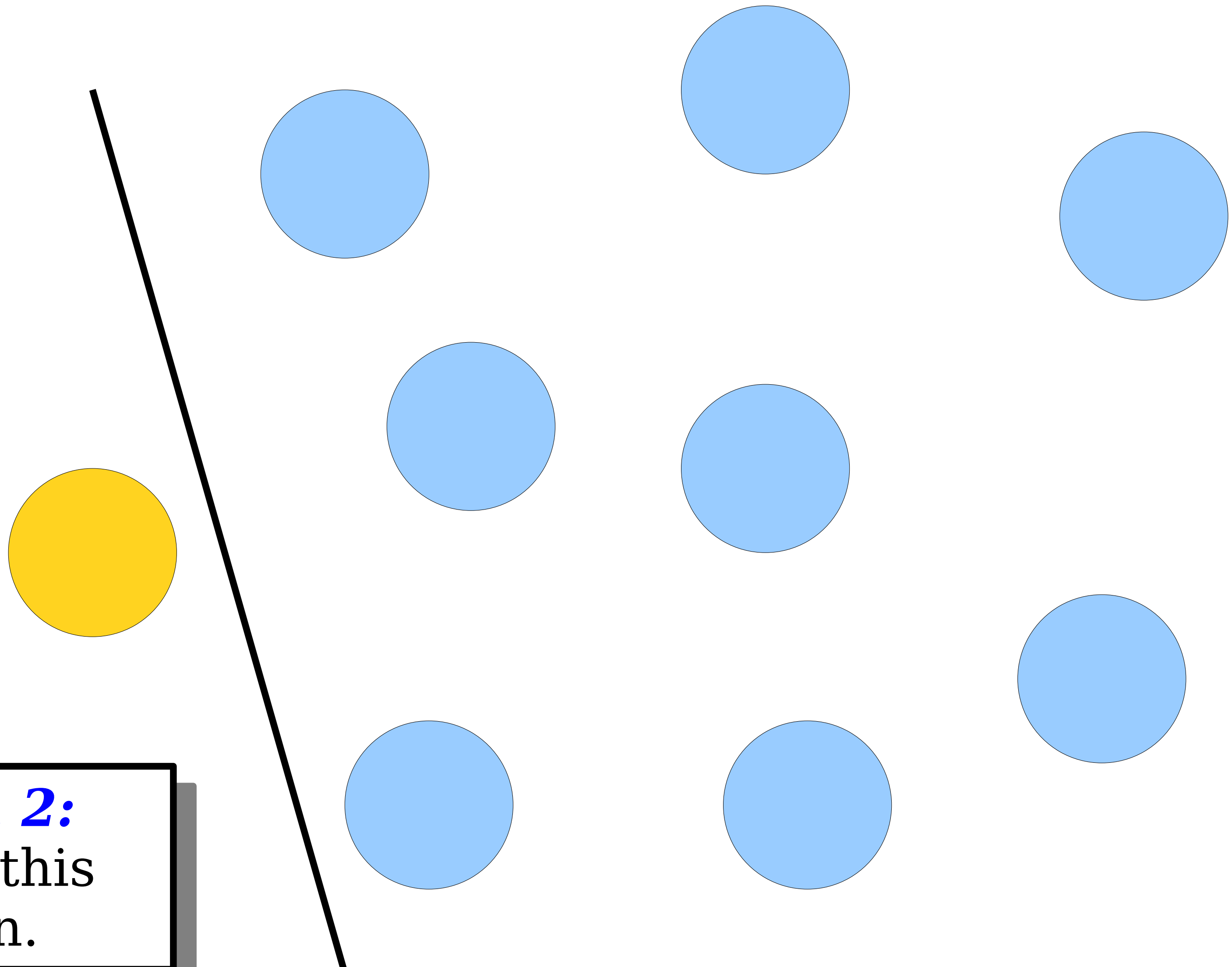


Generating Combinations



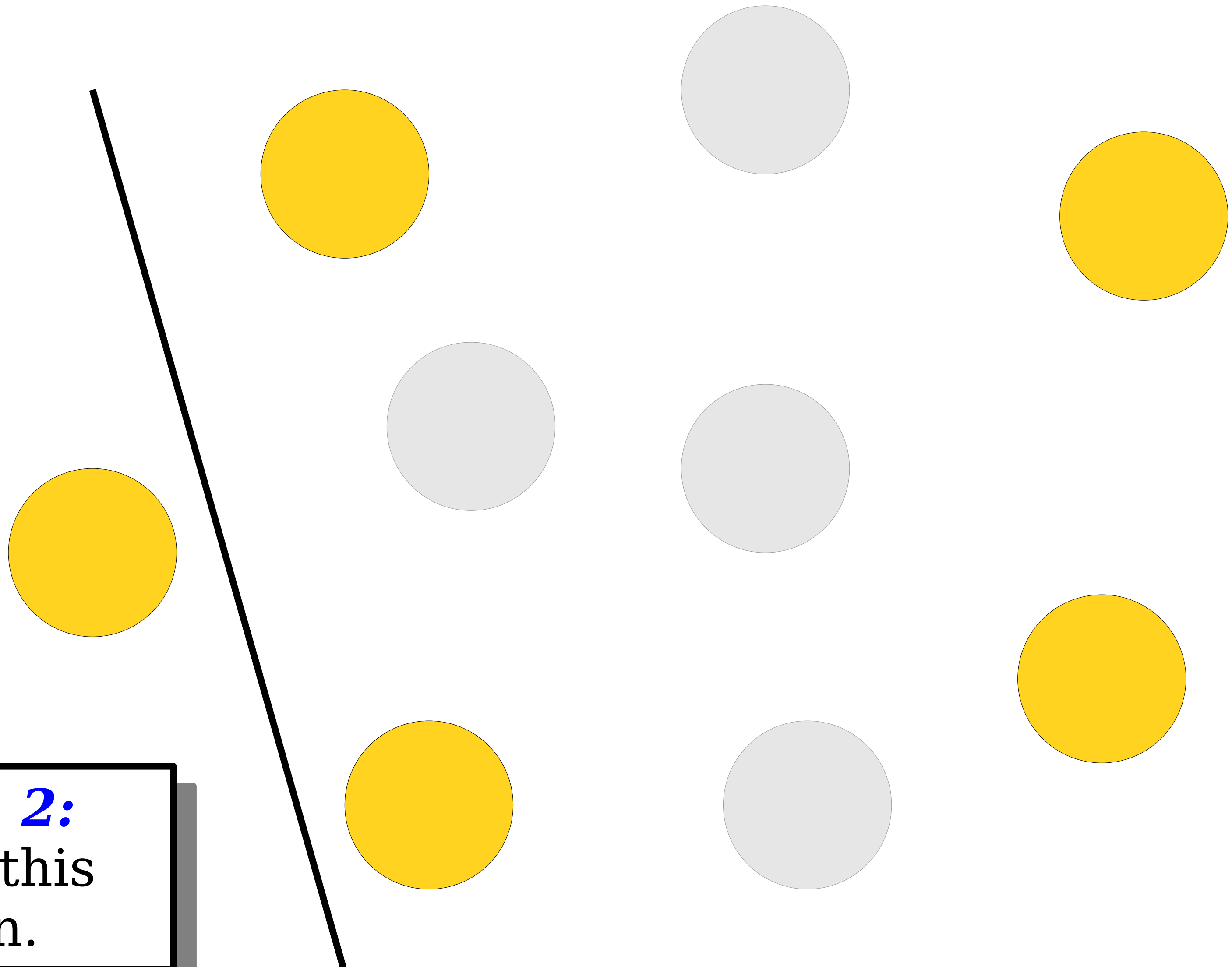
Option 2:
Include this person.

Generating Combinations



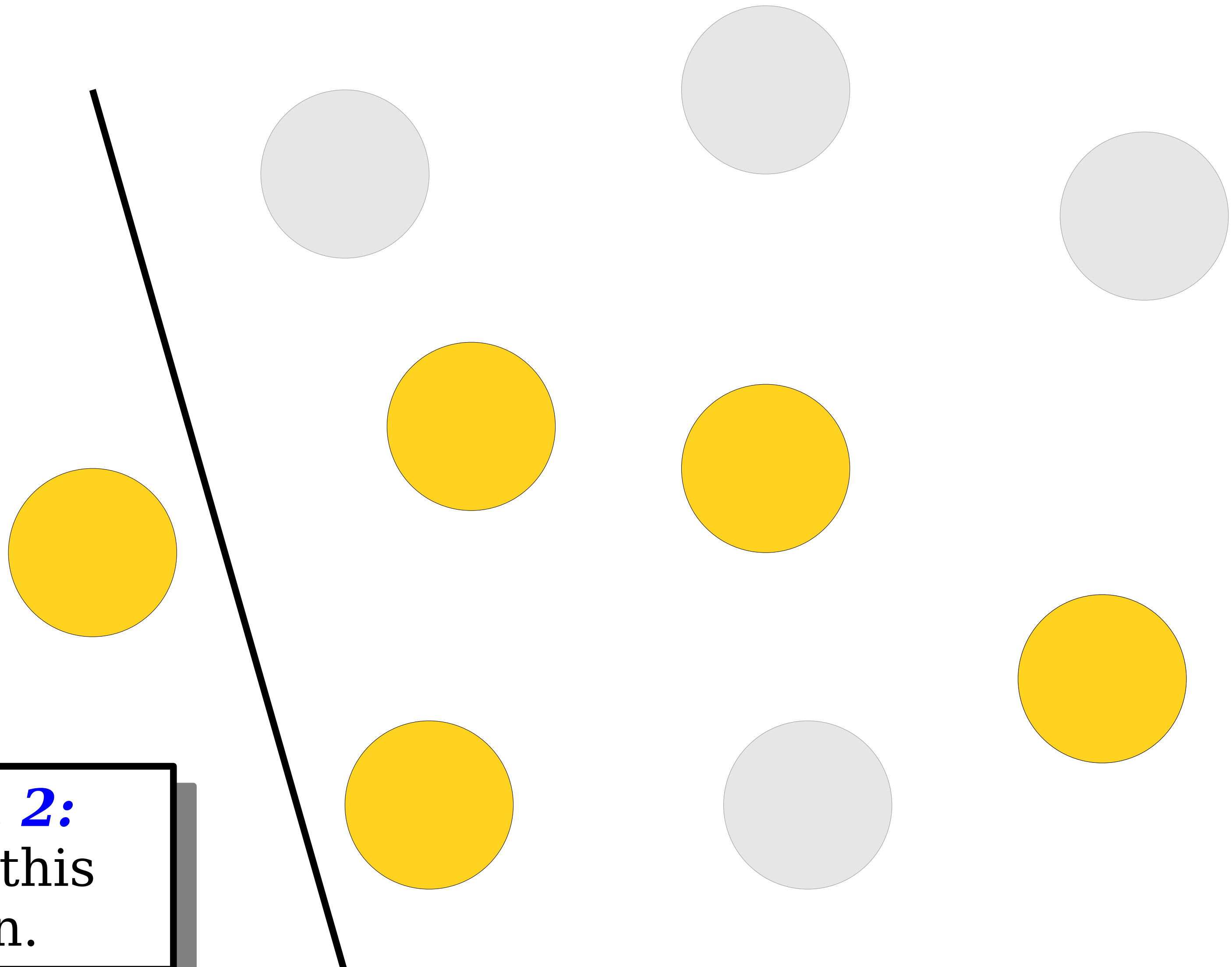
Option 2:
Include this person.

Generating Combinations



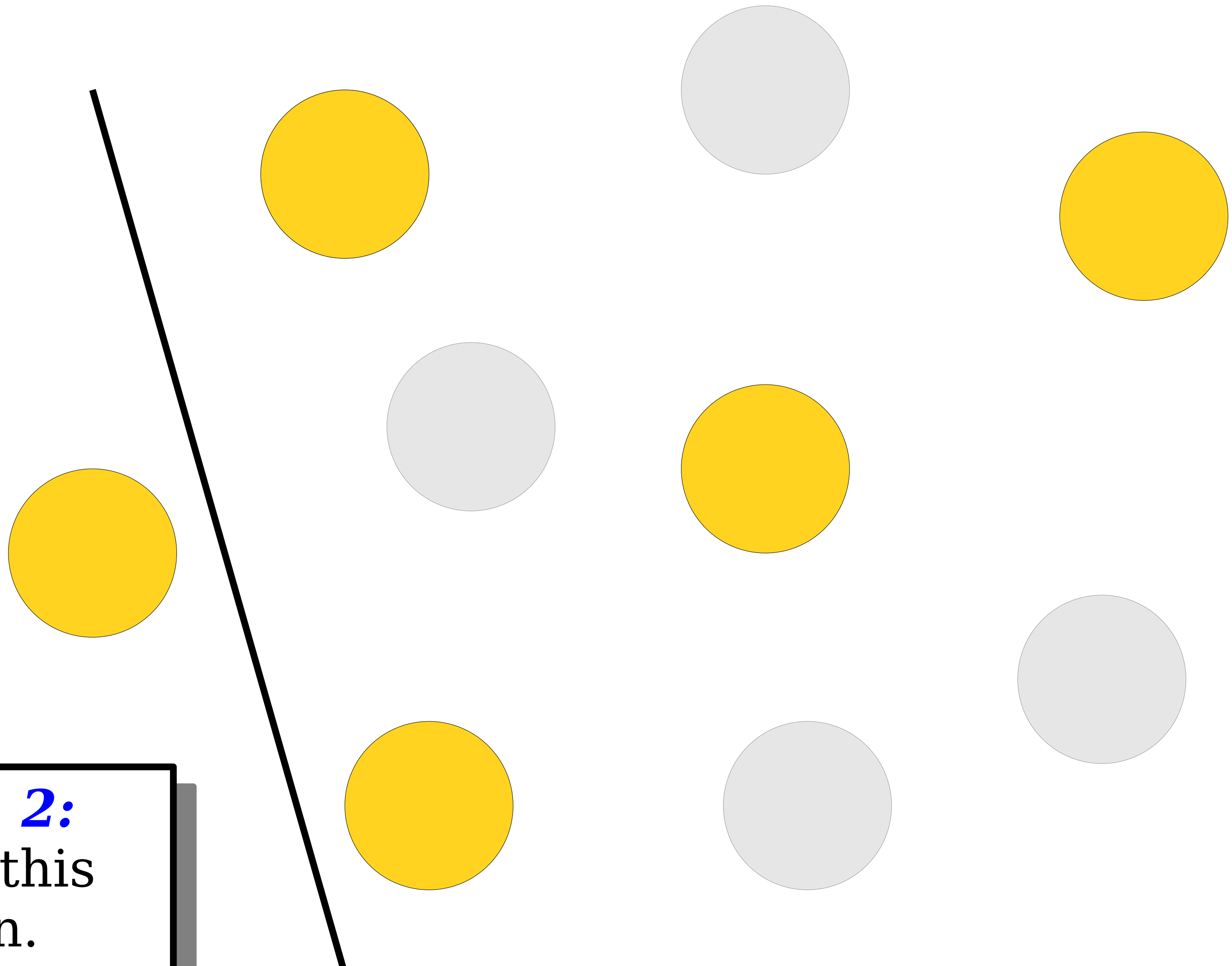
Option 2:
Include this
person.

Generating Combinations



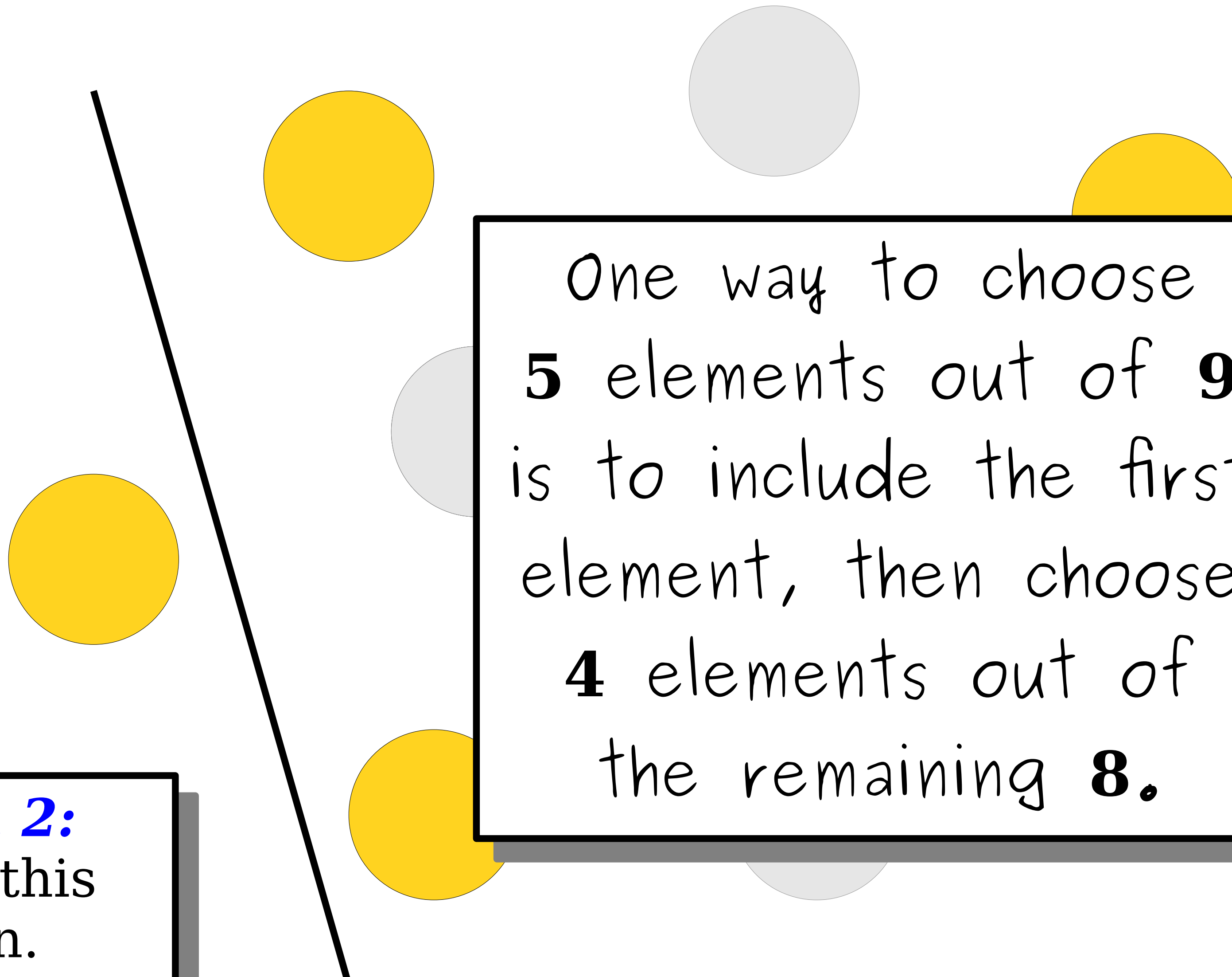
Option 2:
Include this person.

Generating Combinations



Option 2:
Include this
person.

Generating Combinations



One way to choose **5** elements out of **9** is to include the first element, then choose **4** elements out of the remaining **8**.

Option 2:
Include this person.

A Comment on Types

The Wonderful `auto` Keyword

- There are many cases in which there is exactly one possible type that a variable could have.
- In that case, rather than explicitly writing out the type, you can use the `auto` keyword:

auto *var* = *expression*;

- Don’t go crazy with this one; use it mostly to save typing when working with container types.

Base Case: No decisions remain.

Decisions yet to be made

Decisions already made

```
Container exploreRec(decisions remaining,
                     decisions already made) {
  if (no decisions remain) {
    return container of decisions made
  } else {
    Container result;
    for (each possible next choice) {
      result += exploreRec(all remaining decisions,
                           decisions made + that choice);
    }
    return result;
  }
}
```

Recursive Case: Try all options for the next decision.

```
Container exploreAllTheThings(initial state) {
  return exploreRec(initial state, no decisions made);
}
```

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?

S	T	R	I	N	G

The Startling Truth?

S	T	I	N	G

The Startling Truth?

S	I	N	G

The Startling Truth?

S	I	N

The Startling Truth?

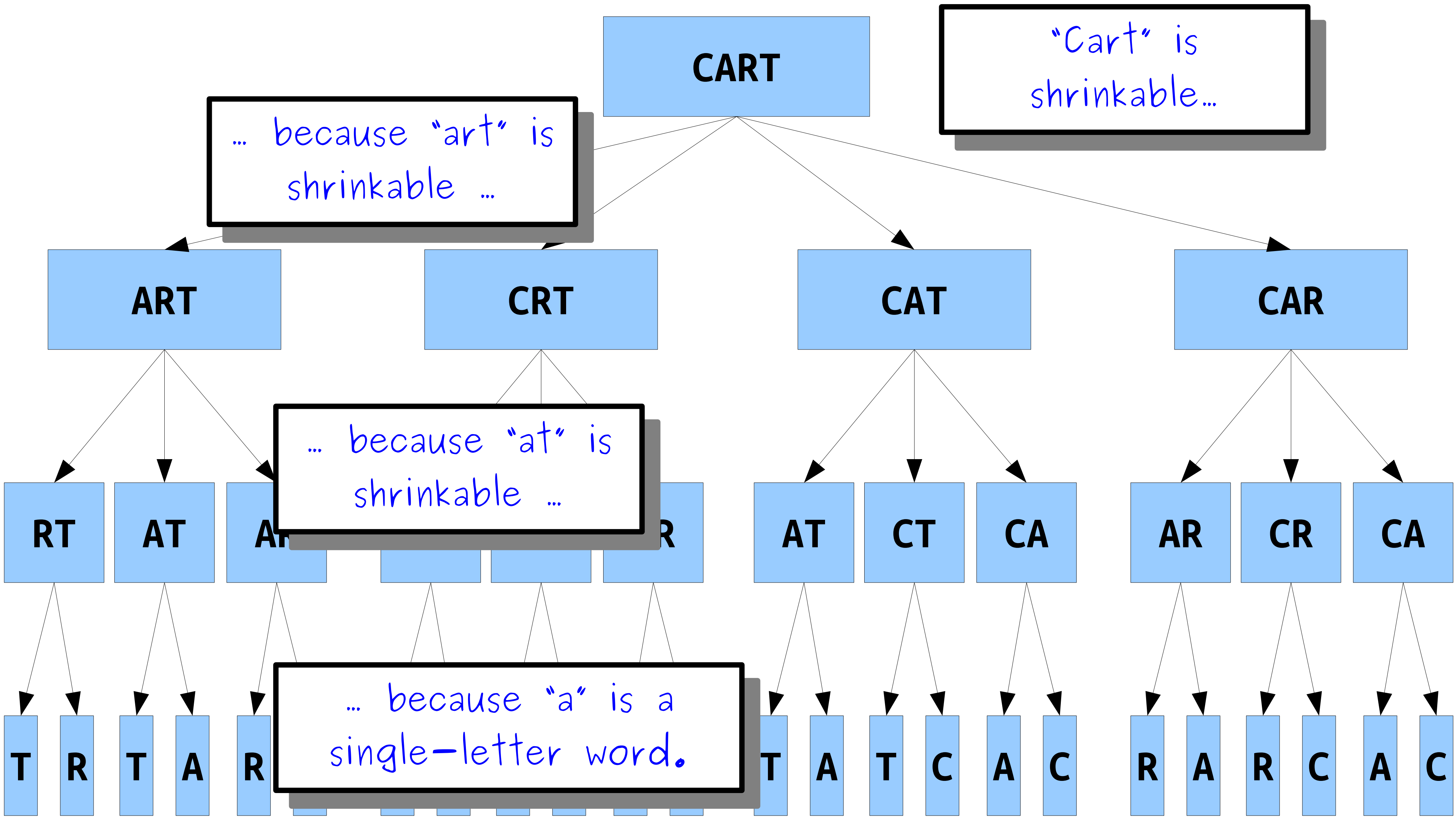
I	N

The Startling Truth?

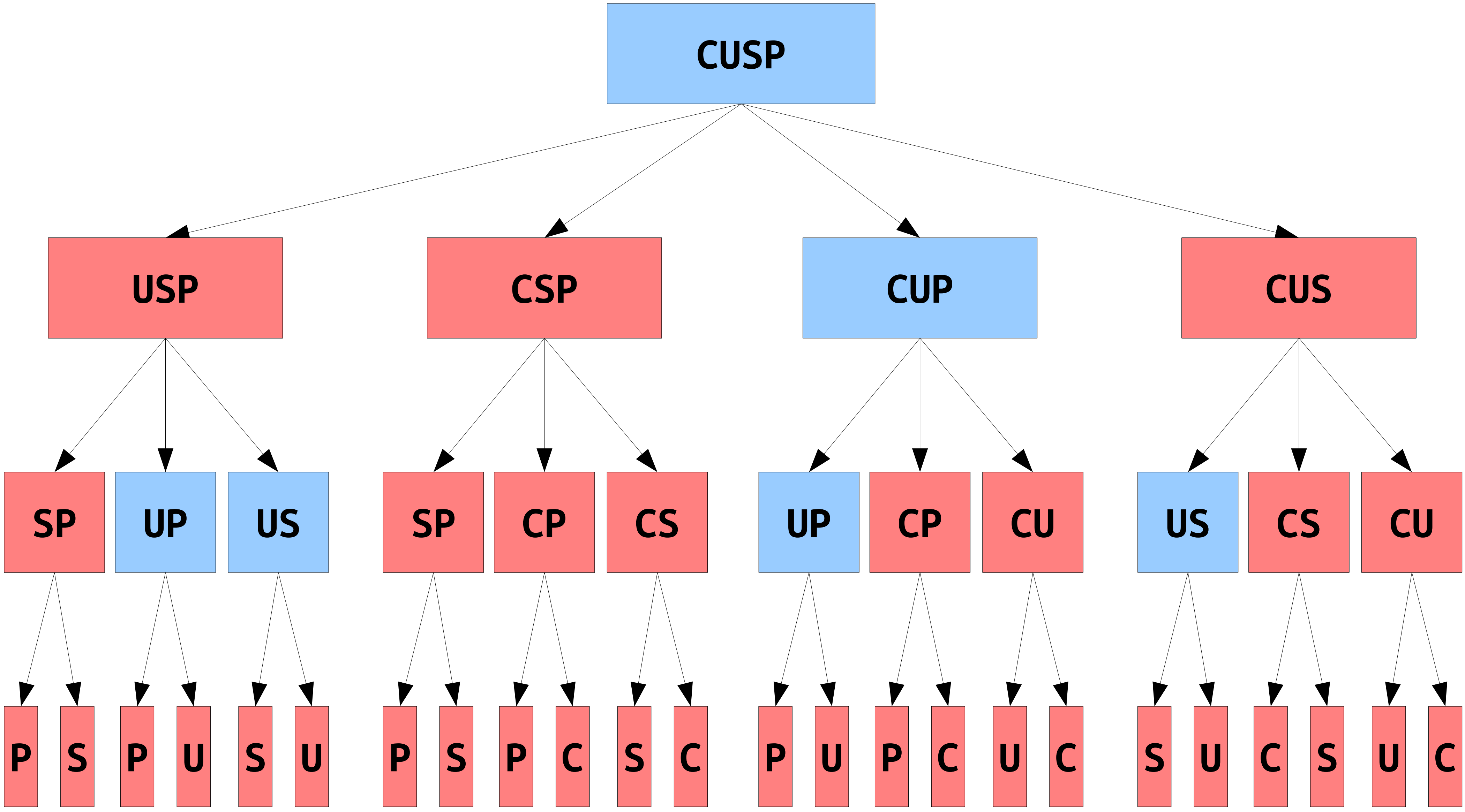
I

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

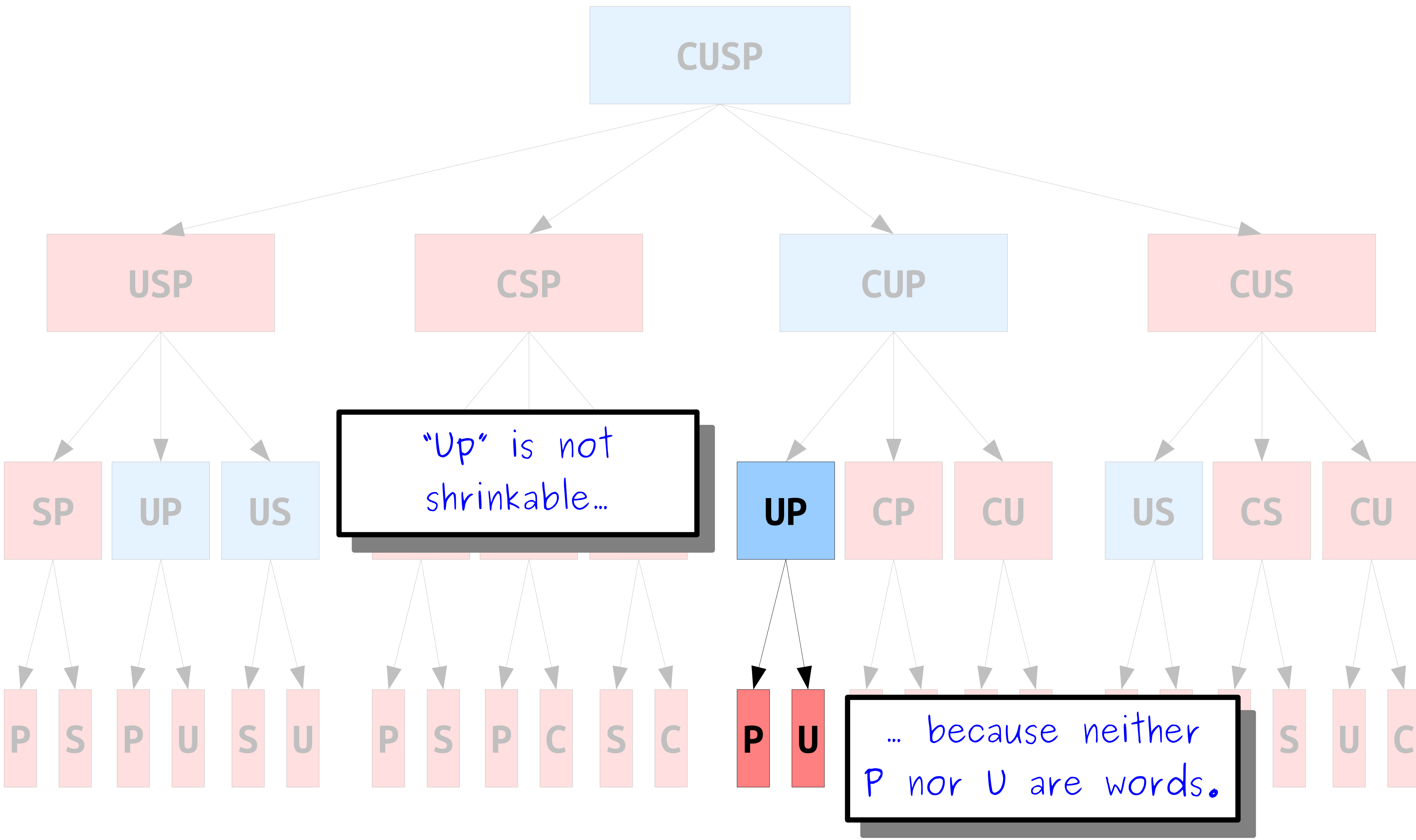
All Possible Paths



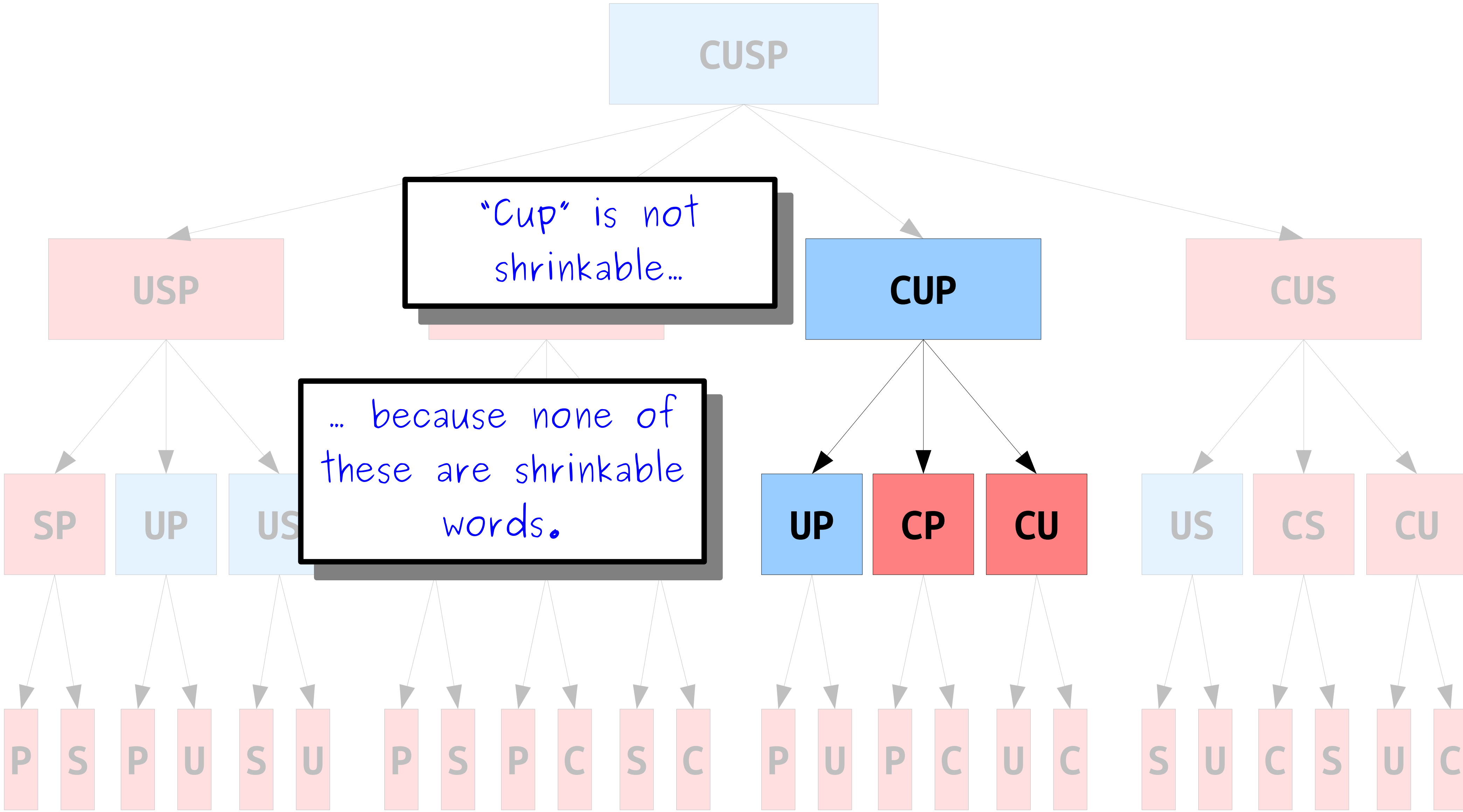
All Possible Paths



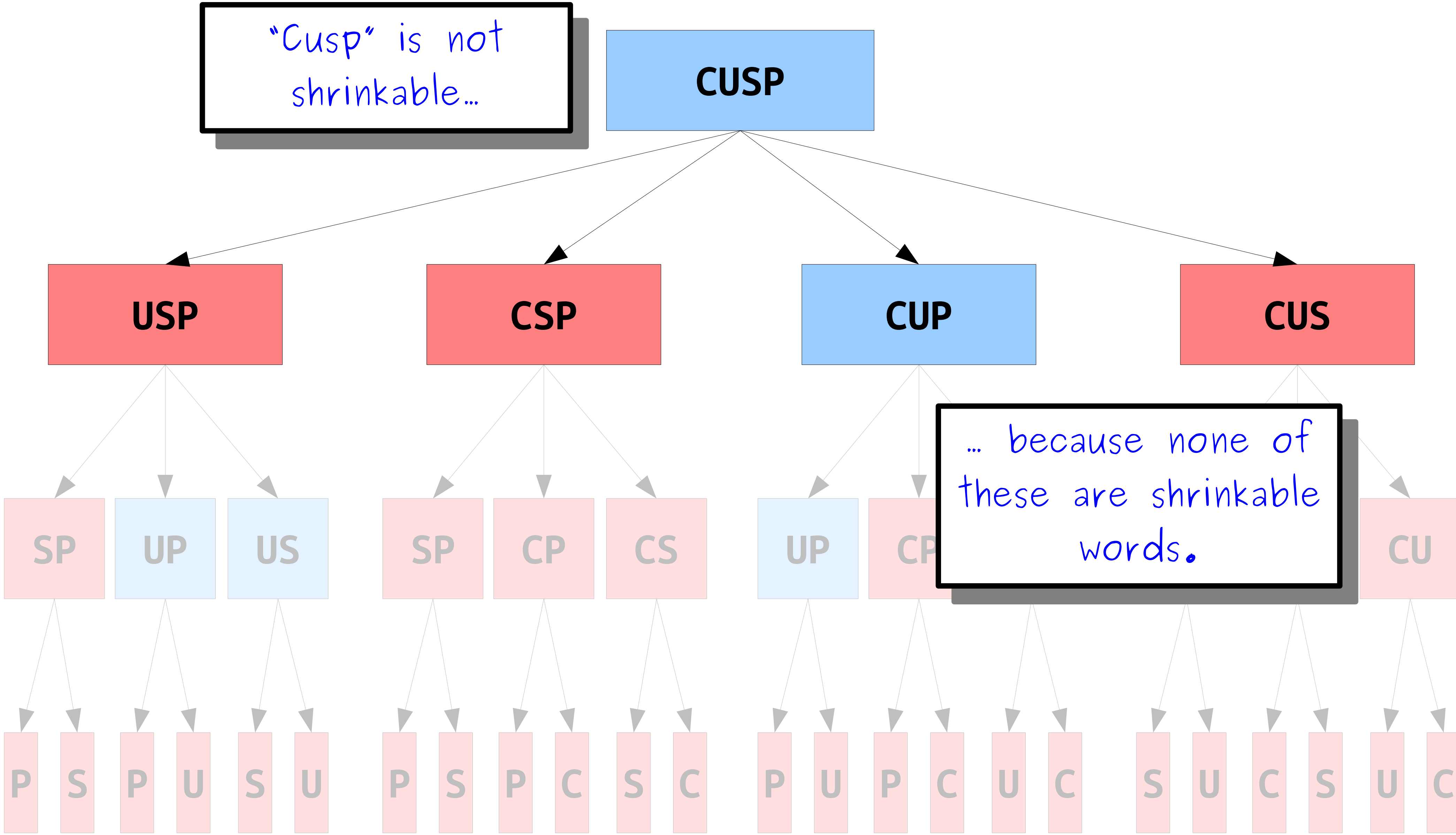
All Possible Paths



All Possible Paths



All Possible Paths



Shrinkable Words

- A ***shrinkable word*** is 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*:**
  - A string that is not a word is not a shrinkable word.
  - Any single-letter word is shrinkable (A, I, and O).
- ***Recursive Step*:**
  - A multi-letter word is shrinkable if you can remove a letter to form a shrinkable word.
  - A multi-letter word is not shrinkable if no matter what letter you remove, it’s not shrinkable.

Your Action Items

- ***Read Chapter 9 of the textbook.***
  - There’s tons of cool backtracking examples there, and it will help you prep for Friday.
- ***Keep working on Assignment 3.***
  - If you’re following our timetable, you should be working on “What Are YOU Doing?” See if you can finish it by Wednesday and start working on Shift Scheduling.
  - Ask for help if you need it! That’s what we’re all here for.

Next Time

- ***Output Parameters***
  - Recovering the solution to a backtracking problem.
- ***More Backtracking***
  - Techniques in searching for feasibility.
- ***Closing Thoughts on Recursion***
  - It’ll come back, but we’re going to focus on other things for a while!