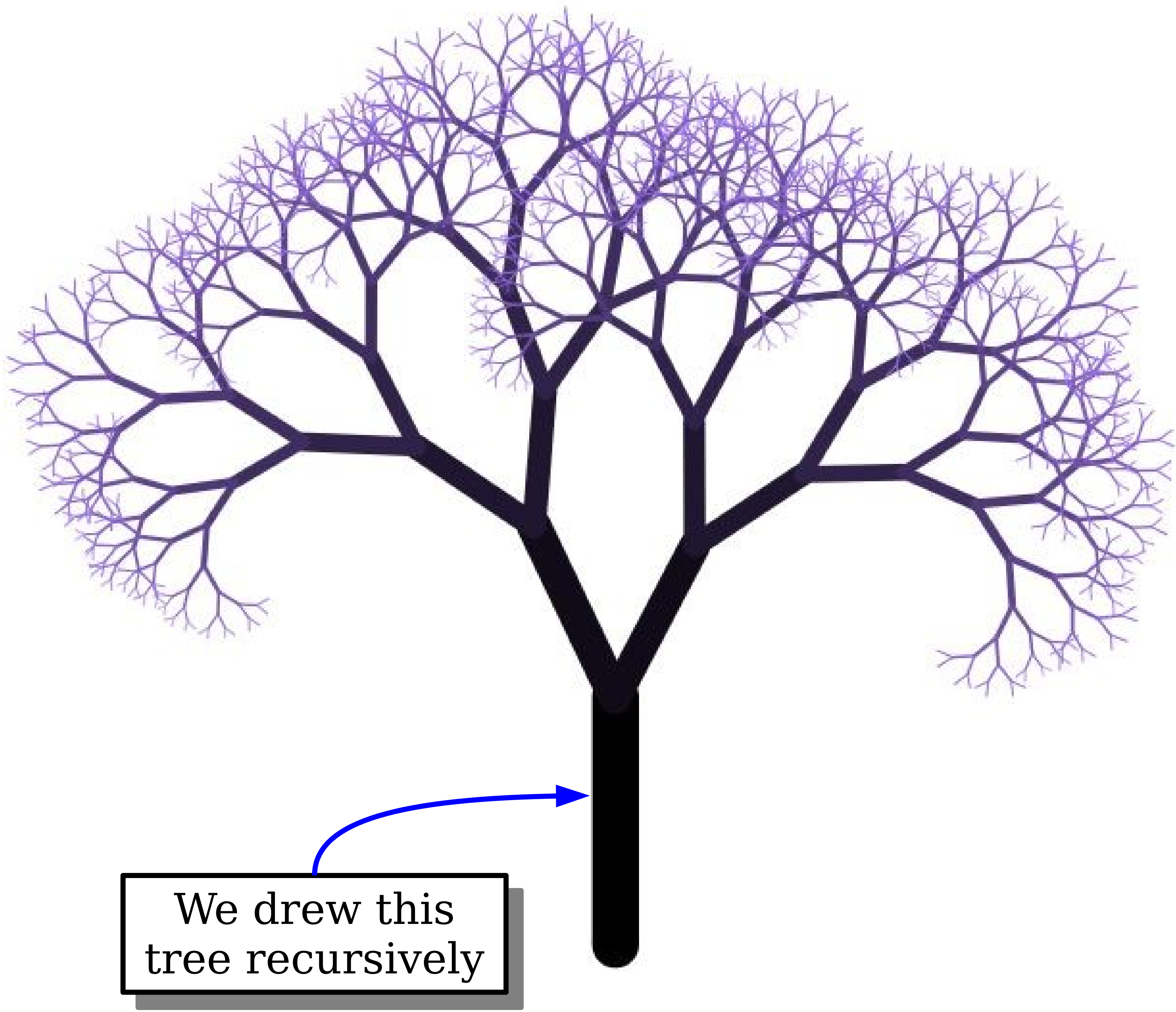
Thinking Recursively

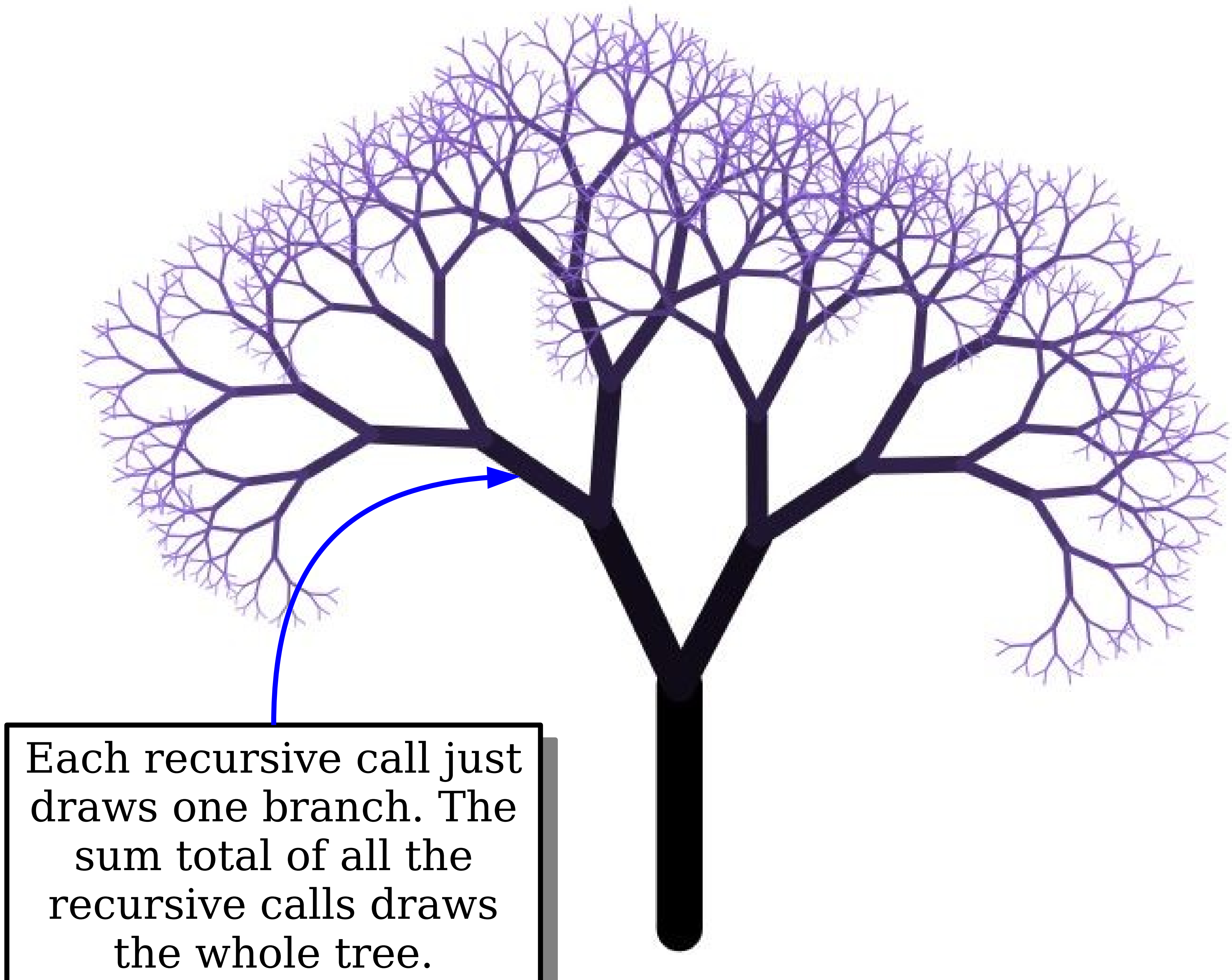
Part II

Outline for Today

- ***Recap from Last Time***
  - Where are we, again?
- ***Wrapper Functions***
  - Cleaning up some code.
- ***Enumerating Subsets***
  - A classic combinatorial problem.
- ***Decision Trees***
  - Generating all solutions to a problem.

Recap from Last Time


We drew this
tree recursively

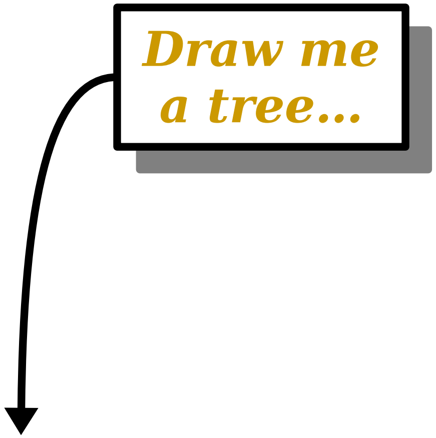

Each recursive call just draws one branch. The sum total of all the recursive calls draws the whole tree.

New Stuff!

Cleaning Up our Code

```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```

```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```




```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```

Draw me a tree…

… in this window …

```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```

Draw me a tree…

… in this window …

… at this position …

```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```

Draw me a tree…

… in this window …

… at this position …

… that's this big …

```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```

Draw me a tree...

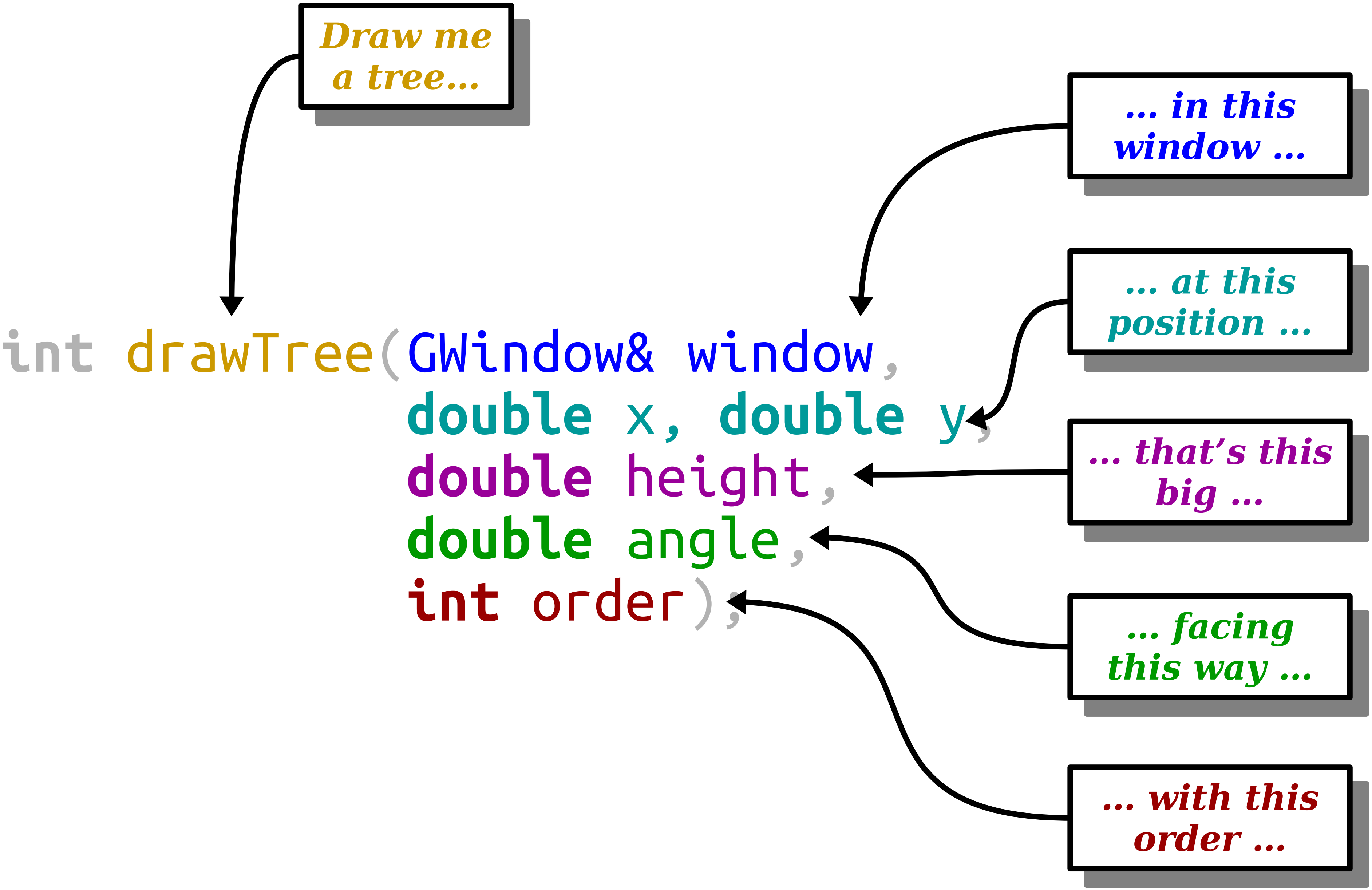
... in this window ...

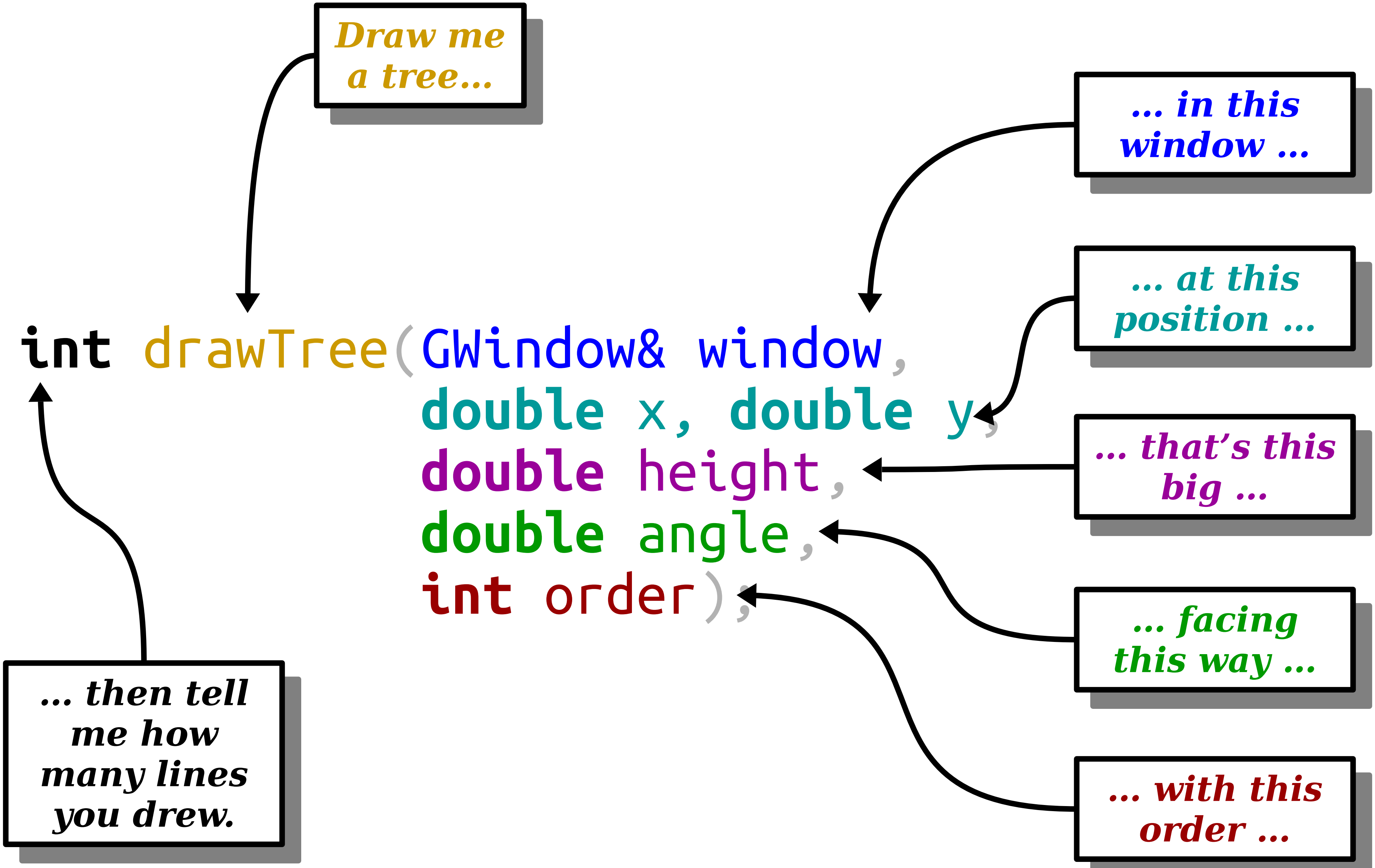
... at this position ...

... that's this big ...

... facing this way ...

```
int drawTree(GWindow& window,
             double x, double y,
             double height,
             double angle,
             int order);
```


Draw me a tree…
… in this window …
… at this position …
… that's this big …
… facing this way …
… with this order …
int drawTree(GWindow& window,
double x, double y,
double height,
double angle,
int order);



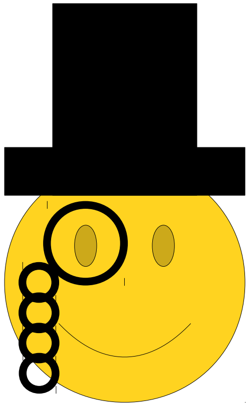
Draw me a tree…
… in this window …
… at this position …
… that's this big …
… facing this way …
… with this order …
… then tell me how many lines you drew.
int drawTree(GWindow& window,
double x, double y,
double height,
double angle,
int order);

```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight,
                             90, 8);
```

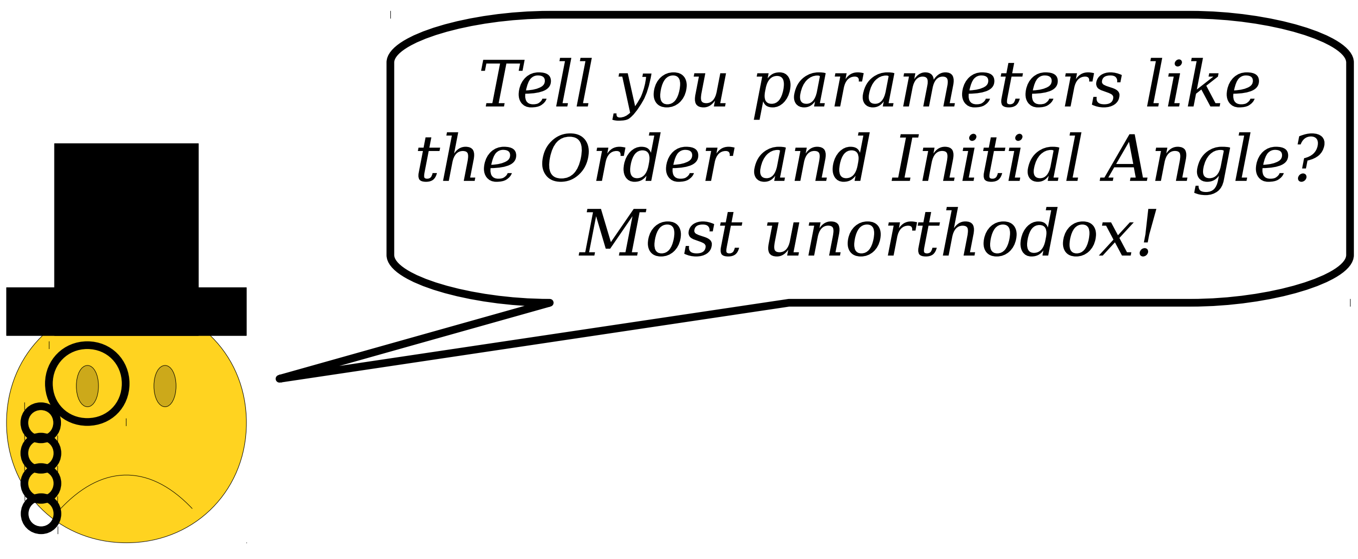
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight,
                             90, 8);
```

```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight,
                             90, 8);
```

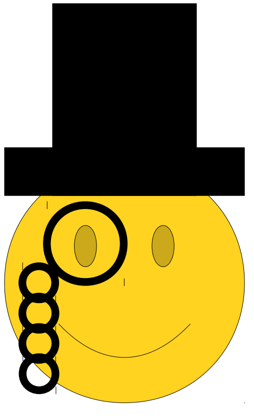
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight,
                             90, 8);
```



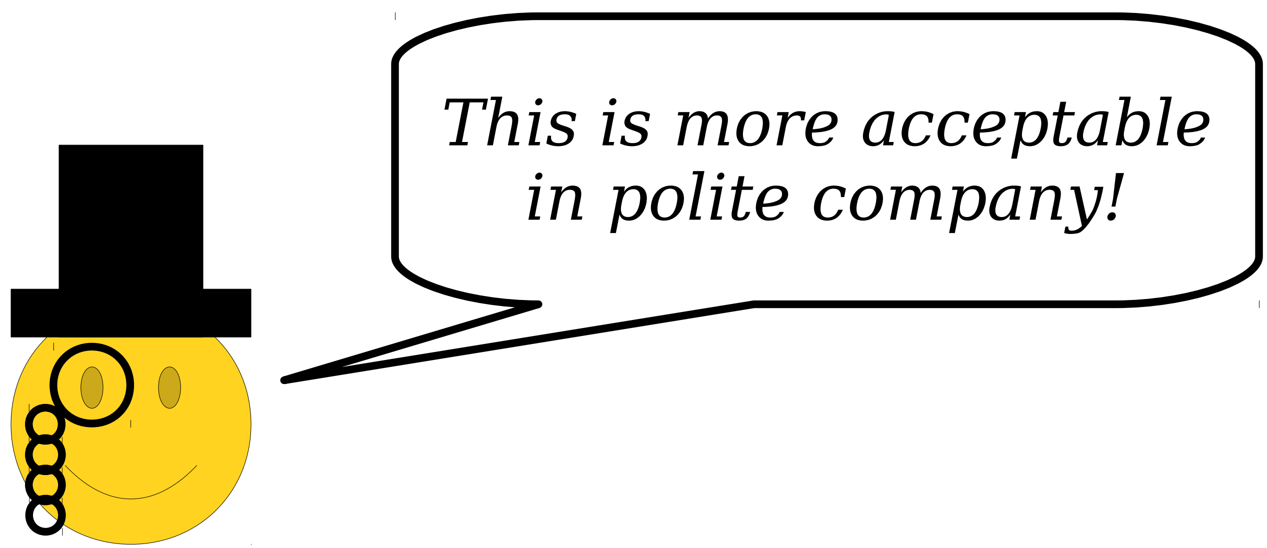
```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight,
                             90, 8);
```



```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight);
```

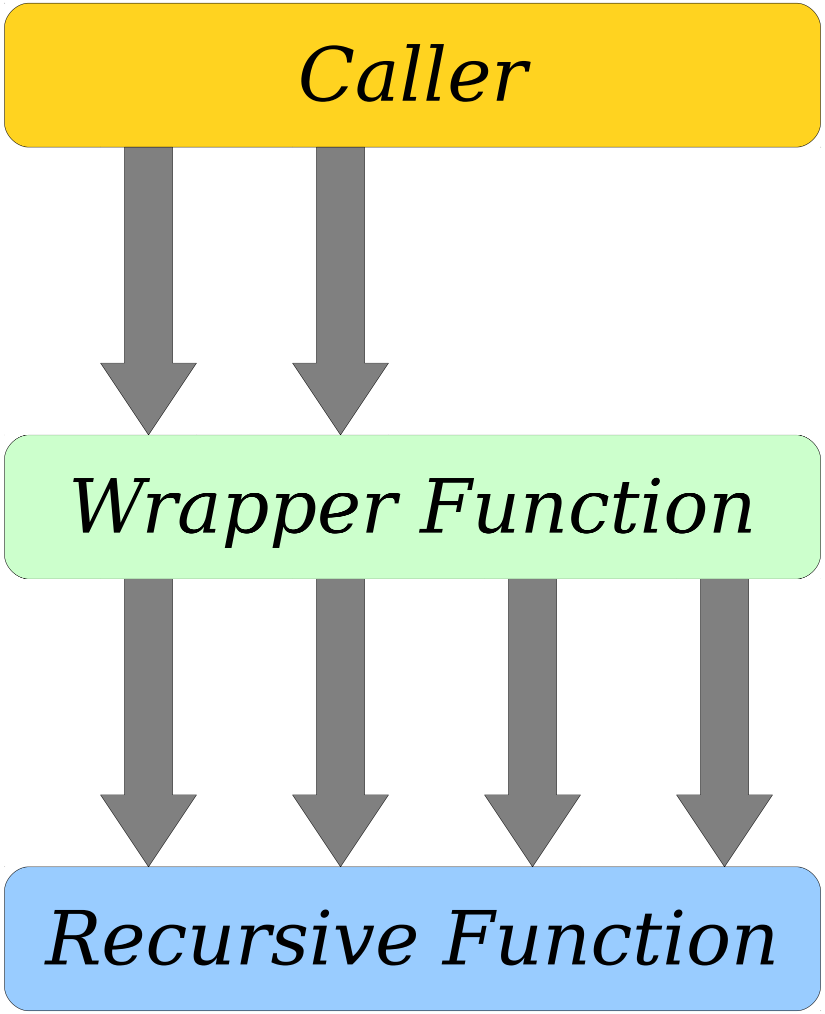


```
GWindow window(kWindowWidth, kWindowHeight);

double treeRootX  = /* Here be dragons */;
double treeRootY  = /* Dragons, dragons, dragons */;
double treeHeight = /* I like dragons! */;

int numLinesDrawn = drawTree(window,
                             treeRootX, treeRootY,
                             treeHeight);
```



Wrapper Functions

- Some recursive functions need extra arguments as part of an implementation detail.
  - In our case, the order and angle of the tree is not something we want to expose.
- A ***wrapper function*** is a function that does some initial prep work, then fires off a recursive call with the right arguments.



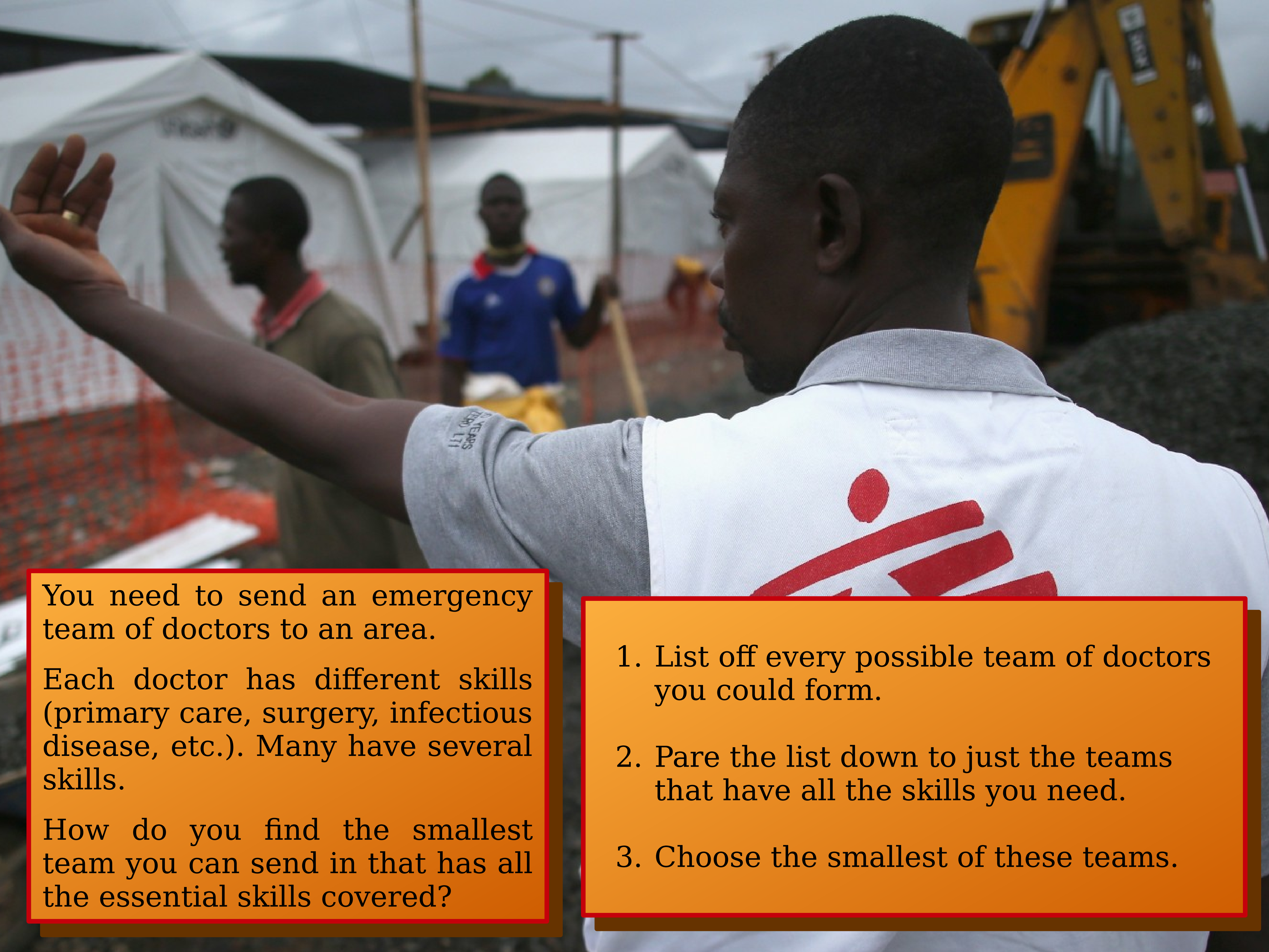
Recursive Enumeration

e·nu·mer·a·tion

noun

The act of mentioning a number of things one by one.

(Source: Google)



You need to send an emergency team of doctors to an area.

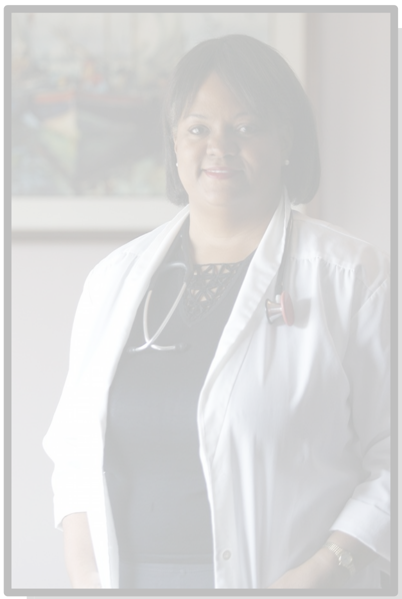
Each doctor has different skills (primary care, surgery, infectious disease, etc.). Many have several skills.

How do you find the smallest team you can send in that has all the essential skills covered?

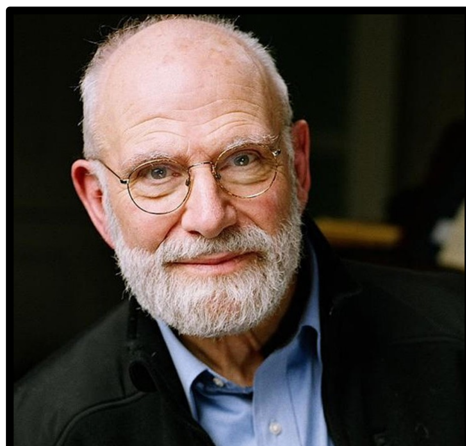
1. List off every possible team of doctors you could form.

2. Pare the list down to just the teams that have all the skills you need.

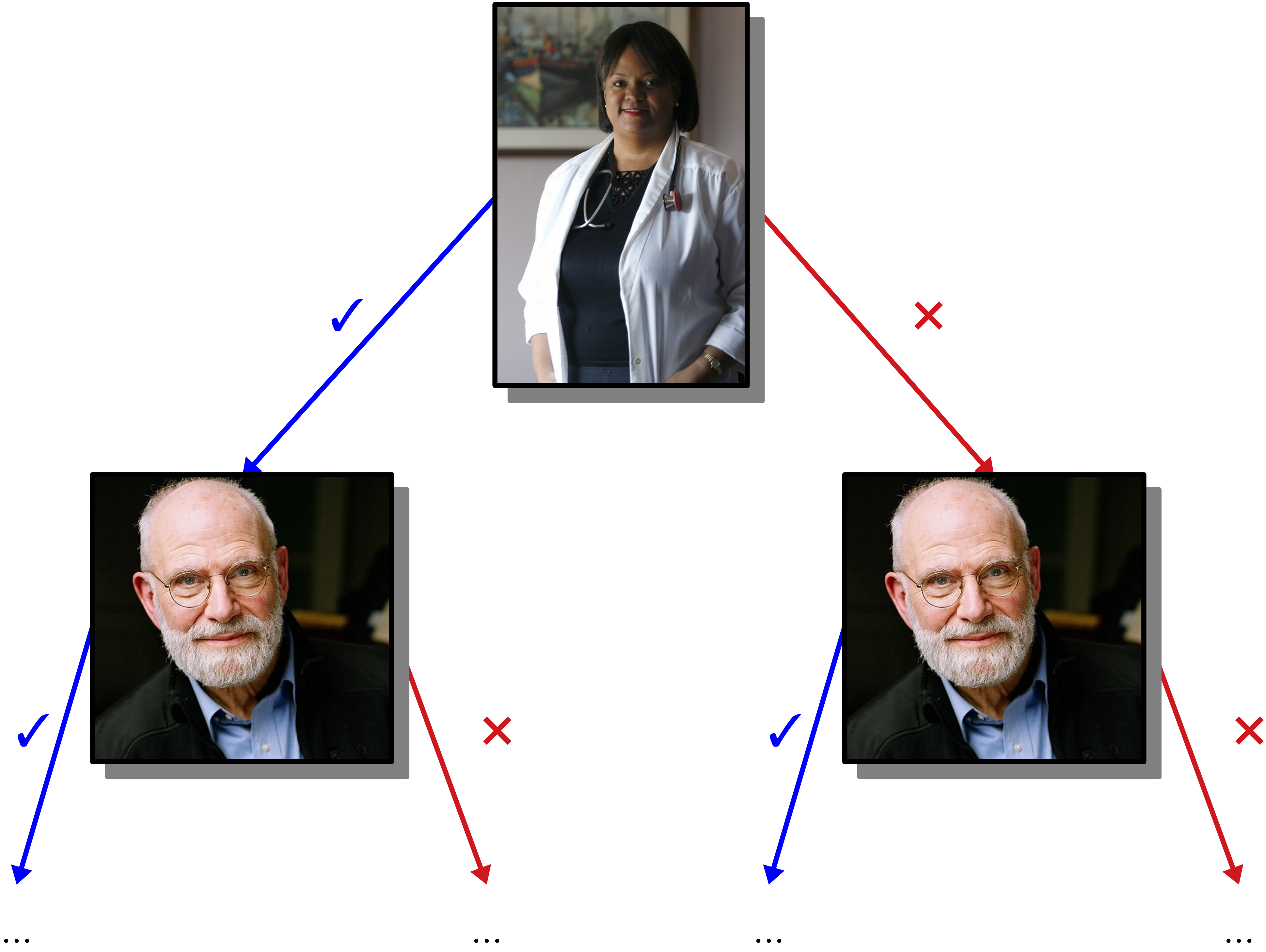
3. Choose the smallest of these teams.




NYU
Medical
Center

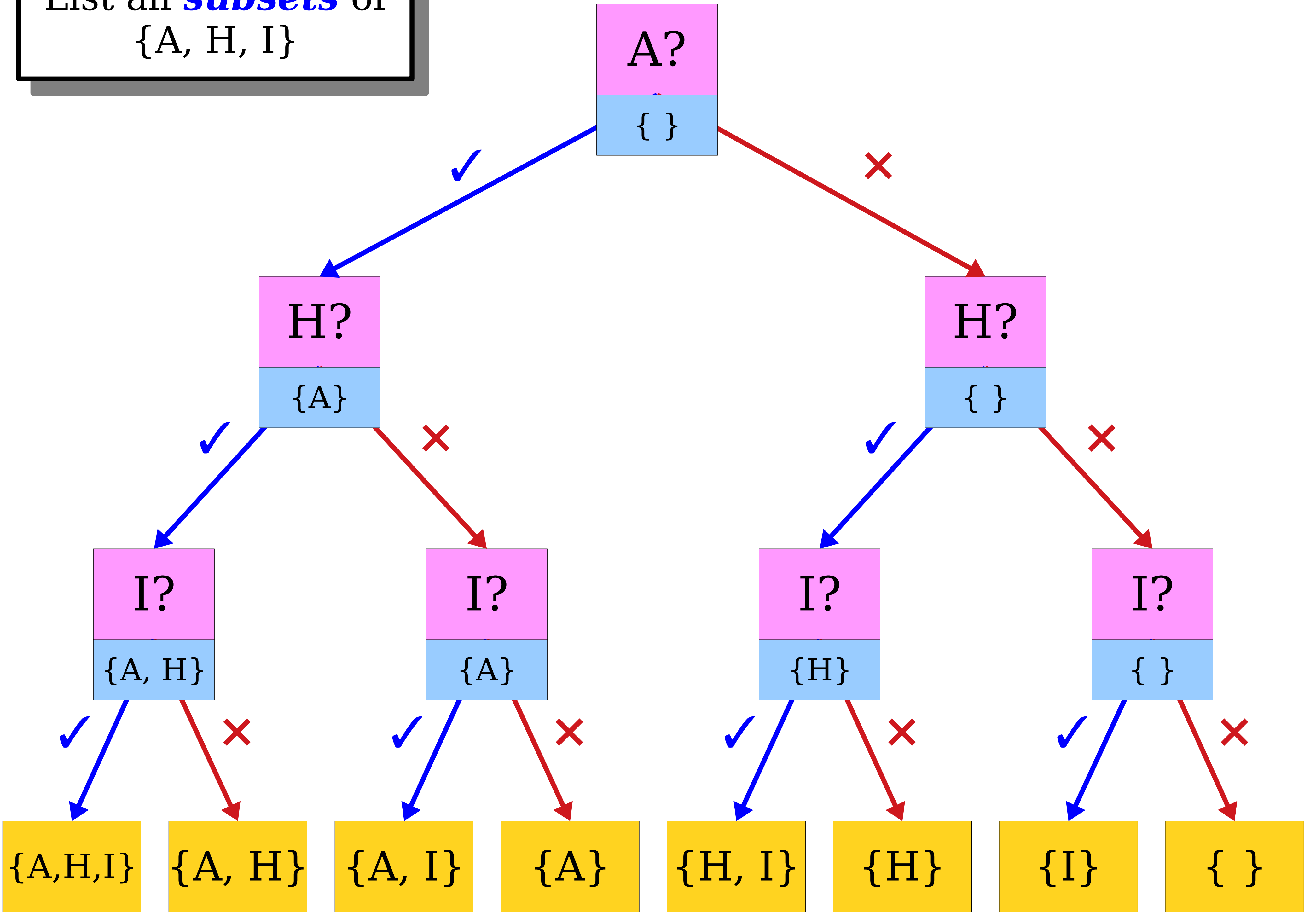



NYU
Medical
Center

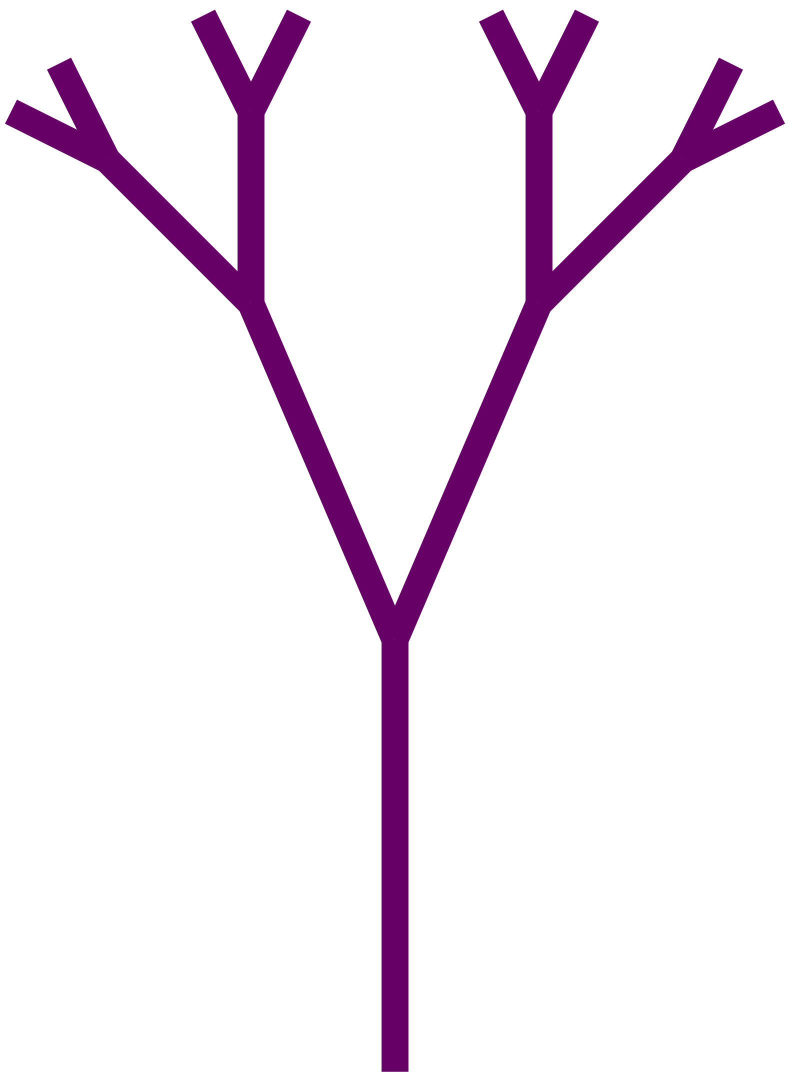


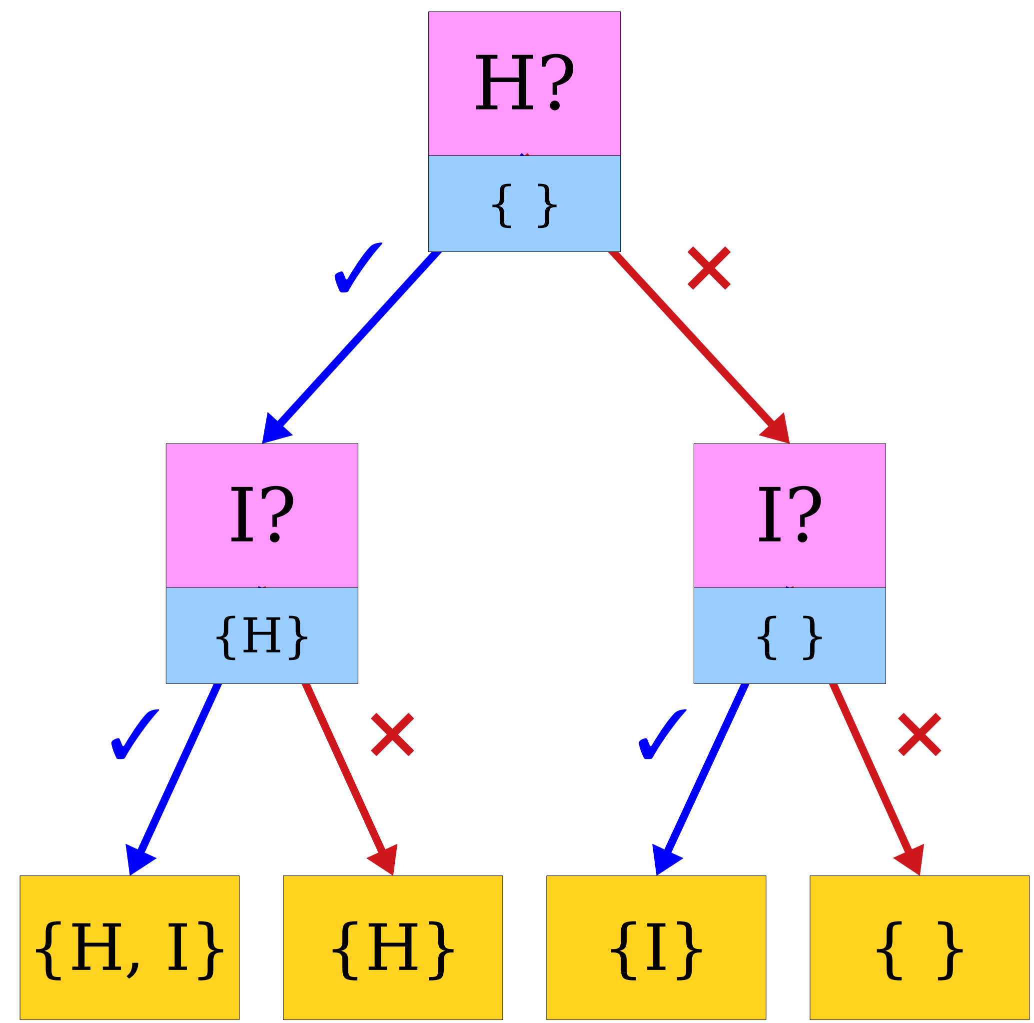
This structure is called a ***decision tree***.

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

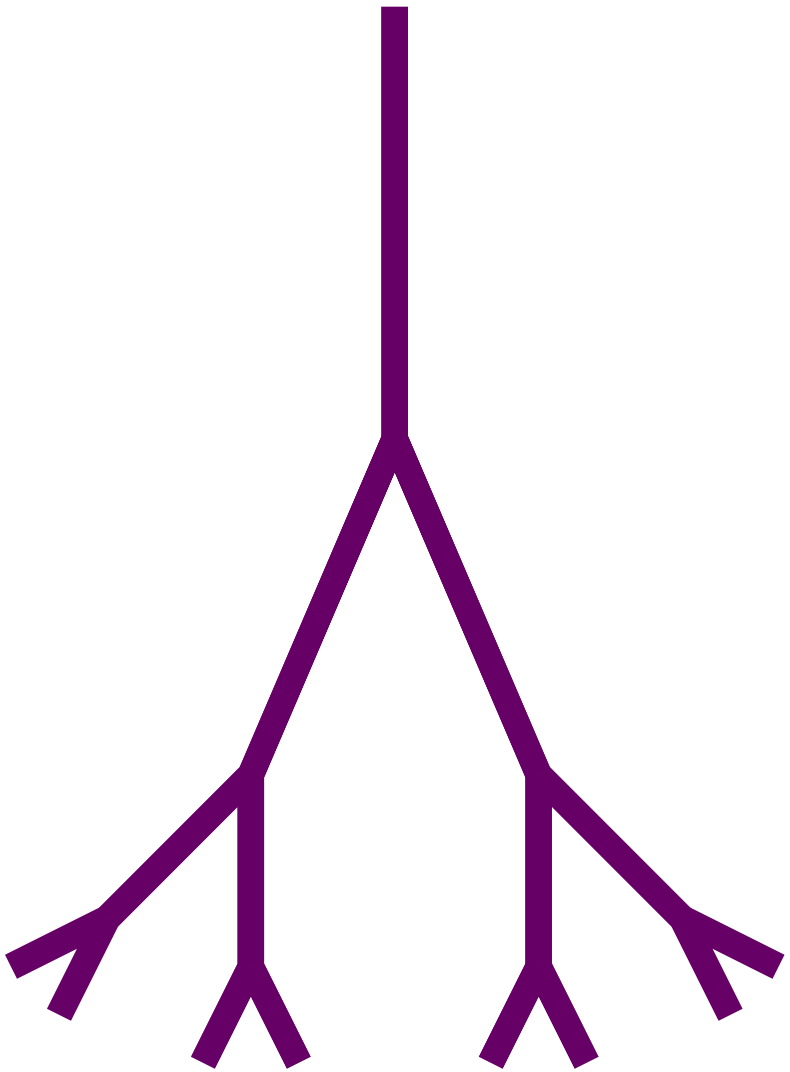


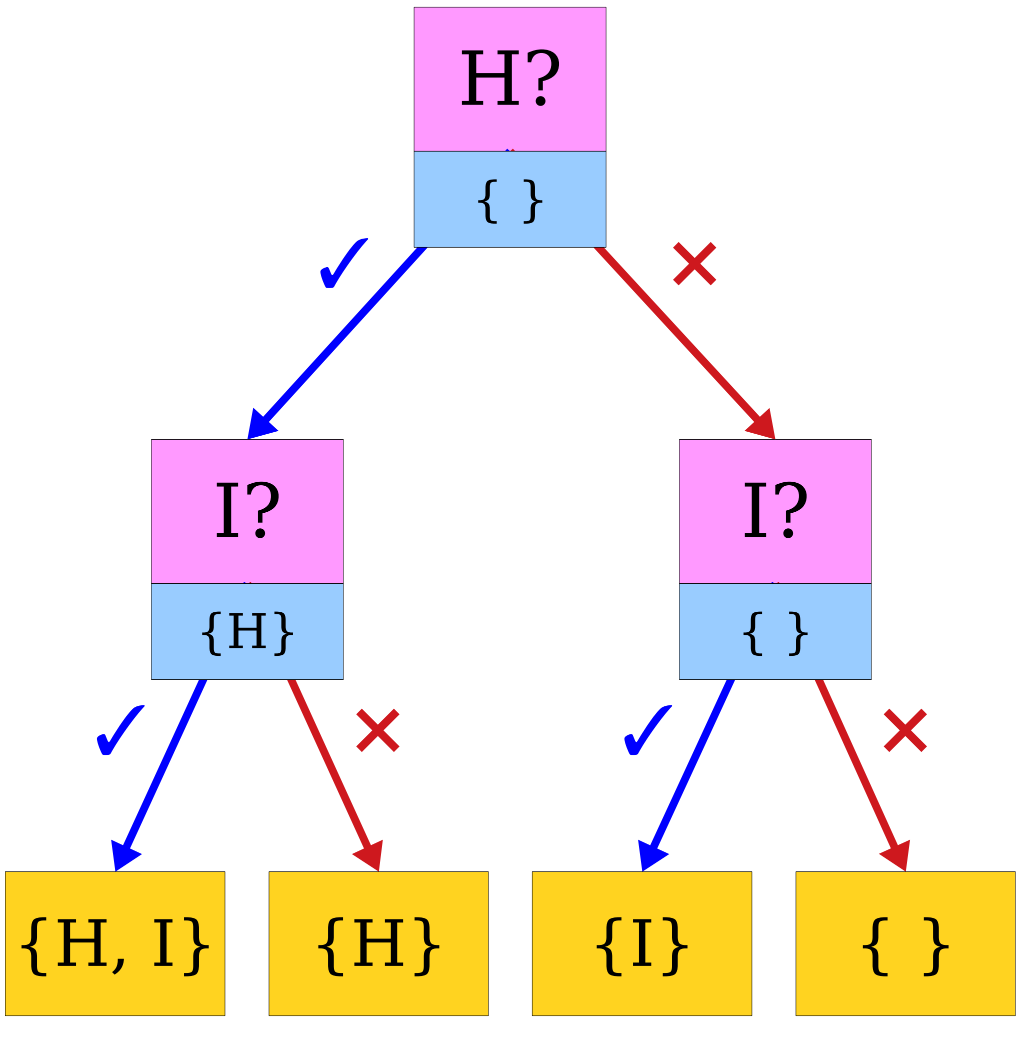
Two Trees

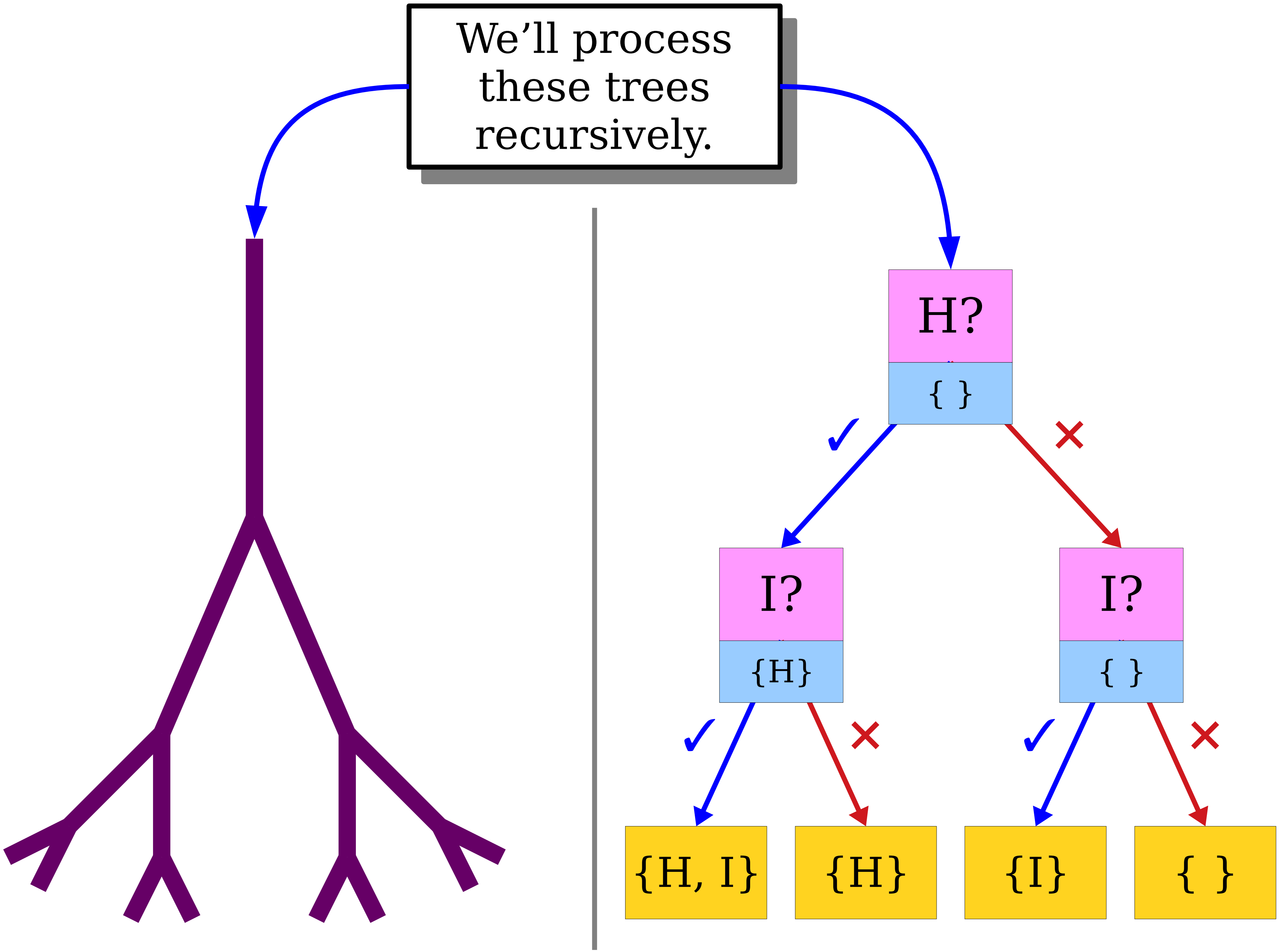




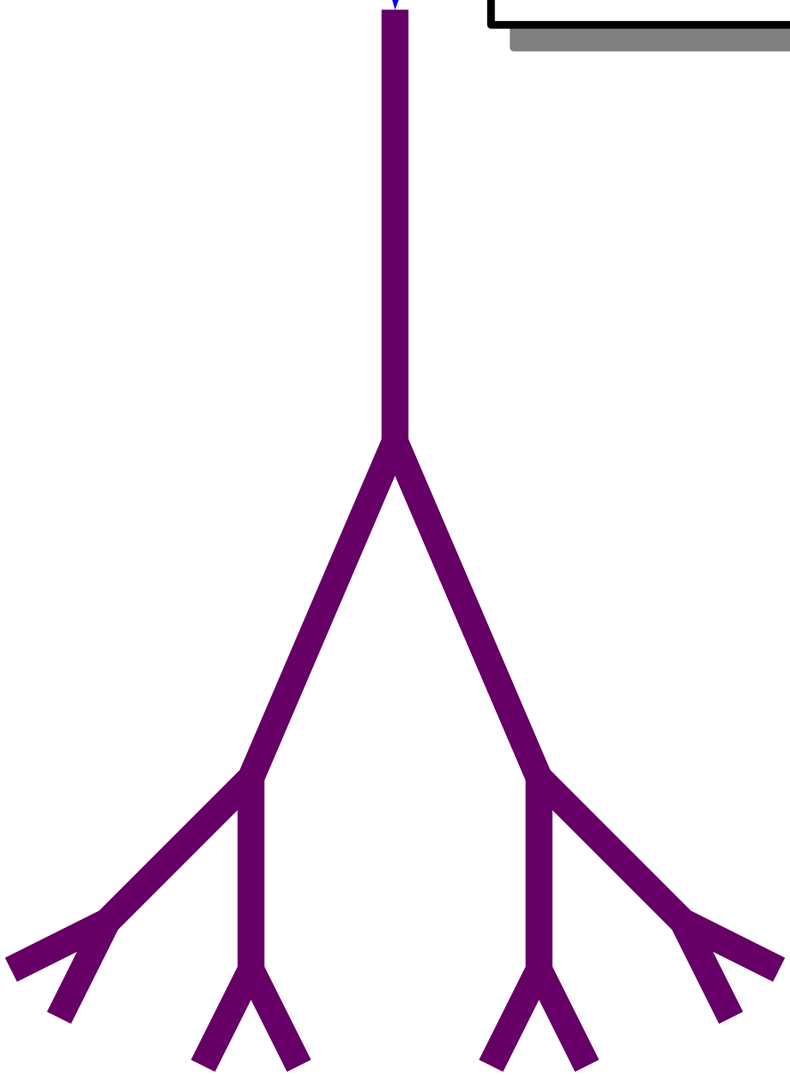
Two Trees

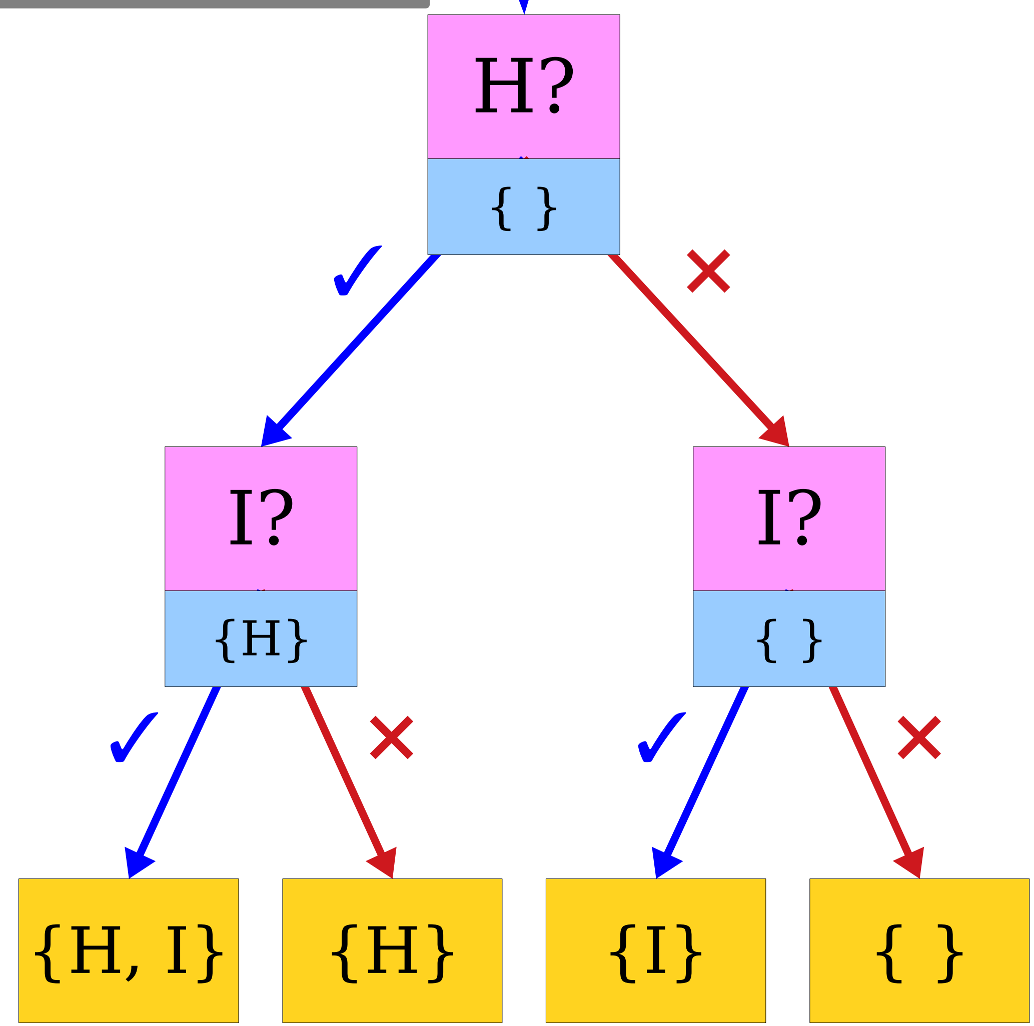





We'll process these trees recursively.
H?
{ }
I?
{H}
I?
{ }
{H, I}
{H}
{I}
{ }

Each recursive call just processes one part of the tree. The sum of all recursive calls processes the whole tree.

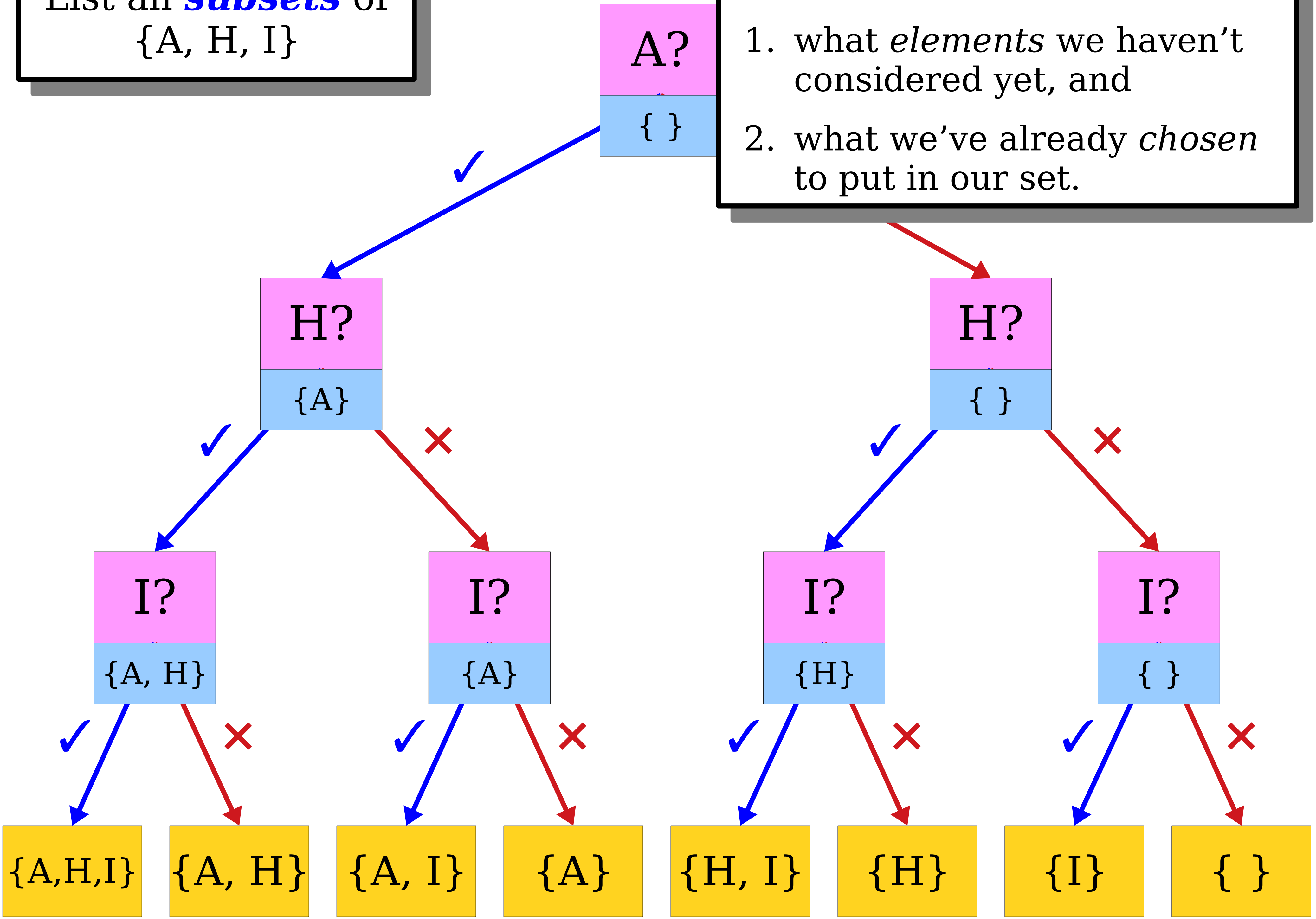




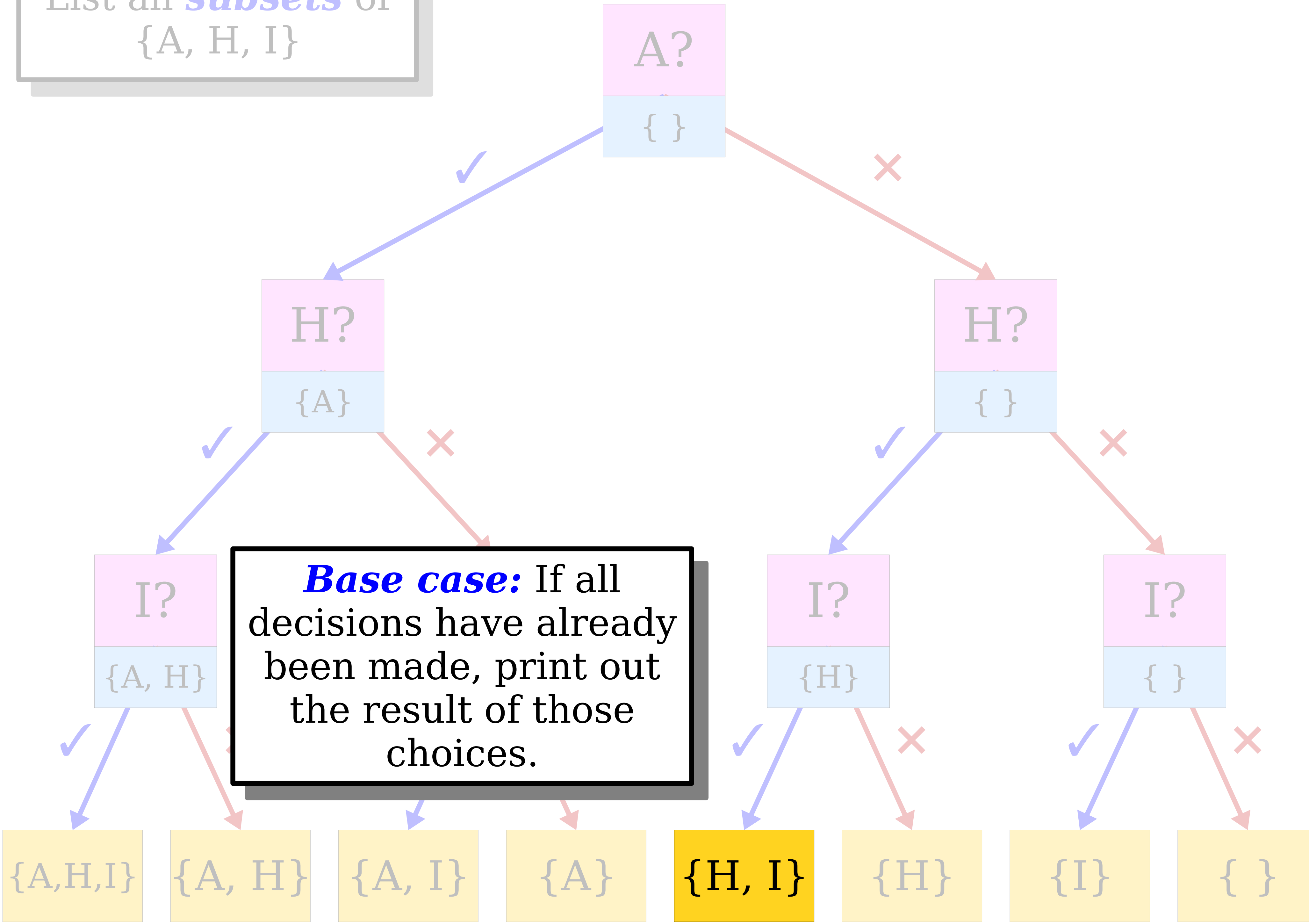
List all ***subsets*** of {A, H, I}

At each step, we need to know

1. what *elements* we haven't considered yet, and
2. what we've already *chosen* to put in our set.

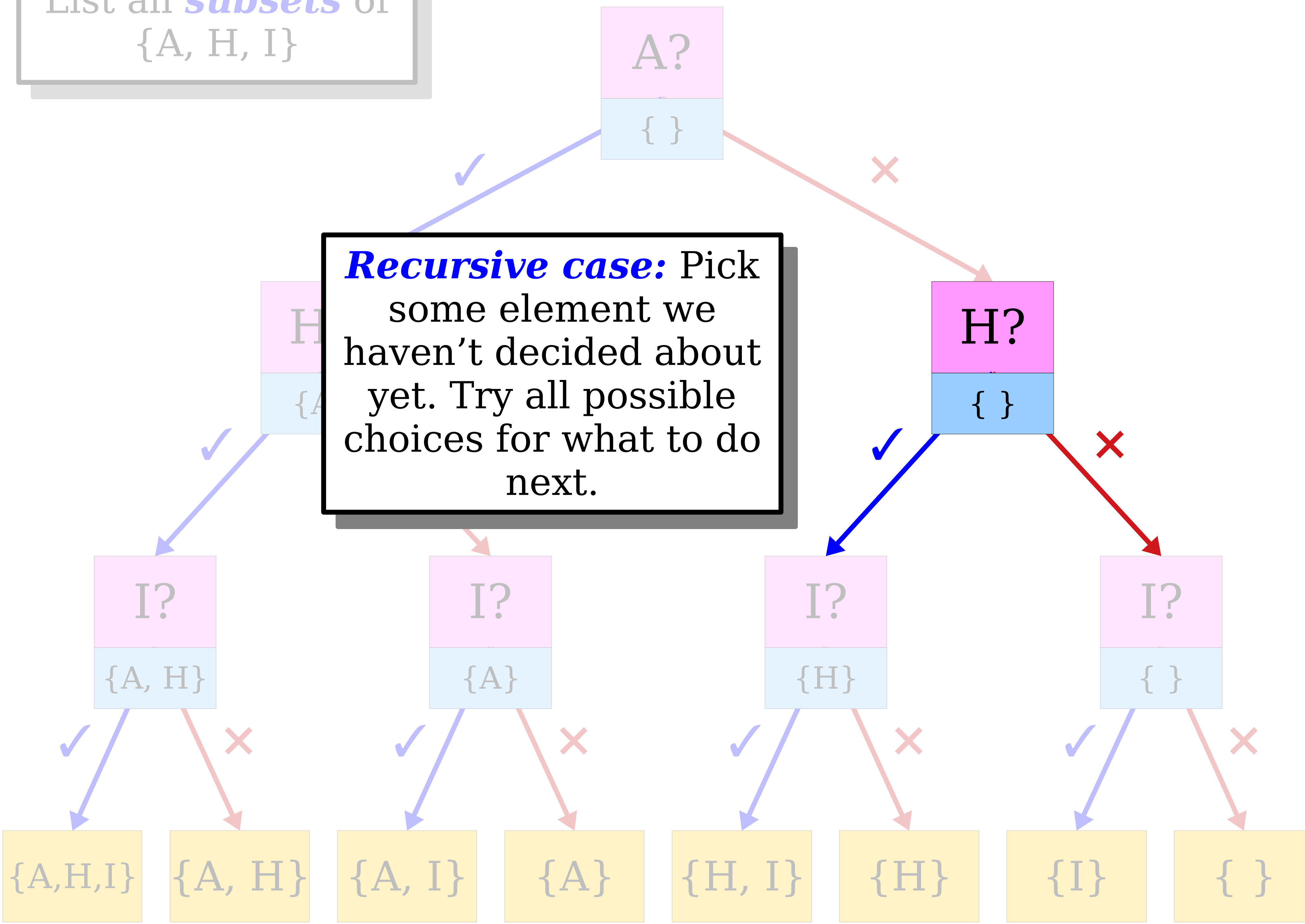


List all ***subsets*** of {A, H, I}

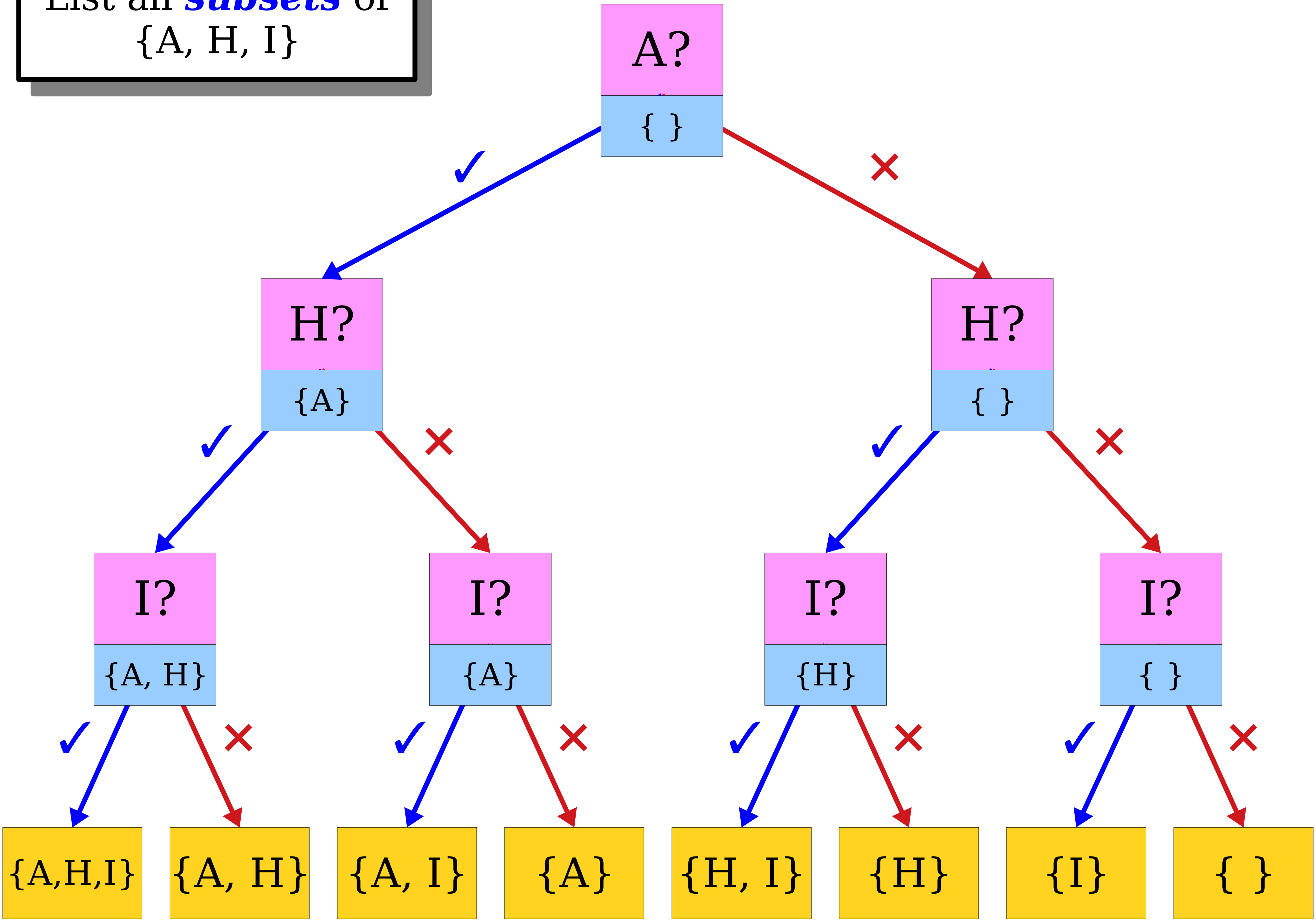


Base case: If all decisions have already been made, print out the result of those choices.

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



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



Base Case:
No decisions remain.

Decisions yet to be made

```
void listSubsetsRec(const HashSet<int>& elems,
                    const HashSet<int>& chosen) {
    if (elems.isEmpty()) {
        cout << chosen << endl;
    } else {
        int elem = elems.first();
        HashSet<int> remaining = elems - elem;

        /* Option 1: Include this element. */
        HashSet<int> includingElem = chosen + elem;
        listSubsetsRec(remaining, includingElem);

        /* Option 2: Exclude this element. */
        HashSet<int> excludingElem = chosen;
        listSubsetsRec(remaining, excludingElem);
    }
}
```

Decisions already made

Recursive Case:
Try all options for the next decision.

```
void listSubsetsRec(const HashSet<int>& elems,
                    const HashSet<int>& chosen) {

  if (elems.isEmpty()) {
    cout << chosen << endl;
  } else {
    int elem = elems.first();
    HashSet<int> remaining = elems - elem;

    /* Option 1: Include this element. */
    HashSet<int> includingElem = chosen + elem;
    listSubsetsRec(remaining, includingElem);

    /* Option 2: Exclude this element. */
    HashSet<int> excludingElem = chosen;
    listSubsetsRec(remaining, excludingElem);
  }
}
```

```
void listSubsetsRec(const HashSet<int>& elems,
                    const HashSet<int>& chosen) {

  if (elems.isEmpty()) {
    cout << chosen << endl;
  } else {
    int elem = elems.first();
    HashSet<int> remaining = elems - elem;

    /* Option 1: Include this element. */
    listSubsetsRec(remaining, chosen + elem);

    /* Option 2: Exclude this element. */
    listSubsetsRec(remaining, chosen);
  }
}
```

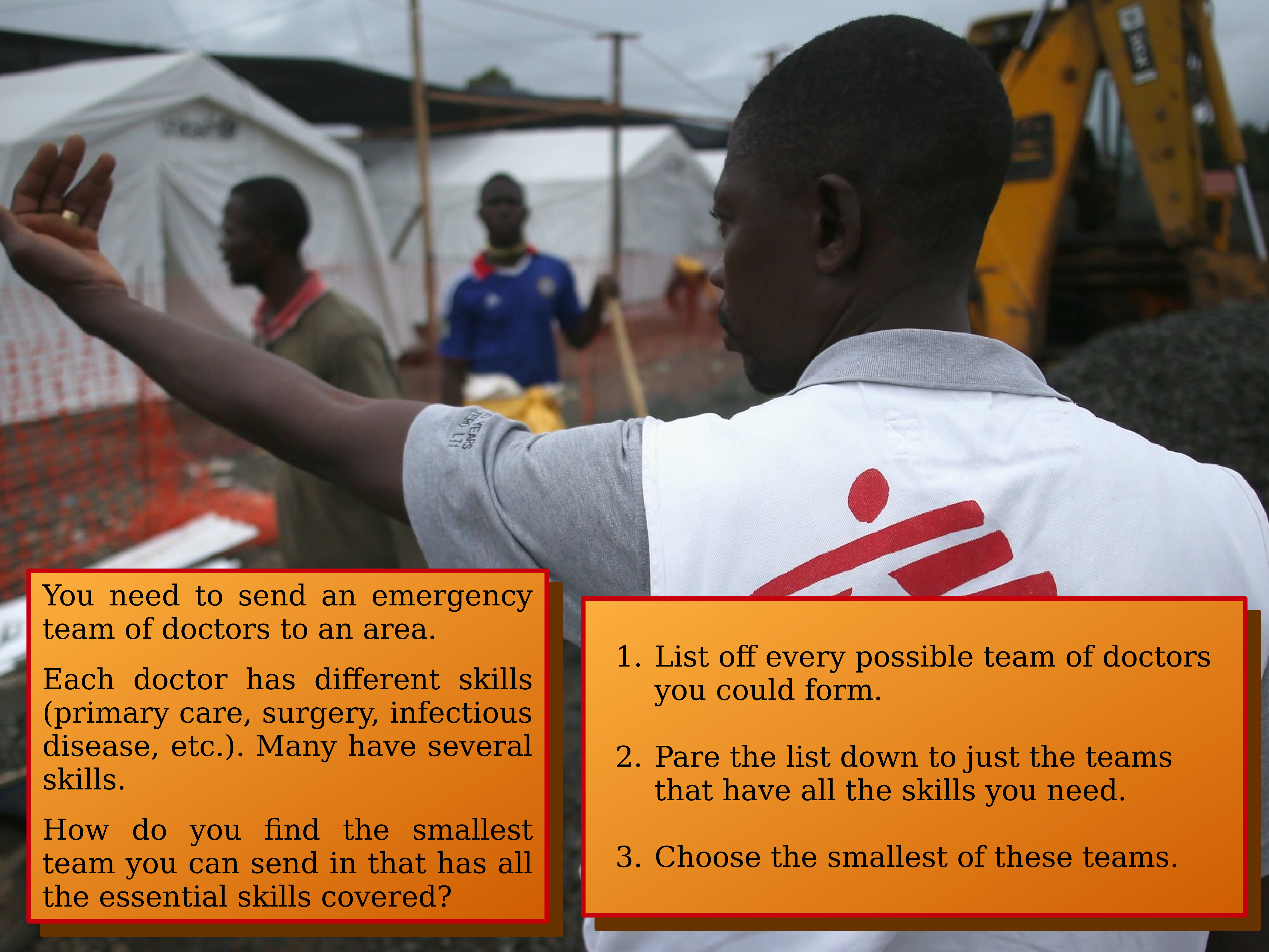
Time-Out for Announcements!

Assignment 3

- Assignment 3 (***Recursion!***) goes out today. It's due, as usual, next Friday at the start of class.
- You are welcome to work in pairs on this assignment. As a reminder:
  - You can only partner with someone who is in your discussion section.
  - You are ***strongly encouraged*** to physically sit at the same computer and work on the assignment at the same time, bouncing ideas off each other.
  - You are ***strongly discouraged*** from splitting the work in half and rejoining at the end. This is an extremely bad idea.
- Assignment 2 was due at the start of class today. Feel free to use a late period to extend the deadline to Monday if you need more time.

(The Curtain Rises for Act II)

Making Every Subset



You need to send an emergency team of doctors to an area.

Each doctor has different skills (primary care, surgery, infectious disease, etc.). Many have several skills.

How do you find the smallest team you can send in that has all the essential skills covered?

1. List off every possible team of doctors you could form.
2. Pare the list down to just the teams that have all the skills you need.
3. Choose the smallest of these teams.

```
void listSubsetsOf(const HashSet<int>& elems);
```

Right now, these get printed to the console.

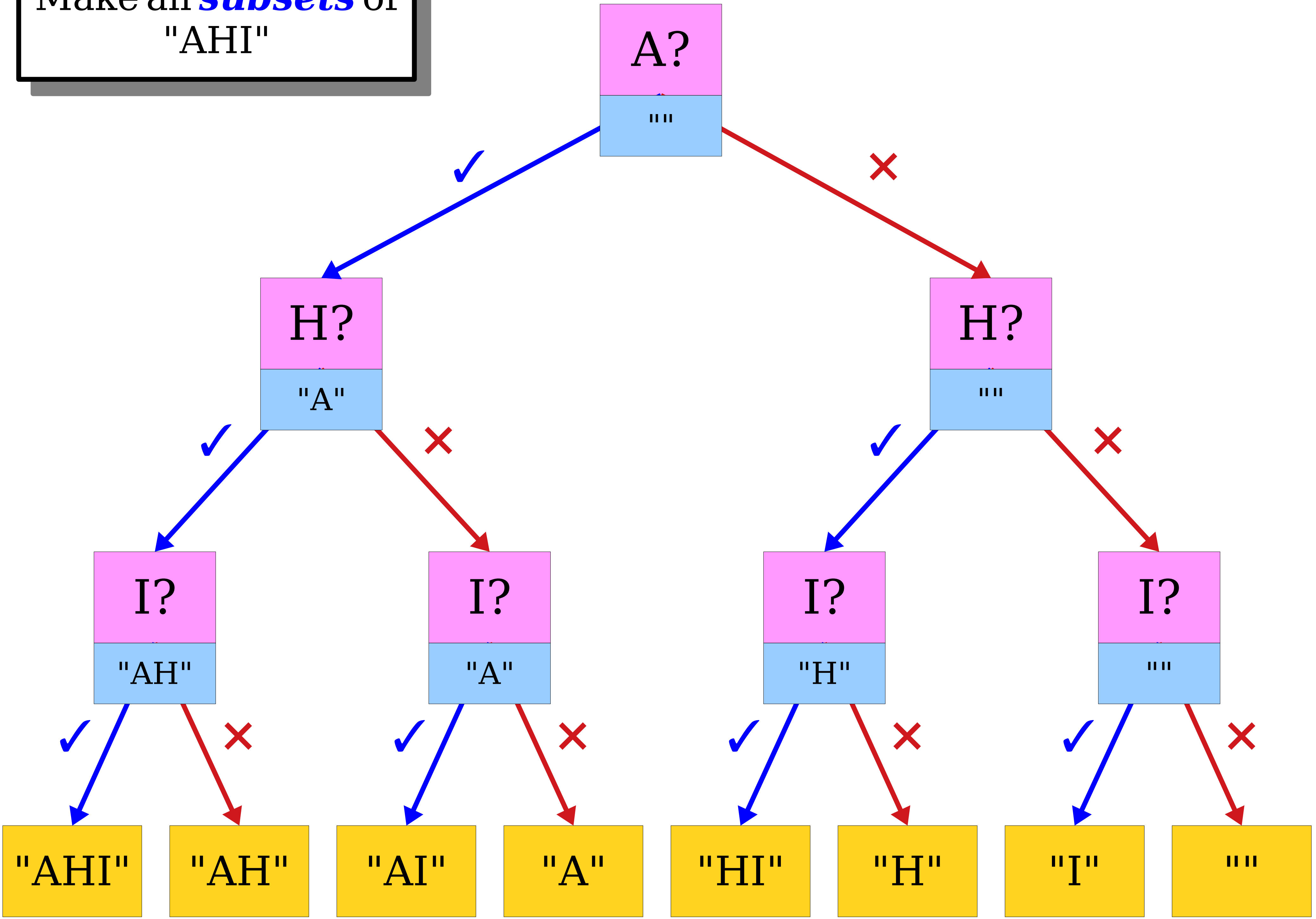
What if we want this function to hand back a list of all the subsets?

```cpp
HashSet<string> subsetsOf(const string& text);
```

`subsetsOf("code")` should return a set containing these strings:

```
"", "c", "o", "co", "d",
"cd", "od", "cod", "e", "ce",
"oe", "coe", "de", "cde",
"ode", "code"
```

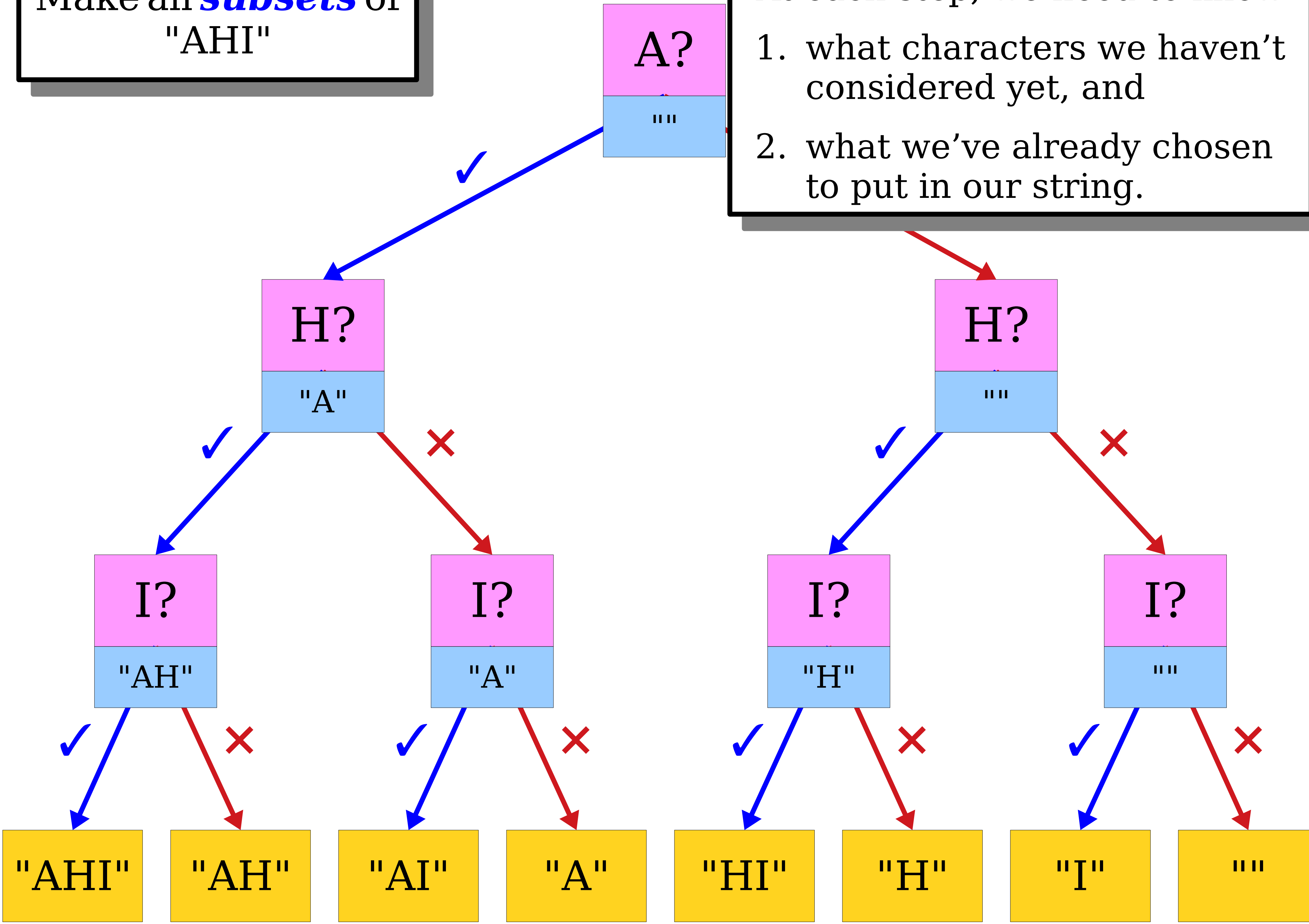
Make all ***subsets*** of "AHI"

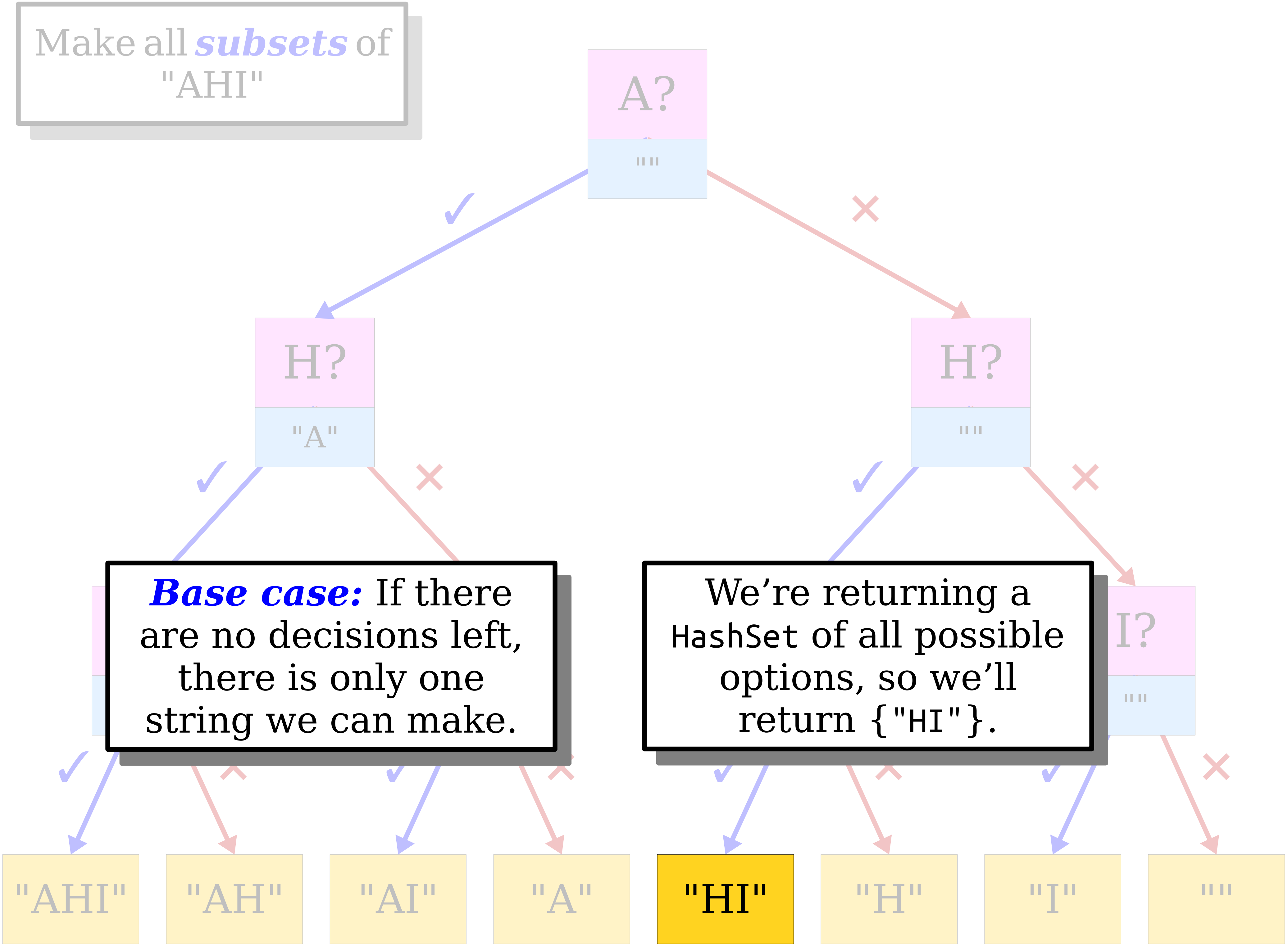


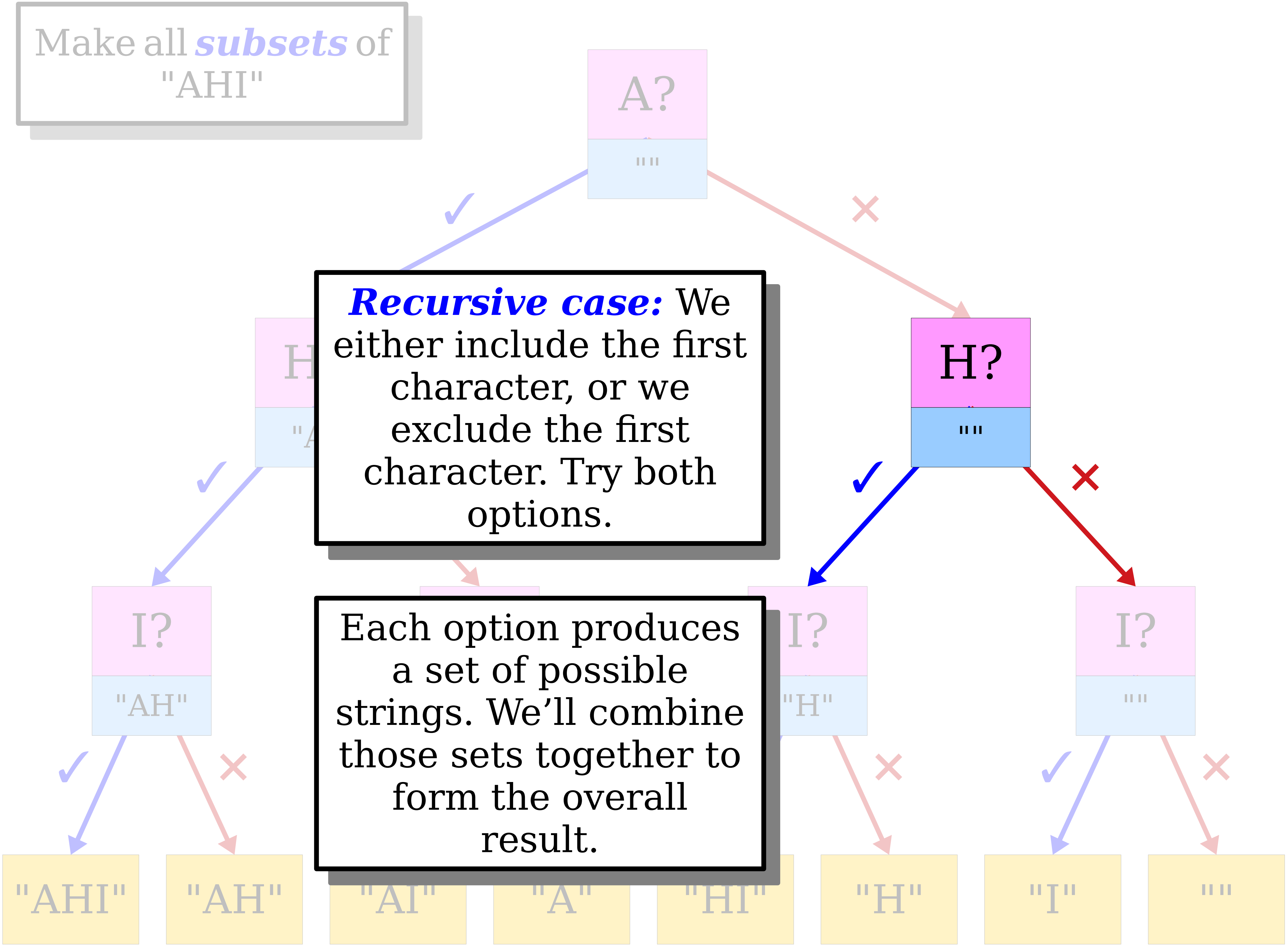
Make all ***subsets*** of "AHI"

At each step, we need to know

1. what characters we haven't considered yet, and
2. what we've already chosen to put in our string.

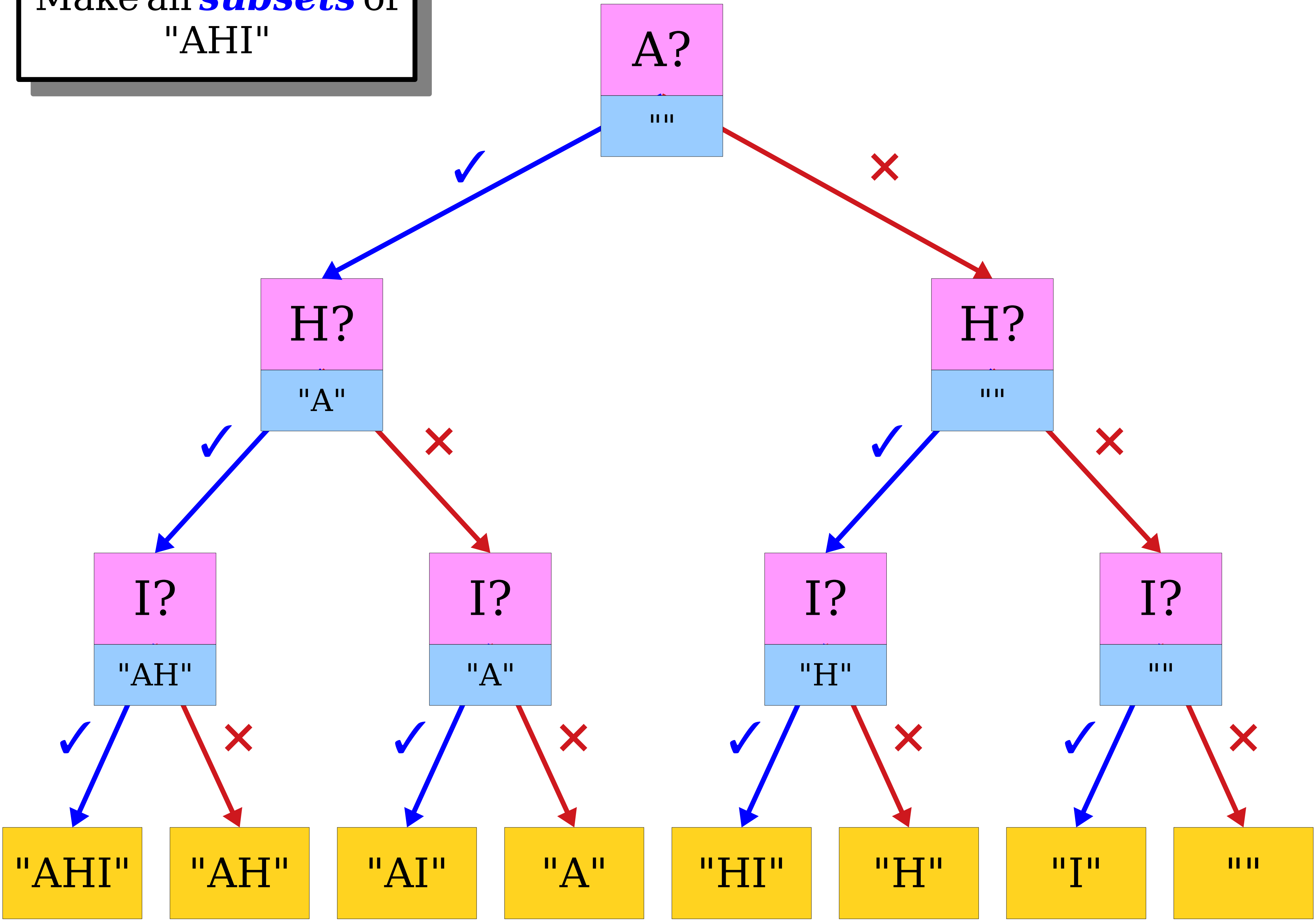



Make all *subsets* of "AHI"
A?
""
H?
"A"
H?
""
I?
""
Base case: If there are no decisions left, there is only one string we can make.
We're returning a HashSet of all possible options, so we'll return {"HI"}.
"AHI"
"AH"
"AI"
"A"
"HI"
"H"
"I"
""



Make all *subsets* of "AHI"
A?
""
H?
""
Recursive case: We either include the first character, or we exclude the first character. Try both options.
I?
"AH"
I?
"H"
I?
""
Each option produces a set of possible strings. We'll combine those sets together to form the overall result.
"AHI"
"AH"
"AI"
"A"
"HI"
"H"
"I"
""

Make all ***subsets*** of "AHI"



Base Case:
No decisions remain.

Decisions yet to be made

```
HashSet<string> subsetsRec(const string& str,
                           const string& chosen) {
  if (str == "") {
    return { chosen };
  } else {
    char ch = str[0];
    string remaining = str.substr(1);

    /* Option 1: Include this character. */
    string includingCh = chosen + ch;
    HashSet<string> with = subsetsRec(remaining, includingCh);

    /* Option 2: Exclude this character. */
    string excludingCh = chosen;
    HashSet<string> without = subsetsRec(remaining, excludingCh);

    return with + without;
  }
}
```

Decisions already made

Recursive Case:
Try all options for the next decision.

```
HashSet<string> subsetsRec(const string& str,
                           const string& chosen) {

  if (str == "") {
    return { chosen };
  } else {
    char ch = str[0];
    string remaining = str.substr(1);

    /* Option 1: Include this character. */
    string includingCh = chosen + ch;
    HashSet<string> with = subsetsRec(remaining, includingCh);

    /* Option 2: Exclude this character. */
    string excludingCh = chosen;
    HashSet<string> without = subsetsRec(remaining, excludingCh);

    return with + without;
  }
}
```

```
HashSet<string> subsetsRec(const string& str,
                           const string& chosen) {

  if (str == "") {
    return { chosen };
  } else {
    string remaining = str.substr(1);

    /* Either include the first character, or don't. */
    return subsetsRec(remaining, chosen + str[0]) +
           subsetsRec(remaining, chosen);

  }
}
```

Your Action Items

- ***Read Chapter 8.***
  - There's a lot of great information there about recursive problem-solving, and it's a great resource.
- ***Start Assignment 3.***
  - Aim to complete the Sierpinski Triangle and Human Pyramids, and try starting What Are YOU Doing?

Next Time

- ***Iteration + Recursion***
  - Combining two techniques together.
- ***Enumerating Permutations***
  - What order should we do these tasks in?