

Collections, Part Two

Outline for Today

- ***The Vector type***
 - Storing sequences.
- ***Reference Parameters***
 - A key part of C++ programming.
- ***Recursion on Vectors***
 - A problem with cell towers.

Vector

Vector

- The **Vector** is a collection class representing a list of things.
 - Similar to Java's ArrayList type.
- Syntax is pretty friendly:
 - Read/write an element: **vec**[*index*]
 - Append elements: **vec** += *a, b, c, d*;
 - Remove elements: **vec**.remove(*index*);
 - Check the size: **vec**.size()
 - Loop over elements: **for** (*T elem*: **vec**) { ... }

Reference Parameters in C++

Pass-by-Value

- In C++, objects are passed into functions by *value*. The function gets its own local copy of the argument to work with.
 - There's a cool nuance where this isn't 100% true; come talk to me after class if you're curious!
- You can see this by running some code samples with our new `Vector` type.

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
  
    growUp(moonlight);  
  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|


```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
void growUp(Vector<string> characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

characters

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin"

```
void growUp(Vector<string> characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

characters "Little" "Teresa" "Kevin"

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin"

```
void growUp(Vector<string> characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

characters "Little" "Teresa" "Kevin" "Paula"

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin"

```
void growUp(Vector<string> characters) {  
    characters += "Paula",  
    characters[0] = "Chiron";  
}
```

characters "Little" "Teresa" "Kevin" "Paula"

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin"

```
void growUp(Vector<string> characters) {  
    characters += "Paula",  
    characters[0] = "Chiron";  
}
```

characters "Chiron" "Teresa" "Kevin" "Paula"

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
void growUp(Vector<string> characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

characters

| | | | |
|----------|----------|---------|---------|
| "Chiron" | "Teresa" | "Kevin" | "Paula" |
|----------|----------|---------|---------|

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
  
    growUp(moonlight);  
  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin"

Pass-by-Reference

- In C++, there's the option to pass parameters into function by reference.
- This doesn't send a copy of the argument - it really sends the actual, honest-to-goodness argument into the function.
- To declare a function that takes an argument by reference, put an ampersand after the name of the type of the argument.
- Calling a function that takes a reference looks identical to one that takes a value. You just have to know what the function expects.

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
  
    growUp(moonlight);  
  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

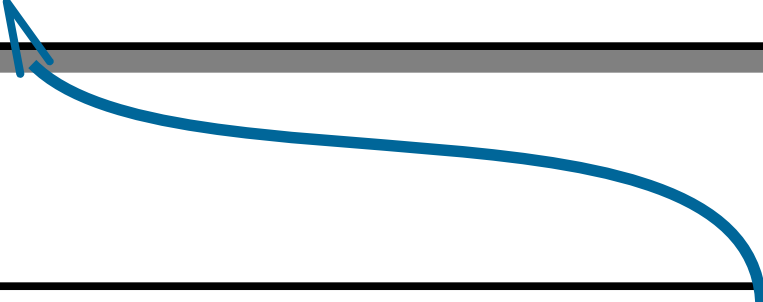
moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|



```
void growUp(Vector<string>& characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | |
|----------|----------|---------|
| "Little" | "Teresa" | "Kevin" |
|----------|----------|---------|



```
void growUp(Vector<string>& characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin" "Paula"

```
void growUp(Vector<string>& characters) {  
    characters += "Paula";  
    characters[0] = "Chiron";  
}
```

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight "Little" "Teresa" "Kevin" "Paula"

```
void growUp(Vector<string>& characters) {  
    characters += "Paula",  
    characters[0] = "Chiron";  
}
```

```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

moonlight

| | | | |
|----------|----------|---------|---------|
| "Chiron" | "Teresa" | "Kevin" | "Paula" |
|----------|----------|---------|---------|

```
void growUp(Vector<string>& characters) {  
    characters += "Paula",  
    characters[0] = "Chiron";  
}
```



```
int main() {  
    Vector<string> moonlight;  
    moonlight += "Little", "Teresa", "Kevin";  
    growUp(moonlight);  
    /* ... */  
}
```

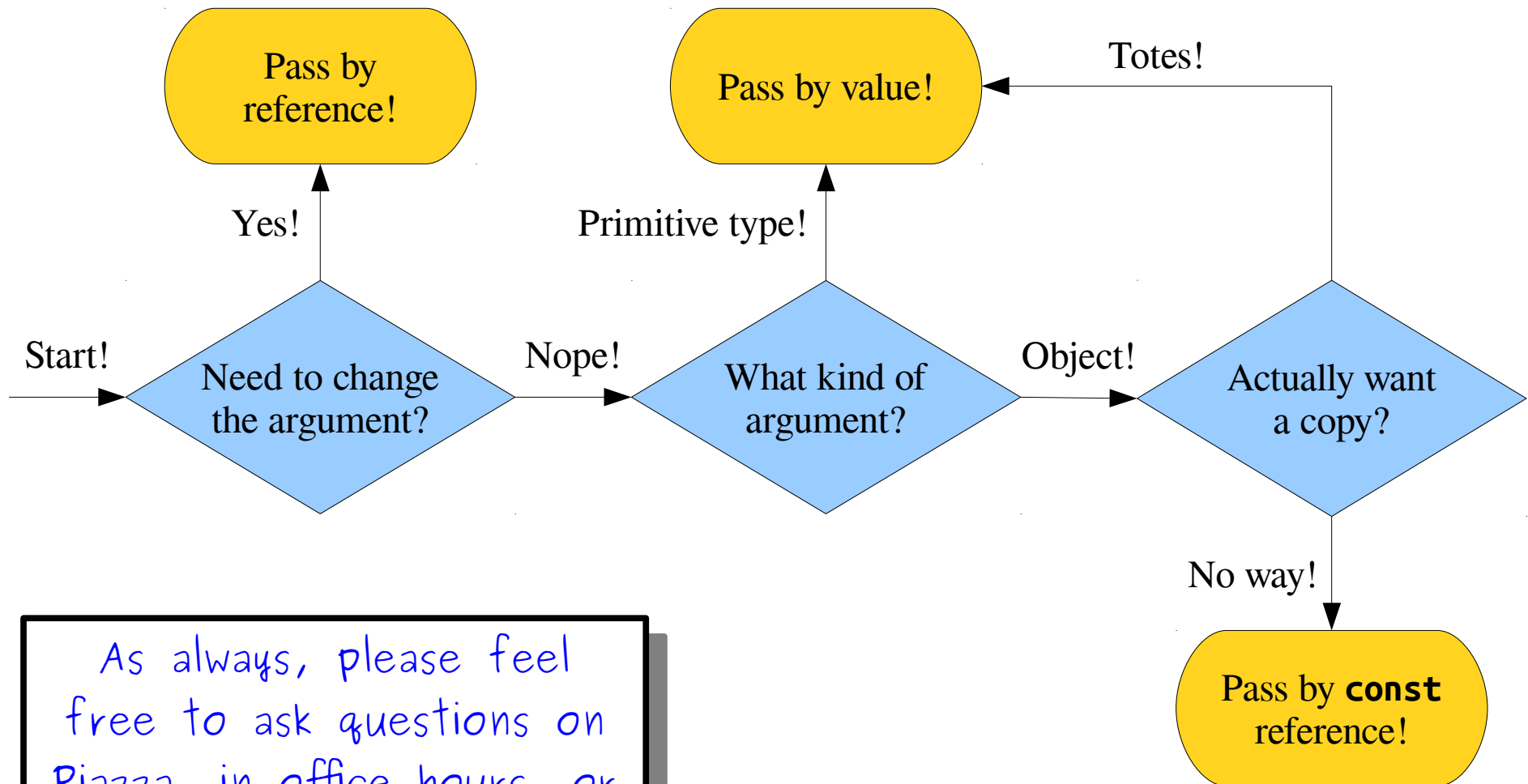
moonlight

| | | | |
|----------|----------|---------|---------|
| "Chiron" | "Teresa" | "Kevin" | "Paula" |
|----------|----------|---------|---------|

Pass-by-const-Reference

- Passing a large object (Vector, Stack, Queue, string, etc.) into a function by value can take a *lot* of time.
- Taking parameters by reference avoids making a copy, but risks that the object gets tampered with in the process.
- As a result, it's common to have functions that take objects as parameters take their argument by ***const reference***:
 - The “by reference” part avoids a copy.
 - The “const” (constant) part means that the function can't change that argument.

What Should You Use?



As always, please feel free to ask questions on Piazza, in office hours, or at the LaIR!

Recursion on Vectors

Example: Cell Tower Purchasing

Buying Cell Towers



137



42



95



272



52

Buying Cell Towers



137



42



95



272



52

Buying Cell Towers



14



22



13



25



30



11



9

Buying Cell Towers



14



22



13



25



30



11



9

Buying Cell Towers



99



100



99

Buying Cell Towers



99



100



99

How can we solve this problem?



14



22



13



25



30



11



9



14

22

13

25

30

11

9



14



13

25

30

11

9



14



~~22~~

13

25

30

11

9

Maximize what's left in here.



14



~~22~~

13

25

30

11

9

Maximize what's left in here.



14

22

13

25

30

11

9



14



~~22~~

13

25

30

11

9

Maximize what's left in here.



~~14~~

22

13

25

30

11

9



14



~~22~~

13

25

30

11

9

Maximize what's left in here.



~~14~~

22

13

25

30

11

9

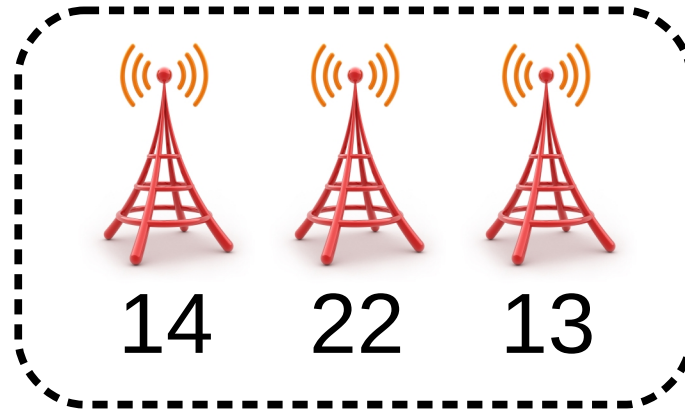
Maximize what's left in here.

The Insight

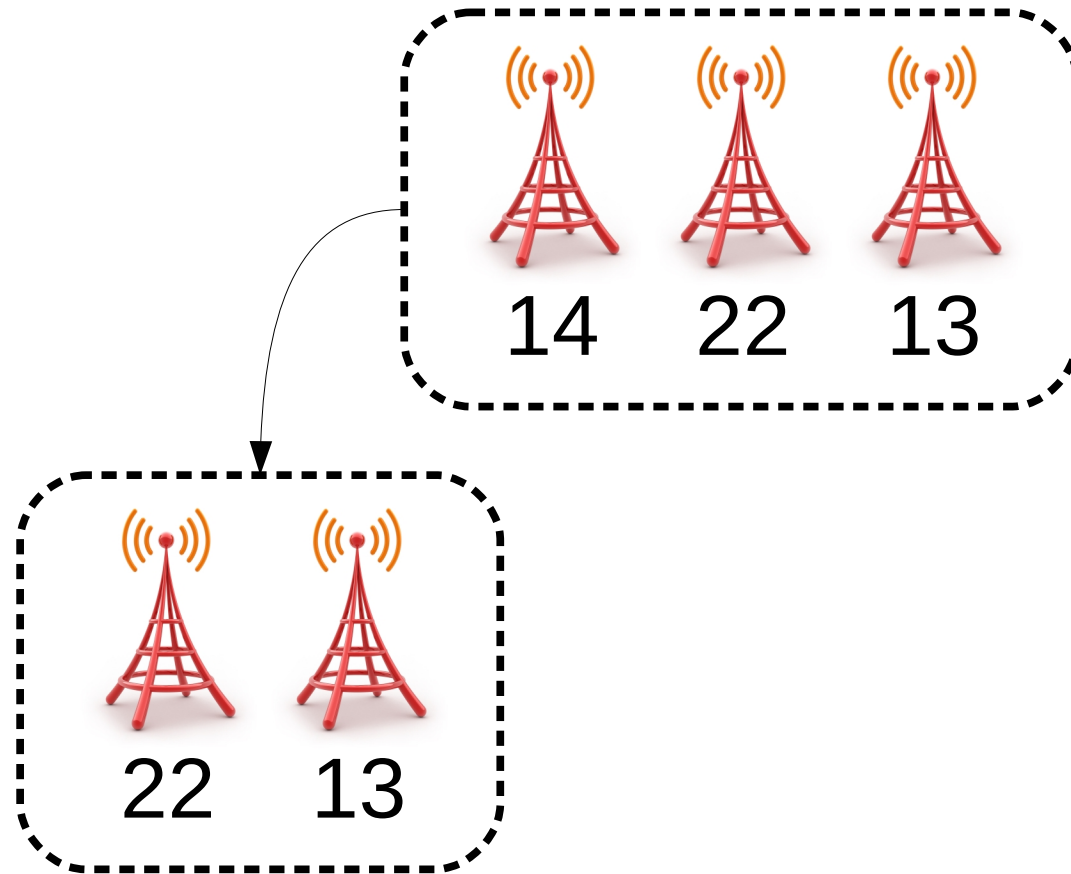
- If there's no cities, the best you can do is cover zero people.
- If there's one city, the best you can do is build a tower there and cover everyone in it.
- Otherwise, you either
 - **do** build in the first city, skip the second city, then do the best you can with the remaining cities; or
 - **don't** build in the first city, and then do the best you can with the remaining cities.

So we just try out both options and see which one is better!

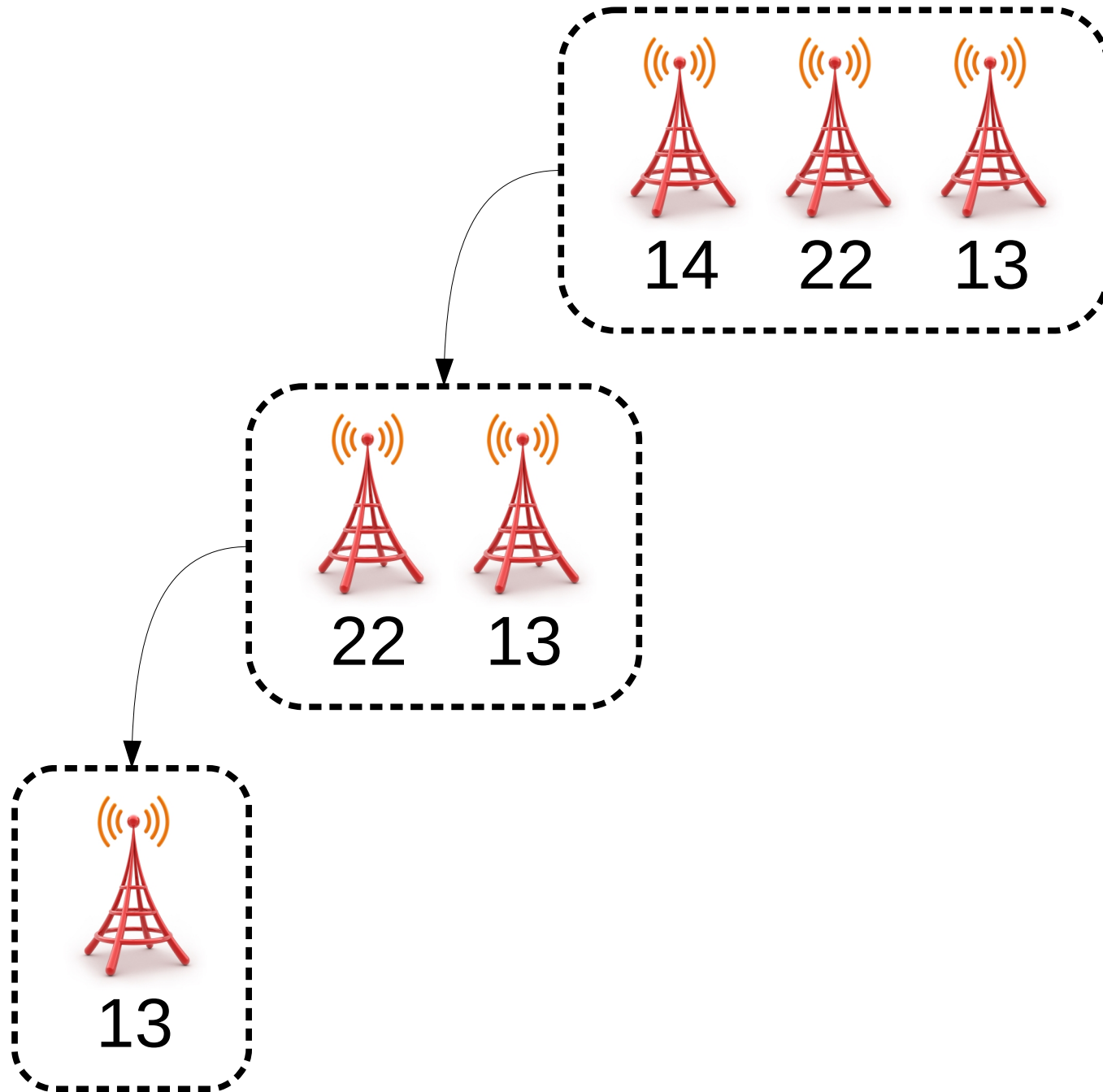
How the Recursion Works



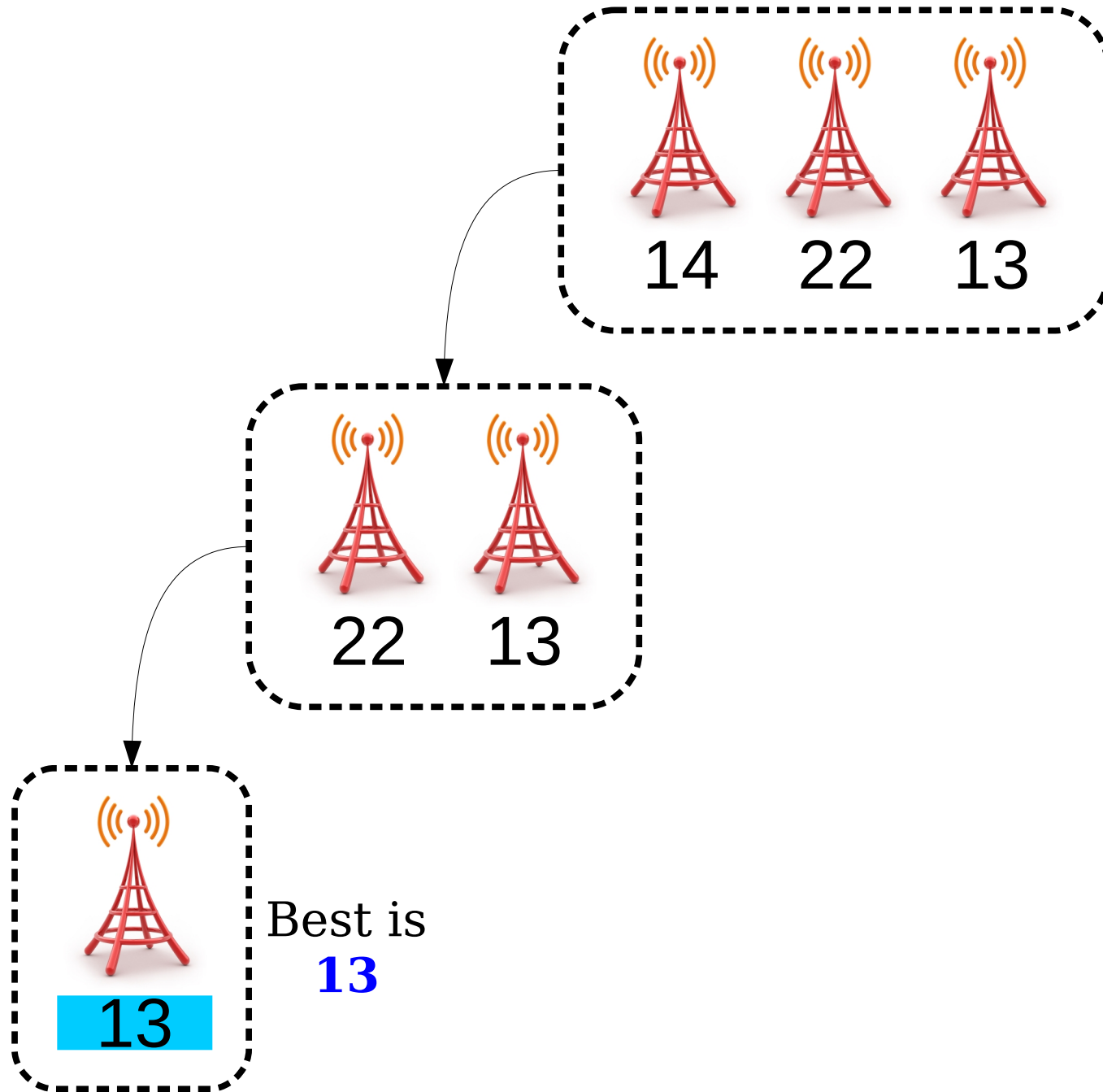
How the Recursion Works



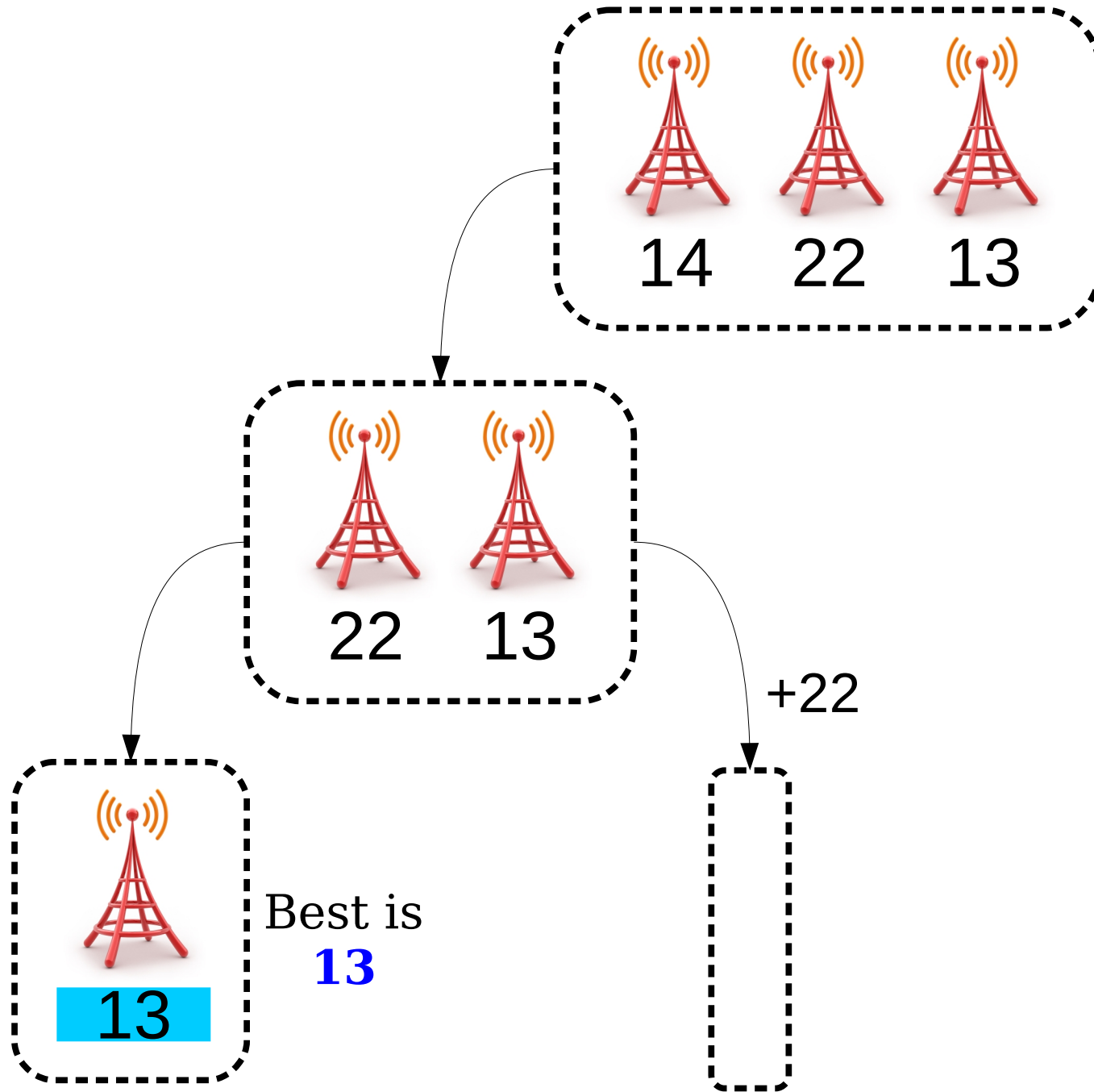
How the Recursion Works



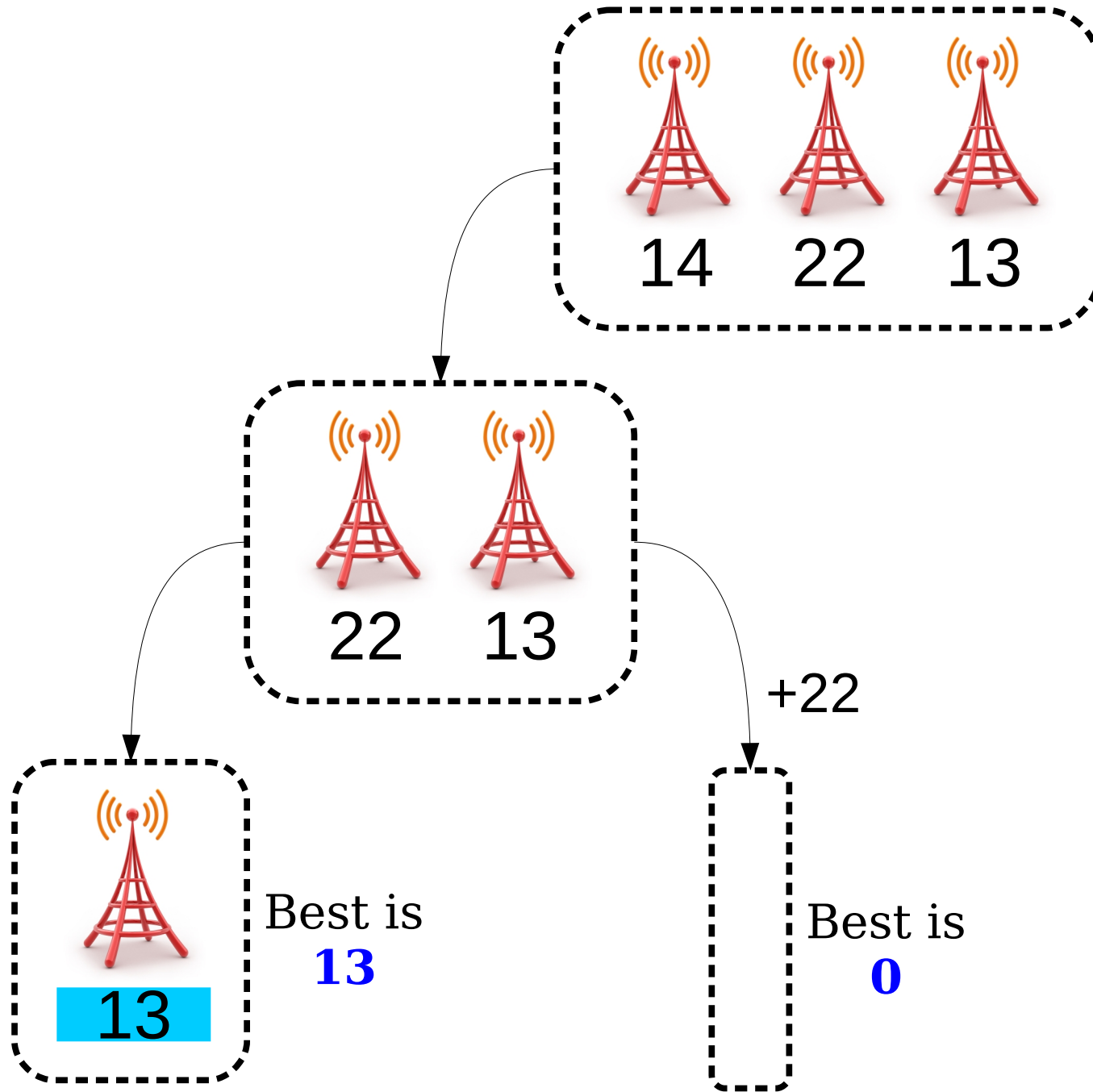
How the Recursion Works



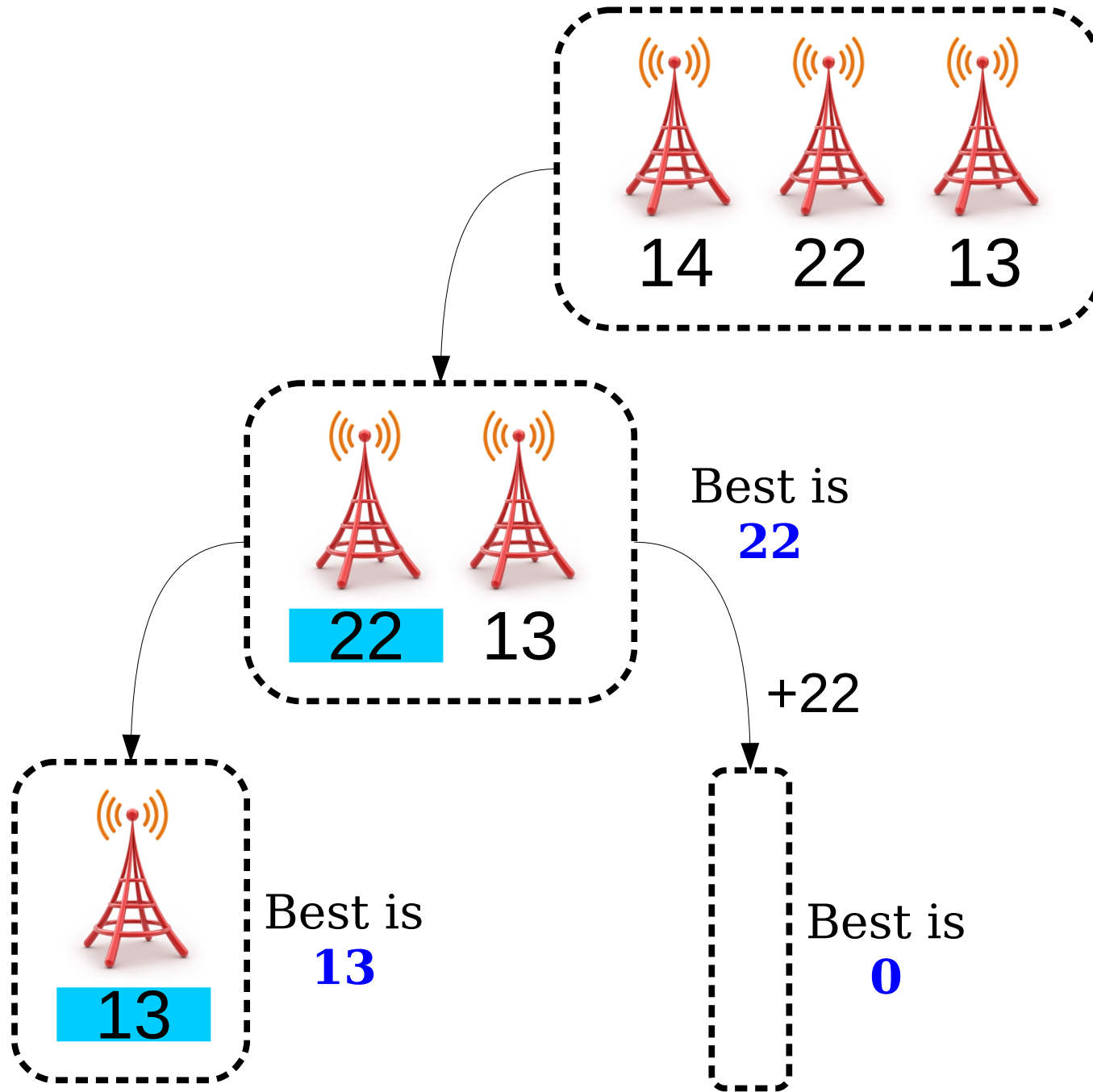
How the Recursion Works



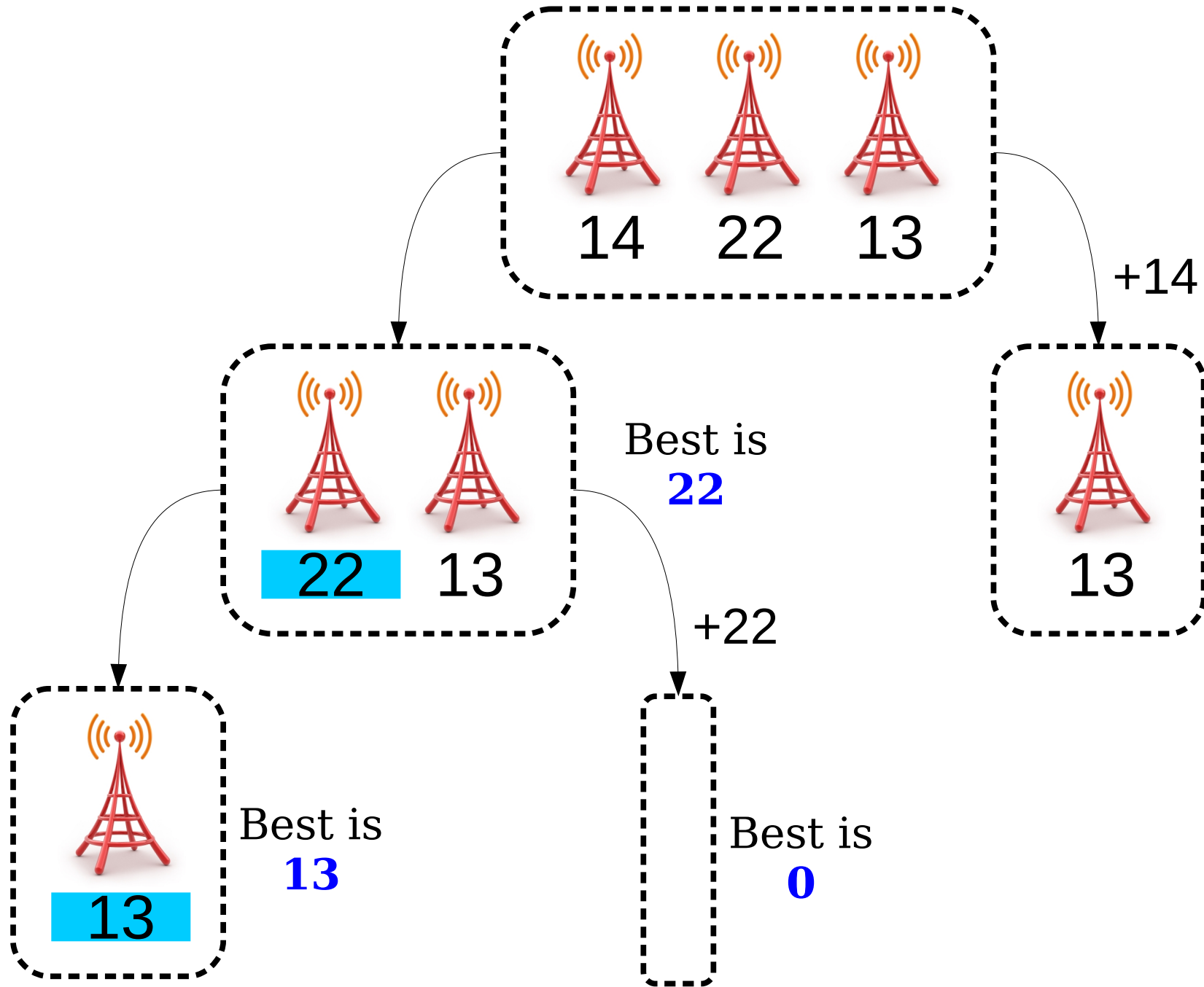
How the Recursion Works



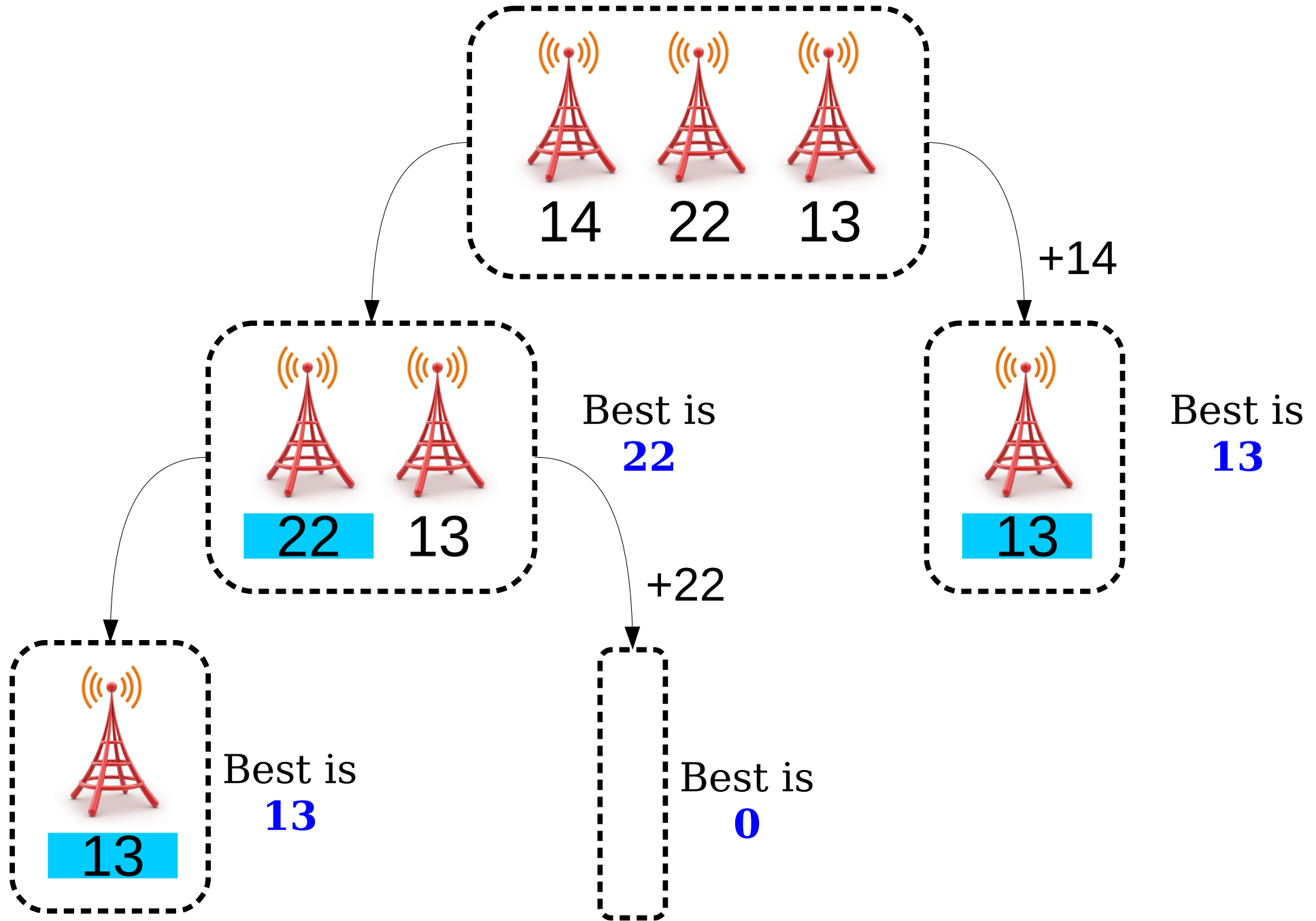
How the Recursion Works



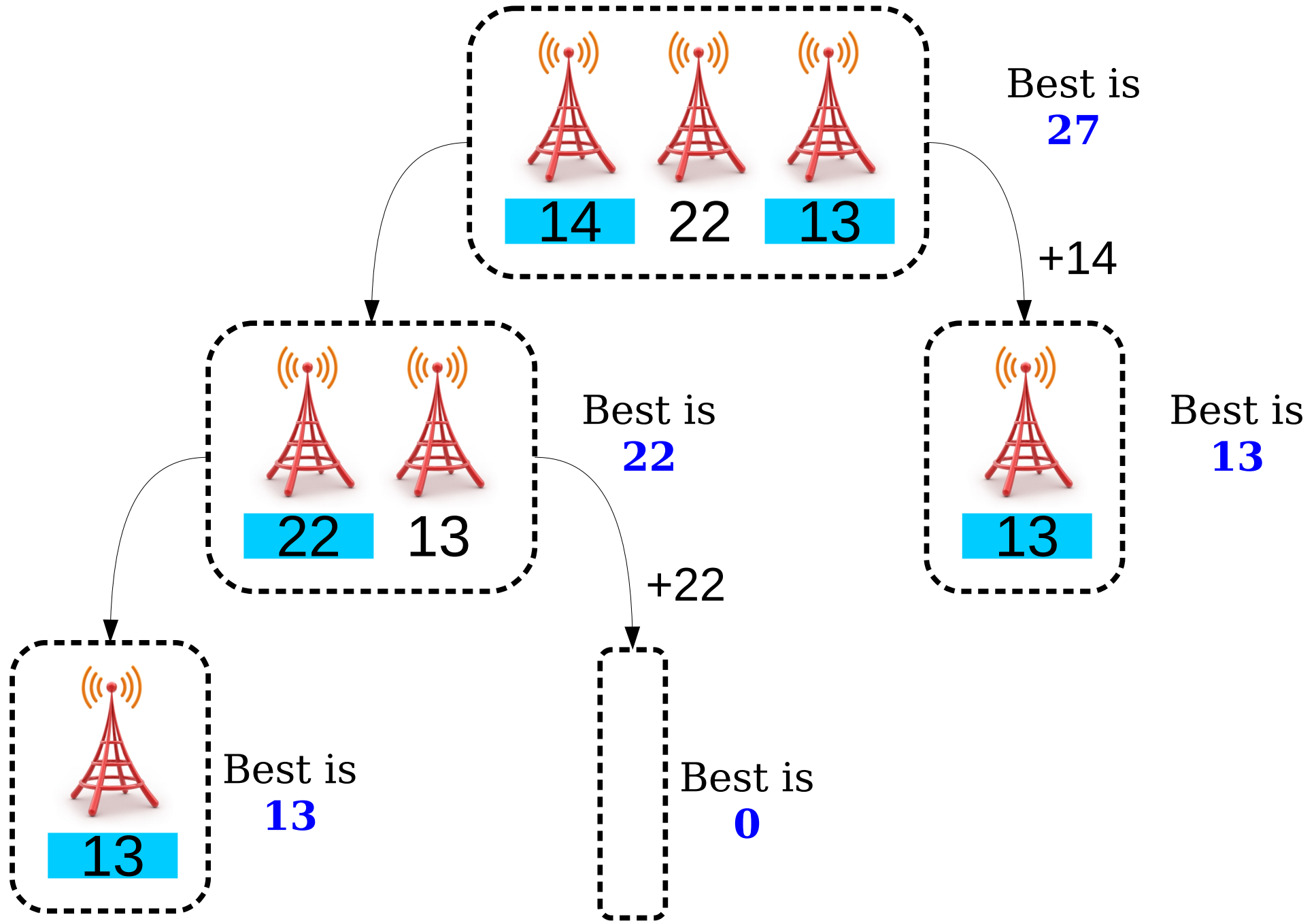
How the Recursion Works



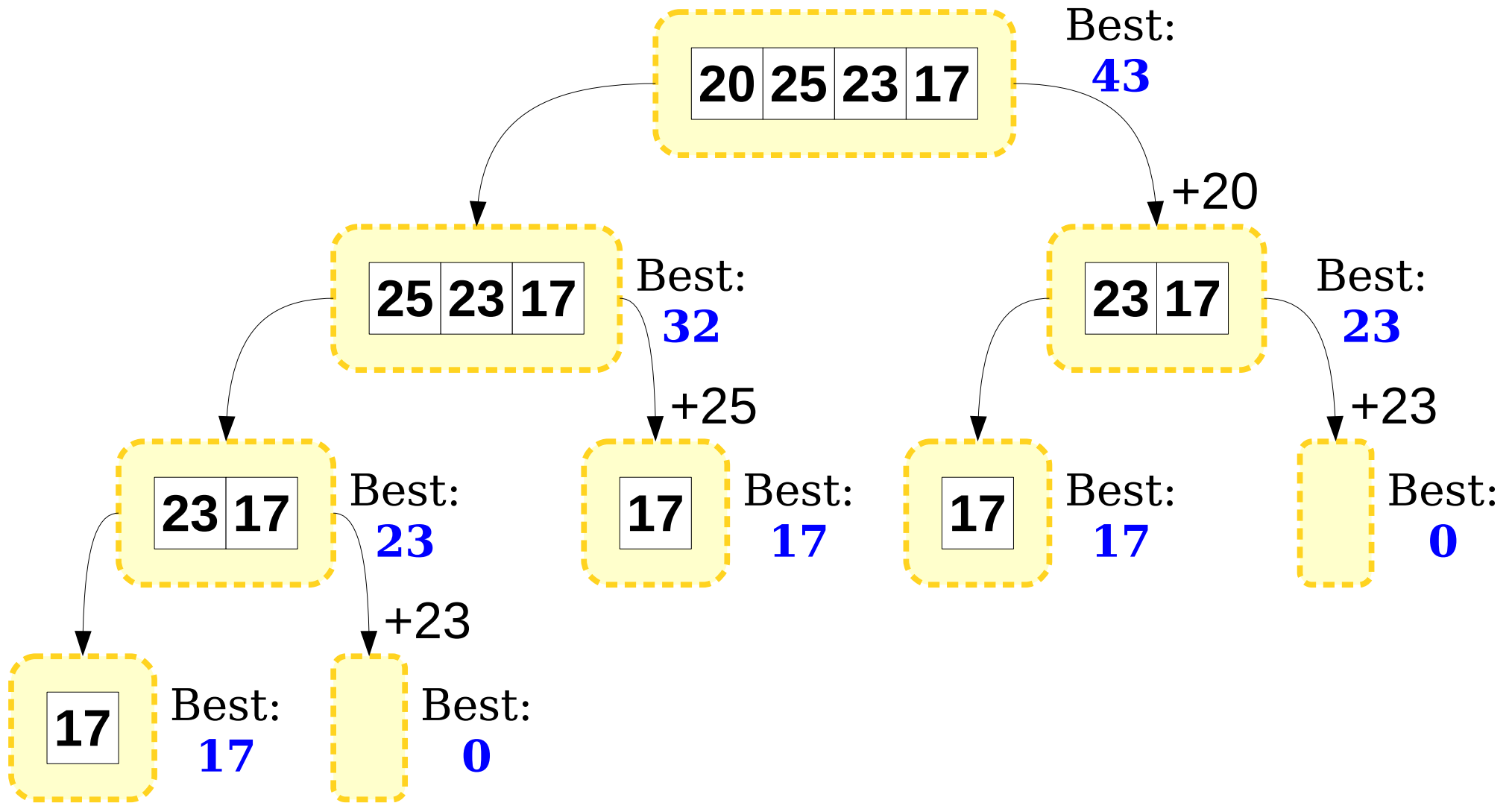
How the Recursion Works



How the Recursion Works



How the Recursion Works



Your Action Items

- Keep reading Chapter 5 on the different container types.
- Finish up Assignment 1! It's due on Monday and you'll probably not want to use your late days here.

Next Time

- *Map*
 - A collection for storing associations between elements.
- *Set*
 - A collection for storing an unordered group of elements.
- *Lexicon*
 - A special kind of Set.