Collections, Part Two

Outline for Today

- ***Stacks***
  - Pancakes meets parsing!
- ***Queues***
  - Waiting in line at the Library of Babel.

Stack

Stack

- A `Stack` is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.

Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it’s called the call *stack* and we talk about *stack* traces.

Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.

Stack

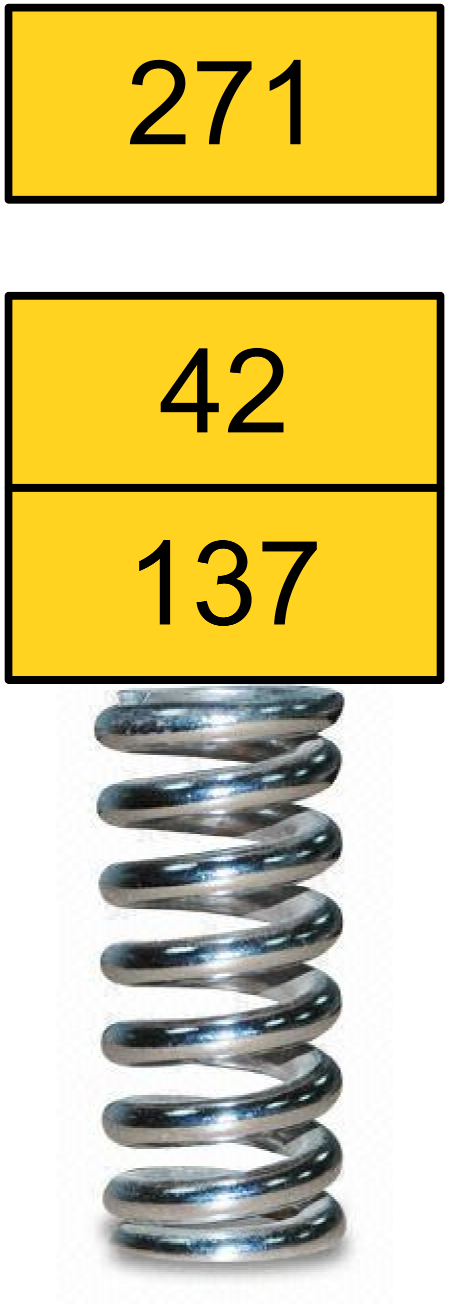
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.

Stack

- A `Stack` is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.

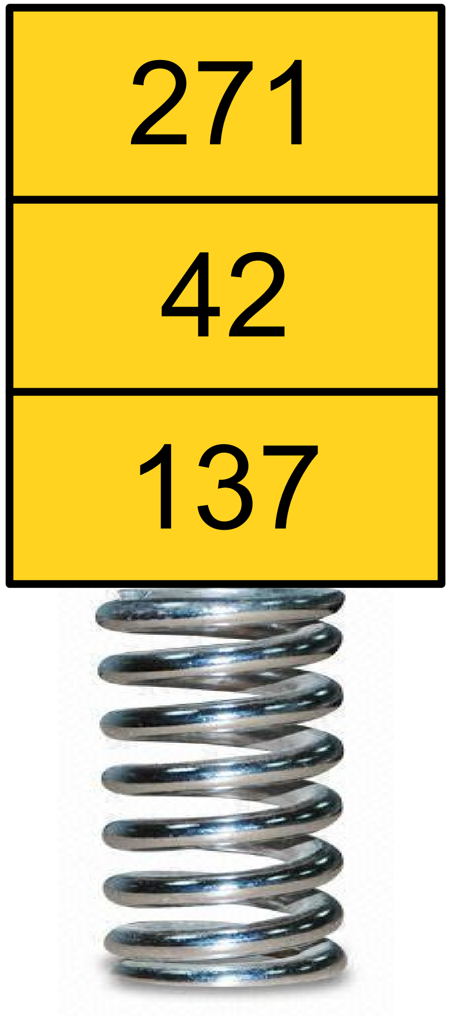
Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.



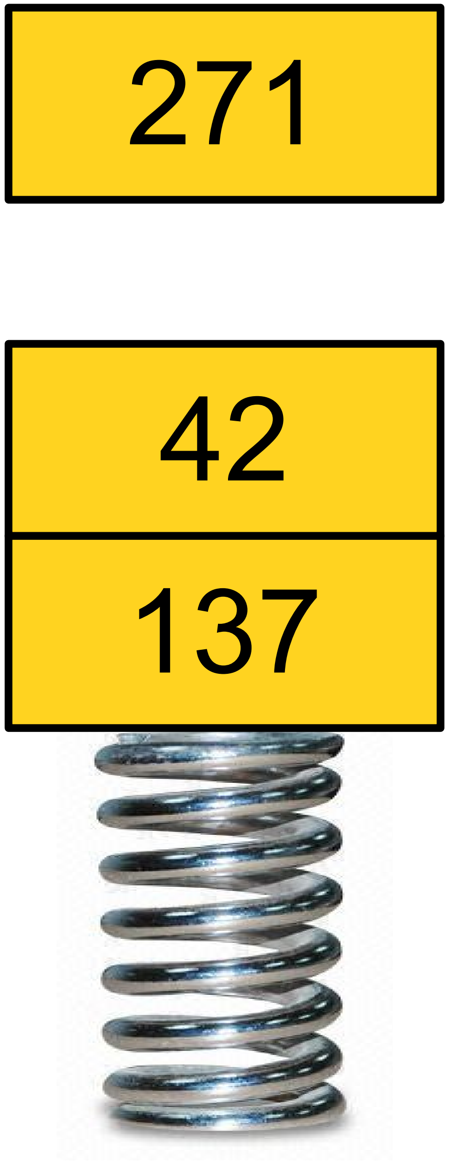
Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.



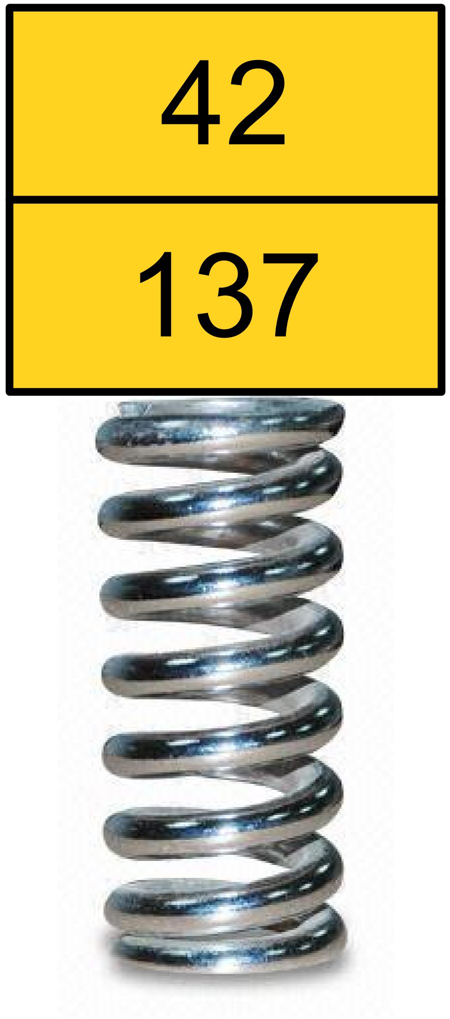
Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.



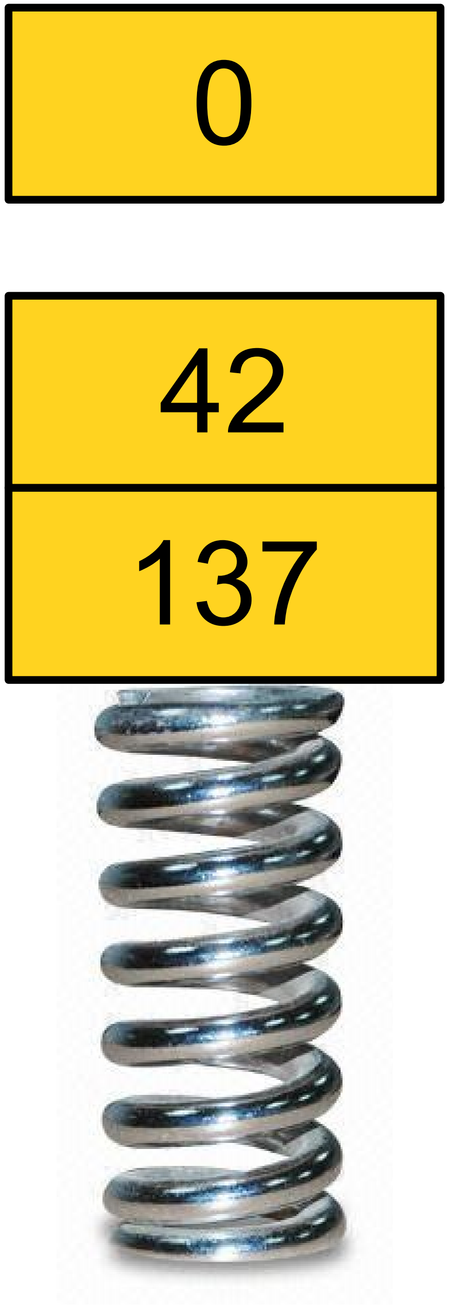
Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.



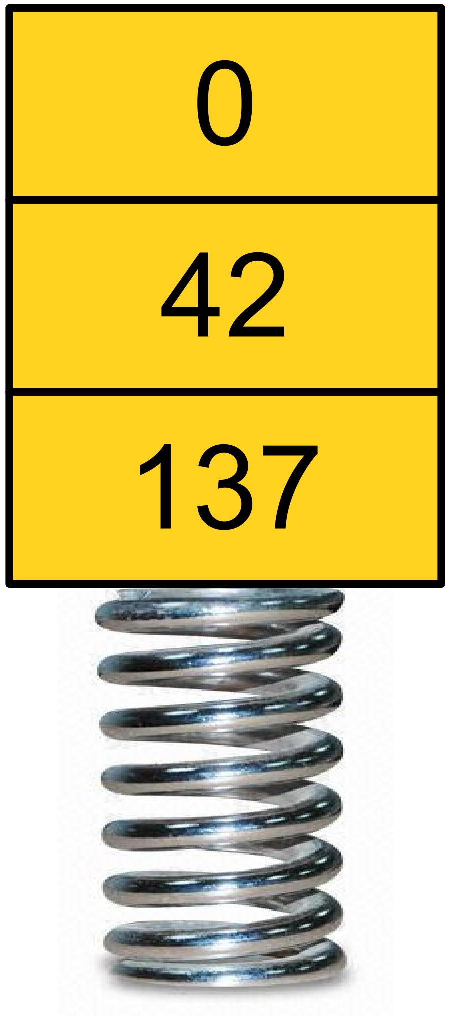
Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.



Stack

- A `Stack` is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- This is why it's called the call *stack* and we talk about *stack* traces.



An Application: ***Balanced Parentheses***

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
 ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
  ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
   ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
    ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
     ^
```

Balancing Parentheses

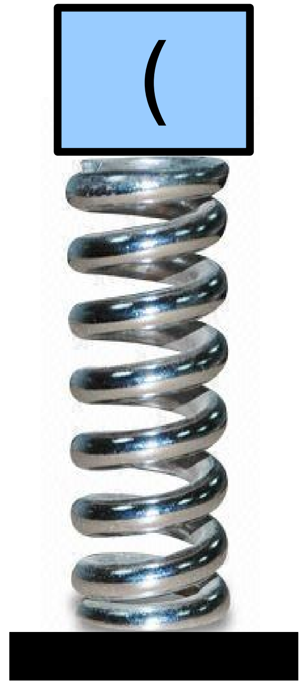
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
       ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
       ^
```

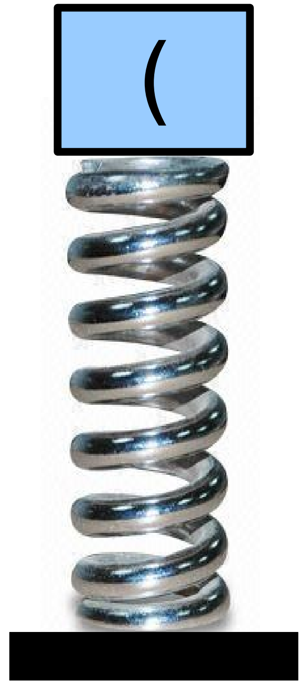
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
        ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
         ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
         ^
```

Balancing Parentheses

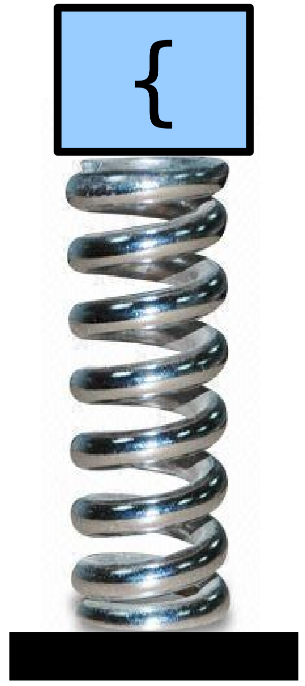
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

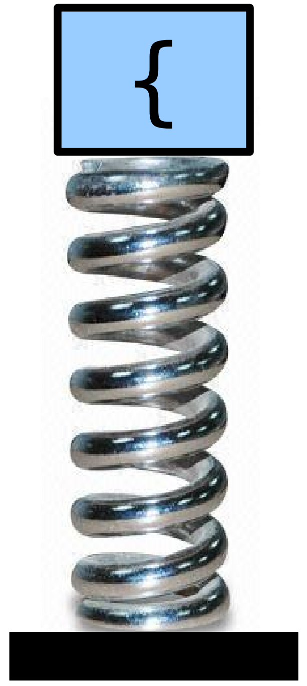
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```



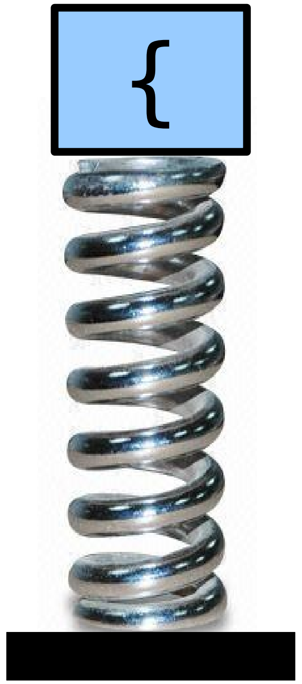
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
           ^
```



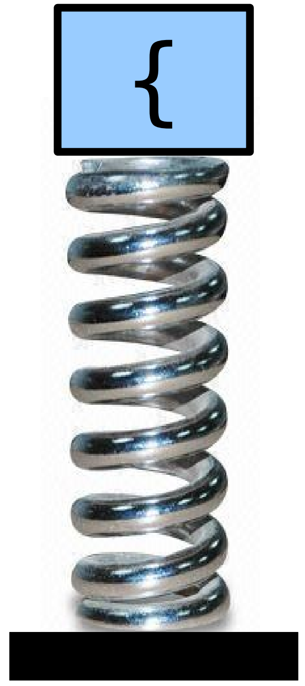
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
            ^
```



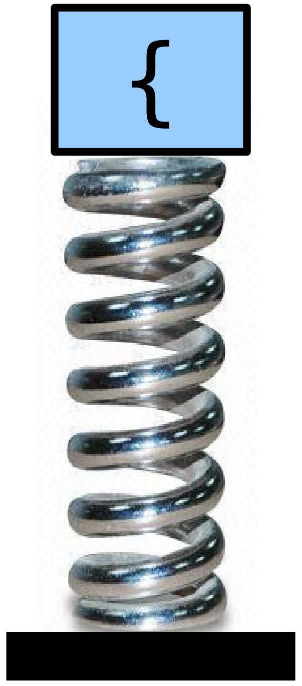
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
              ^
```



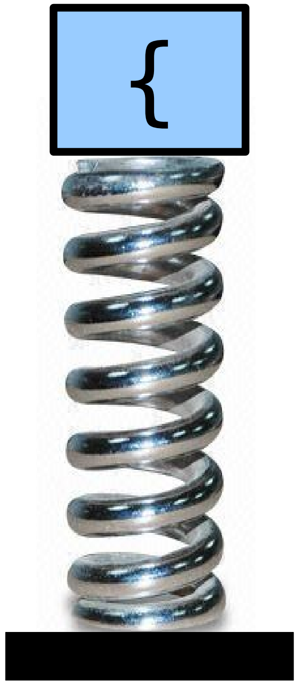
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
               ^
```



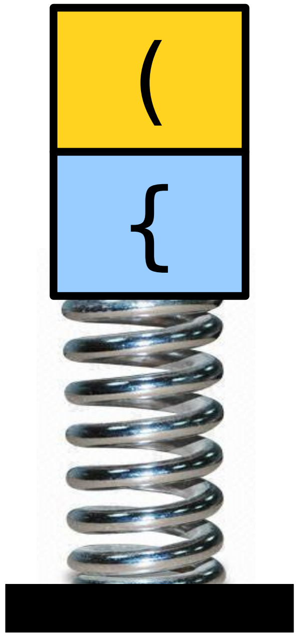
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
               ^
```



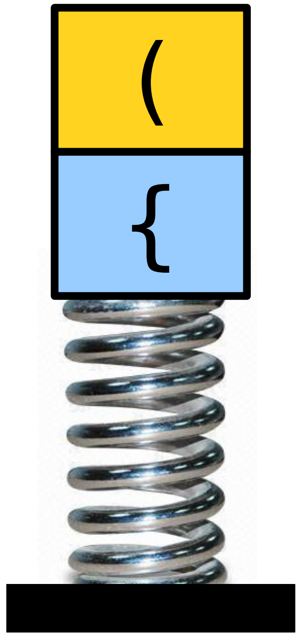
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
               ^
```



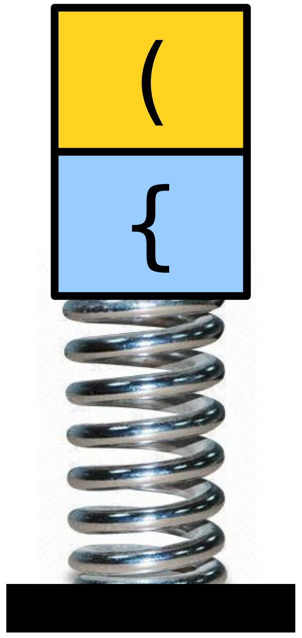
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                 ^
```



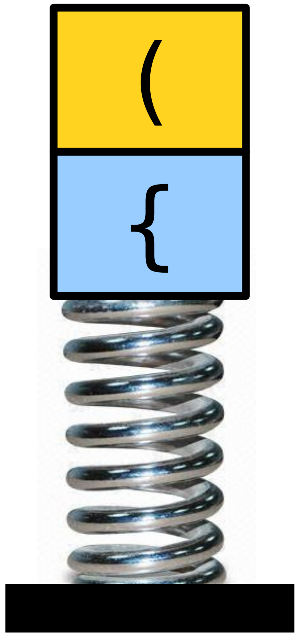
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                  ^
```



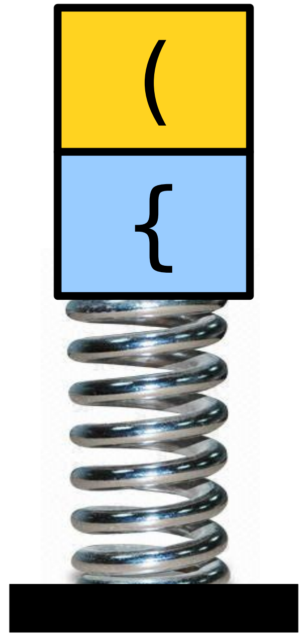
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                  ^
```



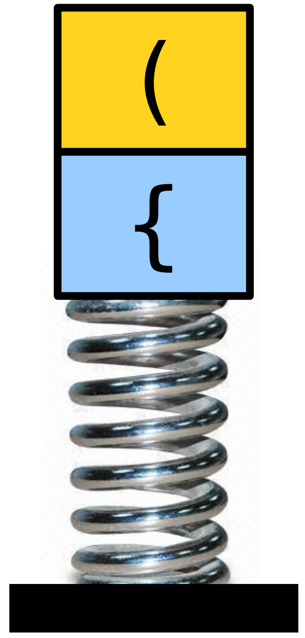
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                   ^
```



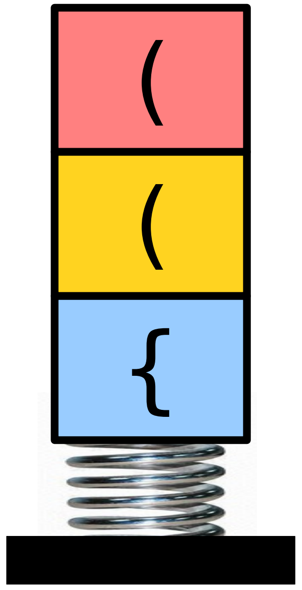
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                    ^
```



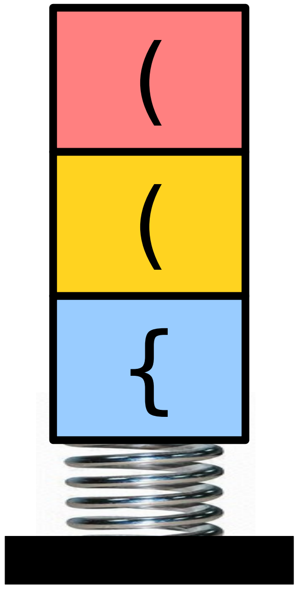
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                    ^
```



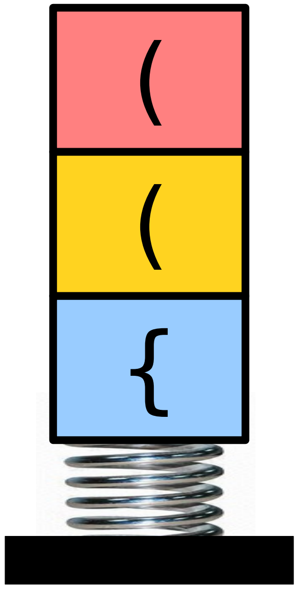
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                     ^
```



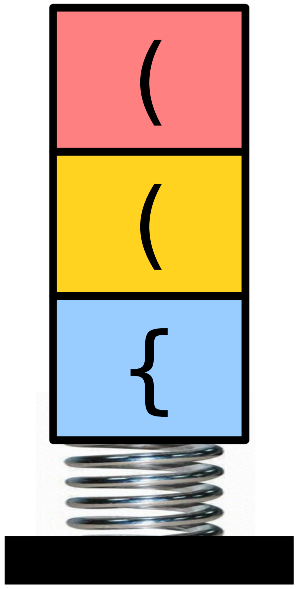
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                     ^
```



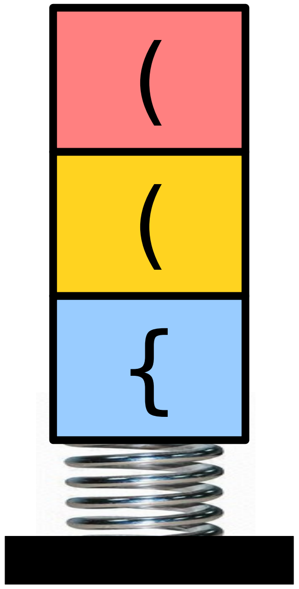
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                       ^
```



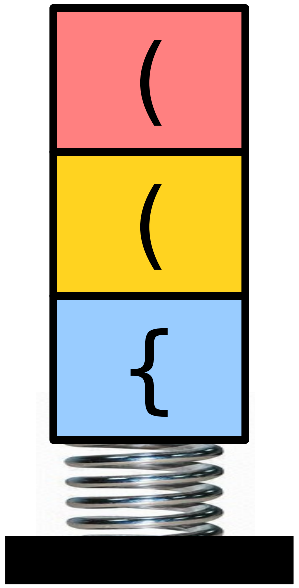
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



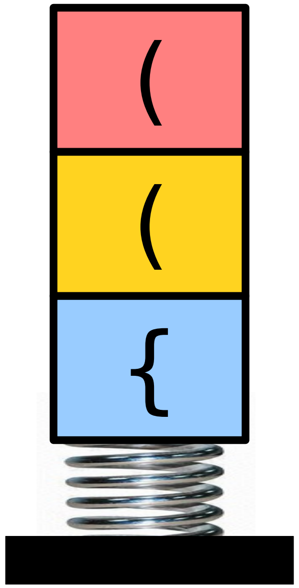
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                         ^
```



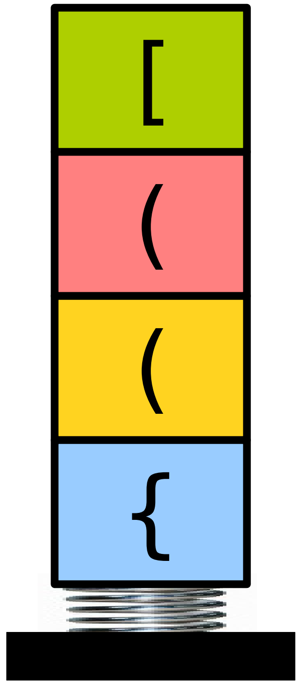
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                   ^
```



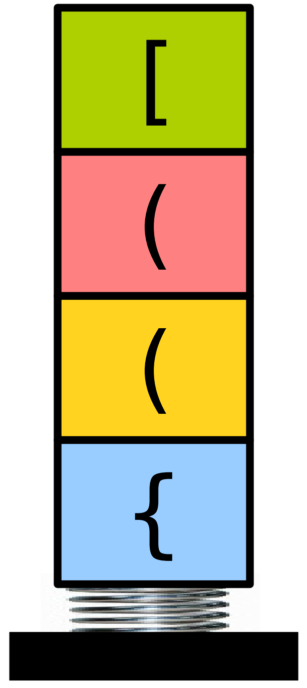
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



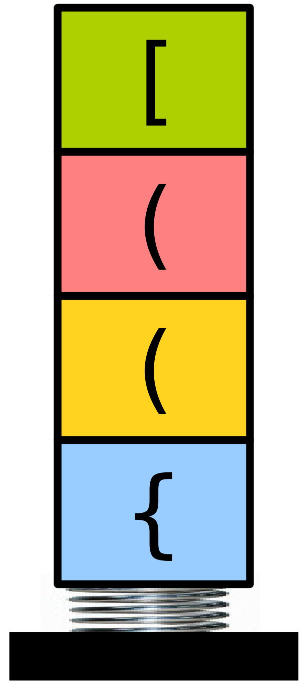
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                ^
```



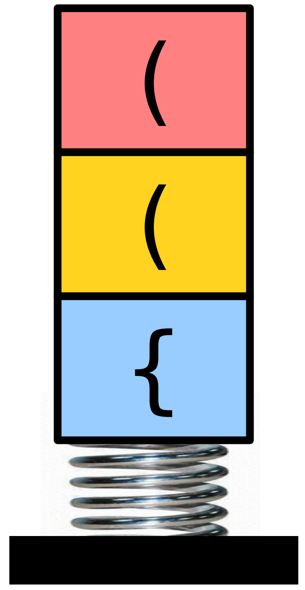
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                               ^
```



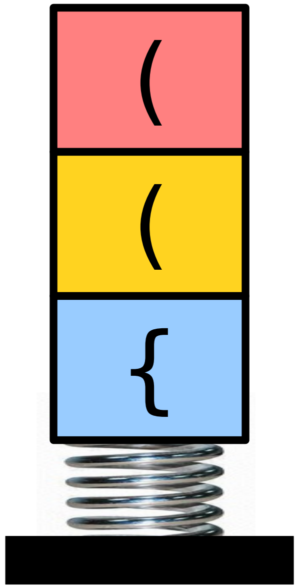
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                             ^
```



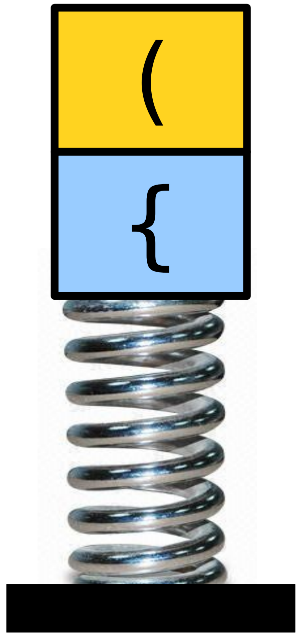
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                              ^
```



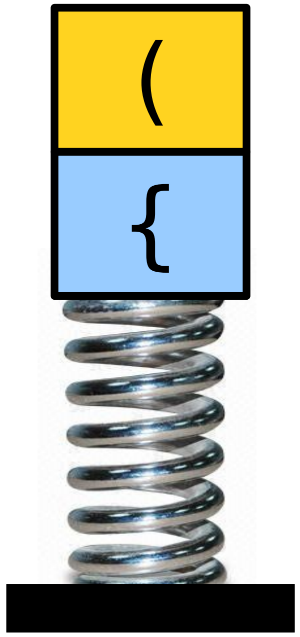
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                              ^
```



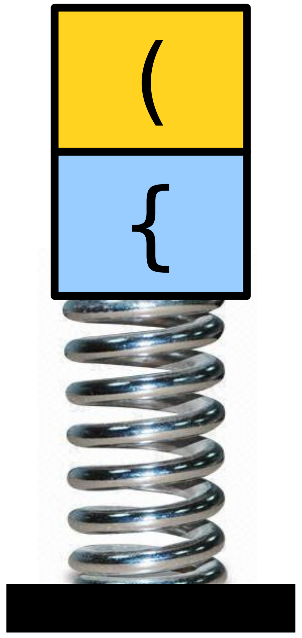
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                               ^
```



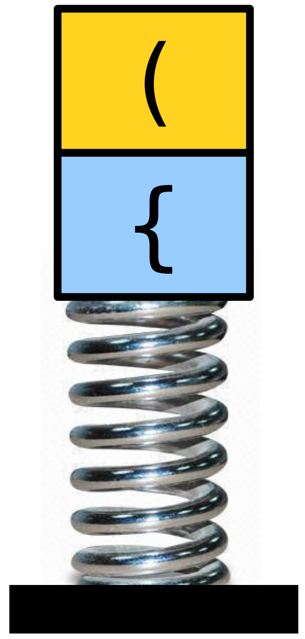
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                ^
```



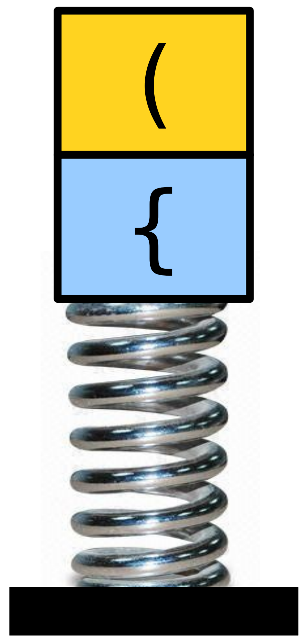
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                 ^
```



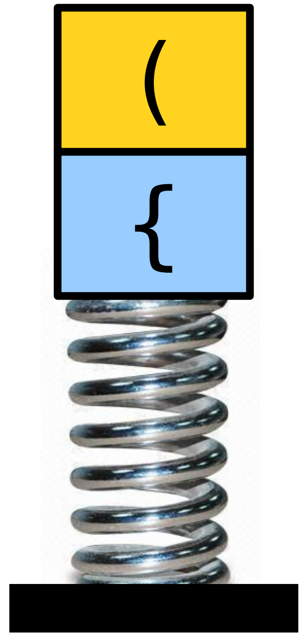
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                 ^
```



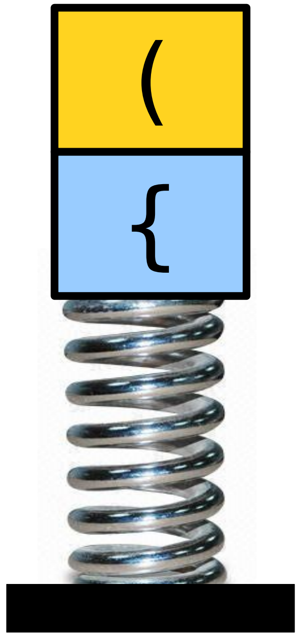
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                   ^
```



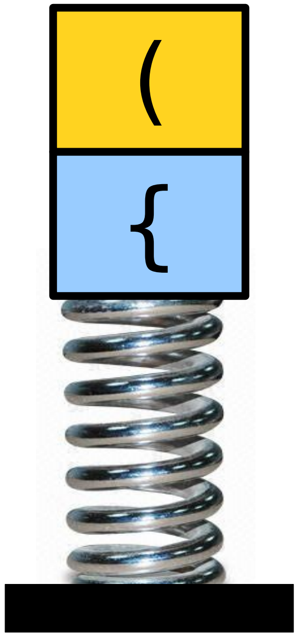
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                  ^
```



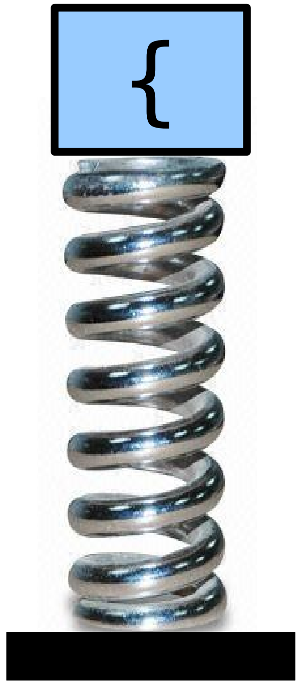
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                    ^
```



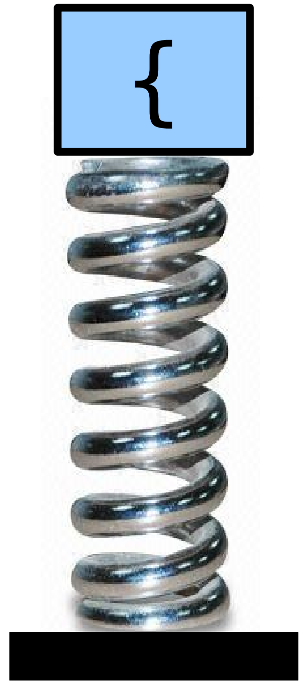
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                    ^
```



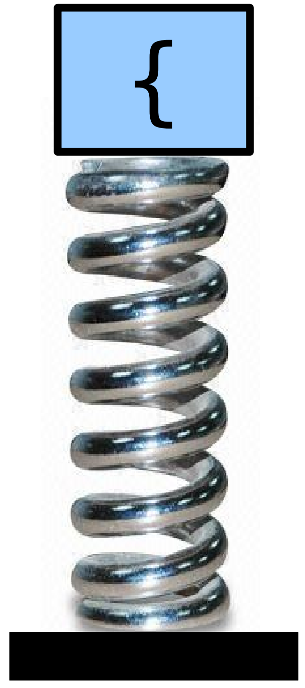
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```



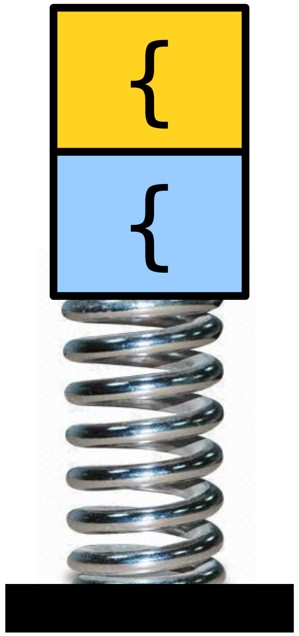
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```



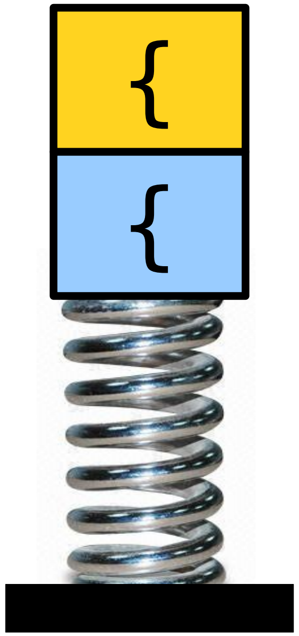
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```



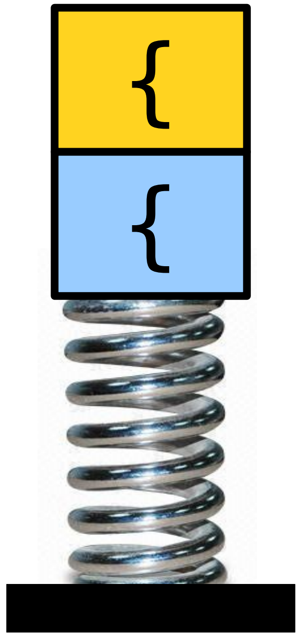
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                       ^
```



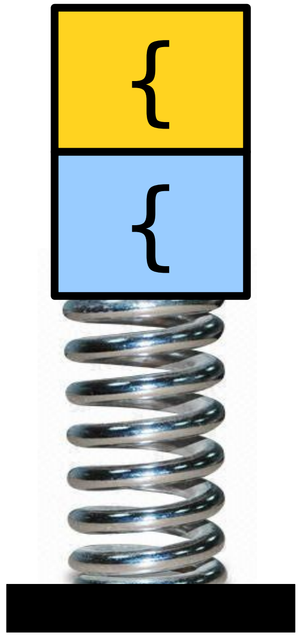
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                        ^
```



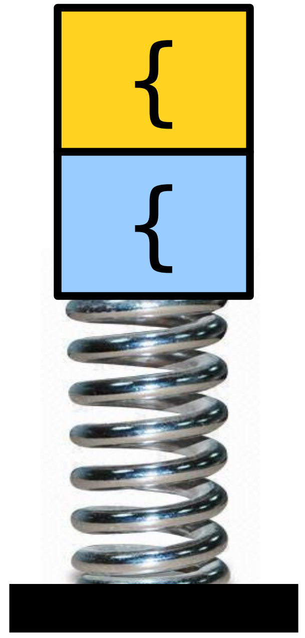
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                         ^
```



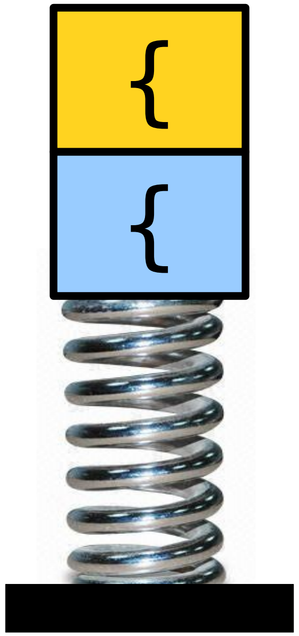
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                          ^
```



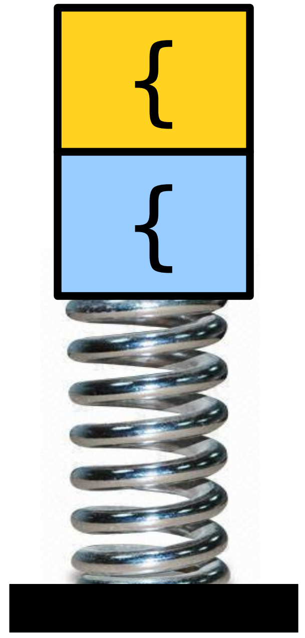
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                              ^
```



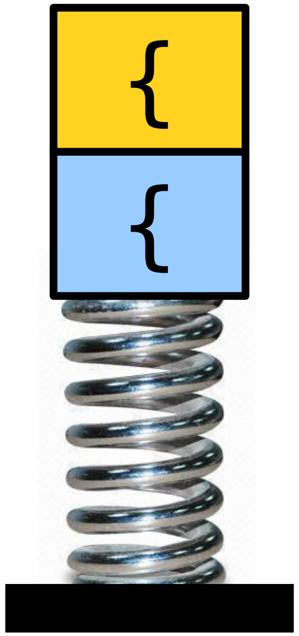
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                             ^
```



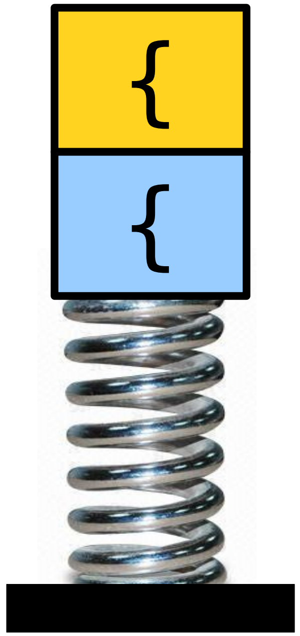
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                              ^
```



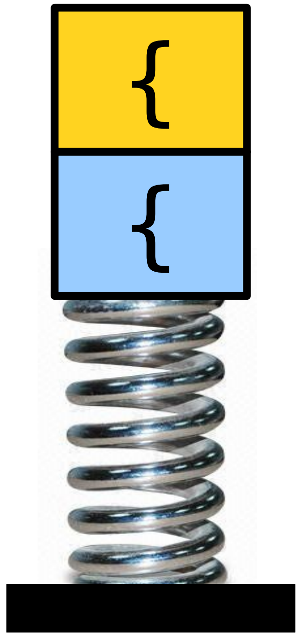
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                               ^
```



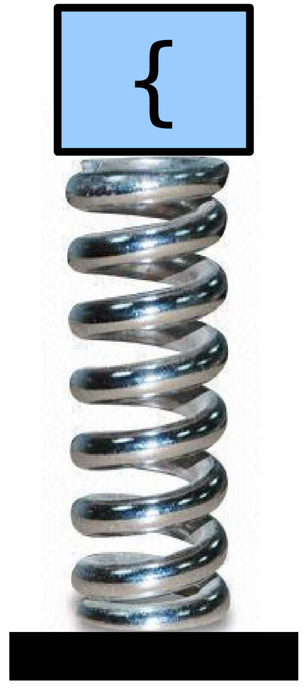
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                               ^
```



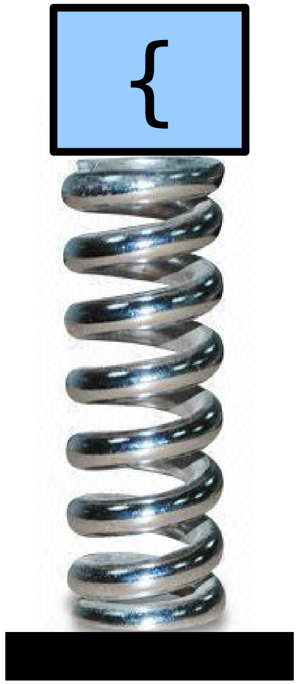
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                ^
```



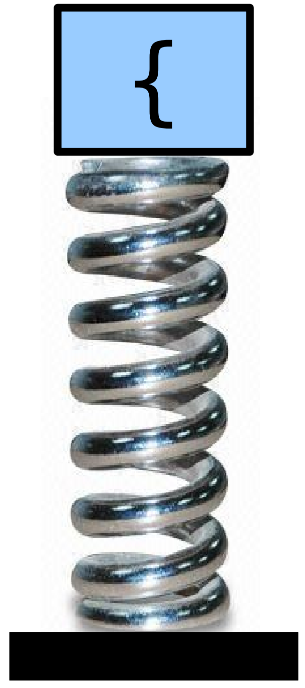
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                  ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                 ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                  ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

Let's go code this up!

Our Algorithm

- For each character:
  - If it's an open parenthesis or brace, push it onto the stack.
  - If it's a close parenthesis or brace:
    - If the stack is empty, report an error.
    - If the character doesn't pair with the character on top of the stack, report an error.
- At the end, return whether the stack is empty (nothing was left unmatched.)

Great Exercise: Reimplement this function purely using the *call stack* and *recursion* rather than a `Stack<char>`.

Other Stack Applications

- Stacks show up all the time in ***parsing***, recovering the structure in a piece of text.
  - Often used in natural language processing; take CS224N for details!
  - Used all the time in compilers – take CS143 for details!
  - There's a deep theorem that says that many structures appearing in natural language are perfectly modeled by operations on stacks; come talk to me after class if you're curious!
- They're also used as building blocks in larger algorithms for doing things like
  - making sure a city's road networks are navigable (finding *strongly connected components*; take CS161 for details!) and
  - searching for the best solution to a problem (stay tuned!)

Time-Out for Announcements!

Assignment 1

- Assignment 1 is due this Friday at the start of class.
- Have questions?
    - Stop by the LaIR!
    - Ask on Piazza!
    - Email your section leader!

Sections This Week

- Sections start this week! You should have received your section assignment over email on Tuesday.
  - Didn’t get it? Check cs198.stanford.edu for your assignment.
- Section solutions are available on the course website. We recommend looking at them only after working through the relevant problems.

YEAH Hours

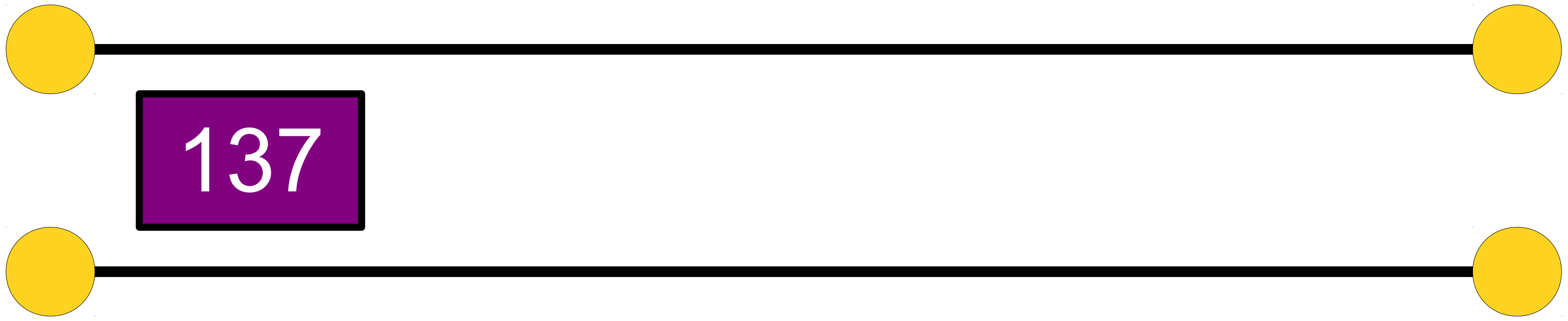
- We'll be holding YEAH Hours (Your Early Assignment Help Hours), an assignment review session, for Assignment 2 when it goes out on Friday.
- YEAH Hours will be held this Friday at 3:30PM in Shriram 104.
- Can't make it? No worries! We'll post slides on the website.

Many Happy Returns!

Queue

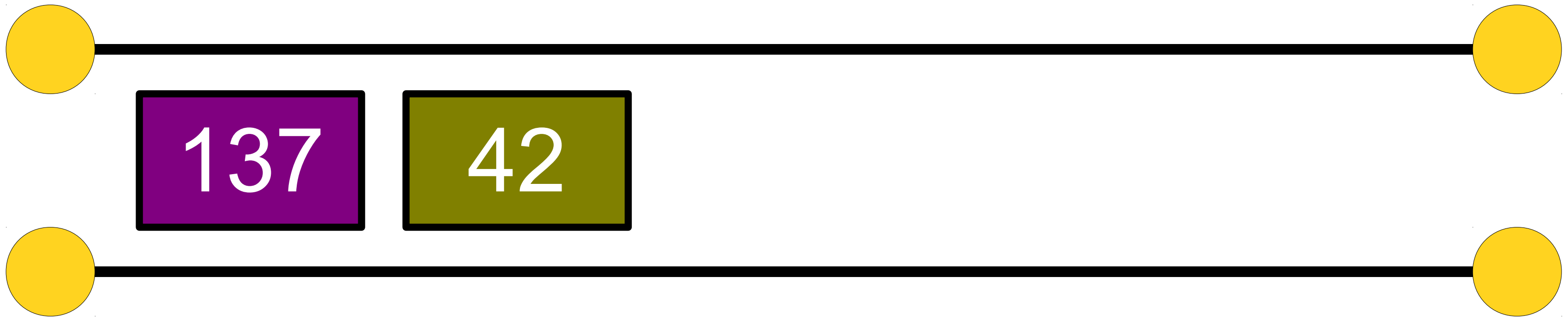
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



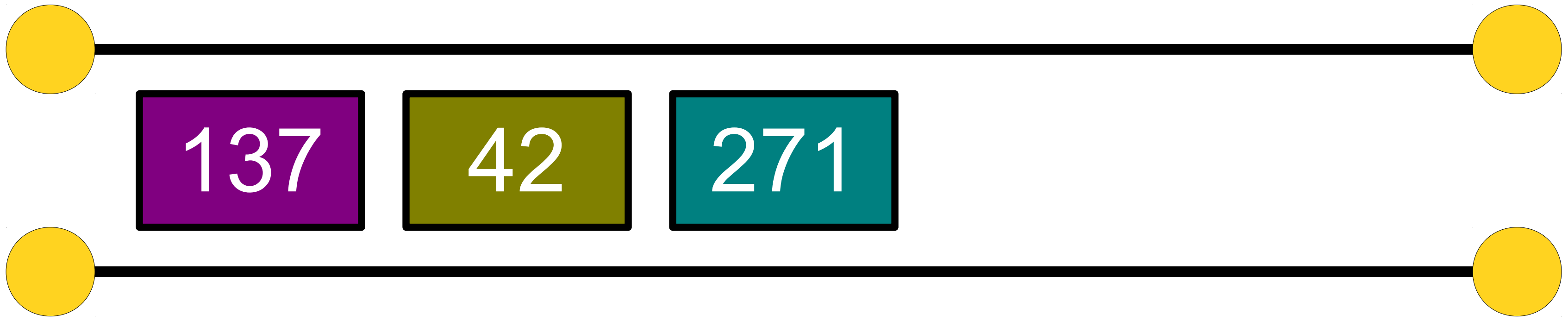
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



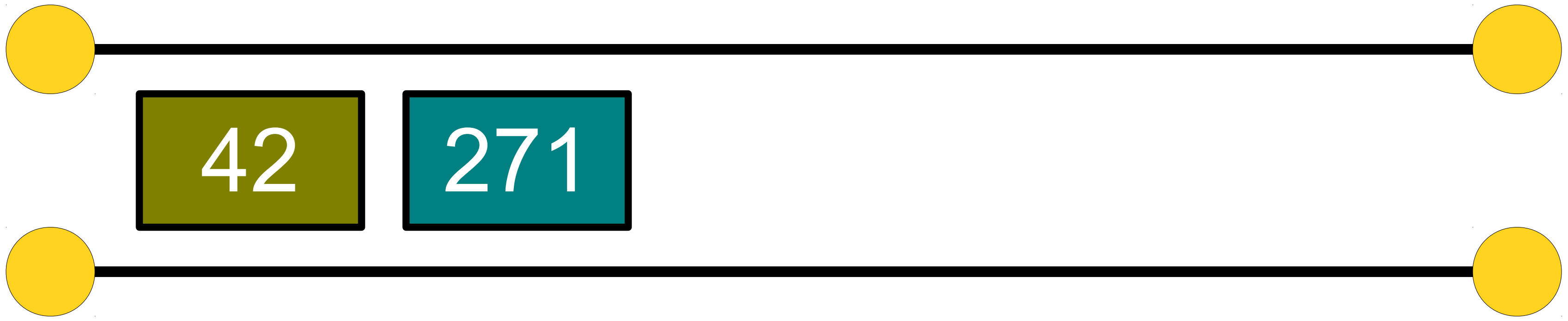
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



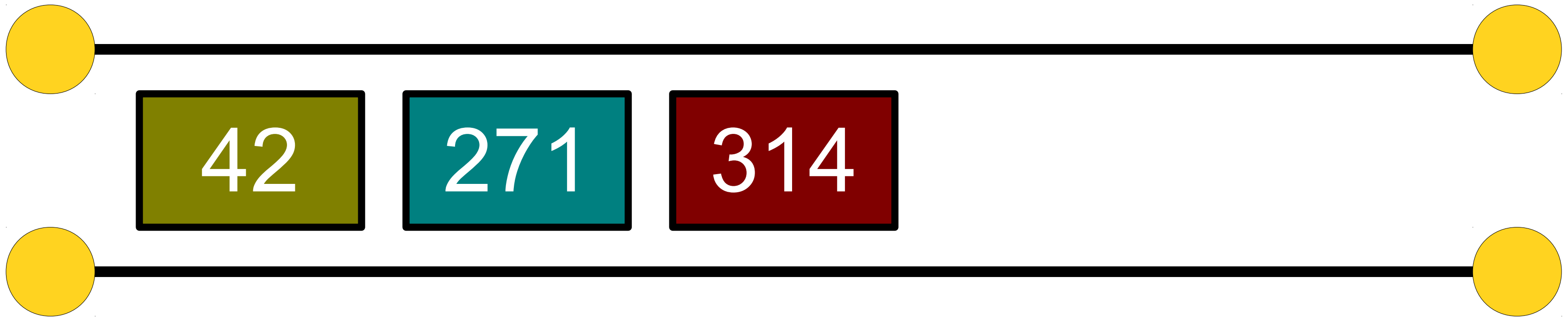
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



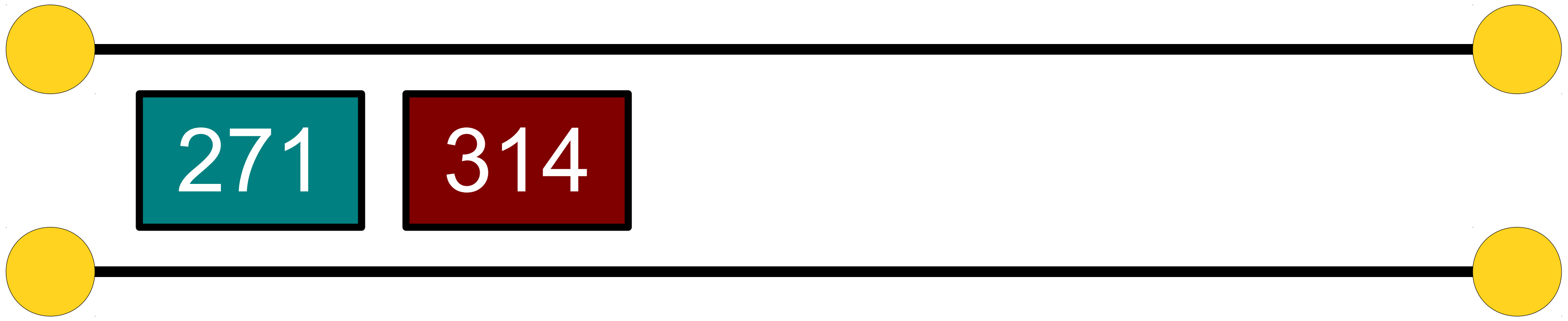
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



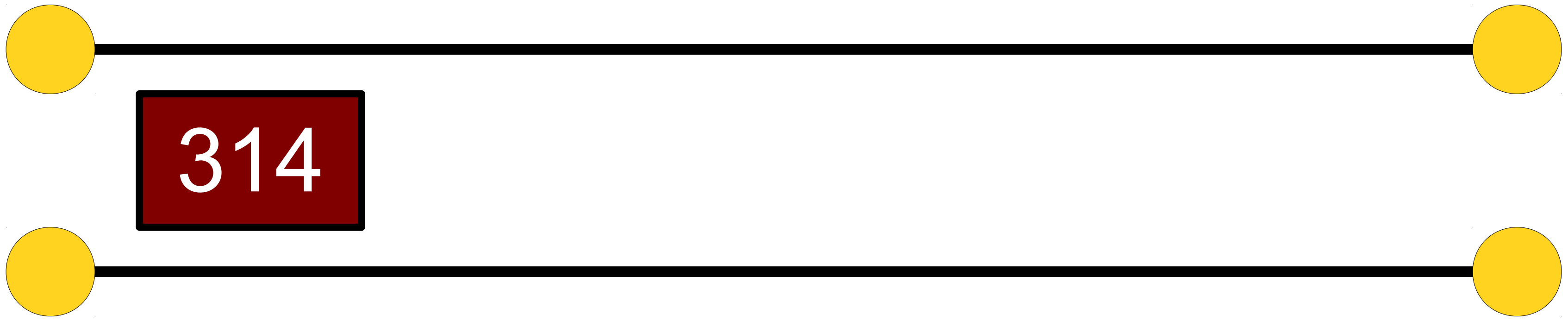
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



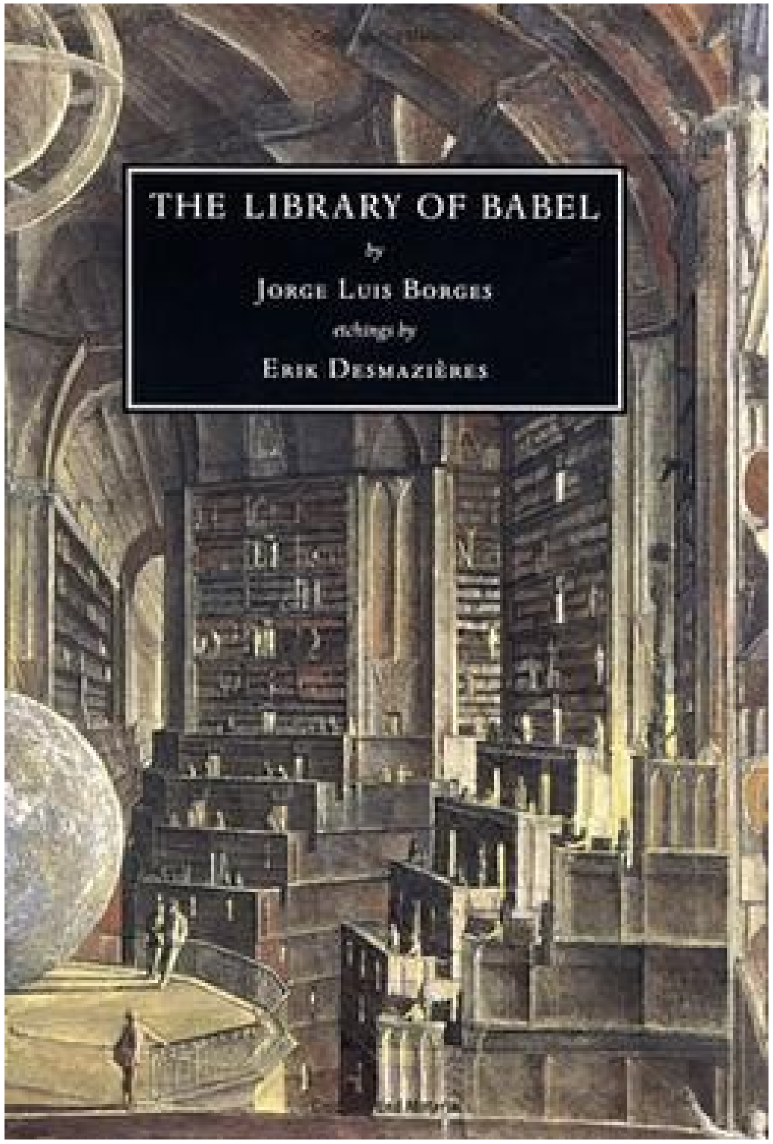
Queue

- A `Queue` is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



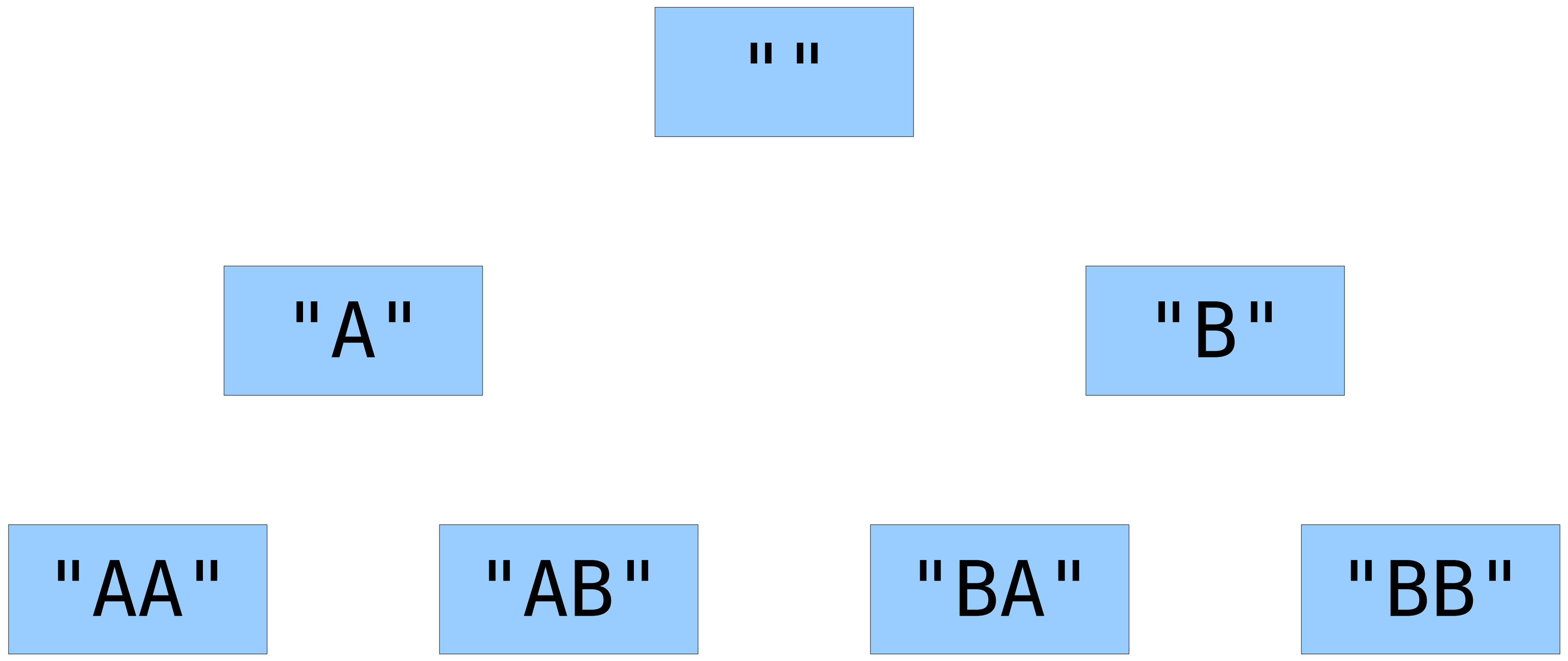
An Application: ***The Library of Babel***

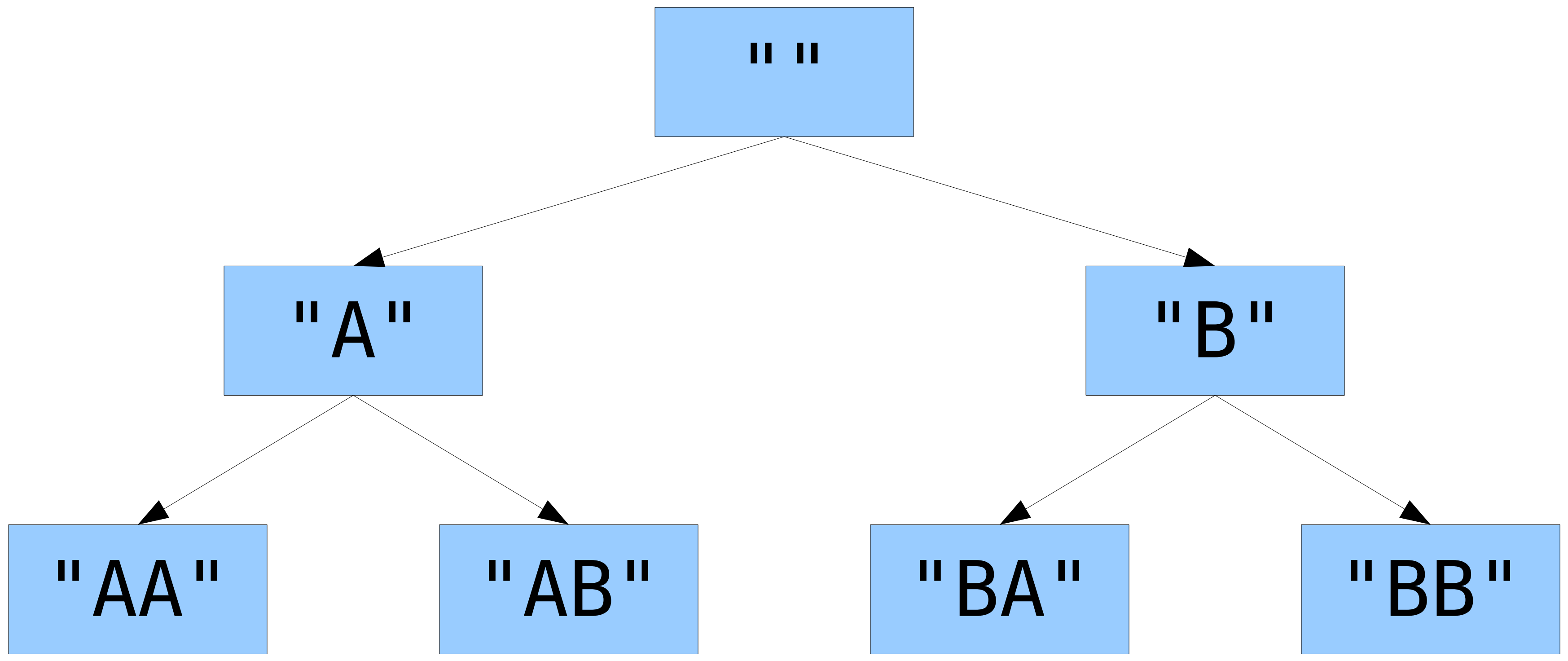
The Library of Babel

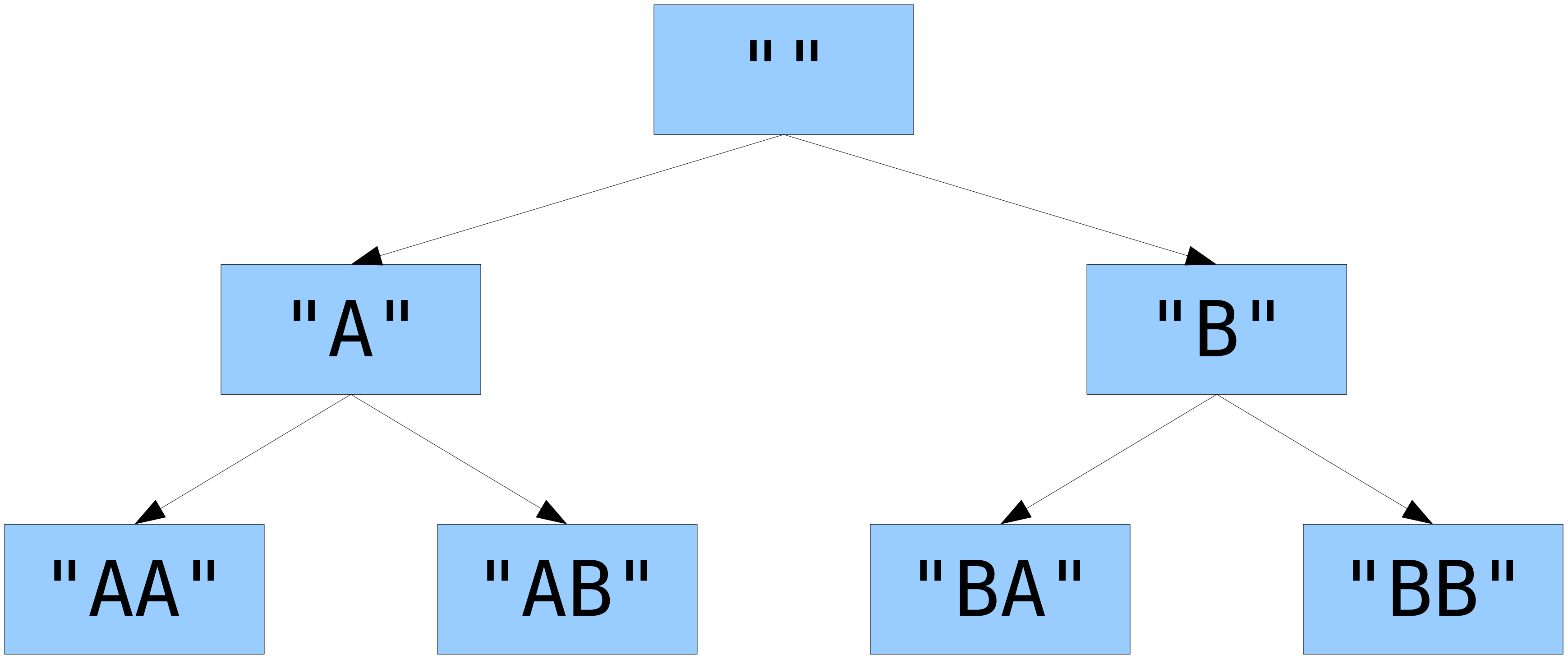


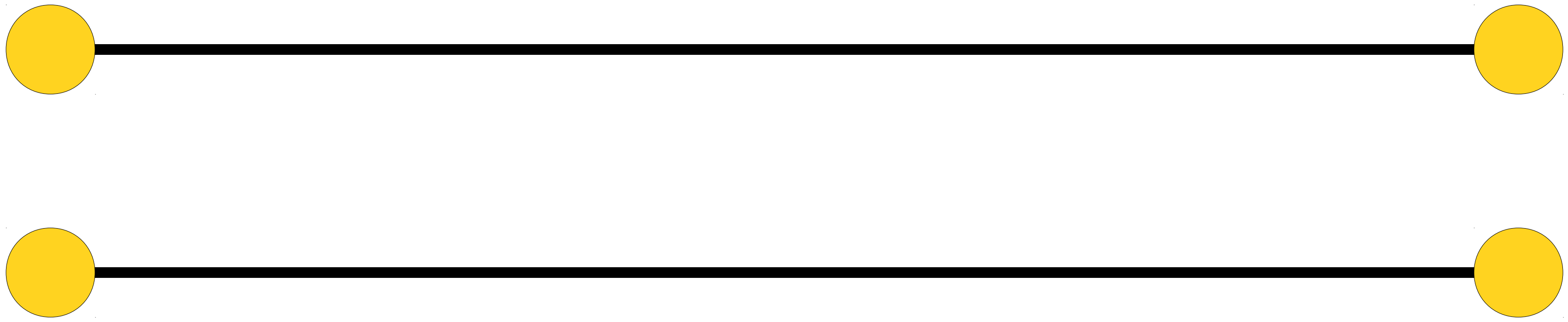
- Short story by Jorge Luis Borges about a library containing all possible books of a certain length.
- Fun exposition about the nature of the finite and the infinite, about what truth means, and how people react to these concepts.
- ***Question:*** How would you generate all these books?

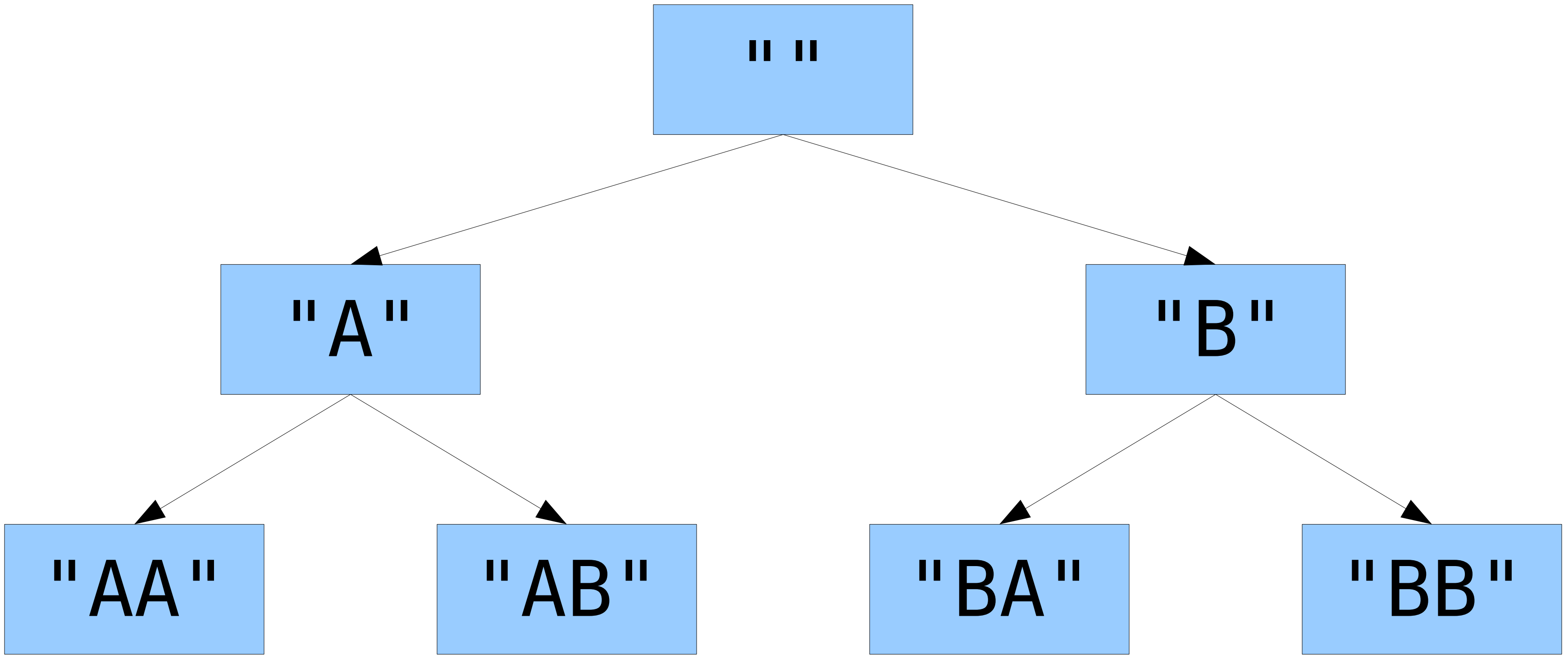
" "	"A"	"B"	"AA"	"AB"	"BA"	"BB"

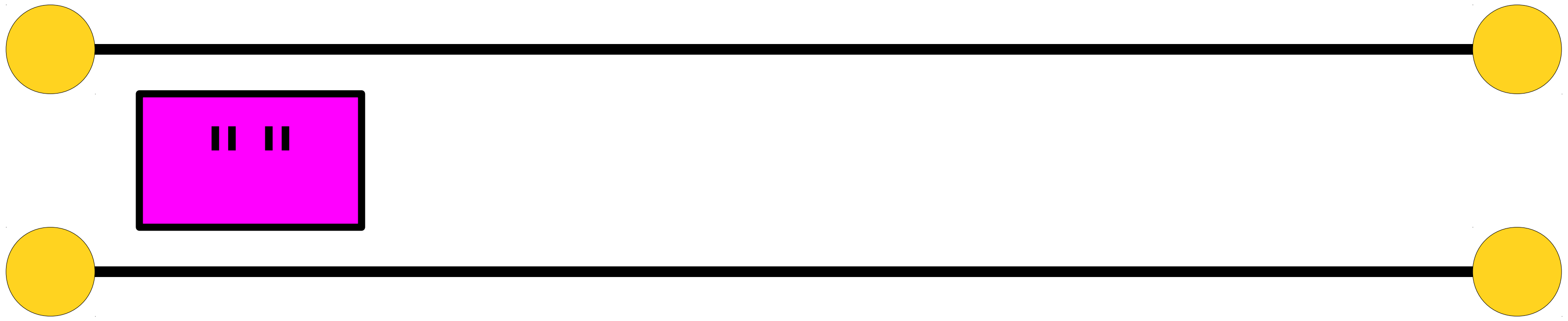

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

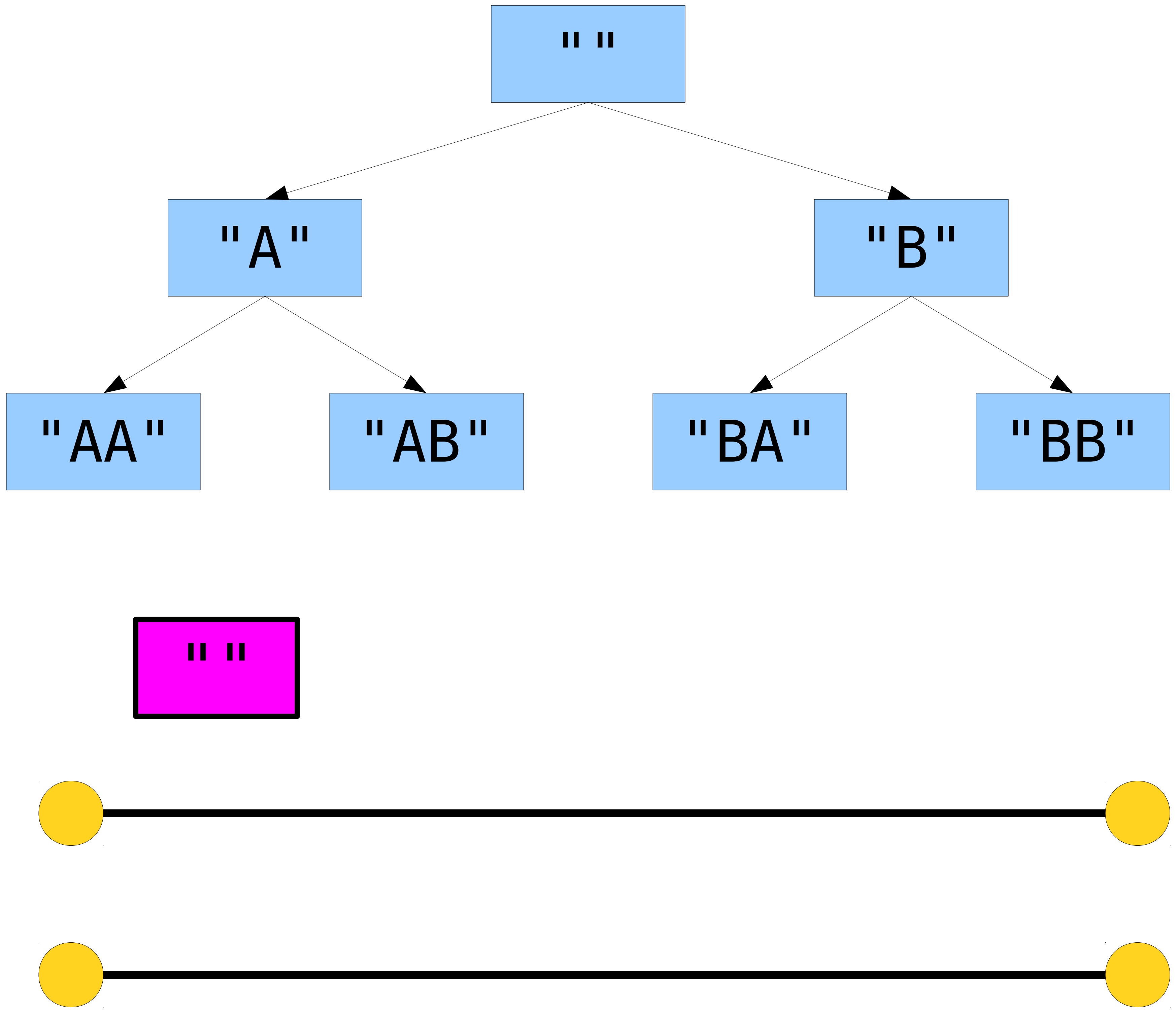

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

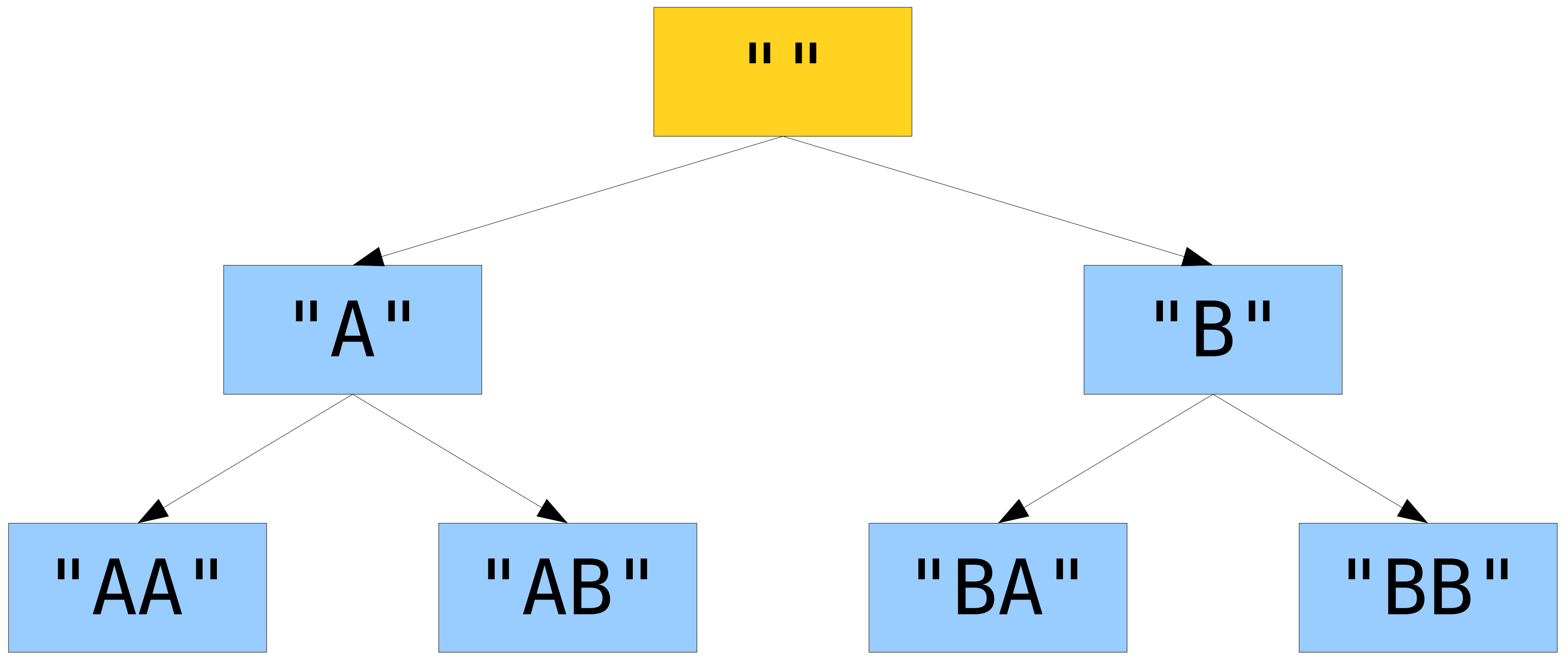

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

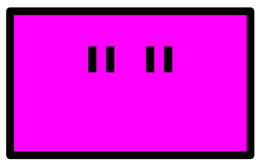


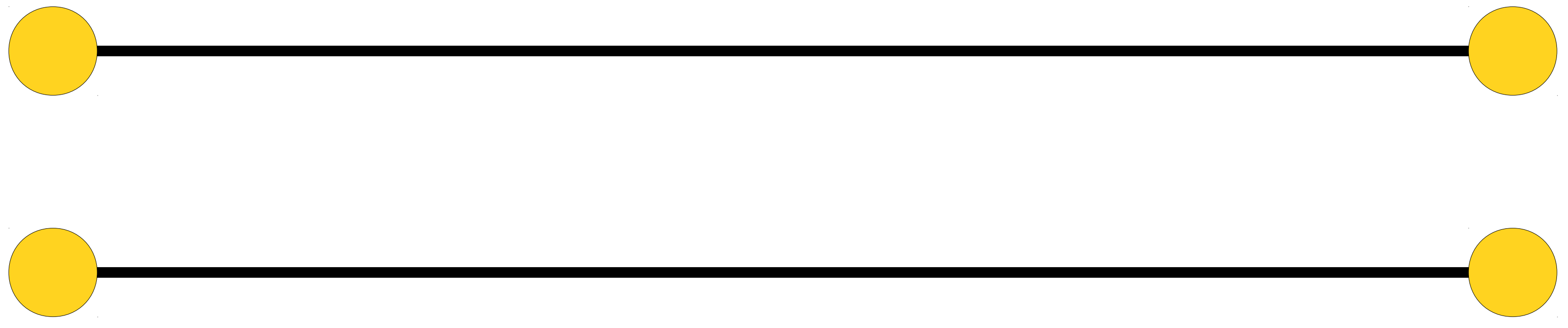

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

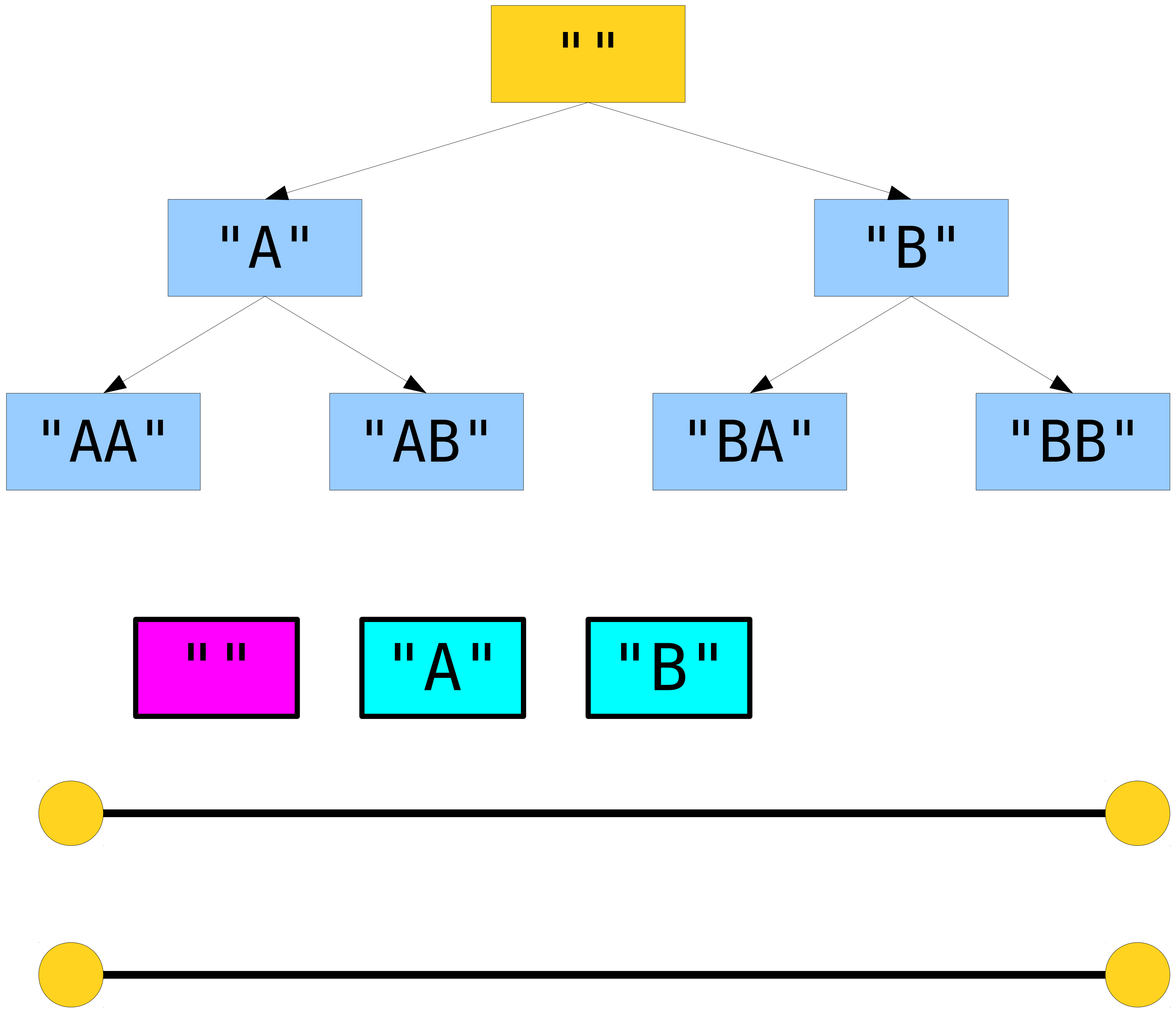

" "

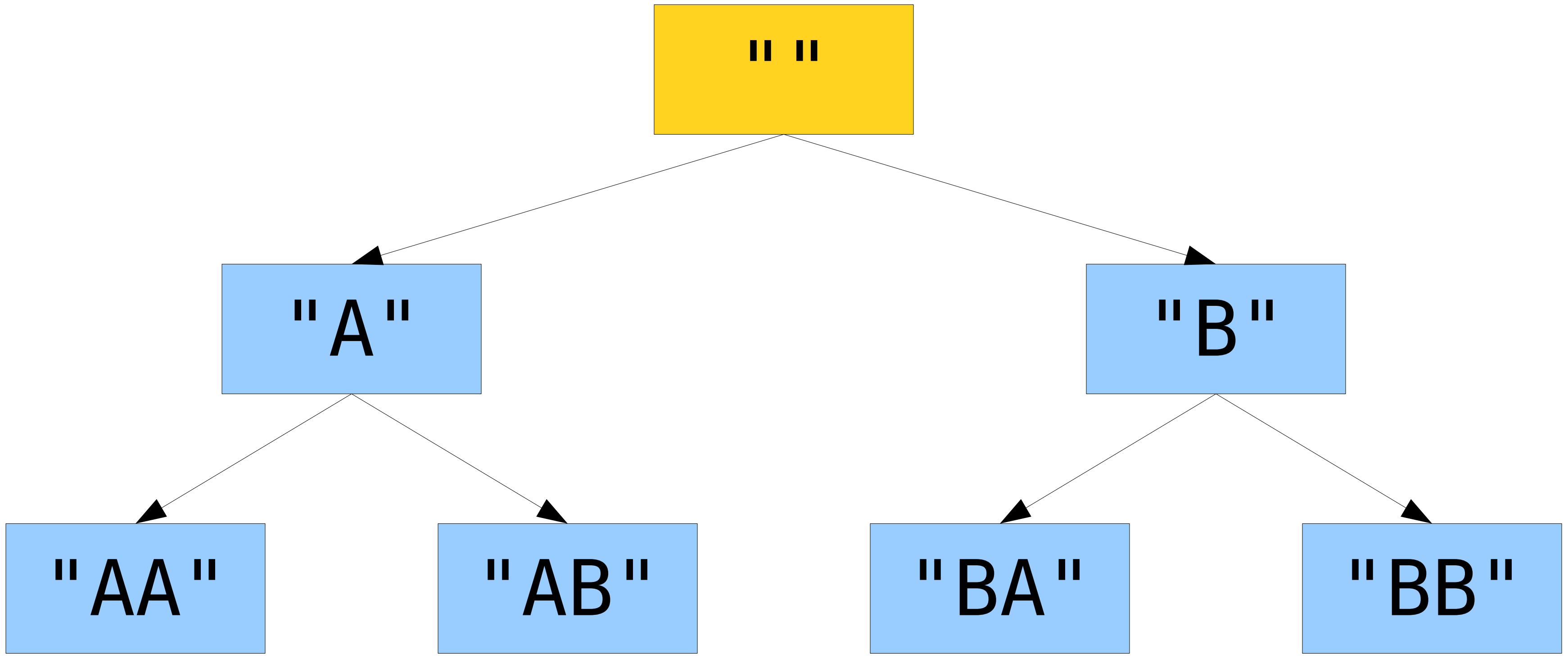

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"
" "

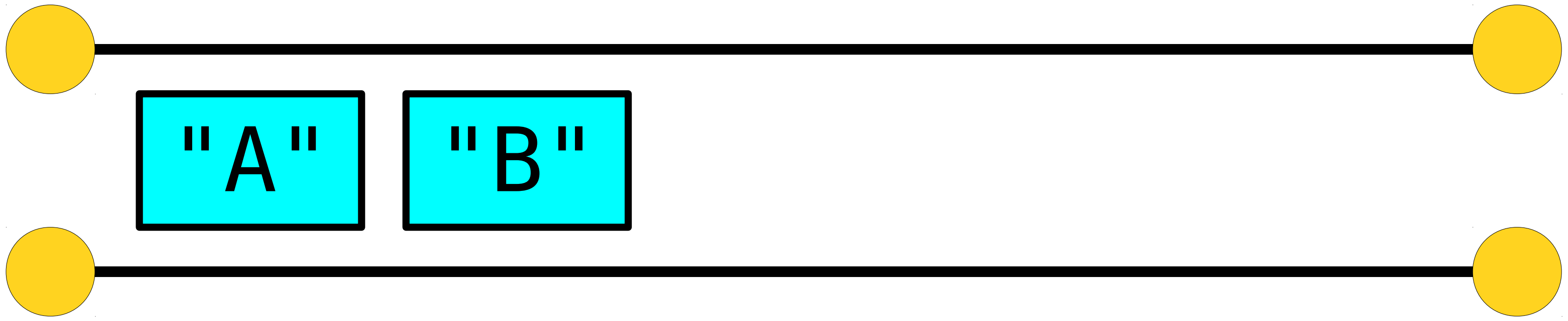

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

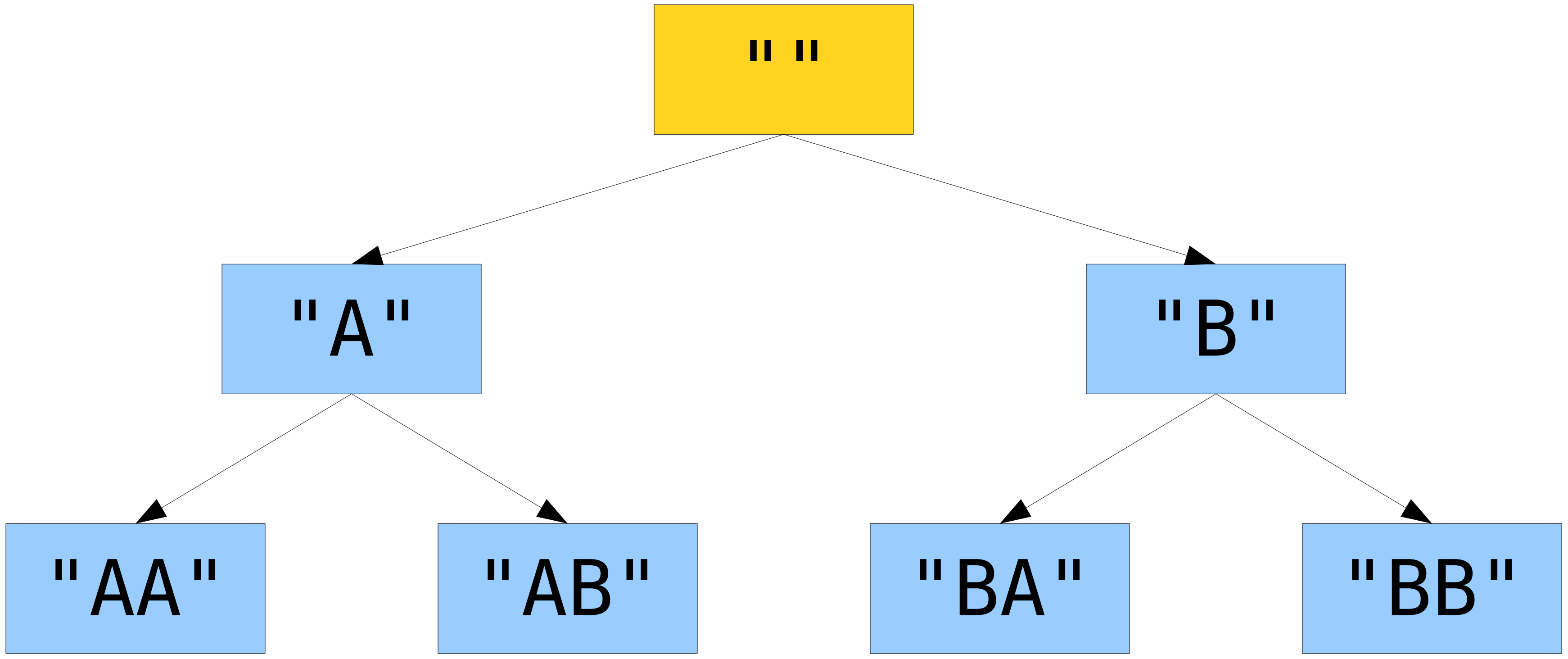

" "

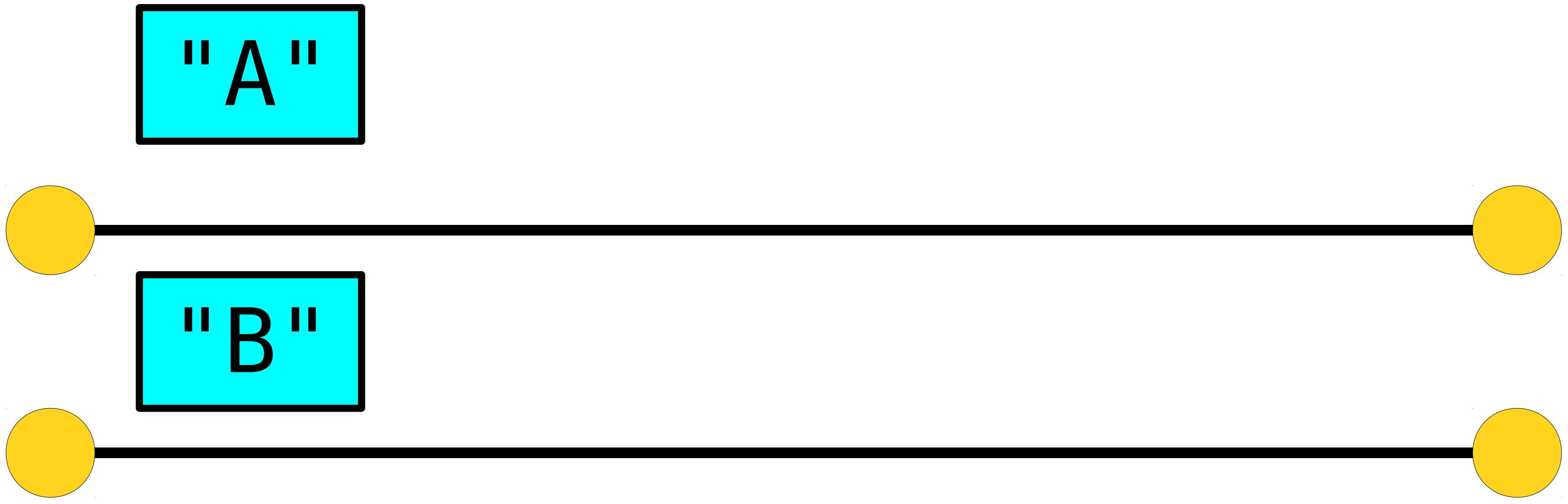


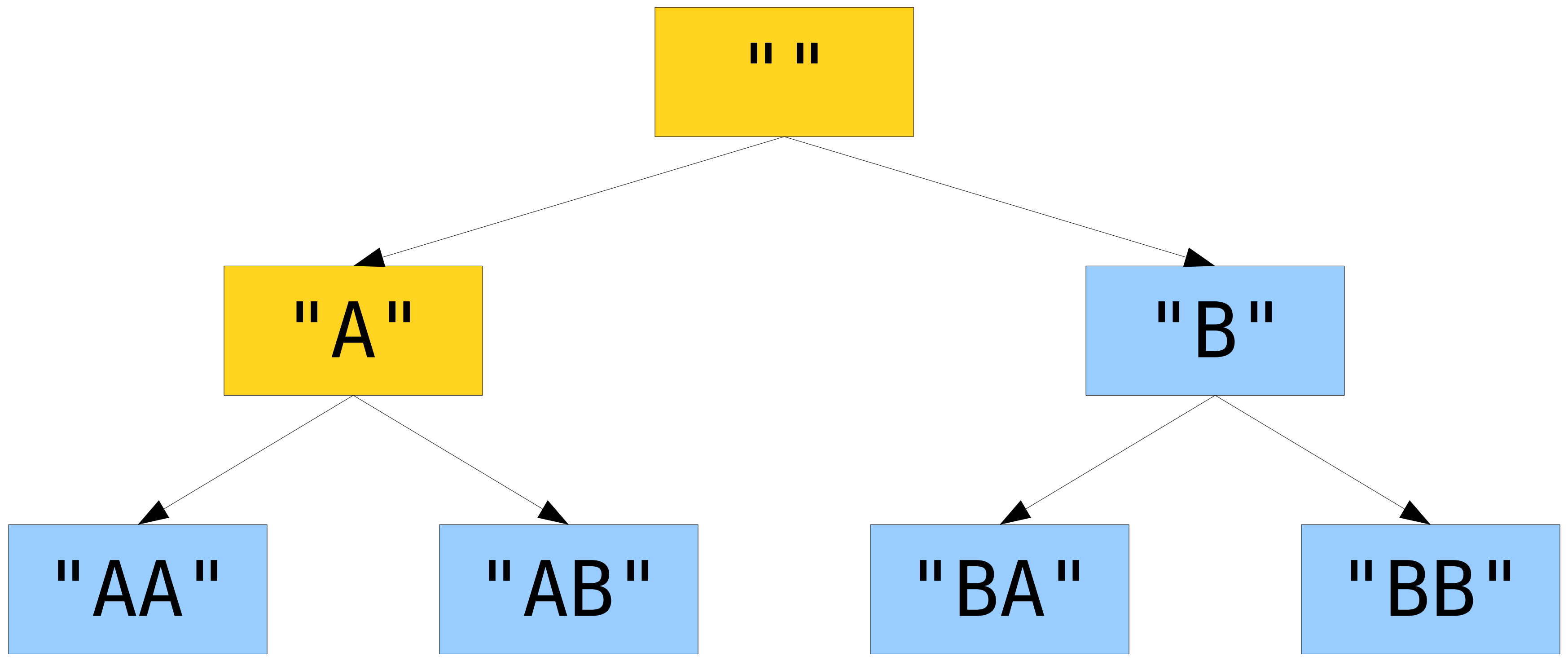

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"
" "
"A"
"B"


" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

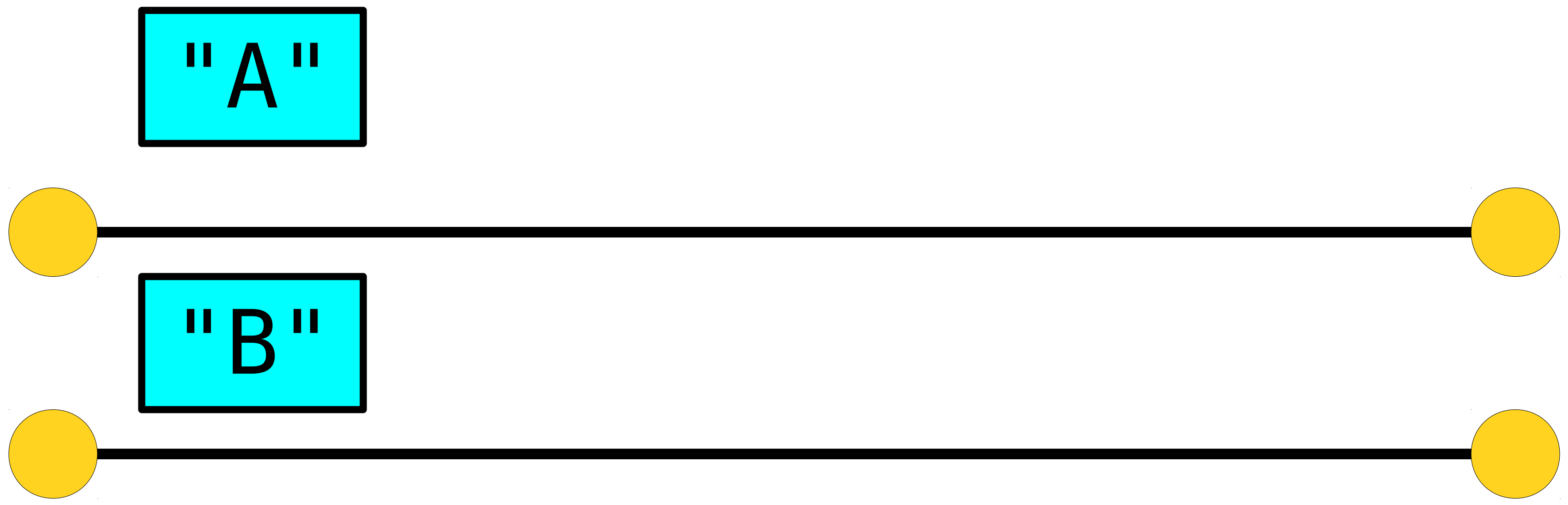

"A"
"B"


" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

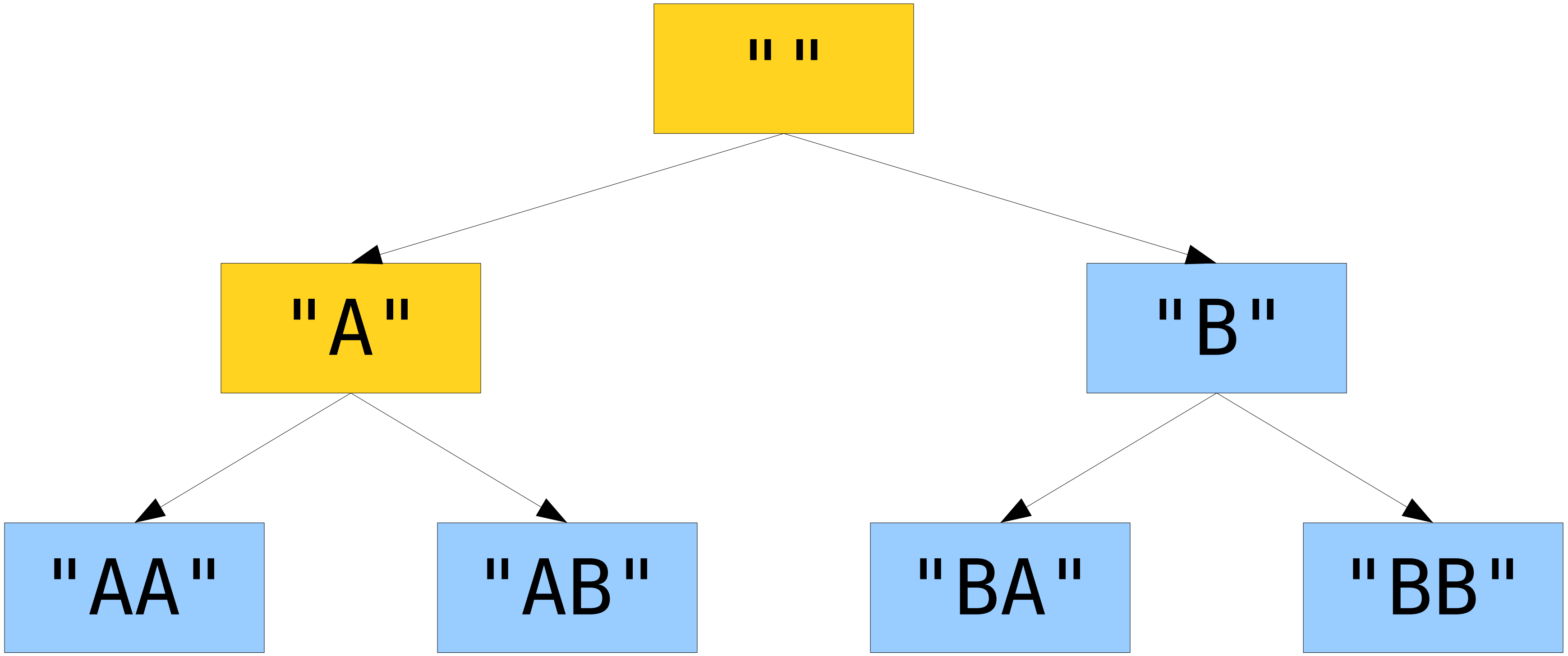

"A"
"B"

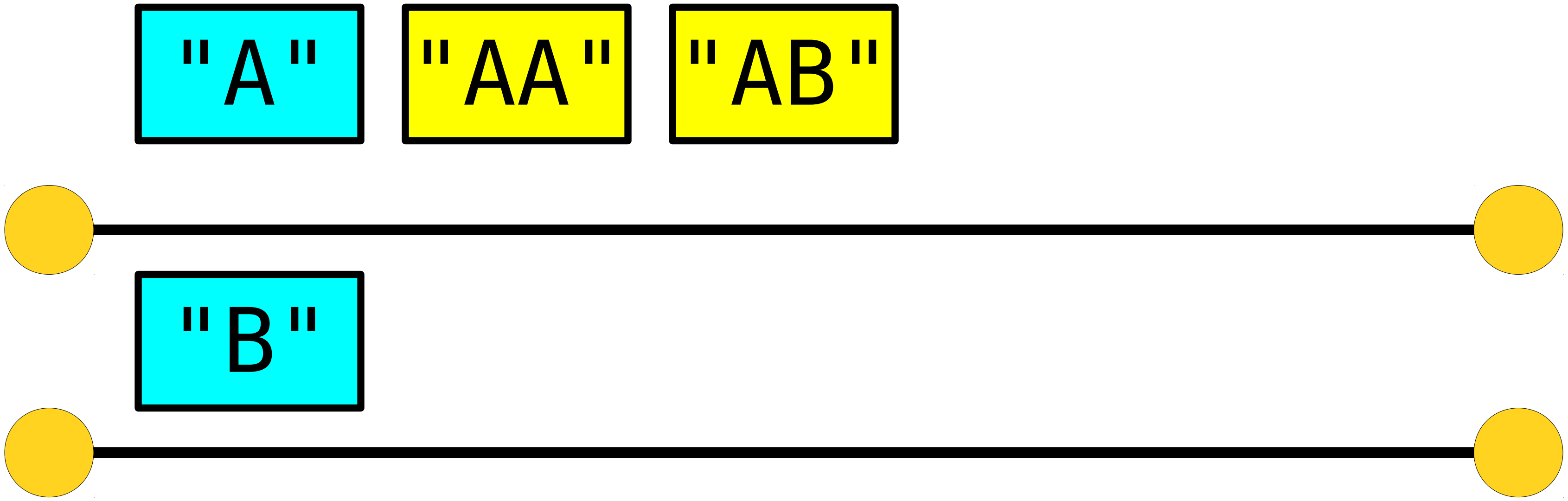


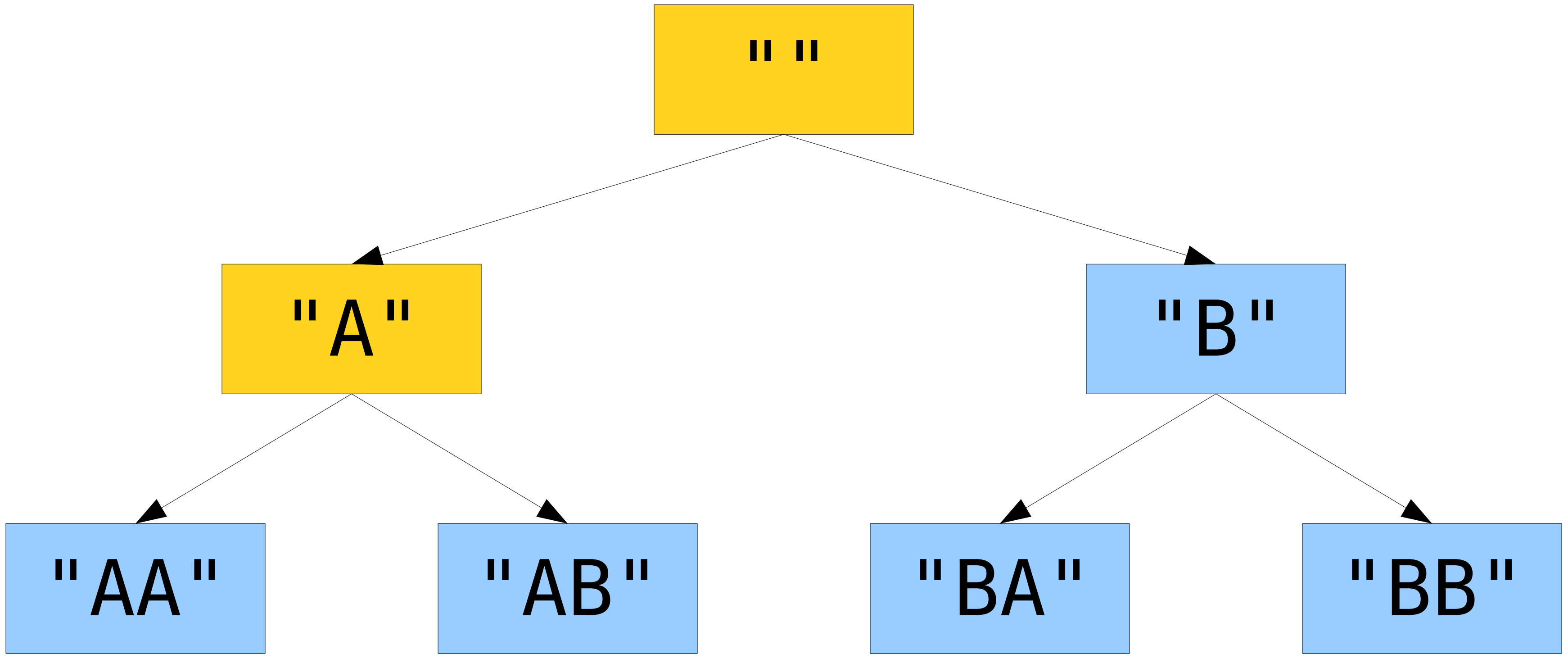
" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

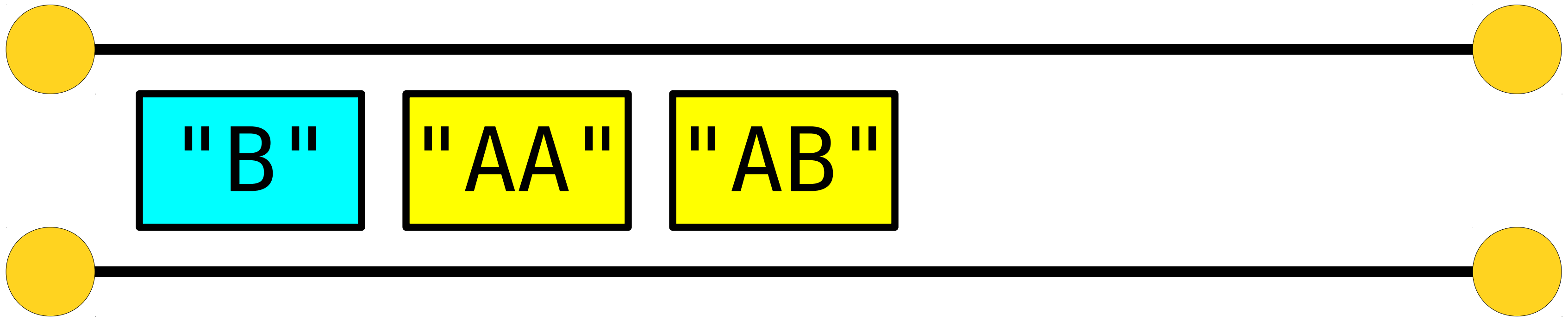


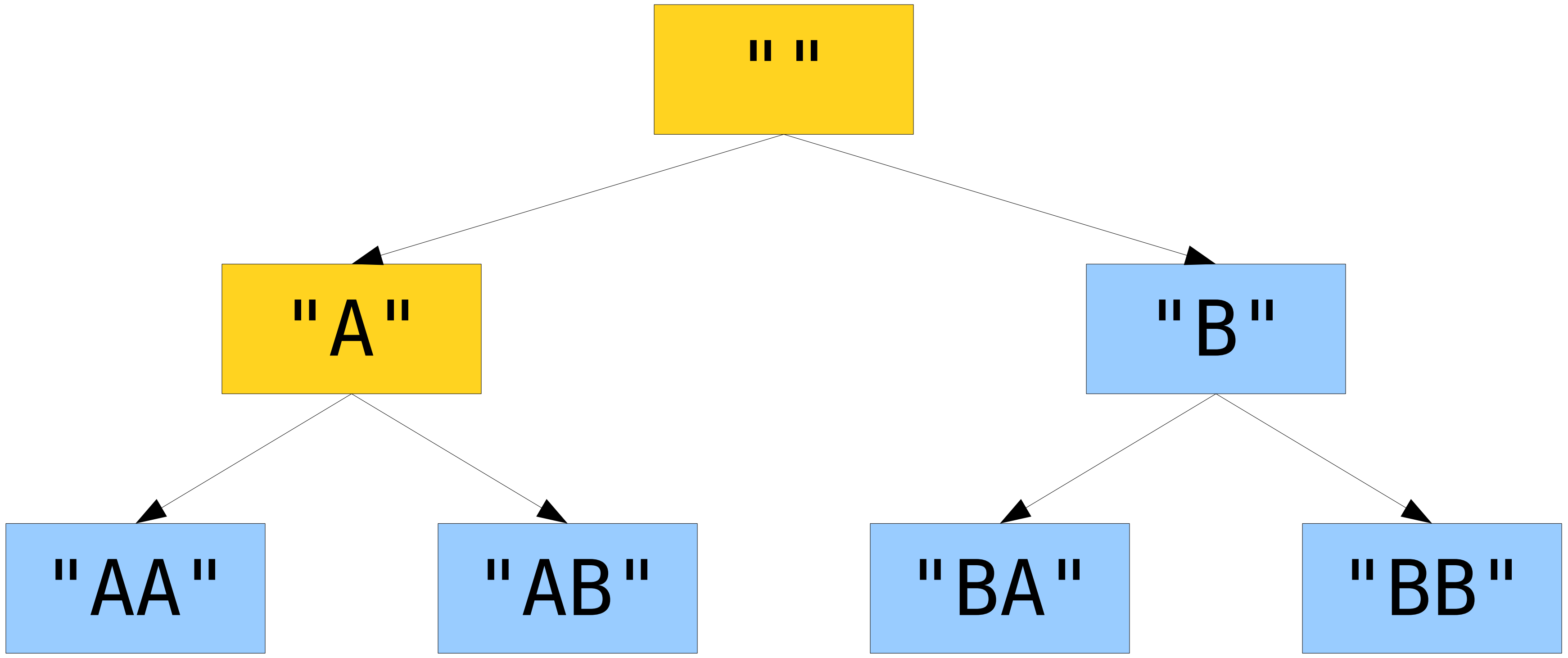
"A"
"B"

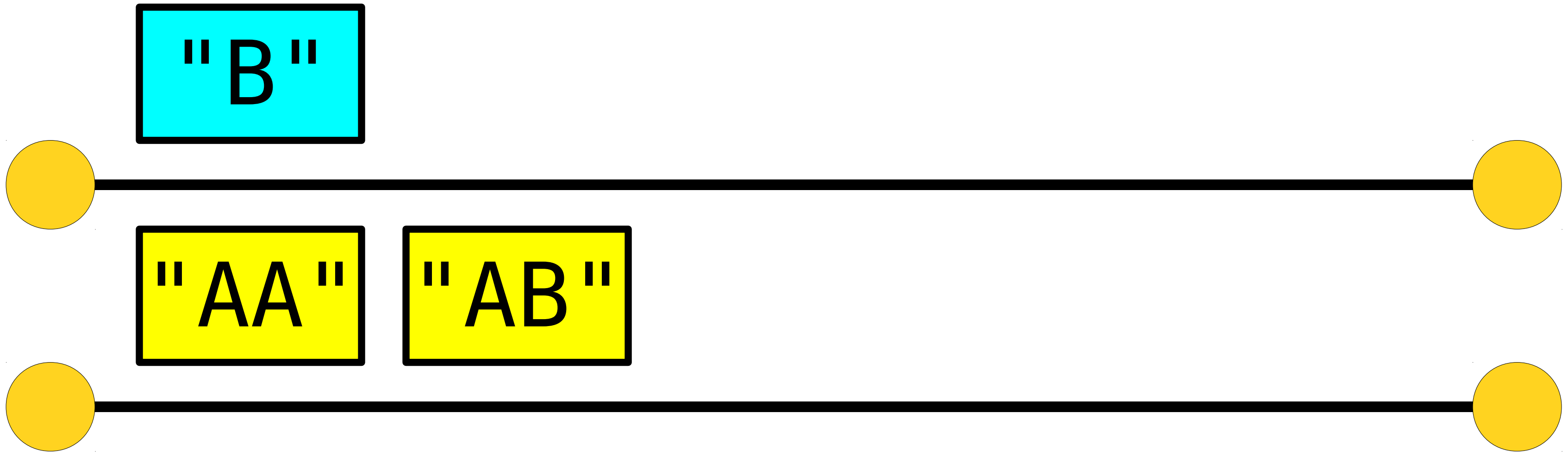

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

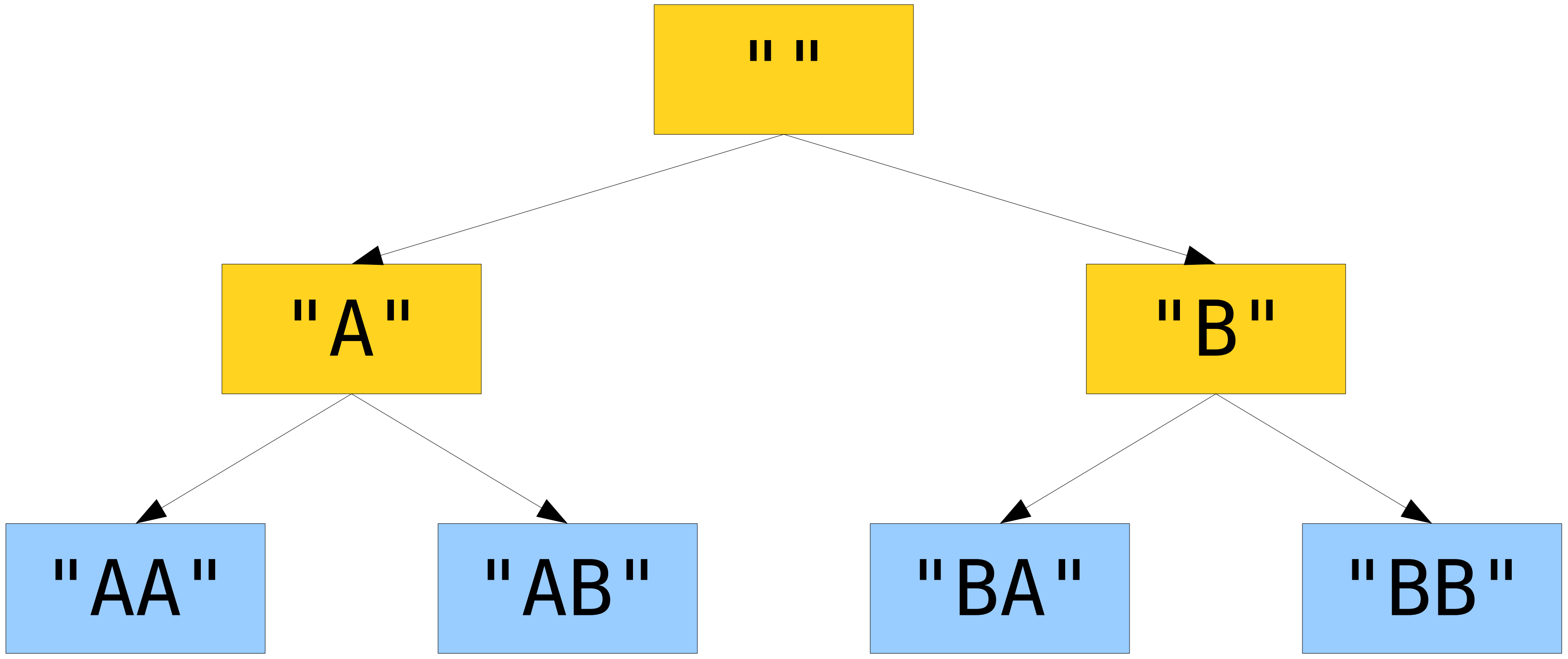
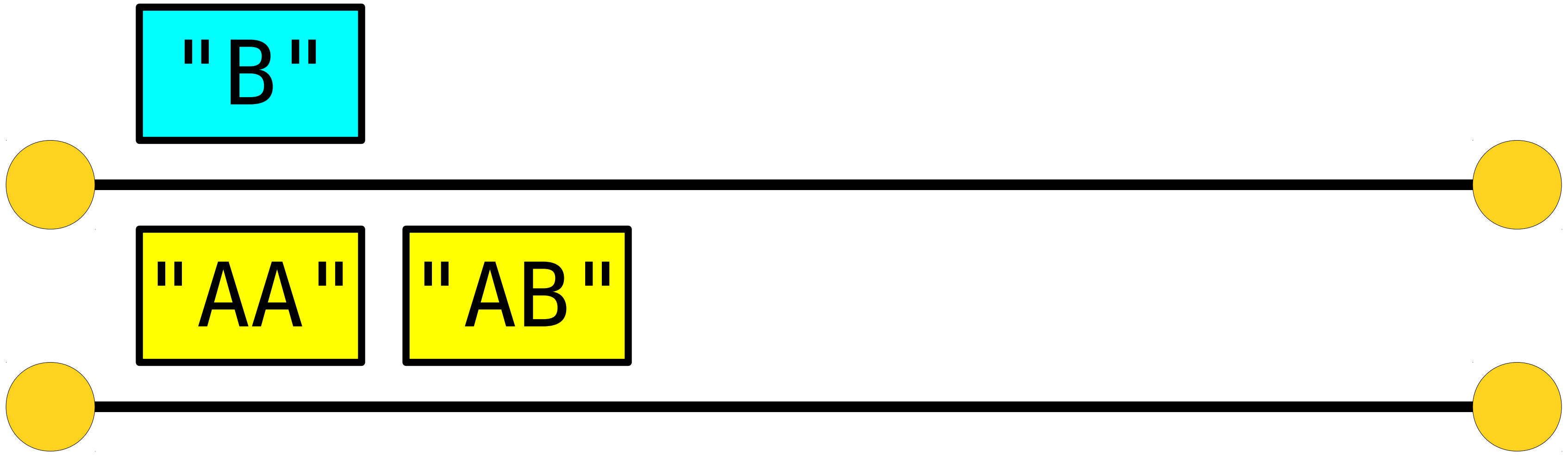

"A"
"AA"
"AB"
"B"

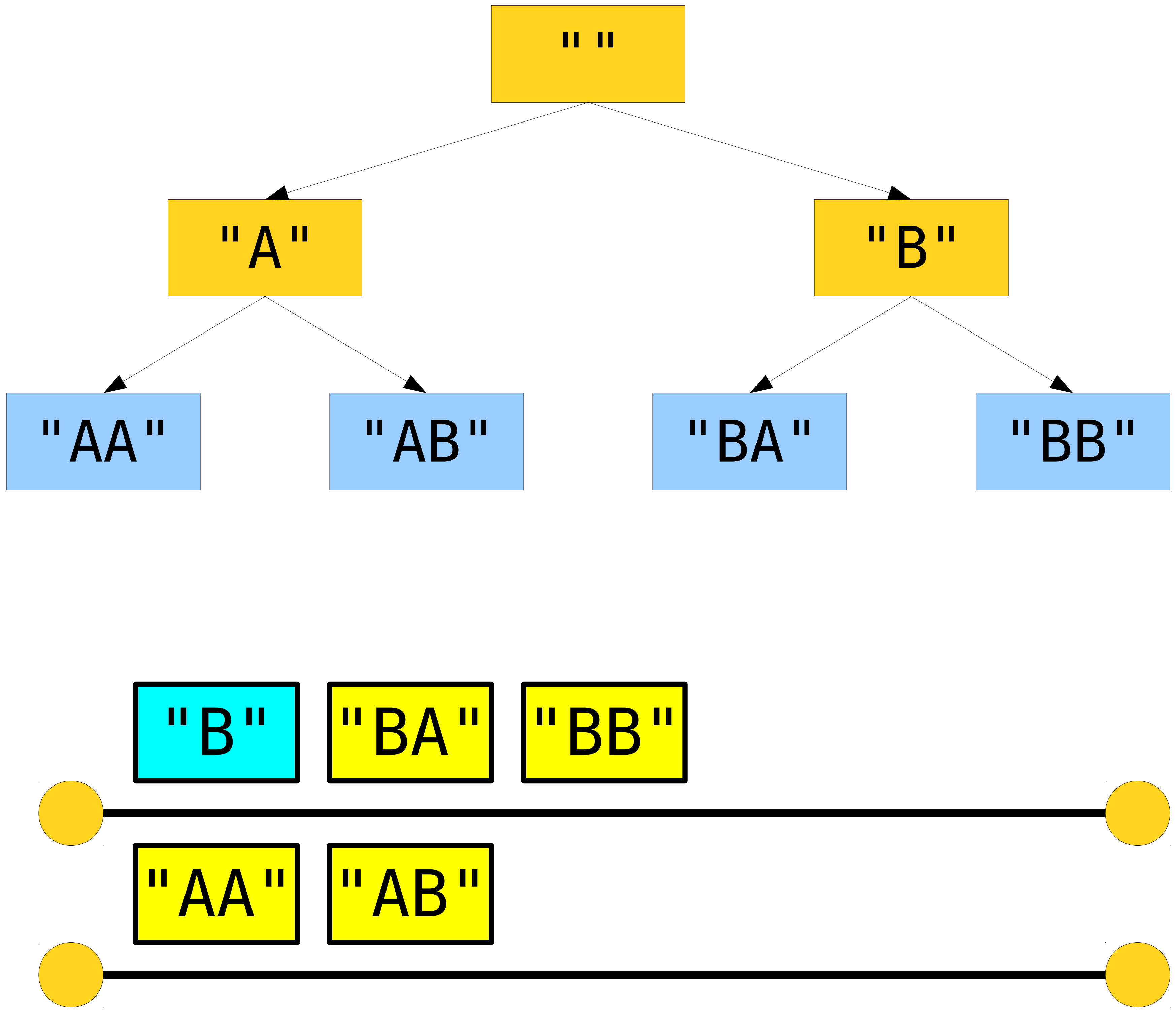

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

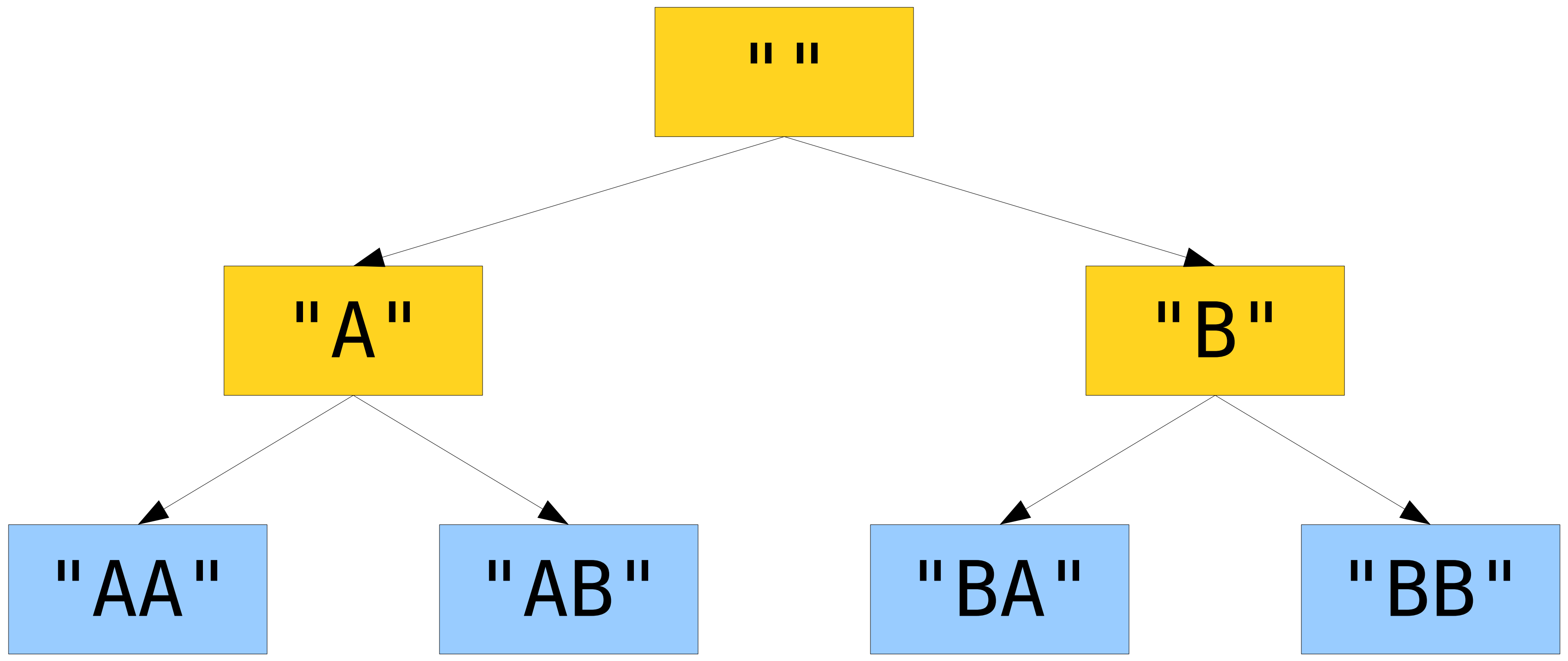

"B"
"AA"
"AB"

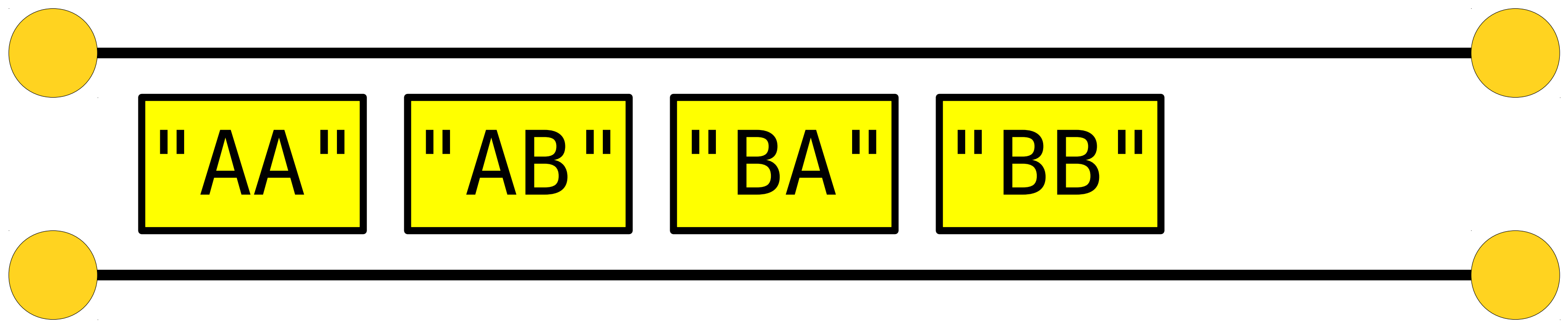

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

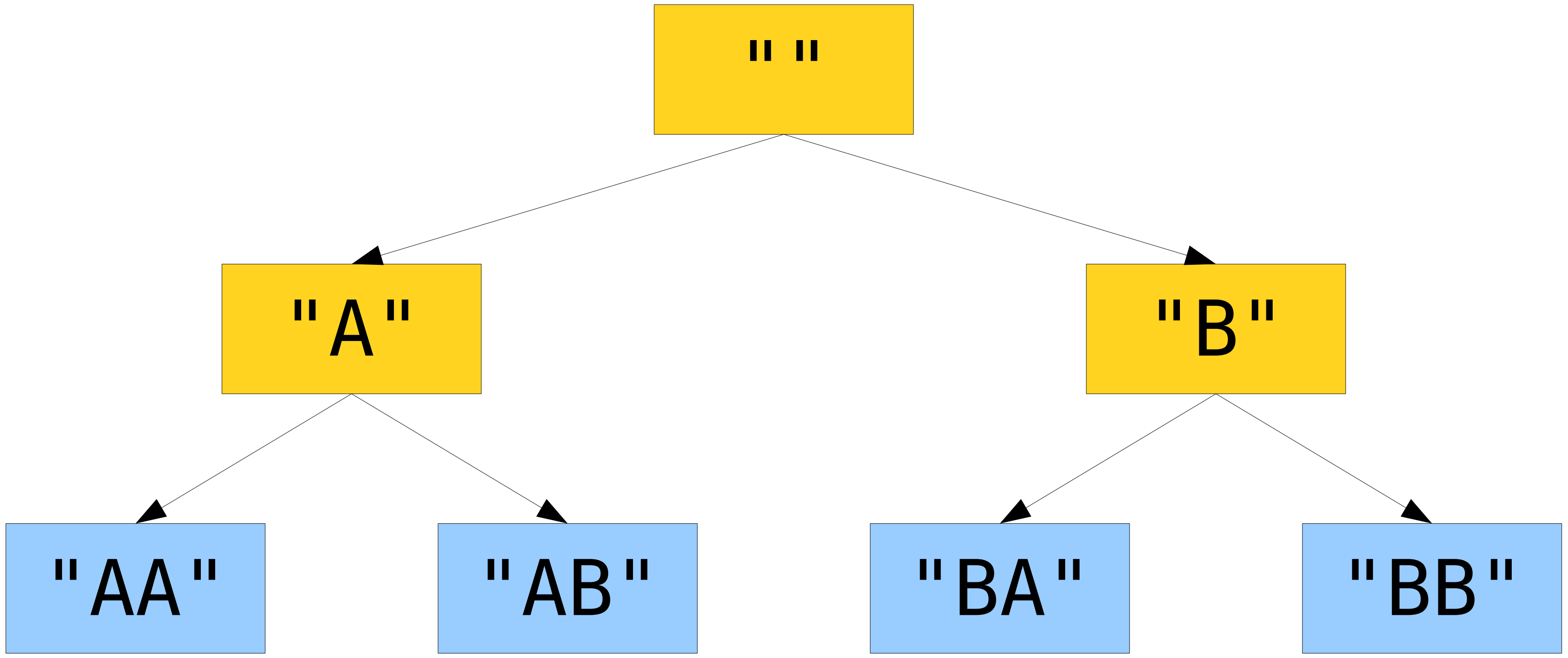

"B"
"AA"
"AB"

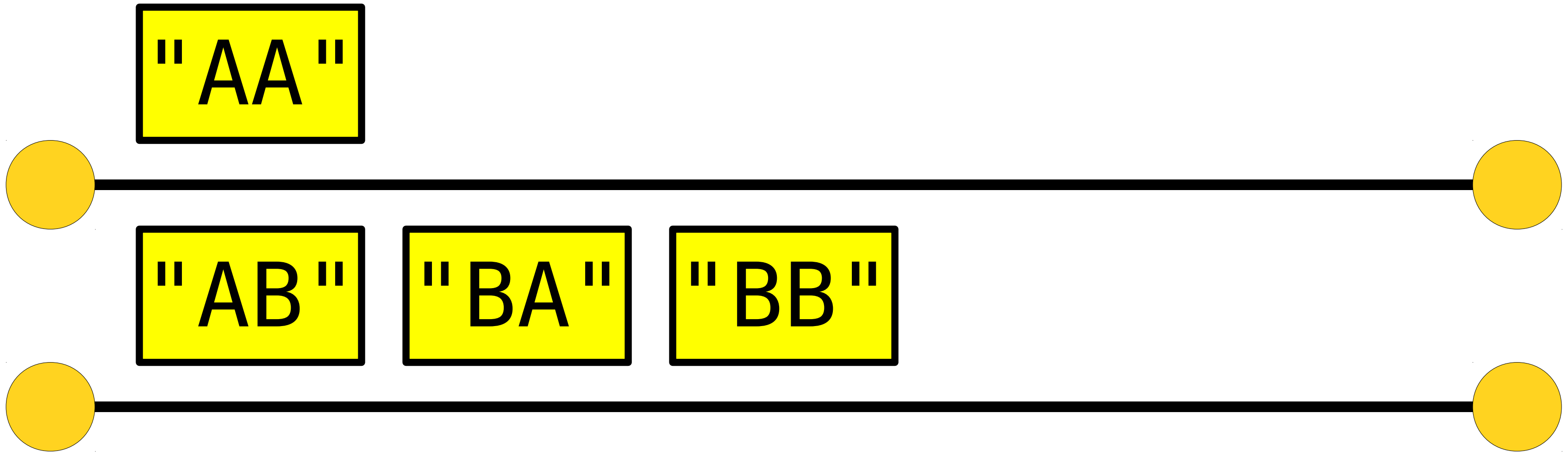

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

"B"
"AA"
"AB"

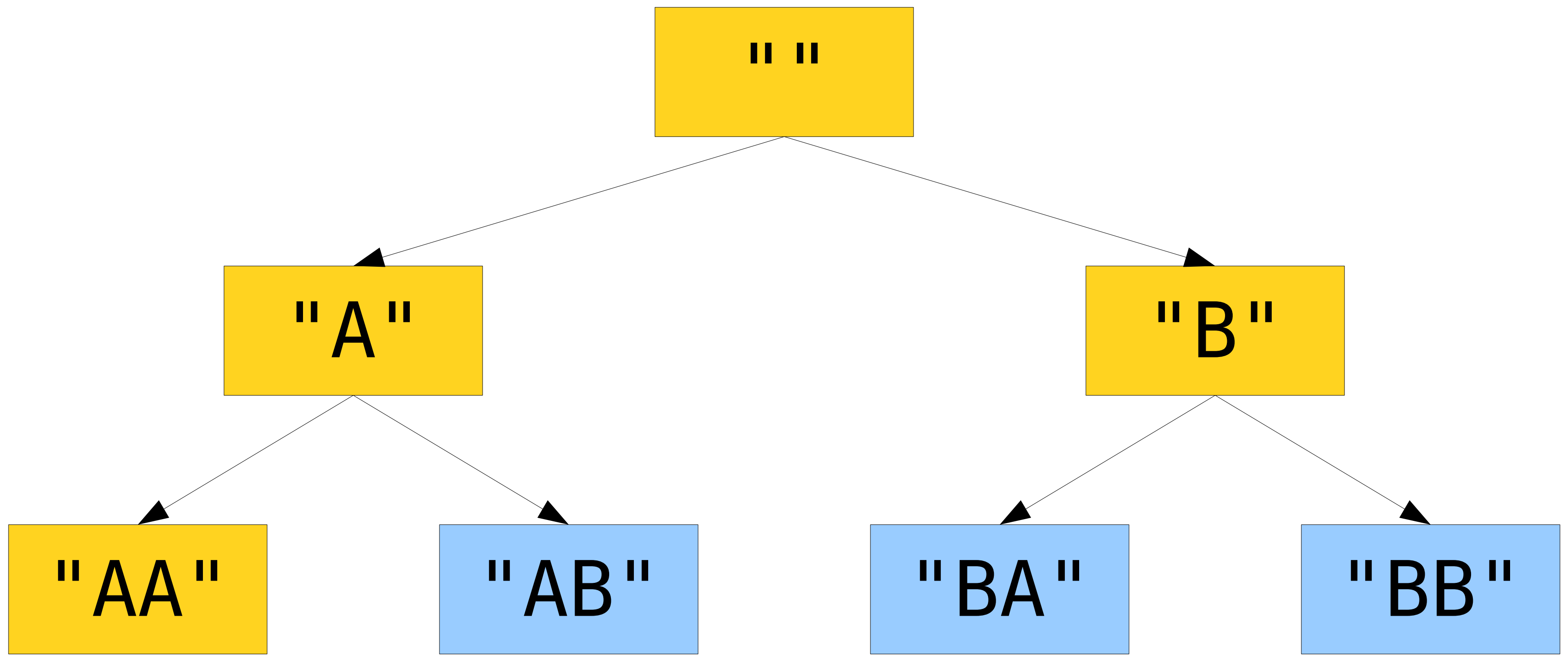

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"
"B"
"BA"
"BB"
"AA"
"AB"

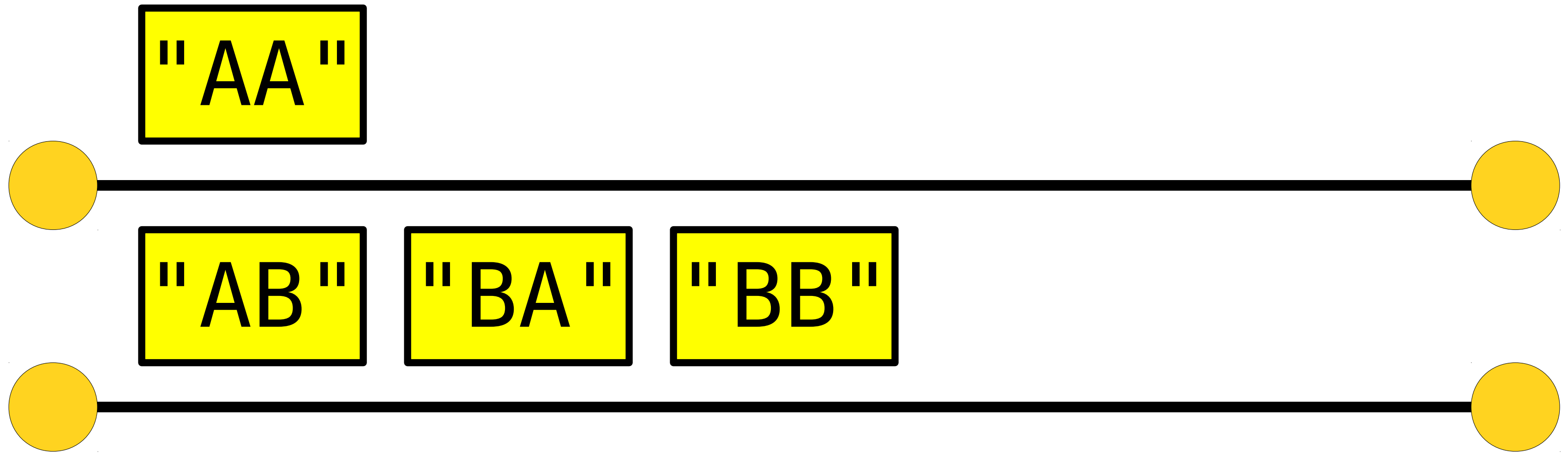

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

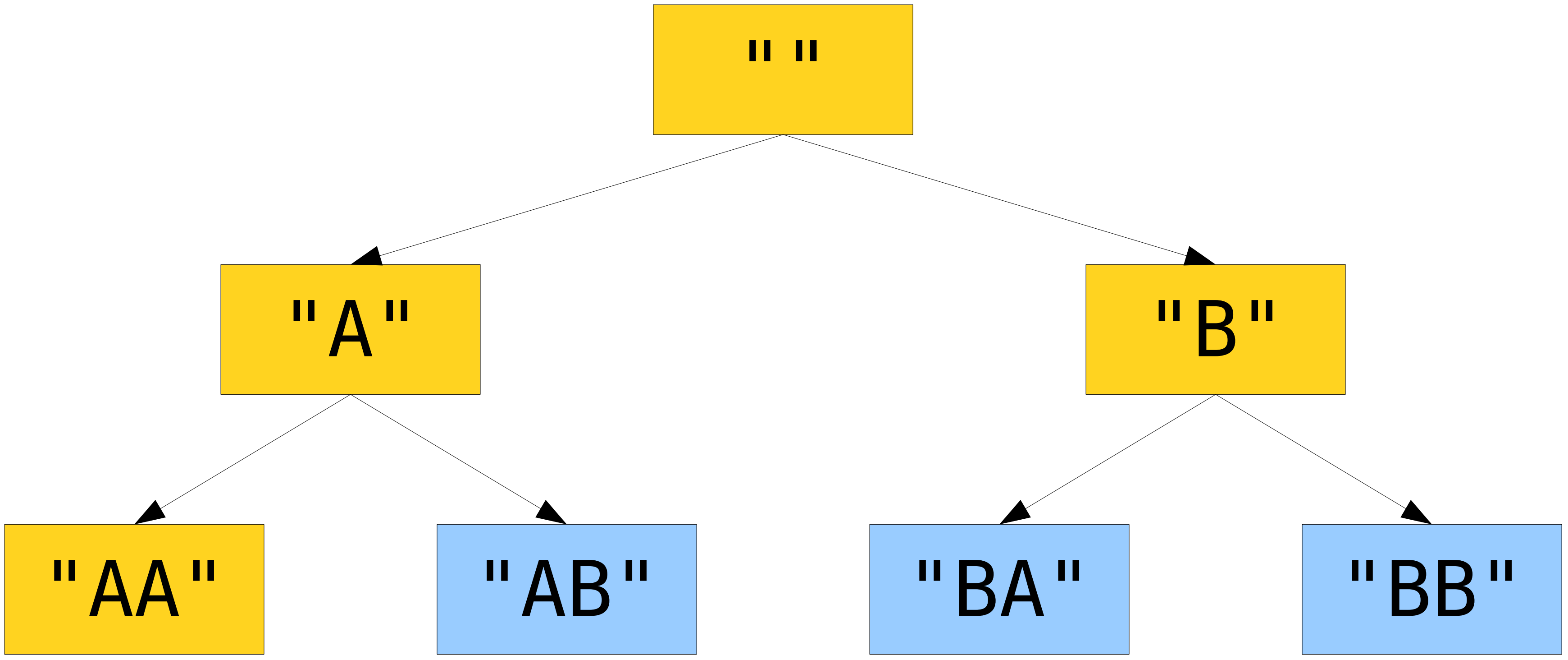

"AA"
"AB"
"BA"
"BB"

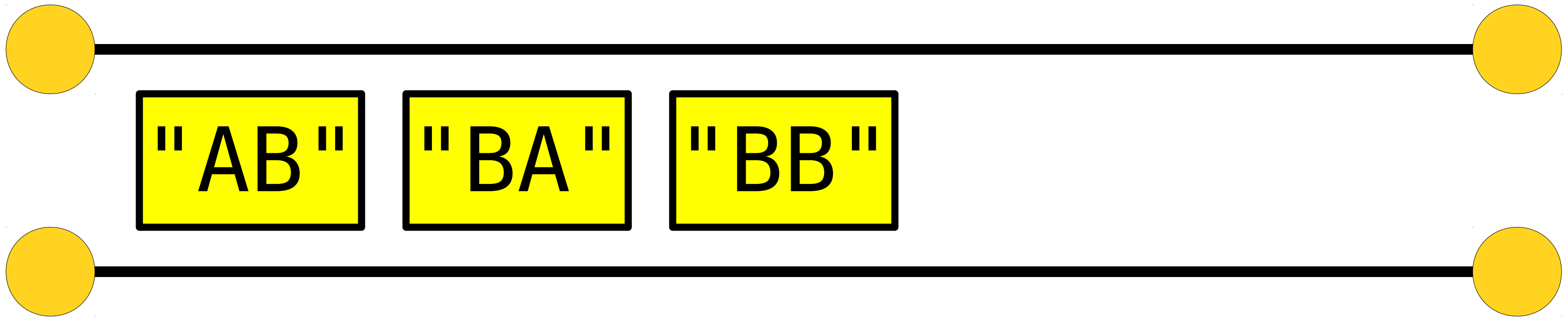

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

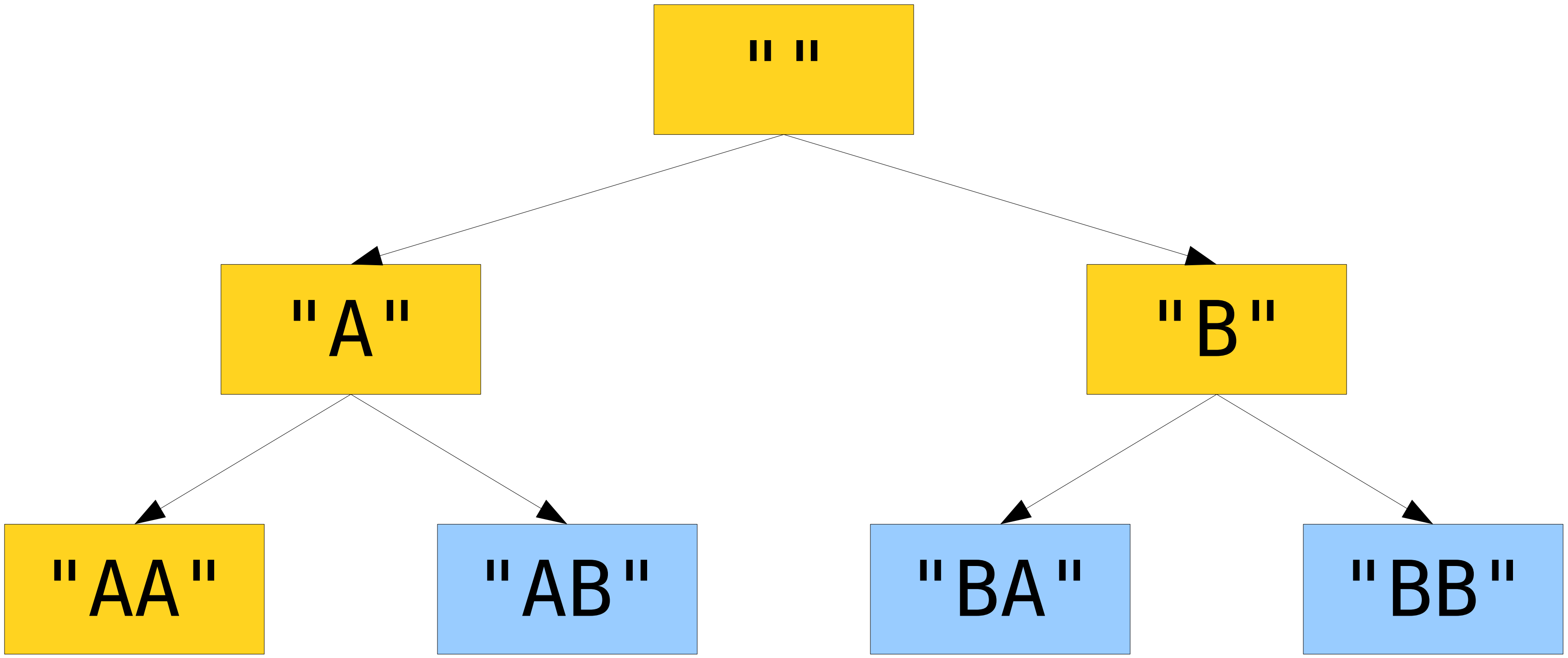
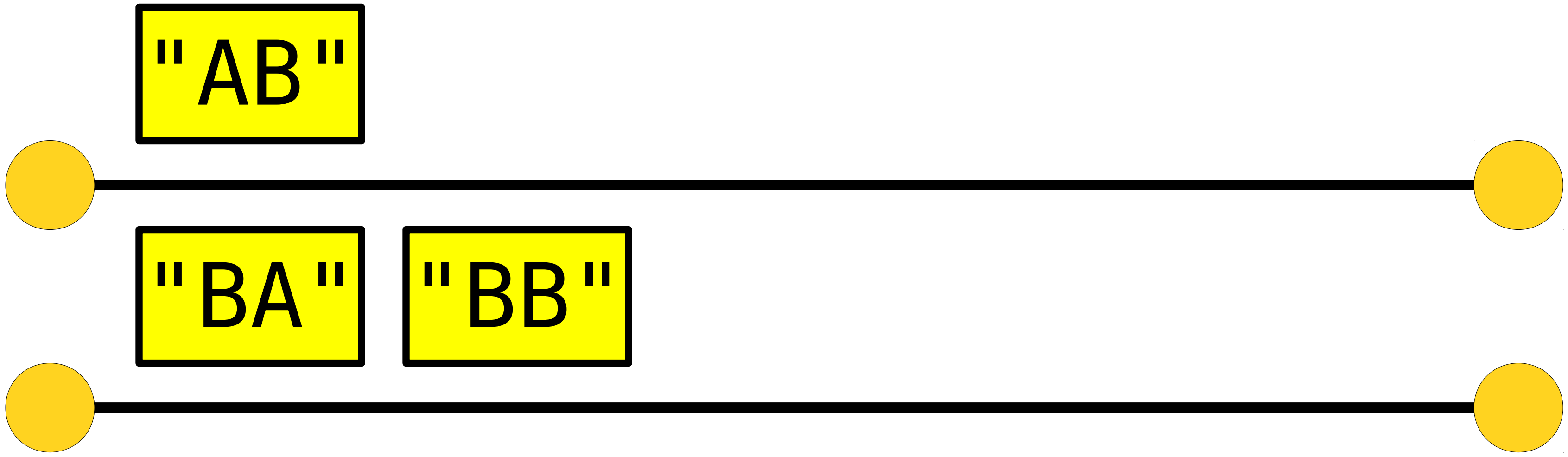

"AA"
"AB"
"BA"
"BB"

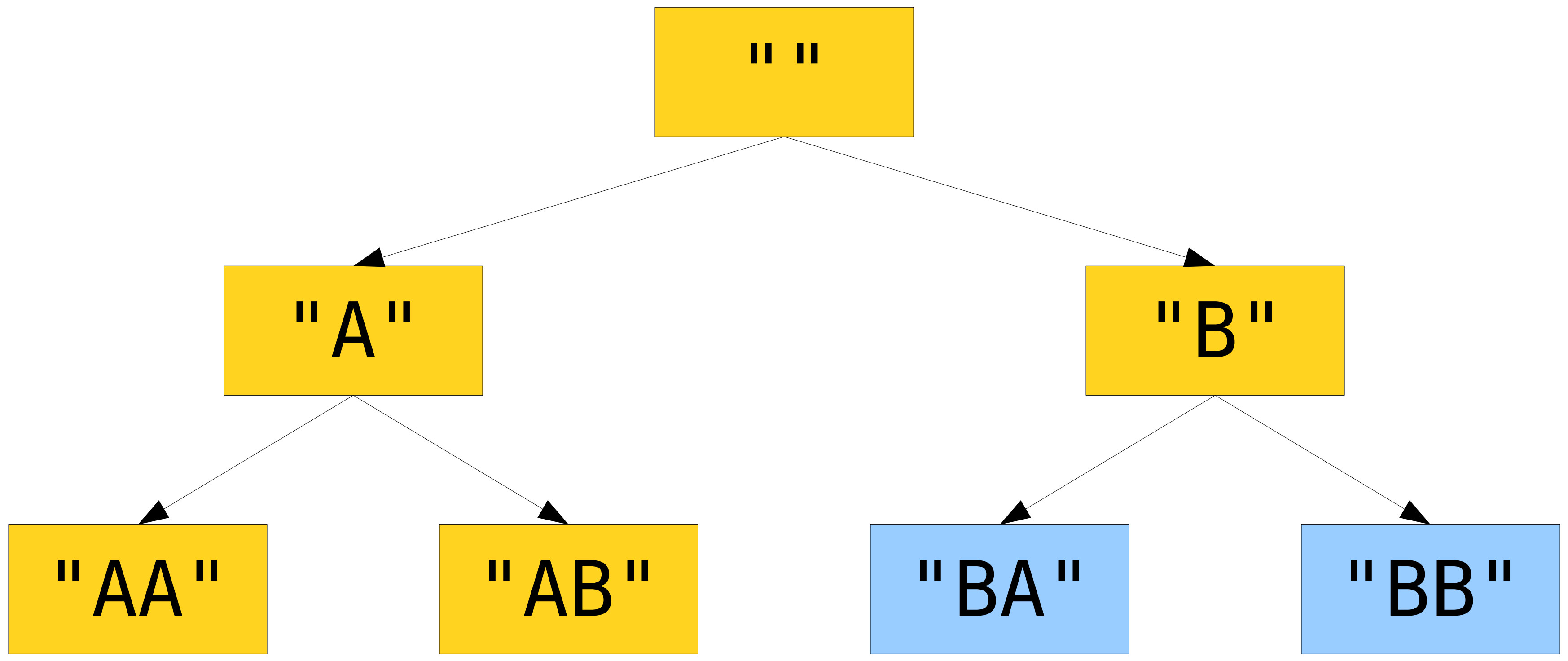
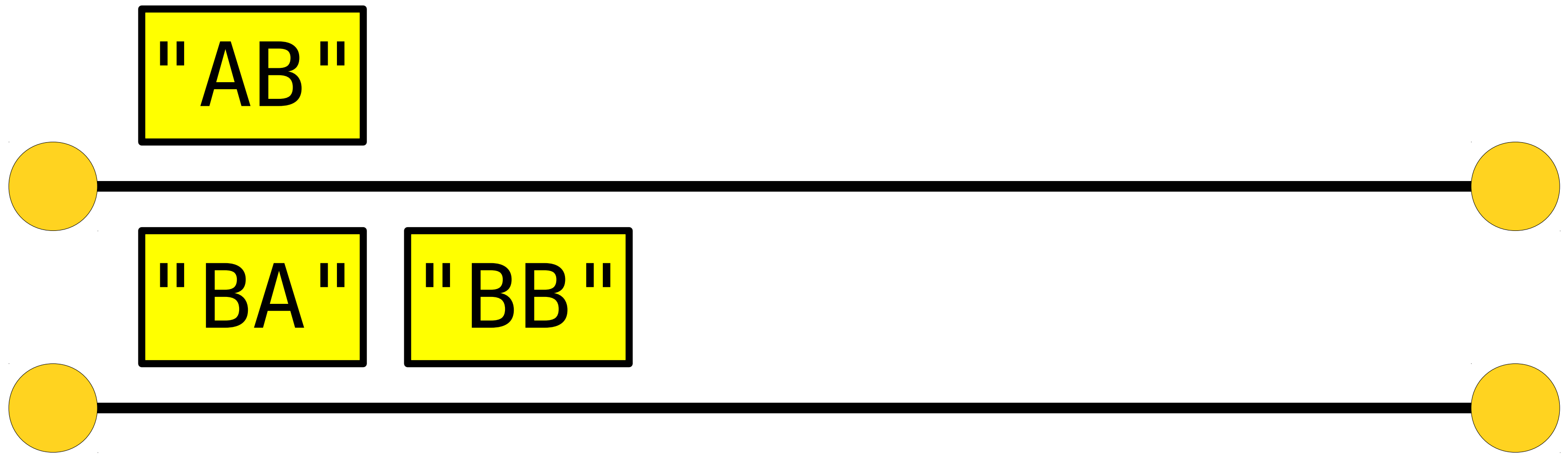

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

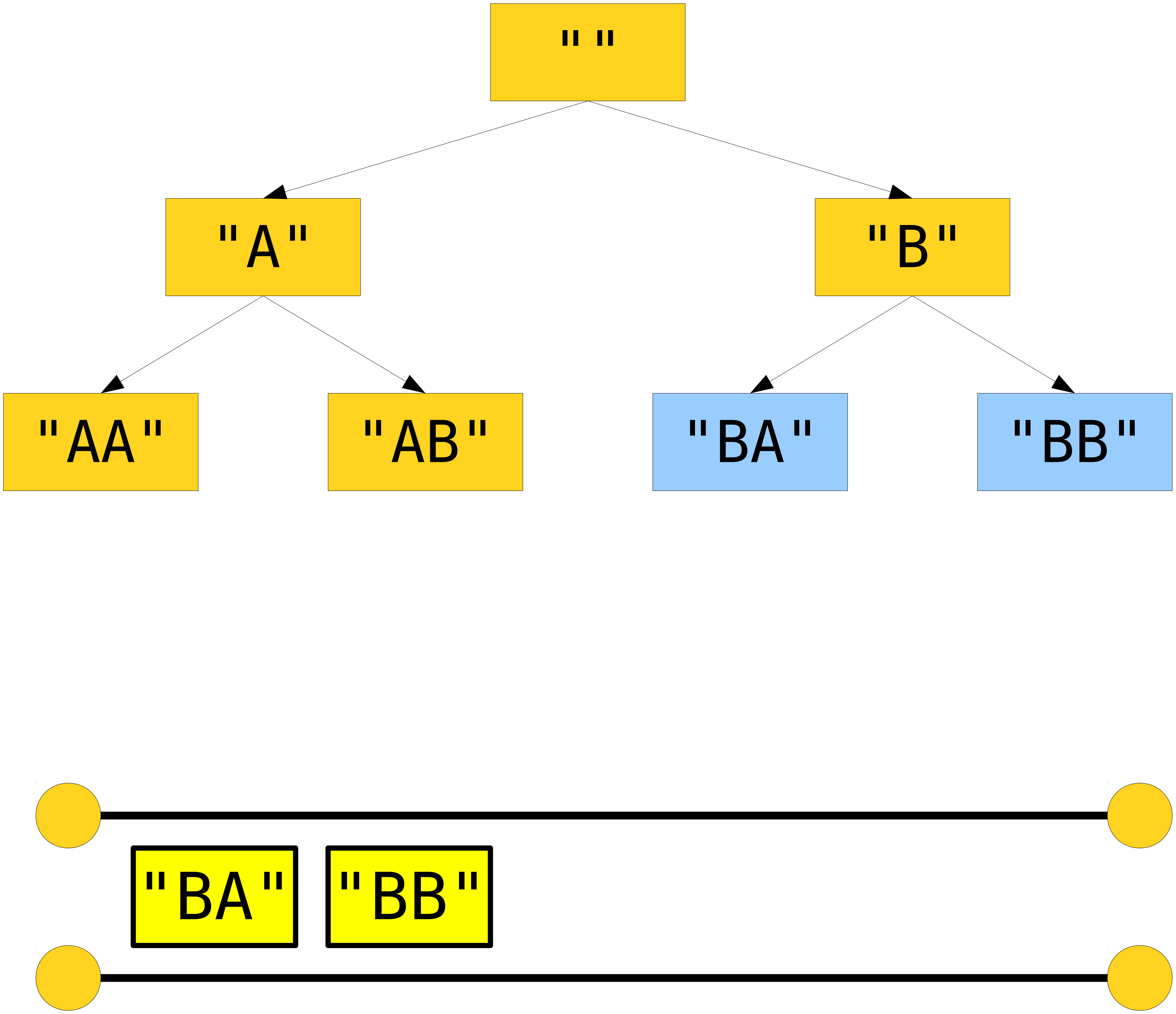

"AA"
"AB"
"BA"
"BB"

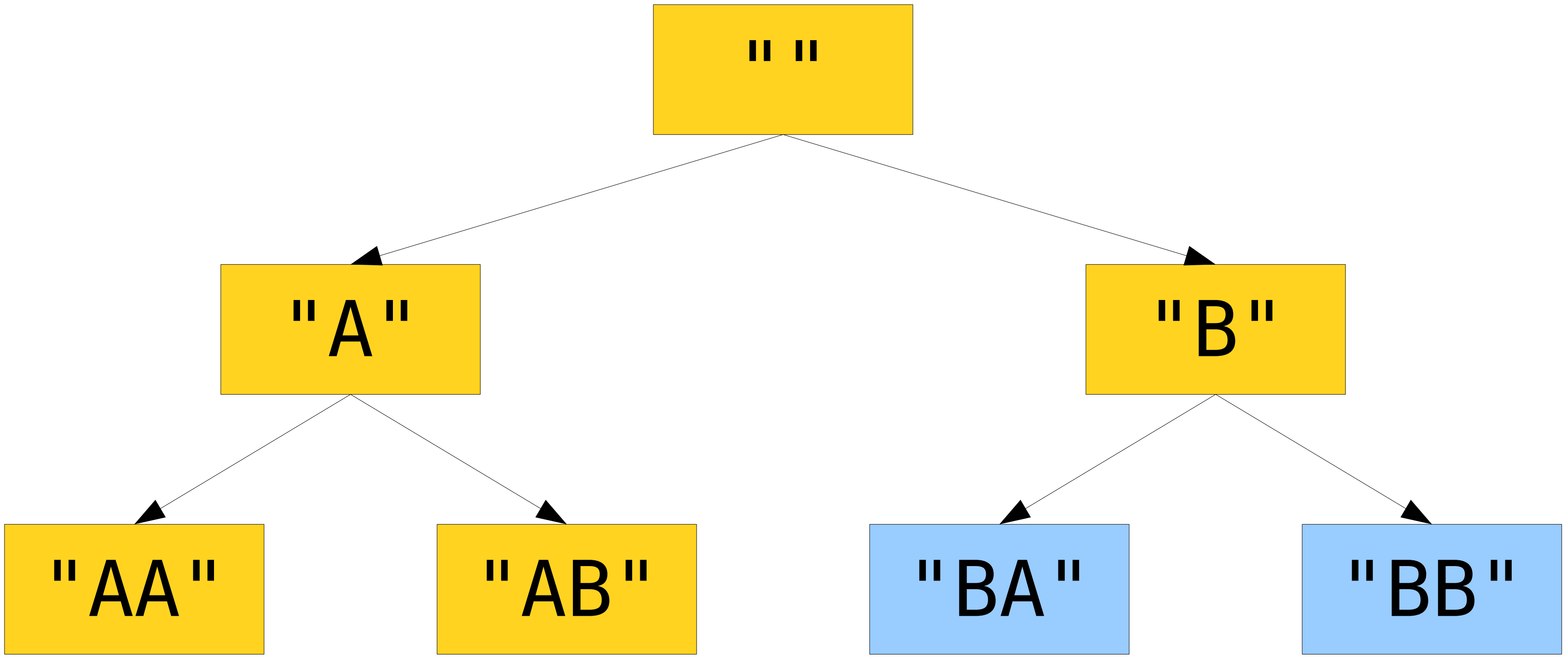
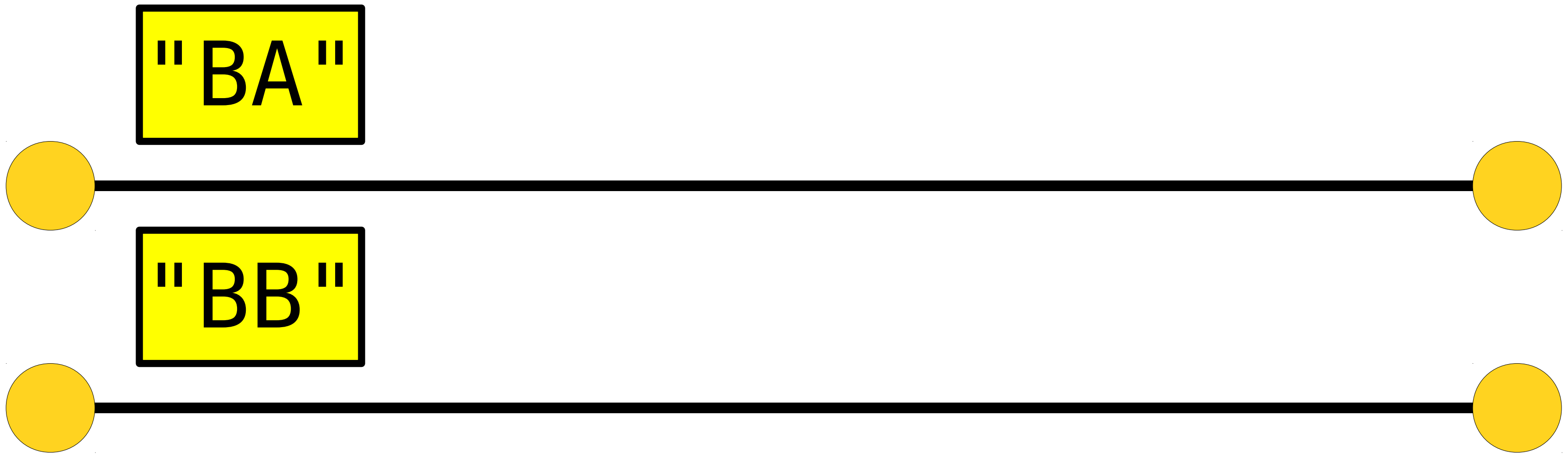

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

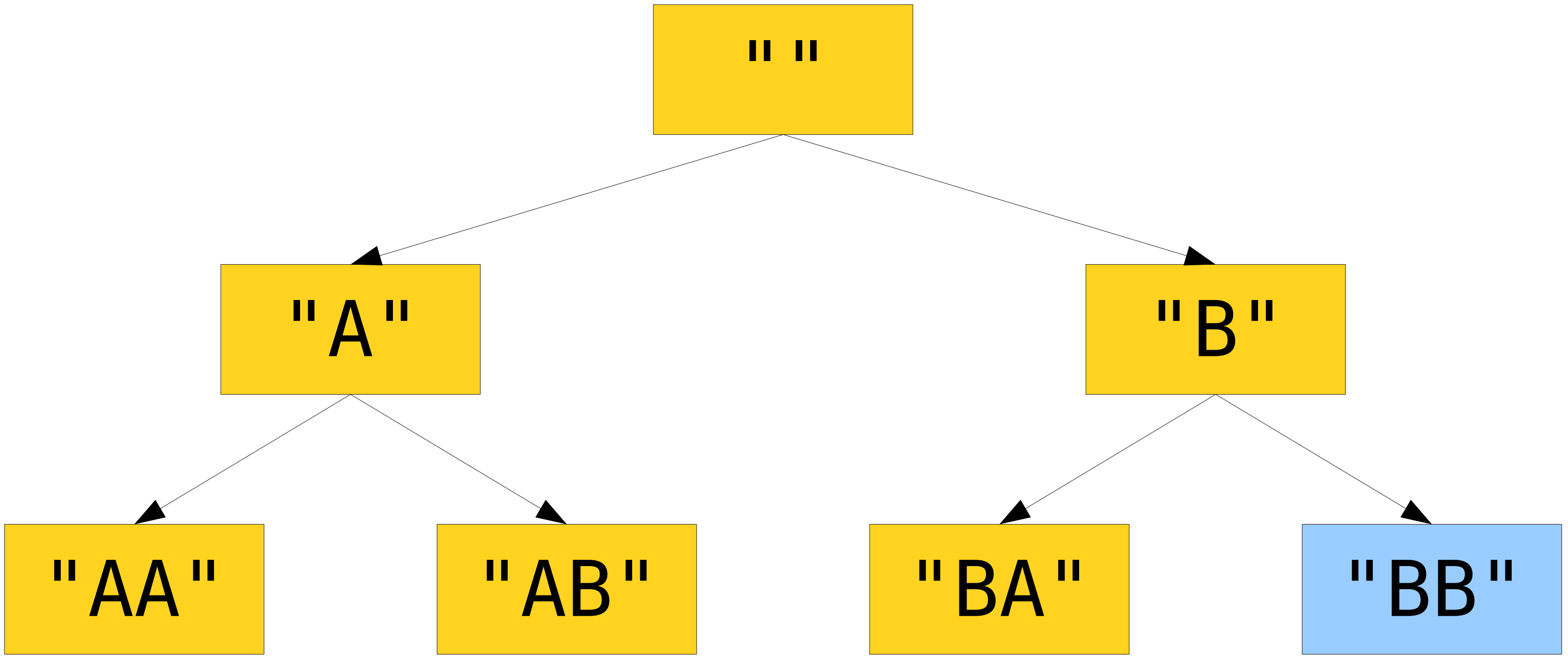
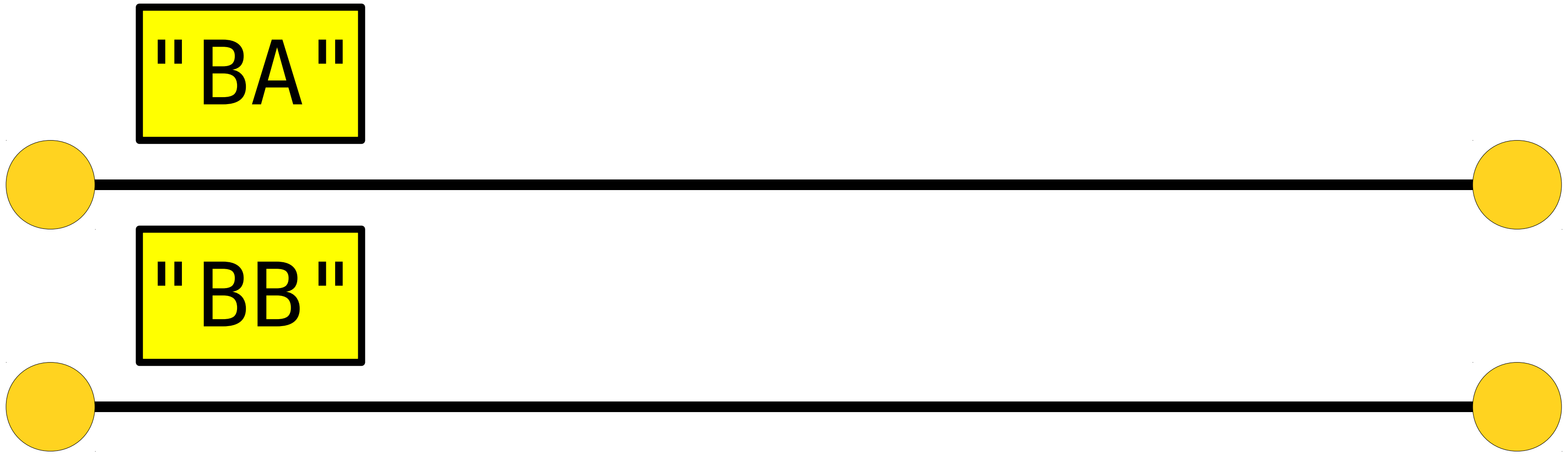

"AB"
"BA"
"BB"

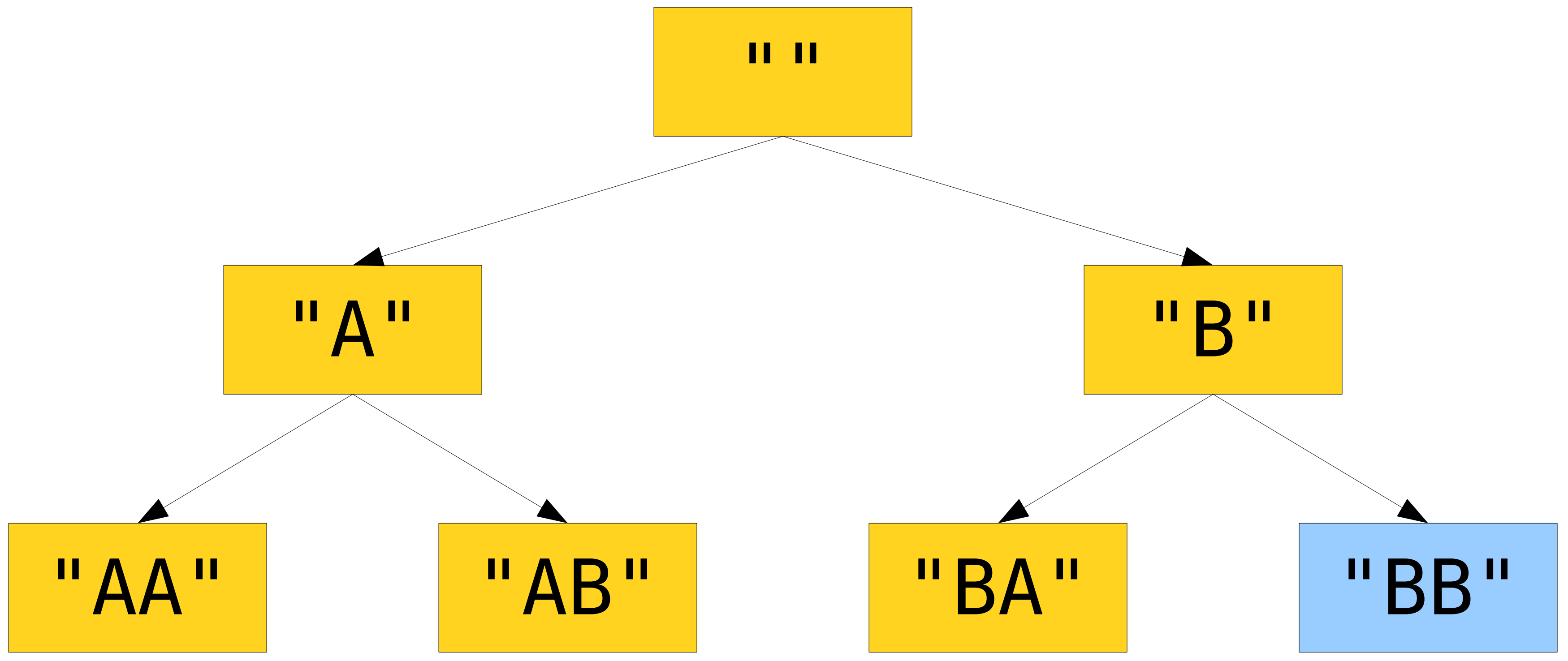

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

"AB"
"BA"
"BB"

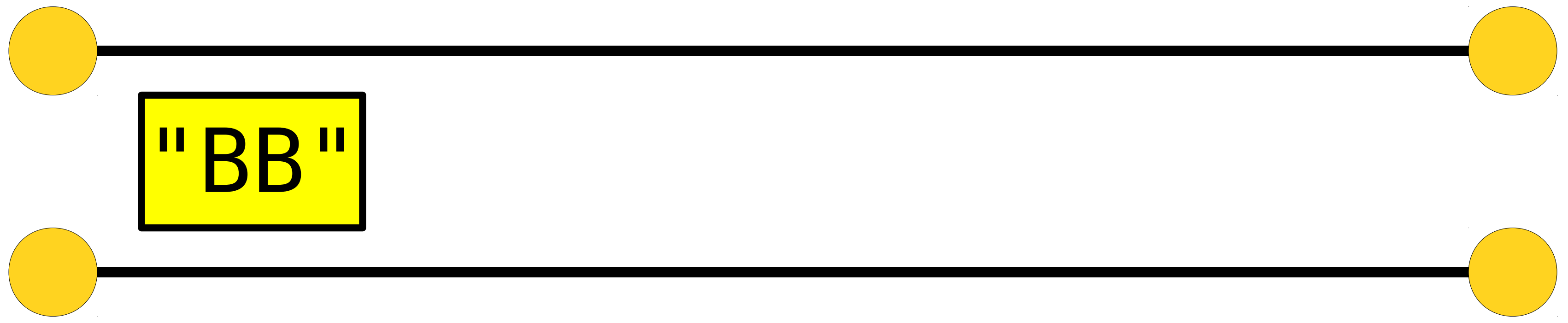

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

"AB"
"BA"
"BB"

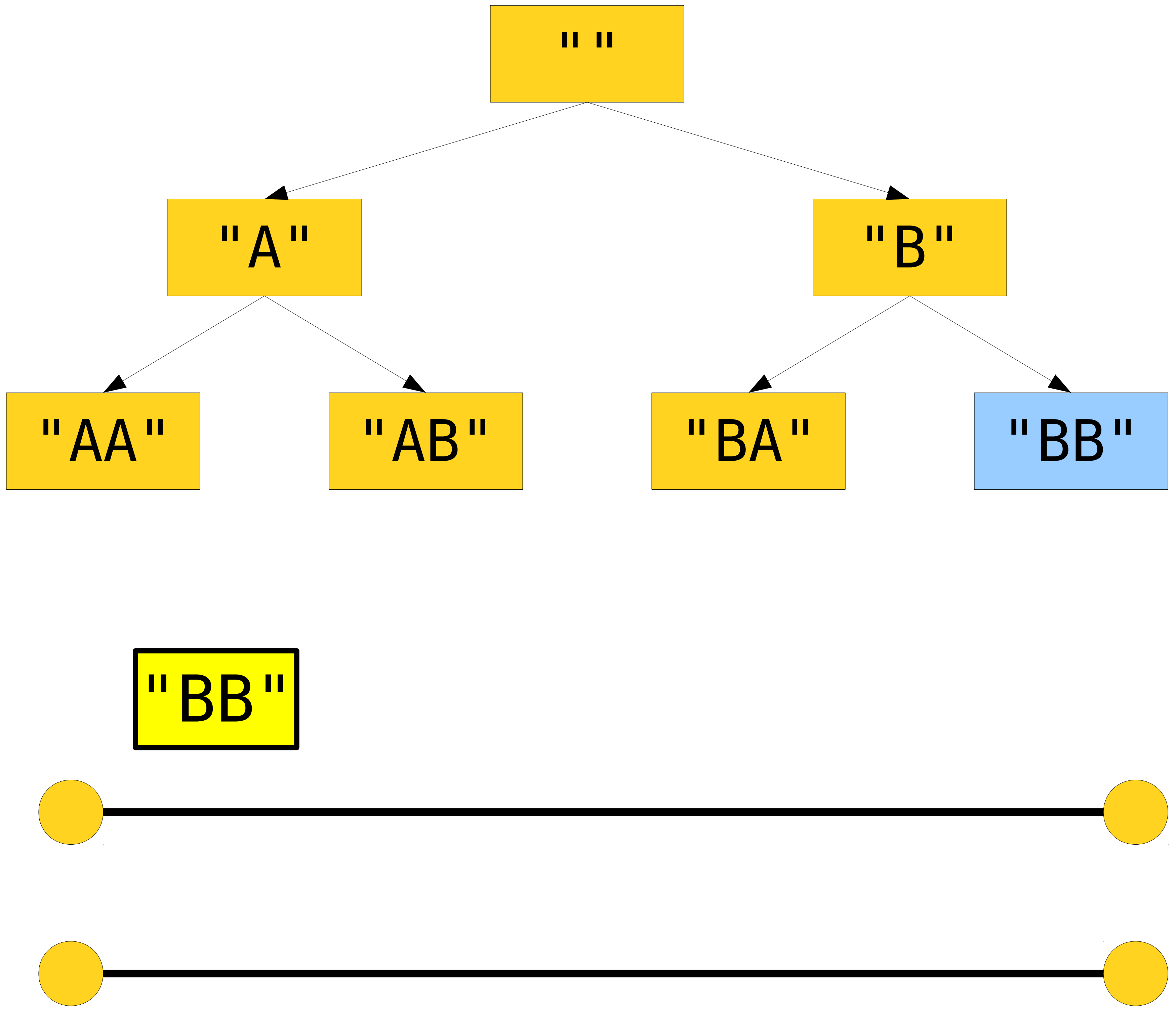

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"
"BA"
"BB"

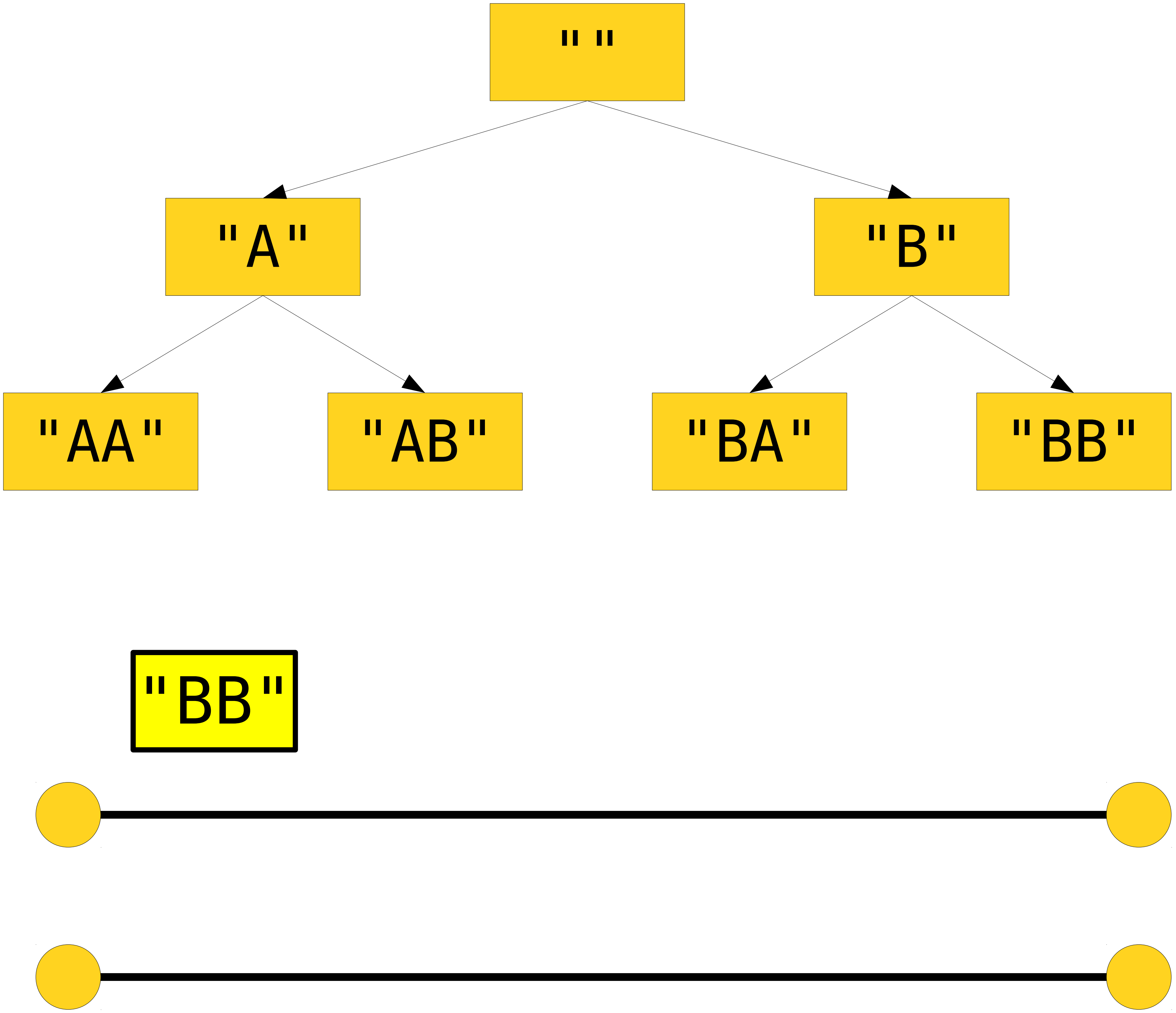

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

"BA"
"BB"

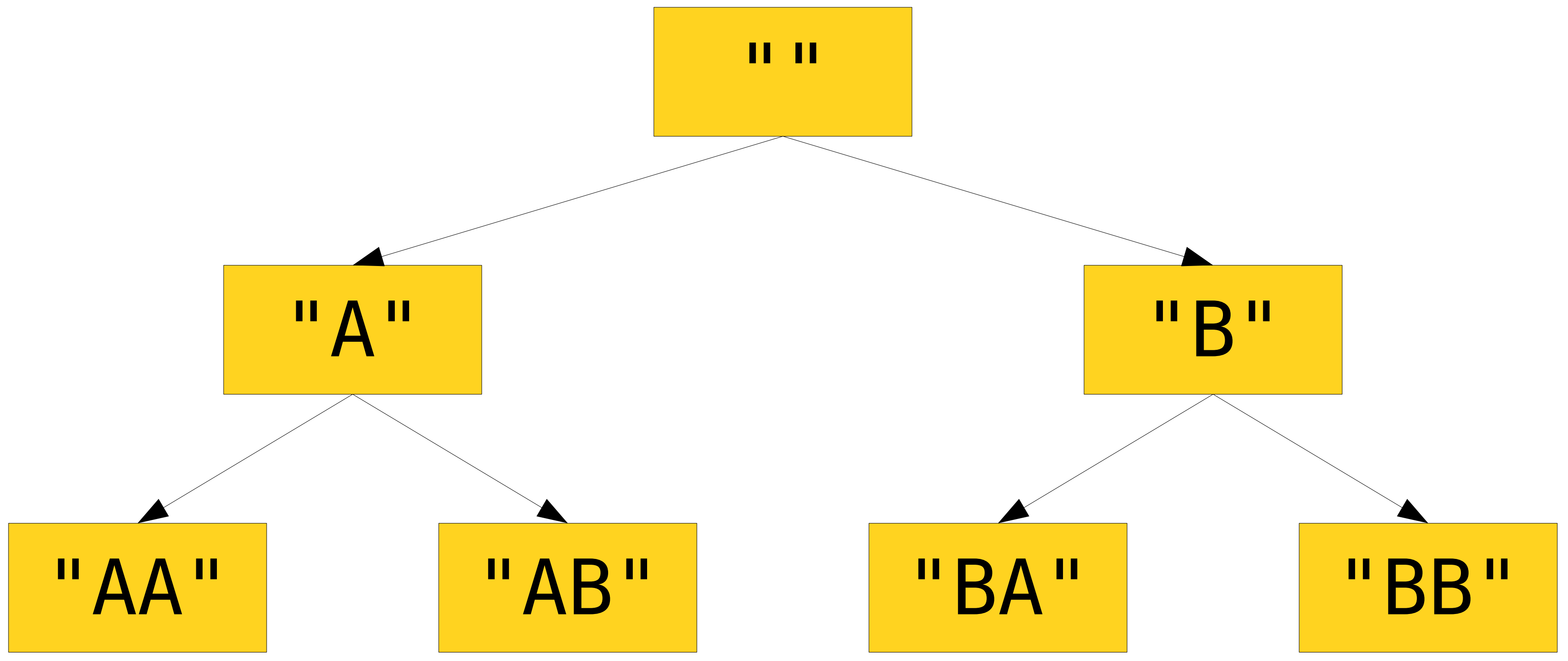

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"

"BA"
"BB"

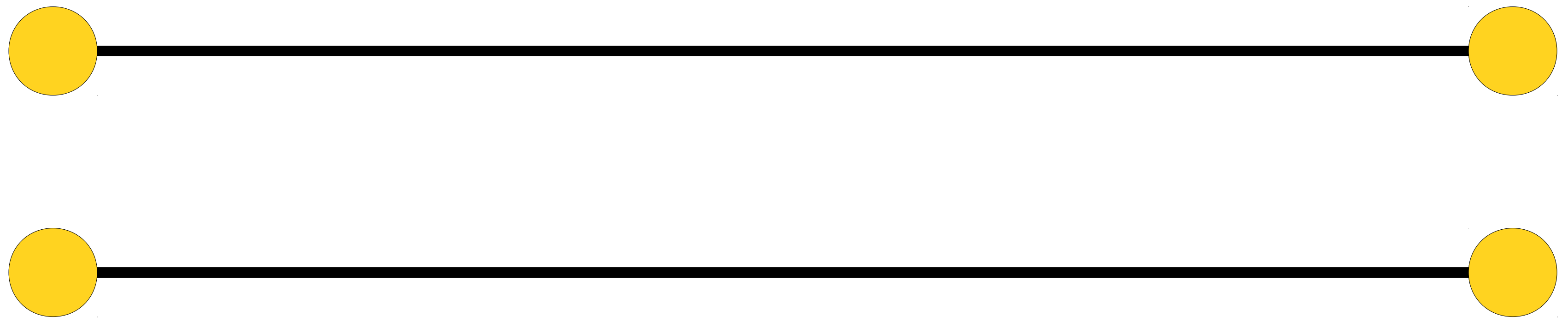

" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"


"BB"


" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"
"BB"


" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"
"BB"


" "
"A"
"B"
"AA"
"AB"
"BA"
"BB"



Let’s code it up!

Our Algorithm

- Start with the empty string in a queue.
- While there are still things in the queue:
  - Pull one off.
  - Print it.
  - If it isn't at the maximum length:
    - For each possible next character, enqueue the string formed by tacking the character onto the end of the recently-dequeued string.

Breadth-First Search

- This general algorithm is called ***breadth-first search***.
- It’s often used to find the fastest or best way to do something, since it lists objects in increasing order of “size.”
- The algorithm that Google Maps uses is closely related to this algorithm. Stay tuned for details!
- You’ll see some other applications of this algorithm in Assignment 2.

Your Action Items

- Read Chapter 5 of the textbook, which talks about container classes.
- Finish Assignment 1. It’s due on Friday.
  - Read the style guide up on the course website for more information about good programming style.
  - Review the Assignment Submission Checklist to make sure your code is ready to submit.

Next Time

- ***Associative Containers***
  - Data sets aren’t always linear!
- ***Maps, Sets, and Lexicons***
  - Three ways to organize information.