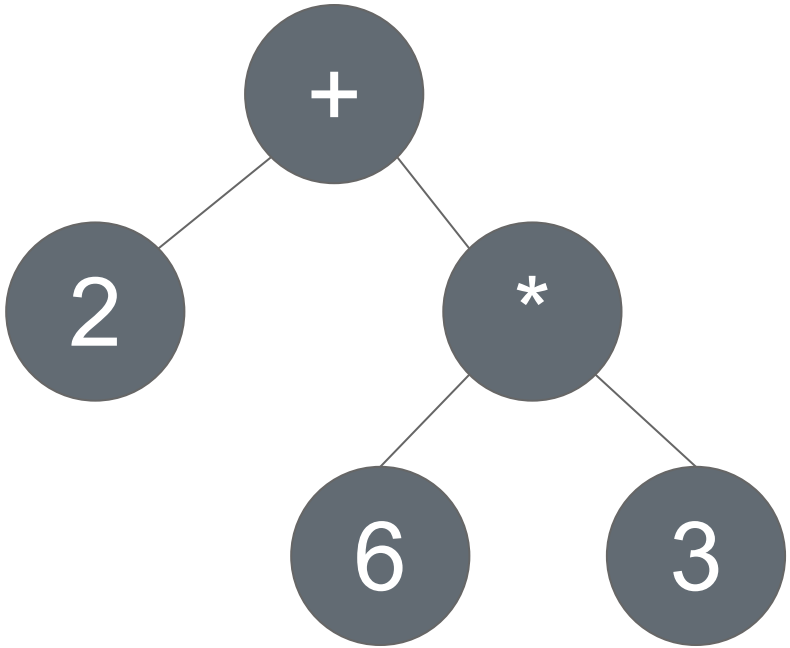
AST

Myfind 2020

Anis Ladram / Adrien Hellec

Abstract Syntax Tree

```
2 + 6 * 3
```



Evaluate the AST

Evaluate the AST

```
enum node_type
{
    ADD,
    MULT,
    NUMBER
}
```

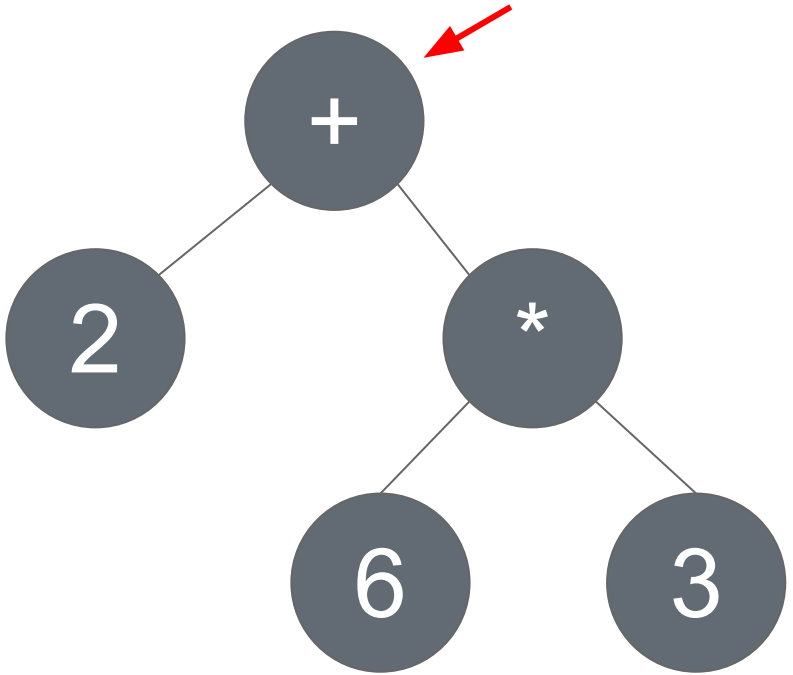
```
struct node
{
    enum node_type type;
    struct node *left;
    struct node *right;
    int value;
};
```

Evaluate the AST

```
int evaluate(struct node* ast)
{
    if (node->type == ADD)
    {
        return evaluate(node->left) + evaluate(node->right);
    } else if (node->type == MULT)
    {
        return evaluate(node->left) * evaluate(node->right);
    } else
    {
        return node->value;
    }
}
```

Evaluate the AST

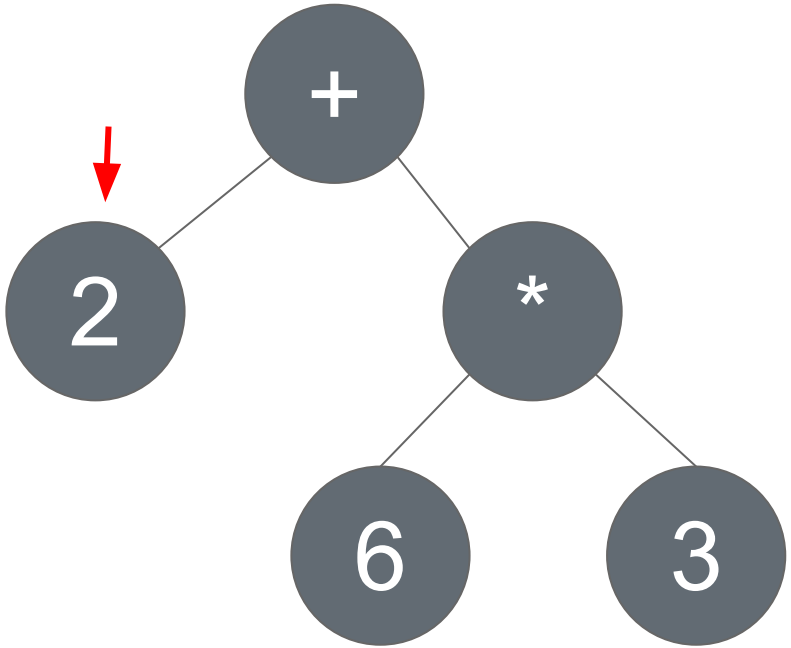
```
return evaluate(node->left) +
evaluate(node->right);
```



Evaluate the AST

```
return node->value;
```

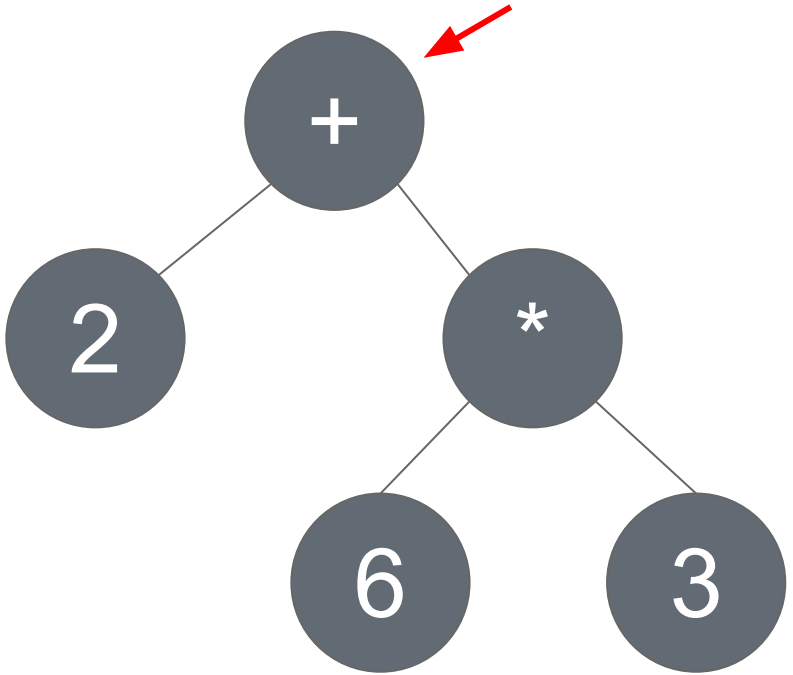
2



Evaluate the AST

```
return evaluate(node->left) +
evaluate(node->right);
```

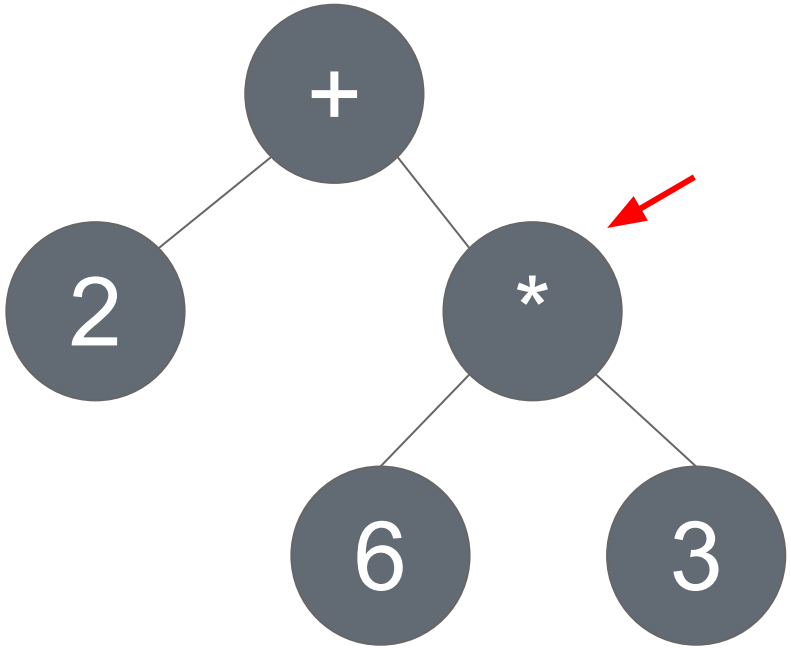
2



Evaluate the AST

```
return evaluate(node->left) *
evaluate(node->right);
```

2

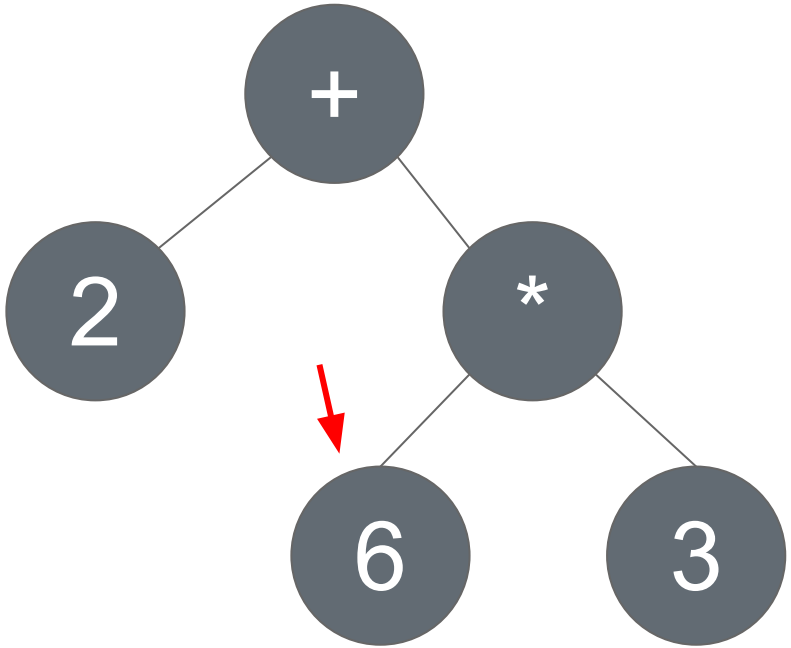


Evaluate the AST

```
return node->value;
```

2

6

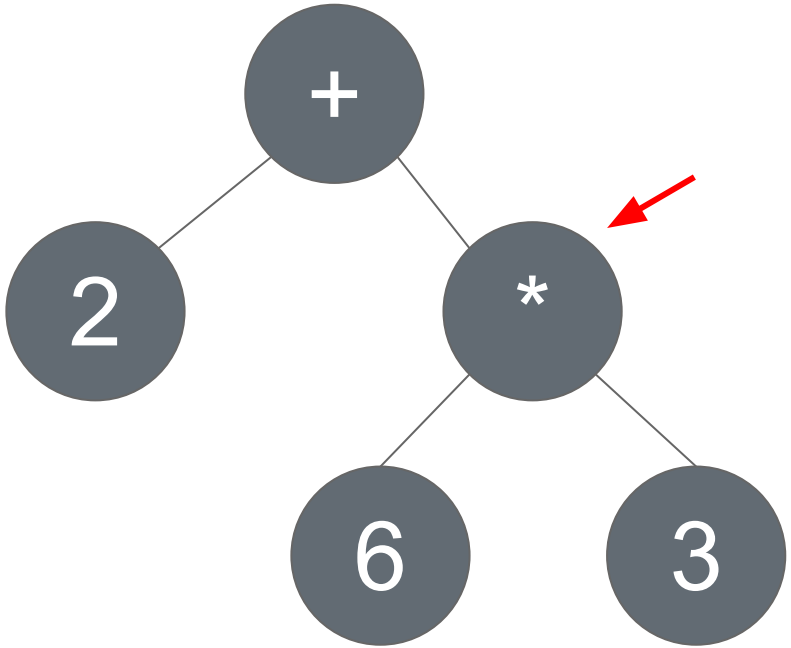


Evaluate the AST

```
return evaluate(node->left) *
evaluate(node->right);
```

2

6

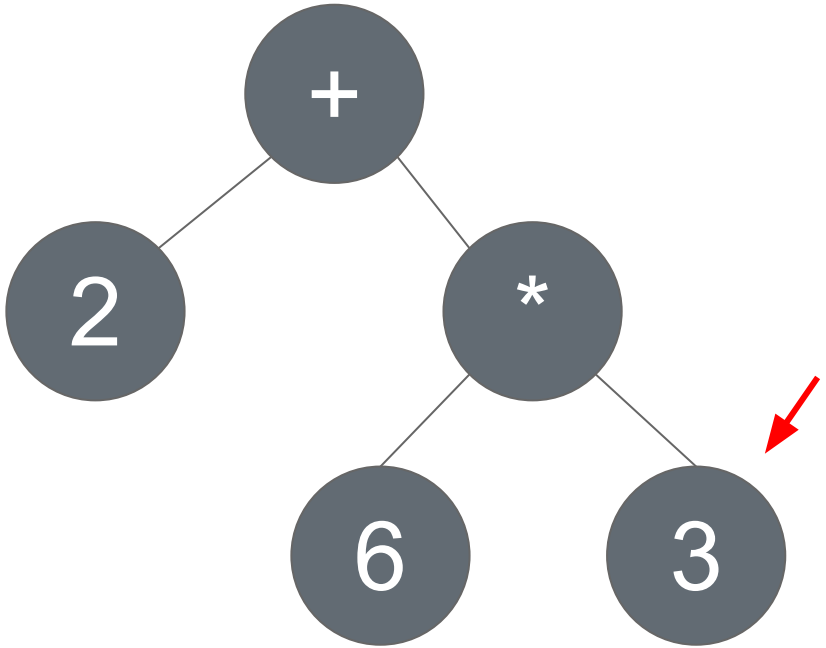


Evaluate the AST

```
return node->value;
```

2

3

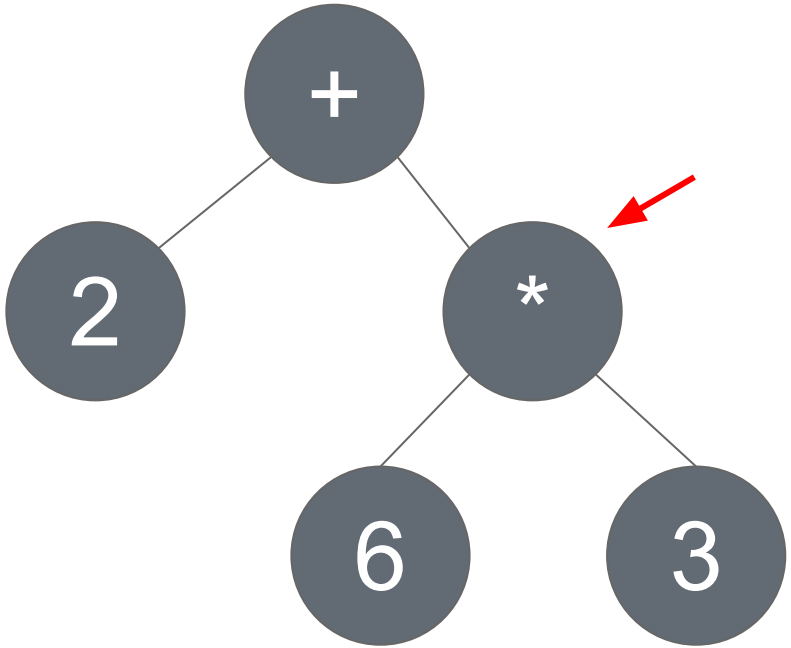


Evaluate the AST

```
return evaluate(node->left) *
evaluate(node->right);
```

2

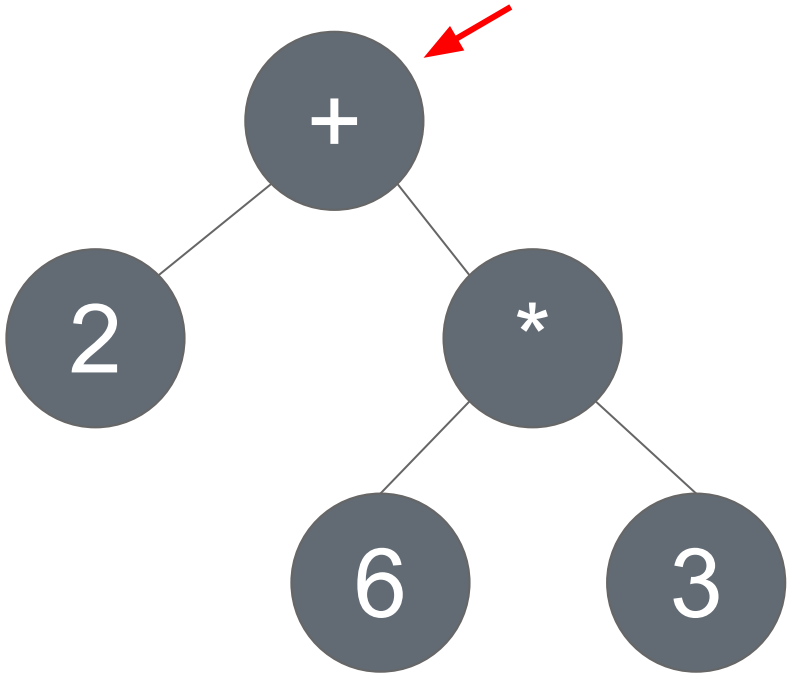
6 * 3



Evaluate the AST

```
return evaluate(node->left) +
evaluate(node->right);
```

2 + 6 * 3



Evaluate the AST – Myfind case

```
int evaluate(struct node* ast)
{
    if (node->type == OR)
    {
        return evaluate(node->left) || evaluate(node->right);
    } else if (node->type == AND)
    {
        return evaluate(node->left) && evaluate(node->right);
    } else
    {
        for (int i = 0; i < funs_length; i++)
            if (funs[i].type == node->type)
                return funs[i].fun(node->value);
    }
}
```

Evaluate the AST - Myfind case

```
struct function {
    enum node_type type;
    int (*fun)(char *);
}

struct function funs[];
```

```
for (int i = 0; i < funs_length; i++)
    if (funs[i].type == node->type)
        return funs[i].fun();
```

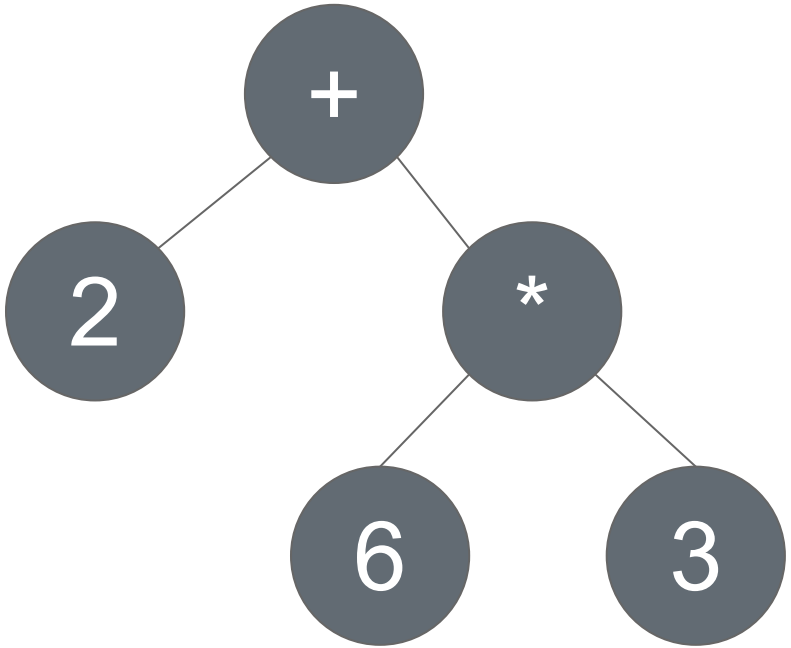
Build the AST

Build the AST

FROM

```
char *input = “2 + 6 * 3”;
```

TO



Build the AST - From string to tokens

```
enum node_type
{
    ADD,
    MULT,
    NUMBER
}
```

```
struct token {
    enum node_type type;
    int value;
}
```

Build the AST - From string to tokens

FROM

```
char *input = “2 + 6 * 3”;
```

TO

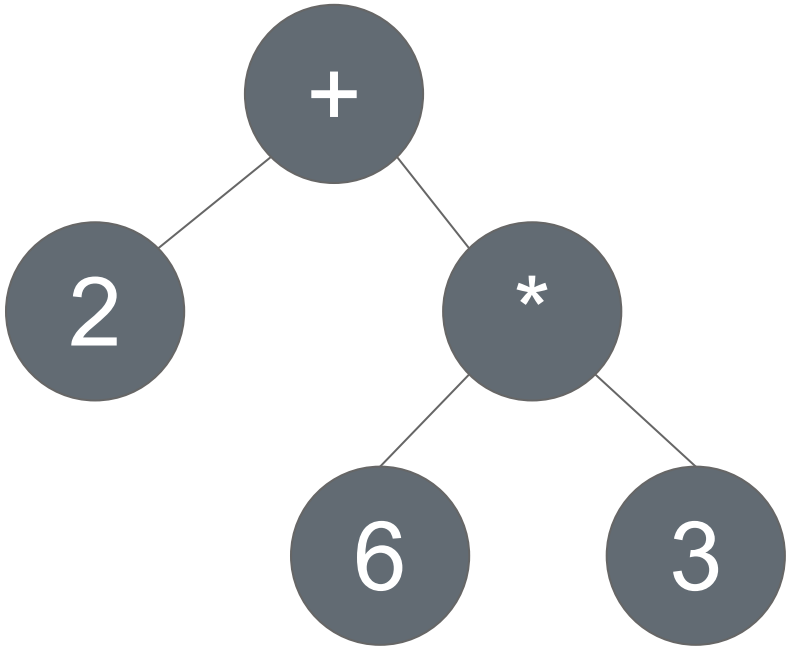
```
struct token tokens[] = {
    {NUMBER, 2},
    {ADD, 0},
    {NUMBER, 6},
    {MULT, 0},
    {NUMBER, 3},
}
```

Build the AST - From tokens to nodes

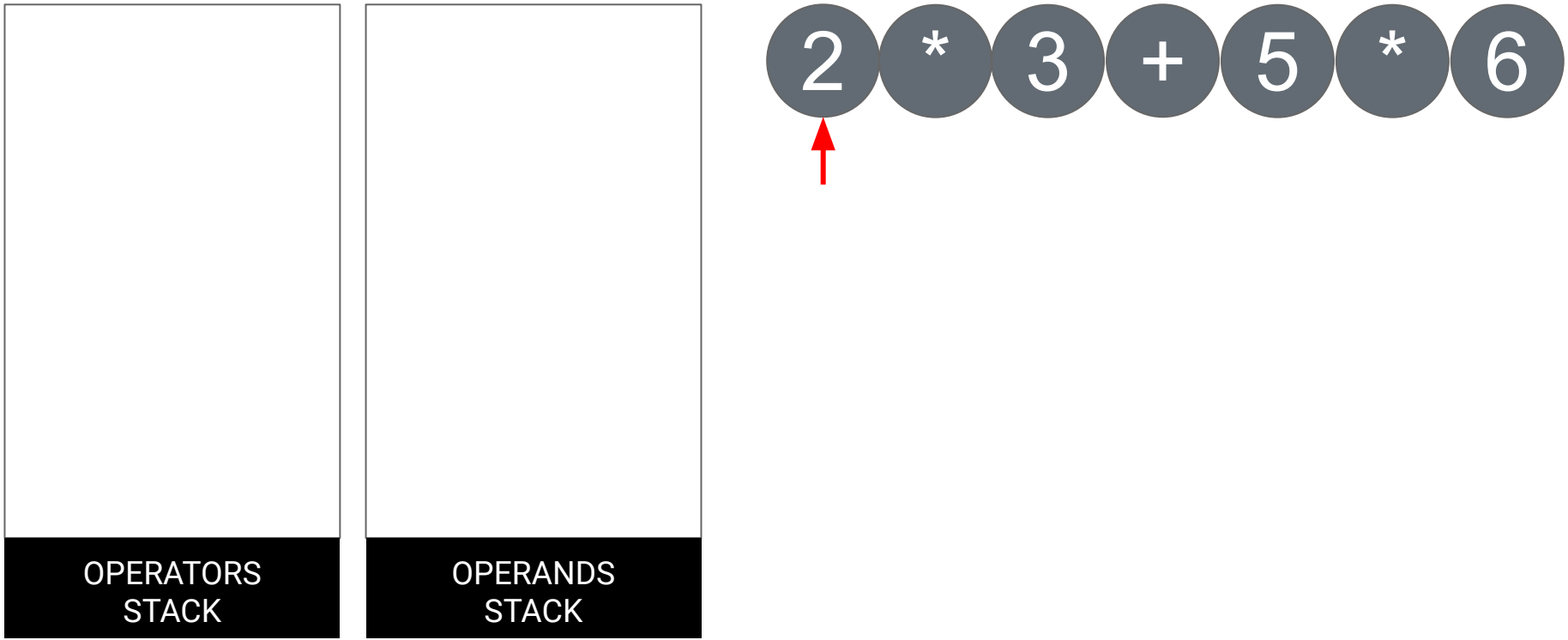
FROM

```
struct token tokens[] = {
    {NUMBER, 2},
    {ADD, 0},
    {NUMBER, 6},
    {MULT, 0},
    {NUMBER, 3},
}
```

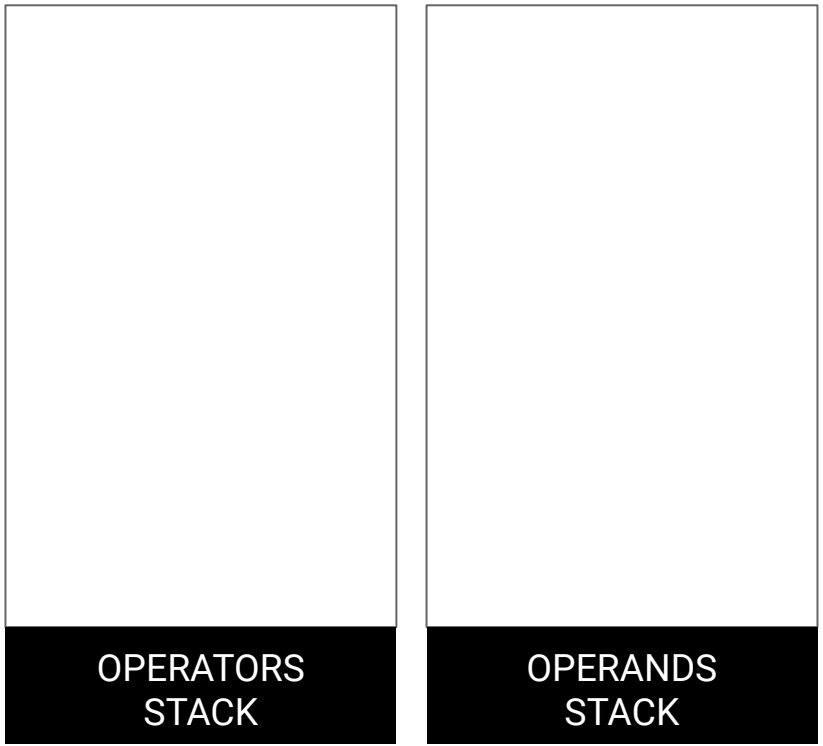
TO

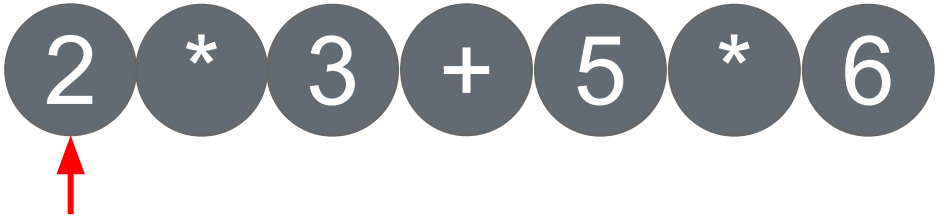


Build the AST – From tokens to nodes



Build the AST – From tokens to nodes

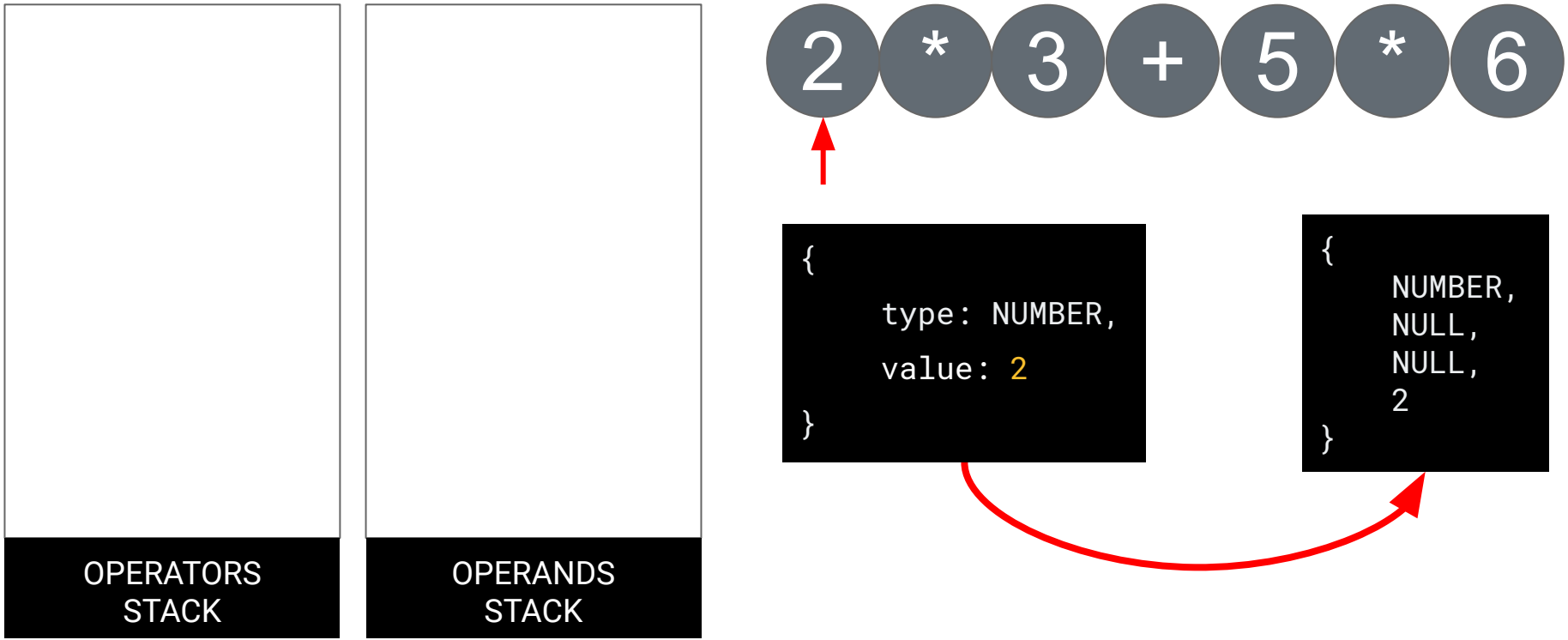




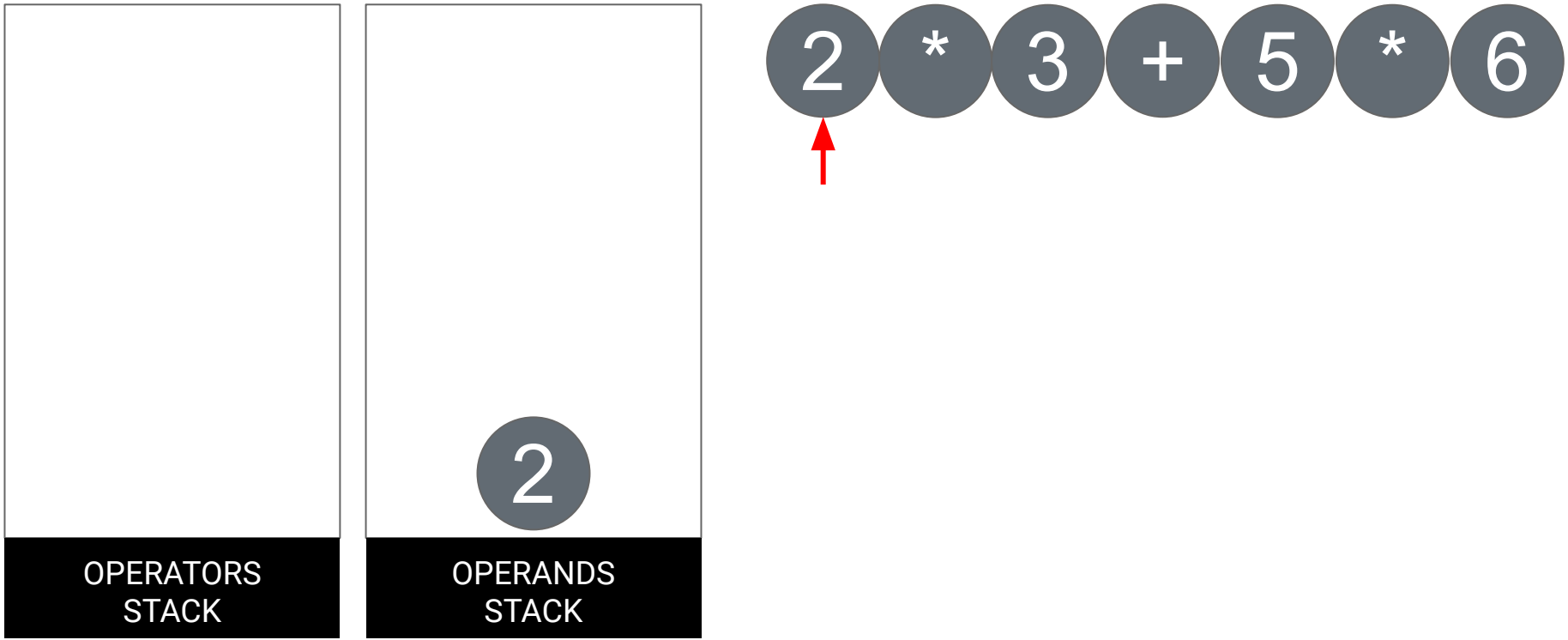

```
struct token {
    enum node_type type;
    int value;
}
```

```
struct node
{
    enum node_type type;
    struct node *left;
    struct node *right;
    int value;
};
```

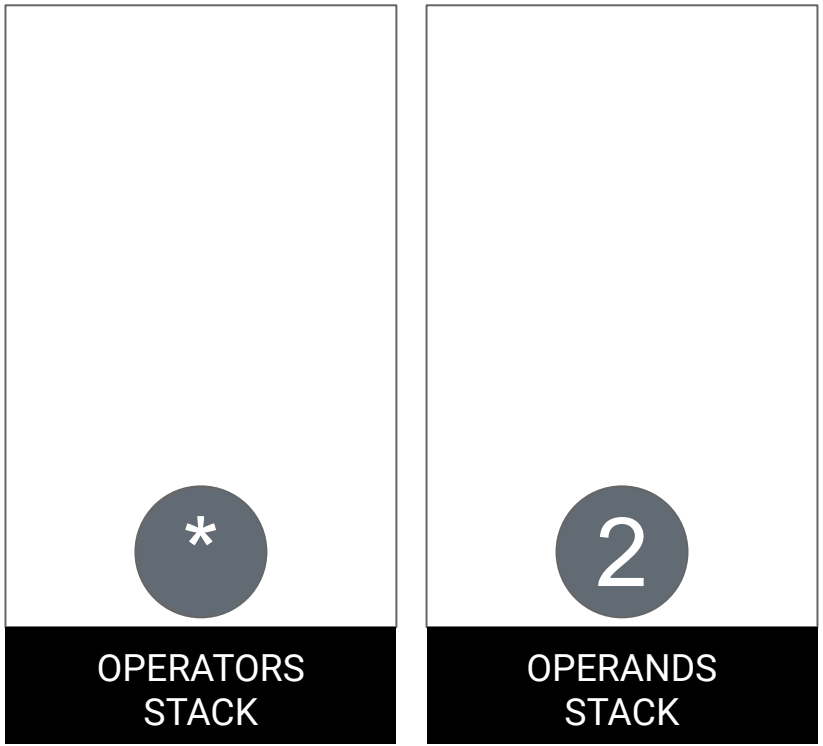
Build the AST – From tokens to nodes

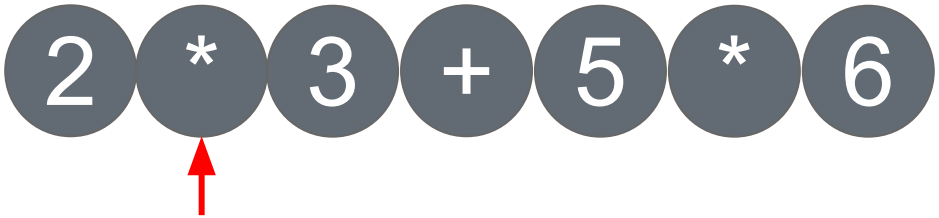


Build the AST – From tokens to nodes

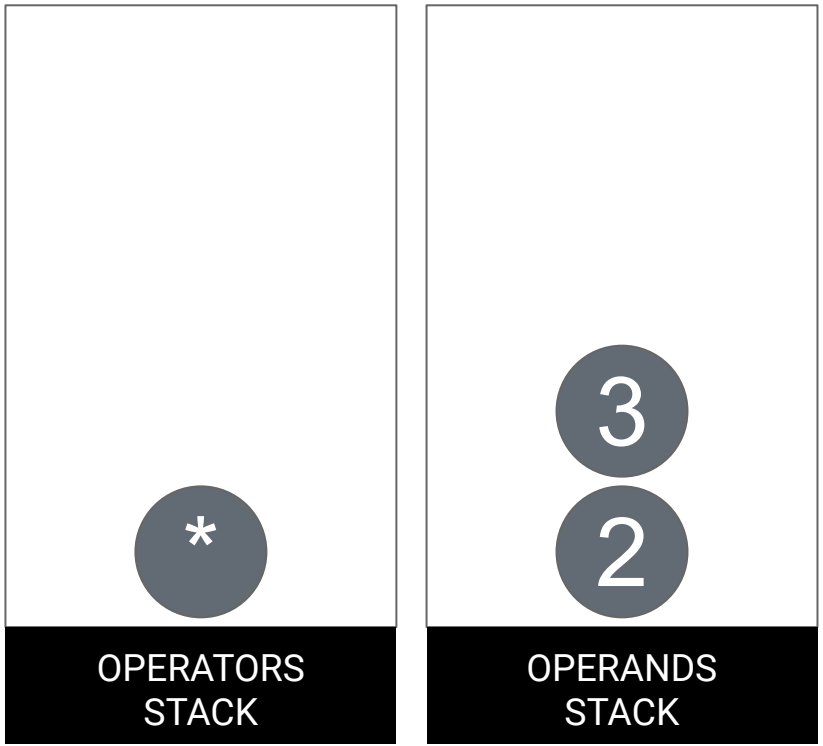


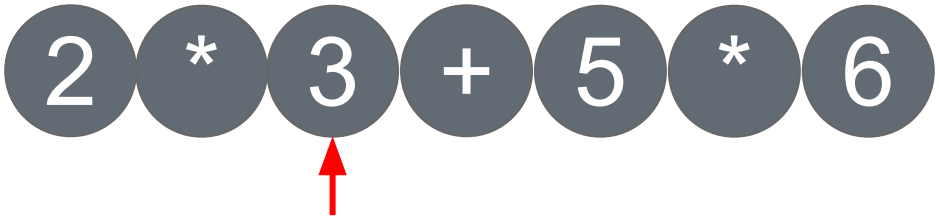
Build the AST – From tokens to nodes



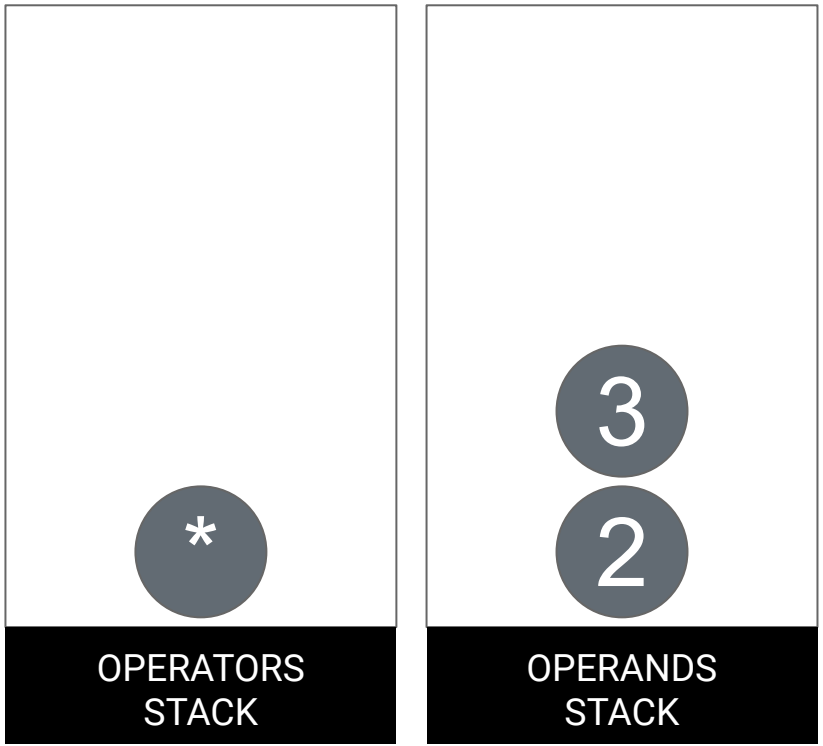


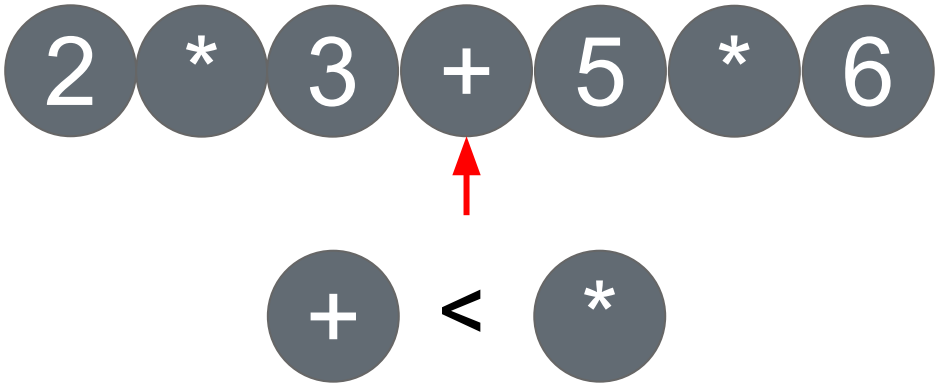
Build the AST – From tokens to nodes



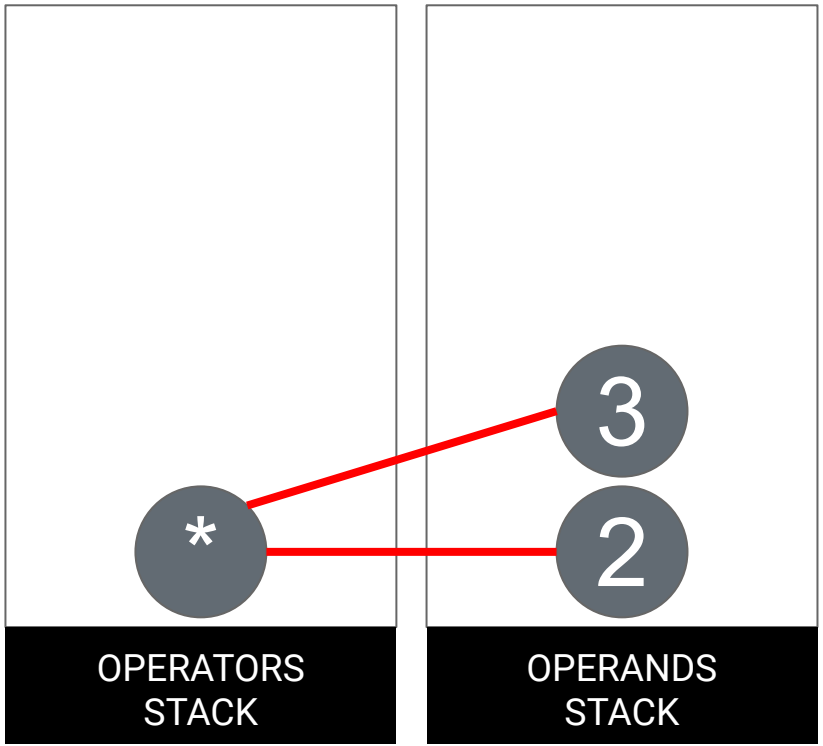


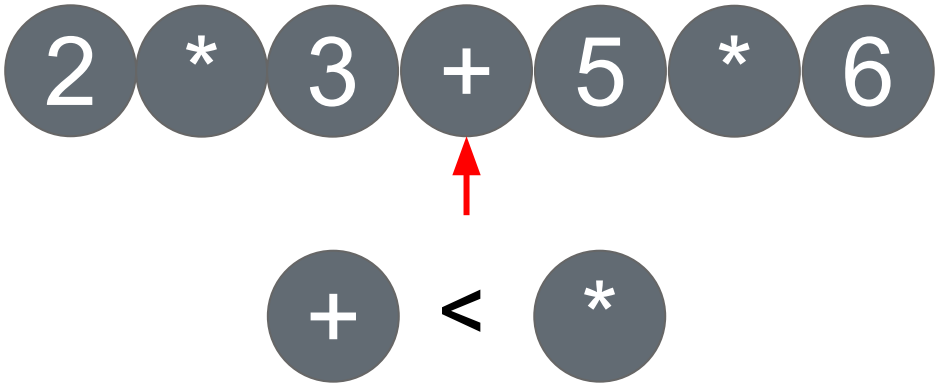
Build the AST - From tokens to nodes



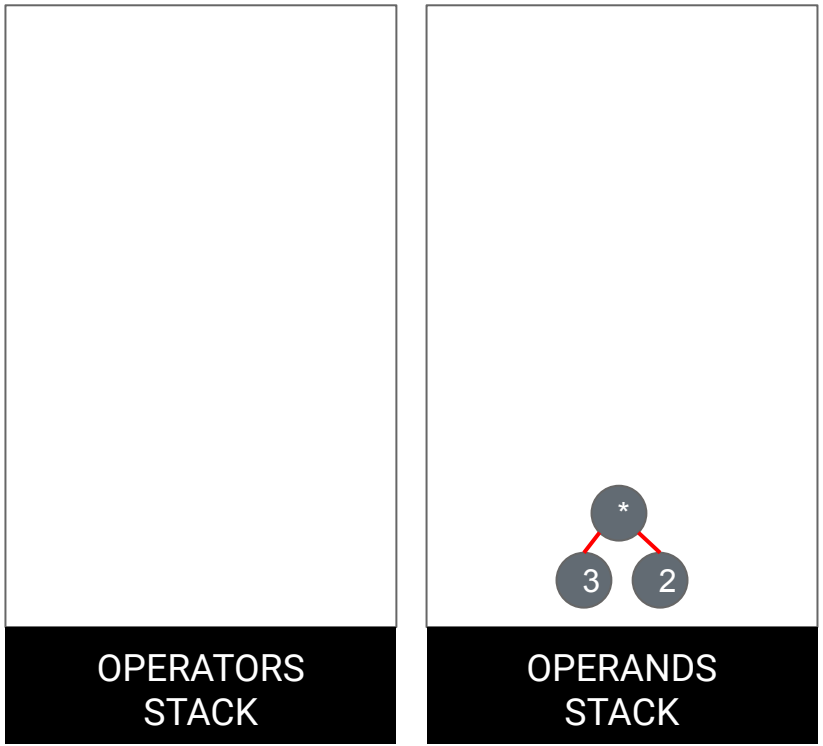


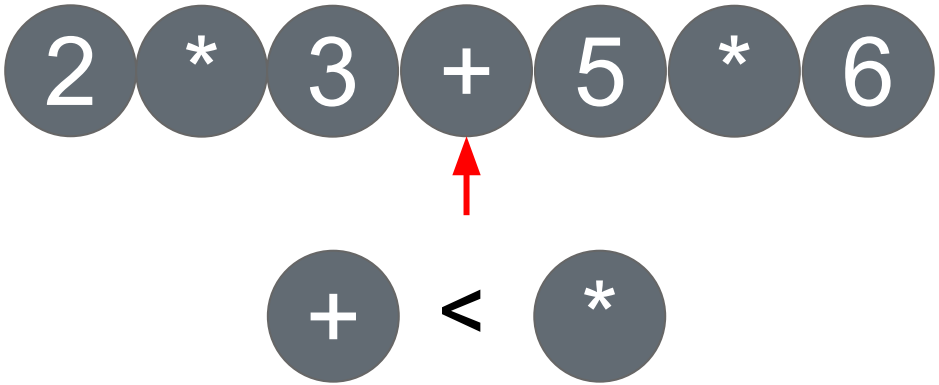
Build the AST - From tokens to nodes



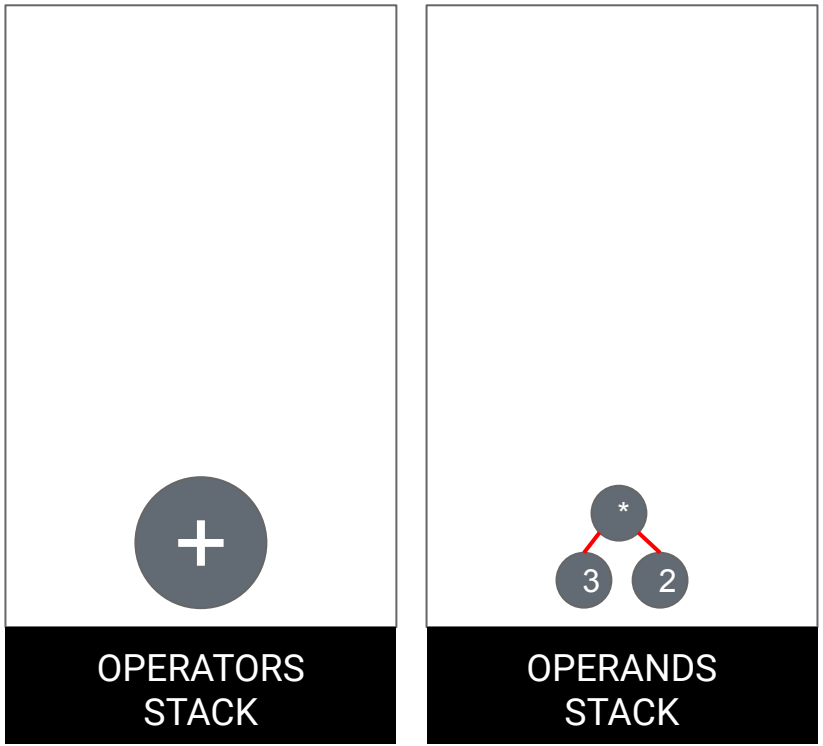


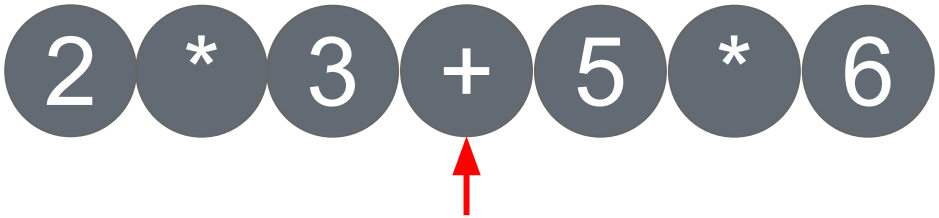
Build the AST - From tokens to nodes



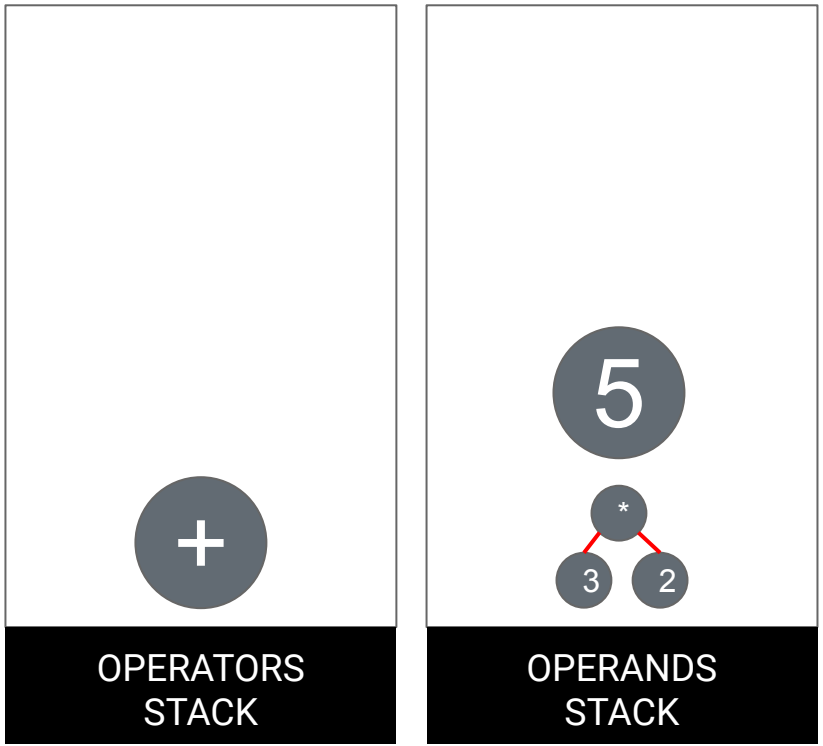


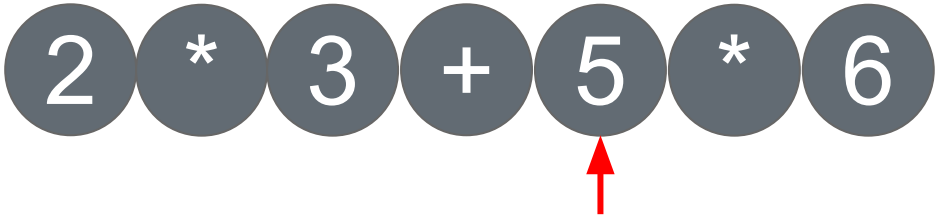
Build the AST – From tokens to nodes



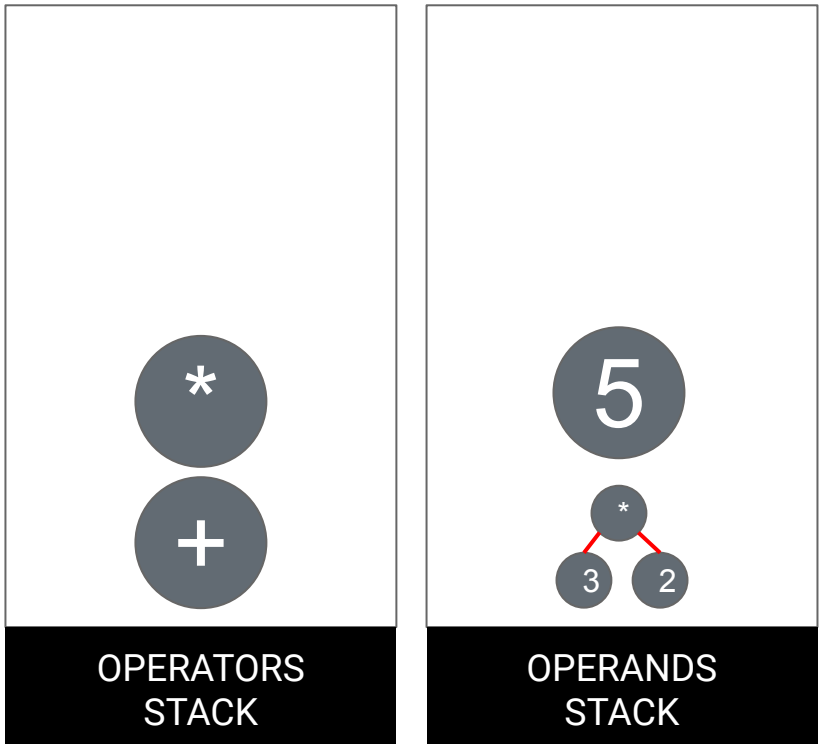


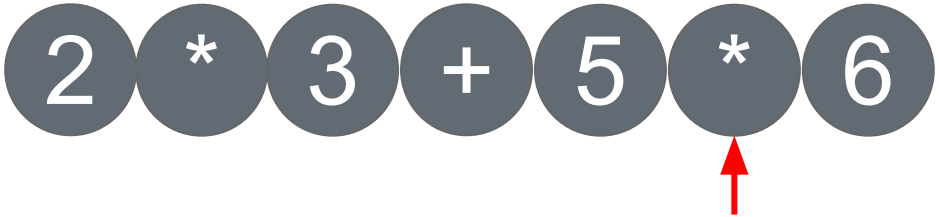
Build the AST - From tokens to nodes



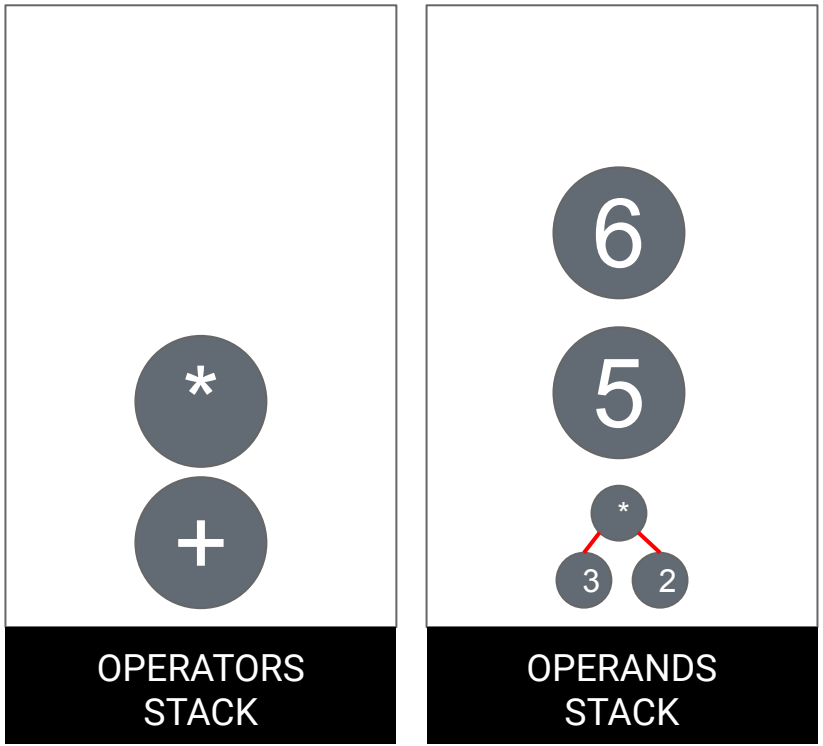


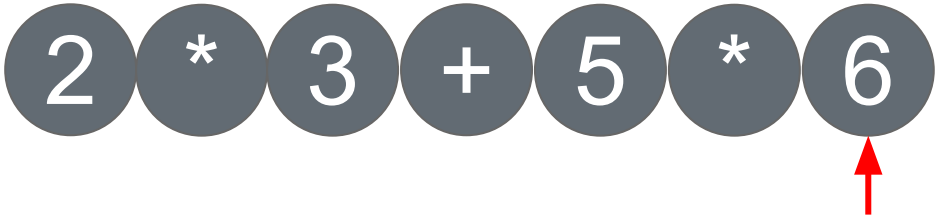
Build the AST – From tokens to nodes



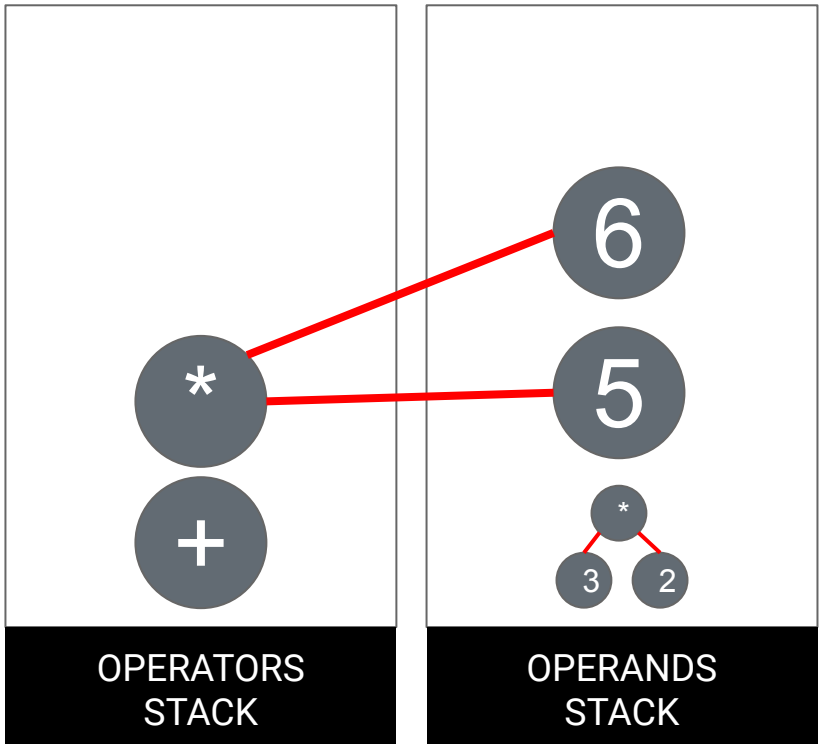


Build the AST – From tokens to nodes

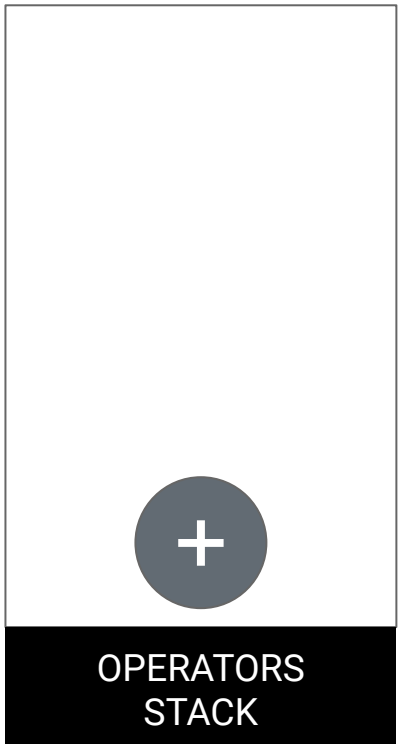


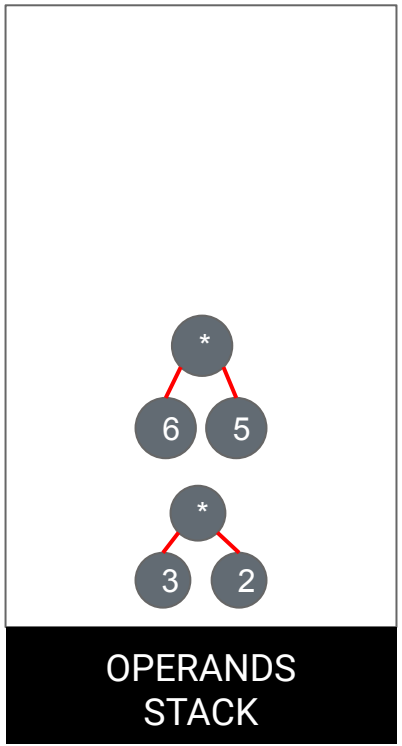


Build the AST – From tokens to nodes

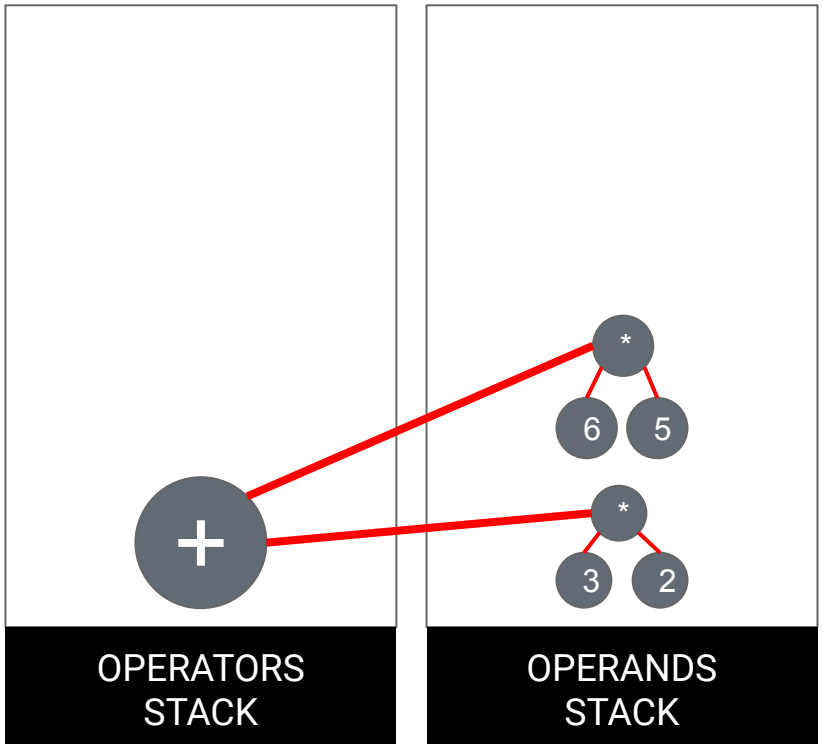


Build the AST – From tokens to nodes





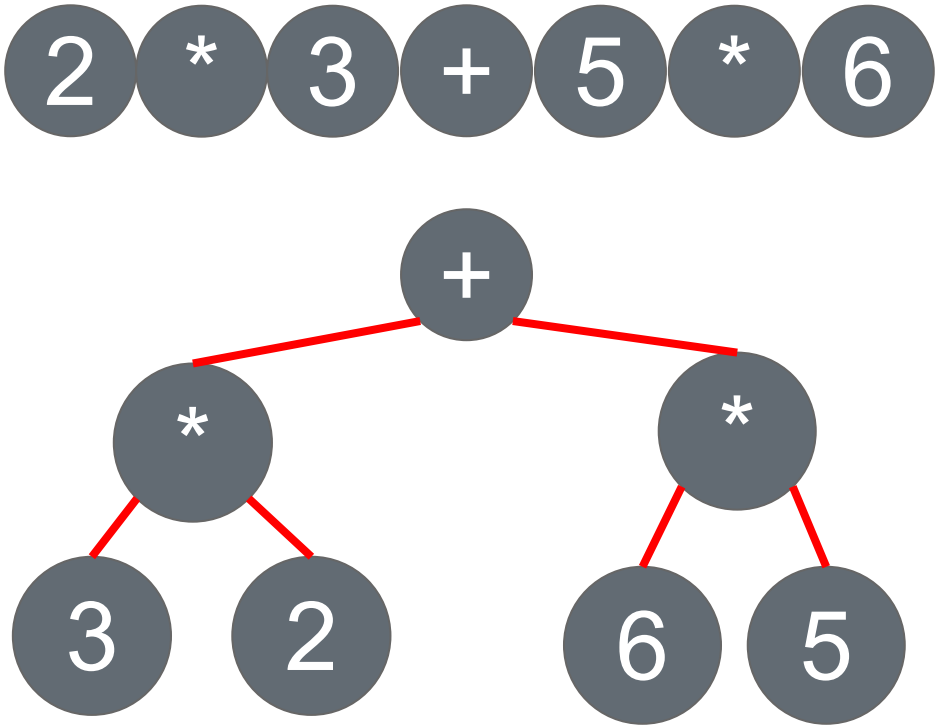
Build the AST - From tokens to nodes



Build the AST – From tokens to nodes

OPERATORS STACK

OPERANDS STACK

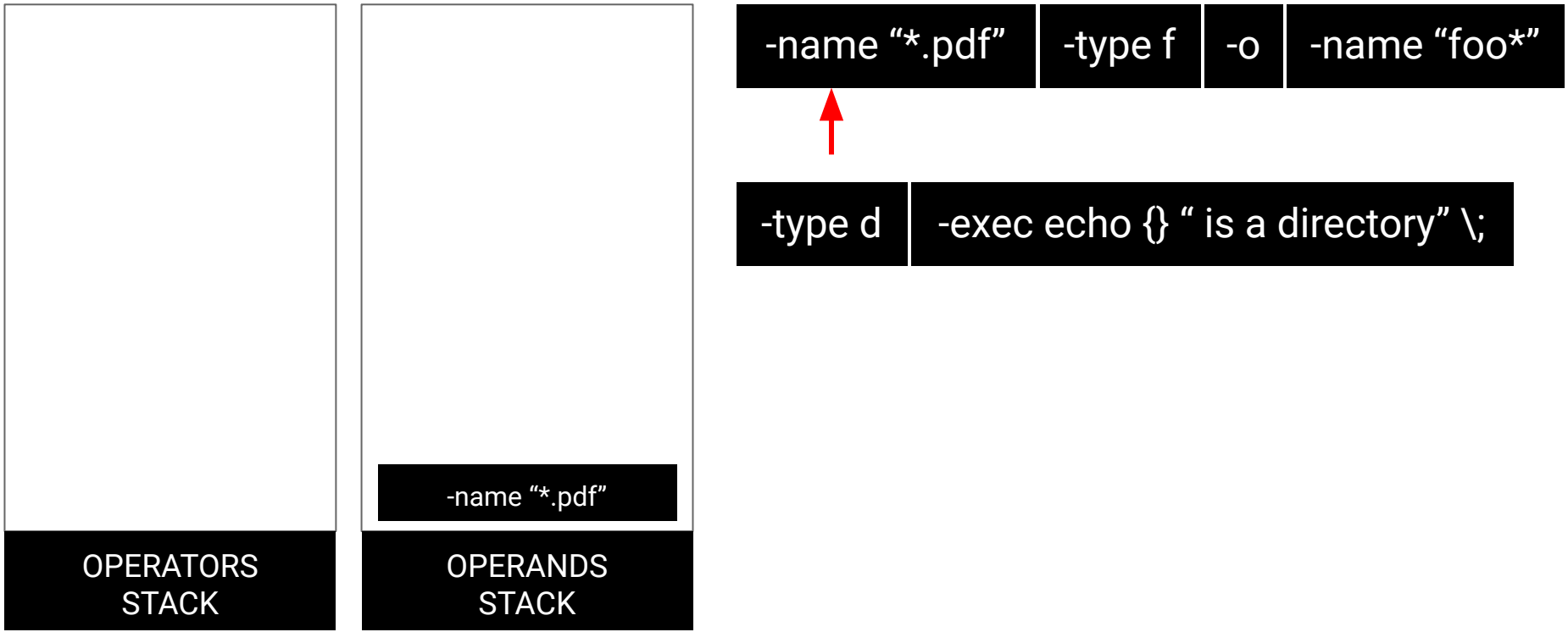


Build the AST – Myfind case

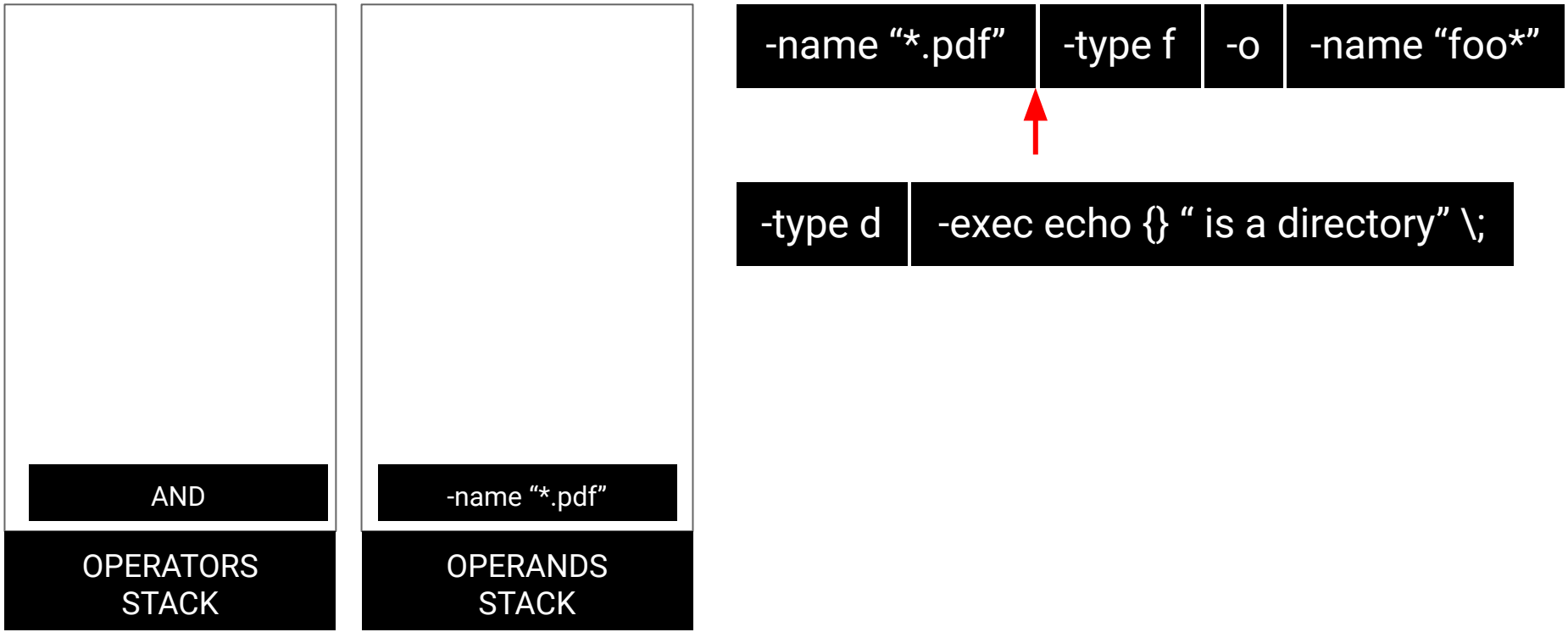
OPERATORS
OR AND NOT PARENTHESES

OPERANDS
NAME TYPE NEWER DELETE EXEC EXECDIR ...

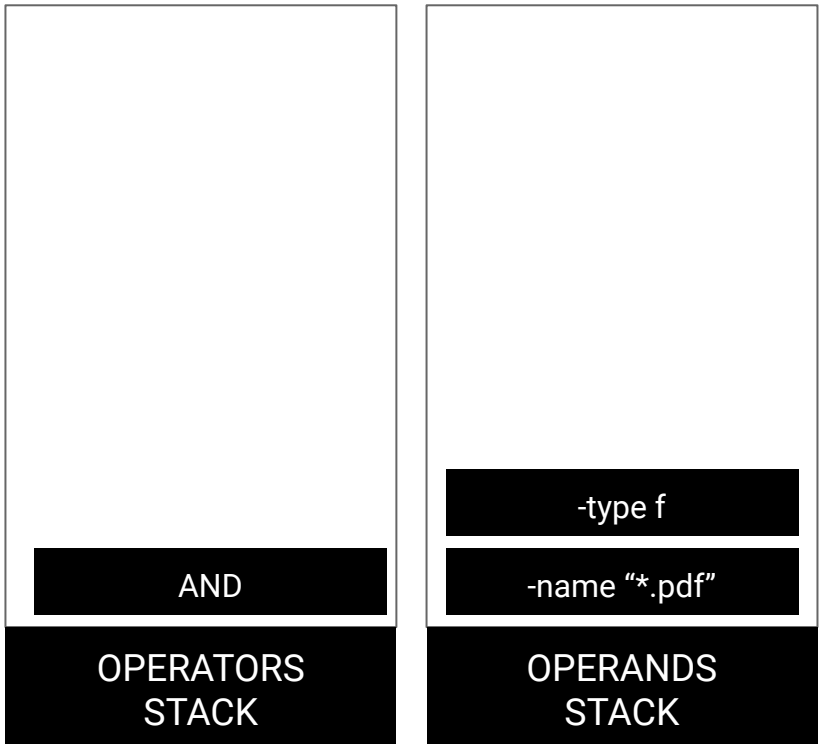
Build the AST – Myfind case

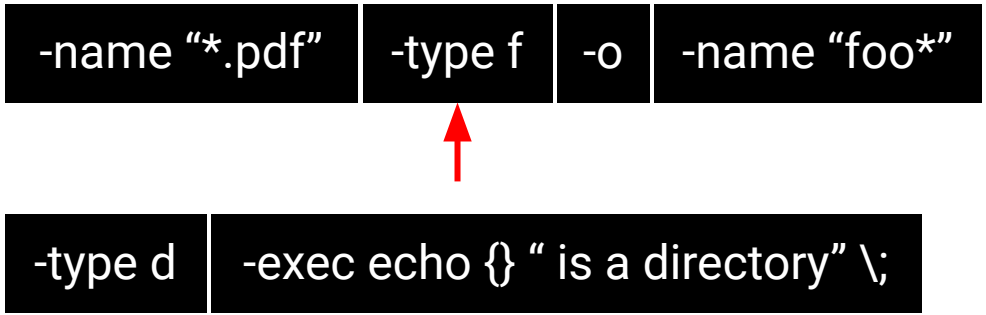


Build the AST – Myfind case

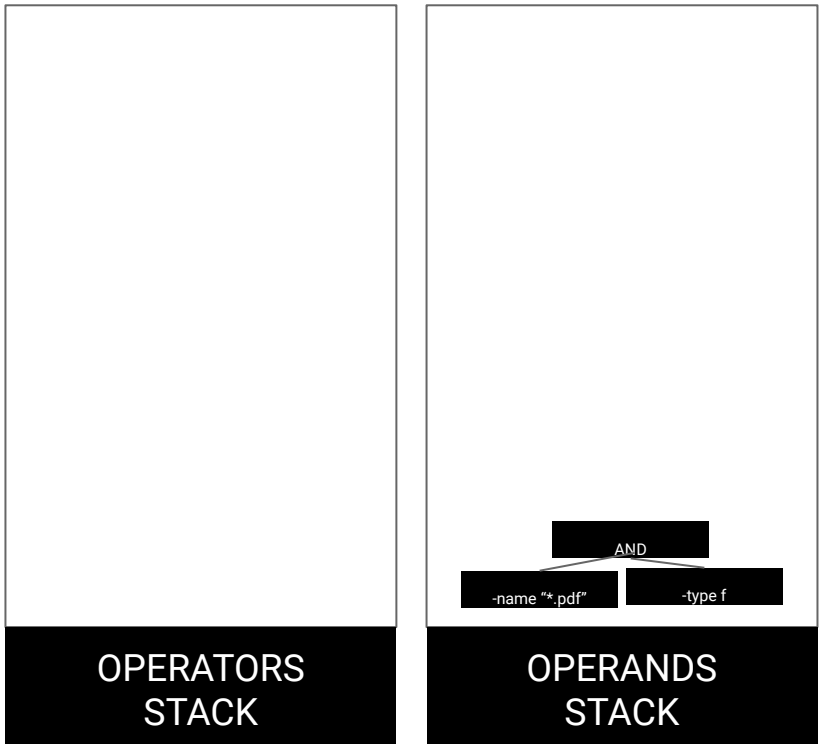


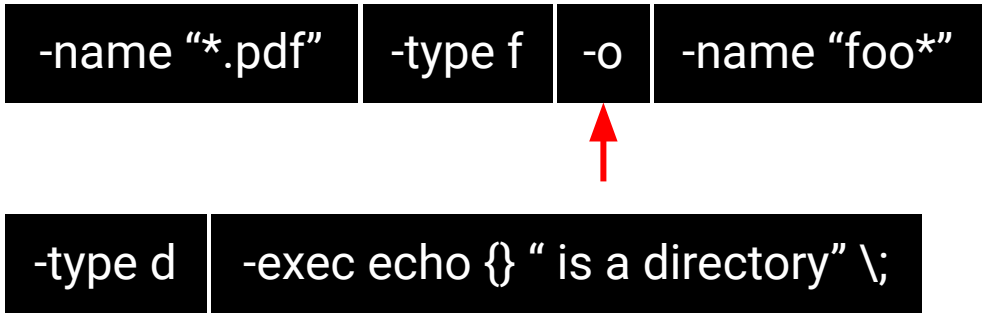
Build the AST – Myfind case



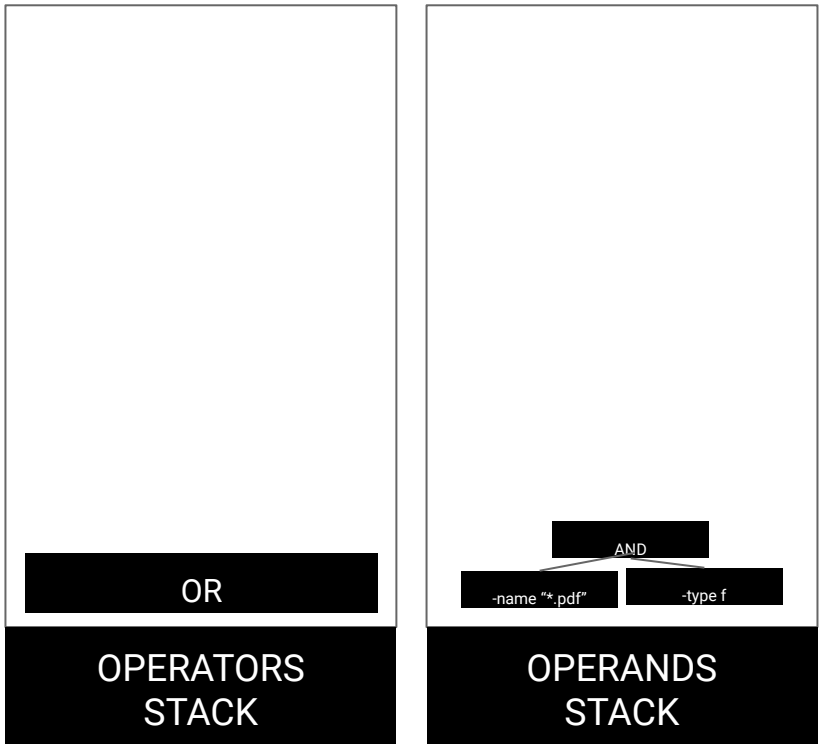


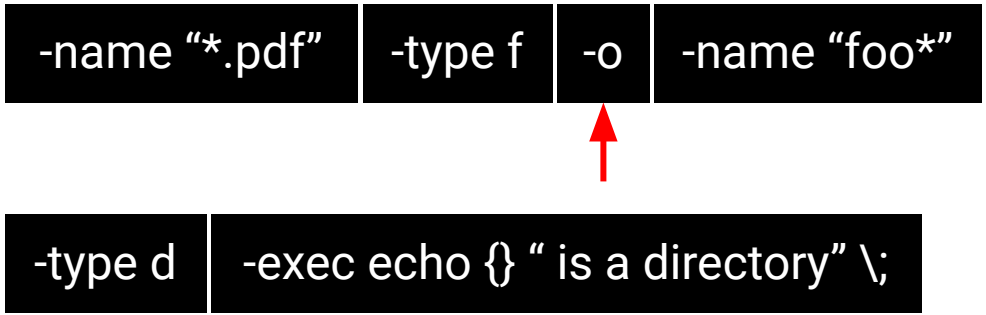
Build the AST – Myfind case



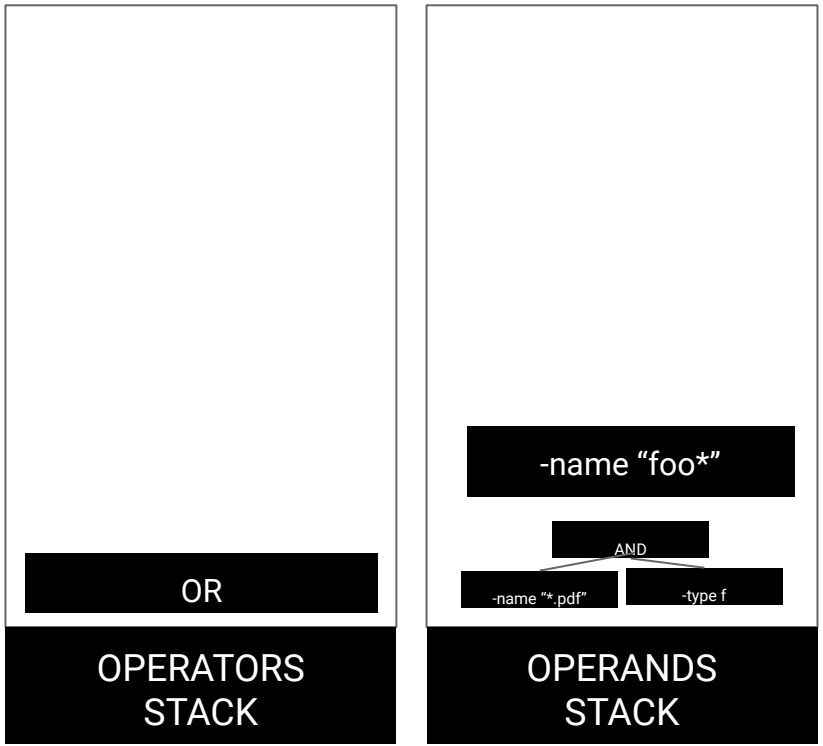


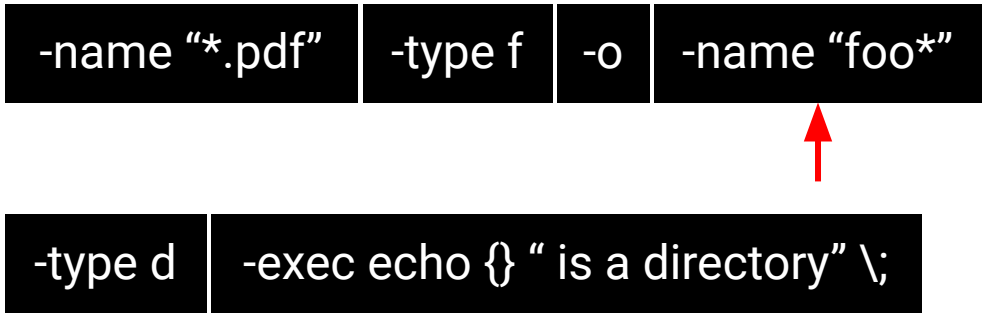
Build the AST – Myfind case



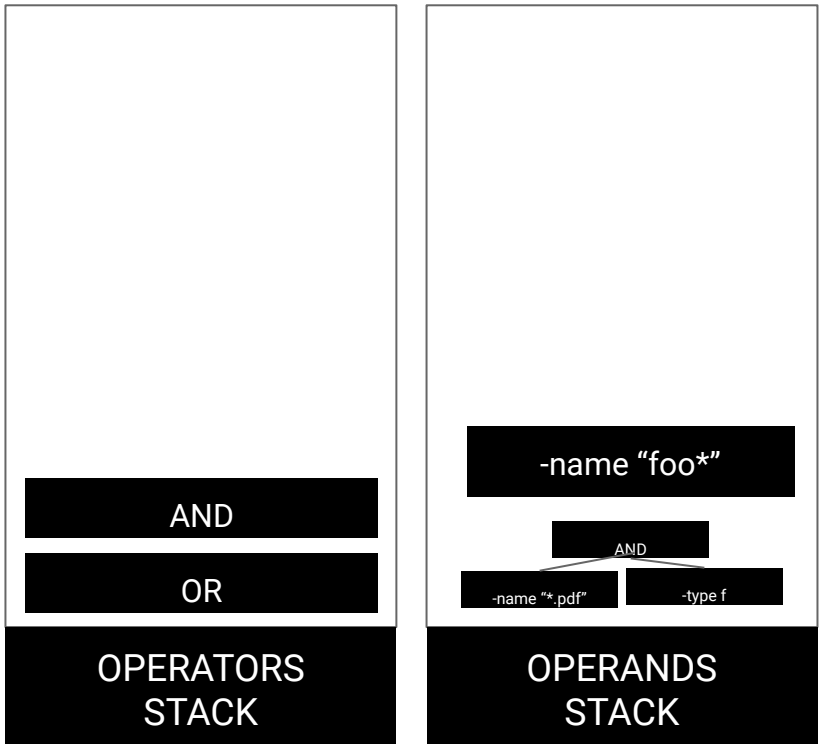


Build the AST – Myfind case





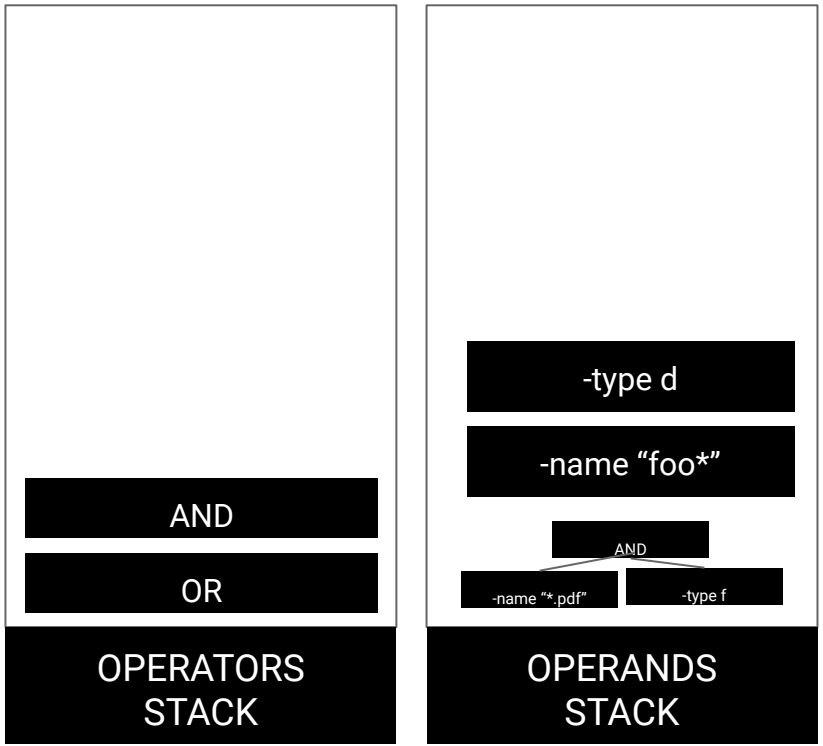
Build the AST – Myfind case



-name “*.pdf” | -type f | -o | -name “foo*”

-type d | -exec echo {} “ is a directory” \;

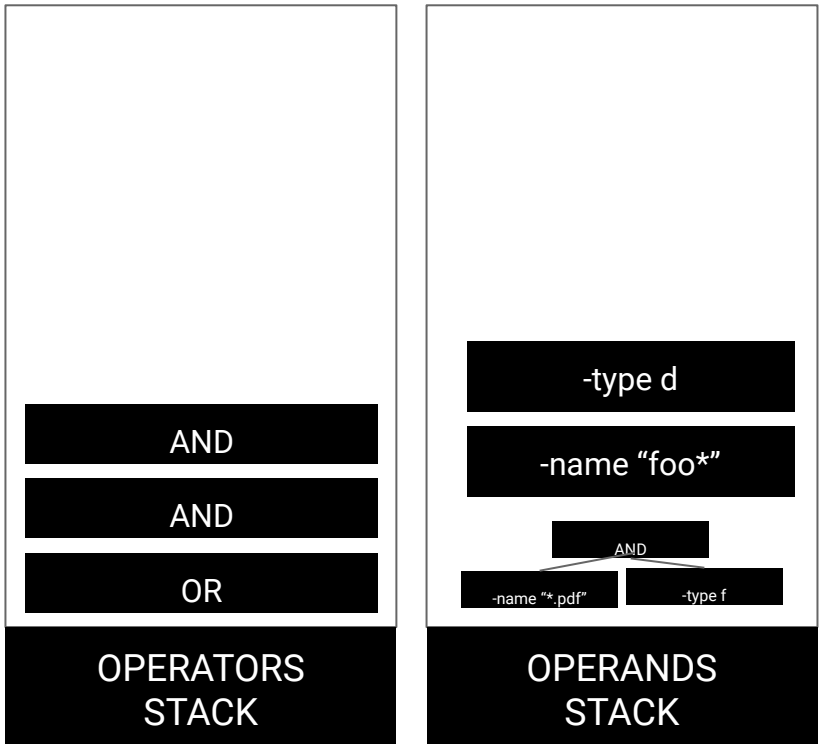
Build the AST – Myfind case

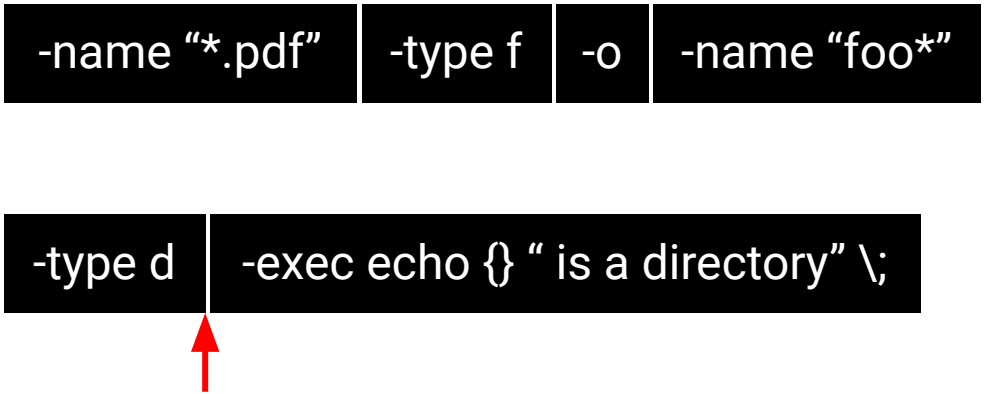


-name “*.pdf” | -type f | -o | -name “foo*”

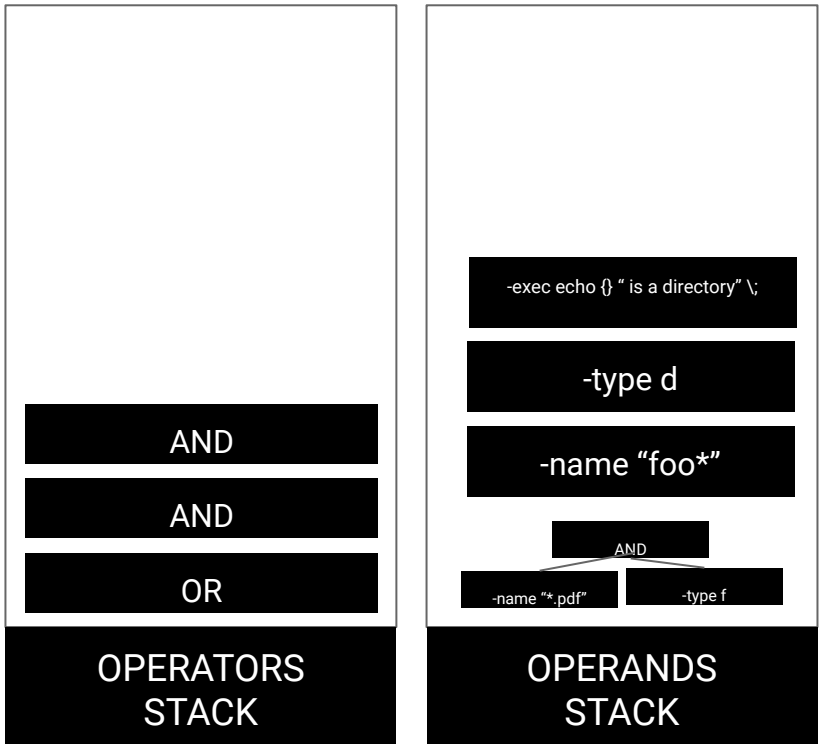
-type d | -exec echo {} “ is a directory” \;

Build the AST – Myfind case





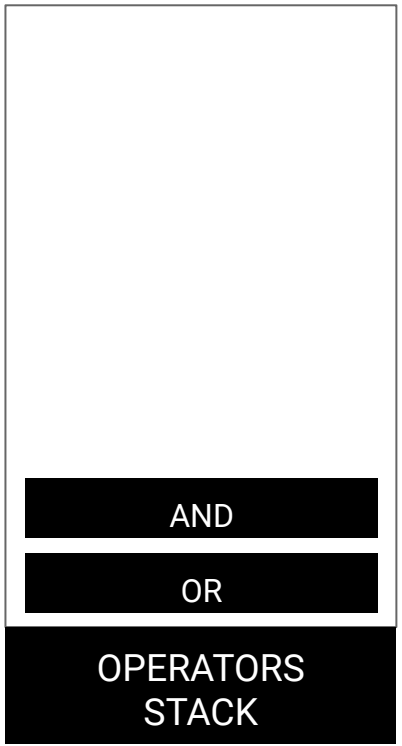
Build the AST – Myfind case

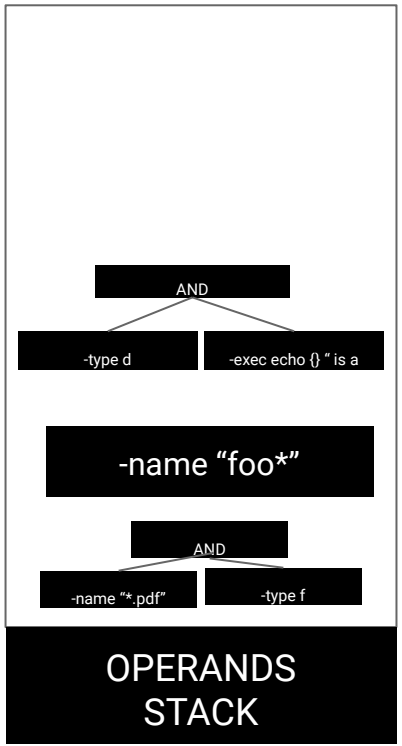


-name “*.pdf” | -type f | -o | -name “foo*”

-type d | -exec echo {} “ is a directory” \;

Build the AST – Myfind case

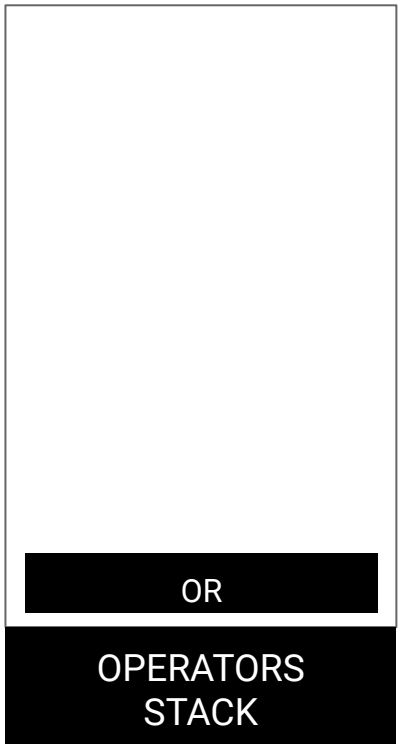


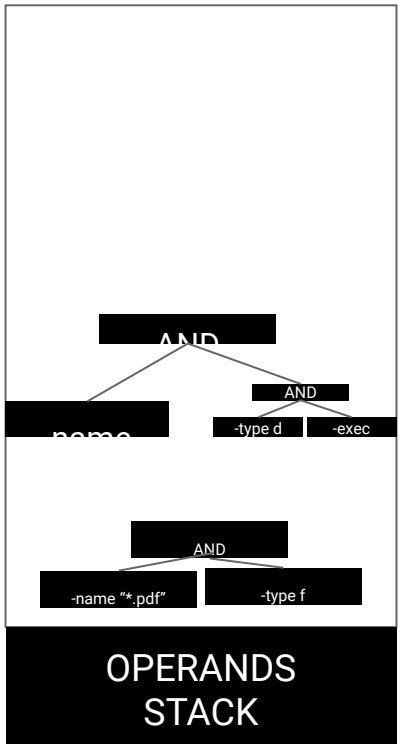


-name “*.pdf”	-type f	-o	-name “foo*”

-type d	-exec echo {} “ is a directory” \;

Build the AST – Myfind case

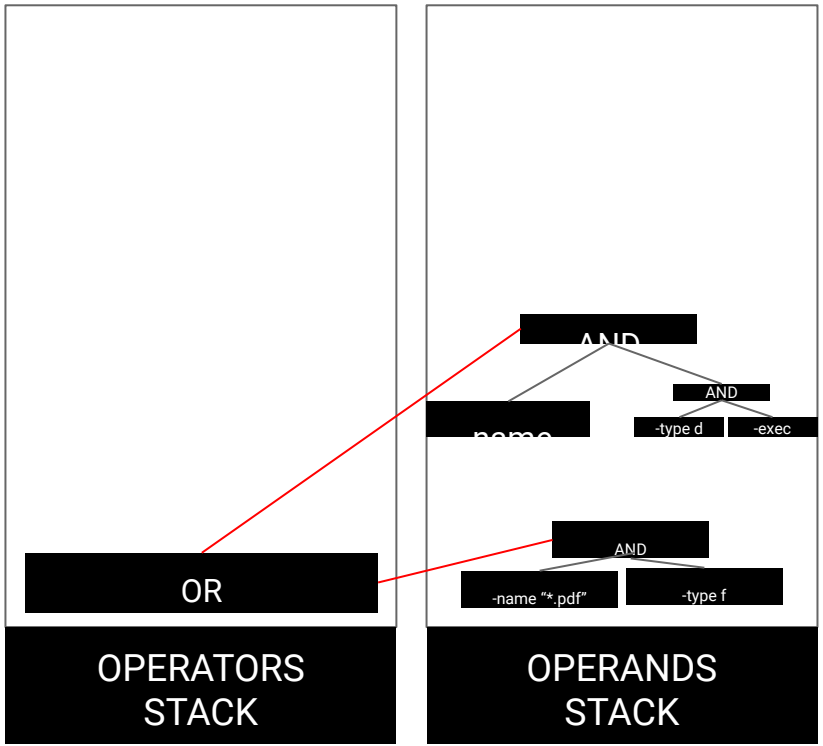




-name “*.pdf”	-type f	-o	-name “foo*”

-type d	-exec echo {} “ is a directory” \;

Build the AST – Myfind case



-name “*.pdf”	-type f	-o	-name “foo*”

-type d	-exec echo {} “ is a directory” \;

Build the AST – Myfind case