PostScript

Steven R. Bagley

POSTSCRIPT

- Page Description Language…
- but also a full interpretative programming language
- Very similar to Forth
- Designed by Adobe Systems Inc, first released in the Apple Laserwriter (1985)

Interpreted Language

- C/C++/Java languages are compiled
  - Source code converted to machine code and then executed
- PostScript interpreted
  - PostScript tokens are executed as they are encountered

Tokens

- PostScript language is defined in terms of tokens
  - Binary or ASCII representation
- We are only concerned with the ASCII form

e.g. `add 42 sub div moveto arc selectfont`

Token Types

- Simple Types
  - Boolean
  - Numbers
    (real and integer)
  - Names
  - Operators

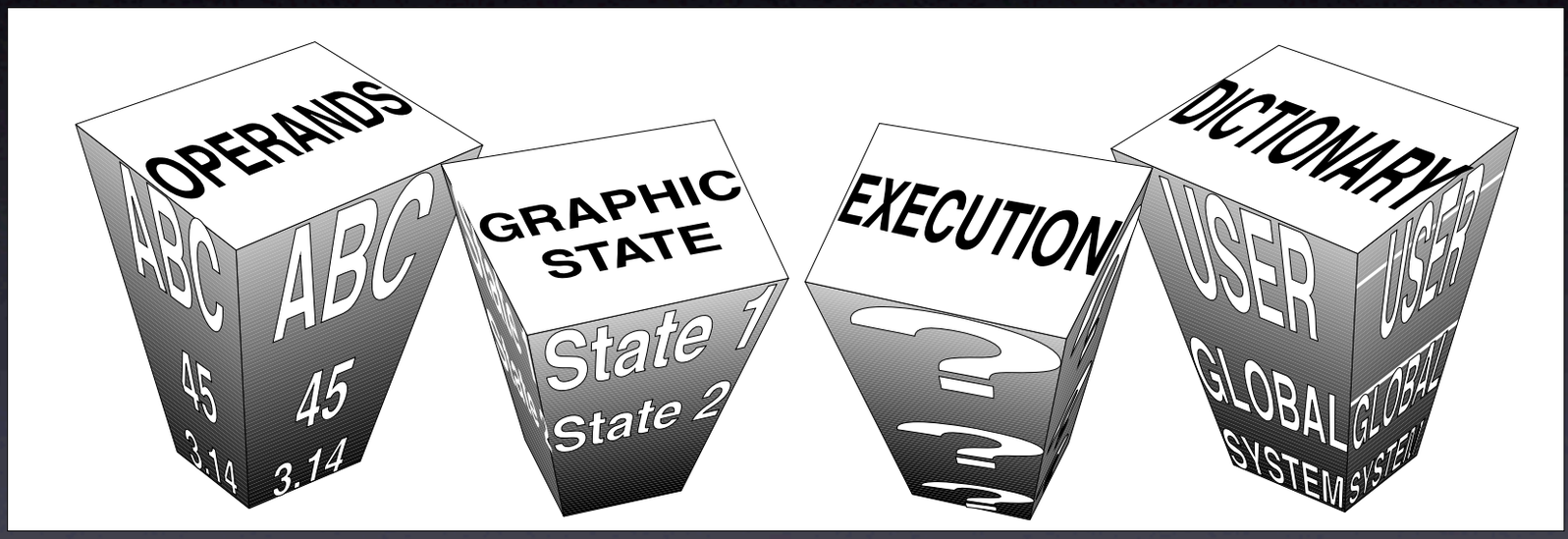
Token types

- Executable
  (e.g. operators — actually names)
- Literal
  object pushed onto the stack…

Stack

- PostScript is based around a stack
- Actually four stacks are used…

Four Stacks

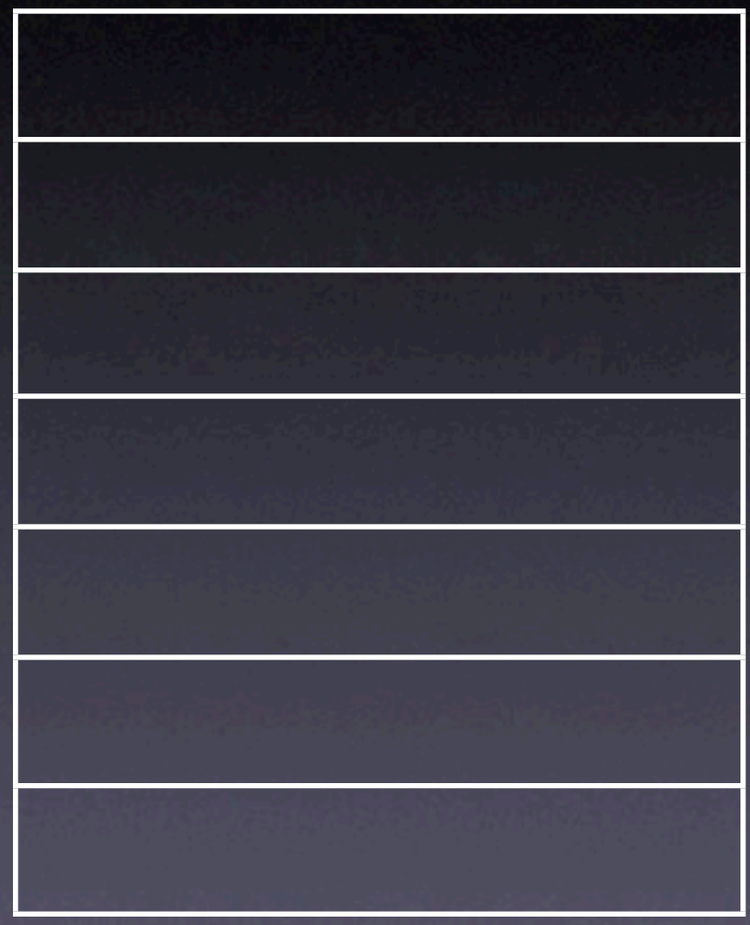

OPERANDS
ABC
45
3.14
ABC
45
3.14
GRAPHIC STATE
State 1
State 2
EXECUTION
DICTIONARY
USER
GLOBAL
SYSTEM
USER
GLOBAL
SYSTEM

Operand Stack

- Literal values are placed onto the operand stack

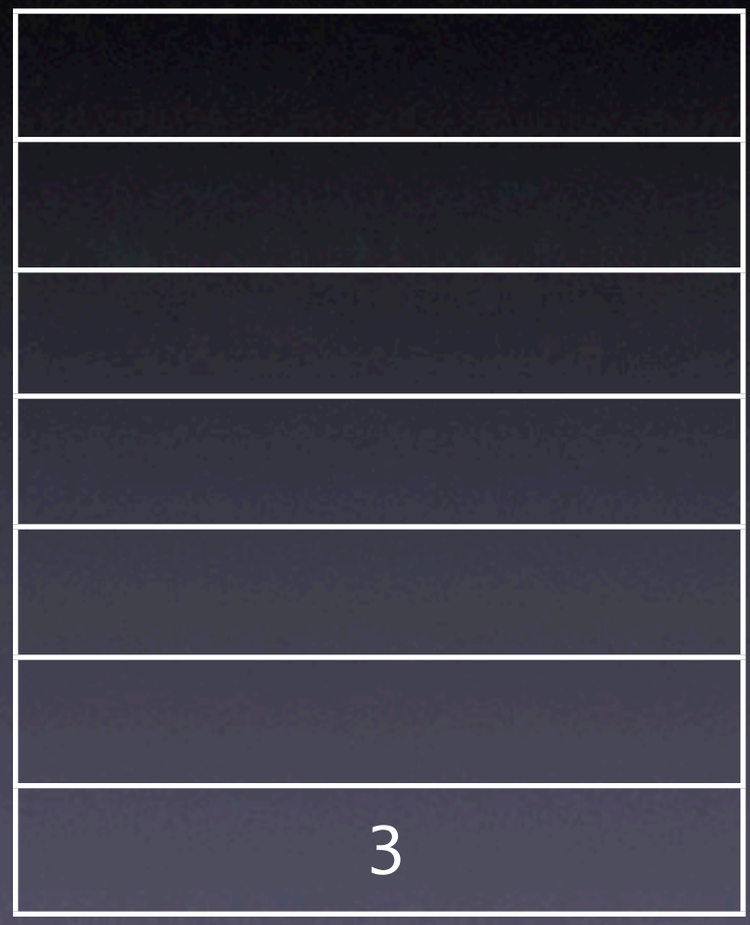
Operand Stack

```
3 4 5 mul add
```



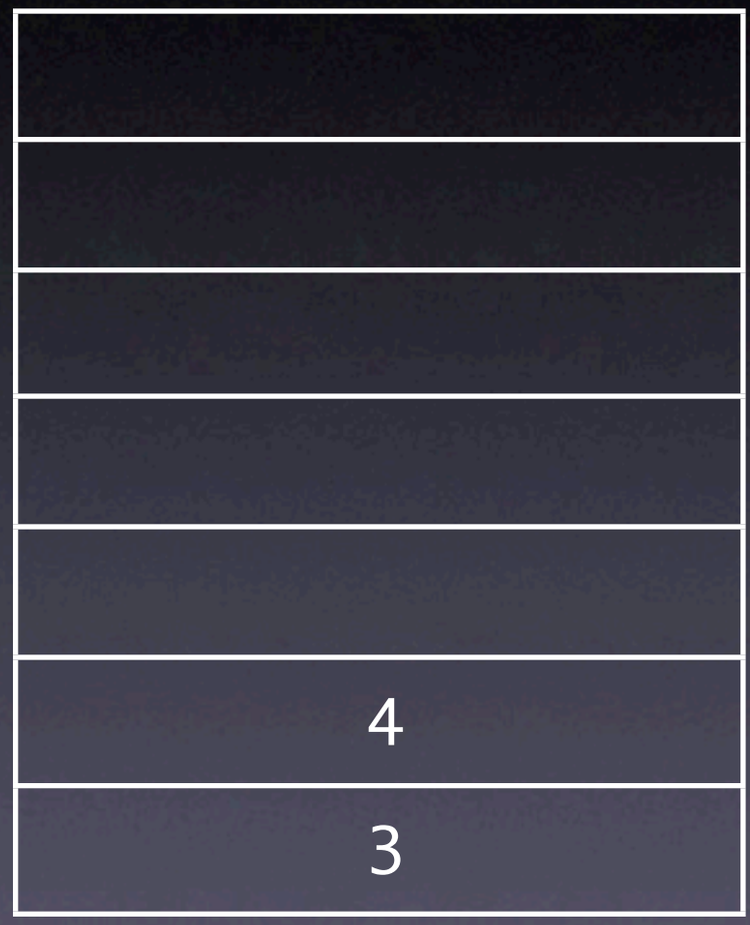
Operand Stack

```
4 5 mul add
```



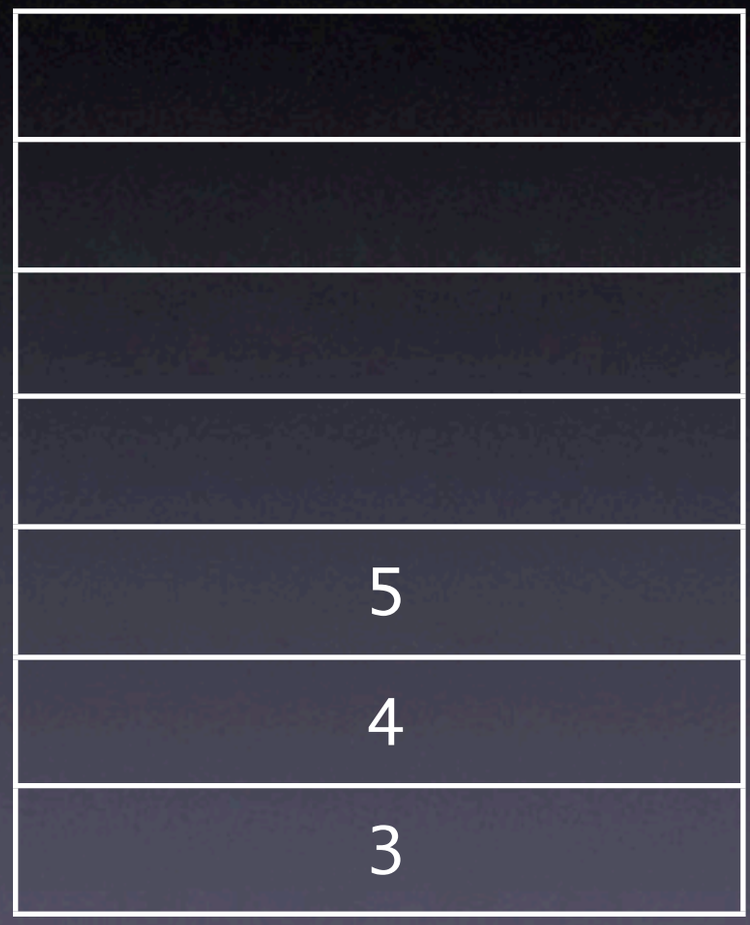
Operand Stack

```
5 mul add
```



Operand Stack

```
mul add
```



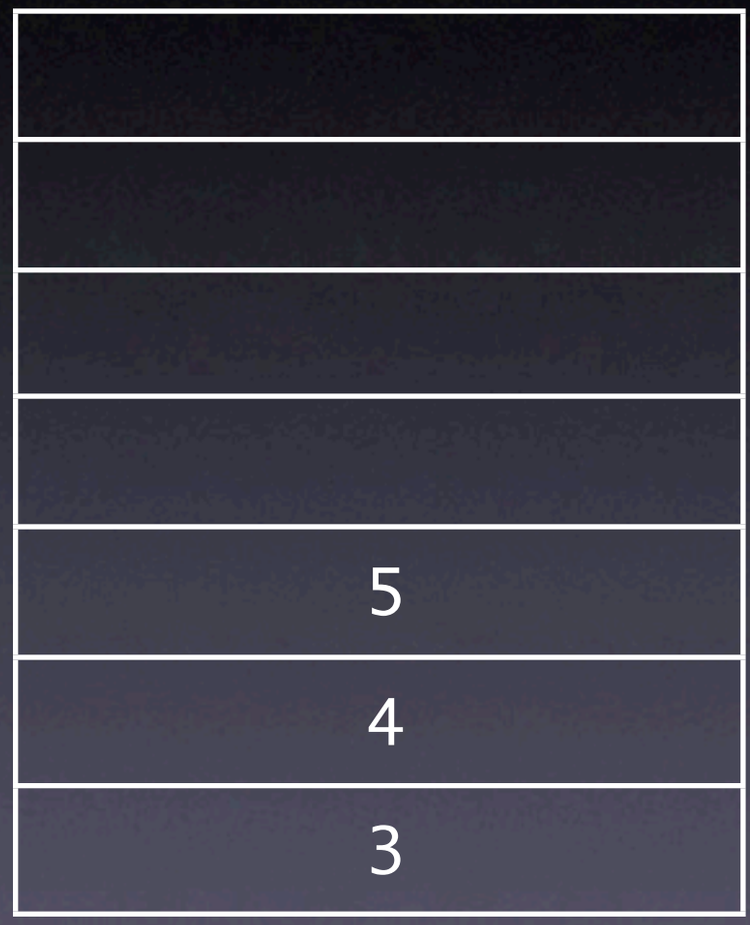
Operand Stack

- Executable operators take their parameters off the stack
- This is why PostScript uses Reverse Polish Notation
- Internally, all computer languages tend to compile down to this type of mechanism

Reverse Polish Notation -- specify the operands before the operator

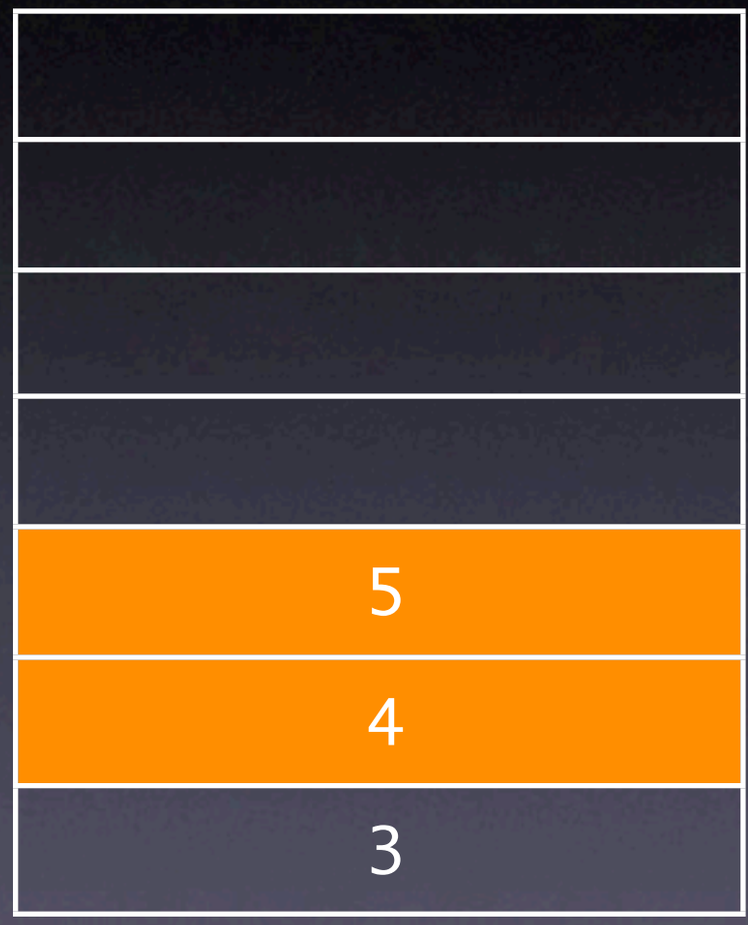
Operand Stack

```
mul add
```



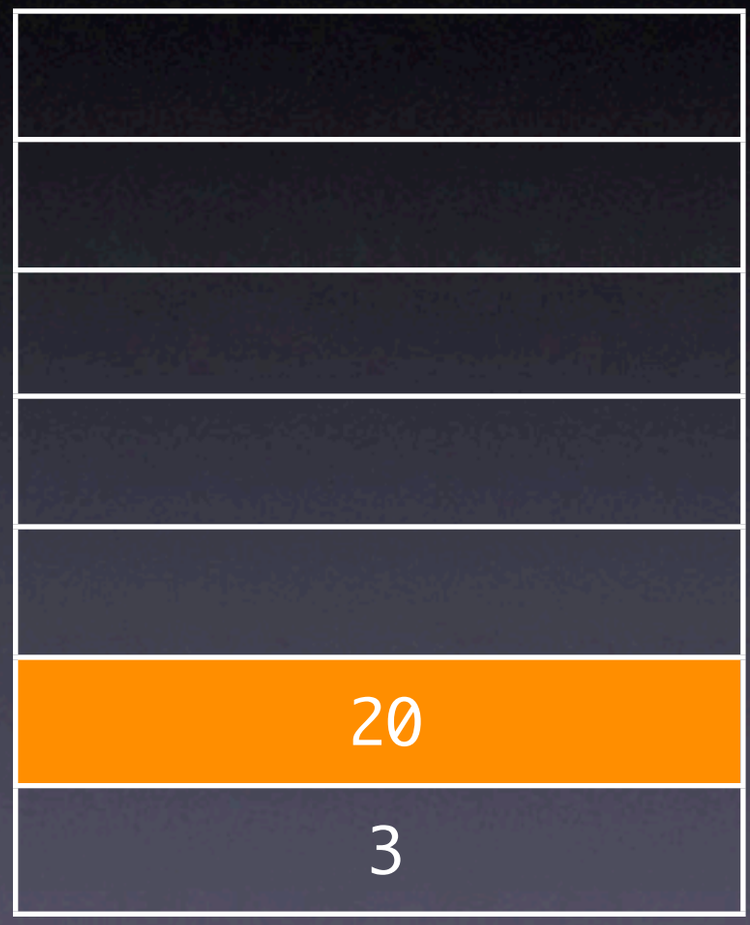
Operand Stack

```
mul add
```



Operand Stack

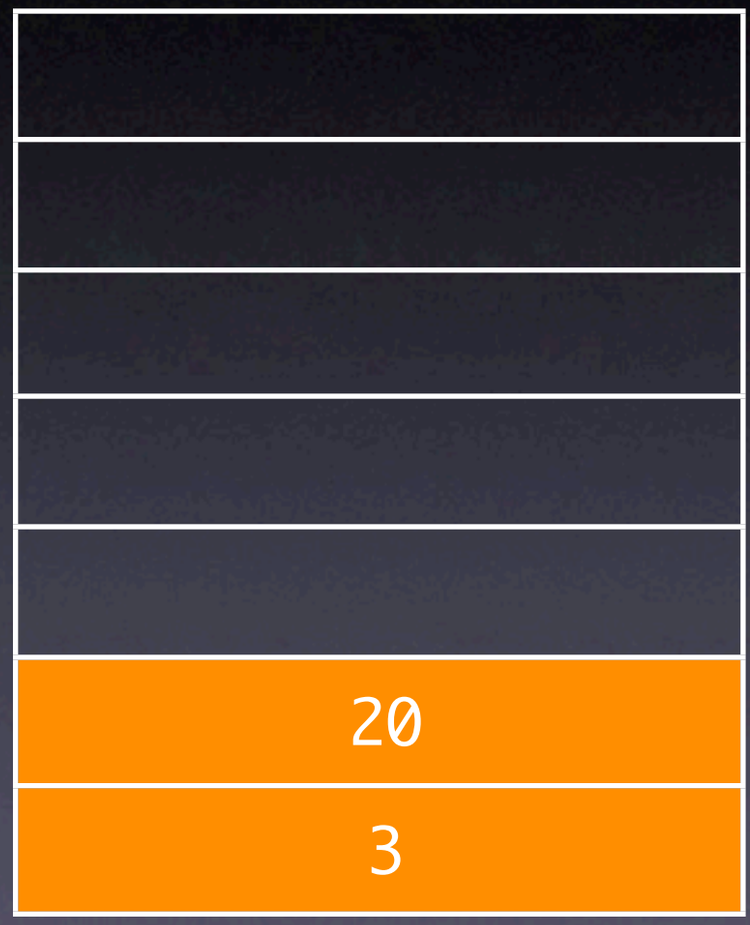
add



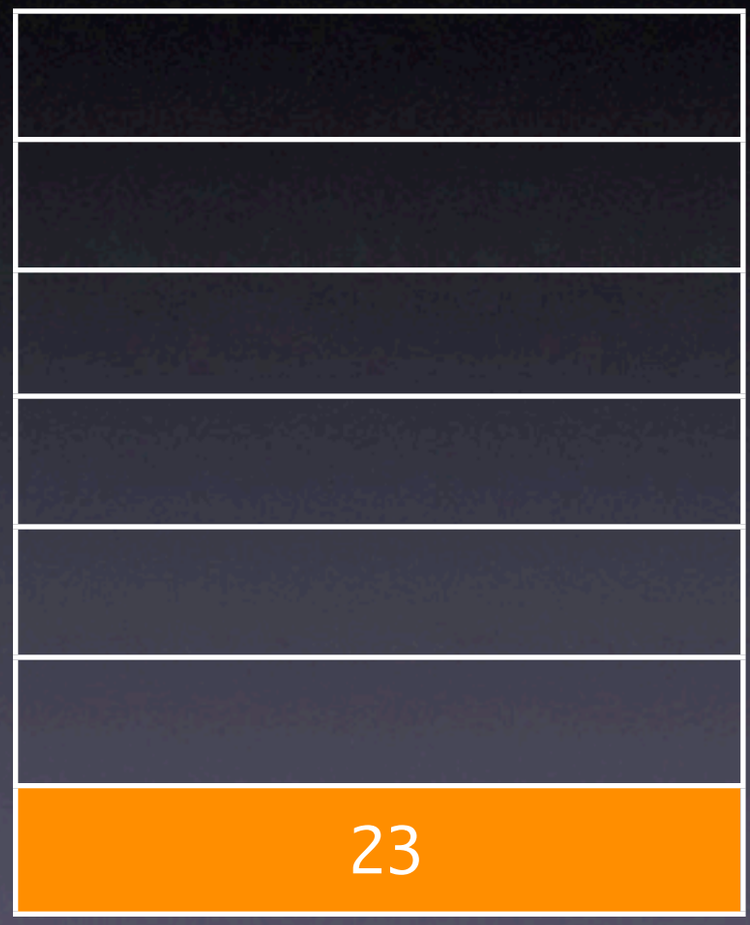
Also puts the result(s) back on the operand stack

Operand Stack

```
add
```



Operand Stack

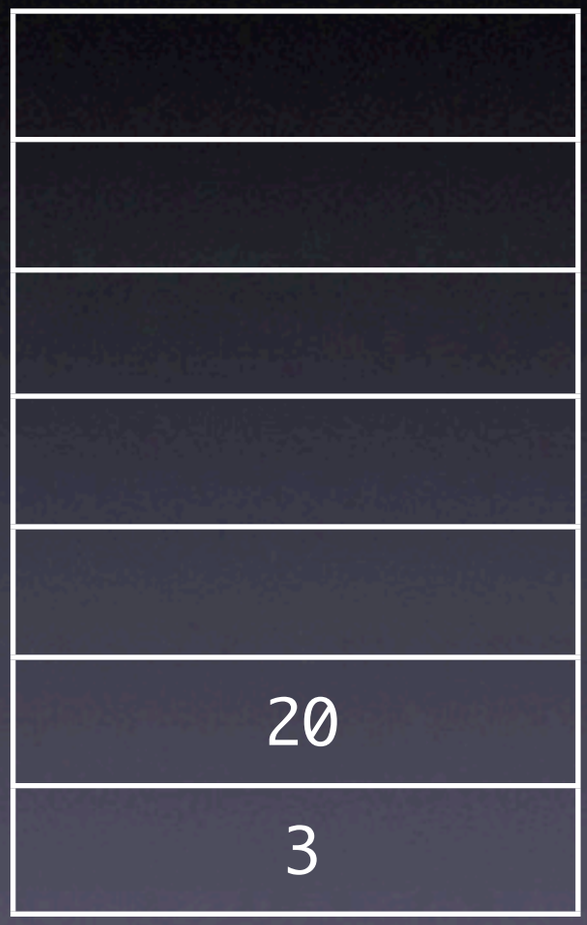


Stack operations

- PostScript provides operators that let you manipulate the operand stack

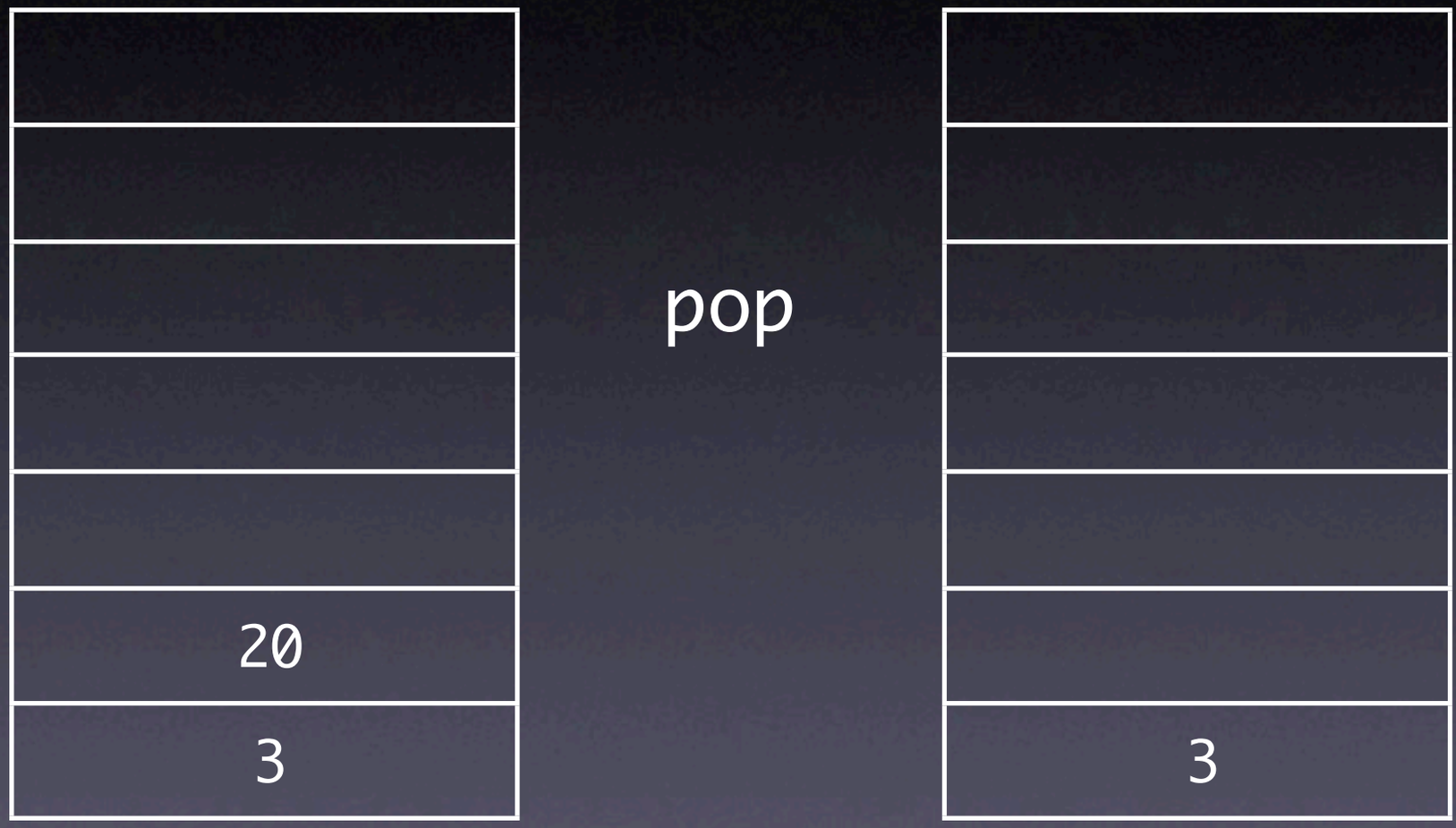
```
pop
clear
exch
dup
roll
```

pop operator

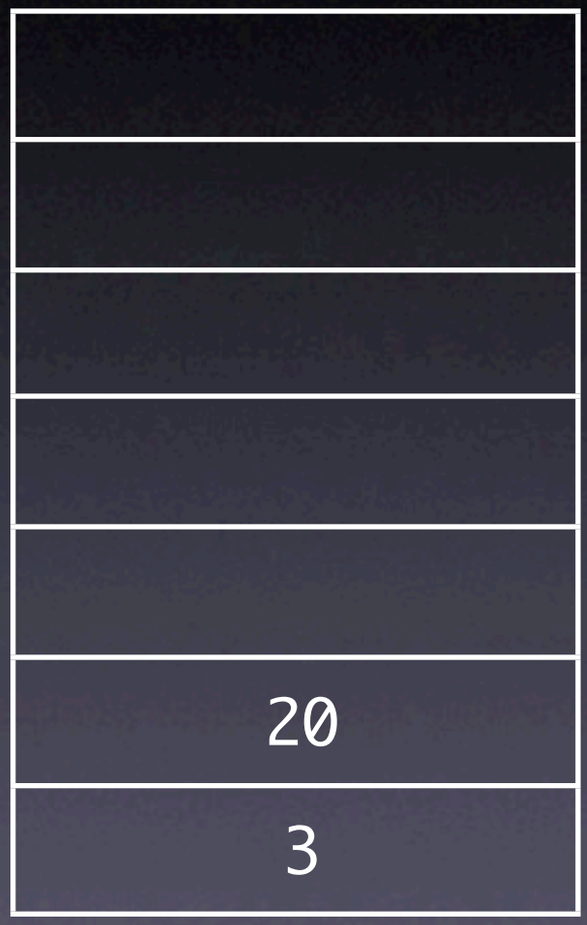


pop

pop operator

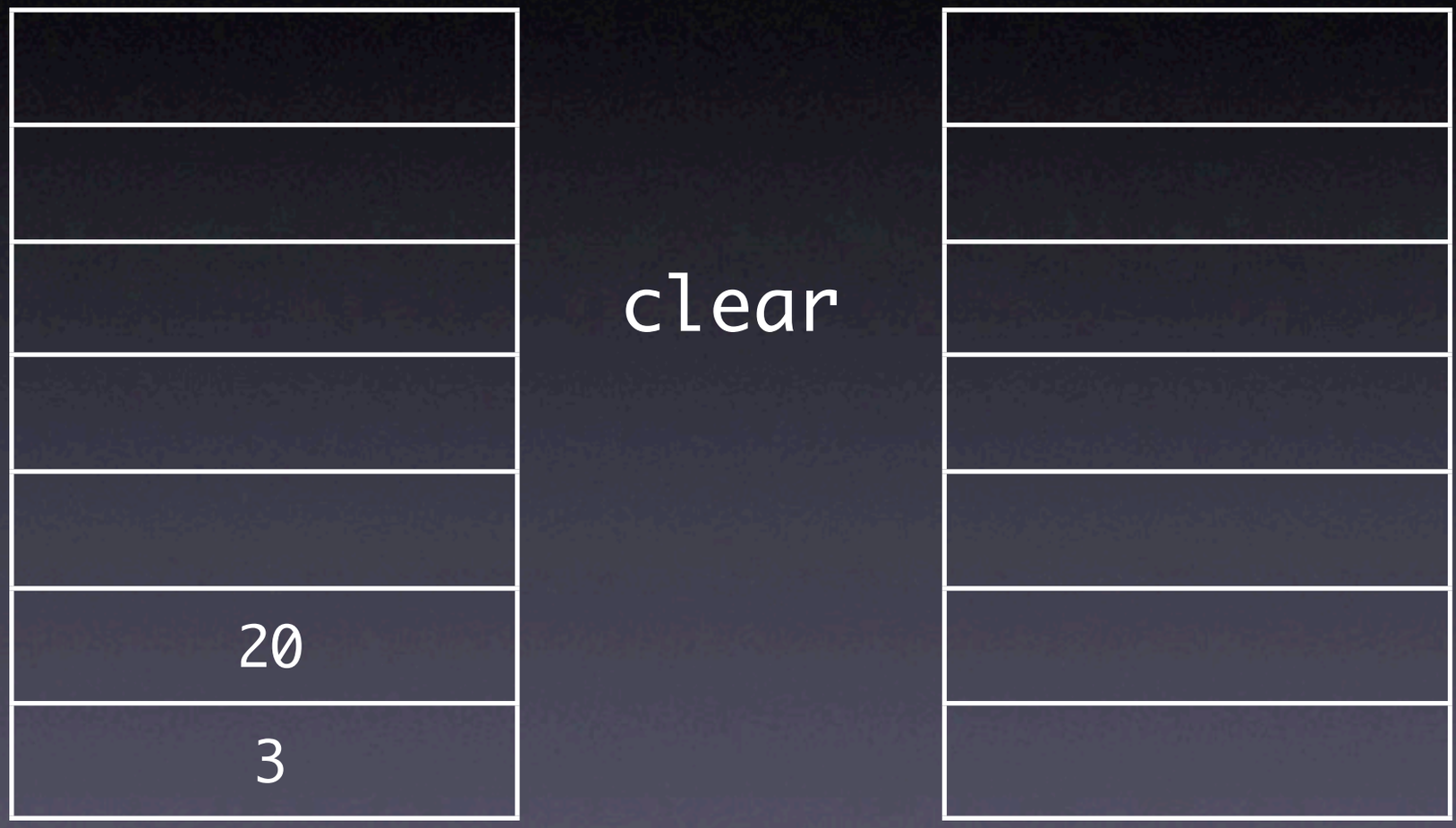


clear operator

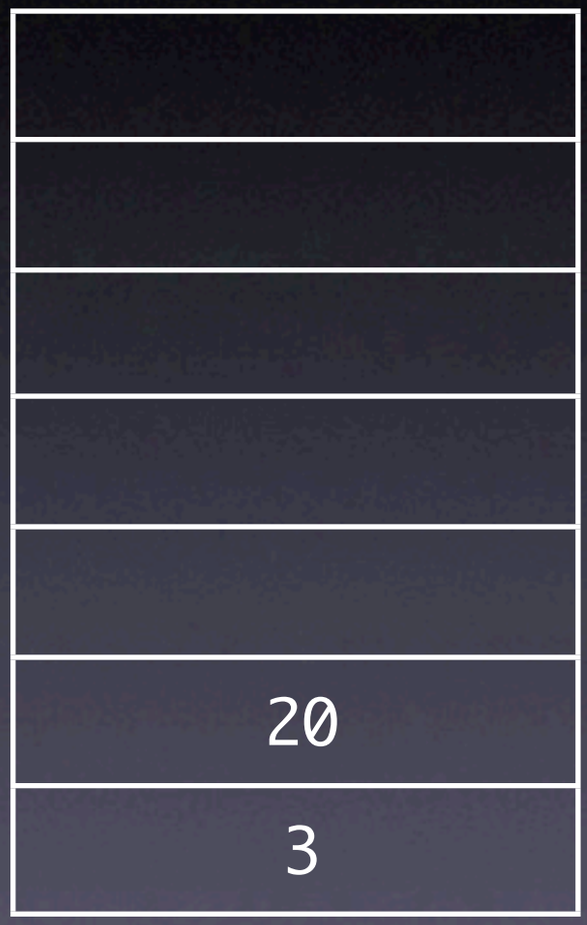


clear

clear operator

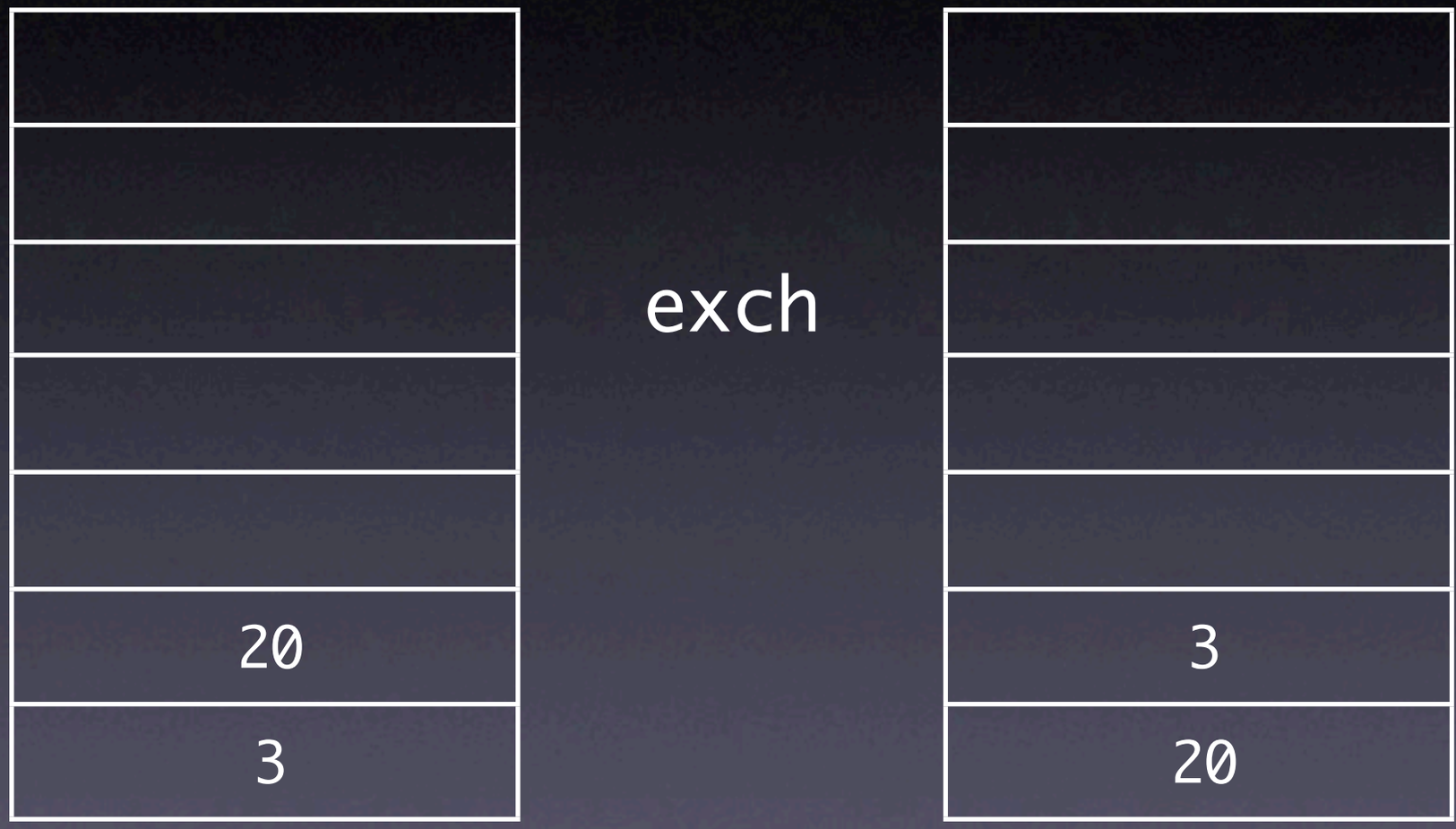


exch operator

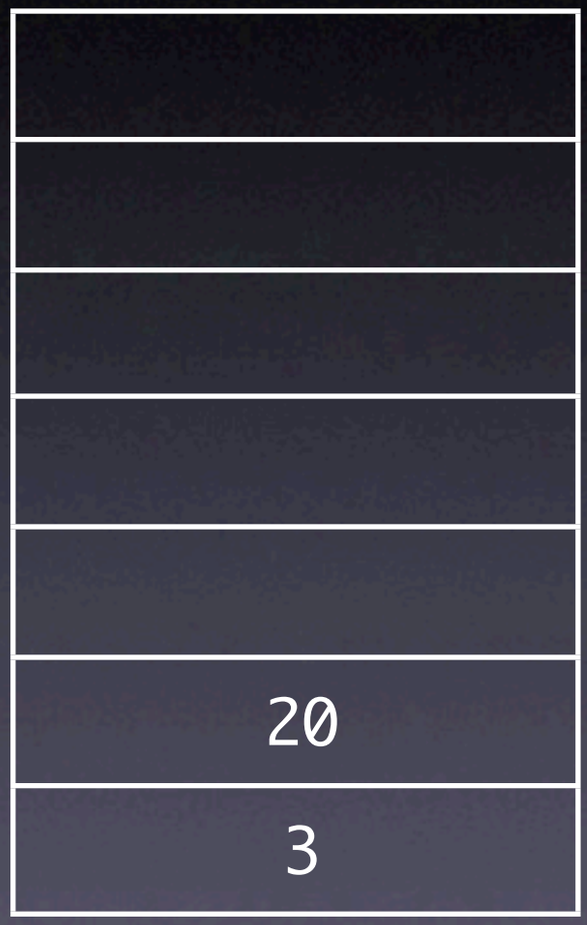


exch

exch operator

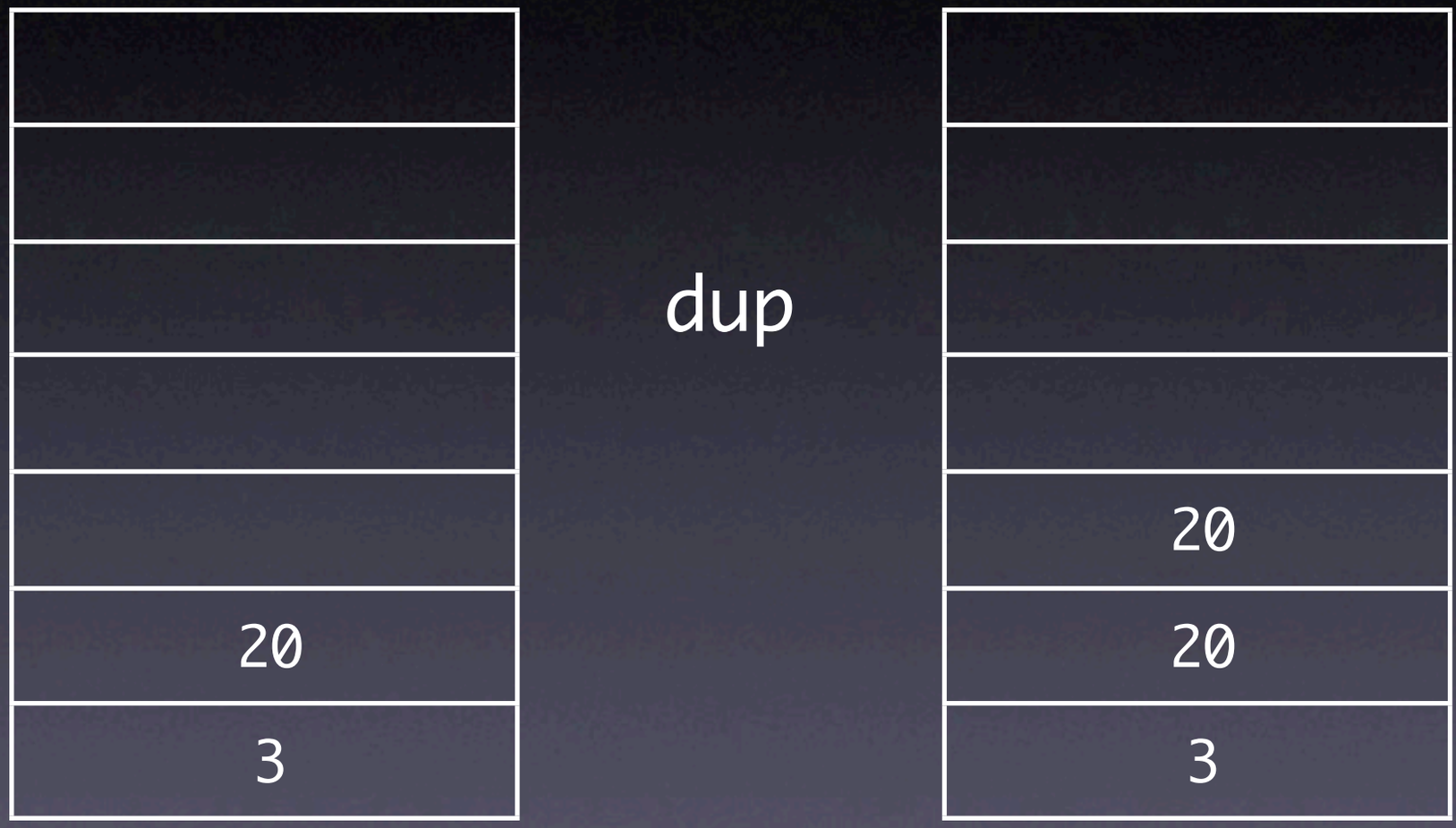


dup operator



dup

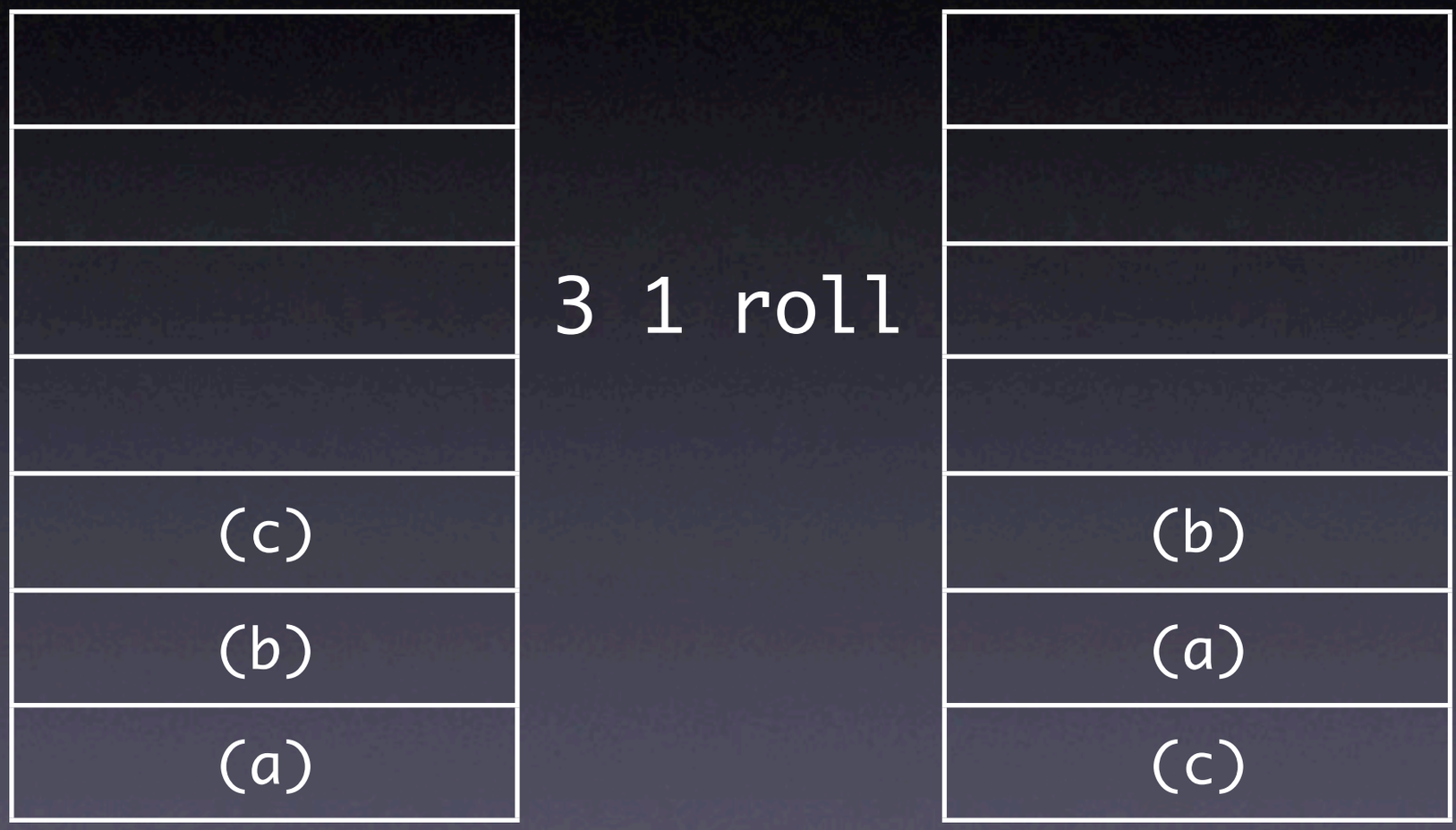
dup operator



roll operator


```
3 1 roll
```

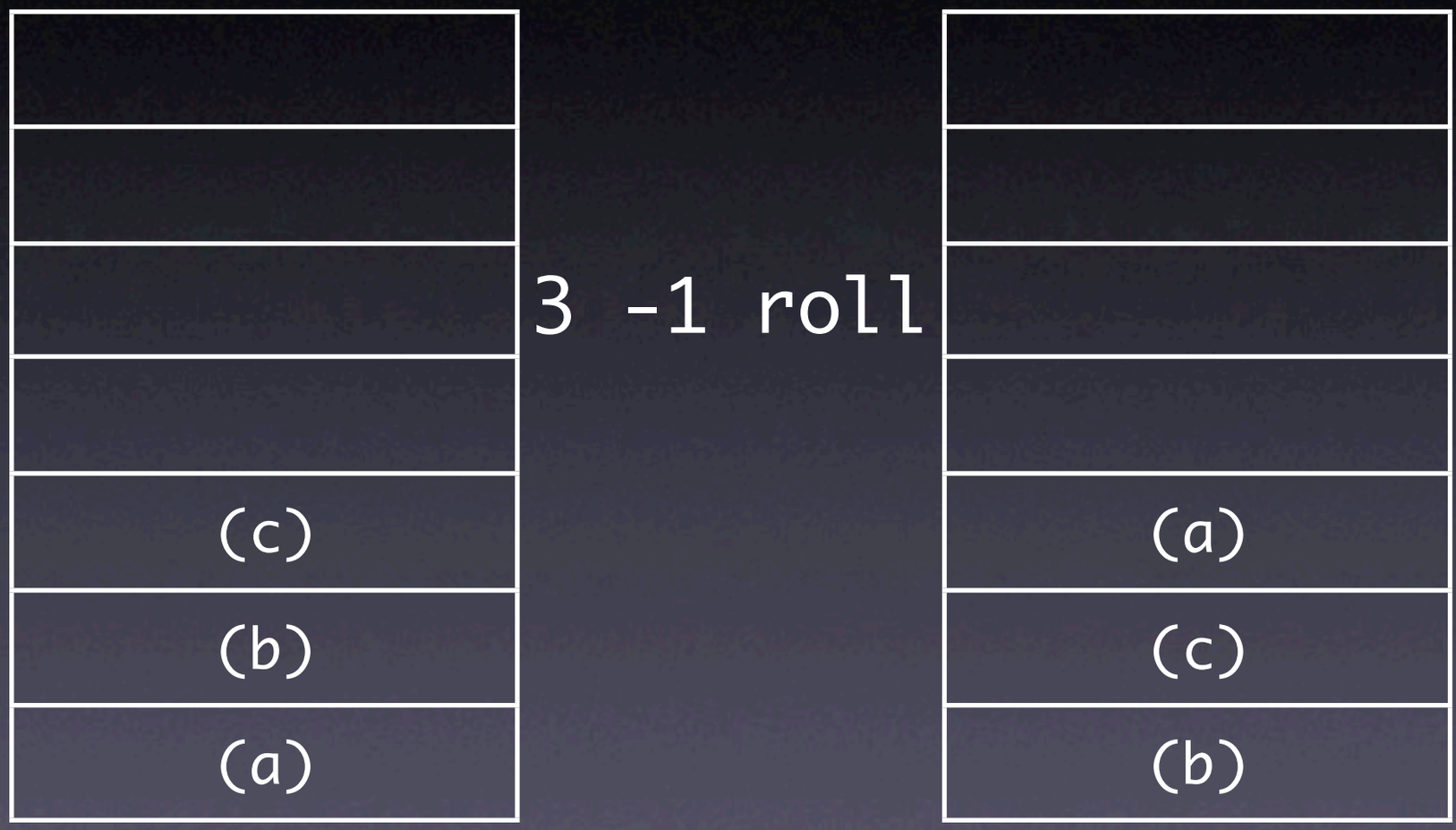
roll operator



roll operator

3 -1 roll

roll operator



Token Types

- Composite Types
  - Strings
  - Arrays
  - Dictionaries (associative arrays)
- These are all placed on the stack as references to the object

So dup only duplicates the reference on the stack

Strings

- Strings are enclosed in parentheses () e.g. `(this is a string including () in it)`
- Use a \ to escape unmatched parentheses
- `string` operator creates an empty string e.g. `10 string`
- `cvs` operator will create a textual representation of an object on the stack

Arrays

- One-dimensional, but untyped
  `[ 1 /Fred 42.5 (Fred) cvs ]`
- Array creation is a stack-based operation
- Access array values using `get`/`put` operators
- Create an empty array with the `array` operator

[places a mark on the stack
] creates an array containing all the objects on the stack
So this array will be 1, /Fred, (42.5) because the cvs operator will have been executed
get/put take an array and an index as parameters -- the array is popped from the stack so make sure you store a reference somewhere

Executable Arrays

- In PostScript, the procedure is just an array of objects
- But the array is marked as *executable*
- Defined:
  ```
  { 1 2 add }
  ```
- Can define it to a name and then use it like an operator
  ```
  /sum { 1 2 add } def
  ```

Dictionaries

- Objects associated to a PostScript name
- Created by `n dict` operator
- Where n is the size of the dictionary
  - Fixed size in Level 1 PostScript

Dictionary Stack

- On the top of the stack is *userdict*, which is used by default to store values
  e.g. `/pointsize 42 def`
- `userdict` is limited by the interpreter's memory

Dictionary Stack

- A non-literal name causes the dictionary stack to be searched to find an association
- If found, that object is executed
- PostScript operators are defined in `systemdict` which is always at the bottom of the stack
- So it is possible to redefine operators. Take care!

Using Dictionaries

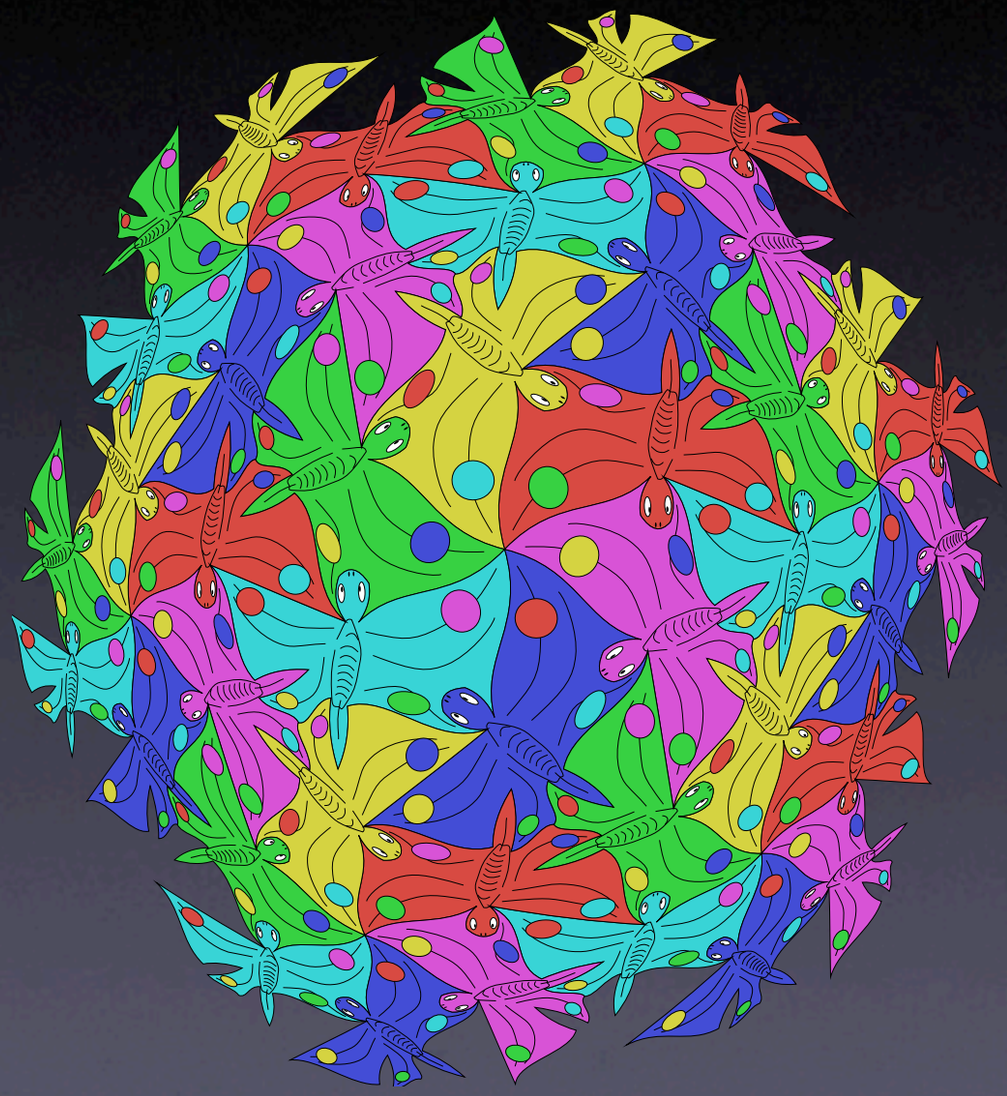
- `/mydict 5 dict def`
  - Creates a dictionary and associates it with the key `/mydict` in `userdict`
- `put`/`get` allow access to values in a dictionary
- `begin` places a dictionary on the dictionary stack, this allows scoping of names
- `end` pops top dictionary from the stack

Anatomy of PostScript

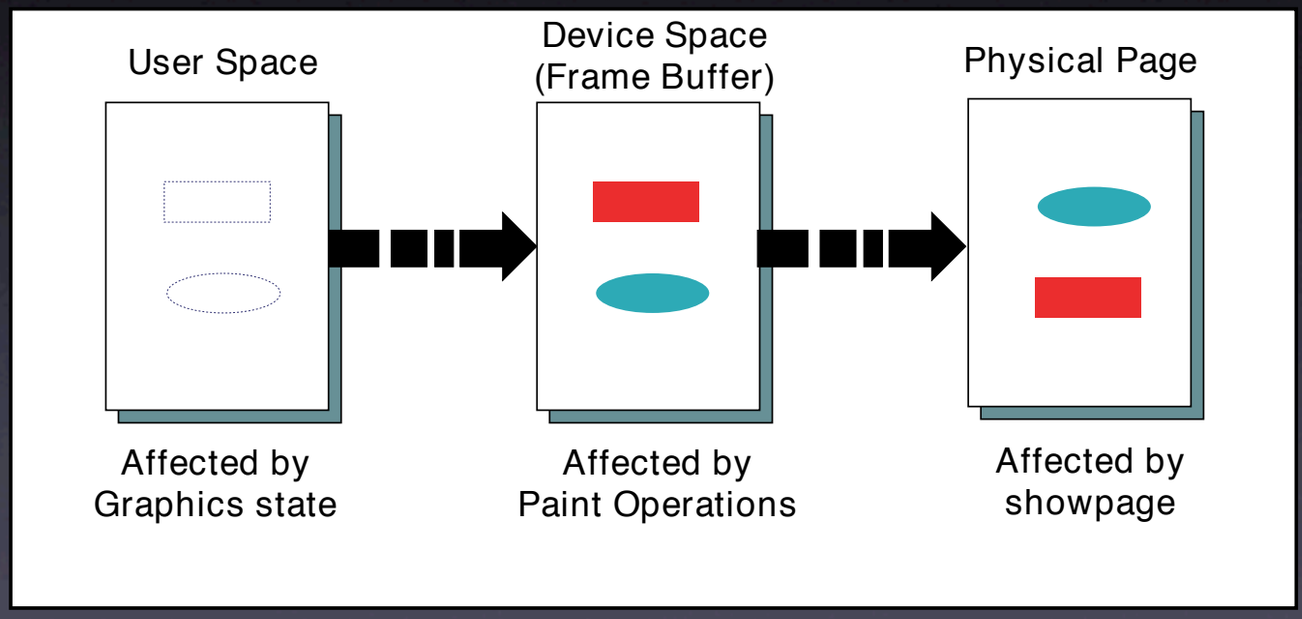
- Document Structuring Convention
- Guidelines, not mandatory
- However, it is good practice to include the standard header at the top of your files

```
%!PS-Adobe-2.0
```

Fancy Graphics



Imaging model



Imaging model

- Origin is at the bottom-left of the page
- Positive is up and to the right
- The current scale is 72dpi
- There is no current point,
  There is no current path

Drawing

- Graphics are created by using operators to form paths
- The path can then be `stroked` or `filled`
- The painting operators use the current colour and change bits in *device space*

Drawing

- Drawing operators are:

`newpath`	clear the current path
`x y moveto`	set the current point to (x,y)
`x y lineto`	draw a line to point (x,y)
`x y rmoveto`	move to currentpoint + (x,y)
`x y rlineto`	draw line to currentpoint +(x,y)
`x y r ang1 ang2 arc`	append anticlockwise circular arc

rmoveto/rlineto allow for relative motion
arcs are centred on x y, with raidus r from angle 1 to angle 2

Transformation

- All drawing operations are run through the Current Transformation Matrix (CTM)
- Set by operators thus:

```
1 2 scale
72 72 moveto
27.5 rotate
```

- Each operator concatenates the relevant new matrix with the current CTM

Graphic Demos

Don’t forget to showpage or you won’t see things!