declarative thinking, declarative practice

@KevlinHenney

Share a Coke with
Kevlin
FROME VALLEY
HEREFORDSHIRE
HENNEY'S
DRY CIDER
MADE FROM 100% FRESH PRESSED JUICE
THE SINGLETON
AGED 12 YEARS
OF DUFFTOWN
TRADE MARK
SINGLETON
gle Malt Scotch Whisk
of Dufftown
DUFFTOWN
STD 1896
TURED FOR
12 YEARS
12
PRODUCT OF
SCOTLAND

Computer Science in the 1960s to 80s spent a lot of effort making languages which were as powerful as possible. Nowadays we have to appreciate the reasons for picking not the most powerful solution but the least powerful.

Tim Berners-Lee

https://www.w3.org/DesignIssues/Principles.html

The reason for this is that the less powerful the language, the more you can do with the data stored in that language. If you write it in a simple declarative form, anyone can write a program to analyze it in many ways.

Tim Berners-Lee

https://www.w3.org/DesignIssues/Principles.html

Make — A Program for Maintaining Computer Programs

S. I. Feldman

ABSTRACT

In a programming project, it is easy to lose track of which files need to be reprocessed or recompiled after a change is made in some part of the source. *Make* provides a simple mechanism for maintaining up-to-date versions of programs that result from many operations on a number of files. It is possible to tell *Make* the sequence of commands that create certain files, and the list of files that require other files to be current before the operations can be done. Whenever a change is made in any part of the program, the *Make* command will create the proper files simply, correctly, and with a minimum amount of effort.

The basic operation of *Make* is to find the name of a needed target in the description, ensure that all of the files on which it depends exist and are up to date, and then create the target if it has not been modified since its generators were. The description file really defines the graph of dependencies; *Make* does a depth-first search of this graph to determine what work is really necessary.

Make also provides a simple macro substitution facility and the ability to encapsulate commands in a single file for convenient administration.

August 15, 1978

The makefile language is similar to declarative programming. This class of language, in which necessary end conditions are described but the order in which actions are to be taken is not important, is sometimes confusing to programmers used to imperative programming.

http://en.wikipedia.org/wiki/Make_(software)

```
object
    {}
    { members }

members
    pair
    pair , members

pair
    string : value

array
    []
    [ elements ]

elements
    value
    value , elements

value
    string
    number
    object
    array
    true
    false
    null
```

```
string
    ""
    " chars "

chars
    char
    char chars

char
    any-non-control-char
    \"
    \\
    \/
    \b
    \f
    \n
    \r
    \t
    \u four-hex-digits
```

```
number
    integer
    integer fraction
    integer exponent
    integer fraction exponent

integer
    digit
    non-zero-digit digits
    - digit
    - non-zero-digit digits

fraction
    . digits

exponent
    e digits

e
    e
    e+
    e-
    E
    E+
    E-
```

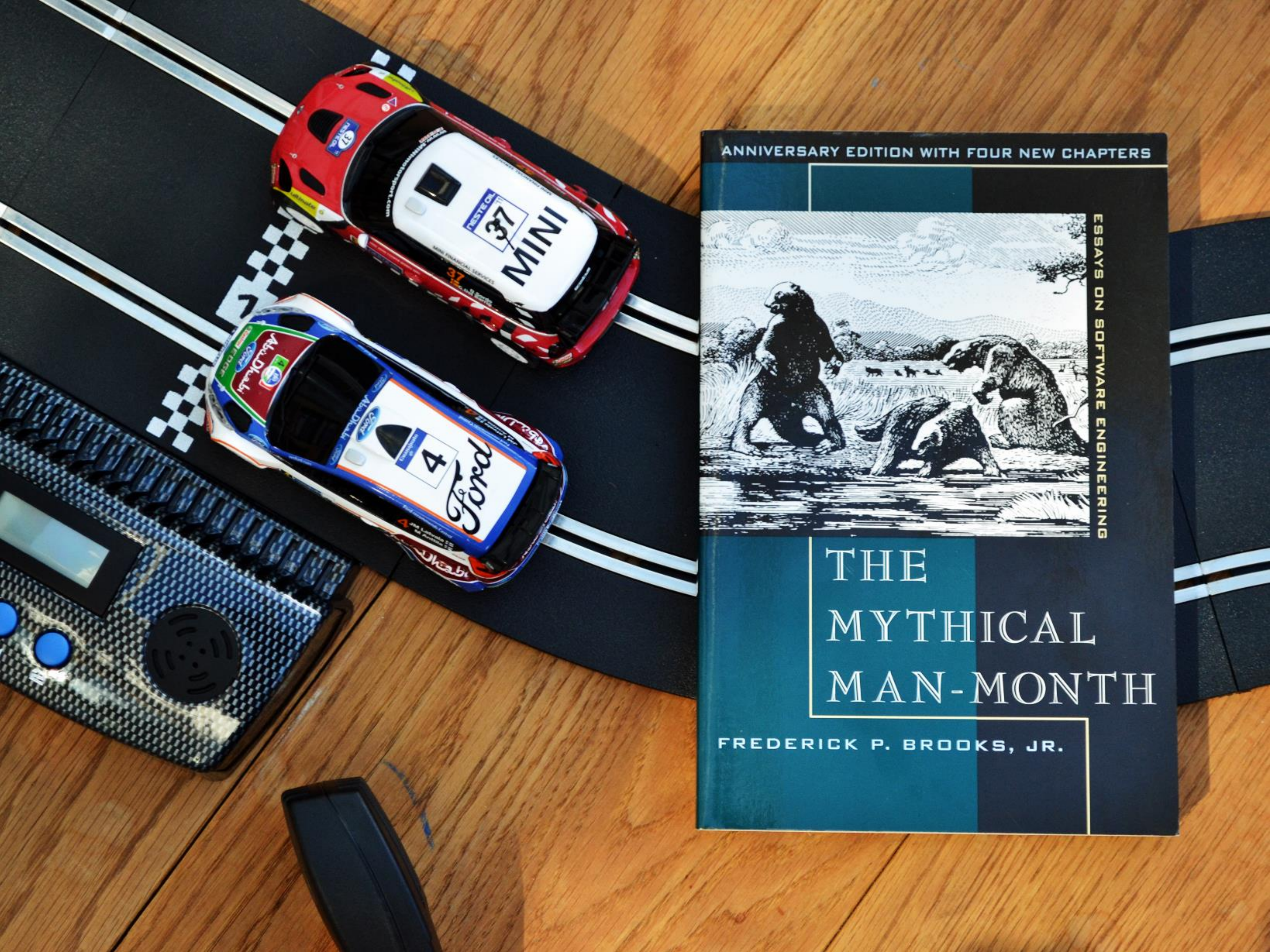
```
group ::=
    '(' expression ')'
factor ::=
    integer | group
term ::=
    factor (('*' factor) | ('/' factor))*
expression ::=
    term (('+' term) | ('-' term))*
```

```
group =
    '(' >> expression >> ')';
factor =
    integer | group;
term =
    factor >> *(('*' >> factor) | ('/' >> factor));
expression =
    term >> *(('+' >> term) | ('-' >> term));
```

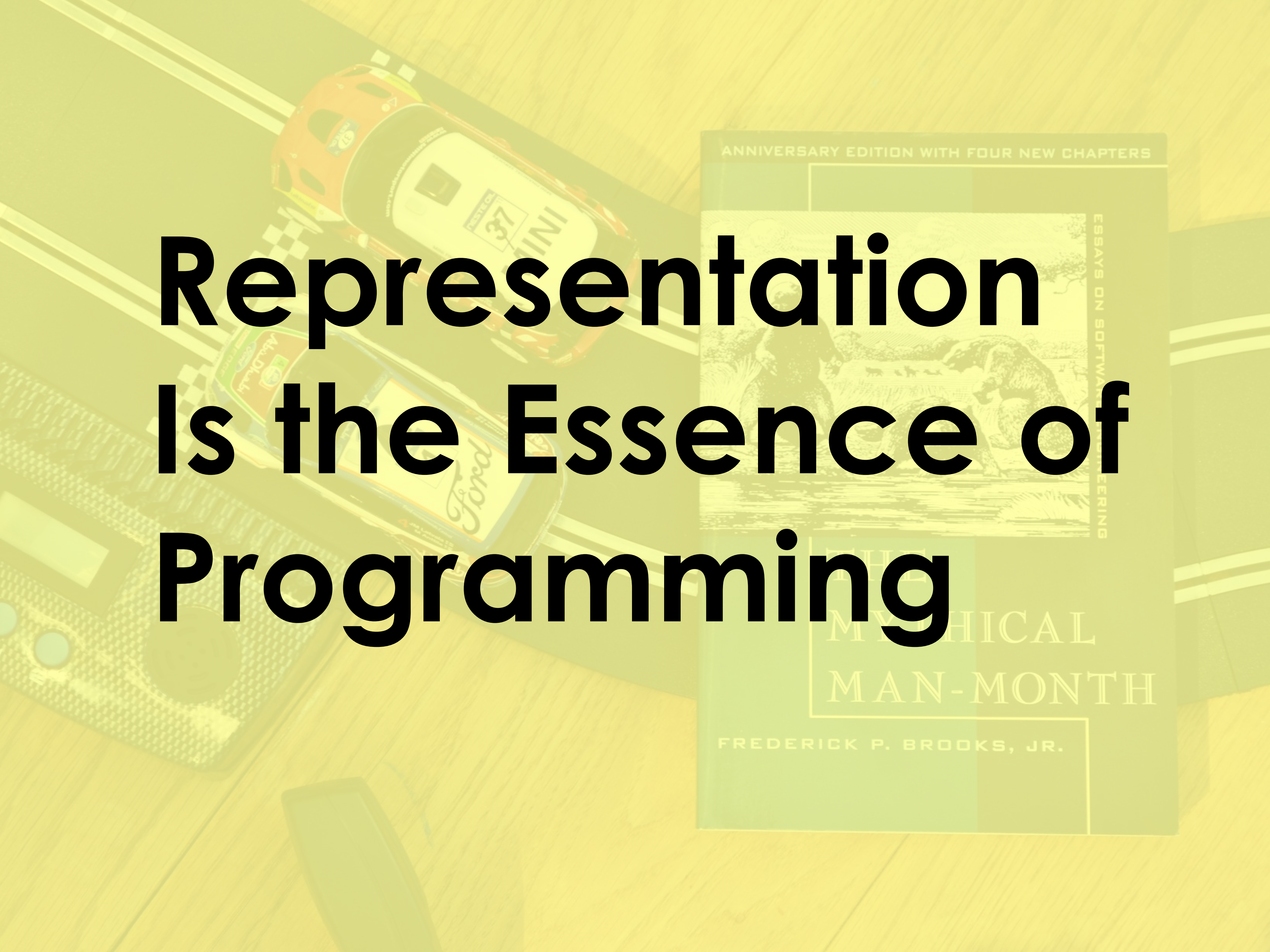
Algorithms +
Data Structures =
Programs

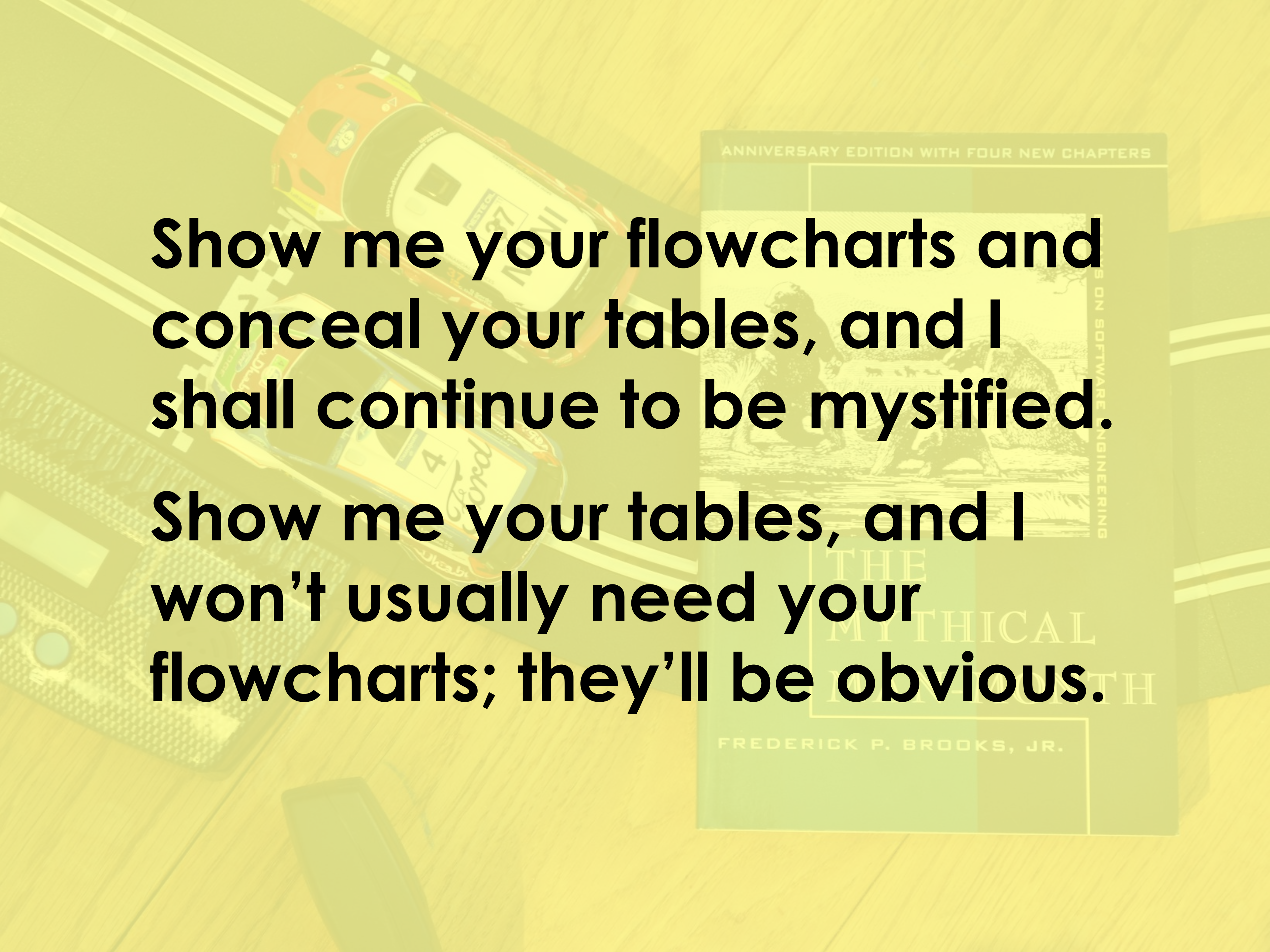
Niklaus Wirth

Data Structures = Programs


37
MINI
4
Ford
ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

Representation Is the Essence of Programming





ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
S ON SOFTWARE ENGINEERING
THE MYTHICAL
TH
FREDERICK P. BROOKS, JR.
4
Ford

Show me your flowcharts and conceal your tables, and I shall continue to be mystified.

Show me your tables, and I won't usually need your flowcharts; they'll be obvious.

> **Excel is the world's most popular functional language.**
>
> Simon Peyton-Jones

```
squares = []
i = 1
while i < 101:
    squares.append(i**2)
    i += 1
```

```python
squares = []
for i in range(1, 101):
    squares.append(i**2)
```

```python
squares = [i**2 for i in range(1, 101)]
```

intension, *n. (Logic)*

- the set of characteristics or properties by which the referent or referents of a given expression is determined; the sense of an expression that determines its reference in every possible world, as opposed to its actual reference. For example, the intension of *prime number* may be *having non-trivial integral factors*, whereas its **extension** would be the set {2, 3, 5, 7, ...}.

E J Borowski and J M Borwein
Dictionary of Mathematics

$$\{\, \underset{\substack{\uparrow\\ \textit{select}}}{x^2} \mid x \underset{\substack{\uparrow\\ \textit{from}}}{\in} \mathrm{N},\ x \underset{\substack{\uparrow\\ \textit{where}}}{\geq} 1 \land x \leq 100 \,\}$$

A list comprehension is a syntactic construct available in some programming languages for creating a list based on existing lists. It follows the form of the mathematical *set-builder notation* (*set comprehension*) as distinct from the use of map and filter functions.

http://en.wikipedia.org/wiki/List_comprehension

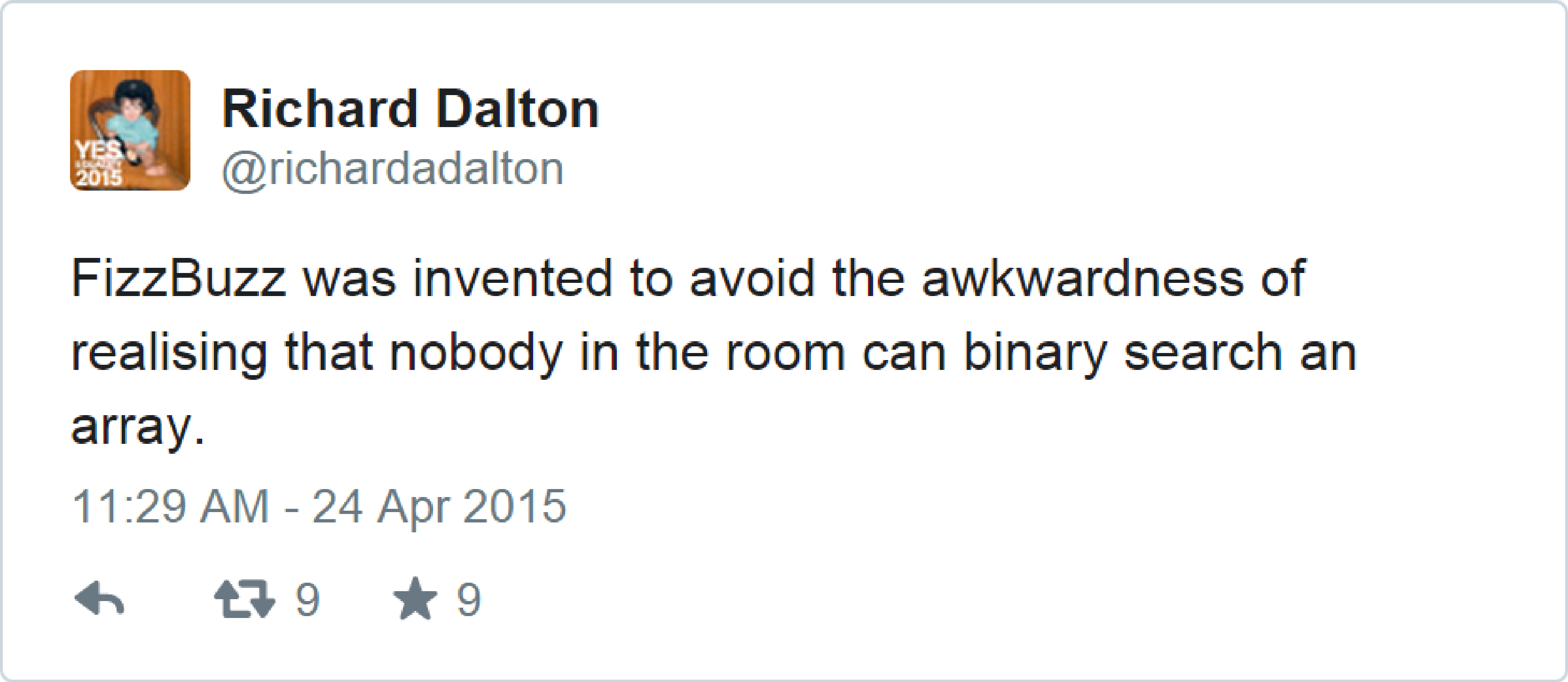
```
[x^2 | x <- [1..100]]
```

$$\{\, x^2 \mid x \in N,\ x \geq 1 \,\}$$

```
[x^2 | x <- [1..]]
```

lazy evaluation

```
take 100 [x^2 | x <- [1..]]
```



Richard Dalton
@richardadalton

FizzBuzz was invented to avoid the awkwardness of realising that nobody in the room can binary search an array.

11:29 AM - 24 Apr 2015

https://twitter.com/richardadalton/status/591534529086693376

```python
def fizzbuzz(n):
    result = ''
    if n % 3 == 0:
        result += 'Fizz'
    if n % 5 == 0:
        result += 'Buzz'
    if not result:
        result = str(n)
    return result
```

```
def fizzbuzz(n):
    if n % 15 == 0:
        return 'FizzBuzz'
    elif n % 3 == 0:
        return 'Fizz'
    elif n % 5 == 0:
        return 'Buzz'
    else:
        return str(n)
```

```python
def fizzbuzz(n):
    return (
        'FizzBuzz' if n % 15 == 0 else
        'Fizz' if n % 3 == 0 else
        'Buzz' if n % 5 == 0 else
        str(n))
```

```python
def fizzbuzz(n):
    return (
        'FizzBuzz' if n in range(0, 101, 15) else
        'Fizz' if n in range(0, 101, 3) else
        'Buzz' if n in range(0, 101, 5) else
        str(n))
```

```python
fizzes  = [''] + ([''] * 2 + ['Fizz']) * 33 + ['']

buzzes  = [''] + ([''] * 4 + ['Buzz']) * 20

numbers = list(map(str, range(0, 101)))

def fizzbuzz(n):
    return fizzes[n] + buzzes[n] or numbers[n]
```

```
actual = [fizzbuzz(n) for n in range(1, 101)]
truths = [
  every result is 'Fizz', 'Buzz', 'FizzBuzz' or a decimal string,
  every decimal result corresponds to its ordinal position,
  every third result contains 'Fizz',
  every fifth result contains 'Buzz',
  every fifteenth result is 'FizzBuzz',
  the ordinal position of every 'Fizz' result is divisible by 3,
  the ordinal position of every 'Buzz' result is divisible by 5,
  the ordinal position of every 'FizzBuzz' result is divisible by 15
]
all(truths)
```

```
actual = [fizzbuzz(n) for n in range(1, 101)]
truths = [
  all(a in {'Fizz', 'Buzz', 'FizzBuzz'} or a.isdecimal() for a in actual),
  all(int(a) == n for n, a in enumerate(actual, 1) if a.isdecimal()),
  all('Fizz' in a for a in actual[2::3]),
  all('Buzz' in a for a in actual[4::5]),
  all(a == 'FizzBuzz' for a in actual[14::15]),
  all(n % 3 == 0 for n, a in enumerate(actual, 1) if a == 'Fizz'),
  all(n % 5 == 0 for n, a in enumerate(actual, 1) if a == 'Buzz'),
  all(n % 15 == 0 for n, a in enumerate(actual, 1) if a == 'FizzBuzz')
]
all(truths)
```

fizzes

buzzes

words

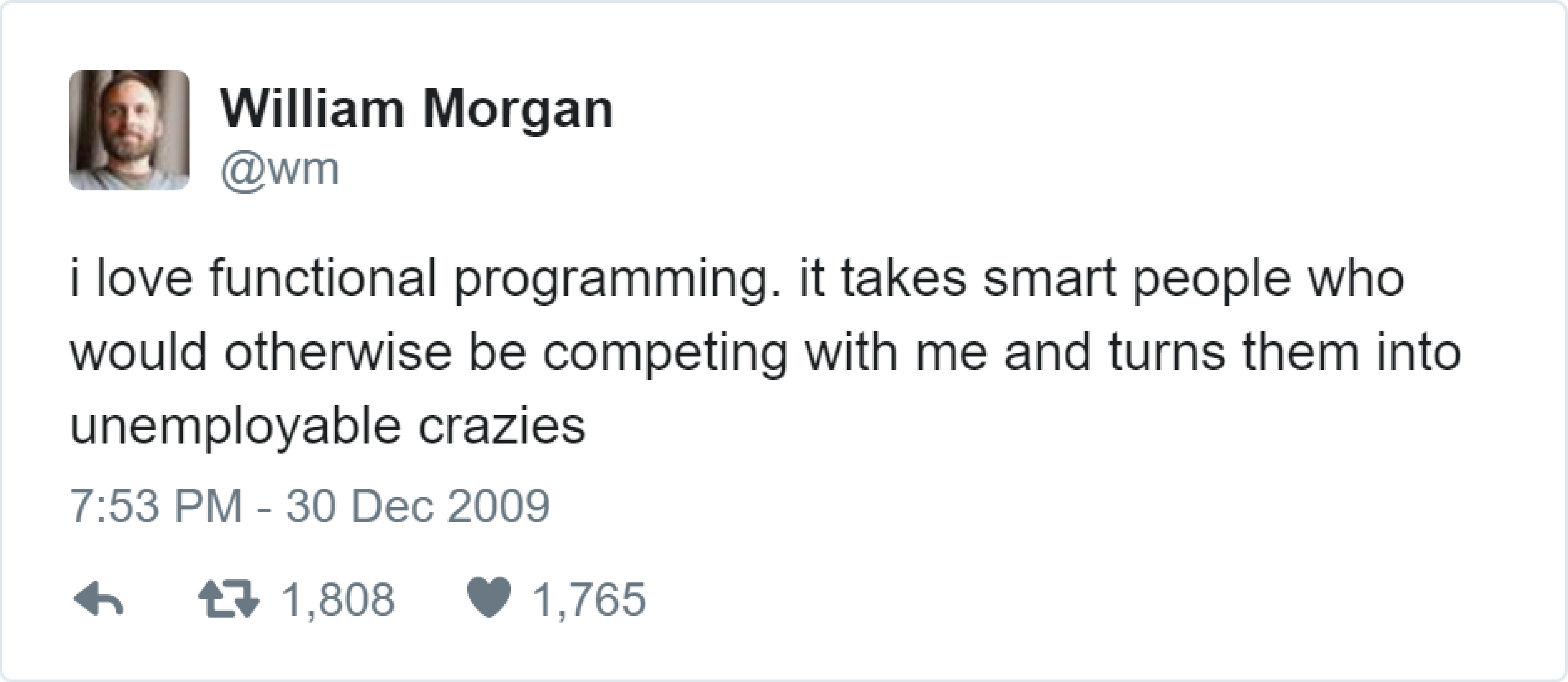
numbers

choice

fizzbuzz

```
fizzes   = cycle ["", "", "Fizz"]
buzzes   = cycle ["", "", "", "", "Buzz"]
words    = zipWith (++) fizzes buzzes
numbers  = map show [1..]
choice   = max
fizzbuzz = zipWith choice words numbers
```

```
fizzes   = cycle ["", "", "Fizz"]
buzzes   = cycle ["", "", "", "", "Buzz"]
words    = zipWith (++) fizzes buzzes
numbers  = map show [1..]
choice   = max
fizzbuzz = zipWith choice words numbers
take 100 fizzbuzz
```



William Morgan
@wm

i love functional programming. it takes smart people who would otherwise be competing with me and turns them into unemployable crazies

7:53 PM - 30 Dec 2009

1,808 ♥ 1,765

https://twitter.com/wm/status/7206700352

f / WordFriday

bi-quinary coded decimal, *noun*

- A system of representing numbers based on counting in fives, with an additional indicator to show whether the count is in the first or second half of the decimal range, i.e., whether the number represented is in the range 0–4 or 5–9.
- This system is found in many abacus systems, with paired columns of counters (normally aligned) representing each bi-quinary range.
- The Roman numeral system is also a form of bi-quinary coded decimal.

```python
def roman(number):
    result = ''
    while number >= 1000:
        result += 'M'
        number -= 1000
    if number >= 900:
        result += 'CM'
        number -= 900
    if number >= 500:
        result += 'D'
        number -= 500
    if number >= 400:
        result += 'CD'
        number -= 400
    while number >= 100:
        result += 'C'
        number -= 100
    if number >= 90:
```

```python
def roman(number):
    result = ''
    while number >= 1000:
        result += 'M'
        number -= 1000
    if number >= 900:
        result += 'CM'
        number -= 900
    if number >= 500:
        result += 'D'
        number -= 500
    if number >= 400:
        result += 'CD'
        number -= 400
    while number >= 100:
        result += 'C'
        number -= 100
    if number >= 90:
        result += 'XC'
        number -= 90
    if number >= 50:
        result += 'L'
        number -= 50
    if number >= 40:
        result += 'XL'
        number -= 40
    while number >= 10:
        result += 'X'
        number -= 10
    if number >= 9:
        result += 'IX'
        number -= 9
    if number >= 5:
        result += 'V'
        number -= 5
    if number >= 4:
        result += 'IV'
        number -= 4
    while number >= 1:
        result += 'I'
        number -= 1
    return result
```

```python
def roman(number):
    multiples = [
        (1000, 'M'), (900, 'CM'),
        (500, 'D'),  (400, 'CD'),
        (100, 'C'),  (90, 'XC'),
        (50, 'L'),   (40, 'XL'),
        (10, 'X'),   (9, 'IX'),
        (5, 'V'),    (4, 'IV'),
        (1, 'I')
    ]
    result = ''
    for value, letters in multiples:
        result += (number // value) * letters
        number %= value
    return result
```

```
def roman(number):
    return (
        (number * 'I')
        .replace('IIIII', 'V')
        .replace('IIII', 'IV')
        .replace('VV', 'X')
        .replace('VIV', 'IX')
        .replace('XXXXX', 'L')
        .replace('XXXX', 'XL')
        .replace('LL', 'C')
        .replace('LXL', 'XC')
        .replace('CCCCC', 'D')
        .replace('CCCC', 'CD')
        .replace('DD', 'M')
        .replace('DCD', 'CM')
    )
```

Java 8 Streams Cheat Sheet

REBELLABS by ZEROTURNAROUND

Definitions

☑ A stream **is** a pipeline of functions that can be evaluated.

☑ Streams **can** transform data.

☒ A stream **is not** a data structure.

☒ Streams **cannot** mutate data.

Intermediate operations

- Always return streams.
- Lazily executed.

Common examples include:

Function	Preserves count	Preserves type	Preserves order
map	✓	✗	✓
filter	✗	✓	✓
distinct	✗	✓	✓
sorted	✓	✓	✗
peek	✓	✓	✓

Stream examples

Get the unique surnames in uppercase of the first 15 book authors that are 50 years old or over.

```
library.stream()
       .map(book -> book.getAuthor())
       .filter(author -> author.getAge() >= 50)
       .limit(15)
       .map(Author::getSurname)
       .map(String::toUpperCase)
       .distinct()
       .collect(toList());
```

Compute the sum of ages of all female authors younger than 25.

```
library.stream()
       .map(Book::getAuthor)
       .filter(a -> a.getGender() == Gender.FEMALE)
       .map(Author::getAge)
       .filter(age -> age < 25)
       .reduce(0, Integer::sum):
```

Terminal operations

- Return concrete types or produce a side effect.
- Eagerly executed.

Common examples include:

Function	Output	When to use
reduce	concrete type	to cumulate elements
collect	list, map or set	to group elements
forEach	side effect	to perform a side effect on elements

Parallel streams

Parallel streams use the common ForkJoinPool for threading.

```
library.parallelStream()...
```

or intermediate operation:

```
IntStream.range(1, 10).parallel()...
```

Useful operations

Grouping:

```
library.stream().collect(
      groupingBy(Book::getGenre));
```

Stream ranges:

```
IntStream.range(0, 20)...
```

Infinite streams:

```
IntStream.iterate(0, e -> e + 1)...
```

Max/Min:

```
IntStream.range(1, 10).max();
```

FlatMap:

```
twitterList.stream()
      .map(member -> member.getFollowers())
      .flatMap(followers -> followers.stream())
      .collect(toList());
```

Pitfalls

☒ Don't update shared mutable variables i.e.

```
List<Book> myList = new ArrayList<>();
library.stream().forEach
       (e -> myList.add(e));
```

☒ Avoid blocking operations when using parallel streams.

http://zeroturnaround.com/rebellabs/java-8-streams-cheat-sheet/

Java 8 Streams Cheat Sheet

REBELLABS by ZEROTURNAROUND

Definitions

☑ A stream **is** a pipeline of functions that can be evaluated.

☑ Streams **can** transform data.

☒ A stream **is not** a data structure.

☒ Streams **cannot** mutate data.

Intermediate operations

- Always return streams.
- Lazily executed.

Common examples include:

Function	Preserves count	Preserves type	Preserves order
map	✓	✗	✓
filter	✗	✓	✓
distinct	✗	✓	✓
sorted	✓	✓	✗
peek	✓	✓	✓

Stream examples

Get the unique surnames in uppercase of the first 15 book authors that are 50 years old or over.

```
library.stream()
    .map(book -> book.getAuthor())
    .filter(author -> author.getAge() >= 50)
    .map(Author::getSurname)
    .map(String::toUpperCase)
    .distinct()
    .limit(15)
    .collect(toList());
```

Compute the sum of ages of all female authors younger than 25.

```
library.stream()
    .map(Book::getAuthor)
    .filter(a -> a.getGender() == Gender.FEMALE)
    .map(Author::getAge)
    .filter(age -> age < 25)
    .reduce(0, Integer::sum):
```

Terminal operations

- Return concrete types or produce a side effect.
- Eagerly executed.

Common examples include:

Function	Output	When to use
reduce	concrete type	to cumulate elements
collect	list, map or set	to group elements
forEach	side effect	to perform a side effect on elements

Parallel streams

Parallel streams use the common ForkJoinPool for threading.

```
library.parallelStream()...
```

or intermediate operation:

```
IntStream.range(1, 10).parallel()...
```

Useful operations

Grouping:

```
library.stream().collect(
    groupingBy(Book::getGenre));
```

Stream ranges:

```
IntStream.range(0, 20)...
```

Infinite streams:

```
IntStream.iterate(0, e -> e + 1)...
```

Max/Min:

```
IntStream.range(1, 10).max();
```

FlatMap:

```
twitterList.stream()
    .map(member -> member.getFollowers())
    .flatMap(followers -> followers.stream())
    .collect(toList());
```

Pitfalls

☒ Don't update shared mutable variables i.e.

```
List<Book> myList = new ArrayList<>();
library.stream().forEach
    (e -> myList.add(e));
```

☒ Avoid blocking operations when using parallel streams.

http://zeroturnaround.com/rebellabs/java-8-streams-cheat-sheet/

```
// Get the unique surnames in uppercase of the
// first 15 book authors that are 50 years old
// or older?

library.stream()
       .map(book -> book.getAuthor())
       .filter(author -> author.getAge() >= 50)
       .limit(15)
       .map(Author::getSurname)
       .map(String::toUpperCase)
       .distinct()
       .collect(toList()))
```

```
// Get the first 15 unique surnames in
// uppercase of the book authors that are 50
// years old or older.

library.stream()
        .map(book -> book.getAuthor())
        .filter(author -> author.getAge() >= 50)
        .map(Author::getSurname)
        .map(String::toUpperCase)
        .distinct()
        .limit(15)
        .collect(toList()))
```

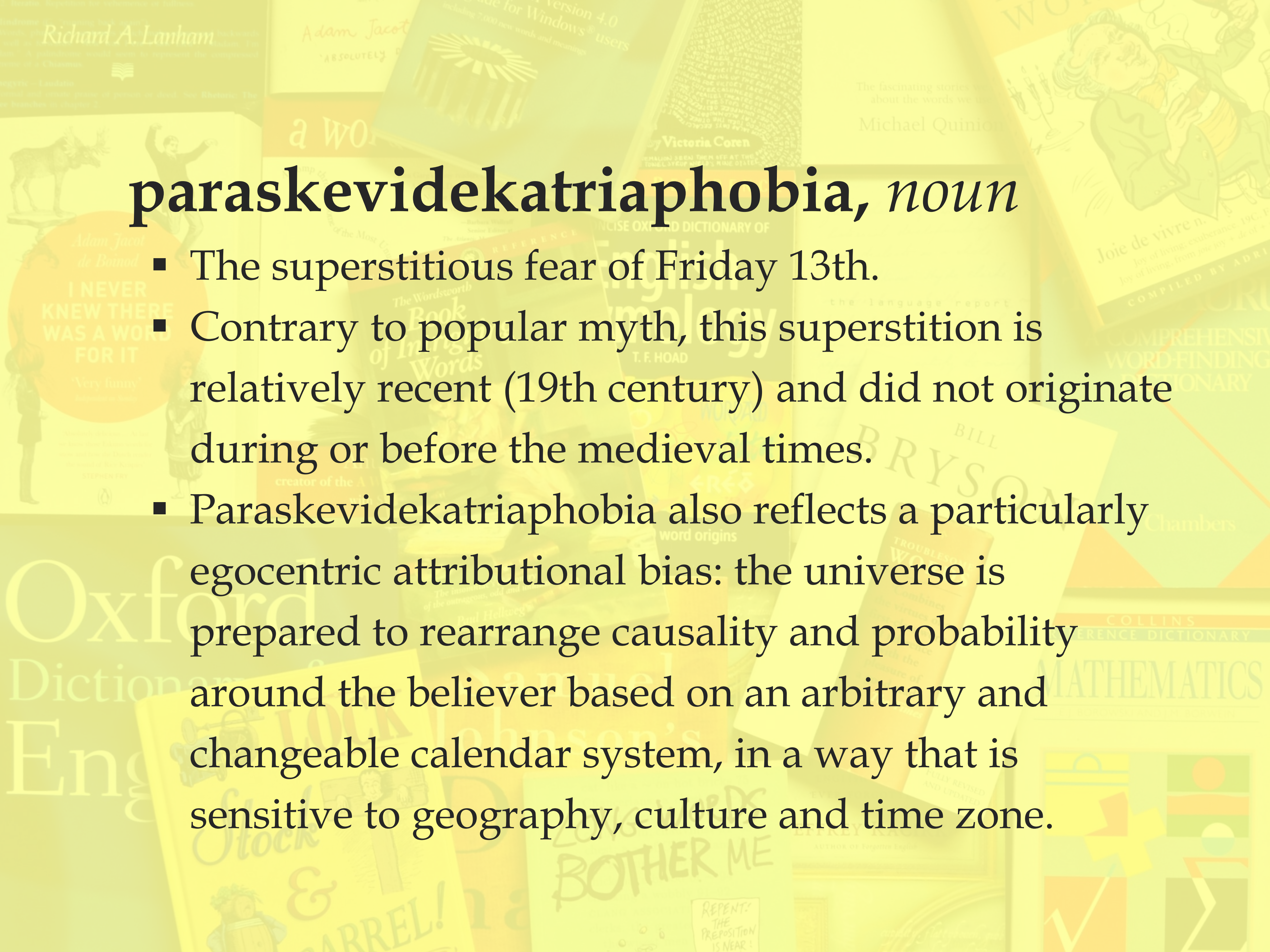
```
// Get the unique surnames in uppercase of the
// first 15 book authors that are 50 years old
// or older.

library.stream()
       .map(book -> book.getAuthor())
       .filter(author -> author.getAge() >= 50)
       .distinct()
       .limit(15)
       .map(Author::getSurname)
       .map(String::toUpperCase)
       .distinct()
       .collect(toList()))
```

Simple filters that can be arbitrarily chained are more easily re-used, and more robust, than almost any other kind of code.

Brandon Rhodes
http://rhodesmill.org/brandon/slides/2012-11-pyconca/

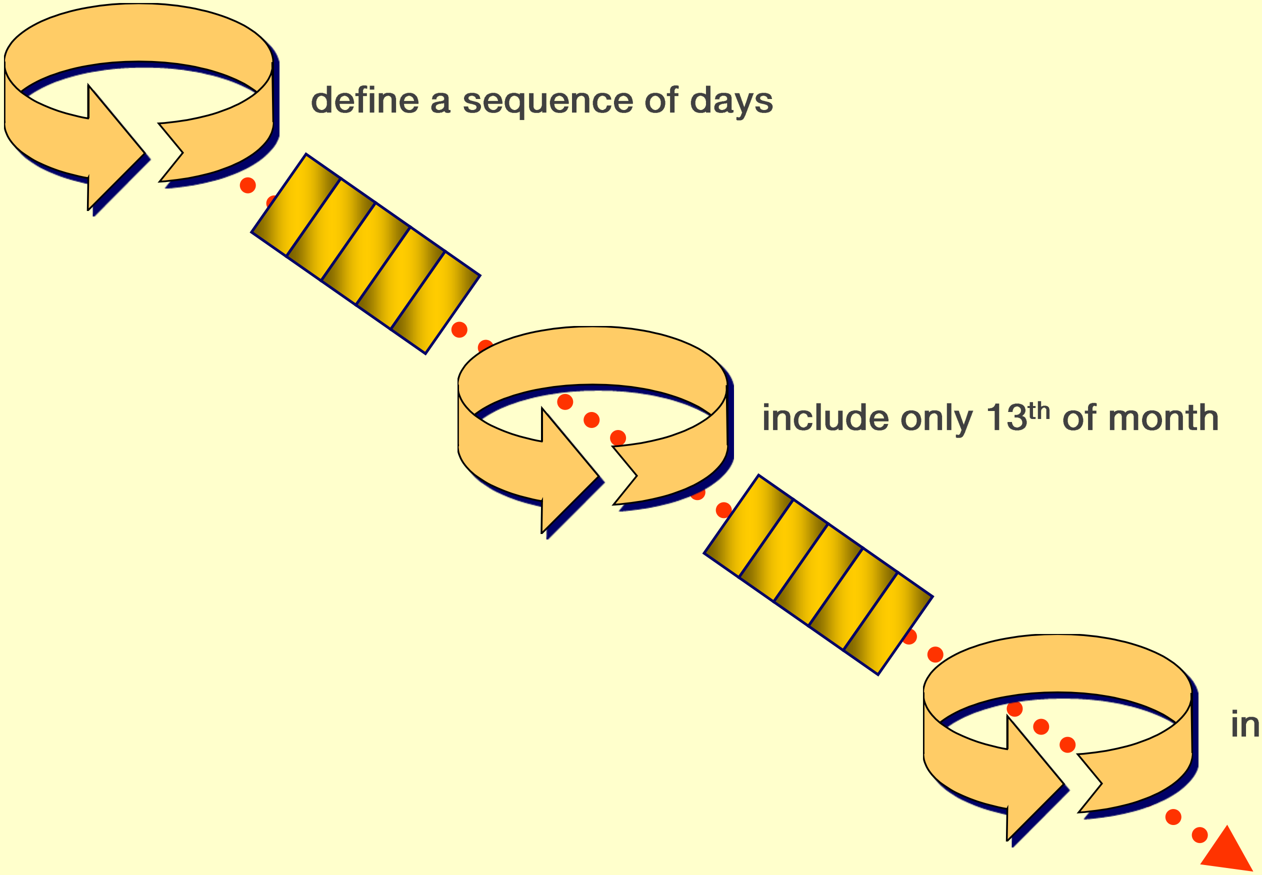
Richard A. Lanham
Adam Jacot de Boinod
I NEVER KNEW THERE WAS A WORD FOR IT
'Very funny'
Independent on Sunday
STEPHEN FRY
a word
A Romp through Some of the Most Unusual
Version 4.0
guide for Windows® users
including 7,000 new words and meanings
by Victoria Coren
The fascinating stories we
about the words we use
Michael Quinion
Joie de vivre n.
COMPILED BY ADRI
WORDSWORTH REFERENCE
The Wordsworth Book of Intriguing Words
The insomniac's dictionary of the outrageous, odd and unusual
Paul Hellweg
OXFORD REFERENCE
CONCISE OXFORD DICTIONARY OF
English Etymology
T. F. HOAD
word origins
the language report
A COMPREHENSIVE WORD-FINDING DICTIONARY
Chambers
BILL BRYSON
TROUBLESOME WORDS
'Combines the virtues of a first-class work of reference with the pleasure of a good read'
The Times
FULLY REVISED AND UPDATED
Oxford Dictionary of English
LOCK Stock & BARREL!
Samuel Johnson's
LONG WORDS BOTHER ME
REPENT! THE PREPOSITION IS NEAR!
JEFFREY KACIRK
AUTHOR OF Forgotten English
COLLINS REFERENCE DICTIONARY
MATHEMATICS
E.J. BOROWSKI AND J.M. BORWEIN

f / WordFriday


Richard A. Lanham
Adam Jacot de Boinod
I NEVER KNEW THERE WAS A WORD FOR IT
Victoria Coren
Michael Quinion
Joie de vivre n.
CONCISE OXFORD DICTIONARY OF English Etymology
T. F. HOAD
The Wordsworth Book of Intriguing Words
BILL BRYSON
Chambers
word origins
COLLINS REFERENCE DICTIONARY MATHEMATICS
Oxford Dictionary English
Johnson's
BOTHER ME
REPENT! THE PREPOSITION IS NEAR!

paraskevidekatriaphobia, *noun*

- The superstitious fear of Friday 13th.
- Contrary to popular myth, this superstition is relatively recent (19th century) and did not originate during or before the medieval times.
- Paraskevidekatriaphobia also reflects a particularly egocentric attributional bias: the universe is prepared to rearrange causality and probability around the believer based on an arbitrary and changeable calendar system, in a way that is sensitive to geography, culture and time zone.



define a sequence of days

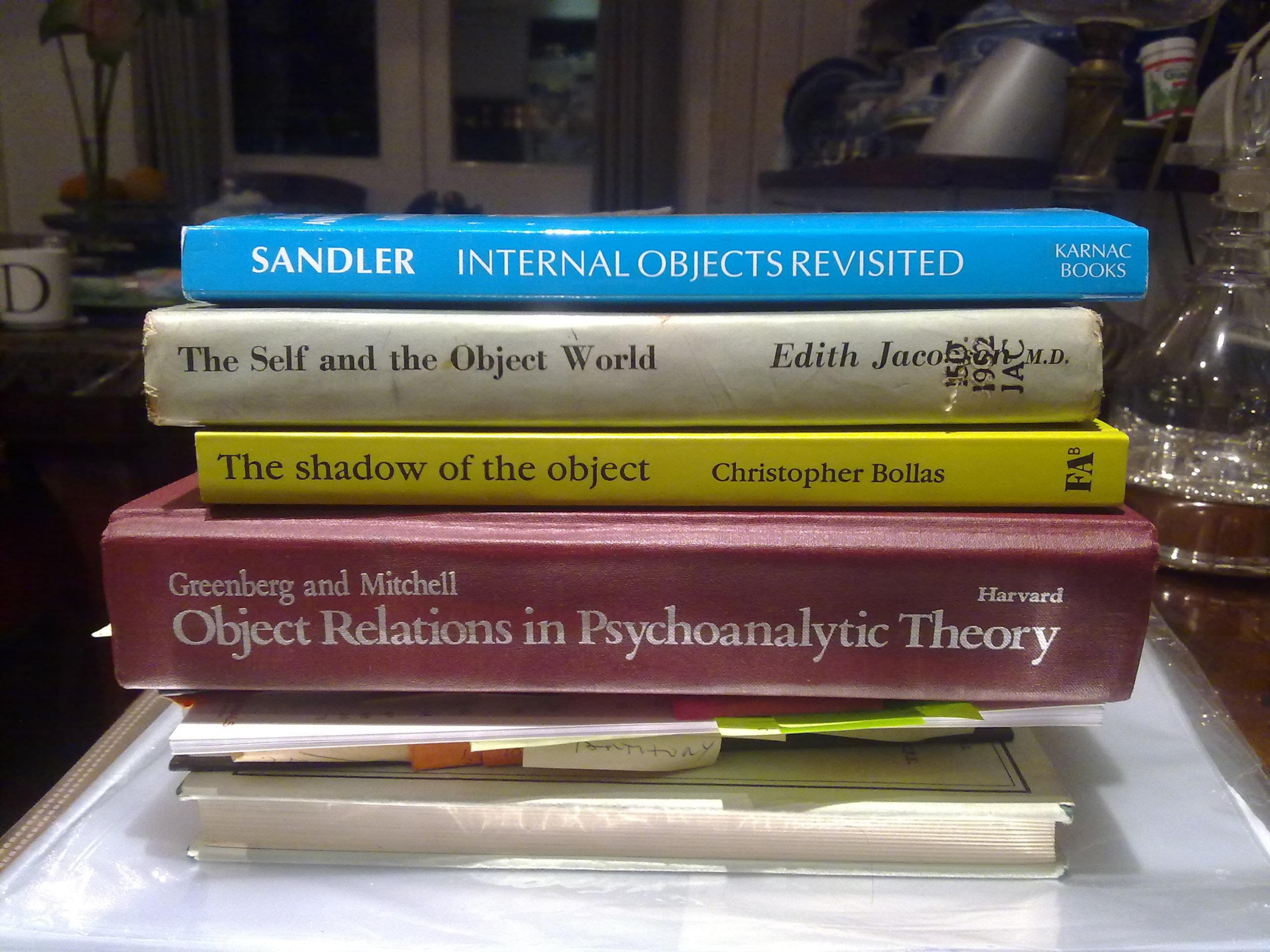
include only 13th of month

include only Fridays

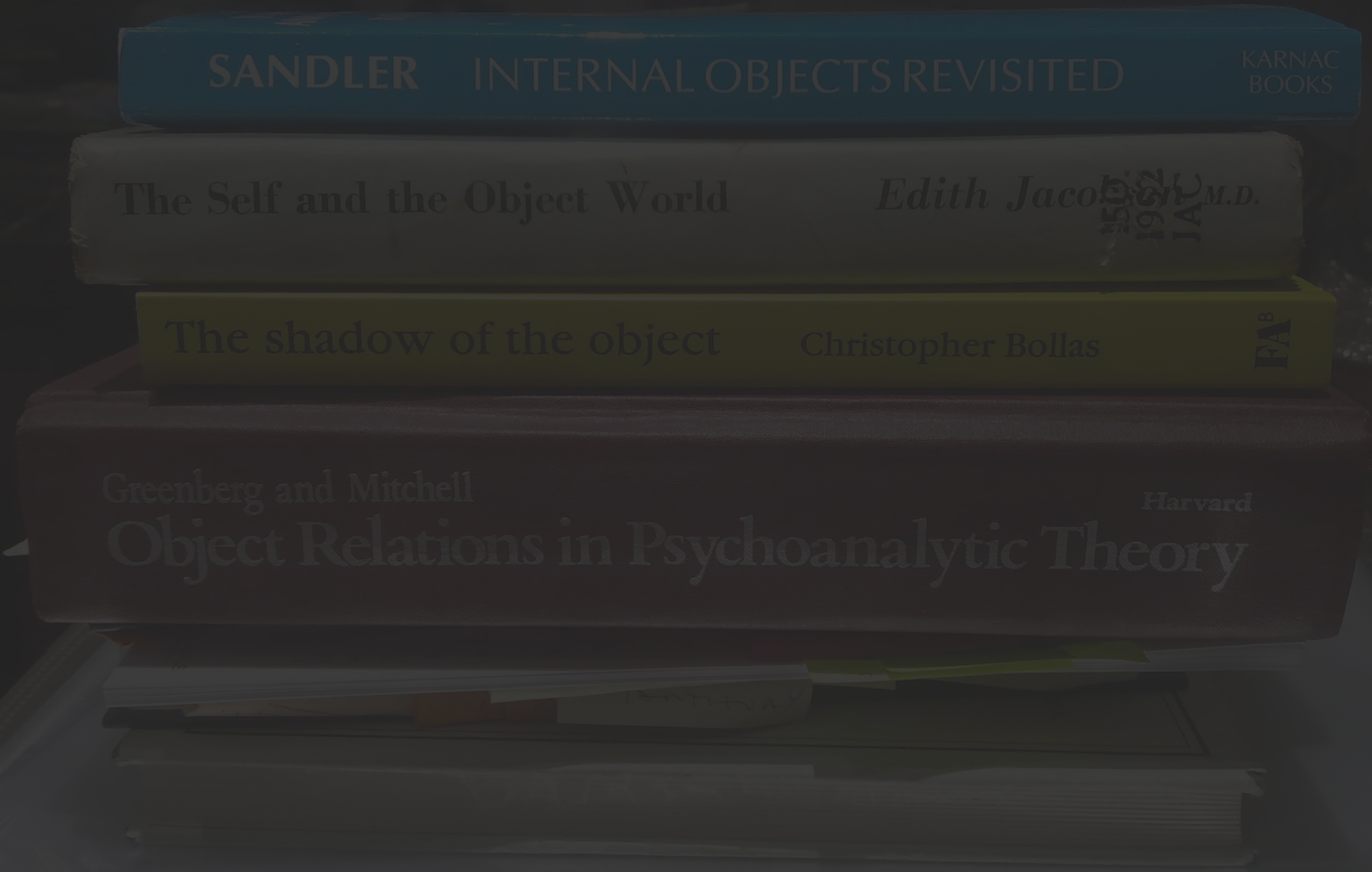
```
function NextFriday13thAfter($from) {
  (1..500) |
  %{ $from.AddDays($_) } |
  ?{ $_.Day -eq 13} |
  ?{ $_.DayOfWeek -eq [DayOfWeek]::Friday } |
  select –first 1
}
```

```
$ ./roman 42
XLII
$ cat roman
printf %$1s |
tr ' ' 'I' |
sed '
    s/IIIII/V/g
    s/IIII/IV/
    s/VV/X/g
    s/VIV/IX/
    s/XXXXX/L/g
    s/XXXX/XL/
    s/LL/C/g
    s/LXL/XC/
    s/CCCCC/D/g
    s/CCCC/CD/
    s/DD/M/g
    s/DCD/CM/
'
echo
```

```
$ ./roman 42
XLII
$ cat roman
printf %$1s |
tr ' ' 'I' |
sed s/IIIII/V/g |
sed s/IIII/IV/ |
sed s/VV/X/g |
sed s/VIV/IX/ |
sed s/XXXXX/L/g |
sed s/XXXX/XL/ |
sed s/LL/C/g |
sed s/LXL/XC/ |
sed s/CCCCC/D/g |
sed s/CCCC/CD/ |
sed s/DD/M/g |
sed s/DCD/CM/
echo
```

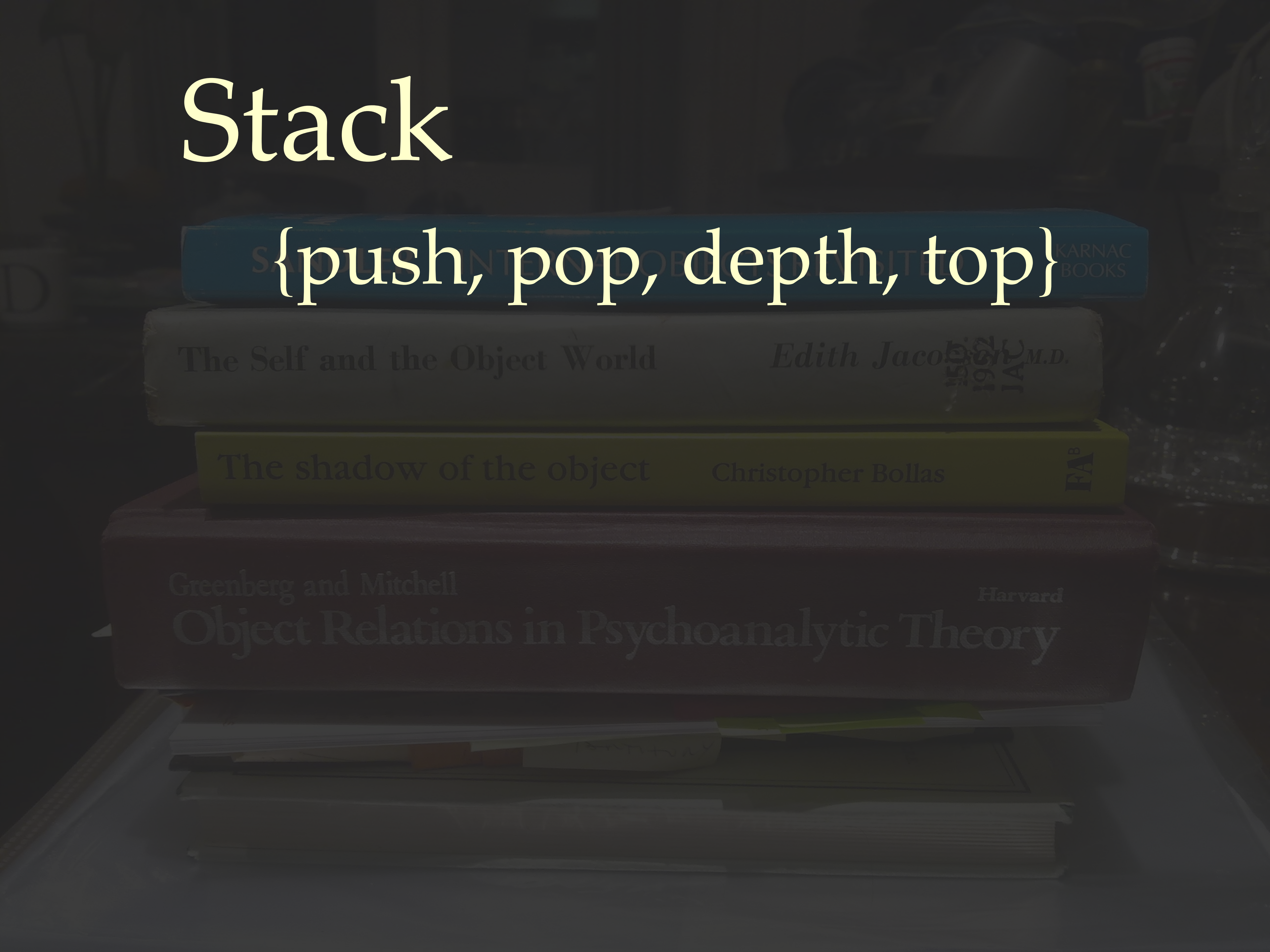

SANDLER INTERNAL OBJECTS REVISITED
KARNAC BOOKS
The Self and the Object World
Edith Jacobson M.D.
150 1962 JAC
The shadow of the object
Christopher Bollas
FAB
Greenberg and Mitchell
Object Relations in Psychoanalytic Theory
Harvard

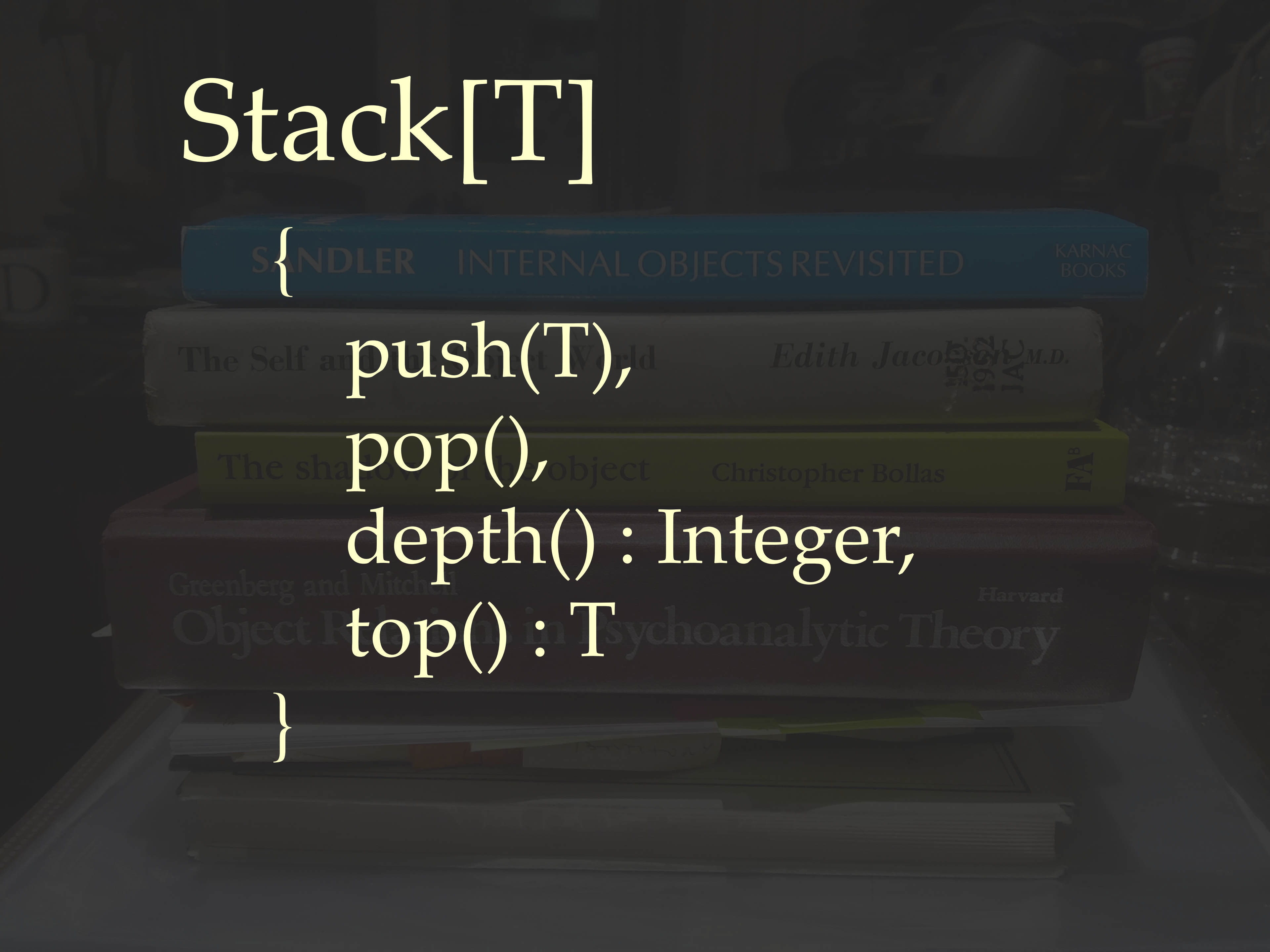
Stack


SANDLER INTERNAL OBJECTS REVISITED
KARNAC BOOKS
The Self and the Object World
Edith Jacobson, M.D.
The shadow of the object
Christopher Bollas
FAB
Greenberg and Mitchell
Object Relations in Psychoanalytic Theory
Harvard

Stack

{push, pop, depth, top}





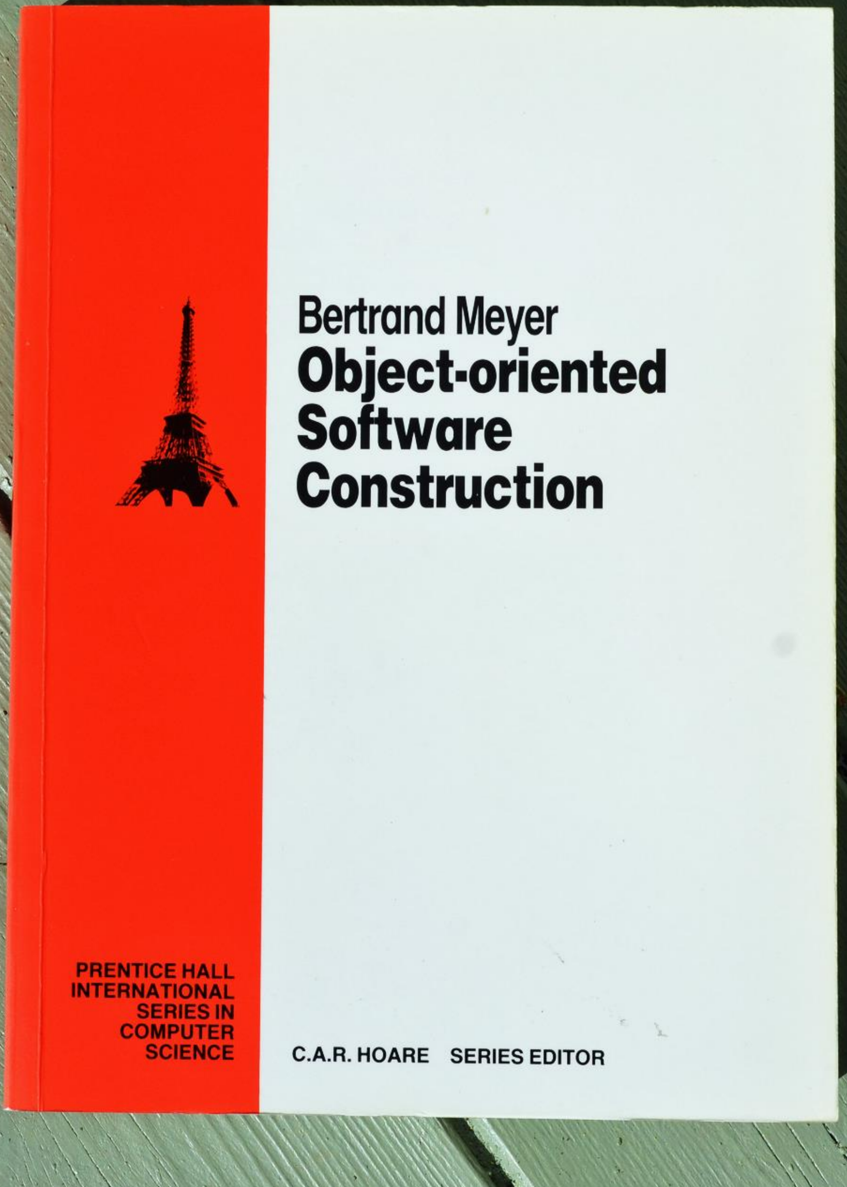
Stack[T]

```
{
    push(T),
    pop(),
    depth() : Integer,
    top() : T
}
```

An interface is a contract to deliver a certain amount of service.

Clients of the interface depend on the contract, which is usually documented in the interface specification.

Butler W Lampson
"Hints for Computer System Design"

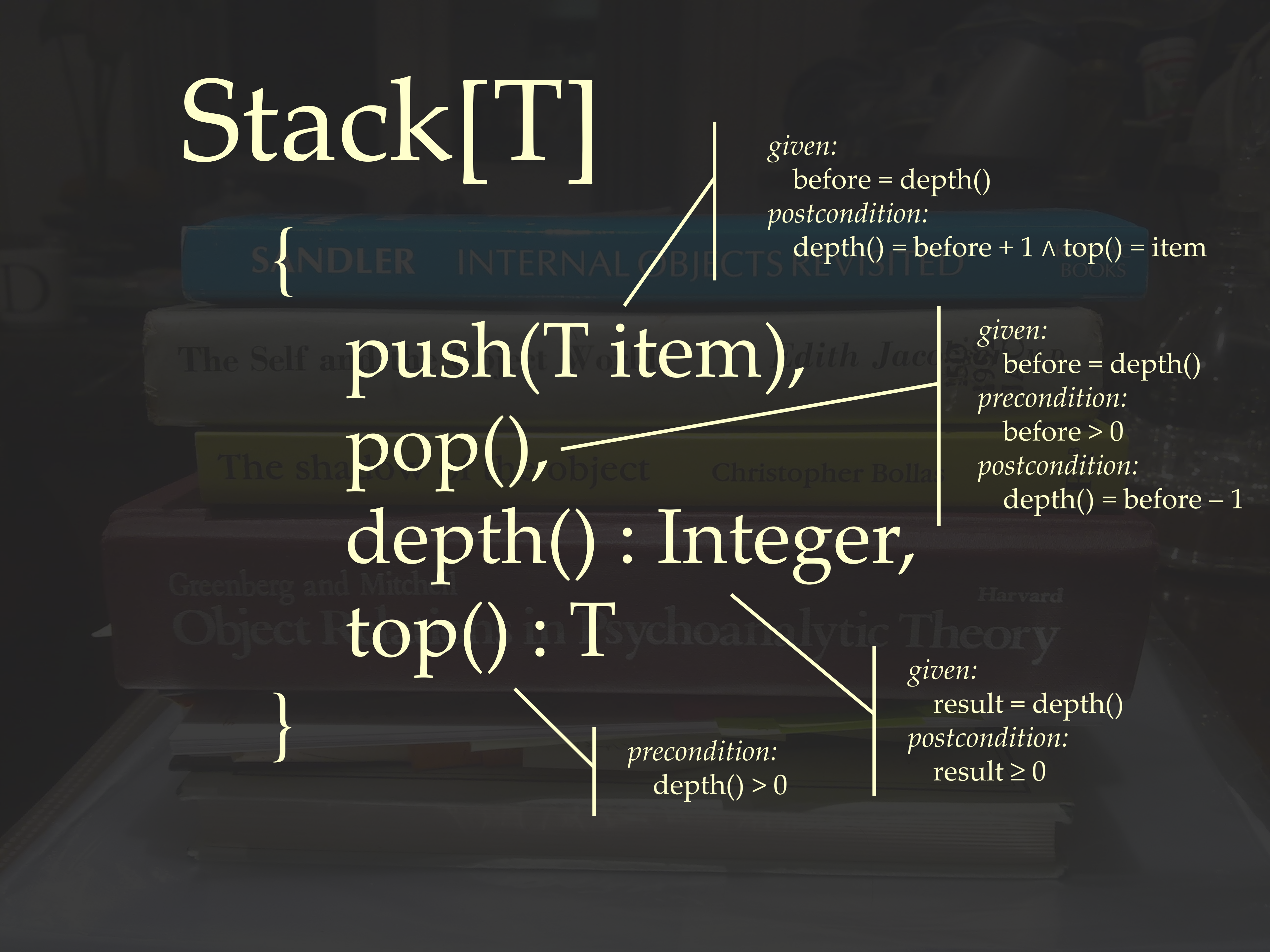


Bertrand Meyer

Object-oriented Software Construction

PRENTICE HALL
INTERNATIONAL
SERIES IN
COMPUTER
SCIENCE

C.A.R. HOARE SERIES EDITOR


SANDLER INTERNAL OBJECTS REVISITED
BOOKS
The Self and the Object World
Edith Jacobson
The shadow of the object
Christopher Bollas
Greenberg and Mitchell
Object Relations in Psychoanalytic Theory
Harvard

Stack[T]

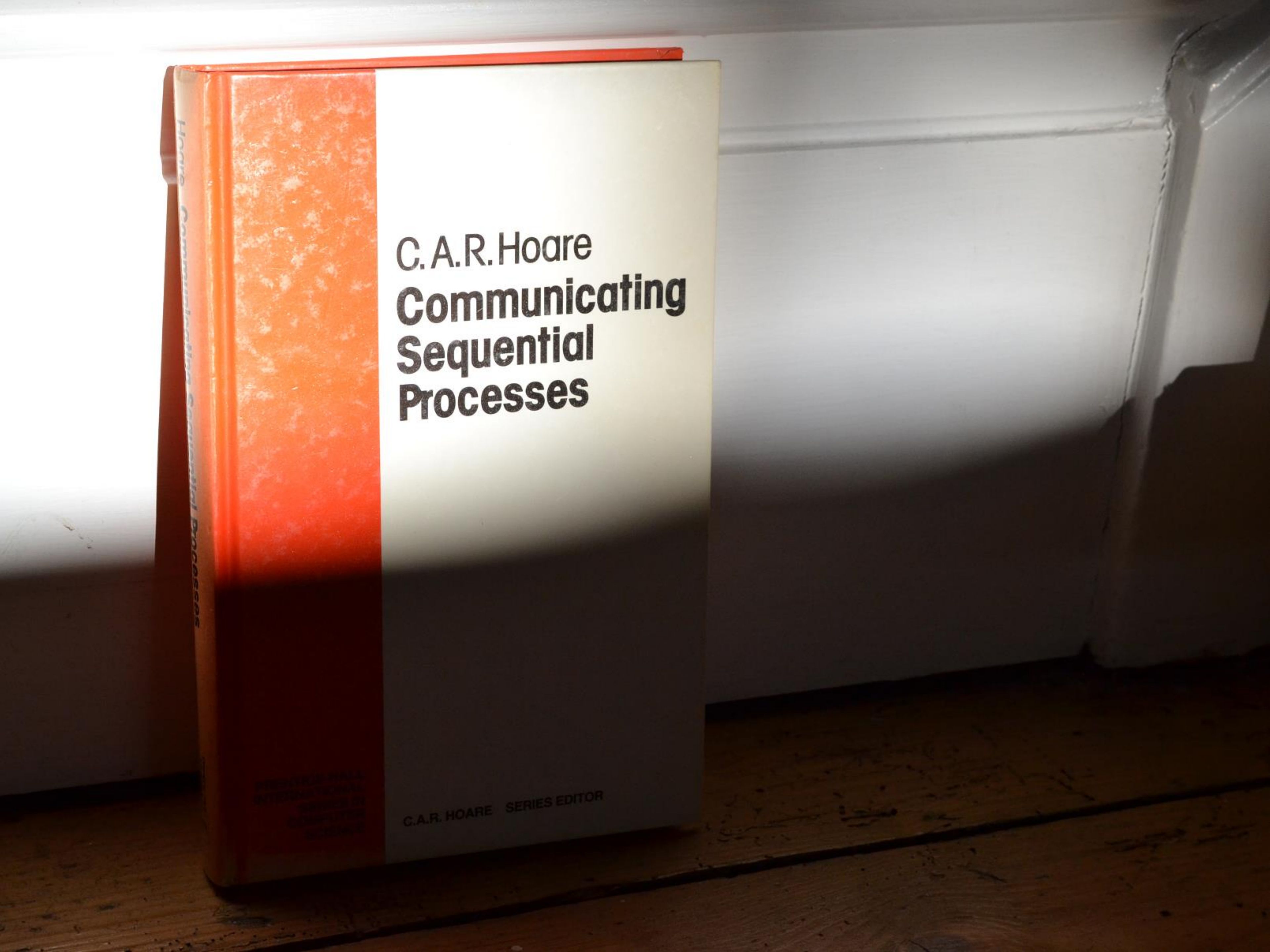
```
{
    push(T item),
    pop(),
    depth() : Integer,
    top() : T
}
```

given:
before = depth()
postcondition:
depth() = before + 1 ∧ top() = item

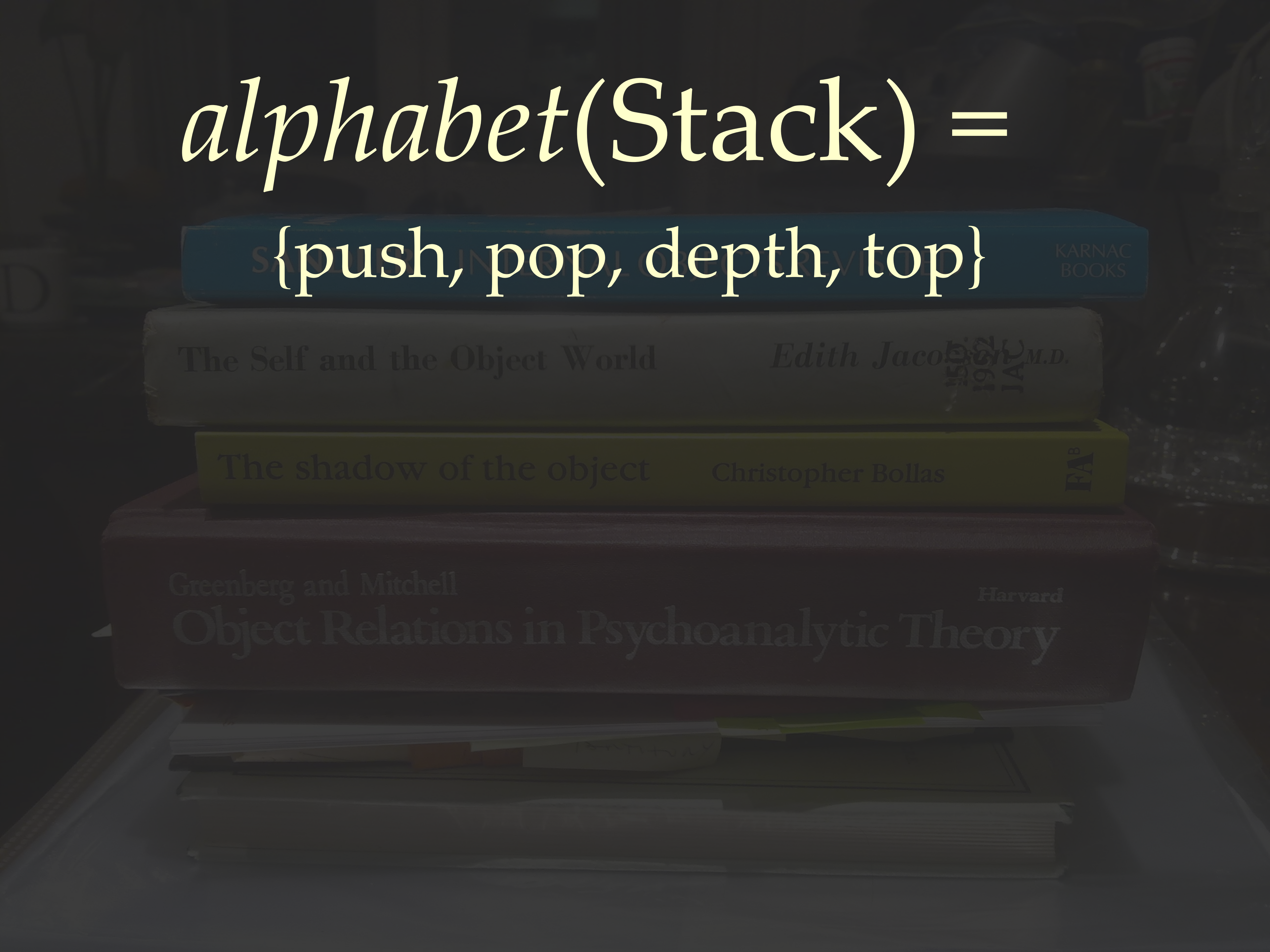
given:
before = depth()
precondition:
before > 0
postcondition:
depth() = before – 1

given:
result = depth()
postcondition:
result ≥ 0

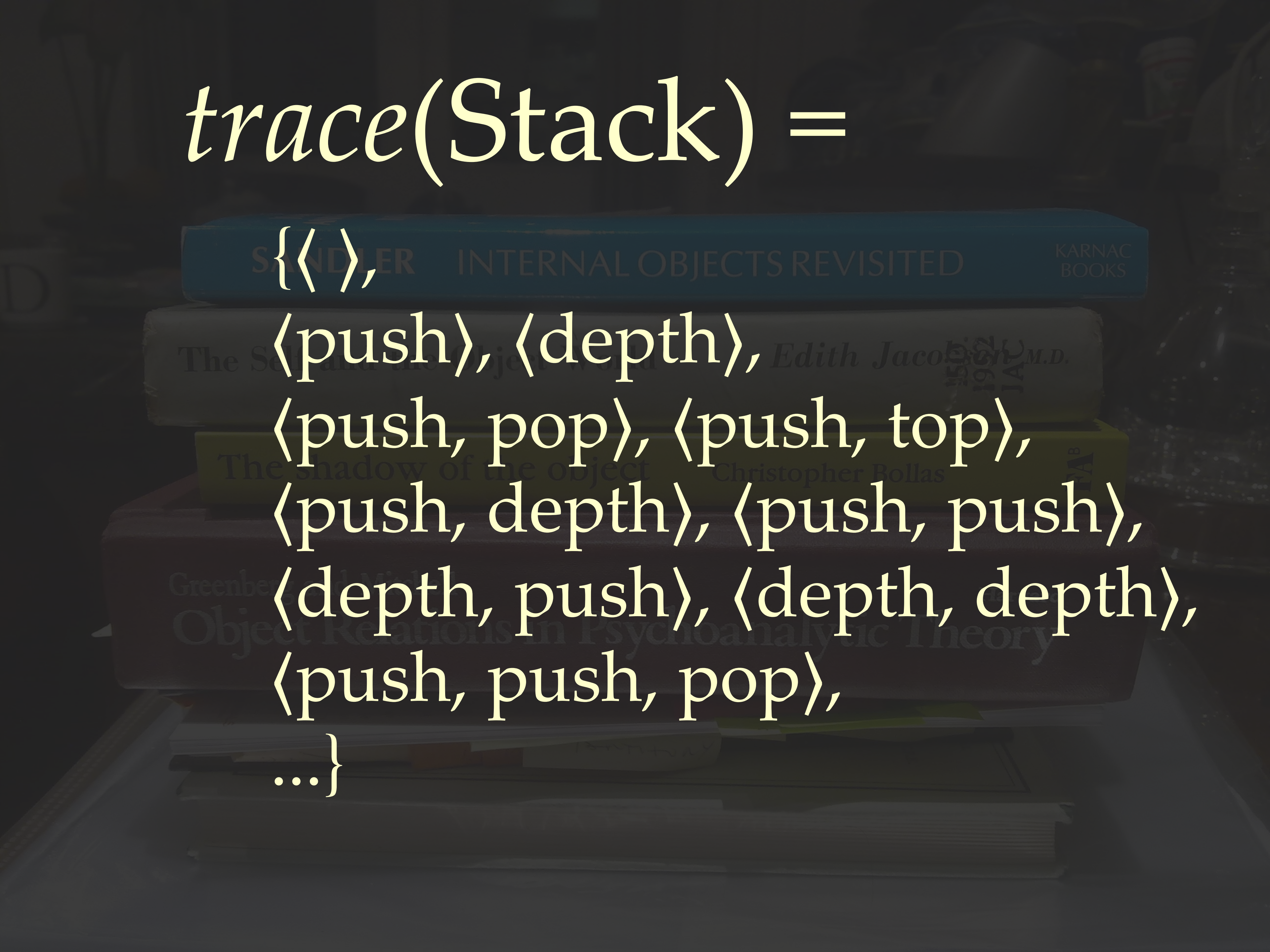
precondition:
depth() > 0



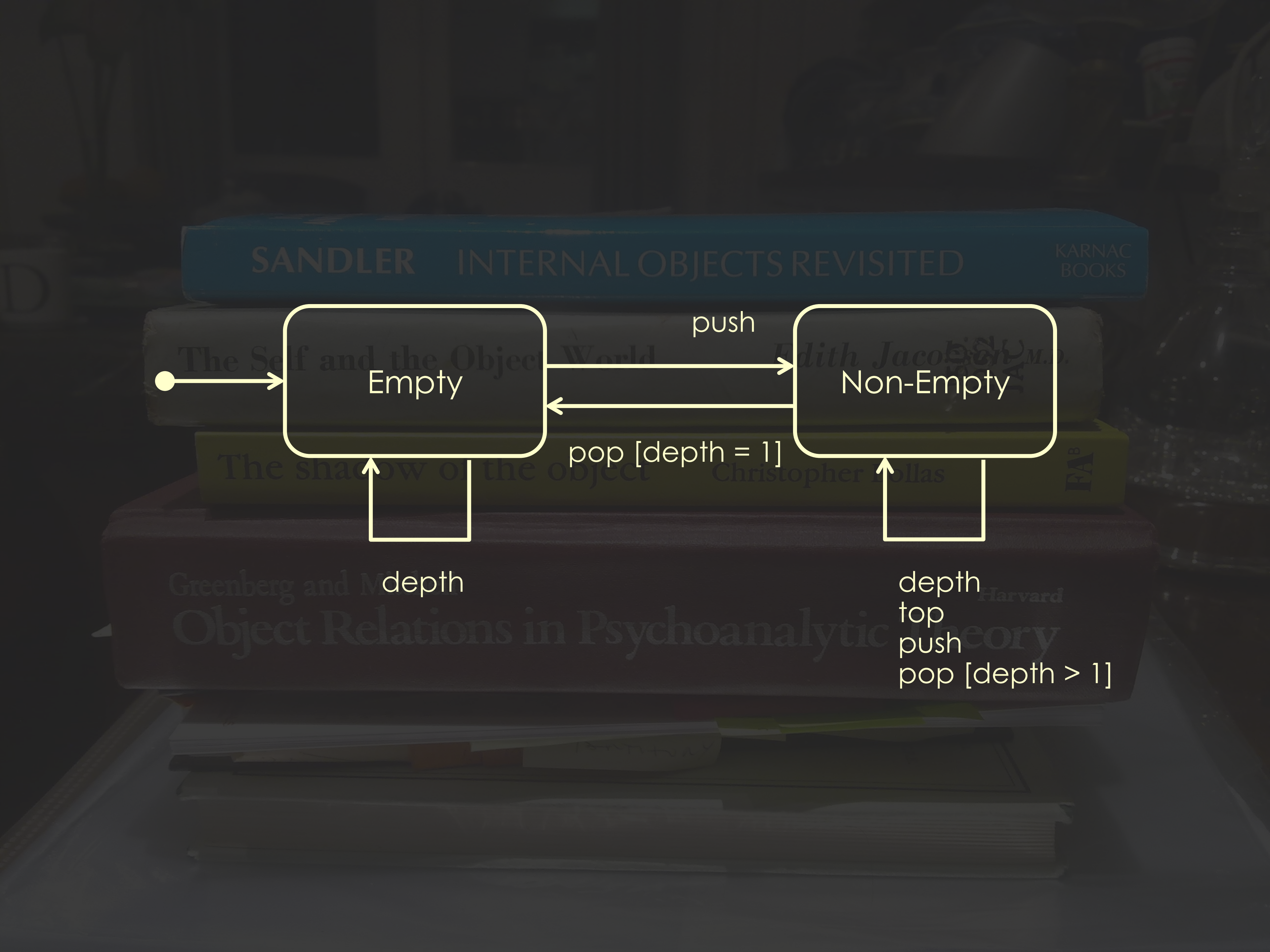
C. A. R. Hoare
Communicating Sequential Processes
C.A.R. HOARE SERIES EDITOR
Hoare Communicating Sequential Processes



alphabet(Stack) =
{push, pop, depth, top}


SANDLER INTERNAL OBJECTS REVISITED
KARNAC BOOKS
Edith Jacobson M.D.
The shadow of the object
Christopher Bollas
Object Relations in Psychoanalytic Theory

$$
\begin{aligned}
&\mathit{trace}(\text{Stack}) = \\
&\quad \{\langle\,\rangle, \\
&\quad \langle \text{push}\rangle, \langle \text{depth}\rangle, \\
&\quad \langle \text{push, pop}\rangle, \langle \text{push, top}\rangle, \\
&\quad \langle \text{push, depth}\rangle, \langle \text{push, push}\rangle, \\
&\quad \langle \text{depth, push}\rangle, \langle \text{depth, depth}\rangle, \\
&\quad \langle \text{push, push, pop}\rangle, \\
&\quad \ldots\}
\end{aligned}
$$



Empty

Non-Empty

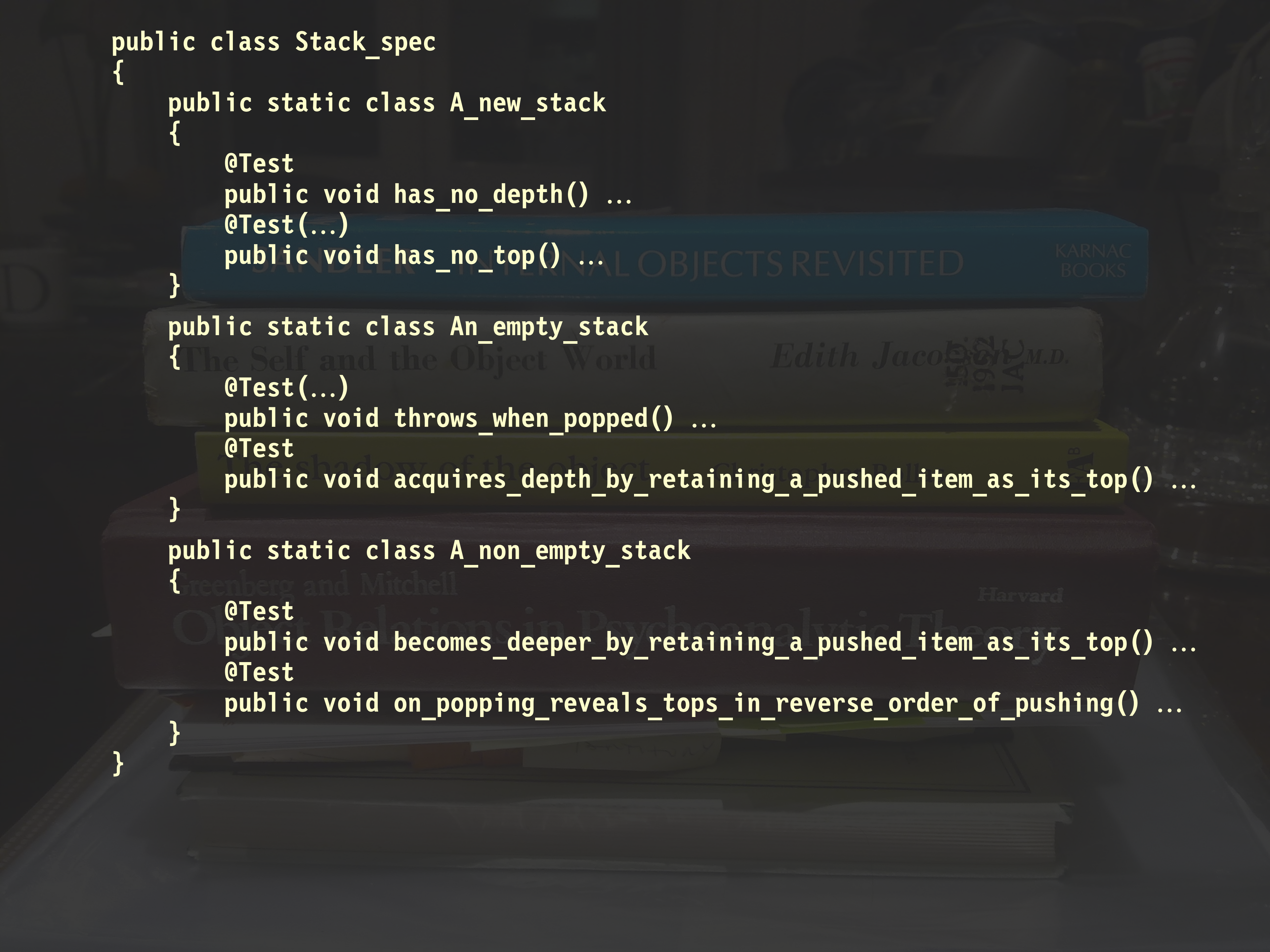
push

pop [depth = 1]

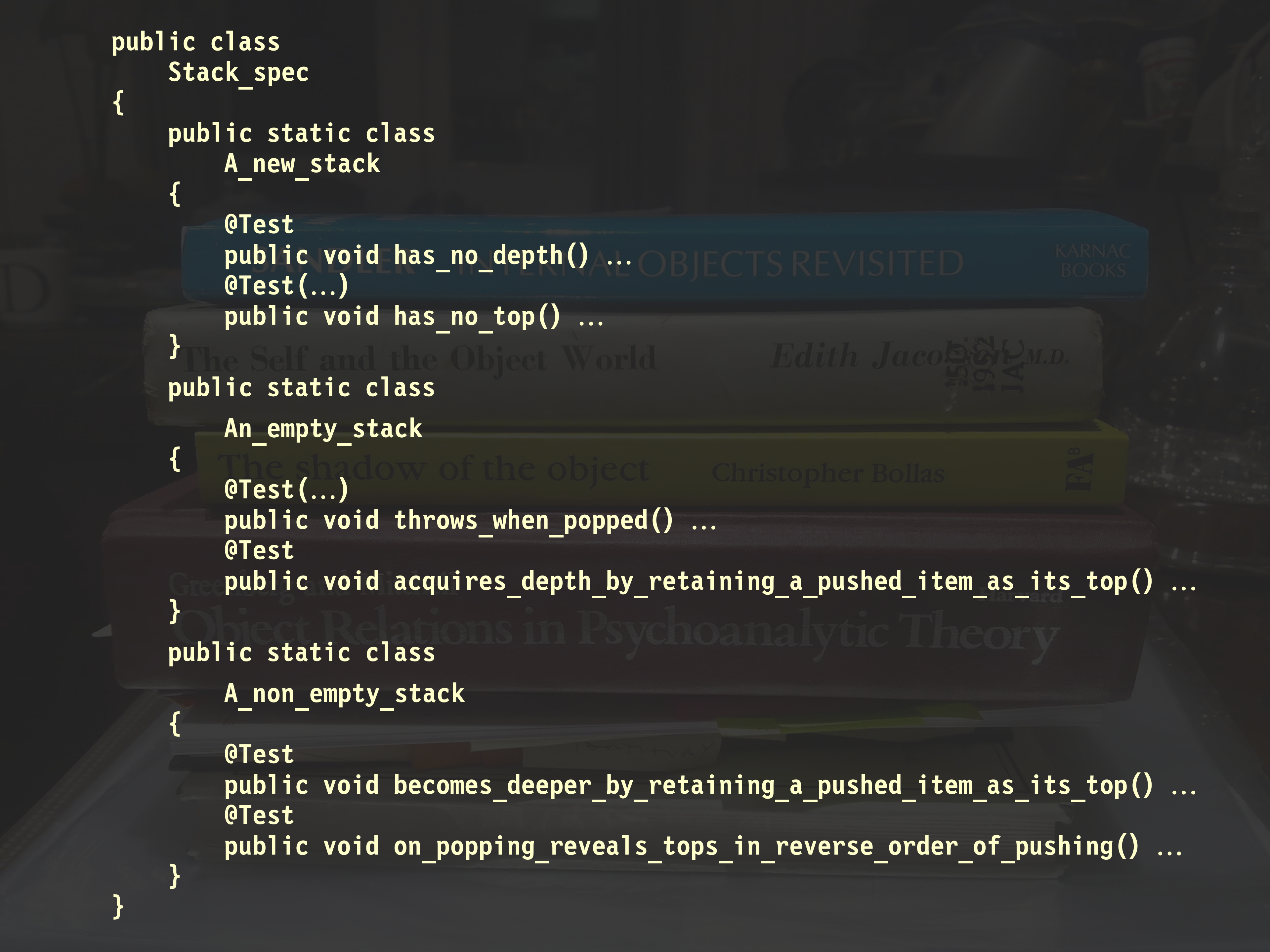
depth

depth
top
push
pop [depth > 1]

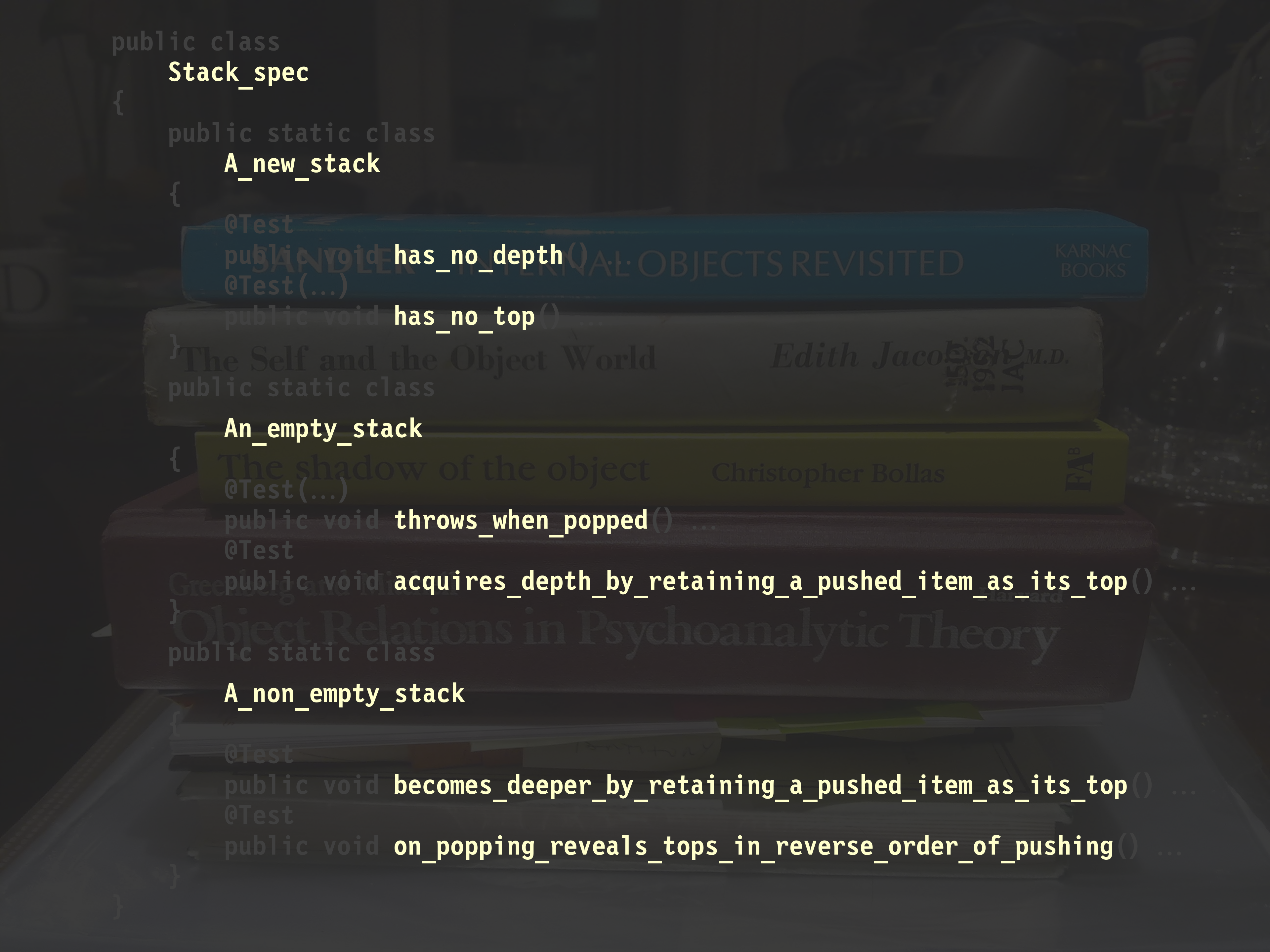
SANDLER INTERNAL OBJECTS REVISITED
KARNAC BOOKS
The Self and the Object World
Edith Jacobson M.D.
The shadow of the object
Christopher Bollas
Greenberg and Mitchell
Object Relations in Psychoanalytic Theory
Harvard


SANDLER INTERNAL OBJECTS REVISITED
KARNAC BOOKS
The Self and the Object World
Edith Jacobson M.D.
The shadow of the object
Christopher Bollas
Greenberg and Mitchell
Object Relations in Psychoanalytic Theory
Harvard

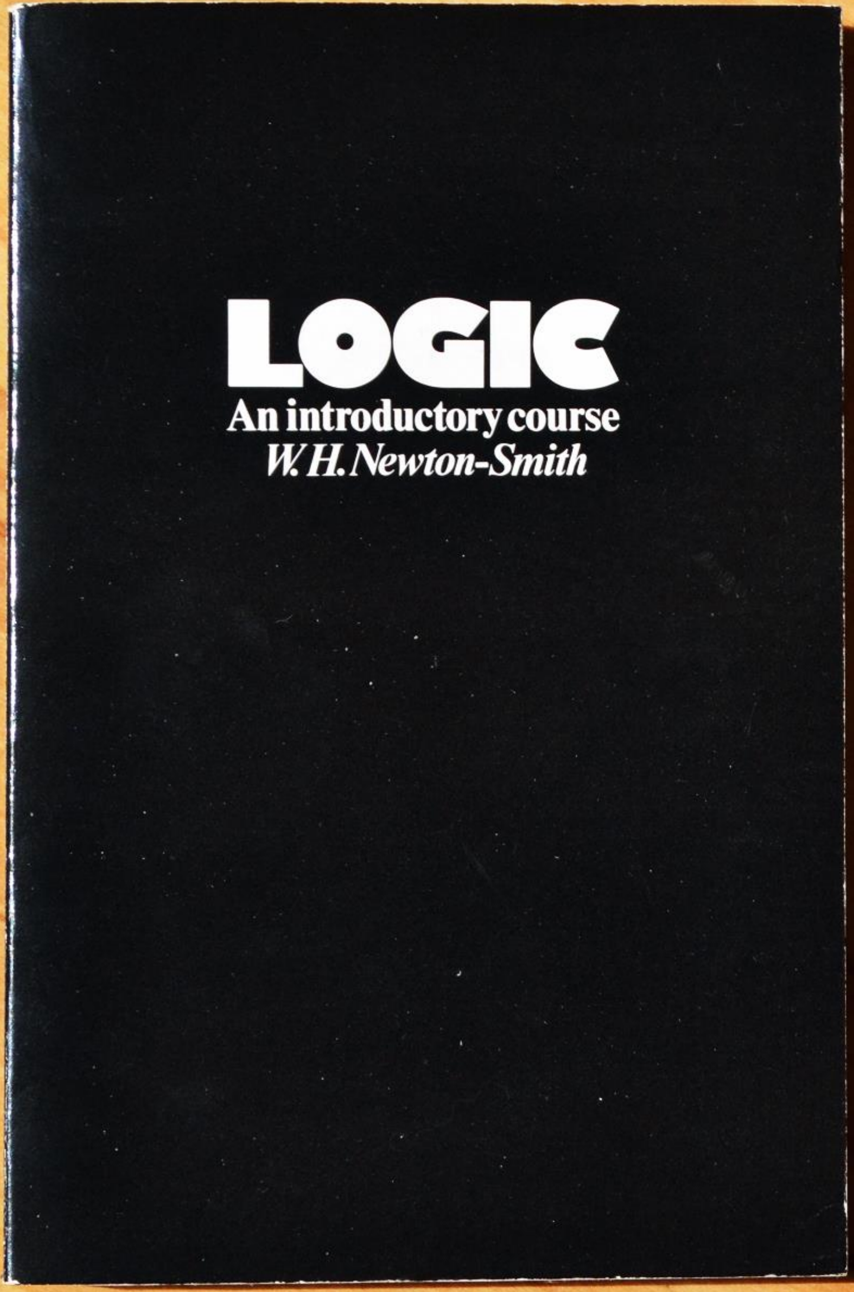
```
public class Stack_spec
{
    public static class A_new_stack
    {
        @Test
        public void has_no_depth() …
        @Test(…)
        public void has_no_top() …
    }
    public static class An_empty_stack
    {
        @Test(…)
        public void throws_when_popped() …
        @Test
        public void acquires_depth_by_retaining_a_pushed_item_as_its_top() …
    }
    public static class A_non_empty_stack
    {
        @Test
        public void becomes_deeper_by_retaining_a_pushed_item_as_its_top() …
        @Test
        public void on_popping_reveals_tops_in_reverse_order_of_pushing() …
    }
}
```



```
public class
    Stack_spec
{
    public static class
        A_new_stack
    {
        @Test
        public void has_no_depth() …
        @Test(…)
        public void has_no_top() …
    }
    public static class
        An_empty_stack
    {
        @Test(…)
        public void throws_when_popped() …
        @Test
        public void acquires_depth_by_retaining_a_pushed_item_as_its_top() …
    }
    public static class
        A_non_empty_stack
    {
        @Test
        public void becomes_deeper_by_retaining_a_pushed_item_as_its_top() …
        @Test
        public void on_popping_reveals_tops_in_reverse_order_of_pushing() …
    }
}
```



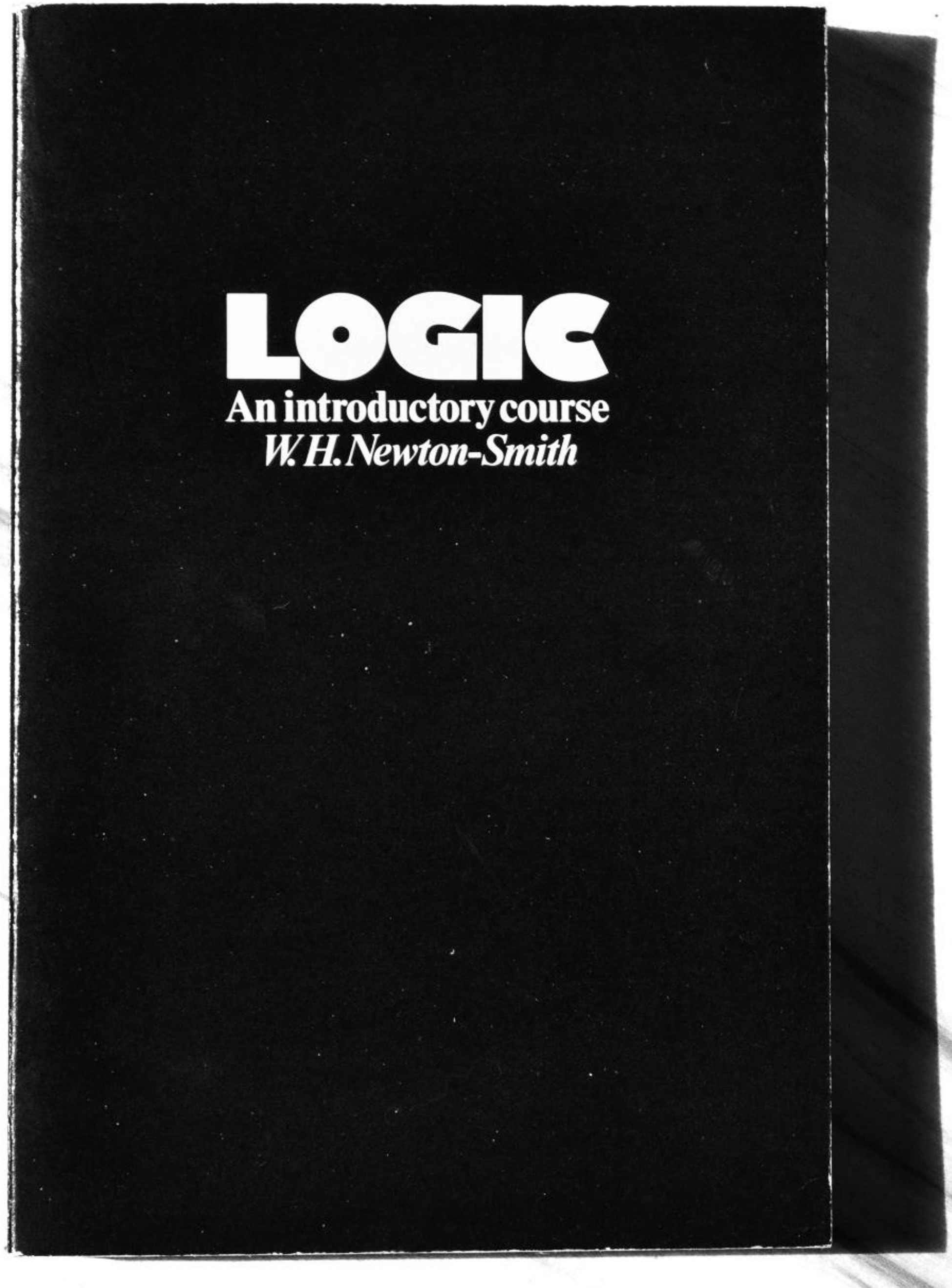
```
public class
    Stack_spec
{
    public static class
        A_new_stack
    {
        @Test
        public void has_no_depth() ...
        @Test(...)
        public void has_no_top() ...
    }
    public static class
        An_empty_stack
    {
        @Test(...)
        public void throws_when_popped() ...
        @Test
        public void acquires_depth_by_retaining_a_pushed_item_as_its_top() ...
    }
    public static class
        A_non_empty_stack
    {
        @Test
        public void becomes_deeper_by_retaining_a_pushed_item_as_its_top() ...
        @Test
        public void on_popping_reveals_tops_in_reverse_order_of_pushing() ...
    }
}
```



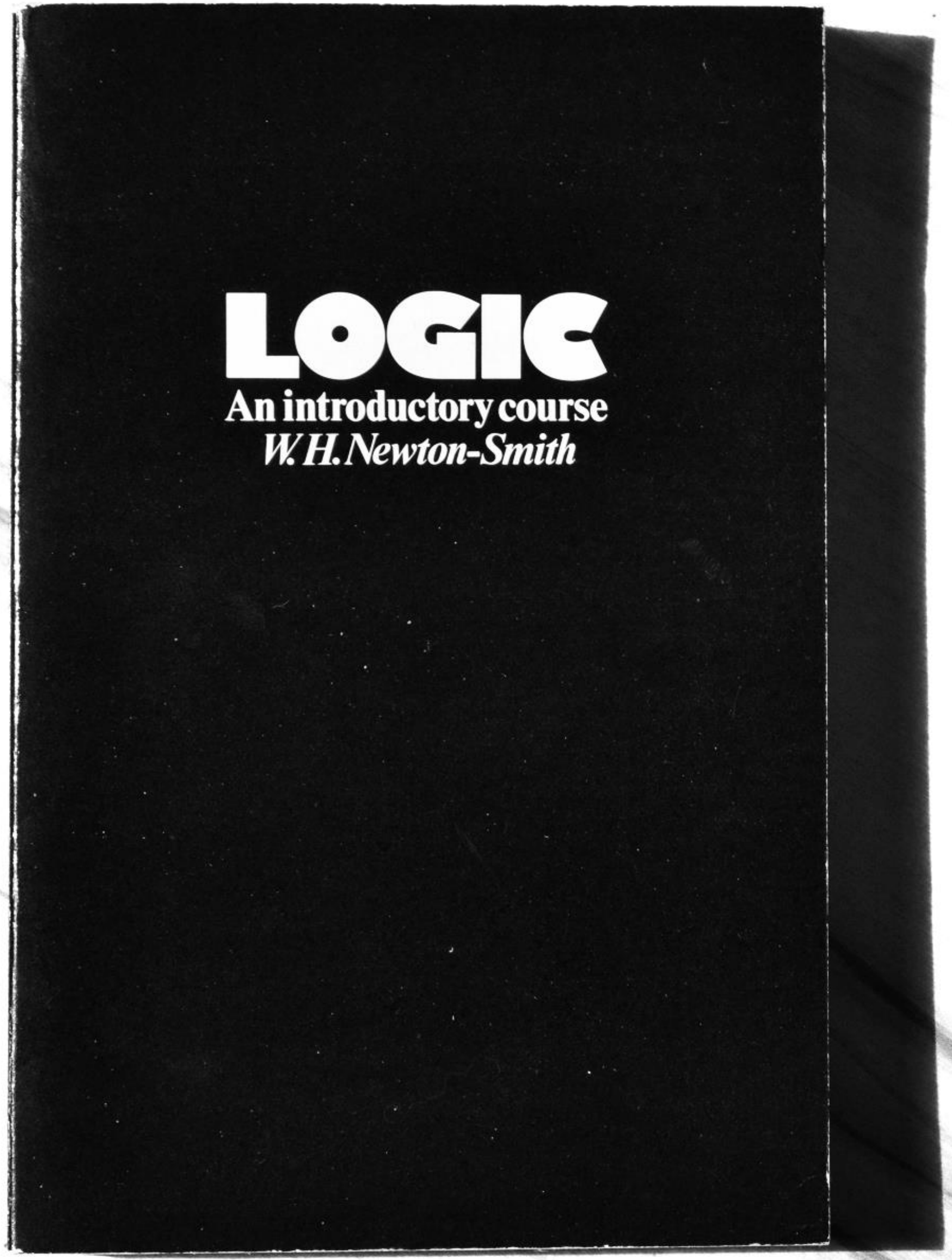
LOGIC

An introductory course

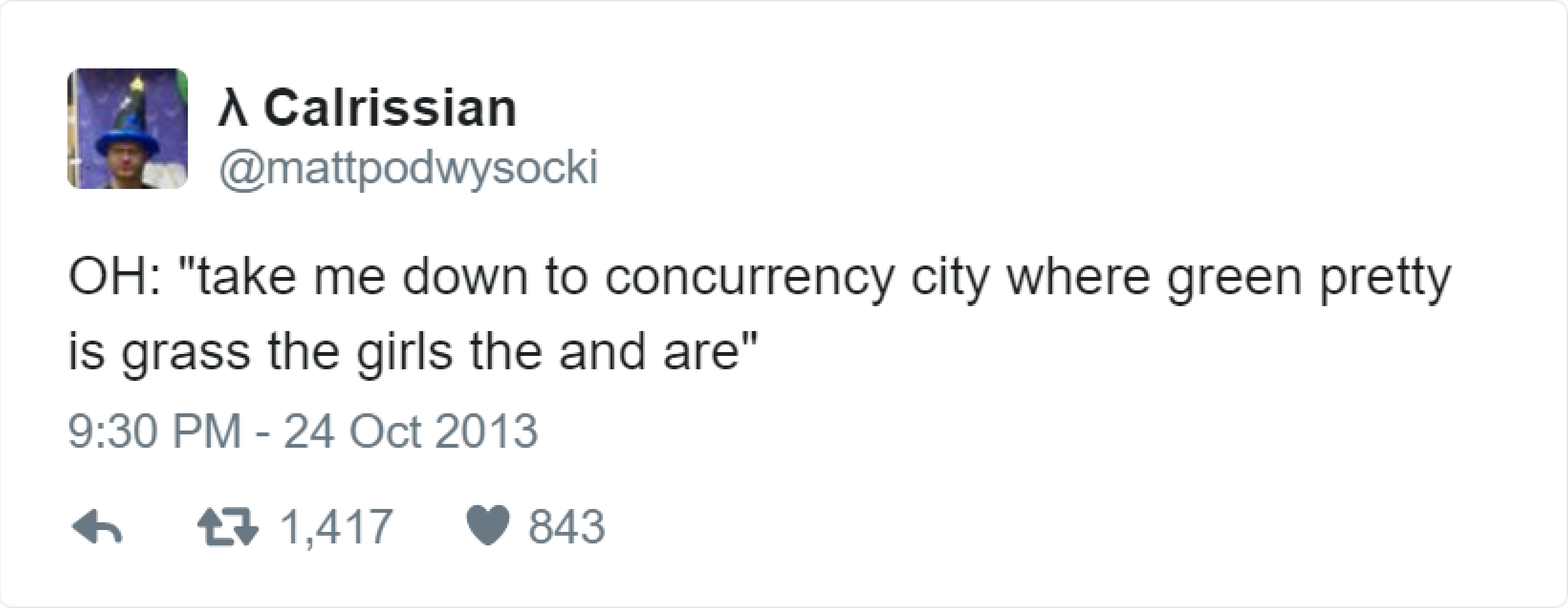
W. H. Newton-Smith



Propositions are vehicles for stating how things are or might be.



Thus only indicative sentences which it makes sense to think of as being true or as being false are capable of expressing propositions.



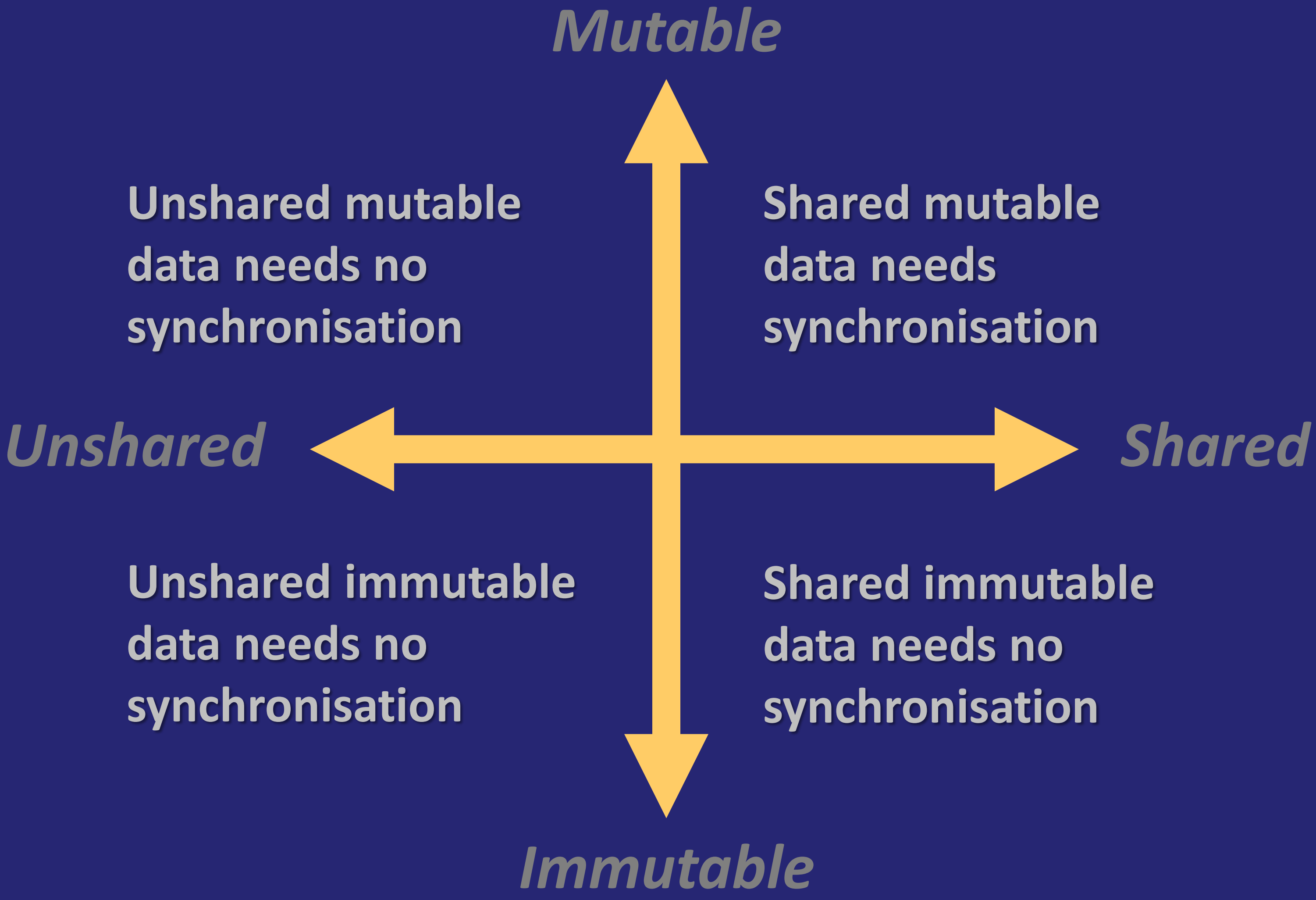
λ Calrissian
@mattpodwysocki

OH: "take me down to concurrency city where green pretty is grass the girls the and are"

9:30 PM - 24 Oct 2013

↩ ⟲ 1,417 ♥ 843

https://twitter.com/mattpodwysocki/status/393474697699921921



Mutable

Unshared mutable data needs no synchronisation

Shared mutable data needs synchronisation

Unshared

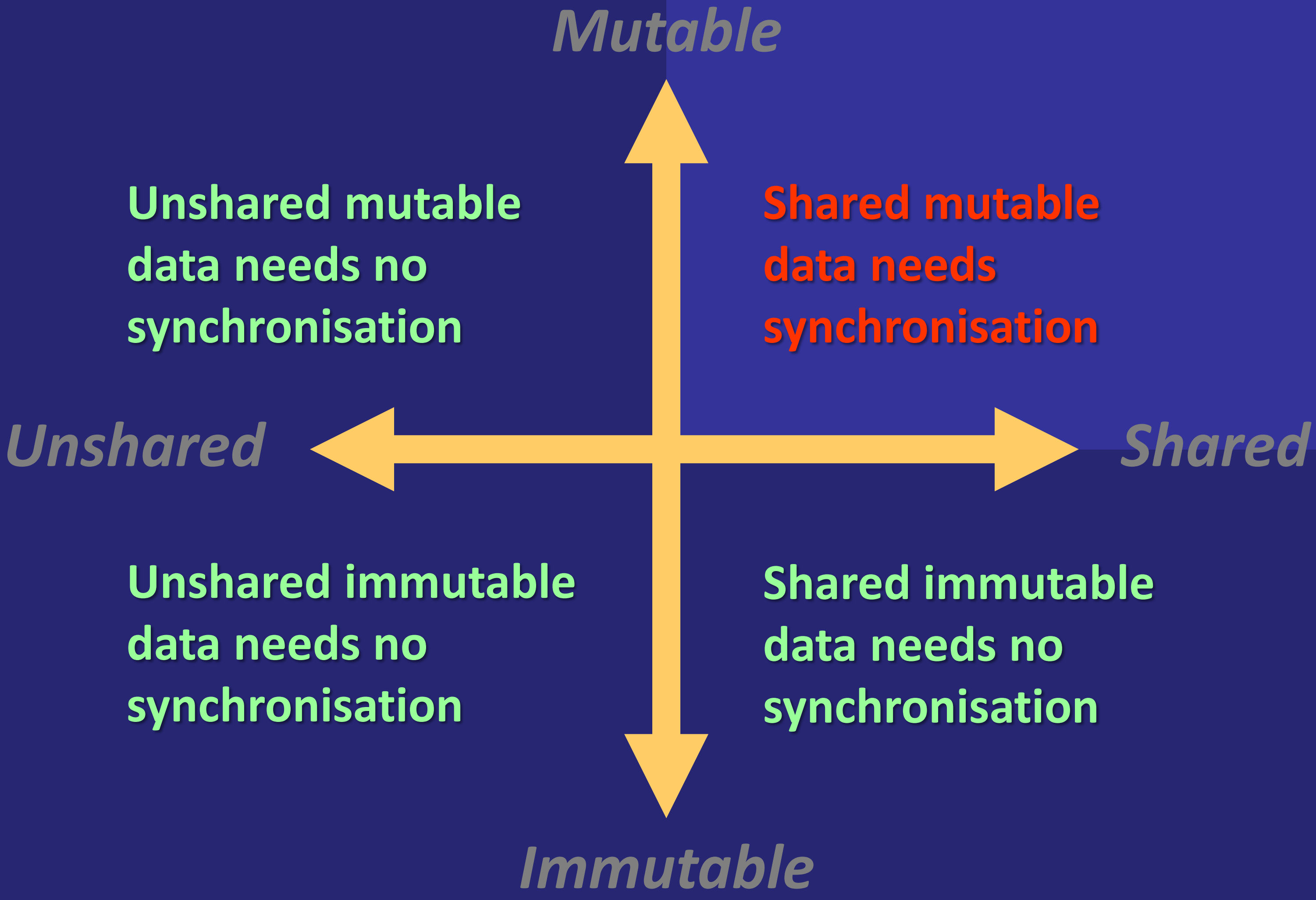
Shared

Unshared immutable data needs no synchronisation

Shared immutable data needs no synchronisation

Immutable

The Synchronisation Quadrant



Unshared mutable data needs no synchronisation

Shared mutable data needs synchronisation

Unshared immutable data needs no synchronisation

Shared immutable data needs no synchronisation

Article development led by queue.acm.org

DOI:10.1145/2844112

We need it, we can afford it, and the time is now.

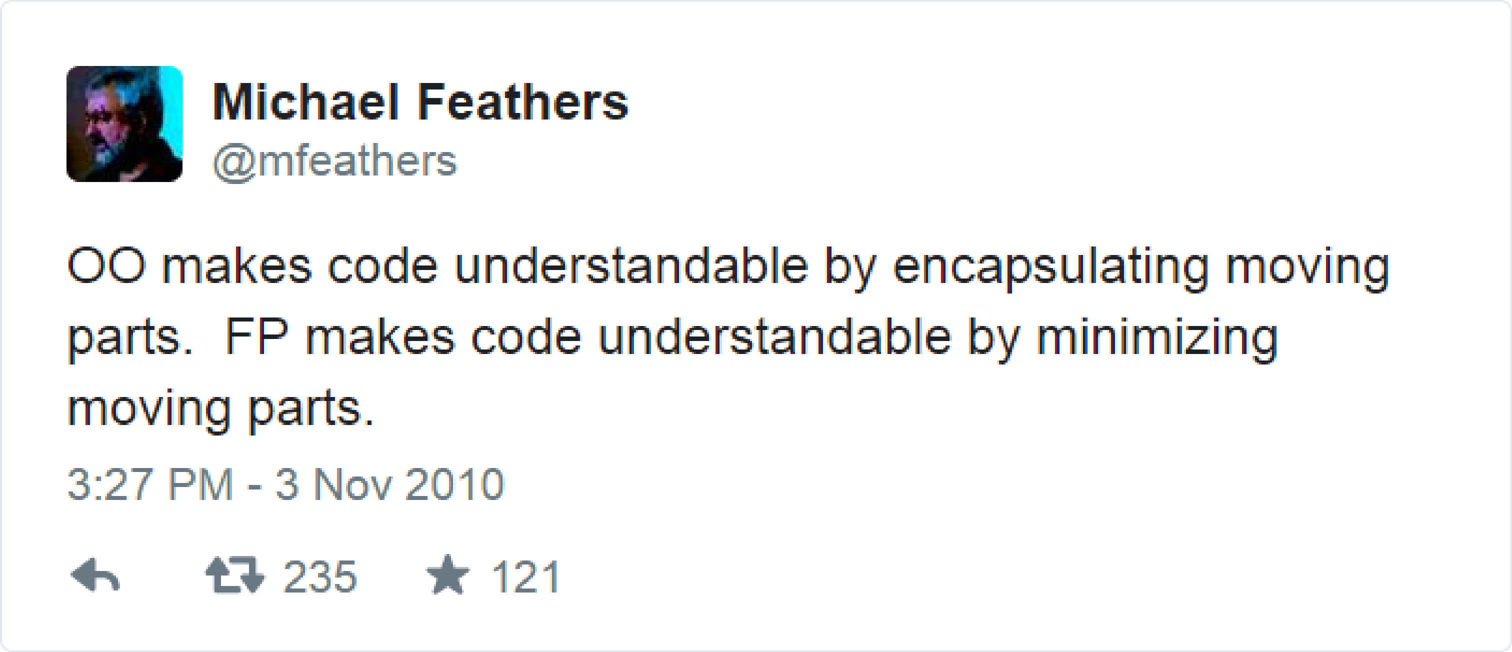
BY PAT HELLAND

Immutability Changes Everything

latches has become harder latch latency loses lots of opportunities. Keeping copies of lots of data is now afforda and one payoff is reduced coordinat challenges.

Storage is increasing as the cost terabyte of disk keeps dropping. Th means a lot of data can be kept fo long time. Distribution is incr ing as more and more data and w are spread across a great distan Data within a data center seems away." Data within a many-core c may seem "far away." Ambiguity increasing when trying to coordinat with systems that are far away—mo stuff has happened since you ha heard the news. Can you take actio with incomplete knowledge? Can wait for enough knowledge?

Turtles all the way down." As vari ous technological areas have evolved, they have responded to these trend of increasing storage, distributio and

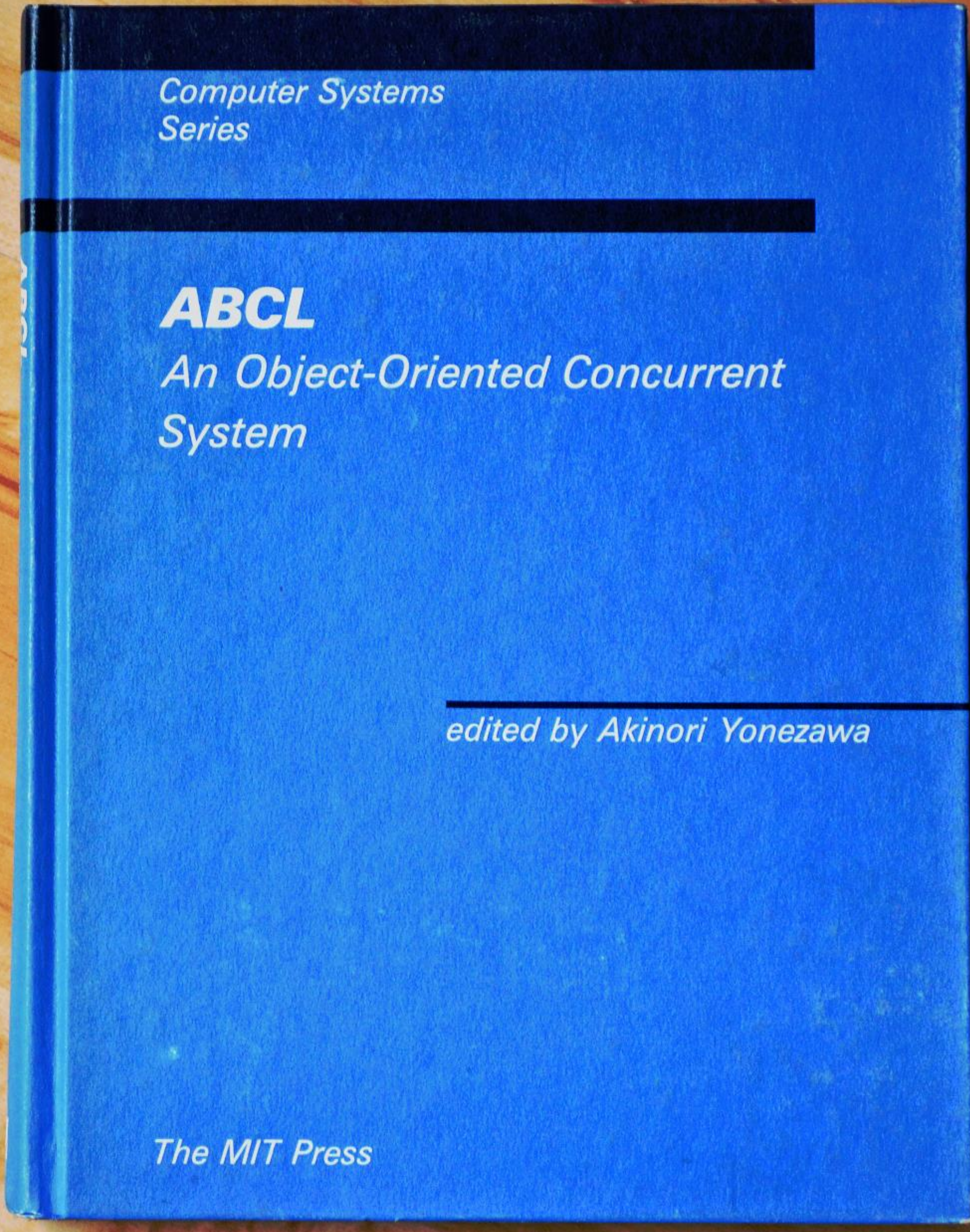


Michael Feathers
@mfeathers

OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.

3:27 PM - 3 Nov 2010

https://twitter.com/mfeathers/status/29581296216



Computer Systems
Series

ABCL

An Object-Oriented Concurrent System

edited by Akinori Yonezawa

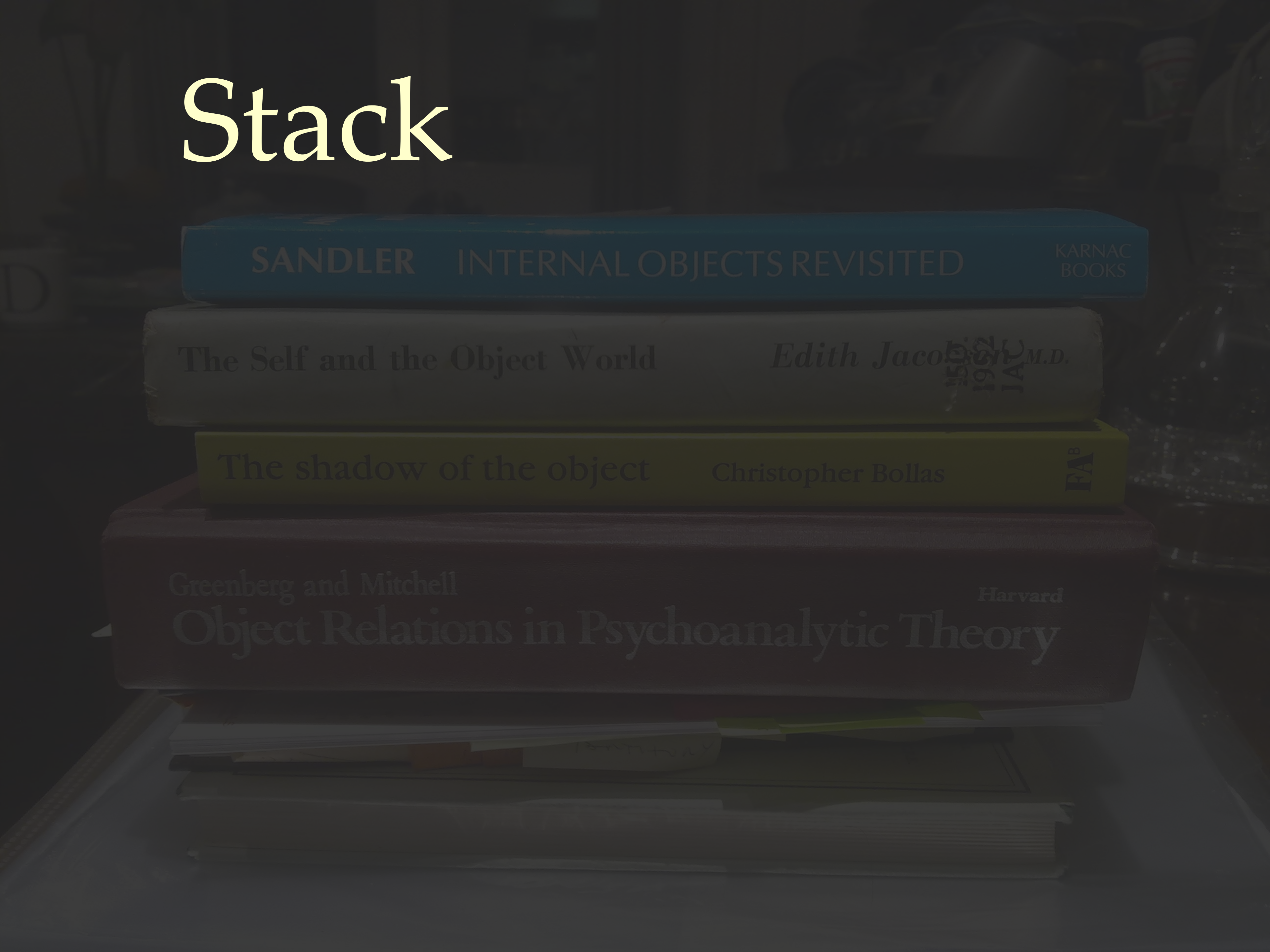
The MIT Press

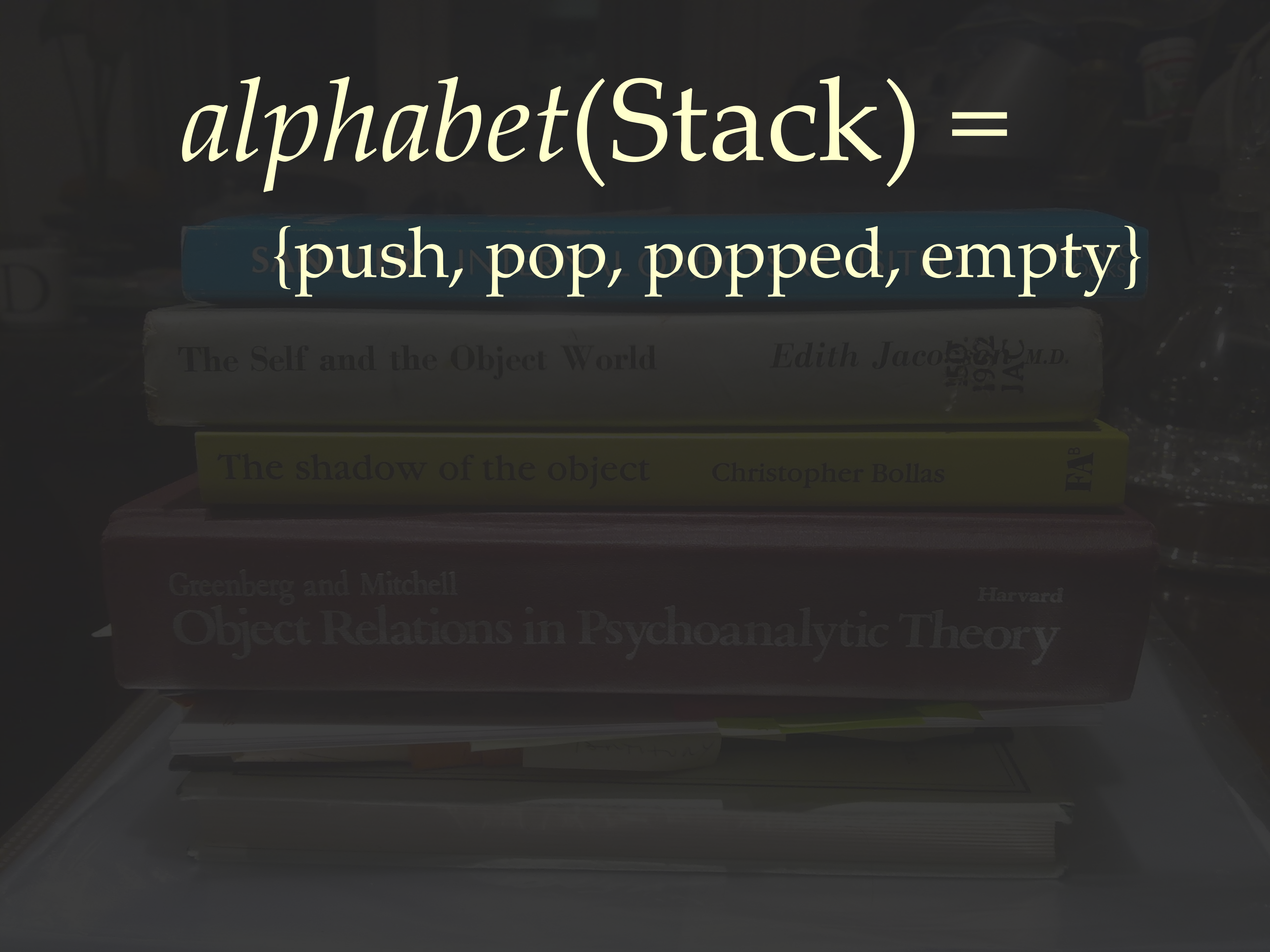
Multithreading is just one damn thing after, before, or simultaneous with another.

Andrei Alexandrescu

Actor-based concurrency is just one damn message after another.

Stack


SANDLER INTERNAL OBJECTS REVISITED
KARNAC BOOKS
The Self and the Object World
Edith Jaco M.D.
150 1952 JAC
The shadow of the object
Christopher Bollas
FAB
Greenberg and Mitchell
Object Relations in Psychoanalytic Theory
Harvard

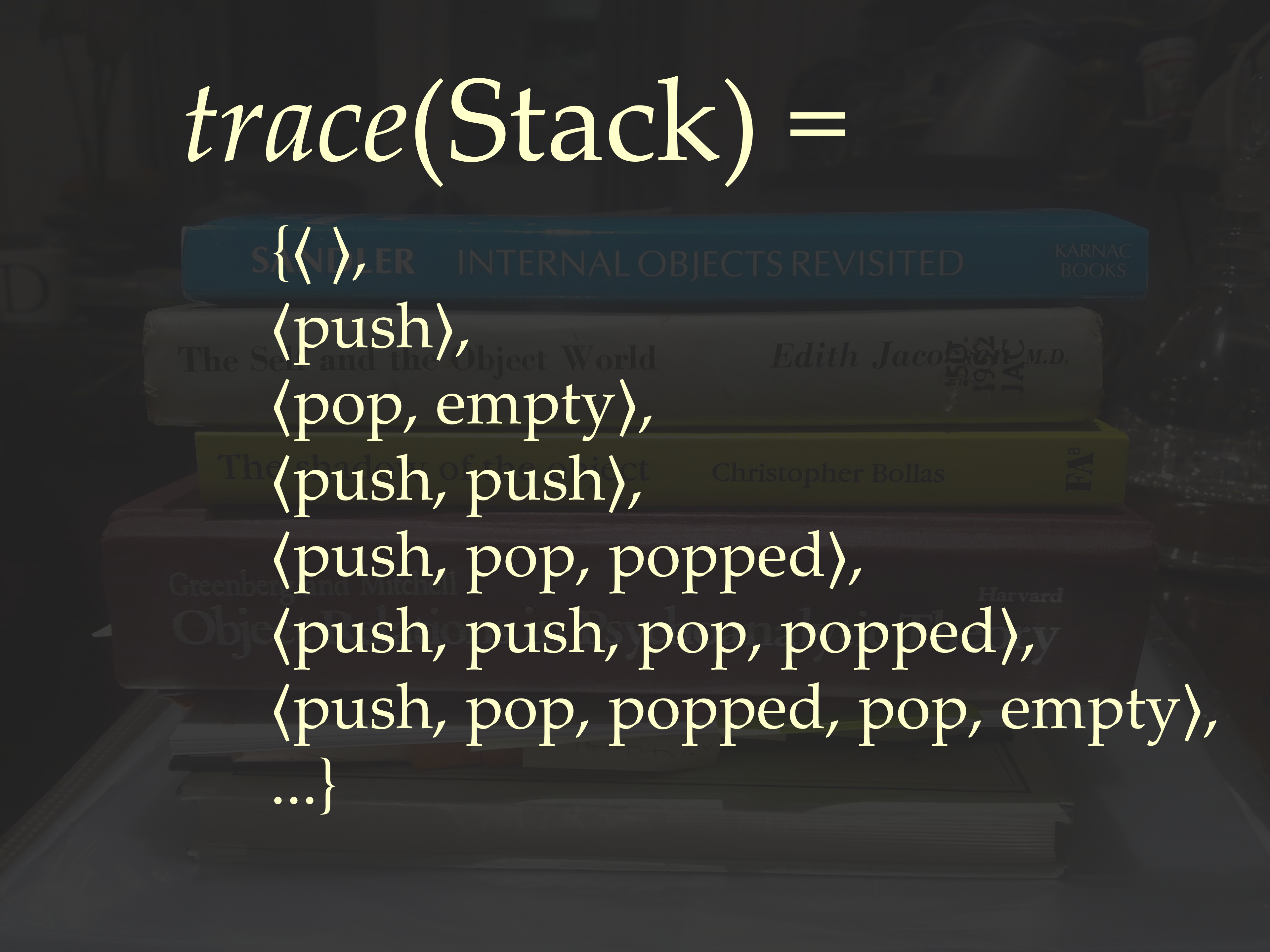


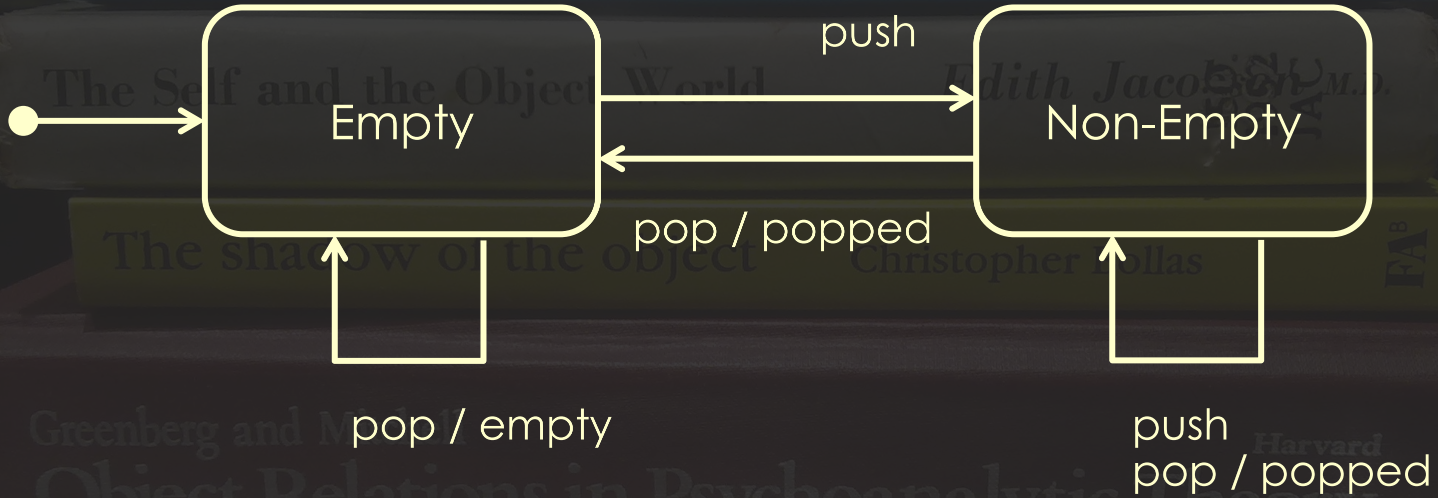
alphabet(Stack) =

{push, pop, popped, empty}

trace(Stack) =

{⟨ ⟩,
⟨push⟩,
⟨pop, empty⟩,
⟨push, push⟩,
⟨push, pop, popped⟩,
⟨push, push, pop, popped⟩,
⟨push, pop, popped, pop, empty⟩,
…}





Empty

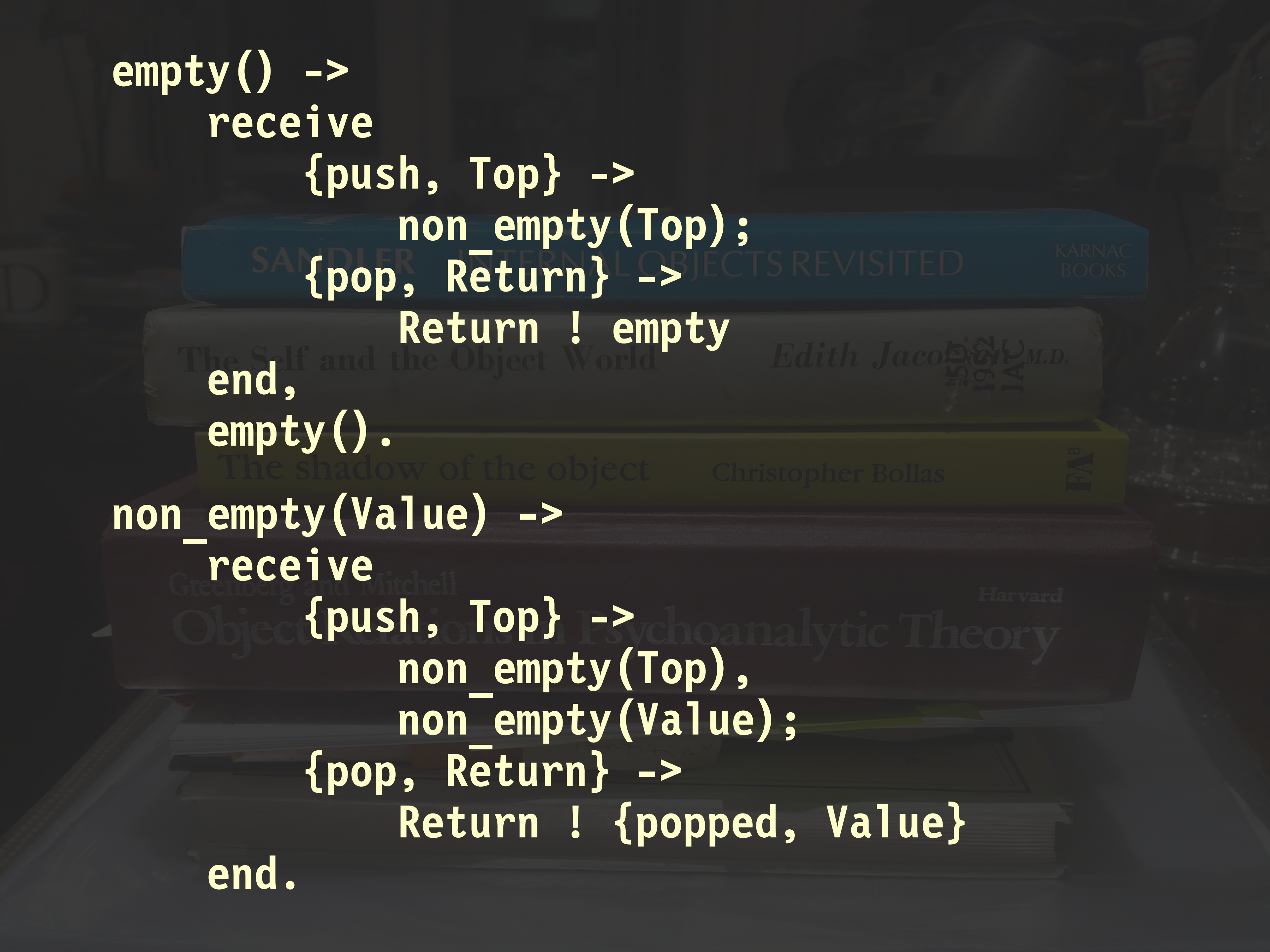
Non-Empty

push

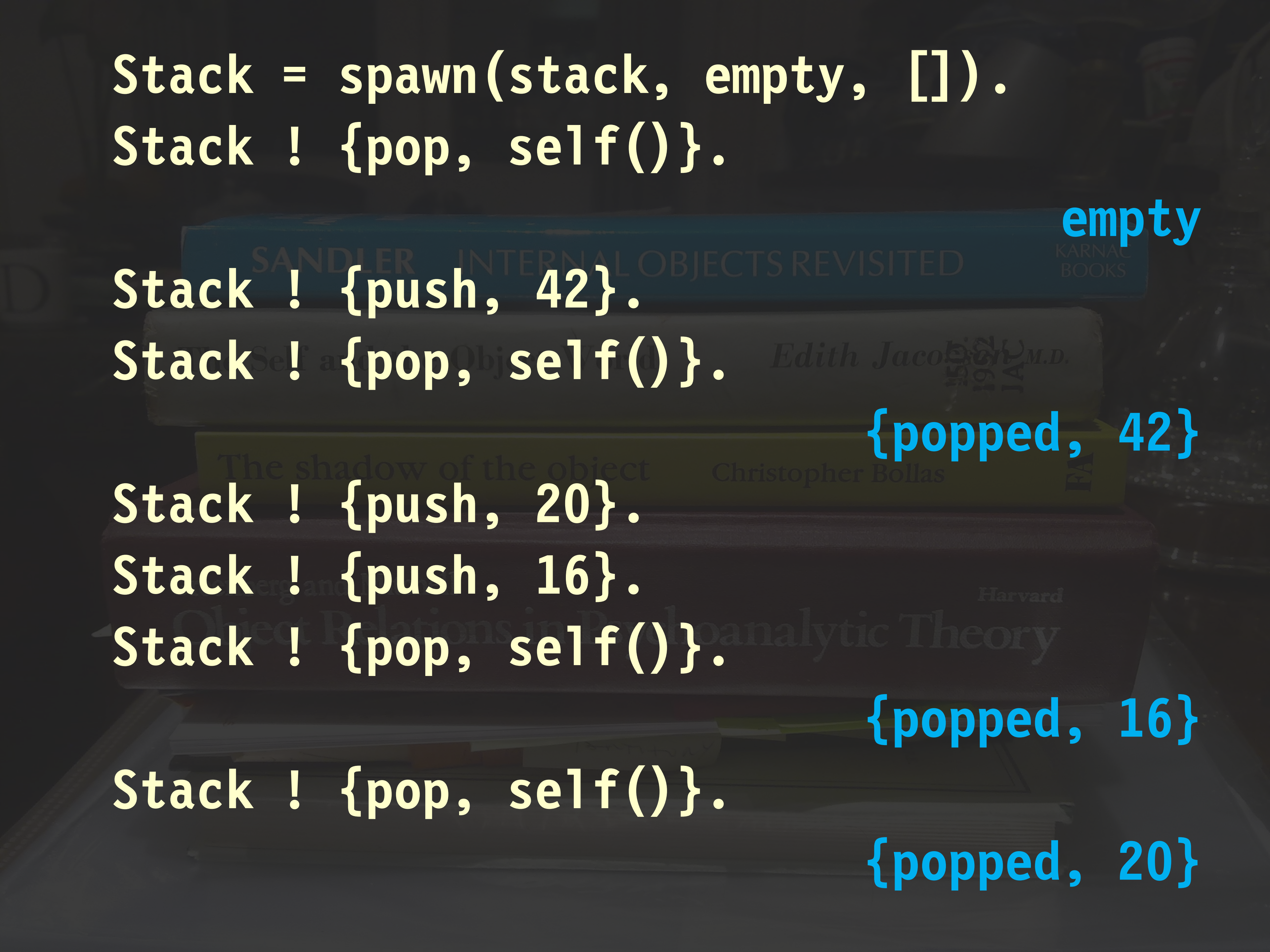
pop / popped

pop / empty

push
pop / popped



```erlang
empty() ->
    receive
        {push, Top} ->
            non_empty(Top);
        {pop, Return} ->
            Return ! empty
    end,
    empty().

non_empty(Value) ->
    receive
        {push, Top} ->
            non_empty(Top),
            non_empty(Value);
        {pop, Return} ->
            Return ! {popped, Value}
    end.
```



```
Stack = spawn(stack, empty, []).
Stack ! {pop, self()}.
                                  empty

Stack ! {push, 42}.
Stack ! {pop, self()}.
                           {popped, 42}

Stack ! {push, 20}.
Stack ! {push, 16}.
Stack ! {pop, self()}.
                           {popped, 16}

Stack ! {pop, self()}.
                           {popped, 20}
```

Requirement

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Requirement

- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
- Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
- Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
- Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.
- Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.
- Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.
- Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Requirement

Trigger

- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Precondition

- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Sequence

- Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
- Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
- Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Postcondition

- Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Requirement

Precondition

- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Trigger

- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Sequence

- Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
- Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
- Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Postcondition

- Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Requirement

Precondition

- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Trigger

- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Postcondition

- Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Sequence

- Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
- Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
- Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Requirement

Precondition

- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Trigger

- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Postcondition

- Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Requirement

Given

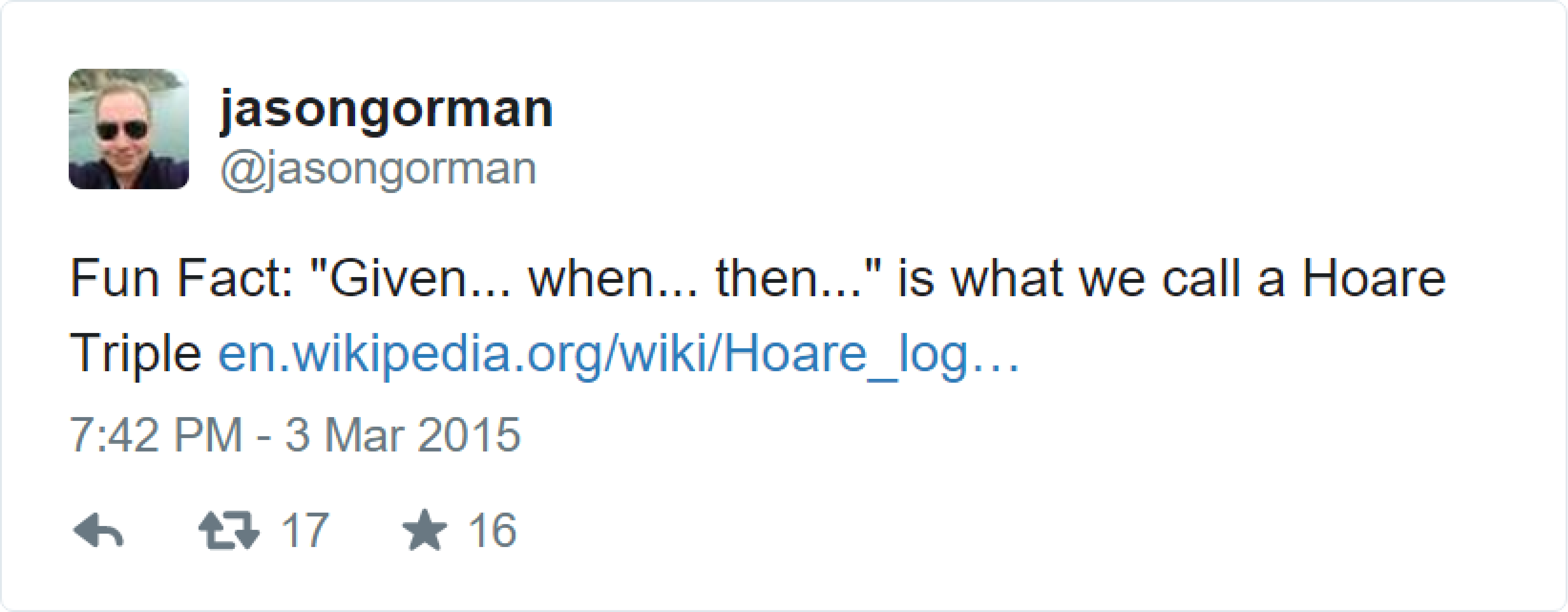
- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

When

- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Then

- Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.



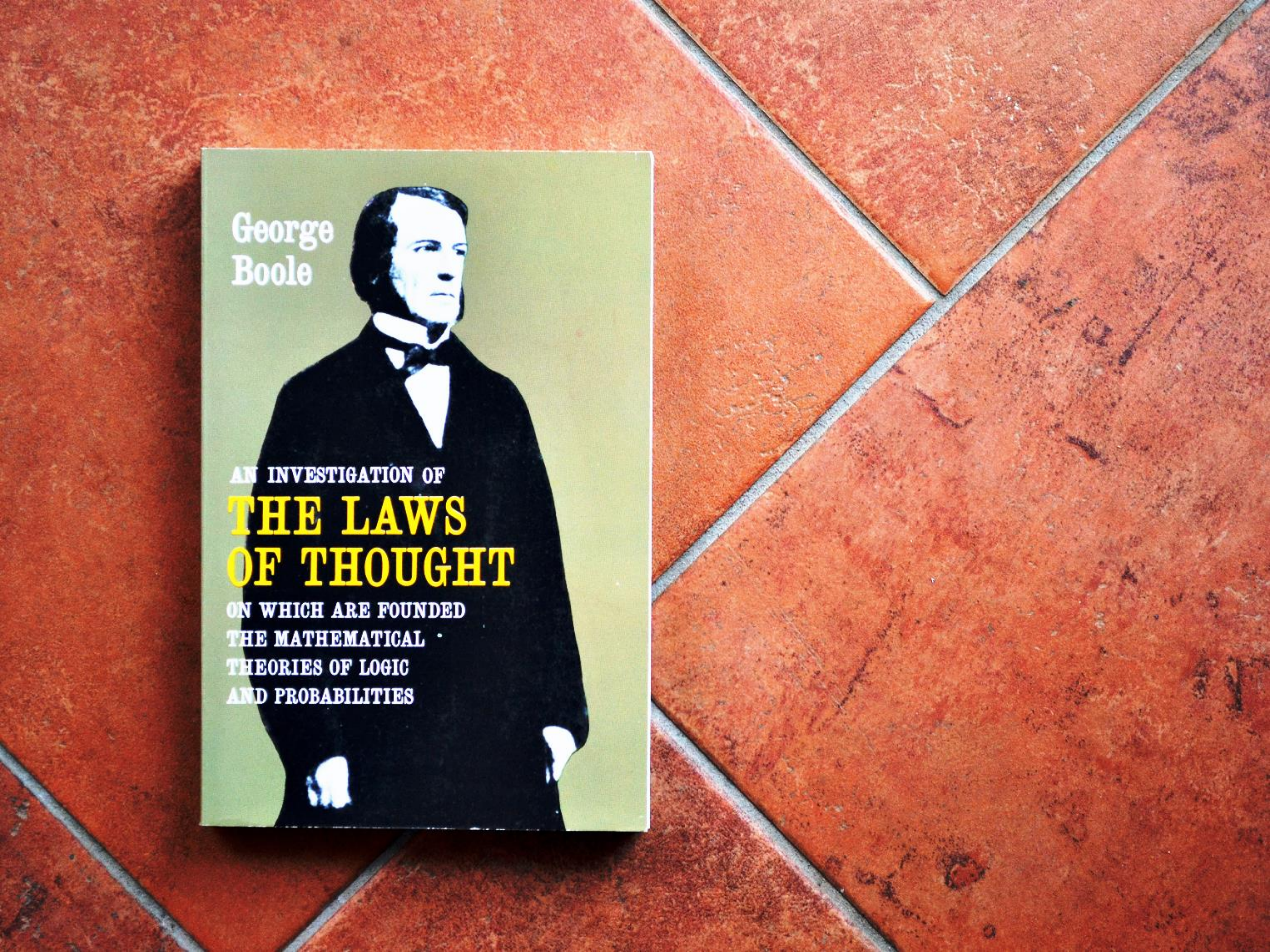
jasongorman
@jasongorman

Fun Fact: "Given... when... then..." is what we call a Hoare Triple en.wikipedia.org/wiki/Hoare_log…

7:42 PM - 3 Mar 2015

↩ 🔁 17 ★ 16

$$\{P\}\ S\ \{Q\}$$



George
Boole

AN INVESTIGATION OF

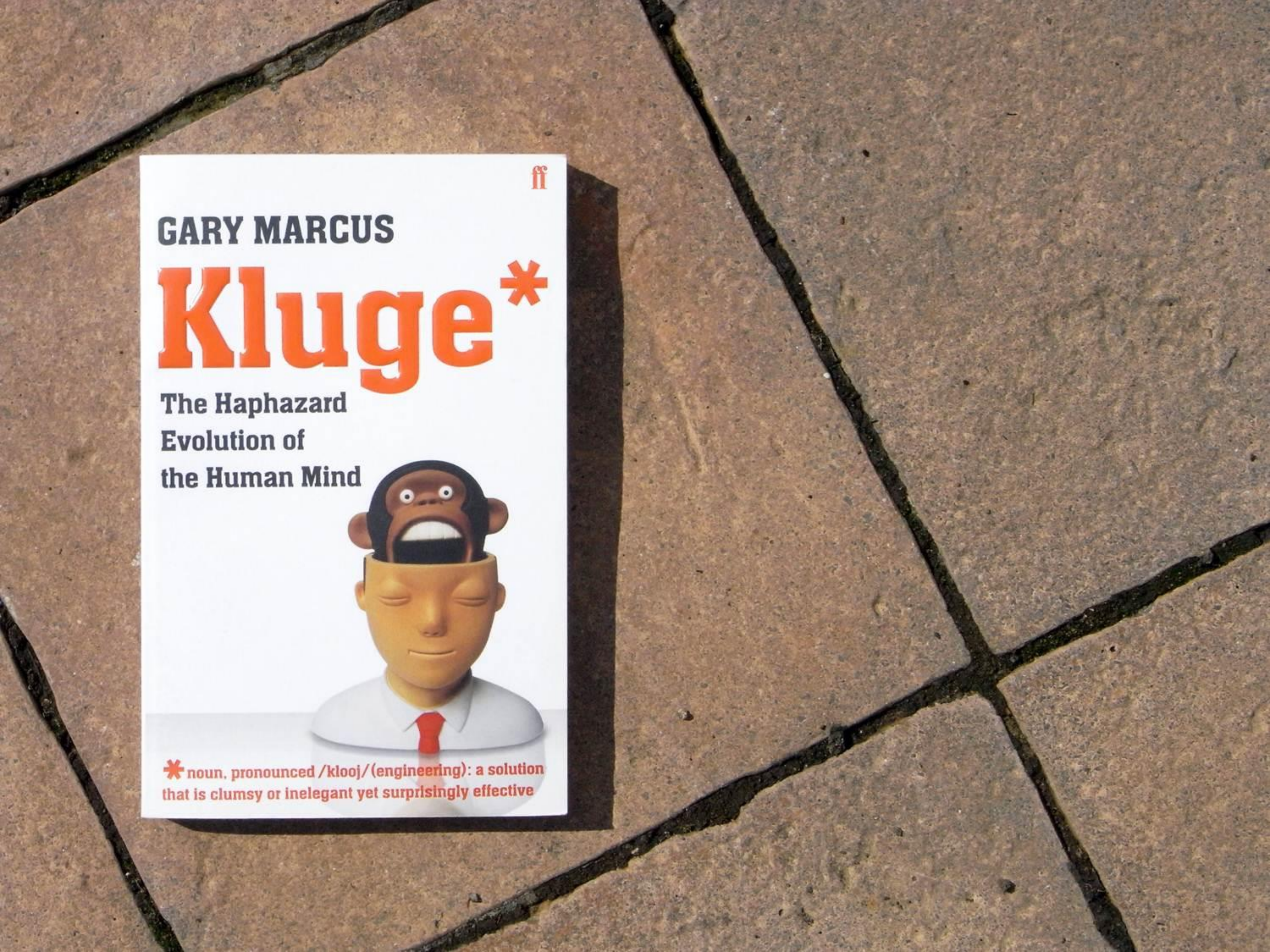
THE LAWS OF THOUGHT

ON WHICH ARE FOUNDED
THE MATHEMATICAL
THEORIES OF LOGIC
AND PROBABILITIES

```
void qsort(
    void * base,
    size_t element_count,
    size_t element_size,
    int compare(const void * lhs, const void * rhs));
```

```
void qsort(
    void * base,
    size_t element_count,
    size_t element_size,
    int compare(const void * lhs, const void * rhs))
{
    assert(???); // What is the precondition?
    ...
}
```

```
void qsort(
    void * base,
    size_t element_count,
    size_t element_size,
    int compare(const void * lhs, const void * rhs))
{
    assert(???); // What is the precondition?
    ...
    assert(???); // What is the postcondition?
}
```



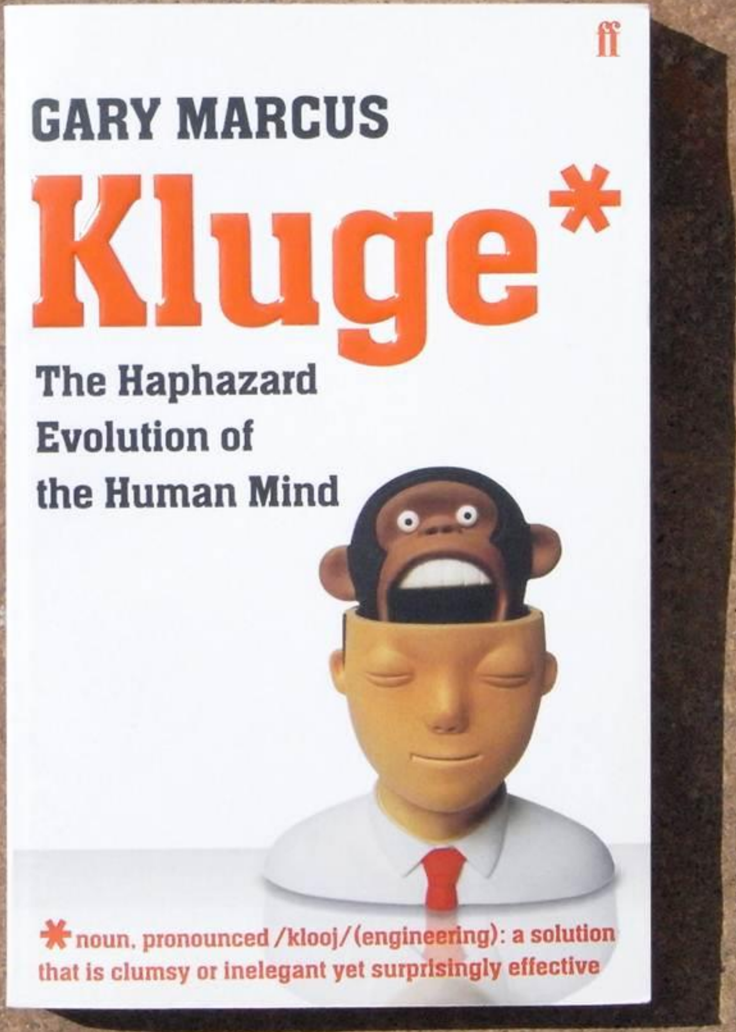
ff

GARY MARCUS

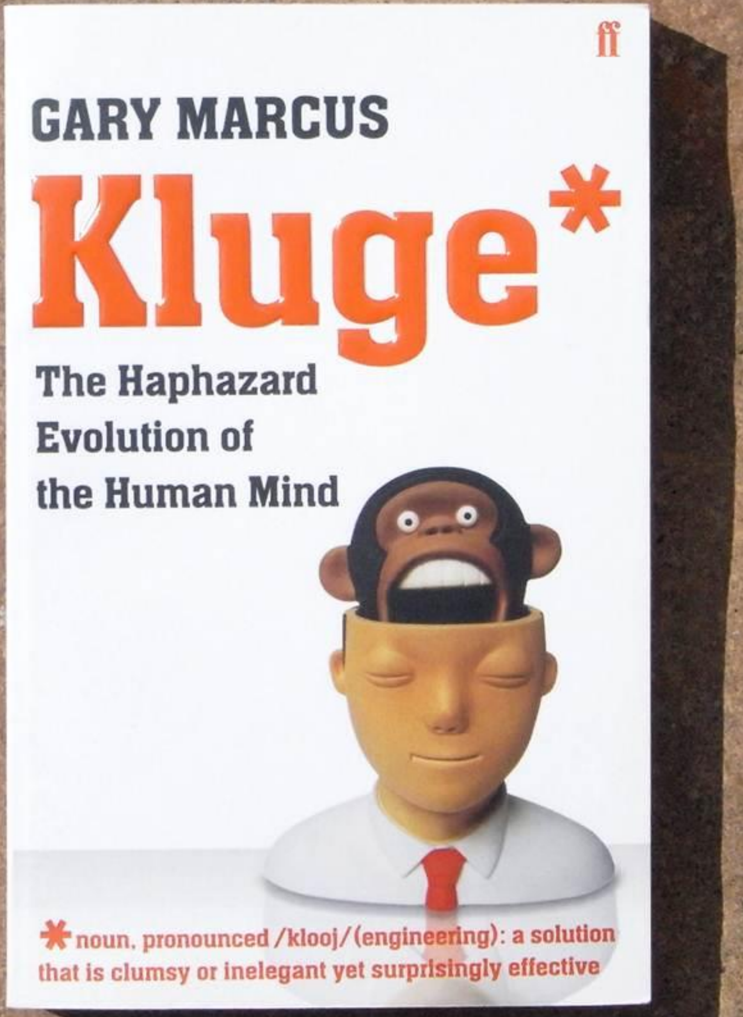
Kluge*

The Haphazard Evolution of the Human Mind

*noun, pronounced /klooj/(engineering): a solution that is clumsy or inelegant yet surprisingly effective


ff
GARY MARCUS
Kluge*
The Haphazard Evolution of the Human Mind
*noun, pronounced /klooj/ (engineering): a solution that is clumsy or inelegant yet surprisingly effective

Are human beings "noble in reason" and "infinite in faculty" as William Shakespeare famously wrote? Perfect, "in God's image," as some biblical scholars have asserted?


ff
GARY MARCUS
Kluge*
The Haphazard Evolution of the Human Mind
*noun, pronounced /klooj/(engineering): a solution that is clumsy or inelegant yet surprisingly effective

Hardly.

```python
def is_leap_year(year):
    ...
    # What is the postcondition?
```

```
def is_leap_year(year):
    ...
    # Given
    #  result = is_leap_year(year)
    # Then
    #  result == (
    #      year % 4 == 0 and
    #      year % 100 != 0 or
    #      year % 400 == 0)
```

```python
def is_leap_year(year):
    return (
        year % 4 == 0 and
        year % 100 != 0 or
        year % 400 == 0)
```

```
def test_is_leap_year():
    ...
```

```python
def test_is_leap_year_works():
    ...
```

Express intention
of code usage
with respect to data

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)
    @test
    def years_divisible_by_400_are_leap_years(self):
        assert is_leap_year(2000)
```

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)
    @test
    def years_divisible_by_400_are_leap_years(self):
        assert is_leap_year(2000)
```

```
def test(function):
    function.is_test = True
    return function

def check(suite):
    tests = [
        attr
        for attr in (getattr(suite, name) for name in dir(suite))
        if callable(attr) and hasattr(attr, 'is_test')]
    for to_test in tests:
        try:
            to_test(suite())
        except:
            print('Failed: ' + to_test.__name__ + '()')
```

Express intention
of code usage
with respect to data

Express intention

Naming

Grouping and nesting

# of code usage

Realisation of intent

with respect to data

One exemplar or many

Explicit or generated

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
        assert not is_leap_year(1999)
        assert not is_leap_year(1)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)
    @test
    def years_divisible_by_400_are_leap_years(self):
        assert is_leap_year(2000)
```

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
        assert not is_leap_year(1999)
        assert not is_leap_year(1)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
        assert is_leap_year(1984)
        assert is_leap_year(4)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)
    @test
    def years_divisible_by_400_are_leap_years(self):
        assert is_leap_year(2000)
```

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
        assert not is_leap_year(1999)
        assert not is_leap_year(1)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
        assert is_leap_year(1984)
        assert is_leap_year(4)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)
    @test
    def years_divisible_by_400_are_leap_years(self):
        for year in range(400, 2401, 400):
            assert is_leap_year(year)
```

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
        assert not is_leap_year(1999)
        assert not is_leap_year(1)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
        assert is_leap_year(1984)
        assert is_leap_year(4)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        for year in range(100, 2101, 100):
            if year % 400 != 0:
                assert not is_leap_year(year)
    @test
    def years_divisible_by_400_are_leap_years(self):
        for year in range(400, 2401, 400):
            assert is_leap_year(year)
```

```
class LeapYearSpec:
    @test
    def years_not_divisible_by_4_are_not_leap_years(self):
        assert not is_leap_year(2015)
        assert not is_leap_year(1999)
        assert not is_leap_year(1)
    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)
        assert is_leap_year(1984)
        assert is_leap_year(4)
    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        for year in range(100, 2101, 100):
            if year % 400 != 0:
                assert not is_leap_year(year)
    @test
    def years_divisible_by_400_are_leap_years(self):
        for year in range(400, 2401, 400):
            assert is_leap_year(year)
```

```python
def test(function):
    function.is_test = True
    return function

def data(*values):
    def prepender(function):
        function.data = values + getattr(function, 'data', ())
        return function
    return prepender

def check(suite):
    tests = [
        attr
        for attr in (getattr(suite, name) for name in dir(suite))
        if callable(attr) and hasattr(attr, 'is_test')]
    for to_test in tests:
        try:
            if hasattr(to_test, 'data'):
                for value in to_test.data:
                    call = '(' + str(value) + ')'
                    to_test(suite(), value)
            else:
                call = '()'
                to_test(suite())
        except:
            print('Failed: ' + to_test.__name__ + call)
```

```
class LeapYearSpec:
    @test
    @data(2015)
    @data(1999)
    @data(1)
    def years_not_divisible_by_4_are_not_leap_years(self, year):
        assert not is_leap_year(year)

    @test
    def years_divisible_by_4_but_not_by_100_are_leap_years(self):
        assert is_leap_year(2016)

    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)

    @test
    def years_divisible_by_400_are_leap_years(self):
        assert is_leap_year(2000)
```

```
class LeapYearSpec:
    @test
    @data(2015)
    @data(1999)
    @data(1)
    def years_not_divisible_by_4_are_not_leap_years(self, year):
        assert not is_leap_year(year)

    @test
    @data(2016, 1984, 4)
    def years_divisible_by_4_but_not_by_100_are_leap_years(self, year):
        assert is_leap_year(year)

    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)

    @test
    def years_divisible_by_400_are_leap_years(self):
        assert is_leap_year(2000)
```

```
class LeapYearSpec:
    @test
    @data(2015)
    @data(1999)
    @data(1)
    def years_not_divisible_by_4_are_not_leap_years(self, year):
        assert not is_leap_year(year)

    @test
    @data(2016, 1984, 4)
    def years_divisible_by_4_but_not_by_100_are_leap_years(self, year):
        assert is_leap_year(year)

    @test
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self):
        assert not is_leap_year(1900)

    @test
    @data(*range(400, 2401, 400))
    def years_divisible_by_400_are_leap_years(self, year):
        assert is_leap_year(year)
```

```
class LeapYearSpec:
    @test
    @data(2015)
    @data(1999)
    @data(1)
    def years_not_divisible_by_4_are_not_leap_years(self, year):
        assert not is_leap_year(year)
    @test
    @data(2016, 1984, 4)
    def years_divisible_by_4_but_not_by_100_are_leap_years(self, year):
        assert is_leap_year(year)
    @test
    @data(*(year for year in range(100, 2101, 100) if year % 400 != 0))
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self, year):
        assert not is_leap_year(year)
    @test
    @data(*range(400, 2401, 400))
    def years_divisible_by_400_are_leap_years(self, year):
        assert is_leap_year(year)
```

```
class LeapYearSpec:
    @test
    @data(2015)
    @data(1999)
    @data(1)
    def years_not_divisible_by_4_are_not_leap_years(self, year):
        assert not is_leap_year(year)
    @test
    @data(2016, 1984, 4)
    def years_divisible_by_4_but_not_by_100_are_leap_years(self, year):
        assert is_leap_year(year)
    @test
    @data(*(year for year in range(100, 2101, 100) if year % 400 != 0))
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self, year):
        assert not is_leap_year(year)
    @test
    @data(*range(400, 2401, 400))
    def years_divisible_by_400_are_leap_years(self, year):
        assert is_leap_year(year)
```

```python
class LeapYearSpec:
    @test
    @data(2015)
    @data(1999)
    @data(1)
    def years_not_divisible_by_4_are_not_leap_years(self, year):
        assert not is_leap_year(year)

    @test
    @data(2016, 1984, 4)
    def years_divisible_by_4_but_not_by_100_are_leap_years(self, year):
        assert is_leap_year(year)

    @test
    @data(*(year for year in range(100, 2101, 100) if year % 400 != 0))
    def years_divisible_by_100_but_not_by_400_are_not_leap_years(self, year):
        assert not is_leap_year(year)

    @test
    @data(*range(400, 2401, 400))
    def years_divisible_by_400_are_leap_years(self, year):
        assert is_leap_year(year)
```

Express intention

Naming
Grouping and nesting

# of code usage

Realisation of intent

with respect to data

One exemplar or many
Explicit or generated

intention

declaration of intent

Our task is not to find the maximum amount of content in a work of art.

Our task is to cut back content so that we can see the thing at all.

Susan Sontag