
DIE SCHLEIFE IST TOT, ...
©by s.ralph2010


LOOPS MUST
DIE
©bgs.ralph2010

@MarcoEmrich

https://OWL.Institute


OPINEL
Coccaft

Go To Statement Considered Harmful

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like, **while** *B* **repeat** *A* or **repeat** *A* **until** *B*). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.

1968

Go To Statement Considered Harmful

— Edsger Dijkstra

GOTO last month?

Loop last month?

2018

Loops are considered harmful

~~*— Marco Emrich*~~

— … and many others before

'GOTO Considered Harmful'

REACTIONS

'GOTO Considered Harmful'

We'll never publish anything like this again!

— ACM

1987

'GOTO Considered Harmful'

Considered Harmful

— Frank Rubin

1987

"GOTO Considered Harmful" Considered Harmful' *Considered Harmful?*

— Various Authors

From C2 Wiki

*For **every feature** of programming languages I have been able to find at **least one publication** claiming that it was harmful.*

— Dick Botting

Dick Botting Considered Harmful!

— Anonymous on C2 Wiki

X CONSIDERED

HARMFUL

Agile Considered Harmful

OOP Considered Harmful

MD5 Considered Harmful

Sleeping in loops considered harmful

Star Trek Considered Harmful

Java's new Considered Harmful

Redirect Considered Harmful

Global Variable Considered Harmful

Letter O Considered Harmful

UNIX Style Considered Harmful

“Considered Harmful” Essays Considered Harmful

It is not uncommon, in the context of academic debates over computer science and Web standards topics, to see the pub
has passed. Because “considered harmful” essays are, by their nature, so incendiary, they are counter-productive both i
they do good.

What Are “Considered Harmful” Essays?

The Jargon File has a short entry on “considered harmful” that encapsulates the genesis of such essays:

> *Edsger W. Dijkstra's note in the March 1968* Communications of the ACM, “*Go To Statement Considered Harmful*
> *Wirth.*

The controversy resulting from the article’s publication became so heated that the CACM subsequently decided to neve

The seeds of conflict were already in the ground, however, and in the years since 1968 there have been thousands of pie
exact phrase “considered harmful” in the document title. A similar search which looked for the exact phrase “considere

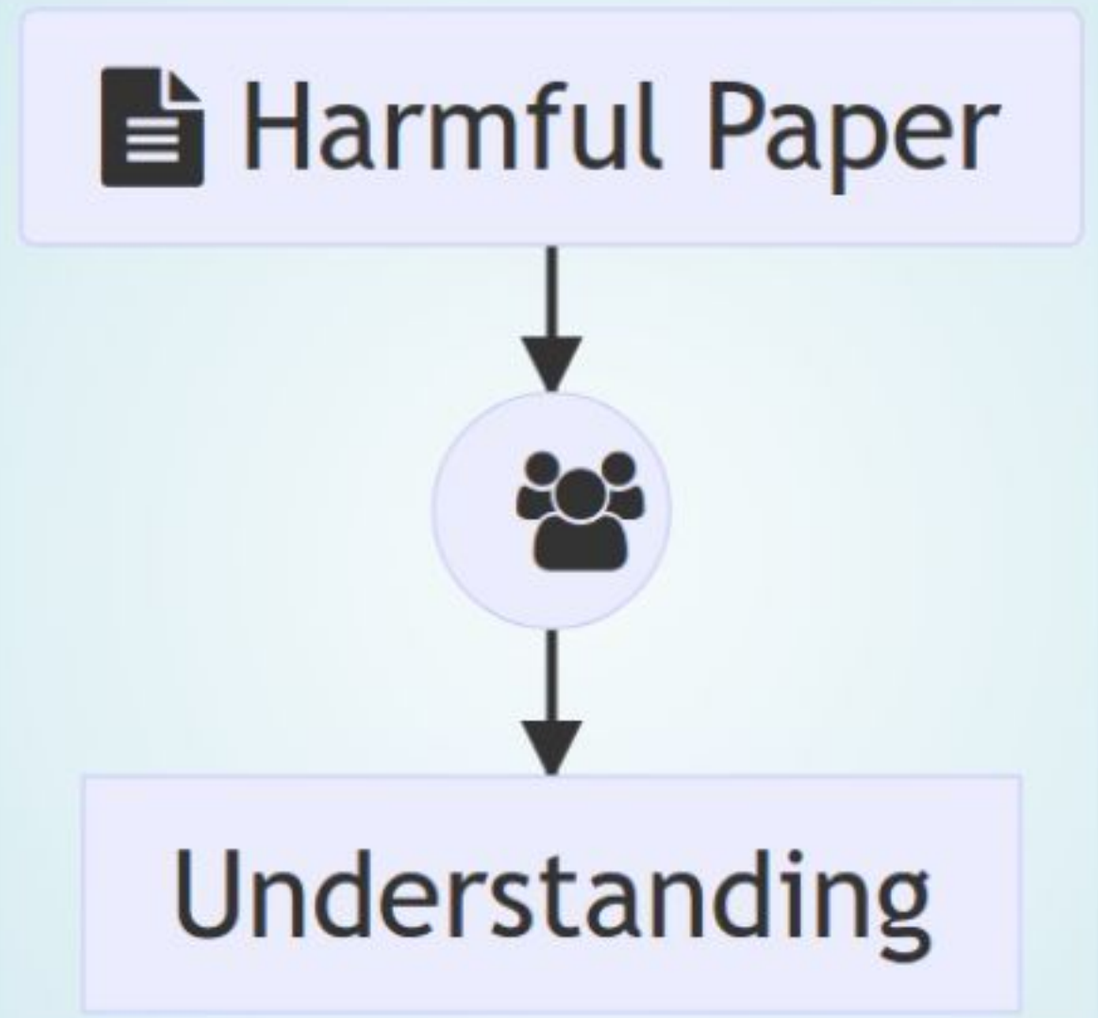
All of this content is the more wasteful because “considered harmful” essays have become something of a joke. In som
harmful” essays rarely, if ever, have the intended effect of weakening support for whatever it is they consider harmful.

Why Do People Write “Considered Harmful” Essays?

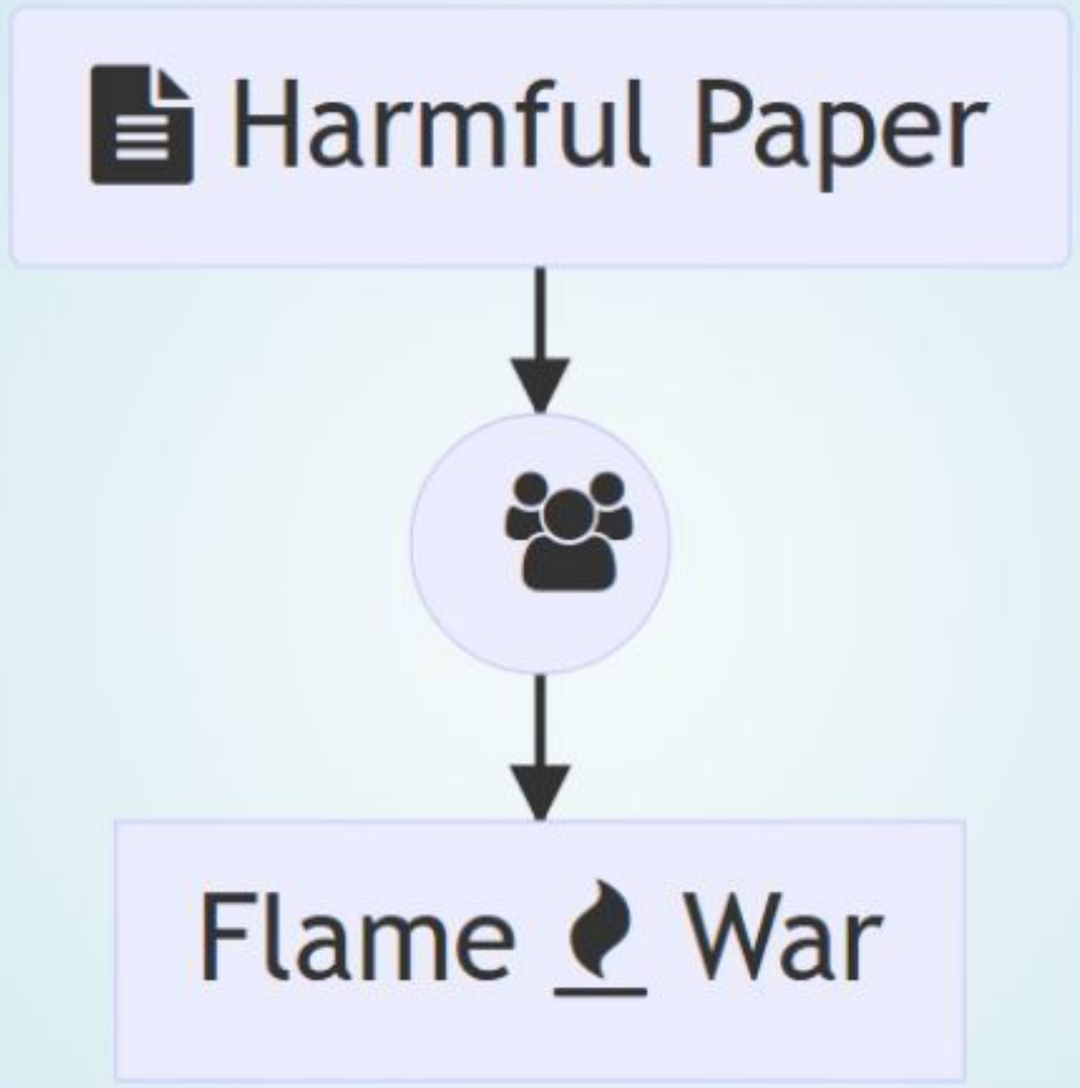
There are those cases where such essays are written because the author enjoys grandstanding, and knows that use of the
Essays Considered Harmful” would very likely be a case of using the “considered harmful” format to draw attention fo

Typically, “considered harmful” essays gets written because someone has an axe to grind, and they feel like making tha
“considered harmful” essays are intended to draw attention to a little-known subject about which the author is passiona

In addition, there are those “considered harmful” essays that are written as part of a long-running argument that has gra
doomsday devices in the eyes of their authors. The idea is that the arguments presented will be so devastating to the op
Godwin’s Law, we can draw a similar maxim: As a theoretical debate grows longer, the probability of a “considered ha


Harmful Paper
Understanding


Harmful Paper
My World View


Harmful Paper
Flame War


LOOPS MUST
DIE ?
©bgs.ralph2010

HEY LOOP!

What's your Problem?

LOOP ISSUE #1


ARCHERY?


"Almost hit"

ONE-OFF

+/-1

Uuups Loop

```
const a = [1, 2, 3, 4, 5];

for (let i = 1; i <= a.length; ++i) {
    console.log(a[i]);
}
```

Correct

```
const a = [1, 2, 3, 4, 5];

for (let i = 0; i < a.length; ++i) {
    console.log(a[i]);
}
```


Voodoo Chicken Coding

#1

ONE-OFF

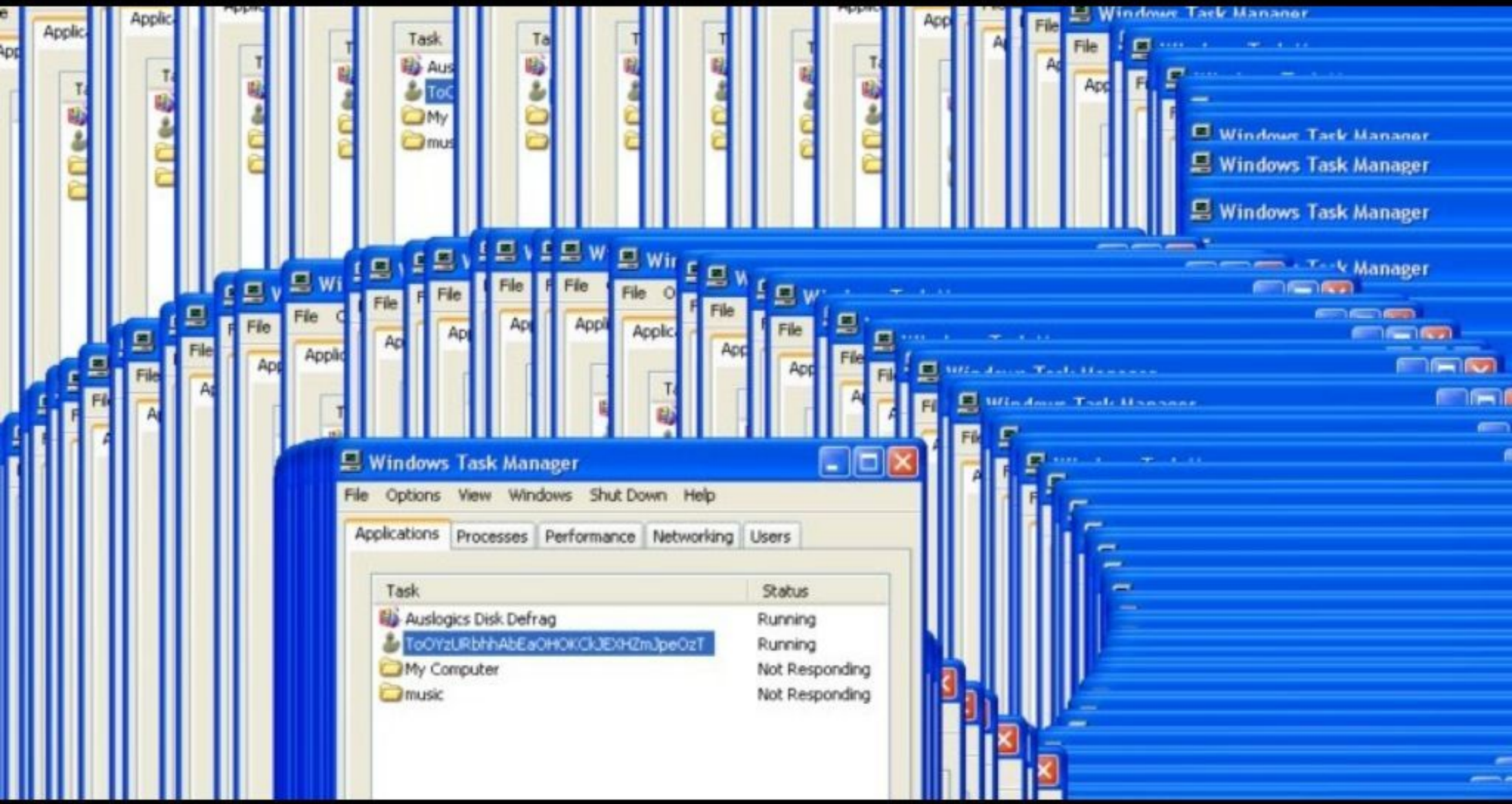
LOOP ISSUE #2

WARNING!

The system is either busy or has become unstable. You can wait and see if it becomes available again, or you can restart your computer.

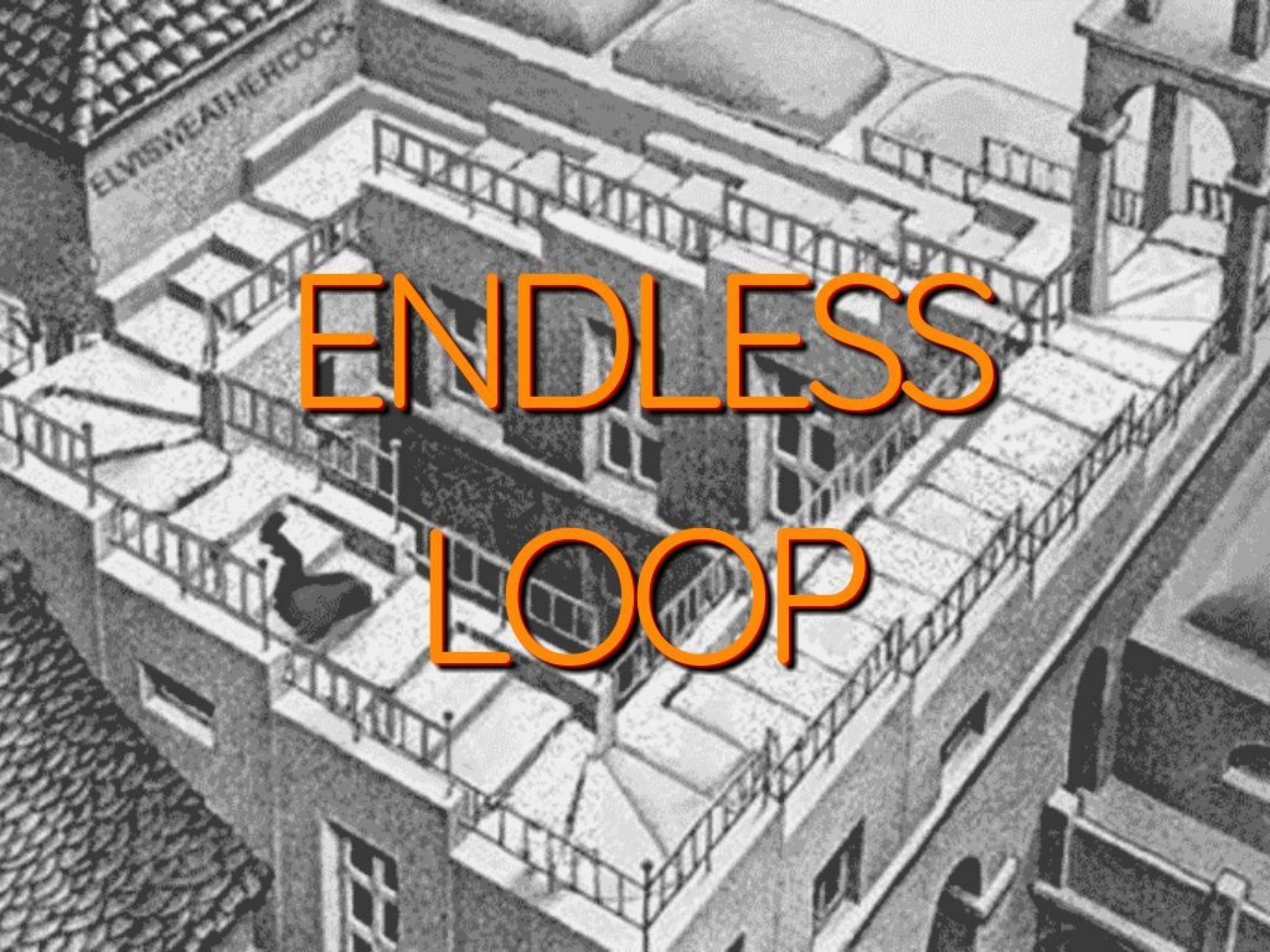
* Press any key to return to Windows and wait.
* Press CTRL+ALT+DEL again to restart your computer. You will lose unsaved information in any programs that are running.

Press any key to continue _


Windows Task Manager
File Options View Windows Shut Down Help
Applications Processes Performance Networking Users
Task Status
Auslogics Disk Defrag Running
ToOYzURbhhAbEaOHOlKClkJEXHZmJpeOzT Running
My Computer Not Responding
music Not Responding


Search
Recent (Read-Only)
TOSHIBA


FLAGSHIP


ELVIS WEATHERCOCK
ENDLESS
LOOP

Endless Loop

```
let j = -10;
while (j < 80);
  j *= -2;
```

Correct

```
let j = -10
while (j < 80)
  j *= -2;
```

#2

ENDLESS LOOP

LOOP ISSUE #3

BUGS IN PRODUCTION


FINLAND
Debugger

FINLAND

WHY?

STATE

CODE

≠

RUNTIME

Loops are

STATEFUL

STATE-

INFESTED

STATE IS THE

DEVIL


State

#3

Statefulness

LOOP ISSUE #4

<!-- ---

<!-- .slide: fullscreen-img="images/rick-forgo-filter.jpg" fullscreen-size="contain"

Given

```
const even = n => n % 2 === 0
```

Task

Loop (JS)

```
let a = [1, 2, 3, 4, 5, 6];
let r = [];

for (let i = 0; i < a.length; ++i) {
    if (even(a[i]) {
        r.push(a[i]);
    }
}
```

Filter (JS)

```
let r = [1, 2, 3, 4, 5, 6].filter(even)
```

Given

```
let users = [...];
const isAdmin = user => ... // true or false
```

Filter (JS)

```
let admins = user.filter(isAdmin)
```

Loop (JS)

```
let admins = []

for (let i = 0; i < a.length; ++i) {
    if (isAdmin(users[i]) {
        admins.push(users[i]);
    }
}
```

PLUMBING

What vs. How?

#4

HIDDEN INTENT

Loop Issues

1. One-Off
2. Endless Loops
3. Statefulness
4. Hidden Indent


USEFUL
ALTERNATIVES


ALTERNATIVE
#1

PROBLEMS HARD TOO LOOP

```
function guess_my_name(Array, Left, Right)
var
    L2, R2, PivotValue
begin
    Stack.Push(Left, Right);
    while not Stack.Empty do
    begin
        Stack.Pop(Left, Right);
        repeat
            PivotValue := Array[(Left + Right) div 2];
            L2 := Left;
            R2 := Right;
            repeat
                while Array[L2] < PivotValue do
                    L2 := L2 + 1;
                while Array[R2] > PivotValue do
                    R2 := R2 - 1;
                if L2 <= R2 then
                begin
                    if L2 != R2 then
                        Swap(Array[L2], Array[R2]);
                    L2 := L2 + 1;
                    R2 := R2 - 1;
                end;
            until L2 >= R2;
            if R2 - Left > Right - L2 then
            begin
                if Left < R2 then
                    Stack.Push(Left, R2);
                Left := L2;
            end;
            else
            begin
                if L2 < Right then
                    Stack.Push(L2, Right);
                Right := R2;
            end;
        until Left >= Right;
    end;
end;
```

Source: https://en.wikibooks.org/wiki/Algorithm_Implementation

Quicksort

iterative

```
function quicksort(Array, Left, Right)
var
    L2, R2, PivotValue
begin
    Stack.Push(Left, Right);
    while not Stack.Empty do
    begin
        Stack.Pop(Left, Right);
        repeat
            PivotValue := Array[(Left + Right) div 2];
            L2 := Left;
            R2 := Right;
            repeat
                while Array[L2] < PivotValue do
                    L2 := L2 + 1;
                while Array[R2] > PivotValue do
                    R2 := R2 - 1;
                if L2 <= R2 then
                begin
                    if L2 != R2 then
                        Swap(Array[L2], Array[R2]);
                    L2 := L2 + 1;
                    R2 := R2 - 1;
                end;
            until L2 >= R2;
            if R2 - Left > Right - L2 then
            begin
                if Left < R2 then
                    Stack.Push(Left, R2);
                Left := L2;
            end;
            else
            begin
                if L2 < Right then
                    Stack.Push(L2, Right);
                Right := R2;
            end;
        until Left >= Right;
    end;
end;
```

Source: https://en.wikibooks.org/wiki/Algorithm_Implementation

RECURSION

Recursive Quicksort (Haskell)

```haskell
qsort [] = []
qsort (p:xs) = (qsort lesser) ++ [p] ++ (qsort greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

Source: https://wiki.haskell.org/Introduction

Quicksort iterative

```
function quicksort(Array, Left, Right)
var
    L2, R2, PivotValue
begin
    Stack.Push(Left, Right);
    while not Stack.Empty do
    begin
        Stack.Pop(Left, Right);
        repeat
            PivotValue := Array[(Left + Right) div 2];
            L2 := Left;
            R2 := Right;
            repeat
                while Array[L2] < PivotValue do
                    L2 := L2 + 1;
                while Array[R2] > PivotValue do
                    R2 := R2 - 1;
                if L2 <= R2 then
                begin
                    if L2 != R2 then
                        Swap(Array[L2], Array[R2]);
                    L2 := L2 + 1;
                    R2 := R2 - 1;
                end;
            until L2 >= R2;
            if R2 - Left > Right - L2 then
            begin
                if Left < R2 then
                    Stack.Push(Left, R2);
                Left := L2;
            end;
            else
            begin
                if L2 < Right then
                    Stack.Push(L2, Right);
                Right := R2;
            end;
        until Left >= Right;
    end;
end;
```

Source: https://en.wikibooks.org/wiki/Algorithm_Implementation

Notes:

Recursive Quicksort (JS)

```
const qsort = list => {
  if (list.length === 0) return [];

  const [p, ...xs] = list;
  const lesser = xs.filter(x => x < p);
  const greater = xs.filter(x => x >= p);

  return [...qsort(lesser), p, ...qsort(greater)];
}
```

Source: 15 minutes time & 2 Glasses of Wine

Recursive Quicksort (Java)

```
public static void recursiveQsort(int[] arr, Integer start, Integ
    if (end - start < 2) return;

    int p = start + ((end - start) / 2);
    p = partition(arr, p, start, end);

    recursvieQsort(arr, start, p);
    recursvieQsort(arr, p+1, end);
}
```

Source: https://stackoverflow.com/questions/12553238/quicksort-iterative-or-recursive

Recursive-friendly Problems

- complex list operations
- multi dimensional
- trees-based algorithms
- graphs-based algorithms


ESCAPE HATCH

Solved by Recursion?

1. One-Off
2. Endless Loops→ Stack Overflow
3. Statefulness
4. Hidden Intent


ALTERNATIVE
#2


Map
Reduce
Filter
Knights of the Higher Order

Fun with

Higher-Order Functions

on List-alikes


MAP

Task

[1, 2, 3, 4] —sqr→ [1, 4, 9, 16]

Loop (JS)

```
let a = [1, 2, 3, 4];
let r = [];

for (let i = 0; i < a.length; ++i) {
    r[i] = a[i] * a[i];
}
```

Map (JS)

```
let r = [1, 2, 3, 4].map(x => x * x)
```

Haskell

```haskell
map (^2) [1, 2, 3, 4]
```

Ruby

```ruby
[1, 2, 3, 4].map {|x| x ** 2}
```

Java

```
Arrays.asList(1,2,3,4).stream()
    .map(x -> x * x)
    .toArray(Integer[]::new)
```

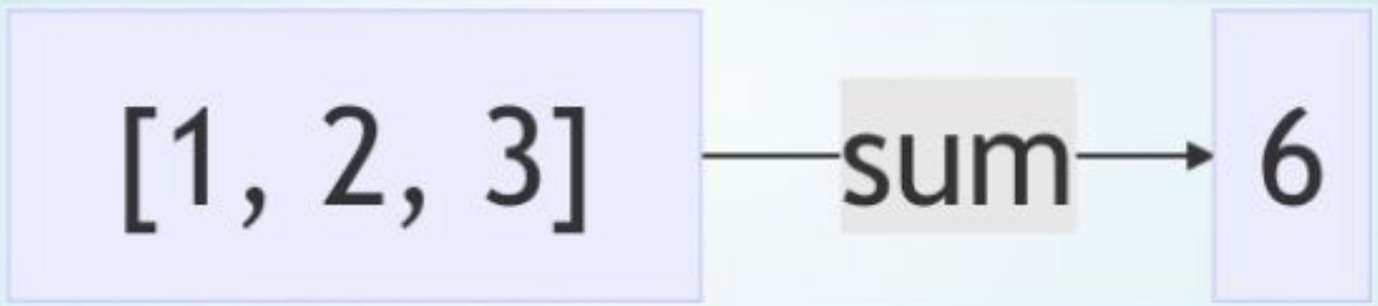
PHP

```
function sqr($x) { return $x ** 2; }
array_map("sqr", [1, 2, 3, 4]);
```


FILTER


Reduce/Fold

Task



Loop (JS)

```
let a = [1, 2, 3];
let r = 0;

for (let i = 0; i < a.length; ++i) {
    r += a[i];
}
```

Reduce (JS)

```
const add = (a,b) => a + b

let r = [1, 2, 3].reduce(add)
```

Sum (Java)

```java
int myArray[] = { 1, 2, 3 };

int sum = Arrays.stream(myArray).sum();
```

More

- some/any
- every/all
- keys
- values
- find
- findIndex
- flatten
- groupBy
- dropRepeats
- dropWhile
- evolve
- zip

...

Haskel, Elm, ...

- core libs

Ruby

- core libs

JS

- core libs ES5+
- Ramda, LoDash

Java

- core libs > Java 8

C#

- core libs
- LINQ

Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Lua: RxLua
- Ruby: Rx.rb
- Python: RxPY
- Go: RxGo
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift
- PHP: RxPHP
- Elixir: reaxive
- Dart: RxDart

ReactiveX for platforms and frameworks

- RxNetty
- RxAndroid
- RxCocoa

Solved by HOF?

1. One-Off
2. Endless Loops
3. Statefulness
4. Hidden Intent

PARADISE?

PARADISE!

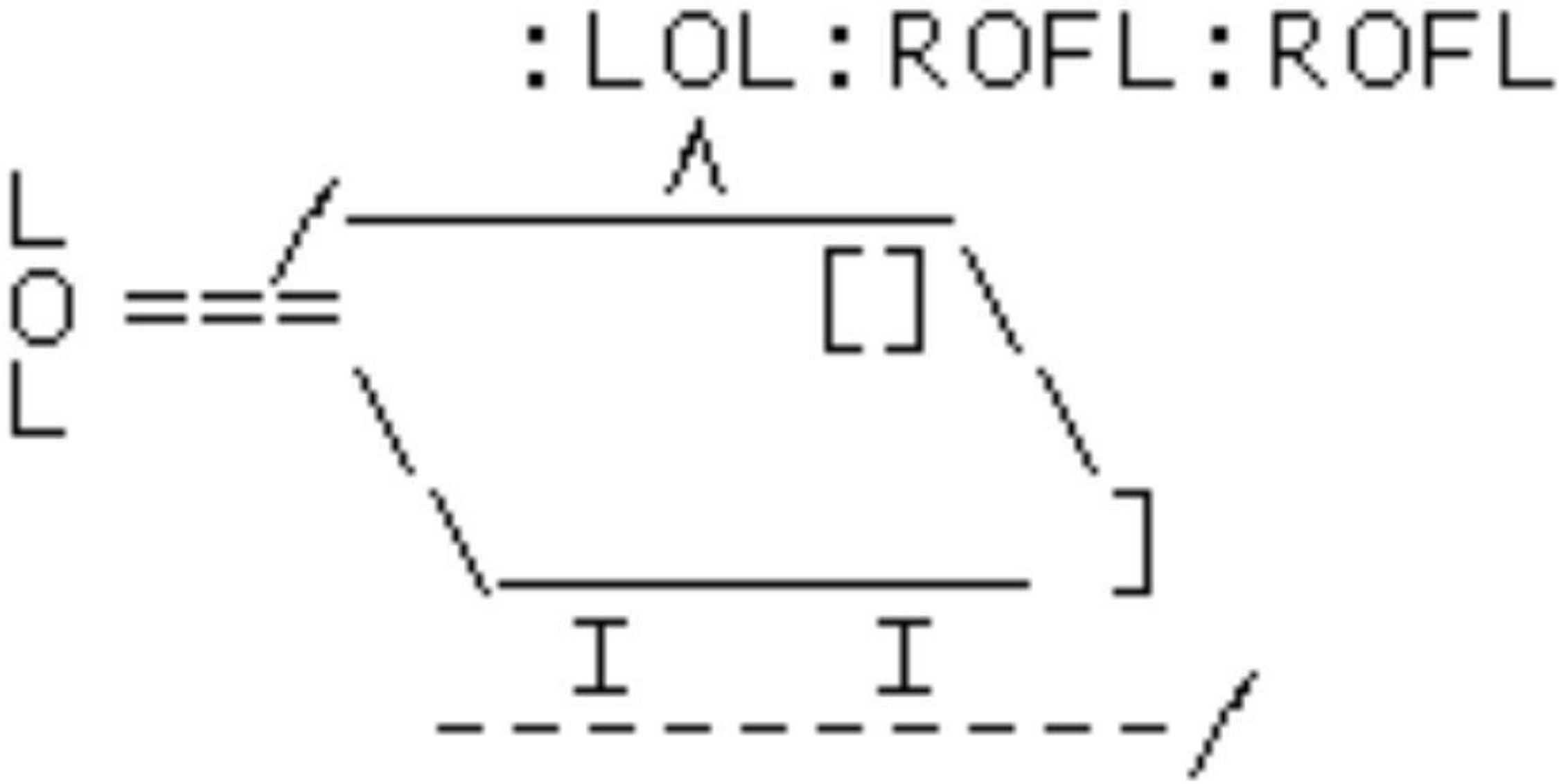

...not for all situatio
→ List-alikes

NO LIST?

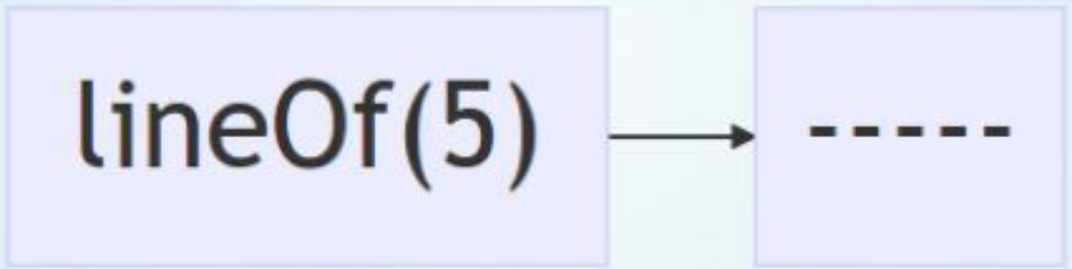
¯_(ツ)_/¯


ALTERNATIVE
#3


LIST
GENERATOR
FUNCTIONS


:LOL:ROFL:ROFL
ROFL COPTER!!!

Task



Loop (JS)

```
let r = [];

for (let i = 1; i <= 5; ++i) {
    r[i] += '-';
}

let s = r.join('');
```


Good Ol'
TIMES

Times (JS, Ramda)

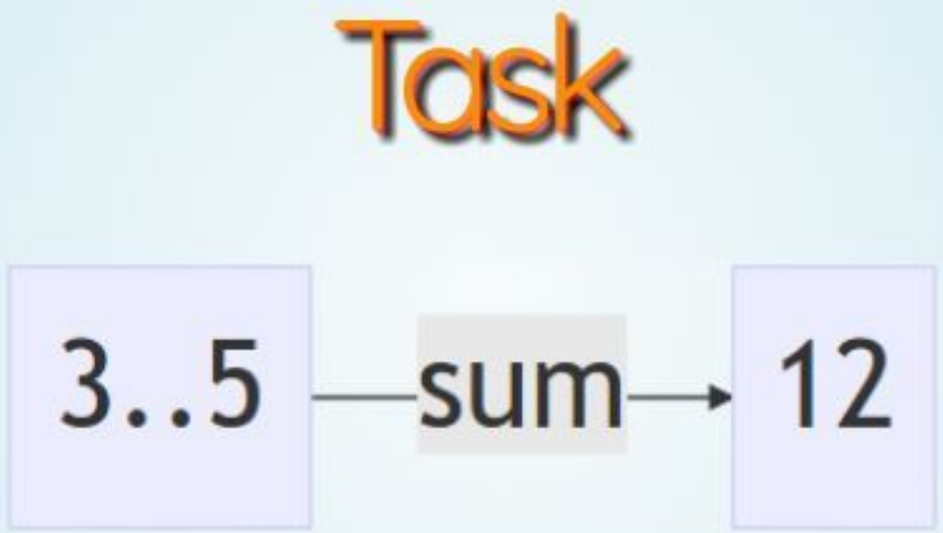
```
let s = times(_ => '-', 5).join();
```

Loop (JS)

```
let r = [];

for (let i = 1; i <= 5; ++i) {
    r[i] += '-';
}

let s = r.join('');
```


Task
3..5
sum
12

Loop (JS)

```
let sum = 0;

for (let i = 3; i <= 5; ++i) {
    sum += i;
}
```


RANGE

Range (JS, Ramda)

```
sum(range(3, 6))
```

More

- unfold
- upTo
- repeat
- downTo

...

times (Ruby)

```
5.times {puts 'Hello!'}
```

range (Ruby)

```
(1..5).map {'-'}.join
```

operator * (Ruby)

```
5 * '_'
```

Solved by T&R?

1. One-Off
2. Endless Loops
3. Statefulness
4. Hidden Intent


When is a Loop, the right
Tool?

THE EVIL

P-WORD

Performance

1974

> *Premature Optimization is the root of all evil*

— Donald Knuth

Bottleneck

Performance of HOFs

Map, Filter, ...?

Java

Anglika Langer

https://jaxenter.com/java-performance-tutorial-how-fast-are-the-java-8-streams-118830.html

LOOP VS.

REDUCE

(500.000 Random Values)

ArrayList, for-loop : 6.55 ms

```
int m = Integer.MIN_VALUE;
for (int i : myList)
     if (i>m) m=i;
```

ArrayList, seq. Reduce : 8.33 ms

```
int m = myList.stream()
          .reduce(Integer.MIN_VALUE, Math::max);
```

(2 CPUs)

sequential: 5.35

```
int m = Arrays.stream(ints)
          .reduce(Integer.MIN_VALUE, Math::max);
```

parallel: 3.35

```
int m = Arrays.stream(ints).parallel()
          .reduce(Integer.MIN_VALUE, Math::max);
```

Pure int-arrays

Loop: 0.36 ms

```
int[] a = ints;
int e = ints.length;
int m = Integer.MIN_VALUE;
for(int i=0; i < e; i++)
  if(a[i] > m) m = a[i];
```

Reduce: 5.35ms

```
int m = Arrays.stream(ints)
         .reduce(Integer.MIN_VALUE, Math::max);
```

... the performance model of streams is not a trivial one

— Anglika Langer

… don't guess; instead, benchmark a lot

— Anglika Langer

Performance of Recursion?

Tail Call Optimization

~~tail position~~

```
function x() {
    ...
    return x(...) + 1;
}
```

tail position

```
function x() {
    ...
    return x(...);
}
```

Languages supporting TCO

- JS (Safari/WK)
- Scheme
- Erlang / Elixir
- Scala
- Kotlin
- (Lisp)
- (Ruby)
- (Python)

Languages without TCO

- JS (Chrome, Firefox, Edge, Node.JS)
- Java
- C#
- C/C++

⇨ LOOP ❤

Conclusion: Approach

- fits a recursive solution?
- list-alike? or stream-ish?
- list-generator

Performance Issue? → Loop (maybe)


LOOPS MUST
DIE?
©bgs.ralph2010

LOOPS MUST DIE?

A PRESENTATION BY

MARCO EMRICH

TITLE PHOTO


S. RALPH

CREDIT ANIMATION CSS

OLIVER KNOBLICH

MISSED TARGET ARTWORK

INFINITE LOOP

QUIZZ

Loop 1 (JS)

```
let i = 3;
let j = 5;
do {
  j *= 2;
  i -= 1;
} while (j + i < j - 1);
```

Loop 2 (JS)

```
let i = 123768;
while (i != 1) {
  i = i / 10;
}
```

Loop 3 (JS)

```
let j = -10;
while (j < 80);
  j *= -2;
```

Loop 4 (Java)

```java
int i = -1;
while (i != 0) {
  i--;
}
```

Loop 5 (Java)

```java
int i = 81;
int j = 625;
while (i != j) {
  j = j / 5;
  i = i / 3;
}
```

Loop6(C)

```c
int a = 0;
while (a < 10) {
  printf("%d\n", a);
  if (a = 5)
      printf("a equals 5!\n");
  a++;
}
```

Loop 7 (C)

```c
float x = 0.1;
while (x != 1.1) {
  printf("x = %f\n", x);
  x = x + 0.1;
}
```