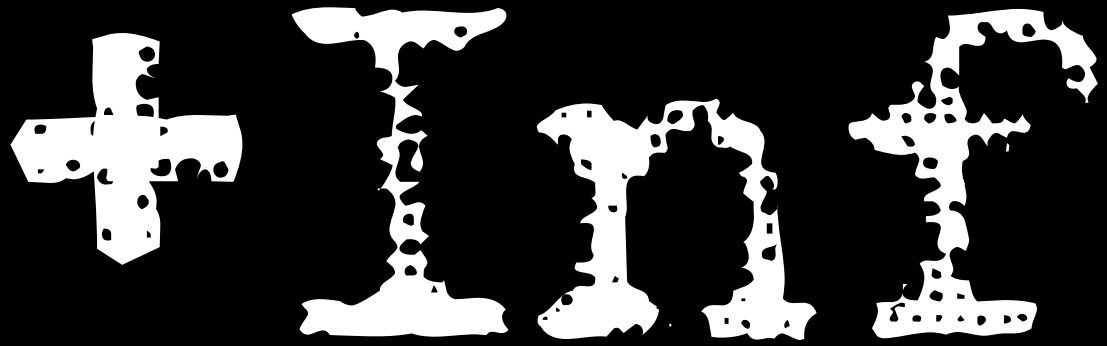
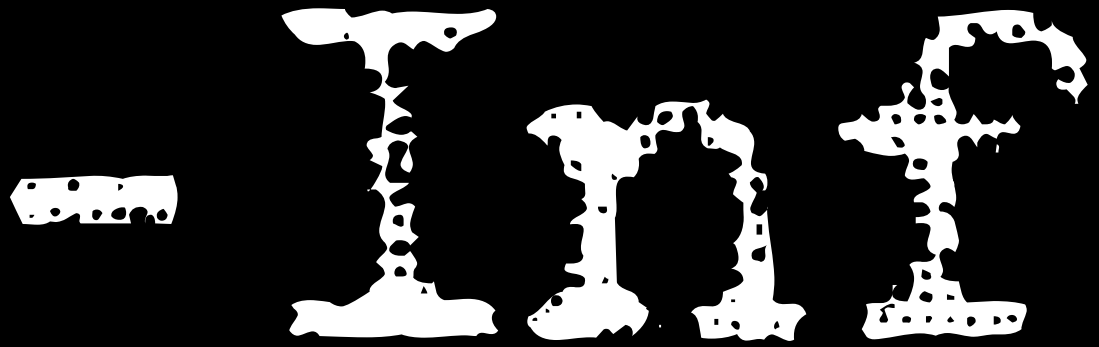
6 Impossible Things

@KevlinHenney

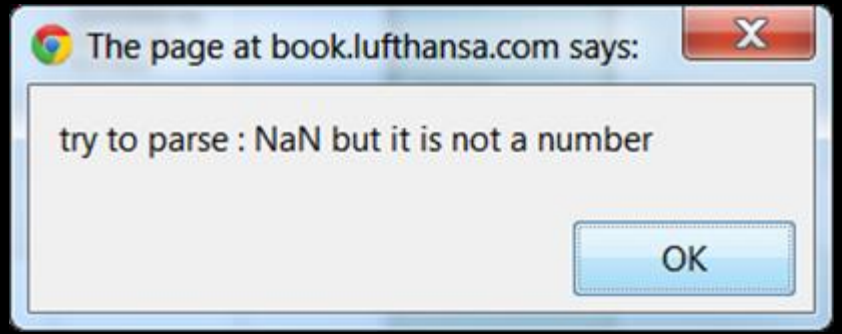
“Sometimes I’ve believed as many as six impossible things before breakfast.”

6 Representations can be infinite


+Imf


-Inf


NaN


The page at book.lufthansa.com says:
try to parse : NaN but it is not a number
OK

Driverless racecar drives straight into a wall

So during this initialization lap something happened which apparently caused the steering control signal to go to NaN and subsequently the steering locked to the maximum value to the right.

reddit.com/r/formula1/comments/jk9jrg/ot_roborace_driverless_racecar_drives_straight


NaNaNaNaNaNaN
NaNaNaNaNaNaN
NaNaNaNaNaNaN
NaNaNaNaNaNaN

Kevlin Henney
@KevlinHenney

Replying to @brayniverse @ignotus_ph and @Spotify

Perhaps that's what we should call these: Batman bugs 🤔

10:52 AM · Mar 6, 2021

♡ 1 Share this Tweet

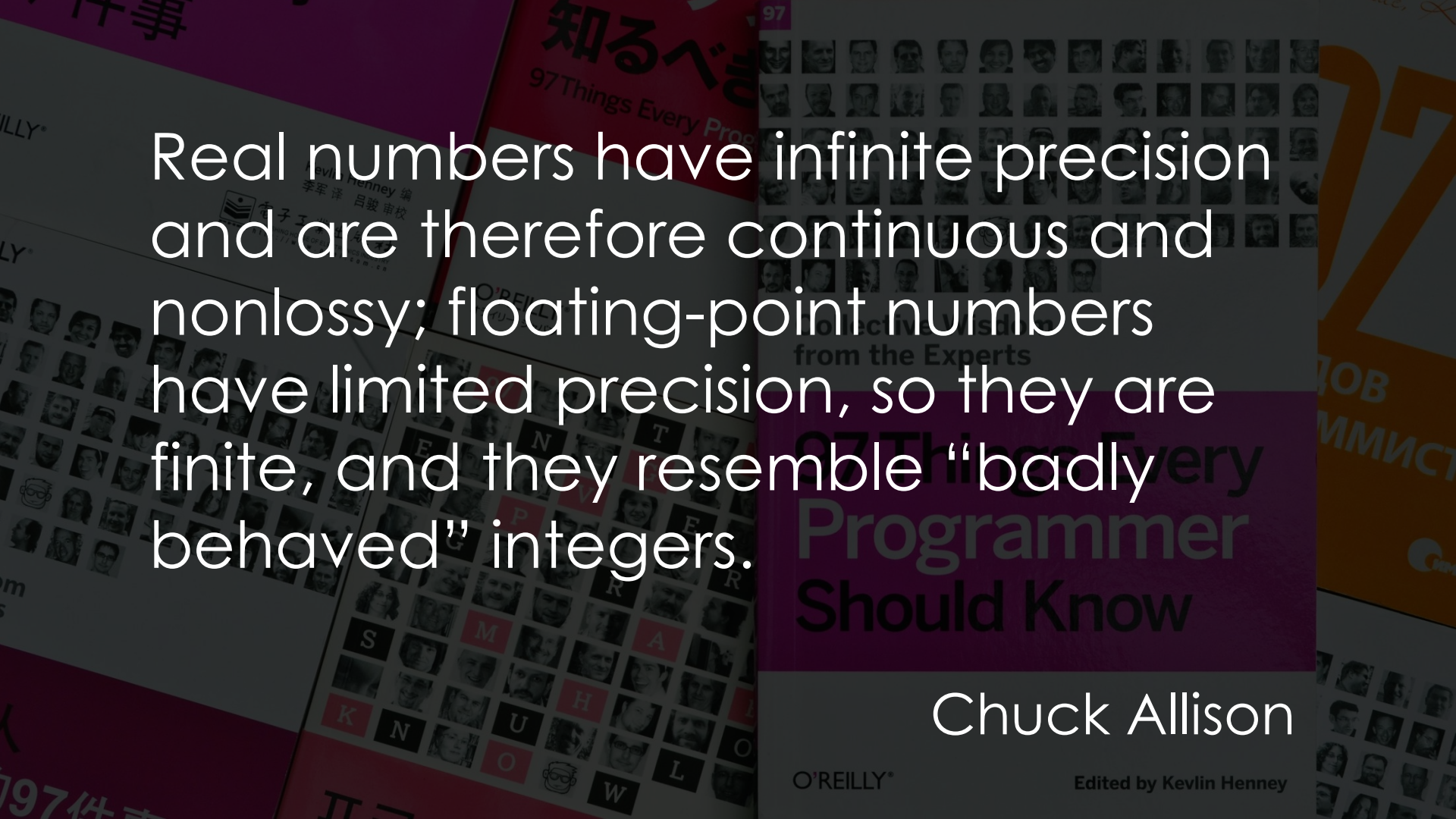
twitter.com/KevlinHenney/status/1368152437111001089


Kevlin Henney 编
李军 译 吕骏 审校
電子工業出版社
O'REILLY
オライリー・ジャパン
97
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
O'REILLY
Edited by Kevlin Henney


Floating-Point Numbers
Aren't Real
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
Chuck Allison
O'REILLY
Edited by Kevlin Henney

Real numbers have infinite precision and are therefore continuous and nonlossy; floating-point numbers have limited precision, so they are finite, and they resemble "badly behaved" integers.

Chuck Allison



Product Information

This license will expire in 2147483647 days.

⚠ Your license has gone stale and must be updated. Check for an updated license to continue using this product.

Check for an updated license

Product Information

This license will expire in 2147483647 days.

Your license has gone stale and must be updated. Check for an updated license to continue using this product.

Check for an updated license

Product Information

Visual Studio®

Professional 2013

This license will expire in 2147483647 days.

⚠ **Your license has gone stale and must be updated.** Check for an updated license to continue using this product.

Check for an updated license

```
int BinSearch(int x, const int *a, int n)
{  int middle, left=0, right=n-1;
   if (x <= a[left]) return 0;
   if (x > a[right]) return n;
   while (right - left > 1)
   {  middle = (right + left)/2;
      (x <= a[middle] ? right : left) = middle;
   }
   return right;
}
```

```
int BinSearch(int x, const int *a, int n)
{
    int middle, left=0, right=n-1;
    if (x <= a[left])
        return 0;
    if (x > a[right])
        return n;
    while (right - left > 1)
    {
        middle = (right + left)/2;
        (x <= a[middle] ? right : left) = middle;
    }
    return right;
}
```

```
int BinSearch(int x, const int *a, int n)
{
    int middle, left = 0, right = n - 1;
    if (x <= a[left])
        return 0;
    if (x > a[right])
        return n;
    while (right - left > 1)
    {
        middle = (right + left) / 2;
        (x <= a[middle] ? right : left) = middle;
    }
    return right;
}
```

```
int BinSearch(int x, const int *a, int n)
{
    int middle, left = 0, right = n - 1;
    if (x <= a[left])
        return 0;
    if (x > a[right])
        return n;
    while (right - left > 1)
    {
        middle = (right + left) / 2;
        (x <= a[middle] ? right : left) = middle;
    }
    return right;
}
```

```
int BinSearch(int x, const int *a, int n)
{
    int left = 0, right = n - 1;
    if (x <= a[left])
        return 0;
    if (x > a[right])
        return n;
    while (right - left > 1)
    {
        int middle = (right + left) / 2;
        (x <= a[middle] ? right : left) = middle;
    }
    return right;
}
```

programming pearls

By Jon Bentley

WRITING CORRECT PROGRAMS

In the late 1960s people were talking about the promise of programs that verify the correctness of other programs. Unfortunately, it is now the middle of the 1980s, and, with precious few exceptions, there is still little more than talk about automated verification systems. Despite unrealized expectations, however, the research on program verification has given us something far more valuable than a black box that gobbles programs and flashes "good" or "bad"—we now have a fundamental understanding of computer programming.

The purpose of this column is to show how that fundamental understanding can help programmers write correct programs. But before we get to the subject itself, we must keep it in perspective. Coding skill is just one small part of writing correct programs. The majority of the task is the subject of the three previous columns: problem definition, algorithm design, and data structure selection. If you perform those tasks well, then writing correct code is usually easy.

The Challenge of Binary Search

I've given this problem as an in-class assignment in courses at Bell Labs and IBM. The professional programmers had one hour (sometimes more) to convert the above description into a program in the language of their choice; a high-level pseudocode was fine. At the end of the specified time, almost all the programmers reported that they had correct code for the task. We would then take 30 minutes to examine their code, which the programmers did with test cases. In many different classes and with over a hundred programmers, the results varied little: 90 percent of the programmers found bugs in their code (and I wasn't always convinced of the correctness of the code in which no bugs were found).

I found this amazing: only about 10 percent of professional programmers were able to get this small program right. But they aren't the only ones to find this task difficult. In the history in Section 6.2.1 of his *Sorting and Searching*, Knuth points out that while the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962.

```
L:=1; U:=N
loop
    { MustBe(L,U) }
    if L>U then
        P:=0; break
    M := (L+U) div 2
    case
        X[M] < T:  L:=M+1
        X[M] = T:  P:=M; break
        X[M] > T:  U:=M-1
  endloop
```

```
{ MustBe(1,N) }
L := 1; U := N
{ MustBe(L,U) }
loop
    { MustBe(L,U) }
    if L>U then
        { L>U and MustBe(L,U) }
        { T is nowhere in the array }
        P := 0; break
    { MustBe(L,U) and L<=U }
    M := (L+U) div 2
    { MustBe(L,U) and L<=M<=U }
    case
      X[M] < T:
        { MustBe(L,U) and CantBe(1,M) }
        { MustBe(M+1,U) }
        L := M+1
        { MustBe(L,U) }
      X[M] = T:
        { X[M] = T }
        P := M; break
      X[M] > T:
        { MustBe(L,U) and CantBe(M,N) }
        { MustBe(L,M-1) }
        U := M-1
        { MustBe(L, U)}
    { MustBe(L,U) }
  endloop
```

Communications of the ACM, 26(12), December 1983

One of the major benefits of program verification is that it gives programmers a language in which they can express that understanding.

These techniques are only a small part of writing correct programs; keeping the code simple is usually the key to correctness.

On the other hand, several professional programmers familiar with these techniques have related to me an experience that is too common in my own programming: when they construct a program, the “hard” parts work the first time, while the bugs are in the “easy” parts.

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```


public.nrao.edu/telescopes/vla

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = low + ((high - low) / 2);
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
```

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

```
L:=1; U:=N
loop
    { MustBe(L,U) }
    if L>U then
        P:=0; break
    M := (L+U) div 2
    case
        X[M] < T:   L:=M+1
        X[M] = T:   P:=M; break
        X[M] > T:   U:=M-1
  endloop
```

Kevlin Henney
@KevlinHenney

Epistemologically speaking, assumptions are the barefoot-trodden Lego bricks in the dark of knowledge. You don't know they're there until you know that they're there. And even if you know there are some there, you don't know exactly where and you'll still end up stepping on some.

♡ 26 2:29 PM - Apr 22, 2020

twitter.com/KevlinHenney/status/1252952622128128000

```
int BinSearch(int x, const int *a, int n)
{  int middle, left=0, right=n-1;
   if (x <= a[left]) return 0;
   if (x > a[right]) return n;
   while (right - left > 1)
   {  middle = (right + left)/2;
      (x <= a[middle] ? right : left) = middle;
   }
   return right;
}
```

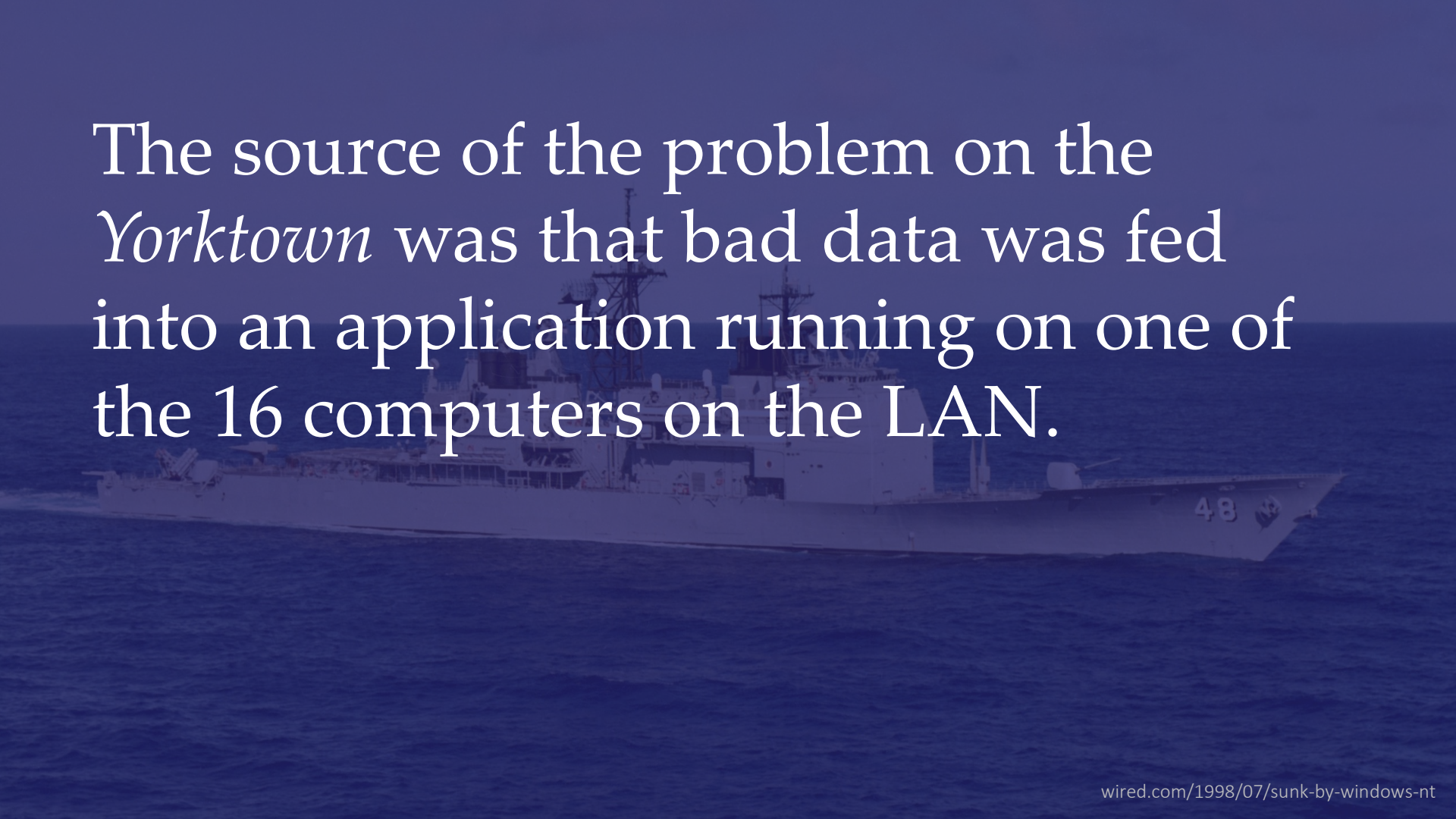
```
int BinSearch(int x, const int *a, int n)
{  int middle, left=0, right=n-1;
   if (x <= a[left]) return 0;
   if (x > a[right]) return n;
   while (right - left > 1)
   {  middle = (right + left)/2;
      (x <= a[middle] ? right : left) = middle;
   }
   return right;
}
```

```
int BinSearch(int x, const int *a, int n)
{  int middle, left=0, right=n-1;
   if (x <= a[left]) return 0;
   if (x > a[right]) return n;
   while (right - left > 1)
   {  middle = std::midpoint(left, right);
      (x <= a[middle] ? right : left) = middle;
   }
   return right;
}
```

```
int BinSearch(int x, const int *a, int n)
{
    return std::lower_bound(a, a + n, x) - a;
}
```

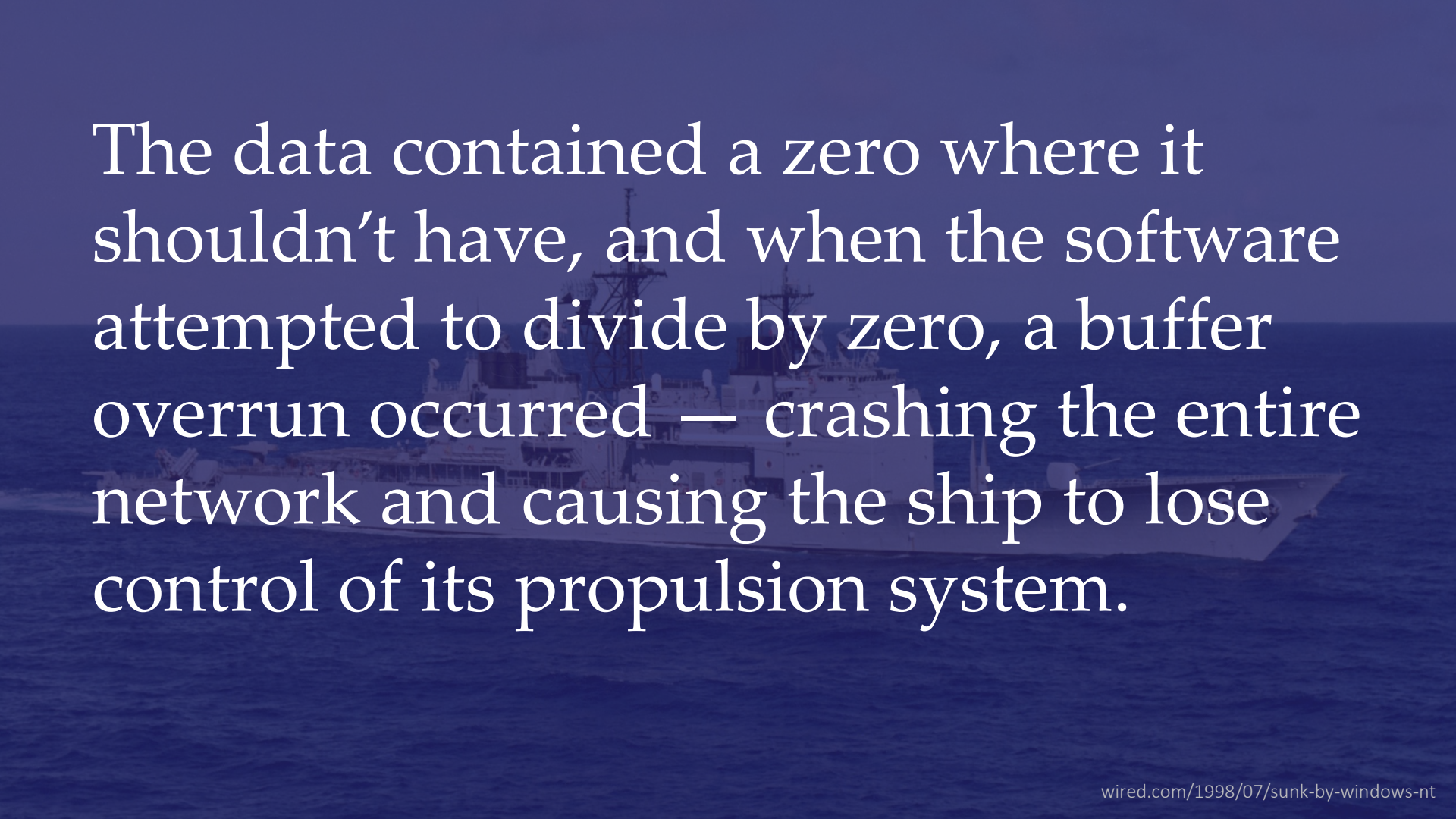
```
std::lower_bound(a, a + n, x)
```


48



The source of the problem on the *Yorktown* was that bad data was fed into an application running on one of the 16 computers on the LAN.

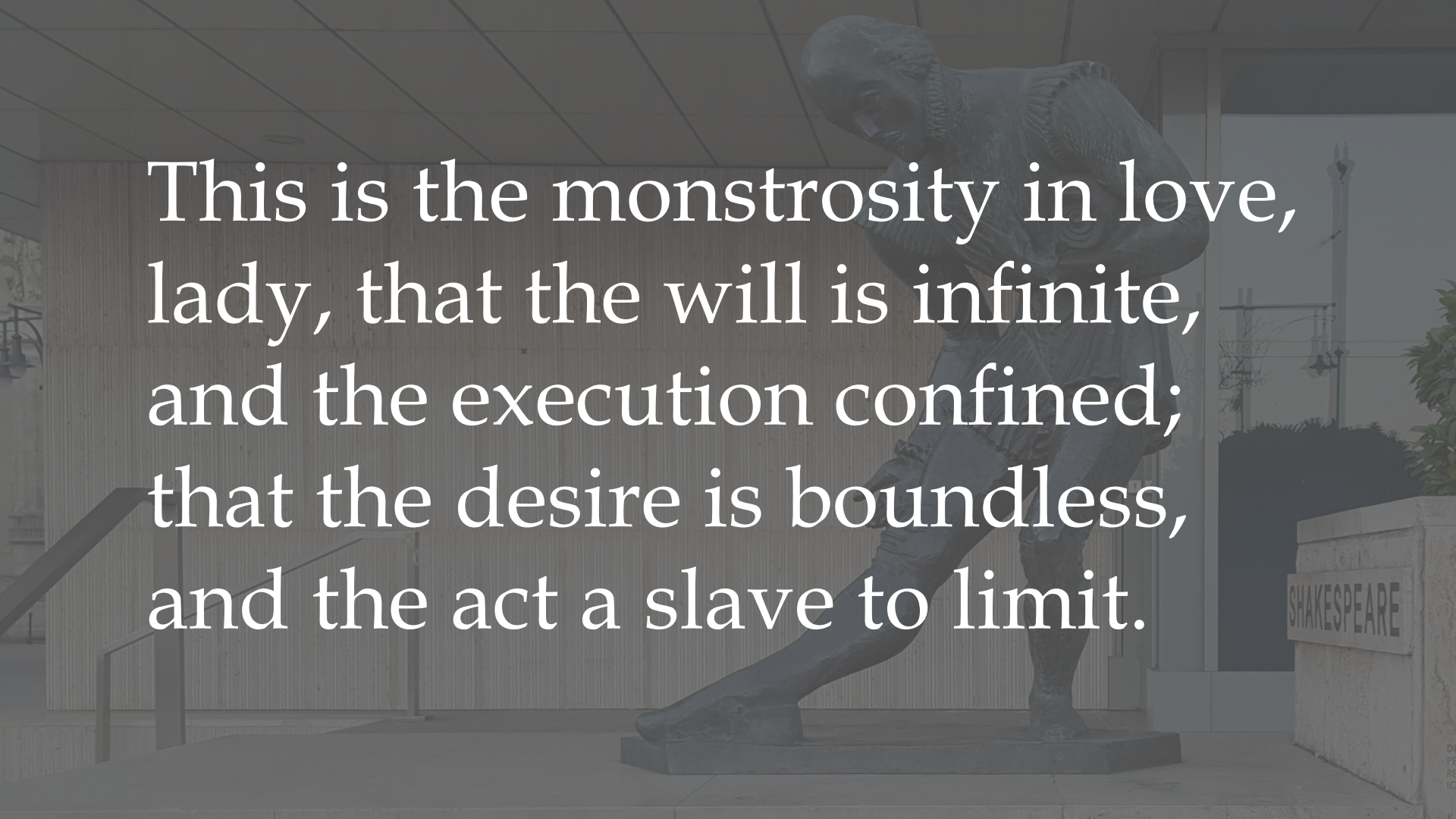
wired.com/1998/07/sunk-by-windows-nt



The data contained a zero where it shouldn’t have, and when the software attempted to divide by zero, a buffer overrun occurred — crashing the entire network and causing the ship to lose control of its propulsion system.

wired.com/1998/07/sunk-by-windows-nt


SHAKESPEARE



This is the monstrosity in love,
lady, that the will is infinite,
and the execution confined;
that the desire is boundless,
and the act a slave to limit.

5

Every question has an answer

Our Reply

31 December 1969

Your feedback will be used to improve Facebook. Thanks for taking the time to make a report.

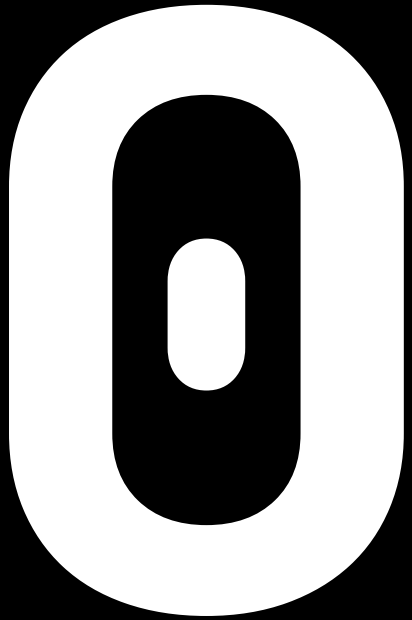
Our Reply

31 December 1969

Your feedback will be used to improve Facebook. Thanks for taking the time to make a report.

The `time` function shall return the value of time in seconds since the Epoch.

IEEE Std 1003.1-2017



Optional updates

Choose the updates you want and then select Download and install.

Driver updates

If you have a specific problem, one of these drivers might help.

- [ ] INTEL - System - 10/3/2016 12:00:00 AM - 10.1.1.38
- [ ] INTEL - System - 1/1/1970 12:00:00 AM - 10.1.1.42
- [ ] Intel - System - 4/12/2017 12:00:00 AM - 14.28.47.630

twitter.com/craiu/status/1395269207345057798

Anil Dash
@anildash

The natural enemy of the programmer is the timezone.

5:32 AM · Dec 16, 2019

1.4K 333 Share this Tweet

twitter.com/anildash/status/1206447093872812033

Our Reply

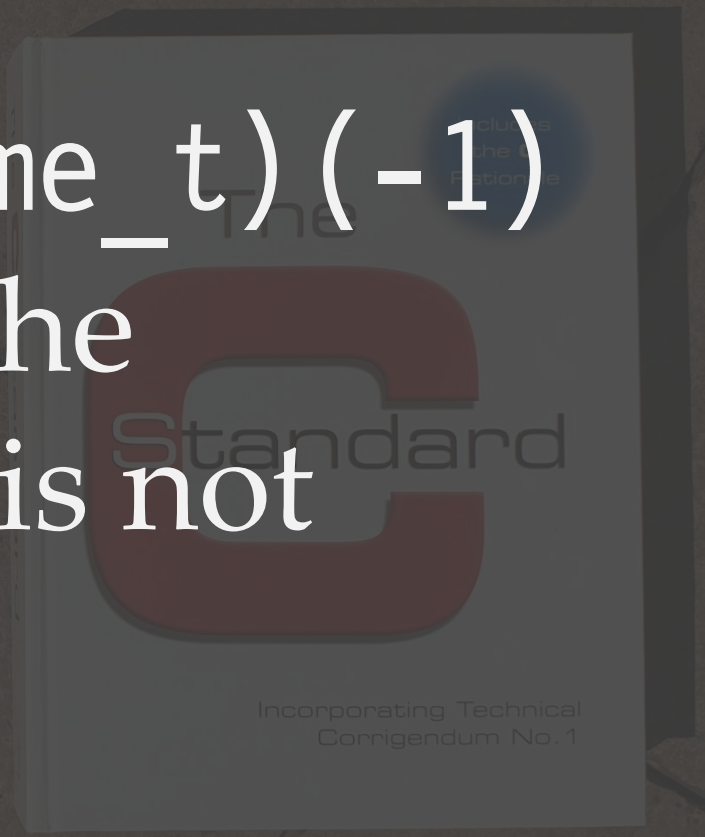
31 December 1969

Your feedback will be used to improve Facebook. Thanks for taking the time to make a report.

-1


includes
the C
Rationale
The
C
Standard
Incorporating Technical
Corrigendum No.1

The value `(time_t)(-1)` is returned if the calendar time is not available.



الـخـوارزمـی

al-Khwarizmi

Muhammad ibn Musa al-Khwarizmi

Algorithmi

algorithmus

algorithm

a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer

algorithm

Concise Oxford English Dictionary

<algorithm>

Permutation sort takes us to $O(n!)$ — that's right, factorial time. $O(MG)$!

In essence, it is an unoptimised search through the permutations of the input values until it finds the one arrangement that is sorted.

Kevlin Henney
"A Sort of Permutation"
kevlinhenney.medium.com/a-sort-of-permutation-768c1a7e029b

```
void permutation_sort(auto begin, auto end)
{
    while (std::next_permutation(begin, end))
        ;
}
```

Surely, there can be nothing worse than permutation sort in terms of performance?

Don’t be so sure.

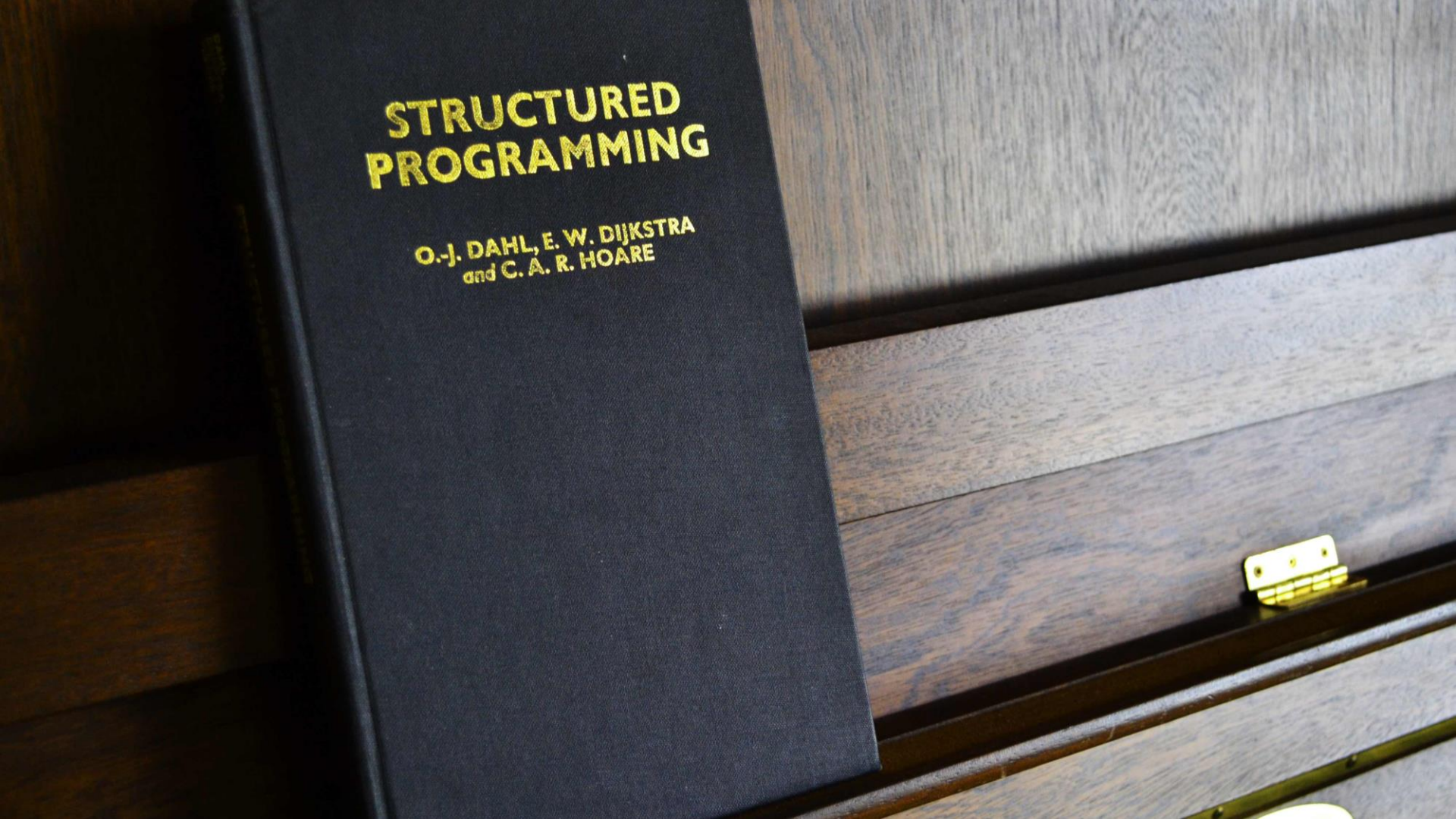
The essence of bogosort is to shuffle the values randomly until they are sorted.

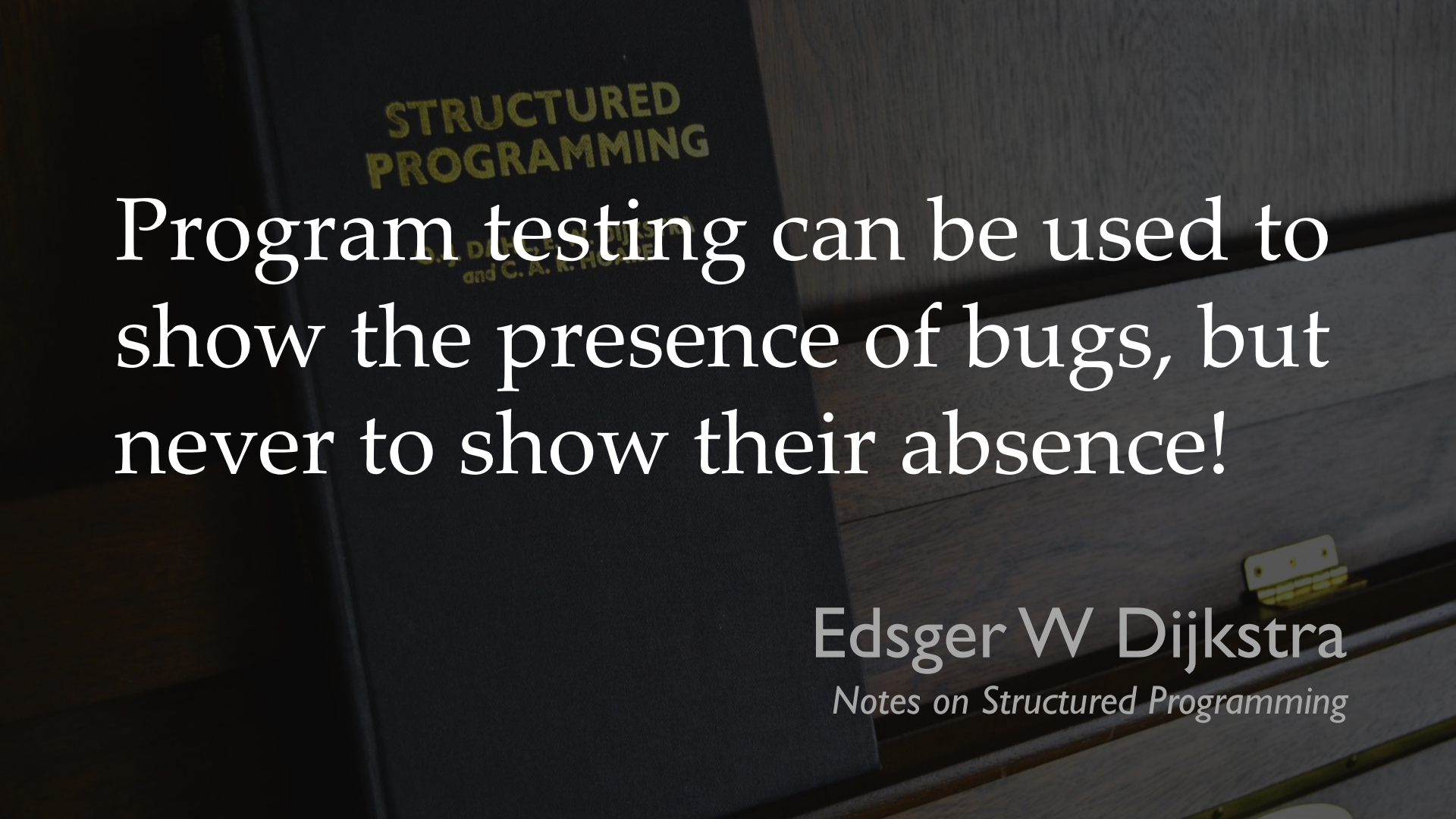
Kevlin Henney
kevlinhenney.medium.com

```
void bogosort(auto begin, auto end)
{
    std::random_device randomness;
    do
        std::shuffle(begin, end, randomness);
    while (!std::is_sorted(begin, end));
}
```

A procedure which always terminates is called an *algorithm*.

John E Hopcroft & Jeffrey D Ullman
Formal Languages and their Relation to Automata


STRUCTURED
PROGRAMMING
O.-J. DAHL, E. W. DIJKSTRA
and C. A. R. HOARE



Program testing can be used to show the presence of bugs, but never to show their absence!

Edsger W Dijkstra

Notes on Structured Programming

```
std::vector actual {3, 1, 4, 1, 5, 9};

bogosort(actual.begin(), actual.end());
assert(std::is_sorted(actual.begin(), actual.end()));
```

```
std::vector actual {3, 1, 4, 1, 5, 9};
const auto copy = actual;
bogosort(actual.begin(), actual.end());
assert(std::is_sorted(actual.begin(), actual.end());
assert(
    std::is_permutation(
        actual.begin(), actual.end(), copy.begin()));
```

```
std::vector actual {3, 1, 4, 1, 5, 9};
const std::vector expected {1, 1, 3, 4, 5, 9};
bogosort(actual.begin(), actual.end());
assert(actual == expected);
```

```
std::signal(SIGALRM, [](int) {assert(false);});
alarm(???);
std::vector actual {3, 1, 4, 1, 5, 9};
const std::vector expected {1, 1, 3, 4, 5, 9};
bogosort(actual.begin(), actual.end());
assert(actual == expected);
```

```
std::signal(SIGALRM, [](int) {assert(false);});
alarm(ENDOFTIME);
std::vector actual {3, 1, 4, 1, 5, 9};
const std::vector expected {1, 1, 3, 4, 5, 9};
bogosort(actual.begin(), actual.end());
assert(actual == expected);
```

```
std::signal(SIGALRM, [](int) {assert(false);});
alarm(1);
std::vector actual {3, 1, 4, 1, 5, 9};
const std::vector expected {1, 1, 3, 4, 5, 9};
bogosort(actual.begin(), actual.end());
assert(actual == expected);
```

Halting Problem


Baking
Problem

4 Every truth can be established where it applies

On Formally Undecidable Propositions Of Principia Mathematica And Related Systems

KURT GÖDEL

Translated by
B. MELTZER

Introduction by
R B. BRAITHWAITE

In 1911 Russell & Whitehead published Principia Mathematica, with the goal of providing a solid foundation for all of mathematics.

In 1931 Gödel's Incompleteness Theorem shattered the dream, showing that for any consistent axiomatic system there will always be theorems that cannot be proven within the system.

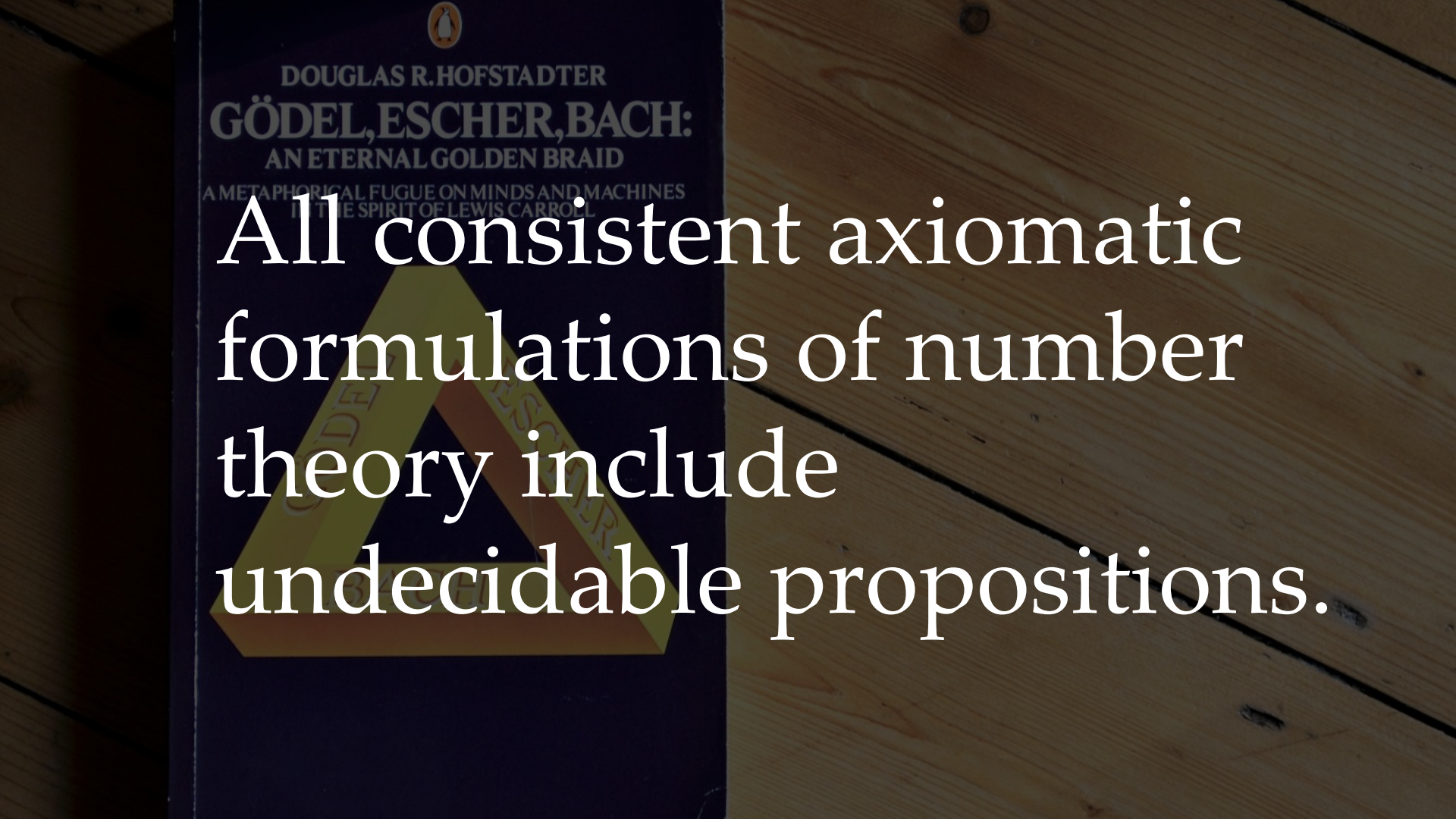
Adrian Colyer

blog.acolyer.org/2020/02/03/measure-mismeasure-fairness/

DOUGLAS R. HOFSTADTER

GÖDEL, ESCHER, BACH:
AN ETERNAL GOLDEN BRAID

A METAPHORICAL FUGUE ON MINDS AND MACHINES
IN THE SPIRIT OF LEWIS CARROLL



All consistent axiomatic formulations of number theory include undecidable propositions.

undecidable propositions

How long is a piece of string?

```
size_t strlen(const char * s);
```

```
size_t strlen(const char * s)
{
    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
```

```
size_t strlen(const char * s)
{
    assert(s != NULL);

    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
```

```
size_t strlen(const char * s)
{
    assert(s != NULL);
    assert(∃n (s[n] == '\0') &&
           ∀i∈0..n (s[i] is defined));

    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
```

```
void well_defined(void)
{
    char s[] = "Be excellent to each other";
    printf("%s\n%zu\n", s, strlen(s));
}
```

Be excellent to each other
26

```
void insufficient_space(void)
{
    char s[5] = {'B', 'o', 'g', 'u', 's'};
    printf("%s\n%zu\n", s, strlen(s));
}
```

```
void insufficient_space(void)
{
    char s[5] = "Bogus";
    printf("%s\n%zu\n", s, strlen(s));
}
```

Bogus
5

```
void undefined_pointer_value(void)
{
    char * s;
    printf("%s\n%zu\n", s, strlen(s));
}
```

One premise of many models of fairness in machine learning is that you can measure ('prove') fairness of a machine learning model from within the system – i.e. from properties of the model itself and perhaps the data it is trained on.

To show that a machine learning model is fair, you need information from outside of the system.

Adrian Colyer

blog.acolyer.org/2020/02/03/measure-mismeasure-fairness/


99 second hand smartphones
are transported in a handcart
to generate virtual traffic jam
in Google Maps.
Simon Weckert
simonweckert.com/googlemapshacks.html

engagement

the state of being engaged is

engagement

engagement

is emotional involvement or commitment

“engagement”

clicks & shares

We must be careful not to confuse data with the abstractions we use to analyse them.

William James

3 The future is knowable before it happens

To me programming is more than an important practical art. It is also a gigantic undertaking in the foundations of knowledge.

Grace Hopper

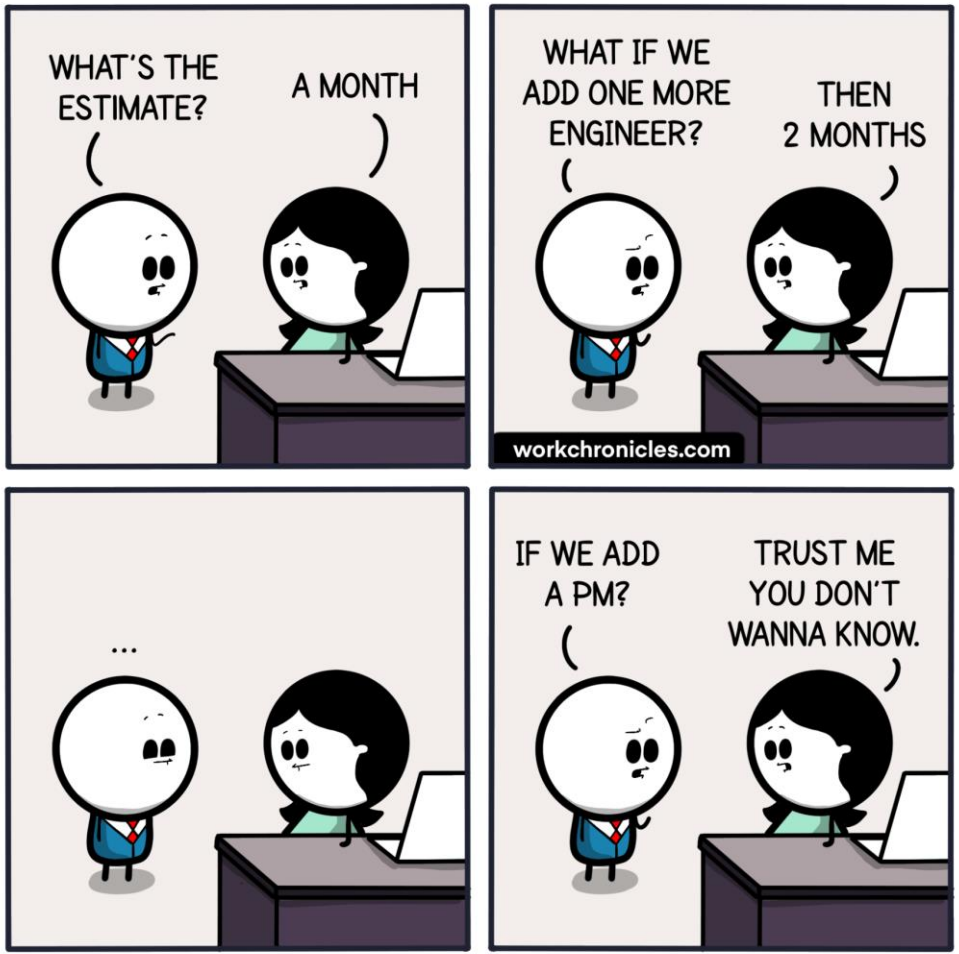
known knowns
known unknowns
unknown unknowns
unknowable unknowns

known knowns
known unknowns
unknown unknowns
unknowable unknowns

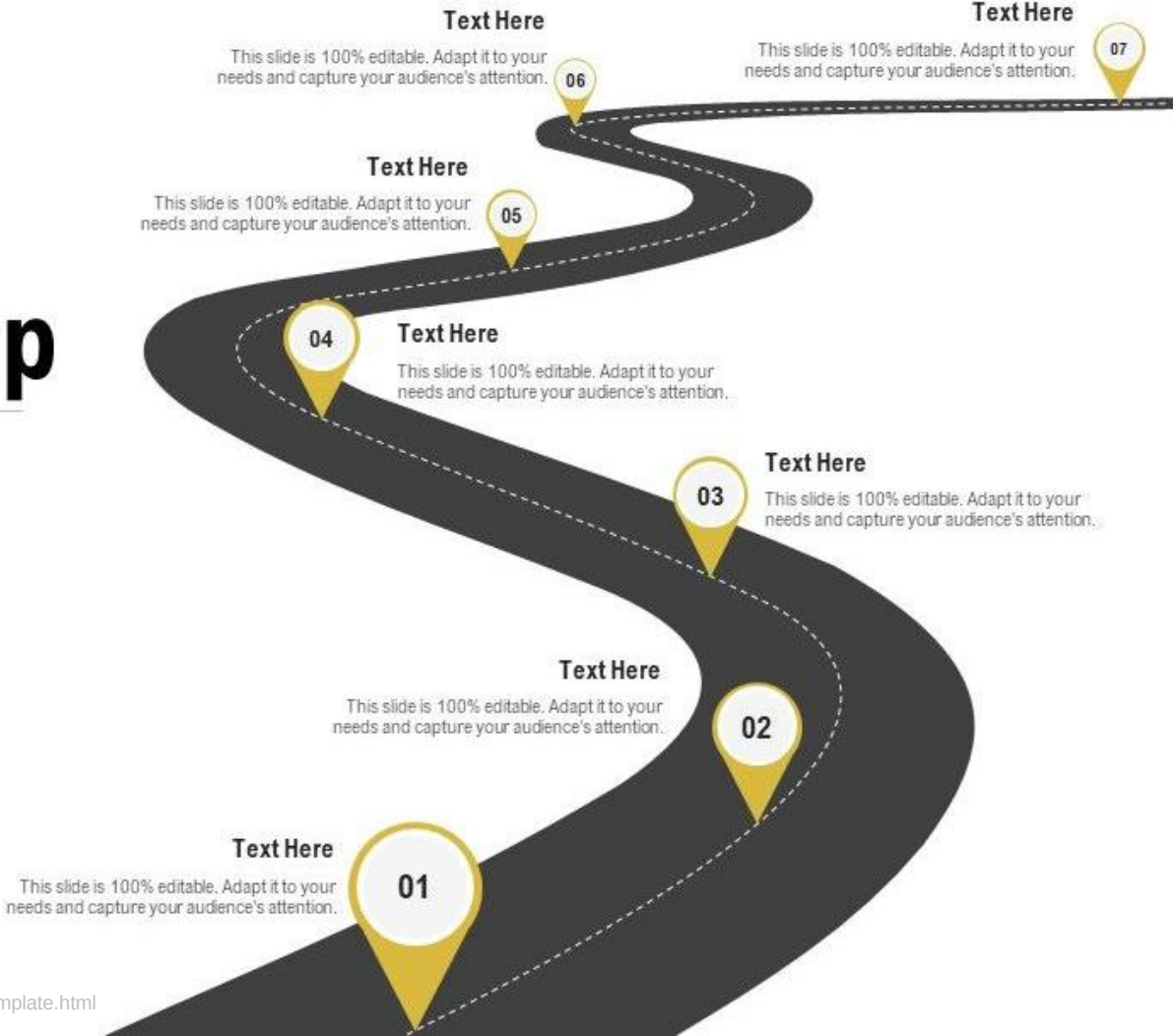
Halting Problem

Prediction is very difficult, especially about the future.

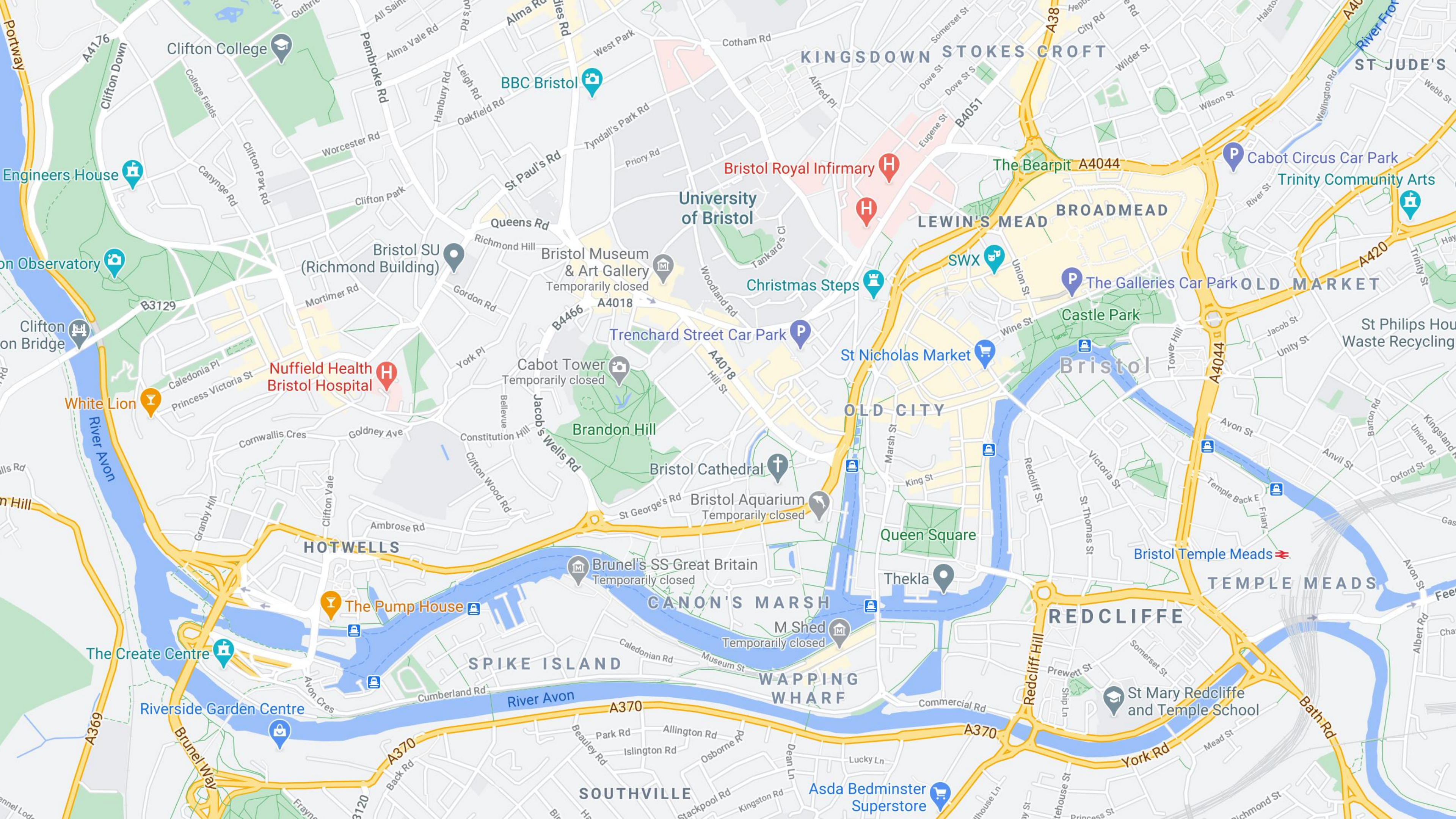
Niels Bohr?


WHAT'S THE ESTIMATE?
A MONTH
WHAT IF WE ADD ONE MORE ENGINEER?
THEN 2 MONTHS
workchronicles.com
...
IF WE ADD A PM?
TRUST ME YOU DON'T WANNA KNOW.

Roadmap

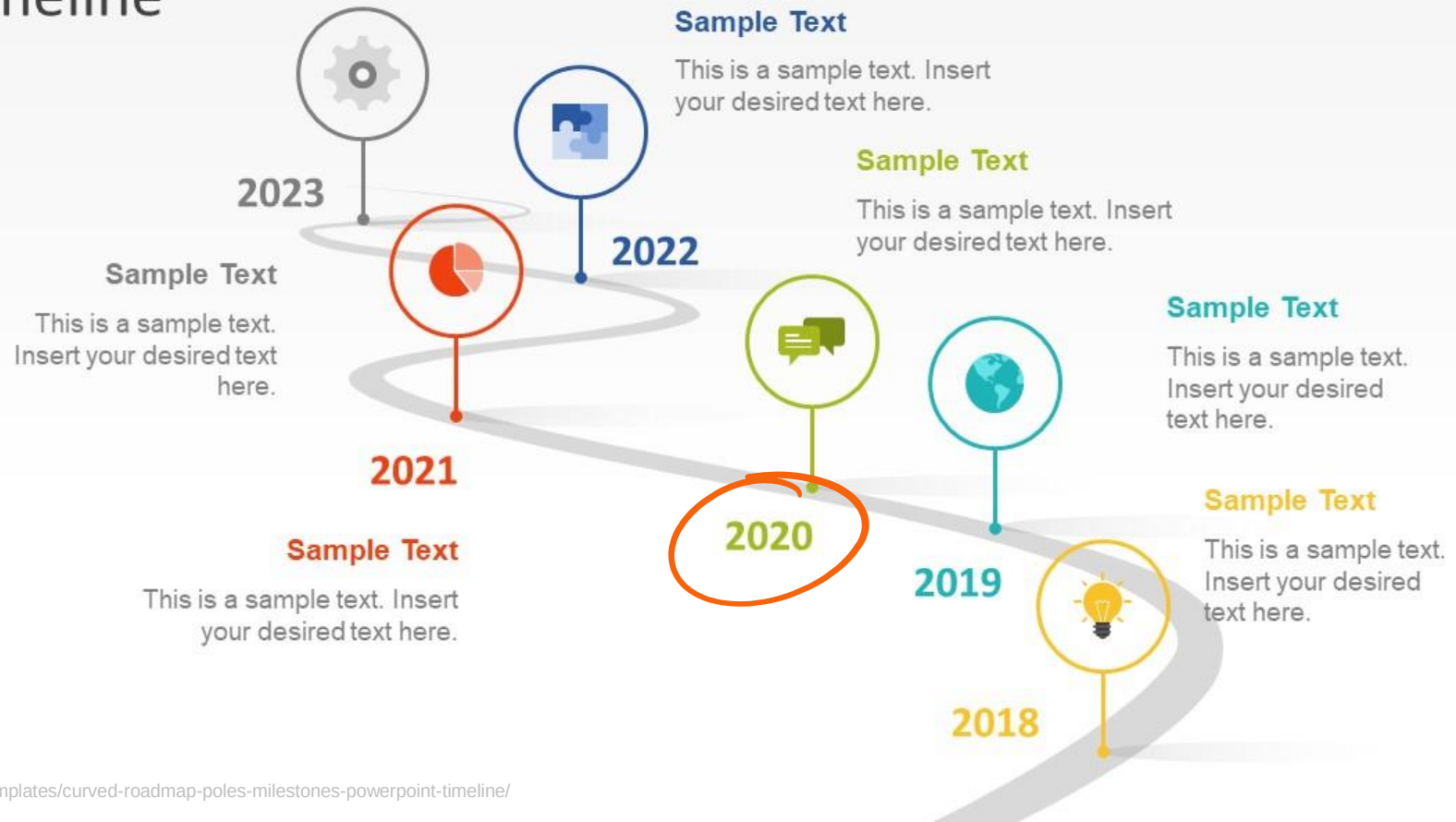


slideteam.net/roadmap-seven-c1304-ppt-powerpoint-presentation-infographic-template.html



Clifton College
BBC Bristol
Engineers House
Clifton Observatory
Bristol SU (Richmond Building)
Bristol Museum & Art Gallery
Temporarily closed
University of Bristol
Bristol Royal Infirmary
Christmas Steps
Trenchard Street Car Park
Cabot Tower
Temporarily closed
Brandon Hill
Nuffield Health Bristol Hospital
White Lion
Bristol Cathedral
Bristol Aquarium
Temporarily closed
Queen Square
Brunel's SS Great Britain
Temporarily closed
The Pump House
The Create Centre
Riverside Garden Centre
M Shed
Temporarily closed
Thekla
St Nicholas Market
Castle Park
The Galleries Car Park
SWX
The Bearpit
Cabot Circus Car Park
Trinity Community Arts
Bristol Temple Meads
St Mary Redcliffe and Temple School
Asda Bedminster Superstore
KINGSDOWN
STOKES CROFT
ST JUDE'S
LEWIN'S MEAD
BROADMEAD
OLD MARKET
OLD CITY
Bristol
HOTWELLS
CANON'S MARSH
SPIKE ISLAND
WAPPING WHARF
REDCLIFFE
TEMPLE MEADS
SOUTHVILLE
River Avon
River Avon
Portway
A4176
Clifton Down
Pembroke Rd
Alma Vale Rd
All Saints
West Park
Cotham Rd
Worcester Rd
Oakfield Rd
St Paul's Rd
Tyndall's Park Rd
Priory Rd
Clifton Park
Queens Rd
Richmond Hill
Mortimer Rd
Gordon Rd
B3129
B4466
A4018
A4018
A4044
A4044
A38
B4051
A420
A370
A370
A370
A369
York Pl
Caledonia Pl
Princess Victoria St
Cornwallis Cres
Goldney Ave
Constitution Hill
Jacob's Wells Rd
Clifton Wood Rd
St George's Rd
Ambrose Rd
Granby Hill
Clifton Vale
Woodland Rd
Tankard's Cl
Hill St
Union St
Wine St
Marsh St
King St
Redcliff St
Victoria St
St Thomas St
Redcliff Hill
Temple Back E
Avon St
Anvil St
Oxford St
Jacob St
Unity St
Tower Hill
Wilson St
Wilder St
Eugene St
Dove St
Alfred Pl
Wellington Rd
River St
Caledonian Rd
Museum St
Cumberland Rd
Commercial Rd
Prewett St
Somerset St
York Rd
Bath Rd
Mead St
Park Rd
Allington Rd
Islington Rd
Beauley Rd
Osborne Rd
Dean Ln
Lucky Ln
Kingston Rd
Stackpool Rd
Back Rd
Brunel Way
Avon Cres
College Fields
Canynge Rd
Clifton Park Rd
Hanbury Rd
Leigh Rd
Bellevue
Barton Rd
Union Rd
Albert Rd
Friary

Curved Roadmap with Poles Milestones PowerPoint Timeline

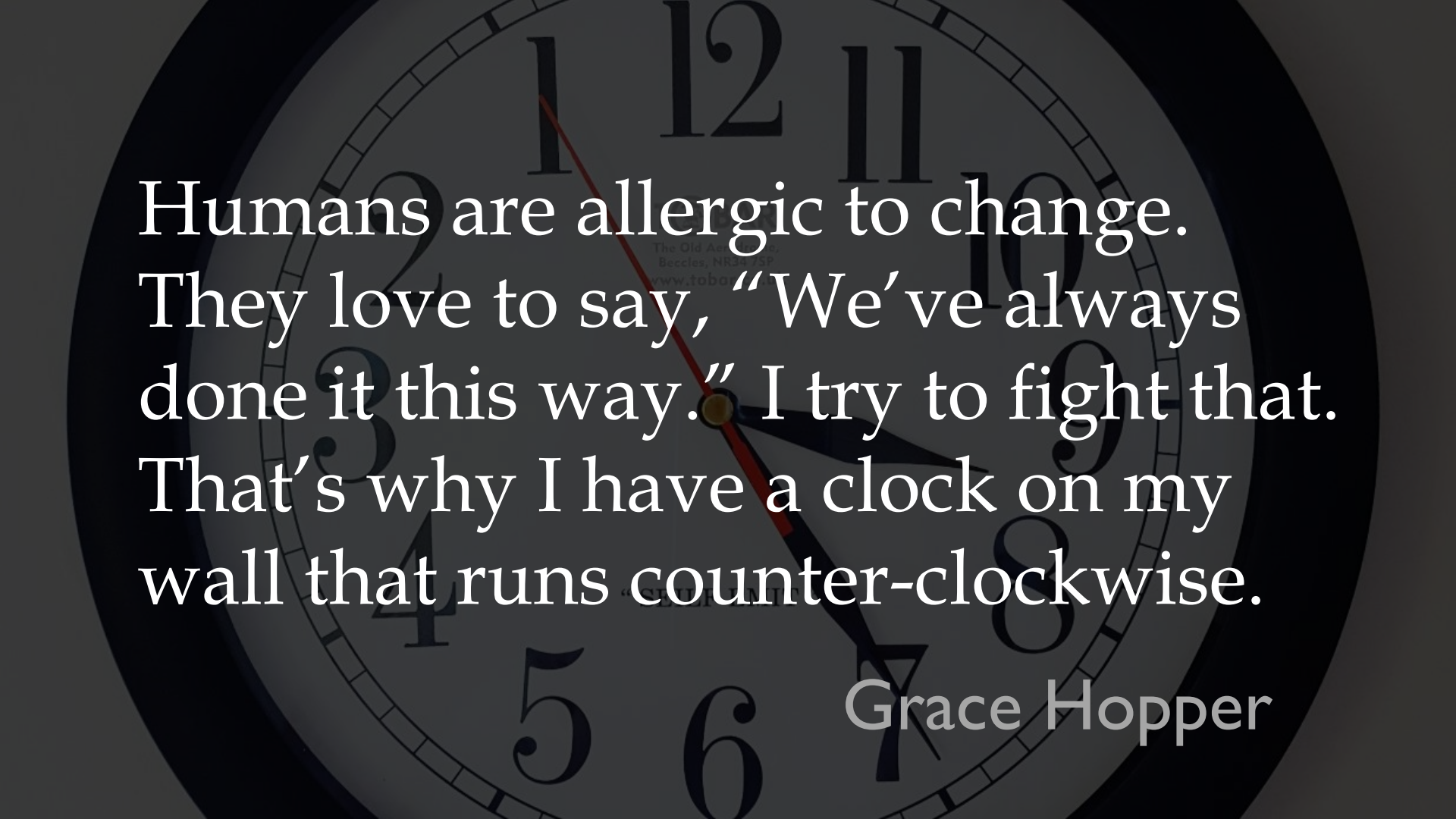


slidemodel.com/templates/curved-roadmap-poles-milestones-powerpoint-timeline/


prioritise by
business value


POLICE
PUBLIC
CALL
BOX
FREE
PUBLIC
PULL TO OPEN
SPACE
INVADERS
PLAYER
01

prioritise by estimated business value



Humans are allergic to change. They love to say, “We’ve always done it this way.” I try to fight that. That’s why I have a clock on my wall that runs counter-clockwise.

Grace Hopper


TOBAR
The Old Aerodrome,
Beccles, NR34 7SP
www.tobar.co.uk
" SEILF EMIT "

2

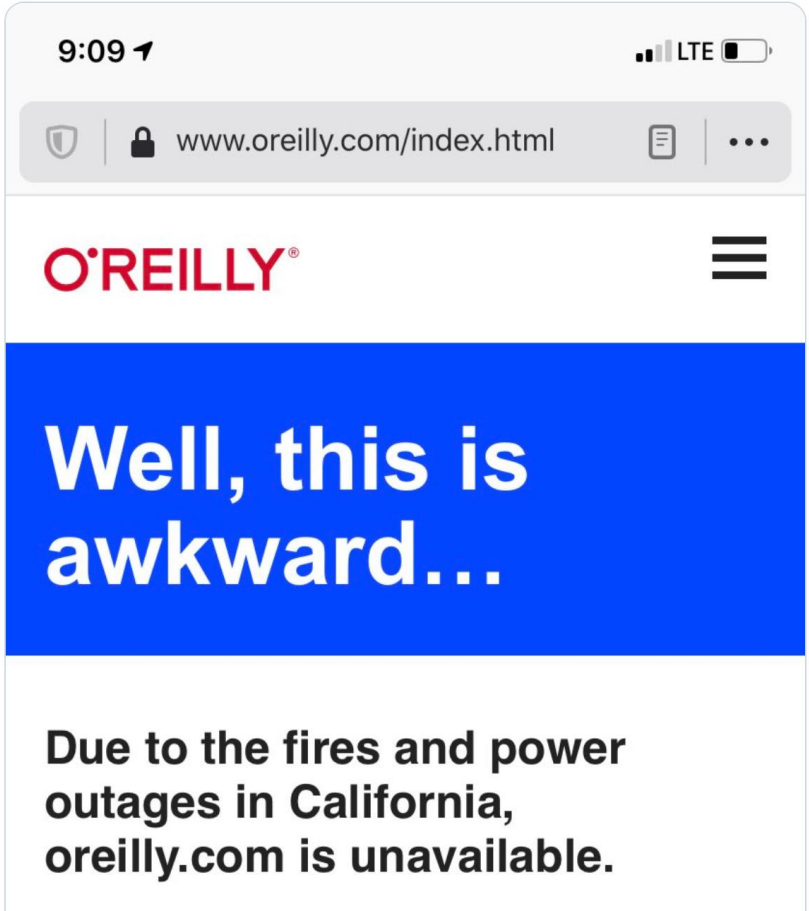
A distributed system is knowable

A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.

Leslie Lamport

Charlie Morris
@cdmo

Fire in California, can't read your ebook in Pennsylvania



twitter.com/cdmo/status/1188806227079389184

Brewer's theorem

CAP theorem

C

A

P

Consistency
Availability
Partition tolerance

Consistency

Availability

Partition tolerance

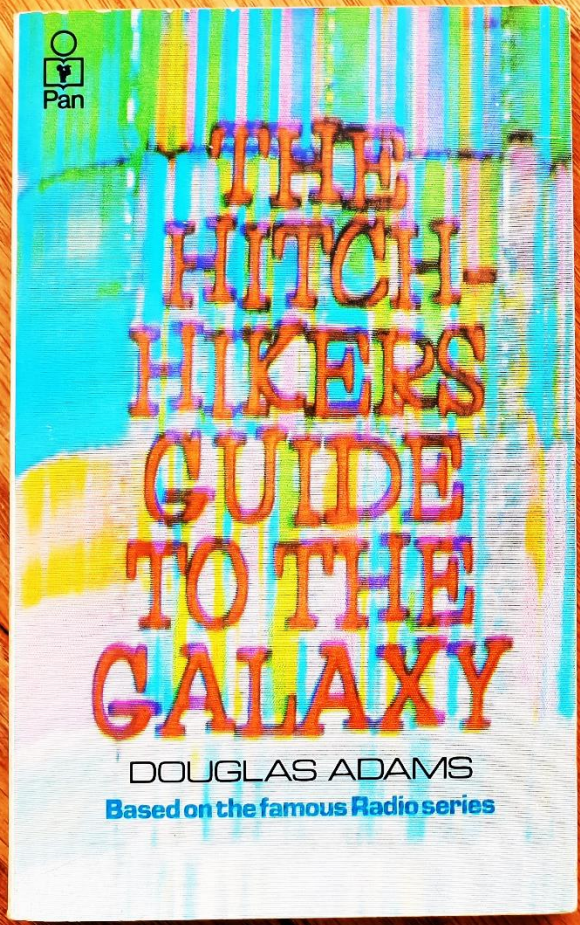
Consistency

Availability

Partition tolerance

Consistency
Availability
Partition tolerance

$$\Delta x \, \Delta p \geq \frac{\hbar}{2}$$


Pan
THE HITCH-HIKERS GUIDE TO THE GALAXY
DOUGLAS ADAMS
Based on the famous Radio series

We demand rigidly defined areas of doubt and uncertainty!

It is a feature of a distributed system that it may not be in a consistent state, but it is a bug for a client to contradict itself.

twitter.com/KevlinHenney/status/1351956942877552646

Technical debt is quantifiable as financial debt

As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.

Meir M Lehman

Programs, life cycles, and laws of software evolution

Technical Debt is a wonderful metaphor developed by Ward Cunningham to help us think about this problem.

Martin Fowler

martinfowler.com/bliki/TechnicalDebt.html

Like a financial debt, the technical debt incurs interest payments, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice.

Martin Fowler

martinfowler.com/bliki/TechnicalDebt.html

Technical Debt is a wonderful metaphor developed by Ward Cunningham to help us think about this problem.

Martin Fowler

martinfowler.com/bliki/TechnicalDebt.html

metaphor

Found myself again cautioning against the category error of treating the technical debt metaphor literally and numerically: converting code quality into a currency value on a dashboard.

Kevlin Henney

twitter.com/KevlinHenney/status/1265766381692846080

technical debt = cost of repaying the debt

technical debt ≠ cost of repaying the debt

technical debt = cost of owning the debt

That is the message of the technical debt metaphor: it is not simply a measure of the specific work needed to repay the debt; it is the additional time and effort added to all past, present, and future work that comes from having the debt in the first place.

Kevlin Henney

"On Exactitude in Technical Debt"

oreilly.com/radar/on-exactitude-in-technical-debt/

0