ARTIPELLAG
ART EVENT
OF THE YEAR!

the art of the javascript metaobject protocol

# Duck Typing, Compatibility, and The Adaptor Pattern

20

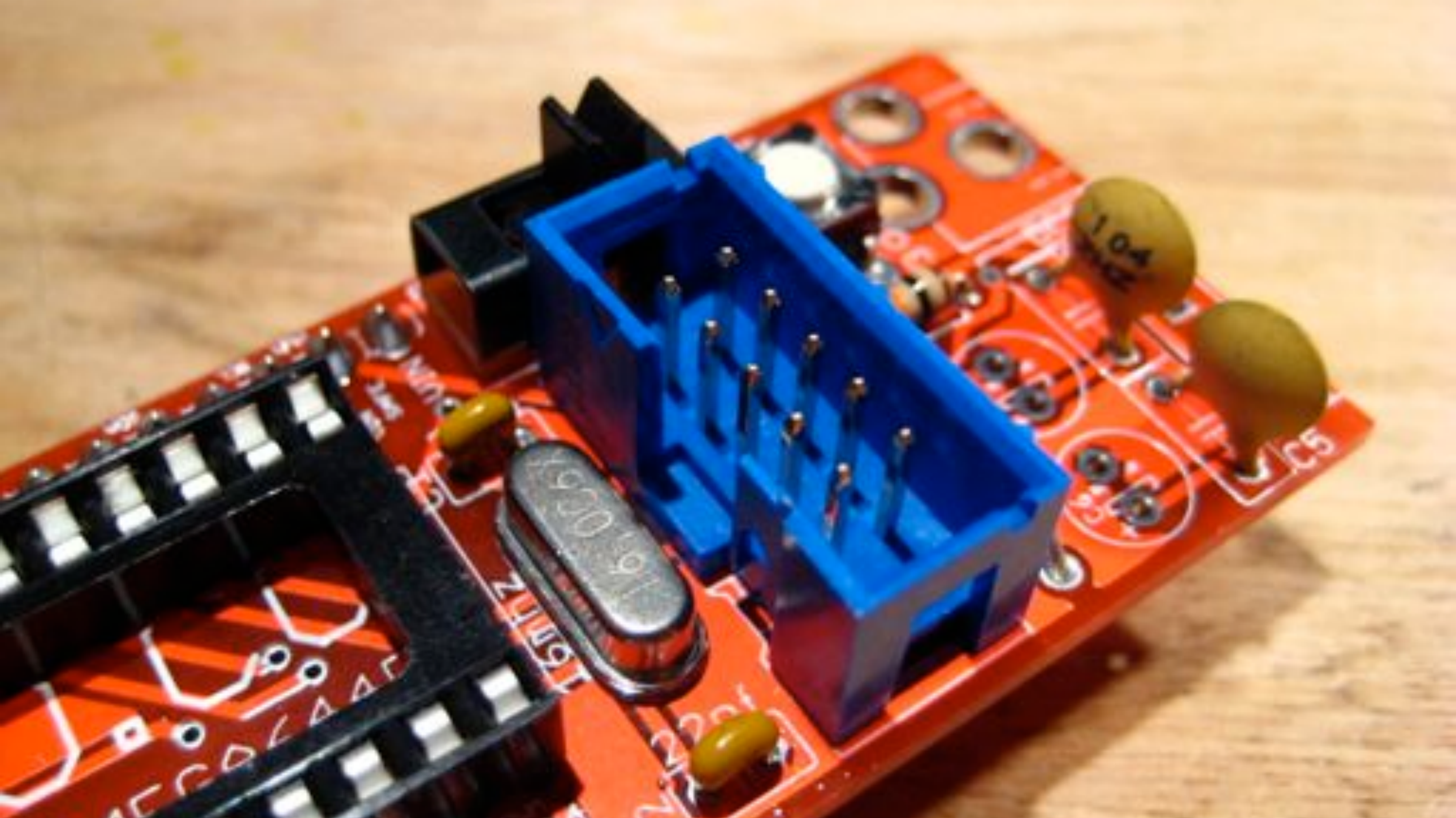
Duck typing is a style of typing in which an object's methods and properties determine the valid semantics,

…rather than its inheritance from a particular class or implementation of an explicit interface.

Tastes Like Fruit

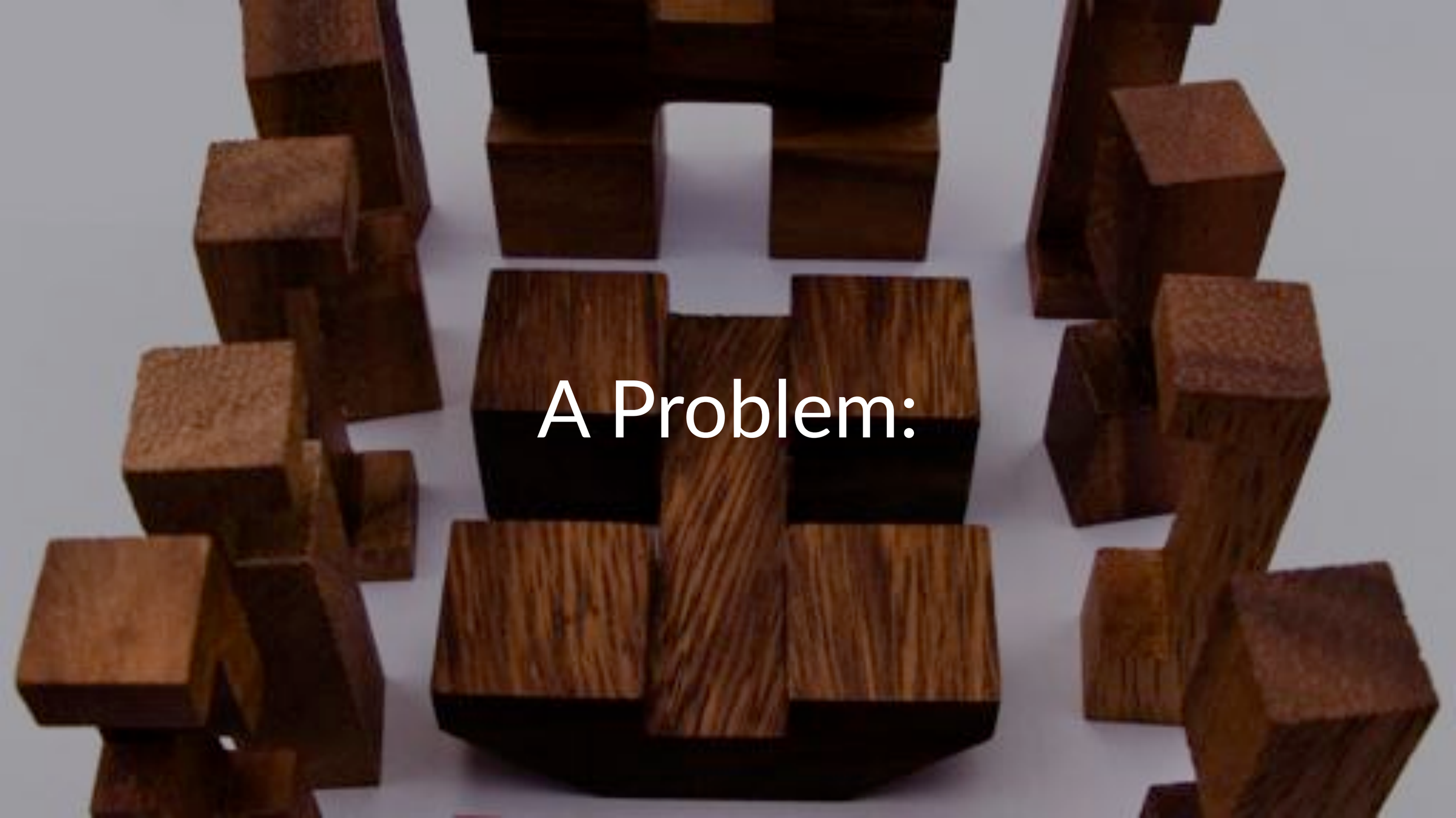
Also Tastes Like Fruit

"Write code that depends upon tastes-like(fruit) rather than depending upon is-a(fruit)."

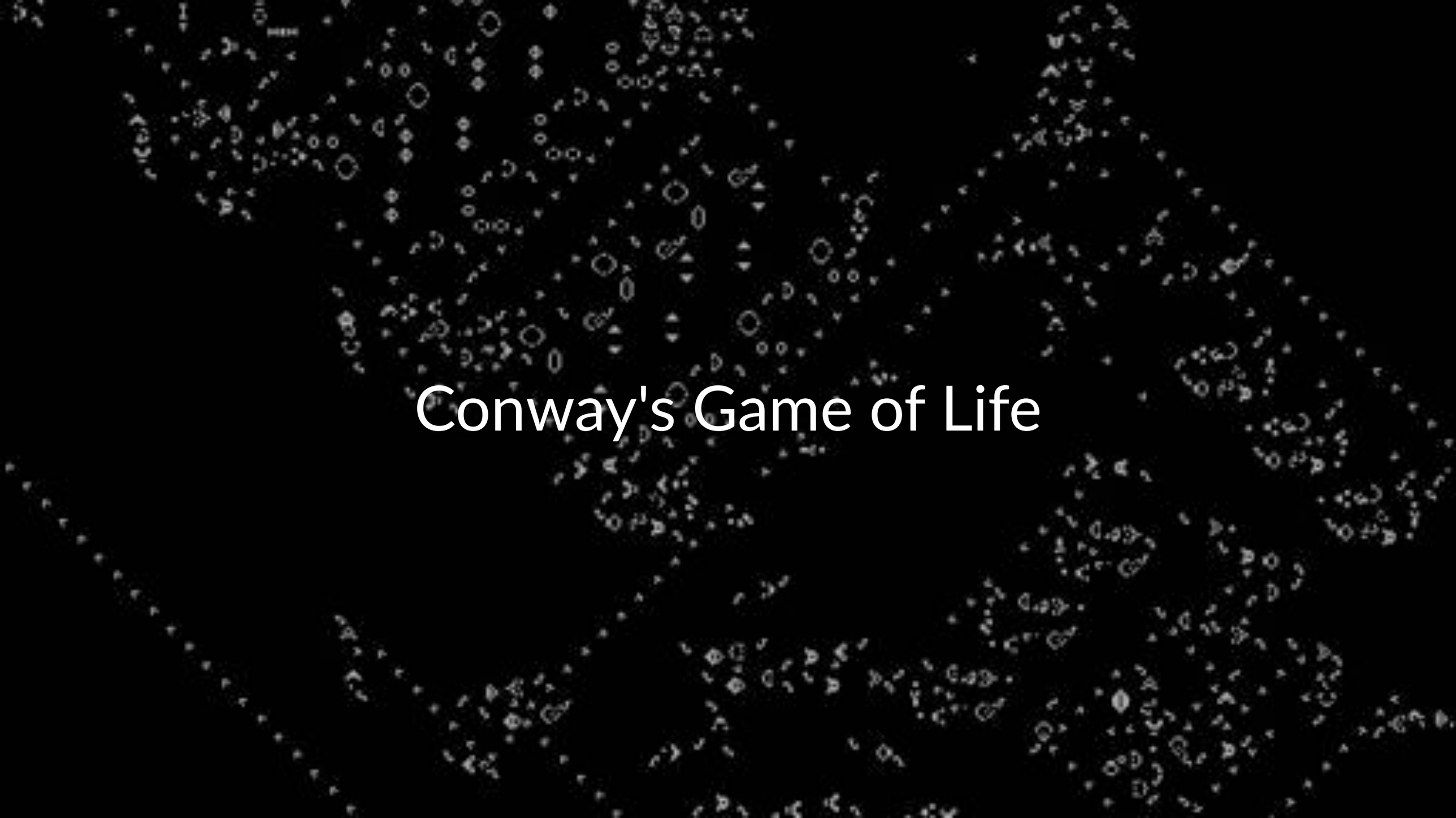


duck typing is a

# Compatibility Technique



# A Problem:



# Conway's Game of Life

"Conway's Game of Life is a two-state cellular automata."

# An Implementation

1. Cells,

# An Implementation

1. Cells,
2. Universe,

# An Implementation

1. Cells,
2. Universe, and
3. View.

# Cell

1. Neighbours,

# Cell

1. Neighbours, and
2. State.

```
function StandardCell () {
  this._neighbours = [];
  this._alive      = false;
}
```

```
StandardCell.prototype.neighbours =
  function neighbours (neighbours) {
    return this._neighbours;
  };

StandardCell.prototype.setNeighbours =
  function setNeighbours (neighbours) {
    this._neighbours = neighbours.slice(0);
    return this;
  };
```

```
StandardCell.prototype.alive =
  function alive () {
    return this._alive;
  };

StandardCell.prototype.setAlive =
  function setAlive (alive) {
    this._alive = alive;
    return this;
  };
```

Today
4 billion years
11 billion ye

moving through

# Time

# Cell

1. Neighbours,
2. State, and
3. Transformation of State.

# Universe

1. Neighbourhoods (not shown),

# Universe

1. Neighbourhoods (not shown), and
2. Transformation of States.

```
StandardCell.prototype.nextAlive =
  function nextAlive () {
    var alives =
      this._neighbours.filter(function (n) {
        return n.alive();
      }).length;
    if (this.alive()) {
      return alives === 2 ||
             alives == 3;
    }
    else {
      return alives == 3;
    }
  };
```

```
Universe.prototype.iterate =
  function iterate () {
    var aliveInNextGeneration = this.cells().map(
      function (c) {
        return [c, c.nextAlive()];
      }
    );

    aliveInNextGeneration.forEach(function (a) {
      var cell = a[0],
          next = a[1];

      cell.setAlive(next);
    });
  };
```

drawing

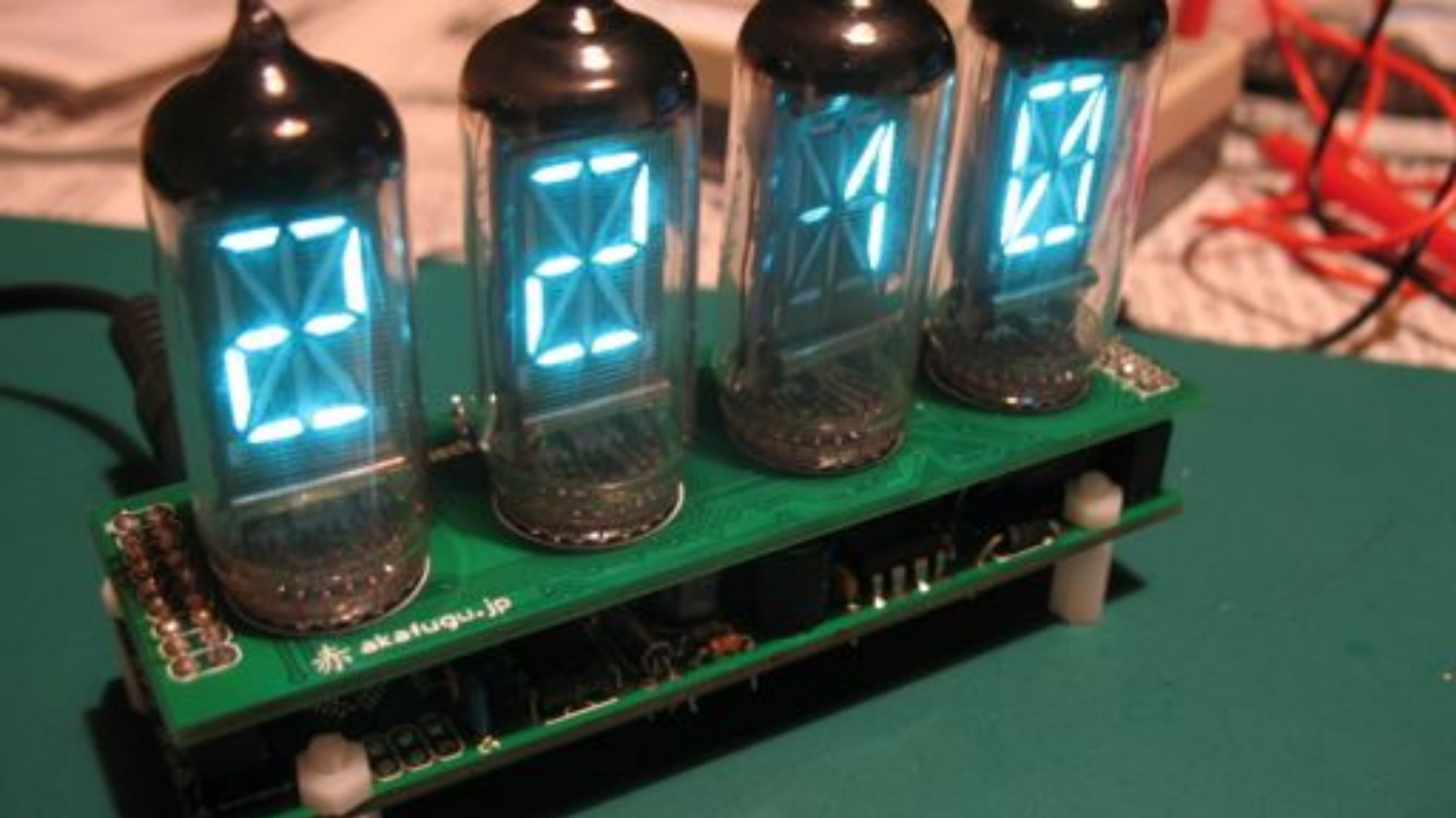
# Life

# View

1. Drawing a Cell:

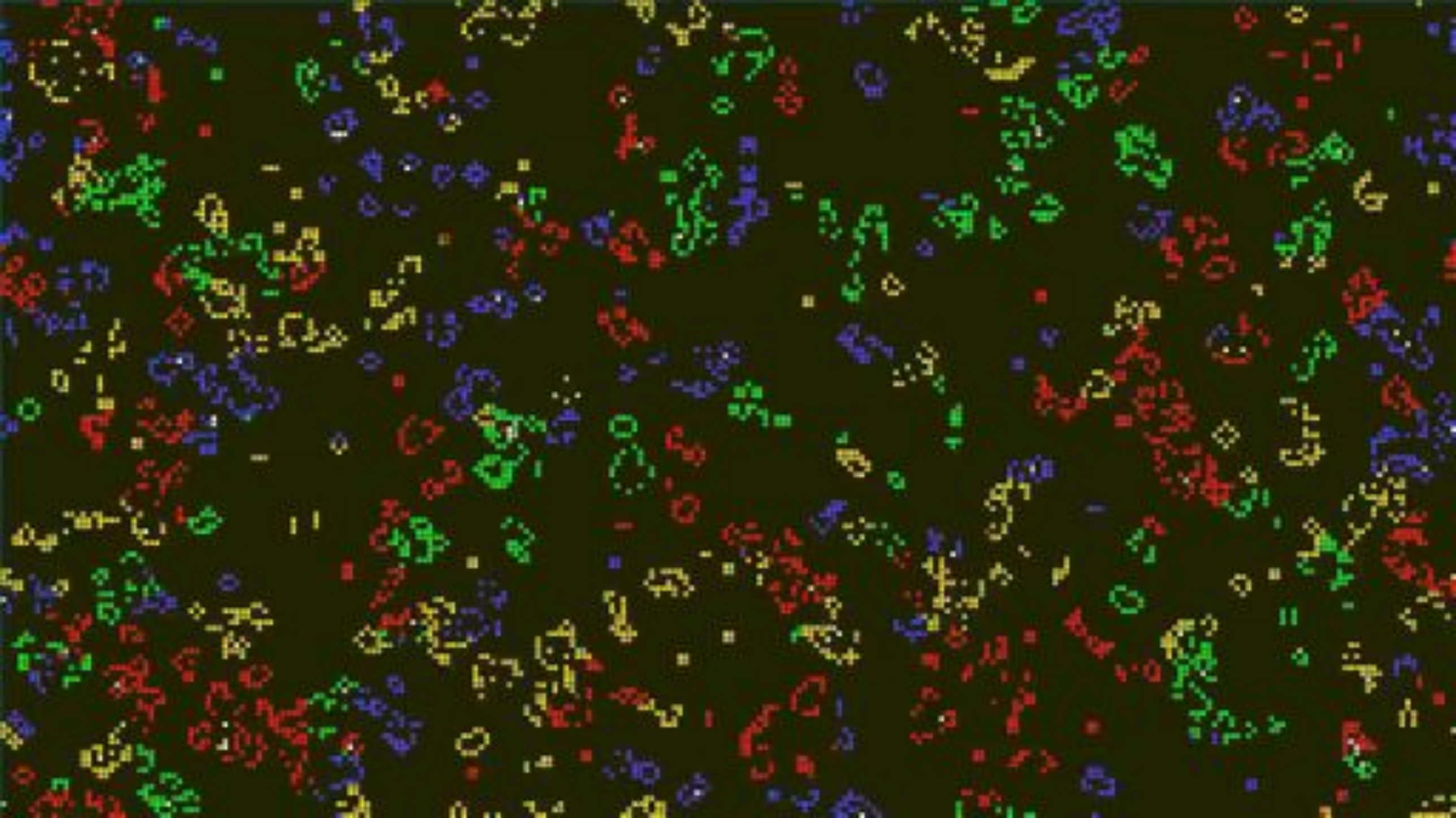
```
View.prototype.drawCell =
  function drawCell (cell, x, y) {
    var xPlus = x + this.cellSize(),
        yPlus = y + this.cellSize()
    this._canvasContext.clearRect(x, y, xPlus, yPlus);
    this._canvasContext.fillStyle = this.cellColour(cell);
    this._canvasContext.fillRect(x, y, xPlus, yPlus);
    return self;
  };
```

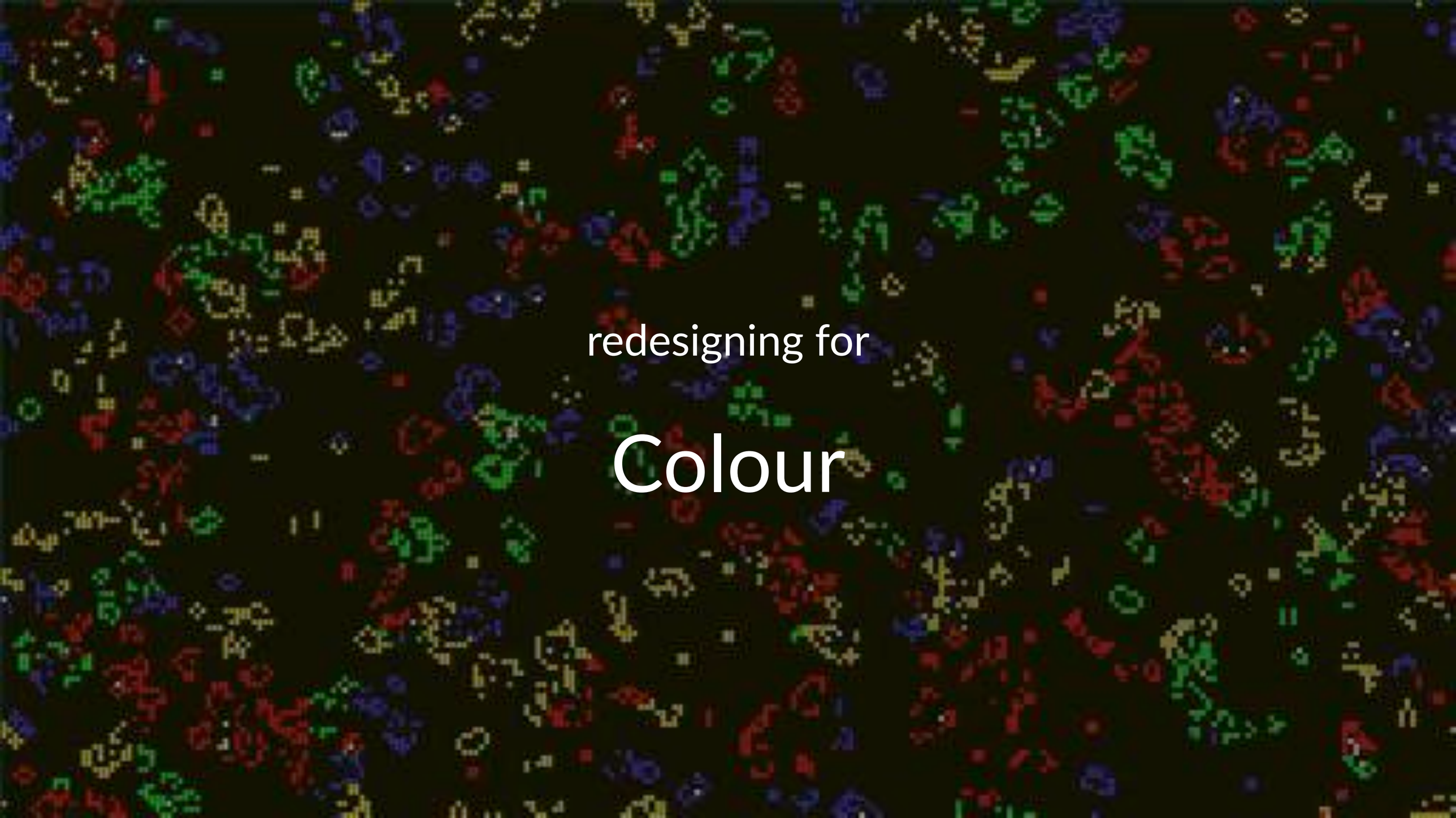
```
View.prototype.cellColour =
  function cellColour (cell) {
    return cell.alive()
           ? WHITE
           : BLACK;
  };
```


akafuqu.jp

CHANGE

Ch-ch-ch-changes!





redesigning for

# Colour

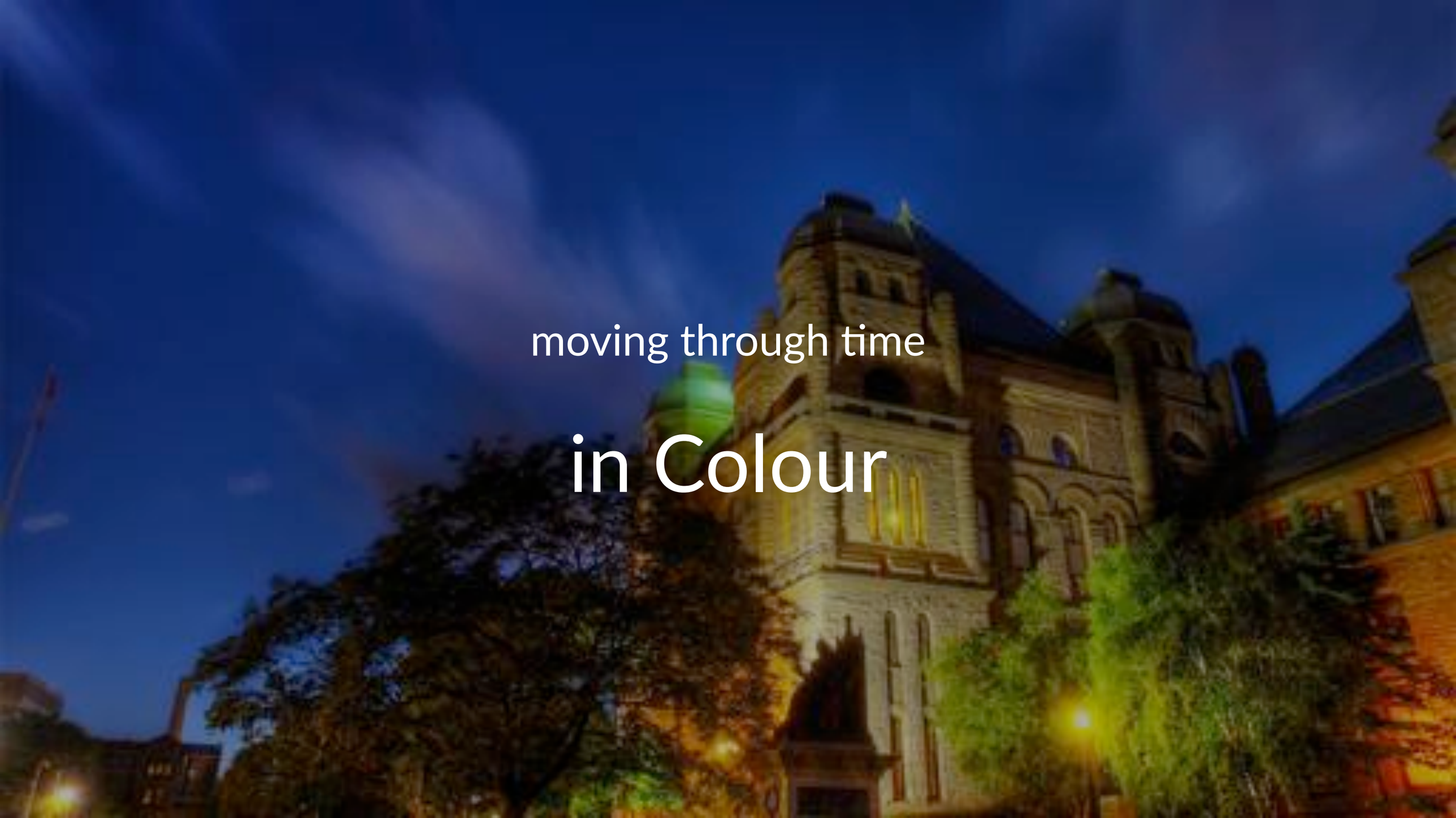
```
function ColourCell () {
  this._neighbours = [];
  this._age        = 0;
}

ColourCell.prototype.neighbours =
  StandardCell.prototype.neighbours;


ColourCell.prototype.setNeighbours =
  StandardCell.prototype.setNeighbours;
```

```
ColourCell.prototype.age =
  function age () {
    return this._age;
  };

ColourCell.prototype.setAge =
  function setAge (age) {
    this._age = age;
    return this;
  };
```



moving through time

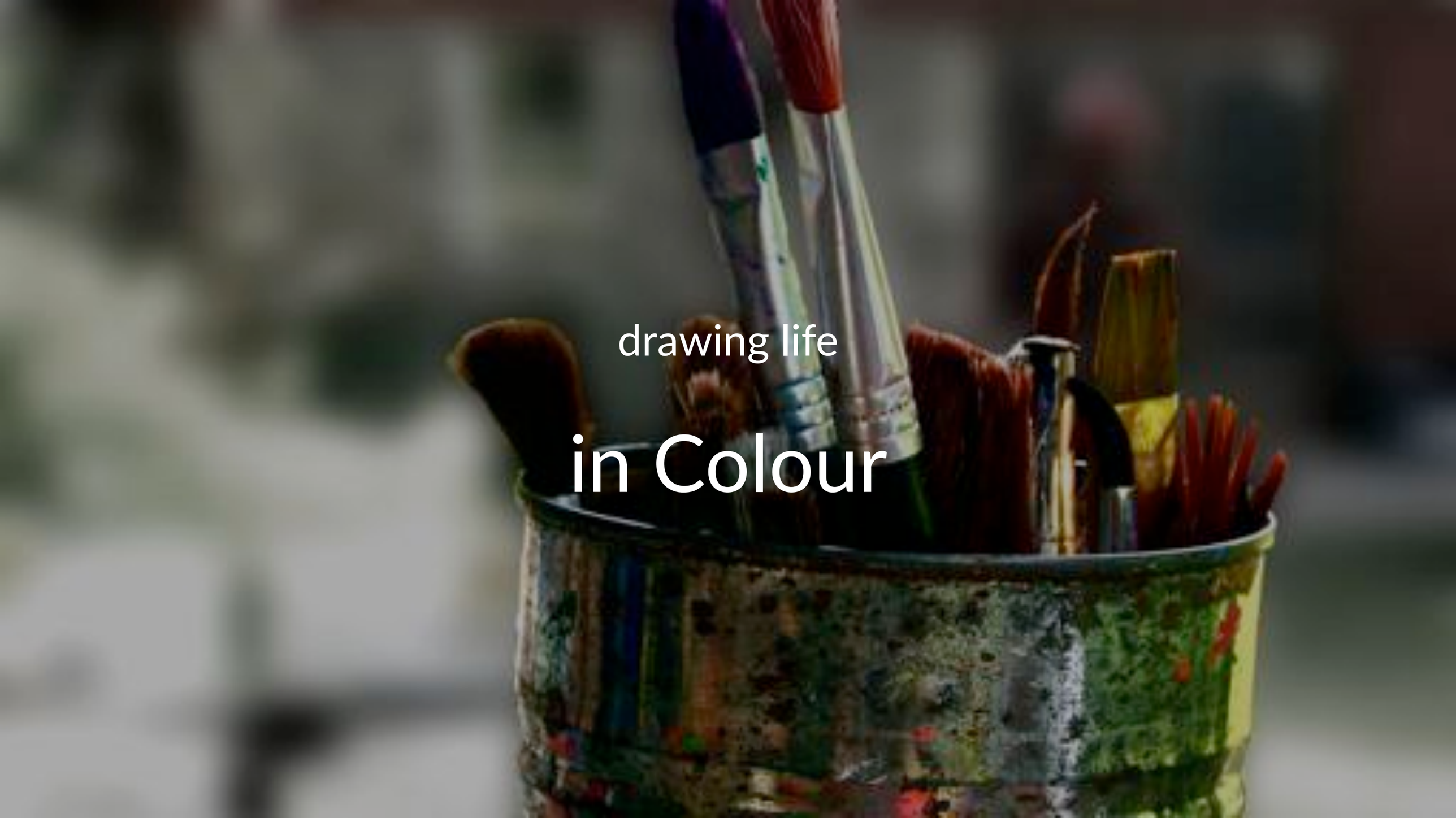
# in Colour

```
ColourCell.prototype.nextAge =
  function next () {
    var alives =
      this._neighbours.filter(function (n) {
        return n.age() > 0;
      }).length;
    if (this.age() > 0) {
      return (alives === 2 || alives == 3)
             ? (this.age() + 1)
             : 0;
    }
    else {
      return (alives == 3)
             ? (this.age() + 1)
             : 0;
    }
  };
```

```
Universe.prototype.iterate =
  function iterate () {
    var ageInNextGeneration = this.cells().map(
      function (c) {
        return [c, c.nextAge()];
      }
    );

    ageInNextGeneration.forEach(function (a) {
      var cell = a[0],
          next = a[1];

      cell.setAge(next);
    });
  };
```



drawing life

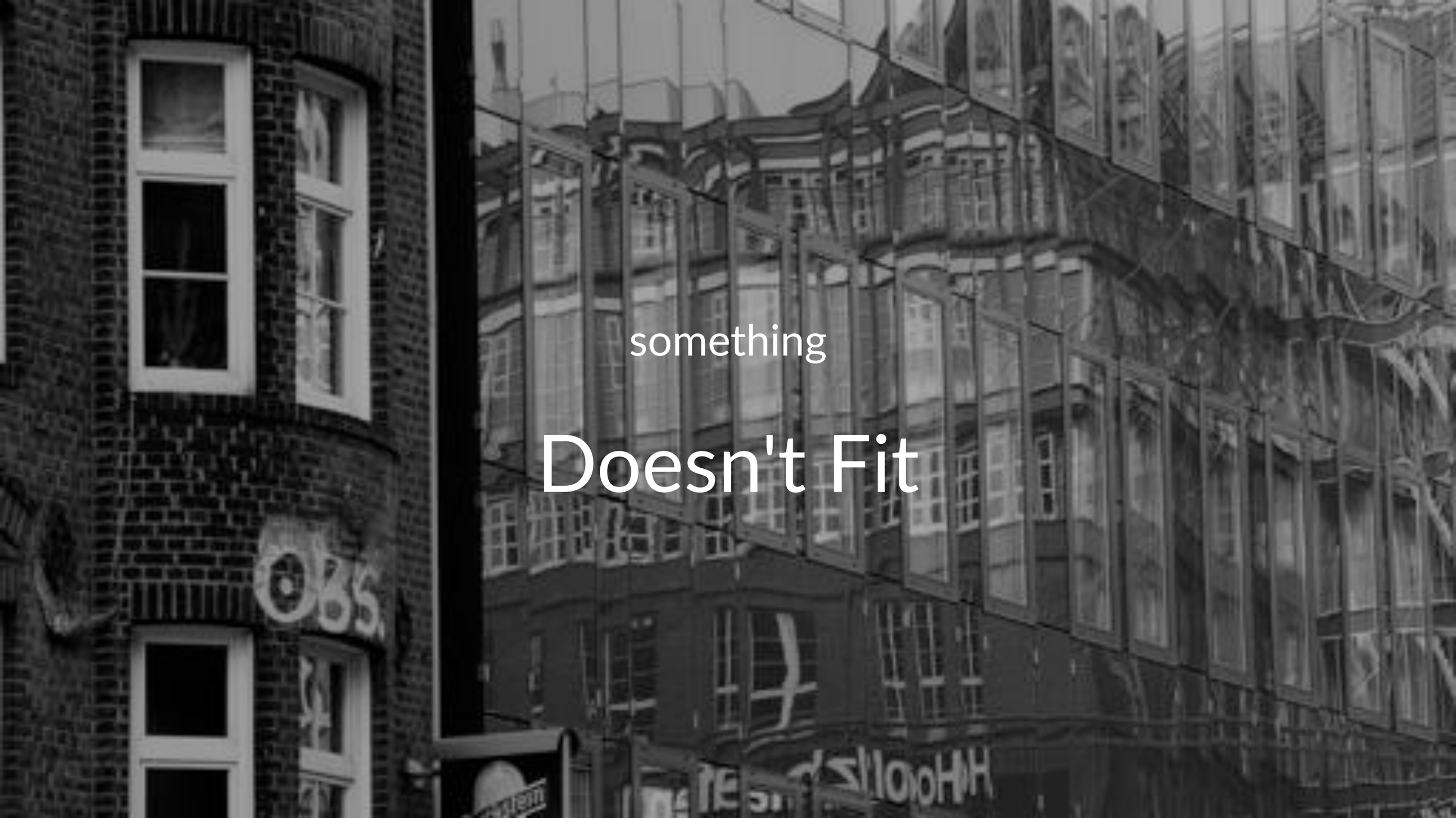
# in Colour

```
var COLOURS =
  [ BLACK, GREEN, BLUE, YELLOW, WHITE, RED ];

View.prototype.cellColour =
  function cellColour (cell) {
    return COLORS[
                  (cell.age() >= COLOURS.length)
                  ? (COLOURS.length - 1)
                  : cell.age()
                ];
  };

// ...
```

OBS



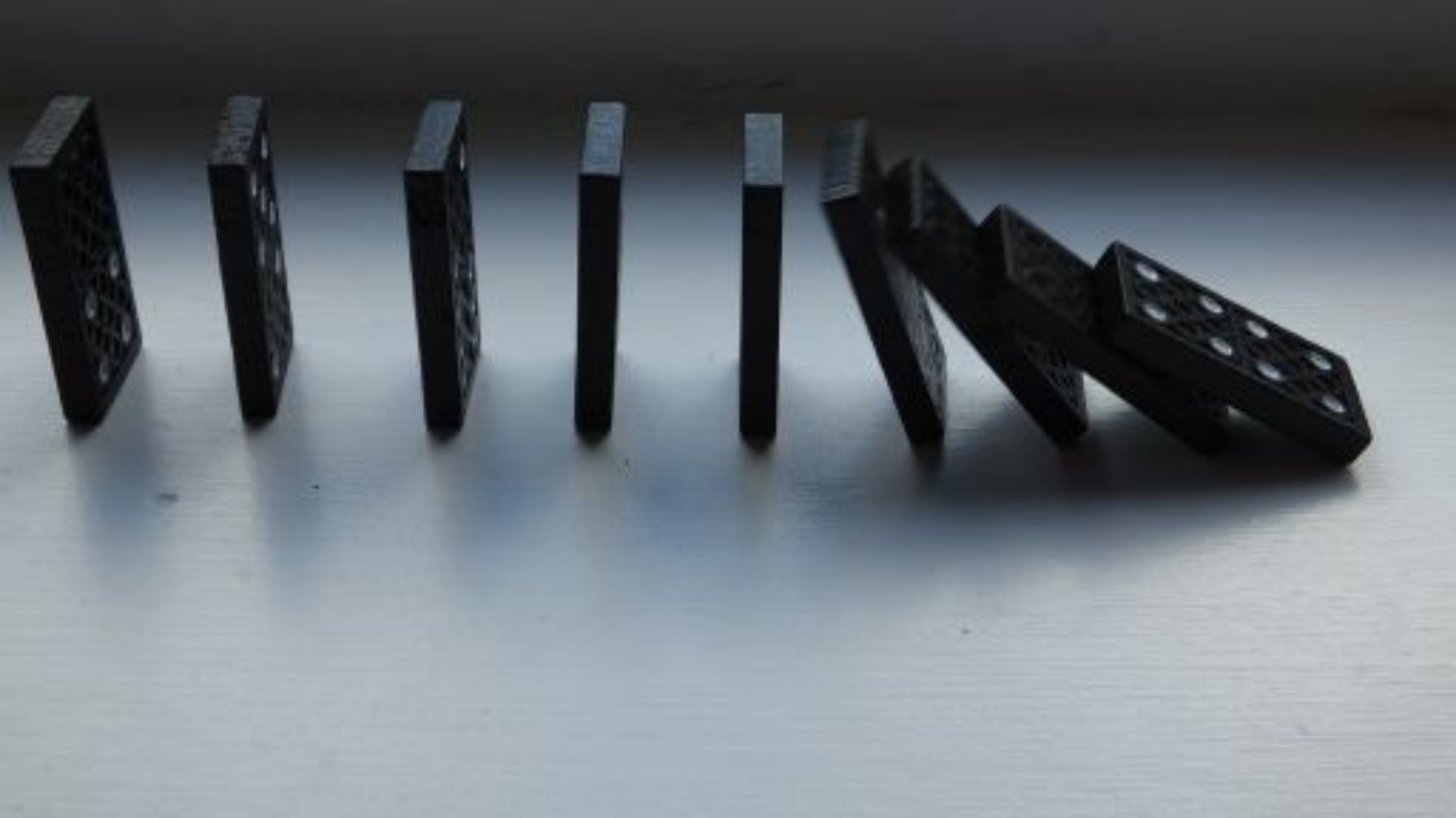
something

# Doesn't Fit

```
Object.keys(StandardCell.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'alive',
    'setAlive',
    'nextAlive' ]
```

```
Object.keys(ColourCell.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'age',
    'setAge',
    'nextAge' ]
```

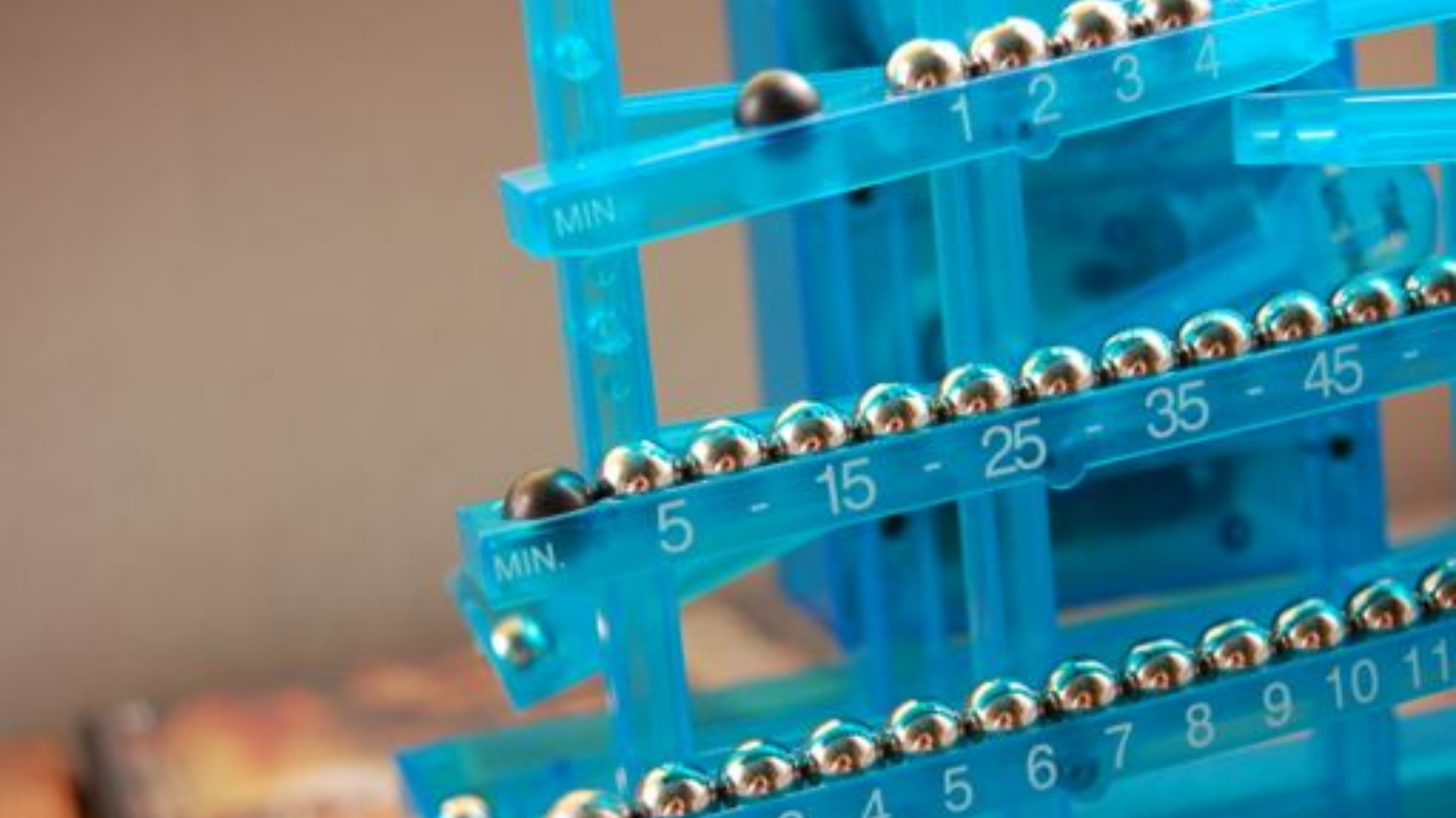
The state of a cell is now "age" instead of "aliveness."

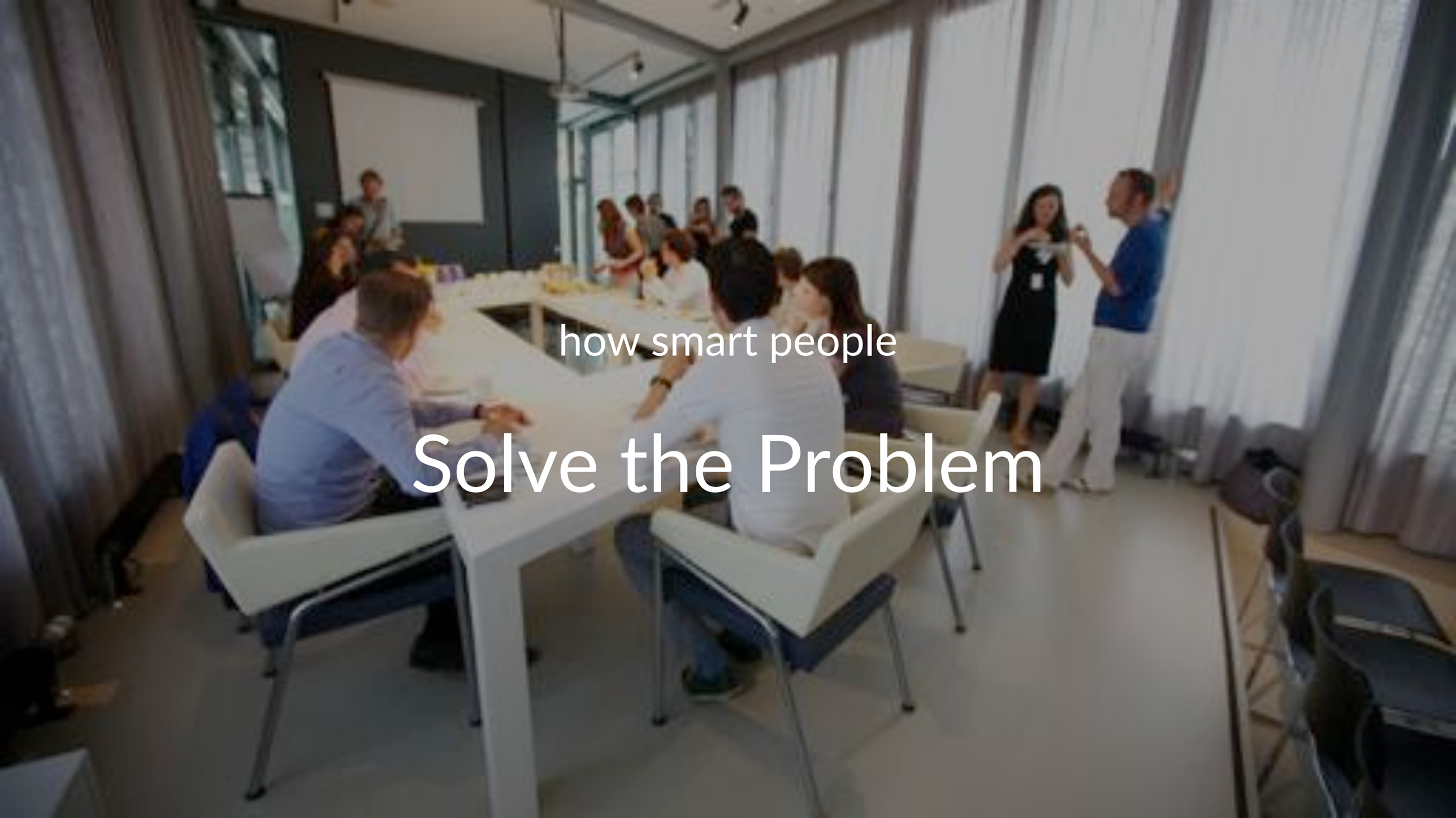


These changes will
Ripple Through Universe and View

our problem is that

# Universe is coupled to Cell's Interface


MIN.
1 2 3 4
MIN.
5 - 15 - 25 - 35 - 45
6 7 8 9 10 11
4 5



how smart people

# Solve the Problem

"Make a coloured cell behave like a standard cell."

```
ColourCell.prototype.alive =
  function alive () {
    return this._age > 0;
  };
```

```
ColourCell.prototype.setAlive =
  function setAlive (alive) {
    if (alive) {
      this.setAge(this.age() + 1);
    }
    else this.setAge(0);
    return this;
  };
```

```
ColourCell.prototype.nextAlive =
  StandardCell.prototype.nextAlive;
```

```
Object.keys(ColourCell.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'age',
    'setAge',
    'nextAge',
    'alive',
    'setAlive',
    'nextAlive' ]
```

# Presto! Backwards Compatibility.

(it "tastes like a standard cell")

universe and cell are
No Longer Coupled

4
6
3
30
3
TOTAL
2
20
2

# At What Cost?

Increasing Flexibility
Increases Complexity

there is
Another Way

"Make a cell that behaves like a Standard Cell, but is implemented in terms of a Coloured Cell."

```
function AsStandard (colouredCell) {
  this._colouredCell = colouredCell;
}

var tastesLikeAStandardCell =
  new AsStandard(aColourCell);
```

```
AsStandard.prototype.neighbours =
  function neighbours () {
    return this._colouredCell.neighbours();
  };

AsStandard.prototype.setNeighbours =
  function setNeighbours (neighbours) {
    this._colouredCell.setNeighbours(neighbours);
    return this;
  };
```

```
AsStandard.prototype.alive =
  function alive () {
    return this._colouredCell.age() > 0;
  };
```

```
AsStandard.prototype.setAlive =
  function setAlive (alive) {
    if (alive) {
      this._colouredCell.setAge(this._colouredCell.age() + 1);
    }
    else this._colouredCell.setAge(0);
    return this;
  };
```

```
AsStandard.prototype.nextAlive =
  function nextAlive () {
    return this._colouredCell.nextAge() > 0;
  }
```

```
Object.keys(AsStandard.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'alive',
    'setAlive',
    'nextAlive' ]
```

# When we Upgrade to ColourCell

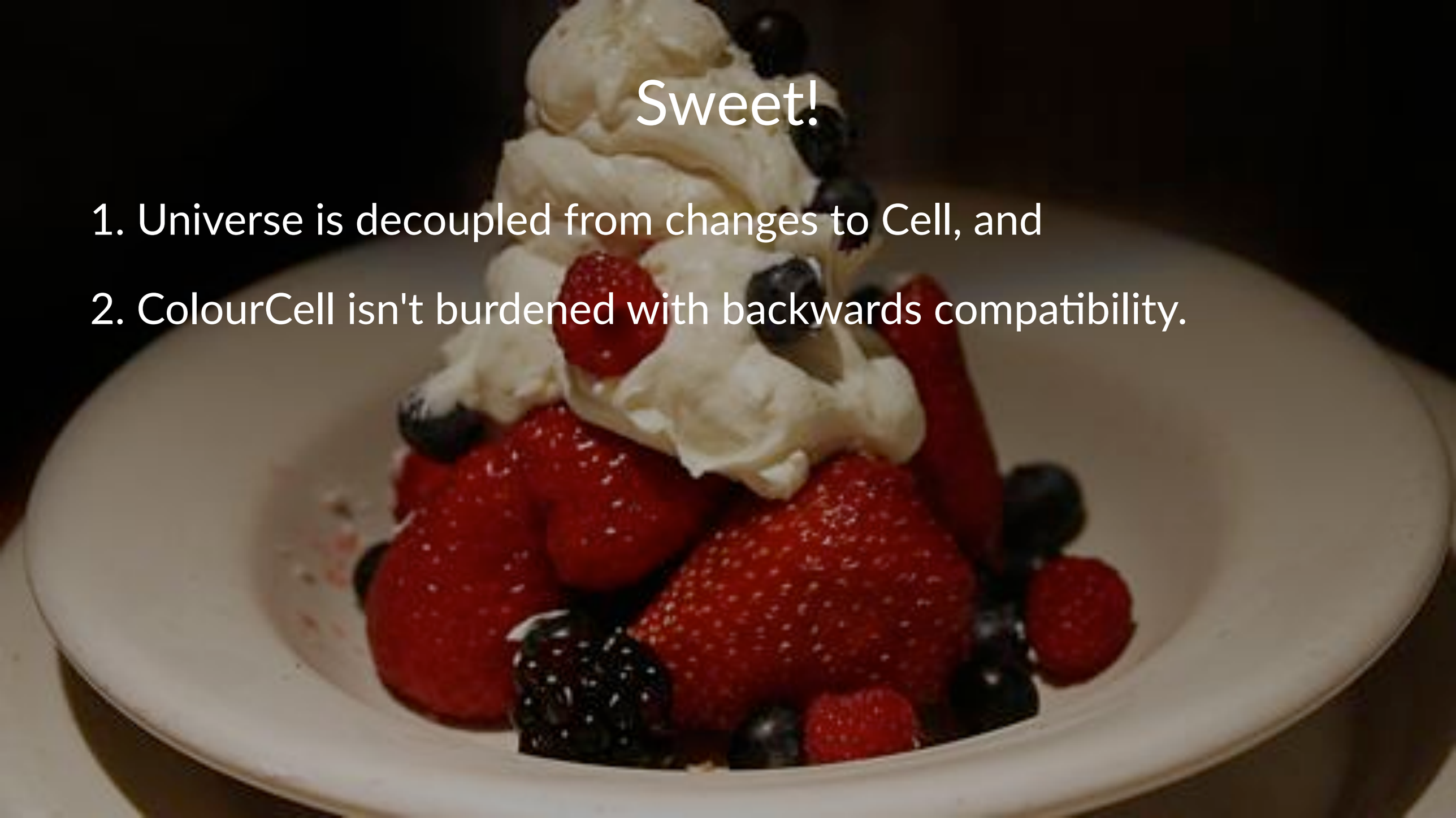
1. View can depend upon ColourCell,

# When we Upgrade to ColourCell

1. View can depend upon ColourCell, and
2. Universe can depend upon AsStandard.

# Sweet!

1. Universe is decoupled from changes to Cell,



# Sweet!

1. Universe is decoupled from changes to Cell, and
2. ColourCell isn't burdened with backwards compatibility.

# AsStandard is an *Adaptor*

The adapter (sic) pattern is a software design pattern that allows the interface of an existing class to be used from another interface...

It is often used to make existing classes work with others without modifying their source code.

TIMTOWTDI

possibilities

good programmers borrow from smalltalk,
Great Programmers Steal from C++

```
function colourFromStandard (standard) {
  return new ColourCell()
          .setNeighbours(standard.neighbours())
          .setAge(standard.alive() ? 1 : 0);
}
```

```
function standardFromColour (colour) {
  return new StandardCell()
         .setNeighbours(colour.neighbours())
         .setAlive(colour.age() > 0);
}
```

# Coproxies

# Approaching the End

30

adaptors

# Separate Concerns

and thus, adaptors

# Isolate Change

57309

and they

# Decouple Modules

adaptors

# Solve Compatibility Problems

but they also gives us the freedom to
Make Changes, Safely

when we're using teams to build software at scale

# Adaptors are an Important Tool

THE END

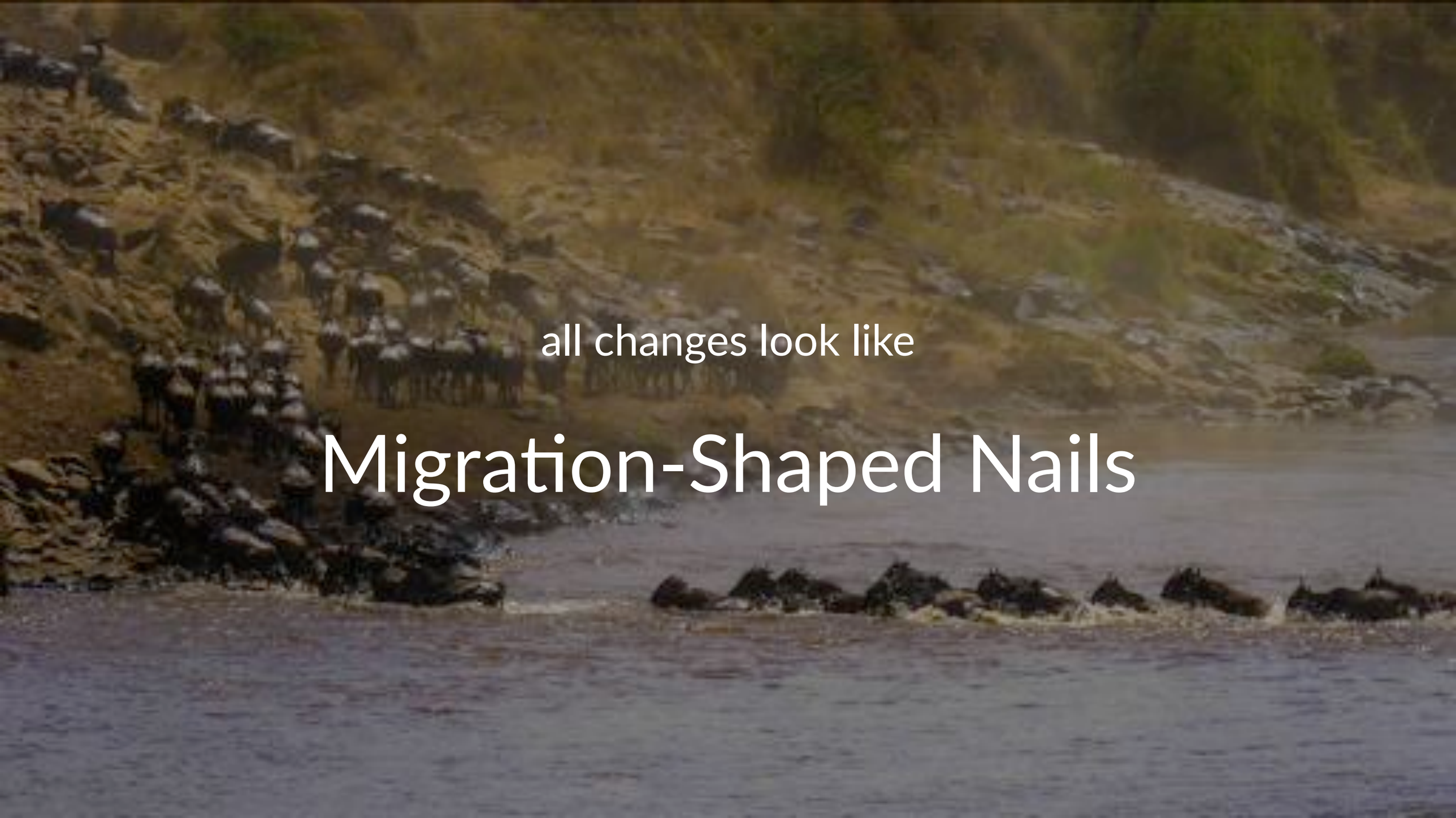
one more thing:

PACKAGES

LIMIT
25
$44.
SINGLE
50¢
1 75
5TH
5LEG

when you're holding a

# Version-Shaped Hammer



all changes look like

# Migration-Shaped Nails

# Packages Have Interlocking Dependencies

# Migrations Can Cascade

you never know how much work a single upgrade can cause

migration is expensive, so semver values

# Backwards Compatibility

"duck typed" backwards compatibility

# Holds Progress Back

breaking compatibilty

# Exposes Clients to Bugs

No smoking
London
Will Pierce rescue the Trojan Women?

adaptors decouple

# Compatibility from Progress

凡進入地盤或在進行工程中
必須配戴適當防護用具，違者可被檢控

**SAFETY EQUIPMENTS MUST BE WORN**
**OFFENDERS WILL BE LIABLE TO PROSECUTION**

adaptors provide

# Safe Compatibility

BY AIR MAIL
PAR AVION

# What would an adaptor-aware package manager look like?

# An Adaptor-Aware Package Manager

1. Interfaces and Implementations would be independantly versioned,

# An Adaptor-Aware Package Manager

1. Interfaces and Implementations would be independantly versioned,
2. Interfaces would have a many-to-many relationship with Implementations, mediated by Adaptors,

# An Adaptor-Aware Package Manager

1. Interfaces and Implementations would be independantly versioned,
2. Interfaces would have a many-to-many relationship with implementations, mediated by adaptors,
3. Clients would specifiy the required interface and implementation,

# An Adaptor-Aware Package Manager

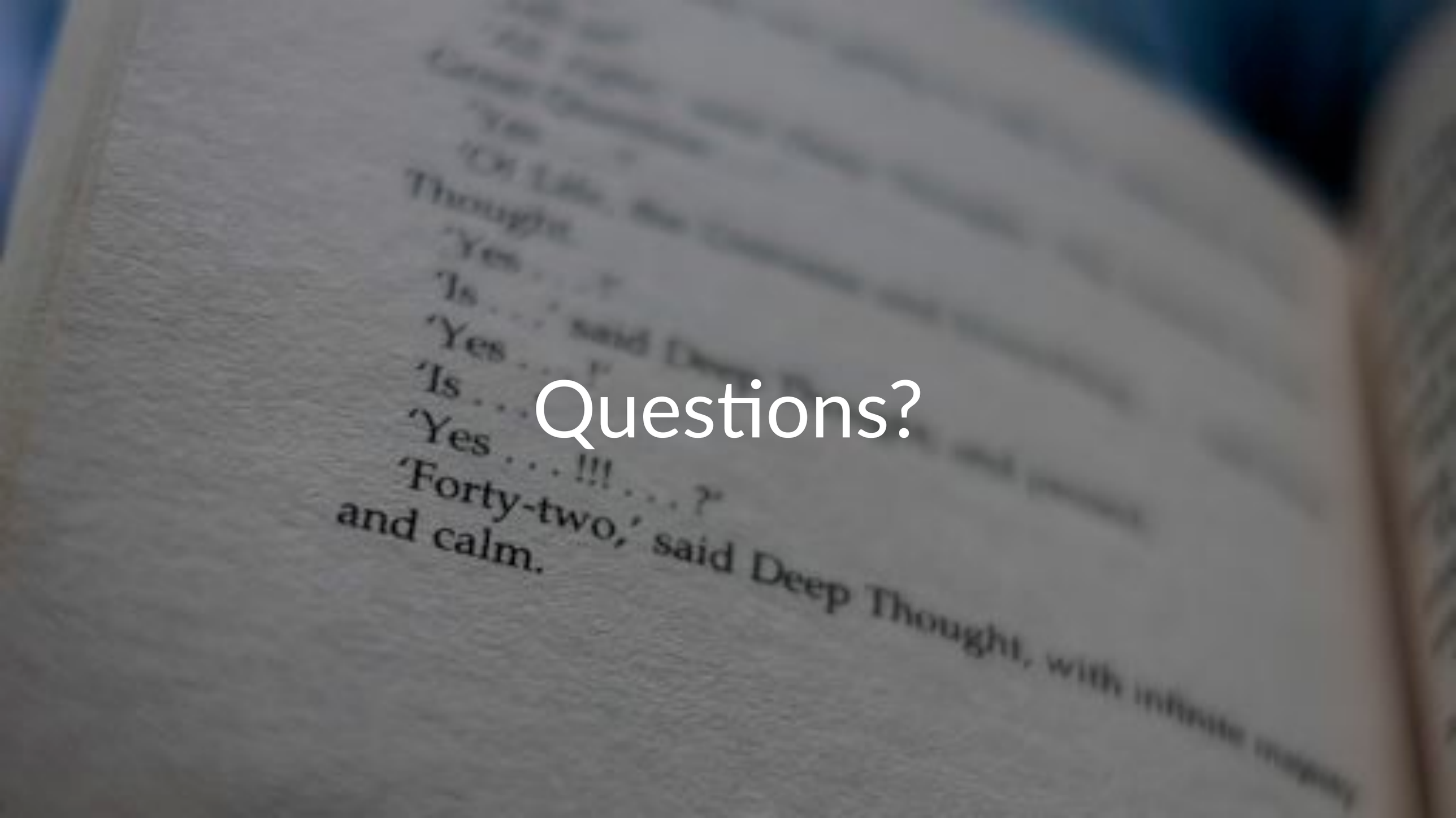
1. Interfaces and Implementations would be independantly versioned,
2. Interfaces would have a many-to-many relationship with implementations, mediated by adaptors,
3. Clients would specifiy the required interface and implementation, and
4. Old clients can benefit from bug fixes without rewrites.

an adaptor-aware package manager would

# Rewire the Way we Grow Software

and that provides
Food for Thought
for a very long time indeed

Tack Så Mycket!


'Yes . . . ?'
'Is . . . ?'
'Yes . . . !!! . . . ?'
'Forty-two,' said Deep Thought, with infinite
and calm.

# Questions?

Reginald Braithwaite

GitHub, Inc.

raganwald.com

@raganwald

Nordic JS
Artipelag, Stockholm, September 19, 2014

# JavaScript Spessore

A Thick Shot of Objects, Metaobjects, & Protocols
by Reginald "raganwald" Braithwaite