The Game Of Life

Java's Siblings and Heirs are populating the Ecosystem

Inspiration for this talk

An APL Version Game Of Life

```
      disp +/ +⌿ 1 0 ¯1 ∘.⊖ 1 0 ¯1 ⌽" ⊂R
┌─────────────┐
│0 0 1 2 2 1 0│
│0 1 3 4 3 1 0│
│0 1 4 5 4 1 0│
│0 1 3 3 2 0 0│
│0 0 1 1 1 0 0│
└─────────────┘

      disp 3 4 = +/ +⌿ 1 0 ¯1 ∘.⊖ 1 0 ¯1 ⌽" ⊂R
┌─────────────┬─────────────┐
│0 0 0 0 0 0 0│0 0 0 0 0 0 0│
│0 0 1 0 1 0 0│0 0 0 1 0 0 0│
│0 0 0 0 0 0 0│0 0 1 0 1 0 0│
│0 0 1 1 0 0 0│0 0 0 0 0 0 0│
│0 0 0 0 0 0 0│0 0 0 0 0 0 0│
└─────────────┴─────────────┘

      disp 1 R 3 4 = +/ +⌿ 1 0 ¯1 ∘.⊖ 1 0 ¯1 ⌽"
```

A Natural View

- ecosystem
- monoculture
- diversity
- evolution
- forces

Ecosystem?

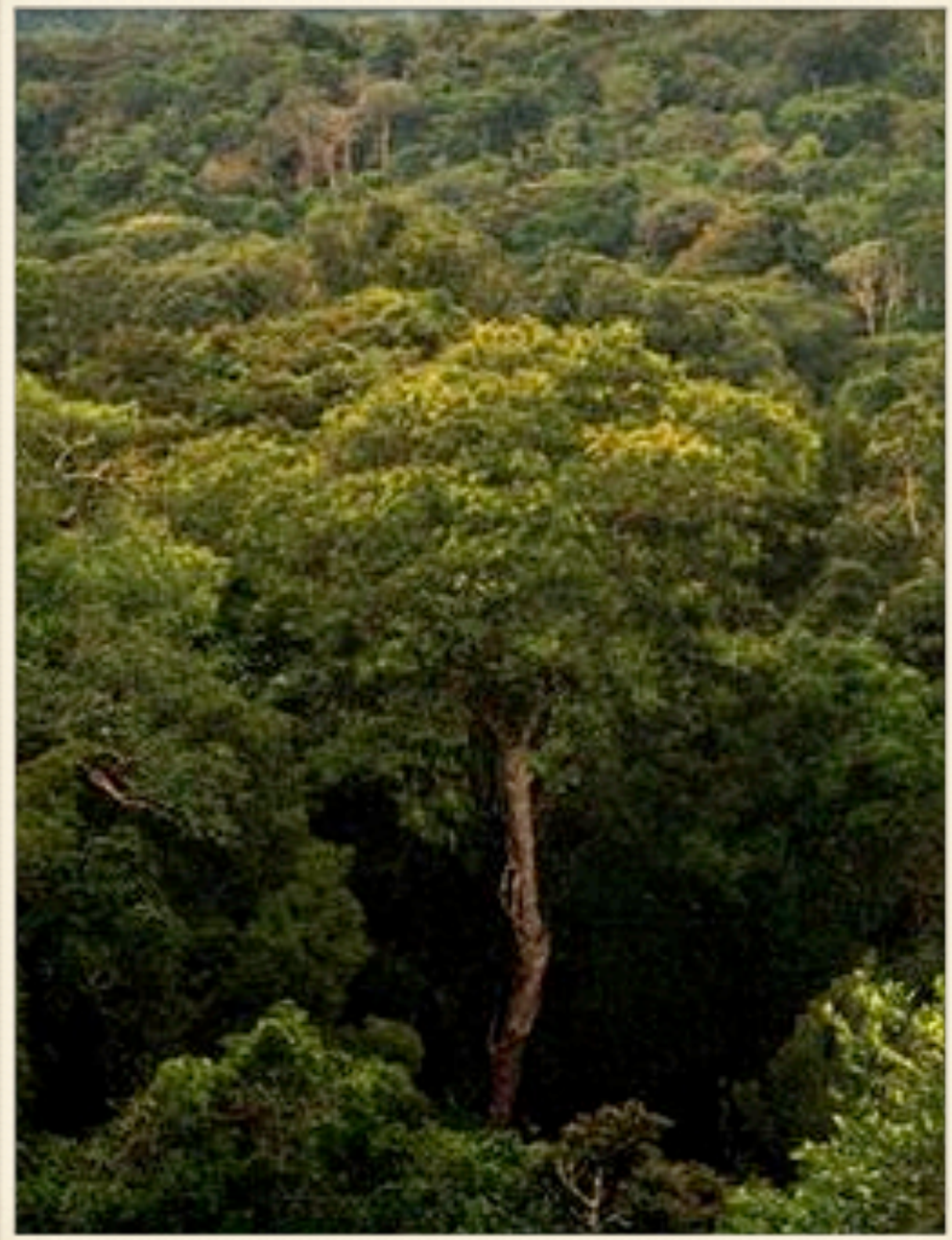
Our working environment can be seen as an ecosystem

It should be habitable for humans and efficient for a machine

It is evolving and changing

Ecosystem

- inhabitants populate
  - an environment
  - are exposed to outside
    - forces and influences
- many supporting relationships
- robust or sensitive variations
- natural and artificial



Our Ecosystem consists of

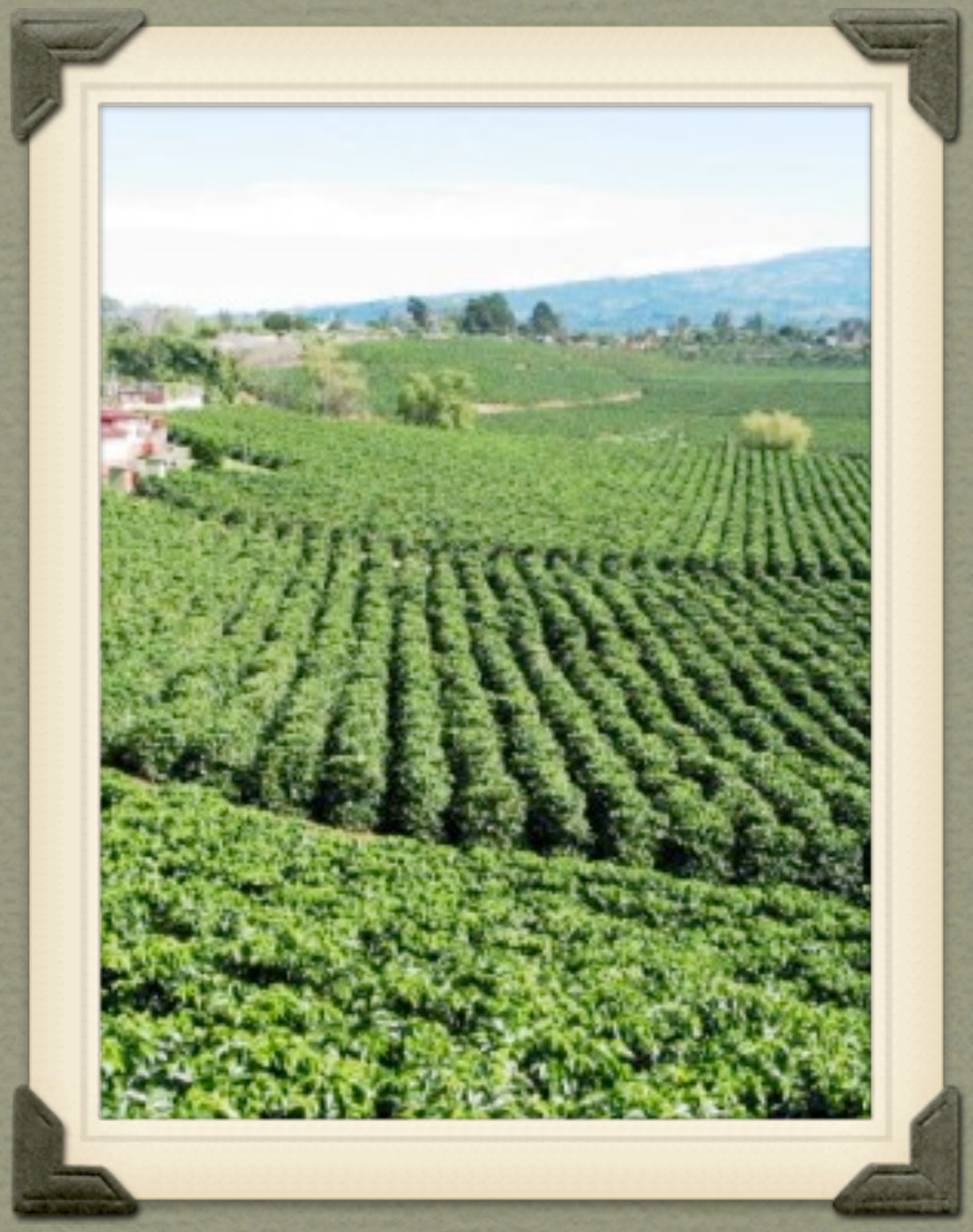
- virtual machine
- tools
- languages
- libraries

The JVM

- abstracts hardware, operating systems
- portable
- fast and getting faster
- runs bytecode
- not just one language
- but many of them
  ... and it does

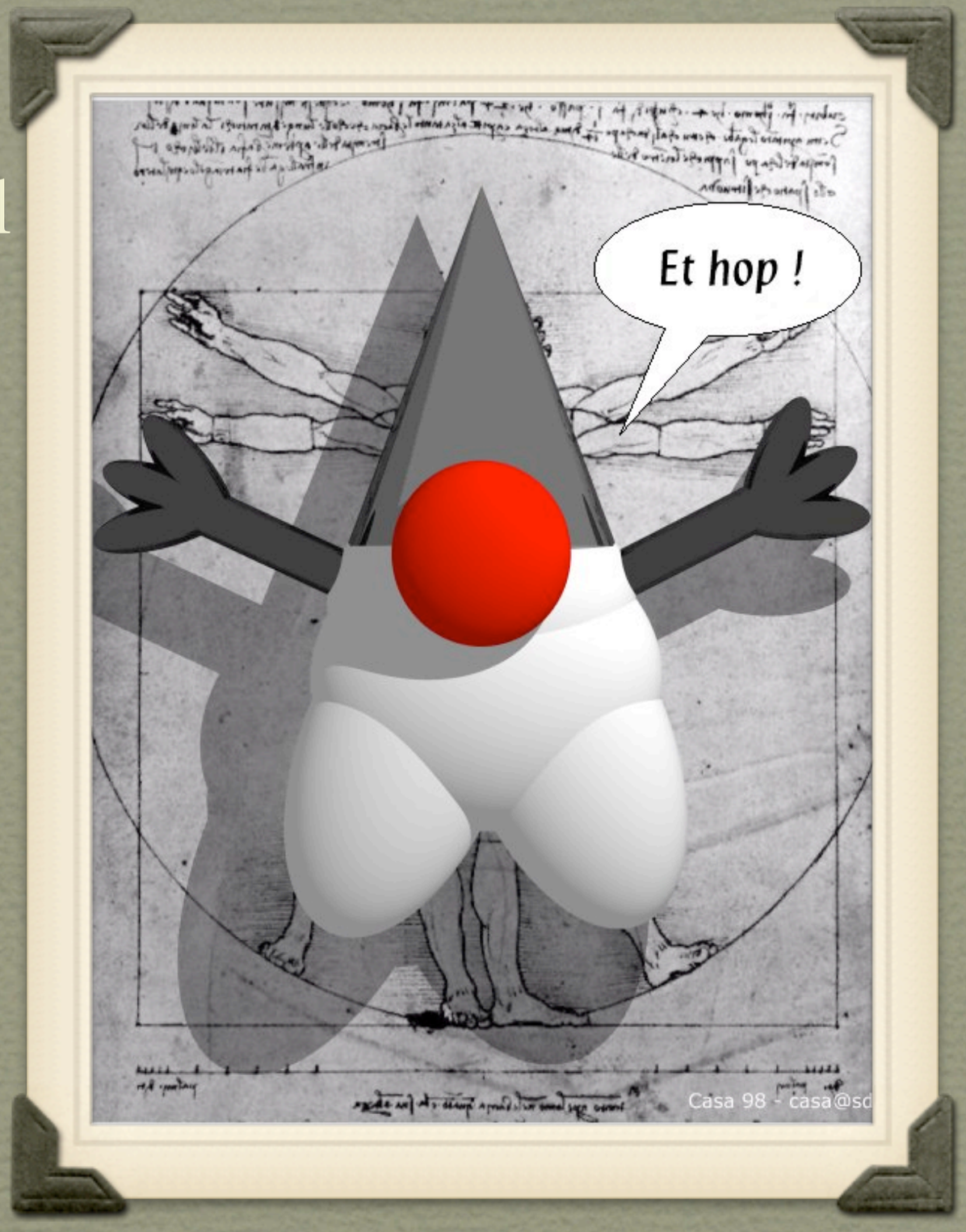
Monoculture

- artificial approach
- a single species
- highly optimized conditions
- economy of scale
- sensitive to changes
- one disruption can destroy whole population
- no evolution



We had a Monoculture

- one language to rule them all
- little knowledge and interest in other languages
- in the mainstream
- economy of scale
- standardization
- single point of failure

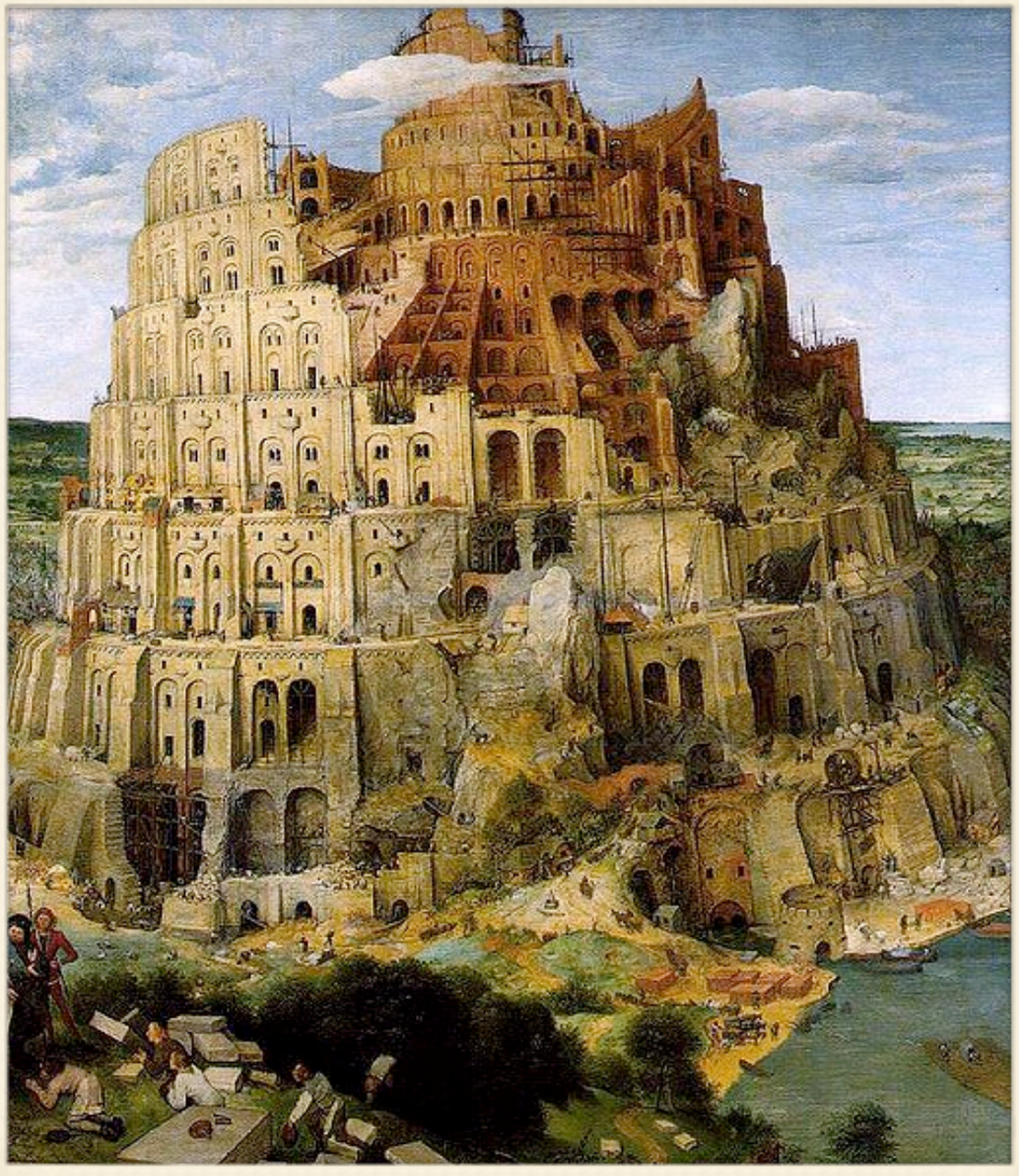


Natural Diversity

- healthy mixture of species
- resistant to disruptions
- supporting ecosystem
- different forces are countered
- many possibilities of evolution
- distribute responsibility
- less artificial conditions needed

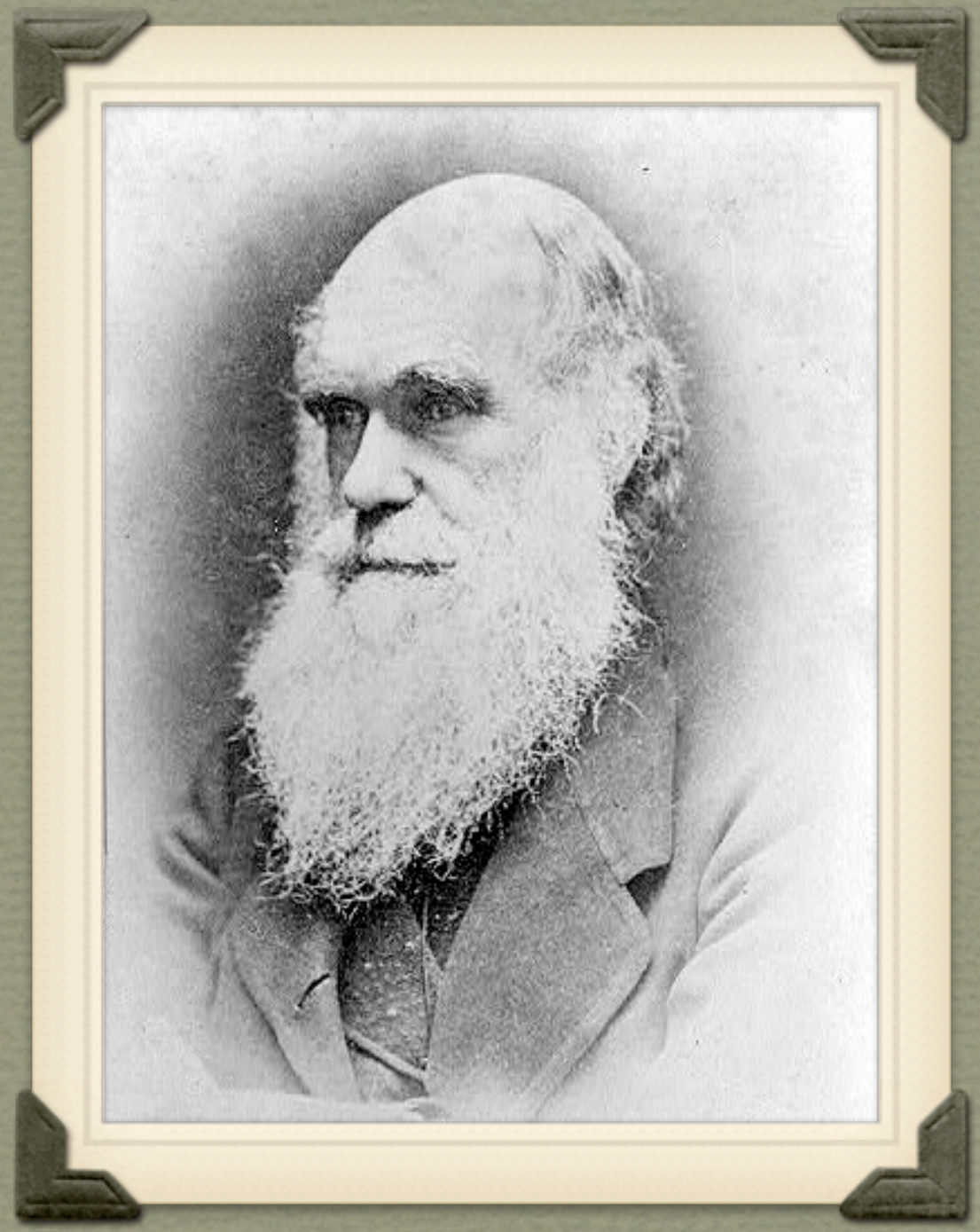
A Breeding Ground For Diversity

- but it was unnoticed
- many languages (400+) running on the JVM
- dynamic and static ones
- address different problems
- many of more expressive than JAVA
- what did it take to make us aware of them?

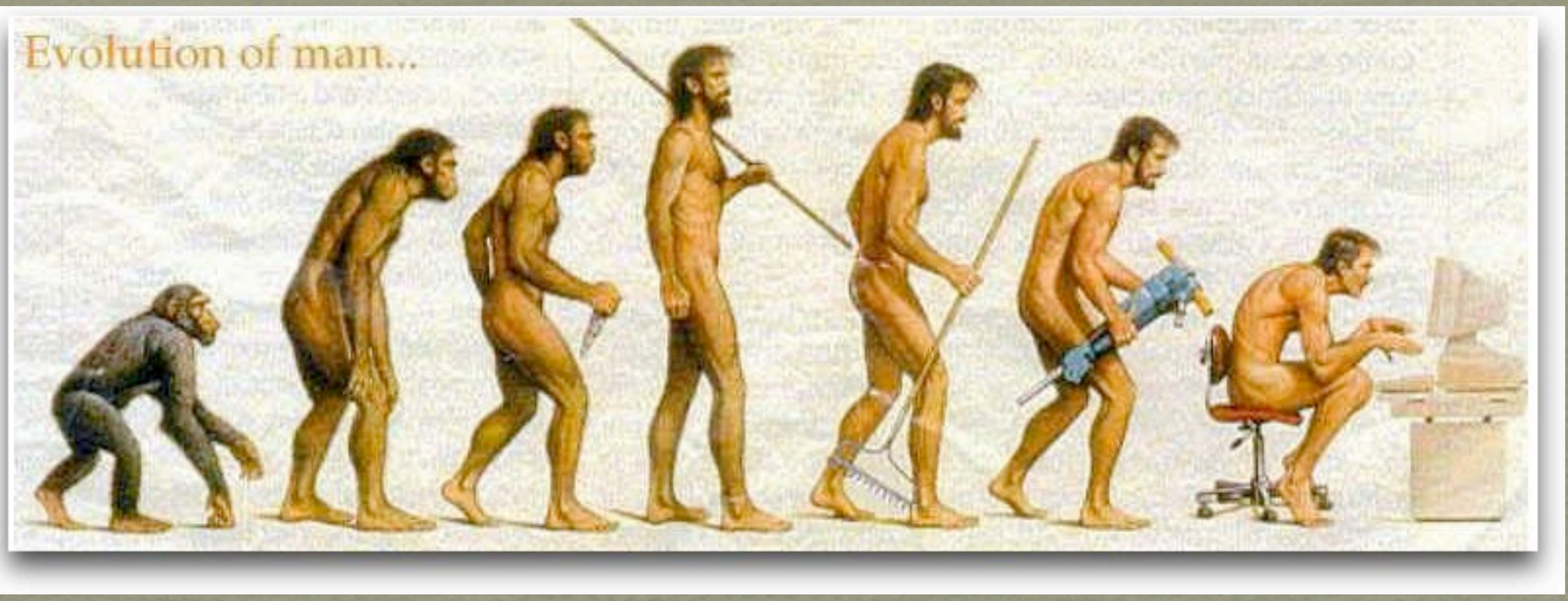


Origin of Languages & Evolution thereof

- Darwins 200th birthday
- external forces shape development of a species
- many different variations are exposed
- only the fittest survive
- form the base for next generations

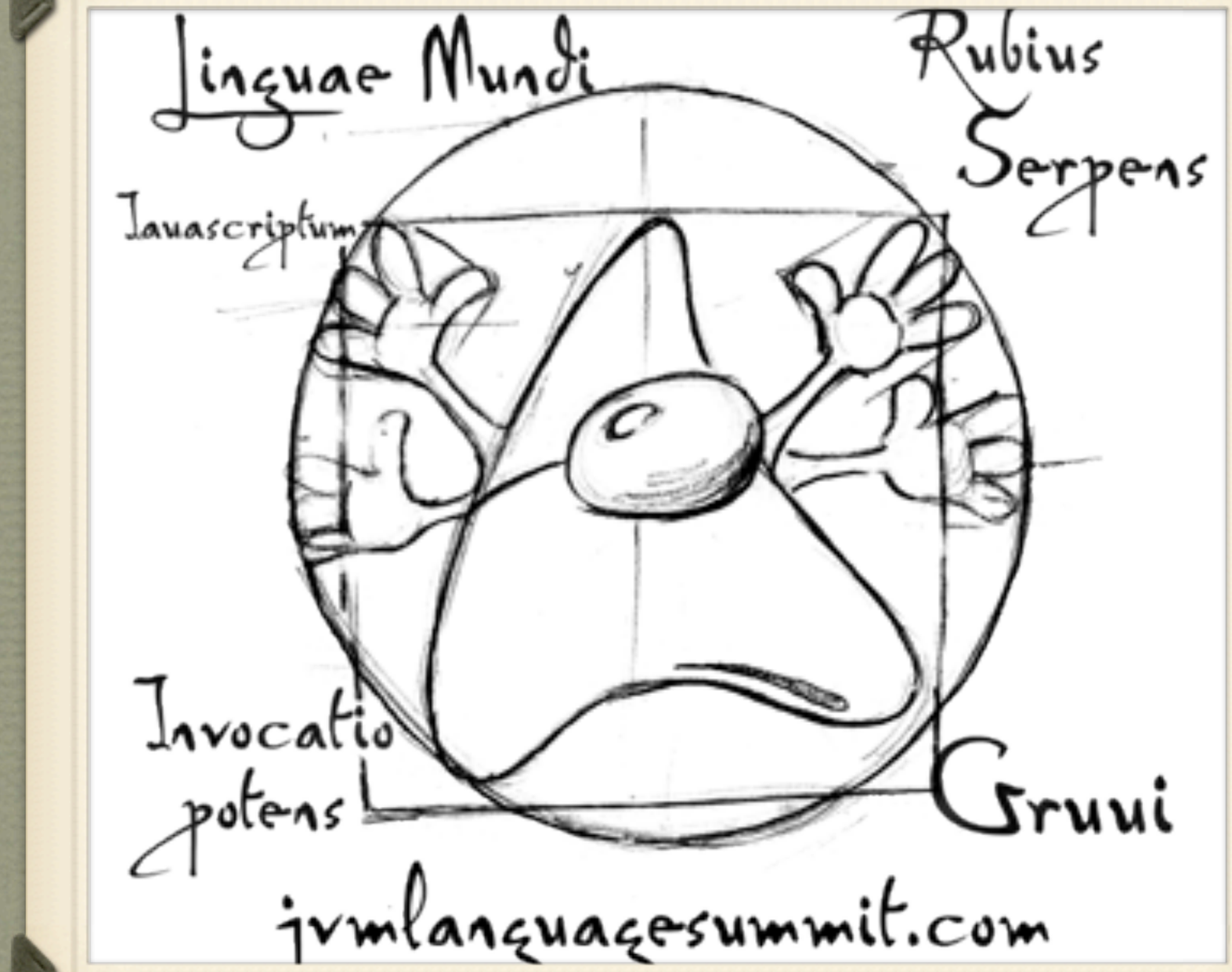


What Happens when the Rules Change?



- concurrency challenge
- cost of human creativity
- complexity of systems
- ease of development / maintenance
- competing approaches

Evolutionary Forces



Now There are more Mainstream JVM Languages

Common JVM Languages

- Scala
- JRuby
- Groovy
- Clojure
- Rhino
- Jython
- JavaFX
- Fortress
- Ioke
- Fan
- Jaskell
- Erjang

Adressing Evolutionary Forces - Concurrency

- functional paradigms
- immutable datastructures
- software transactional memory
- implicit parallelism
- actor model

Adressing Evolutionary Forces - Expressiveness

- dynamic typing
- list comprehensions, high order functions
- metaprogramming, open types
- syntactic sugar, implicit code generation
- dynamic execution (scripting)
- closures
- internal DSLs

Adressing Evolutionary Forces - Competition

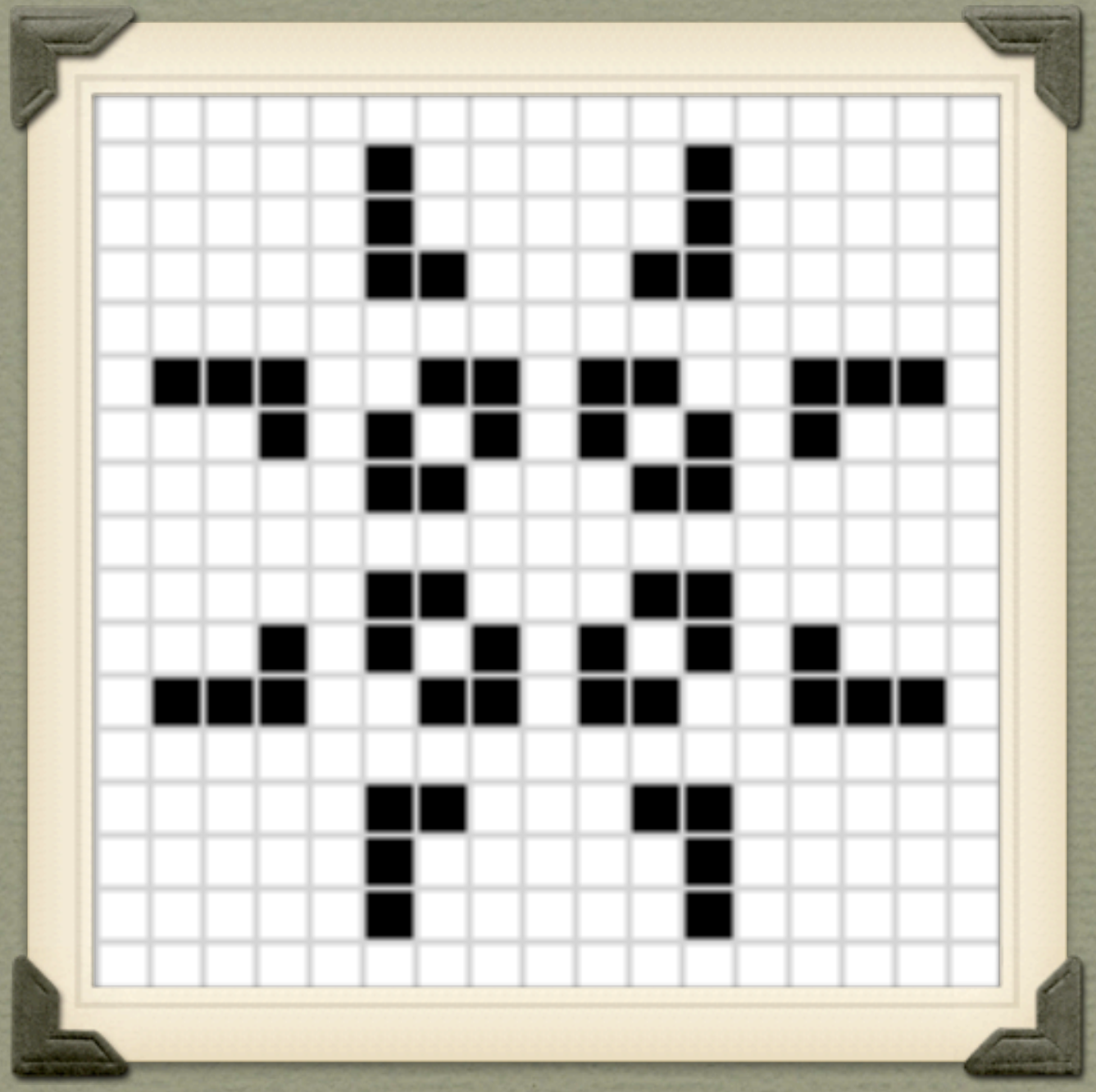
- other languages with an easier syntax and lifecycle
- powerful frameworks (RoR)
- .net Runtime evolves quickly (LinQ & Co)
- simpler paradigms - functional languages
- rich clients - Flash
- highly scalable languages - Erlang

Ola Bini

- language layers in applications
  - stable language at core
  - dynamic layer
  - domain layer
- „end of big languages“
- JVM well suited as polyglot platform
- created Ioke ‚most expressive‘ language experiment

Neal Ford

- polyglot programming
- already a reality
- count the languages in your application
- explicit focus on „the right language for the job“



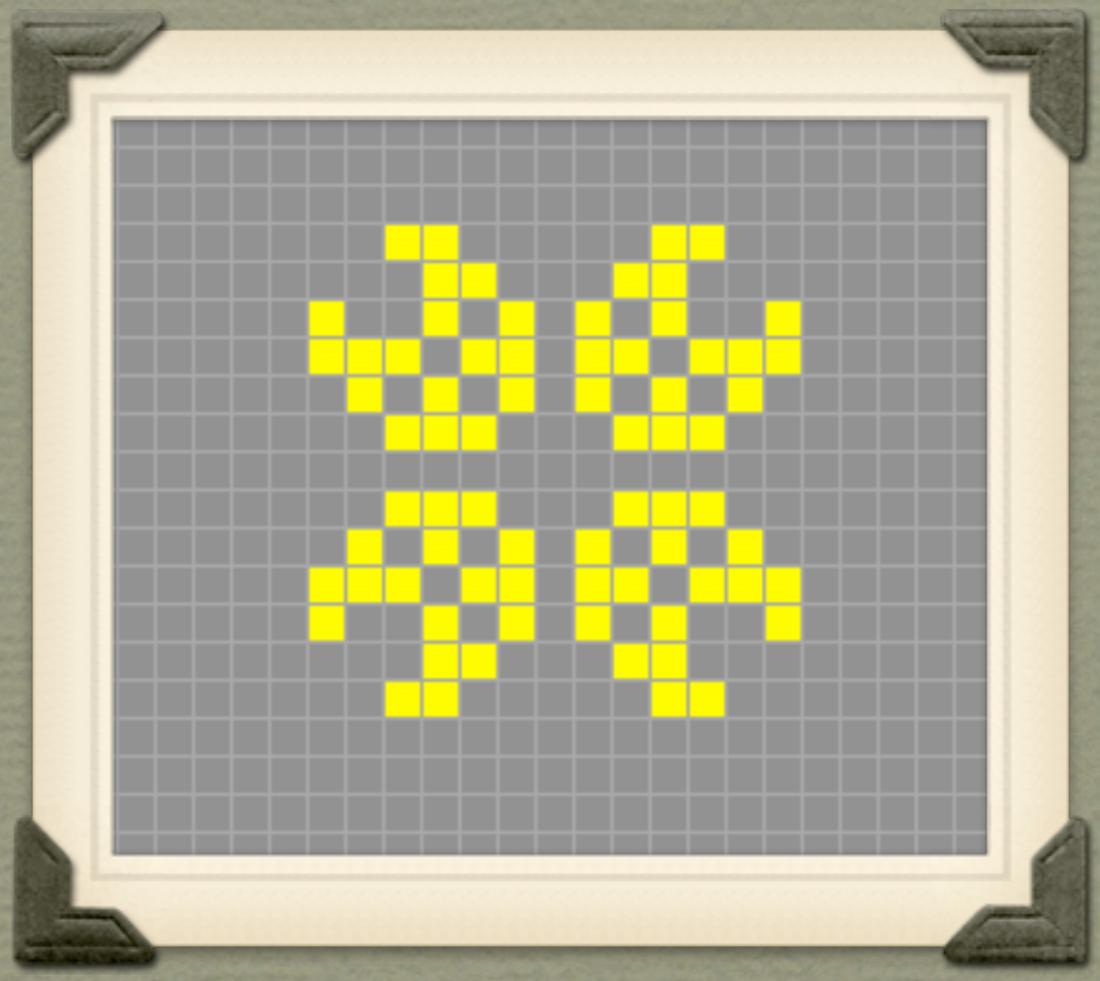
The Game of Life

John H. Conway

- well known mathematician
- invented it 1970 - 40 years ago
- emergence - complexity arises from simple rules
- slightly modeled on real life
- birth, death, survival
- let him tell the story himself

Game of Life

- a simple game with 3 rules
- cell with 2,3 neighbours lives
- space surrounded by 3 cells gets alive
- emulates atoms, laws, forces and time like our universe
- simulates an ecosystem
- explore language expressiveness



Explore JVM Languages

with

Game of Life Implementations

instead of

99 bottles of beer

Different Approaches

limited or unlimited board

array / matrix based

sparse matrix
(coordinates)

mutating or generations

optimizations

(HashLife)

different variations on
rules, visualization etc.

```java
public class GameOfLife {
    private final Set<Point> alive = new HashSet<Point>();

    public GameOfLife(Collection<Point> alive) {
        this.alive.addAll(alive);
    }

    private Set<Point> neighbours(Point p) {
        final Set<Point> result = new HashSet<Point>();
        for (int x = -1; x <= 1; x++)
            for (int y = -1; y <= 1; y++) result.add(new Point(p.x + x, p.y + y));
        return result;
    }

    private Set<Point> aliveNeighbours(Point p) {
        final Set<Point> result = new HashSet<Point>();
        for (Point neighbour : neighbours(p)) {
            if (alive.contains(neighbour)) result.add(neighbour);
        }
        return result;
    }

    private Set<Point> deadNeighbours(Point p) {
        final Set<Point> result = new HashSet<Point>();
        for (Point neighbour : neighbours(p)) {
            if (!alive.contains(neighbour)) result.add(neighbour);
        }
        return result;
    }
```

Java (1)

```
public GameOfLife next() {
    final Set<Point> stayingAlive = new HashSet<Point>();
    for (Point cell : alive) {
        final int livingNeighbourCount = aliveNeighbours(cell).size();
        if (livingNeighbourCount == 2 || livingNeighbourCount == 3) {
            stayingAlive.add(cell);
        }
    }
    final Set<Point> wakingFromDead = new HashSet<Point>();
    for (Point cell : alive) {
        final int livingNeighbourCount = deadNeighbours(cell).size();
        if (livingNeighbourCount == 2 || livingNeighbourCount == 3) {
            stayingAlive.add(cell);
        }
    }
    final HashSet<Point> result = new HashSet<Point>(stayingAlive);
    result.addAll(wakingFromDead);
    return new GameOfLife(result);
}
```

Java (2)

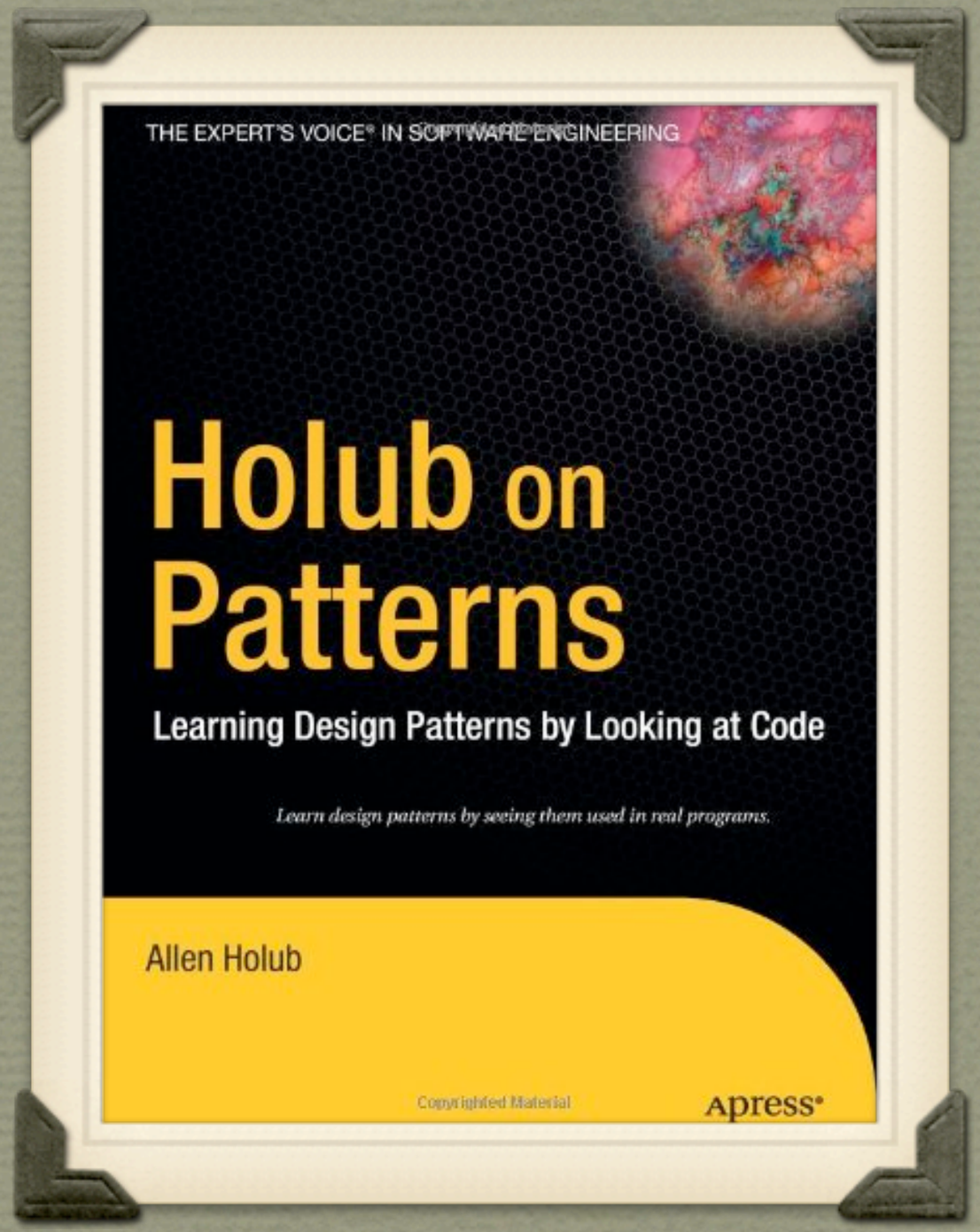
Java & Patterns

1300 Lines of Java Source

Many Patterns crammed in

It is for making a Point

But does it make its Point?



```groovy
class GameOfLife {
  static def env = [[-1,-1],[0,-1],[1,-1],[-1,0],[1,0],[-1,1],[0,1],[1,1]]
  def alive = []

  def neighbours(def cell) {
    env.collect { [cell[0]+it[0],cell[1]+it[1]] }
  }

  def aliveNeighbours(def cell) {
    neighbours(cell).findAll {alive.contains(it) }
  }

  def deadNeighbours(def cell) {
    neighbours(cell) - aliveNeighbours(cell)
  }

  def haveNeighbourCount(def coll, def counts) {
    coll.findAll { counts.contains(aliveNeighbours(it).size())}
  }

  GameOfLife next() {
    def stayingAlive = haveNeighbourCount(alive, [2,3])
    def wakingFromDead = alive.inject([]) { res,cell ->
                           res << haveNeighbourCount(deadNeighbours(cell),[3])}

    new GameOfLife(alive: new HashSet(stayingAlive + wakingFromDead))
  }
}
```

Groovy

```scala
class Generation(val alive: Set[Cell] = Set.empty) {
  require(alive != null, "Illegal argument: alive must not be null!")

  def next: Generation = {
    val stayingAlive = alive filter { (2 to 3) contains aliveNeighbours(_).size  }
    val wakingFromDead = alive flatMap deadNeighbours filter { aliveNeighbours(_).size == 3 }
    new Generation(stayingAlive ++ wakingFromDead)
  }

  private def neighbours(cell: Cell) =
    for {
      x <- (cell.x-1) to (cell.x+1)
      y <- (cell.y-1) to (cell.y+1) if (x != cell.x) || (y != cell.y)
    } yield Cell(x, y)

private def aliveNeighbours(cell: Cell) = neighbours(cell) filter { alive contains _ }

private def deadNeighbours(cell: Cell) = neighbours(cell) filter { cell => !(alive contains cell) }
```

Scala

(Heiko Seeberger)

```clojure
(defn neighbours
  "All direct neighbours around [pos] and pos itself"
  [[x y]]
  (for [dx [-1 0 1] dy [-1 0 1]] [(+ x dx) (+ y dy)])
)

(defn alive
  "is pos alive in the next turn; shortcuts for when counted to 5"
  [pos board]
  (let [cnt (count (take 5 (for [p (neighbours pos) :when (board p)] true)))]
    (if (or (= cnt 3) (and (= cnt 4) (board pos))) pos nil)
))

(defn all-neighbours
    "all direct neighbours of all living cells on the board"
 [size board]
 (let [on-board (fn [[x y]] (and (>= x 0) (< x size) (>= y 0) (< y size)))]
   (set (filter on-board (mapcat #(neighbours %) board)))
 )
)

(defn next-board
  "calculates the new board"
  ([board] (next-board size board))
  ([bsize board]
      (set (filter #(alive % board)
                   (all-neighbours bsize board))))
)
```

Clojure

```
GameOfLife = Origin mimic do(
 initialize = method(rows:, columns:,
   @grid = Grid withDimensions(rows, columns))

 evolve = method(
   nextGrid = grid blankGrid
   newCells       = grid filter(live?) filter(numLiveNeighbours in?(2..3))
   survivingCells = grid reject(live?) filter(numLiveNeighbours == 3)

   (newCells + survivingCells) each(cellData, nextGrid spawnCell(cellData row, cellData col))

   @grid = nextGrid
 )

 spawnCell = method(row, col,
   grid tap(spawnCell(row, col)))
)

GameOfLife Grid = Origin mimic do(
 mimic!(Mixins Sequenced)

 blankGrid = method("Reset the grid", with(state: Dict withDefault(0)))
 spawnCell = method("Animate a given cell", row, col, state[[row, col]] = 1)
 seq = macro(allCells seq)

 ;internal methods
 permutations = method(a, b, a flatMap(i, b map(j, (i,j))))

 countLiveNeighbours = method("Count the number of live neighbours for a given set of cell coordinates",
   row, col,
   neighbourCoords = permutations((-1..1), (-1..1)) - [(0,0)]
   neighbourCoords inject(0, sum, (r_mod,c_mod), state[[row + r_mod, col + c_mod]] + sum))
)
```

Ioke (Sam Aaron)

```
      disp +/ +⌿ 1 0 ¯1 ∘.∘ 1 0 ¯1 ⌽⍤1 ⊂R
┌→──────────────┐
↓0 0 1 2 2 1 0 │
 0 1 3 4 3 1 0 │
 0 1 4 5 4 1 0 │
 0 1 3 3 2 0 0 │
 0 0 1 1 1 0 0↓
└──────────────┘

      disp 3 4 ≡ +/ +⌿ 1 0 ¯1 ∘.∘ 1 0 ¯1 ⌽⍤1 ⊂R
┌→─────────────┬→─────────────┐
↓0 0 0 0 0 0 0 │0 0 0 0 0 0 0 │
 0 0 1 0 1 0 0 │0 0 0 1 0 0 0 │
 0 0 0 0 0 0 0 │0 0 1 0 1 0 0 │
 0 0 1 1 0 0 0 │0 0 0 0 0 0 0 │
 0 0 0 0 0 0 0↓│0 0 0 0 0 0 0↓│
└──────────────┴──────────────┘

      disp 1 R 3 4 ≡ +/ +⌿ 1 0 ¯1 ∘.∘ 1 0 ¯1 ⌽
```

Dyalog APL

Scaling Up

- partitioning / sharding the data
- actors as cells, need supervisors and segmentation
  - asynchronity / generations problem
- map reduce approach, a cell bleeds its life into its neighbours, sum the resulting fragments
- use image operations / GPU
- hashlife exploits regularities, hash in a quadtree

Think Different

```
private static final int COLOR = 0x484848;
private final ConvolveOp convolve = new ConvolveOp(new Kernel(3, 3, new float[]{
        1,   1,    1,
        1,   0.5f, 1,
        1,   1,    1 }));

private final LookupOp lookup = createLookupOp();

private LookupOp createLookupOp() {
    final short[] lookupTable = new short[256];
    lookupTable[180] = 0x48;
    lookupTable[216] = 0x48;
    lookupTable[252] = 0x48;
    return new LookupOp(new ShortLookupTable(0, lookupTable), null);
}


private void nextGeneration() {
    image=convolve.filter(image, null);
    lookup.filter(image, image);
    generation++;
}
```

Java Image Op

It does not end here

- the language is just the first step for better abstractions
- create your own languages & vocabulary
- DDD, ubiquitous language, domain model
- DSLs
- libraries, APIs
- design principles - KISS, DRY & more
- design patterns

Thank You!

Questions & Comments?

Image Attributions: Wikipedia, SXC.HU