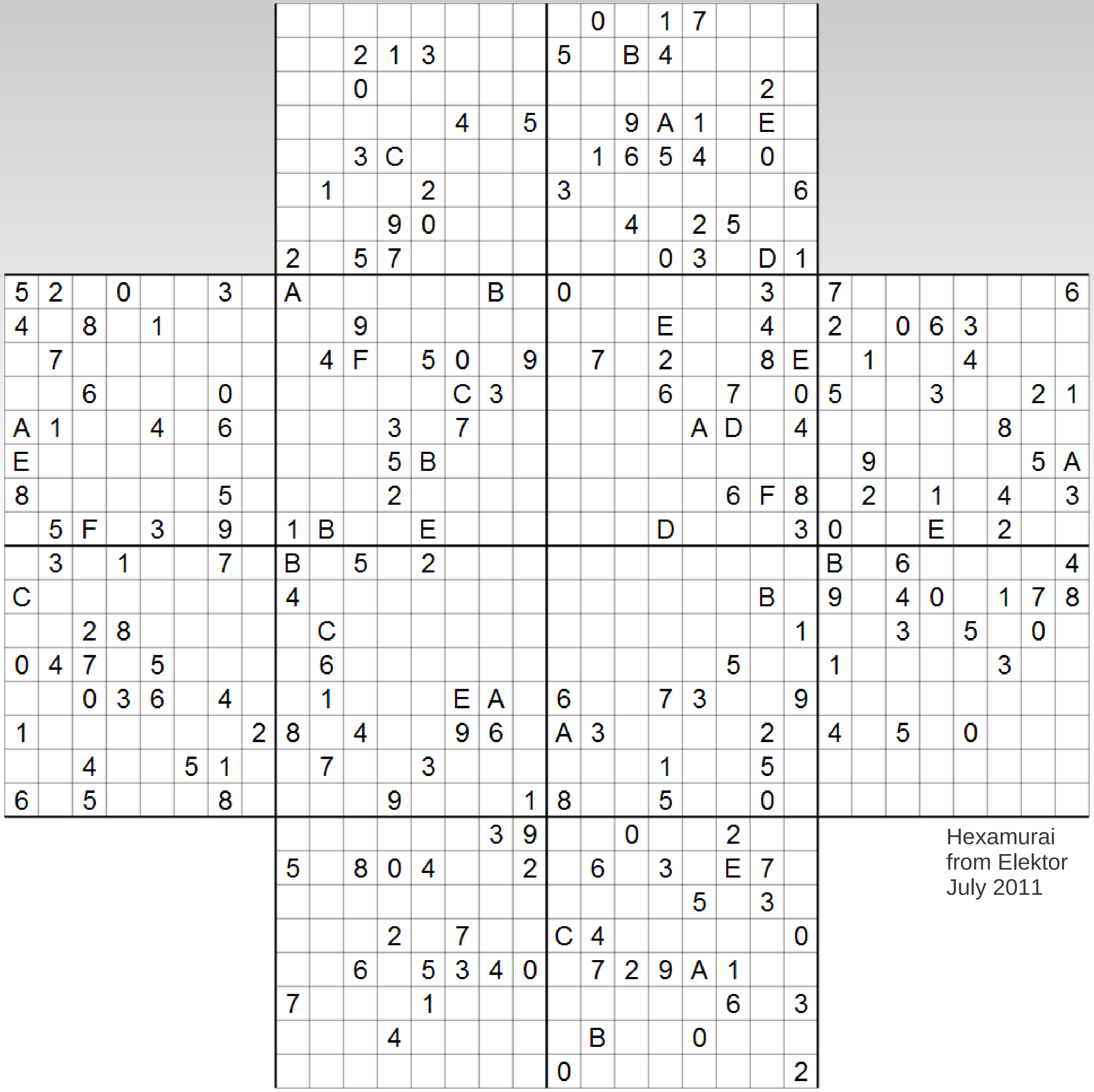
Doku



David Grayson
Las Vegas
Ruby Meetup
2012-4-18

Hexamurai
from Elektor
July 2011

Doku is a Ruby gem for solving Sudoku-like puzzles

Sudoku

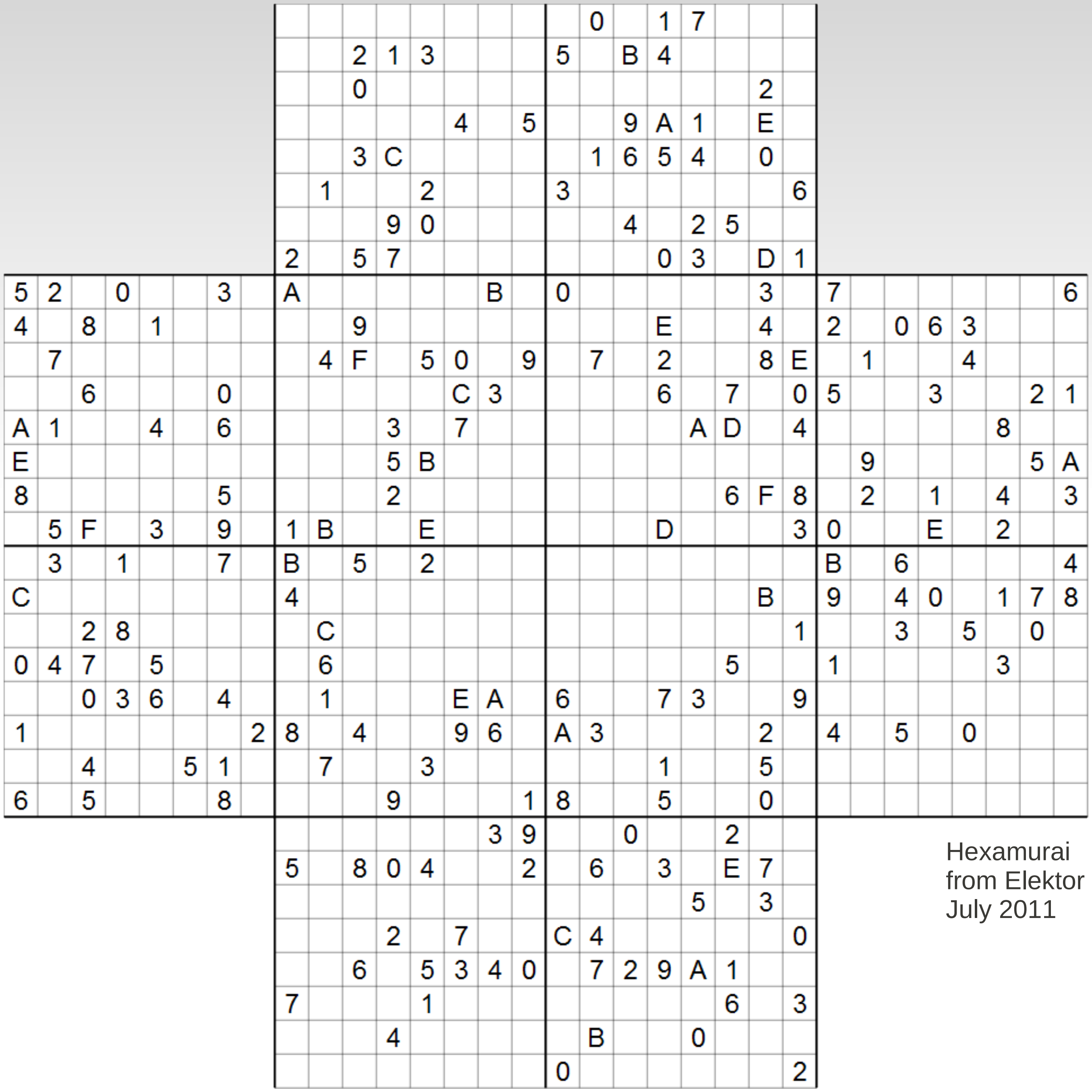
					8			
		7		3	5			9
5			4		6	8		
					4	2		
4							3	7
8						5		
	9			6	7			
		3				1		5
								3

Hexadoku

2	A		7		C			9	D	6	4	8			
		3		A			D	7					2	F	
	1					0		8					4	A	B
					7		2			B	C		0		3
C	2		8		D	3				4	E				
		F	A	7		2		B		3		1	C	0	4
				4			F			1					E
9			B	1								2	3		
		8	C								0	3			D
6					F			1			A				
D	3	7	E		0		1		9		8	A	F		
				3	B				2	D		C		8	0
F		B		5	1			2		A					
3	C	A					7		E					6	
	E	4					9	3			5		D		
			1	F	3	A	4			9		5		E	2

Hexamurai
from Elektor
Oct 2011

Hexamurai



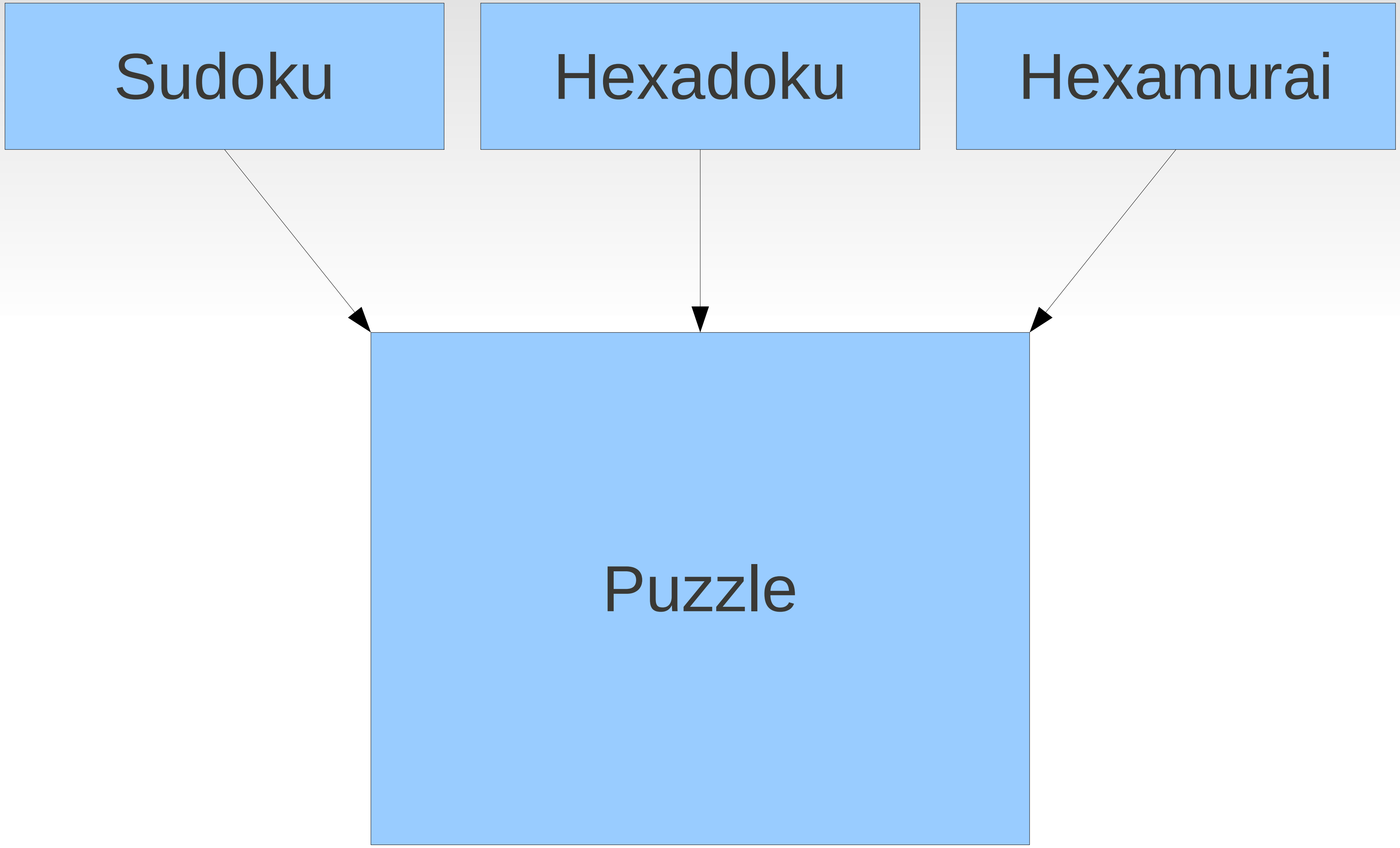
Hexamurai
from Elektor
July 2011

Demo

About Doku

- Written purely in Ruby
- Short methods
- Well-defined objects and classes
- Complete documentation
- Fully functional classes
- Test-driven development
- Lots of time refactoring
- Plus, it can generate SVGs!

Simplified Class Structure



Puzzle class is general

- Every subclass has these attributes:
  - `glyphs` (e.g. 1,2,3,4,5,6,7,8,9)
  - `squares` (every spot on the puzzle)
  - `groups` (sets of squares)
- Every instance has:
  - `glyph_state`: Hash associating squares to glyphs.
- Squares and glyphs can be any ruby object.

Solving a Puzzle

- Solution is a set of glyph assignments
  - e.g. write 6 in the square at (3,4)
- Solution achieves certain goals exactly once:
  I. For each square, assign ONE glyph to it.
  II. Assign every glyph to every group ONCE.
- Each glyph assignment achieve a subset of these goals.

Sudoku-like puzzles can be reduced to exact cover problems!

Exact cover problem

- Given: universe set
- Given: several of subsets of the universe
- Problem: Choose some of those subsets so that every element in the universe set appears exactly once.

Exact cover example

```
Universe: [A,B,C,D,E,F,G]

Subsets:  [C,E,F]
          [A,D,G]
          [B,C,F]
          [A,D]
          [B,G]
          [D,E,G]
```

Exact cover example

```
Universe: [A,B,C,D,E,F,G]

Subsets:  [    C,  E,F  ]
          [A,    D,    G]
          [  B,C,    F  ]
          [A,    D      ]
          [  B,        G]
          [      D,E,  G]

Solution: [  B,        G]
          [A,    D      ]
          [    C,  E,F  ]
```

Algorithm X

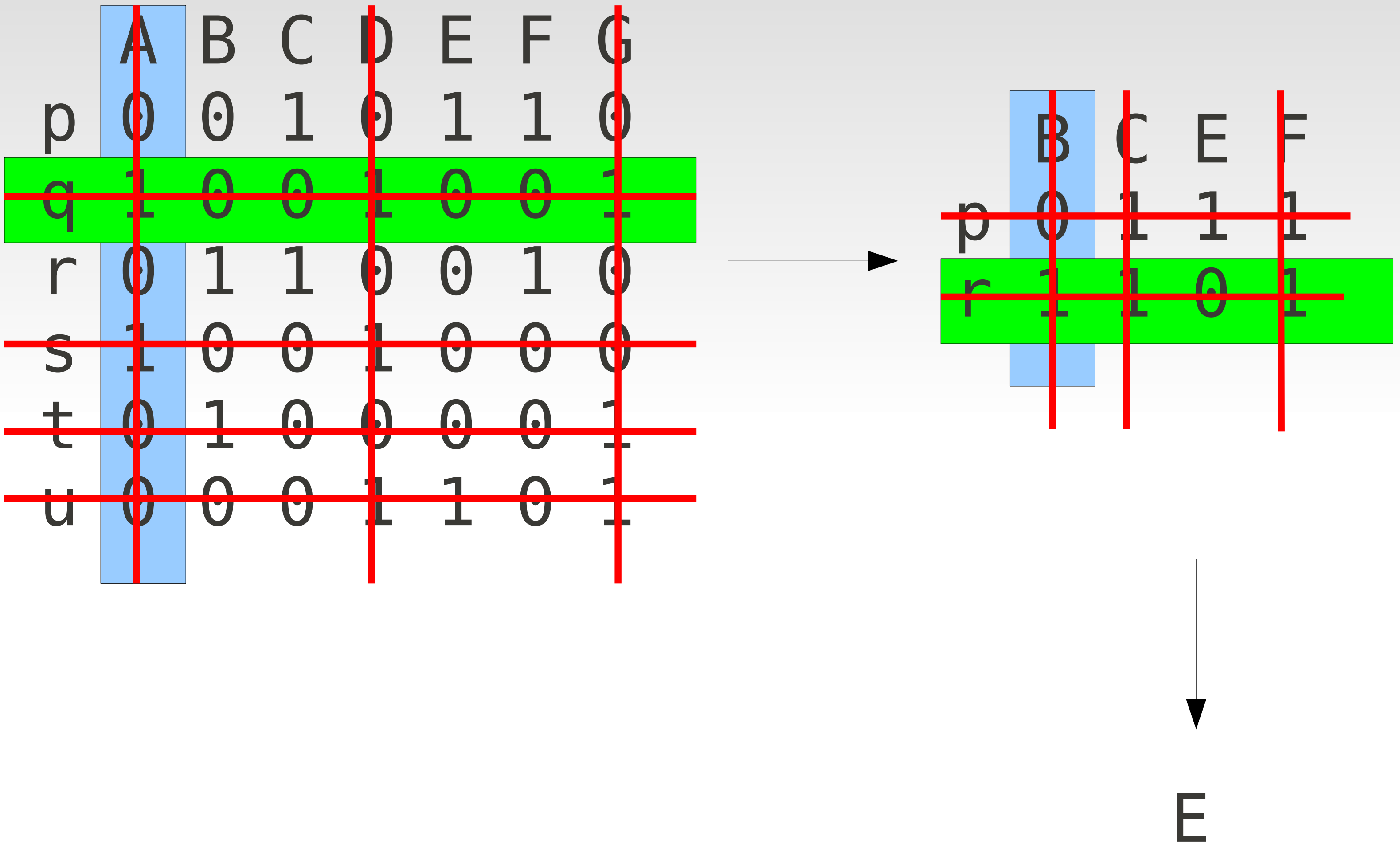
Subsets

```
[    C,  E,F  ]
[A,    D,    G]
[  B,C,    F  ]
[A,    D      ]
[  B,         G]
[      D,E,  G]
```

Matrix

```
0 0 1 0 1 1 0
1 0 0 1 0 0 1
0 1 1 0 0 1 0
1 0 0 1 0 0 0
0 1 0 0 0 0 1
0 0 0 1 1 0 1
```

Algorithm X Demo



	A	B	C	D	E	F	G
p	0	0	1	0	1	1	0
q	1	0	0	1	0	0	1
r	0	1	1	0	0	1	0
s	1	0	0	1	0	0	0
t	0	1	0	0	0	0	1
u	0	0	0	1	1	0	1

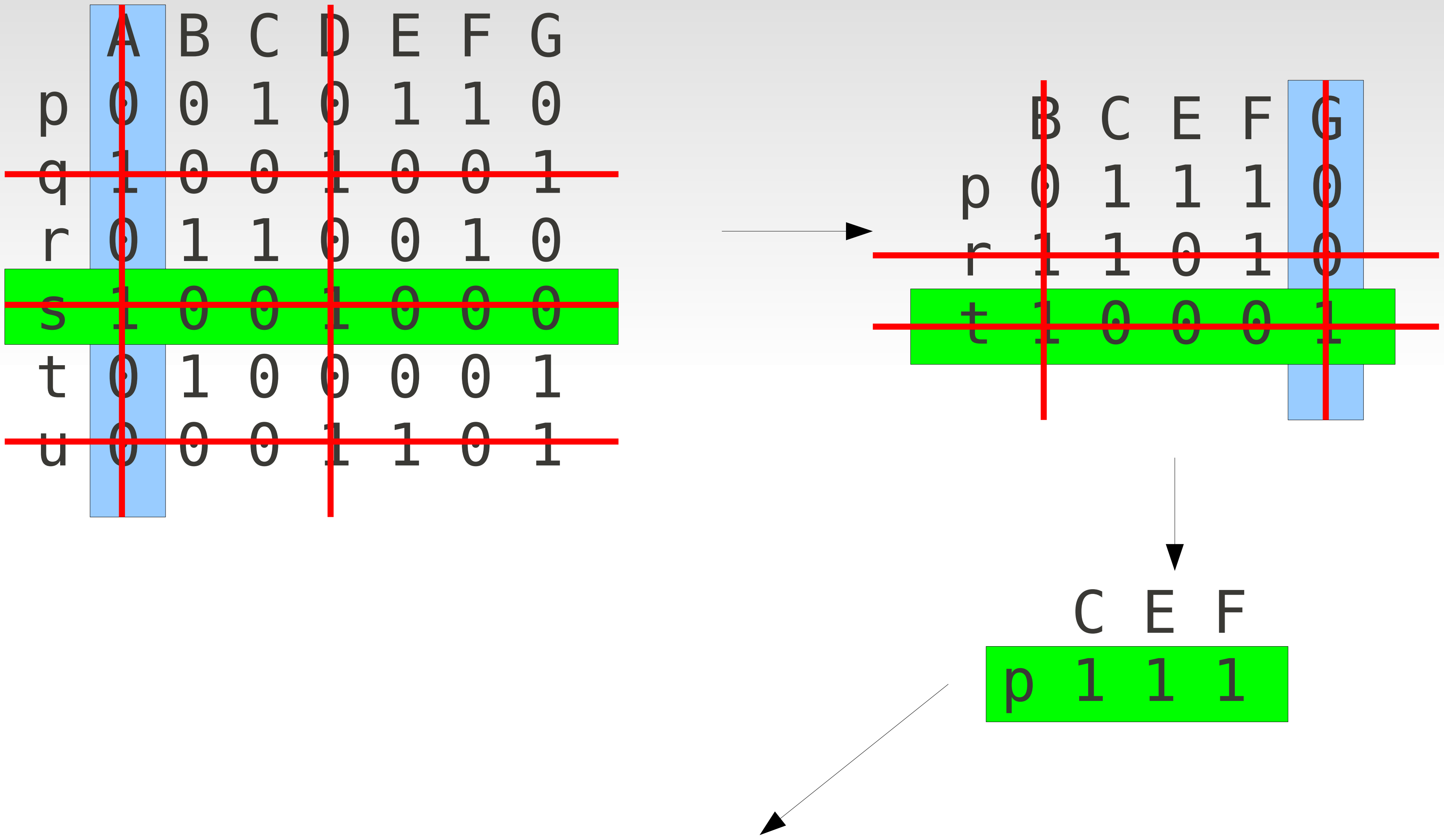
	B	C	E	F
p	0	1	1	1
r	1	1	0	1

E

Algorithm X Demo

$$
\begin{array}{c|ccccccc}
 & A & B & C & D & E & F & G \\
p & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
q & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
r & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
s & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
t & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
u & 0 & 0 & 0 & 1 & 1 & 0 & 1
\end{array}
\longrightarrow
\begin{array}{c|cccc}
 & B & C & E & F \\
p & 0 & 1 & 1 & 1 \\
r & 1 & 1 & 0 & 1
\end{array}
$$

Algorithm X Demo



$$\begin{array}{c|ccccccc} & A & B & C & D & E & F & G \\ p & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ q & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ r & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ s & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ t & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ u & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{c|ccccc} & B & C & E & F & G \\ p & 0 & 1 & 1 & 1 & 0 \\ r & 1 & 1 & 0 & 1 & 0 \\ t & 1 & 0 & 0 & 0 & 1 \end{array}$$

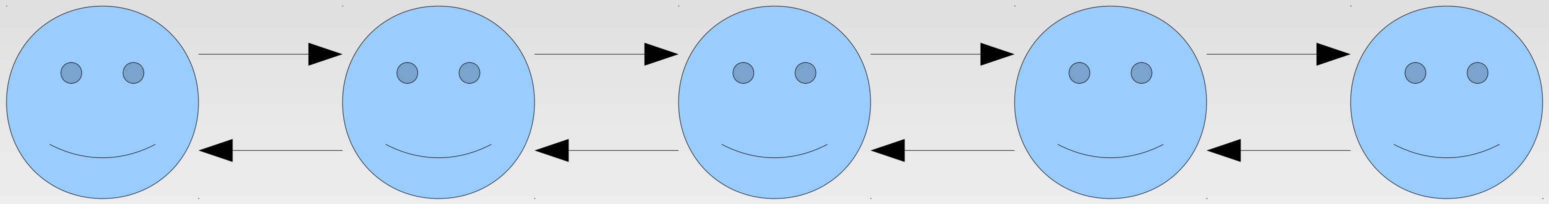
$$\begin{array}{c|ccc} & C & E & F \\ p & 1 & 1 & 1 \end{array}$$

Empty matrix = success!

Efficient Algorithm X

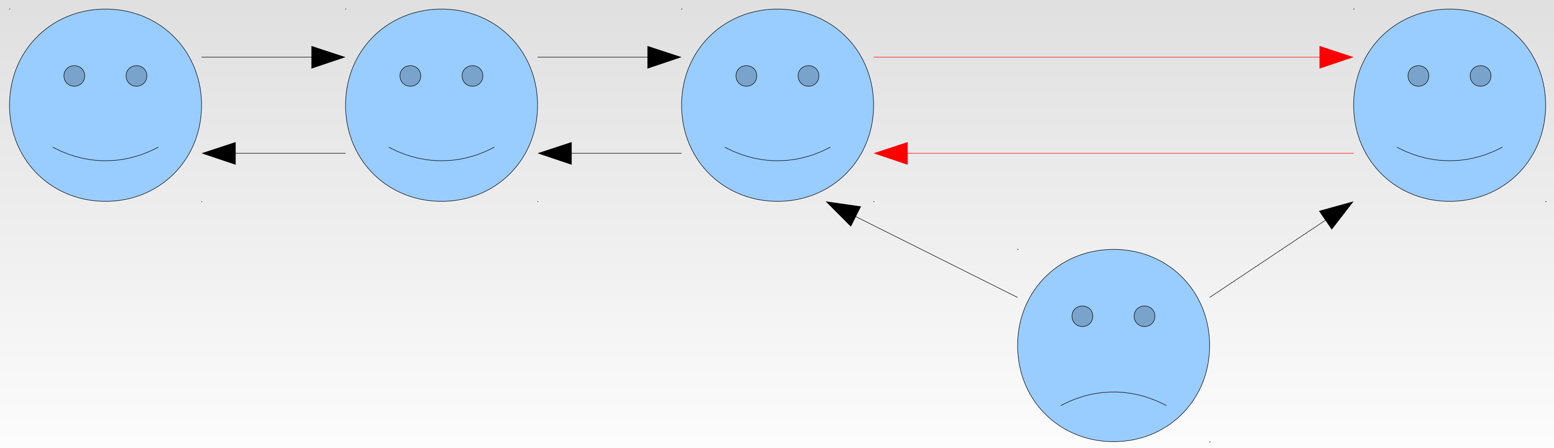
- Data type for a large matrix
- Finding 1s in a given column or row quickly
- Removing columns and rows quickly
- Quick reinserting

Dancing Links Intro



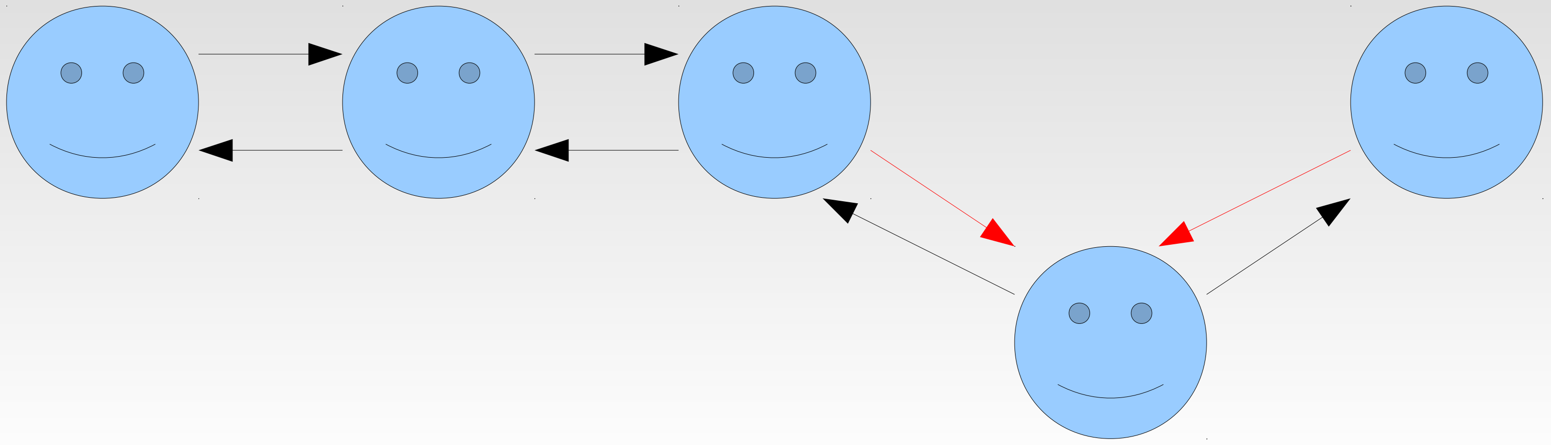
```
class Node
  attr_accessor :left, :right
end
```

Easy removal



```
class Node
  attr_accessor :left, :right

  def remove
    left.right = right
    right.left = left
  end
end
```

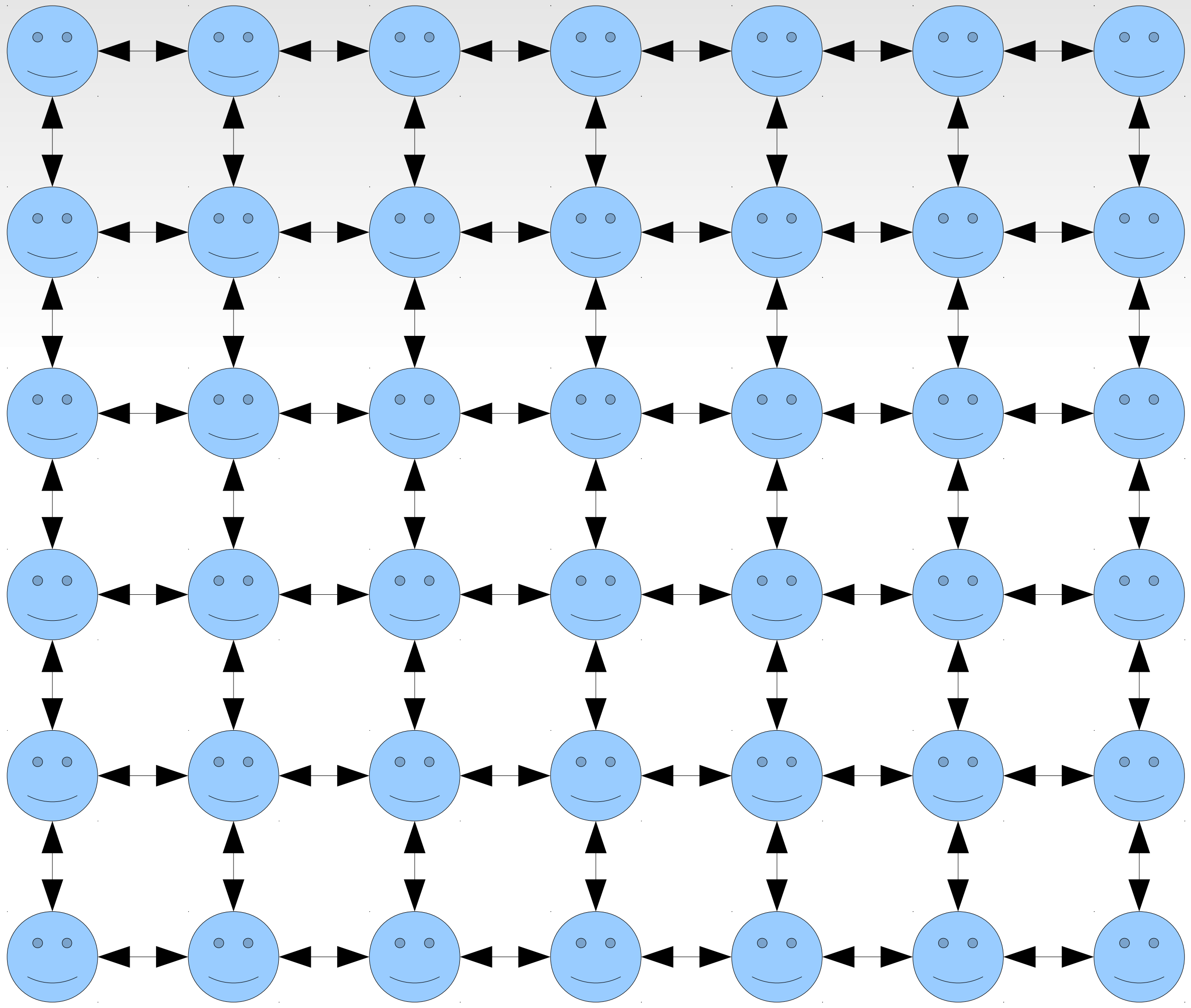
Also easy reinsertion!



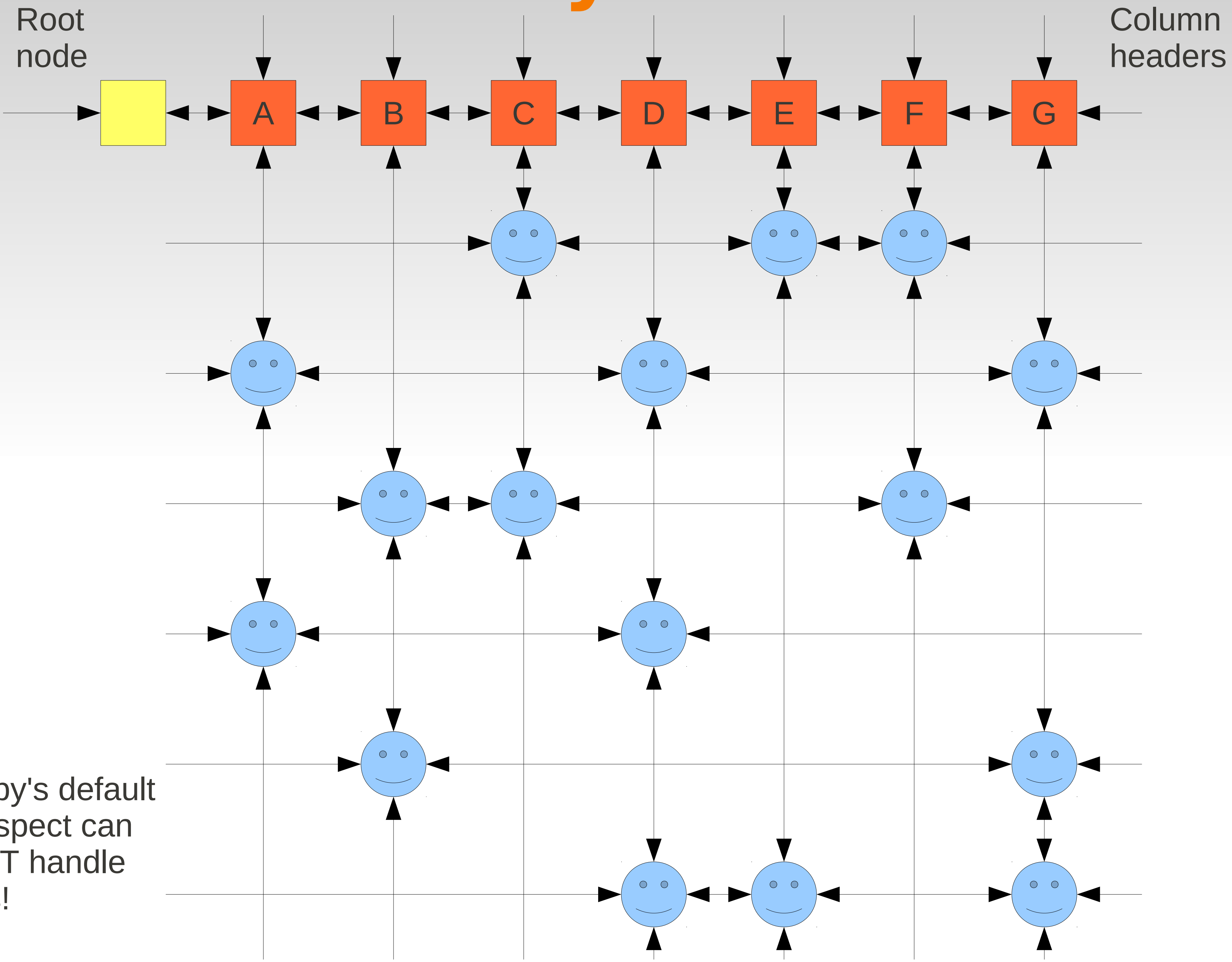
```
class Node
  attr_accessor :left, :right

  def remove
    left.right = right
    right.left = left
  end

  def reinsert
    left.right = right.left = self
  end
end
```

No arguments
For reinsert!

2D doubly-linked list



2D doubly-linked list



Ruby's default #inspect can NOT handle this!

HorizontalLinks module

```ruby
module HorizontalLinks
  include Uninspectable

  def self.included(klass)
    klass.instance_eval do
      attr_accessor :left, :right
    end
  end

  def remove_horizontal
    right.left, left.right = left, right
  end

  def reinsert_horizontal
    left.right = right.left = self
  end

  def insert_left(obj)
    self.left, self.right = obj.left, obj
    reinsert_horizontal
  end
end
```

LinkEnumerator class

```
class LinkEnumerator
  include Enumerable

  def initialize(link, start, include_start=false)
    @link, @start, @include_start = link, start, include_start
  end

  def each
    yield @start if @include_start

    n = @start
    while true
      n = n.send @link
      return if n == @start
      yield n
    end
  end
end
```

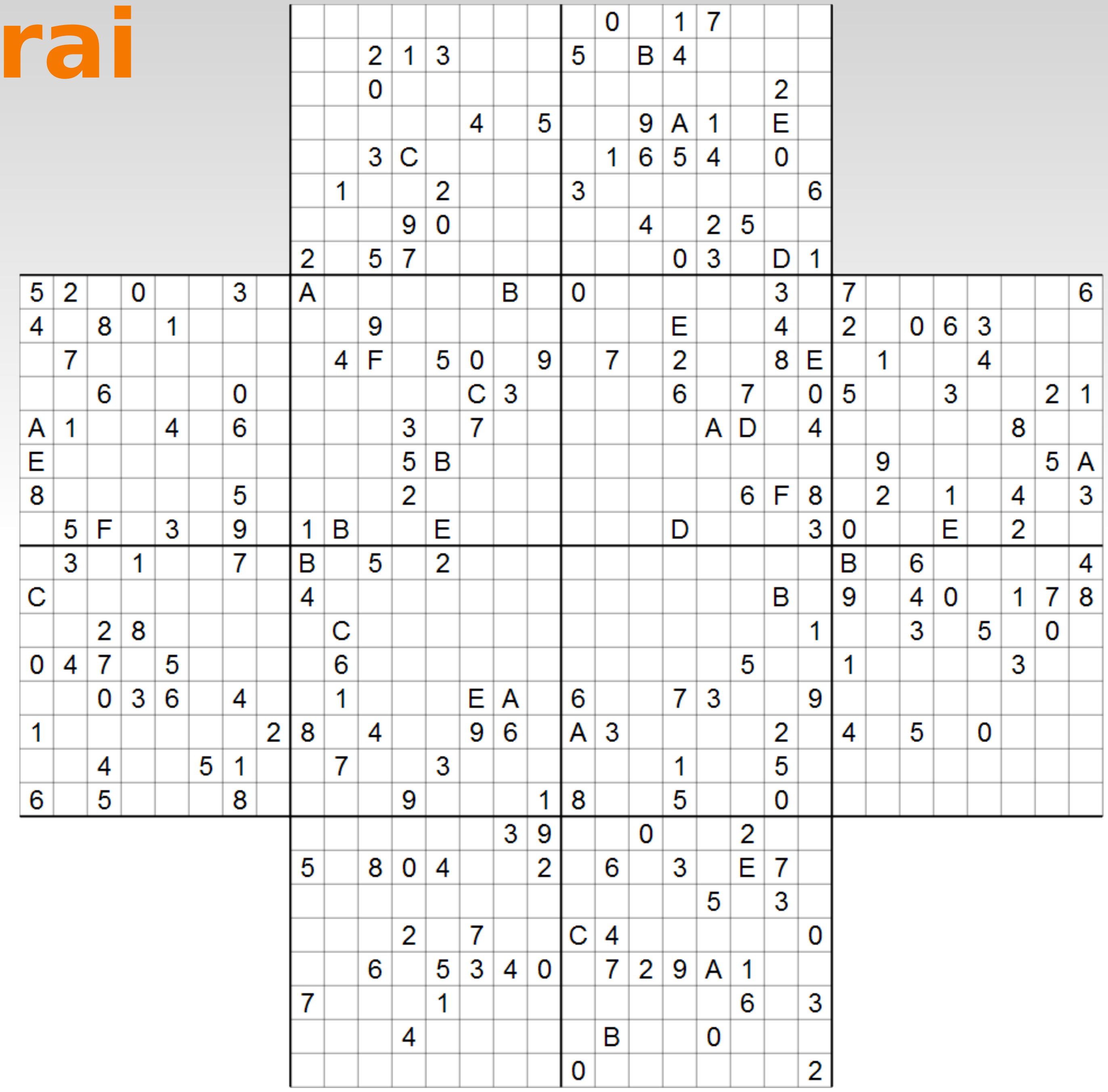
Hexamurai

768 squares

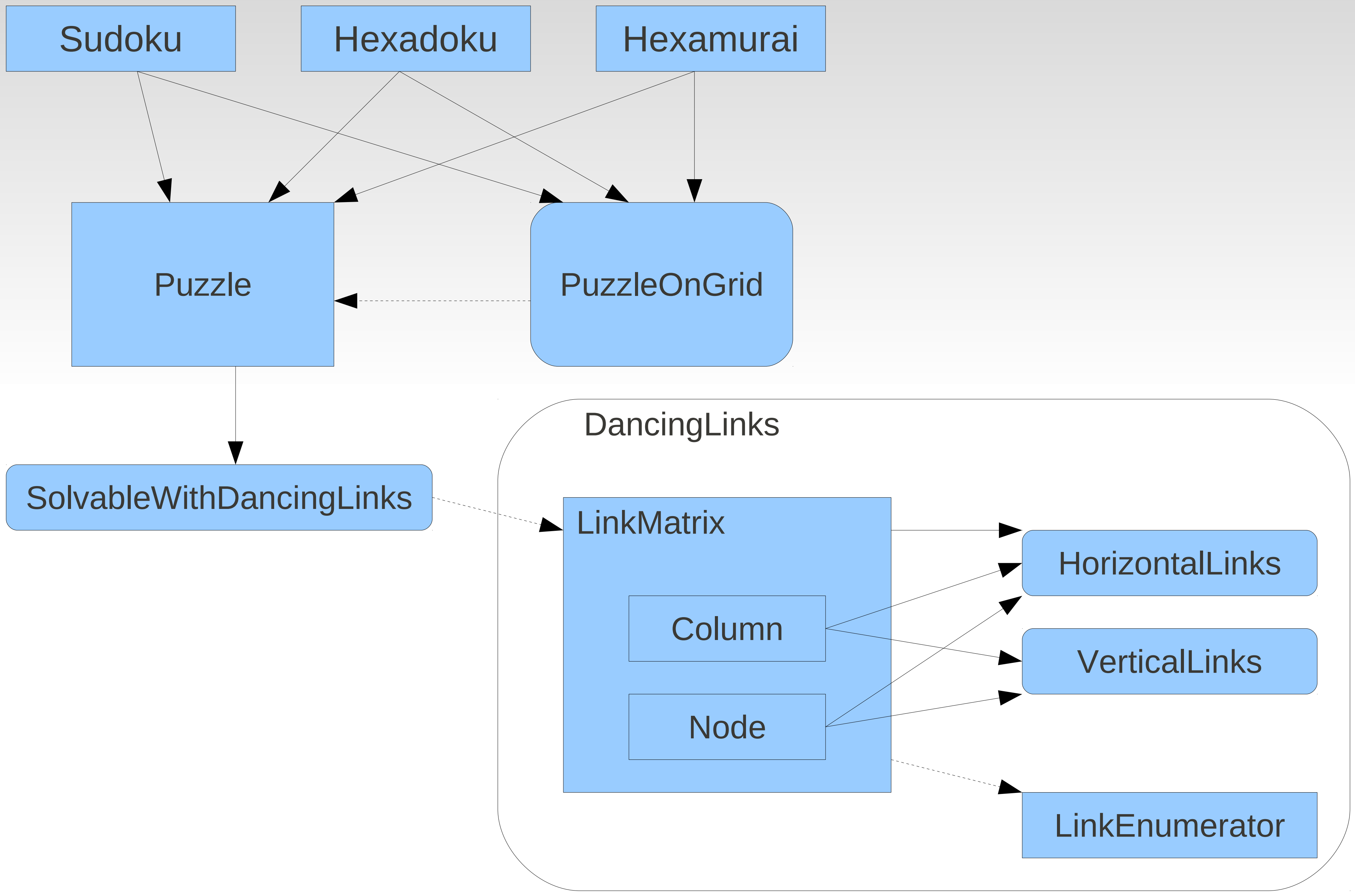
2872 goals

3433 subsets

13 minutes
to find ALL
solutions!



Architecture



The end

Hidden slide