	Q		
			Q
Q			
		Q	

n = 4

	1	
		1
1		

n = 4

0	1	0	0	
0	0	0	1	
1	0	0	0	BINARY ??!??!!!!!!
0	0	1	0	

0	1	0	0	4
0	0	0	1	1
1	0	0	0	8
0	0	nary	0	2  decimal

0	1	0	0	4	<b>2</b> <sup>2</sup>	Wa
0	0	0	1	1	<b>2</b> <sup>0</sup>	Working
1	0	0	0	8	<b>2</b> <sup>3</sup>	solution
0	0	1	0	2	21	ion
	bir	nary		decimal	base 2	

forget about the 1s and 0s

array indices

rowPossibilities = [4, 1, 8, 2];

array index

decimal

base 2



	Working	S	ion
<b>2</b> <sup>2</sup>	<b>2</b> <sup>0</sup>	<b>2</b> <sup>3</sup>	21
4	1	8	2
0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

# red = primary concept

yellow = secondary

failing solution: row conflict

				12	
0	0	0	1	1	<b>2</b> <sup>0</sup>
				8	
0	0	1	0	2	21

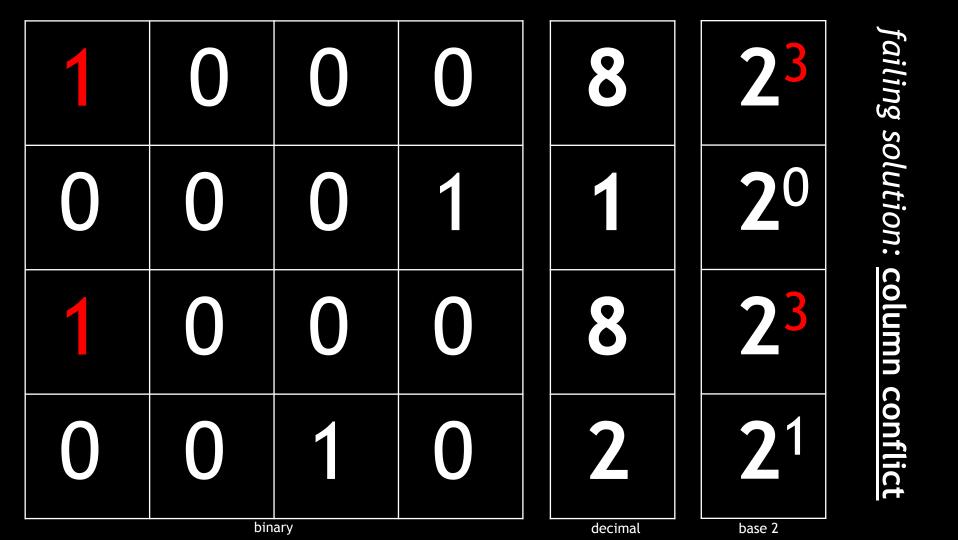
binary

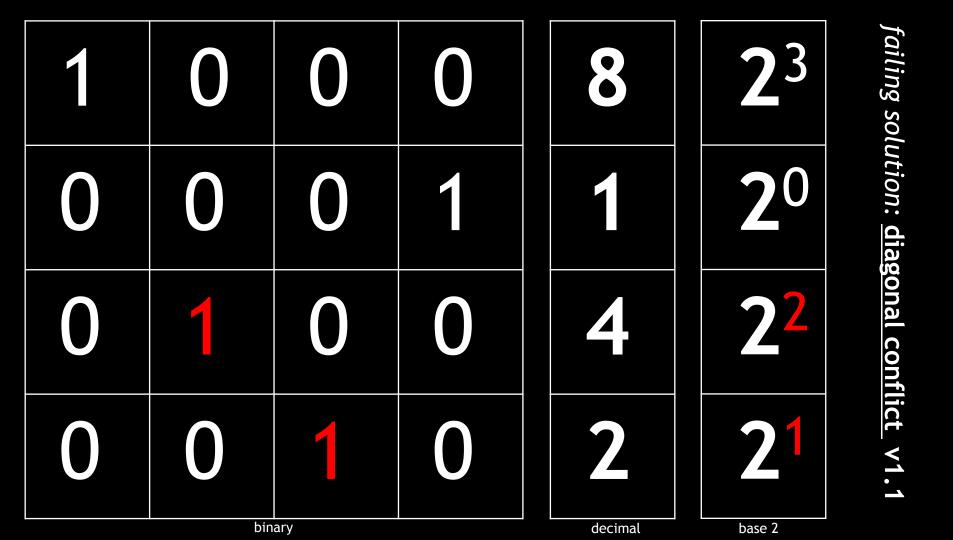
decimal

base 2

failing solution: row conflict

1	1	0	0	12	2?
0	0	0	1	1	<b>2</b> <sup>0</sup>
0		0	0		2?
0	0	nary	0	2 decimal	21 base 2





- 3	)	===	abs(	2	_1
( -1	)	===	abs(	1	)
	1	===	1		

abs(2

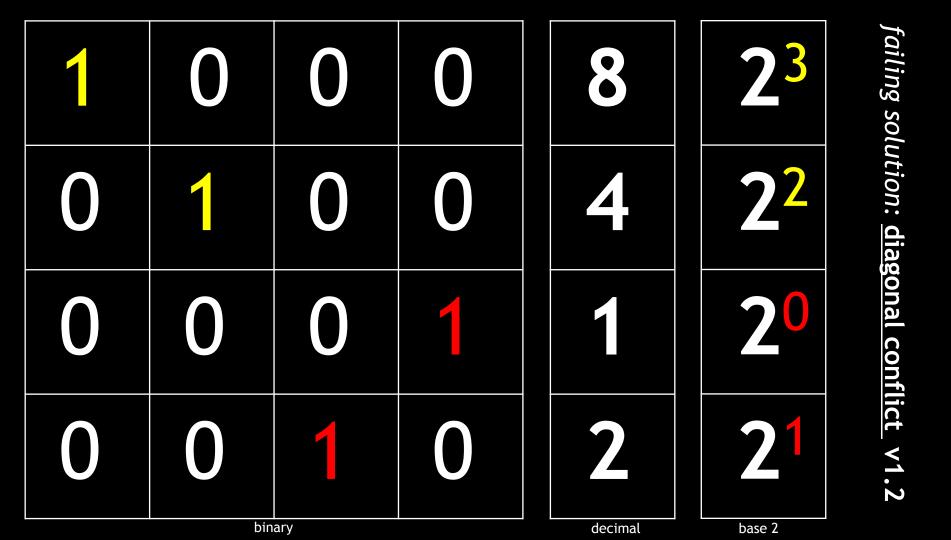
abs(

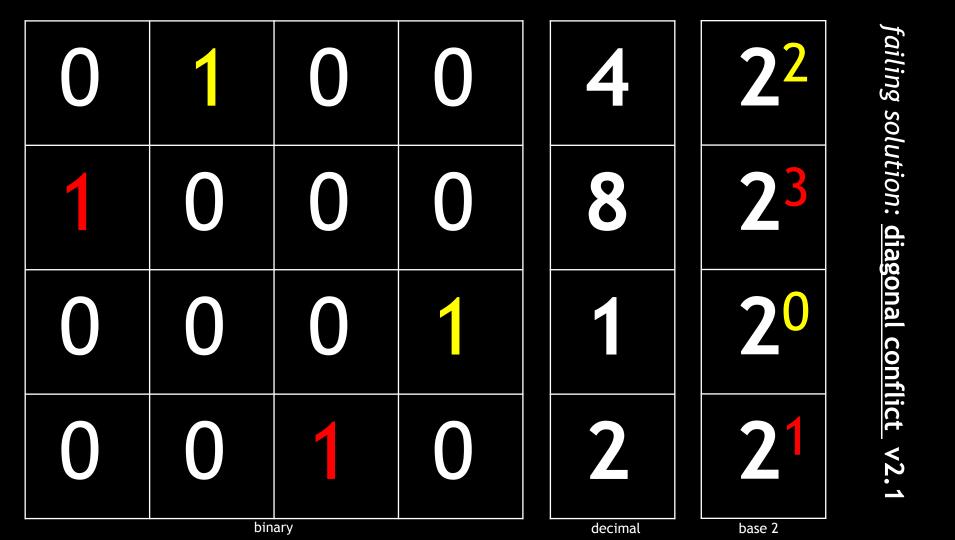


array index

decimal

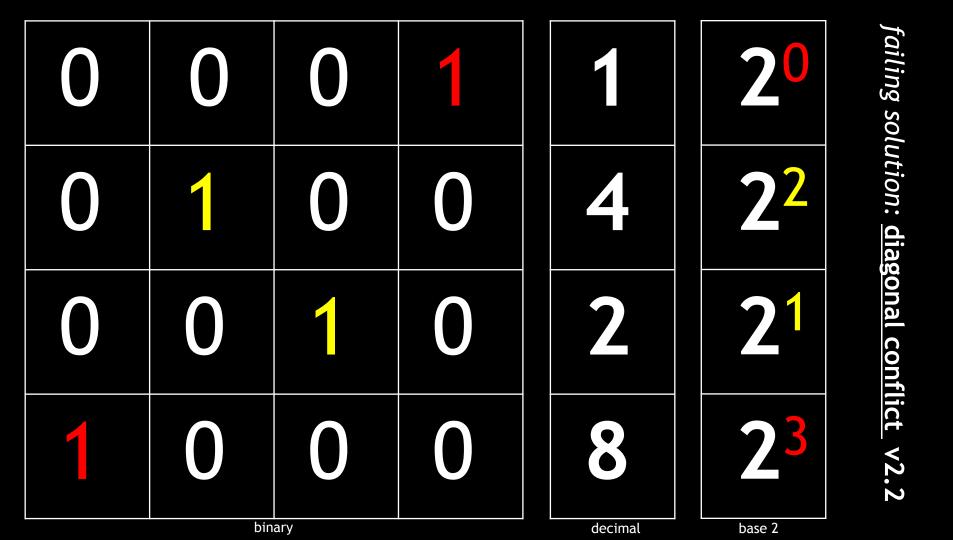






abs(1-3) === abs(3-1)
abs(-2) === abs(2)
2 === 2





abs(0-3) === abs(0-3)abs(-3) === abs(3)3 === 3



conflicts recap:

#### row numbers must be base 2, else *row conflict*

can't be 12 because in binary 12 is 1100 and contains more than one 1

### no identical numbers, else column conflict

can't have 8 and 8 because 1000 and 1000 stack 1s

if the absolute value of the difference in array indices = the absolute value of the difference in base 2 exponents, there exists a *diagonal conflict* 

if n = 2 and board =  $[4, 2] = [2^2, 2^1]$ , abs(base 2 diff) === abs(index diff)  $\rightarrow$  abs(2 - 1) === abs(0 - 1)

## "coding is not fast;

it's 10% typing and 90% thinking.

getting a solution is a process, not a binary light switch."

-Marcus Phillips

## Bitwise Operators

let's jump into the chrome console...

#### **input**: n, a number, representing an n by n board with n non-conflicting queens

- 0. declare a solution counter variable equal to 0.
- 1. get an array of potential rows for an  $n \times n$  board.

```
n = 5 \rightarrow [1, 2, 4, 8, 16]
```

- 2. declare recursive function expression that searches and potentially appends new rows.
  - 2.0 <u>base case</u> if board length = n, add to solutionCounter
  - 2.1 <u>initialization</u> iterate over the list and begin a board search using an array containing each. search([1]), search([2]), search([4]) . . .
  - 2.2 <u>recursion</u> -using our **row**, **column**, and **diagonal** conditions, check to see if we can append from our potential rows array. If all tests pass:
    - 2.2.0 append potential row to board and call recursive search on new board
- 3. invoke recursive search function on empty array
- 4. return solutionCounter

**output**: number of solutions for *n* queens

#### just an outline!

```
var bitwiseNQueens = function(n) {
  var rowOptions = [1];
                                                 0) creating row of base 2 options
  for(var i = 1; i < n; i++) {
    rowOptions.push(rowOptions[i-1] << 1);</pre>
                                                 1) declaring counter variable
  var solutionsCounter = 0;
  var findSolutions = function(board) {-
                                                 2) recursive function expression
  };
  findSolutions([]);
                                                 3) recursive function invocation
  return 'A board with ' + n + ' queens has ' + solutionsCounter + ' solutions.
                                                 4) return # of solutions
};
```

```
if(board.length === n) {
                                                                                                  2.0) base case
                                  return solutionsCounter++;
                                                                                                    updating solutionCounter if full board n x n board exists
                               // initializer
                               if(board.length === 0) {
                                                                                                  2.1) dealing with initial '[]' argument findSolutions([]);
                                  _.each(rowOptions, function(option){
                                                                                                   recurses on singly populated array
                                    findSolutions([option])
                                  });
                               // recursion
                               if(board.length > 0){
                                                                                                  2.2) iterating over potential next row options
                                  _.each(rowOptions, function(potentialRow){
                                                                                                  2.3) keeping track of failures
                                    var failures = 0;
                                    _.each(board.reverse(), function(rowToCompare, i){ 2.4) checking: iterating over current board, backwards
                                                                                                                  2.4.0) checking column conflicts
                                       if(potentialRow === rowToCompare |
                                          potentialRow === (rowToCompare >> (board.length - i)) ||
                                                                                                                   2.4.1) checking both diagonal conflicts
                                          potentialRow === (rowToCompare << (board.length - i))) {</pre>
                                                                                                                   while accounting for absolute
                                           failures++;
                                                                                                                   changes in:
                                                                                                                    1) base 2 exponent
                                    });
                                                                                                                    2) indices
                                    if(!failures){
                                                                                                   2.5) if the potential row passed all the tests
                                                                                                   set potentialRow to confirmedRow for readability
                                       var confirmedRow = potentialRow;
                                                                                                   slice our current board to avoid modification conflict
                                       var newBoard = board.slice();
                                                                                                   add our newly tested row to the board
                                       newBoard.push(confirmedRow);
                                                                                                   test it!
                                       findSolutions(newBoard);
underscore!
                                  });
                             };
```

var findSolutions = function(board) {