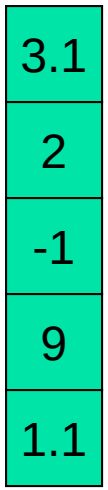
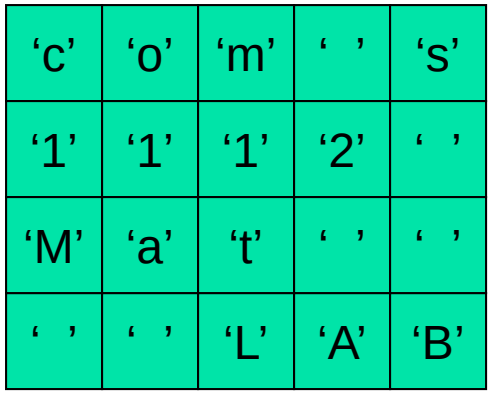
- Previous Lecture:
  - Intro to cell arrays
  - File input/output
- Today's Lecture:
  - More on cell arrays
  - Detailed file I/O example
- Announcements:
  - Project 5 due Thursday at 11pm
  - Prelim 2 on Nov 6th (Tues) at 7:30pm
  - Review questions will be posted and there'll be a review session Sunday Nov 4th 1-2:30pm in HLS B14

Matrix vs. Cell Array

Vectors and matrices store values of the same type in all components

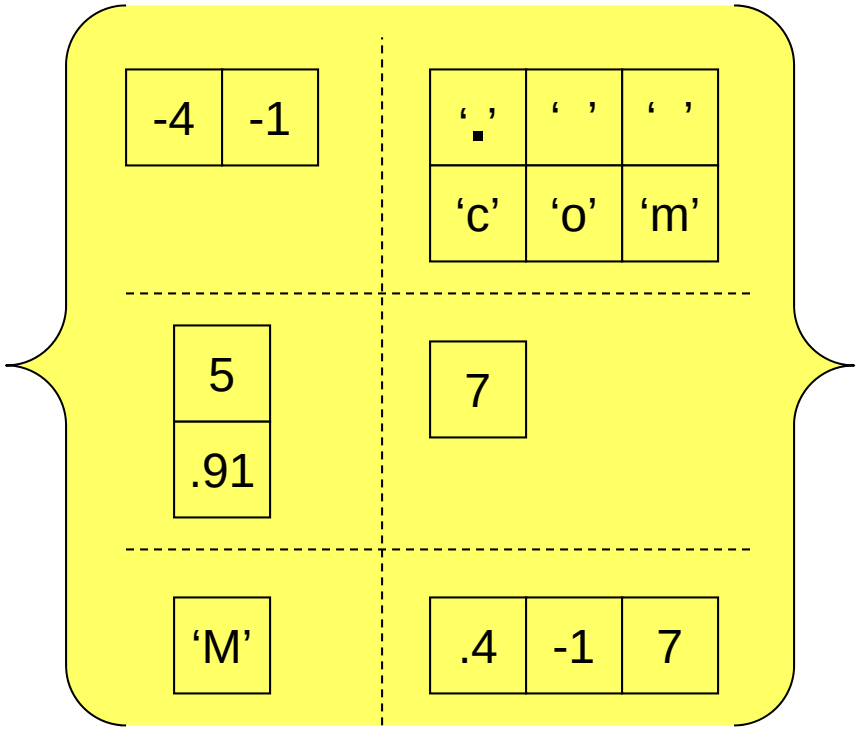


5 x 1 matrix



4 x 5 matrix

A cell array is a special array whose individual components may contain different types of data



3 × 2 cell array

Example: Represent a deck of cards with a cell array

```
D{1} = 'A Hearts';
D{2} = '2 Hearts';
          :
D{13} = 'K Hearts';
D{14} = 'A Clubs';
          :
D{52} = 'K Diamonds';
```

But we don't want to have to type all combinations of suits and ranks in creating the deck... How to proceed?

Make use of a suit array and a rank array …

```
suit = {'Hearts', 'Clubs', …
        'Spades', 'Diamonds'};

rank = {'A','2','3','4','5','6',…
    '7','8','9','10','J','Q','K'};
```

Then concatenate to get a card. E.g.,

```
str = [rank{3} ' ' suit{2} ];
D{16} = str;
```

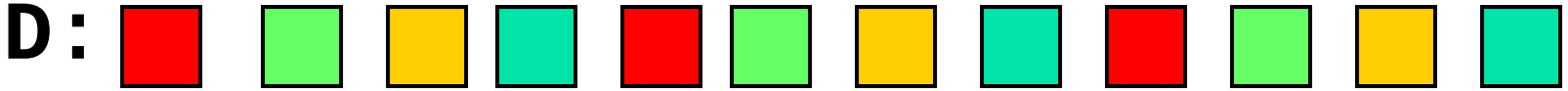
So **D{16}** stores **'3 Clubs'**

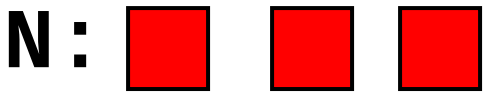
To get all combinations, use nested loops

```
i = 1;  % index of next card

for k= 1:4
   % Set up the cards in suit k
   for j= 1:13
      D{i} = [ rank{j} ' ' suit{k} ];
      i = i+1;
   end
end
```

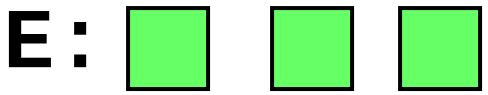
See function **CardDeck**

Example: deal a 12-card deck

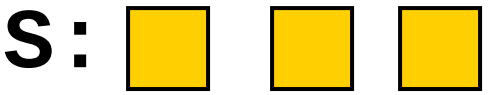




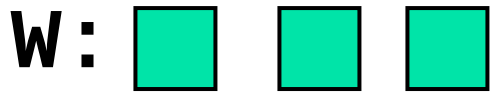
1,5,9 4k-3



2,6,10 4k-2



3,7,11 4k-1

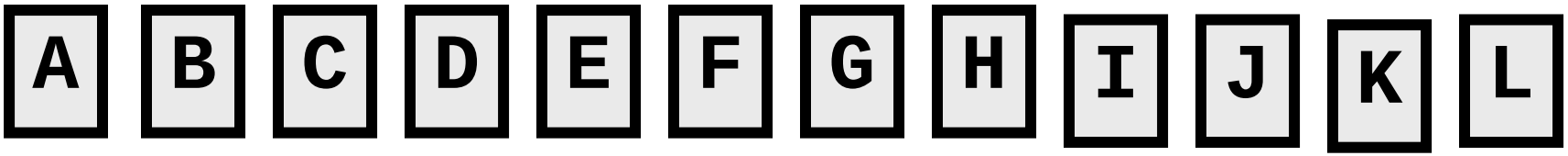


4,8,12 4k

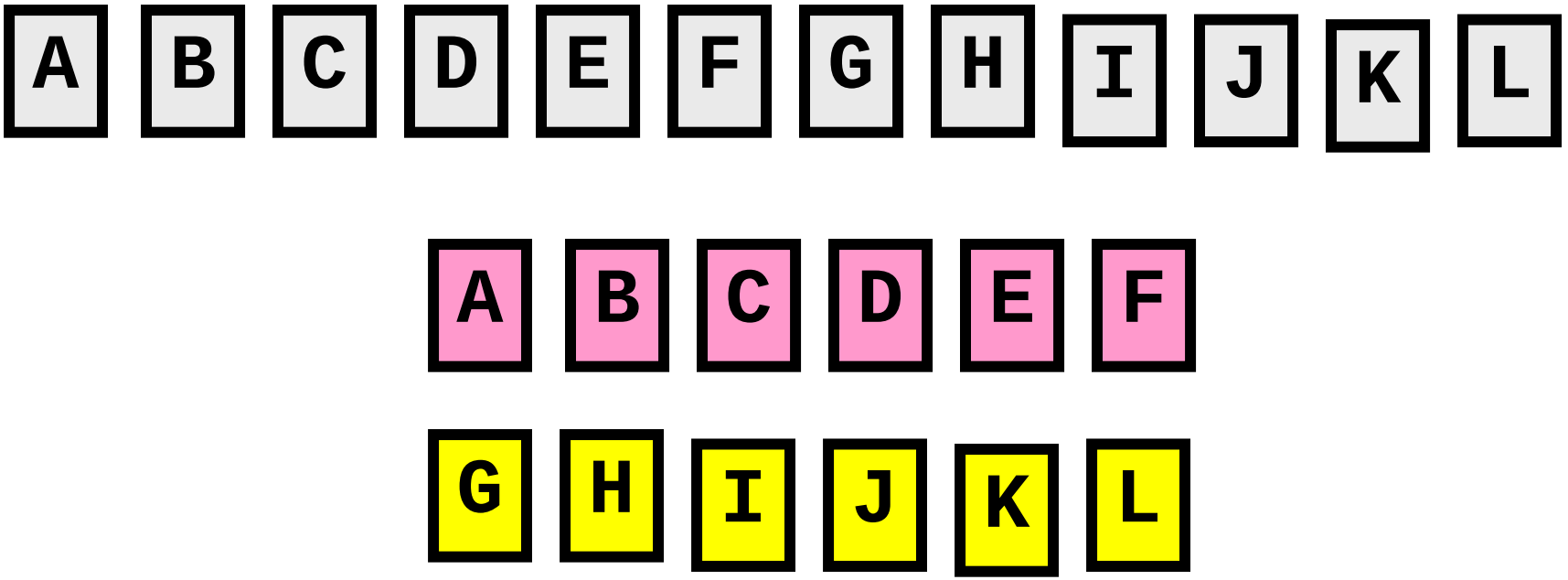
```
% Deal a 52-card deck
N = cell(1,13); E = cell(1,13);
S = cell(1,13); W = cell(1,13);

for k=1:13
    N{k} = D{4*k-3};
    E{k} = D{4*k-2};
    S{k} = D{4*k-1};
    W{k} = D{4*k};
end
```

See function **Deal**

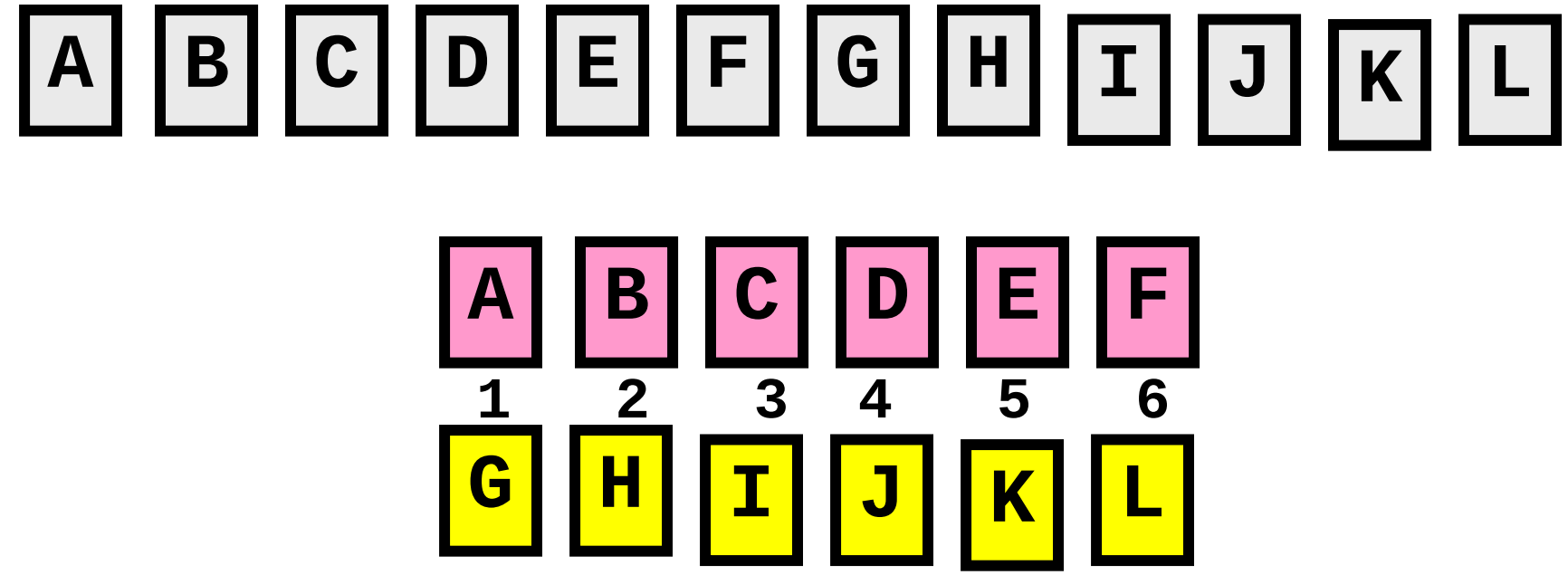
The “perfect shuffle” of a 12-card deck



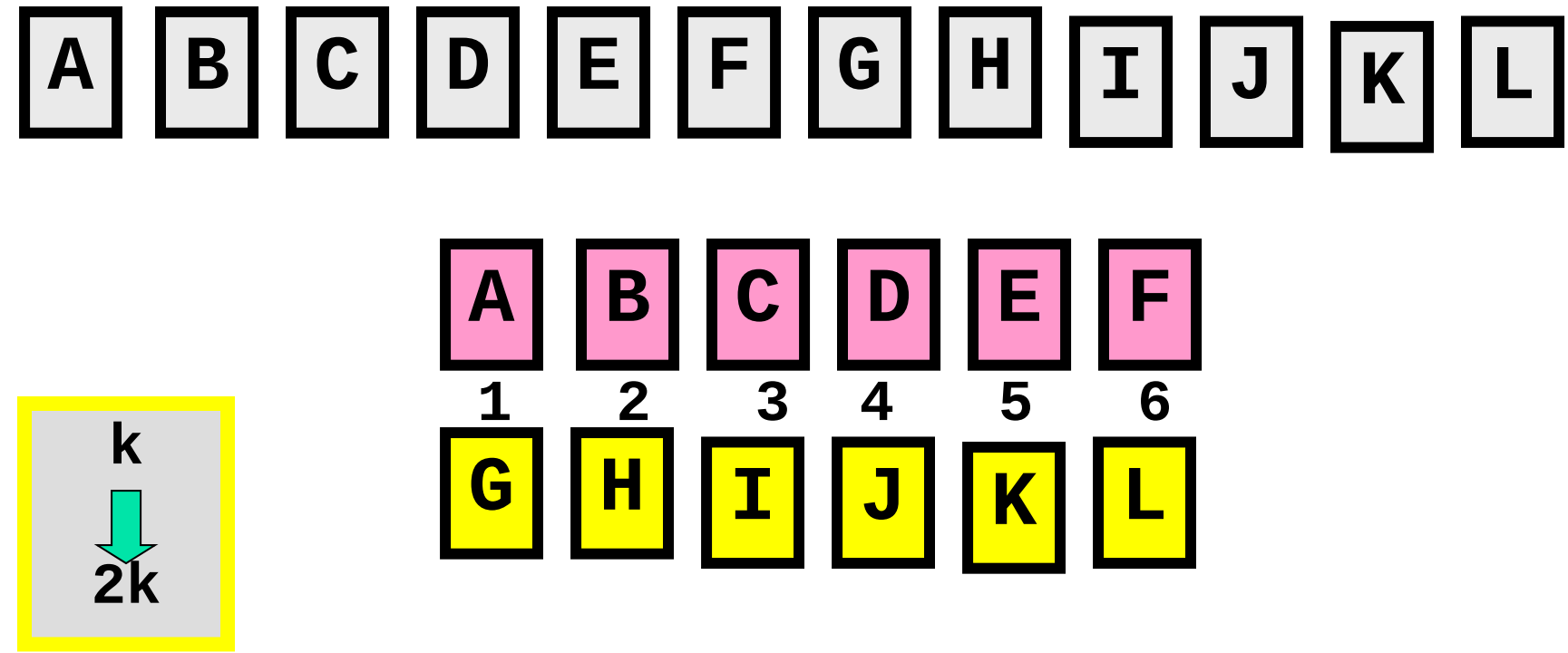
Perfect Shuffle, Step 1: cut the deck



Perfect Shuffle, Step 2: Alternate

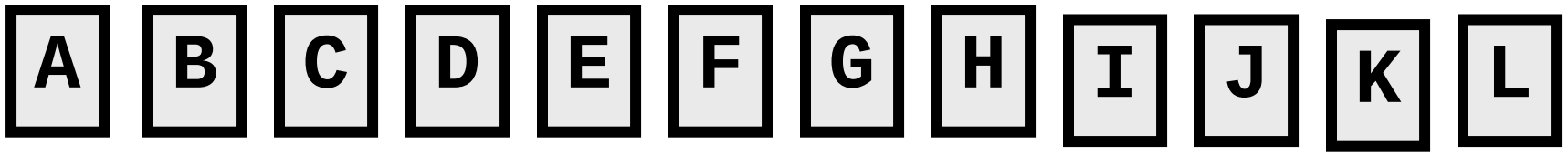


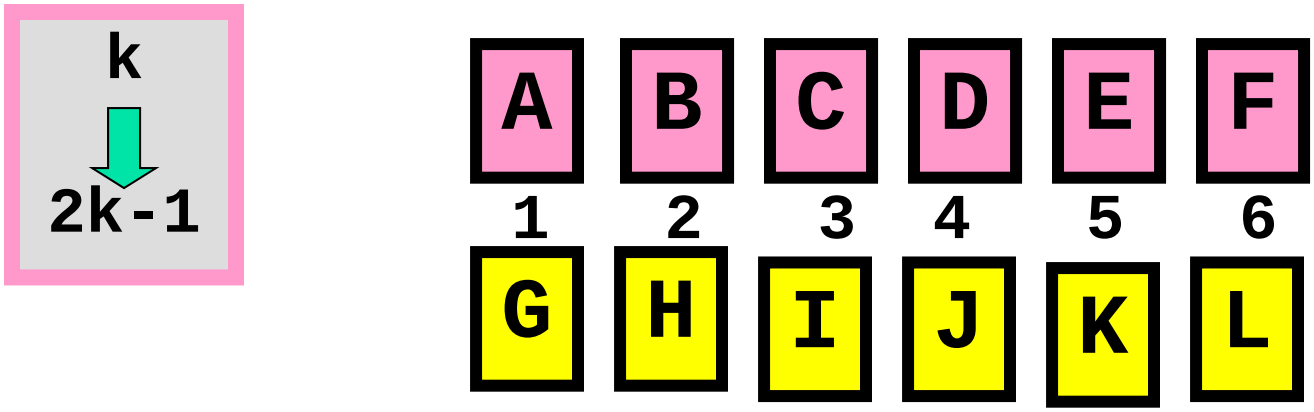
Perfect Shuffle, Step 2: Alternate

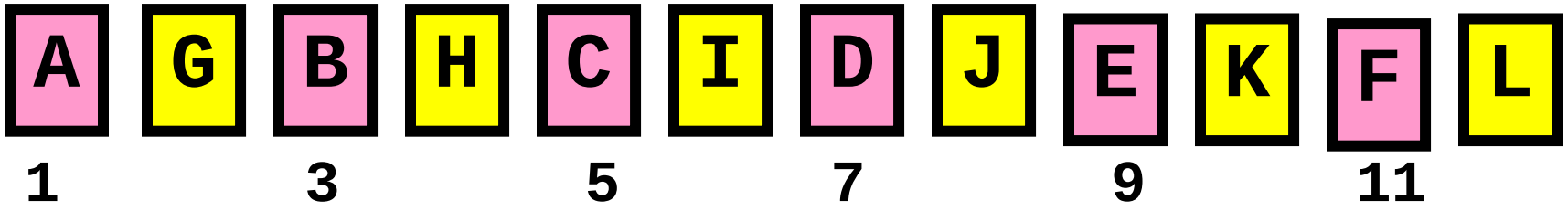


Perfect Shuffle, Step 2: Alternate

See **Shuffle.m**







Example: Build a cell array of Roman numerals for 1 to 3999

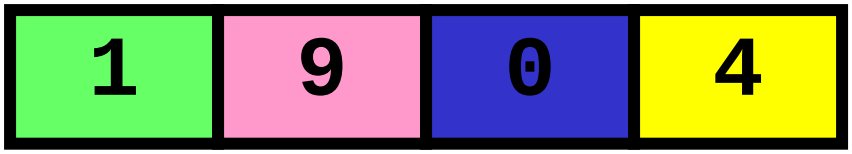
```
C{1} = 'I'
C{2} = 'II'
C{3} = 'III'
   :
C{2007} = 'MMVII'
   :
C{3999} = 'MMMXMXCIX'
```

Example

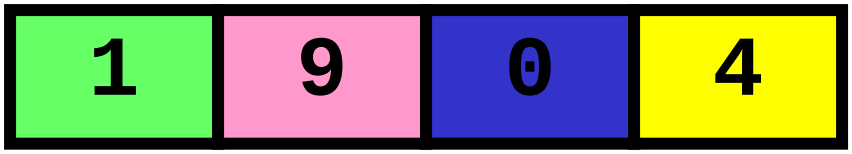
1904 = 1*1000 + 9*100 + 0*10 + 4*1

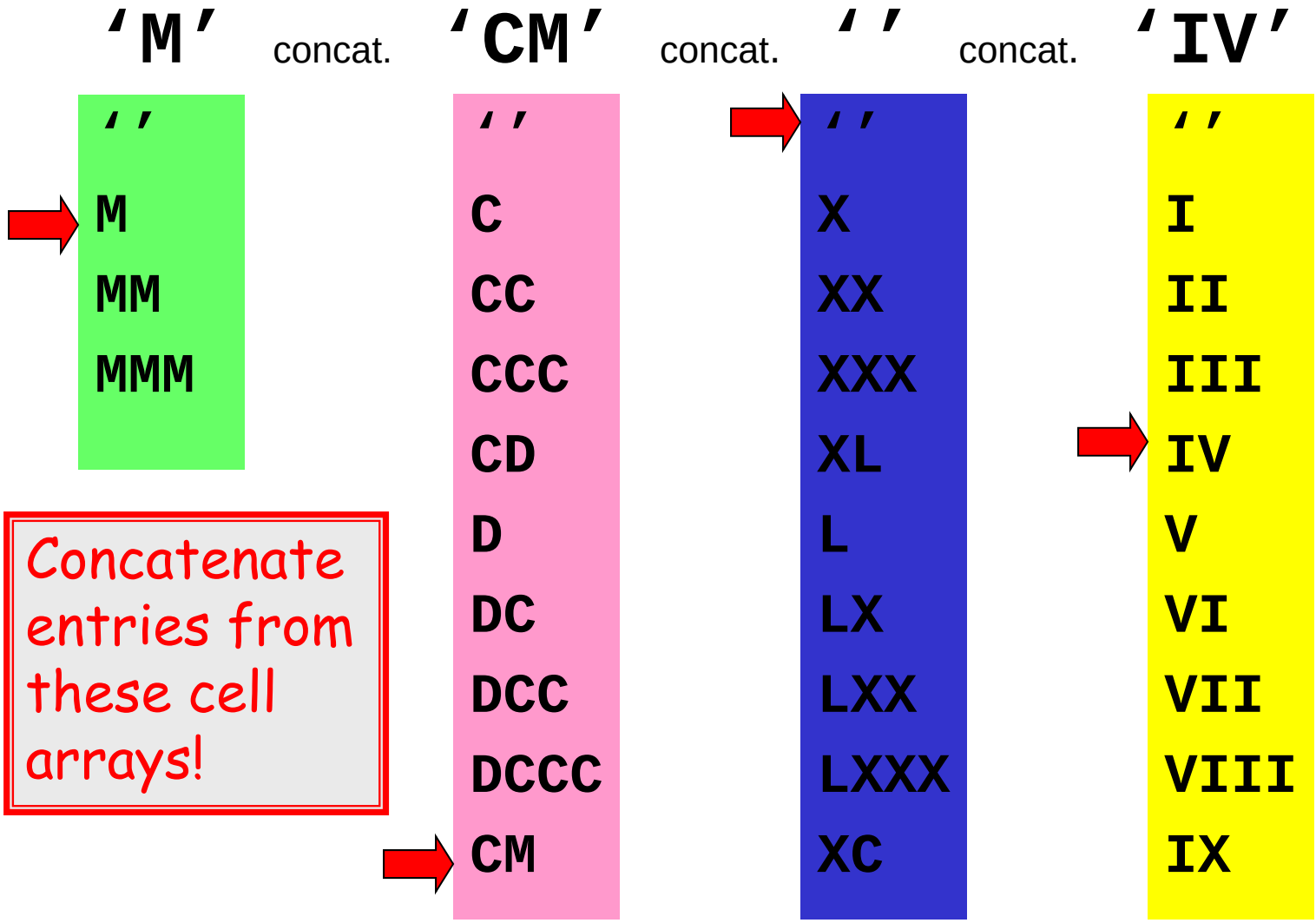
= M CM IV

= MCMIV



MCMIV


1
9
0
4


'M'
concat.
'CM'
concat.
''
concat.
'IV'
''
M
MM
MMM
''
C
CC
CCC
CD
D
DC
DCC
DCCC
CM
''
X
XX
XXX
XL
L
LX
LXX
LXXX
XC
''
I
II
III
IV
V
VI
VII
VIII
IX
Concatenate entries from these cell arrays!

Ones-Place Conversion

```
function r = Ones2R(x)
% x is an integer that satisfies
%       0 <= x <= 9
% r is the Roman numeral with value x.

Ones = {'I', 'II', 'III', 'IV', ...
         'V', 'VI','VII', 'VIII', 'IX'};

if x==0
     r = '';
else
     r = Ones{x};
end
```

Ones-Place Conversion

```
function r = Ones2R(x)
% x is an integer that satisfies
%       0 <= x <= 9
% r is the Roman numeral with value x.

Ones = {'I', 'II', 'III', 'IV', ...
         'V', 'VI','VII', 'VIII', 'IX'};

if x==0
     r = '';
else
     r = Ones{x};
end
```

Similarly, we can implement these functions:

```
function r = Tens2R(x)
% x is an integer that satisfies
%       0 <= x <= 9
% r is the Roman numeral with value 10*x.
```

```
function r = Hund2R(x)
% x is an integer that satisfies
%       0 <= x <= 9
% r is the Roman numeral with value 100*x
```

```
function r = Thou2R(x)
% x is an integer that satisfies
%       0 <= x <=3
% r is the Roman numeral with value 1000*x
```

We want all the Roman Numerals from 1 to 3999. We have the functions Ones2R, Tens2R, Hund2R, Thou2R.

The code to generate all the Roman Numerals will include loops—nested loops. How many are needed?

A: 2 B: 4 C: 6 D: 8

Now we can build the Roman numeral cell array for 1,…,3999

```
for a = 0:3
  for b = 0:9
    for c = 0:9
      for d = 0:9
        n = a*1000 + b*100 + c*10 + d;
        if n>0
          C{n} = [Thou2R(a) Hund2R(b)...
                                Tens2R(c) Ones2R(d)];
        end
      end
    end
  end
end
```

Now we can build the Roman numeral cell array for 1,…,3999

```
for a = 0:3  % possible values in thous place
  for b = 0:9  % values in hundreds place
    for c = 0:9  % values in tens place
      for d = 0:9  % values in ones place
        n = a*1000 + b*100 + c*10 + d;
        if n>0
          C{n} = [Thou2R(a) Hund2R(b)...
                  Tens2R(c) Ones2R(d)];
        end
      end
    end
  end
end
```

Four strings concatenated together

The n^{th} component of cell array C

The reverse conversion problem

Given a Roman Numeral, compute its value.

Assume cell array **`C(3999,1)`** available where:

```
C{1} = 'I'
C{2} = 'II'
   :
C{3999} = 'MMMCMXCIX'
```

```
function k = RN2Int(r)
% r is a string that represents
%       Roman numeral (<=3999)
% k is its value

C = RomanNum();
k = 1;
while ~strcmp(r,C{k})
   k=k+1;
end
```

See **RN2Int.m**

Example: subset of clicker IDs

IDs

```
['d091314'; ...
 'h134d83'; ...
 'h4567s2'; ...
 'fr83209']
```

Find subset that begins with ‘h’

L

```
{'h134d83', ...
 'h4567s2'}
```

```
L= {};
k= 0;
for r=1:size(IDs,1)
  if IDs(r,1)==‘h’
    k= k+1;
    L{k }= IDs(r,:);
  end
end
```

Directly assign into a particular cell—good!

```
L= {};

for r=1:size(ID,1)
  if IDs(r,1)==‘h’

    L= [L, IDs(r,:)];
  end
end
```

Concatenate cells or cell arrays—prone to problems!

A detailed sort-a-file example

Suppose each line in the file

`statePop.txt`

is structured as follows:

Cols 1-14: State name

Cols 16-24: Population (millions)

The states appear in alphabetical order.

Alabama	4557808
Alaska	663661
Arizona	5939292
Arkansas	2779154
California	36132147
Colorado	4665177
:	:
:	:
Texas	22859968
Utah	2469585
Vermont	623050
Virginia	7567465
Washington	6287759
West Virginia	1816856
Wisconsin	5536201
Wyoming	509294

A detailed sort-a-file example

Create a new file

`statePopSm2Lg.txt`

that is structured the same as **`statePop.txt`**

except that *the states are ordered from smallest to largest according to population.*

```
Alabama              4557808
Alaska                663661
Arizona              5939292
Arkansas             2779154
California          36132147
Colorado             4665177
   :                    :
   :                    :
```

- Need the pop as *numbers* for sorting.
- Can't just sort the pop—have to maintain association with the state names.

First, get the populations into an array

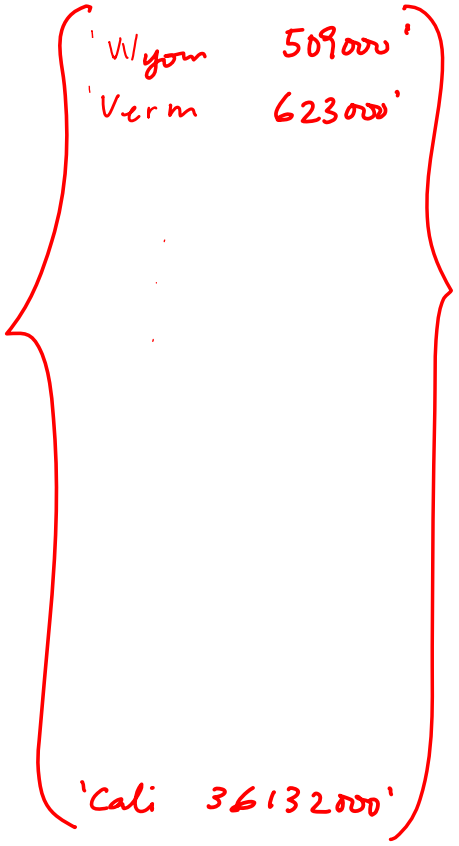
```
C = file2cellArray('StatePop');
n = length(C);
pop = zeros(n,1);
for i=1:n
  S = C{i};
  pop(i) = str2double(S(16:24));
end
```

Converts a string representing a numeric value (digits, decimal point, spaces) to the numeric value → scalar of type double. E.g., `x=str2double(' 3.24  ')` assigns to variable `x` the numeric value 3.24

C

cell array
of strings
in alpha. order

Cnew



C

{ 'Alab 4558000'
'Alas 664000'
⋮
'Cali 36132000'
'Verm 623000'
⋮
'Wyom 509000' }

cell array
of strings
in alpha. order

Pop

[4558000
664000
⋮
36132000
⋮
623000
⋮
509000]

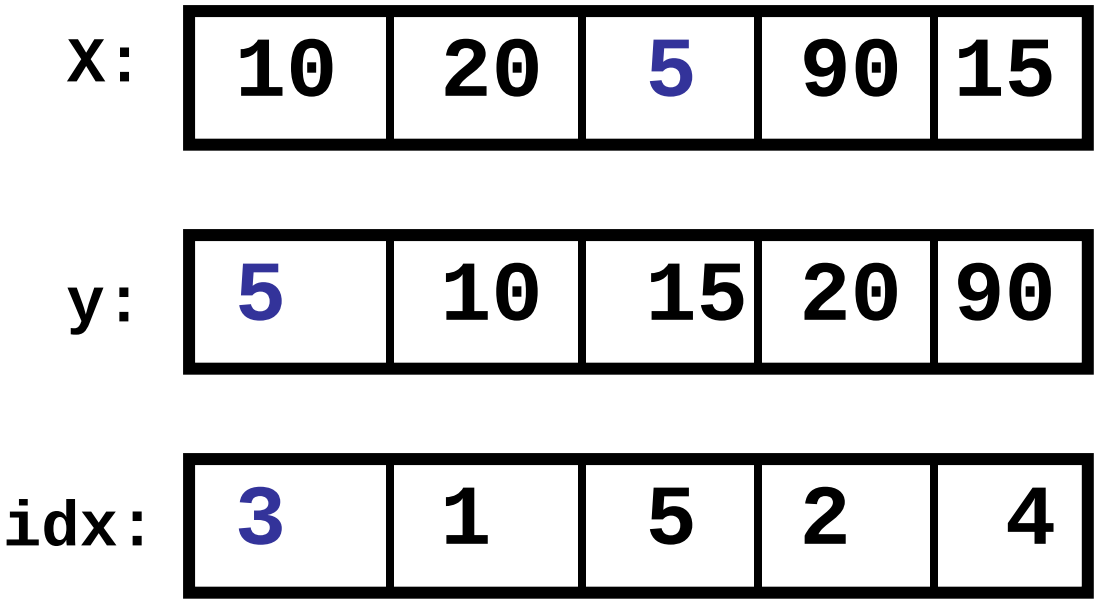
vector
of numbers

Crew

{ 'Wyom 509000'
'Verm 623000'
⋮
'Cali 36132000' }

Built-In function `sort`

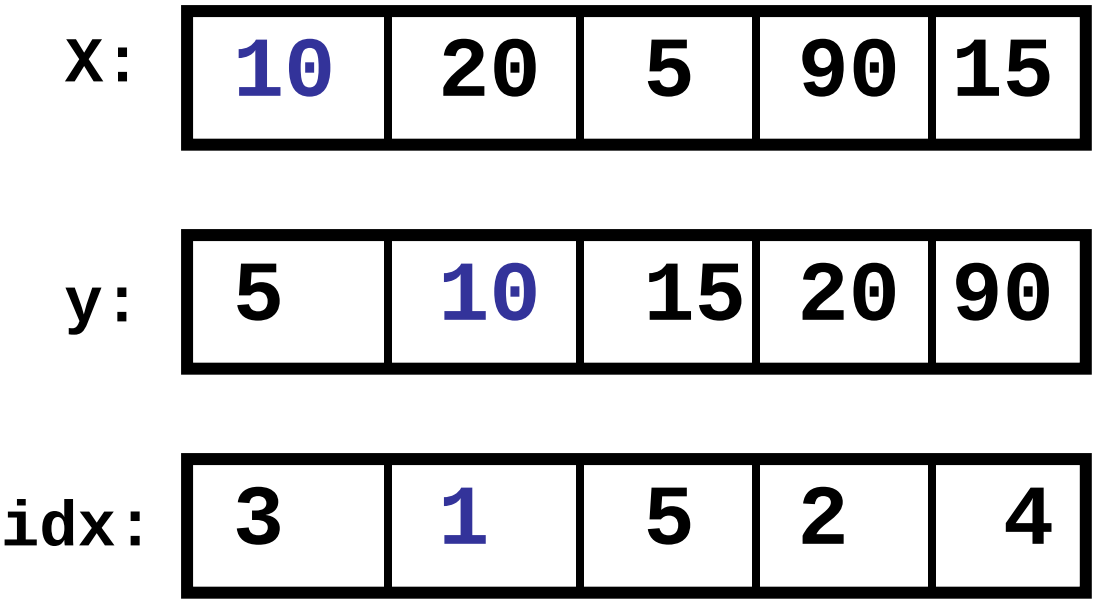
Syntax: `[y,idx] = sort(x)`




```
y(1) = x(3) = x(idx(1))
```

Built-In function `sort`

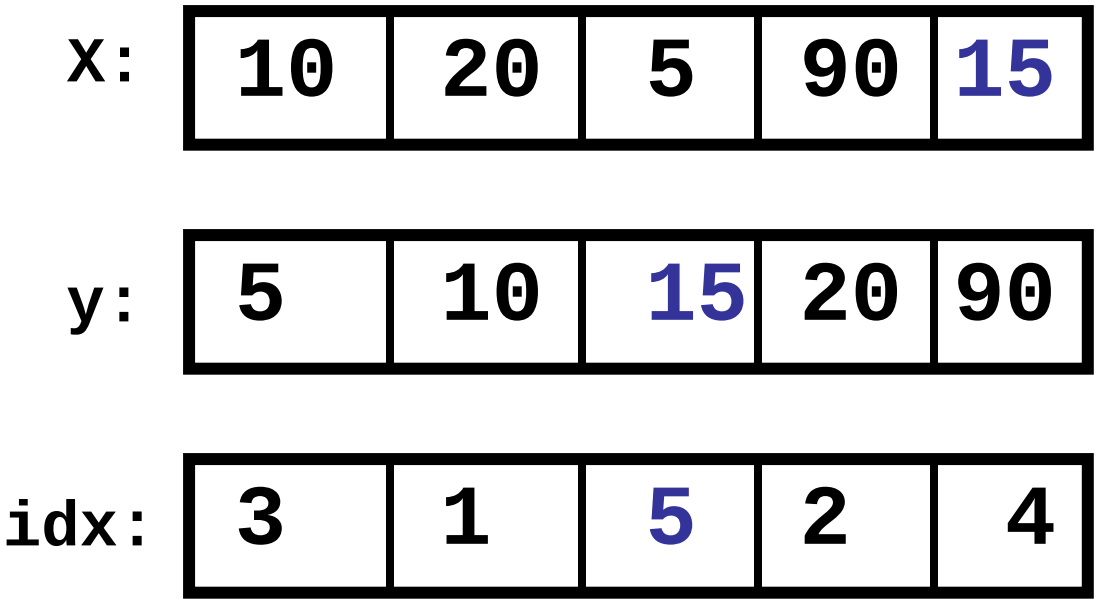
Syntax: `[y,idx] = sort(x)`




```
y(2) = x(1) = x(idx(2))
```

Built-In function `sort`

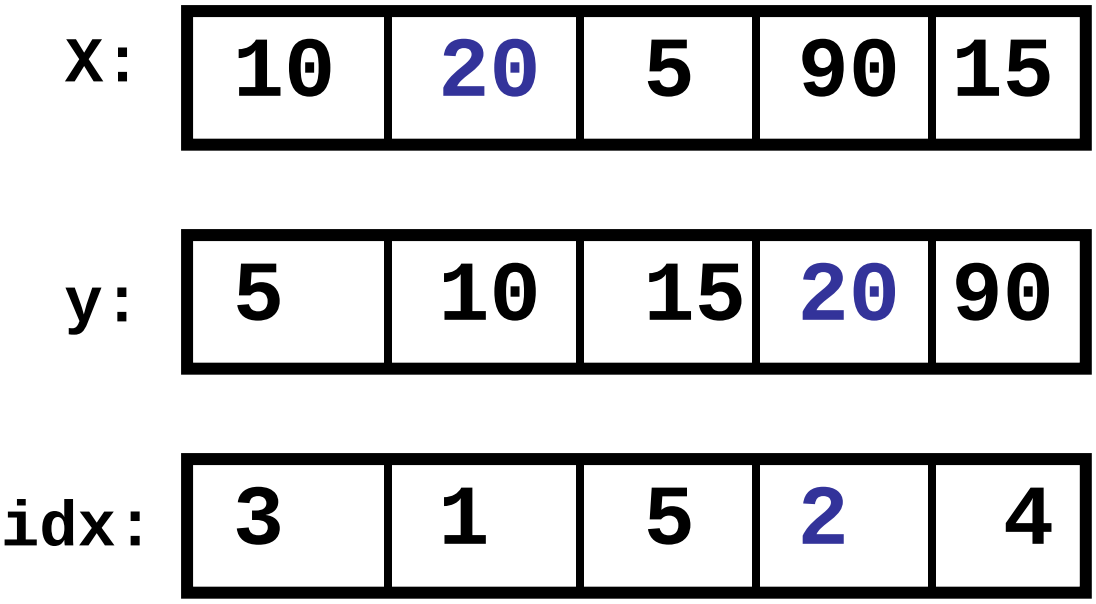
Syntax: `[y,idx] = sort(x)`




```
y(3) = x(5) = x(idx(3))
```

Built-In function `sort`

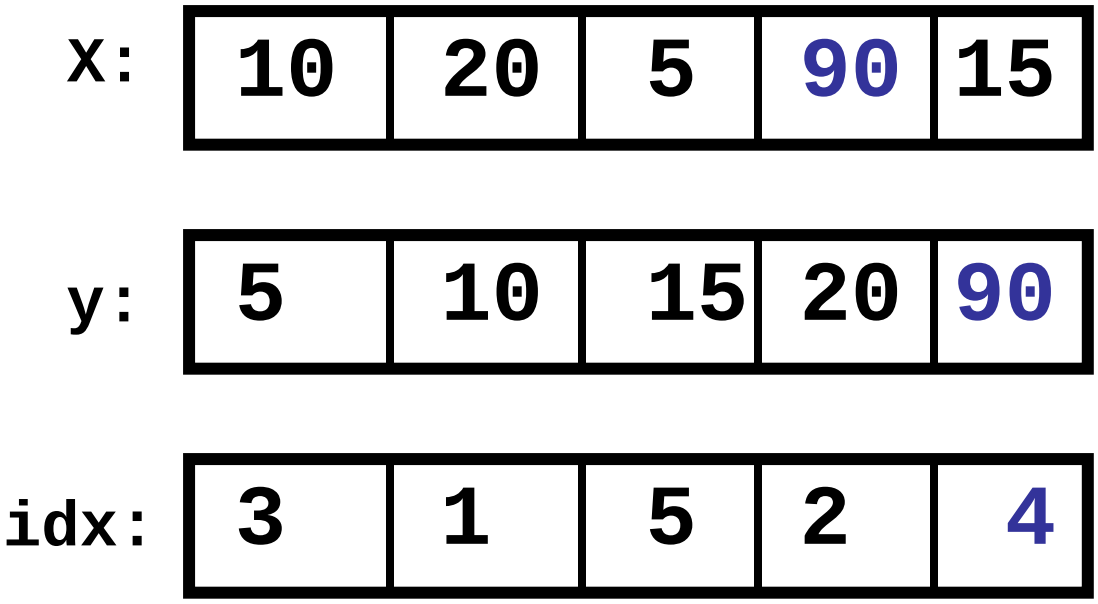
Syntax: `[y,idx] = sort(x)`



`y(4) = x(2) = x(idx(4))`

Built-In function `sort`

Syntax: `[y,idx] = sort(x)`



`y(5) = x(4) = x(idx(5))`

Built-In function `sort`

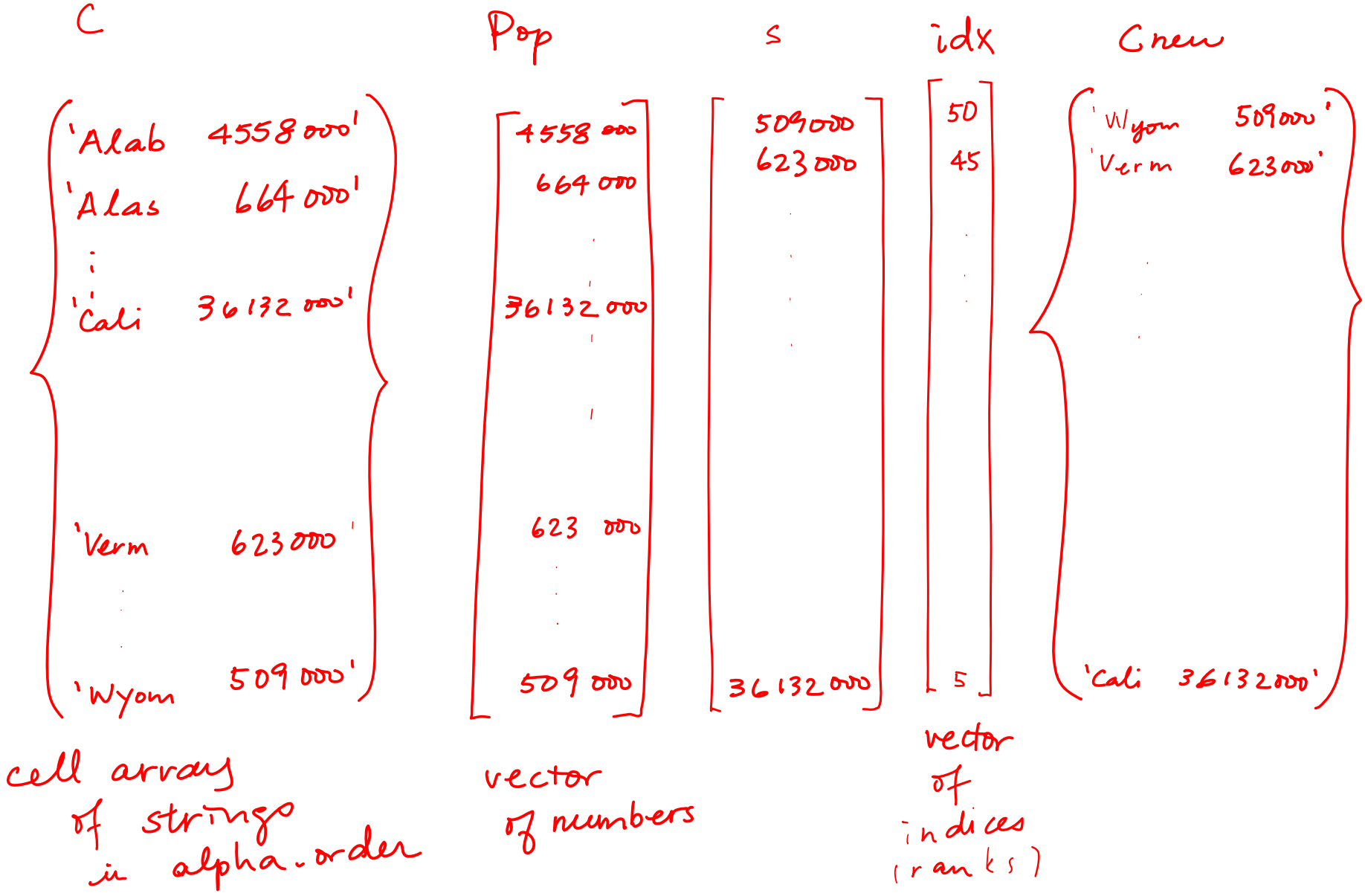
Syntax: `[y,idx] = sort(x)`

x:	10	20	5	90	15

y:	5	10	15	20	90

idx:	3	1	5	2	4

```
y(k) = x(idx(k))
```


C
Pop
s
idx
Cnew
'Alab 4558000'
'Alas 664000'
'Cali 36132000'
'Verm 623000'
'Wyom 509000'
4558000
664000
36132000
623000
509000
509000
623000
36132000
50
45
5
'Wyom 509000'
'Verm 623000'
'Cali 36132000'
cell array of strings in alpha. order
vector of numbers
vector of indices (ranks)

Sort from little to big

```
% C is cell array read from statePop.txt
% pop is vector of state pop (numbers)
[s,idx] = sort(pop);
Cnew = cell(n,1);
for i=1:length(C)
    ithSmallest = idx(i);
    Cnew{i} = C{ithSmallest};
end

cellArray2file(Cnew,'statePopSm2Lg')
```

```
Wyoming            509294
Vermont            623050
North Dakota       636677
Alaska             663661
South Dakota       775933
Delaware           843524
Montana            935670
   :                  :
   :                  :
Illinois         12763371
Florida          17789864
New York         19254630
Texas            22859968
California       36132147
```