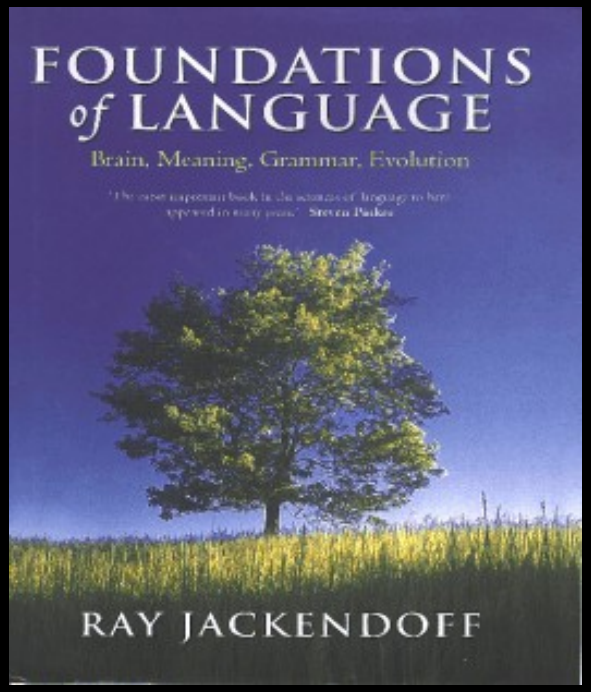
CSCI 252: Neural Networks and Graphical Models

Prof. Levy

Architecture #6a: Tensor Products

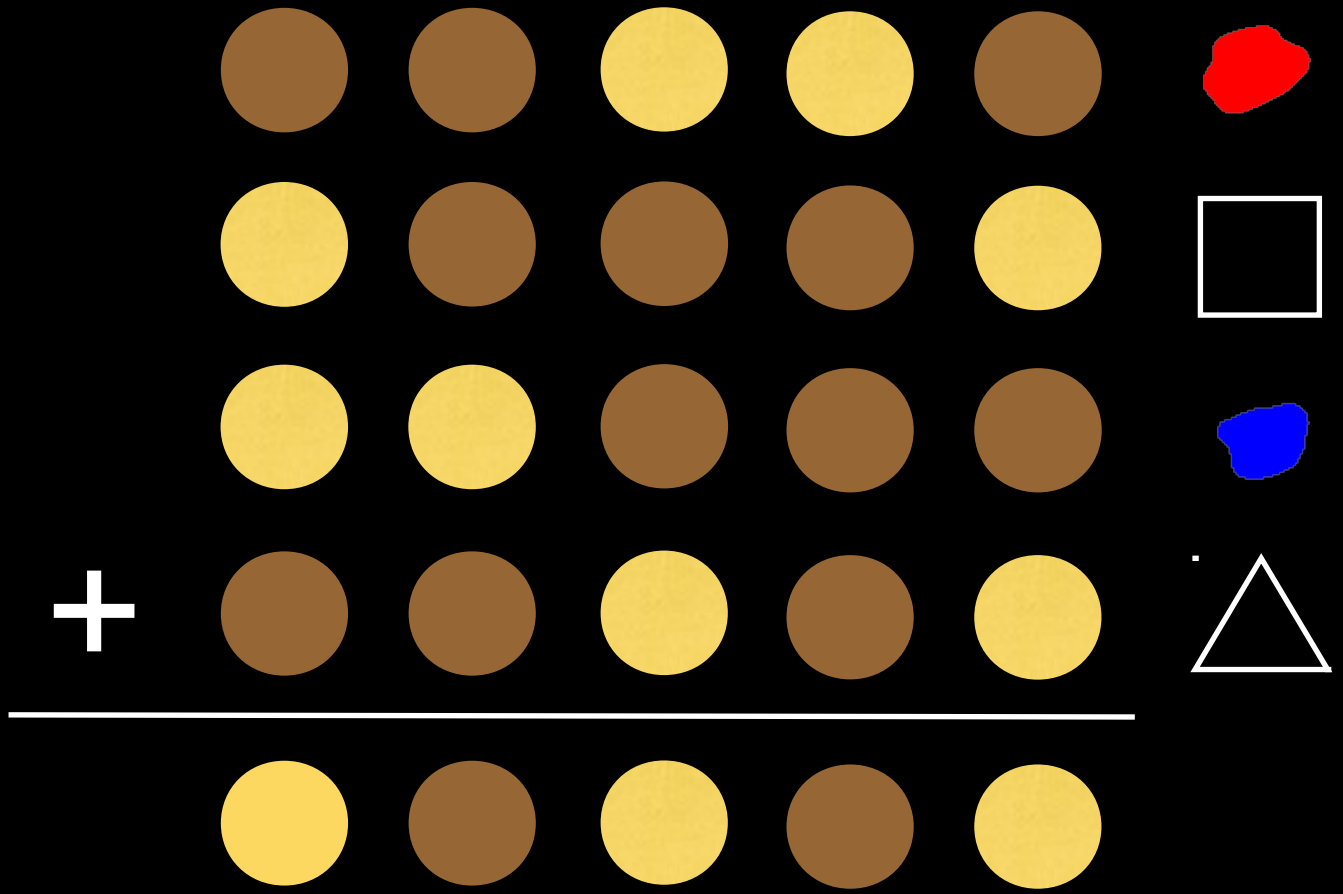
Limitations of LSA

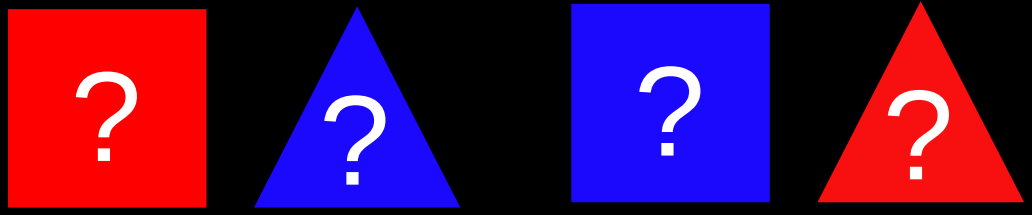
- LSA and other bag of words techniques are great at revealing hidden (latent) associations among words.
- But a bag of words is not a sentence: *Dogs chase cats* does not mean the same thing as *Cats chase dogs.*
- More generally, we would like an architecture that supports data structures (e.g., binary trees, parse trees) while also giving us the advantages of vectors: graceful degradation, content addressability, etc.



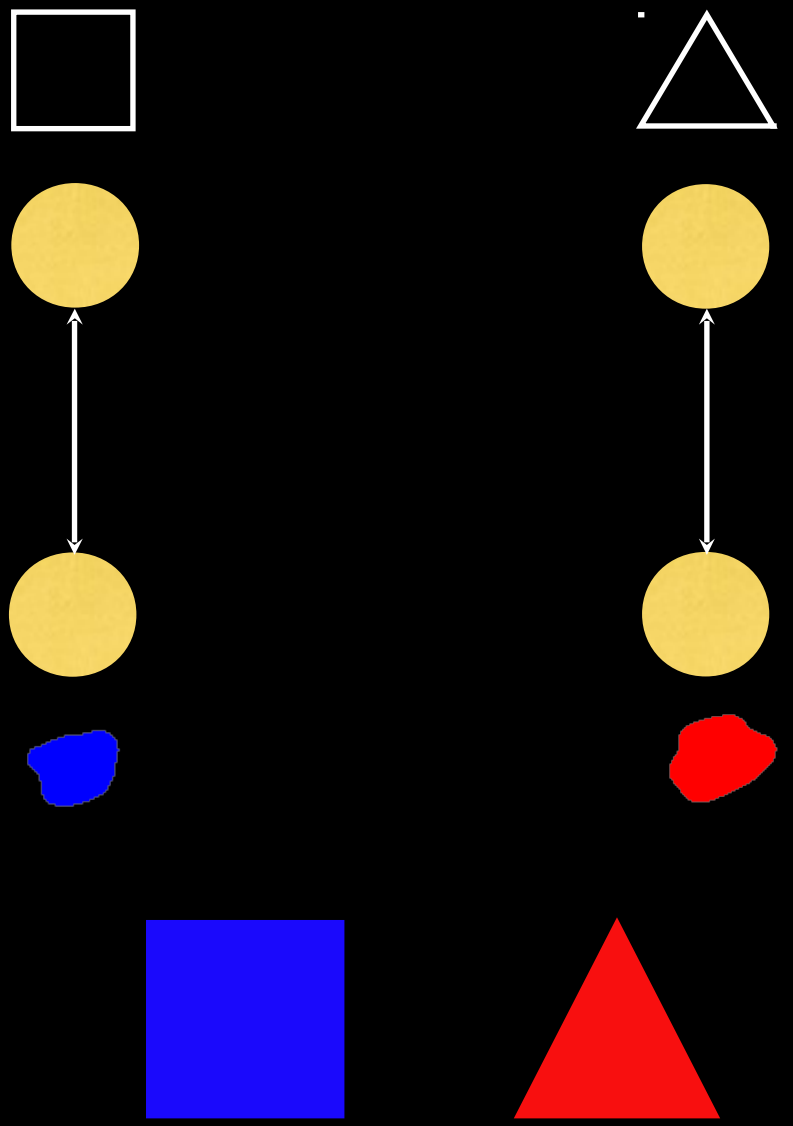
Language Isn't (Just) Association: Jackendoff's Four Challenges for Cognitive Neuroscience

Challenge I: The Binding Problem

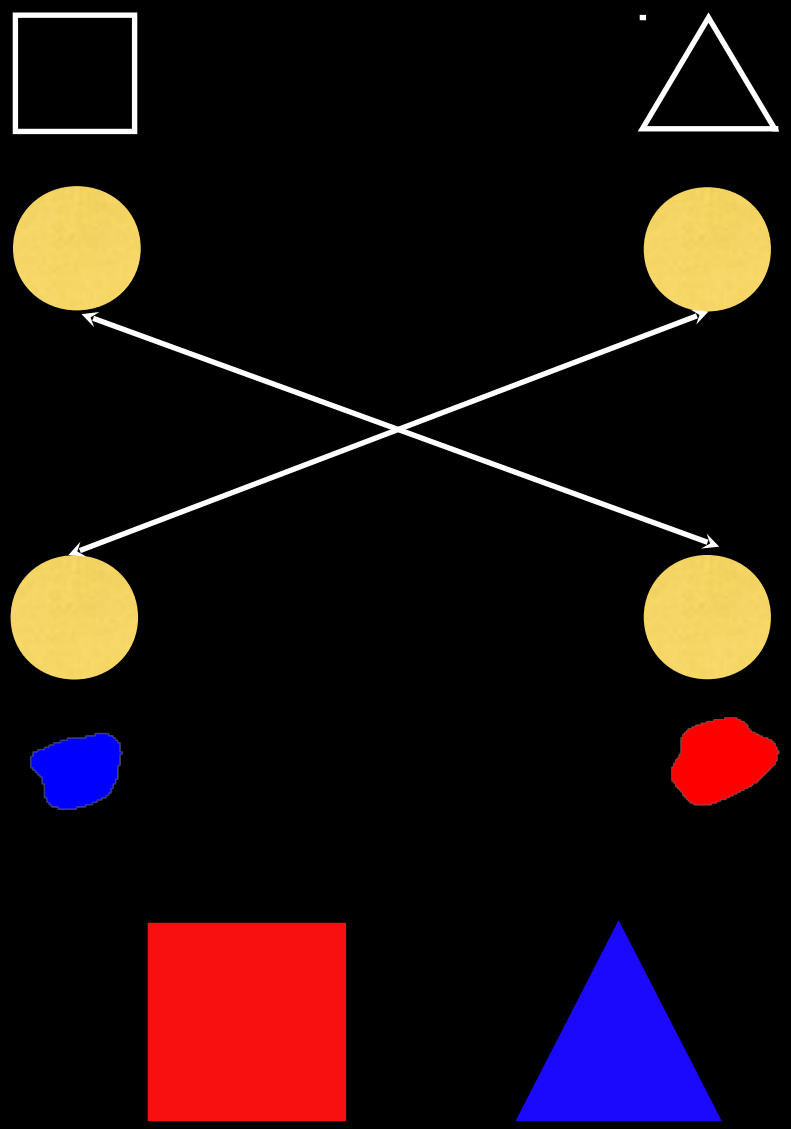




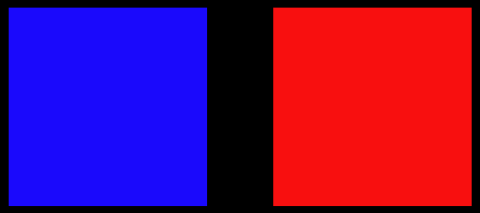
Solution? Local ("Grandmother Cell") Representations

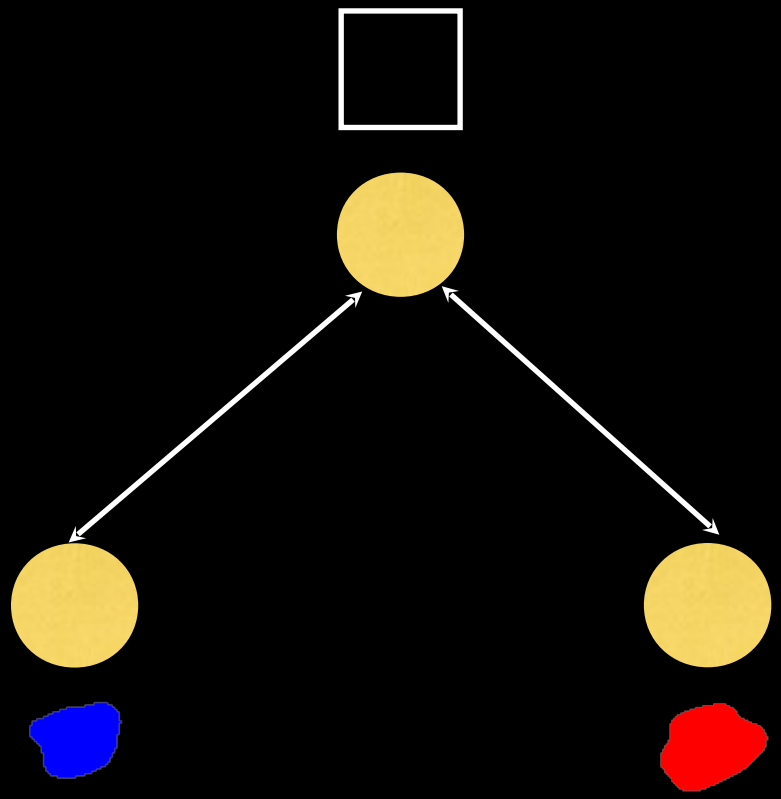


Local Representations



Challenge II:
The "Problem of Two"

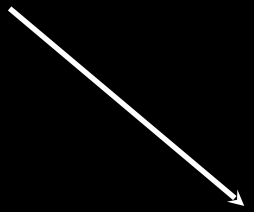




Challenge III:
The Problem of Variables

(Challenge IV: Working Memory vs. Long-Term Memory)

ignores(*X*, *Y*)

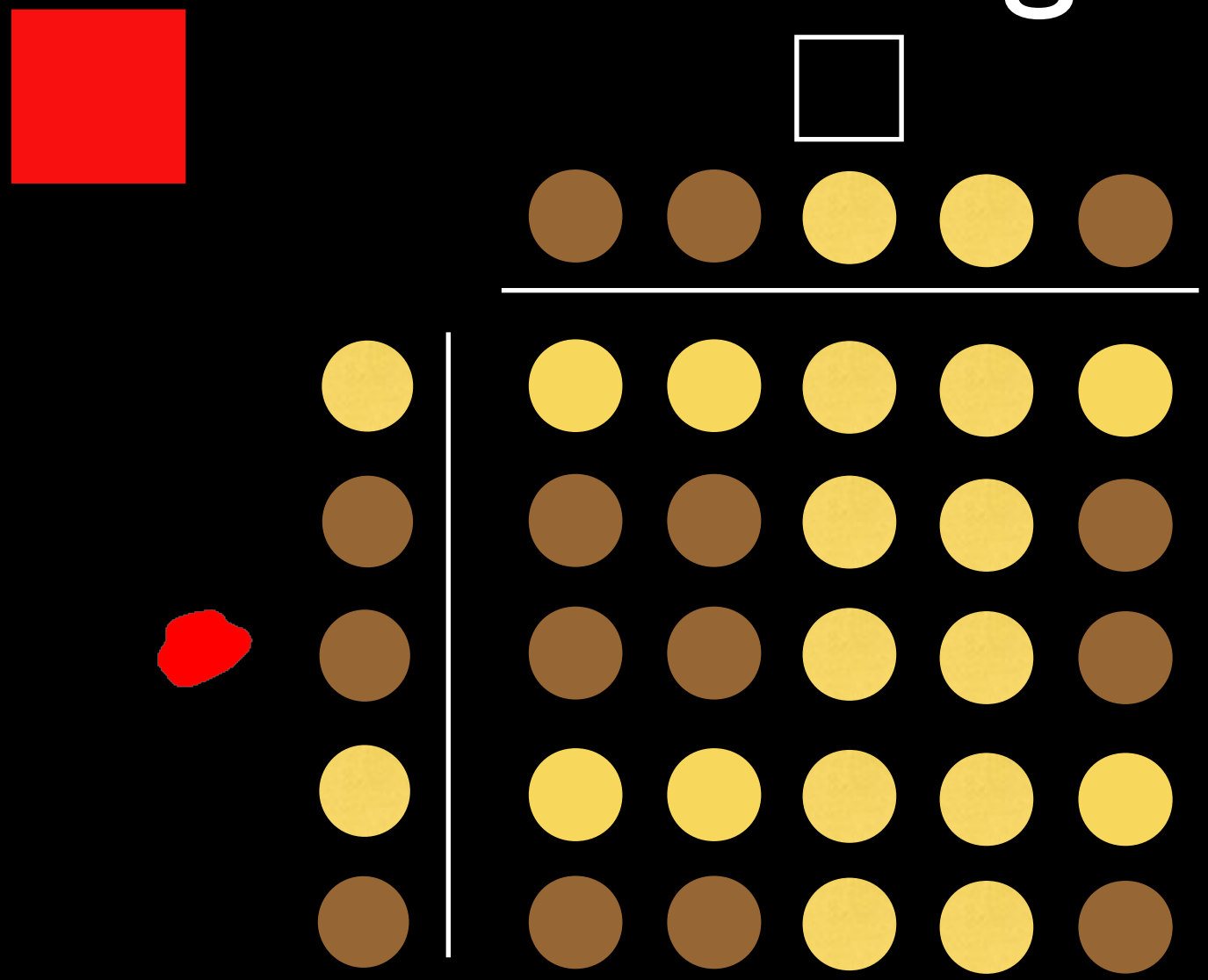


X won't give *Y* the time of day.

Tensor Products (Smolensky 1990)

- Tensor: a generalization of the scalar → vector → matrix concept: what we CS people would call an array.
- Smolenksy proposed tensor product (outer product) as a solution to the binding problem. Because the outer product of two vectors pairs every item from each vector with every other, information is preserved in a way that is not with simple vector addition.

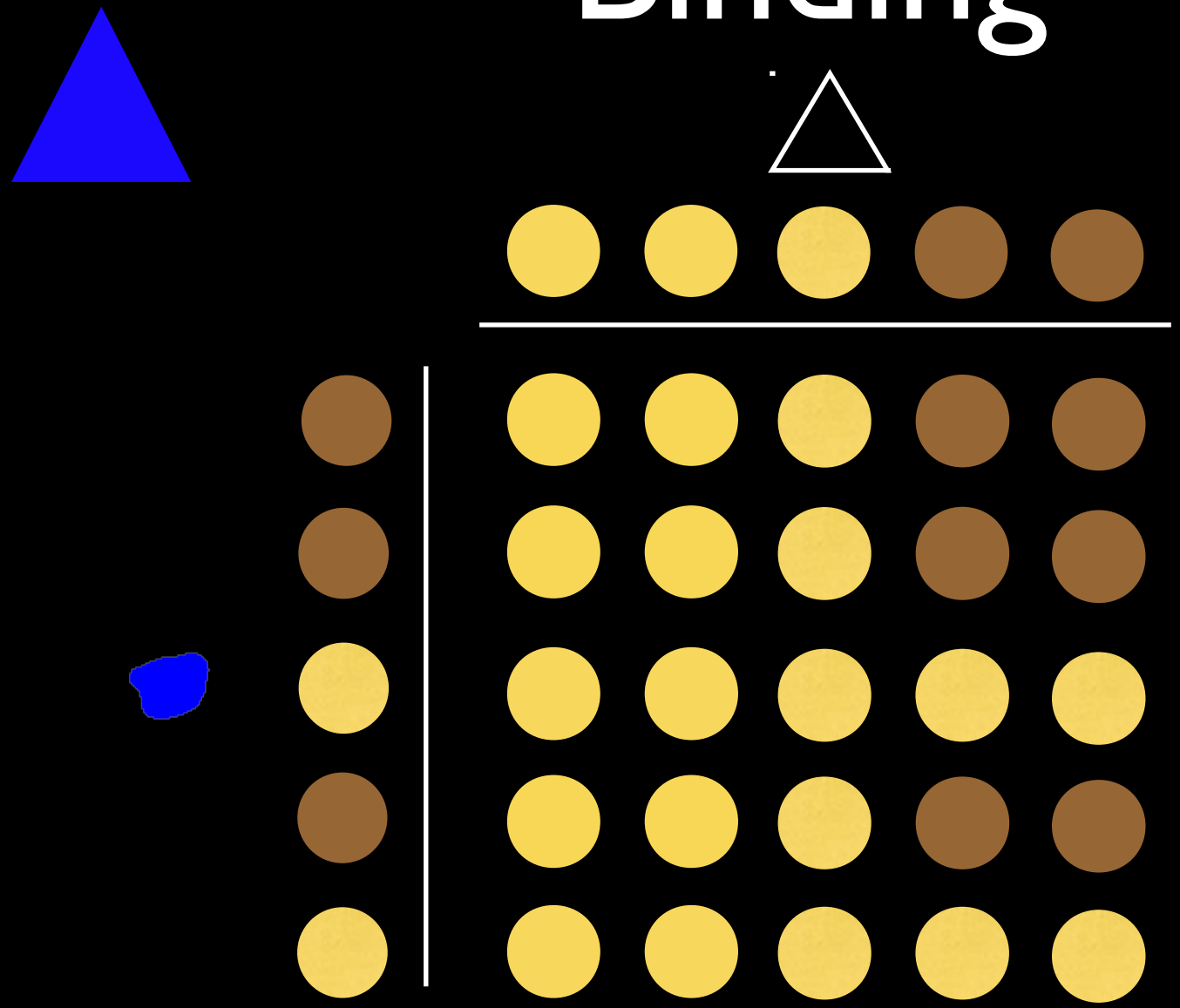
Binding



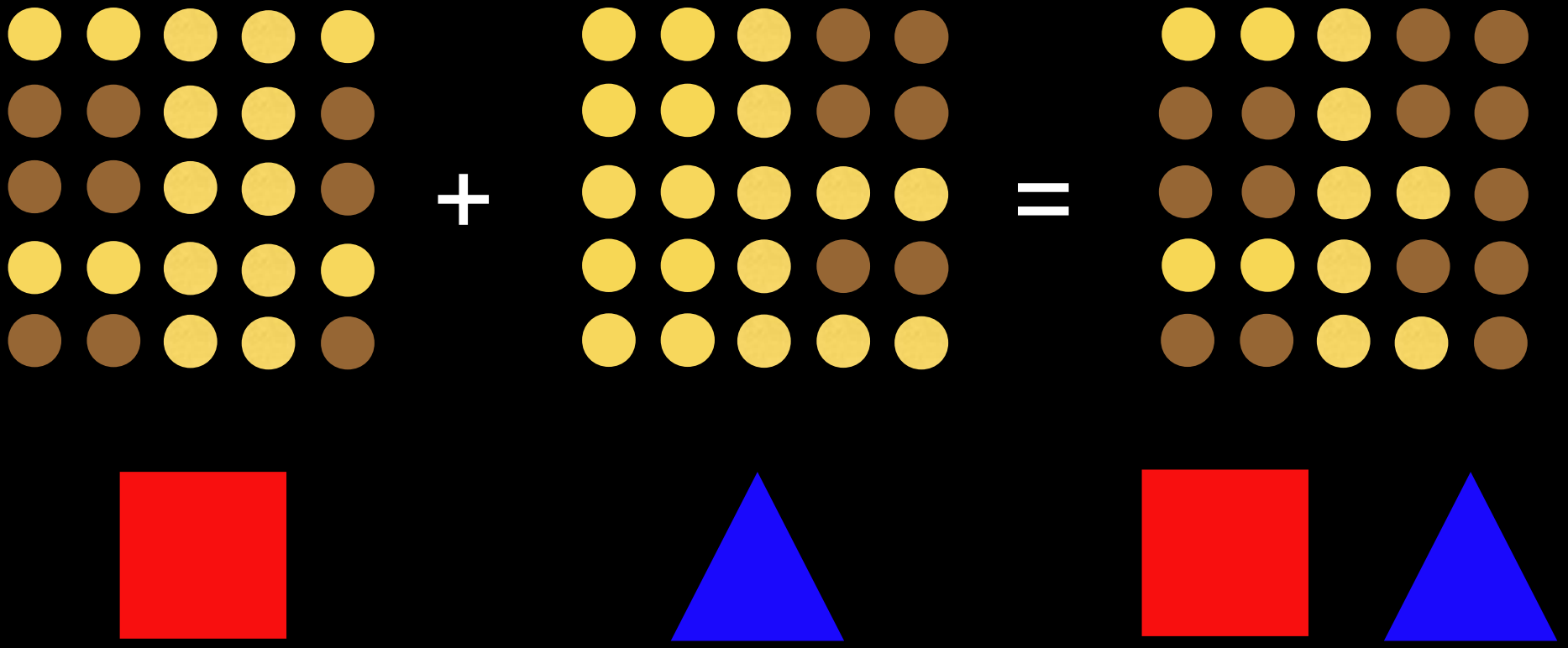
Binding

```
>>> red = np.random.random(5)
>>> square = np.random.random(5)
>>> np.outer(red, square)
array([[ 0.11676181,  0.02064508,  0.02289996,  0.15695528,  0.09769976],
       [ 0.13316499,  0.02354539,  0.02611704,  0.17900502,  0.11142503],
       [ 0.55397617,  0.09795055,  0.10864881,  0.74467403,  0.46353631],
       [ 0.11748984,  0.02077381,  0.02304275,  0.15793393,  0.09830893],
       [ 0.27806933,  0.04916646,  0.05453647,  0.37379046,  0.23267287]])
```

Binding



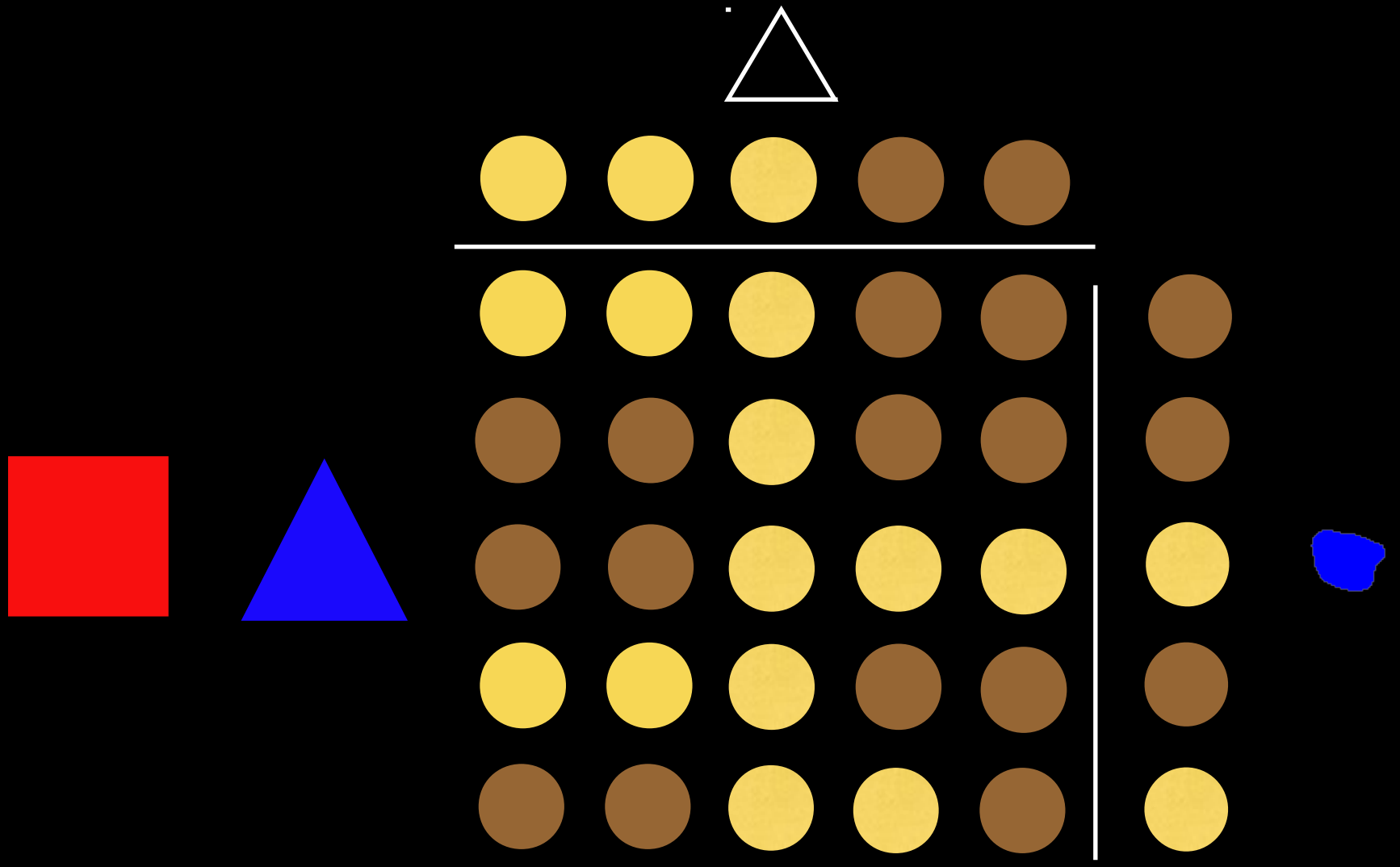
Bundling



Bundling

```
>>> blue = np.random.random(5)
>>> triangle = np.random.random(5)
>>> np.outer(red, square) + np.outer(blue, triangle)
array([[ 0.63963985,  0.5493258 ,  0.1712245 ,  0.24450324,  0.68352538],
       [ 0.89809784,  0.79696713,  0.24310515,  0.30708136,  0.96844564],
       [ 0.77000966,  0.31638148,  0.16993092,  0.78084555,  0.70557736],
       [ 0.37319109,  0.27931272,  0.09557739,  0.2007472 ,  0.38479322],
       [ 0.46505919,  0.23823146,  0.10757979,  0.40509906,  0.44217384]])
```

Unbinding (query)



Unbinding

```
>>> rs = np.outer(red, square)
>>> red
array([ 0.17489043,  0.19945976,  0.82976729,  0.1759809 ,  0.41650317])
>>> rs / square
array([[ 0.17489043,  0.17489043,  0.17489043,  0.17489043,  0.17489043],
       [ 0.19945976,  0.19945976,  0.19945976,  0.19945976,  0.19945976],
       [ 0.82976729,  0.82976729,  0.82976729,  0.82976729,  0.82976729],
       [ 0.1759809 ,  0.1759809 ,  0.1759809 ,  0.1759809 ,  0.1759809 ],
       [ 0.41650317,  0.41650317,  0.41650317,  0.41650317,  0.41650317]])
>>> np.diagonal(rs / square)
array([ 0.17489043,  0.19945976,  0.82976729,  0.1759809 ,  0.41650317])
```

Unbinding is Noisy

```
>>> rsbt = np.outer(red, square) + np.outer(blue, triangle)
>>> np.diagonal(rsbt / square)
array([ 0.95807771,  6.75133754,  1.29778793,  0.22368641,  0.79152676])
>>> red
array([ 0.17489043,  0.19945976,  0.82976729,  0.1759809 ,  0.41650317])
>>>
```

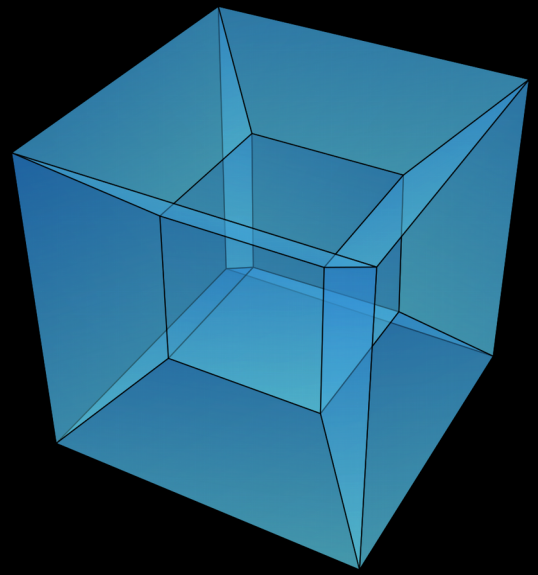
How can we do better?

```
>>> red      = np.random.randint(0, 2, 1000) * 2 - 1 # Use only -1,+1 values
>>> square   = np.random.randint(0, 2, 1000) * 2 - 1
>>> blue     = np.random.randint(0, 2, 1000) * 2 - 1
>>> triangle = np.random.randint(0, 2, 1000) * 2 - 1
>>>
>>> rsbt = np.outer(red, square) + np.outer(blue, triangle)
>>>
>>> query = np.diagonal(rsbt / square)
>>>
>>> cosine(query, red)
0.70427267446636044
>>> cosine(query, square)
0.0454369467397652
>>> cosine(query, blue)
0.0056796183424706491
>>> cosine(query, triangle)
0.028398091712353243
```

Does anyone see a problem with this?

Does anyone see a problem with this?

- Binding two *n*-element vectors gives us a matrix of n^2 elements.
- Binding this matrix with another vector gives us a cube of n^3 elements.
- Binding the cube with another vector gives us a hypercube of n^4 elements.
- Etc.



- Next time we will see how to tame this curse of dimensionality.