Iterating and Indexing Design Space Permutations

Nathan Sturtevant
University of Denver
@nathansttt

Combinations and Permutations

Many design problems can be modeled as combinations and permutations.

Combinations and Permutations

These can often be solved by pre-computing data about the problem.

Combinations and Permutations

A perfect hash function allows an efficient hash table.

Combinations and Permutations

Perfect hash functions for combinations and permutations

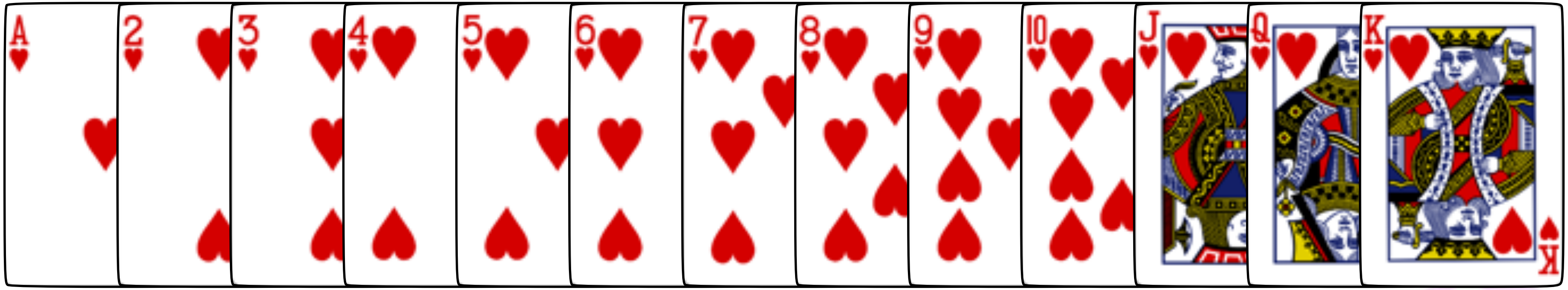
Structure

- Combinations and Permutations
  - Example Problems
  - How to Count
  - Ranking
  - Unranking
  - Application

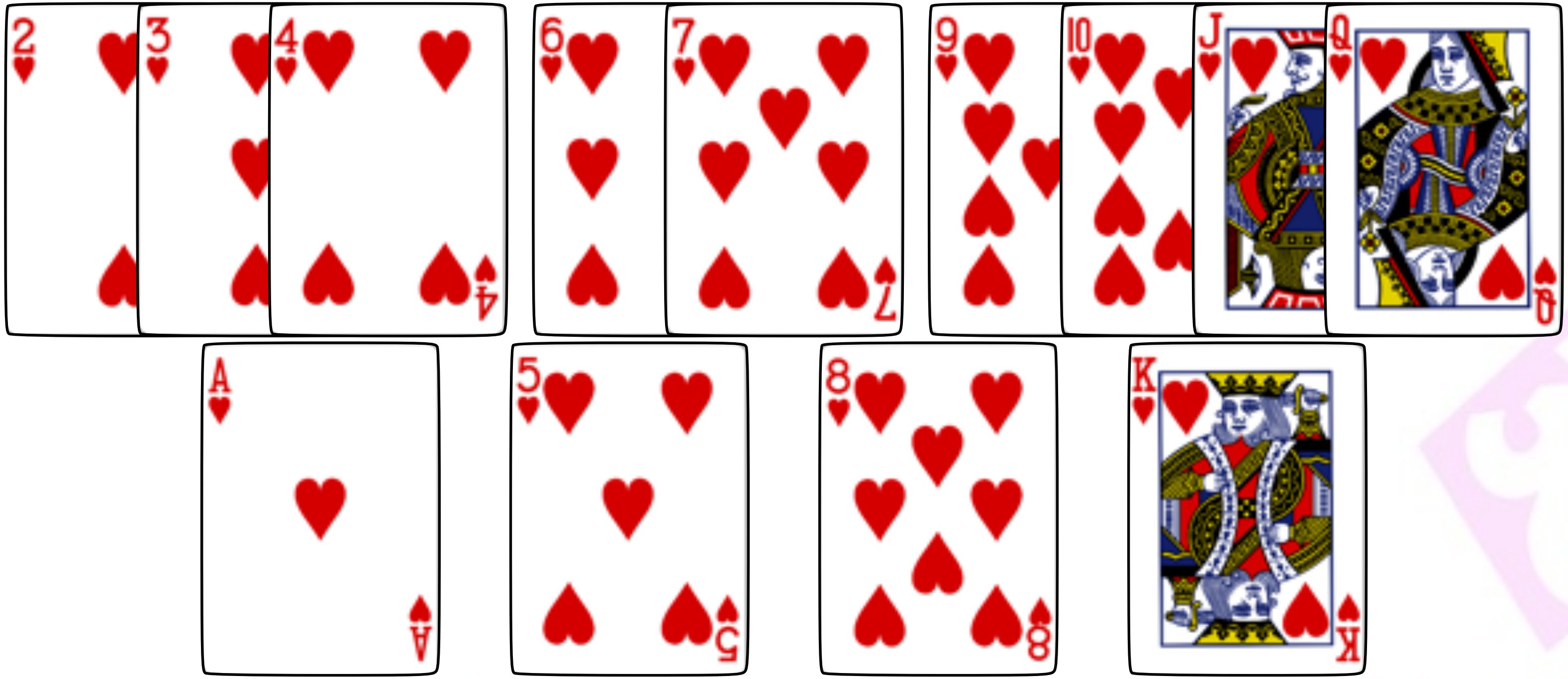
Combinations

Combination: Selecting hands

Combination: Selecting hands

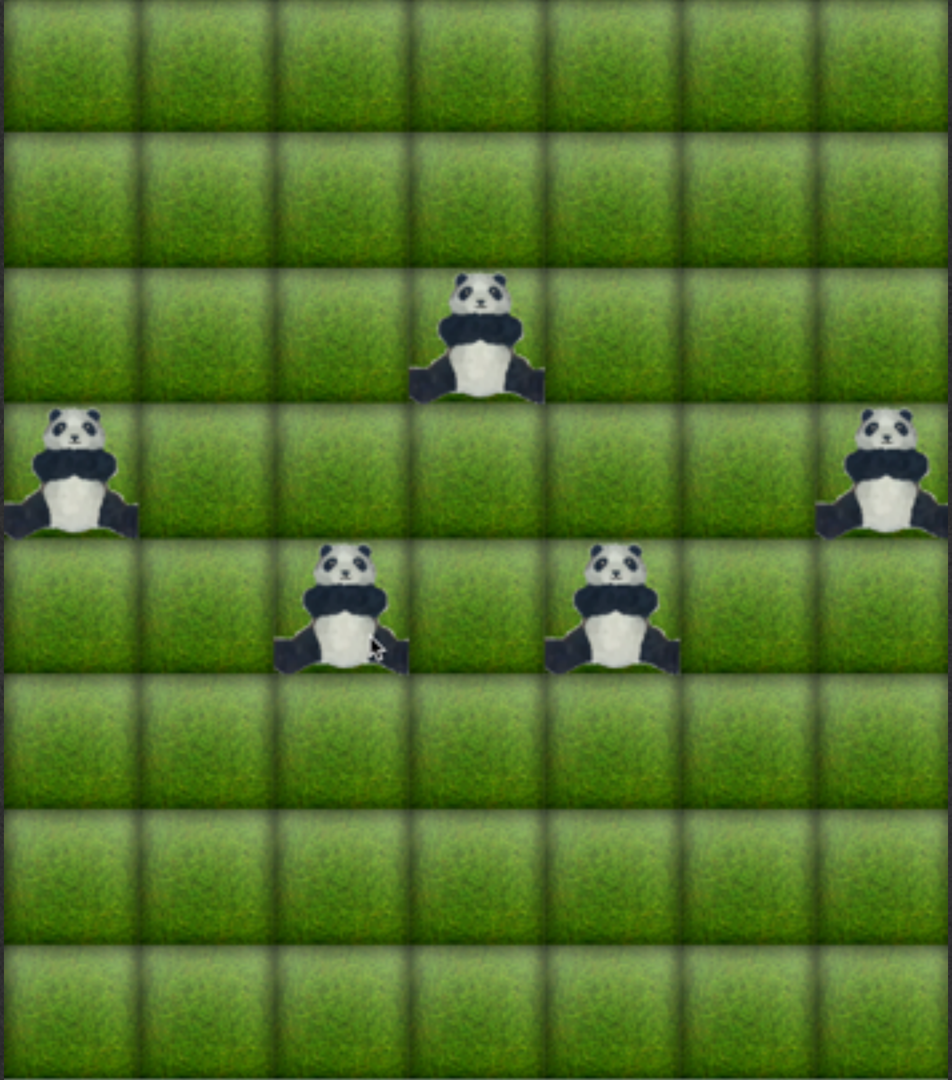


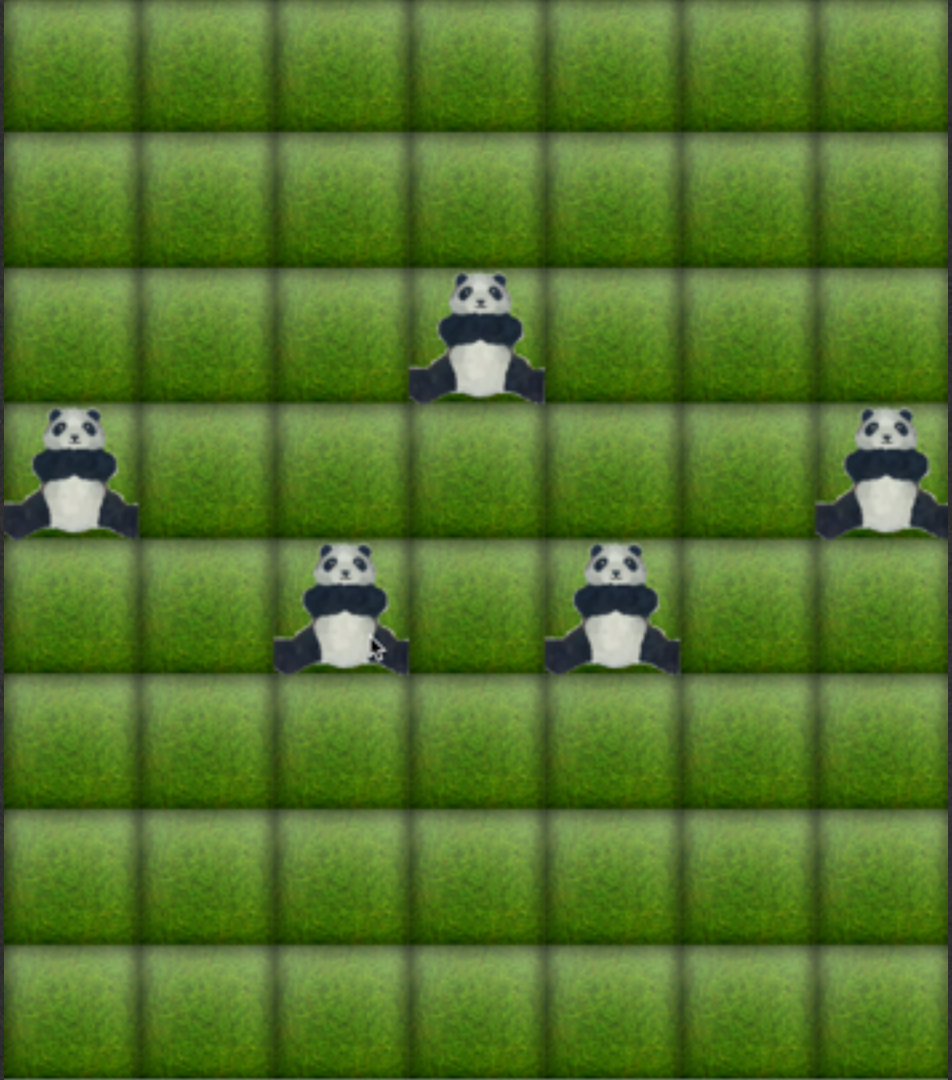
Combination: Selecting hands



Combination: Selecting Puzzles

- Puzzle game with pieces on the board
- Want to select interesting puzzles

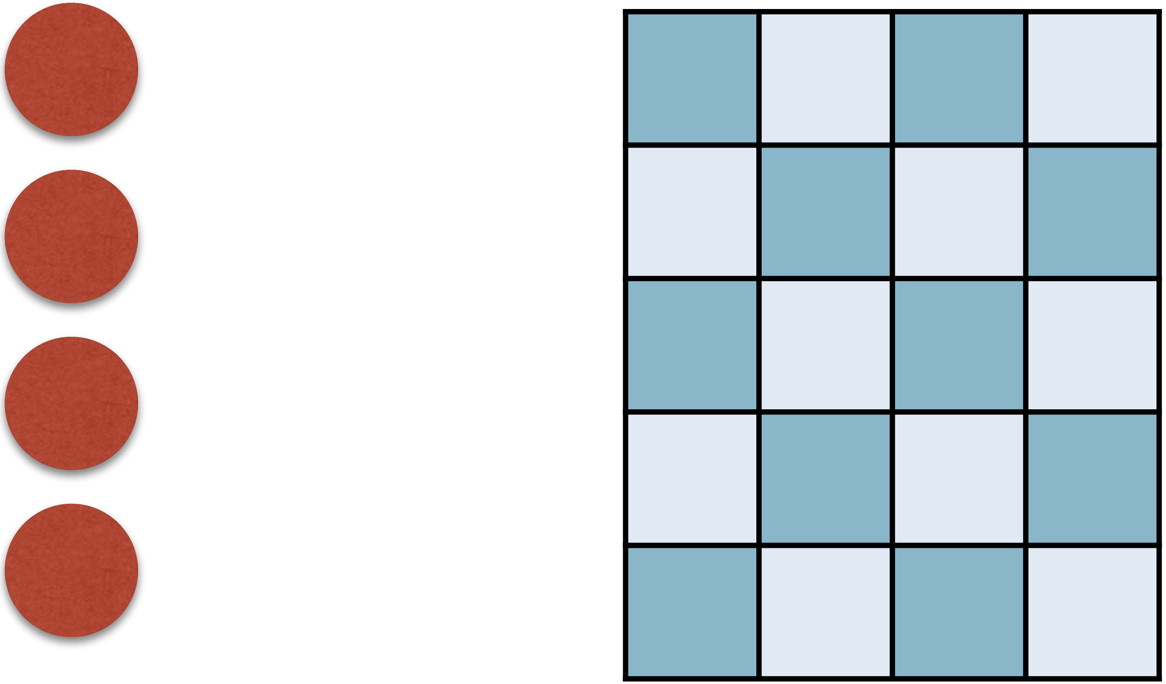




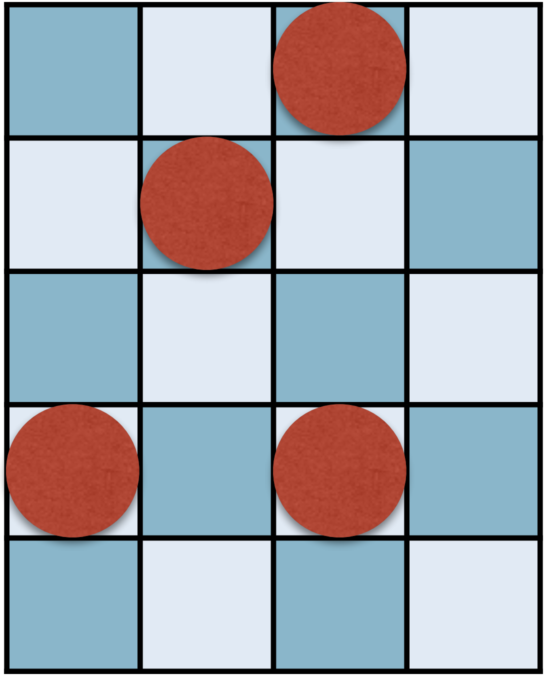
Combination: Selecting Puzzles

- Puzzle game with pieces on the board
- Want to select interesting puzzles
  - Solvable puzzles
  - Puzzles with one solution
  - Puzzles where every move leads to a solution

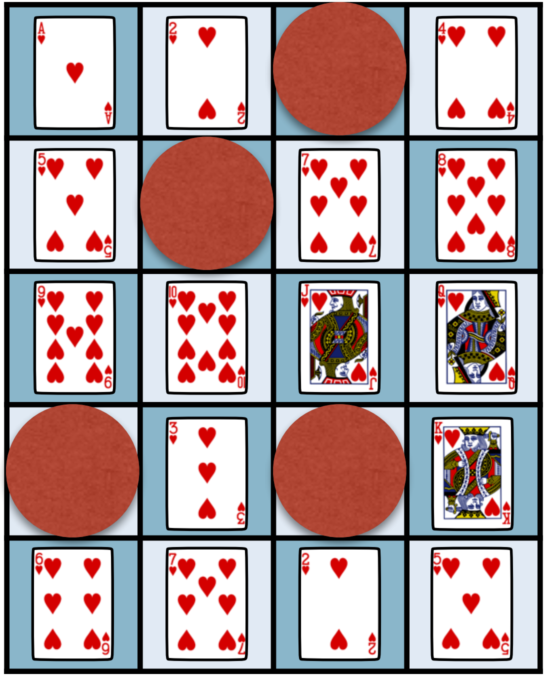
Combination: Placing Pieces



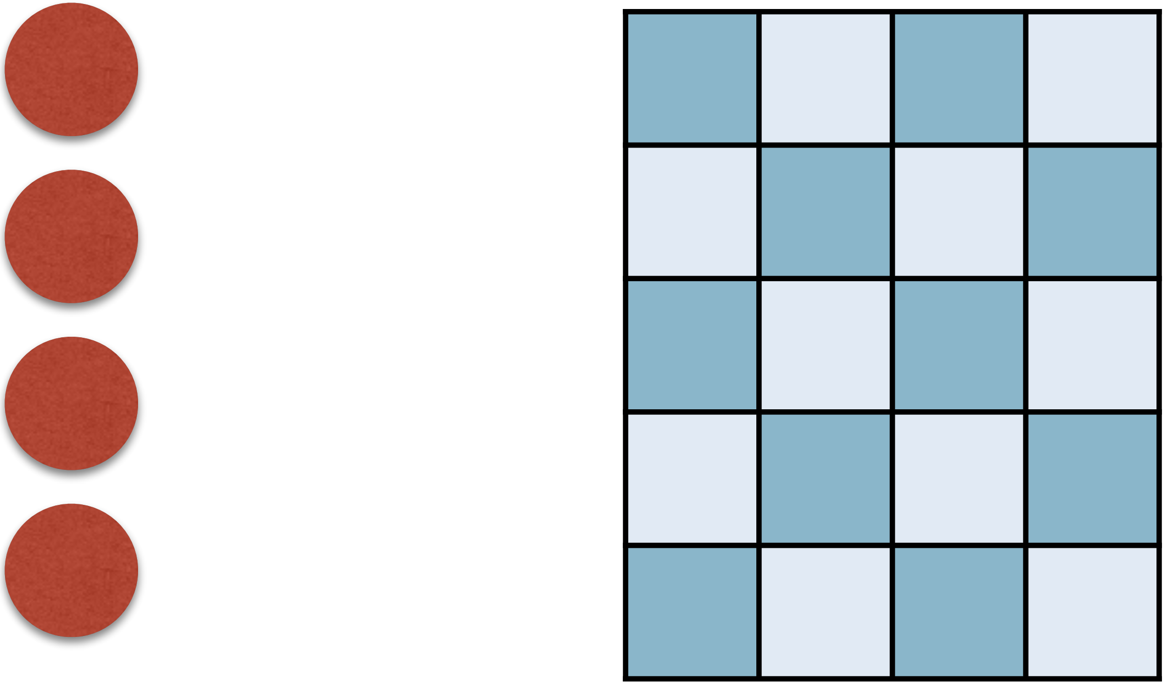
Combination: Placing Pieces



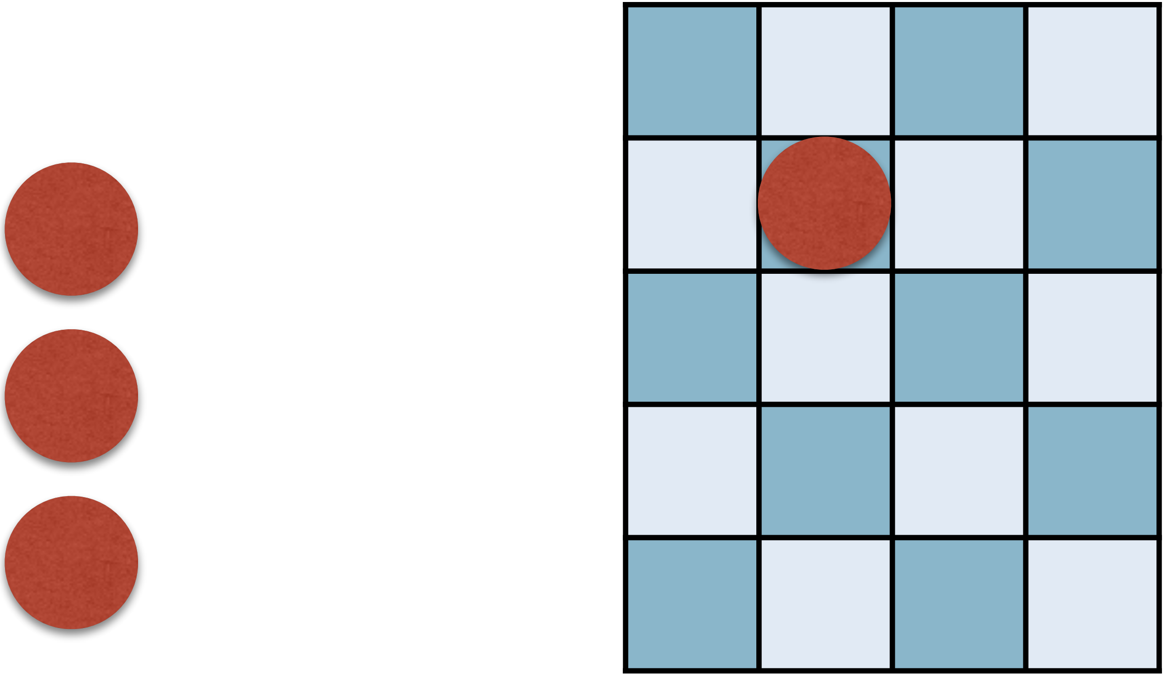
Combination: Placing Pieces



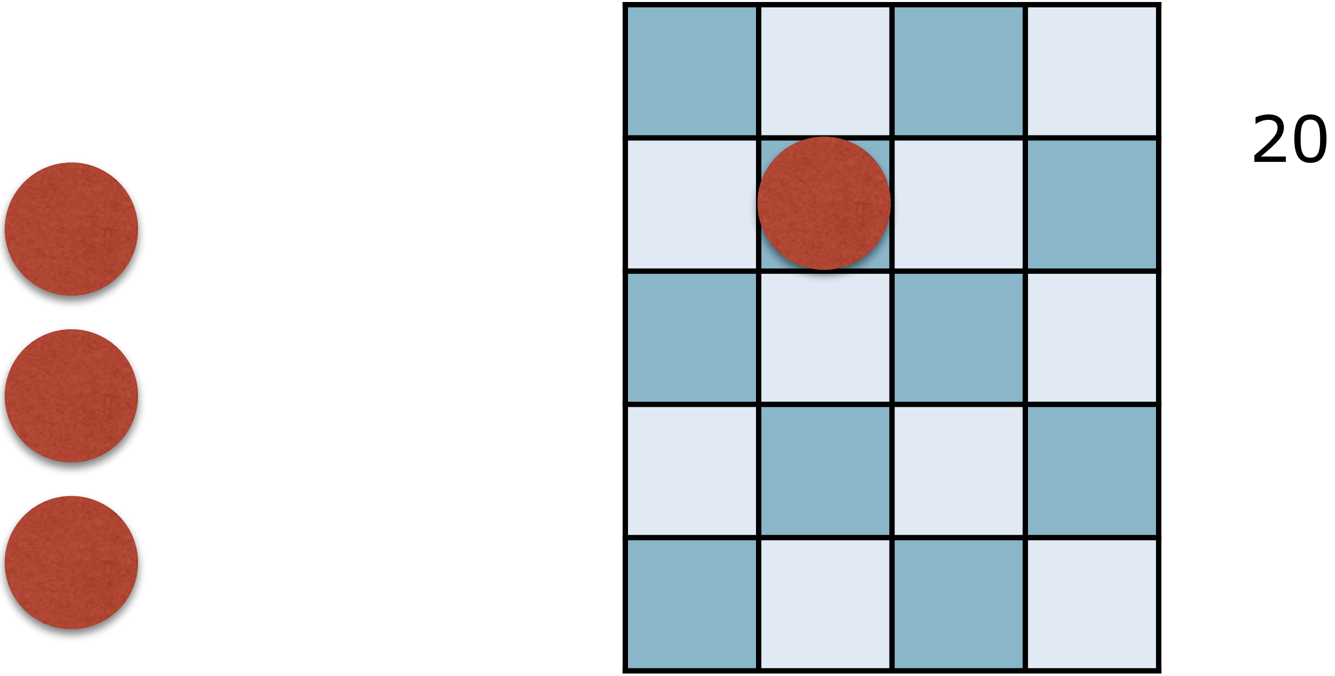
Combination: Counting



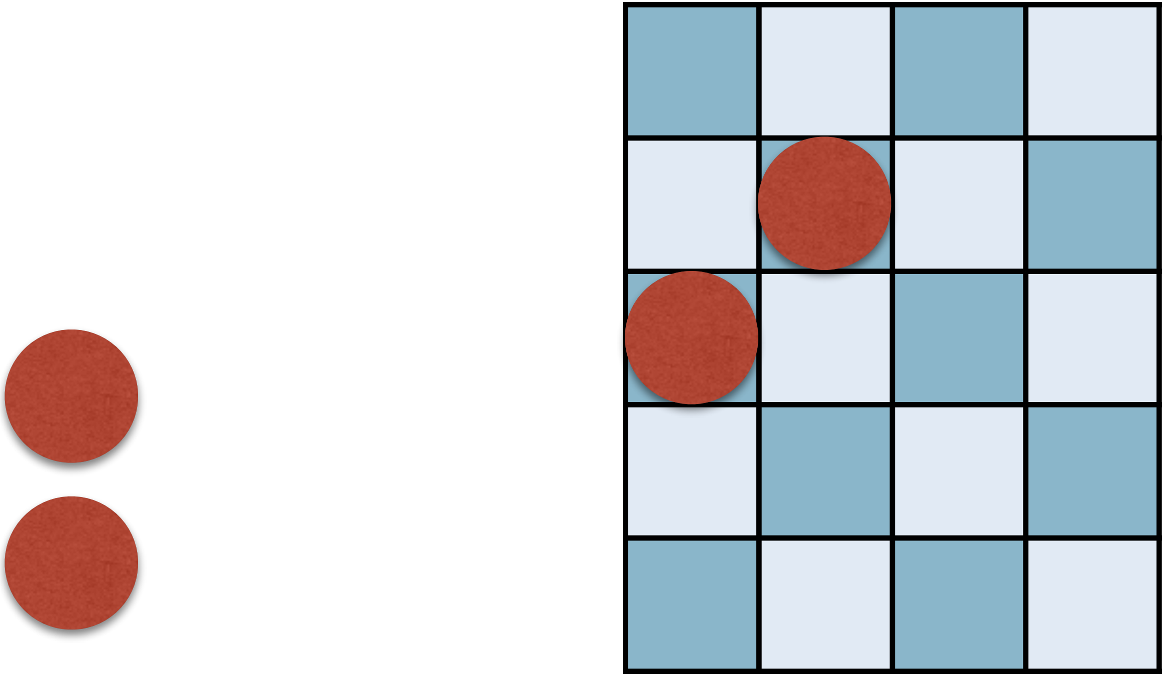
Combination: Counting



Combination: Counting

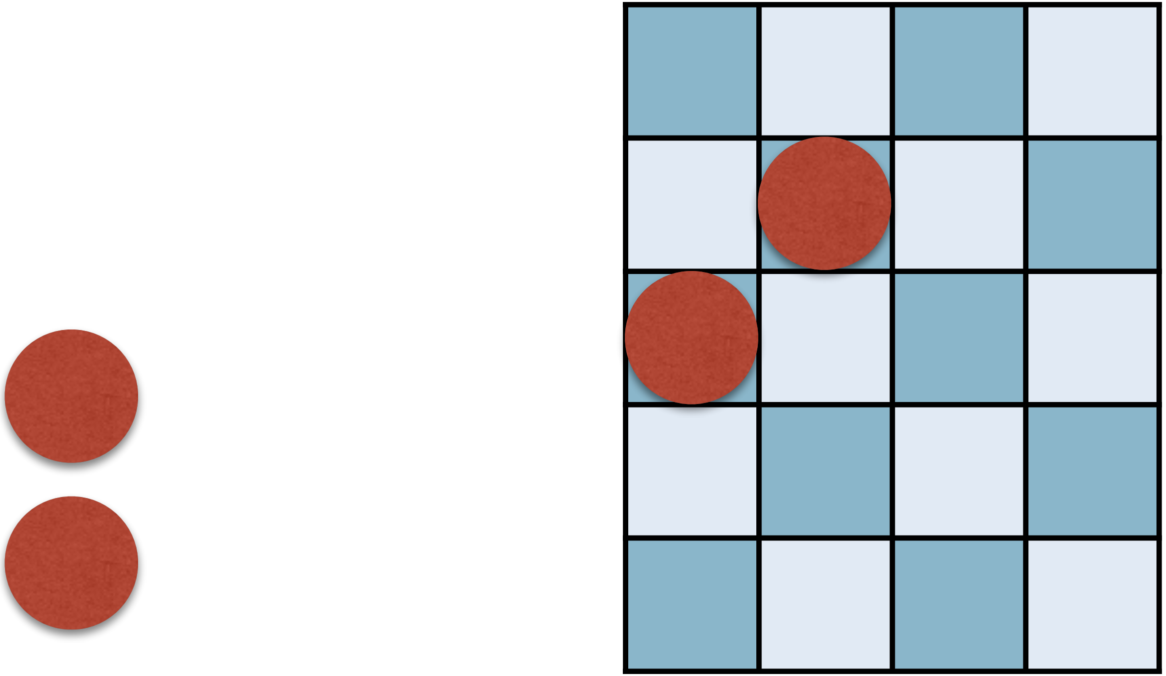


Combination: Counting



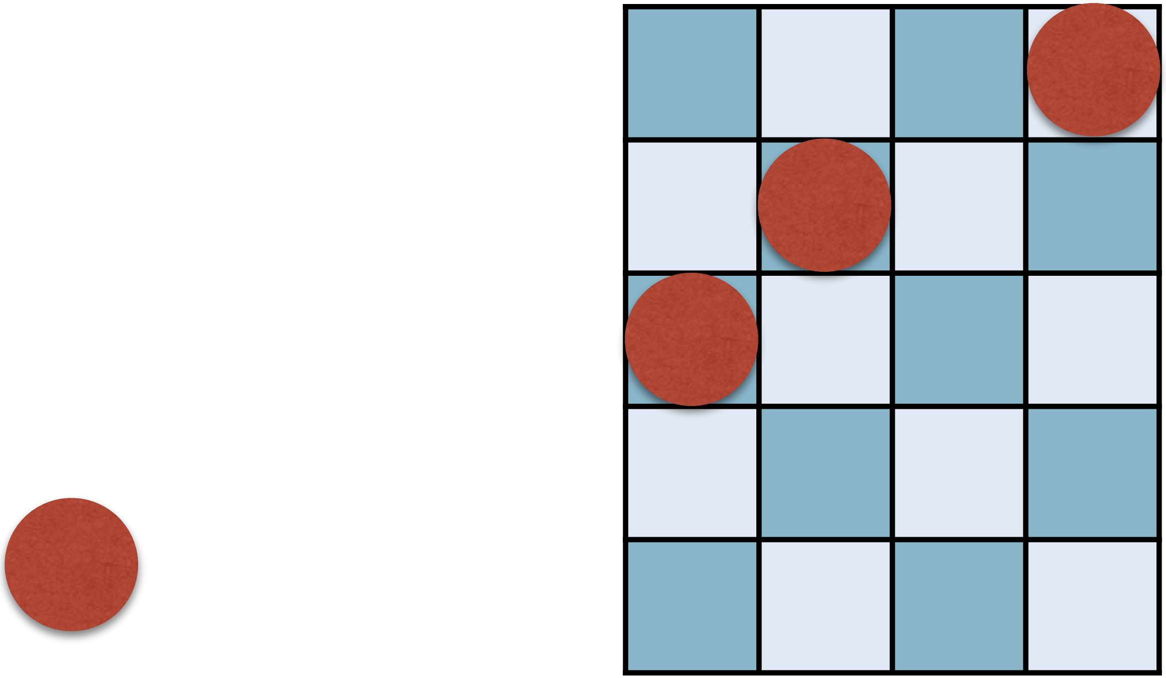
20

Combination: Counting



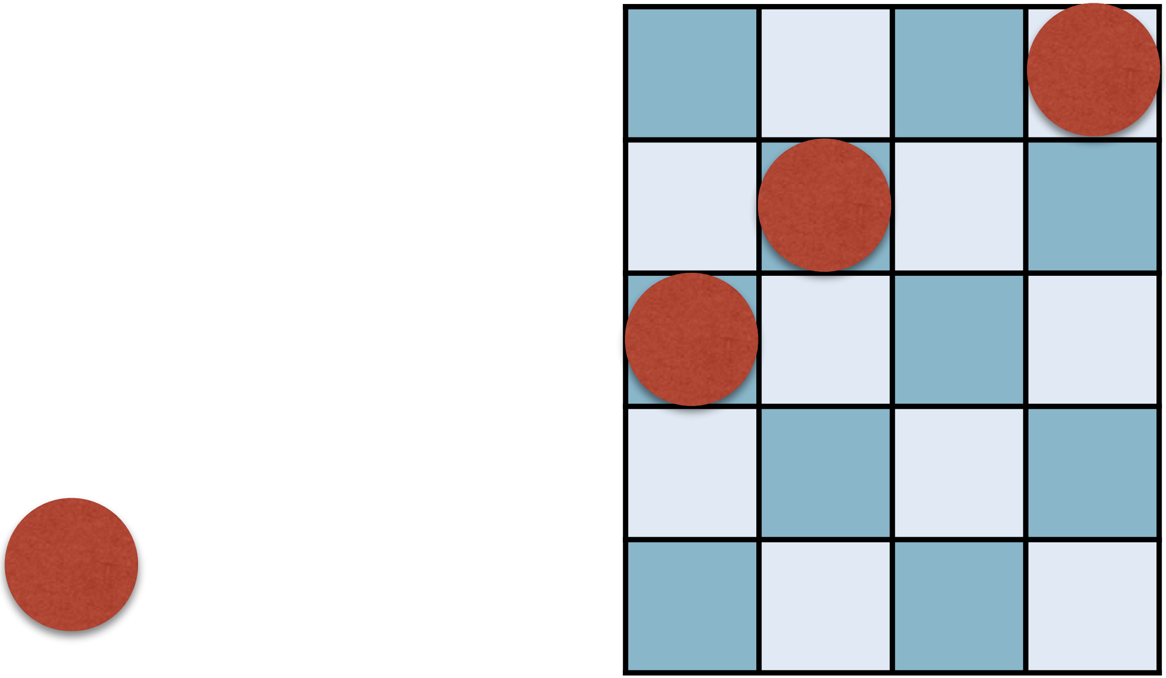
$20 \cdot 19$

Combination: Counting



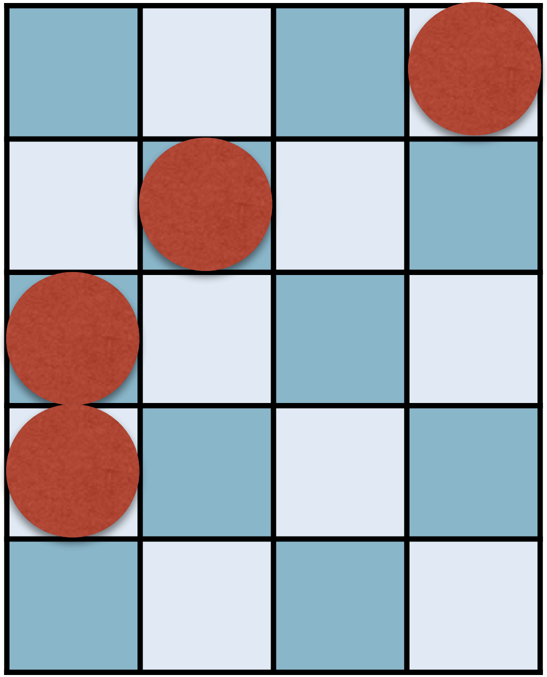
$20 \cdot 19$

Combination: Counting



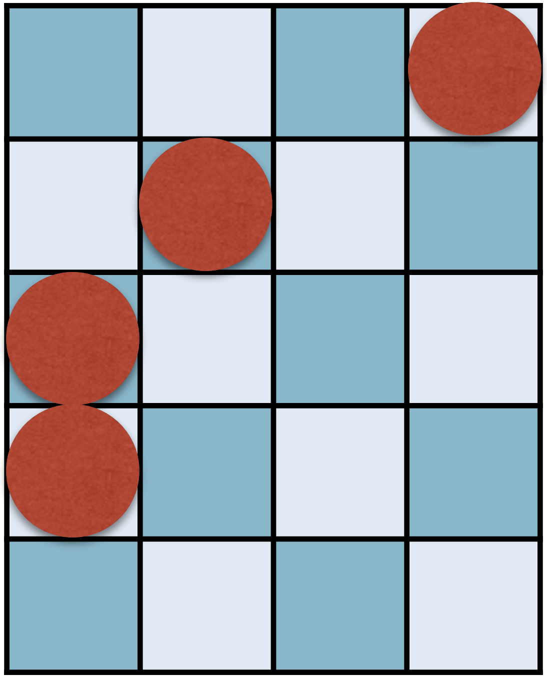
$20 \cdot 19 \cdot 18$

Combination: Counting



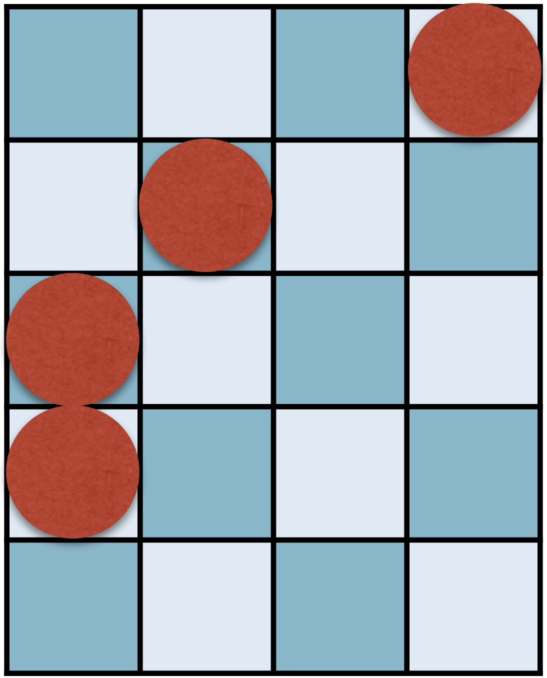
$20 \cdot 19 \cdot 18$

Combination: Counting



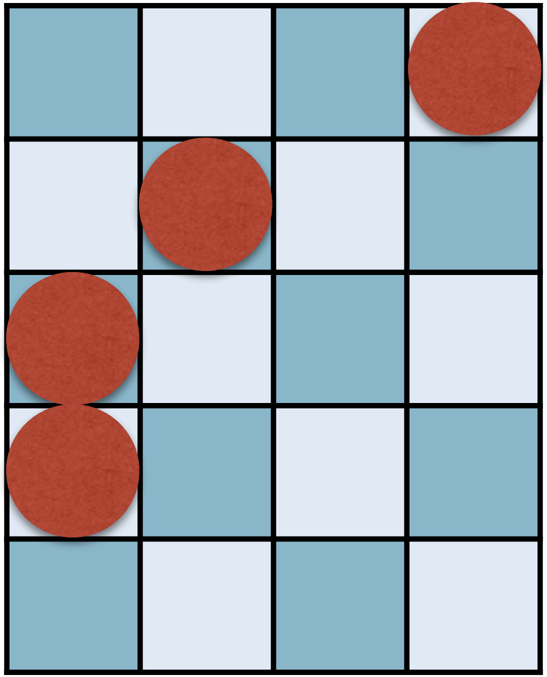
$20 \cdot 19 \cdot 18 \cdot 17$

Combination: Counting



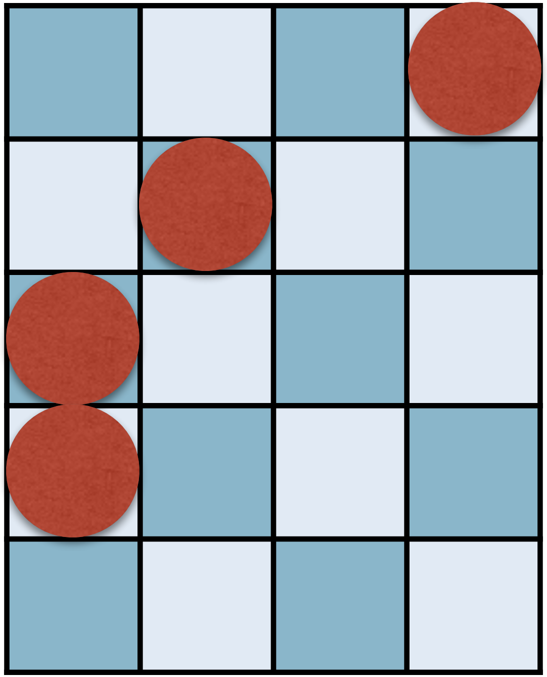
$20 \cdot 19 \cdot 18 \cdot 17$

Combination: Counting



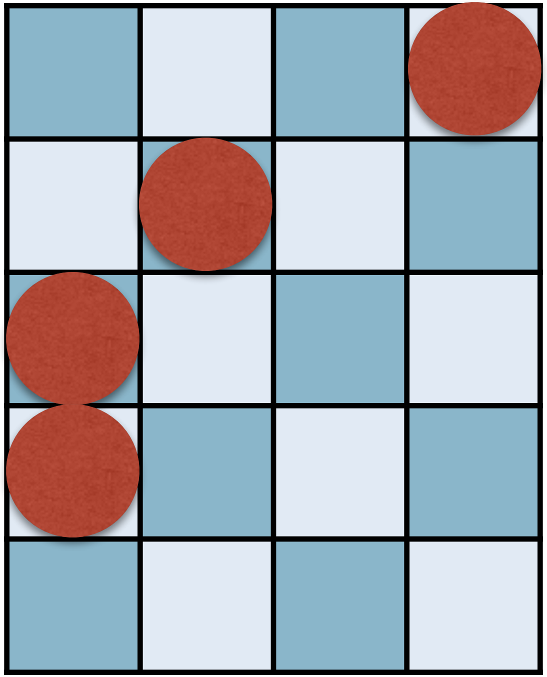
$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{}$$

Combination: Counting



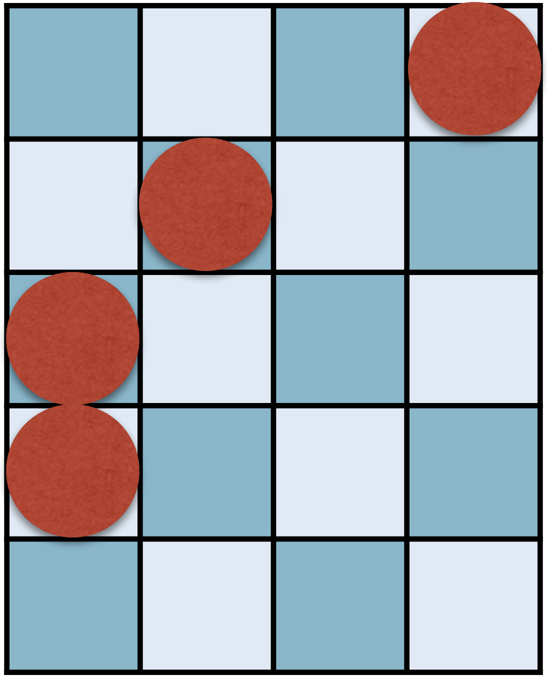
$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4}$$

Combination: Counting



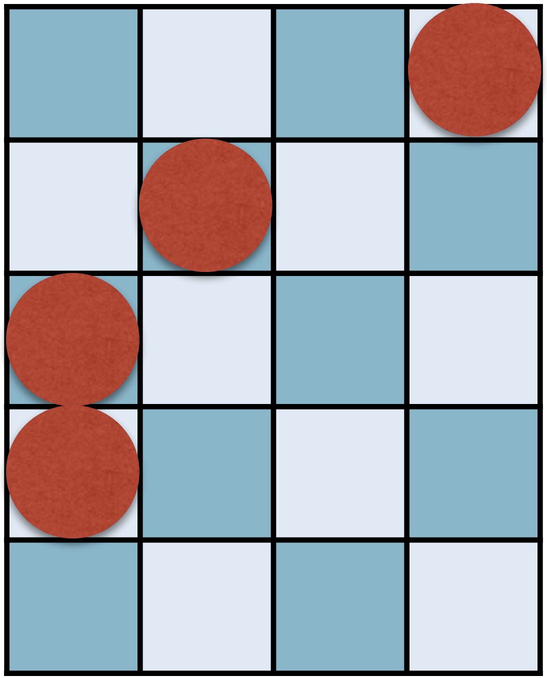
$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3}$$

Combination: Counting



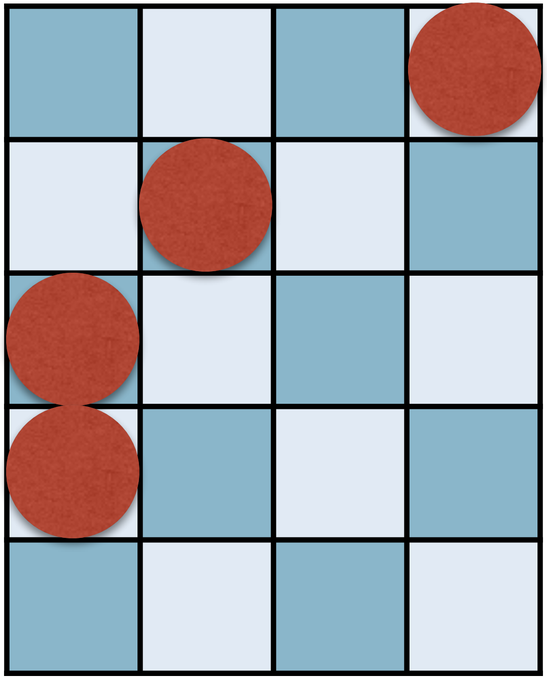
$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3 \cdot 2}$$

Combination: Counting



$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3 \cdot 2 \cdot 1}$$

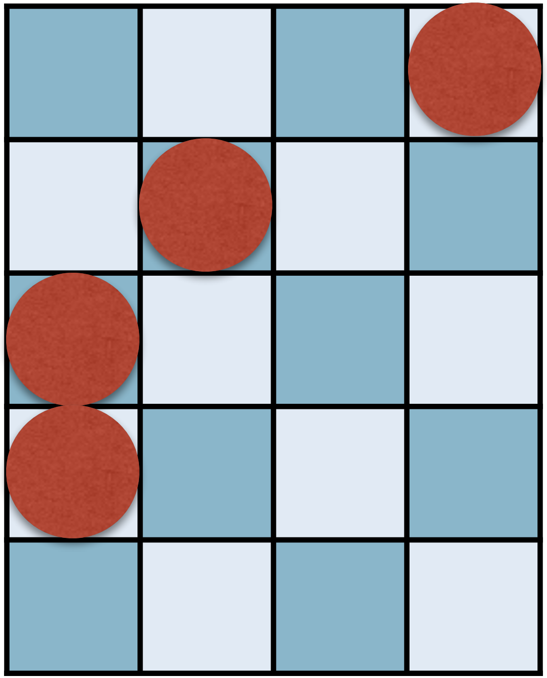
Combination: Counting



$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3 \cdot 2 \cdot 1}$$

20!

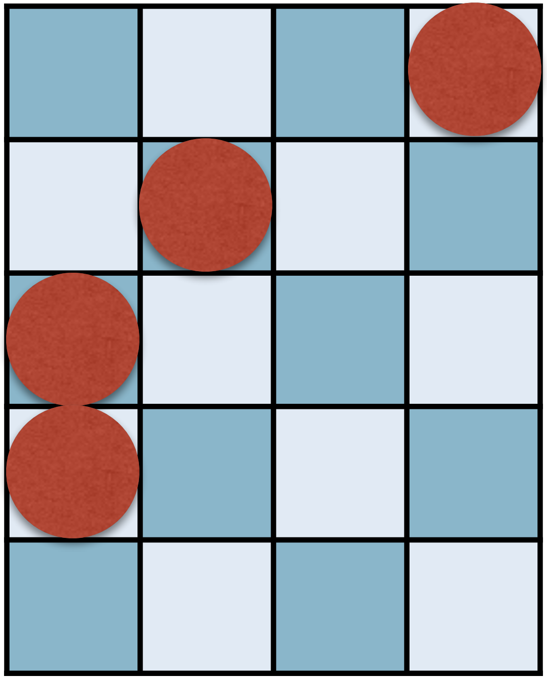
Combination: Counting



$$\frac{20\cdot 19\cdot 18\cdot 17}{4\cdot 3\cdot 2\cdot 1}$$

$$\frac{20!}{}$$

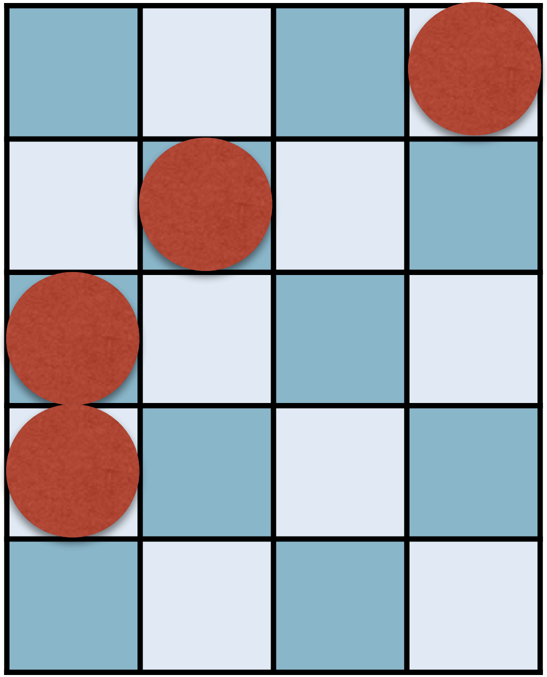
Combination: Counting



$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3 \cdot 2 \cdot 1}$$

$$\frac{20!}{16!}$$

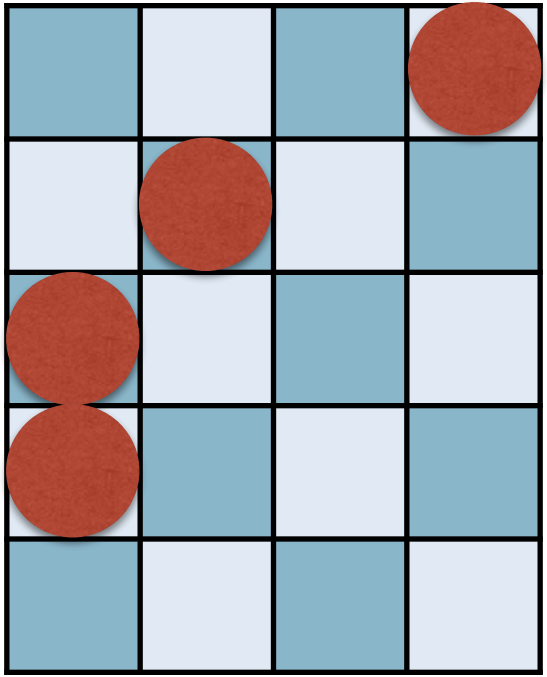
Combination: Counting



$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3 \cdot 2 \cdot 1}$$

$$\frac{20!}{16!\ 4!}$$

Combination: Counting



$$\frac{20 \cdot 19 \cdot 18 \cdot 17}{4 \cdot 3 \cdot 2 \cdot 1}$$

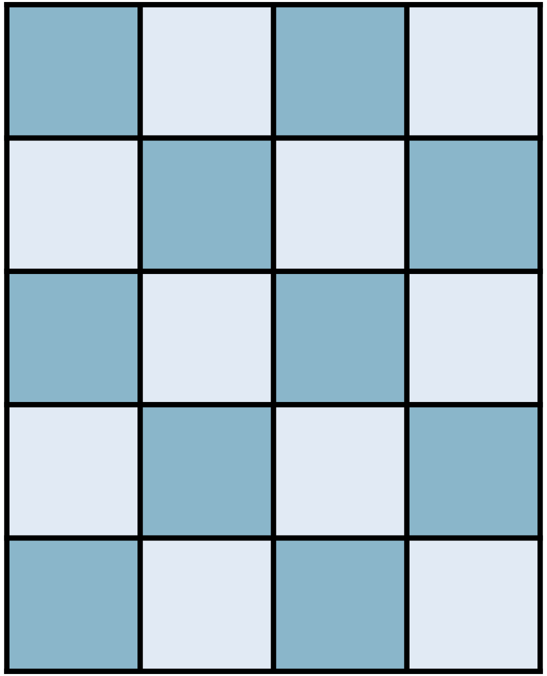
$$\frac{20!}{16!\,4!}$$

$$\binom{20}{4}$$

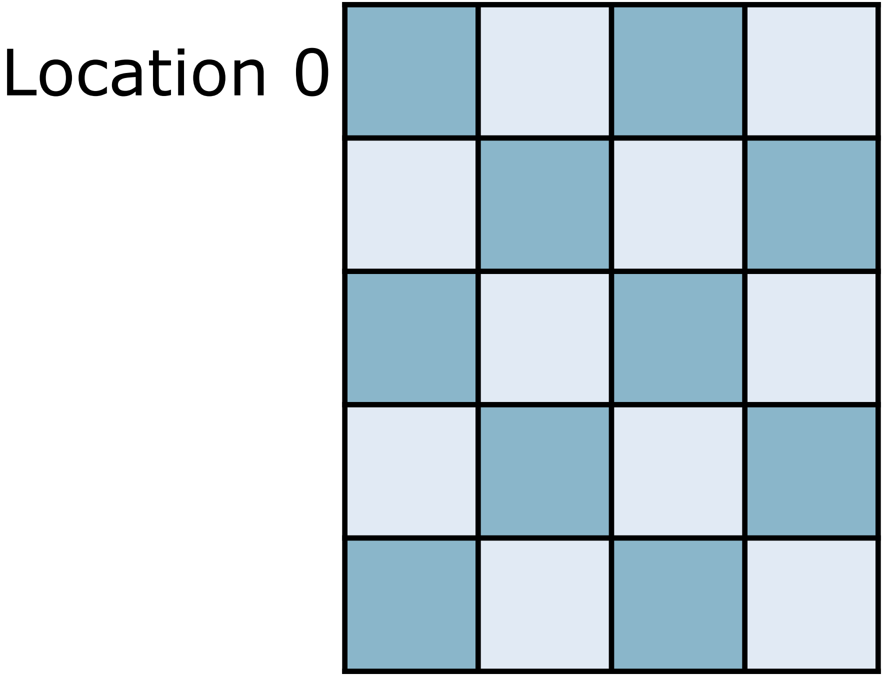
Definition

- **Ranking:** A function that takes a combination and returns an integer between 0…N-1 (where there are N possible combinations).

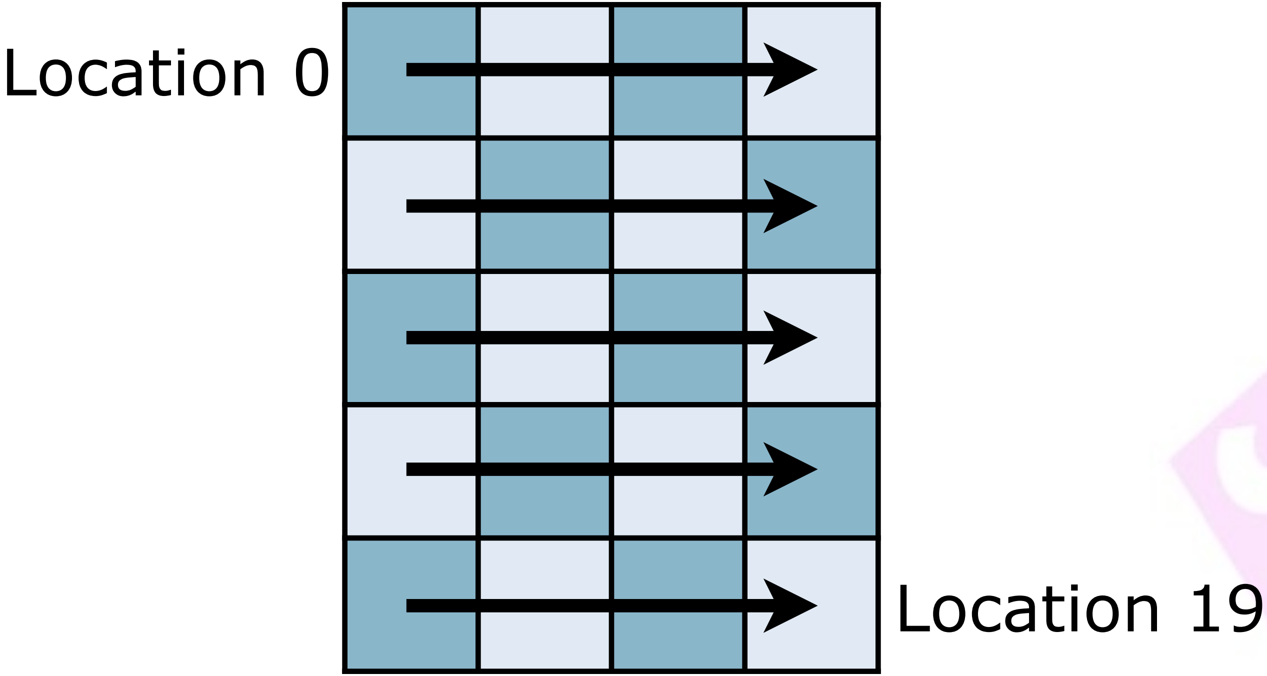
Combination: Ranking



Combination: Ranking

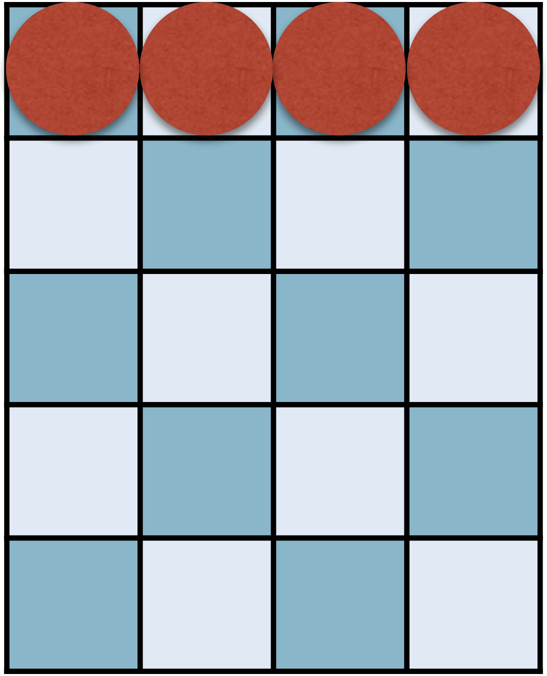


Combination: Ranking



Combination: Ranking

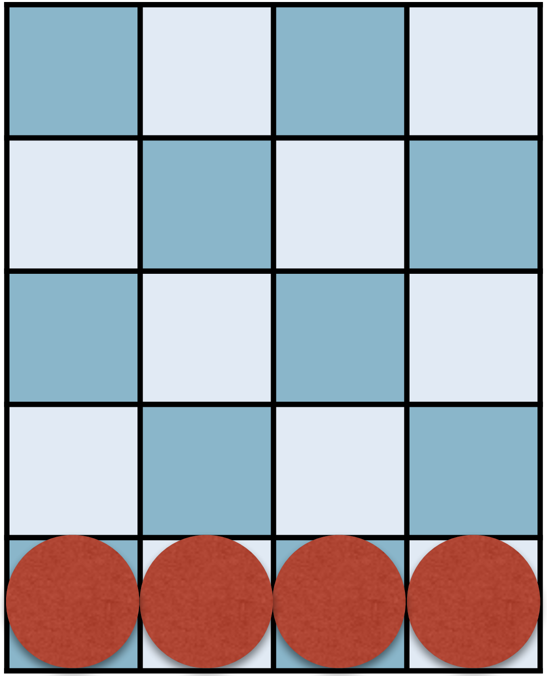
Rank: 0



Combination: Ranking

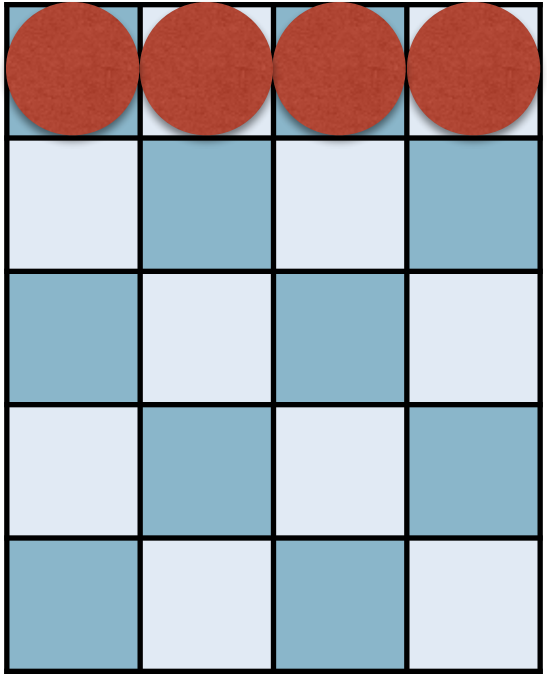
Rank: 4844

$$\binom{20}{4} - 1$$



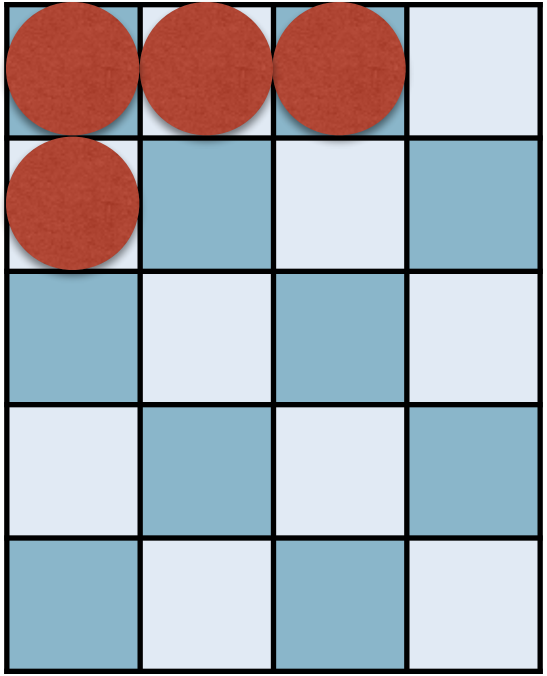
Combination: Ranking

Rank: 0



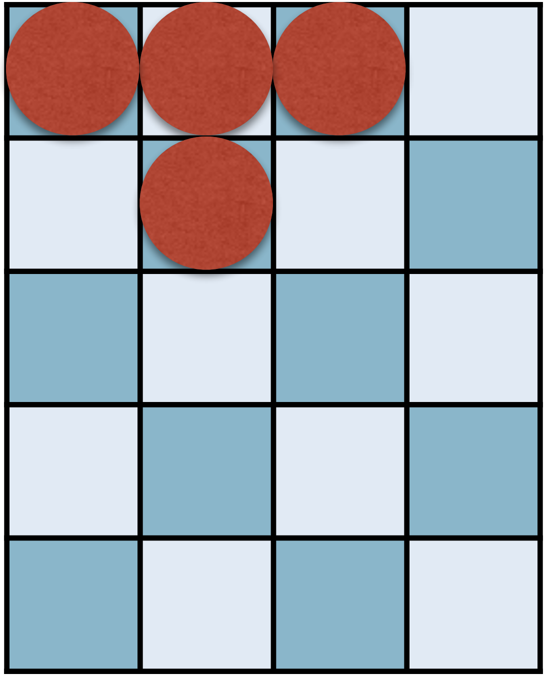
Combination: Ranking

Rank: 1



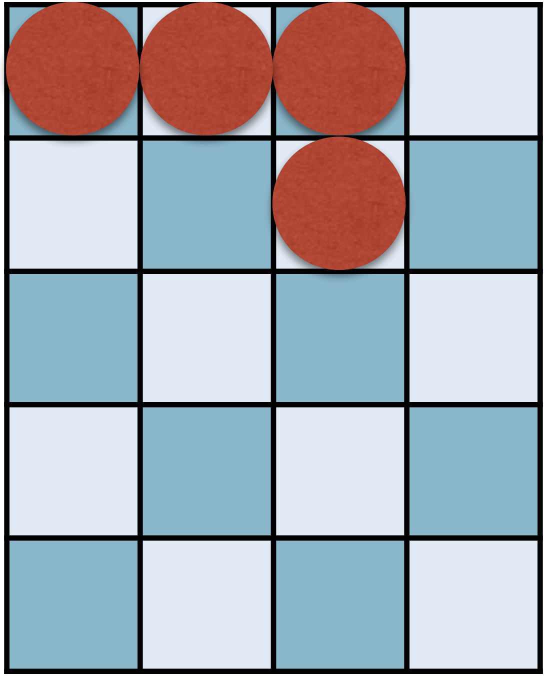
Combination: Ranking

Rank: 2



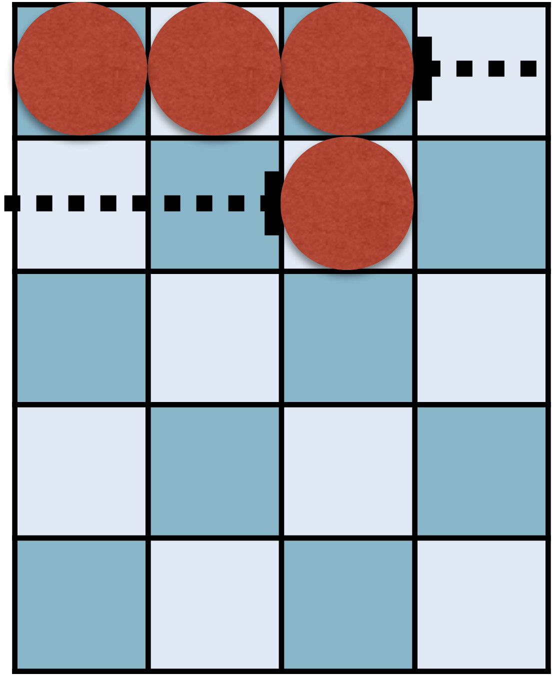
Combination: Ranking

Rank: 3



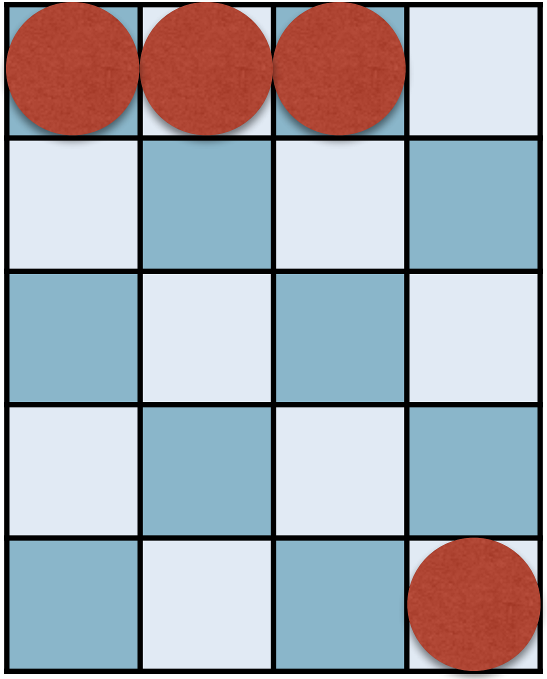
Combination: Ranking

Rank: 3



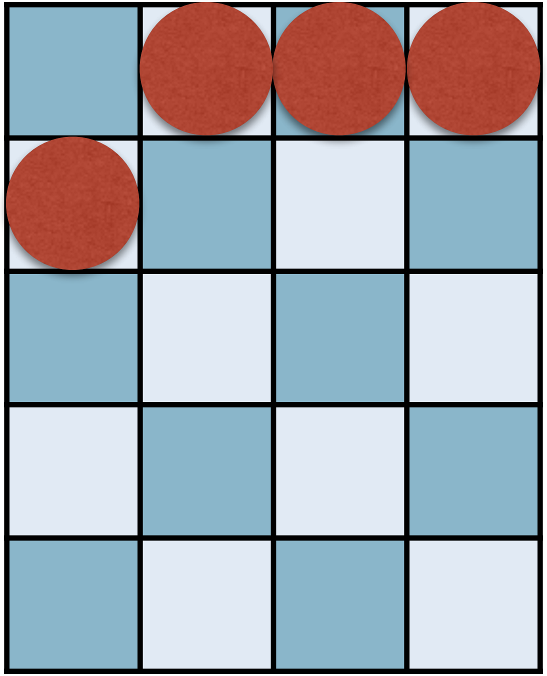
Combination: Ranking

Rank: 16



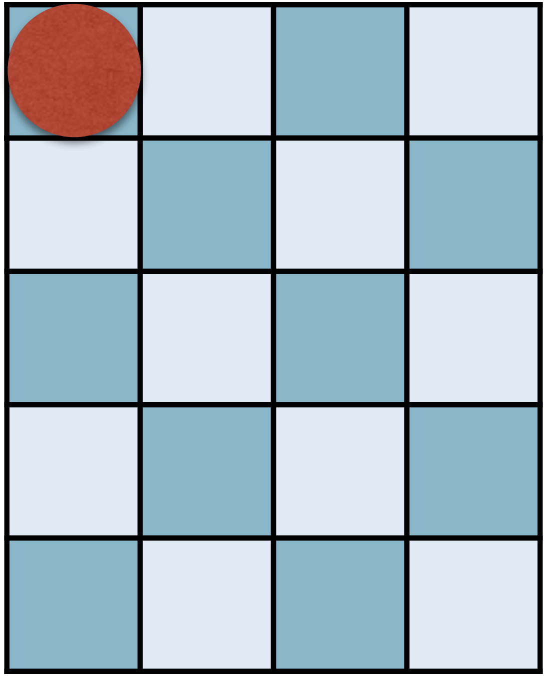
Combination: Ranking

Rank: ?



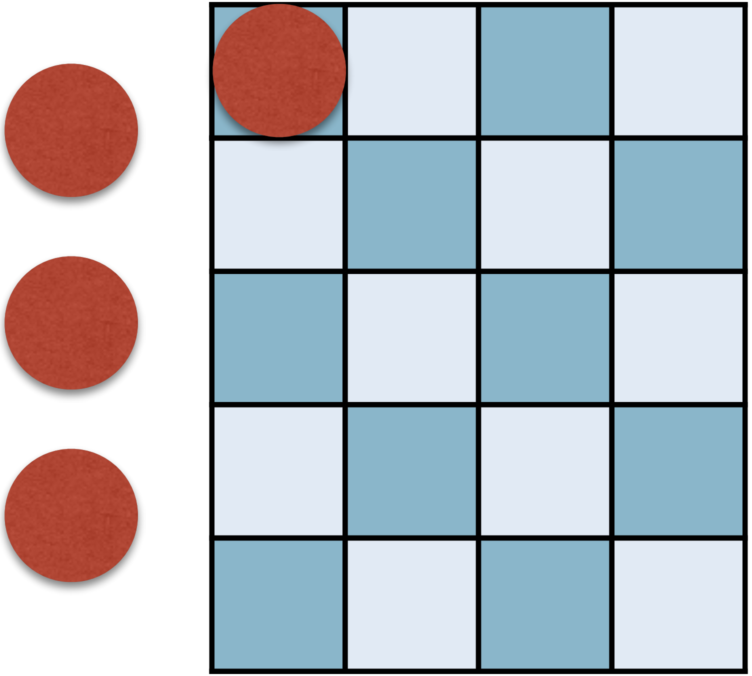
Combination: Ranking

Rank: ?



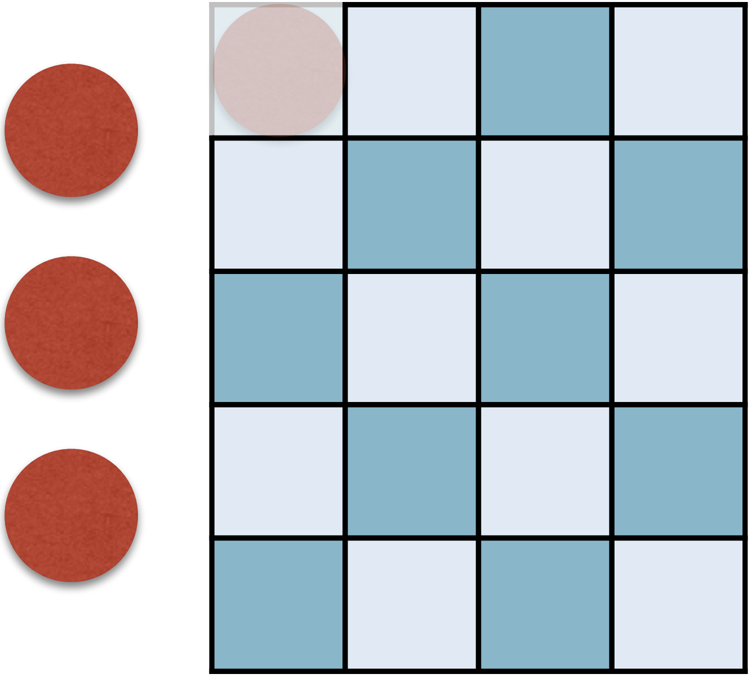
Combination: Ranking

Rank: ?



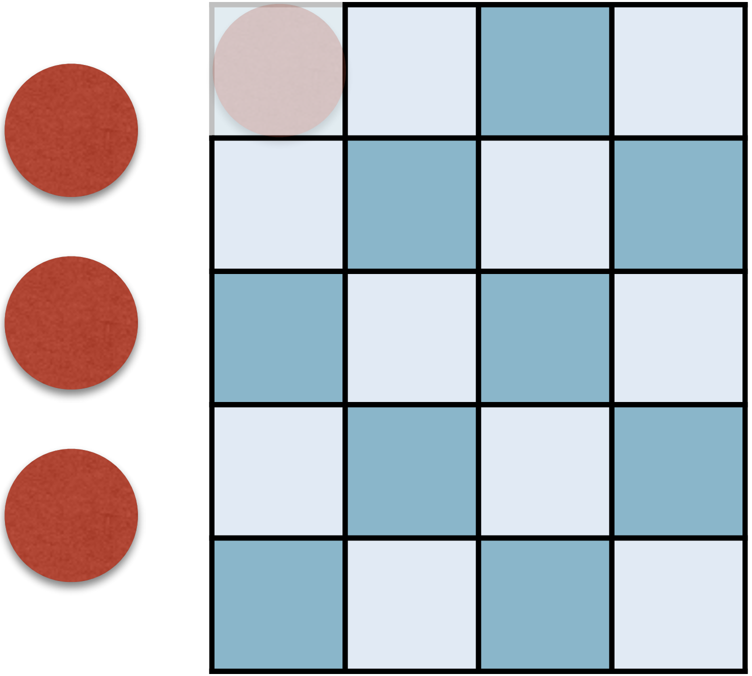
Combination: Ranking

Rank: ?



Combination: Ranking

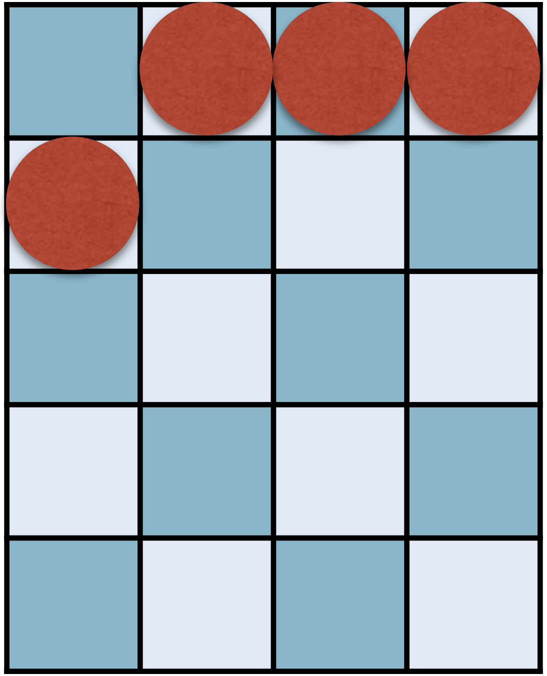
Rank: ?



$$\binom{19}{3}$$

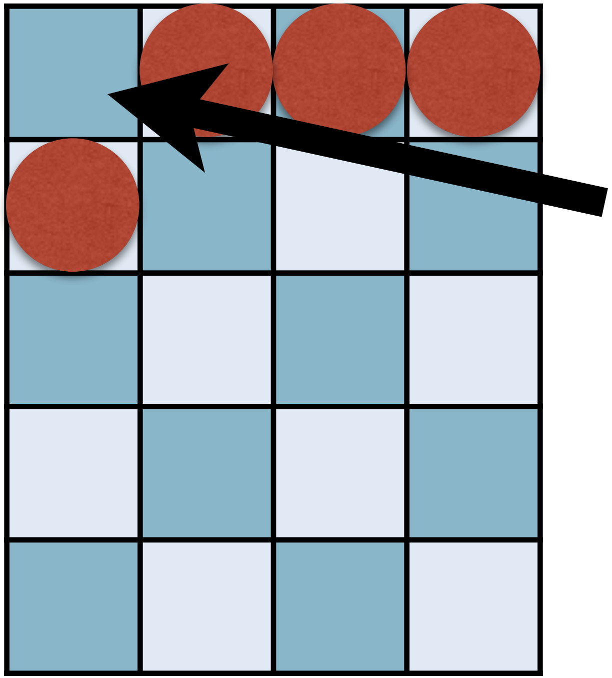
Combination: Ranking

Rank: ?



Combination: Ranking

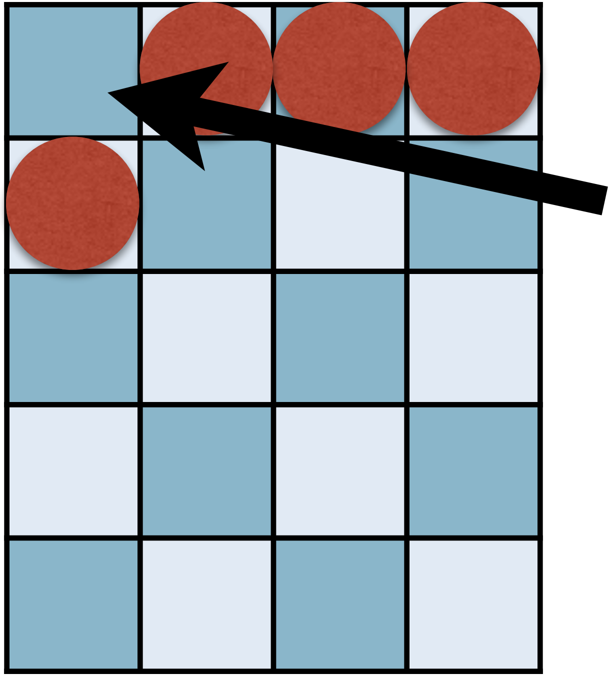
Rank: ?



How many possible boards with a piece here?

Combination: Ranking

Rank: ?

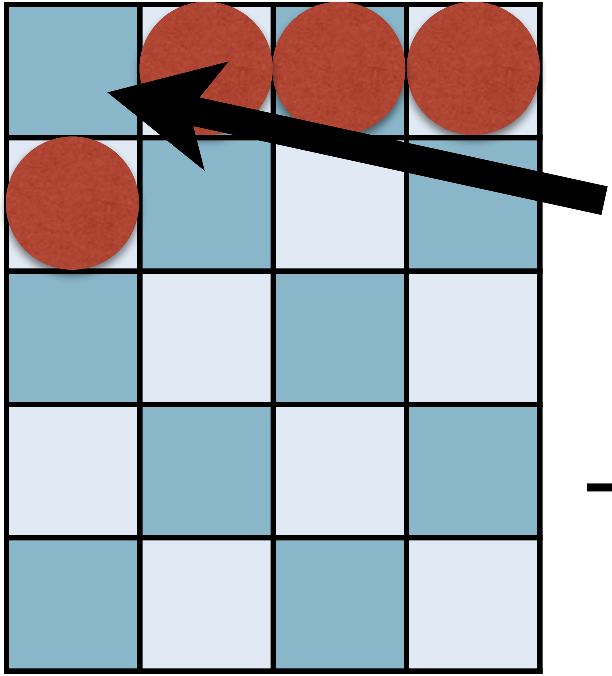


How many possible boards with a piece here?

19!

Combination: Ranking

Rank: ?

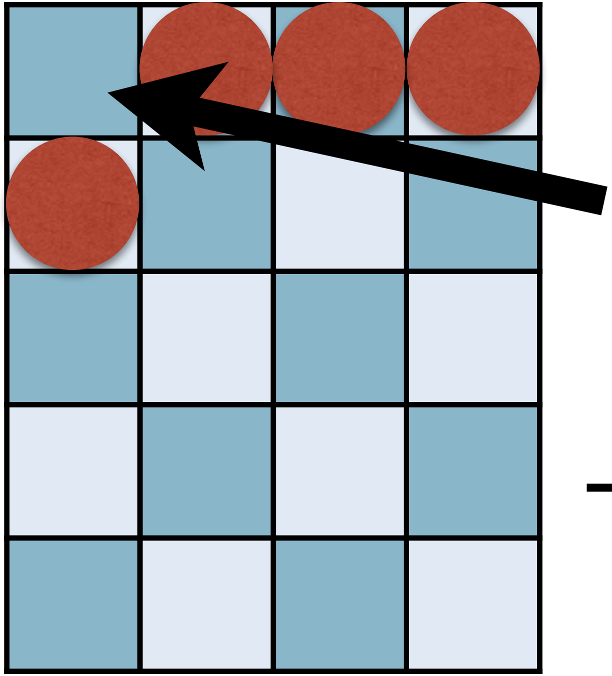


How many possible boards with a piece here?

19!

Combination: Ranking

Rank: ?

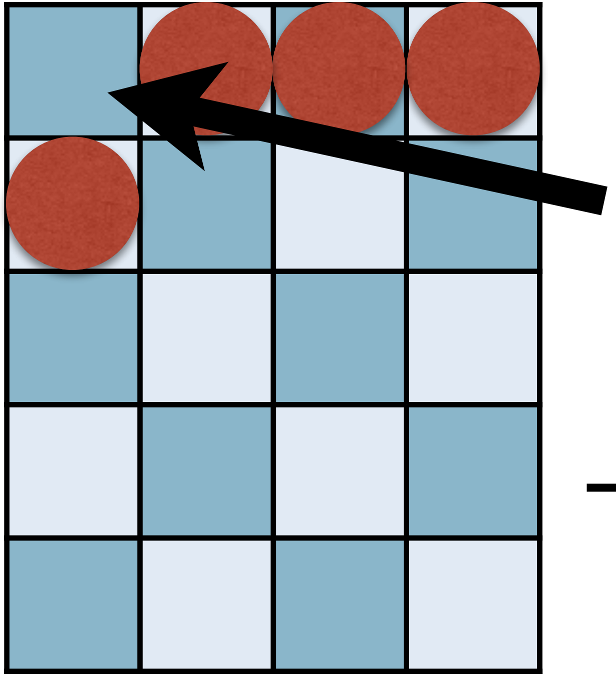


How many possible boards with a piece here?

$$\frac{19!}{16!}$$

Combination: Ranking

Rank: ?

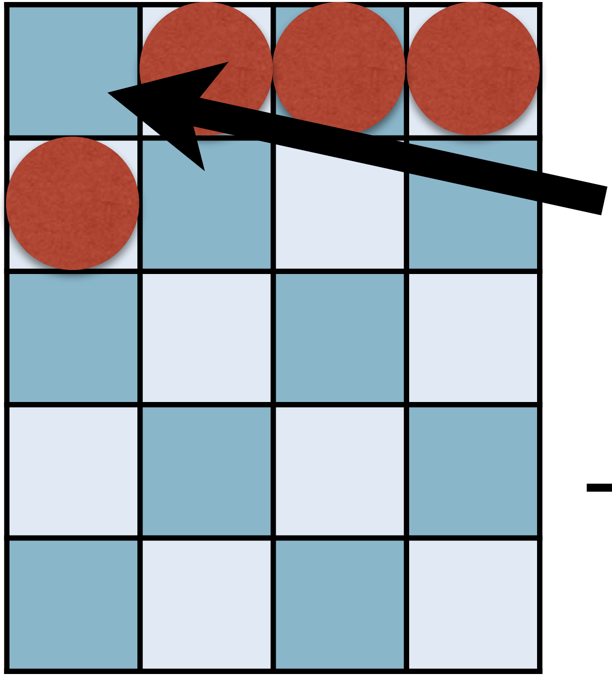


How many possible boards with a piece here?

$$\frac{19!}{16!\,3!}$$

Combination: Ranking

Rank: ?

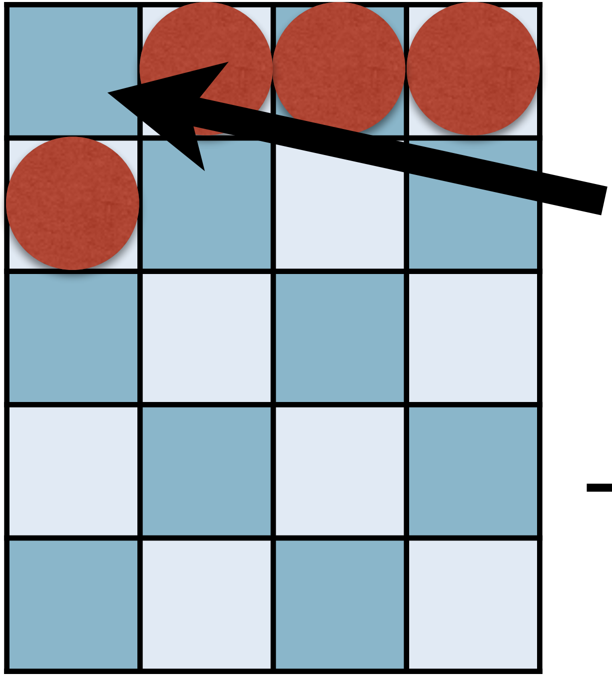


How many possible boards with a piece here?

$$\frac{19!}{16!\,3!} = 969$$

Combination: Ranking

Rank: 969

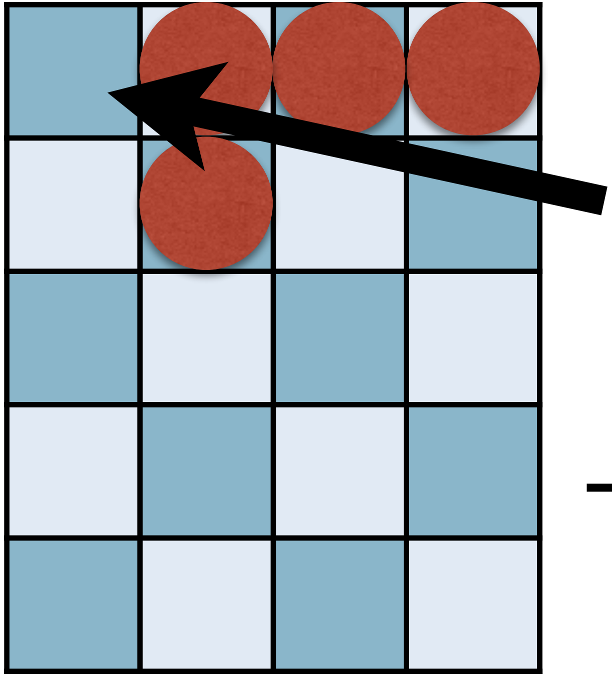


How many possible boards with a piece here?

$$\frac{19!}{16!\,3!} = 969$$

Combination: Ranking

Rank: 969

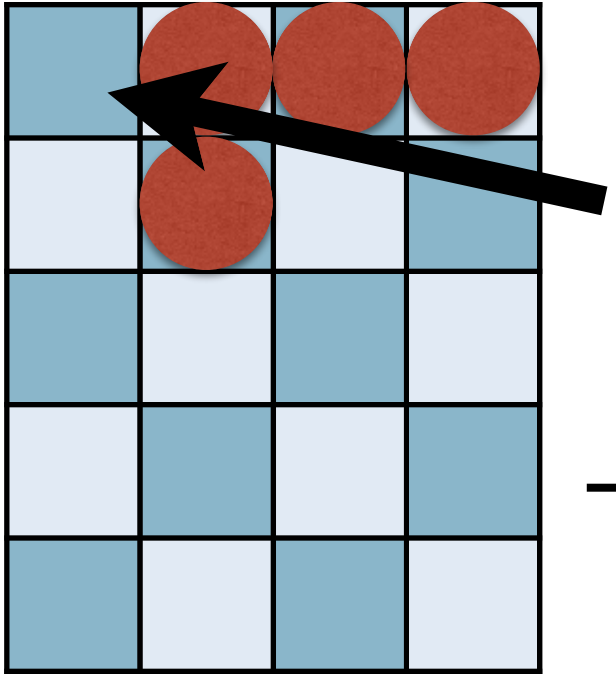


How many possible boards with a piece here?

$$\frac{19!}{16!\,3!} = 969$$

Combination: Ranking

Rank: 970

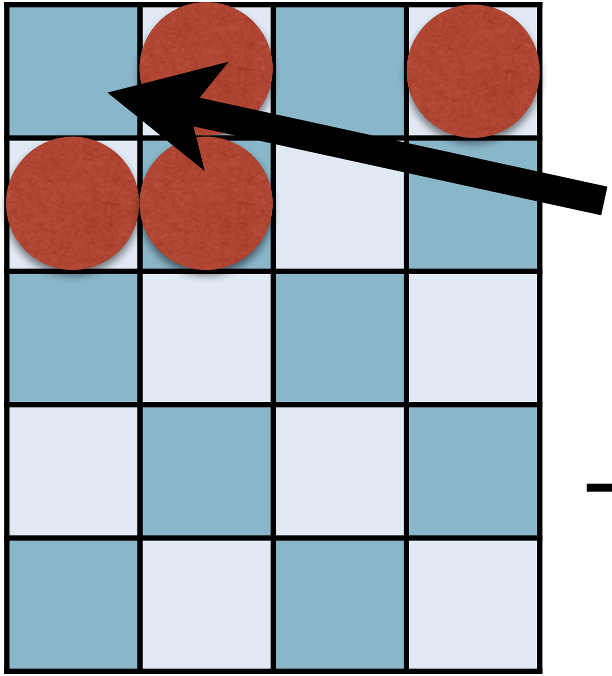


How many possible boards with a piece here?

$$\frac{19!}{16!\,3!} = 969$$

Combination: Ranking

Rank: 970

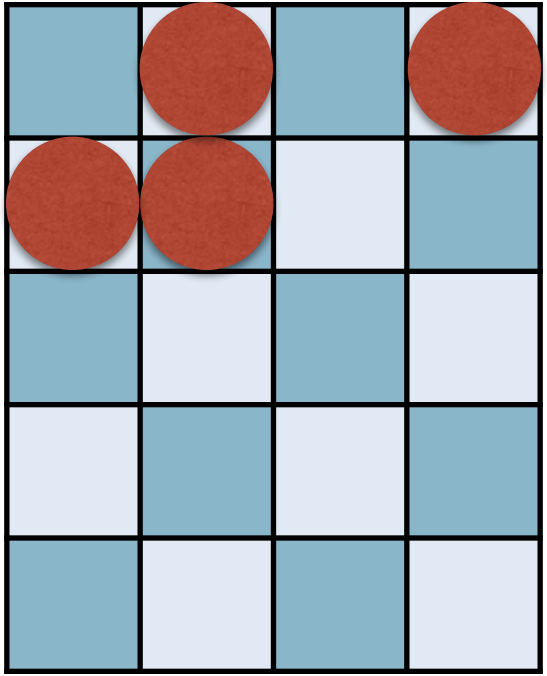


How many possible boards with a piece here?

$$\frac{19!}{16!\,3!} = 969$$

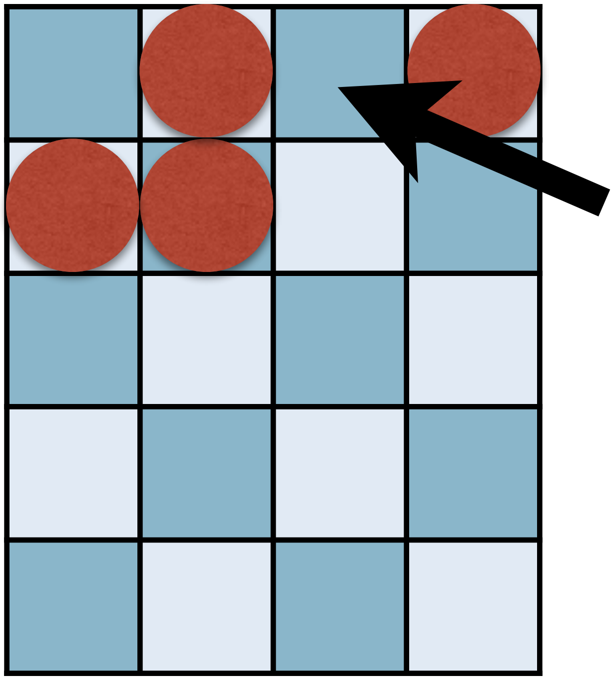
Combination: Ranking

Rank: 969+?



Combination: Ranking

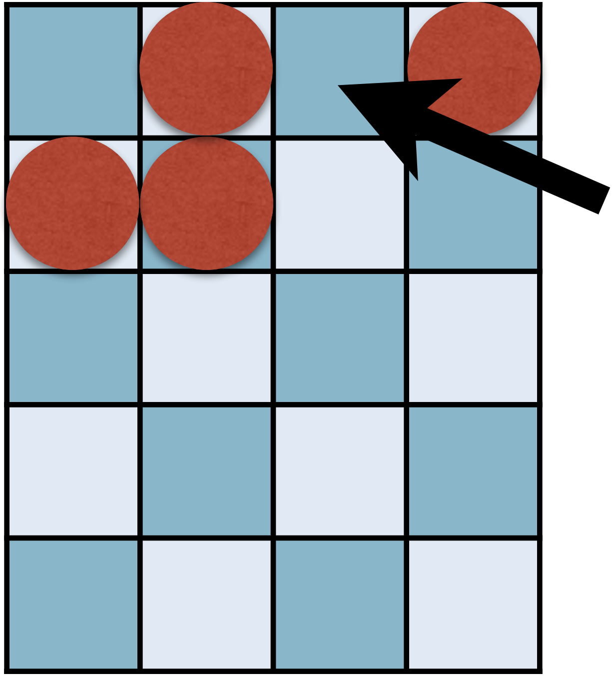
Rank: 969+?



How many possible boards with a piece here?

Combination: Ranking

Rank: 969+?

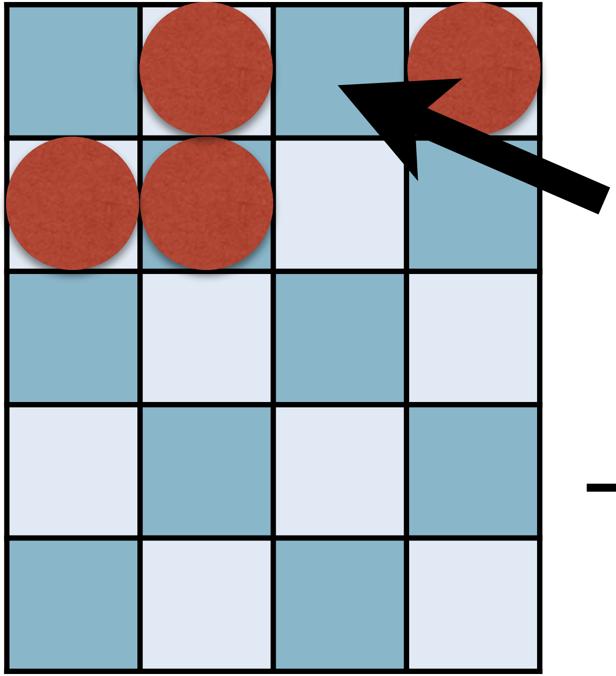


How many possible boards with a piece here?

17!

Combination: Ranking

Rank: 969+?

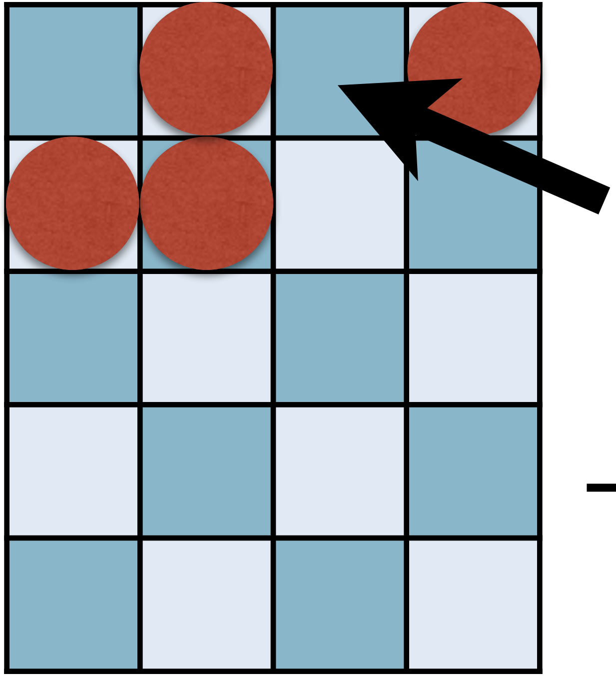


How many possible boards with a piece here?

$$\frac{17!}{}$$

Combination: Ranking

Rank: 969+?

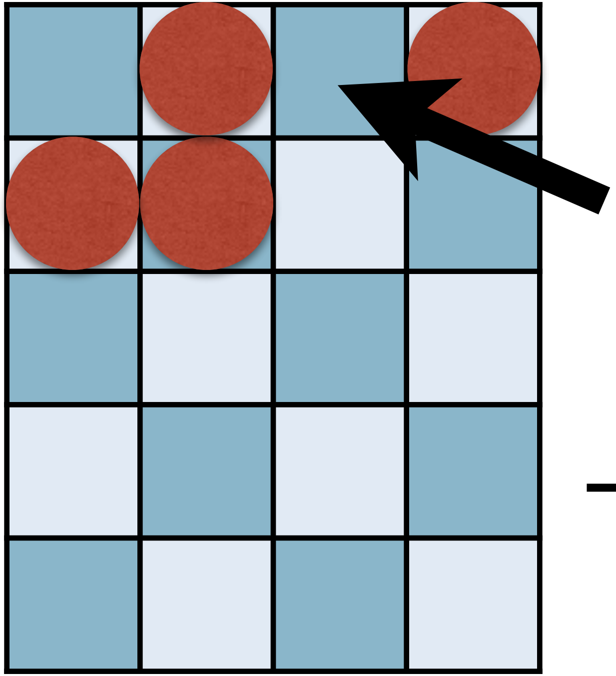


How many possible boards with a piece here?

$$\frac{17!}{15!}$$

Combination: Ranking

Rank: 969+?

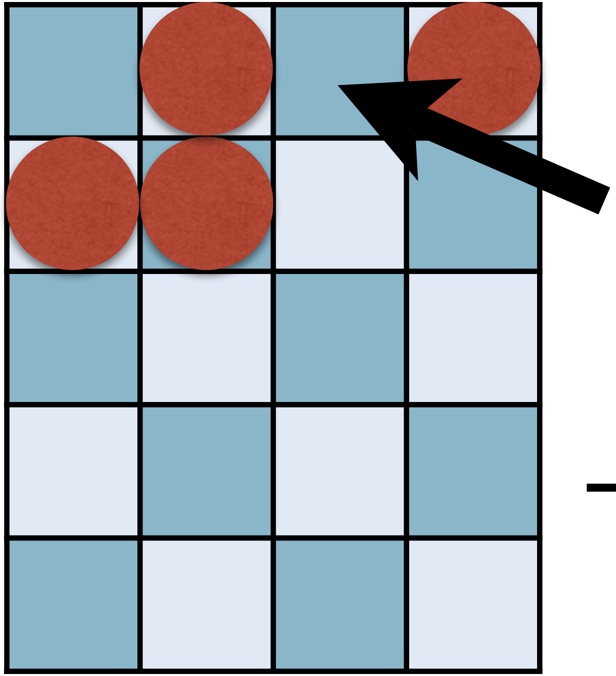


How many possible boards with a piece here?

$$\frac{17!}{15!\,2!}$$

Combination: Ranking

Rank: 969+?

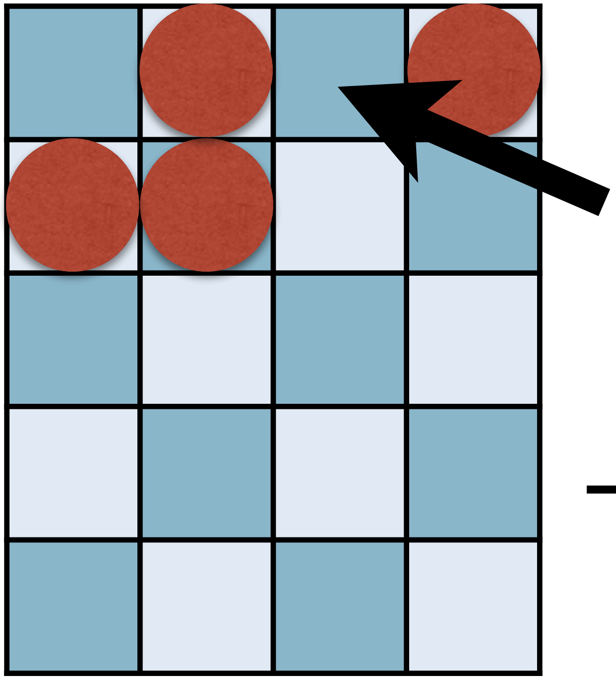


How many possible boards with a piece here?

$$\frac{17!}{15!\,2!} = 136$$

Combination: Ranking

Rank: 969+136

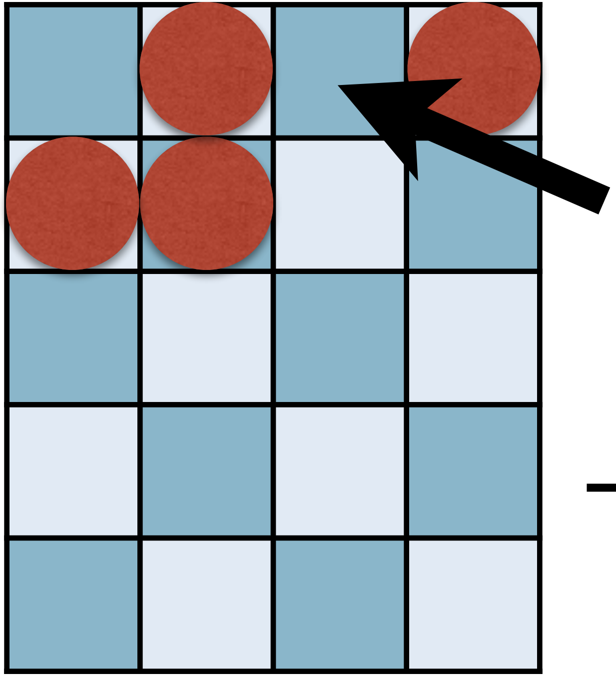


How many possible boards with a piece here?

$$\frac{17!}{15!\,2!} = 136$$

Combination: Ranking

Rank: 1105

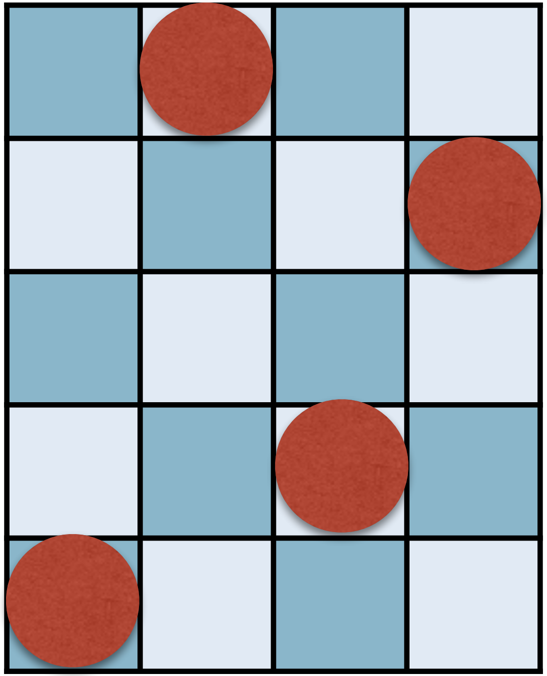


How many possible boards with a piece here?

$$\frac{17!}{15!\,2!} = 136$$

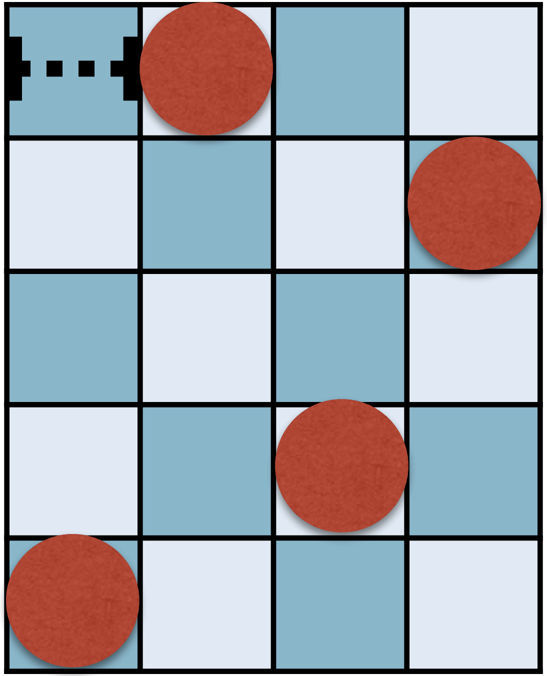
Combination: General Approach

Sum the number of ranks that were skipped for each of the spaces between pieces.



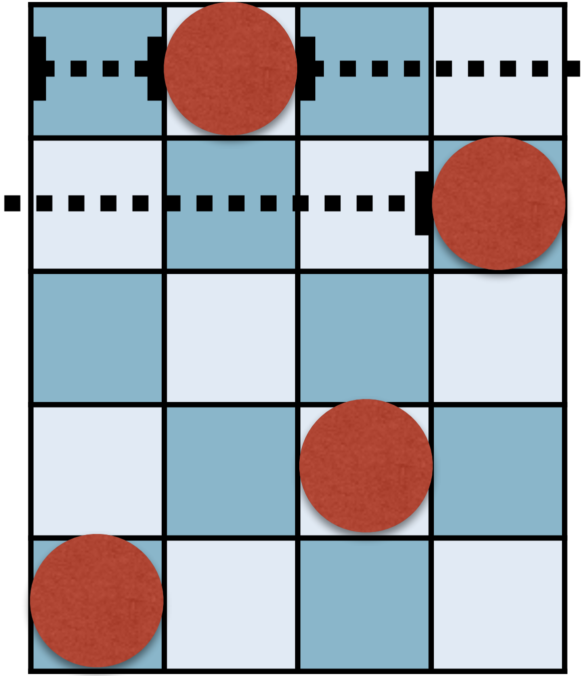
Combination: General Approach

Sum the number of ranks that were skipped for each of the spaces between pieces.



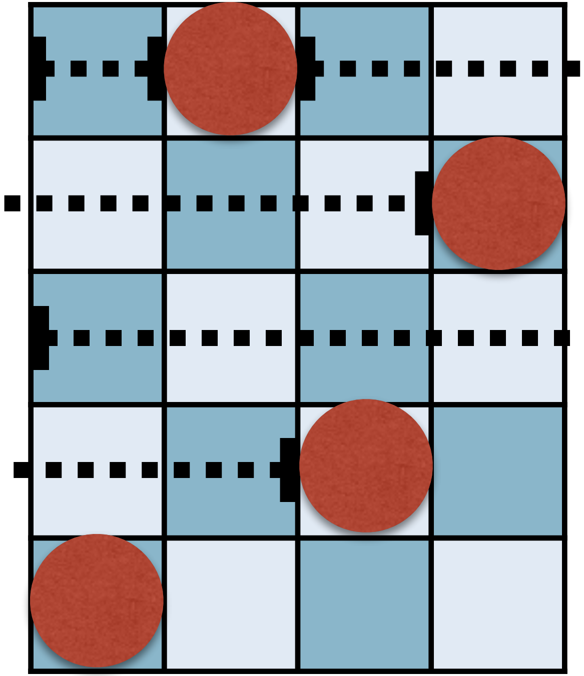
Combination: General Approach

Sum the number of ranks that were skipped for each of the spaces between pieces.



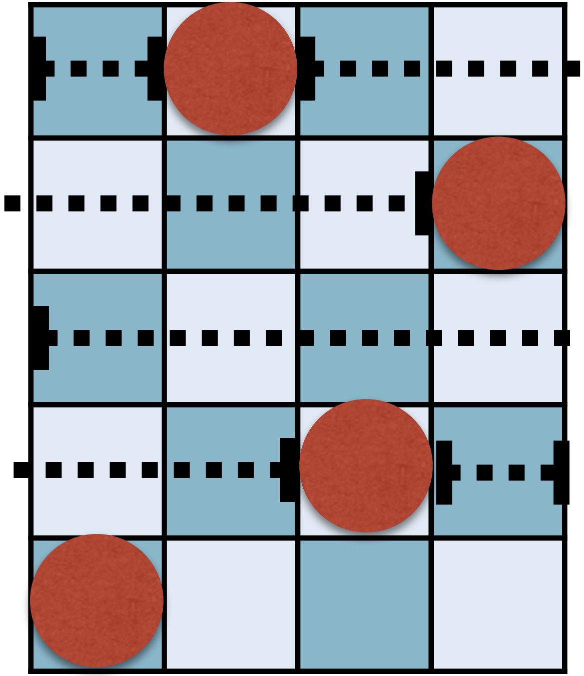
Combination: General Approach

Sum the number of ranks that were skipped for each of the spaces between pieces.



Combination: General Approach

Sum the number of ranks that were skipped for each of the spaces between pieces.



Ranking Combinations (Recursive)

```
uint64_t rank(int *pieces, int count, int spaces, int offset)
{
   if (count == 0)
      return 0;
   if (pieces[0]-offset == 0) // piece in first possible loc?
      return rank(&pieces[1], count-1, spaces-1, offset+1);
   uint64_t skipped = nchoosek(spaces-1, count-1);
   return skipped+rank(pieces, count, spaces-1, offset+1);
}
```

Running time: Linear in board size

Ranking Combinations (Recursive)

```
uint64_t rank(int *pieces, int count, int spaces, int offset)
{
   if (count == 0)
      return 0;
   if (pieces[0]-offset == 0) // piece in first possible loc?
      return rank(&pieces[1], count-1, spaces-1, offset+1);
   uint64_t skipped = nchoosek(spaces-1, count-1);
   return skipped+rank(pieces, count, spaces-1, offset+1);
}
```

Running time: Linear in board size

Ranking Combinations (Recursive)

```
uint64_t rank(int *pieces, int count, int spaces, int offset)
{
   if (count == 0)
      return 0;
   if (pieces[0]-offset == 0) // piece in first possible loc?
      return rank(&pieces[1], count-1, spaces-1, offset+1);
   uint64_t skipped = nchoosek(spaces-1, count-1);
   return skipped+rank(pieces, count, spaces-1, offset+1);
}
```

Running time: Linear in board size

Ranking Combinations (Recursive)

```
uint64_t rank(int *pieces, int count, int spaces, int offset)
{
   if (count == 0)
      return 0;
   if (pieces[0]-offset == 0) // piece in first possible loc?
      return rank(&pieces[1], count-1, spaces-1, offset+1);
   uint64_t skipped = nchoosek(spaces-1, count-1);
   return skipped+rank(pieces, count, spaces-1, offset+1);
}
```

Running time: Linear in board size

Ranking Combinations (Recursive)

```
uint64_t rank(int *pieces, int count, int spaces, int offset)
{
   if (count == 0)
      return 0;
   if (pieces[0]-offset == 0) // piece in first possible loc?
      return rank(&pieces[1], count-1, spaces-1, offset+1);
   uint64_t skipped = nchoosek(spaces-1, count-1);
   return skipped+rank(pieces, count, spaces-1, offset+1);
}
```

Running time: Linear in board size

Ranking Combinations (Recursive)

```
uint64_t rank(int *pieces, int count, int spaces, int offset)
{
   if (count == 0)
      return 0;
   if (pieces[0]-offset == 0) // piece in first possible loc?
      return rank(&pieces[1], count-1, spaces-1, offset+1);
   uint64_t skipped = nchoosek(spaces-1, count-1);
   return skipped+rank(pieces, count, spaces-1, offset+1);
}
```

Running time: Linear in board size

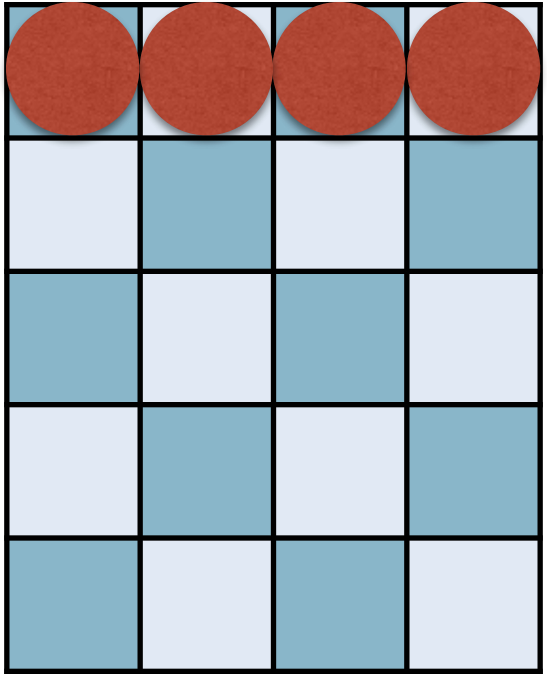
Running time: Linear in # of pieces

Definition

- **Unranking:** A function that takes an integer between 0…N-1 and returns the associated combination.

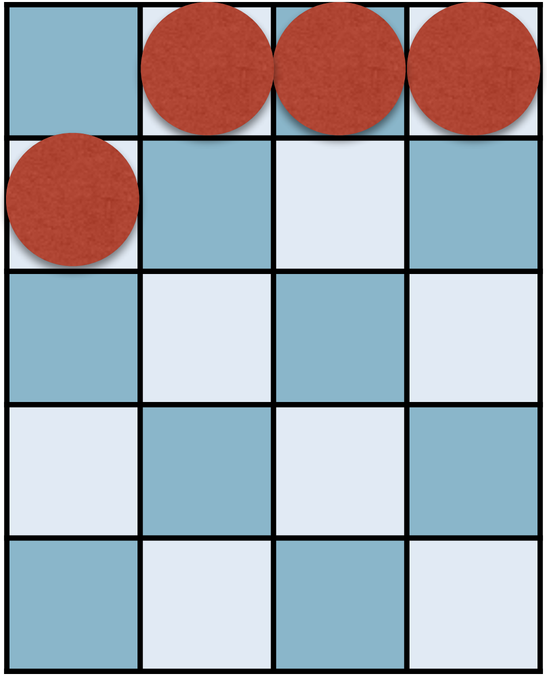
Combination: Ranking

Rank: 0



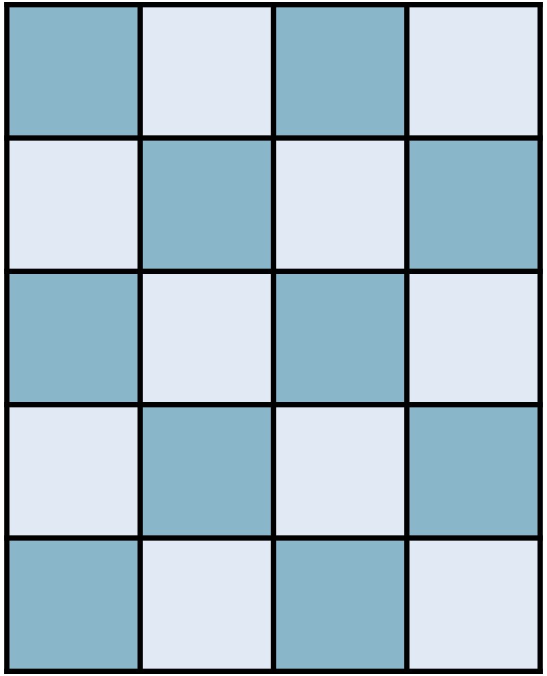
Combination: Ranking

Rank: 969



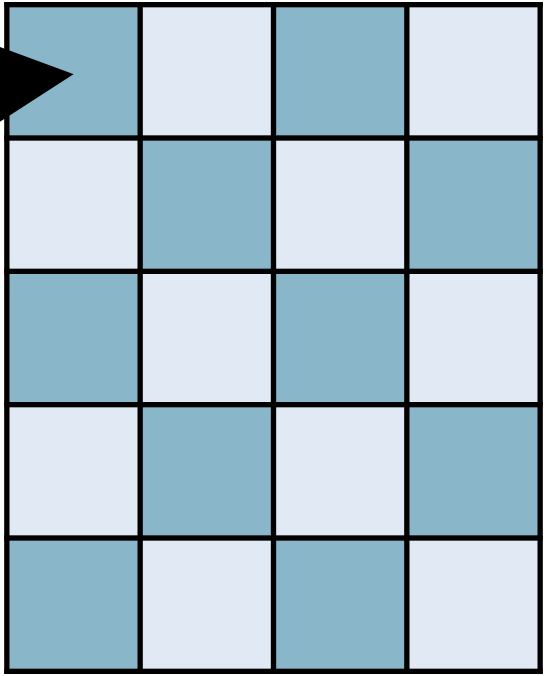
Combination: Unrank 803

Rank: 803



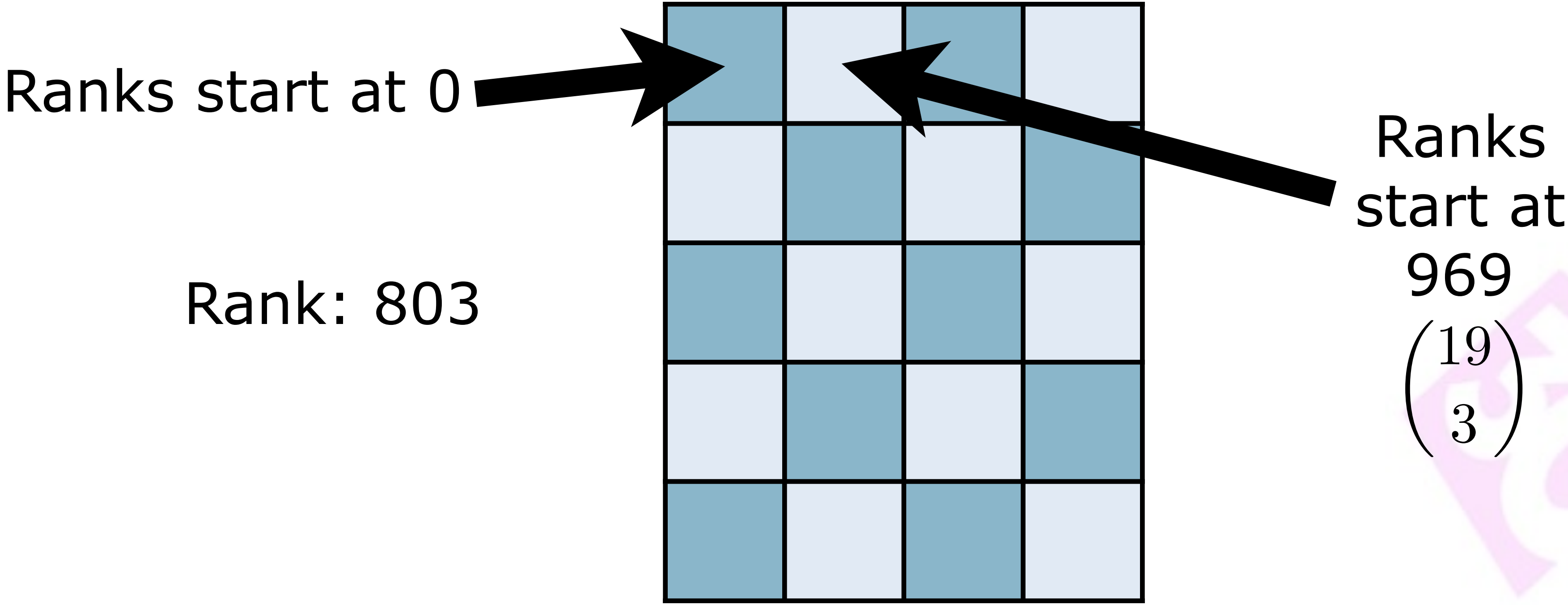
Combination: Unrank 803

Ranks start at 0



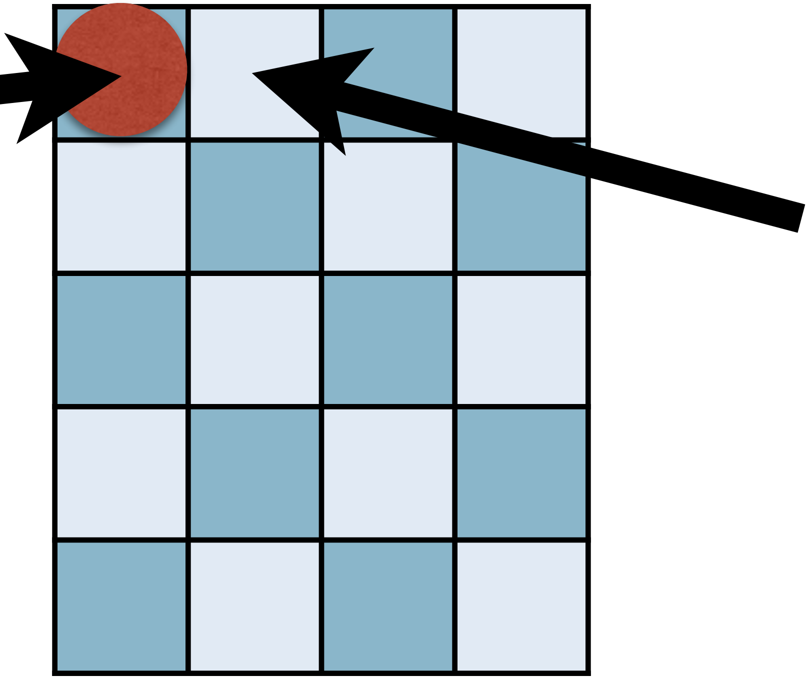
Rank: 803

Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

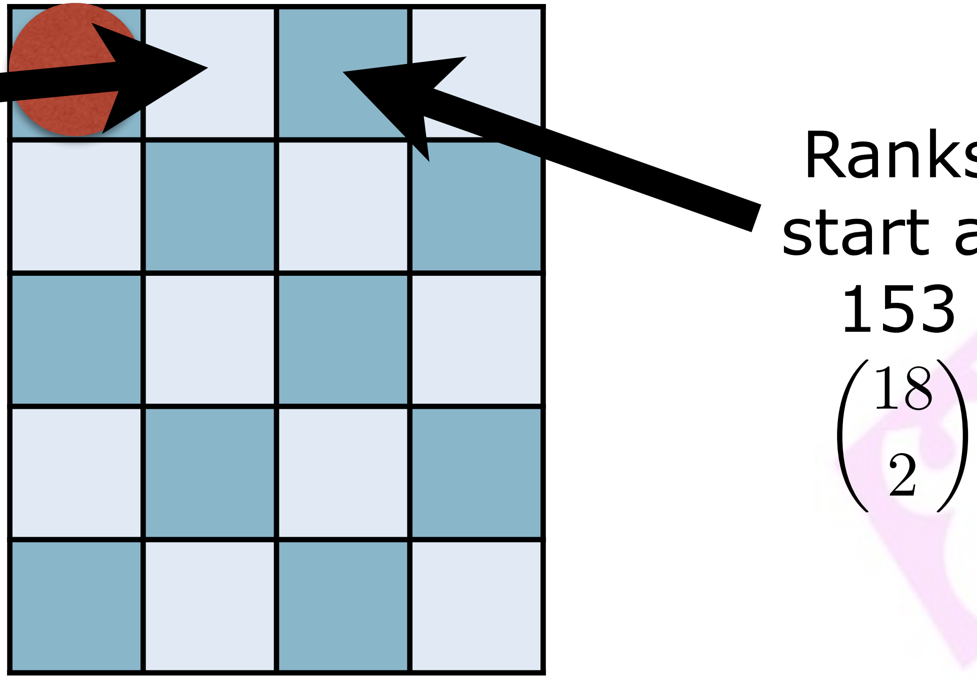


Ranks start at 969

$$\binom{19}{3}$$

Rank: 803

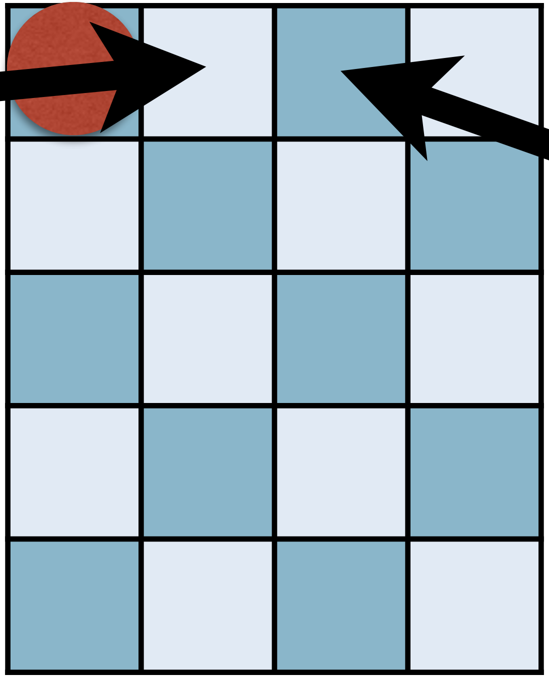
Combination: Unrank 803



Rank: 803

Combination: Unrank 803

Ranks start at 0

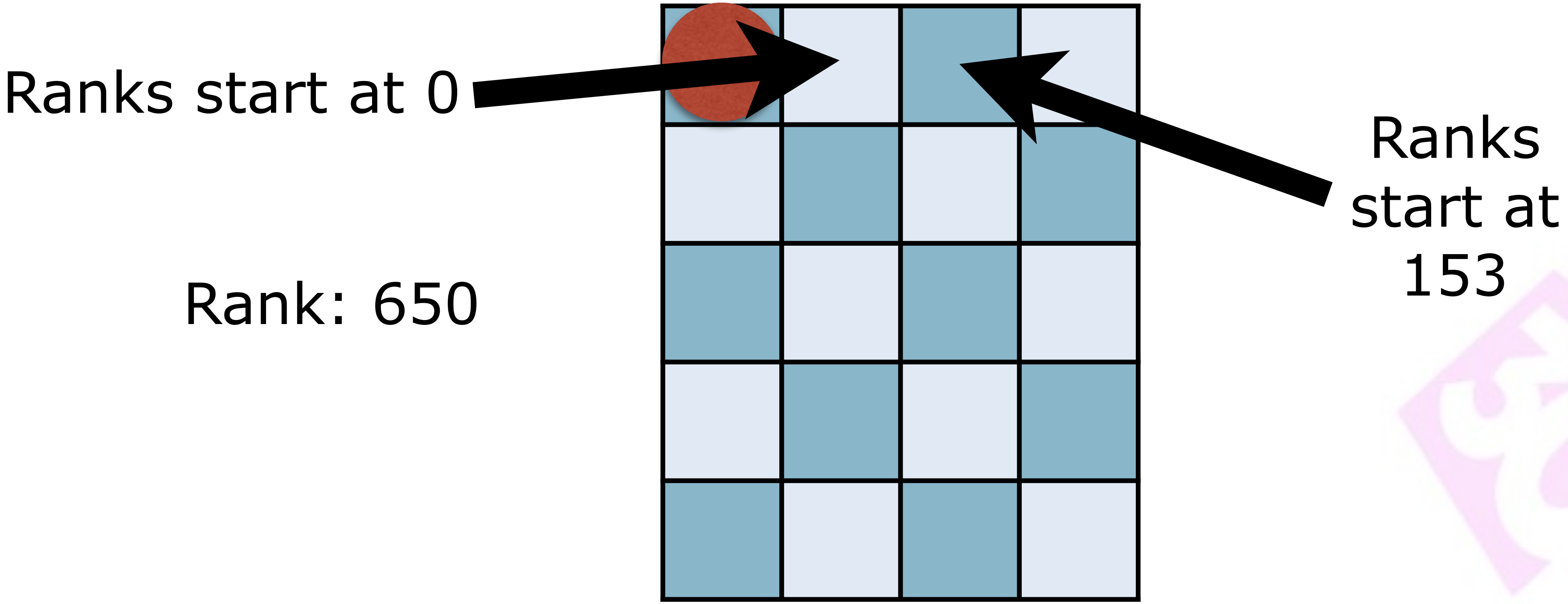


Ranks start at 153

$$\binom{18}{2}$$

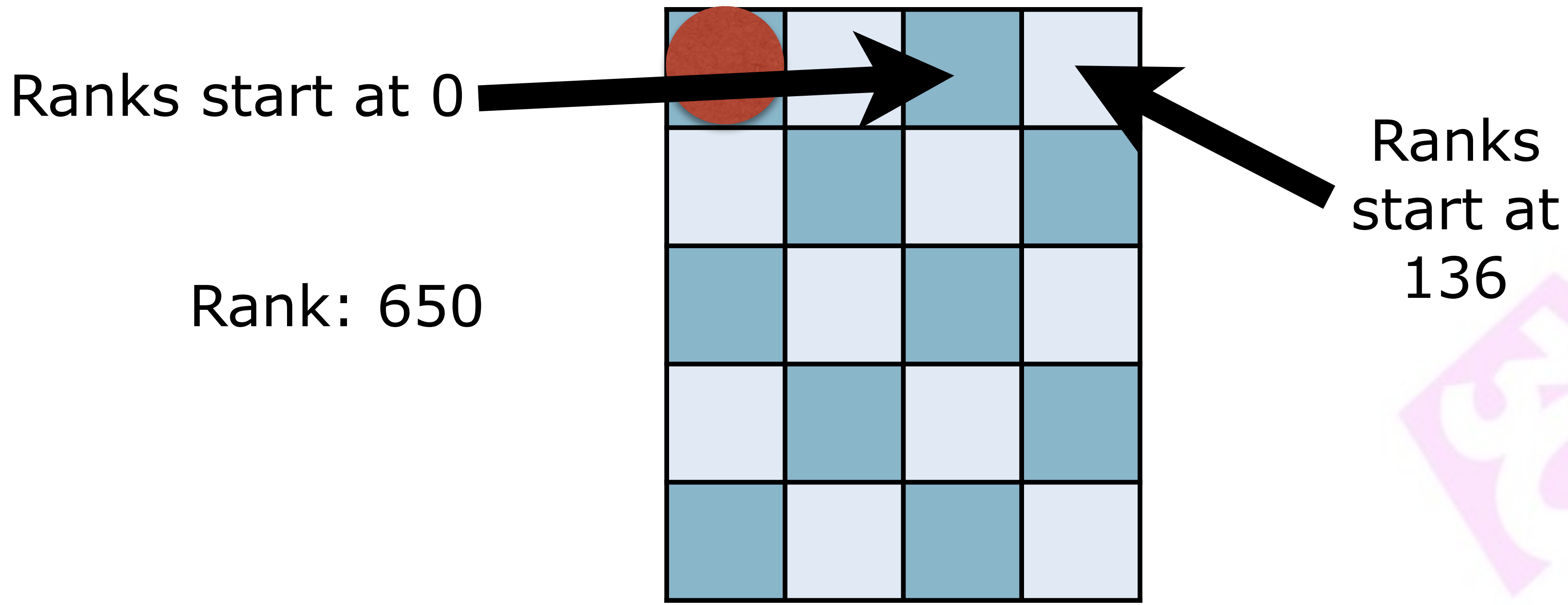
Rank: 803-153

Combination: Unrank 803

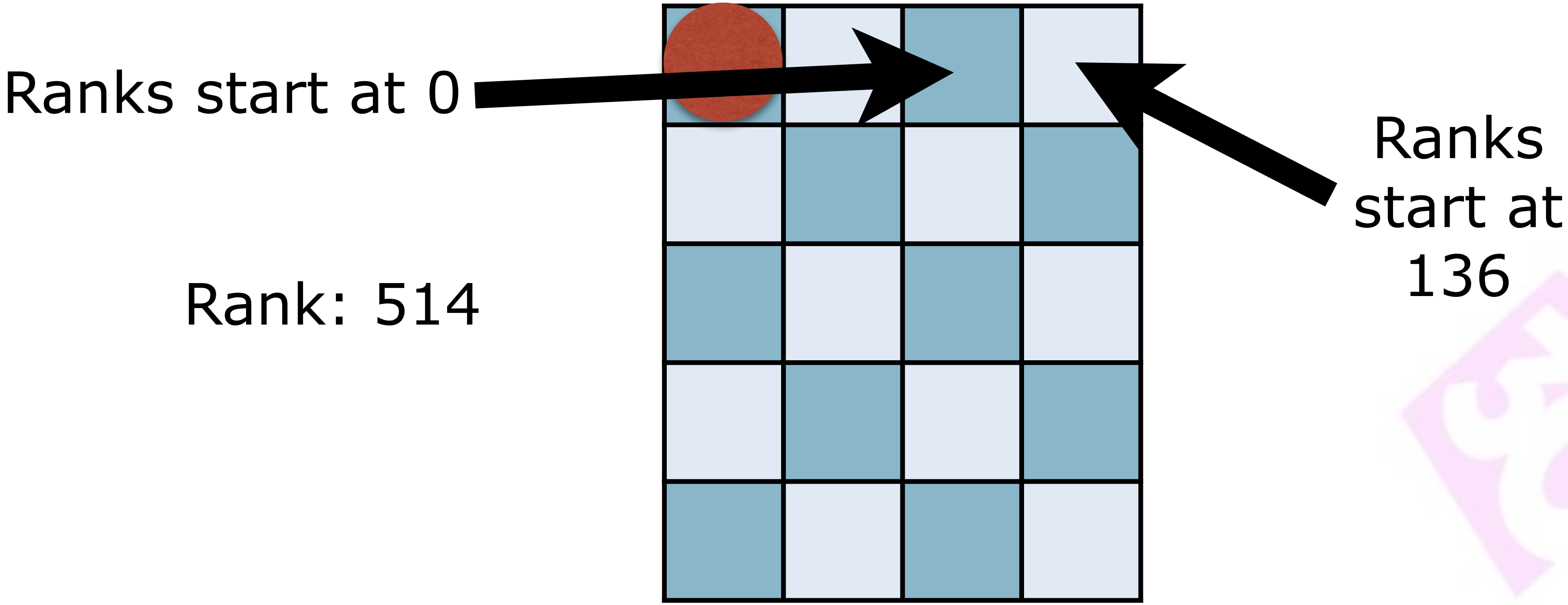


Rank: 650

Combination: Unrank 803



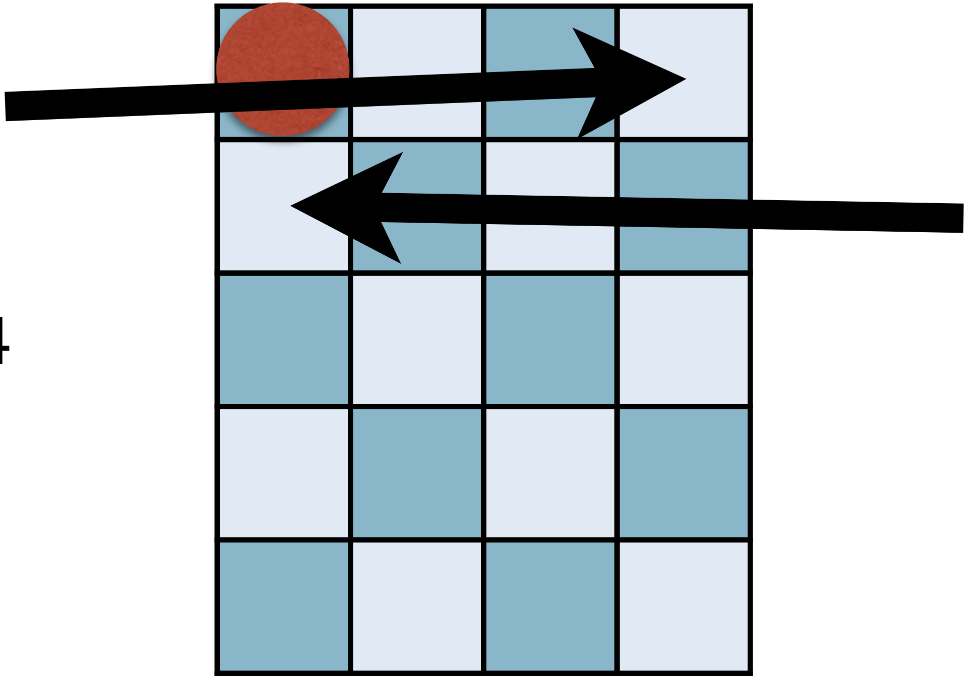
Combination: Unrank 803



Rank: 514

Combination: Unrank 803

Ranks start at 0

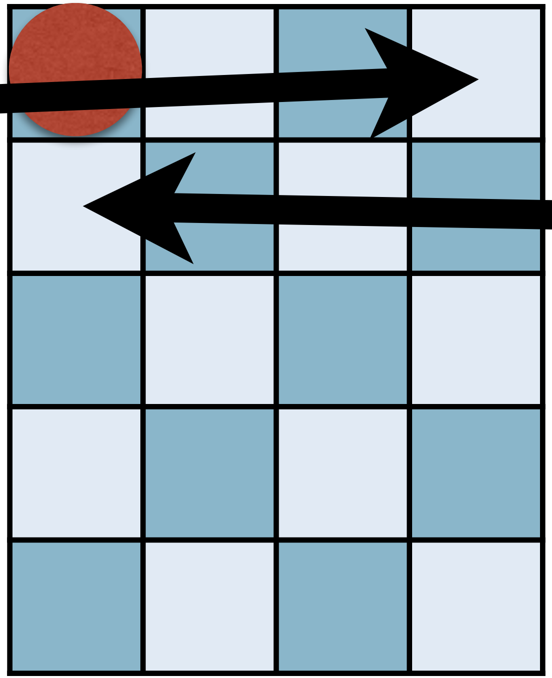


Ranks start at 120

Rank: 514

Combination: Unrank 803

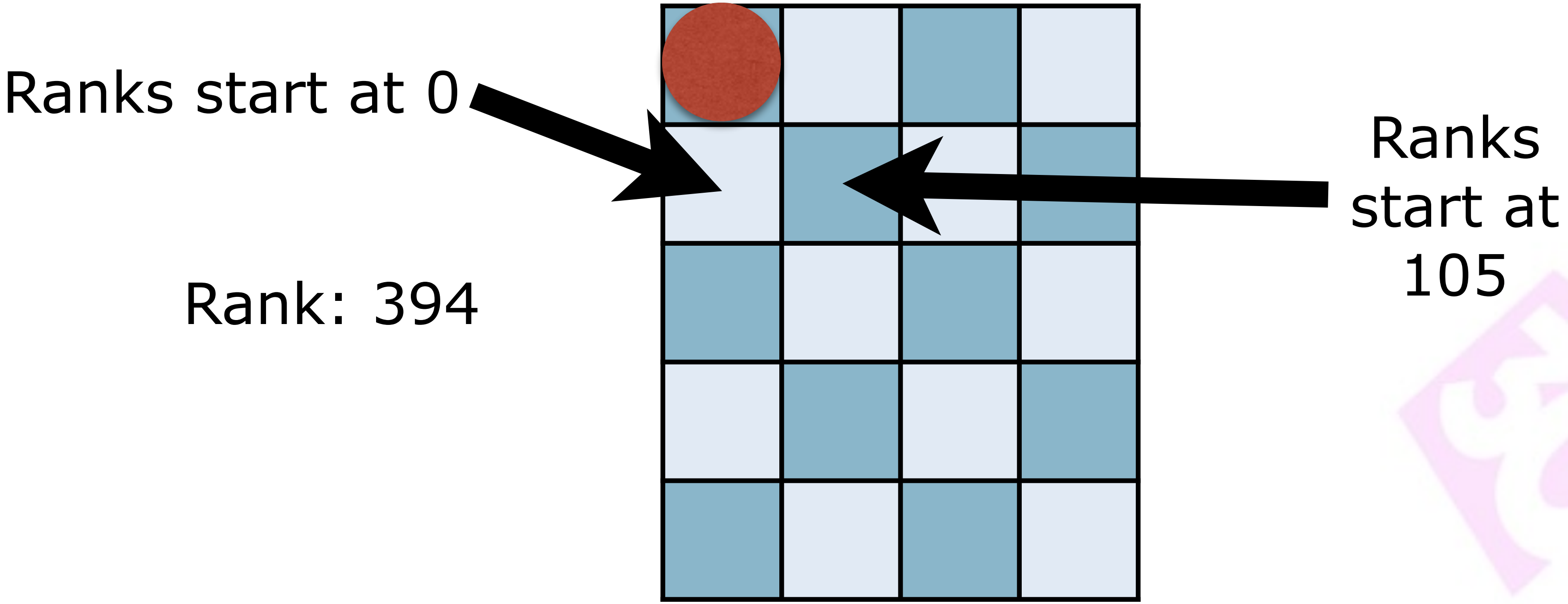
Ranks start at 0



Ranks start at 120

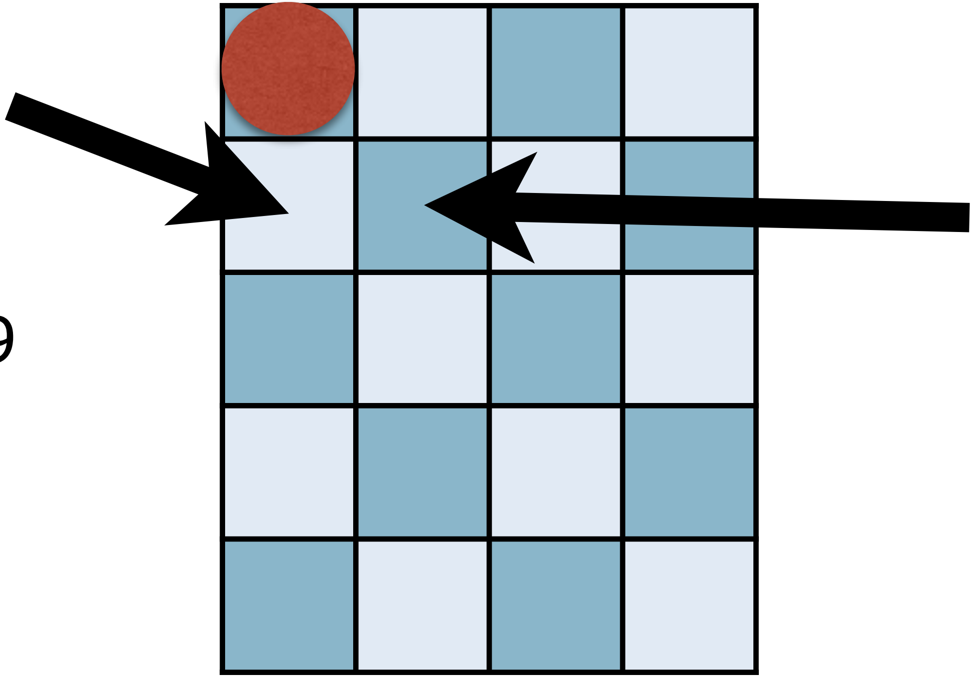
Rank: 394

Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

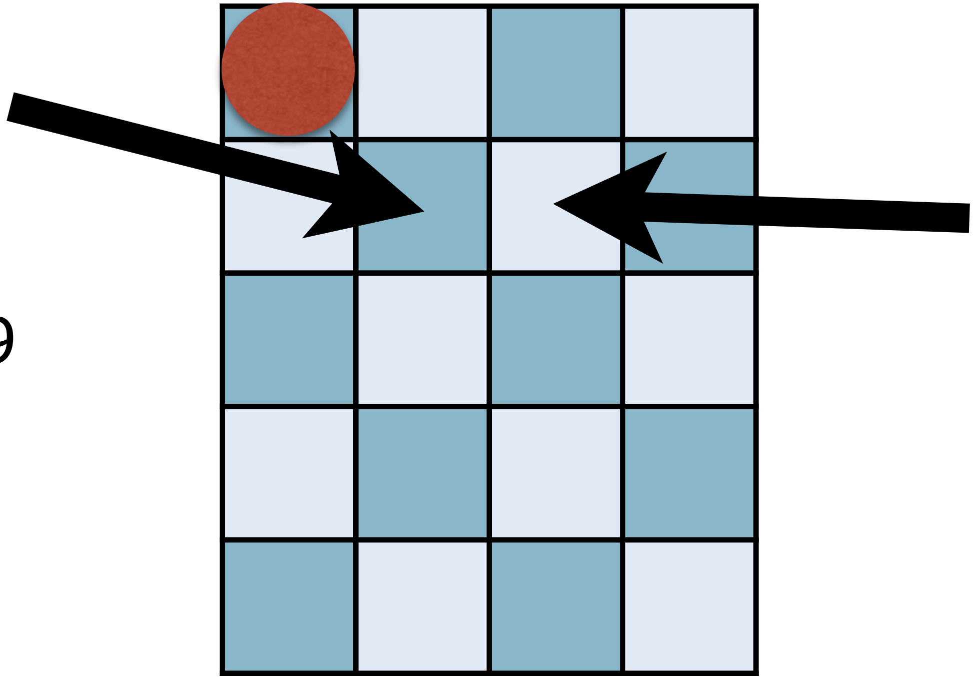


Ranks start at 105

Rank: 289

Combination: Unrank 803

Ranks start at 0

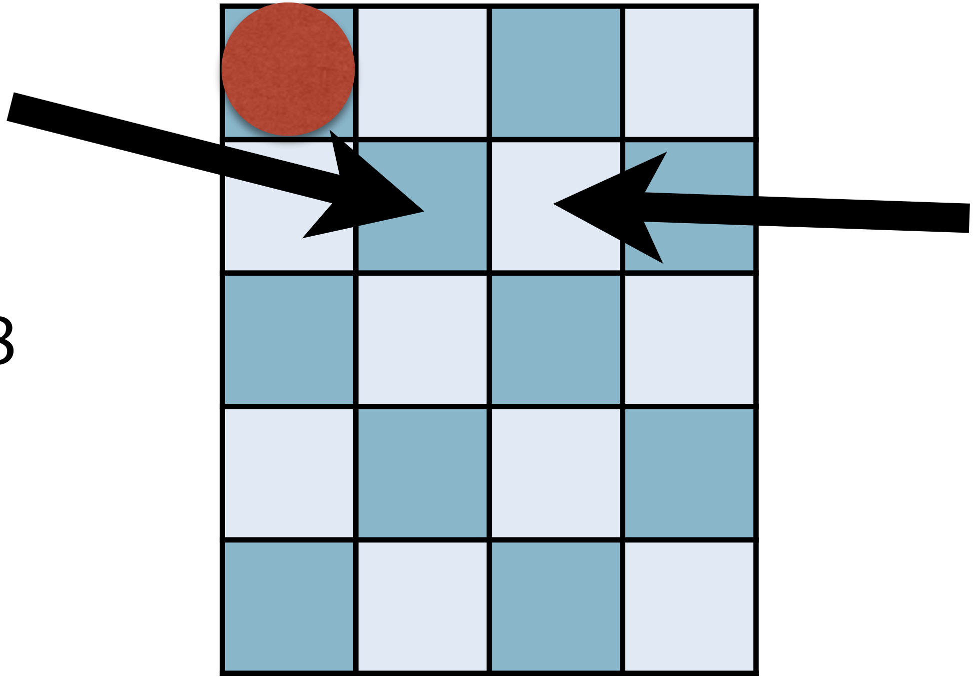


Ranks start at 91

Rank: 289

Combination: Unrank 803

Ranks start at 0

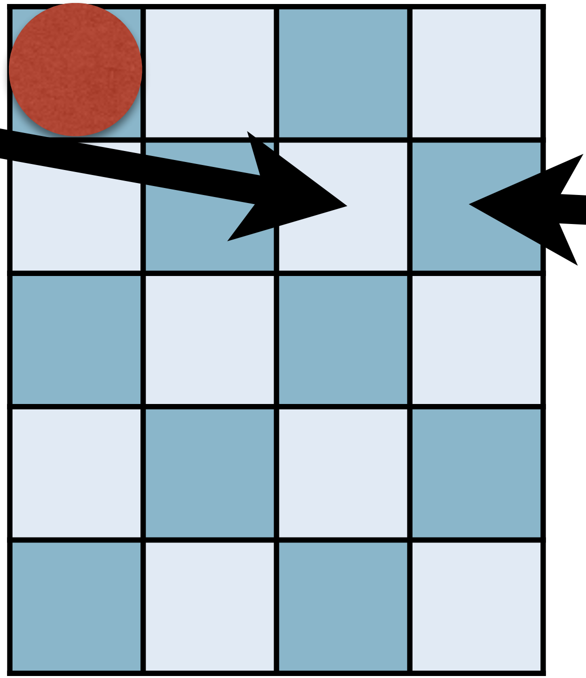


Ranks start at 91

Rank: 198

Combination: Unrank 803

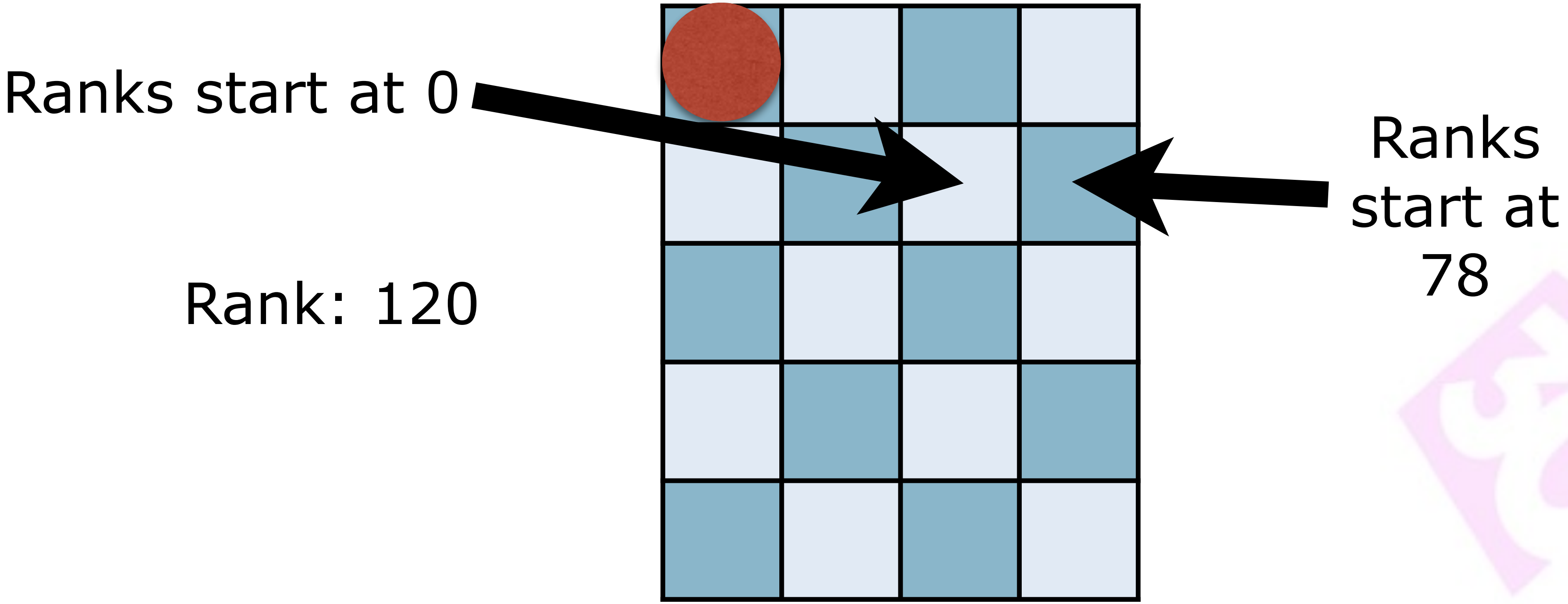
Ranks start at 0



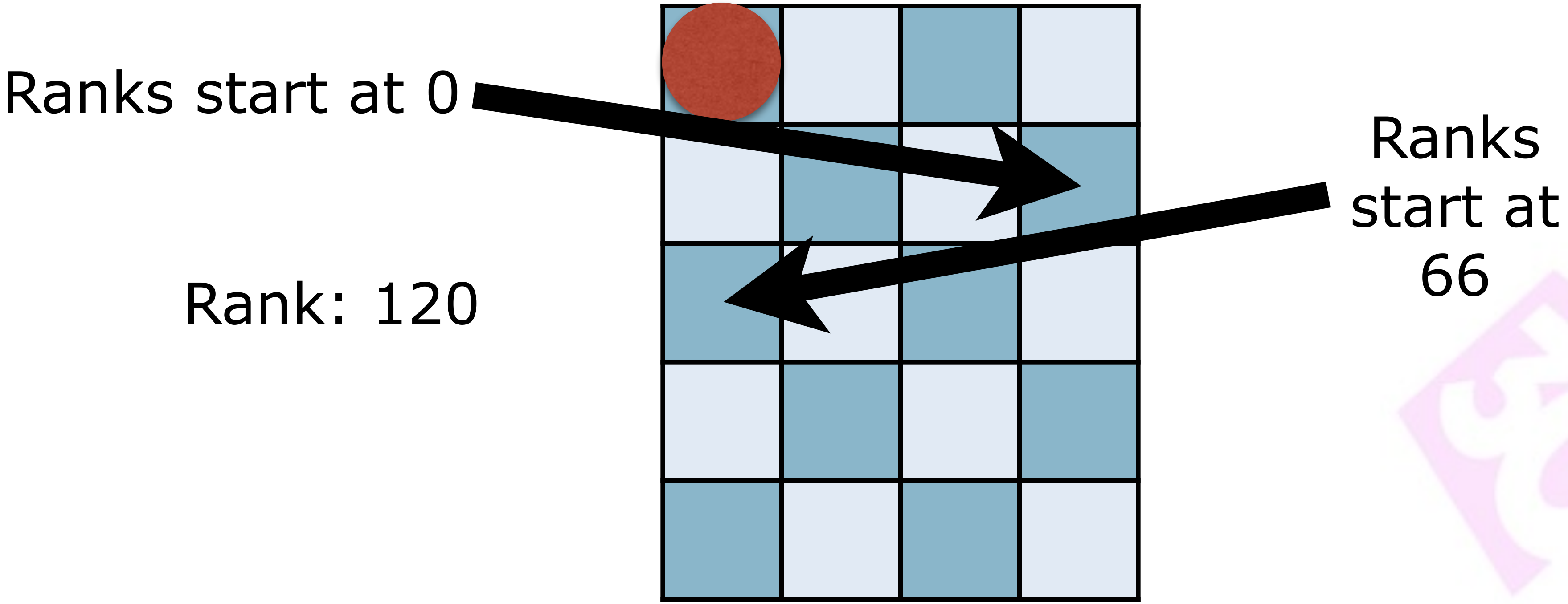
Ranks start at 78

Rank: 198

Combination: Unrank 803

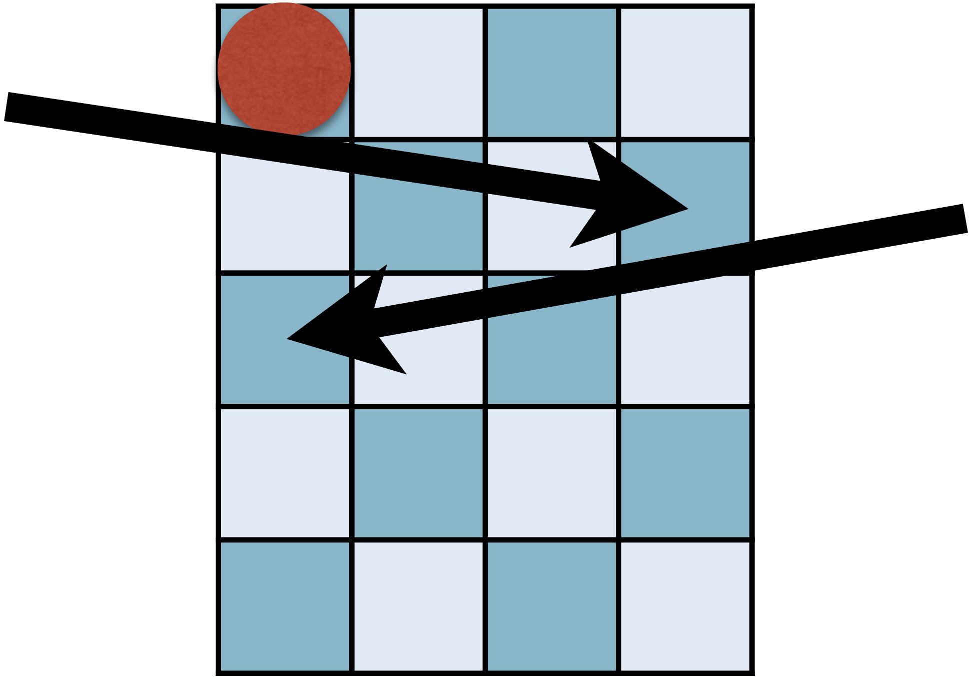


Combination: Unrank 803



Combination: Unrank 803

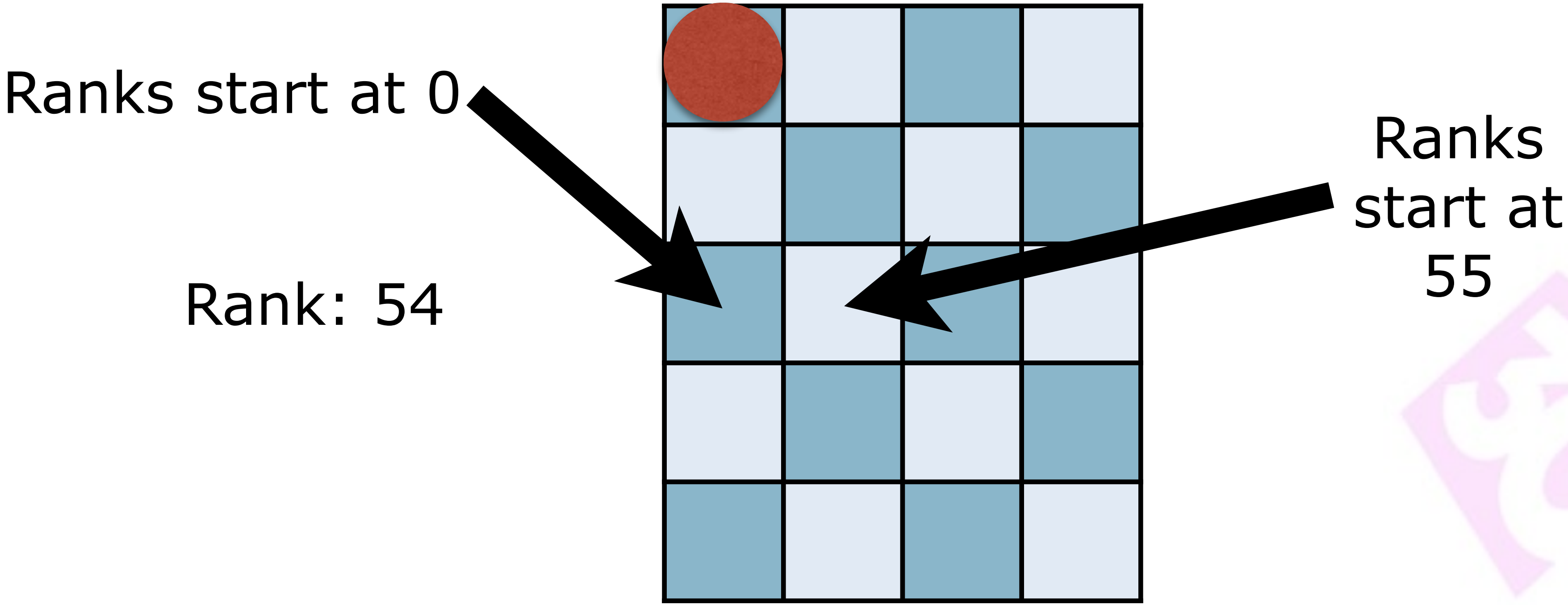
Ranks start at 0



Ranks start at 66

Rank: 54

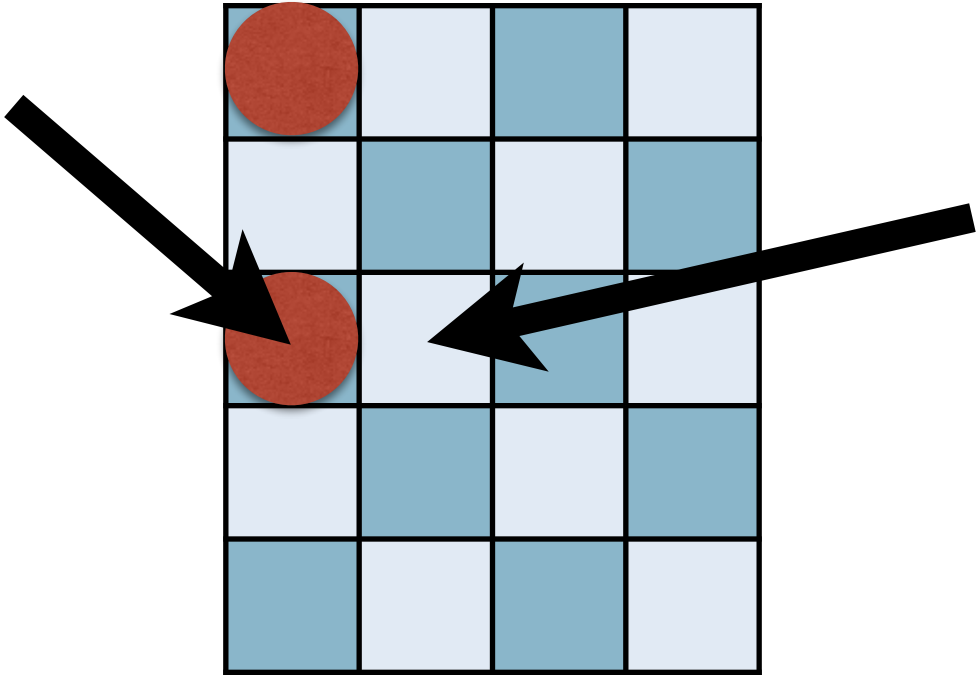
Combination: Unrank 803



Combination: Unrank 803

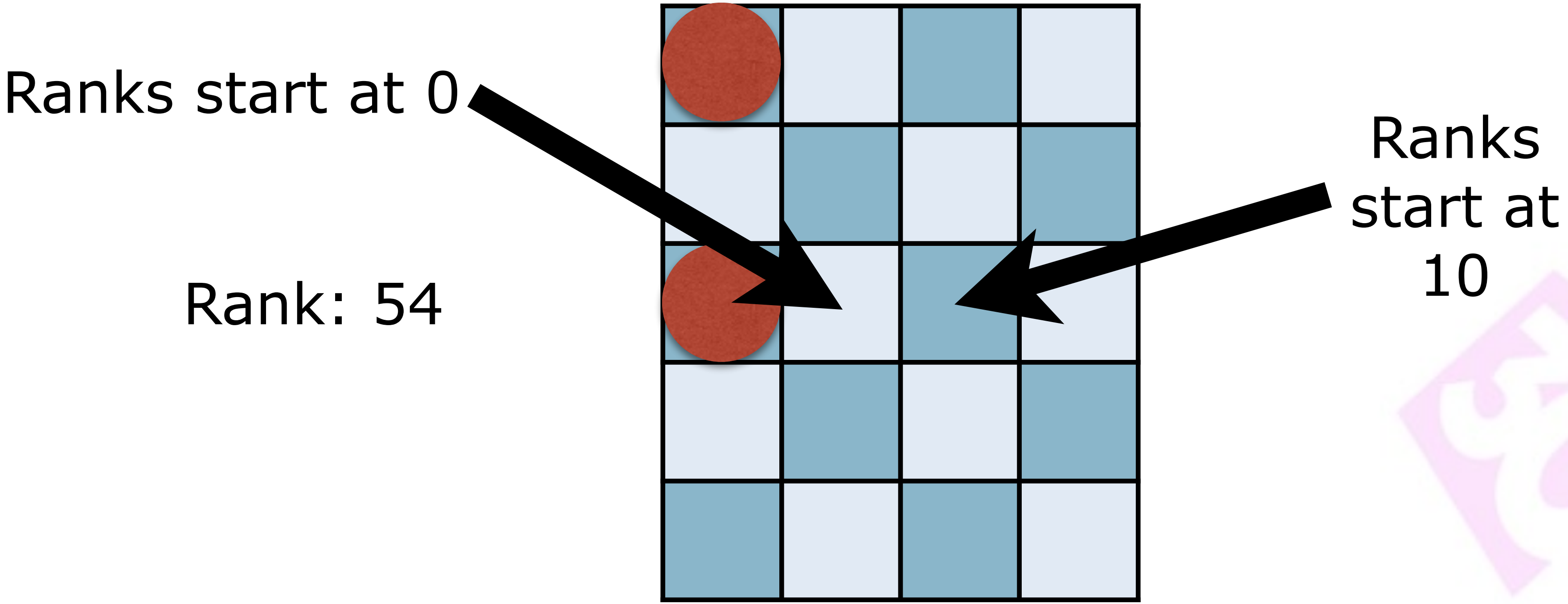
Ranks start at 0

Rank: 54

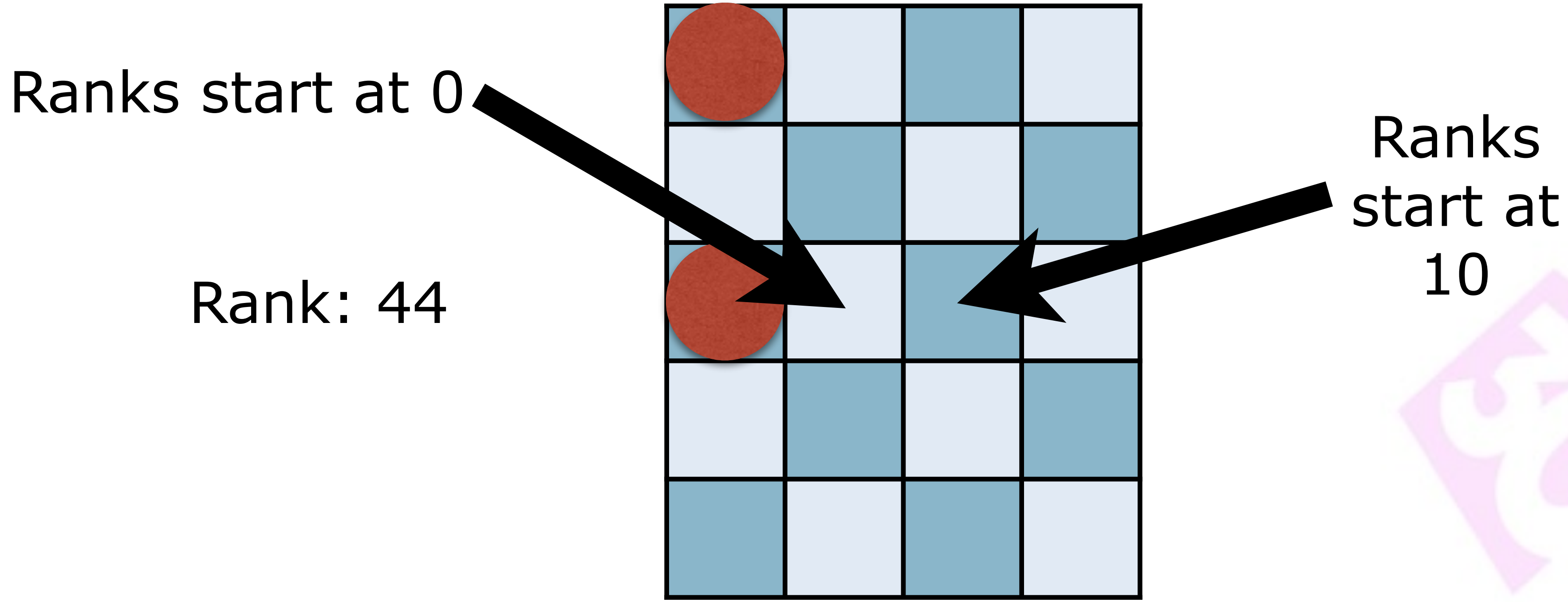


Ranks start at 55

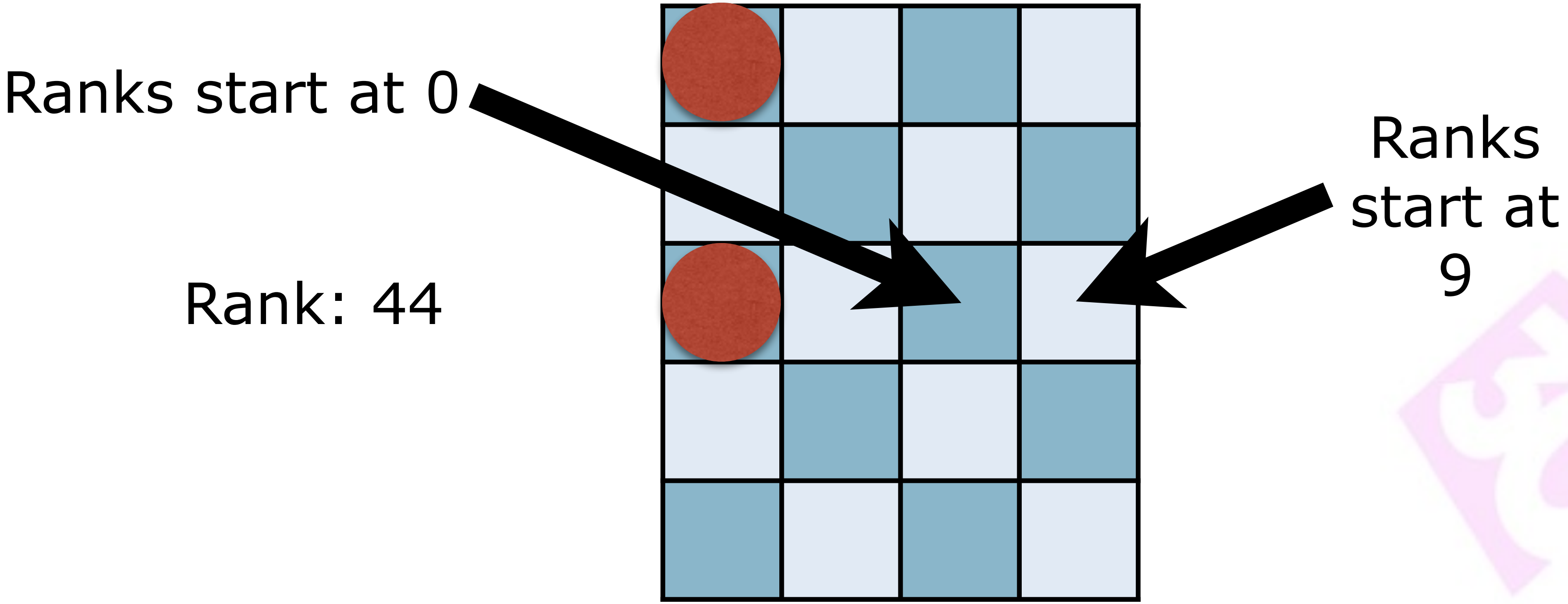
Combination: Unrank 803



Combination: Unrank 803



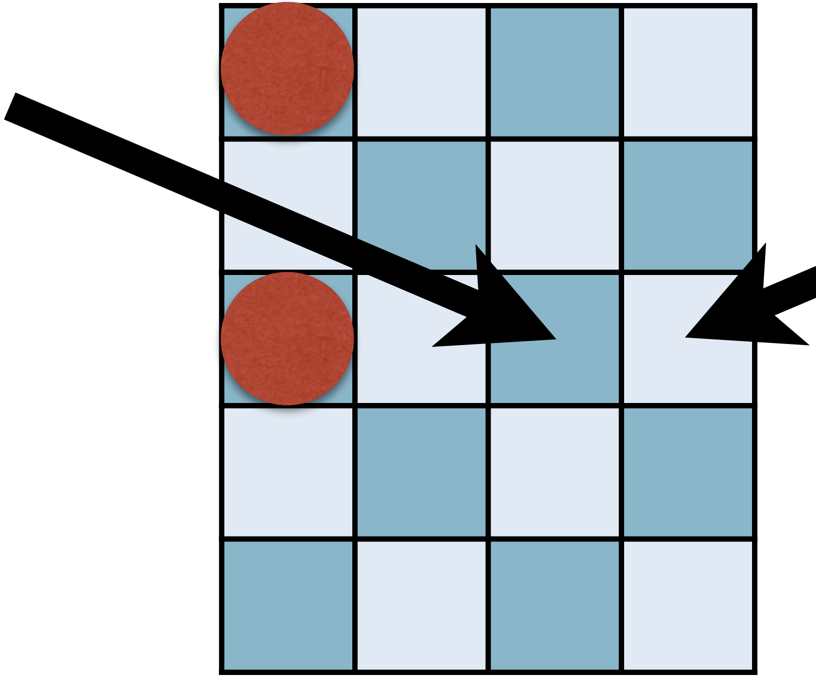
Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

Rank: 35

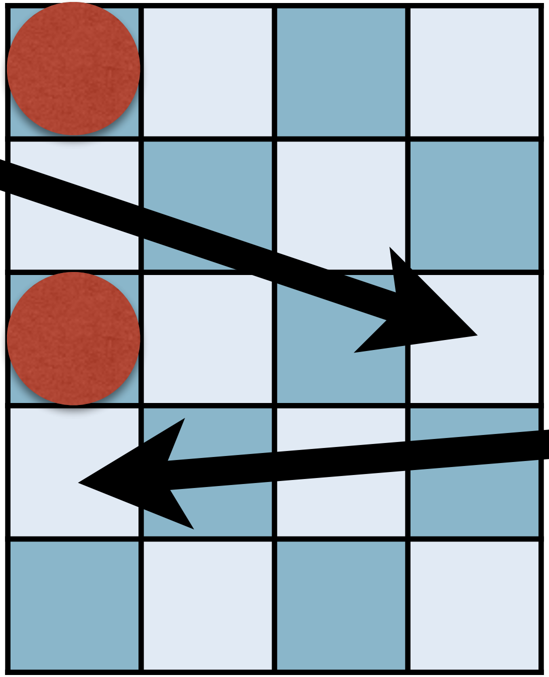


Ranks
start at
9

Combination: Unrank 803

Ranks start at 0

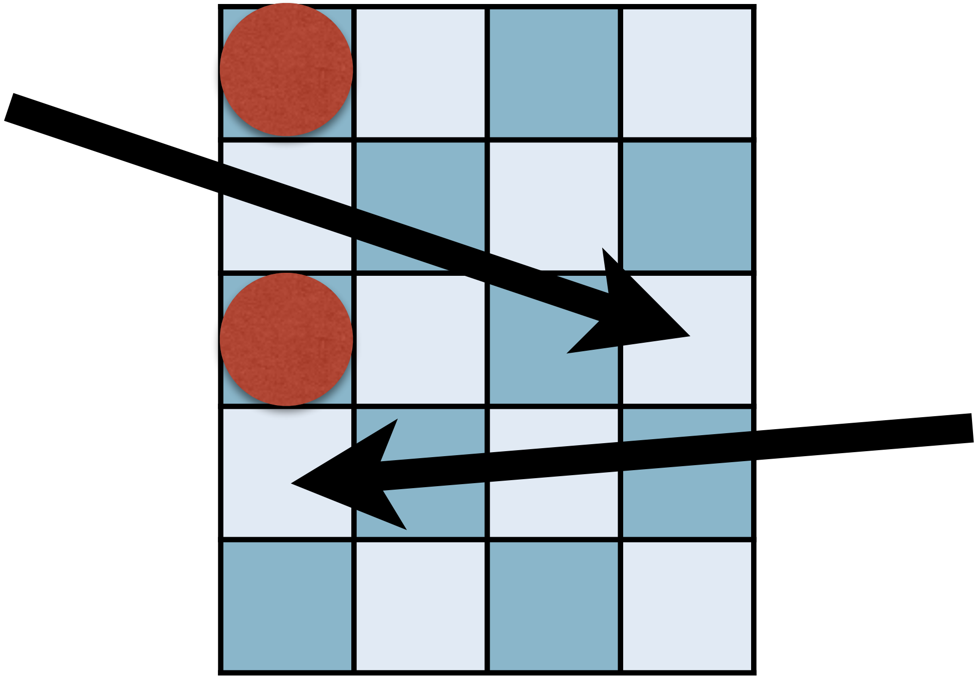
Rank: 35



Ranks start at 8

Combination: Unrank 803

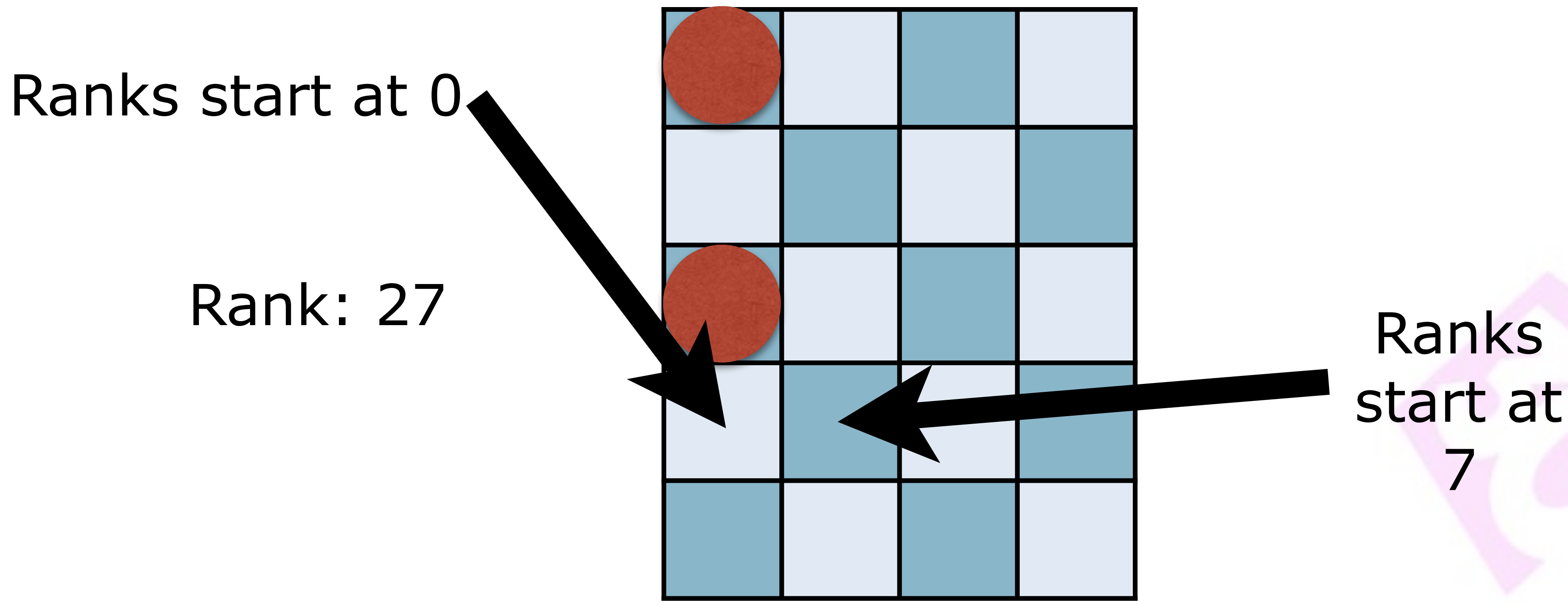
Ranks start at 0



Rank: 27

Ranks start at 8

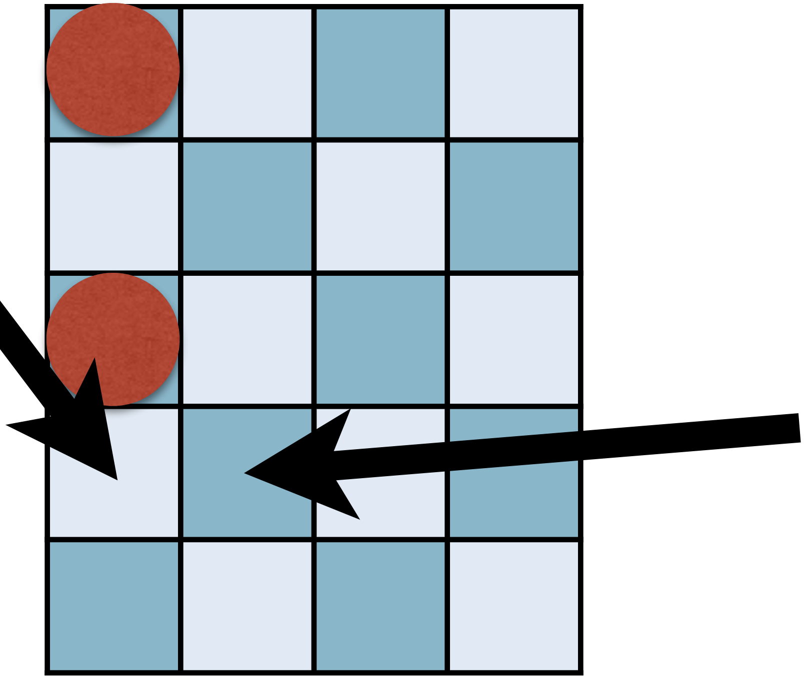
Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

Rank: 20

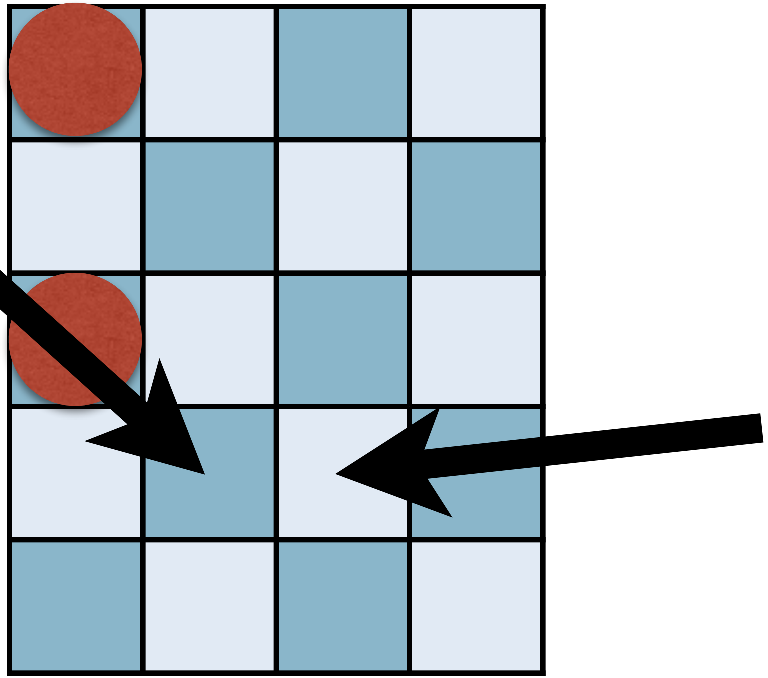


Ranks start at 7

Combination: Unrank 803

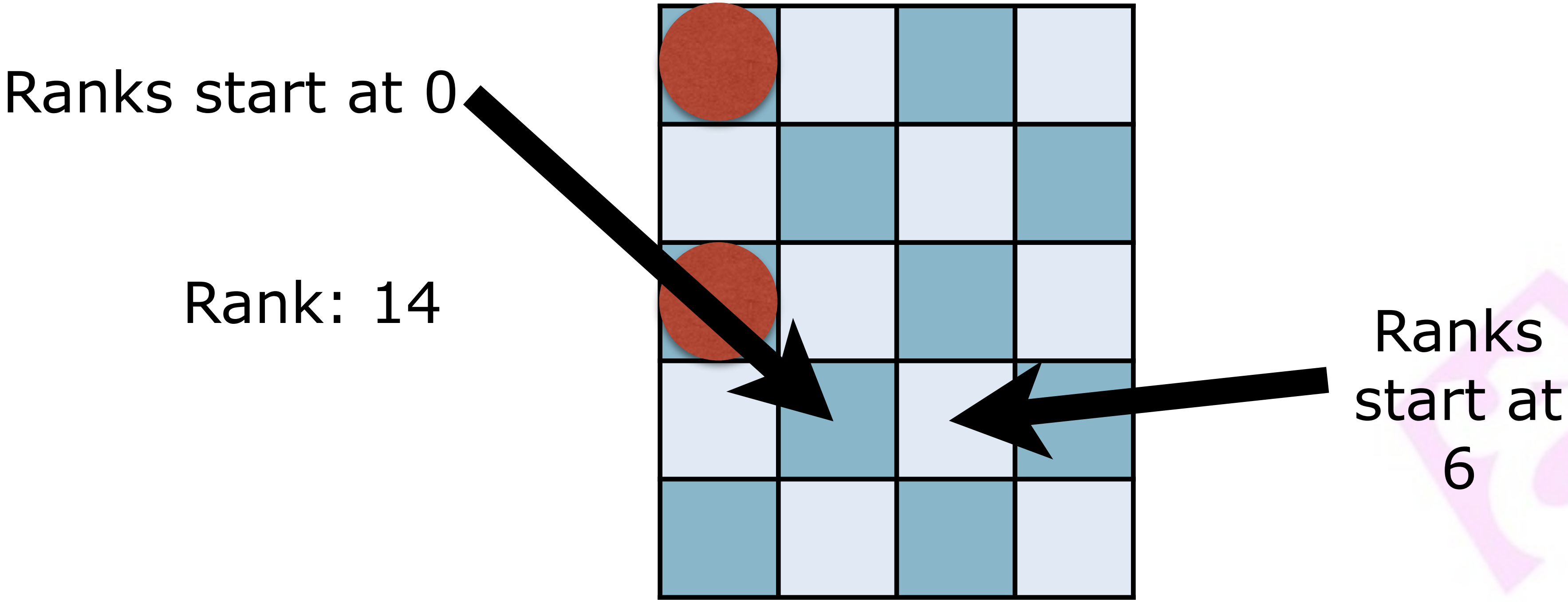
Ranks start at 0

Rank: 20



Ranks start at 6

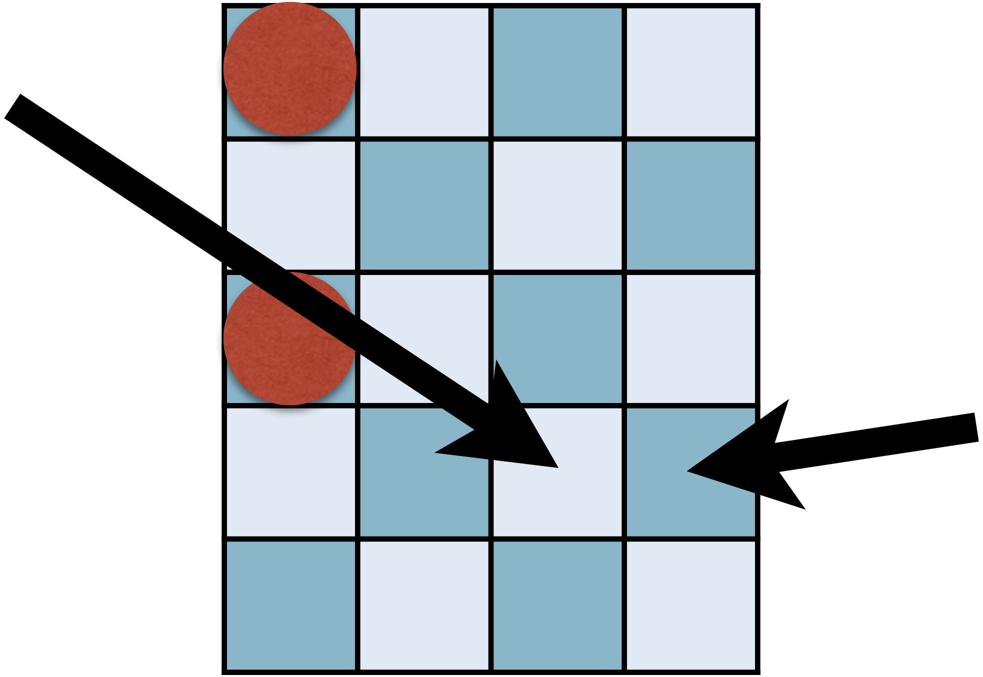
Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

Rank: 14

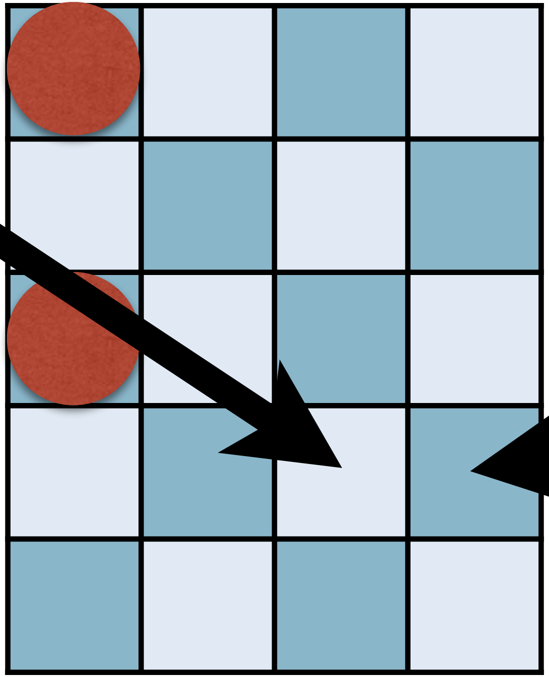


Ranks start at 5

Combination: Unrank 803

Ranks start at 0

Rank: 9

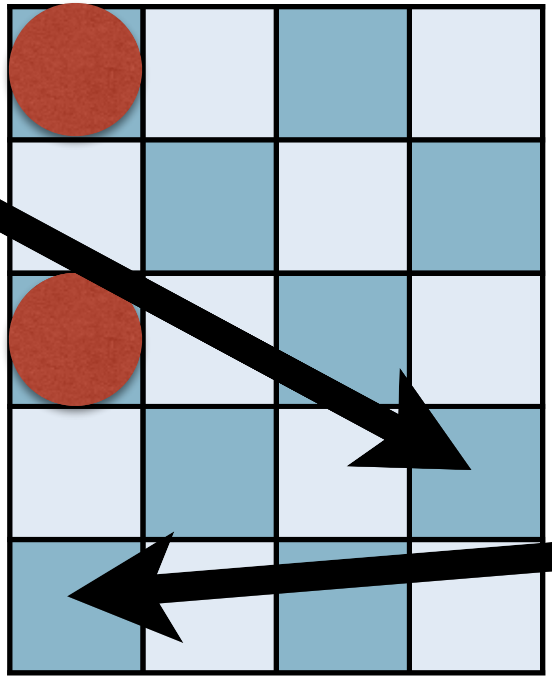


Ranks start at 5

Combination: Unrank 803

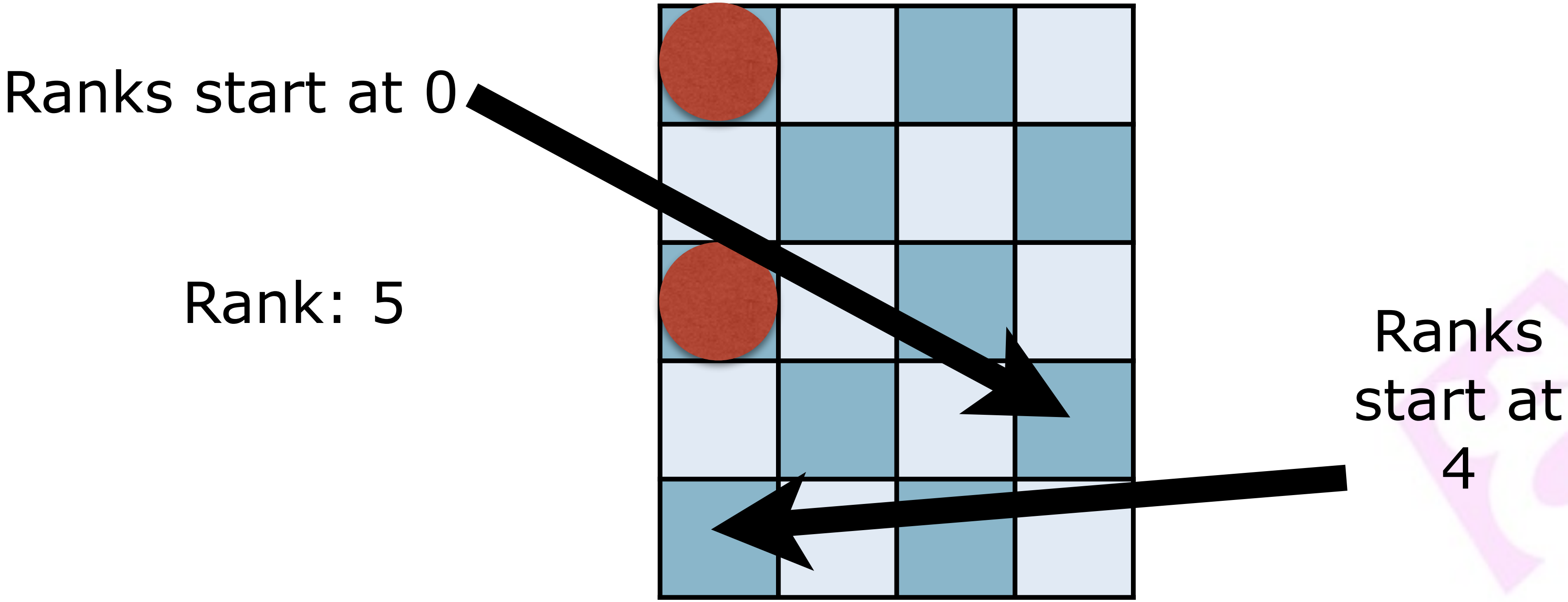
Ranks start at 0

Rank: 9



Ranks start at 4

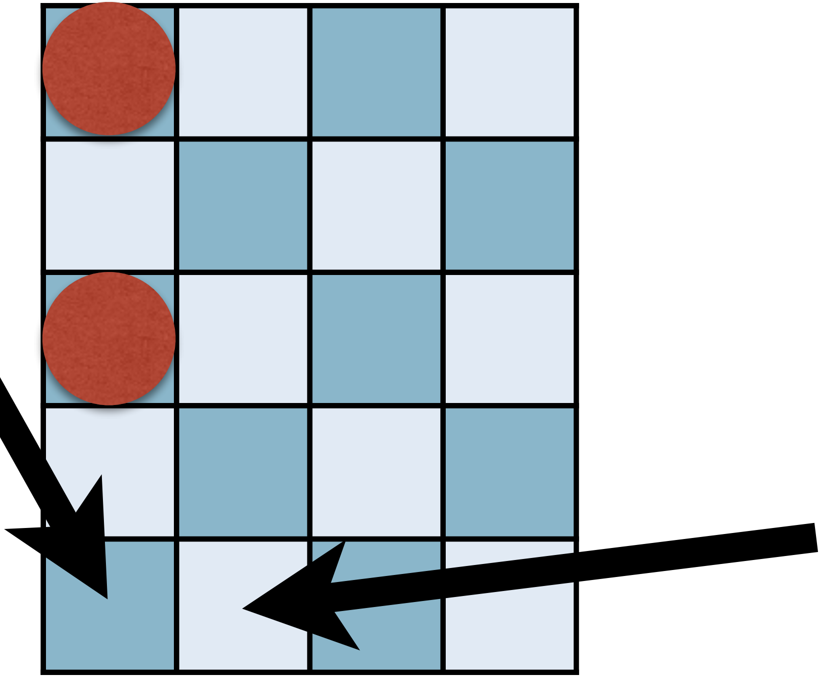
Combination: Unrank 803



Combination: Unrank 803

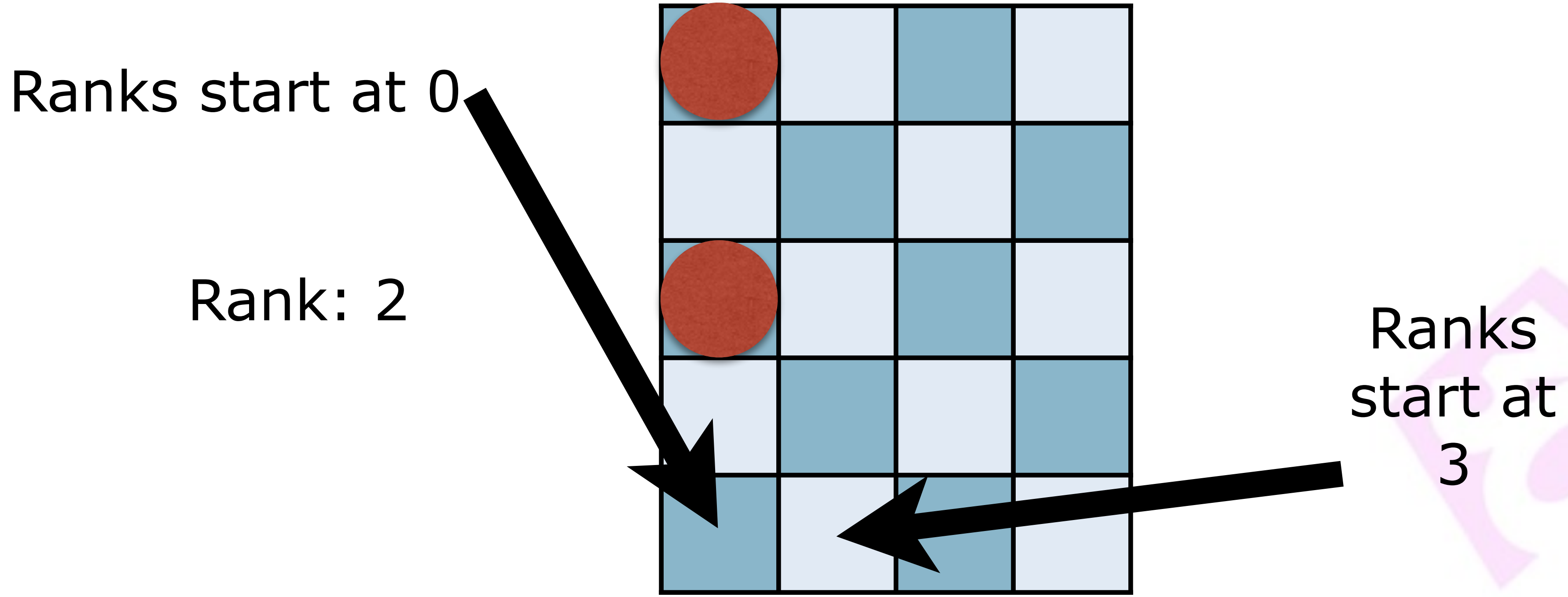
Ranks start at 0

Rank: 5



Ranks start at 3

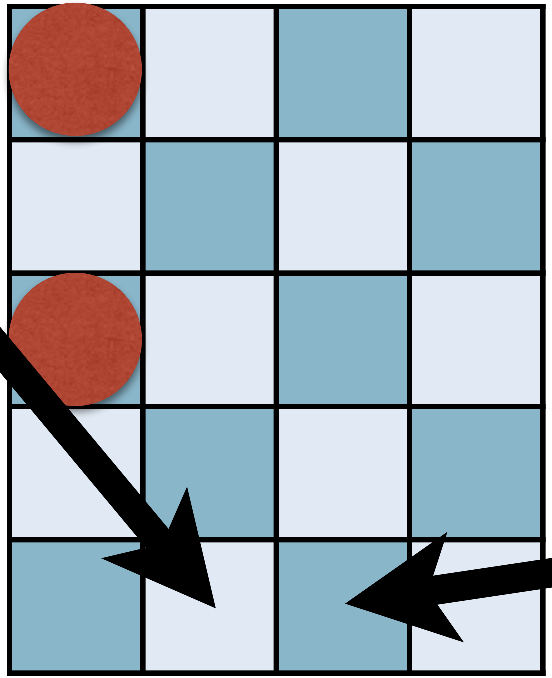
Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

Rank: 2

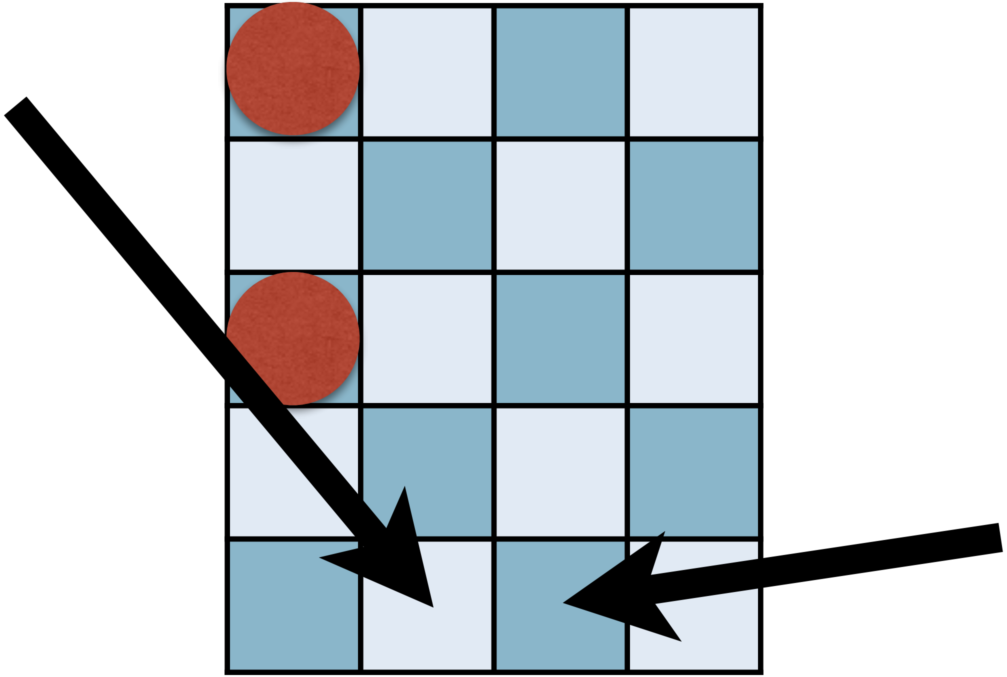


Ranks start at 2

Combination: Unrank 803

Ranks start at 0

Rank: 0

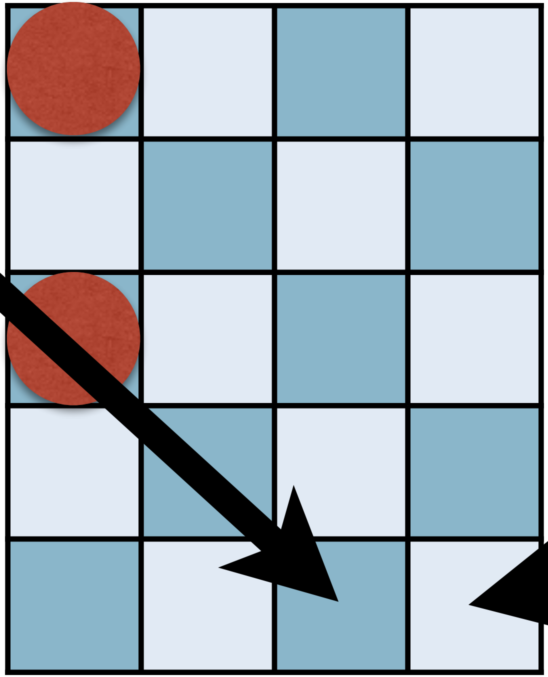


Ranks start at 2

Combination: Unrank 803

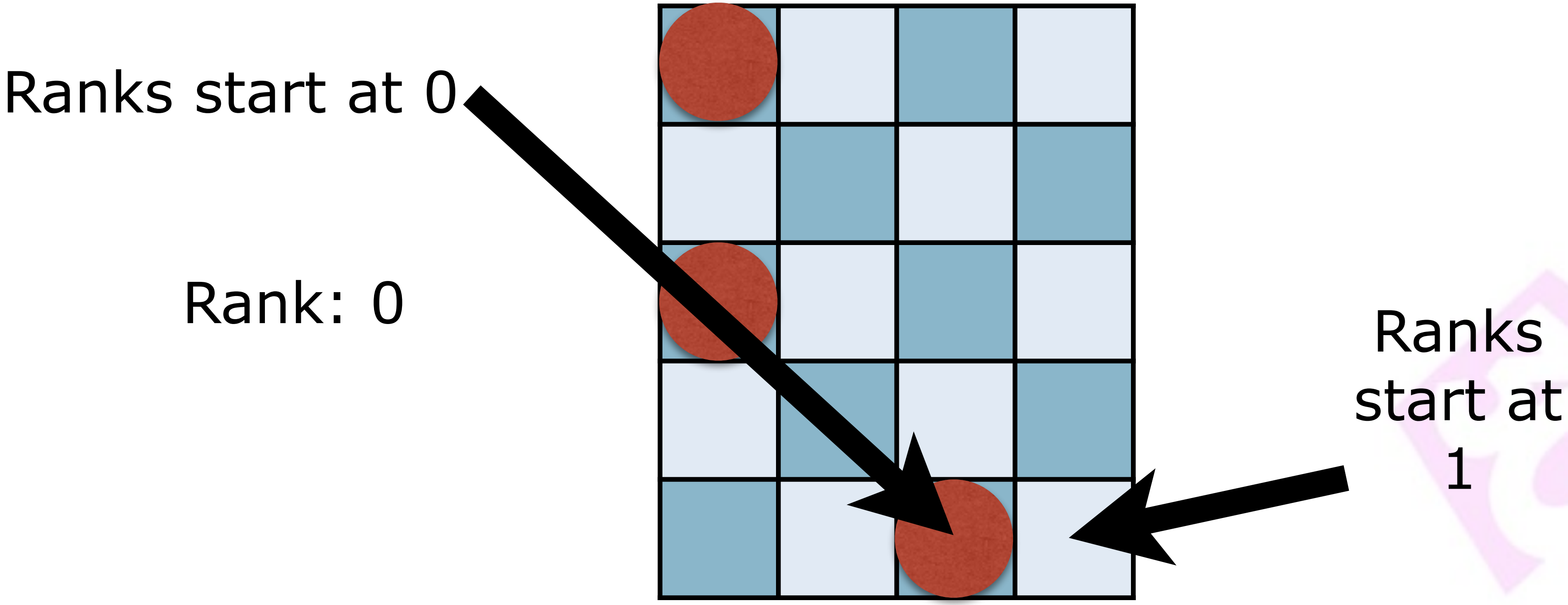
Ranks start at 0

Rank: 0



Ranks start at 1

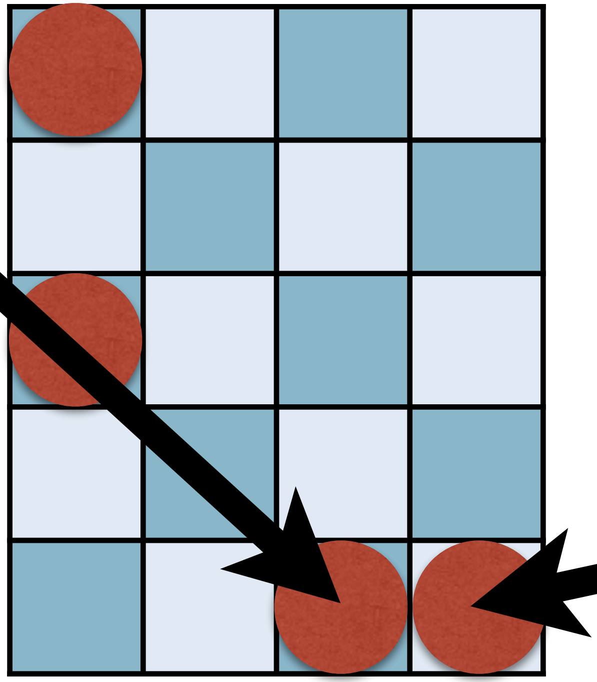
Combination: Unrank 803



Combination: Unrank 803

Ranks start at 0

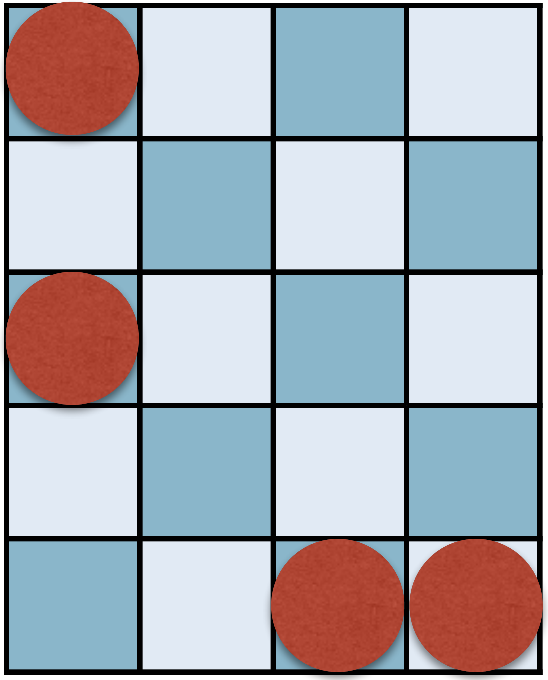
Rank: 0



Ranks start at 1

Combination: Unrank 803

Rank: 803



Unranking function

```
void unrank(uint64_t rank, int *pieces, int count, int spaces, int total)
{
   if (count == 0)
      return;
   uint64_t skipped = nchoosek(spaces-1, count-1);
   if (rank >= skipped)
      unrank(rank-skipped, pieces, count, spaces-1, total);
   else {
      pieces[0] = total-spaces;
      unrank(rank, &pieces[1], count-1, spaces-1, total);
   }
}
```

Unranking function

```
void unrank(uint64_t rank, int *pieces, int count, int spaces, int total)
{
    if (count == 0)
        return;
    uint64_t skipped = nchoosek(spaces-1, count-1);
    if (rank >= skipped)
        unrank(rank-skipped, pieces, count, spaces-1, total);
    else {
        pieces[0] = total-spaces;
        unrank(rank, &pieces[1], count-1, spaces-1, total);
    }
}
```

Unranking function

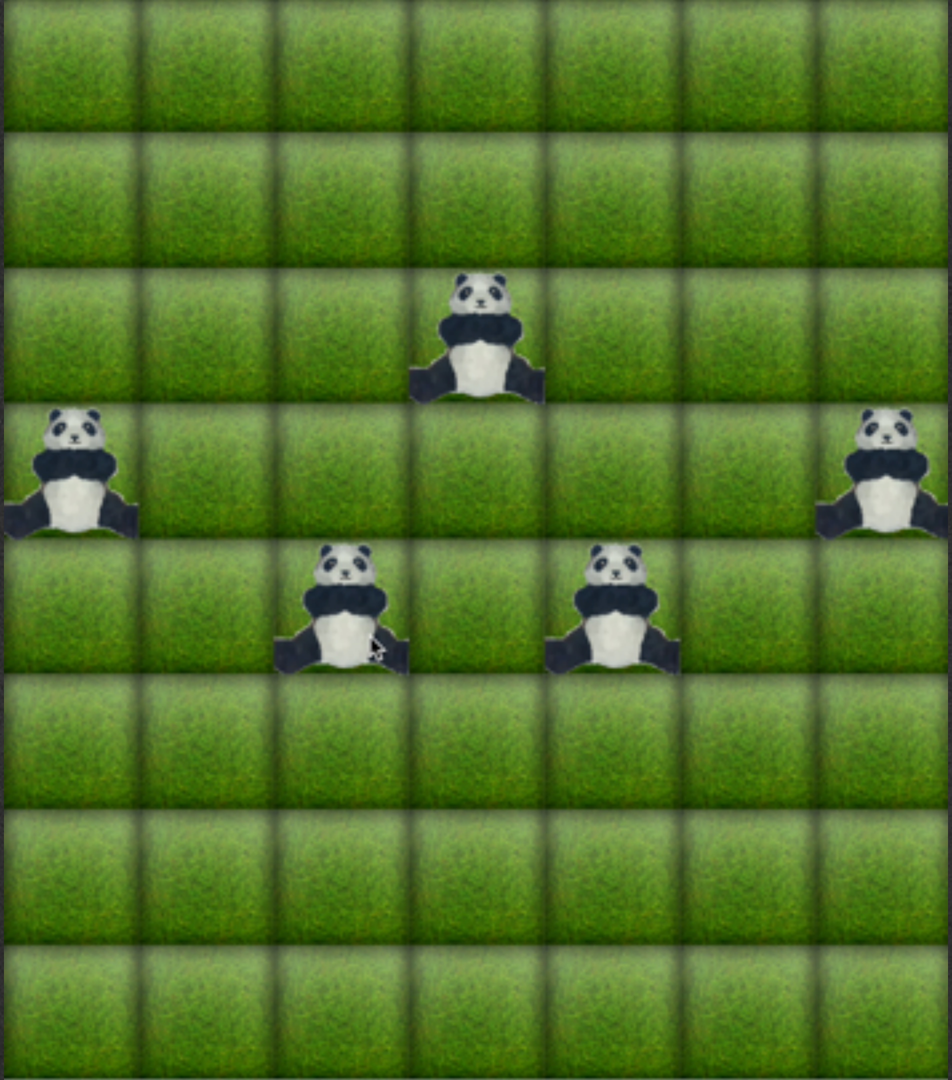
```
void unrank(uint64_t rank, int *pieces, int count, int spaces, int total)
{
    if (count == 0)
        return;
    uint64_t skipped = nchoosek(spaces-1, count-1);
    if (rank >= skipped)
        unrank(rank-skipped, pieces, count, spaces-1, total);
    else {
        pieces[0] = total-spaces;
        unrank(rank, &pieces[1], count-1, spaces-1, total);
    }
}
```

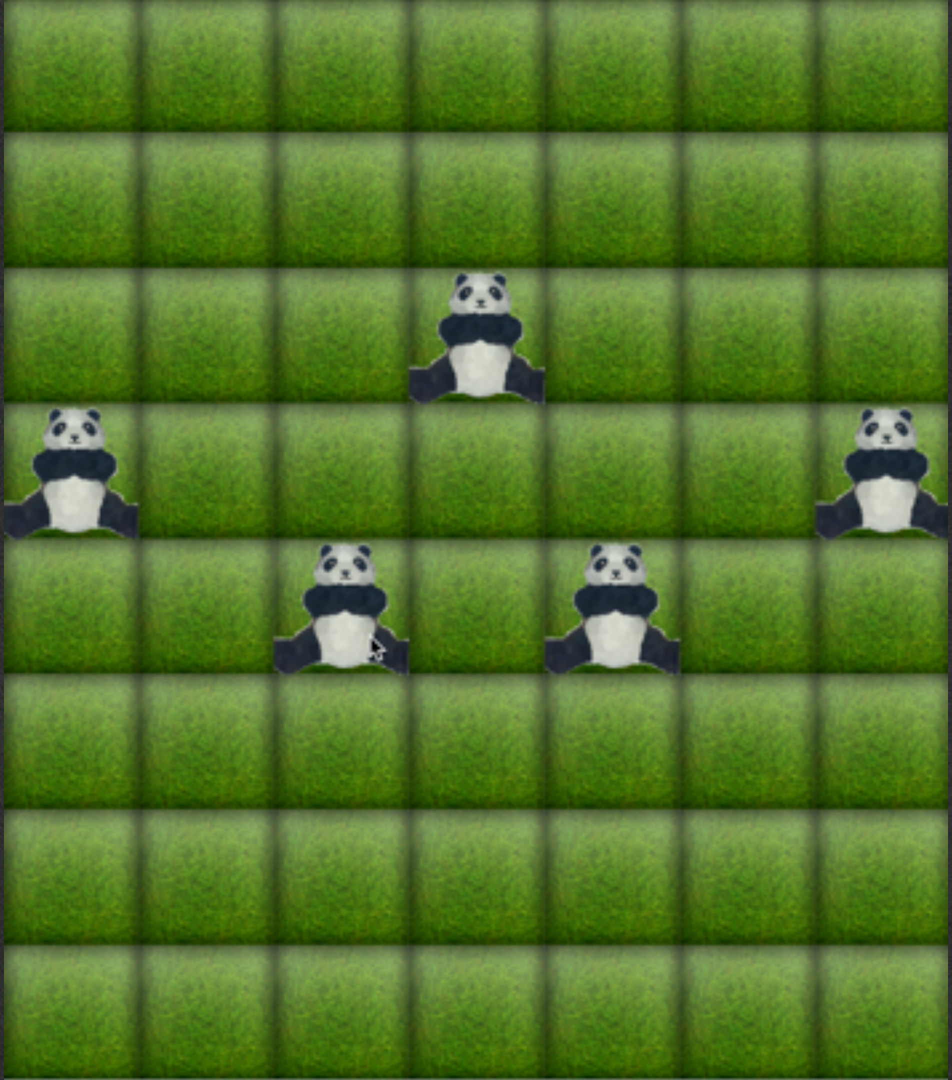
Unranking function

```
void unrank(uint64_t rank, int *pieces, int count, int spaces, int total)
{
   if (count == 0)
      return;
   uint64_t skipped = nchoosek(spaces-1, count-1);
   if (rank >= skipped)
      unrank(rank-skipped, pieces, count, spaces-1, total);
   else {
      pieces[0] = total-spaces;
      unrank(rank, &pieces[1], count-1, spaces-1, total);
   }
}
```

Unranking function

```
void unrank(uint64_t rank, int *pieces, int count, int spaces, int total)
{
    if (count == 0)
        return;
    uint64_t skipped = nchoosek(spaces-1, count-1);
    if (rank >= skipped)
        unrank(rank-skipped, pieces, count, spaces-1, total);
    else {
        pieces[0] = total-spaces;
        unrank(rank, &pieces[1], count-1, spaces-1, total);
    }
}
```





Retrograde Analysis (solvable)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = false;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) == kSolvable)
             solvable = true;
          b.UndoMove(m);
          if (solvable) break;
     }
     Store(i, solvable);
```

Retrograde Analysis (solvable)

```
For i = 0…# states-1
      b = unrank(i)
      bool solvable = false;
      for (int each move m on board b)
      {
            b.ApplyMove(m);
            if (Lookup(rank(b)) == kSolvable)
               solvable = true;
            b.UndoMove(m);
            if (solvable) break;
      }
      Store(i, solvable);
```

Retrograde Analysis (solvable)

```
For i = 0…# states-1
      b = unrank(i)
      bool solvable = false;
      for (int each move m on board b)
      {
            b.ApplyMove(m);
            if (Lookup(rank(b)) == kSolvable)
               solvable = true;
            b.UndoMove(m);
            if (solvable) break;
      }
      Store(i, solvable);
```

Retrograde Analysis (solvable)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = false;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) == kSolvable)
             solvable = true;
          b.UndoMove(m);
          if (solvable) break;
     }
     Store(i, solvable);
```

Retrograde Analysis (solvable)

```
For i = 0…# states-1
        b = unrank(i)
        bool solvable = false;
        for (int each move m on board b)
        {
                b.ApplyMove(m);
                if (Lookup(rank(b)) == kSolvable)
                    solvable = true;
                b.UndoMove(m);
                if (solvable) break;
        }
        Store(i, solvable);
```

Retrograde Analysis (solvable)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = false;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) == kSolvable)
             solvable = true;
          b.UndoMove(m);
          if (solvable) break;
     }
     Store(i, solvable);
```

Retrograde Analysis (all moves)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = true;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) != kSolvable)
             solvable = false;
          b.UndoMove(m);
          if (!solvable) break;
     }
     Store(i, solvable);
```

Retrograde Analysis (all moves)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = true;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) != kSolvable)
             solvable = false;
          b.UndoMove(m);
          if (!solvable) break;
     }
     Store(i, solvable);
```

Retrograde Analysis (all moves)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = true;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) != kSolvable)
            solvable = false;
          b.UndoMove(m);
          if (!solvable) break;
     }
     Store(i, solvable);
```

Retrograde Analysis (all moves)

```
For i = 0…# states-1
     b = unrank(i)
     bool solvable = true;
     for (int each move m on board b)
     {
          b.ApplyMove(m);
          if (Lookup(rank(b)) != kSolvable)
             solvable = false;
          b.UndoMove(m);
          if (!solvable) break;
     }
     Store(i, solvable);
```

Multi-Sets

(combinations allowing duplicates)

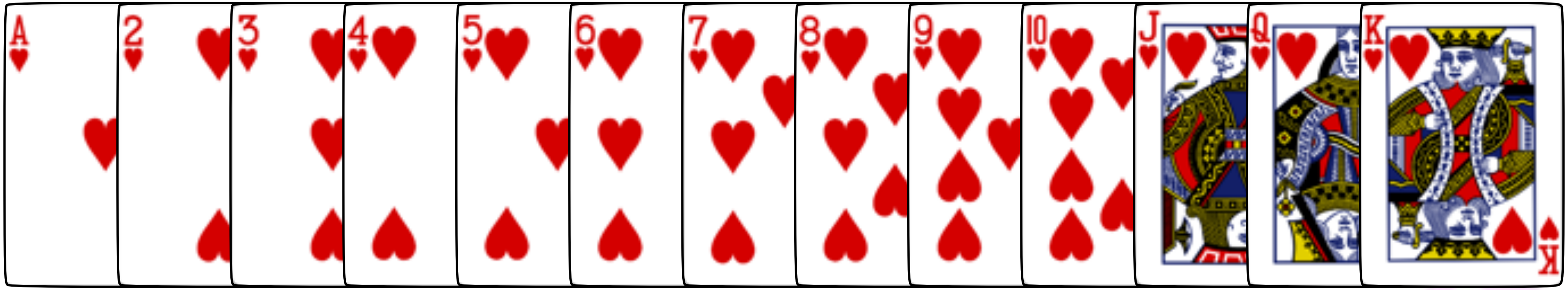
Multi-Set Example

- Build an AI for a card game (duplicate)
  - Pre-compute value of a set of cards
  - At runtime, compute and lookup the index of our current cards.

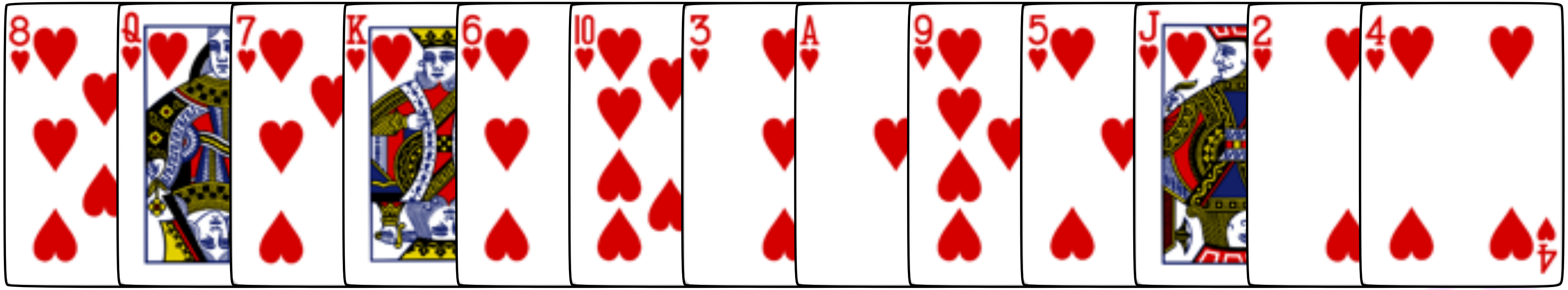
Permutations

Permutations: What decks?

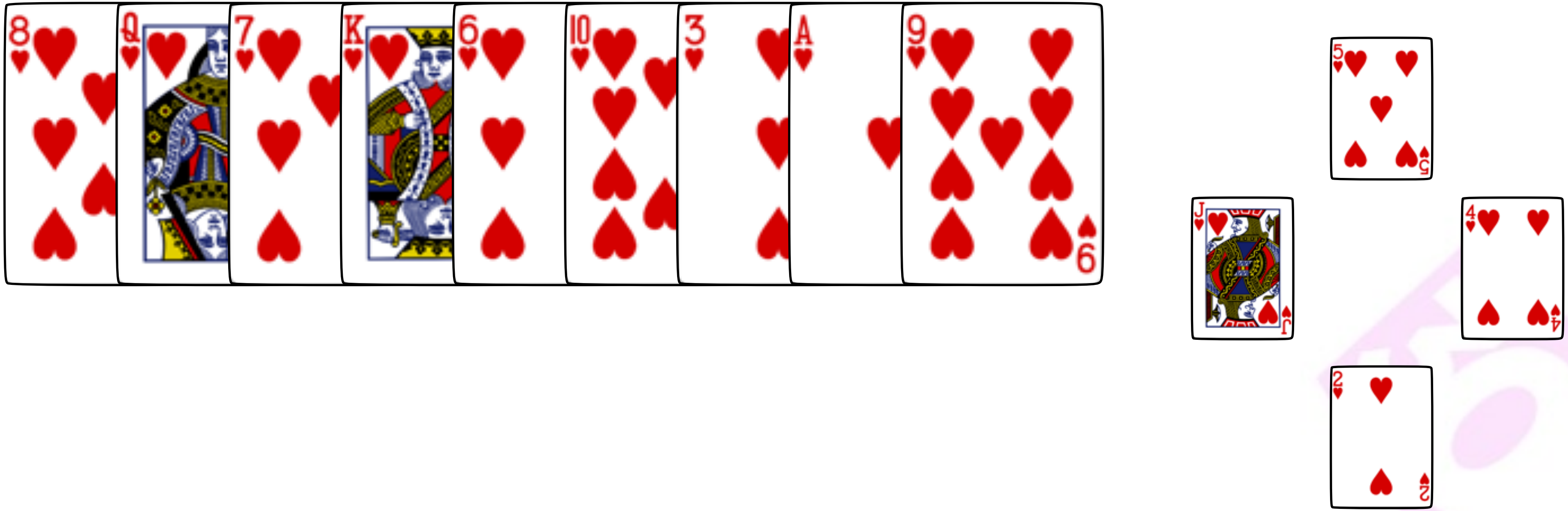
Permutations: What decks?



Permutations



Permutations



Cogs (2009)

Cogs (2009)

5	10	2	14
1	15	3	7
4	6	12	9
8		13	11

Counting Permutations

0 1 2 3

Counting Permutations

0 1 2 3

4

Counting Permutations

0 1 2 3

4 3

Counting Permutations

0 1 2 3

4 3 2

Counting Permutations

0 1 2 3

4 3 2 1

Counting Permutations

0 1 2 3

4 3 2 1

$4! = 24$

Ranking/Unranking Permutations

- Ranking involves mixed-radix numbers
  - Every digit is a different base
  - Time: $7_{24}12_{60}$ (7 hours; 12 min)
  - Currency: $15_{\infty}39_{100}$ (\$15.39)

Conversion to Mixed Radix

$$0_4 \; 1_4 \; 2_4 \; 3_4$$

Conversion to Mixed Radix

$$0_4\ 1_4\ 2_4\ 3_4$$

Conversion to Mixed Radix

$$0_4\ 1_3\ 2_3\ 3_3$$

Conversion to Mixed Radix

$$0_4\ 0_3\ 1_3\ 2_3$$

Conversion to Mixed Radix

$$0_4\ 0_3\ 1_3\ 2_3$$

Conversion to Mixed Radix

$$0_4\,0_3\,\underbrace{1_3\,2_3}$$

Conversion to Mixed Radix

$$0_4\ 0_3\ 1_2\ 2_2$$

Conversion to Mixed Radix

$$0_4\ 0_3\ 0_2\ 1_2$$

Conversion to Mixed Radix

$$0_4\ 0_3\ 0_2\ 0_1$$

Full Ranking Process

$2_4\ 1_4\ 3_4\ 0_4$

Full Ranking Process

$\boxed{2_4}\ 1_4\ 3_4\ 0_4$

$\underbrace{1_4\ 3_4\ 0_4}_{3!}$

Full Ranking Process

$2_4\ 1_4\ 3_4\ 0_4$

$2 \cdot 3!$

Full Ranking Process

$$2_4\ 1_3\ 2_3\ 0_3$$

$2 \cdot 3!$

Full Ranking Process

$2_4 \; \boxed{1_3} \; 2_3 \; 0_3$

$2!$

$2 \cdot 3!$

Full Ranking Process

$$2_4\ 1_3\ 2_3\ 0_3$$

$$2 \cdot 3! + 1 \cdot 2!$$

Full Ranking Process

$$2_4\ 1_3\ 1_2\ 0_2$$

$$2 \cdot 3! + 1 \cdot 2!$$

Full Ranking Process

$$2_4\ 1_3\ \boxed{1_2}\ 0_2$$

$1!$

$2 \cdot 3! + 1 \cdot 2!$

Full Ranking Process

$$2_4\ 1_3\ 1_2\ 0_2$$

$$2 \cdot 3! + 1 \cdot 2! + 1 \cdot 1!$$

Full Ranking Process

$$2_4\ 1_3\ 1_2\ 0_1$$

$$2 \cdot 3! + 1 \cdot 2! + 1 \cdot 1! = 15$$

Pseudo-code

```
uint64_t rank(int *pieces, int count)
{
    uint64_t hashVal = 0;
    int numEntriesLeft = count;

    for (unsigned int x = 0; x < count; x++)
    {
     hashVal += pieces[x]*Factorial(numEntriesLeft-1);
     numEntriesLeft--;

     // decrement locations of remaining items
     for (unsigned y = x; y < count; y++)
     {
         if (pieces[y] > pieces[x])
             pieces[y]--;
     }
    }
    return hashVal;
}
```

Pseudo-code

```
uint64_t rank(int *pieces, int count)
{
    uint64_t hashVal = 0;
    int numEntriesLeft = count;

    for (unsigned int x = 0; x < count; x++)
    {
     hashVal += pieces[x]*Factorial(numEntriesLeft-1);
     numEntriesLeft--;

     // decrement locations of remaining items
     for (unsigned y = x; y < count; y++)
     {
         if (pieces[y] > pieces[x])
             pieces[y]--;
     }
    }
    return hashVal;
}
```

Pseudo-code

```
uint64_t rank(int *pieces, int count)
{
    uint64_t hashVal = 0;
    int numEntriesLeft = count;

    for (unsigned int x = 0; x < count; x++)
    {
     hashVal += pieces[x]*Factorial(numEntriesLeft-1);
     numEntriesLeft--;

     // decrement locations of remaining items
     for (unsigned y = x; y < count; y++)
     {
          if (pieces[y] > pieces[x])
               pieces[y]--;
     }
    }
    return hashVal;
}
```

Pseudo-code

```
uint64_t rank(int *pieces, int count)
{
   uint64_t hashVal = 0;
   int numEntriesLeft = count;

   for (unsigned int x = 0; x < count; x++)
   {
    hashVal += pieces[x]*Factorial(numEntriesLeft-1);
    numEntriesLeft--;

    // decrement locations of remaining items
    for (unsigned y = x; y < count; y++)
    {
         if (pieces[y] > pieces[x])
              pieces[y]--;
    }
   }
   return hashVal;
}
```

Unranking to Mixed Radix

$$?_4 \; ?_3 \; ?_2 \; ?_1$$

Rank = 15

Unranking to Mixed Radix

$$?_4 \; ?_3 \; ?_2 \; ?_1$$

Rank = 15

Unranking to Mixed Radix

$$?_4 \ ?_3 \ ?_2 \ ?_1$$

$15\%1 = 0$

Rank = 15

Unranking to Mixed Radix

$$?_4\ ?_3\ ?_2\ 0_1$$

Rank = 15

15%1 = 0

Unranking to Mixed Radix

$?_4\ ?_3\ ?_2\ 0_1$

Rank = 15

$15\%1 = 0$

Next Rank: $15/1 = 15$

Unranking to Mixed Radix

$$?_4 \; ?_3 \; ?_2 \; 0_1$$

Rank = 15

Unranking to Mixed Radix

$$?_4 \; ?_3 \; ?_2 \; 0_1$$

Rank = 15

15%2 = 1

Unranking to Mixed Radix

$$?_4\ ?_3\ 1_2\ 0_1$$

$15\%2 = 1$

Rank = 15

Unranking to Mixed Radix

$$?_4 \; ?_3 \; 1_2 \; 0_1$$

Rank = 15

15%2 = 1

Next Rank: 15/2 = 7

Unranking to Mixed Radix

$$?_4\ ?_3\ 1_2\ 0_1$$

Rank = 7

15%2 = 1

Next Rank: 15/2 = 7

Unranking to Mixed Radix

$$?_4 \ ?_3 \ 1_2 \ 0_2$$

Rank = 7

Unranking to Mixed Radix

$$?_4\ ?_3\ 1_2\ 0_2$$

Rank = 7

Unranking to Mixed Radix

$$?_4 \; ?_3 \; 1_2 \; 0_2$$

$7\%3 = 1$

Rank = 7

Unranking to Mixed Radix

$?_4 \; 1_3 \; 1_2 \; 0_2$

$7\%3 = 1$

Rank = 7

Unranking to Mixed Radix

$$?_4 \ 1_3 \ 1_2 \ 0_2$$

Rank = 7

7%3 = 1

Next Rank: 7/3 = 2

Unranking to Mixed Radix

$$?_4\ 1_3\ 1_2\ 0_2$$

Rank = 2

Unranking to Mixed Radix

$?_4 \; 1_3 \; 1_2 \; 0_2$

Rank = 2

Unranking to Mixed Radix

$$?_4 \; 1_3 \; 2_3 \; 0_3$$

Rank = 2

Unranking to Mixed Radix

$$2_4\ 1_3\ 2_3\ 0_3$$

Rank = 2

Unranking to Mixed Radix

$$2_4\ 1_3\ 2_3\ 0_3$$

Rank = 2

Unranking to Mixed Radix

$$2_4\ 1_4\ 3_4\ 0_4$$

Pseudo-code

```
void unrank(uint64_t hash, int *pieces, int count)
{
    int numEntriesLeft = 1;
    for (int x = count-1; x >= 0; x--)
    {
     pieces[x] = hash%numEntriesLeft;
     hash /= numEntriesLeft;
     numEntriesLeft++;
     for (int y = x+1; y < count; y++)
     {
          if (pieces[y] >= pieces[x])
               pieces[y]++;
     }
    }
}
```

Pseudo-code

```
void unrank(uint64_t hash, int *pieces, int count)
{
    int numEntriesLeft = 1;
    for (int x = count-1; x >= 0; x--)
    {
        pieces[x] = hash%numEntriesLeft;
        hash /= numEntriesLeft;
        numEntriesLeft++;
        for (int y = x+1; y < count; y++)
        {
            if (pieces[y] >= pieces[x])
                pieces[y]++;
        }
    }
}
```

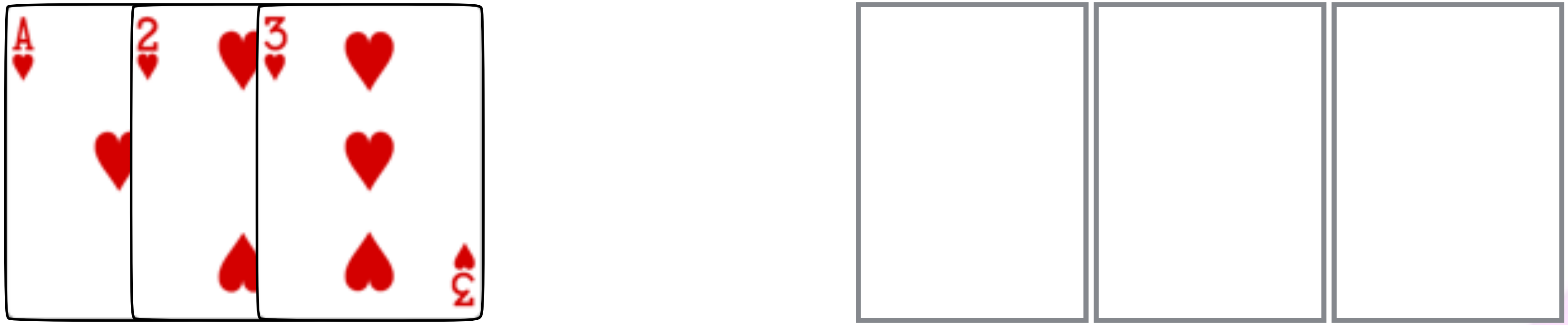
Pseudo-code

```
void unrank(uint64_t hash, int *pieces, int count)
{
	int numEntriesLeft = 1;
	for (int x = count-1; x >= 0; x--)
	{
	  pieces[x] = hash%numEntriesLeft;
	  hash /= numEntriesLeft;
	  numEntriesLeft++;
	  for (int y = x+1; y < count; y++)
	  {
		if (pieces[y] >= pieces[x])
			pieces[y]++;
	  }
	}
}
```

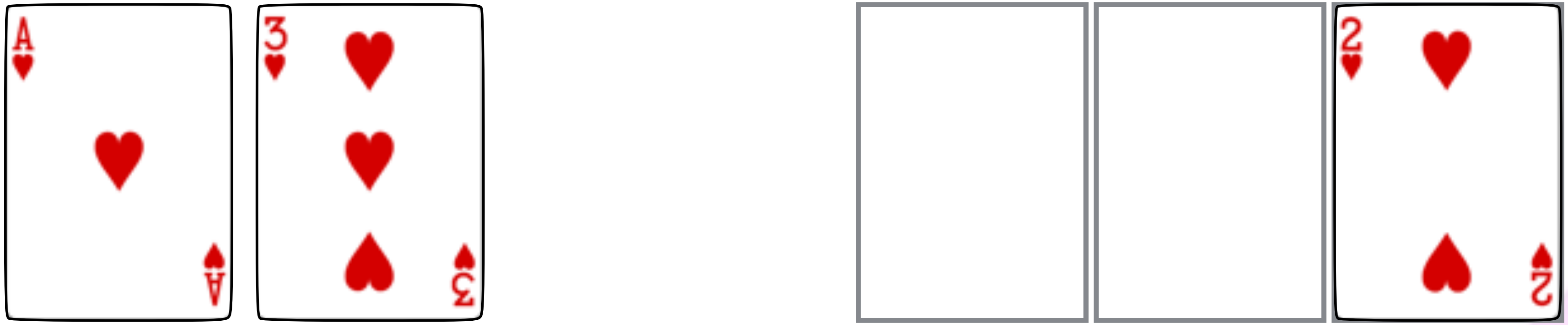
Pseudo-code

```
void unrank(uint64_t hash, int *pieces, int count)
{
    int numEntriesLeft = 1;
    for (int x = count-1; x >= 0; x--)
    {
        pieces[x] = hash%numEntriesLeft;
        hash /= numEntriesLeft;
        numEntriesLeft++;
        for (int y = x+1; y < count; y++)
        {
            if (pieces[y] >= pieces[x])
                pieces[y]++;
        }
    }
}
```

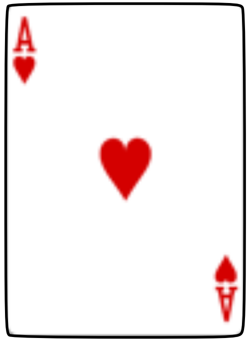
Detour: Randomize Deck

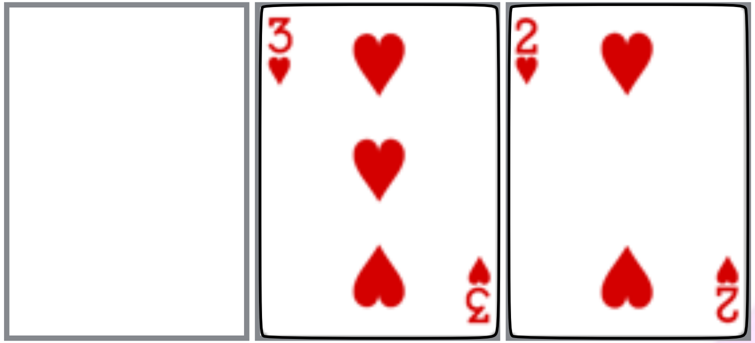


Detour: Randomize Deck

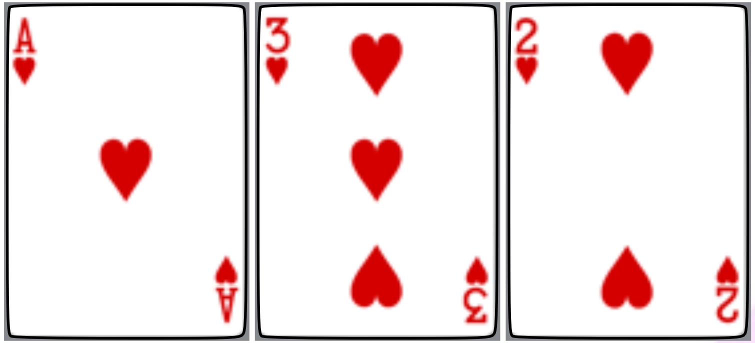


Detour: Randomize Deck

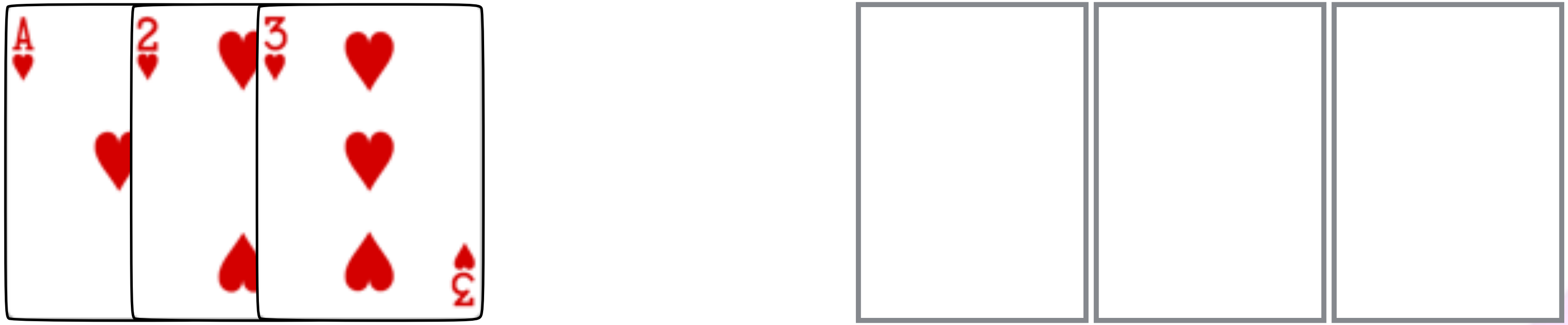




Detour: Randomize Deck

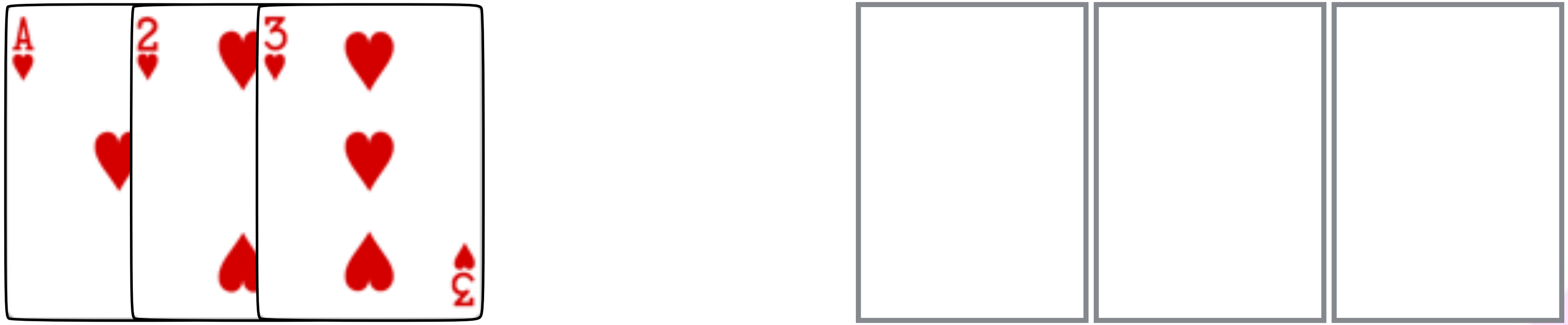


Myrvold & Ruskey



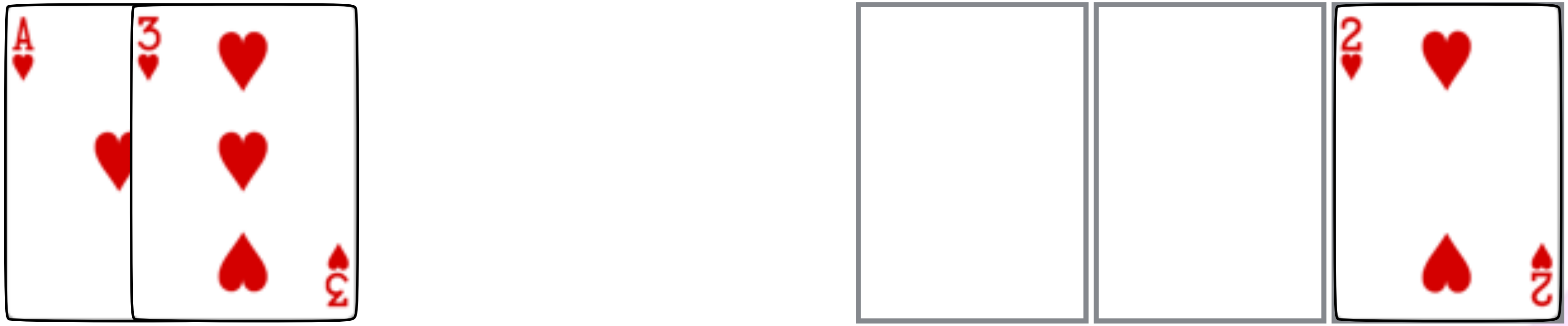
Rank: 4

Myrvold & Ruskey



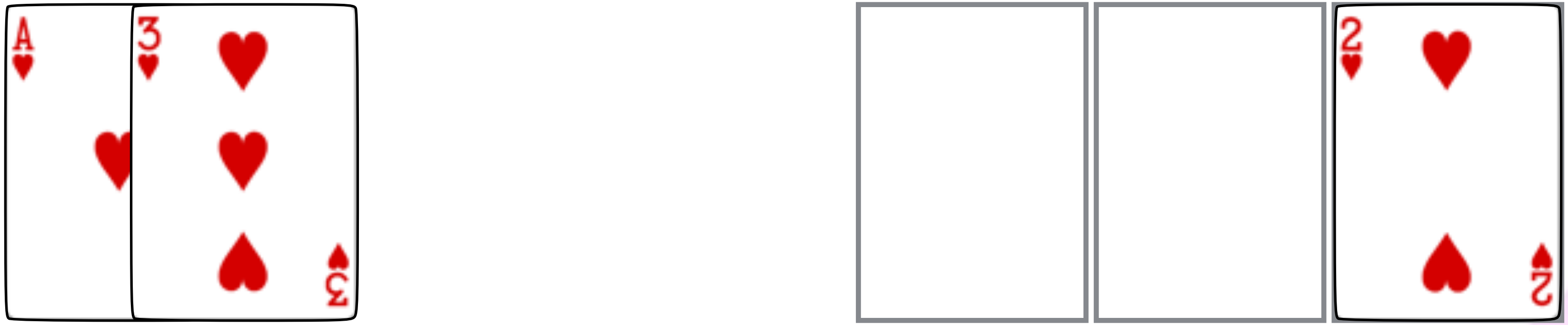
Rank: 4 Next card: 4%3 = 1

Myrvold & Ruskey



Rank: 4   Next card: 4%3 = 1

Myrvold & Ruskey

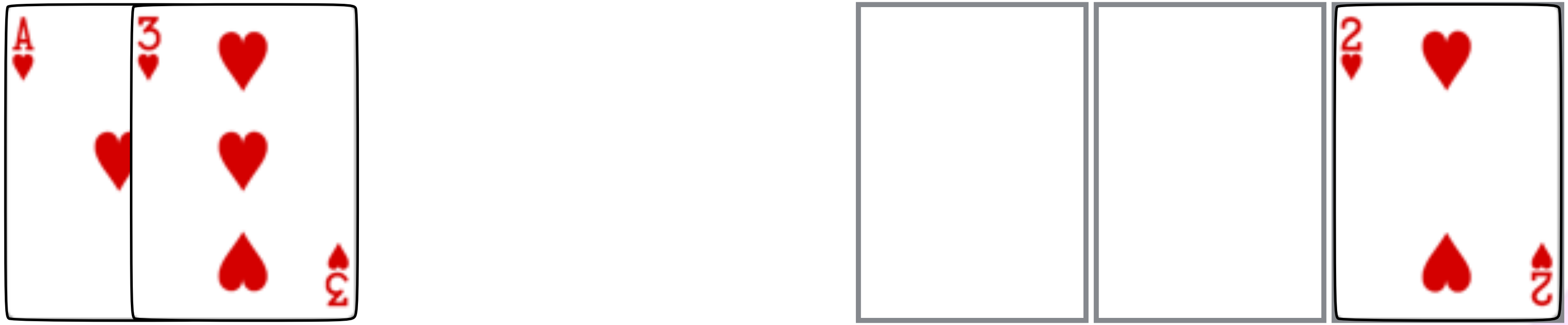


Rank: 4

Next card: 4%3 = 1

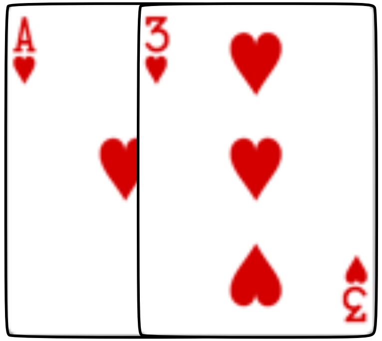
Next rank: 4/3 = 1

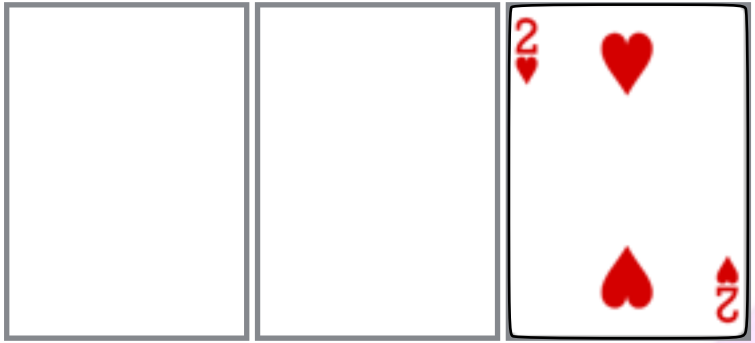
Myrvold & Ruskey



Rank: 1

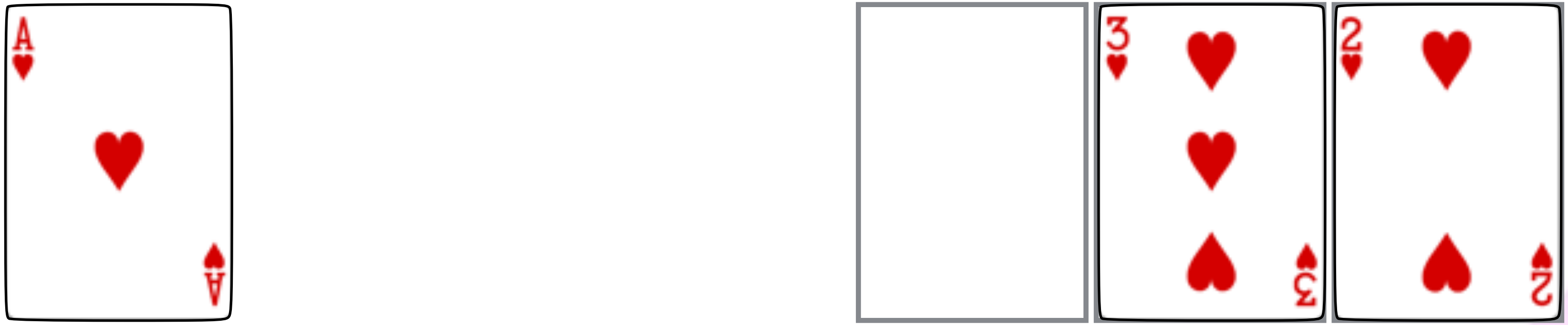
Myrvold & Ruskey





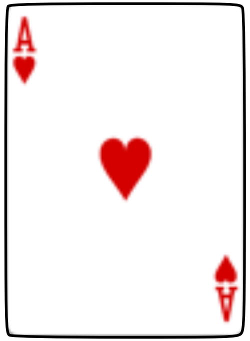
Rank: 1 Next card: 1%2 = 1

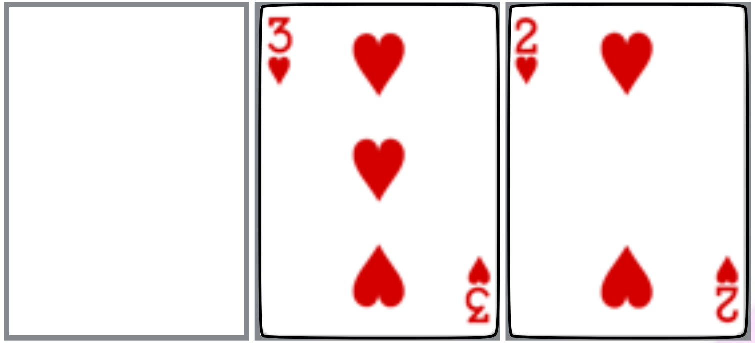
Myrvold & Ruskey



Rank: 1 Next card: 1%2 = 1

Myrvold & Ruskey



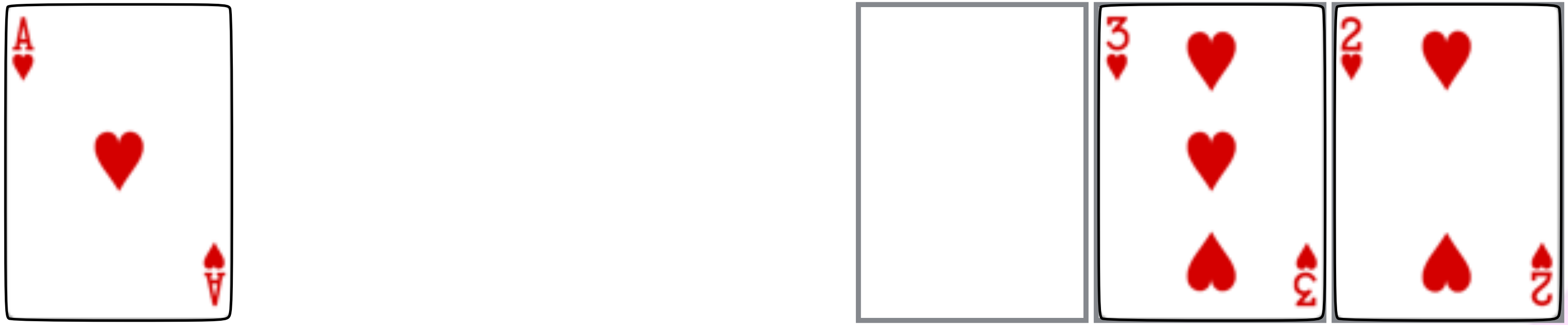


Rank: 1

Next card: 1%2 = 1

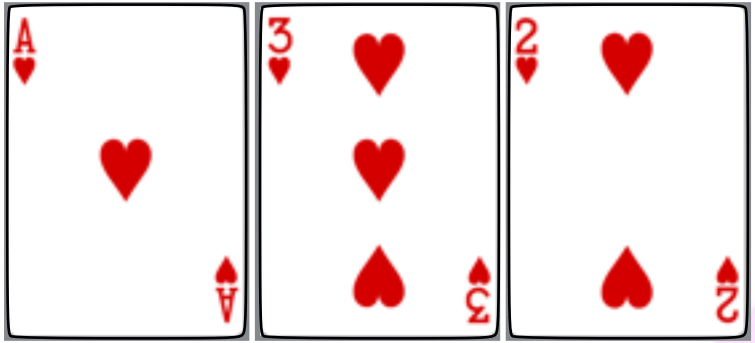
Next rank: 1/2 = 0

Myrvold & Ruskey



Rank: 0 Next card: 0%1 = 0

Myrvold & Ruskey



Rank: 0 Next card: 0%1 = 0

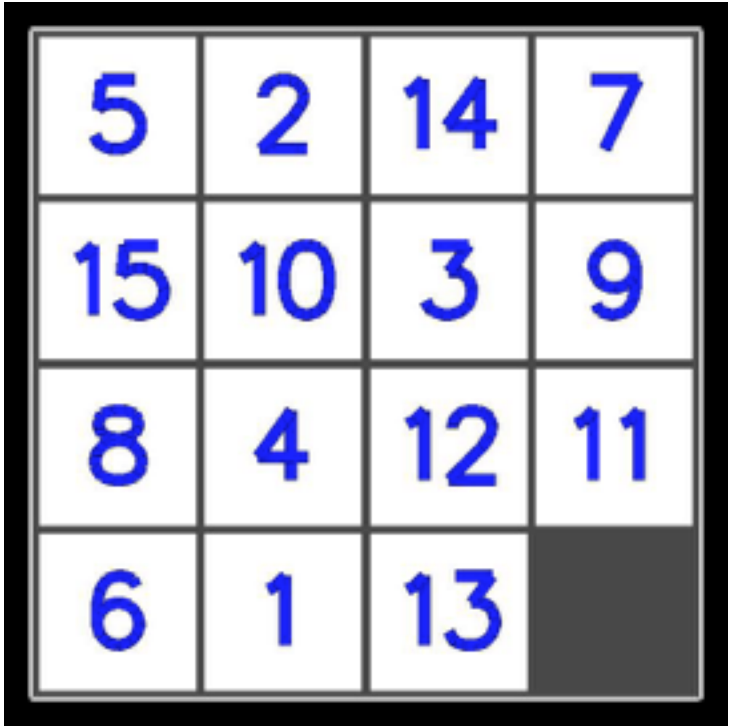
Pseudo-code

```
void unrank(uint64_t rank, int *pieces, int count)
{
    size_t last = 0;

    for (int i = count; i > 0; i--)
    {
      swap(pieces[rank%i], pieces[i-1]);
      rank = rank/i;
    }
}
```

Pseudo-code

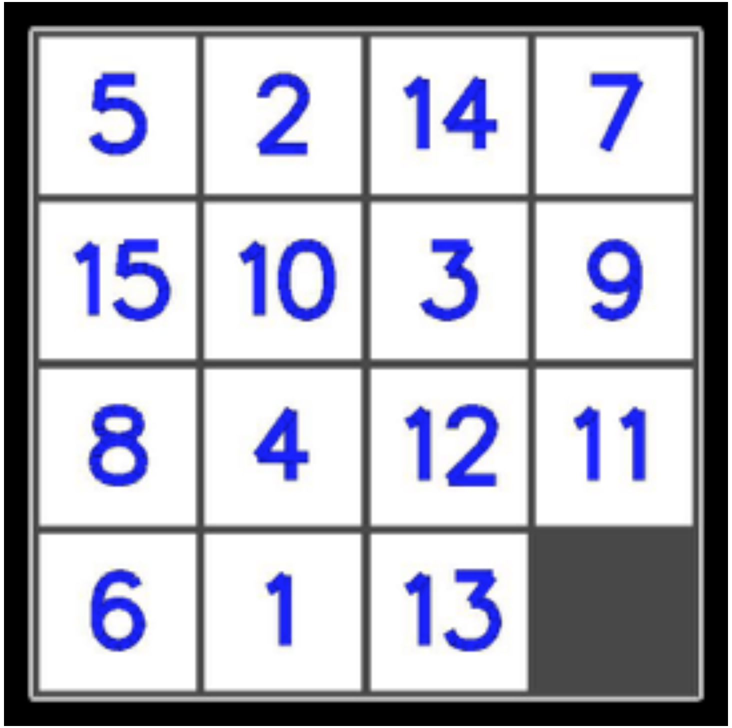
```
void unrank(uint64_t rank, int *pieces, int count)
{
    size_t last = 0;

    for (int i = count; i > 0; i--)
    {
        swap(pieces[rank%i], pieces[i-1]);
        rank = rank/i;
    }
}
```

Sliding Tile Puzzle (k-permutation)



Sliding Tile Puzzle (k-permutation)

Sliding Tile Puzzle (k-permutation)



Software

- http://www.movingai.com/GDC16/
- Find software to compute:
  - Permutations, k-permutations
    - Both lexicographical and MR
  - Combinations
  - Rankings & Unrankings for all approaches

For more information

- *Combinatorics A Guided Tour*
  David Mazur

- http://www.movingai.com/GDC16/