Latest Advancements in Automatic Vectorization Research

Stefanos Baziotis
NEC Deutschland
National and Kapodistrian University of Athens

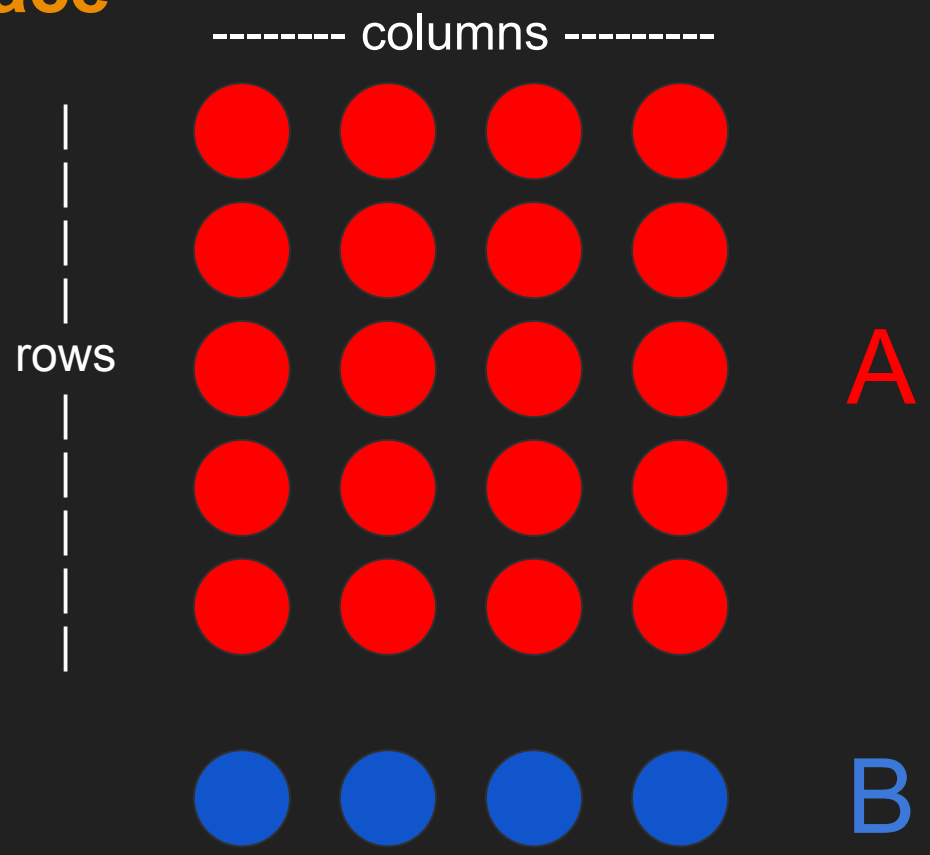
stefanos.baziotis@gmail.com http://users.uoa.gr/~sdi1600105/

LLVM ❤ Compilers Research

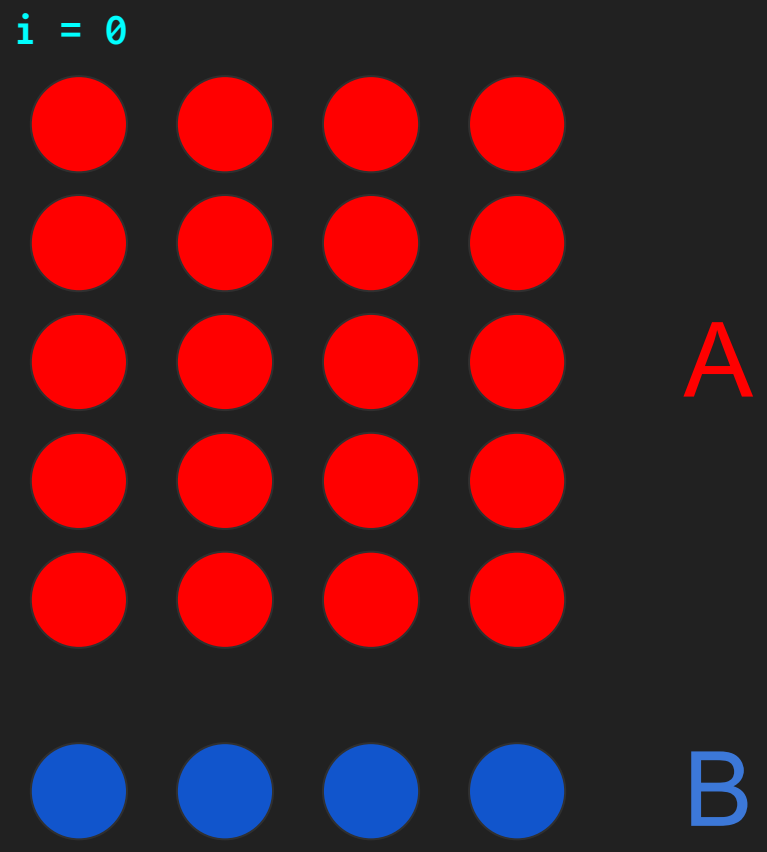
Sum Columns Original

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

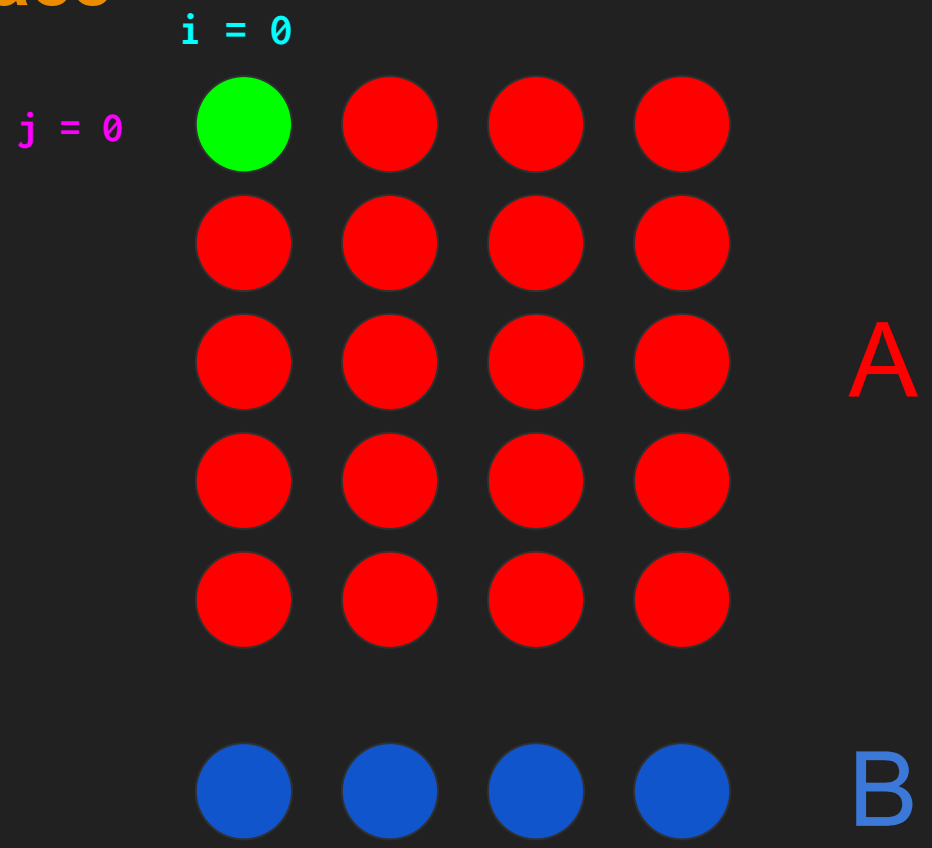
Memory Space



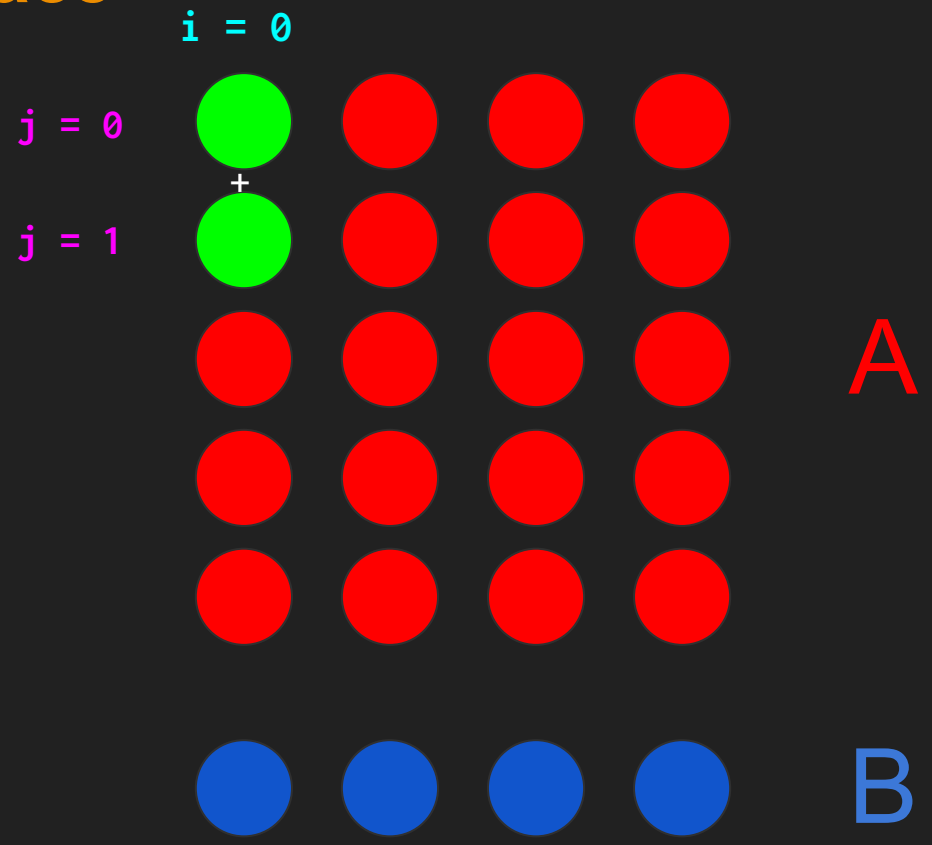
Memory Space



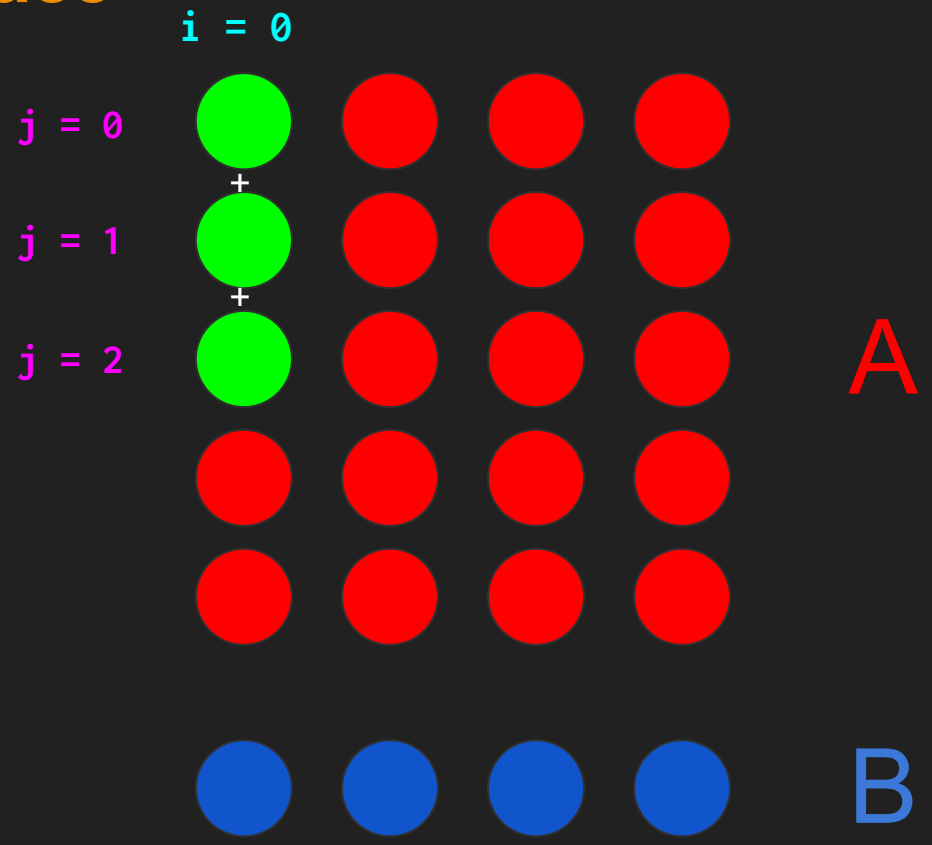
Memory Space



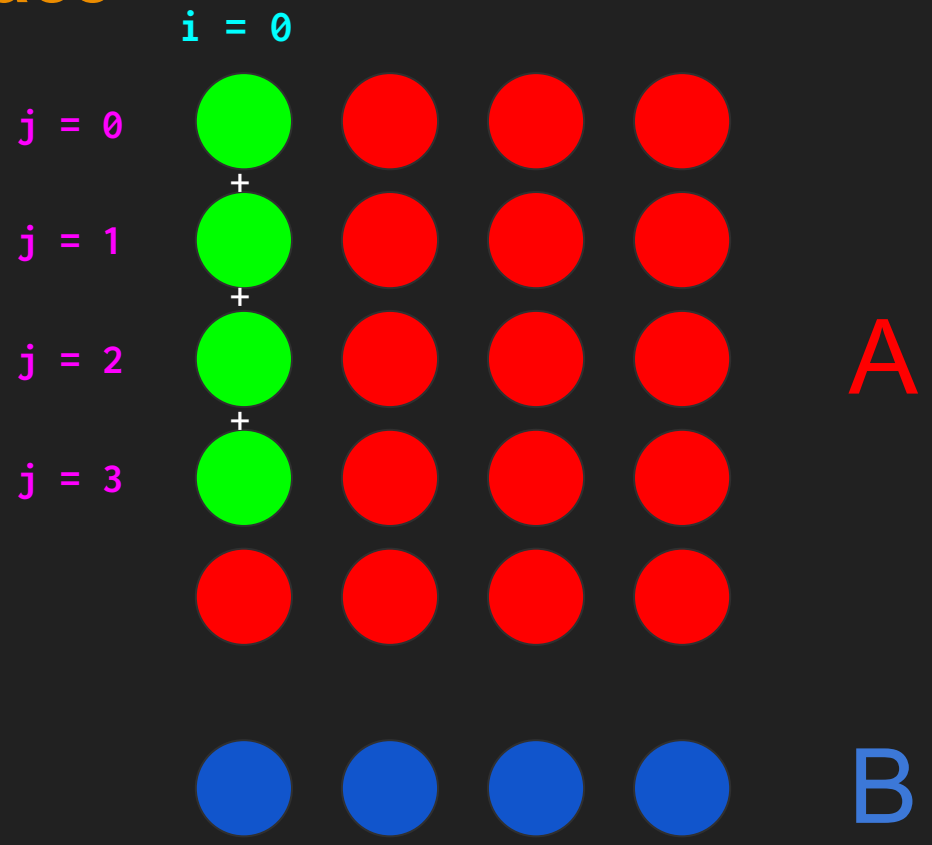
Memory Space



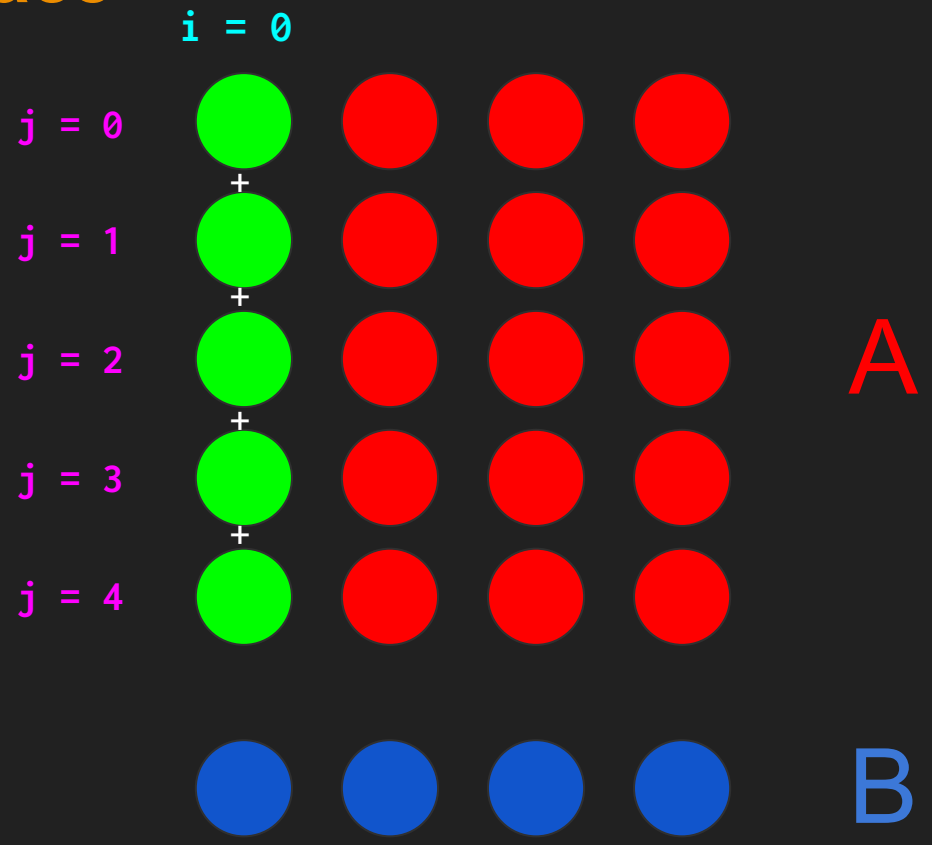
Memory Space



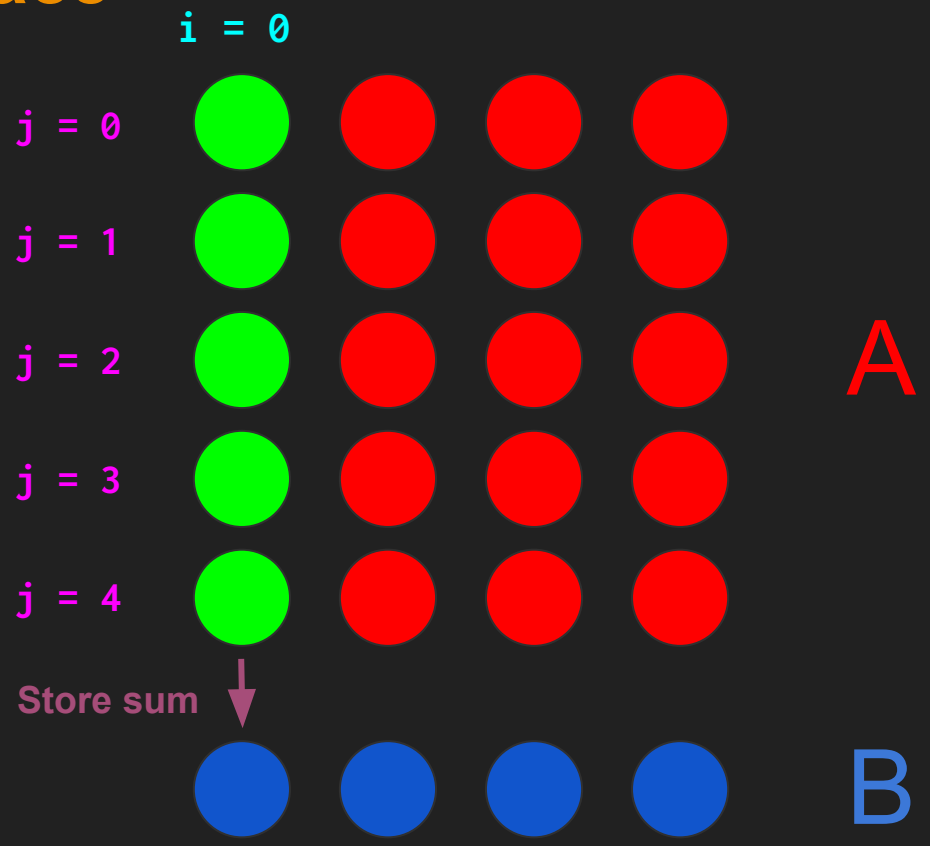
Memory Space



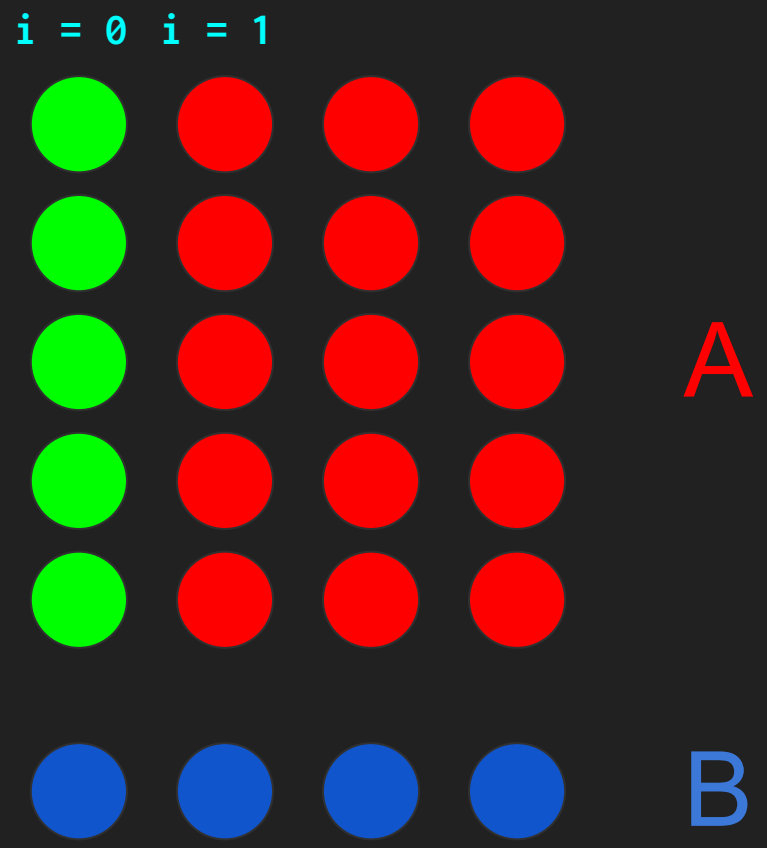
Memory Space



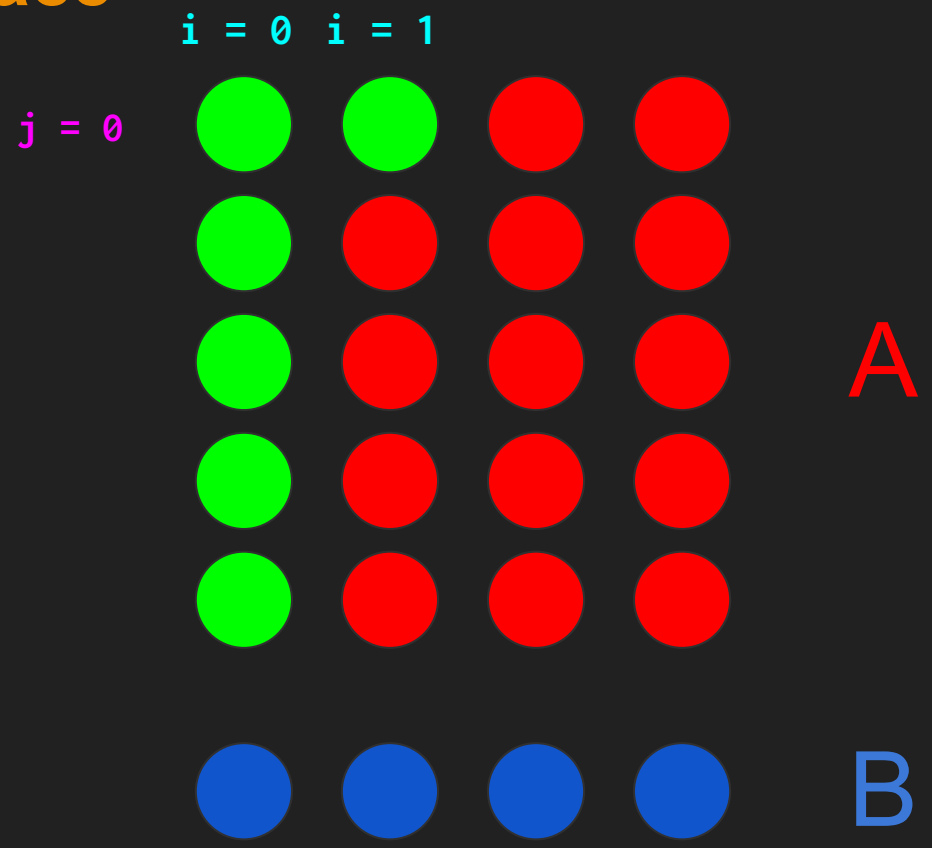
Memory Space



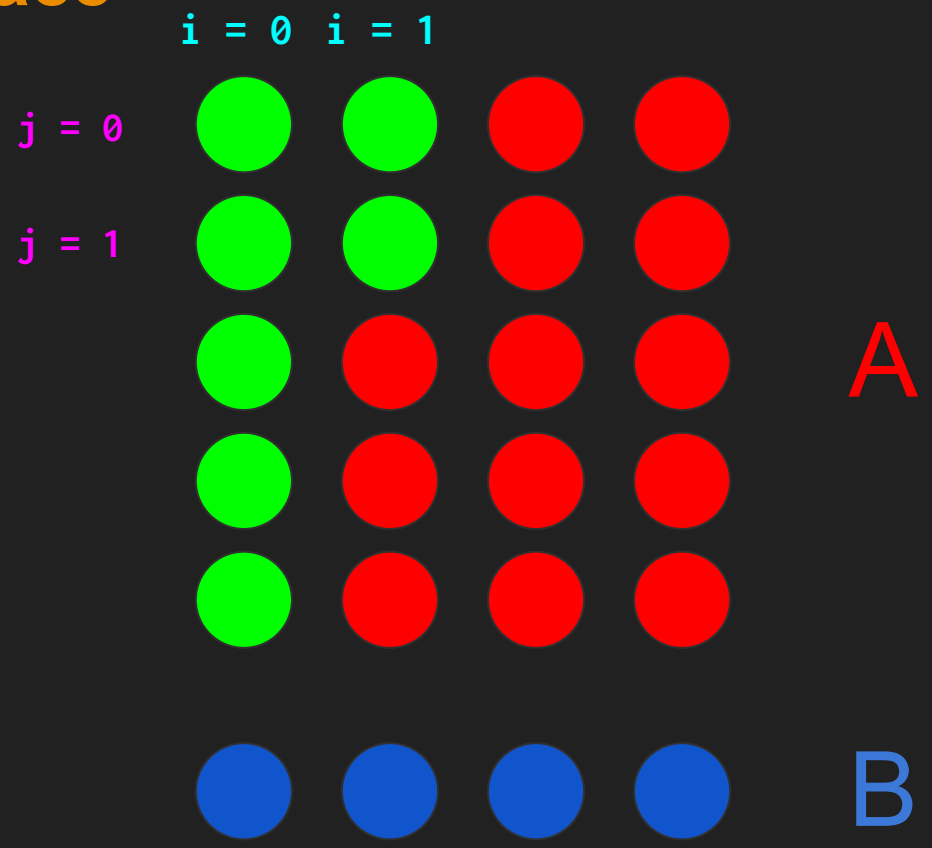
Memory Space



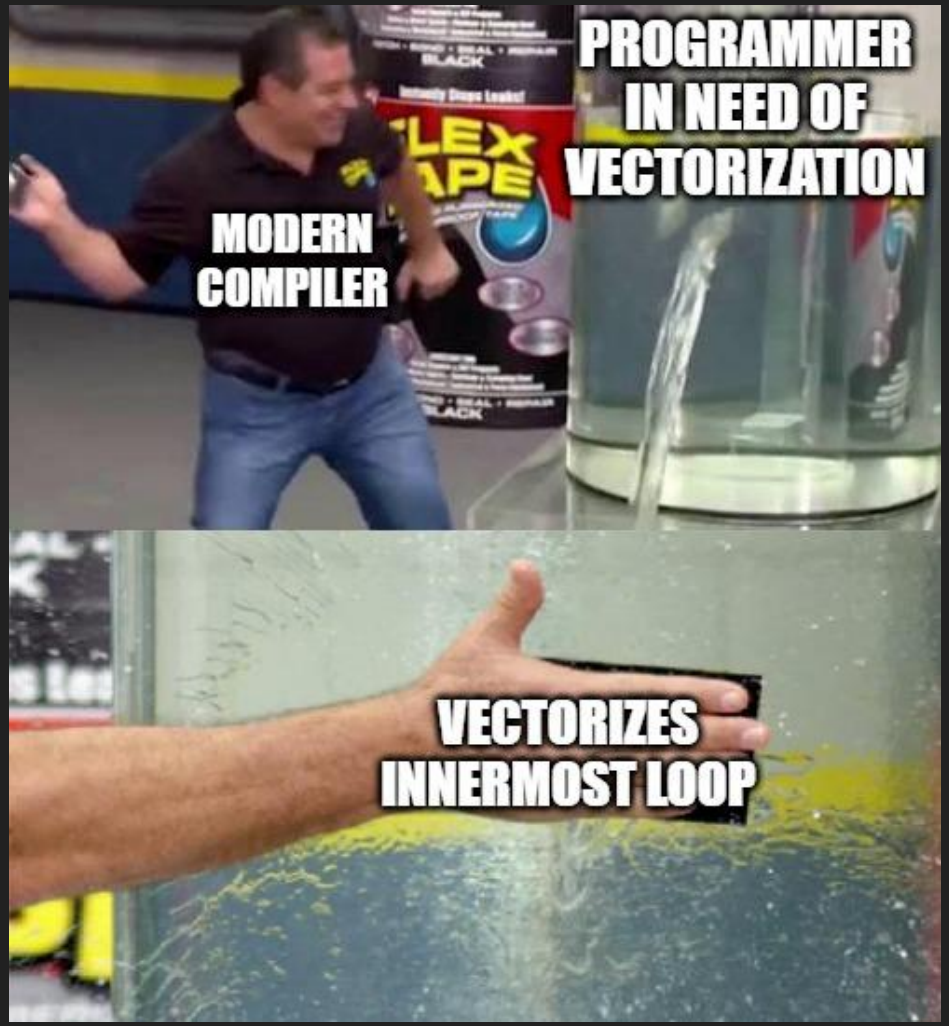
Memory Space



Memory Space



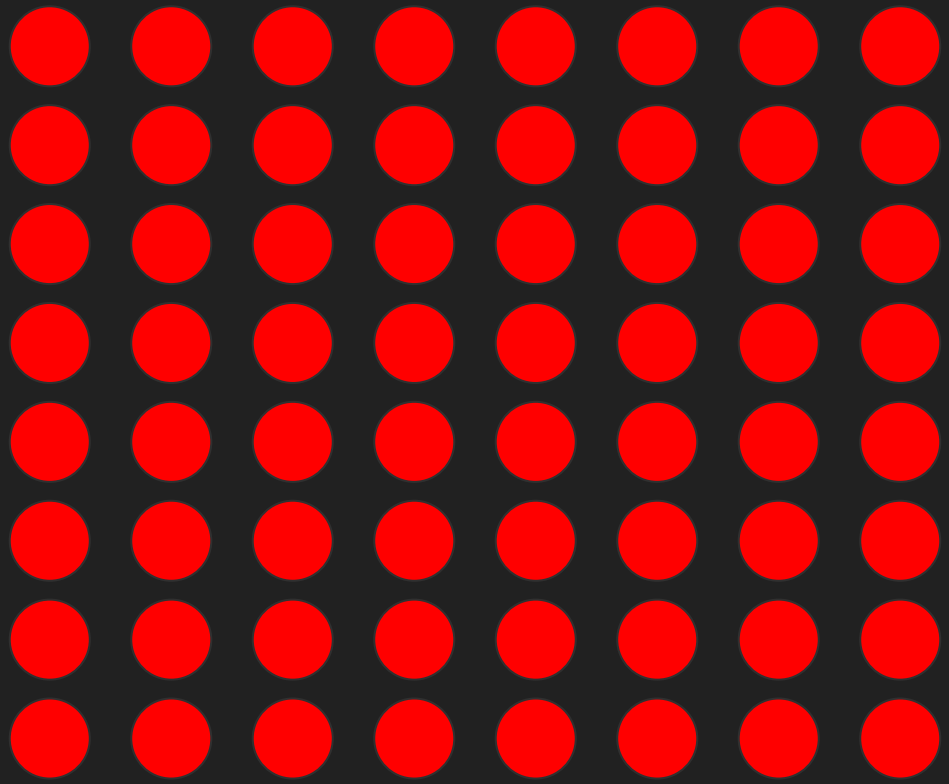

I WANT
VECTORIZATION


MODERN COMPILER
PROGRAMMER IN NEED OF VECTORIZATION
VECTORIZES INNERMOST LOOP

Sum Columns Original

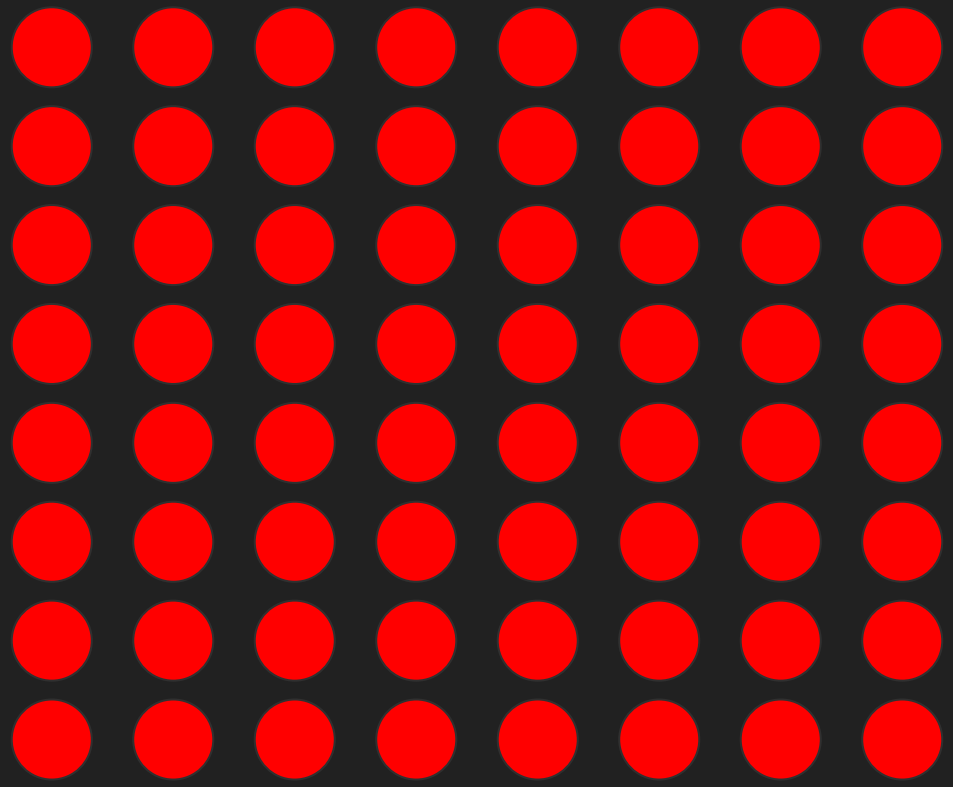
```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Memory Space

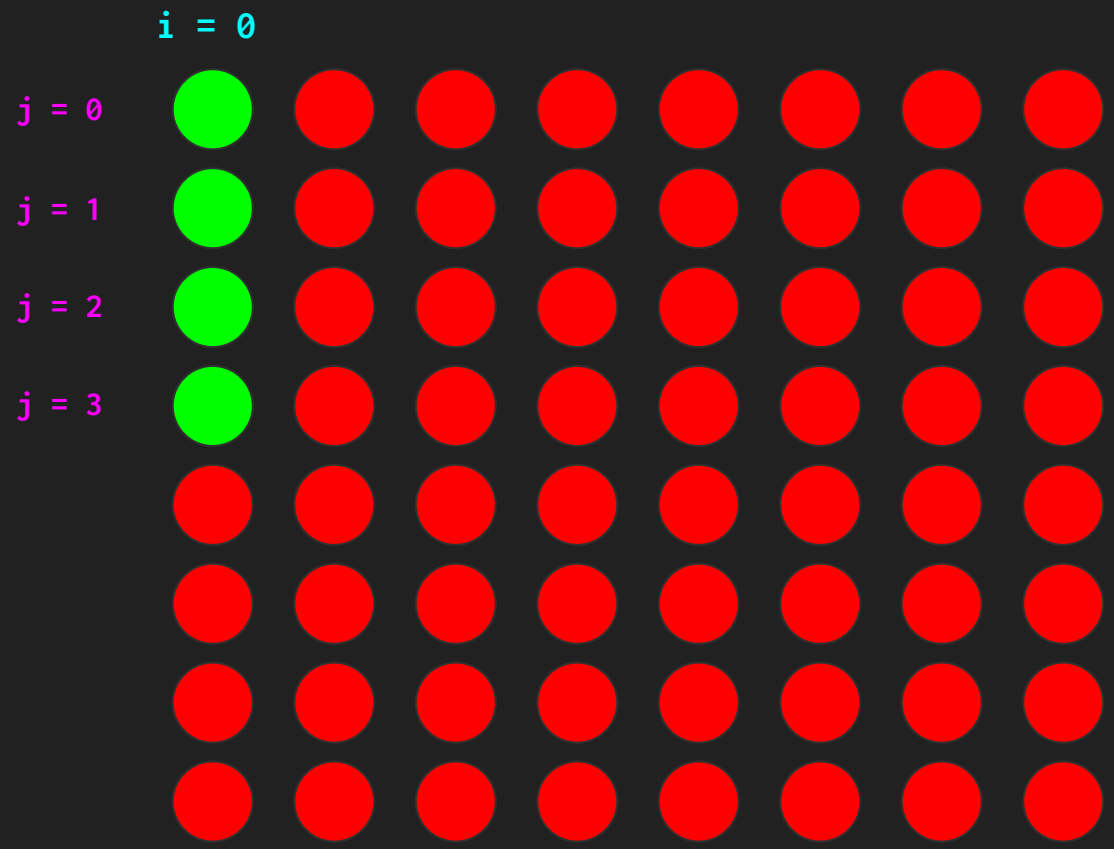


Memory Space

i = 0

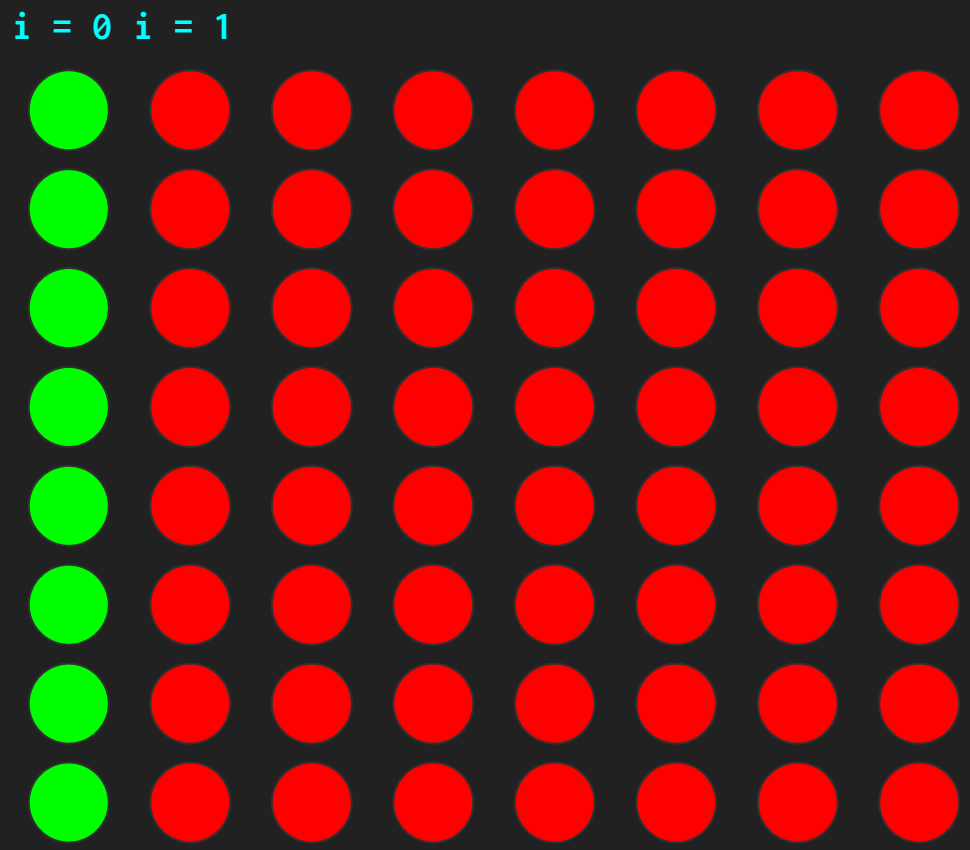


Memory Space

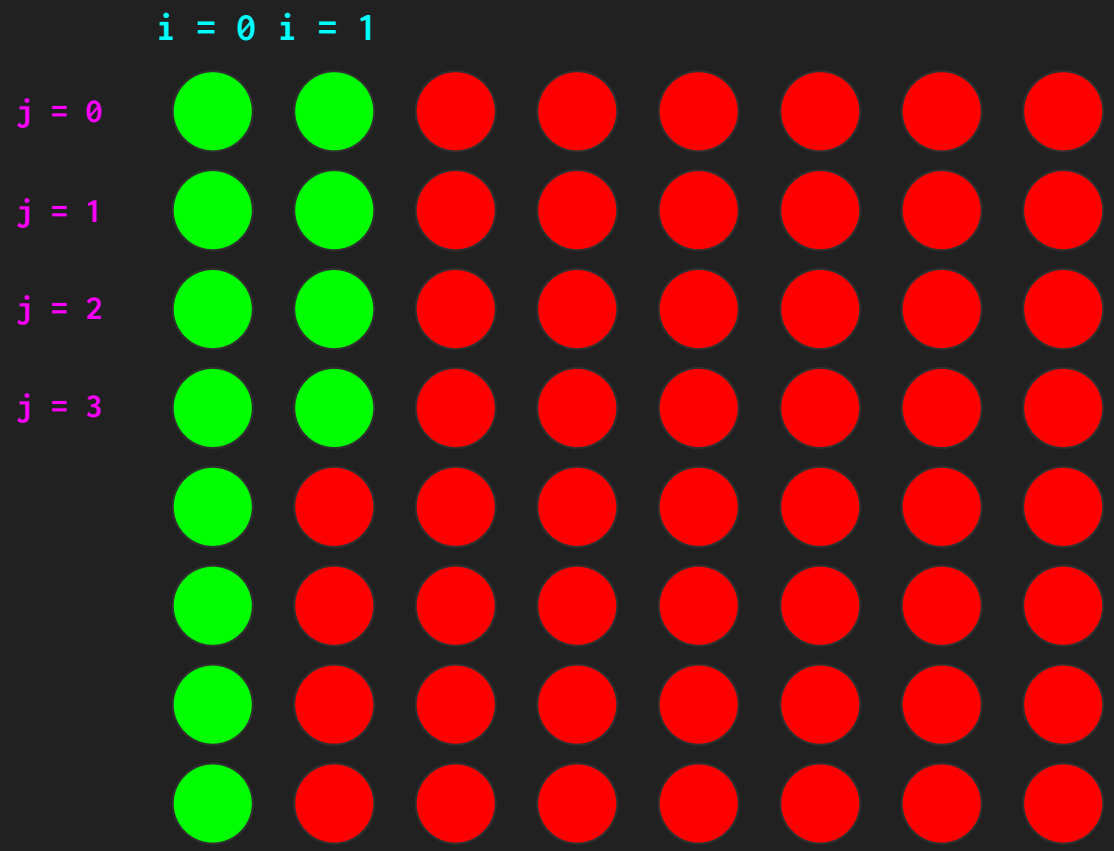


Memory Space

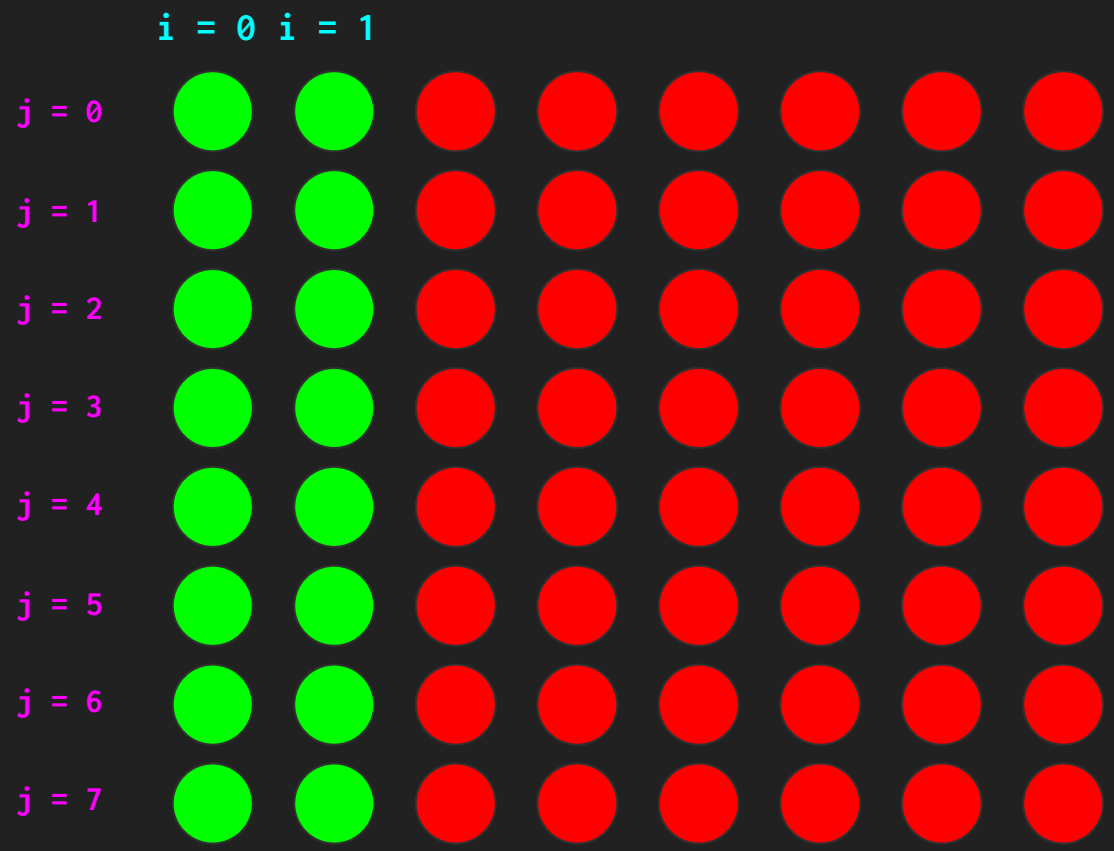
Memory Space



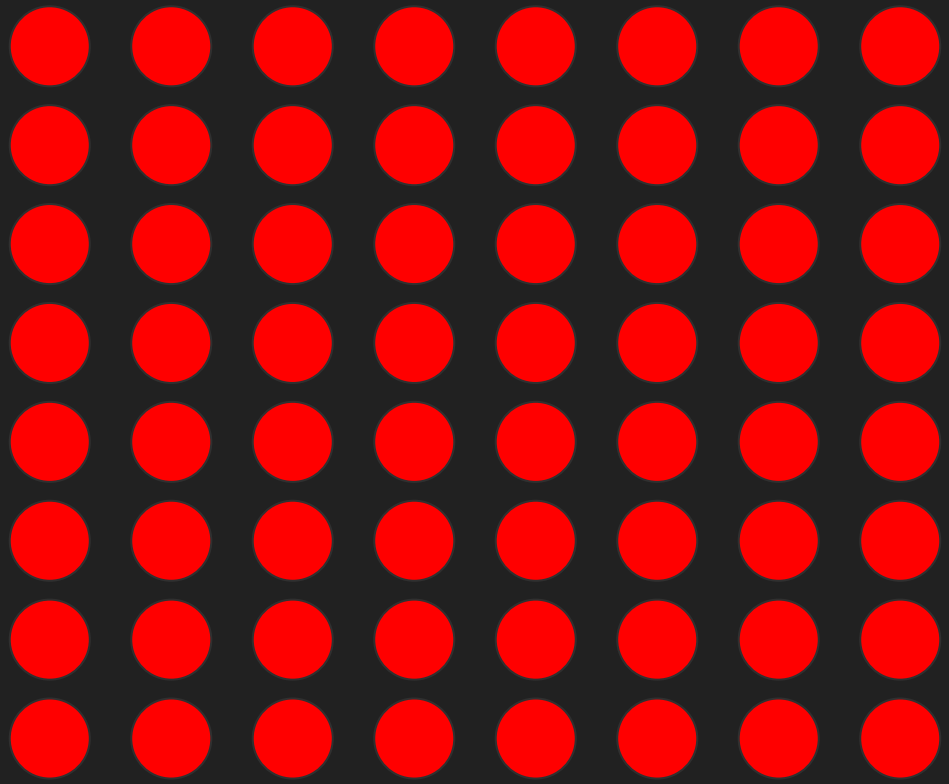
Memory Space



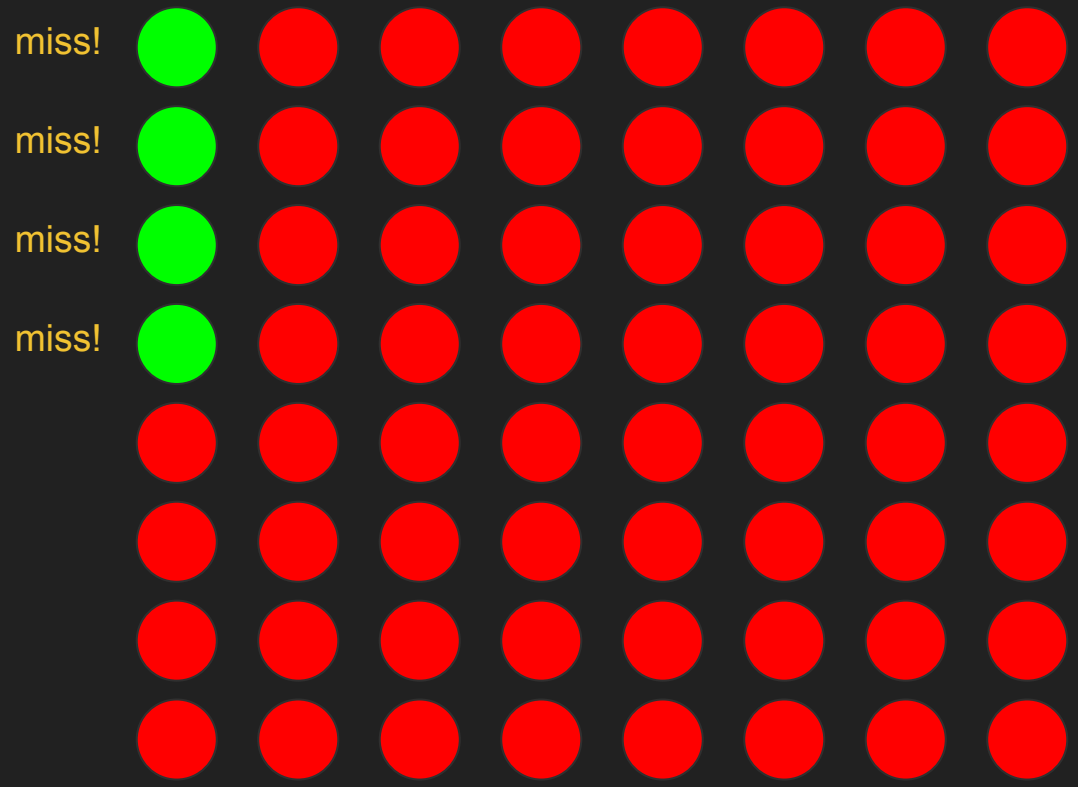
Memory Space



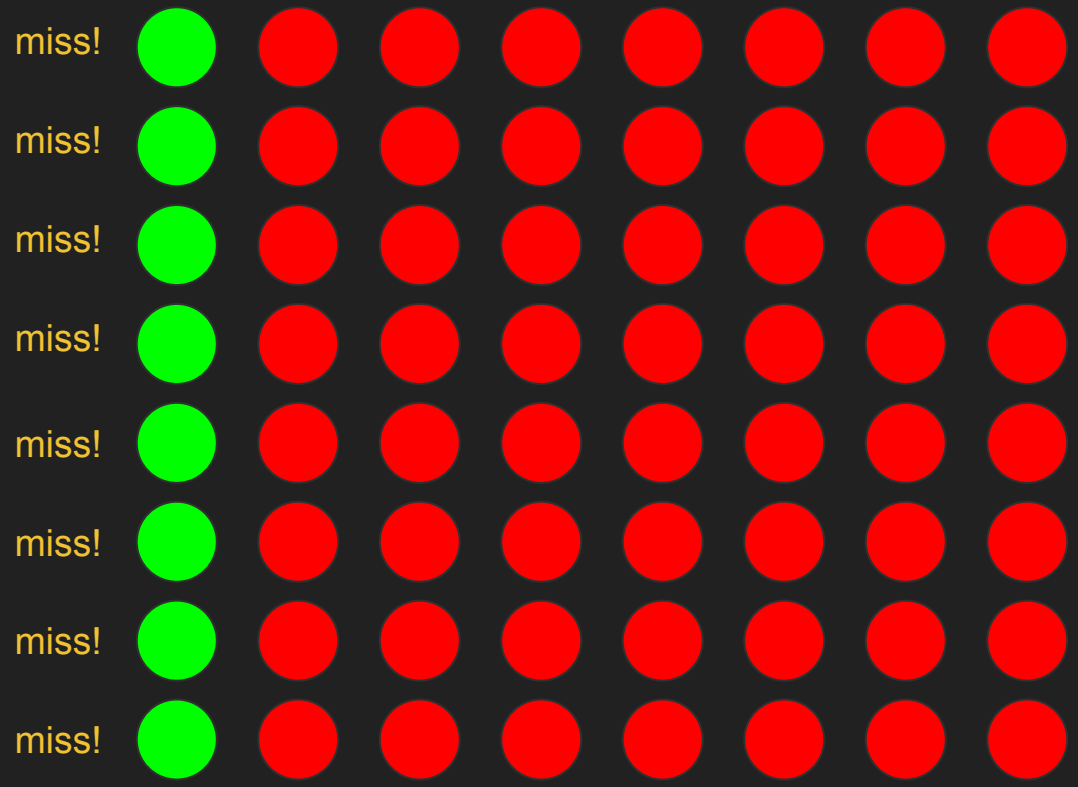
Memory Space



Memory Space

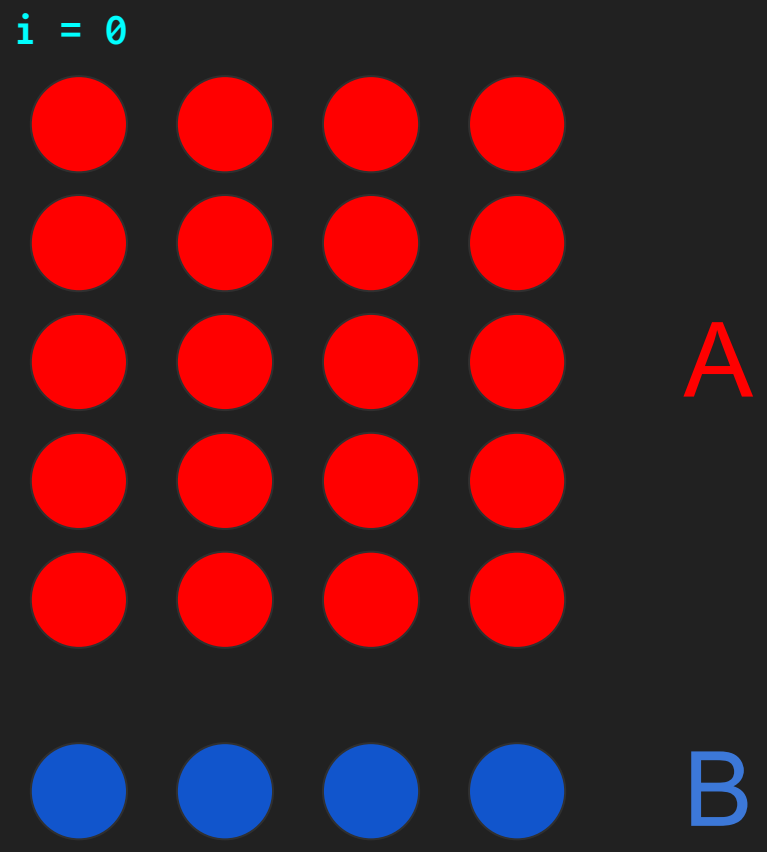


Memory Space

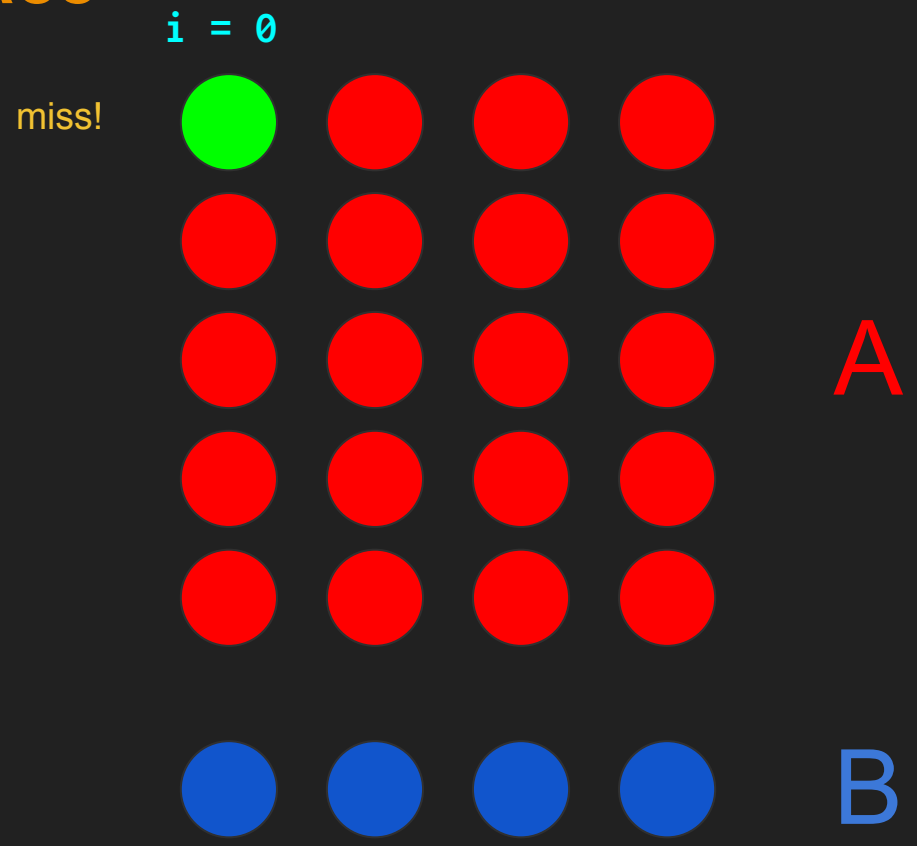


Memory Space

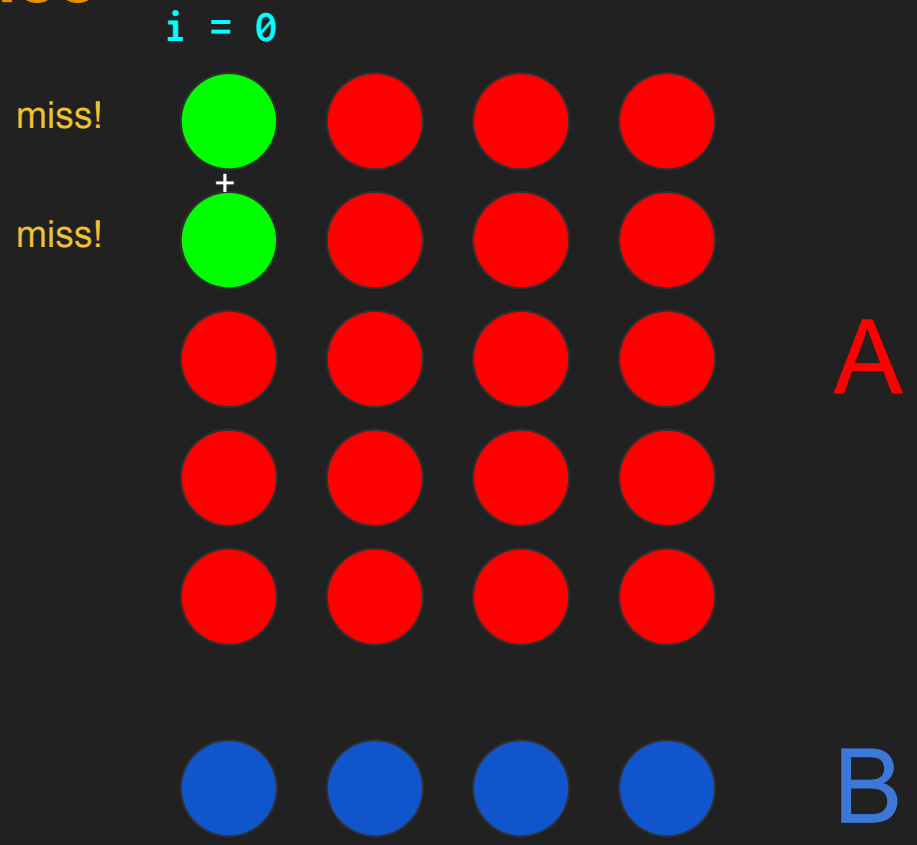
Memory Space



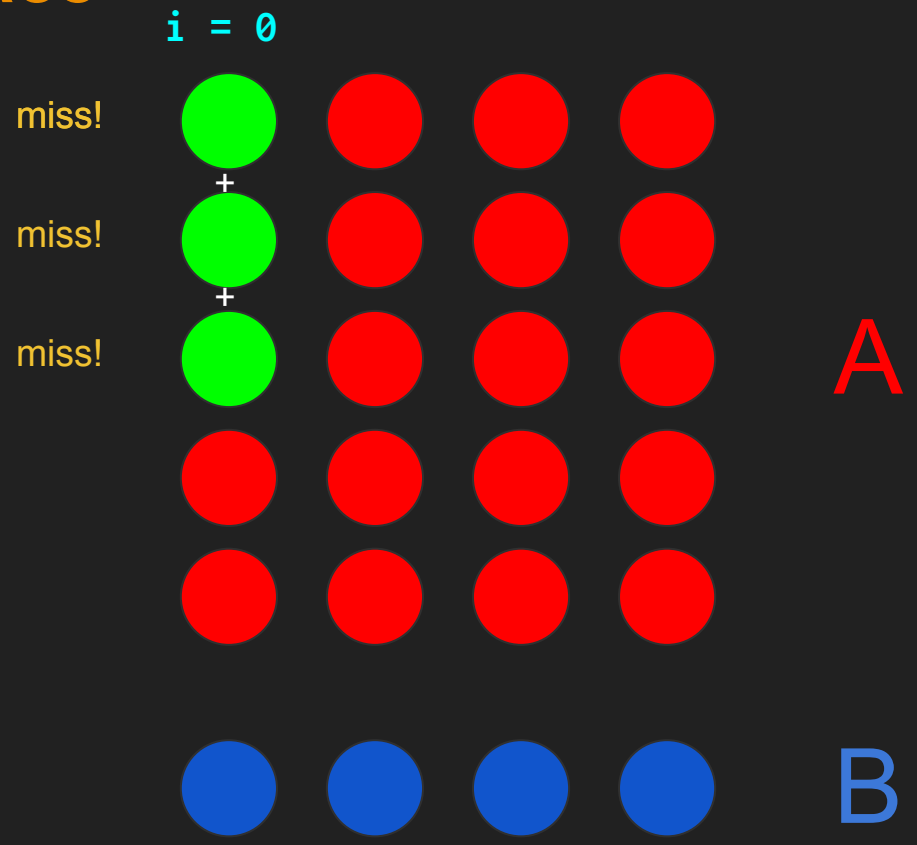
Memory Space



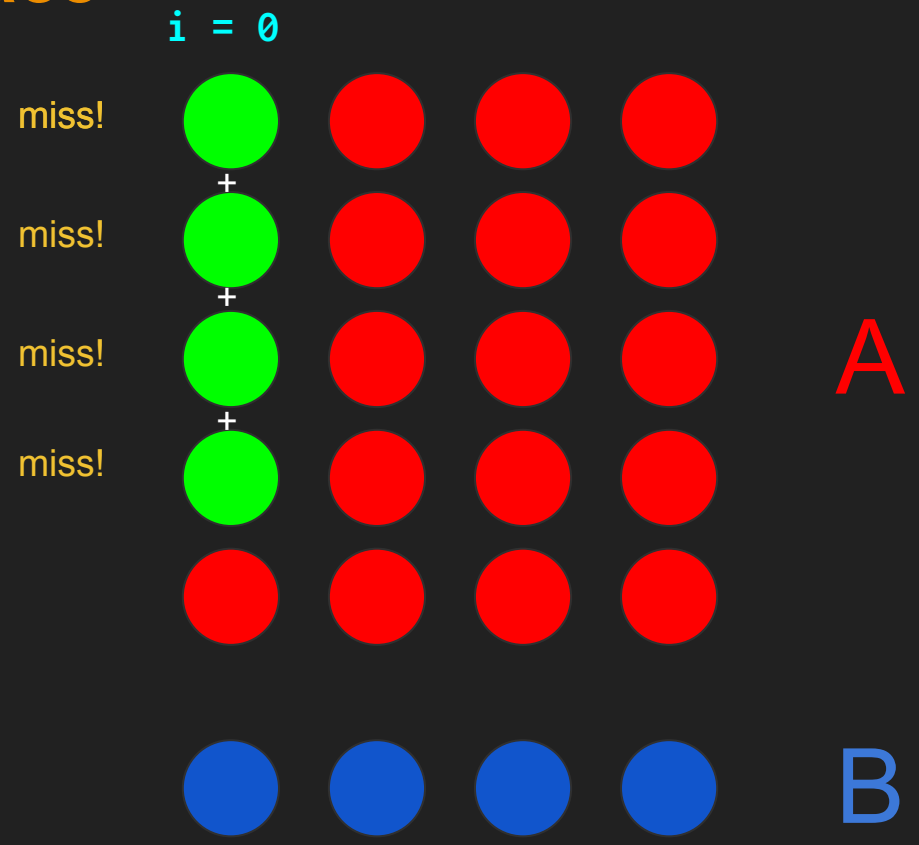
Memory Space



Memory Space



Memory Space

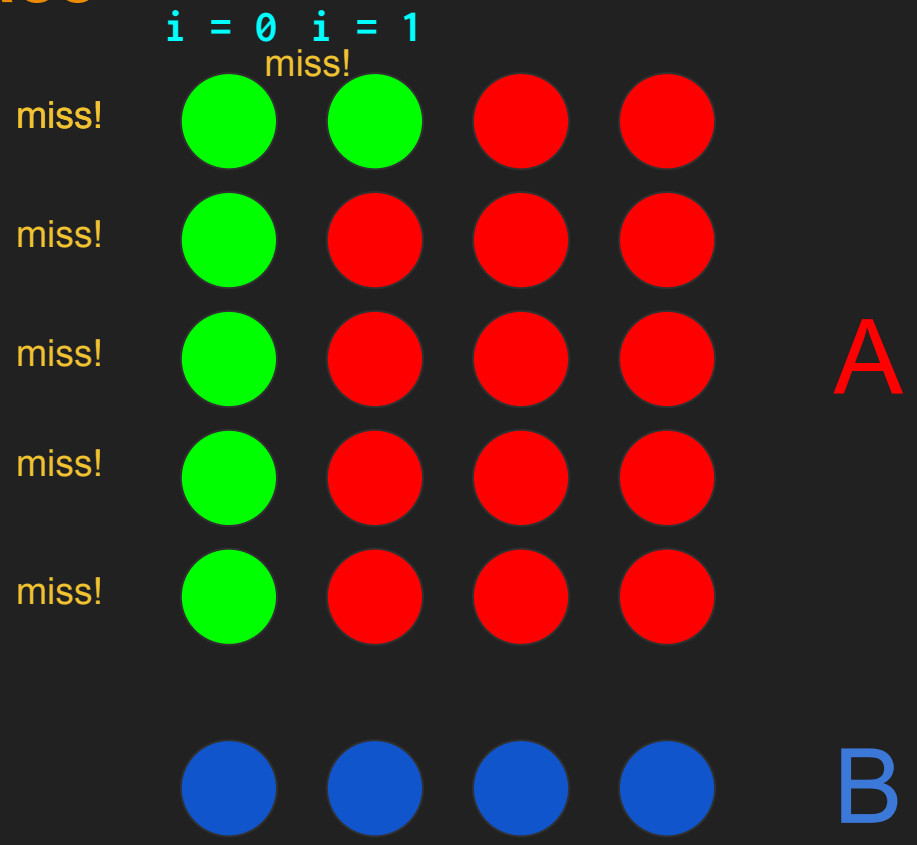


Memory Space

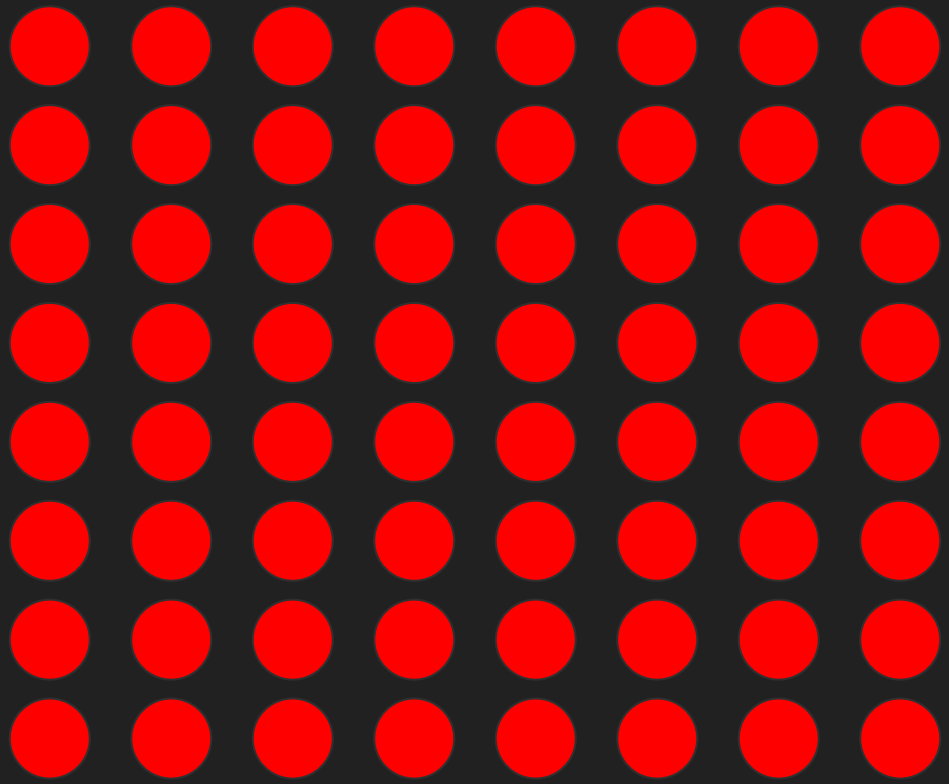
Memory Space

Memory Space

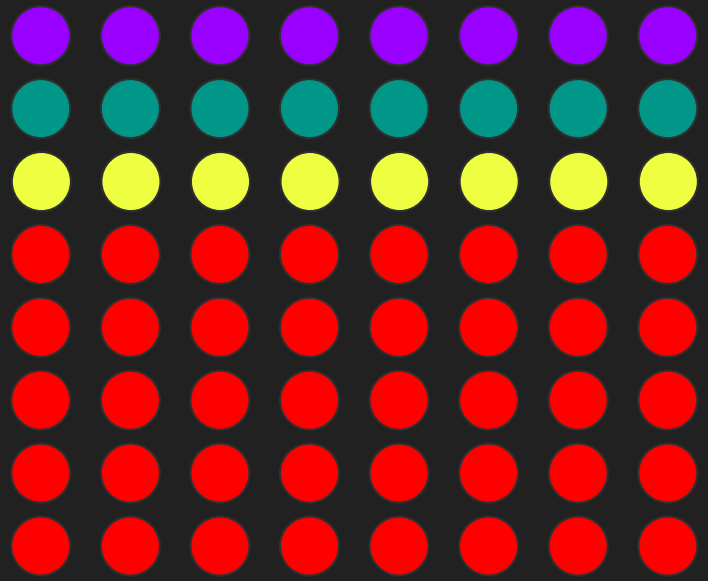
Memory Space



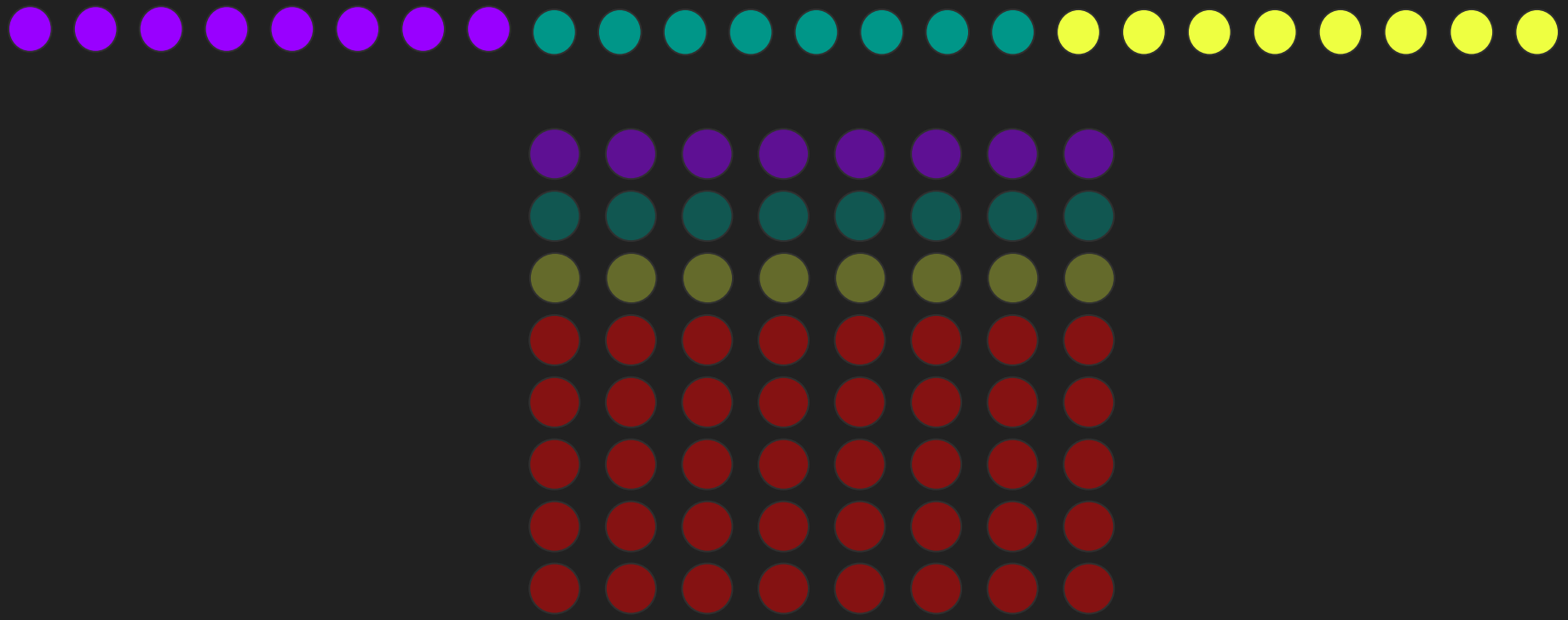
Memory Space



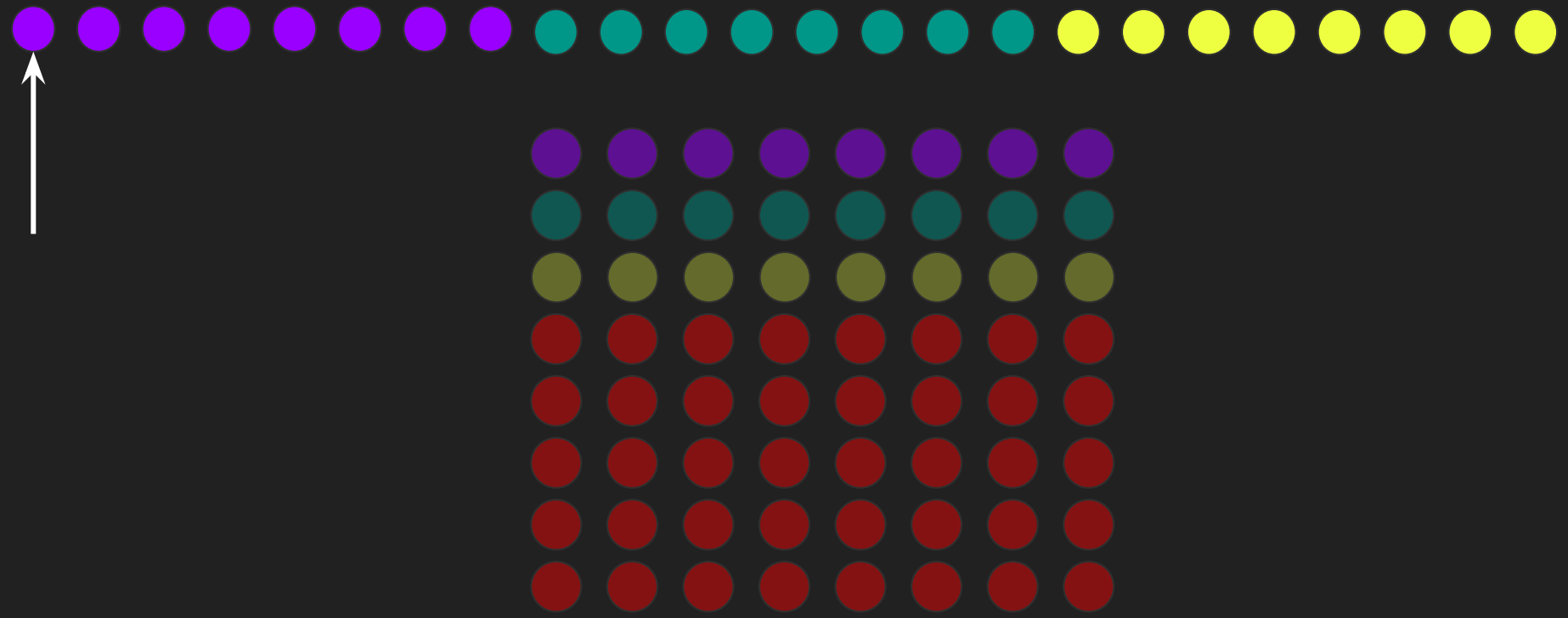
Memory Space



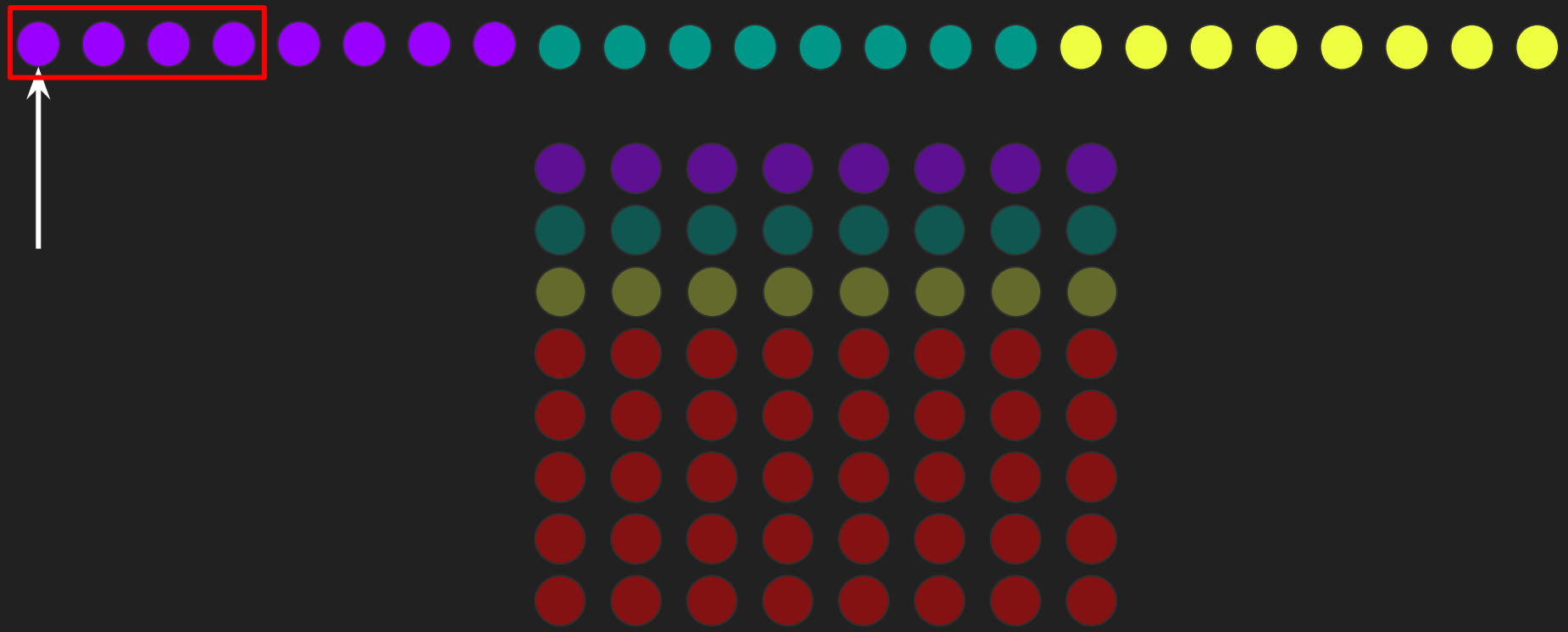
Memory Space



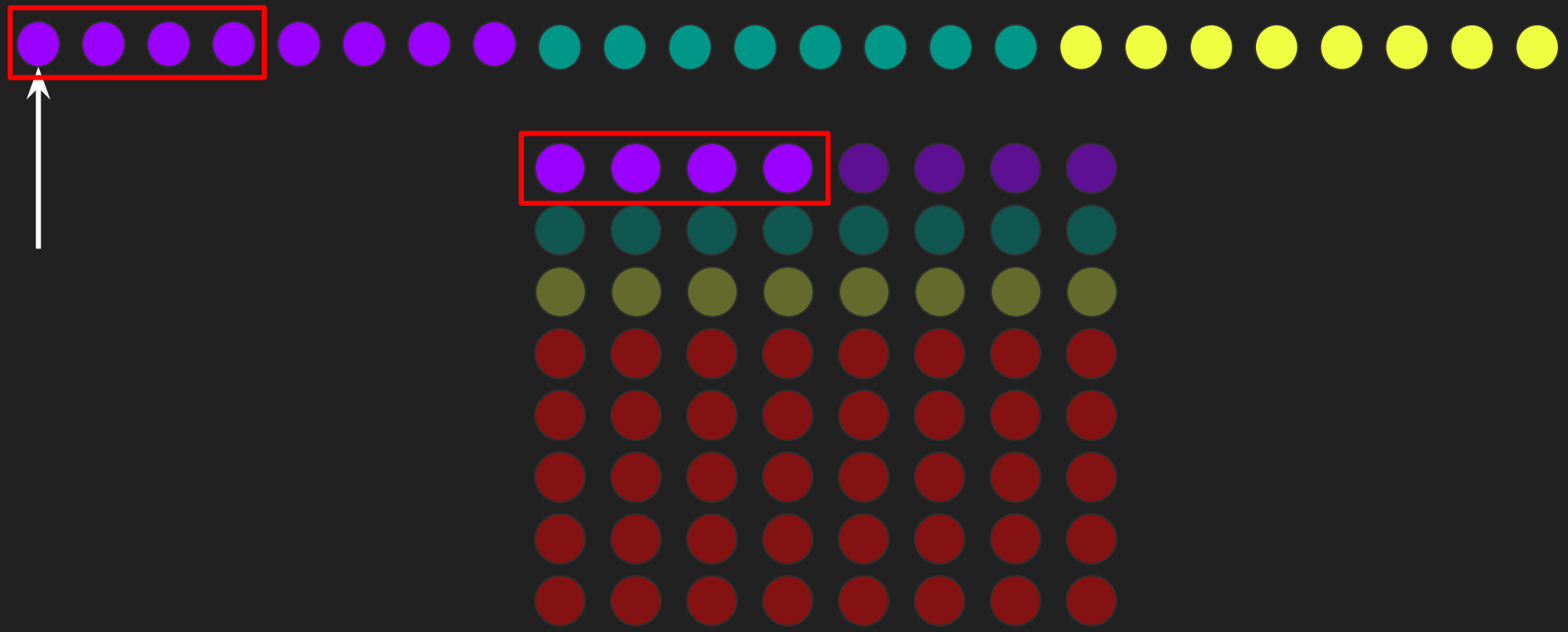
Memory Space



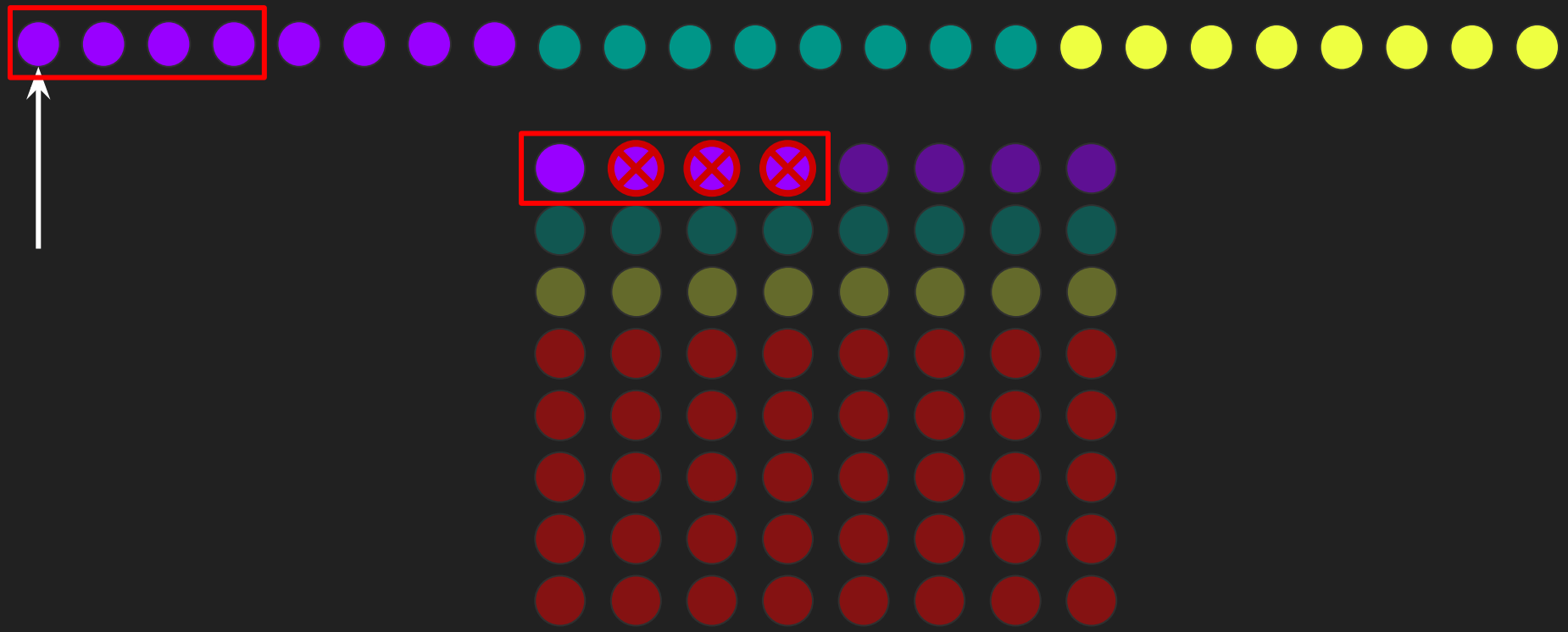
Memory Space



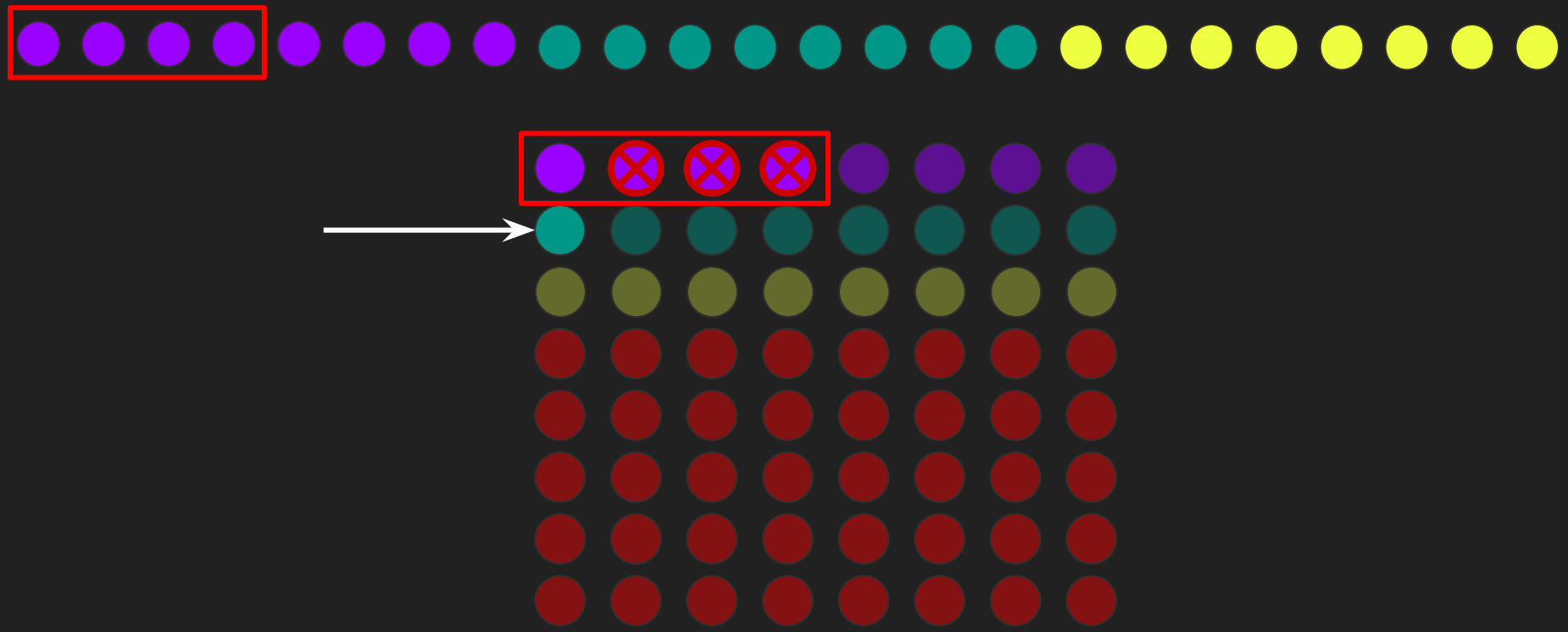
Memory Space



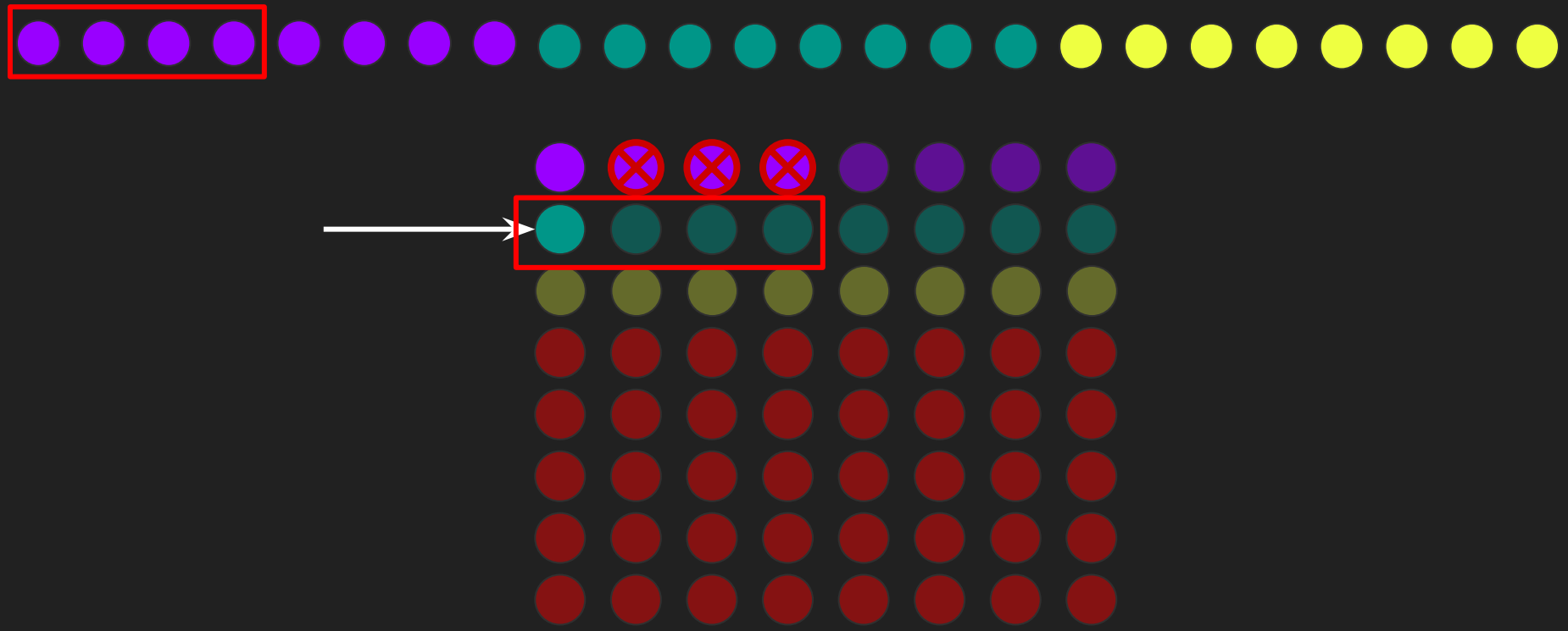
Memory Space



Memory Space



Memory Space



Outer-Loop Vectorization

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a = i;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = i, a1 = i+1, a2 = i+2, a3 = i+3;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < m; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < 2; ++j) {
    int v = A[j][i];
    a += v;
  }
  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j0 = 0;
  int v0 = A[j0][i];
  a += v0;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j0 = 0;
  int v0 = A[j0][i];
  a += v0;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v0 = A[j0][i];
  a += v0;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v0 = A[j0][i];
  a += v0;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v00 = A[j00][i];   int v01 = A[j01][i+1];
  int v02 = A[j02][i+2]; int v03 = A[j03][i+3];
  a += v0;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v00 = A[j00][i];   int v01 = A[j01][i+1];
  int v02 = A[j02][i+2]; int v03 = A[j03][i+3];
  a += v0;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v00 = A[j00][i];   int v01 = A[j01][i+1];
  int v02 = A[j02][i+2]; int v03 = A[j03][i+3];
  a0 += v00; a1 += v01; a2 += v02; a3 += v03;

  int j1 = 1;
  int v1 = A[j1][i];
  a += v1;

  B[i] = a;
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v00 = A[j00][i];   int v01 = A[j01][i+1];
  int v02 = A[j02][i+2]; int v03 = A[j03][i+3];
  a0 += v00; a1 += v01; a2 += v02; a3 += v03;

  int j10 = 1, j11 = 1, j12 = 1, j13 = 1;
  int v10 = A[j10][i];   int v11 = A[j11][i+1];
  int v12 = A[j12][i+2]; int v13 = A[j13][i+3];
  a0 += v10; a1 += v11; a2 += v12; a3 += v13;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j00 = 0, j01 = 0, j02 = 0, j03 = 0;
  int v00 = A[j00][i];   int v01 = A[j01][i+1];
  int v02 = A[j02][i+2]; int v03 = A[j03][i+3];
  a0 += v00; a1 += v01; a2 += v02; a3 += v03;

  int j10 = 1, j11 = 1, j12 = 1, j13 = 1;
  int v10 = A[j10][i];   int v11 = A[j11][i+1];
  int v12 = A[j12][i+2]; int v13 = A[j13][i+3];
  a0 += v10; a1 += v11; a2 += v12; a3 += v13;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j0 = 0;
  int v00 = A[j0][i];   int v01 = A[j0][i+1];
  int v02 = A[j0][i+2]; int v03 = A[j0][i+3];
  a0 += v00; a1 += v01; a2 += v02; a3 += v03;

  int j10 = 1, j11 = 1, j12 = 1, j13 = 1;
  int v10 = A[j10][i];   int v11 = A[j11][i+1];
  int v12 = A[j12][i+2]; int v13 = A[j13][i+3];
  a0 += v10; a1 += v11; a2 += v12; a3 += v13;

  B[i] = a;
}
```

Outer-Loop Hand-Vectorized

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;

  int j0 = 0;
  int v00 = A[j0][i];   int v01 = A[j0][i+1];
  int v02 = A[j0][i+2]; int v03 = A[j0][i+3];
  a0 += v00; a1 += v01; a2 += v02; a3 += v03;

  int j1 = 1;
  int v10 = A[j1][i];   int v11 = A[j1][i+1];
  int v12 = A[j1][i+2]; int v13 = A[j1][i+3];
  a0 += v10; a1 += v11; a2 += v12; a3 += v13;

  B[i] = a;
}
```

Sum Columns Original

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < m; ++j) {
    int v0 = A[j][i];   int v1 = A[j][i+1];
    int v2 = A[j][i+2]; int v3 = A[j][i+3];

    a0 += v0; a1 += v1; a2 += v2; a3 += v3;
  }
  B[i] = a;
}
```

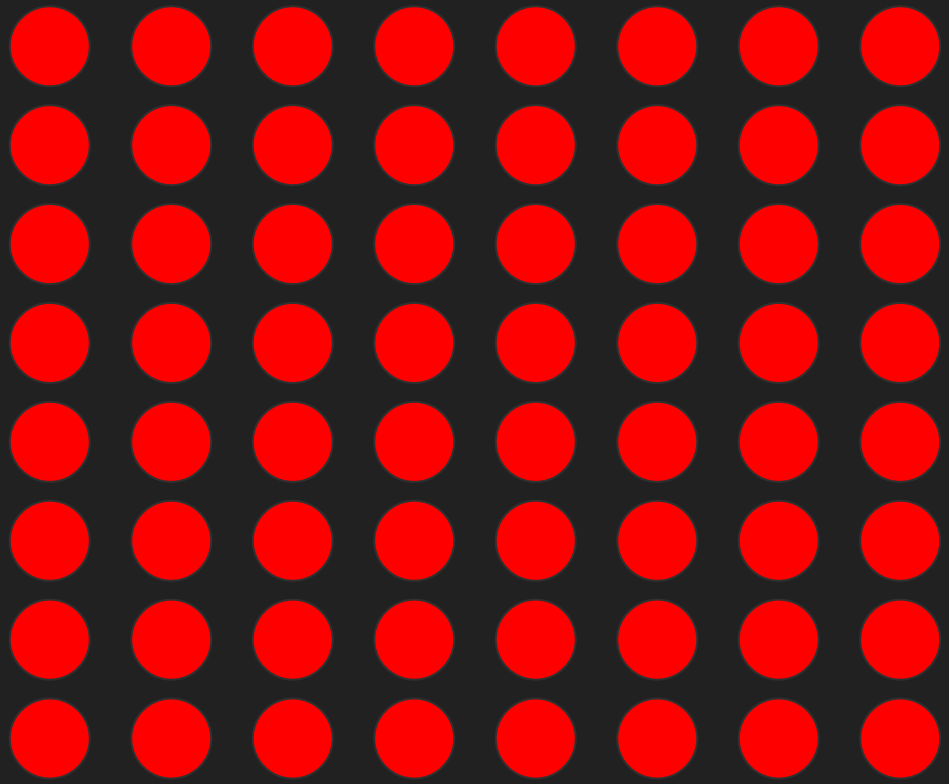
Sum Columns Original

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < m; ++j) {
    int v0 = A[j][i];   int v1 = A[j][i+1];
    int v2 = A[j][i+2]; int v3 = A[j][i+3];

    a0 += v0; a1 += v1; a2 += v2; a3 += v3;
  }
  B[i] = a;
}
```

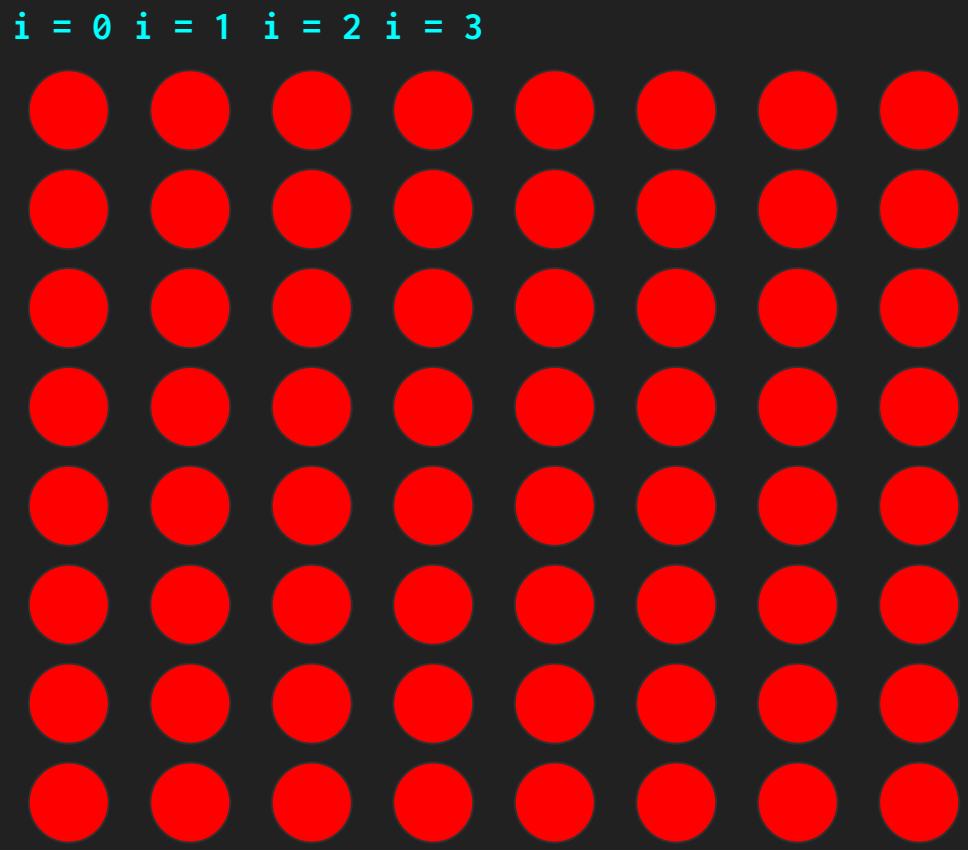
Sum Columns Original

```
for (int i = 0; i < n; i += 4) {
  int a0 = 0, a1 = 0, a2 = 0, a3 = 0;
  for (int j = 0; j < m; ++j) {
    int v0 = A[j][i];   int v1 = A[j][i+1];
    int v2 = A[j][i+2]; int v3 = A[j][i+3];

    a0 += v0; a1 += v1; a2 += v2; a3 += v3;
  }
  B[i] = a0;   B[i+1] = a1;
  B[i+2] = a2; B[i+3] = a3;
}
```

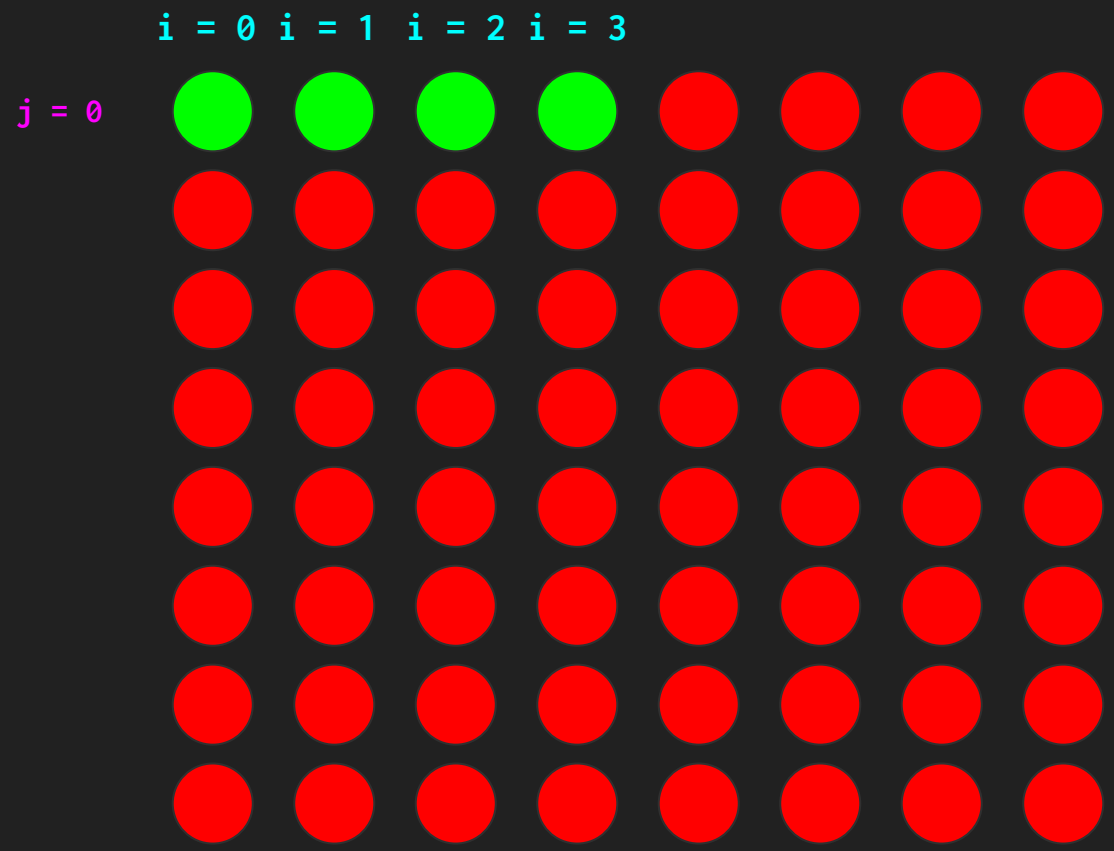
Memory Space



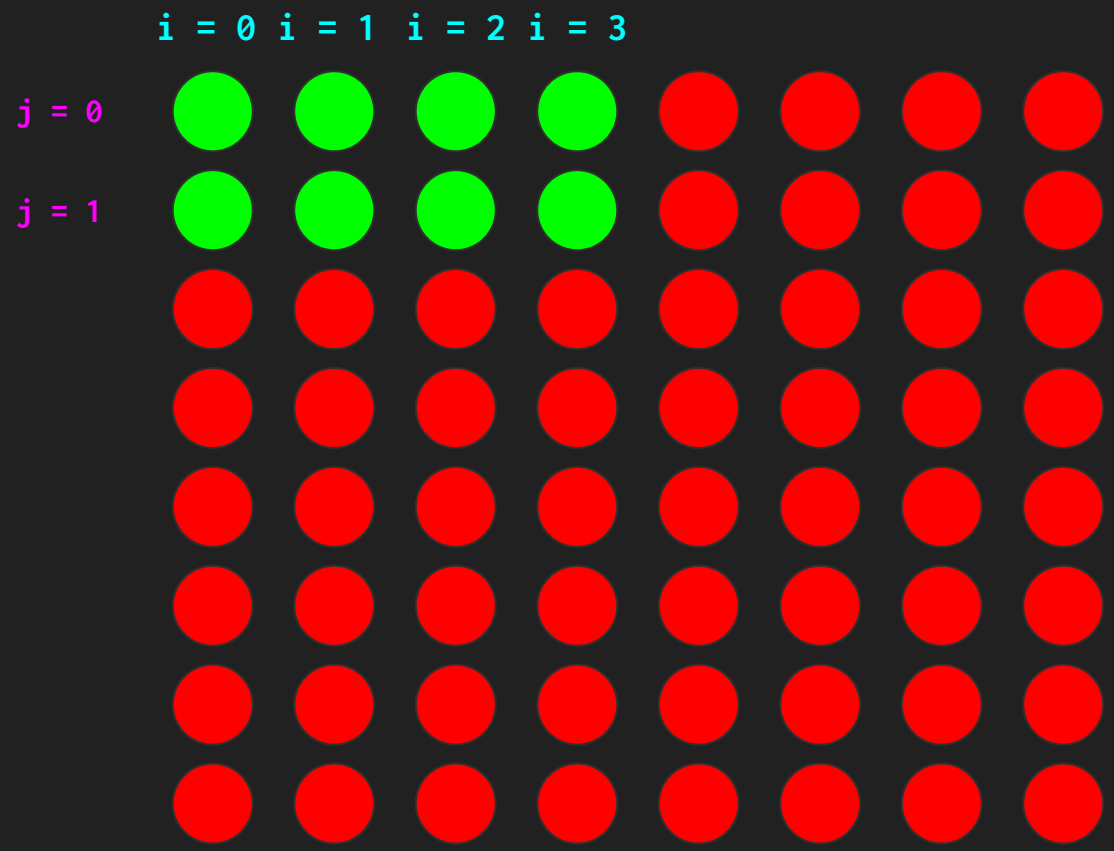
Memory Space



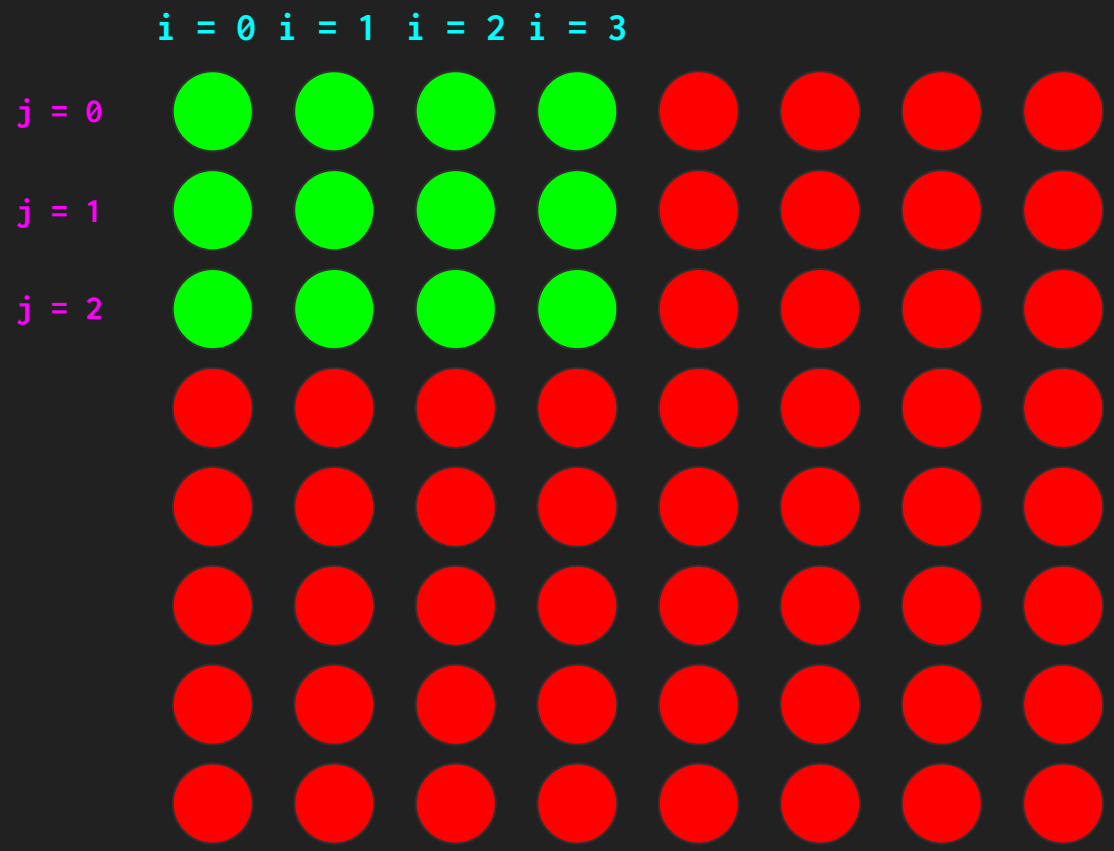
Memory Space



Memory Space

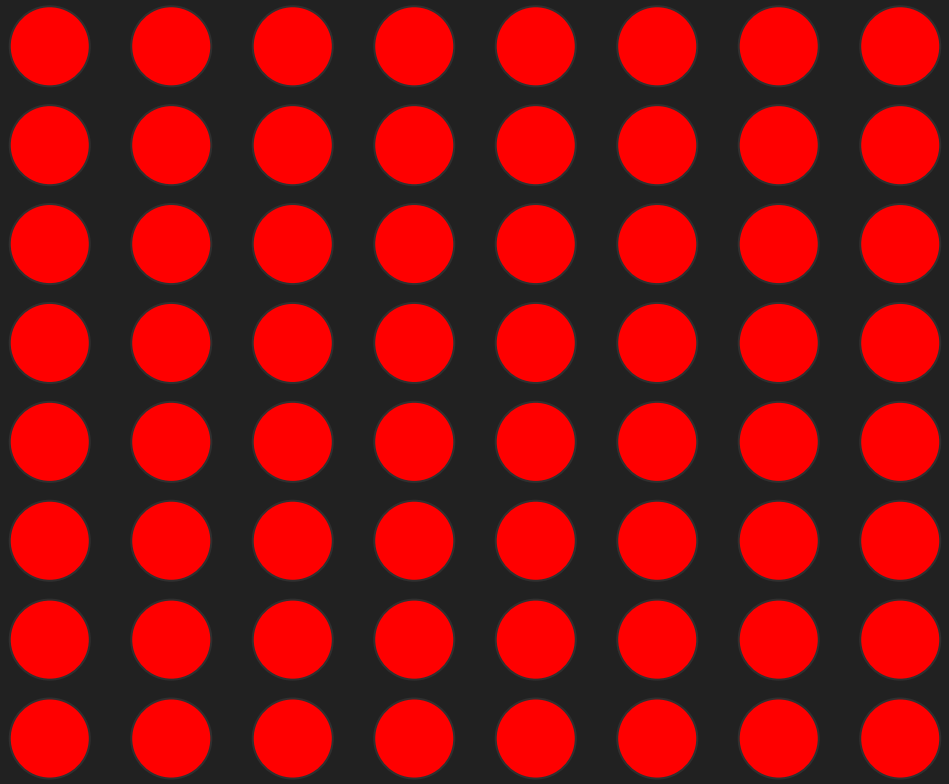


Memory Space

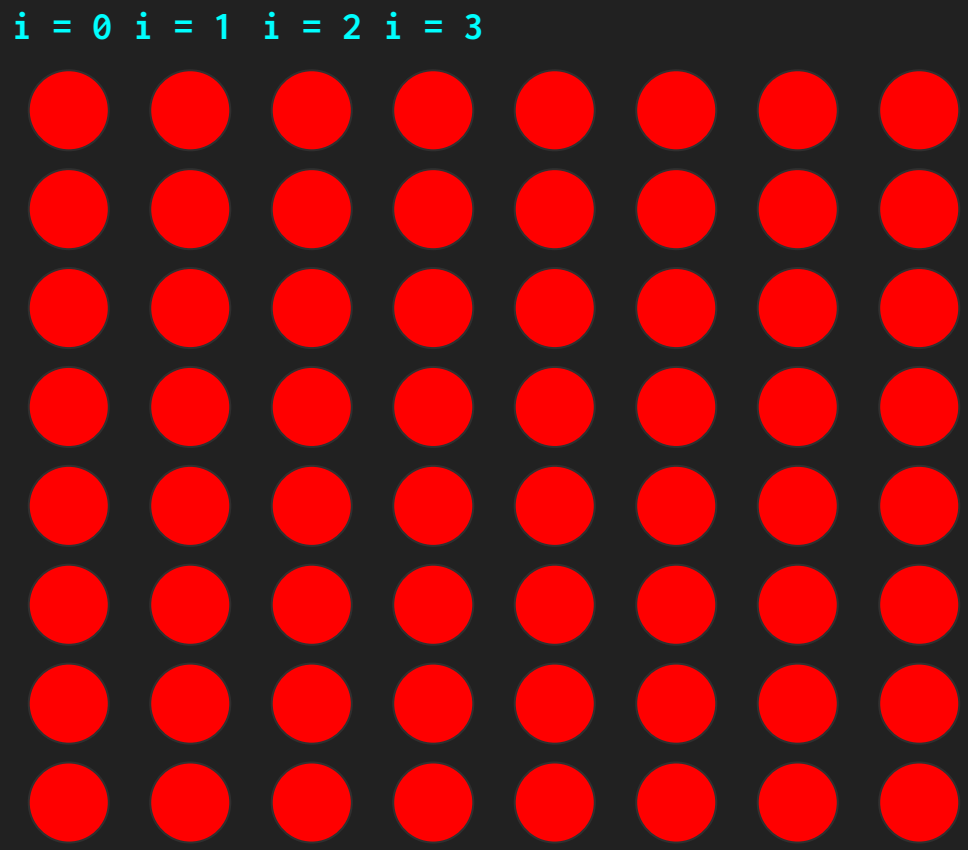


Outer-Loop Parallelization (i.e., Multi-threading)

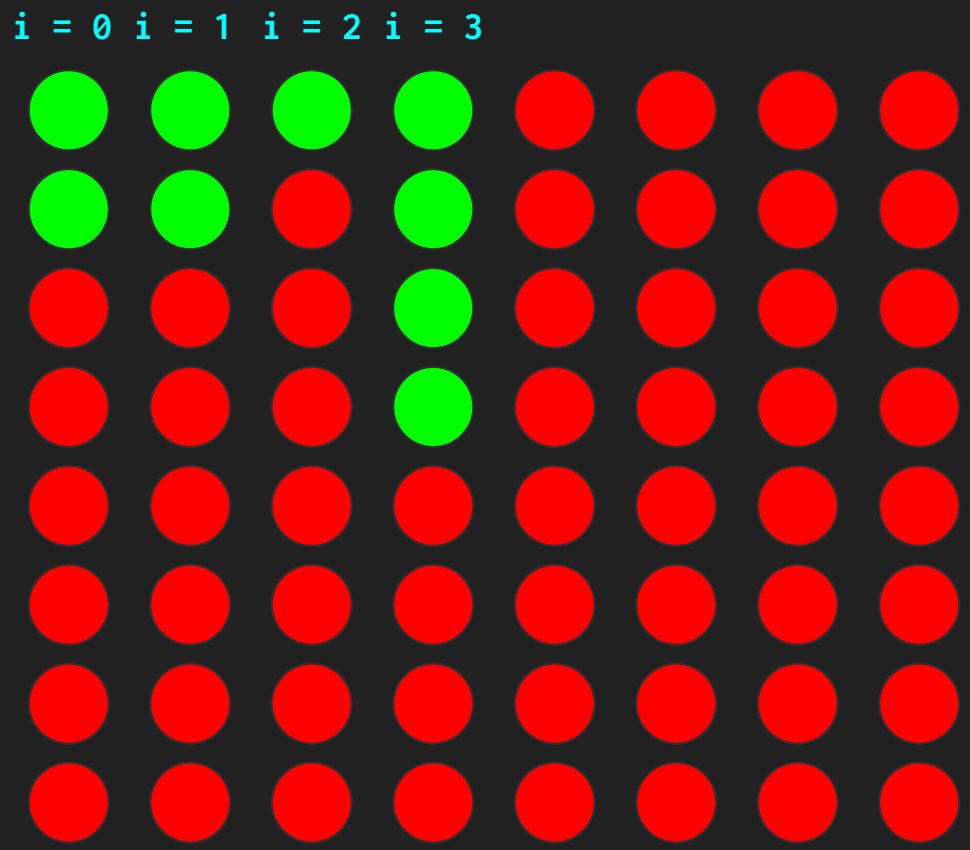
Memory Space



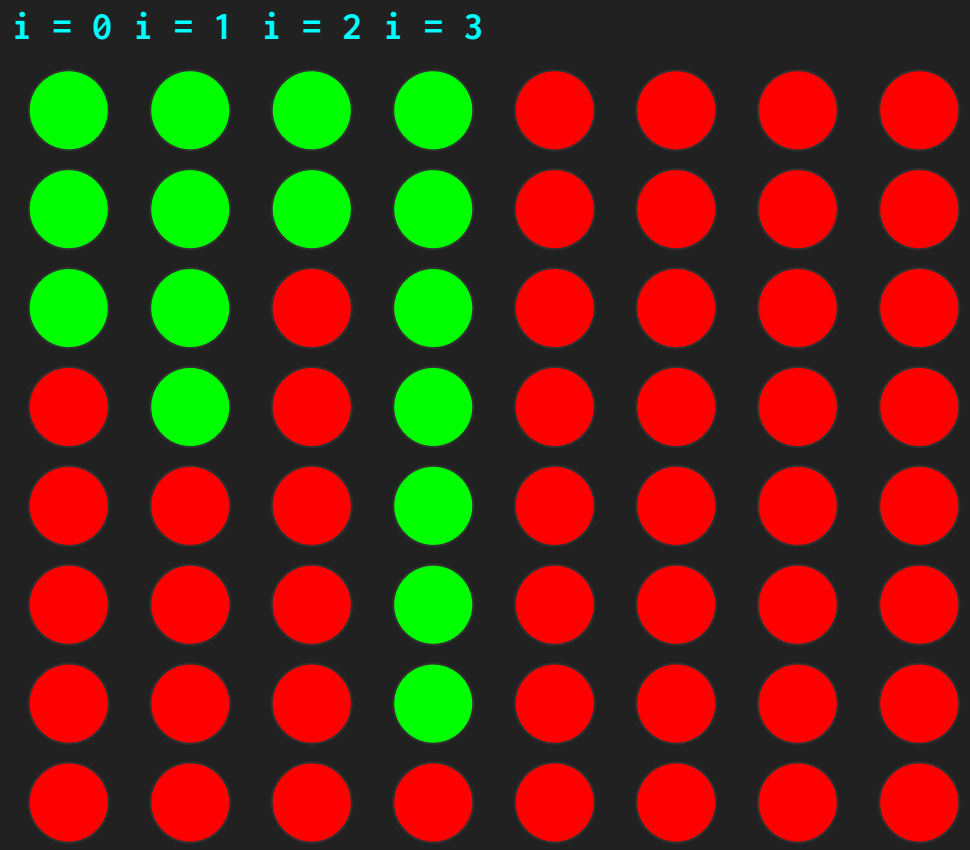
Memory Space



Memory Space



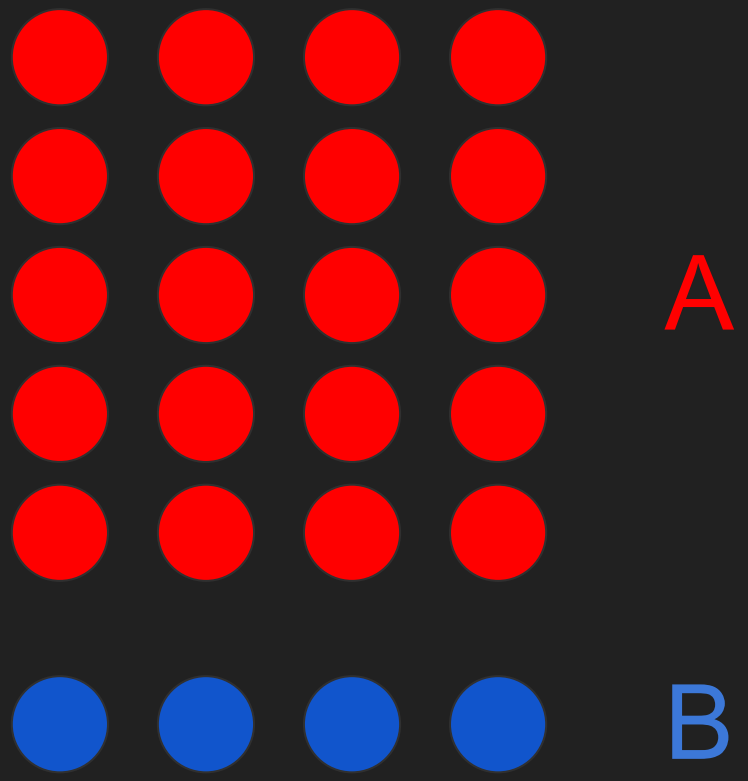
Memory Space



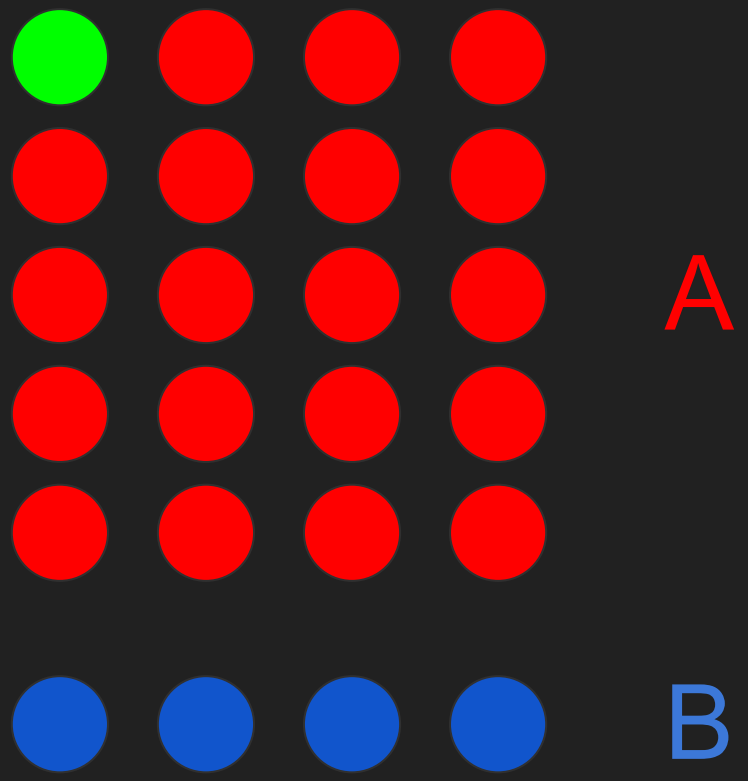
Sum Columns Conditionally

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    bool p = C[j];
    if (p) {
      int v = A[j][i];
      a += v;
    }
  }
  B[i] = a;
}
```

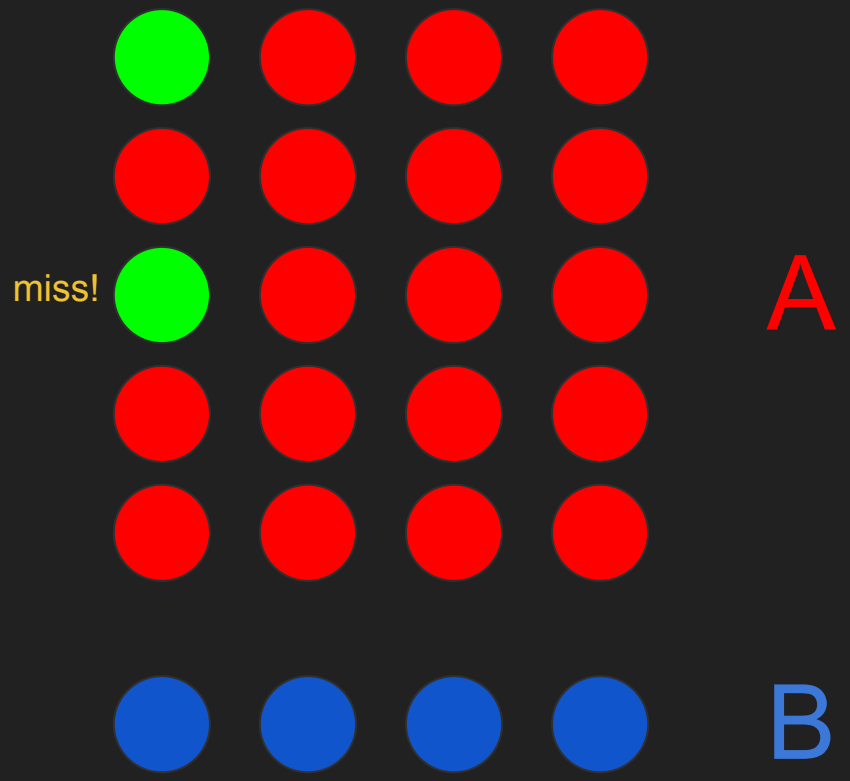
Memory Space



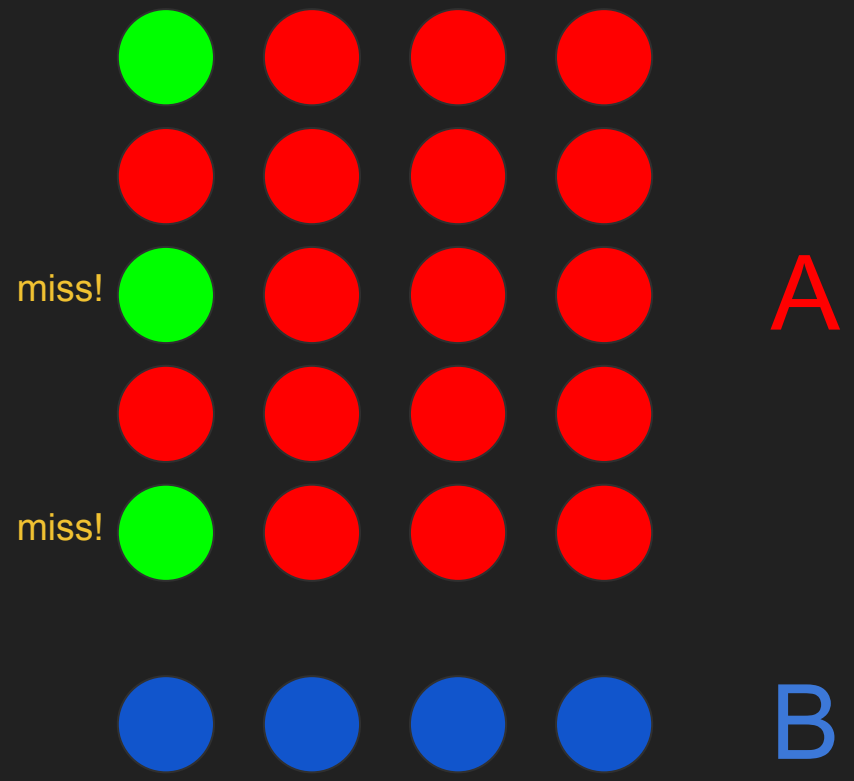
Memory Space



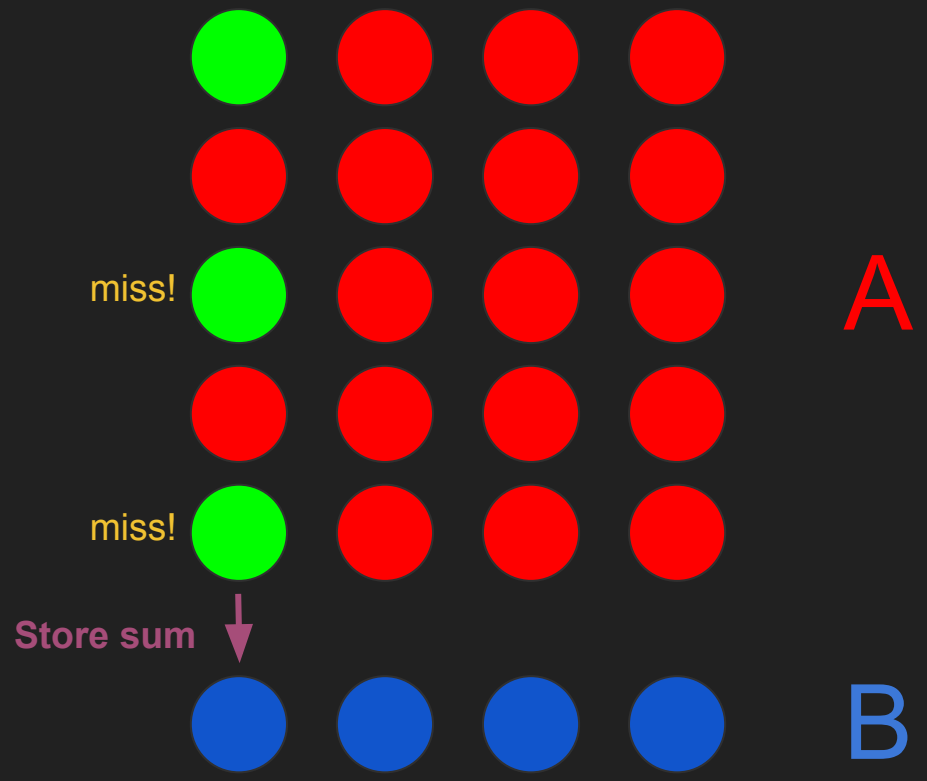
Memory Space



Memory Space



Memory Space



Uniformity

A uniform value is characterized in relation to a loop and is one that does <u>not</u> vary *because of this loop*.

Divergence

```
Divergent = !Uniform
```

Uniformity in Innermost Loops

```
for (int i = 0; i < N; ++i) {
  int v = 1 + 2;
  a[i] = v;
}
```

Uniformity in Innermost Loops

```
for (int i = 0; i < N; ++i) {
  int v = 1 + 2;
  a[i] = v;
}
```

v is uniform and also loop-invariant

Hoist Out

```
int v = 1 + 2;
for (int i = 0; i < N; ++i) {
  a[i] = v;
}
```

Conditional Code

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    bool p = C[j];
    if (p) {
      int v = A[j][i];
      a += v;
    }
  }
  B[i] = a;
}
```

Conditional Code

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    bool p = C[j];
    if (p) {
      int v = A[j][i];
      a += v;
    }
  }
  B[i] = a;
}
```

What is p in the i-loop?

Conditional Code

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    bool p = C[j];
    if (p) {
      int v = A[j][i];
      a += v;
    }
  }
  B[i] = a;
}
```

What is p in the i-loop?

Uniform? ✓

Conditional Code

```
for (int i = 0; i < n; ++i) {
  int a = 0;
  for (int j = 0; j < m; ++j) {
    bool p = C[j];
    if (p) {
      int v = A[j][i];
      a += v;
    }
  }
  B[i] = a;
}
```

What is p in the i-loop?

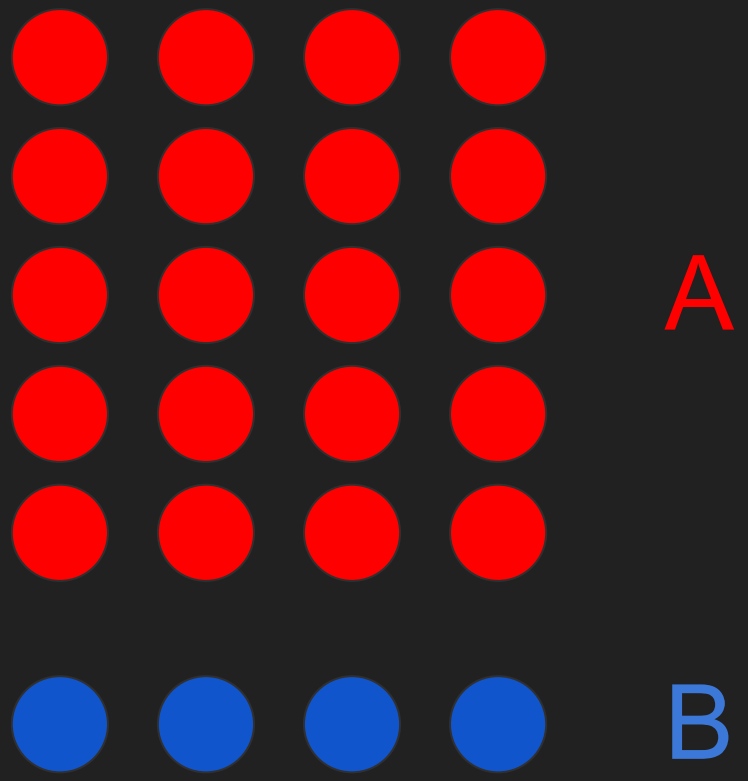
Uniform? ✓

Loop-Invariant? ✗

Uniformity Enables Outer-Loop Vectorization

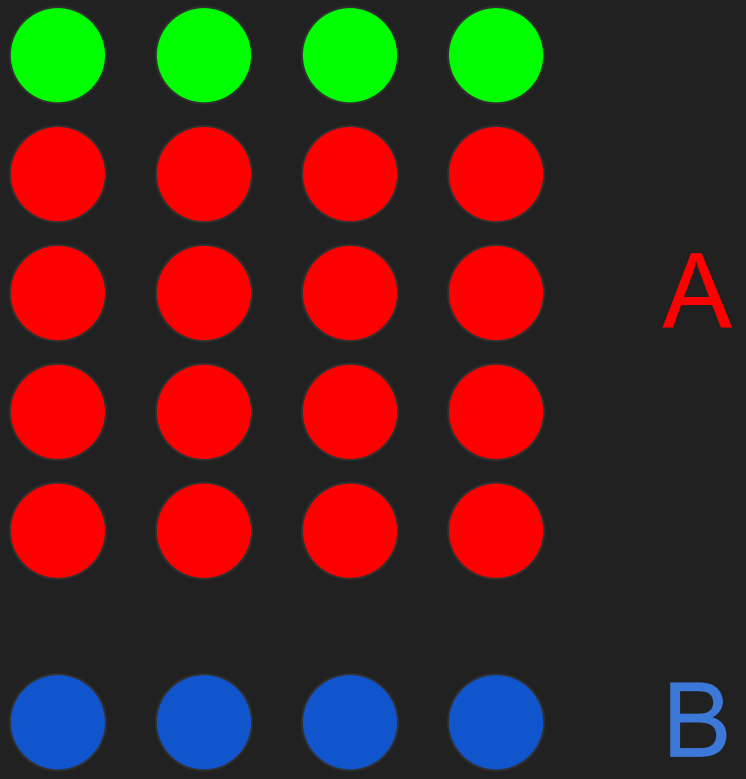
```
for (int i = 0; i < end; i += 4) {  // vectorized
  __m128i a = [0,0,0,0];
  for (int j = 0; j < m; ++j) {
    bool p = C[j];
    if (p) {
      __m128i v = loadu4(&A[j][i];);
      a = add4(a, v);
    }
  }
  storeu4(&B[i], a);
}
… residual
```

Memory Space



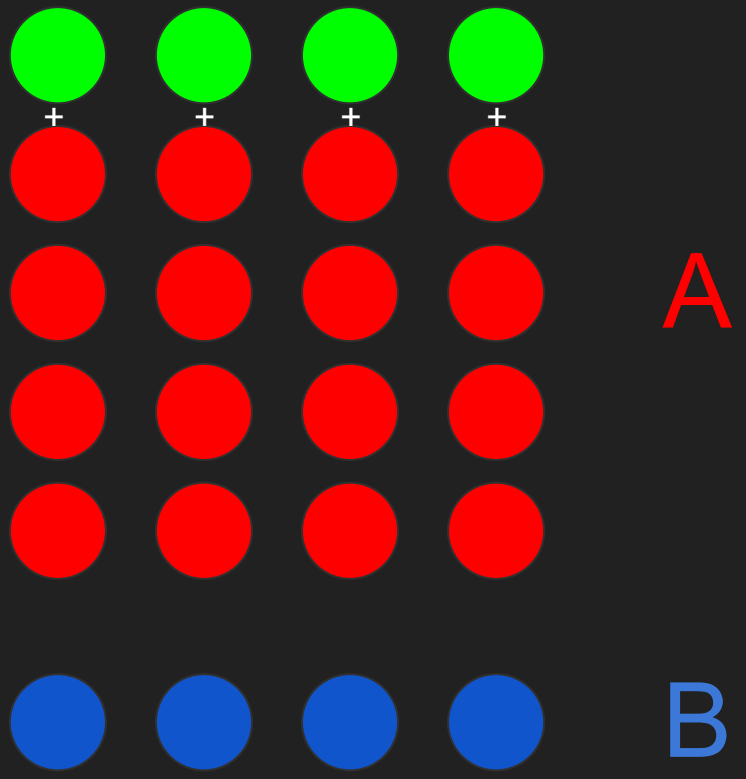
Memory Space

```
p = true
```



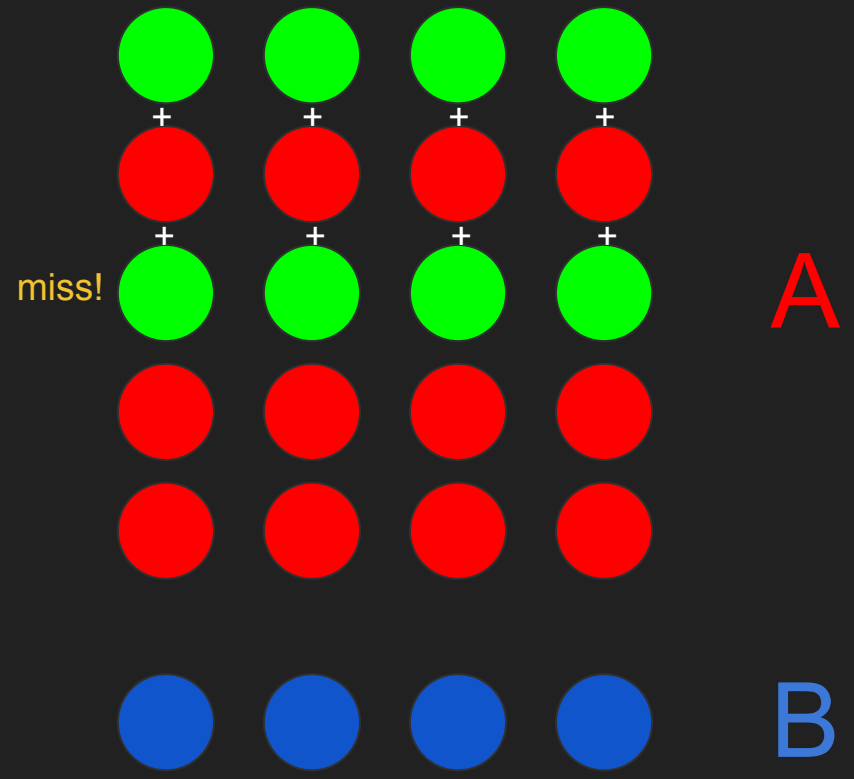
Memory Space

```
p = true

p = false
```



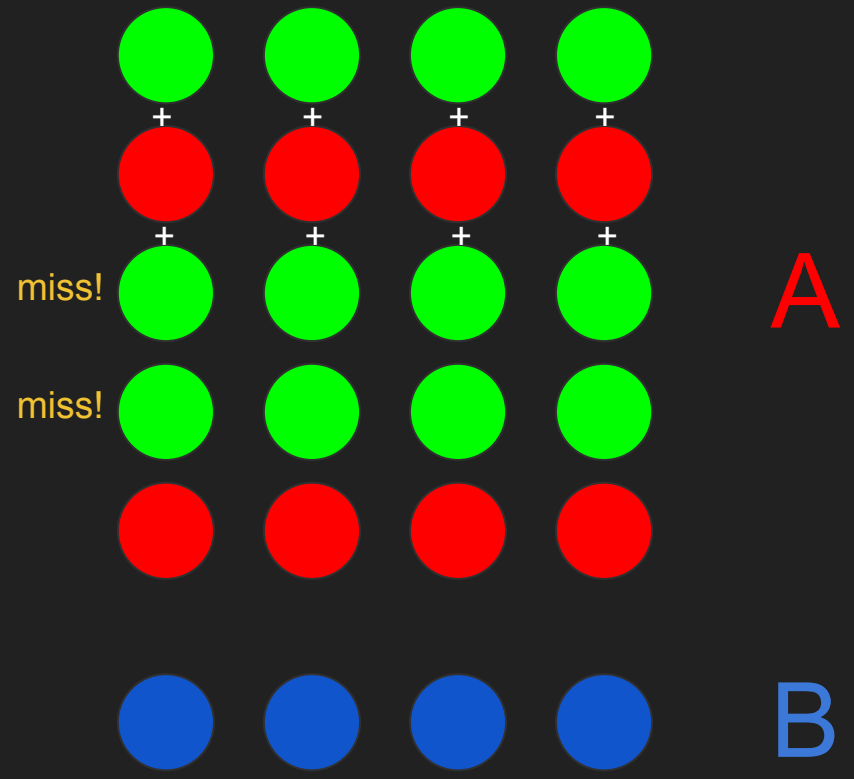
Memory Space

```
p = true
p = false
p = true
```

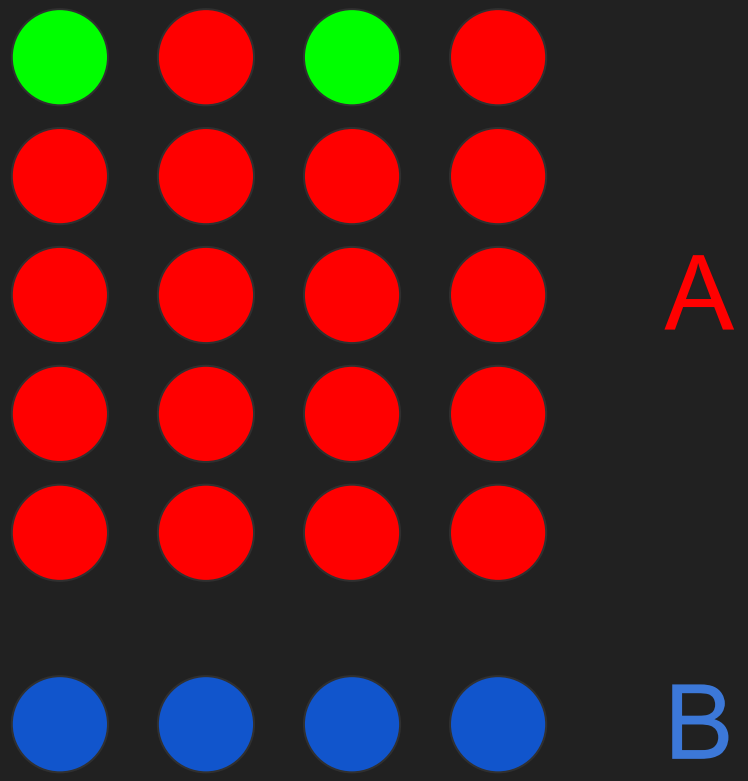


Memory Space

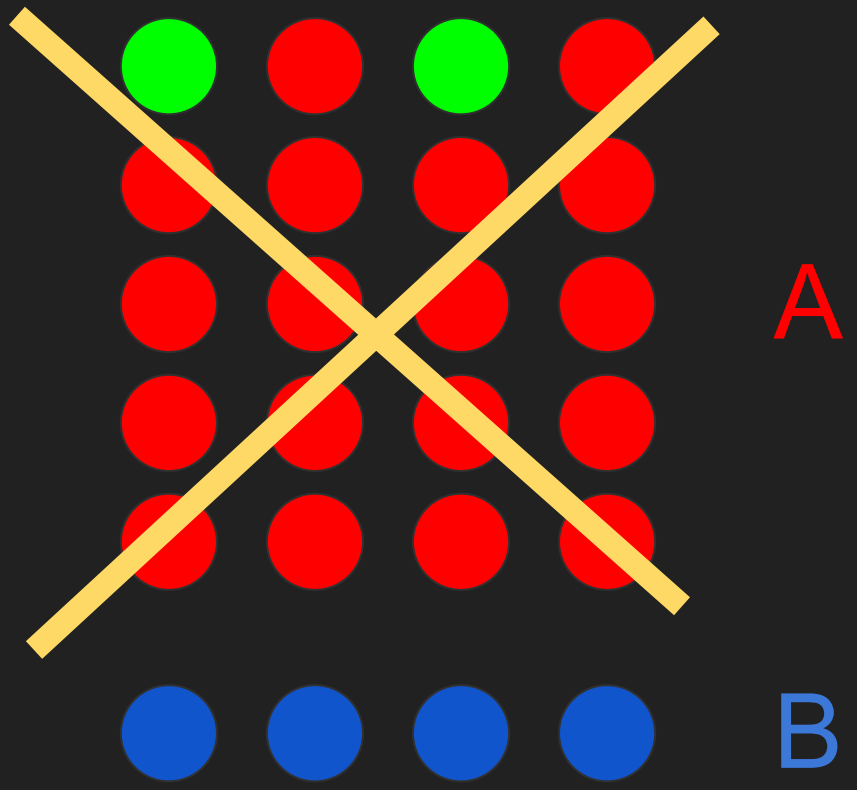
```
p = true
p = false
p = true
p = true
```



Memory Space



Memory Space



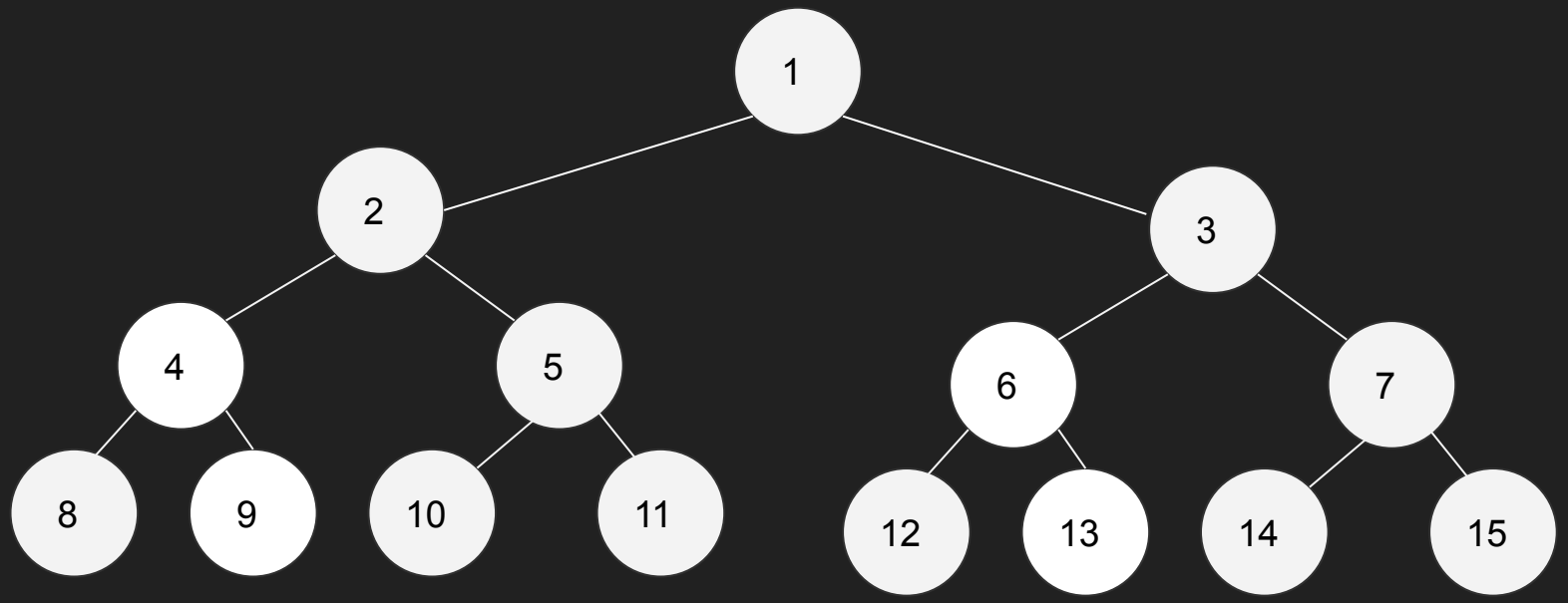
Region Vectorizer (RV)

RV provides a unified interface to vectorize code regions, such as inner and outer loops, up to whole functions.

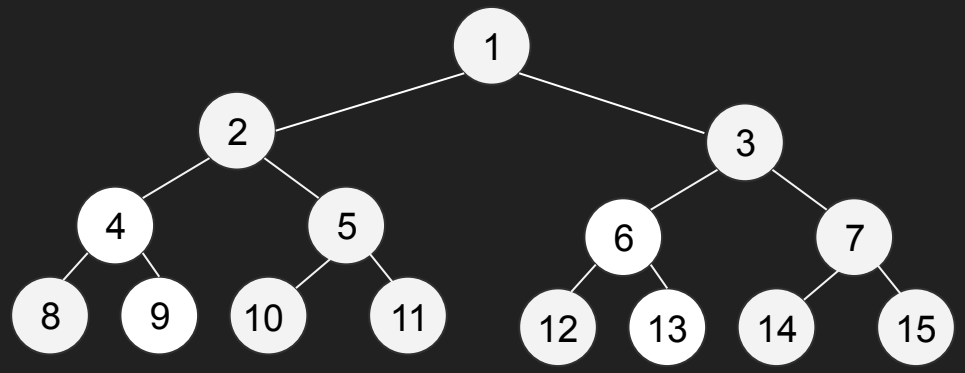
https://github.com/cdl-saarland/rv

Recursive Tree Traversal Vectorization

Tree



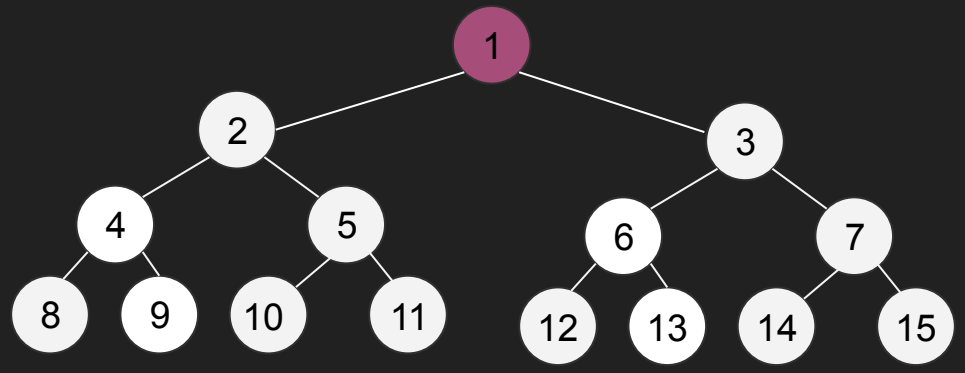
Tree



Nodes

	1	2	4	8	9	5	10	11	3	6	12	13	7	14	15
Traversals															
A															
B															
C															
D															

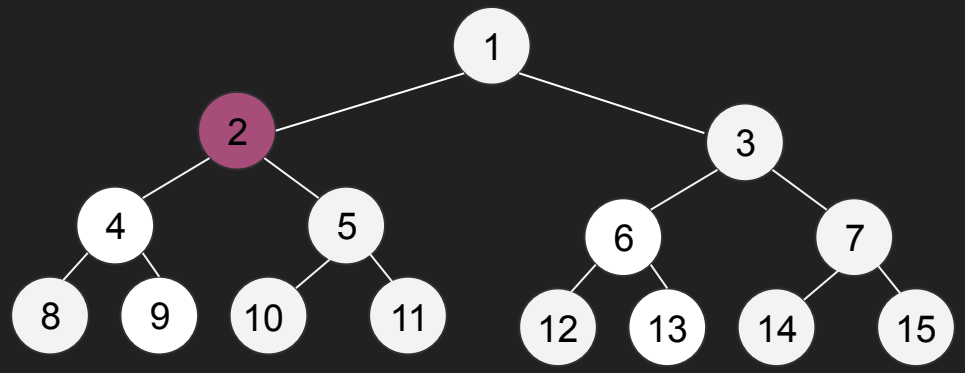
Tree

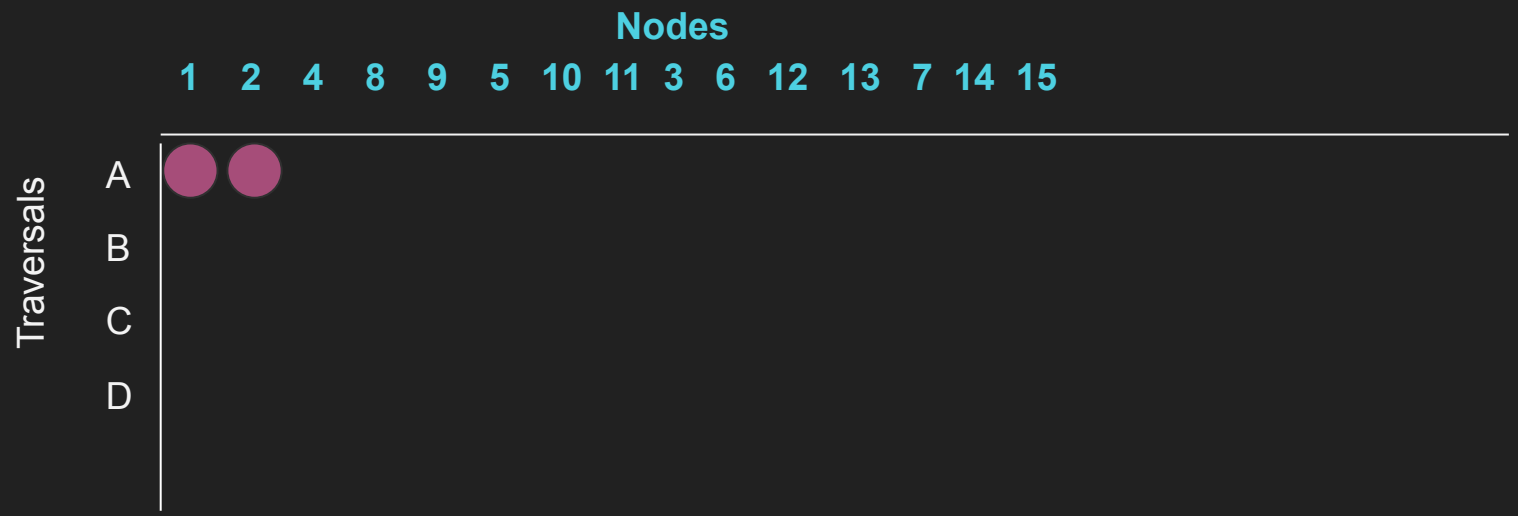


Nodes

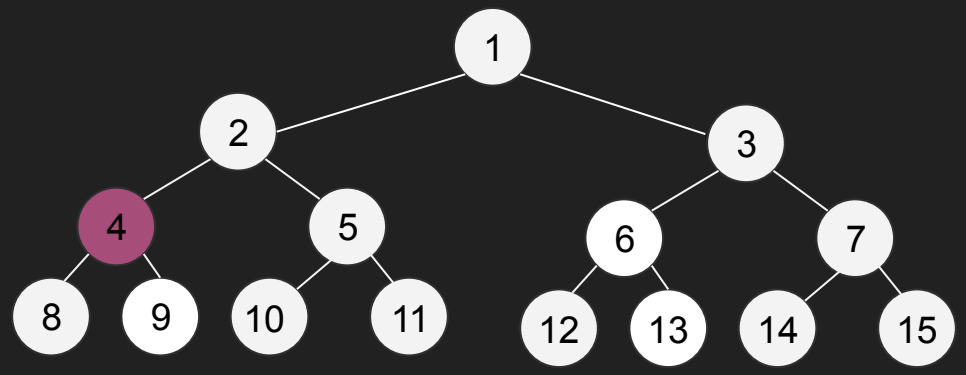
	1	2	4	8	9	5	10	11	3	6	12	13	7	14	15
Traversals															
A															
B															
C															
D															

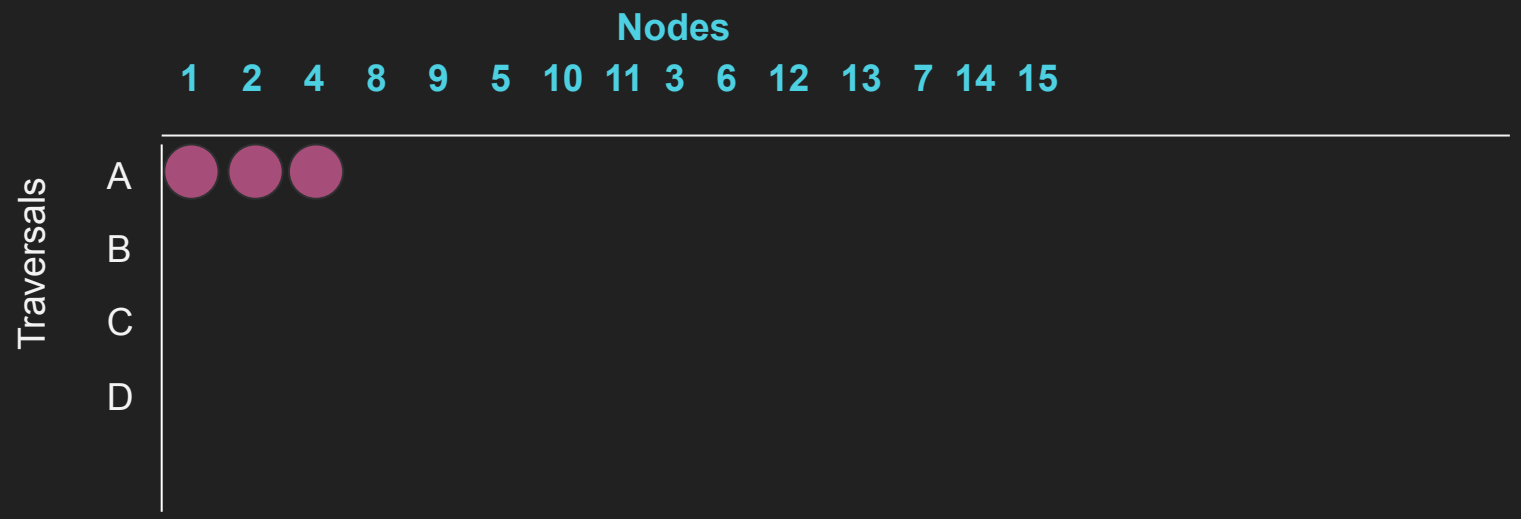
Tree


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

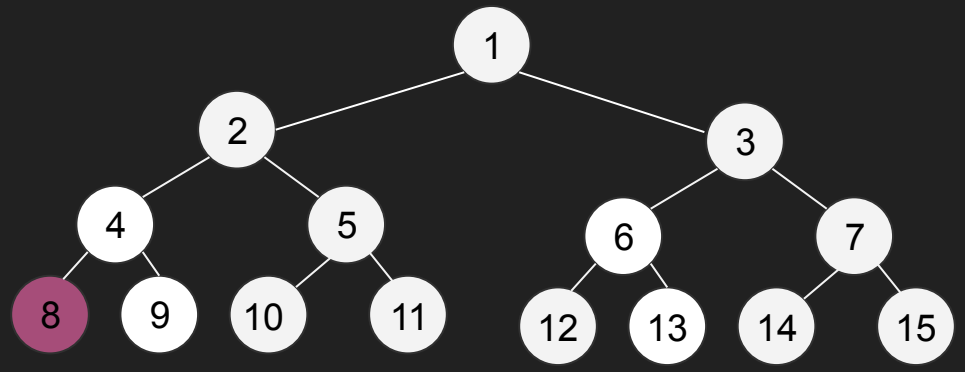

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

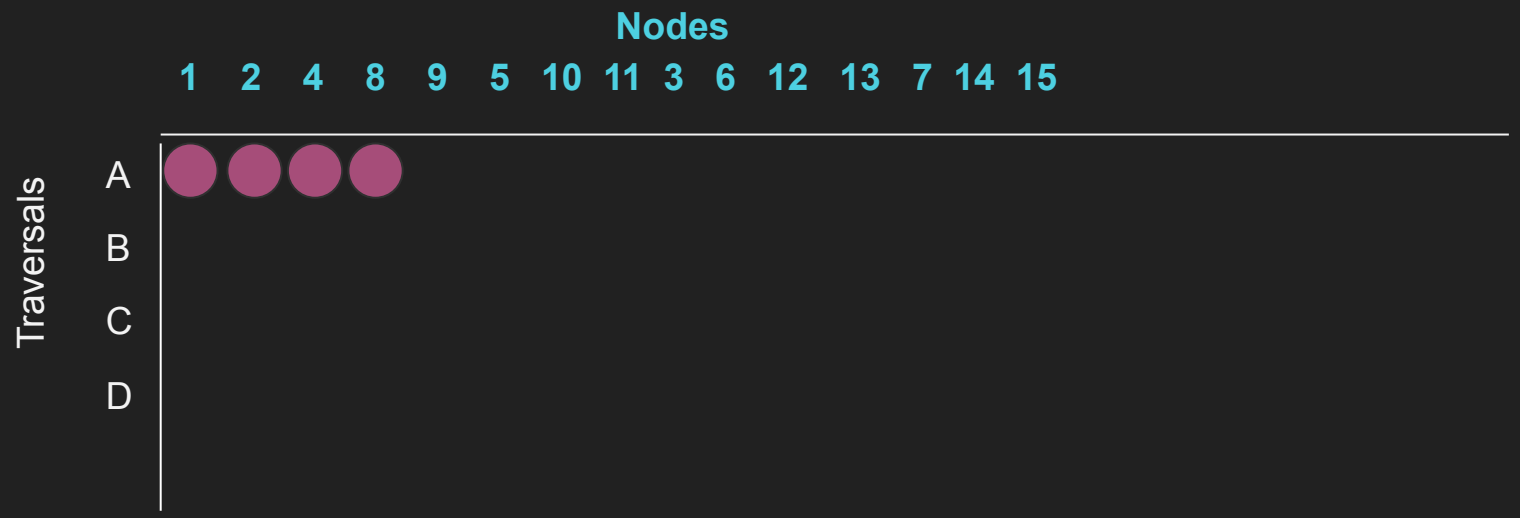
Tree


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

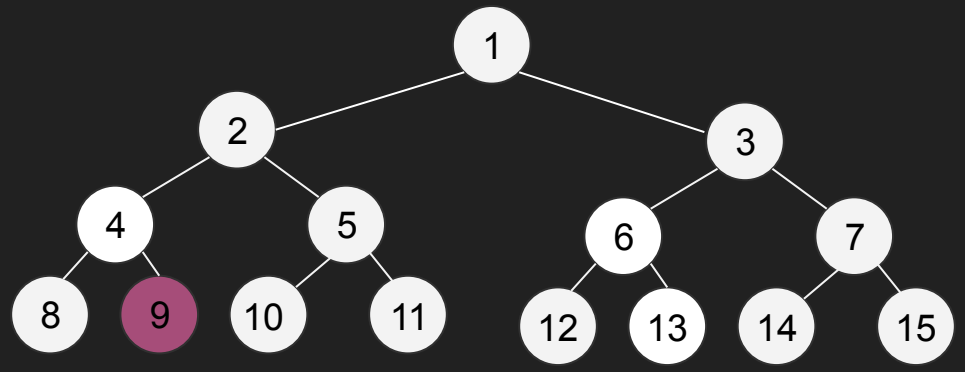

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

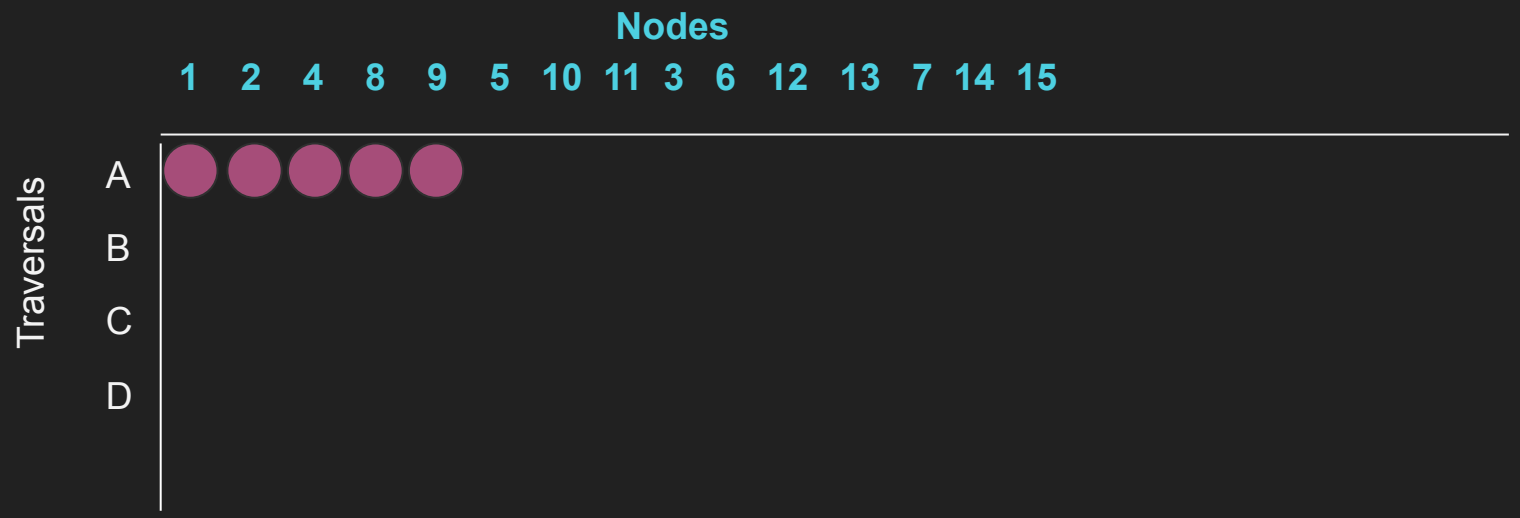
Tree


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

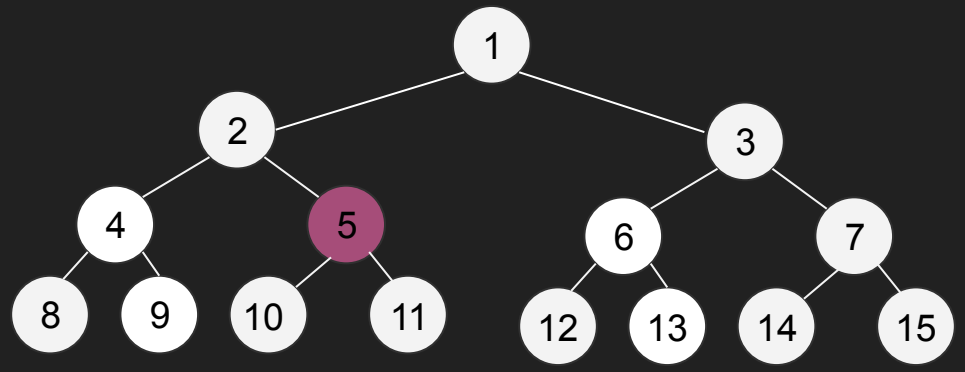

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

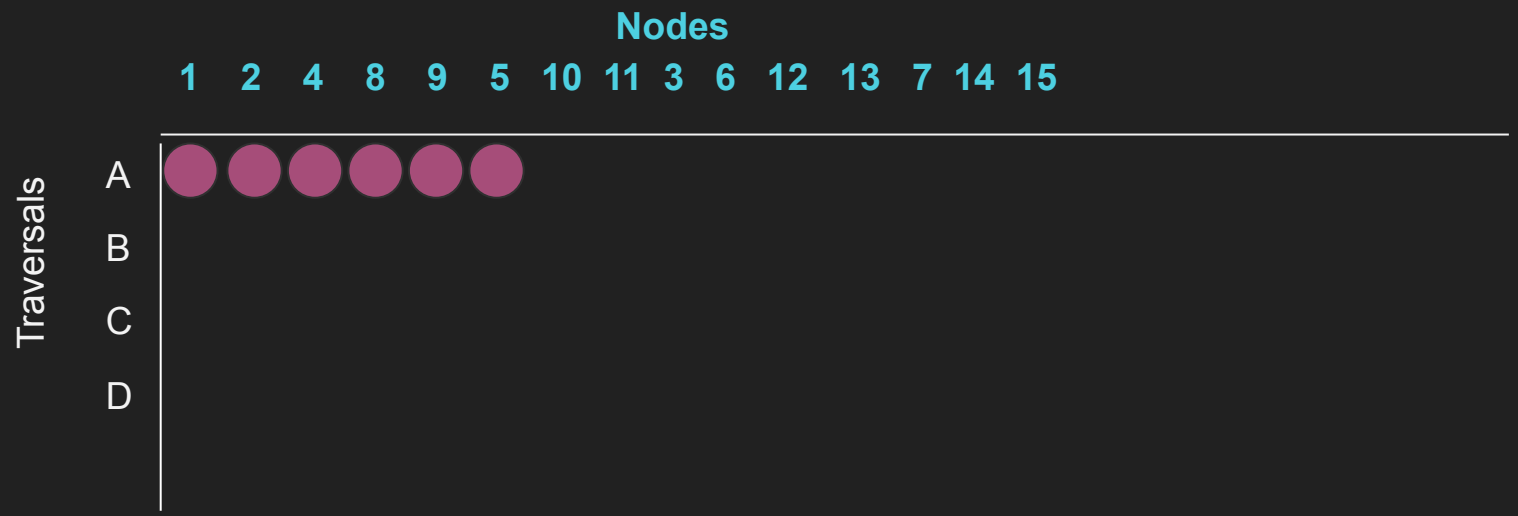
Tree


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

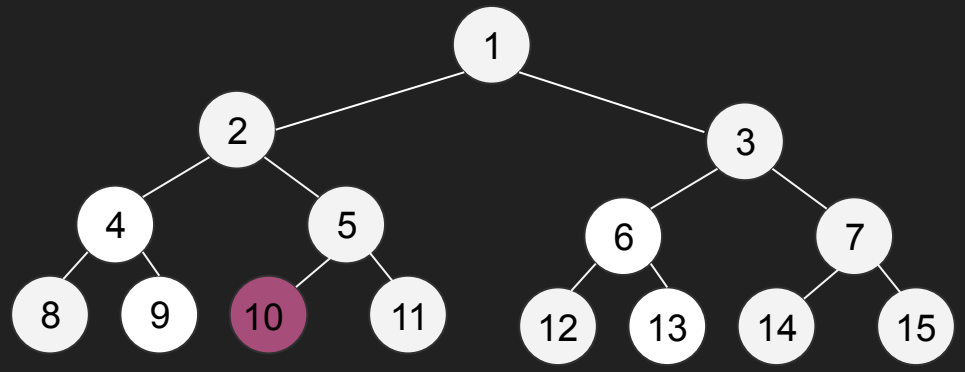

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

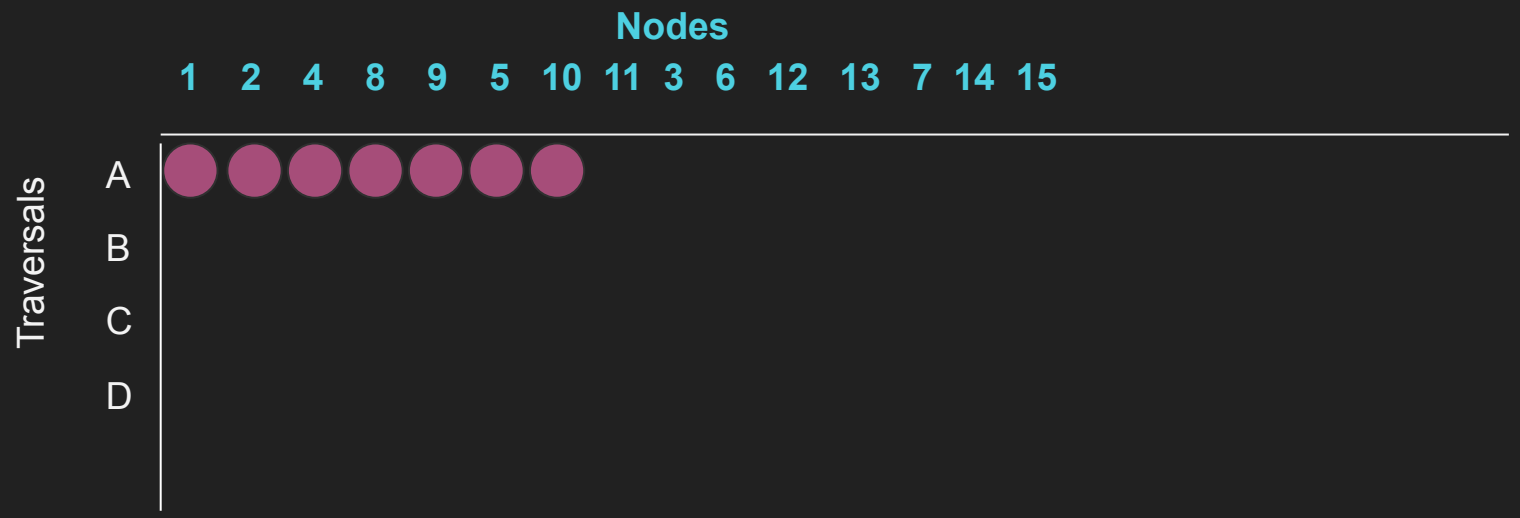
Tree


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

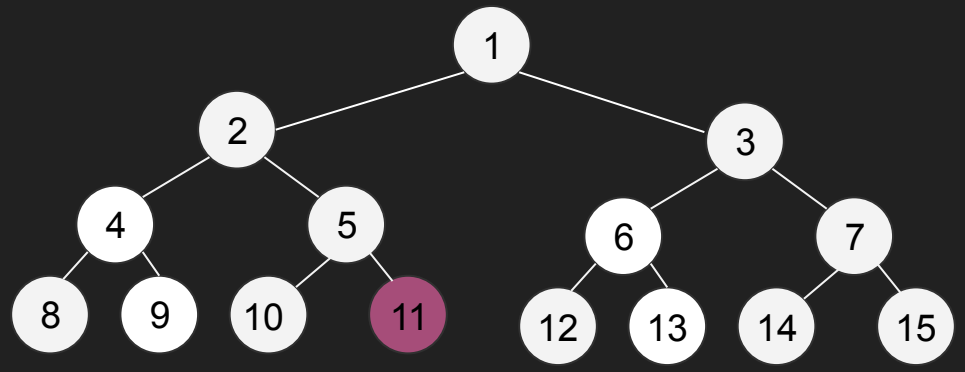

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

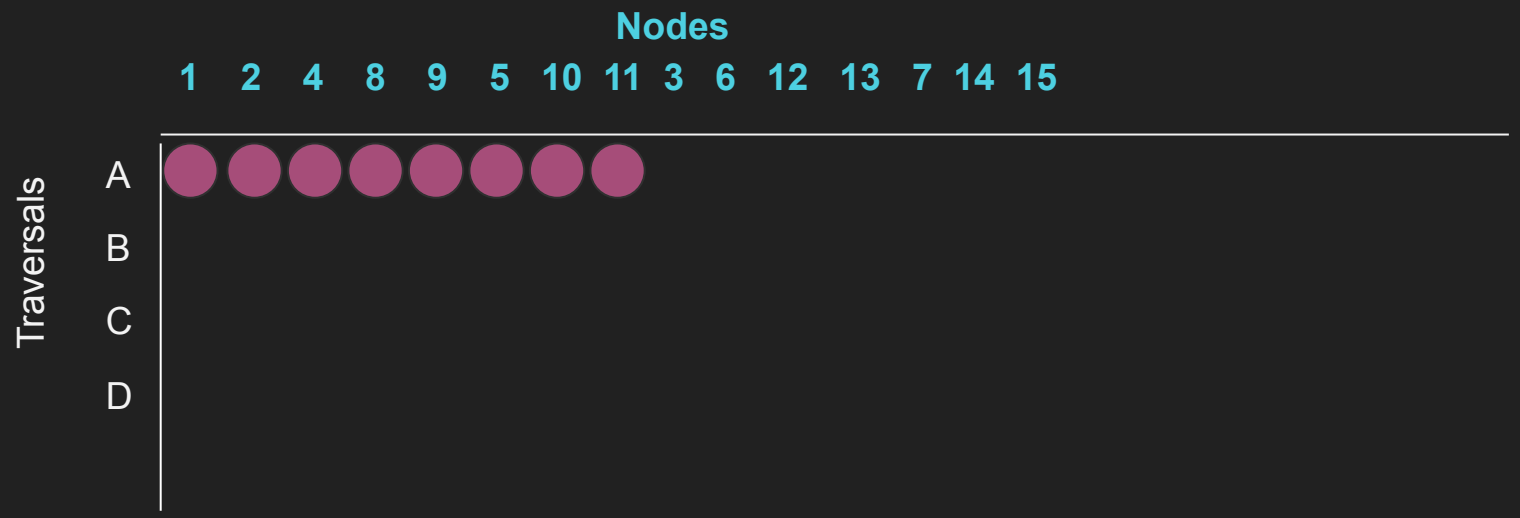
Tree


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

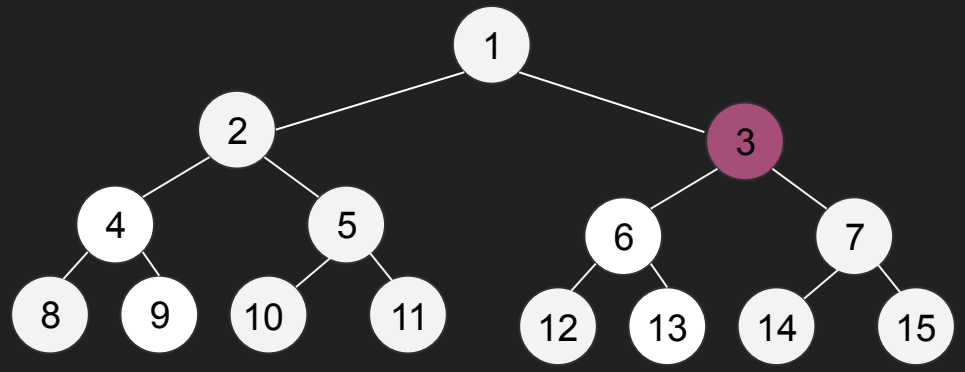

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

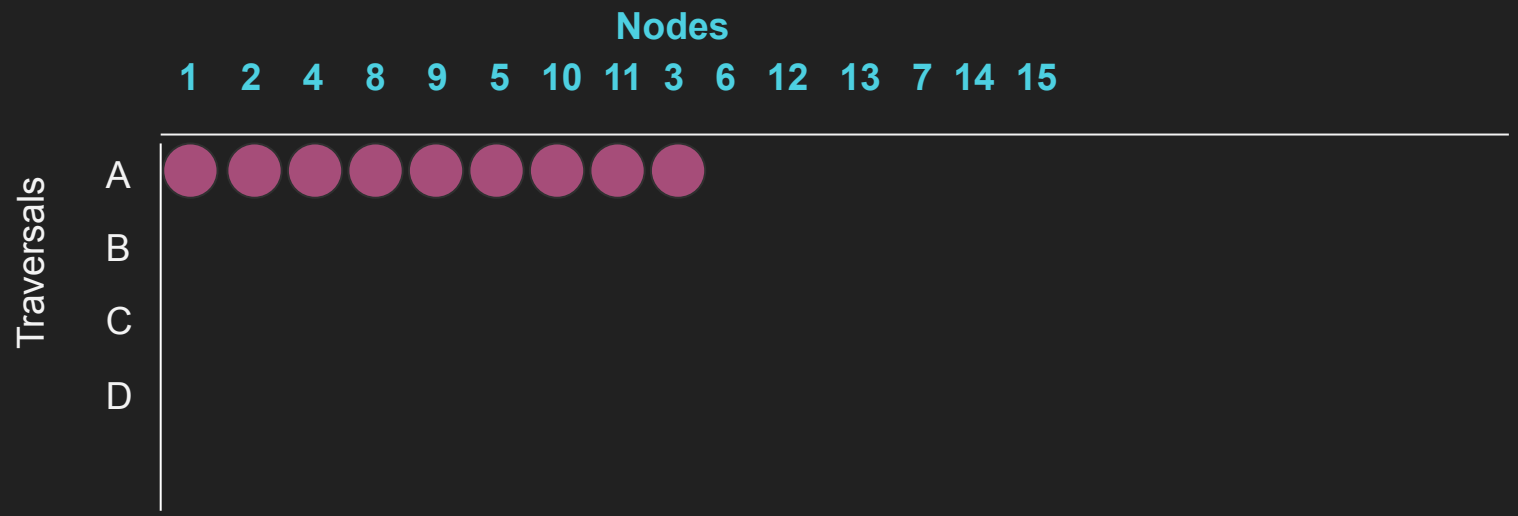
Tree


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

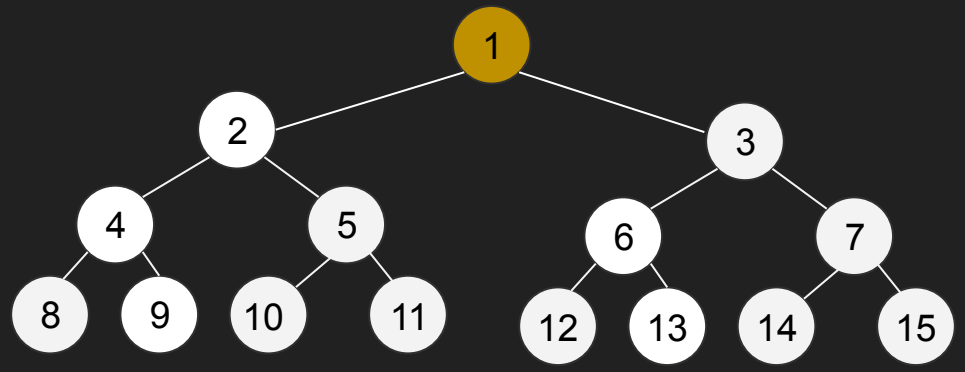

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

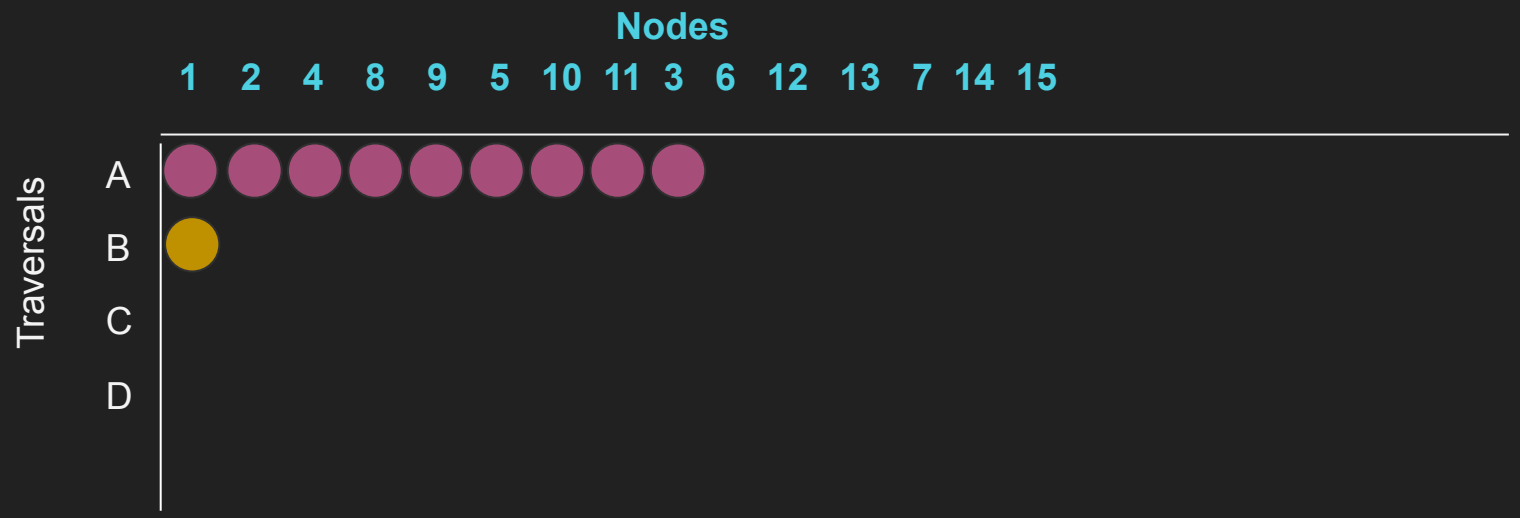
Tree


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

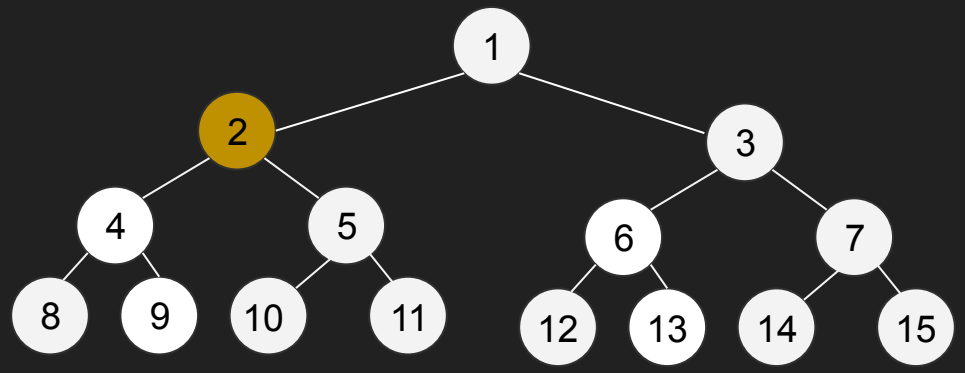

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

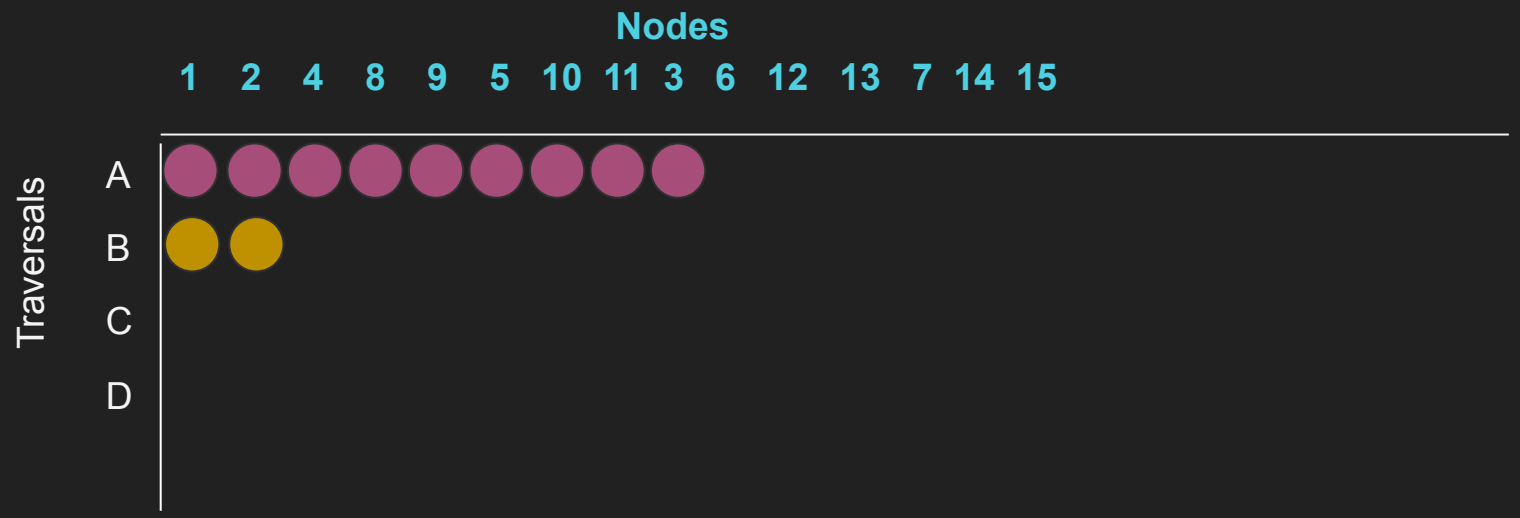
Tree


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

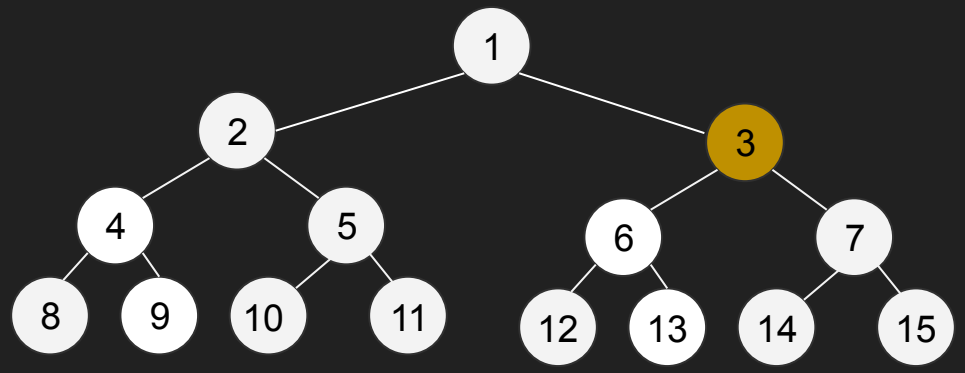

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

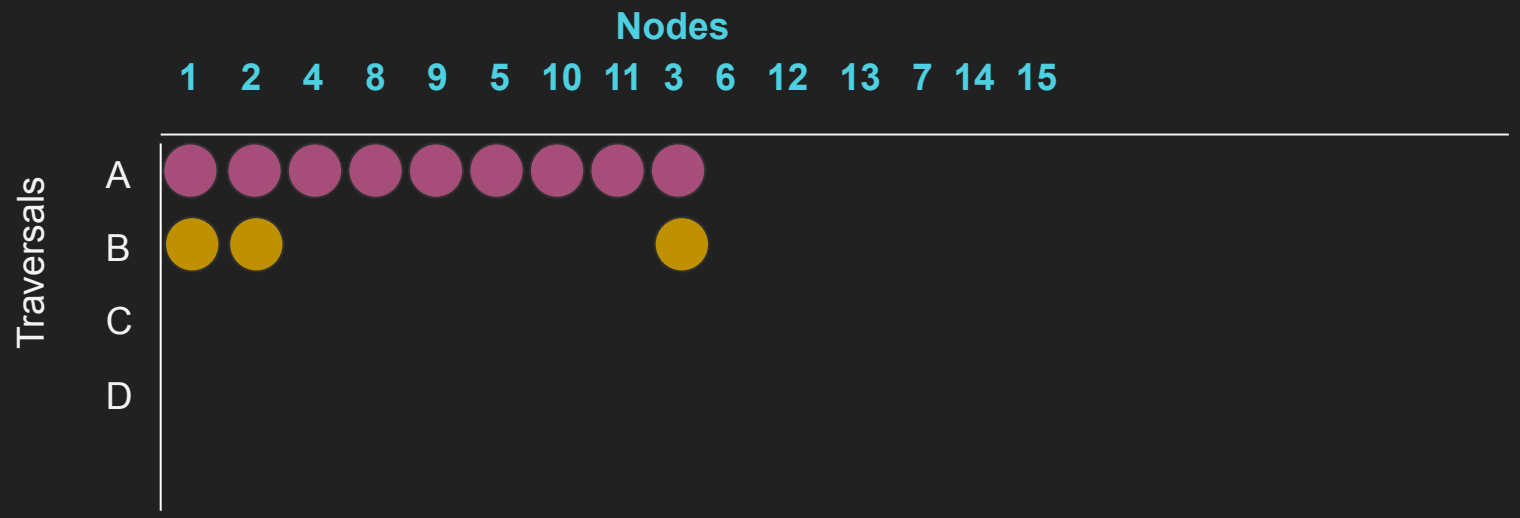
Tree


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

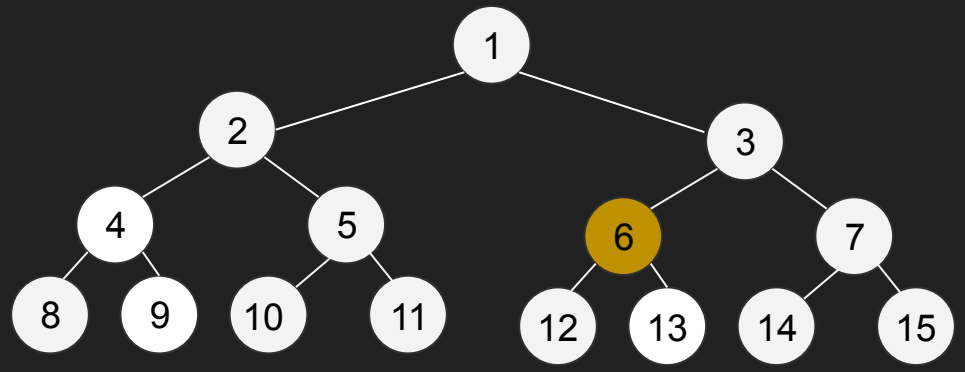

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

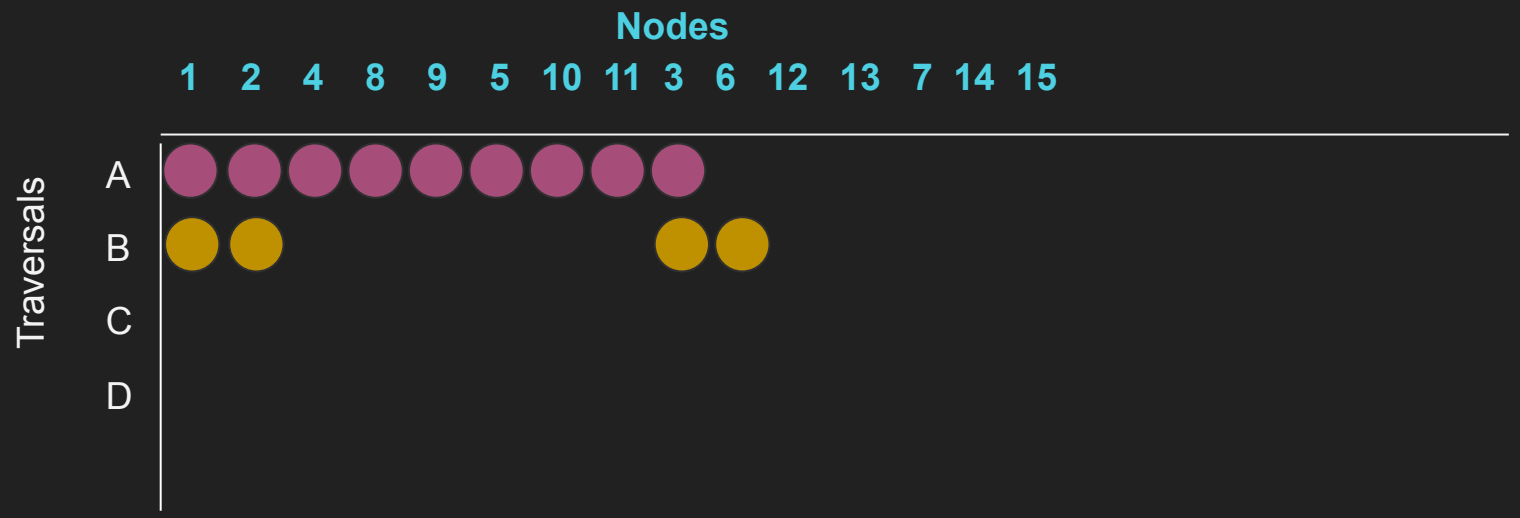
Tree


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

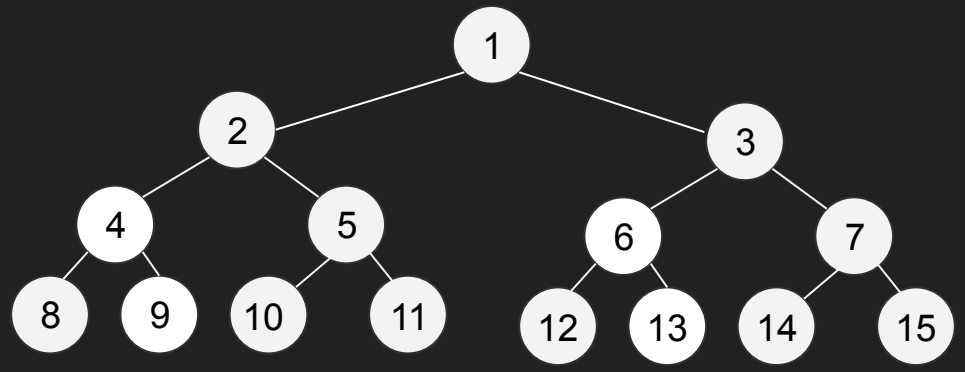

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

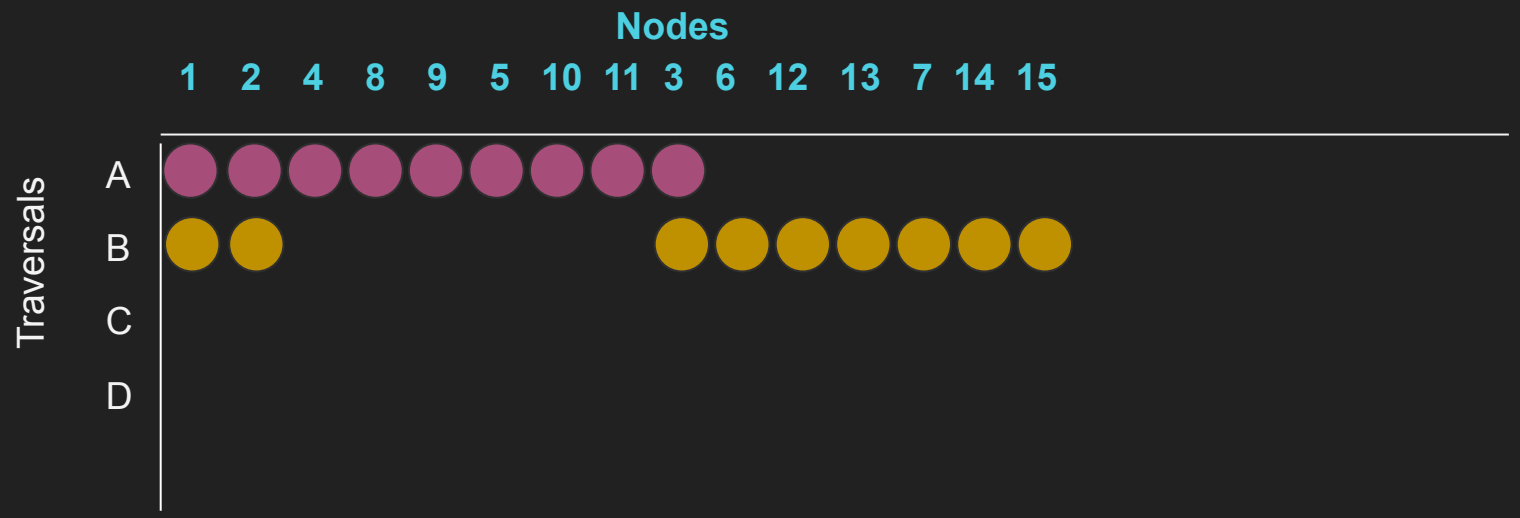
Tree


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

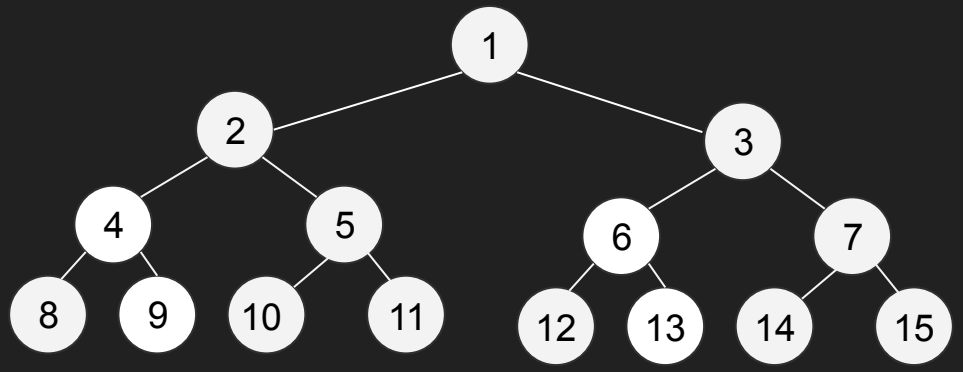

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

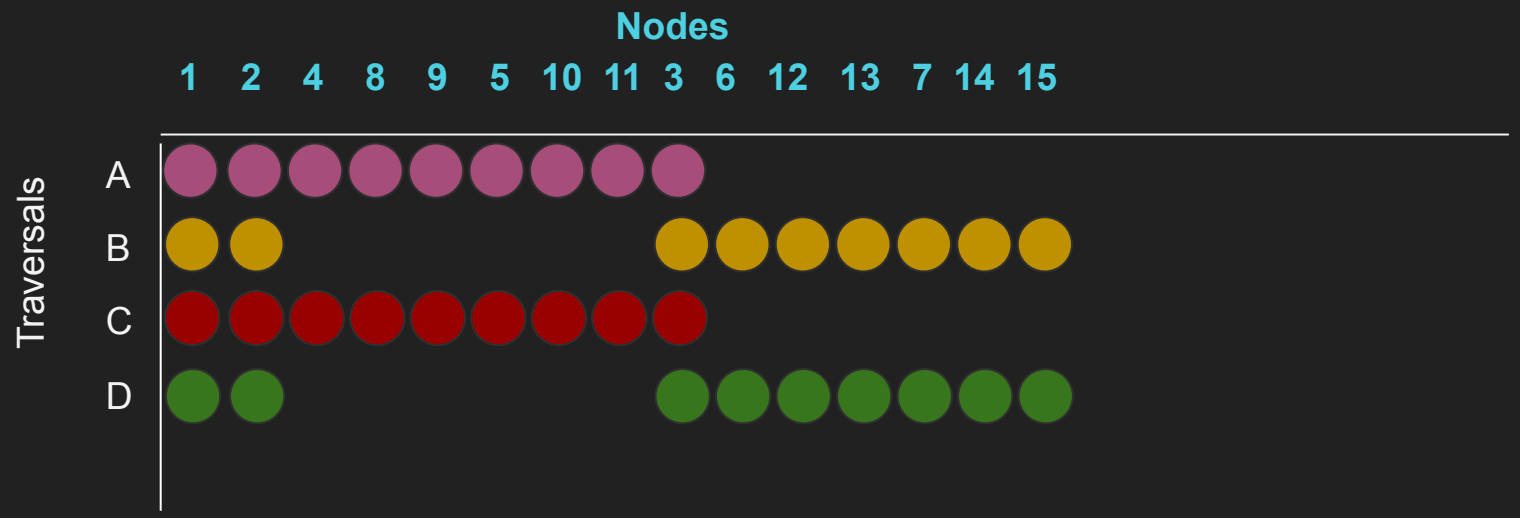
Tree


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

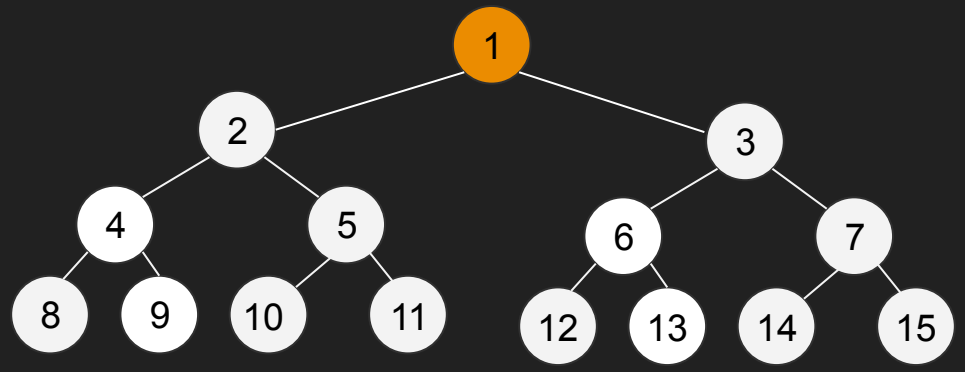

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

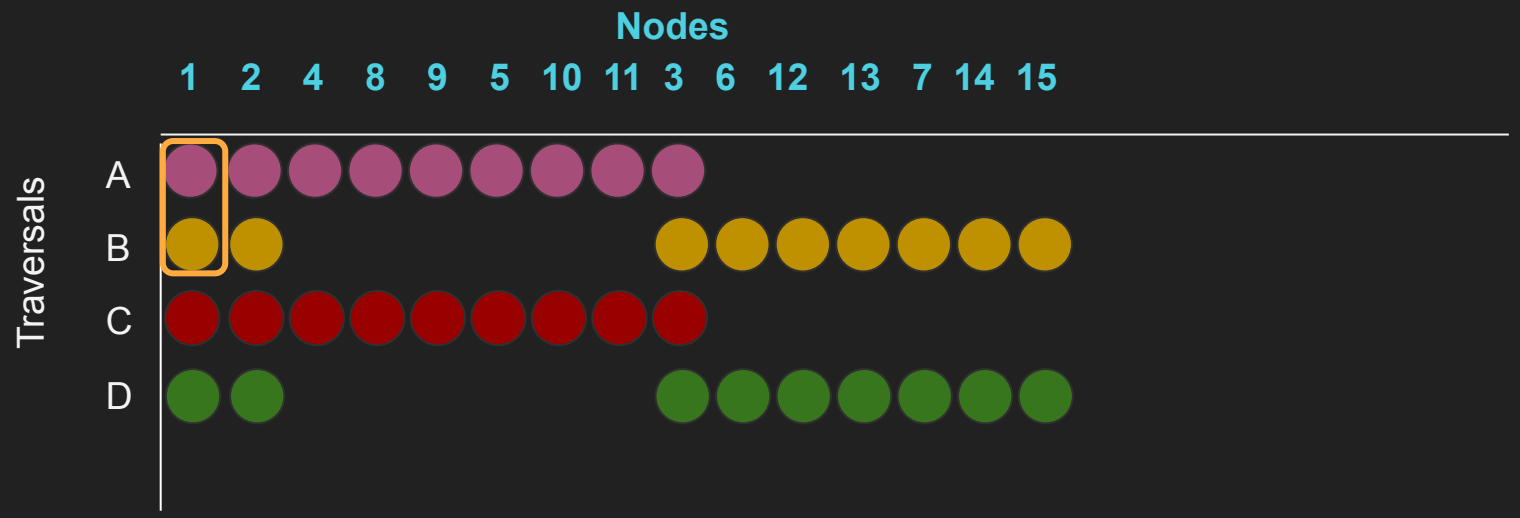
Tree


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

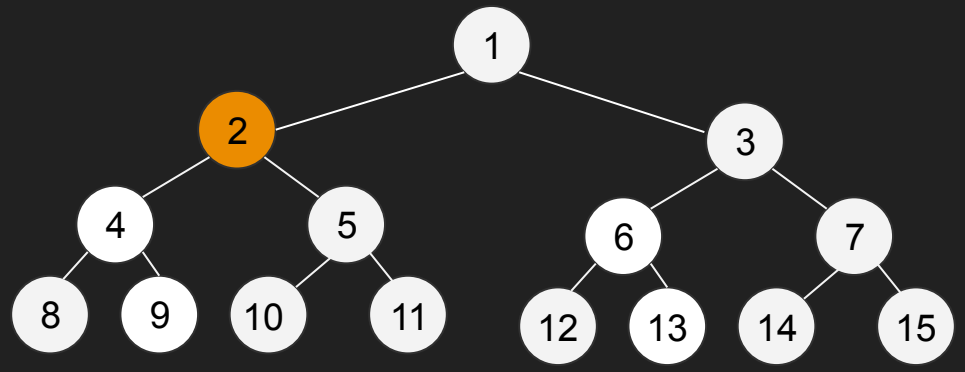

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

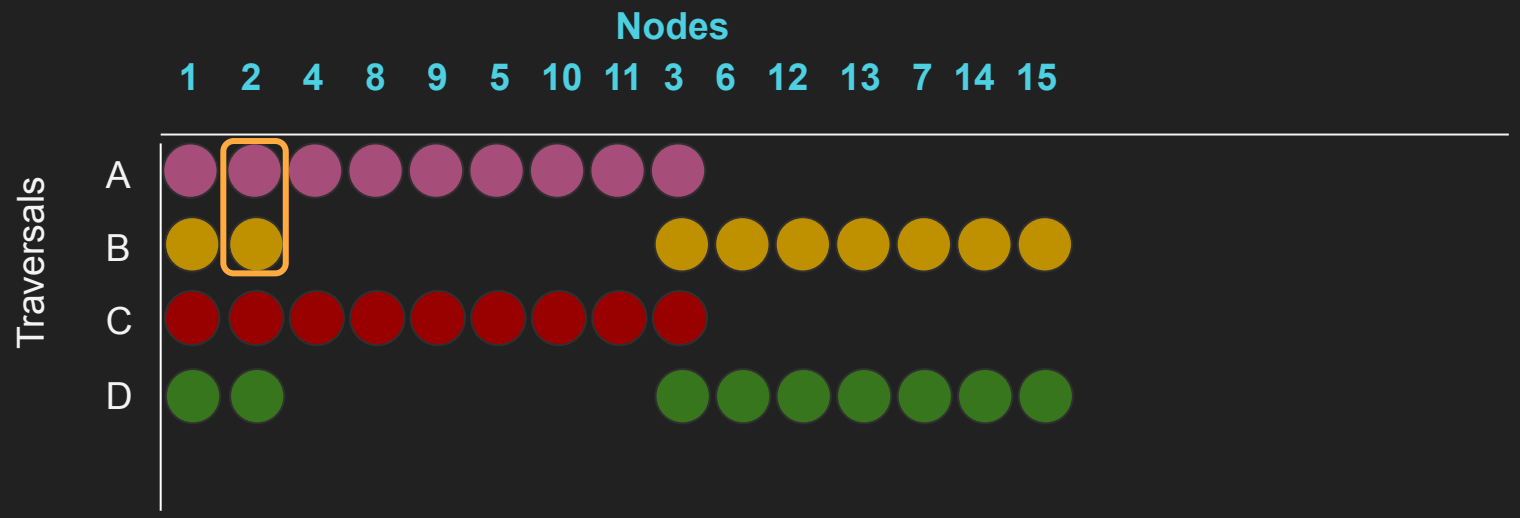
Tree


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

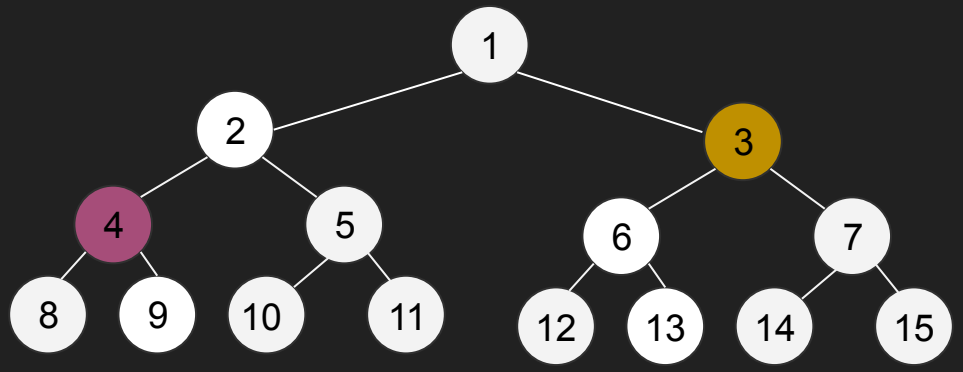

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

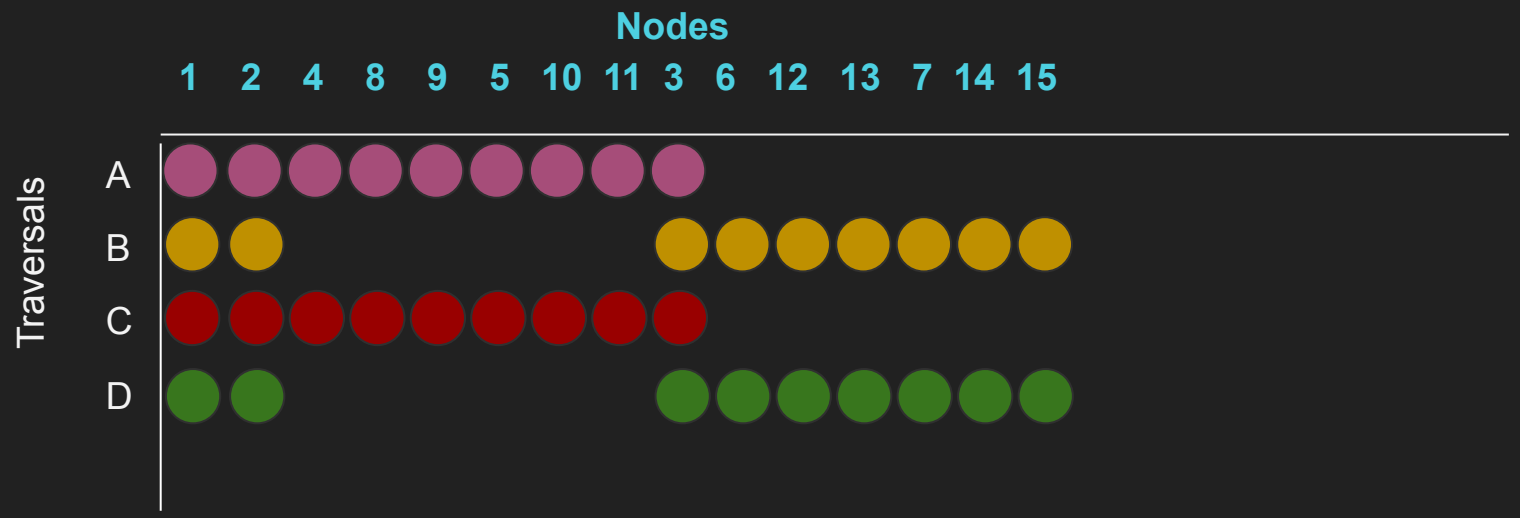
Tree


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

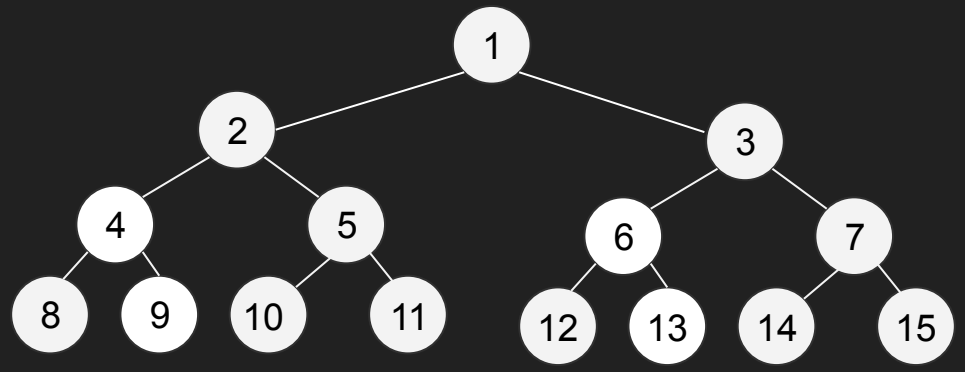

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

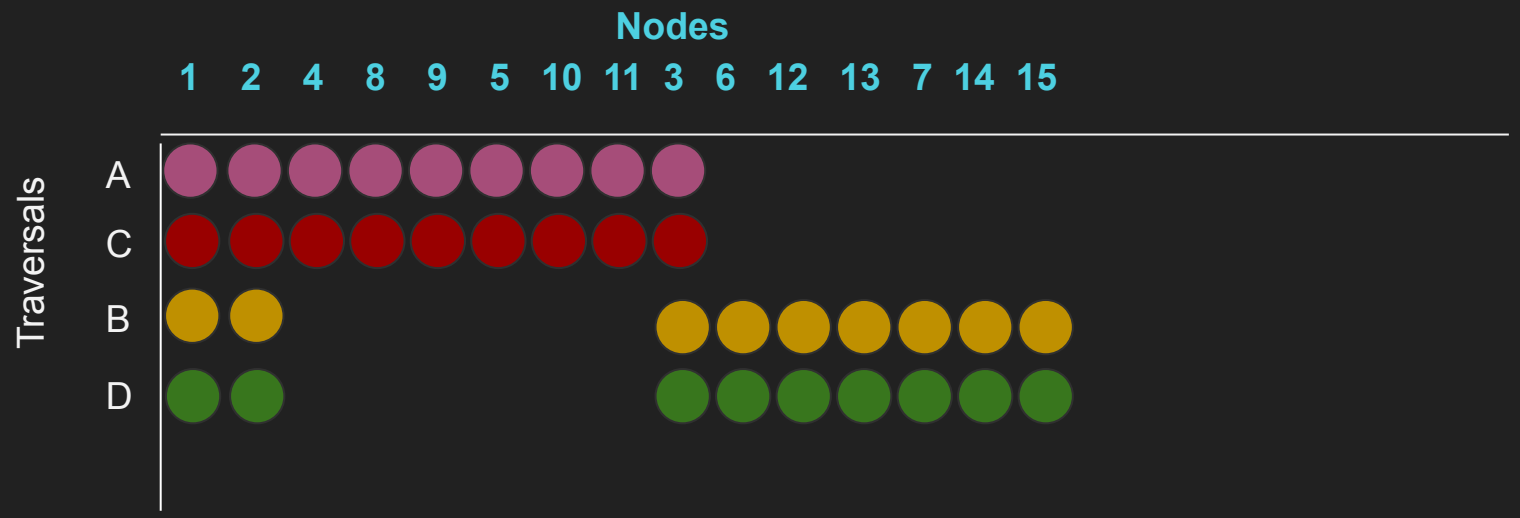
Tree


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

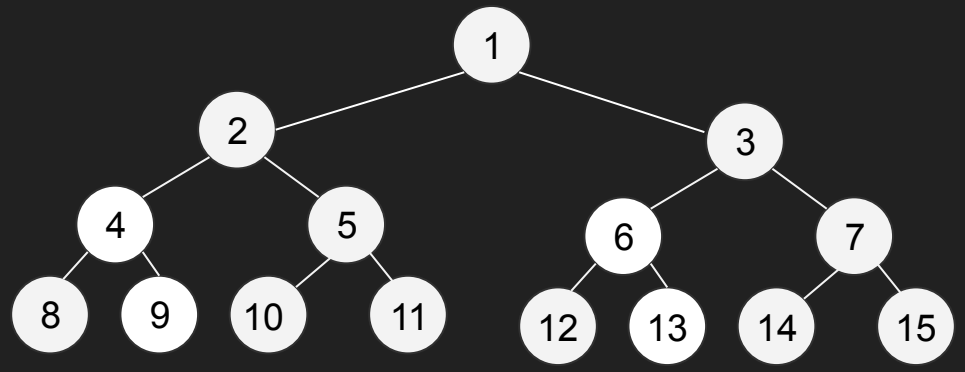

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
B
C
D

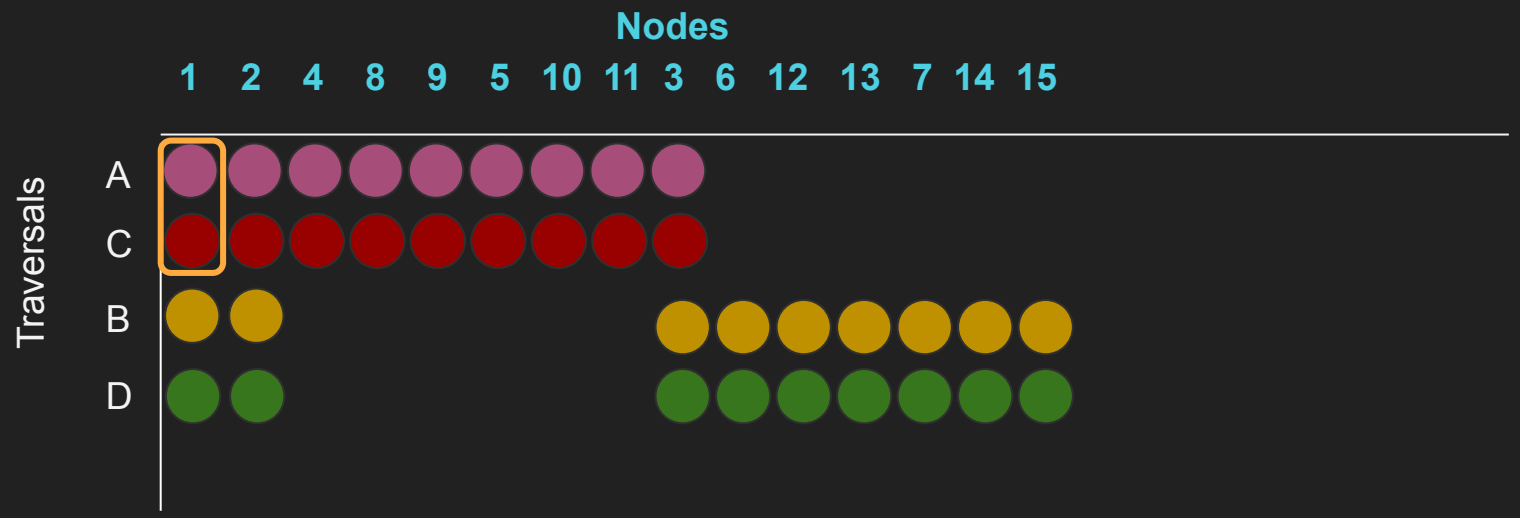
Tree


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

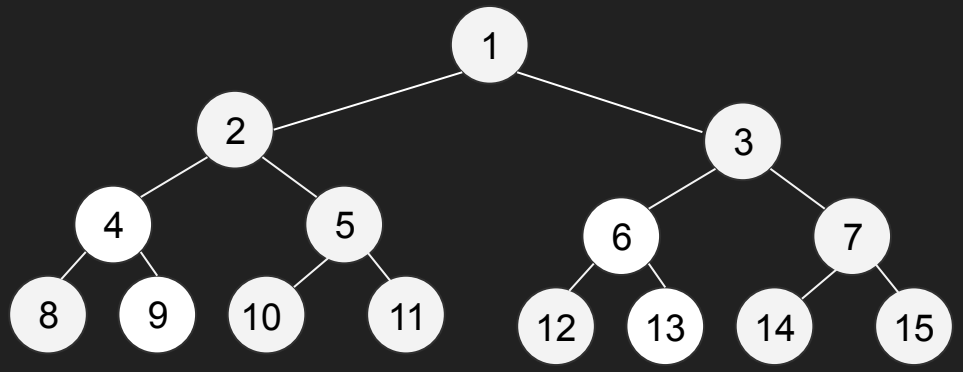

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

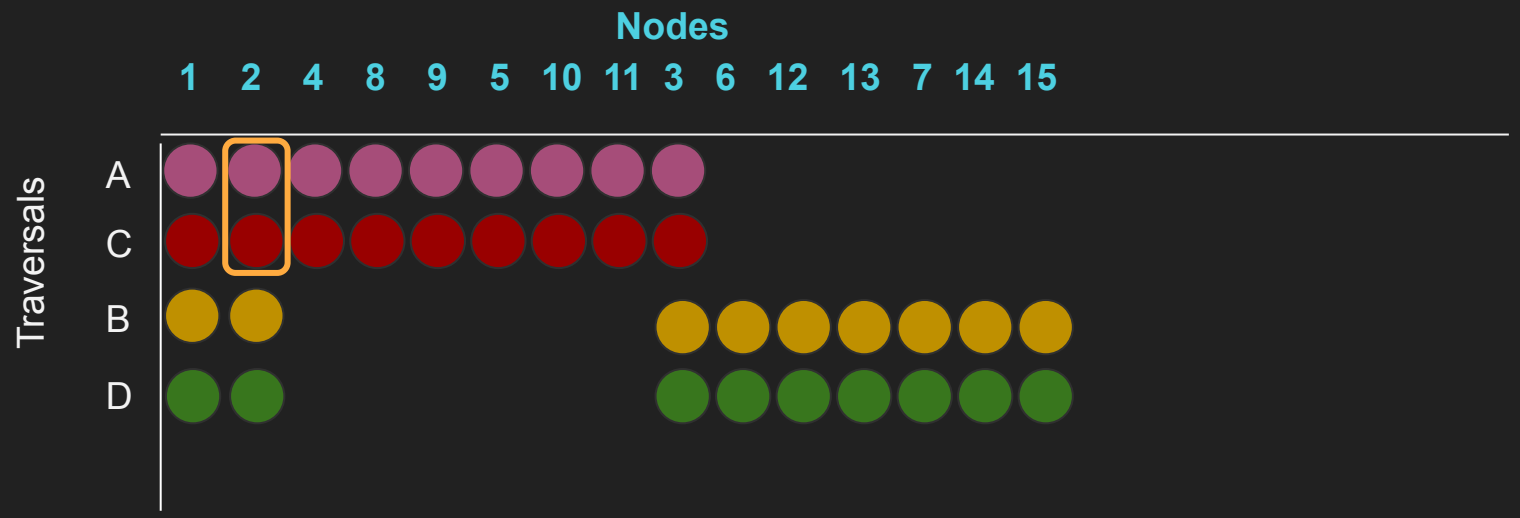
Tree


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

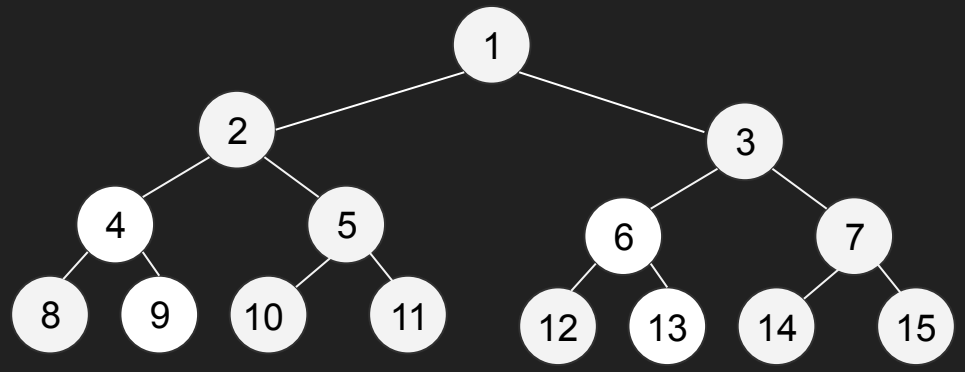

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

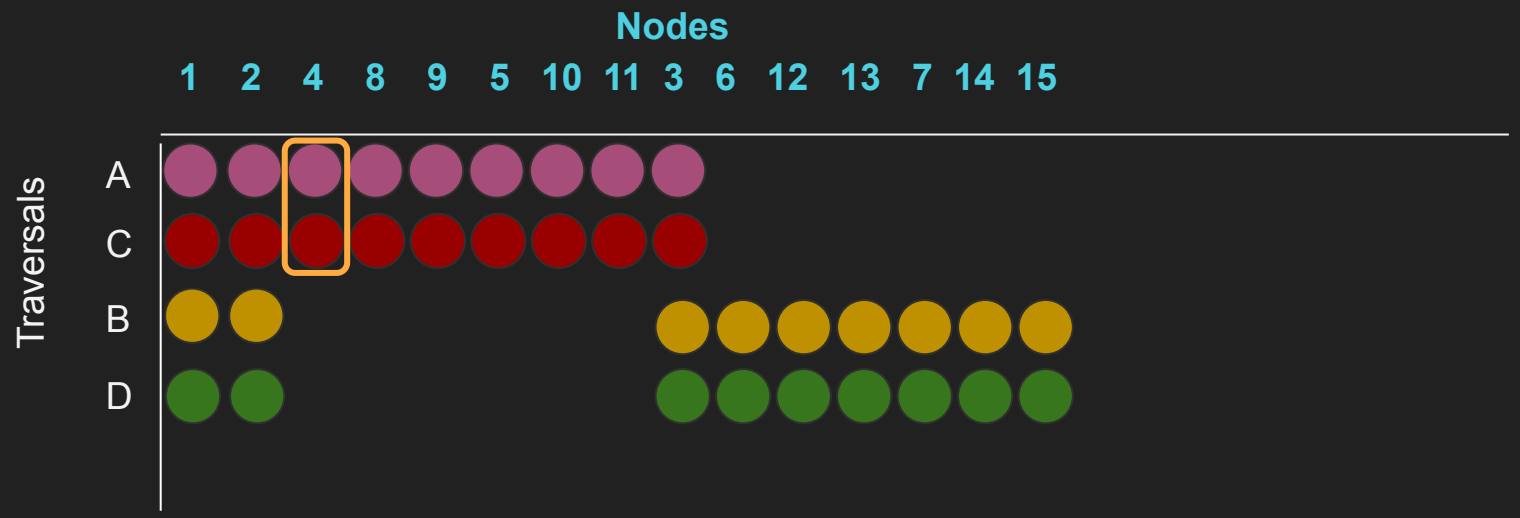
Tree


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

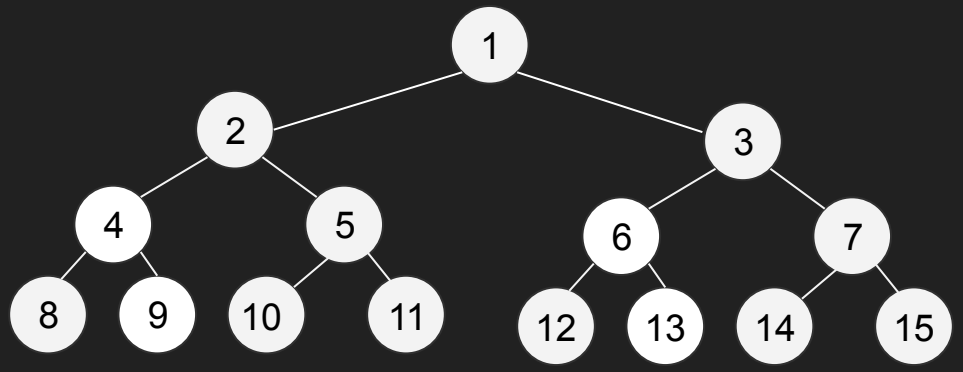

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

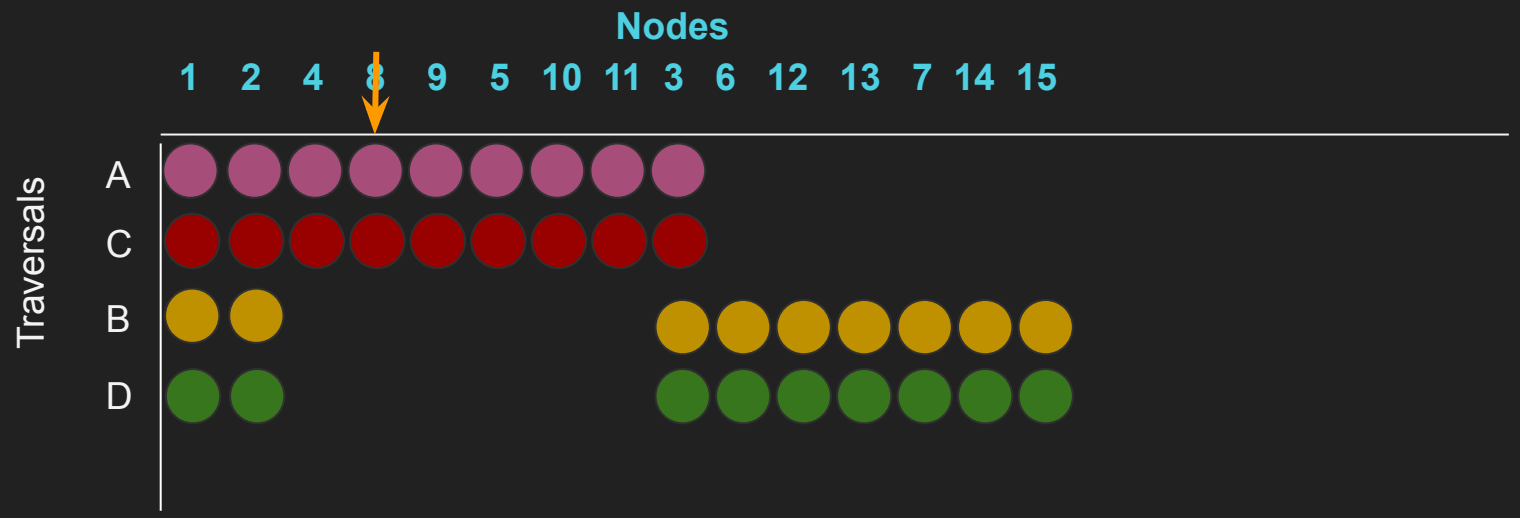
Tree


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

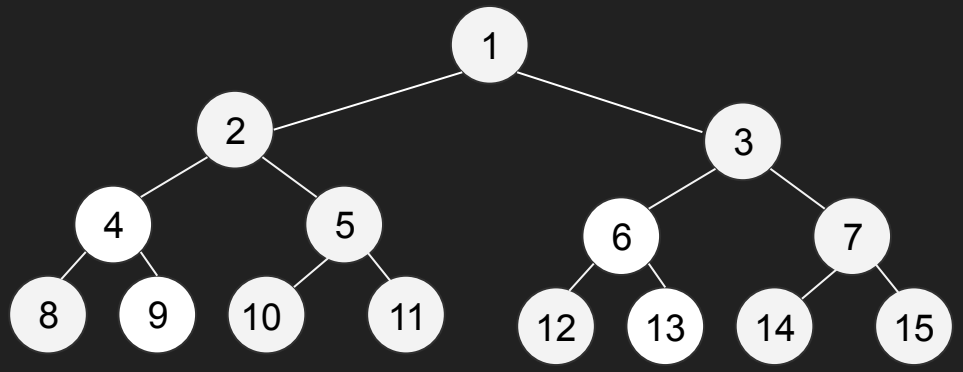

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

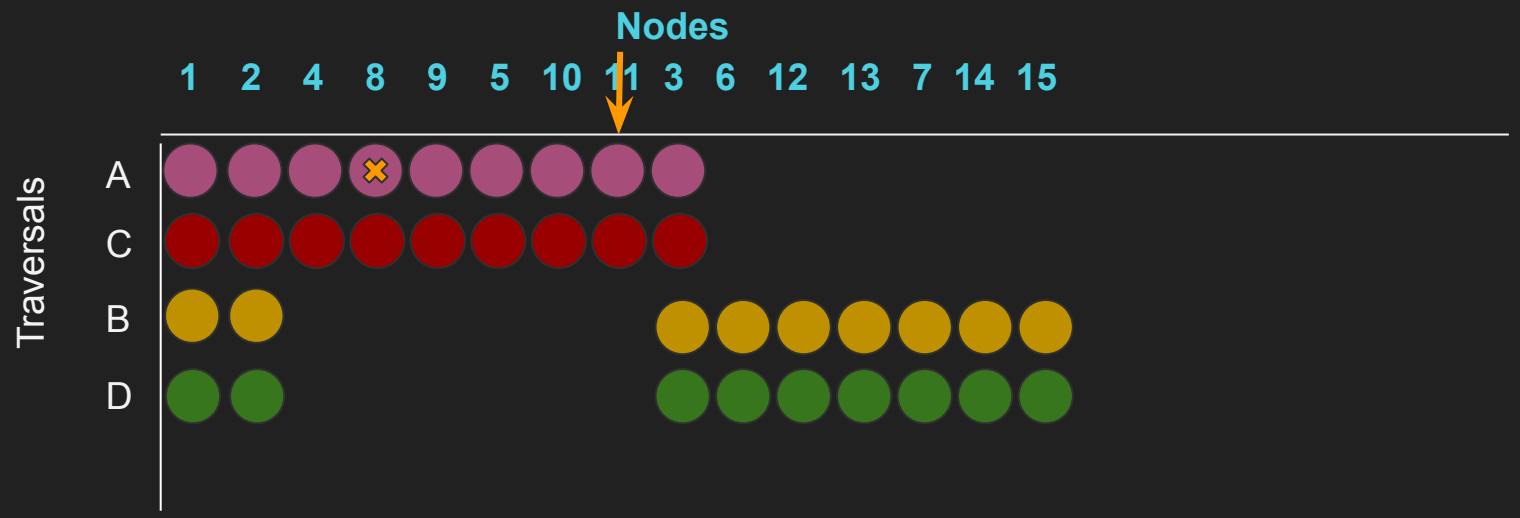
Tree


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

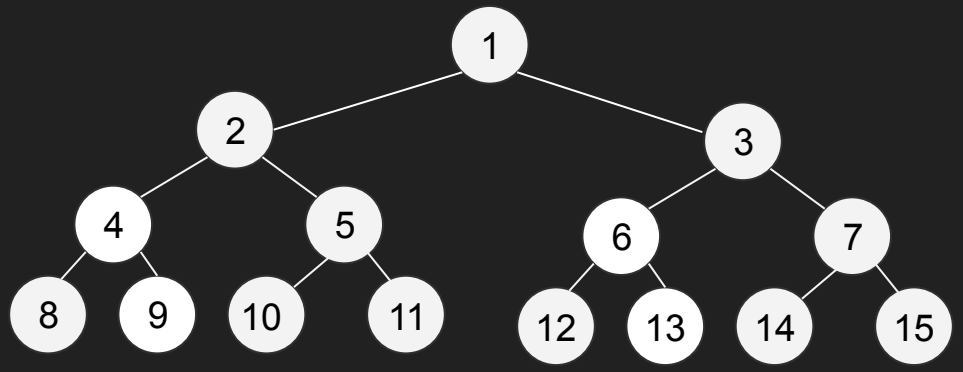

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

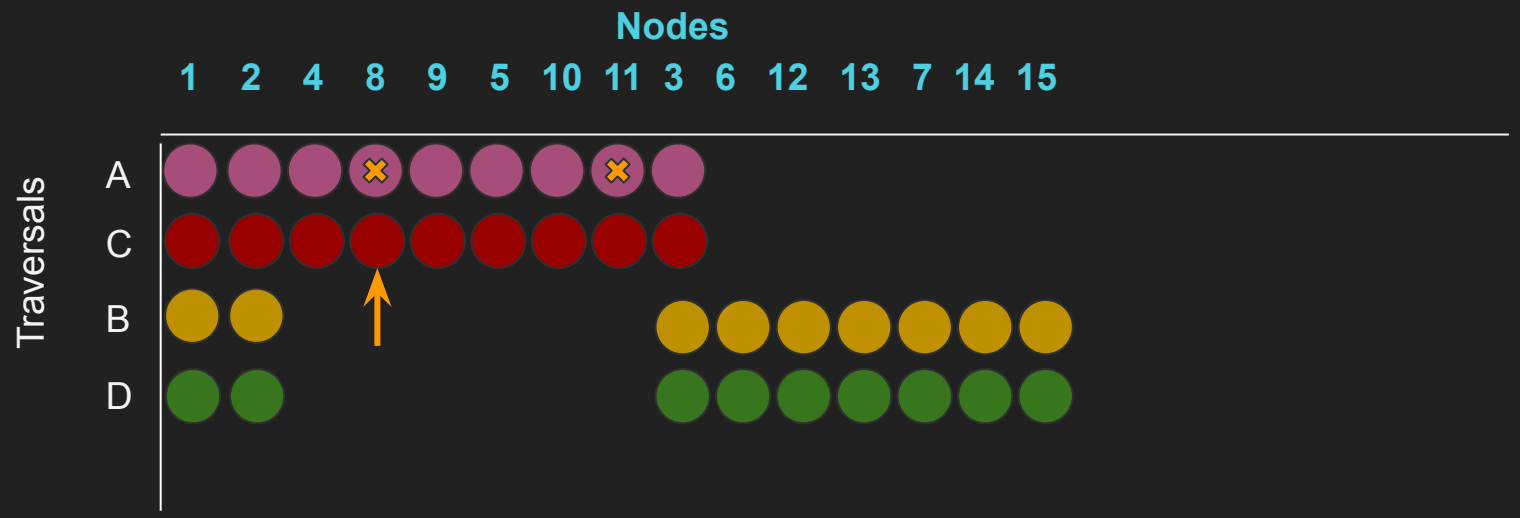
Tree


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

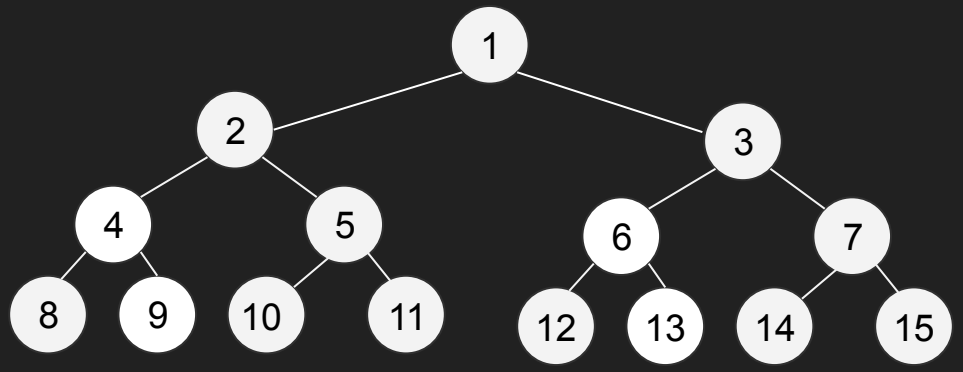

Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

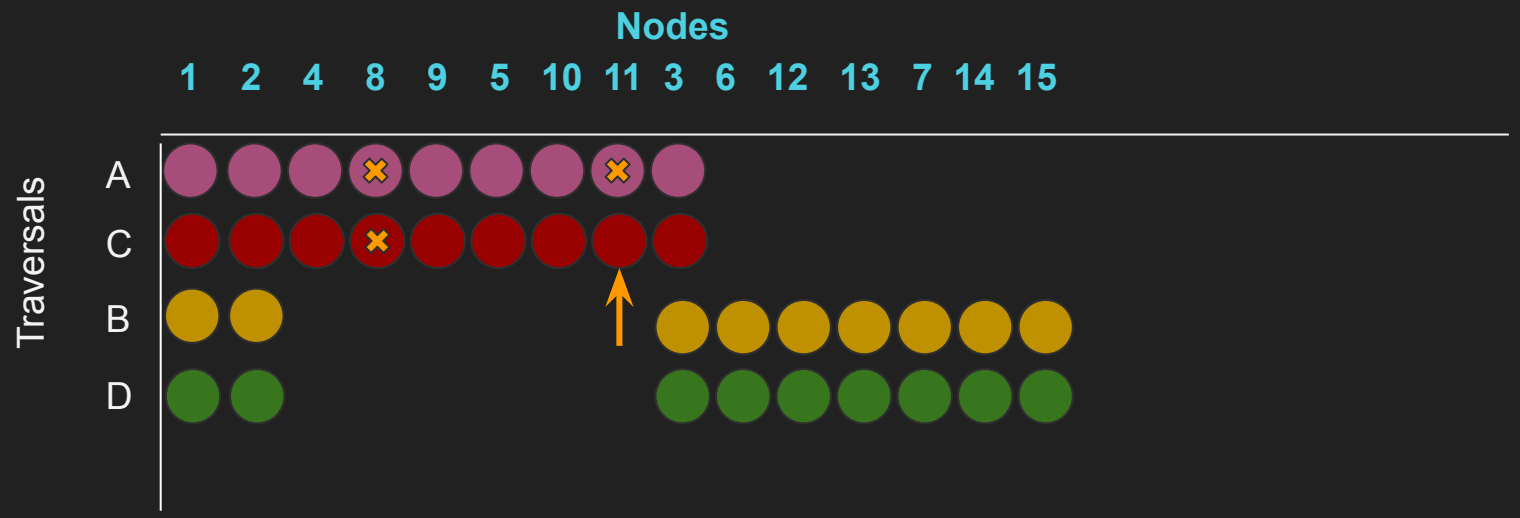
Tree


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


Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

Tree


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


Nodes
1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Traversals
A
C
B
D

Milind Kulkarni’s Group at Purdue

Dynamic Vectorization

Dynamic Factorization

OPEN PROBLEM

Serial Reduction

```
int reduce_serial(int x) {
  assert(x > 0 && !(x % 2));
  int finish_time = 0;
  while (true) {
    if (!x) {
      break;
    }
    x = x - 2;
    ++finish_time;
  }
  return finish_time;
}
```

Clobber to Disable the Optimizer Locally

```
int reduce_serial(int x) {
  assert(x > 0 && !(x % 2));
  int finish_time = 0;
  while (true) {
    clobber();
    if (!x) {
      break;
    }
    x = x - 2;
    ++finish_time;
  }
  return finish_time;
}
```

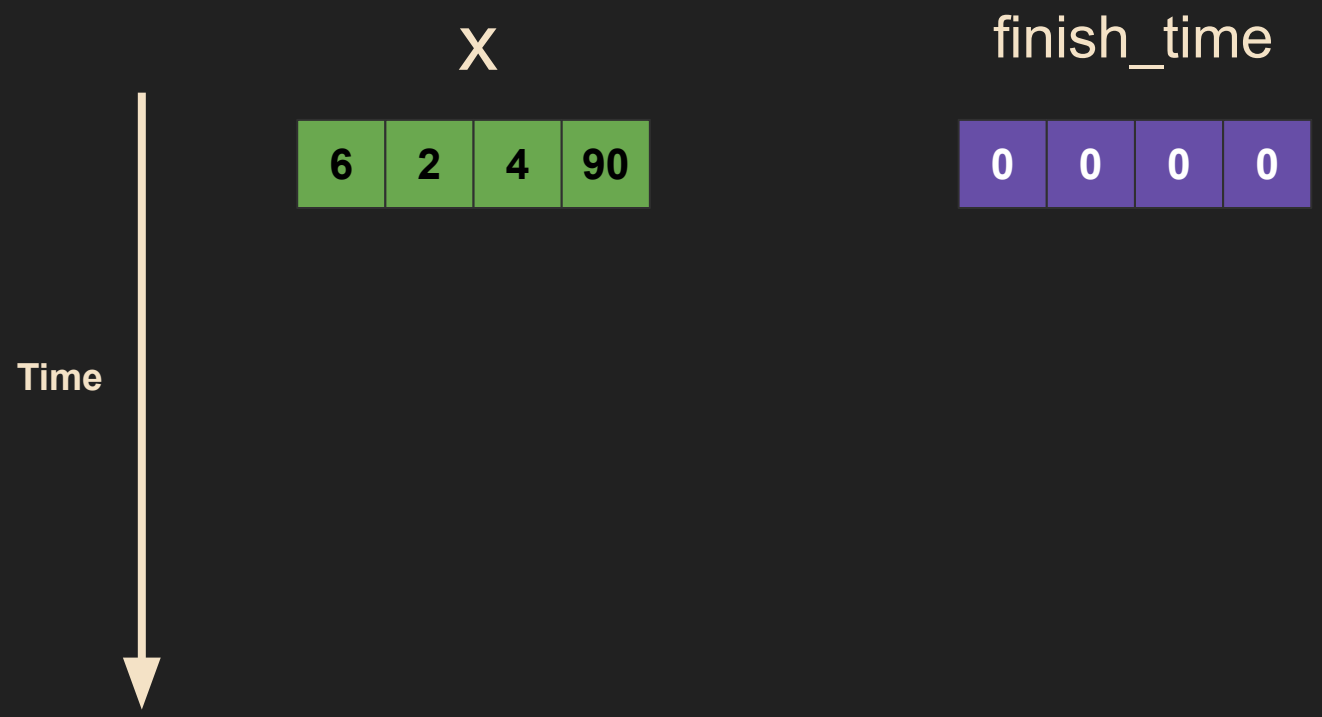
Clobber

```
void clobber() {
  asm volatile(“” : : : “memory”);
}
```

Serial Reduction

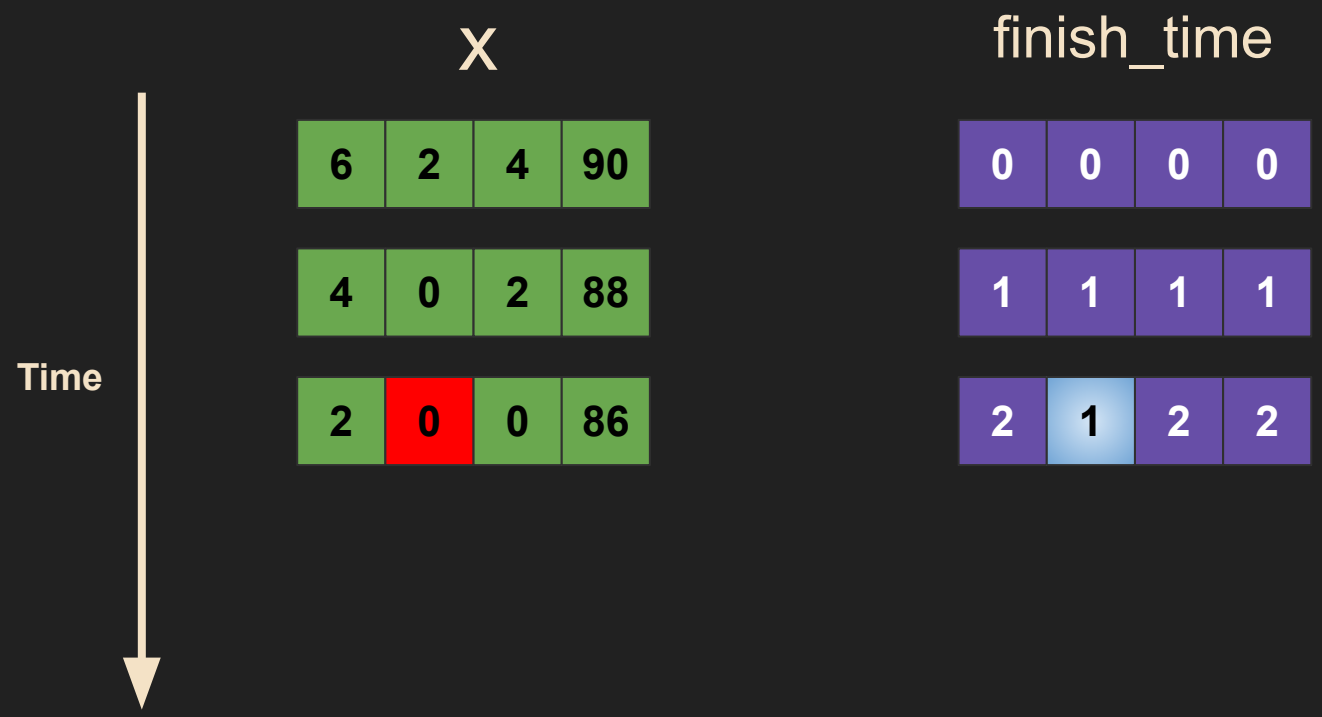
```
int run_serial(int *in, int *out, int len) {
  for (int i = 0; i < len; ++i) {
    out[i] = reduce_serial(in[i]);
  }
}
```

Statically Vectorized Reduction

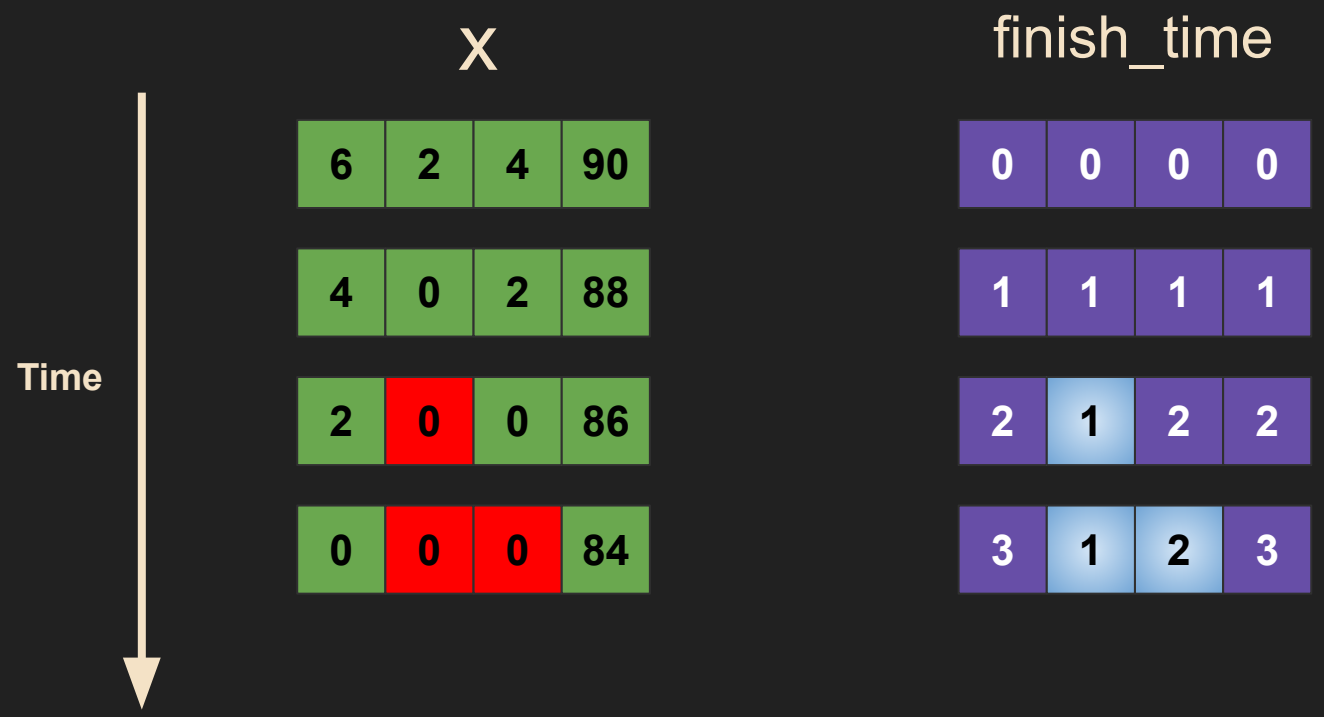


Statically Vectorized Reduction

Statically Vectorized Reduction



Statically Vectorized Reduction



Statically Vectorized Reduction

Statically Vectorized Reduction

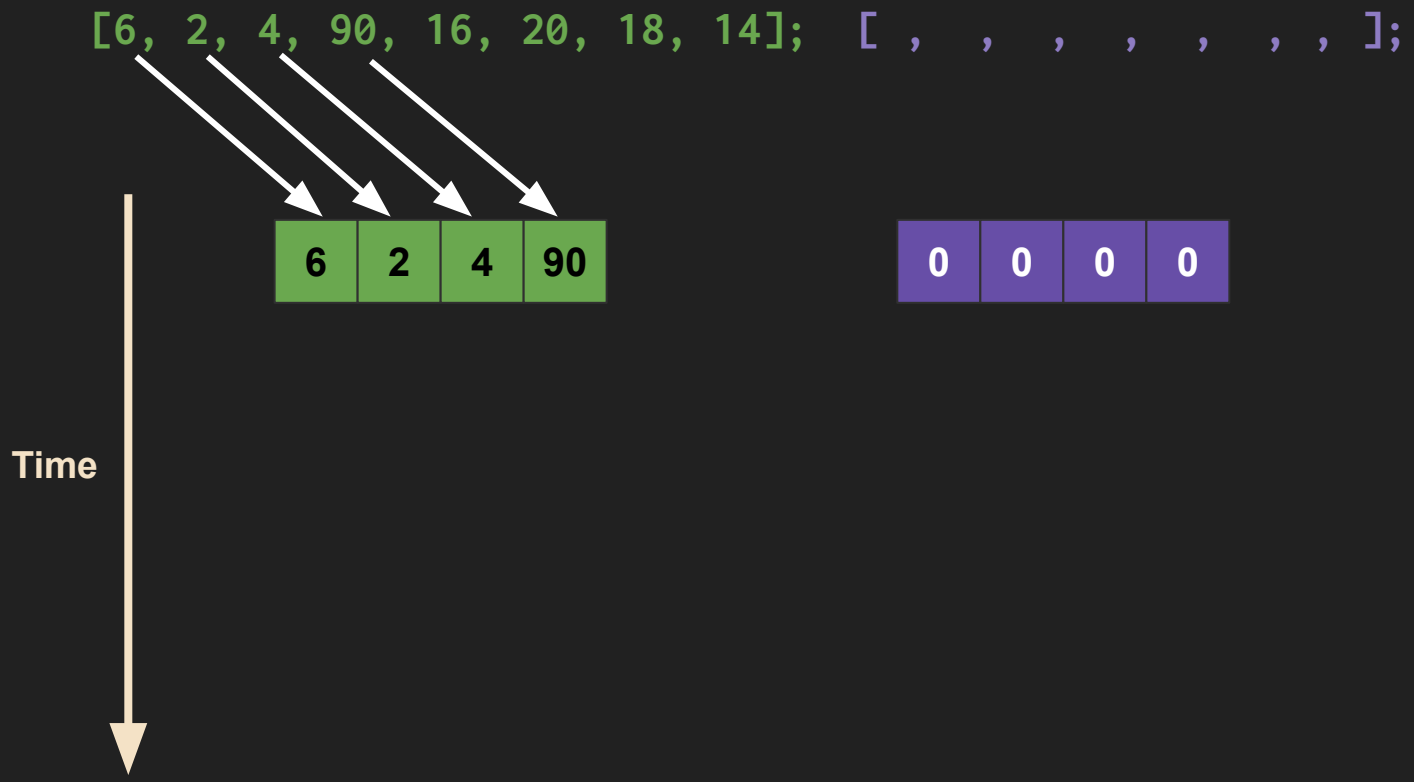
```
int4 reduce_vec(int4 x) {
  int4 finish_time(0);
  while (true) {
    int4 mask = x > int4(0);
    if (mask.are_all_zeroes())
      break;
    x = x - 2;
    finish_time = finish_time - mask;
  }
  return finish_time;
}
```

Statically Vectorized Reduction

Dynamically Vectorized Reduction

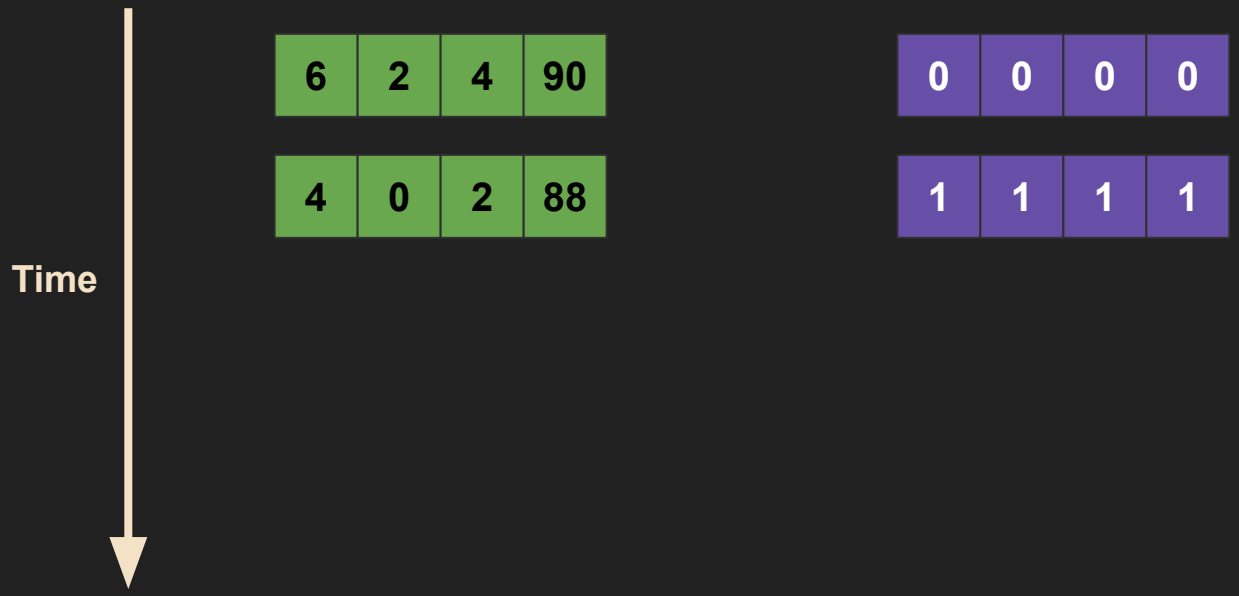
```
[6, 2, 4, 90, 16, 20, 18, 14];  [ ,  ,  ,  ,  ,  , , ];
```

Dynamically Vectorized Reduction



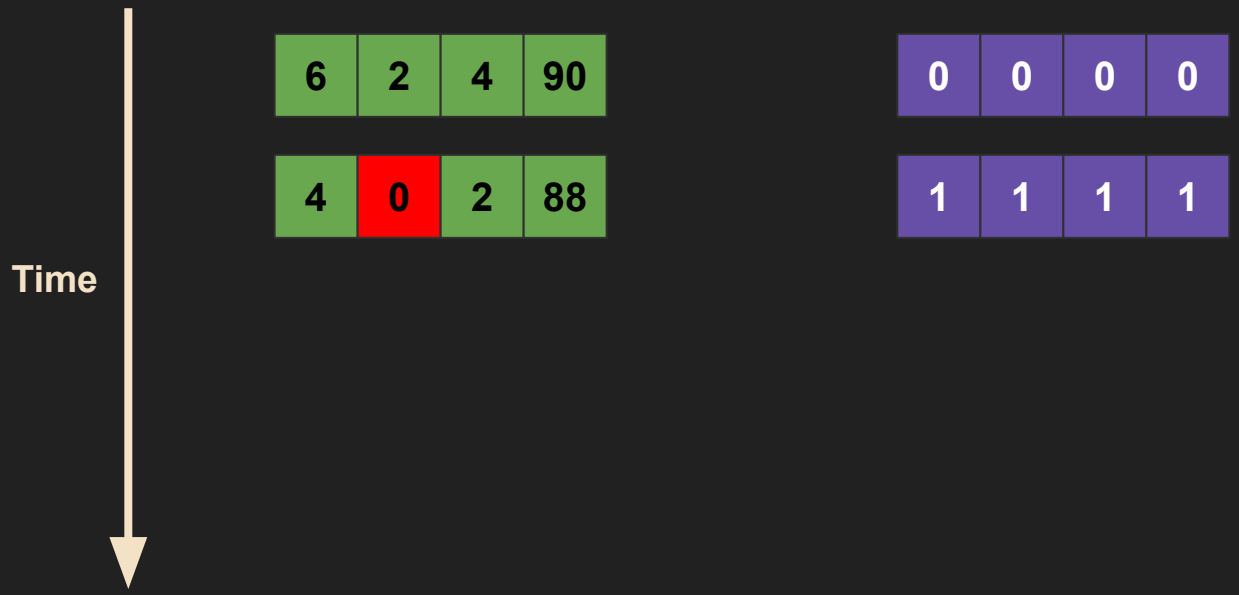
Dynamically Vectorized Reduction

```
[6, 2, 4, 90, 16, 20, 18, 14];  [ ,  ,  ,  ,  ,  , , ];
```



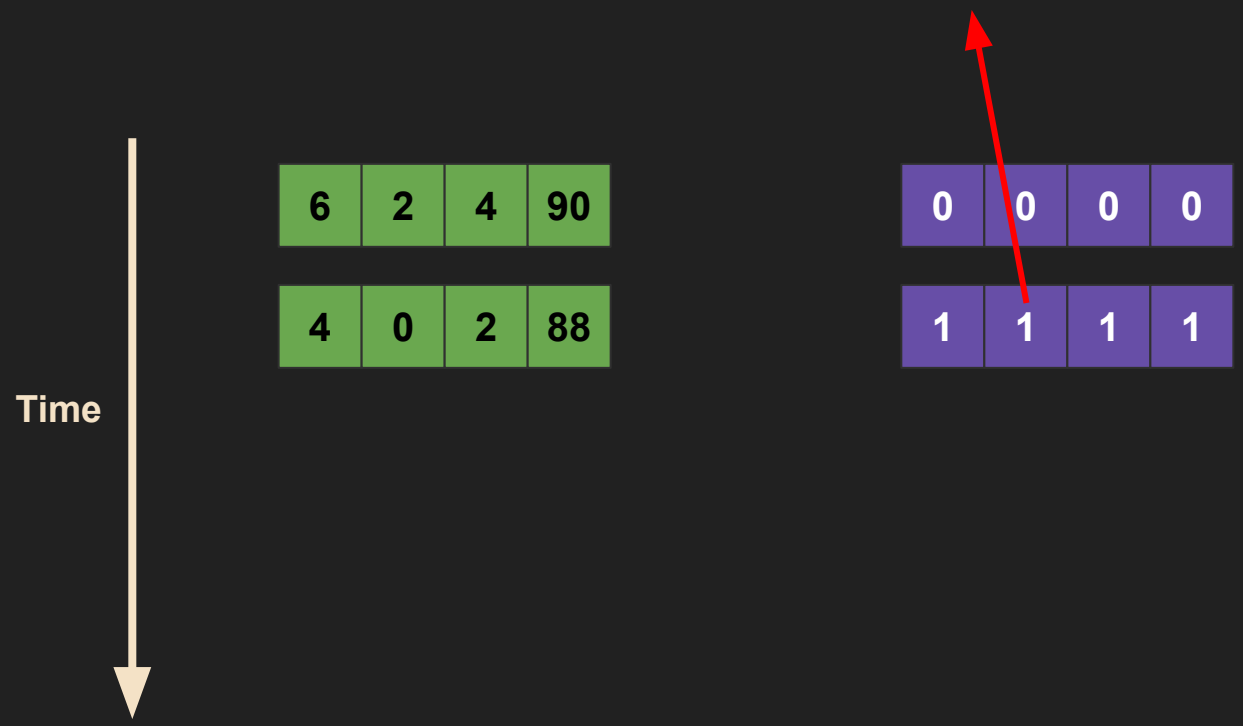
Dynamically Vectorized Reduction

```
[6, 2, 4, 90, 16, 20, 18, 14];  [ ,  ,  ,  ,  ,  , , ];
```

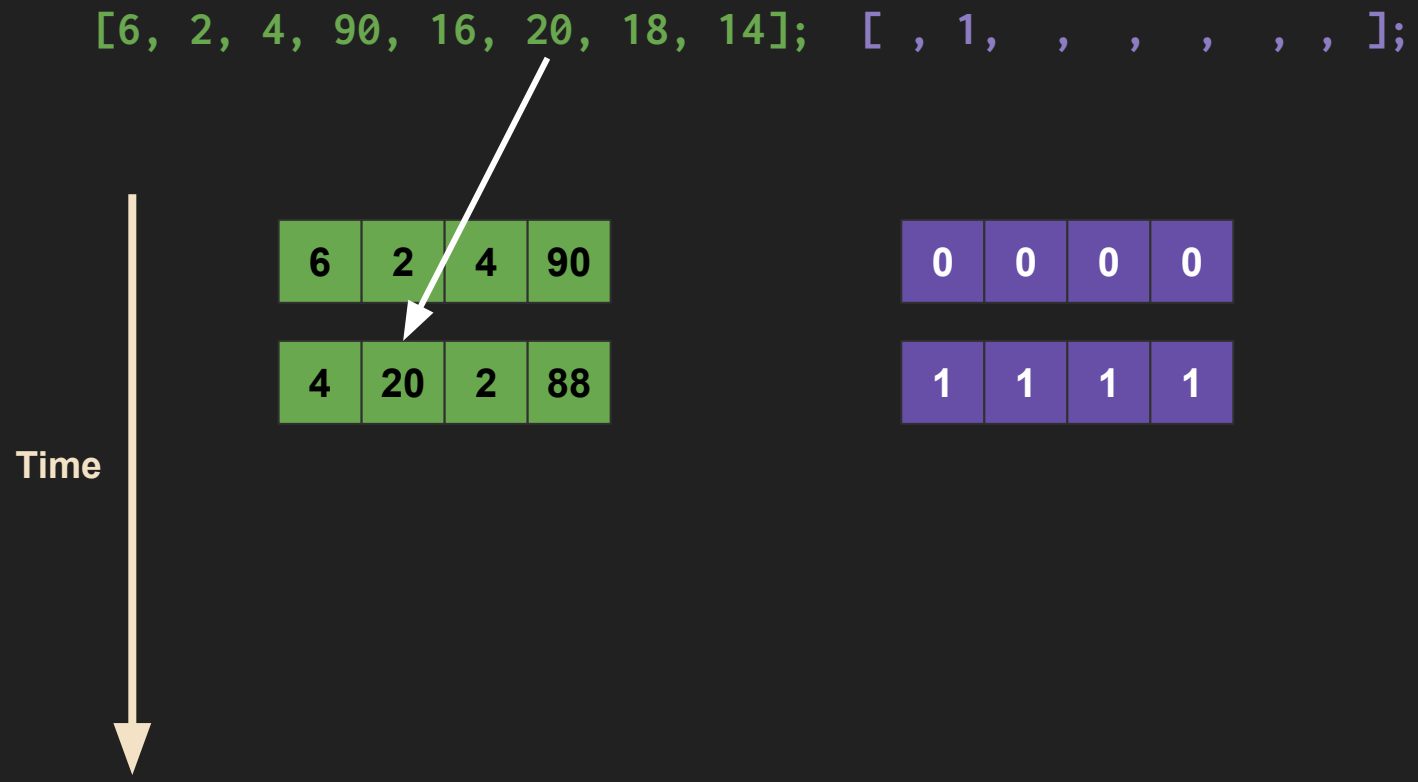


Dynamically Vectorized Reduction

```
[6, 2, 4, 90, 16, 20, 18, 14];  [ , 1,  ,  ,  ,  , , ];
```

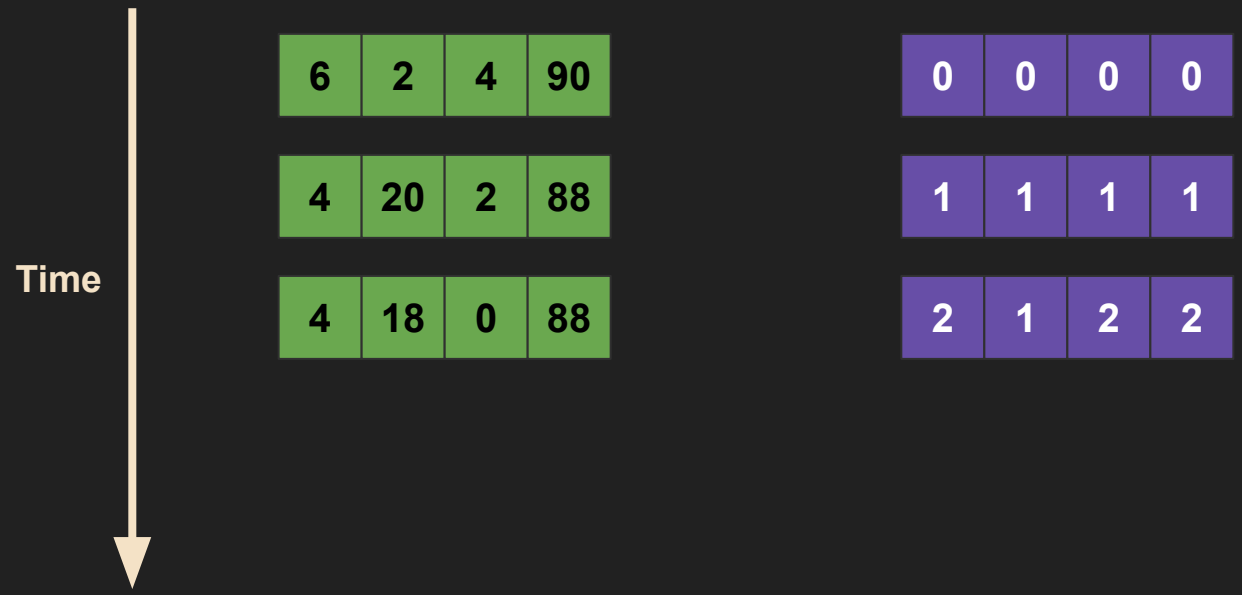


Dynamically Vectorized Reduction



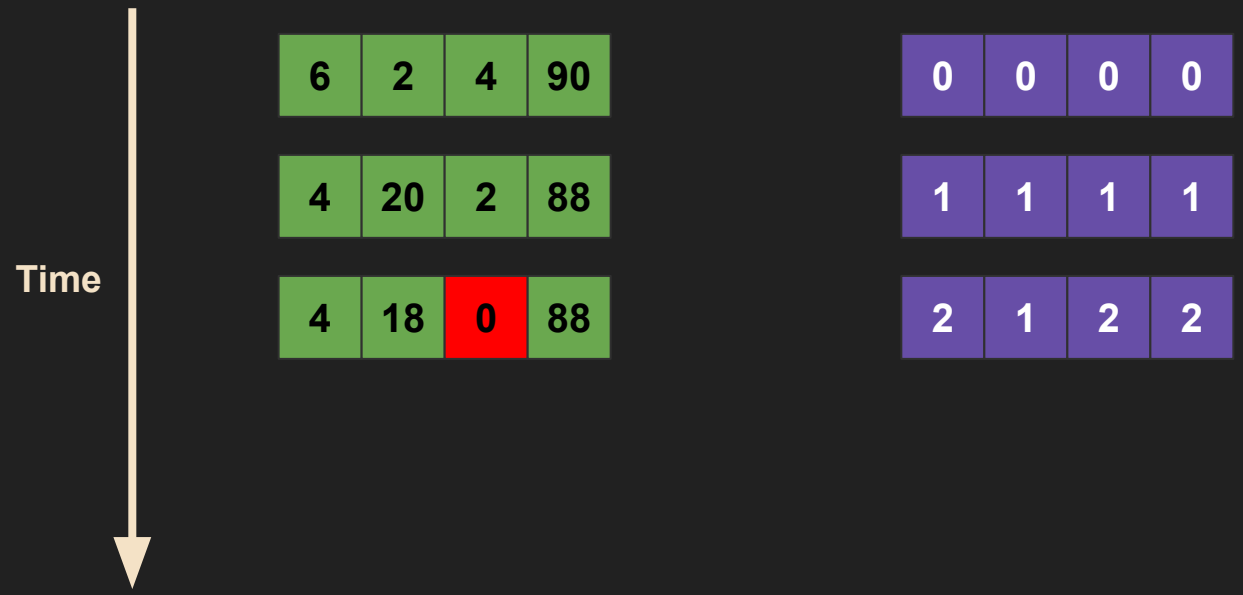
Dynamically Vectorized Reduction

```
[6, 2, 4, 90, 16, 20, 18, 14];  [ , 1,  ,  ,  ,  , , ];
```



Dynamically Vectorized Reduction

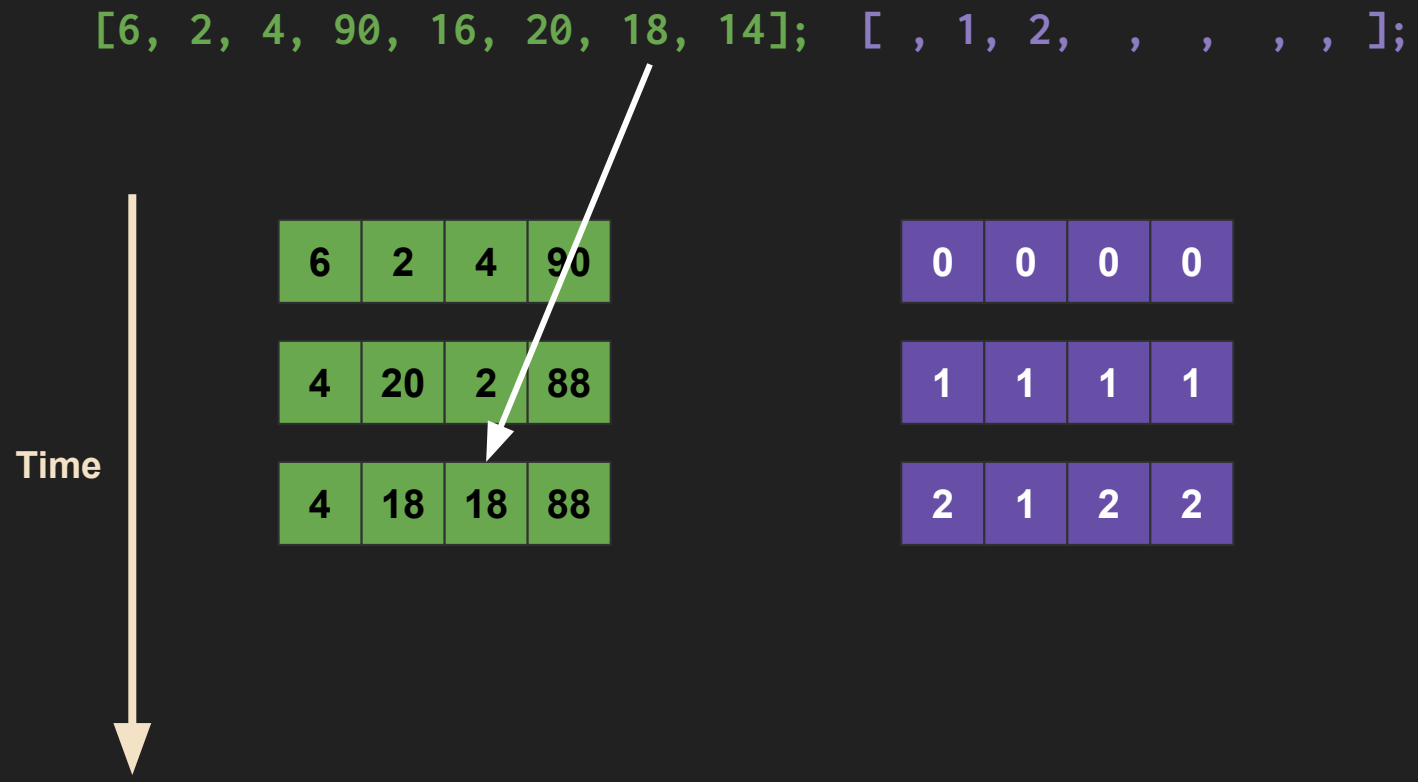
```
[6, 2, 4, 90, 16, 20, 18, 14];  [ , 1,  ,  ,  ,  , , ];
```



Dynamically Vectorized Reduction

```
[6, 2, 4, 90, 16, 20, 18, 14];  [ , 1, 2,  ,  ,  , , ];
```

Dynamically Vectorized Reduction




HOW FAST
IS IT?

1.25x faster than statically vectorized

Thank You!

Thanks to:

- Liberty Research Group at Princeton
- Dr. Kulkarni’s group at Purdue
- Simon Moll

for the feedback