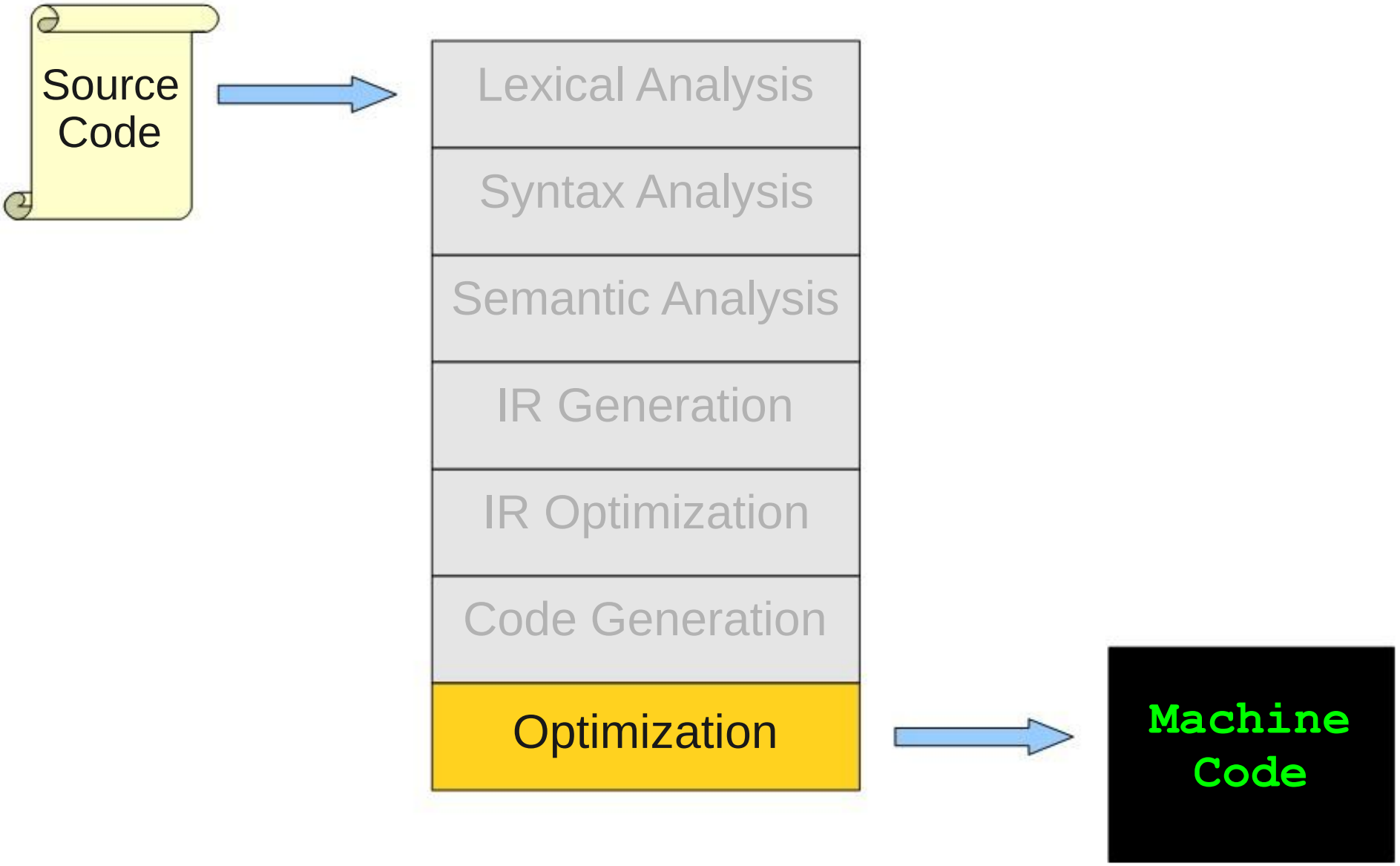
Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

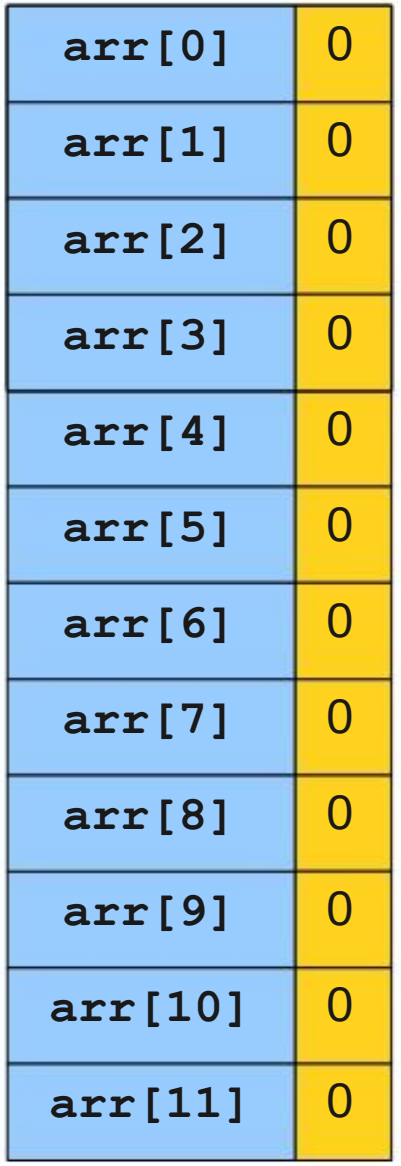
Optimizations for Locality

Locality

- Empirically, many programs exhibit **temporal locality** and **spatial locality**.
- **Temporal locality**: Memory read recently is likely to be read again in the near future.
- **Spatial locality**: Memory read recently will likely have nearby objects read as well.
- Most memory caches are designed to exploit temporal and spatial locality by
  - Holding recently-used memory addresses in cache.
  - Loading nearby memory addresses into cache.

Memory Caches

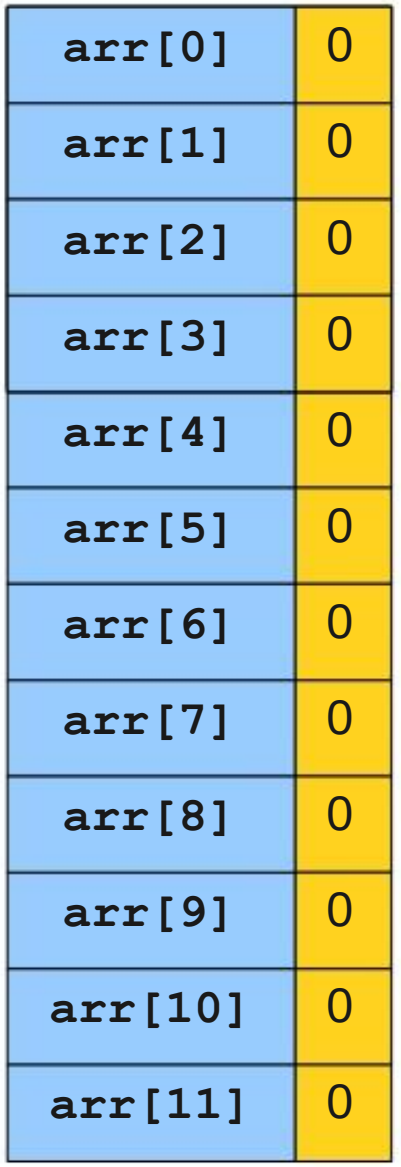
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

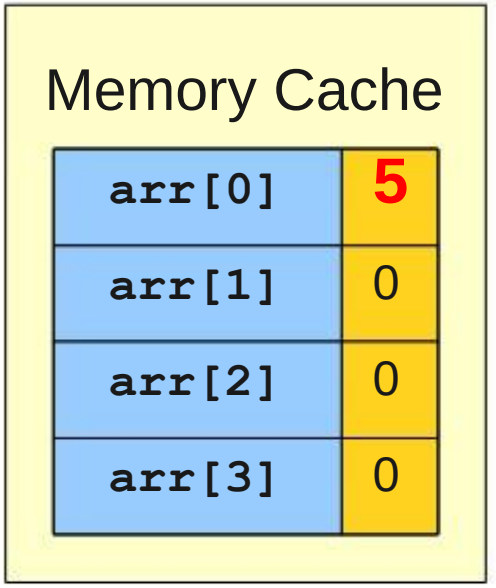


Memory Cache

Memory Caches

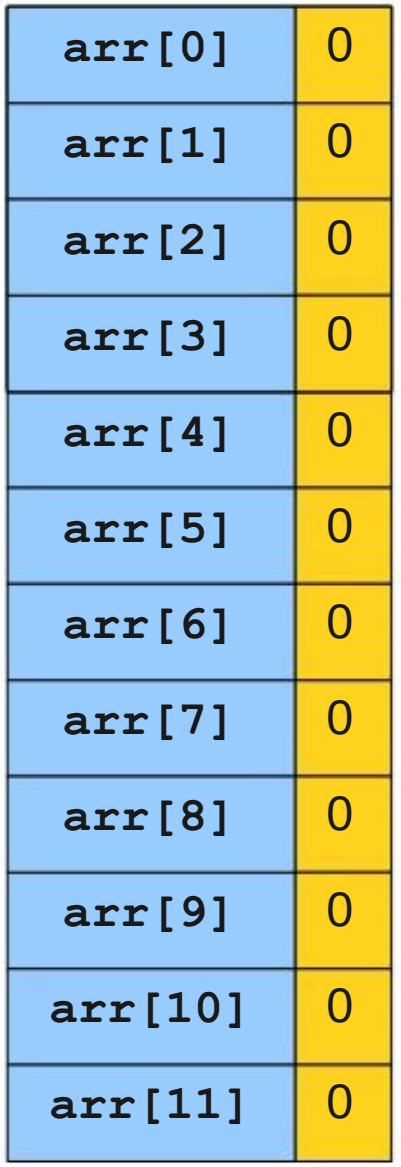
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```



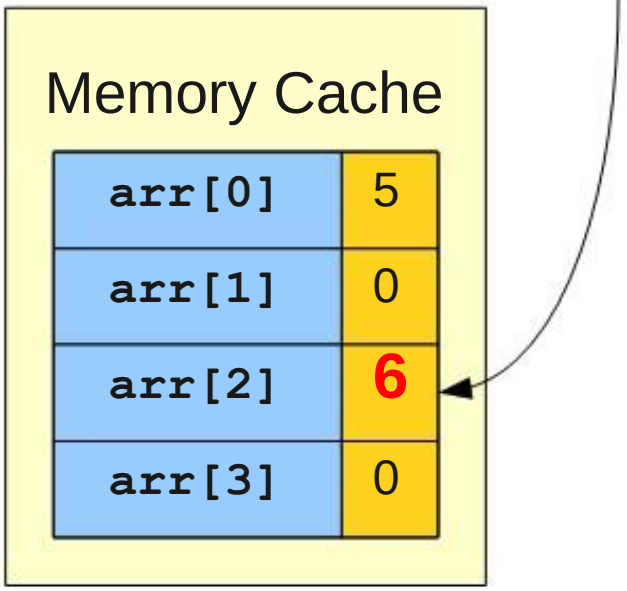


Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

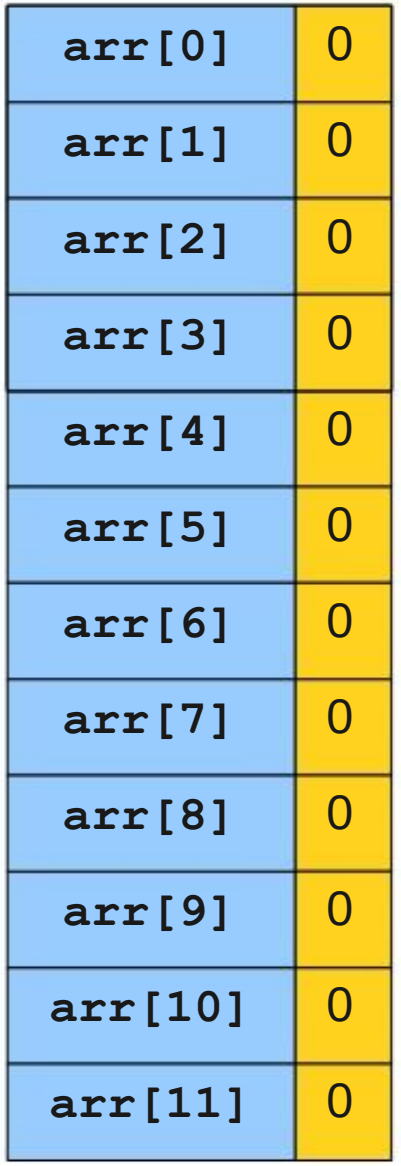


arr[2] is already in cache!

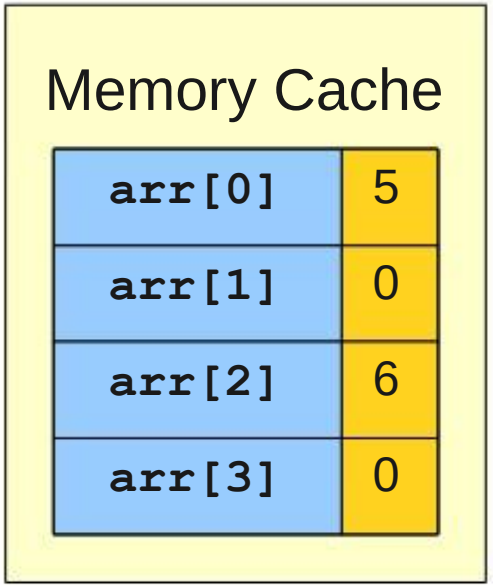


Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

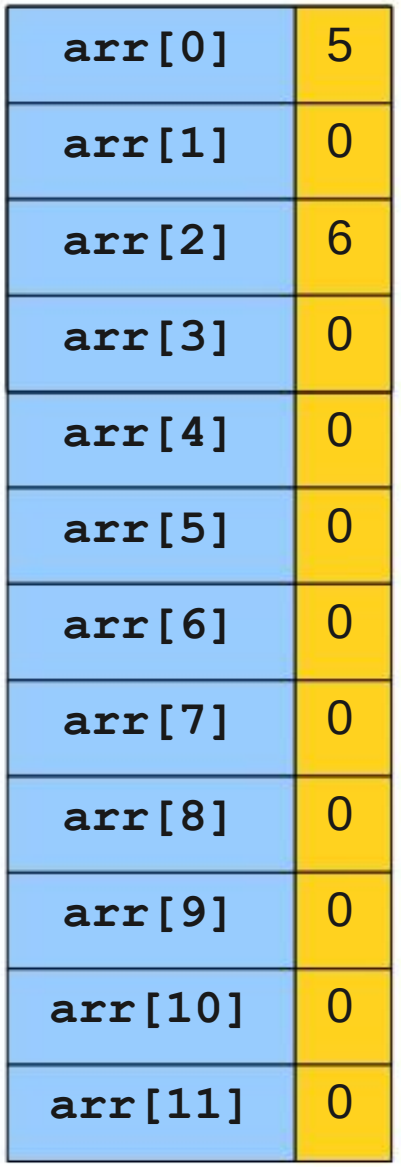


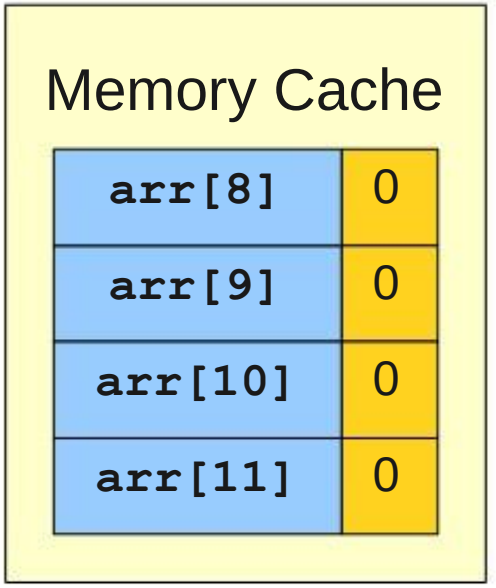
Cache miss!



Memory Caches

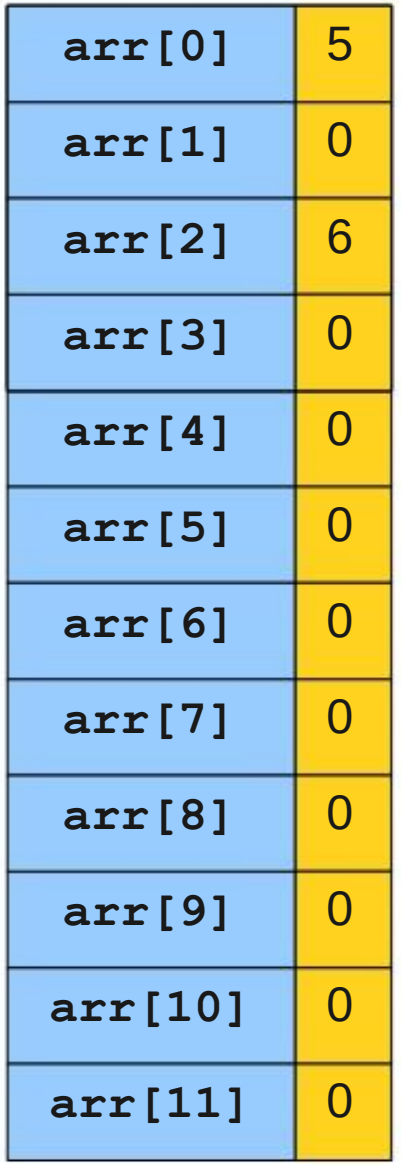
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

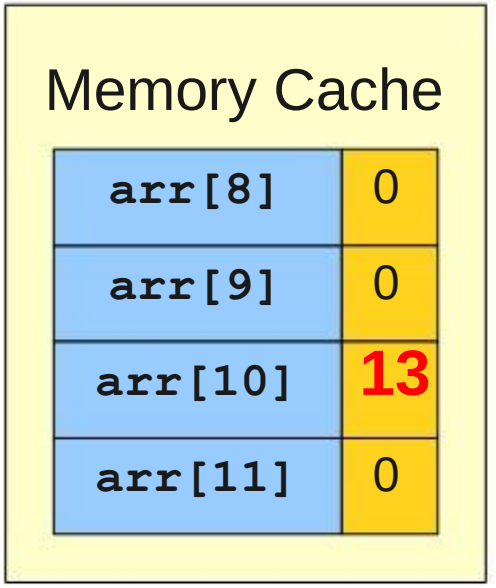




Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```





Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	0
arr[11]	0

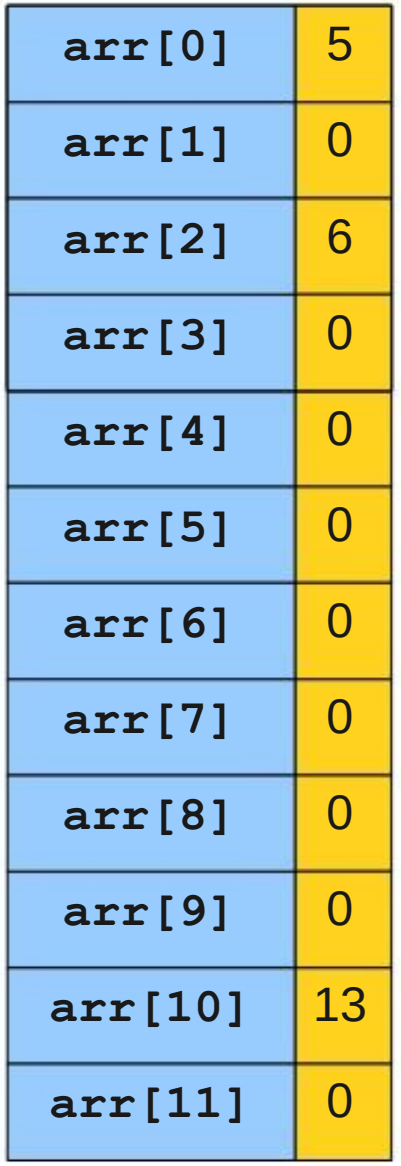
Cache miss again!

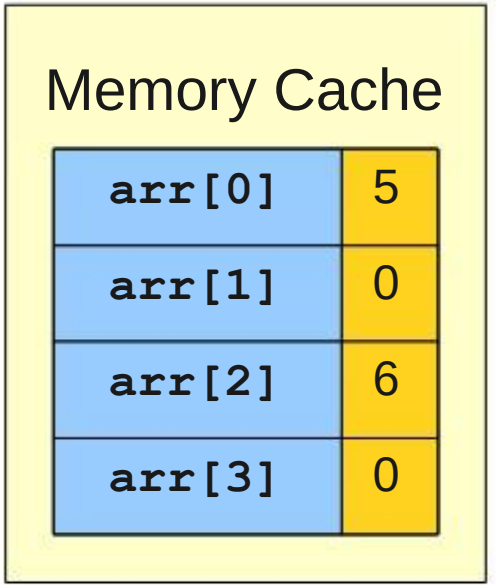
Memory Cache

arr[8]	0
arr[9]	0
arr[10]	13
arr[11]	0

Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```





Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

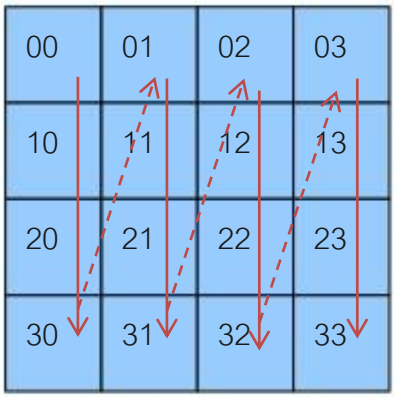
arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	13
arr[11]	0

Memory Cache

arr[0]	5
arr[1]	4
arr[2]	6
arr[3]	0

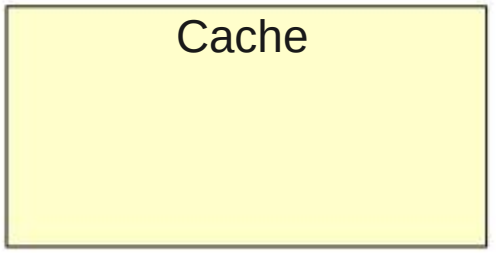
The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



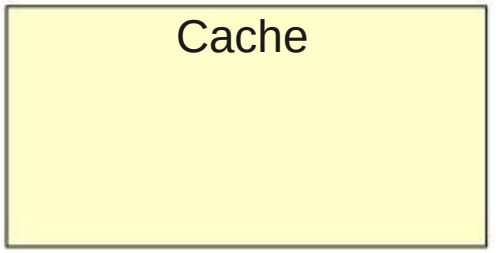
The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



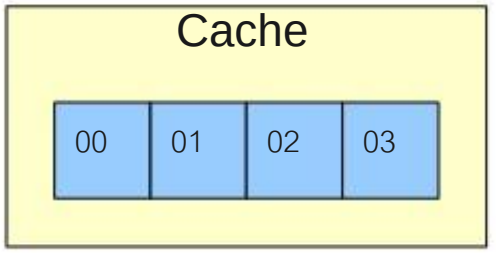
The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



The Problem with Caches

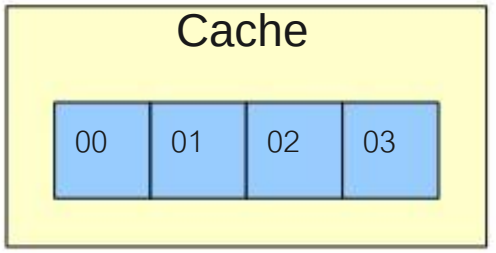
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



The Problem with Caches

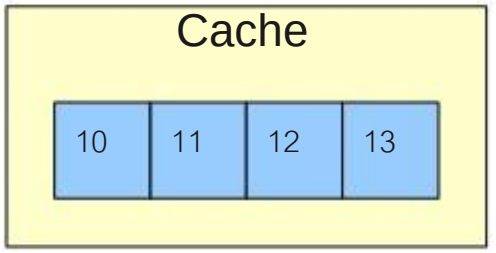
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

Cache miss!



The Problem with Caches

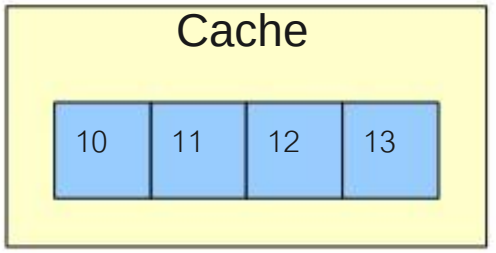
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



The Problem with Caches

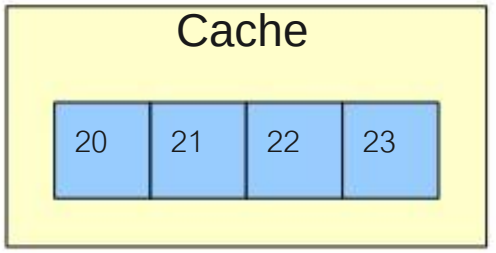
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

Cache miss!



The Problem with Caches

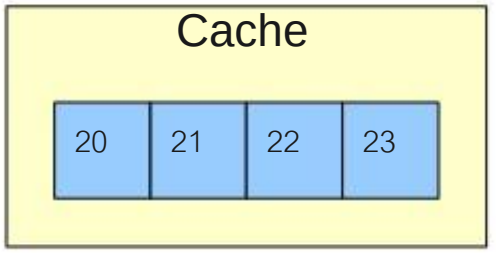
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



The Problem with Caches

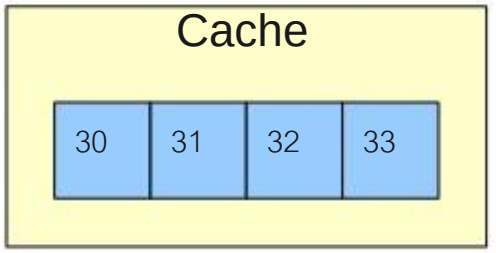
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

Cache miss!



The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



Programmers frequently write code without understanding the locality implications.

Structure Peeling

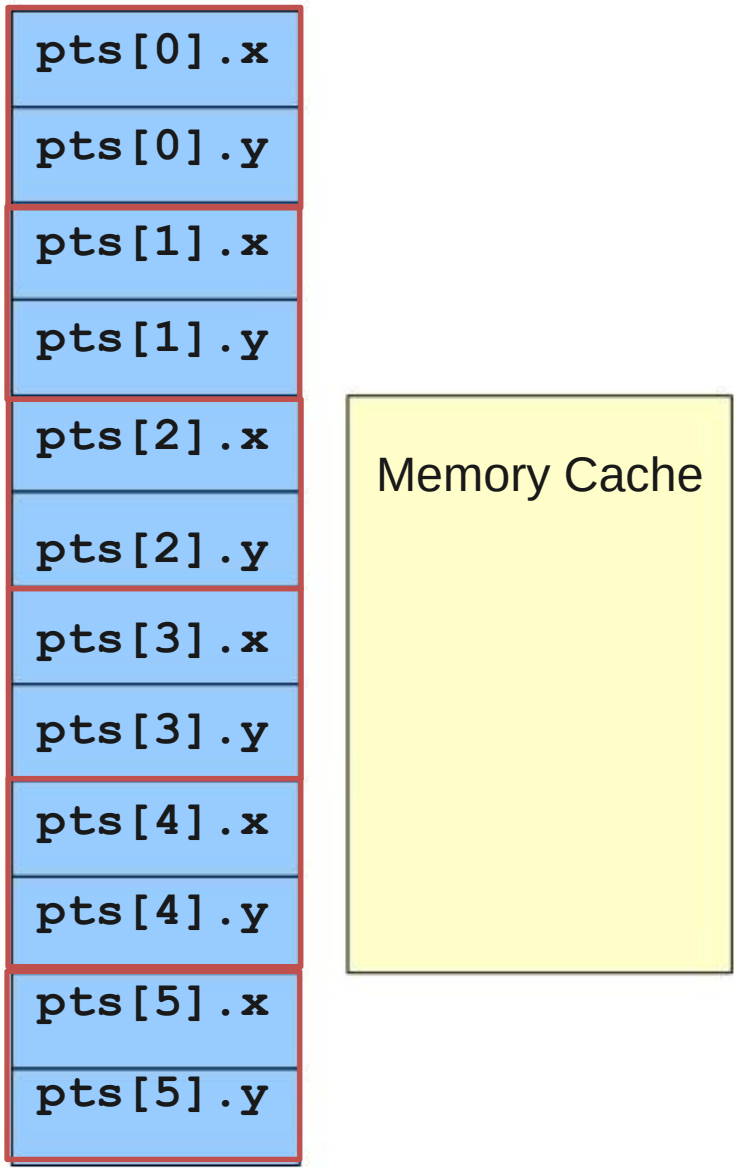
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

Another way to improve locality



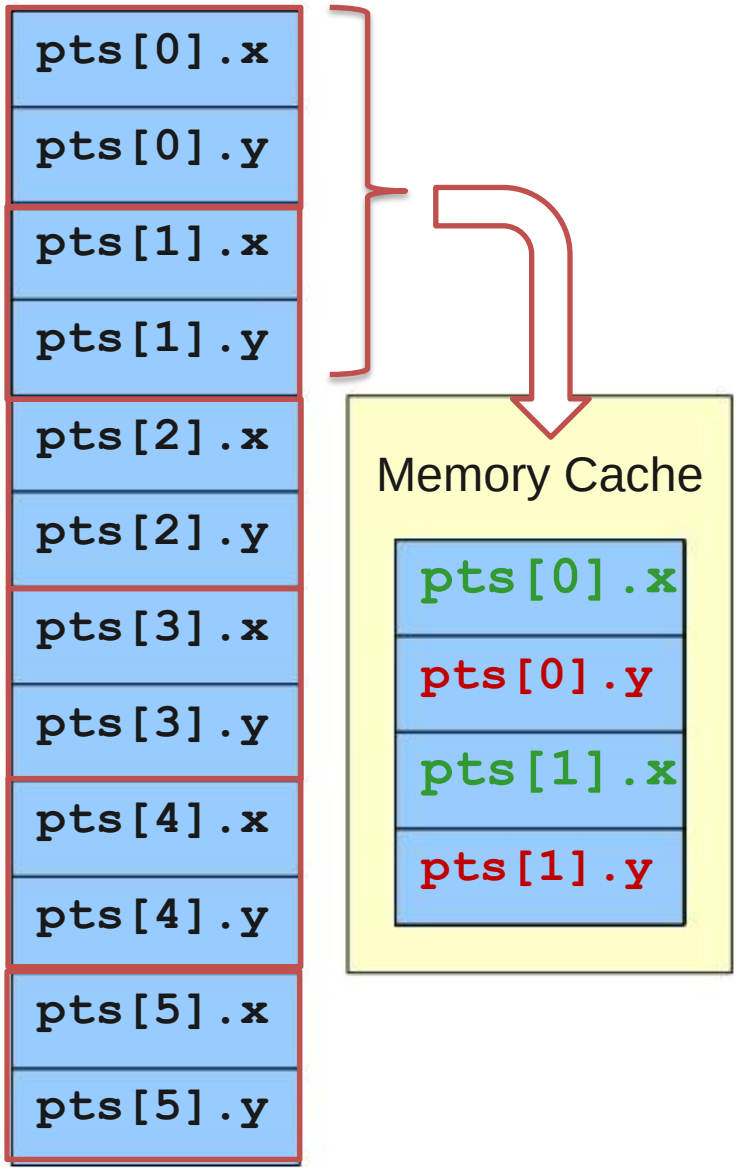
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



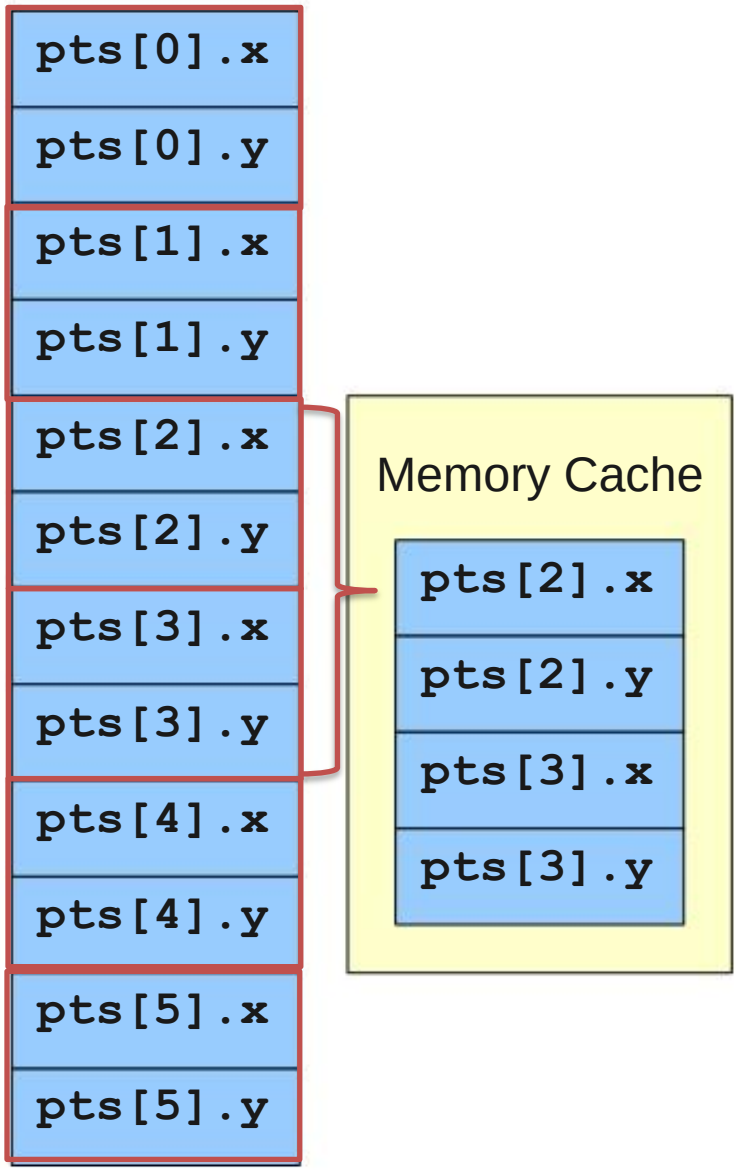
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



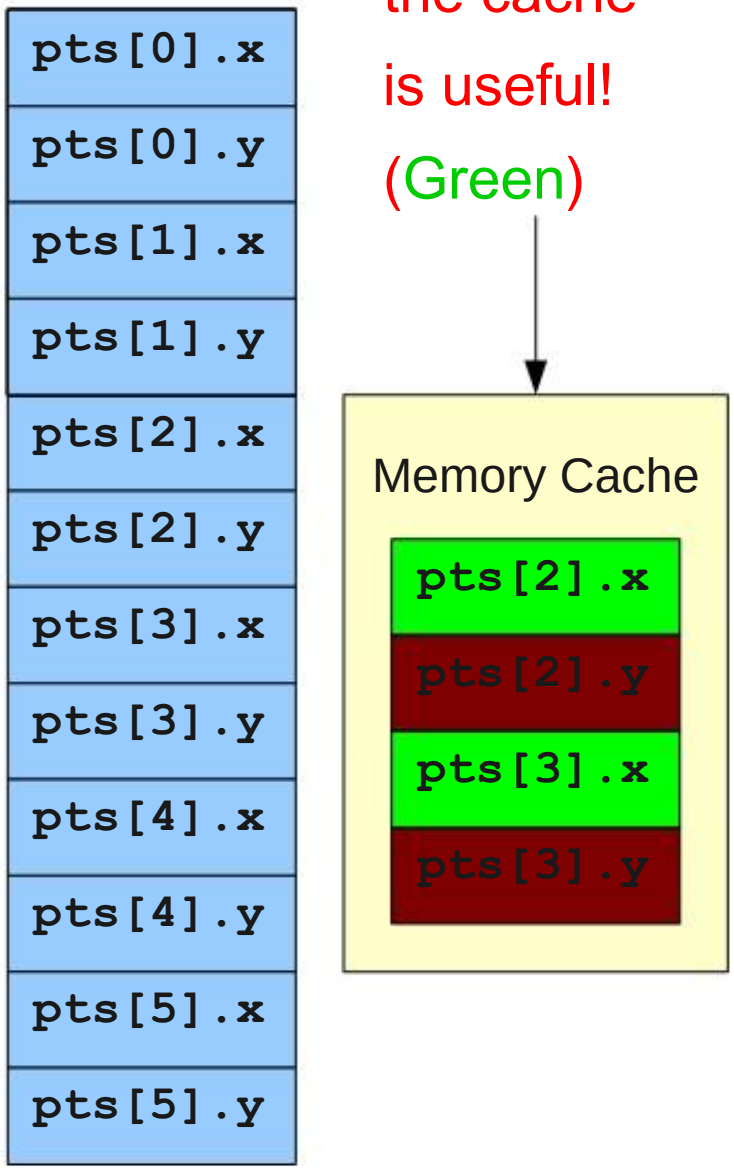
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



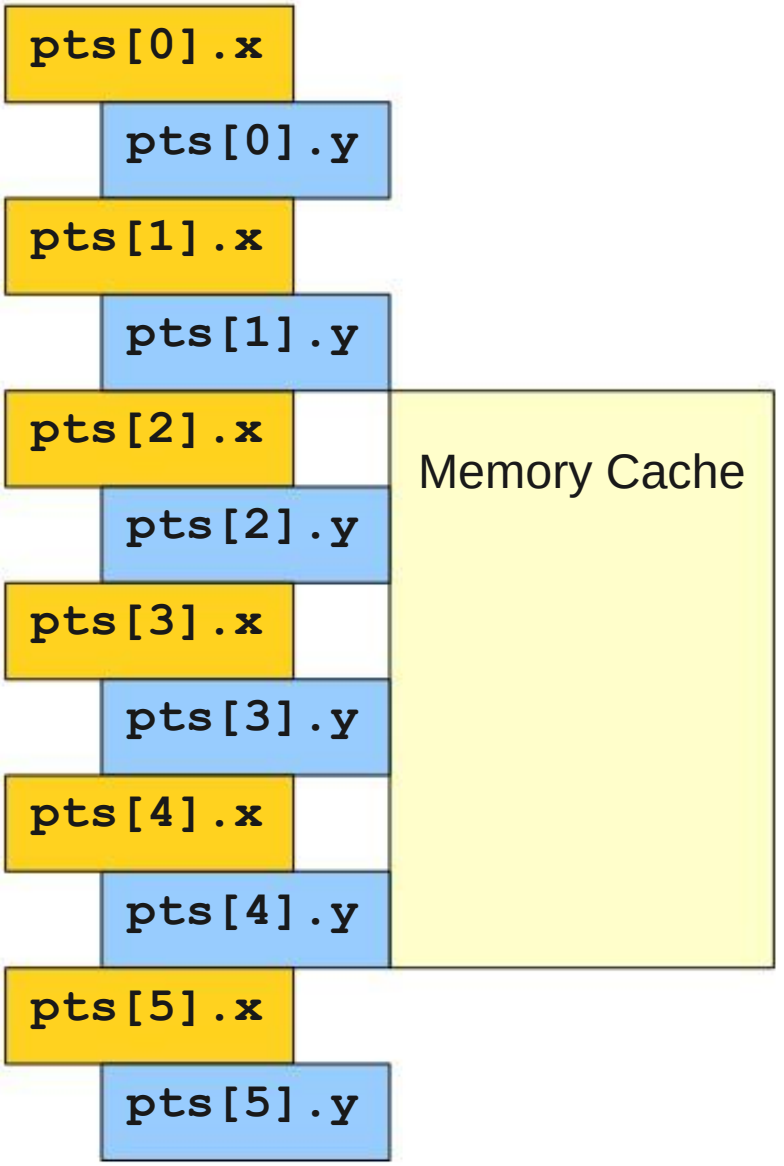
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



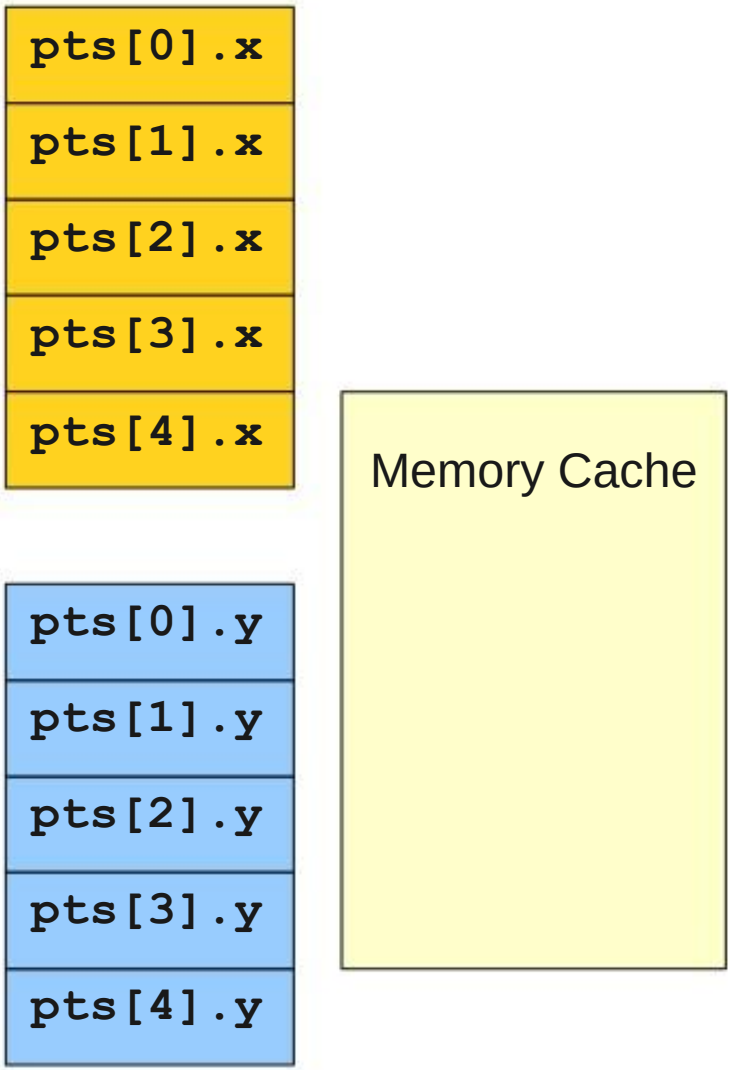
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



Internally restructure by the smart compiler

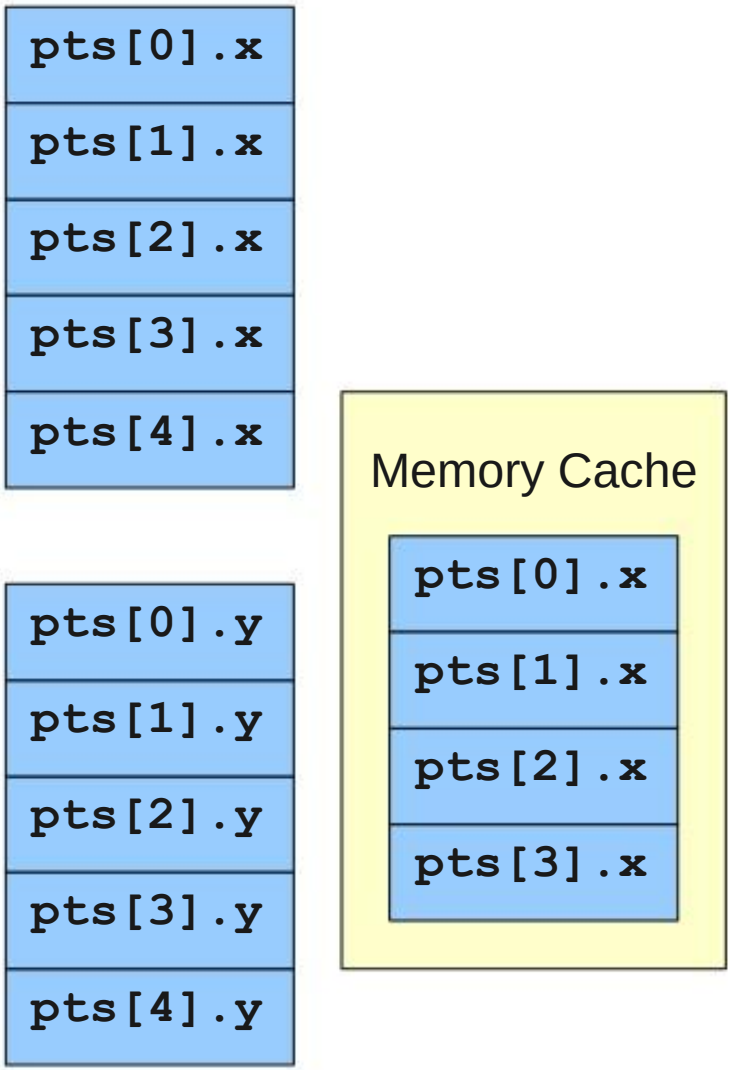
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



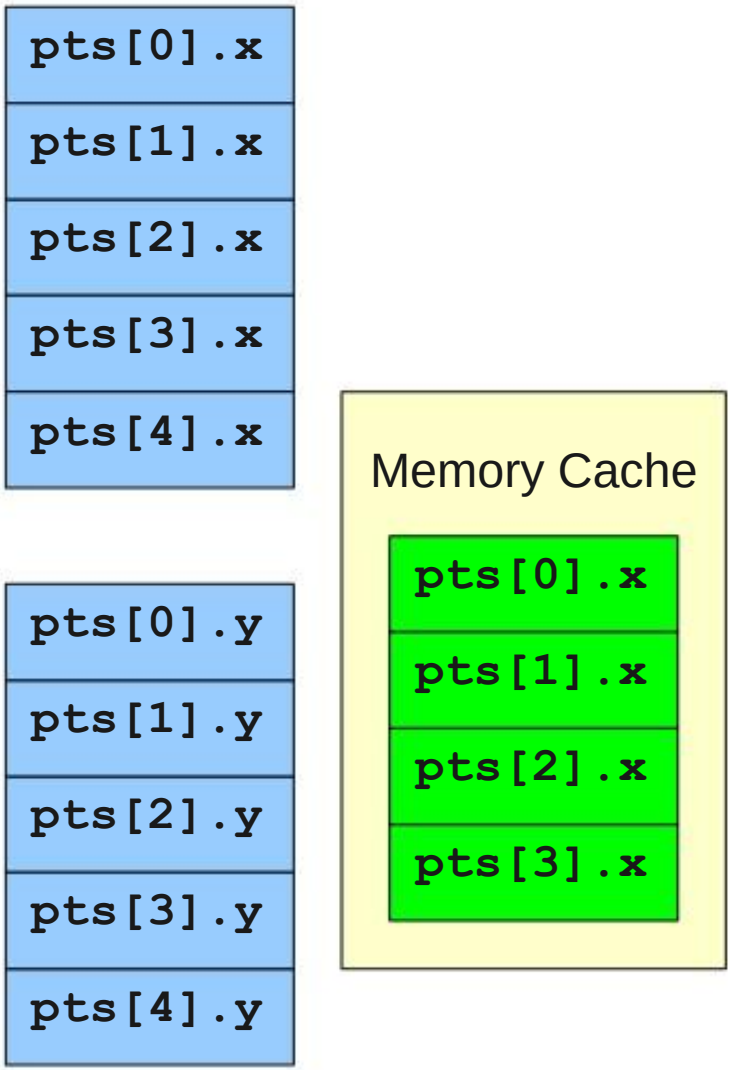
Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



Optimizations for Parallelism

Loop Parallelization

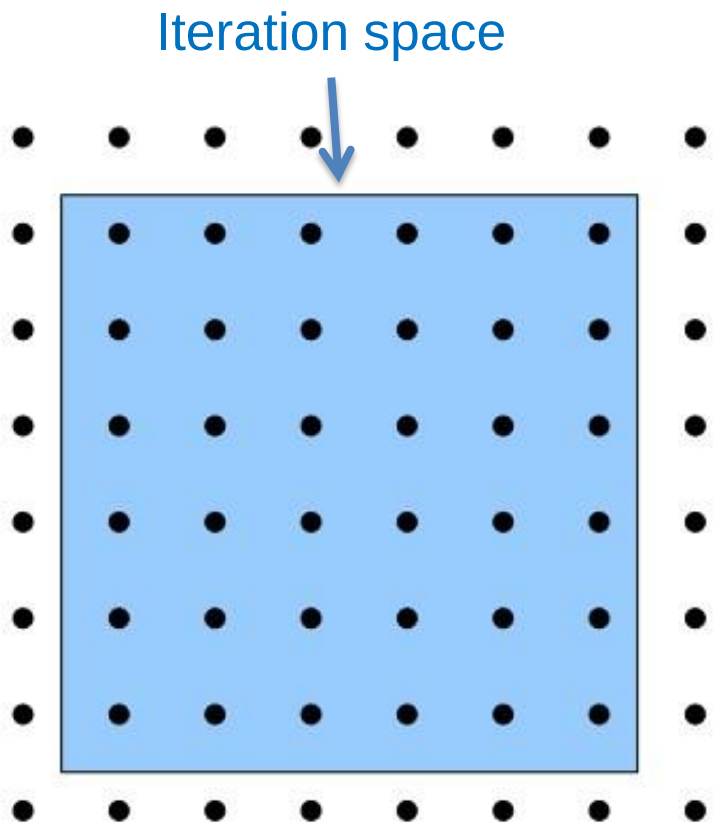
(For multi-processor machine)

- Optimize loops over arrays by identifying inherent parallelism.

- Three-step process:
  - **Identify** which array values depend on one another.
  - **Split** each group of dependent values into its own task.
  - **Map** each task onto one processor.

Iteration Spaces

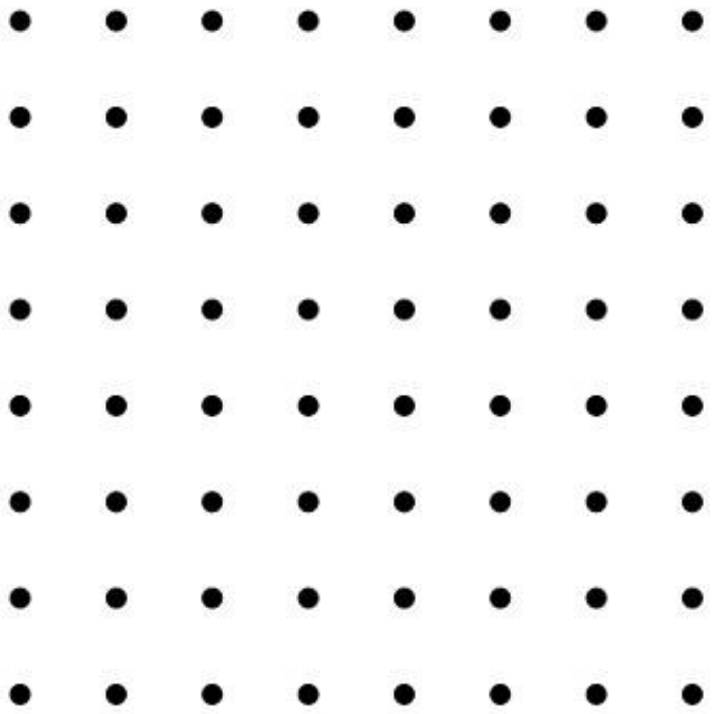
- The **iteration space** of a set of loops is the set of valid indices referenced by the loop counters.



Each point represents an element in an array

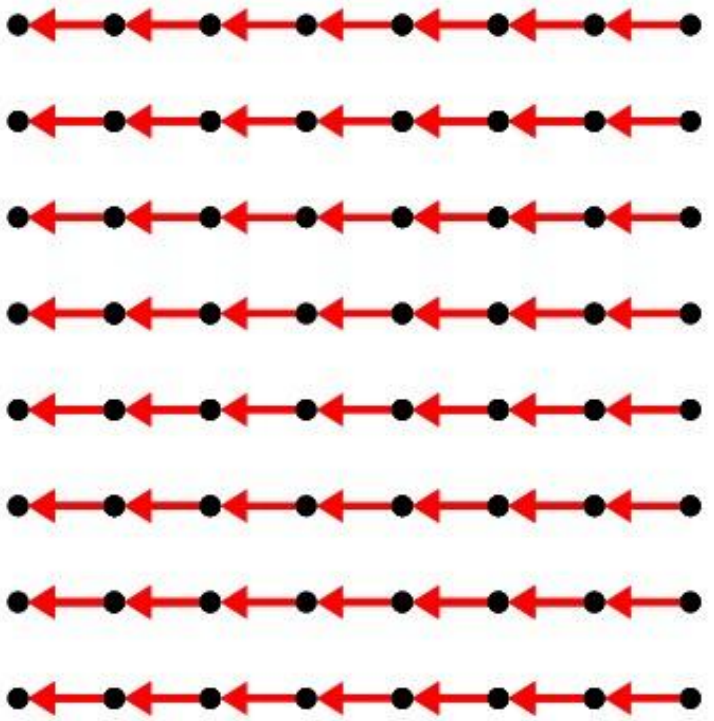
Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```



Identifying Dependent Accesses

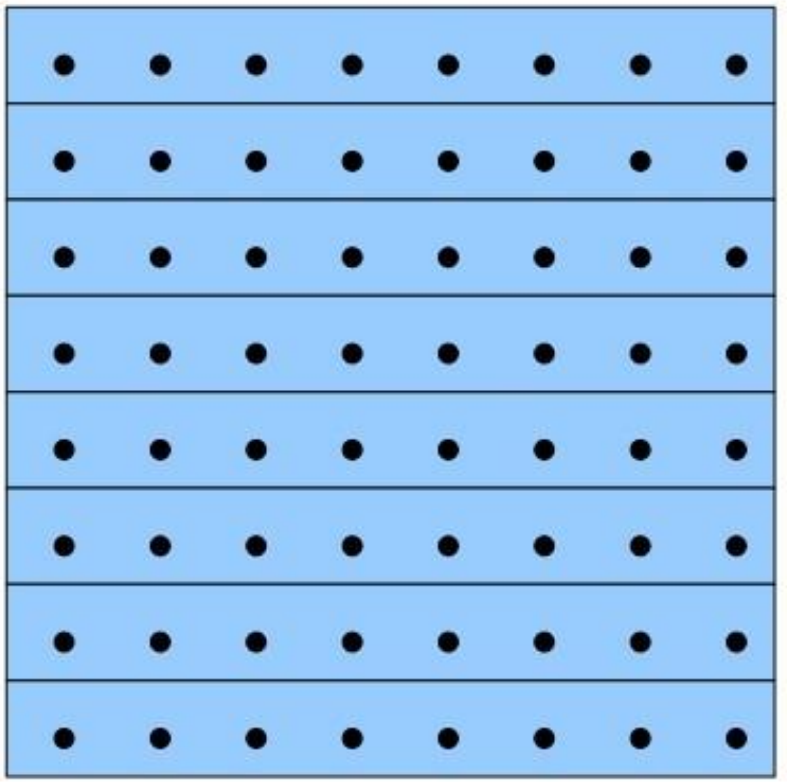
```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```



Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

Dependency in row



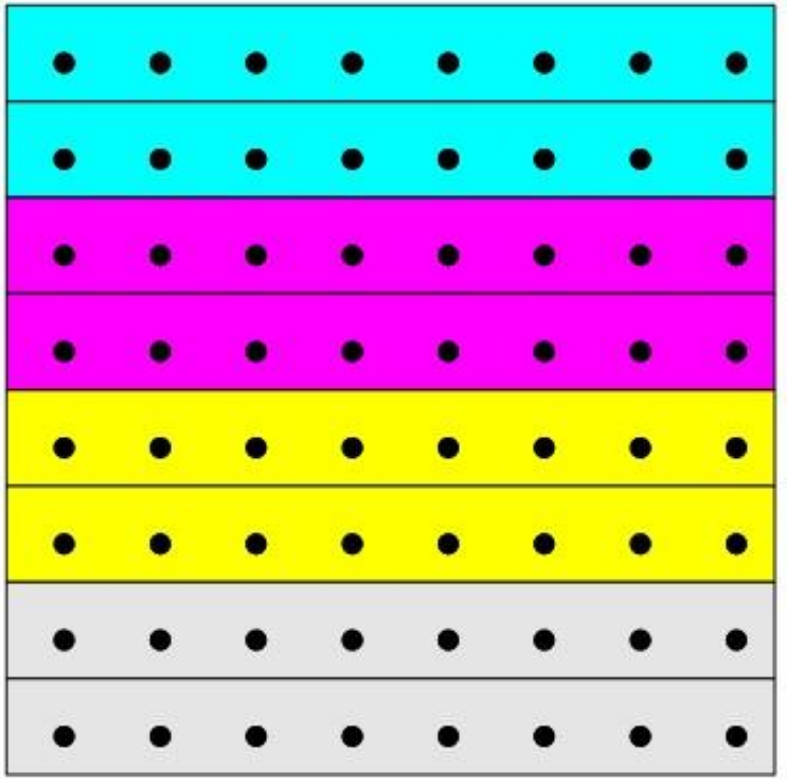
- When assigning iterations to processors:
  - Pick an assignment that is **good for locality**.
  - Pick an assignment that **maximizes the degree of parallelism**.

Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

Processors

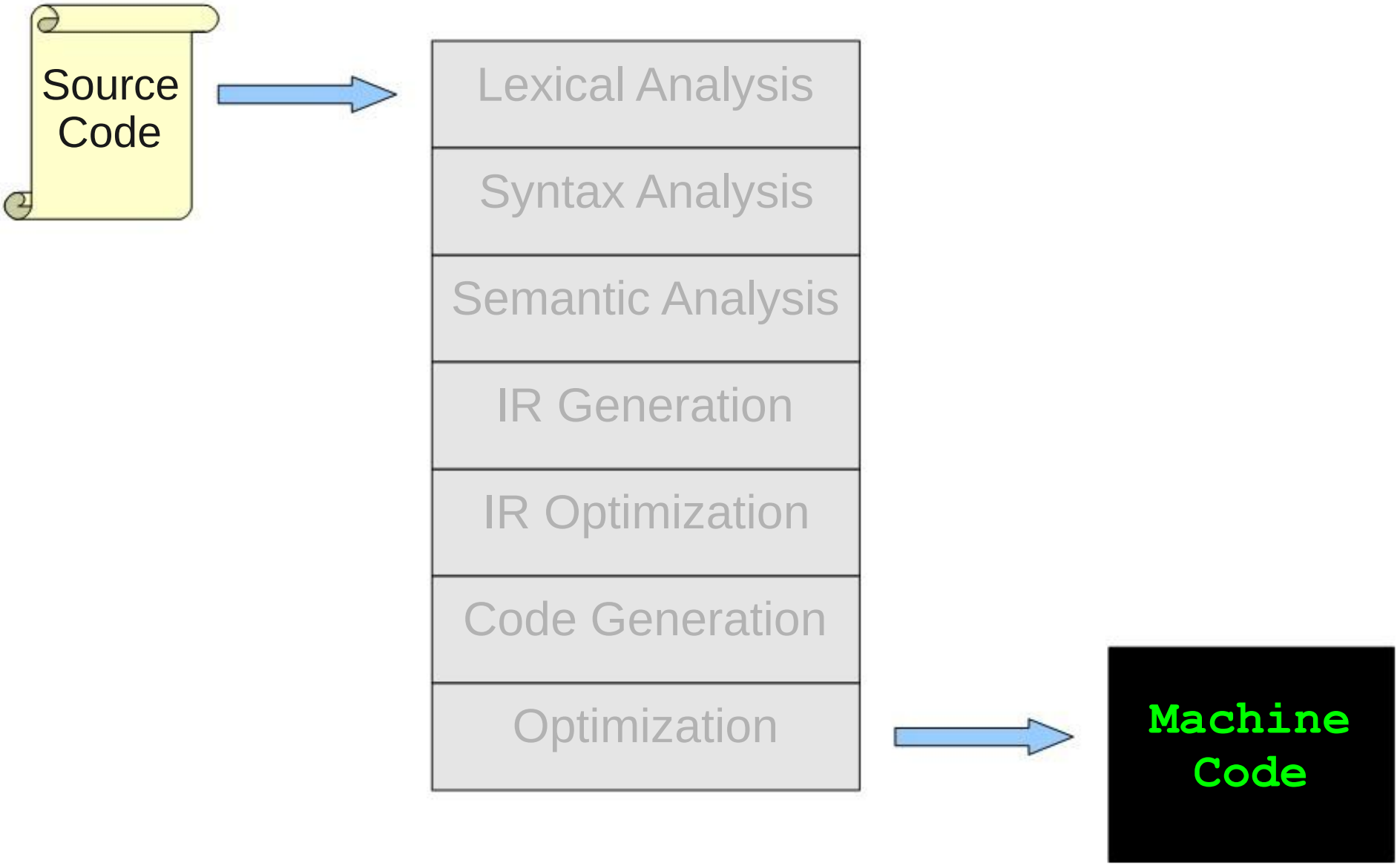
Good



Summary

- **Instruction scheduling** optimizations try to take advantage of the processor pipeline.
- **Locality** optimizations try to take advantage of cache behavior.
- **Parallelism** optimizations try to take advantage of multicore machines.
- There are *many more* optimizations out there!

Where We've Been



Why Study Compilers? (Recap)

- Build a **large, ambitious software system**.
- See theory **come to life**.
- Learn how to **build programming languages**.
- Learn **how programming languages work**.
- Learn **tradeoffs in language design**.