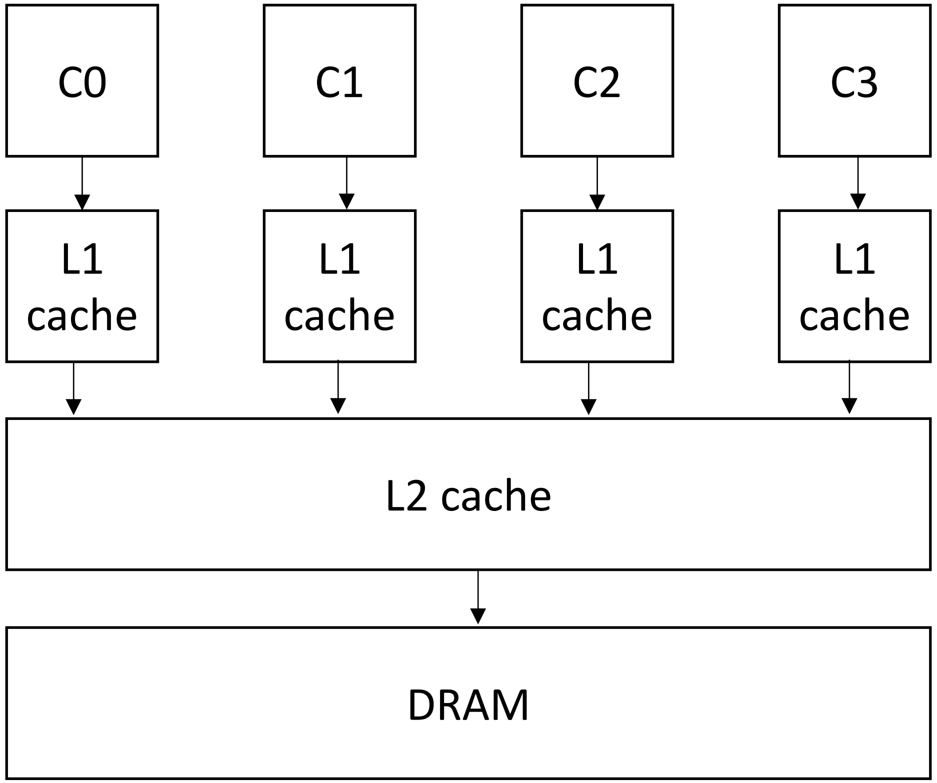
CSE211: Compiler Design

Nov. 5, 2021

- **Topic**: restructuring loops



Announcements

- Homework 3 is due Nov. 17
  - 1 more office hour before then (next Thursday)
  - part 1 and 2: generating c code from python
  - part 3: creating and checking z3 constraints

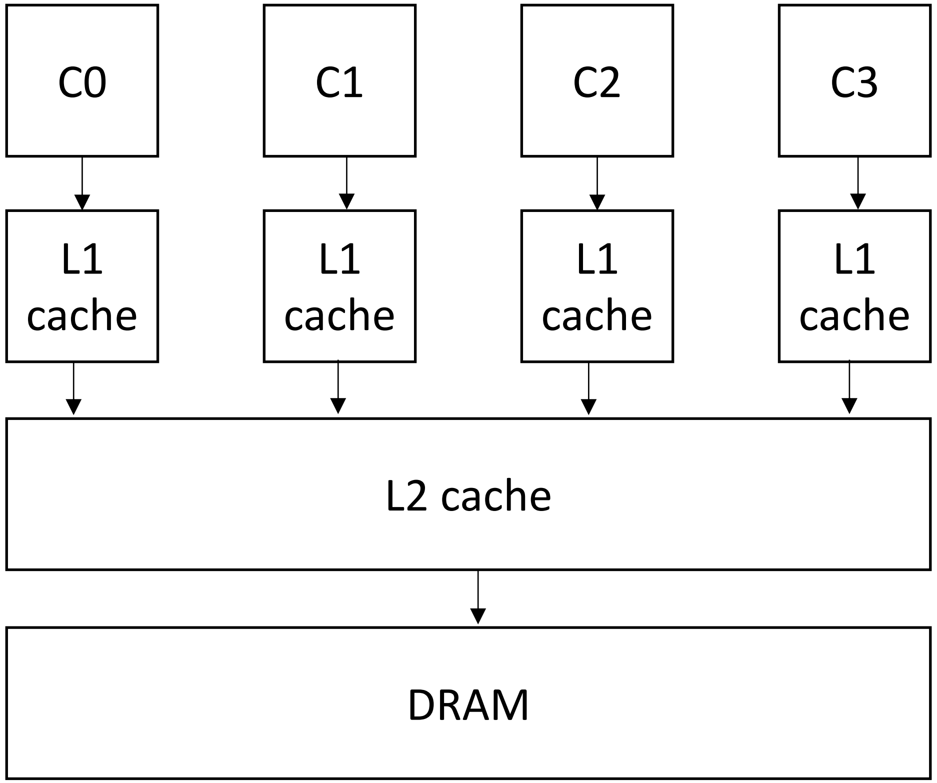
Paper/Project proposals

- Please start thinking about these.
  - Message me for recommendations
  - Tell me what you're interested in so we can find a good fit!
- Proposals due on Nov. 14 (less than 2 weeks)
  - Please be pro-active about this. If you don't have one in mind, please send me an email with some of your interests ASAP
- Midterm is a good indicator for how the final will be.

CSE211: Compiler Design

Nov. 3, 2021

- **Topic**: restructuring loops



Review

- Compiler approach for checking if DOALL loops are safe to do in parallel
  - What is a DOALL loop?
  - What conditions are required for safety?

Review

- Creating constraints

```
for (i = 0; i < 128; i++) {
   a[i]= a[i]*2;
}
```

two integers: i_x != i_y

$i_x >= 0$

$i_x < 128$

$i_y >= 0$

$i_y < 128$

write-write conflict $i_x == i_y$

read-write conflict $i_x == i_y$

Ask if these constraints are satisfiable (if so, it is not safe to parallelize)

Review: another example

```
for (i = 0; i < 128; i++) {
   a[i%64]= a[i+64]*2;
}
```

push bounds constraints

two integers: i_x != i_y

i_x >= 0

i_x < 128

i_y >= 0

i_y < 128

Review: another example

```
for (i = 0; i < 128; i++) {
   a[i%64]= a[i+64]*2;
}
```

two integers: i_x != i_y

i_x >= 0

i_x < 128

i_y >= 0

i_y < 128

push bounds constraints

i_x % 64 == i_y % 64

write-write conflict checking

Review: another example

```
for (i = 0; i < 128; i++) {
   a[i%64]= a[i+64]*2;
}
```

two integers: i_x != i_y

i_x >= 0

i_x < 128

i_y >= 0

i_y < 128

push bounds constraints

pop

Review: another example

```
for (i = 0; i < 128; i++) {
   a[i%64]= a[i+64]*2;
}
```

two integers: i_x `!=` i_y

$i_x >= 0$

$i_x < 128$

$i_y >= 0$

$i_y < 128$

push bounds constraints

$i_x \,\%\, 64 == i_y + 64$

read-write conflict checking

Moving onto loop structures

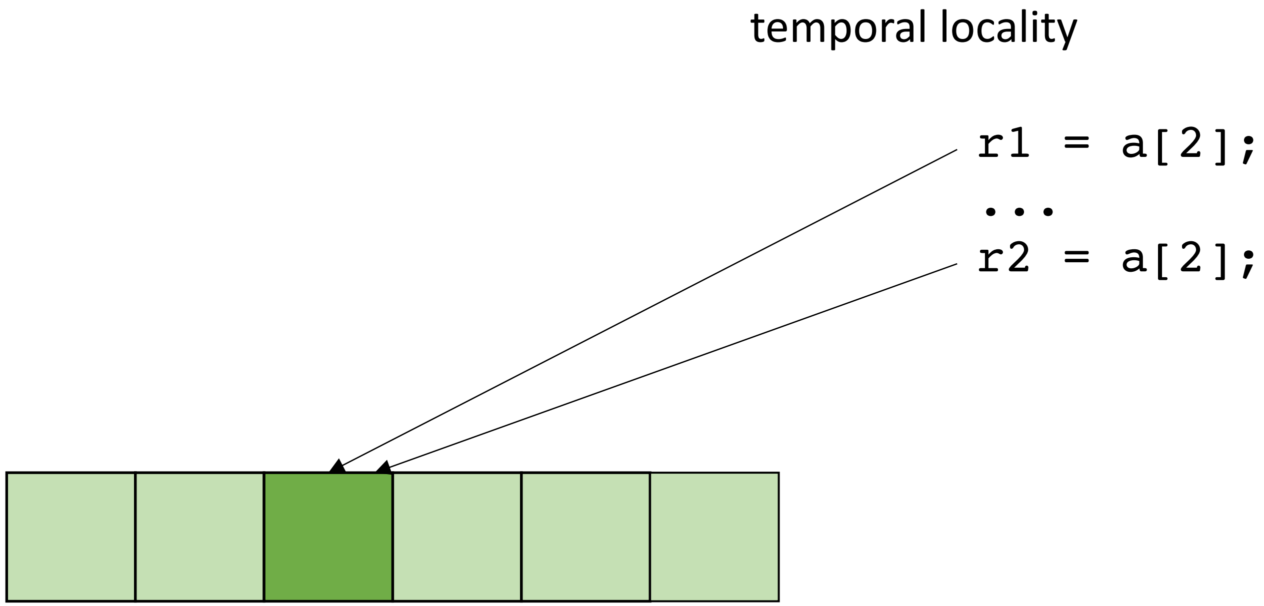
Transforming Loops

- Locality is key for good parallel performance:

Transforming Loops

- Locality is key for good parallel performance:

- Two types of locality:
  - Temporal locality
  - Spatial locality



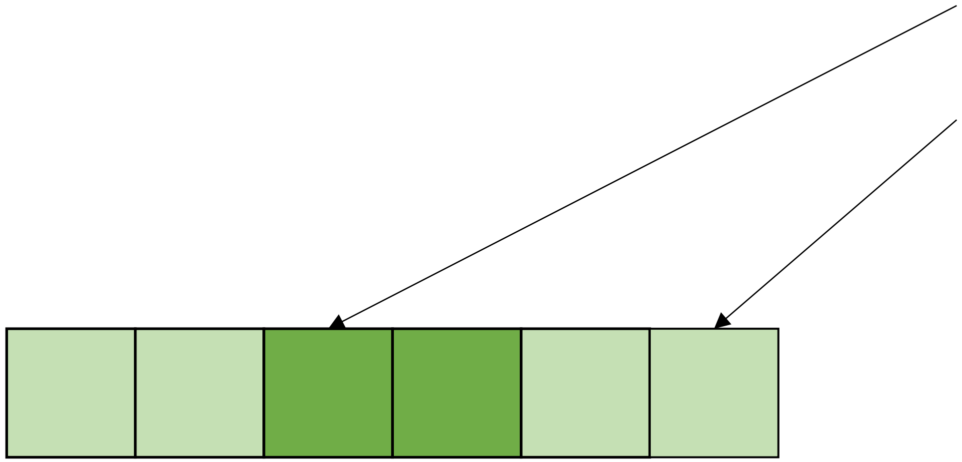
Transforming Loops

- Locality is key for good parallel performance:

- Two types of locality:
    - Temporal locality
    - Spatial locality

spatial locality

```
r1 = a[2];
...
r2 = a[3];
```

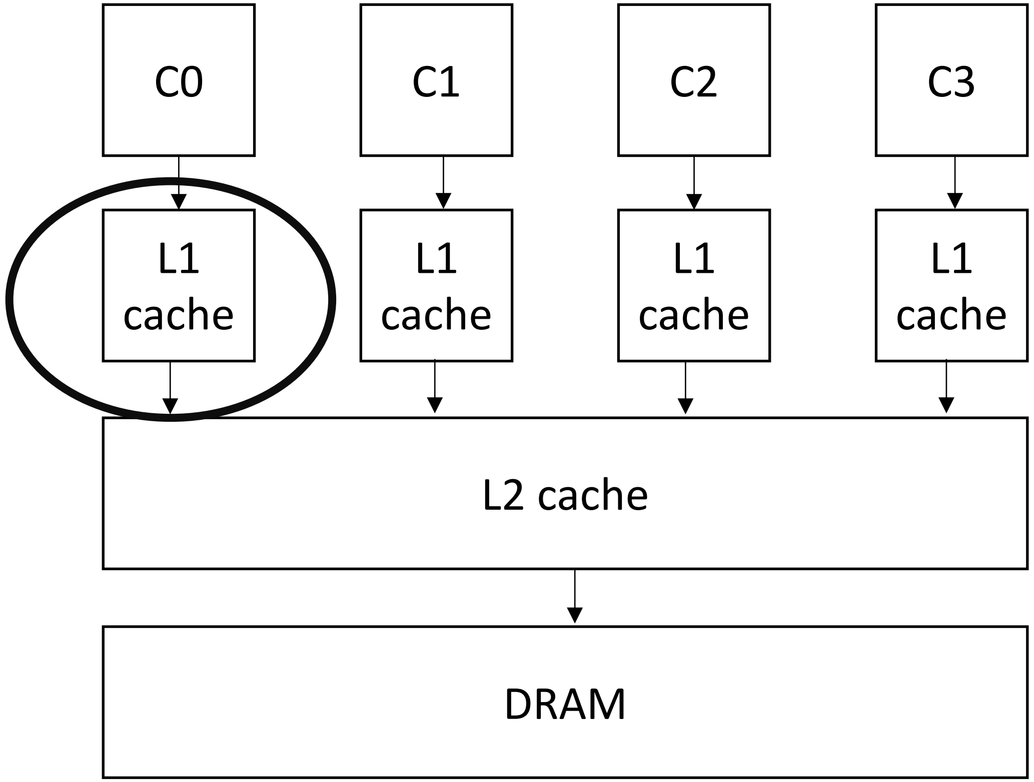


how far apart can memory locations be?

Transforming Loops

- Locality is key for good parallel performance:

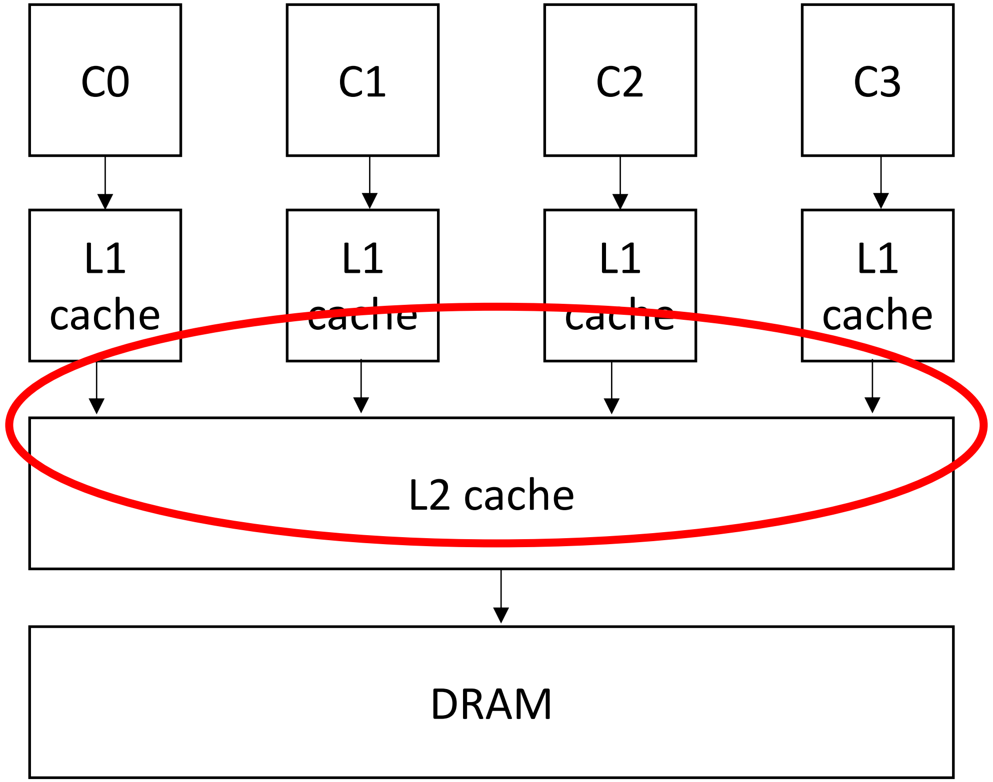
good data locality: cores will
spend most of their time accessing
private caches



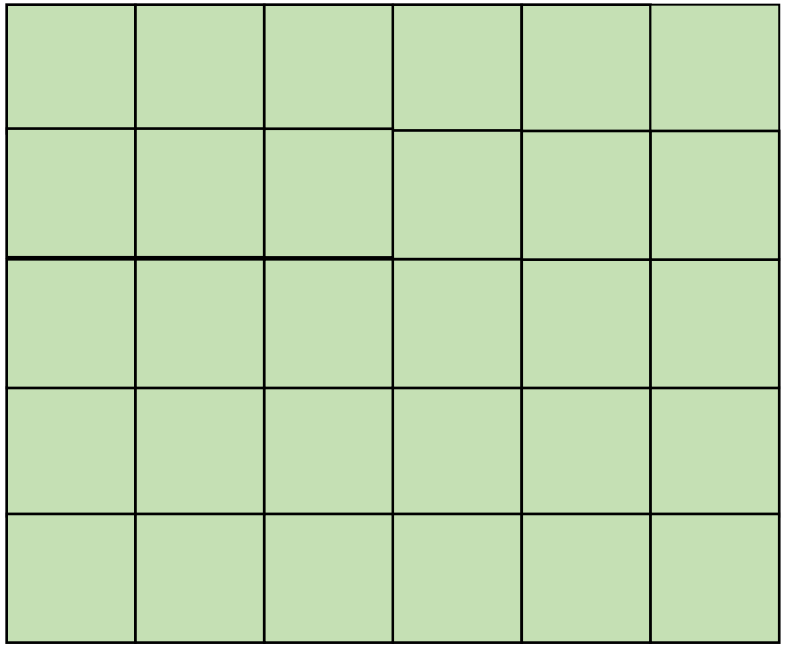
Transforming Loops

- Locality is key for good parallel performance:

Bad data locality: cores will pressure and thrash shared memory resources

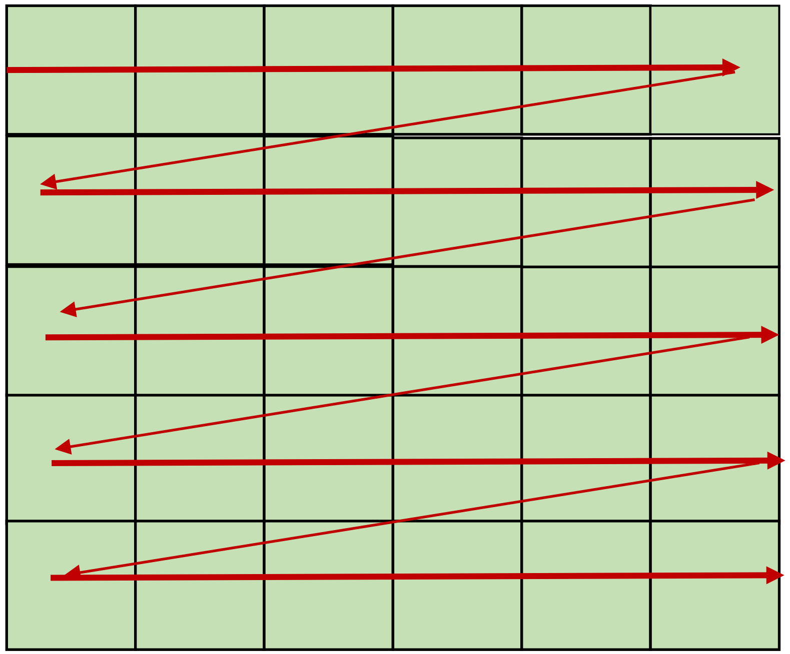


How multi dimensional arrays are stored:



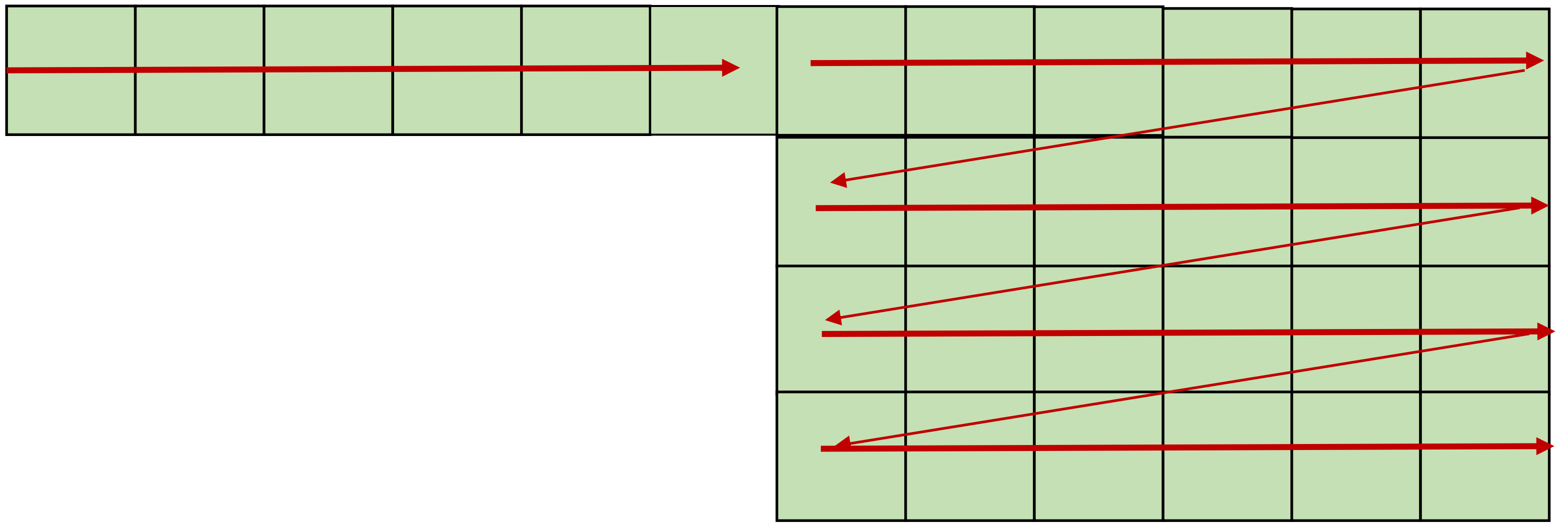
How multi dimensional arrays are stored:

Row major



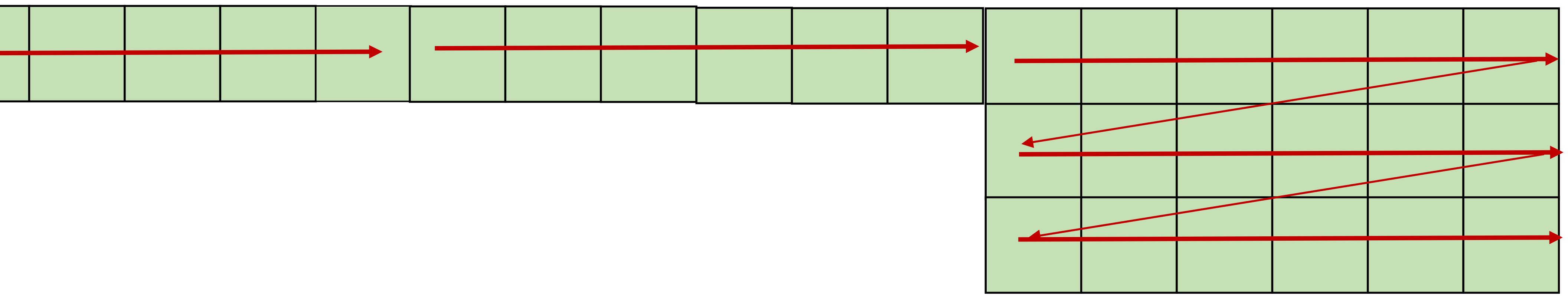
How multi dimensional arrays are stored:

Row major



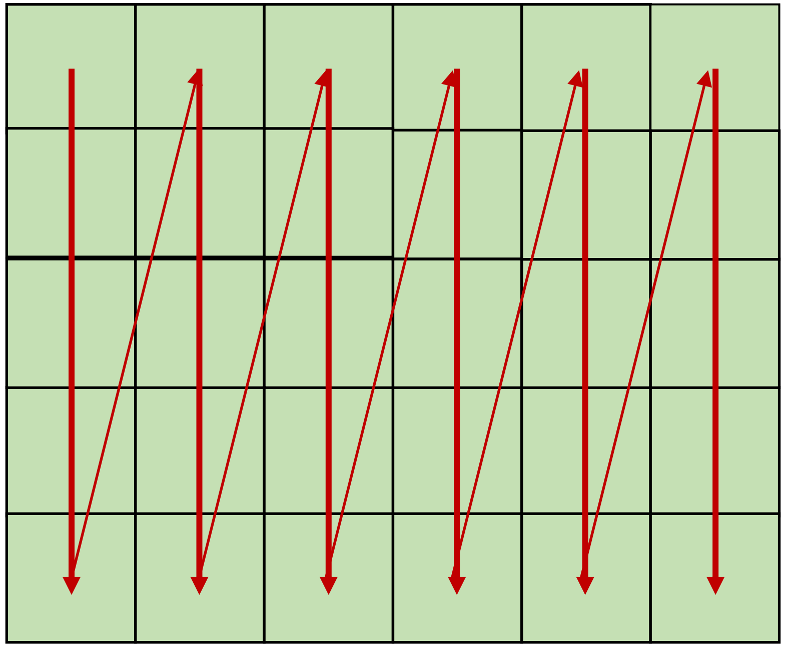
How multi dimensional arrays are stored:

Row major

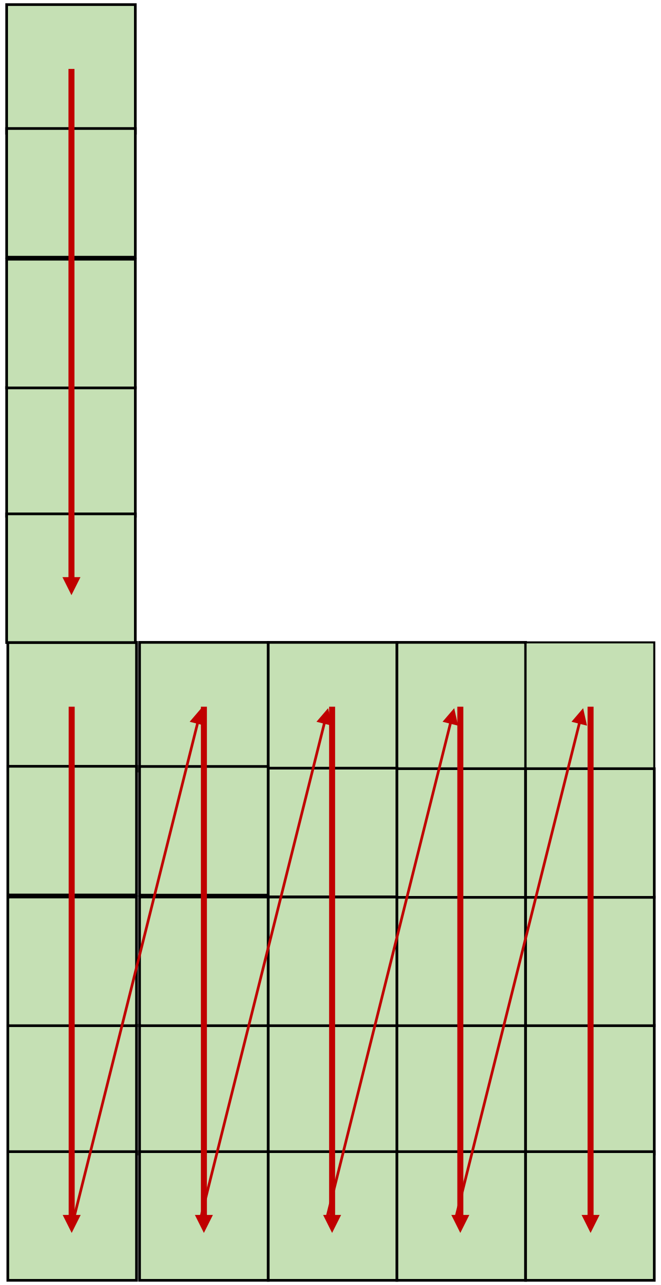


How multi dimensional arrays are stored:

Column major?
Fortran
Matlab
R



How multi dimensional arrays are stored:

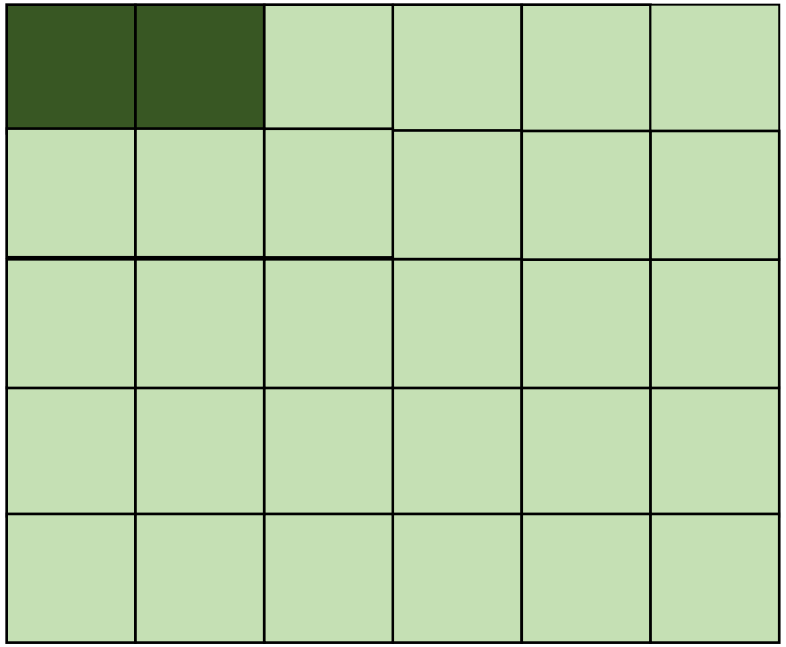


Column major?
Fortran
Matlab
R

How multi dimensional arrays are stored:

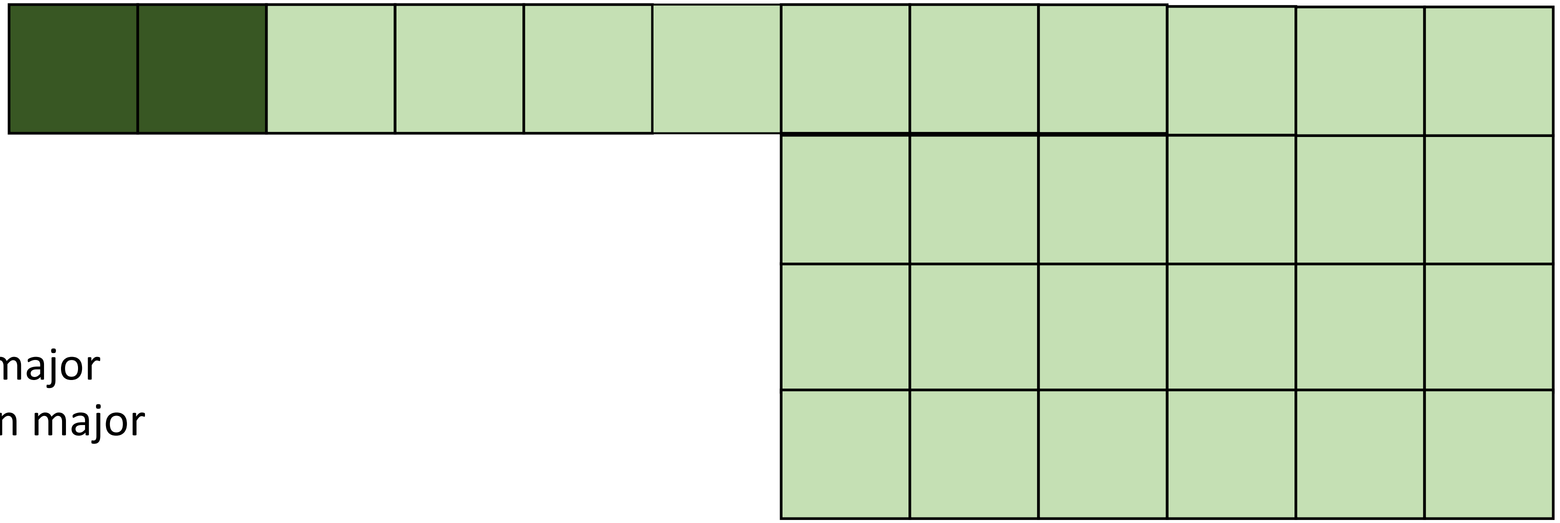
```
x1 = a[0,0];
x2 = a[0,1];
```

good pattern for row major
bad pattern for column major



How multi dimensional arrays are stored:

unrolled row major: still has locality



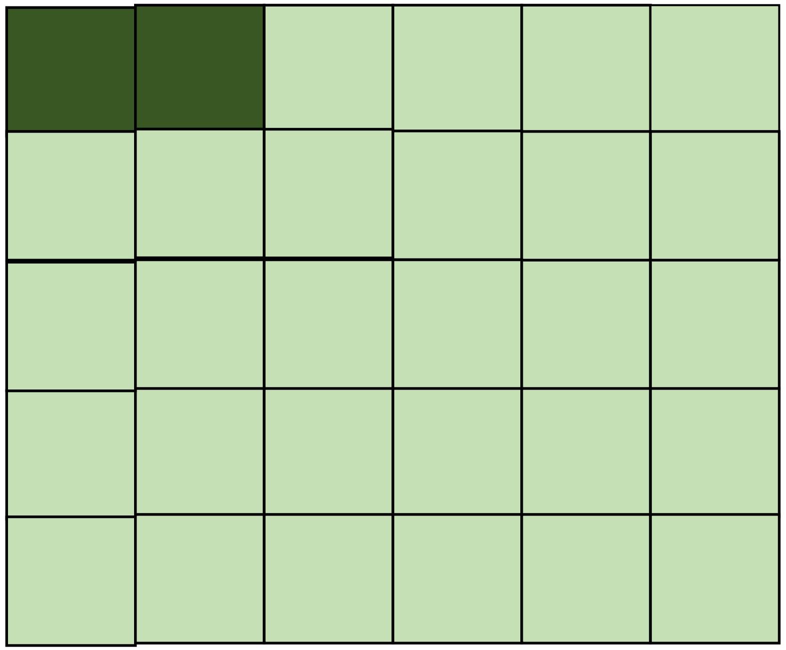
```
x1 = a[x,y];
x2 = a[x, y+1];
```

good pattern for row major
bad pattern for column major

How multi dimensional arrays are stored:

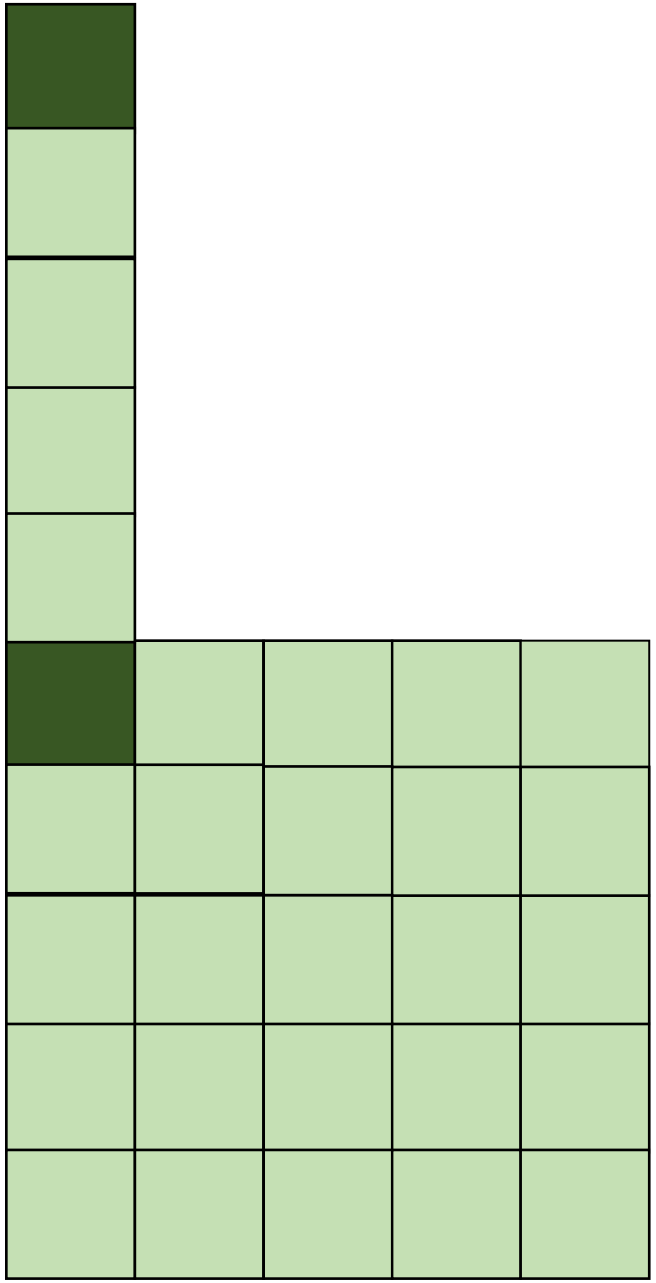
```
x1 = a[x,y];
x2 = a[x, y+1];
```

good pattern for row major
bad pattern for column major



How multi dimensional arrays are stored:

unrolled
column
major:
Bad locality



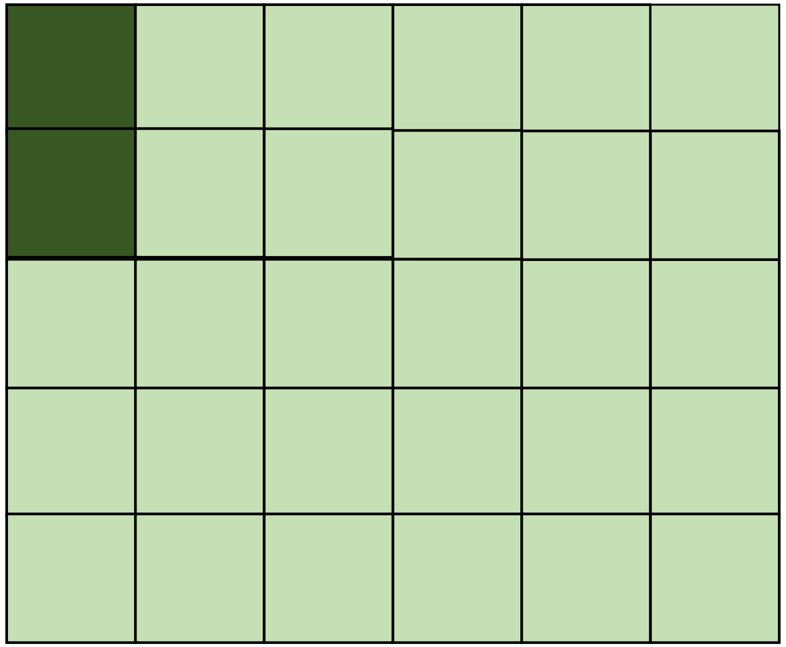
```
x1 = a[x,y];
x2 = a[x, y+1];
```

good pattern for row major
bad pattern for column major

How multi dimensional arrays are stored:

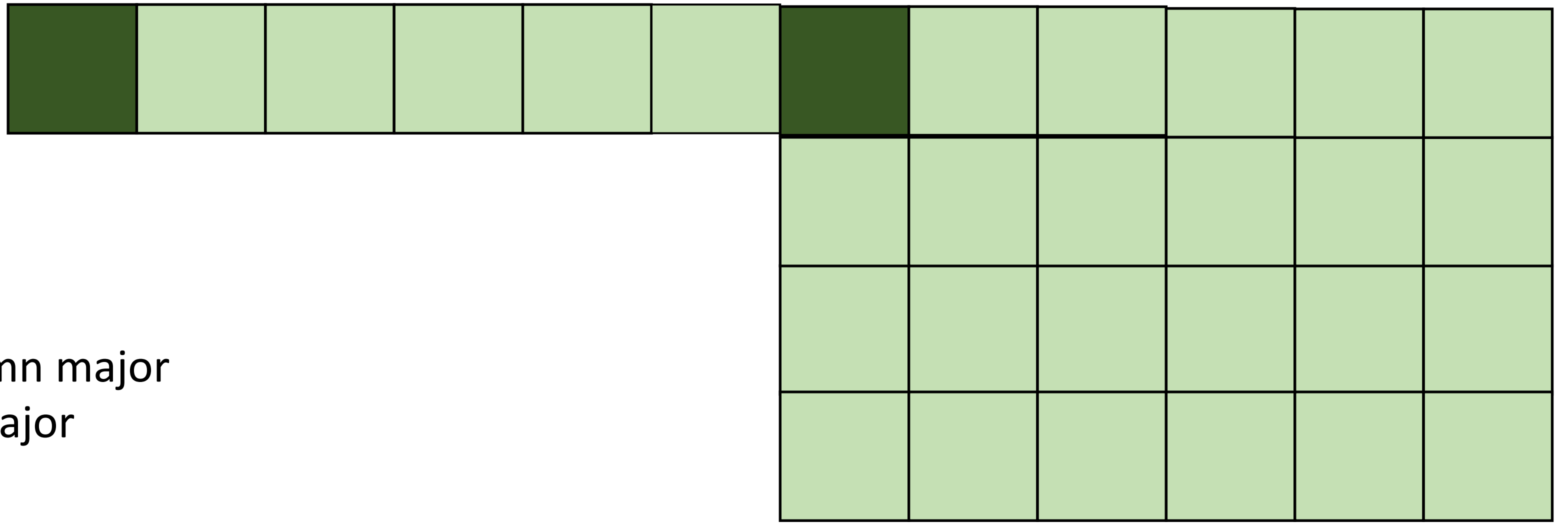
```
x1 = a[0,0];
x2 = a[1, 0];
```

good pattern for column major
bad pattern for row major



How multi dimensional arrays are stored:

row major unrolled: bad spatial locality

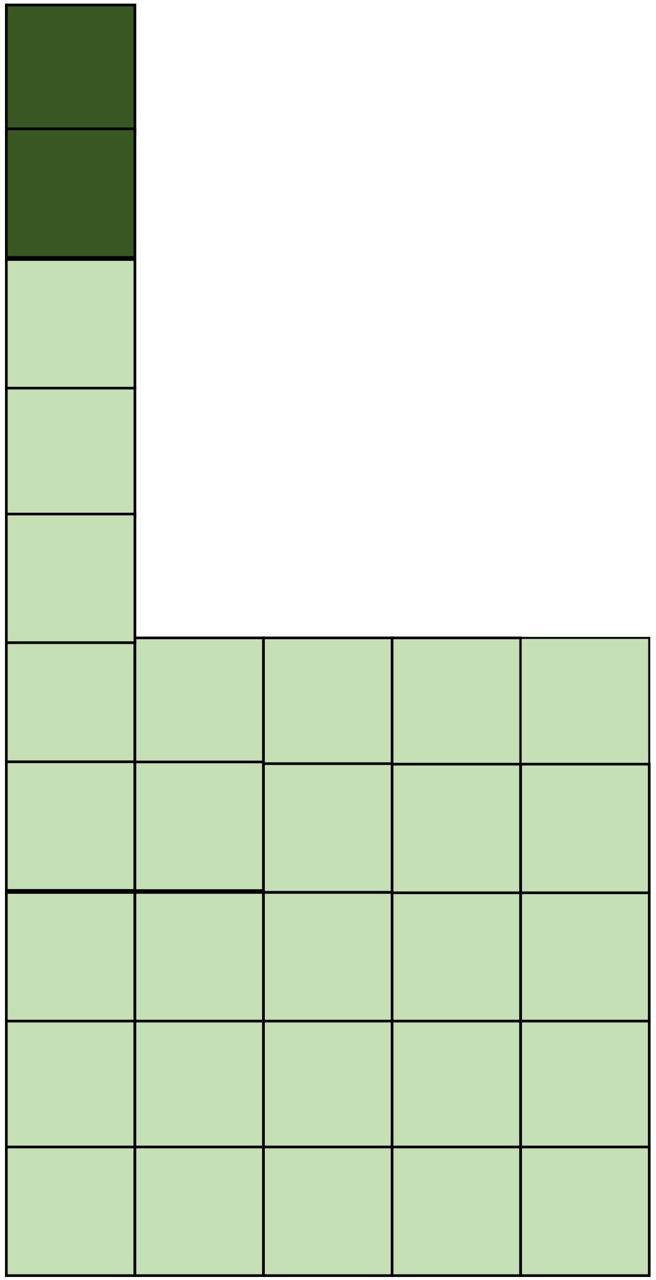


```
x1 = a[x,y];
x2 = a[x+1, y];
```

good pattern for column major
bad pattern for row major

How multi dimensional arrays are stored:

unrolled
column
major:
good locality



```
x1 = a[x,y];
x2 = a[x+1, y];
```

good pattern for column major
bad pattern for row major

How much does this matter?

```
for (int x = 0; x < x_size; x++) {
  for (int y = 0; y < y_size; y++) {
        a[x,y] = b[x,y] + c[x,y];
   }
}
```

```
for (int y = 0; y < y_size; y++) {
  for (int x = 0; x < x_size; x++) {
        a[x,y] = b[x,y] + c[x,y];
   }
}
```

which will be faster?
by how much?

Demo

How to reorder loop nestings?

- For a DOALL loop, if loop bounds are independent, they can simply be re-ordered.

- If they are dependent...

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

bad nesting order for row-major!

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

bad nesting order for row-major!

but iteration variables are dependent

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

bad nesting order for row-major!

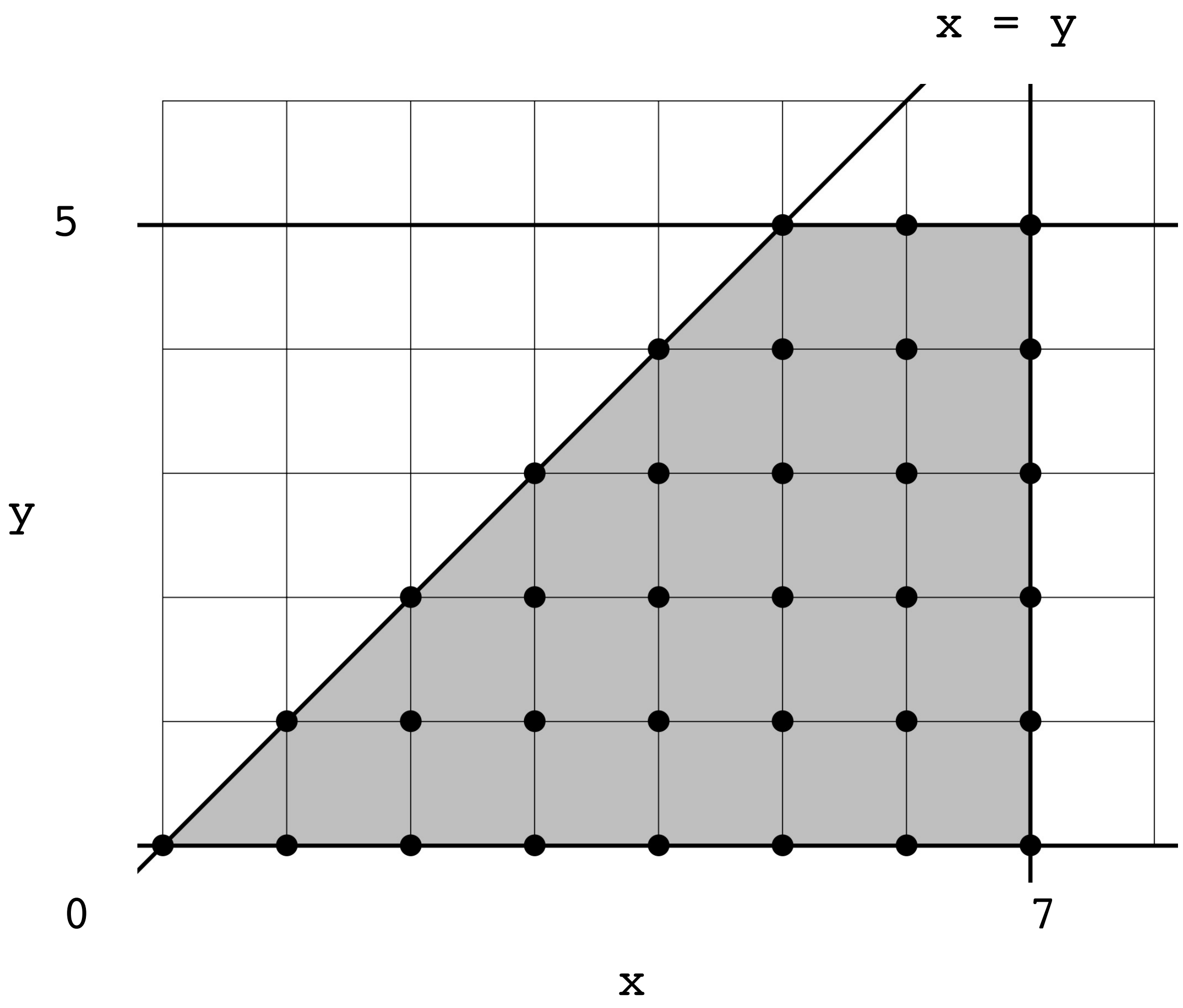
but iteration variables are dependent

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

Example:

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

System with N variables can be viewed as an N dimensional polyhedron



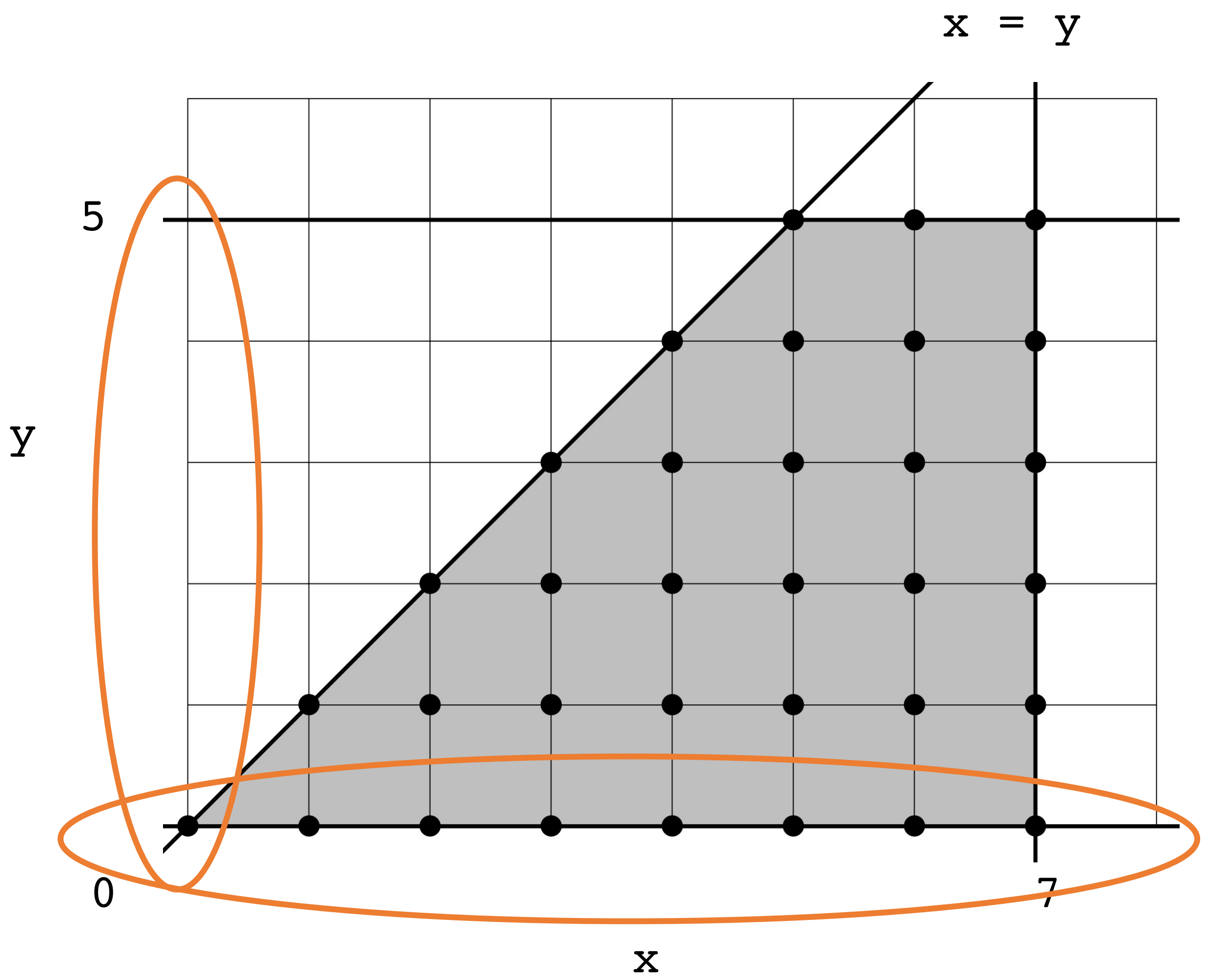
Fourier-Motzkin elimination:

- Given a system of inequalities with N variables, reduce it to a system with N - 1 variables.
- A system of inequalities describes an N-dimensional polyhedron. Produce a system of equations that projects the polyhedron onto an N-1 dimensional space

Example:

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

System with N variables can be viewed as an N dimensional polyhedron



Fourier-Motzkin elimination:

- To eliminate variable x_i:
  For every pair of lower bound L_i and upper bound U_i on x_i, create:

  $$L_i \leq x_i \leq U_i$$

  Then simply remove x_i :

  $$L_i \leq U_i$$

Example: remove y from the constraints

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

All pairs of upper/lower bounds on y:

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

Example: remove y from the constraints

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

All pairs of upper/lower bounds on y:

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

```
0 <= y <= 5
0 <= y <= x
```

Example: remove y from the constraints

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

All pairs of upper/lower bounds on y:

```
0 <= y <= 5
0 <= y <= x
```

Then eliminate y:

```
0 <= 5
0 <= x
```

Example: remove y from the constraints

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

All pairs of upper/lower bounds on y:

```
0 <= y <= 5
0 <= y <= x
```

Then eliminate y:

```
0 <= 5
0 <= x
```

Example: remove y from the constraints

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

All pairs of upper/lower bounds on y:

```
0 <= y <= 5
0 <= y <= x
```

Then eliminate y:

```
0 <= x
```

Example: remove y from the constraints

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

All pairs of upper/lower bounds on y:

```
0 <= y <= 5
0 <= y <= x
```

Then eliminate y:

```
0 <= x
```

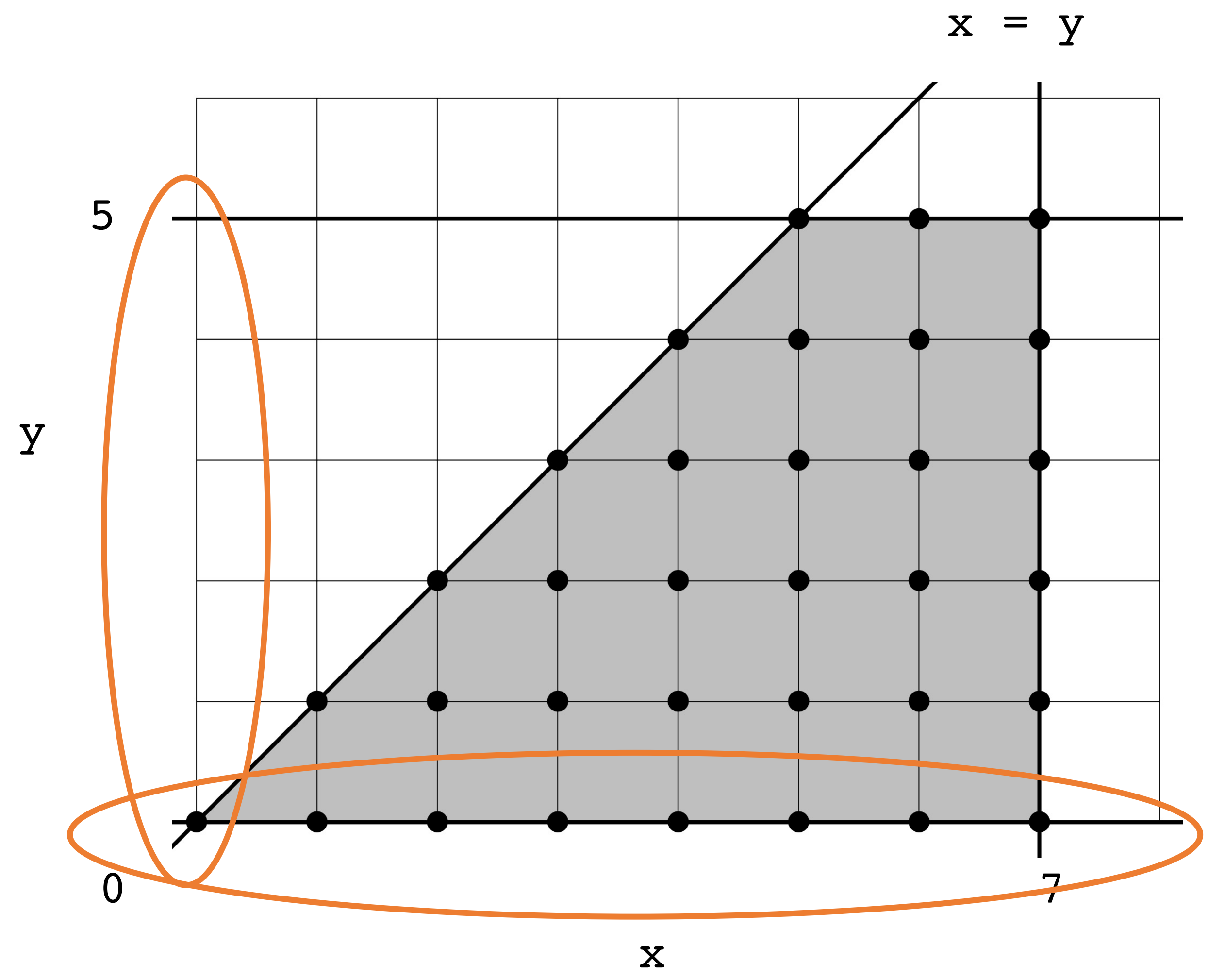
loop constraints without y:

```
x >= 0
x <= 7
```

Example:

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

System with N variables can be viewed as an N dimensional polyhedron



Reording Loop bounds:

- Given a new order: $[x_0, x_1, x_2, \ldots x_n]$
- For each variable x_i : perform Fourier-Motzkin elimination to eliminate any variables that come after x_i in the new order.
- Instantiate loop conditions for x_i, potentially using `max/min` operators

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

new order: [x,y]

for x: eliminate y using FM elimination:

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

new order: [x,y]

for x: eliminate y using FM elimination:

x loop constraints without y:

```
x >= 0
x <= 7
```

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

new order: [x,y]

for x: eliminate y using FM elimination:

x loop constraints without y:

```
x >= 0
x <= 7
```

```
y loop constraints:
y >= 0
y <= 5
y <= x
```

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

new order: [x,y]

for x: eliminate y using FM elimination:

x loop constraints without y:

```
x >= 0
x <= 7
```

```
y loop constraints:
y >= 0
y <= 5
y <= x
```

Example:

```
for (y = 0; y <= 5; y++) {
  for (x = y; x <= 7; x++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```

```
loop constraints
y >= 0
y <= 5
x >= y
x <= 7
```

new order: [x,y]

for x: eliminate y using FM elimination:

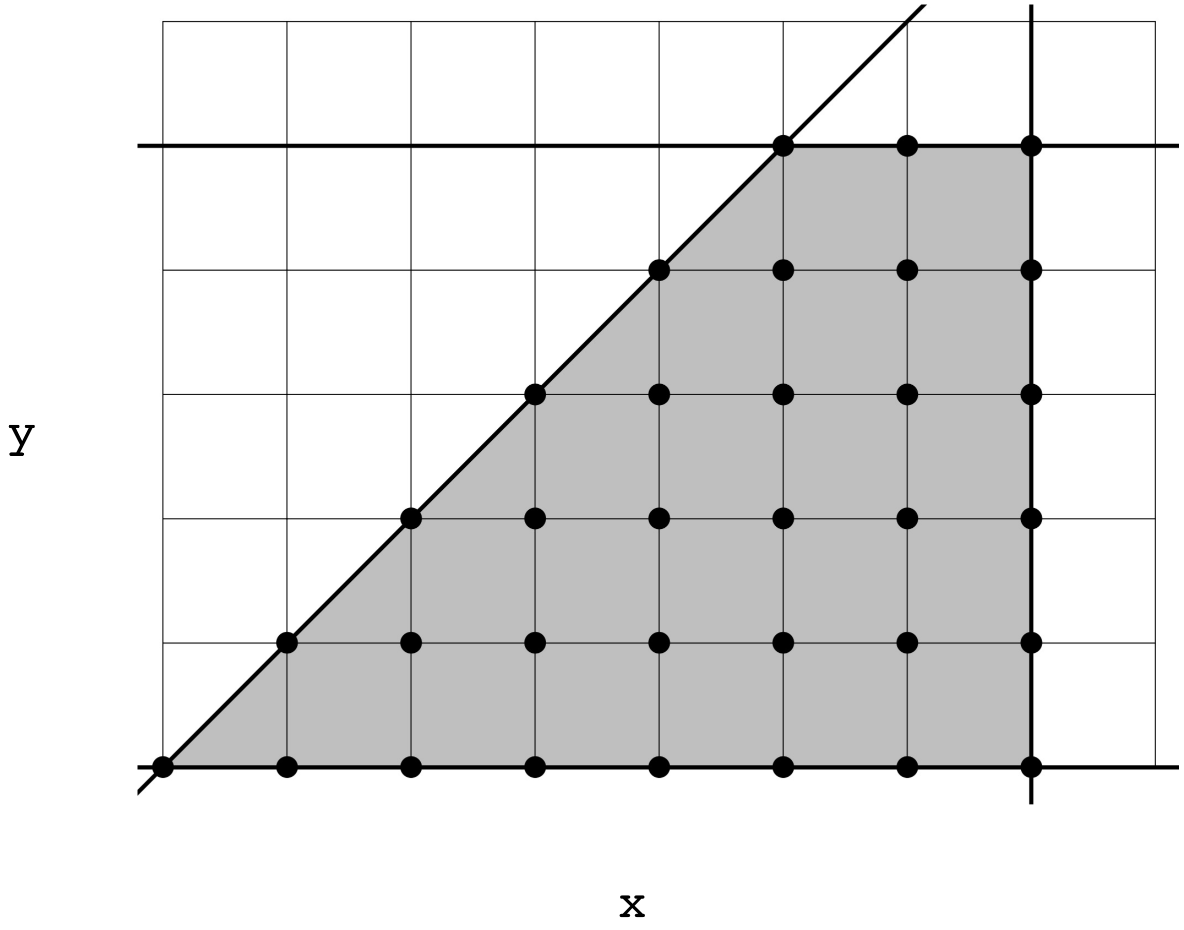
x loop constraints without y:

```
x >= 0
x <= 7
```

```
y loop constraints:
y >= 0
y <= min(x,5)
```

Example:

```
for (x = 0; x <= 7; x++) {
  for (y = 0; y <= min(x,5); y++) {
    a[x,y] = b[x,y] + c[x,y];
  }
}
```



x loop constraints without y:

```
x >= 0
x <= 7
```

```
y loop constraints:
y >= 0
y <= min(x,5)
```

Reordering loop bounds

- only works if loop increments by 1; assumes a closed polyhedron

- best performance when array indexes are simple:
  - e.g.: `a[x,y]`
  - harder with, e.g.: `a[x*5+127, y+x*37]`
  - There exists schemes to automatically detect locality. Reach chapter 10 of the Dragon book

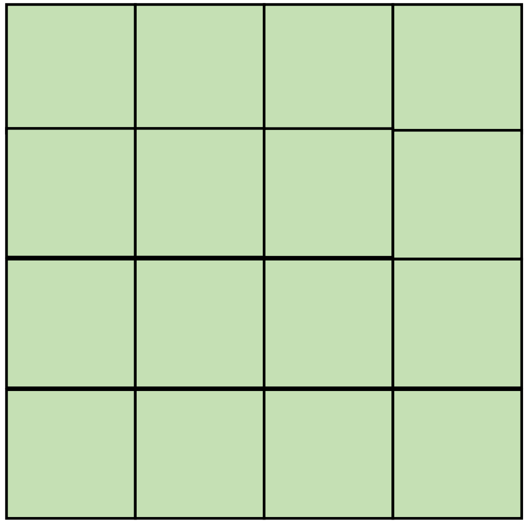
- compiler implementation allows exploration and auto-tuning

Adding loop nestings

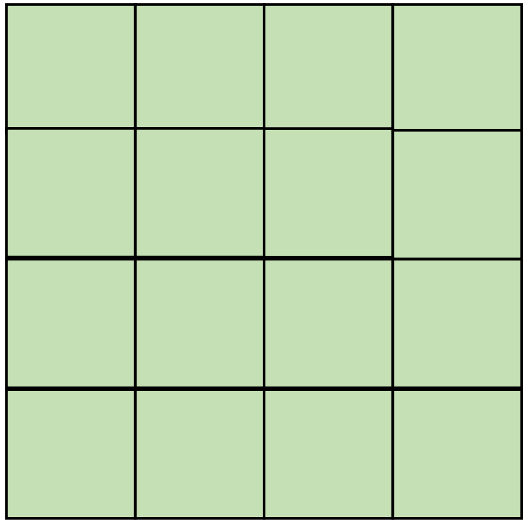
- In some cases, there might not be a good nesting order for all accesses:

$$A = B + C^T$$

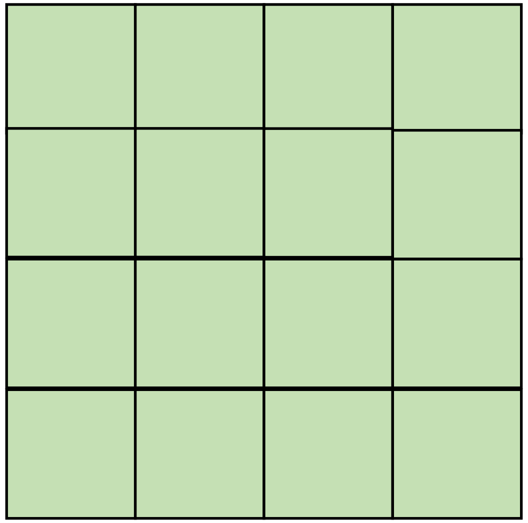
A



B



C

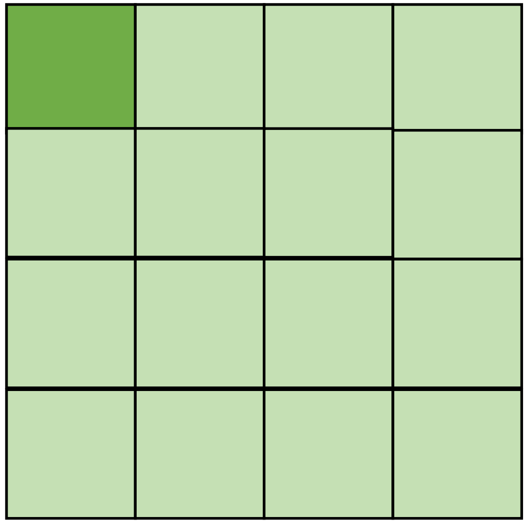


Adding loop nestings

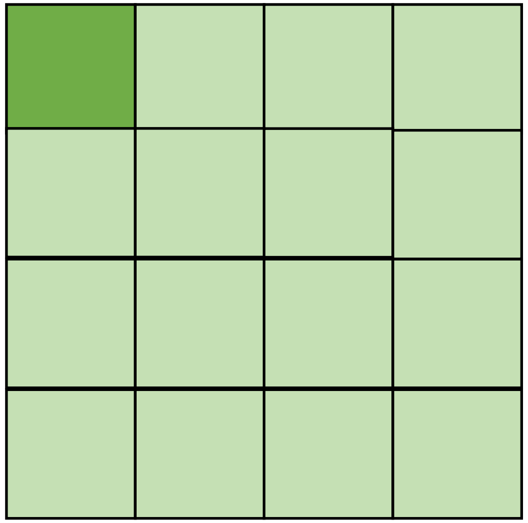
- In some cases, there might not be a good nesting order for all accesses:

$$A = B + C^T$$

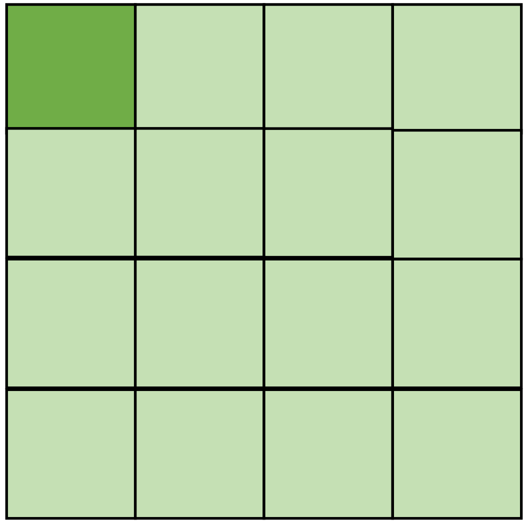
A



B



C

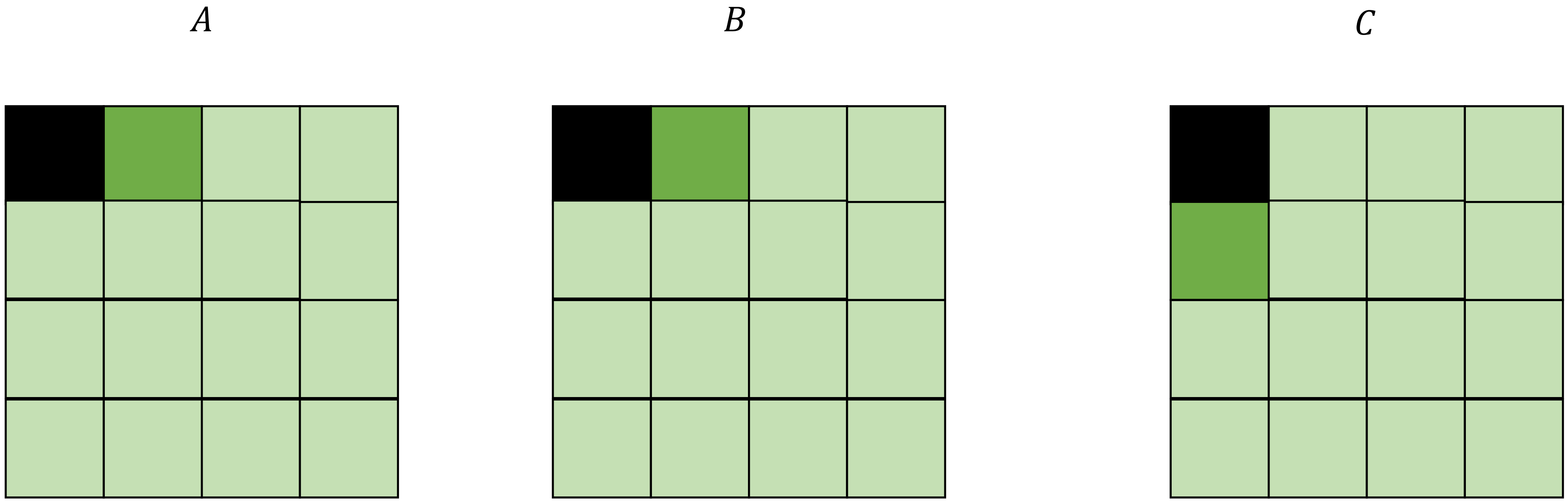


cold miss for all of them

Adding loop nestings

- In some cases, there might not be a good nesting order for all accesses:

$$A = B + C^T$$



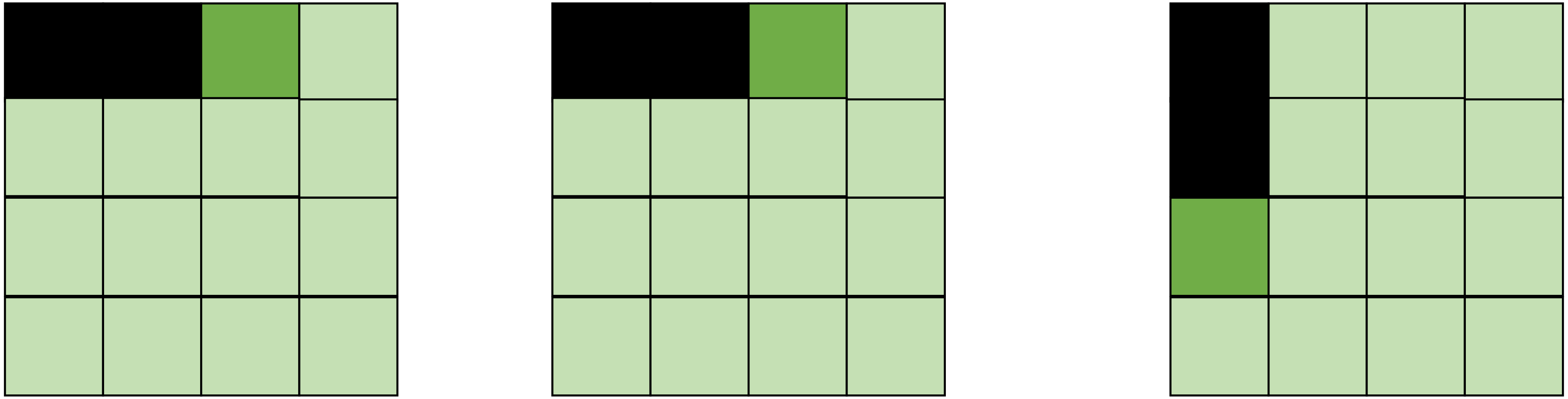
Hit on A and B. Miss on C

Adding loop nestings

- In some cases, there might not be a good nesting order for all accesses:

$$A = B + C^T$$

A B C



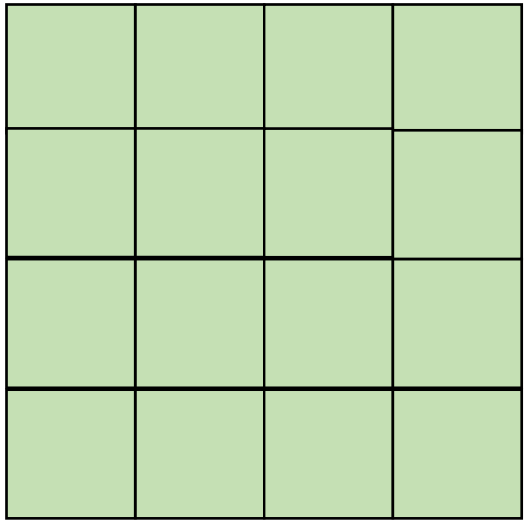
Hit on A and B. Miss on C

Adding loop nestings

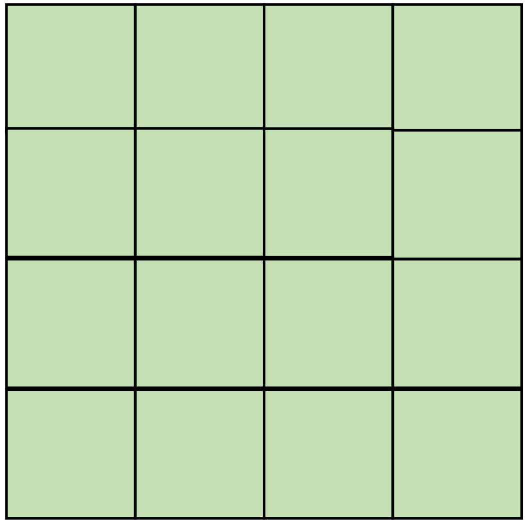
- Blocking operates on smaller chunks to exploit locality in column increment accesses. Example 2x2

$$A = B + C^T$$

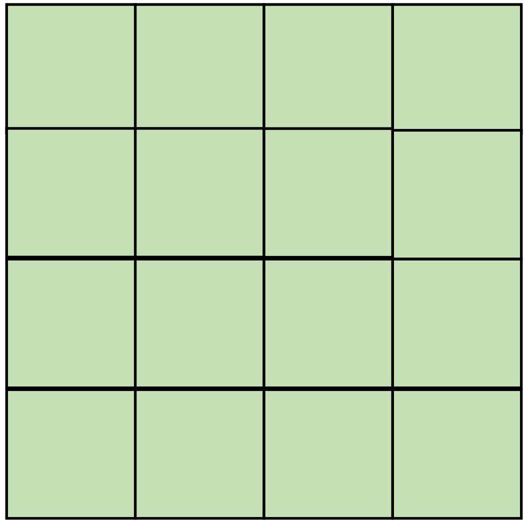
A



B



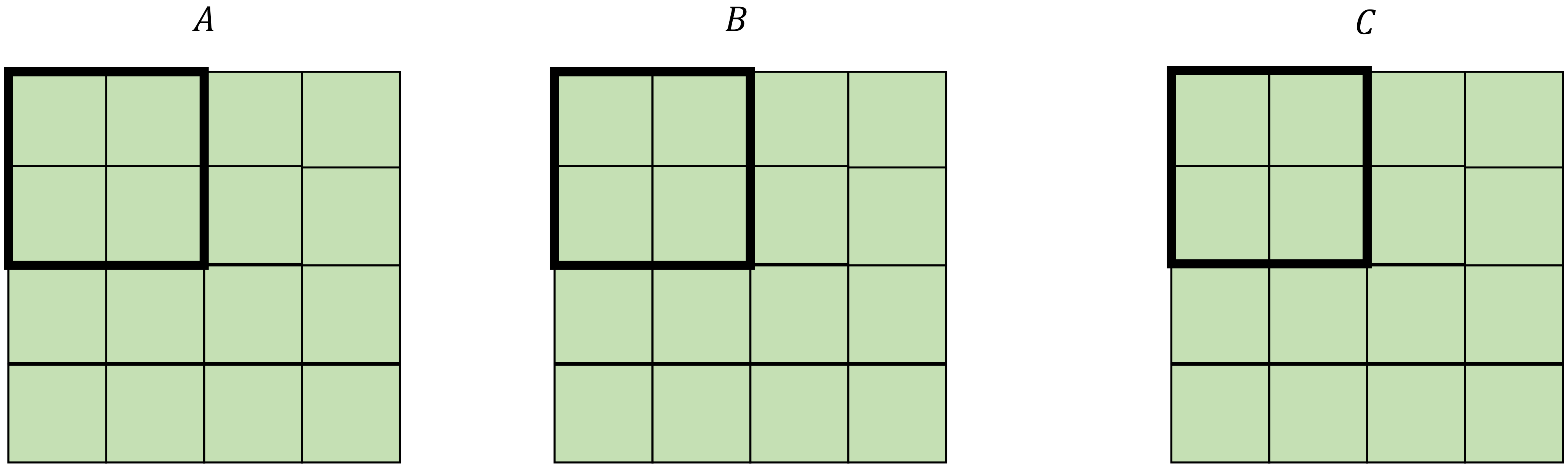
C



Adding loop nestings

- Blocking operates on smaller chunks to exploit locality in column increment accesses. Example 2x2

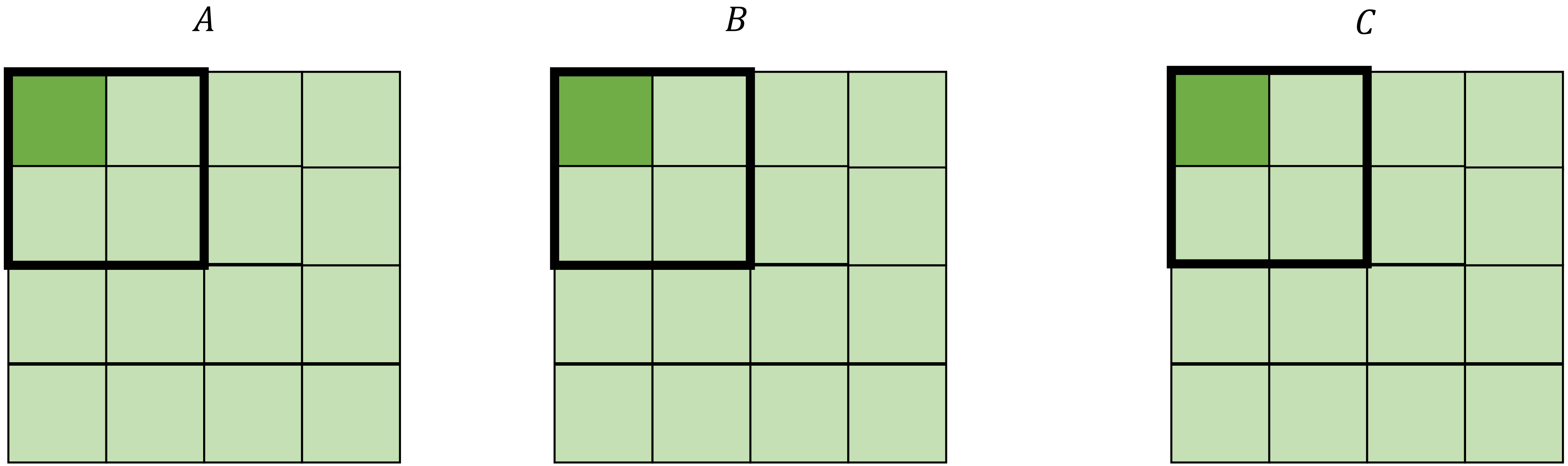
$$A = B + C^T$$



Adding loop nestings

- Blocking operates on smaller chunks to exploit locality in column increment accesses. Example 2x2

$$A = B + C^T$$

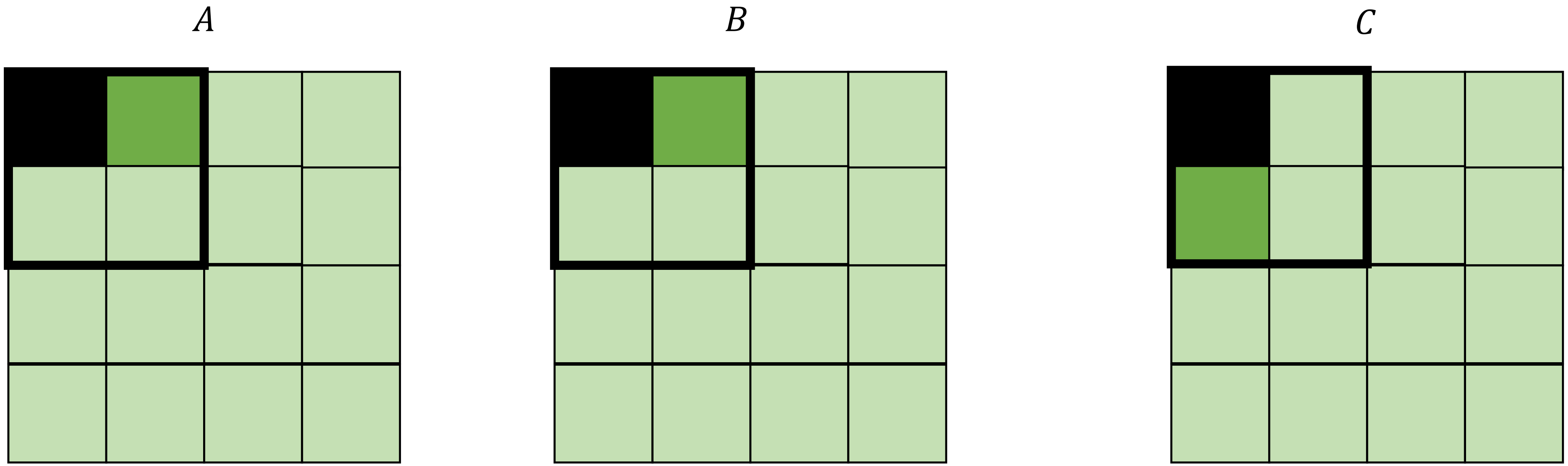


cold miss for all of them

Adding loop nestings

- Blocking operates on smaller chunks to exploit locality in column increment accesses. Example 2x2

$$A = B + C^T$$

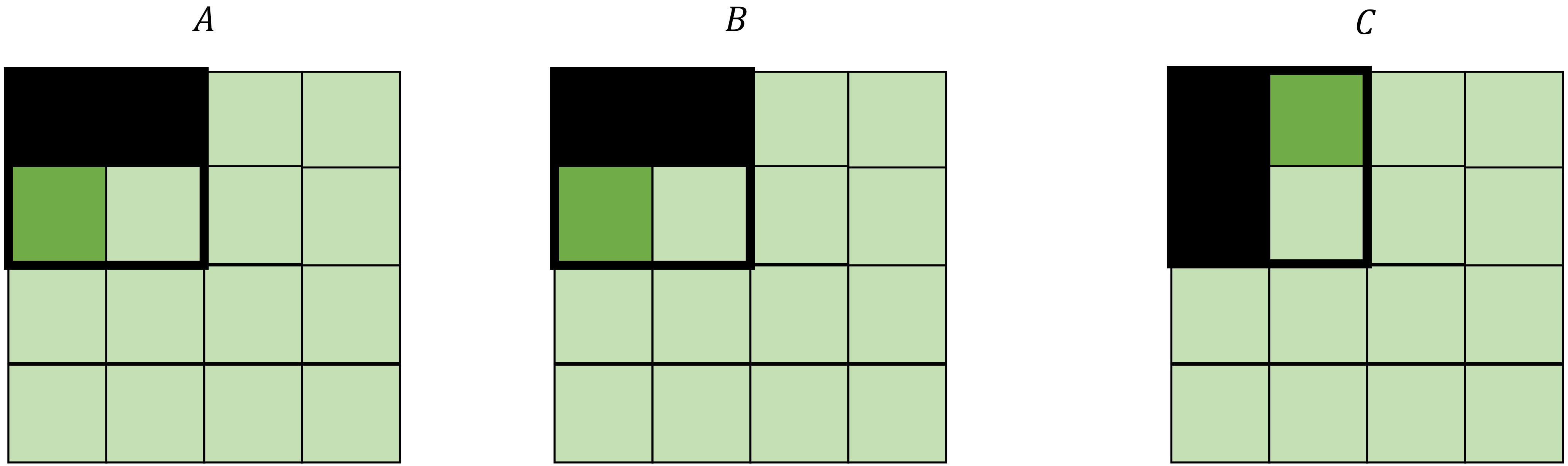


Miss on C

Adding loop nestings

- Blocking operates on smaller chunks to exploit locality in column increment accesses. Example 2x2

$$A = B + C^T$$



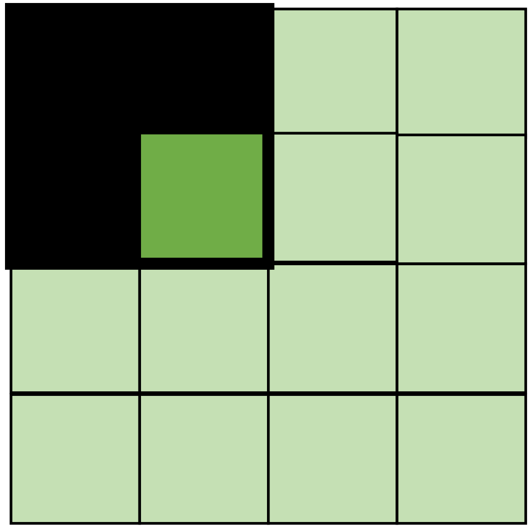
Miss on A,B, hit on C

Adding loop nestings

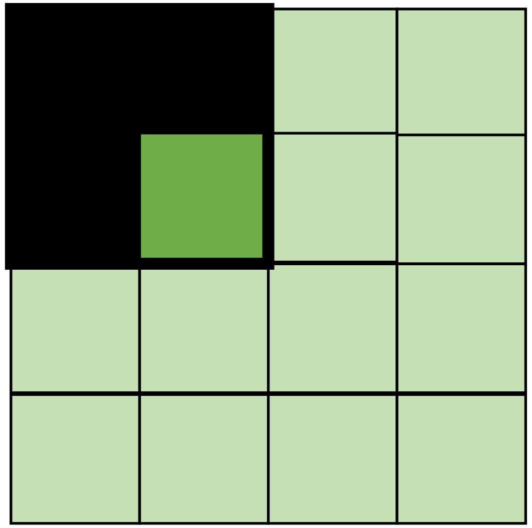
- Blocking operates on smaller chunks to exploit locality in column increment accesses. Example 2x2

$$A = B + C^T$$

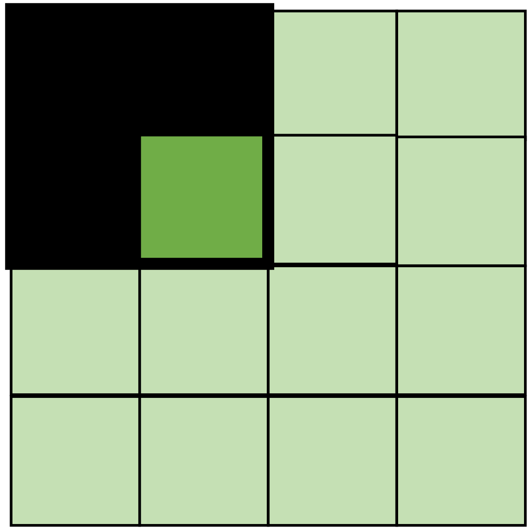
A



B



C



Hit on all!

Adding loop nestings

- Add two outer loops for both x and y

```
for (int x = 0; x < SIZE; x++) {
    for (int y = 0; y < SIZE; y++) {
      a[x*SIZE + y] = b[x*SIZE + y] + c[y*SIZE + x];
    }
  }
```

Adding loop nestings

- Add two outer loops for both x and y

```
for (int xx = 0; xx < SIZE; xx += B) {
  for (int yy = 0; yy < SIZE; yy += B) {
    for (int x = xx; x < xx+B; x++) {
      for (int y = yy; y < yy+B; y++) {
        a[x*SIZE + y] = b[x*SIZE + y] + c[y*SIZE + x];
      }
    }
  }
}
```

Adding loop nestings

- Add two outer loops for both x and y

```
for (int xx = 0; xx < SIZE; xx += B) {
  for (int yy = 0; yy < SIZE; yy += B) {
    for (int x = xx; x < xx+B; x++) {
      for (int y = yy; y < yy+B; y++) {
        a[x*SIZE + y] = b[x*SIZE + y] + c[y*SIZE + x];
      }
    }
  }
}
```

Adding loop nestings

- Add two outer loops for both x and y

```
for (int xx = 0; xx < SIZE; xx += B) {
  for (int yy = 0; yy < SIZE; yy += B) {
    for (int x = xx; x < xx+B; x++) {
      for (int y = yy; y < yy+B; y++) {
        a[x*SIZE + y] = b[x*SIZE + y] + c[y*SIZE + x];
      }
    }
  }
}
```

Adding loop nestings

- Add two outer loops for both x and y

```
for (int xx = 0; xx < SIZE; xx += B) {
  for (int yy = 0; yy < SIZE; yy += B) {
    for (int x = xx; x < xx+B; x++) {
      for (int y = yy; y < yy+B; y++) {
        a[x*SIZE + y] = b[x*SIZE + y] + c[y*SIZE + x];
      }
    }
  }
}
```

Demo

Next class

- Topics:
  - Implementing parallelism for DOALL loops
- Enjoy your weekend