

Loop Optimization – 2

Locality

15-411/15-611 Compiler Design

Seth Copen Goldstein

April 1, 2025

Today

- Review
 - Loop Transformation Theory
 - Unimodular Transformations
 - A Data Locality Algorithm
 - Reuse \rightarrow (Localized Vector Space) \rightarrow Locality
 - Reuse
 - Self-Temporal
 - Self-Spatial
 - Group Spatial
- ⌘SRP Algorithm

Review

- Loop Transformation Theory
 - Iteration spaces
 - Dependence information
 - Dependence Analysis
- Transformations
 - interchange
 - reversal
 - skewing
 - Tiling

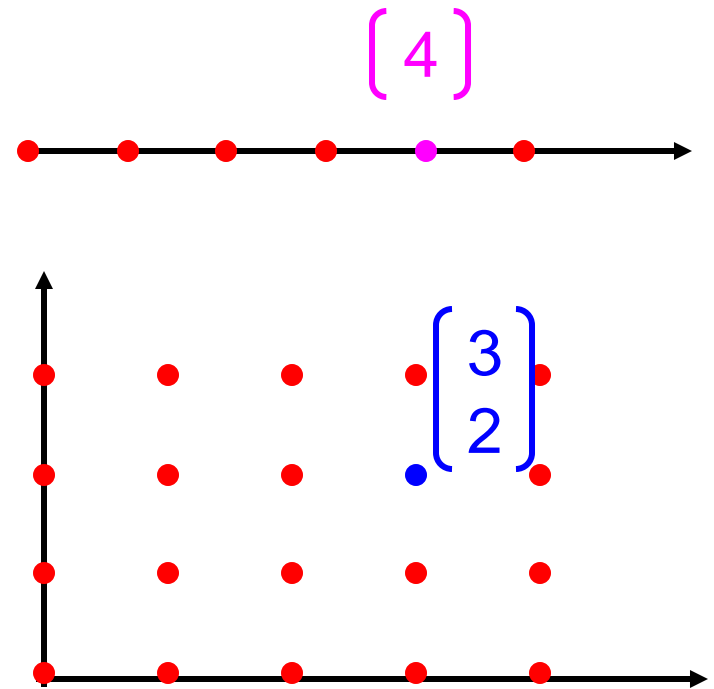
Loop Transformation Theory

- Iteration Space
- Dependence vectors
- Unimodular transformations

Iteration Space

Every iteration generates a point in an n-dimensional space, where n is the depth of the loop nest.

```
for (i=0; i<n; i++) {  
    ...  
}  
  
for (i=0; i<n; i++)  
    for (j=0; j<4; j++) {  
        ...  
    }
```



Loop Nests and the Iter space

- General form of tightly nested loop

```
for I1 := low1 to high1 by step1
  for I2 := low2 to high2 by step2
    ...
    for Ii := lowi to highi by stepi
      ...
      for In := lown to highn by stepn
        Stmts
```

- The iteration space is a convex polyhedron in \mathbb{Z}^n bounded by the loop bounds.
- Each iteration is a node in the polyhedron identified by its vector: $\mathbf{p}=(p_1, p_2, \dots, p_n)$

Lexicographic Order

Consider the vectors **a** and **b** below :

$$\mathbf{a} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

We say that **a** is lexicographically less than **b** at level 3, $\mathbf{a} \prec_3 \mathbf{b}$, or simply that $\mathbf{a} \prec \mathbf{b}$.

Both **a** and **b** are lexicographically positive because $\mathbf{0} \prec \mathbf{a}$, and $\mathbf{0} \prec \mathbf{b}$.

Data Dependences

Loop carried: between two statements instances in two different iterations of a loop.

Loop independent: between two statements instances in the same loop iteration.

Lexicographically forward: the source comes before the target.

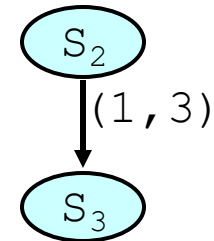
Lexicographically backward: otherwise.

The right-hand side of an assignment is considered to precede the left-hand side.

Data Dependence in Loops

```

(S1)   for i = 2 to 9 do
(S2)     X[i] = Y[i] + Z[i]
(S3)     A[i] = X[i-1] + 1
(S4)   end for
  
```



Data dependence graph for statements in a loop
 (1,3) := iteration distance is 1, latency is 3.

	i = 2	i = 3	i = 4
(s ₂)	X[2] = Y[2] + Z[2]	X[3] = Y[3] + Z[3]	X[4] = Y[4] + Z[4]
(s ₃)	A[2] = X[1] + 1	A[3] = X[2] + 1	A[4] = X[3] + 1

There is a *loop-carried, lexically forward, flow dependence* from S_2 to S_3 .

Dependence Vectors

- Dependence vector in an n-nested loop is denoted as a vector: $\mathbf{d}=(d_1, d_2, \dots, d_n)$.
- Each d_i is a possibly infinite range of ints in $[d_i^{\min}, d_i^{\max}]$ where

$$d_i^{\min} \in \mathbb{Z} \cup \{-\infty\}, d_i^{\max} \in \mathbb{Z} \cup \{\infty\} \text{ and } d_i^{\min} \leq d_i^{\max}$$
- A single dep vector represents a set of distance vectors.
- A distance vector defines a distance in the iteration space.
- A dependence vector is a distance vector if each d_i is a singleton.

↑
↓

Other defs

- Common ranges in dependence vectors

- $[1, \infty]$ as + or >
- $[-\infty, -1]$ as – or <
- $[-\infty, \infty]$ as \pm or *

Handwritten red notes showing symbols for the ranges:

Top row: +, -, \emptyset , *

Bottom row: >, <, \emptyset , *

- A distance vector is the difference between the target and source iterations (for a dependent ref), e.g.,
$$\mathbf{d} = \mathbf{l}_t - \mathbf{l}_s$$

The General Problem

```
DO i1 = L1, U1
  DO i2 = L2, U2
    ...
    DO in = Ln, Un
      S1      A(f1(i1, ..., in), ..., fm(i1, ..., in)) = ...
      S2      ... = A(g1(i1, ..., in), ..., gm(i1, ..., in))
    ENDDO
  ...
ENDDO
ENDDO
```

A dependence exists from S1 to S2 if:

– There exist α and β such that

- $\alpha < \beta$ (control flow requirement)
- $f_i(\alpha) = g_i(\beta)$ for all i , $1 \leq i \leq m$ (common access requirement)

General Solver?

- Looking for an integer solution to:

$$f_i(\alpha) = g_i(\beta) \text{ for all } i, 1 \leq i \leq m$$

- N-deep loop nest
- M subscripts per array reference
- General case, too hard
- Restrict to linear functions of loop-indices
- System of linear equations (2xn variables and m equations)

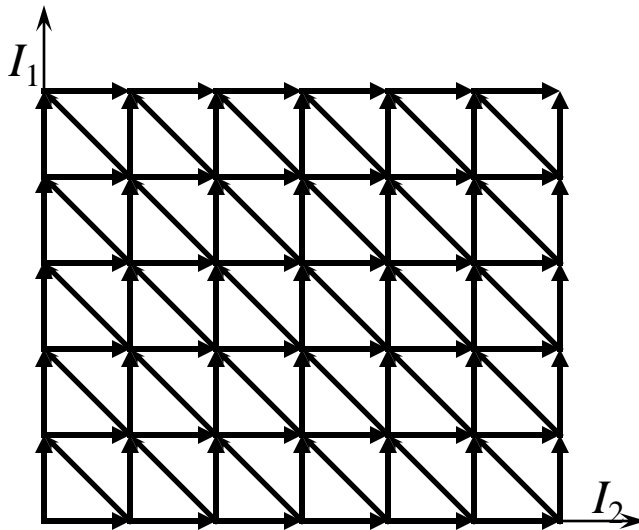
Conservative Testing

- Consider only linear subscript expressions
- For each pair of R/W references:
 - Create subscript pair at each position
 - Perform test based on complexity of pair:
ZIV, SIV, MIV, Coupled, ...
 - Prove independent or get direction/distance
- Merge each pair's information into a single dependence vector

$A[f_1, f_2]$
 $\leftarrow A[f_1, f_2]$
 $\langle f_1, f_2 \rangle$
 $\langle g_1, g_2 \rangle$
 $*$

Examples

```
for  $I_1 := 0$  to 5  
  for  $I_2 := 0$  to 6  
     $A[I_2 + 1] := 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$ 
```



$$D = \{(0,1), (1,0), (1,1)\}$$

Create Subscript Pairs

```
for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] := 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
```

$\langle A[I_2 + 1], A[I_2] \rangle$

$\langle A[I_2 + 1], A[I_2 + 1] \rangle$

$\langle A[I_2 + 1], A[I_2 + 2] \rangle$

read & write
true depd

Do Strong SIV

write after read
anti-dependence

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6

$A[I_2 + 1]$ $:= 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$

	$A[1] \leftarrow$ $\leftarrow A[0]$ $\leftarrow A[1]$ $\leftarrow A[2]$	$A[2] \leftarrow$ $\leftarrow A[1]$ $\leftarrow A[2]$ $\leftarrow A[3]$	$A[3] \leftarrow$ $\leftarrow A[2]$ $\leftarrow A[3]$ $\leftarrow A[4]$
I_1	$A[1] \leftarrow$ $\leftarrow A[0]$ $\leftarrow A[1]$ $\leftarrow A[2]$	$A[2] \leftarrow$ $\leftarrow A[1]$ $\leftarrow A[2]$ $\leftarrow A[3]$	$A[3] \leftarrow$ $\leftarrow A[2]$ $\leftarrow A[3]$ $\leftarrow A[4]$
	$A[1] \leftarrow$ $\leftarrow A[0]$ $\leftarrow A[1]$ $\leftarrow A[2]$	$A[2] \leftarrow$ $\leftarrow A[1]$ $\leftarrow A[2]$ $\leftarrow A[3]$	$A[3] \leftarrow$ $\leftarrow A[2]$ $\leftarrow A[3]$ $\leftarrow A[4]$

I_2

$\langle A[I_2 + 1], A[I_2] \rangle$

$\langle A[I_2 + 1], A[I_2 + 1] \rangle$

$\langle A[I_2 + 1], A[I_2 + 2] \rangle$

\emptyset

Do Strong SIV

```

for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] := 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
  
```

I ₁	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]
	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]
	<u>A[1] ←</u> ← A[0] ← A[1] ← A[2]	A[2] ← <u>← A[1]</u> ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]

<A[I₂ + 1], A[I₂]>

<A[I₂ + 1], A[I₂ + 1]>

<A[I₂ + 1], A[I₂ + 2]>

✓

0

I₂

Do Strong SIV

```

for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] := 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
  
```

I ₁	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]
	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]
	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]

I₂

<A[I₂ + 1], A[I₂]> 1
 <A[I₂ + 1], A[I₂ + 1]> 0
<A[I₂ + 1], A[I₂ + 2]> 1

Merge

```

for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] := 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
  
```

I ₁	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]
	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]
	A[1] ← ← A[0] ← A[1] ← A[2]	A[2] ← ← A[1] ← A[2] ← A[3]	A[3] ← ← A[2] ← A[3] ← A[4]

I₂

$\langle A[I_2 + 1], A[I_2] \rangle \quad 1$
 $\langle A[I_2 + 1], A[I_2 + 1] \rangle \quad 0$
 $\langle A[I_2 + 1], A[I_2 + 2] \rangle \quad -1$

$\begin{bmatrix} * \\ 1 \end{bmatrix} \quad \begin{bmatrix} * \\ 0 \end{bmatrix} \quad \begin{bmatrix} * \\ -1 \end{bmatrix}$

[Handwritten red diagram showing a vertical bracket with an arrow pointing to the middle element, likely representing the merge operation.]

Examples

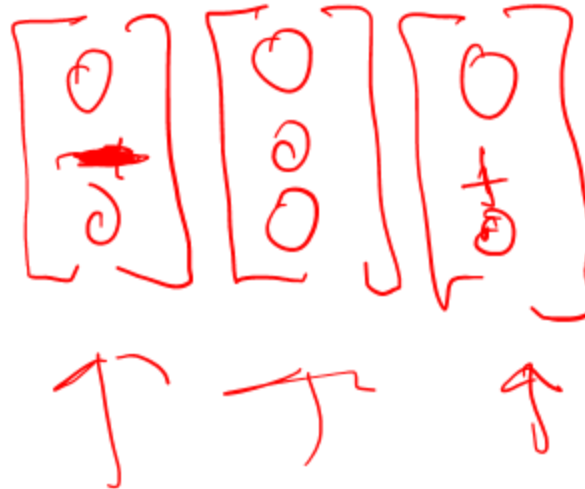
```

for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1, I3] += A[I1, I2] * B[I2, I3]
  
```

$\leftarrow C[I_1, I_3]$
 $C[I_1, I_3] \leftarrow$

$\langle I_1, I_3 \rangle \rightarrow$
 $\langle I_2, I_3 \rangle \rightarrow$

$$\begin{bmatrix} 0 \\ * \\ 0 \end{bmatrix}$$



Examples

for $I_1 := 1$ to n

for $I_2 := 1$ to n

for $I_3 := 1$ to n

$C[I_1, I_3] += A[I_1, I_2] * B[I_2, I_3]$

map [3]

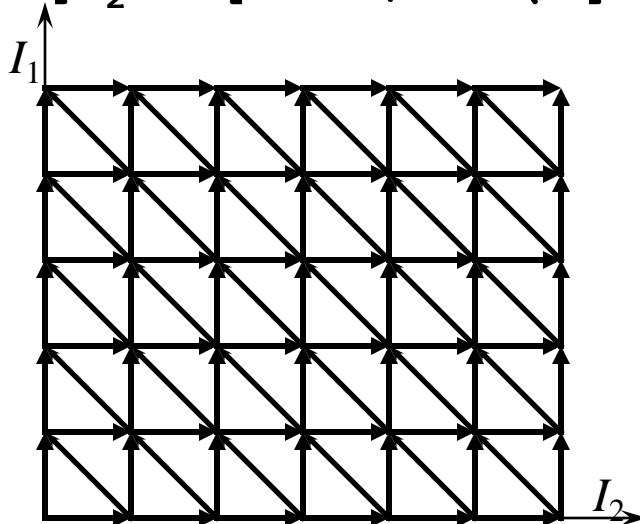
$A[\text{map}[i]]$

$(0, 1, 0)$

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6

$A[I_2 + 1] := 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$



$D = \{(0, 1), (1, 0), (1, -1)\}$

Uniformly Generated references

- f and g are indexing functions: $Z^n \rightarrow Z^d$
 - n is depth of loop nest
 - d is dimensions of array, A
- Two references $A[f(i)]$ and $A[g(i)]$ are uniformly generated if

$$f(i) = Hi + c_f \text{ AND } g(i) = Hi + c_g$$

- H is a linear transform
- c_f and c_g are constant vectors

Eg of Uniformly generated sets

These references all belong to the same
uniformly generated set: $H = [0 \ 1]$

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6

$$A[I_2 + 1] = 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$$

$$A[I_2 + 1] \quad [0 \ 1] \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + [1]$$

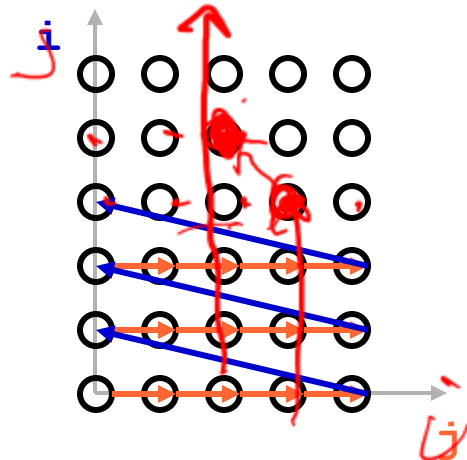
$$A[I_2] \quad [0 \ 1] \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + [0]$$

$$A[I_2 + 2] \quad [0 \ 1] \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + [2]$$

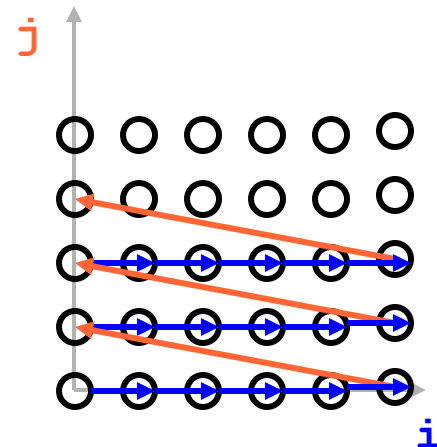
Loop Transforms

- A loop transformation changes the order in which iterations in the iteration space are visited.
- For example, Loop Interchange

```
for i := 0 to n  
  for j := 0 to m  
    body
```



```
for j := 0 to m  
  for i := 0 to n  
    body
```




Unimodular Transforms

- ↗ • Interchange
permute nesting order
- ↶ • Reversal
reverse order of iterations
- ↷ • Skewing
scale iterations by an outer loop index

Interchange

- Change order of loops
- For some permutation p of $1 \dots n$

`for $I_1 := \dots$
 for $I_2 := \dots$
 ...
 for $I_n := \dots$
 body`



`for $I_{p(1)} := \dots$
 for $I_{p(2)} := \dots$
 ...
 for $I_{p(n)} := \dots$
 body`

- When is this legal?



Transform and matrix notation

- If dependences are vectors in iter space, then transforms can be represented as matrix transforms
- E.g., for a 2-deep loop, interchange is:

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_2 \\ p_1 \end{bmatrix}$$

- Since, T is a linear transform, Td is transformed dependence:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_1 \end{bmatrix}$$

Reversal

- Reversal of i^{th} loop reverses its traversal, so it can be represented as:
Diagonal matrix with i^{th} element = -1.
- For 2 deep loop, reversal of outermost is:

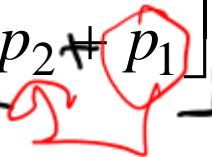
$$T \cong \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \cong \begin{bmatrix} -p_1 \\ p_2 \end{bmatrix}$$

Skewing

- Skew loop I_j by a factor f w.r.t. loop I_i maps

$$(p_1, \dots, p_i, \dots, p_j, \dots) \quad (p_1, \dots, p_i, \dots, p_j + fp_i, \dots)$$

- Example for 2D

$$T \equiv \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \equiv \begin{bmatrix} p_1 \\ p_2 + p_1 \end{bmatrix}$$


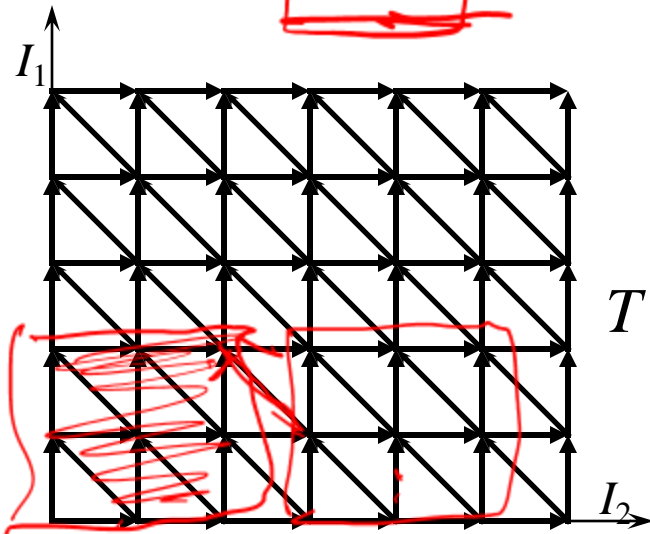
Loop Skewing Example

for $I_1 := 0$ to 5

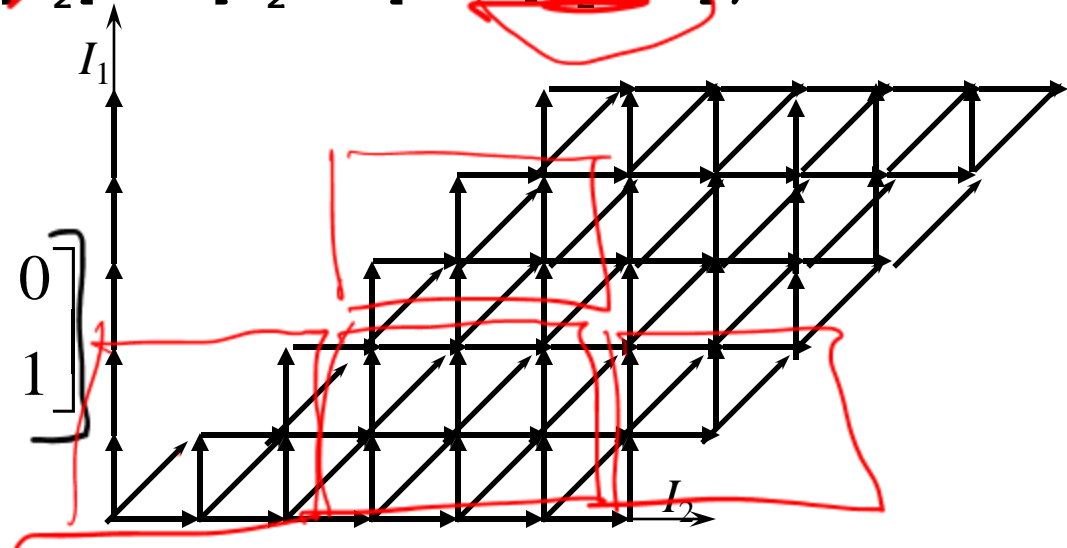
for $I_2 := 0$ to 6

$A[I_2 + 1] := 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$

$D = \{(0,1), (1,0), (1,-1)\}$



$$T \approx \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$



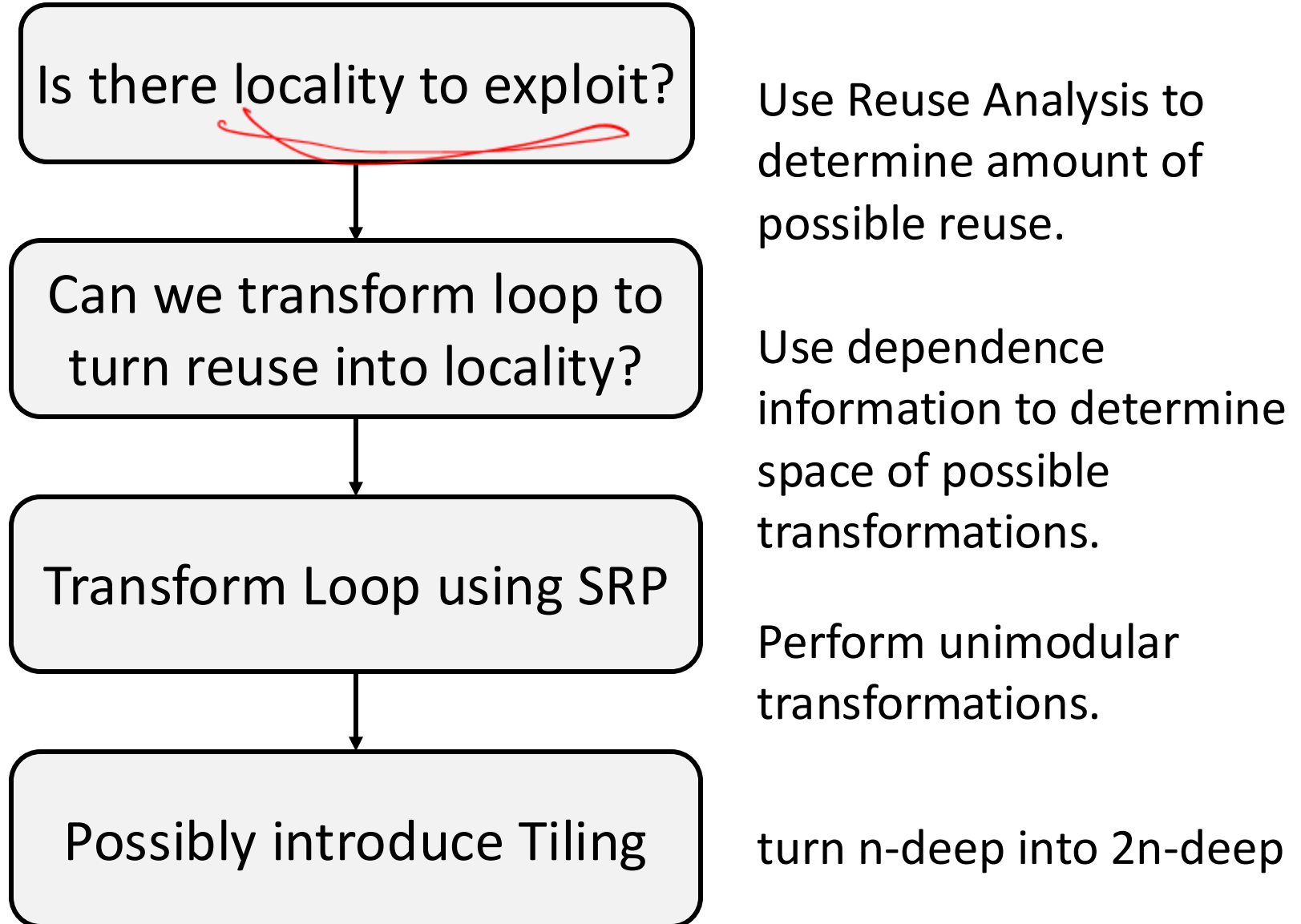
for $I_1 := 0$ to 5

for $I_2 := I_1$ to $6 + I_1$

$A[I_2 - I_1 + 1] := 1/3 * (A[I_2 - I_1] + A[I_2 - I_1 + 1] + A[I_2 - I_1 + 2])$

$D = \{(0,1), (1,1), (1,0)\}$

Our Goal: Increase locality



Predicting Cache Behavior through “Locality Analysis”

- Definitions:

- Reuse:

- accessing a location that has been accessed in the past

- Locality:

- accessing a location that is now found in the cache

- Key Insights

- Locality only occurs when there is reuse!

- BUT, reuse does not necessarily result in locality.

- Why not?

Steps in Locality Analysis

1. Find data reuse

- if caches were infinitely large, we would be finished

2. Determine “localized iteration space”

- set of inner loops where the data accessed by an iteration is expected to fit within the cache

3. Find data locality:

- reuse \supseteq localized iteration space \supseteq locality



Types of Data Reuse

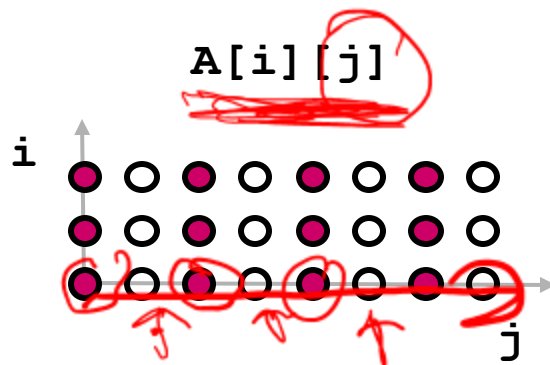
IRIS

*for j
for i
B[j][0]*

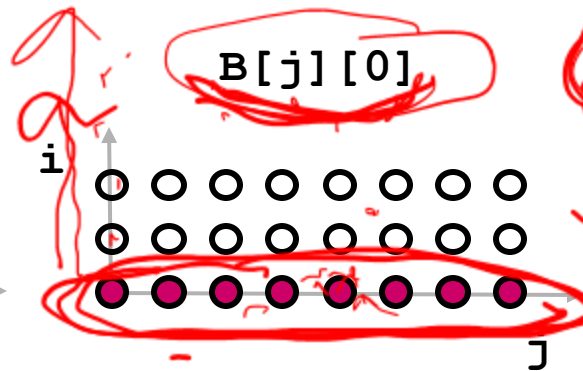
```
for i = 0 to 2
  for j = 0 to 100
    A[i][j] = B[j][0] + B[j+1][0];
```

reused

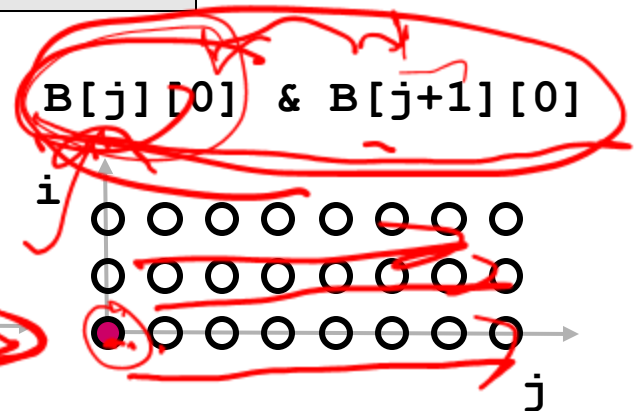
○	Hit
●	Miss



**Self
Spatial**



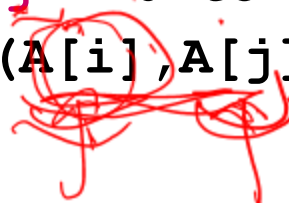
**Self
Temporal**



**Group
(temporal)**

Kinds of reuse and the factor

```
for i = 0 to N-1  
  for j = 0 to N-1  
    f(A[i], A[j]);
```



What kinds of reuse are there?

A[i]?

~~Self-reuse~~ in ~~i~~
Self-spatial in ~~i~~

A[j]?

Self-temporal in ~~i~~
Self-spatial in ~~j~~

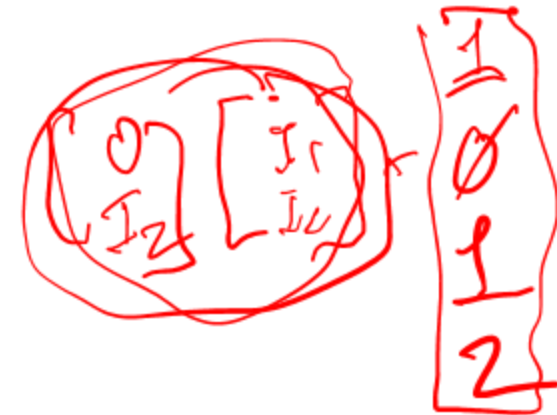


Kinds of reuse and the factor

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6

$$A[I_2+1] = 1/3 * (A[I_2] + A[I_2+1] + A[I_2+2])$$



Self-referential $A[I_2+1]$ in I_2 loop

Self-spatial $A[I_2+1]$ in I_2 loop

$A[I_2]$

$A[I_1+1]$

$A[I_2+2]$

Kinds of reuse and the factor

for $l_1 := 0$ to 5

for $l_2 := 0$ to 6

$A[l_2 + 1] = 1/3 * (A[l_2] + A[l_2 + 1] + A[l_2 + 2])$

self-temporal in 1, self-spatial in 2

Also, group in 1 and 2

What is different about this and previous?

for $i = 0$ to $N-1$

for $j = 0$ to $N-1$

$f(A[i], A[j]);$



Uniformly Generated references

- f and g are indexing functions: $\mathbb{Z}^n \rightarrow \mathbb{Z}^d$
 - n is depth of loop nest
 - d is dimensions of array, A
- Two references $A[f(i)]$ and $A[g(i)]$ are uniformly generated if

$$f(i) = Hi + c_f \text{ AND } g(i) = Hi + c_g$$

- H is a linear transform
- c_f and c_g are constant vectors

Eg of Uniformly generated sets

These references all belong to the same
uniformly generated set: $H = [0 \ 1]$

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6

$$A[I_2 + 1] = 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$$

$$A[I_2 + 1] \quad [0 \ 1] \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + [1]$$

$$A[I_2] \quad [0 \ 1] \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + [0]$$

$$A[I_2 + 2] \quad [0 \ 1] \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + [2]$$

Quantifying Reuse

- Why should we quantify reuse?
- How do we quantify locality?

Quantifying Reuse

- Why should we quantify reuse?
- How do we quantify locality?
- Use vector spaces to identify loops with reuse
- We convert that reuse into locality by making the “best” loop the inner loop
- Metric: memory accesses/iter of innermost loop.
No locality → mem access

Self-Temporal

- For a reference, $A[H\mathbf{i}+\mathbf{c}]$, there is self-temporal reuse between \mathbf{m} and \mathbf{n} when $H\mathbf{m}+\mathbf{c}=H\mathbf{n}+\mathbf{c}$, i.e., $H(\mathbf{r})=\mathbf{0}$, where $\mathbf{r}=\mathbf{m}-\mathbf{n}$.
- The direction of reuse is \mathbf{r} .
- The self-temporal reuse vector space is: $R_{ST} = \text{Ker } H$.
- There is locality if R_{ST} is in the localized vector space.

Recall that for $n \times m$ matrix A ,
the $\text{ker } A = \text{nullspace}(A) = \{\mathbf{x}^m \mid A\mathbf{x} = \mathbf{0}\}$

Example of self-temporal reuse

```

for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1, I3] += A[I1, I2] * B[I2, I3]
  
```



Access	H	ker H	reuse?	Local?
C[I ₁ , I ₃]	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\text{span}\{(0, 1, 0)\}$	n in I ₂	
A[I ₁ , I ₂]	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	$\text{span}\{(0, 0, 1)\}$		
B[I ₂ , I ₃]	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\text{span}\{(1, 0, 0)\}$		

Self Temporal Reuse & Locality

- Reuse is $s^{\dim(R_{ST})}$
- $R_{ST} \cap L = \text{locality}$
- # of mem refs = $\frac{1}{s^{\dim(R_{ST} \cap L)}}$

Self-Spatial

- Occurs when we access in order
 - $A[i,j]$: best gain, l
 - $A[i,j*k]$: best gain, l/k if $|k| \leq l$
- How do we get spatial reuse for UG: H ?

Self-Spatial




- Occurs when we access in order
 - $A[i,j]$: best gain, l
 - $A[i,j*k]$: best gain, l/k if $|k| \leq l$
- How do we get spatial reuse for UG: H ?
- Since all but last index must be identical, so, set last row in H to 0, H_s
 self-spatial reuse vector space = R_{ss}

$$R_{ss} = \ker H_s$$
- Notice, $\ker H \subseteq \ker H_s$
- If, $R_{ss} \cap L = R_{ST} \cap L$, then no additional benefit to SS

Example of self-spatial reuse

```

for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1, I3] += A[I1, I2] * B[I2, I3]
  
```

Access	H_s	$\ker H_s$	reuse?	Local?
$C[I_1, I_3]$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\text{span}\{(0, 1, 0), (0, 0, 1)\}$		
$A[I_1, I_2]$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\text{span}\{(0, 0, 1), (0, 1, 0)\}$		
$B[I_2, I_3]$	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\text{span}\{(1, 0, 0), (0, 0, 1)\}$		

Handwritten red annotations: A red circle around the bottom row of the $A[I_1, I_2]$ matrix, and red arrows pointing from the circled zeros to the corresponding rows in the $\ker H_s$ column.

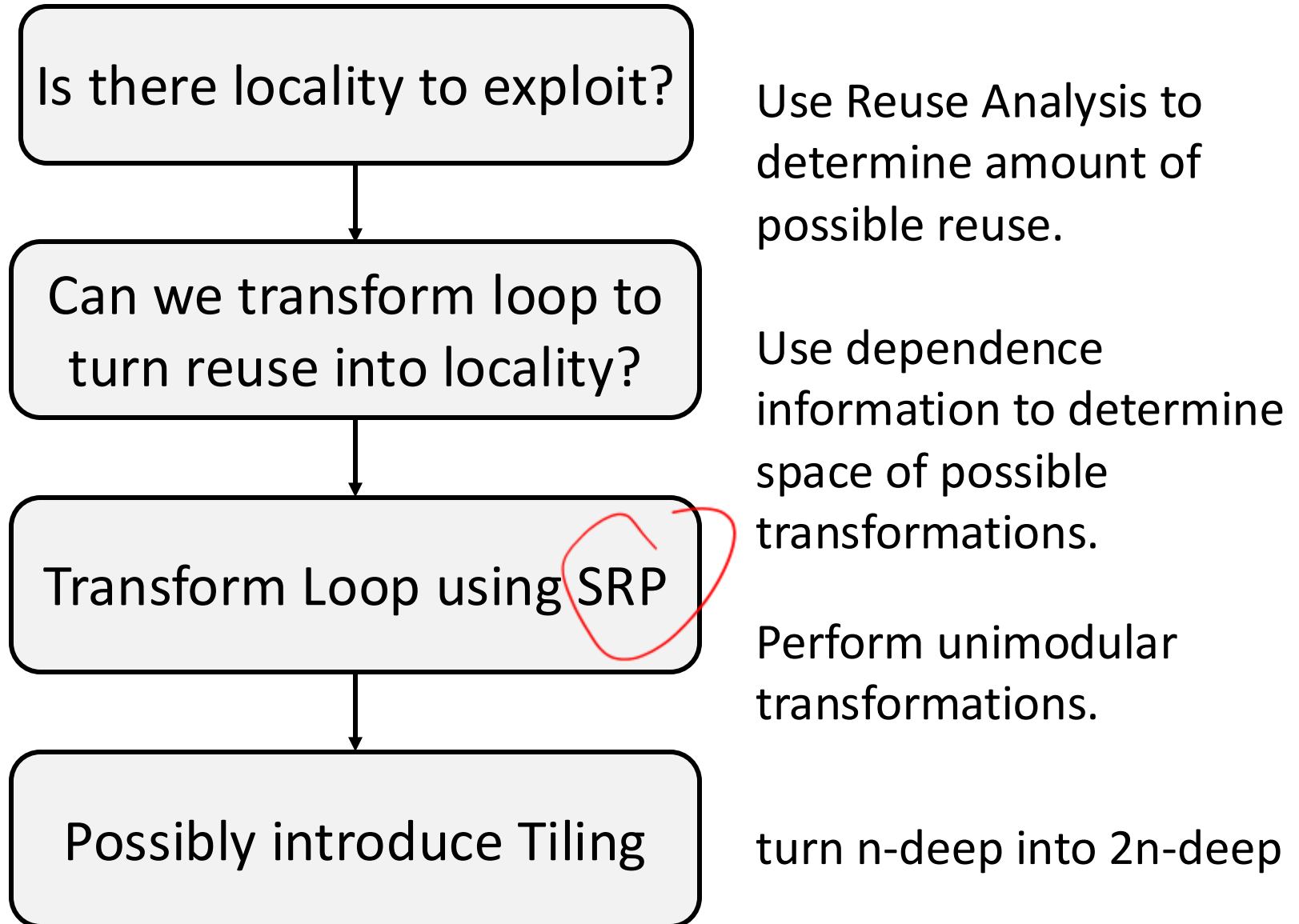
Self-spatial reuse/locality

- $\text{Dim}(R_{SS})$ is dimensionality of reuse vector space.
- If $R_{SS}=0 \rightarrow$ no reuse
- If $R_{SS}=R_{ST}$ no extra reuse from spatial
- Reuse of each element is $k/\ell_s^{\text{dim}(R_{SS})}$ where, s is number of iters per dim.
- $R_{SS} \cap L$ is amount of reuse exploited, therefore number of memory references generated is:
$$k/\ell_s^{\text{dim}(R_{SS} \cap L)}$$

Group Temporal

- Two refs $A[Hi+c]$ and $A[Hi+d]$ can have group temporal reuse in L iff
 - they are from same uniformly generated set
 - There is an $r \in L$ s.t. $Hr = c - d$
- if $c-d = r_p$, then there is group temporal reuse,
 $R_{GT} = \ker H + \text{span}\{r_p\}$
- However, there is no extra benefit if $R_{GT} \cap L = R_{ST} \cap L$

Our Goal: Increase locality



Example of ST reuse

```
for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] = 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
```

Uniformly Generated Set:

$$\{A[I_2], A[I_2+1], A[I_2+2]\} \quad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

<u>Type</u>	<u>reuse space</u>	<u>reuse factor</u>
Self-Temporal:	$\text{Ker}(H) = \text{span}\{(1,0)\}$	s

Example of SS reuse

```

for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] = 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
  
```

Uniformly Generated Set:

$$\{A[I_2], A[I_2+1], A[I_2+2]\} \quad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad H_s = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

<u>Type</u>	<u>reuse space</u>	<u>reuse factor</u>
Self-Temporal:	$\text{Ker}(H) = \text{span}\{(1,0)\}$	s
Self-Spatial:	$\text{Ker}(H_s) = \text{span}\{(1,0), (0,1)\}$	l

Example of GT reuse

```
for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] = 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
```

Uniformly Generated Set:

$$\{A[I_2], A[I_2+1], A[I_2+2]\} \quad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

<u>Type</u>	<u>reuse space</u>	<u>reuse factor</u>
Self-Temporal:	$\text{Ker}(H) = \text{span}\{(1,0)\}$	s
Self-Spatial:	$\text{Ker}(H_s) = \text{span}\{(1,0), (0,1)\}$	l
Group-Temporal:	$\text{span}\{(1,0), (0,1)\}$	3

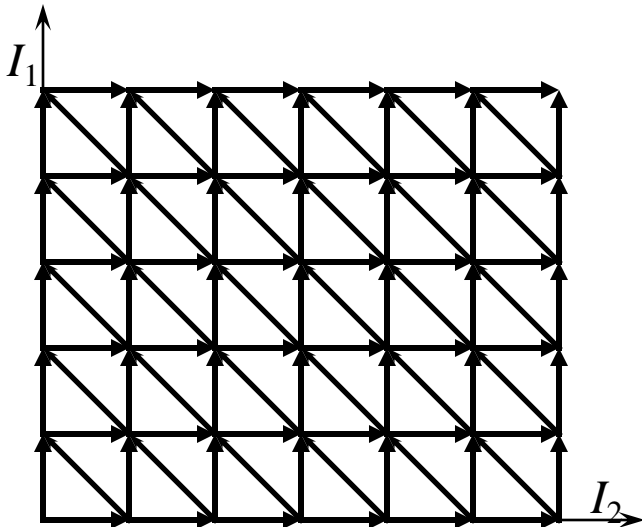
Turning Reuse into Locality

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6

$A[I_2 + 1] = 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$

<u>Type</u>	<u>reuse space</u>	<u>reuse factor</u>
Self-Temporal:	$\text{Ker}(H) = \text{span}\{(1,0)\}$	s
Self-Spatial:	$\text{Ker}(H_s) = \text{span}\{(1,0),(0,1)\}$	l
Group-Temporal:	$\text{span}\{(1,0),(0,1)\}$	3



The Problem

- How to increase locality by transforming loop nest
- Matrix Mult is simple as it is both
 - legal to tile
 - advantageous to tile
- Can we determine the benefit?
(reuse vector space and locality vector space)
- Is it legal (and if so, how) to transform loop?
(unimodular transformations)

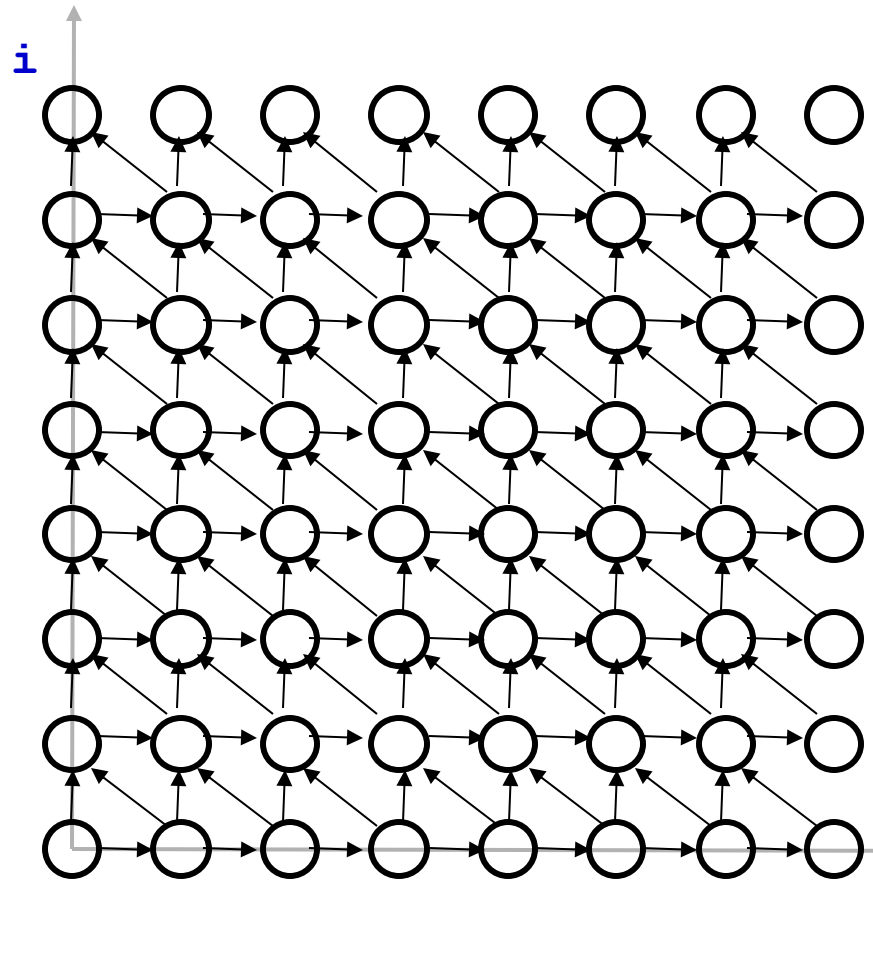
But...is the transform legal?

- Distance/direction vectors give a partial order among points in the iteration space
- A loop transform changes the order in which 'points' are visited
- The new visit order must respect the dependence partial order!

But...is the transform legal?

- What other visit order is legal here?

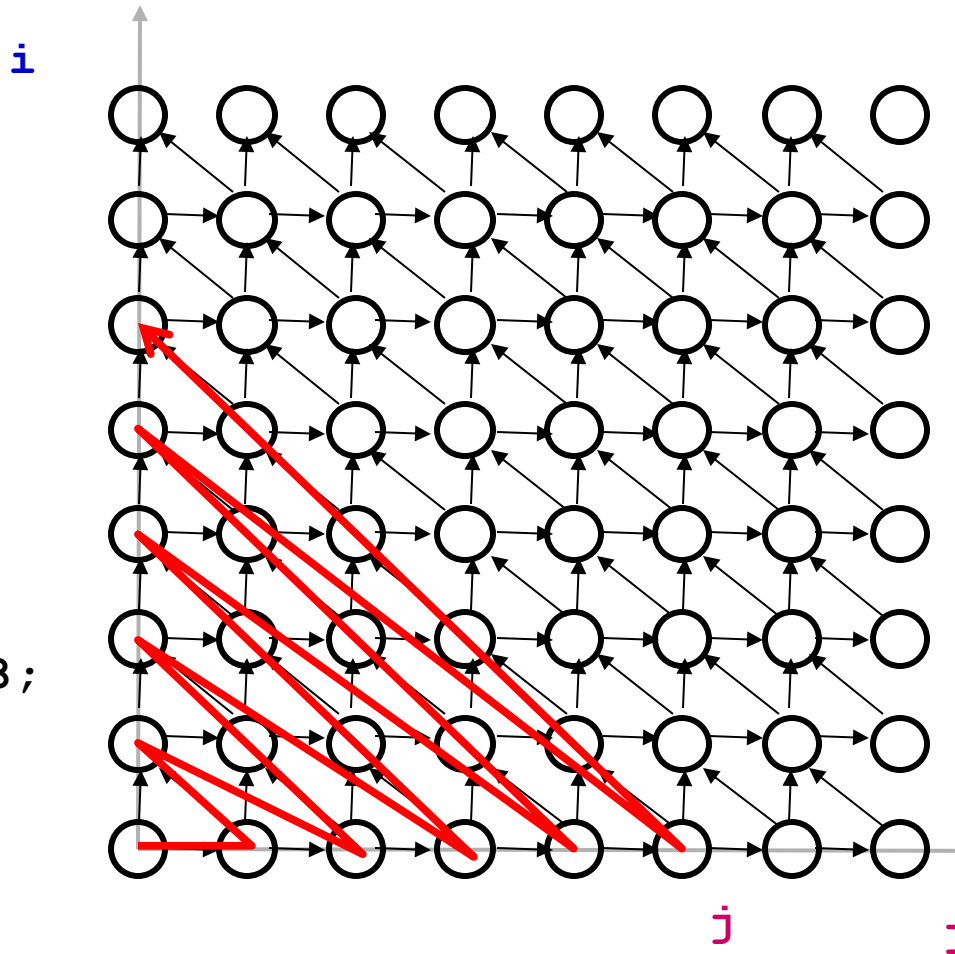
```
for i = 0 to TS
  for j = 0 to N-2
    A[j+1] =
      (A[j] + A[j+1] + A[j+2])/3;
```



But...is the transform legal?

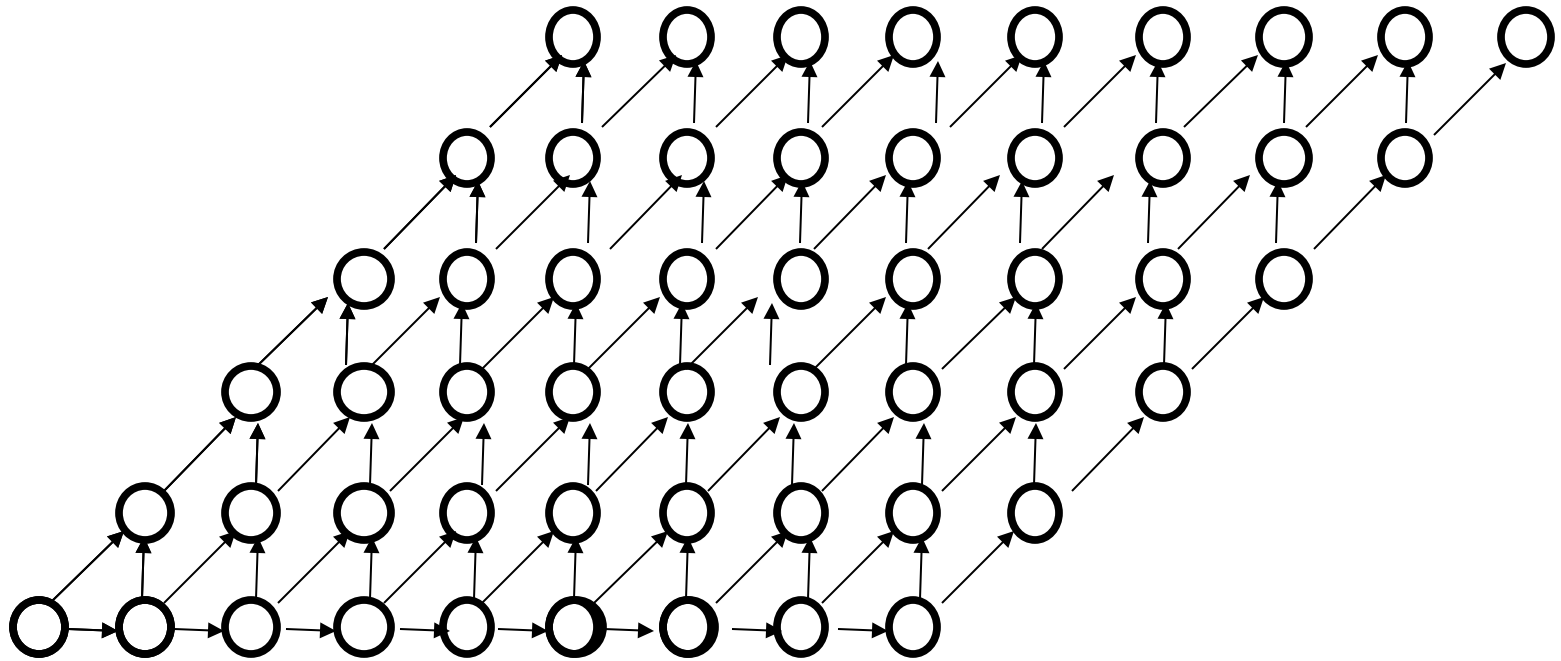
- What other visit order is legal here?

```
for i = 0 to TS  
  for j = 0 to N-2  
    A[j+1] =  
      (A[j] + A[j+1] + A[j+2])/3;
```



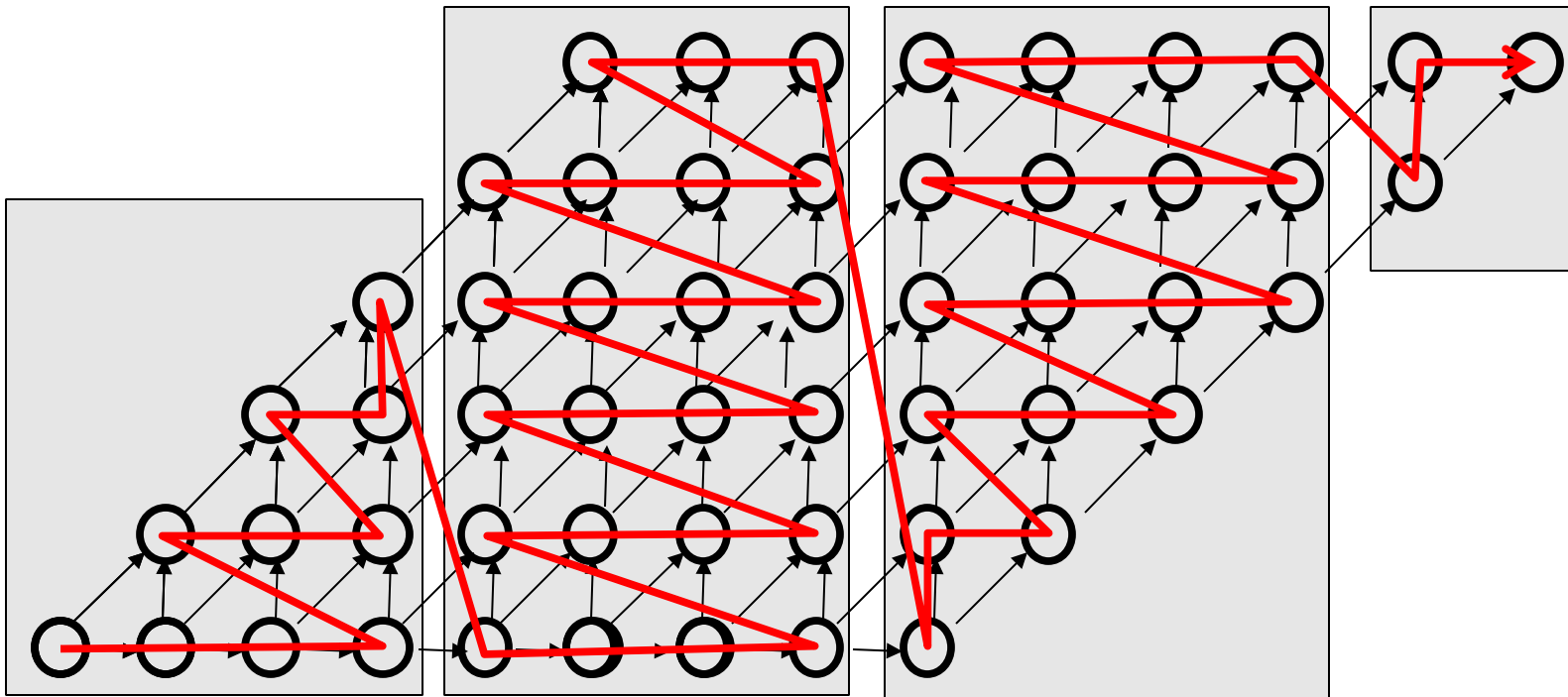
But...is the transform legal?

- Skewing...



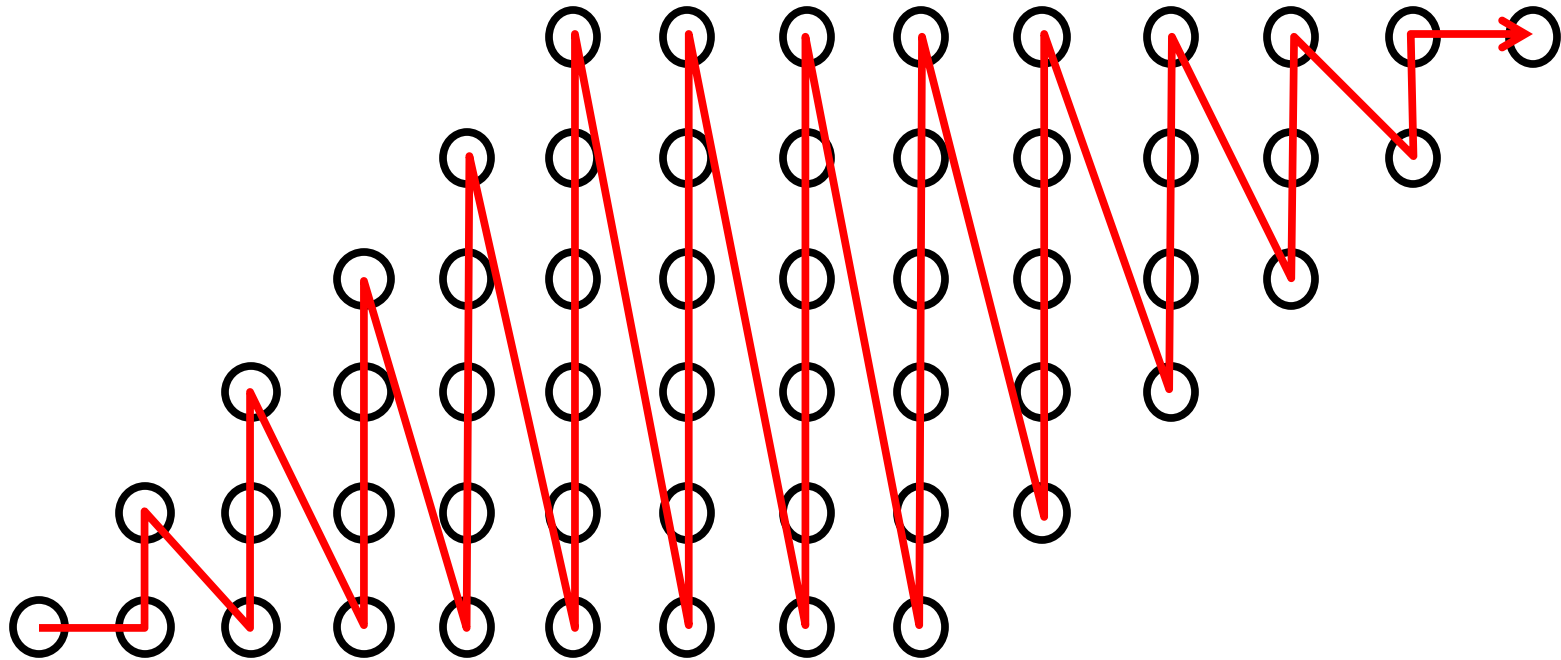
But...is the transform legal?

- Skewing...now we can block



But...is the transform legal?

- Skewing...now we can loop interchange



Unimodular transformations

- Express loop transformation as a matrix multiplication
- Check if any dependence is violated by multiplying the distance vector by the matrix – if the resulting vector is still lexicographically positive, then the involved iterations are visited in an order that respects the dependence.

Reversal

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Interchange

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Skew

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Turning Reuse -> Locality

- Inner most loop(s) will convert their reuse into locality
- We can use unimodular transforms to make the best loop the innermost (obeying dependencies)
- Sometimes reuse is along multiple dimensions, then we need to tile

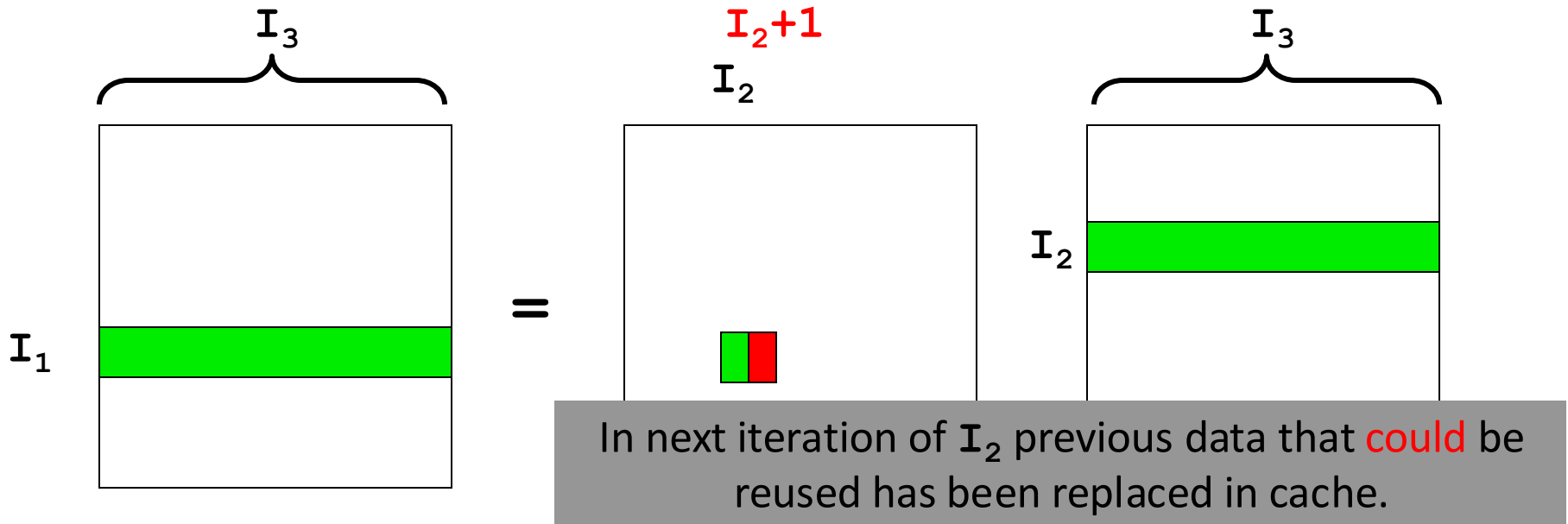
Tiling

- Tiling a loop nest is legal if it is *fully permutable*
- I.e., all dependences in loop nest are
 - lexicographically positive, and,
 - Outer-loops are non-negative
- Transformation to make dependencies legal and then to tile also called:
 - Strip-mine and interchange
 - Unroll and jam
- How big to make tile?

Matrix Multiply

- Fully permutable & L in two loops
- Canonical simple case: Matrix Multiply

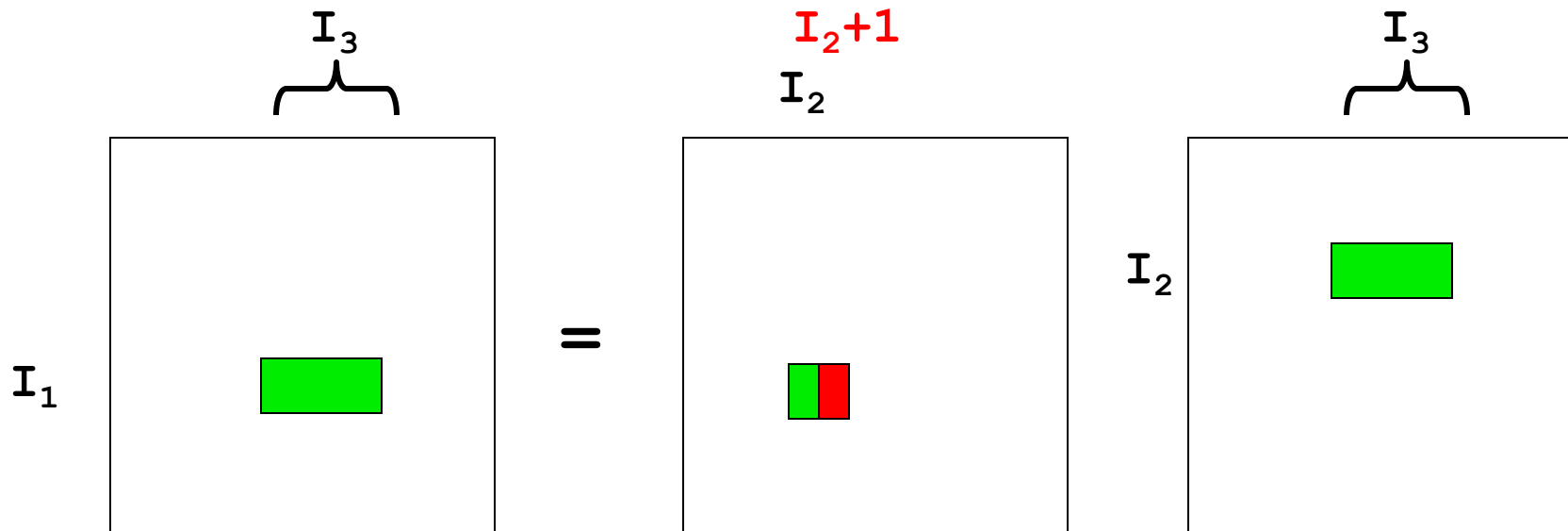
```
for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1, I3] += A[I1, I2] * B[I2, I3]
```



Tiling solves problem

```
for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n
      C[I1, I3] += A[I1, I2] * B[I2, I3]
```

for II₂ := 1 to n by s
 for II₃ := 1 to n by s
 for I₁ := 1 to n
 for I₂ := II₂ to min(II₂ + s - 1, n)
 for I₃ := II₃ to min(II₃ + s - 1, n)
 C[I₁, I₃] += A[I₁, I₂] * B[I₂, I₃];



How much Reuse is Locality?

```

for  $I_1 := 0$  to 5
  for  $I_2 := 0$  to 6
     $A[I_2 + 1] = 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$ 
  
```

<u>Type</u>	<u>reuse space</u>	<u>reuse factor</u>
Self-Temporal:	$\text{Ker}(H) = \text{span}\{(1,0)\}$	s
Self-Spatial:	$\text{Ker}(H_s) = \text{span}\{(1,0),(0,1)\}$	l
Group-Temporal:	$\text{span}\{(1,0),(0,1)\}$	3

If L, localized space, is

$\text{span}\{(0,1)\} \rightarrow 1/\ell$
 $\text{span}\{(1,0)\} \rightarrow 1/\ell s$
 $\text{span}\{(0,1),(1,0)\} \rightarrow 1/\ell s$



How much Reuse is Locality?

```

for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] = 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
    
```

<u>Type</u>	<u>reuse space</u>	<u>reuse factor</u>
Self-Temporal:	$\text{Ker}(H) = \text{span}\{(1,0)\}$	s
Self-Spatial:	$\text{Ker}(H_s) = \text{span}\{(1,0), (0,1)\}$	l
Group-Temporal:	$\text{span}\{(1,0), (0,1)\}$	3

If L, localized space, is

$\text{span}\{(0,1)\} \rightarrow 1/\ell$

~~$\text{span}\{(1,0)\} \rightarrow 1/\ell s$~~

$\text{span}\{(0,1), (1,0)\} \rightarrow 1/\ell s$

So, we want to tile!

Finding best L that is legal

- Of course, exponential in depth of loop nest
- But, loop nest depth is usually small
- And, simplify to look at only elementary basis vectors carrying reuse
- Furthermore, ignore loops
 - With no reuse
 - Must not be innermost due to dependencies
- For each of the remaining loops, look at all subsets to determine which can/should be innermost

SRP

- Heuristic to:
 - make a loop nest fully permutable
 - Or, partition loops into non-negative outer loops and remaining loops

- Thrm: N deep loop nest with lex-pos deps

and a i^{th} loop has d s.t. $(d_{i1} \dots d_{in}) \neq \emptyset$, then for $i < j$
 I_i & I_j can be fully permutable if

$$\forall d \in D^i: \underline{d_j^{\min}} \neq -\infty \text{ and } \underline{d_j^{\min}} < \emptyset \rightarrow \underline{d_i^{\min}} > \emptyset$$

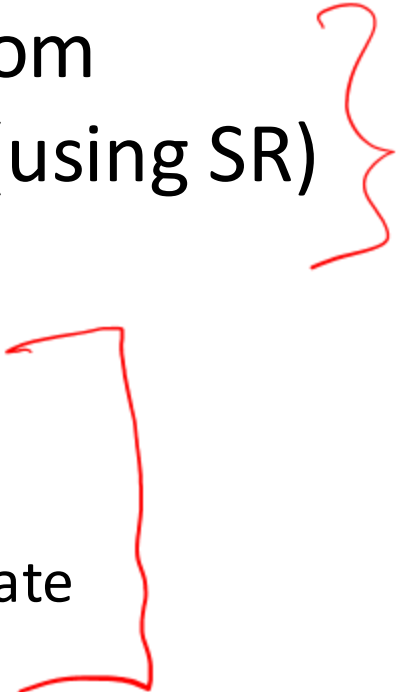
or
vice versa

skew I_j by f wrt I_i

or, reverse & skew

$$f \geq \max \left[\frac{\underline{d_j^{\min}}}{\underline{d_i^{\min}}} \right]$$

Using SRP

- Removing serializing loops(using P)
 - Try and find a fully permutable from remaining loops 1 loop at a time (using SR)
 - If succeed
 - rewrite loops using ~~P~~
 - rewrite loop bounds using T
 - If skewed, rewrite indices to compensate
 - Potentially Tile
- 

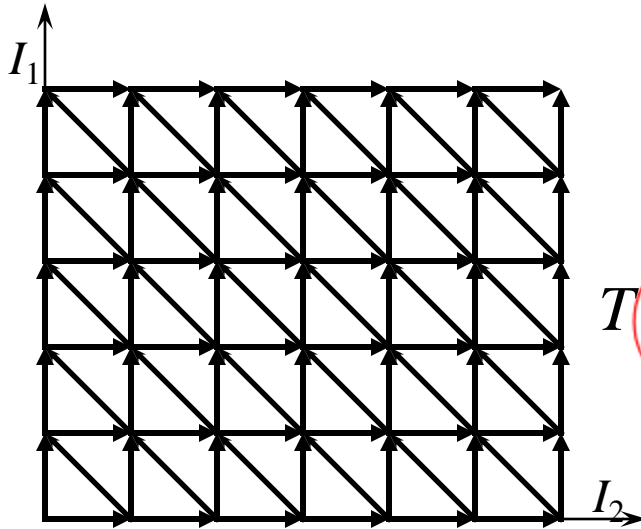
Loop Skewing Example

for $I_1 := 0$ to 5

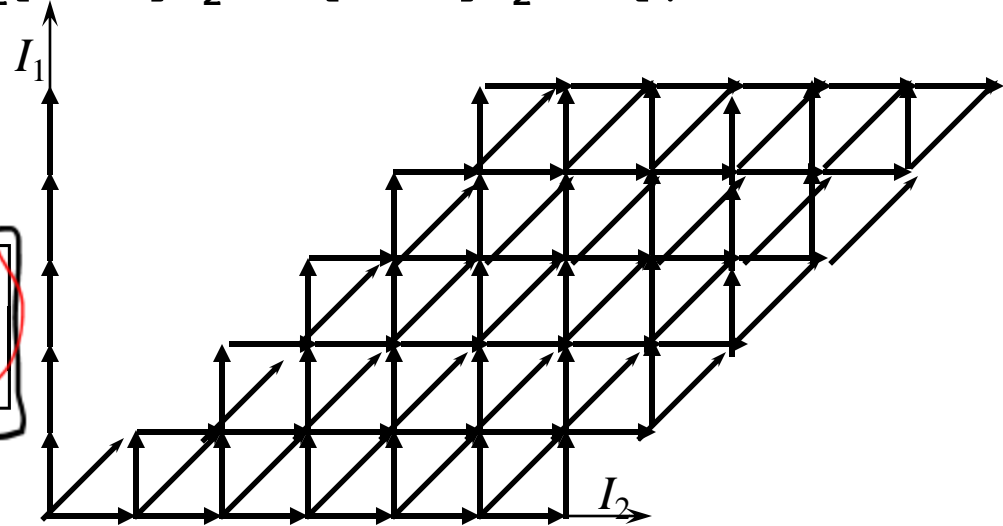
for $I_2 := 0$ to 6

$A[I_2 + 1] := 1/3 * (A[I_2] + A[I_2 + 1] + A[I_2 + 2])$

$D = \{(0,1), (1,0), (1,-1)\}$



$$T \equiv \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$



for $I_1 := 0$ to 5

for $I_2 := I_1$ to $6 + I_1$

$A[I_2 - I_1 + 1] := 1/3 * (A[I_2 - I_1] + A[I_2 - I_1 + 1] + A[I_2 - I_1 + 2])$

$I_1 + I_2$

$D = \{(0,1), (1,1), (1,0)\}$