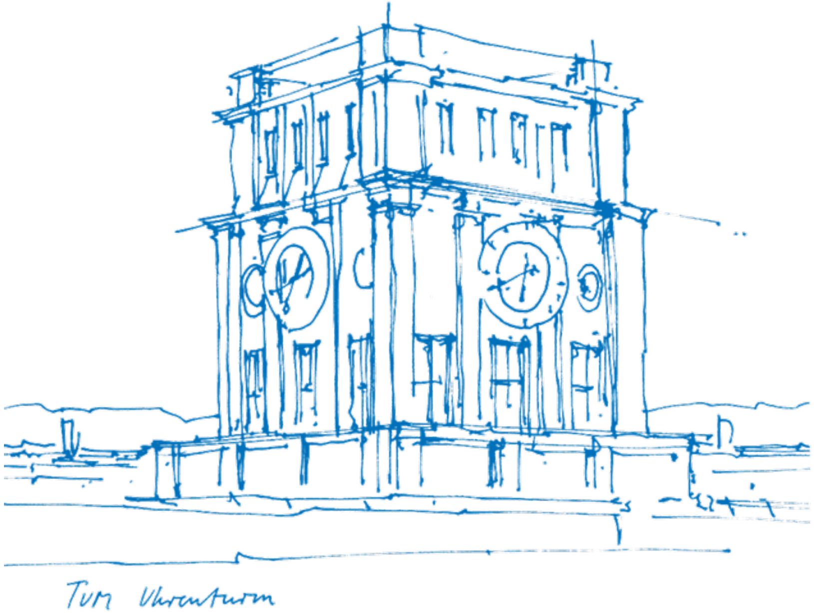
Parallel Programming Tutorial - Loop Transformations

M.Sc. Andreas Wilhelm

Technichal University Munich

June 26, 2017

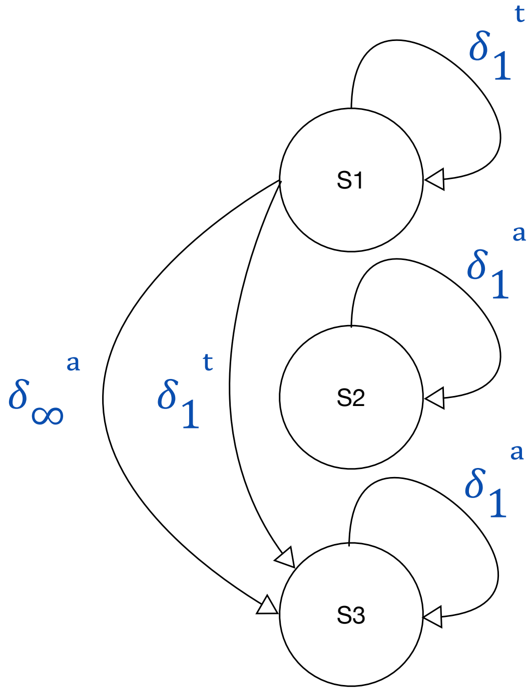


Solution for Assignment 6

Solution for Loop 1

```
for (i = 1; i < N; i++) {
   S1:  A(i+1) = A(i-1) + 3 * B(i)
   S2:  C(i)     = 2 * C(i+1)
   S3:  B(i)     = A(i) + C(i+2)
}
```

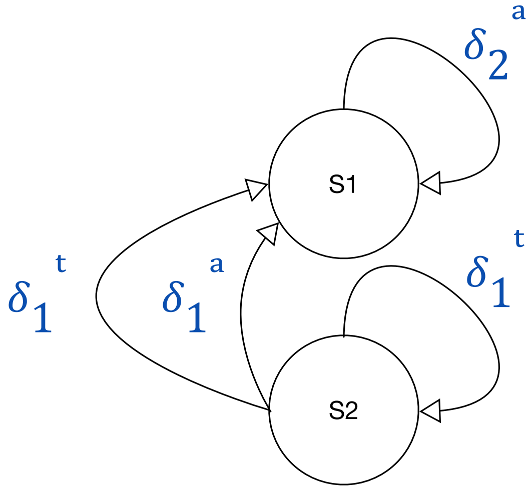
Source	Sink	Dep.Type	Dist. Vector	Dir. Vector
S1: A(i+1)	S1: A(i-1)	true	(2)	(<)
S1: B(i)	S3: B(i)	anti	(0)	(=)
S2: C(i+1)	S2: C(i)	anti	(1)	(<)
S1: A(i+1)	S3: A(i)	true	(1)	(<)
S3: C(i+2)	S2: C(i)	anti	(2)	(<)



Solution for Loop 2

```
for (i = 1; i < n; i++) {
   for (j = 1; j < m; j++) {
      S1: B(i, j) = B(i, j+2) + A(i-1, j+1)
      S2: A(i+1, j) = A(i-1, j) * B(i+1, j-1)
   }
}
```

Source	Sink	Dep.Type	Dist. Vector	Dir. Vector
S1: B(i, j+2)	S1: B(i, j)	anti	(0,2)	(=,<)
S2: A(i+1, j)	S2: A(i-1, j)	true	(2,0)	(<,=)
S2: A(i+1, j)	S1: A(i-1, j+1)	true	(2,-1)	(<,>)
S2: B(i+1, j-1)	S1: B(i,j)	anti	(1,-1)	(<,>)



Solution for Loop 3

```
for (i = 1; i < 3; i++) {
   for (j = 1; j < 3; j++) {
      S1: B(2*i, j) = A(i, 3-j)
      S2: A(i, j) = B(i+2, j+1)
   }
}
```

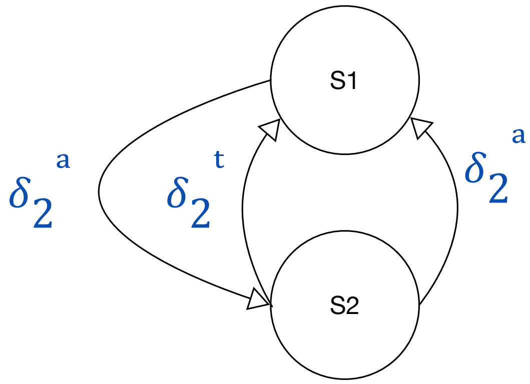
Source	Sink	Dep. Type	Dist. Vector	Dir. Vector
S2: A(i, j)	S1: A(i, 3-j)	true	(0,1)	(=,<)
S1: A(i, 3-j)	S2: A(i, j)	anti	(0,1)	(=,<)
S2: B(i+2, j+1)	S1: B(2*i, j)	anti	(0,1)	(=,<)

(1,1)
```
S1: B(2, 1) = A(1, 2)
S2: A(1, 1) = B(3, 2)
```

(1,2)
```
S1: B(2, 2) = A(1, 1)
S2: A(1, 2) = B(3, 3)
```

(2,1)
```
S1: B(4, 1) = A(2, 2)
S2: A(2, 1) = B(4, 2)
```

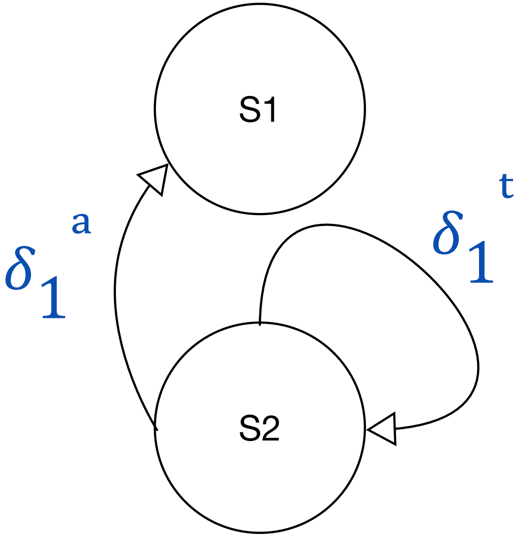
(2,2)
```
S1: B(4, 2) = A(2, 1)
S2: A(2, 2) = B(4, 3)
```



Solution for Loop 4

```
for (i = 1; i < n; i++) {
   for (j = 1; j < m; j++) {
      S1: A(i, j) = B(2*i, j)
      S2: C(2*i, j) = C(i, j-1) + A(i+1, j-1)
   }
}
```

Source	Sink	Dep.Type	Dist. Vector	Dir. Vector
S2: C(2*i, j)	S2: C(i, j-1)	true	(i,1)	(<,<)
S2: A(i+1, j-1)	S1: A(i, j)	anti	(1,-1)	(<,>)



Evaluation

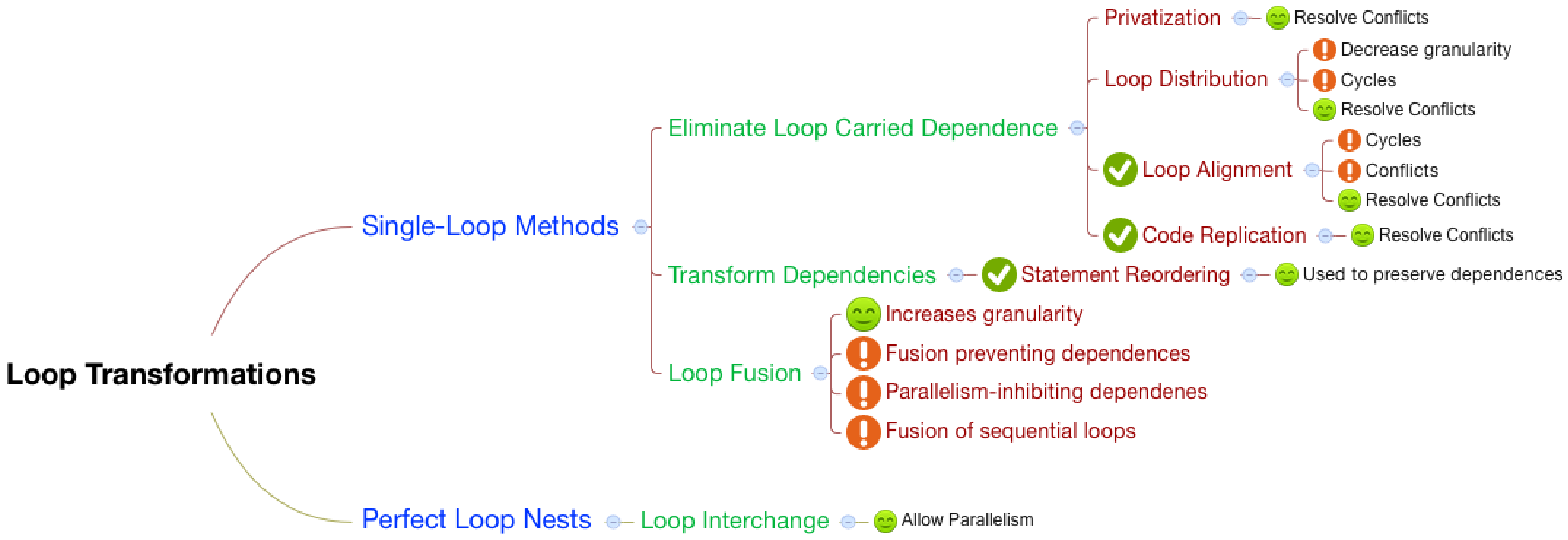
Loop Transformations

Transformations

Theorem

Any reordering transformation that preserves every dependence in a program preserves the meaning of that program.

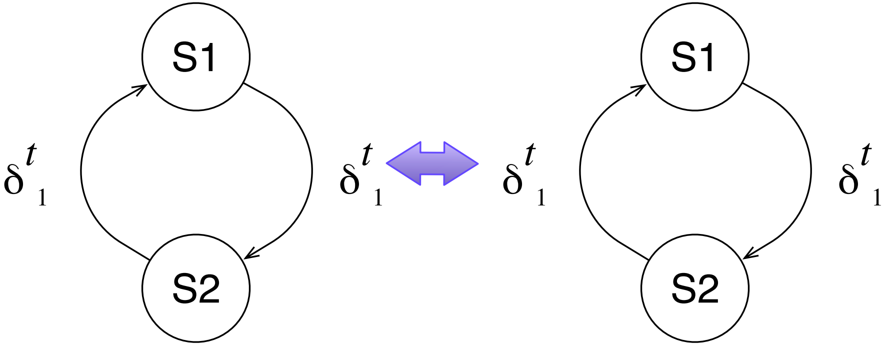
Transformations - Mindmap



Statement Reordering

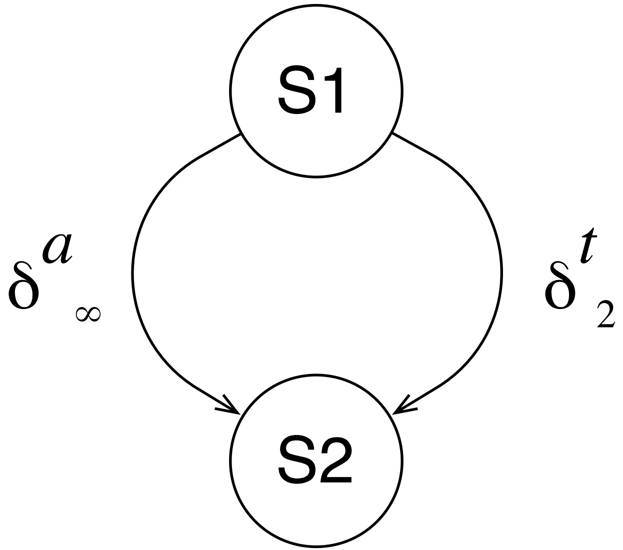
```
for (i=1; i<10; i++) {
   S1:   A(i+1) = F(i)
   S2:   F(i+1) = A(i)
}
```

```
for (i=1; i<10; i++) {
   S2:   F(i+1) = A(i)
   S1:   A(i+1) = F(i)
}
```



Loop Distribution I

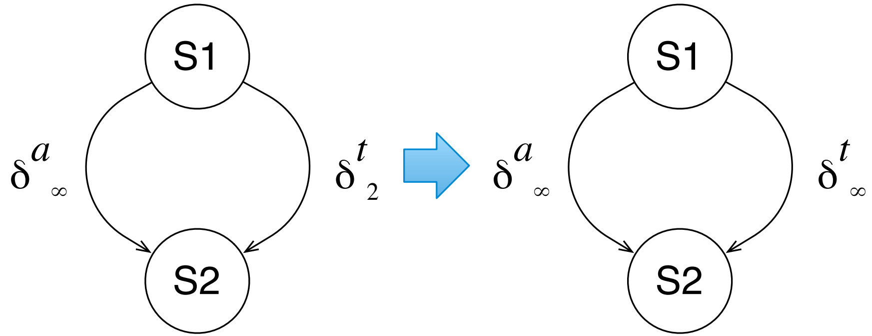
```
for (i=1; i<n; i++) {
   for (j=1; j<m; j++) {
      S1:   A(i,j) = B(i,j)
      S2:   B(i,j) = A(i,j-1)
   }
}
```



Loop Distribution I

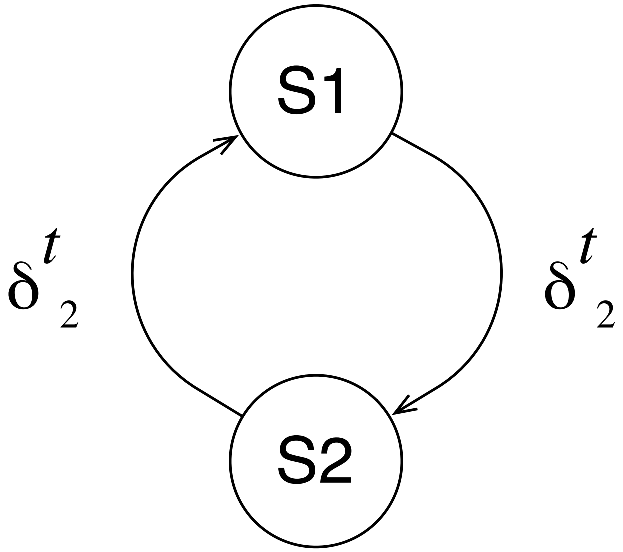
```
for (i=1; i<n; i++) {
   for (j=1; j<m; j++) {
      S1:   A(i,j) = B(i,j)
      S2:   B(i,j) = A(i,j-1)
   }
}
```

```
for (i=1; i<n; i++) {
   for (j=1; j<m; j++) {
      S1:   A(i,j) = B(i,j)
   }
   for (j=1; j<m; j++) {
      S2:   B(i,j) = A(i,j-1)
   }
}
```



Loop Distribution II - Cycle

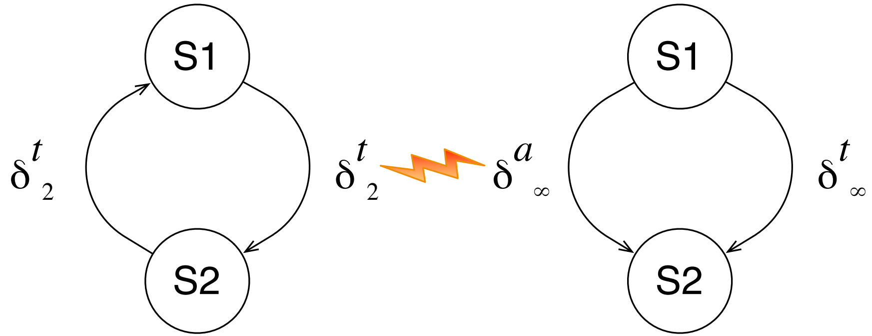
```
for (i=1; i<n; i++) {
   for (j=1; j<m; j++) {
      S1:   A(i,j) = B(i,j)
      S2:   B(i,j+1) = A(i,j-1)
   }
}
```



Loop Distribution II - Cycle

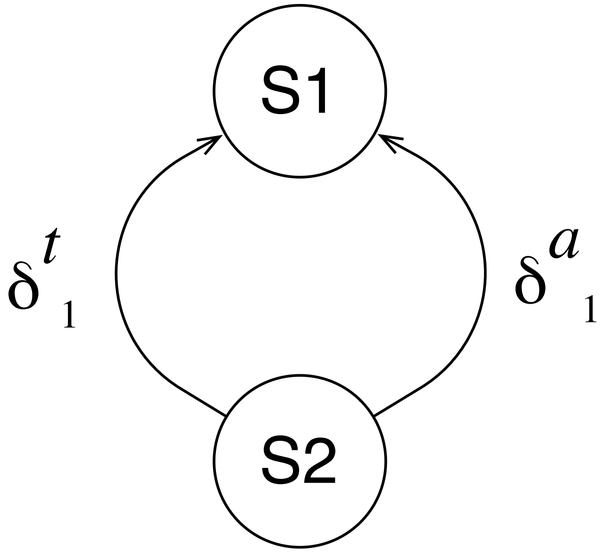
```
for (i=1; i<n; i++) {
   for (j=1; j<m; j++) {
      S1:   A(i,j) = B(i,j)
      S2:   B(i,j+1) = A(i,j-1)
   }
}
```

```
for (i=1; i<n; i++) {
   for (j=1; j<m; j++) {
      S1:   A(i,j) = B(i,j)
   }
   for (j=1; j<m; j++) {
      S2:   B(i,j+1) = A(i,j-1)
   }
}
```



Loop Alignment I

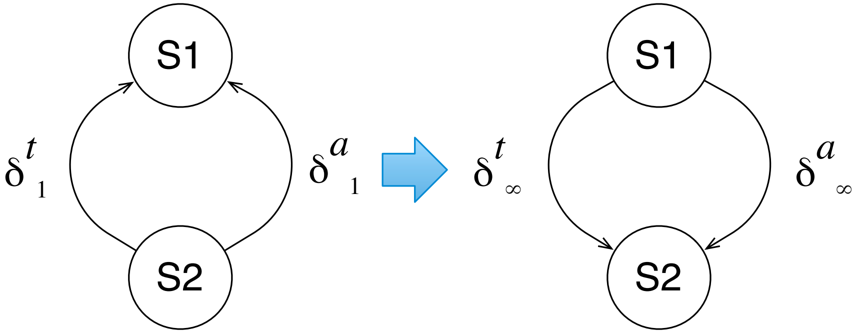
```
for (i=1; i<n; i++) {
   S1:   A(i)   = B(i)
   S2:   B(i+1) = A(i+1)
}
```



Loop Alignment I

```
for (i=1; i<n; i++) {
   S1:   A(i)   = B(i)
   S2:   B(i+1) = A(i+1)
}
```

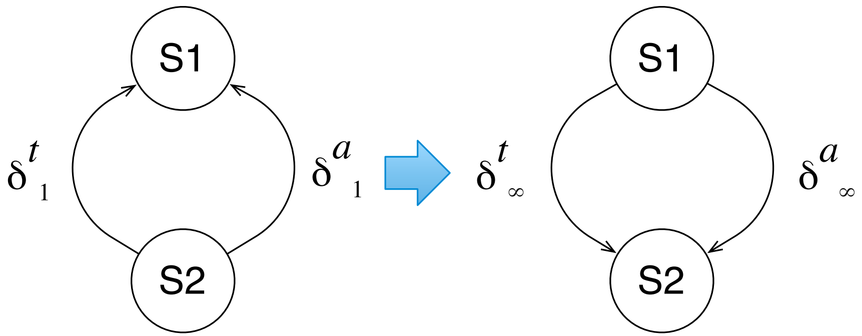
```
for (i=1; i<n+1; i++) {
   S1:   if (i<n) A(i) = B(i)
   S2:   if (i>1) B(i) = A(i)
}
```



Loop Alignment I - Peeling Off Executions

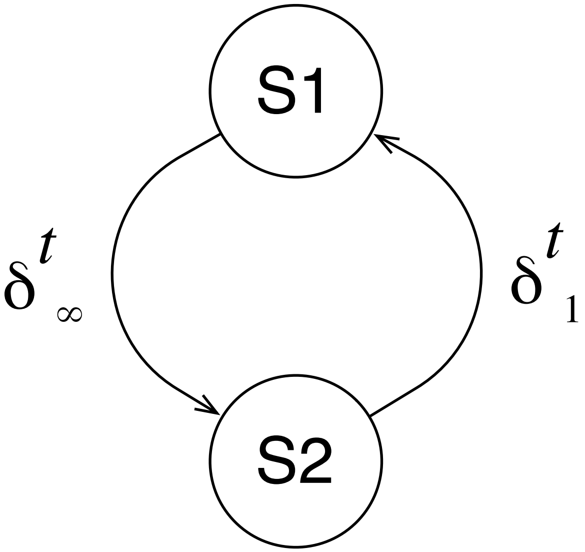
```
for (i=1; i<n; i++) {
   S1:   A(i)   = B(i)
   S2:   B(i+1) = A(i+1)
}
```

```
A(1) = B(1)
for (i=2; i<n; i++) {
   S2:   B(i) = A(i)
   S1:   A(i) = B(i)
}
B(n) = A(n)
```



Loop Alignment II - Cycle

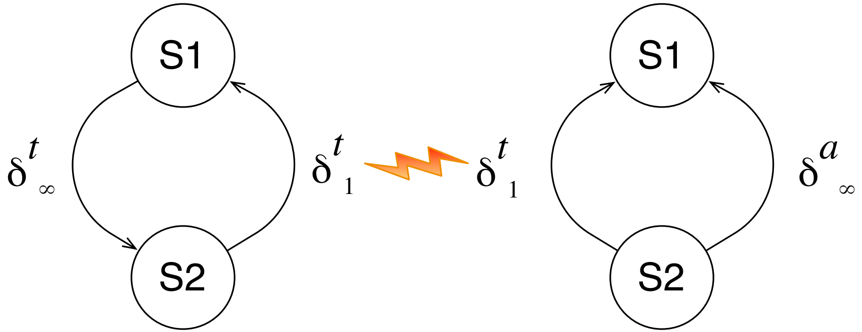
```
for (i=1; i<n; i++) {
   S1:   A(i)   = B(i)
   S2:   B(i+1) = A(i)
}
```



Loop Alignment II - Cycle

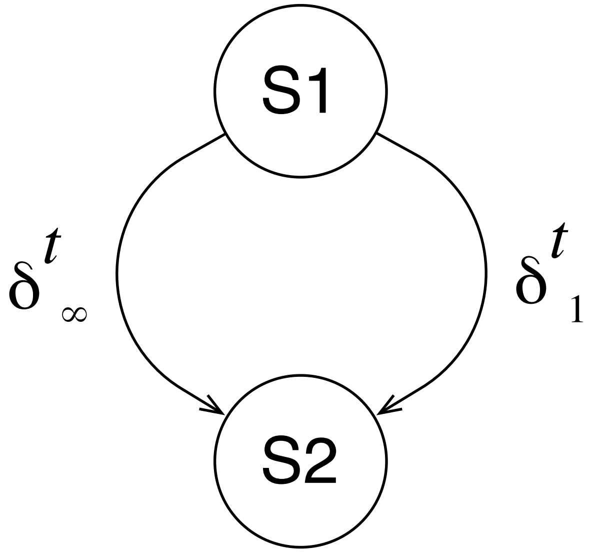
```
for (i=1; i<n; i++) {
   S1:   A(i)   = B(i)
   S2:   B(i+1) = A(i)
}
```

```
for (i=1; i<n+1; i++) {
   S1:   if (i<n) A(i) = B(i)
   S2:   if (i>1) B(i) = A(i-1)
}
```



Loop Alignment III - Conflict

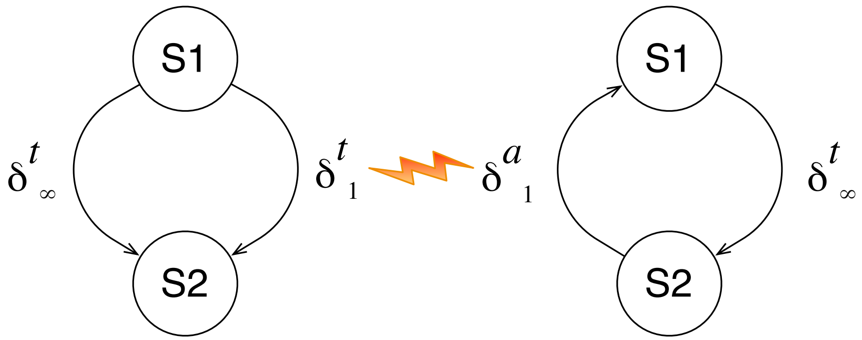
```
for (i=1; i<n; i++) {
   S1: A(i) = B(i)
   S2: C(i) = A(i) + A(i-1)
}
```



Loop Alignment III - Conflict

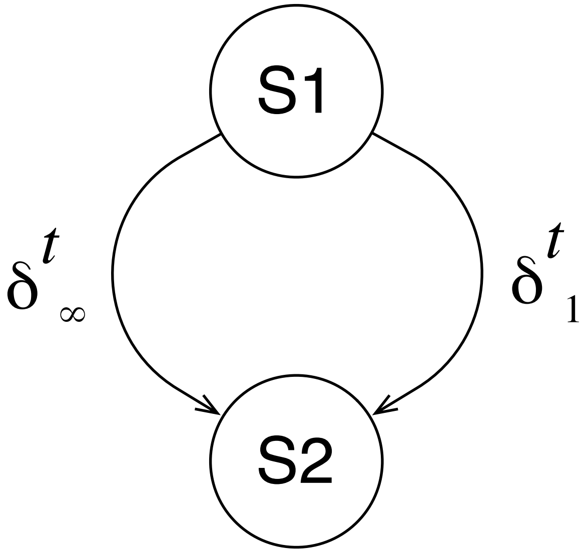
```
for (i=1; i<n; i++) {
   S1: A(i) = B(i)
   S2: C(i) = A(i) + A(i-1)
}
```

```
for (i=0; i<n; i++) {
   S1: if (i>0)   A(i)   = B(i)
   S2: if (i<n+1) C(i+1) = A(i+1)+A(i)
}
```



Code Replication

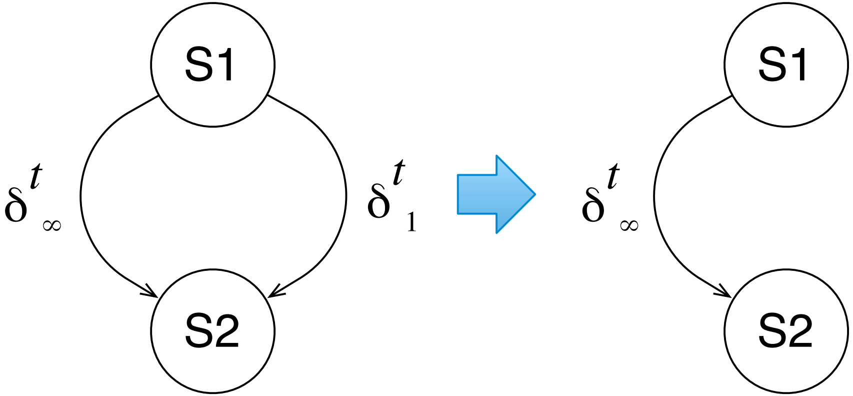
```
for (i=1; i<n; i++) {
    S1:   A(i) = B(i)
    S2:   C(i) = A(i) + A(i-1)
}
```



Code Replication

```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i)
   S2:   C(i) = A(i) + A(i-1)
}
```

```
for (i=1; i<n; i++) {
   private(T)
   S1:   A(i)   = B(i)
   if (i=1) T = A(0)
   else     T = B(i-1)
   S2:   C(i) = A(i) + T
}
```



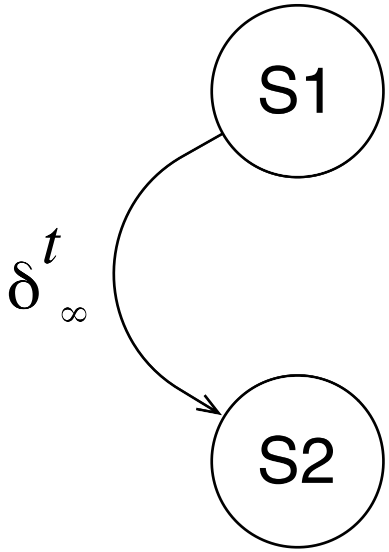
Transformations

Theorem

Alignment, replication, and statement reordering are sufficient to eliminate all carried dependences in a single loop that contains no cyclic dependence and in which the distance of each dependence is a constant independent of the loop index.

Loop Fusion I

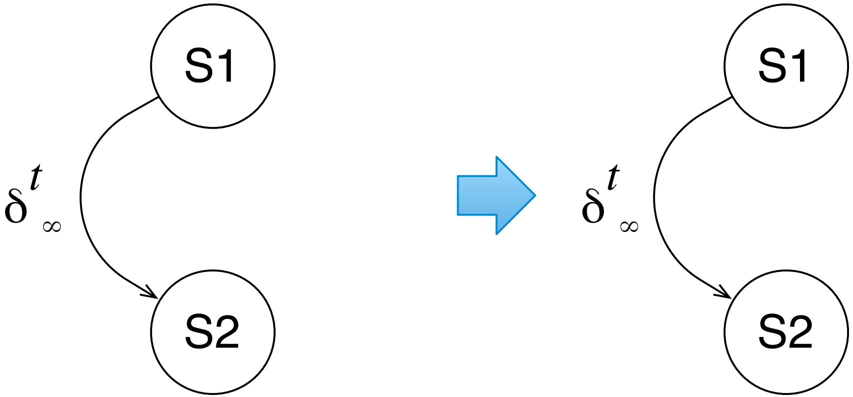
```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i+1)
}
for (i=1; i<n; i++) {
   S2:   C(i) = A(i) + B(i)
}
```



Loop Fusion I

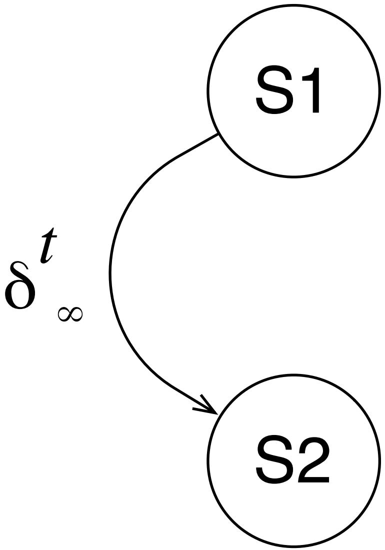
```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i+1)
}
for (i=1; i<n; i++) {
   S2:   C(i) = A(i) + B(i)
}
```

```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i+1)
   S2:   C(i) = A(i) + B(i)
}
```



Loop Fusion II - Fusion preventing Dependency

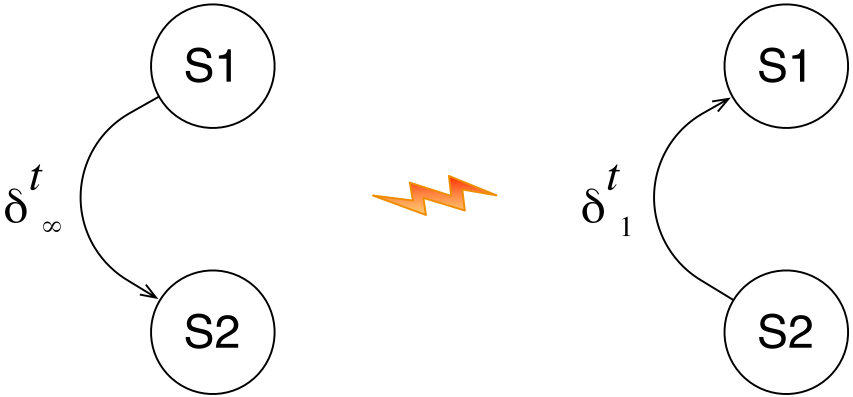
```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i+1)
}
for (i=1; i<n; i++) {
   S2:   C(i) = A(i+1) + B(i)
}
```



Loop Fusion II - Fusion preventing Dependency

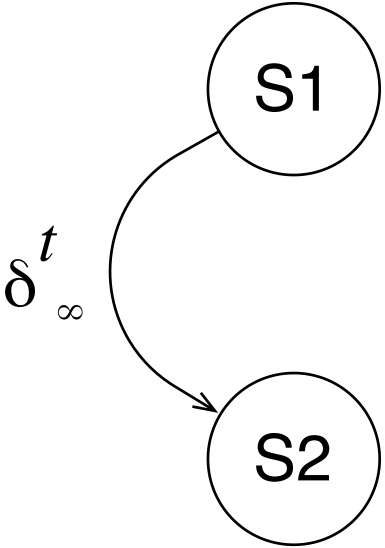
```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i+1)
}
for (i=1; i<n; i++) {
   S2:   C(i) = A(i+1) + B(i)
}
```

```
for (i=1; i<n; i++) {
   S1:   A(i) = B(i+1)
   S2:   C(i) = A(i+1) + B(i)
}
```



Loop Fusion III - Parallelism inhibiting Dependency

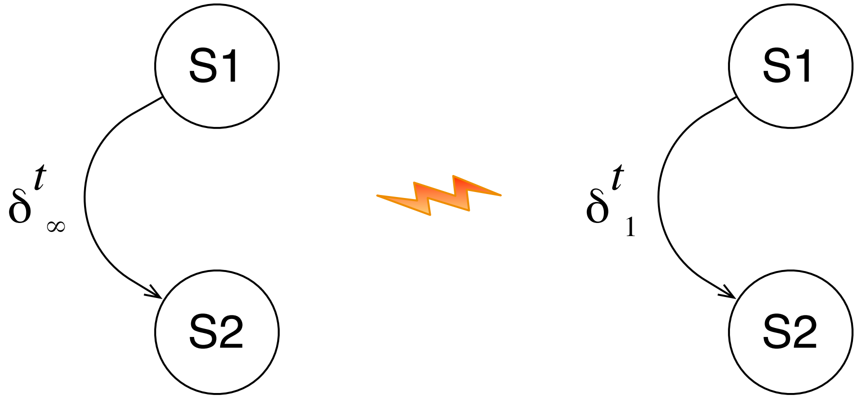
```
for (i=1; i<n; i++) {
   S1:   A(i+1) = B(i+1)
}
for (i=1; i<n; i++) {
   S2:   C(i) = A(i) + B(i)
}
```



Loop Fusion III - Parallelism inhibiting Dependency

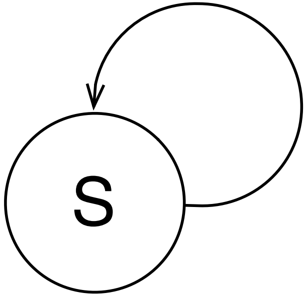
```
for (i=1; i<n; i++) {
   S1:   A(i+1) = B(i+1)
}
for (i=1; i<n; i++) {
   S2:   C(i) = A(i) + B(i)
}
```

```
for (i=1; i<n; i++) {
   S1:   A(i+1) = B(i+1)
   S2:   C(i)   = A(i) + B(i)
}
```



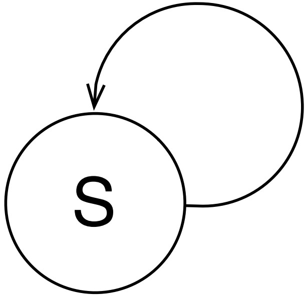
Loop Interchange

```
for (i=1; i<n; i++) {
   for(j=1; j<m; j++) {
      S:    A(i+1,j) = A(i,j) + B(i,j)
   }
}
```



δ_1^t

```
for (j=1; j<m; j++) {
   for(i=1; i<n; i++) {
      S:    A(i+1,j) = A(i,j) + B(i,j)
   }
}
```

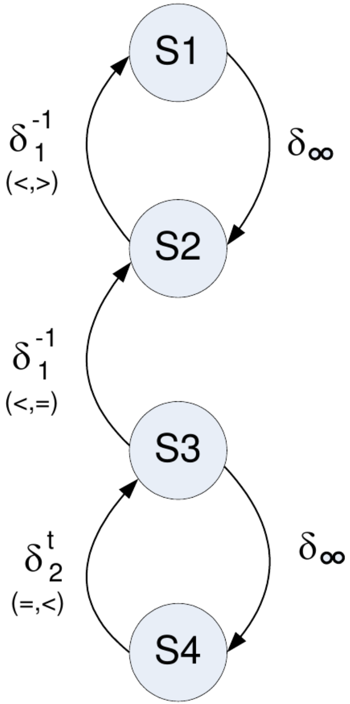


δ_2^t

Exam 2010 Q4

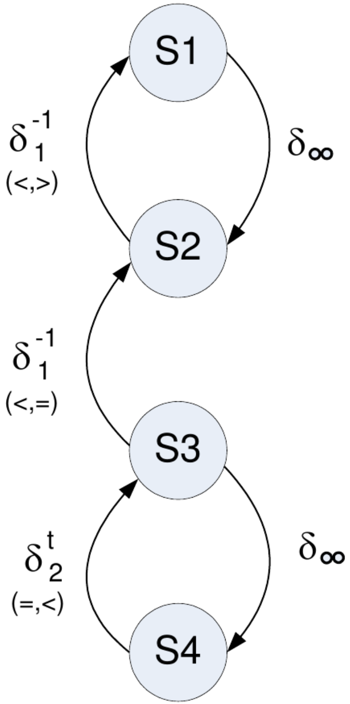
Apply loop distribution to the following loop nest. Distribute it as far as possible.Other transformations might help you. Mark loops with OMP FOR that can be run in parallel.

```
    for (i=...)
      for (j=...)
      {
S1:     ...
S2:     ...
S3:     ...
S4:     ...
      }
```



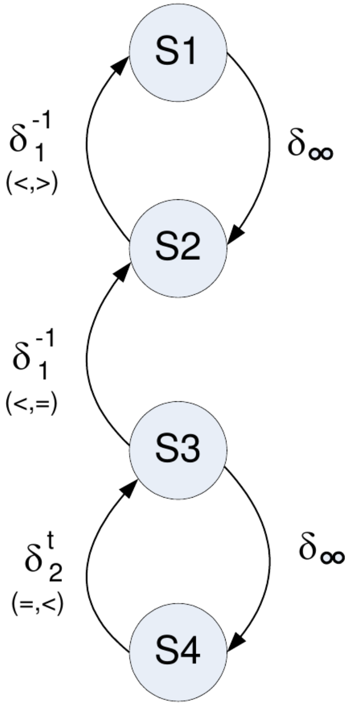
Exam 2010 Q4: Solution Step 1

```
    for (i=...)
      for (j=...)
      {
S3:     ...
S4:     ...
      }

    for (i=...)
      for (j=...)
      {
S1:     ...
S2:     ...
      }
```



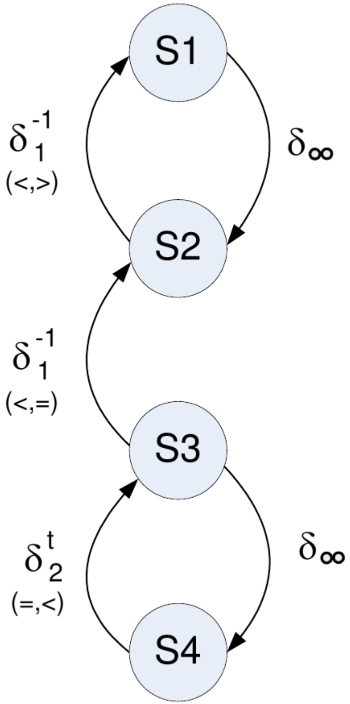
Exam 2010 Q4: Solution Step 2

```
    for (j=...)
      for (i=...)
      {
S3:     ...
S4:     ...
      }

    for (i=...)
      for (j=...)
      {
S1:     ...
S2:     ...
      }
```



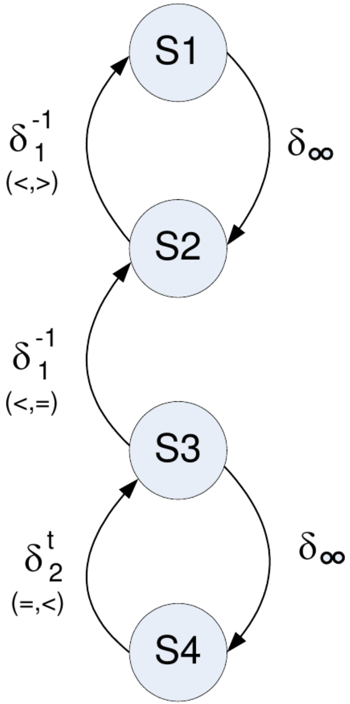
Exam 2010 Q4: Solution Step 3

```
     for (j=...)
     {
       for (i=...)
S3:      ...
       for (i=...)
S4:      ...
     }

     for (i=...)
       for (j=...)
       {
S1:      ...
S2:      ...
       }
```



Exam 2010 Q4: Solution Step 4

```
    for (j=...)
    {
      #pragma omp parallel for
      for (i=...)
S3:     ...
      #pragma omp parallel for
      for (i=...)
S4:     ...
    }
    for (i=...)
      for (j=...)
      {
S1:     ...
S2:     ...
      }
```



Exam 2012 Q7

Transform the following loop into a parallel loop using loop alignment. Do not distribute the loop. The generated code should not have if-statements in the loop body.

```
      for(i=2; i<n; i++) {
S1:      B(i) = A(i)
S2:      C(i) = C(i) + B(i+1)
      }
```

Exam 2012 Q7: Solution Step 1

```
        for(i=2; i<n; i++) {
S1:         B(i) = A(i)
S2:         C(i) = C(i) + B(i+1)
        }
```

Shift S1 right: i+1

```
        for(i=1; i<n; i++) {
S1:     if(i < n-1) B(i+1) = A(i+1)
S2:     if(i > 1  ) C(i) = C(i) + B(i+1)
        }
```

Exam 2012 Q7: Solution Step 2

```
        for(i=2; i<n; i++) {
S1:         B(i) = A(i)
S2:         C(i) = C(i) + B(i+1)
        }
```

Shift S1 right: i+1

```
        i=1
S1:     if(i < n-1) B(i+1) = A(i+1)
S2:     if(i > 1  ) C(i) = C(i) + B(i+1)
        for(i=2; i<n-1; i++) {
S1:     if(i < n-1) B(i+1) = A(i+1)
S2:     if(i > 1  ) C(i) = C(i) + B(i+1)
        }
        i=n-1
S1:     if(i < n-1) B(i+1) = A(i+1)
S2:     if(i > 1  ) C(i) = C(i) + B(i+1)
```

Exam 2012 Q7: Solution Step 3

```
      for(i=2; i<n; i++) {
S1:      B(i) = A(i)
S2:      C(i) = C(i) + B(i+1)
      }
```

Shift S1 right: i+1

```
      i=1
S1:   B(i+1) = A(i+1)
      for(i=2; i<n-1; i++) {
S1:      B(i+1) = A(i+1)
S2:      C(i) = C(i) + B(i+1)
      }
      i=n-1
S2:   C(i) = C(i) + B(i+1)
```

Exam 2012 Q7: Solution Step 4

```
      for(i=2; i<n; i++) {
S1:      B(i) = A(i)
S2:      C(i) = C(i) + B(i+1)
      }
```

Shift S1 right: i+1

```
S1:   B(2) = A(2)
      for(i=2; i<n-1; i++) {
S1:      B(i+1) = A(i+1)
S2:      C(i) = C(i) + B(i+1)
      }
S2:   C(n-1) = C(n-1) + B(n)
```

Exam 2012 Q7: Solution Step 4

```
      for(i=2; i<n; i++) {
S1:      B(i) = A(i)
S2:      C(i) = C(i) + B(i+1)
      }
```

Shift S1 right: i+1

```
S1:   B(2) = A(2)
      for(i=2; i<n-1; i++) {
S2:      C(i) = C(i) + B(i+1)
S1:      B(i+1) = A(i+1)
      }
S2:   C(n-1) = C(n-1) + B(n)
```

Assignment 7

Assignment 7: Loop Transformations

1. Assignment 7a
   - Apply loop distribution to the loop in `loop_fission_seq.c`
   - Distribute it as far as possible, other transformations may help
   - Parallelize the loop with OpenMP in `loop_fission_par.c` and upload it
2. Assignment 7b
   - Apply loop alignment to the loop in `loop_alignment_seq.c`
   - Do not distribute the loop
   - Parallelize with OpenMP in `loop_alignment_par.c` and upload it
3. Assignment 7c
   - Apply loop fusion to the loop in `loop_fusion_seq.c`
   - Parallelize the loop with OpenMP in `loop_fusion_par.c` and upload it