Dependence Analysis

(CS 3006)

Department of Computer Science,

National University of Computer & Emerging Sciences,
Islamabad Campus

The Big Picture

1. **What are our goals?**

- **Simple Goal**: Make execution time as small as possible

2. **Which leads to:**

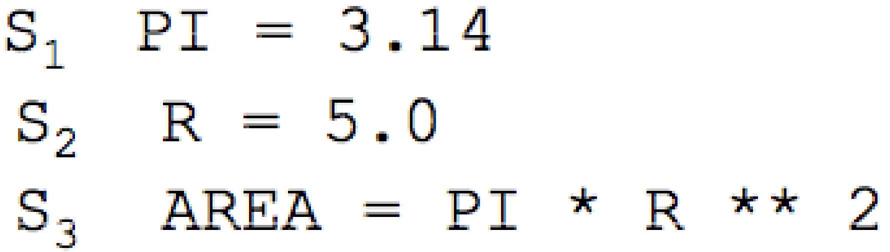
– Achieve **execution** of **many** (all, in the best case) **instructions** in **parallel**

– But, you have to find **INDEPENDENT** instructions

Data Dependence

- **Data** must be **produced** and **consumed** in the ***correct order***

- **Simple example of data dependence:**



S_1 PI = 3.14

S_2 R = 5.0

S_3 AREA = PI * R ** 2

- Statement ***S3*** cannot be moved before either ***S1*** or ***S2*** without compromising correct results

Motivation

- **DOALL loops: loops whose iterations can execute in parallel**

```
for i = 11, 20
     a[i] = a[i] + 3
```

Examples

```
for i = 11, 20
    a[i] = a[i] + 3
```

Parallel

```
for i = 11, 20
  a[i] = a[i-1] + 3
```

NOT Parallel?

```
for i = 11, 20
  a[i] = a[i-10] + 3
```

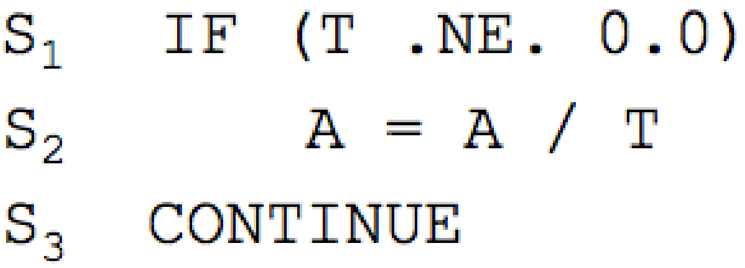
Parallel ?

Dependence Analysis

- A **dependence** is a **relationship** between **2 computations** that **places constraints** on their **execution order**
- **Dependence analysis identifies** these **constraints**
- **Constraints** are used to **determine** whether a **particular transformation** can be **applied** without changing the computation's **semantics**
- **2 types of dependences**: **control** and **data** dependences
- ***Both of them must be considered*** when parallelizing programs.

Control Dependence

- There is a **control dependence** between **S1** and **S2,** when **S1 determines whether S2 will be executed or not**
- **Example:**



```
S1   IF (T .NE. 0.0)
S2        A = A / T
S3   CONTINUE
```

- Executing **S2** before **S1** could cause a **divide by zero** exception (in this example).
- **S2 is conditional** upon the execution of the branch in S1.

Data Dependence

- Two **statements** have a **data dependence** if they **cannot be executed simultaneously** due to **conflicting uses of the same data.**

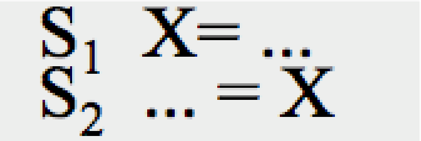
- **Ensure that data is produced and consumed in the right order:**
    1. do not interchange **loads** and **stores** to the same location
    2. **two stores** take place in the **correct order**

- **Formally:**
    - There is a **data dependence from statement S1 to statement S2 (S2 depends on S1)** if:
        - Both statements **access the same memory location** and at least **one of them stores** onto it, and
        - There is a **feasible run-time execution** path from **S1 to S2**

Load/Store Classification

- Dependences classified in terms of load-store order:

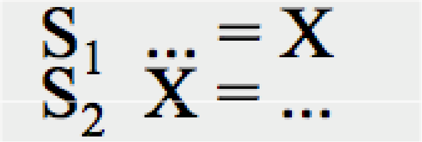
1. True dependences
   - S_2 depends on S_1 is denoted by $\mathbf{S_1\ \delta\ S_2}$



```
S1  X= ...
S2  ... = X
```

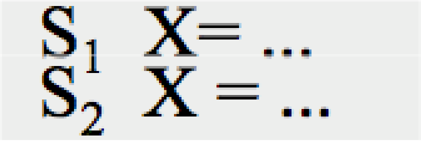
This is a crucial dependence!

2. Antidependence
   - S_2 depends on S_1 is denoted by $\mathbf{S_1\ \delta^{-1}\ S_2}$



```
S1  ... = X
S2  X = ...
```

3. Output dependence
   - S_2 depends on S_1 is denoted by $\mathbf{S_1\ \delta^{0}\ S_2}$



```
S1  X= ...
S2  X = ...
```

Data Dependence of Scalar Variables

- **True/Flow dependence**

```
a =
  = a
```

- Output dependence

```
a =
a =
```

- **Anti-dependence**

```
  = a
a =
```

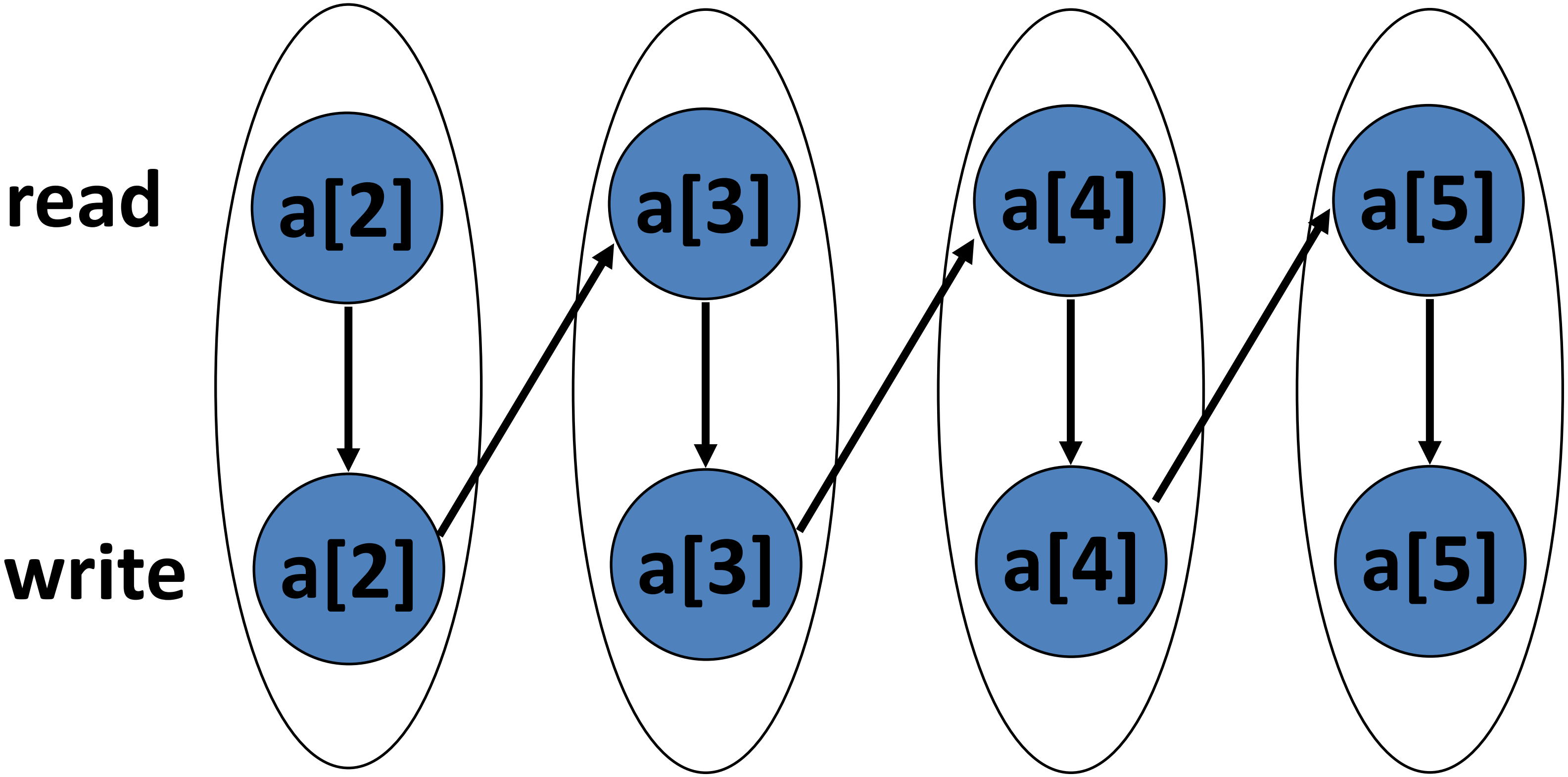
- Input dependence

```
= a
= a
```

- **Only data flow dependences are true dependences.**
- **Anti and output can be removed by renaming**

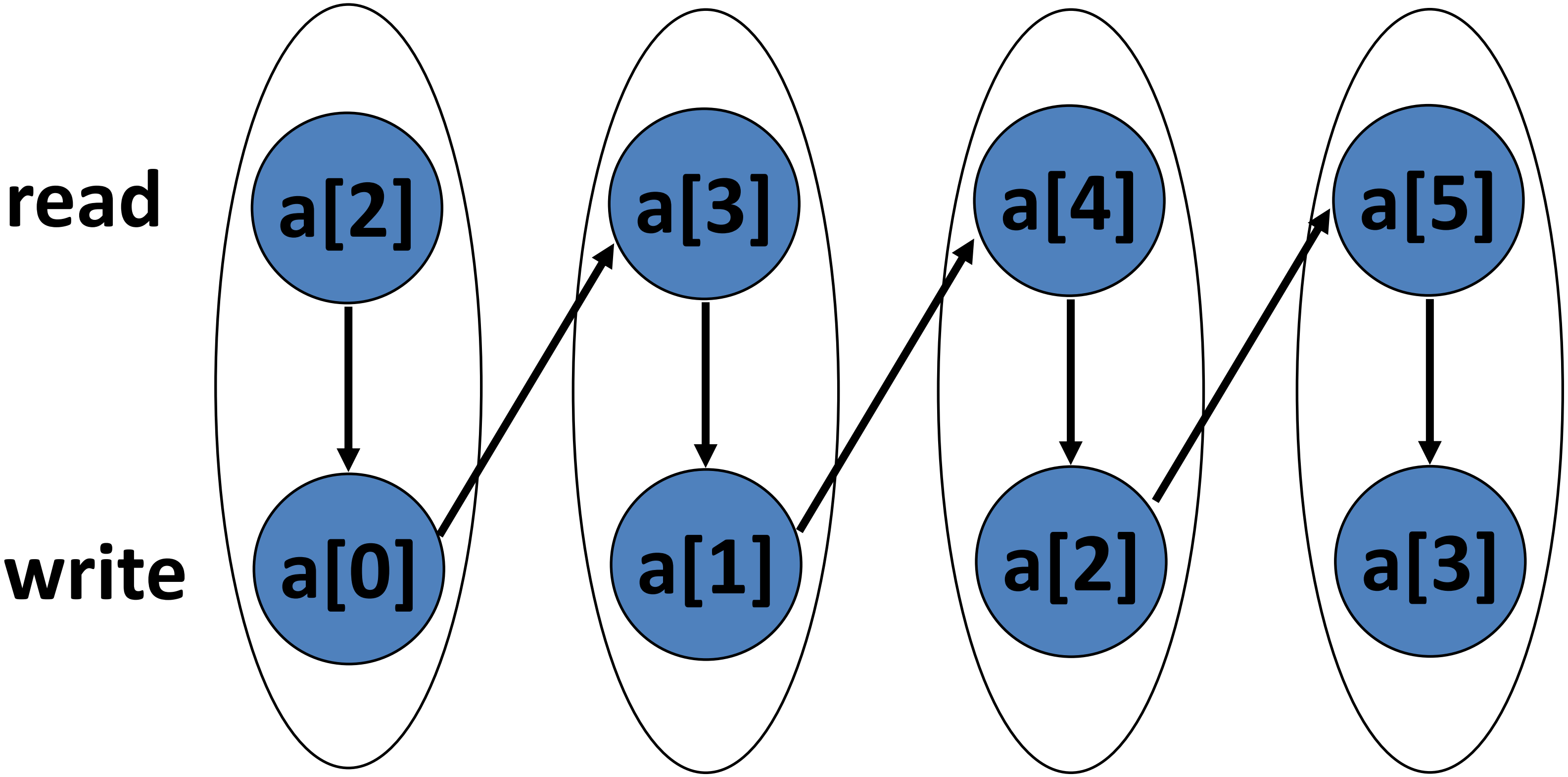
Array Accesses in a Loop

```
for i= 2, 5
    a[i] = a[i] + 3
```



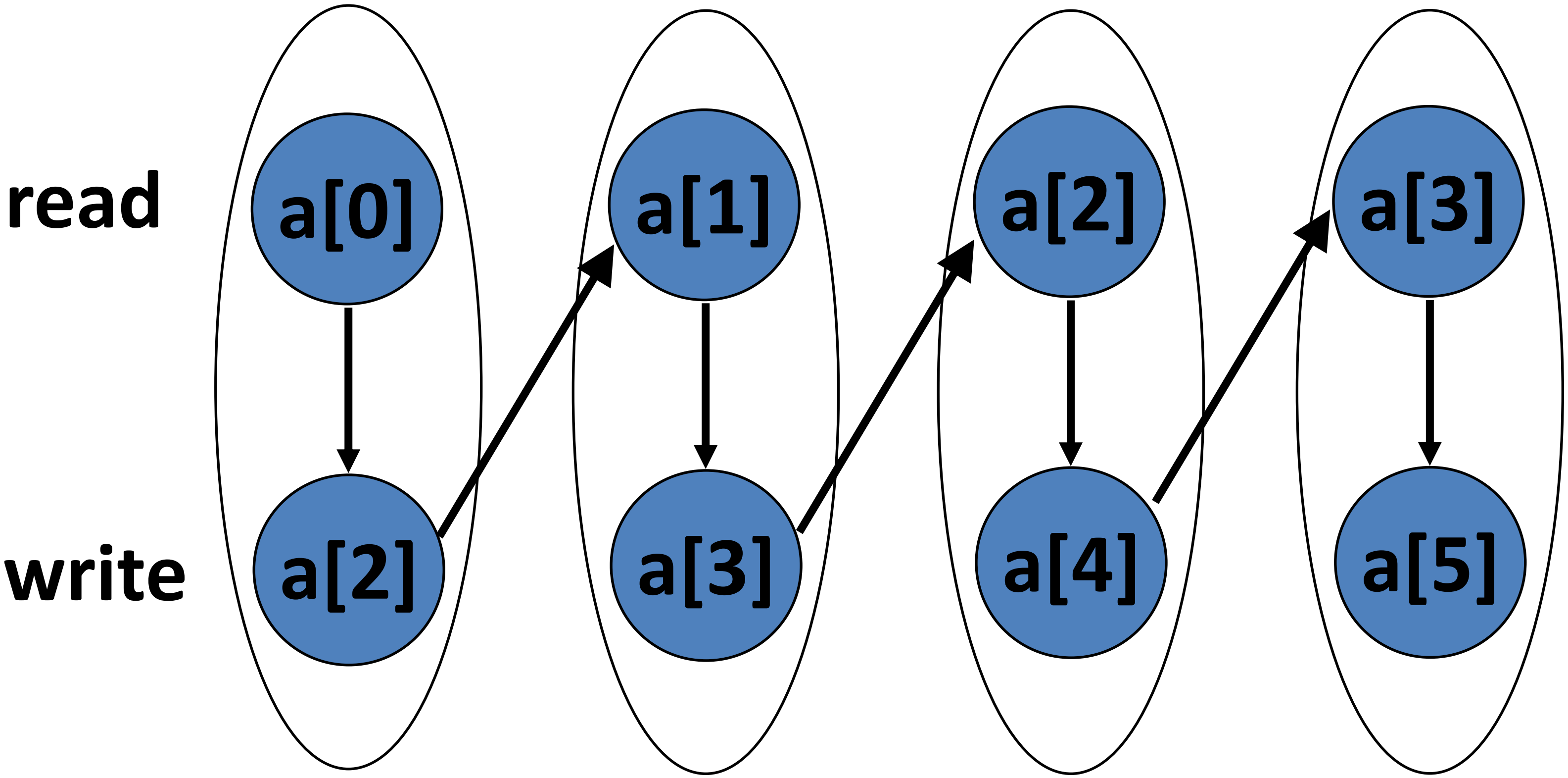
Array Anti-dependence

```
for i= 2, 5
   a[i-2] = a[i] + 3
```



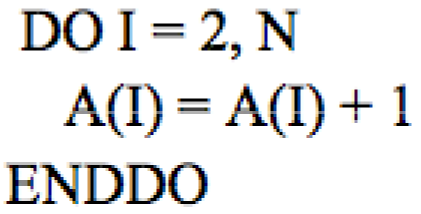
Array True-dependence

```
for i= 2, 5
   a[i] = a[i-2] + 3
```

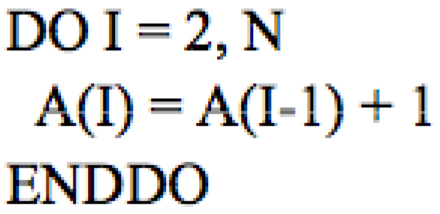


A Parallel DOALL Loop

- A **loop is fully parallel** if **no dependencies flow across iterations**:



```
DO I = 2, N
   A(I) = A(I) + 1
ENDDO
```



```
DO I = 2, N
  A(I) = A(I-1) + 1
ENDDO
```

- **Parallel loops** are found through **dependence analysis** and **dependence tests**

- Usually **done** at the **source-code level**, **focus is on arrays**

Recognizing DOALL Loops

- **Find** data **dependences** in loop
- **Definition:** a **dependence** is **loop-carried** if it **crosses an iteration boundary**
- **If there are no loop-carried dependences only then loop is parallelizable**

Example: Loop Parallelization

```
Do i=1,n
 A(i)=5*B(i)+A(i)
Enddo
```

```
Do i=1,n
 A(i-1)=5*B(i)+A(i)
Enddo
```

```
Do i=1,n
 tmp=5*B(i)
 A(i)=tmp
Enddo
```

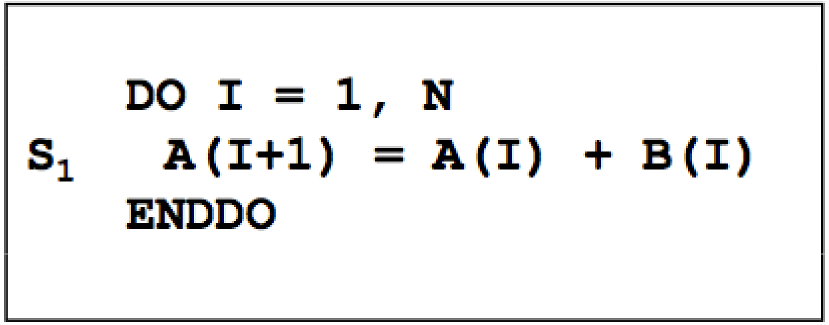
Which of the following loops are parallelizable?

Dependence in Loops

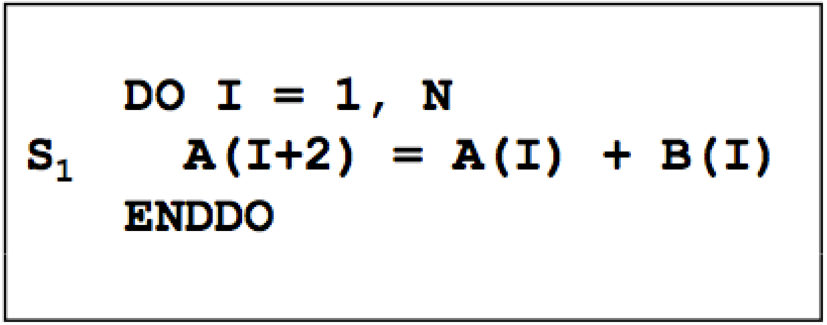
1: A(2) = A(1) + B(1)
2: A(3) = A(2) + B (2)
3: A(4) = A(3) + B(3)
S1 &S1

1: A(3) = A(1) + B(1)
2: A(4) = A(2) + B(2)
3: A(5) = A(3) +B(3)
4: A(6) = A(4) + B(4)

- Let us look at two different loops:



```
     DO I = 1, N
S1     A(I+1) = A(I) + B(I)
     ENDDO
```



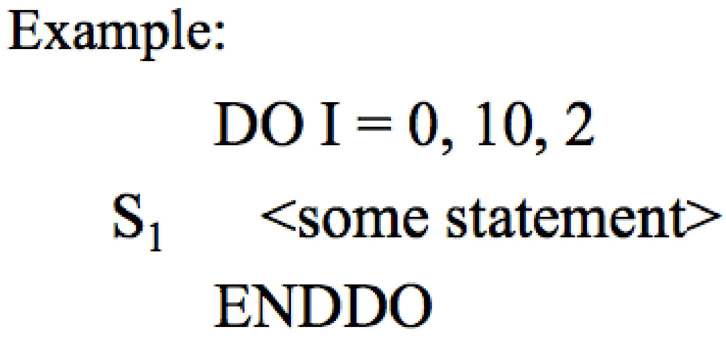
```
     DO I = 1, N
S1     A(I+2) = A(I) + B(I)
     ENDDO
```

- In both cases, statement S_1 depends on itself
- However, there is a significant difference.
- We need a formalism to describe and distinguish such dependences

Iteration Numbers

- The **iteration number** of a **loop** is **equal** to the **value** of the **loop index**

- **Definition:**
  - For an arbitrary loop in which the **loop index *I*** runs from **L** to **U** in **steps of S**, the iteration number *i* of a specific iteration is equal to the **index value *I*** on that **iteration:**



Example:

```
        DO I = 0, 10, 2
    S1      <some statement>
        ENDDO
```