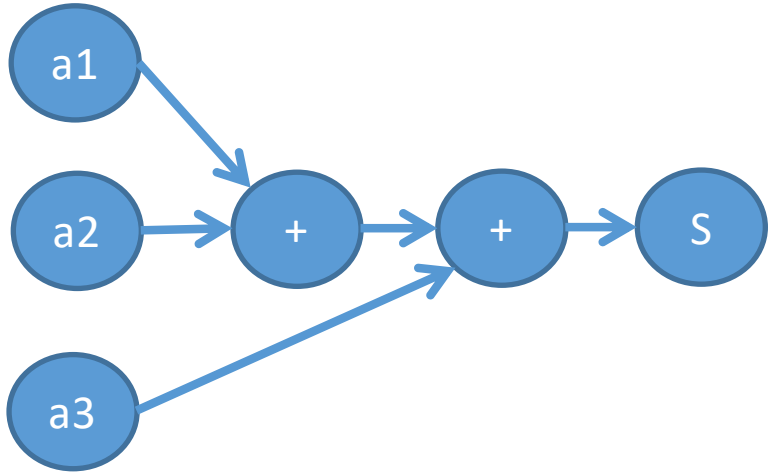
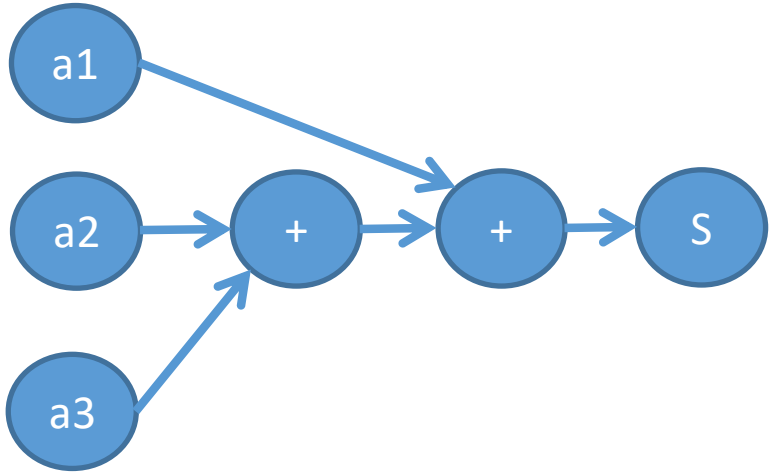
Циклы

Ярусно-параллельная форма

S = (a1 + a2) + a3

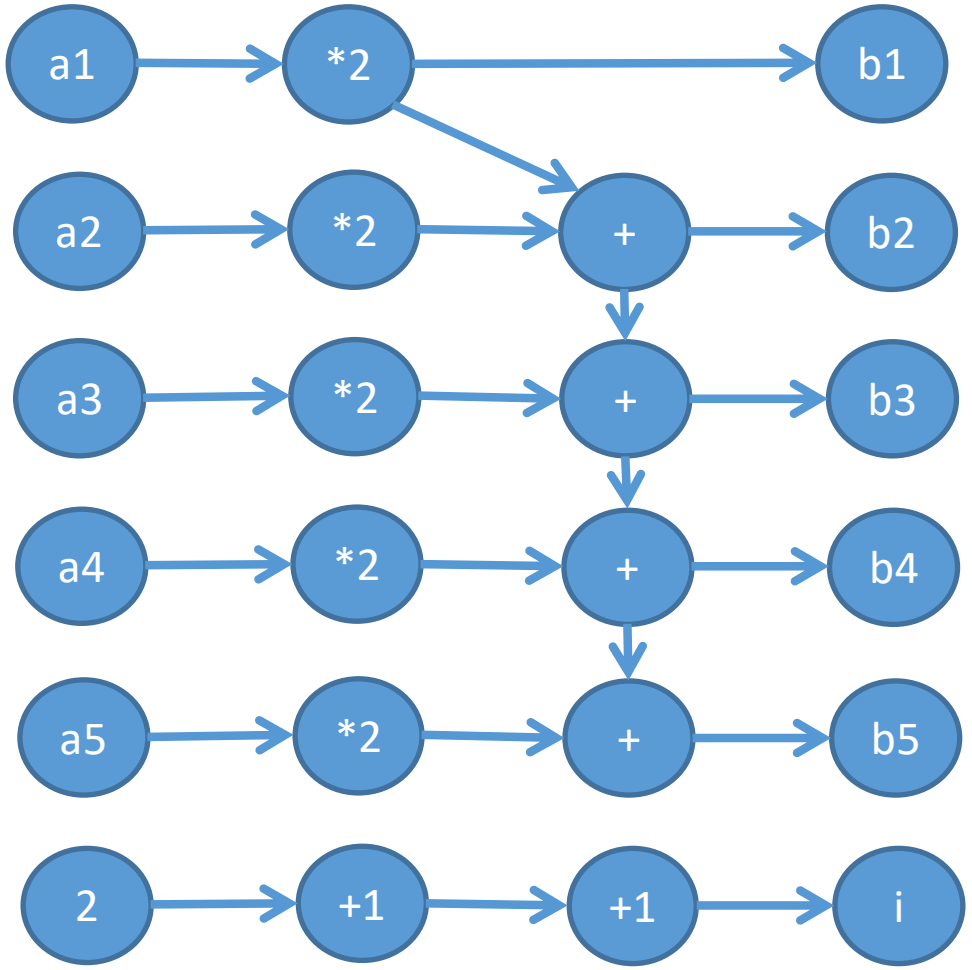


S = a1 + (a2 + a3)

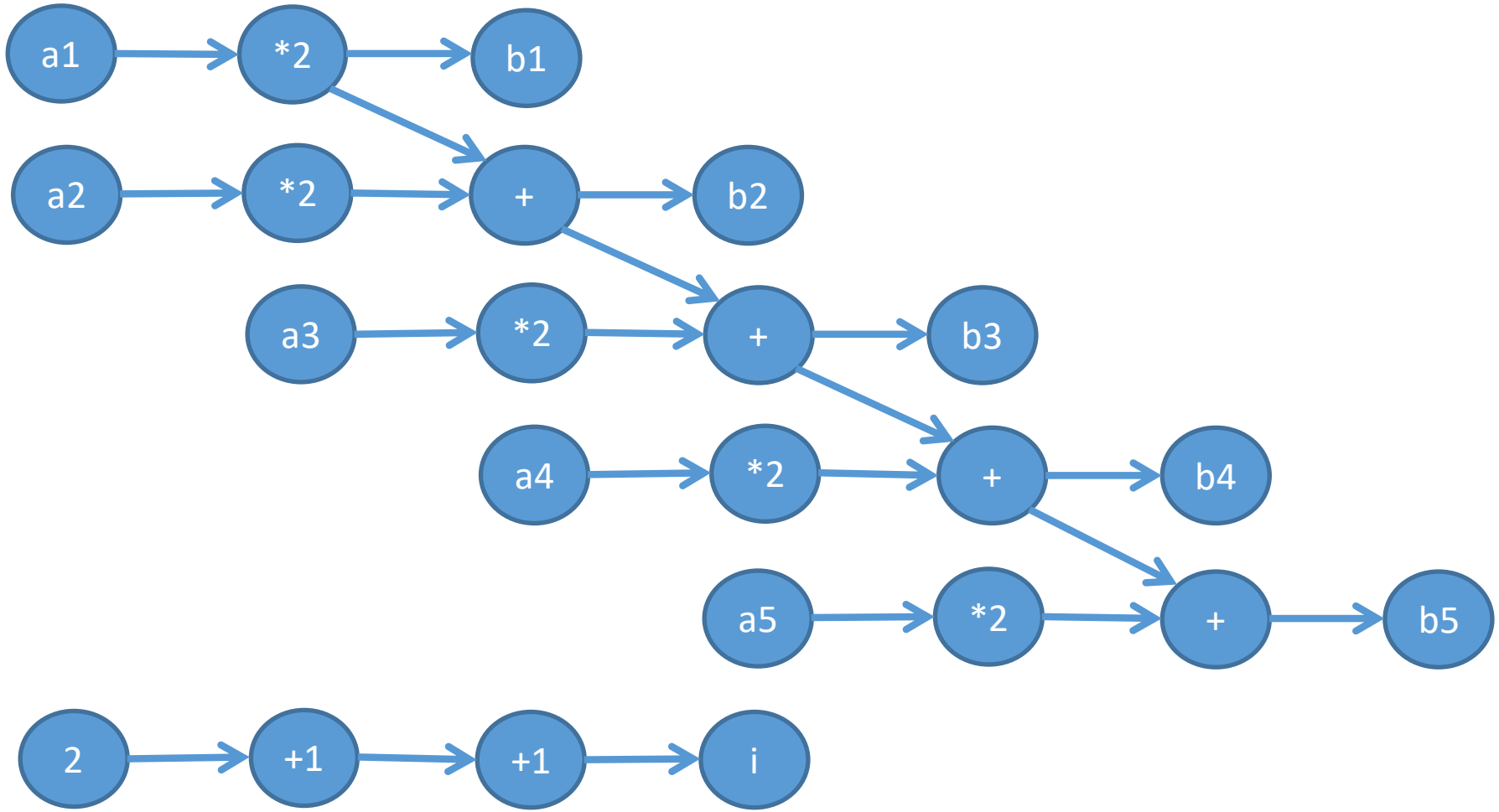


Циклы

```
b(1) = a(1)*2
do i = 2 , 5
   b(i) = b(i-1) + a(i)*2
enddo
```



Циклы



Циклы

P:

x = 2*u

y = x – 1

Q:

x = w + x

y = u*x

Входные элементы:

R(P) = {u, x}

R(Q) = {w, x, u}

Выходные элементы:

W(P) = {x, y}

W(Q) = {x, y}

Условие Бернстайна:

Пересечения:

1) W(P) и W(Q)

2) W(P) и R(Q)

3) R(P) и W(Q)

Пусты => Выполнение P и Q детерминировано.

Циклы

P:

x = 2*u

y = t – 1

Q:

z = t + u

P:

x = 2*u

y = t – 1

Q:

x = t + u

P:

x = 2*u

y = t – 1

Q:

z = t + x

P:

x = 2*u

y = t – 1

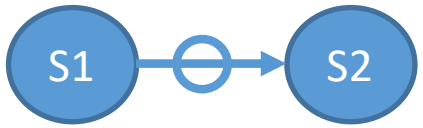
Q:

u = t + z

Циклы

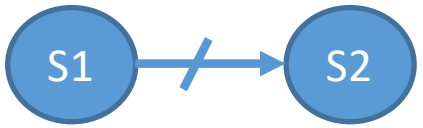
1)W(S1) и W(S2)
output dependence

S1: x = 2*y + z
S2: x = a – b



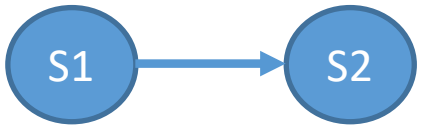
2) R(S1) и W(S2)
anti-dependence

S1: x = 2*y + z
S2: y = a – b



3) W(S1) и R(S2)
(true) dependence

S1: x = 2*y + z
S2: b = a – x



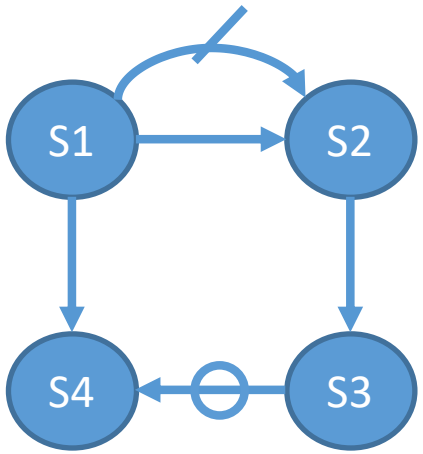
Циклы

S1: a = 2*b + 15

S2: b = a + 10*x

S3: d = b – c

S4: d = a/c



Циклы

SSA форма

```
x = b – c       x1 = b1 – c1
x = x + a       x2 = x1 + a1
```

```
x1 = y0 * z0

If (x1 > 0)
        y1 = z0
else
        y2 = x1

y3 = phy(y1, y2)
z1 = x1 * y3
```

```
int i1 = 0
int res1 = 0
while (phy(i1, i3) < x0)
        res2 = phy(res1, res3)
        res3 = re2 + i1
        i2 = phy(i1, i3)
        i3 = i2 + 1
```

Циклы

Зависимость по управлению

S1: x = 2*y + z

S2: y = (x > 0)? a – b : a + b

Зависимость по ресурсам

S1: x = 2*y/z

S2: x = a/b

Циклы

Расстояние зависимости D

```
do i=1, N
S1:     A[f(i)] = ….          // Source (исток) = i
S2:     … = … A[g(i)] …       // Sink (сток) = I’
```

f(i) = g(i’)

D = Sink – Source

$S_1^{source} \delta S_2^{sink}$

```
for(int i = 3; i < SIZE; i++)
        A[i] = ….
        … = A[i - 3]
```

f(i) = i
g(i‘) = i’ – 3
i = i‘ – 3
D = i‘ – i = 3

```
for(int i = 0; i < SIZE - 3; i++)
        A[i + 3] = ….
        … = A[i]
```

Циклы

```
for(int i = 0; i < SIZE; i++)
    A[i - 3] = ….
    … = A[i]
```

D < 0 – анти-зависимость

```
for(int i = 0; i < SIZE; i++)
    A[i] = ….
    … = A[i]
```

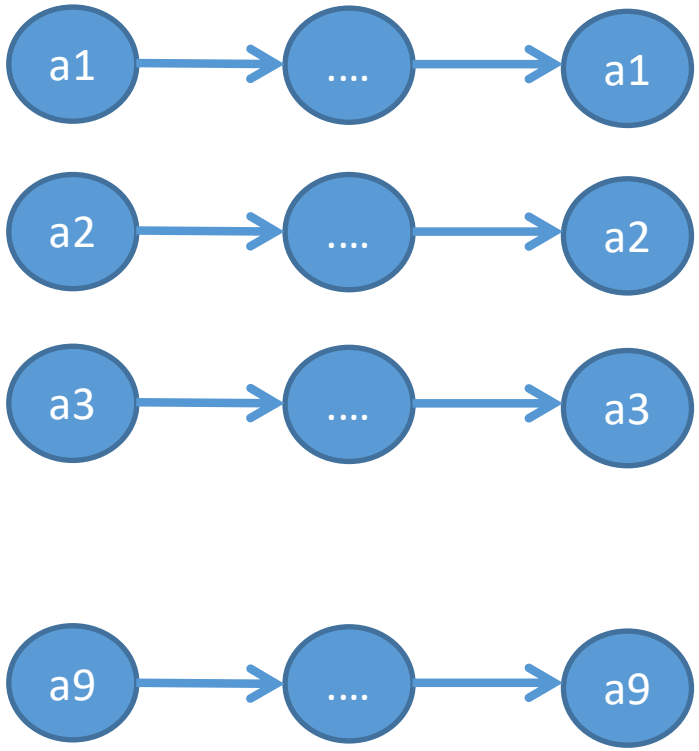
D = 0 – отсутствует зависимость

D > 0 – истинная зависимость

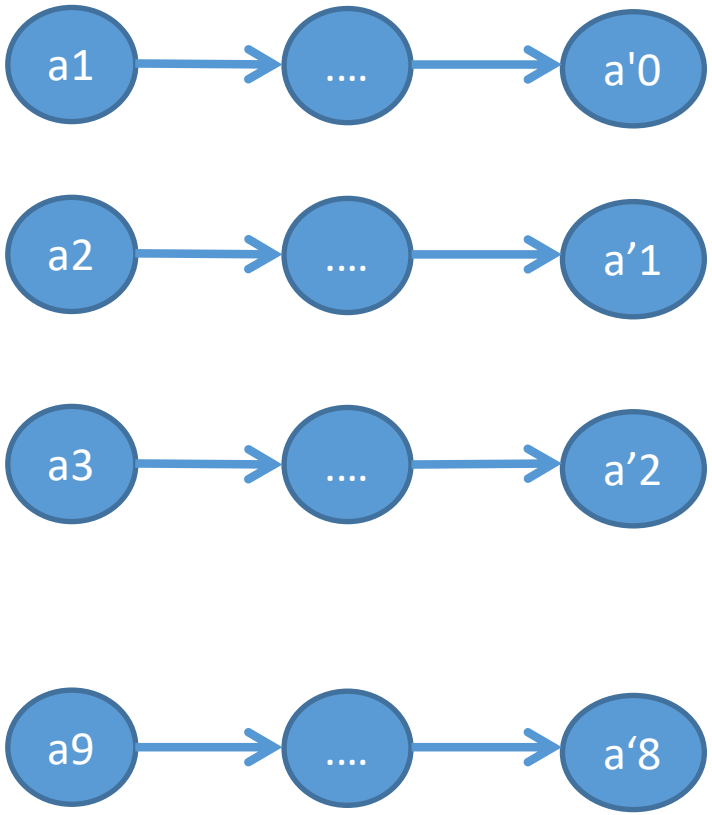
Можно распараллелить на D исполнителях

```
for(int i = 3; i < SIZE; i++)
    A[i + 3] = ….
    … = A[i]
```

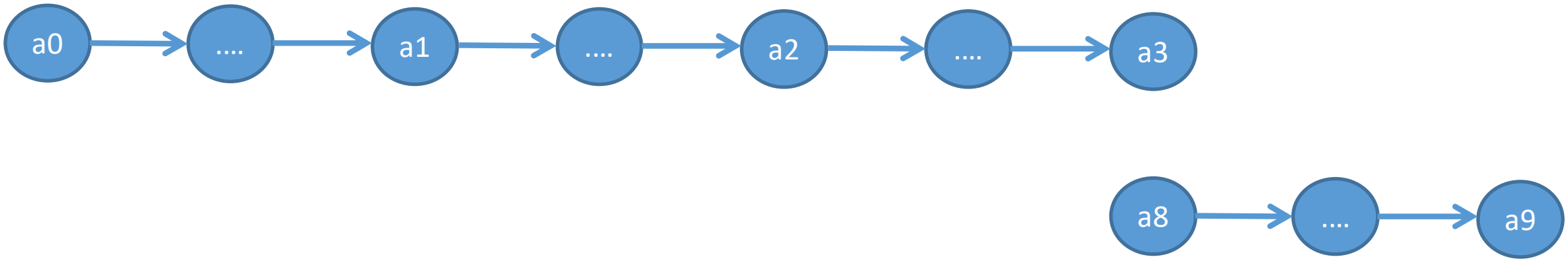
```
for(int i = 1; i < SIZE; i+=2)
        A[i] = …. A[i]
```



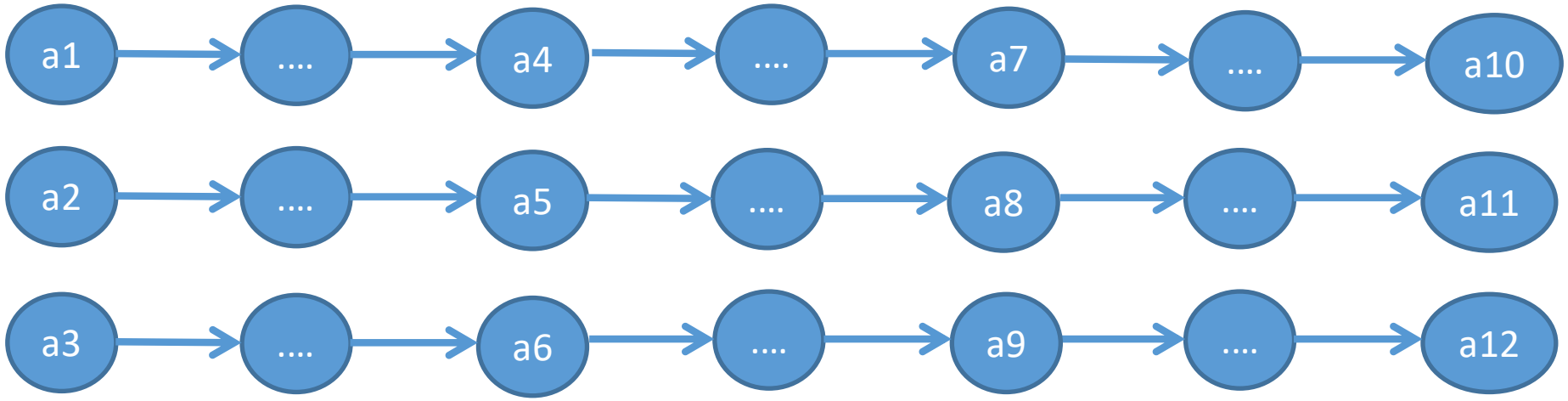
for(int i = 1; i < SIZE; i+=2)
A’[i-1] = …. A[i]



```
for(int i = 1; i < SIZE; i+=2)
        A[i+1] = …. A[i]
```



```
for(int i = 1; i < SIZE; i+=2)
        A[i+3] = …. A[i]
```



Циклы

Вектор расстояний **D**

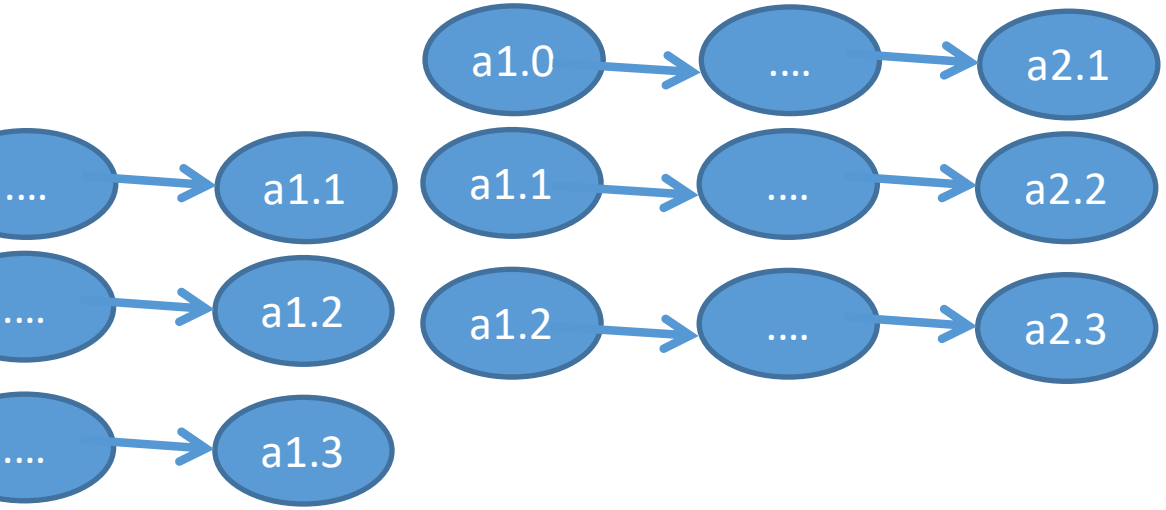
```
for(int y = 3; y < YSIZE; y++)
for(int x = 3; x < XSIZE; x++)
        A[y, x] = ….
        … = A[y - 1, x - 3]
```

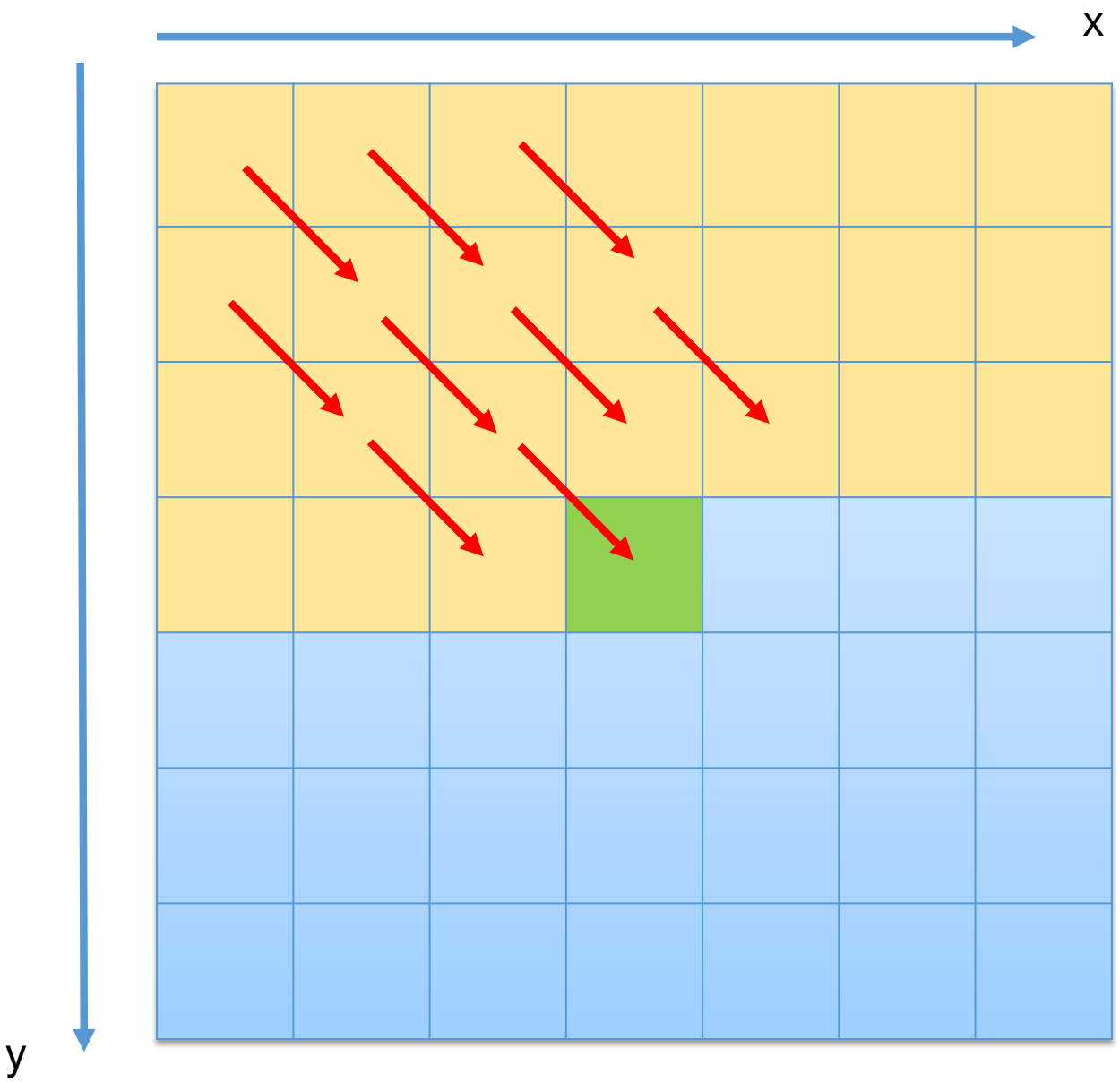
```
do y=1, N
do x=1, N
S1:     A[f1(y),f2(x)] = ….          // Source (исток) = (y, x)
S2:     … = … A[g1(y),g2(x)] …       // Sink (сток) = (y’, x’)
```

(f1(y), f2(x)) = (g1(y’), g2(x’))

D = **Sink** – **Source**

```
for(int y = 0; y < YSIZE; y++)
for(int x = 0; x < XSIZE; x++)
        A[y + 1, x + 1] = ….A[y, x]
```





```
for(int y = 0; y < YSIZE; y++)
for(int x = 0; x < XSIZE; x++)
        A[y - 1, x + 1] = ….A[y, x]
```

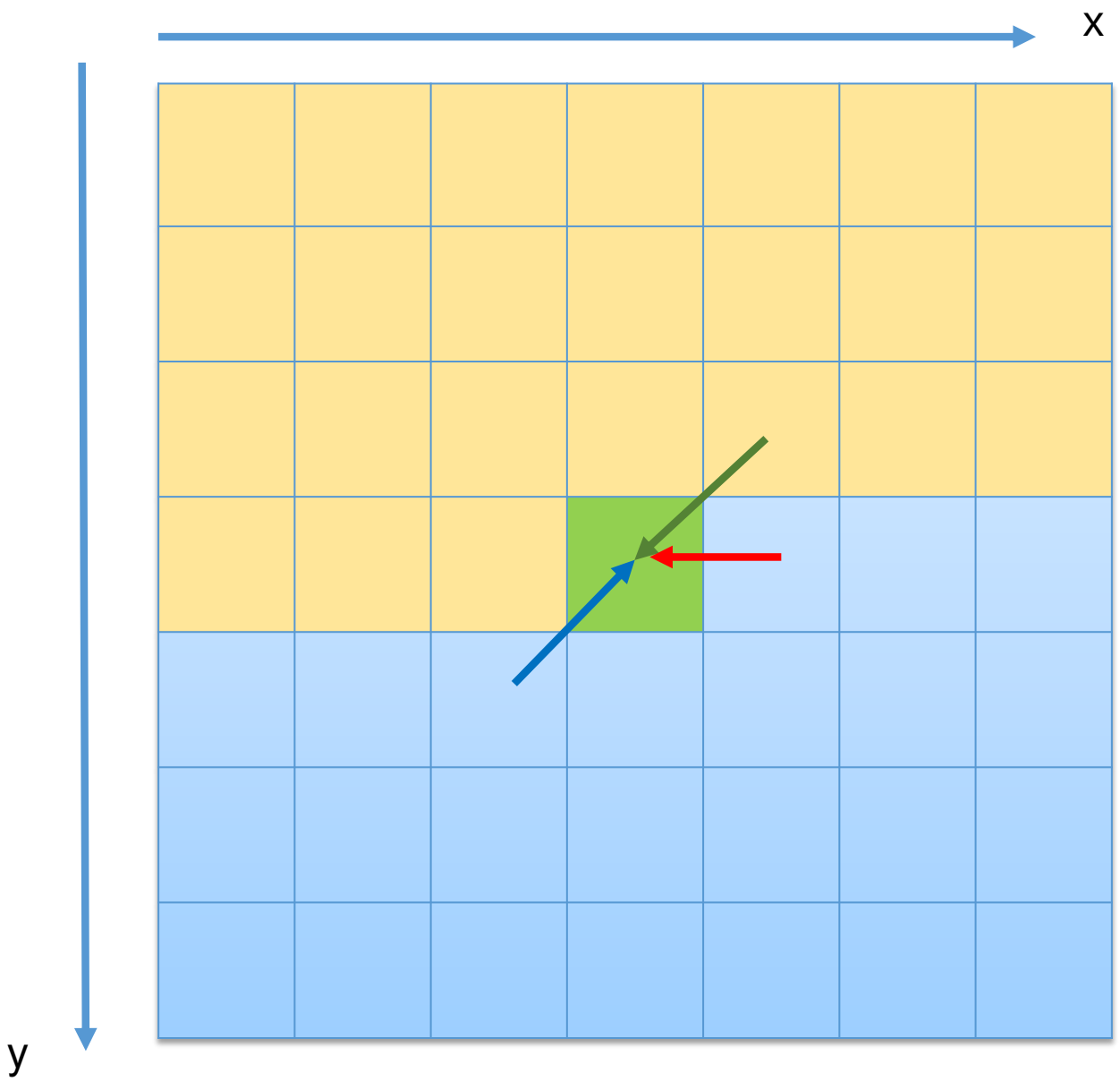
(-1;1) - антизависимость

```
for(int y = 0; y < YSIZE; y++)
for(int x = 0; x < XSIZE; x++)
        A[y + 1, x - 1] = ….A[y, x]
```

(1;-1) - истинная

```
for(int y = 0; y < YSIZE; y++)
for(int x = 0; x < XSIZE; x++)
        A[y, x - 1] = ….A[y, x]
```

(0;-1) - антизависимость



Циклы

Вектор направлений

d = “=“, если D = 0 (нет зависимости)

d = “>“, если D < 0 (анти-зависимость)

d = “<“, если D > 0 (истинная зависимость)

(0;1; 5; 3; -2)
(=;<;<;<;>)