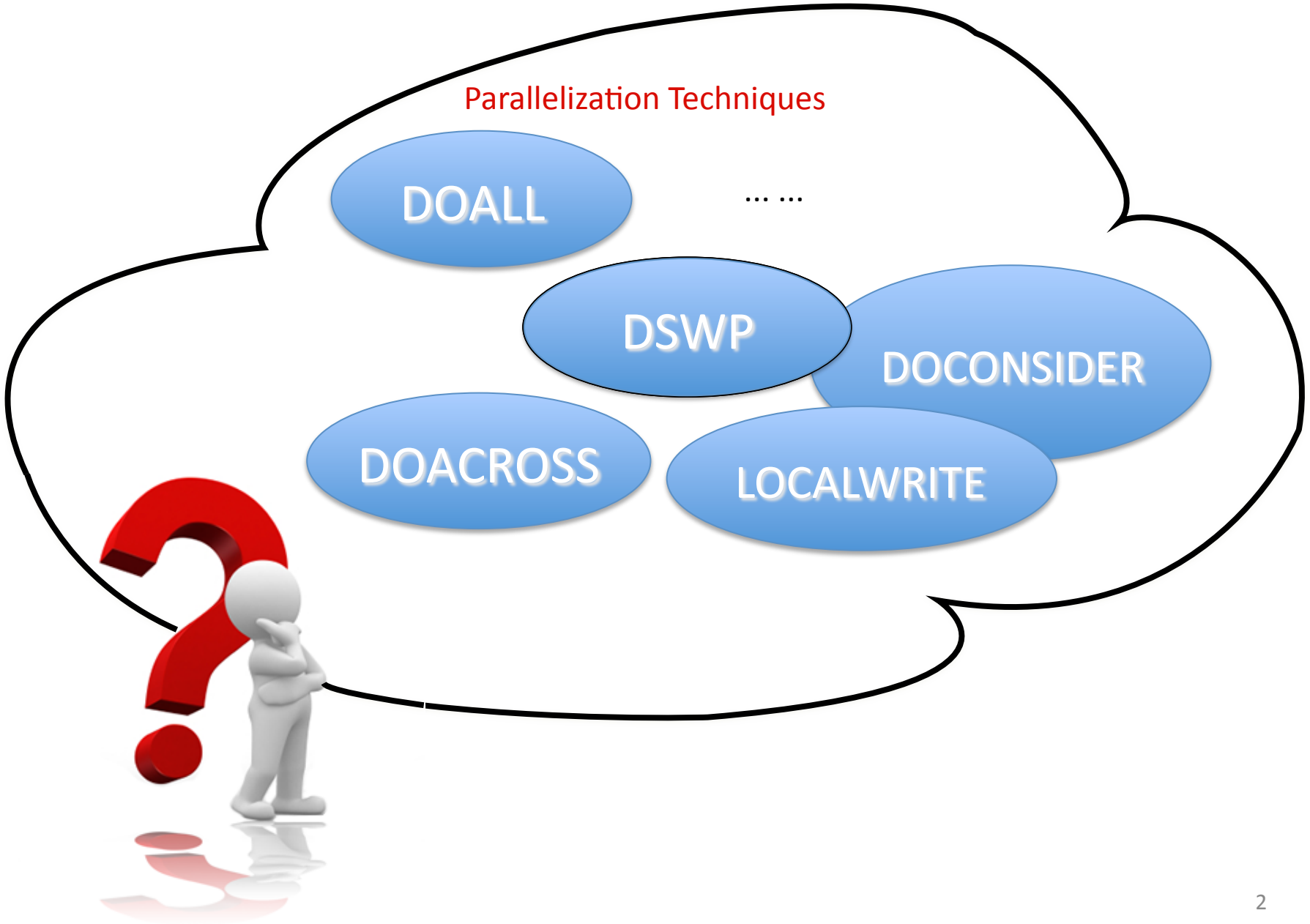
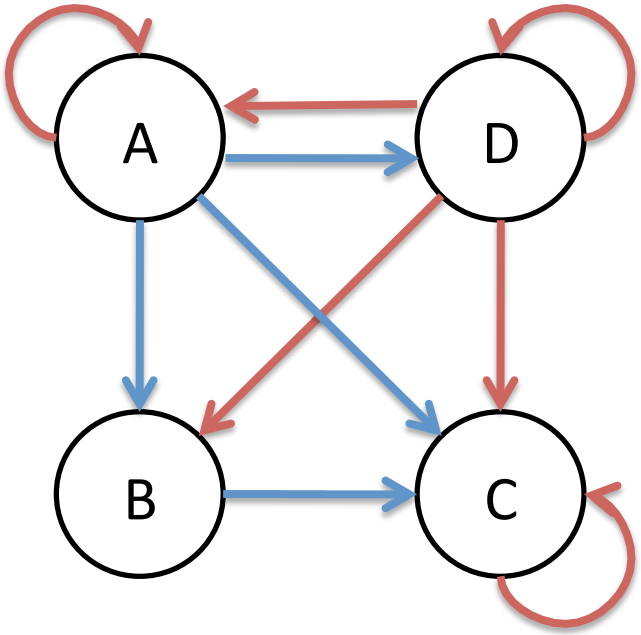
Enabling Automatic Parallelization

Jialu Huang

2010-12-02


Parallelization Techniques
DOALL
… …
DSWP
DOCONSIDER
DOACROSS
LOCALWRITE

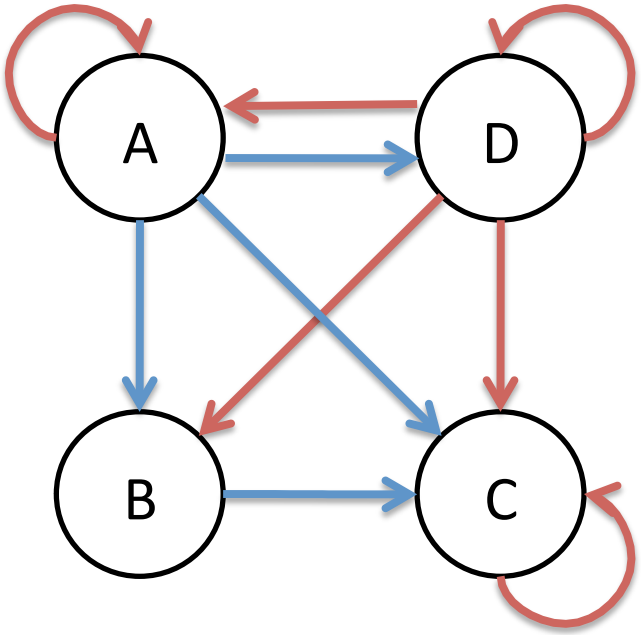
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



intra-loop dependence

inter-loop dependence

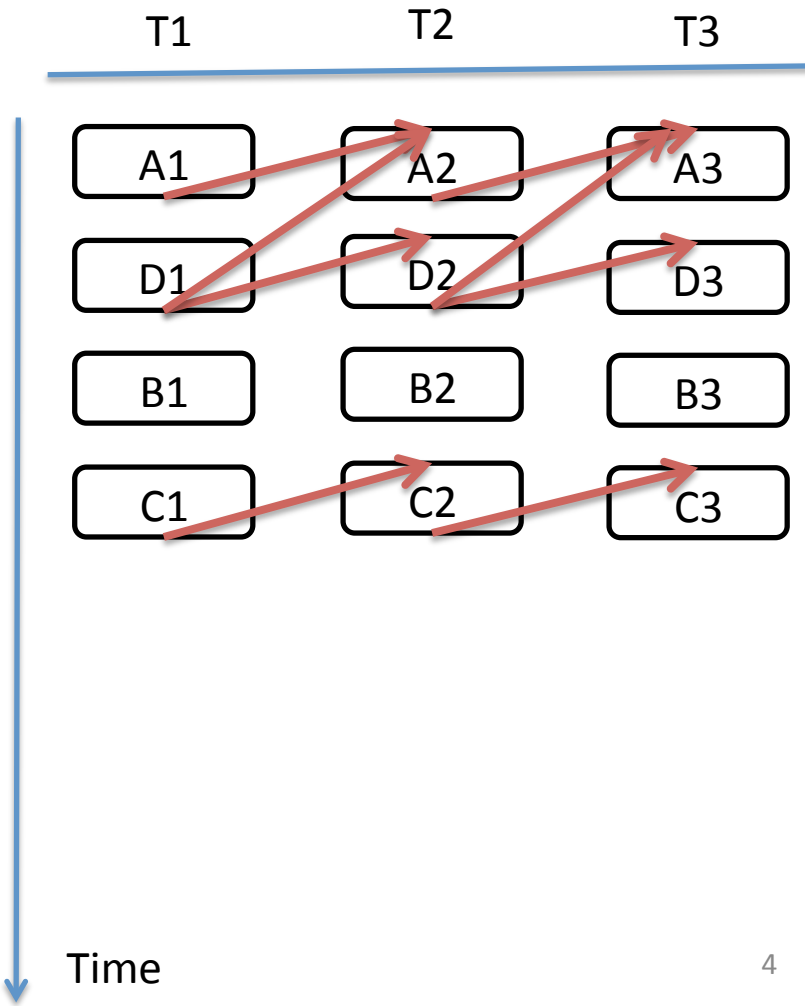
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



intra-loop dependence

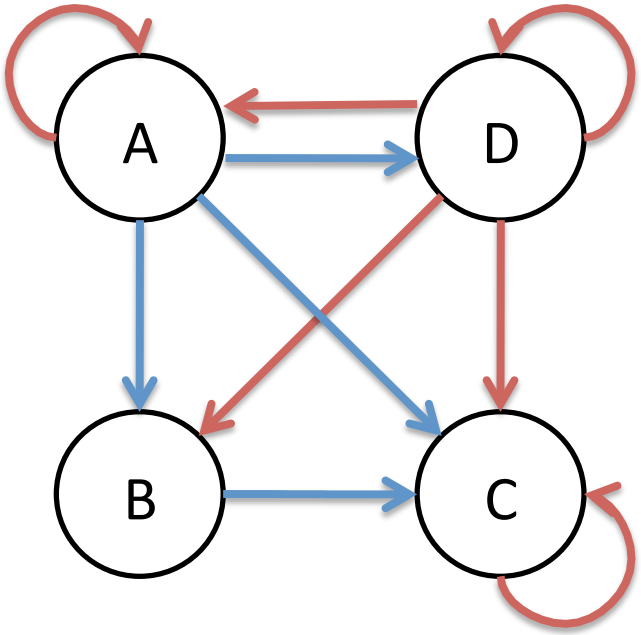
inter-loop dependence

DOALL



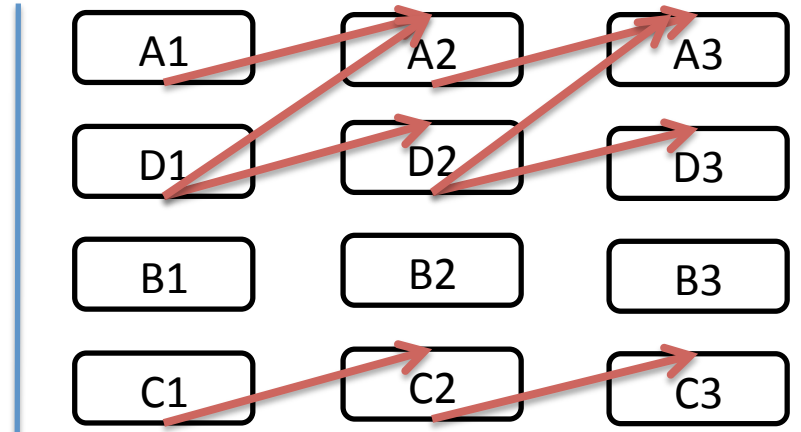
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```

~~DOALL~~

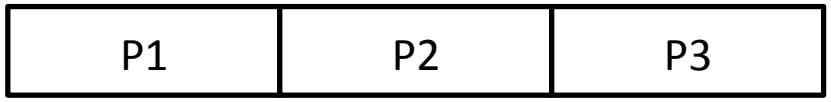


intra-loop dependence

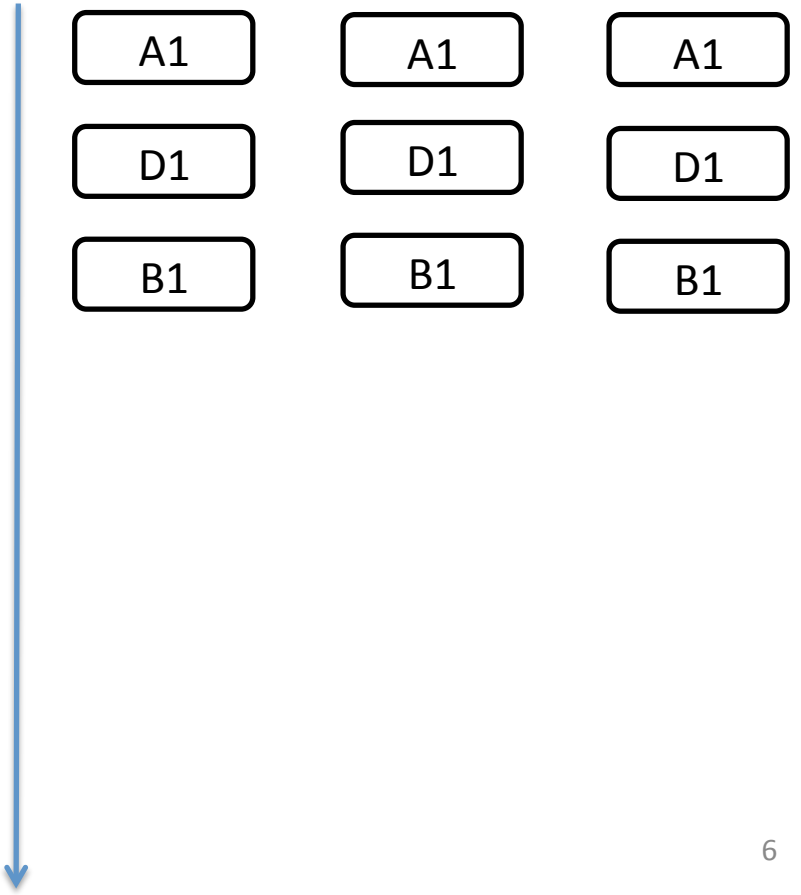
inter-loop dependence



```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```

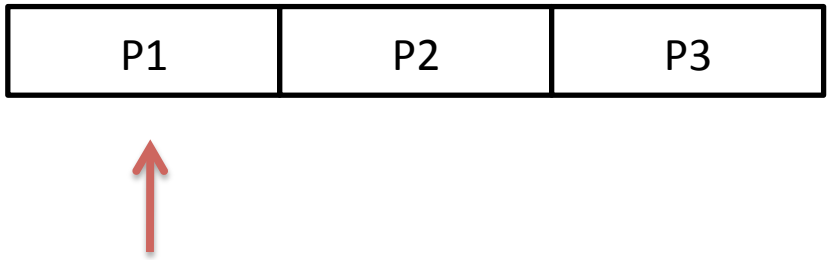


LOCALWRITE

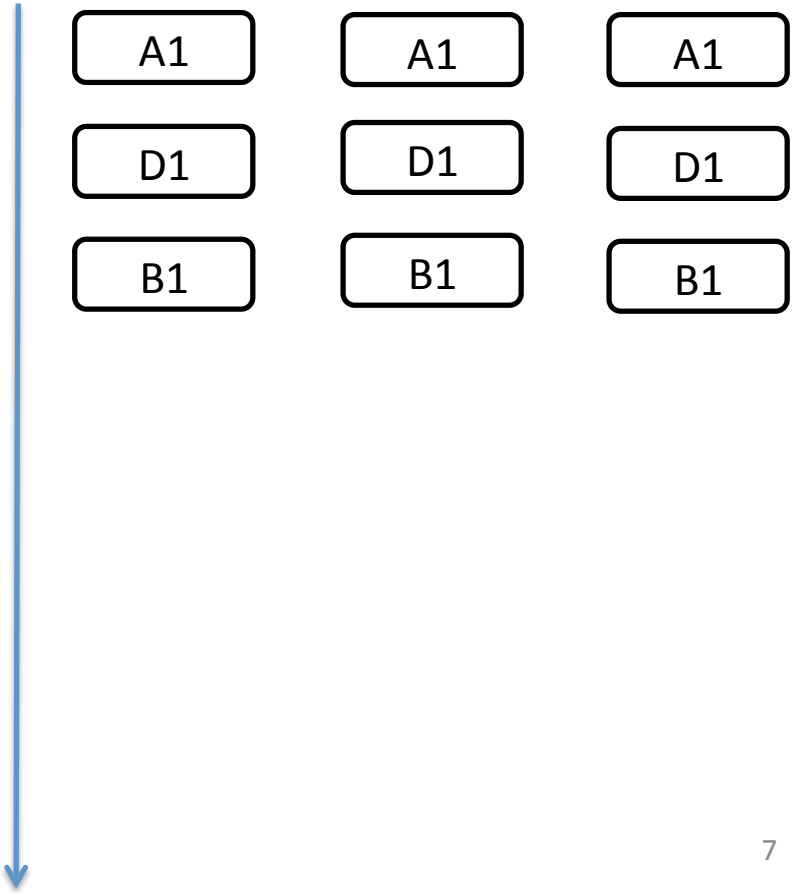


Han and Tseng [LCPC '98]

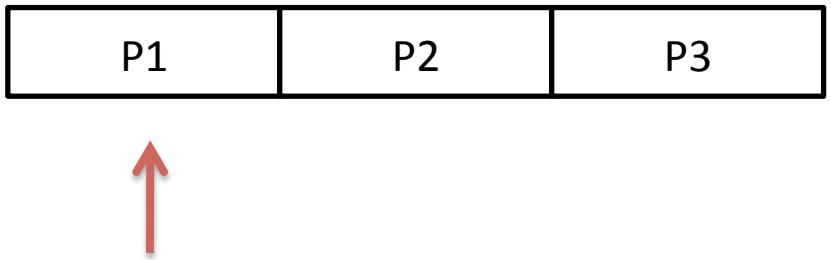
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



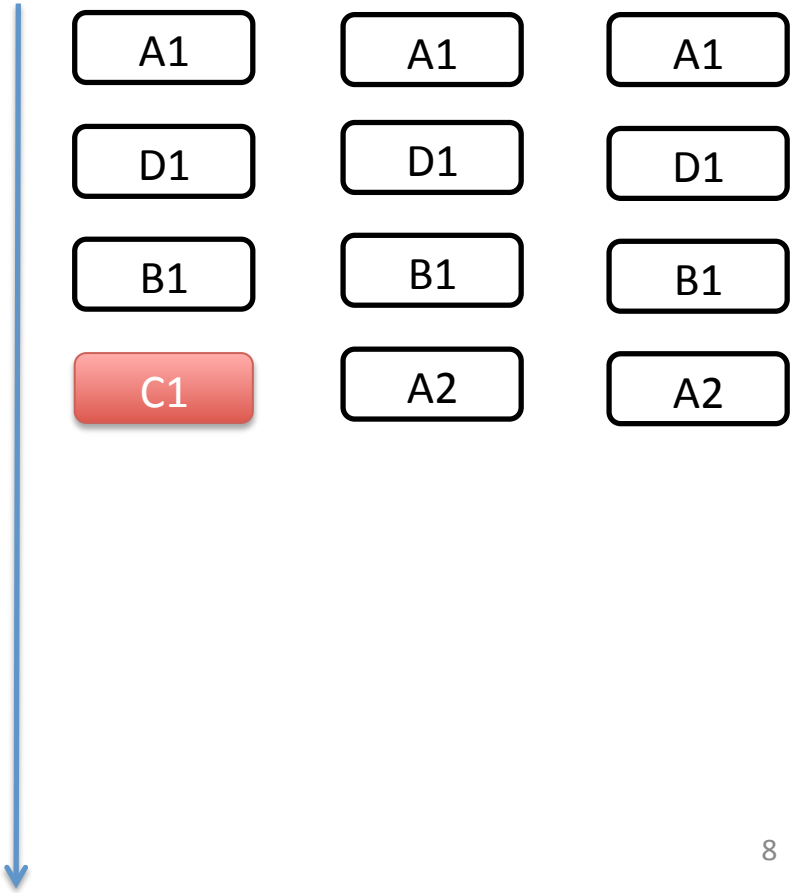
LOCALWRITE



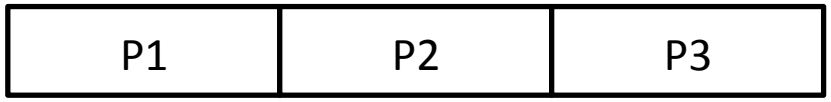
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



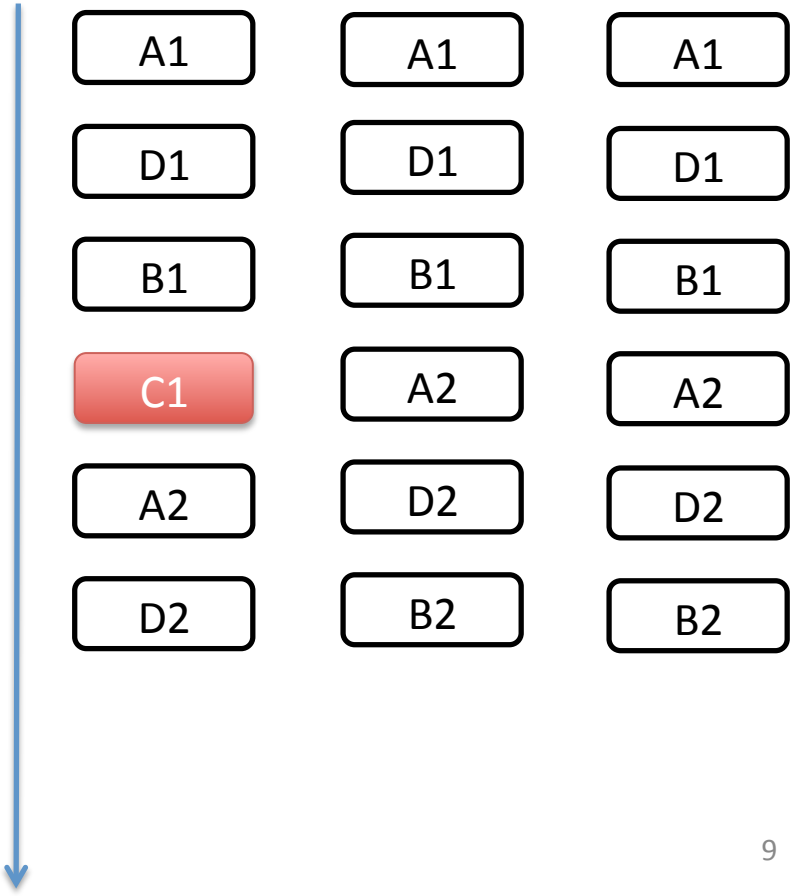
LOCALWRITE



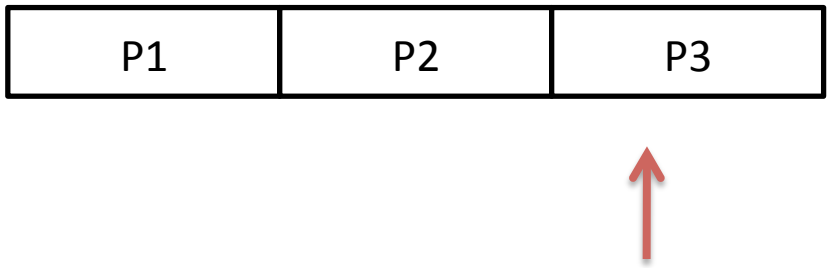
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



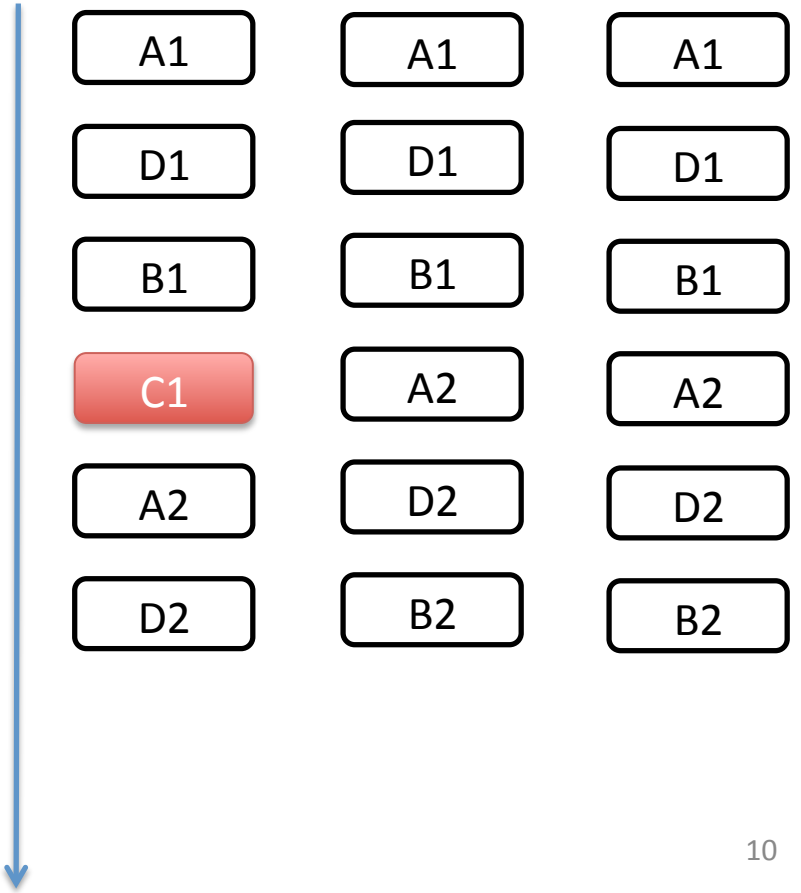
LOCALWRITE



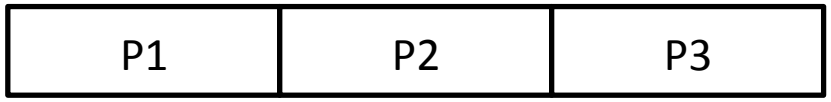
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



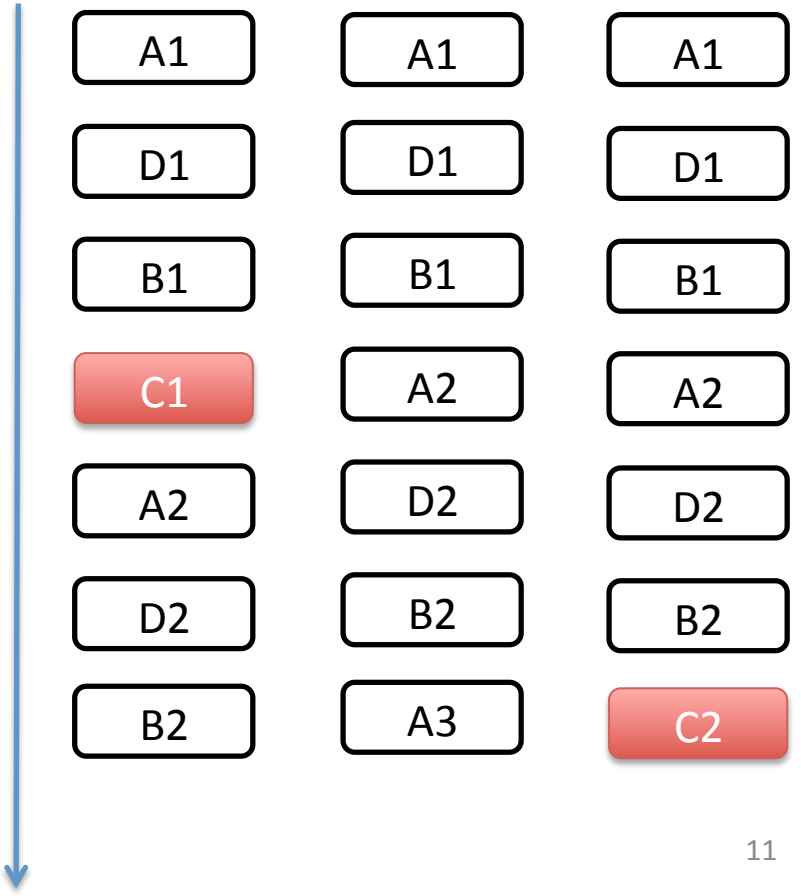
LOCALWRITE



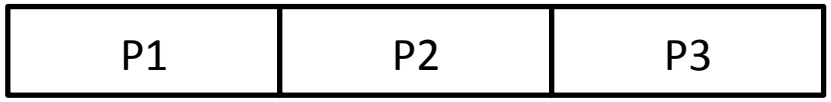
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



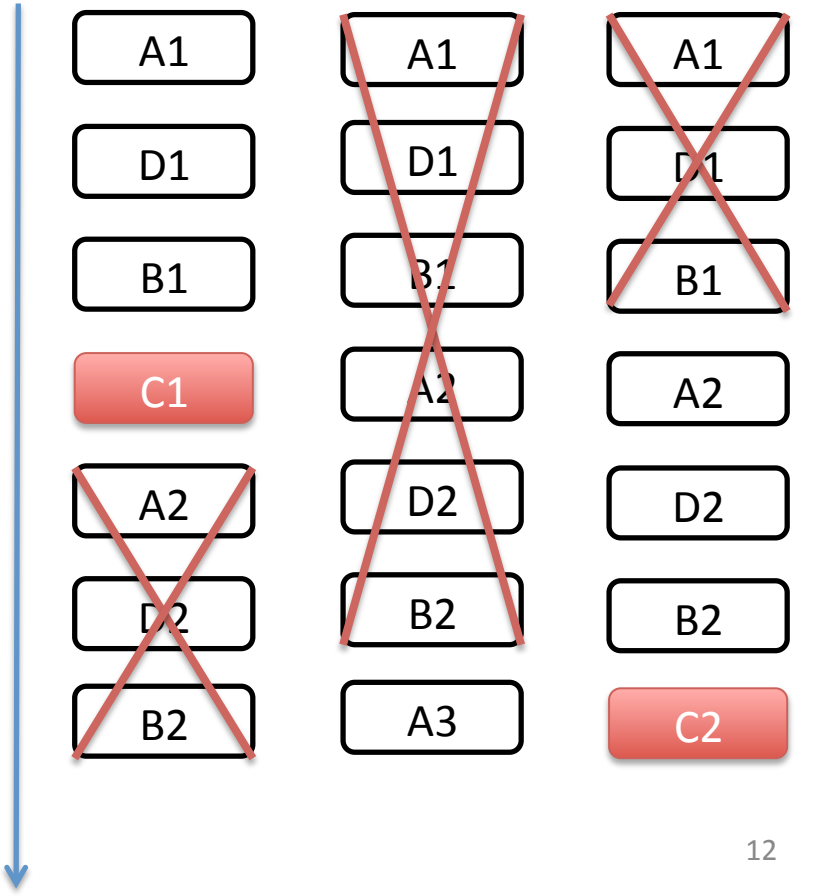
LOCALWRITE



```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



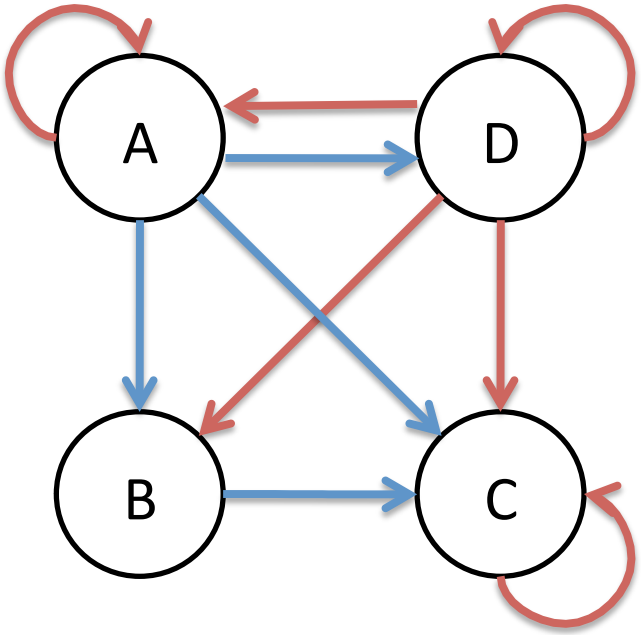
LOCALWRITE



DSWP+

Original Loop:

```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```

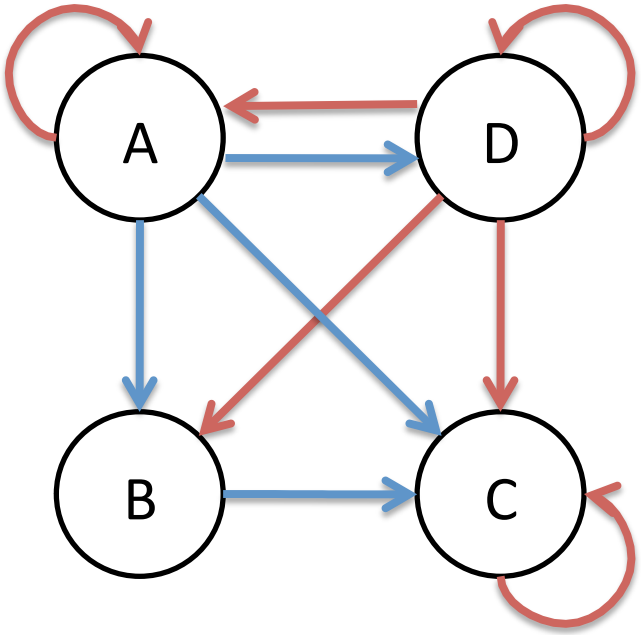


intra-loop dependence

inter-loop dependence

Original Loop:

```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```

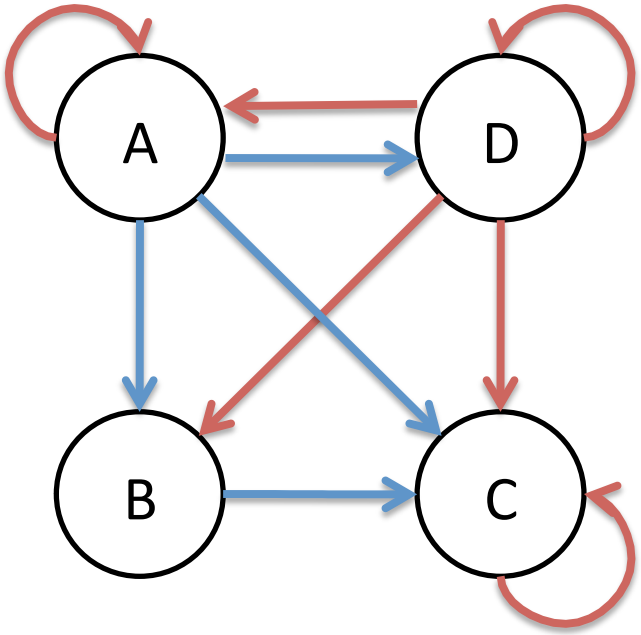


intra-loop dependence

inter-loop dependence

Original Loop:

```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```

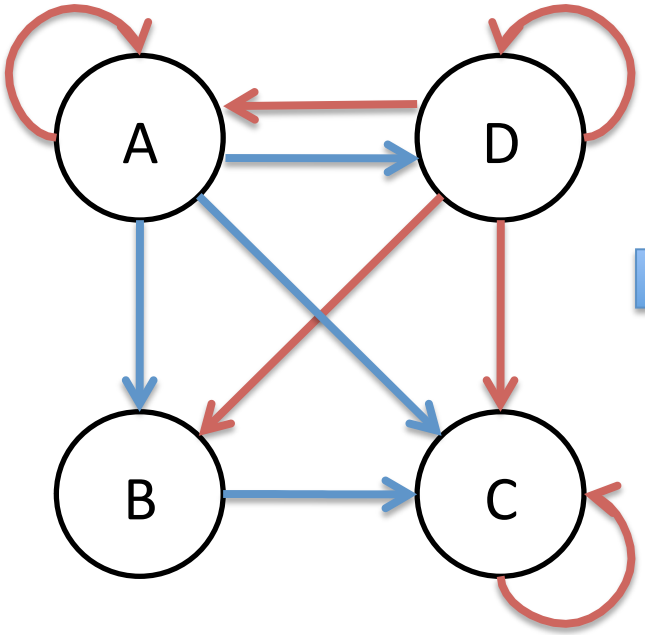


intra-loop dependence

inter-loop dependence

Original Loop:

```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



intra-loop dependence

inter-loop dependence

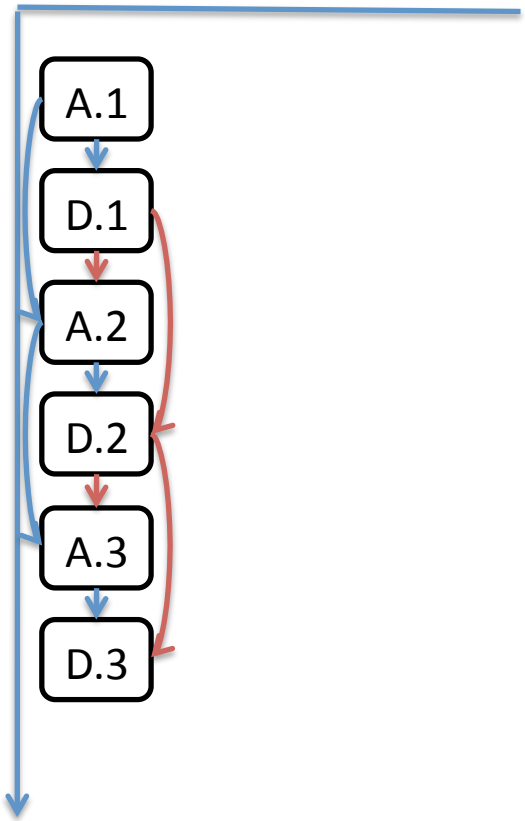
```
   node = list->head;
A: while (node != NULL) {
D:   node = node->next;
}
```

```
  while (TRUE) {
E:  node = getNodeOrExit();
B:  index = calc
      (node->data);
  }
```

```
  while (TRUE) {
F: node = getNodeOrExit();
G: index = getIndex();
C: density[index] =
     update_density
     (density[index], node->data);
 }
```

Sequential

```
   node=list->head;
A: while(node!=NULL){
D:   node=node->next;
}
```

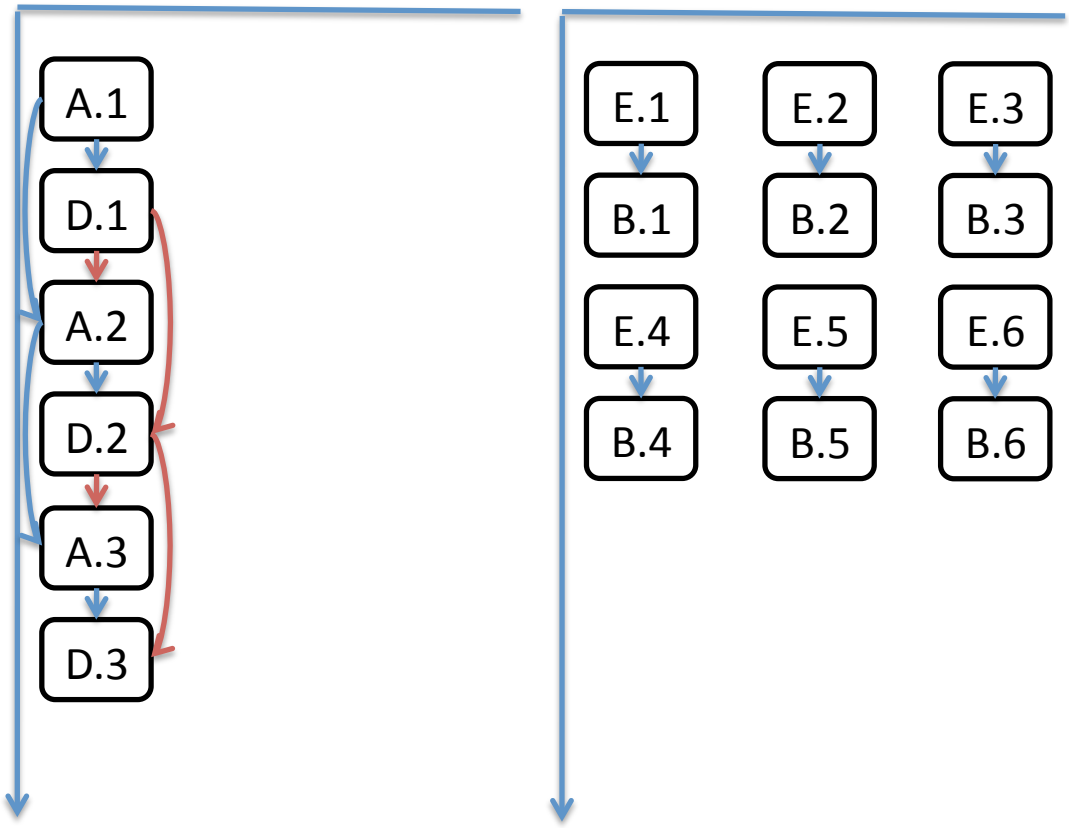


Sequential

```
   node=list->head;
A: while(node!=NULL){
D:   node=node->next;
}
```

DOALL

```
  while(TRUE){
E:  node=getNodeOrExit();
B:  index=calc
      (node->data);
  }
```



Sequential

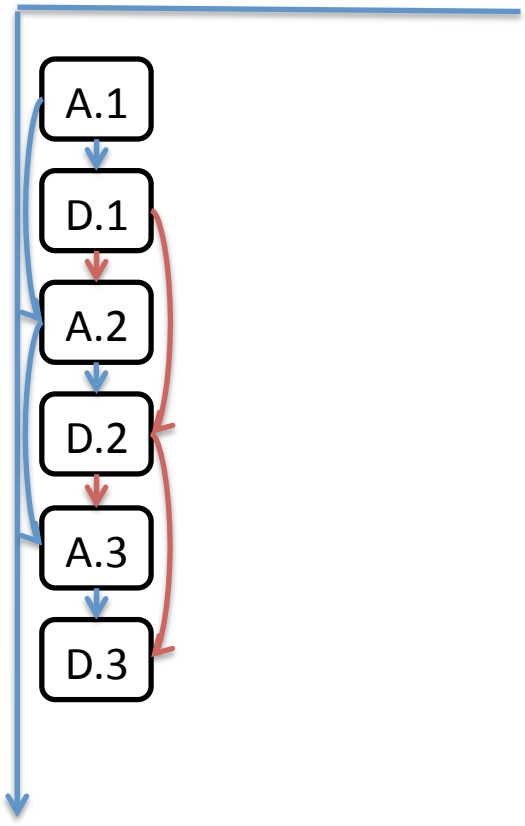
```
   node=list->head;
A: while(node!=NULL){
D:   node=node->next;
}
```

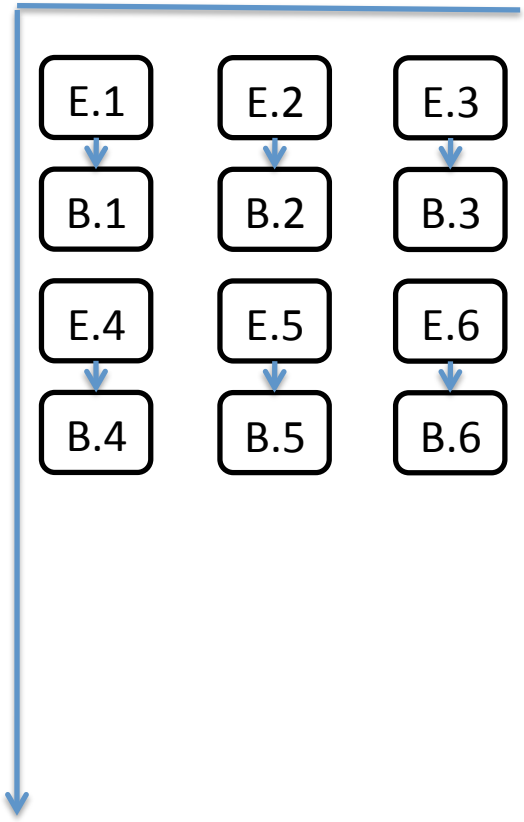
DOALL

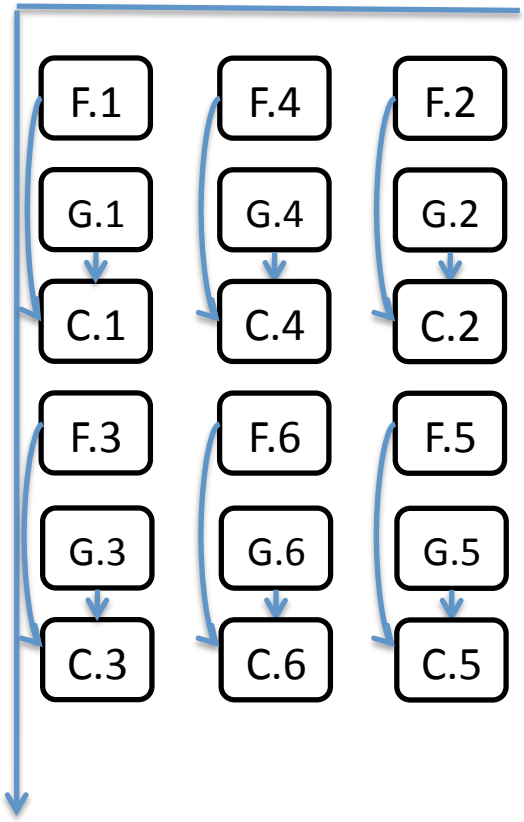
```
  while(TRUE){
E:  node=getNodeOrExit();
B:  index=calc
      (node->data);
  }
```

LOCALWRITE

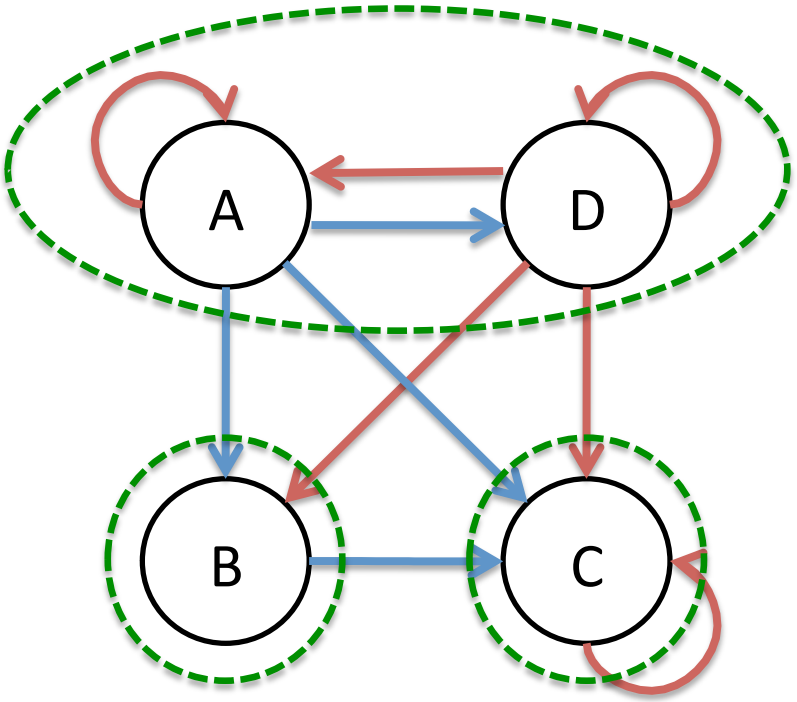
```
  while(TRUE){
F: node=getNodeOrExit();
G: index=getIndex();
C: density[index]=
     update_density
     (density[index],
       node->data);  }
```







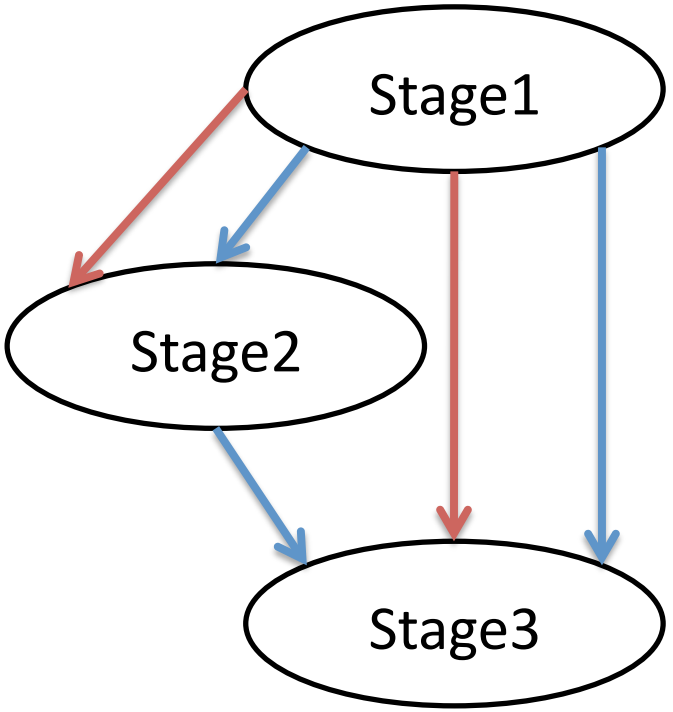
```
   node = list->head;
A: while (node != NULL) {
B:   index = calc(node->data);
C:   density[index] = update_density
        (density[index], node->data);
D:   node = node->next;
   }
```



intra-loop dependence

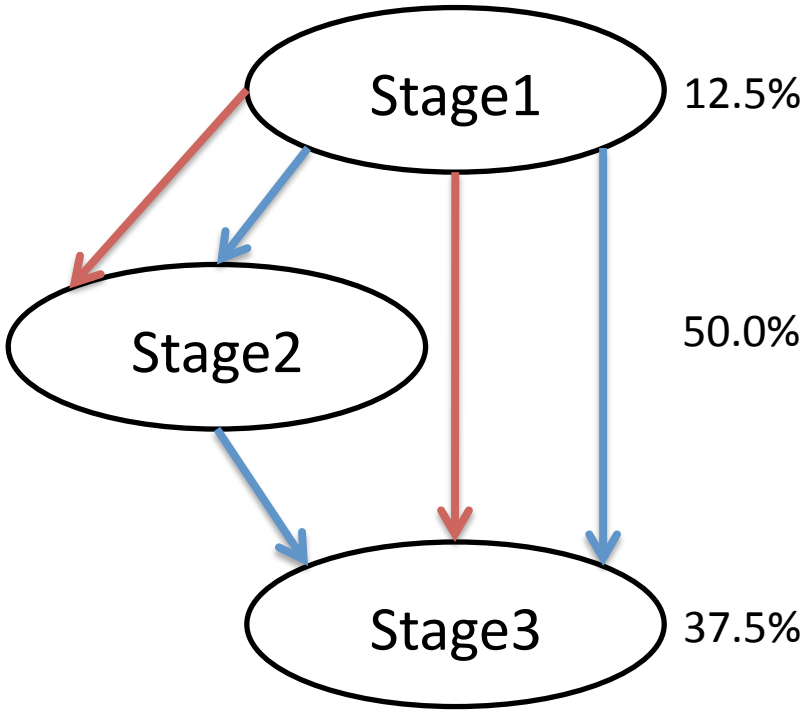
inter-loop dependence

DSWP+

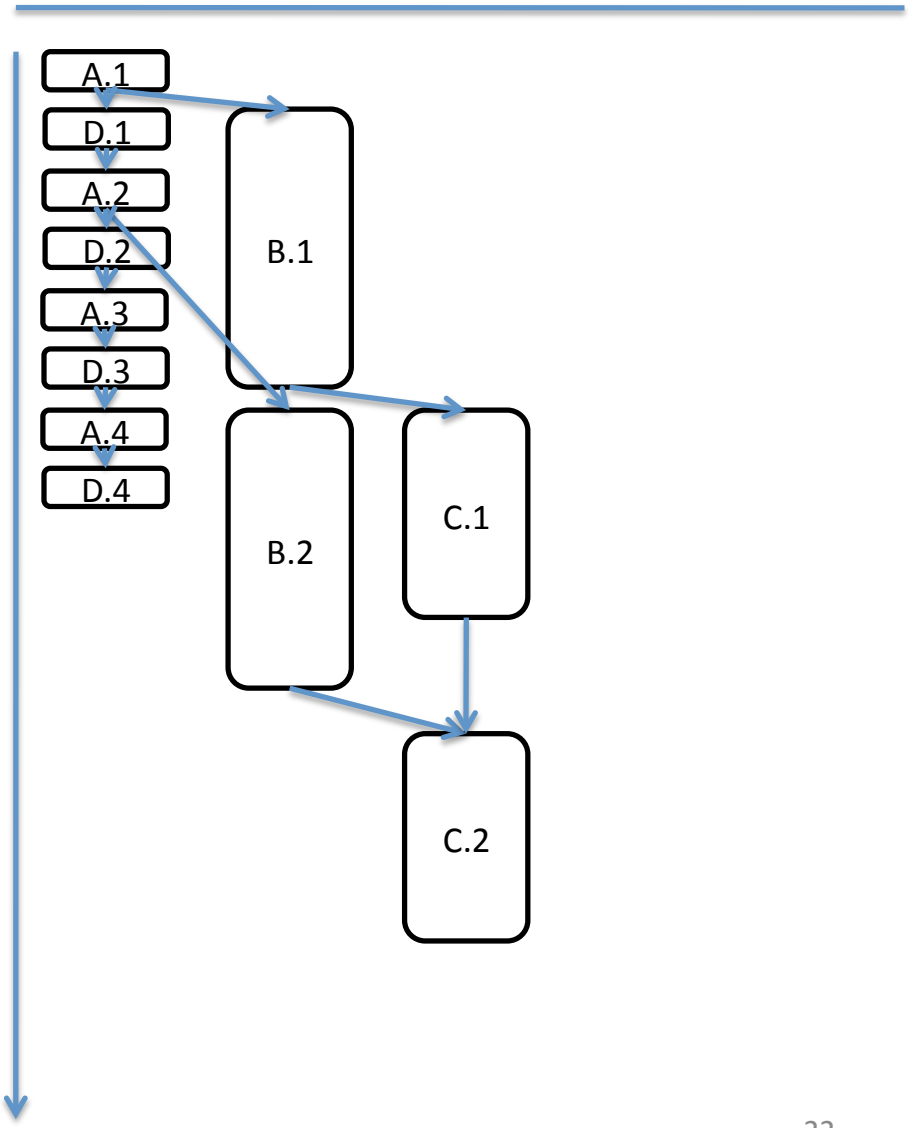


Huang et al. [CGO '10]

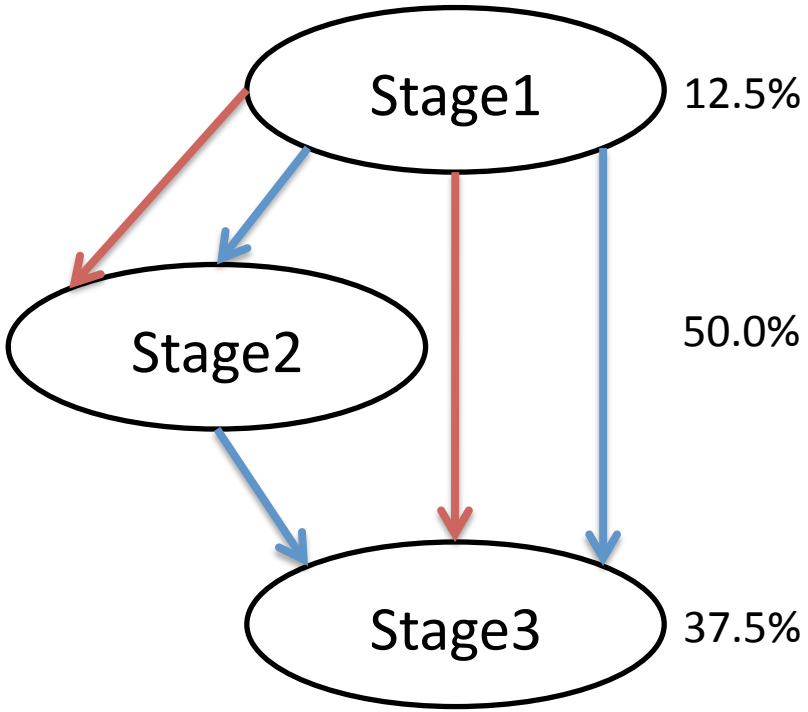
DSWP+

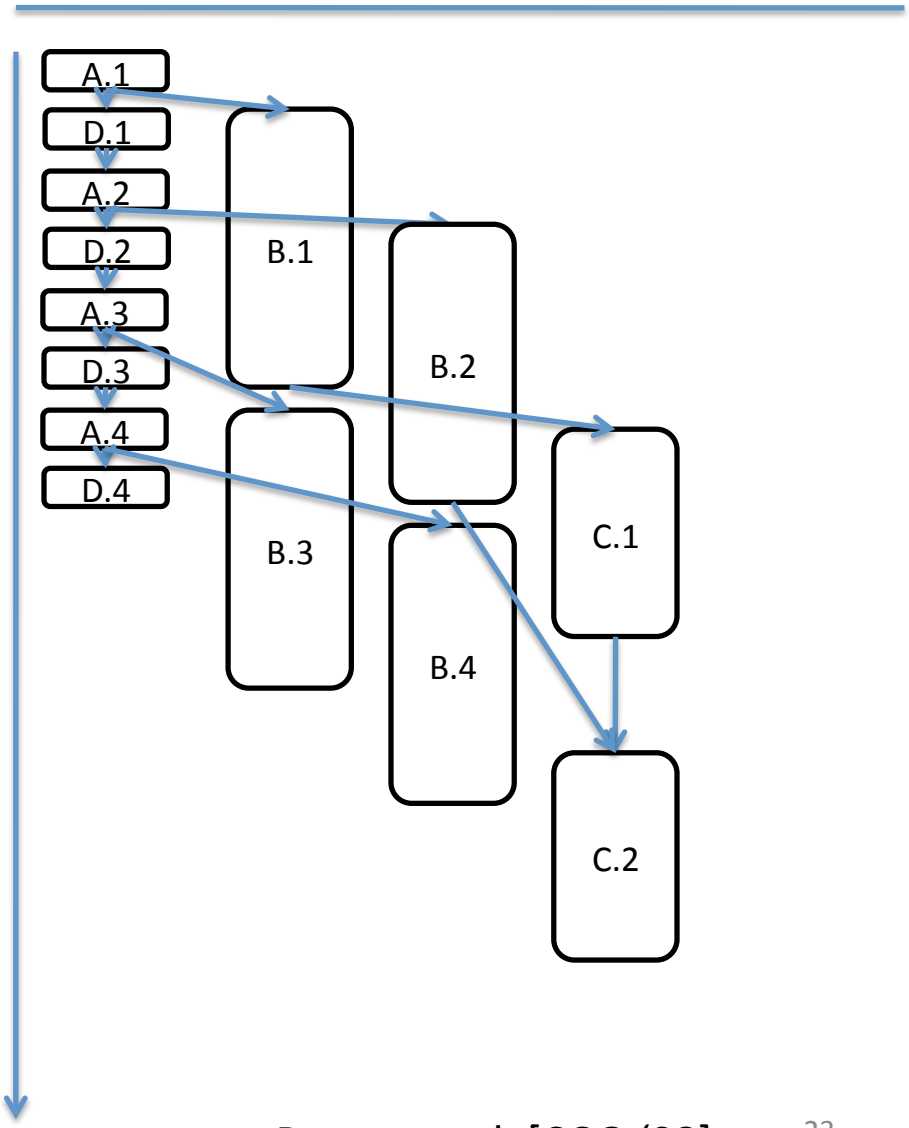


Max(12.5, 50.0, 37.5)
= 50.0 %
=> 2X (speedup)



DSWP+DOALL

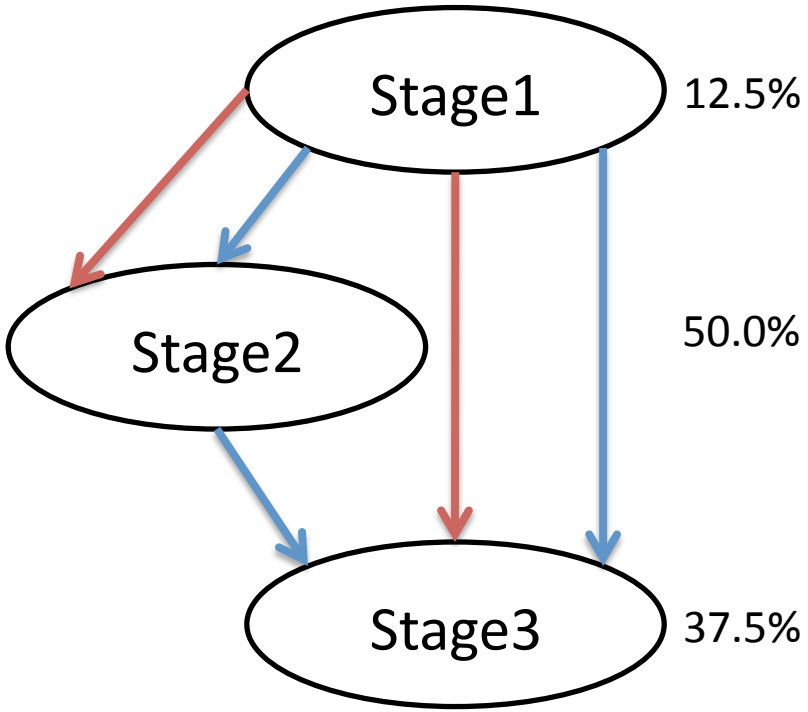


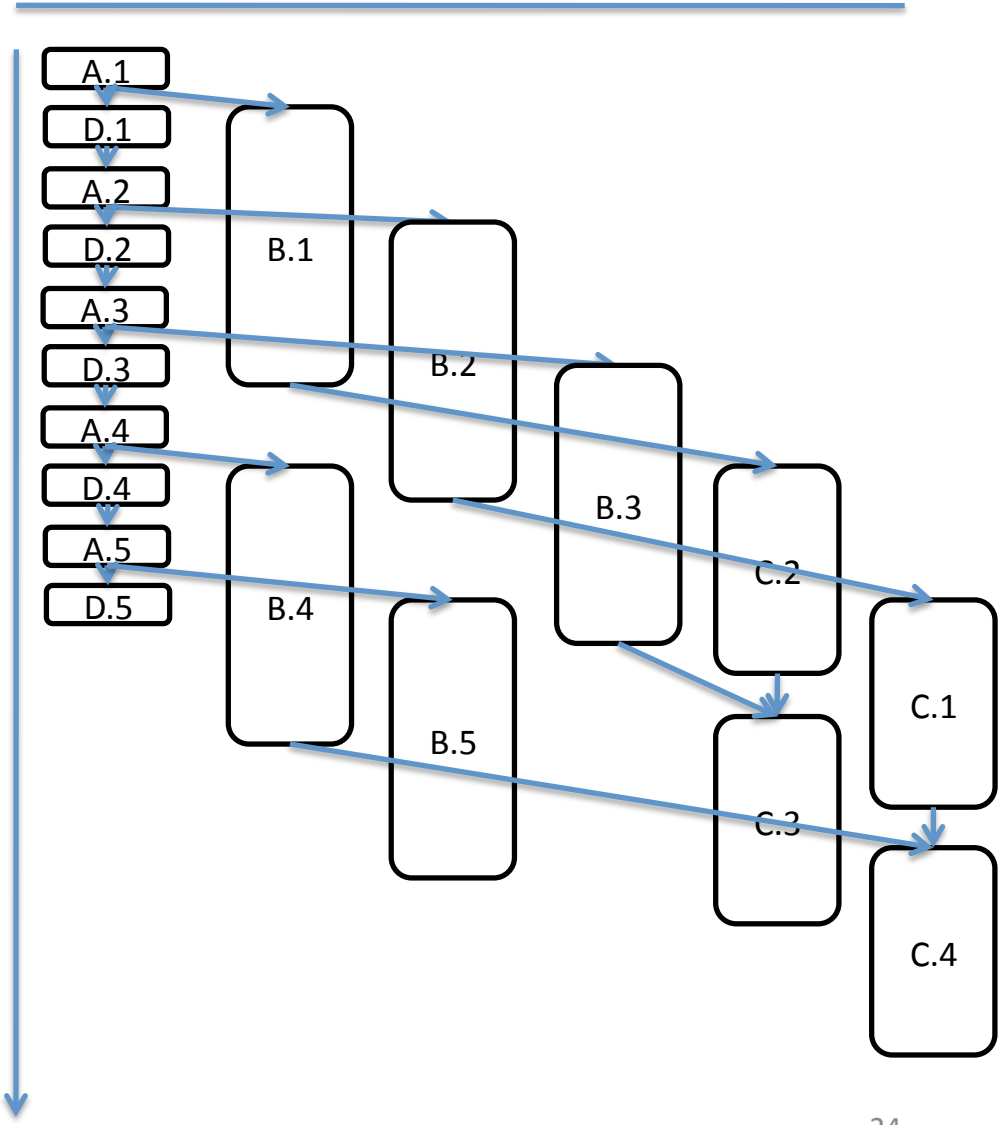


Max(12.5, 50.0/2, 37.5)
= 37.5 %
=> 2.7X (speedup)

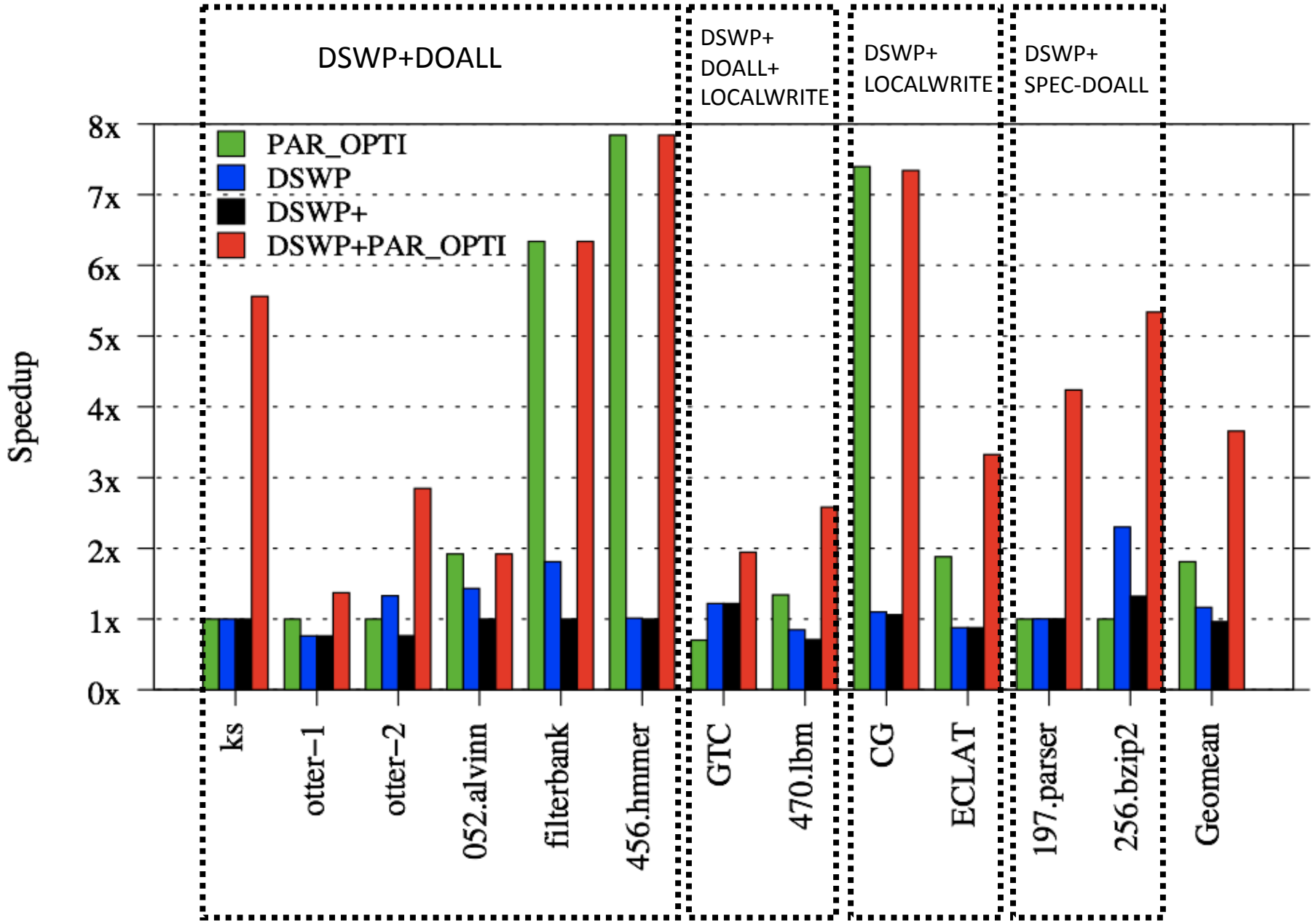
Raman et al. [CGO '08]

DSWP+DOALL+LOCALWRITE





Max(12.5, 50.0/3, 37.5/2)
= 18.8 %
=> 5.3X (speedup)

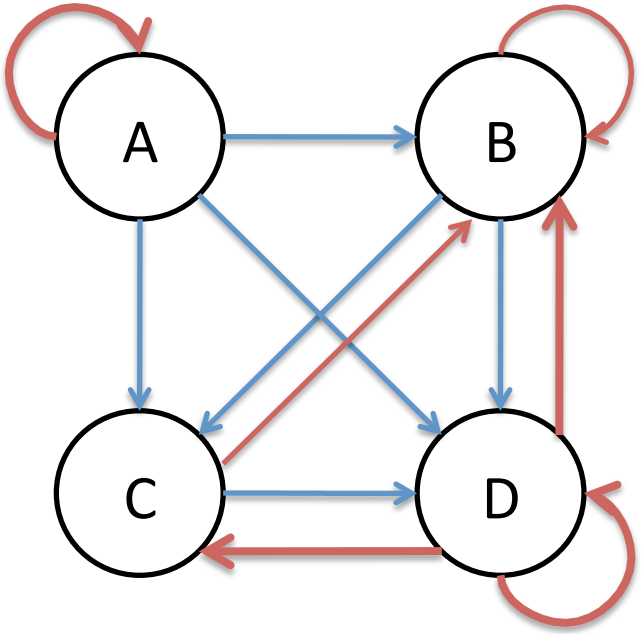

DSWP+DOALL
DSWP+ DOALL+ LOCALWRITE
DSWP+ LOCALWRITE
DSWP+ SPEC-DOALL
PAR_OPTI
DSWP
DSWP+
DSWP+PAR_OPTI
Speedup
8x
7x
6x
5x
4x
3x
2x
1x
0x
ks
otter−1
otter−2
052.alvinn
filterbank
456.hmmer
GTC
470.lbm
CG
ECLAT
197.parser
256.bzip2
Geomean

Speculative Solution

Types of Speculation

- Branch prediction
- Memory speculation
- Value prediction

```
A: for (i = 0; i < N; i++) {
B:   A[K[i]] = value1;
C:   B[i] = A[L[i]];
D:   A[R[i]] = A[R[i]] + value2;
   }
```

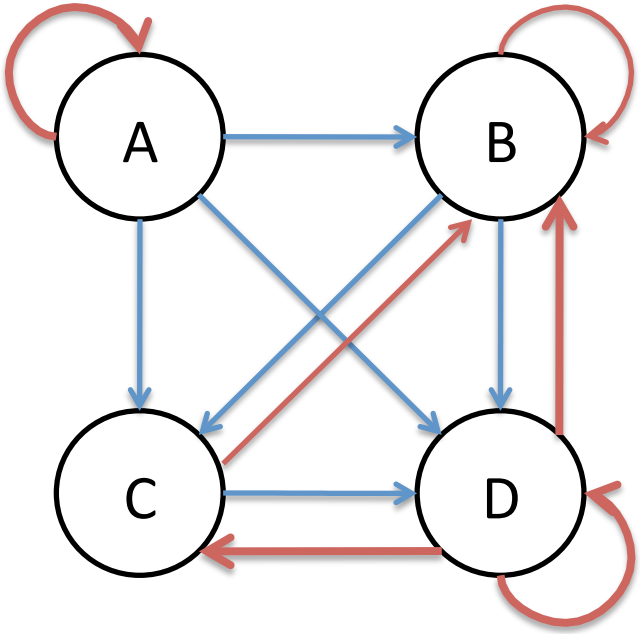


intra-loop dependence

inter-loop dependence

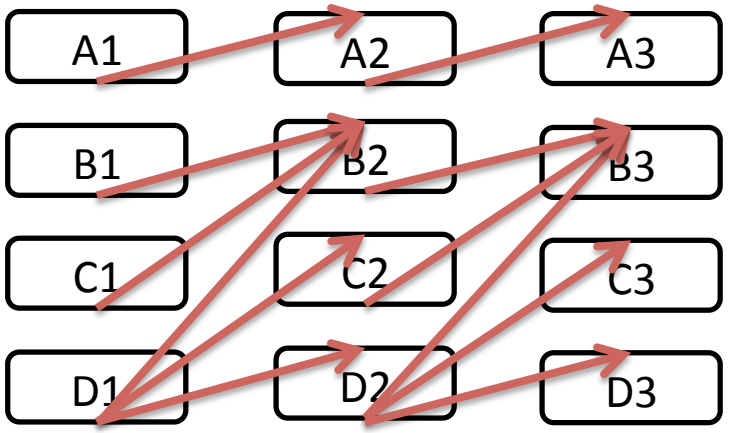
DOALL

```
A: for (i = 0; i < N; i++) {
B:   A[K[i]] = value1;
C:   B[i] = A[L[i]];
D:   A[R[i]] = A[R[i]] + value2;
   }
```



intra-loop dependence

inter-loop dependence



```
A: for (i = 0; i < N; i++) {
B:   A[K[i]] = value1;
C:   B[i] = A[L[i]];
D:   A[R[i]] = A[R[i]] + value2;
   }
```

R[1:5] = (1, 0, 1, 0, 1)
K[1:5] = (1, 2, 3, 4, 1)
L[1:5] = (1, 2, 4, 5, 1)

LRPD test

A_{write}	0	0	0	0	0	0	0	0
A_{read}	0	0	0	0	0	0	0	0
A_{np}	0	0	0	0	0	0	0	0
A_{redux}	0	0	0	0	0	0	0	0

Rauchwerger et al. [Parallel and Distributed Systems ‘99]

```
A: for (i = 0; i < N; i++) {
     markwrite(K[i]);
B:   A[K[i]] = value1;
C:   B[i] = A[L[i]];
     markwrite(R[i]);
D:   A[R[i]] = A[R[i]] + value2;
   }
```

R[1:5] = (1, 0, 1, 0, 1)
K[1:5] = (1, 2, 3, 4, 1)
L[1:5] = (1, 2, 4, 5, 1)

LRPD test

A_{write}	1	1	1	1	1	0	0	0
A_{read}	0	0	0	0	0	0	0	0
A_{np}	0	0	0	0	0	0	0	0
A_{redux}	0	0	0	0	0	0	0	0

```
A: for (i = 0; i < N; i++) {
B:   A[K[i]] = value1;
     markread(L[i]);
C:   B[i] = A[L[i]];
D:   A[R[i]] = A[R[i]] + value2;
   }
```

R[1:5] = (1, 0, 4, 0, 1)
K[1:5] = (1, 2, 3, 4, 1)
L[1:5] = (1, 2, 4, 5, 1)

LRPD test

A_{write}	1	1	1	1	1	0	0	0
A_{read}	0	0	0	0	0	1	0	0
A_{np}	0	0	0	0	1	0	0	0
A_{redux}	0	0	0	0	0	0	0	0

```
A: for (i = 0; i < N; i++) {
     markredux(K[i]);
B:   A[K[i]] = value1;
     markredux(L[i]);
C:   B[i] = A[L[i]];
D:   A[R[i]] = A[R[i]] + value2;
   }
```

R[1:5] = (1, 0, 4, 0, 1)
K[1:5] = (1, 2, 3, 4, 1)
L[1:5] = (1, 2, 4, 5, 1)

LRPD test

A_{write}	1	1	1	1	1	0	0	0
A_{read}	0	0	0	0	0	1	0	0
A_{np}	0	0	0	0	1	0	0	0
A_{redux}	1	1	1	1	1	1	0	0

```
A: for (i = 0; i < N; i++) {
B:   A[K[i]] = value1;
C:   B[i] = A[L[i]];
D:   A[R[i]] = A[R[i]] + value2;
   }
```

R[1:5] = (1, 0, 4, 0, 1)
K[1:5] = (1, 2, 3, 4, 1)
L[1:5] = (1, 2, 4, 5, 1)

LRPD test

A_{write}	1	1	1	1	1	0	0	0
A_{read}	0	0	0	0	0	1	0	0
A_{np}	0	0	0	0	1	0	0	0
A_{redux}	0	1	1	1	1	1	0	0

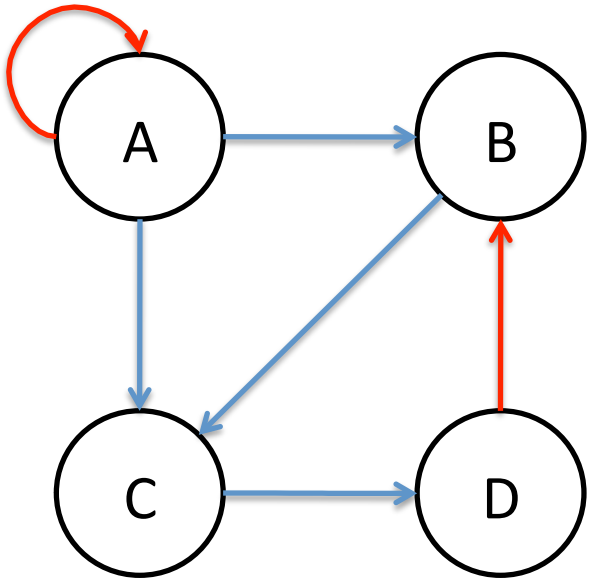
$$A_{write} \wedge A_{read} \neq 0$$

$$A_{write} \wedge A_{np} \wedge A_{redux} \neq 0$$

Misspeculation Recovery for LRPD

- Before speculative execution, copy values in the shared array to the privatized array.
- After speculative execution, copy out the privatized array values back to the original shared array.
- If speculative execution fails, throw away the values in privatized array, and re-execute the loop sequentially.

```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```

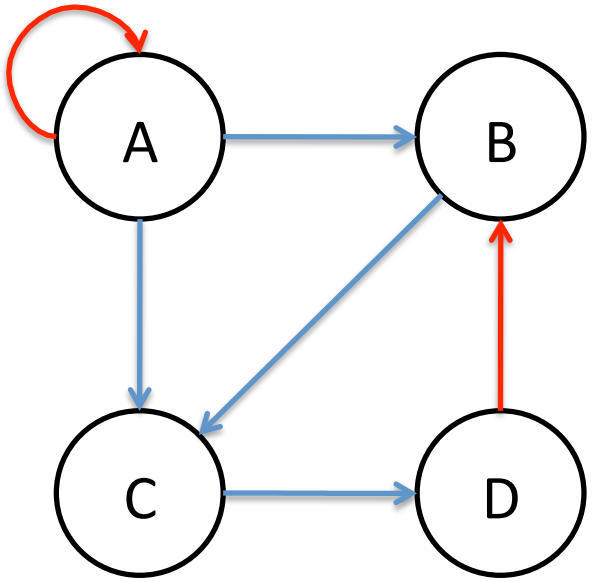


intra-loop dependence

inter-loop dependence

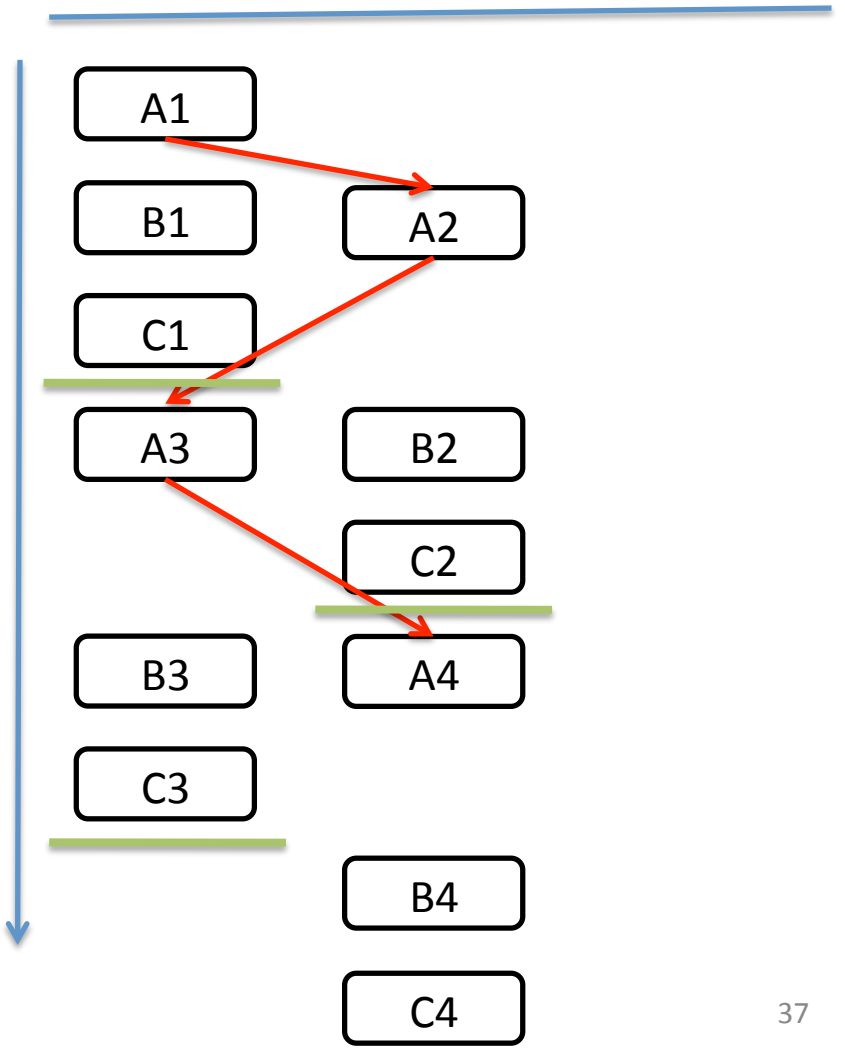
DOACROSS

```
node = list->head;
A: while (node = node->next) {
B:    node->val += inc;
C:    if (foo(node->val))
D:       inc++;
   }
```



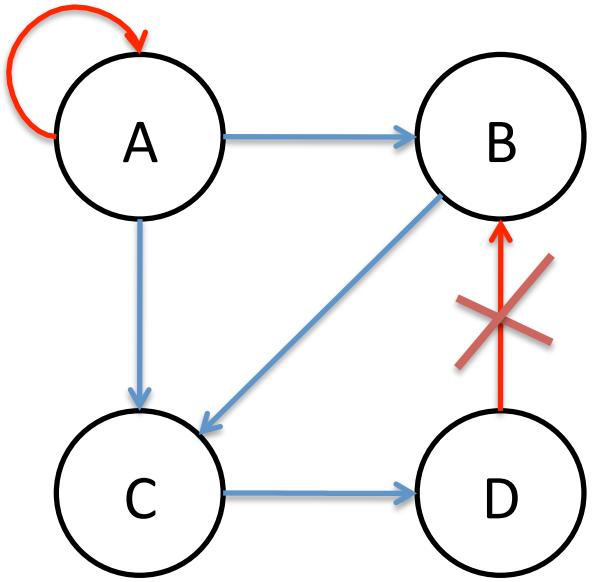
intra-loop dependence

inter-loop dependence



Steffan et al. [Computer Systems '05]

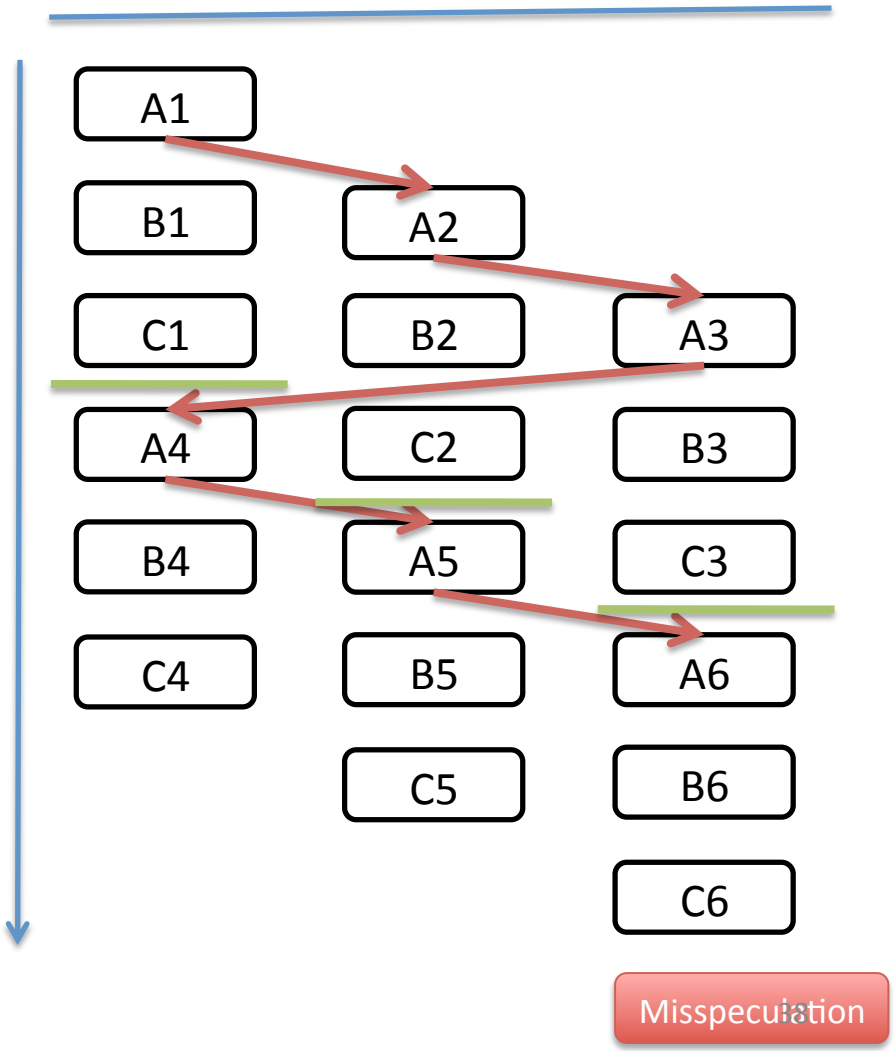
```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```



intra-loop dependence

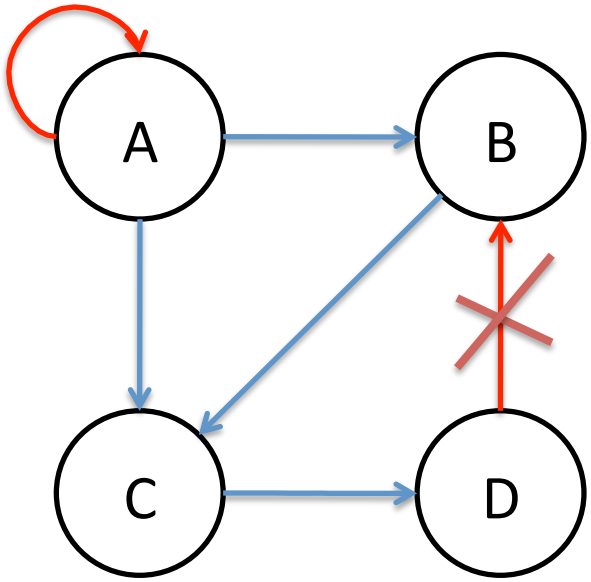
inter-loop dependence

TLS



Steffan et al. [Computer Systems '05]

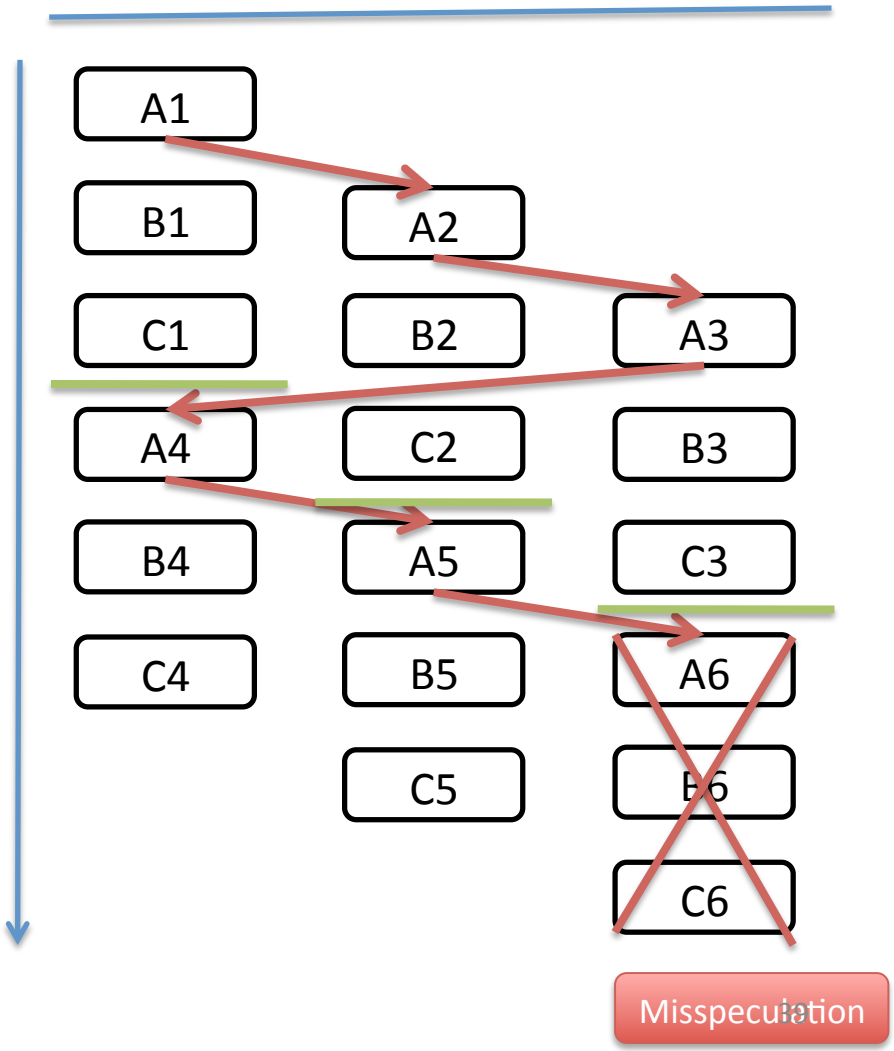
```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```



intra-loop dependence

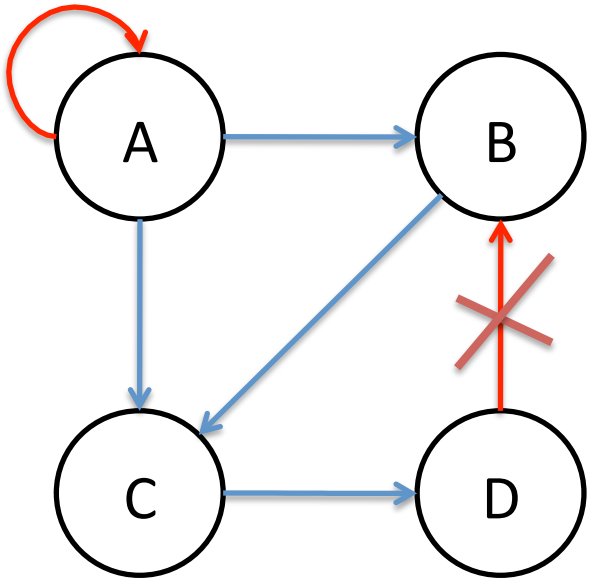
inter-loop dependence

TLS



Steffan et al. [Computer Systems '05]

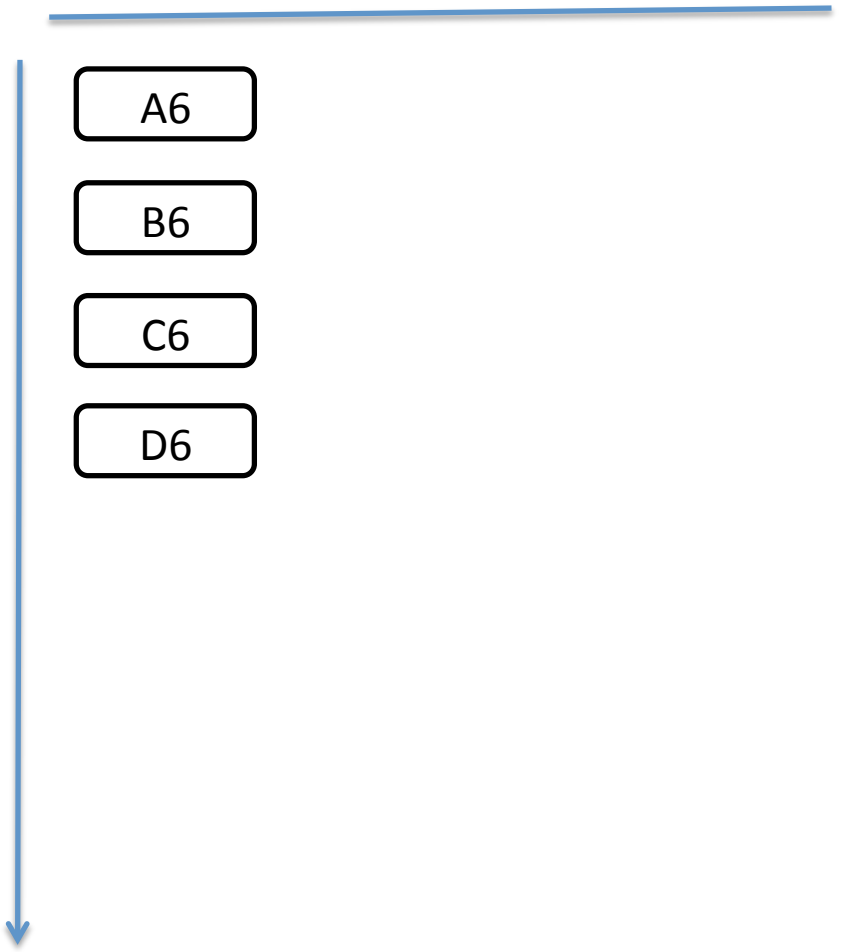
```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```



intra-loop dependence

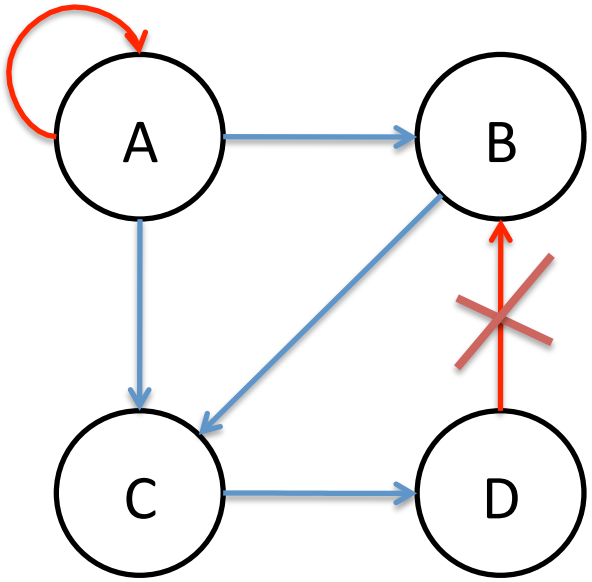
inter-loop dependence

TLS



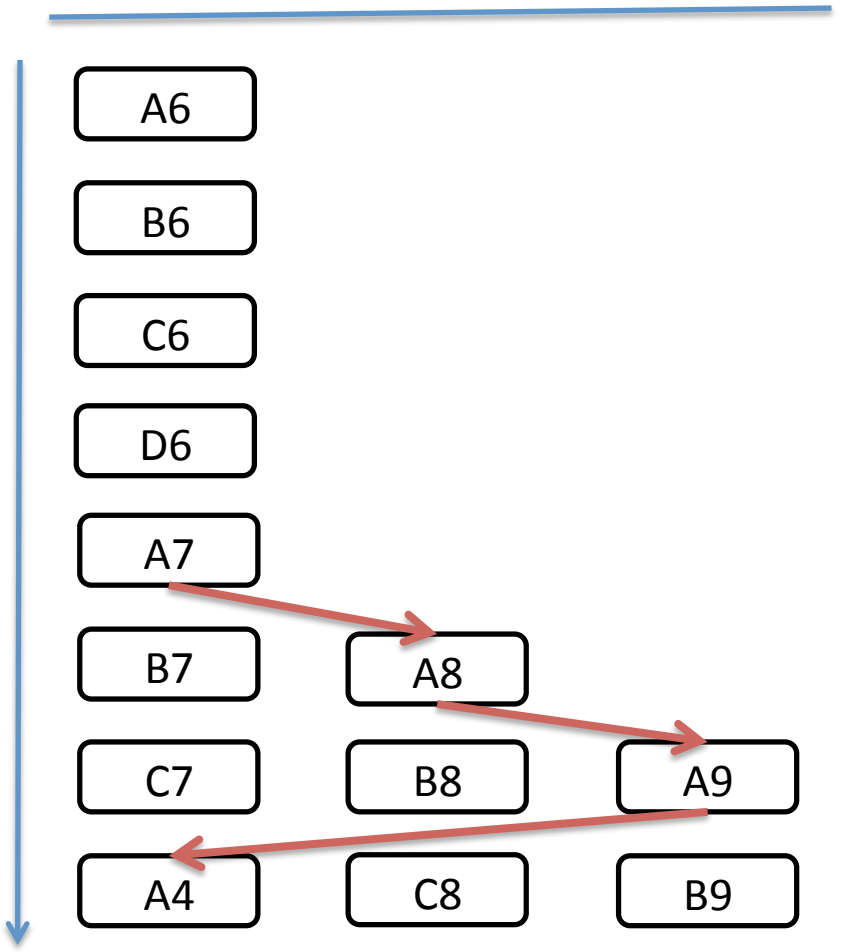
Steffan et al. [Computer Systems '05]

```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```

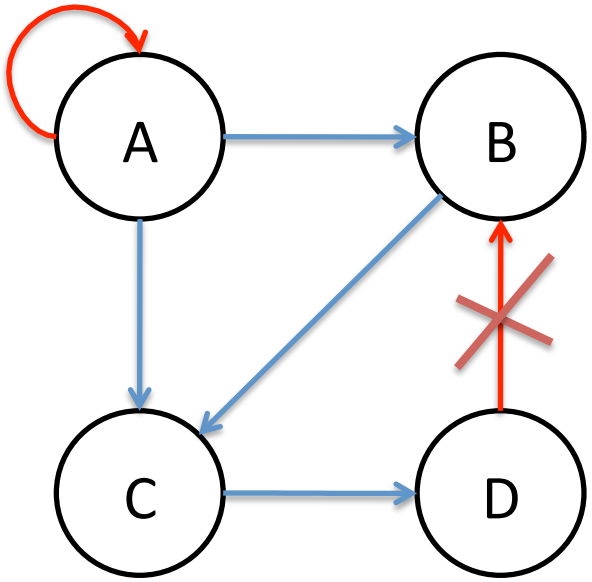


intra-loop dependence
inter-loop dependence

TLS



```
node = list->head;
A: while (node = node->next) {
B:    node->val += inc;
C:    if (foo(node->val))
D:      inc++;
   }
```

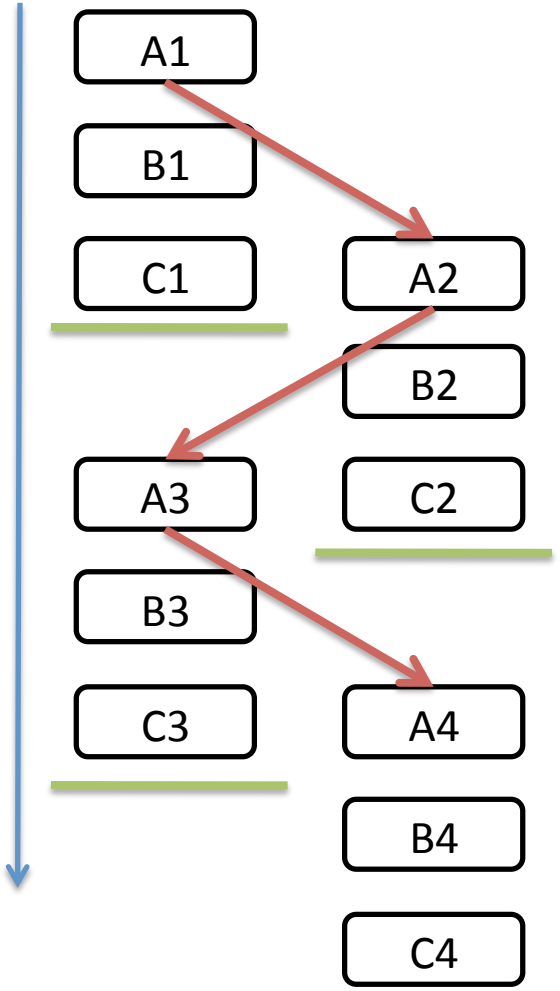


intra-loop dependence

inter-loop dependence

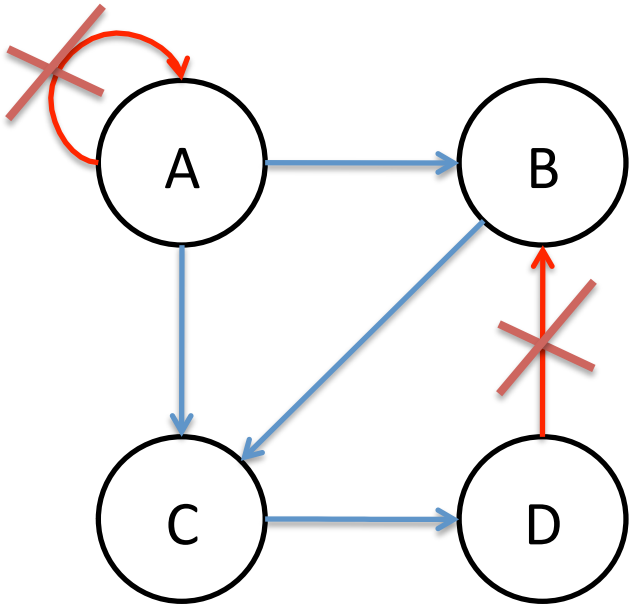
TLS

(communication latency)



Raman et al. [CGO '08]

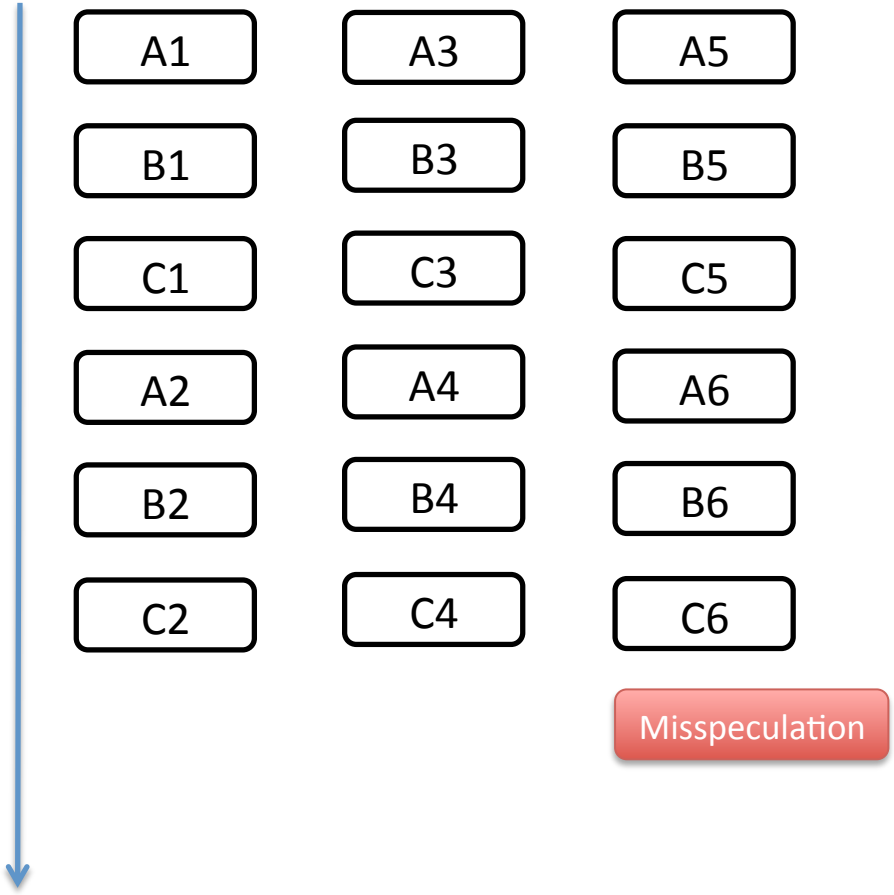
```
node = list->head;
A: while (node = node->next) {
B:    node->val += inc;
C:    if (foo(node->val))
D:       inc++;
   }
```



intra-loop dependence

inter-loop dependence

TLS with value prediction - spice

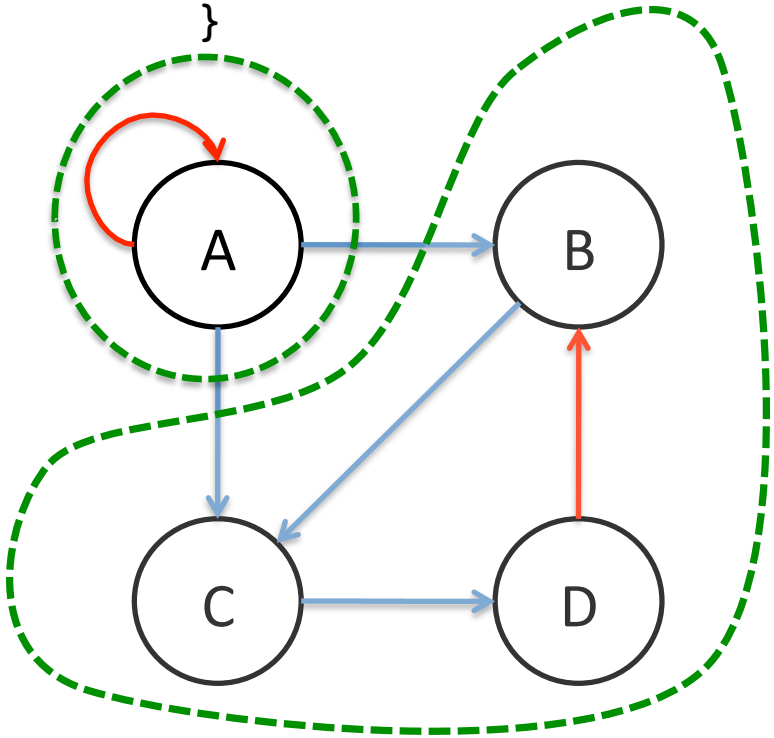


Misspeculation Recovery for TLS

- Requires hardware support to detect mis-speculations and recover from them.
- Cache coherence protocol is used to identify RAW dependence violation.
- Updates from speculative iterations are buffered in private caches and written to shared caches or main memory only after the current iteration is guaranteed to be mis-speculation free.

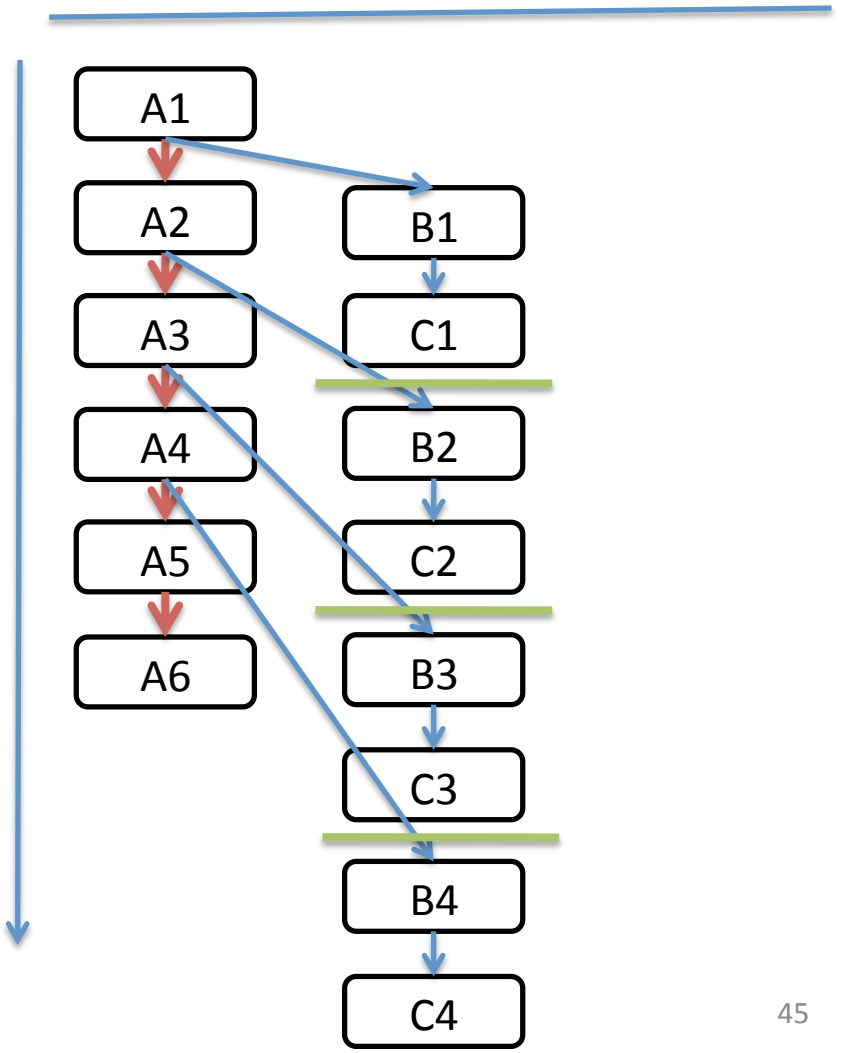
DSWP

```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```



intra-loop dependence

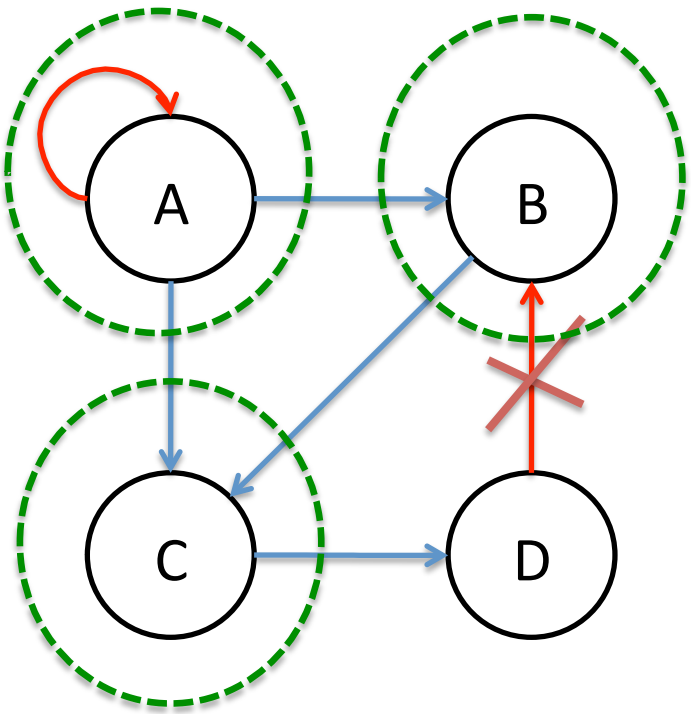
inter-loop dependence



Vachharajani et al. [PACT '07]

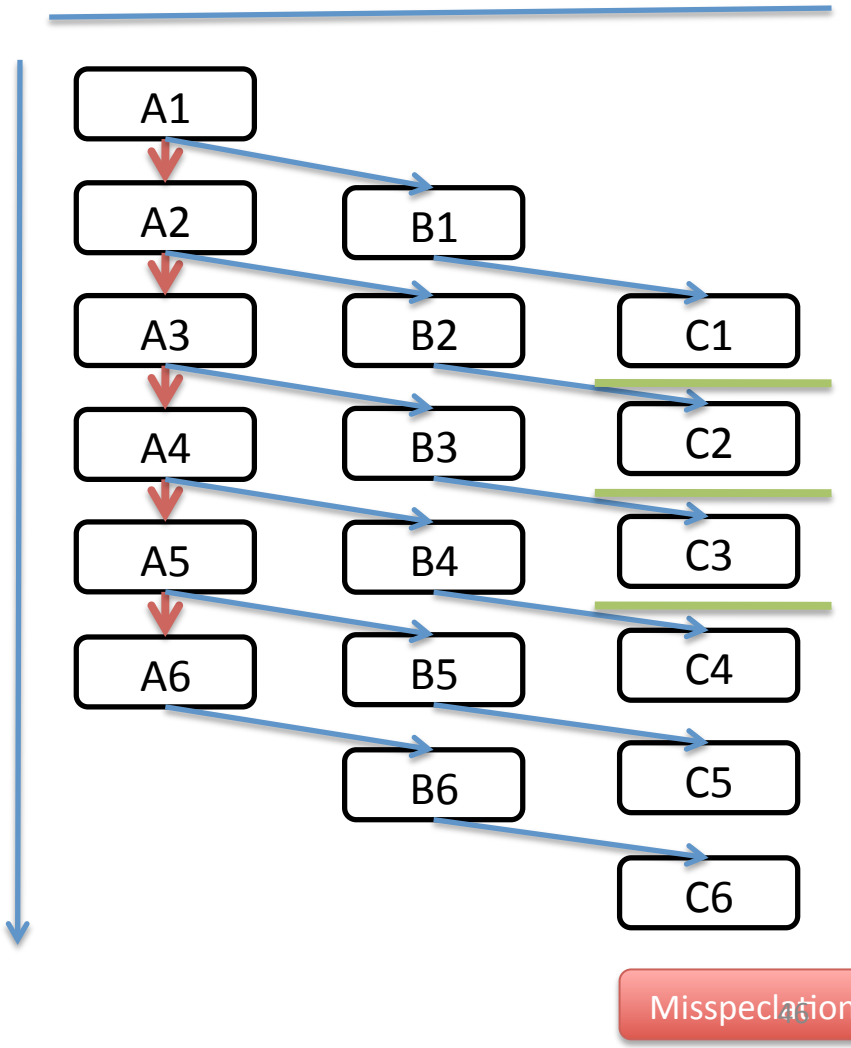
Spec-DSWP

```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```



intra-loop dependence

inter-loop dependence

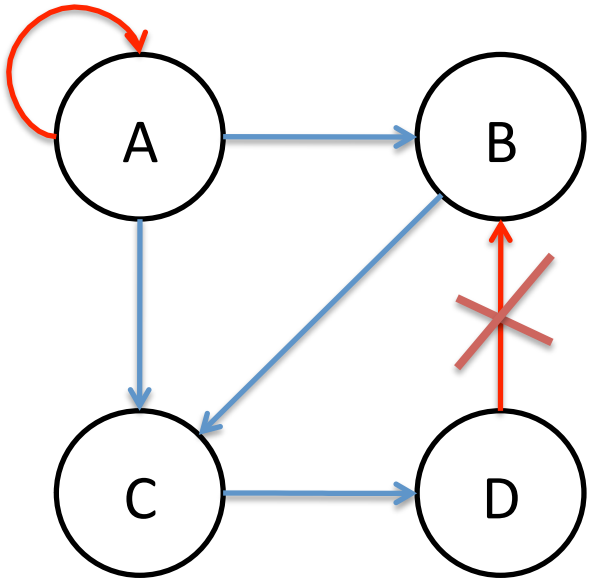


Misspeculation Recovery for Spec-DSWP

- A separate thread “commit thread” is used to detect mis-speculation and orchestrate the recovery of the correct state.
- Recovery involves undoing the effects of the mis-speculated statements and sequential re-execution of the mis-speculated code.
- Multi-threaded transaction memory is used to recover from the effects of speculation on memory state.

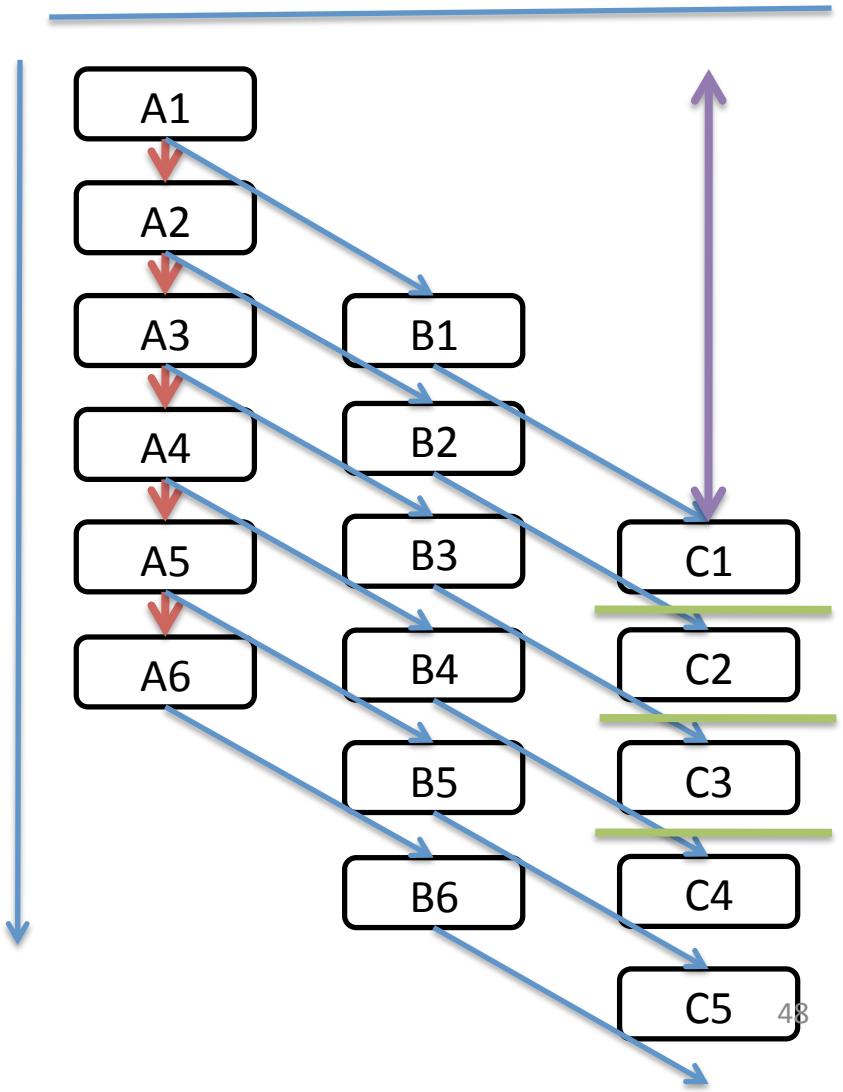
Spec-DSWP

```
node = list->head;
A: while (node = node->next) {
B:   node->val += inc;
C:   if (foo(node->val))
D:     inc++;
   }
```

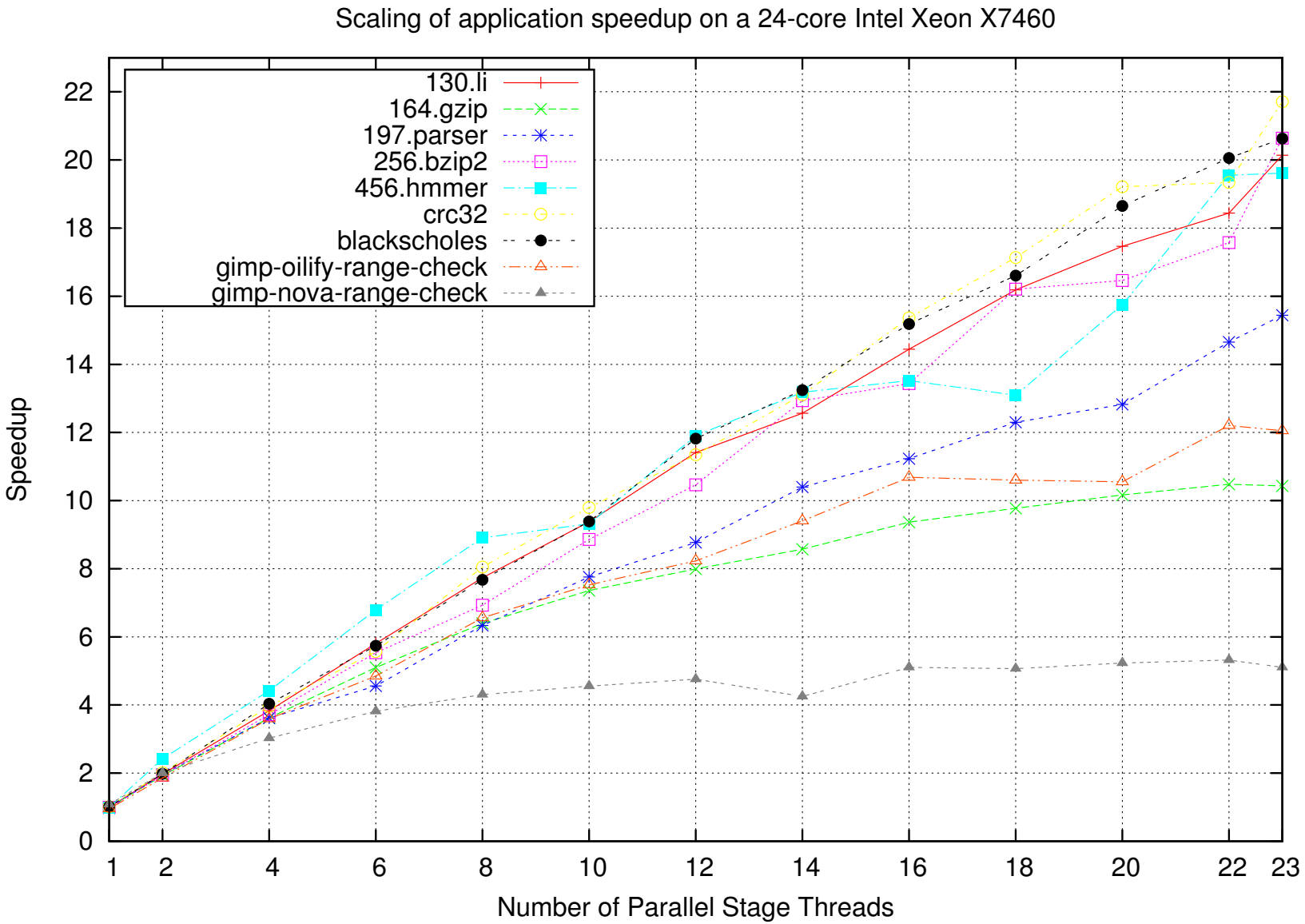


intra-loop dependence

inter-loop dependence



Raman et al. [ASPLOS '10]



Conclusion

- Existing automatic parallelization techniques have limitations when applied alone.
- DSWP+ technique helps improve their applicability by separating the inter-iteration dependences to a separate pipeline stage.
- Speculative technique helps improve their applicability by speculatively cutting dependences.

Thanks!