Join Example and its Join Graph

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot$

```
SELECT   A.a1, B.a2, B.a3, D.a4
FROM     A JOIN  B ON A.id=B.a_id
         JOIN    D ON D.b_id=B.id
         JOIN    C ON C.id=D.c_id
WHERE    A.a1 = 42
         AND B.a3=12
         AND D.a2=25
```

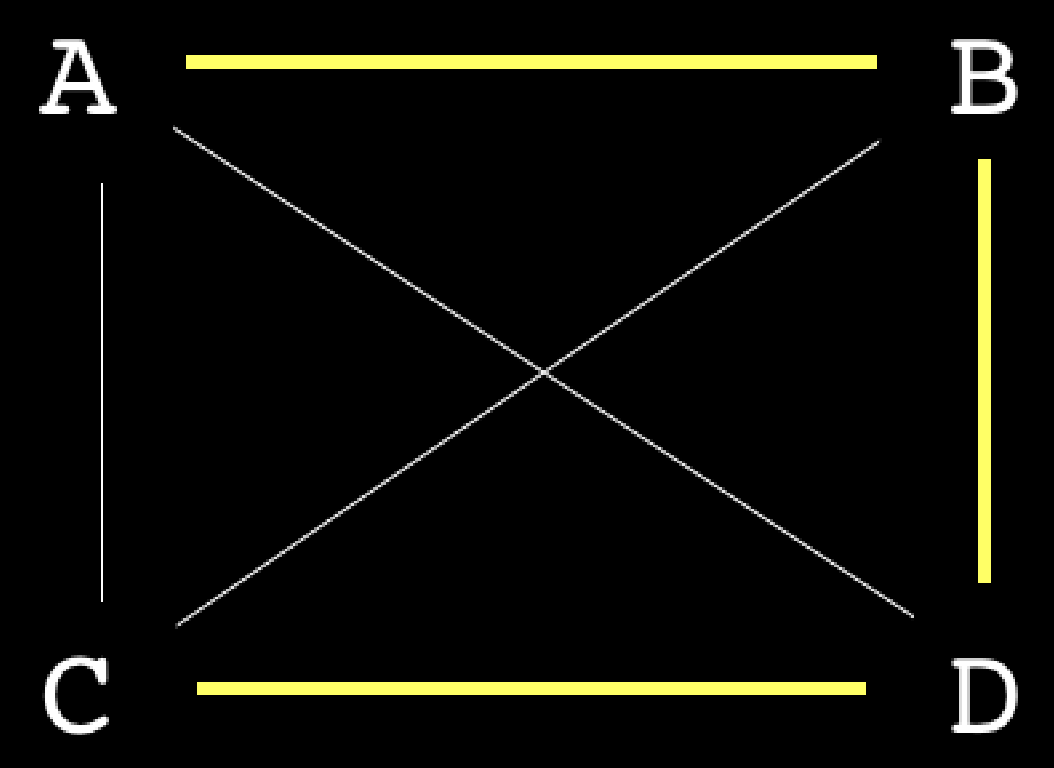
Join Example and its Join Graph

```
SELECT   A.a1, B.a2, B.a3, D.a4
FROM     A JOIN  B ON A.id=B.a_id
         JOIN    D ON D.b_id=B.id
         JOIN    C ON C.id=D.c_id
WHERE    A.a1 = 42
         AND B.a3=12
         AND D.a2=25
```

$A \bowtie B \bowtie D \bowtie C$

	D	C	B
A	$\times$	$\times$	$\bowtie$
B	$\bowtie$	$\times$	
C	$\bowtie$		

Join Graph

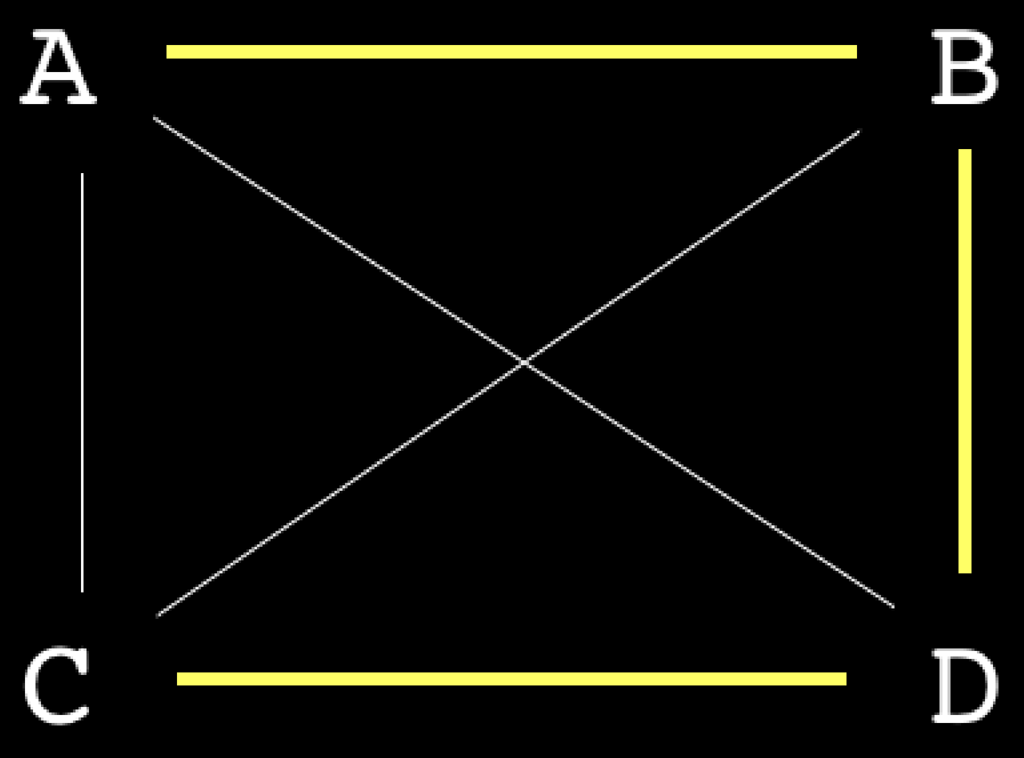


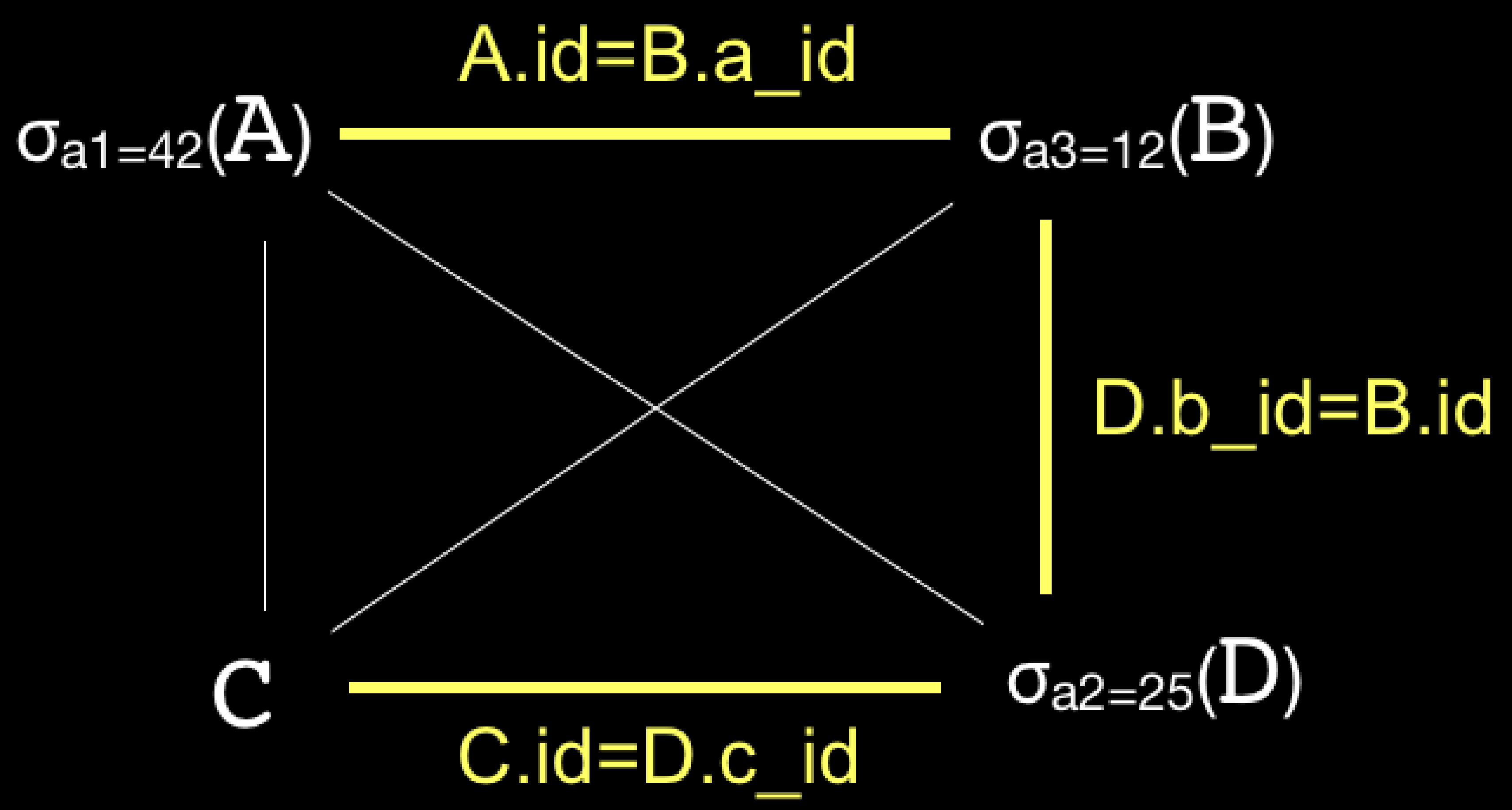
Join Example and its Join Graph

```
SELECT   A.a1, B.a2, B.a3, D.a4
FROM     A JOIN  B ON A.id=B.a_id
         JOIN    D ON D.b_id=B.id
         JOIN    C ON C.id=D.c_id
WHERE    A.a1 = 42
         AND B.a3=12
         AND D.a2=25
```

$A \bowtie B \bowtie D \bowtie C$

	D	C	B
A	×	×	⋈
B	⋈	×	
C	⋈		





Core Idea of Dynamic Programming

start computing optimal plan for **each** input relation

compute plans for i>1 (sub-)plans using optimal subplans for i-1 inputs

Two Requirements for Dynamic Programming

(1) optimality principle

$\boxed{A, B}\ \boxed{C, D}$

$A, B \rightarrow SP1$

$C, D \rightarrow SP2$

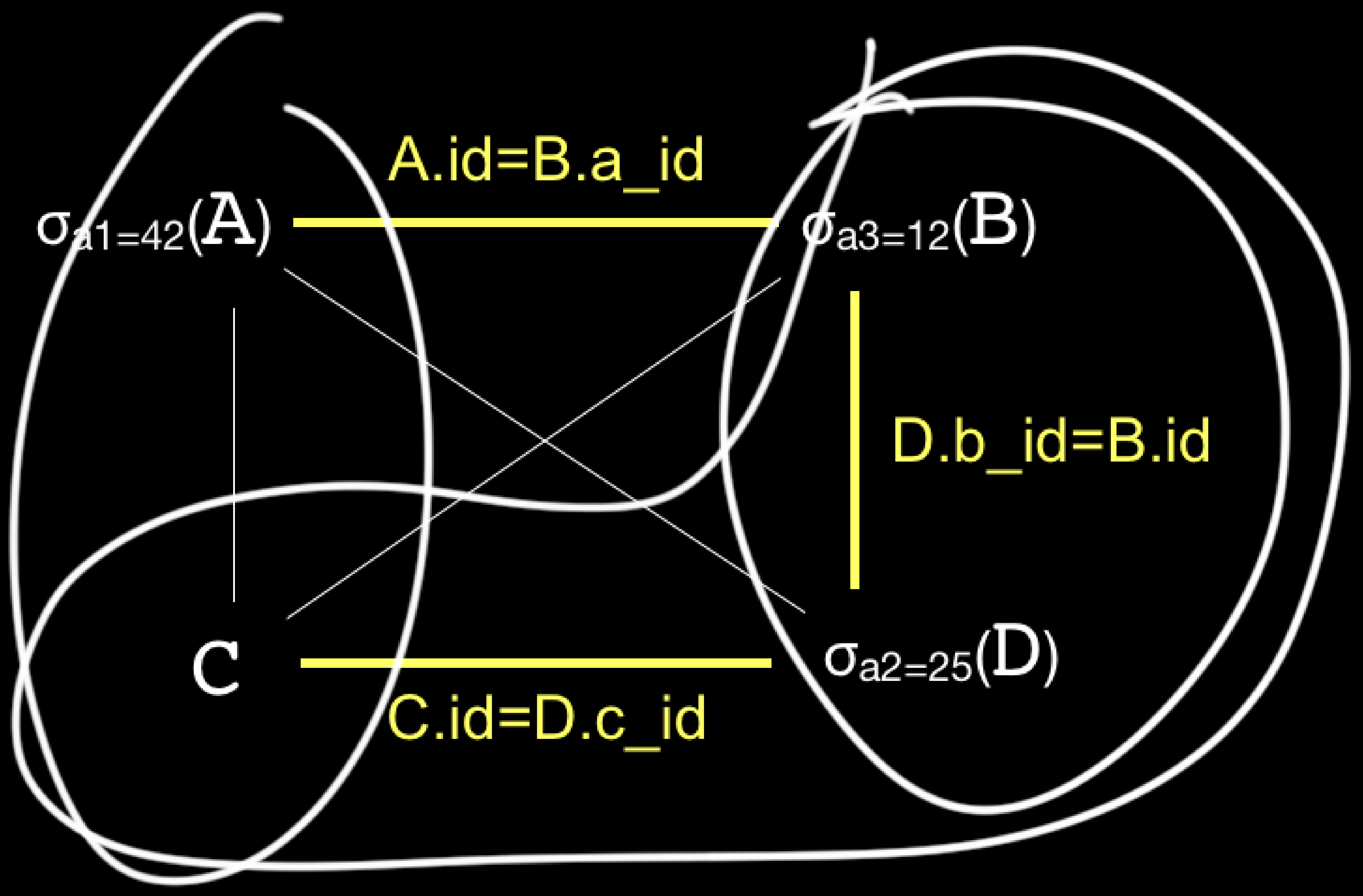
$SP1 \oplus SP2 = \text{Plan } 4\ 2$

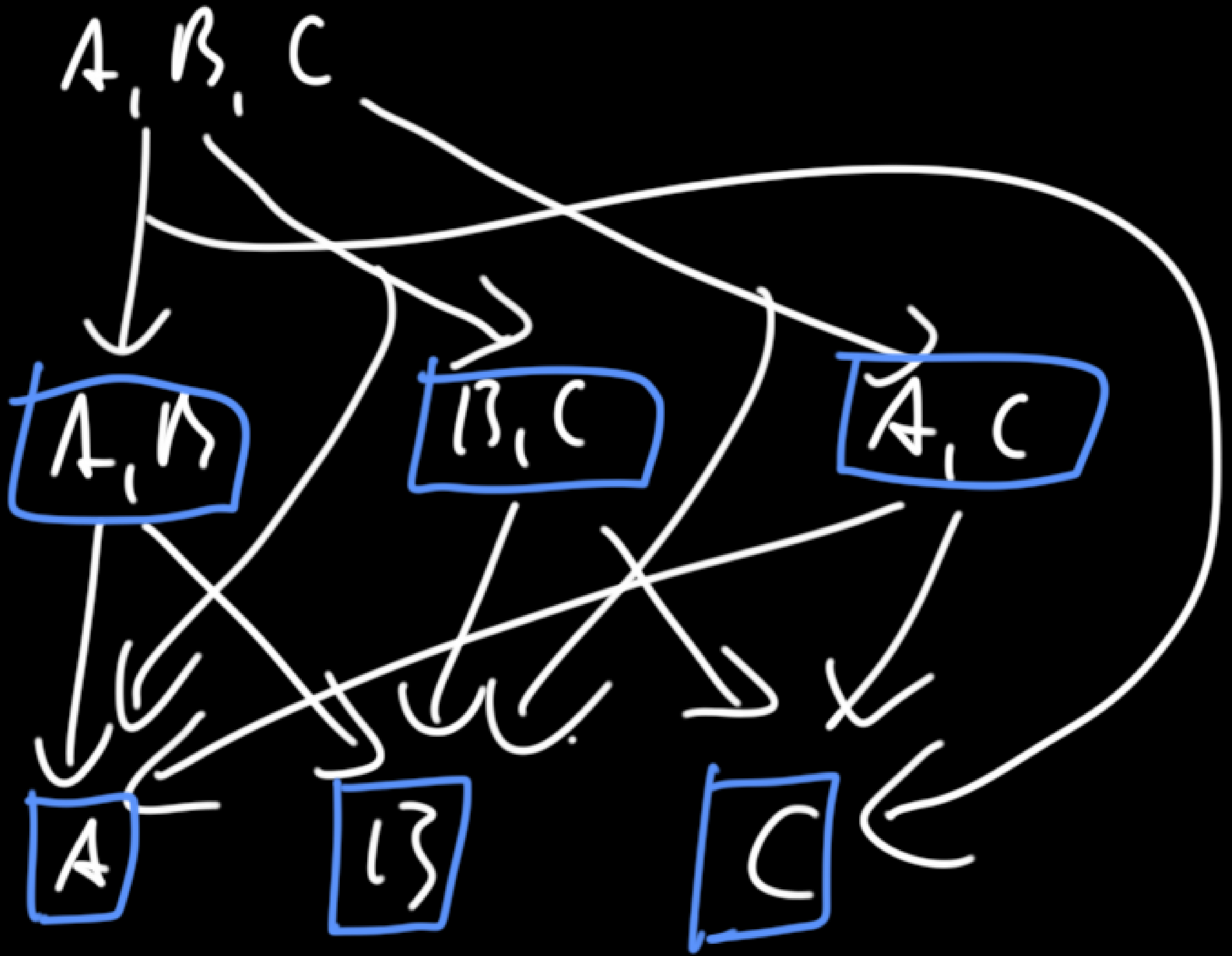
optimal optimal optimal

Two Requirements for Dynamic Programming

(1) optimality principle

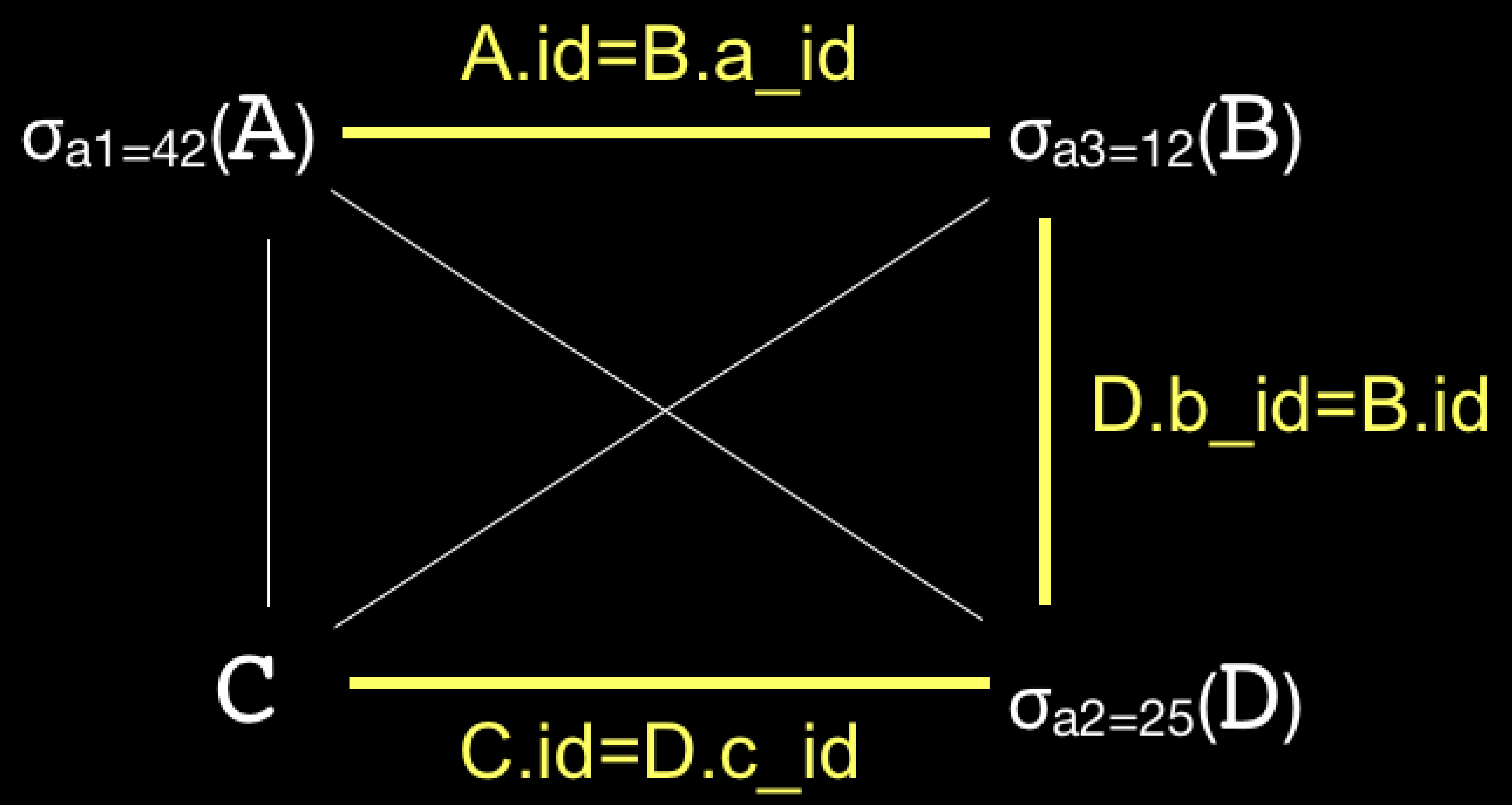
(2) overlapping subproblems





Example: Size 1 Plans (generating...)

assuming no interesting orders



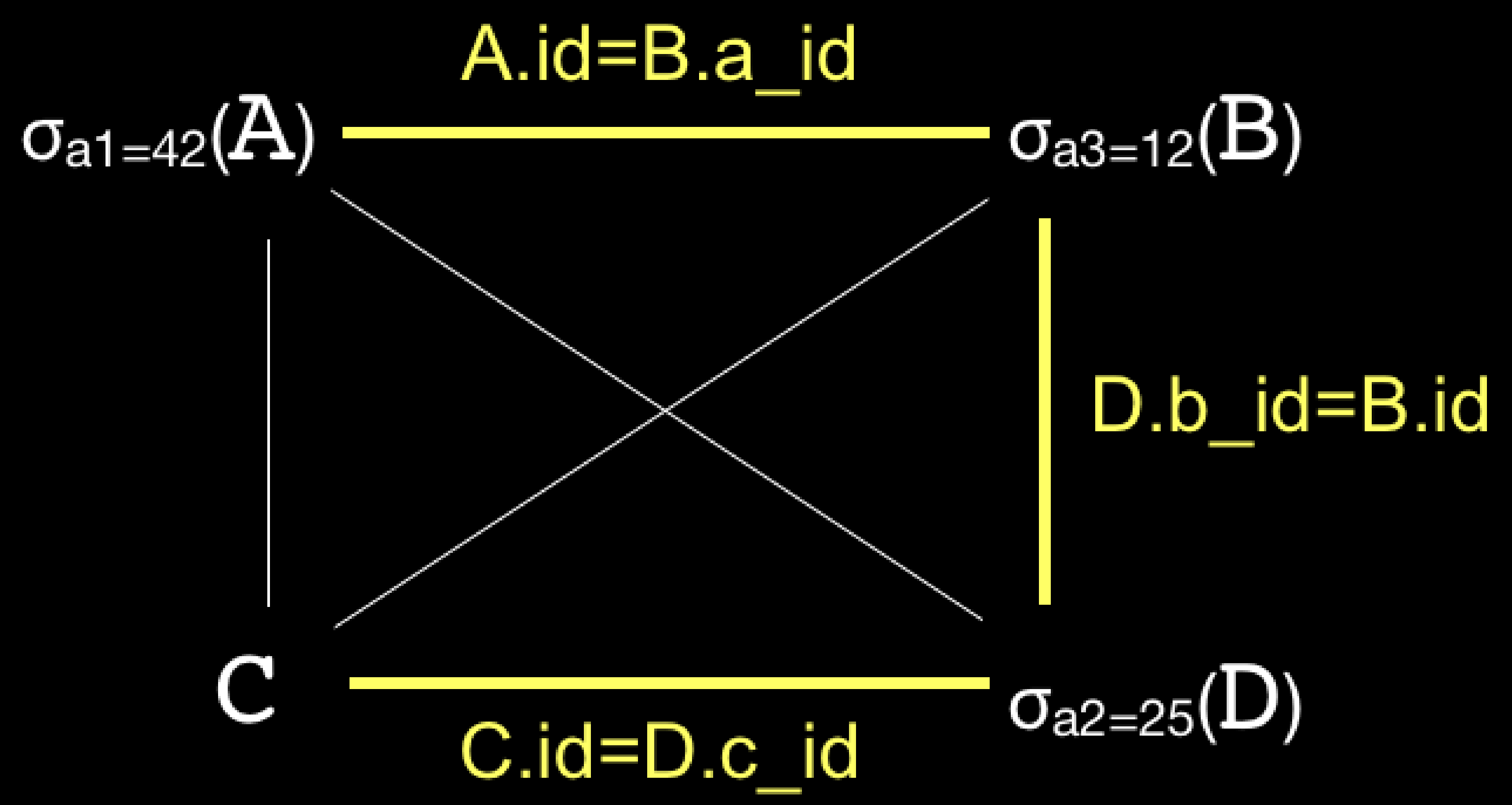
Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				scan(A), iseek(a1, A), isam(id, A)
	X			scan(B), iseek(a3, B), isam(id, B)
		X		scan(C), isam(id, C)
			X	scan(D), iseek(a2, D), isam(id, D)

space complexity:

$O(2^n)$

Example: Size 1 Plans (pruning...)

assuming no interesting orders

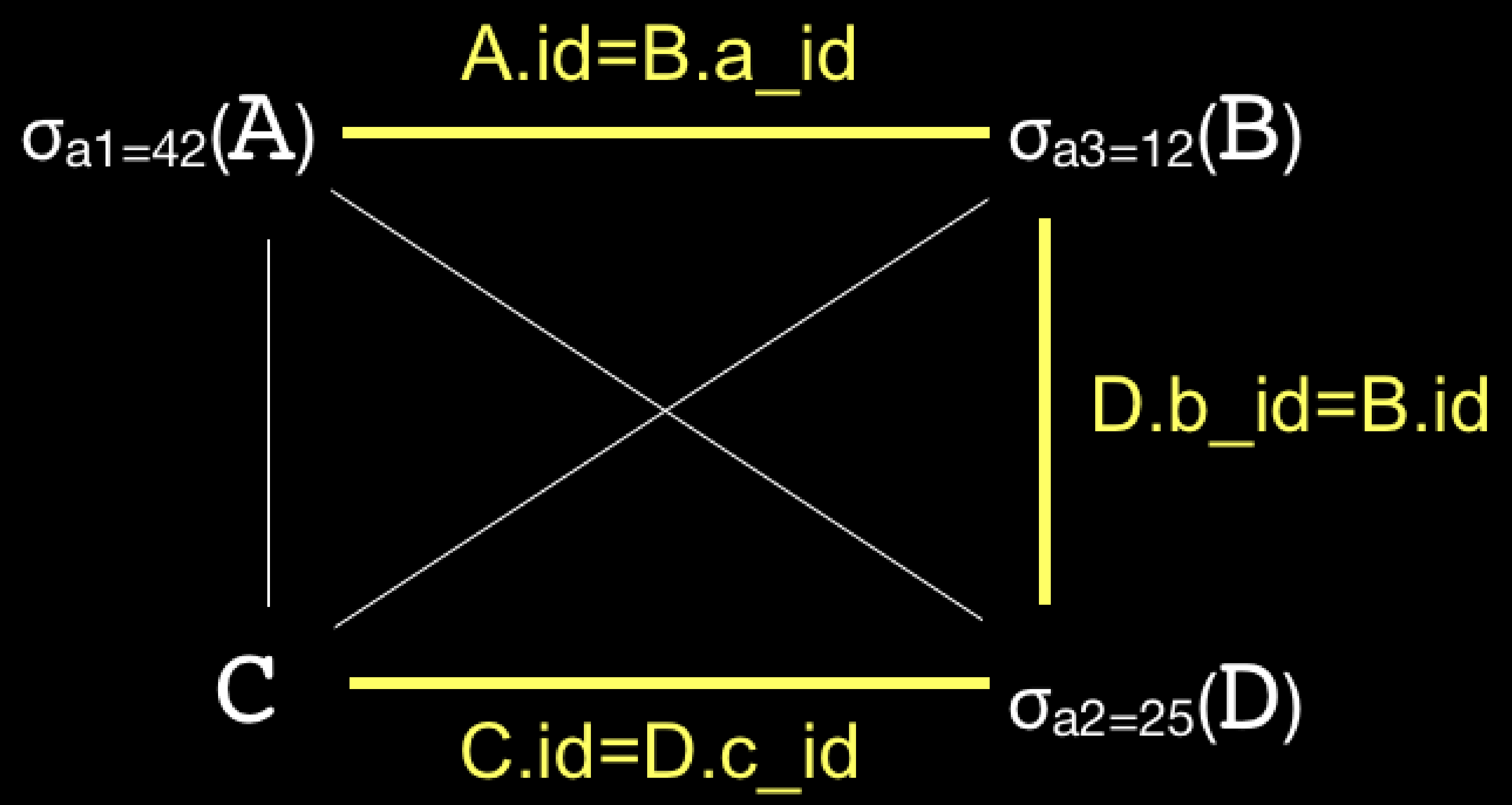


cost models!

Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				~~scan(A)~~, iseek(a1, A), ~~isam(id, A)~~
	X			~~scan(B)~~, iseek(a3, B), ~~isam(id, B)~~
		X		scan(C), ~~isam(id, C)~~
			X	~~scan(D)~~, iseek(a2, D), ~~isam(id, D)~~

Example: Size 1 Plans (pruned)

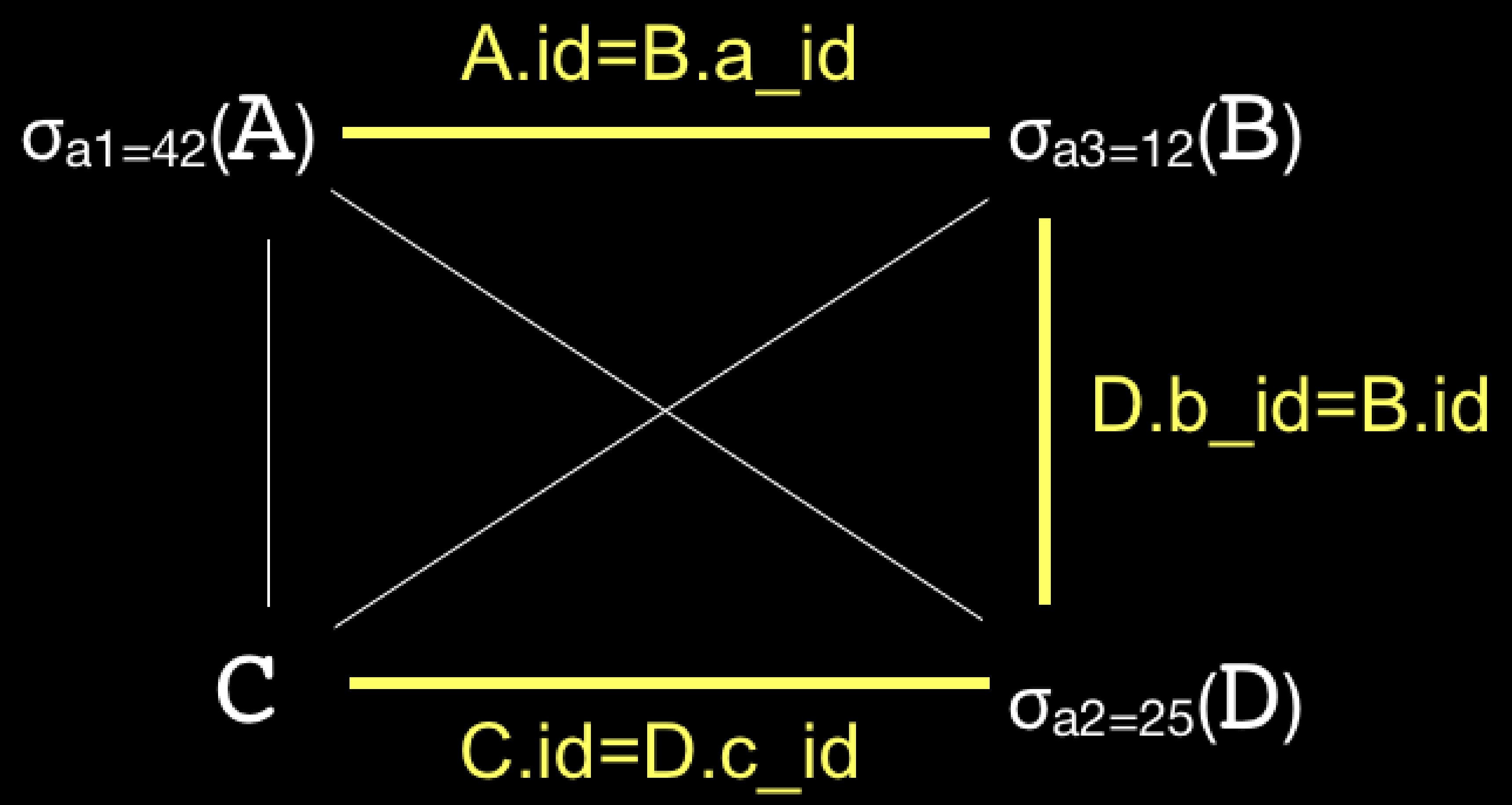
assuming no interesting orders



Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)

Example: Size 2 Plans (generating...)

assuming no interesting orders



$\oplus$: SHJ, INLJ, SMJ

$$\binom{n}{k} = \binom{4}{2} = \frac{4!}{2!(4-2)!}$$

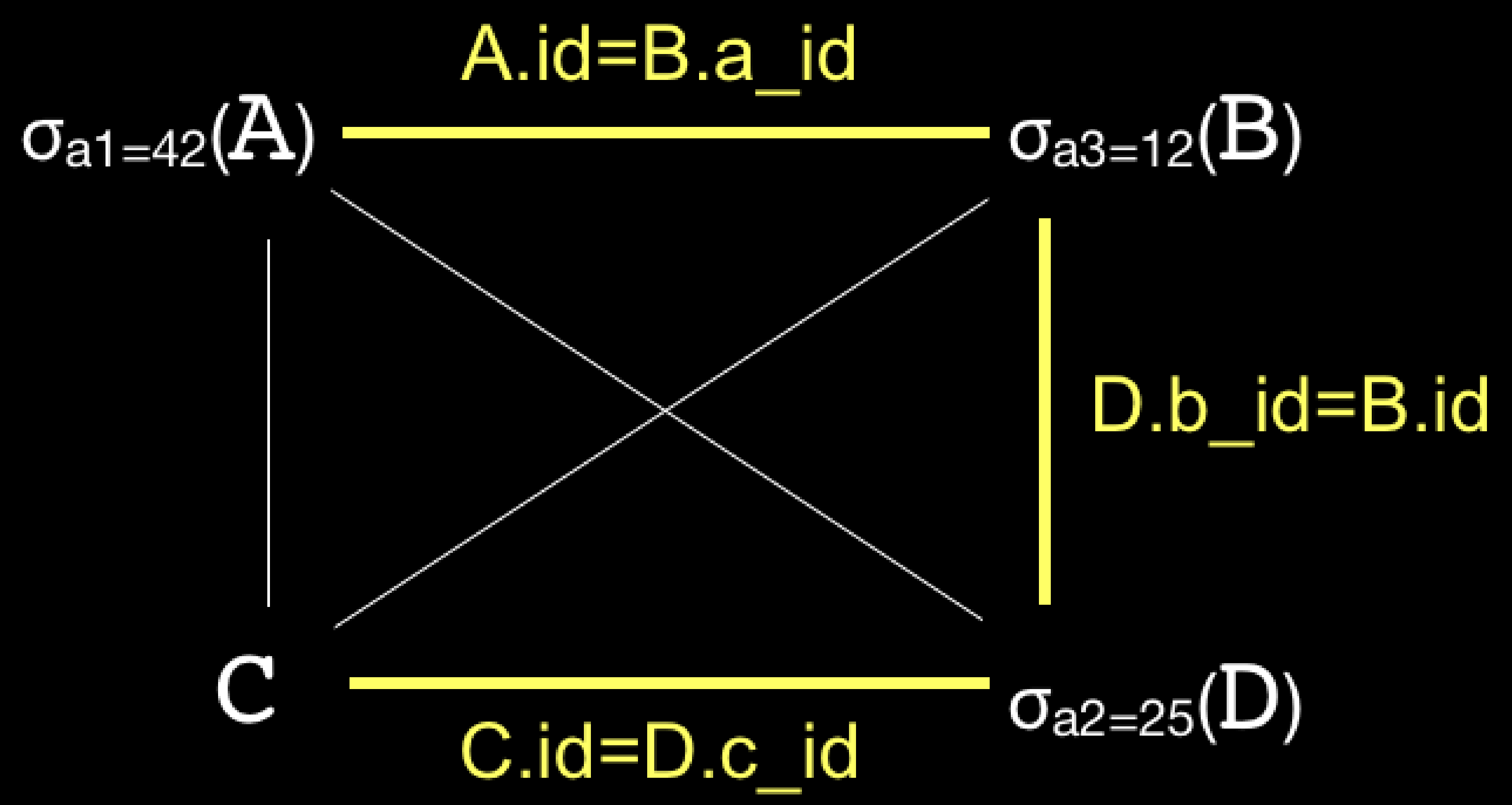
$$= \frac{4!}{2!\,2!} = 2\cdot3$$

$$= 6$$

Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)
X	X			iseek(a1, A) SHJ iseek(a3, B), ...
X		X		iseek(a1, A) CP scan(C), ...
X			X	iseek(a1, A) CP iseek(a2, D), ...
	X	X		iseek(a3, B) CP scan(C), ...
	X		X	iseek(a3, B) SHJ P iseek(a2, D), ...
		X	X	scan(C) SHJ iseek(a2,D), iseek(a2,D) SHJ scan(C), ...

Example: Size 2 Plans (pruning...)

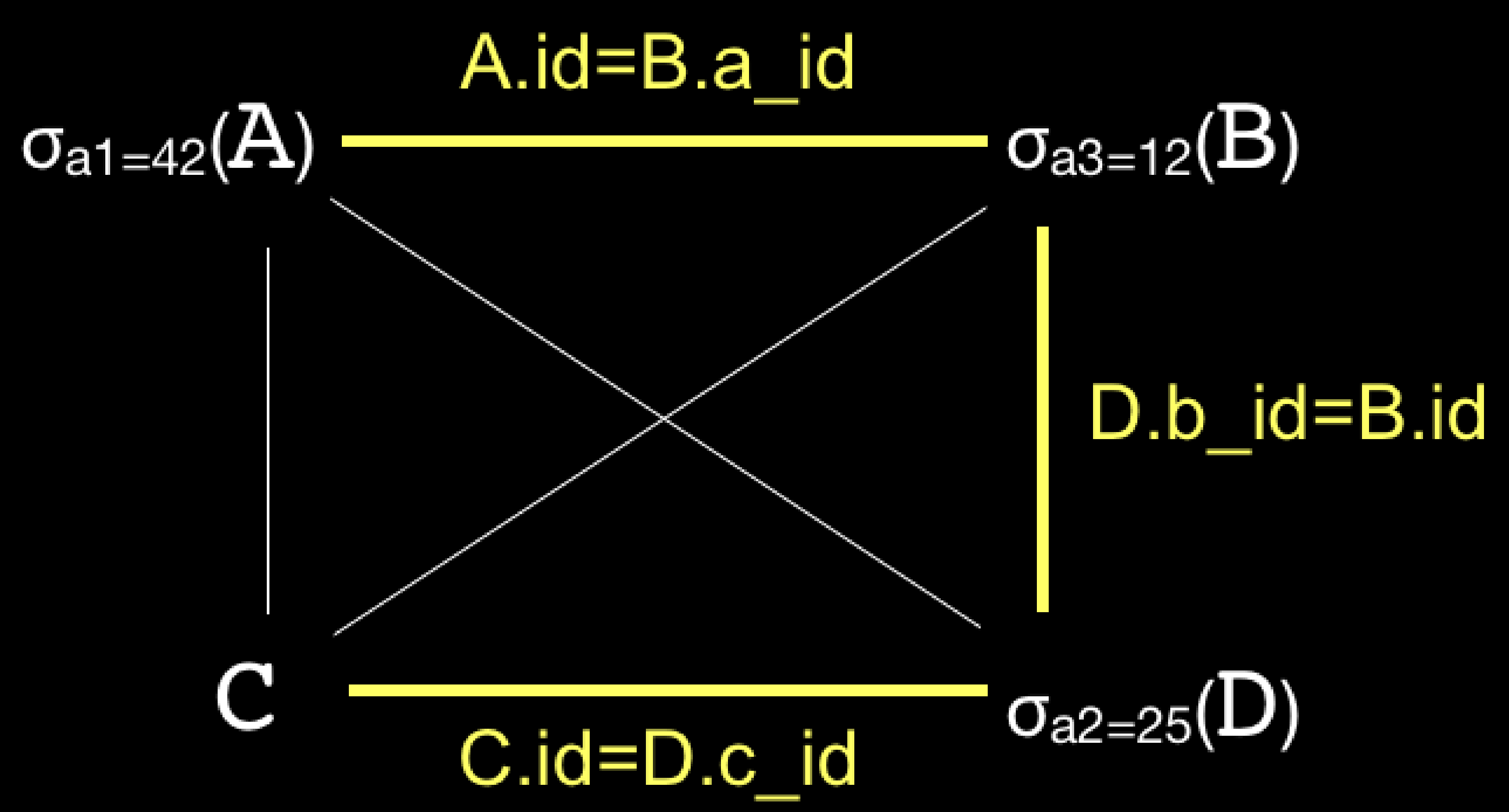
assuming no interesting orders



Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)
X	X			iseek(a1, A) SHJ iseek(a3, B), ...
X		X		iseek(a1, A) CP scan(C), ...
X			X	iseek(a1, A) CP iseek(a2, D), ...
	X	X		iseek(a3, B) CP scan(C), ...
	X		X	iseek(a3, B) SHJ P iseek(a2, D), ...
		X	X	scan(C) SHJ iseek(a2,D), iseek(a2,D) SHJ scan(C), ...

Example: Size 2 Plans (pruned)

assuming no interesting orders

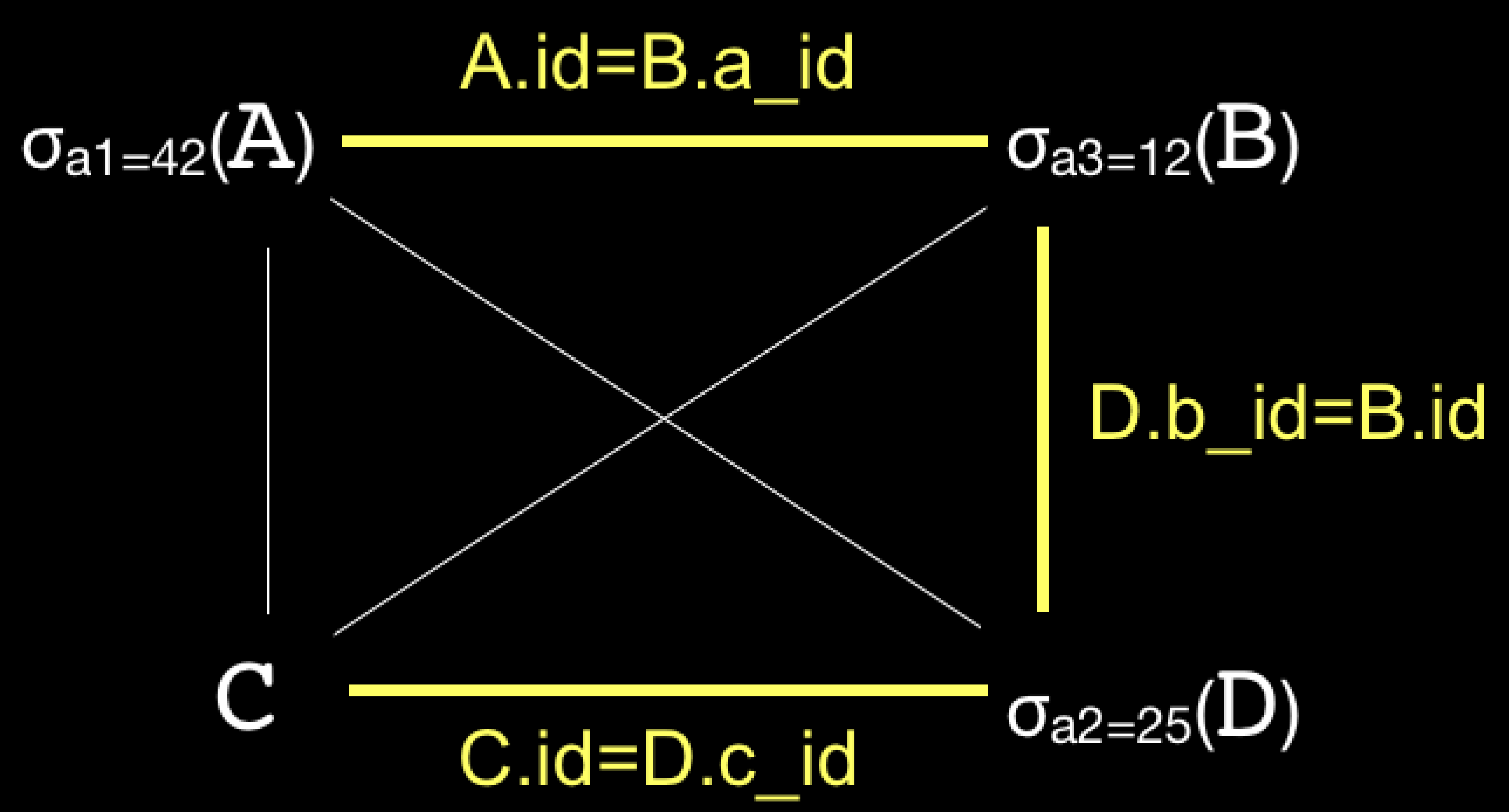


Optimal Subplans

subgraph considered				**best plans**
A	B	C	D	
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)
X	X			iseek(a1, A) SHJ iseek(a3, B)
X		X		iseek(a1, A) CP scan(C)
X			X	iseek(a1, A) CP iseek(a2, D)
	X	X		iseek(a3, B) CP scan(C)
	X		X	iseek(a3, B) SHJ P iseek(a2, D)
		X	X	iseek(a2, D) SHJ scan(C)

Example: Size 3 Plans (pruned)

assuming no interesting orders



Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)
X	X			iseek(a1, A) SHJ iseek(a3, B)
X		X		iseek(a1, A) CP scan(C)
X			X	iseek(a1, A) CP iseek(a2, D)
	X	X		iseek(a3, B) CP scan(C)
	X		X	iseek(a3, B) SHJ P iseek(a2, D)
		X	X	iseek(a2, D) SHJ scan(C)
X	X	X		(iseek(a1, A) SHJ iseek(a3, B)) CP scan(C)
X	X		X	(iseek(a1, A) SHJ iseek(a3, B)) SHJ iseek(a2, D)
X		X	X	(iseek(a2, D) SHJ scan(C)) CP iseek(a1, A)
	X	X	X	(iseek(a3, B) SHJ iseek(a2, D)) SHJ scan(C)

Dynamic Programming

costs(Plan): $\mapsto$ Integer //cost function estimating costs for a plan

DynamicProgramming(R_1, ..., R_n, costs()): //relations R_i; cost function for pruning

```
For i = 1 to n:                                   //i.e. all S ⊂ {R1, ..., Rn} with |S| == 1:
    optPlan[{Ri}] := AccessPlans( Ri );           //get all possible plans for Ri
    prune( optPlan[{Ri}], costs() );              //prune set of plans using cost function

For plansize = 2 to n:                            //for all subplans having at least two inputs
    ForEach S ⊂ {R1, ..., Rn} with |S| == plansize:   //inspect each proper subset S of that size
        optPlan[S] := ∅;                          //initialize optPlan[S] with empty set
        ForEach O ⊂ S:                            //inspect each proper subset O of S
```

$|O| \geq 1$

$|O| < |S|$

Dynamic Programming

```
costs( Plan ):  ↦ Integer                                   //cost function estimating costs for a plan

DynamicProgramming( R1, ..., Rn, costs() ):                 //relations Ri; cost function for pruning
    For i = 1 to n:                                         //i.e. all S ⊂ {R1, ..., Rn} with |S| == 1:
        optPlan[{Ri}] := AccessPlans( Ri );                 //get all possible plans for Ri
        prune( optPlan[{Ri}], costs() );                    //prune set of plans using cost function

    For plansize = 2 to n:                                  //for all subplans having at least two inputs
        ForEach S ⊂ {R1, ..., Rn} with |S| == plansize:     //inspect each proper subset S of that size
            optPlan[S] := ∅;                                //initialize optPlan[S] with empty set
            ForEach O ⊂ S:                                  //inspect each proper subset O of S
                optPlan[S] ∪=                               //extend optPlan[S]-entry to contain...
                    mergePlans( optPlan[O], optPlan[S\O] ); //..the merged plan of two optimal subplans
```

$S := \{R_1, R_2, R_4, R_5\}$

$O := \{R_2, R_5\} \Rightarrow S \setminus O = \{R_1, R_4\}$

Dynamic Programming

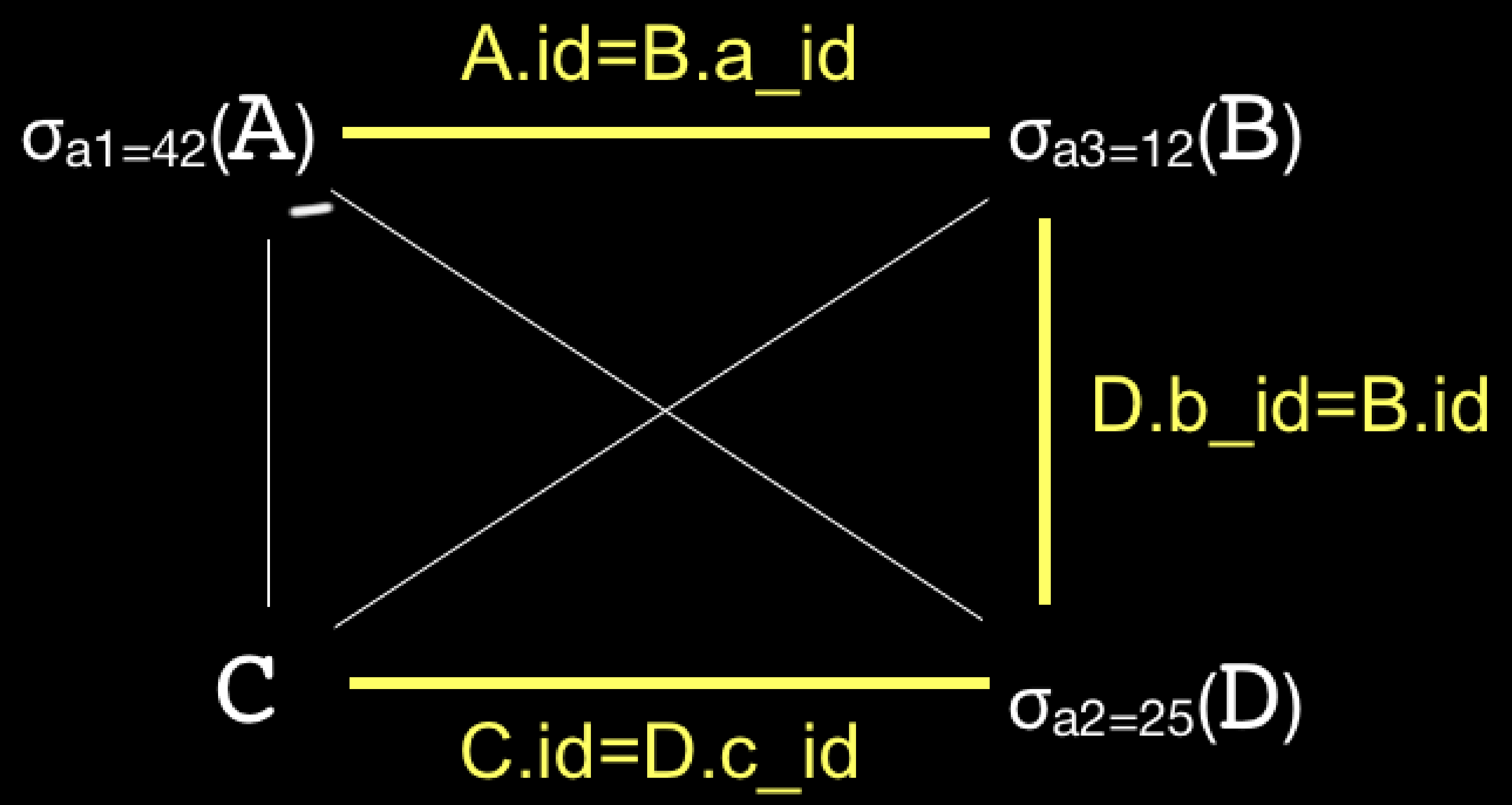
```
costs( Plan ):  ↦ Integer                                  //cost function estimating costs for a plan

DynamicProgramming( R1, ..., Rn, costs() ):                //relations Ri; cost function for pruning
    For i = 1 to n:                                        //i.e. all S ⊂ {R1, ..., Rn} with |S| == 1:
        optPlan[{Ri}] := AccessPlans( Ri );                //get all possible plans for Ri
        prune( optPlan[{Ri}], costs() );                   //prune set of plans using cost function

    For plansize = 2 to n:                                 //for all subplans having at least two inputs
        ForEach S ⊂ {R1, ..., Rn} with |S| == plansize:    //inspect each proper subset S of that size
            optPlan[S] := ∅;                               //initialize optPlan[S] with empty set
            ForEach O ⊂ S:                                 //inspect each proper subset O of S
                optPlan[S] ∪=                              //extend optPlan[S]-entry to contain...
                        mergePlans( optPlan[O], optPlan[S\O] );  //..the merged plan of two optimal subplans
                prune( optPlan[S], costs() );              //prune set of plans using cost function

    prune( optPlan[{R1, ..., Rn}], costs() );              //final pruning, i.e. pick the final plan
    return optPlan[{R1, ..., Rn}];                         //return the final plan
```

So Far: No Interesting Orders



Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				scan(A), iseek(a1, A), isam(id, A)
	X			scan(B), iseek(a3, B), isam(id, B)
		X		scan(C), isam(id, C)
			X	scan(D), iseek(a2, D), isam(id, D)

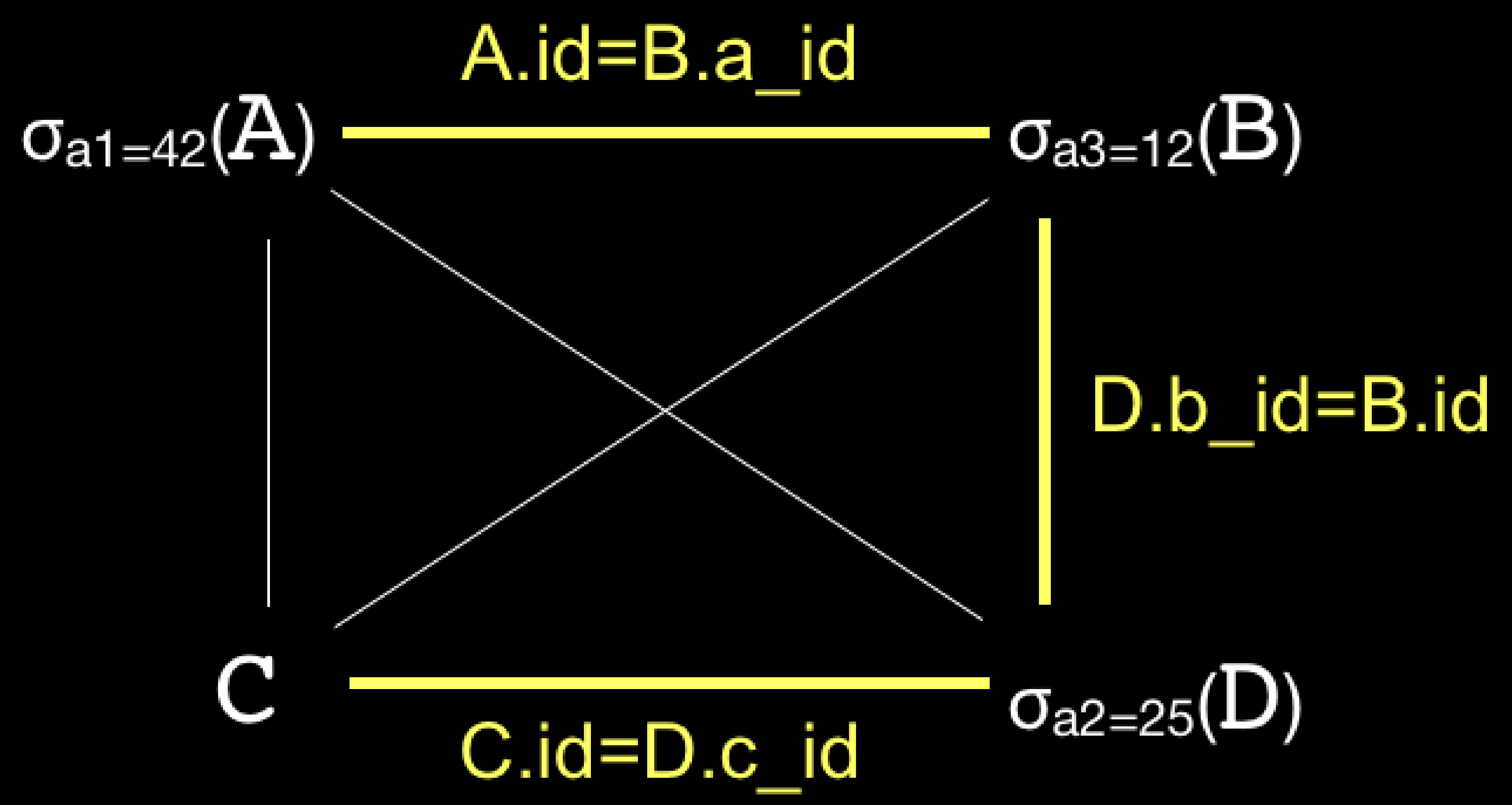
SMJ

isam(A)

sort(a_id)

iseek(a3, 13)

Optimization 1: Interesting Orders

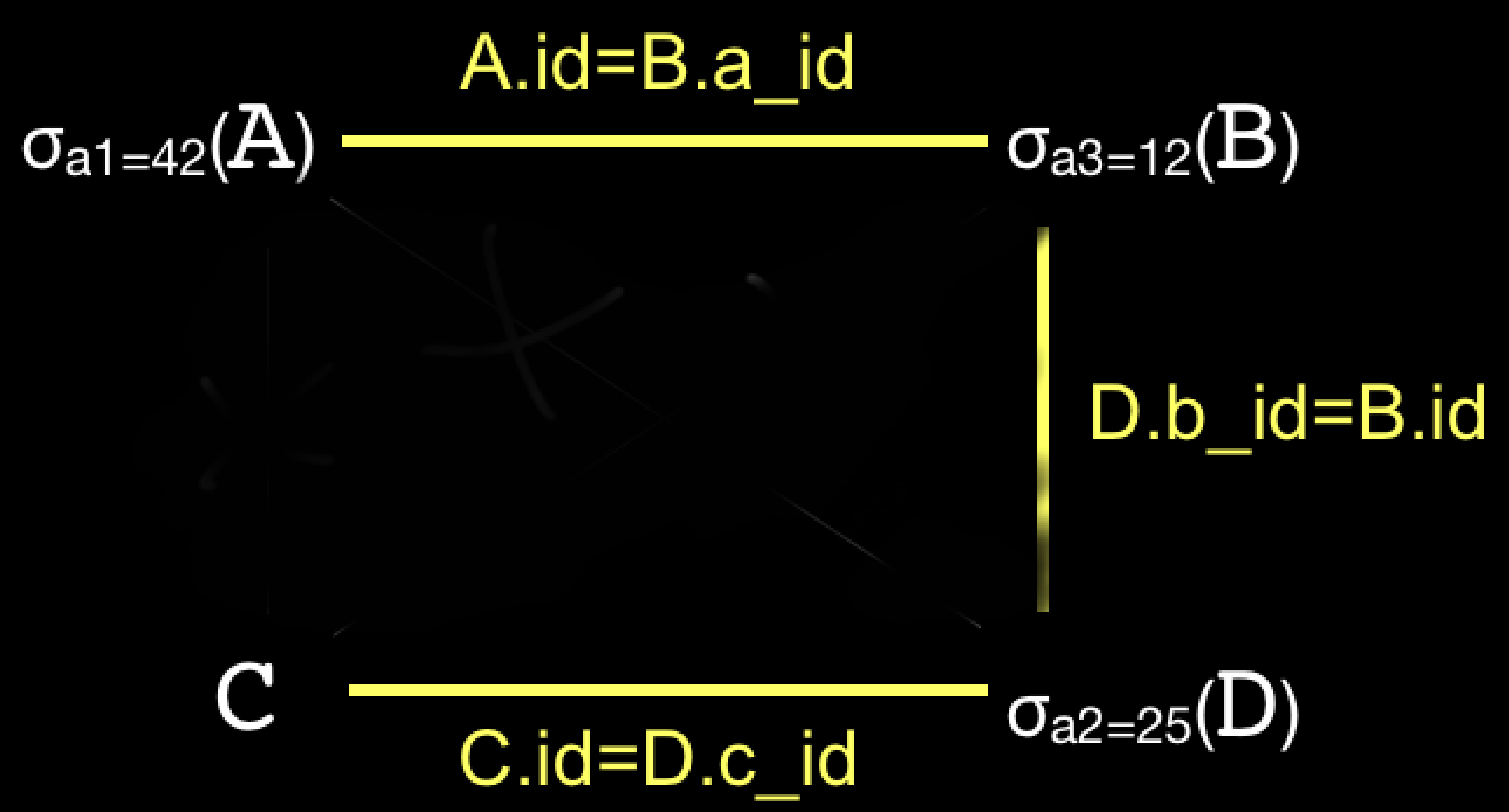


interesting/physical property:

sort order

Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				~~scan~~(A), iseek(a1, A), isam(id, A), iseek(a2, A)
	X			~~scan~~(B), iseek(a3, B), isam(id, B)
		X		~~scan~~(C), isam(id, C)
			X	~~scan~~(D), iseek(a2, D), isam(id, D)
		X	X	isam(id, C) MJ sort(iseek(a2, D)), ...

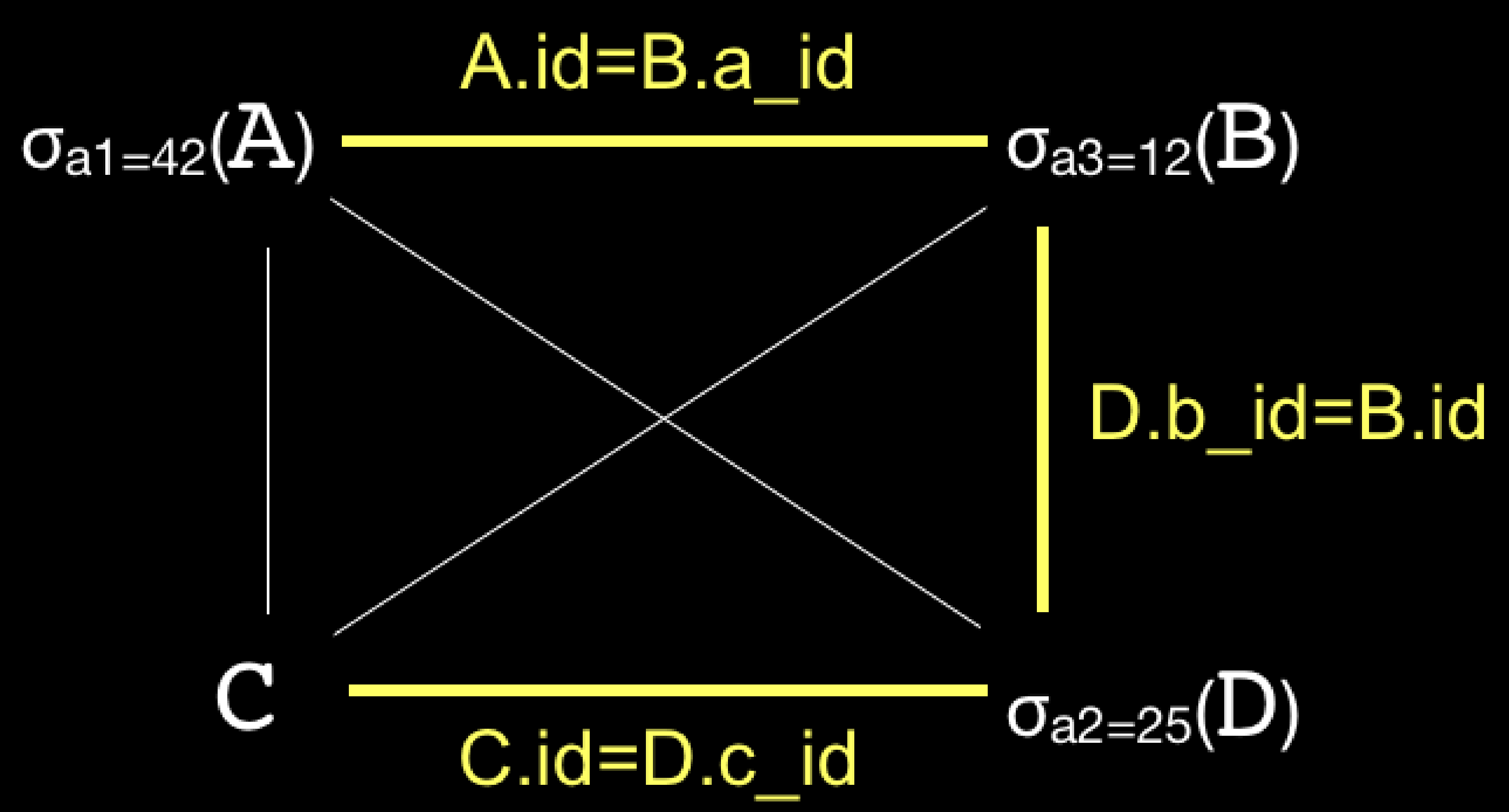
So Far: Not Exploiting Graph Structure



Optimal Subplans

subgraph considered A	B	C	D	best plans
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)
X	X			iseek(a1, A) SHJ iseek(a3, B), ...
X		X		iseek(a1, A) CP scan(C), ...
X			X	iseek(a1, A) CP iseek(a2, D), ...
	X	X		iseek(a3, B) CP scan(C), ...
	X		X	iseek(a3, B) SHJ P iseek(a2, D), ...
		X	X	scan(C) SHJ iseek(a2,D), iseek(a2,D) SHJ scan(C), ...

Optimization 2: Exploit Graph Structure



Optimal Subplans				
subgraph considered				best plans
A	B	C	D	
X				iseek(a1, A)
	X			iseek(a3, B)
		X		scan(C)
			X	iseek(a2, D)
X	X			iseek(a1, A) SHJ iseek(a3, B), ...
X		X		...
X			X	...
	X	X		...
	X		X	iseek(a3, B) SHJ P iseek(a2, D), ...
		X	X	scan(C) SHJ iseek(a2,D), iseek(a2,D) SHJ scan(C), ...