

CPSC 304 – Administrative notes

November 22 and 26, 2024

- Project:
 - Milestone 4: Project implementation – due November 29
 - You cannot change your code after this point!
 - Milestone 5: Group demo – week of December 2
 - Sign up for demos NOW – see Piazza
 - A small number of you have had your project TA changed – check Canvas messages (groups 14, 34, 45, 56, 84)
 - Milestone 6: **Individual** Assessment – Due November 29
- Tutorials: basically project group work time/office hours
 - No tutorials the last week of class (so the TAs have more time to grade/do demos)
- Final exam: December 16 at 12pm! Osborne A

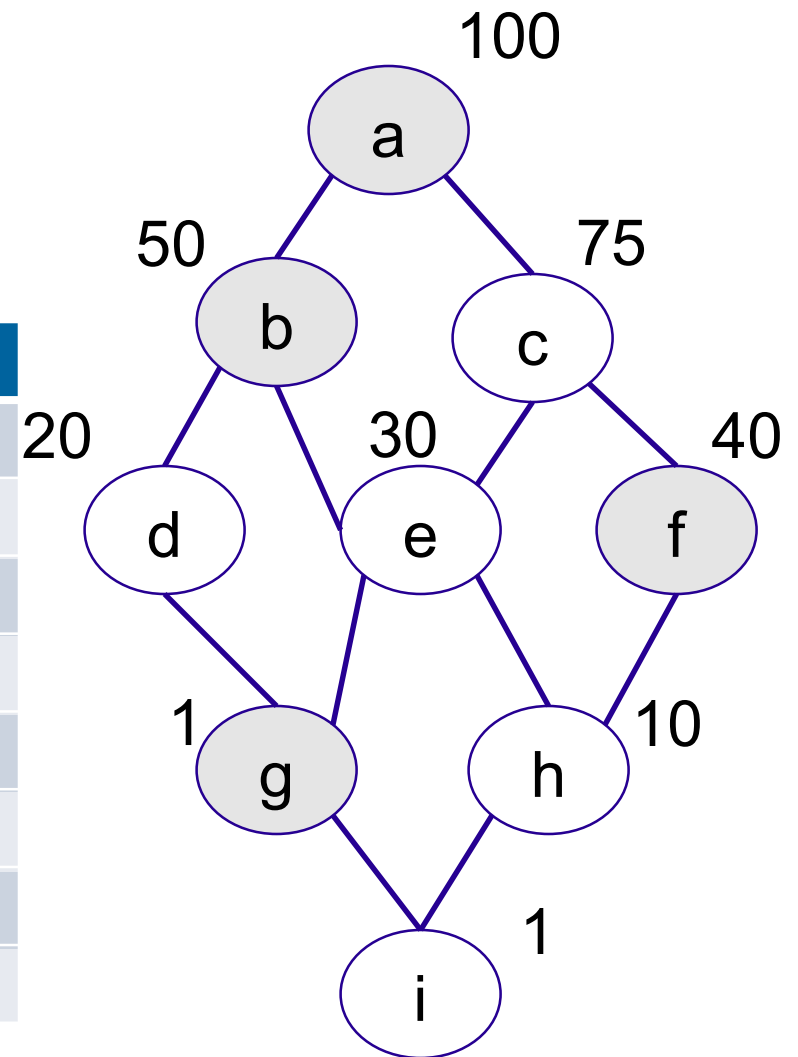
Now where were we...

- We were discussing data warehousing
- We'd figured out, if we're only going to materialize some parts of the warehouse, what to materialize
- We'd done a long example...

In-class Exercise

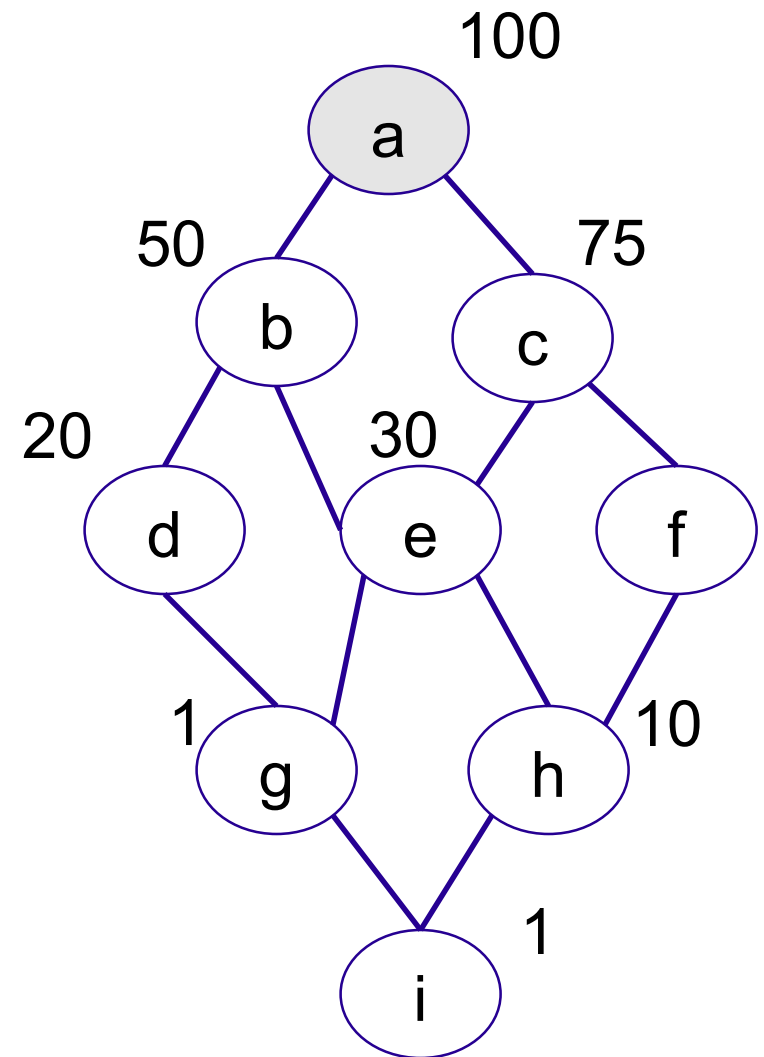
Assuming 'a' is already materialized, what are the best 3 other views that we should materialize?

View	1 st choice	2 nd choice	3 rd choice
b	50*6=300		
c	25*6=150	25*2=50	25*2=50
d	80*3=240	30*3=90	30
e	70*4=280	20*4=80	20*2=40
f	60*3=180	60+10*2=80	60+10=70
g	99*2=198	49*2=98	
h	90*2=180	40*2=80	40
i	99	49	0



A note on workload

- Thus far, we've been assuming that the workload is evenly distributed.
- If it's not, then multiplying by the workload expected at each spot in the lattice will give you a more precise answer
- For example, assume that queries at g and h each make up 20% of the workload, and the remainder of the nodes make 10% each



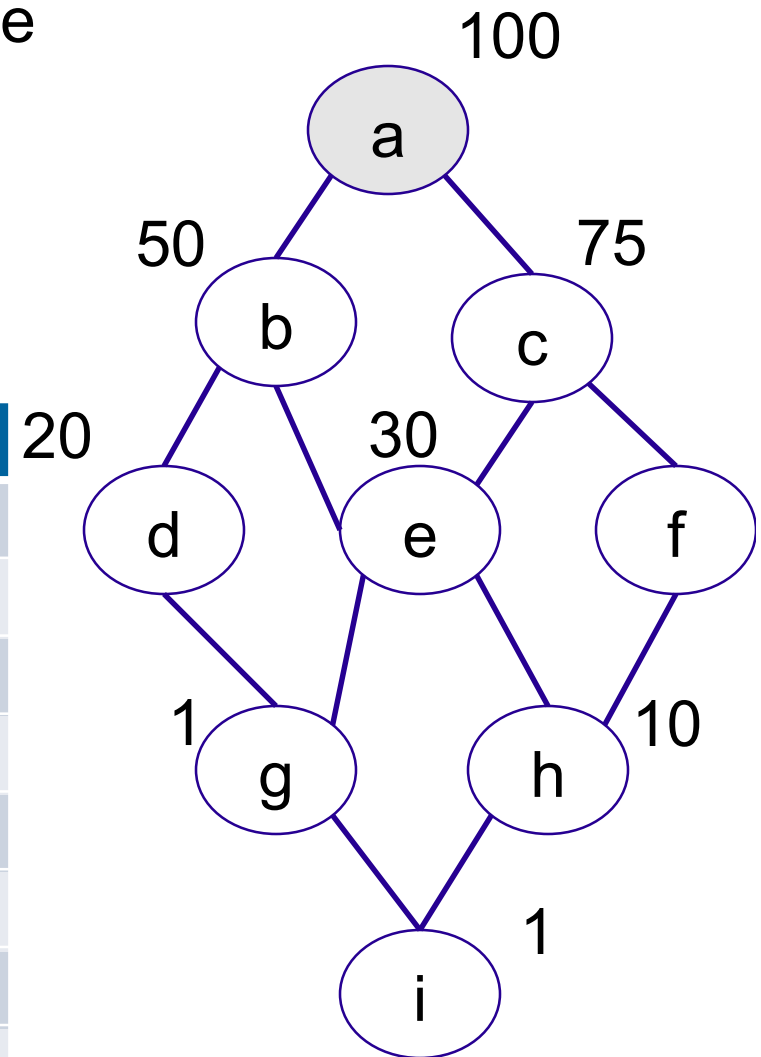
Taking workload into account

- Assume queries at g and h each make up 20% of the workload, and the remainder of the nodes make 10% each

Original:

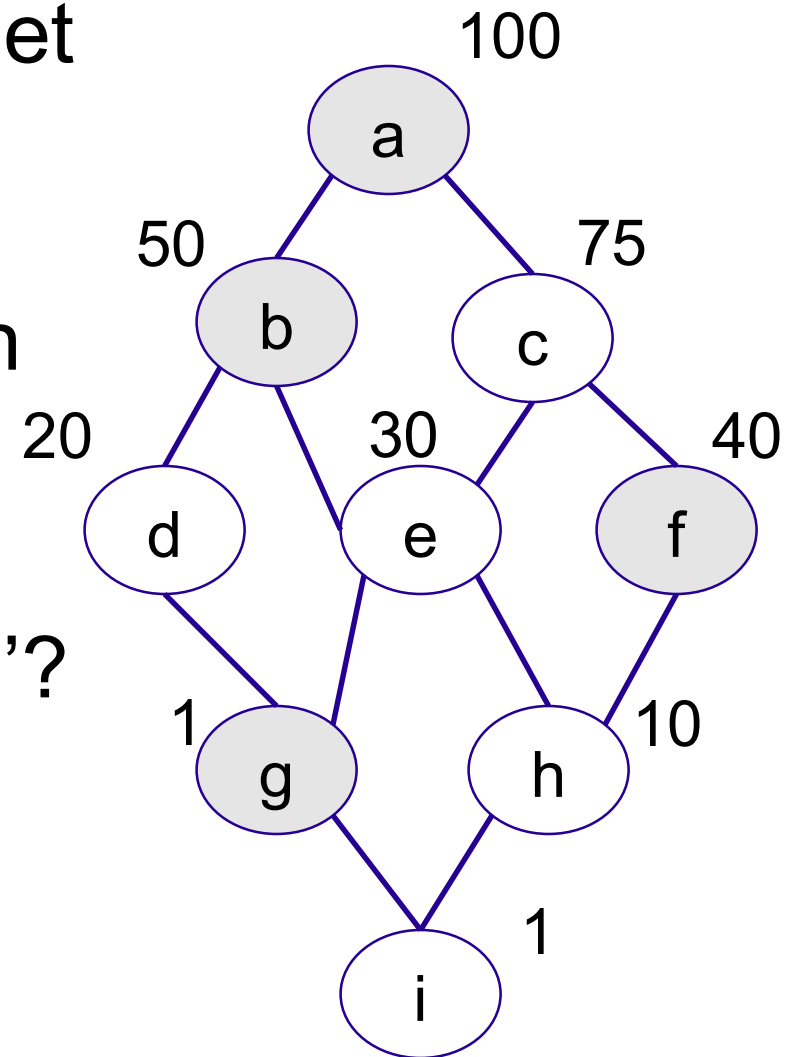
With workload:

View	1 st choice	View	1 st choice
b	$50 \times 6 = 300$	b	$.2 \times 2 \times 50 + .1 \times 4 \times 50 = 40$
c	$25 \times 6 = 150$	c	$.2 \times 2 \times 25 + .1 \times 4 \times 25 = 20$
d	$80 \times 3 = 240$	d	$.2 \times 1 \times 80 + .1 \times 2 \times 80 = 32$
e	$70 \times 4 = 280$	e	$.2 \times 2 \times 70 + .1 \times 2 \times 70 = 42$
f	$60 \times 3 = 180$	f	$.2 \times 1 \times 60 + .1 \times 2 \times 60 = 24$
g	$99 \times 2 = 198$	g	$.2 \times 1 \times 99 + .1 \times 1 \times 99 = 29.7$
h	$90 \times 2 = 180$	h	$.2 \times 1 \times 90 + .1 \times 1 \times 90 = 27$
i	99	i	$.1 \times 99 = 9.9$



Using the Materialized Views

- Once we have chosen a set of views, we need to consider how they can be used to answer queries on other views.
- What is the best way to answer queries on view 'h'?



Issues in View Materialization:

Incremental recomputing

- How do we maintain views incrementally without recomputing them from scratch?
 - Two steps
 - Modify the changes to the view when the data changes
 - Apply only those changes to the materialized view
- There may be challenges in refreshing, especially if the base tables are distributed across multiple locations

Issues in View Materialization:

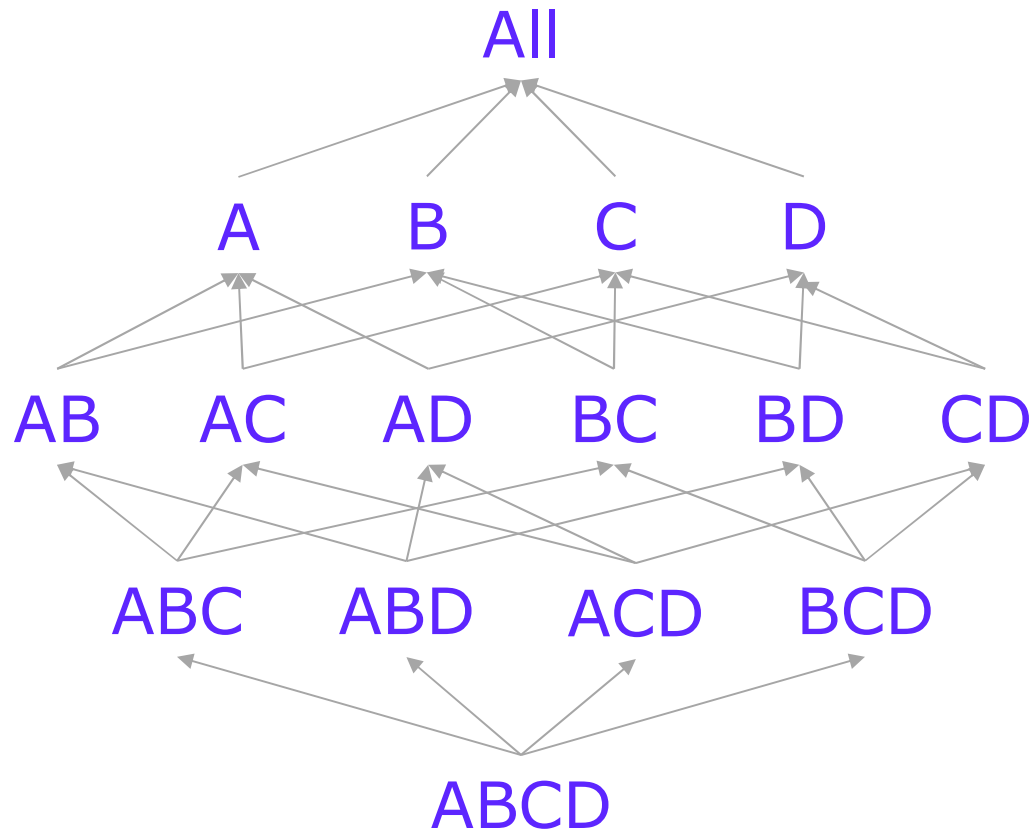
Handling updates

- How should we refresh and maintain a materialized view when an underlying table is modified?
- **Maintenance policy:** Controls when we refresh
 - **Immediate:** As part of the transaction that modifies the underlying data tables
 - + Materialized view is always consistent
 - Updates are slow
 - **Deferred:** Some time later, in a separate transaction
 - View is inconsistent for a while
 - + Can scale to maintain many views without slowing updates

Deferred Maintenance

- Three flavors:
 - **Lazy**: Delay refresh until next query on view; then refresh before answering the query.
 - This approach slows down queries rather than updates, in contrast to immediate maintenance.
 - **Periodic (Snapshot)**: Refresh periodically. Queries possibly answered using outdated version of view tuples. Also widely used in asynchronous replication in distributed databases
 - **Event-based (Forced)**: e.g., Refresh after a fixed number of updates to underlying data tables

That's how to materialize *some* of the views. What if we want *all* views?



Given:

- A cube to materialize
- For each view in the cube:
 - Cost to materialize the view if the inputs are already sorted (A)
 - Cost to materialize the view if the inputs are NOT already sorted (S)
- Need completely different algorithm!

The difference between which views to materialize and how to materialize all views.

- Previously, we looked at how to choose which views to materialize
 - There we were optimizing what is the cost to query the cube
- Now we are materializing all elements, we're just figuring out the cheapest way to do it
 - Querying the cube will cost the same no matter what order – everything is materialized
 - The ordering of the attributes at each node in the lattice matters!

Let's revisit our student example.

Consider group by {standing, age}

```
SELECT standing, age, count(*)
FROM student
GROUP BY standing, age
ORDER BY standing, age
```

ST	AGE	COUNT(*)
FR	17	2
FR	18	4
JR	18	1
JR	19	1
JR	20	4
SO	17	1
SO	18	1
SO	19	3
SR	19	1
SR	20	1
SR	21	4
SR	22	1

12 rows selected.

```
SELECT age, standing, count(*)
FROM student
GROUP BY age, standing
ORDER BY age, standing
```

AGE	ST	COUNT(*)
17	FR	2
17	SO	1
18	FR	4
18	JR	1
18	SO	1
19	JR	1
19	SO	3
19	SR	1
20	JR	4
20	SR	1
21	SR	4
22	SR	1

12 rows selected.

There are two ways to compute this – {standing, age}, and {age, standing}. Both have the same # of tuples and the same information. But the ordering makes a difference on which of age or standing is easier to compute next

Pipelining

- Pipelining means that I can keep working without having to wait for an operation to entirely finish before doing the next one.
- In this case, if tuples are sorted in the right order, I can start the next operation early, and probably at a lower cost.

Let's revisit our student example.

Consider group by {standing, age}

```
SELECT standing, age, count(*)
FROM student
GROUP BY standing, age
ORDER BY standing, age
```

ST	AGE	COUNT(*)
FR	17	2
FR	18	4
JR	18	1
JR	19	1
JR	20	4
SO	17	1
SO	18	1
SO	19	3
SR	19	1
SR	20	1
SR	21	4
SR	22	1

12 rows selected.

```
SELECT age, standing, count(*)
FROM student
GROUP BY age, standing
ORDER BY age, standing
```

AGE	ST	COUNT(*)
17	FR	2
17	SO	1
18	FR	4
18	JR	1
18	SO	1
19	JR	1
19	SO	3
19	SR	1
20	JR	4
20	SR	1
21	SR	4
22	SR	1

12 rows selected.

If I compute {standing, age}, I can pipeline the computation of standing – I can just send the tuples on without having to resort. Super fast! But to compute the answers for standing from {age, standing} I have to resort.

Let's revisit our student example.

Consider group by {standing, age}

```
SELECT standing, age, count(*)
FROM student
GROUP BY standing, age
ORDER BY standing, age
```

ST	AGE	COUNT(*)
FR	17	2
FR	18	4
JR	18	1
JR	19	1
JR	20	4
SO	17	1
SO	18	1
SO	19	3
SR	19	1
SR	20	1
SR	21	4
SR	22	1

12 rows selected.

```
SELECT age, standing, count(*)
FROM student
GROUP BY age, standing
ORDER BY age, standing
```

AGE	ST	COUNT(*)
17	FR	2
17	SO	1
18	FR	4
18	JR	1
18	SO	1
19	JR	1
19	SO	3
19	SR	1
20	JR	4
20	SR	1
21	SR	4
22	SR	1

12 rows selected.

Similarly, if I compute {age,standing}, I can pipeline the computation of age – I can just send the tuples on without having to resort. Super fast! But to compute the answers for age from {standing, age}, I have to resort.

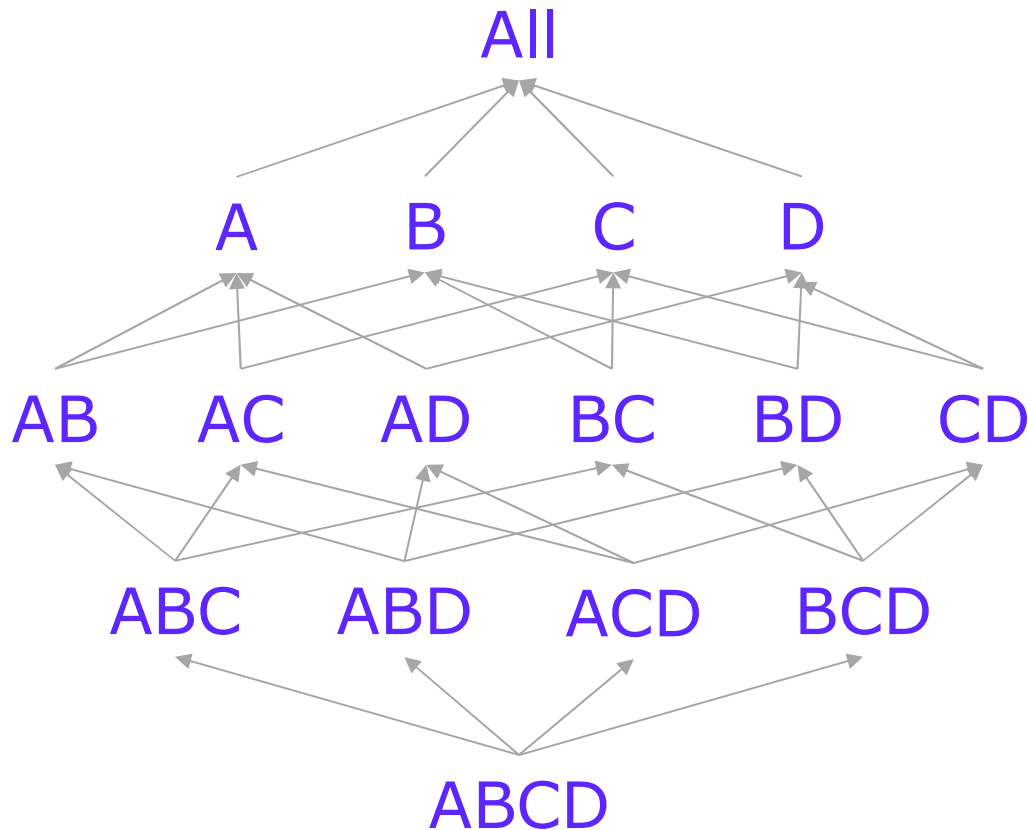
Administrative Notes

April 6, 2021

- Upcoming due dates:
 - April 5-9: Project milestone 5
 - April 9: Tutorial 8 (SQL Server)
 - Start early; there is a lot of configuration required
 - Go to tutorial (any tutorial) if you are having issues
 - April 13: Project milestone 6
- Office hours with Jessica Thursday after class to talk about the exam. We will be using a waiting room for this office hour so that you can talk in private.
- Posted office hour schedule will end on the last day of classes. We will have a new schedule for the final exam period.

The PipeSort Algorithm

[Agarwal et al., VLDB 1996]



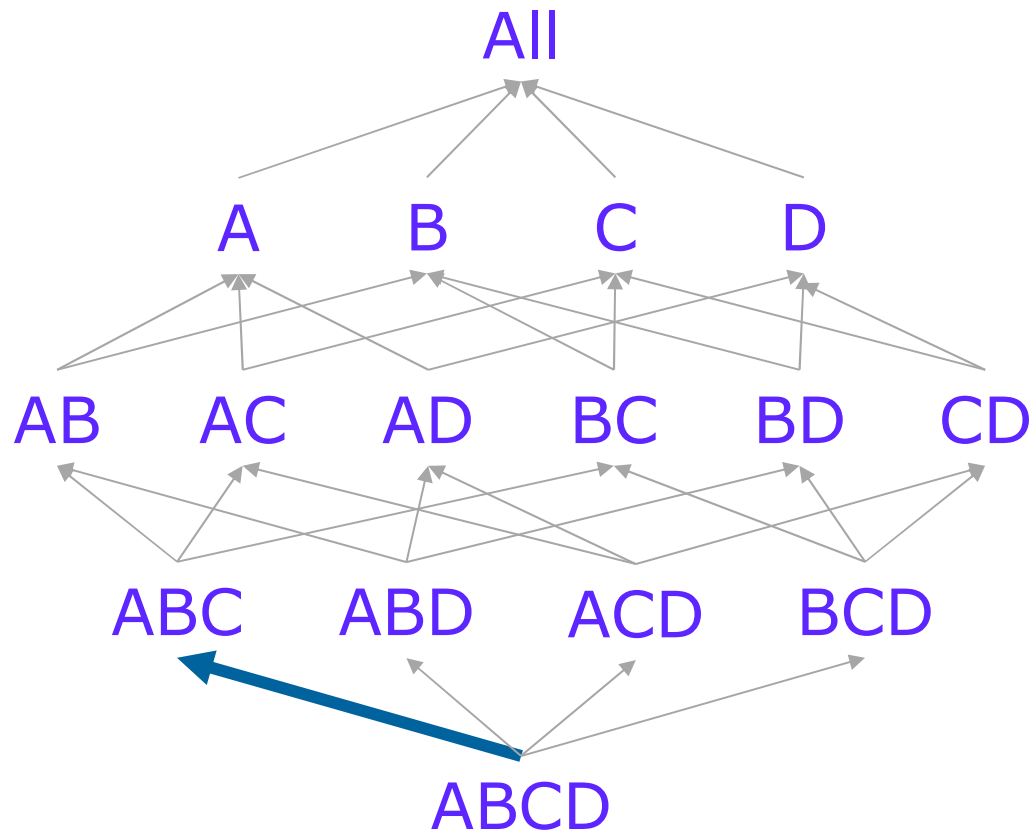
Output:

- Exactly which views to materialize in which order

Key ideas:

- Pipelining the computation is cheaper
- Ordering matters to the pipelining but NOT the cube.
- Compute each view once
- Greedy works well

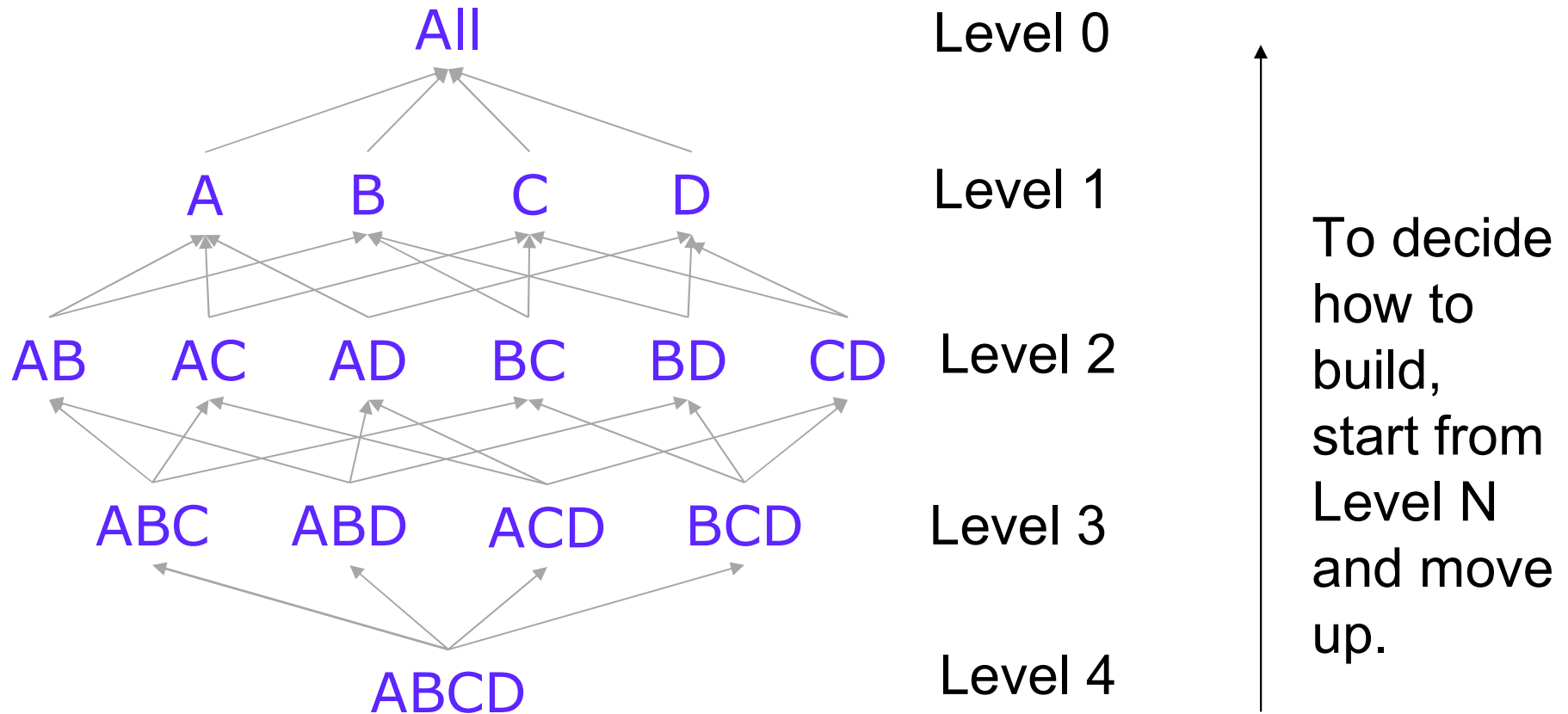
Example of sorting issues



Take the current ordering of the views

- Given that we have computed ABCD:
 - Computing ABC can be pipelined (i.e., cheap)
 - Computing ABD, ACD, and BCD cannot be pipelined (i.e., less cheap)

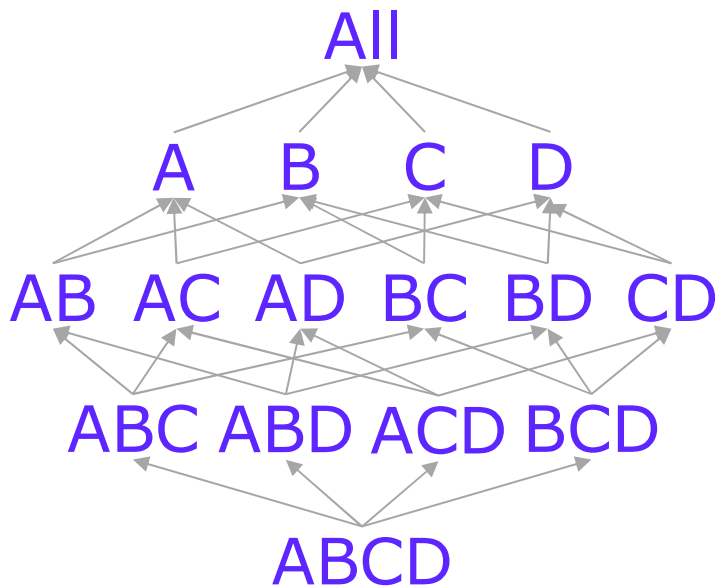
Pipesort



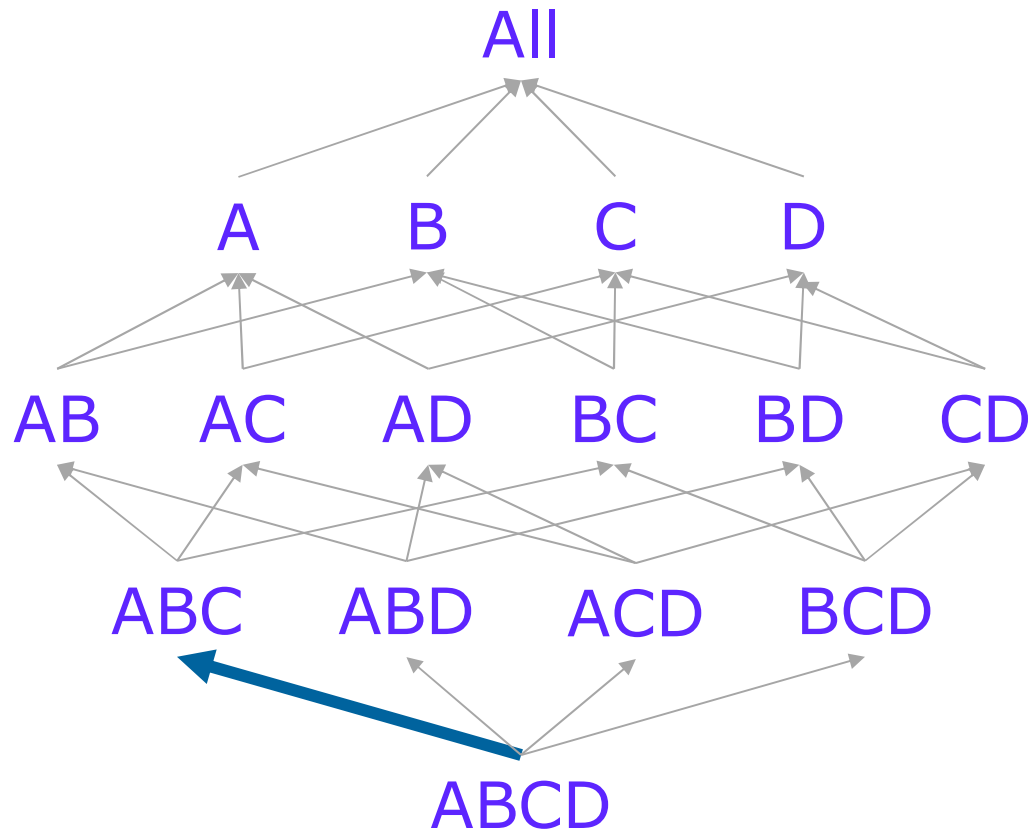
Find the best way to construct level k from level $k+1$.

Ordering Matters!

- ABC means the tuples are ordered by A, B, and then C
- The ordering of the view has been determined during the previous round



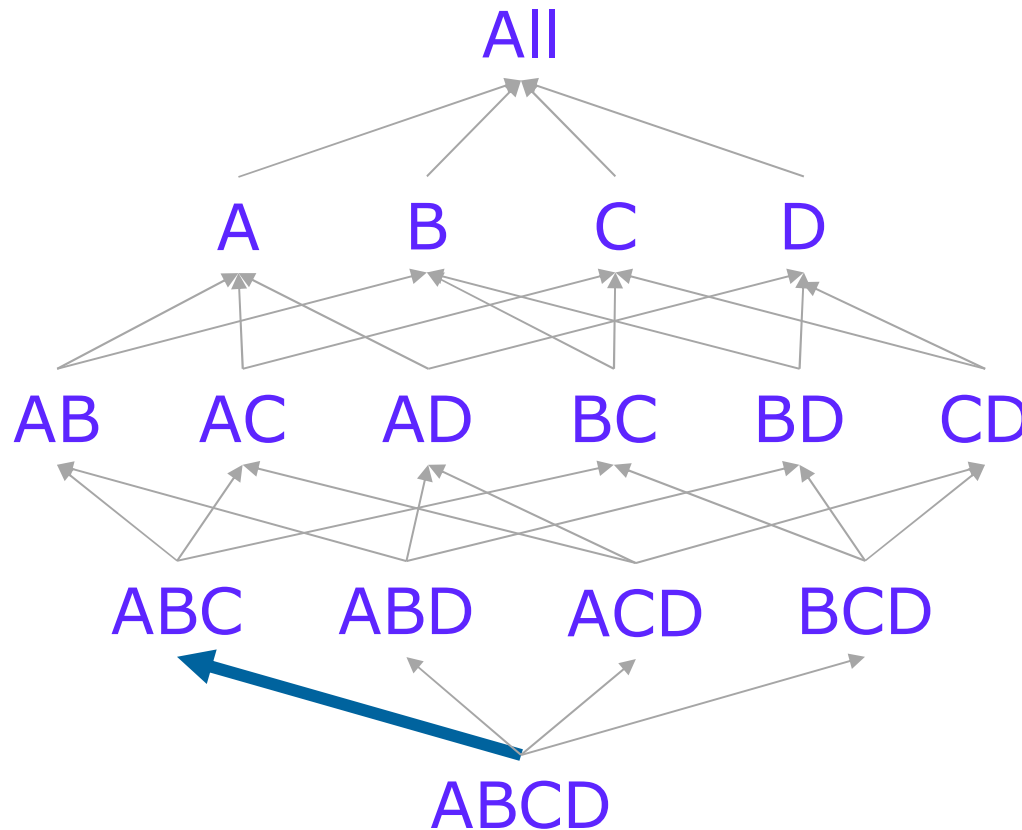
Clicker question



Assume ABC has been materialized in that order. Which materializations can take full advantage of this for PipeSort?

- A. AB
- B. AC
- C. BC
- D. All of the above
- E. None of the above


Clicker question



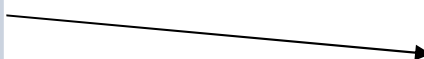
Assume ABC has been materialized in that order. Which materializations can take full advantage of this for PipeSort?

- A. AB
- B. AC
- C. BC
- D. All of the above
- E. None of the above

Clicker question explanation




a	b	c	d	count(*)
1	1	1	1	3
1	1	1	2	2
1	1	2	1	4
1	3	4	1	3
1	3	4	2	2
1	3	4	3	1
1	4	1	1	4
1	4	1	2	3




a	b	c	count(*)

a	b	count(*)

Clicker question explanation




a	b	c	d	count(*)
1	1	1	1	3
1	1	1	2	2
1	1	2	1	4
1	3	4	1	3
1	3	4	2	2
1	3	4	3	1
1	4	1	1	4
1	4	1	2	3



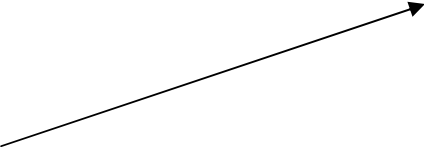
a	b	c	count(*)

a	b	count(*)

Clicker question explanation



a	b	c	d	count(*)
1	1	1	1	3
1	1	1	2	2
1	1	2	1	4
1	3	4	1	3
1	3	4	2	2
1	3	4	3	1
1	4	1	1	4
1	4	1	2	3

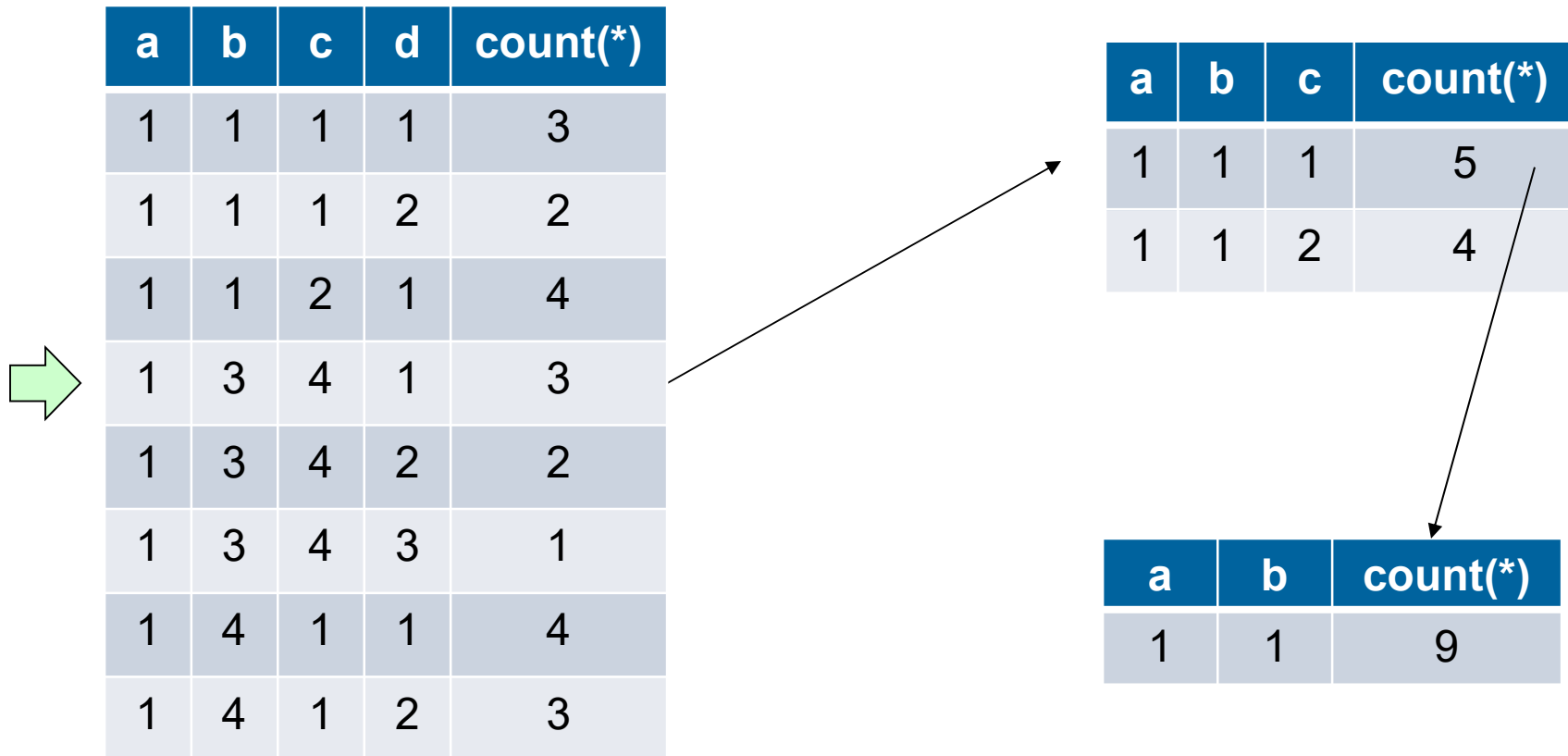


a	b	c	count(*)
1	1	1	5

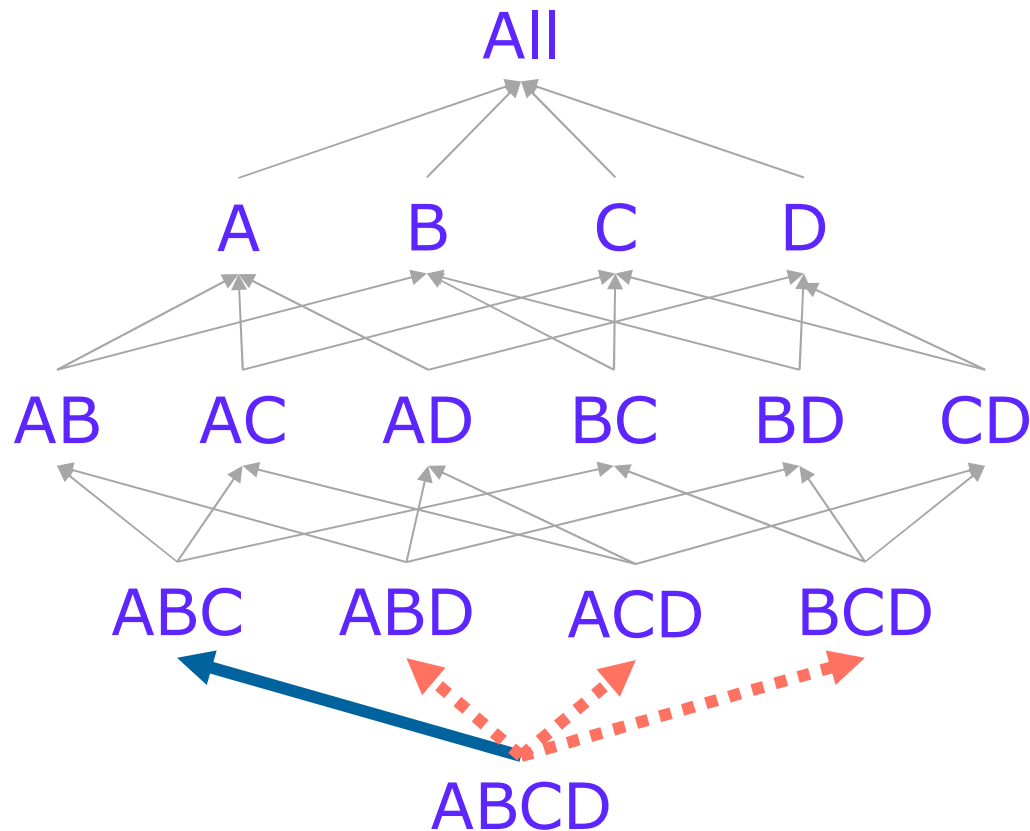
All of the (a, b, c) tuples with (1, 1, 1) have been pipelined to view ABC so we can generate a tuple for this view.

a	b	count(*)

Clicker question explanation



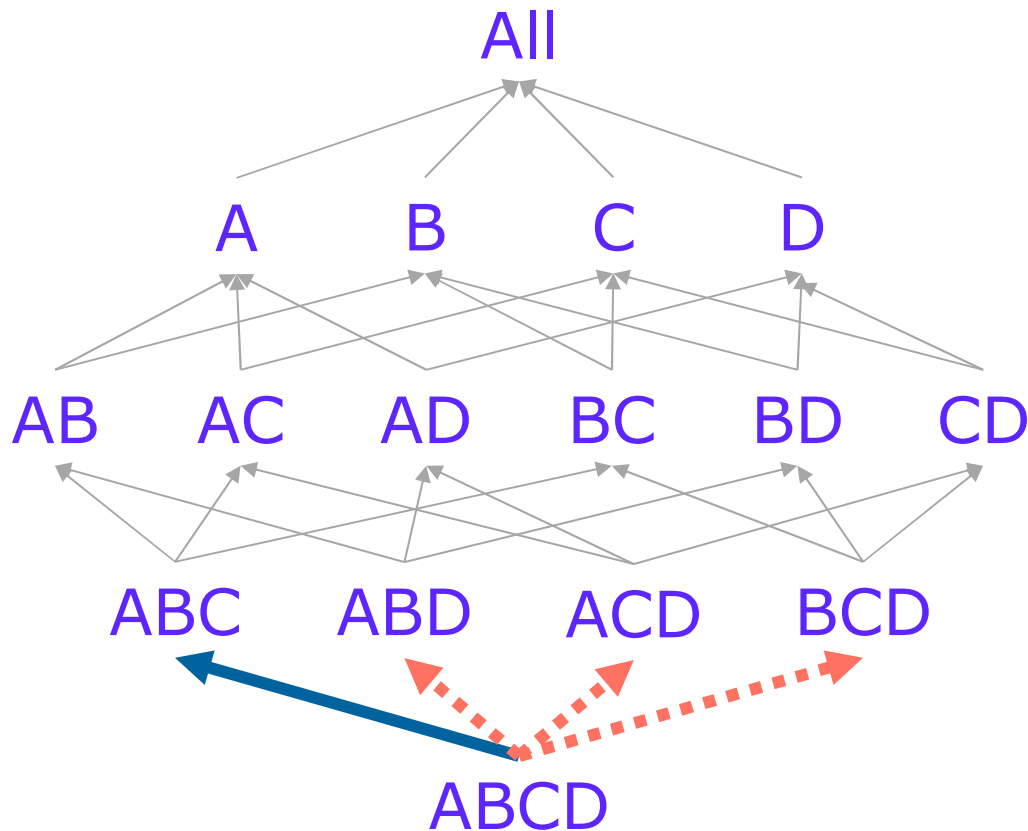
Greedy works really well



← A Edge (sorted) ← S Edge (not sorted)

- At each level, look at the costs to compute the next level
- Use A edges if already sorted
- Use S edges if not already sorted
- Each view can be the origin of **one** A edge
- Each view can be the origin of as many S edges as necessary
- Greedy: minimize per-level cost

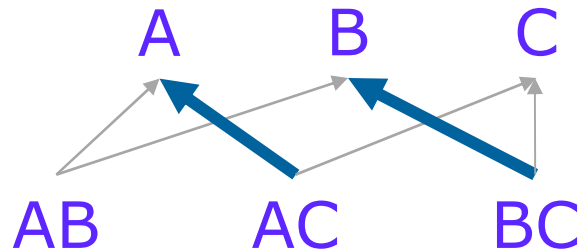
Valid solutions & Costs



← A Edge (sorted) ← S Edge (not sorted)

- Valid solution:
 - Each view in the lower level has to be the origin of at most one A edge (multiple S edges are okay)
 - Each view at the upper level has to be the destination of at least one edge (A or S)
- Cost:
 - Sum of the cost of all A and S edges (if more than one S edge leaves a view, need to include the cost twice, because may need to resort)

Is this a valid solution? What is its cost? What would this mean?



A cost:	2	5	13
S cost:	10	12	20

This would mean that we compute AC in the order AC. The results are used to compute A. Ditto for BC and B.

← A Edge (sorted) ← S Edge (not sorted)

Is this a valid solution?

A. Yes

B. No

No. It does not tell us how to calculate C.

What is its cost?

A. 5

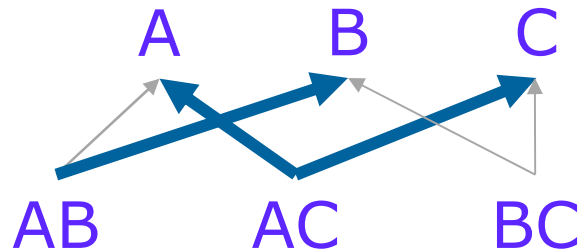
B. 13

C. 17

D. 18

E. None of the above

Is this a valid solution? What is its cost? What would it mean?



A cost: 2 5 13
S cost: 10 12 20

This would mean that we compute AB in the order BA. The results are used to compute B. We would compute AC in the order AC and the order CA, but we only want to compute it once.

← A Edge (sorted) ← S Edge (not sorted)

Is this a valid solution?

A. Yes

B. No

No. Can't have two A edges from AC

What is its cost?

A. 14

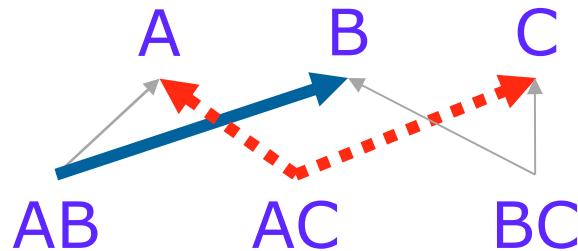
B. 19

C. 26

D. None of the above

$2+5+5 = 12$

Is this a valid solution? What is its cost? What would it mean?



A cost:	2	5	13
S cost:	10	12	20

This would mean that we compute AB in the order BA. The results are used to compute B. We would compute AC in some random manner and then resort to compute A and C

← A Edge (sorted) ← S Edge (not sorted)

Is this a valid solution?

A. Yes

B. No

Yes, but you'd probably not use it because you might as well use an A edge to compute A or C – hard to imagine as lowest cost

What is its cost?

A. 14

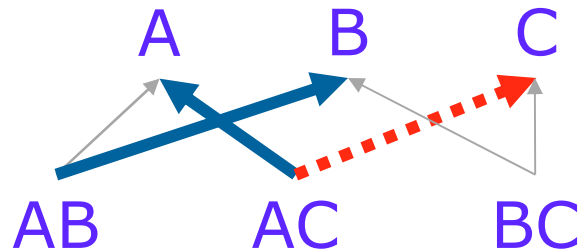
B. 19

C. 26

D. None of the above

$2+12+12 = 26$

Is this a valid solution? What is its cost? What would it mean?



A cost:	2	5	13
S cost:	10	12	20

This would mean we compute AB in the order BA. The results are used to compute B. We would compute AC in order AC would be used to compute A. We would resort AC to compute C.

← A Edge (sorted) ← S Edge (not sorted)

Is this a valid solution?

A. Yes

B. No

Yes, totally valid.

What is its cost?

A. 14

B. 19

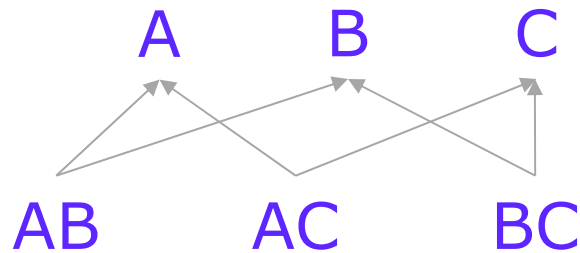
C. 26

D. 29

E. None of the above

$2+5+12 = 19$

Group exercise: determine minimum cost from lower to upper levels



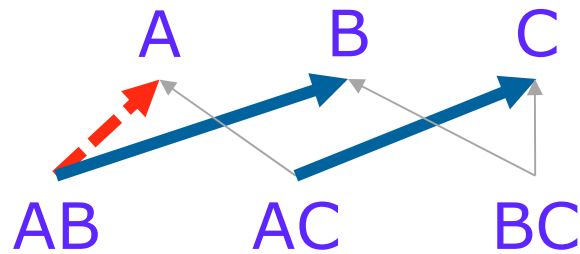
A cost:	2	5	13
S cost:	10	12	20

Clicker question: Which of the following **S** edges did you use?

- A. $AC \rightarrow A$
- B. $AC \rightarrow C$
- C. $BC \rightarrow B$
- D. All of the above
- E. None of the above

← A Edge (sorted) ← S Edge (not sorted)

Group exercise: determine minimum cost from lower to upper levels

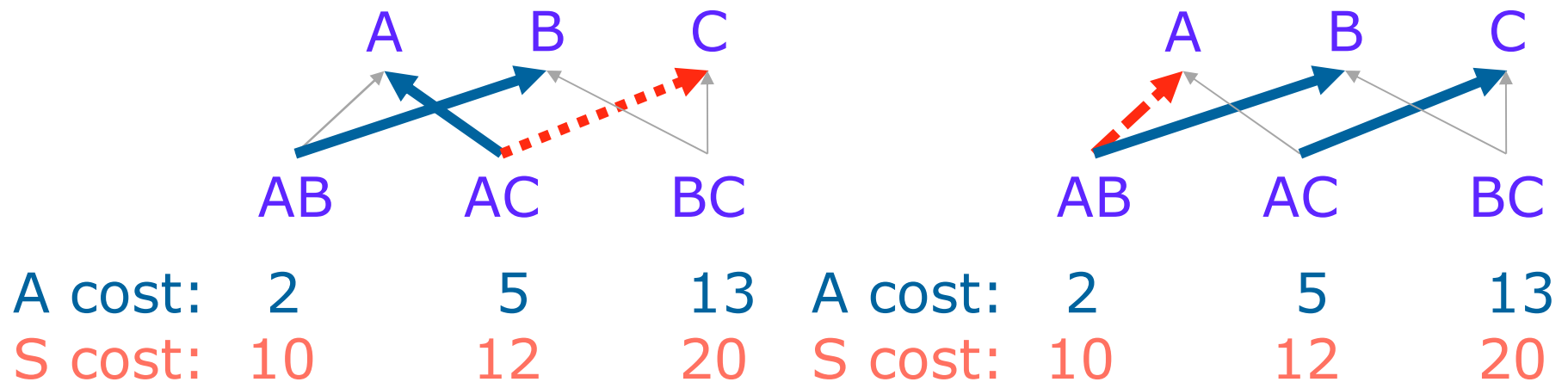


A cost:	2	5	13
S cost:	10	12	20

- $\text{Cost} = 2 + 10 + 5$
- AB will be ordered BA so that B can be generated without sorting
- Generating A requires resorting BA
- Also works to have A edge from $AB \rightarrow A$ and S edge from $AB \rightarrow B$
- C is computed from CA
- BC is not used later, so it can be computed in any order

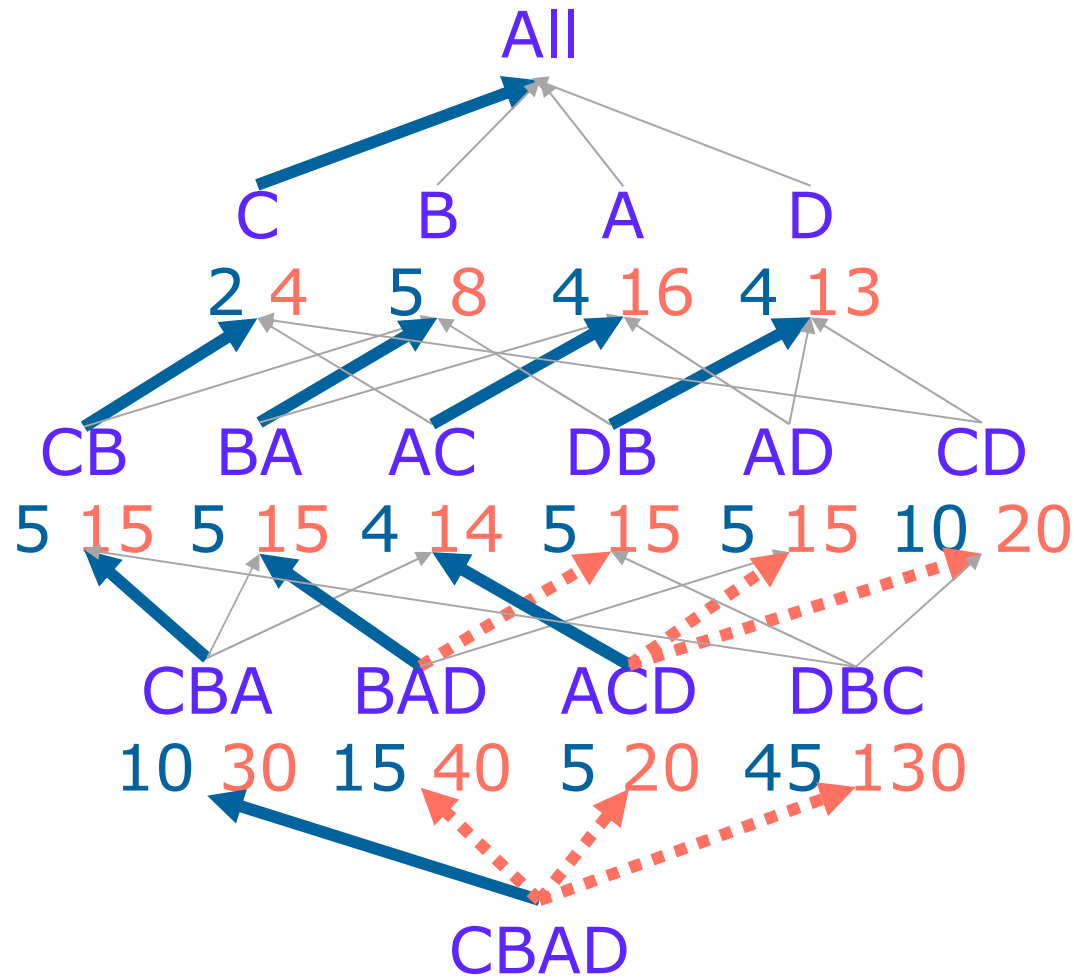
← A Edge (sorted) ← S Edge (not sorted)

Now we've seen two valid potential solutions



- The solution on the left costs 19. The solution on the right costs 17. Therefore, the solution on the right is the better choice.
- On an exam, if we ask you to find the best solution, we'll use a small enough example that you can do exhaustive search

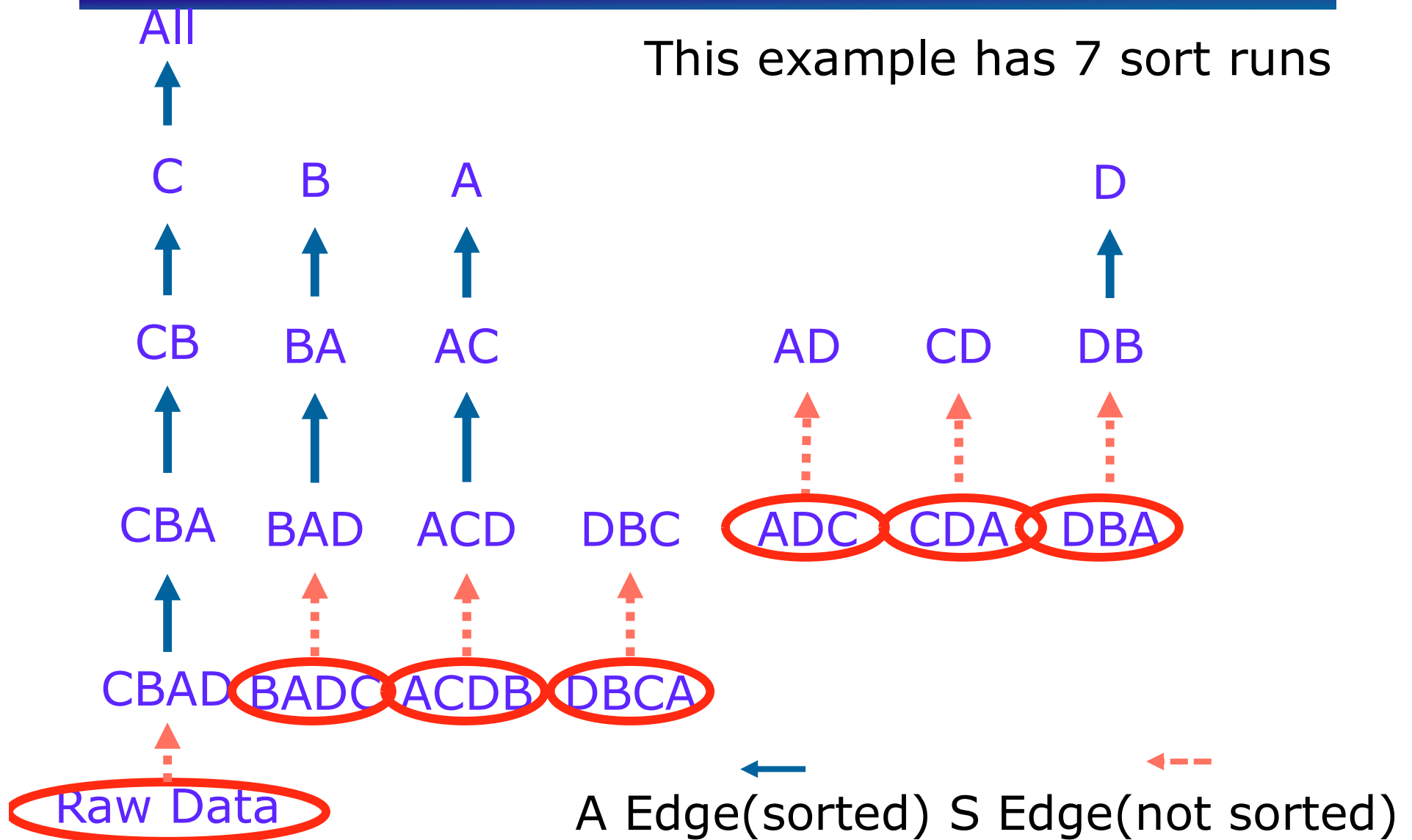
Can greedily compute entire cube materialization



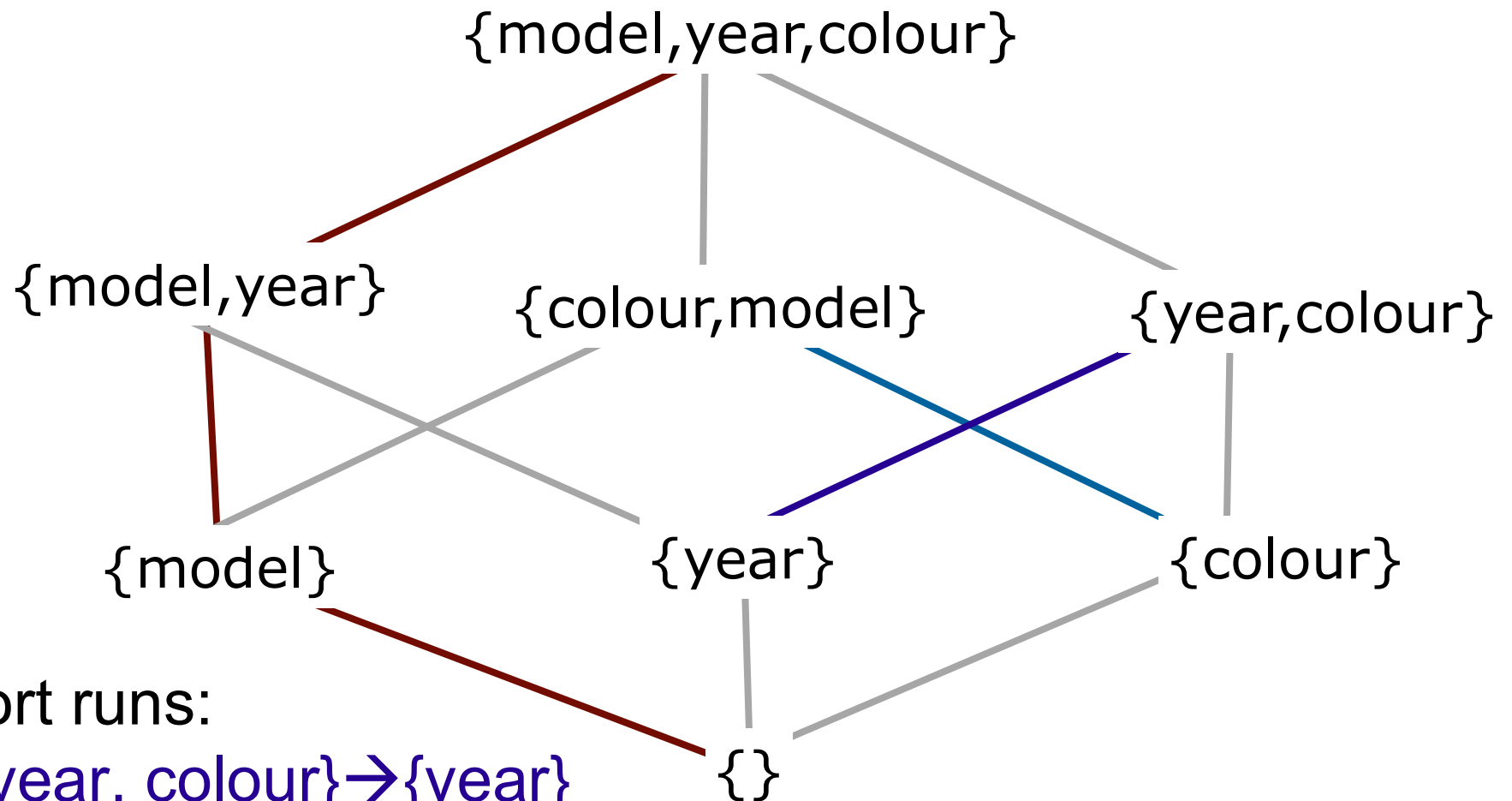
← A Edge(sorted) ←-- S Edge(not sorted)

Then execute the pipelines

Circles indicate sorts. Each column is a *sort run*



If orderings are fixed, you can compute sort runs without considering costs



3 sort runs:

$\{\text{year}, \text{colour}\} \rightarrow \{\text{year}\}$

$\{\text{colour}, \text{model}\} \rightarrow \{\text{colour}\}$


$\{\text{model}, \text{year}, \text{colour}\} \rightarrow \{\text{model}, \text{year}\} \rightarrow \{\text{model}\} \rightarrow \{\}$ ¹⁵⁶

Top N Queries

- For complex queries, users like to get an approximate answer quickly and keep refining their query.
- **Top N Queries:** If you want to find the 10 (or so) cheapest items, it would be nice if the DBMS could avoid computing the costs of *all* items before sorting to determine the 10 cheapest.
 - **Idea:** Guess a cost **c** such that the 10 cheapest items all cost less than c, and that not too many more cost less; then, add the selection “**cost < c**” and evaluate the query.
 - If the guess is right, great; we avoid computation for items that cost more than c.
 - If the guess is wrong, then we need to reset the selection and re-compute the query.

Top N Queries

Some DBMSs (e.g., DB2) offer special features for this.



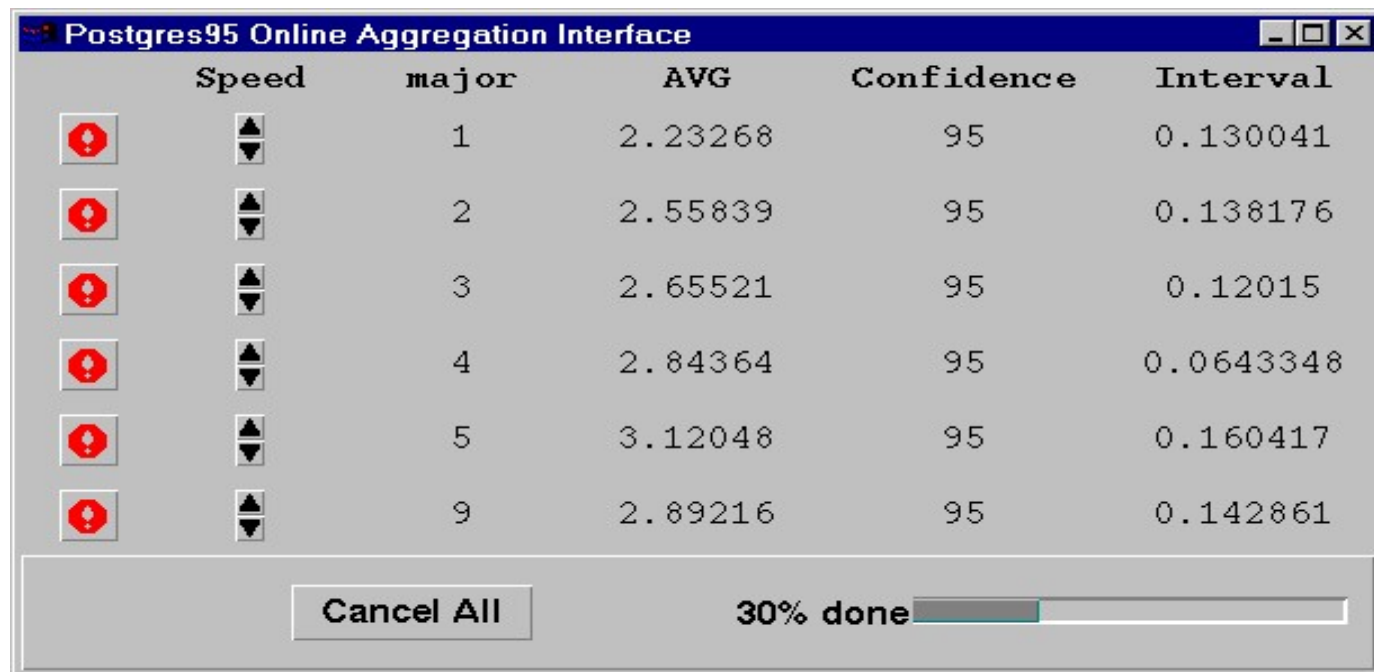
```
SELECT  P.pid, P.pname, S.sales
FROM    AllSales S, Products P
WHERE   S.pid=P.pid AND S.locid=1 AND S.timeid=3
ORDER BY S.sales DESC
OPTIMIZE FOR 10 ROWS;
```

```
SELECT  P.pid, P.pname, S.sales
FROM    AllSales S, Products P
WHERE   S.pid=P.pid AND S.locid=1 AND S.timeid=3
        AND S.sales > c
ORDER BY S.sales DESC;
```

- “OPTIMIZE FOR” construct is not in SQL:1999, but supported in DB2 or Oracle 9i
- Cut-off value c is chosen by the optimizer

Online Aggregation

- **Online Aggregation:** Consider an aggregate query. We can provide the user with some information before the exact average is computed.
 - e.g., Can show the current “running average” as the computation proceeds:



That's how to *build* an OLAP system

- But how do you use it?

Multidimensional Expressions (MDX)

- Multidimensional Expressions (MDX) is a query language for OLAP databases, much like SQL is a query language for relational databases.
- Like SQL, MDX has SELECT, FROM, and WHERE clauses (and others).
- Sample MDX syntax:

```
SELECT <content> ON COLUMNS,  
        <content> ON ROWS,  
        <content> ON PAGES  
FROM    <name_of_cube>  
WHERE
```

Multidimensional Expressions (MDX)

- SQL returns query results in the form of a 2-dimensional table; therefore, the SELECT clause defines the column layout. MDX, however, returns query results in the form of a k -dimensional sub-cube. The SELECT clause defines the k axes.
- The 3 default axes are named as follows:
 - Axis 0 = “columns” (or just write “axis(0)”)
 - Axis 1 = “rows”
 - Axis 2 = “pages”
- The ON keyword specifies the axes.
 - e.g., SELECT (...) ON COLUMNS

Multidimensional Expressions (MDX)

- The queries can be simple or complex.
- MDX allows you to restrict analysis/calculations to a particular sub-cube of the overall data cube. This is useful for slice and dice operations.
 - e.g., You may wish to focus only the set of T-shirts that were sold to 18 year olds.
- MDX queries can be optimized, similar in spirit to the way SQL queries are optimized; but, the process is more complex.

Learning Goals Revisited



- Compare and contrast OLAP and OLTP processing (e.g., focus, clients, amount of data, abstraction levels, concurrency, and accuracy).
- Explain the ETL tasks (i.e., extract, transform, load) for data warehouses.
- Explain the differences between a star schema design and a snowflake design for a data warehouse, including potential tradeoffs in performance.
- Argue for the value of a data cube in terms of:
 - The type of data in the cube (numeric, categorical, temporal, counts, sums)
 - The goals of OLAP (e.g., summarization, abstractions), and
 - The operations that can be performed (drill-down, roll-up, slicing/dicing).
- Estimate the complexity of a data cube, in terms of the number of equivalent aggregation queries.