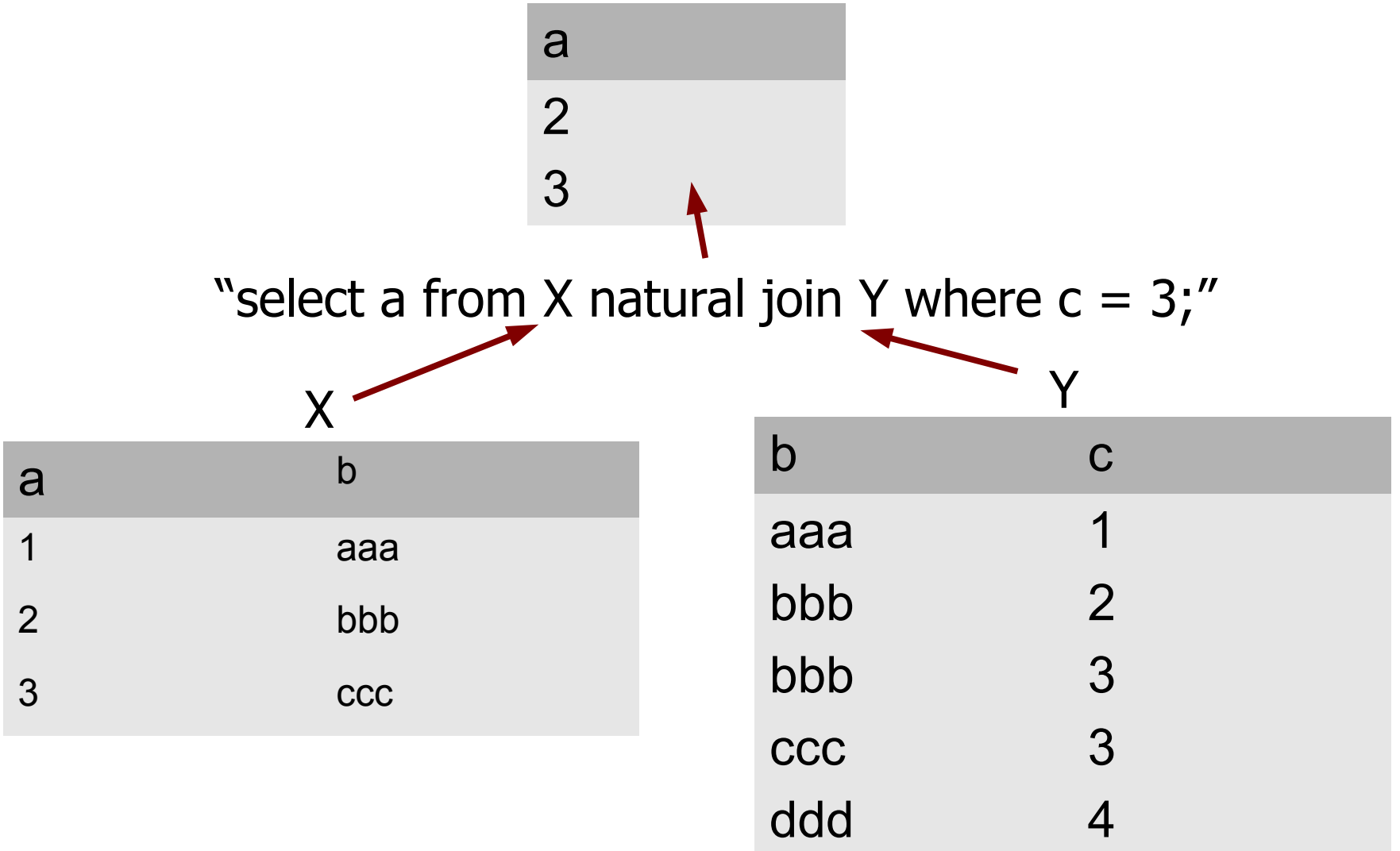
Database Administration

José Orlando Pereira

Departamento de Informática
Universidade do Minho

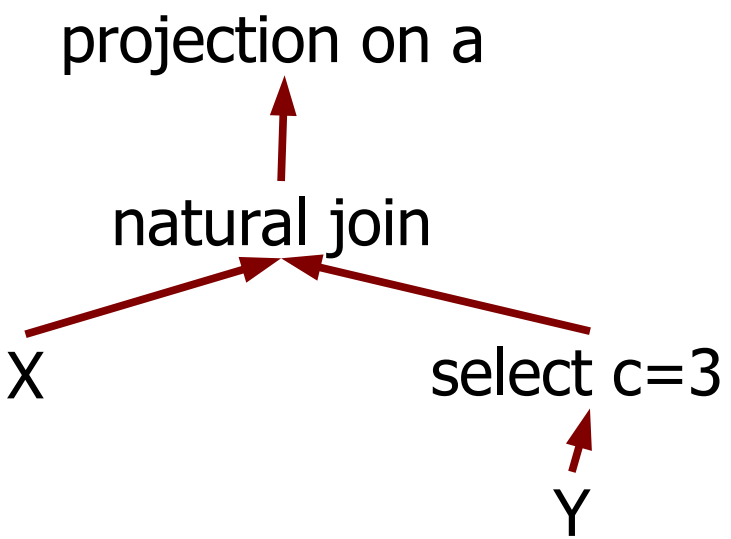
Query processing



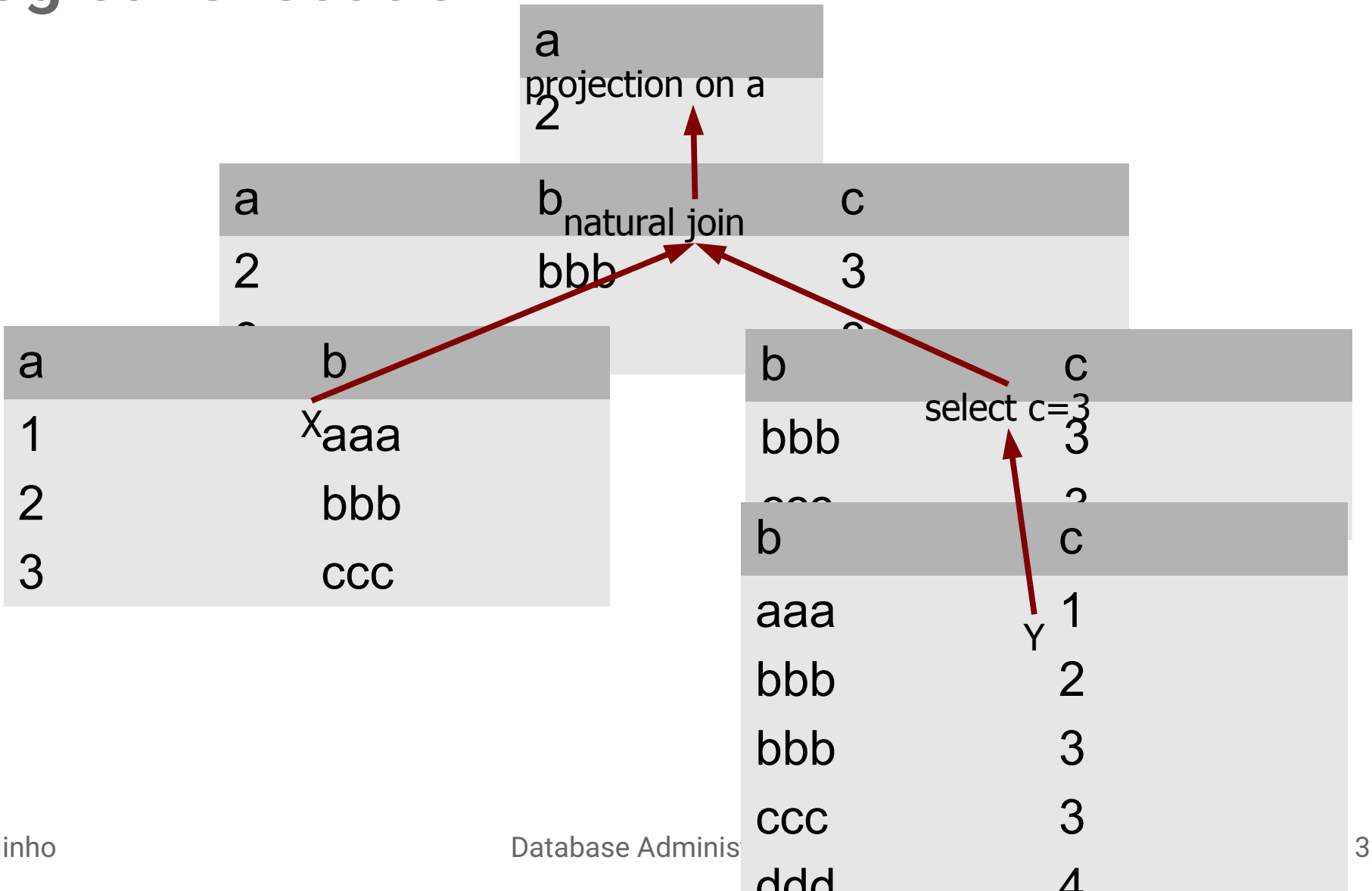
Compilation

SQL { "select a from X natural join Y where c = 3;"

Relational algebra {



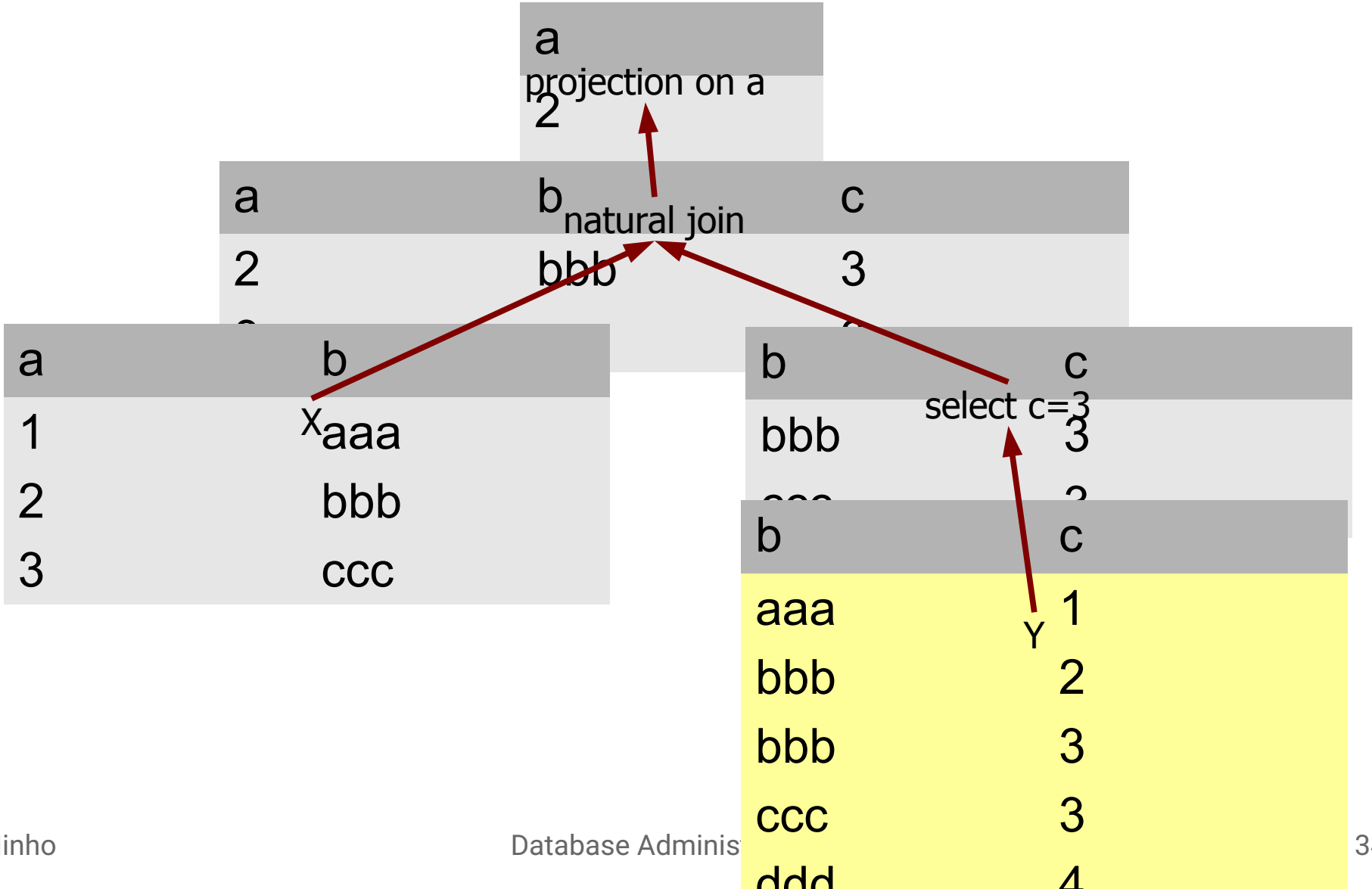
Logical execution



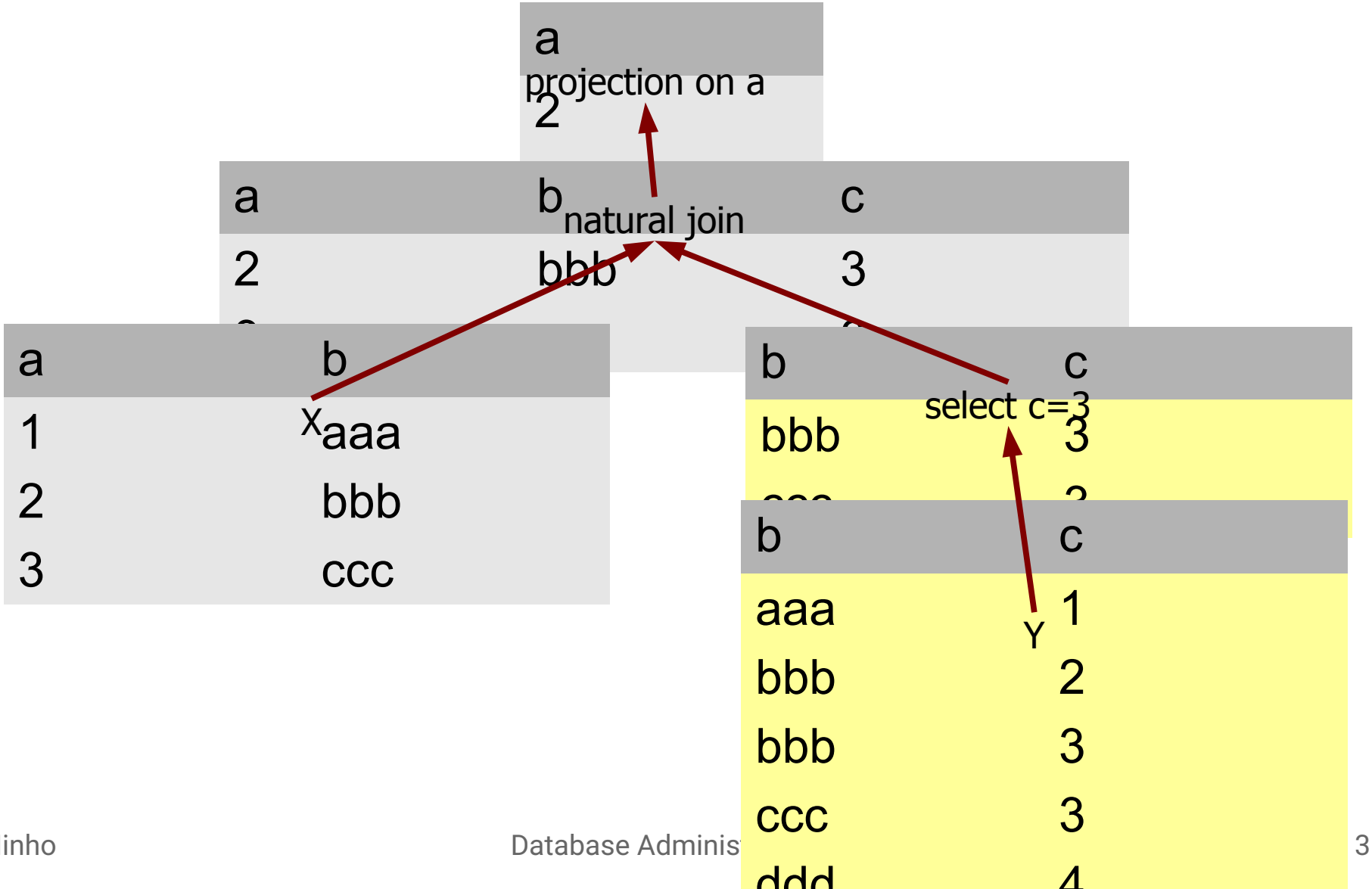
Materialization

- Each operator is a function:
  - Returns a relation
  - Parameters are other relations (possibly, returned from operators)
- Computation order:
  - From leaves to root

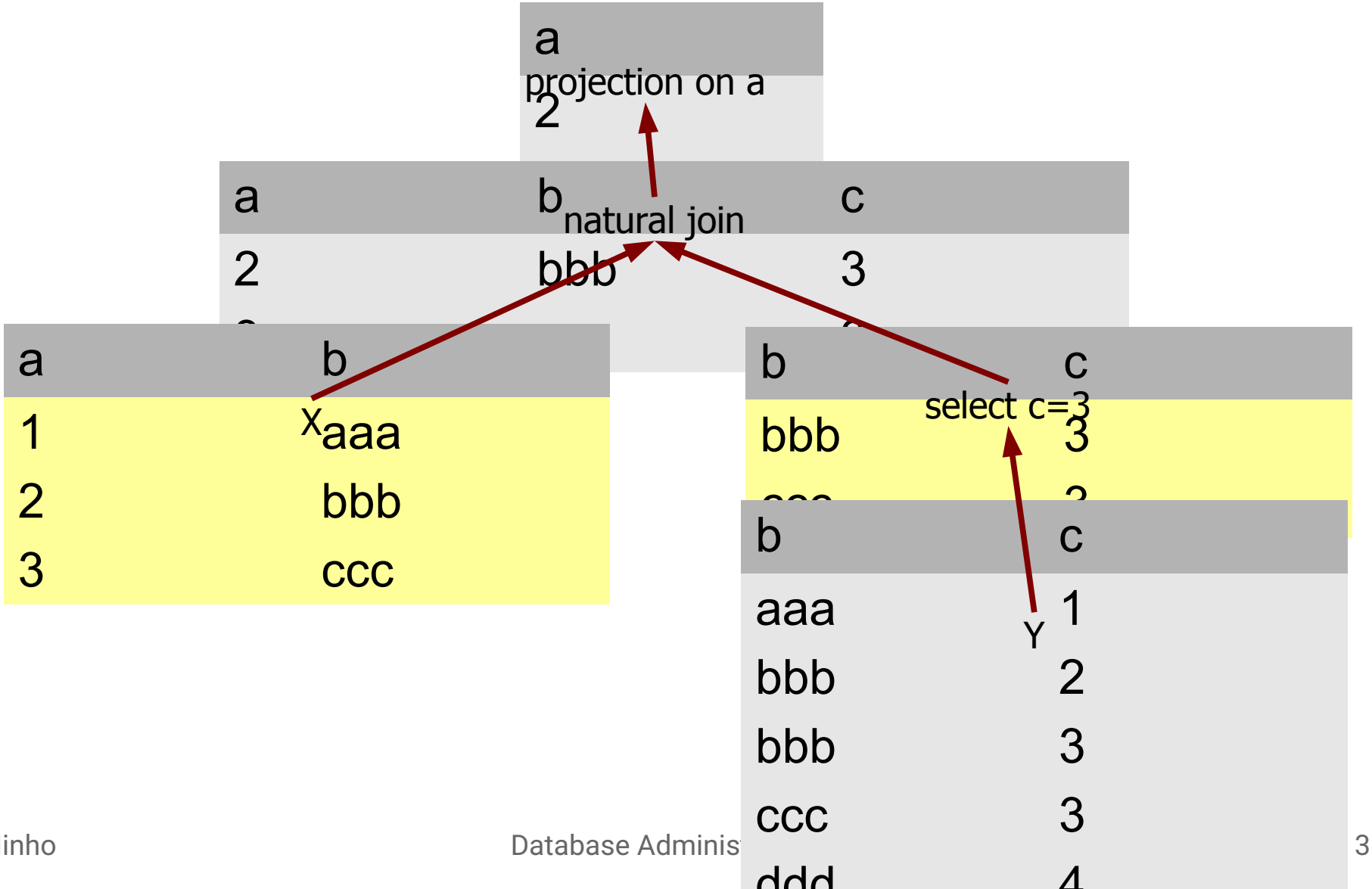
Execution with materialization



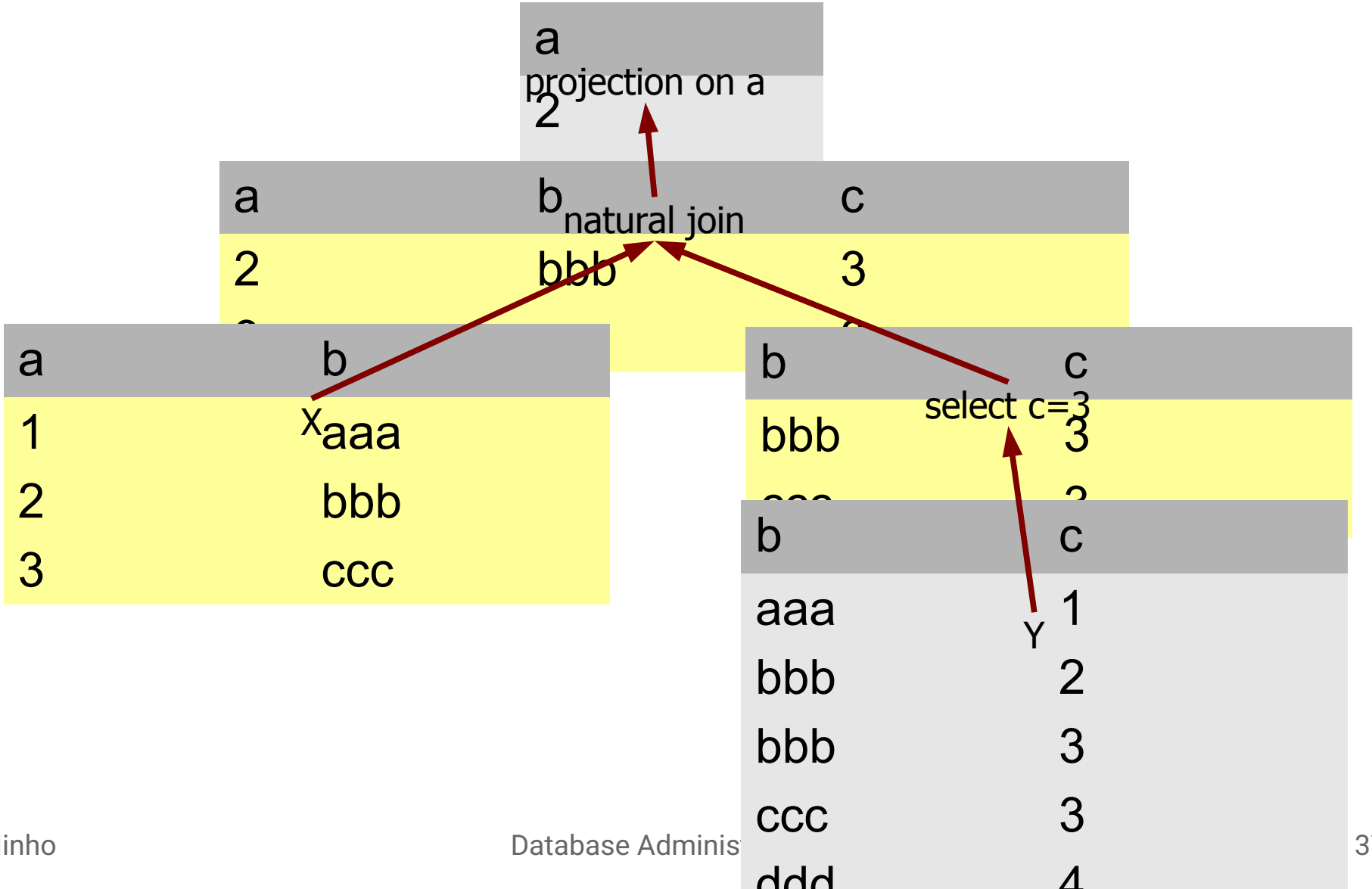
Execution with materialization



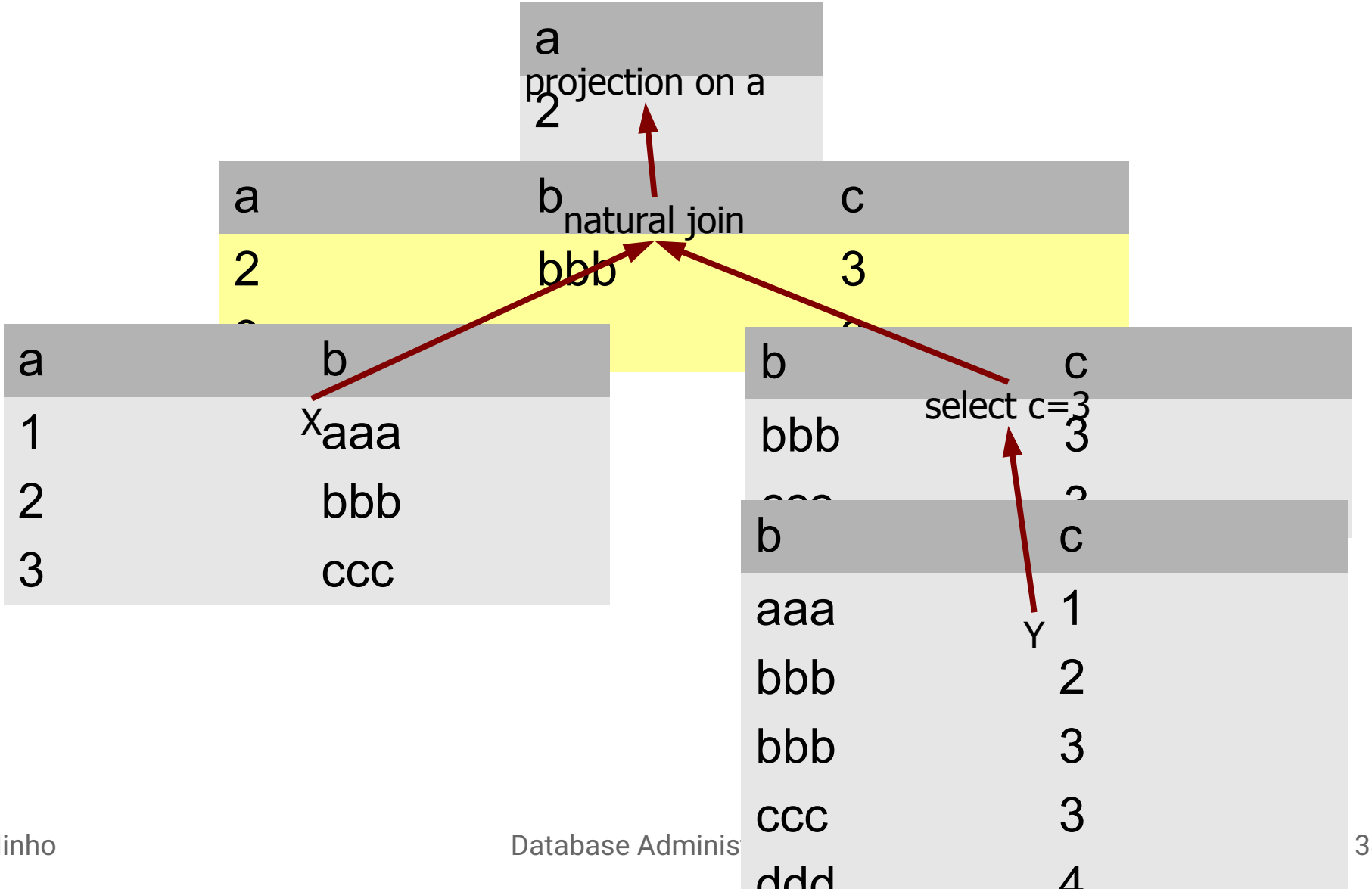
Execution with materialization



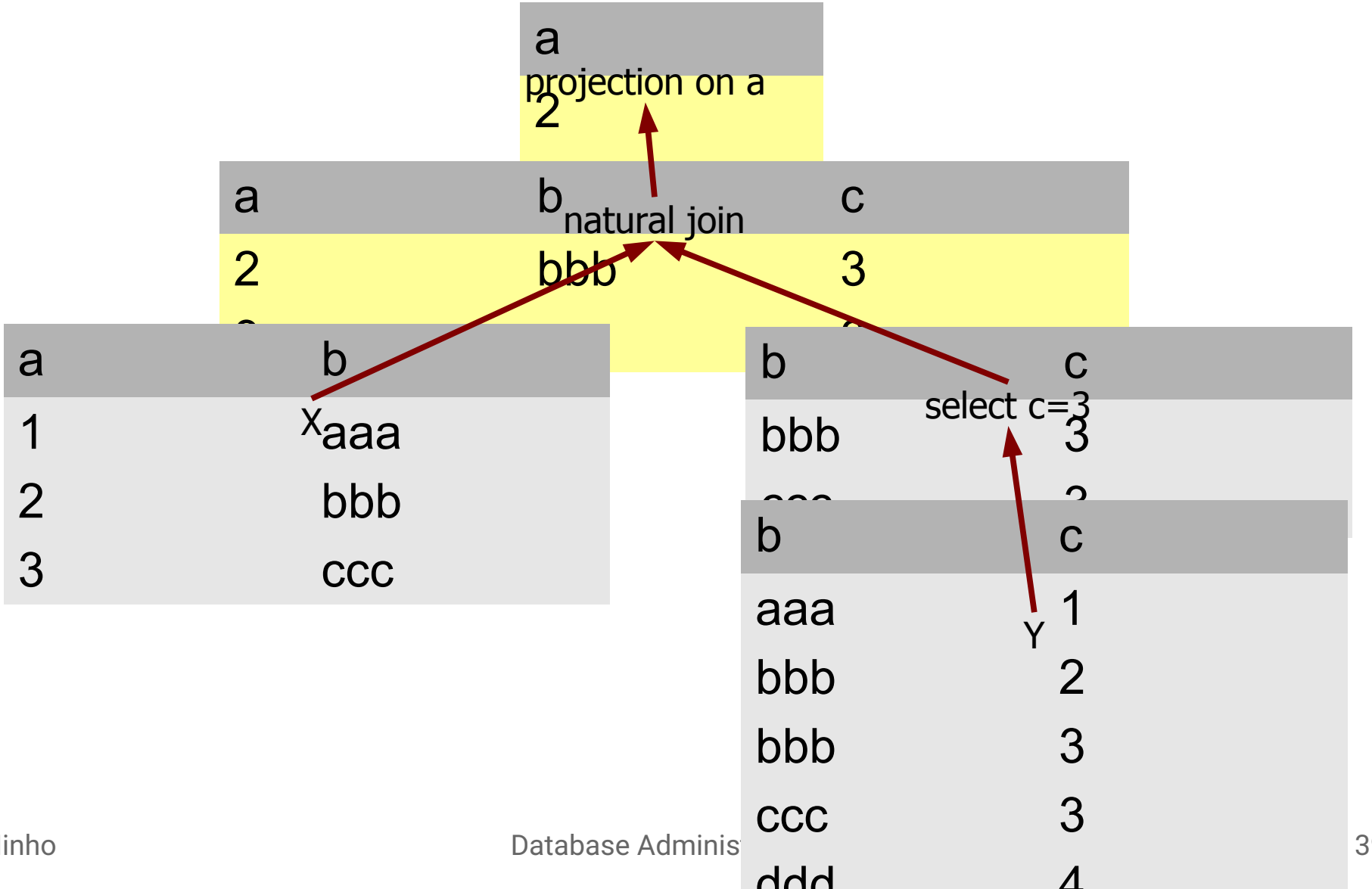
Execution with materialization



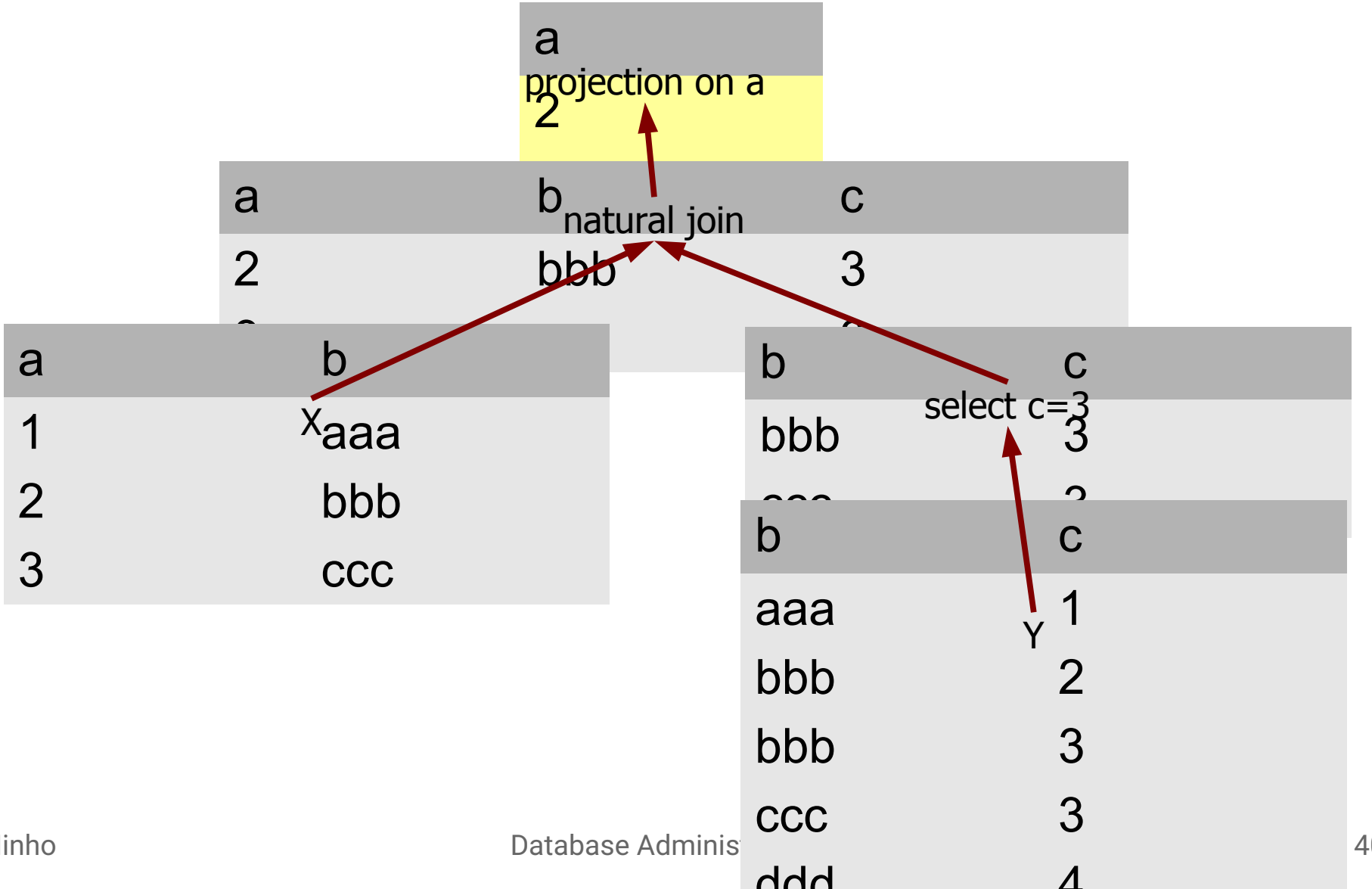
Execution with materialization



Execution with materialization



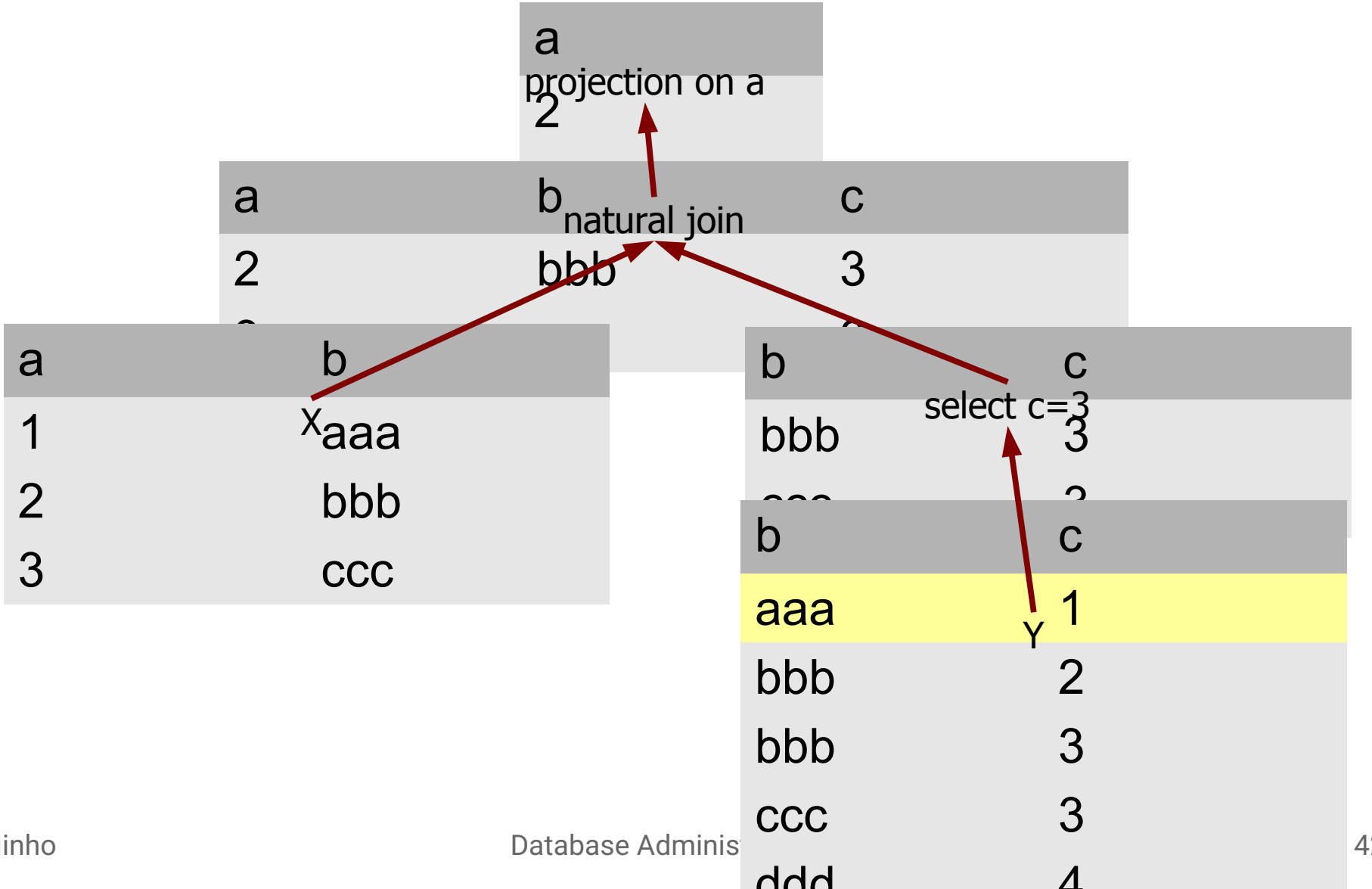
Execution with materialization



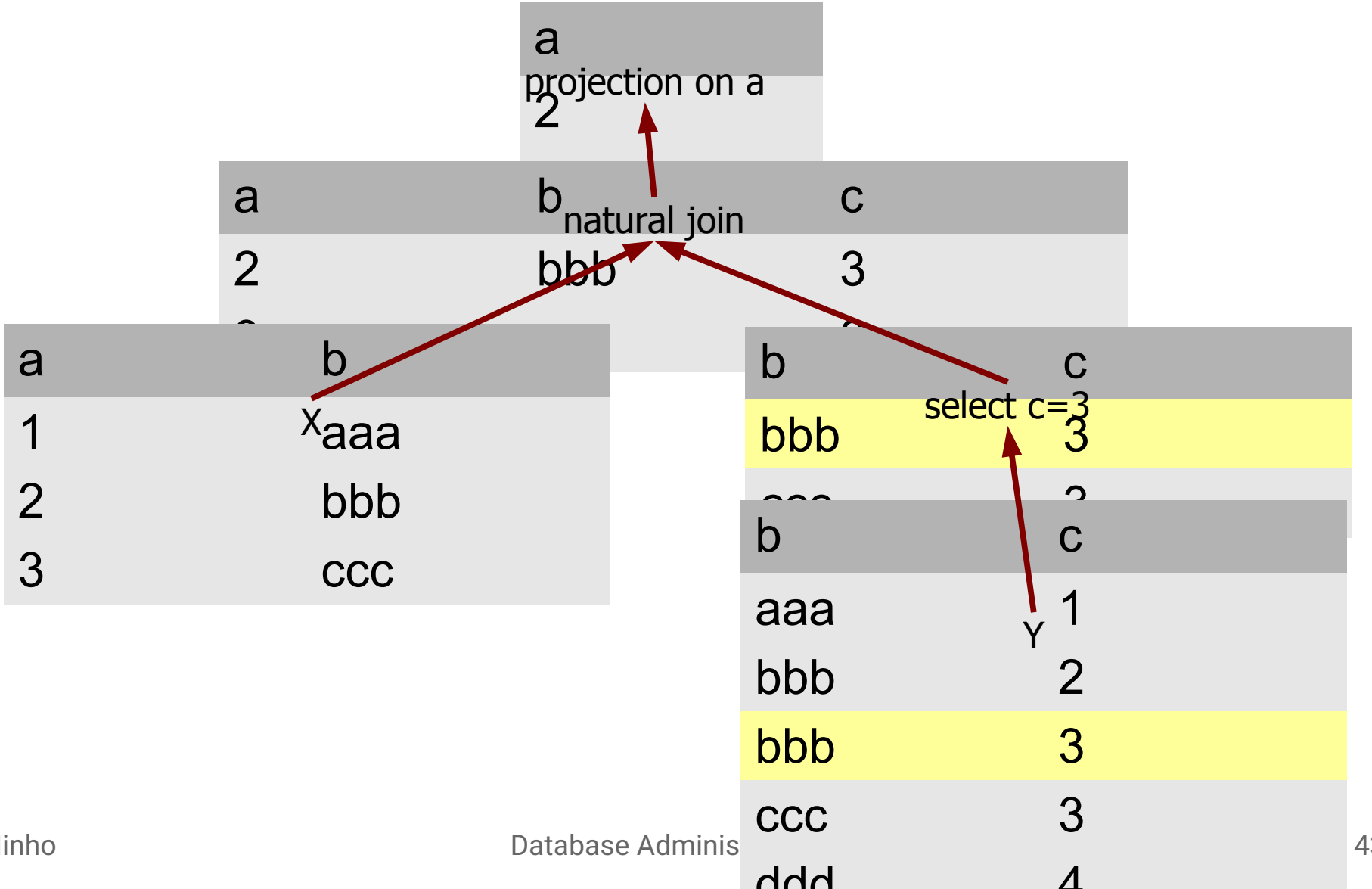
Iteration

- Each operator is an object:
  - Interface similar to java.util.Iterator:
    - open() - get ready to return first record
    - next() - return next record
    - close() - no more records required
  - Constructor parameters:
    - Other operator objects
- Computation order:
  - From leaves to root, for each record

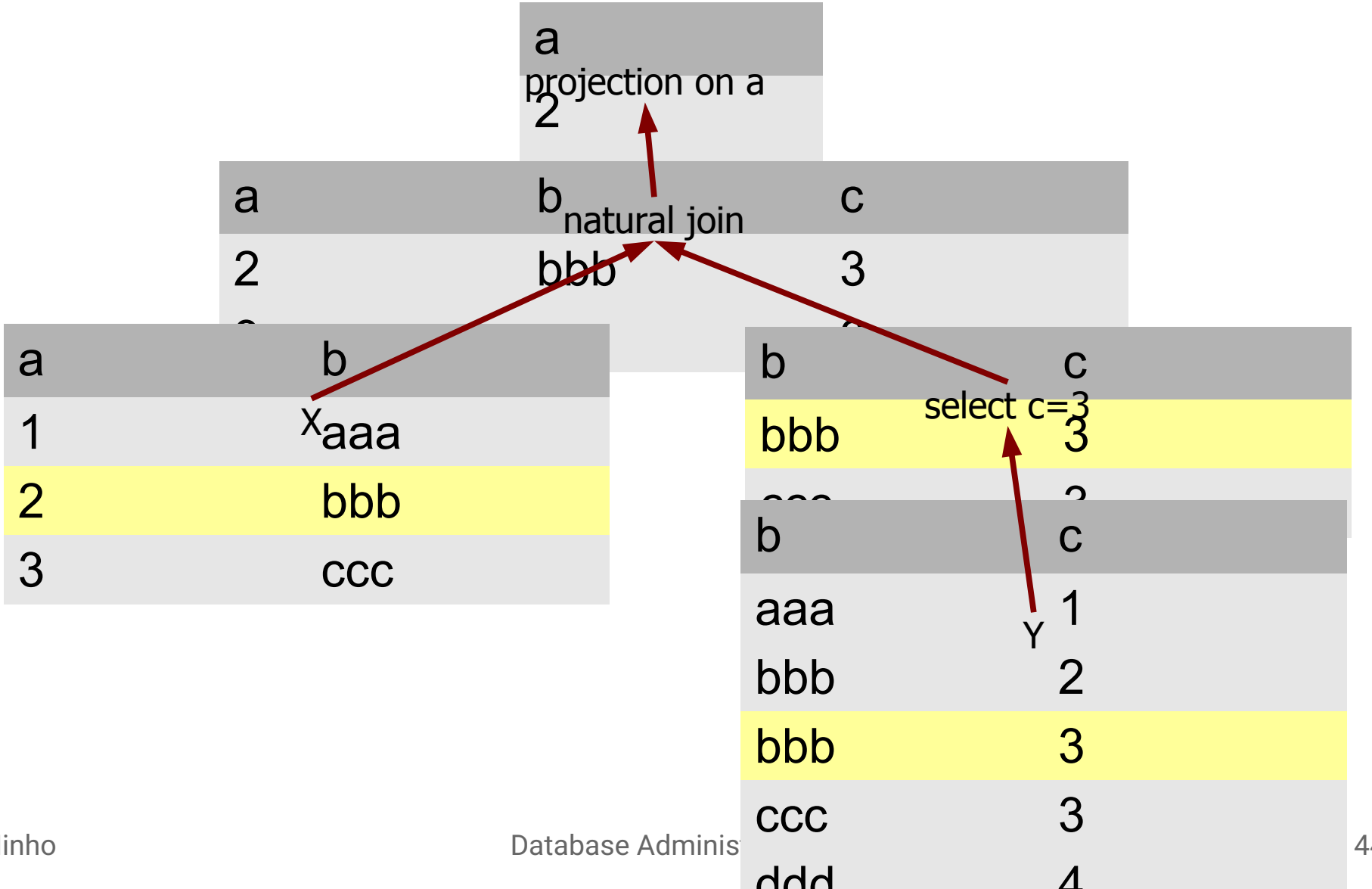
Execution with iteration



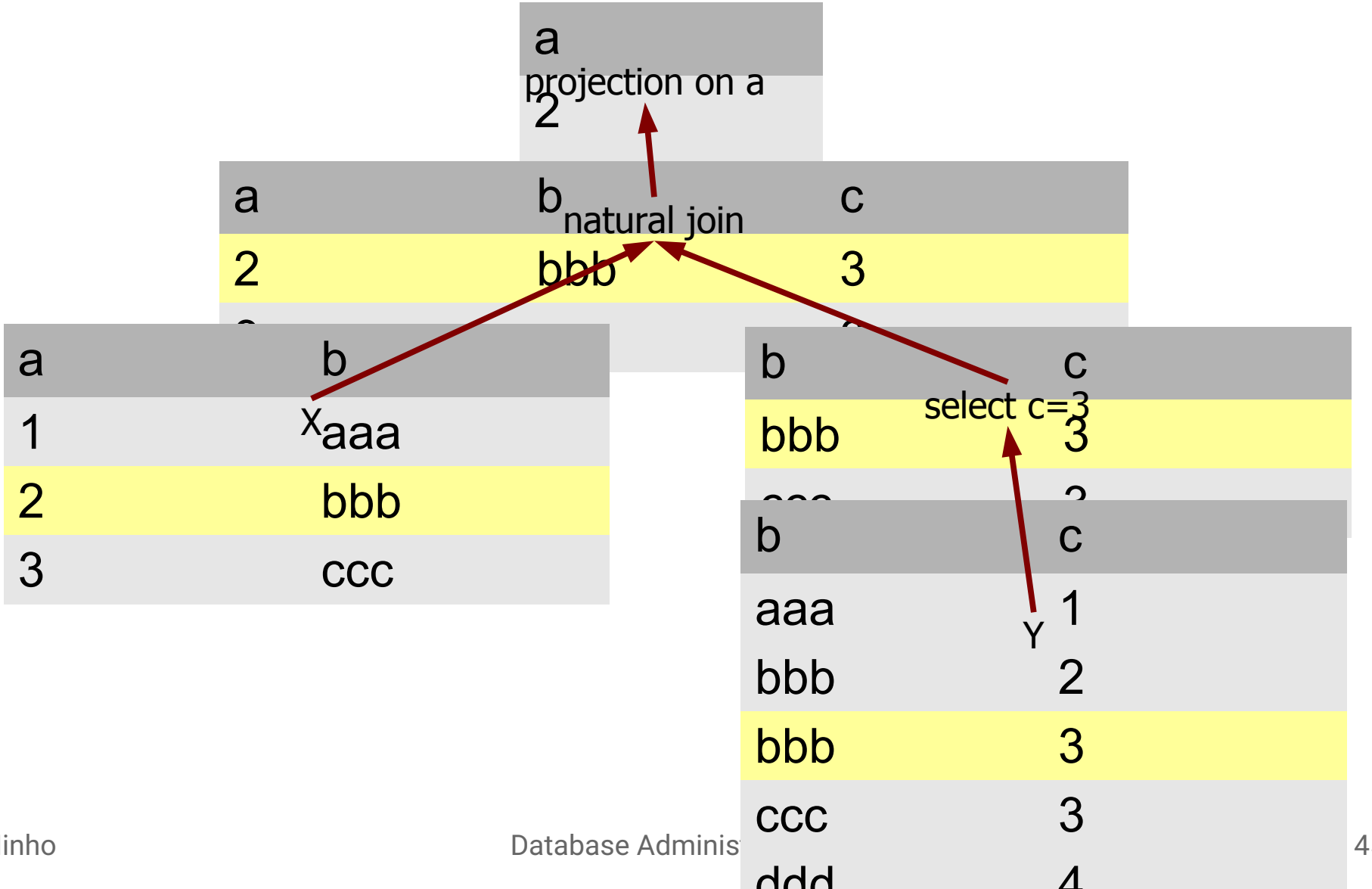
Execution with iteration



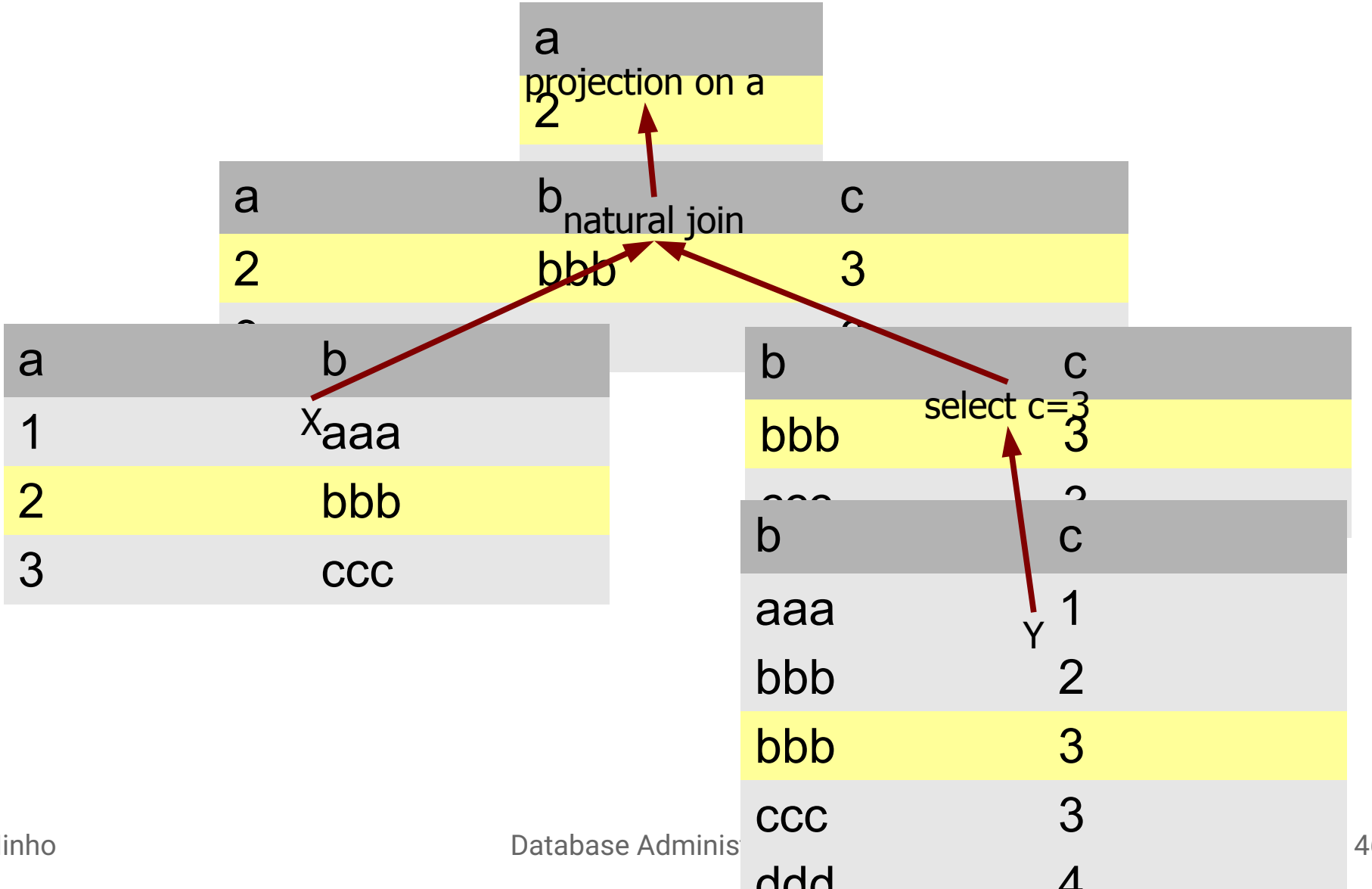
Execution with iteration



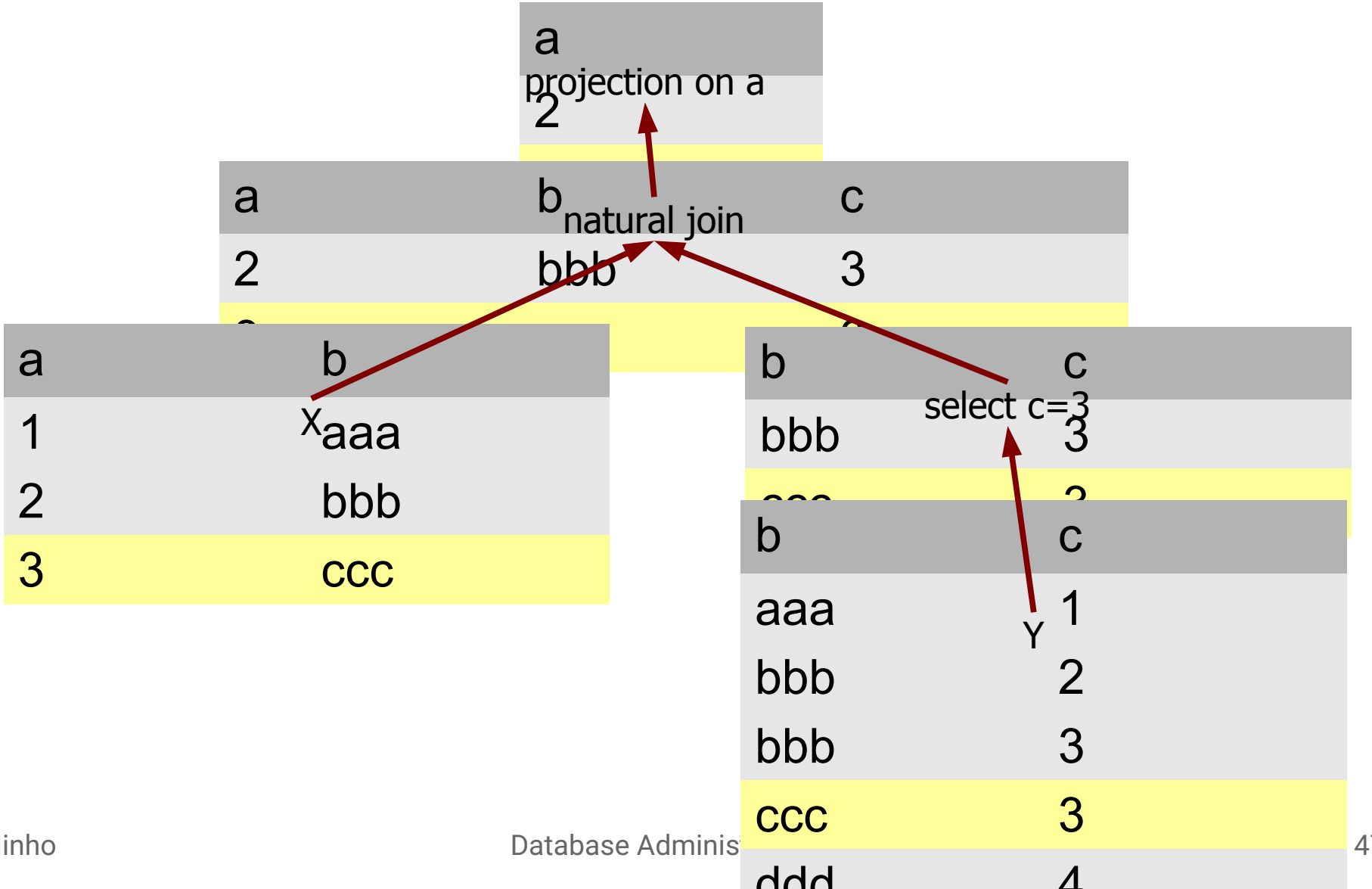
Execution with iteration



Execution with iteration



Execution with iteration



Materialization vs Iteration

- Iteration avoids caching entire relations
- Materialization avoid reading records more than once
- Can mix both:
  - A materialization operator obtains all records upon first invocation of open
  - Returns records from cached copy on iteration

Roadmap

- What physical operators exist for each logical operation?

- Later: How are physical operators selected?

One-pass, record-at-a-time

- Operators:
  - Sequential scan
  - Selection
  - Projection
- Memory requirements:
  - No more than one record required
  - Always possible

User defined functions (UDFs)

- Functions can be defined in various languages
  - Python
- Scalar functions used in projections/selections:

  SELECT a, f1(a) FROM t;

  SELECT * FROM t WHERE f2(a)
- Table functions can be used in sequential scans:

  SELECT * FROM f3(…);
- User defined functions can access external services:
  - Web services
  - GenAI

One-pass, full relation, unary

- Duplicate elimination:
  - Cache unique records
  - “select distinct * from X;”
- Grouping and aggregation:
  - Cache groups
  - “select count(*) from X group by b;”
- Sorting:
  - Cache all records and sort in memory
  - “select * from X order by b;”

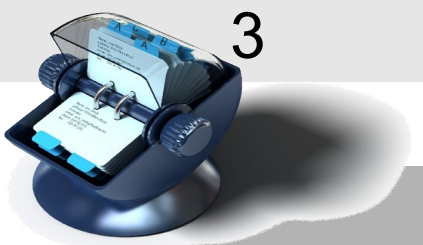
One-pass, full relation, binary

- Union, difference, intersection, product, join:
  - Read and cache the smallest relation
  - Organize for fast look-up (e.g. hash)
  - Read and operate on each record from the largest relation

One-pass, full relation, binary

- Load smaller table into memory and add search structure:

a	b	c
2	bbb	3
3	ccc	3



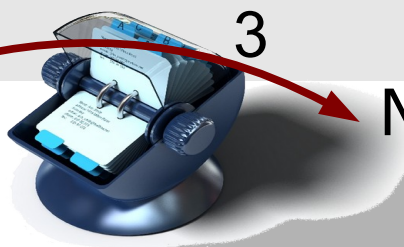
a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

One-pass, full relation, binary

- Test each record from the largest relation:

a	b	c
2	bbb	3
3	ccc	3



Not found

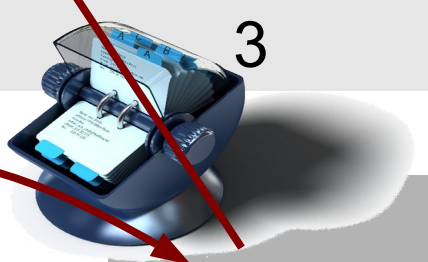
a	b
1	aaa
2	bbb
3	ccc

c
bbb 3
ccc 3

One-pass, full relation, binary

- Test each record from the largest relation:

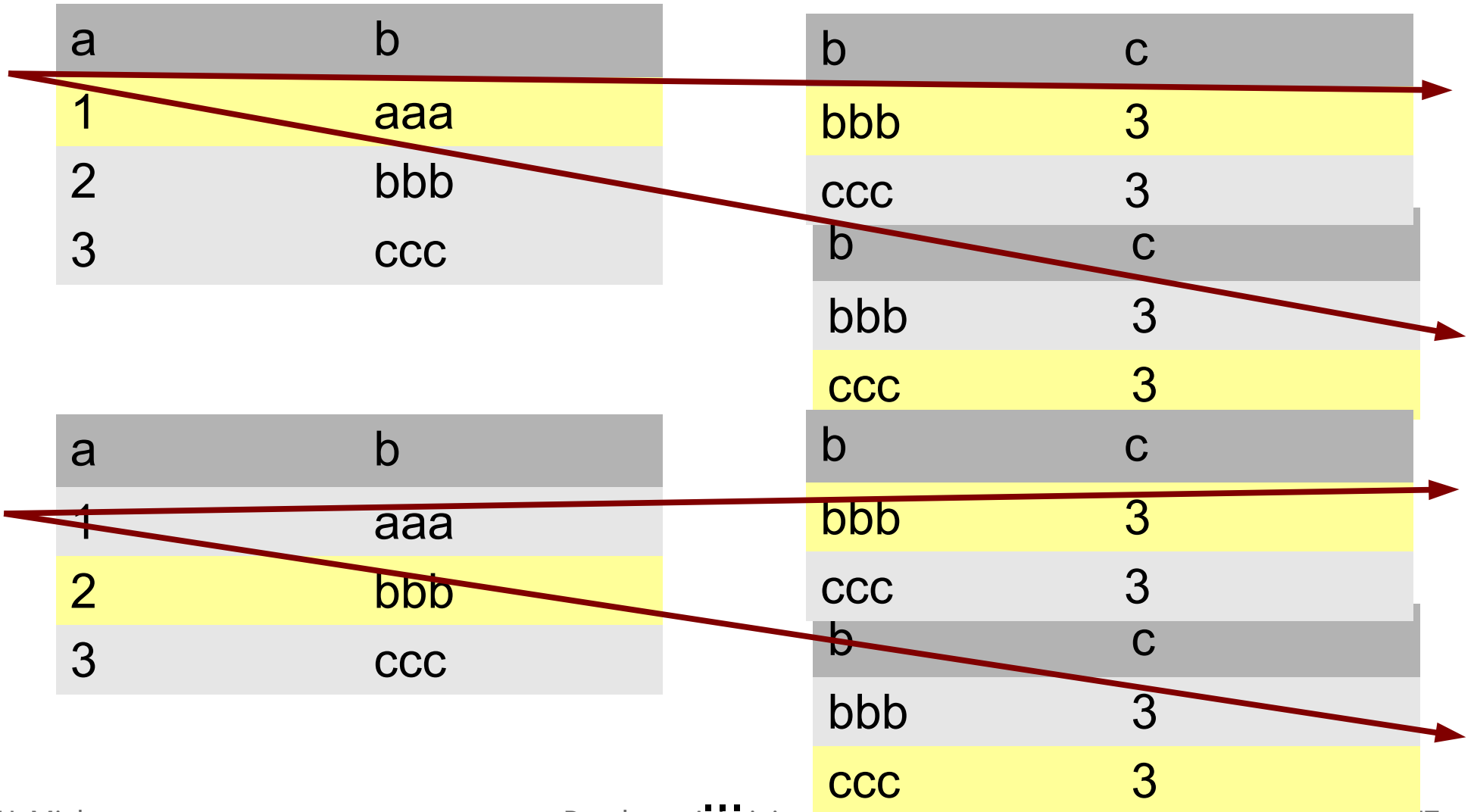
a	b	c
2	bbb	3
3	ccc	3



a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

Nested-loop join (NLJ)



Nested-loop join (NLJ)

- Memory requirements:
  - One record from each relation
- Operations:
  - If outer loop has N records
  - Reads inner relation N times

Block-based NLJ

- Much smarter: Execute NLJ by blocks
- Memory requirements:
  - One block from each relation
- Operations:
  - If outer loop has N records / B blocks (B << N)
  - Reads inner relation B times (B << N!)

Benchmarking

- Repeat workload for a variable number of client threads
- Discard initial and final periods
- Measure:
  - Response time (duration of transactions)
  - Throughput (rate of execution)