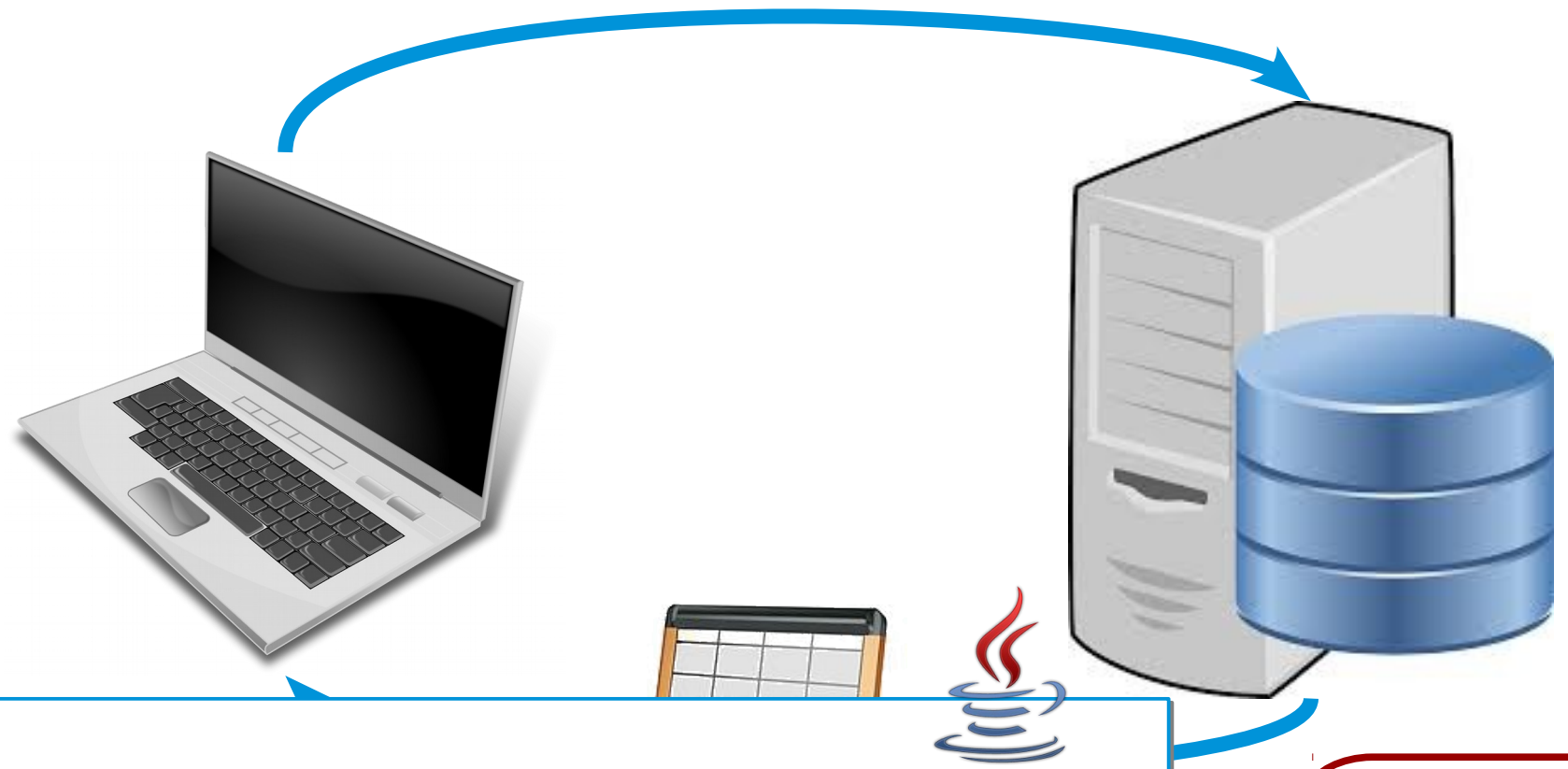
Databases and JDBC

J. Pereira
jop@di.uminho.pt

R. Vilaça
rmv@di.uminho.pt

Motivation

“select * from t where a = 0”



```
Connection c = new PostgreSQL(...);
List<Row> r = c.execute(“select * from ...”);
```

Why not?

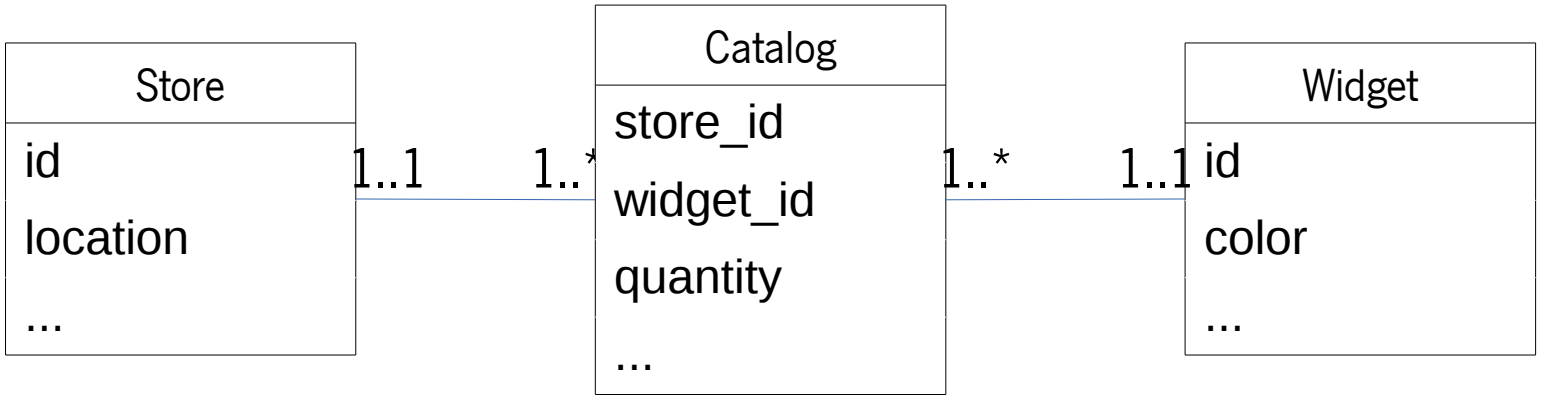
Overview

- Example
- Query optimization and compilation
- Query execution
- Transactions and isolation
- More...
  - Distributed databases research @ HASLab
  - Distributed databases @ MEI

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Example

- The colorful widget finder:
  - Suppliers at multiple locations
  - Widgets come in different colors
- Operations:
  - "Buy a widget."
  - "Where to find red widgets in Porto?"



Example

› “Buy widget.”

id	*location*
1	Rome
2	Lisbon
3	Porto
4	Lisbon
5	Brussels
6	Lisbon
7	Lisbon
8	Porto
...	...

s_id	*w_id*	*quan*
1	4	...
2	3	...
2	1	...

id	*color*
1	green
2	red
3	green

```
SELECT quantity FROM Catalog
     WHERE s_id = ... AND w_id = ...
if (q > 0)
  UPDATE Catalog
     SET quantity = quantity - 1
     WHERE s_id = ... AND w_id = ...
```


HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Example

› "Buy widget."

id	*location*
1	Rome
2	Lisbon
3	Porto
4	Lisbon
5	Brussels
6	Lisbon
7	Lisbon
8	Porto
...	...

s_id	*w_id*	*quan*
1	4	...
2	3	...
2	1	...
2	2	...
3	4	...
4	4	...

id	*color*
1	green
2	red
3	green
4	red
5	blue
...	...

```
UPDATE Catalog
      SET quantity = quantity - 1
      WHERE s_id = ... AND w_id = ...
              AND quantity > 0
```


HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Example

› "Where to find red widgets in Porto?"

id	*location*
1	Rome
2	Lisbon
3	Porto
4	Lisbon
5	Brussels
6	Lisbon
7	Lisbon
8	Porto
...	...

s_id	*w_id*	*quan*
1	4	...
2	3	...
2	1	...
2	2	...
3	4	...

id	*color*
1	green
2	red
3	green
4	red

```
SELECT * FROM Store WHERE location='Porto'
for(...)
     SELECT * FROM Catalog WHERE s_id = ...
for(...)
     SELECT * FROM Widget WHERE w.color='red'
```


HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Example

› "Where to find red widgets in Porto?"

id	*location*
1	Rom
2	Lisbon
3	Porto
4	Lisbon

s_id	*w_id*	*quan*
1	4	...
2	3	...
2	1	...
2	2	...

id	*color*
1	green
2	red
3	green
4	red
5	blue
...	...

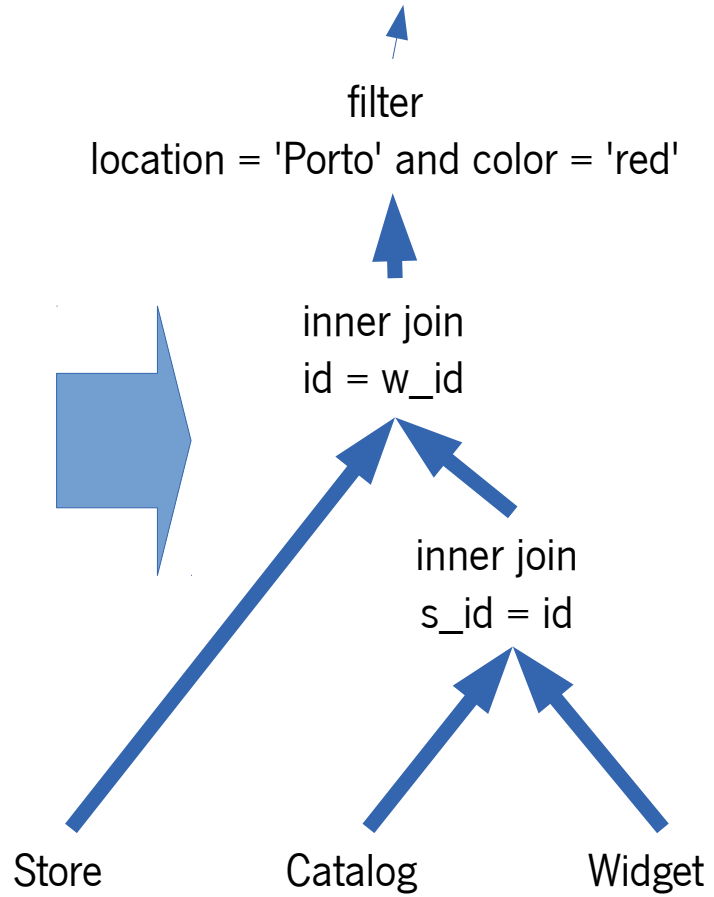
```
SELECT
      s.id, s.location
FROM
      Store s
      INNER JOIN Catalog c ON s.id=c.sid
      INNER JOIN Widget w ON c.wid=w.id
WHERE
      s.location='Porto' AND w.color='red'
```


HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Query processing

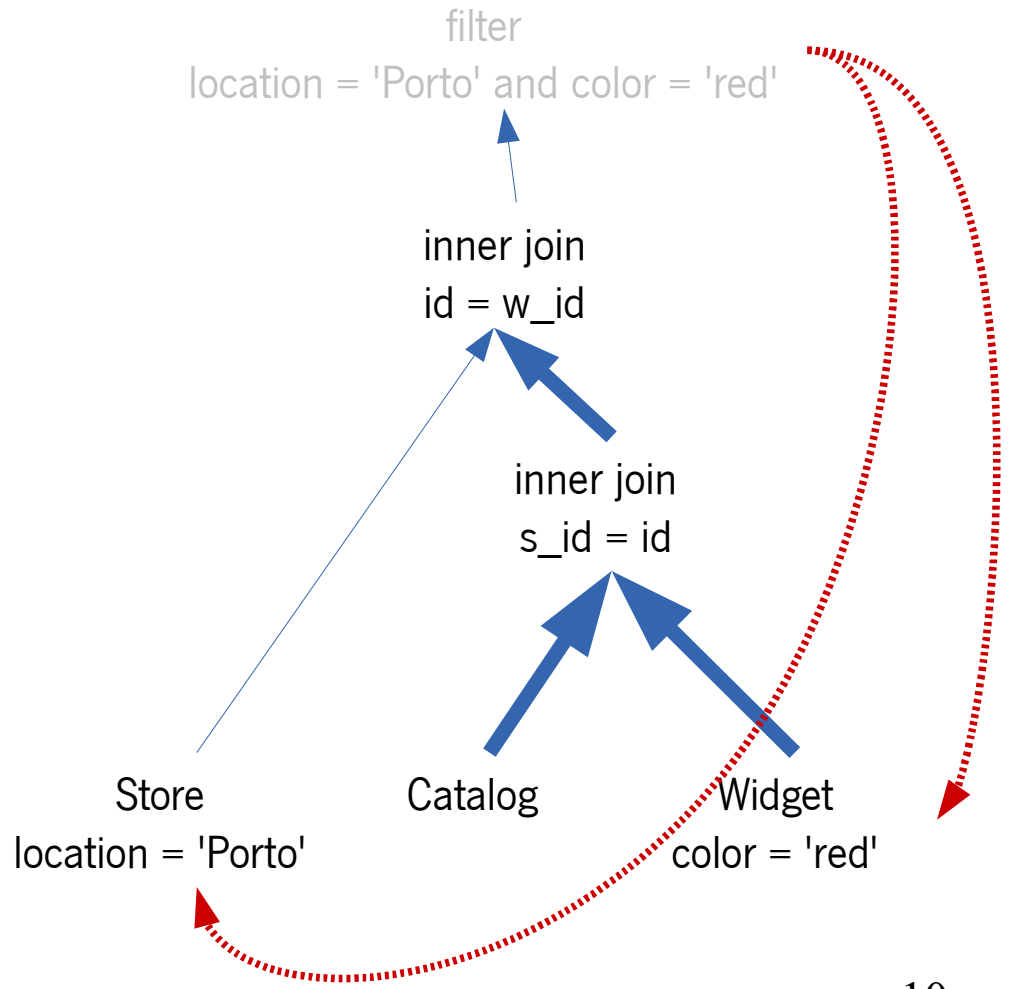
› Query compilation to a relational operator expression:

```
SELECT
      s.id, s.location
FROM
      Store s
      INNER JOIN Catalog c ON s.id=c.s_id
      INNER JOIN Widget w ON c.w_id=w.id
WHERE
      s.location='Porto' AND w.color='red'
```



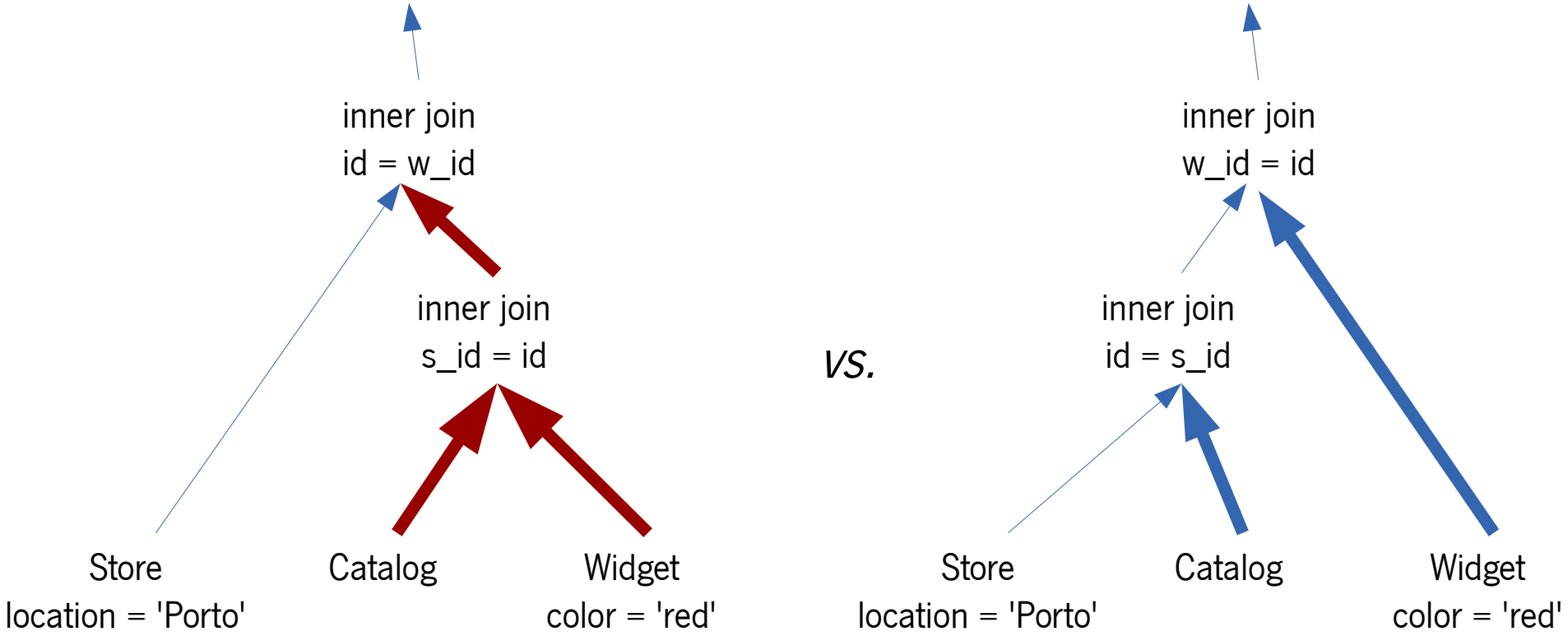
Query processing

› Heuristic optimization:



Query processing

- Cost-based optimization using:
  - Estimated row counts
  - Row cost



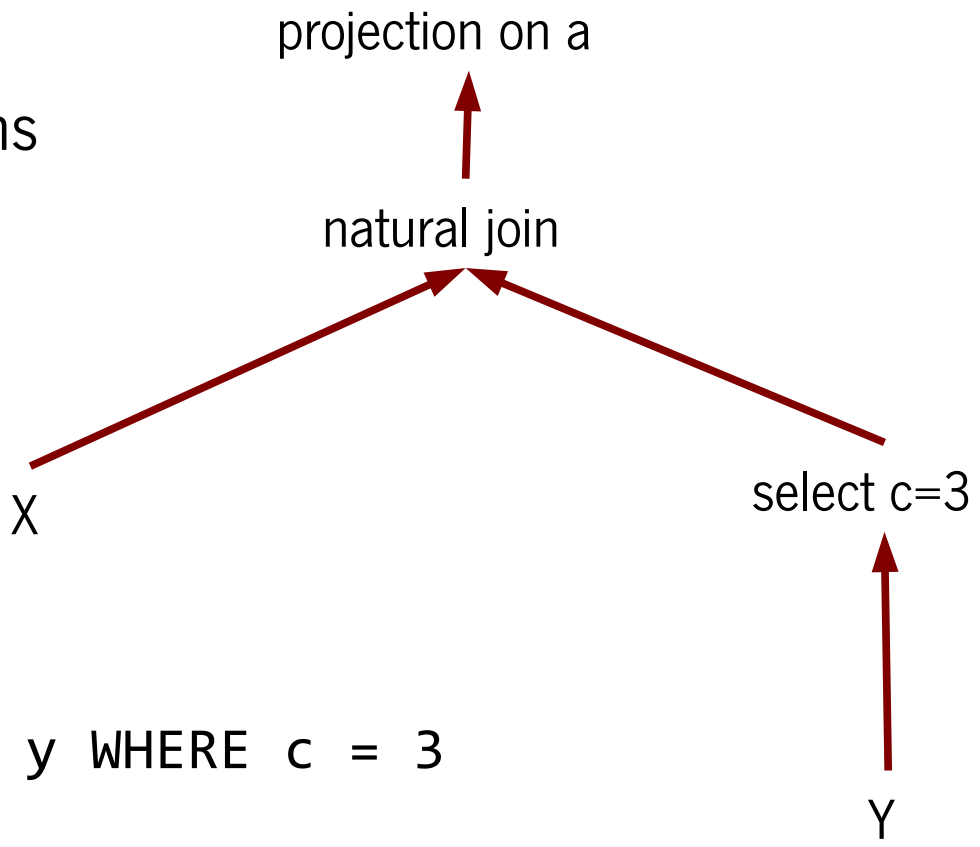
Conclusions

- SQL is declarative
  - Describes “What to do”, not “How to do it”
  - The DBMS decides “How to do it”
  - The DBMS often knows best!
- Consequences:
  - Let the DBMS do as much as possible (prefer slide 8 to 7)
  - Use “prepare” judiciously
  - Meta-data is available before execution

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Query execution

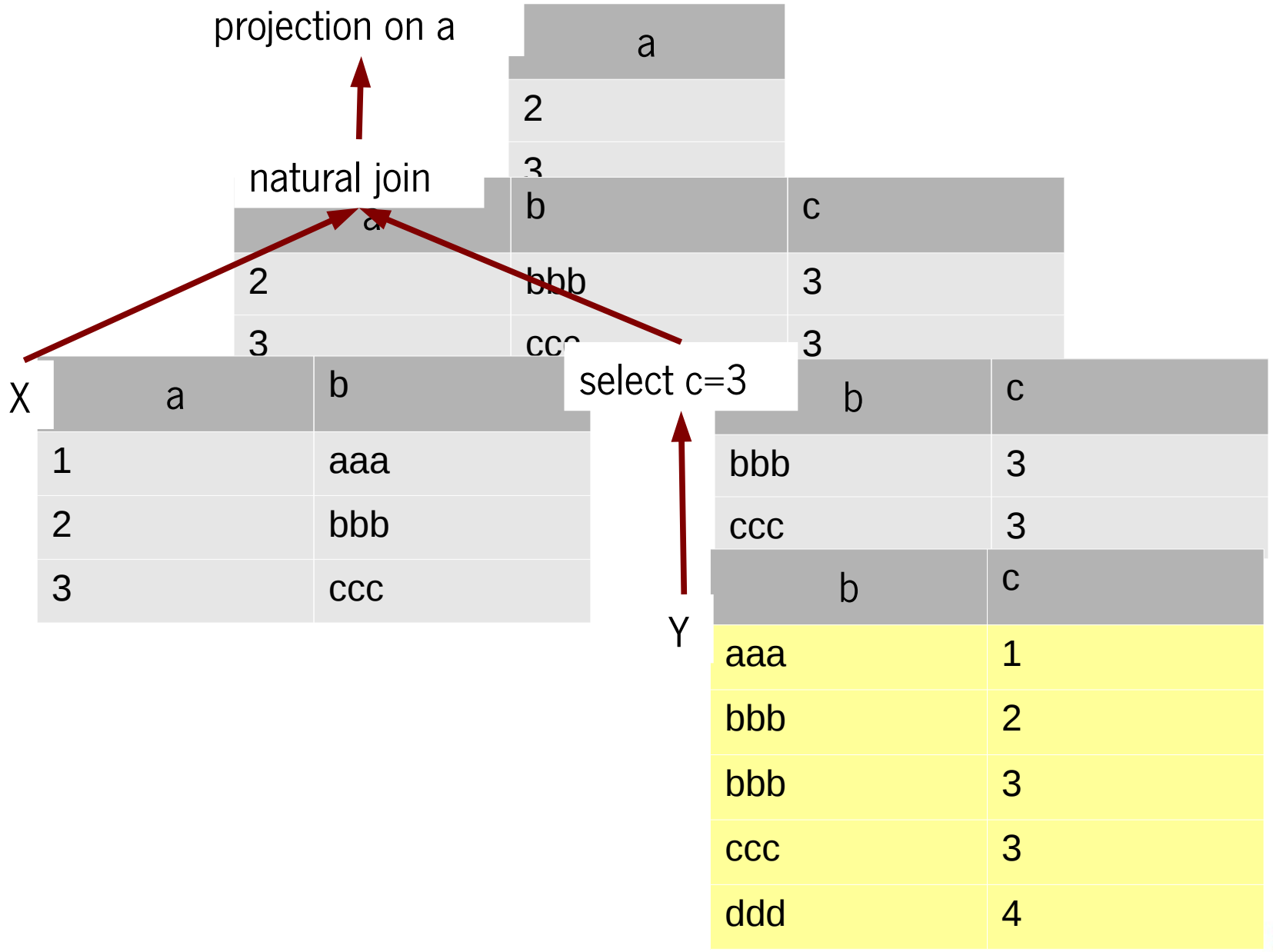
- Each operator is a function:
  - Returns a relation
  - Parameters are other relations
- Computation order:
  - From leaves to root




```
SELECT a FROM x NATURAL JOIN y WHERE c = 3
```


HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with materialization



Execution with materialization

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with materialization

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with materialization

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with materialization

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with materialization

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with materialization

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Iteration

- Each operator is an iterator
  - Typical interface:
    - has_next() / next()
- Computation order:
  - From leaves to root, for each record

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with iteration

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with iteration

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with iteration

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with iteration

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with iteration

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Execution with iteration

a
2
3

a	b	c
2	bbb	3
3	ccc	3

a	b
1	aaa
2	bbb
3	ccc

b	c
bbb	3
ccc	3

b	c
aaa	1
bbb	2
bbb	3
ccc	3
ddd	4

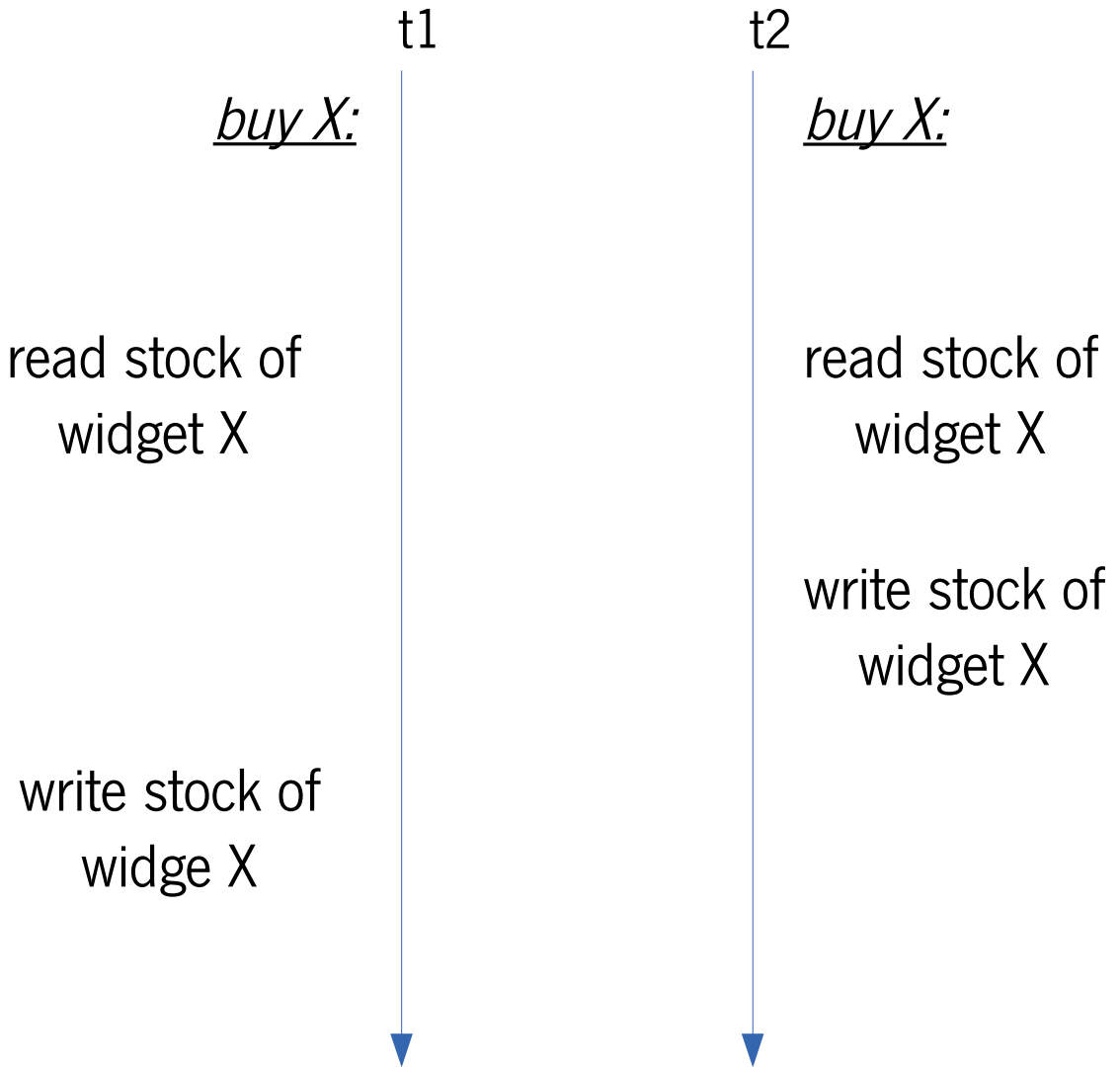
HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Conclusions

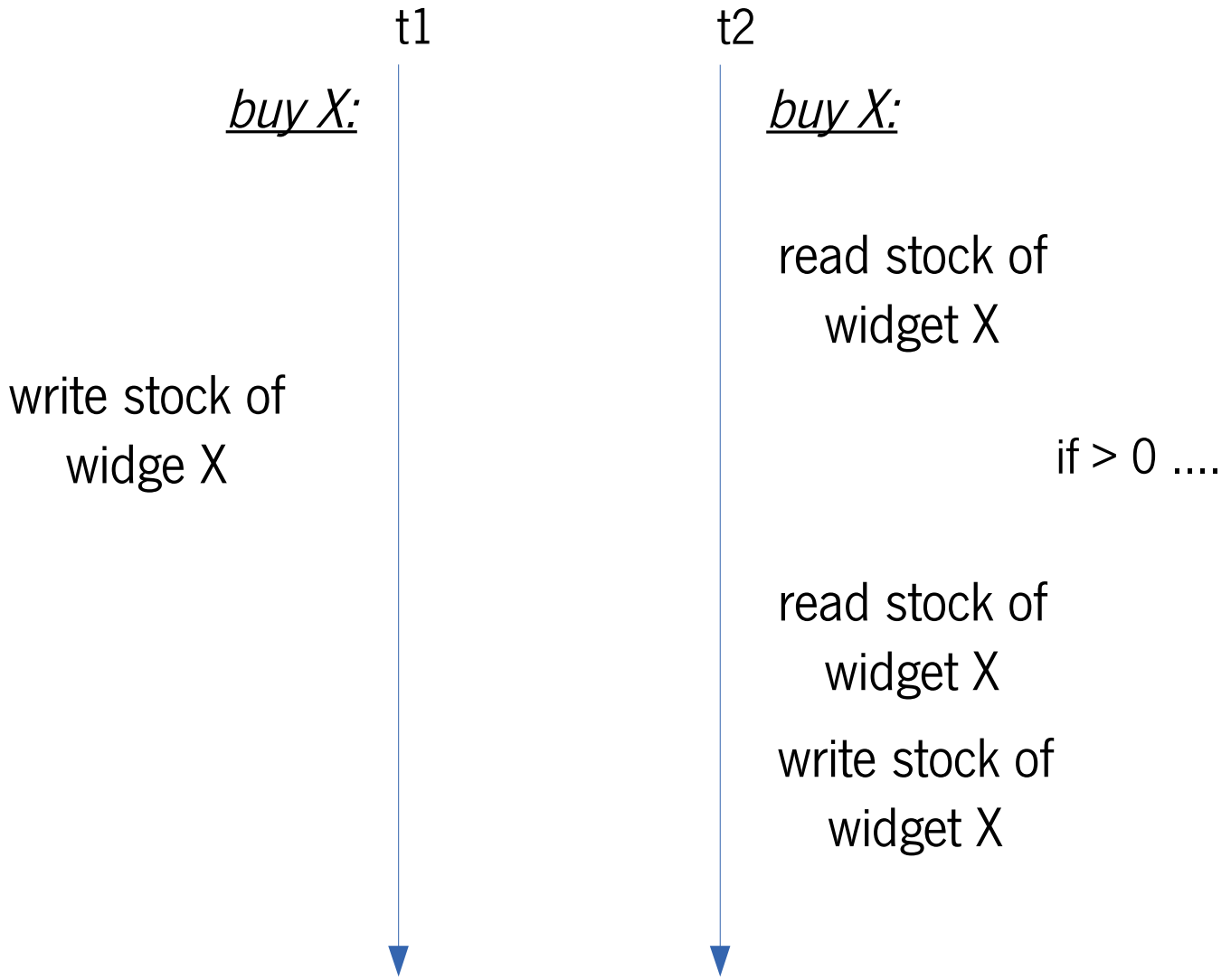
- The DBMS is lazy:
  - Query execution happens on demand
- Consequences:
  - At most one active ResultSet in the same Connection
  - Don't force it to work when not needed
  - Avoid operations that cannot be executed on demand
    - Sort
    - Aggregates

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Lost update

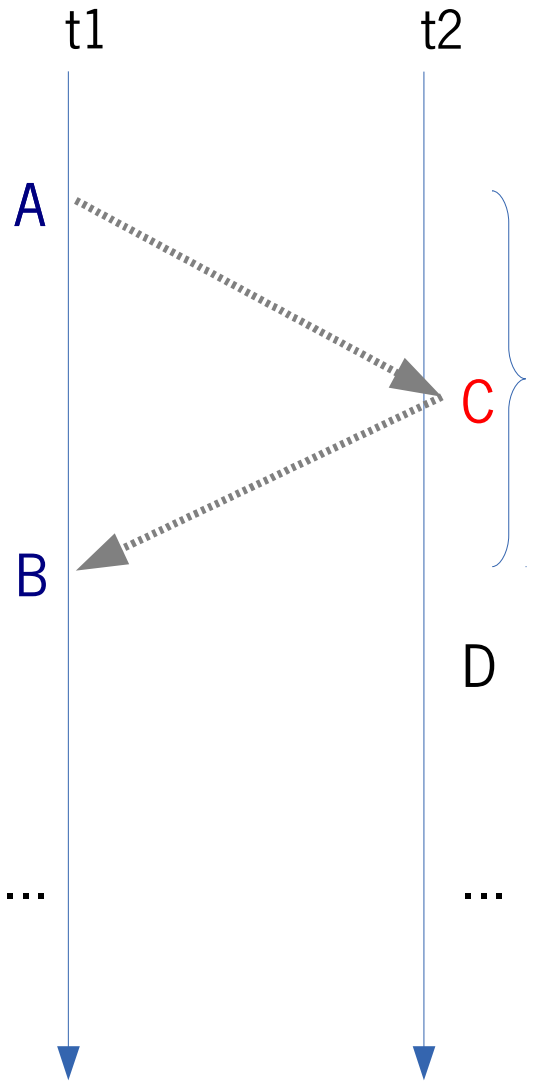

t1
t2
buy X:
buy X:
read stock of widget X
read stock of widget X
write stock of widget X
write stock of widge X

Non-repeatable read



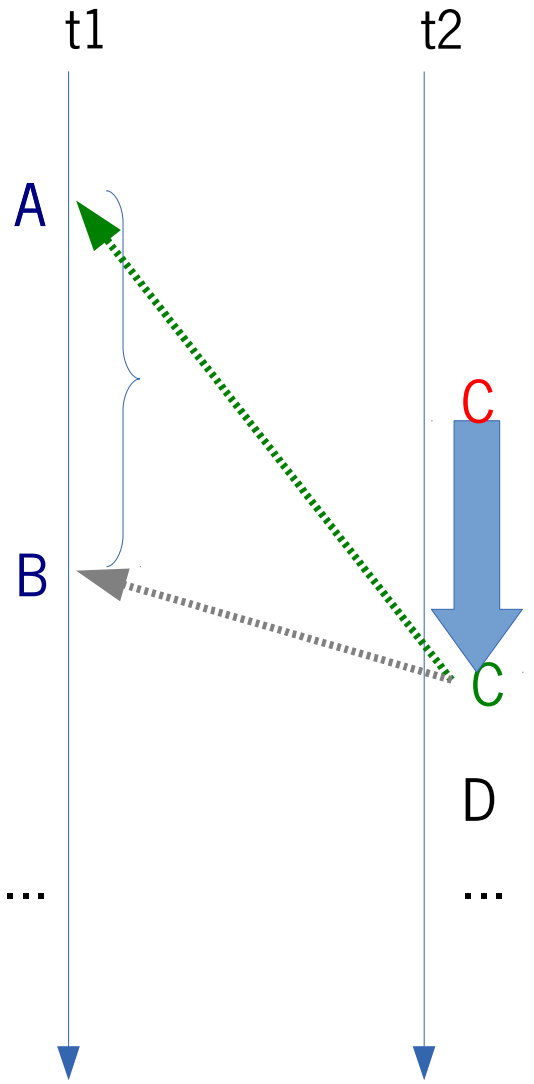
General problem

- Transaction execution overlaps in time
- In detail, some operation C should not be happening between operations A and B...



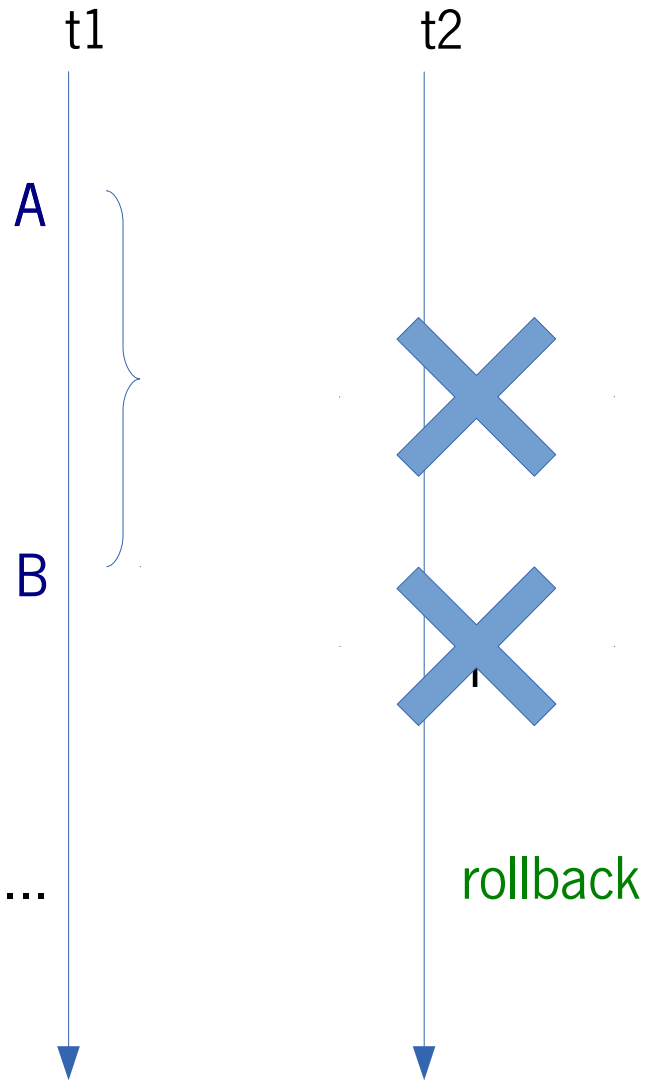
General approach

- Postpone C
  - t1 precedes t2



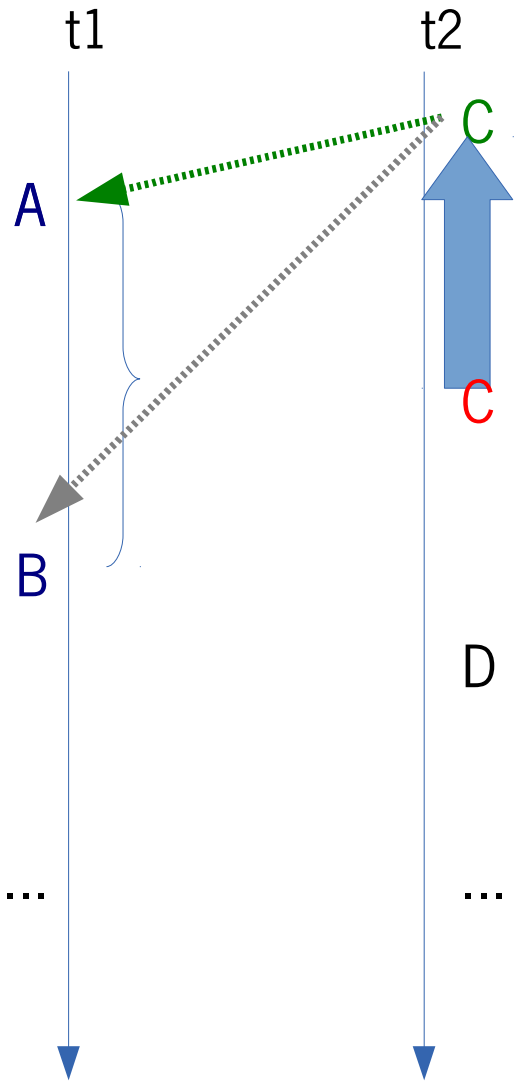
General approach

- Remove C and related operations
  - t1 executes alone



General approach

- Anticipate C (I.e. execute C before the application has requested it!)
  - t2 precedes t1
- (Can you propose a mechanism to do this!?)



Conclusions

- The DBMS deals with concurrent data access and modification
- Consequences:
  - Delimit transactions and set isolation level
  - Expect (unexpected) rollbacks
  - Take advantage of rollback to simplify code (prefer slide 6 to 5)

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

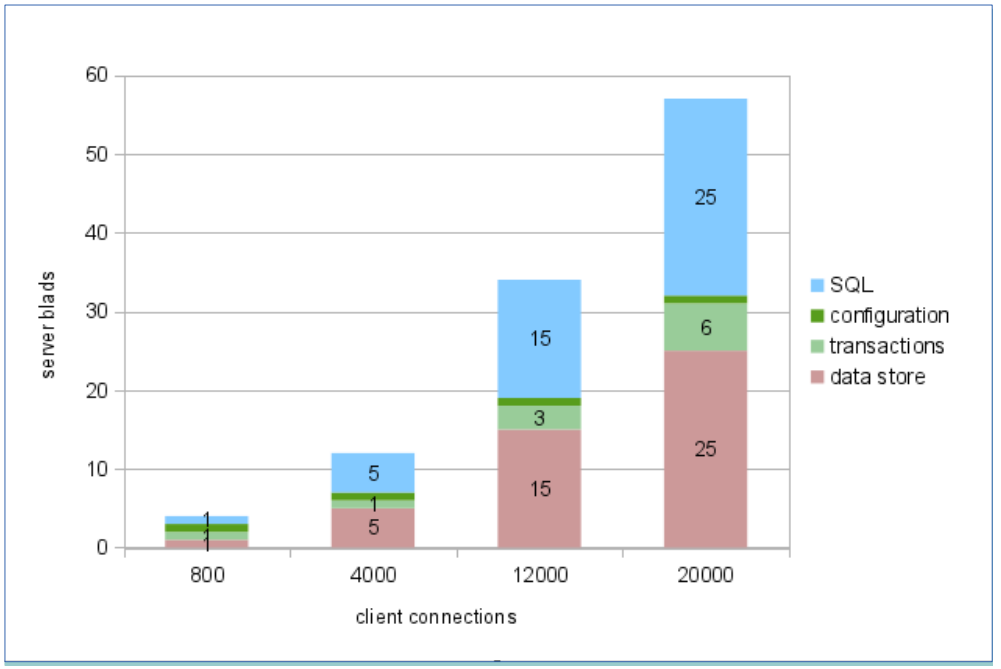
Database research @ HASLab

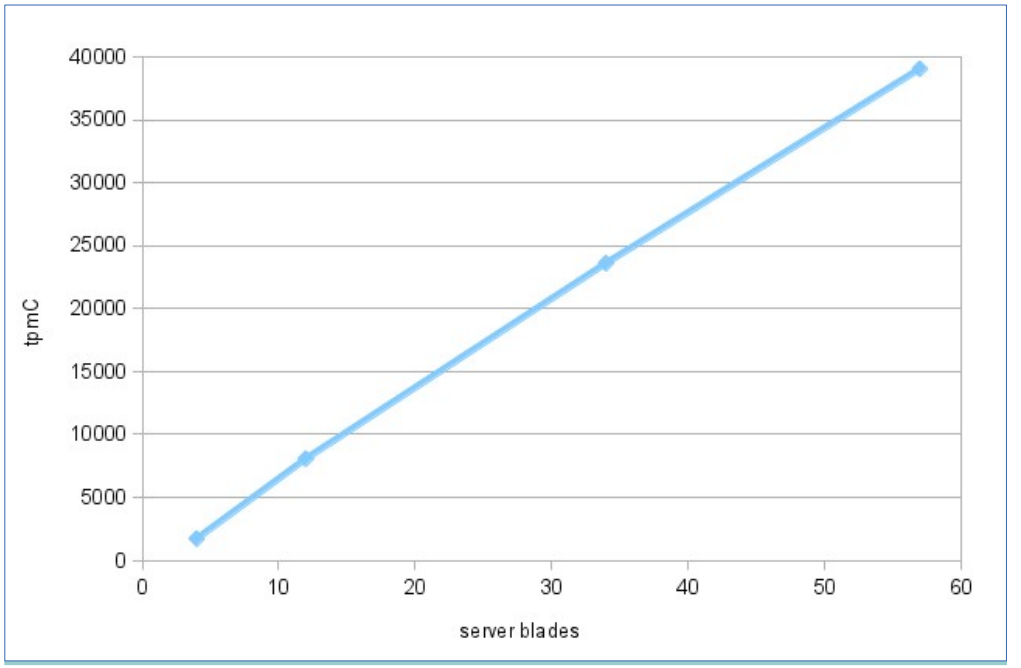
- CumuloNimbo: Highly Scalable Transactional Multi-Tier PaaS
  - http://cumulonimbo.eu
- CoherentPaaS: A Coherent and Rich PaaS with a Common Programming Model
  - http://coherentpaas.eu
- LeanBigData: Ultra-Scalable and Ultra-Efficient Integrated and Visual Big Data Analytics
  - http://leanbigdata.eu

https://www.youtube.com/watch?v=2ShTc0v-3Aw
https://www.youtube.com/watch?v=Z8I1Y57CCyQ
https://www.youtube.com/watch?v=R9QTmjxAdOk

Database research @ HASLab

- Standard SQL / JDBC
- Independently elastic layers
- Evaluated with TPC-C (OLTP standard)



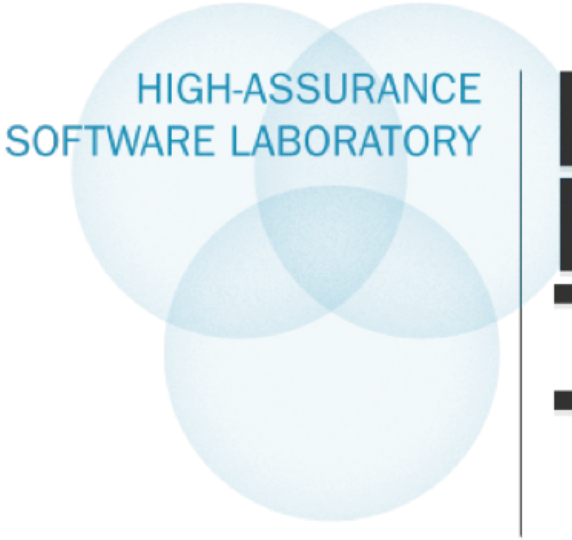


quad-core Intel Xeon X3220 @ 2.40GHz / 8GB RAM
500GB SATA disk / 1Gbits Ethernet

Distributed databases @ MEI/MIEI

- Distributed Systems and Cryptography
  - Transactional Distributed Systems
  - Fault Tolerance and Large Scale Distributed Systems
- Applications Engineering
  - Database Administration

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY



IMPROVING PRACTICE THROUGH THEORY