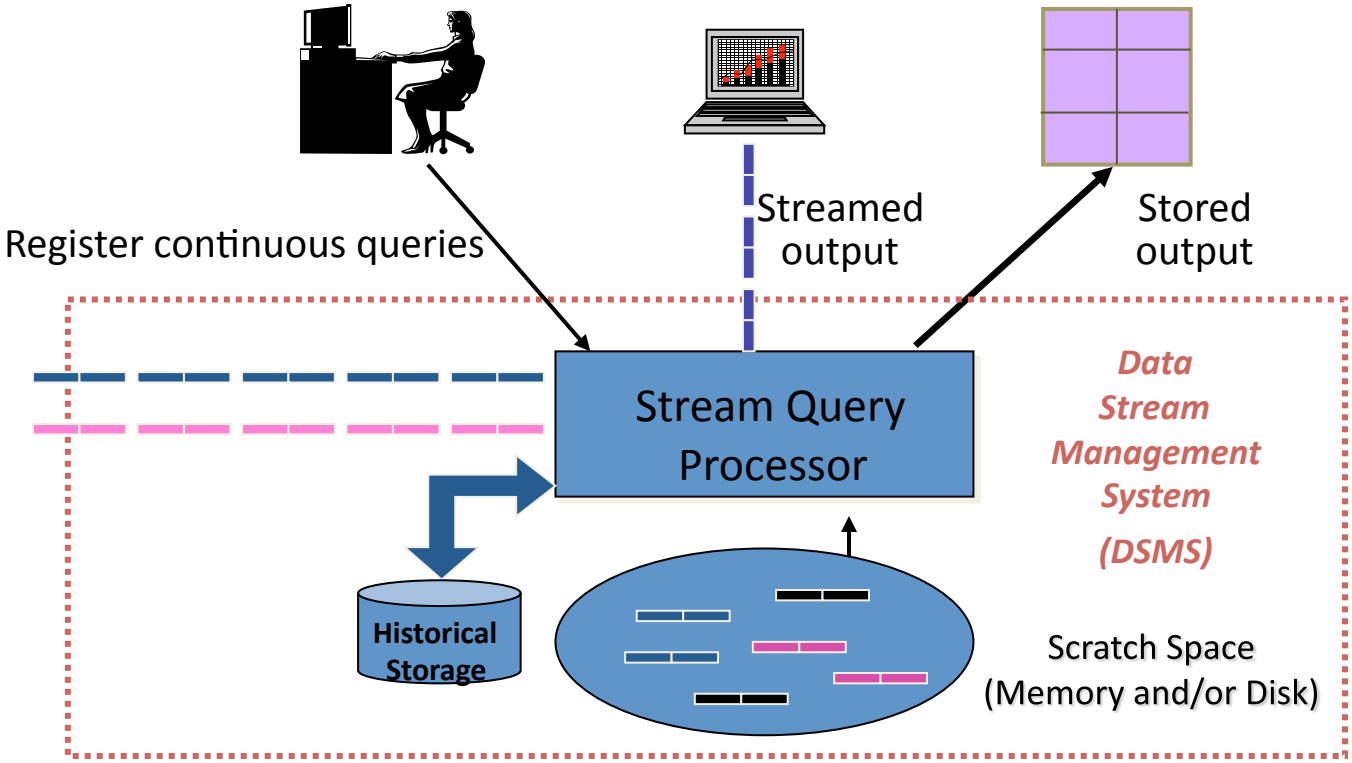
Securing the Borealis Data Stream Engine

Wolfgang Linder and Jorg Meier

Presented by Thao N. Pham

Introduction - DSMS

- DSMS – Data Stream Management Systems



(This figure is from the talk of Prof. Rajeev Motwani at PODS 2002)

Introduction

- 3 kinds of major threads to a DSMS (W, Lindner and J. Meier [14])
  - Improper release of information
    - Disclosure of data
      - Data value
      - Data schema
    - Disclosure of systems internals
      - Operations of other users
      - What else?

Introduction

- Major threats (cont.)
  - Improper modification of data
    - Changes outside the system:
      - Fake inputs
      - Modified query results
    - Changes inside the system:
      - Modify query's semantics, modify data between operators

Introduction

- Major threats (cont.)
  - Denial of Service attacks
    - Overload the system with huge input or with lots of expensive operators

Introduction

- Basic characteristics of a DSMS that affects the design of a security model for it:
  - Continuous data & continuous queries
    - Policy changes: change of data sensitivity, policies expiration
    - Merging of operators among multiple queries of different users

Contributions of the paper

- A security framework that addresses the mentioned threats
- An implementation on Borealis DSMS with an example scenario
- Experimental results showing that the security layer has negligible effect on the system's performance.

Proposed security framework

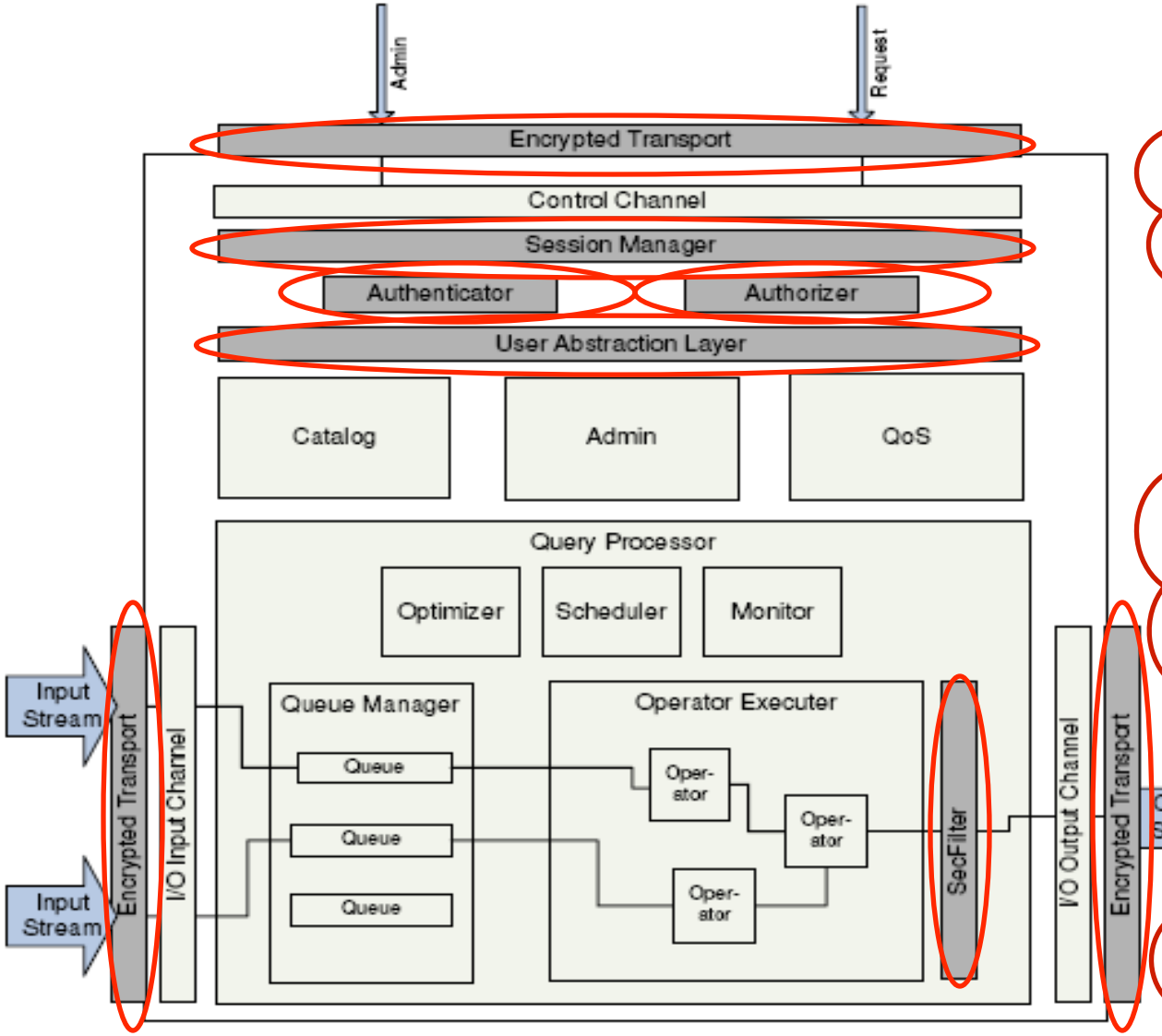


Figure 1: General Secure DSMS Architecture

Session manager and **Authenticator:** associating a request with a user id

Authorizer, User Abstraction Layer and **SecFilter:** ensuring that a user only gets the information he is allowed to see

Encrypted Transport : ensuring the confidentiality and integrity of the transferred requests and data

Proposed security framework (cont.)

- The paper focuses on the authorization part:
  - Authorizer: OxRBAC - owner-extended role-based access control
  - User abstraction layer: provides user-specific view of the system for each user, enforcing the ***object level*** security (*if a query does not satisfy object level security, it is rejected*)
  - SecFilter: ensures that results tuples only reach the authorized users, enforcing the ***data level*** security (why? – we will discuss this shortly)

OxRBAC Security model

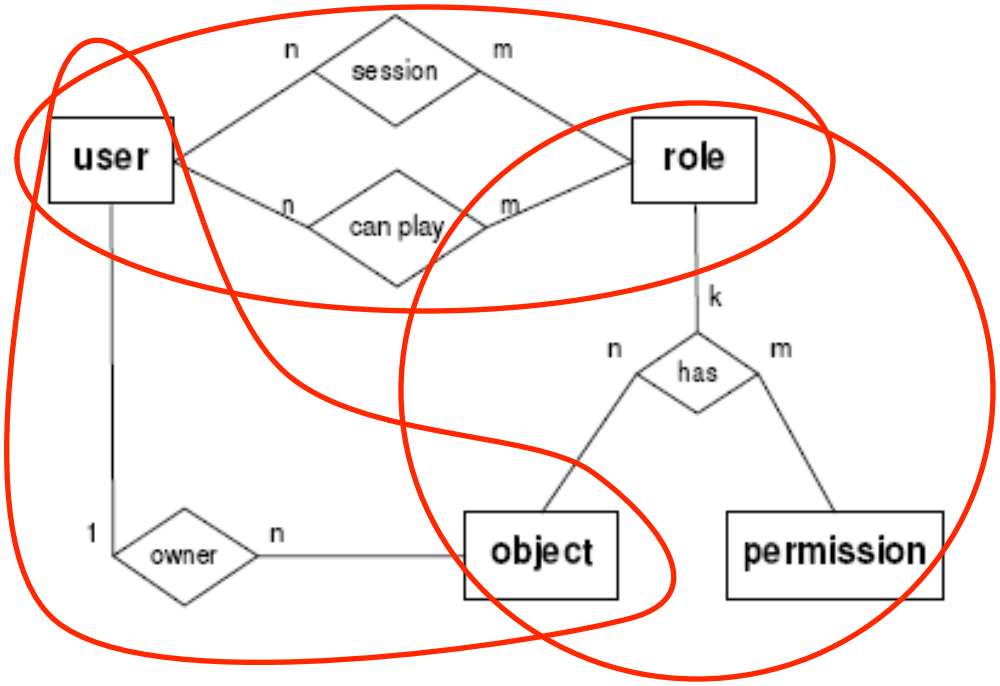


Figure 2: OxRBAC Security Model

A roles has a set of permissions on systems' objects

A user can play a set of roles in a session and has all the permissions of these roles

An object belongs to an owner, who can further grant access permissions on the object to others

Implementation on Borealis

- Secure a single Borealis server node (no distribution)
- The Admin module of Borealis is extended to SecAdmin, including the Session Manager, Authenticator, Authorizer, altogether form the User Abstraction Layer.
- SecFilter is incorporated into the Borealis's Query Processor

Implementation on Borealis (cont.)

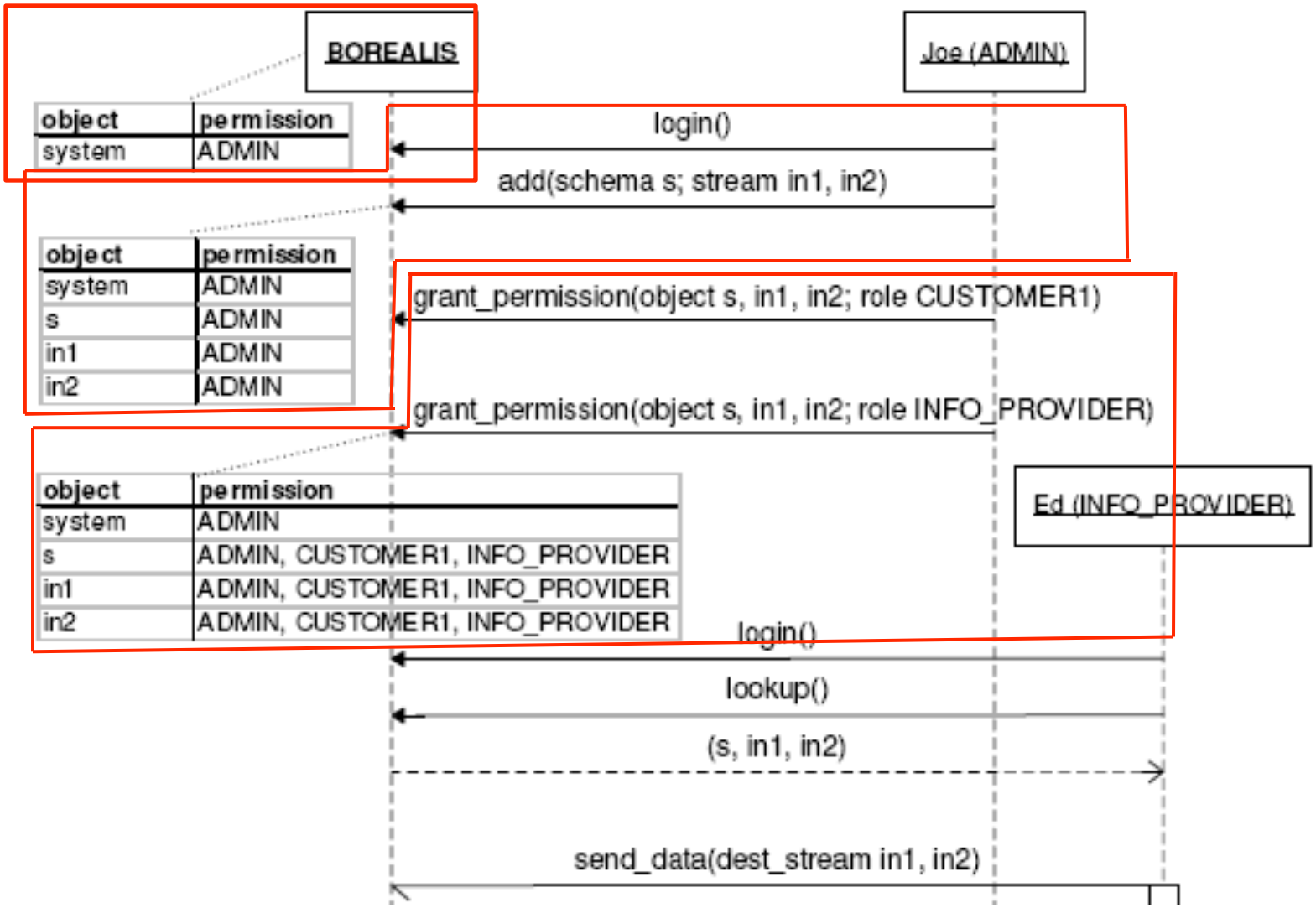
- Objects and permissions:
  - Objects: SCHEMA, STREAM, QUERY, SYSTEM
  - Permissions:
    - VIEW CATALOG, VIEW OBJECT, ADD, SUBSCRIBE, CHANGE PERM, CHANGE SYSTEM
    - SET QUERY STATUS, READ TUPLE (?)

An example scenario

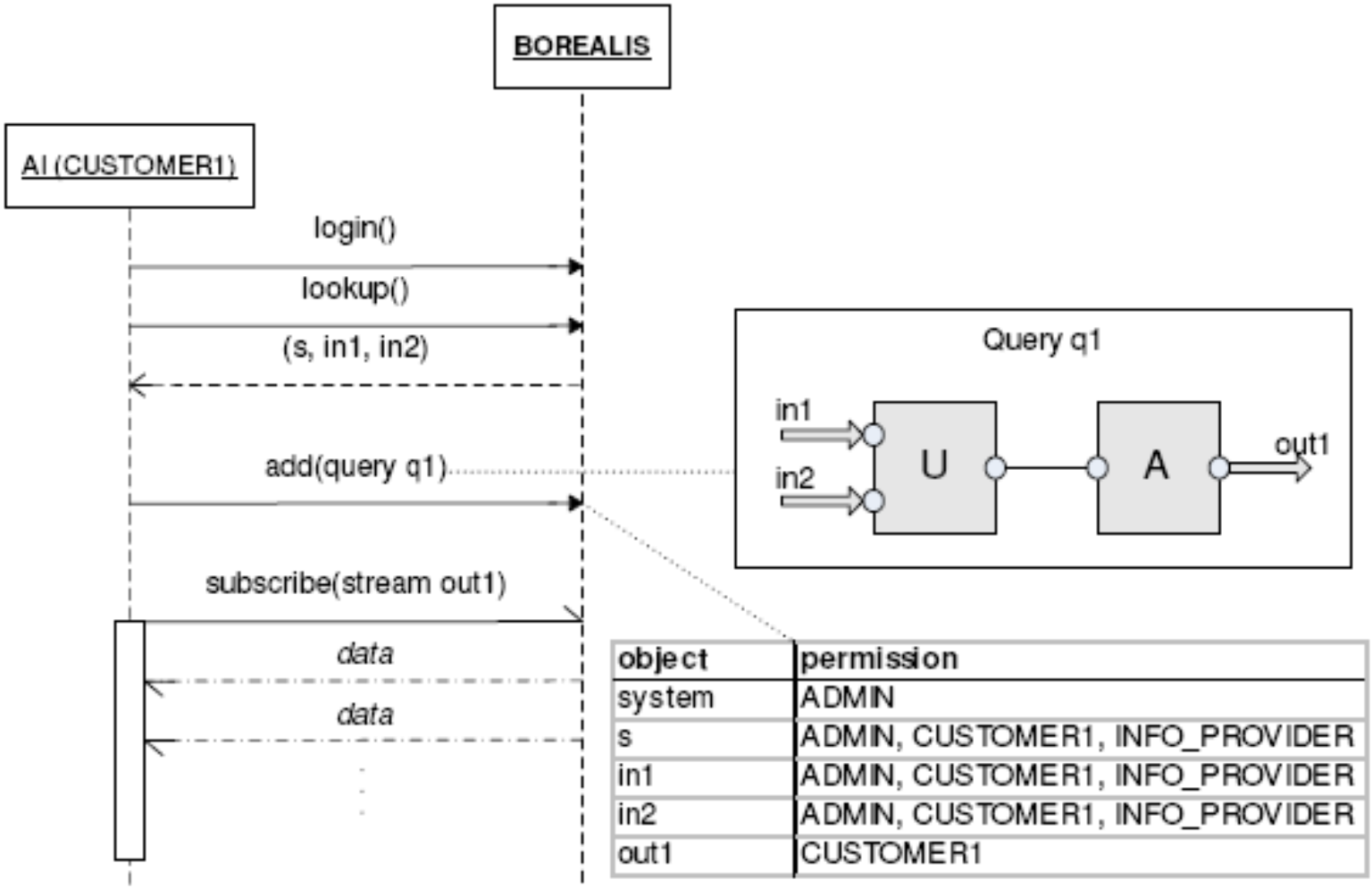
- An organization use Boreallis to provide data stream analysis for its customers. Certain partners provide stock prices as source data

- Roles and users:
  - ADMIN: Joe,
  - CUSTOMER1: Al
  - CUSTOMER2: Bob
  - INFO PROVIDER: Ed

An example scenario (tt)

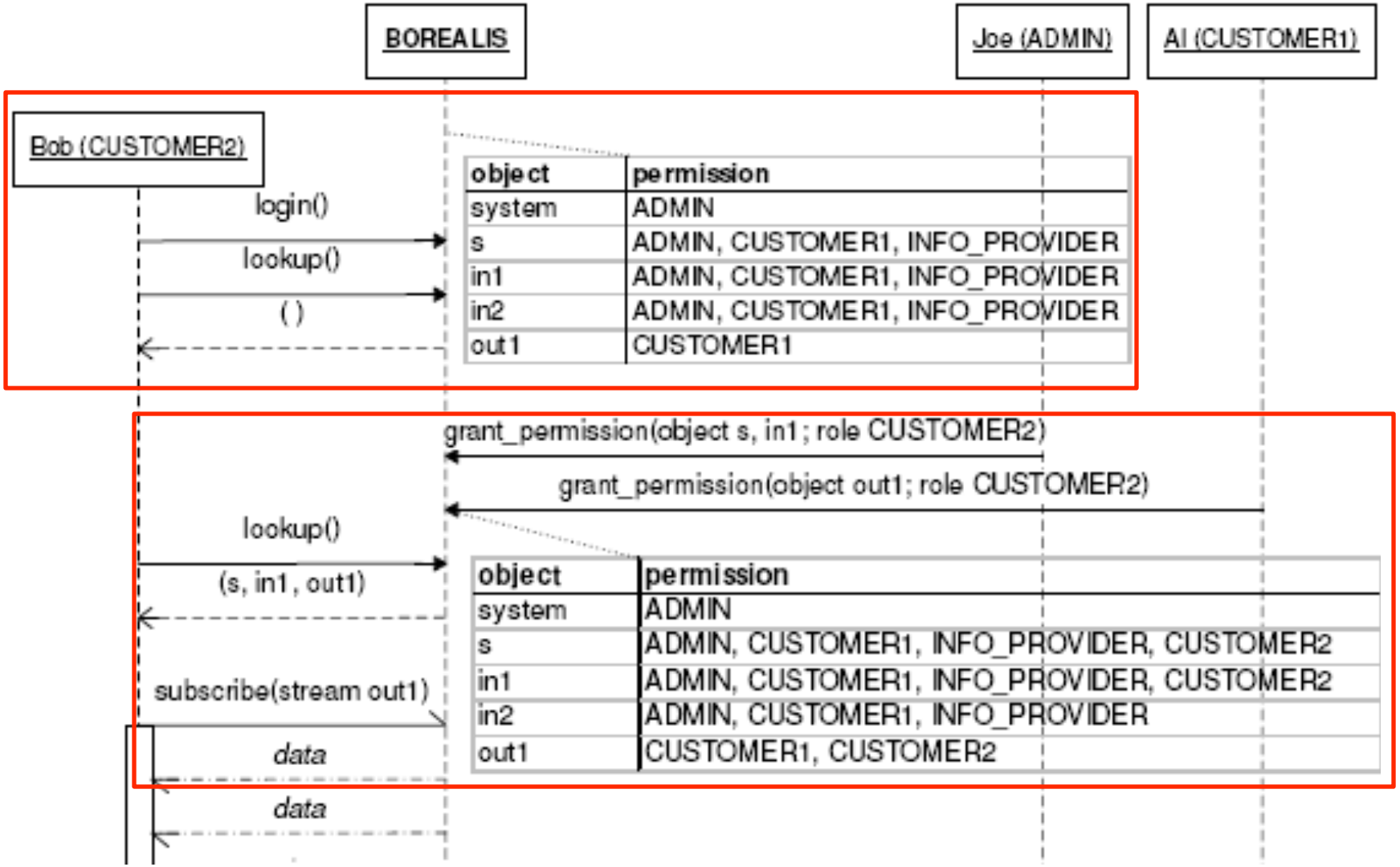


An example scenario (cont.)



object	permission
system	ADMIN
s	ADMIN, CUSTOMER1, INFO_PROVIDER
in1	ADMIN, CUSTOMER1, INFO_PROVIDER
in2	ADMIN, CUSTOMER1, INFO_PROVIDER
out1	CUSTOMER1

An example scenario (cont.)



How does the SecFilter work?

- The SecFilter check every result tuple with the security policy and makes sure the tuple is delivered only to authorized user.

- On its way from the source stream to the output, the tuple is marked to tell the SecFilter which data source it is related to (through 4 bytes in the tuple's header, 1 bit for each stream).

- In current implementation, if a tuple is related to multiple streams, the tuple is released only to those who have permissions on all the stream (*strong semantic*)

How does the SecFilter work? (cont.)

- In the example, since Bob is just granted permission on *in1*, the SecFilter only delivers to him those result tuples that are only related to the stream *in1*.
  - Actually, it's is not clear in this example how a tuple can be defined to be only related to one stream!
  - Is it better to define a policy that a user can subscribe to an output if he is granted to do so by the output's owner AND he has proper permission on all related objects?

How does the SecFilter work? (cont.)

- Other reasons to have SecFilter:
  - Policy changes: SecFilter allows the change to be applied immediately
    - But, there's a waste of processing cost to process tuples that are finally not used.
  - Merging of operators that belong to different queries (for optimization)
    - I don't think this is a problem, since the optimizer should never change the semantic of each query.

Experiments

- Goal of the experiments: shows that the additional layer of security does not affect the performance of the original DSMS
- Using 6 queries of increasing complexity
- Compare the average set up time and the total processing time of the systems with and without the SecFilter

Experimental results

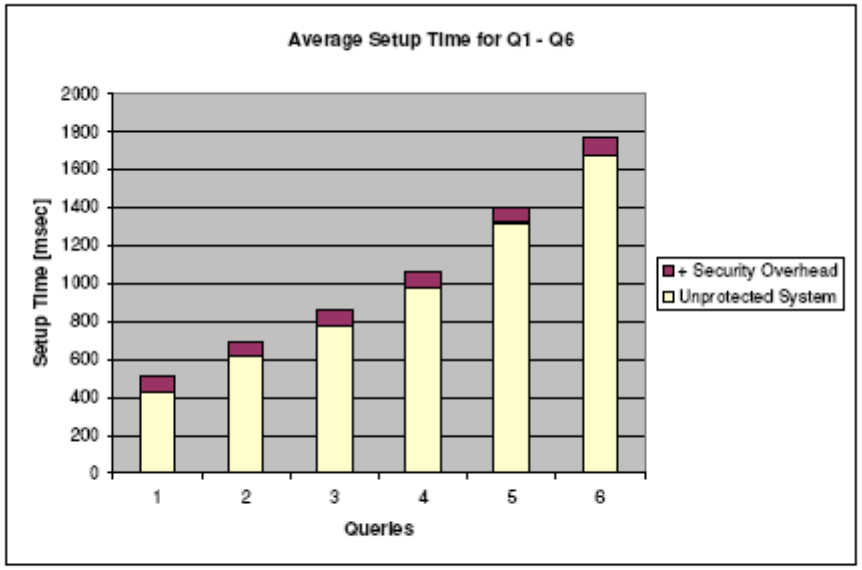


Figure 13: Average Setup Time for Q1 - Q6

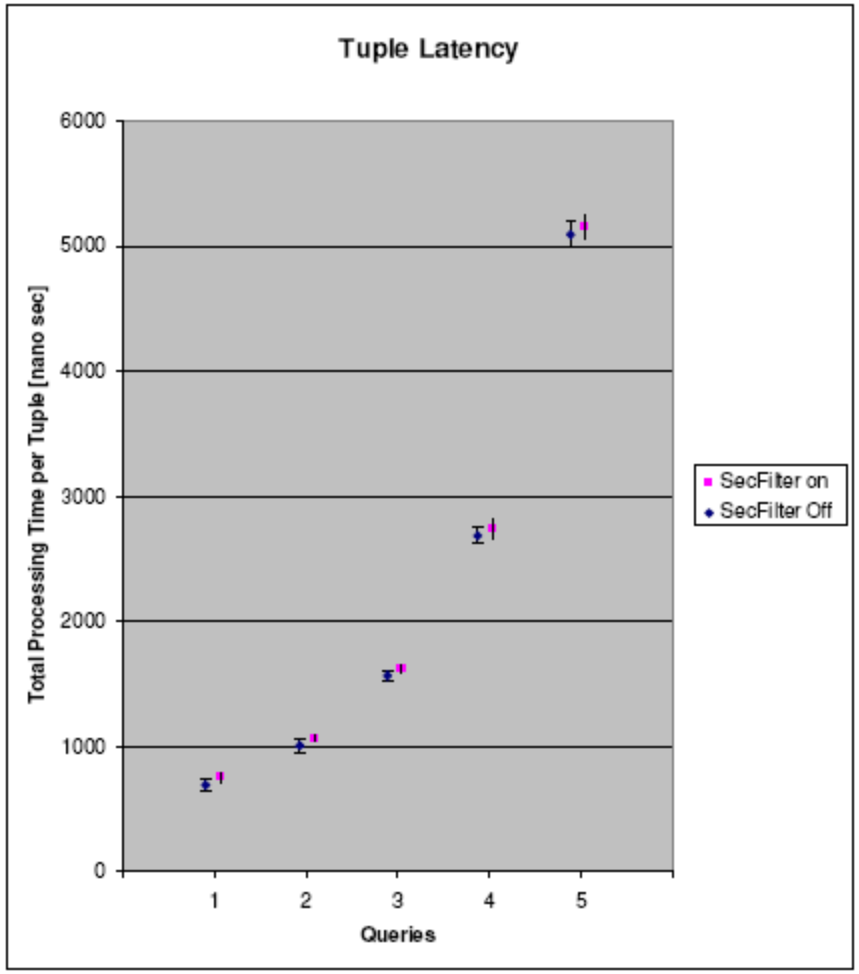


Figure 16: Tuple Latency

Conclusion

- Strengths:
  - First work on a security model for DSMS
  - The paper identifies some interesting security issues for a DSMS:
    - Consider a query as an object on which the owner can grant access permission to others.
    - Mention (although not thoroughly examine) the "aggregate reading" permission
    - Consider the changes of policies when queries are executed.

Conclusion

- Weaknesses:
  - Allows only access control at stream level. No support for more fine-grain access control.
    - Extension is not trivial because it would be very difficult for a general SecFilter to identify whether an output tuple satisfies a user's permissions.
  - More detailed policy definition is needed: e.g., a user can play several roles at the time he creates a object, then which role will be the owner of the object?
  - The experimental results are not really meaningful, given that there are just a few users, objects and permissions in the experiments.