Privacy-Aware Data-Intensive Applications

Michele Guerriero
Politecnico di Milano
michele.guerriero@polimi.it

Today's Information and Communication Technologies

- Advancements in ICTs enable the development of powerful and more efficient infrastructures and services:

➡ collection of big data from different sources

➡ increase demand for **Data-Intensive Applications** (DIAs)

The Evolution of Modern Data Processing

- From Map-Reduce to Directed Acyclic Graph-based execution
- The rise of distributed stream processors for large-scale and real-time data processing
- The Lambda architecture for balancing batch and streaming computations
- The Google Dataflow Model: a unified programming model for both batch and streaming data pipelines

Are We Missing Something?

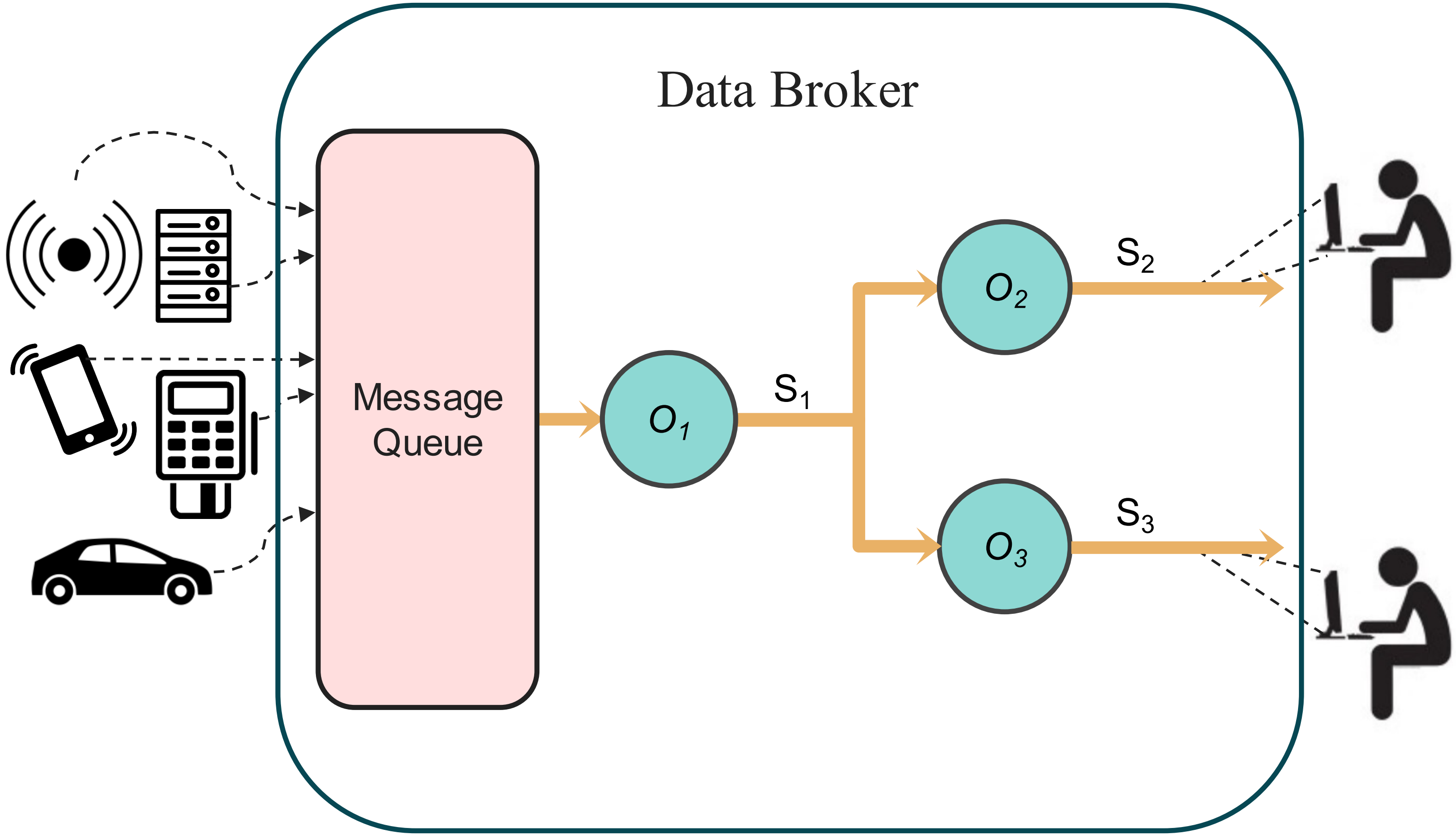
- The problem with Big Data is not just how do we process them
- In many cases Big Data are personal and often **sensitive**
- **Privacy** becomes more and more a primary concern in modern DIAs

Towards Privacy-Aware DIAs

- Data subjects should be able to specify requirements on how their data are used
- DIA designers and developers should be able to easily enforce such requirements
- Solution:

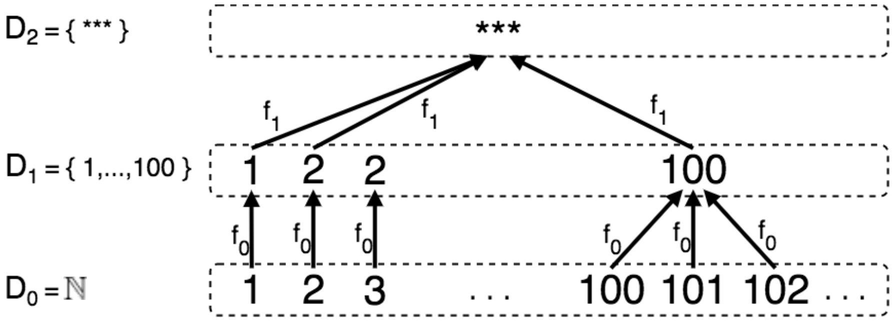
1. a language to let data subjects to specify privacy policies on modern DIAs
2. an automatic mechanism to enforce such policies

A Privacy Model for Modern Data-Intensive Applications



View Generalization Policies for Data Subject-Specific Streams

- **View generalization policies (VGP)**: allow data subject to define views over data **subject-specific streams**
- A VGP attached on a data subject-specific stream defines how tuples refering to a given data subject should be published when a given context holds
- Use Domain Generalization Hierarchy (DGH) to define views

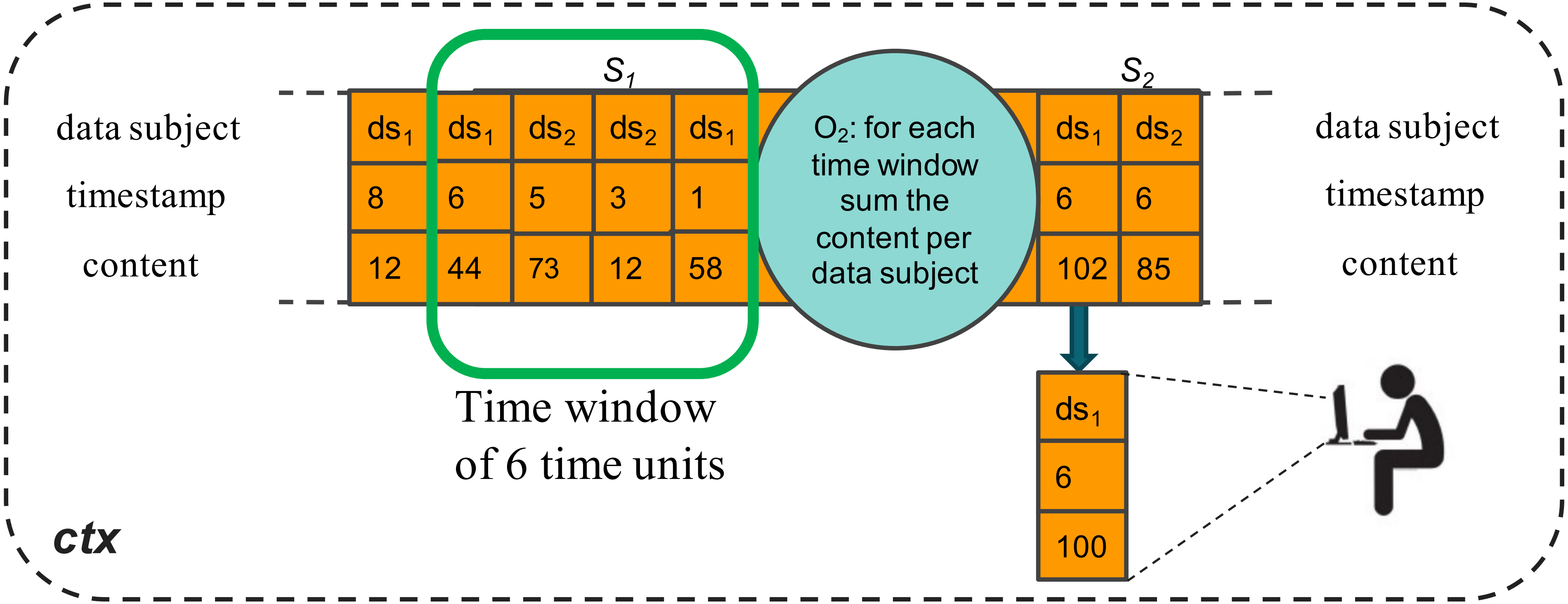


VGP Desired Effect

VGP by data subject ds_1:

```
    if context(ctx)
    then generalise($S_2$, 1)
```

Specifies to which level of the associated DGH the content of S_2 must be generalised



S_1

data subject	ds_1	ds_1	ds_2	ds_2	ds_1
timestamp	8	6	5	3	1
content	12	44	73	12	58

Time window of 6 time units

O_2: for each time window sum the content per data subject

S_2

data subject	ds_1	ds_2
timestamp	6	6
content	102	85

data subject	ds_1
timestamp	6
content	100

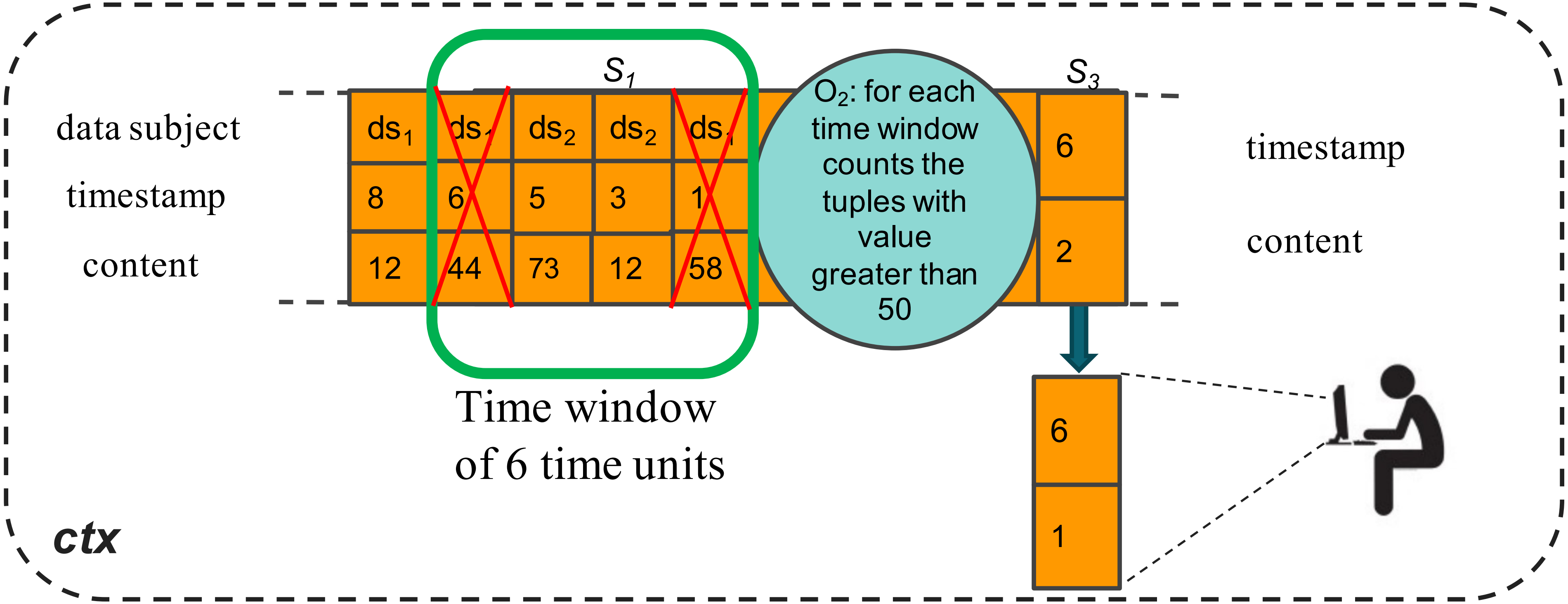
Data Subject Eviction Policies

- **Data subject eviction policies (DSEP)**: allow data owners to avoid their data to be considered by a given computation
- A DSEP attached on **a data subject-generic stream** *S* defines in which context tuples refering to a given data subject should be evicted from the input streams of the operator that produces *S*

DSEP Desired Effect

DSEP by data subject ds_1:

```
    if context(ctx)
    then evict($S_3$)
```



S_1

data subject	~~ds~~	~~ds~~	ds	ds	~~ds~~
	ds_1	~~ds_1~~	ds_2	ds_2	~~ds_1~~
timestamp	8	~~6~~	5	3	~~1~~
content	12	~~44~~	73	12	~~58~~

S_3

timestamp	6
content	2

O_2: for each time window counts the tuples with value greater than 50

Defining the Context

- Context modeled as a set of **contextual variables**:

1. **dynamic variables** change during a user session (e.g. the various real-time data computed by a given DIA, the user location, etc.)

   ➡ past values might be of interest

2. **static variables** does not change during a user session (e.g. the user identity, her purpose, etc.)

   ➡ only their current value is of interest

- **Policy enabling context**: Metric Temporal Logic formula specifying conditions over the past value of dynamic variables as well as the current value of static variables

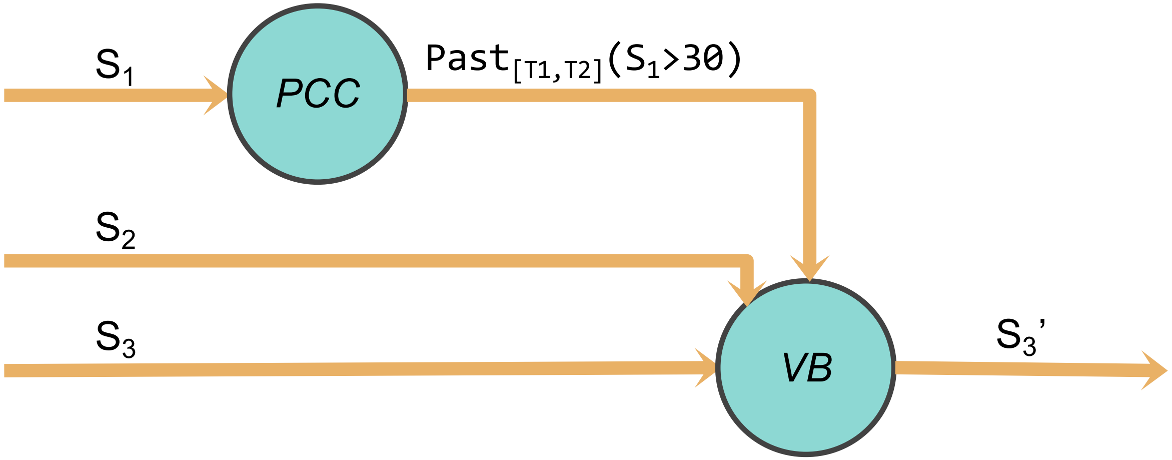
Automatic Policy Enforcement via Dataflow Rewriting

- Define a set of privacy enhancing dataflow operator
- **PastConditionChecker** (PCC): checks the validity of past conditions over dynamic variables
- **ViewBuilder** (VB): enforces the VGPs specified on a given data subject specific stream
- **DataSubjectEvictor** (DSE): enforces the DSEPs on a given data subject generic stream

Enforcing View Creation Policies

VGP by data subject *ds_1*:

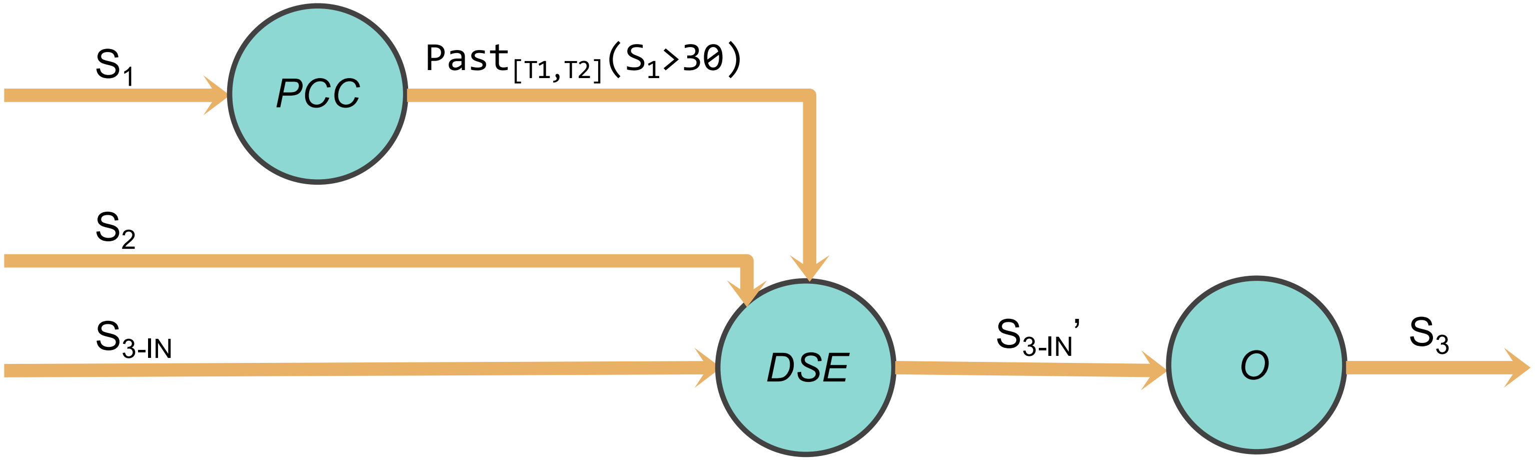
```
if Past[T1,T2](S1>30) & S2<10
then generalise(S3, 1)
```



Enforcing Data Subject Eviction Policies

DSEP by data subject *ds_1*:

```
if Past[T1,T2](S1>30) & S1<10
then evict(S3)
```



Evaluation Plan

- Performance evaluation focused on:

1. understanding the introduced performance overhead
2. understanding the main model variables that affect performance and how

- Apply trace-checking to verify the correctness of the policy enforcement implementation
- Apply the proposed approach on real-world use cases (how? How to find them?)
- How to compare when there are really no similar approaches out there?

Preliminary Results

- Prototype implementation on top of the Apache Flink dataflow processor

- Preliminary performance evaluation on a cluster of 30 cores:

Example Application 1	Latency	Throughput
No Policy	1.5 ms	61.11 t/ms
1 VGP	2.8 ms	56.24 t/ms

Example Application 2	Latency	Throughput
No Policy	1.9 ms	60.74 t/ms
1 DSEP	5.3 ms	57.14 t/ms

Future Work and Thesis Plan

- Rigorously follow the evaluation plan
- Dataflow computing and programming fits very well with model-based approaches:

1. Apply model-driven apporach to further simplify the development of privacy-aware DIAs
2. Extend results from previous research on model-driven engineering for DIAs

Conclusion

- Novel scenarios require new solutions to protect data
- Need to provide data owners with control over their data
- Design and development of privacy-aware applications needs to be made easy
- Data protection solutions are beneficial to both:

1. data owners (empowered with control)
2. data controllers (increased confidence of users, decreased liability)

Thank You!