Electronics and
Computer Science

UNIVERSITY OF
Southampton

Data Streams

COMP3211 Advanced Databases

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk
2014-2015

University of Southampton

From Databases to Data Streams

Traditional DBMS makes several assumptions:

- persistent data storage
- relatively static records
- (typically) no predefined notion of time
- complex one-off queries

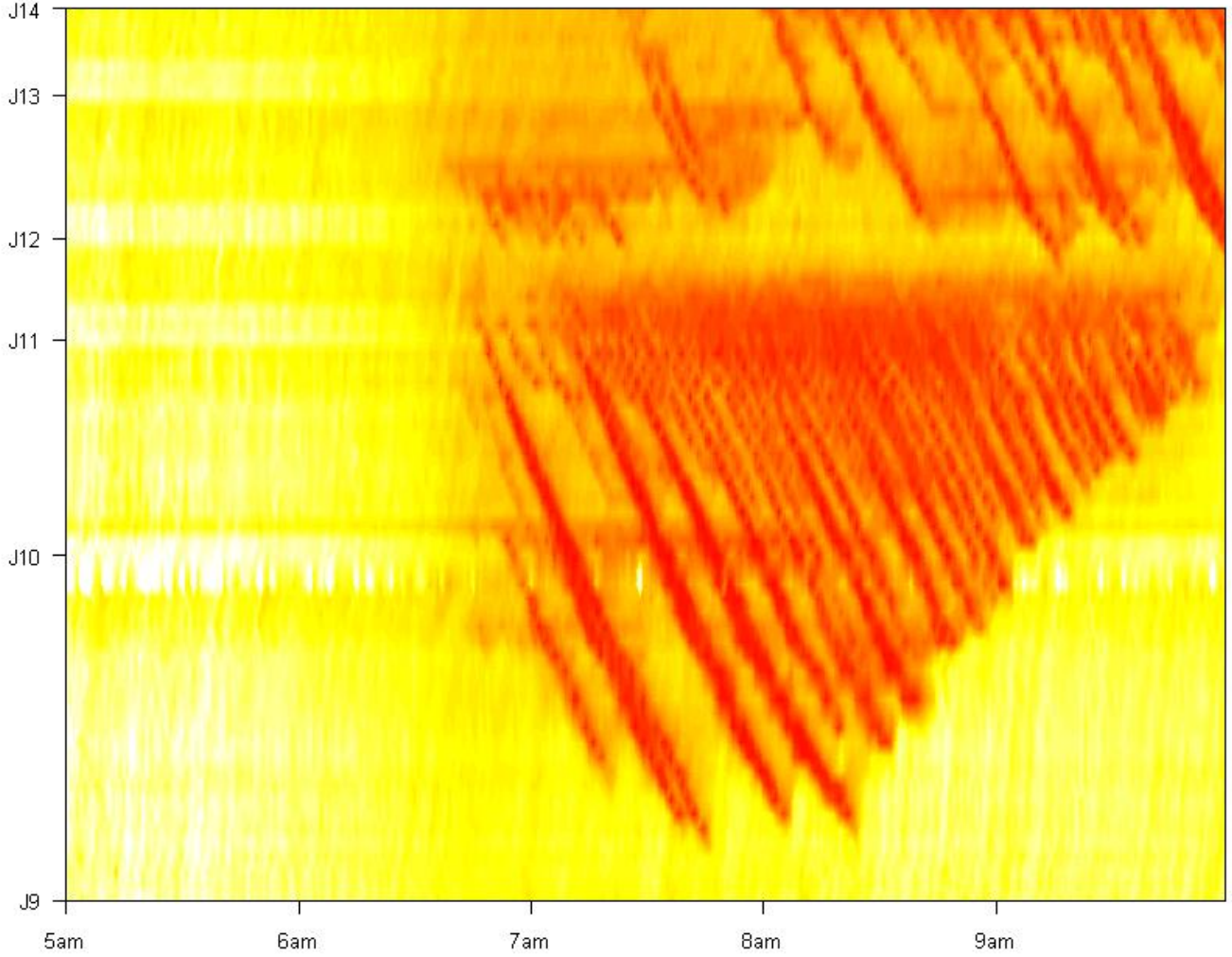
From Databases to Data Streams

Some applications have very different requirements:

- data arrives in real-time
- data is ordered (implicitly by arrival time or explicitly by timestamp)
- too much data to store!
- data never stops coming
- ongoing analysis of rapidly changing data

UNIVERSITY OF Southampton

Example Application: MIDAS



Application Domains

- Network monitoring and traffic engineering
- Sensor networks, RFID tags
- Telecommunications call records
- Financial applications
- Web logs and click-streams
- Manufacturing processes

UNIVERSITY OF Southampton

Data Streams

A (potentially unbounded) sequence of tuples

Transactional data streams: log interactions between entities

- Credit card: purchases by consumers from merchants
- Telecommunications: phone calls by callers to dialed parties
- Web: accesses by clients of resources at servers

Measurement data streams: monitor evolution of entity states

- Sensor networks: physical phenomena, road traffic
- IP network: traffic at router interfaces
- Earth climate: temperature, moisture at weather stations

One-Time versus Continuous Queries

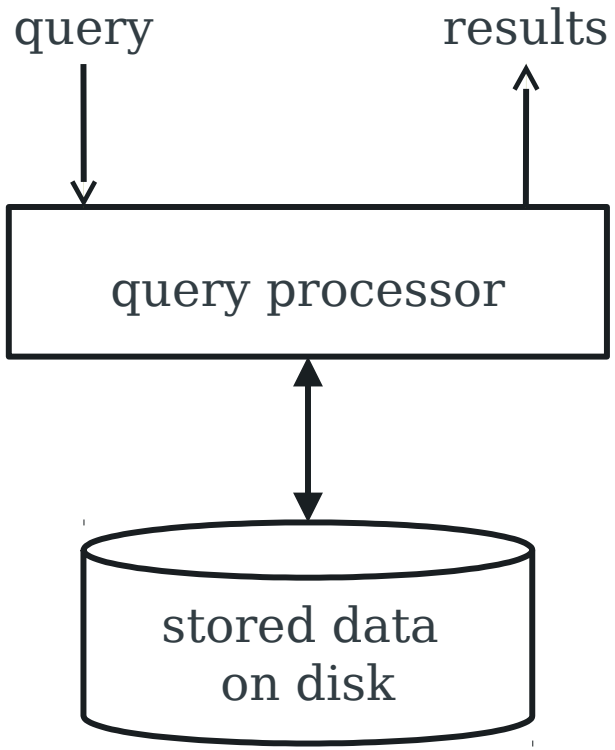
One-time queries

- Run once to completion over the current data set

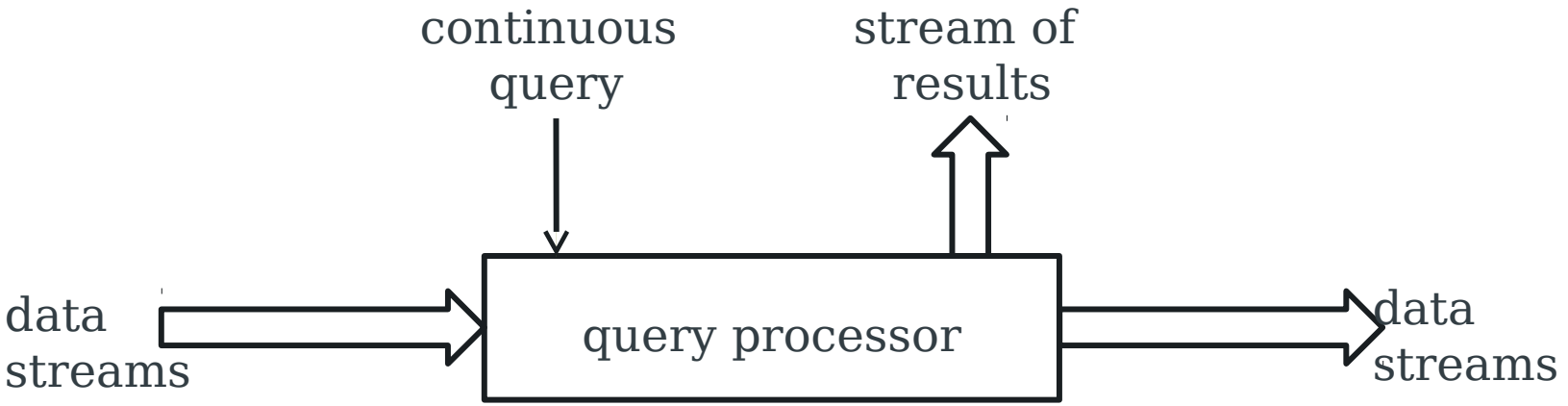
Continuous queries

- Issued once and then continuously evaluated over a data stream
  - "Notify me when the temperature drops below X"
  - "Tell me when prices of stock Y > 300"

Database Management System



Data Stream Management System (DSMS)



University of Southampton

DBMS versus DSMS

DBMS

- Persistent relations (relatively static, stored)
- One-time queries
- Random access
- “Unbounded” disk store
- Only current state matters

DSMS

- Transient streams (on-line analysis)
- Continuous queries (CQs)
- Sequential access
- Bounded main memory
- Historical data is important

UNIVERSITY OF Southampton

DBMS versus DSMS

DBMS

- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

DSMS

- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics

UNIVERSITY OF Southampton

A Motivation for Stream Processing

Over the past twenty-five years:

- CPU performance has increased by a factor of >1,000,000
- Typical RAM capacity increased by a factor of >1,000,000
- RAM access time has decreased by a factor of >50,000
- Typical HD capacity increased by a factor of >50,000

- HD access time has decreased by a factor of ~10

UNIVERSITY OF Southampton

Architectural Issues

DBMS

- Resource (memory, disk, per-tuple computation) rich
- Extremely sophisticated query processing, analysis
- Useful to audit query results of data stream systems.
- Query Evaluation: Arbitrary
- Query Plan: Fixed.

DSMS

- Resource (memory, per-tuple computation) limited
- Reasonably complex, near real time, query processing
- Useful to identify what data to populate in database
- Query Evaluation: One pass
- Query Plan: Adaptive

UNIVERSITY OF Southampton

Query Processing

Example: Continuous Query Language

Queries produce/refer to relations and streams

Based on SQL, with the addition of:

- Streams as new data type
- Continuous instead of one-time semantics
- Windows on streams (derived from SQL-99)
- Sampling on streams (basic)

UNIVERSITY OF Southampton

Query Processing

Construct query plan based on relational operators, as in an RDBMS

- Selection
- Projection
- Join
- Aggregation (group by)

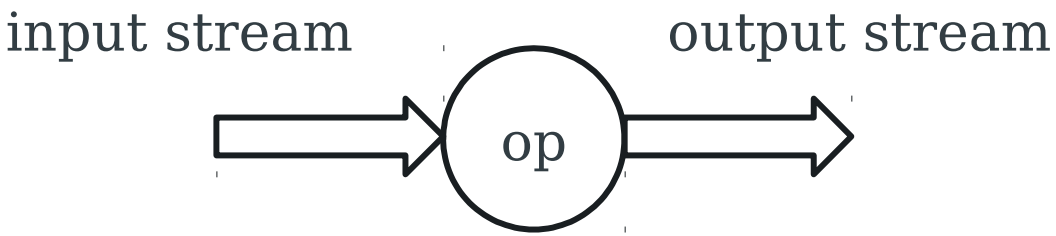
Combine plans from continuous queries (reduce redundancy)

Stream tuples through the resulting network of operators

Tuple-at-a-time Operators

Evaluation requires consideration of only one tuple at a time

- Selection and projection

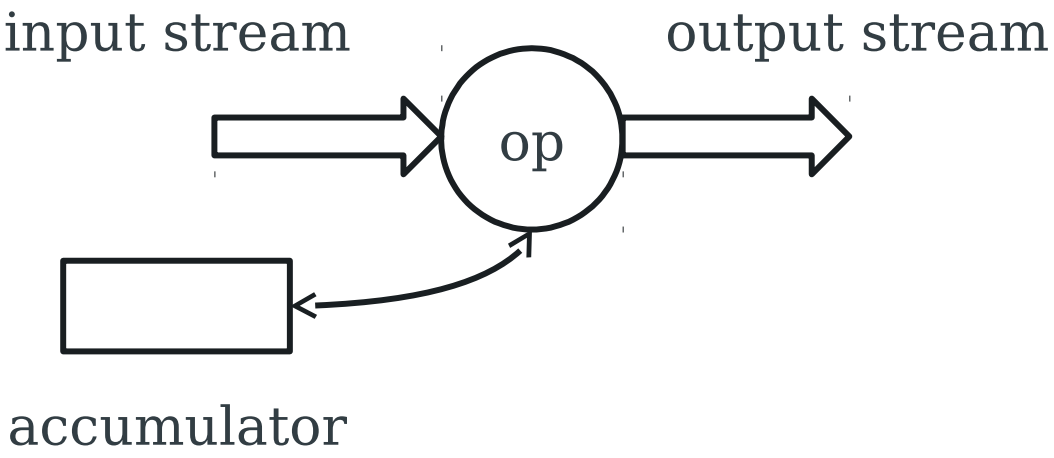


University of Southampton

Full Relation Operators

Some full relation operators can work on a tuple at a time

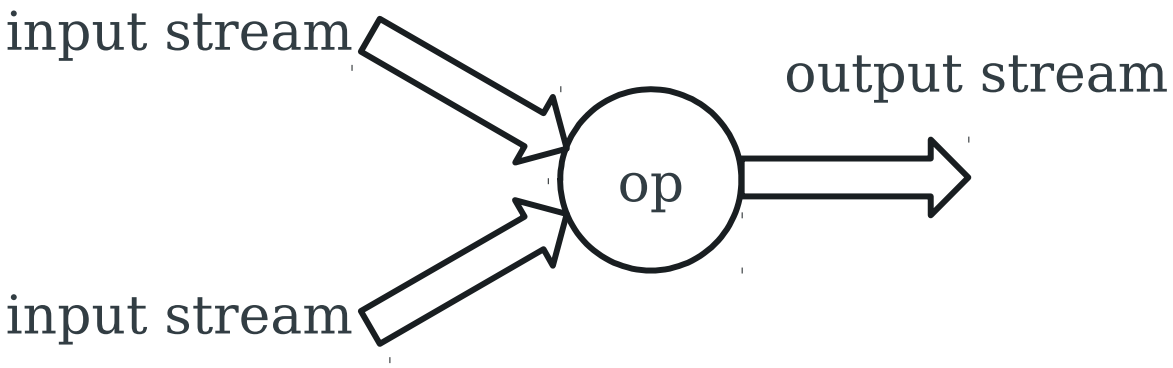
- Count, sum, average, max, min (even with group by)
- (order by, however, can't)



Full Relation Operators

Other (binary) full relation operators can't

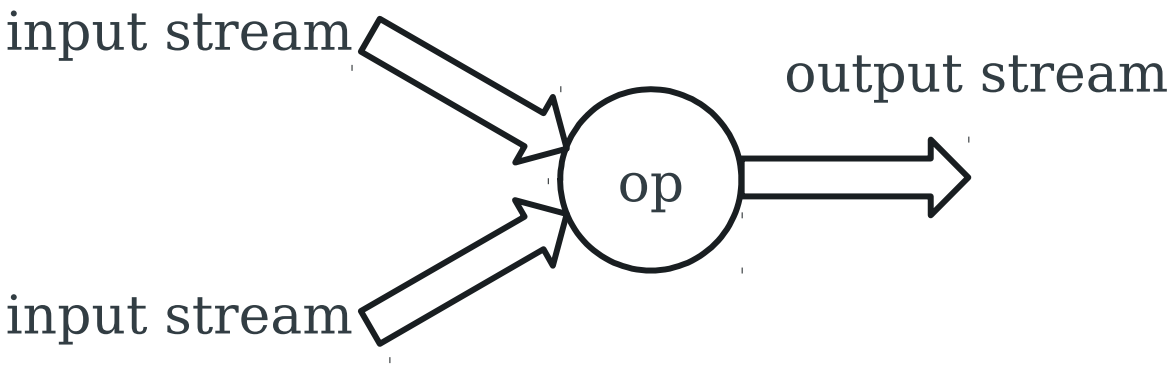
- Intersection, difference, product, join
- (union, however, can be evaluated tuple-by-tuple)



Full Relation Operators

May block when applied to streams

- no output until entire input seen, but streams are unbounded
- joins may need to join tuples that are arbitrarily far apart



UNIVERSITY OF Southampton

Relation/Stream Translation

Some relational operators can work directly on streams

- Selection, projection, union, some aggregates

Some relational operators need to work on relations

- Join, product, difference, intersection, other aggregates

Stream-to-relation operators

- Windows

Relation-to-stream operators

- Istream, Dstream, Rstream

UNIVERSITY OF Southampton

Windows

Mechanism for extracting a finite relation (synopsis) from an infinite stream

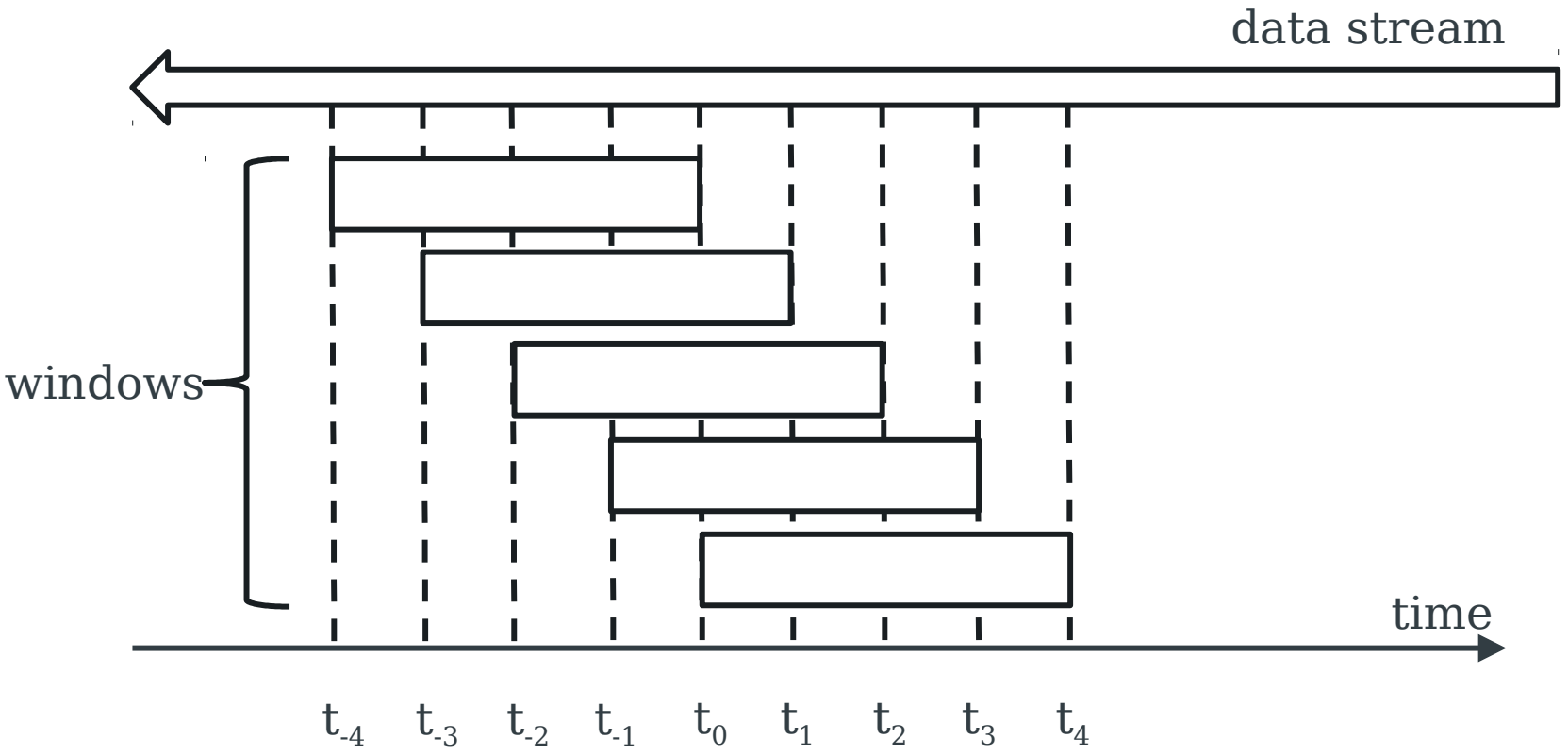
Various window proposals for restricting operator scope.

- Windows based on ordering attribute (e.g. last 5 minutes of tuples)
- Windows based on tuple counts (e.g. last 1000 tuples)
- Windows based on explicit markers (e.g. punctuations)
- Variants (e.g., partitioning tuples in a window)

Various window behaviours

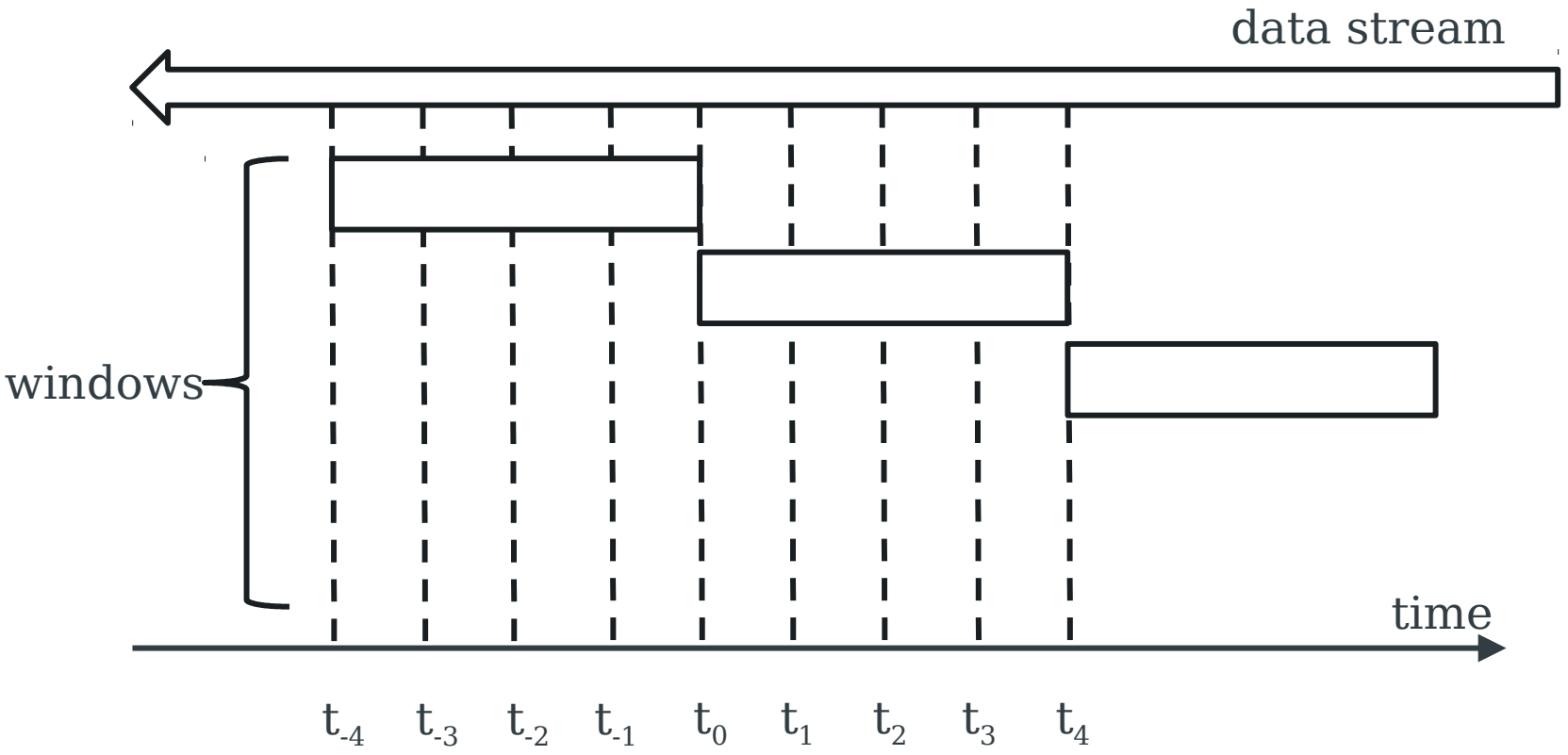
- Sliding, tumbling

Sliding Windows



UNIVERSITY OF Southampton

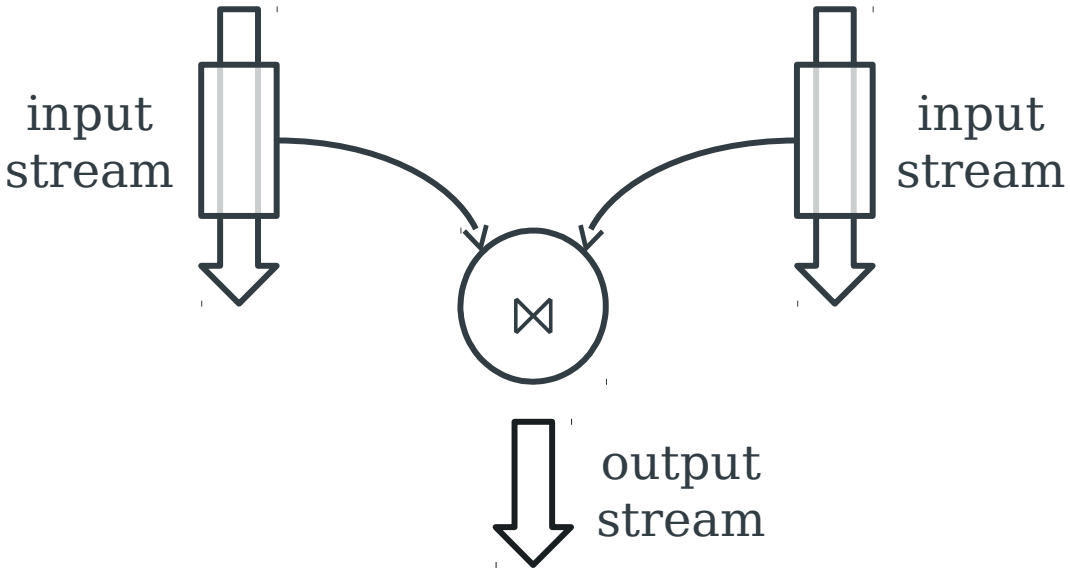
Tumbling Windows



Join Evaluation

Consider a stream-based join operation:

- a conventional join over a pair of windows on the input streams
- outputs a stream of tuples joined from the input streams



Scalability and Completeness

DBMS deals with finite relations

- query evaluation should produce all results for a given query

DSMS deals with unbounded data streams

- may not be possible to return all results for a given query
- trade-off between resource use and completeness of result set
- size of buffers used for windows is one example of a parameter that affects resource use and completeness
- can further reduce resource use by randomly sampling from streams

Relation-to-Stream Operators

Insert Stream (Istream)

- Whenever a tuple is inserted into the relation, emit it on the stream

Delete Stream (Dstream)

- Whenever a tuple is deleted from the relation, emit it on the stream

Relation Stream (Rstream)

- At every time instant, emit every tuple in relation on the stream

UNIVERSITY OF Southampton

Example CQL Query

```
SELECT Istream(*)
FROM S [rows unbounded]
WHERE S.A > 10
```

S is converted into a relation (of unbounded size!)

Resulting relation is converted back to a stream via Istream

UNIVERSITY OF Southampton

Example CQL Query

```
SELECT *
FROM S
WHERE S.A > 10
```

S is a stream – query plan involves only selection, so window is now unnecessary

UNIVERSITY OF Southampton

Example CQL Query

```
SELECT *
FROM    S1 [rows 1000],
     S2 [range 2 minutes]
WHERE S1.A = S2.A AND S1.A > 10
```

Windows specified on streams

- Tuple-based sliding window – [rows 1000]
- Time-based sliding window – [range 2 minutes]

UNIVERSITY OF Southampton

Example CQL Query

```
SELECT Rstream(S.A, R.B)
FROM S [now], R
WHERE S.A = R.A
```

Query probes a stored table R based on each tuple in stream S and streams the result

- [now] – time-based sliding window containing tuples received in last time step

UNIVERSITY OF Southampton

Query Optimisation

Traditionally relation cardinalities used in query optimiser

- Minimize the size of intermediate results.

Problematic in a streaming environment

- All streams are unbounded = infinite size!

Query Optimisation

Need novel optimisation objectives that are relevant when input sources are streams

- Stream rate based (e.g. NiagaraCQ)
- Resource-based (e.g. STREAM)
- QoS based (e.g. Aurora)

Continuous adaptive optimisation

Notable DSMS Projects

- Aurora, Borealis (Brown/MIT) – sensor monitoring
- Niagara (OGI/Wisconsin) – Internet XML databases
- OpenCQ (Georgia) – triggers, incr. view maintenance
- STREAM (Stanford) – general-purpose DSMS
- Telegraph (Berkeley) – adaptive engine for sensors

Further Reading

A. Arasu et al. STREAM: The Stanford Data Stream Management System, Technical Report, Stanford InfoLab, 2004.

A. Arasu, S. Babu and J. Widom. The CQL continuous query language: semantic foundations and query execution, The VLDB Journal, 15(2), 121-142, 2006.

M. Cherniack et al, Scalable Distributed Stream Processing, Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003), 2003.