

Join Operations in Temporal Databases

Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, Michael D. Soo

Georgios Giannikis



Outline

Motivation

What is different in Temporal Joins

Modeling Temporal Joins

Evaluating Temporal Joins

Conclusions

Own Assessment

A close-up, shallow depth-of-field photograph of several pencils resting on a light-colored, textured surface. The pencils are in various orientations, with some sharp points visible. A semi-transparent blue rectangular box is overlaid on the middle of the image, containing the word 'Motivation' in white text.

Motivation

Motivation

Why bother with Joins

- Most important relational operator
- Most frequent due to normalization

What makes Temporal Databases different

- Size:
 - Temporal DBs usually bigger due to versioning
- Evaluation:
 - Existing join operators focused on equality predicates
 - Temporal joins make heavy use of range predicates
i.e. `SELECT d.name, e.name`
`FROM departments d T-JOIN employees e on e.did = d.id`
`WHERE 01.01.2010 < time < 01.02.2010`

Contributions

A formal definition of Temporal Joins Operators

- How the notion of temporal data can be introduced in existing Join Operators

A taxonomy of all existing T-Join Operators

- A framework that can model all existing operators

A proper comparison of T-Join Operators

- Definition of space of database parameters that affect performance of T-Join Operators
- Empirical comparison of T-Join Operators



Introducing: The Time Dimension

OLIVE SCHWARTZ 2006 1951

What is a Temporal Join

- Join between two relations, both containing time-stamped tuples

i.e. Find the managers of all employees

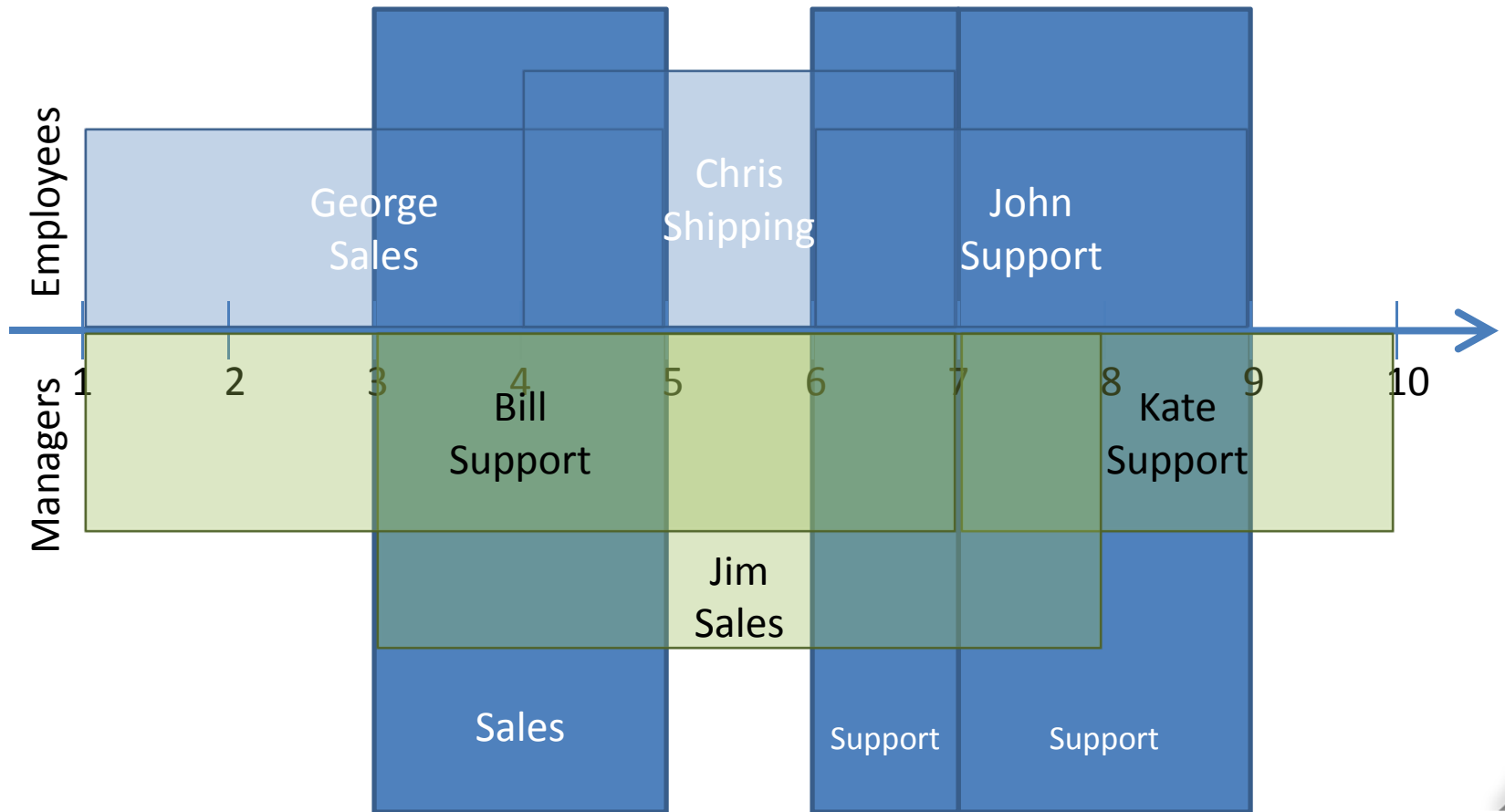
Employees		
Name	Department	TimeFrame
George	Sales	[1, 5]
Chris	Shipping	[4, 7]
John	Support	[6, 9]

$\bowtie^T$
E.Dprt = M.Dprt

Managers		
Name	Department	TimeFrame
Jim	Sales	[3, 8]
Bill	Support	[1, 7]
Kate	Support	[7, 10]

What is a Temporal Join

- Join between two relations, both containing time-stamped tuples



What is a Temporal Join

- Join between two relations, both containing time-stamped tuples

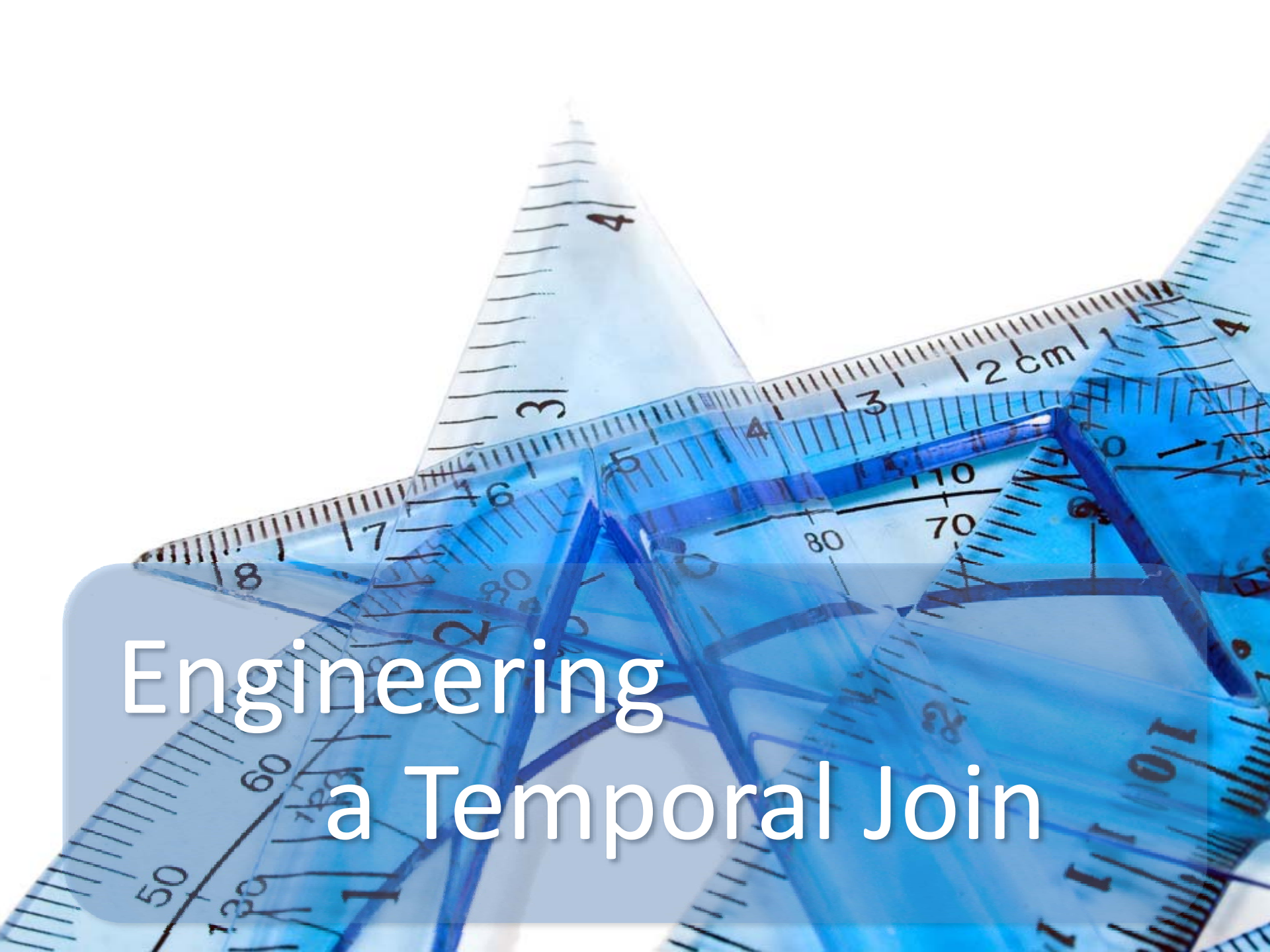
Employees		
Name	Department	TimeFrame
George	Sales	[1, 5]
Chris	Shipping	[4, 7]
John	Support	[6, 9]

$$\bowtie^T$$

E.Dprt = M.Dprt

Managers		
Name	Department	TimeFrame
Jim	Sales	[3, 8]
Bill	Support	[1, 7]
Kate	Support	[7, 10]

Employee	Manager	Department	TimeFrame
George	Jim	Sales	[3, 5]
John	Bill	Support	[6, 7]
John	Kate	Support	[7, 9]



Engineering a Temporal Join

Taxonomy

Nested Loop Join

- A loop inside a loop
- Straightforward

Index Based

- For each tuple r in R do
- Requires secondary data access
- If r and s satisfy join condition
- Not taken into account in this survey
- Then output the tuple $\langle r, s \rangle$

Partition Based

- Divide & Conquer

Sort-Merge Based

- Sort relations and
- Merge the sorted results

Duality

Partition based & Sort-merge based

- Introduce time notion in algorithms
- Allows for two level partitioning/sorting

Four different possibilities

- Only relational attributes
- Only time attribute
- Two-level: First on relational, then on time
- Two-level: First on time, then on relational

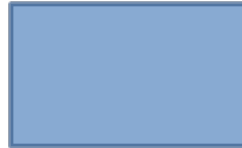
More Options

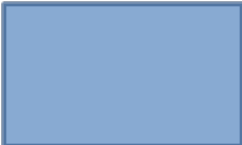
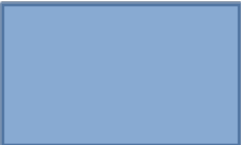
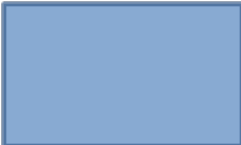
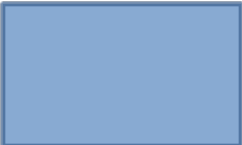
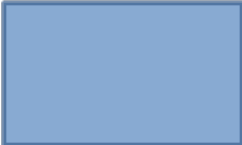
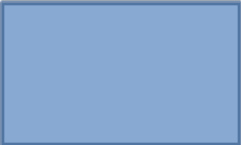
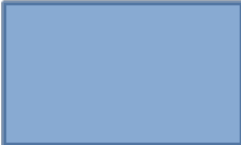
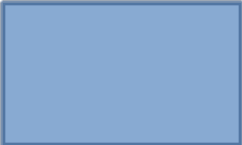
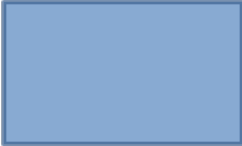
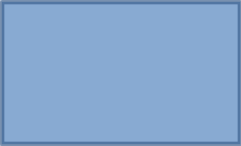
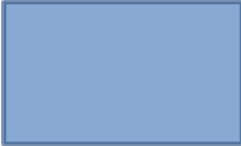
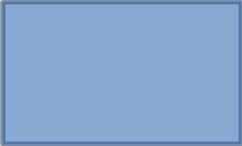
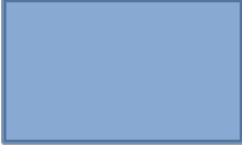
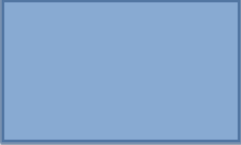
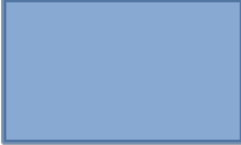
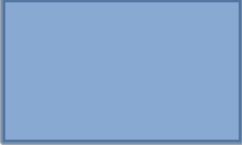
Buffer allocation strategies

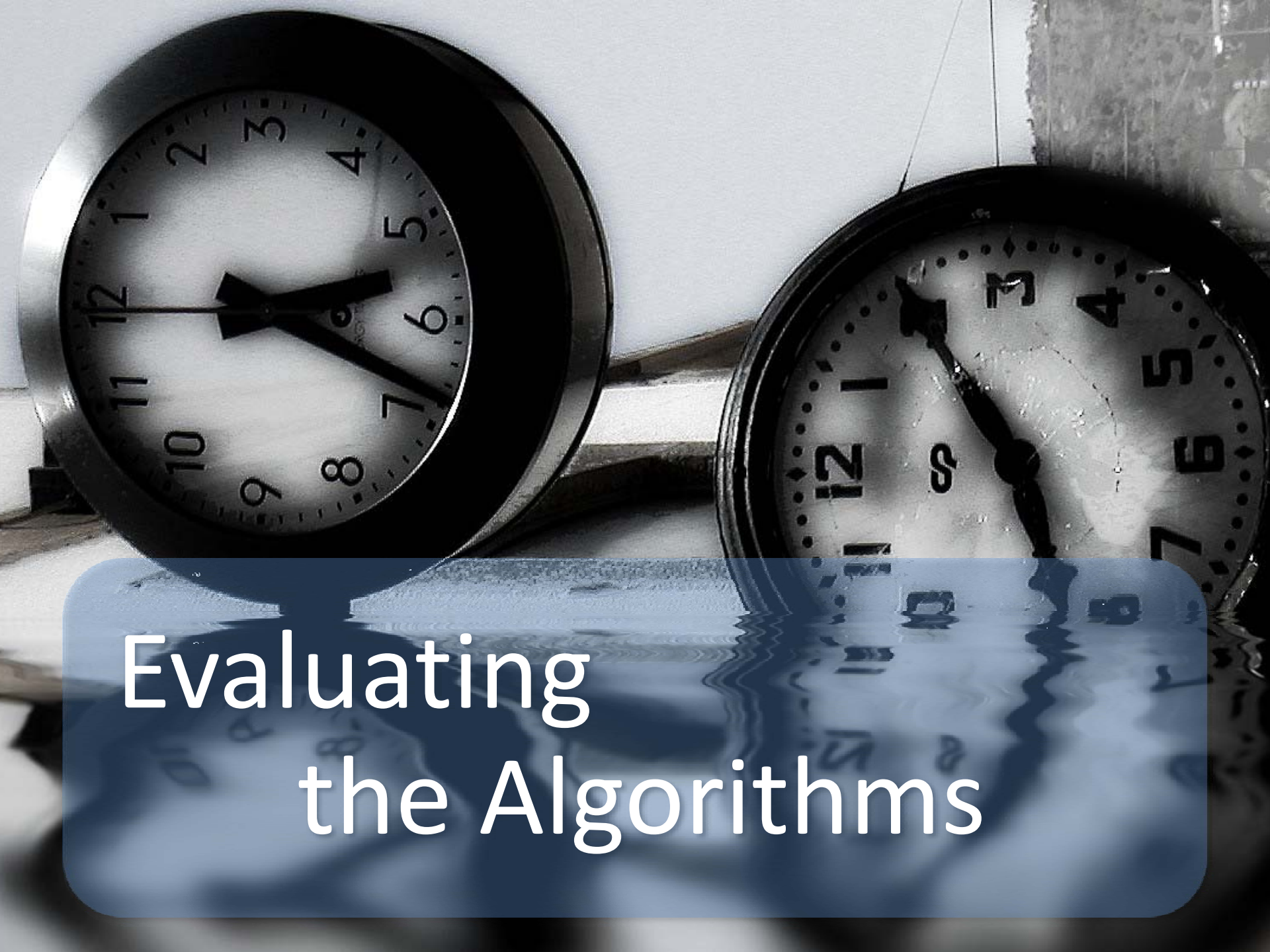
- Applies only to Partition based and Sort-merge based
- Two ways to allocate buffer space:
 - GRACE
 - Dynamically allocate buffer space as requested
 - Hybrid
 - Use static buffer space for the outer relation

Temporal Join Framework

Nested Loop Join



	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				



Evaluating the Algorithms

Experimental Setup

Scaled down instance of a temporal database

- Relation size: 64MB
- Tuple size: 16 bytes
 - Relational attribute size: 8 bytes
 - Time attribute size: 8bytes
- Tuples per relation: 4 million
- Relation lifespan: 1,000,000 time units
- Page size: 1KB

Workload

- Execute equijoins over two relations

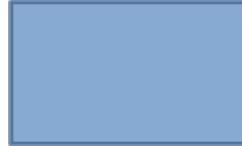
Evaluation

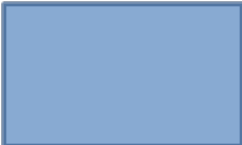
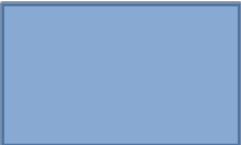
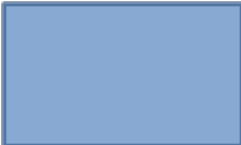
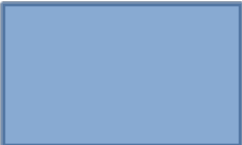
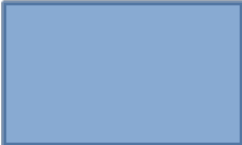
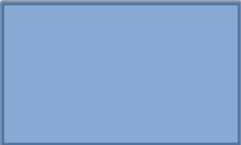
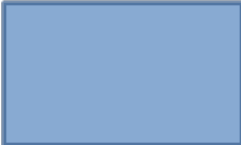
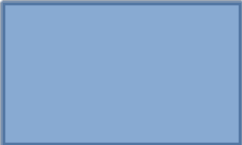
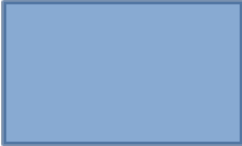
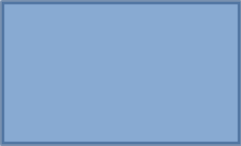
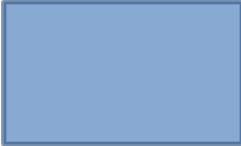
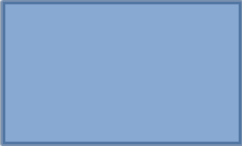
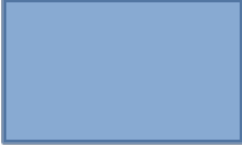
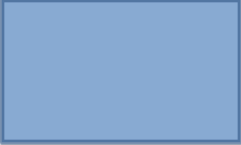
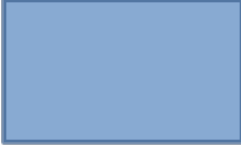
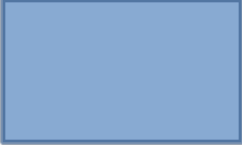
Examined many dimensions

- Relational attributes selectivity
- Large vs small time intervals
- Different relation sizes
- Skew
 - Relational attribute skew
 - Timestamp skew
 - Mixed skew

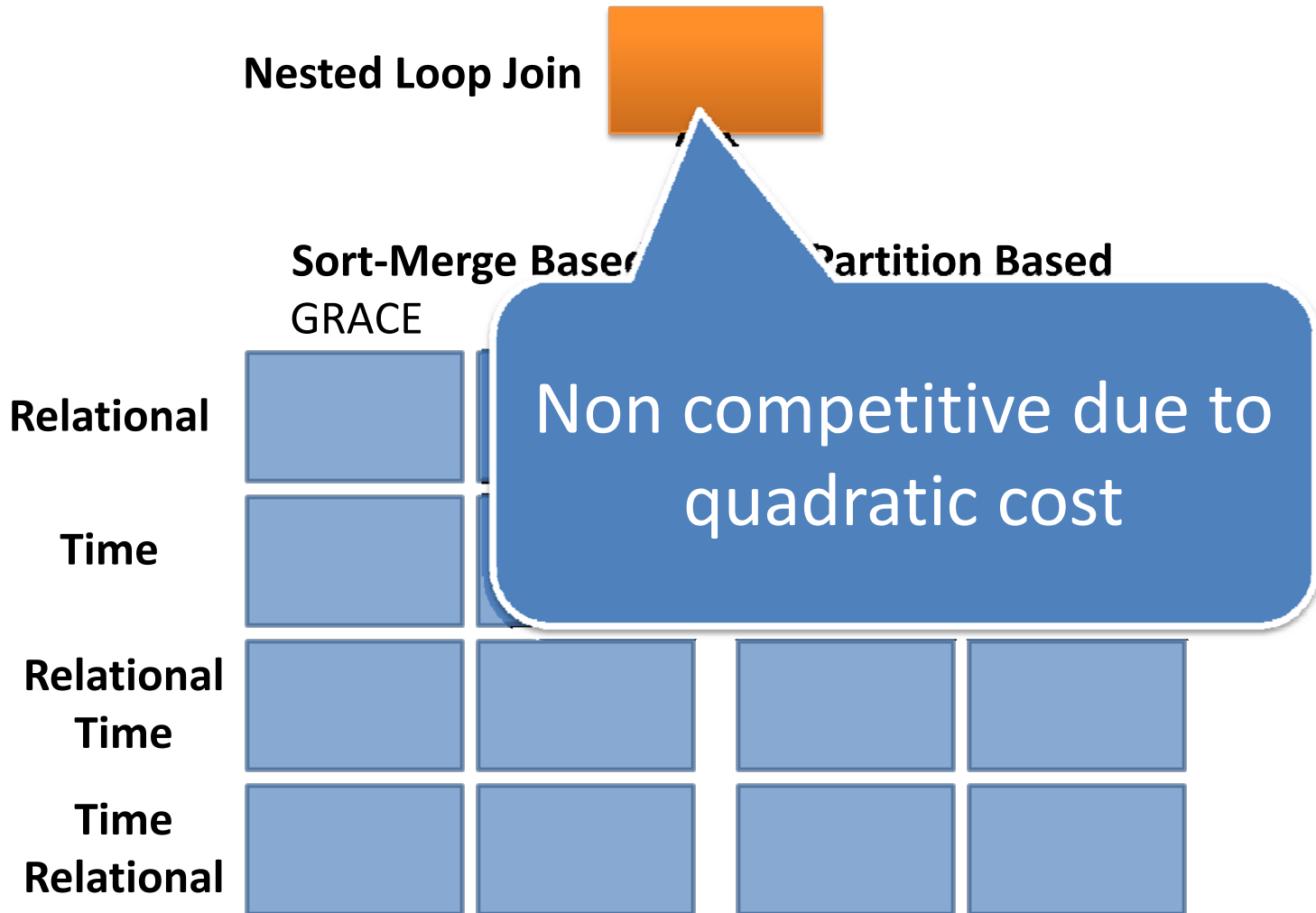
Results

Nested Loop Join



	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				

Results



Results

Very sensitive to
bigger time intervals
and time skew

	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relation				
Time				
Relational Time				
Time Relational				

Performance drops dramatically
as relation size grows

	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				

Very sensitive to relational attribute skew

Nested

	Sort-Merge Base		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				

Performance degrades if relations have different sizes

Nested

	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				

Results

Nested

Sensitive to relational attribute skew

	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				

Results

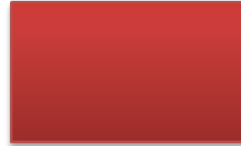
Nested

Sensitive to time attribute skew

	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				

Results

Nested Loop Join



	Sort-Merge Based		Partition Based	
	GRACE	Hybrid	GRACE	Hybrid
Relational				
Time				
Relational Time				
Time Relational				



Conclusions

Conclusions

Defined Temporal Join Operators

- Designed a Framework that allows
 - classification of all existing T-Join algorithms
 - empirical comparison between these classes of algorithms

Evaluation & Comparison of T-Join Algorithms

- Many dimensions explored
 - Easy to distinguish the non-competitive algorithms
- Recommended algorithms for T-Joins:
 - Partition-based, using Hybrid buffer allocation strategy with dual partitioning: either relational/time or time/relational

Own assessment

Evaluation contradicts with motivation

- Relational DBs are optimized for equijoins
- Temporal DBs shall be optimized for inequality predicates

Experimental setup too minimal

- L1 and L2 cache size are not taken into account
 - L1 cache can fit 64 pages!
- No comparison with an existing system
 - Everything run on prototyping software
- No comparisons between existing algorithms
 - Only with “models” of existing algorithms