A Forward Scan based Plane Sweep Algorithm for Parallel Interval Joins

Panagiotis Bouros[1] and Nikos Mamoulis[2]

[1] Aarhus University, Denmark

[2] University of Ioannina, Greece

Interval Joins

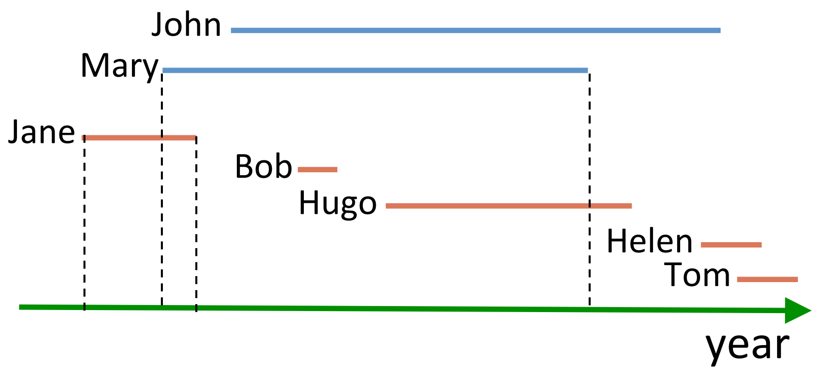
employee	start	end
John	1994	2006
Mary	1992	2002

employee	start	end
Jane	1990	1993
Bob	1995	1996
Hugo	1997	2003
Helen	2005	2007
Tom	2006	2008

Interval Joins

employee	start	end
John	1994	2006
Mary	1992	2002

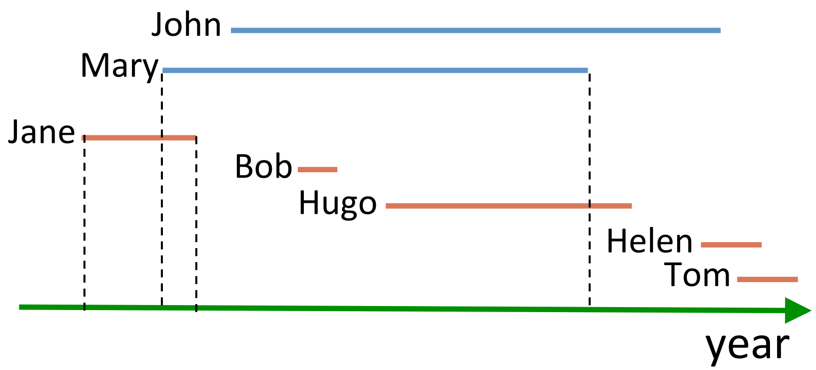
employee	start	end
Jane	1990	1993
Bob	1995	1996
Hugo	1997	2003
Helen	2005	2007
Tom	2006	2008



Interval Joins

employee	start	end
John	1994	2006
Mary	1992	2002

employee	start	end
Jane	1990	1993
Bob	1995	1996
Hugo	1997	2003
Helen	2005	2007
Tom	2006	2008

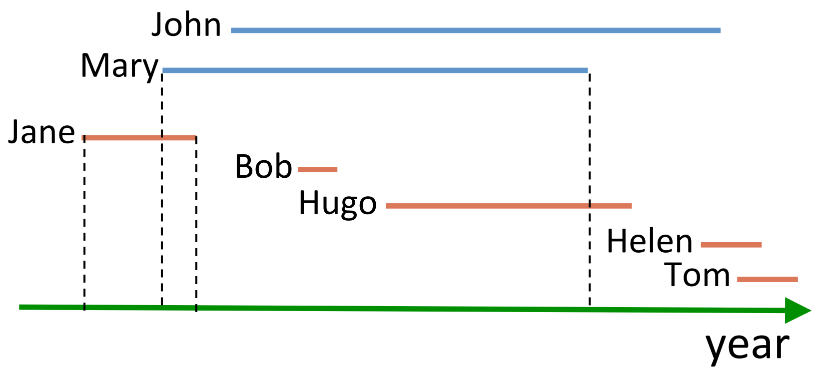


Find all pair of employees whose period of work on D1 and D2 intersect

Interval Joins

employee	start	end
John	1994	2006
Mary	1992	2002

employee	start	end
Jane	1990	1993
Bob	1995	1996
Hugo	1997	2003
Helen	2005	2007
Tom	2006	2008



Find all pair of employees whose period of work on D1 and D2 intersect

- Applications
  - Temporal databases
  - Multidimensional data management
  - Uncertain data management

Our focus

- Efficient evaluation of interval joins
  - Single-threaded processing
    - Simple plane sweep based method
    - Competitive to state-of-the-art
  - Parallel processing
    - Partitioning-based join
    - Share nothing

SINGLE-THREADED PROCESSING

Related work

Nested loops &
Sort-merge join

[Segev and Gunadhi, VLDB'89]
[Gunadhi and Segev, ICDE'91]

Index-based

[Zhang et al., ICDE'02]
[Enderle et al., SIGMOD'04]

Partitioning-
based

[Soo et al., ICDE'94]
[Dignös et al., SIGMOD'14]
[Cafagna and Böhlen, VLDBJ'17]

Plane-sweep
based

[Brinkhoff al., SIGMOD'93]
[Arge et al., VLDB'98]
[Piatov et al., ICDE'16]

Related work

Nested loops &
Sort-merge join

[Segev and Gunadhi, VLDB'89]
[Gunadhi and Segev, ICDE'91]

Index-based

[Zhang et al., ICDE'02]
[Enderle et al., SIGMOD'04]

Partitioning-based

[Soo et al., ICDE'94]
[Dignös et al., SIGMOD'14]
[Cafagna and Böhlen, VLDBJ'17]

Plane-sweep based

[Brinkhoff al., SIGMOD'93]
[Arge et al., VLDB'98]
[Piatov et al., ICDE'16]

Plane sweep methods

- Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]

  - Sweep line stops both on *start* and *end*
  - Backwards scan, Gapless hash map buffers open intervals

Pros

✓ No domain-point comparisons
✓ Tailored to modern hardware
✓ Main memory cache-aware
✓ Fast

Cons

✗ Special structure needed

Plane sweep methods

- Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]

  - Sweep line stops both on *start* and *end*
  - Backwards scan, Gapless hash map buffers open intervals

Pros

✓ No domain-point comparisons
✓ Tailored to modern hardware
✓ Main memory cache-aware
✓ Fast

Cons

✗ Special structure needed

- Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]

  - Sweep lines stops only on *start*

Pros

✓ Simple
✓ No special structure needed

Cons

✗ $|R| + |S| + |R \bowtie S|$ comparisons in total

Plane sweep methods

- Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]

  - Sweep line stops both on *start* and *end*
  - Backwards scan, Gapless hash map buffers open intervals

Pros

✓ No domain-point comparisons
✓ Tailored to modern hardware
✓ Main memory cache-aware
✓ Fast

Cons

✗ Special structure needed

- Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]

  - Sweep lines stops only on *start*

Pros

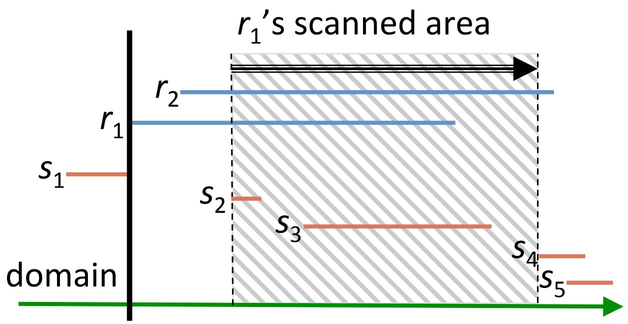
✓ Simple
✓ No special structure needed

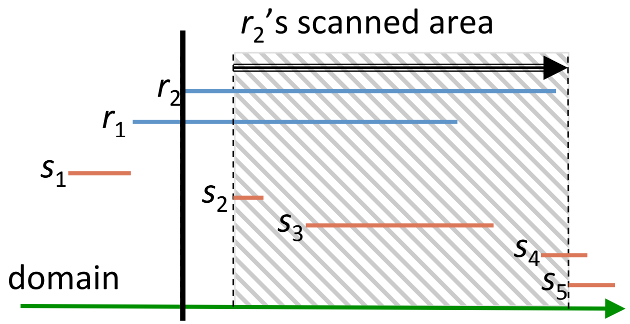
Cons

✗ $\boxed{|R| + |S| + |R \bowtie S|}$ comparisons in total

Optimizing FS

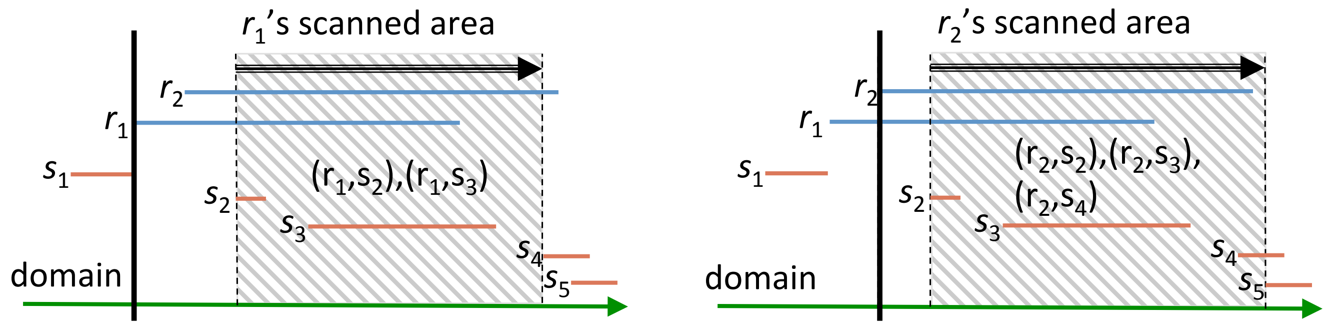
FS



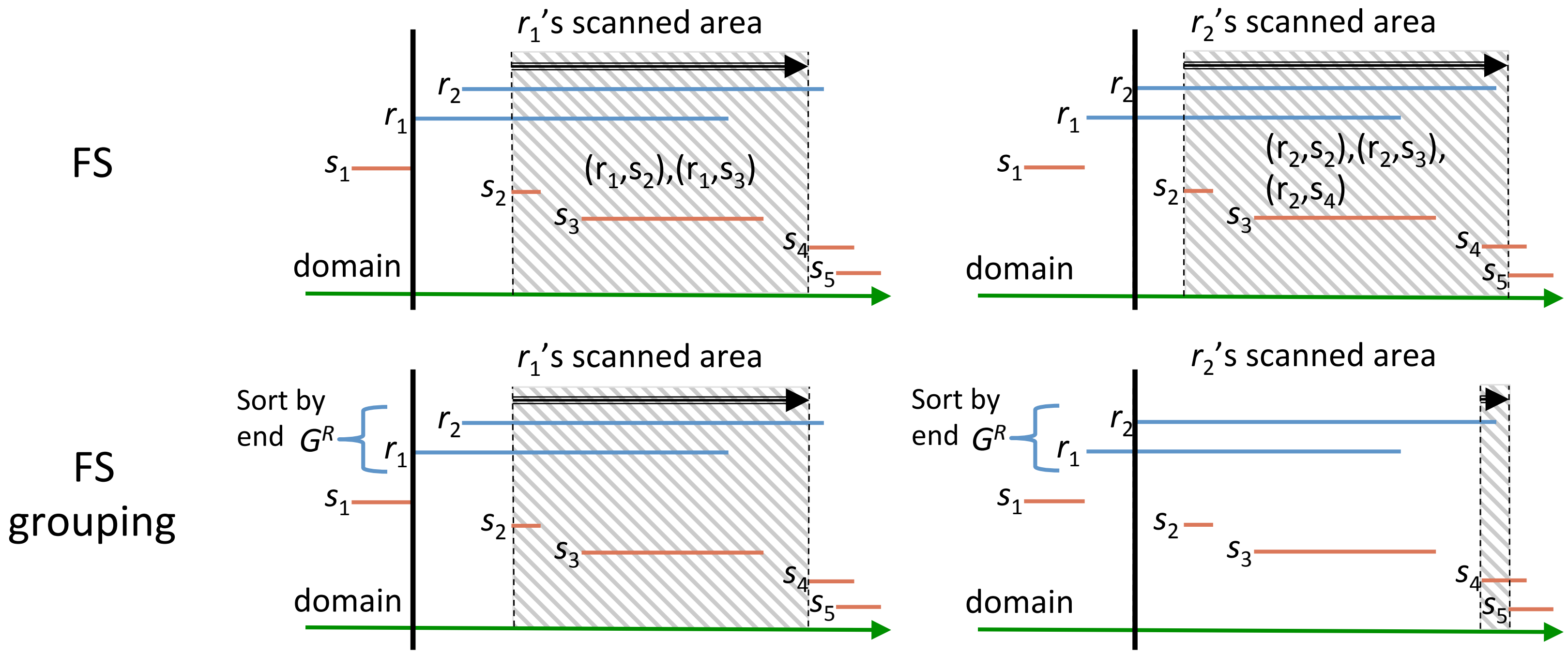


Optimizing FS

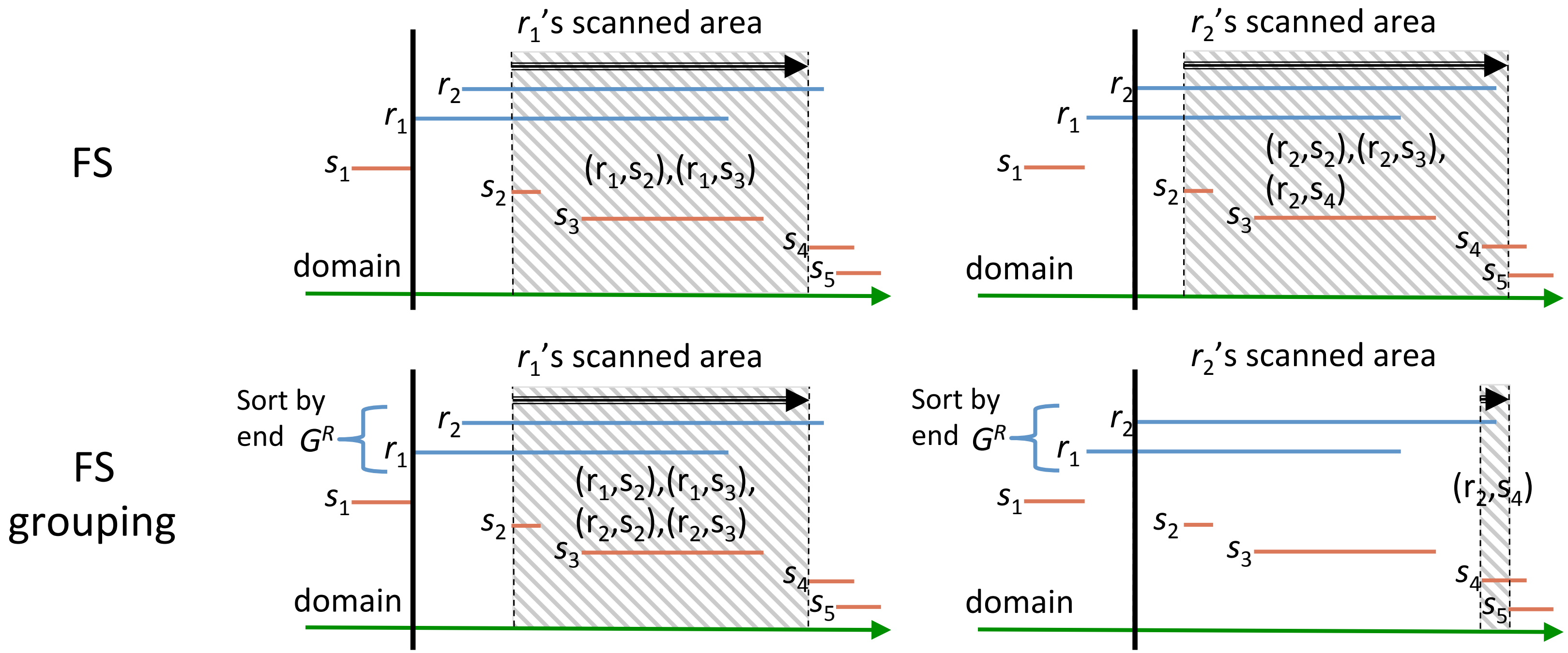
FS



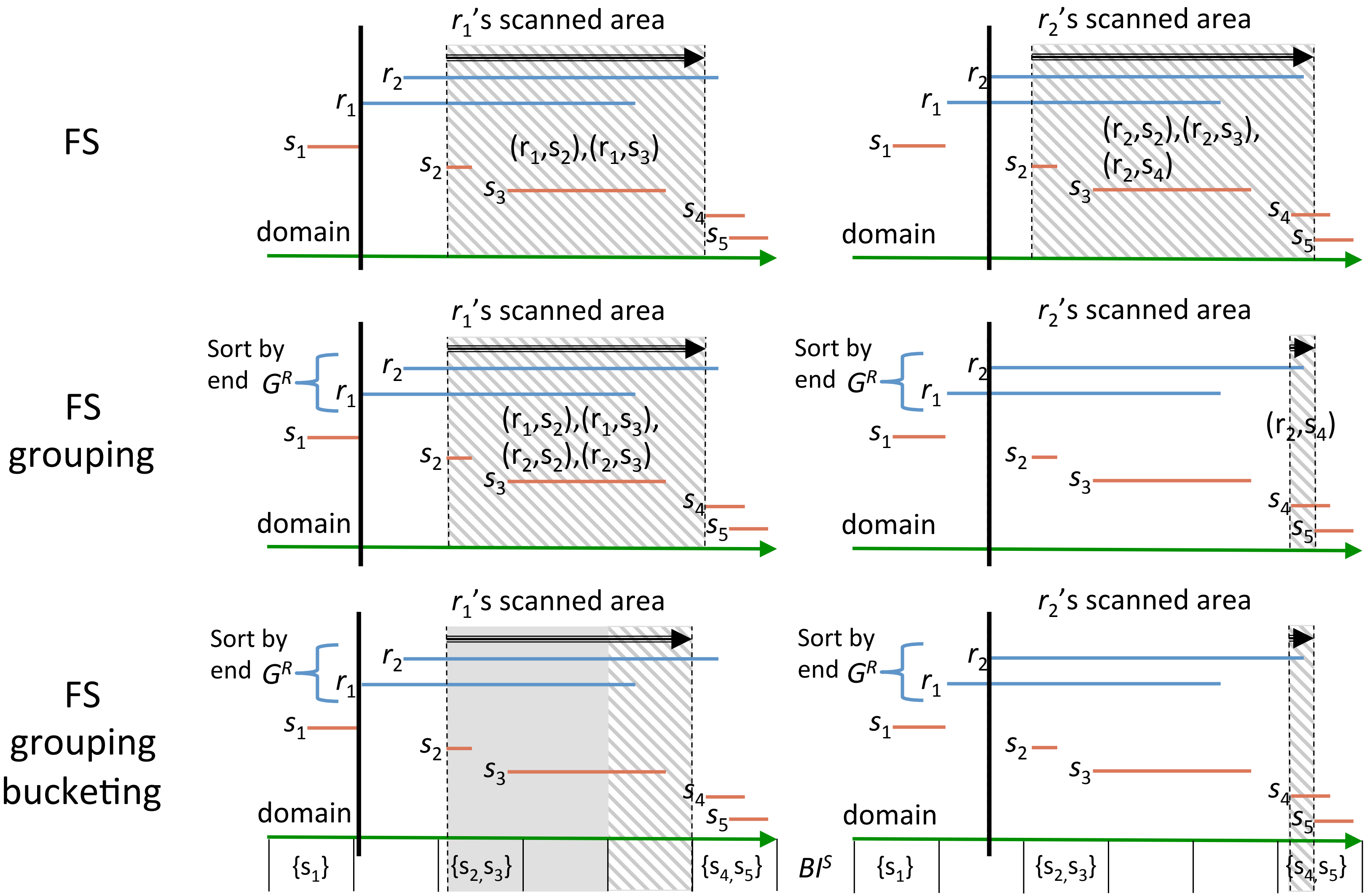
Optimizing FS



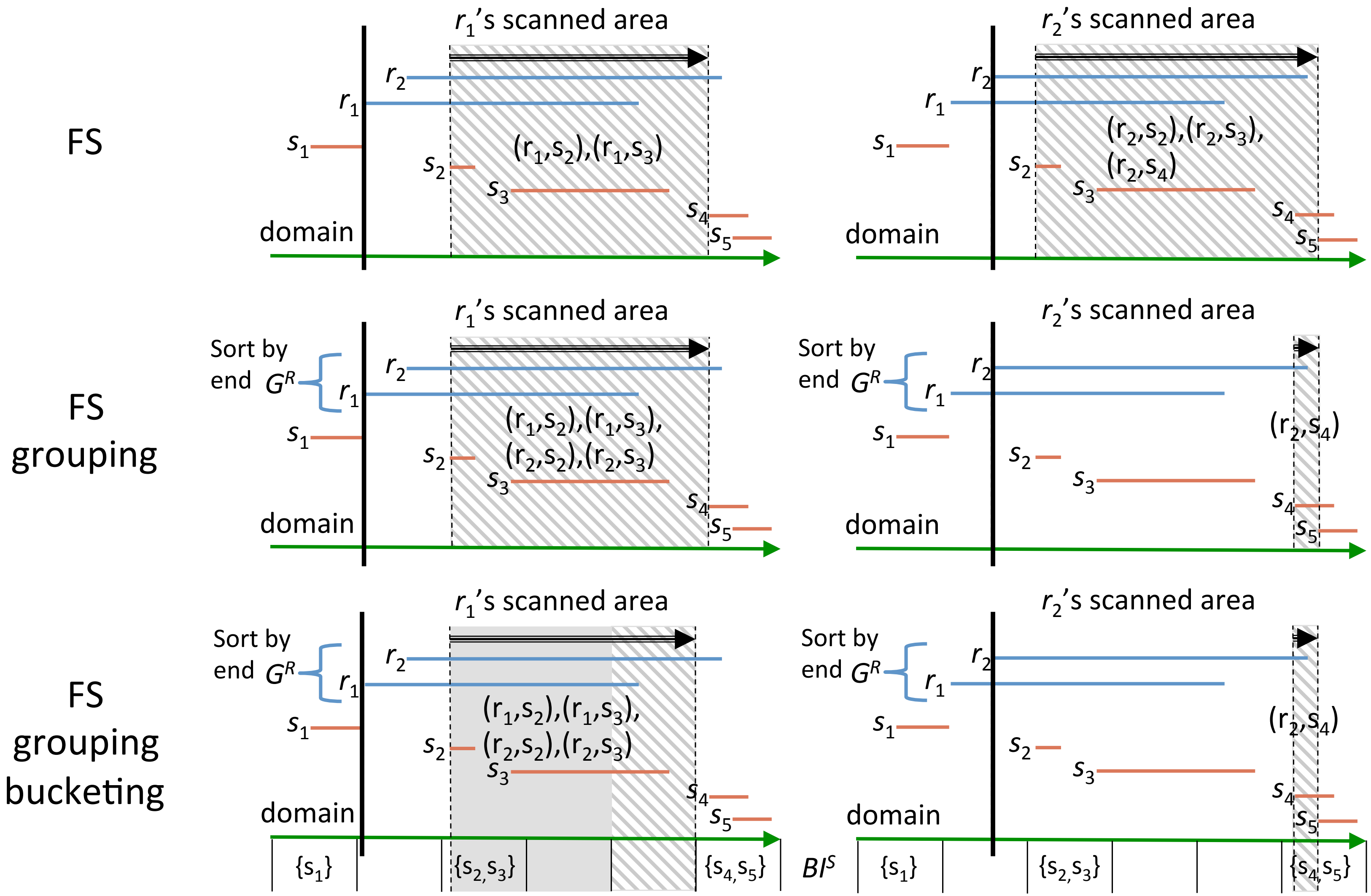
Optimizing FS



Optimizing FS



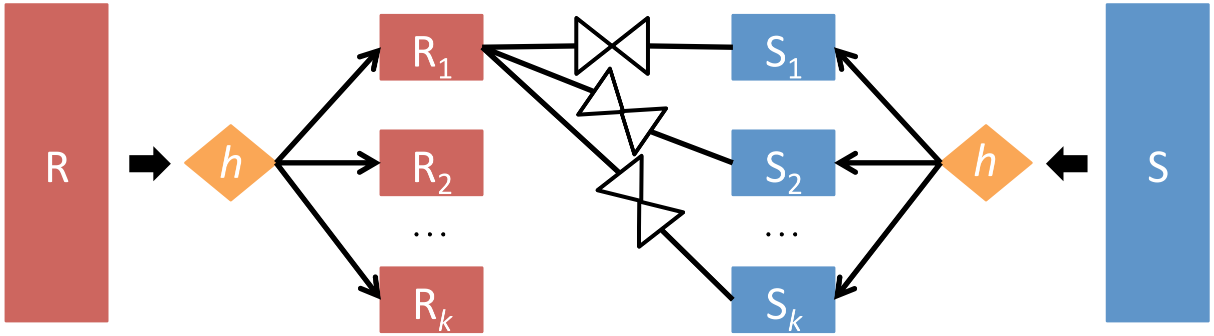
Optimizing FS



PARALLEL PROCESSING

Hash-based partitioning

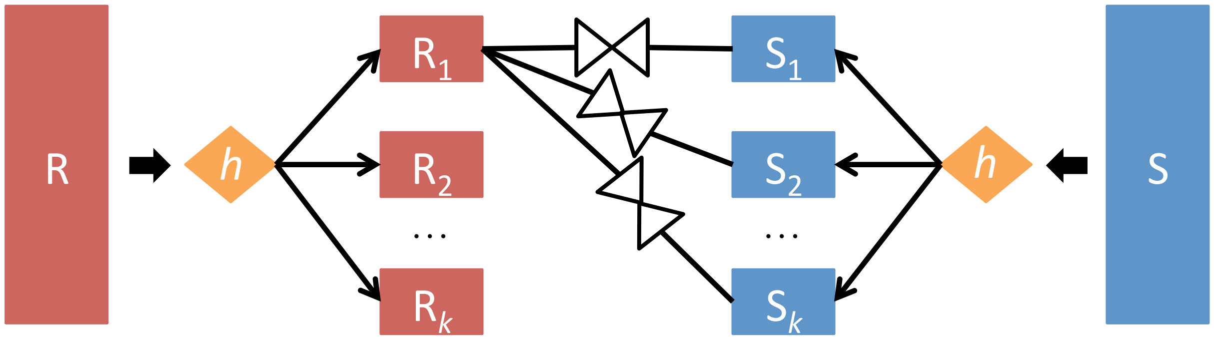
[Piatov et al., ICDE’16]



- Idea
  - Randomly split each input into *k* partitions using hash function *h*
  - Evaluate k^2 independent partition joins

Hash-based partitioning

[Piatov et al., ICDE'16]



- Idea
  - Randomly split each input into *k* partitions using hash function *h*
  - Evaluate k^2 independent partition joins

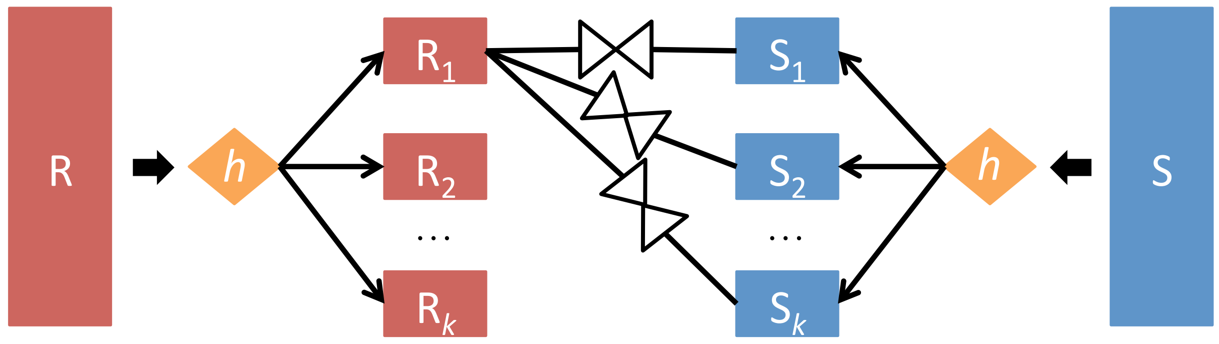
Pros

✓ Simple

✓ Load balancing

Hash-based partitioning

[Piatov et al., ICDE'16]



- Idea
  - Randomly split each input into k partitions using hash function h
  - Evaluate k^2 independent partition joins

Pros

✓ Simple

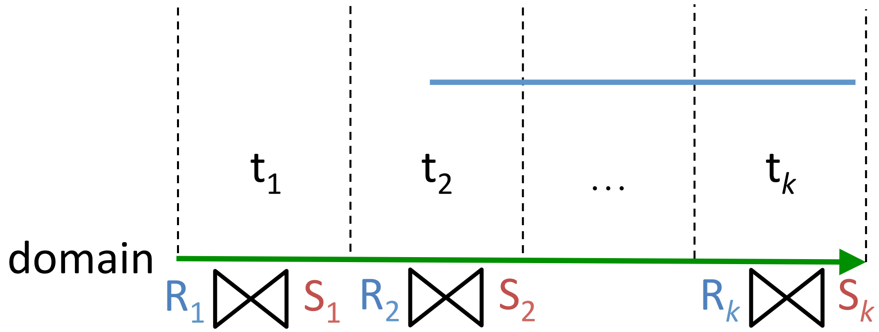
✓ Load balancing

Cons

✗ Domain-point comparisons rise
  - $2 \cdot k \cdot (|R|+|S|)$ for EBI/LEBI, $k \cdot (|R|+|S|)$ for FS

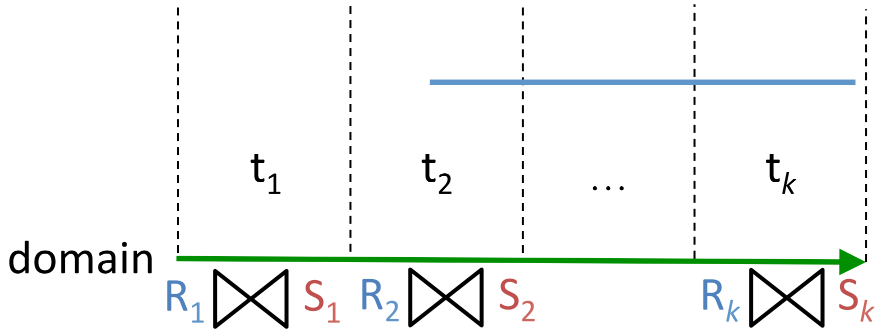
✗ Degree of parallelism
  - n CPU cores → $k = \sqrt{n}$ partitions

Domain-based partitioning



- Idea
  - Split domain into *k* tiles
  - Replicate intervals
  - Evaluate *k* independent partition joins

Domain-based partitioning



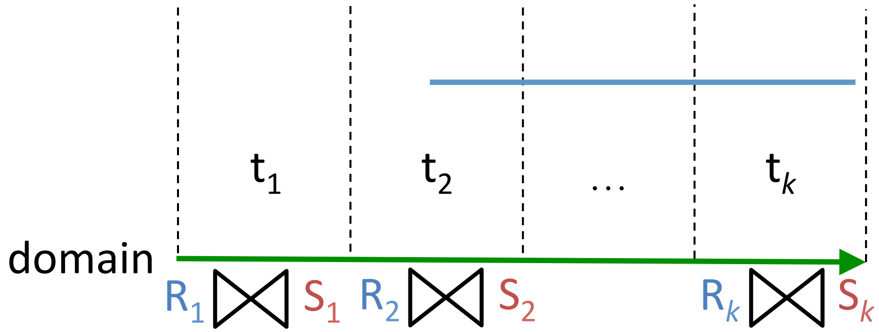
- Idea
  - Split domain into *k* tiles
  - Replicate intervals
  - Evaluate *k* independent partition joins

Pros

✓ Degree of parallelism
  - *n* CPU cores → *k* = *n* partitions

✓ Automatic duplicate elimination

Domain-based partitioning



- Idea
  - Split domain into *k* tiles
  - Replicate intervals
  - Evaluate *k* independent partition joins

Pros

✓ Degree of parallelism
  - *n* CPU cores → *k* = *n* partitions

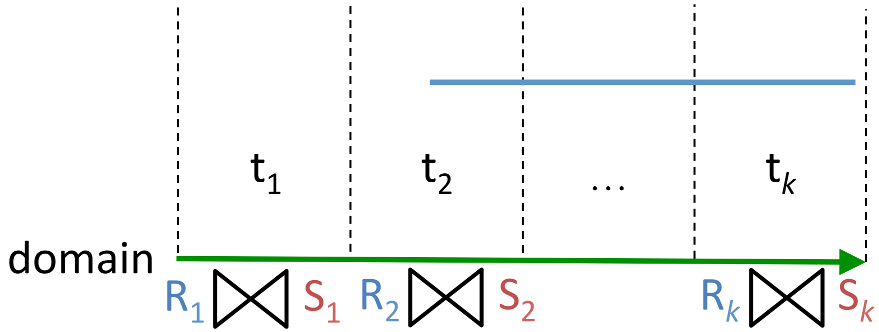
✓ Automatic duplicate elimination

Cons

✗ Replication

✗ Load balancing

Domain-based partitioning



- Idea
  - Split domain into *k* tiles
  - Replicate intervals
  - Evaluate *k* independent partition joins

Pros

✓ Degree of parallelism
  - *n* CPU cores → *k* = *n* partitions

✓ Automatic duplicate elimination

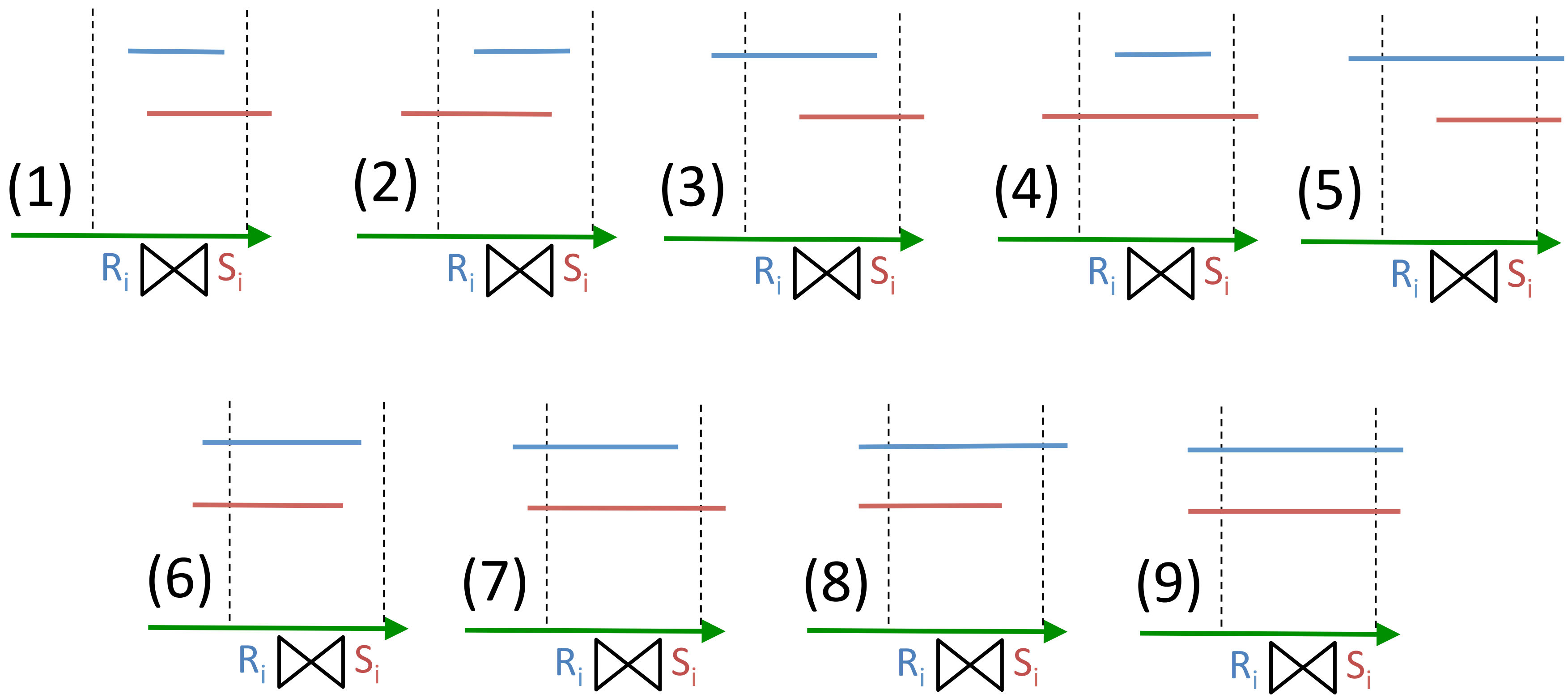
Cons

✗ Replication

✗ Load balancing

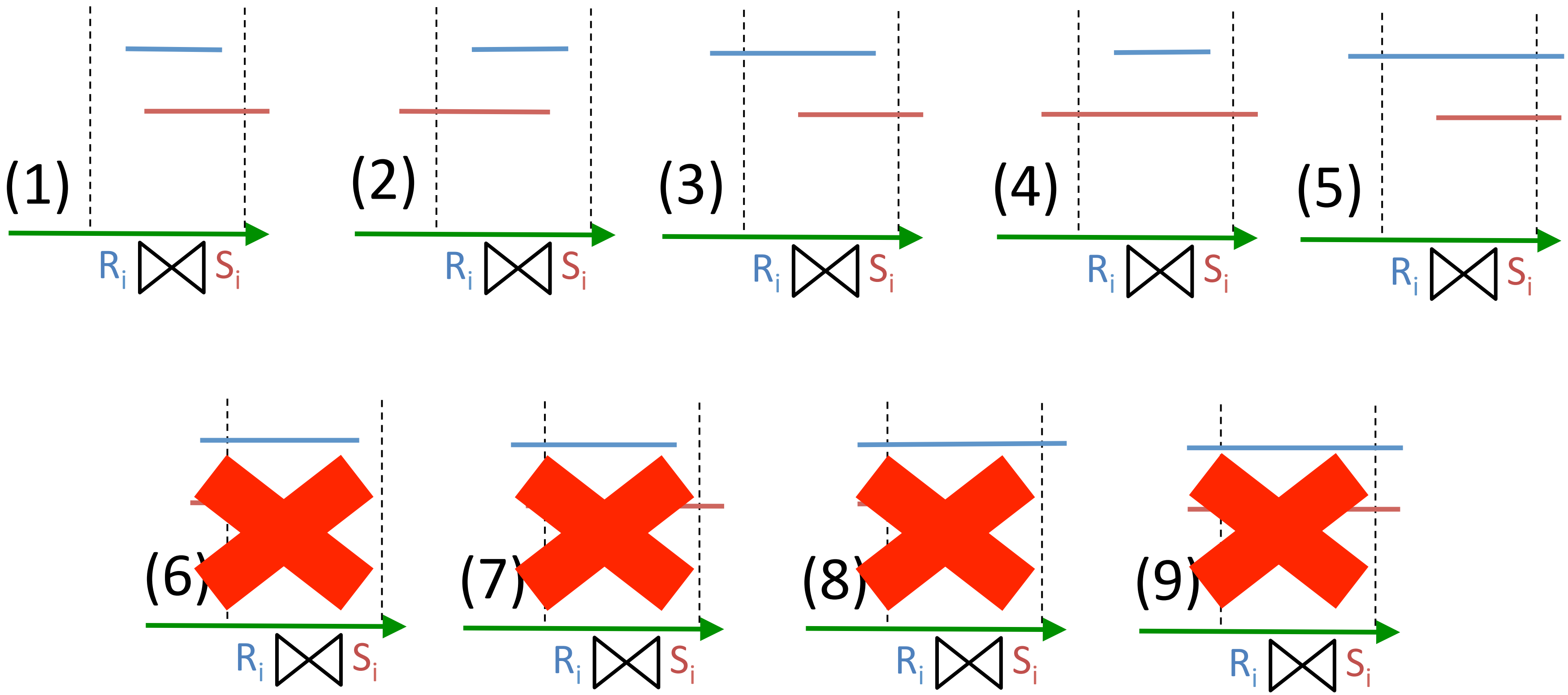
Mini-joins break down

- 3 types of intervals → 9 types of mini-tasks



Mini-joins break down

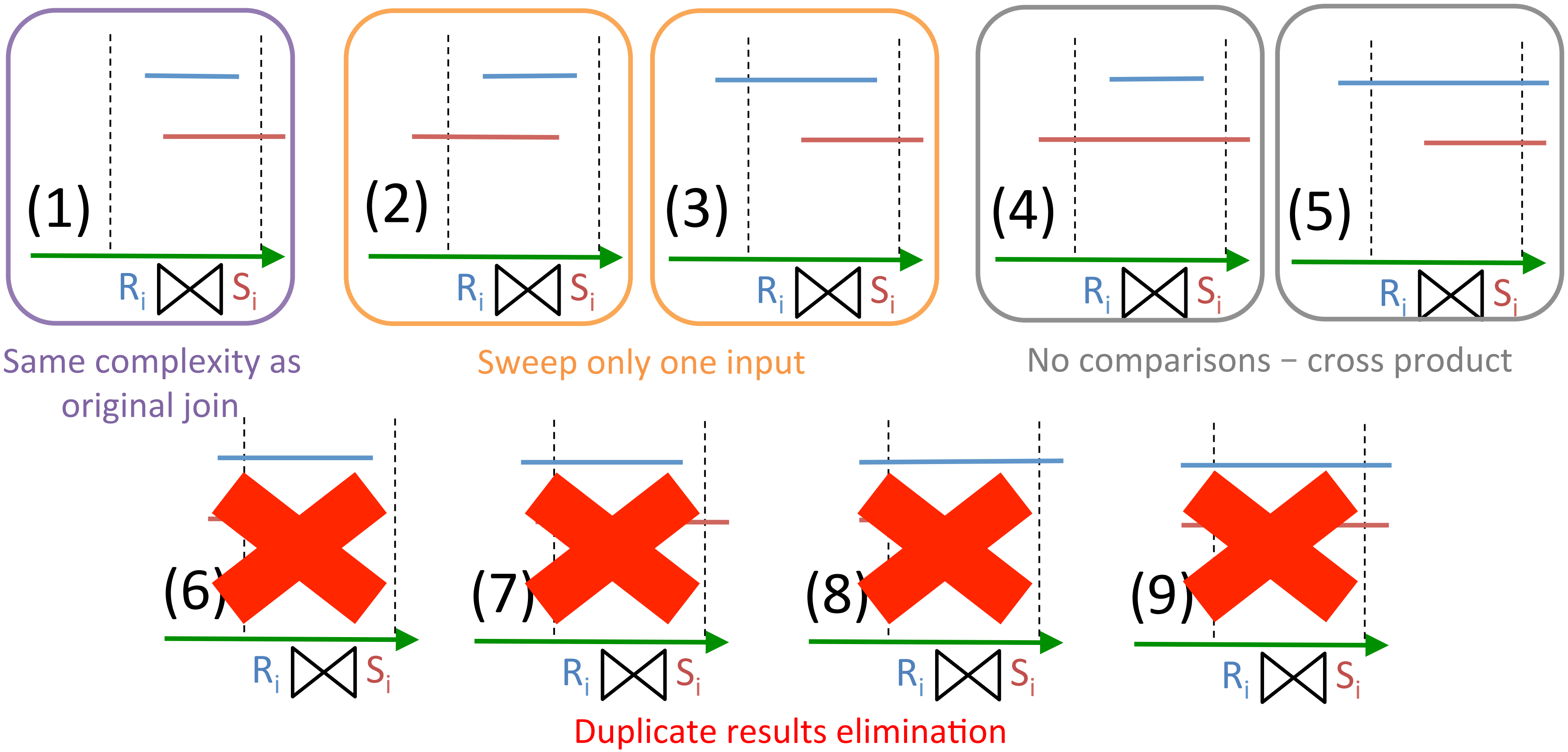
- 3 types of intervals → 9 types of mini-tasks



Duplicate results elimination

Mini-joins break down

- 3 types of intervals → 9 types of mini-tasks

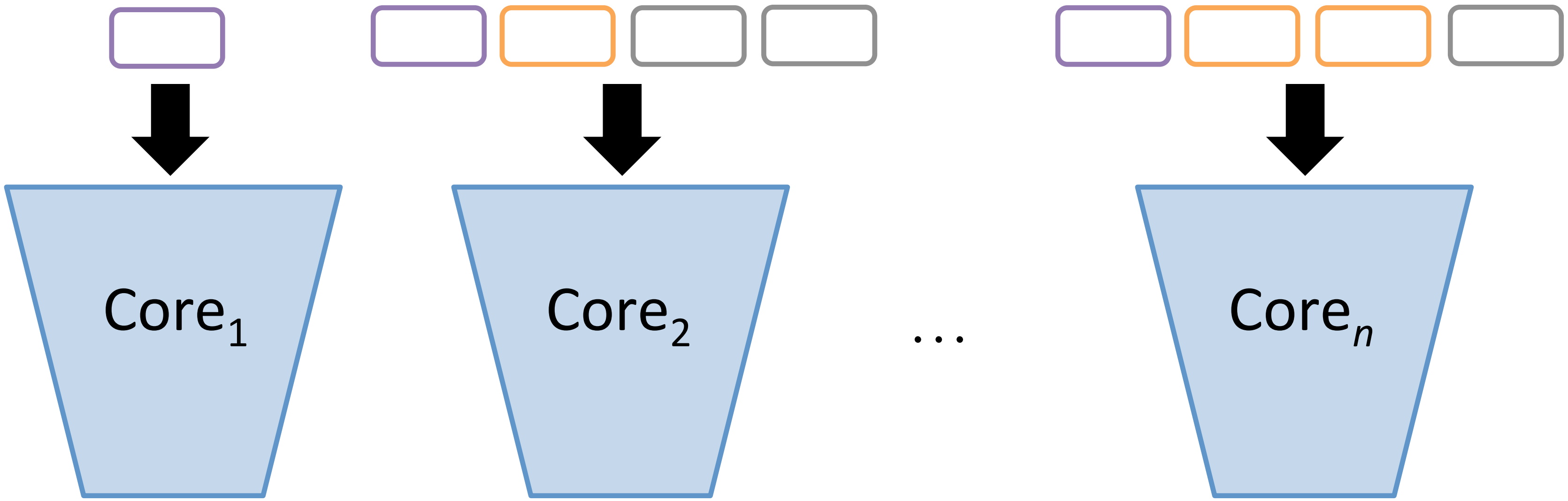


Greedy scheduling

- Idea
  - Distribute $1 + 5 \cdot (k\text{-}1)$ mini-joins to different cores
  - Evenly distribute load → minimize max load
  - NP-hard problem
    - Greedy approximation algorithm

Greedy scheduling

- Idea
  - Distribute $1 + 5 \cdot (k\text{-}1)$ mini-joins to different cores
  - Evenly distribute load → minimize max load
  - NP-hard problem
    - Greedy approximation algorithm

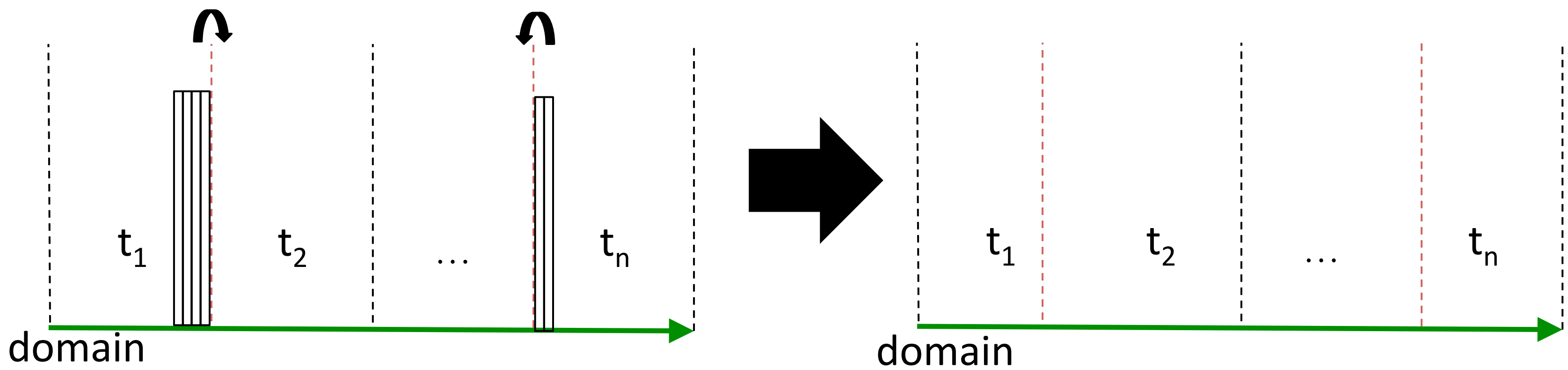


Adaptive partitioning

- Idea
  - Create an initial uniform partitioning
  - Employ a very fine tiling – granules
  - Move load between neighboring tiles
    - Move granules between neighboring tiles
    - Reposition borders of tiles

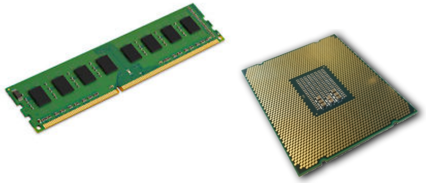
Adaptive partitioning

- Idea
  - Create an initial uniform partitioning
  - Employ a very fine tiling – granules
  - Move load between neighboring tiles
    - Move granules between neighboring tiles
    - Reposition borders of tiles

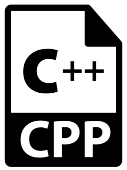


EXPERIMENTAL ANALYSIS

Setup

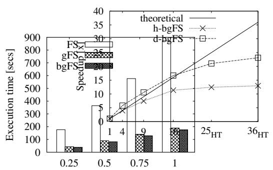


- Hardware
  - dual 10-core Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10 GHz with 128 GBs of RAM
  - Hyper-threading enabled, up to 40 threads



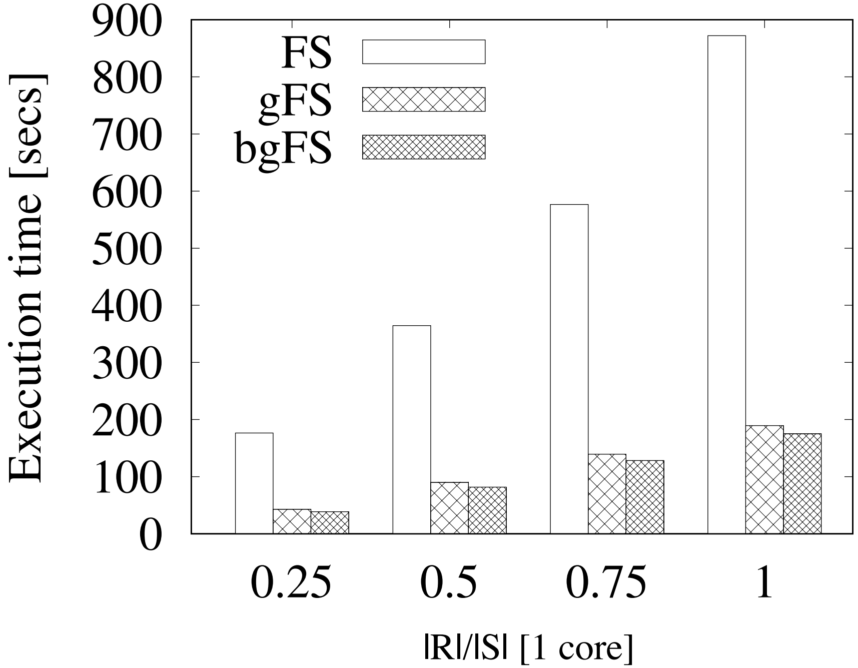
- Software
  - Workload [ICDE'16] → XOR of *start*
  - Loop unrolling forced, OpenMP for multi-threading

- Datasets
  - WEBKIT git repo, interval = period of time file unchanged
  - BOOKS Aarhus libraries, interval = period of time book lent
  - Synthetic
    - Interval duration follows exponential distribution, uniformly distributed *start* plus peaks

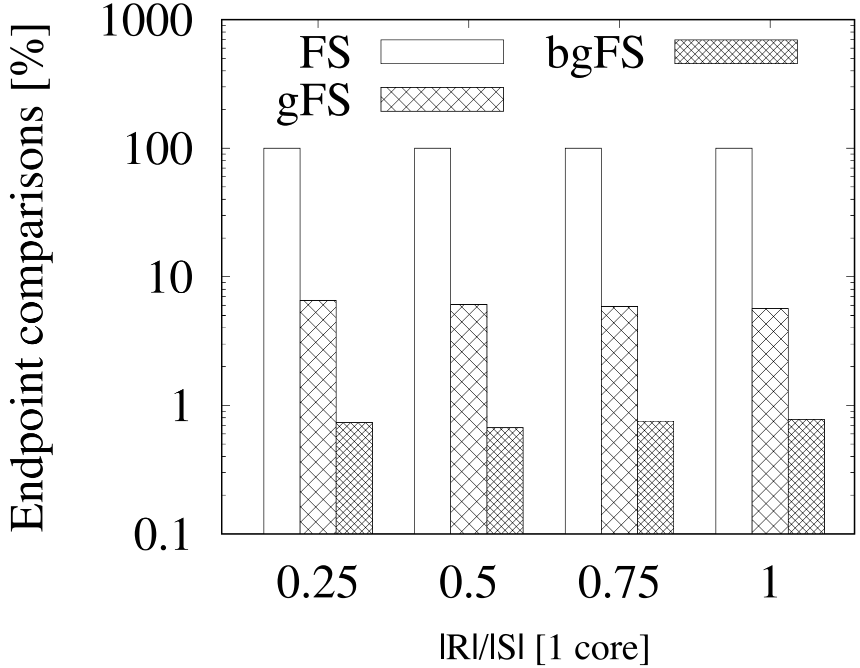


- Experiments
  - Execution time, # comparisons, memory footprint
  - Both self joins and non-self joins
  - Vary |R|/|S|, # cores (threads)

Optimizing FS

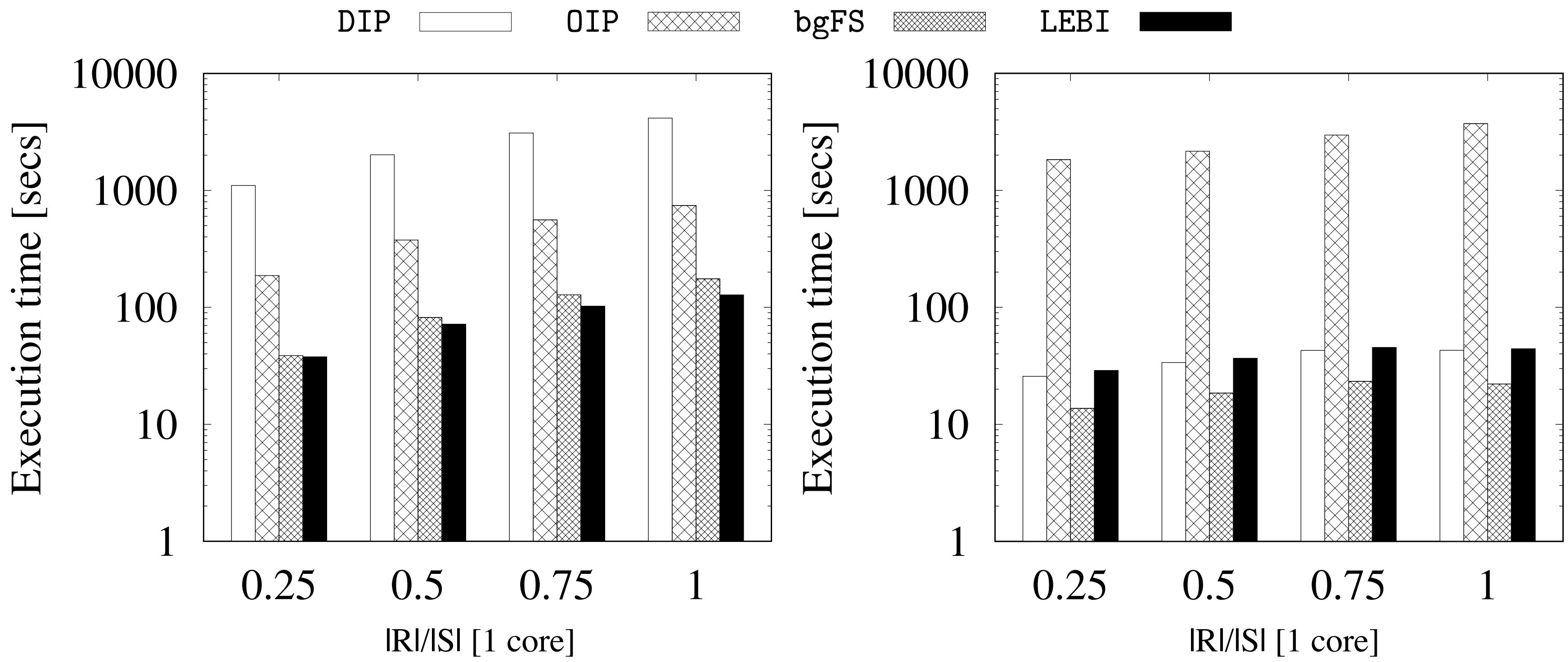


WEBKIT



WEBKIT

Single-threaded processing



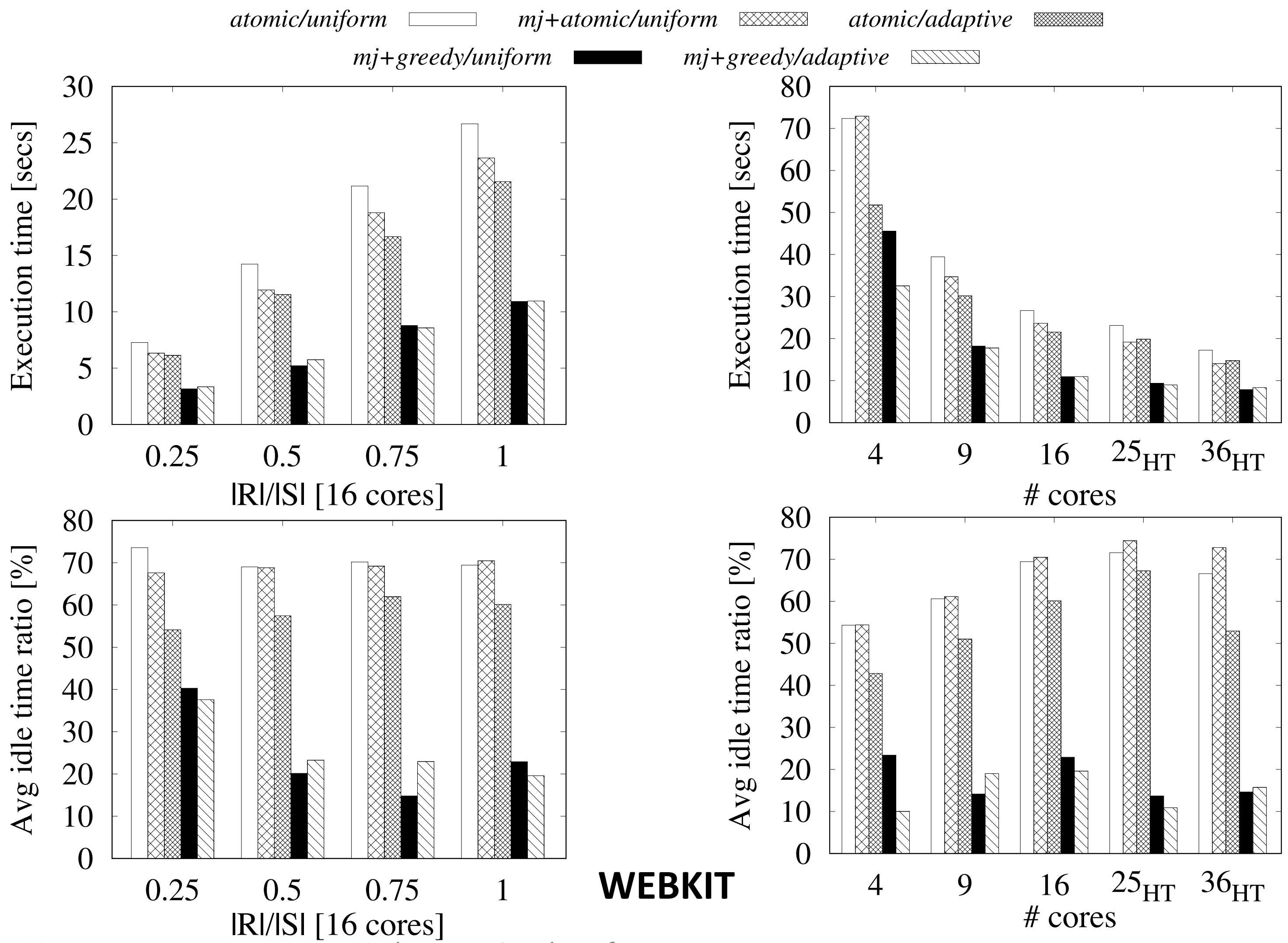
WEBKIT

GREEND

[Cafagna and Bölhen, VLDBJ’17]

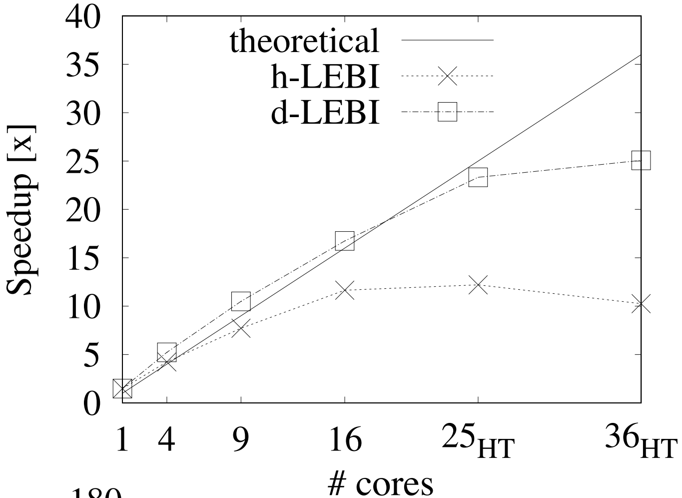
DIP: [Cafagna and Bölhen, VLDBJ’17]
OIP: [Dignös et al., SIGMOD’14]

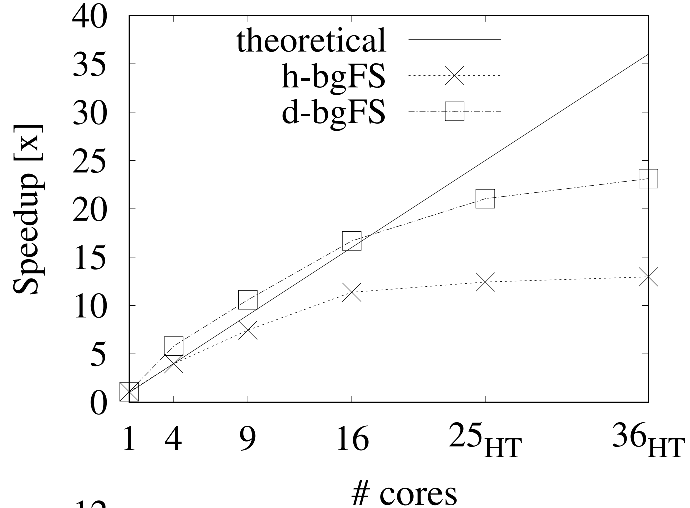
Optimizing Domain-based Paradigm

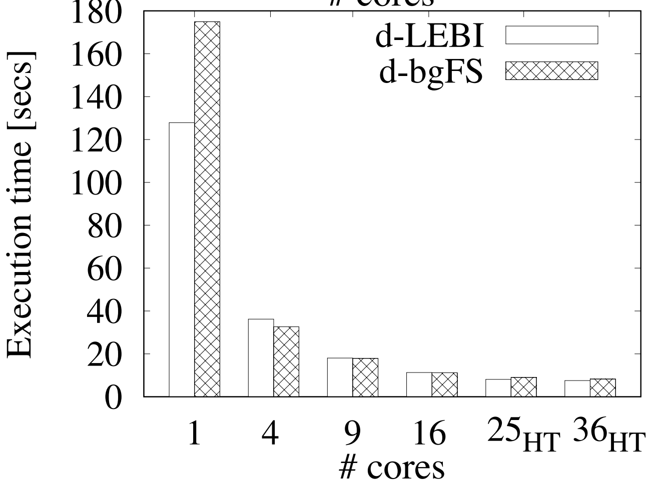


WEBKIT

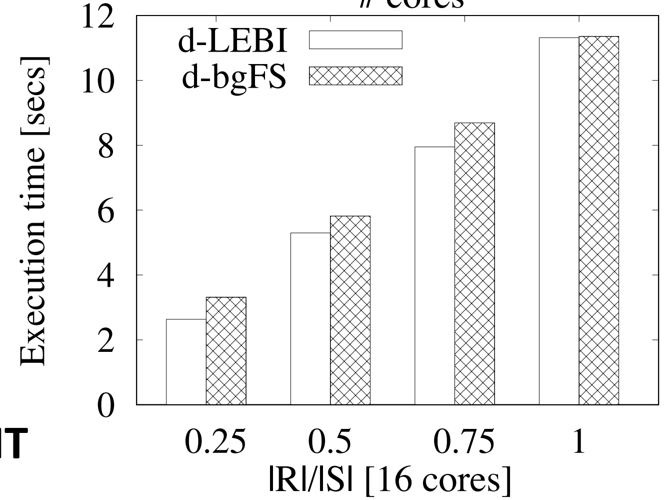
Parallel processing







WEBKIT



To sum up

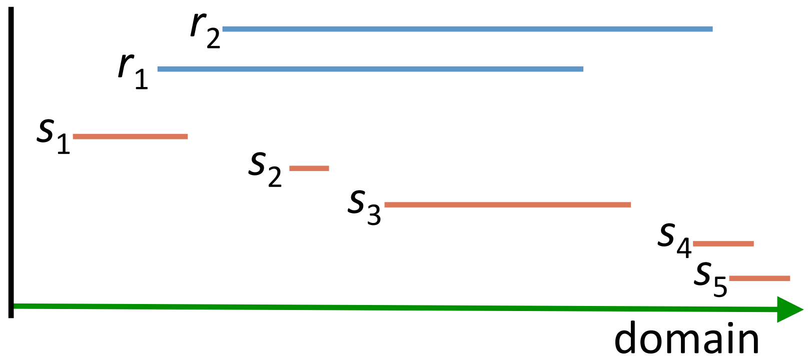
- Contributions
  - Efficient evaluation of interval joins
  - Single-threaded processing
    - Optimized bgFS, competitive to state-of-the-art EBI/LEBI
    - Lower memory footprint
  - Parallel processing
    - Novel domain-based partitioning paradigm
    - Higher speedup
- Future work
  - Other types of temporal joins
  - Other types of temporal operators
  - Parallel processing
    - Data-level parallelism, share data between threads

Questions ?

EXTRAS

Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]



Endpoint indices

$EI^R = \{r_1.\mathit{start}, r_2.\mathit{start}, r_1.\mathit{end}, r_2.\mathit{end}\}$

$EI^S = \{s_1.\mathit{start}, s_1.\mathit{end}, s_2.\mathit{start}, s_2.\mathit{end}, s_3.\mathit{start}, s_3.\mathit{end}, s_4.\mathit{start}, s_5.\mathit{start}, s_4.\mathit{end}, s_5.\mathit{end}\}$

Active sets

$A^R = \{\}$

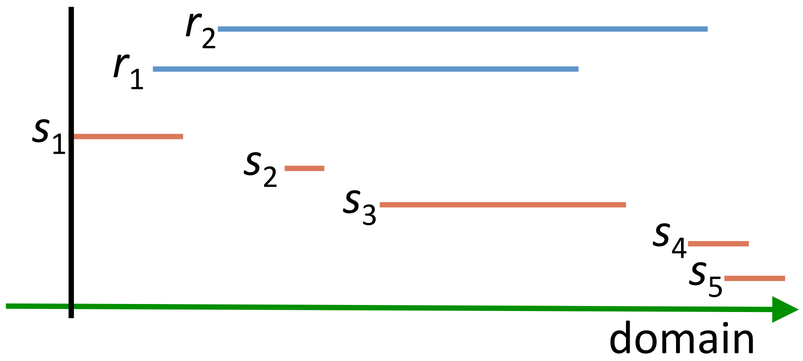
$A^S = \{\}$

Result

$\{\}$

Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]



Endpoint indices

$EI^R = \{r_1.\textit{start}, r_2.\textit{start}, r_1.\textit{end}, r_2.\textit{end}\}$

$EI^S = \{s_1.\textit{start}, s_1.\textit{end}, s_2.\textit{start}, s_2.\textit{end}, s_3.\textit{start}, s_3.\textit{end}, s_4.\textit{start}, s_5.\textit{start}, s_4.\textit{end}, s_5.\textit{end}\}$

Active sets

$A^R = \{\}$

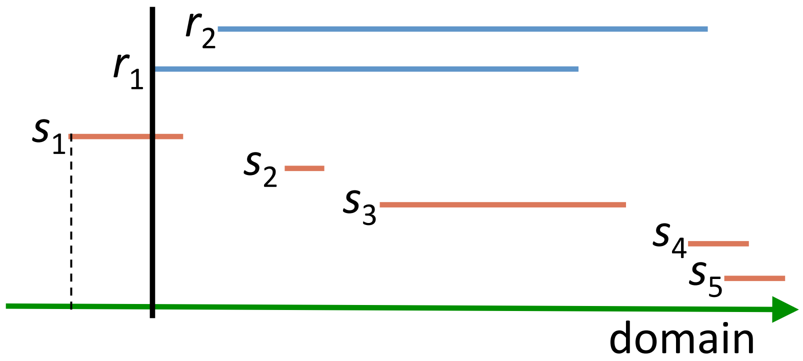
$A^S = \{s_1\}$

Result

$\{\}$

Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]



Endpoint indices

$EI^R = \{r_1.start, r_2.start, r_1.end, r_2.end\}$

$EI^S = \{s_1.start, s_1.end, s_2.start, s_2.end, s_3.start, s_3.end, s_4.start, s_5.start, s_4.end, s_5.end\}$

Active sets

$A^R = \{\}$

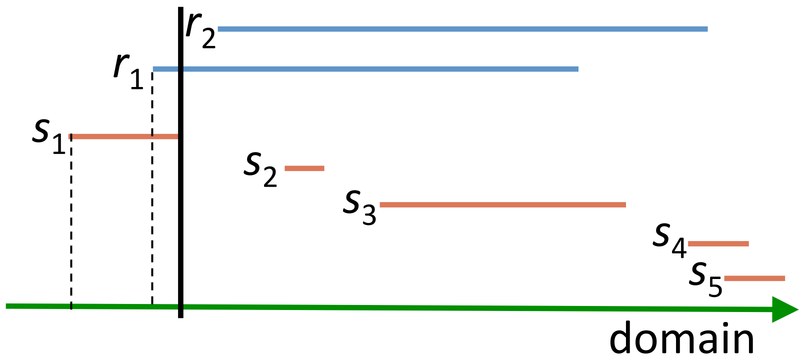
$A^S = \{s_1\}$

Result

$\{(r_1,s_1)\}$

Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]



Endpoint indices

$EI^R = \{r_1.start, r_2.start, r_1.end, r_2.end\}$

$EI^S = \{s_1.start, s_1.end, s_2.start, s_2.end, s_3.start, s_3.end, s_4.start, s_5.start, s_4.end, s_5.end\}$

Active sets

$A^R = \{\}$

$A^S = \{\cancel{s_1}\}$

Result

$\{(r_1,s_1)\}$

Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE'16]

Pros

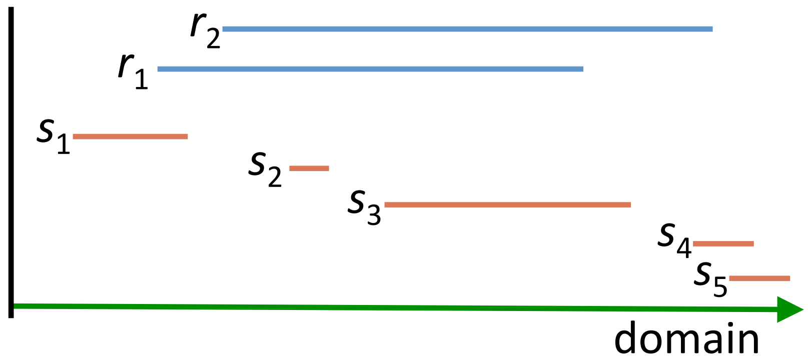
- ✓ No domain-point comparisons when producing results
- ✓ Tailored to modern hardware
- ✓ Main memory cache-aware
- ✓ Fast

Cons

- ✗ Special data structure needed for active sets, support for efficient updates and scans

Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]



Sorted inputs

R = {r_1, r_2}

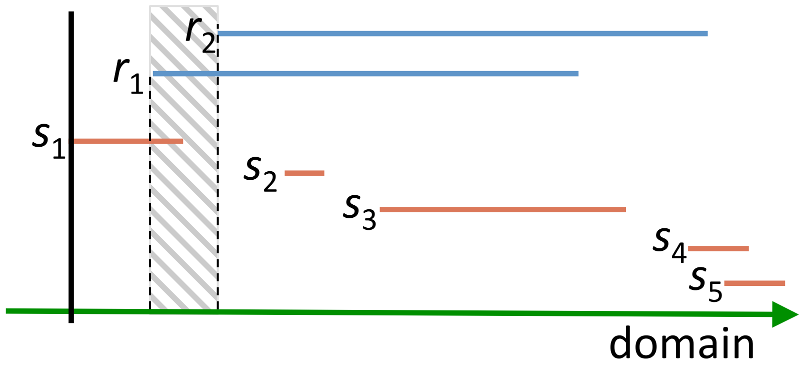
S = {s_1, s_2, s_3, s_4, s_5}

Result

{}

Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]



Sorted inputs

R = $\{r_1, r_2\}$

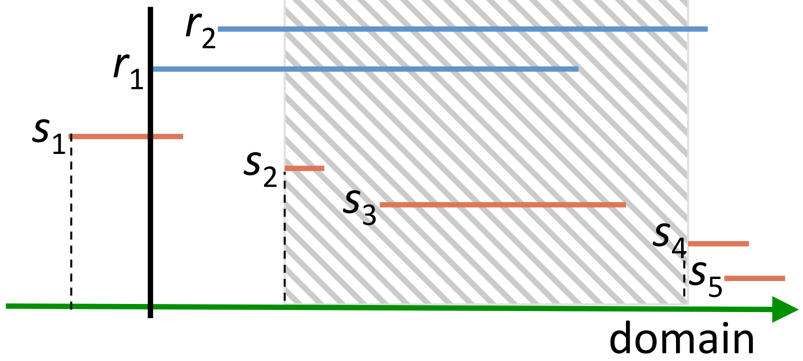
S = $\{s_1, s_2, s_3, s_4, s_5\}$

Result

$\{(r_1,s_1)\}$

Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]



Sorted inputs

R = {r_1, r_2}

S = {s_1, s_2, s_3, s_4, s_5}

Result

{(r_1,s_1), (r_1,s_2), (r_1,s_3)}

Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]

Pros

✓ Simple

✓ No special structure needed

Cons

✗ Each join result requires a domain-point comparison, $|R| + |S| + |R \bowtie S|$ comparisons in total

Forward Scan based (FS)

[Brinkhoff et al., SIGMOD'93]

Pros

✓ Simple

✓ No special structure needed

Cons

✗ Each join result requires a domain-point comparison,
$\boxed{|R| + |S| + |R \bowtie S|}$ comparisons in total