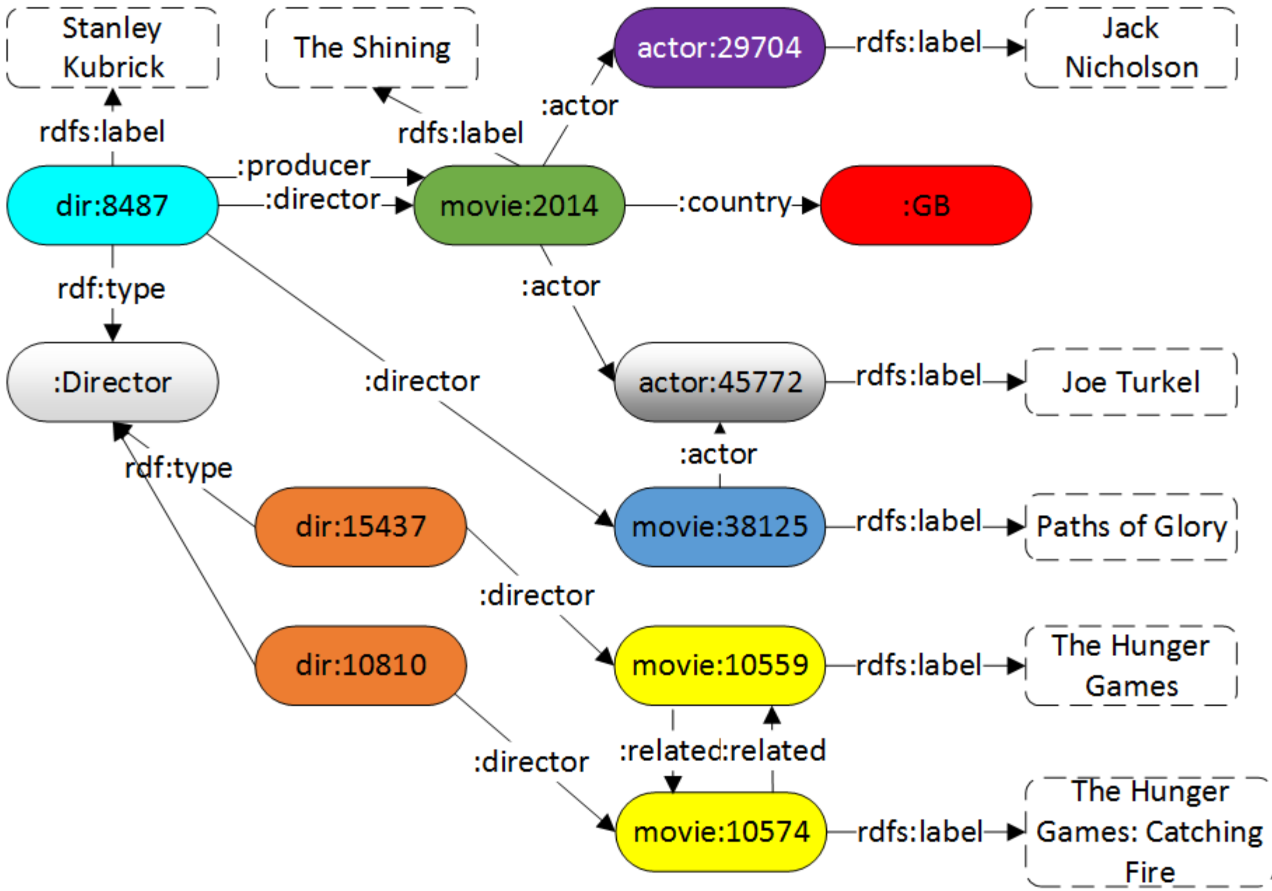
MIE1512 Data Analytics

Graph Analytics

Big Graphs

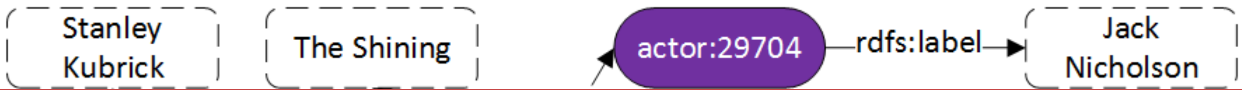
Natural to model relationships as **graphs**



UofT

Big Graphs

Natural to model relationships as **graphs**

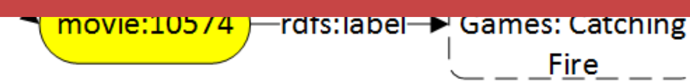


It is a challenge to **query and analyze large graphs**

due to **variety** and **volume**

DBpedia describes ~***315 million*** people, places, etc.

Twitter has ~***2 billion*** follower edges between people.



Motivation to Summarize Large Graphs

- Well Known Problem
  - Compute the ***bisimilar contraction*** (a summary) of a labelled graph.
- Practical Applications
  - Data Management: ***exploration*** and ***query optimization*** for semi-structured/XML/RDF data. Motivation to **summarize large graphs** for exploration and query optimization.
- Construction Implementations Exploiting Parallelism
  - Message Passing Interface, Map-Reduce
  - ***Scalable Graph Systems with a Vertex-centric Model***

Background: Bisimulation Contractions

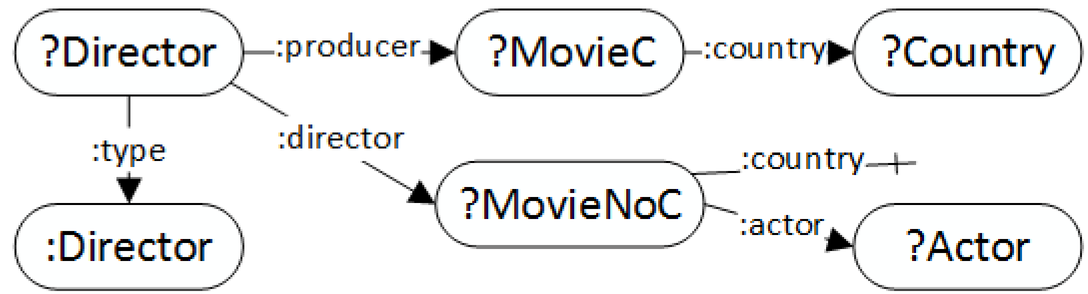
- Two nodes v, v' are ***FWBW bisimilar*** (written v ≈ v'):
  - (FW) if (v, w) is an edge with label l, then there is an outgoing edge (v', w') with label l, and w ≈ w'; and
  - (BW) if (w, v) is an edge with label l, then there is an incoming edge (w', v') with label l, and w ≈ w'.

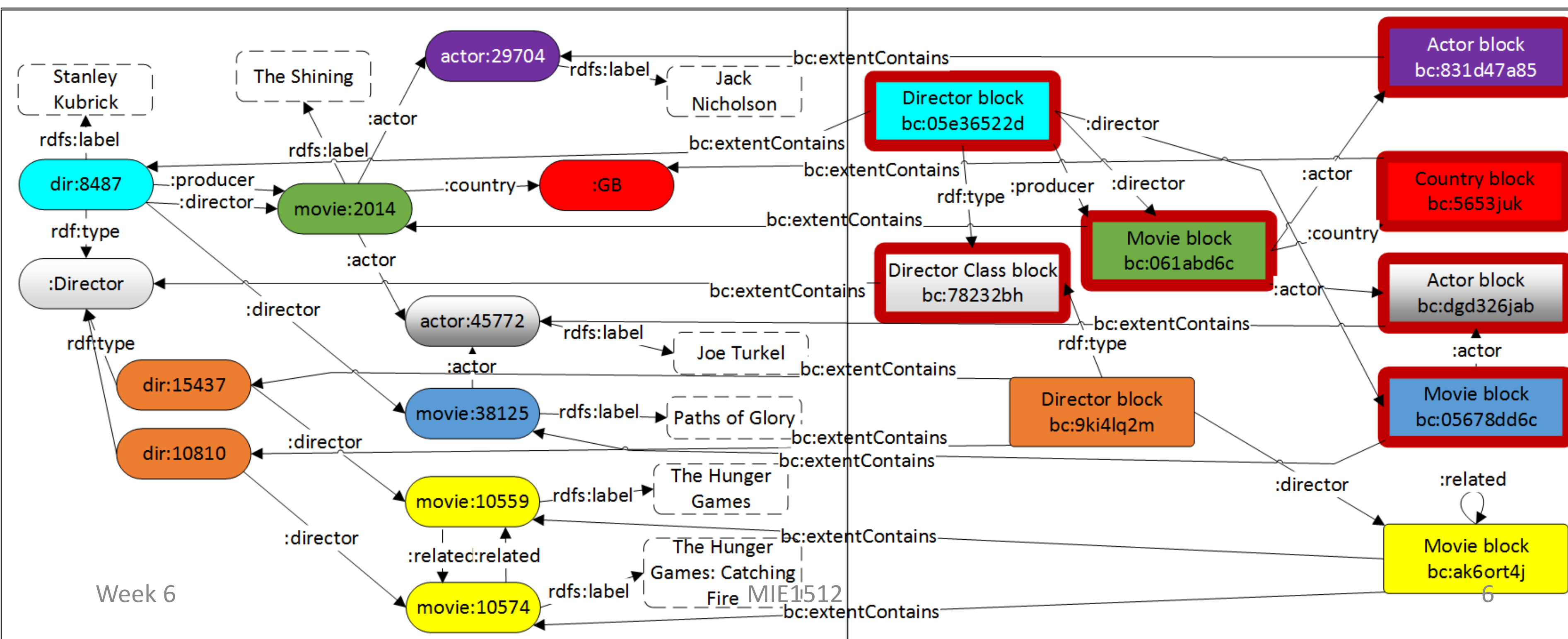
- A summary has ***blocks*** and ***block edges***
  - Each block has an ***extent*** that groups ***all*** bisimilar nodes; this leads to a summary containing the fewest possible blocks and is commonly referred to as a *contraction*.
  - A ***singleton*** block contains exactly 1 node in its extent, i.e., the node is not bisimilar to any other node.

Query Optimization Using Summaries

A ***navigational query*** returns only the query's endpoints.

EXAMPLE: *Find **actors** who appear in **movies without country** information; additionally, the **director** of those movies should also be the **producer** of a movie with **country** information.*





Query Optimization Using Summaries

A … s.

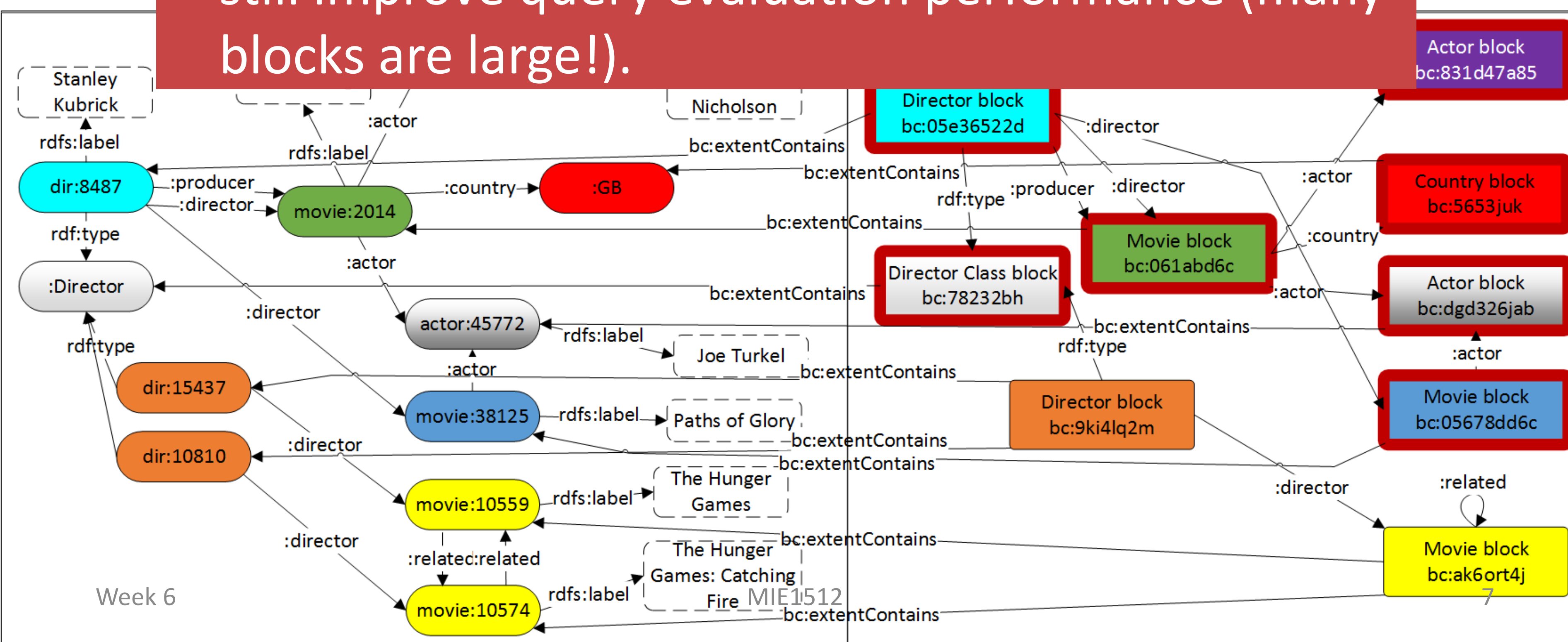
EXAMPL…
movies v…
additiona…
movies s…
a movie

Like in the example below, in many real-world datasets, the ***majority*** of blocks are **singletons**.

Even if a summary has mostly singletons, it can still improve query evaluation performance (many blocks are large!).

…Country

…Actor



Summary Construction State of the Art

- Naïve algorithm by [Kanellakis,Smolka'90]
- MPI implementation by [Blom,Orzan'02] processes ***unstable*** blocks
  - A block is unstable if a node in that block has an edge to a block that was split
- Map-Reduce implementations
  - Address block size (and reducer load) skew [Lange,Fletcher,Bra,Hidders,Wu'13]
  - (HP,HL) Reduce MR tasks per iteration from 3 to 2 [Schatzle,Neu,Lausen,PrzyjacielZablocki'13]

Summary Construction State of the Art

- Naïve algorithm by [Kanellakis,Smolka'90]
- MPI implementation by [Blom,Orzan'02] processes ***unstable*** blocks

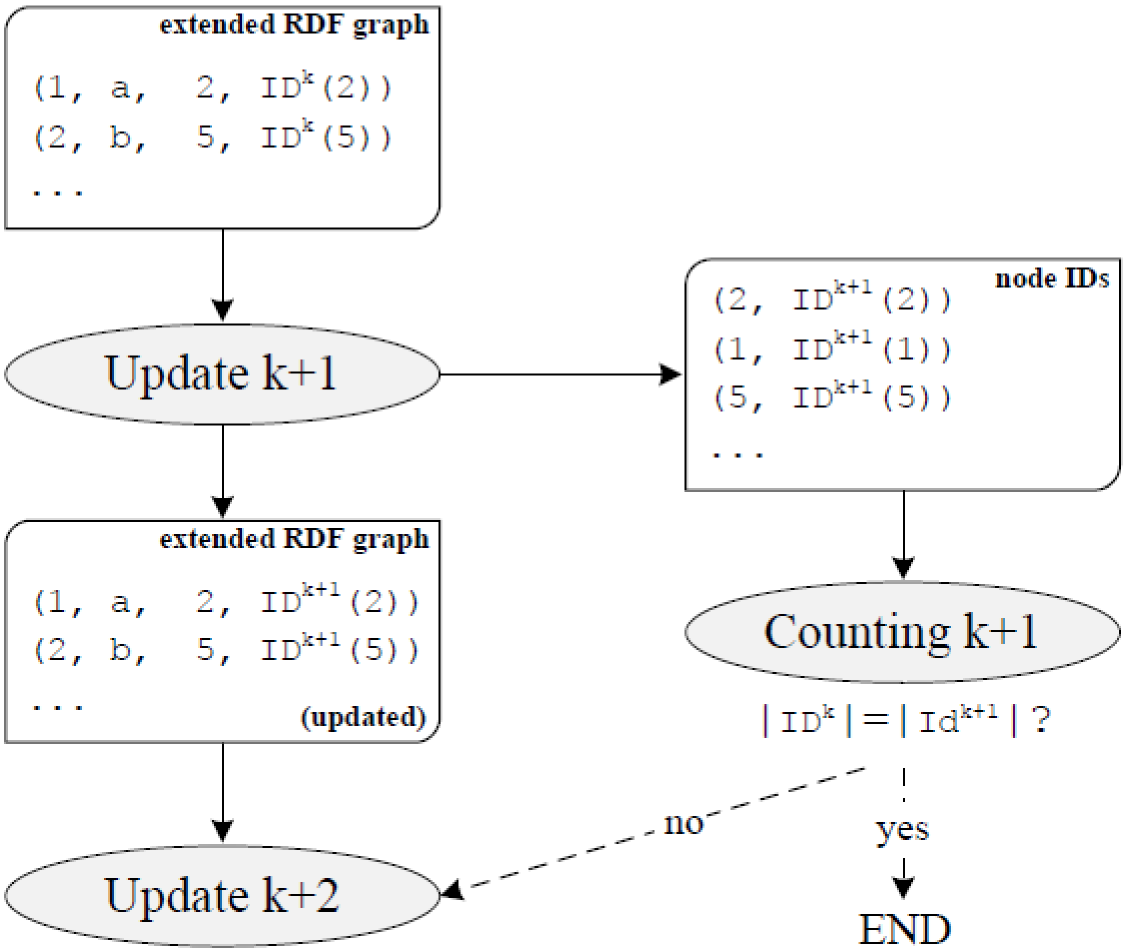
All exhibit similar per-iteration times.

Empirical observation: 10-30 iterations are needed.

- Map-Reduce implementations
  - Address block size (and reducer load) skew [Lange,Fletcher,Bra,Hidders,Wu'13]
  - (HP,HL) Reduce MR tasks per iteration from 3 to 2 [Schatzle,Neu,Lausen,PrzyjacielZablocki'13]

Bisimilarity in MR

From [Schatzle,Neu,Lausen,PrzyjacielZablocki’13]



Map

Algorithm 4: update job - **map**(key, value)

input : *key*: byte offset in input file, can be ignored
value: a quad $(s, p, o, ID^k(o))$

$(s, p, o, ID^k(o)) \leftarrow \text{parseQuad}(value)$

emit$((s,0),\ (s, p, o, ID^k(o)))$

emit$((o,1),\ (s, p, o, ID^k(o)))$

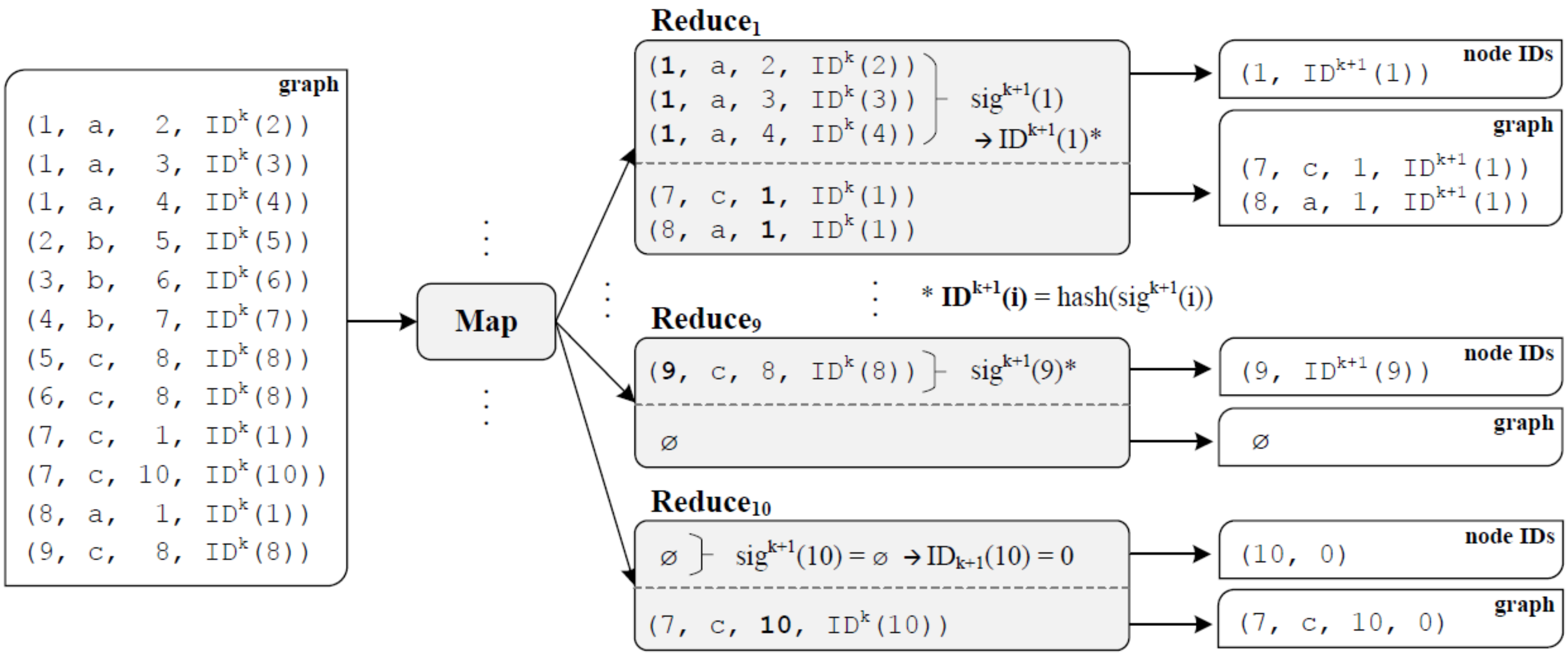
Reduce

Algorithm 5: update job - **reduce**(key, values)

input : *key*: composite key (node s, sortOrder)
values: a list of quads $[(s, p, o, ID^k(o))]$

$sig^{k+1}(s) \leftarrow \{\ \}$
while $value\ in\ values \wedge key.sortOrder = 0$ **do**
    $(s, p, o, ID^k(o)) \leftarrow \text{parseQuad}(value)$
    $sig^{k+1}(s) \leftarrow sig^{k+1}(s) \cup \{(p, ID^k(o))\}$
end
// new signature of s complete
$ID^{k+1}(s) \leftarrow hash(sig^{k+1}(s))$
while $value\ in\ values \wedge key.sortOrder = 1$ **do**
    $(x, p, s, ID^k(s)) \leftarrow \text{parseQuad}(value)$
    emit("graph", $(x, p, s, ID^{k+1}(s))$)
end
// generate separate node ID list for counter job
emit("node IDs", s, $ID^{k+1}(s)$)

MR Iteration Workflow



Efficient Summary Construction

Joint work with Shahan Khatchadourian, Valeria Fiona, Giuseppe Pirro

- Vertex-centric construction of bisimulation summaries using the ***GraphChi*** [Kyrola,Blelloch,Guestrin'12] parallel graph processing framework
  - Similar model to other vertex-centric scalable graph analytics tools (Pregel, Giraph, Hama, GraphLab, PowerGraph, GraphX)
- A novel and very effective ***singleton optimization*** that drastically reduces per-iteration times after only a few iterations
- Experimental validation
  - Significantly outperforms Hadoop state of the art
  - Validation of summary construction times comparable to dataset load and summary write
  - Fast construction of different summaries

GraphChi Processing Model

- Support for ***Bulk Synchronous Parallel*** model [Valiant'90]
  - Nodes execute an Update method in parallel that compute a new block identifier using previous iteration values as input.

$$\boxed{\textbf{parallel foreach } v \in V_G \textbf{ do } \text{V-Update}(v) \textbf{ end}}$$

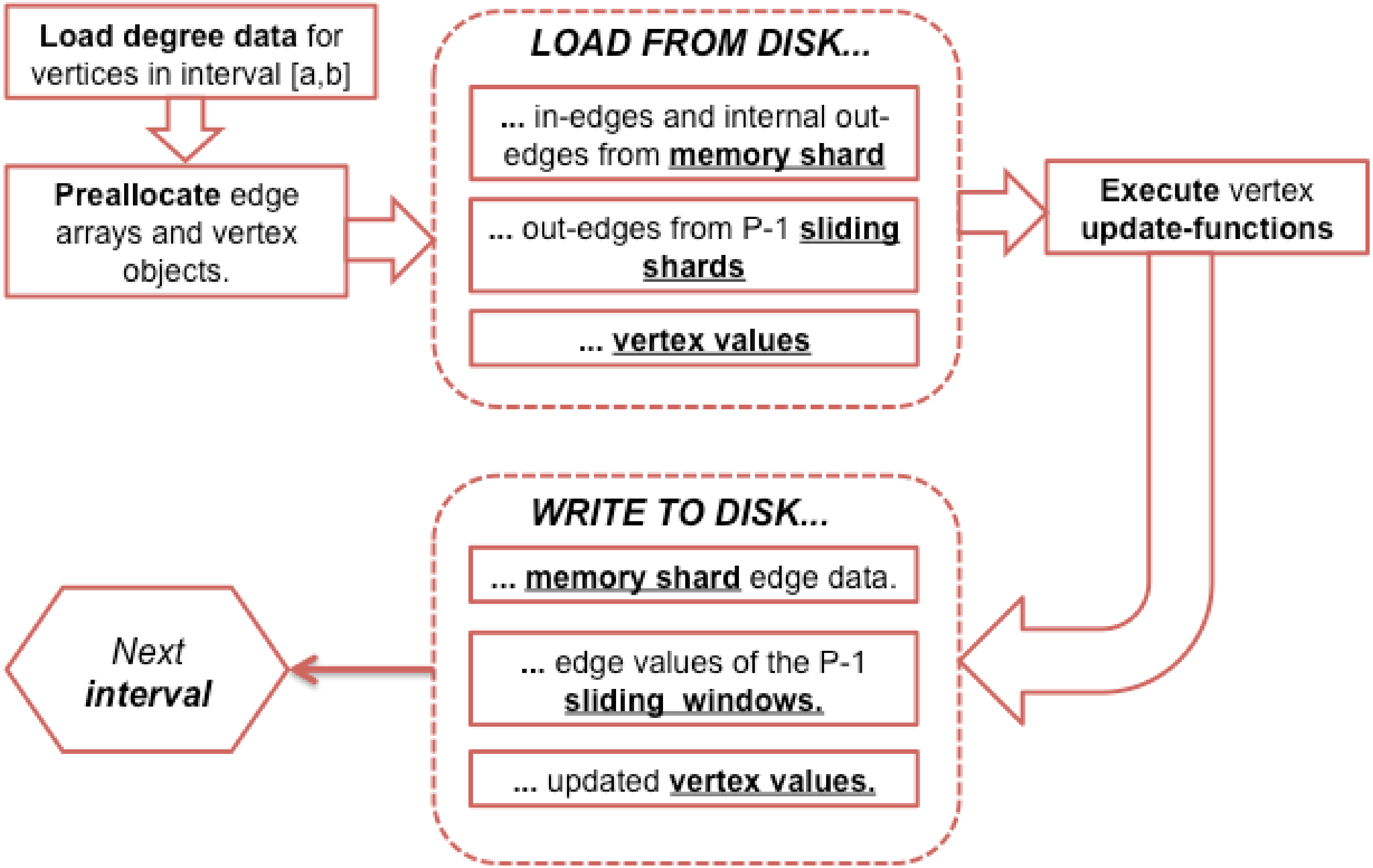
  - GraphChi is effective at processing large graphs because it loads outgoing edges of nodes into main-memory then streams their incoming edges from disk.

Used by singleton optimization to disable singletons

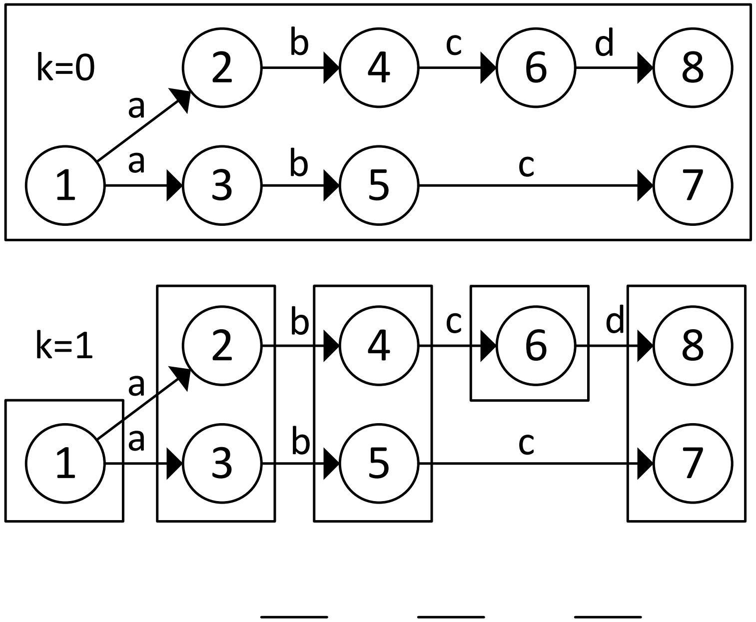
- Support for ***scheduling*** $\boxed{\textbf{disable vertex } v}$
  - Common in graph algorithms, e.g., BFS.
  - GraphChi skips loading disabled nodes from disk.

UofT

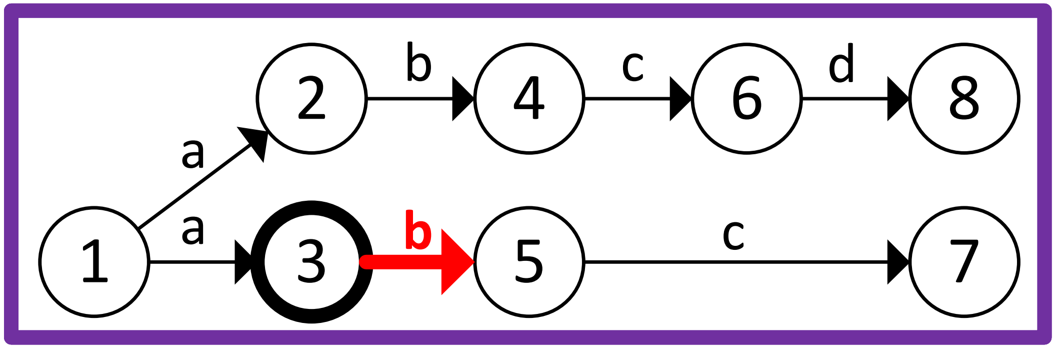
GraphChi Platform on Multicore+Disk



Node V-Update function



Iteration 1



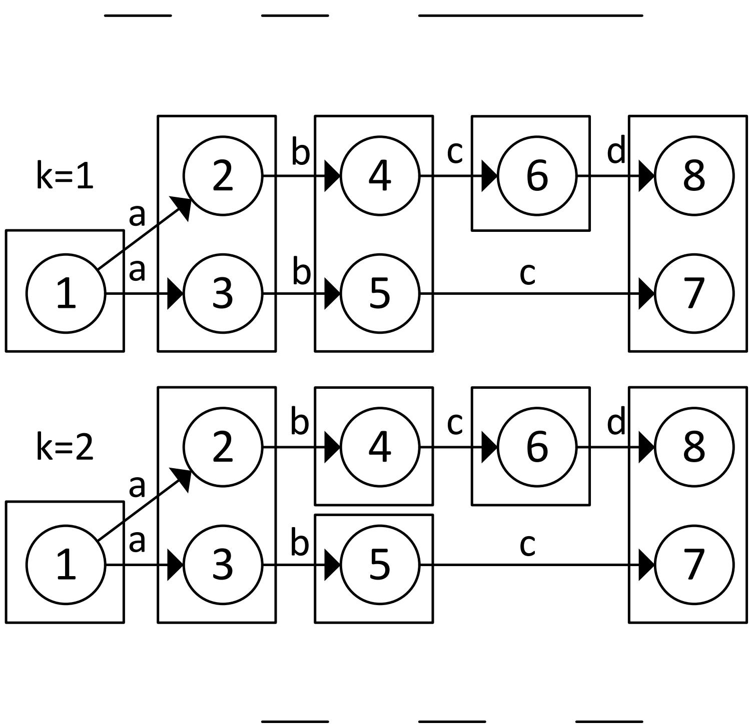
block$_1$(1)=hash(sort({(**a**,**block$_0$(2)**)}))

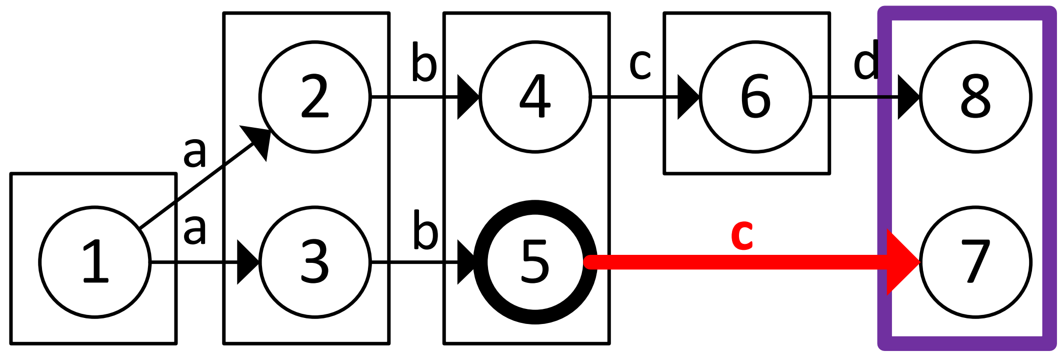
block$_1$(2)=hash(sort({(**b**,**block$_0$(4)**)}))

block$_1$(3)=hash(sort({(**b**,**block$_0$(5)**)}))

Node V-Update function

Iteration 2





$\mathbf{block_2(2)} = \text{hash}(\text{sort}(\{(b, block_1(4))\}))$

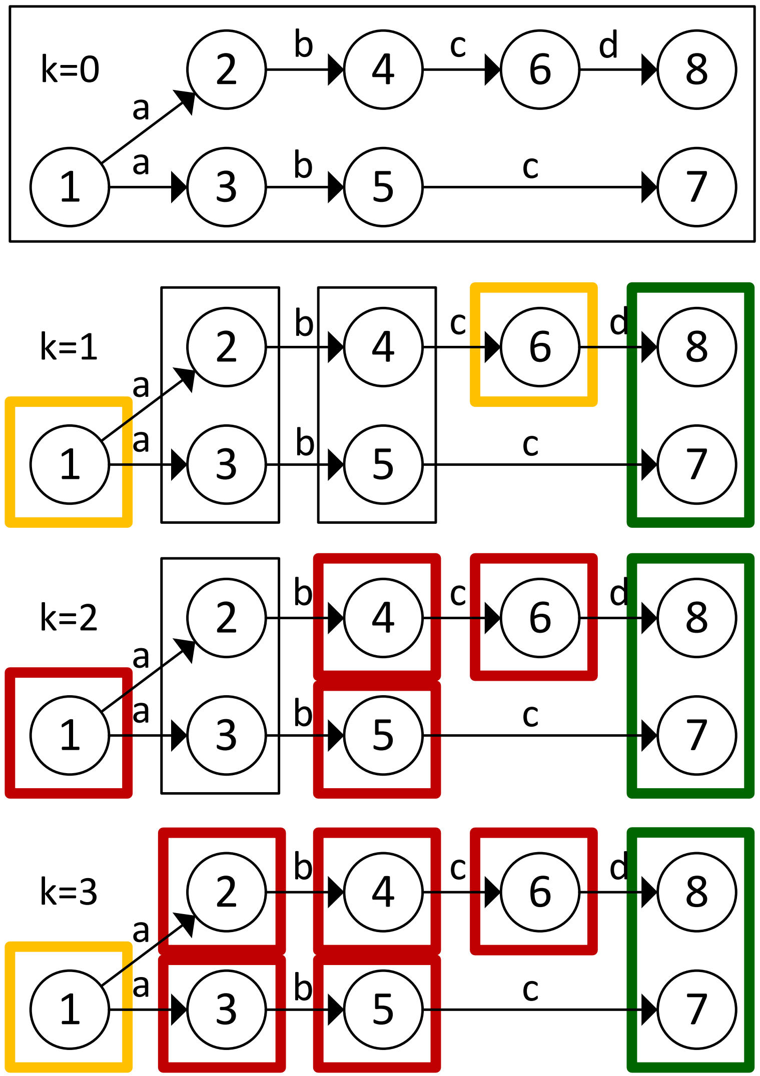
$\mathbf{block_2(3)} = \text{hash}(\text{sort}(\{(b, block_1(5))\}))$

$\mathbf{block_2(4)} = \text{hash}(\text{sort}(\{(c, block_1(6))\}))$

$\mathbf{block_2(5)} = \text{hash}(\text{sort}(\{(c, block_1(7))\}))$

UofT

Stability ≠ Singleton



Stability and Singleton do not reduce to the other.

- **Non-singleton blocks that are Stable**
- **Singleton blocks that are Unstable**

Key empirical observation:
Most blocks are singleton and stable
and singleton is cheaper to check.

Bisimilar Contraction Singleton (B) variation

L1. **foreach** $v \in V_G$ **do** $block_0(v) =$ '0' **end** // initialize
L2. $count_{old} \leftarrow 1, count_{new} \leftarrow 0, k \leftarrow 0$
L3. **while** $count_{old} \neq count_{new}$ **do**
L4.   **parallel foreach** $v \in V_G$ **do** V-Update(v) **end**
L5. $count_{old} \leftarrow count_{new}$
L6. $count_{new} \leftarrow |\{block_{k+1}(v) \mid v \in V_G\}|$
L7. $k \leftarrow k + 1$
L8. **end**
L9. $V_S \leftarrow \{block_k(v) \mid v \in V_G\}$,
$E_S \leftarrow \{(block_k(v), block_k(v')) \mid (v, v') \in E_G\}$

Method: V-Update(v)

V1. $fwsig(v) = \{(+m(v, v'), block_k(v')) \mid (v, v') \in E_G, m(v, v') \in M\}$
V2. $bwsig(v) = \{(-m(v', v), block_k(v')) \mid (v', v) \in E_G, m(v', v) \in M\}$
V3. $block_{k+1}(v) = hash(sort(block_k(v) \cup fwsig(v) \cup bwsig(v)))$

Bisimilar Contraction Singleton (S) variation

L1. **foreach** $v \in V_G$ **do** $block_0(v) =$ '0' **end** // initialize
L2. $count_{old} \leftarrow 1, count_{new} \leftarrow 0, k \leftarrow 0$
L3. **while** $count_{old} \neq count_{new}$ **do**
L4′. **parallel foreach** $v \in V_G$ **do** V-Update-Singleton(v)
L5. $\quad count_{old} \leftarrow count_{new}$
L6. $\quad count_{new} \leftarrow |\{block_{k+1}(v) \mid v \in V_G\}|$
L7. $\quad k \leftarrow k + 1$
L8. **end**
L9. $V_S \leftarrow \{block_k(v) \mid v \in V_G\}$,
$E_S \leftarrow \{(block_k(v), block_k(v')) \mid (v, v') \in E_G\}$

Method: V-Update-Singleton(v)

S1. **if** $|\{v' \mid v' \in block_k(v)\}| > 1$ **then do**
S2. $\quad$ V-Update(v)
S3. **else do**
S4. $\quad$ **disable vertex** v
S5. **end**

Experimental Validation Datasets

Dataset	$\|N\|$	$\|E\|$
lmdb	1,327,120	6,147,717
dbp	48,603,466	317,220,816
Twitter	52,579,678	1,963,263,821

- LinkedMDB [Hassanzadeh,Consens'09] (lmdb) describes movies and related entities using 222 edge labels
- DBpedia (dbp) describes real-world entities such as people and places using 1,393 edge labels
- Twitter has unlabeled edges to describe follower relationships amongst its users

FW Summaries - GraphChi vs. Hadoop

For **LinkedMDB**

- (HP) Pseudo-distributed is over ***2x faster*** than (HL) Local mode.
- S is ***8x faster*** than HP.

For **DBPedia**

- S is around ***6x faster*** than HP and ***3x faster*** than B.
- HP is more than ***7x faster*** than HL.
- HL requires ~ 10 hours per iteration (vs. 10-30 mins).

Summary	$\|N\|$	$\|E\|$	B	S	HL	HP
lmdb	85,714	999,934	4.2	4.0	73	32
dbp	13,429,903	229,490,296	941	331	5,832*	1,972

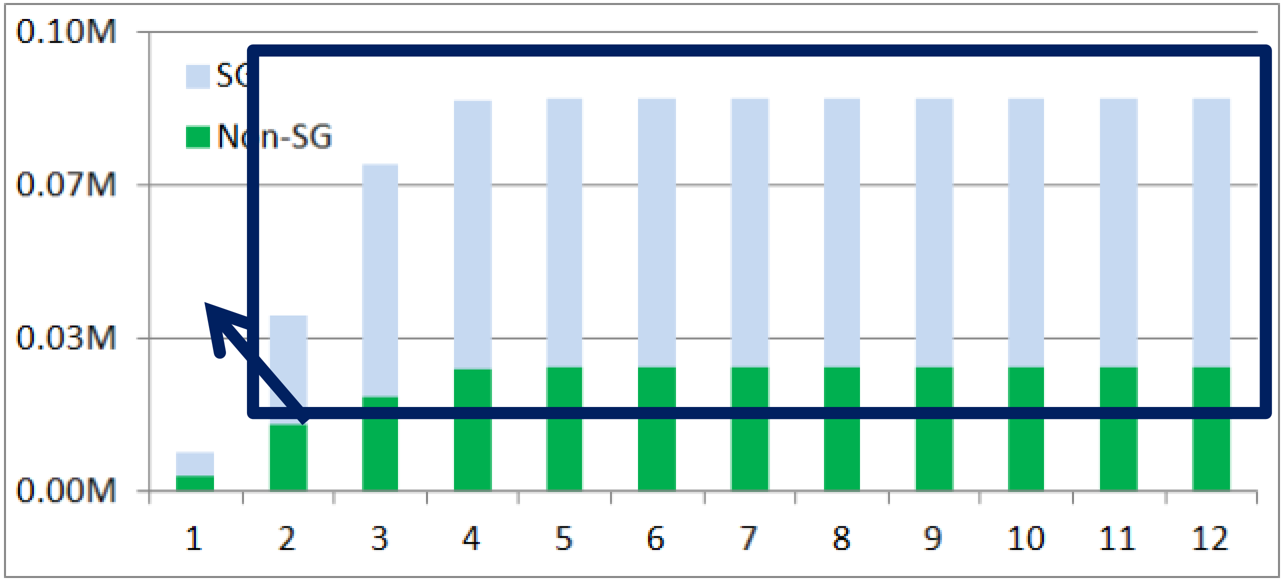
* time for first 10 iterations

UofT

FW Summaries Experimental Results

LinkedMDB

DBpedia



Millions of blocks and singletons per iteration

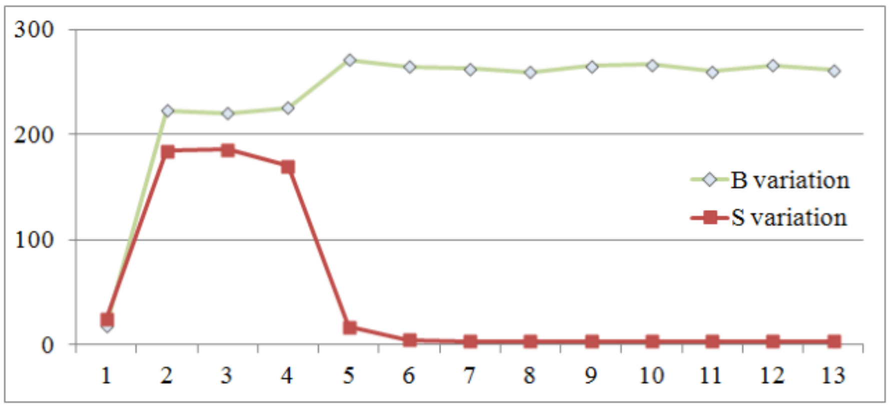
FWBW Construction vs Load plus Write

- S variation is ~***3x, 4x, and 5x faster*** than the B variation for *LinkedMDB*, *DBpedia*, and *Twitter*, respectively.
- Time taken to *compute* dbp's FWBW summary using the S variation is ***over 50% faster*** than the time to *load* the dataset graph plus *write* the summary.

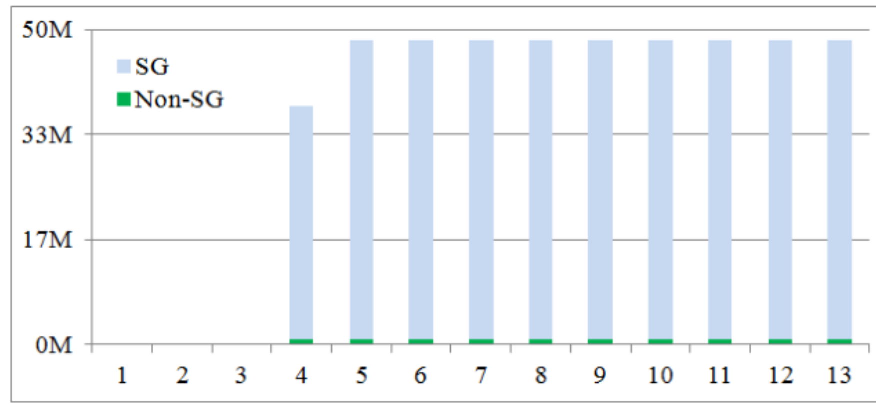
Summary	$\|N\|$	$\|E\|$	Load	B	S	Write
lmdb	844,877	4,311,098	0.12	8.6	2.8	0.75
dbp	32,274,111	278,182,230	107	775	187	207
Twitter	48,332,025	1,945,307,755		3,118	632	

UofT

FWBW Summaries Experimental Results



(a) M1 time (in mins) per iteration



(b) Millions of singletons and non-singletons per iteration

Twitter FWBW summary construction

(a) M1 time (in mins) per iteration

(b) Millions of singletons and non-singletons per iteration

Twitter

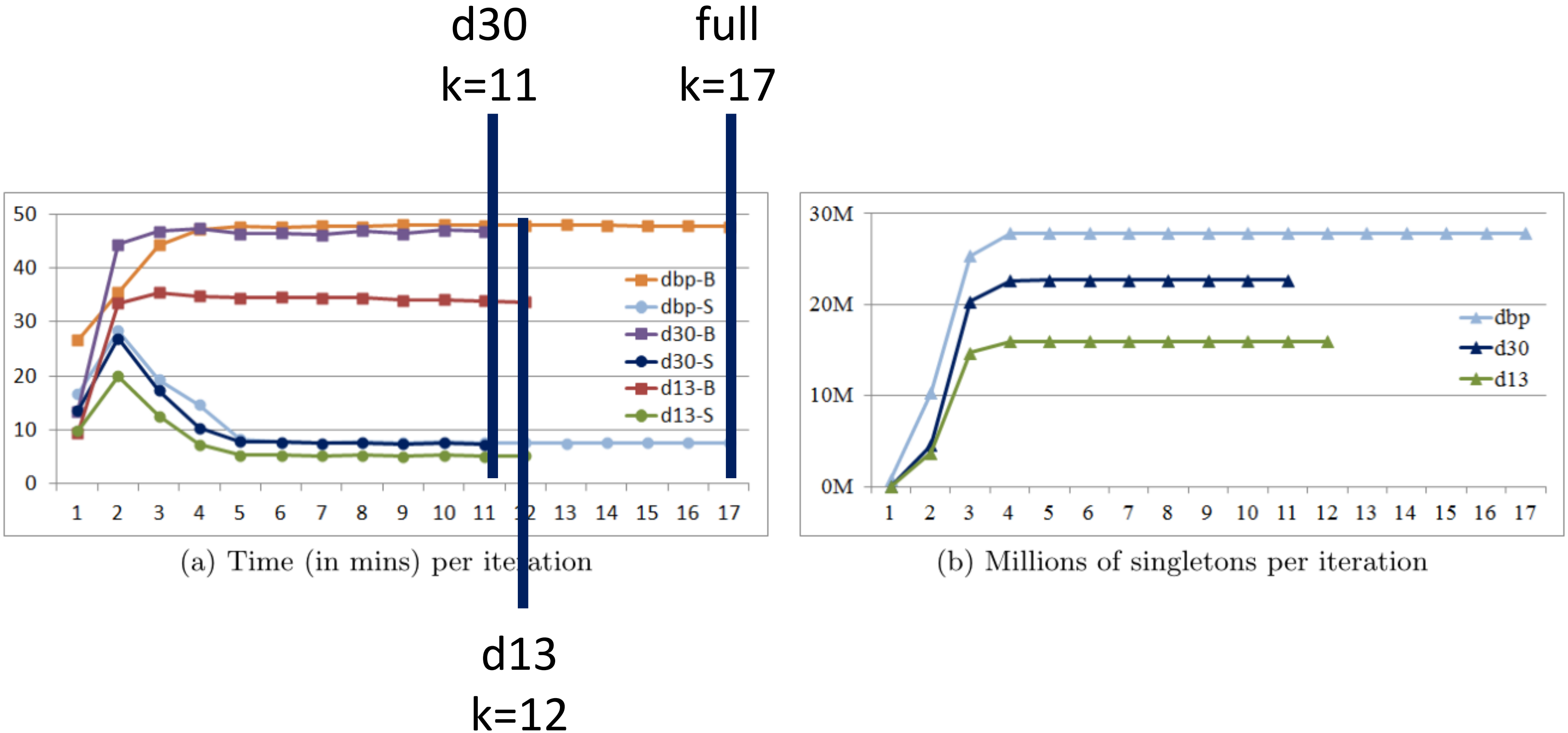
Full and Selective FWBW Summaries

Selective summaries, that summarize a subset of a dataset's predicates, are almost ***2x faster*** to compute than the full summary. The S variation constructs the d13 summary over ***2x faster*** than the instance load plus summary write.

Summary	Load	B	S	Write
dbp	107	775	187	207
d30	160	611	125	125
d13	131	411	95	95

Selective summaries can improve query performance more than a full summary.

Selective Summaries Experimental Results



(a) Time (in mins) per iteration

(b) Millions of singletons per iteration

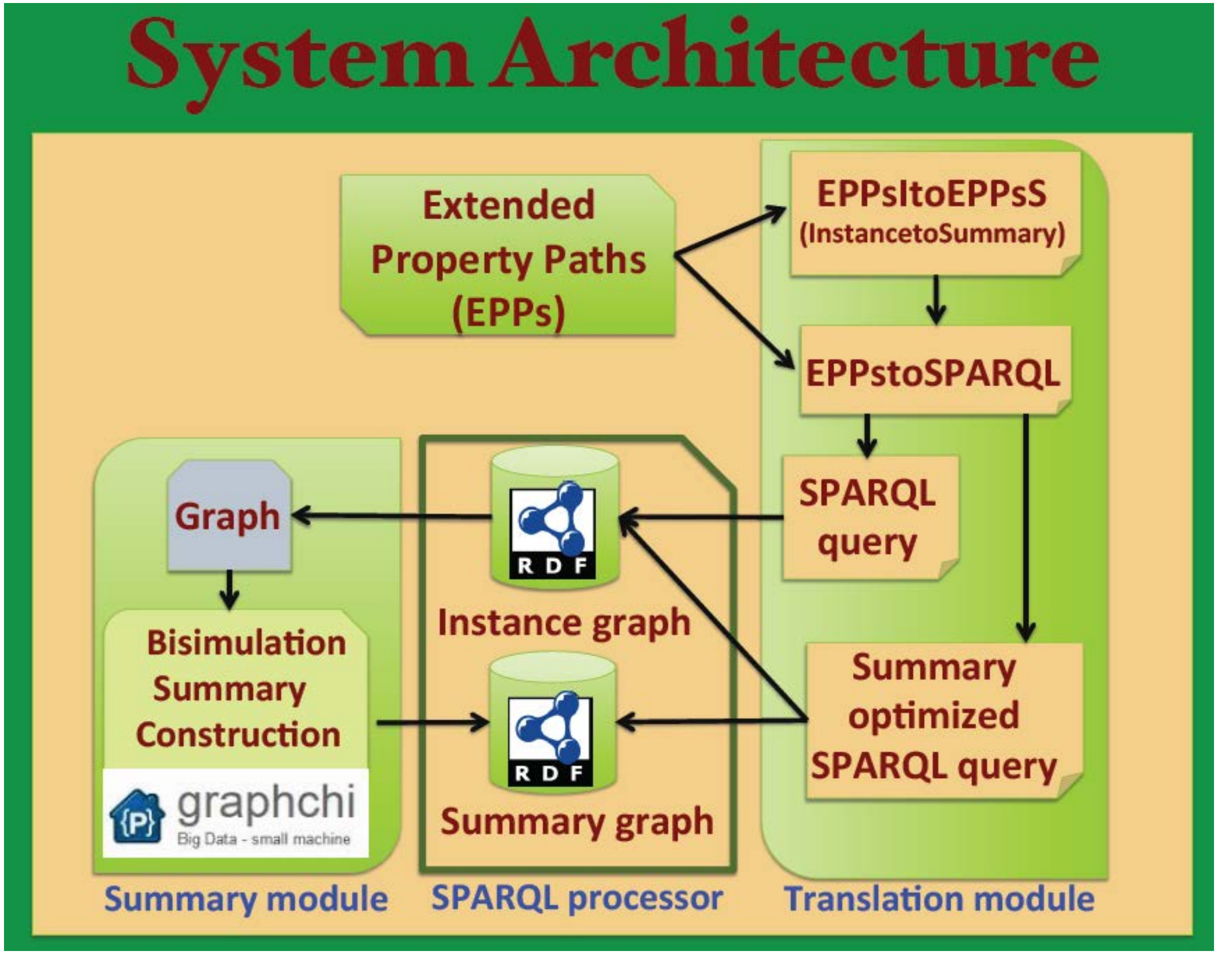
Selective summaries can require different iterations.

Remarks

Large bisimulation summaries can be ***constructed efficiently*** using scalable graph analytics platforms (with big improvements thanks to a novel ***singleton optimization***)

Full and selective summaries can ***improve query performance*** – we developed a system to achieve this on existing (unmodified) SPARQL processors using query translations

An Overview of S+EPPs



[CFKP'15] Consens M. P., Fionda V., Khatchadourian S., Pirrò G. S+EPPs: Construct and Explore Bisimulation Summaries, plus Optimize Navigational Queries; all on Existing SPARQL Systems, PVLDB 2015