Exploiting data staleness for high-performance machine learning

Jim Cipar

Qirong Ho, Greg Ganger,

Eric Xing, Kim Keeton (HP Labs)

PARALLEL DATA LABORATORY

Carnegie Mellon University

Carnegie Mellon
Parallel Data Laboratory

Overview

- Intermediate data crucial for ML performance
- LazyTables: very fast intermediate data
- Achieve high performance by allowing stale data
  - This is OK for many ML algorithms

Carnegie Mellon
Parallel Data Laboratory

Outline

- Insights from LazyBase
- Machine learning applications
- LazyTables design
- Future research

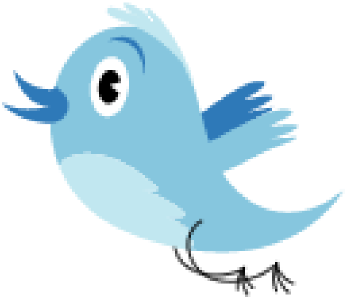
Carnegie Mellon
Parallel Data Laboratory

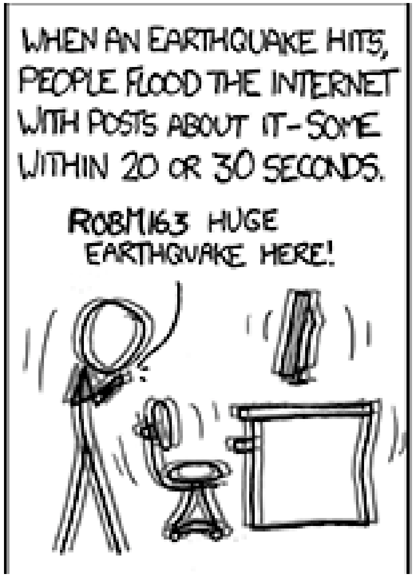
LazyBase

- Database designed for analysis of observations
  - E.g. Information management, social network data
  - Continuous high-throughput updates
- **Key observation:** Applications can use stale data
  - Different queries have different freshness requirements
  - Allowing for staleness can improve performance

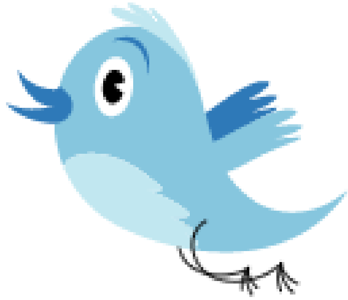
Example application

- High bandwidth stream of Tweets
  - 200 million per day
  - Up to 20k per second

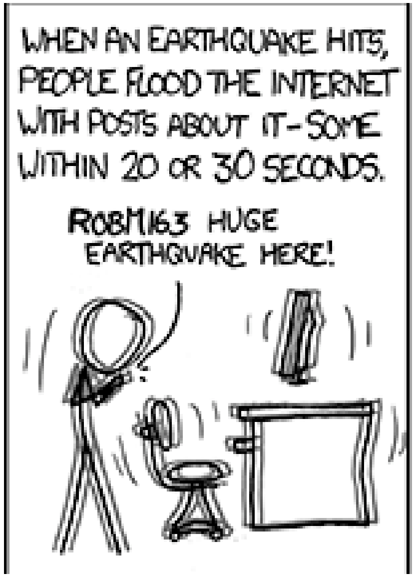




Example application



- High bandwidth stream of Tweets
  - 200 million per day
  - Up to 20k per second



•Queries accept different freshness levels

•Freshest: USGS Twitter Earthquake Detector

•Fresh: Hot news in last 10 minutes

•Stale: social network graph analysis

•**Freshness depends on query** not data

Applications and freshness

Freshness / Domain	Seconds	Minutes	Hours+
Transportation	Emergency response	Real-time traffic maps	Traffic engineering, route planning

Applications and freshness

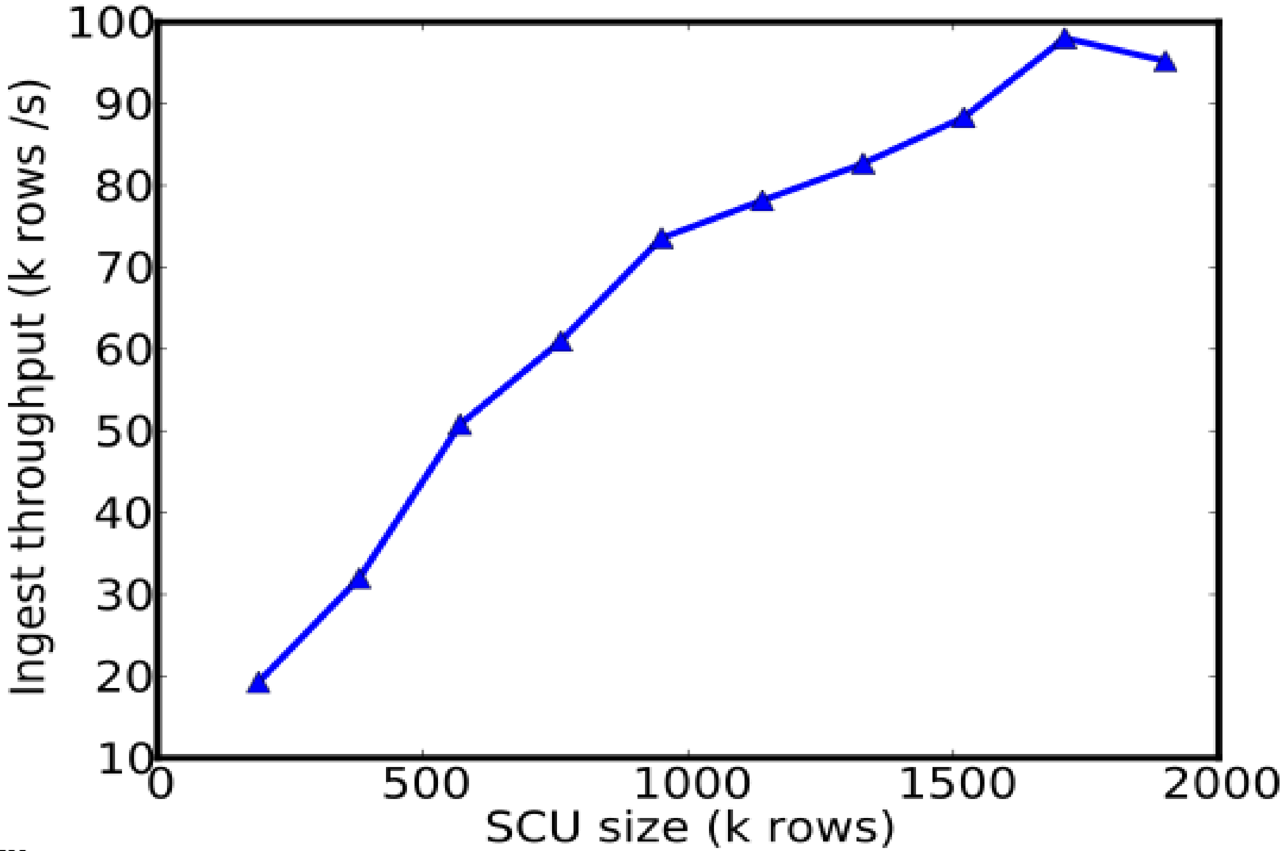
Freshness / Domain	Seconds	Minutes	Hours+
Transportation	Emergency response	Real-time traffic maps	Traffic engineering, route planning
Retail	Real-time coupons, targeted ads	Just-in-time inventory	Product search, earnings reports
Enterprise information management	Infected machine identification	File-based policy validation	E-discovery requests, search

High throughput updates

- Must support continuous high-volume update
- Batching: group many updates, apply at once
- **Batching updates provides high performance**
  - Common technique for high throughput
  - Amortize bookkeeping costs for performing updates

Batching and performance

Large batches of updates increase throughput



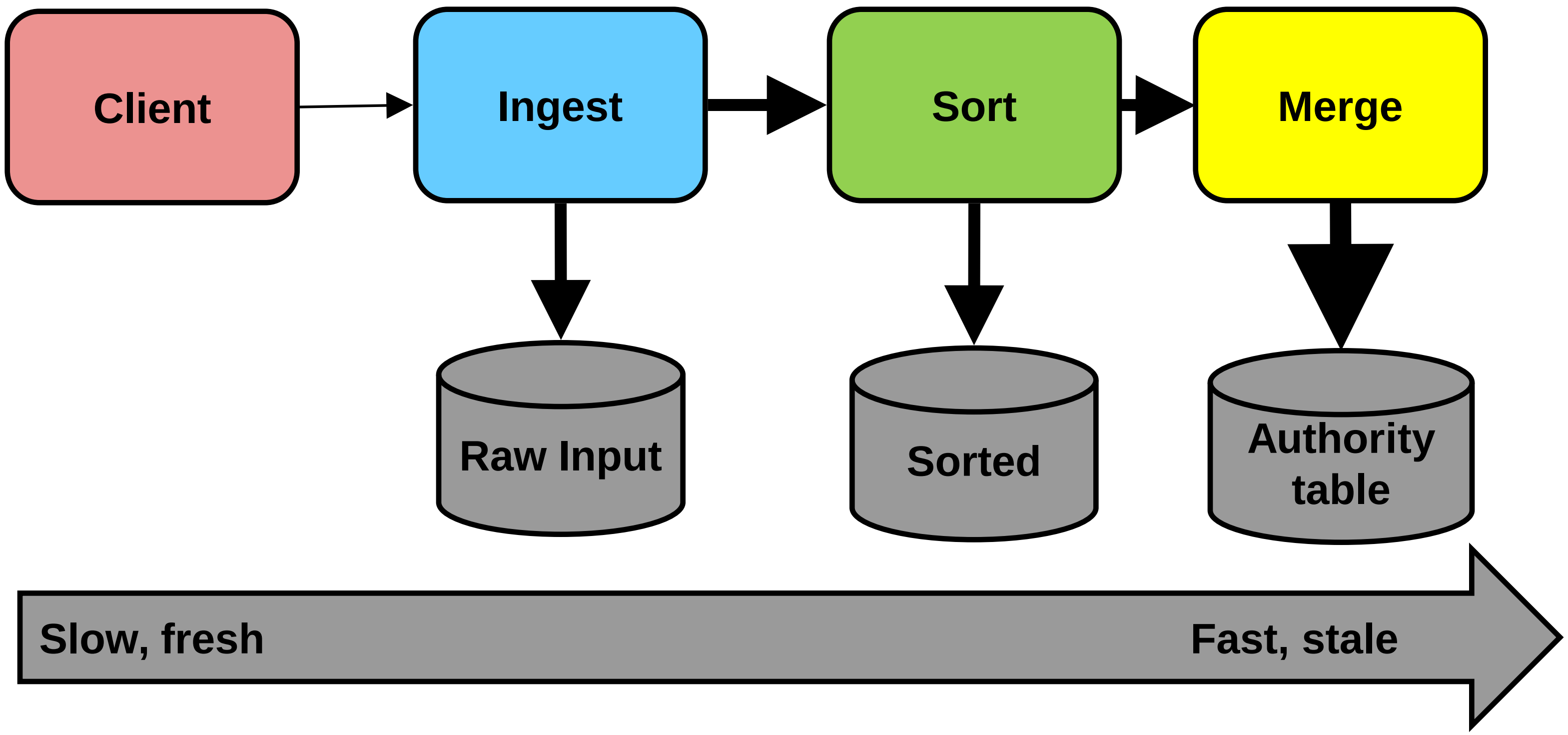
Batching causes staleness

- Large updates take a long time to process
  - Large batches ➔ database is very stale
  - Very large batches/busy system ➔ could be hours old

- Staleness OK for some queries, bad for others

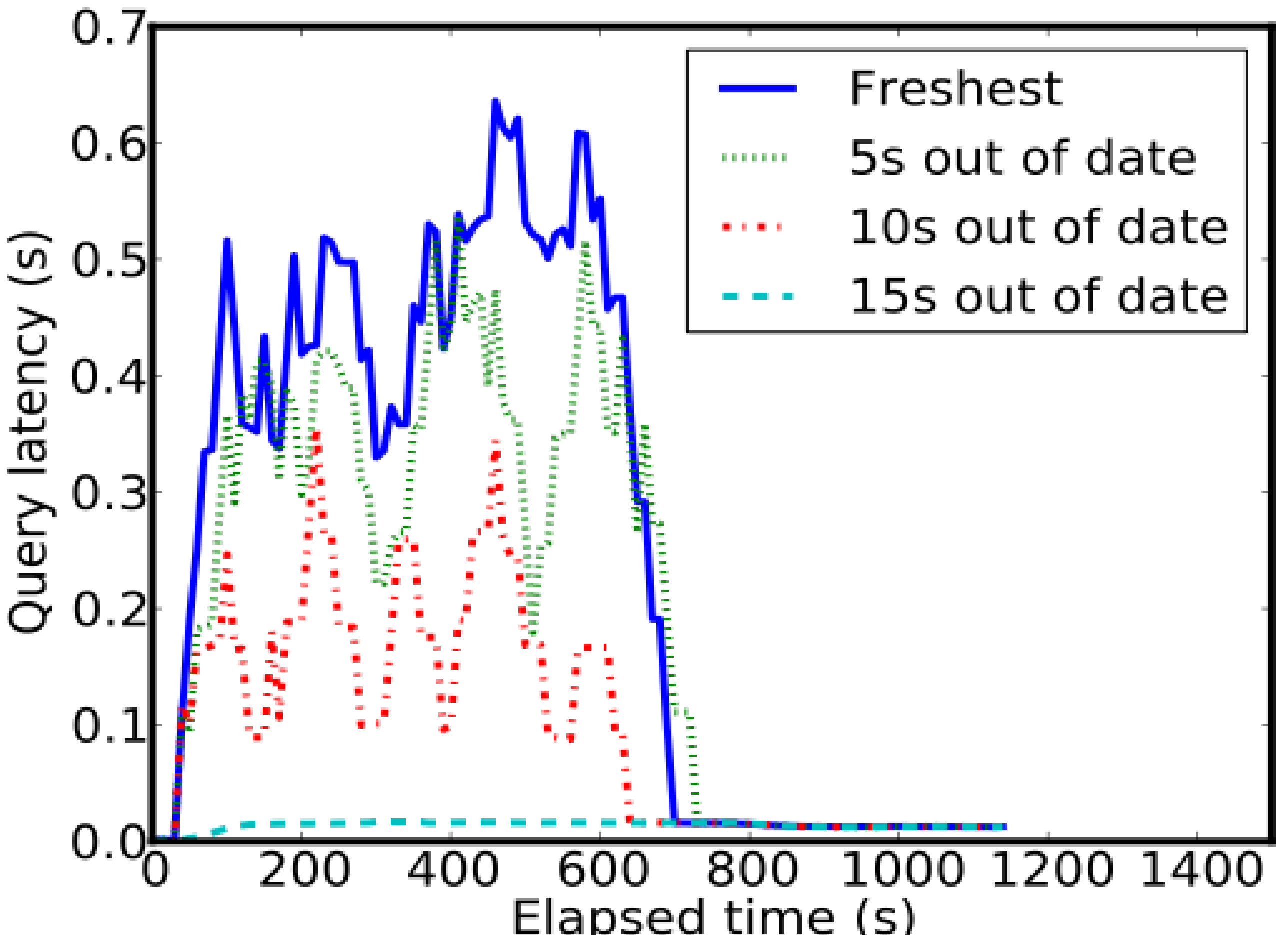
Solution: allow queries to access data before it's been applied to database

LazyBase pipeline



Query latency/freshness

Queries allowing staler results return faster



Carne
Paralle

Insights from LazyBase

- Improving performance can cause staleness

- Many applications tolerate data staleness

- Freshness requirements are important
  - Property of query, not data
  - Can change over time
  - Make them explicit, not implied

Outline

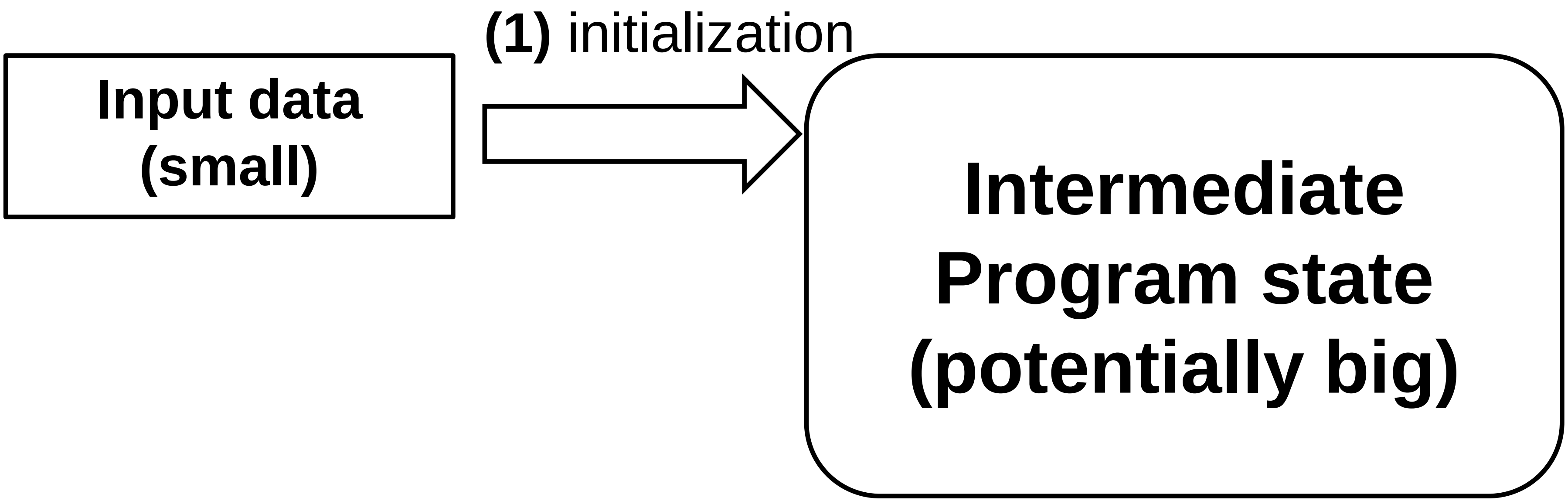
- Insights from LazyBase
- Machine learning applications
- Lazy writes and initial results
- System design
- Future research

Carnegie Mellon
Parallel Data Laboratory

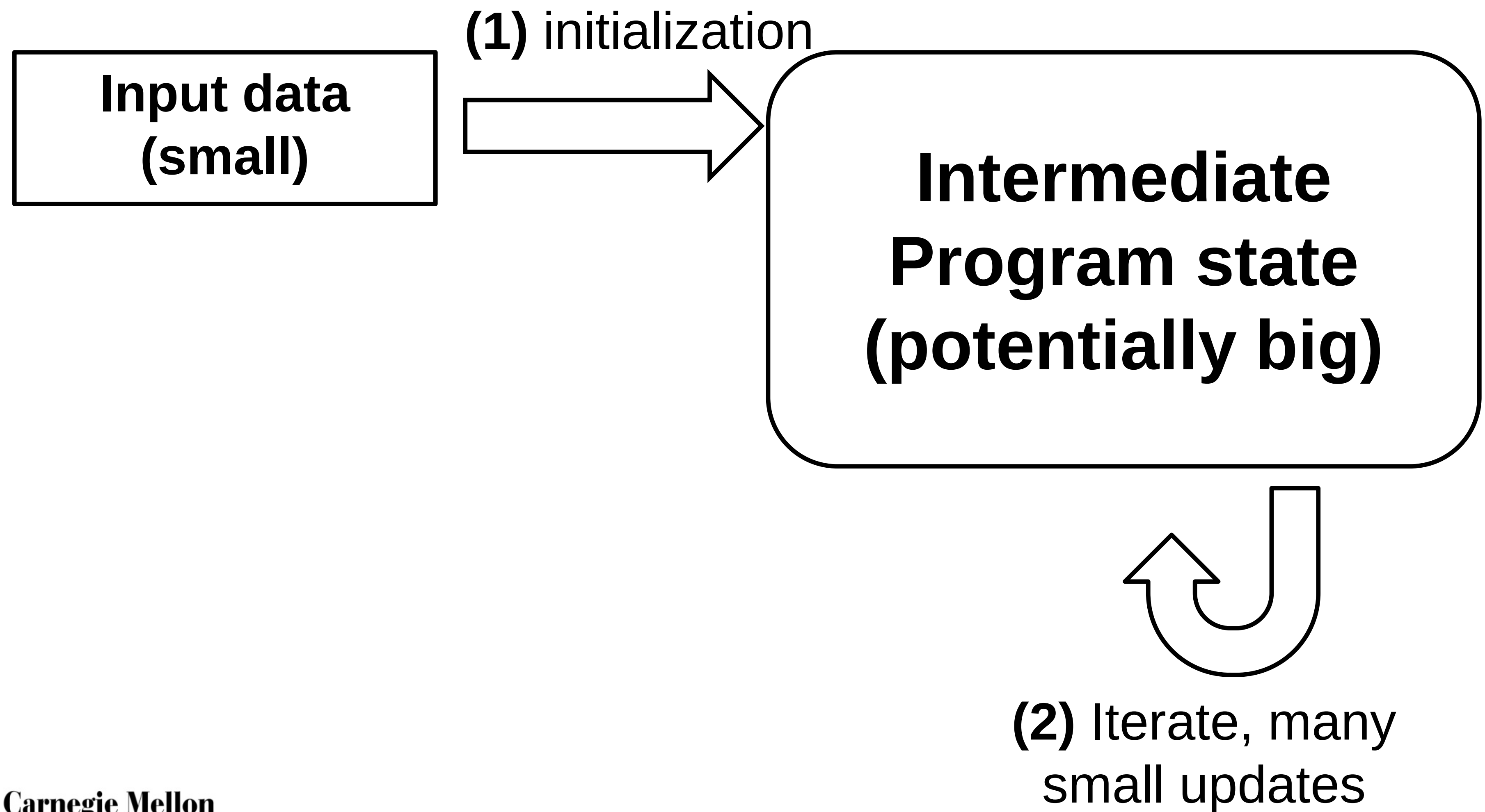
A typical ML algorithm

Input data (small)

A typical ML algorithm

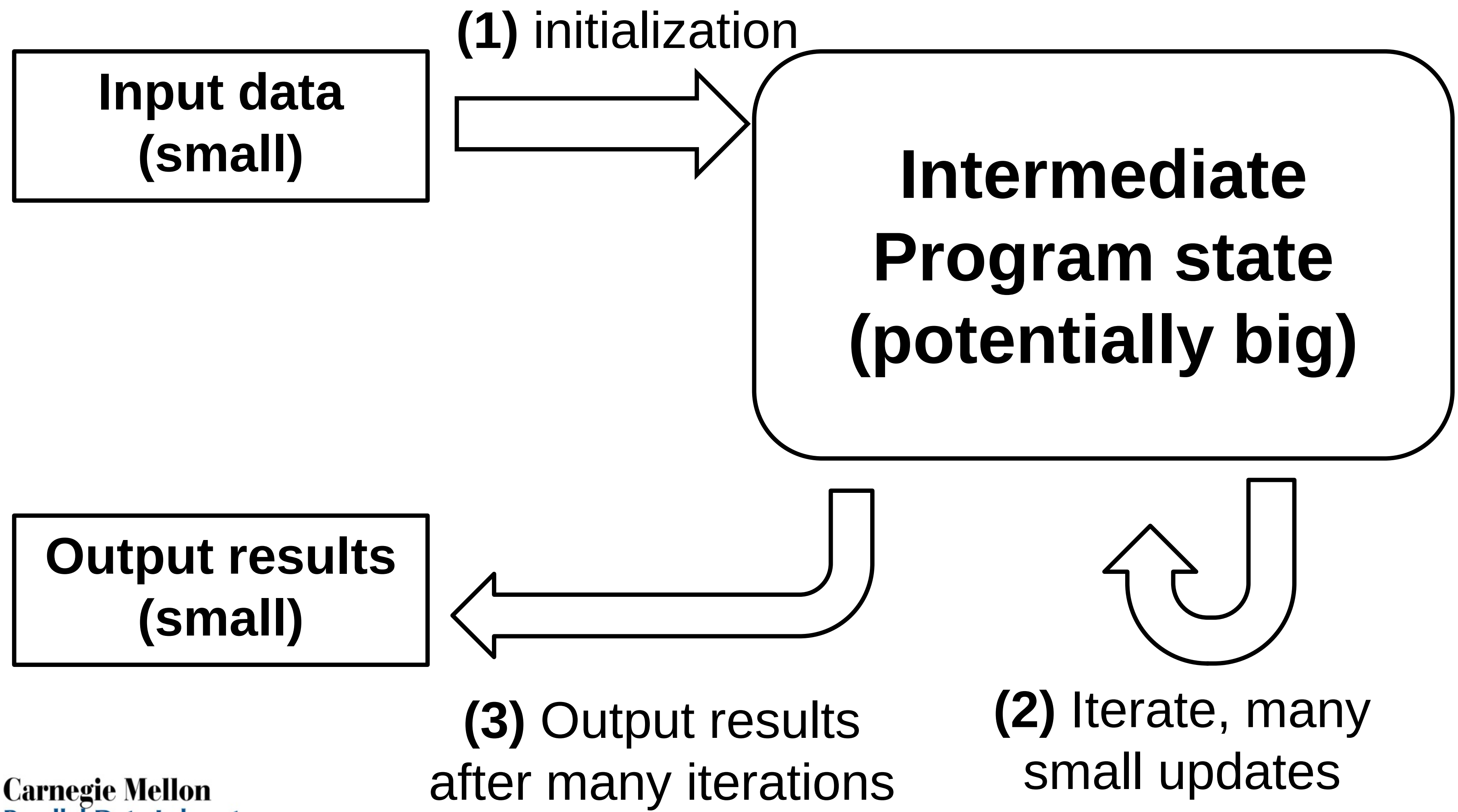


A typical ML algorithm



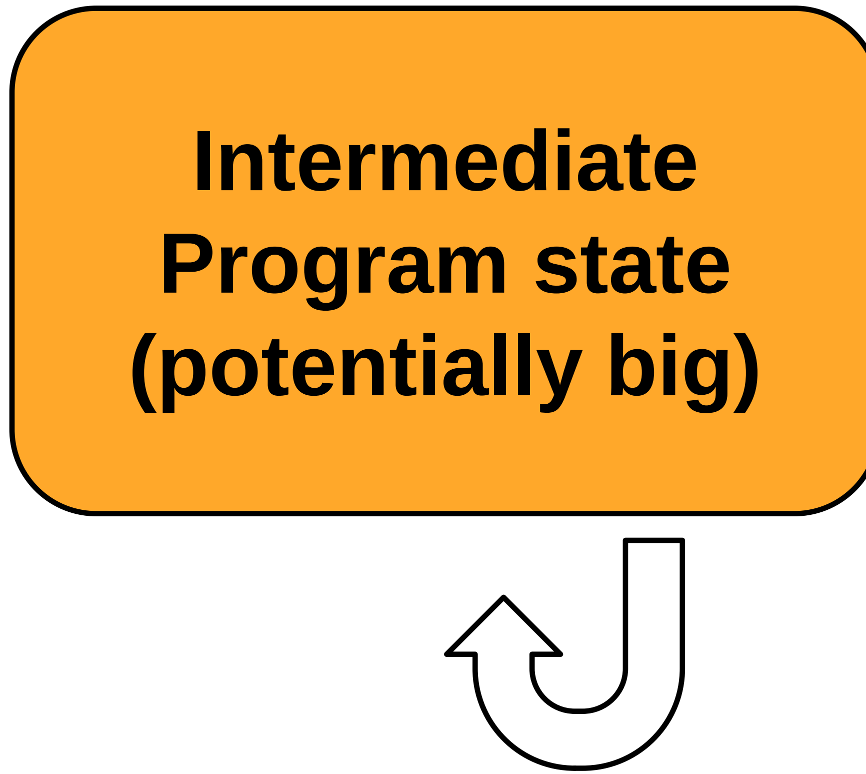
Carnegie Mellon
Parallel Data Laboratory

A typical ML algorithm

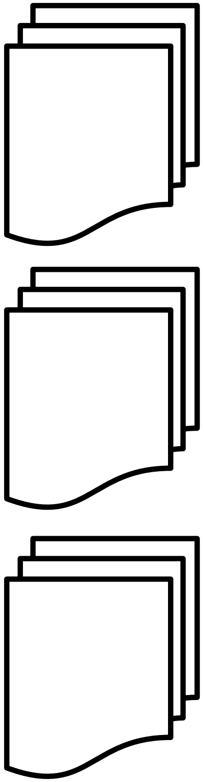


A typical ML algorithm

- **Bulk of time spent in iteration steps**
- **Performance of intermediate data crucial to performance of algorithm**



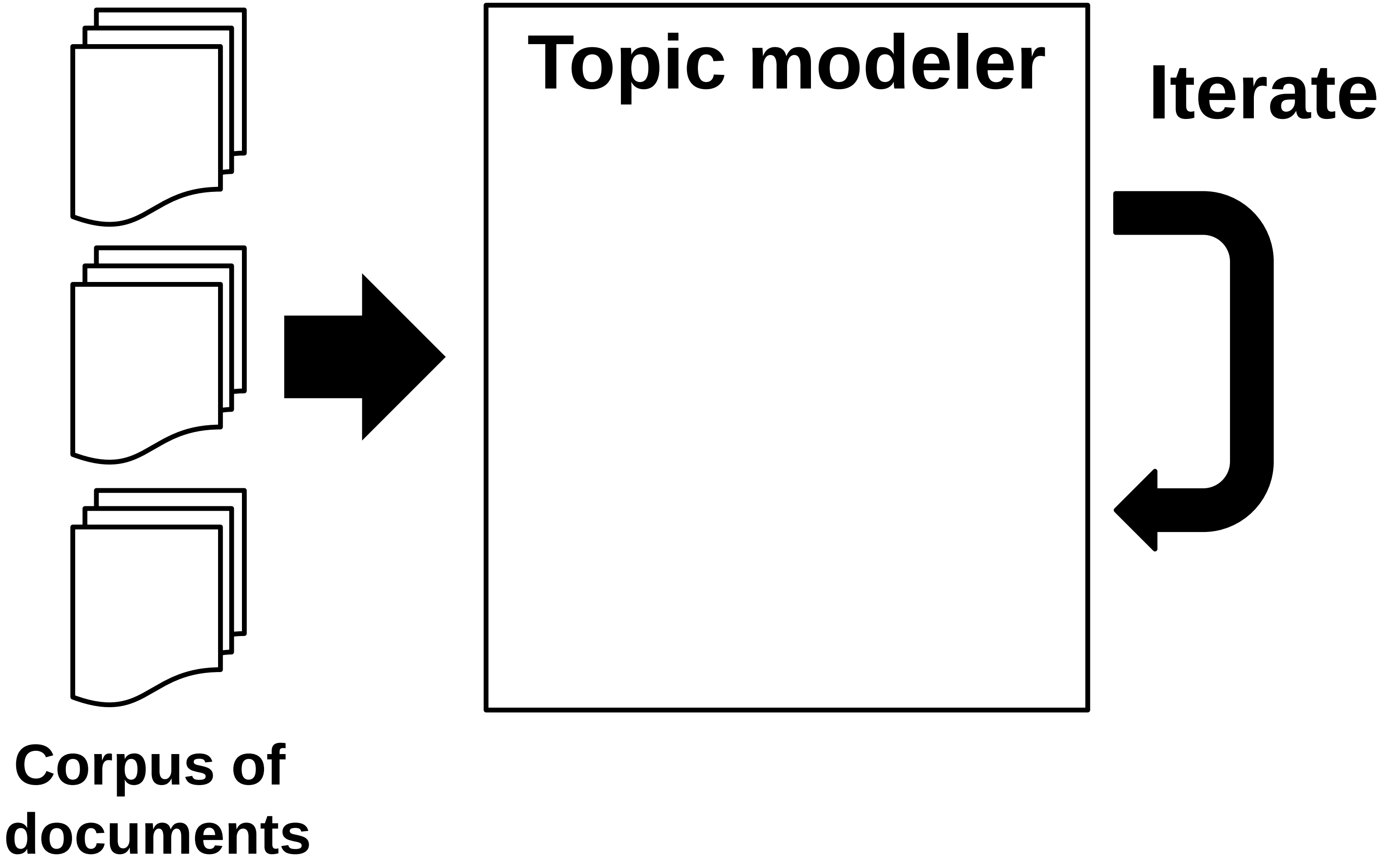
Example: Topic modeling



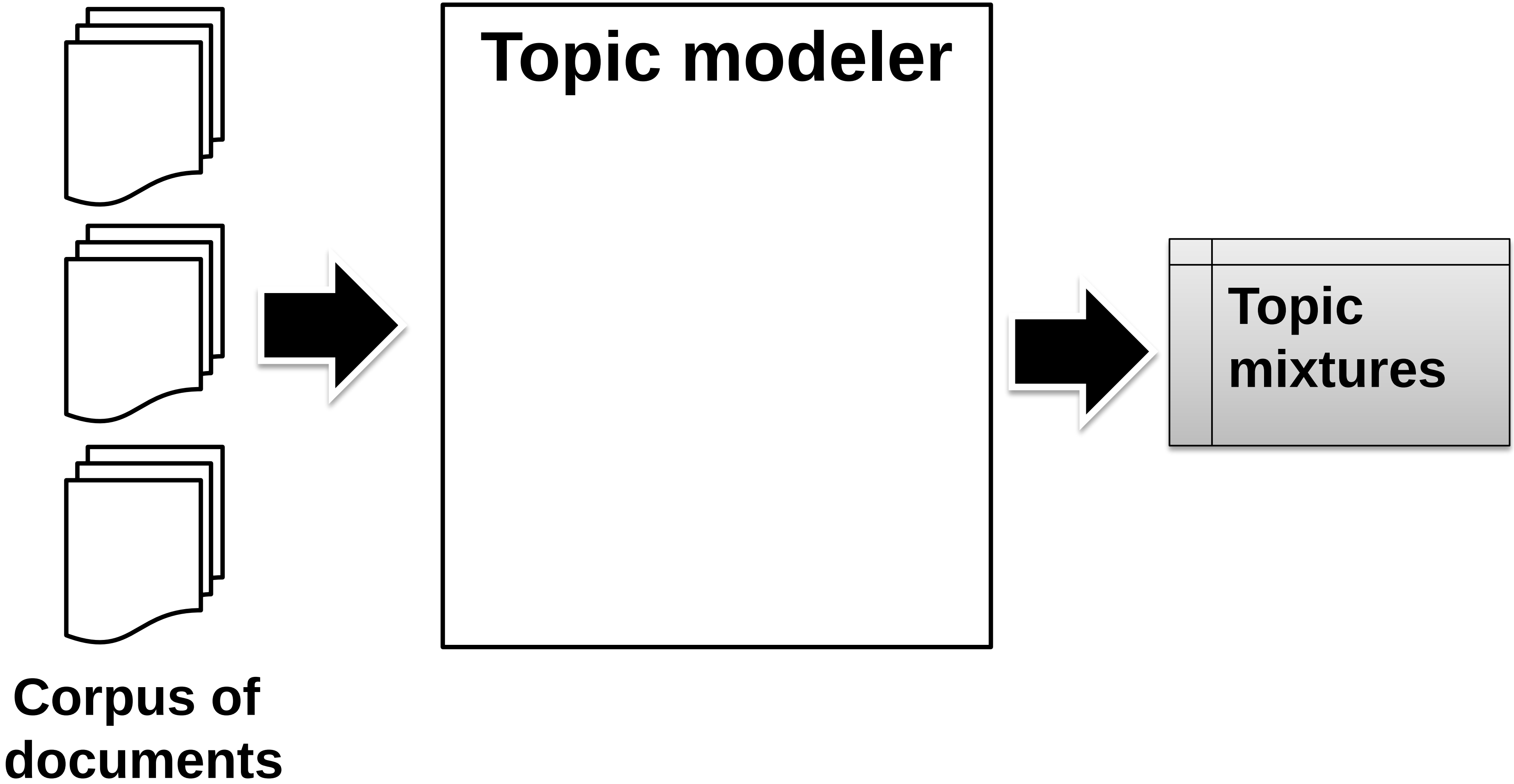
Corpus of documents

Carnegie Mellon
Parallel Data Laboratory

Example: Topic modeling



Example: Topic modeling



LDA topic modeling

- Assign each word in each document to a topic
  - Guided by LDA model and other word assignments
- Continue reassigning until model looks “good”
- Two main data structures
  - Topic-word table
  - Document-topic table

Topic-word table

Number of times a word (in any doc) is associated with a particular topic

	Jobs	**Economy**	**Obama**	**Romney**	**His**	**Says**
Generic	5	1	0	0	51	78
Politics	2	10	105	121	1	2
Finance	231	312	22	3	0	1

Document-topic table

Number of times any word in that document is associated with a topic

	Generic	Politics	Finance
Document 1	40	49	11
Document 2	75	12	13
Document 3	20	4	151

LDA iteration step

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105	2
Finance	231	22	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49	11
Doc. 2	75	12	13
Doc. 3	20	4	151

<u>Document 1</u>

Obama says jobs…

Read document

LDA iteration step

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105	2
Finance	231	22	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49	11
Doc. 2	75	12	13
Doc. 3	20	4	151

Document 1

Obama says jobs…

For each word, look at column of topic-word table

LDA iteration step

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105 -1	2
Finance	231	22 +1	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49 -1	11 +1
Doc. 2	75	12	13
Doc. 3	20	4	151

Document 1
Obama says jobs…

Potentially assign word to different topic

If so, update tables accordingly

LDA iteration step

Topic-word

	Jobs	**Obama**	**Says**
Generic	5	0	78
Politics	2	105 -1	2
Finance	231	22 +1	1

Document-topic

	Gen.	**Pol.**	**Fin.**
Doc. 1	40	49 -1	11 +1
Doc. 2	75	12	13
Doc. 3	20	4	151

<u>Document 1</u>

Obama **says** jobs…

Move on to next word and repeat

Parallelizing LDA

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105	2
Finance	231	22	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49	11
Doc. 2	75	12	13
Doc. 3	20	4	151

Process 1

Process 2

Multiple processes to speed up iteration steps

Doc. 1

Doc. 2

Doc. 3

Doc. 4

Parallelizing LDA

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105	2
Finance	231	22	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49	11
Doc. 2	75	12	13
Doc. 3	20	4	151

Process 1

Process 2

Doc. 1

Doc. 2

Doc. 3

Doc. 4

Assign each document to a particular process

Parallelizing LDA

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105	2
Finance	231	22	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49	11
Doc. 2	75	12	13
Doc. 3	20	4	151

Process 1

Process 2

Process can "own" rows of doc-topic table

Doc. 1

Doc. 2

Doc. 3

Doc. 4

Parallelizing LDA

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Politics	2	105	2
Finance	231	22	1

Document-topic

	Gen.	Pol.	Fin.
Doc. 1	40	49	11
Doc. 2	75	12	13
Doc. 3	20	4	151

Process 1

Process 2

But topic-word table is shared by all processes

Doc. 1

Doc. 2

Doc. 3

Doc. 4

Carnegie Mellon
Parallel Data Laboratory

Parallelizing LDA

Topic-word

	Jobs	Obama	Says
Generic	5	0	78
Polit[illegible]	[illegible]	[illegible]	[illegible]
Fina[illegible]	[illegible]	[illegible]	[illegible]

Doc[illegible]	[illegible]	[illegible]	[illegible]
Doc. 2	75	12	13
Doc. 3	20	4	151

Process 1

Process 2

Doc. 1

[illegible] 2

[illegible] 3

Doc. 4

Performance of algorithm depends on performance of topic-word table!

Other algorithms

- Coordinate descent
  - Finding points in multidimensional space
  - Each process updates subset of coordinates
  - Must read updates from other threads
- K-means
  - Grouping points by location
  - Processes update subset of points…
  - Based on shared grouping information

(Brief) related work

- GraphLab represents intermediate state as graph
  - Each node has local state, update function
  - When neighbor state changes, call update function
  - Works well when variable interactions are local
- Spark stores large tables in memory
  - Tables are updated via bulk operations
  - Keep log of operations for fault tolerance
  - Replace entire data set at once, not point updates
- Piccolo provides distributed table of values

Table API (Piccolo, LazyTables)

- Basic operations:
  - `read, read_row, put`

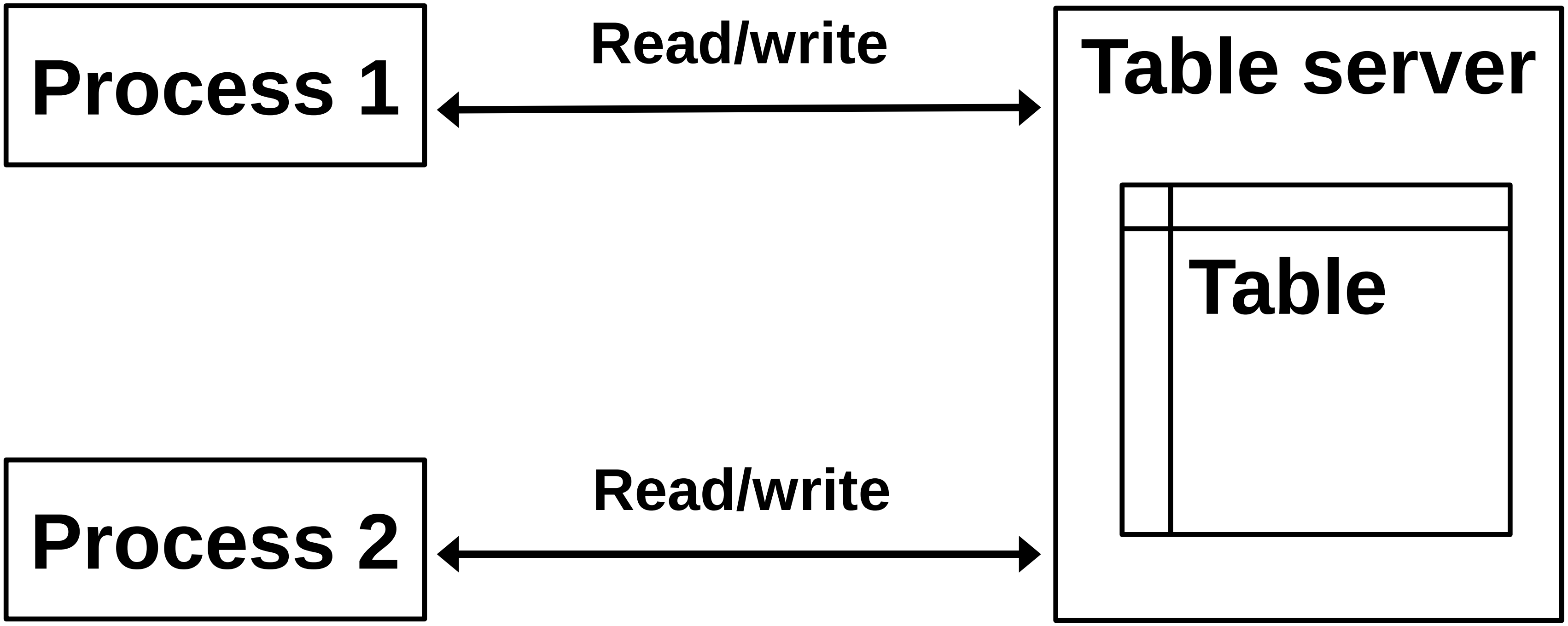
- Table can use one self-commutative update:

`increment(row, col, val)`	`table[row, col] += val`
`multiply(row, col, val)`	`table[row, col] *= val`
`update(row, col, val, f)`	`table[row,col] = f(table[row, col], val)`

Outline

- Insights from LazyBase
- Machine learning applications
- LazyTables design
- Future research

System diagram



Design overview

- **Problem**: frequent reads and writes to shared data
  - Dominate performance of algorithm
  - Need very low latency

Insights from LazyBase

- Improving performance can cause staleness
- Many applications tolerate data staleness
- Freshness requirements are important
  - Property of query, not data
  - Can change over time
  - Make them explicit, not implied

Insights from LazyBase

- Improving performance can cause staleness
- Many applications tolerate data staleness
- Freshness requirements are important
  - Property of query, not data
  - Can change over time
  - Make them explicit, not implied

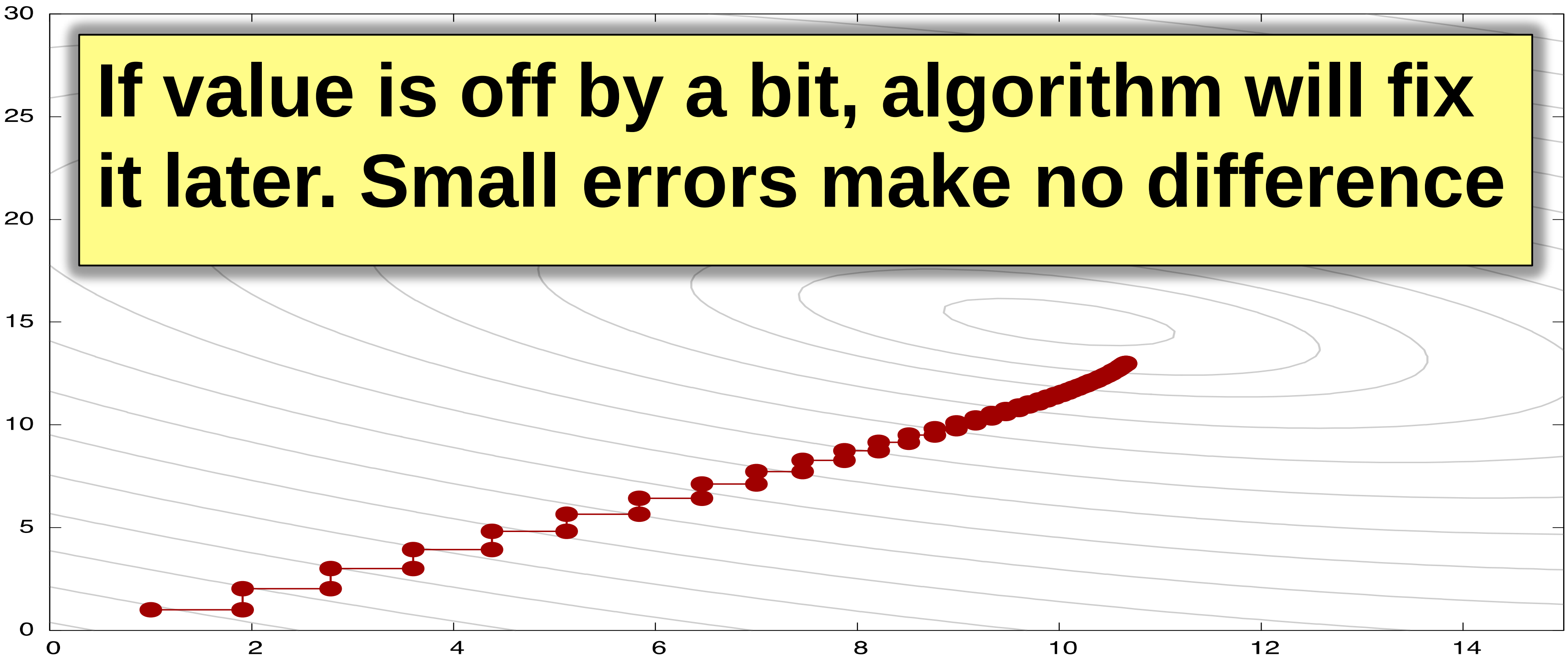
ML algorithms tolerate staleness

- Algorithms are convergent
  - Start with “bad” solution
  - Iteratively improve solution
  - Eventually converge on “good” solution

- If they get thrown off, they can just continue

- Example: coordinate descent
  - Finding minimum point in space

ML algorithms tolerate “errors”

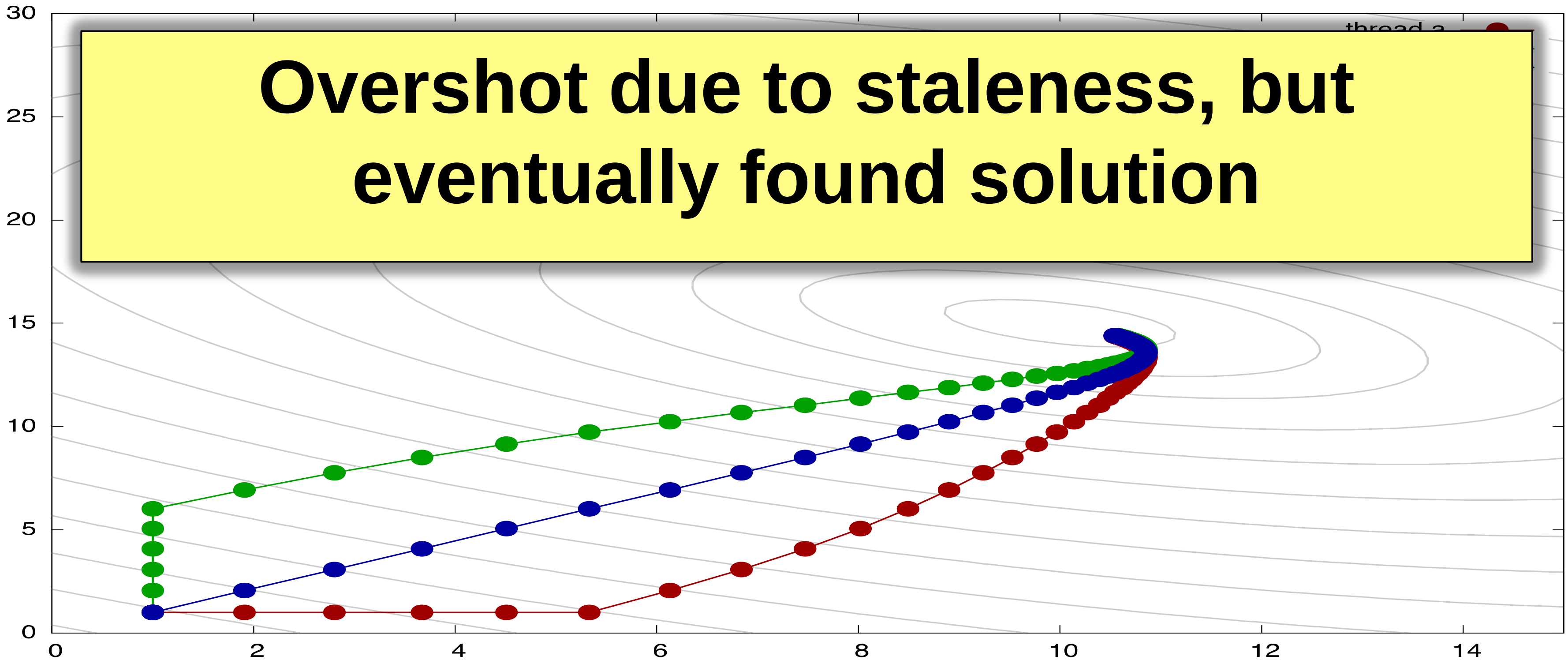


- Starts with initial guess, iteratively improves
- Eventually converges to “correct” result

Coord. Descent and staleness

- Simulated coordinate descent with stale data
- Two processes, updating X and Y respectively
- Take 5 iterations to propagate between processes

ML algorithms tolerate staleness



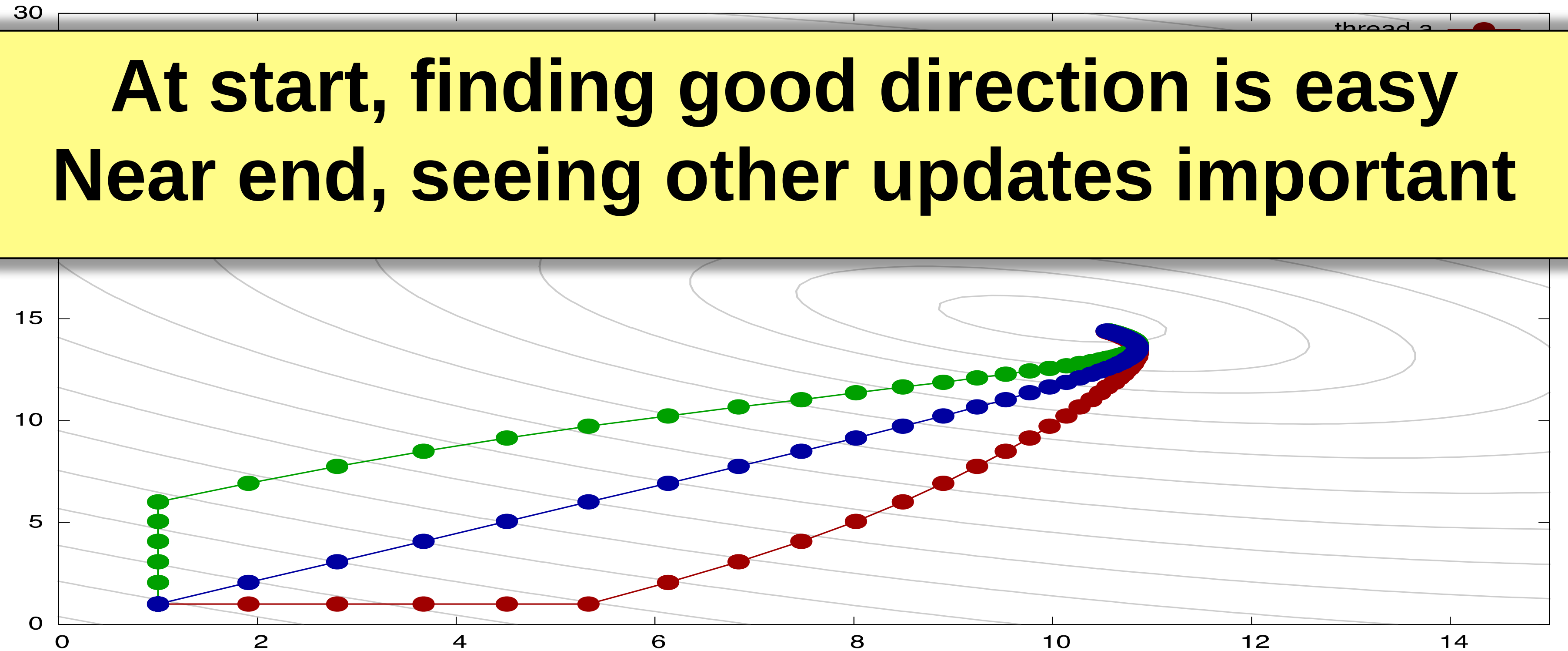
- Processes don’t get updates immediately
- Shared state converges to correct result

Insights from LazyBase

- Improving performance can cause staleness
- Many applications tolerate data staleness
- Freshness requirements are important
  - Property of query, not data
  - Can change over time
  - Make them explicit, not implied

ML algorithms tolerate staleness

At start, finding good direction is easy
Near end, seeing other updates important



- Processes don't get updates immediately
- Shared state converges to correct result

Carnegie Mellon
Parallel Data Laboratory

Specifying freshness

- Each read operation specifies requirement
  - E.g. "read row 12 with all updates as of iteration 5"

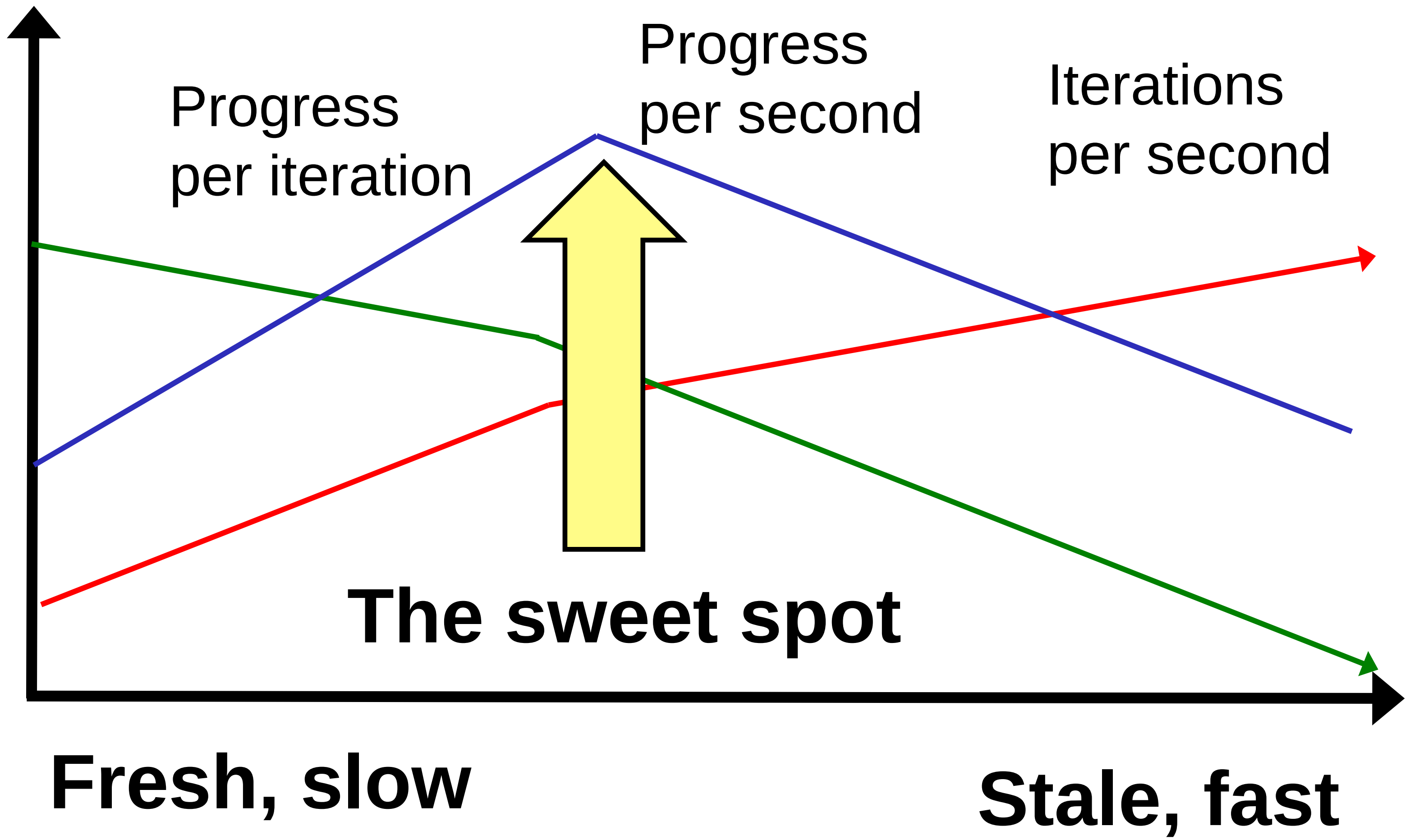
- If data from all processes is ready, return
- Otherwise wait for other processes to update

- Requires fresher data → may wait longer

Is stale data really a win?

- Stale data can slow down convergence
  - Could mean more iterations required to finish
- …but each iteration is much faster
- Likely a "sweet spot" in freshness requirement
  - Could depend on input data, algorithm progression…

Freshness/latency sweet spot



Insights from LazyBase

- Improving performance can cause staleness

- Many applications tolerate data staleness

- Freshness requirements are important
  - Property of query, not data
  - Can change over time
  - Make them explicit, not implied

Design overview

- **Problem**: frequent reads and writes to shared data
  - Dominate performance of algorithm
  - Need very low latency

- **Read solution**: Caching
  - Reads exhibit locality (set of words in doc. constant)

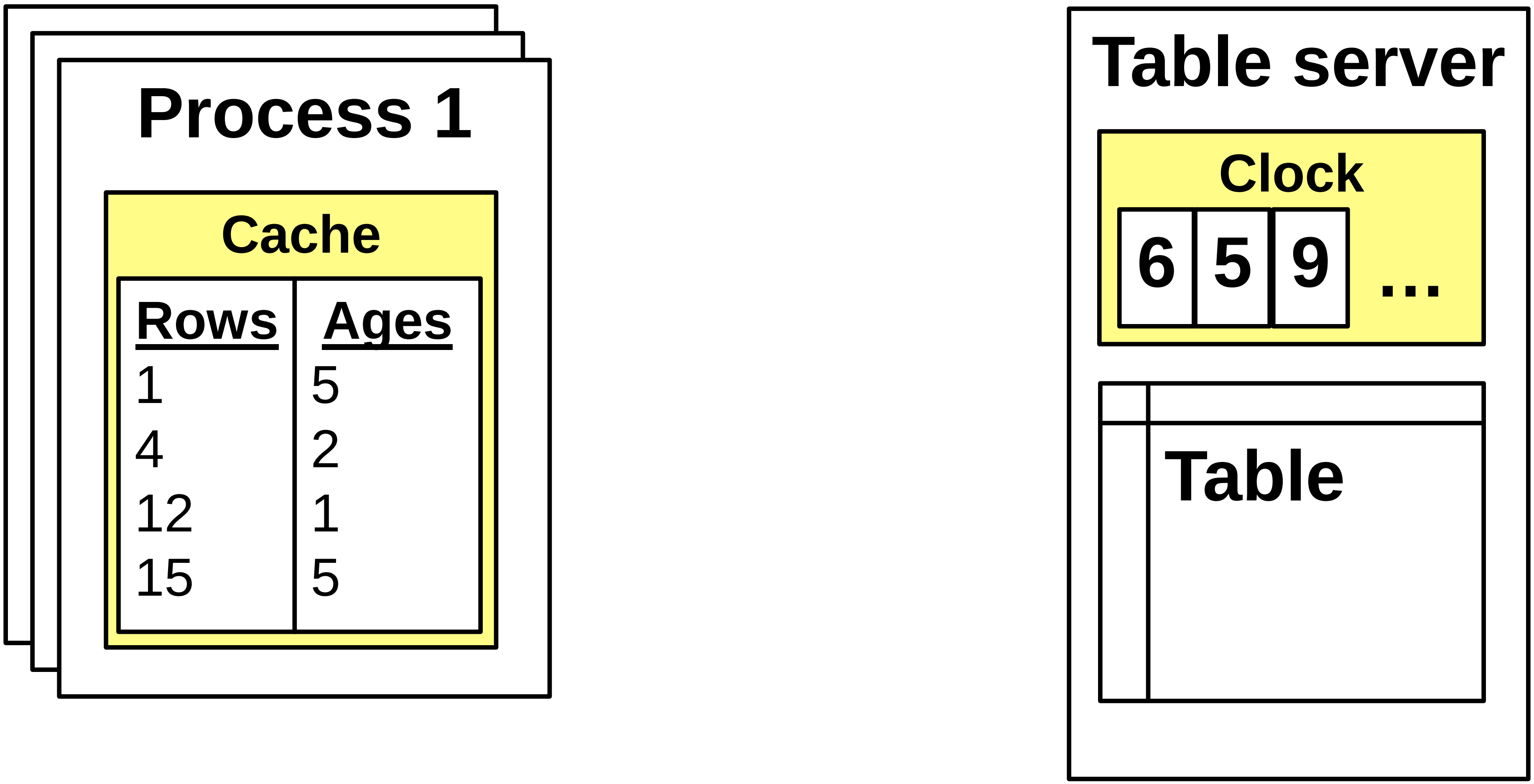
Carnegie Mellon
Parallel Data Laboratory

Cache requires 2 data structures

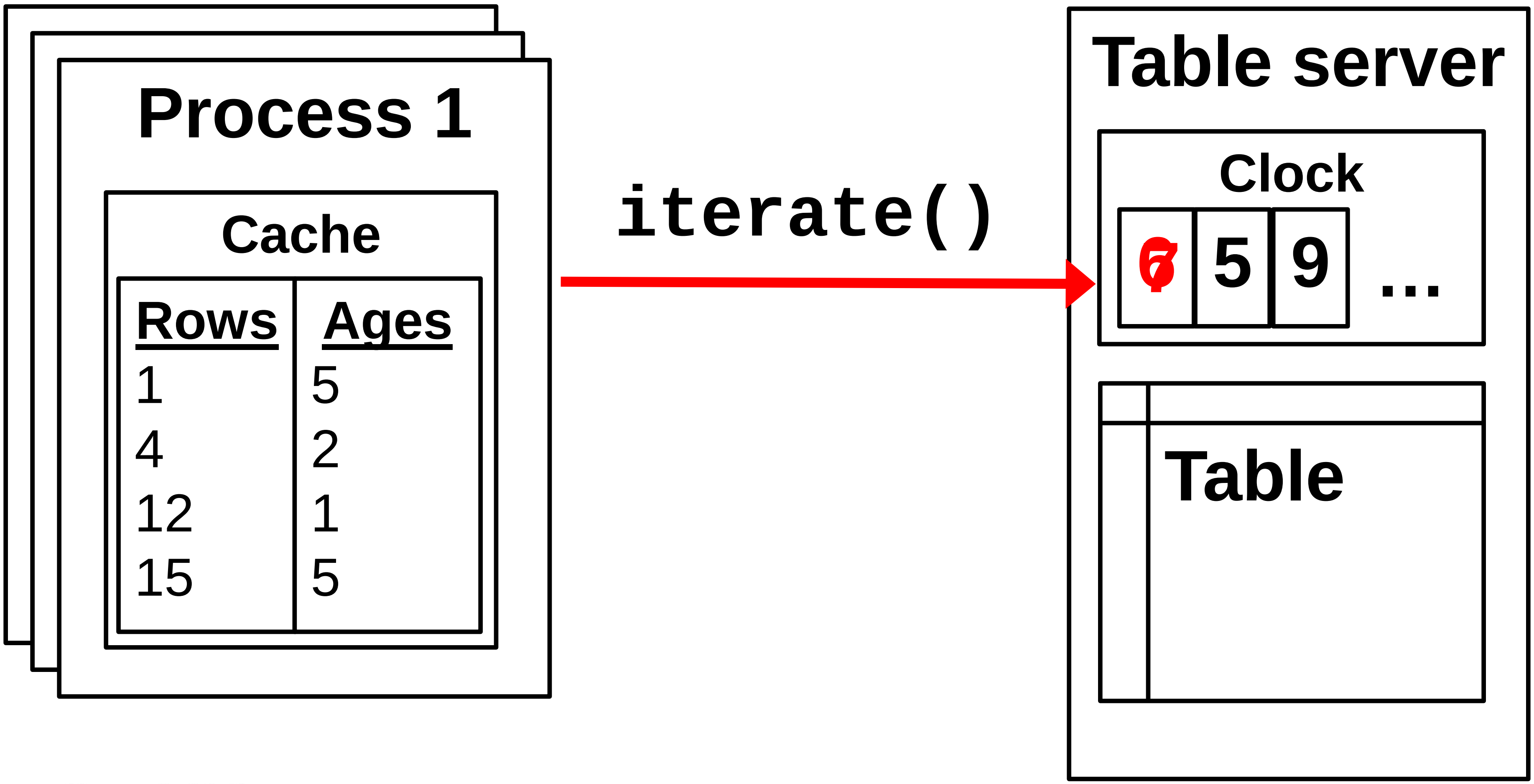
- Per-process cache of table rows
  - Each row tagged with age of row
  - When reading, check age
  - **Too old ➔ freshness miss, re-read row**

- Vector clock in table server
  - Track what iteration each process is on
  - On read, age of data is minimum value in clock
  - `iterate()` operation increments clock for a process

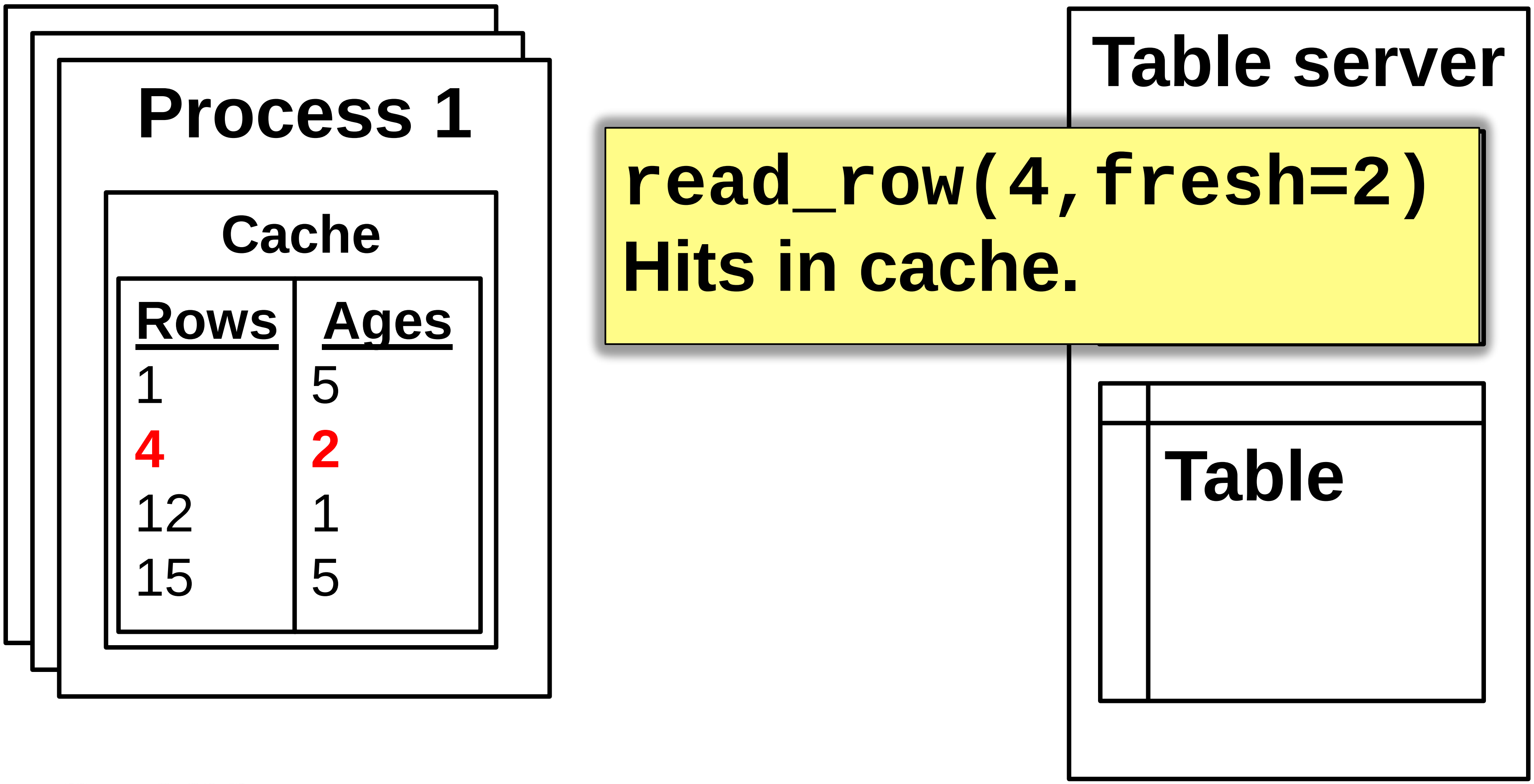
Adding a cache



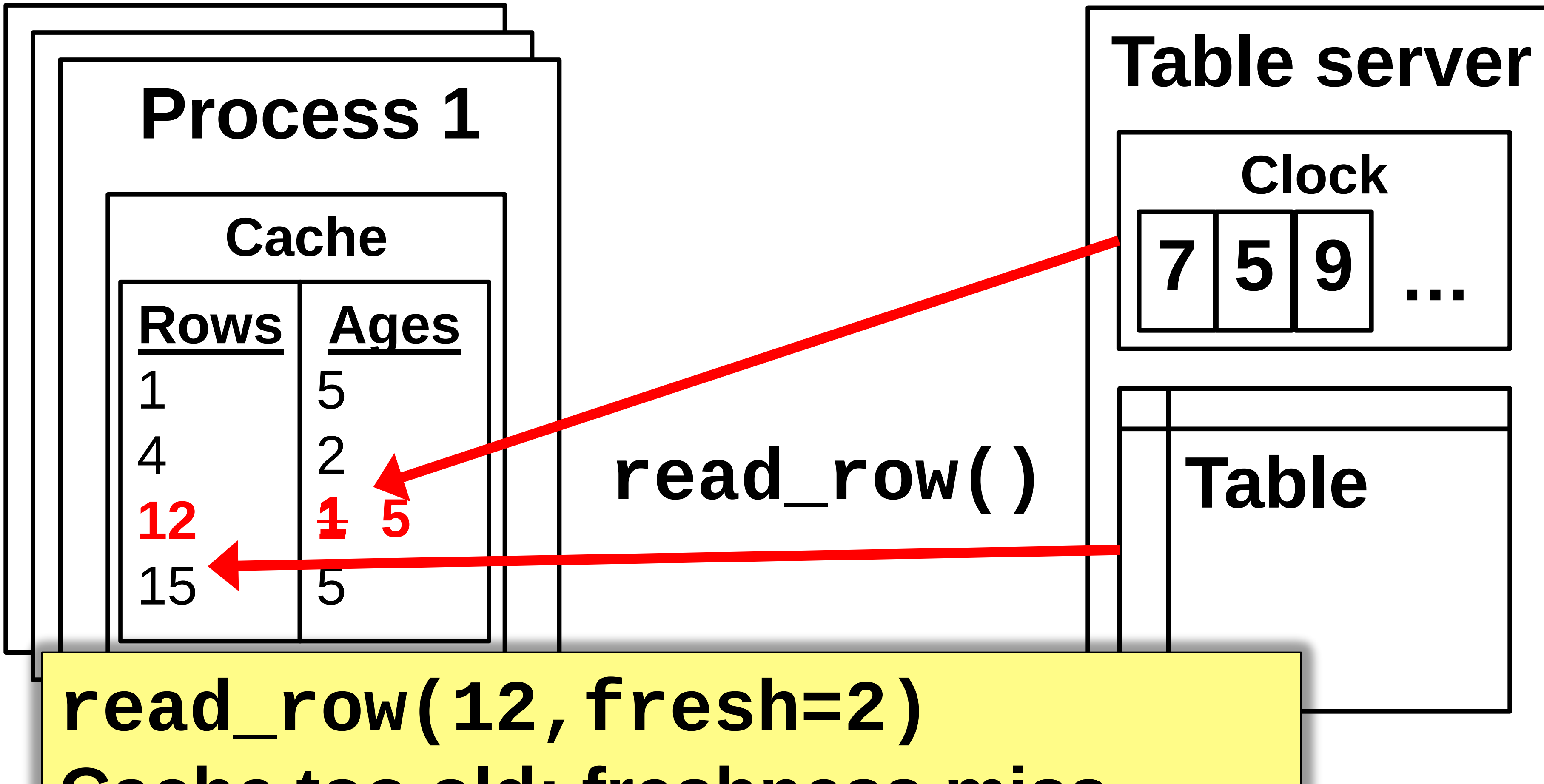
Adding a cache



Adding a cache



Adding a cache



`read_row(12,fresh=2)`

Cache too old: freshness miss

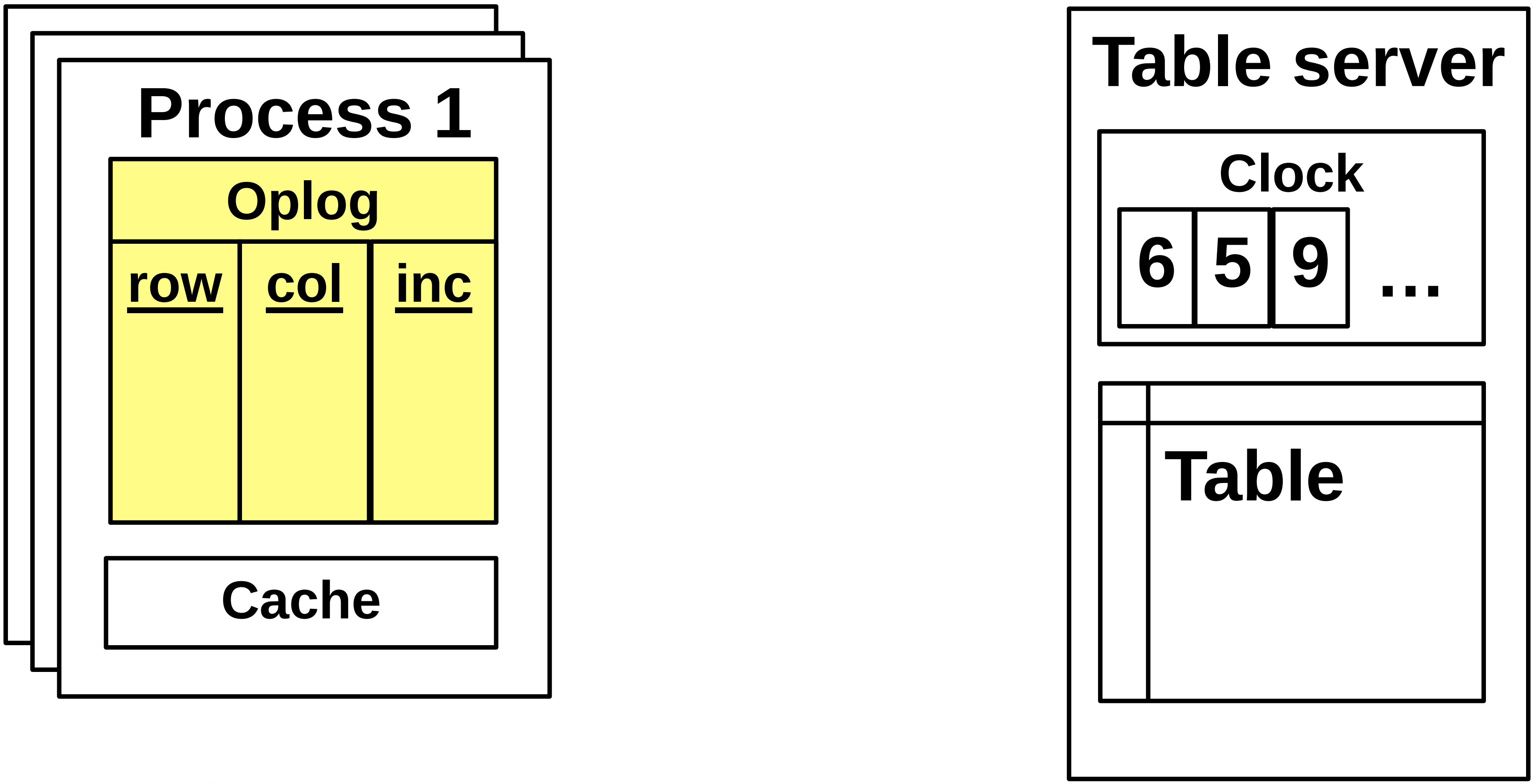
Design overview

- **Problem**: frequent reads and writes to shared data
  - Dominate performance of algorithm
  - Need very low latency
- **Read solution**: Caching
  - Reads exhibit locality (set of words in doc. constant)
- **Write solution:** Operation logging
  - Batch many updates and apply at once

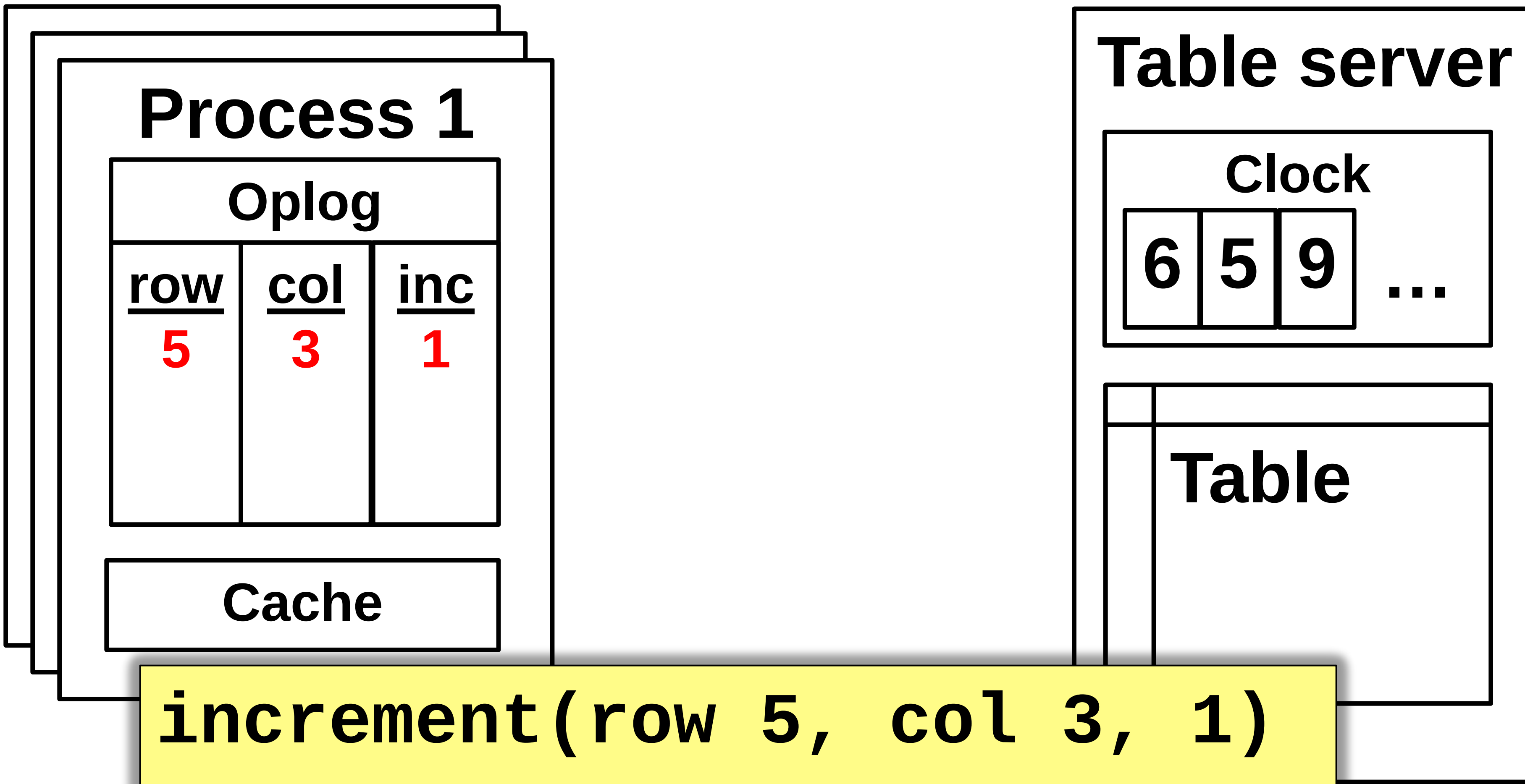
Oplog data structure

- Log of update operations, not values
  - E.g. “add one to row 5, column 2”
- Batch many operations at process
- Send batch on `iterate()` call

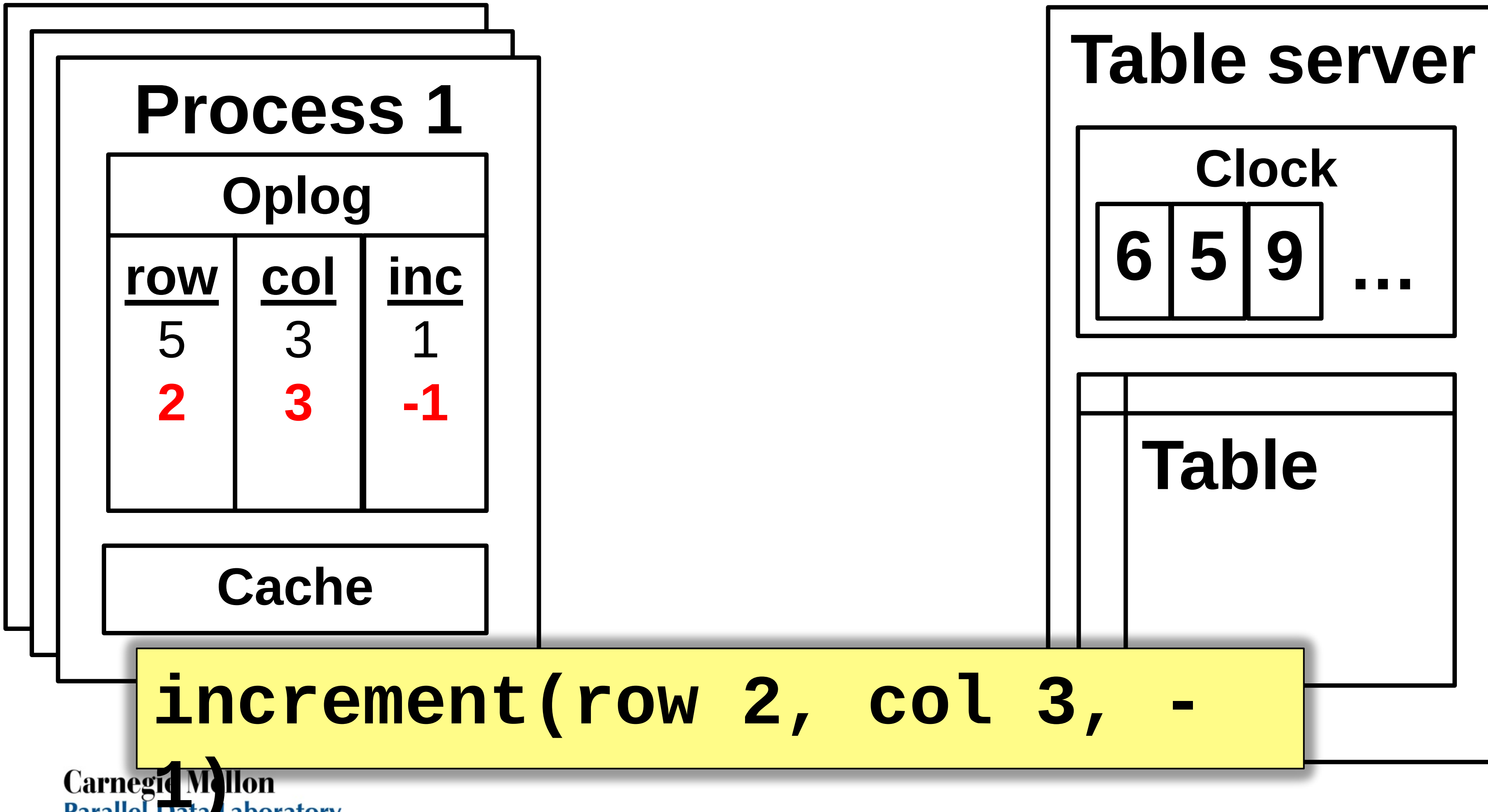
Adding an oplog



Adding an oplog

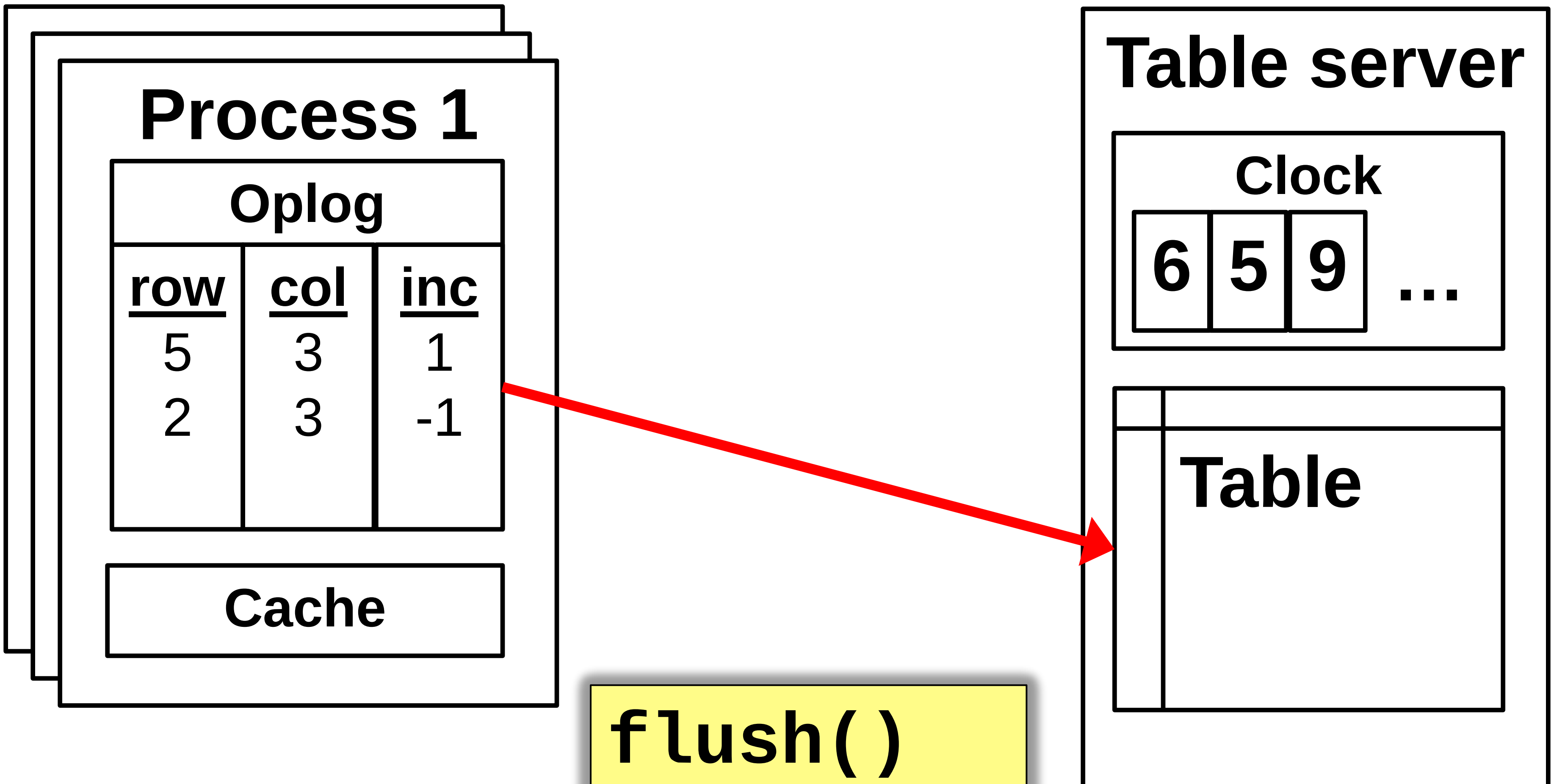


Adding an oplog



Carnegie Mellon
Parallel Data Laboratory

Adding an oplog

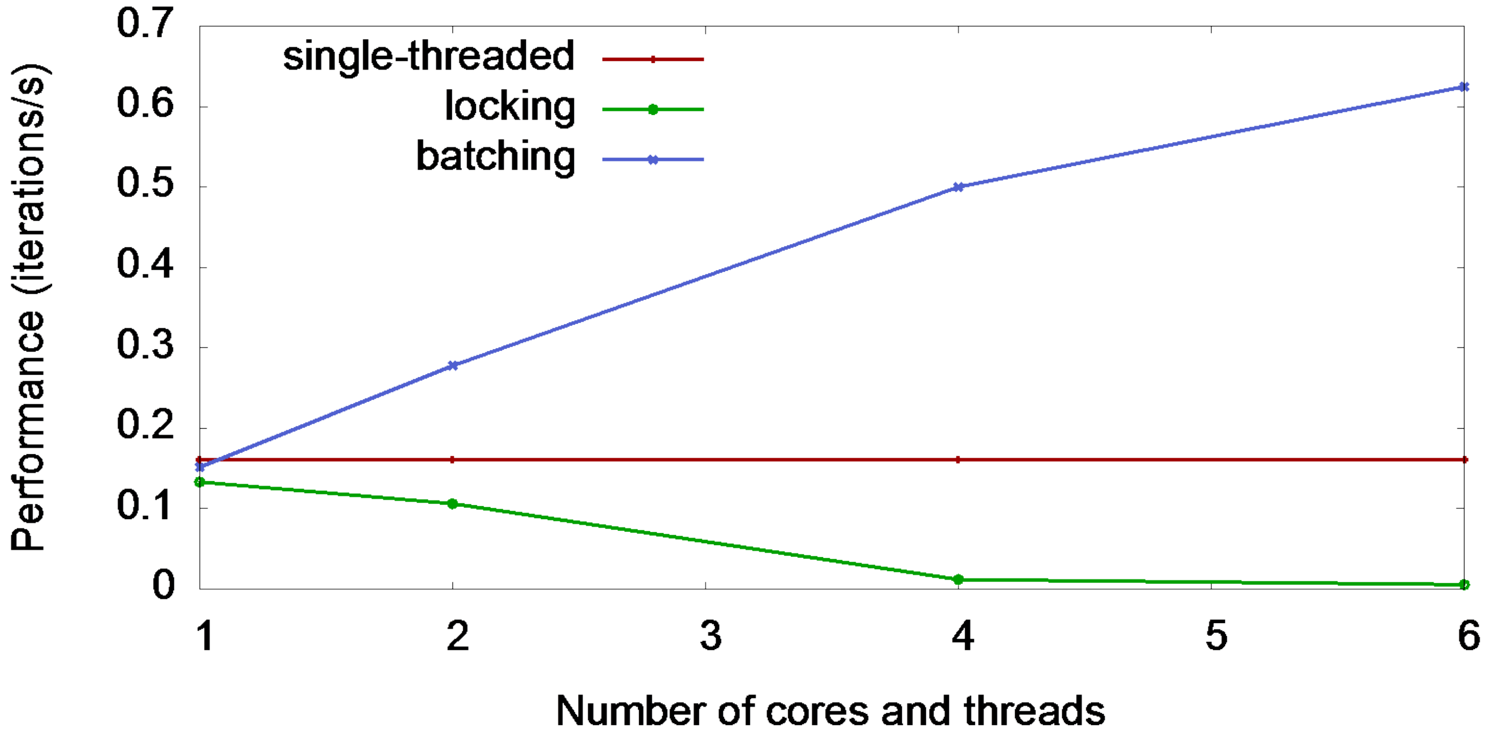



```
flush()
```

Initial experiments

- Simple C++ table implementation
  - Based on STL `map<>` data structure
  - Get/put, increment/decrement, multiply
- Basic implementation: reader/writer locks
- Lazy implementation
  - Queue updates in thread-local storage
  - After 1k updates - or `flush()` - perform bulk update
- Used actual document classification code
  - Latent Dirichlet Allocation algorithm
  - Similar in behavior to coordinate descent

Initial results



Batching updates improves performance

Locking too expensive for every update

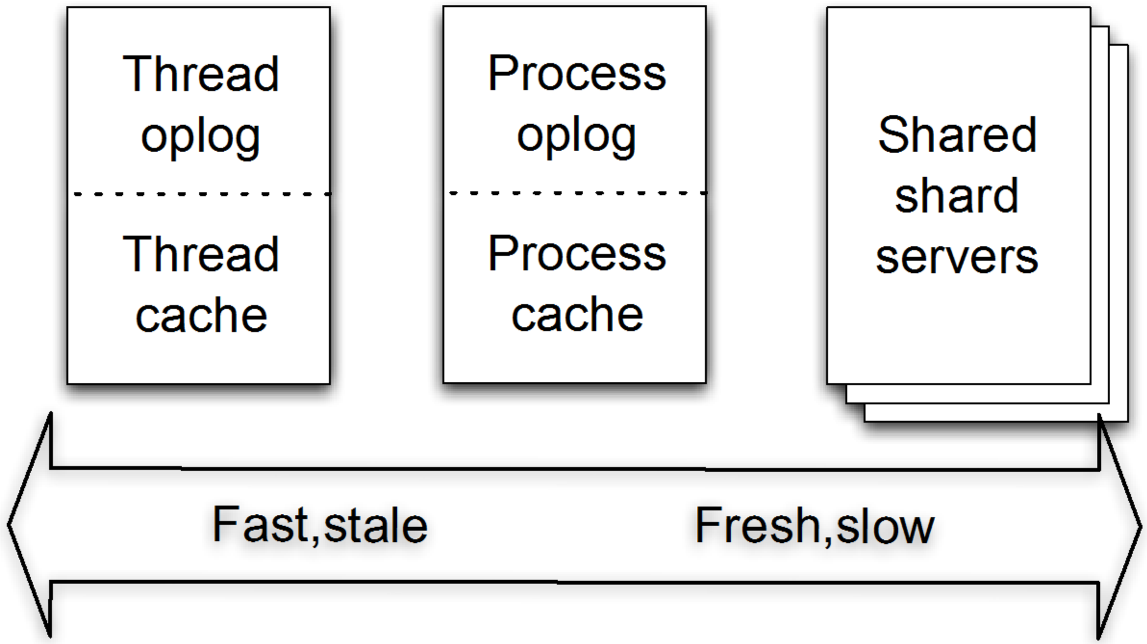
Outline

- Insights from LazyBase
- Machine learning applications
- LazyTables design
- Future research

Which algorithms can benefit?

- Does staleness affect some applications more?
- Differences in update rate
  - Little benefit to lazy writes
- Differences in freshness requirements
  - Lazy writes could be too costly

Freshness/latency tradeoff



Layers of cache provide tradeoff between freshness of data and latency of reads

Conclusions

- LazyTables: shared intermediate state for ML
  - High-throughput updates
- Improve performance by allowing stale data
  - Extensive use of batching and caching
- Make freshness requirements explicit
  - Different requirements for each read operation

References

- Apache Mahout, http://mahout.apache.org.
- D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation.
- J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization.
- J. Cipar, G. Ganger, K. Keeton, C. B. Morrey, III, C. A. Soules, and A. Veitch. LazyBase: trading freshness for performance in a scalable database.
- Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning.
- Y. Low, G. Joseph, K. Aapo, D. Bickson, C. Guestrin, and M. Hellerstein, Joseph. Distributed GraphLab: A framework for machine learning and data mining in the cloud.
- R. Power and J. Li. Piccolo: building fast, distributed programs with partitioned tables.
- M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster comput- ing.