Social Network Analysis on Twitter with Chapel

Michael Ferguson, Ben Albrecht
Cray Inc,
CHIUW
May 27, 2016

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Talk Outline

- Introduce the label propagation benchmark
- Describe a Chapel version
- Show one part of the benchmark and improvements to it
- Compare with the Spark version (*)

Processing Tweets: Background

Twitter: an online social networking service that enables users to send and read short 140-character messages called "tweets" -- Wikipedia

- tweets support mentioning other users via @*username*

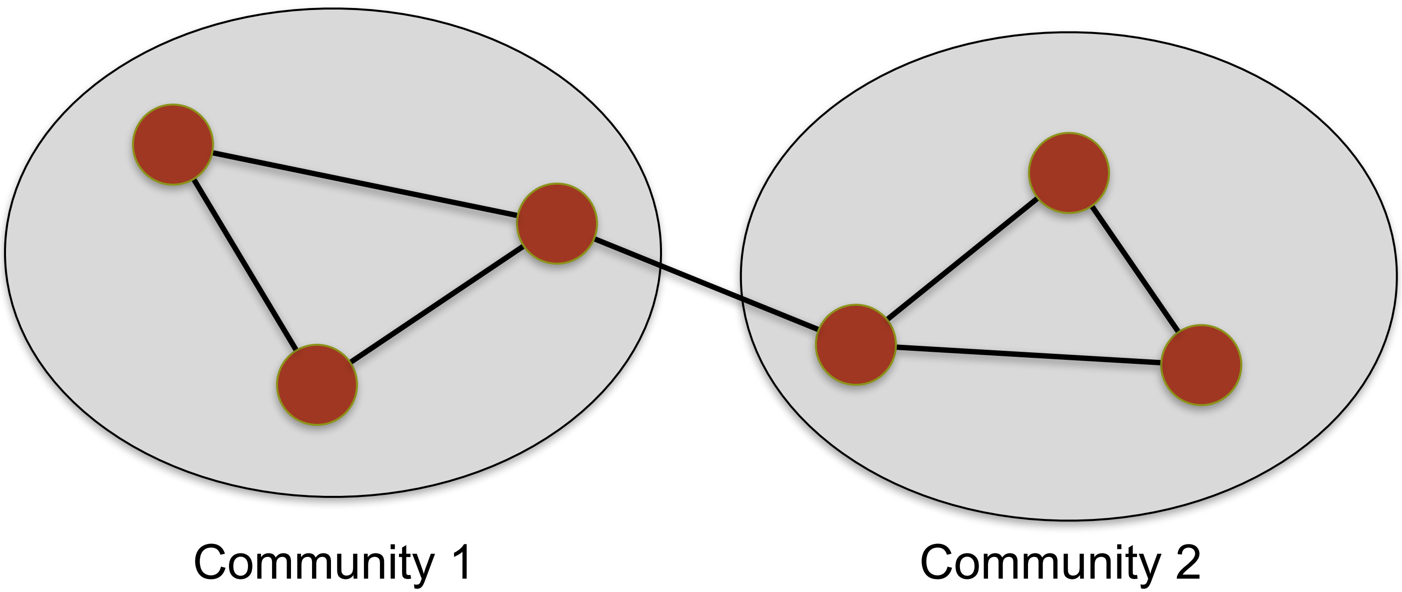
Benchmark: Label Propagation for Community Detection

- A form of data analytics - a hot topic in big data
- Identifies communities of users
- Useful for advertising or bot detection

- see CUG'15 paper: *Implementing a social-network analytics pipeline using Spark on Urika XA*

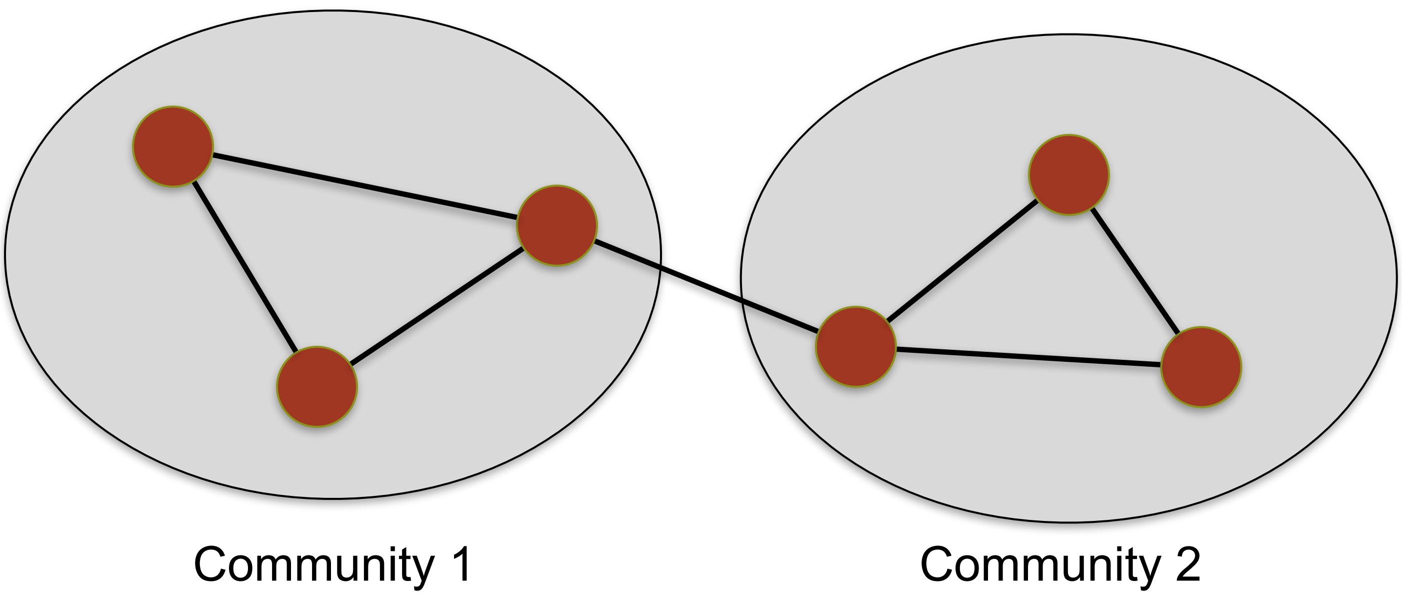
Processing Tweets: Computation Steps

- **Computation consists of these steps:**
  - Read in gzip files storing JSON-encoded tweets
  - Find pairs of Twitter users that @mention each other
  - Construct a graph from such users
  - Run a label propagation algorithm on that graph
  - Output the community structure resulting from label propagation



Processing Tweets: Computation Steps

- Computation consists of these steps:
  - Read in gzip files storing JSON-encoded tweets
  - Find pairs of Twitter users that @mention each other
  - Construct a graph from such users
  - Run a label propagation algorithm on that graph
  - Output the community structure resulting from label propagation



Processing Tweets: First Part

- Files are processed in a forall loop
  - Reads file using `gunzip` via the new Spawn module
  - Uses new functionality for parsing JSON
    - concept: use types and I/O that ignore irrelevant fields
- Constructs distributed associative domain to find mutual mentions

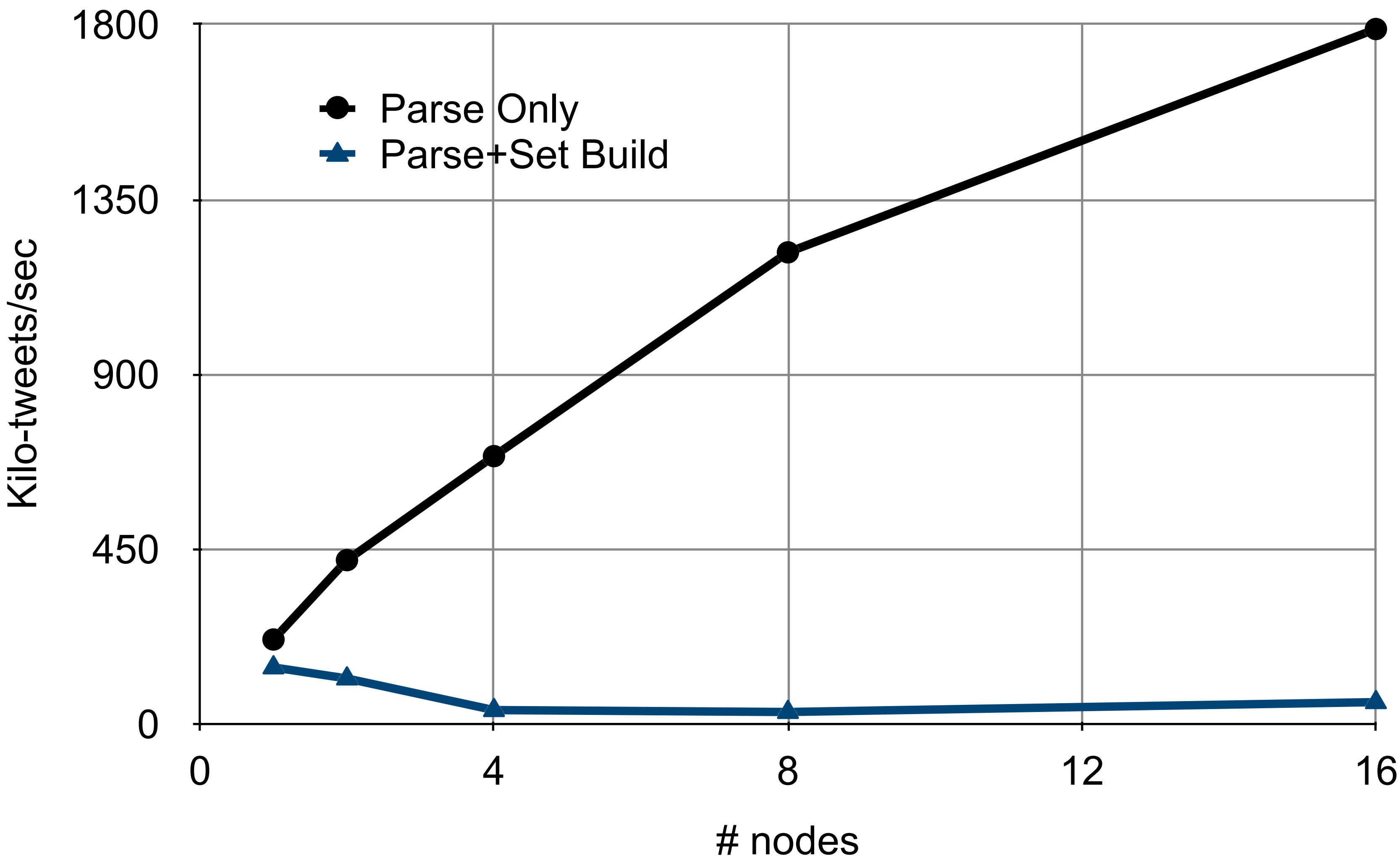
First Part Kernel

```
var Pairs: domain( (int, int) ) dmapped new UserMapAssoc(...);

forall logfile in distributedFiles() {
  while logfile.readf("%~jt", tweet) {
    var id = tweet.user.id;
    for mentions in tweet.entities.user_mentions {
      var other_id = mentions.id;
      if max_id < other_id then max_id = other_id;
      // Add (id, other_id) to Pairs,
      // but leave out self-mentions
      if id != other_id {
        Pairs += (id, other_id);
      }
    }
  }
}
```

First Part Kernel

```
var Pairs: domain( (int, int) ) dmapped new UserMapAssoc(...);

forall logfile in distributedFiles() {
  while logfile.readf("%~jt", tweet) {
    var id = tweet.user.id;
    for mentions in tweet.entities.user_mentions {
      var other_id = mentions.id;
      if max_id < other_id then max_id = other_id;
      // Add (id, other_id) to Pairs,
      // but leave out self-mentions
      if id != other_id {
        Pairs += (id, other_id);
      }
    }
  }
}
```

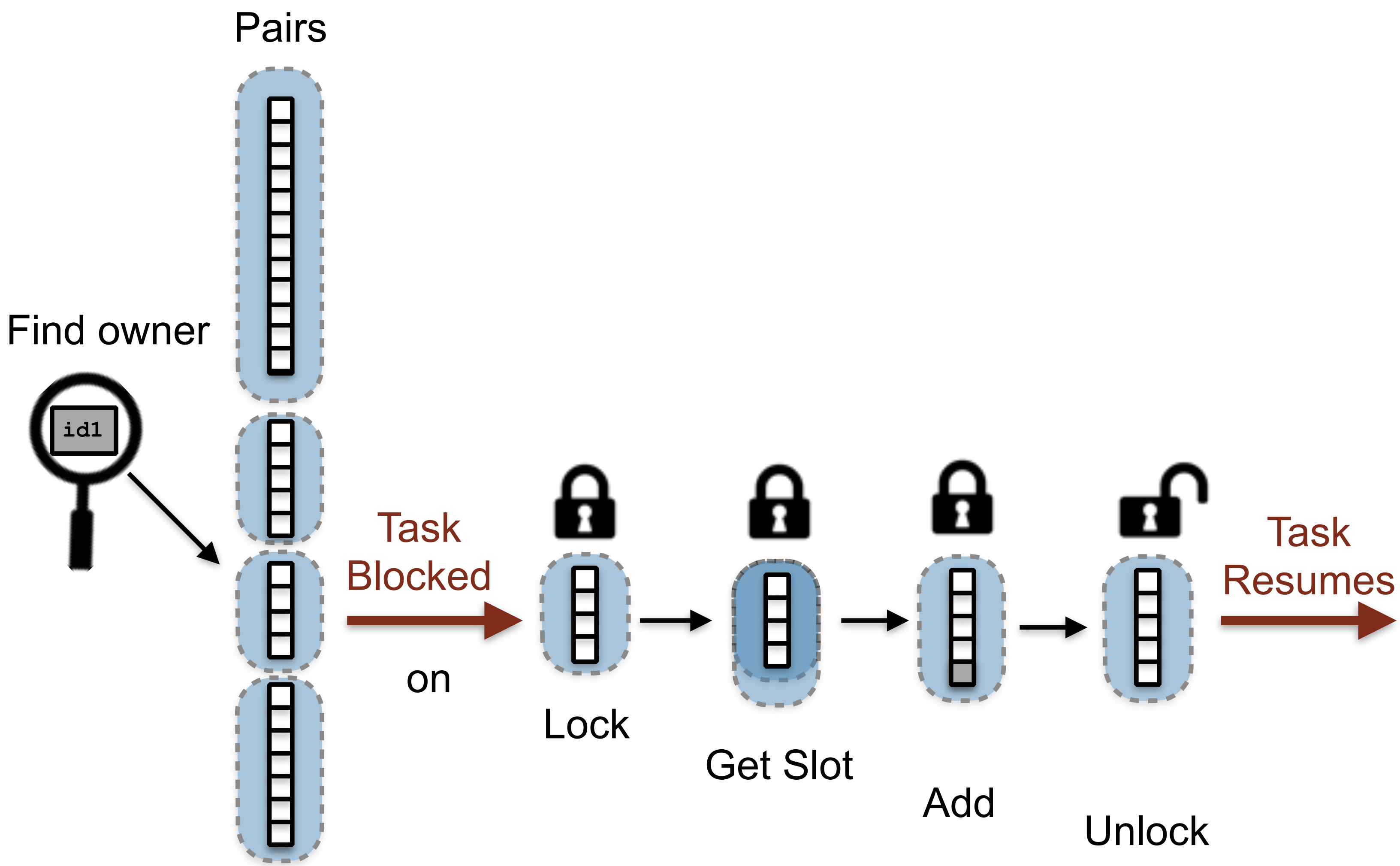
First Part Scalability



384 files, gasnet, fifo, gnu, XC30 24 cores/locale, fe29555c

Steps in Set Addition

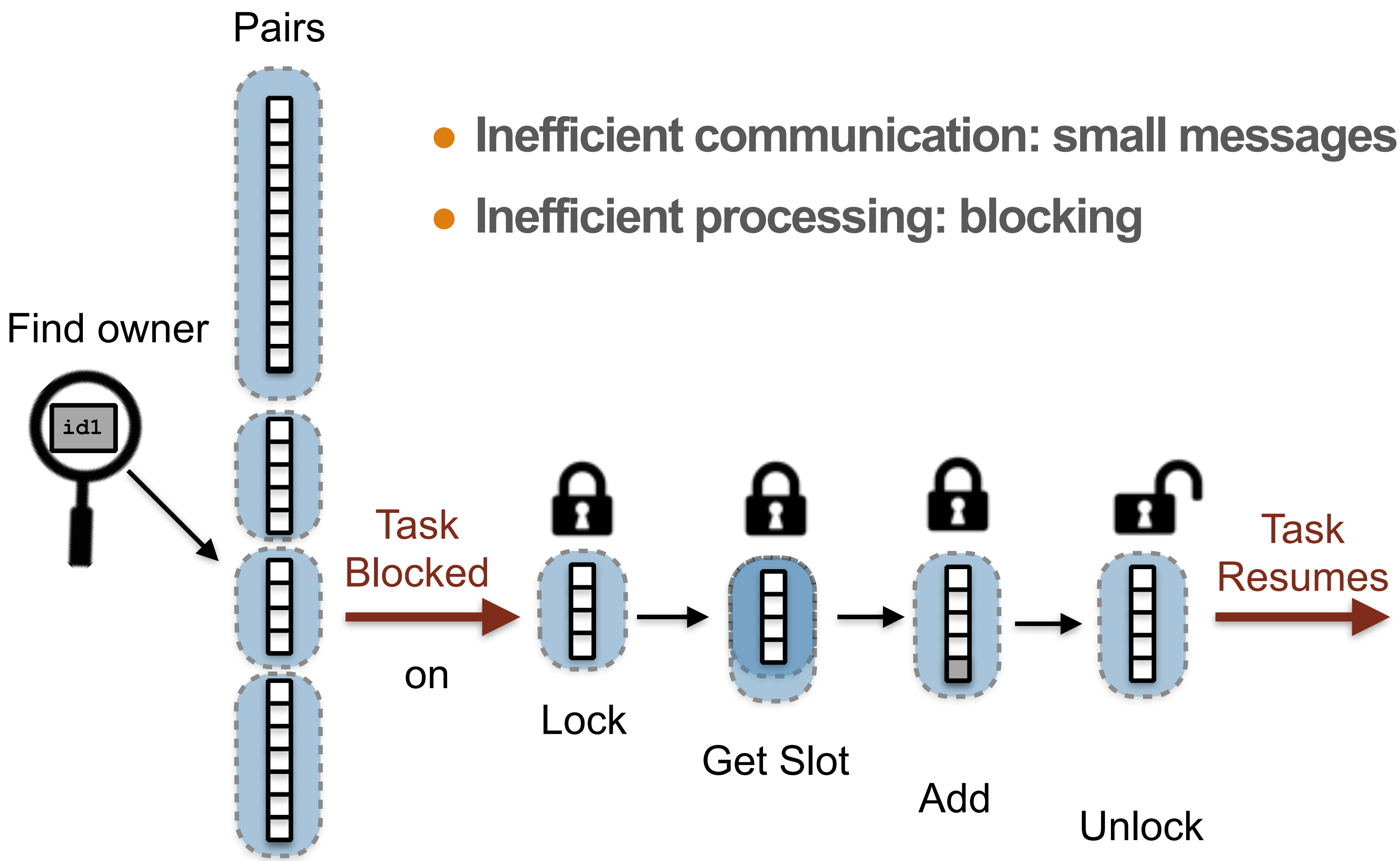
```
Pairs += (id1, other1);
```



Steps in Set Addition

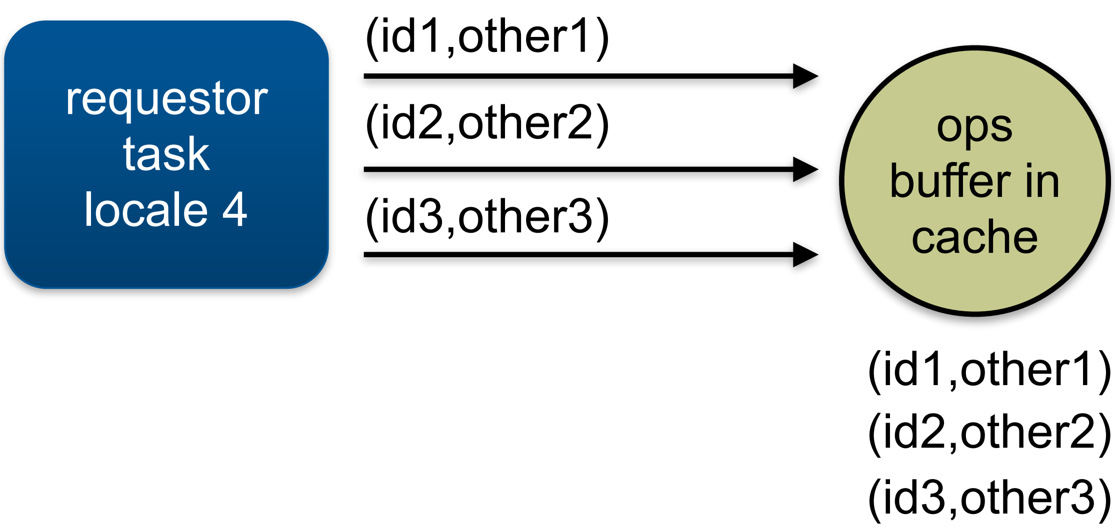
```
Pairs += (id1, other1);
```

- Inefficient communication: small messages
- Inefficient processing: blocking



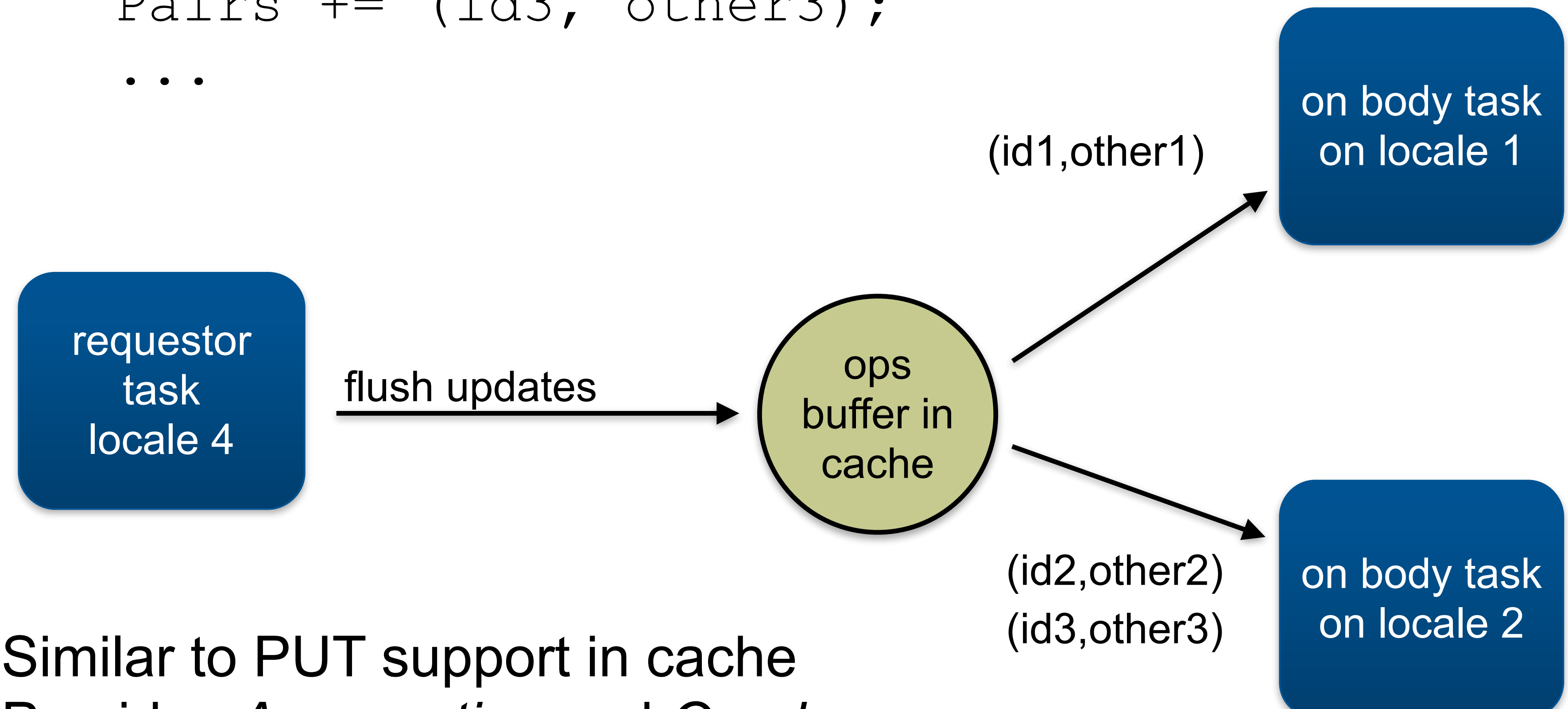
Operations Buffer Makes this Code Faster

```
Pairs += (id1, other1);
Pairs += (id2, other2);
Pairs += (id3, other3);
...
```



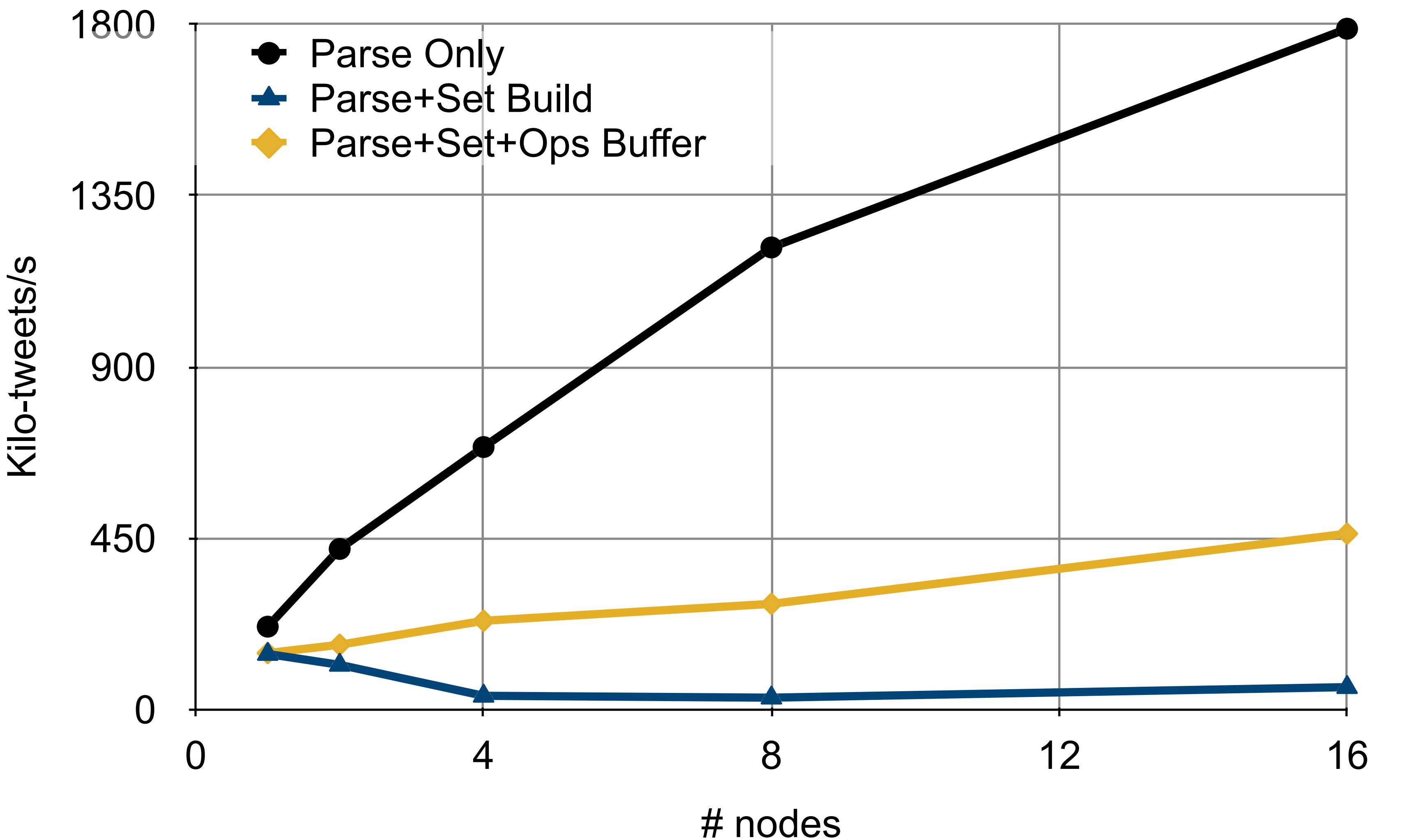
Operations Buffer Makes this Code Faster

```
Pairs += (id1, other1);
Pairs += (id2, other2);
Pairs += (id3, other3);
...
```



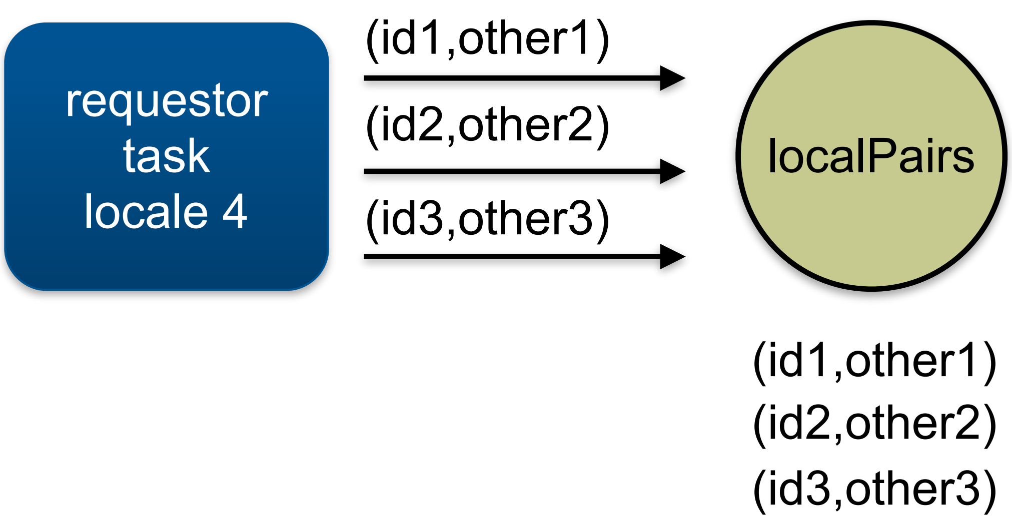
Similar to PUT support in cache
Provides *Aggregation* and *Overlap*

First Part Scalability with Operations Buffer



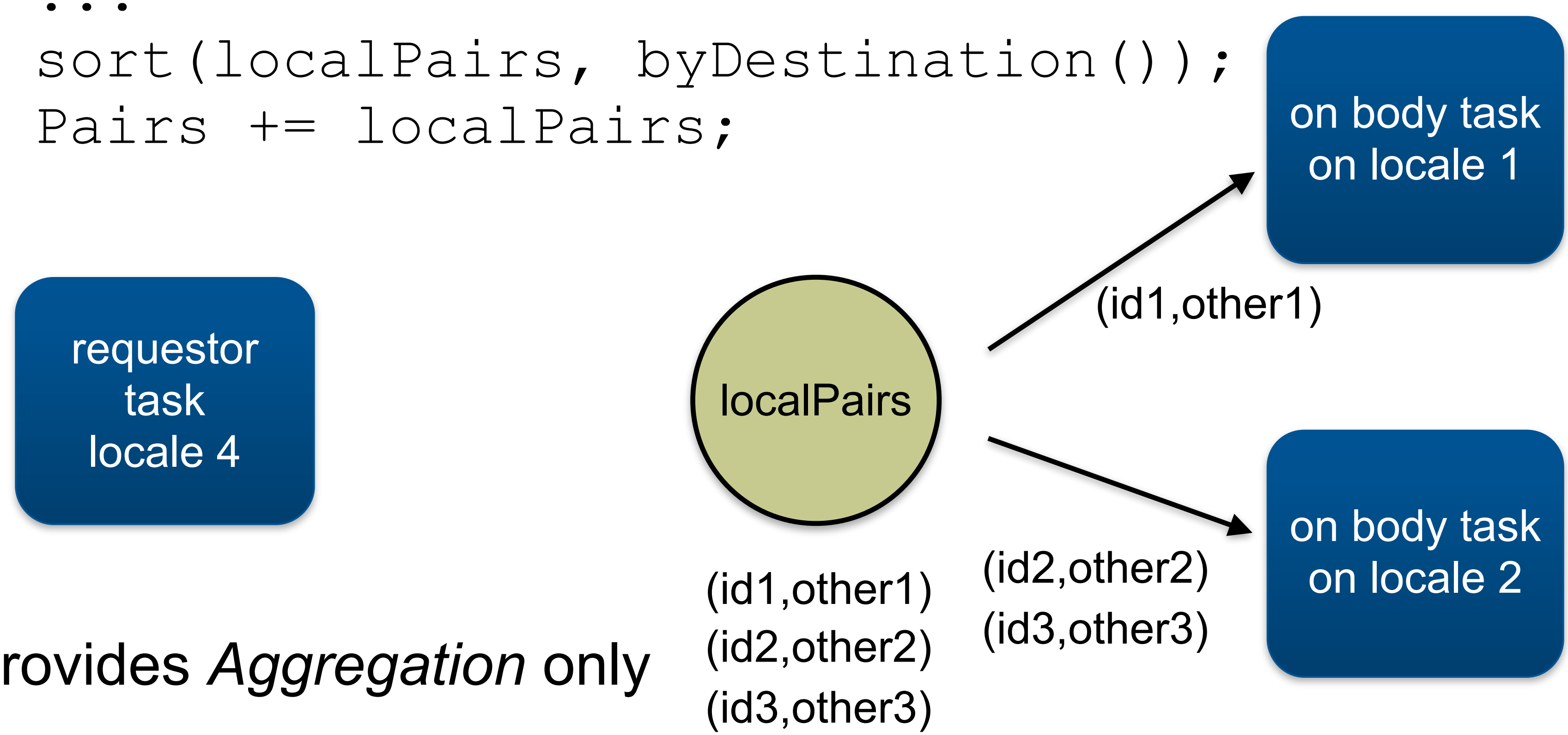
Bulk Addition is a Manual Alternative

```
localPairs.push_back((id1, other1));
localPairs.push_back((id2, other2));
localPairs.push_back((id3, other3));
...
sort(localPairs, byDestination());
Pairs += localPairs;
```



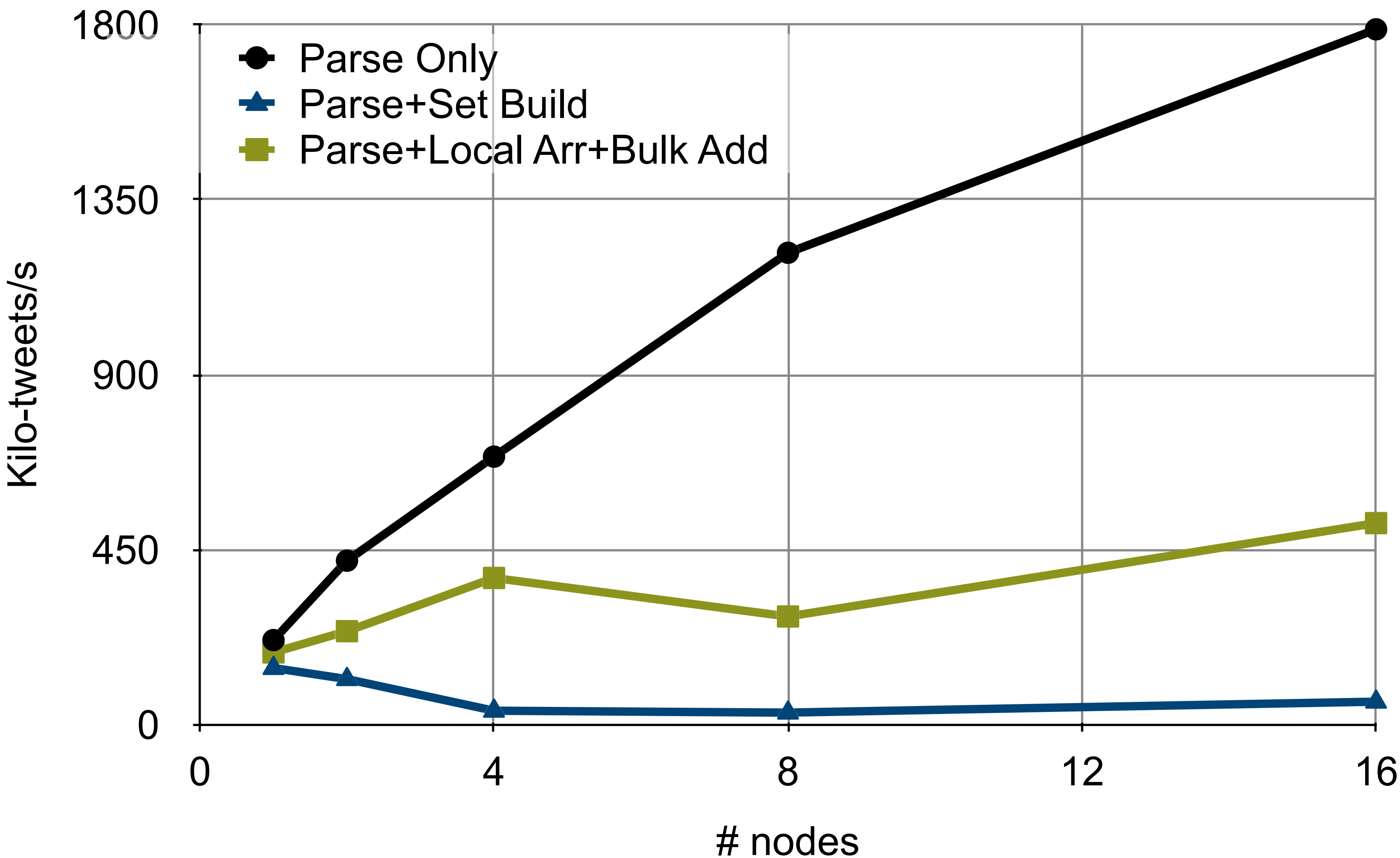
Bulk Addition is a Manual Alternative

```
localPairs.push_back((id1, other1));
localPairs.push_back((id2, other2));
localPairs.push_back((id3, other3));
...
sort(localPairs, byDestination());
Pairs += localPairs;
```

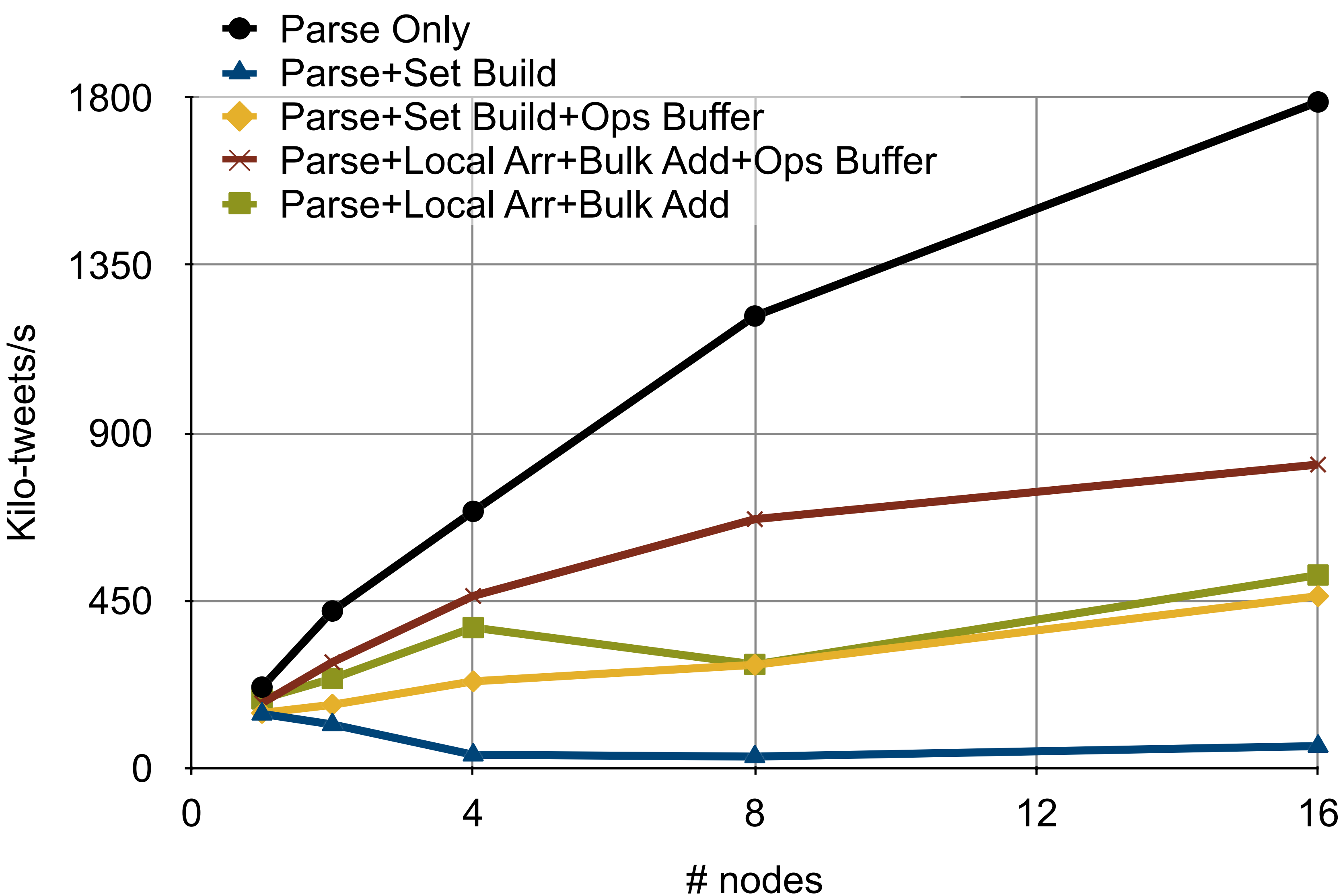


Provides *Aggregation* only

First Part Scalability + Manual Aggregation



First Part Scalability: Combining Approaches



Processing Tweets: Productivity Comparison

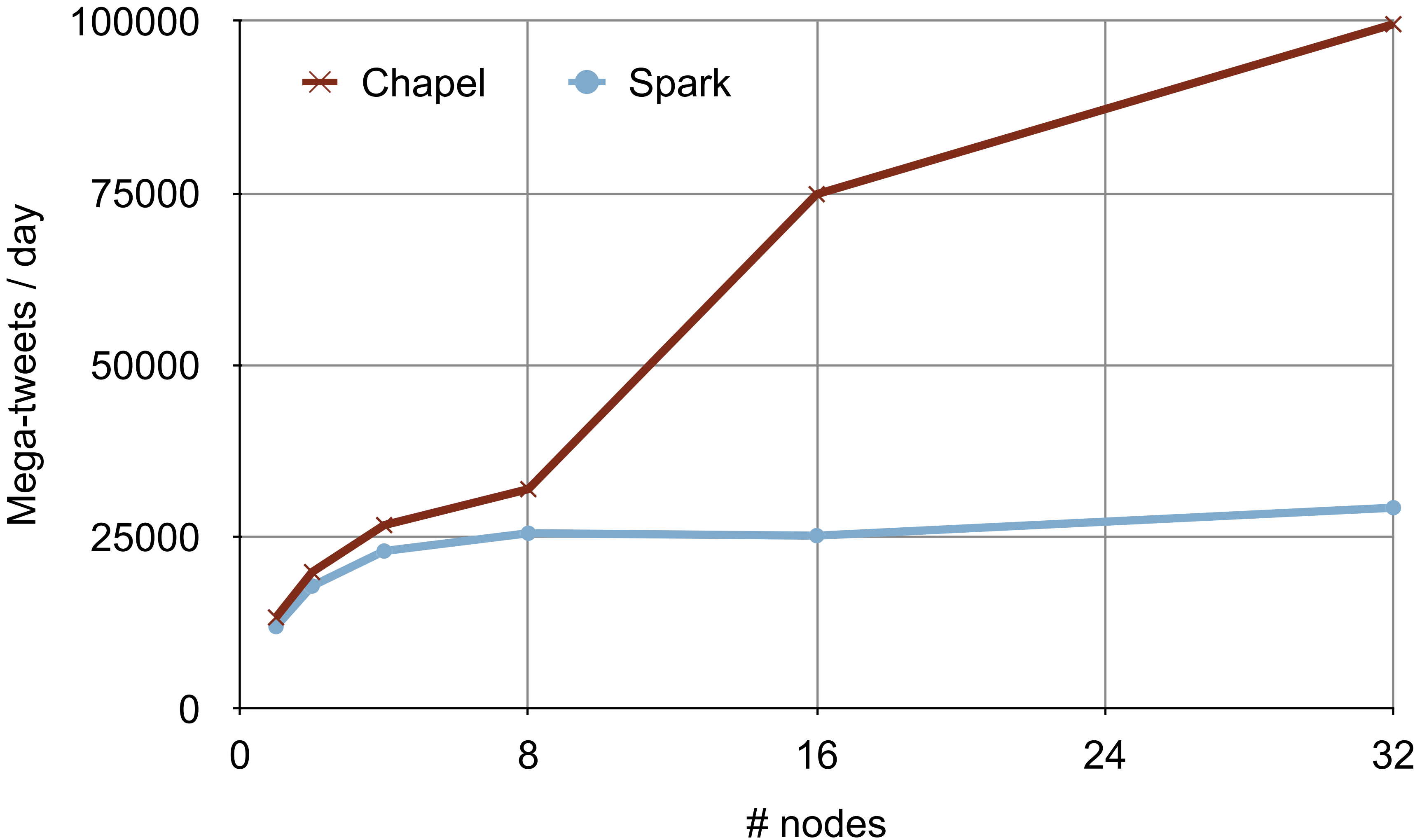
Spark

- RDDs are immutable
- Algorithm written in terms of mapping a fn on data

Chapel

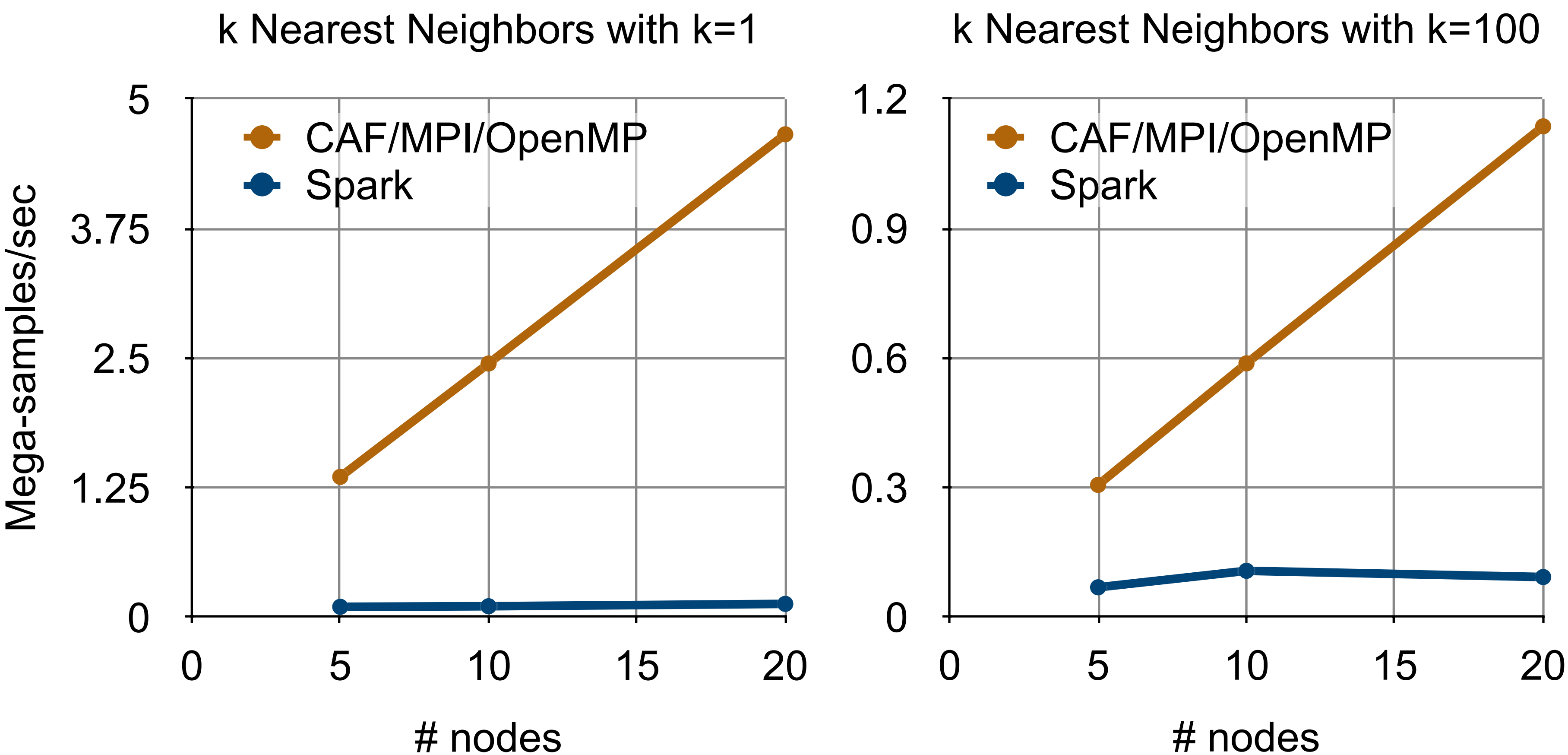
- Chapel arrays are mutable
- Algorithm written in terms of parallel loops

First Part: Chapel vs Spark*



* Lots of caveats. Chapel and Spark implementations are not necessarily optimal. Computing mutual mentions only. 420 files, XC30 36-cores/locale, Chapel version used gasnet, fifo, gnu, fe29555c. Spark 1.5.2

Previous Research on Spark Scalability: k Nearest Neighbors



Data from experiments reported in Reyes-Ortiz, Oneto, Anguita. Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf

Concluding

- We improved scalability for distributed domain +=
- Chapel performance compared favorably with Spark
- We think Chapel has a compelling future in data analytics

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

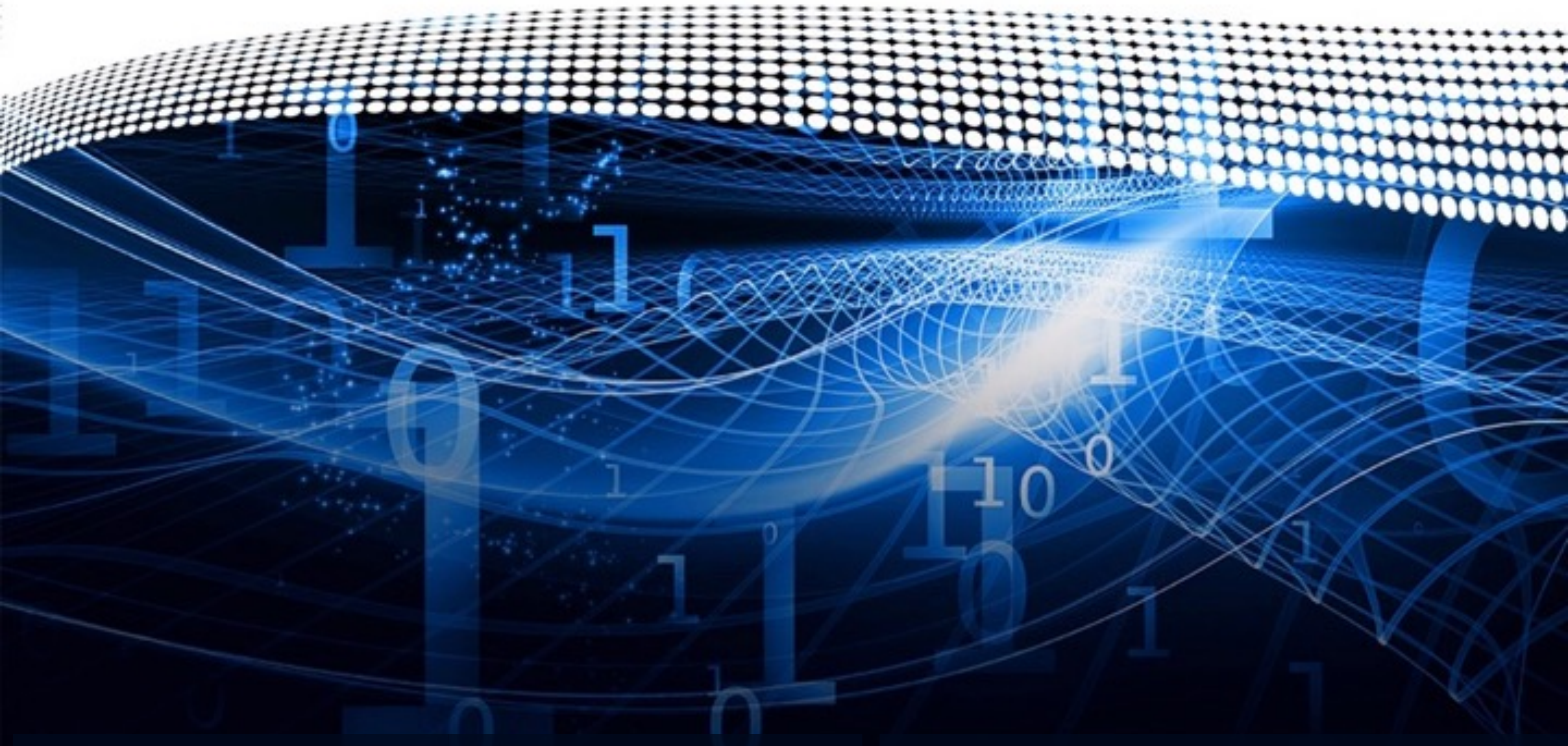
Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

COMPUTE | STORE | ANALYZE



http://chapel.cray.com chapel_info@cray.com http://github.com/chapel-lang/chapel/

Backup Slides

Processing Tweets: Motivation

Motivating Question: Is Chapel useful for Data Analytics?

- What would it look like?
- What features are we missing?

Processing Tweets: Label Propagation

Label Propagation Algorithm

(described in *Near linear time algorithm to detect community structures in large-scale networks*)

1. Initialize the labels at all nodes in the network.
2. Set $i = 1$.
3. Arrange the nodes in the network in a random order and set it to X.
4. For each x in X, set node x's label to the one that occurs most frequently among neighbors, with ties broken uniformly randomly.
5. If every node has a label that the maximum number of neighbors have, stop the algorithm. Otherwise, set $i = i + 1$ and go to step 3.

Processing Tweets: Implementation Overview

- **First version < 400 lines of Chapel code**
  - plus a Graph module (< 300 lines, to become a standard module)
- **current version is partly multi-locale**
- **graph representation similar to other Chapel graph codes**
  - e.g., SSCA#2
- **I/O is different**

Processing Tweets: Algorithm in Chapel

Algorithm closely matches the psuedocode:

```
var i = 0;
var go: atomic bool;
go.write(true);
while go.read(…) && i < maxiter {
  go.write(false);
  // for each x in the randomized order
  forall vid in reordered_vertices {
    // set the label to the most frequent among neigbors
    mylabel = labels[vid].read(memory_order_relaxed);
    maxlabel = mostCommonLabelInNeighbors(vid);
    if countNeighborsWith(vid, mylabel) <
       countNeighborsWith(vid, maxlabel) then
       go.write(true); // stop the algorithm if …
    labels[vid].write(maxlabel, memory_order_relaxed);
  }
  i += 1;
}
```

Sidebar on I/O for Twitter Processing in Chapel

Example Tweet in JSON format

- **Tweets have 34 top-level fields**
  - including nested structures containing much more data

{ "coordinates": null, "created_at": "Fri Oct 16 16:00:00 +0000 2015", "favorited": false, "truncated": false, "id_str": "28031452151", "entities": { "urls": [{ "expanded_url": null, "url": "http://chapel.cray.com", "indices": [69, 100] }], "hashtags": [ ], "user_mentions": [{ "name": "Cray Inc.", "id_str": "23424245", "id": 23424245, "indices": [25, 30], "screen_name": "cray" }] }, "in_reply_to_user_id_str": null, "text": "Let's mention the user @cray – here is an embedded url .... http://chapel.cray.com", "contributors": null, "id": 28039652140, "retweet_count": null, "in_reply_to_status_id_str": null, "geo": null, "retweeted": false, "in_reply_to_user_id": null, "user": { "profile_sidebar_border_color": "C0DEED", "name": "Cray Inc.", "profile_sidebar_fill_color": "DDEEF6", "profile_background_tile": false, "profile_image_url": "http://a3.twimg.com/profile_images/2342452/icon_normal.png", "location": "Seattle, WA", "created_at": "Fri Oct 10 23:10:00 +0000 2008", "id_str": "23502385", "follow_request_sent": false, "profile_link_color": "0084B4", "favourites_count": 1, "url": "http://cray.com", "contributors_enabled": false, "utc_offset": -25200, "id": 23548250, "profile_use_background_image": true, "listed_count": 23, "protected": false, "lang": "en", "profile_text_color": "333333", "followers_count": 1000, "time_zone": "Mountain Time (US & Canada)", "verified": false, "geo_enabled": true, "profile_background_color": "C0DEED", "notifications": false, "description": "Cray Inc", "friends_count": 71, "profile_background_image_url": "http://s.twimg.com/a/2349257201/images/themes/theme1/bg.png", "statuses_count": 302, "screen_name": "gnip", "following": false, "show_all_inline_media": false }, "in_reply_to_screen_name": null, "source": "web", "place": null, "in_reply_to_status_id": null }

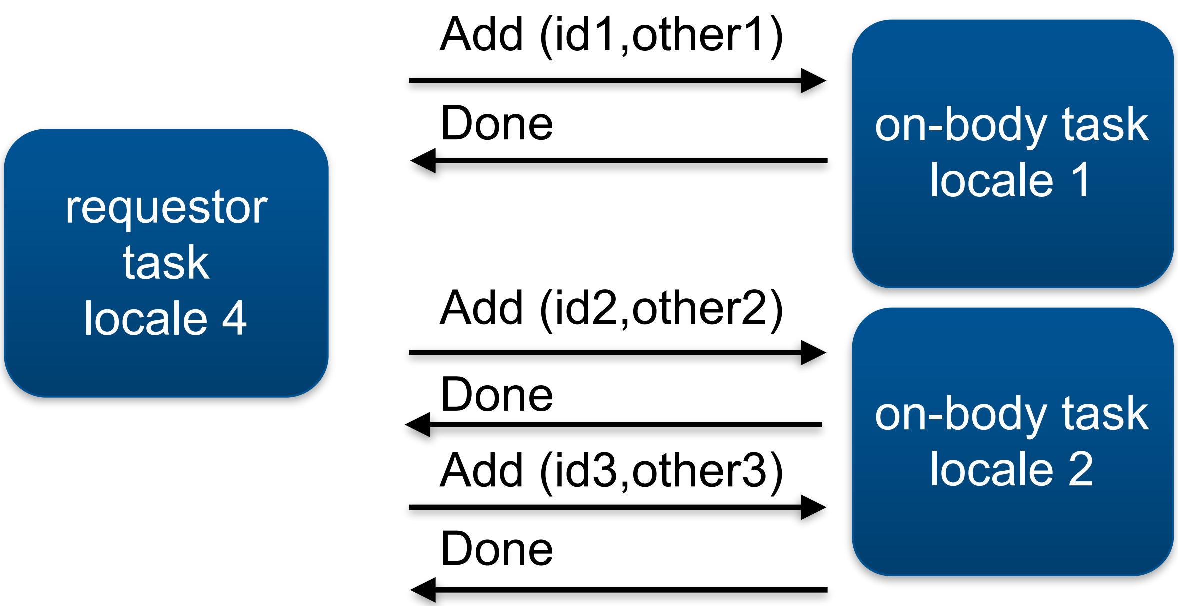
Reading JSON Tweets

```
// define Chapel records whose fields reflect only
// the portions of the JSON data we care about

record TweetUser {
  var id: int;
}
record TweetEntities {
  var user_mentions: list(TweetUser);
}
record User {
  var id: int;
}
record Tweet {
  var id: int,
      user: User,
      entities: TweetEntities;
}
```

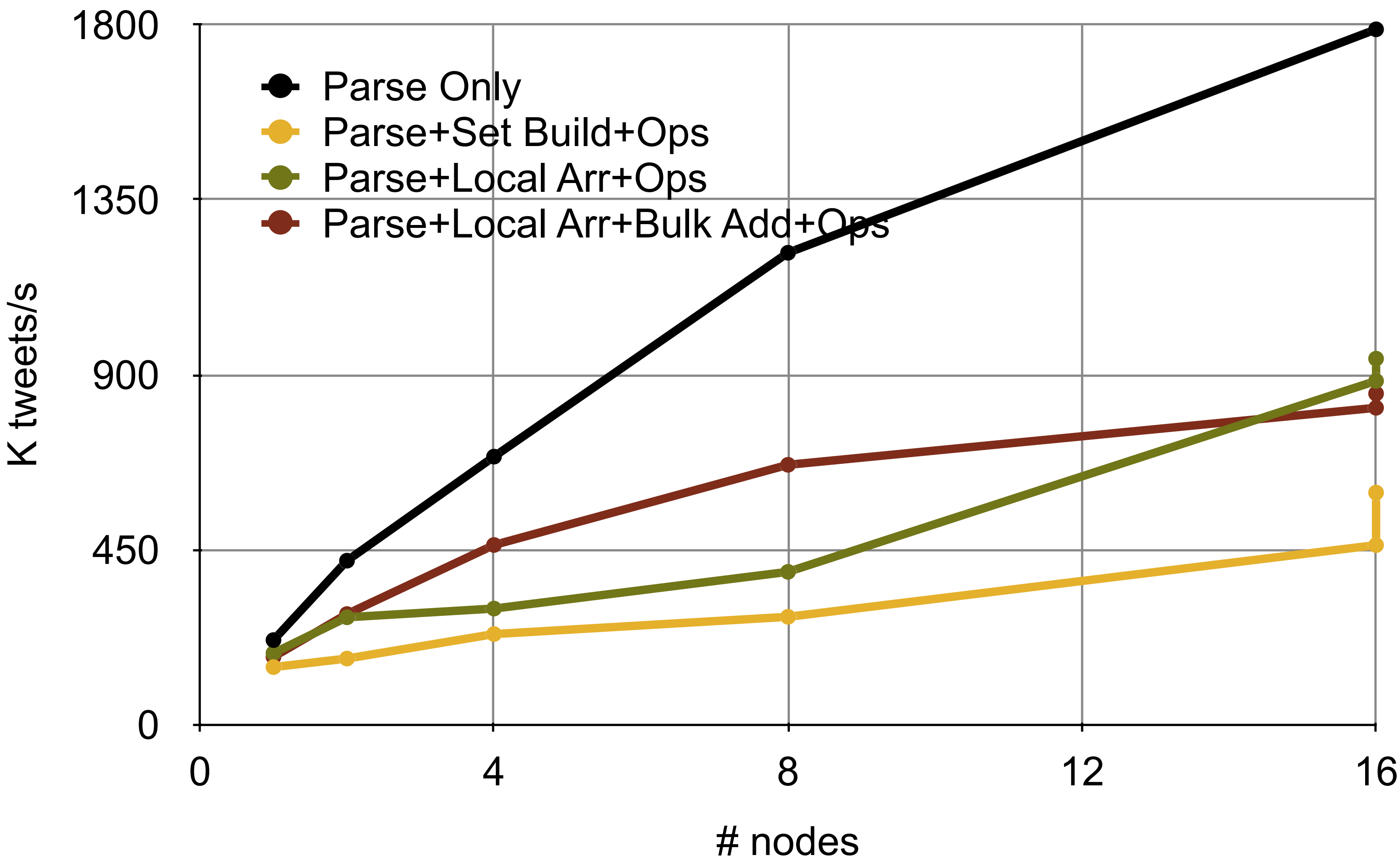
```
proc process_json(…) {
  var tweet: Tweet;

  while true {
    // "%~jt" format string:
    //     j: JSON format
    //     t: any record
    //     ~: skip other fields
    got = logfile.readf("%~jt",
                         tweet,
                         error=err);
    if got && !err then
      handle_tweet(tweet);
    if err == EFORMAT then ...;
    if err == EEOF then break;
}
```

Set Addition is Blocking

```
Pairs += (id1, other1);
Pairs += (id2, other2);
Pairs += (id3, other3);
...
```



First Part Scalability + Operations + Manual



Processing Tweets: Productivity Comparison

Spark

- **RDDs are immutable**
  - create new RDD every iteration through algorithm
- **Algorithm written in terms of mapping a fn on data**
  - difficult to visit vertices in random order
  - movement of information is described as messages contributing to a new RDD
  - breaking ties randomly might require a custom operator

Chapel

- **Chapel arrays are mutable**
  - Algorithm can update labels in-place
- **Algorithm written in terms of parallel loops**
  - straightforward to visit vertices in random order
  - movement of information occurs through variable reads and writes
  - breaking ties randomly is an easy change

These differences reflect Spark's declarative nature vs. Chapel's imperative design.