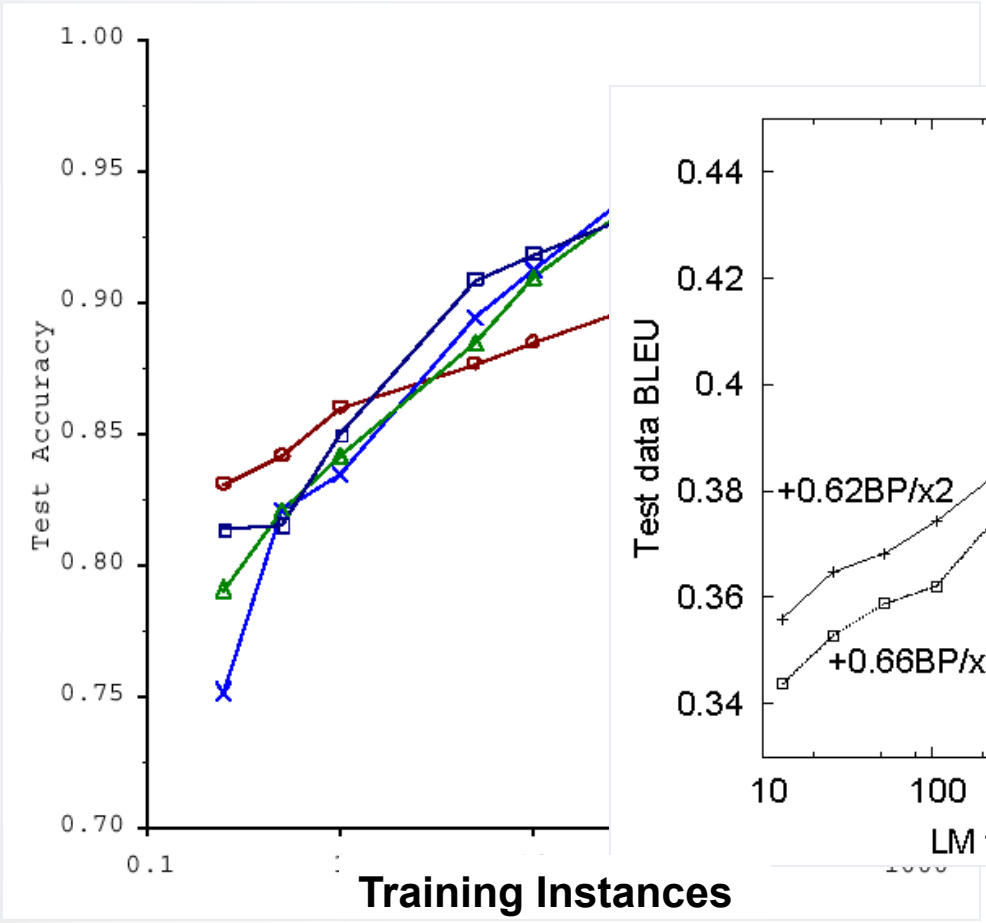
Learning with Big Data I: Divide and Conquer

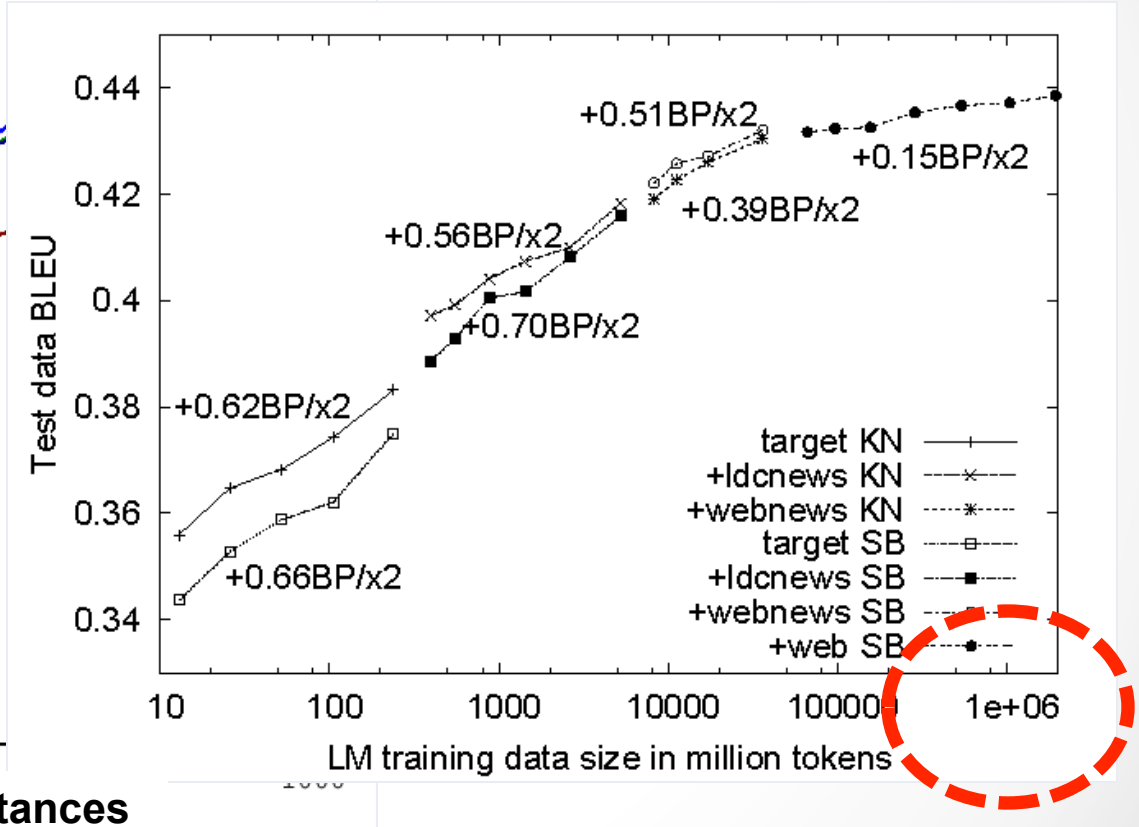
Chris Dyer

28 July 2014 – LxMLS

Carnegie Mellon

No data like more data!





(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)

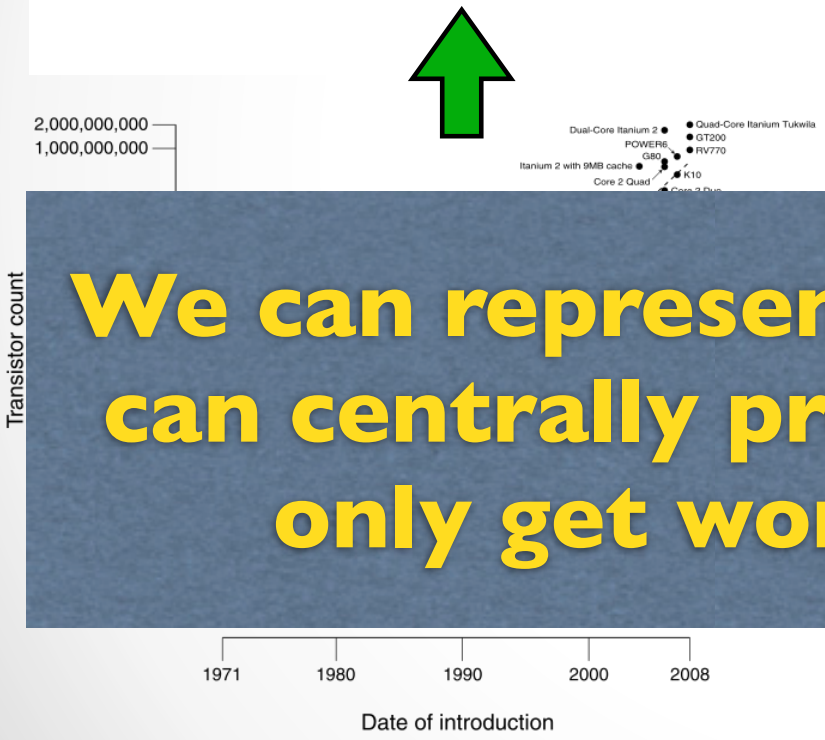
Problem

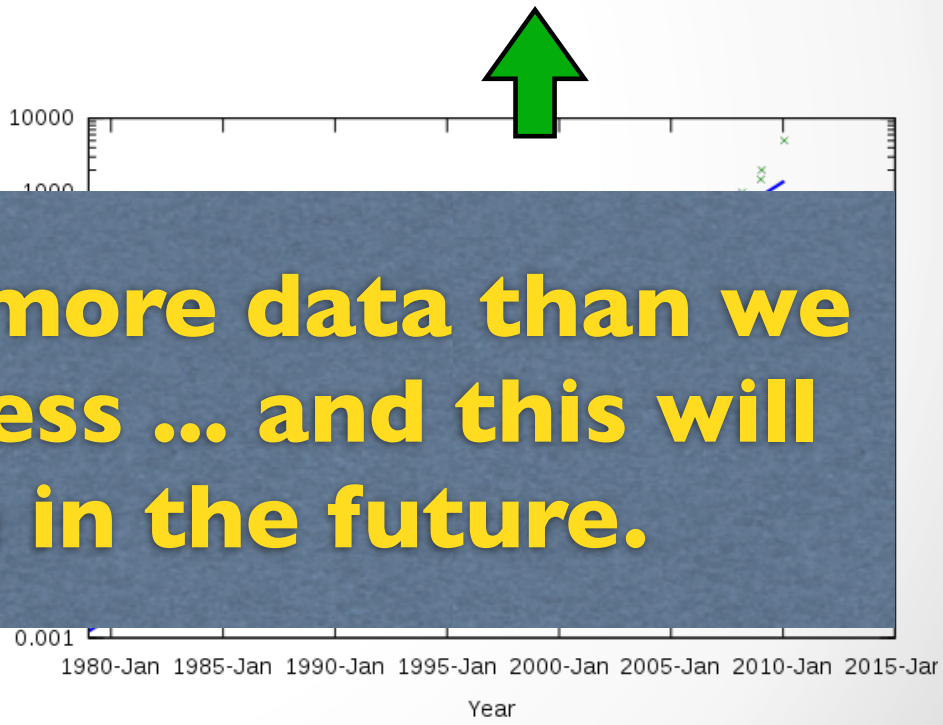
Moore's law (~ power/CPU)

$$\#(t) \approx n_0 \times 2^{t/(2\text{ years})}$$

Hard disk capacity

$$\#(t) \approx m_0 \times 3.2^{t/(2\text{ years})}$$





We can represent more data than we can centrally process ... and this will only get worse in the future.

Solution

- **Partition training data into subsets**
  - Core concept: **data shards**
  - Algorithms that work primarily "inside" shards and **communicate minimally**
  - Nodes is a graph, edges represent communication pathways
- Alternative solutions
  - Reduce data points by instance selection (e.g. core sets)
  - Dimensionality reduction / compression
- Related problems
  - Large numbers of dimensions (horizontal partitioning)
    - Model is too big to fit in memory
  - Large numbers of related tasks
    - every Gmail user has his own opinion about what is spam

Outline

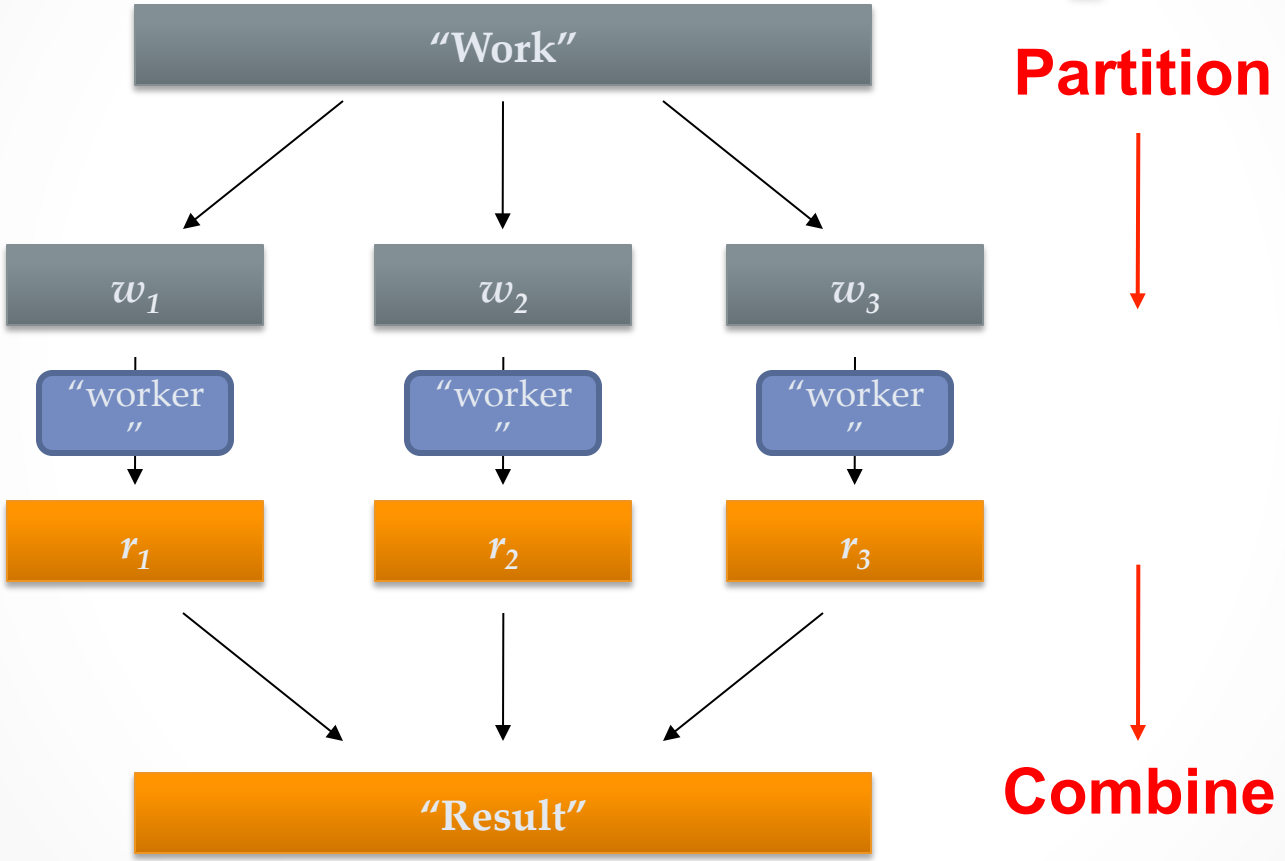
- MapReduce
- Design patterns for MapReduce

- Batch Learning Algorithms on MapReduce
- Distributed Online Algorithms

MapReduce

cheap commodity clusters
+ simple, distributed programming models
= data-intensive computing for all

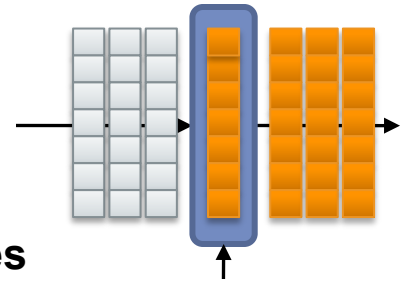
Divide and Conquer



It's a bit more complex…

Fundamental issues

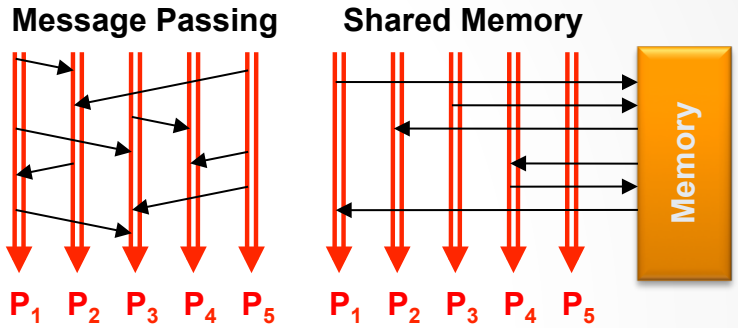
scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …



Architectural issues

Flynn's taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth
UMA vs. NUMA, cache coherence

Different programming models



Different programming constructs

mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues, …

Common problems

livelock, deadlock, data starvation, priority inversion…
dining philosophers, sleeping barbers, cigarette smokers, …

The reality: programmer shoulders the burden of managing concurrency…

Source: Ricardo Guimarães Herrmann

Typical Problem

Map

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results

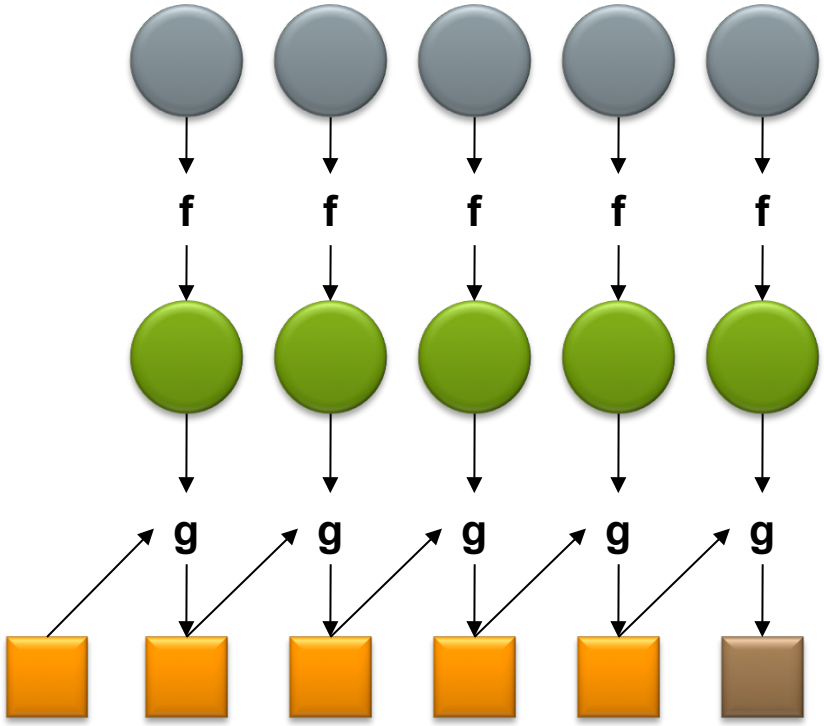
Reduce

- Generate final output

Key idea: functional abstraction for these two operations

Map

Fold



Map

Reduce

MapReduce

- Programmers specify two functions:

  map (k, v) → <k', v'>*

  reduce (k', v') → <k', v'>*

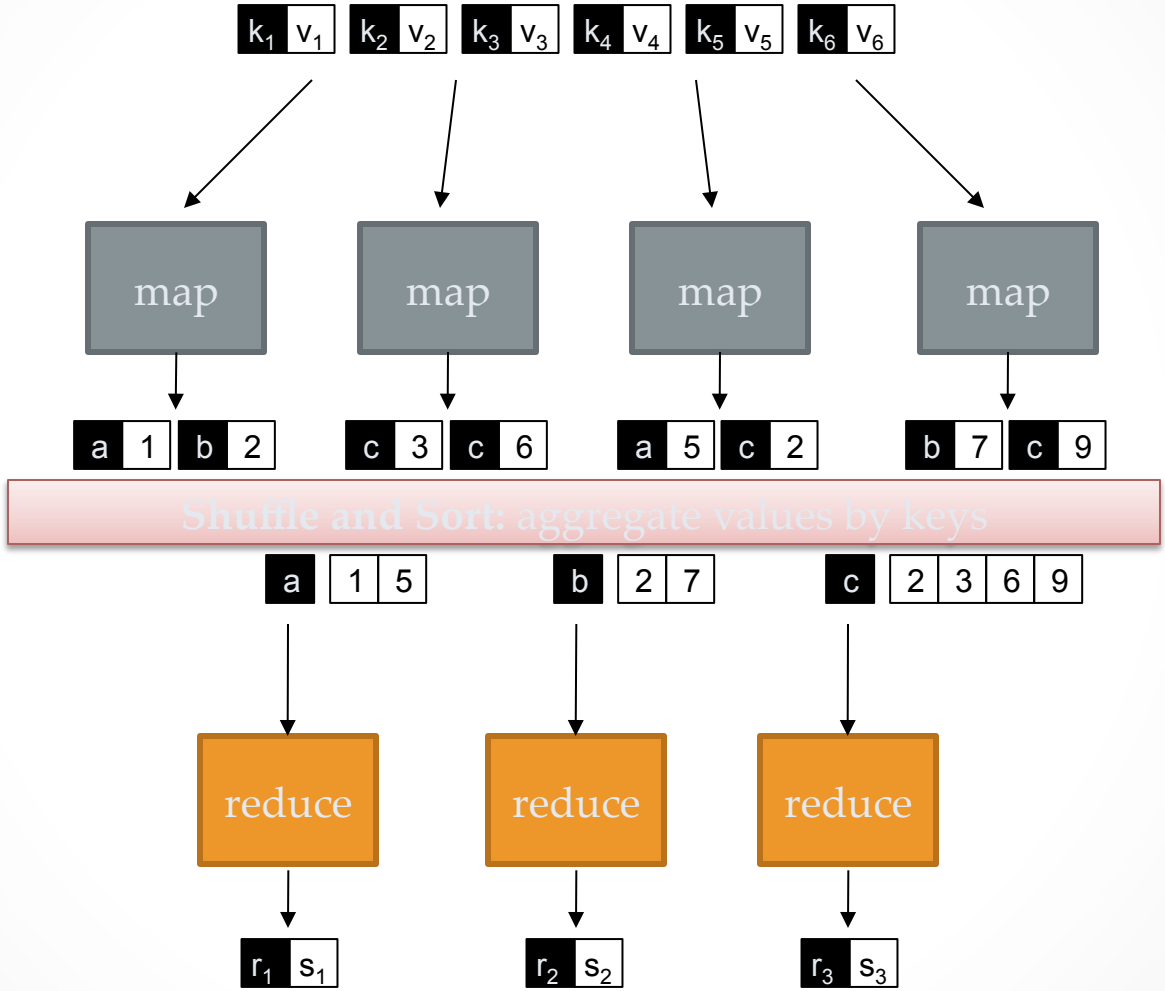
  - All values with the same key are reduced together
- Usually, programmers also specify:

  partition (k', number of partitions) → partition for k'

  - Often a simple hash of the key, e.g. hash(k') mod n
  - Allows reduce operations for different keys in parallel

  combine(k',v') → <k',v'>

  - "Mini-reducers" that run in memory after the map phase
  - Optimizes to reduce network traffic & disk writes
- Implementations:
  - Google has a proprietary implementation in C++
  - Hadoop is an open source implementation in Java


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 9
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 9
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

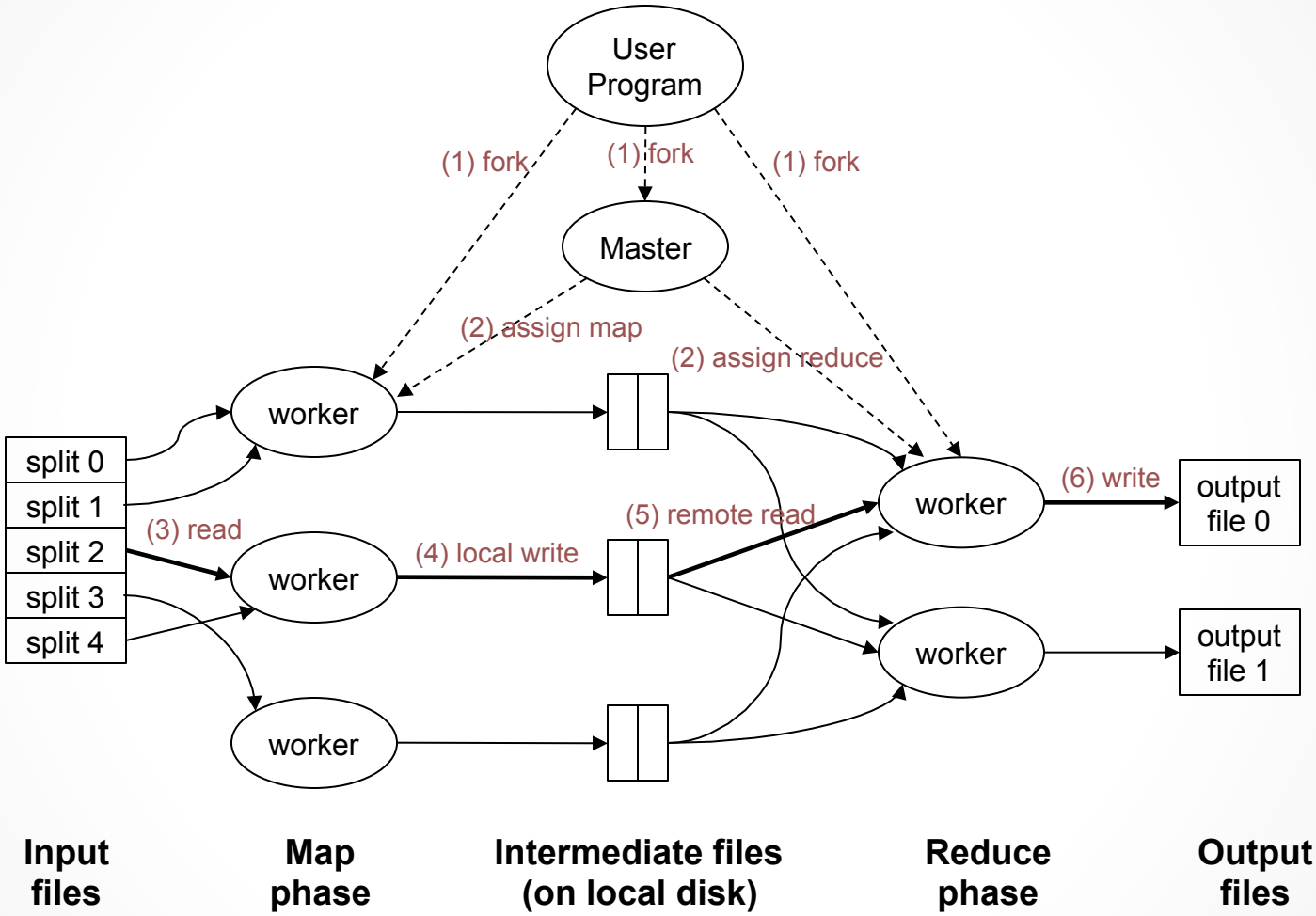
MapReduce Runtime

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves the process to the data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

"Hello World": Word Count

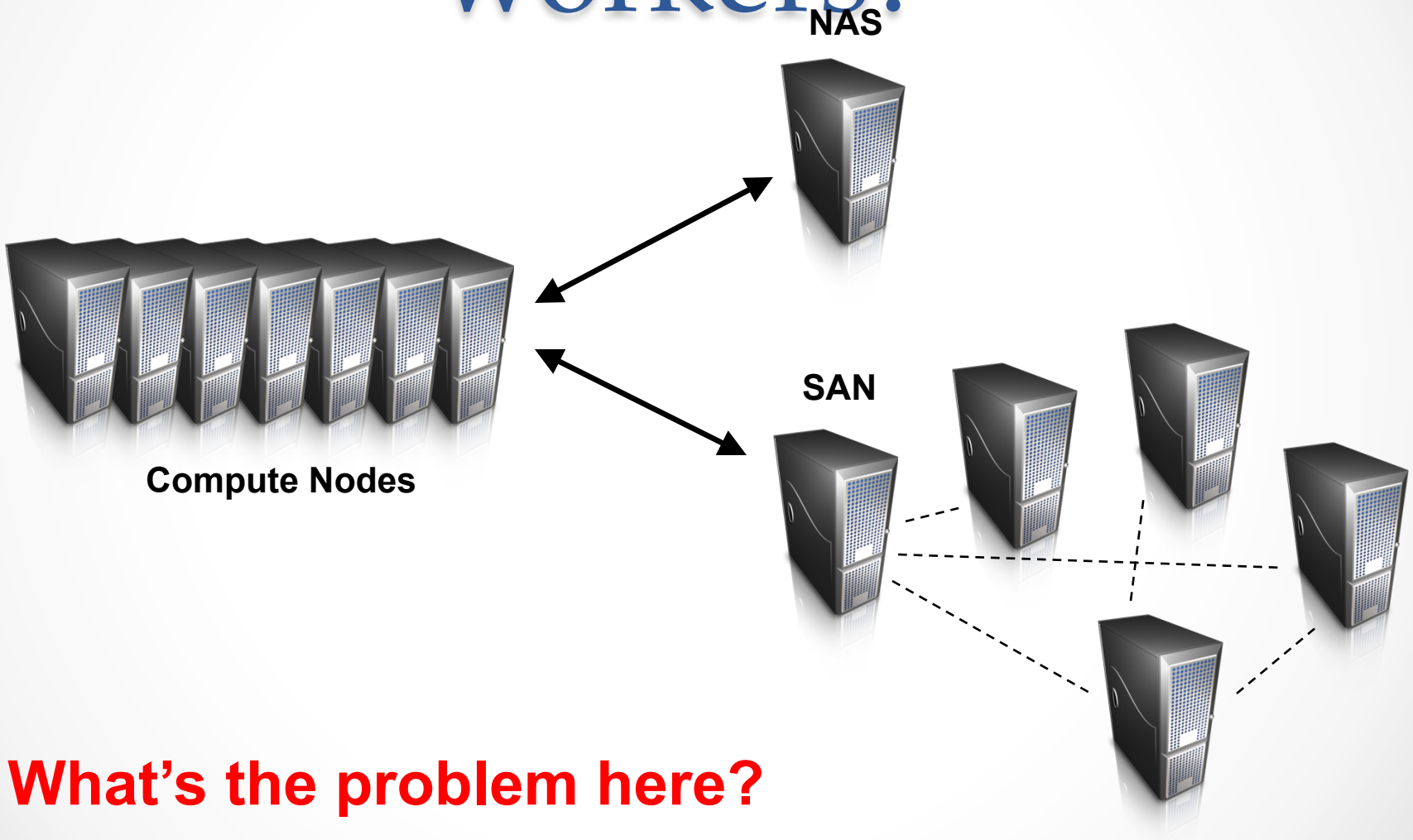
```
Map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_values:
        EmitIntermediate(w, "1");

Reduce(String key, Iterator intermediate_values):
    // key: a word, same for input and output
    // intermediate_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
        Emit(AsString(result));
```



Redrawn from Dean and Ghemawat (OSDI 2004)

How do we get data to the workers?



What's the problem here?

Distributed File System

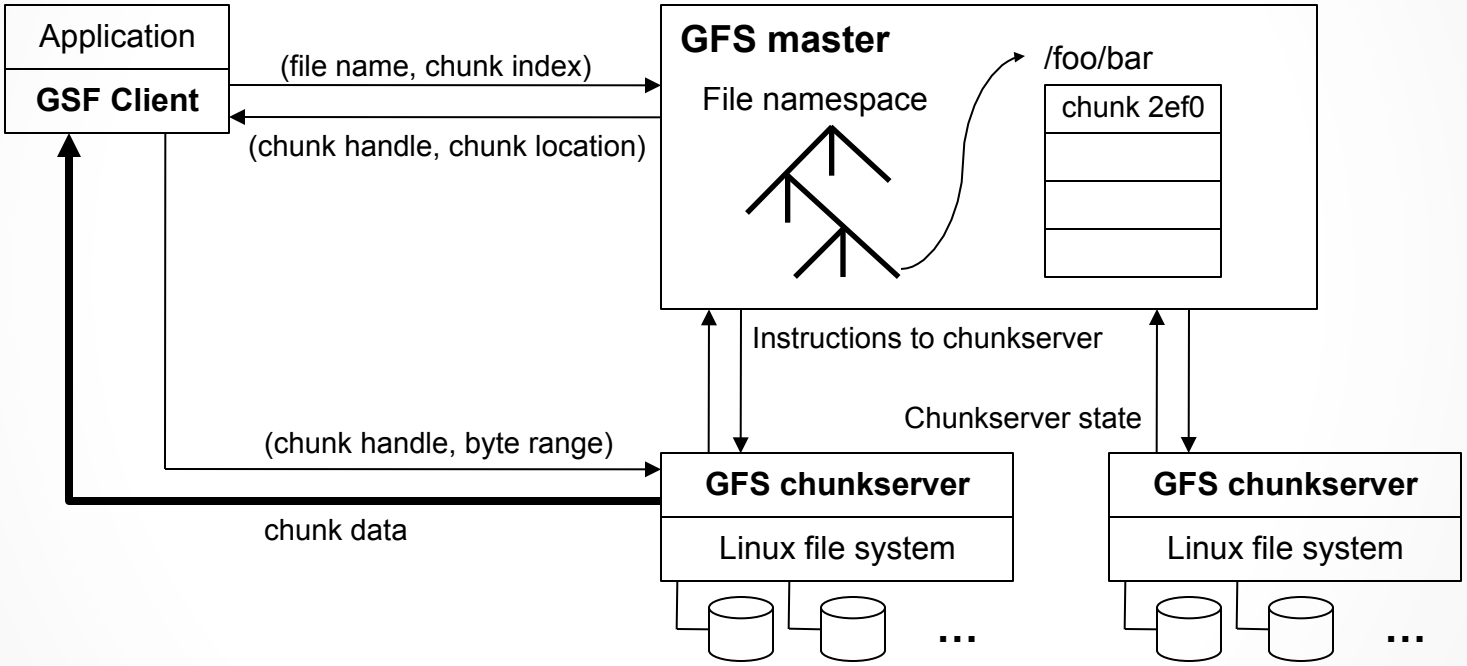
- Don't move data to workers… Move workers to the data!
  - Store data on the local disks for nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, disk throughput is good
- A distributed file system is the answer
  - GFS (Google File System)
  - HDFS for Hadoop (= GFS clone)

GFS: Assumptions

- Commodity hardware over "exotic" hardware
- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of HUGE files
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
- High sustained throughput over low latency

GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Simplify the API
  - Push some of the issues onto the client



Redrawn from Ghemawat et al. (SOSP 2003)

Master's Responsibilities

- Metadata storage
- Namespace management/locking
- Periodic communication with chunkservers
- Chunk creation, replication, rebalancing
- Garbage collection

Questions?

MapReduce “killer app”:
Graph Algorithms

Graph Algorithms: Topics

- Introduction to graph algorithms and graph representations
- Single Source Shortest Path (SSSP) problem
  - Refresher: Dijkstra's algorithm
  - Breadth-First Search with MapReduce
- PageRank

What's a graph?

- G = (V,E), where
  - V represents the set of vertices (nodes)
  - E represents the set of edges (links)
  - Both vertices and edges may contain additional information
- Different types of graphs:
  - Directed vs. undirected edges
  - Presence or absence of cycles
  - …

Some Graph Problems

- Finding shortest paths
  - Routing Internet traffic and UPS trucks
- Finding minimum spanning trees
  - Telco laying down fiber
- Finding Max Flow
  - Airline scheduling
- Identify "special" nodes and communities
  - Breaking up terrorist cells, spread of swine/avian/… flu
- Bipartite matching
  - Monster.com, Match.com
- And of course… PageRank

Representing Graphs

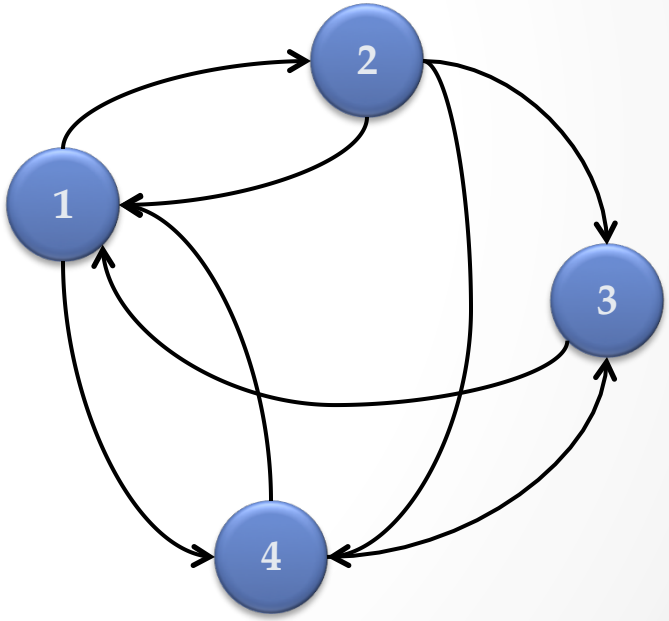
- G = (V, E)
  - A poor representation for computational purposes
- Two common representations
  - Adjacency matrix
  - Adjacency list

Adjacency Matrices

Represent a graph as an n x n square matrix M

- $n = |V|$
- $M_{ij} = 1$ means a link from node i to j

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



Adjacency Lists

Take adjacency matrices… and throw away all the zeros

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

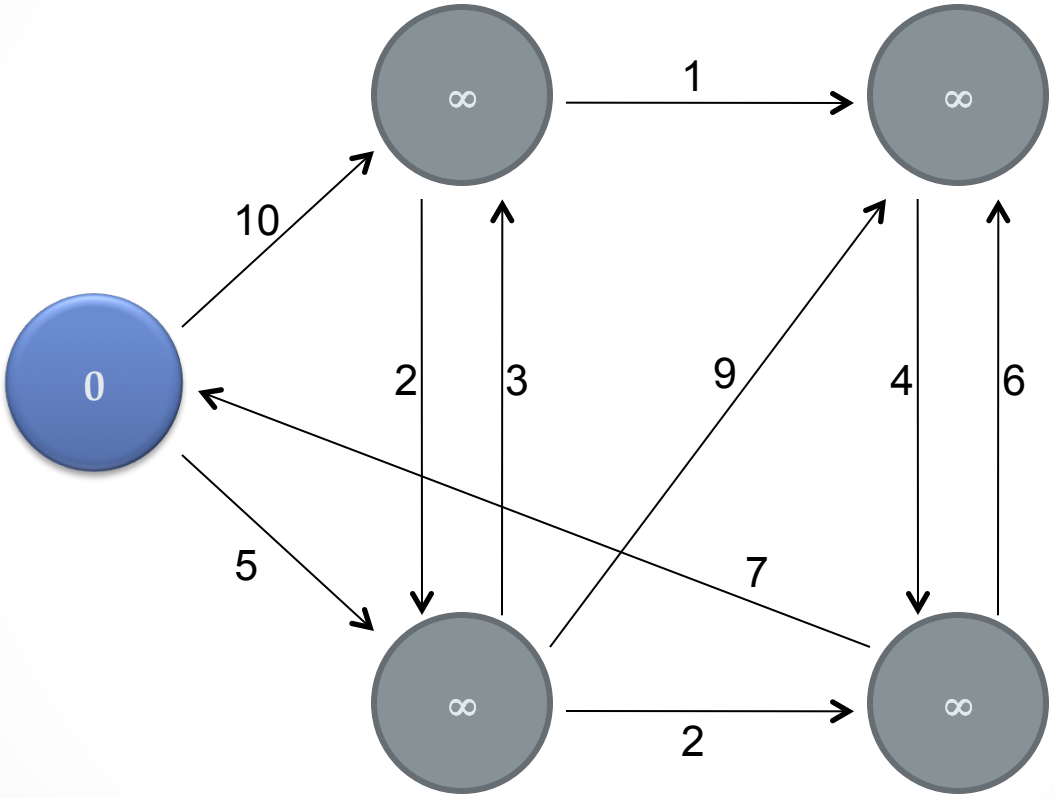
Adjacency Lists: Critique

- Advantages:
  - Much more compact representation
  - Easy to compute over outlinks
  - Graph structure can be broken up and distributed
- Disadvantages:
  - Much more difficult to compute over inlinks

Single Source Shortest Path

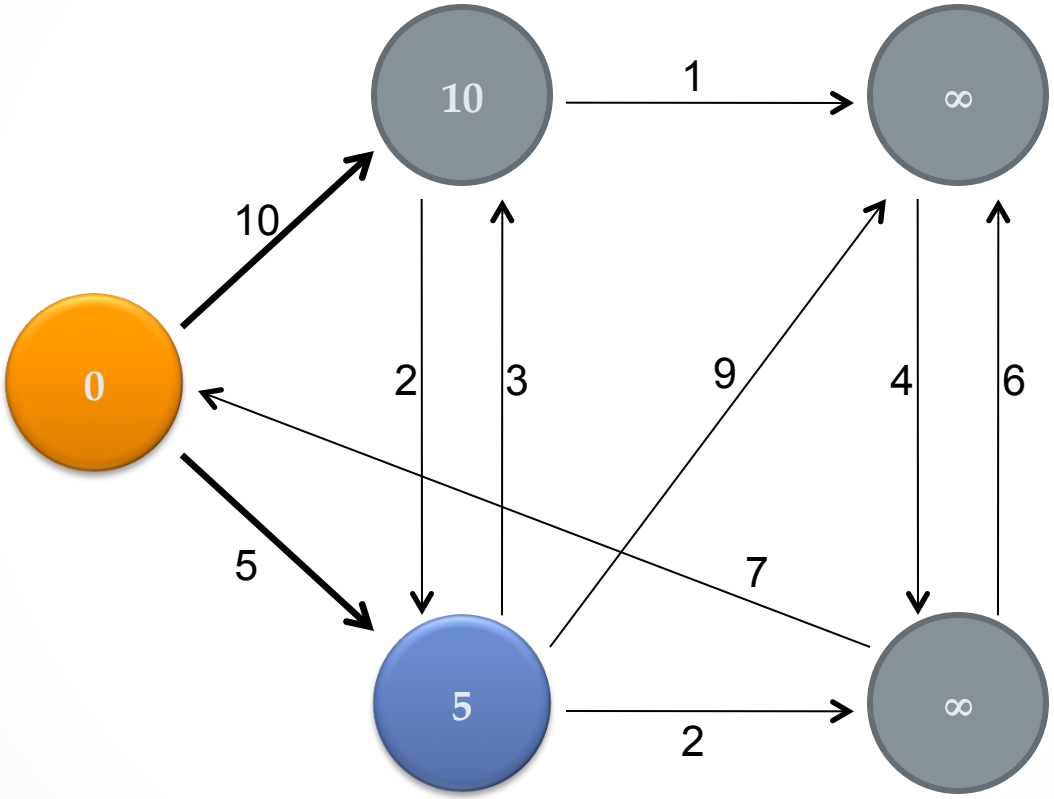
- **Problem:** find shortest path from a source node to one or more target nodes
- First, a refresher: Dijkstra's Algorithm

Dijkstra's Algorithm Example



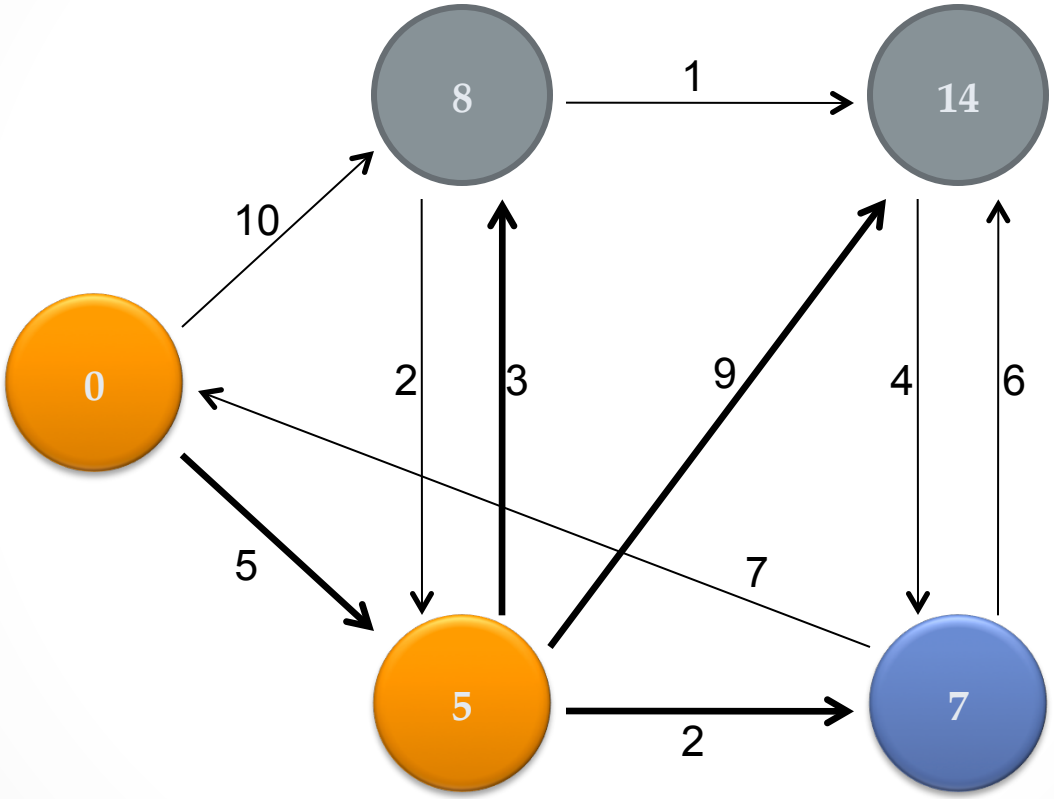
Example from CLR

Dijkstra's Algorithm Example



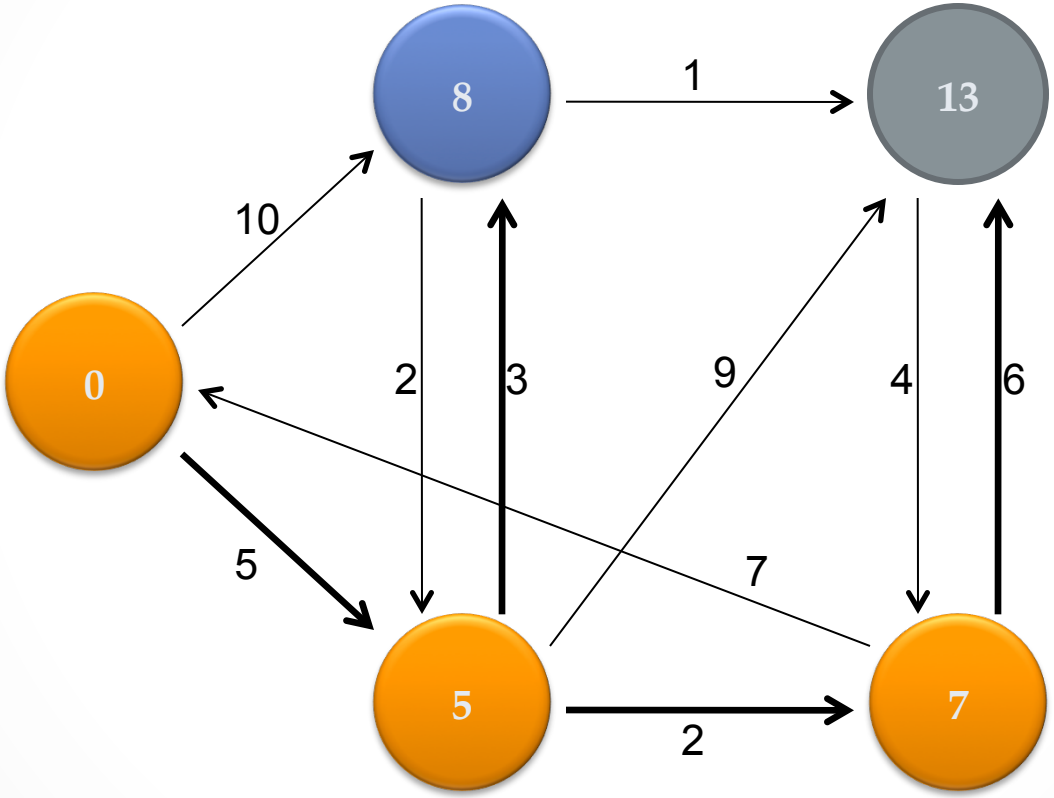
Example from CLR

Dijkstra's Algorithm Example



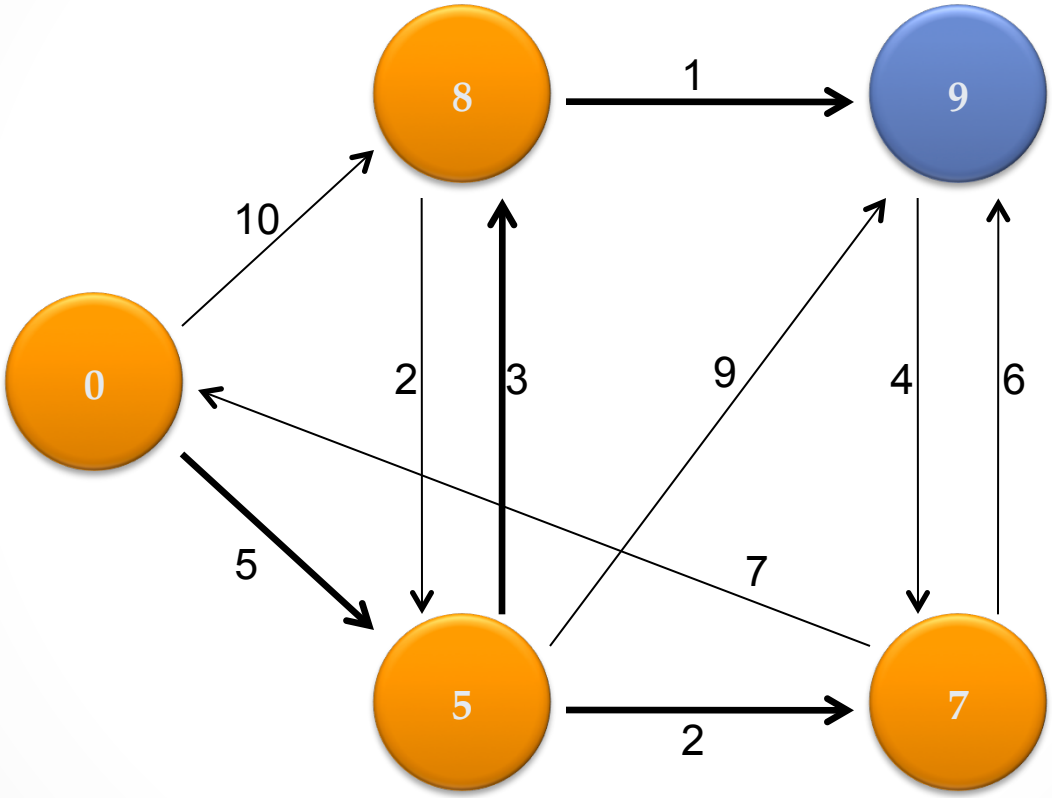
Example from CLR

Dijkstra's Algorithm Example



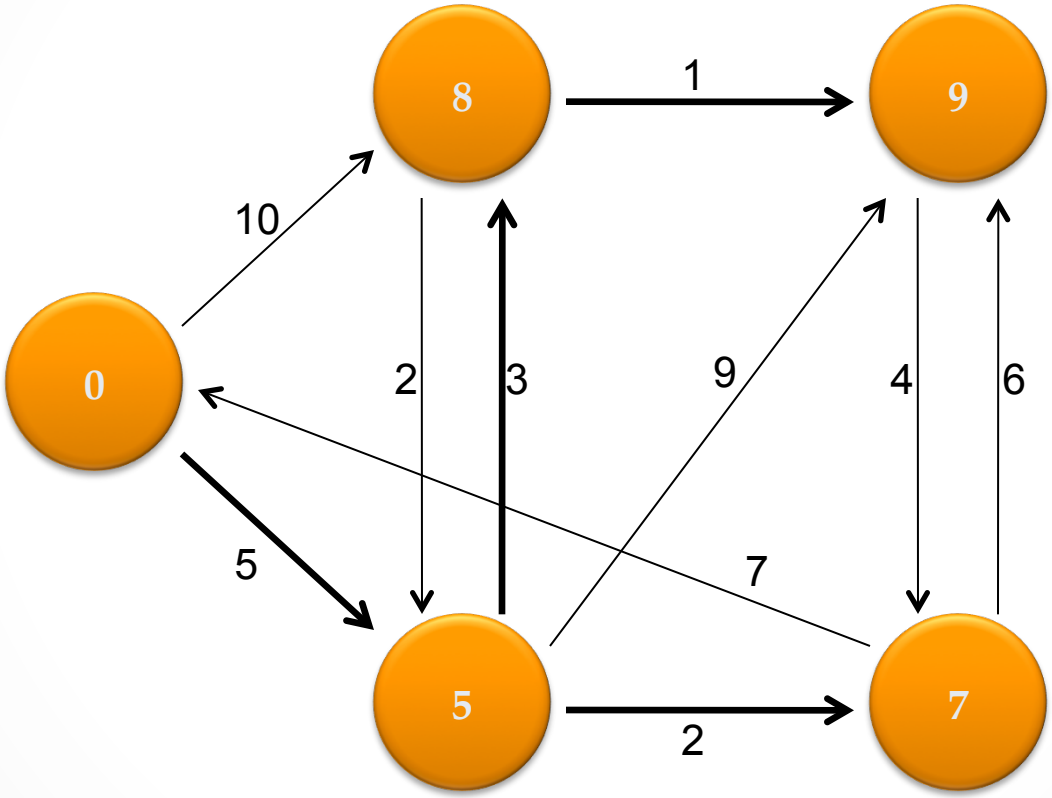
Example from CLR

Dijkstra's Algorithm Example



Example from CLR

Dijkstra's Algorithm Example



Example from CLR

Single Source Shortest Path

- **Problem:** find shortest path from a source node to one or more target nodes
- Single processor machine: Dijkstra's Algorithm
- MapReduce: parallel Breadth-First Search (BFS)

Finding the Shortest Path

- First, consider equal edge weights
- Solution to the problem can be defined inductively
- Here's the intuition:
  - DistanceTo(startNode) = 0
  - For all nodes *n* directly reachable from startNode, DistanceTo(*n*) = 1
  - For all nodes *n* reachable from some other set of nodes S, DistanceTo(n) = 1 + min(DistanceTo(m), $m \in S$)

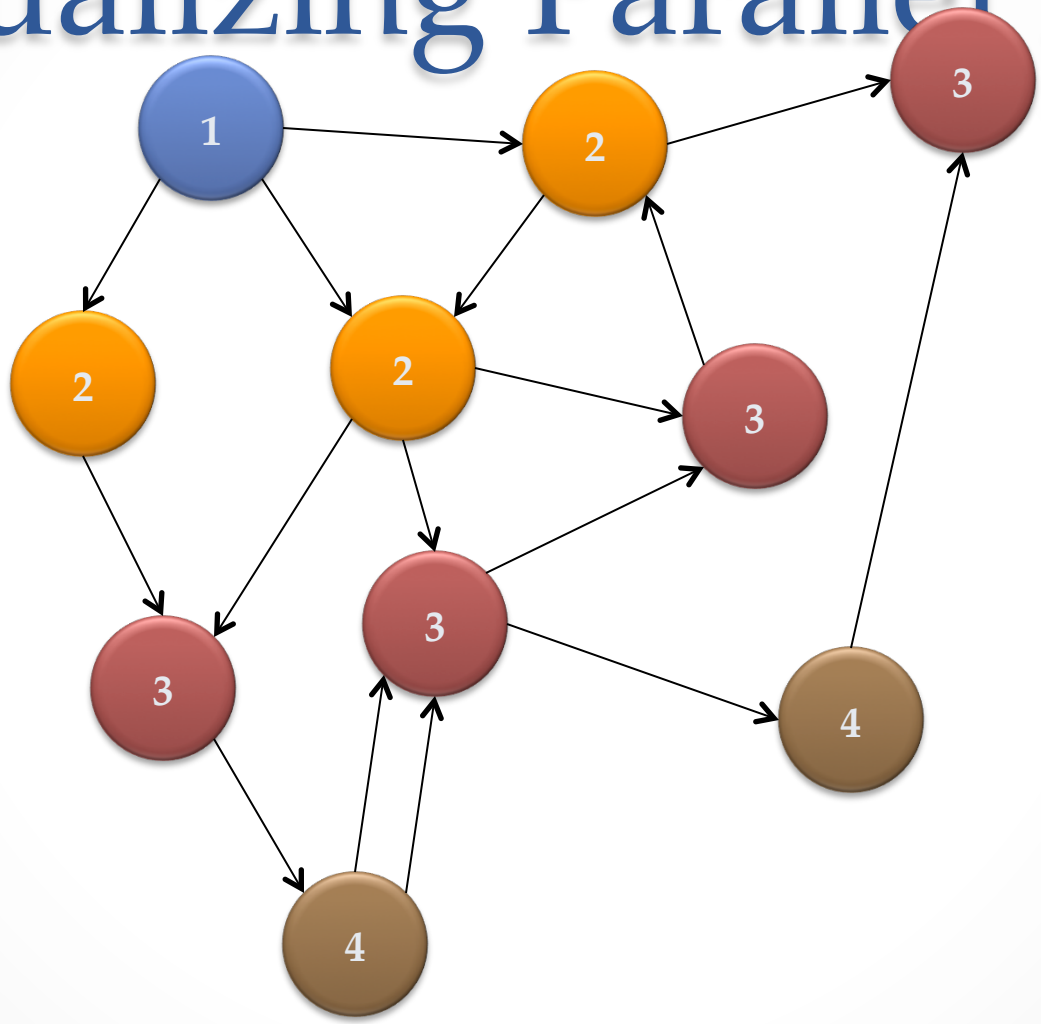
From Intuition to Algorithm

- A map task receives
  - Key: node *n*
  - Value: D (distance from start), points-to (list of nodes reachable from *n*)
- $\forall p \in$ points-to: emit (*p*, D+1)
- The reduce task gathers possible distances to a given *p* and selects the minimum one

Multiple Iterations Needed

- This MapReduce task advances the "known frontier" by one hop
  - Subsequent iterations include more reachable nodes as frontier advances
  - Multiple iterations are needed to explore entire graph
  - Feed output back into the same MapReduce task
- Preserving graph structure:
  - Problem: Where did the points-to list go?
  - Solution: Mapper emits (*n*, points-to) as well

Visualizing Parallel BFS



Termination

- Does the algorithm ever terminate?
  - Eventually, all nodes will be discovered, all edges will be considered (in a connected graph)
- When do we stop?

Weighted Edges

- Now add positive weights to the edges
- Simple change: points-to list in map task includes a weight *w* for each pointed-to node
  - emit (*p*, D+w_p) instead of (*p*, D+1) for each node *p*
- Does this ever terminate?
  - Yes! Eventually, no better distances will be found. When distance is the same, we stop
  - Mapper should emit (*n*, D) to ensure that "current distance" is carried into the reducer

Comparison to Dijkstra

- Dijkstra's algorithm is more efficient
  - At any step it only pursues edges from the minimum-cost path inside the frontier
- MapReduce explores all paths in parallel
  - Divide and conquer
  - Throw more hardware at the problem

General Approach

- MapReduce is adept at manipulating graphs
  - Store graphs as adjacency lists
- Graph algorithms with for MapReduce:
  - Each map task receives a node and its outlinks
  - Map task compute some function of the link structure, emits value with target as the key
  - Reduce task collects keys (target nodes) and aggregates
- Iterate multiple MapReduce cycles until some termination condition
  - Remember to "pass" graph structure from one iteration to next

Random Walks Over the Web

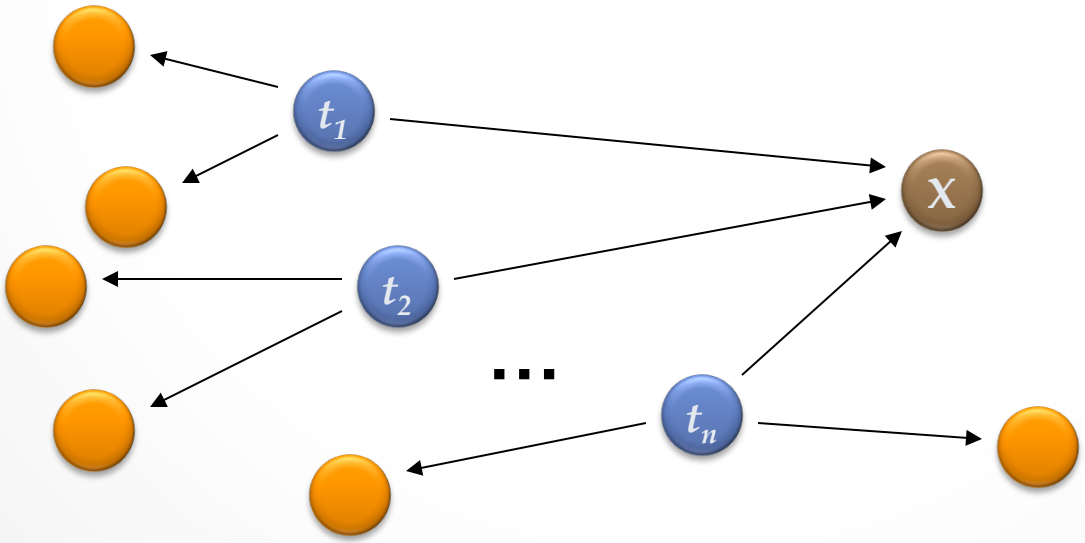
- Model:
  - User starts at a random Web page
  - User randomly clicks on links, surfing from page to page
- PageRank = the amount of time that will be spent on any given page

PageRank: Defined

Given page x with in-bound links $t_1 \ldots t_n$, where

- $C(t)$ is the out-degree of t
- α is probability of random jump
- N is the total number of nodes in the graph

$$PR(x) = \alpha\left(\frac{1}{N}\right) + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

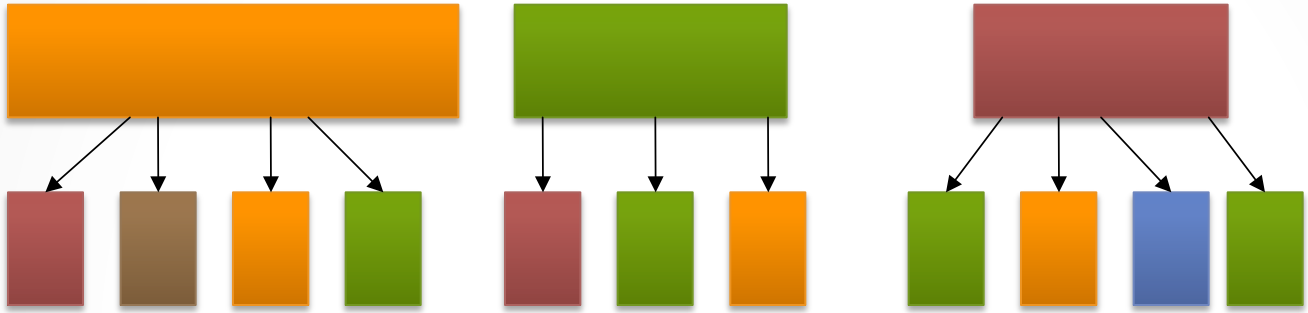


Computing PageRank

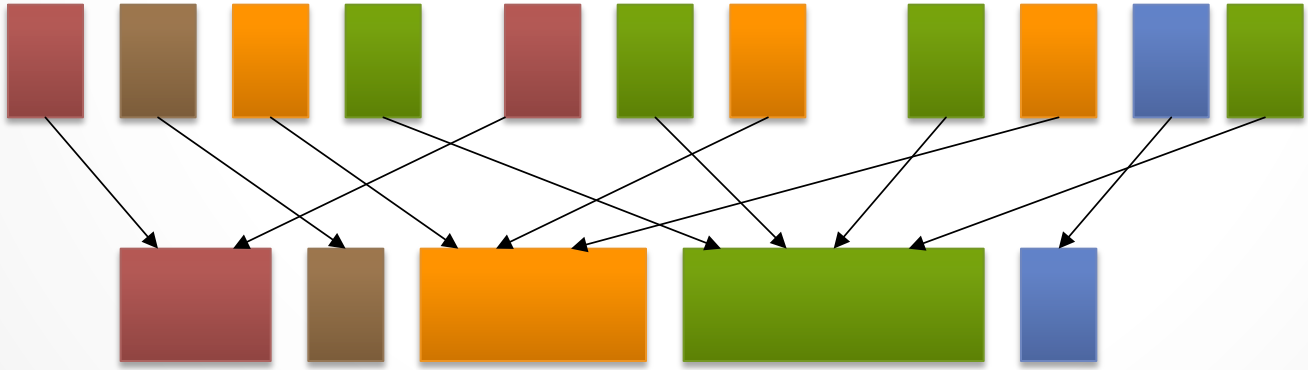
- Properties of PageRank
  - Can be computed iteratively
  - Effects at each iteration is local
- Sketch of algorithm:
  - Start with seed PR_i values
  - Each page distributes PR_i "credit" to all pages it links to
  - Each target page adds up "credit" from multiple in-bound links to compute PR_{i+1}
  - Iterate until values converge

PageRank in MapReduce

Map: distribute PageRank "credit" to link targets



Reduce: gather up PageRank "credit" from multiple sources to compute new PageRank value



Iterate until convergence

...

PageRank: Issues

- Is PageRank guaranteed to converge? How quickly?
- What is the "correct" value of α, and how sensitive is the algorithm to it?
- What about dangling links?
- How do you know when to stop?

Graph algorithms in MapReduce

- General approach
  - Store graphs as adjacency lists (node, points-to, points-to …)
  - Mappers receive (node, points-to*) tuples
  - Map task computes some function of the link structure
  - Output key is usually the target node in the adjacency list representation
  - Mapper typically outputs the graph structure as well
- Iterate multiple MapReduce cycles until some convergence criterion is met

Questions?

MapReduce Algorithm Design

Managing Dependencies

- Remember: Mappers run in isolation
  - You have no idea in what order the mappers run
  - You have no idea on what node the mappers run
  - You have no idea when each mapper finishes
- Tools for synchronization:
  - Ability to hold state in reducer across multiple key-value pairs
  - Sorting function for keys
  - Partitioner
  - Cleverly-constructed data structures

Motivating Example

- Term co-occurrence matrix for a text collection
  - M = N x N matrix (N = vocabulary size)
  - M_{ij}: number of times *i* and *j* co-occur in some context (for concreteness, let's say context = sentence)
- Why?
  - Distributional profiles as a way of measuring semantic distance
  - Semantic distance useful for many language processing tasks

"You shall know a word by the company it keeps" (Firth, 1957)

e.g., Mohammad and Hirst (EMNLP, 2006)

MapReduce: Large Counting Problems

- Term co-occurrence matrix for a text collection = specific instance of a large counting problem
  - A large event space (number of terms)
  - A large number of events (the collection itself)
  - Goal: keep track of interesting statistics about the events
- Basic approach
  - Mappers generate partial counts
  - Reducers aggregate partial counts

How do we aggregate partial counts efficiently?

First Try: "Pairs"

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For all pairs, emit (a, b) → count
- Reducers sums up counts associated with these pairs
- Use combiners!

Note: in these slides, we donate a key-value pair as k → v

"Pairs" Analysis

- Advantages
  - Easy to implement, easy to understand
- Disadvantages
  - Lots of pairs to sort and shuffle around (upper bound?)

Another Try: "Stripes"

- Idea: group together pairs into an associative array

$(a, b) \rightarrow 1$
$(a, c) \rightarrow 2$
$(a, d) \rightarrow 5$
$(a, e) \rightarrow 3$
$(a, f) \rightarrow 2$

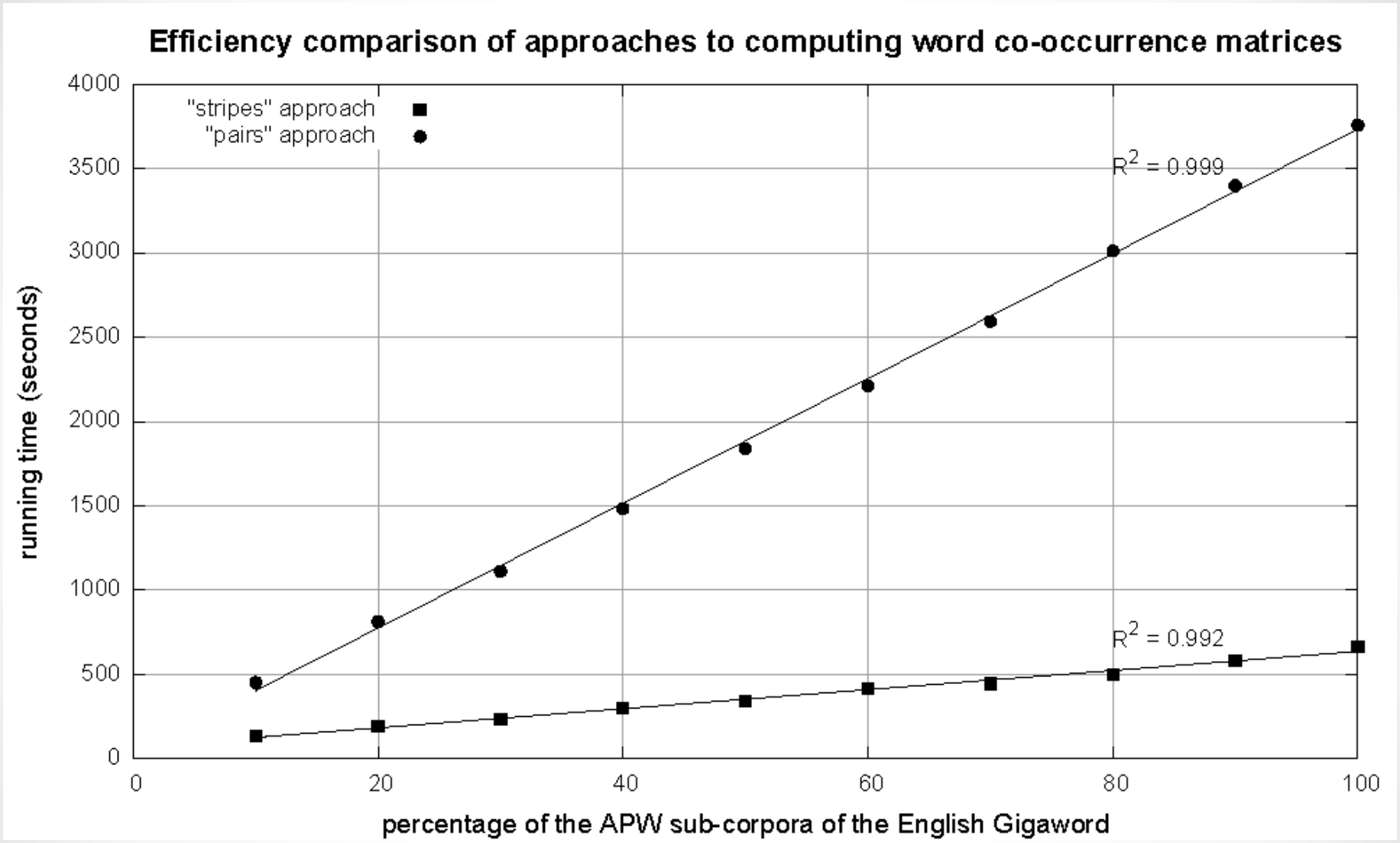
$a \rightarrow \{ b: 1, c: 2, d: 5, e: 3, f: 2 \}$

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For each term, emit $a \rightarrow \{ b: count_b, c: count_c, d: count_d \ldots \}$
- Reducers perform element-wise sum of associative arrays

$$
\begin{array}{rl}
 & a \rightarrow \{ b: 1, \quad d: 5, e: 3 \} \\
+ & a \rightarrow \{ b: 1, c: 2, d: 2, \quad f: 2 \} \\
\hline
 & a \rightarrow \{ b: 2, c: 2, d: 7, e: 3, f: 2 \}
\end{array}
$$

"Stripes" Analysis

- Advantages
  - Far less sorting and shuffling of key-value pairs
  - Can make better use of combiners
- Disadvantages
  - More difficult to implement
  - Underlying object is more heavyweight
  - Fundamental limitation in terms of size of event space



Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

Relative frequency estimates

- How do we compute relative frequencies from counts?

$$P(B \mid A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}$$

- Why do we want to do this?
- How do we do this with MapReduce?

P(B|A): "Pairs"

$(a, *) \rightarrow 32$ **Reducer holds this value in memory**

$(a, b_1) \rightarrow 3$
$(a, b_2) \rightarrow 12$
$(a, b_3) \rightarrow 7$
$(a, b_4) \rightarrow 1$
…

$(a, b_1) \rightarrow 3 / 32$
$(a, b_2) \rightarrow 12 / 32$
$(a, b_3) \rightarrow 7 / 32$
$(a, b_4) \rightarrow 1 / 32$
…

- For this to work:
  - Must emit extra (a, *) for every b_n in mapper
  - Must make sure all a's get sent to same reducer (use partitioner)
  - Must make sure (a, *) comes first (define sort order)
  - Must hold state in reducer across different key-value pairs

P(B | A): "Stripes"

$a \rightarrow \{b_1:3, b_2:12, b_3:7, b_4:1, \ldots\}$

- Easy!
  - One pass to compute (a, *)
  - Another pass to directly compute $f(B \mid A)$

Synchronization in MapReduce

- Approach 1: turn synchronization into an ordering problem
  - Sort keys into correct order of computation
  - Partition key space so that each reducer gets the appropriate set of partial results
  - Hold state in reducer across multiple key-value pairs to perform computation
  - Illustrated by the "pairs" approach
- Approach 2: construct data structures that "bring the pieces together"
  - Each reducer receives all the data it needs to complete the computation
  - Illustrated by the "stripes" approach

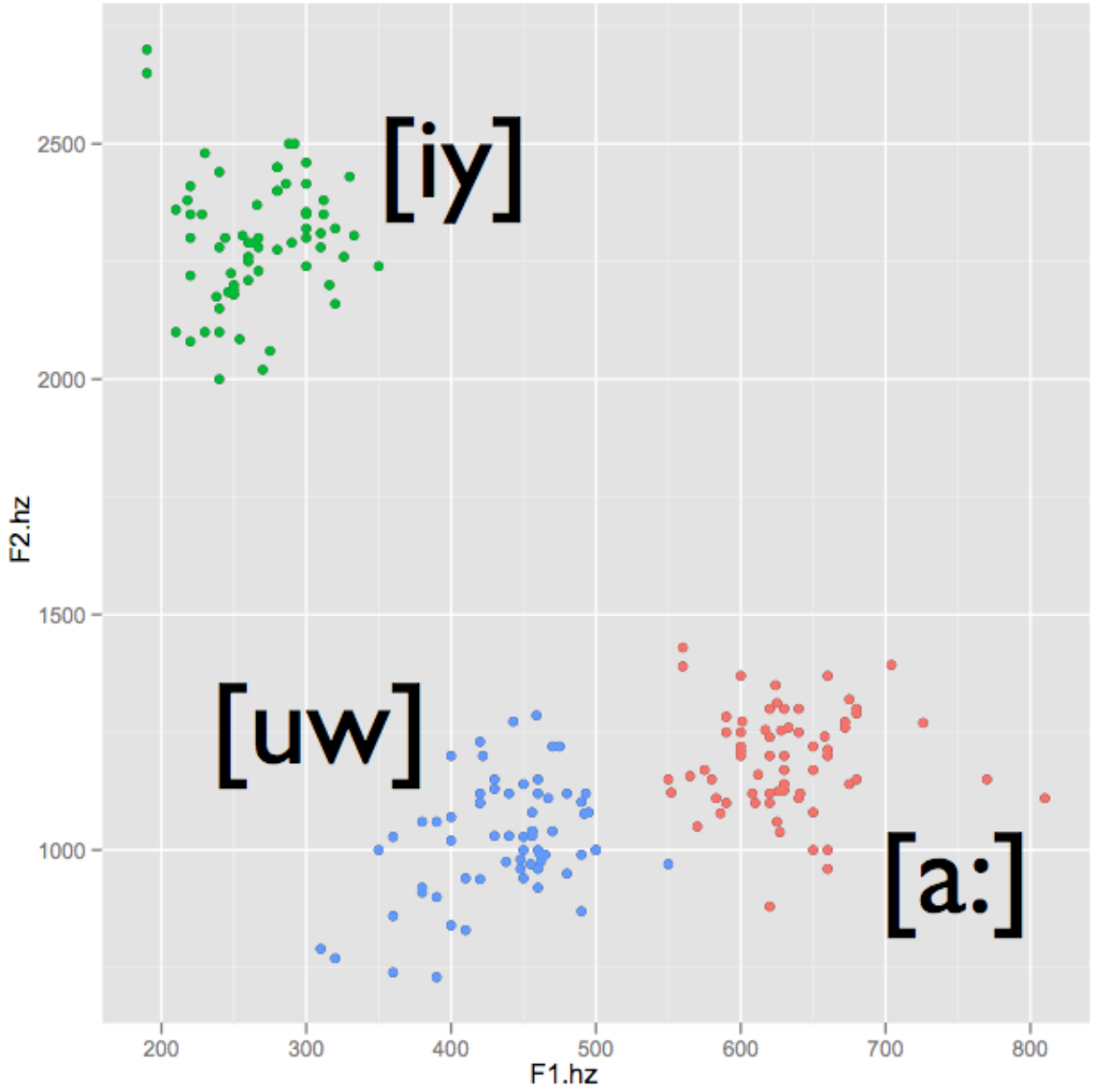
Issues and Tradeoffs

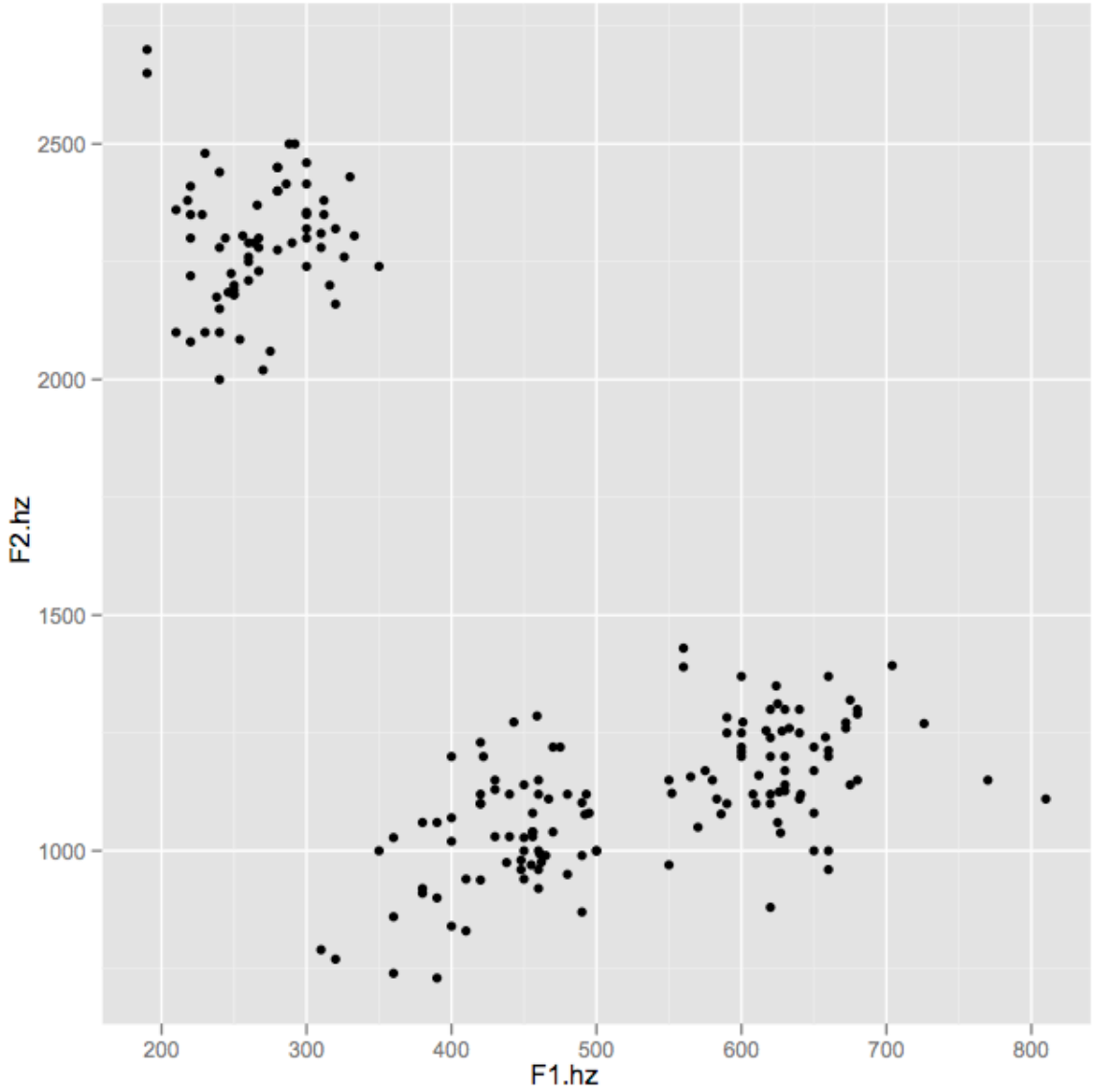
- Number of key-value pairs
  - Object creation overhead
  - Time for sorting and shuffling pairs across the network
  - In Hadoop, every object emitted from a mapper is written to disk
- Size of each key-value pair
  - De/serialization overhead
- Combiners make a big difference!
  - RAM vs. disk and network
  - Arrange data to maximize opportunities to aggregate partial results

Batch Learning Algorithms in MR

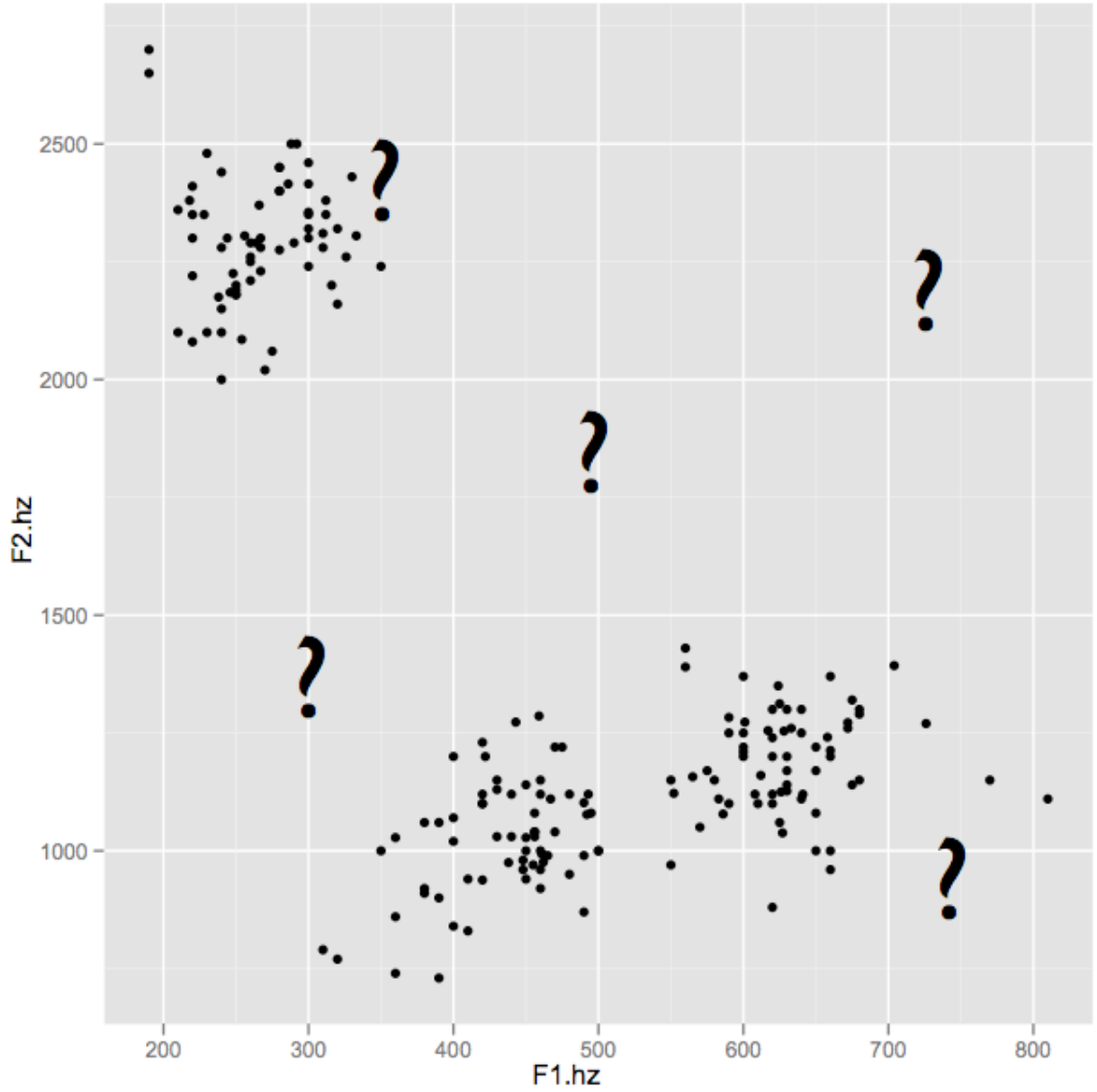
Batch Learning Algorithms

- Expectation maximization
  - Gaussian mixtures / *k*-means
  - Forward-backward learning for HMMs
- Gradient-ascent based learning
  - Computing (gradient, objective) using MapReduce
  - Optimization questions

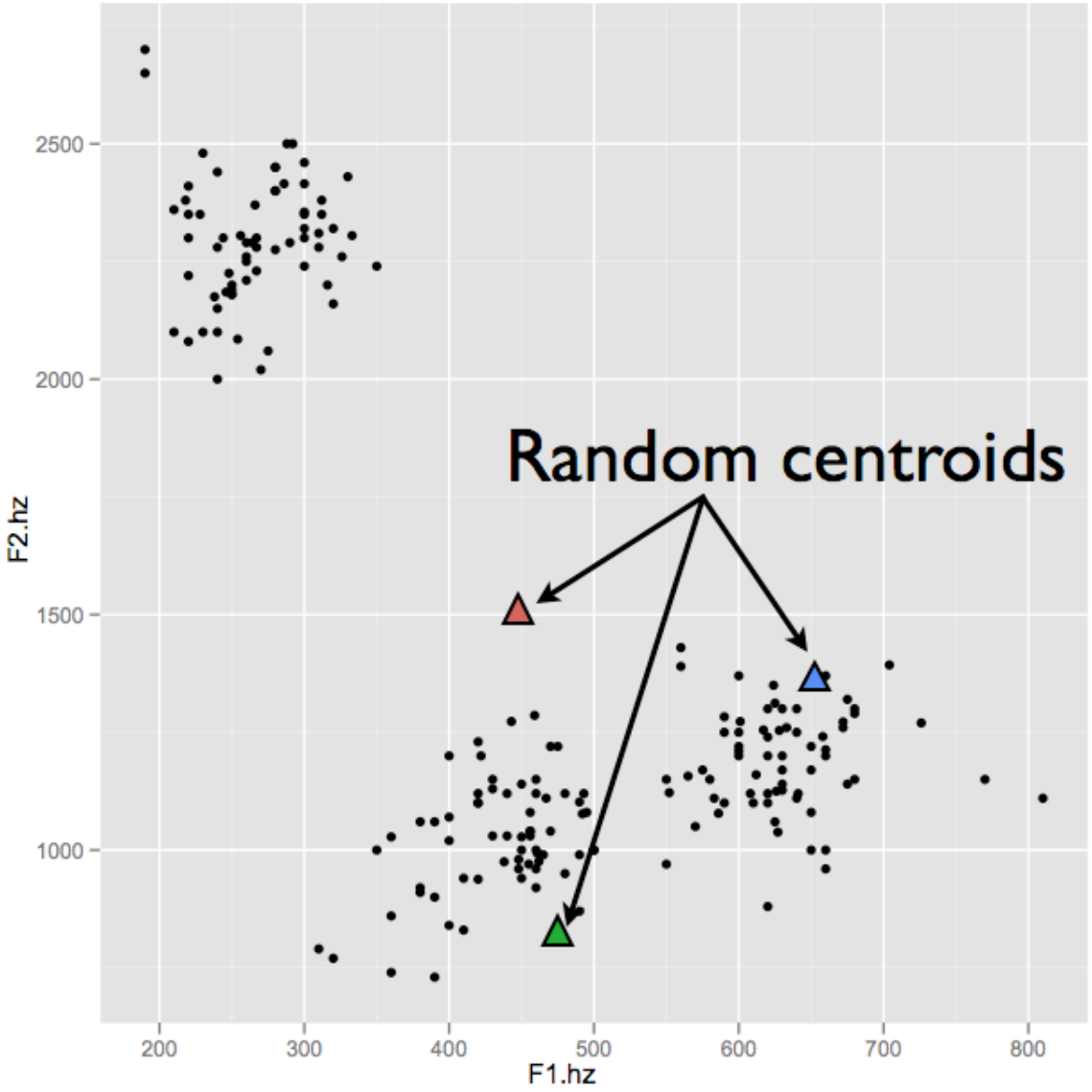

[iy]
[uw]
[a:]
2500
2000
1500
1000
F2.hz
200
300
400
500
600
700
800
F1.hz

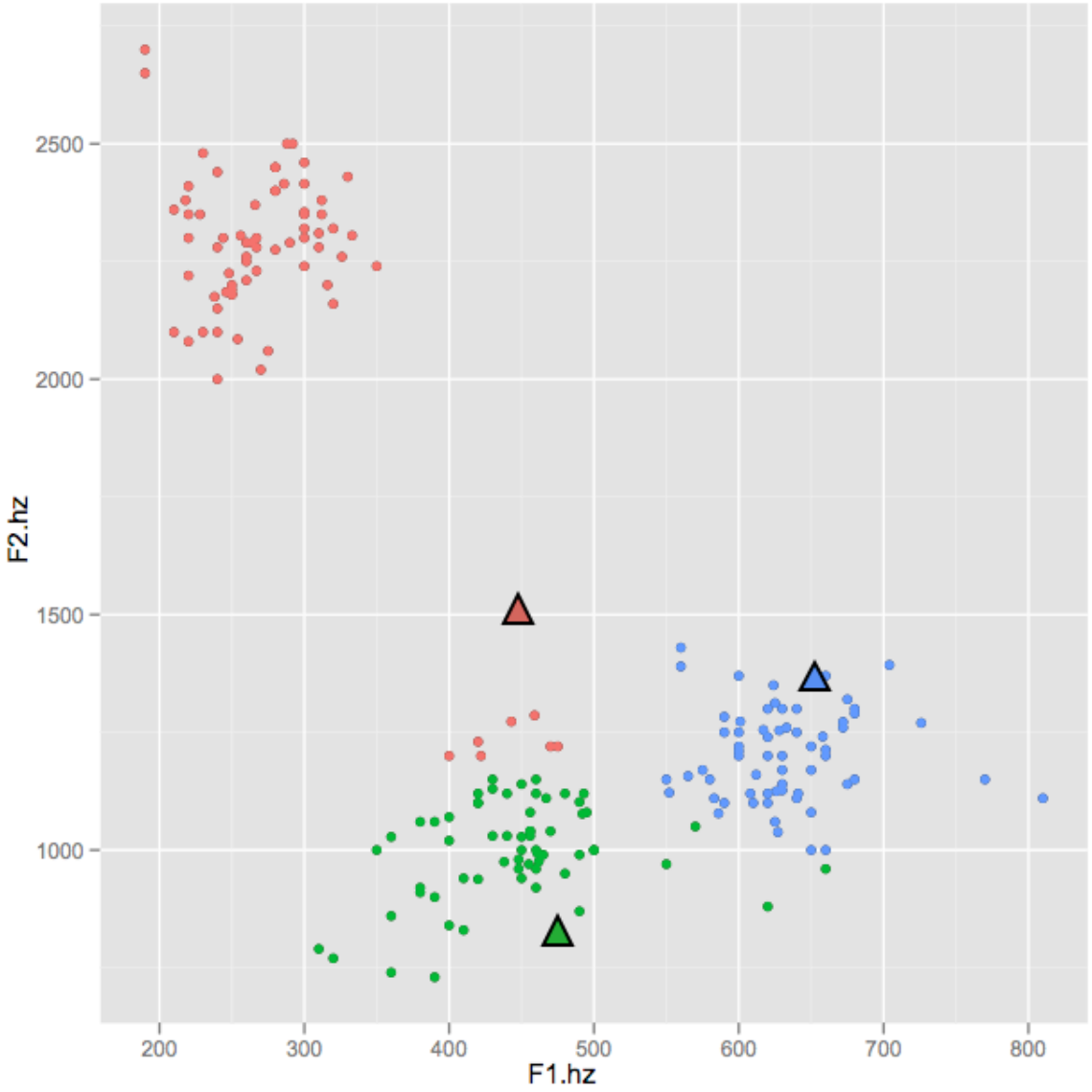


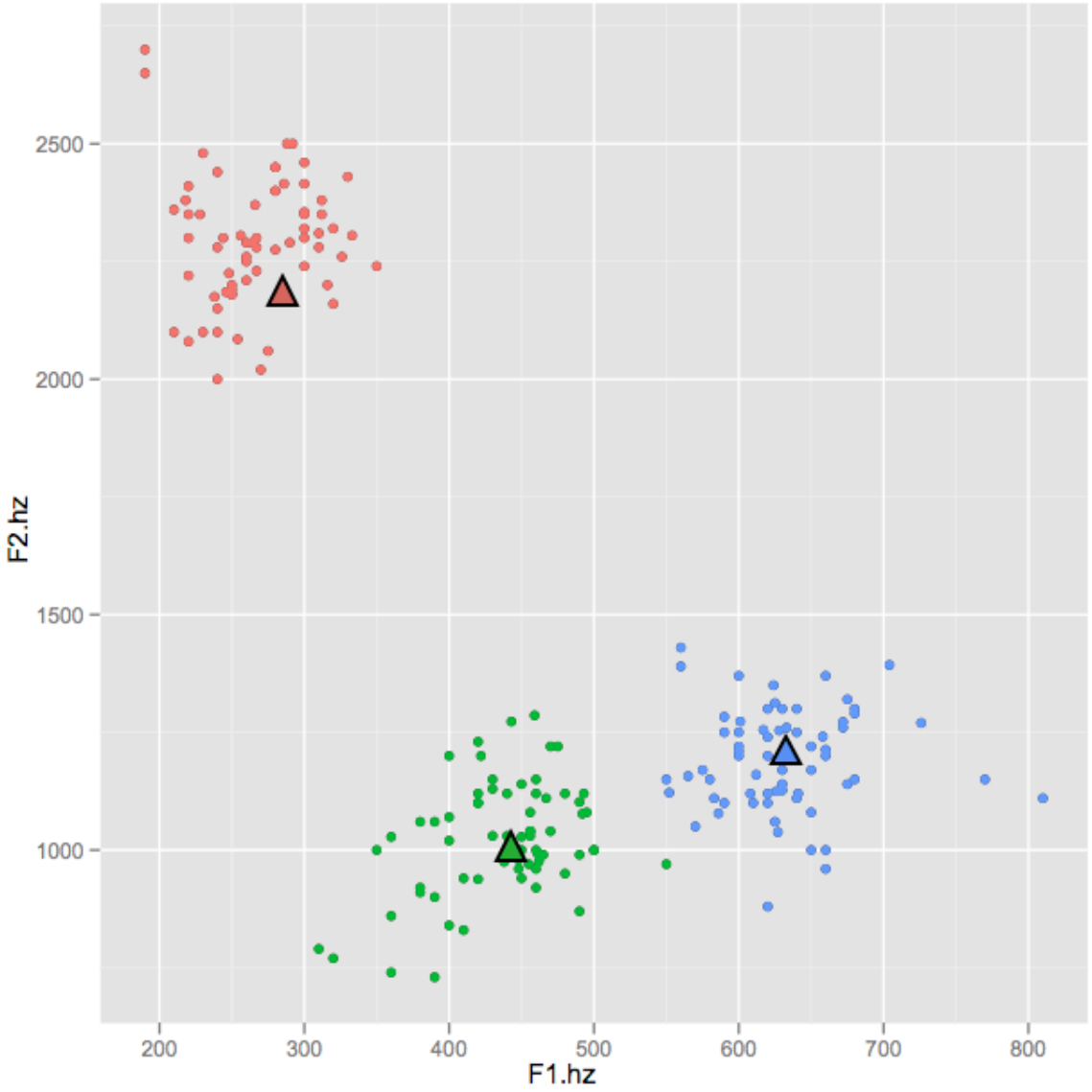
But what if the data look like this?

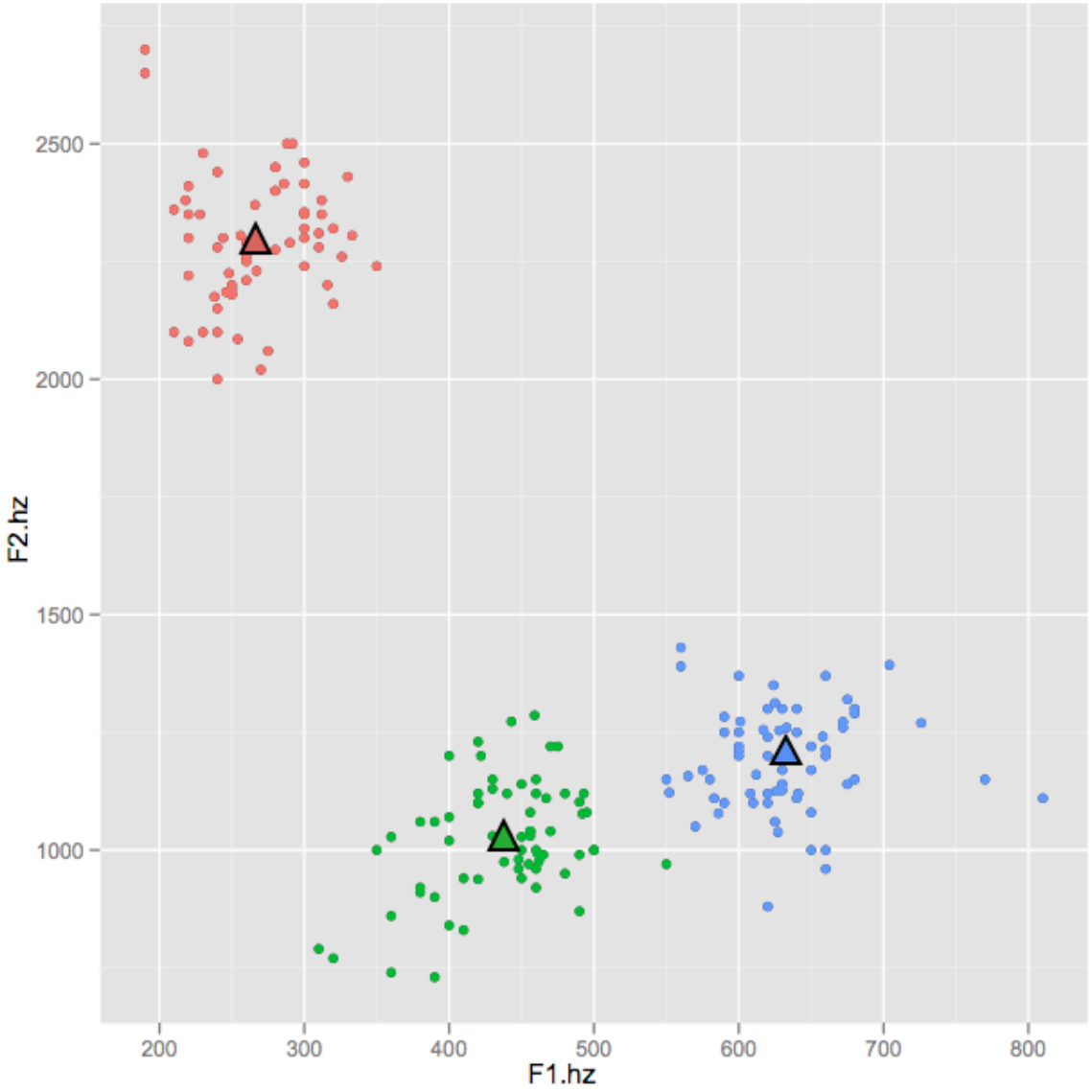


How do we learn with no labels??


Random centroids
F2.hz
F1.hz
2500
2000
1500
1000
200
300
400
500
600
700
800


2500
2000
1500
1000
F2.hz
200
300
400
500
600
700
800
F1.hz


2500
2000
1500
1000
F2.hz
200
300
400
500
600
700
800
F1.hz


2500
2000
1500
1000
F2.hz
200
300
400
500
600
700
800
F1.hz

k-means on MR

- Until convergence
  - Distribute parameters (centroids) to all worker nodes
  - Mappers
    - Assign labels to each data point
    - Produce a (label, point) output
  - Reducers
    - Compute centroids for each label
    - Can be as many reducers as cardinality of label set
- Every k-means iteration is one map-reduction

EM Assumptions

- **E-step**
  - Compute posterior distribution over latent variable
  - In the case of k-means, that's the class label, given the data and the parameters
- **M-step**
  - Solving an optimization problem given the posteriors over latent variables
  - K-means, HMMs, PCFGs: analytic solutions
  - Not always the case

EM Algorithms in MapReduce

E step (mappers)

Compute the posterior distribution over the latent variables y and the current parameters $\boldsymbol{\theta}^{(t)}$

$$\mathbb{E}[\mathbf{y}] = p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{(t)})$$

M step (reducer)

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbb{E}[\mathbf{y}])$$

EM Algorithms in MapReduce

E step (mappers)

Compute the posterior distribution over the latent variables y and the current parameters $\boldsymbol{\theta}^{(t)}$

$$\mathbb{E}[\mathbf{y}] = p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{(t)})$$

Cluster labels **Data points**

M step (reducer)

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbb{E}[\mathbf{y}])$$

EM Algorithms in MapReduce

E step (mappers)

Compute the posterior distribution over the latent variables y and the current parameters $\boldsymbol{\theta}^{(t)}$

$$\mathbb{E}[\mathbf{y}] = p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{(t)})$$

State sequence **Words**

M step (reducer)

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbb{E}[\mathbf{y}])$$

EM Algorithms in MapReduce

E step

Compute the expected log likelihood with respect to the conditional distribution of the latent variables with respect to the observed data.

$$\mathbb{E}[\mathbf{y}] = p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{(t)})$$

Expectations are just sums of function evaluation over an event times that event's probability: *perfect for MapReduce*!

Mappers compute model likelihood given small pieces of the training data (scale EM to large data sets!)

EM Algorithms in MapReduce

M step

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbb{E}[\mathbf{y}])$$

The solution to this problem depends on the parameterization used. For HMMs, PCFGs with multinomial parameterizations, this is just computing the relative frequency.

- Easy!
  - One pass to compute (a, *)
  - Another pass to directly compute *f*(B | A)

$$a \rightarrow \{b_1{:}3,\ b_2{:}12,\ b_3{:}7,\ b_4{:}1,\ \ldots\}$$

Challenges

- Each iteration of EM is one MapReduce job
- Mappers require the current model parameters
  - Certain models may be very large
  - Optimization: any particular piece of the training data probably depends on only a small subset of these parameters
- Reducers may aggregate data from many mappers
  - Optimization: Make smart use of combiners!

Log-linear Models

- NLP's favorite discriminative model:

$$p(y|x) = \frac{1}{Z(x)} \exp \sum_i \lambda_i h_i(x, y)$$

- Applied successfully to classificiation, POS tagging, parsing, MT, word segmentation, named entity recognition, LM…
  - Make use of millions of features (h_i's)
  - Features may overlap
  - Global optimum easily reachable, assuming no latent variables

Exponential Models in MapReduce

- Training is usually done to maximize likelihood (minimize negative llh), using first-order methods
  - Need an objective and gradient with respect to the parameterizes that we want to optimize

$$\mathcal{L}(\theta) = -\sum_{\langle x,y \rangle} \log p(y|x;\theta)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{\langle x,y \rangle} \mathbf{E}_{p(y'|x;\theta)} \left[h_i(x,y') \right] - h_i(x,y)$$

Exponential Models in MapReduce

- How do we compute these in MapReduce?

$$\mathcal{L}(\theta) = -\sum_{\langle x,y \rangle} \log p(y|x;\theta)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{\langle x,y \rangle} \mathbf{E}_{p(y'|x;\theta)} \left[h_i(x,y') \right] - h_i(x,y)$$

As seen with EM: expectations map nicely onto the MR paradigm.

Each mapper computes two quantities: the LLH of a training instance <x,y> under the current model and the contribution to the gradient.

Exponential Models in MapReduce

- What about reducers?

$$\mathcal{L}(\theta) = -\sum_{\langle x,y \rangle} \log p(y|x;\theta)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{\langle x,y \rangle} \mathbf{E}_{p(y'|x;\theta)} \left[h_i(x,y') \right] - h_i(x,y)$$

The objective is a single value – make sure to use a combiner!

The gradient is as large as the feature space – but may be quite sparse. Make use of sparse vector representations!

Exponential Models in MapReduce

- After one MR pair, we have an objective and gradient
- Run some optimization algorithm
  - LBFGS, gradient descent, etc…
- Check for convergence
- If not, re-run MR to compute a new objective and gradient

Challenges

- Each iteration of training is one MapReduce job
- Mappers require the current model parameters
- Reducers may aggregate data from many mappers
- Optimization algorithm (LBFGS for example) may require the full gradient
  - This is okay for millions of features
  - What about billions?
  - …or trillions?

Case study:
statistical machine translation

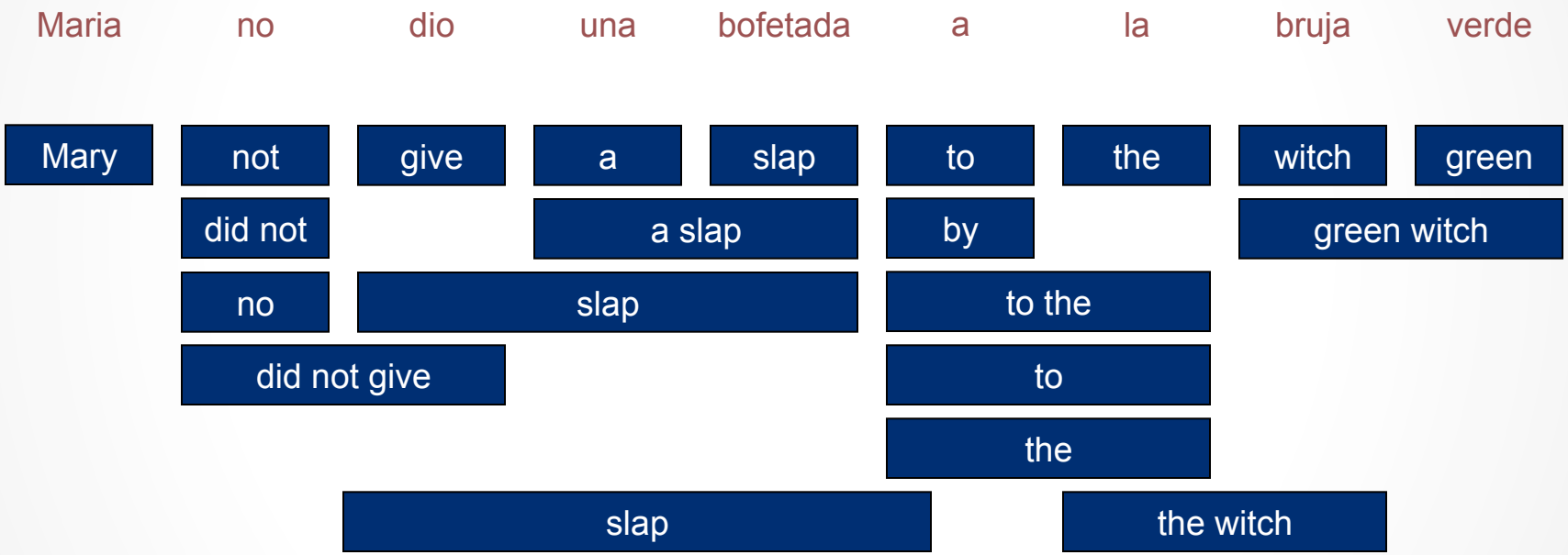
Statistical Machine Translation

- Conceptually simple:
  (translation from foreign f into English e)

$$\hat{e} = \arg\max_{e} P(f \mid e) P(e)$$

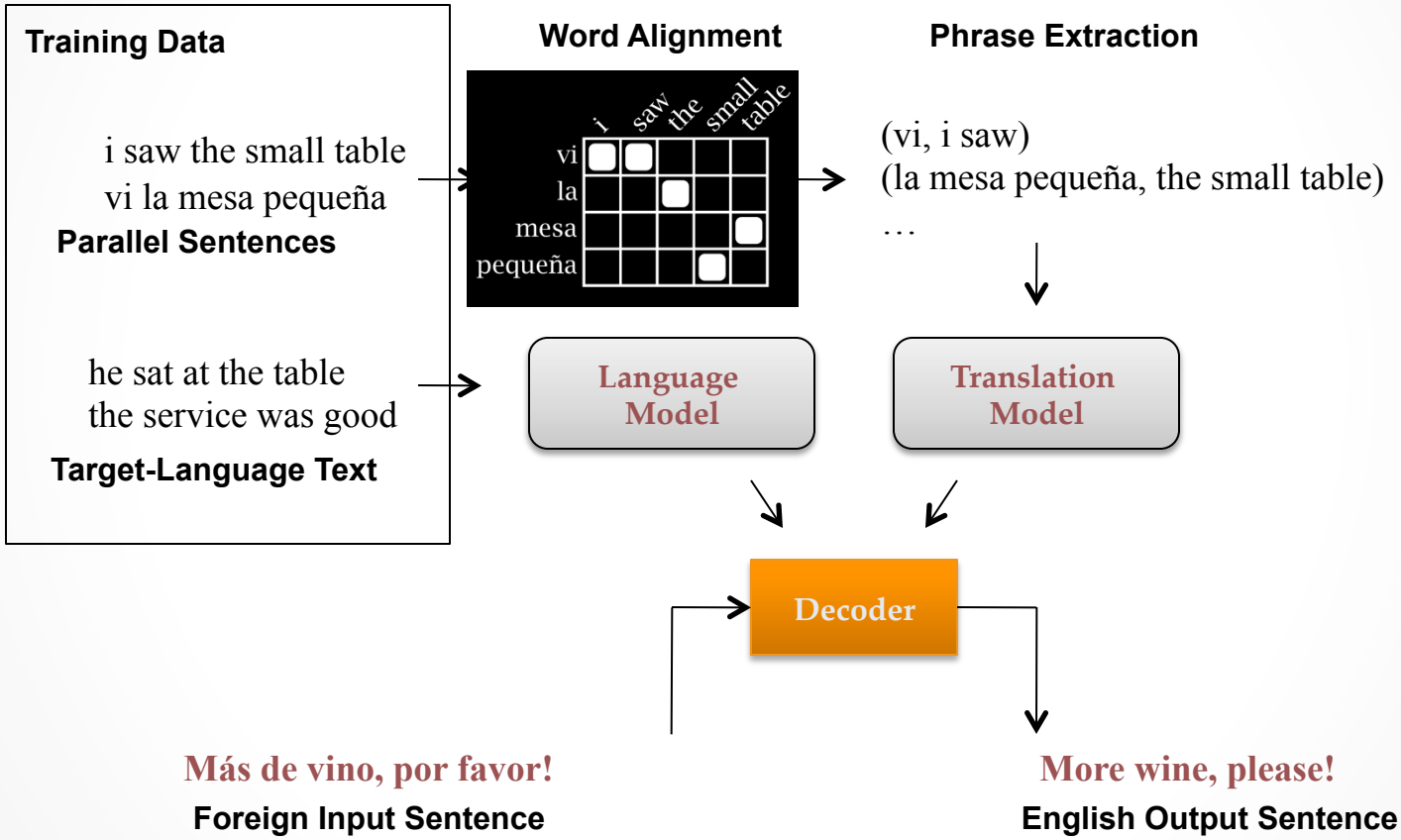
- Difficult in practice!
- Phrase-Based Machine Translation (PBMT) :
  - Break up source sentence into little pieces (phrases)
  - Translate each phrase individually

Dyer et al. (Third ACL Workshop on MT, 2008)

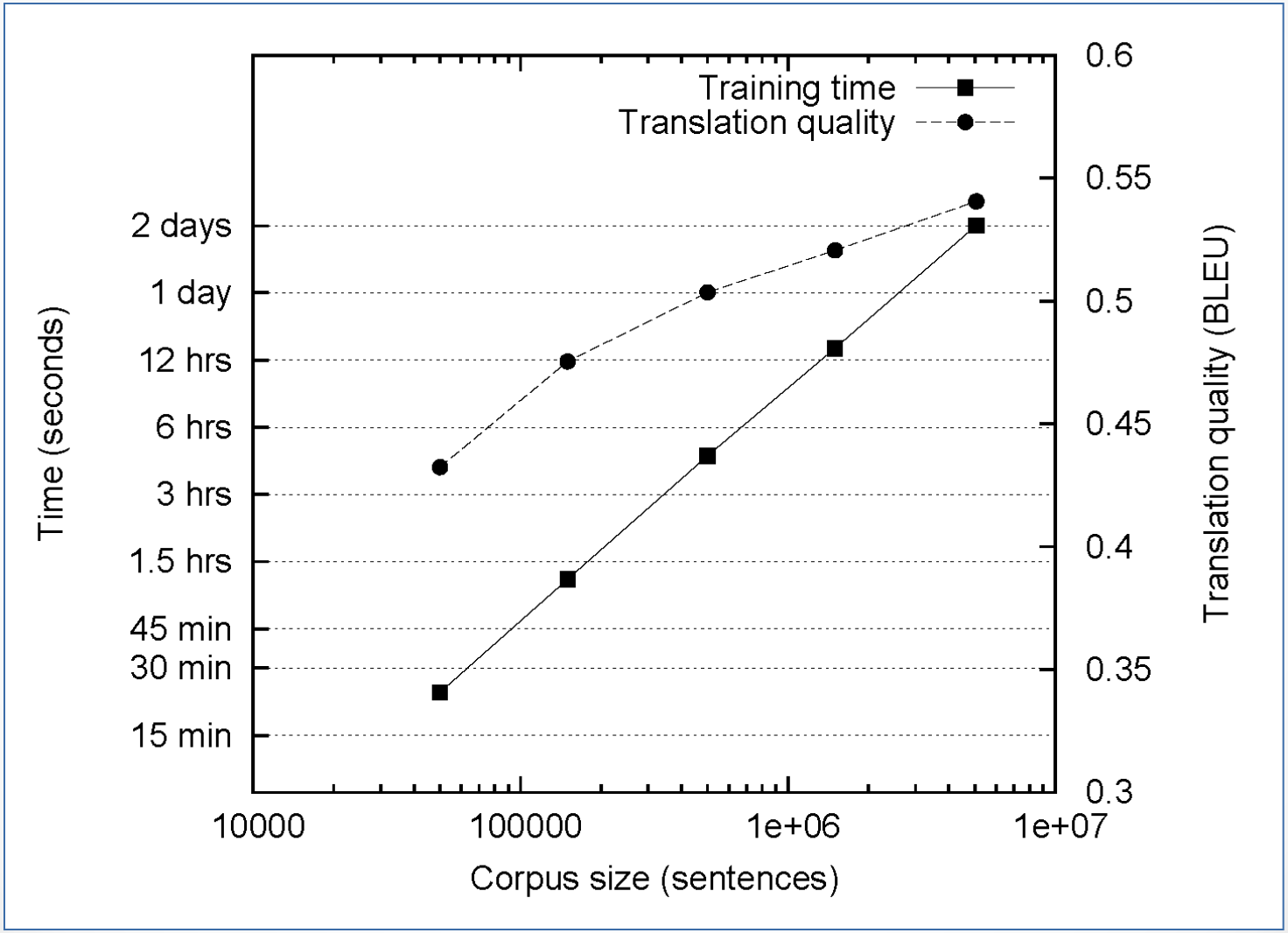


Example from Koehn (2006)

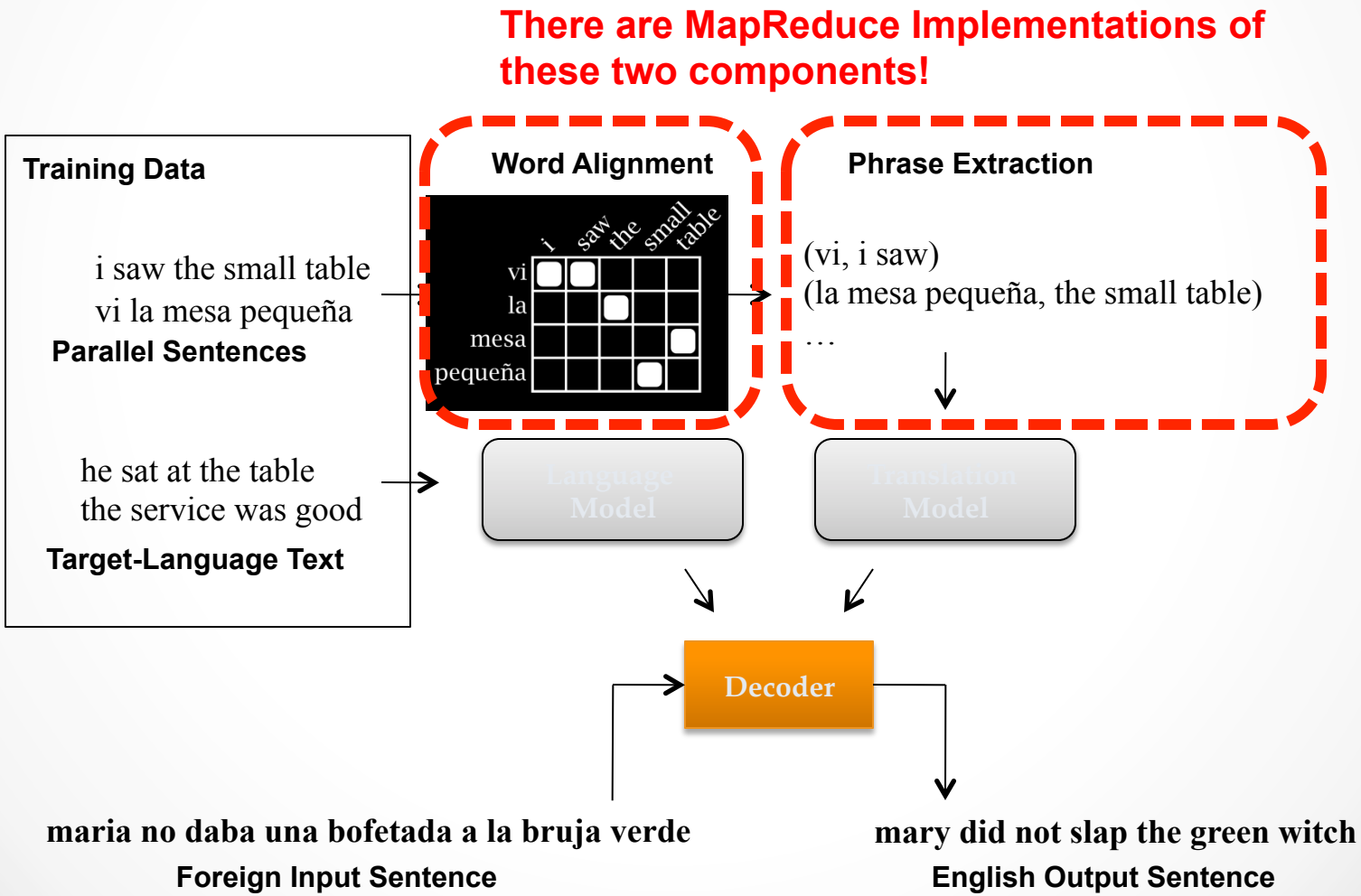
MT Architecture



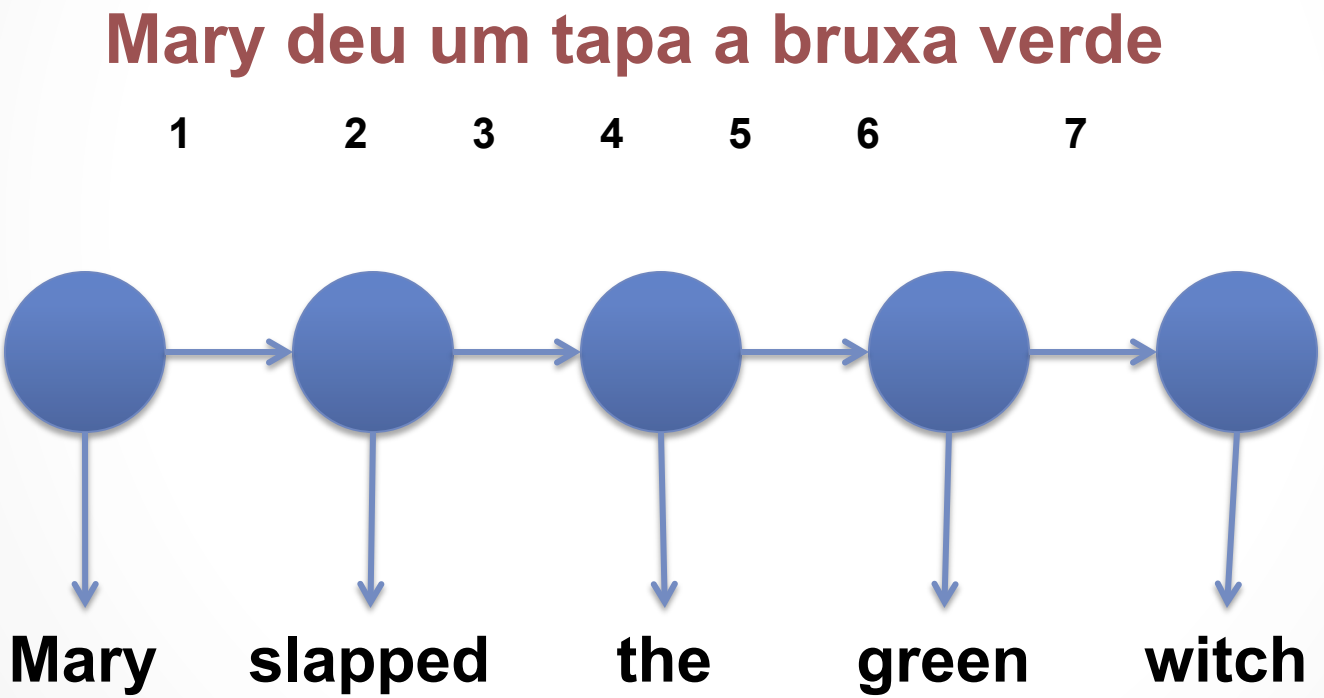
The Data Bottleneck



MT Architecture



Alignment with HMMs

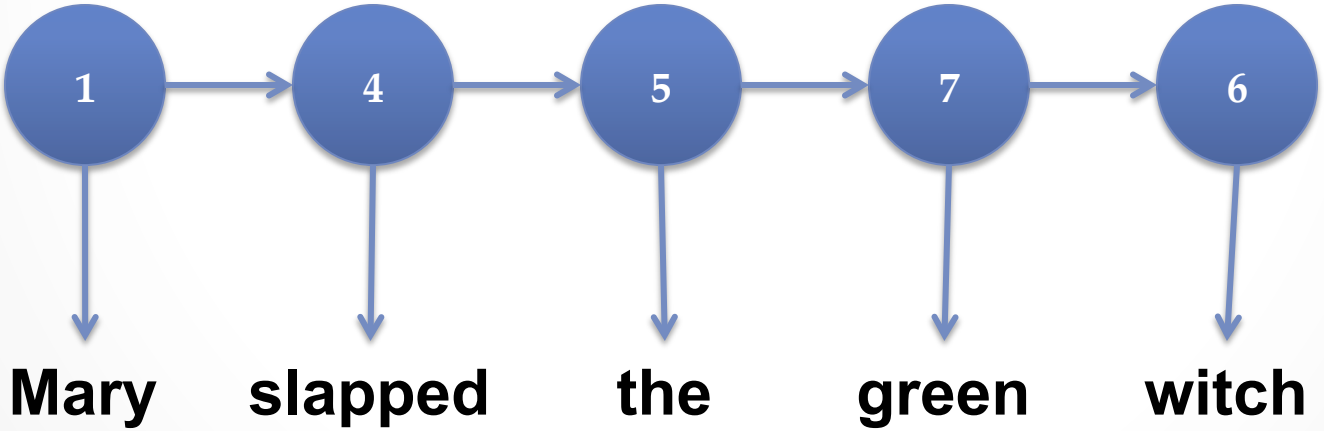


Vogel et al. (1996)

Alignment with HMMs

Mary deu um tapa a bruxa verde

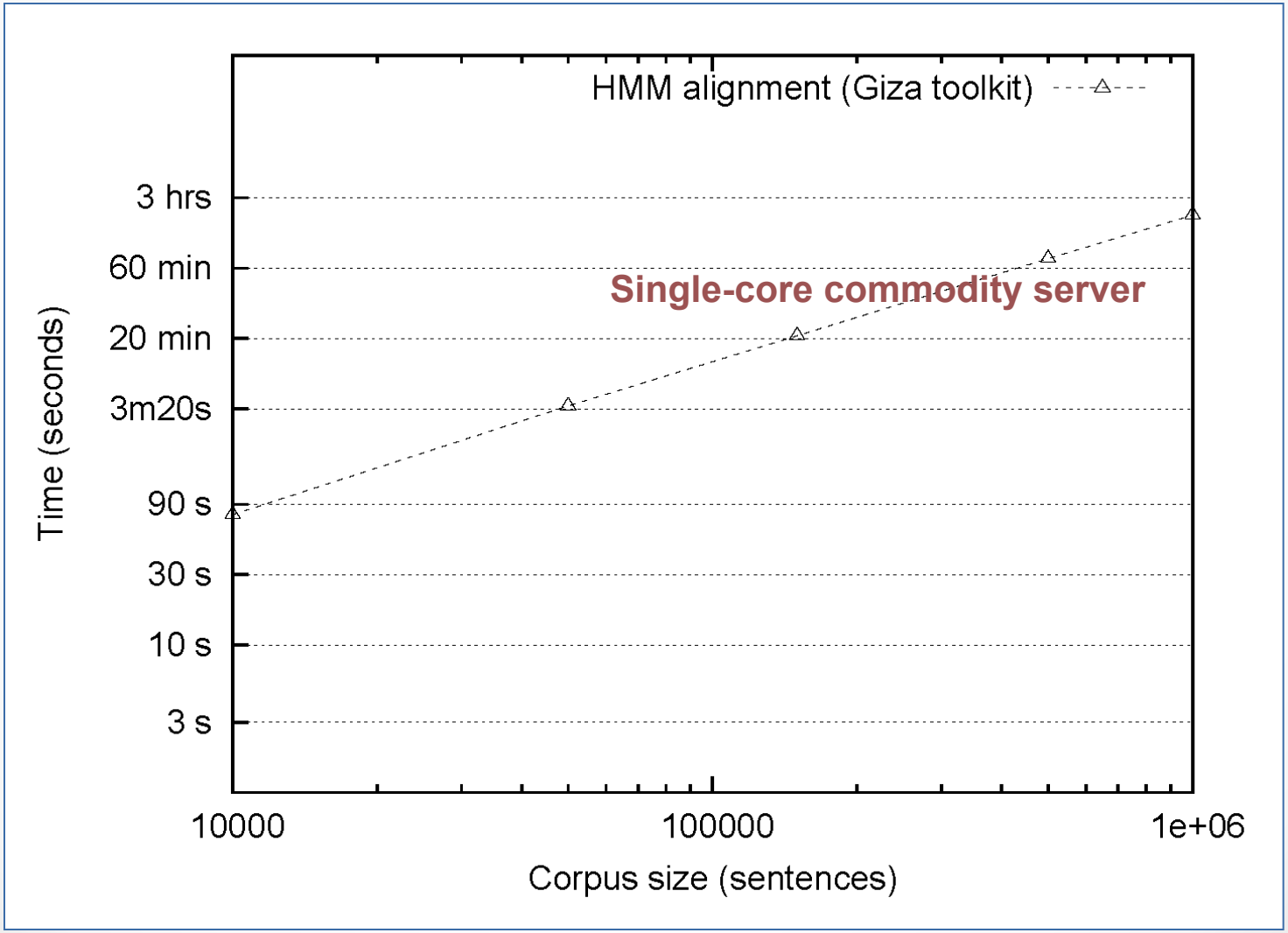
1 2 3 4 5 6 7



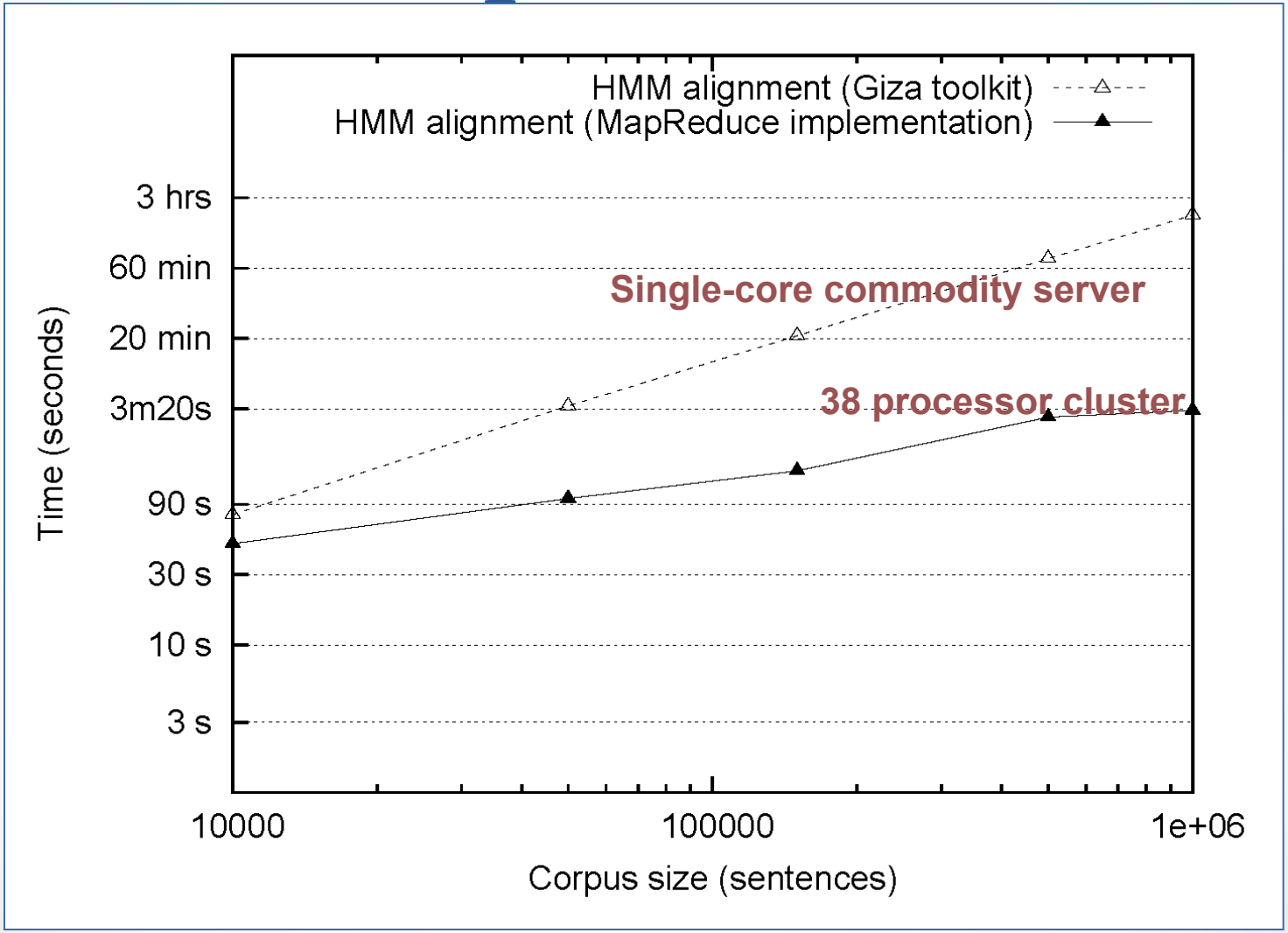
Alignment with HMMs

- Emission parameters: translation probabilities
- States: words in source sentence
- Transition probabilities: probability of jumping +1, +2, +3, -1, -2, etc.
- Alternative parameterization of state probabilities:
  - Uniform ("Model 1")
  - Independent of previous alignment decision, dependent only on global position in sentence ("Model 2")
  - Many other models…
- This is still state-of-the-art in Machine Translation
- How many parameters are there?

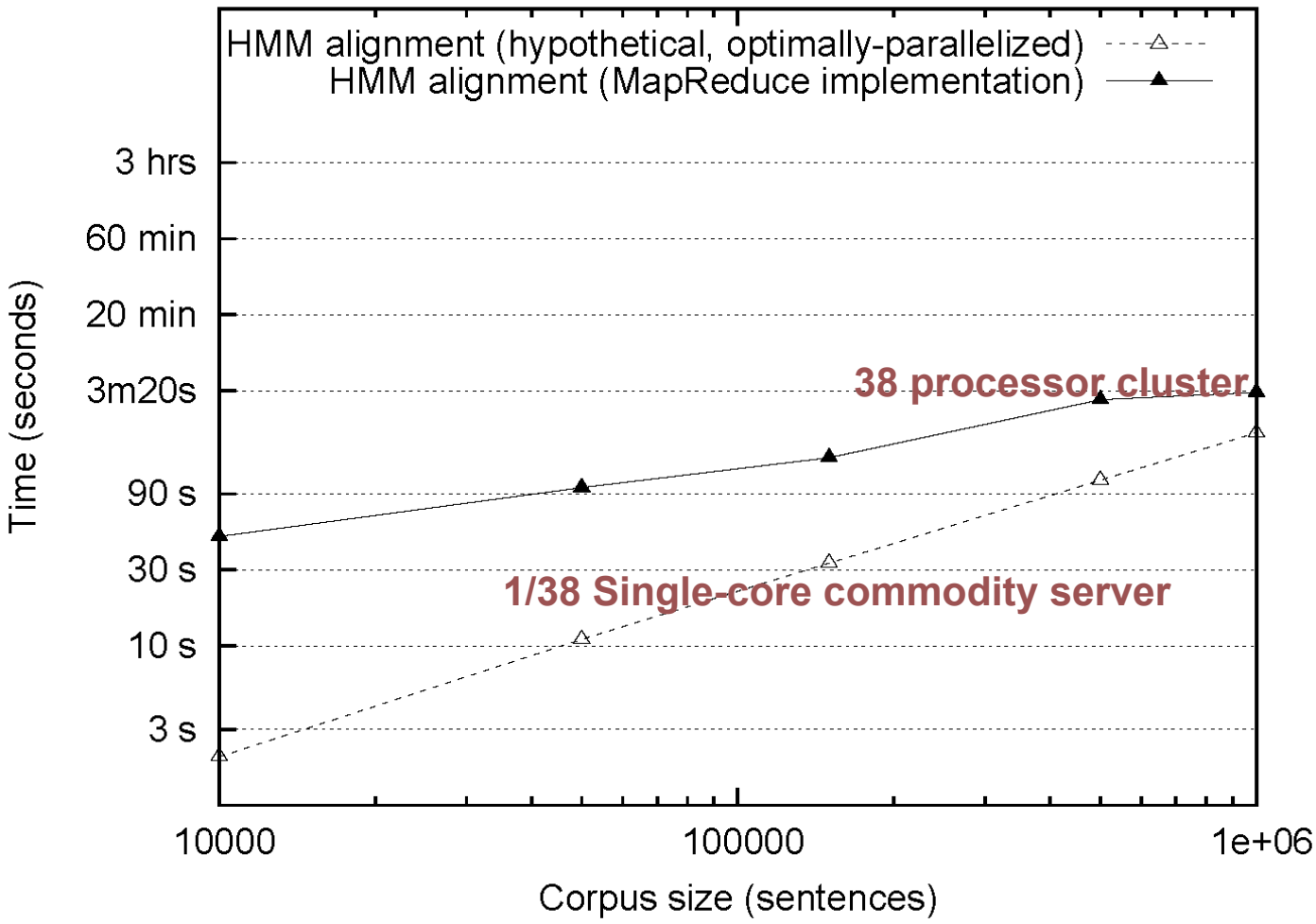
HMM Alignment: Giza



HMM Alignment: MapReduce



HMM Alignment: MapReduce



Online Learning Algorithms in MR

Online Learning Algorithms

- MapReduce is a batch processing system
  - Associativity used to factor large computations into subproblems that can be solved
  - Once job is running, processes are independent
- Online algorithms
  - Great deal of research currently in ML
  - Theory: strong convergence, mistake guarantees
  - Practice: rapid convergence
  - Good (empirical) performance on nonconvex problems
  - **Problems**
    - Parameter updates made **sequentially** after each training instance
    - Theoretical analysis rely on sequential model of computation
    - **How do we parallelize this?**

Review: Perceptron

Sequential perceptron algorithm.

algorithm PERCEPTRON($\boldsymbol{w}^{(0)}, \mathcal{T} = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}_{i=1}^{\ell}, N$) ▷ Run N iterations of perceptron training
  $k \leftarrow 0$ ▷ k is the mistake counter
  for $i \leftarrow 1 \ldots N$ **do**
    for $t \leftarrow 1 \ldots \ell$ **do**
      $\mathbf{y}' \leftarrow \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_t)} \left[\boldsymbol{w}^{(k)} \cdot \Phi(\mathbf{x}_t, \mathbf{y})\right]$ ▷ Viterbi or other max inference algorithm
      if $\mathbf{y}' \neq \mathbf{y}_t$ **then**
        $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} + \Phi(\mathbf{x}_t, \mathbf{y}_t) - \Phi(\mathbf{x}_t, \mathbf{y}')$
        $k \leftarrow k + 1$
  return $\boldsymbol{w}^{(k)}$

Collins (2002), Rosenblatt (1957)

Review: Perceptron

Sequential perceptron algorithm.

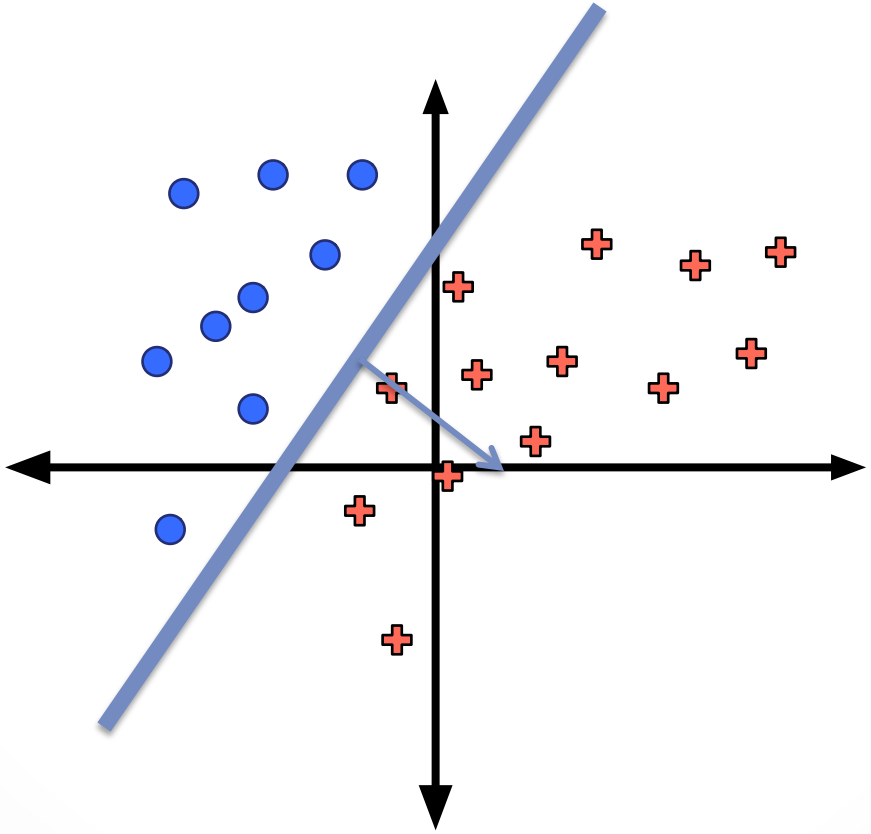
algorithm PERCEPTRON($\boldsymbol{w}^{(0)}, \mathcal{T} = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}_{i=1}^{\ell}, N$) ▷ Run N iterations of perceptron training
  $k \leftarrow 0$ ▷ k is the mistake counter
  for $i \leftarrow 1 \ldots N$ **do**
    for $t \leftarrow 1 \ldots \ell$ **do**
      $\mathbf{y}' \leftarrow \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_t)} \left[\boldsymbol{w}^{(k)} \cdot \Phi(\mathbf{x}_t, \mathbf{y})\right]$ ▷ Viterbi or other max inference algorithm
      if $\mathbf{y}' \neq \mathbf{y}_t$ **then**
        $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} + \Phi(\mathbf{x}_t, \mathbf{y}_t) - \Phi(\mathbf{x}_t, \mathbf{y}')$
        $k \leftarrow k + 1$
  return $\boldsymbol{w}^{(k)}$

Sequential perceptron algorithm with averaging.

algorithm AVGPERCEPTRON($\boldsymbol{w}^{(0)}, \mathcal{T} = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}_{i=1}^{\ell}, N$) ▷ Run N iterations of perceptron training
  $\boldsymbol{v} \leftarrow \mathbf{0}$
  $k \leftarrow 0$ ▷ k is the mistake counter
  for $i \leftarrow 1 \ldots N$ **do**
    for $t \leftarrow 1 \ldots \ell$ **do**
      ★ $\boldsymbol{v} \leftarrow \boldsymbol{v} + \boldsymbol{w}^{(k)}$ ▷ Keep track of weight vectors.
      $\mathbf{y}' \leftarrow \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_t)} \left[\boldsymbol{w}^{(k)} \cdot \Phi(\mathbf{x}_t, \mathbf{y})\right]$ ▷ Viterbi or other max inference algorithm
      if $\mathbf{y}' \neq \mathbf{y}_t$ **then**
        $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} + \Phi(\mathbf{x}_t, \mathbf{y}_t) - \Phi(\mathbf{x}_t, \mathbf{y}')$
        $k \leftarrow k + 1$
  return $\frac{\boldsymbol{v}}{N\ell}$ ★

Collins (2002), Rosenblatt (1957)

Assume separability



Algorithm 1: Parameter Mixing

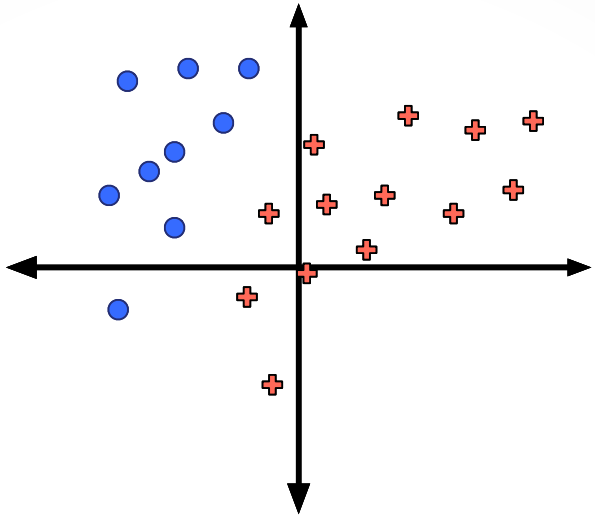
- Inspired by the averaged perceptron, let's run P perceptrons on **shards** of the data
- Return the average parameters

Perceptron algorithm with parameter mixing

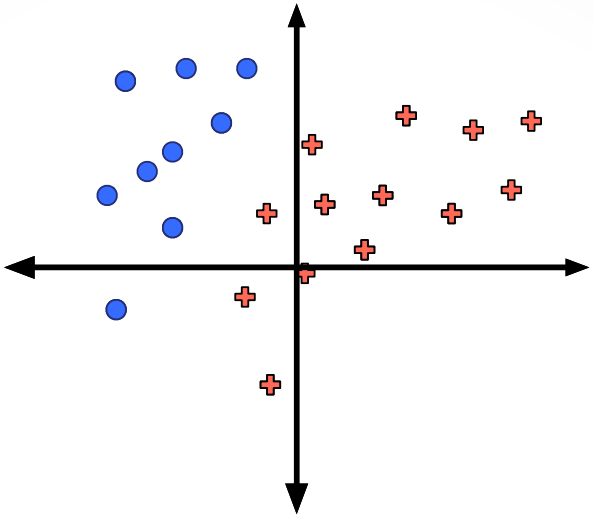
algorithm PARAMMIX($\mathcal{T} = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}_{i=1}^{\ell}, N, P$) ▷ Run perceptron on P shards and mix
  Shard $\mathcal{T}$ into P shards $\mathcal{T}_P = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_P\}$
  for $p \leftarrow 1 \ldots P$ **do** ▷ Execute loop in parallel
    $\boldsymbol{w}^{(p)} \leftarrow$ PERCEPTRON($\mathbf{0}, \mathcal{T}_p, N$)
  return $\frac{\sum_{p=1}^{P} \boldsymbol{w}^{(p)}}{P}$

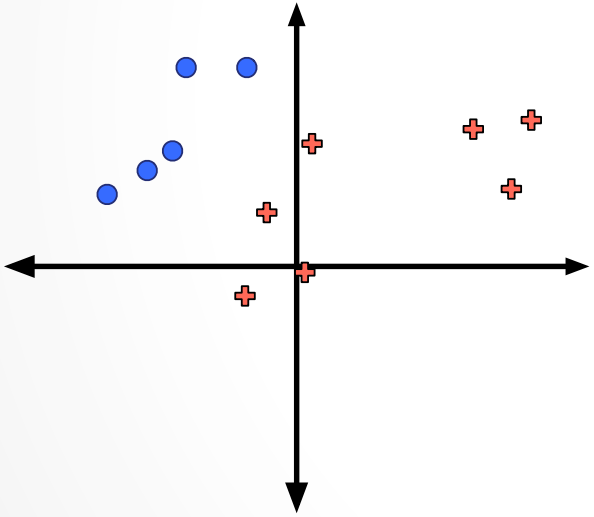
Return average parameters from P runs

McDonald, Hall, & Mann (NAACL, 2010)



Full Data

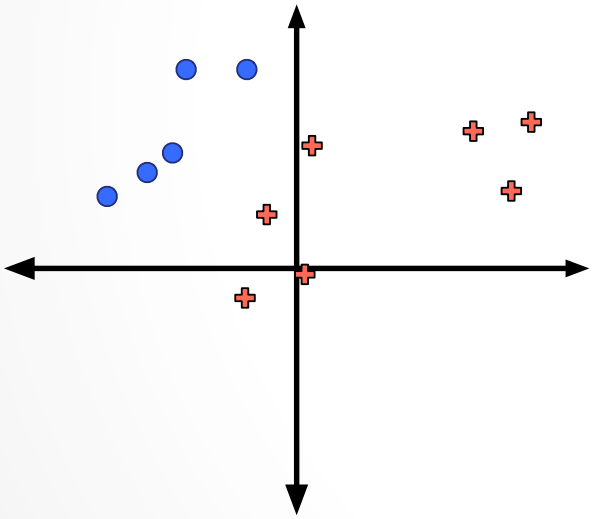




Shard 1

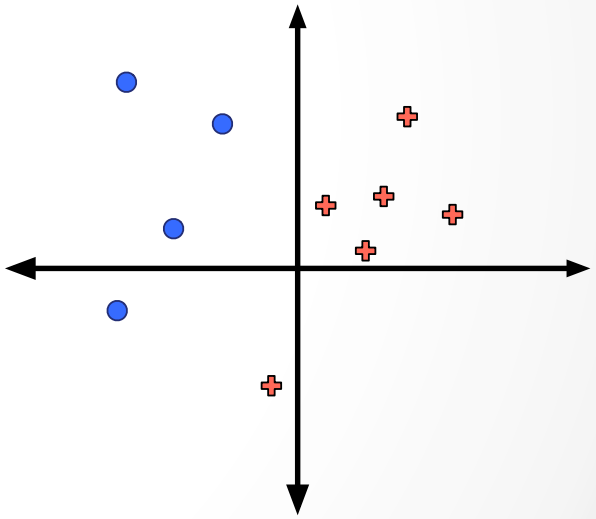
Shard 2

Processor 1



Shard 1

Processor 2



Shard 2

Attempt 1: Parameter Mixing

- Unfortunately…
- Theorem (proof by counterexample). For any training set T separable by margin γ, **ParamMix does not necessarily return a separating hyperplane**.

	Perceptron	Avg. Perceptron
Serial	85.8	88.8
1/P - serial	75.3	76.6
ParamMix	81.5	81.6

Researchers at Carnegie Mellon University in Pittsburgh, PA, have put together an iOS app, DrawAFriend, …

Algorithm 2: Iterative Mixing

- Rather than mixing parameters just once, mix after each epoch (pass through a shard)
- Redistribute parameters and use them as a starting point on next epoch

Perceptron algorithm with iterative parameter mixing

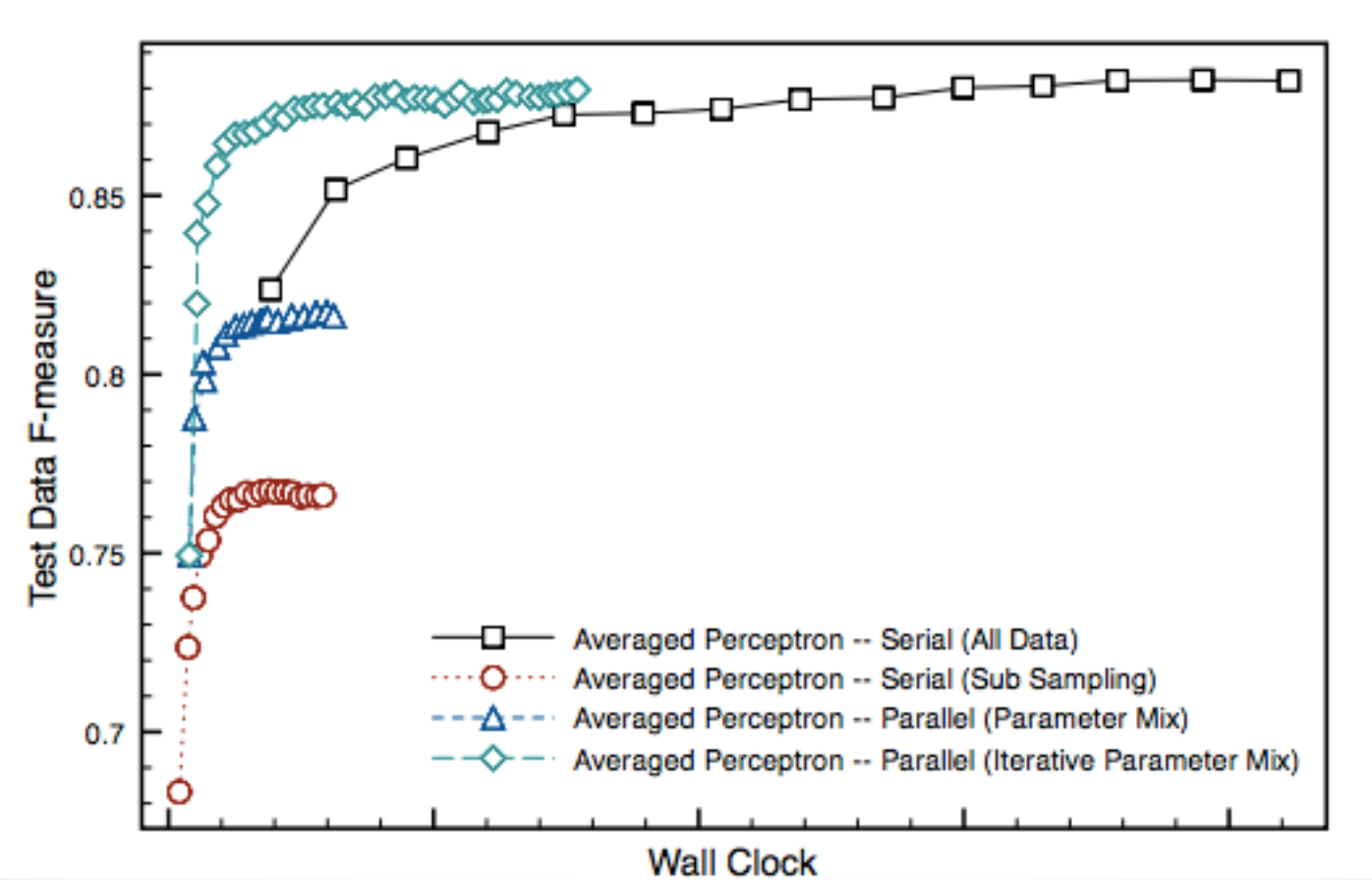
algorithm ITERPARAMMIX($\mathcal{T} = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}_{i=1}^{\ell}, N, P$) ▷ Sharded perceptron with iterative mixing
  Shard $\mathcal{T}$ into P shards $\mathcal{T}_P = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_P\}$
  $\boldsymbol{w}^{(0)} \leftarrow \mathbf{0}$
  for $i \leftarrow 1 \ldots N$ **do**
    for $p \leftarrow 1 \ldots P$ **do** ▷ Execute loop in parallel
      $\boldsymbol{v}^{(p)} \leftarrow$ PERCEPTRON($\boldsymbol{w}^{(i-1)}, \mathcal{T}_p, 1$) ▷ Do **one** epoch of perceptron training
    $\boldsymbol{w}^{(i)} \leftarrow \sum_p \frac{\boldsymbol{v}^{(p)}}{P}$
  return $\frac{\sum_i \boldsymbol{w}^{(i)}}{NP}$

Algorithm 2: Analysis

- Theorem. The weighted number of mistakes has is bound (like the sequential perceptron) by
  - The worst-case number of epochs (through the full data) is the same as the sequential perceptron
  - With non-uniform mixing, it is possible to obtain a speedup of P

	Perceptron	Avg. Perceptron
Serial	85.8	88.8
1/P - serial	75.3	76.6
ParamMix	81.5	81.6
IterParamMix	**87.9**	88.1

Algorithm 2: Empirical



Summary

- Two approaches to learning
  - Batch: process the entire dataset and make a single update
  - Online: sequential updates
- Online learning is hard to realize in a distributed (parallelized) environment
  - Parameter mixing approaches offer a theoretically sound solution with practical performance guarantees
  - Generalizations to log-loss and MIRA loss have also been explored
  - Conceptually some workers have "stale parameters" that eventually (after a number of synchronizations) reflect the true state of the learner
- Other challenges (next couple of days)
  - Large numbers of parameters (randomized representations)
  - Large numbers of related tasks (multitask learning)

Obrigado!