PARAMETER SERVER

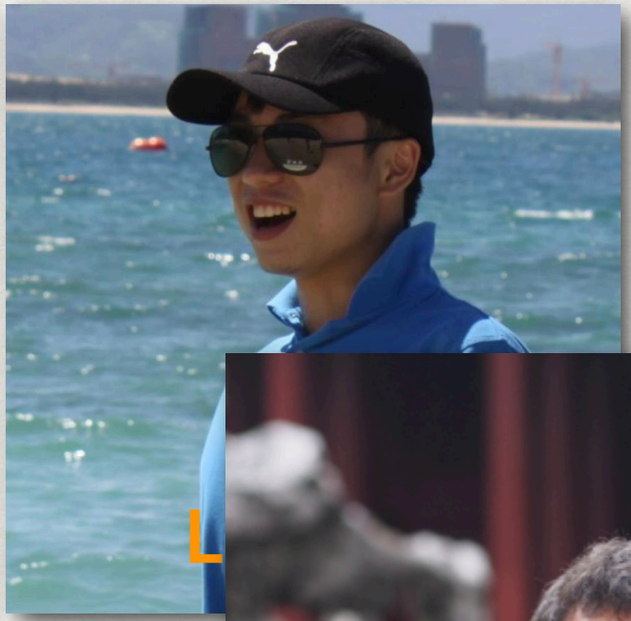
09-22-2018 YIQING MA

OUTLINE

- # First : Basic Knowledge PS (Story, Principle and Detail)
- # Second: Tutorial of PS (how to code Parameter Sever)

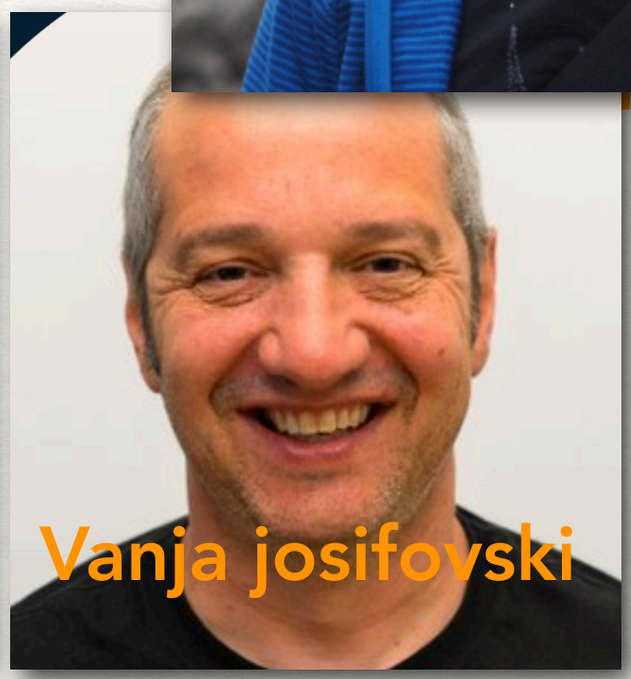
#	Third: State-Of-The-Art of Parameter Server
1	• Scaling Distributed Machine Learning with the Parameter Server
2	• Parameter Hub: High Performance Parameter Servers for Efficient Distributed Deep Neural Network Training
3	• Blink: A fast NVLink-based collective communication library
4	• HiPS: Hierarchical Parameter Synchronization in Large-Scale Distributed Machine Learning

LEVEL 1 PART



FAST … EEP

SCAL… …RVER



Bor-Yiing Su

Machine Intelligence

Eugene J. Shekita

Google

在 google.com 的电子邮件经过验证

Databases Machine Learning

OUTLINE

1. BackGround
   Models , hardware

2. Bipartite design
   Communication, Key layout, Recovery

3. Efficiency
   Consistency models

4. Improving the layout
   submodular load balancing

BACKGROUND

DATA FLOW

Df loss(x)

LOSS GRADIENT

f(x)

DATA

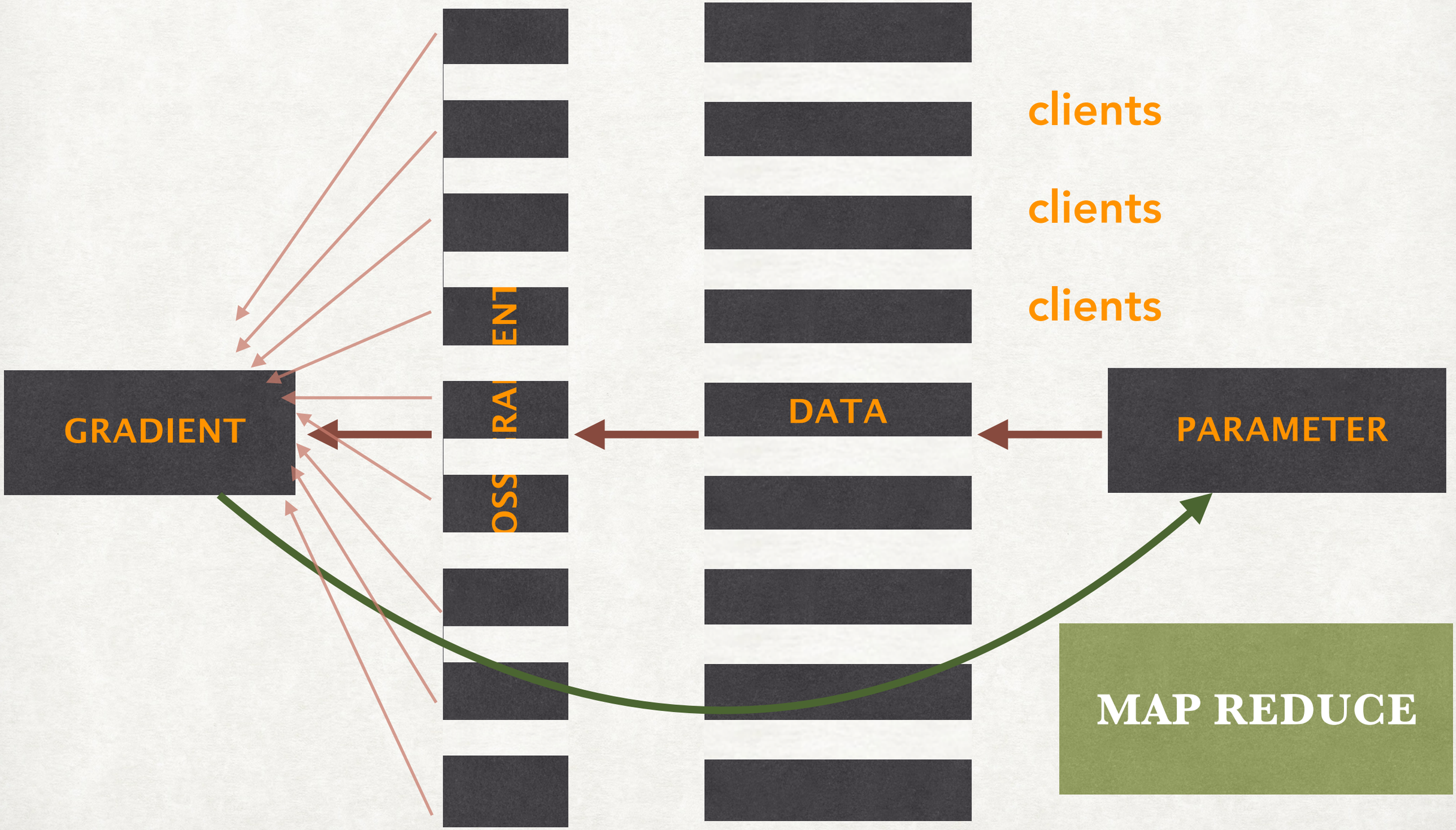
$$x \cdot w$$

$$x^t \cdot g$$

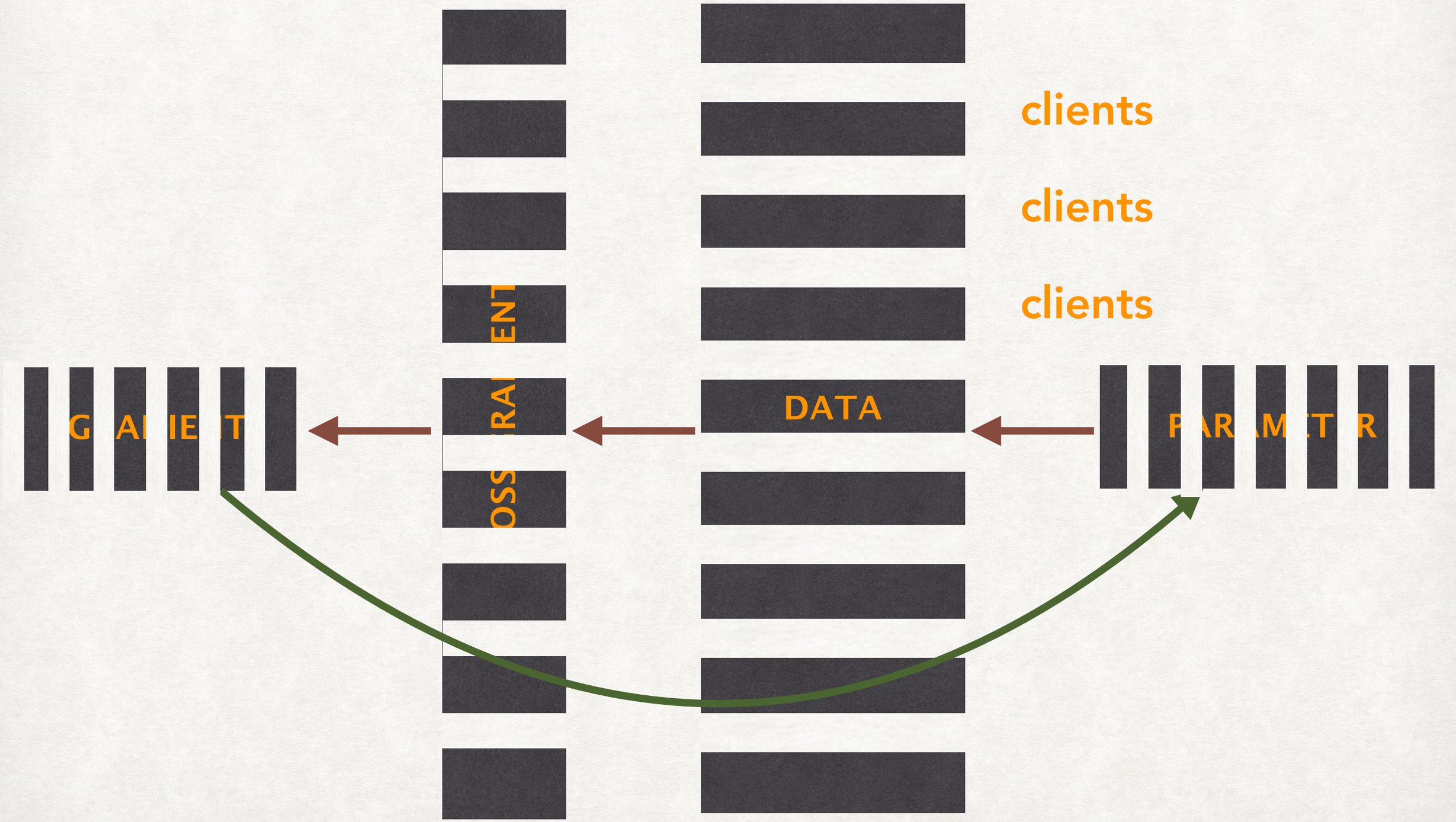
GRADIENT

PARAMETER

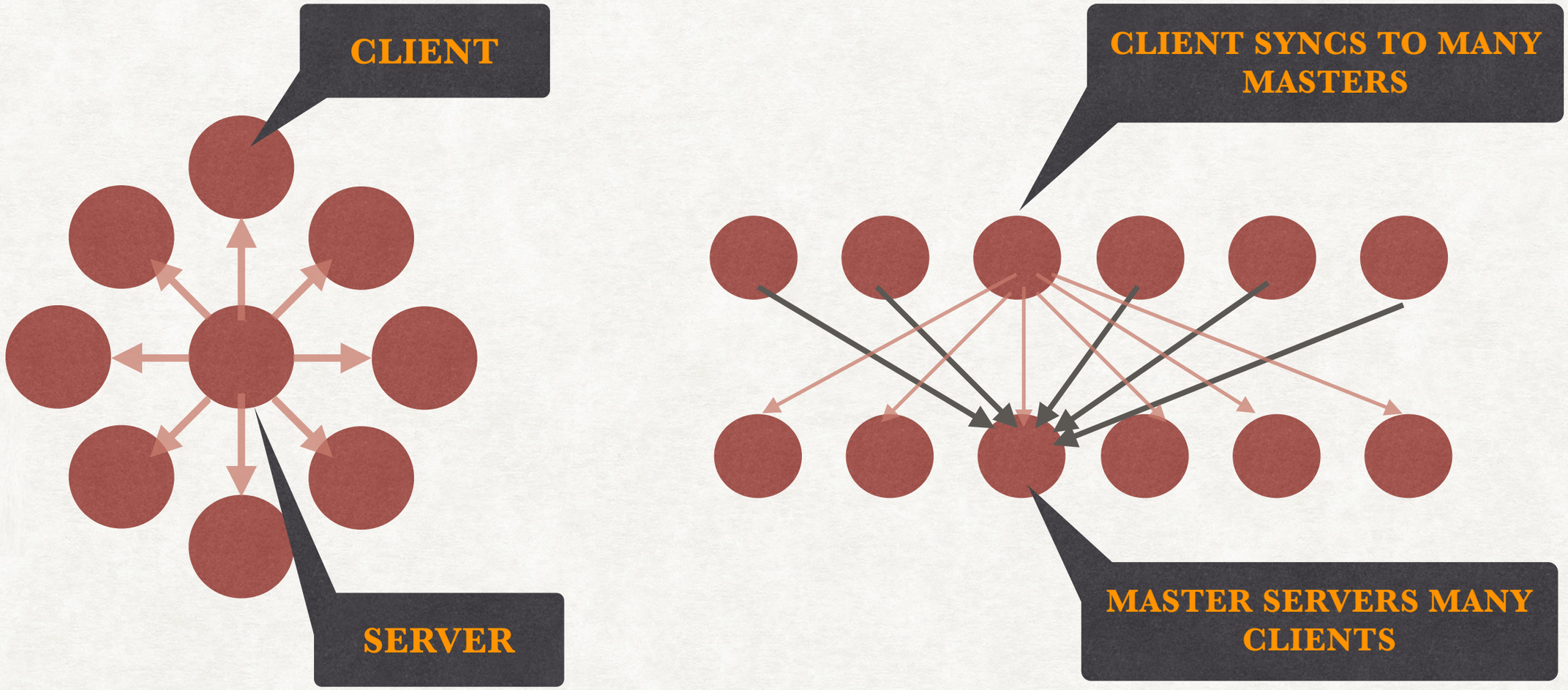
DATA FLOW


clients
clients
clients
GRADIENT
DATA
PARAMETER
MAP REDUCE

DATA FLOW

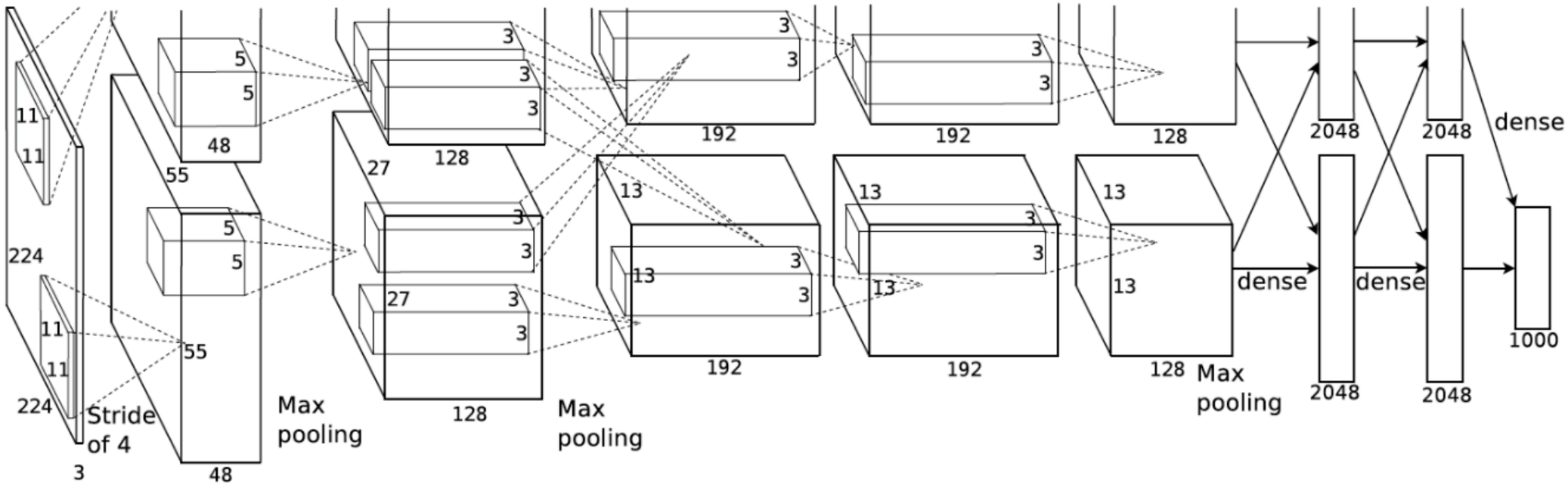


COMMUNICATION PATTERN



Put(keys,values,clock) ,get (keys,valuse,clock)

DEEP NETWORK

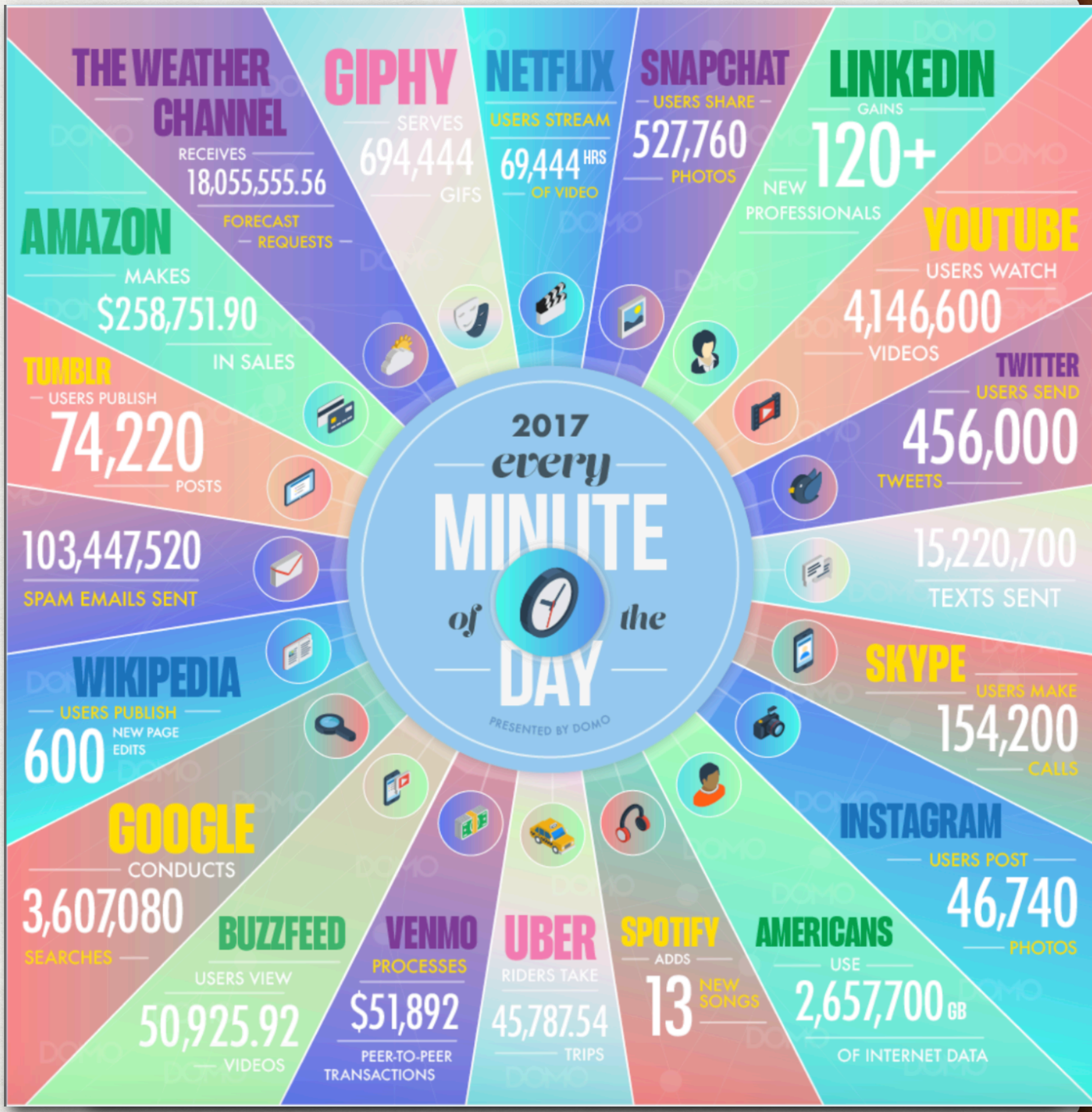


Architecture for the paper

- Gradients are more structured (groups per layer)
- Hierarchical structure (multi GPU to host to server)

DATA PER MIN 2018

We scale:
100TB data
1000 machine
100B parameters
1B inserts/s
4B documents
2M topics/s



STUFF FAILS A LOT . DEAL WITH IT

Typical first year for a new cluster:

- 0.5 overheating (power down most machines in < 5mins,~1-2 days to recover)
- 1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
- 1 rack-move (plenty of warning,~500-1000 machines powered down, ~6 hours)
- 1 network rewiring (rolling ~5% of machines down over 2-day span)
- 20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- 5 racks go wonky (40-80 machines see 50% packet loss)
- 8 network maintenances(4 might cause ~30-minute random connectivity losses)
- 12 router reloads (takes out DNS and external vips for a couple minutes)
- 3 router failures (have to immediately pull traffic for an hour)
- dozens of minor 30-second blips for dns
- 1000 individual machine failures
- Thousands of hard drive failures

(Slide courtesy Jeff Dean)

Slow disks, bad memory, misconfigured machines, flaky machines, etc.

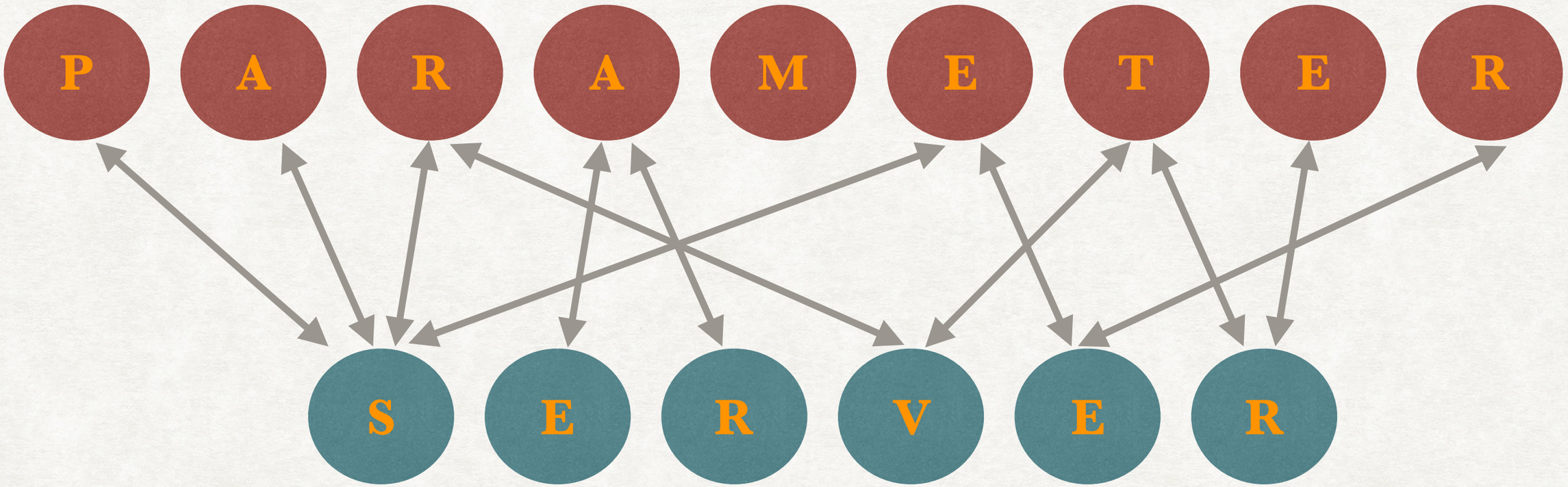
OUTLINE

1. BackGround
   Models , hardware

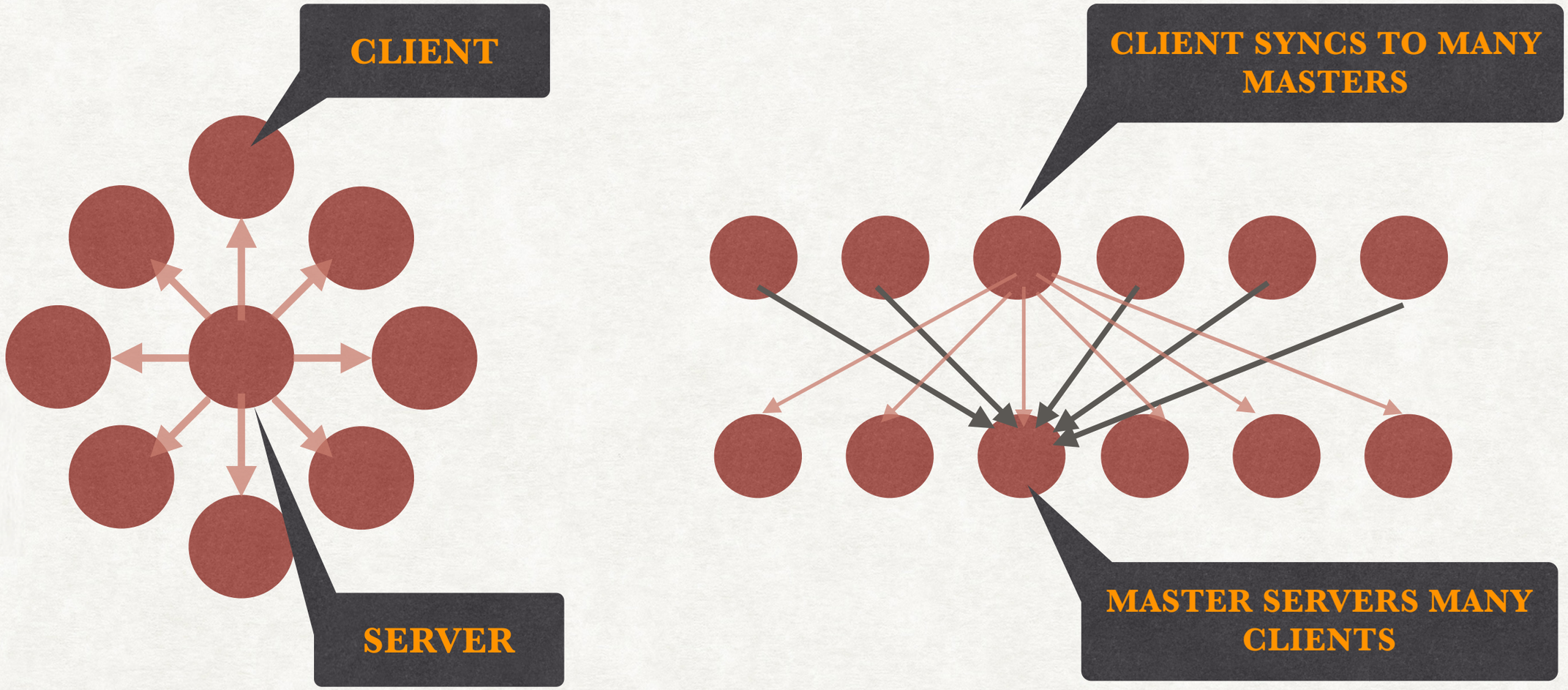
2. Bipartite design
   Communication, Key layout, Recovery

3. Efficiency
   Consistency models

4. Improving the layout
   submodular load balancing


P A R A M E T E R
S E R V E R

COMMUNICATION PATTERN



Put(keys,values,clock) ,get (keys,valuse,clock)

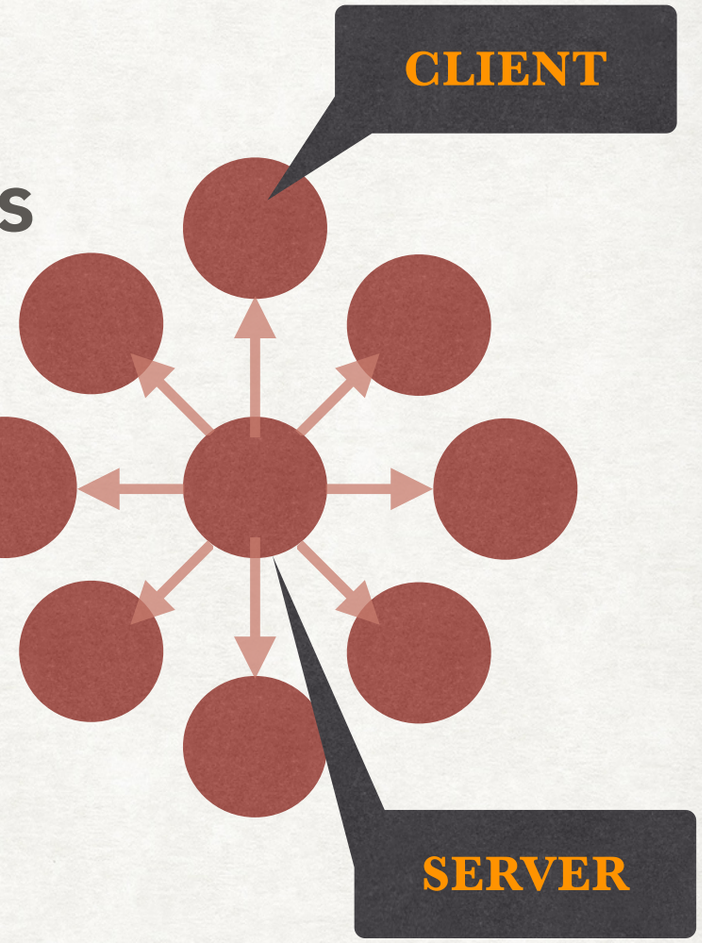
GENERAL PARALLEL ALGORITHM TEMPLATE

Clients have local view of parameters.

P2P is infeasible since $O(n^2)$ connections

Synchronize with parameter server

- Reconciliation protocol
  average parameters, lock variables
- Synchronization schedule
  asynchronous, synchronous, episodic
- Load distribution algorithm
  uniform distribution, fault tolerance, recovery.

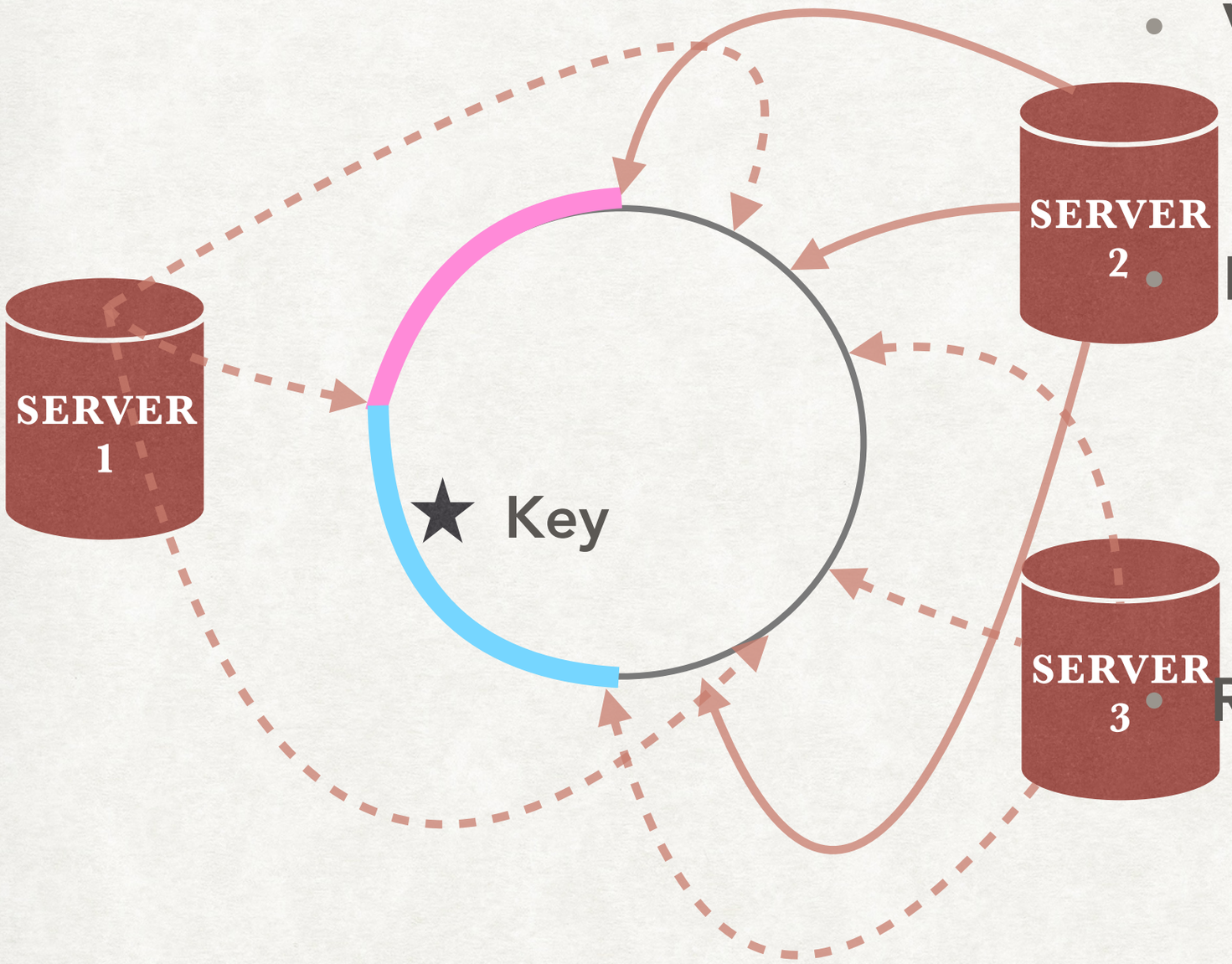


Smola & Narayanamurthy, 2010, VLDB
Gonzalez et al., 2012, WSDM
Shervashidze et al., 2013 , WWW

Also at Google, Baidu,
Facebook, Amazon,
Yahoo, Microsoft…

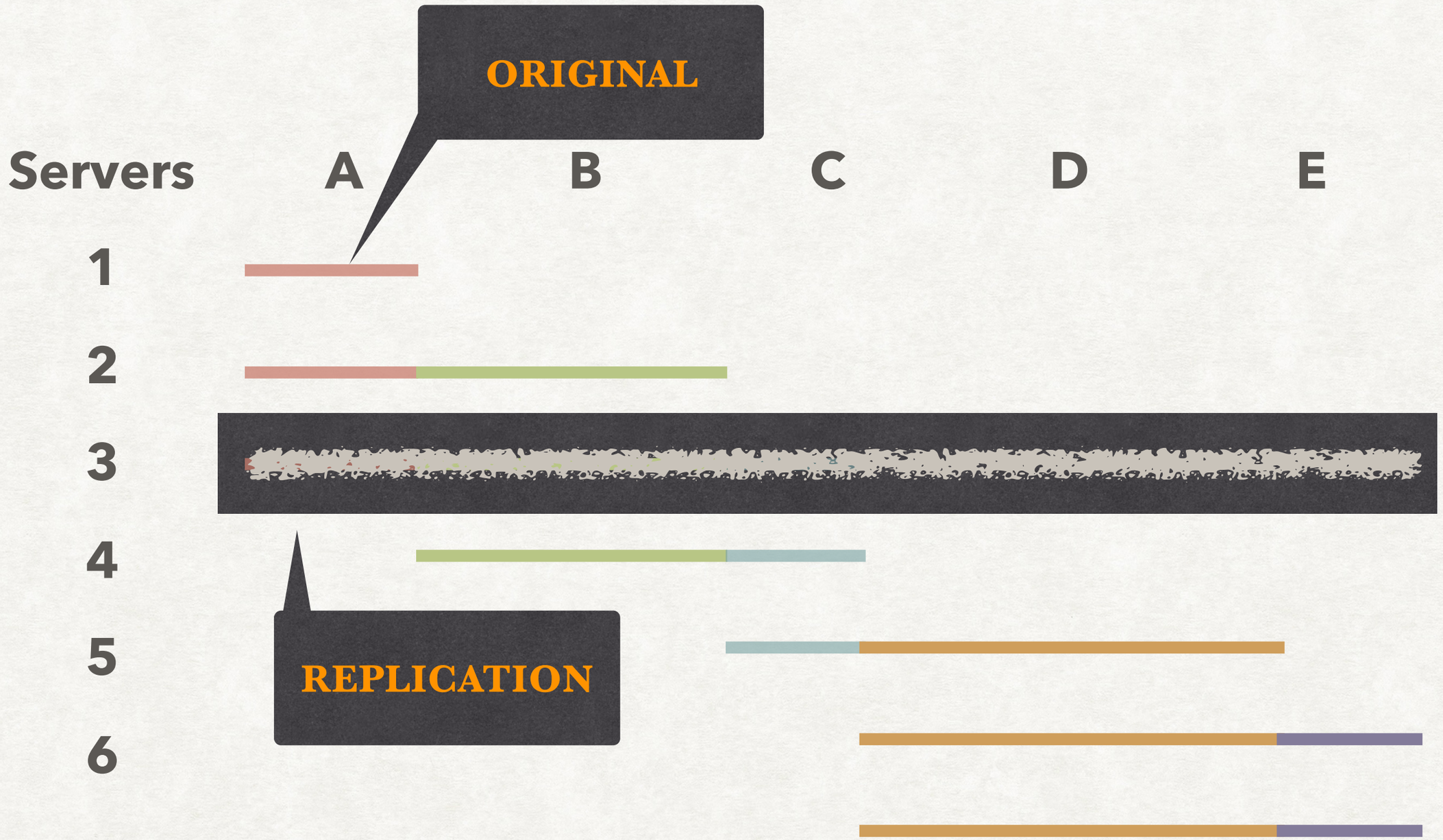
KEY LAYOUT AND RECOVERY

KEYS ARRANGED IN A DHT

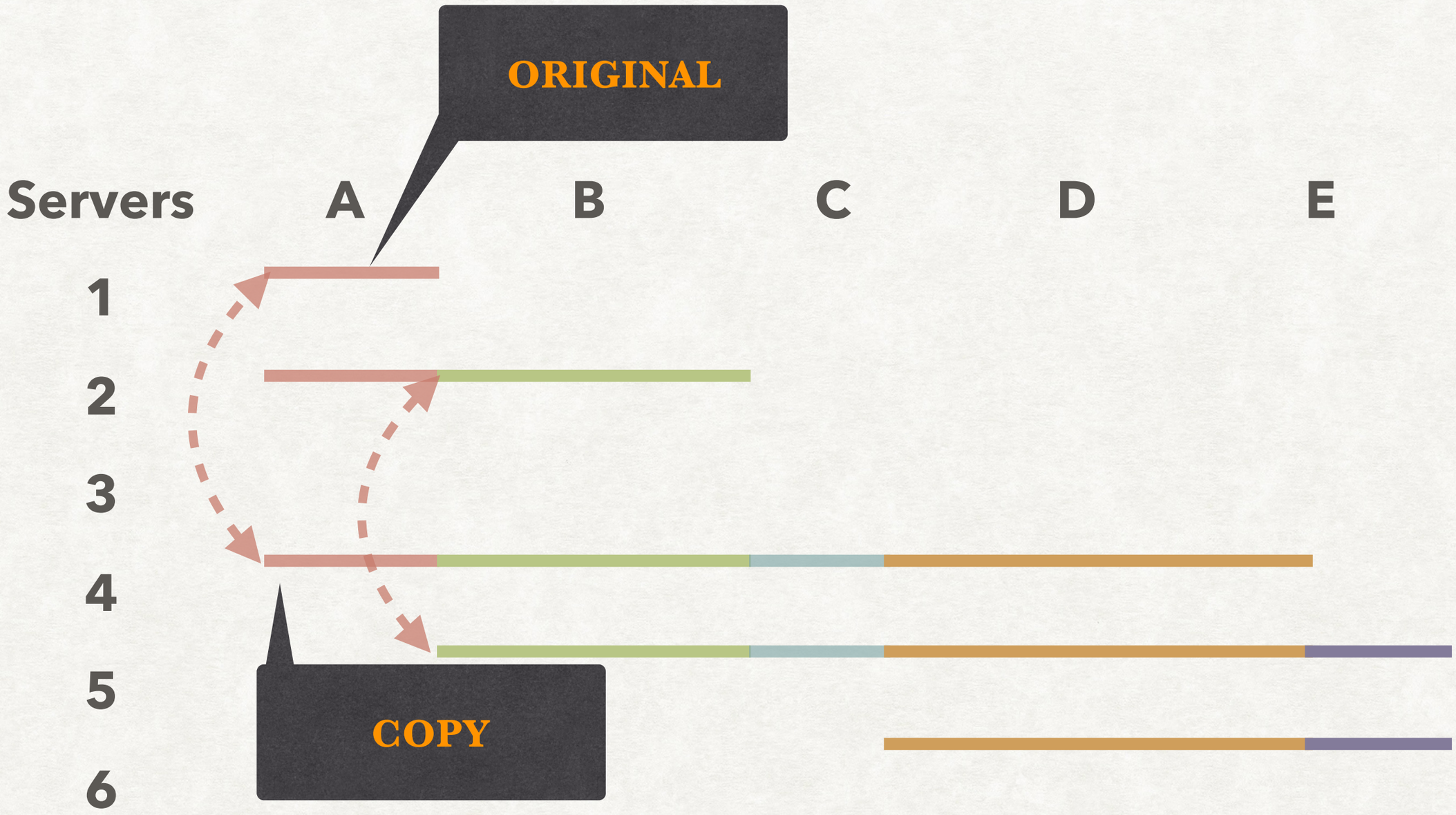


- Virtual Servers
  - Load balancing
  - multithreading
- DHT
  - Contiguous key range for clients
  - Easy bulk sync
  - Easy insertion of servers
- Replication
  - Machines hold replicas
  - Easy fallback
  - Easy insertion /repair

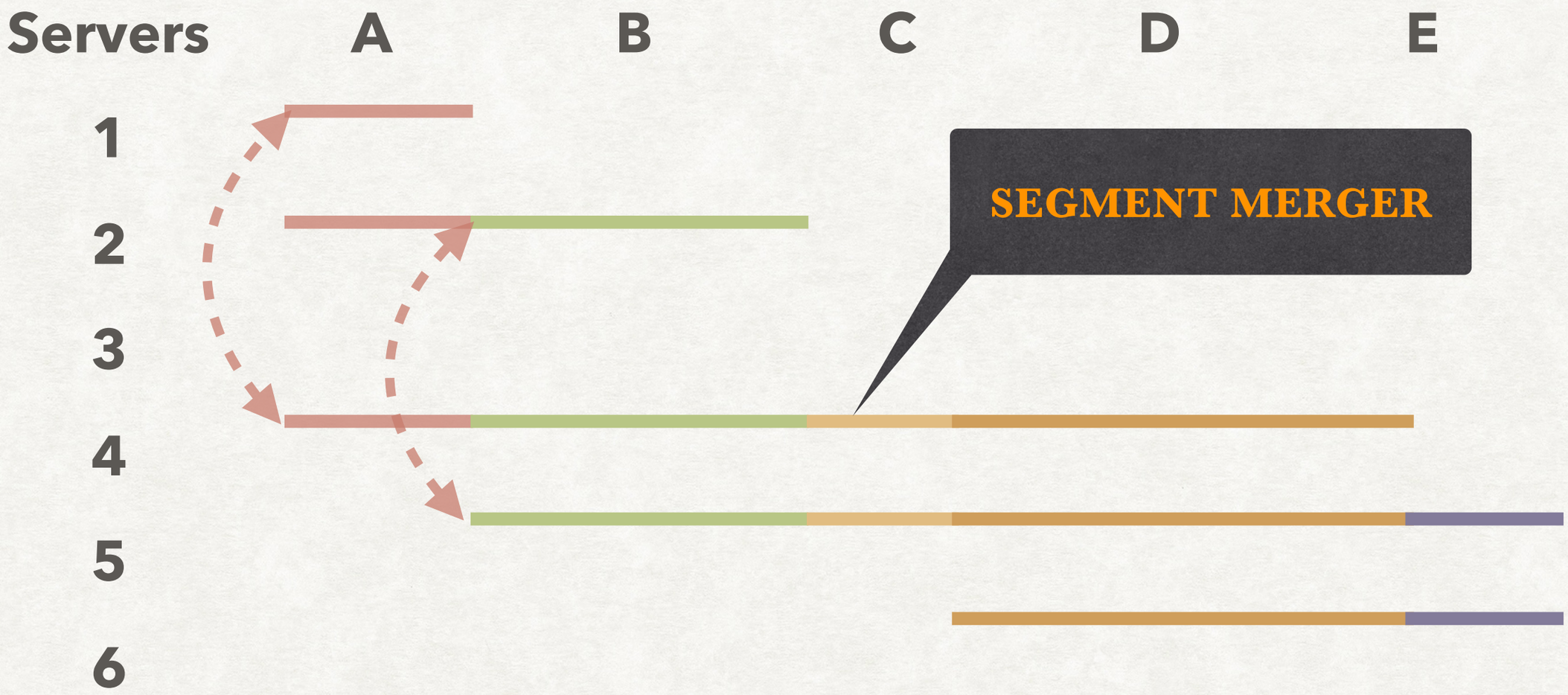
KEY LAYOUT



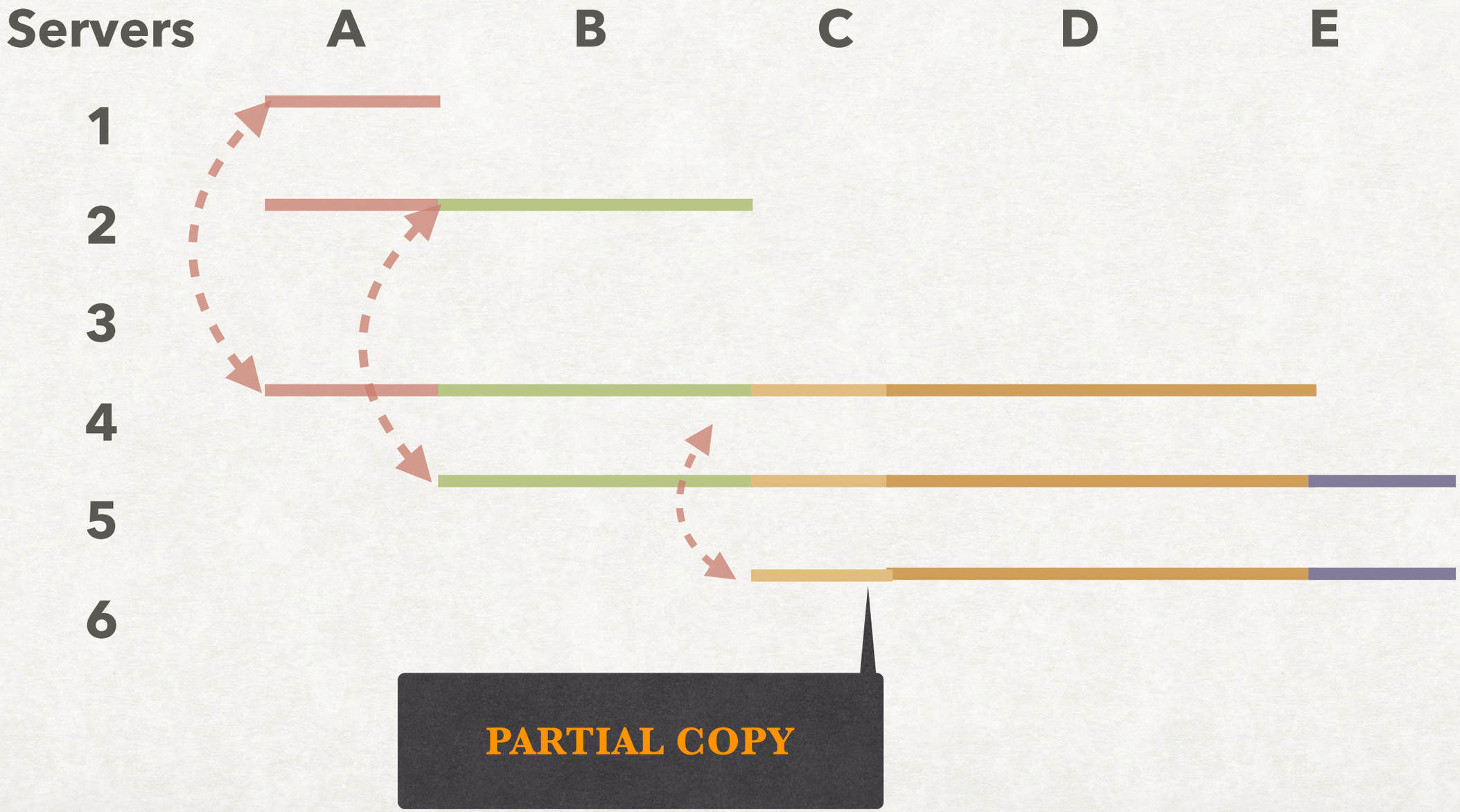
KEY LAYOUT



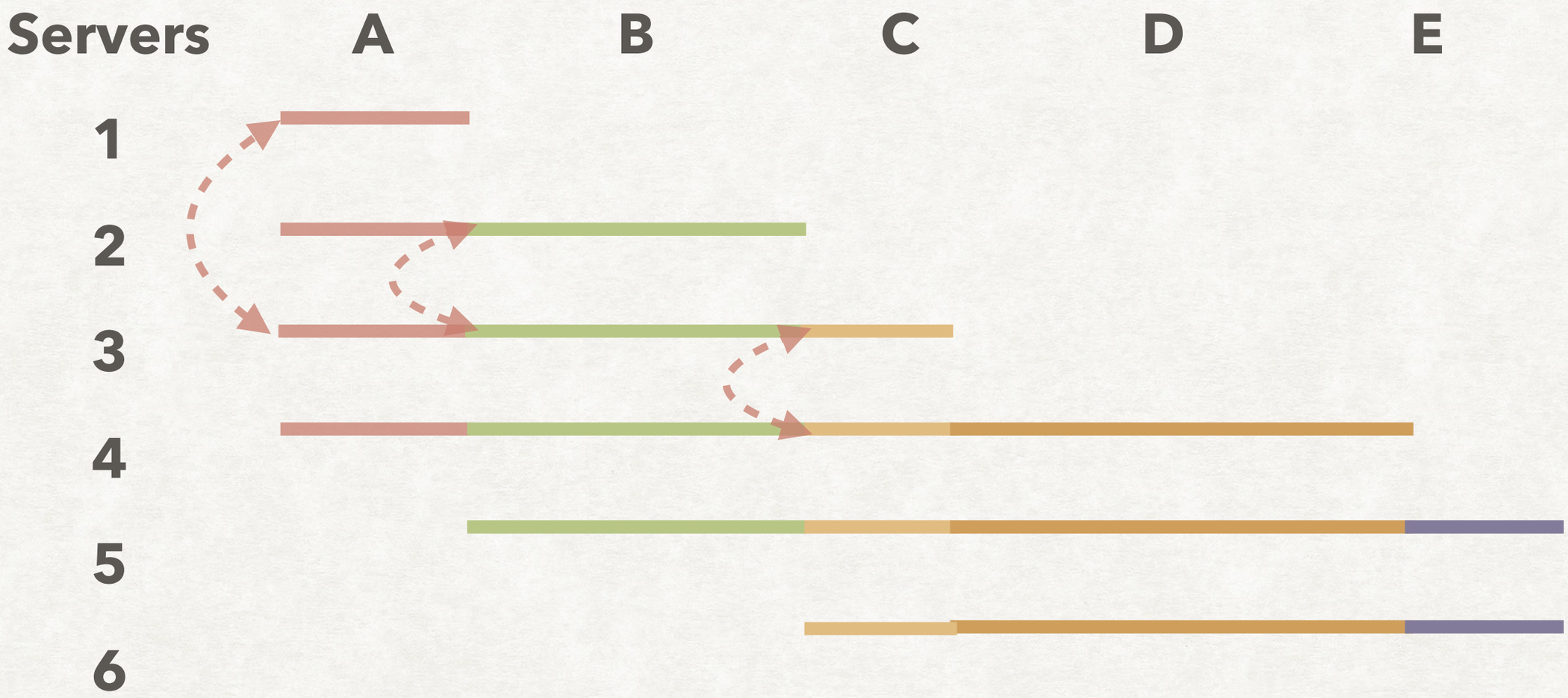
KEY LAYOUT



KEY LAYOUT



KEY LAYOUT ★



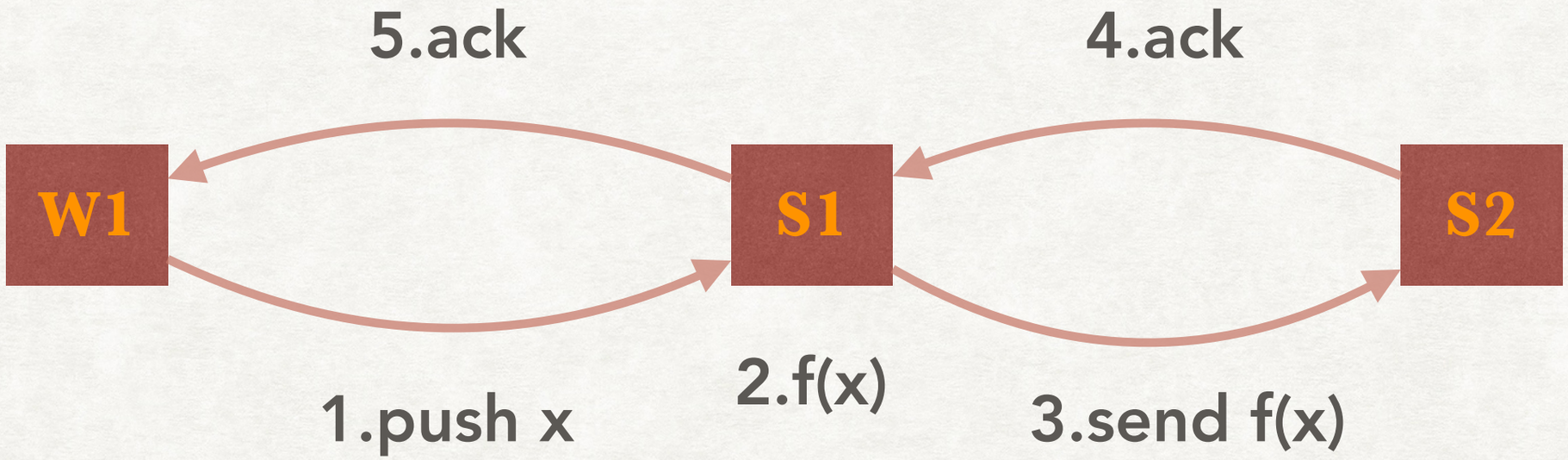
- Precopy server content to new candidate(3)
- After precopy ended, send log
- For k virtual servers this cause O(k^{-2}) delay
- Consistency using vector clocks

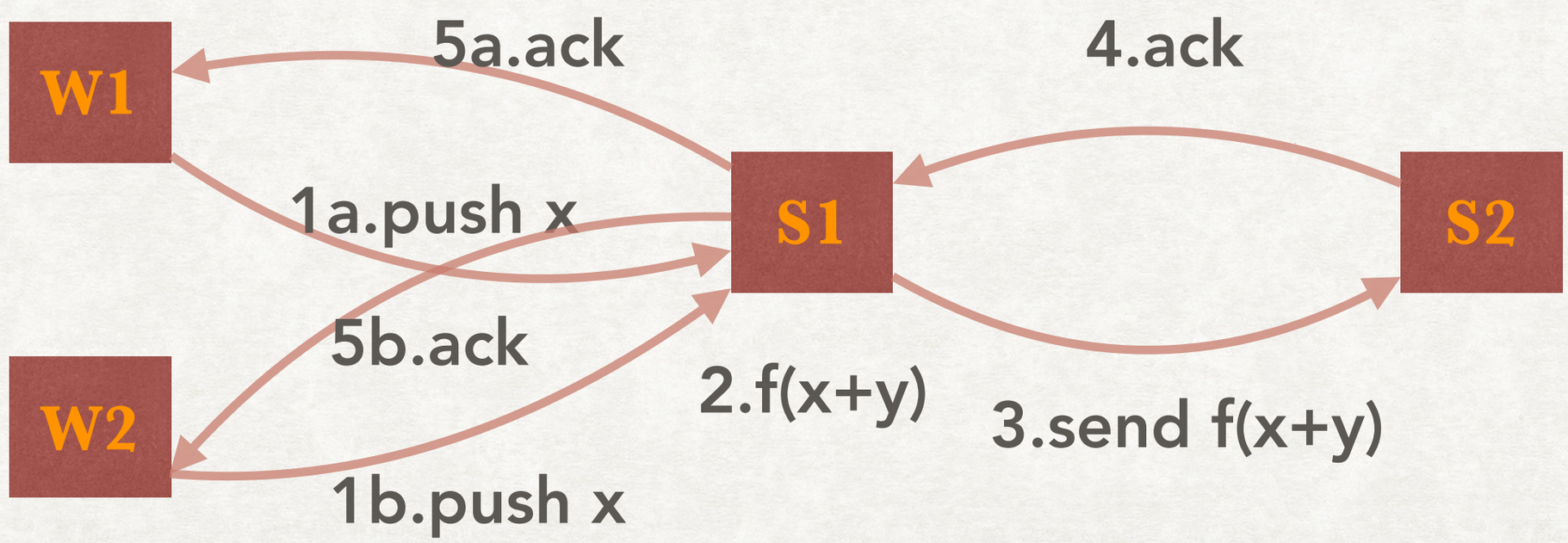
COMMUNICATION

SIMPLE API

- Clients and Servers share much code
- Send data to server
  - Asynchronously in an interval
    push(key_list, value_list, flag)
  - Receive data from server in an interval
    pull(key_list, value_list, flag)
- Avoid sending single items
  - Serialization overhead - protobuf message
  - Consistency overhead - O(c) vector clocks

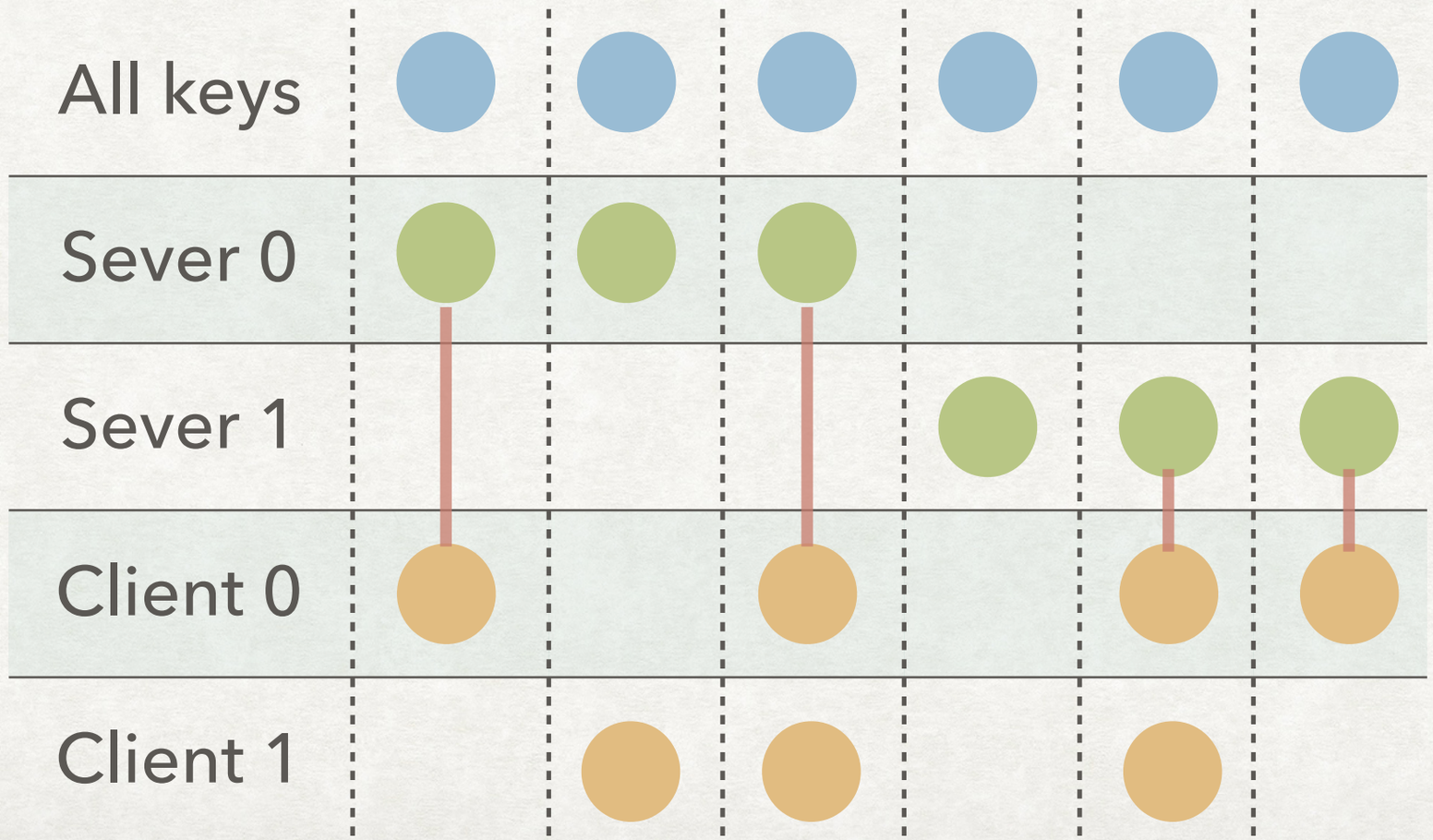
MESSAGE AGGREGATION ON SERVER





SEND AS LITTLE AS POSSIBLE

- Only send data the receiver needs
  - A server node maintain segments of keys
  - Client nodes may have different working sets



OUTLINE

1. BackGround
   Models , hardware

2. Bipartite design
   Communication, Key layout, Recovery

3. Efficiency
   Consistency models

4. Improving the layout
   submodular load balancing

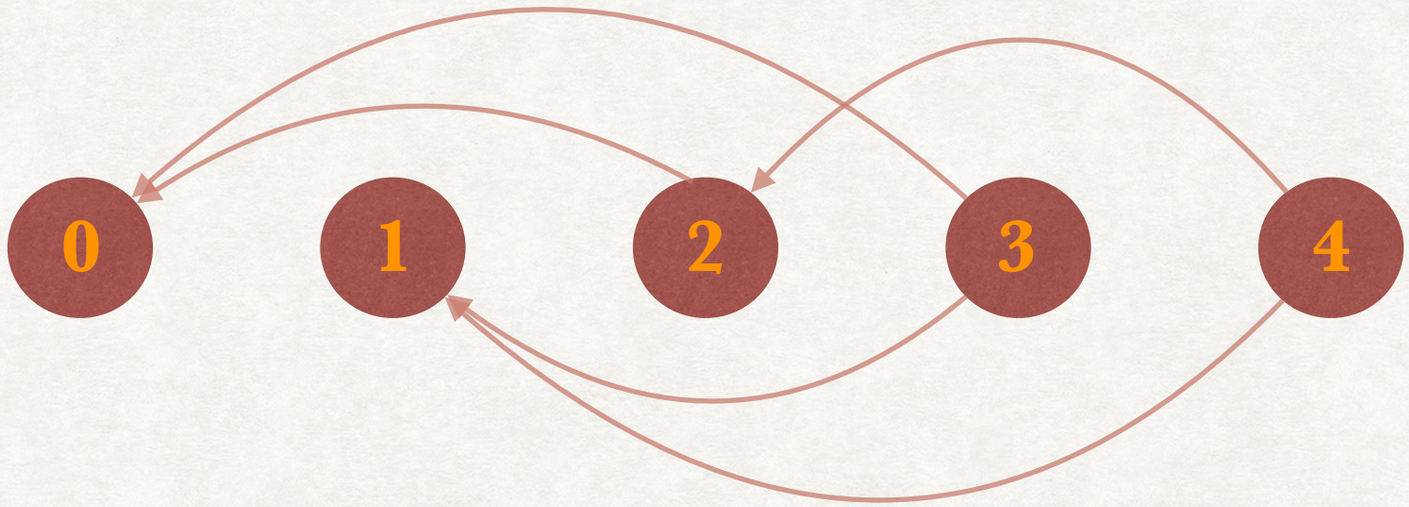
CLOCKS AND CONSISTENCY

CONSISTENCY MODEL

Sequential

Eventual

Bounded delay



Via task processing engine on client/controller

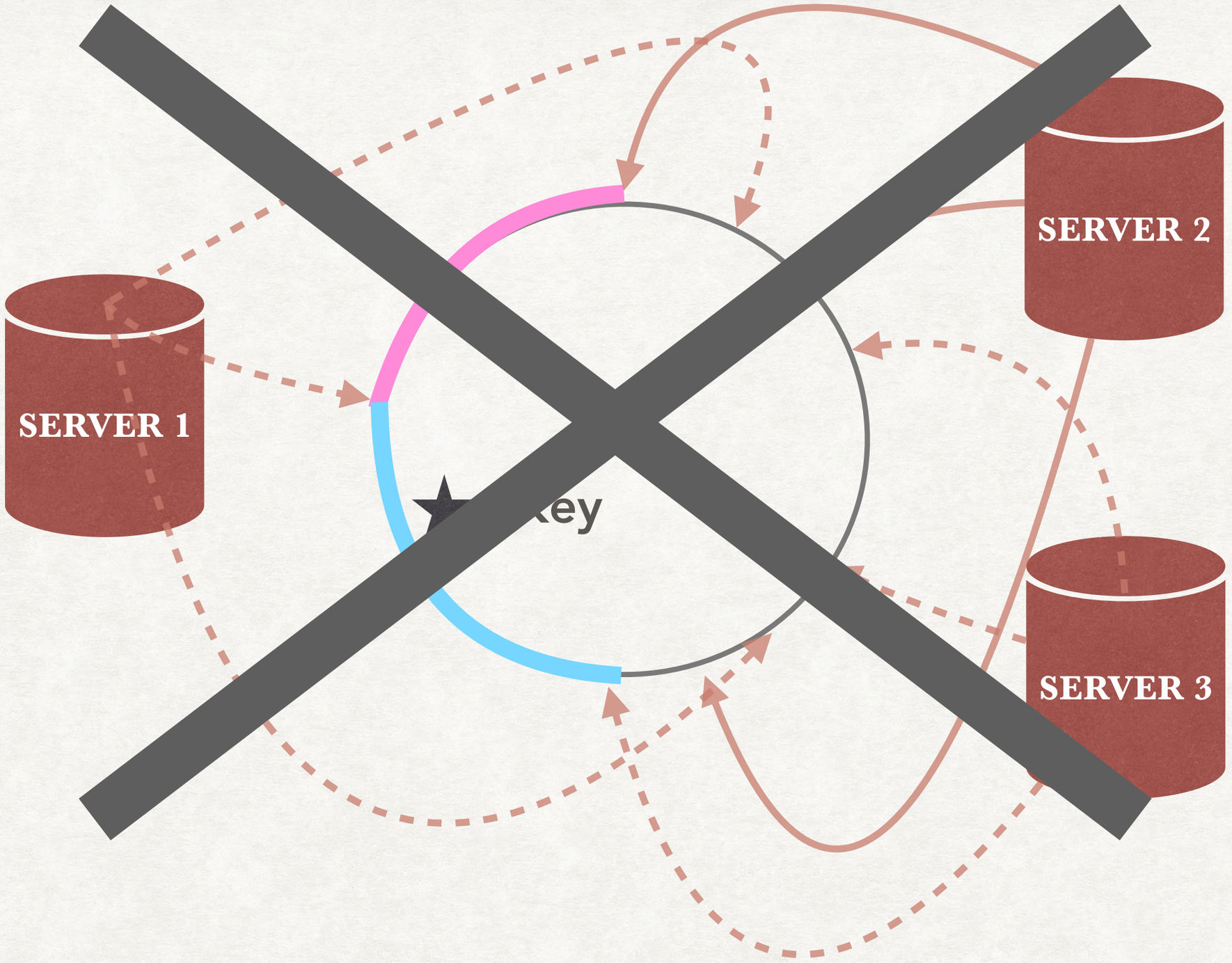
OUTLINE

1. BackGround
   Models , hardware

2. Bipartite design
   Communication, Key layout, Recovery

3. Efficiency
   Consistency models

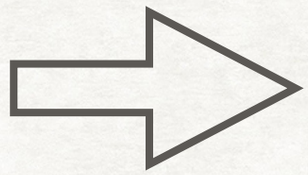
4. Improving the layout
   submodular load balancing

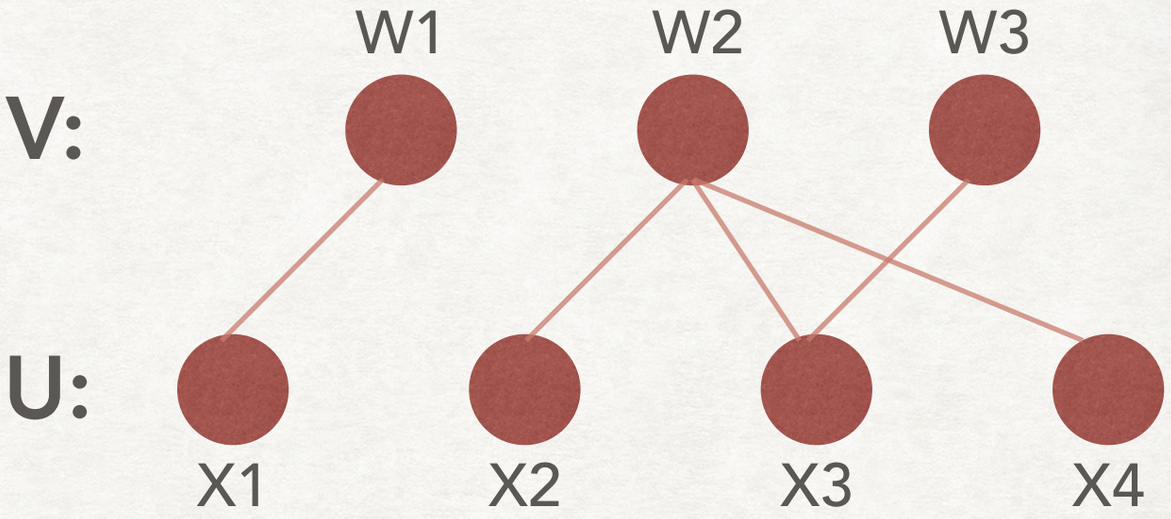


Improved key layout

LOCAL KEY DISTRIBUTION

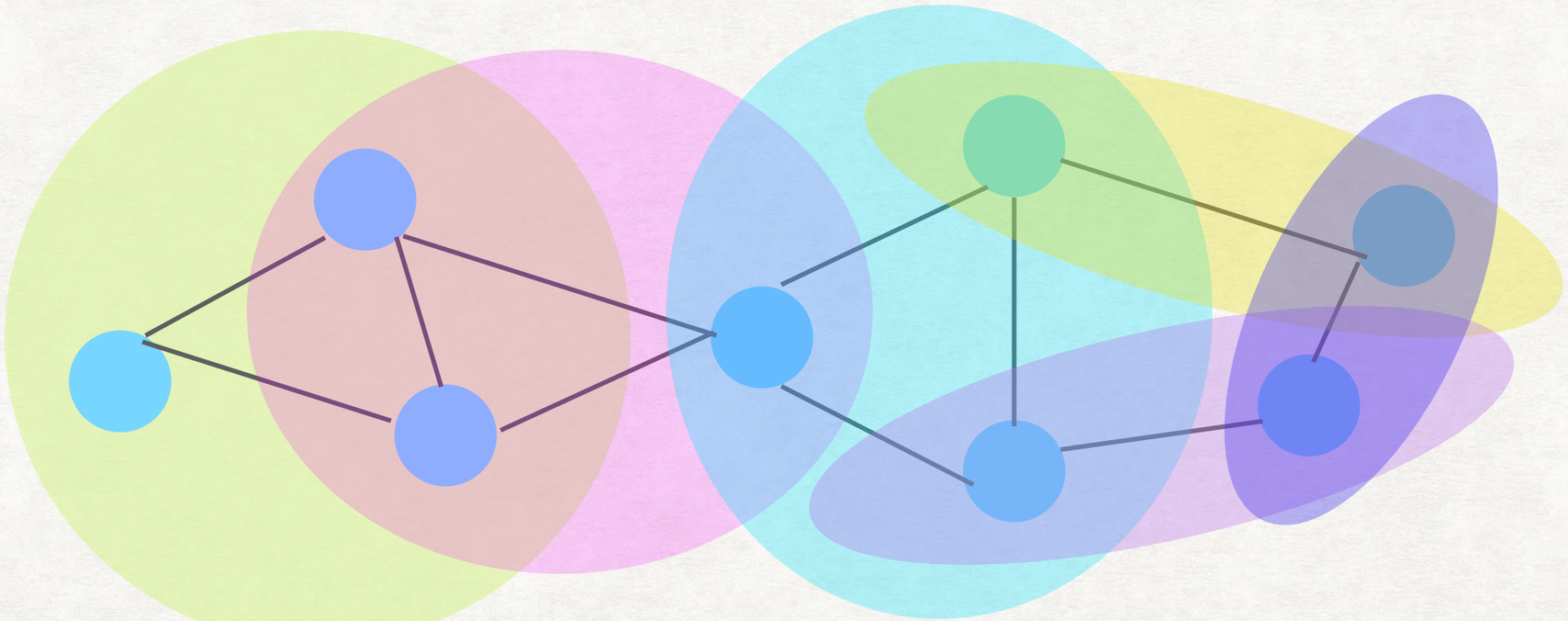
X1 = (.1, __, __)
X2 = (__, .3, __)
X3 = (__, .4, .3)
X4 = (__, .9, __)





- Randomly partitioning data leads to lots of network traffic between clients & servers
- Clients:documents ,user activity
  (needs to cache all relevant parameters)
- Servers :parameters

LOCAL KEY DISTRIBUTION



- Randomly partitioning data leads to lots of network traffic between clients & servers
- Clients:documents ,user activity
  (needs to cache all relevant parameters)
- Servers :parameters

GOALS

- Memory

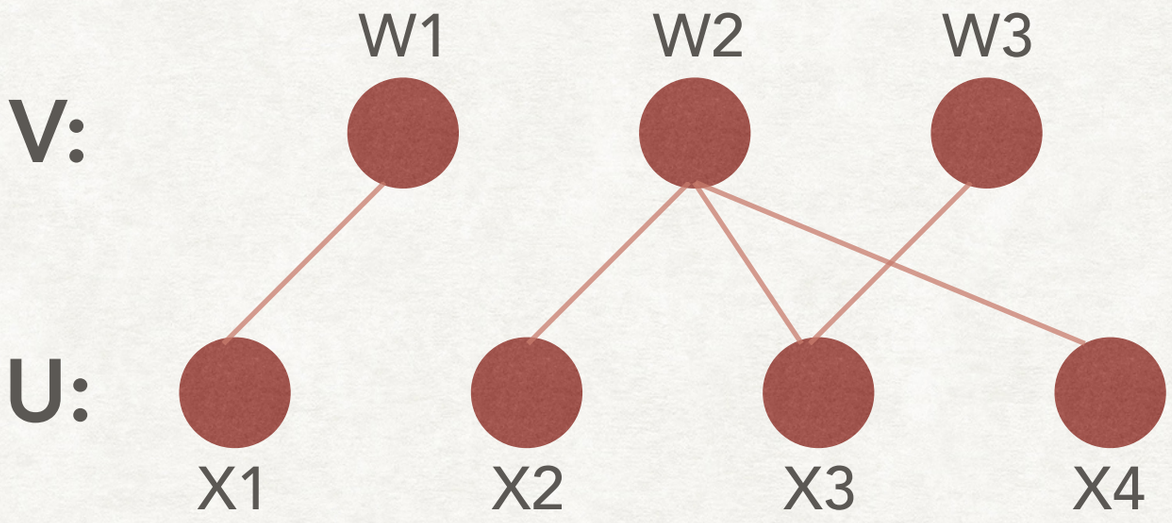
  Must not exceed client memory allowance (cache all relevant variables)
- Work

  Should balance workload over clients
- Network

  Should minimize communication cost
- Without loss of generality assume bipartite graph to be partitioned

MEMORY

- Graph G(U, V, E)
- Select vertices in U with few neighbors
- Minimizing memory



$$\min imize\_\_\max_{i}\left|N(U_i)\right|\_\_ whereN(U_i) := \bigcup_{u \in U_i} N(u)$$

WORST CLIENT

MEMORY LOAD

NEIGHBORS IN V

MEMORY

- # Neighbors of U_i is a submodular function (if v already a neighbor, adding u is free)
- Submodular load balancing problem (Svitkina and Fleischer, 2011)

Submodular Approximation: Sampling-Based Algorithms and Lower Bounds*

Zoya Svitkina Lisa Fleischer

$$\text{min}imize \_\_ \max_i \left| N(U_i) \right| \_\_ where N(U_i) := \bigcup_{u \in U_i} N(u)$$

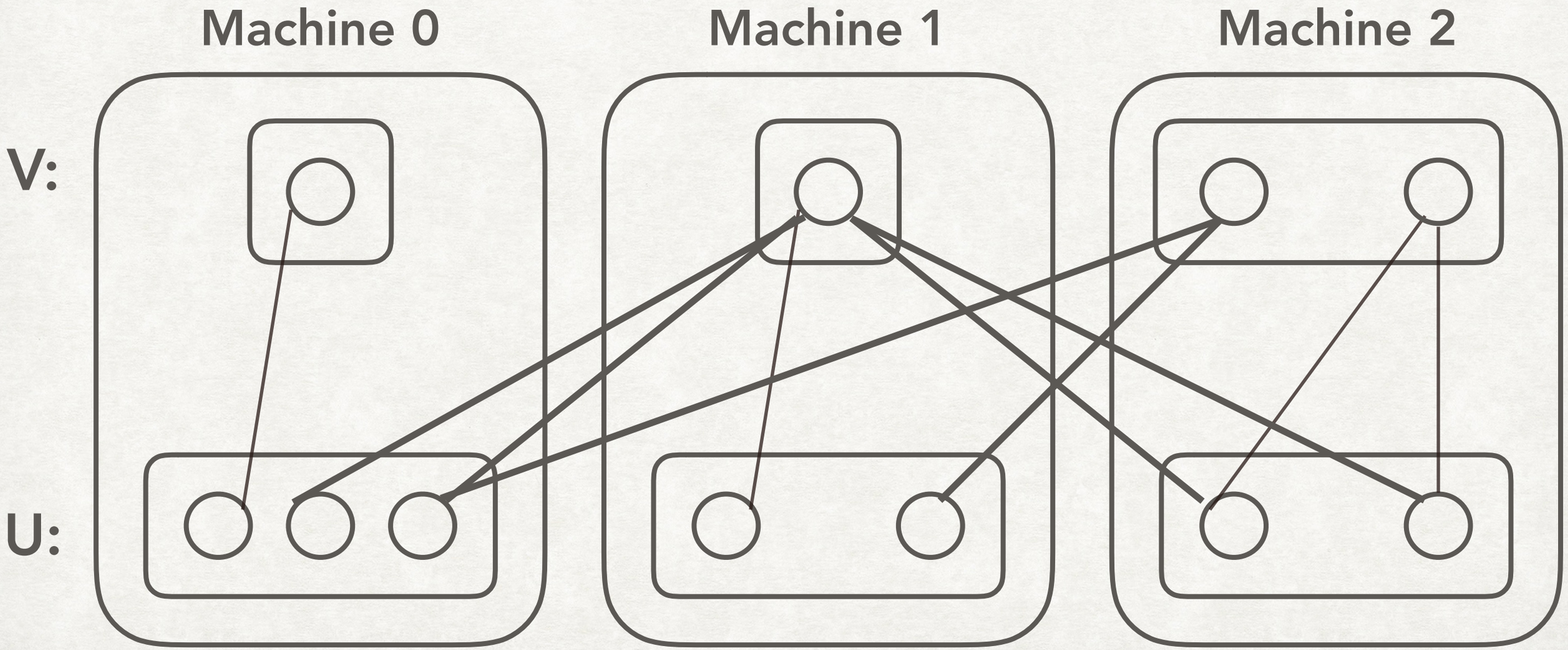
WORST CLIENT

MEMORY LOAD

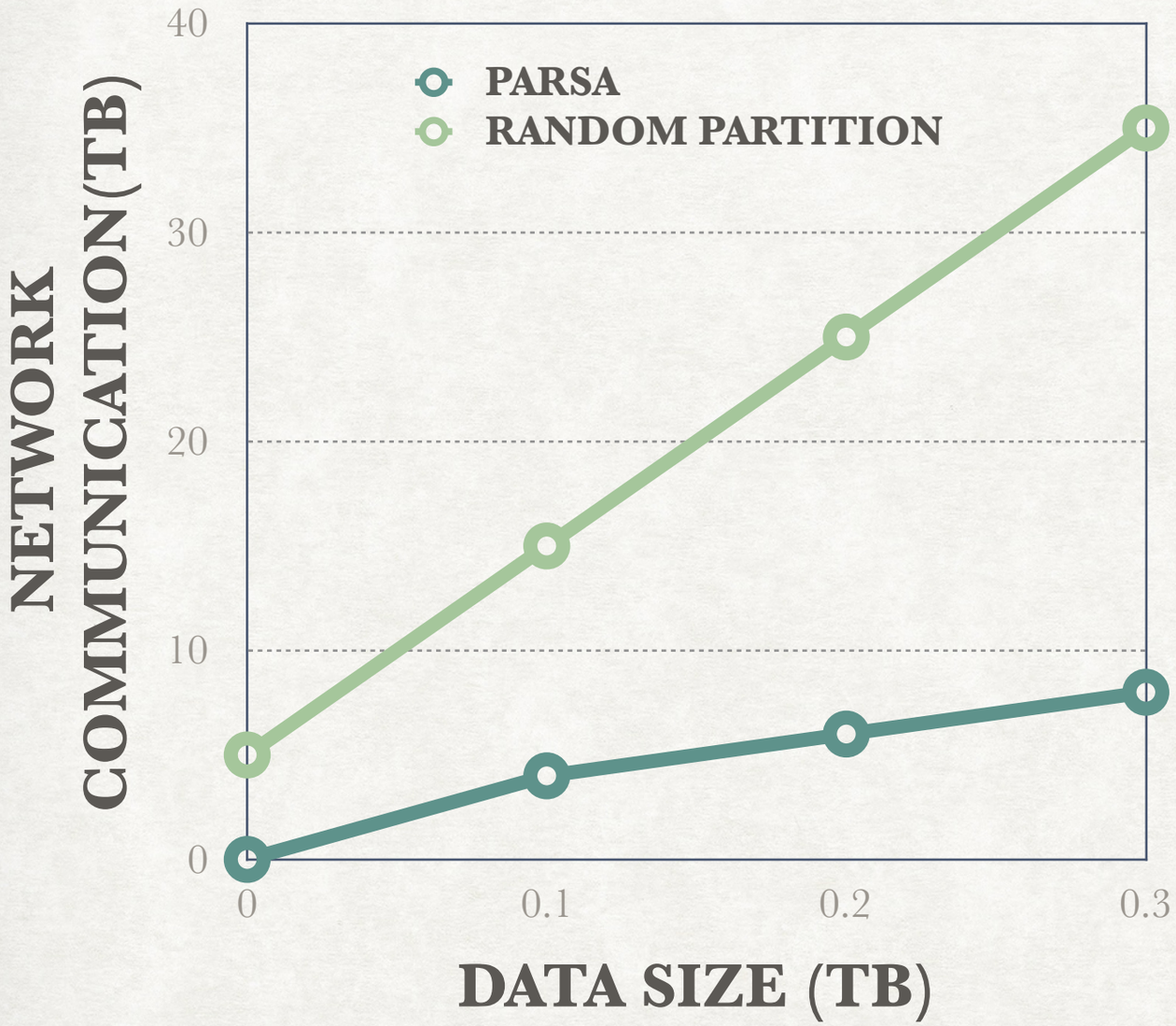
NEIGHBORS IN V

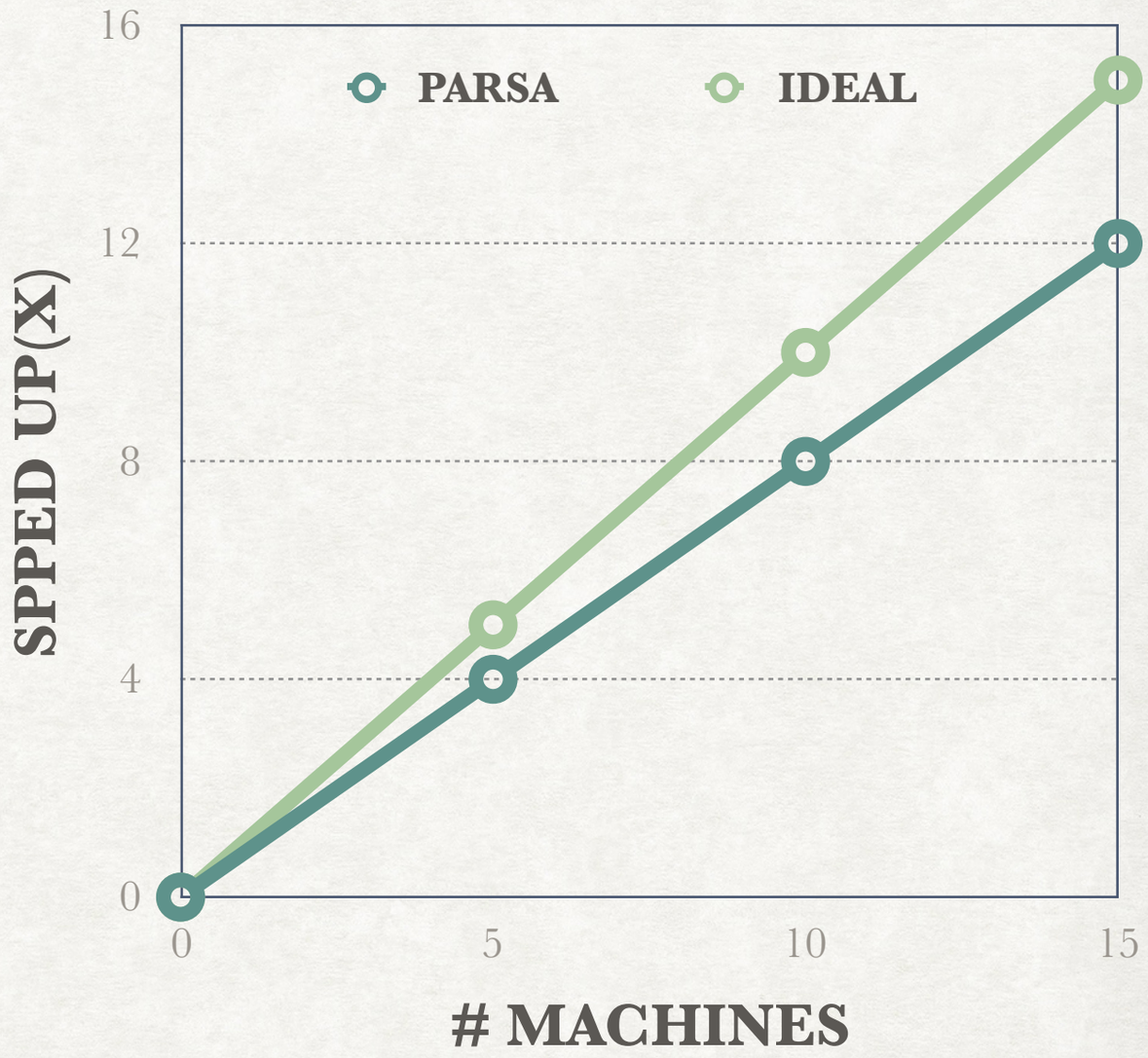
NETWORK

- Put a server on each client.
- Communication cost per machine



BANDWIDTH SAVING





CONCLUDE

- 0.Communication —— PS Parellel Pattern
- 1.Clock —— Version Number
- 2.Key layout and RecoveryDHT
- 3.Consistency model —— Bounded Delay
- 4.Partition Data —— Submodular Load Balancing & Local key distribution

LEVEL 2 PART

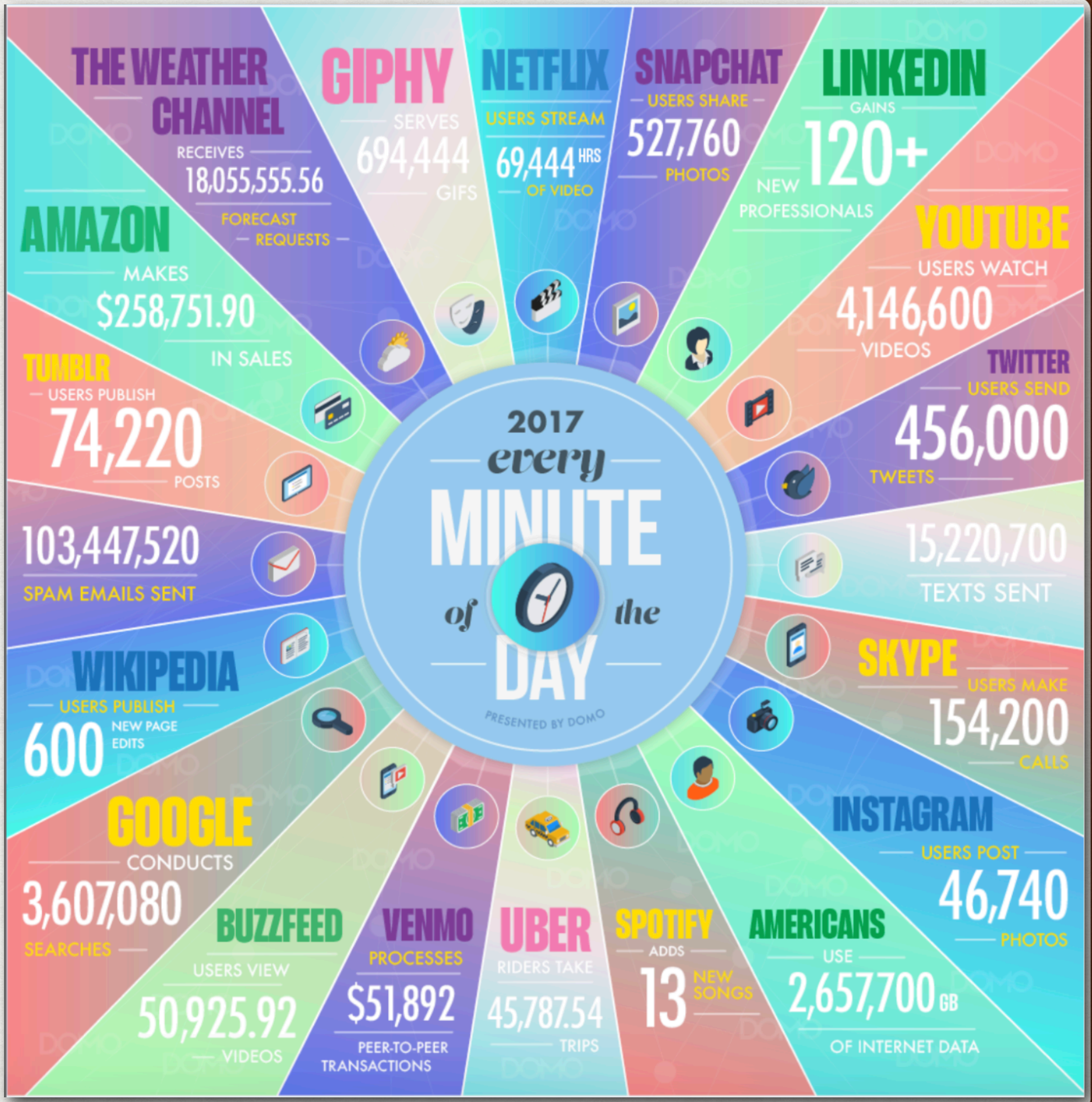
TUTORIAL

- Focus on the design and implementation of large scale machine learning systems
- Parallel and distributed algorithms.
- Several coding examples
- Provide real datasets and machines

APPLICATION & DATA

DATA PER MIN 2018

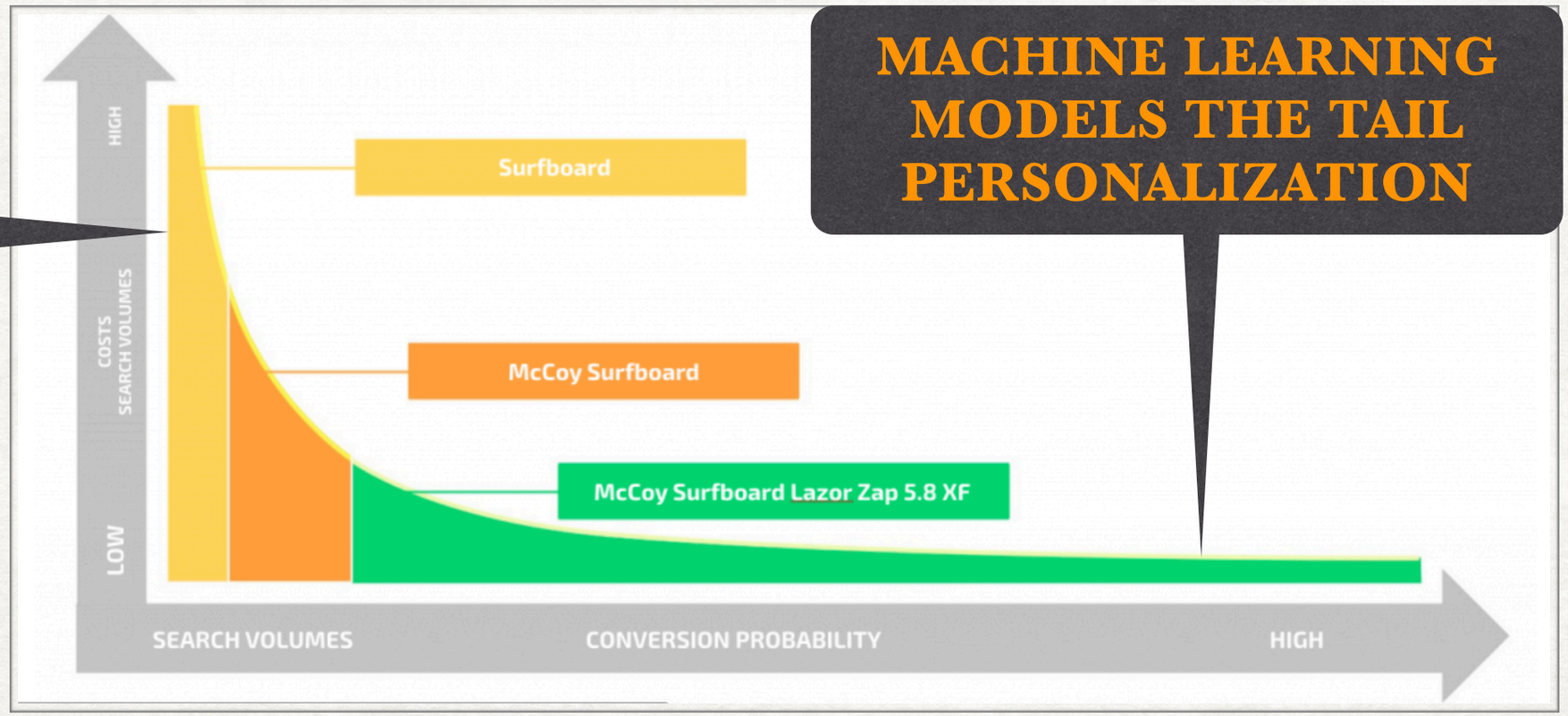
We scale:
100TB data
1000 machine
100B parameters
1B inserts/s
4B documents
2M topics/s



DATA ARE SPARSE

- Most categories have only few data
- A little of category has most of the data

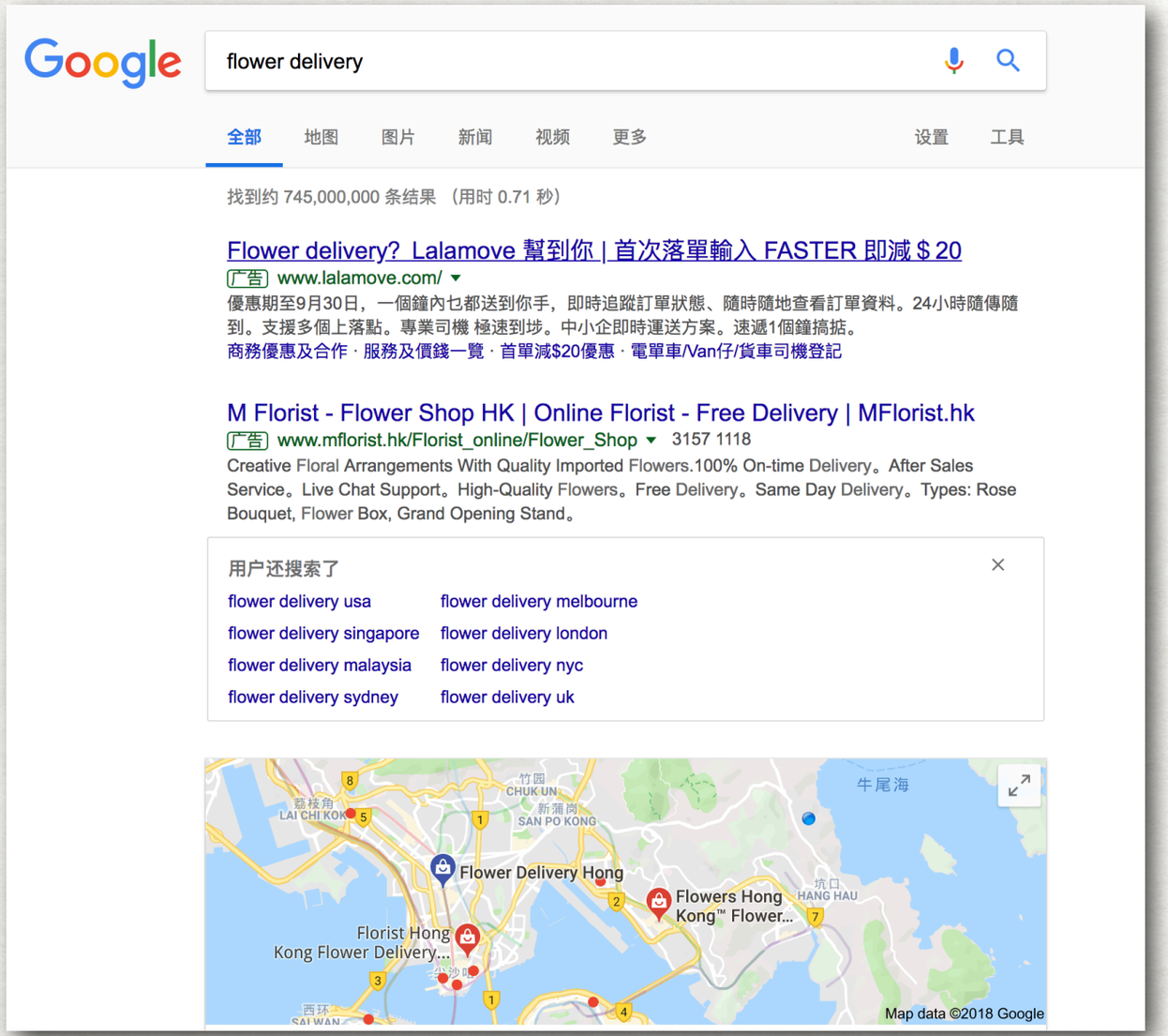
SIMPLE STATISTIC TOOLS MODEL THE HEAD WELL



MACHINE LEARNING MODELS THE TAIL PERSONALIZATION

ONLINE ADVERTISING

- The major revenue source of internet search companies

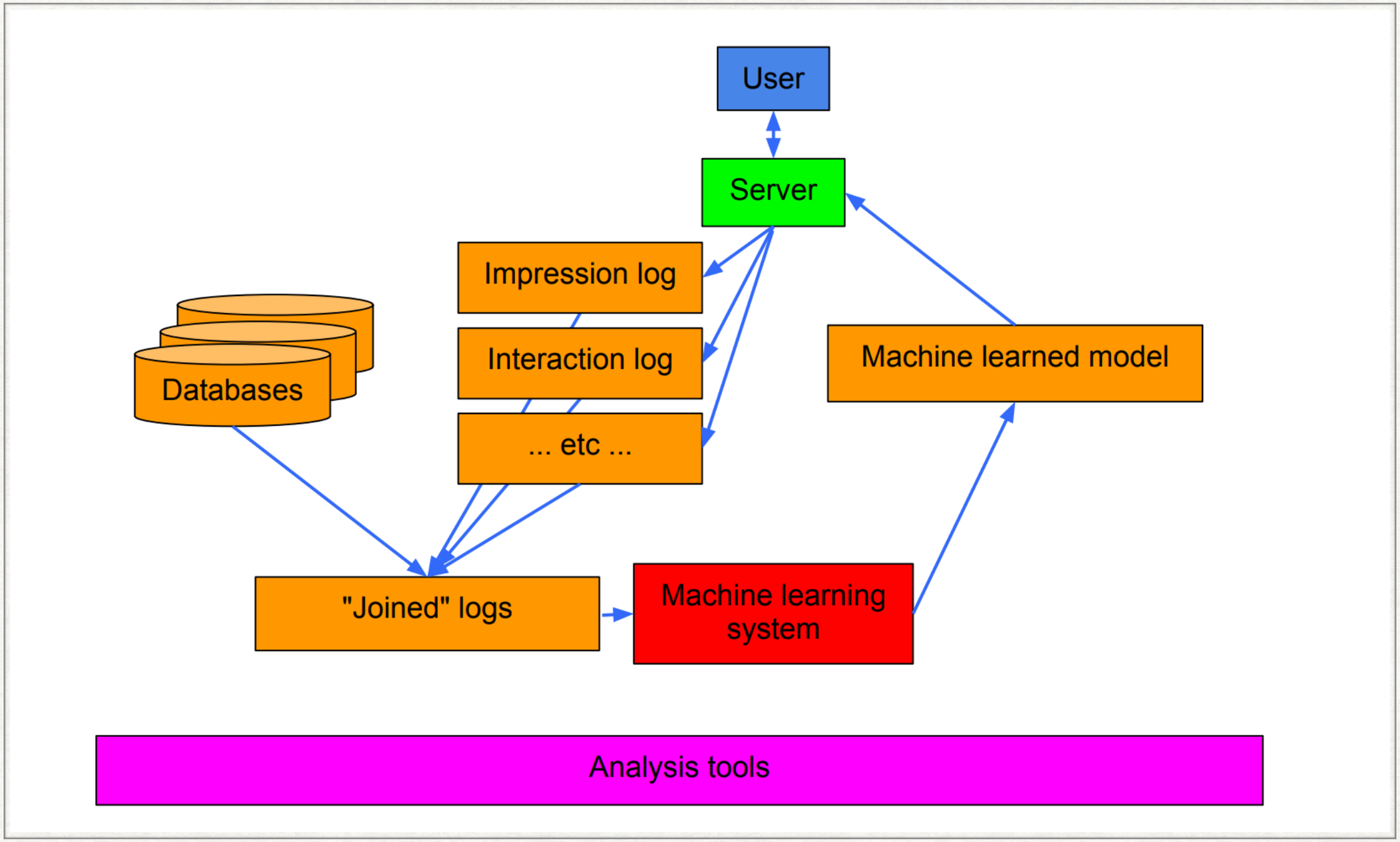


- Ads are ranked by:

$$p(click \mid Ad, user, scene) \times bid\_price(Ad)$$

- Our goal is here to : predict the click-through-rate of this Ads.

SYSTEM ARCHITECTURE



- Google Sibyl

MACHINE LEARNING APPROACH

- Represent {Ad,user,scene} as a feature vector x, let y (clicked or not clicked) be the label, then model p (y | x)
- A common way

$$p(y \mid x, w) = \frac{1}{1 + \exp(y \langle x, w \rangle)}$$

- Then learn w by **FACTORIZATION MACHINE**
- Also increasing interests on Deep learning

FEATURE ENGINEERING

- Feature Engineering is the most effective way to improve the model performance
- Often contain multiple feature groups
- Three major sources : ads, users, advertisers

N-GRAM

- Uni-gram:international, flower, delivery…
- Bi-gram : international flower,flower delivery
- For short text, even desirable generate all possible n-grams, then filter out unimportant ones

IMPORTANT THINGS

- * Style - bold style really catches eyes.
- * Personalization - User profile,Ad profile…
- * Feature combination
  - Produce a combination group of feature

DATA SCALE OF AD CTR

- Only 1 year search log produces 2 trillions examples.
- Sub-sampling? Not works because of personalization.
- The feature for this task is mainly **16 hundred billions of features.**

INDUSTRY DATA SCALE

- * 100 billions of samples
- * 10 billions of features
- * 1T - 1P training data

```
1102654752 Jun 30 17:18 part-13075.gz
1099726070 Jun 30 17:18 part-13076.gz
1101590533 Jun 30 17:18 part-13077.gz
1100016199 Jun 30 17:19 part-13078.gz
1100637016 Jun 30 17:19 part-13079.gz
1102254166 Jun 30 17:20 part-13080.gz
1103309165 Jun 30 17:21 part-13081.gz
```

WHERE TO STORE THE DATA

- * Lots of disks
- * Fail at any time
- [hardware]

ACCESS PATTERN

- Files are large10GB
- * sequential read and append.

GOOGLE FILE SYSTEM

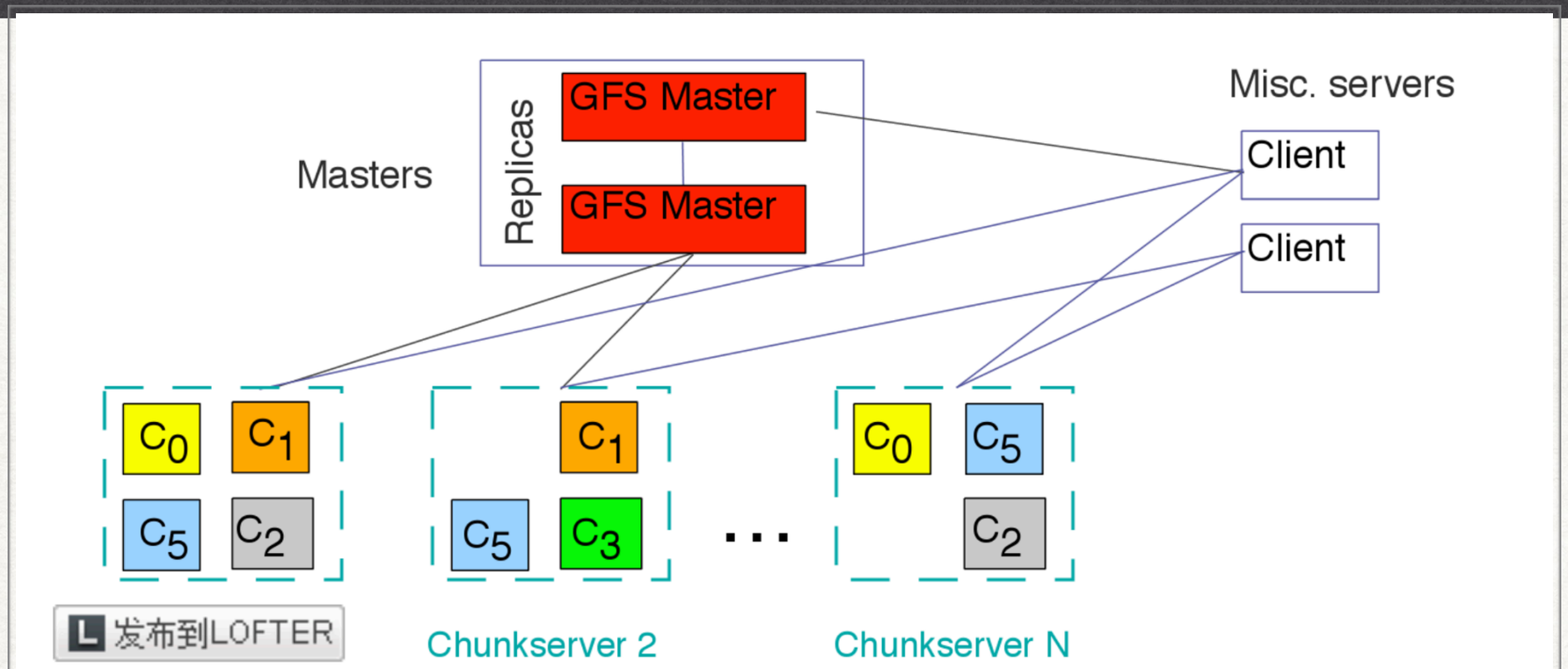


图1. GFS的架构图

- ✦ Data are replicated
  - ★ write success only if all replicas are done
- ✦ Request data:
  - ★ ask master for the location
  - ★ ask chunk server for the data
- ✦ New generations: Colossus

HDFS

- ✦ Open source implementation of GFS
- ✦ operations:
  - ★ hadoop fs -ls, -get, -put, -head, -cat, ...
  - ★ libhdfs: C API
  - ★ mount to local filesystem
- ✦ A little bit slower than GFS (personal experience)
- ✦ Large delay
  - ★ hadoop fs -ls /xxx (8000 files)
- ✦ Sometimes reading the training data uses more times than training

CONCLUDE 1

- ✦ 数据长尾分布
- ✦ 广告预测任务需要用机器学习模型去二分预测CTR
- ✦ CTR可以用特征工程，特征组合去做优化
- ✦ 数据量巨大的时候，怎么存，DISK，datacenter, google file system,HDFS

SPARSE TRAINING DATA

HIGH-DIMENSIONAL DATA ARE SPARSE

- ✦ Why high dimension?
  - ★ make the classifier's job easier
  - ★ linear method is often good enough
- ✦ Why sparse?
  - ★ easy to storage (only store non-zero entries)
  - ★ affordable computation cost
- ✦ Key difference to dense data when using
  - ★ require a lot of random read/write

STORE AND COMPUTING

COMPRESS STORAGE

	0	1	2
0		10	20
1	3		
2		87	
3	57		

- ✦ Sparse row-major:

  offset = [0, 2, 3, 4, 5]

  index = [1, 2, 0, 1, 0]

  value = 10 | 20 | 3 | 87 | 57

- ✦ Sparse column-major:

  offset = [0, 2, 4, 5]

  index = [1, 3, 0, 2, 0]

  value = 3 | 57 | 10 | 87 | 20

✦Access a(i,j) under row-major:

```
k = binary_search(index[offset[i]], index[offset[i+1]], j)
return valid(k) ? value(offset[i]+k) : 0
```

Y=AX

✦ Sample C++ codes.

ALL OFFSET, INDEX, VALUE ARE READ SEQUENTIALLY

```
// matrix-vector multiplication: y = A * x
void SparseMatrix<V>::times(const V* x const, V* y) const {
                                    ; ++i) {  // i-th row
                                    j < offset[i+1]; ++j)
      y_i += x[index[j]] * value[j];
    y[i] = y_i;
  }
} else {
  memset(y, 0, sizeof(V)*rows());
  for (size_t i = 0; i < cols(); ++i) {  // i-th column
    V x_i = x[i];
    for (size_t j = offset[i]; j < offset[i+1]; ++j)
                                    ue[j];
}
```

WRITE Y SEQUENTIALLY
BUT READ X IN RANDOM

WRITE X SEQUENTIALLY
BUT WRITE Y IN RANDOM

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

10 times { 0.5 ns, 5 ns

10 times { 7 ns, 100 ns

slides by Jeff Dean

Google

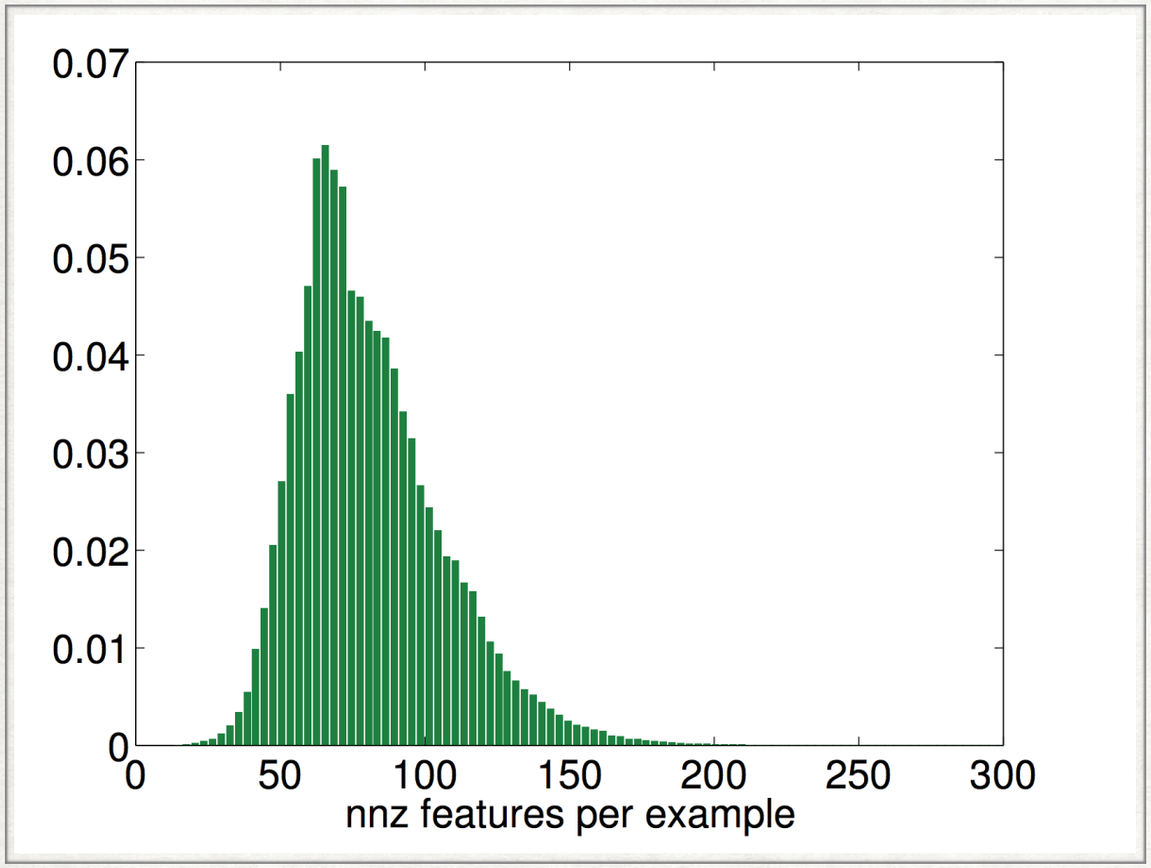
COST OF Y=AX

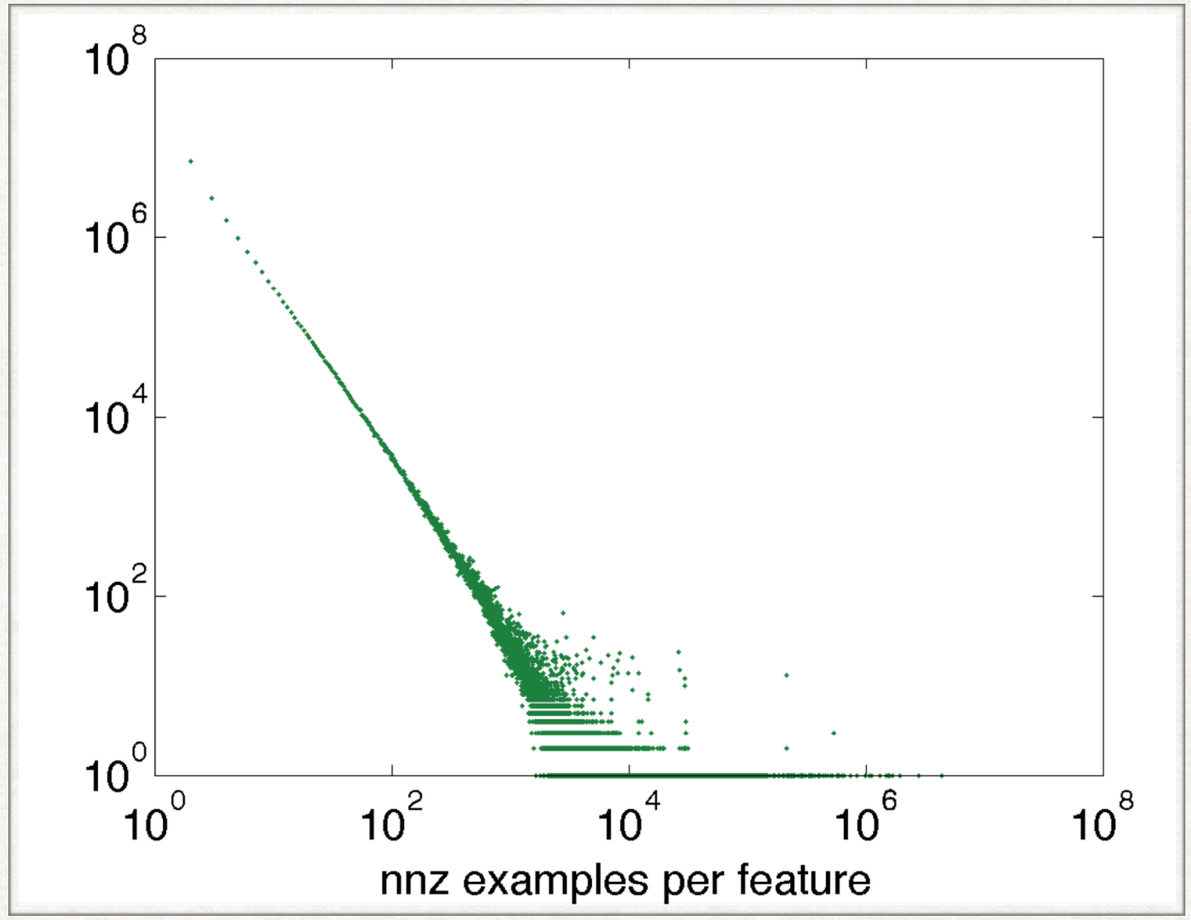
- ✦ The computation cost is O(nnz(A))
- ✦ The random access dominates the cost:

$$\approx L2-cache-reference(nnz(A))$$

- ✦ In theory: process 1.4e8 nnz entries per second
- ✦ In reality: 8.4e7 nnz entries per second
  - ★ 4.3M x 17.4M sparse matrix
  - ★ mac mbp, Intel i7 2.3GHz cpu
  - ★ single thread

PATTERNS OF SPARSITY





✦ Non-zero entries are distributed irregularly on features

★ imbalanced workload partition

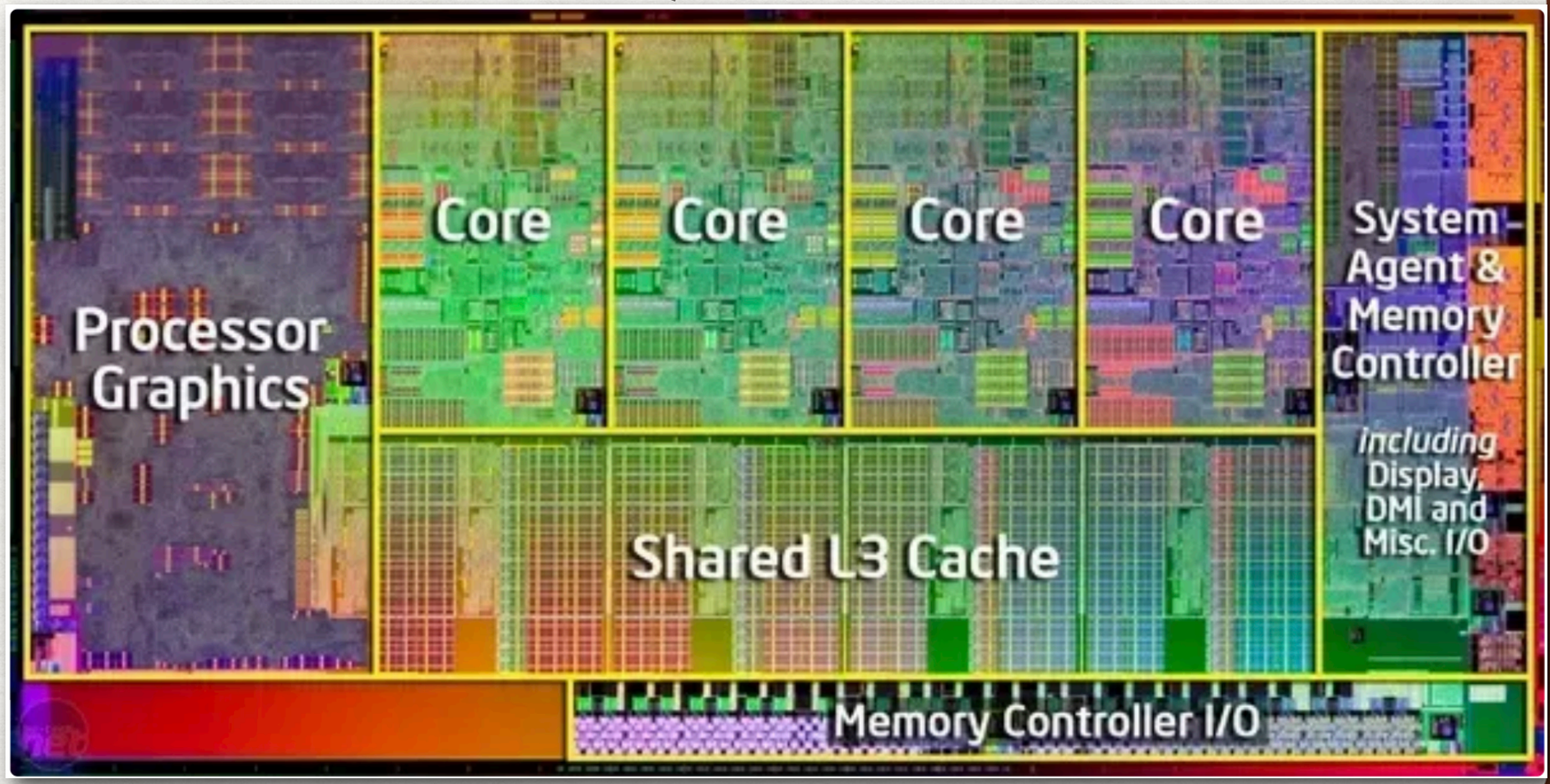
★ ill conditional number

MULTI-THREAD IMPLEMENTATION

CPU

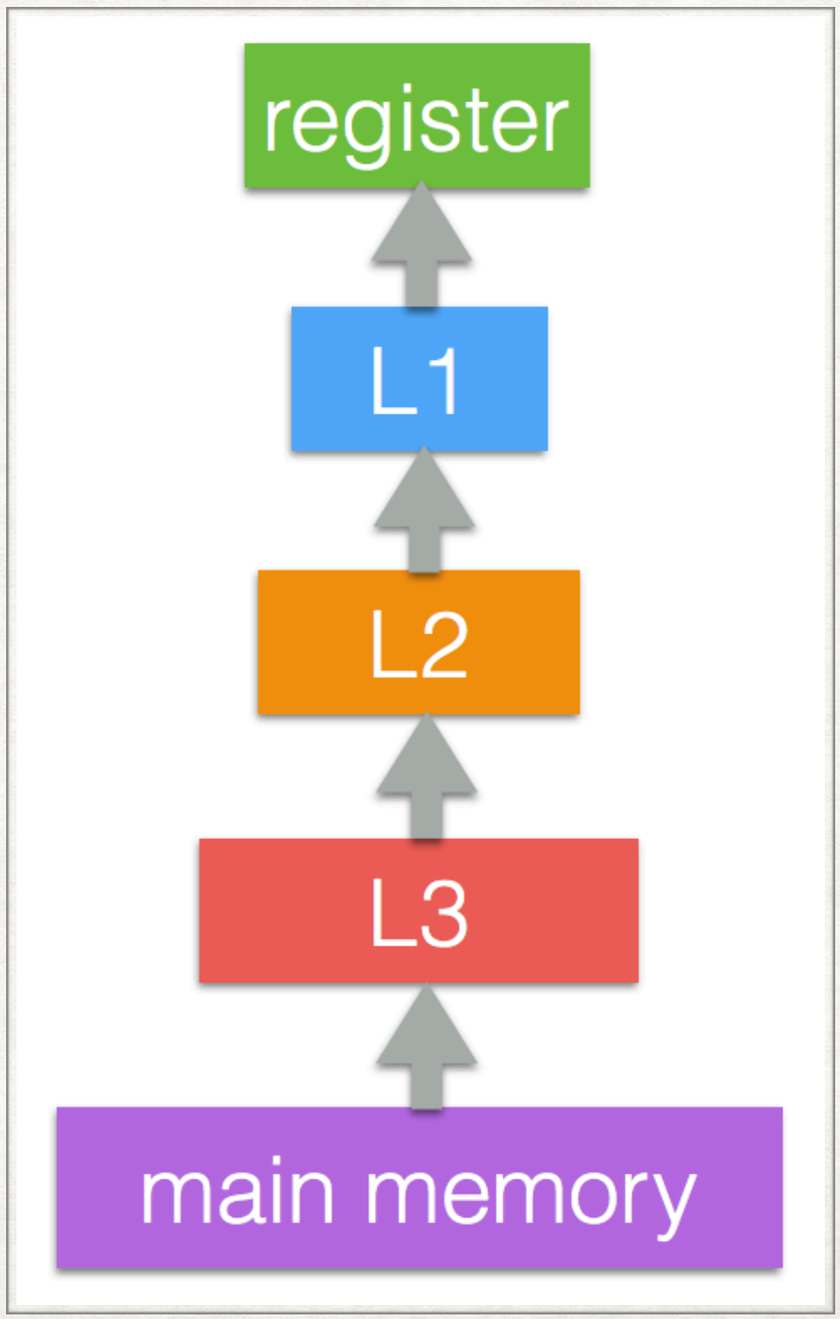
- Multiple cores (4-8)
- Multiple sockets (1-4)
- 2-4 GHz clock
- Memory interface 20-40 GB/s
- Internal bandwidth > 100GB/s

core i7



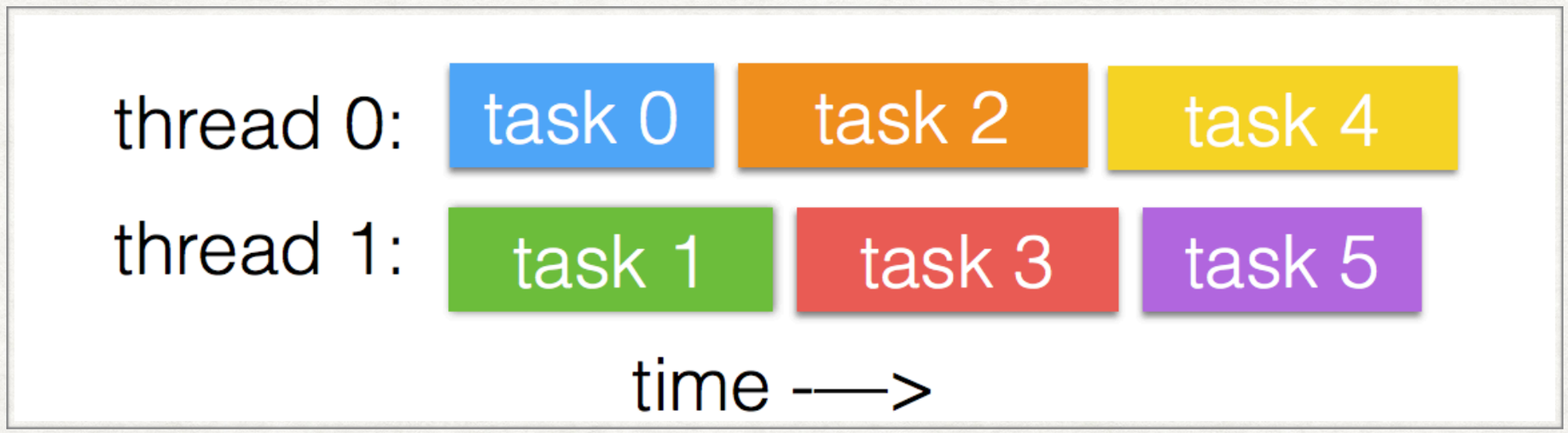
BENEFITS OF MULTI-THREAD

- Use more computation units
  - Float point units
- Hide memory latency
  - Run something else when the data are not ready
    - Fetch data from memory ~100 cycles



USING THREAD POOL

- ✦ A pool of threads, each one keeps fetching and executing unfinished tasks
- ✦ Create a pool with n threads: ThreadPool pool(n)
- ✦ Add a task into the pool: pool.add(task)
- ✦ Start executing: pool.startWorkers()



MULTI-THREAD Y= AX

- ✦ Assume row major
- ✦ Compute a segment of y

```
void rangeTimes(SizeR row_range, const V* const x, V* y) const;
```

- ✦ Divide y into several segments, each one is assigned to a thread

```
ThreadPool pool(num_threads);
int num_tasks = rowMajor() ? num_threads * 10 : num_threads;
for (int i = 0; i < num_tasks; ++i) {
  pool.add([this, x, y, row_range, num_tasks, i](){
      rangeTimes(row_range.evenDivide(num_tasks, i), x, y);
    });
}
pool.startWorkers();
```

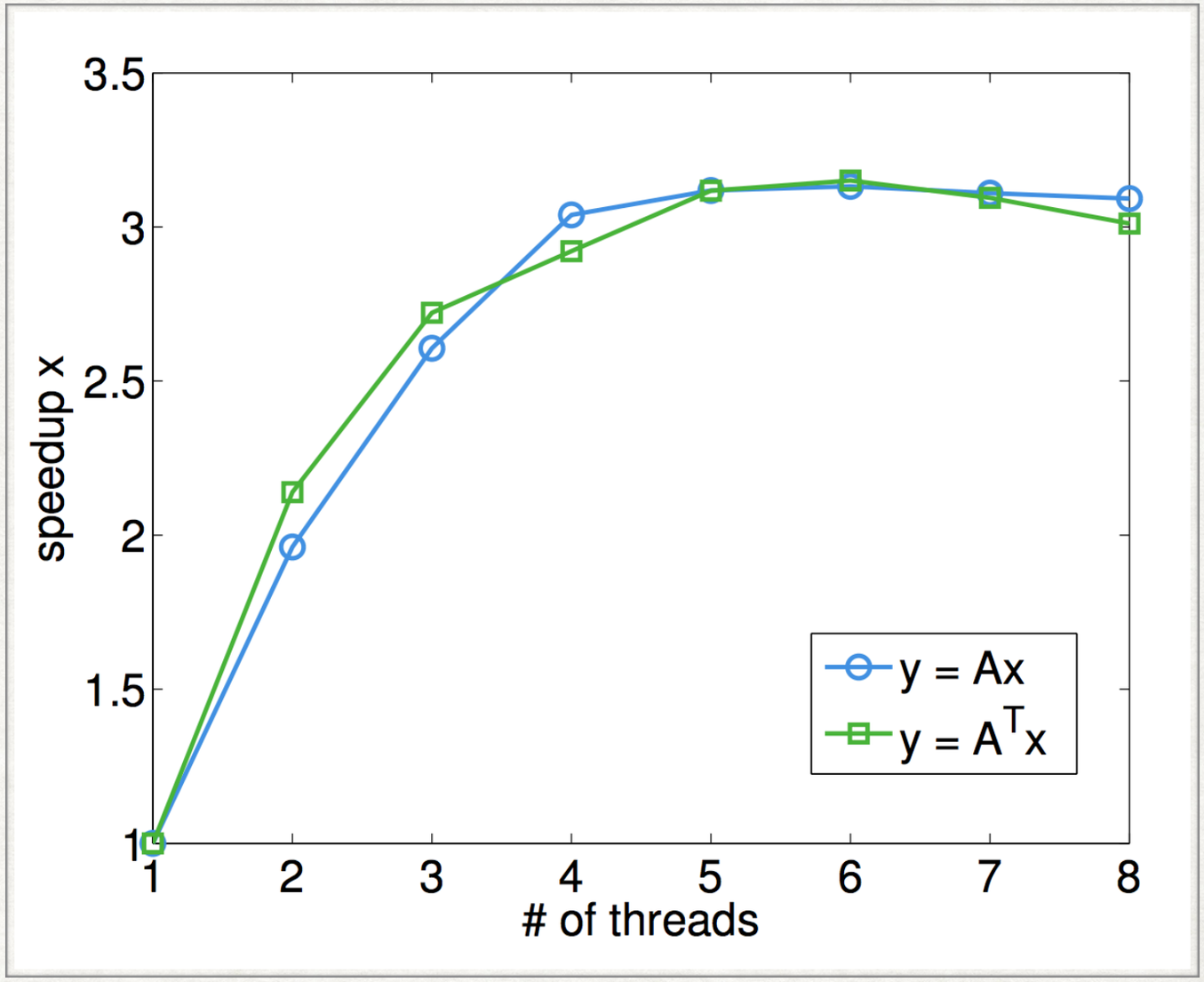
C++11 LAMBDA FUNCTIONS

HOW ABOUT COLUMN-MAJOR?

- ✦ Equivalence to $x = A^T y$ for row-major A
- ✦ multi-threads concurrently write the same y
- ✦ Several possible solutions:
  - ★ convert into a row-major matrix first
  - ★ lock y[i] (or a segment) before write it
  - ★ each thread writes only a segment of y

EXPERIMENTS

- ✦ data: CTRa
- ✦ row major,
  4.3M rows,
  17.4M columns,
  354M nnz entries
- ✦ MBP Pro,
  Intel i7 2.3GHz,
  4 cores,
  8 hyper-threads



ROW MAJOR OR COLUMN MAJOR

✦No big difference for individual and whole access

★Choose the one how data are stored

✦ Use row major when need read individual rows

★ SGD, minibatch SGD, online learning

✦ Use column major when need read columns

★ (block) Coordinate descent

✦ Converting cost 2*nnz(A) random access

MORE

- Other Operations? BLAS,LAPACK
  - ★ Timothy A. Davis, Direct Methods for Sparse Linear Systems, SIAM, 2006
- Existing packages:
  - ★ SuiteSparse, Eigen3…
  - ★ Jeff Dean is using it…

Bug 613 - Bug in internal::psqrt SSE implementation

Status: RESOLVED FIXED	**Reported**: 2013-06-13 17:54 UTC by Jeff Dean
Product: Eigen	**Modified**: 2013-06-14 09:52 UTC (History)
Component: Core - general	

add subtract	`mat3 = mat1 + mat2;` `mat3 = mat1 - mat2;`	`mat3 += mat1;` `mat3 -= mat1;`
scalar product	`mat3 = mat1 * s1;` `mat3 = mat1 / s1;`	`mat3 *= s1;` `mat3 /= s1;`

matrix/vector products *	`col2 = mat1 * col1;`
	`row2 = row1 * mat1;`
	`mat3 = mat1 * mat2;`
		`row1 *= mat1;`
		`mat3 *= mat1;`

transposition adjoint *	`mat1 = mat2.transpose();` `mat1 = mat2.adjoint();`	`mat1.transposeInP` `mat1.adjointInPla`

dot product
inner product *
	`scalar = vec1.dot(vec2);`
	`scalar = col1.adjoint() * col2;`
	`scalar = (col1.adjoint() * col2).value();`	

outer product *	`mat = col1 * col2.transpose();`	

CONCLUDE

- ✦ 高维稀疏数据，怎么优化存储——压缩存储法
- ✦ 高维稀疏数据，怎么优化计算——
- Y=AX
- 两种方法：一个是按行分开，一个是按列分开
- ✦ 多线程模型——Thread pool
- ✦ 按行分 or 按列分 去并行计算 Y=AX

到现在我们都只是多线程的并行。没有多machine.
拆分的都是对A而言，数据拆分，没有对模型也就是x做拆分。

DISTRIBUTED OPTIMIZATION

MOTIVATION

✦For big data, machines are scarce resources

✦ We must find the solution as economic as possible

★ saving several iterations may mean hundreds of machine hours

(hundreds of $ on ec2)

BATCH AND ONLINE

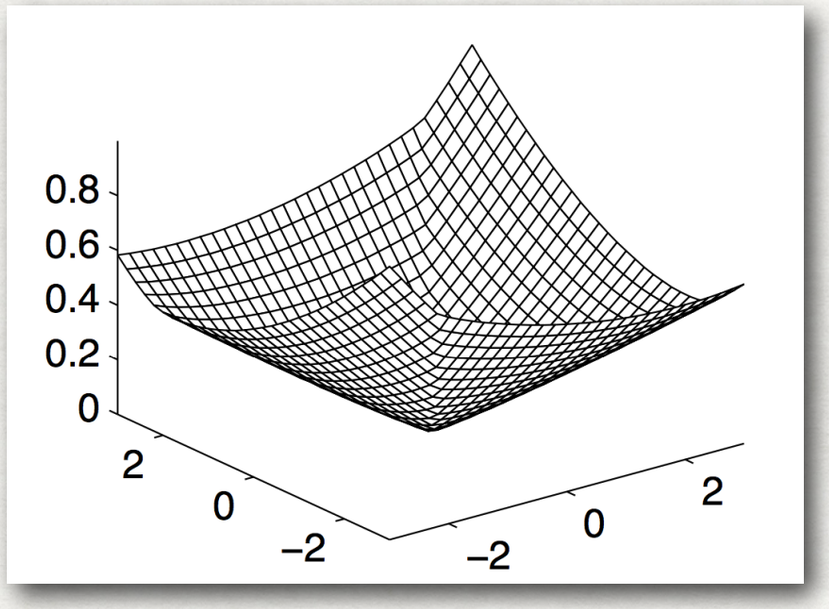
✦BATCH ——Very large dataset available

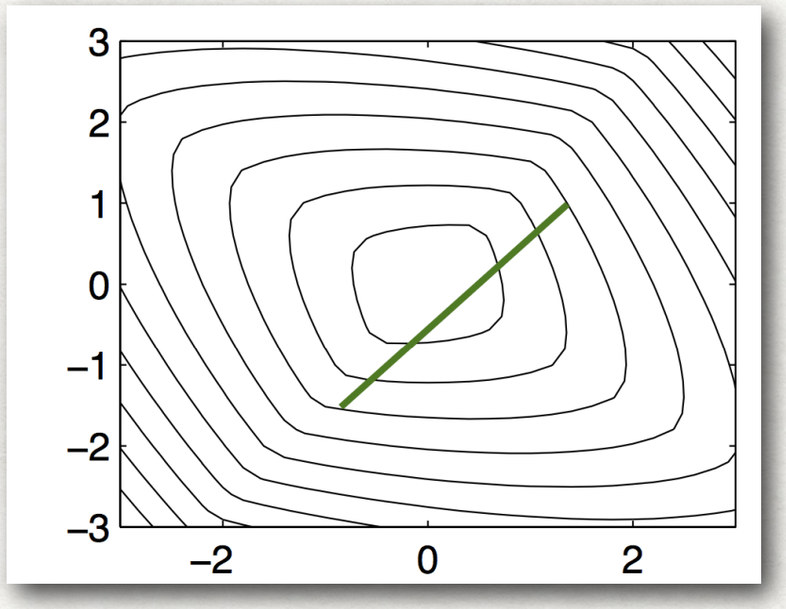
✦Online —— Data -> Compute -> Loop

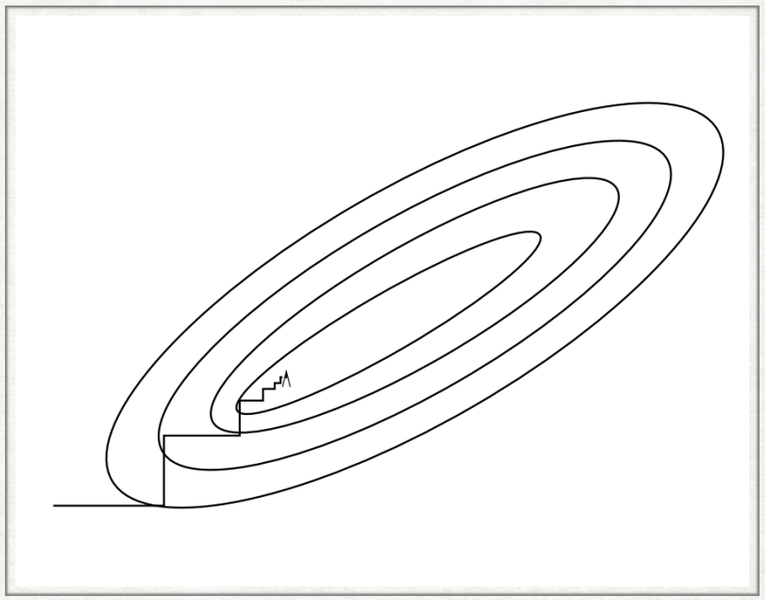
ISSUE FOR SPARSE DATA

- ML is all about Gradient Descent

May sensitive to the learning rate







- $w^{t+1} = w^t - \eta \left[\sum_{k \leq t} \mathrm{diag}(\nabla f(w^k))^2 \right]^{-\frac{1}{2}} \nabla f(w^t)$

MULTI-THREAD

LOGISTIC REGRESSION

- The most expensive part: compute gradients
- For logistic loss:

$$f(w) = \text{sum}(\log(1 + \exp(y. * (Xw))$$
$$\nabla f(w) = X^T(y./(1 + \exp(y. * (Xw))))$$

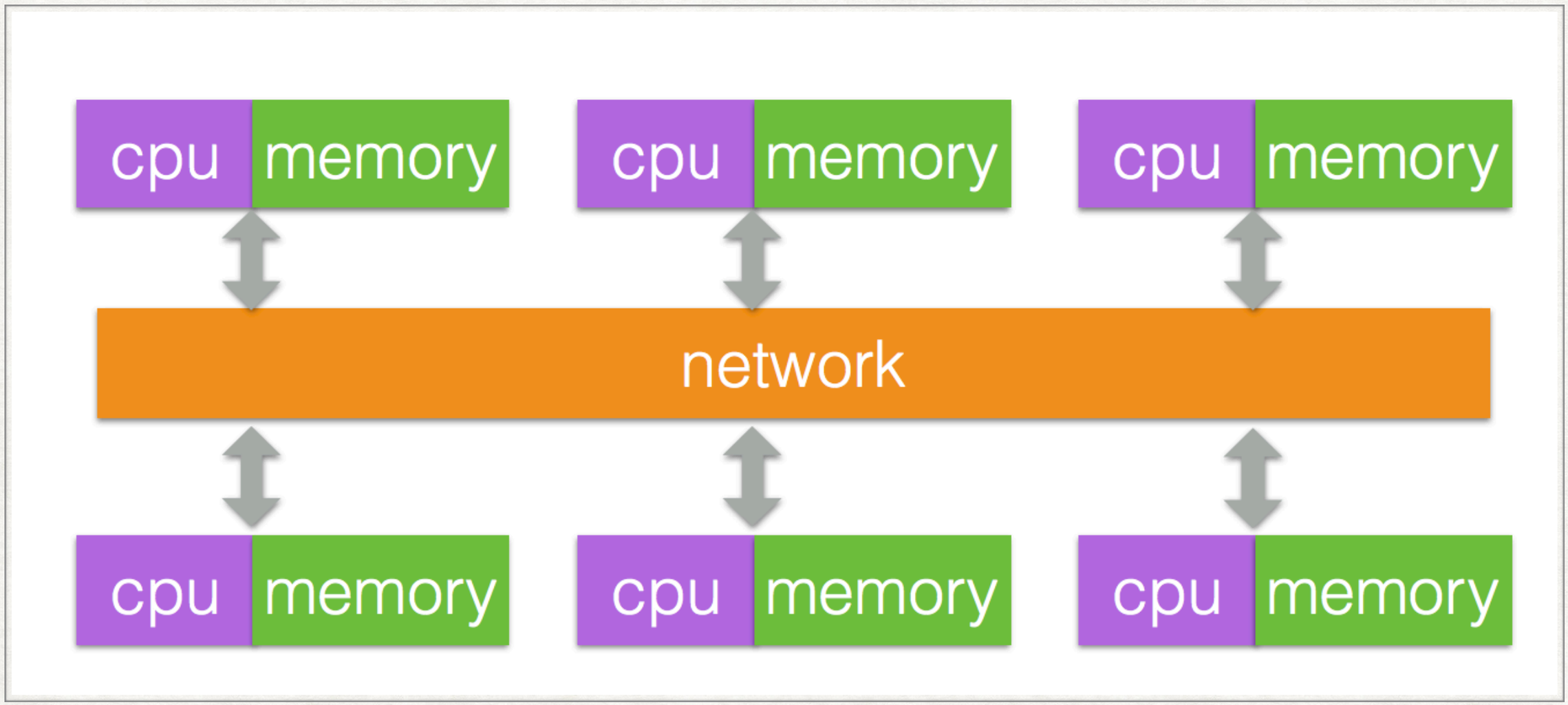
- Two key steps

- Multi-threaded by previous technologies

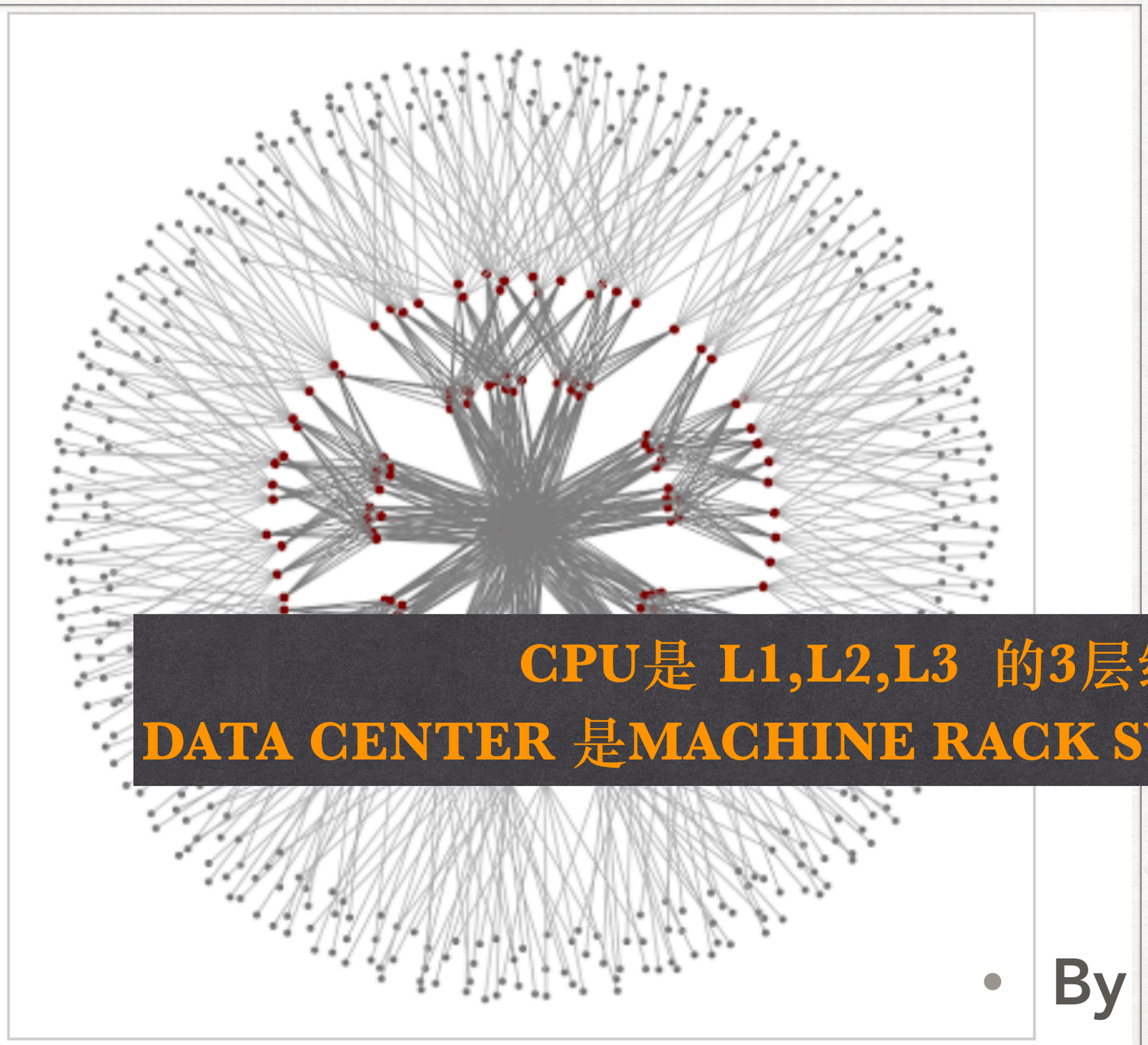
DISTRIBUTED IMPLEMENTATION

DISTRIBUTED ARCHITECTURE

- Data are exchange via network



DATA CENTER



- 3-level fat-tree
- 432 servers
- (grey)
- 180 switches
- (red)

CPU是 L1,L2,L3 的3层级
DATA CENTER 是MACHINE RACK SWITCH 的3层级

- By Brighten Godfrey

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

500 times

slides by Jeff Dean

Google

DATA PARTITION

- Assume Data is not so large

W

X

Y = AX

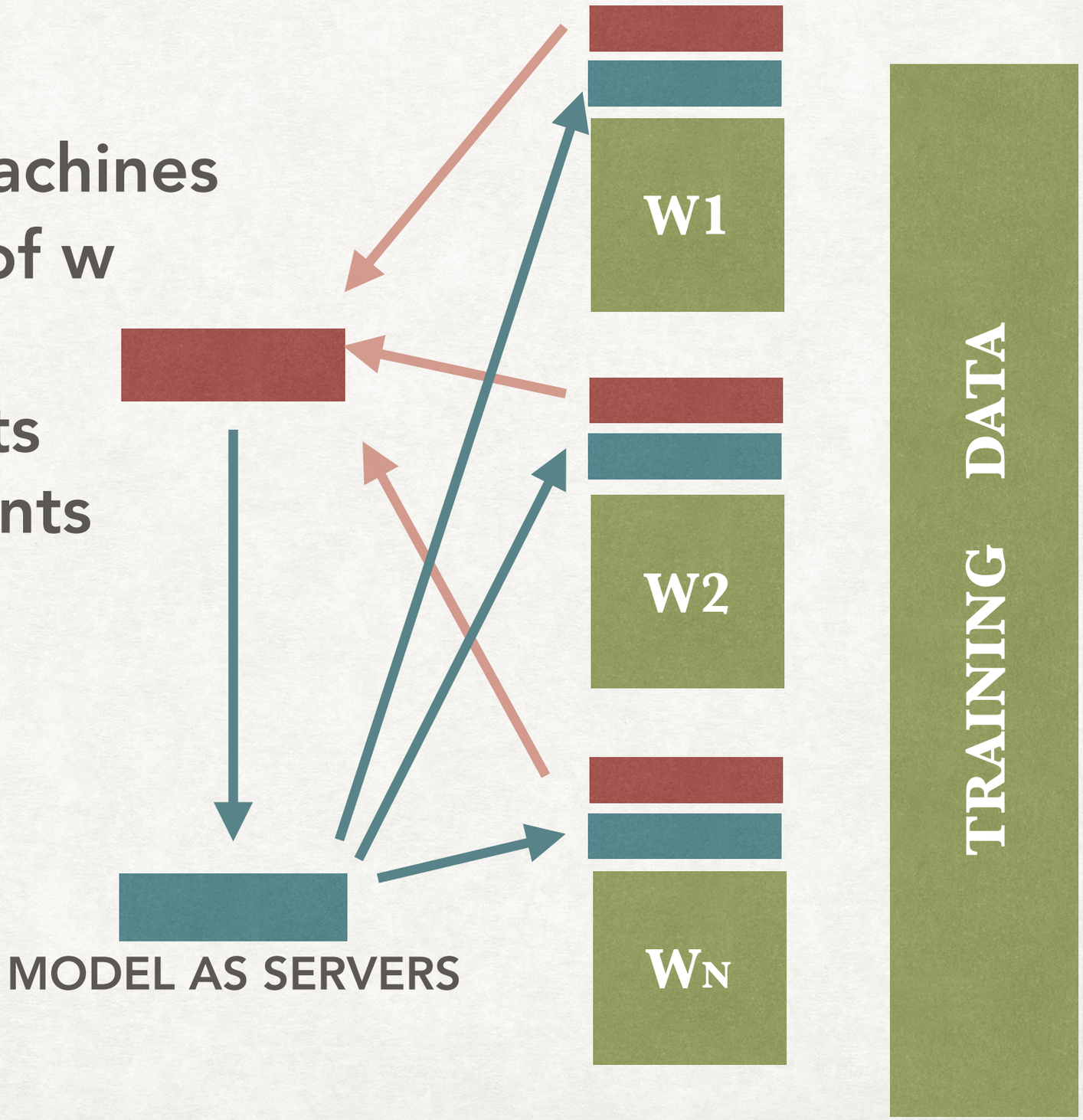
A 代指这里的 X

X 代指这里的 W

我们这里只是分割的X，（数据），没有分割到模型

DISTRIBUTED IMPLEMENTAION

Partition data into machines
Get the working set of w
Iterate t = 1, 2, ...
    Compute gradients
    Aggregate gradients
    Update w
Get updated w



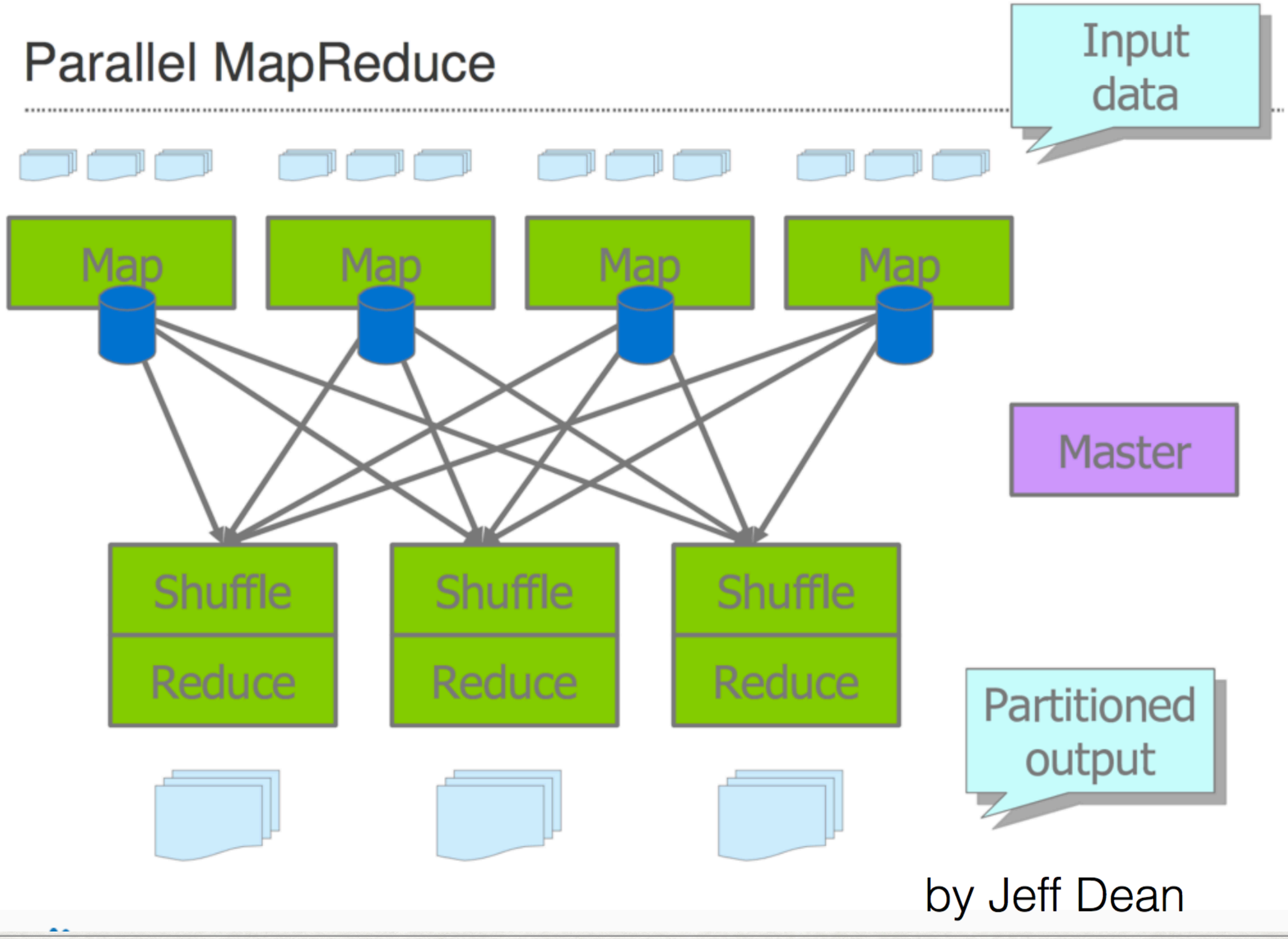
MAP REDUCE

- ✦ Proposed by Google, open source Hadoop, Spark
- ✦ Need to specify two primary methods
  - ★ map (k, v) —> [k’, v’]*
  - ★ reduce (k, [v]*) —> [k,v’]

IMPLEMENT GD

- ✦ Each iteration is a map/reduce
  - ★ map: each worker computes gradients, emits (feature_id, gradient_value)
  - ★ reduce: sum gradient with the same featured id

Parallel MapReduce



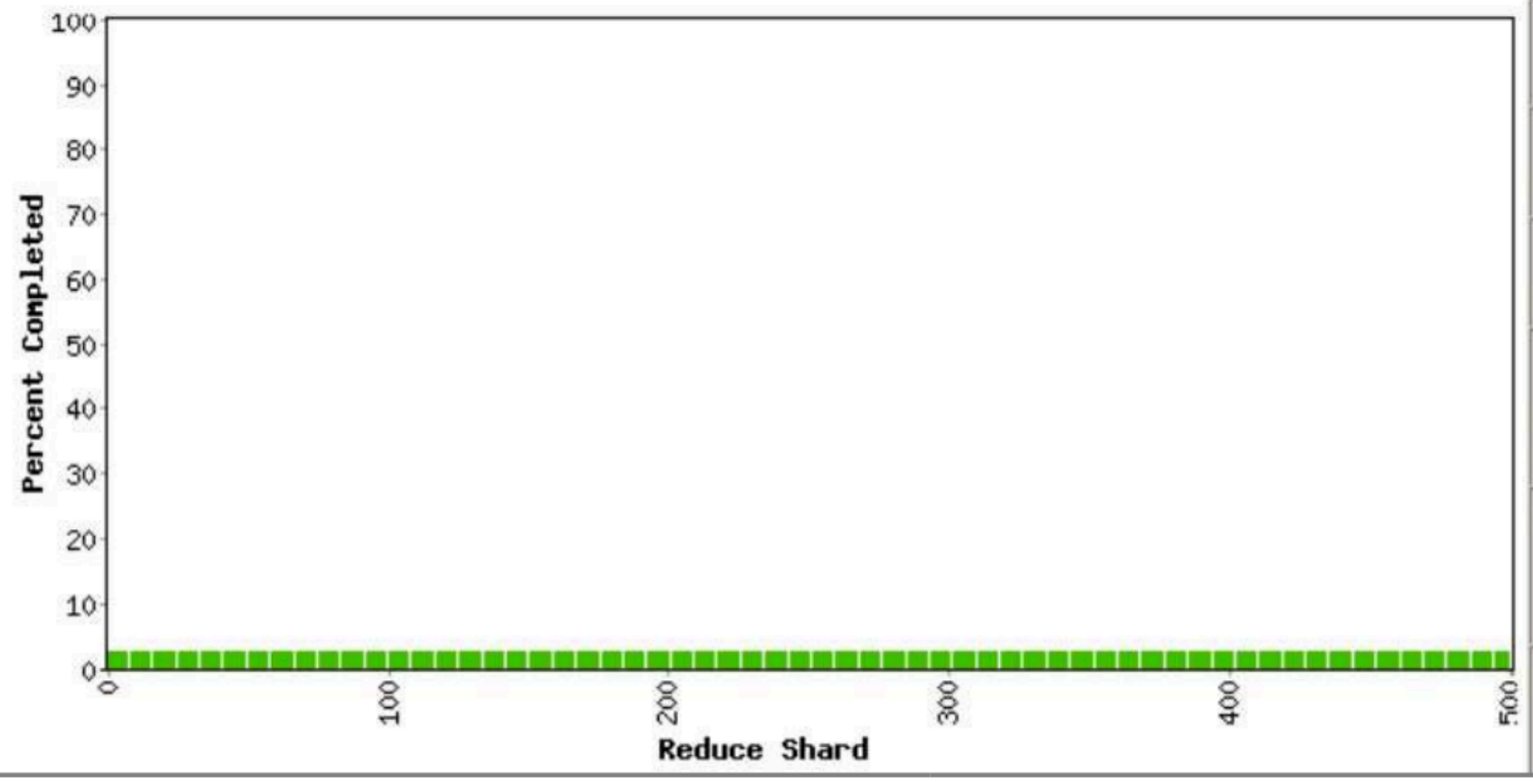
by Jeff Dean

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

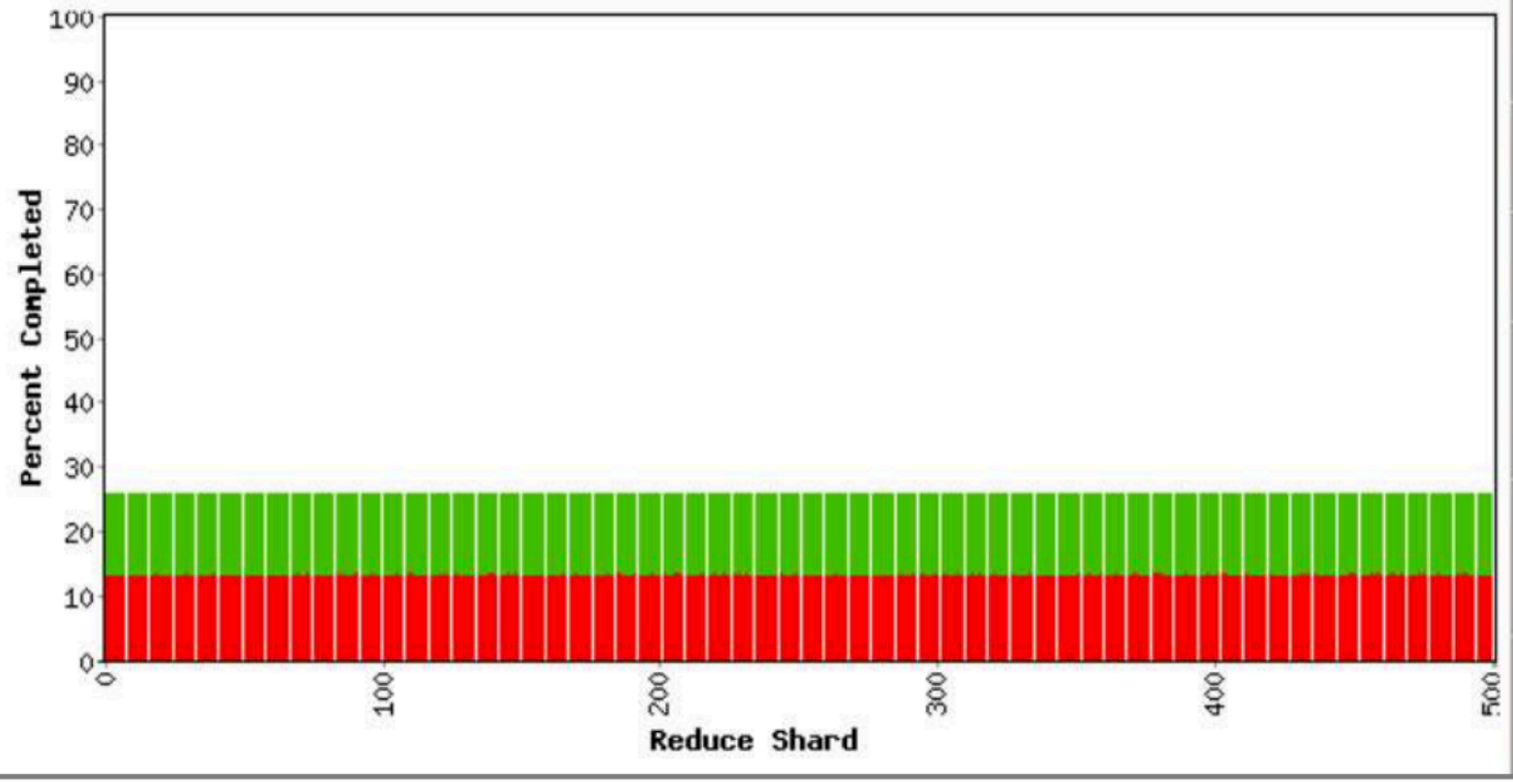
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

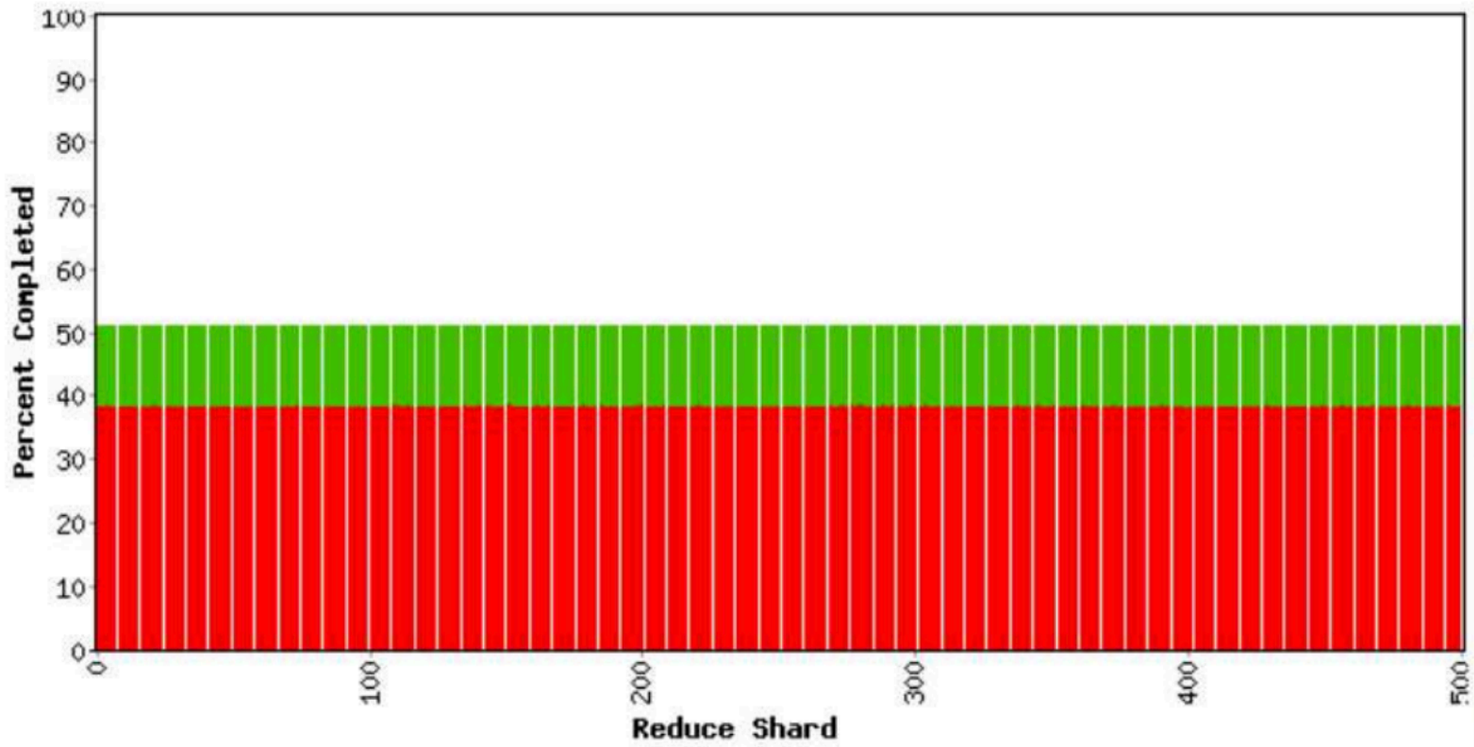
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

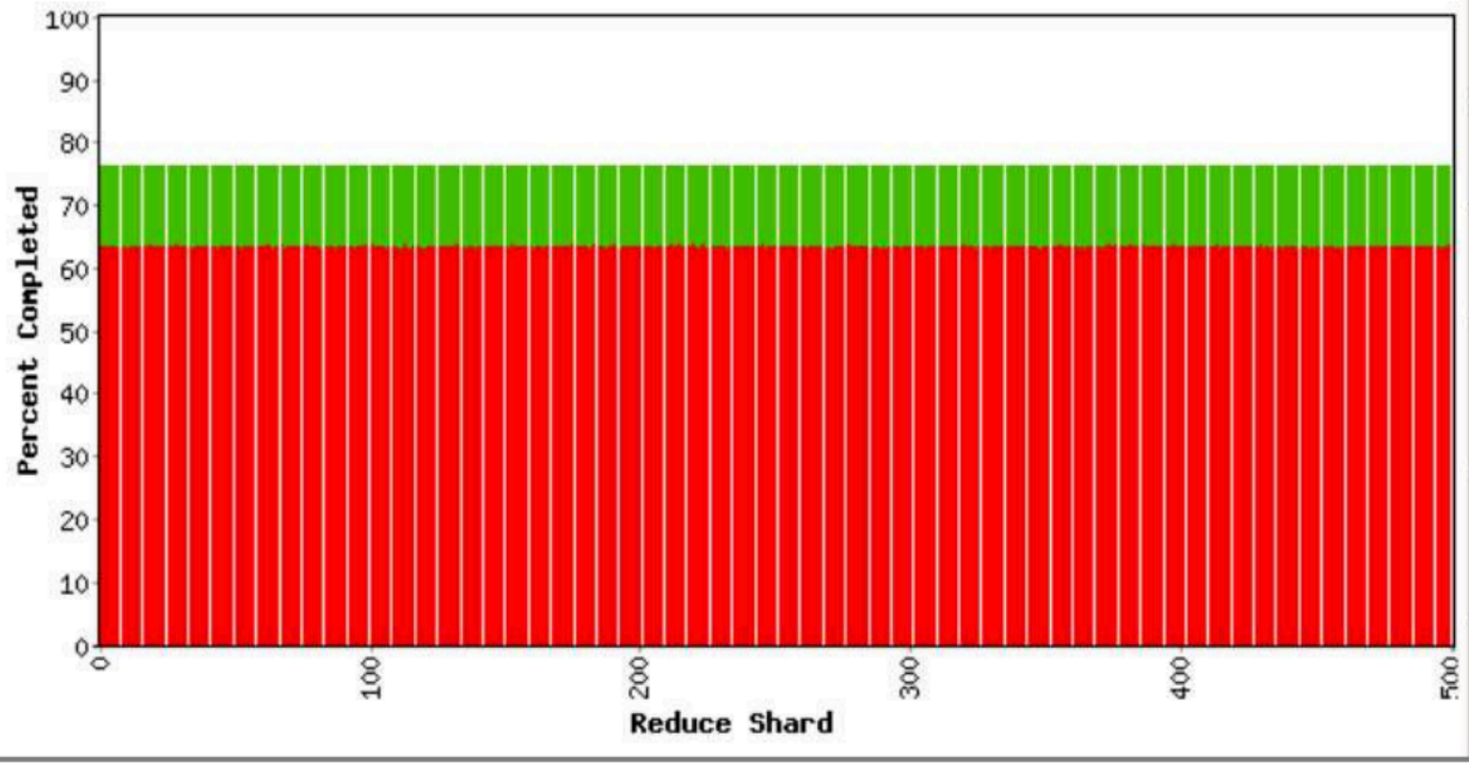
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

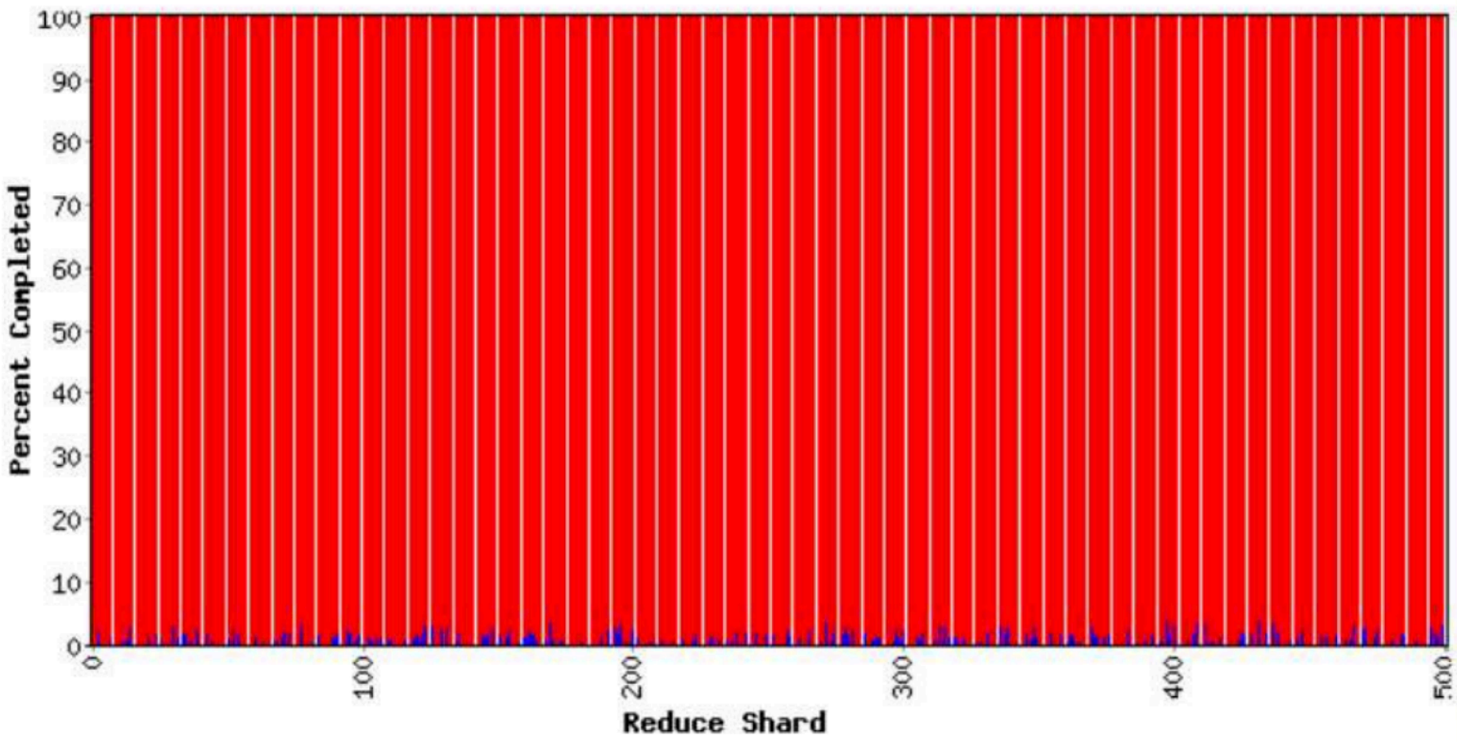
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	105
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

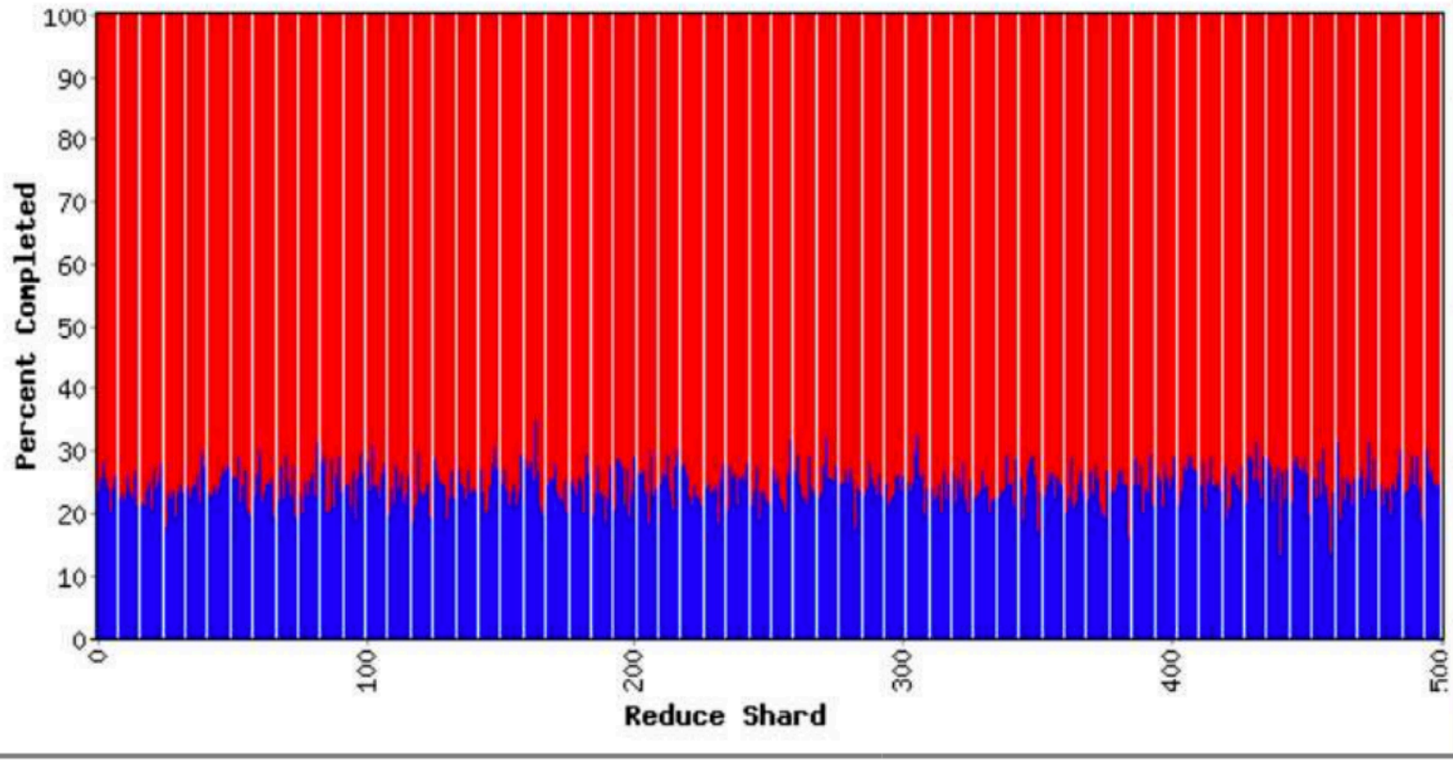
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

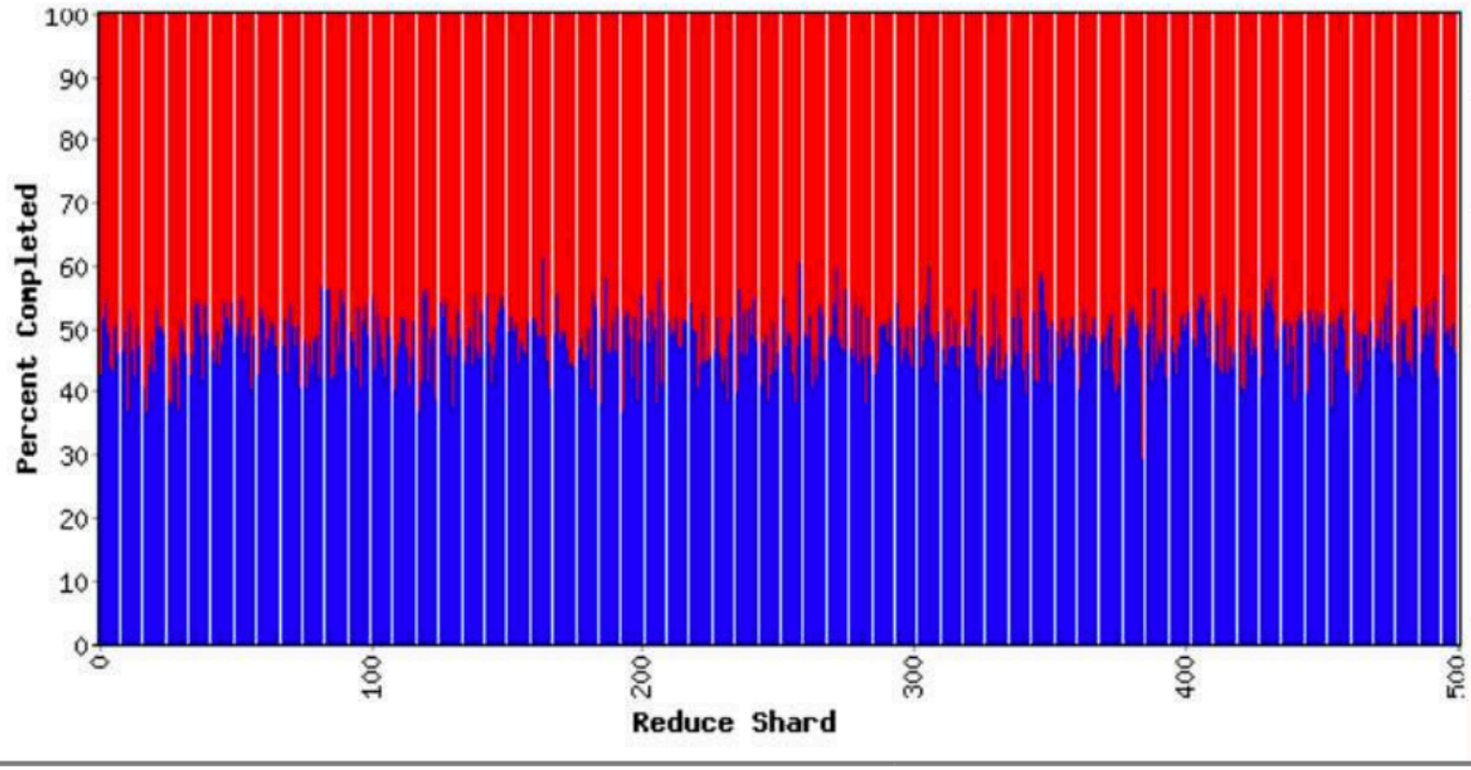
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

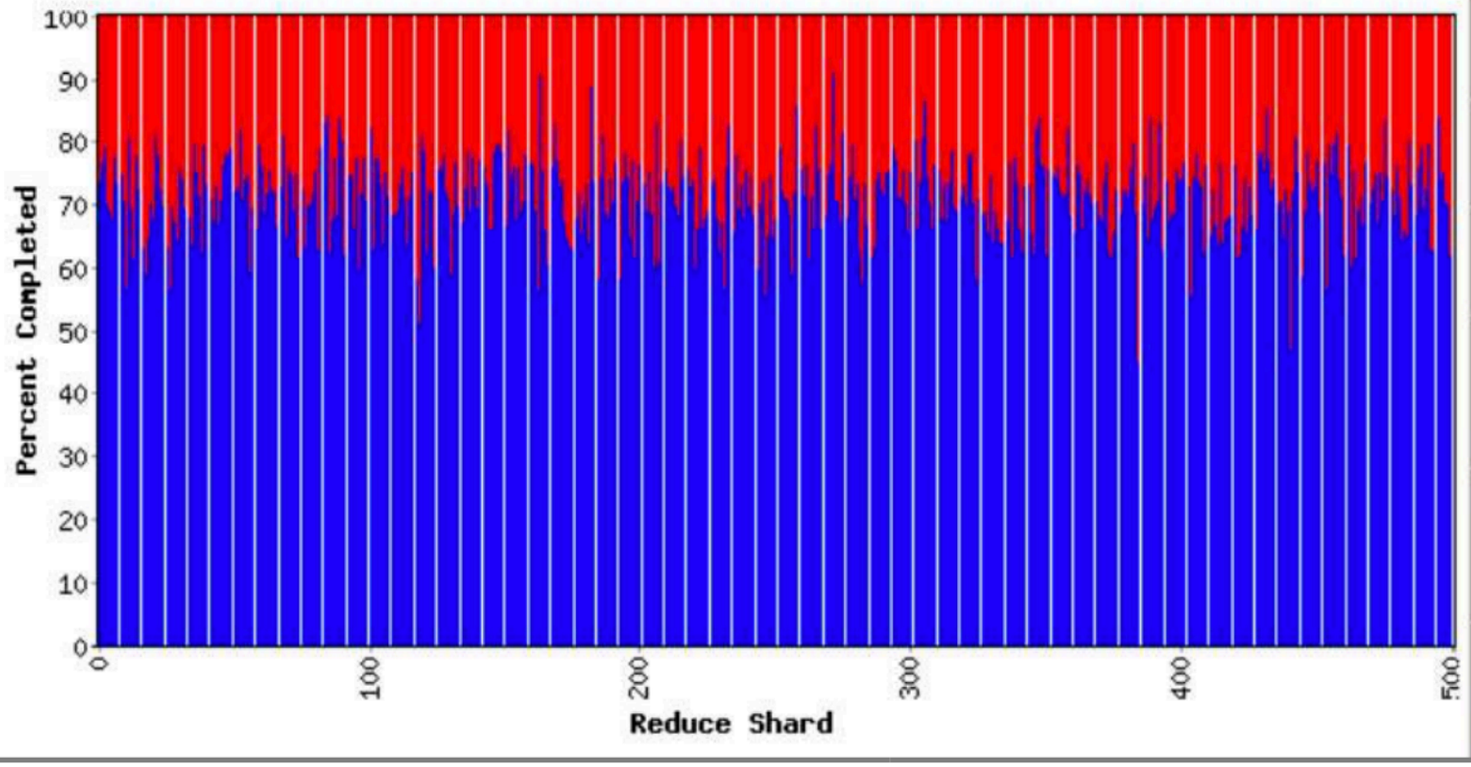
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

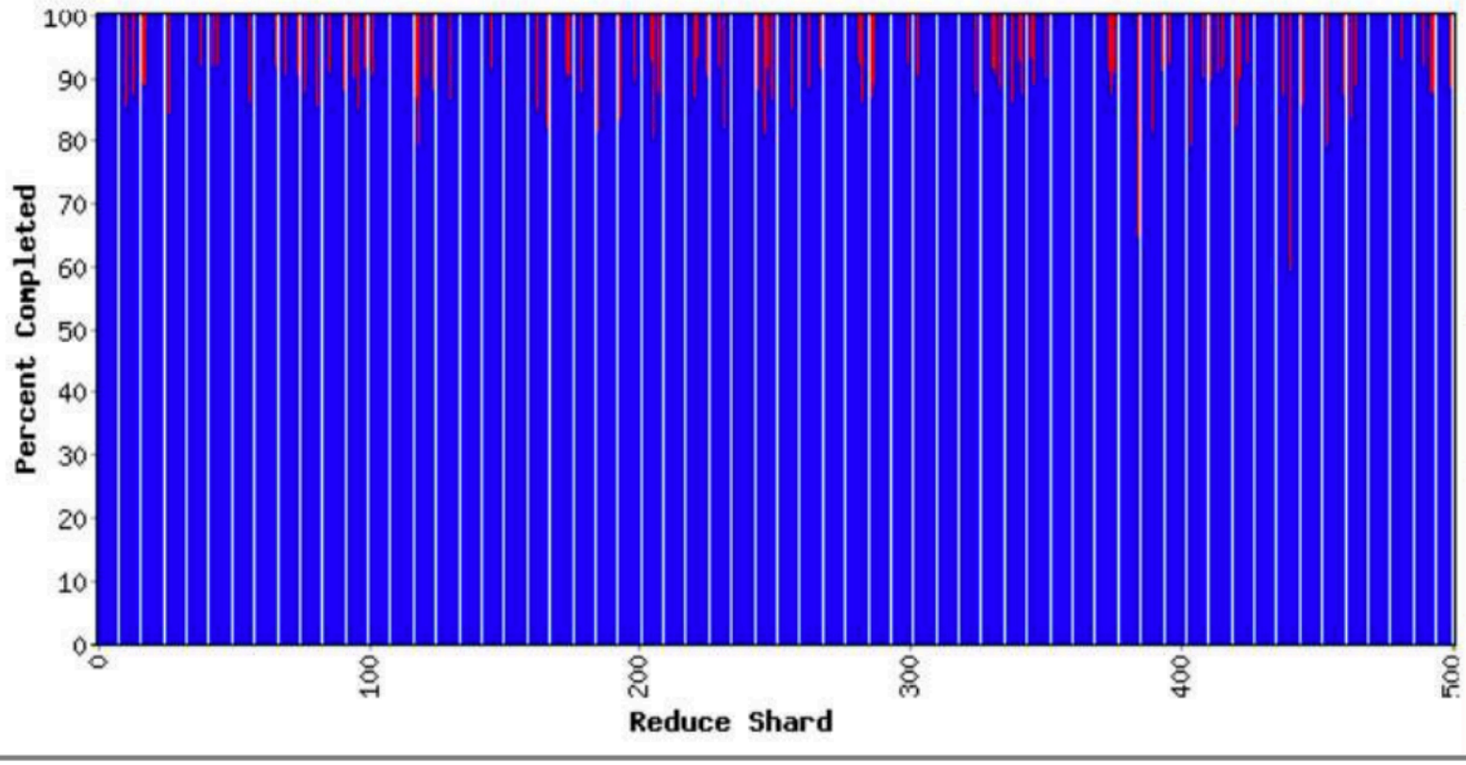
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

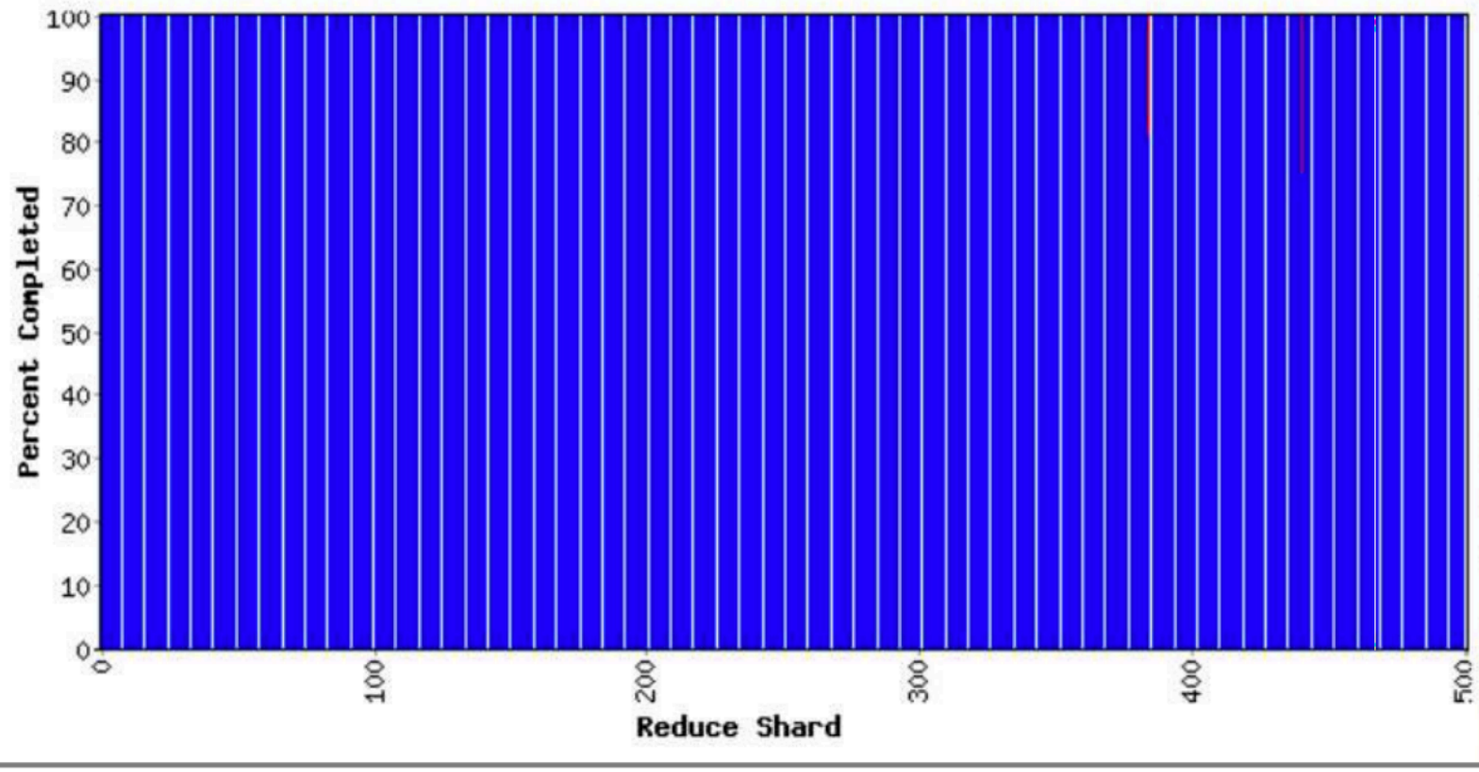
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	394792	
mr-merge-outputs	394792	

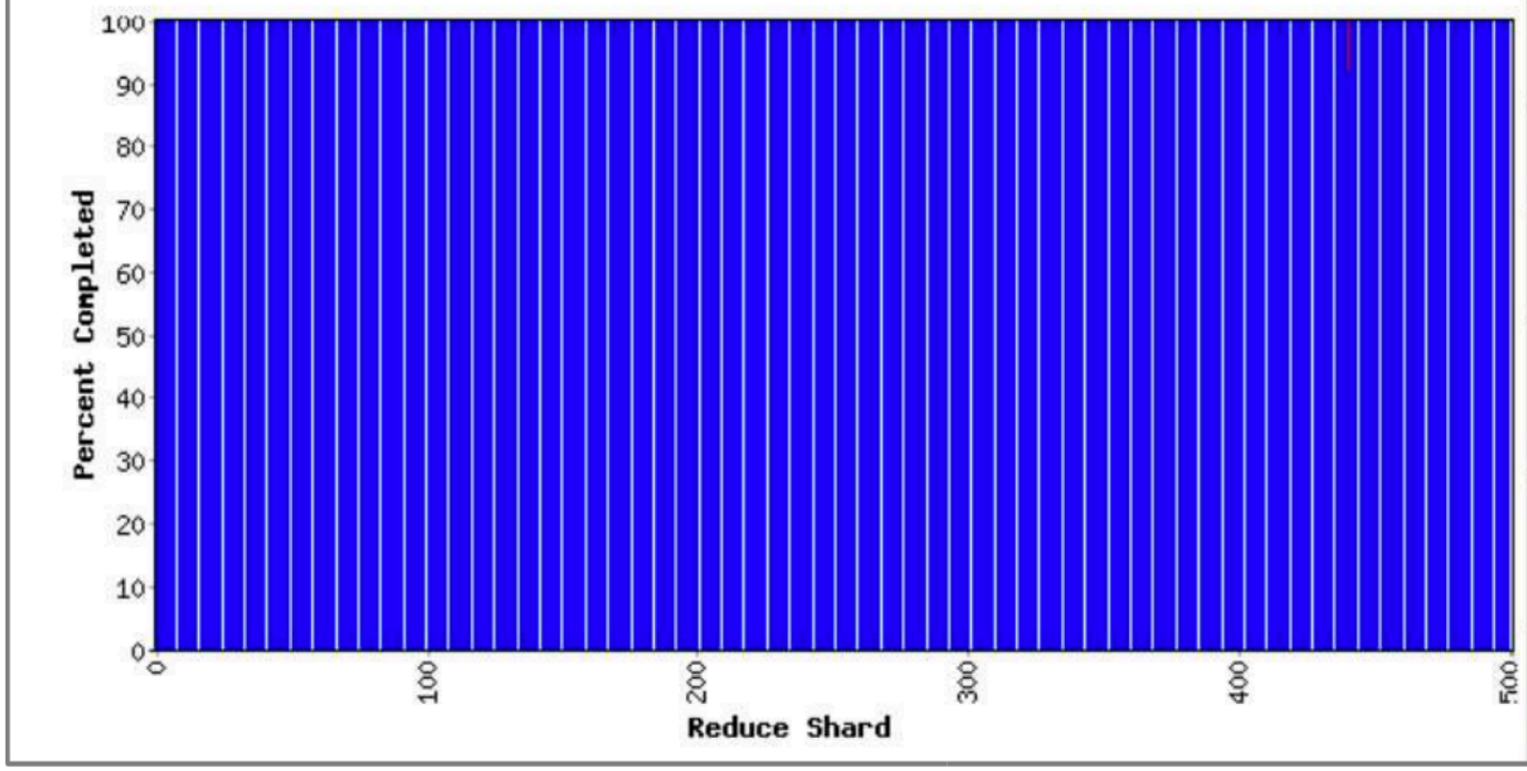
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

Google

PRACTICE GUIDE

- ✦ Use #map >> #reduce > #machines
  - ★ fine granularity tasks
  - ★ better load balance
- ✦ Do not use it if you have a lot of iterations, 10 is fine, 100 is terrible
  - ★ good for feature extraction
  - ★ often not good for optimization

MPI

A message passing interface

- ✦ Proposed and implemented by HPC community
- ✦ OpenMPI: http://www.open-mpi.org/
- ✦ mpich: http://www.mpich.org/

BASIC MODEL

- ✦ Group
  - ★ ordered set of processes
  - ★ rank from 0 to N-1 (for N processes)
- ✦ Communicator
  - ★ group of processes that can communicate with each other
  - ★ default communicator: MPI_COMM_WORLD

A MPI PROGRAM STRUCTURE

- #include “mpi.h”
- initialize MPI environment: MPI_Init
- MPI_xxx()
- MPI_Finalize()

```c
int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  int my_rank, rank_size;
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &rank_size);
  printf("# of processses %d, my rank %d\n", rank_size, my_rank);
  MPI_Finalize();
}
```

COLLECTIVE ROUTINES

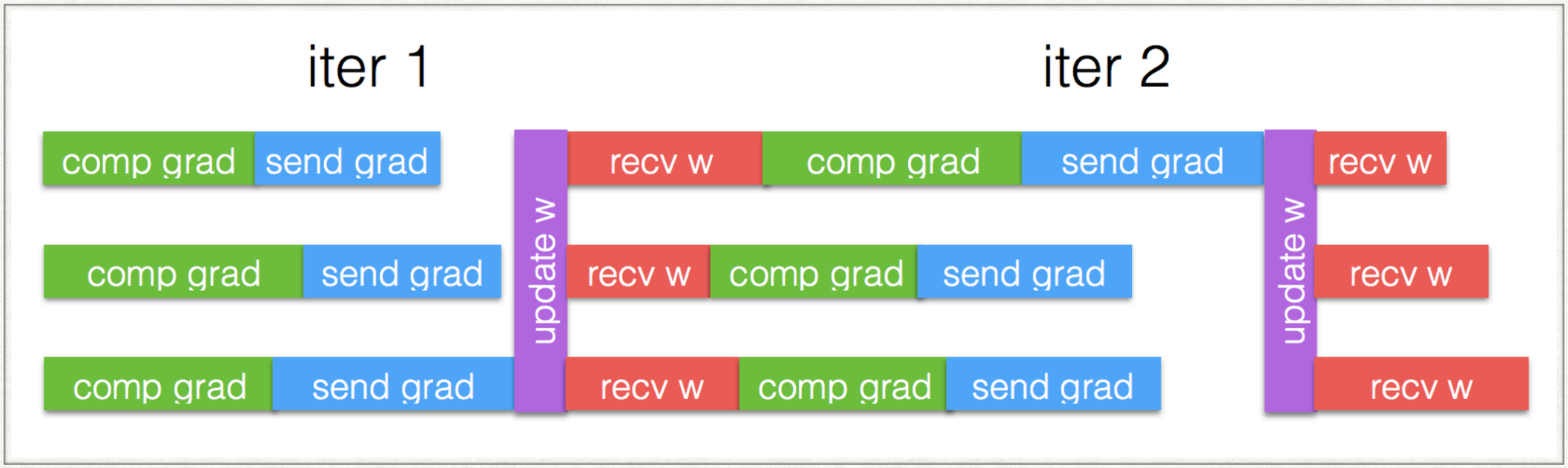
✦Reduce the gradients:

```
MPI_Reduce(local_grad.data( ),
global_grad->data( ),
feature_size,
MPI_DOUBLE,
MPI_SUM
0,
MPI_COMM_WORLD)
```

✦Broadcast the weight

```
MPI_Bcast(weight.data(),
feature_size,
MPI_DOUBLE,
0,
MPI_COMM_WORLD);
```

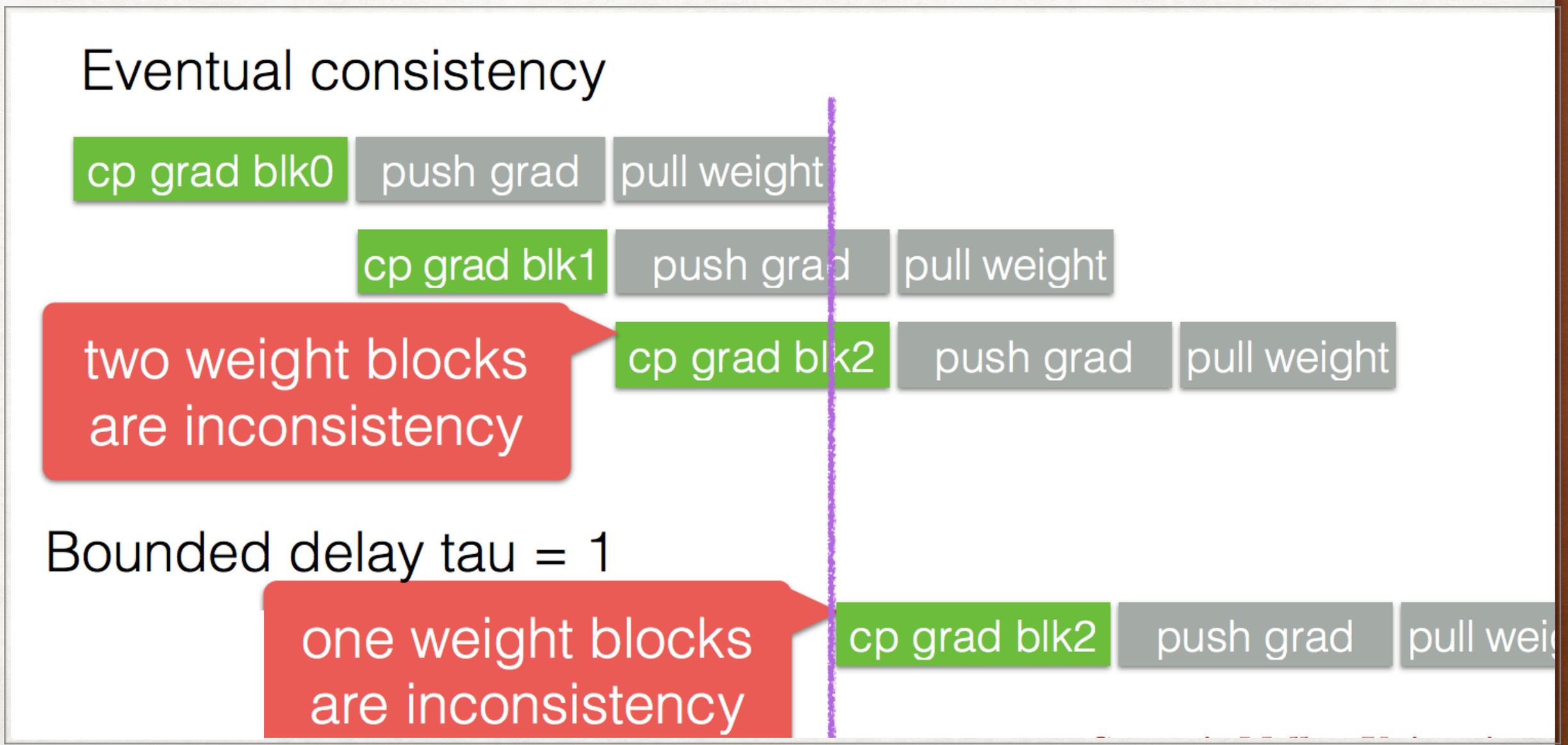
PERFORMANCE ANALYSIS



- ✦ CPU is idle when
  - ★ send gradient and receive weight
  - ★ waiting all machines are done
- ✦ We should parallel CPU and I/O

ASYNCHRONOUS UPDATING

- Parallel CPU and I/O, hide synchronization cost
- A worker:



ANALYSIS

✦ Lvar: correlation of features in a block

✦ Lcov: correlation of features between too neighbor blocks

✦ -bounded delay

✦ fixed learning rate guarantee convergence (even for non-convex problems)

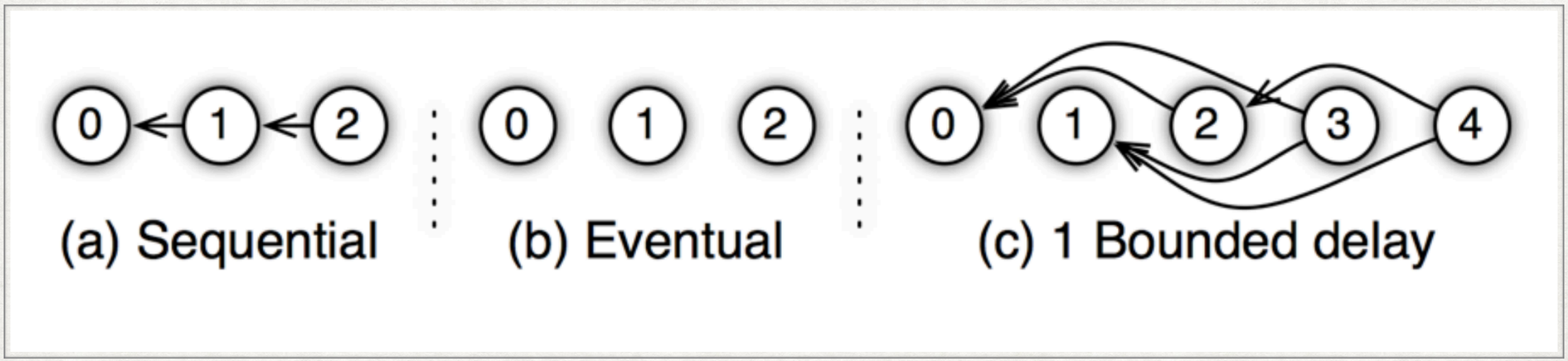
$$\eta_t = O\left(\frac{1}{L_{var} + \tau L_{cov}}\right)$$

make features in a block less correlated

make blocks less correlated

IMPLEMENT VIA PARAMETER SERVER

- Communication API:
- Range push and pull
- ✦ TASK: a pull or push or user-defined function
- One iteration (can contain push and pull)
- ✦ Task dependency graph:



IMPLEMENT BOUNDED-DELAY

- ✦ LinearBlockIterator::run()
- ★ executed at the scheduler

```
for (int iter = 0; iter < cf.max_pass_of_data(); ++iter) {
  std::random_shuffle(block_order.begin(), block_order.end());
  for (int b : block_order)  {
    Task update;
    update.set_wait_time(time - tau);
    auto cmd = RiskMinimization::setCall(&update);
    cmd->set_cmd(RiskMinCall::UPDATE_MODEL);
    // set the feature key range will be updated in this block
    blocks[b].second.to(cmd->mutable_key());
    time = pool->submit(update);
  }
```

EXPERIMENTS

SPARSE LOGISTIC REGRESSION

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

Training

examples	170 B
features	65 B
raw text data	636 T
machines	1,000
cores	16,000

Comparison

	Algorithm	Model	LOC
System-A	L-BFGS	Sequential	10K
System-B	Block CD	Sequential	30K
Parameter Server	Block CD	Bounded Delay + KKT	300

CONVERGE SPEED

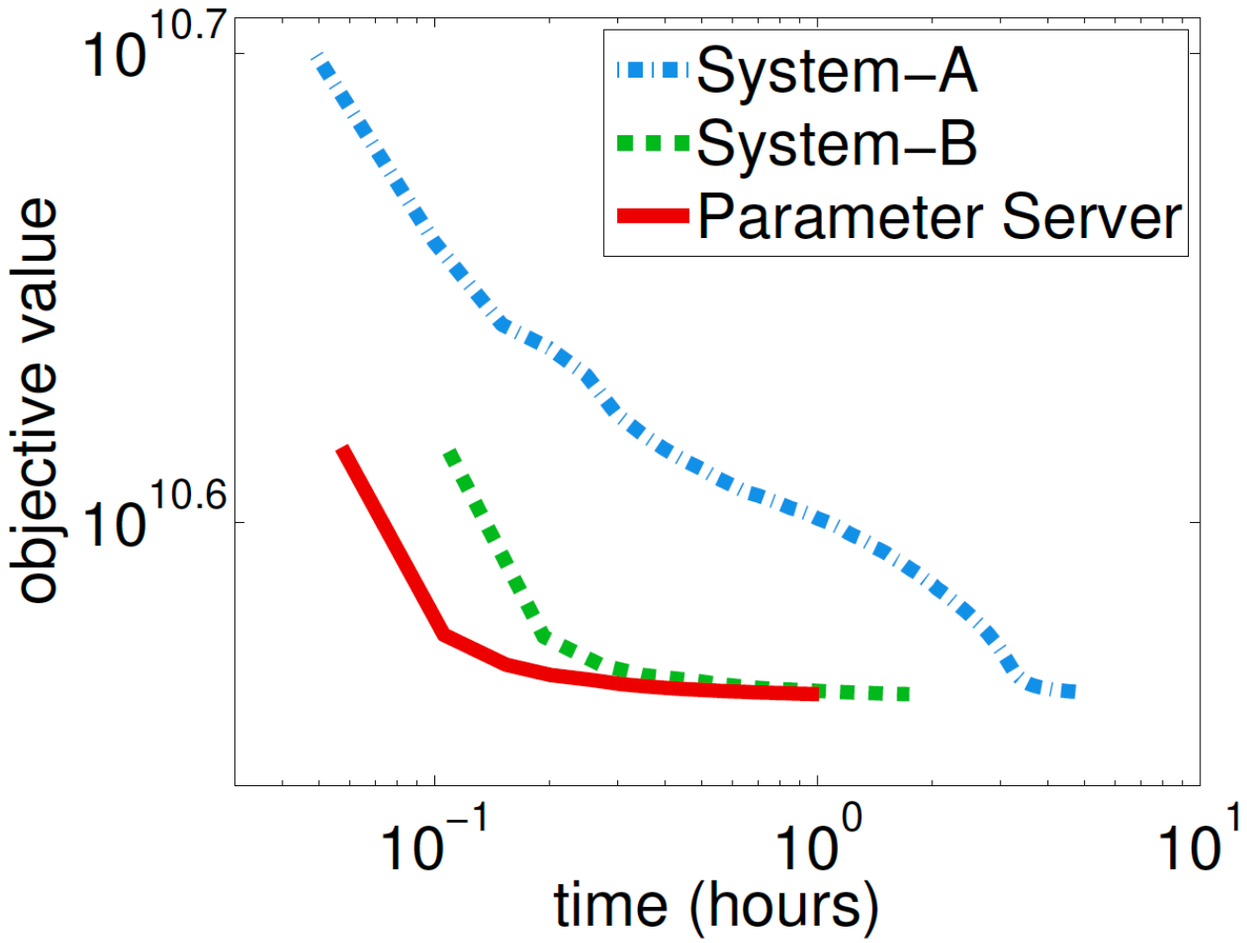
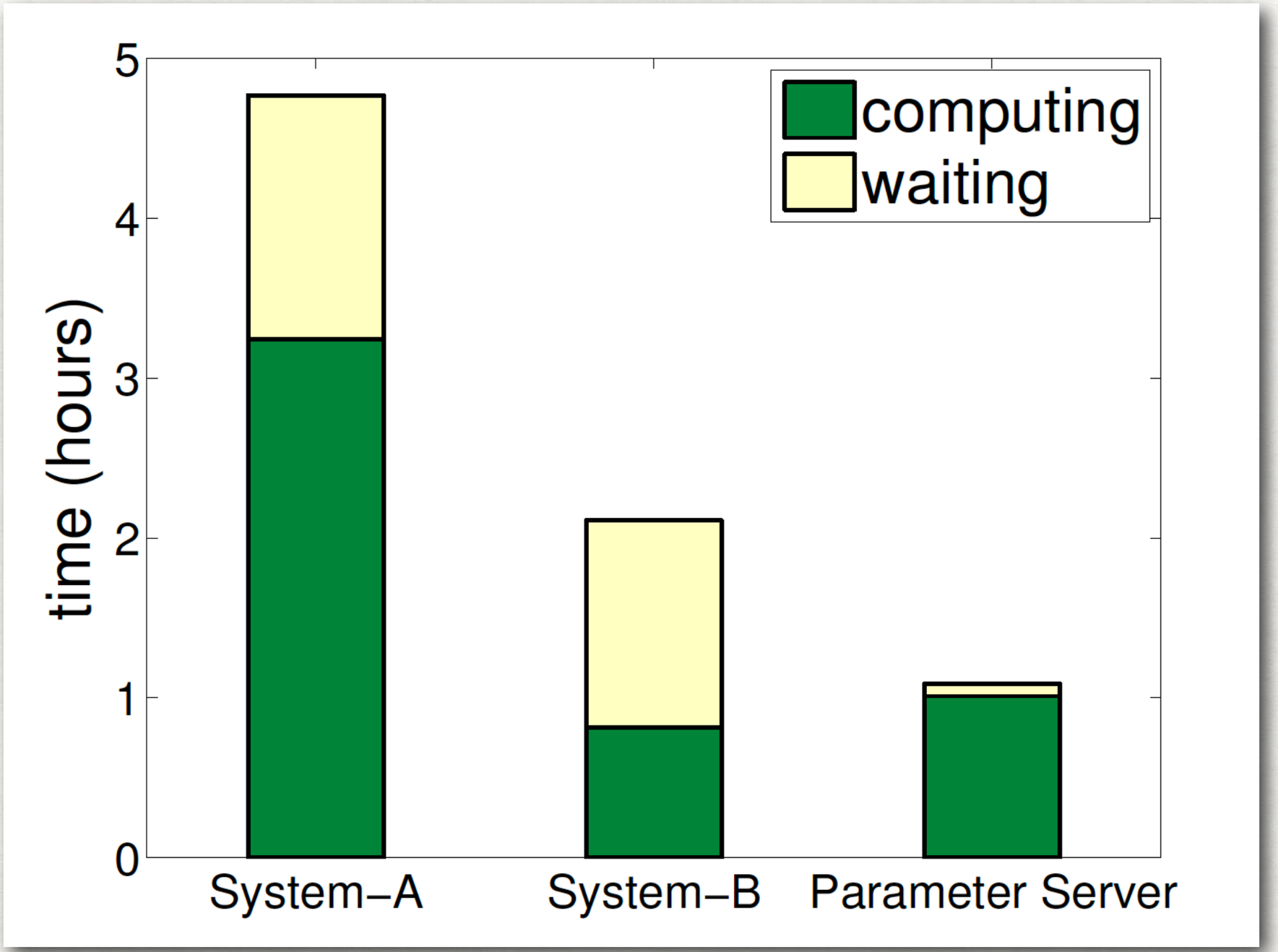


Figure 9: Convergence of sparse logistic regression. The goal is to minimize the objective rapidly.

- 5000TB CTR data
- 100B Variables
- 1000 Machines

- System A, B are on very large internet company

SCHEDULING EFFICIENCY



DEMO

https://github.com/mli/parameter_server

```
mkdir your_working_dir
cd your_working_dir
git clone git@github.com:mli/parameter_server.git .
git clone git@github.com:mli/parameter_server_third_party.git third_party
git clone git@github.com:mli/parameter_server_data.git data
third_party/install.sh
cd src && make -j8
mpirun -np 4 ./ps_mpi -num_servers 1 -num_workers 2 -app ../config/rcv1_l1lr.config
```

BOUNDED-DELAY

Sequential model

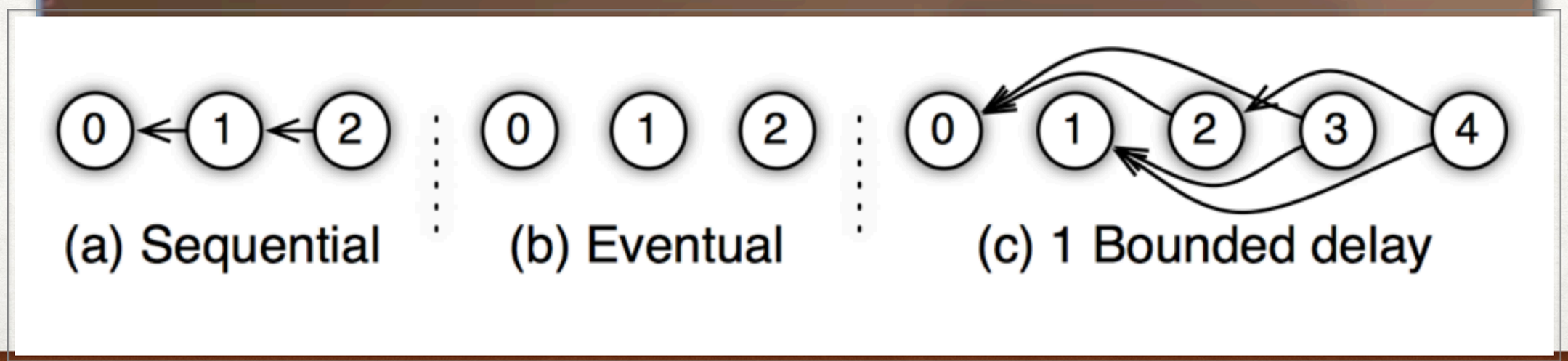
```
block_iterator {
# turn it off for debug use
random_feature_block_order : true

# block-coordinate updating. We divide a feature group into feature_block_ra
# x nnz_feature_per_instance blocks. If = 0, then use all features
feature_block_ratio : 4

max_pass_of_data : 20

# bounded-delay consistency
max_block_delay : 0

# convergance critiria. stop if the relative objective <= epsilon
epsilon : 1e-3
}
```



BOUNDED-DELAY

Bounded-delay = 1

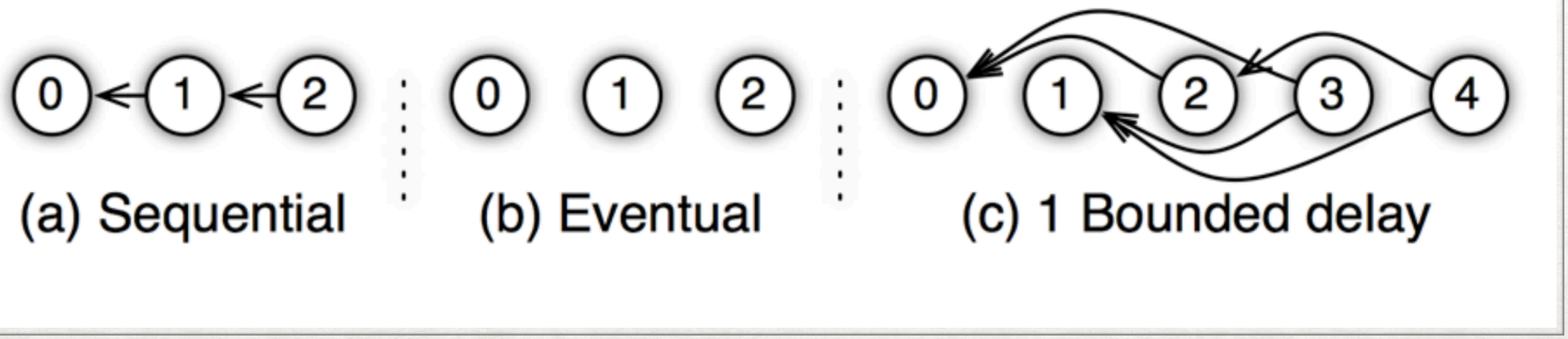
```
block_iterator {
# turn it off for debug use
random_feature_block_order : true

# block-coordinate updating. We divide a feature group into feature_block_ratio
# x nnz_feature_per_instance blocks. If = 0, then use all features
feature_block_ratio : 4

max_pass_of_data : 20

# bounded-delay consistency
max_block_delay : 1

# convergance critiria. stop if the relative objective <= epsilon
epsilon : 1e-3
}
```



BOUNDED-DELAY

Bounded-delay = 10

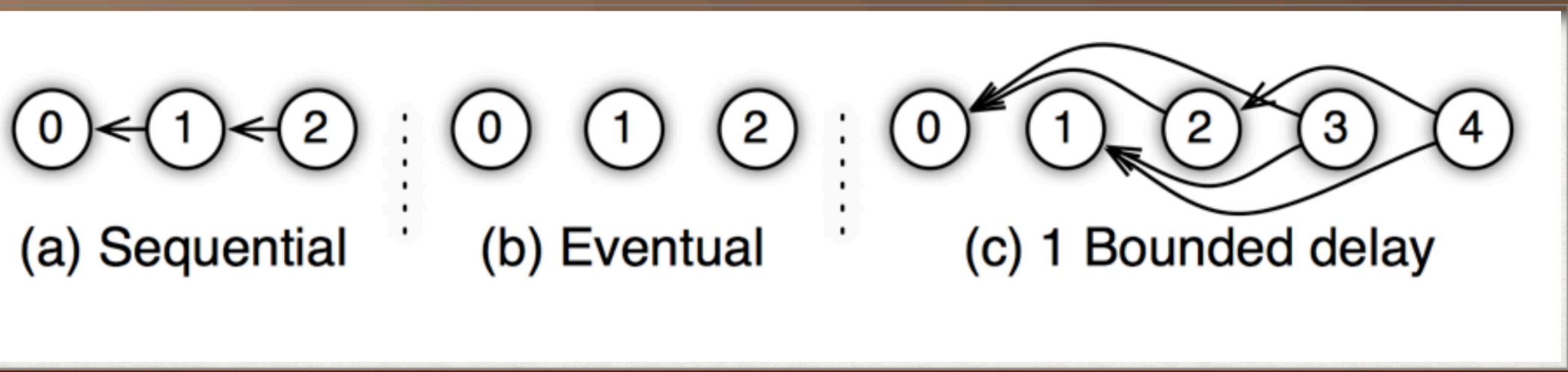
```
block_iterator {
# turn it off for debug use
random_feature_block_order : true

# block-coordinate updating. We divide a feature group into feature_block_ratio
# x nnz_feature_per_instance blocks. If = 0, then use all features
feature_block_ratio : 4

max_pass_of_data : 20

# bounded-delay consistency
max_block_delay : 10

# convergance critiria. stop if the relative objective <= epsilon
epsilon : 1e-3
}
```



BOUNDED-DELAY

Time difference

```
iter |  objective    relative |     |w|_0 | threshold  #activet |(app:min max) total
 ----+-------------------------+----------+---------------------+----------------
   0 | 6.20489e+05  1.000e+00 |   4595903 | 1.0e+20     6236013 |  10.7  12.9  20.1
   1 | 5.44760e+05  1.390e-01 |   3639910 | 1.4e-01     6236013 |   3.1   8.1  12.8
   2 | 5.12328e+05  6.330e-02 |   3270141 | 5.7e-06     3855248 |   2.2   8.5  12.1
   3 | 4.94491e+05  3.607e-02 |   2842948 | 2.2e-06     3354501 |  -0.0  10.8  12.2
[mpiexec@limucn11] Sending Ctrl-C to processes as requested
[mpiexec@limucn11] Press Ctrl-C again to force abort
/home/parameter_server/bin # mpirun -np 20 -hostfile /home/hosts ./ps_mpi -num_servers 2\
 -num_workers 8 -num_threads 8 -app ../config/ctr_l1lr.config -interface venet0:0
training data info: 4102531 examples with feature range [0,6236013)
system started...loaded data... in 4.631 sec
features are partitioned into 328 blocks
     |        training         |  sparsity |      KKT filter     |    time (sec.)
iter |  objective    relative |     |w|_0 | threshold  #activet |(app:min max) total
 ----+-------------------------+----------+---------------------+----------------
   0 | 6.23131e+05  1.000e+00 |   4581403 | 1.0e+20     6236013 |   9.5  10.3  14.2
   1 | 5.46933e+05  1.393e-01 |   3645061 | 1.7e-01     6236013 |   4.2   6.0   8.4
   2 | 5.13948e+05  6.418e-02 |   3279605 | 2.5e-06     3866977 |   4.0   6.8   9.1
   3 | 4.95783e+05  3.664e-02 |   2851889 | 4.6e-06     3368239 |   3.4   6.8   8.5
   4 | 4.84539e+05  2.320e-02 |   2575522 | 2.8e-06     2919564 |   3.5   7.1   8.5
```

BOUNDED-DELAY

Time difference

```
iter |  objective    relative |      |w|_0 | threshold  #activet |(app:min max) total
 ----+------------------------+----------+---------------------+----------------
   0 | 6.23131e+05  1.000e+00 |   4581403 | 1.0e+20     6236013 |   9.5  10.3  14.2
   1 | 5.46933e+05  1.393e-01 |   3645061 | 1.7e-01     6236013 |   4.2   6.0   8.4
   2 | 5.13948e+05  6.418e-02 |   3279605 | 2.5e-06     3866977 |   4.0   6.8   9.1
   3 | 4.95783e+05  3.664e-02 |   2851889 | 4.6e-06     3368239 |   3.4   6.8   8.5
   4 | 4.84539e+05  2.320e-02 |   2575522 | 2.8e-06     2919564 |   3.5   7.1   8.5
   5 | 4.76894e+05  1.603e-02 |   2369592 | 1.3e-06     2625512 |   2.0   7.3   7.7
[mpiexec@limucn11] Sending Ctrl-C to processes as requested
[mpiexec@limucn11] Press Ctrl-C again to force abort
/home/parameter_server/bin # mpirun -np 20 -hostfile /home/hosts ./ps_mpi -num_servers 2\
 -num_workers 8 -num_threads 8 -app ../config/ctr_l1lr.config -interface venet0:0
training data info: 4102531 examples with feature range [0,6236013)
system started...loaded data... in 4.746 sec
features are partitioned into 328 blocks
     |        training          |  sparsity |      KKT filter      |    time (sec.)
iter |  objective    relative |      |w|_0 | threshold  #activet |(app:min max) total
 ----+------------------------+----------+---------------------+----------------
   0 | 6.78717e+05  1.000e+00 |   4445893 | 1.0e+20     6236013 |   8.7  11.4  10.0
   1 | 5.88903e+05  1.525e-01 |   3751689 | 3.8e-01     6236013 |   3.9   8.2   5.2
   2 | 5.48595e+05  7.347e-02 |   3449582 | 1.4e-05     4008882 |   2.6   9.4   5.1
   3 | 5.40386e+05  1.519e-02 |   3132958 | 4.8e-06     3628338 |   0.9   9.3   5.0
```

BOUNDED-DELAY

同样 bound delay =1

features are partitioned into 328 blocks

iter	training objective	training relative	sparsity |w|_0	KKT filter threshold	KKT filter #activet	time (sec.) (app:min	time (sec.) max	time (sec.) total
0	6.23131e+05	1.000e+00	4581403	1.0e+20	6236013	9.5	10.3	14.2
1	5.46933e+05	1.393e-01	3645061	1.7e-01	6236013	4.2	6.0	8.4
2	5.13948e+05	6.418e-02	3279605	2.5e-06	3866977	4.0	6.8	9.1
3	4.95783e+05	3.664e-02	2851889	4.6e-06	3368239	3.4	6.8	8.5
4	4.84539e+05	2.320e-02	2575522	2.8e-06	2919564	3.5	7.1	8.5

features are partitioned into 117 blocks

iter	training objective	training relative	sparsity |w|_0	KKT filter threshold	KKT filter #activet	time (sec.) (app:min	time (sec.) max	time (sec.) total
0	6.44041e+05	1.000e+00	4530211	1.0e+20	6236013	9.2	10.2	12.9
1	5.67173e+05	1.355e-01	3661299	1.7e-01	6236013	4.4	5.5	7.0
2	5.29233e+05	7.169e-02	3359908	1.5e-05	3921960	3.9	6.8	7.2
3	5.09476e+05	3.878e-02	2956505	7.7e-06	3474482	2.5	5.8	7.1
4	4.96425e+05	2.629e-02	2686934	8.3e-06	3057918	1.9	6.8	7.4
5	4.87602e+05	1.809e-02	2478667	5.9e-06	2760234	1.8	9.2	6.4

SGD

Stochastic Gradient Descent OR mini-batch SGD

✦ The cost of an iteration is small

✦ Often converges faster than batch algorithm such as gradient descent

✦ A sequential algorithm

✦ Stochastic gradient descent is used more widely than coordinate descent

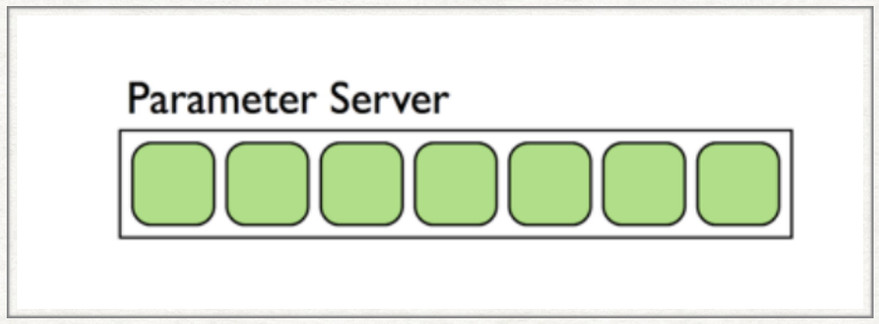
✦ Distributing SGD is more difficulty than CD

★ Samples are often more correlated comparing to features

PARALLEL MODELS

✦ Google brain

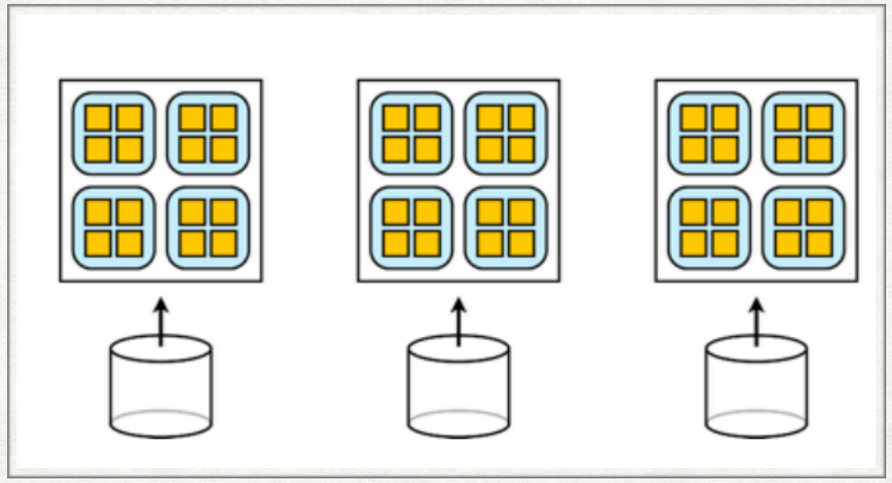
✦ Several worker groups update the parameter at the same time



✦ Good system performance

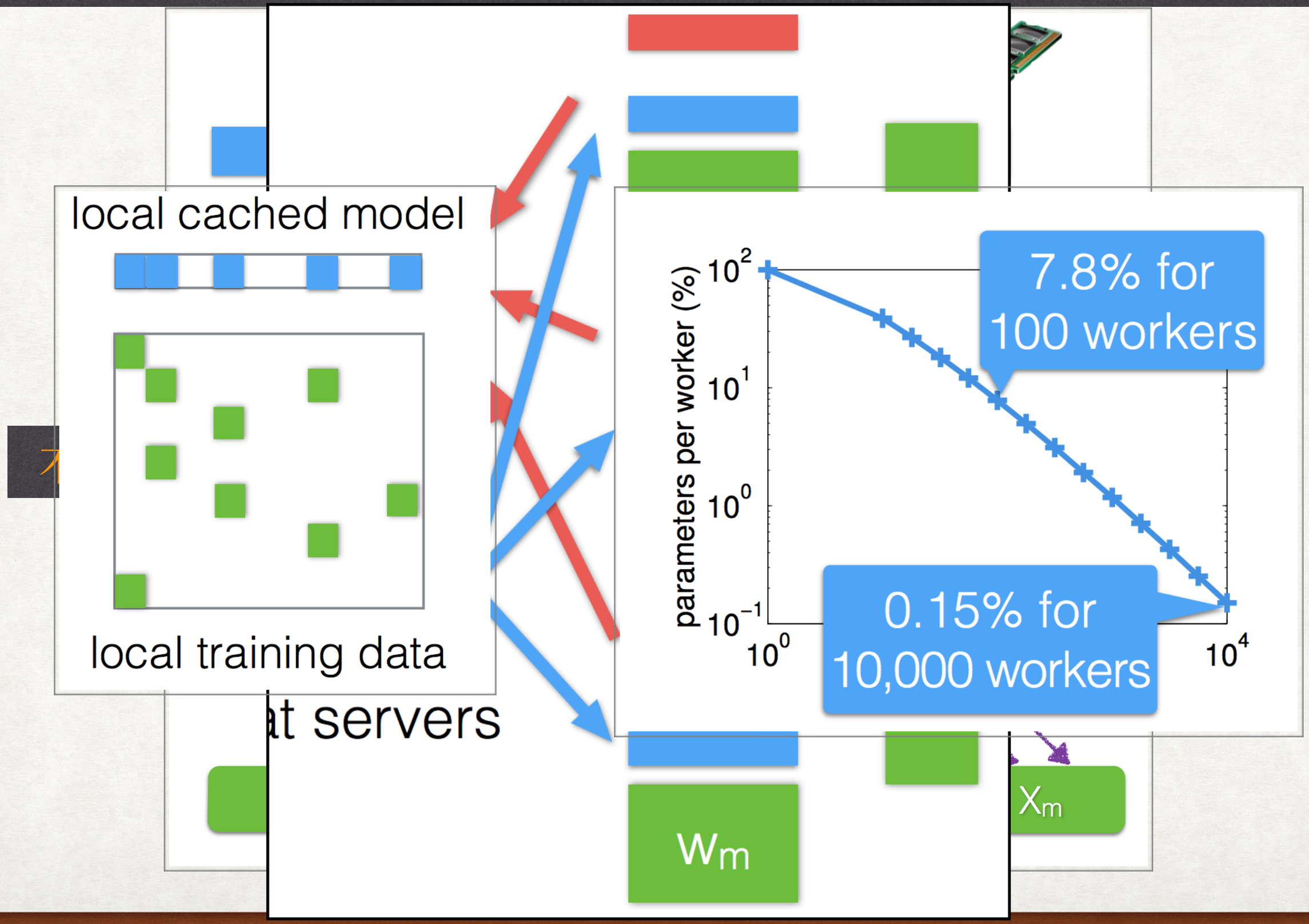
✦ May affect the convergence rate

★ 10x machine may only have 2x speedup



IMPLEMENTATION

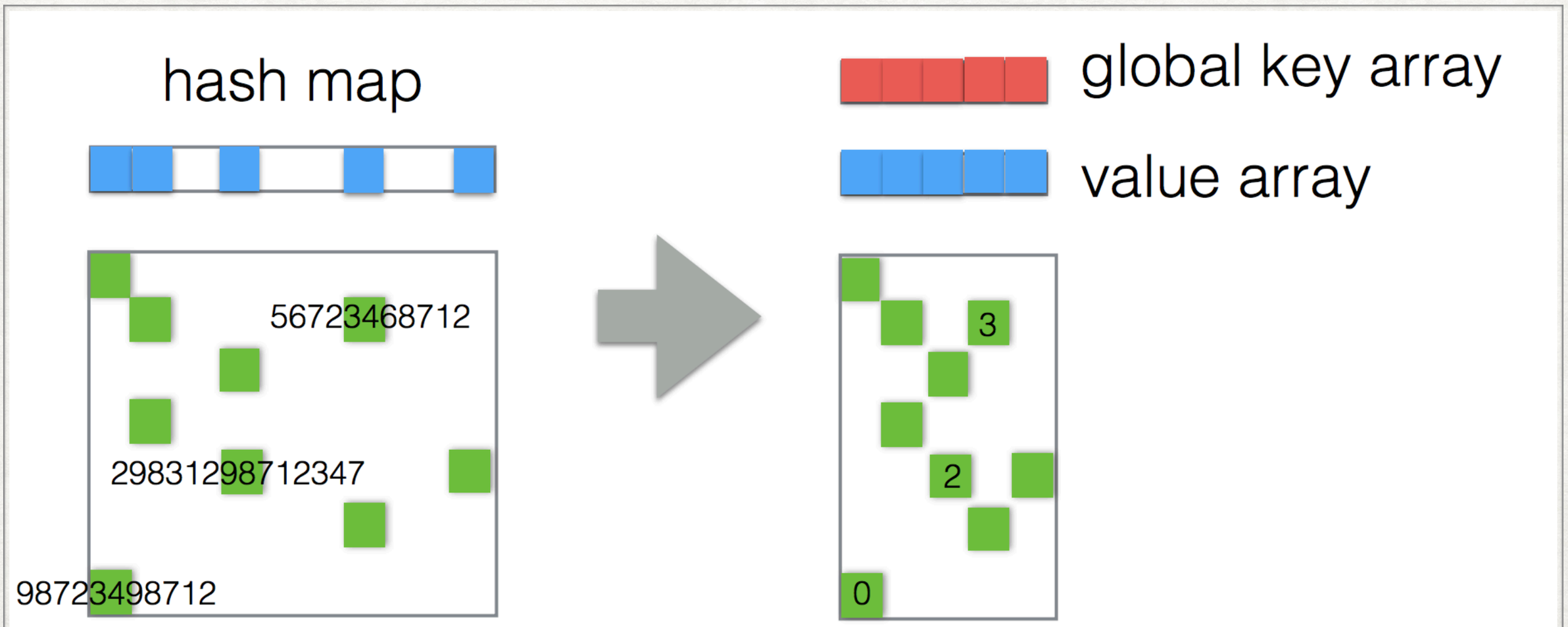
MODEL/DATA PARTITION



COMPACT REPRESENTATION

hash map->Localize Keys

Each worker machine maps global features id into local features id 0, 1, 2,



TERMINOLOGY

✦ Message: data for communication

✦ Van: send/receive message

✦ Customer: an application or a shared parameters

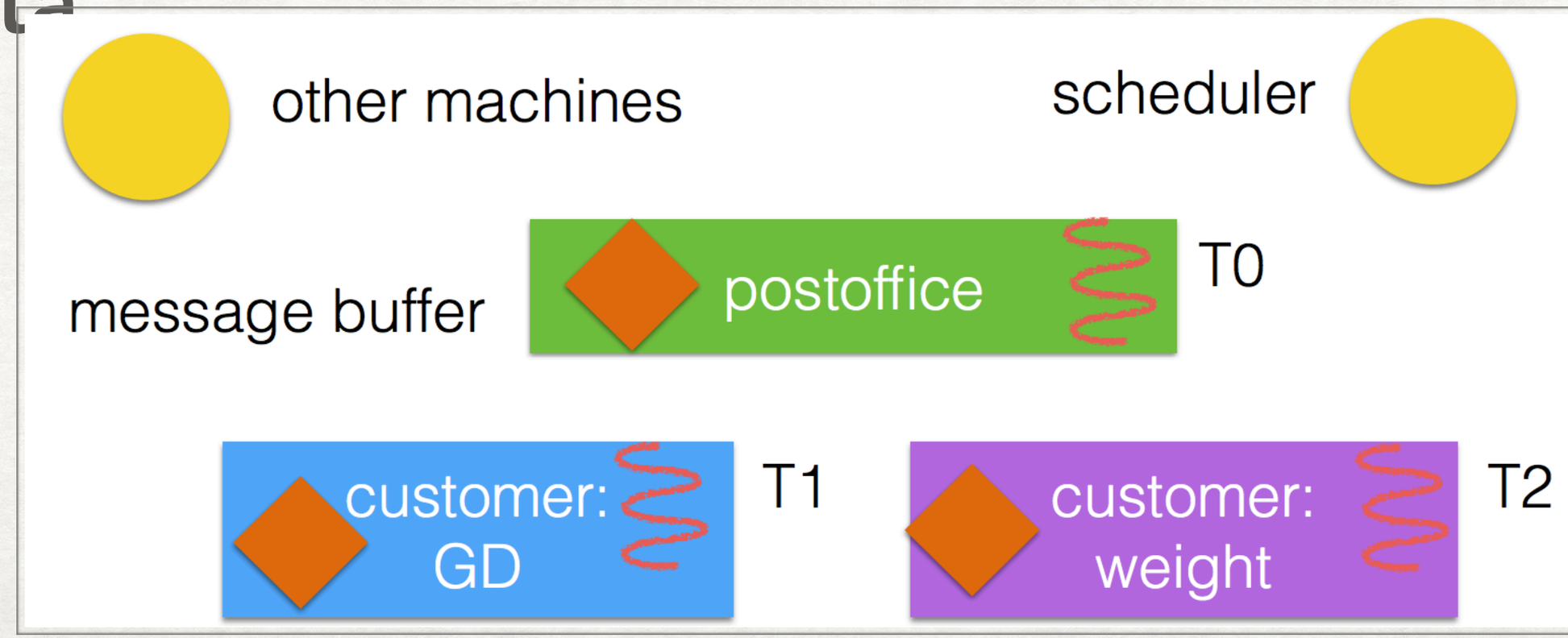
✦ Postoffice: allow customers to deliver messages and notify customers incoming messages

✦ Yellowpages:
all customers and live machines

TERMINOLOGY

```
> mpirun -hostfile ../../hosts -np 20 ./ps_mpi -int
erface eth0 -num_servers 4 -num_workers 12 -num_thre
ads 8 -app ../config/ctr_l1lr.config
```

data



CONCLUSTION

✦Support extremely large model

✦ Asynchronous

Flexible consistency model

✦ Use like writing single-thread matlab/numpy/Rlike codes

✦ A little bit complex to under the system

★ necessary to reach the best performance

LEVEL 3 PART