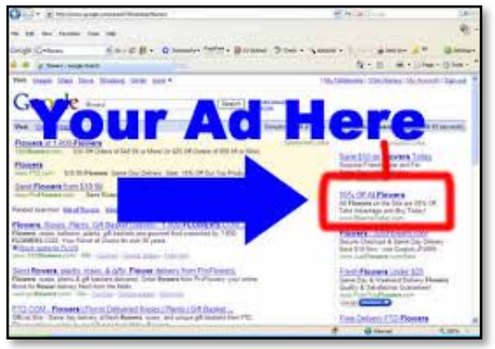
MALT: Distributed Data Parallelism for Existing ML Applications

Hao Li*, Asim Kadav, Erik Kruus, Cristian Ungureanu

*** University of Maryland-College Park**
NEC Laboratories, Princeton

Data, data everywhere…



	Software	User-generated content	Hardware
Data generated by	Transactions, website visits, other metadata	facebook, twitter, reviews, emails	Camera feeds, Sensors
Applications (usually based on ML)	Ad-click/Fraud prediction, Recommendations	Sentiment analysis, Targeted advertising	Surveillance, Anomaly detection

Timely insights depend on updated models

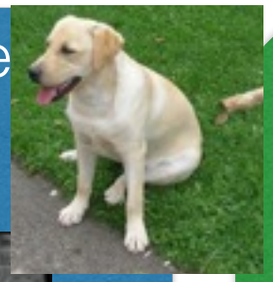
Training

Data (such as image, label pairs)

model parameters

Test/Deploy

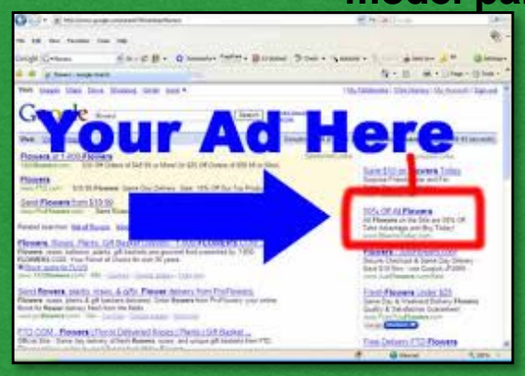
Surveillance/Safe Driving



- Usually trained in *real time*
- Expensive to train HD videos

Advertising (ad prediction, ...)

model parameters



- Usually trained *hourly*
- Expensive to train millions of requests

Kn... (auto... ...swering)

labrador

- Usually trained *daily*
- Expensive to train large corpus

Model training challenges

- Large amounts of data to train
  - Explosion in types, speed and scale of data
    - **Types** : Image, time-series, structured, sparse
    - **Speed** : Sensor, Feeds, Financial
    - **Scale** : Amount of data generated growing exponentially
  - Public datasets: Processed splice genomic dataset is 250 GB and data subsampling is unhelpful
  - Private datasets: Google, Baidu perform learning over TBs of data
- Model sizes can be huge
  - Models with billions of parameters do not fit in a single machine
  - E.g. : Image classification, Genome detection

Model accuracy generally improves by using larger models with more data

Properties of ML training workloads

- Fine-grained and Incremental:
  - Small, repeated updates to model vectors

- Asynchronous computation:
  - E.g. Model-model communication, back-propagation

- Approximate output:
  - Stochastic algorithms, exact/strong consistency maybe an overkill

- Need rich developer environment:
  - Require rich set of libraries, tools, graphing abilities

MALT: Machine Learning Toolset

- Run existing ML software in data-parallel fashion
- Efficient shared memory over RDMA writes to communicate model information
  - Communication: Asynchronously push (**`scatter`**) model information, **`gather`** locally arrived information
  - Network graph: Specify which replicas to send updates
  - Representation: **`SPARSE`**/**`DENSE`** hints to store model vectors
- MALT integrates with existing C++ and Lua applications
  - Demonstrate fault-tolerance and speedup with SVM, matrix factorization and neural networks
  - Re-uses existing developer tools

Outline

Introduction

Background

MALT Design

Evaluation

Conclusion

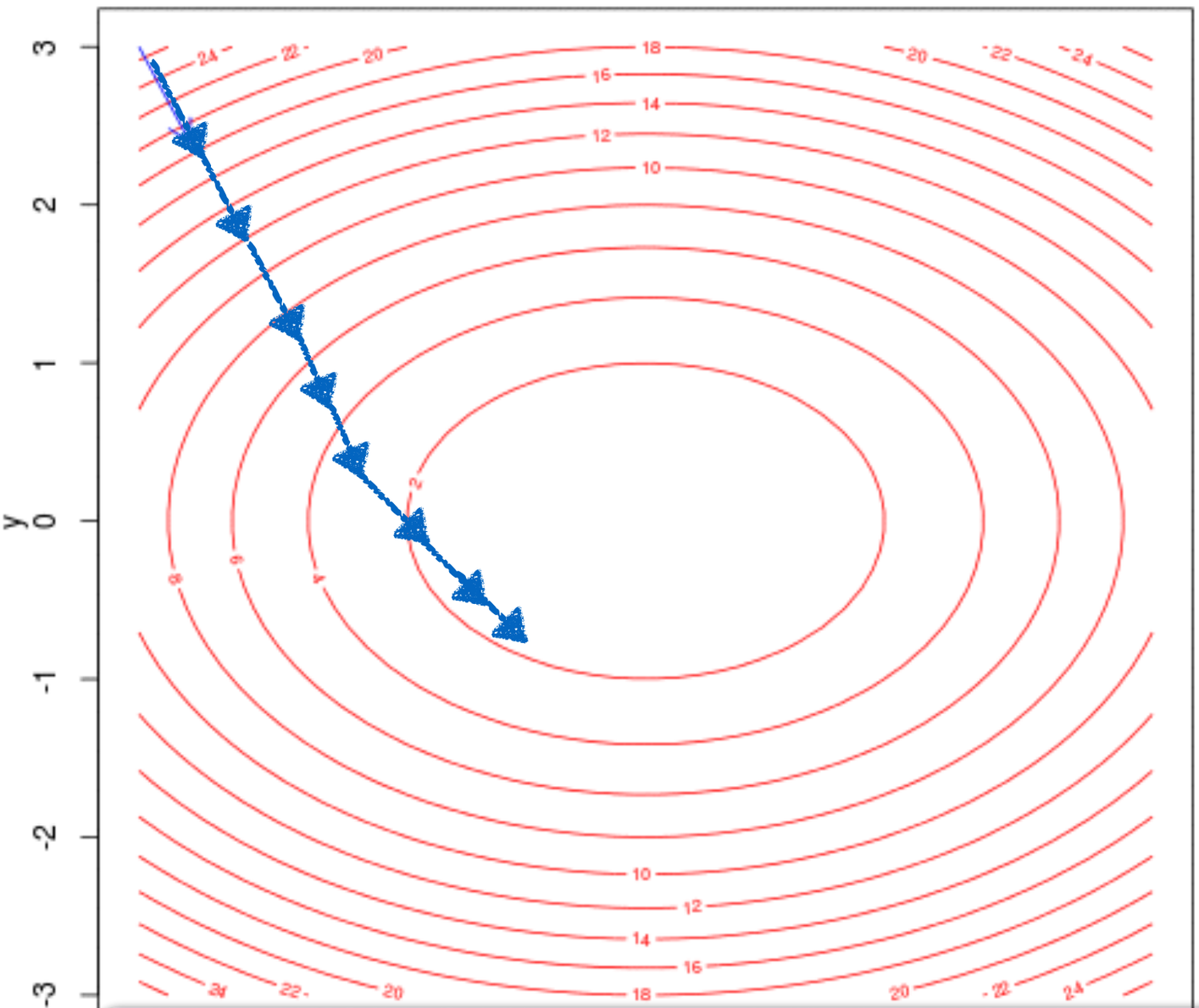
Distributed Machine Learning

- ML algorithms learn incrementally from data
  - Start with an initial guess of model parameters
  - Compute gradient over a loss fn, and update the model

- Data Parallelism: Train over large data
  - Data split over multiple machines
  - Model replicas train over different parts of data and communicate model information periodically

- Model parallelism: Train over large models
  - Models split over multiple machines
  - A single training iteration spans multiple machines

Stochastic Gradient Descent (SGD)



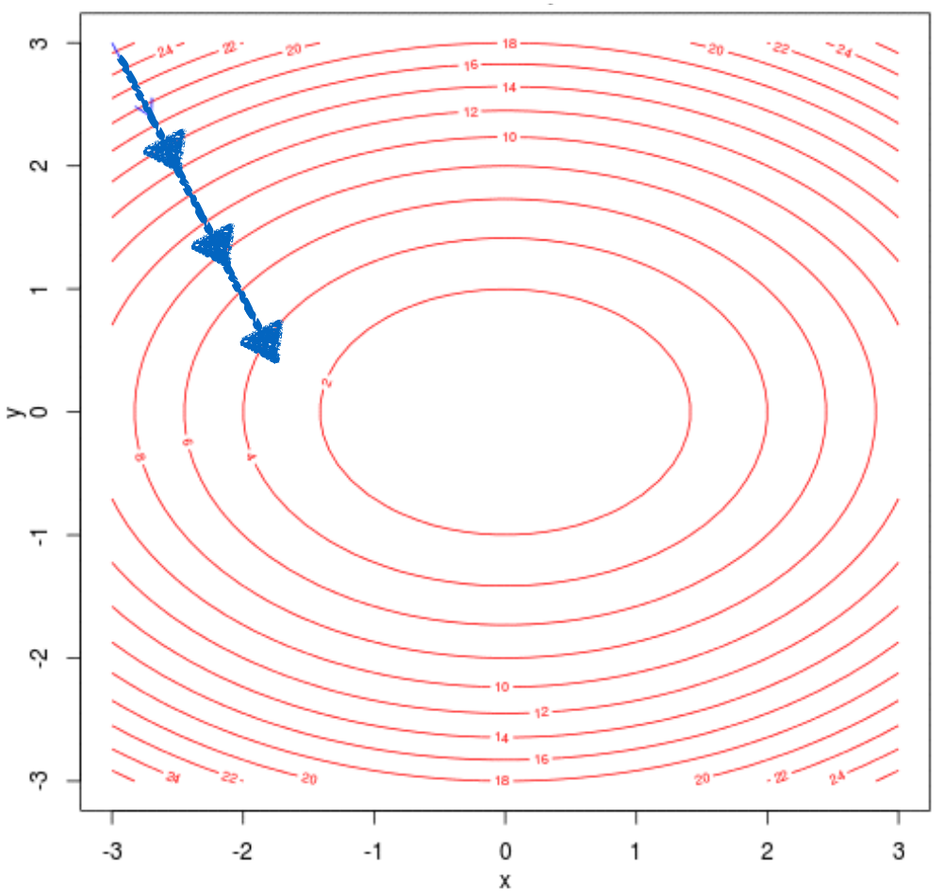
SGD trains over one (or few) training example at a time

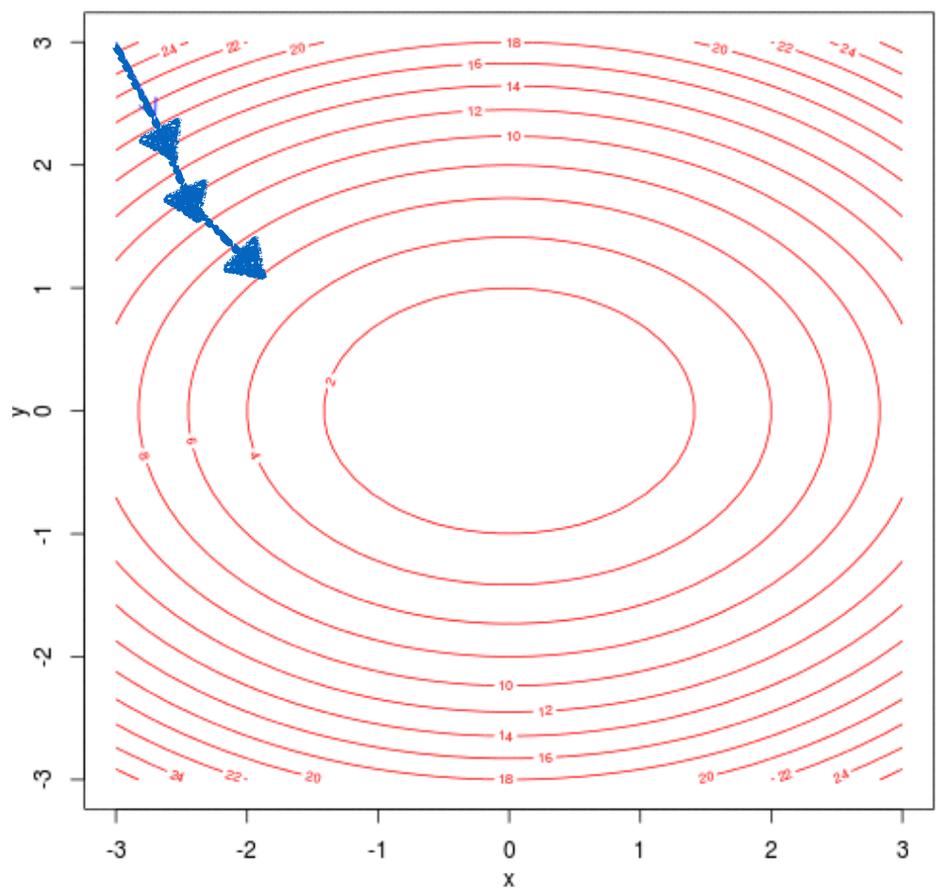
- Every data example processed is an *iteration*
- Update to the model is *gradient*
- Number of iterations to compute gradient is *batch size*
- One pass over the entire data is an *epoch*
- Acceptable performance over test set after multiple epochs is *convergence*

Can train wide range of ML methods : k-means, SVM, matrix factorization, neural-networks etc.

Data-parallel SGD: Mini-batching

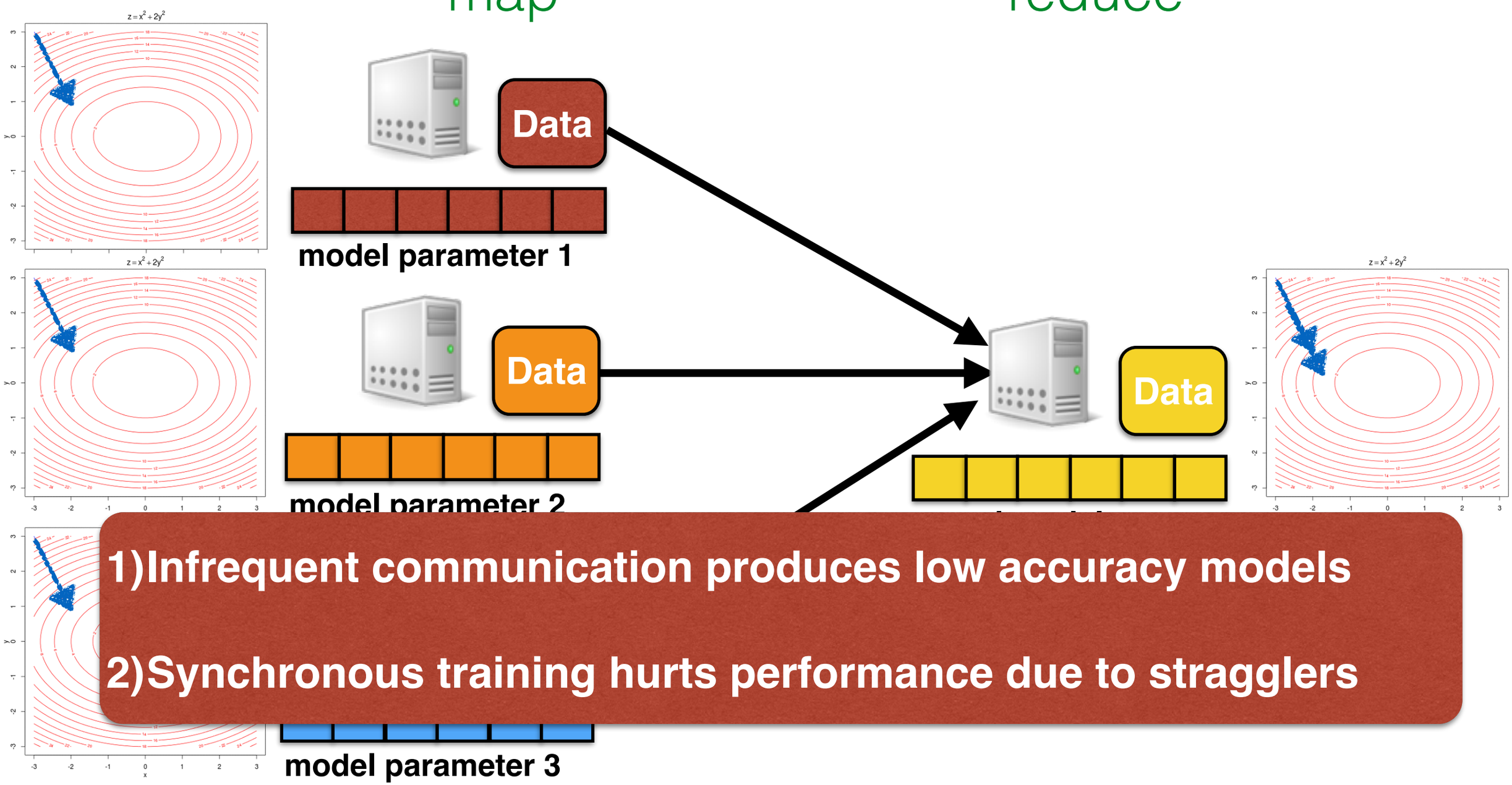
- Machines train in parallel over a *batch* and exchange model information
  - Iterate over data examples faster (in parallel)
  - May need more passes over data than single SGD (poor convergence)





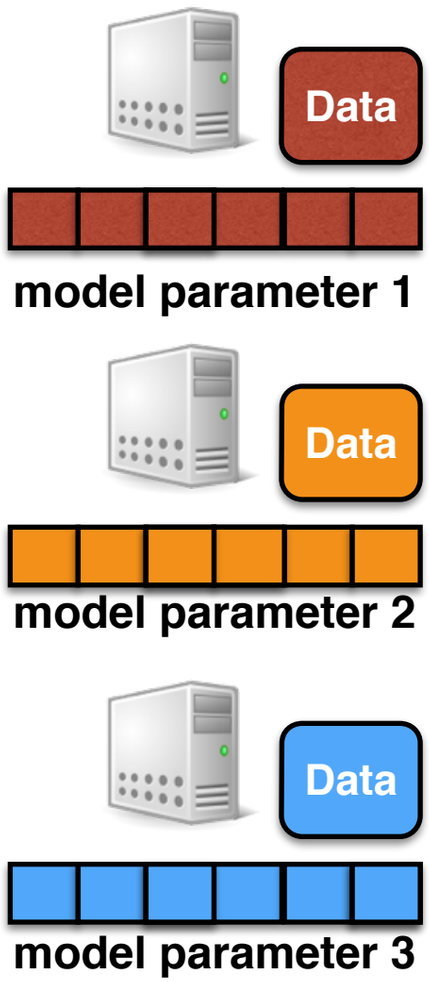
Approaches to data-parallel SGD

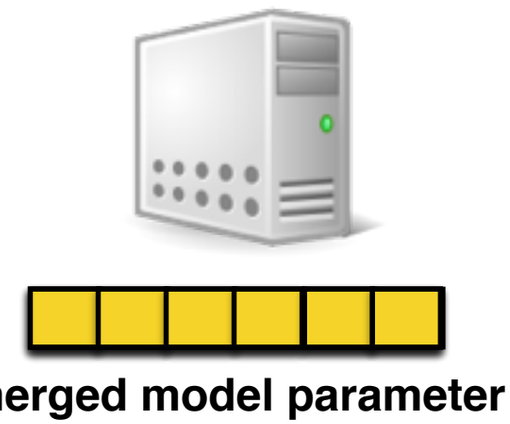
- Hadoop/Map-reduce: A variant of bulk-synchronous parallelism
  - Synchronous averaging of model updates every epoch (during reduce)



Parameter server

- Central server to merge updates every few iterations
  - Workers send updates asynchronously and receive whole models from the server
  - Central server merges incoming models and returns the latest model
  - Example: Distbelief (NIPS 2012), Parameter Server (OSDI 2014), Project Adam (OSDI 2014)



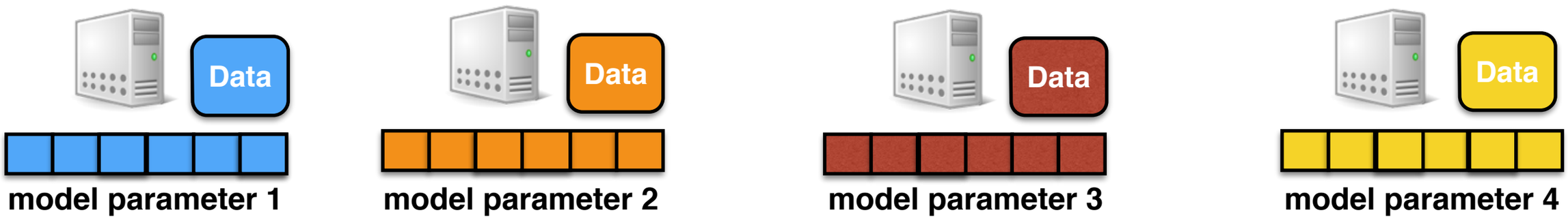


workers

server

Peer-to-peer approach (MALT)

- Workers send updates to one another asynchronously
  - Workers communicate every few iterations
  - No separate master/slave code to port applications)
  - No central server/manager: simpler fault tolerance



workers

Outline

Introduction

Background

MALT Design

Evaluation

Conclusion

MALT framework

Existing ML applications

Existing ML applications

Existing ML applications

MALT VOL
(Vector Object Library)

MALT VOL
(Vector Object Library)

MALT VOL
(Vector Object Library)

MALT dStorm
(distributed one-sided remote memory)

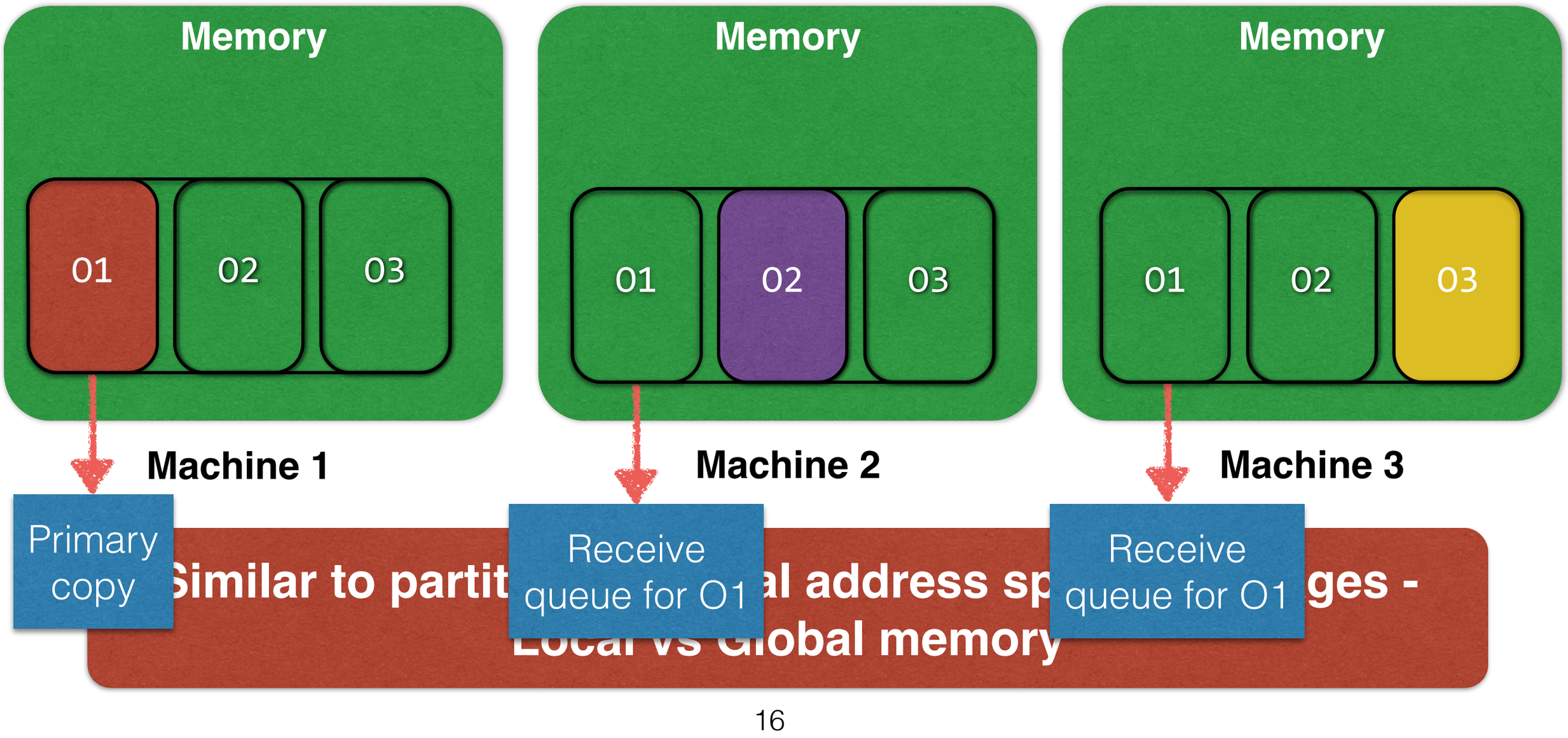
infiniBand communication substrate (such as MPI, GASPI)

Model replicas train in parallel. Use shared memory to communicate. Distributed file system to load datasets in parallel.

dStorm: Distributed one-sided remote memory

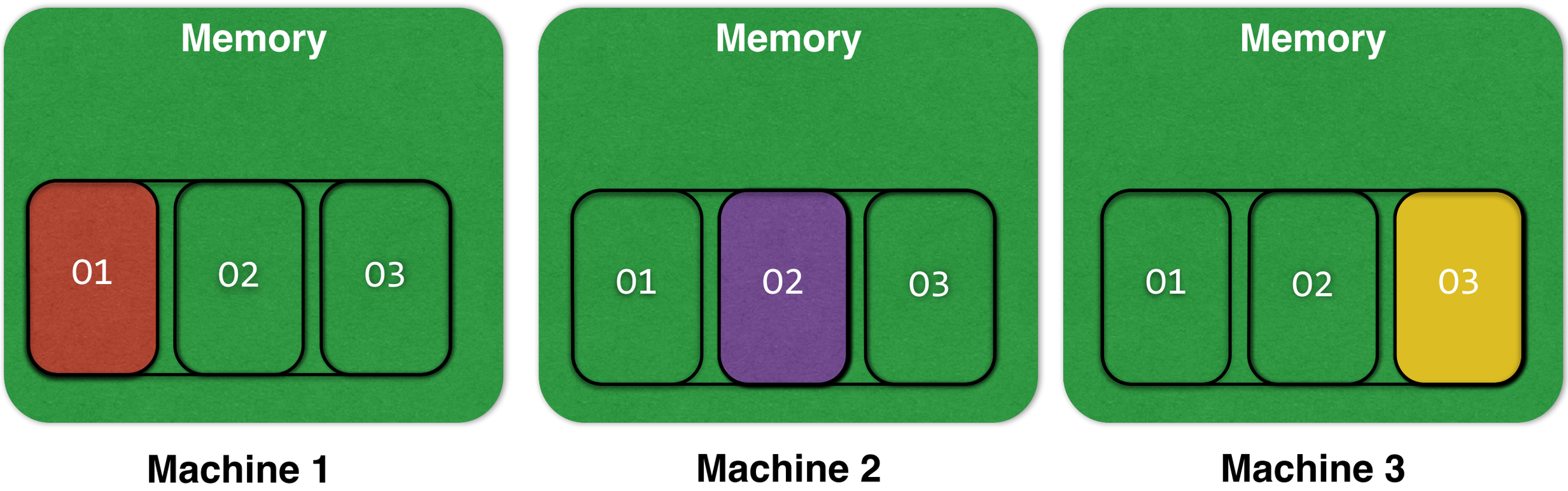
- RDMA over infiniBand allows high-throughput/low latency networking
  - RDMA over Converged Ethernet (RoCE) support for non-RDMA hardware
- Shared memory abstraction based over RDMA one-sided writes (no reads)

```
S1.create(size, ALL)   S2.create(size, ALL)   S3.create(size, ALL)
```



`scatter()` propagates using one-sided RDMA

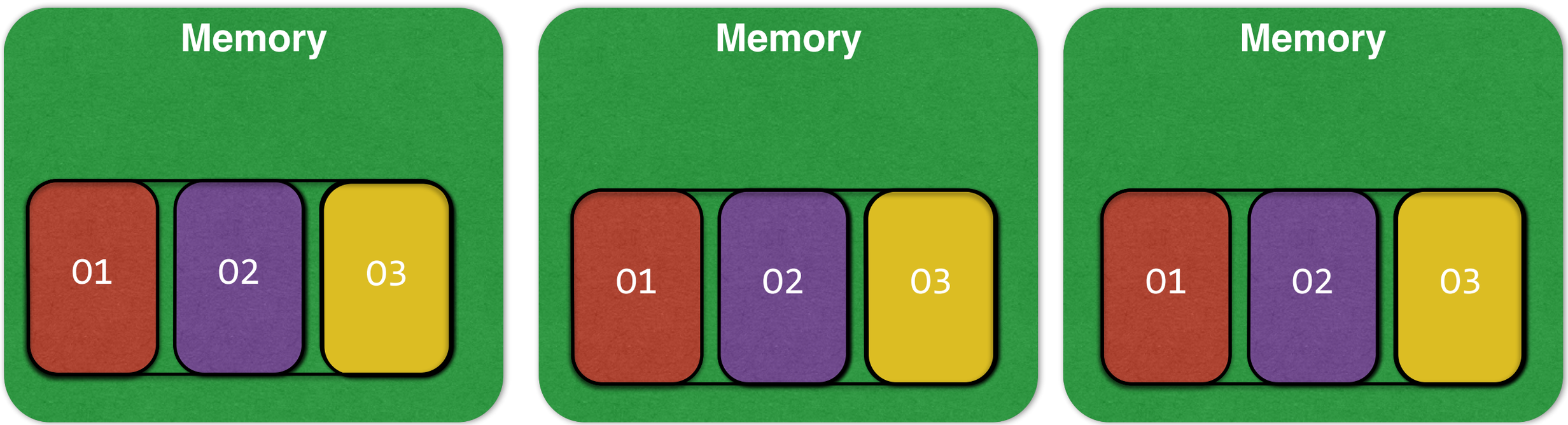
- Updates propagate based on communication graph



Remote CPU not involved: Writes over RDMA. Per-sender copies do not need to be immediately merged by receiver

`gather()` function merges locally

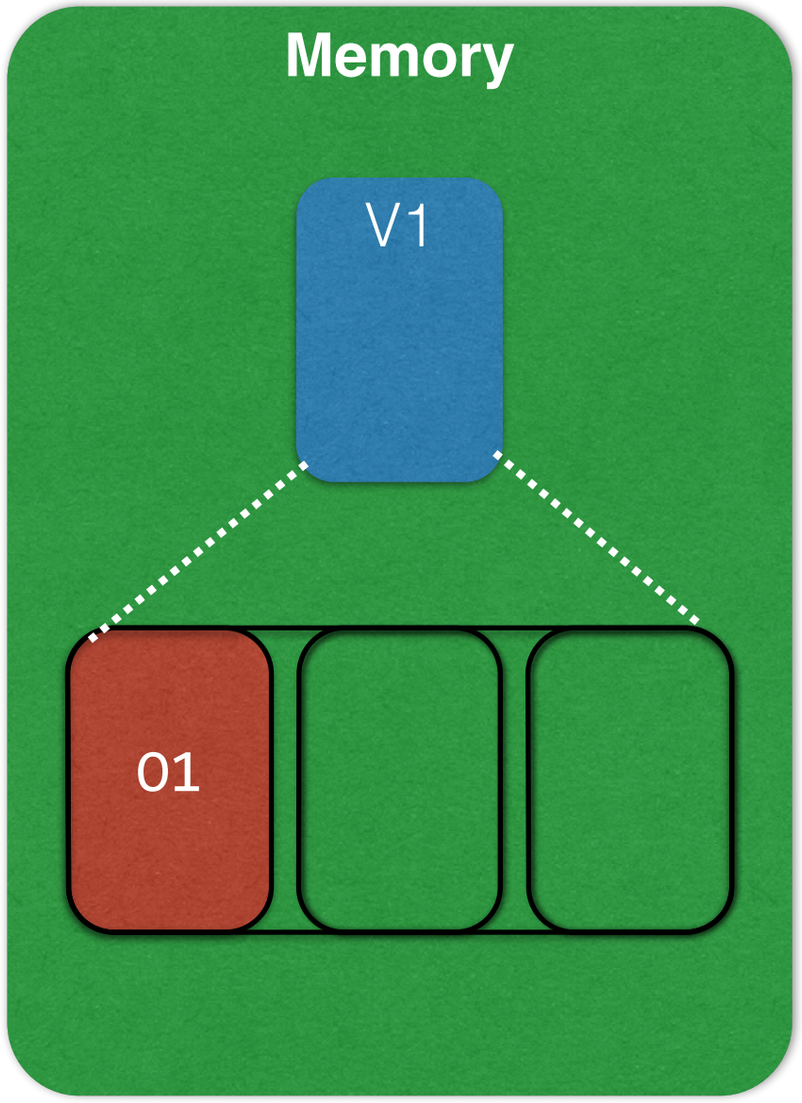
- Takes a user-defined function (UDF) as input such as average



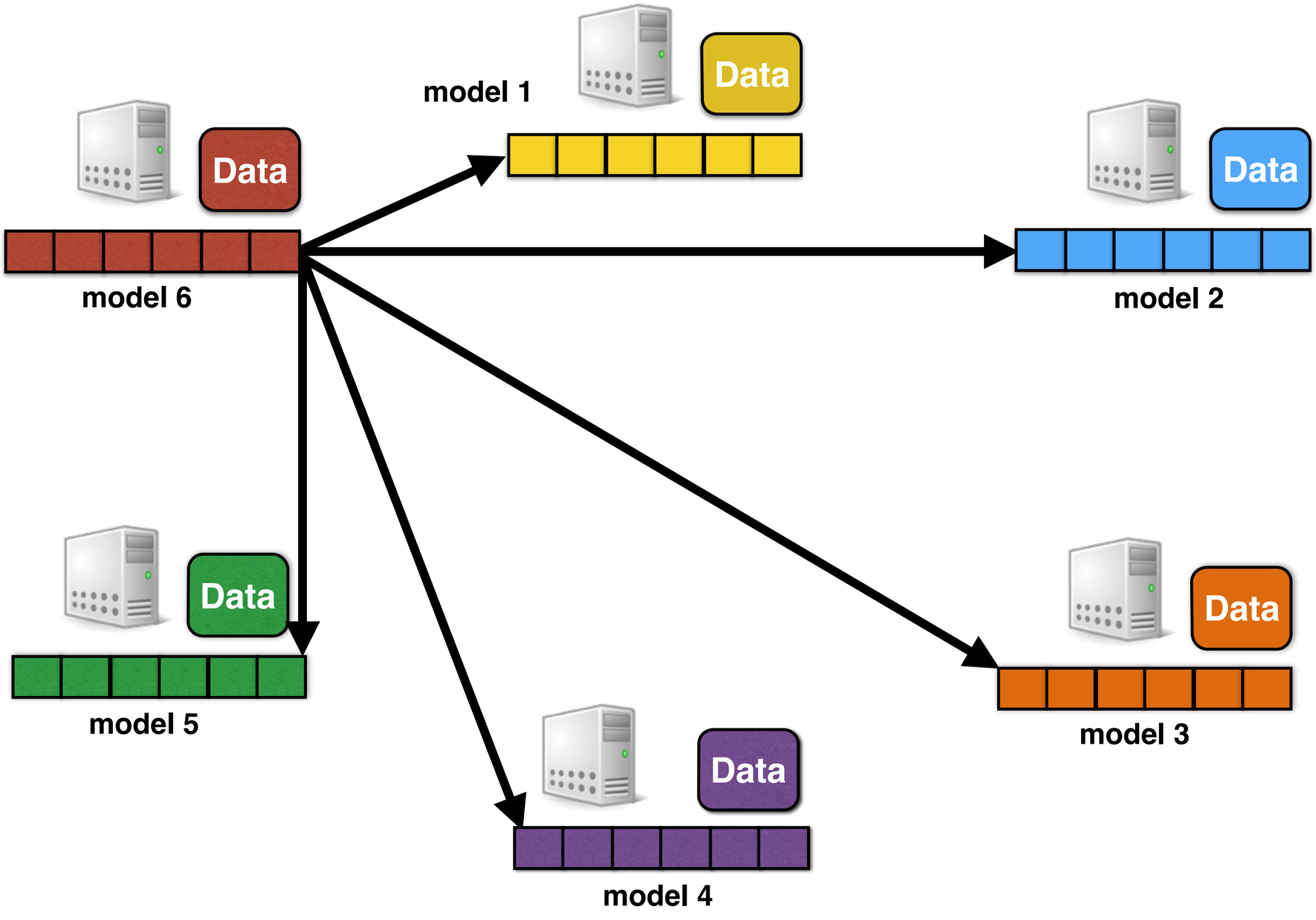
Useful/General abstraction for data-parallel algos: Train and `scatter()` the model vector, `gather()` received updates

VOL: Vector Object Library

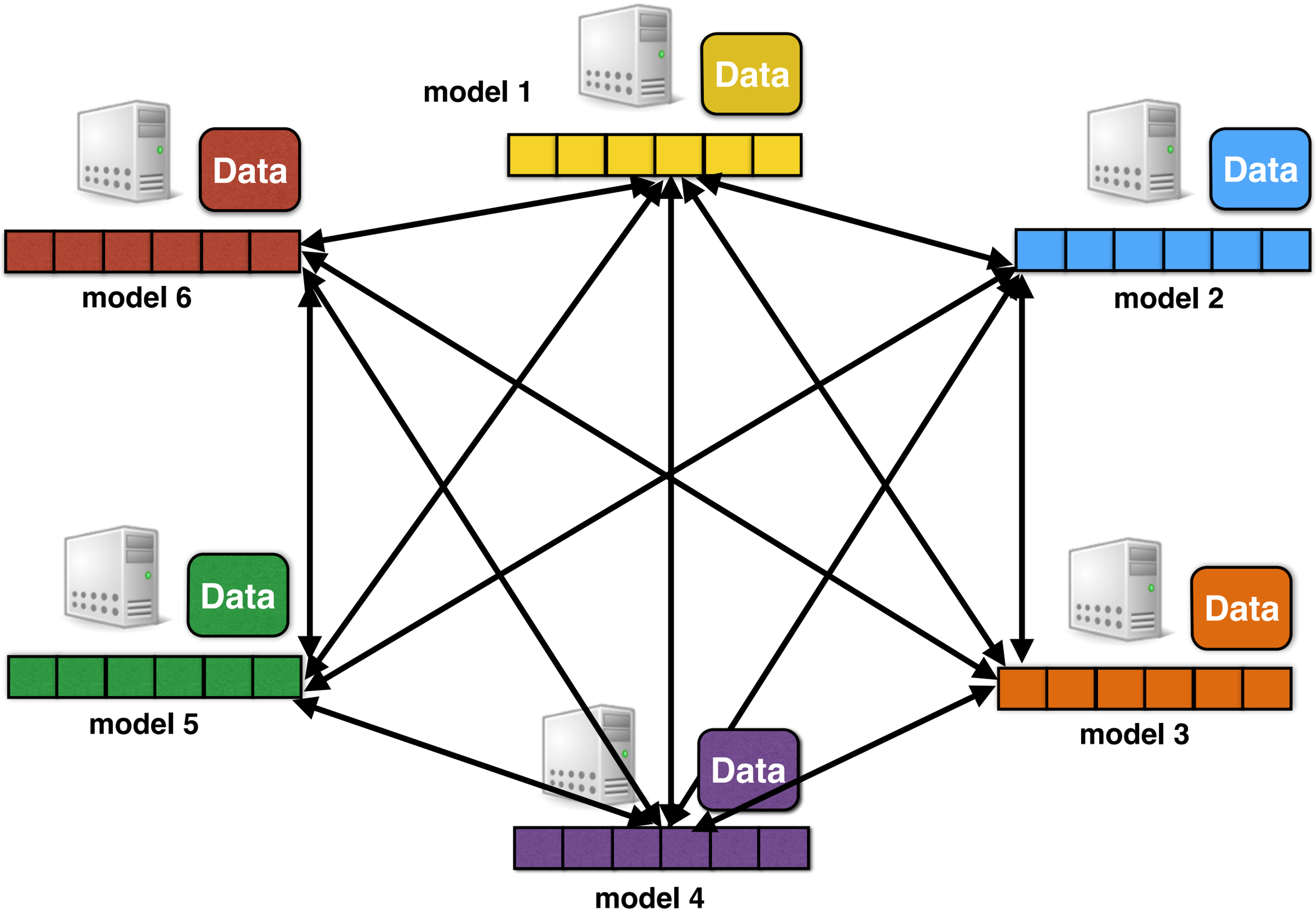
- Expose vectors/tensors instead of memory objects
- Provide representation optimizations
  - `sparse/dense` parameters store as arrays or key-value stores
- Inherits **`scatter()/gather()`** calls from dStorm
- Can use vectors/tensors in existing vectors



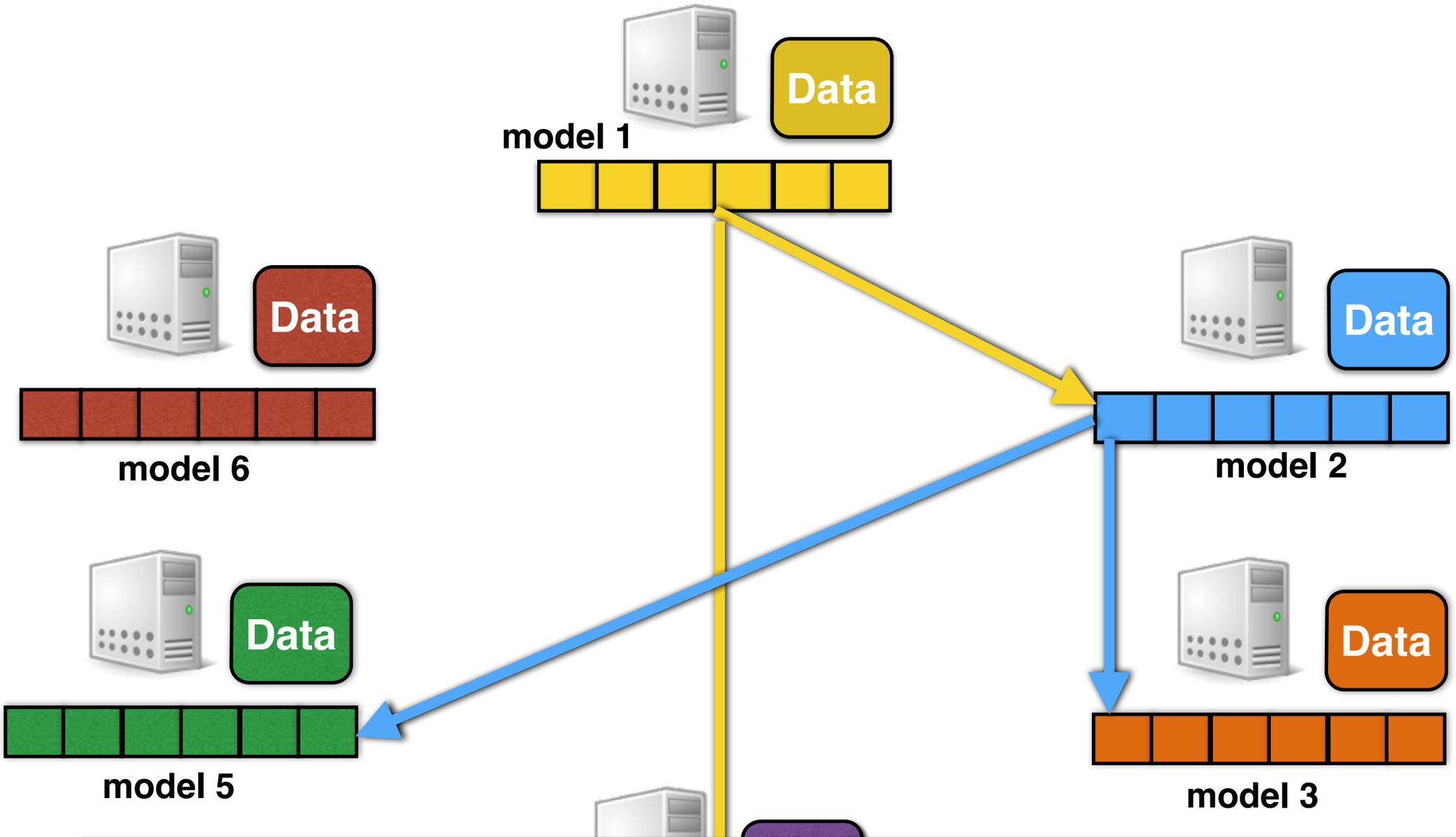
Propagating updates to everyone



$O(N^2)$ communication rounds for N nodes

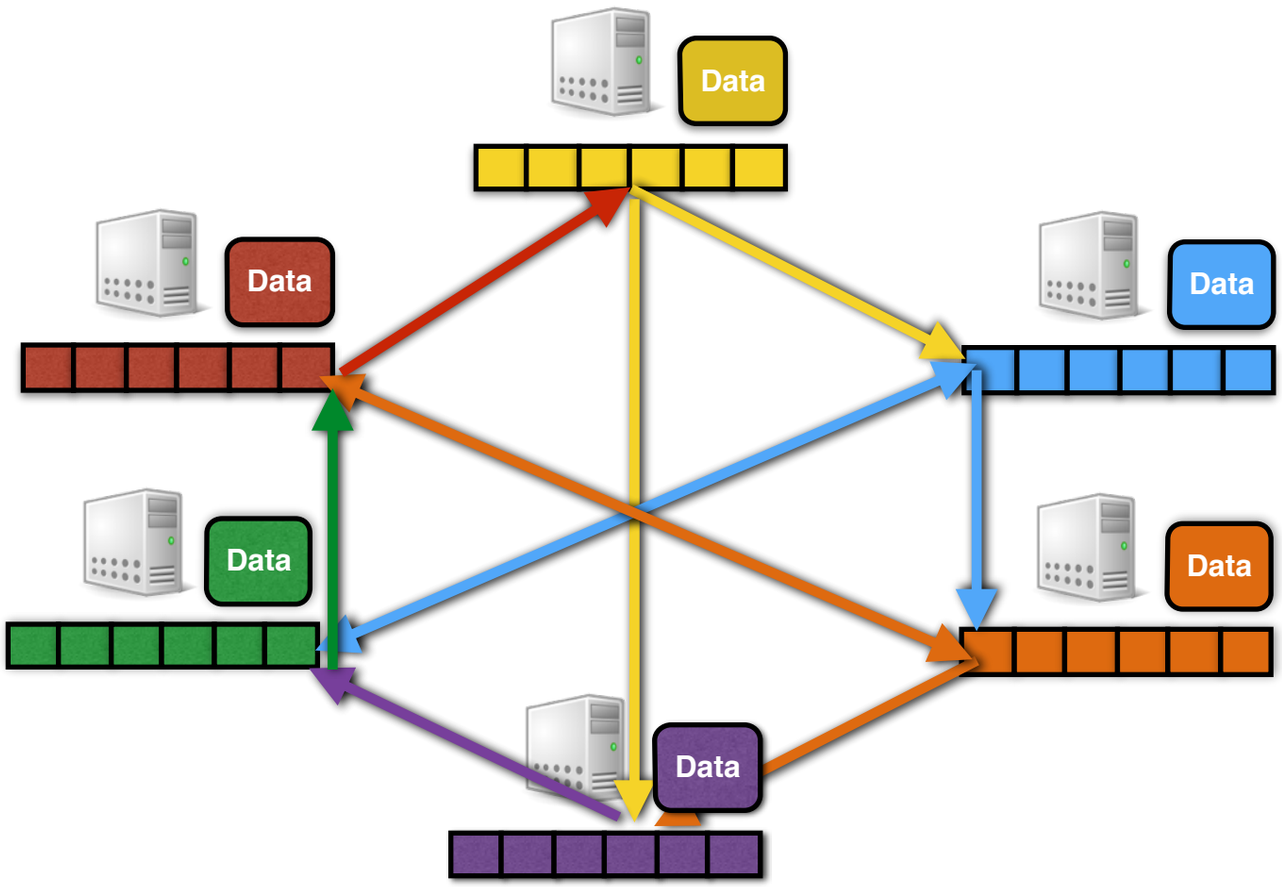


In-direct propagation of model updates



Use a uniform random sequence to determine where to send updates to ensure all updates propagate uniformly. Each node sends to fewer than N nodes (such as logN)

O(Nlog(N)) communication rounds for N nodes



Trade-off model information recency with savings in network and update processing time

- MALT proposes sending models to fewer nodes (log N instead of N)
  - Requires the node graph be connected
  - Use any uniform random sequence
- Reduces processing/network times
  - Network communication time reduces
  - Time to update the model reduces
  - Iteration speed increases but may need more epochs to converge
- Key Idea: Balance communication with computation
  - Send to less/more than log(N) nodes

Converting serial algorithms to parallel

Serial SGD

```
Gradient g;
Parameter w;
for epoch = 1:maxEpochs do
  for i = 1:N do
    g = cal_gradient(data[i]);
    w = w + g;
```

Data-Parallel SGD

```
maltGradient g(sparse, ALL);
Parameter w;
for epoch = 1:maxEpochs do
  for i = 1:N/ranks do
    g = cal_gradient(data[i])
    g.scatter(ALL);
    g.gather(AVG);
    w = w + g;
```

- **`scatter()`** performs one-sided RDMA writes to other machines.
- **`"ALL"`** signifies communication with *all* other machines.
- **`gather(AVG)`** applies *average* to the received gradients.
- Optional **`barrier()`** makes the training synchronous.

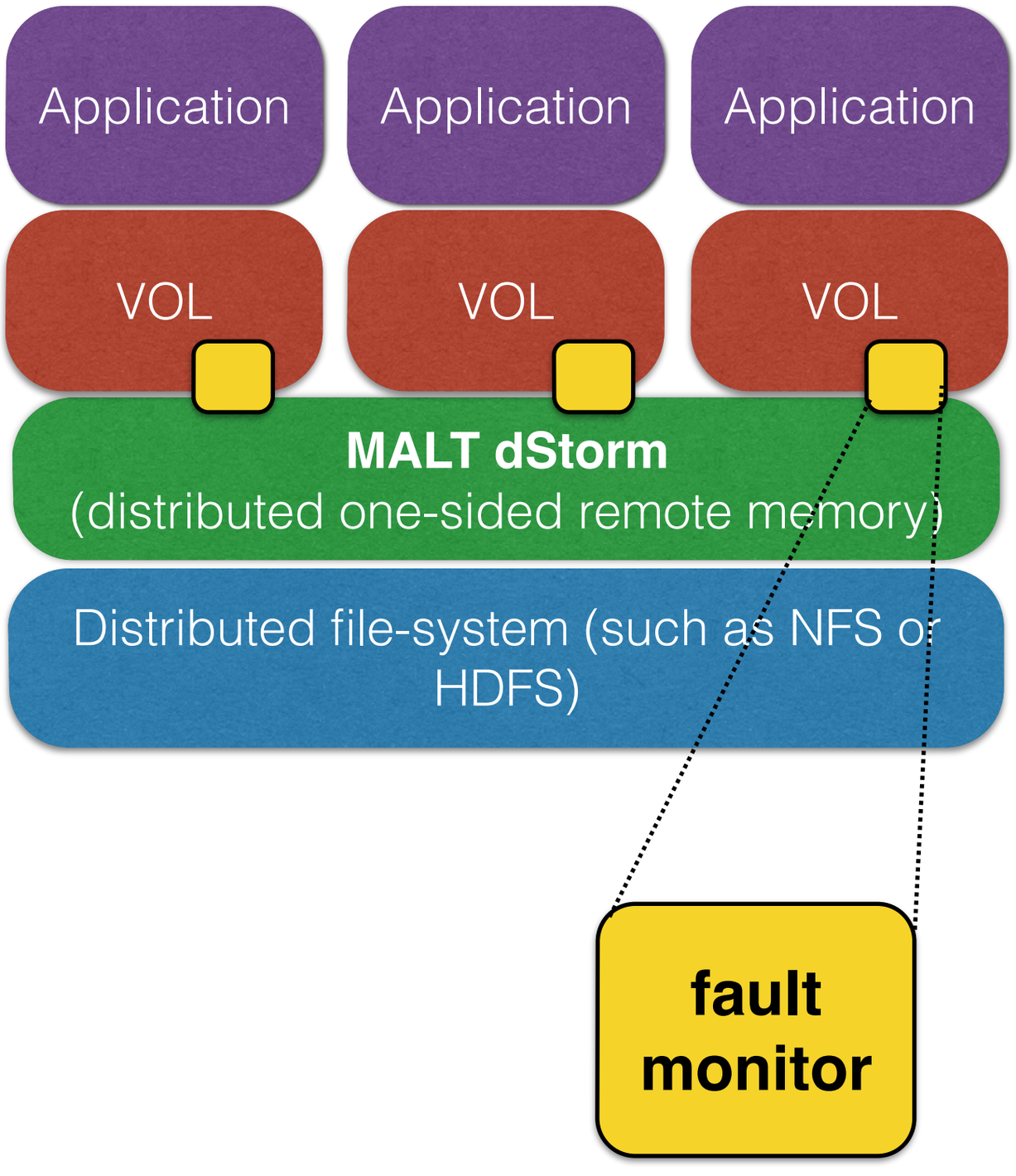
Consistency guarantees with MALT

- Problem: Asynchronous **`scatter/gather`** may cause models to diverge significantly

- Problem scenarios:
  - Torn reads: Model information may get re-written while being read
  - Stragglers: Slow machines send stale model updates
  - Missed updates: Sender may overwrite its queue if receiver is slow

- Solution: All incoming model updates carry iteration count in their header and trailer
    - Read header-trailer-header to skip torn reads
    - Slow down current process if the incoming updates are too stale to limit stragglers and missed updates (Bounded Staleness [ATC 2014])
- Few inconsistencies are OK for model training (Hogwild [NIPS 2011])
    - Use barrier to train in BSP fashion for stricter guarantees

MALT fault tolerance: Remove failed peers



- Each replica has a fault monitor
  - Detects local failures (processor exceptions such as divide-by-zero)
  - Detects failed remote writes (timeouts)
- When failure occurs
  - Locally: Terminate local training, other monitors see failed writes
  - Remote: Communicate with other monitors and create a new group
  - Survivor nodes: Re-register queues, re-assign data, and resume training
- Cannot detect byzantine failures (such as corrupt gradients)

Outline

Introduction

Background

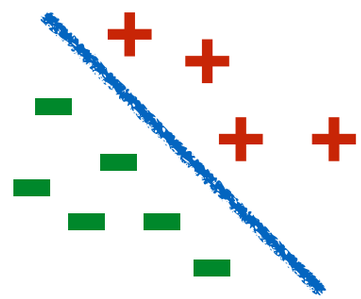
MALT Design

Evaluation

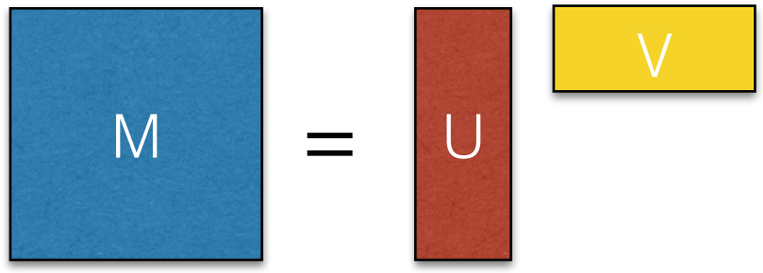
Conclusion

Integrating existing ML applications with MALT

- Support Vector Machines (SVM)
  - Application: Various classification applications
  - Existing application: Leon Bottou's SVM SGD
  - Datasets: RCV1, PASCAL suite, splice (700M - 250 GB size, 47K - 16.6M parameters)

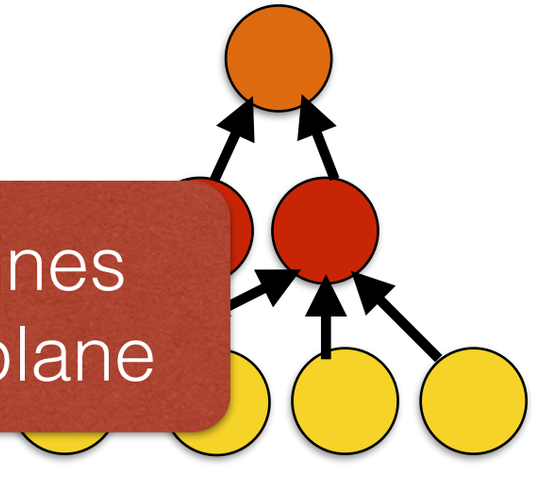


- Matrix Factorization (Hogwild)
  - Application: Movie recommendation (Netflix)
  - Existing application: HogWild (NIPS 2011)
  - Datasets: Netflix (1.6 G size, 14.9M parameters)

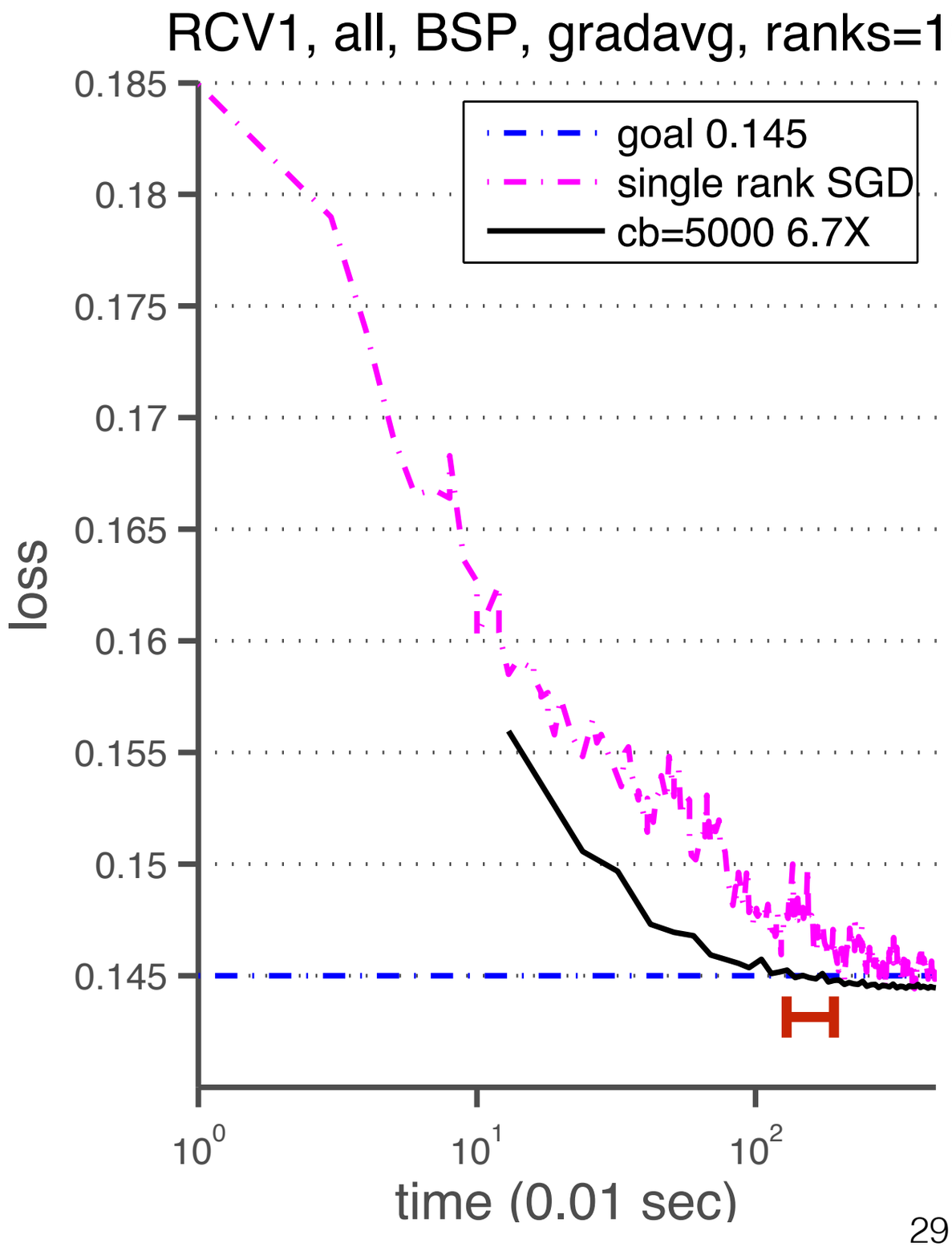


- Neural networks (NEC RAPID)
  - Application: Ad-click prediction (KDD 2012)
  -
  -

Cluster: Eight Intel 2.2 Ghz with 64 GB RAM machines connected with Mellanox 56 Gbps infiniband backplane



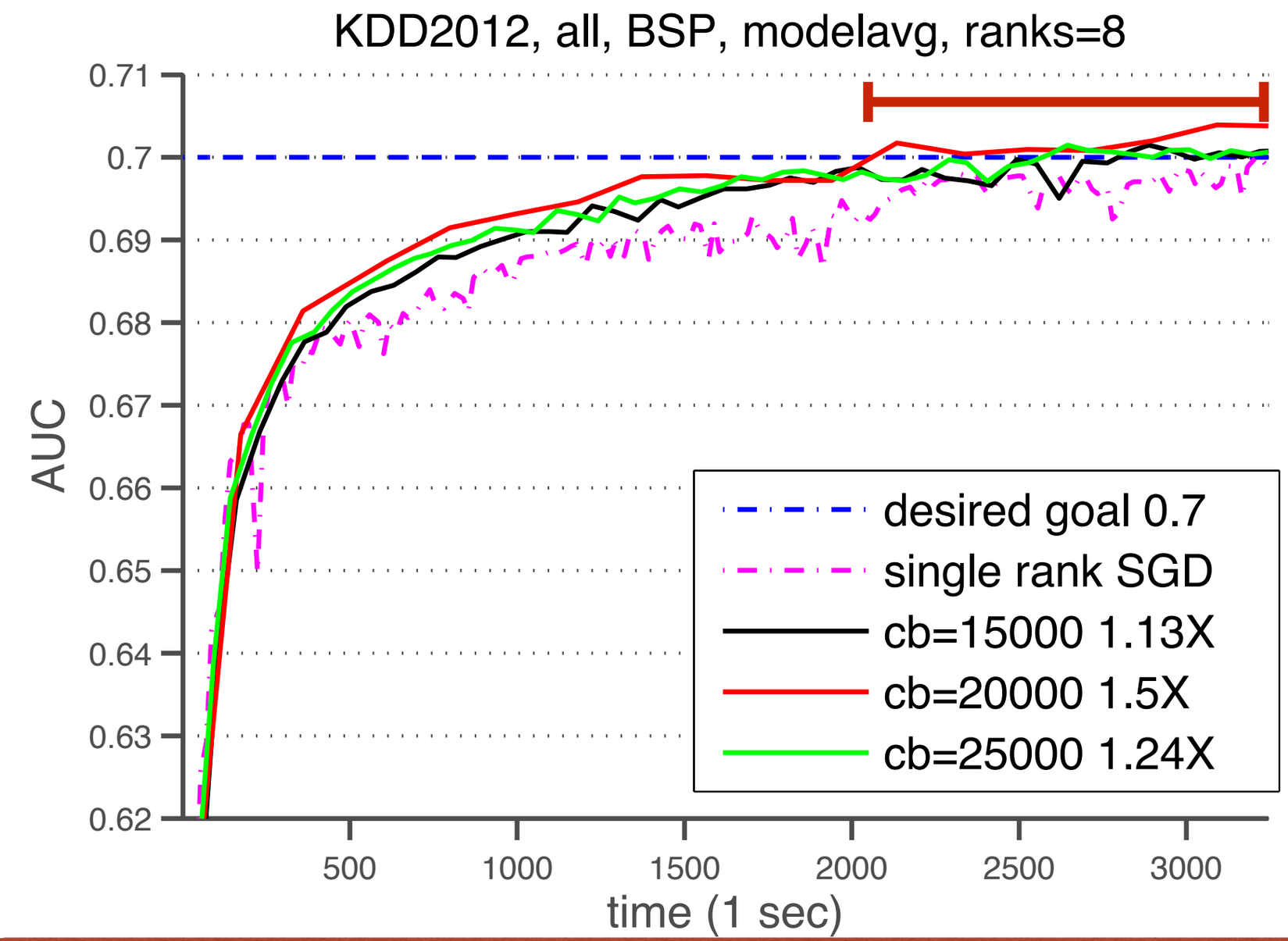
Speedup using SVM-SGD with RCV1 dataset



goal: Loss value as achieved by single rank SGD

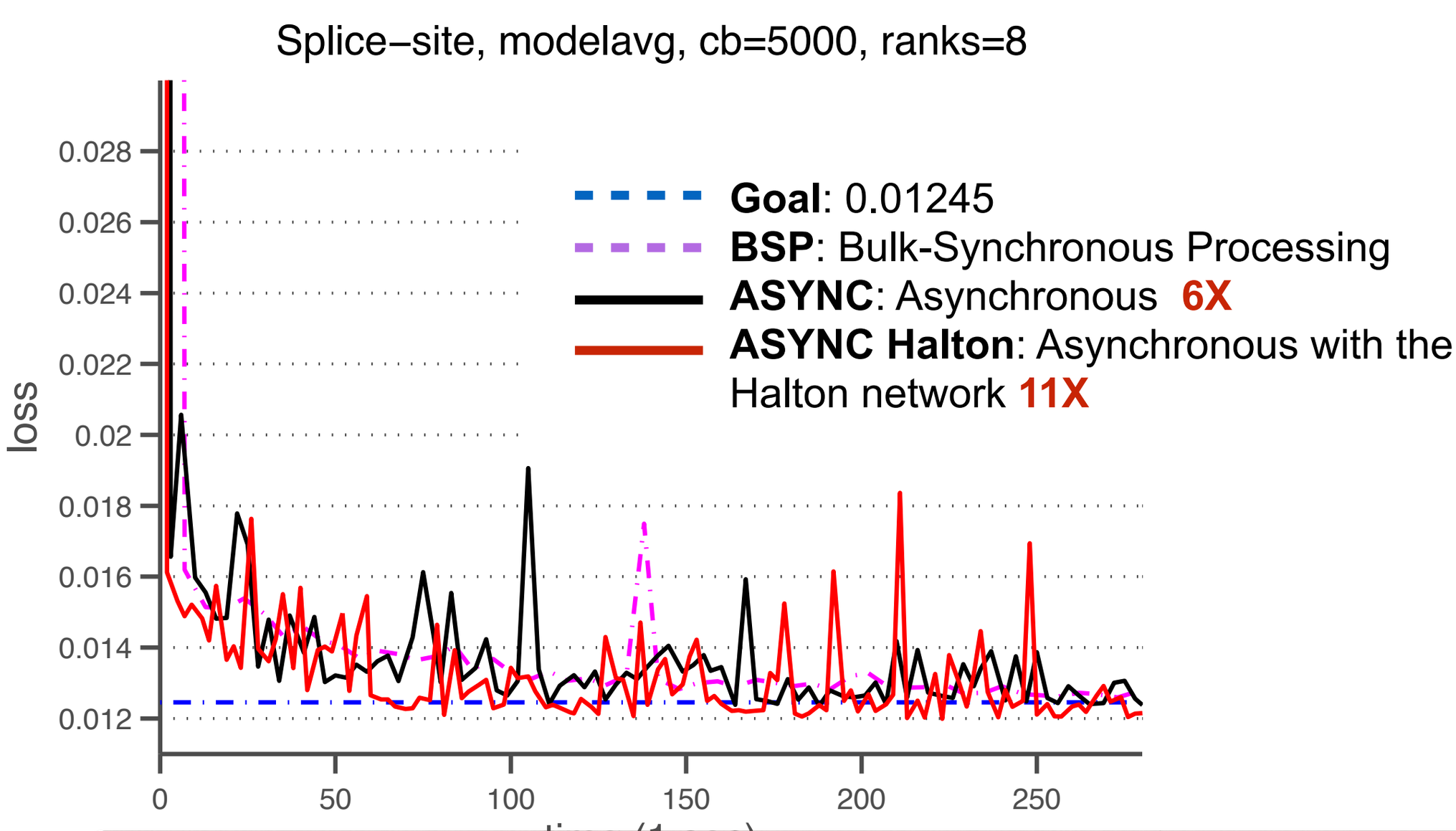
cb size : Communication batch size - Data examples processed before model communication

Speedup using RAPID with KDD 2012 dataset



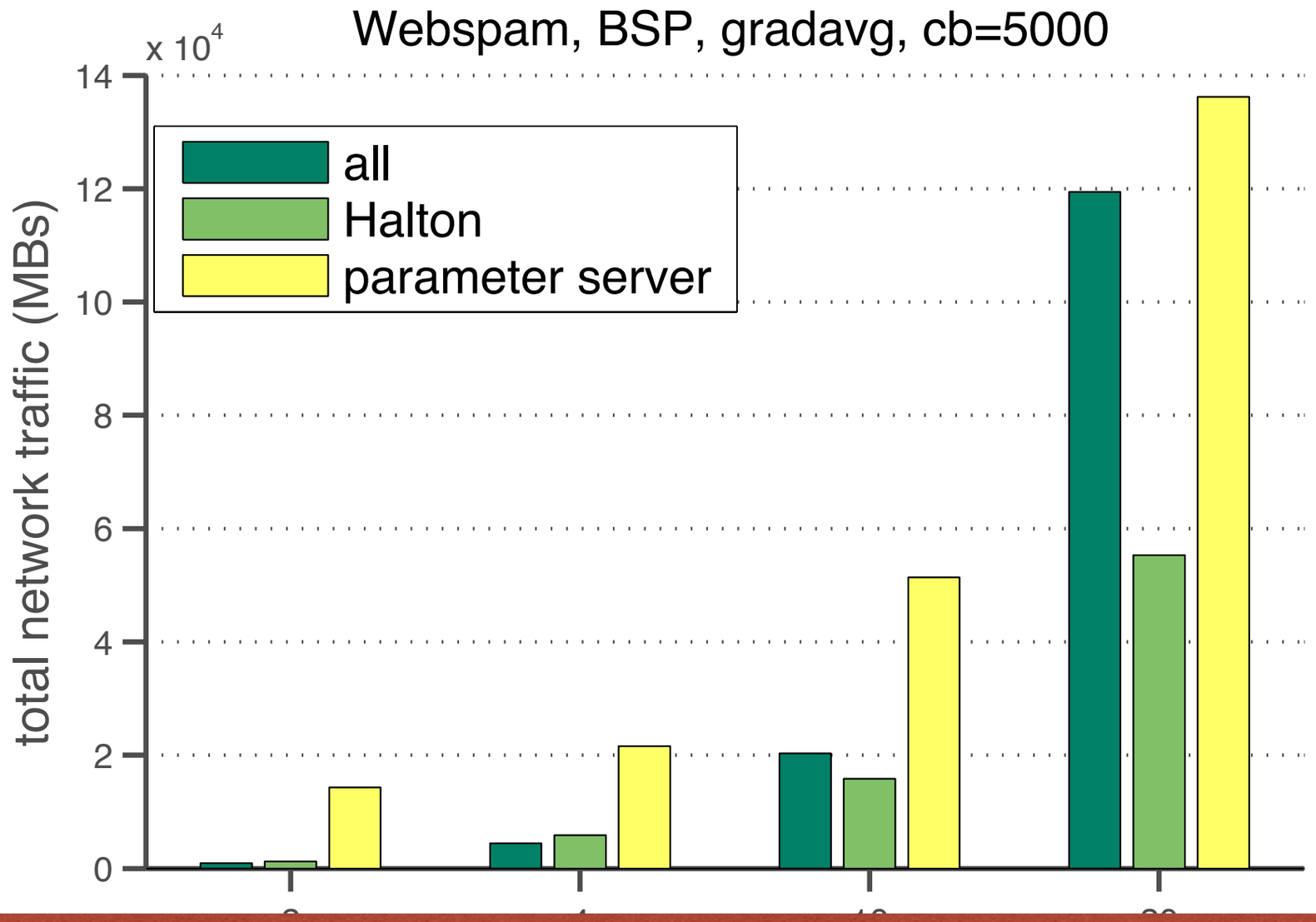
MALT provides speedup over single process SGD

Speedup with the Halton scheme



Indirect propagation of model improves performance

Data transferred over the network for different designs



MALT-Halton provides network efficient learning

Conclusions

- MALT integrates with existing ML software to provide data-parallel learning
  - General purpose **`scatter()/gather()`** API to send model updates using one-sided communication
  - Mechanisms for network/representation optimizations
  - Supports applications written in C++ and Lua

- MALT provides speedup and can process large datasets
  - More results on speedup, network efficiency, consistency models, fault tolerance, and developer efforts in the paper

- MALT uses RDMA support to reduce model propagation costs
  - Additional primitives such as **`fetch_and_add()`** may further reduce model processing costs in software

Thanks

Questions?

Extra slides

Other approaches to parallel SGD

- GPUs
  - Orthogonal approach to MALT, MALT segments can be created over GPUs
  - Excellent for matrix operations, smaller sized models
  - However, a single GPU memory is limited to 6-10 GB
    - Communication costs dominate for larger models
  - Small caches: Require regular memory access for good performance
    - Techniques like sub-sampling/dropout perform poorly
  - Hard to implement convolution (need matrix expansion for good performance)

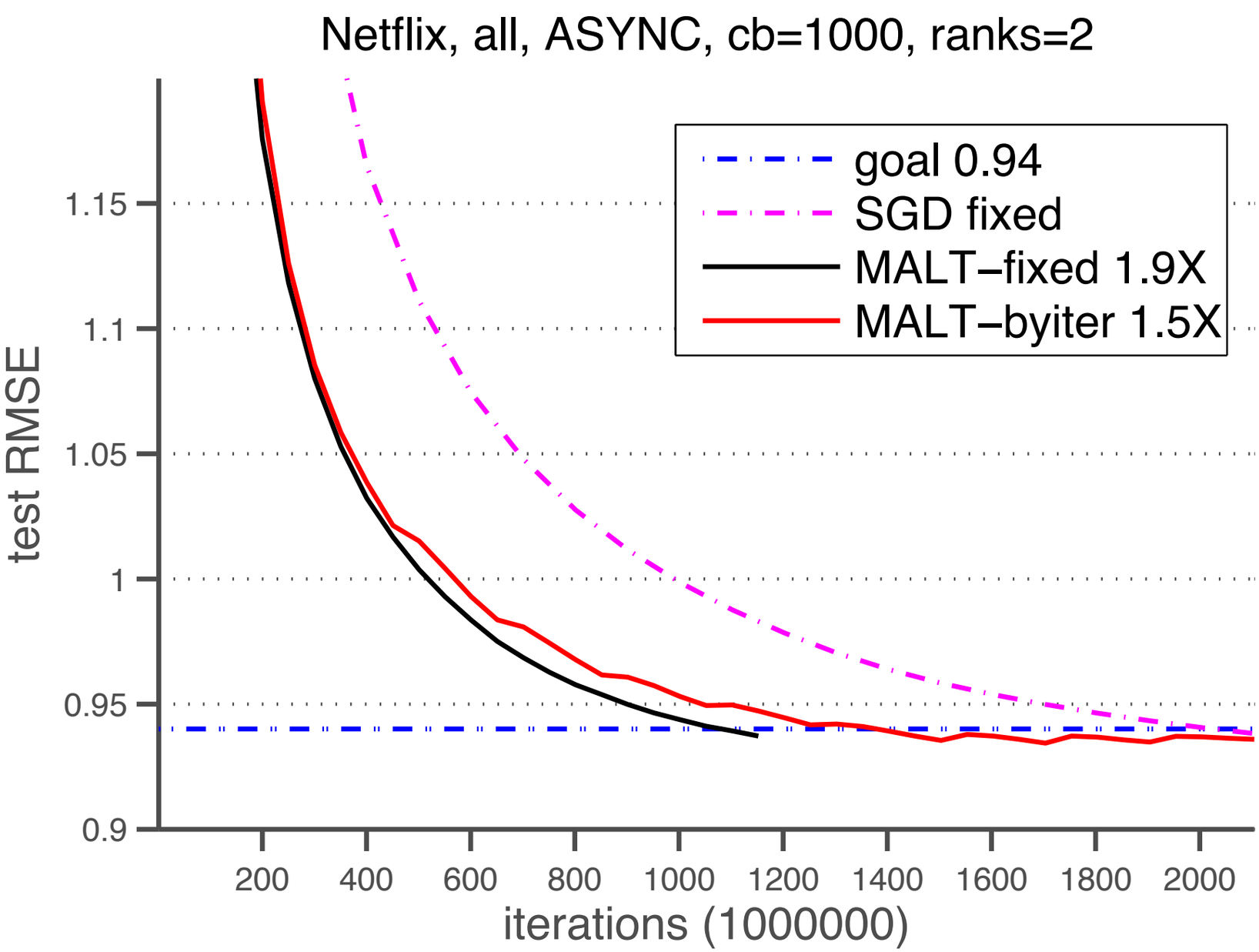
- MPI/Other global address space approaches
  - Offer a low level API (e.g. perform remote memory management)
  - A system like MALT can be built over MPI

Evaluation setup

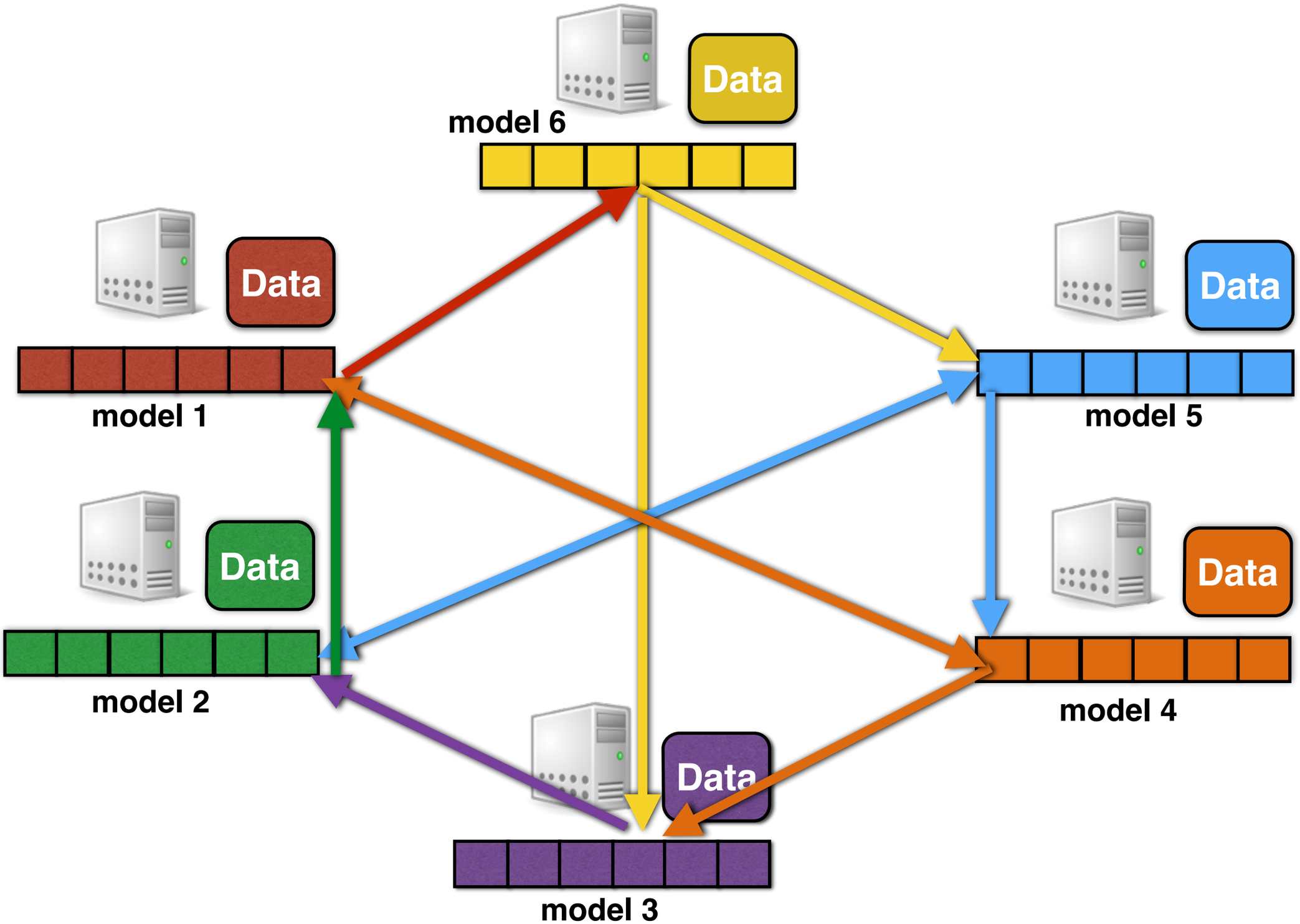
Application	Model	Dataset/Size	# training items	# testing items	# parameters
Document Classification	SVM	RCV1/480M	781K	23K	47K
Image Classification		Alpha/1G	250K	250K	500
DNA Classification		DNA/10G	23M	250K	800
Webspam detection		webspam/ 10G	250K	100K	16.6M
Genome classification		splice-site/ 250G	10M	111K	11M
Collaborative filtering	Matrix factorization	netflix/1.6G	100M	2.8M	14.9M
Click Prediction	Neural networks	KDD2012/ 3.1G	150M	100K	12.8M

- Eight Intel 8-core, 2.2 GHz Ivy-Bridge, with 64 GB
  - All machines connected via Mellanox 56 Gbps infiniband

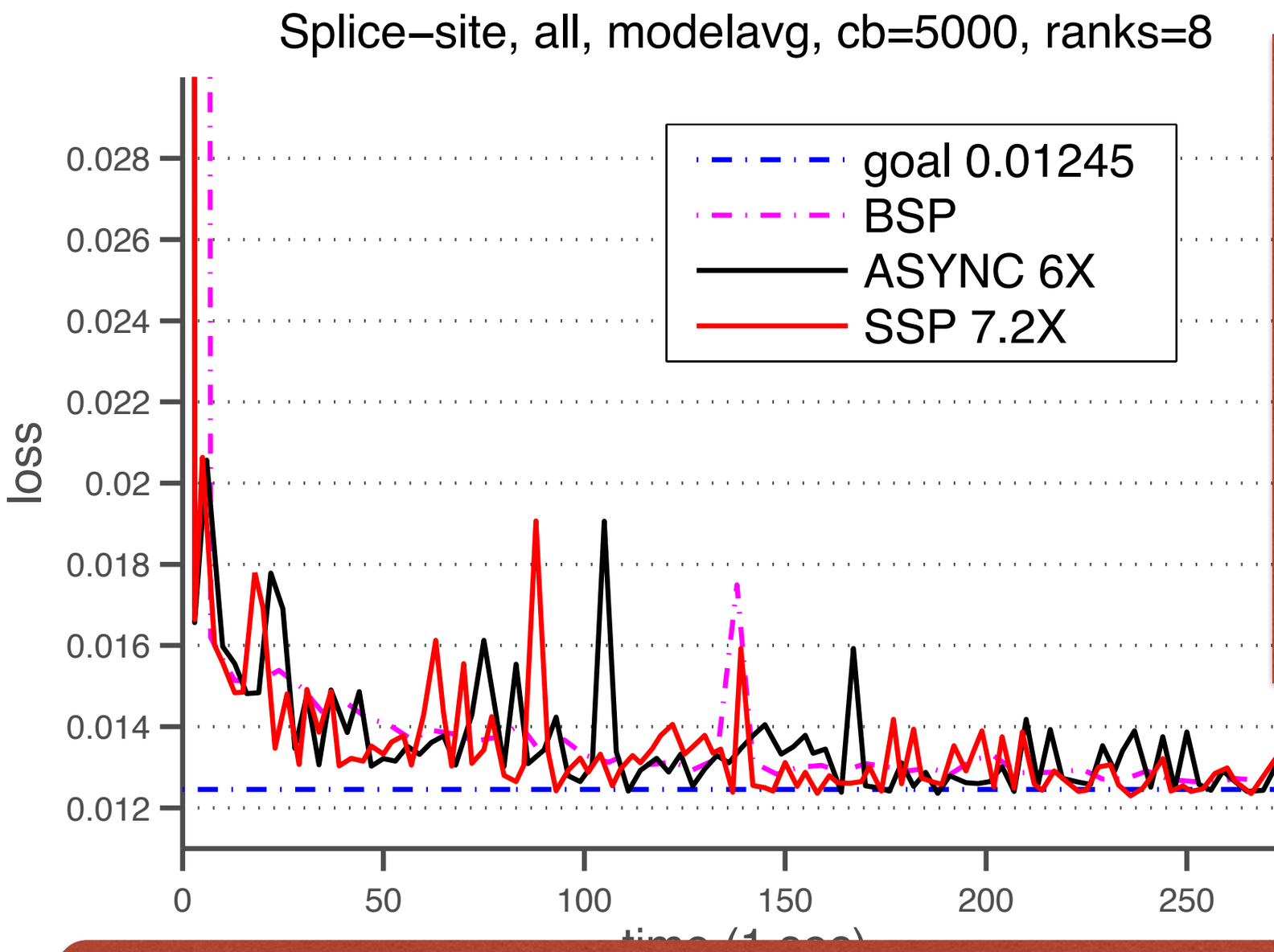
Speedup using NMF with netflix dataset



Halton sequence: n —> n/2, n/4, 3n/4, 5n/8, 7n/8



Speedup with different consistency models



- BSP: Bulk-synchronous parallelism (training with barriers)
- ASYNC: Fully asynchronous training
- SSP: Stale synchronous parallelism (limit stragglers by stalling fore-runners)

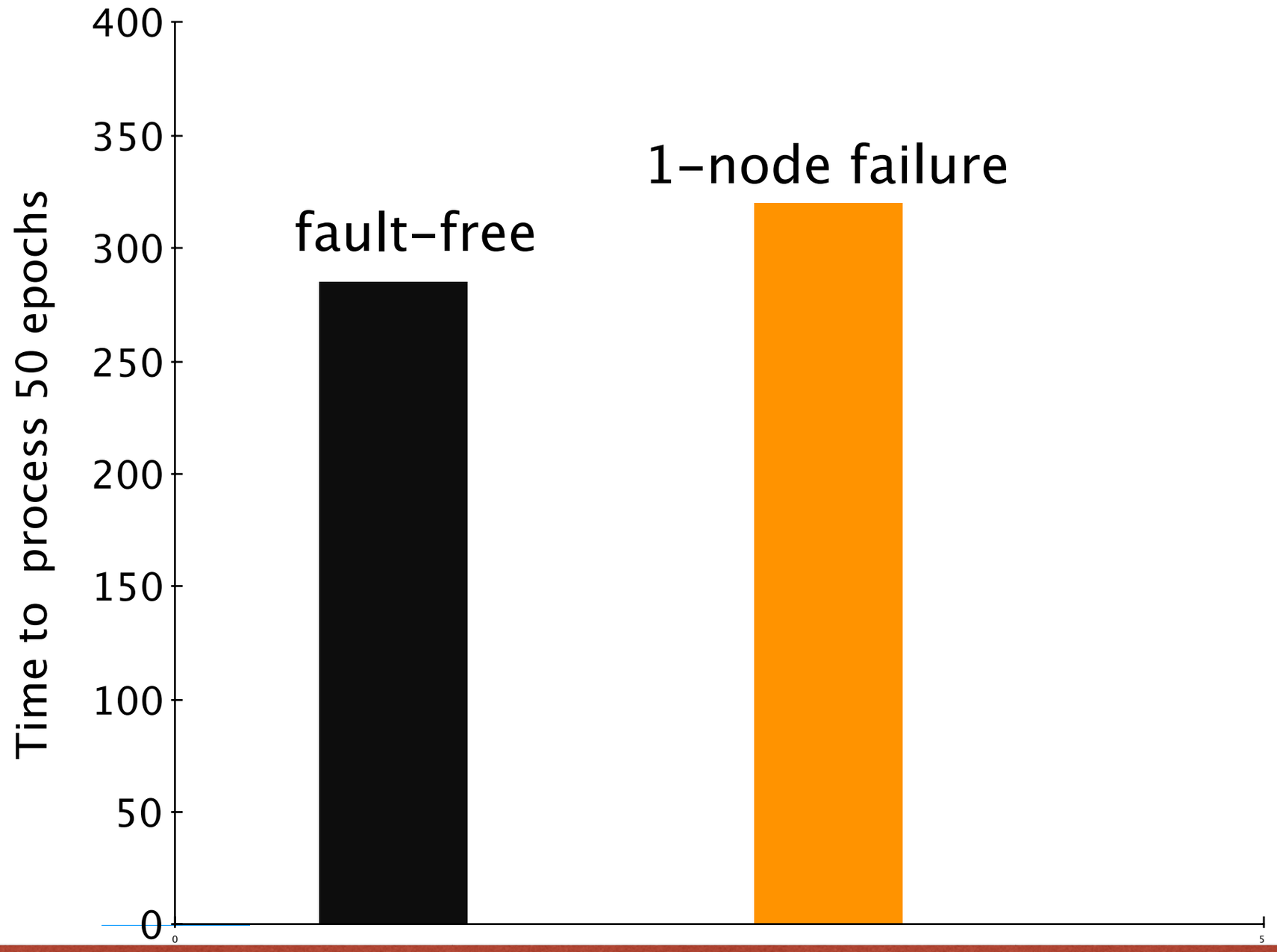
Benefit with asynchronous training

Developer efforts to make code data-parallel

Application	Dataset	LOC Modified	LOC Added
SVM	RCV1	105	107
Matrix Factorization	Netflix	76	82
Neural Network	KDD 2012	82	130

On an average, about 15% of lines modified/added

Fault tolerance



MALT provides robust model training