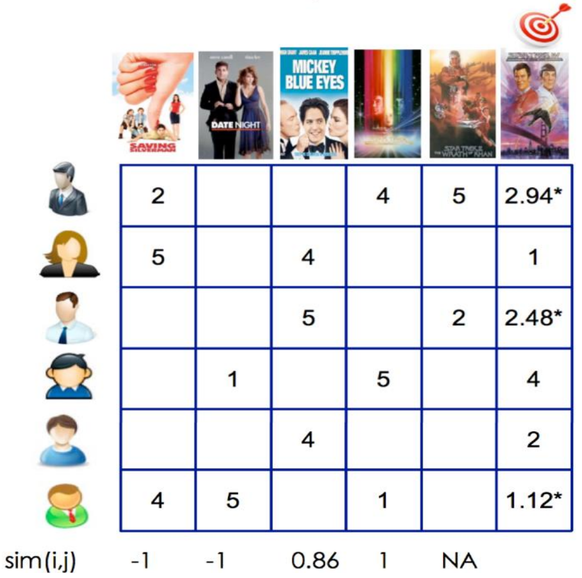
TuX^2: Distributed Graph Computation for Machine Learning

Wencong Xiao[*◊], Jilong Xue[◊†], Youshan Miao[◊], Zhen Li[*◊], Cheng Chen[◊], Ming Wu[◊], Wei Li[*], Lidong Zhou[◊]

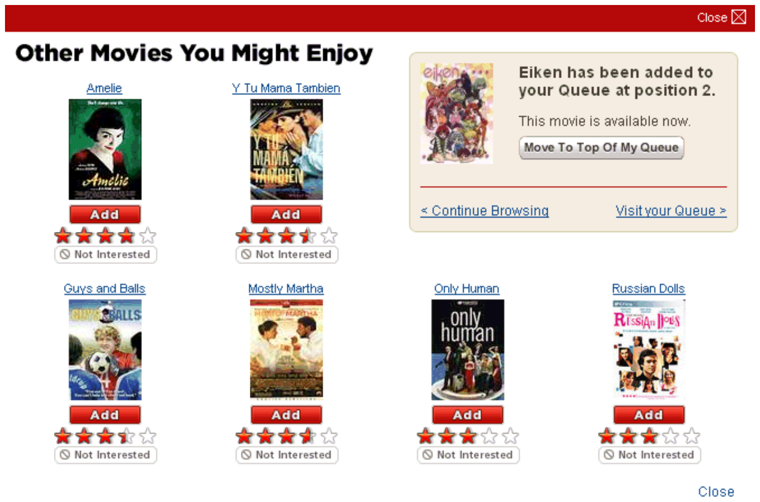
[*]SKLSDE Lab, Beihang University; [◊]Microsoft Research; [†]Peking University

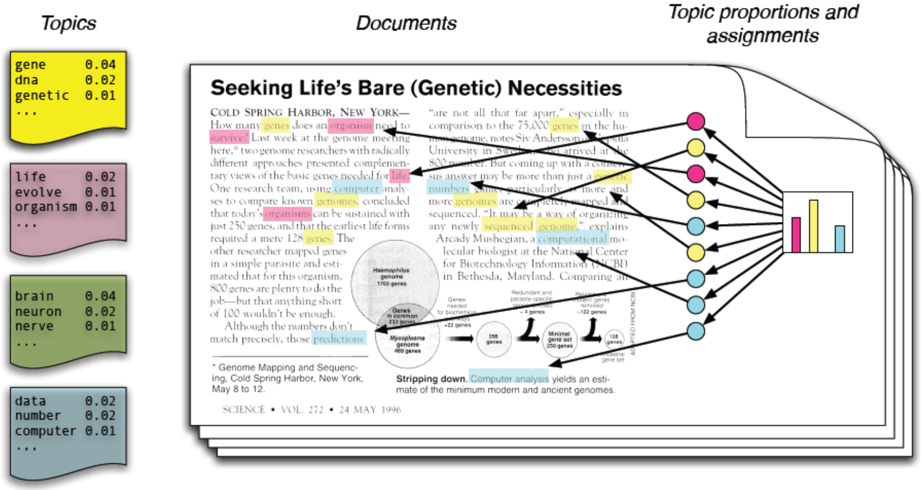
NSDI'17, Boston, MA

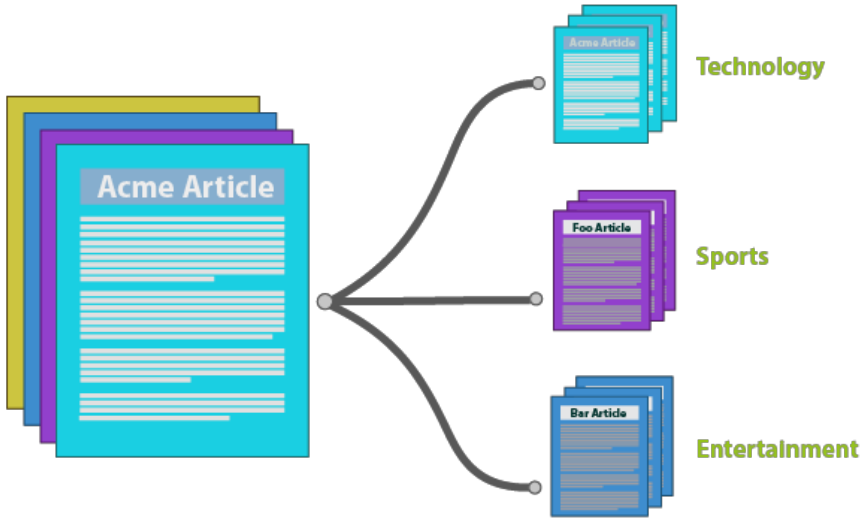
Machine Learning(ML) in real world



Recommendation

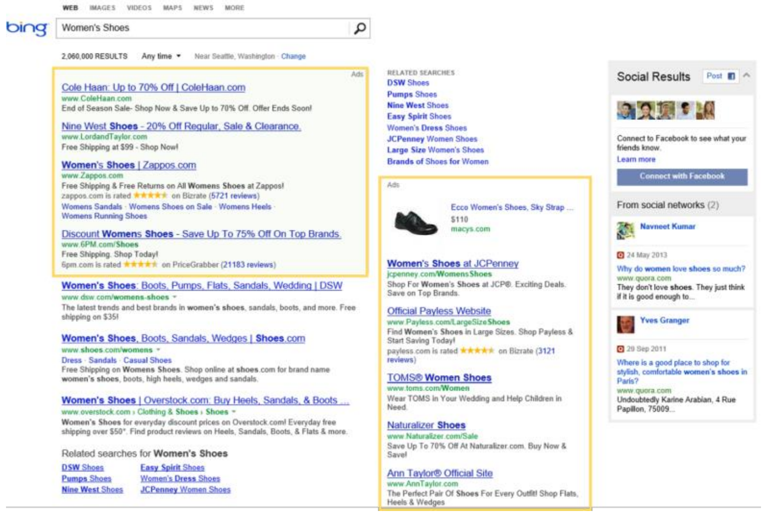




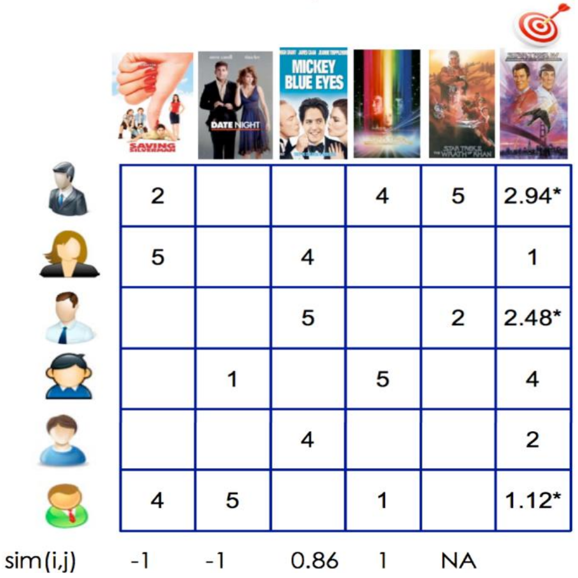


Topic Model

Click Prediction



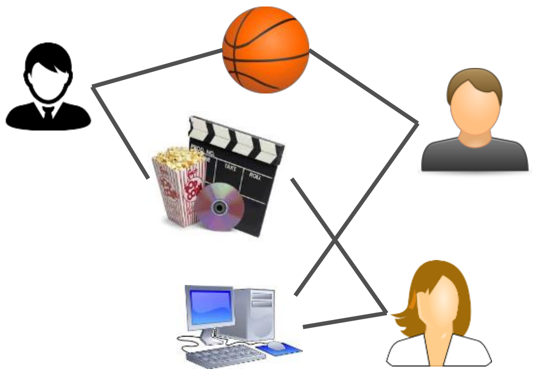
Graph Structures in Machine Learning

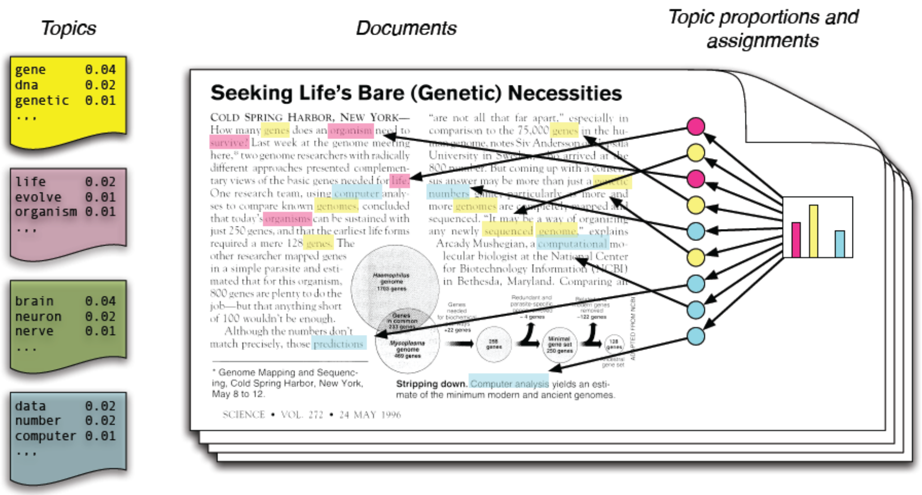


2			4	5	2.94*
5		4			1
		5		2	2.48*
	1		5		4
		4			2
4	5		1		1.12*

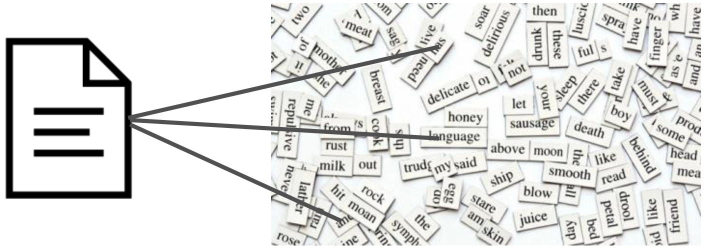
sim(i,j)	-1	-1	0.86	1	NA

Recommendation

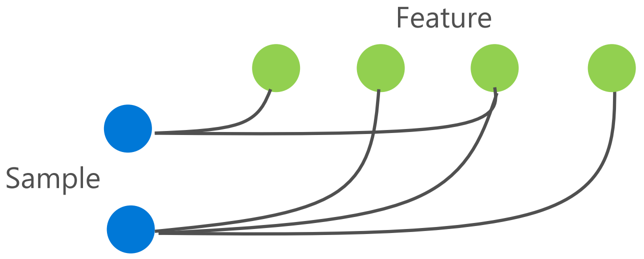




Topic Model



Click Prediction



Advantages of Graph Engine

Simple programming model (e.g. GAS)

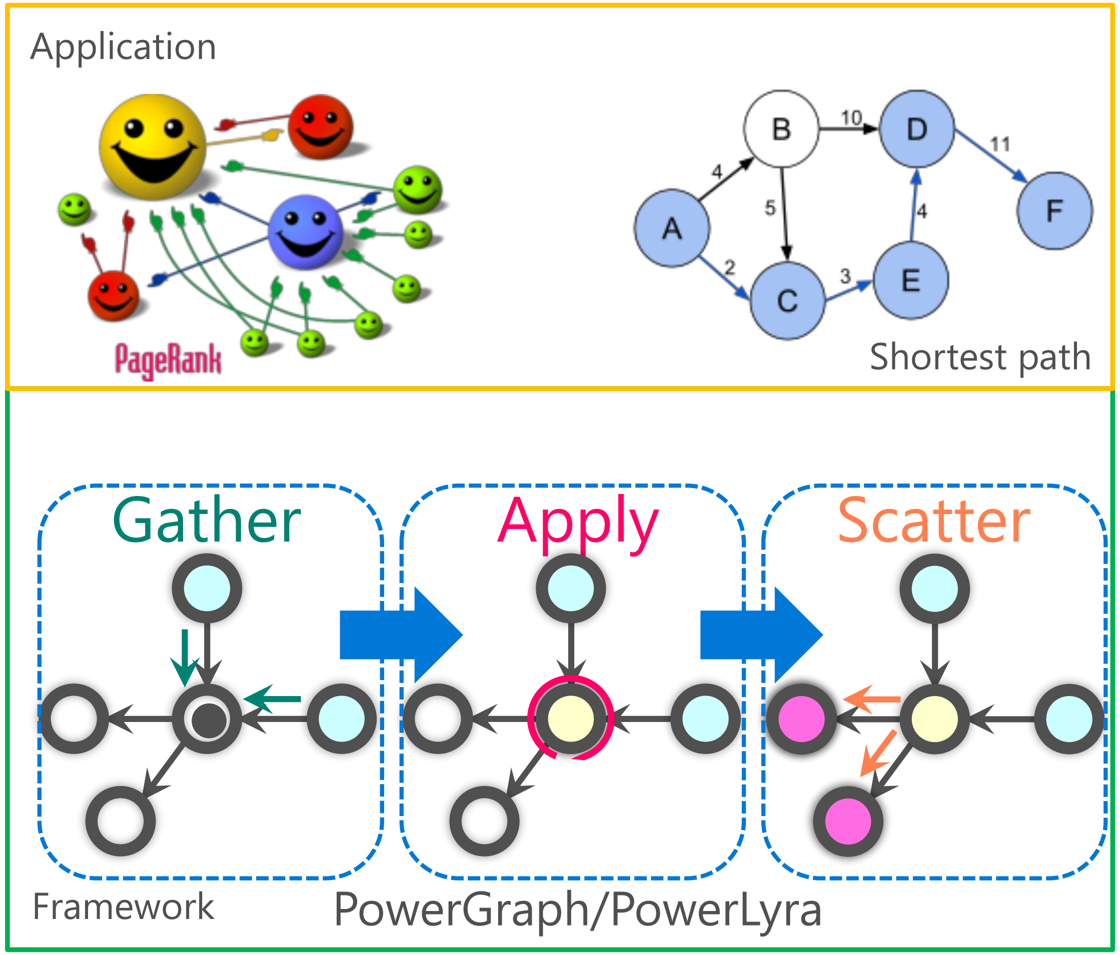
- PageRank, Shortest path, etc.

Graph-aware optimization

- Data layout [Grace(ATC'12), Naiad(SOSP'13)]
- Partitioning [PowerLyra(EuroSys'15)]

Scalability to trillion-edge

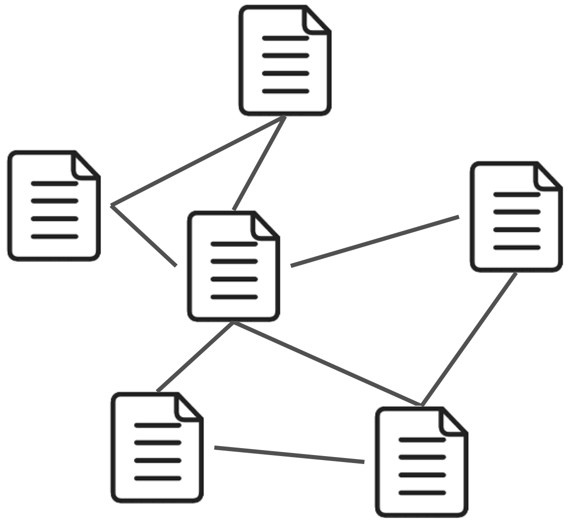
- GraM (SoCC'15)
- Chaos (SOSP'15)
- One Trillion Edges (VLDB'15)



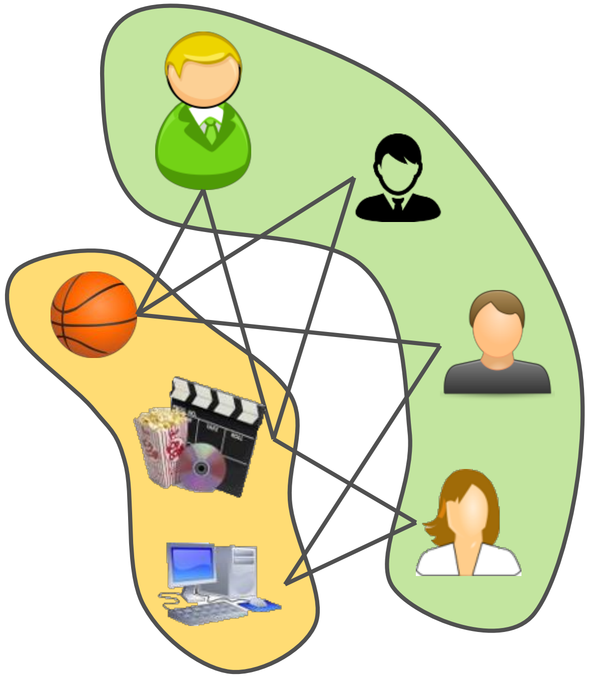
*GAS figure from PowerLyra slides

Gaps for ML on Graph Engine

1. Heterogeneous vertices



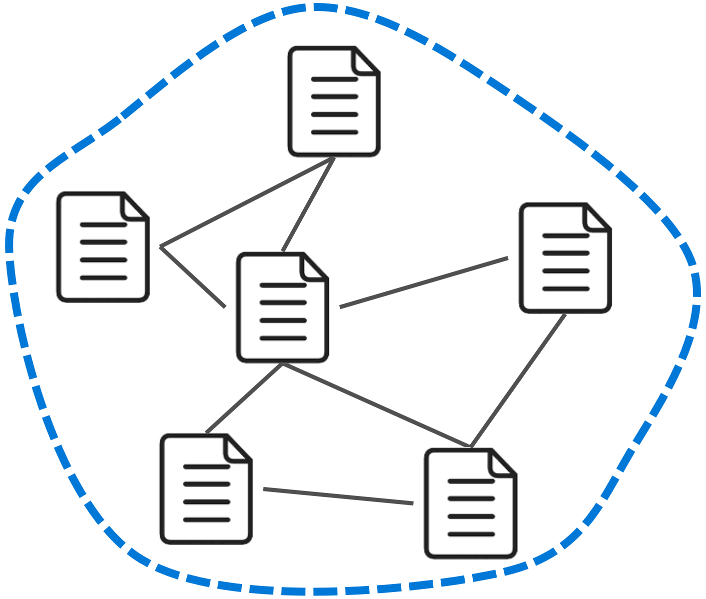
PageRank
for WebPage Ranking



Matrix Factorization(MF)
for Recommendation

Gaps for ML on Graph Engine

2. Mini-Batch

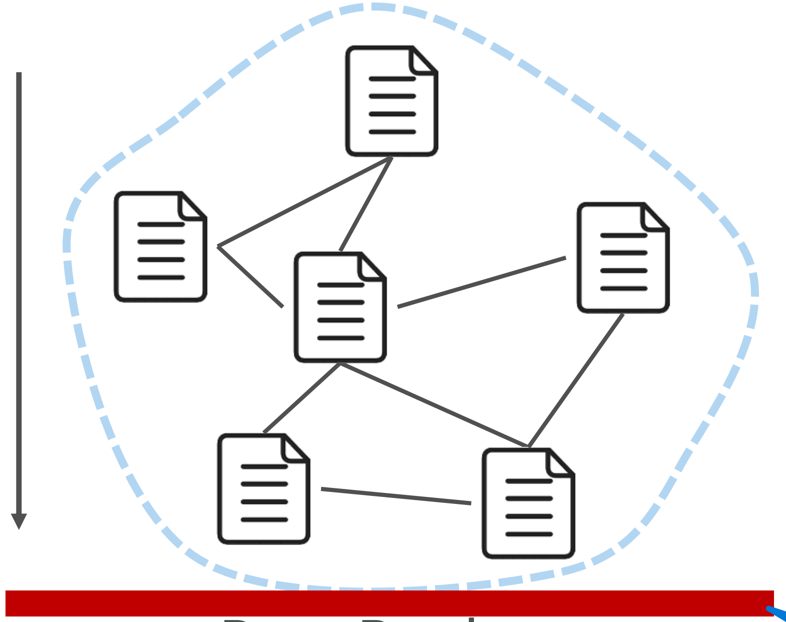


PageRank
for WebPage Ranking

Matrix Factorization(MF)
for Recommendation

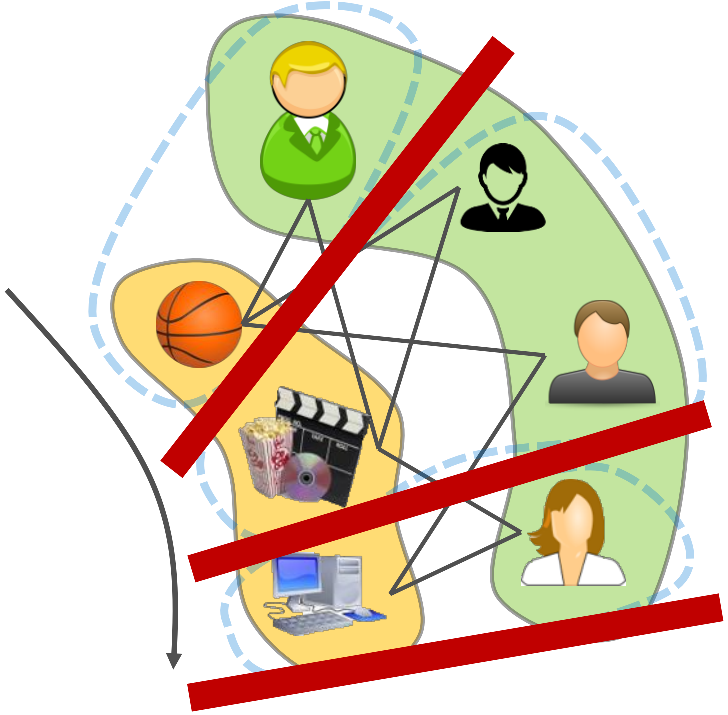
Gaps for ML on Graph Engine

3. Flexible consistency



PageRank
for WebPage Ranking

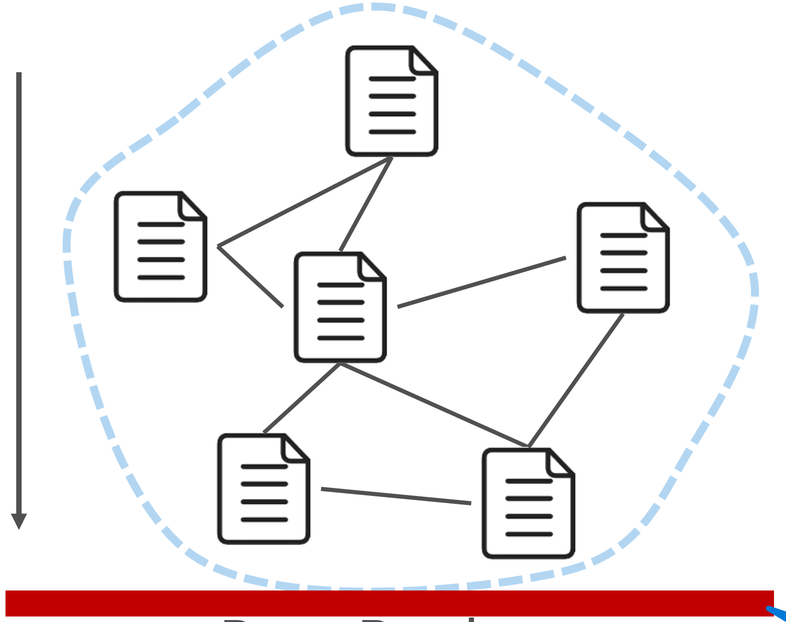
Hard Barrier



Matrix Factorization(MF)
for Recommendation

Gaps for ML on Graph Engine

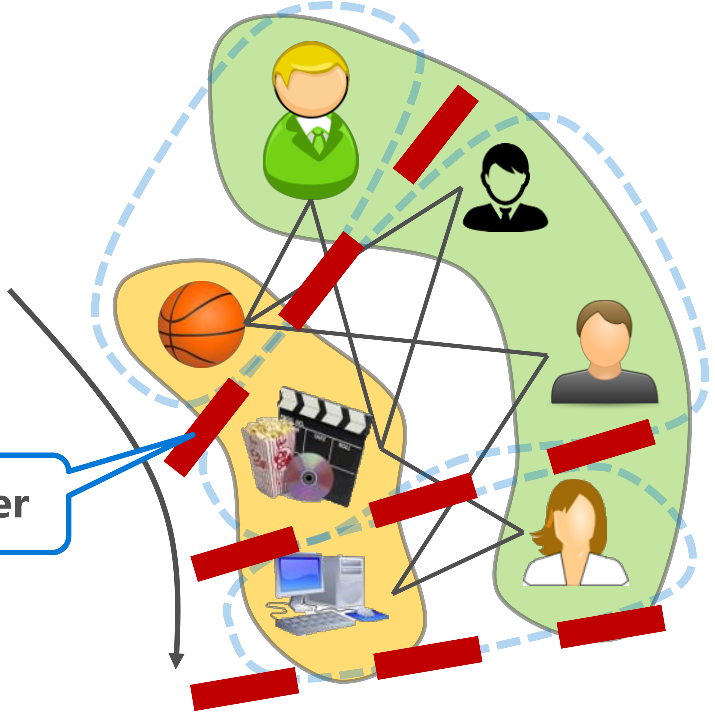
3. Flexible consistency



PageRank
for WebPage Ranking

Hard Barrier

“Soft” Barrier



Matrix Factorization(MF)
for Recommendation

We propose: TuX^2

Bridge Graph and ML research in one system

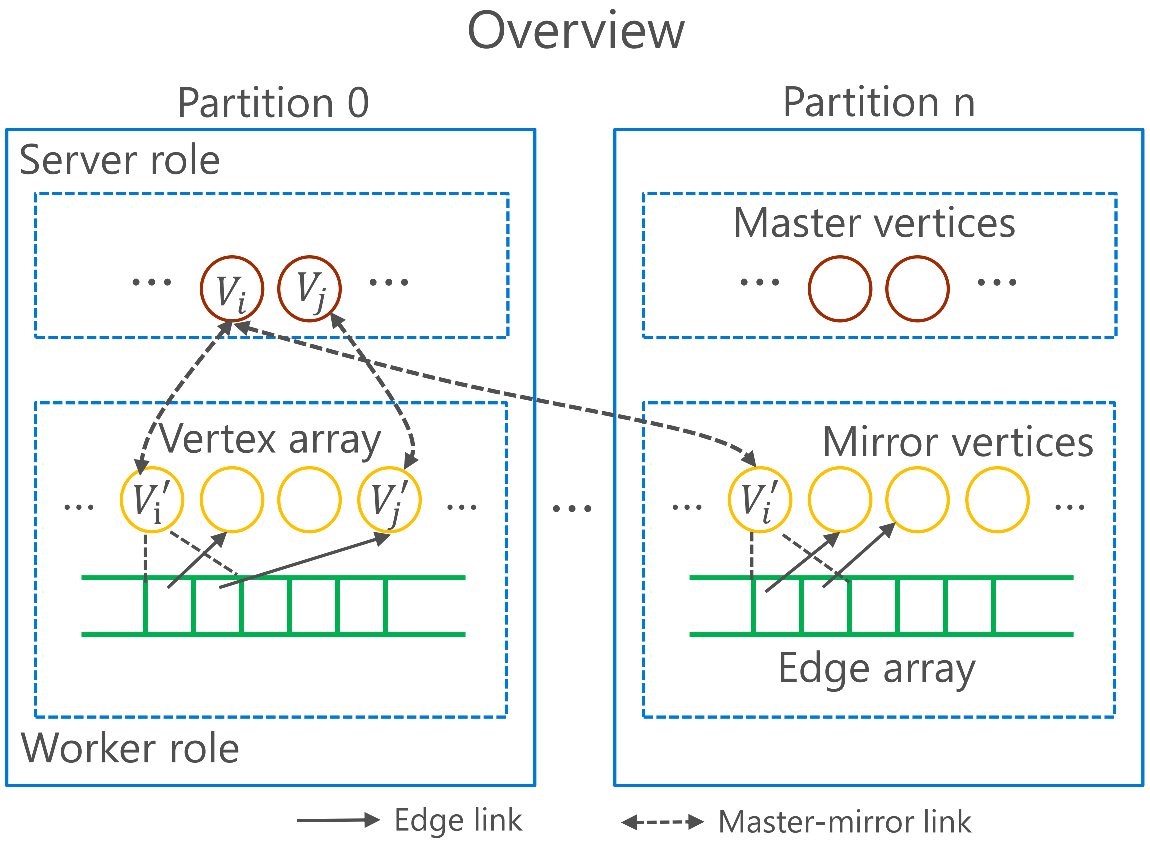
Extend for distributed machine learning

- Scheduling: Stale Synchronous Parallel (SSP) based scheduling
- DataModel: Heterogeneous data model
- Programming: MEGA (Mini-batch, Exchange, GlobalSync, and Apply) graph model

Outperform both Graph and ML systems on ML algorithms

- **10x** ↗ vs. PowerGraph/PowerLyra
  - Mainly due to MEGA model and heterogeneity optimization
- **48%** ↗ vs. Petuum/Parameter-Server(P-S)
  - Mainly due to graph-based optimization

System Architecture



Vertex-cut approach

- Effective for power-law graph
- Naturally fits P-S model
  - Master vertices as the global state
  - Mirror vertices as the local cache

Key designs

Scheduling: Stale Synchronous Parallel (SSP) based scheduling

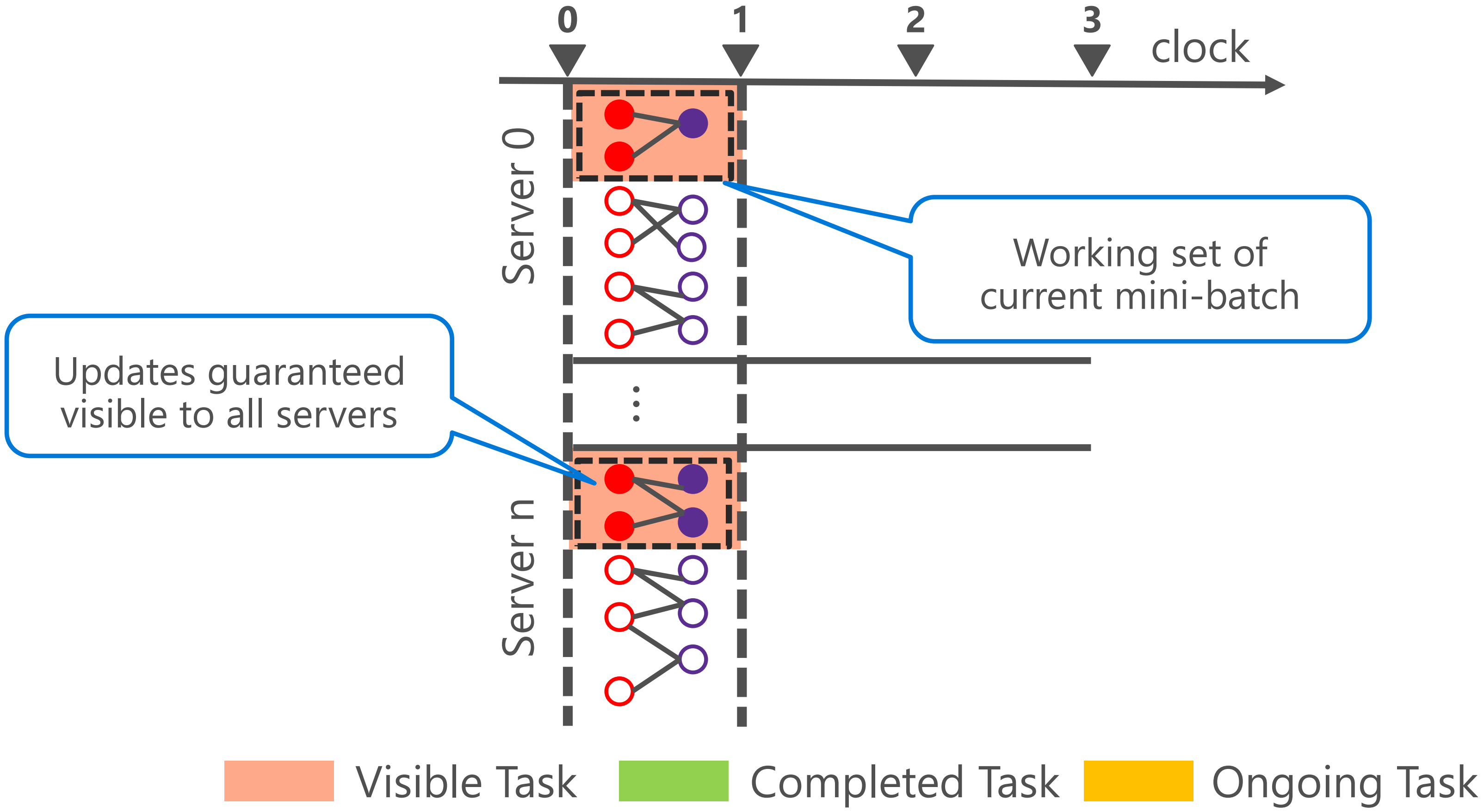
DataModel: Heterogeneous data model

Programming: MEGA graph model

Stale Synchronous Parallel in TuX^2

Slack of 1 clock as an example

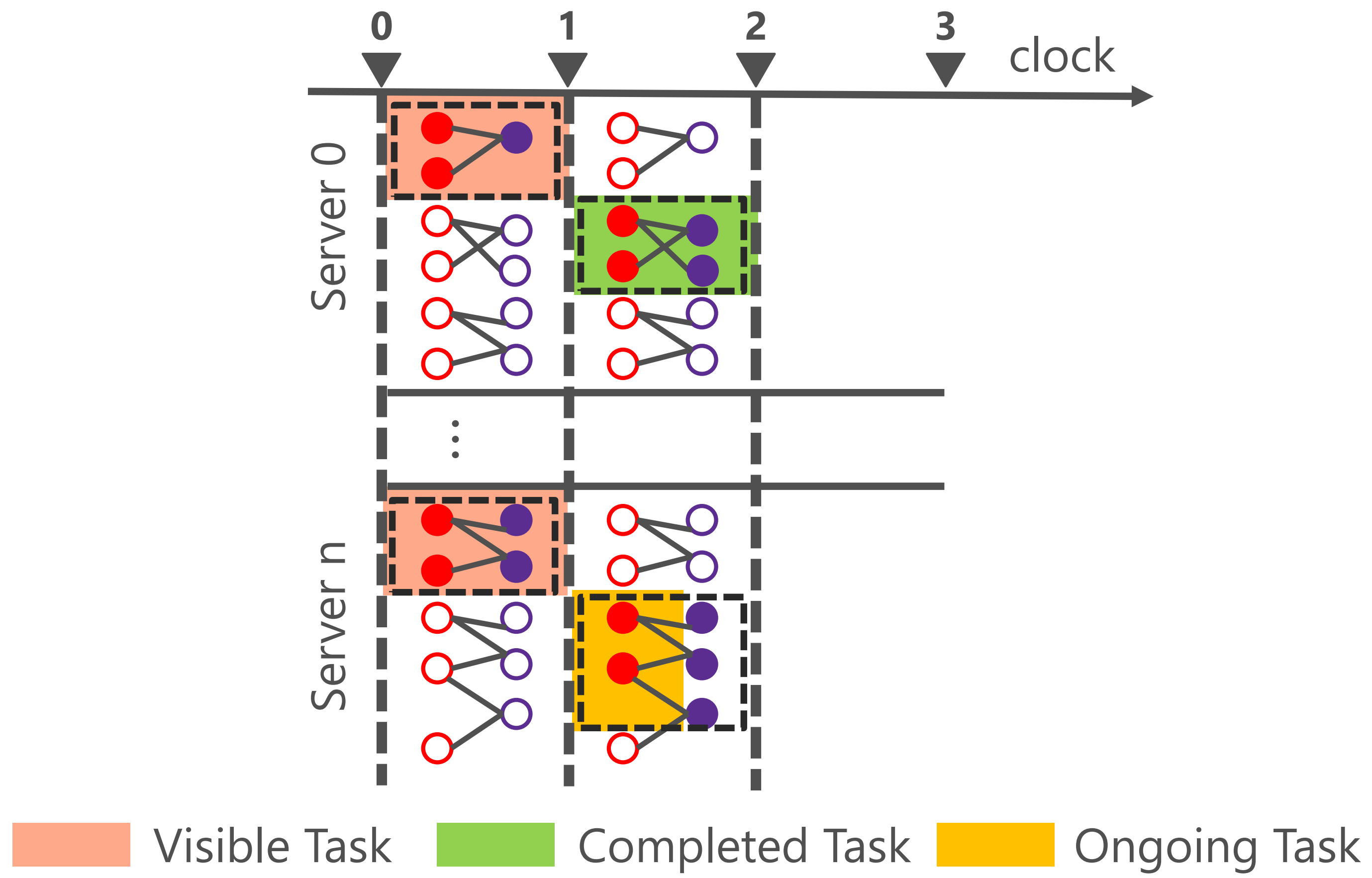
- All servers finish clock1



Stale Synchronous Parallel in TuX^2

Slack of 1 clock as an example

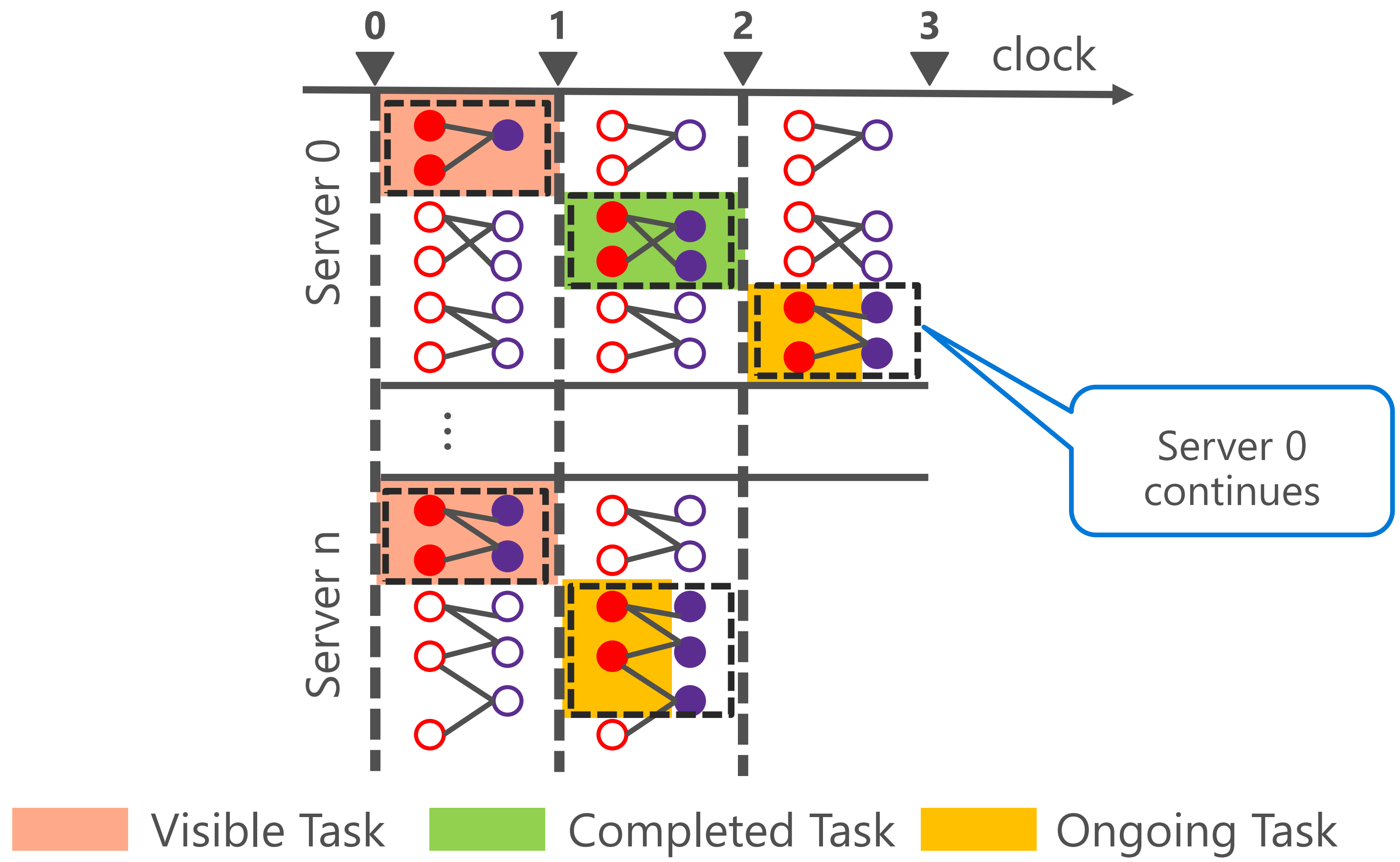
- Slowest server (n) is in clock2
- Fastest server (0) finishes clock2



Stale Synchronous Parallel in TuX^2

Slack of 1 clock as an example

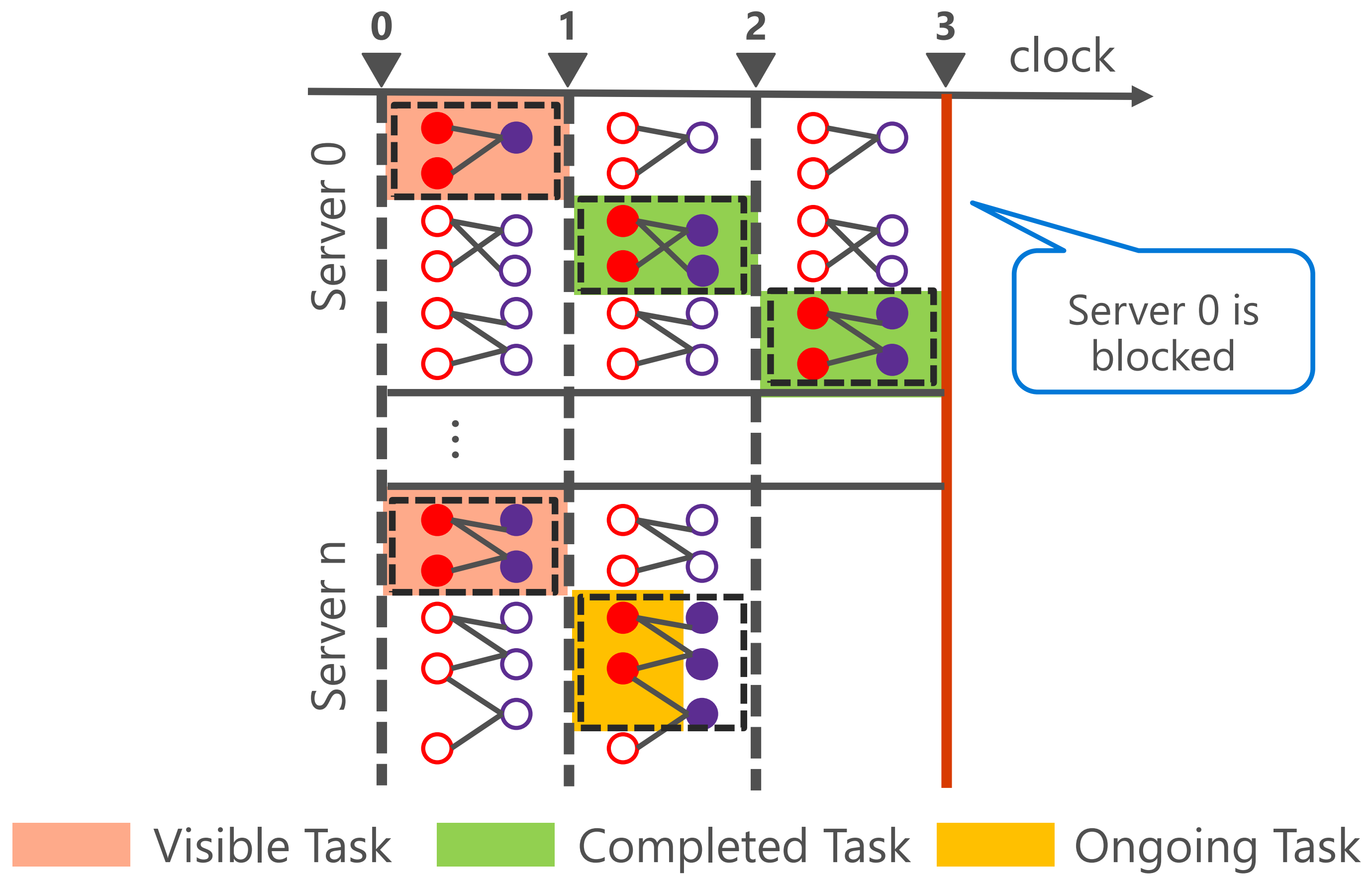
- Slowest server (n) is in clock2
- Fastest server (0) finishes clock2
  - within the staleness bound
  - continue



Stale Synchronous Parallel in TuX^2

Slack of 1 clock as an example

- Slowest server (n) is in clock2
- Fastest server (0) finishes clock3
  - reaching the max slack bound
  - blocked



Key designs

Scheduling: Stale Synchronous Parallel (SSP) based scheduling

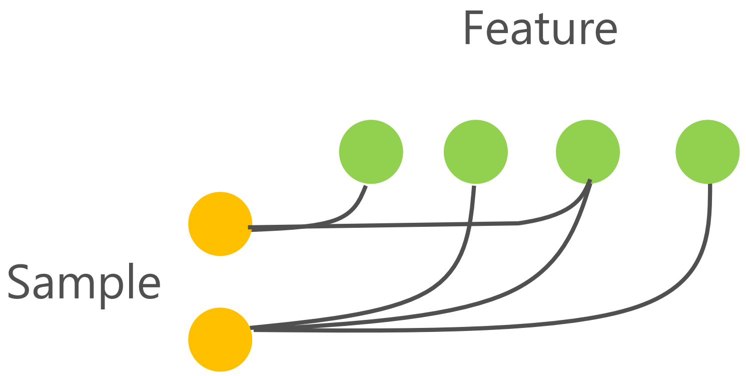
DataModel: Heterogeneous data model

Programming: MEGA graph model

Heterogeneity in ML

Heterogeneous Vertices

- Different properties
  - E.g. Logistic Regression
    - Sample: Label; Feature: Weight, Gradient



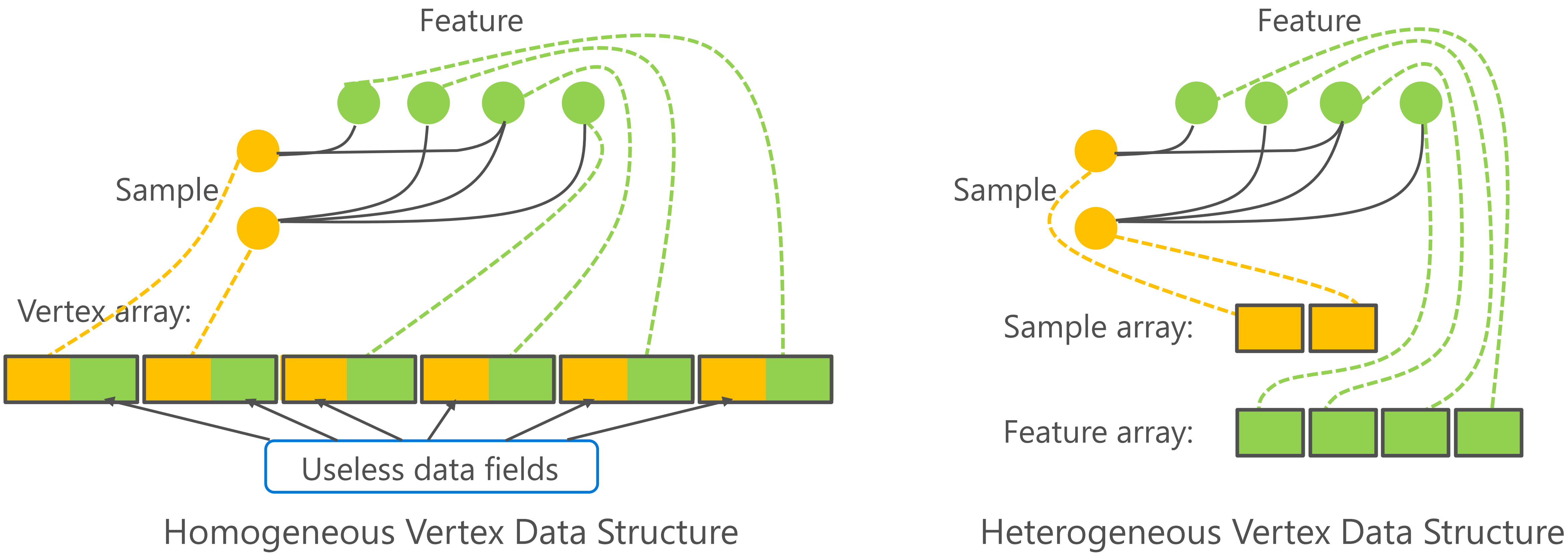
Benefit

- Heterogeneity for compact data structure
- Heterogeneity for efficient execution
- Heterogeneity for less network traffic

Heterogeneity for compact data structure

E.g. Logistic Regression

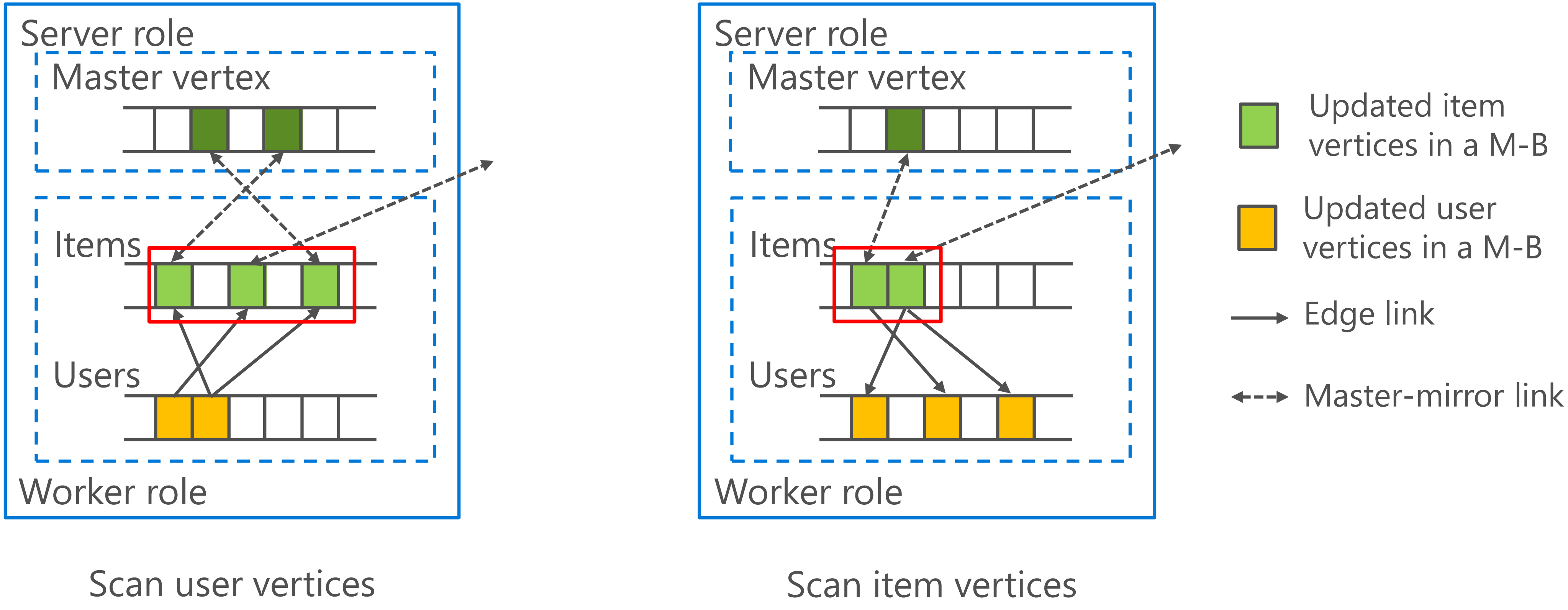
- Sample: Label; Feature: Weight, Gradient



Heterogeneity for efficient execution

E.g. Mini-Batch MF for recommendation

- Benefits of scanning items
  - Sequential access for locality when syncing
  - Less overhead tracing the updated vertices



Key designs

Scheduling: Stale Synchronous Parallel (SSP) based scheduling

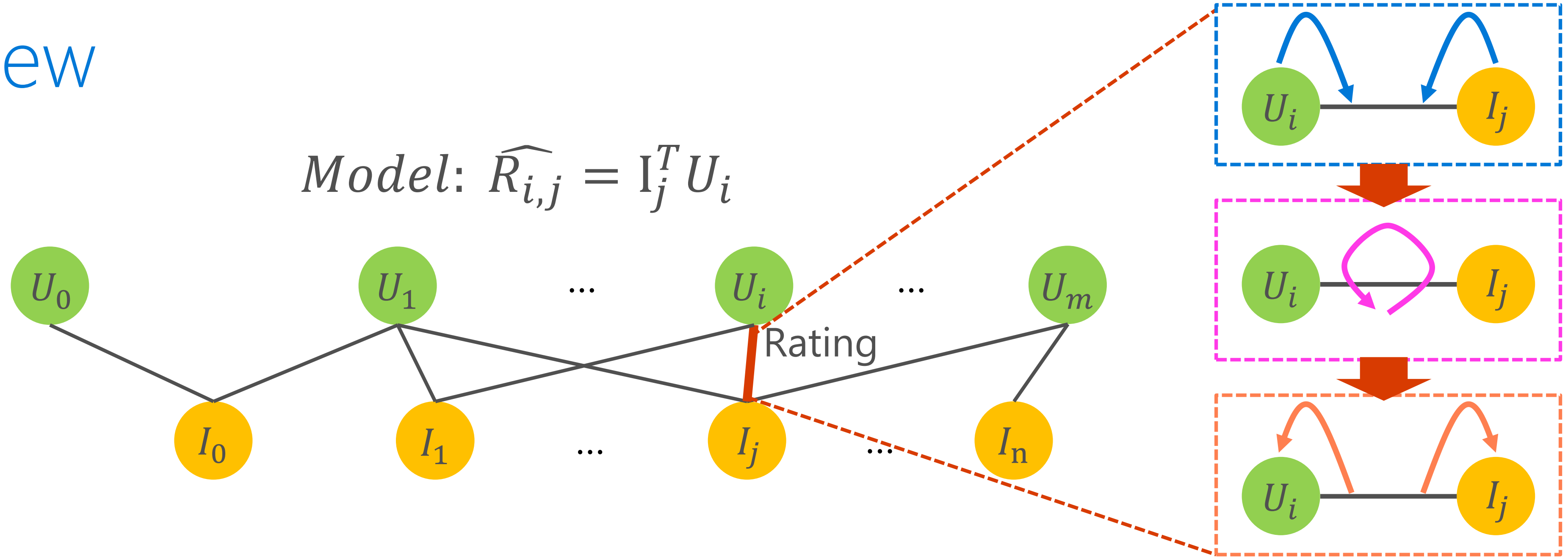
DataModel: Heterogeneous data model

Programming: MEGA graph model

MEGA: e.g. Mini-batch MF for recommendation

Graph View

$$Model:\ \widehat{R_{i,j}} = \mathrm{I}_j^T U_i$$

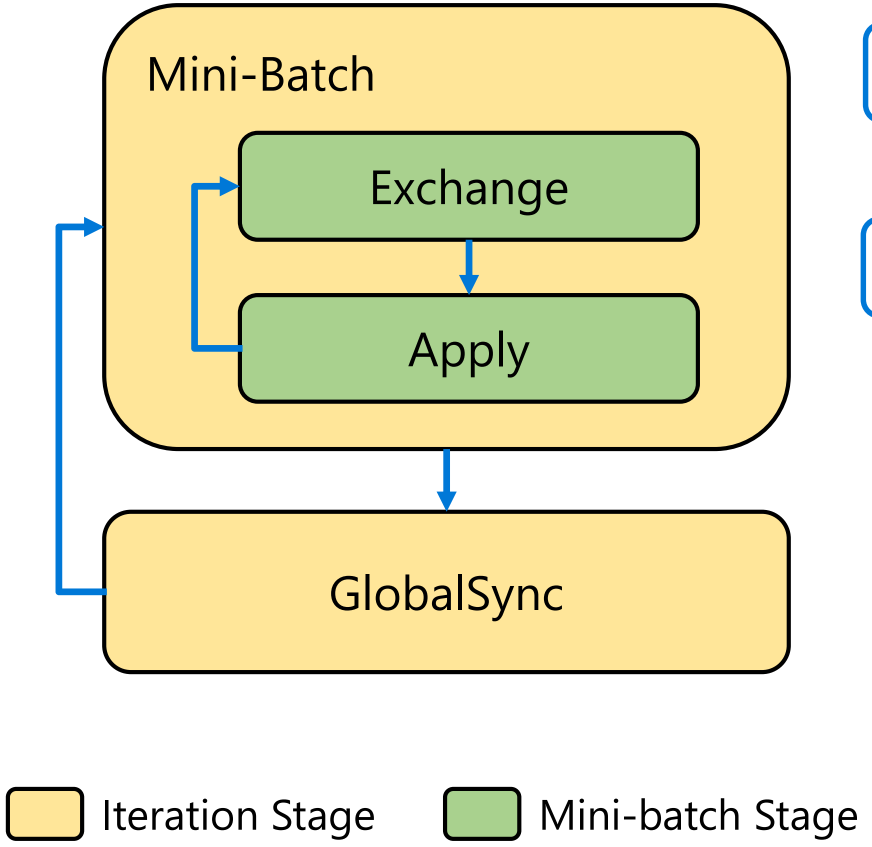


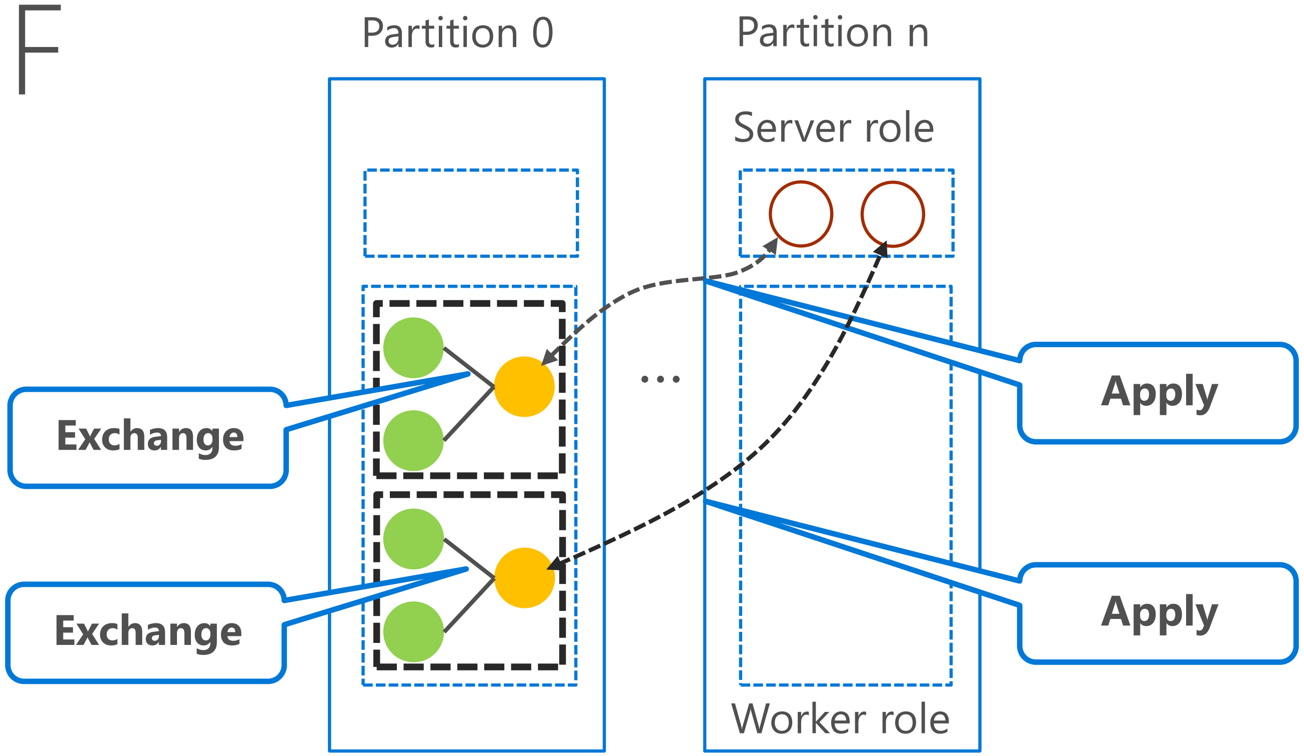

```
Exchange(v_user, v_item, edge,
a_user, a_item, context)
  pred=PredictRating(v_user, v_item);
  loss=edge.rating–pred;
  context.loss+=loss^2;
  (a_user, a_item)+=Gradient(loss,
v_user, v_item);
```

```
Apply(ver, accum,
context)
  ver.data +=accum;
```

Example: Mini-batch MF

Compose stage






```
StageSequenceBuilder(ExecStages)
 mbStage = new MiniBatchStage();
 mbStage.SetBatchSize(100, asEdge);
 mbStage.Add(ExchangeStage);
 mbStage.Add(ApplyStage);
 ExecStages.Add(mbStage);
 ExecStages.Add(GlobalSyncStage);
```

Experiment setup

Machine information

- 16 CPU cores, 256GB memory, 54Gbps InfiniBand NIC

Typical ML algorithms

- MF, LDA, BlockPG

Large-scale dataset

- Up to 64 billion edges graph

Dataset name	# of users/ docs/samples	# of items/ words/features	# of edges
NewsData(LDA)	7.3M	418.4K	1.4B
AdsData(BlockPG)	924.8M	209.3M	64.9B
Netflix(MF)	480.2K	17.8K	100.5M
Synthesized(MF)	30M	1M	6.3M

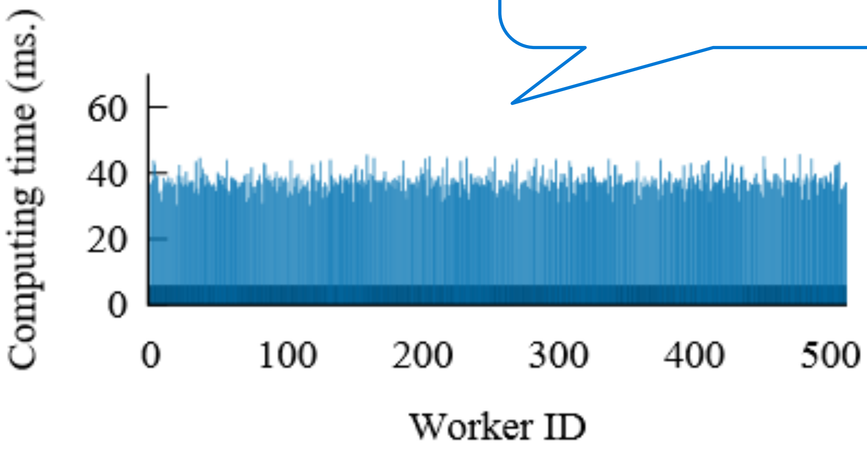
Evaluation

Compare to Parameter Server

- **48%** improvement on 32 servers!
- Algorithm: BlockPG
- Dataset: Microsoft private AdsData (64B edges)

Balance workload with vertex-cut!

Imbalance in Parameter Server

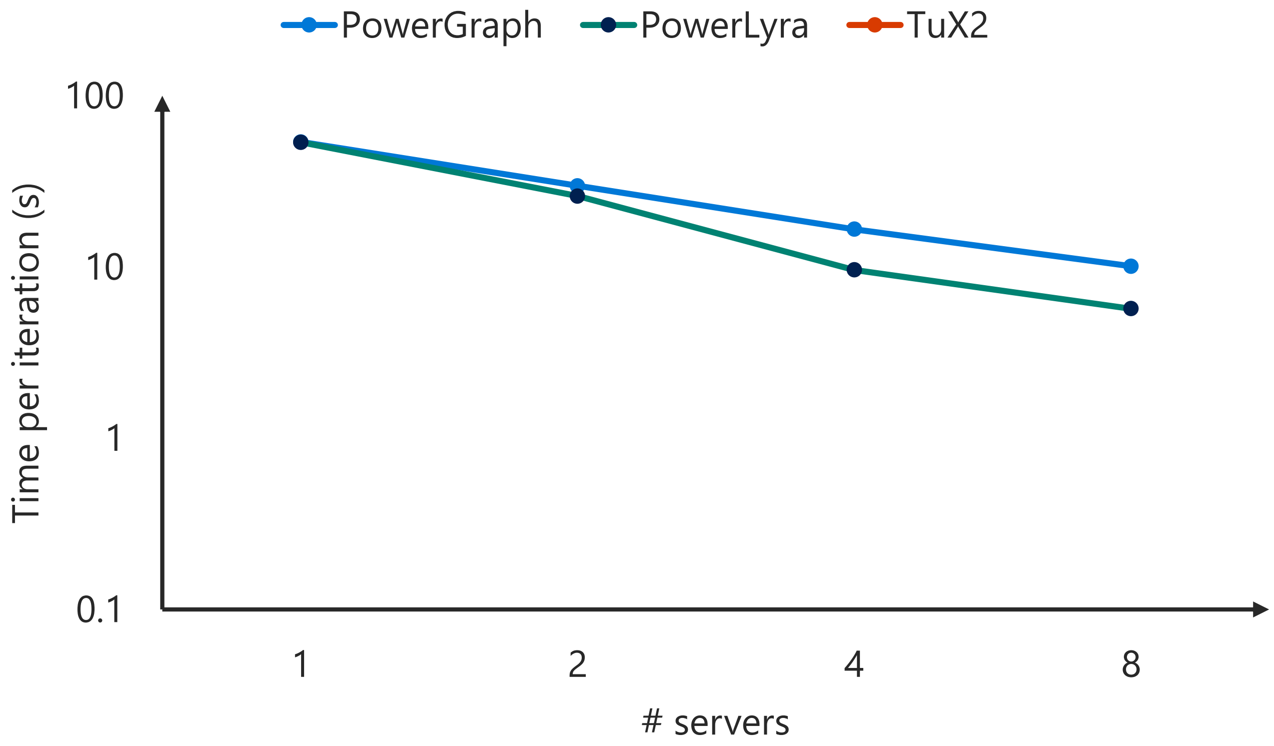


Balance in TuX2

Evaluation

Compare to PowerGraph, PowerLyra

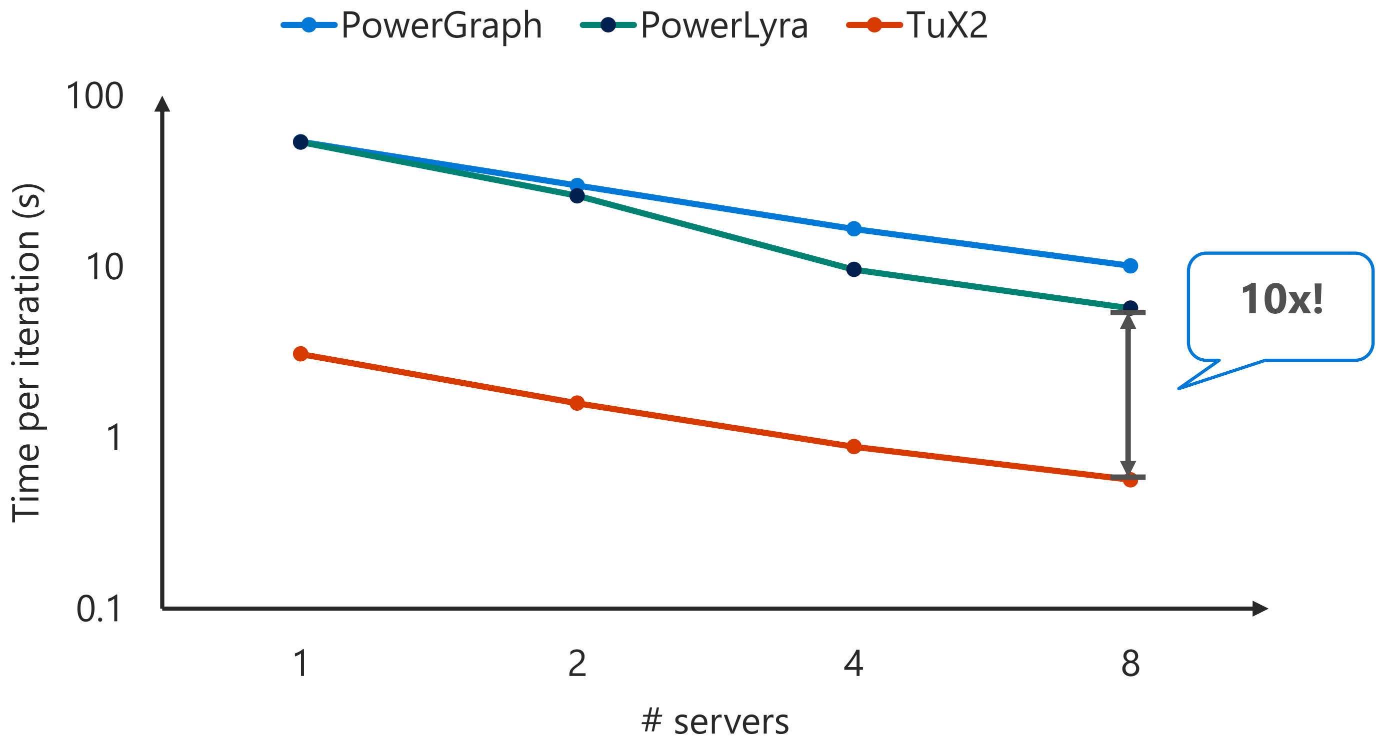
- Algorithm: Matrix Factorization
- Dataset: Netflix



Evaluation

Compare to PowerGraph, PowerLyra

- Algorithm: Matrix Factorization
- Dataset: Netflix



Conclusion

TuX^2: advocates the convergence of graph computation and distributed machine learning

- Introduce important machine learning concepts to graph computation
- Define a new, flexible graph model to express ML algorithms efficiently
- Demonstrate TuX^2 outperform existing Graph and ML systems in representative ML algorithms respectively

Conclusion

parameter server	graphML
pull, push	GAS (or MEGA)
hash-based partitioner (nearly load balance)	graph partitioner
small preprocessing cost	not sure…
server-side computation	achieved by graph partitioner
all ML models	LR, MF, LDA, etc.

Thanks!
Q&A