Gossip Algorithms for Distributed Learning

Mike Rabbat

Joint work with

Mido Assran, Nicolas Loizou, Jianyu Wang, Vinayak Tantia, and Nicolas Ballas

facebook
Artificial Intelligence Research

Data-Parallel Training in Machine Learning

- Contemporary ML involves training large models on very large datasets

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) := \frac{1}{m} \sum_{j=1}^{m} l(x; \xi_j)$$

Data-Parallel Training in Machine Learning

- Contemporary ML involves training large models on very large datasets

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) := \frac{1}{m} \sum_{j=1}^{m} l(x; \xi_j)$$

- Image classification
  - Model: ResNet-50 (25.6M params)
  - Data: ImageNet
    - 1M training instances
    - 1000 classes

- Machine translation
  - Model: Transformer (210M params)
  - Data: WMT'16 En-De
    - 4.56M sentence pairs
    - 32K vocabulary

Data-Parallel Training in Machine Learning

- Contemporary ML involves training large models on very large datasets

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) := \frac{1}{m} \sum_{j=1}^{m} l(x; \xi_j)$$

- Workhorse algorithm: Stochastic gradient descent

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)} \frac{1}{b} \sum_{j=1}^{b} \nabla l(x^{(k)}; \xi_j^{(k)})$$

- Image classification
  - Model: ResNet-50 (25.6M params)
  - Data: ImageNet
    - 1M training instances
    - 1000 classes

- Machine translation
  - Model: Transformer (210M params)
  - Data: WMT'16 En-De
    - 4.56M sentence pairs
    - 32K vocabulary

Data-Parallel Training in Machine Learning

- Contemporary ML involves training large models on very large datasets

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad f(x) := \frac{1}{m} \sum_{j=1}^{m} l(x; \xi_j)$$

- Workhorse algorithm: Stochastic gradient descent

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)} \frac{1}{b} \sum_{j=1}^{b} \nabla l(x^{(k)}; \xi_j^{(k)})$$

Data-Parallel Training:

- Exploit parallel computing to process examples in parallel

- Image classification
  - Model: ResNet-50 (25.6M params)
  - Data: ImageNet
    - 1M training instances
    - 1000 classes

- Machine translation
  - Model: Transformer (210M params)
  - Data: WMT’16 En-De
    - 4.56M sentence pairs
    - 32K vocabulary

Data-Parallel Distributed Optimization

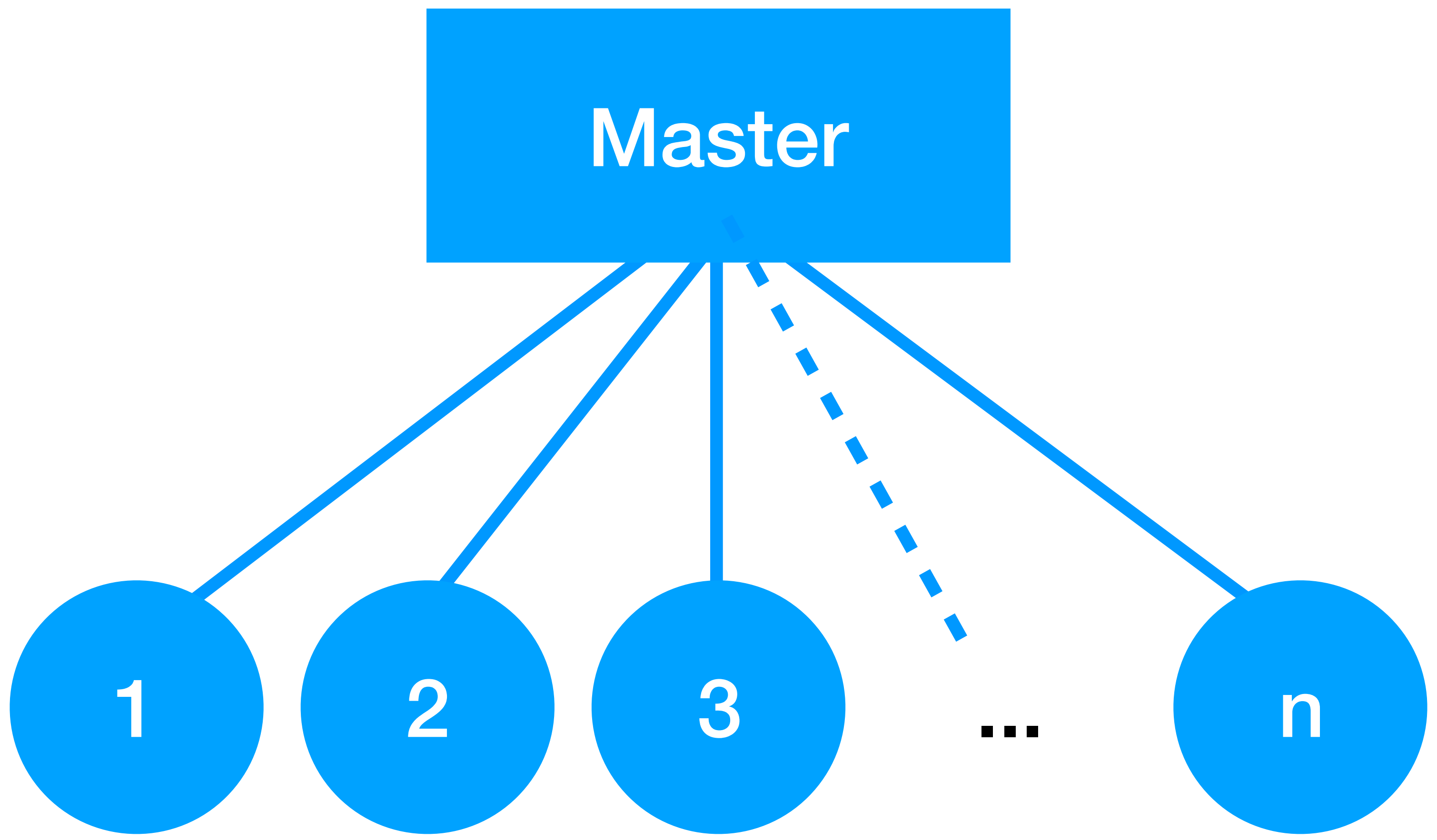
Parallelize gradient evaluation:

$$\frac{1}{b}\left(\sum_{j=1}^{b_1} \nabla l(x^{(k)}; \xi_{j,1}^{(k)}) + \cdots + \sum_{j=1}^{b_n} \nabla l(x^{(k)}; \xi_{j,1}^{(k)})\right)$$

Data-Parallel Distributed Optimization

Parallelize gradient evaluation: $$\frac{1}{b}\left(\sum_{j=1}^{b_1} \nabla l(x^{(k)}; \xi_{j,1}^{(k)}) + \cdots + \sum_{j=1}^{b_n} \nabla l(x^{(k)}; \xi_{j,1}^{(k)})\right)$$

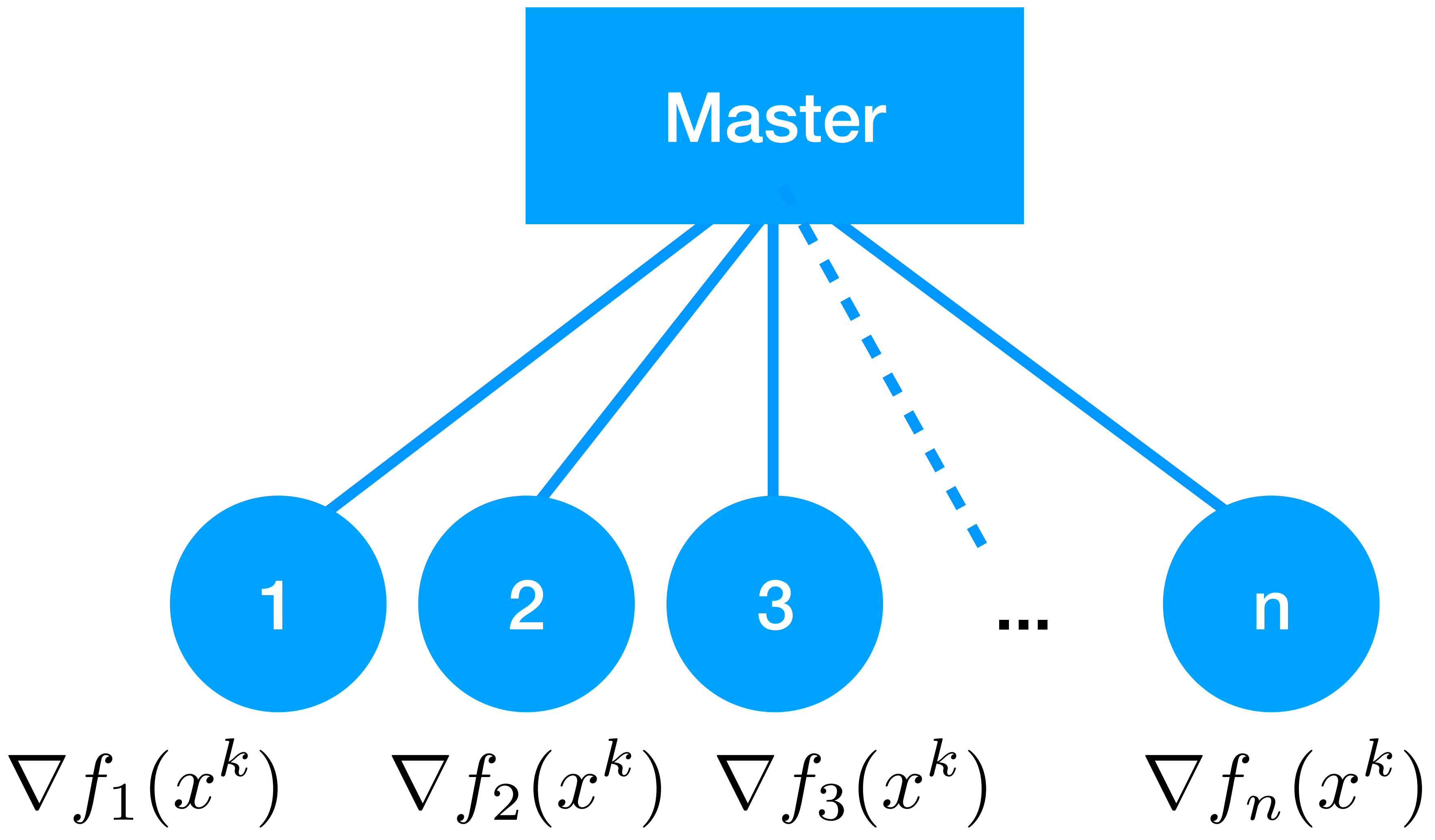
Master-Worker



Data-Parallel Distributed Optimization

Parallelize gradient evaluation: $$\frac{1}{b}\left(\sum_{j=1}^{b_1} \nabla l(x^{(k)};\xi_{j,1}^{(k)}) + \cdots + \sum_{j=1}^{b_n} \nabla l(x^{(k)};\xi_{j,1}^{(k)})\right)$$

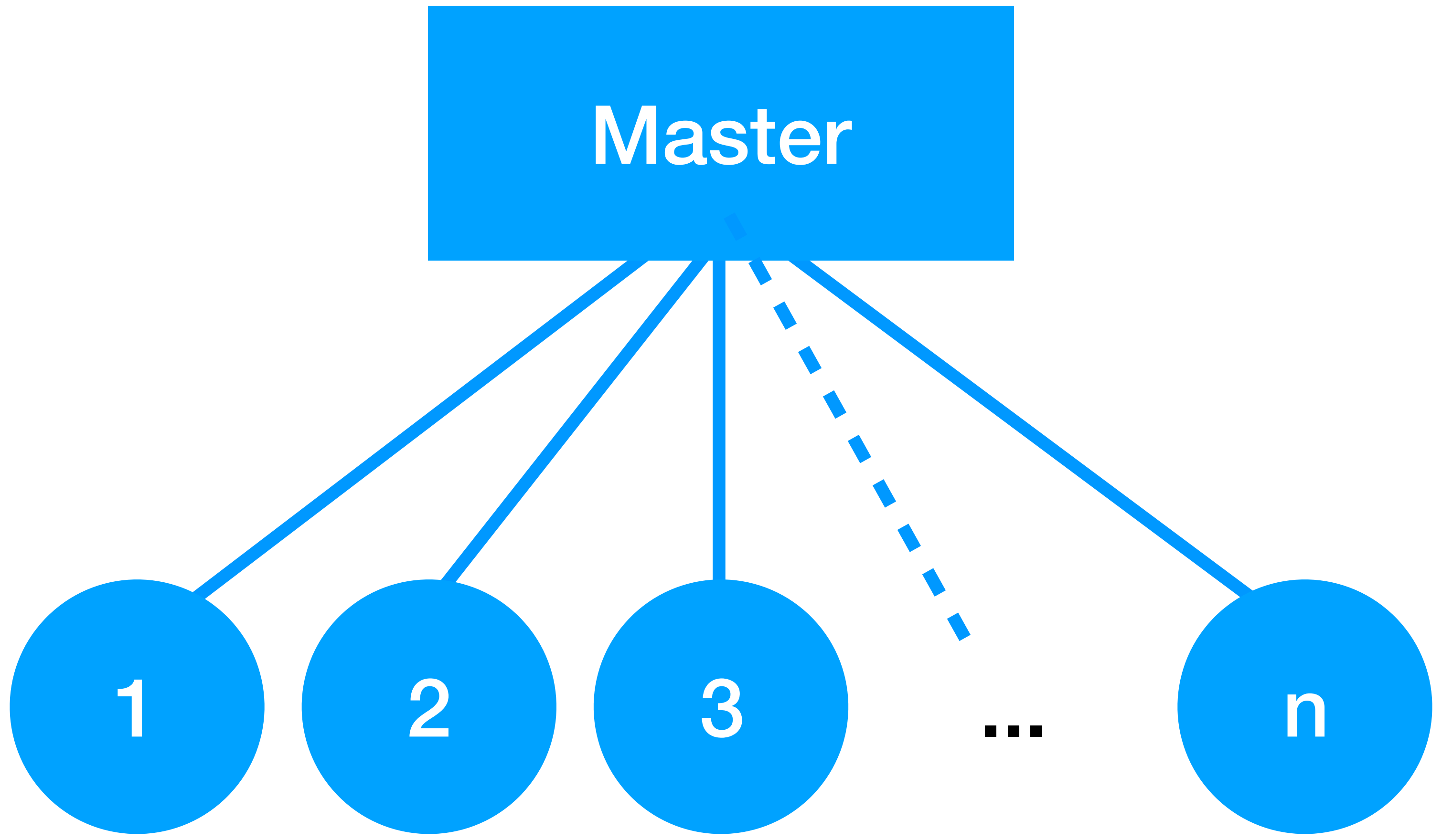
Master-Worker



Data-Parallel Distributed Optimization

Parallelize gradient evaluation: $$\frac{1}{b}\left(\sum_{j=1}^{b_1} \nabla l(x^{(k)}; \xi_{j,1}^{(k)}) + \cdots + \sum_{j=1}^{b_n} \nabla l(x^{(k)}; \xi_{j,1}^{(k)})\right)$$

Master-Worker



$\nabla f_1(x^k)$ $\nabla f_2(x^k)$ $\nabla f_3(x^k)$ $\nabla f_n(x^k)$

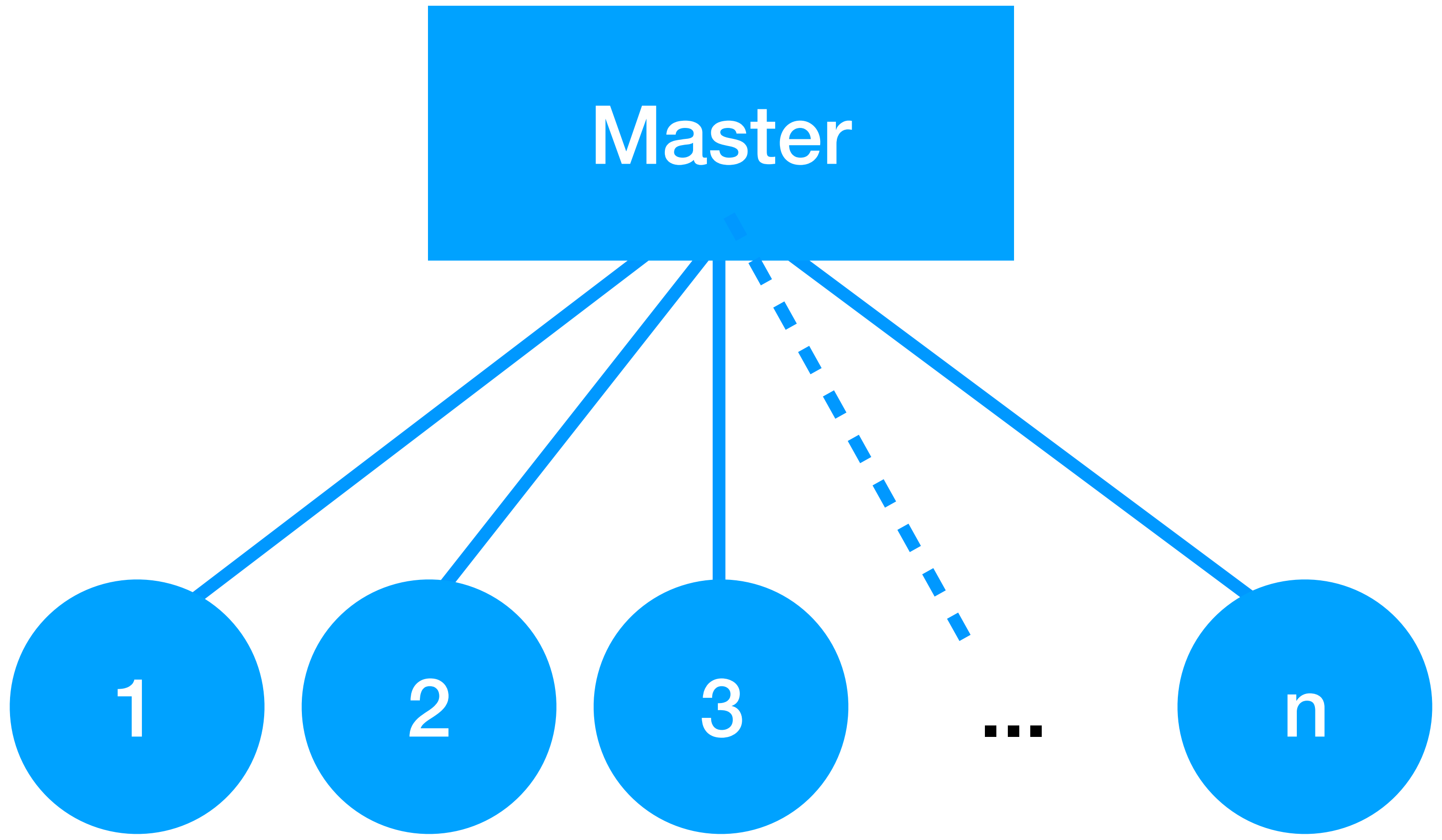
Distributed gradient descent:

$$x^{k+1} = x^k - \alpha \sum_{i=1}^{n} \nabla f_i(x^k)$$

Data-Parallel Distributed Optimization

Parallelize gradient evaluation: $$\frac{1}{b}\left(\sum_{j=1}^{b_1}\nabla l(x^{(k)};\xi_{j,1}^{(k)})+\cdots+\sum_{j=1}^{b_n}\nabla l(x^{(k)};\xi_{j,1}^{(k)})\right)$$

Master-Worker

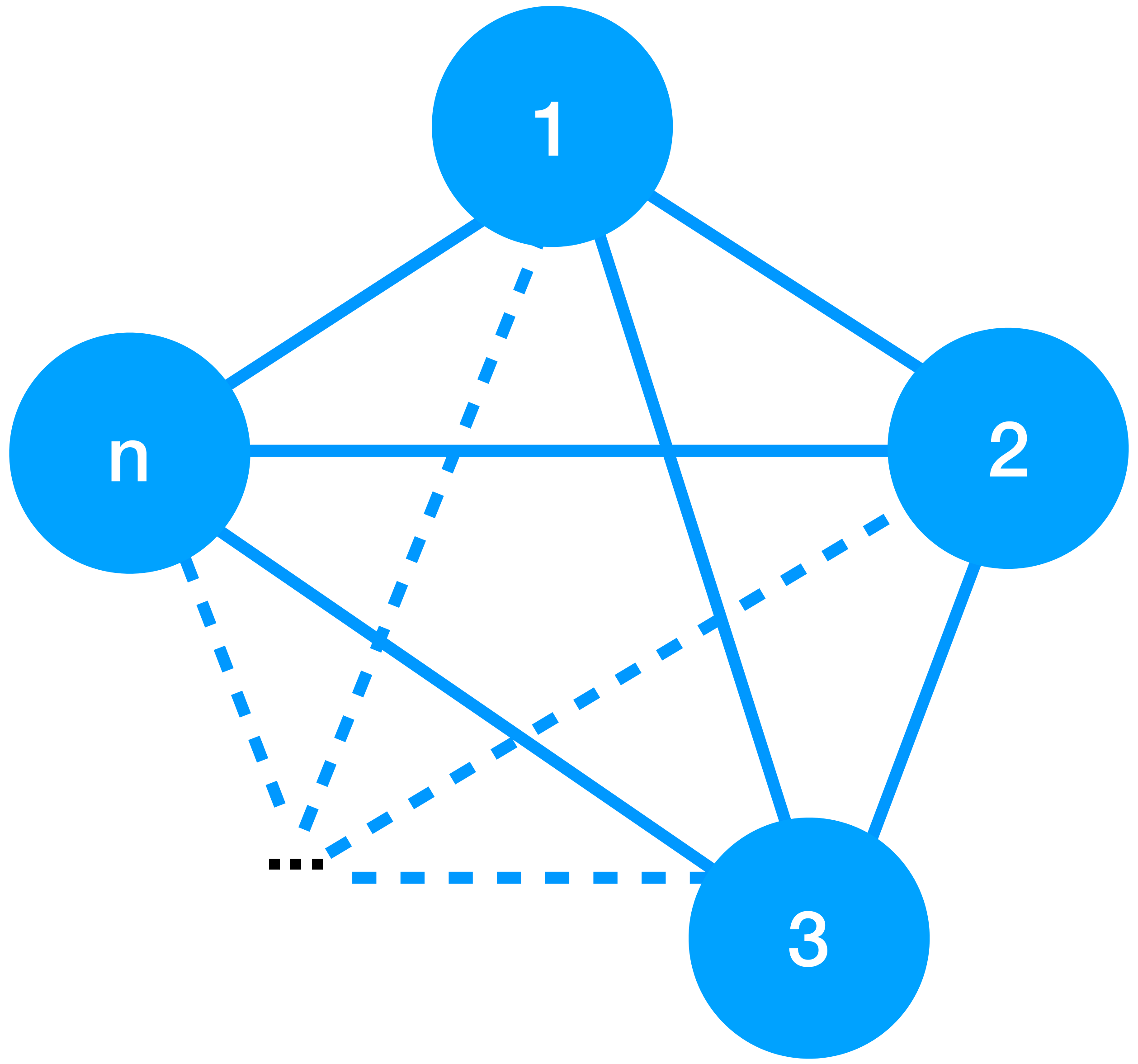


$\nabla f_1(x^k)$ $\nabla f_2(x^k)$ $\nabla f_3(x^k)$ $\nabla f_n(x^k)$

Distributed gradient descent:

$$x^{k+1}=x^k-\alpha\sum_{i=1}^{n}\nabla f_i(x^k)$$

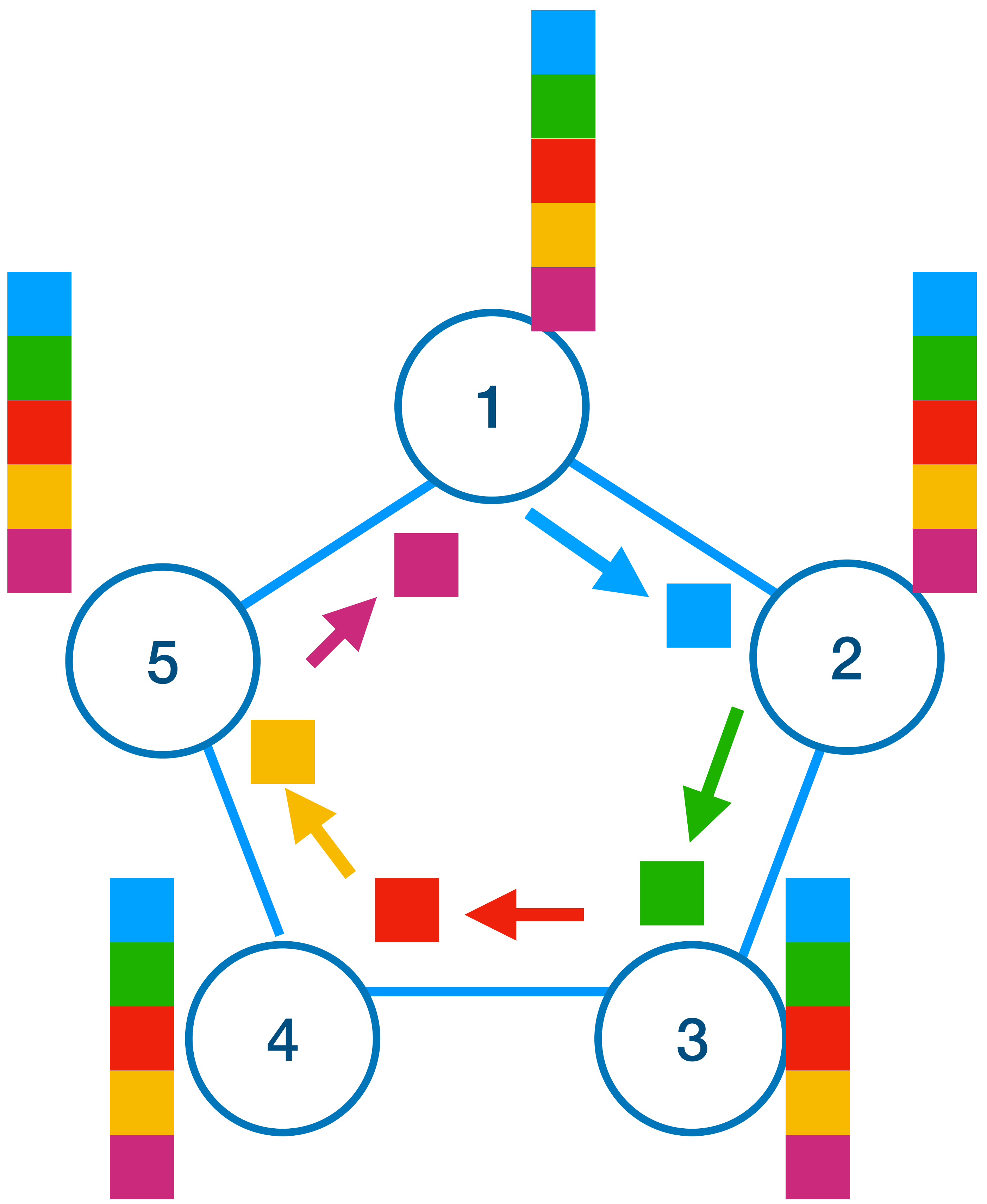
Decentralized



$$x^{k+1}=x^k-\alpha\sum_{i=1}^{n}\nabla f_i(x^k)$$

AllReduce: Exact distributed averaging

Example: Ring Algorithm

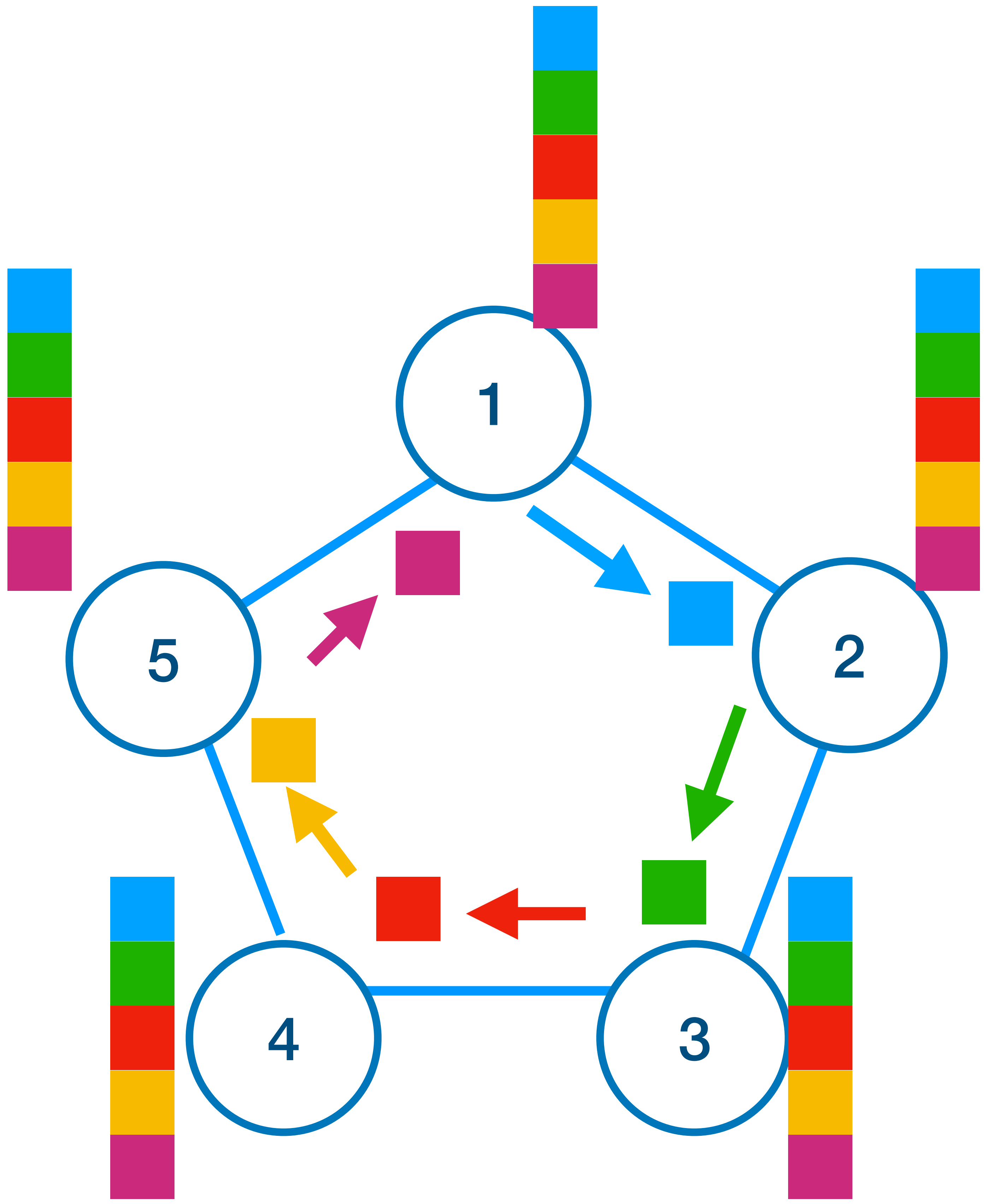


AllReduce: Exact distributed averaging

Example: Ring Algorithm

Each node sends/receives
2d values

Delay: O(n)

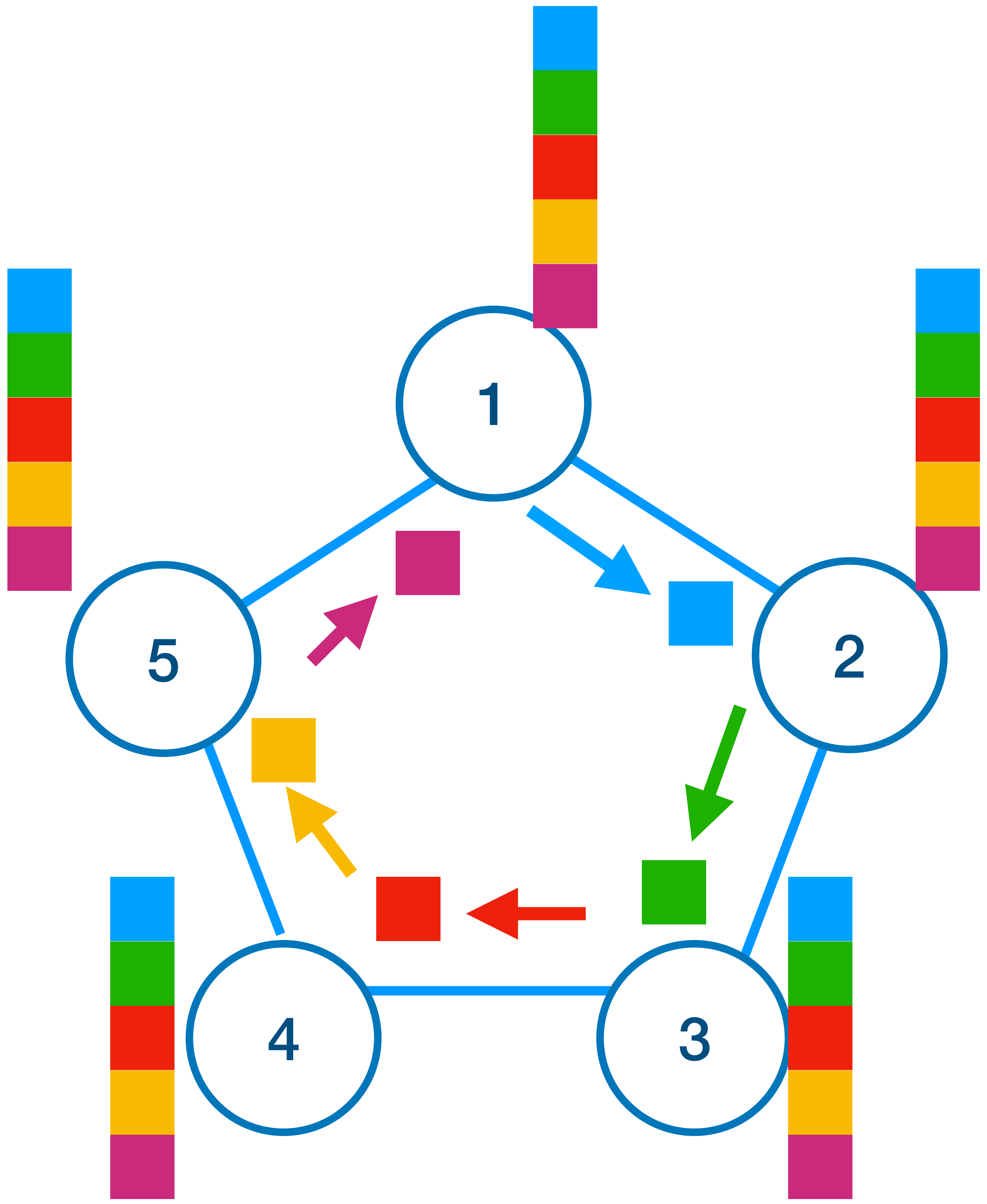


AllReduce: Exact distributed averaging

Example: Ring Algorithm

Each node sends/receives 2d values

Delay: O(n)



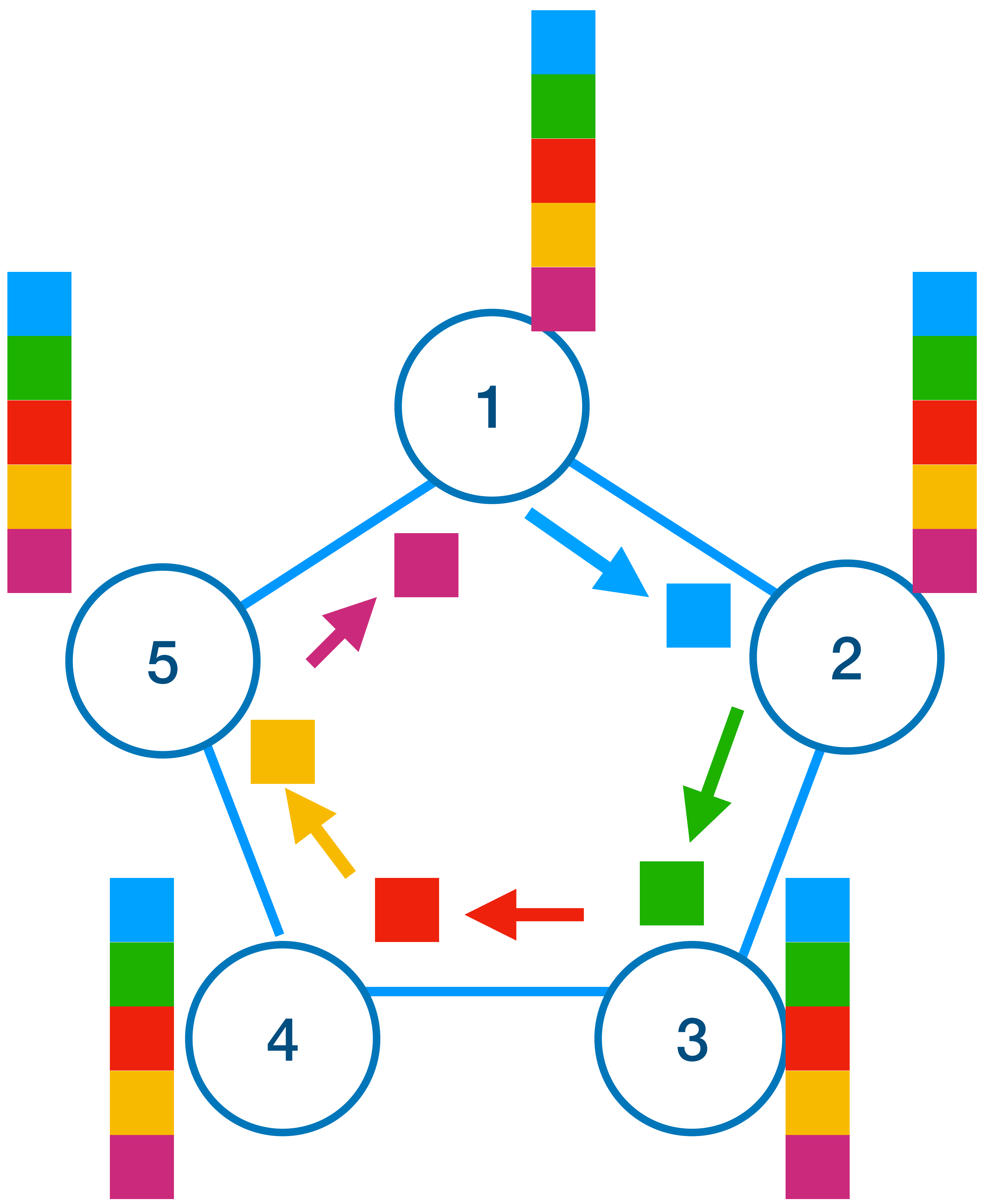
Other algorithms (spanning tree, butterfly) with delay O(log n)

AllReduce: Exact distributed averaging

Example: Ring Algorithm

Each node sends/receives 2d values

Delay: O(n)



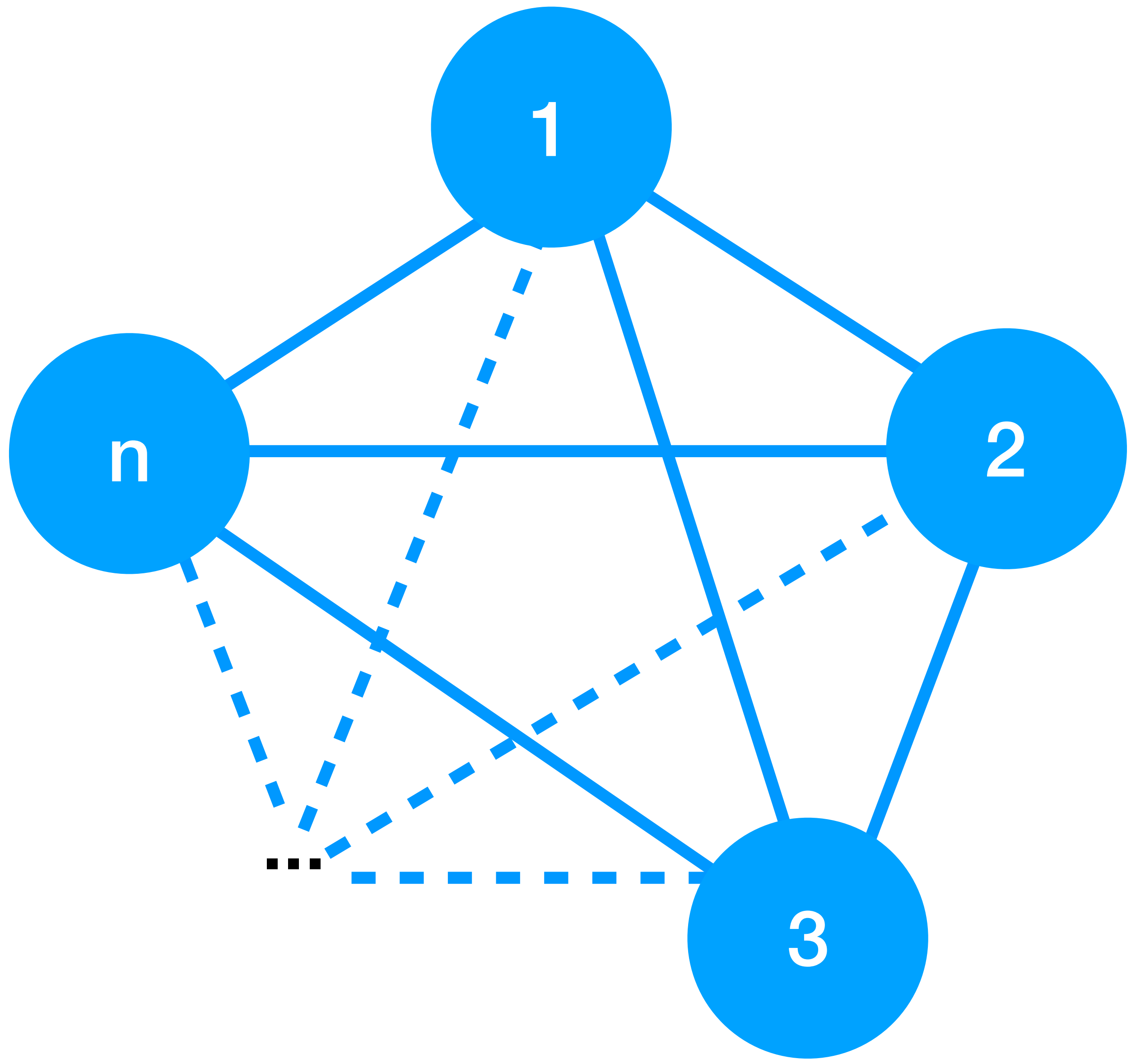
Other algorithms (spanning tree, butterfly) with delay O(log n)

Tightly coupled

Computes <u>exact</u> average of <u>any</u> inputs

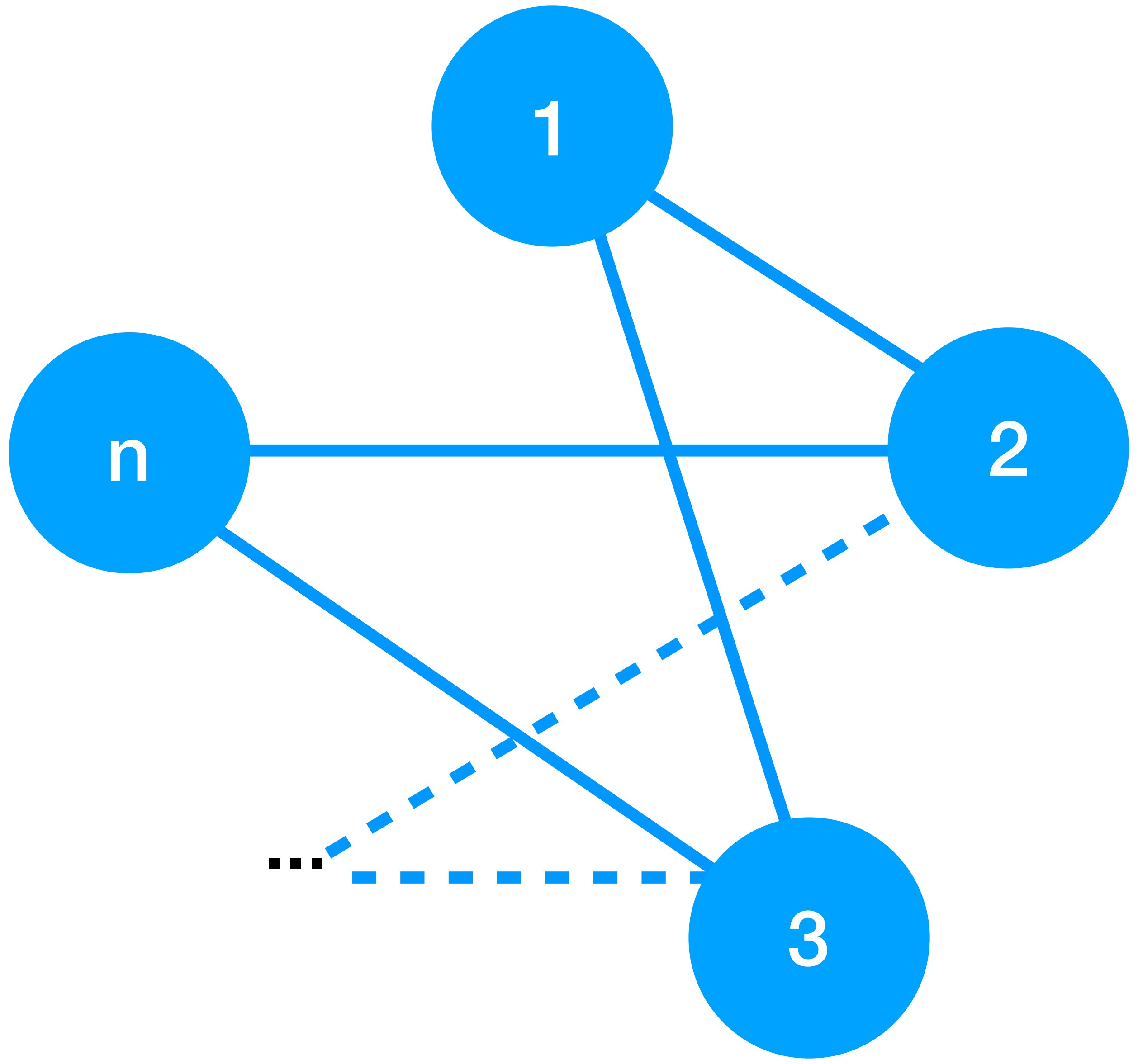
Decentralized Multi-Agent Optimization

Fully-Connected



$$x_i^{k+1} = x_i^k - \frac{\alpha^k}{n} \sum_{j=1}^{n} \nabla f_j(x_j^k)$$

Multi-Agent (aka, "gossip")



$$x_i^{k+1} = x_i^k - \frac{\alpha^k}{|\mathcal{N}_i^k|} \sum_{j \in \mathcal{N}_i^k} \nabla f_j(x_j^k)$$

This Talk

Synchronous methods build on AllReduce have problems:

- Move at the pace of the slowest node
- Sensitive to delay variations

Aspirations:

- Decouple communications to be less sensitive (asynchronous)
- Ultimately, run faster and be more resource-efficient

Contributions:

- Analysis of stochastic gradient push for non-convex functions
- Demonstration of stochastic gradient-push for training deep networks

Brief Historical Perspective:

1986 Tsitsiklis, Bertsekas, & Athans

- Synchronous and asynchronous block coordinate descent
- All agents know the global objective

2003 Kempe, Dobra, & Gehrke

- Push-sum distributed averaging
- Fully-connected, randomized activations

2009 Nedic & Ozdaglar

- Synchronous multi-agent gradient descent

2012 Tsianos & Rabbat

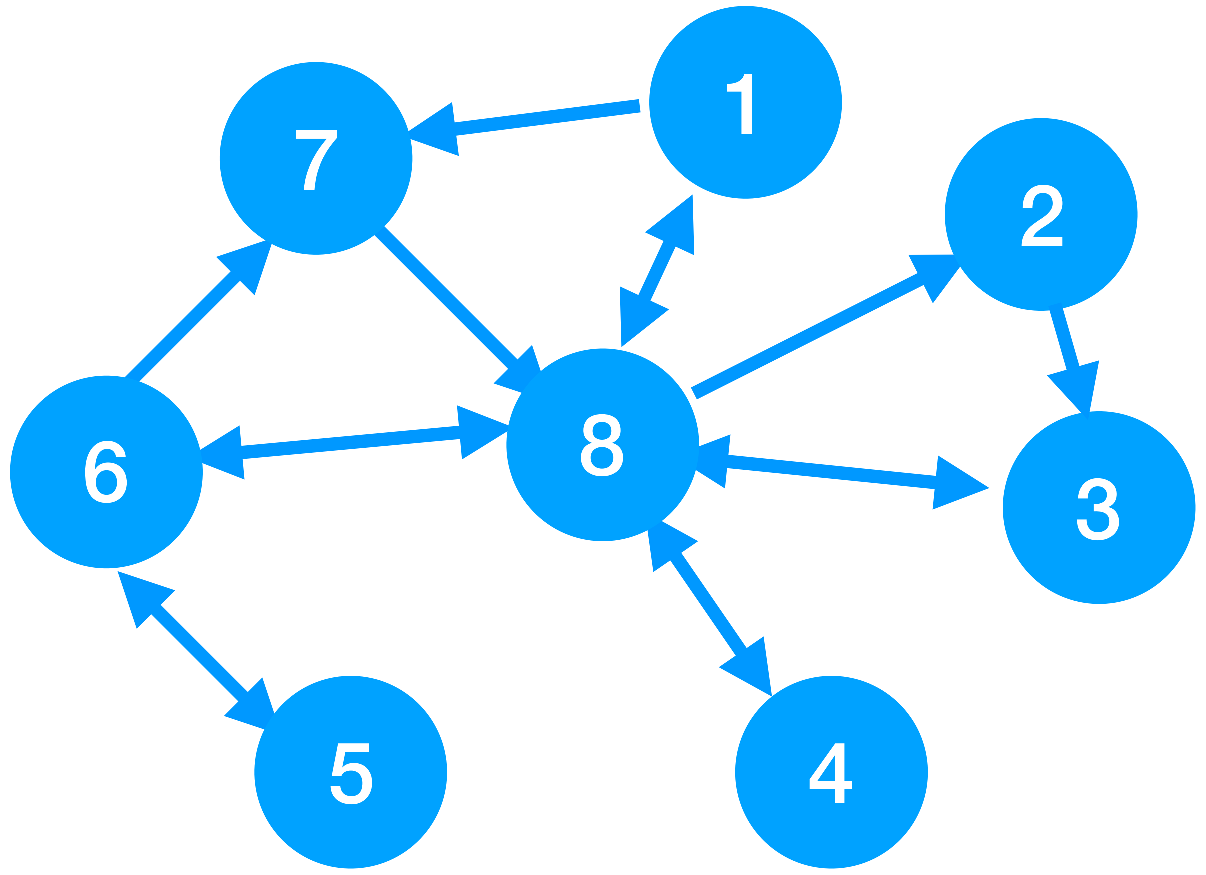
- Push-sum distributed dual averaging

2014-18 Nedic & Olshevsky; Zeng & Yin; Nedic, Olshevsky, & Shi

- Faster, synchronous, push-sum-based optimization

Survey:
A Nedic, A Olshevsky, & M Rabbat
Proceedings of the IEEE, May 2018

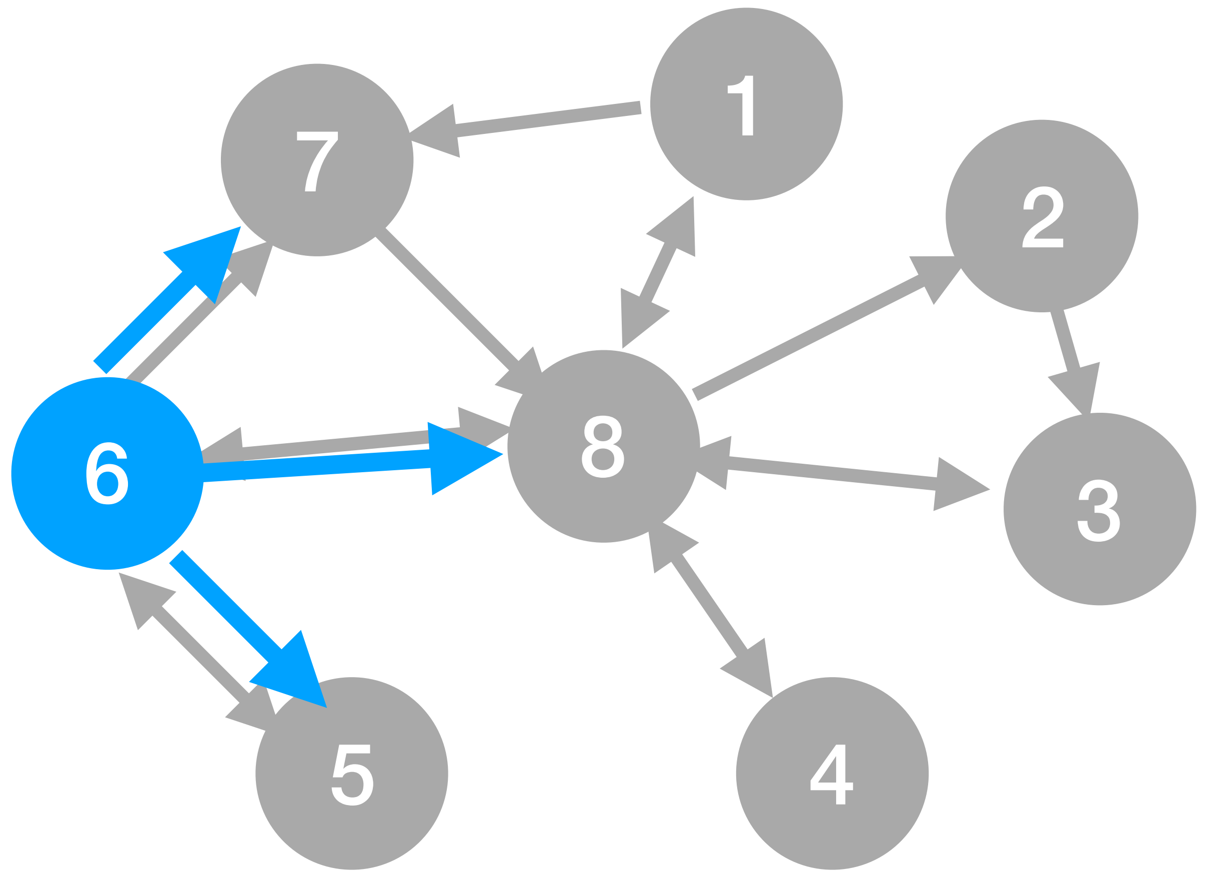
Distributed Averaging over Directed Graphs



Problem: All nodes have an initial value $x_i^{(0)}$ and they should all approximately compute the average $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i^{(0)}$

Design choice: Only use push-type communication

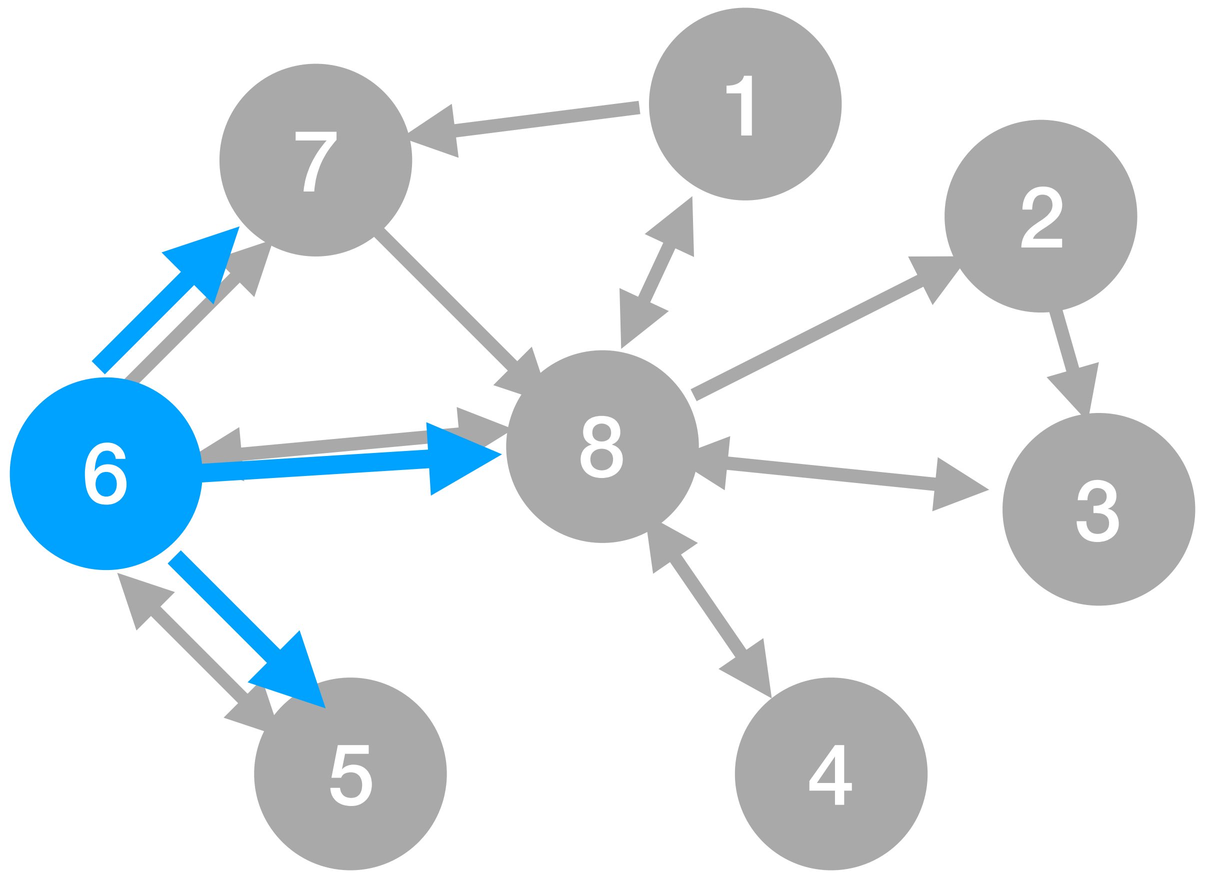
Distributed Averaging over Directed Graphs



Problem: All nodes have an initial value $x_i^{(0)}$ and they should all approximately compute the average $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i^{(0)}$

Design choice: Only use **push**-type communication

Distributed Averaging over Directed Graphs



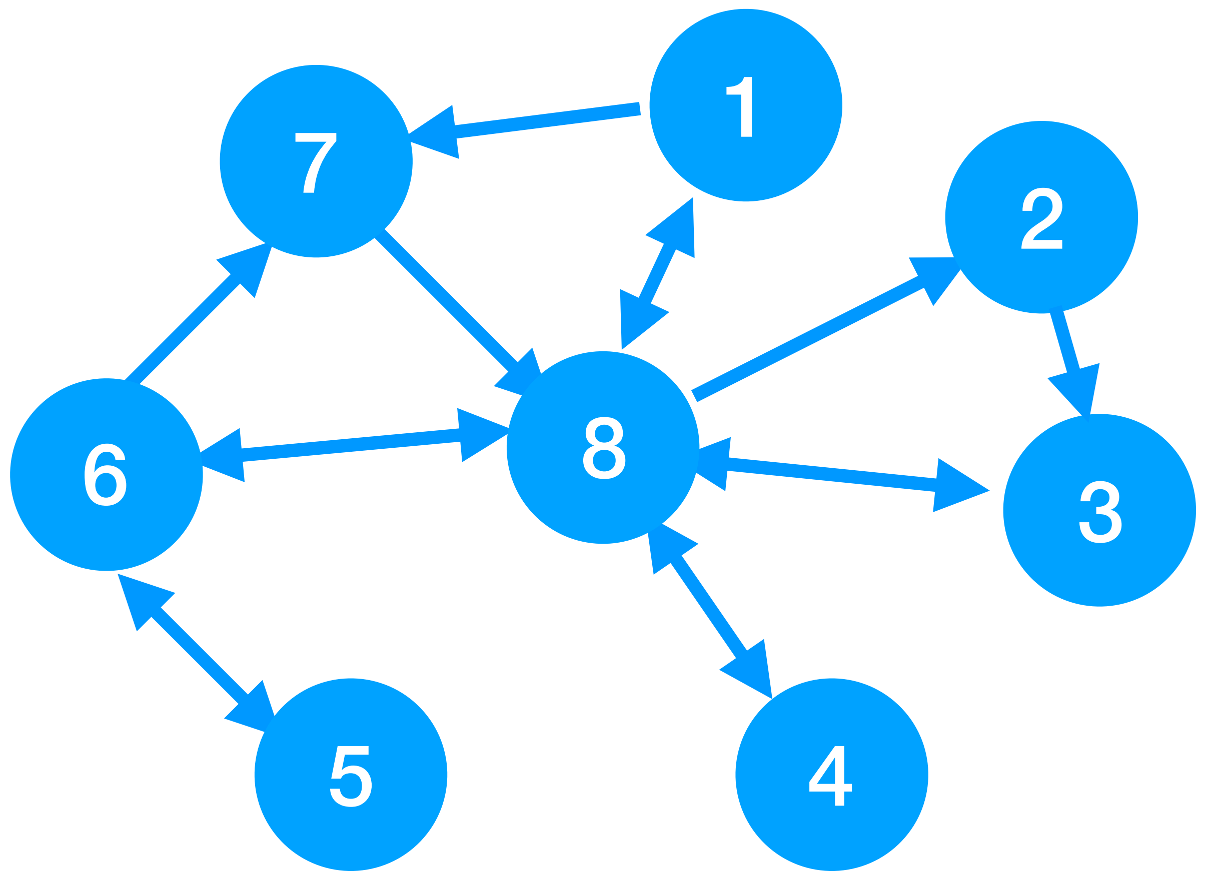
Problem: All nodes have an initial value $x_i^{(0)}$ and they should all approximately compute the average $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i^{(0)}$

Design choice: Only use **push**-type communication

Why?

- Asynchronous operation → directed communication
- Easy to implement
- Amenable to analysis

Distributed Averaging over Directed Graphs



Distributed implementation as linear iterations

$$x_i^{(k)} = \sum_j P_{i,j} x_j^{(k-1)} = \sum_{j \in N_i^{\text{in}}} P_{i,j} x_j^{(k-1)}$$

Column stochastic matrix P

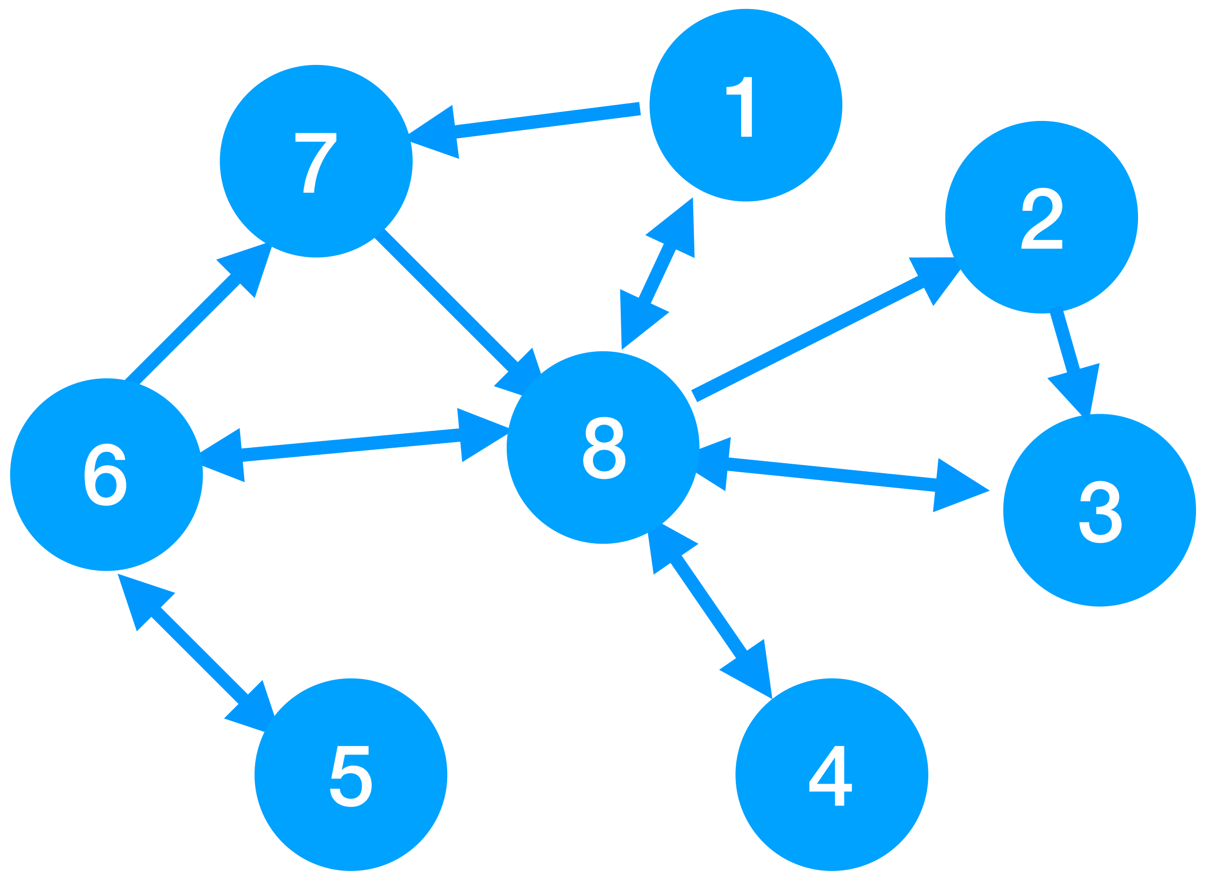
$$P_{i,j} > 0 \Leftrightarrow (j \to i) \in E$$

Products converge (Perron-Frobenius):

$$\lim_{k \to \infty} P^k = \boldsymbol{\pi} \mathbf{1}^T$$

$$\lim_{k \to \infty} P^k \boldsymbol{x} = \boldsymbol{\pi} \left(\mathbf{1}^T \boldsymbol{x} \right)$$

Distributed Averaging over Directed Graphs



Column stochastic matrix P

$$P_{i,j} > 0 \Leftrightarrow (j \to i) \in E$$

Products converge (Perron-Frobenius):

$$\lim_{k\to\infty} P^k = \boldsymbol{\pi}\mathbf{1}^T$$

$$\lim_{k\to\infty} P^k \boldsymbol{x} = \boldsymbol{\pi}\left(\mathbf{1}^T\boldsymbol{x}\right)$$

Distributed implementation as linear iterations

$$x_i^{(k)} = \sum_j P_{i,j} x_j^{(k-1)} = \sum_{j \in N_i^{\text{in}}} P_{i,j} x_j^{(k-1)}$$

Push-Sum Algorithm [Kempe, Dobra, Gehrke 2003]

Initialize $x_i[0] \in \mathbb{R}, w_i[0] = 1$

Implement linear iterations via distributed message passing

$$\boldsymbol{x}[k] = P\boldsymbol{x}[k-1] = P^k\boldsymbol{x}[0] \quad \to \quad \boldsymbol{\pi}\left(\mathbf{1}^T\boldsymbol{x}[0]\right)$$

$$\boldsymbol{w}[k] = P\boldsymbol{w}[k-1] = P^k\boldsymbol{w}[0] \quad \to \quad n\boldsymbol{\pi}$$

$$z_i[k] = x_i[k]/w_i[k] \quad \to \quad (\mathbf{1}^T\boldsymbol{x}[0])/n$$

Stochastic Gradient-Push [Nedic & Olshevsky, 2016]

Node *i* initializes $x_i^{(0)} = z_i^{(0)} \in \mathbb{R}^d \quad \forall i$ and $w_i^{(0)} = 1$

For iterations k=0,1,...,K at node i:

- Sample new mini-batch gradient $\nabla F_i(z_i^{(k)}, \xi_i^{(k)})$
- Update

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \alpha \nabla F_i(z_i^{(k)}, \xi_i^{(k)})$$

Gradient descent locally

- Send/receive messages

$$\left(P_{i,j}^{(k)} x_j^{(k+\frac{1}{2})},\ P_{i,j}^{(k)} w_j^{(k+\frac{1}{2})}\right)$$

Push-Sum averaging

and aggregate

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i^{\text{in}(k)}} P_{i,j}^{(k)} x_j^{(k+\frac{1}{2})}$$

$$w_i^{(k+1)} = \sum_{j \in \mathcal{N}_i^{\text{in}(k)}} P_{i,j}^{(k)} w_j^{(k+\frac{1}{2})}$$

$$z_i^{(k+1)} = x_i^{(k+1)} / w_i^{(k+1)}$$

Stochastic Gradient-Push [Nedic & Olshevsky, 2016]

Node *i* initializes $x_i^{(0)} = z_i^{(0)} \in \mathbb{R}^d \quad \forall i$ and $w_i^{(0)} = 1$

For iterations k=0,1,...,K at node i:

- Sample new mini-batch gradient $\nabla F_i(z_i^{(k)}, \xi_i^{(k)})$
- Update $x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \alpha \nabla F_i(z_i^{(k)}, \xi_i^{(k)})$

Gradient descent locally

- Send/receive messages

$$\left(P_{i,j}^{(k)} x_j^{(k+\frac{1}{2})},\ P_{i,j}^{(k)} w_j^{(k+\frac{1}{2})}\right)$$

and aggregate

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i^{\mathrm{in}(k)}} P_{i,j}^{(k)} x_j^{(k+\frac{1}{2})}$$

$$w_i^{(k+1)} = \sum_{j \in \mathcal{N}_i^{\mathrm{in}(k)}} P_{i,j}^{(k)} w_j^{(k+\frac{1}{2})}$$

$$z_i^{(k+1)} = x_i^{(k+1)} / w_i^{(k+1)}$$

Push-Sum averaging

τ – overlap SGP

Semi-synchronous variant

Gossip and update in separate threads,

Can be up to τ steps out of sync

Convergence guarantees

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\xi_i}\left[F_i(x_i;\xi_i)\right]$$
$$\text{subject to} \quad x_i = x_j, \forall (i,j) \in E$$

Let $f_i(x) = \mathbb{E}_{\xi_i}\left[F_i(x_i;\xi_i)\right]$ and suppose that

1. L-smooth functions $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x-y\|$
2. Bounded variance $\mathbb{E}_{\xi_i}\|\nabla F_i(x_i;\xi_i) - \nabla f_i(x)\|^2 \leq \sigma^2$
3. Similar objectives $\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x) - \nabla f(x)\|^2 \leq \zeta^2$ Bijral, Sarwate, Srebro (2017)
4. Communication topologies are *B*-strongly connected

$\bigcup_{k=lB}^{(l+1)B-1} E^{(k)}$ strongly connected, where $E^{(k)} = \{(i,j) \colon P_{i,j}^{(k)} > 0\}$

with diameter Δ

Theorem. Run SGP for K iterations with $\alpha = \sqrt{n/K}$.
There exist constants $C > 0$, $q \in [0, 1)$, P_1 and P_2 that depend on Δ, $(P^{(k)})$, and τ, such that if

$$K \geq \max\left\{ \frac{nL^4C^4 60^2}{(1-q)^4}, \frac{nL^4C^4P_1^2}{(1-q)^4(f(\bar{x}^{(0)}) - f^* + \frac{L\sigma^2}{2})^2}, \frac{nL^2C^2P_2}{(1-q)^2(f(\bar{x}^{(0)}) - f^* + \frac{L\sigma^2}{2})}, n \right\}$$

then

$$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}\left\|\nabla f(\overline{x}^{(k)})\right\|^2 \leq \frac{12\left(f(\overline{x}^{(0)}) - f^* + \frac{L\sigma^2}{2}\right)}{\sqrt{nK}} .$$

M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic Gradient Push for Distributed Deep Learning," ICML 2019.

Related result for push-pull: Lian, Zhang, Zhang, Hsieh, Zhang, and Liu, "Can Decentralized Algorithms Outperform Centralized Algorithms?" NeurIPS 2017.

Theorem. Choose K sufficiently large and use $\alpha = \sqrt{n/K}$. Then

$$\frac{1}{nK} \sum_{k=0}^{K-1} \sum_{i=1}^{n} \mathbb{E} \left\| \overline{x}^{(k)} - z_i^{(k)} \right\|^2 \leq \mathcal{O}\left(\frac{1}{K} + \frac{1}{K^{2/3}} \right)$$

and

$$\frac{1}{nK} \sum_{k=0}^{K-1} \sum_{i=1}^{n} \mathbb{E} \left\| \nabla f(z_k^{(k)}) \right\|^2 \leq \mathcal{O}\left(\frac{1}{\sqrt{nK}} + \frac{1}{K} + \frac{1}{K^{2/3}} \right).$$

M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic Gradient Push for Distributed Deep Learning," ICML 2019.

Experimental Evaluation

- Training ResNet50 (25.6M parameters) on ImageNet
- System: 32 NVIDIA DGX-1 servers (8 GPUs/server)
  - Look at scaling from 4 - 32 servers (32 - 256 GPUs)
- Communicating over either 10Gbps Ethernet or 100Gbps InfiniBand

- All implemented in PyTorch 0.4, wraps MPI and NCCL

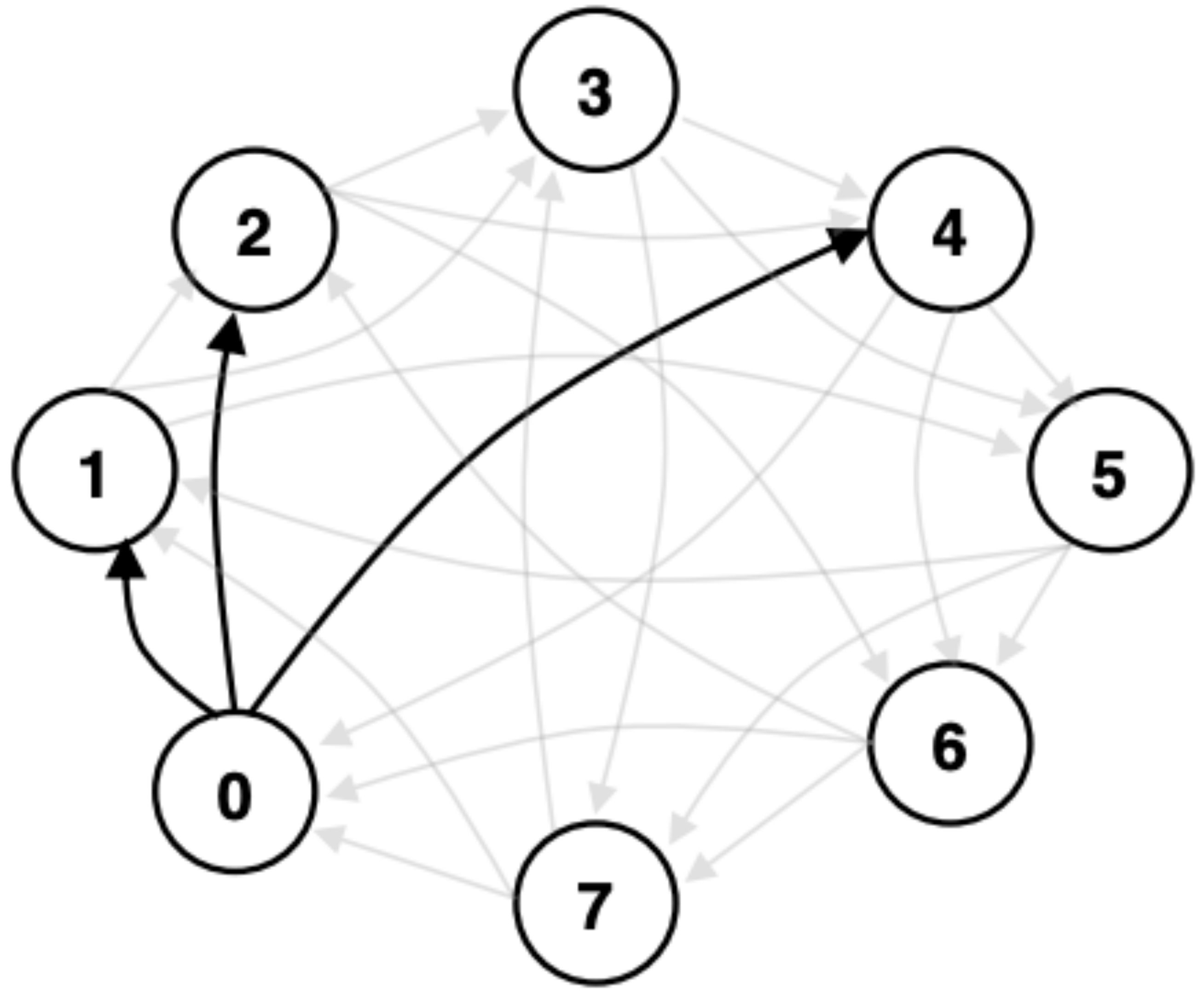
- Comparison with baselines:
  - AllReduce-based SGD
  - D-PSGD, AD-PSGD decentralized push-pull stochastic gradient methods [Lian, Zhang, Zhang, Hsieh, Zhang, and Liu, NeurIPS 2017, ICML 2018]

Directed Exponential Communication Strategy

Cyclic over edges in the binary hypercube

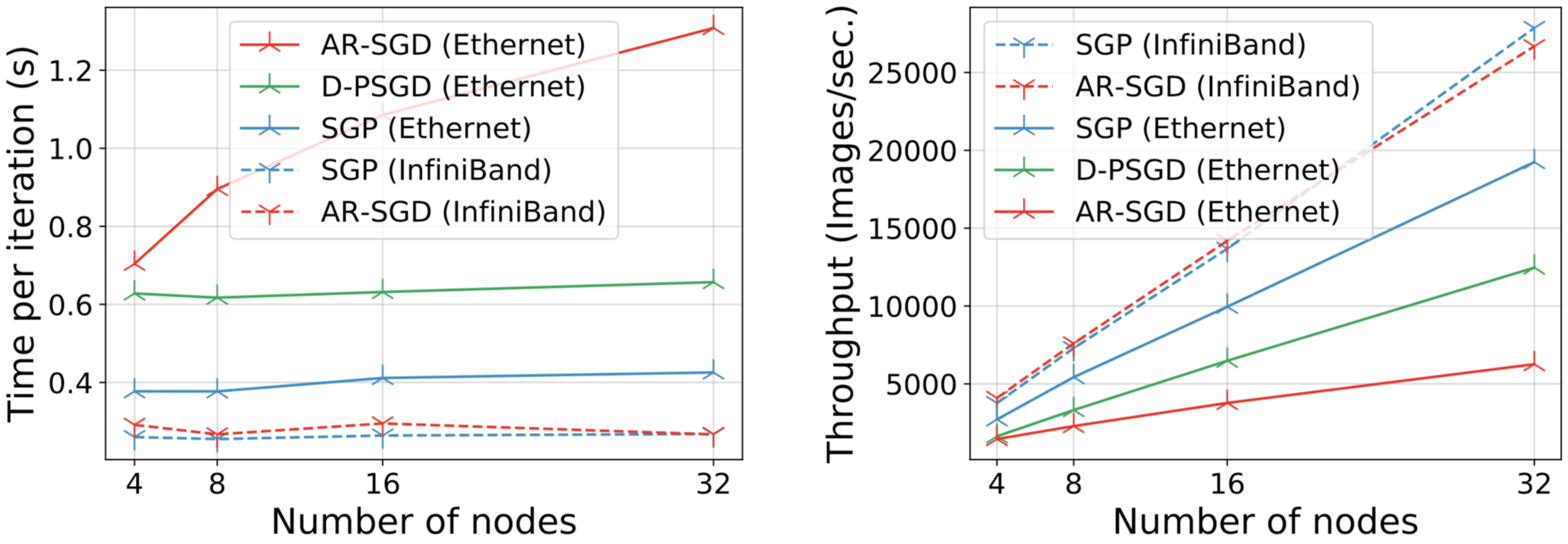
- Each node sends and receives **one** message per update
- Node i sends to

$$i + 2^0 \mod n$$
$$i + 2^1 \mod n$$
$$\vdots$$
$$i + 2^{\lfloor \log_2(n-1) \rfloor} \mod n$$



Interesting properties:

- Balanced communication workload
- For only averaging (no optimization), all nodes exactly have the average after $log_2(n)$ steps


AR-SGD (Ethernet)
D-PSGD (Ethernet)
SGP (Ethernet)
SGP (InfiniBand)
AR-SGD (InfiniBand)
Time per iteration (s)
1.2
1.0
0.8
0.6
0.4
4
8
16
32
Number of nodes
SGP (InfiniBand)
AR-SGD (InfiniBand)
SGP (Ethernet)
D-PSGD (Ethernet)
AR-SGD (Ethernet)
Throughput (Images/sec.)
25000
20000
15000
10000
5000
4
8
16
32
Number of nodes

Experiment Results

- SGP and OSGP are faster per iteration, but introduce additional noise
- Improve accuracy by running for more epochs

	Train Acc.	Val. Acc.	Train Time
AR-SGD	76.9%	76.2%	5.1 hrs. (90 epochs)
AD-PSGD	80.3%	76.9%	4.7 hrs. (270 epochs)
SGP	75.6%	74.9%	1.5 hrs. (90 epochs)
SGP	80.0%	77.1%	4.6 hrs. (270 epochs)
1-OSGP	81.8%	77.1%	2.7 hrs. (270 epochs)

32 nodes (256 GPUs), over 10Gbps Ethernet

Slow Momentum (SlowMo) Improves Convergence

Algorithm 1: Slow Momentum

Input: Base optimizer with learning rate γ_t; Inner loop steps τ; Slow learning rate α; Slow momentum factor β; Number of worker nodes m. Initial point $\boldsymbol{x}_{0,0}$ and initial slow momentum buffer $\boldsymbol{u}_0 = \mathbf{0}$.

for $t \in \{0, 1, \ldots, T-1\}$ **at worker** i **in parallel do**

  Reset/maintain/average base optimizer buffers

  for $k \in \{0, 1, \ldots, \tau - 1\}$ **do**

    Base optimizer step: $\boldsymbol{x}_{t,k+1}^{(i)} = \boldsymbol{x}_{t,k}^{(i)} - \gamma_t \boldsymbol{d}_{t,k}^{(i)}$

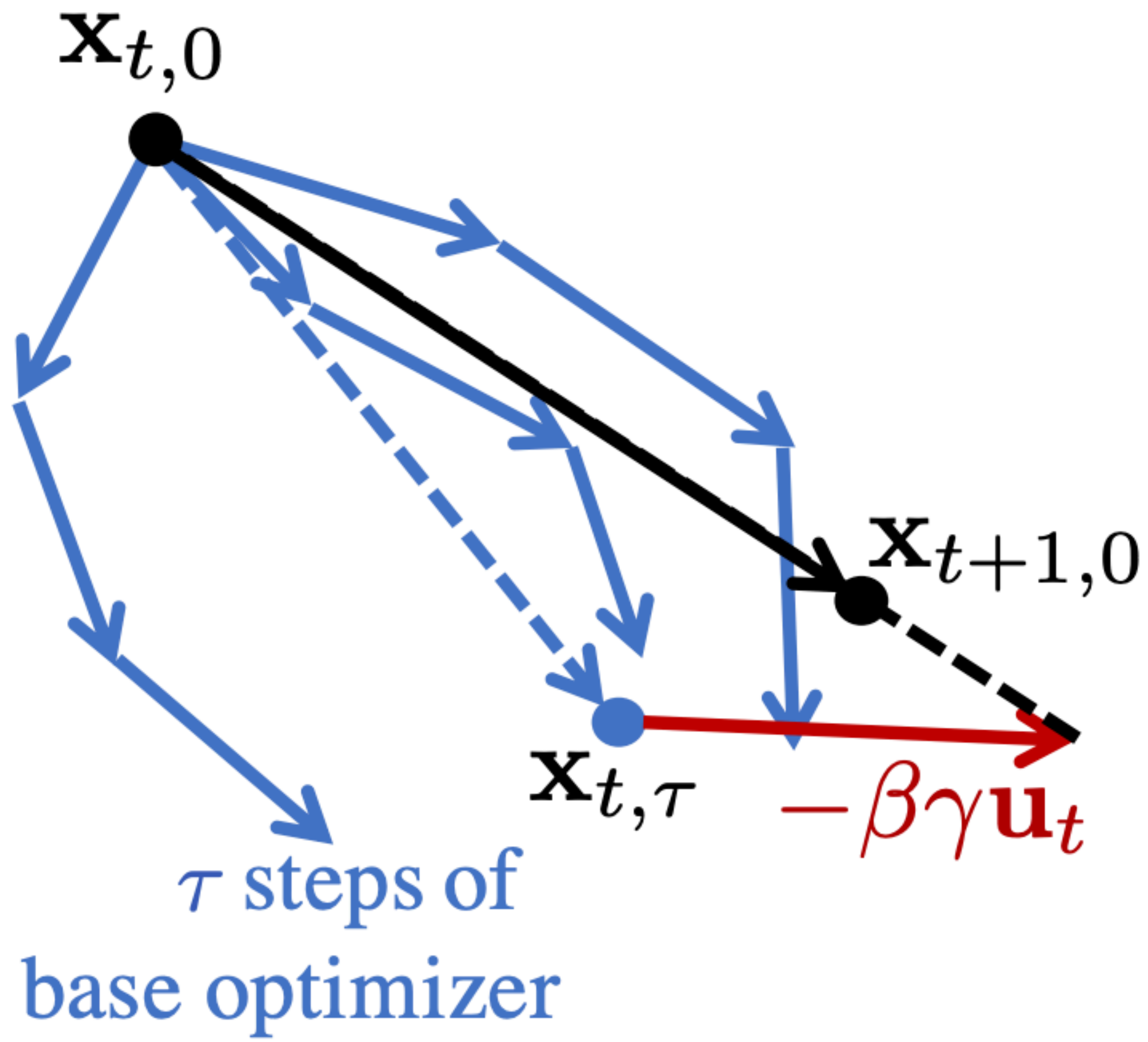
  end

  Exact-Average: $\boldsymbol{x}_{t,\tau} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_{t,\tau}^{(i)}$

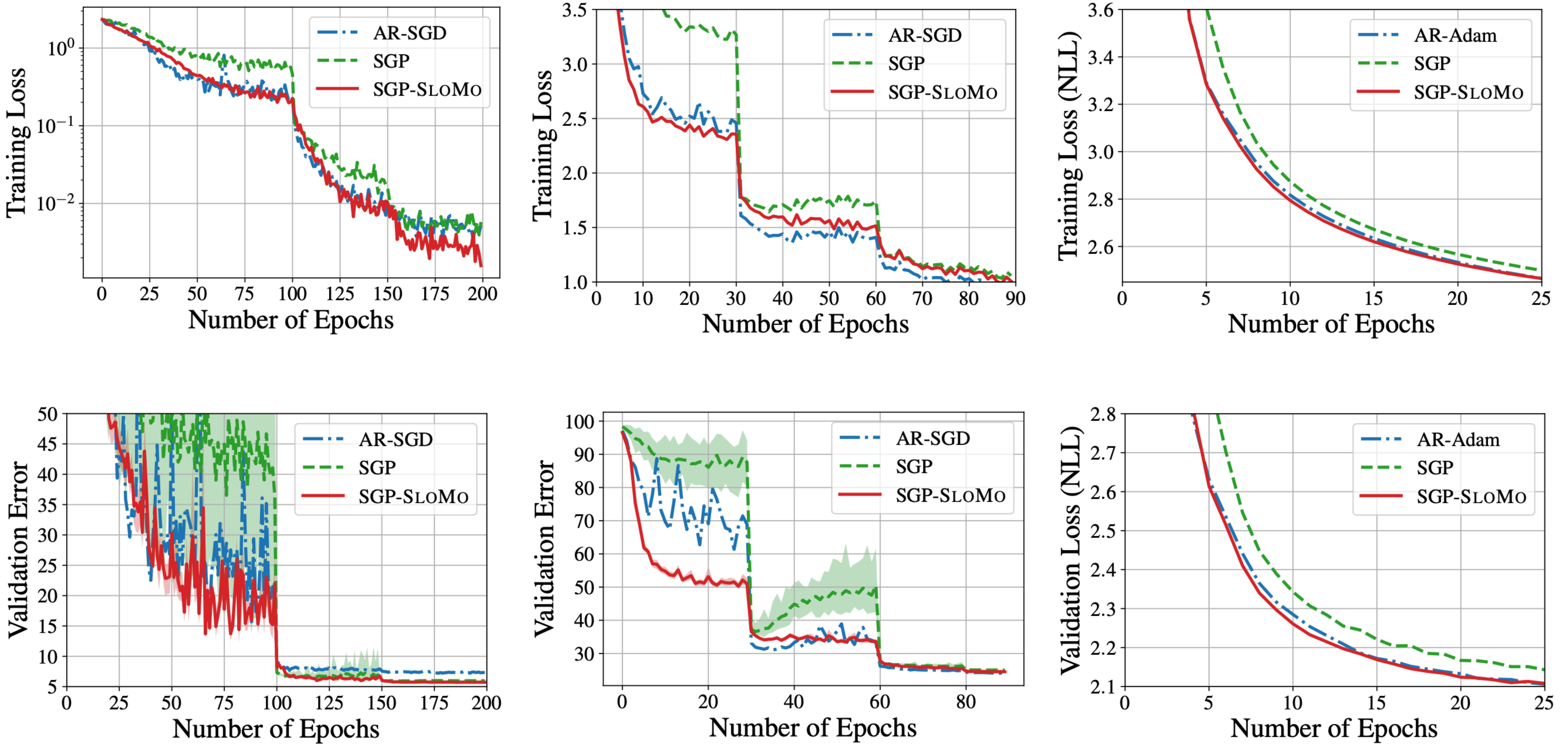
  Update slow momentum: $\boldsymbol{u}_{t+1} = \beta \boldsymbol{u}_t + \frac{1}{\gamma_t}(\boldsymbol{x}_{t,0} - \boldsymbol{x}_{t,\tau})$

  Update outer iterates: $\boldsymbol{x}_{t+1,0} = \boldsymbol{x}_{t,0} - \alpha\gamma_t \boldsymbol{u}_{t+1}$

end



J. Wang, V. Tantia, N. Ballas, and M. Rabbat, "SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum," October 2019. https://arXiv.org/abs/1910.00643,



(a) CIFAR-10, batch size:4k. (b) ImageNet, batch size:8k. (c) WMT16 En-De, batch size:200k.

Summary

Push-only communication makes a difference

- Model non-blocking asynchronous communication
- Communication and computation delays

Ongoing work and extensions

- Quantization, less frequent aggregation
- Improving accuracy with momentum

SGP Paper online at arxiv.org/abs/1811.10792

SlowMo paper at arxiv.org/abs/1910.00643

Code online at github.com/facebookresearch/stochastic_gradient_push

mikerabbat@fb.com