Tutorial: Part 2

Security and Privacy in Distributed Optimization and Learning

Nitin Vaidya

Georgetown University

Slides & Videos

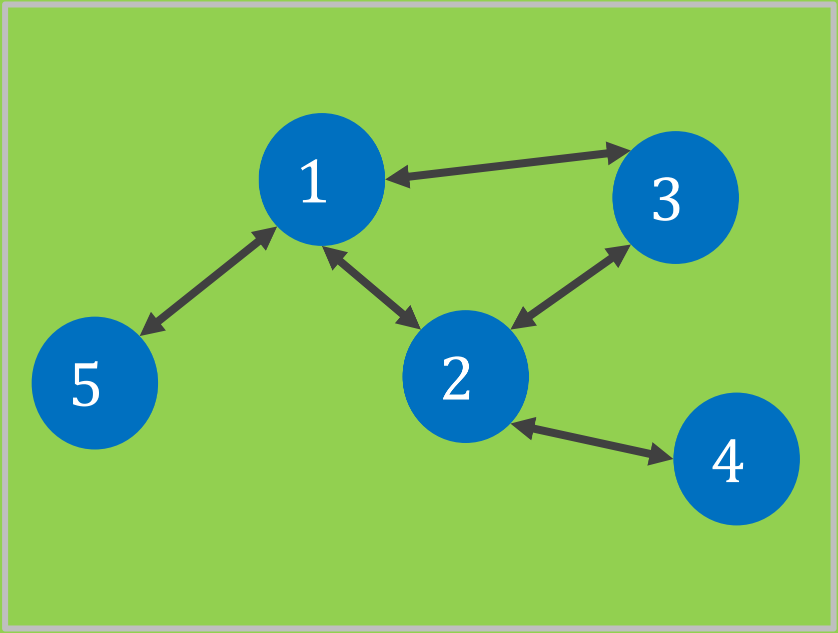
- Slides and videos for the tutorial posted at

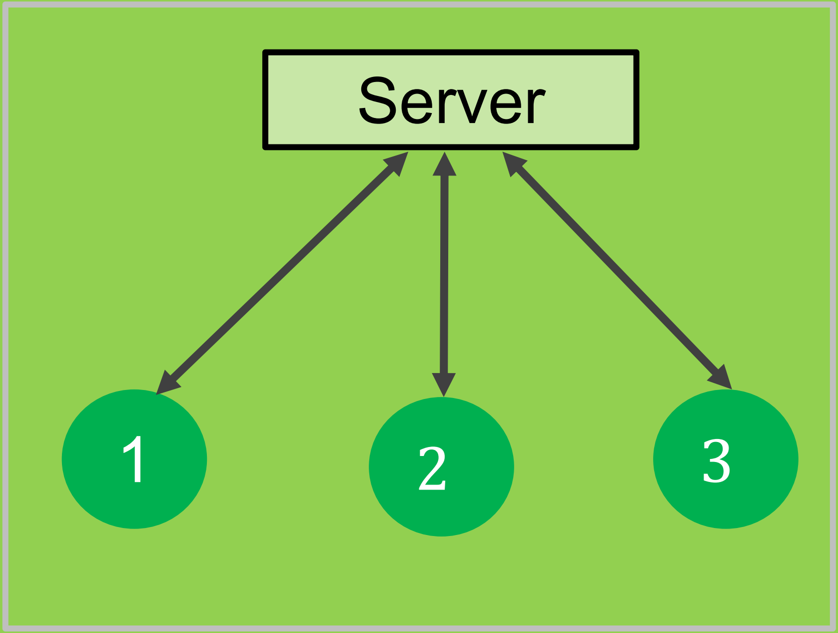
  https://disc.georgetown.domains

  Visit the tab for Talks at the above page

Outline

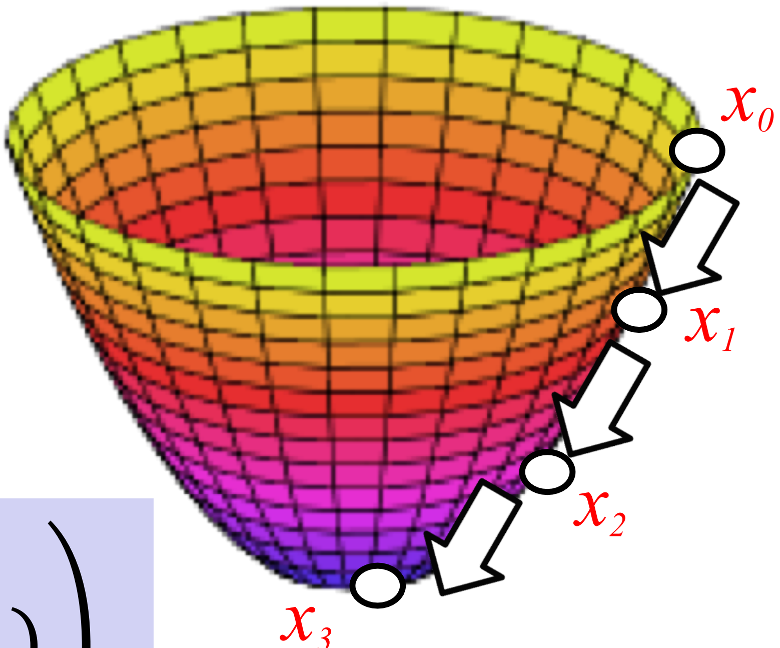
$$\text{argmin} \sum f_i(x)$$





Gradient Method

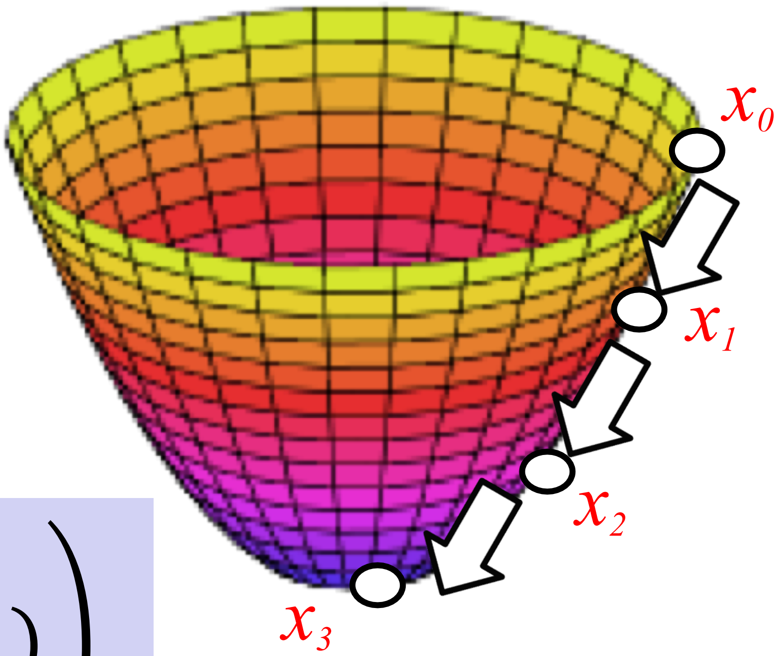
$$f(x) = \sum f_i(x)$$



$$x_{k+1} \leftarrow x_k - \lambda_k \nabla \left(\sum_i f_i(x_k) \right)$$

Gradient Method

$$f(x) = \sum f_i(x)$$

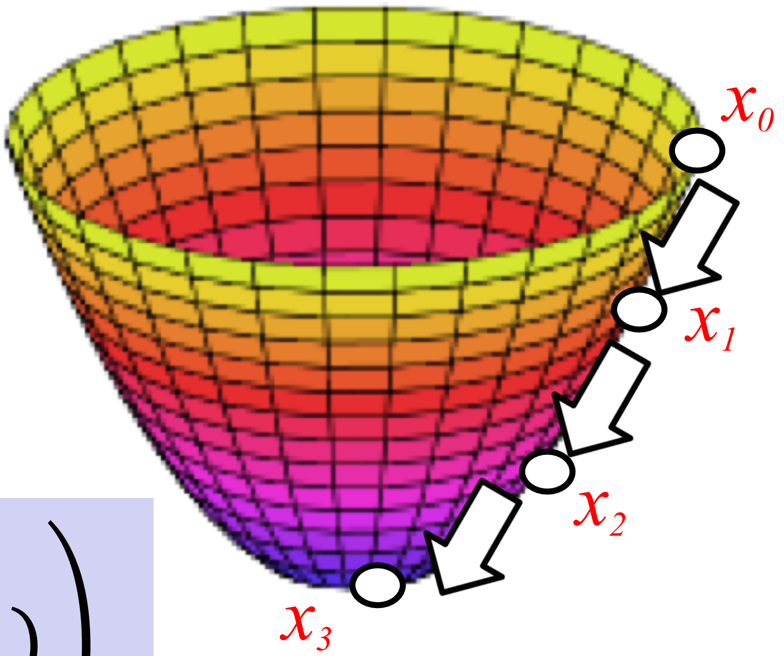


$$x_{k+1} \leftarrow x_k - \lambda_k \nabla \left(\sum_i f_i(x_k) \right)$$

$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \nabla f_i(x_k)$$

Gradient Method

$$f(x) = \sum f_i(x)$$



$$x_{k+1} \leftarrow x_k - \lambda_k \nabla\left(\sum_i f_i(x_k)\right)$$

$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \nabla f_i(x_k)$$

Distributed Optimization

- Each agent i knows own cost function $f_i(x)$
- Need to cooperate to minimize $\sum f_i(x)$

➔ Distributed algorithms

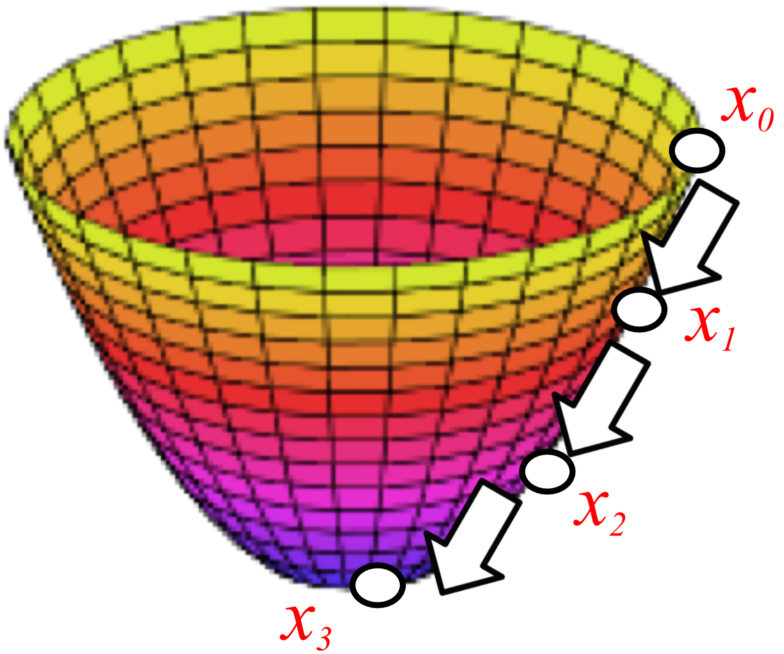
Gradient Method

$$f(x) = \sum f_i(x)$$

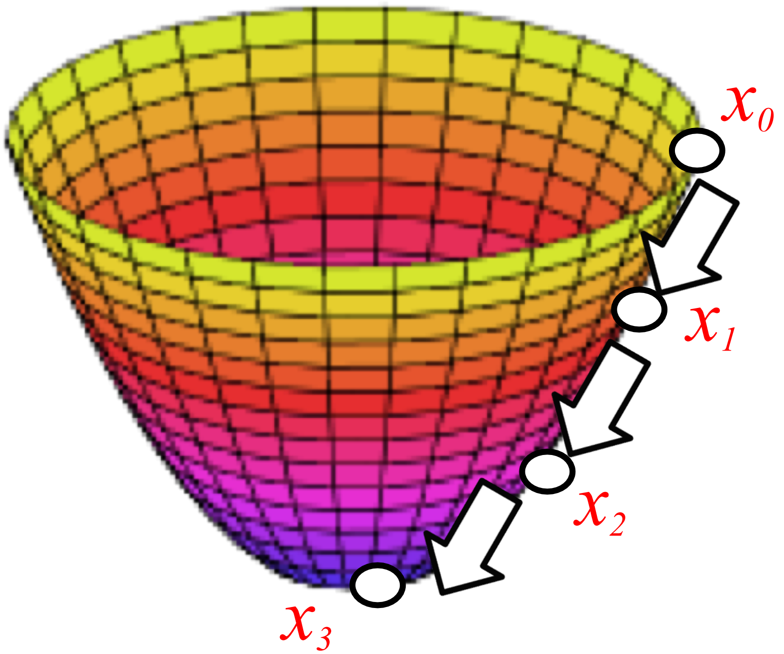
Uniform weights

for f_i's

$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \nabla f_i(x_k)$$



Gradient Method



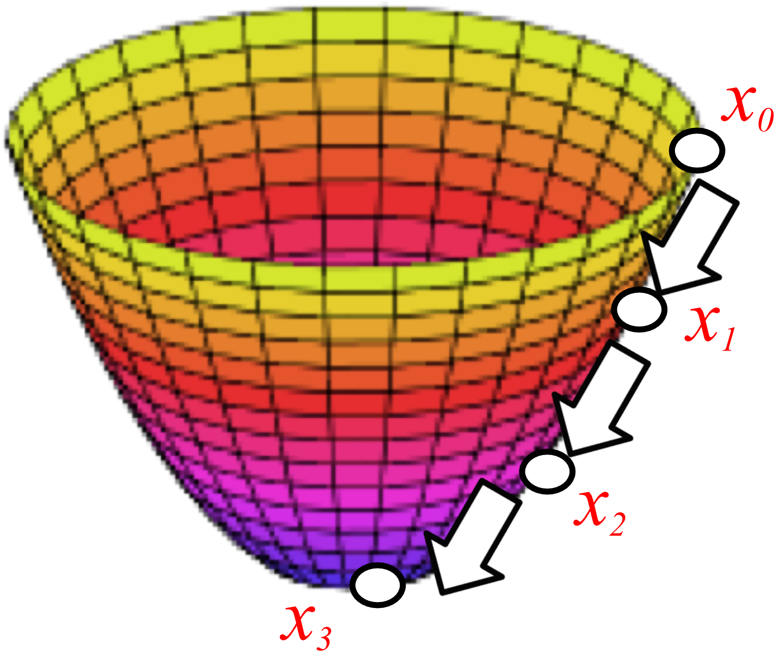
Non-uniform weights
for f_i's

$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \alpha_i \nabla f_i(x_k)$$

Gradient Method

$$f(x) = \sum \alpha_i f_i(x)$$

Non-uniform weights for f_i's



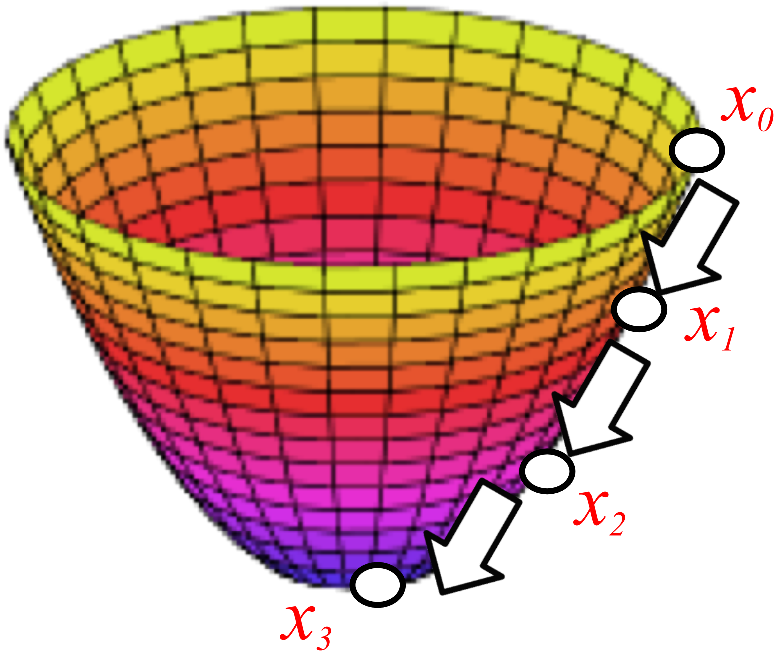
$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \alpha_i \nabla f_i(x_k)$$

Gradient Method

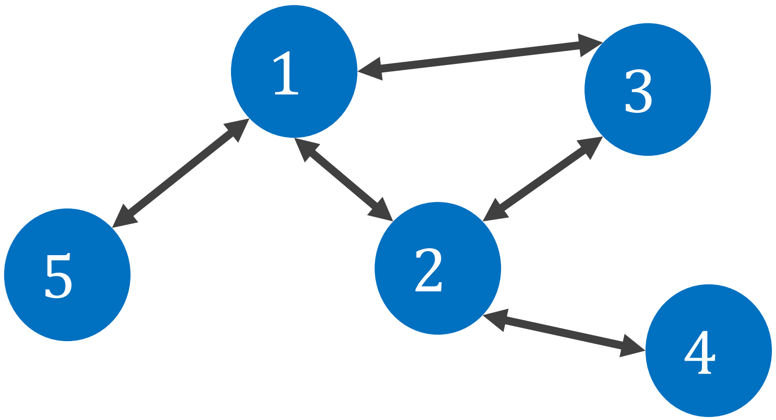
$$f(x) = \sum \alpha_i f_i(x)$$

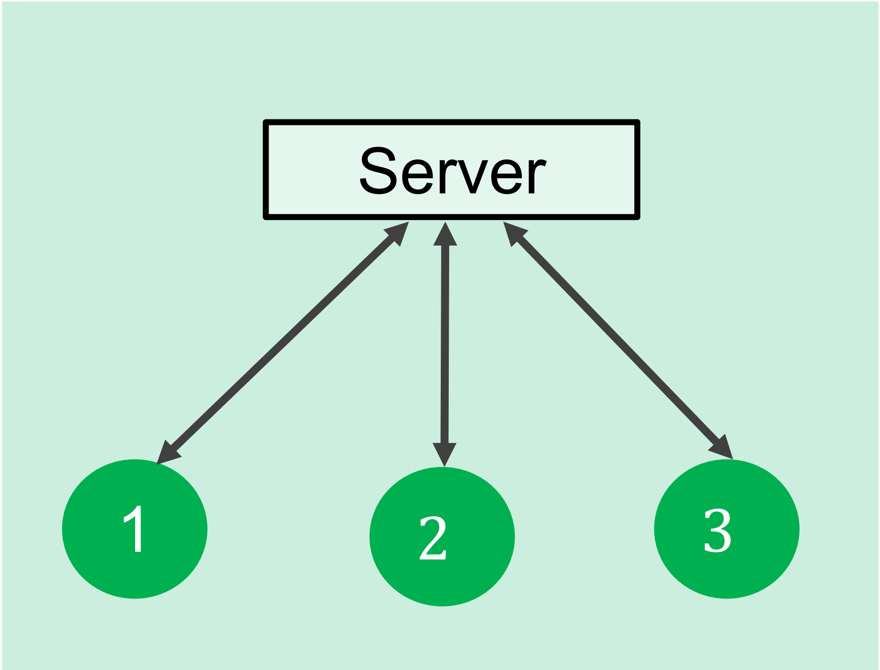
Uniform weights important

$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \alpha_i \nabla f_i(x_k)$$

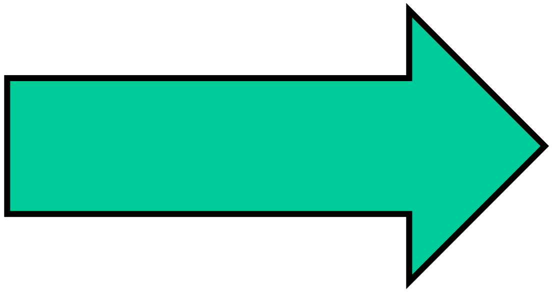


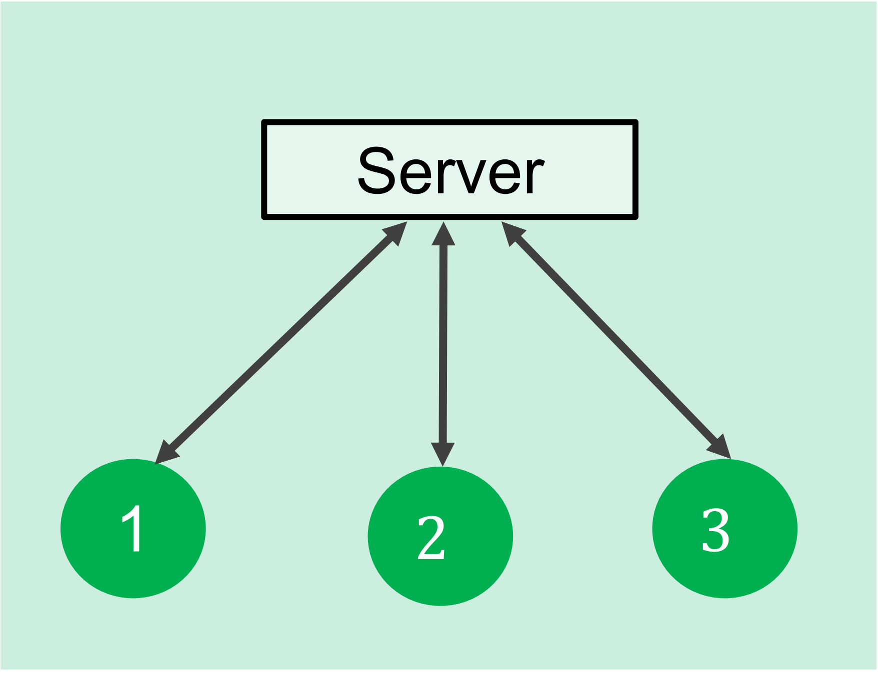
Architectures





Architectures





Jargon

I tend to refer to to all the variants as “distributed”, but the literature uses three terminologies

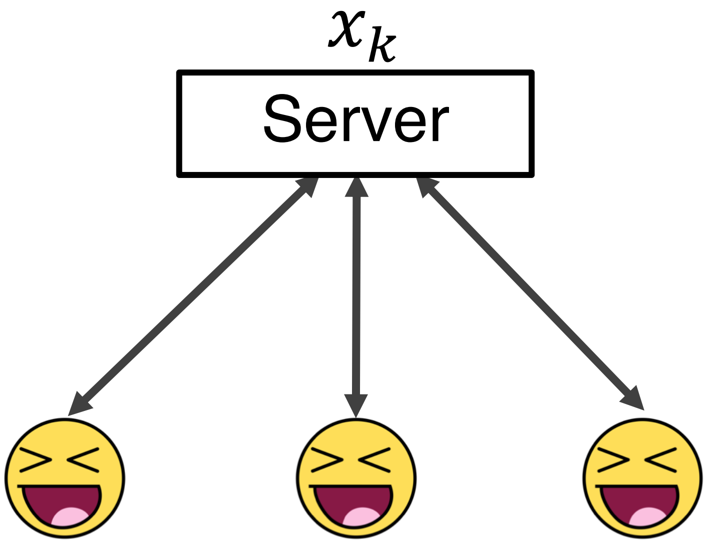
- Decentralized … Peer-to-peer
- Distributed … Server-based (clients supply gradients)
- Federated … Server-based (clients supply estimates)

… all are distributed algorithms

Federated Architecture

[Kairouz et al 2018]

- Server maintains estimate x_k



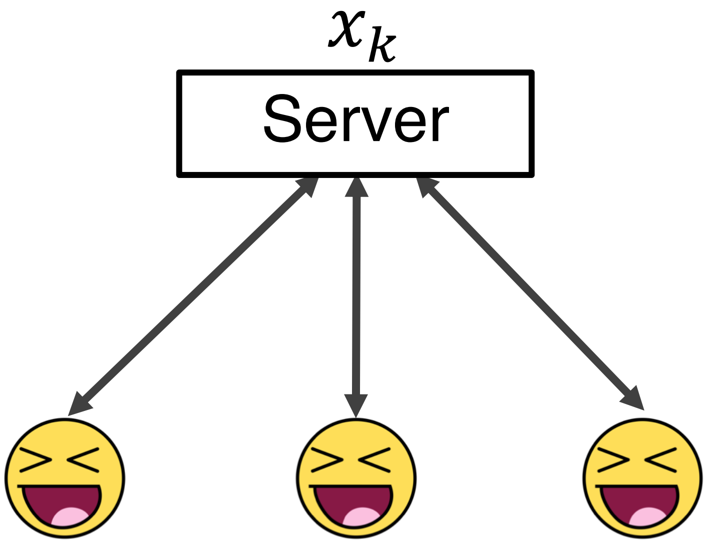
Federated Architecture

[Kairouz et al 2018]

- Server maintains estimate x_k

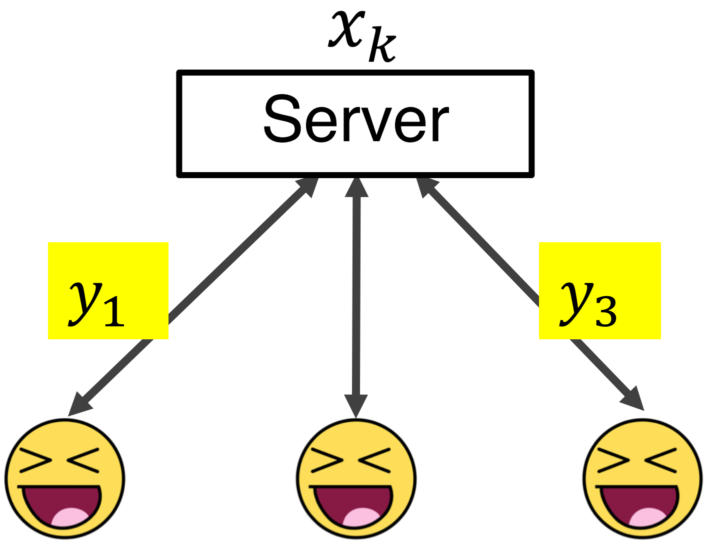
In each iteration

- Each agent i
  - Receive x_k from server



Federated Architecture

- Server maintains estimate x_k



<u>In each iteration</u>

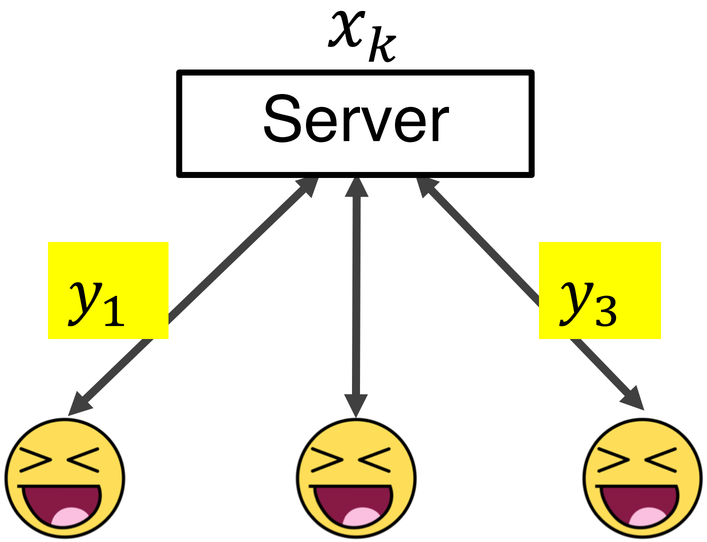
- Each agent i
  - Receive x_k from server
  - Compute $y_k = x_k - \lambda_k \nabla f_i(x_k)$
  - Send y_k to server

Federated Architecture

- Server maintains estimate x_k

In each iteration

- Each agent i
  - Receive x_k from server
  - Compute $y_k = x_k - \lambda_k \nabla f_i(x_k)$
  - Send y_k to server



- Server updates estimate

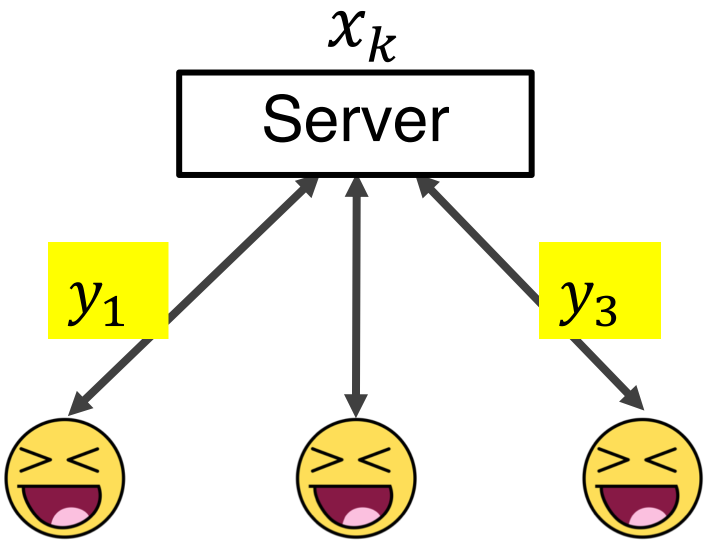
$$x_{k+1} \longleftarrow \frac{1}{n}\sum y_i$$

Federated Architecture

- Server maintains estimate x_k

<u>In each iteration</u>

- Each agent i
  - Receive x_k from server
  - Compute $y_k = x_k - \lambda_k \nabla f_i(x_k)$
  - Send y_k to server

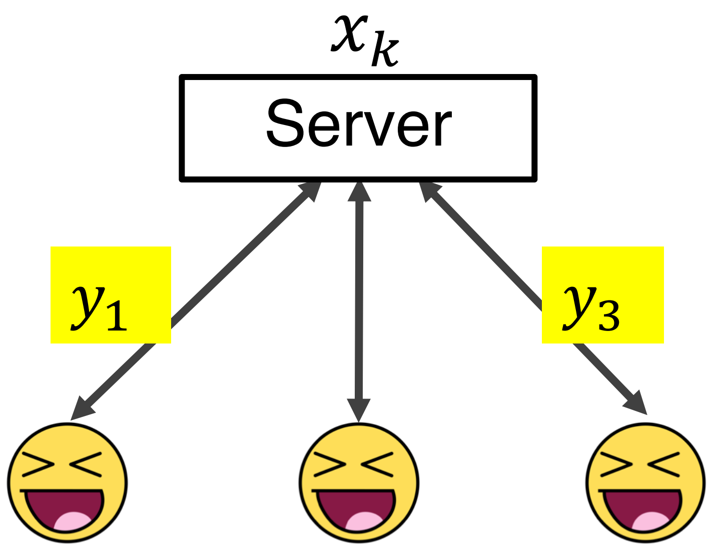


- Server updates estimate

$$x_{k+1} \leftarrow \frac{1}{n} \sum y_i$$

Federated Architecture

- Server maintains estimate x_k



<u>In each iteration</u>

- Each agent i
  - Receive x_k from server
  - Compute $y_k = x_k - \lambda_k \nabla f_i(x_k)$
  - Send y_k to server

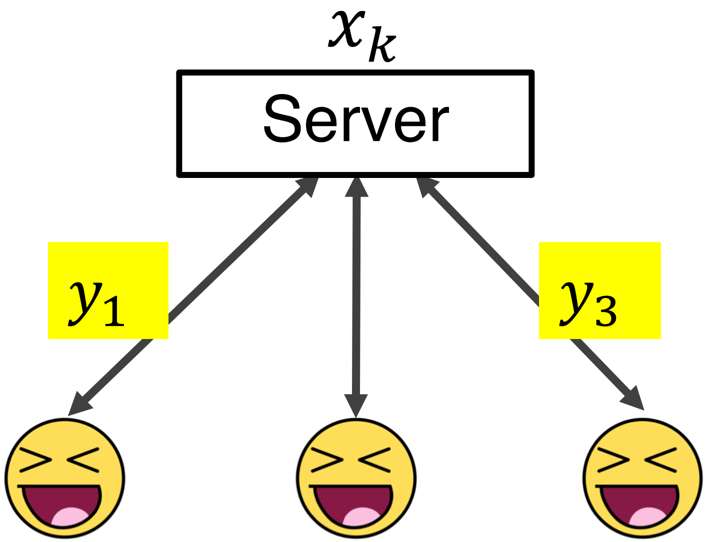
Replace single step by multiple steps

- Server updates estimate

$$x_{k+1} \longleftarrow \frac{1}{n} \sum y_i$$

Federated Architecture:
Stochastic Version

- Server maintains estimate x_k



In each iteration

- Each agent i in a size-s subset
  - Receive x_k from server
  - Compute $y_k = x_k - \lambda_k \nabla f_i(x_k)$
  - Send y_k to server

- Server updates estimate

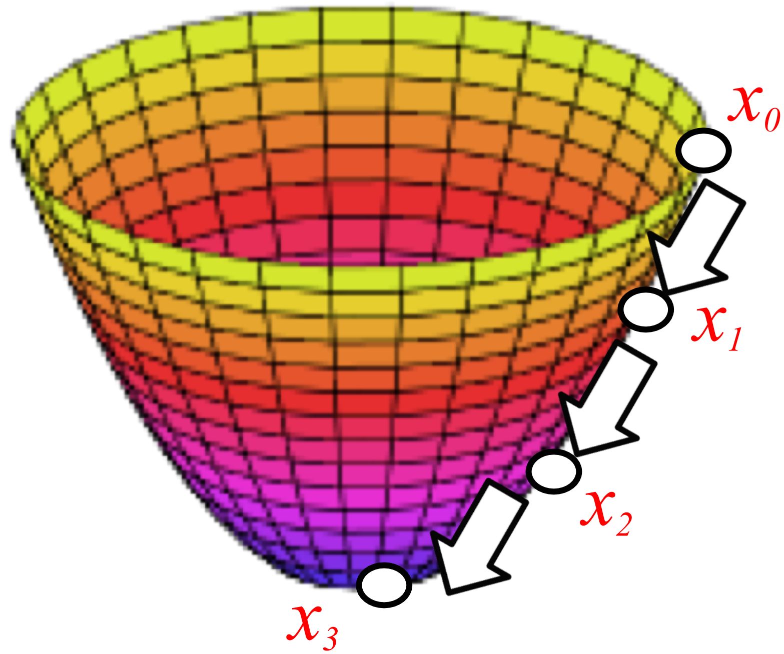
$$x_{k+1} \longleftarrow \frac{1}{s} \sum y_i$$

Recall

$$f(x) = \sum \alpha_i f_i(x)$$

Uniform weights important

$$x_{k+1} \leftarrow x_k - \lambda_k \sum_i \alpha_i \nabla f_i(x_k)$$



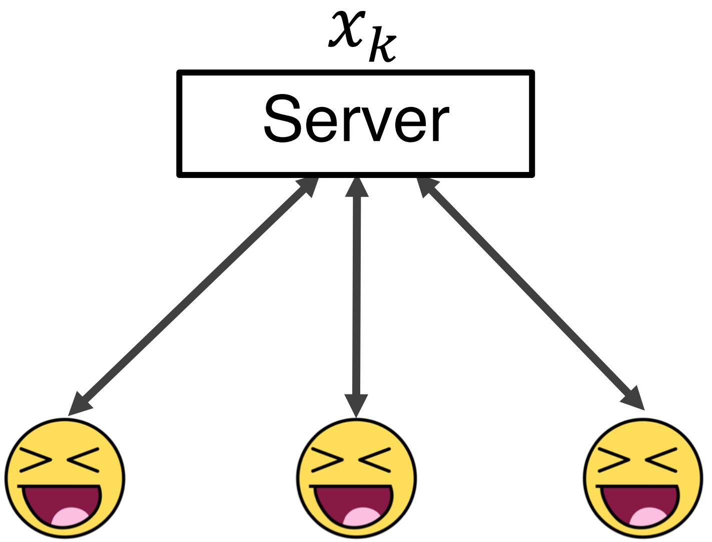
Federated Architecture:
Stochastic Version

- To ensure correct “weights”,

  agents must be sampled uniformly

Distributed Optimization

- Server maintains estimate x_k

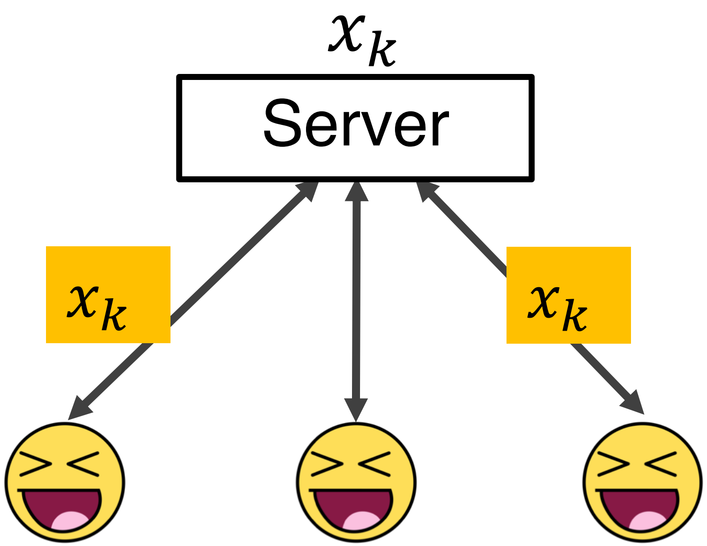


Distributed Optimization

- Server maintains estimate x_k

<u>In each iteration</u>

- Each agent i
  - Receives x_k from server

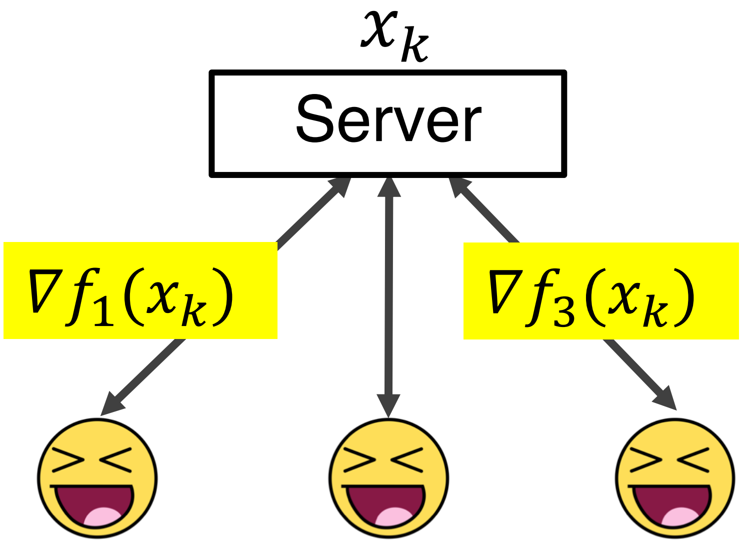


Distributed Optimization

- Server maintains estimate x_k

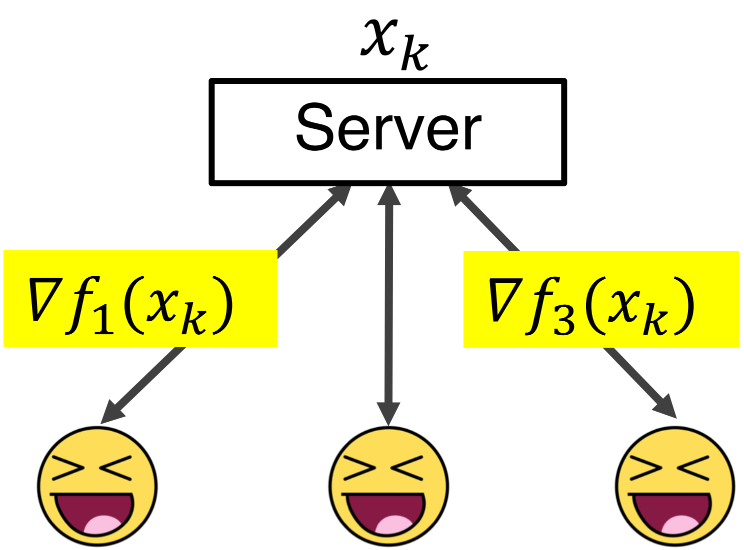
<u>In each iteration</u>

- Each agent i
  - Receives x_k from server
  - Uploads gradient $\nabla f_i(x_k)$



Distributed Optimization

- Server maintains estimate x_k



<u>In each iteration</u>

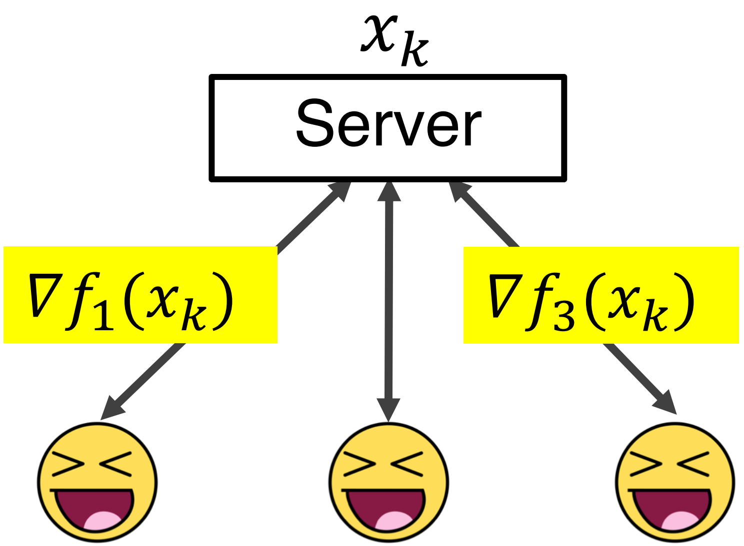
- Each agent i
  - Receives x_k from server
  - Uploads gradient $\nabla f_i(x_k)$

- Server updates estimate

$$x_{k+1} \longleftarrow x_k - \lambda_k \sum \nabla f_i(x_k)$$

Distributed Optimization: Stochastic Version

- Server maintains estimate x_k



<u>In each iteration</u>

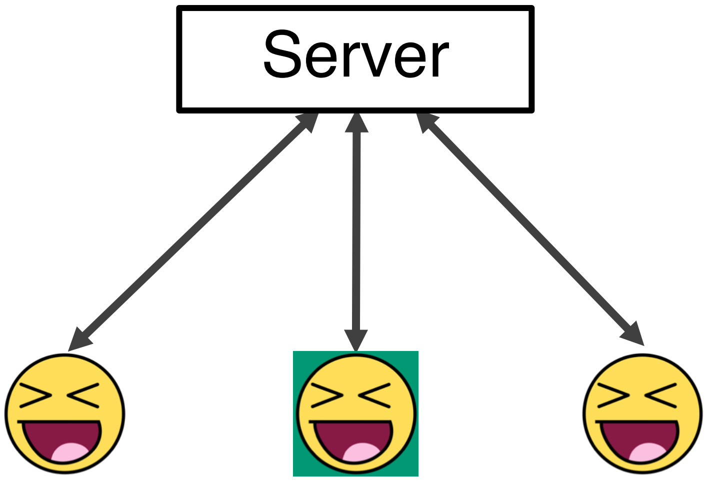
- Each agent i in a subset
  - Receives x_k from server
  - Uploads gradient $\nabla f_i(x_k)$

- Server updates estimate

$$x_{k+1} \leftarrow x_k - \lambda_k \sum \nabla f_i(x_k)$$

Stochastic Distributed Machine Learning
[Bottou,Curtis,Nocedal 2016]

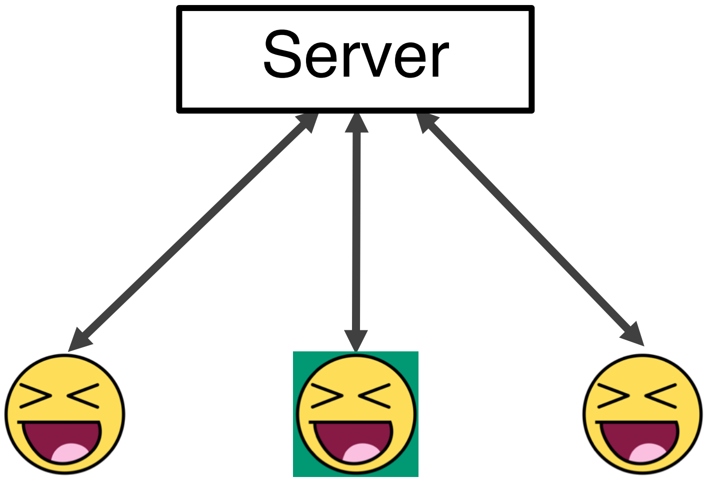
Two dimensions of randomization



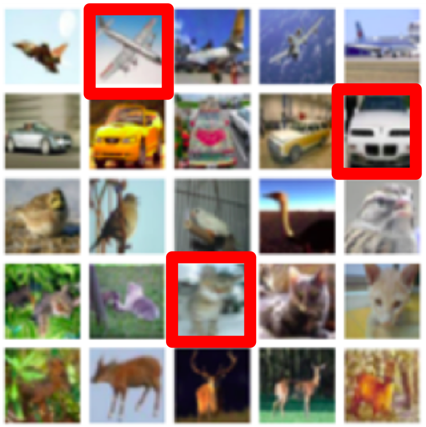
- Select a subset of agents randomly in each round

Stochastic Distributed Machine Learning

Two dimensions of randomization



- Select a subset of agents randomly in each round



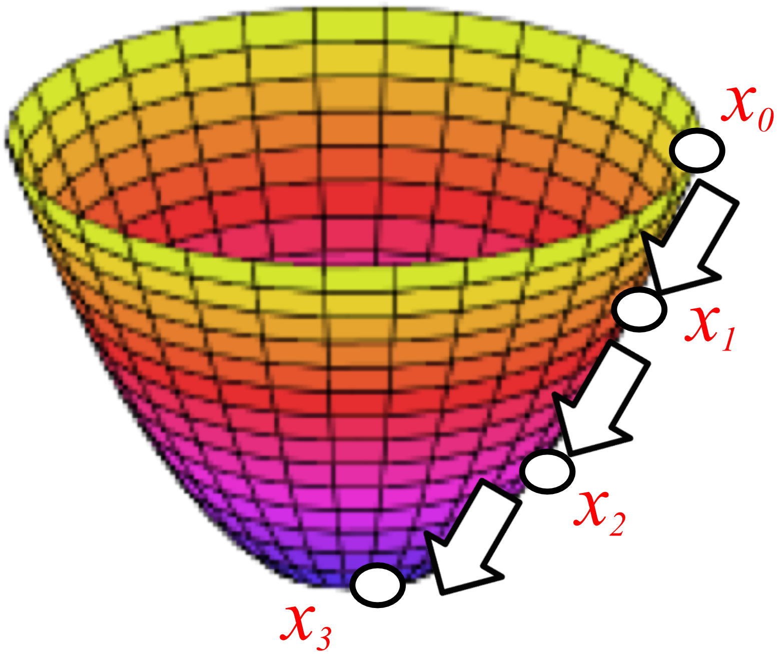
- Each agent may compute gradient over a subset of data available to that agent

Recall

$$f(x) = \sum \alpha_i f_i(x)$$

Uniform weights important

$$x_{k+1} \longleftarrow x_k - \lambda_k \sum_i \alpha_i \nabla f_i(x_k)$$



Stochastic Distributed Machine Learning
Heterogeneous Case (“non-I.I.D.”)

- Each agent has access to a subset of the dataset

➔ $f_i(x) \neq f_j(x)$

➔ Each agent draws gradients from a different distribution

- Need to be careful to ensure equal “weights” for agents
- Availability of multiple agents provides parallelism

Stochastic Distributed Machine Learning
Homogeneous Case ("I.I.D.")

- Each agent has access to the same dataset

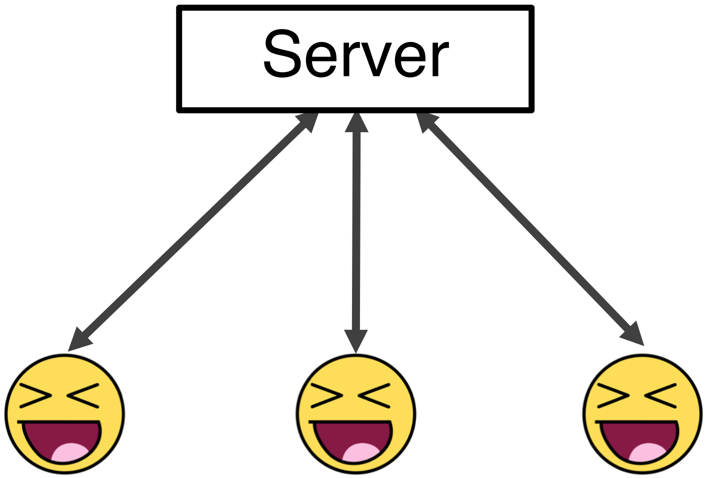
➔ $f_i(x) = f_j(x)$

➔ Each agent draws gradients with the same distribution

- Availability of multiple agents provides parallelism

- Optimization Methods for Large-Scale Machine Learning
  Léon Bottou, Frank E. Curtis, Jorge Nocedal
  2018

Other Variations



… asynchronous

… gradient compression

… shared memory

Disadvantage of Synchronous Computation

- The server cannot update estimate until ALL clients have responded

- Slowest client dictates speed … stragglers are bad

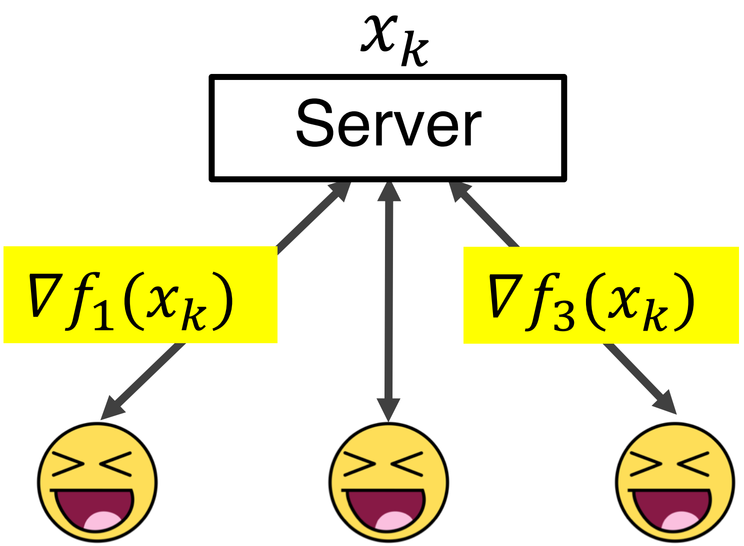
→ Asynchronous computation to the rescue

Recall … Synchronous Algorithm

- Server maintains estimate x_k

<u>In each iteration</u>

- Each agent i
  - Receives x_k from server
  - Uploads gradient $\nabla f_i(x_k)$



- Server updates estimate

$$x_{k+1} \longleftarrow x_k - \lambda_k \sum \nabla f_i(x_k)$$

Asynchrony

Different research communities use the term somewhat differently

- Distributed algorithms (ACM PODC, for instance):
  Delays are finite, but unbounded

- Decentralized control (e.g., CDC) and machine learning (e.g., NeurIPS):
  - Bounded delays, or
  - Strong assumptions on delay distribution

Optimization literature typically uses the latter interpretation

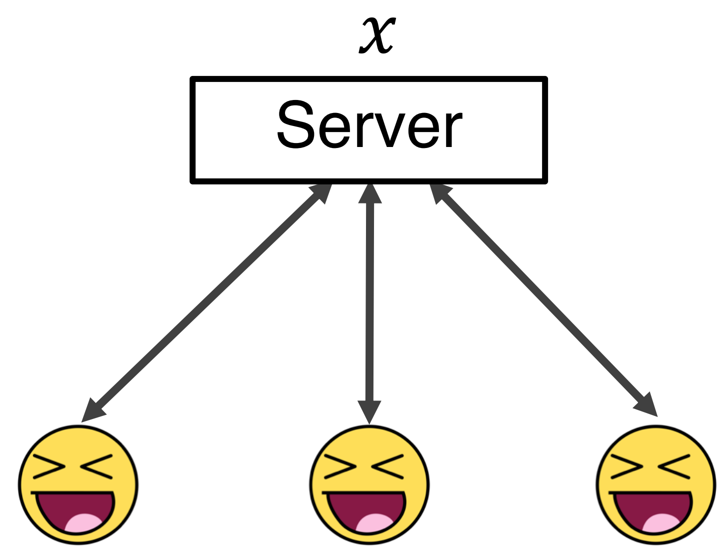
Asynchronous Algorithm

- No need to wait for all gradients
- Example … update server's estimate after receiving gradient from any client

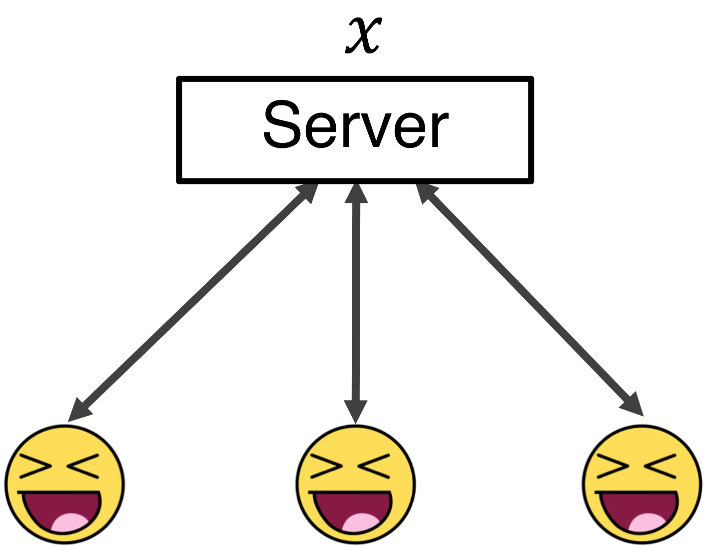
Asynchronous Algorithm

- Server maintains estimate x

- Agent i
  - Receive current x from server
  - Uploads gradient $\nabla f_i(x)$



Asynchronous Algorithm



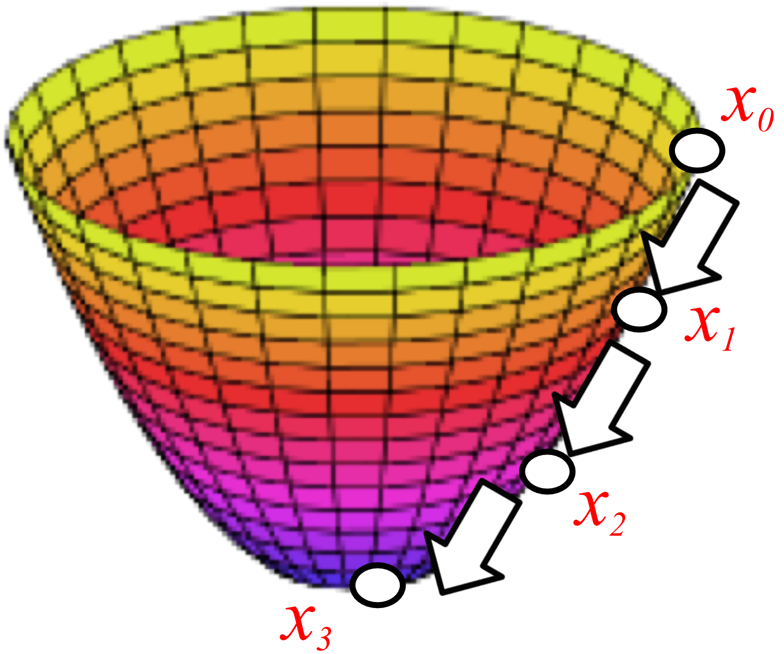
- Server maintains estimate x
- Agent i
  - Receive current x from server
  - Uploads gradient $\nabla f_i(x)$
- Server updates estimate on receiving gradient $\nabla f_j(.)$ from any client j

$$x \longleftarrow x - \lambda_{k,j} \, . \, (??)$$

Recall

$$f(x) = \sum \alpha_i f_i(x)$$

Uniform weights important



$$x_{k+1} \longleftarrow x_k - \lambda_k \sum_i \alpha_i \nabla f_i(x_k)$$

Asynchronous Algorithm

- Different agents may experience different delays

Asynchronous Algorithm

- Different agents may experience different delays

- Need to ensure equal “weights”

  - Adjust step size proportionally with time between updates from a given agent

$$x_{k+1} \longleftarrow x_k - \lambda_{k,j} \nabla f_j(?)$$

Asynchronous Algorithm

- Different agents may experience different delays

- Need to ensure equal “weights”

  - Adjust step size proportionally with time between updates from a given agent

$$x_{k+1} \longleftarrow x_k - \lambda_{k,j} \nabla f_j(?)$$

  - Use “stale” gradients from agents, if needed (use all agents in each iteration)

$$x_{k+1} \longleftarrow x_k - \lambda_k \sum \nabla f_i(?)$$

Asynchronous Message Passing

- Much of the work implicitly assumes message passing

- Agents receives a "consistent" view of the entire state vector x_k from the server

  and their updates are applied "atomically"

$$x_{k+1} \longleftarrow x_k - \lambda_k \sum \nabla f_i(?)$$

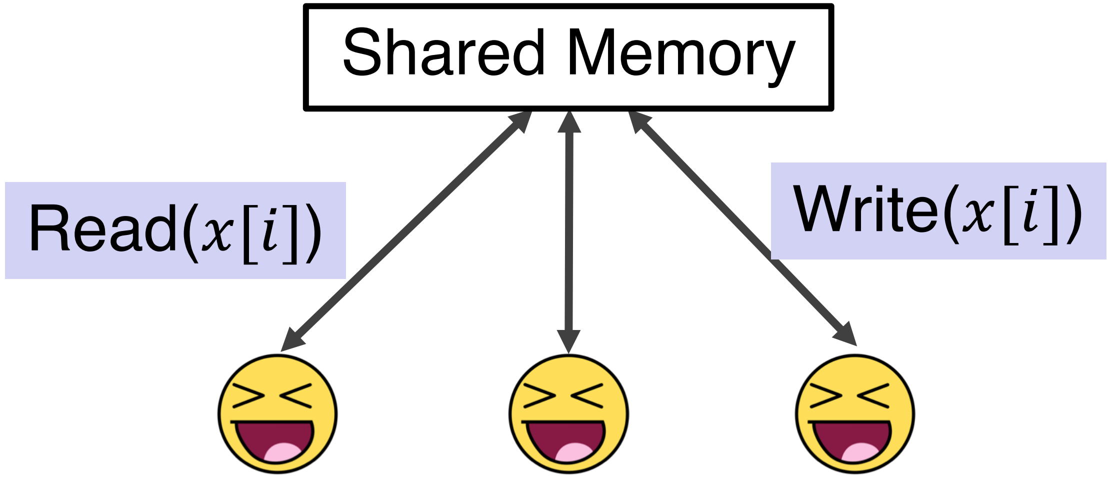
- Behavior may be different in shared memory

Asynchronous Shared Memory
[Alistarh et al. 2018]

- Agents read elements of x independently
  … not an "atomic read"

- Updates of x are also not atomic

- Agents have an inconsistent view of the state of x

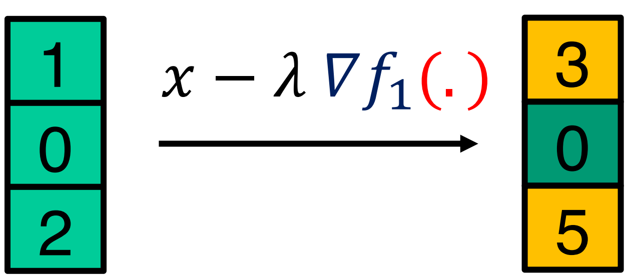


Asynchronous Shared Memory
[Alistarh et al. 2018]

- Agents read elements of x independently
  … not an "atomic read"

- Updates of x are also not atomic

- Agents have an inconsistent view of the state of x



Initial x
read by agent 1

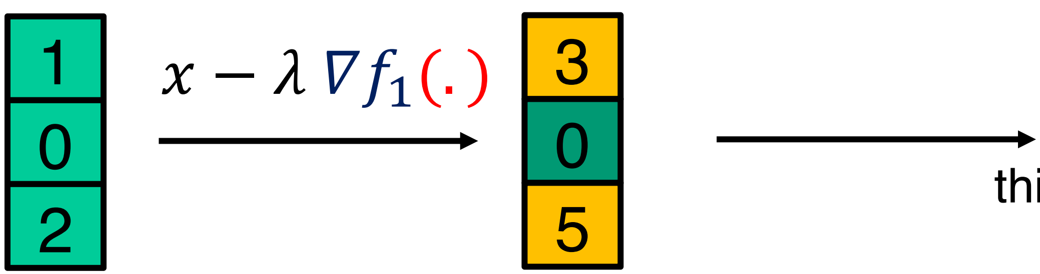
Agent 1's update
applied partially so far

Asynchronous Shared Memory
[Alistarh et al. 2018]

- Agents read elements of x independently
  … not an "atomic read"

- Updates of x are also not atomic

- Agents have an inconsistent view of the state of x



Agent 2 reads
this "inconsistent" vector as x
and computes gradient

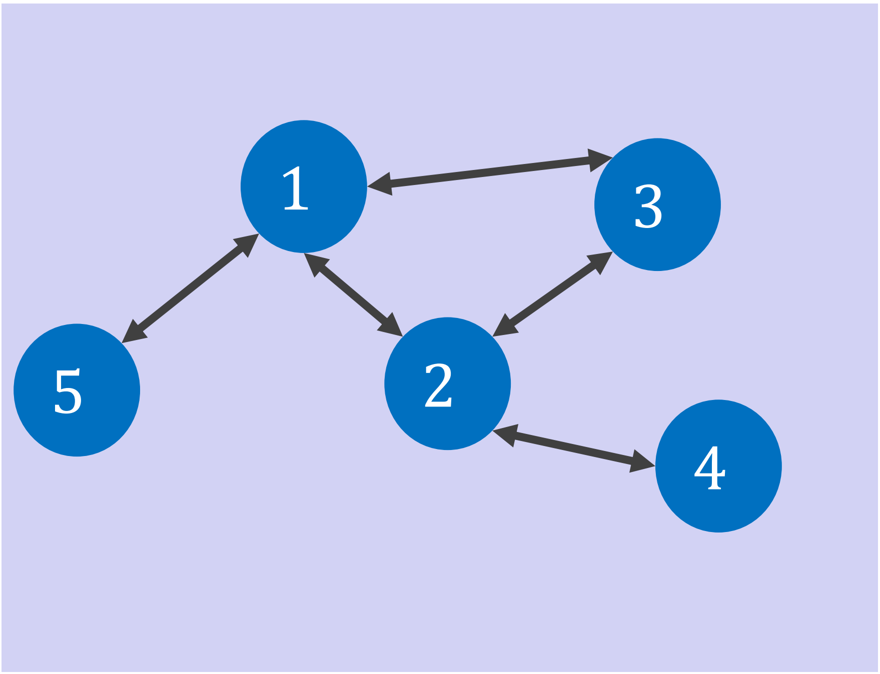
Initial x
read by agent 1

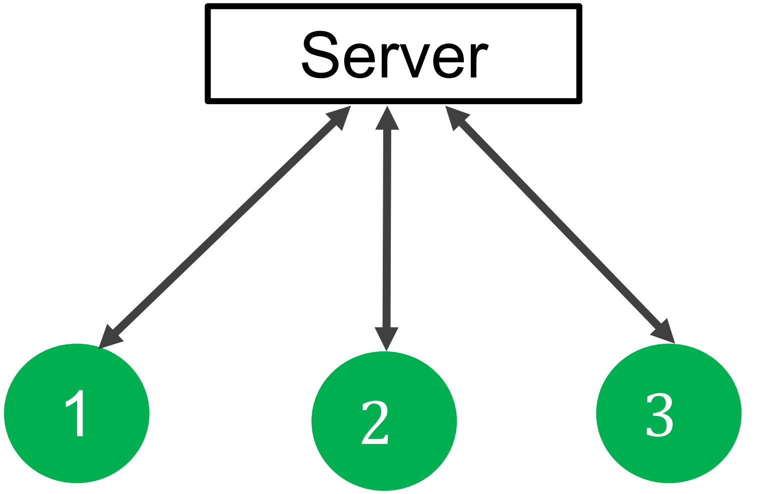
Agent 1's update
applied partially so far

Gradient Compression

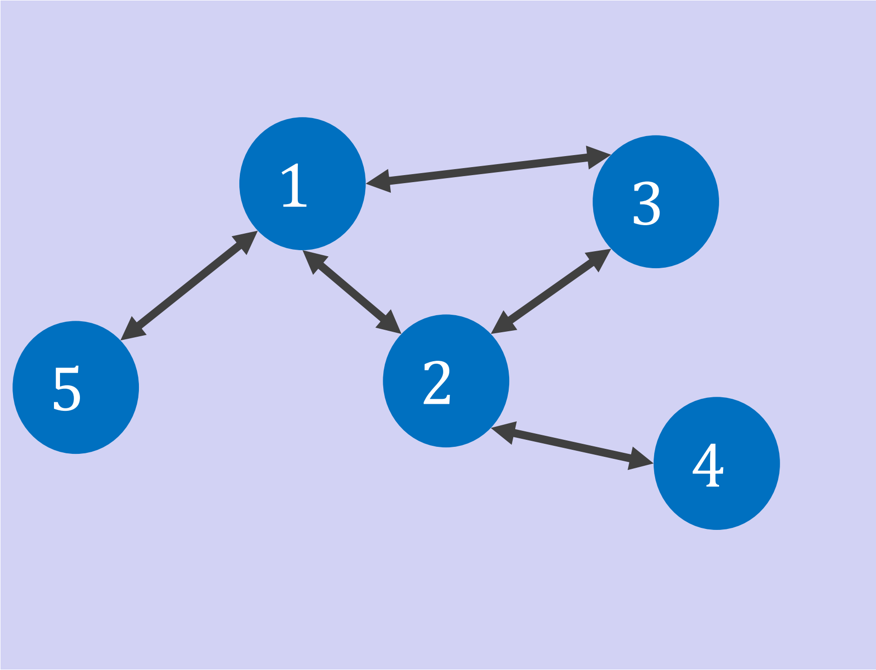
- Length of gradient vector equals length of vector x
- Can be very large … for instance, x may represent parameters of a deep neural network
- Compression … reduce communication cost
  - Only send elements of gradient vector that have changed “significantly” since last transmission of gradient
  - Only send top-K largest elements of the gradient vector

Architectures





Architectures



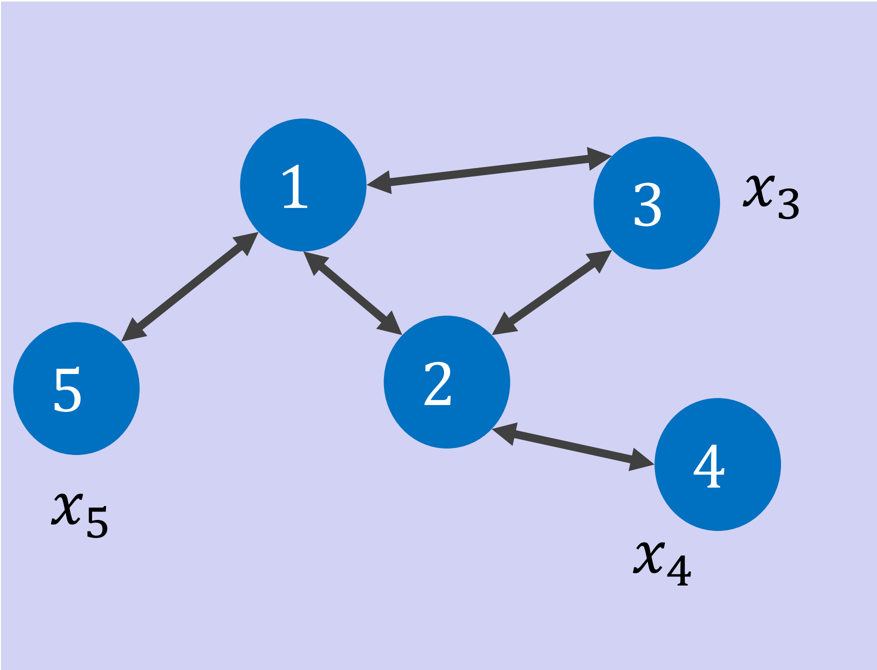
Multi-Agent

Peer-to-Peer (p2p)

Decentralized

Many Variations
[Tsitsiklis 1984]

- Version not considered in this tutorial
  - Each agent knows identical cost function $f(x)$
  - Agents cooperate to determine argmin $f(x)$
  - Agent i responsible to determine i-th element of argmin $f(x)$



Many Variations
[Tsitsiklis 1984]

- Version not considered in this tutorial
  - Each agent knows identical cost function $f(x)$
  - Agents cooperate to determine argmin $f(x)$
  - Agent i responsible to determine i-th element of argmin $f(x)$

- Version considered in this tutorial
  - Agent i knows identical cost function $f_i(x)$
  - Agents cooperate to determine argmin $\sum f_i(x)$
  - Each agent learns argmin $\sum f_i(x)$

Many Variations

- Synchronous or Asynchronous
- Lossy or reliable links

We will consider the synchronous setting and error-free links

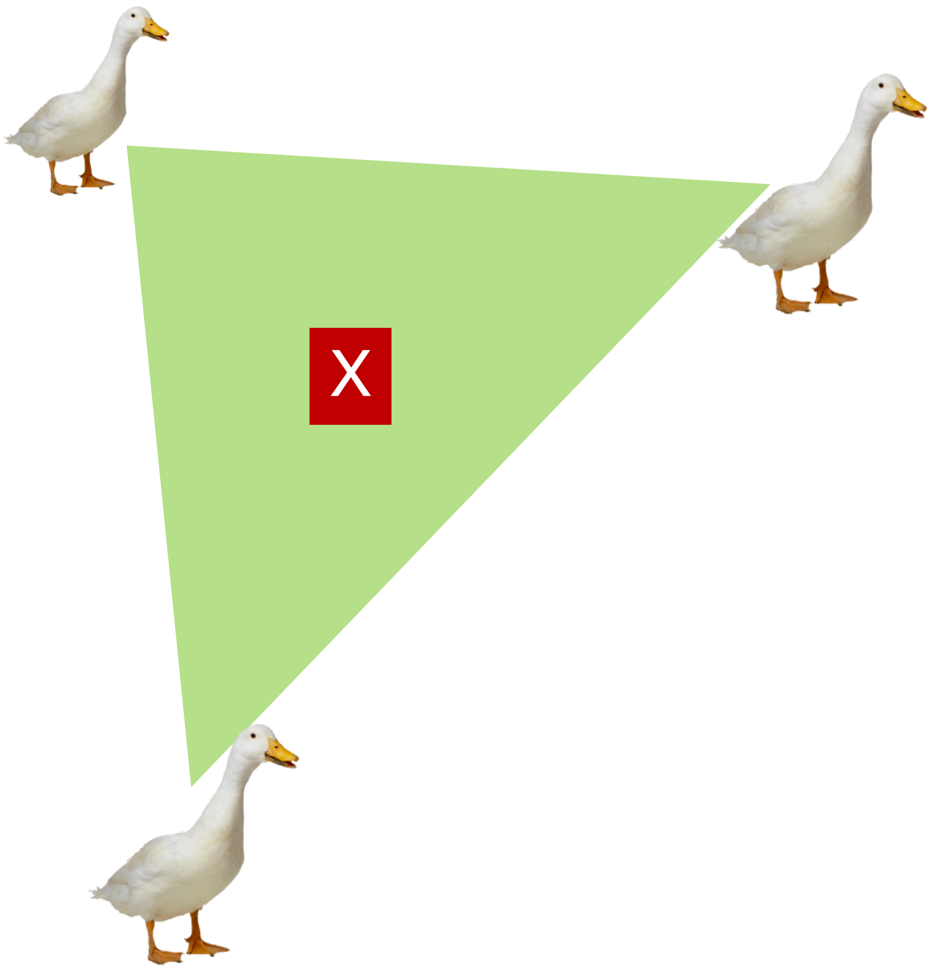
A Detour … Average Consensus

Average Consensus

- Each node has an input (scalar or vector)

- Average consensus: Output = average of inputs

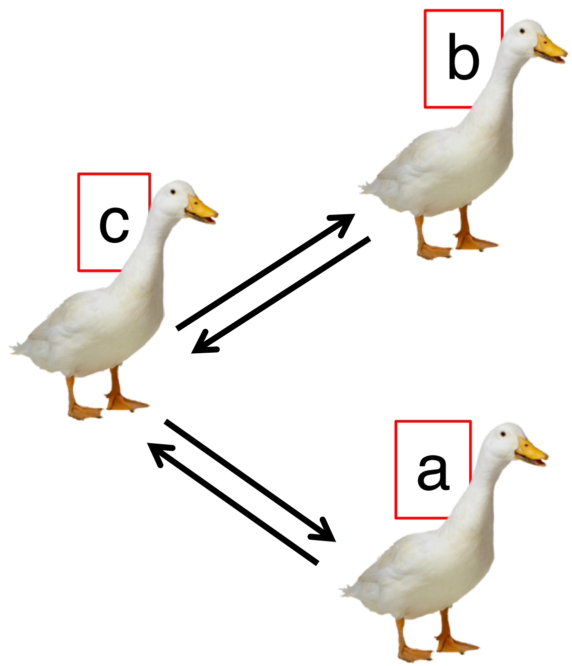
Average Consensus



Average Consensus

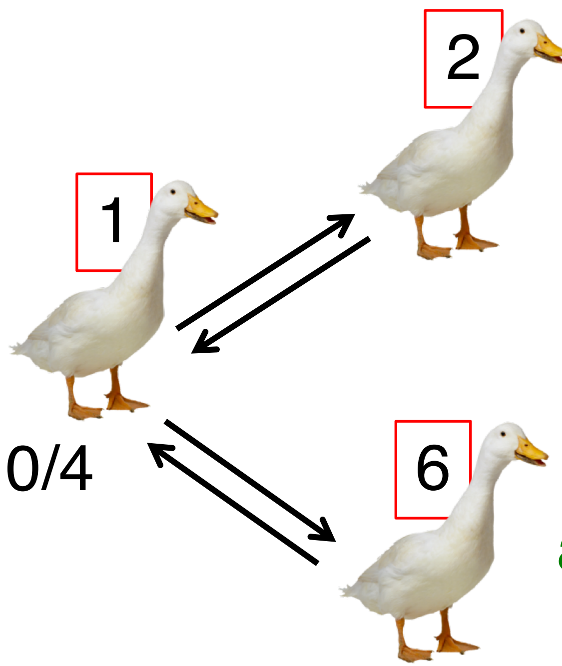
$b = 3b/4 + c/4$

$c = a/4+b/4+c/2$



$a = 3a/4 + c/4$

Average Consensus



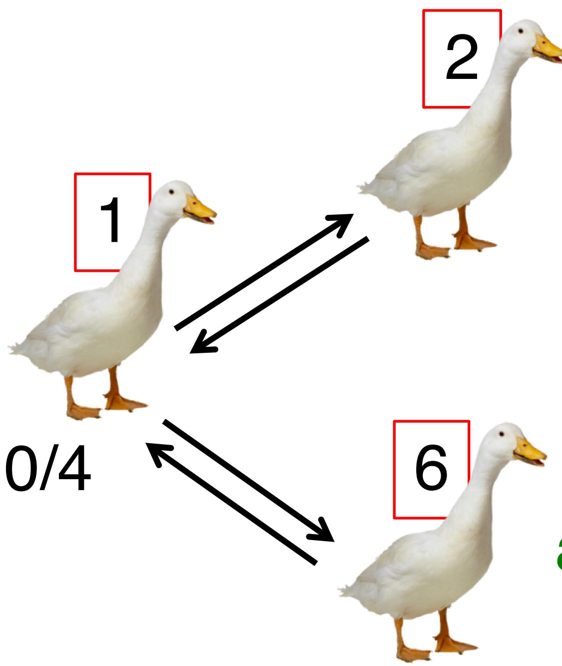
b = (6/4+1/4)=7/4

c = (2/4+6/4+1/2) = 10/4

a = (18/4+1/4) = 19/4

Average Consensus

As time → ∞, values converge to *average* of inputs



b = (6/4+1/4)=7/4

c = (2/4+6/4+1/2) = 10/4

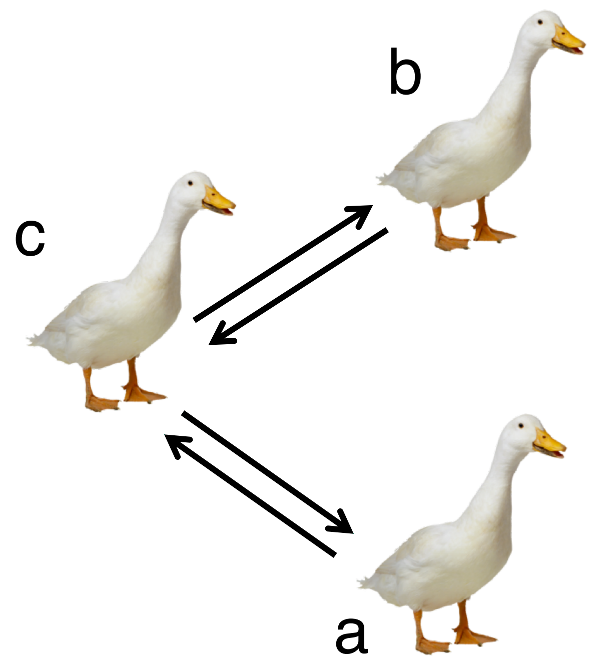
a = (18/4+1/4) = 19/4

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}_1 = \underbrace{\begin{pmatrix} 3/4 & 0 & 1/4 \\ 0 & 3/4 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}}_{M} \begin{pmatrix} a \\ b \\ c \end{pmatrix}_0 = M \begin{pmatrix} a \\ b \\ c \end{pmatrix}_0$$

iteration index

$b = 3b/4 + c/4$

$c = a/4 + b/4 + c/2$



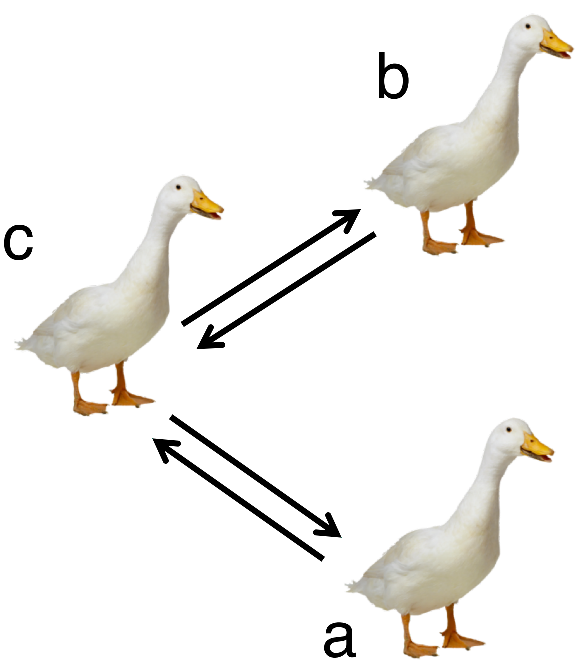
$a = 3a/4 + c/4$

after 2 iterations

after 1 iteration

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}_2 = M \left[M \begin{pmatrix} a \\ b \\ c \end{pmatrix}_0 \right] = M^2 \begin{pmatrix} a \\ b \\ c \end{pmatrix}_0$$

c = a/4+b/4+c/2



b = 3b/4+ c/4

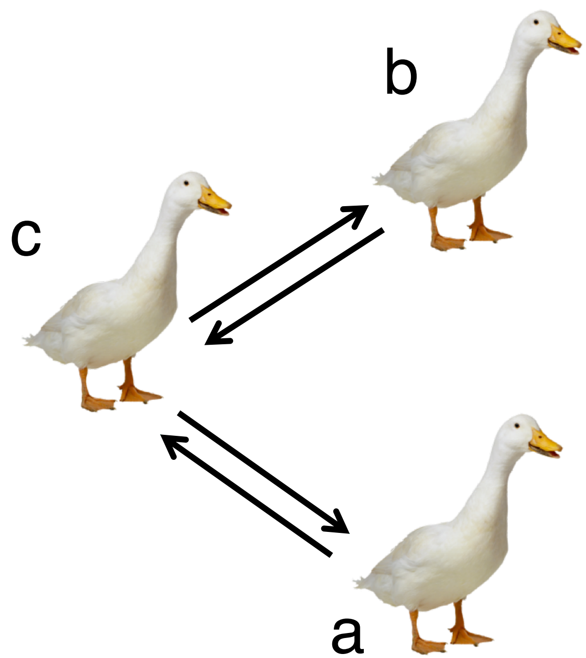
a = 3a/4+ c/4

after k iterations

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}_k = M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix}_0$$

b = 3b/4+ c/4

c = a/4+b/4+c/2



a = 3a/4+ c/4

after k iterations

Iteration index

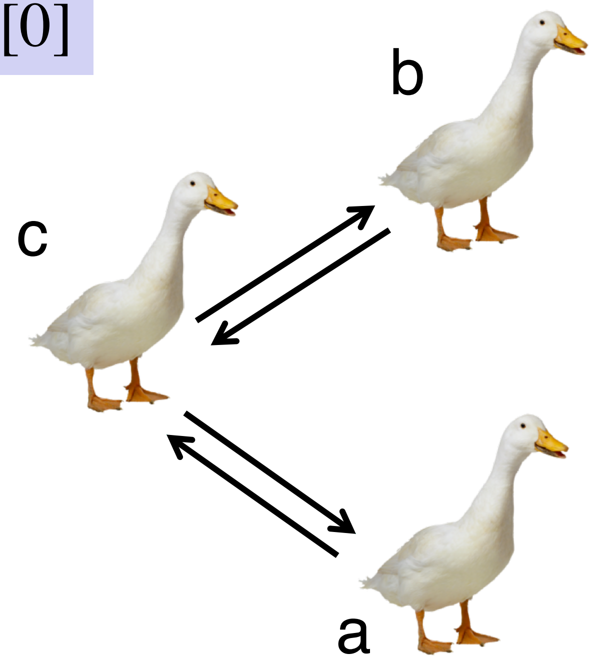
$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}_k = M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix}_0$$

$x[k]$ $x[0]$

$$\Rightarrow \quad x[k] = M^k x[0]$$

$b = 3b/4 + c/4$

$c = a/4 + b/4 + c/2$



$a = 3a/4 + c/4$

Connected Undirected Graphs

after k iterations

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

- *Average* consensus

  if M doubly stochastic

  - Matrix elements in [0,1]
  - M_{ij} non-zero if link (i,j) exists
  - Each row & each column adds to 1

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := \underset{\mathrm{M}}{\begin{pmatrix} 3/4 & 0 & 1/4 \\ 0 & 3/4 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \mathrm{M} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Due to stochastic rows,
each new state
in convex hull
of old states

Due to stochastic columns,
total “mass”
(sum of states)
is preserved

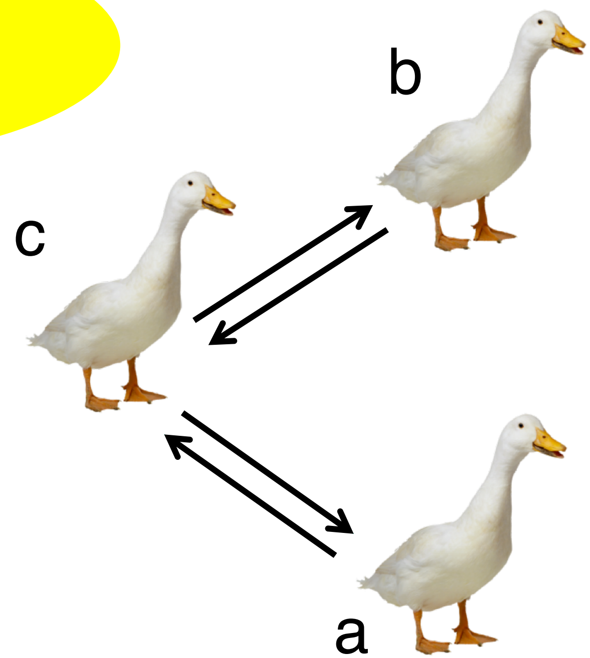
$k \to \infty$

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix} \rightarrow \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Doubly stochastic M

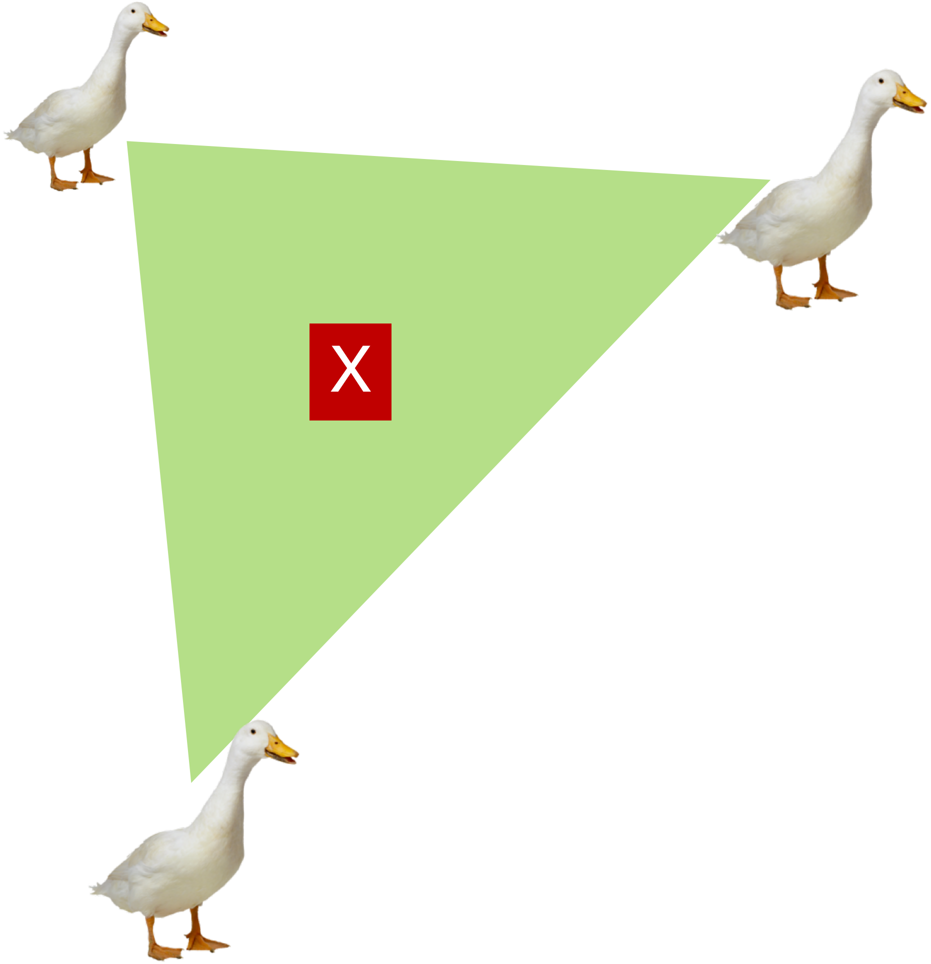
b = 3b/4+ c/4

c = a/4+b/4+c/2

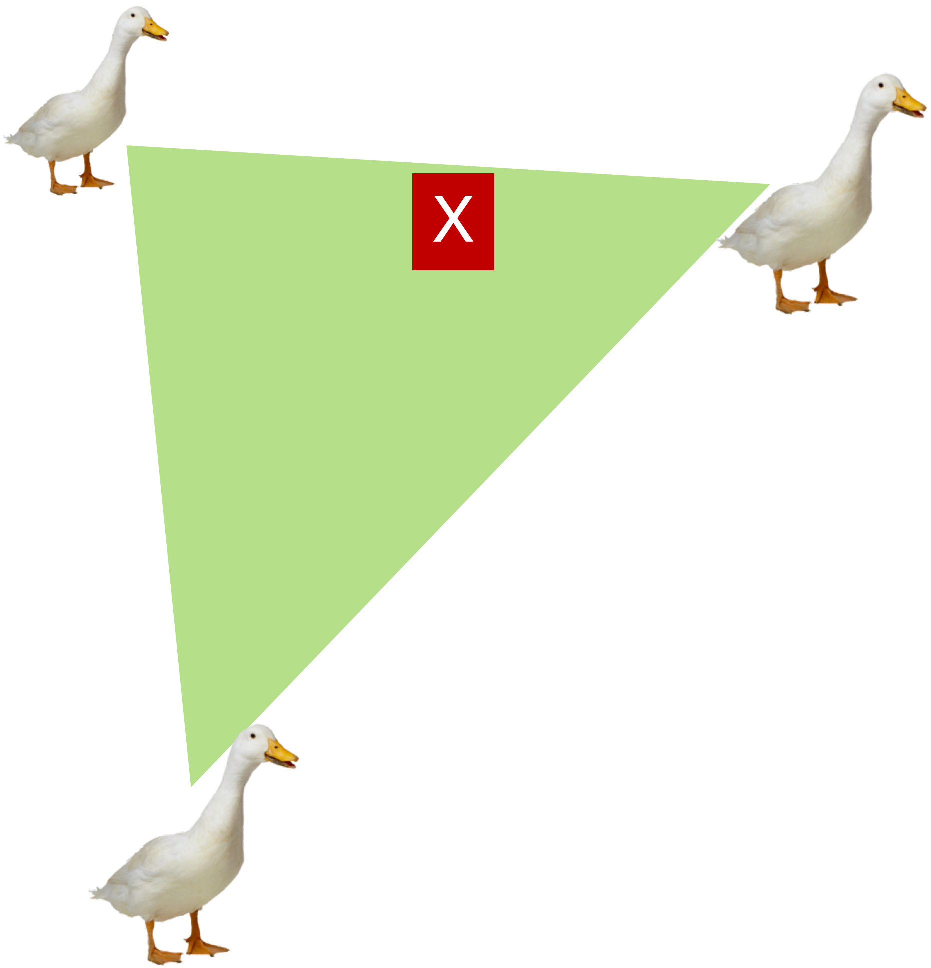


a = 3a/4+ c/4

Average Consensus



Optimization $\text{argmin} \sum f_i(x)$

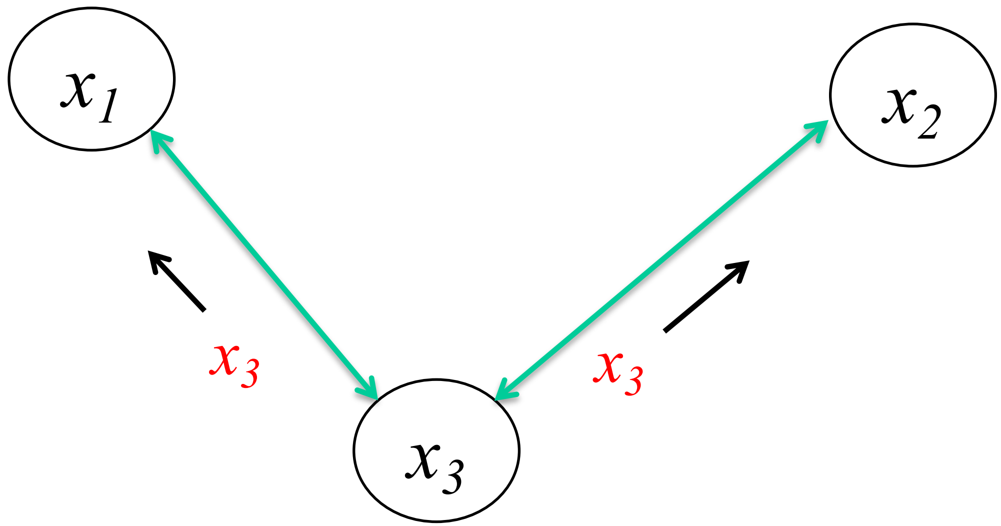


Distributed Optimization

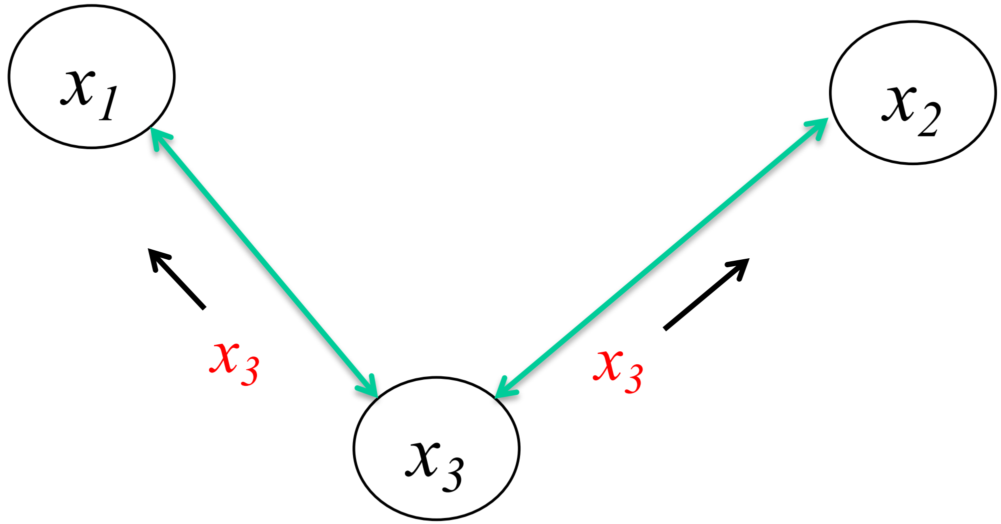
$$f(x) = \sum f_i(x)$$

Iterative algorithm

- Each agent maintains an estimate
- Local estimates shared with neighbors & updated in each iteration
- Estimates converge to optimum



Example based on [Nedic and Ozdaglar, 2009]

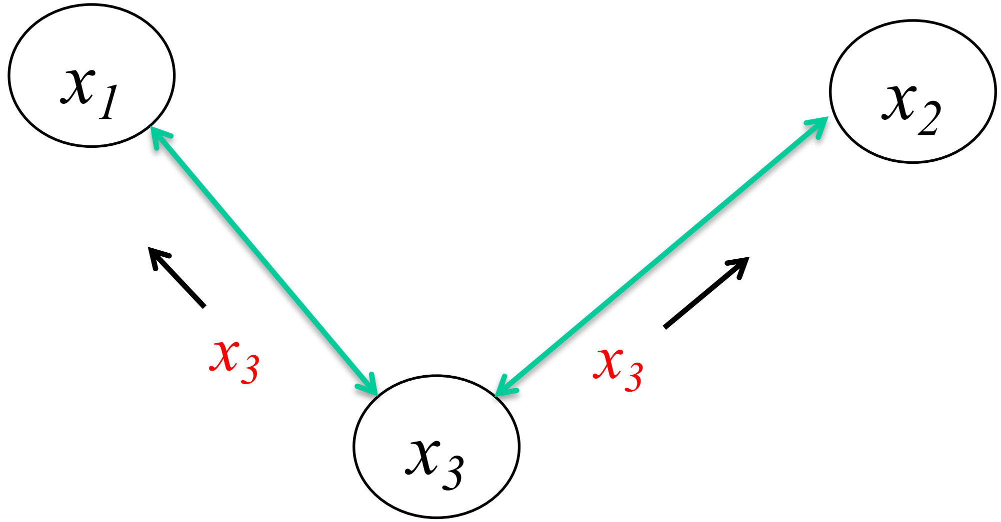


Change of notation

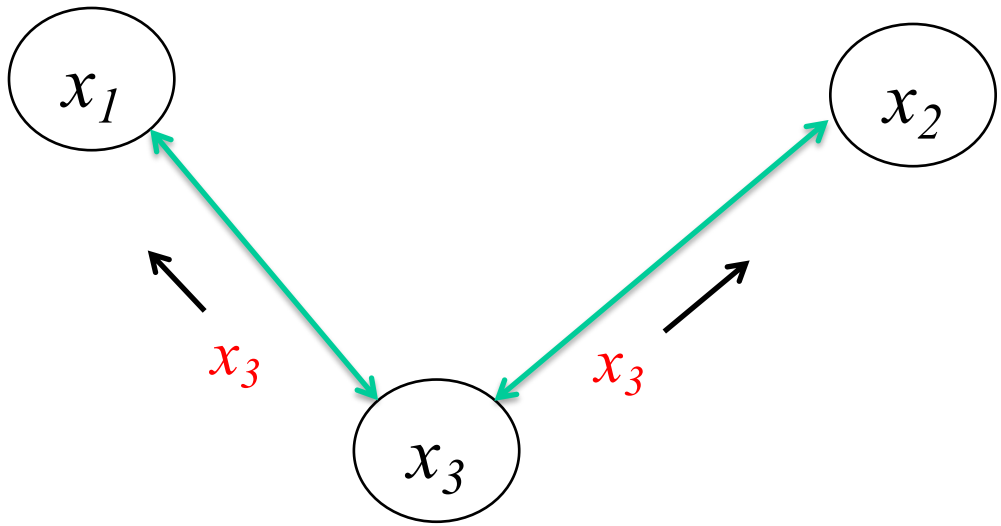
$$x_i[k]$$

agent identifier

Iteration index



$$x_1[t+1] = \frac{2}{3}x_1[t] + \frac{1}{3}x_3[t] - \lambda_t \nabla f_1(x_1[t])$$



$$x_1[t+1] = \boxed{\frac{2}{3}x_1[t] + \frac{1}{3}x_3[t]} - \boxed{\lambda_t \nabla f_1(x_1[t])}$$

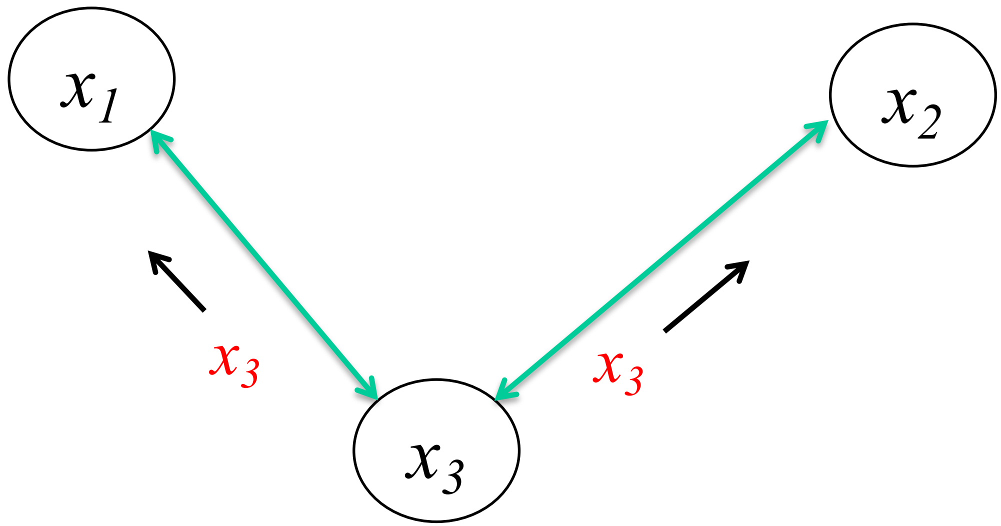
$$x_3[t+1] = \boxed{\frac{1}{3}x_1[t] + \frac{1}{3}x_2[t] + \frac{1}{3}x_3[t]} - \boxed{\lambda_t \nabla f_3(x_3[t])}$$

Decentralized Optimization

In the limit as $t \rightarrow \infty$

- Consensus: All agents converge to same estimate
- Optimality: Estimates converge to identical point in $\operatorname{argmin}_x \sum_i f_i(x)$

Why does this work?



$$x_1[t+1] \longleftarrow \boxed{\frac{2}{3}x_1[t] + \frac{1}{3}x_3[t]} - \boxed{\alpha\,\nabla f_1(x_1[t])}$$

$$x_3[t+1] \longleftarrow \boxed{\frac{1}{3}x_1[t] + \frac{1}{3}x_2[t] + \frac{1}{3}x_3[t]} - \boxed{\alpha\,\nabla f_3(x_3[t])}$$

$$\begin{bmatrix} x_3[t] \\ x_3[t] \\ x_3[t] \end{bmatrix} \qquad \begin{bmatrix} f_1(x_1[t]) \\ f_2(x_2[t]) \\ f_3(x_3[t]) \end{bmatrix}$$

$$x[t+1] \longleftarrow M\, x[t] - \alpha_t \nabla f(x[t])$$

$$x_1[t+1] \longleftarrow \frac{2}{3}x_1[t] + \frac{1}{3}x_3[t] - \alpha_t \nabla f_1(x_1[t])$$

$$x_3[t+1] \longleftarrow \frac{1}{3}x_1[t] + \frac{1}{3}x_2[t] + \frac{1}{3}x_3[t] - \alpha_t \nabla f_3(x_3[t])$$

$$x[t+1] \longleftarrow M\,x[t] - \alpha_t \nabla f(x[t])$$

Doubly
stochastic
M

M here is also doubly stochastic,
but different from
the average consensus example

M identical to that in the
average consensus example
will also suffice

$$x[t+1] \longleftarrow \boxed{M\, x[t]} - \boxed{\alpha_t\, \nabla f(x[t])}$$

Doubly
stochastic
M

$$x[t+1] \longleftarrow \; M\, x[t] - \; \alpha_t\, \nabla f(x[t])$$

$$x[t+1] \longleftarrow \ M\, x[t] - \ \alpha_t \, \nabla f\left(x[t]\right)$$

$$x[1] \longleftarrow \ M\, x[0] - \ \alpha_0 \, \nabla f\left(x[0]\right)$$

$$x[t+1] \longleftarrow \; M\, x[t] - \; \alpha_t\, \nabla f(x[t])$$

$$x[1] \longleftarrow \; M\, x[0] - \; \alpha_0\, \nabla f(x[0])$$

$$x[2] \longleftarrow \; M\, x[1] - \; \alpha_1\, \nabla f(x[1])$$

$$x[t+1] \longleftarrow \; M\,x[t] - \;\alpha_t\,\nabla f(x[t])$$

$$x[1] \longleftarrow \; M\,x[0] - \;\alpha_0\,\nabla f(x[0])$$

$$x[2] \longleftarrow \; M\,x[1] - \;\alpha_1\,\nabla f(x[1])$$

$$= M^2\,x[0] - \;\alpha_0 M\,\nabla f(x[0]) \;-\; \alpha_1\,\nabla f(x[1])$$

$$x[t+1] \longleftarrow \; M\, x[t] - \; \alpha_t \,\nabla f(x[t])$$

$$x[1] \longleftarrow \; M\, x[0] - \; \alpha_0 \,\nabla f(x[0])$$

$$x[2] \longleftarrow \; M\, x[1] - \; \alpha_1 \,\nabla f(x[1])$$

$$= M^2\, x[0] - \; \alpha_0 M \,\nabla f(x[0]) \; - \; \alpha_1 \,\nabla f(x[1])$$

$$x[3] \longleftarrow \; M\, x[2] - \; \alpha_2 \,\nabla f(x[2])$$

$$x[t+1] \longleftarrow \; M\, x[t] - \; \alpha_t\, \nabla f(x[t])$$

$$x[1] \longleftarrow \; M\, x[0] - \; \alpha_0\, \nabla f(x[0])$$

$$x[2] \longleftarrow \; M\, x[1] - \; \alpha_1\, \nabla f(x[1])$$

$$= M^2\, x[0] - \; \alpha_0 M\, \nabla f(x[0]) \; - \; \alpha_1\, \nabla f(x[1])$$

$$x[3] \longleftarrow \; M\, x[2] - \; \alpha_2\, \nabla f(x[2])$$

$$= M^3\, x[0]$$

$$- \alpha_0 M^2 \nabla f(x[0]) \; - \; \alpha_1 M \nabla f(x[1]) - \alpha_2\, \nabla f(x[2])$$

$$x[t+1] \longleftarrow \; M\, x[t] - \; \alpha_t \, \nabla f(x[t])$$

$$x[1] \longleftarrow \; M\, x[0] - \; \alpha_0 \, \nabla f(x[0])$$

$$x[2] \longleftarrow \; M\, x[1] - \; \alpha_1 \, \nabla f(x[1])$$

$$= M^2\, x[0] - \; \alpha_0 M \, \nabla f(x[0]) \; - \; \alpha_1 \, \nabla f(x[1])$$

$$x[3] \longleftarrow \; M\, x[2] - \; \alpha_2 \, \nabla f(x[2])$$

$$= M^3 \, x[0]$$

$$- \alpha_0 M^2 \nabla f(x[0]) \; - \; \alpha_1 M \nabla f(x[1]) - \alpha_2 \, \nabla f(x[2])$$

α_k decreasing with time

Claims

- Estimates at different nodes converge ➔ Consensus
- The estimates converges to argmin $\sum f_i(x)$

Part 3

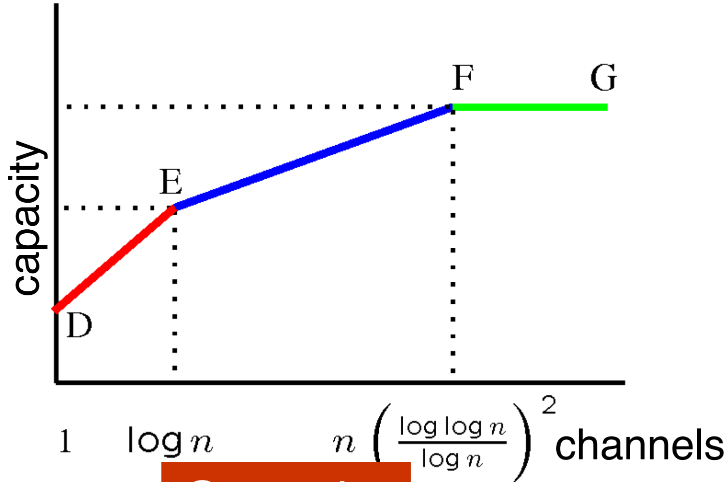
- Byzantine Fault-Tolerant (Secure) Optimization & Learning

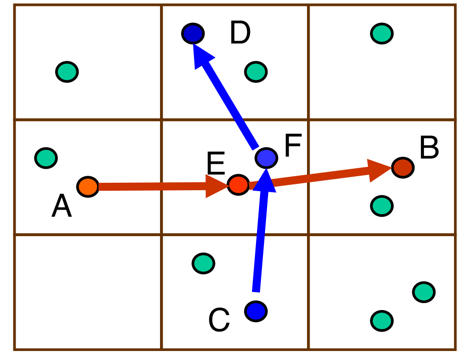
Another Detour …

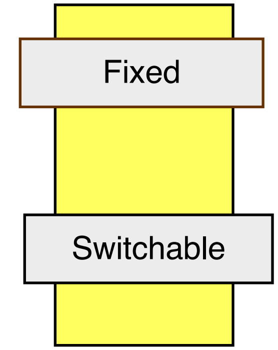
Background

Net-X: Multi-Channel Mesh

Theory to Practice







Capacity bounds

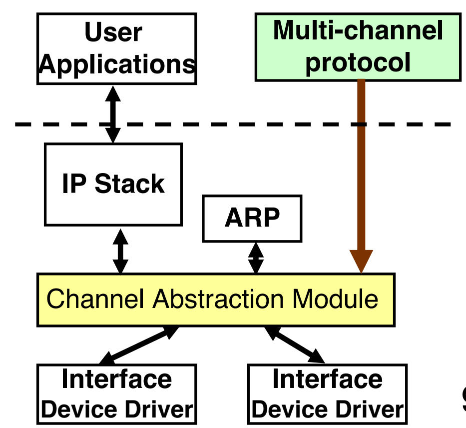
Insights on protocol design

OS improvements
Software architecture

Net-X testbed

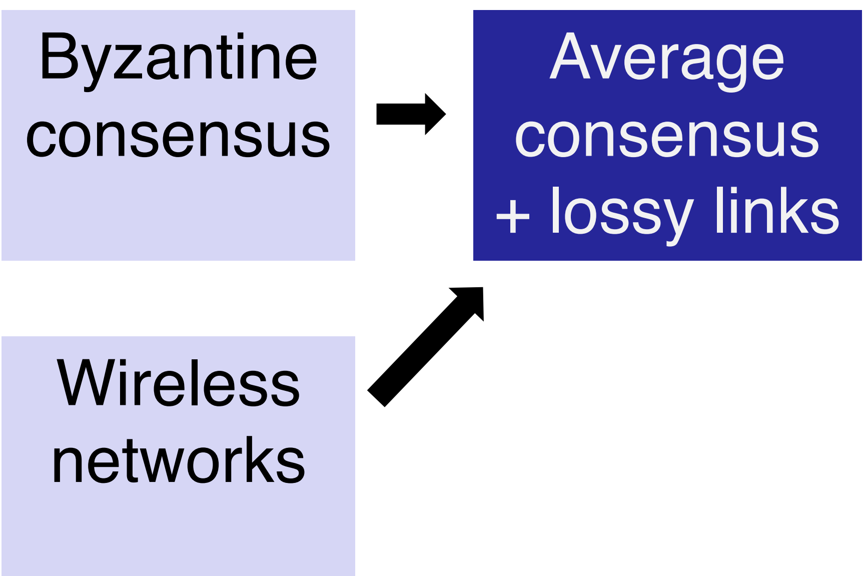
Linux box

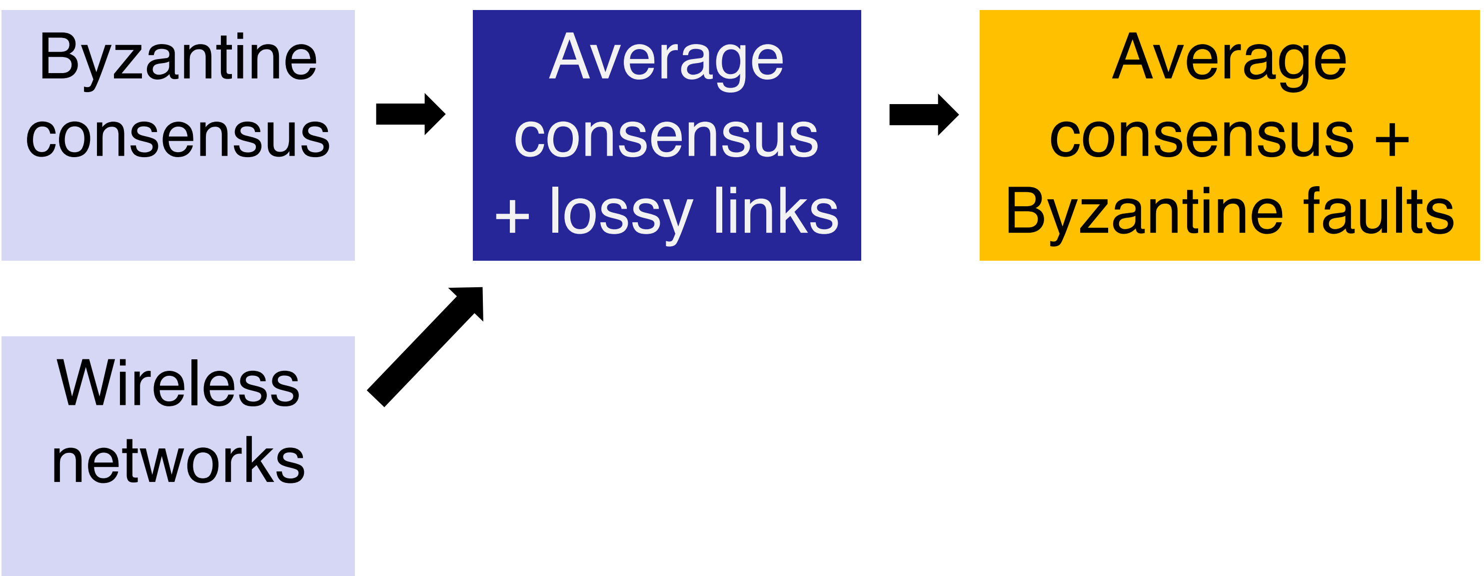
CSL

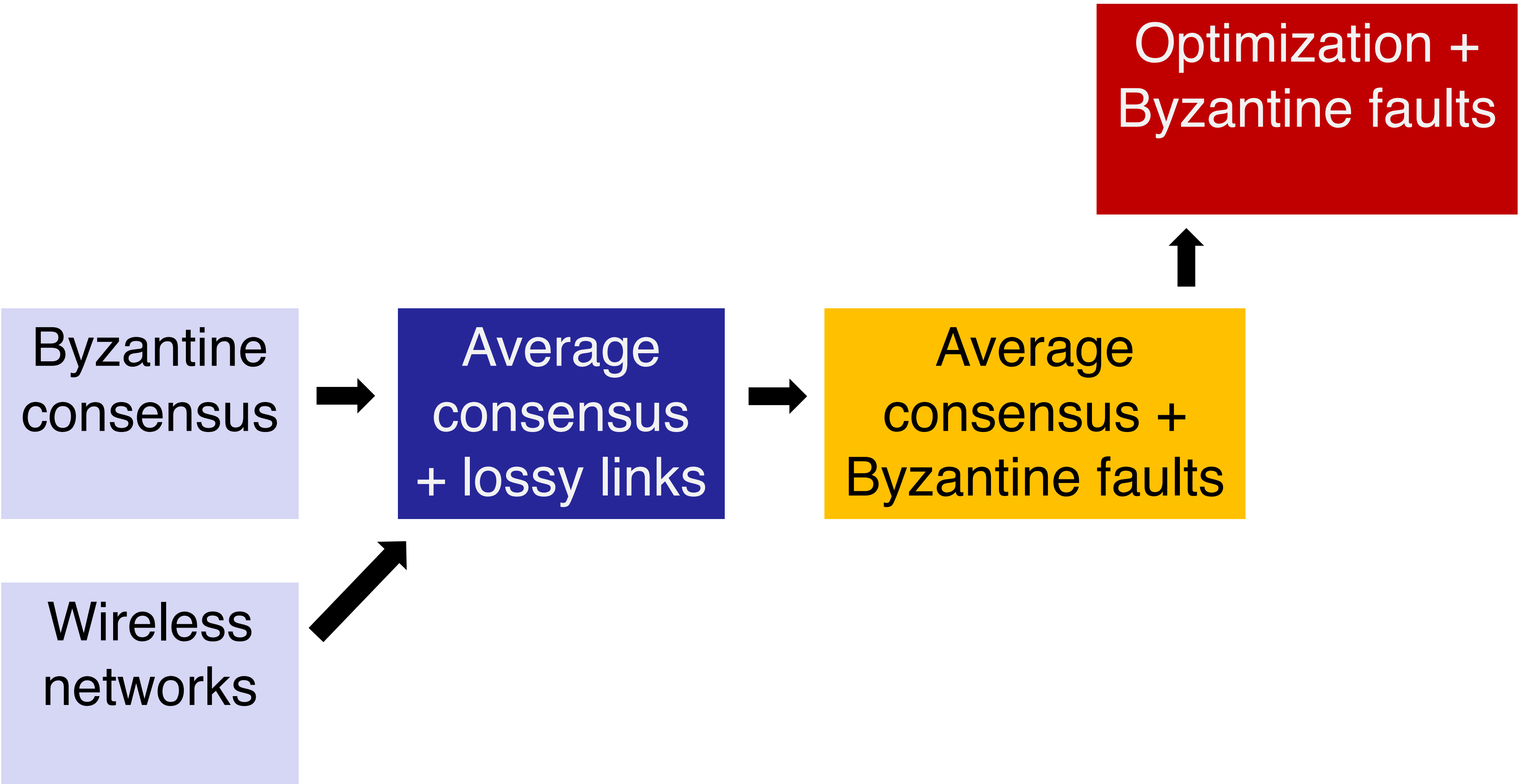


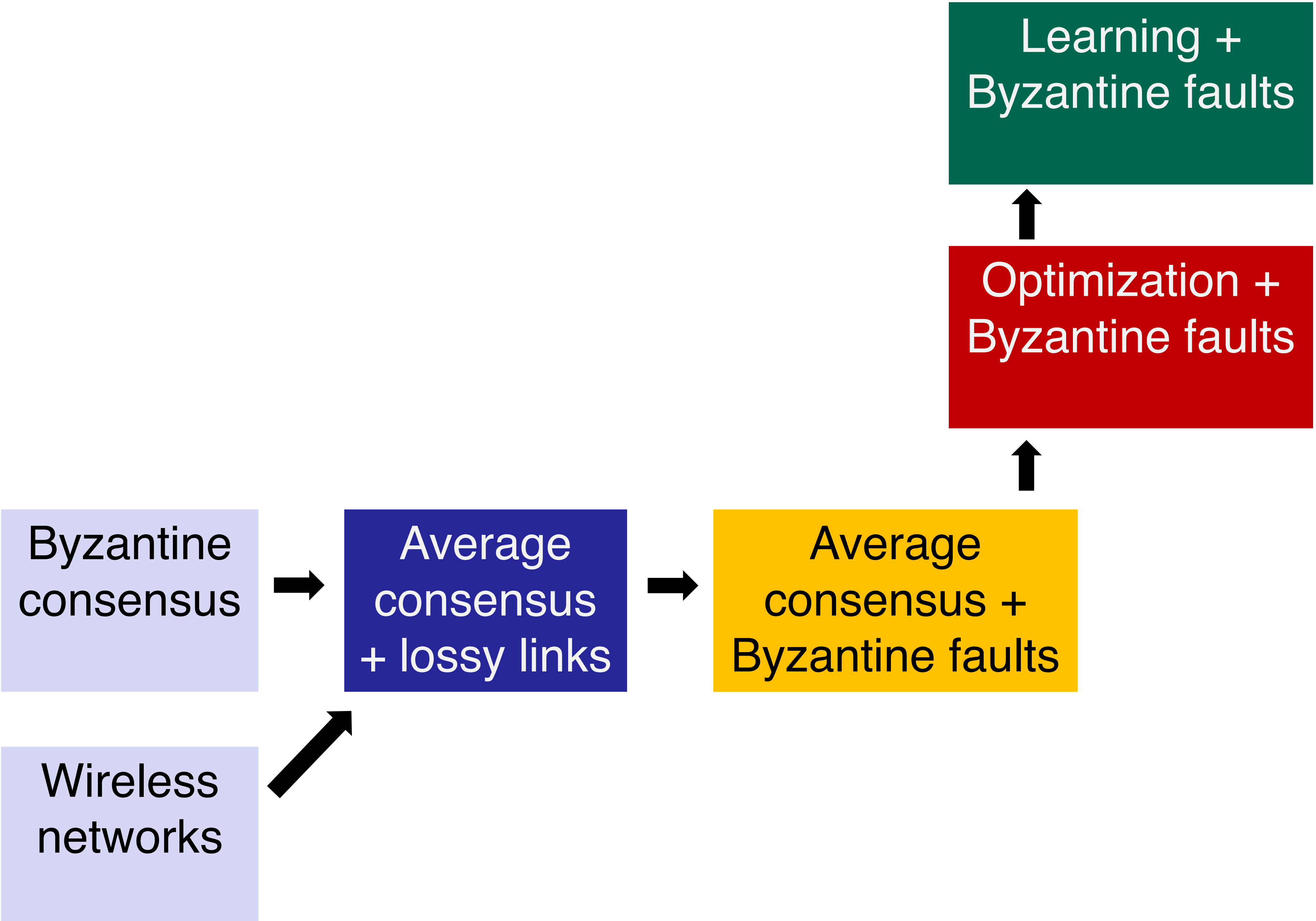
How do you get from

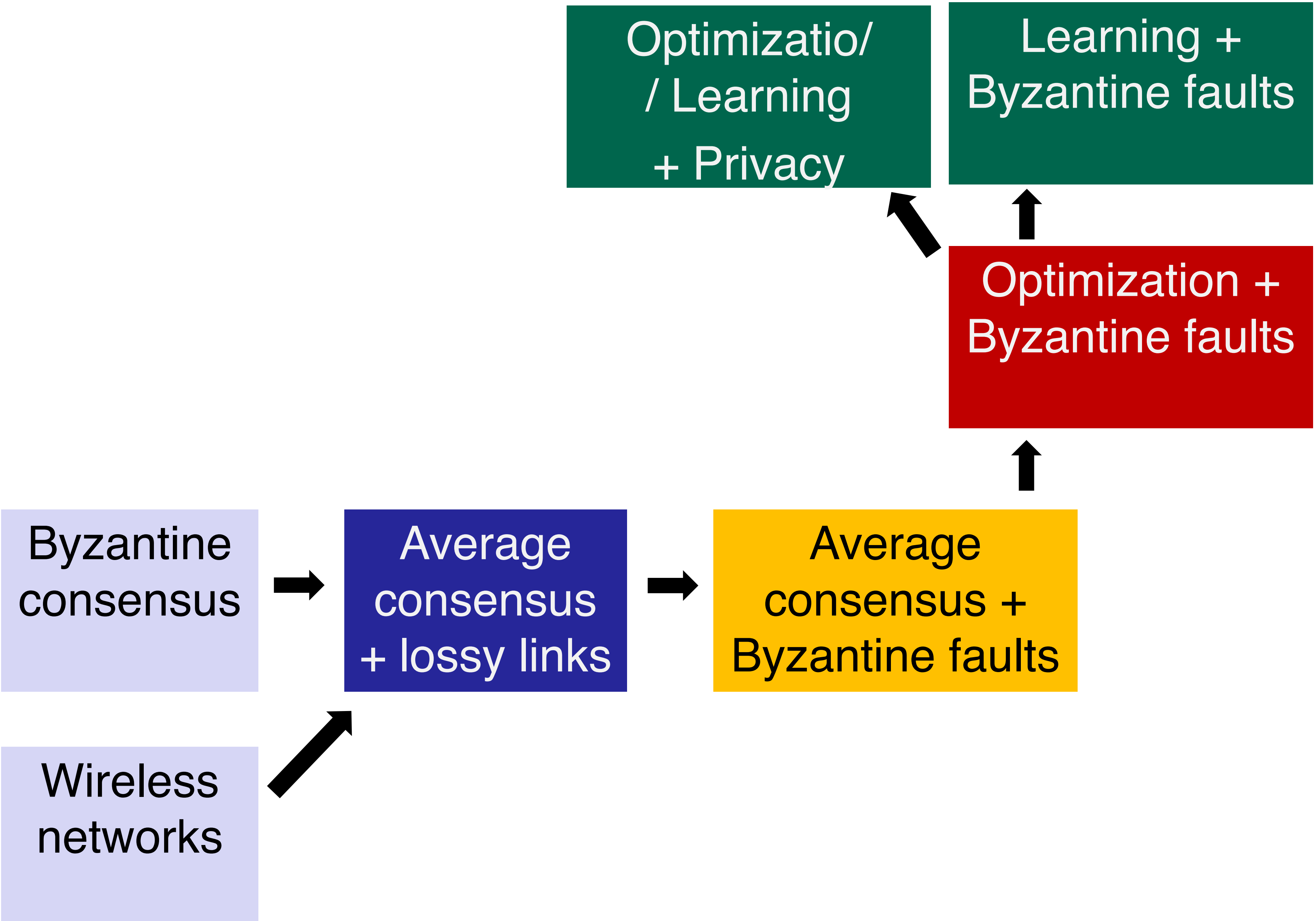
wireless systems to distributed optimization/learning?


Byzantine consensus
Average consensus + lossy links
Wireless networks


Byzantine consensus
Wireless networks
Average consensus + lossy links
Average consensus + Byzantine faults


Optimization + Byzantine faults
Byzantine consensus
Average consensus + lossy links
Average consensus + Byzantine faults
Wireless networks


Learning + Byzantine faults
Optimization + Byzantine faults
Byzantine consensus
Average consensus + lossy links
Average consensus + Byzantine faults
Wireless networks


Byzantine consensus
Wireless networks
Average consensus + lossy links
Average consensus + Byzantine faults
Optimization + Byzantine faults
Optimizatio/ / Learning + Privacy
Learning + Byzantine faults

Hajnal 1958

Weak ergodicity of nonhomogeneous Markov chains

Distributed Computing

DeGroot 1974

Reaching a consensus

1980: Pease, Shostak, Lamport

Byzantine consensus

1983: Fischer, Lynch, Paterson

Asynchronous consensus impossibility result

Decentralized Control

Tsitsiklis 1984

1986: Dolev et al.

Approximate Byzantine consensus

Jadbabaei, Lin, Morse 2003

Flocking problem

Nedich, Ozdaglar 2009

Continue to part 3

Byzantine Fault-Tolerant (Secure)
Optimization & Learning

Additional Slides

Connected Undirected Graphs

after k iterations

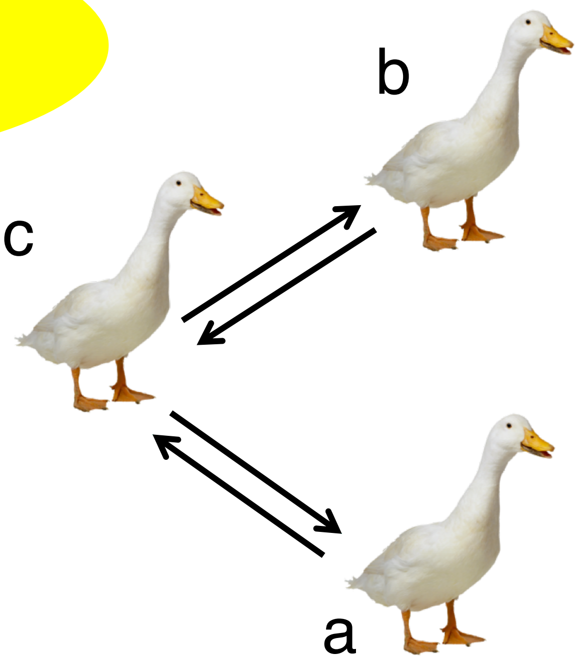
$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

- Consensus if M row stochastic
  - Matrix elements in [0,1]
  - M_{ij} non-zero if link (i,j) exists
  - Each row adds to 1

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix} \rightarrow \begin{pmatrix} p & q & r \\ p & q & r \\ p & q & r \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Row stochastic M

$c = a/4 + b/4 + c/2$



$b = 3b/4 + c/4$

$a = 3a/4 + c/4$

Row
stochastic M

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := \color{blue}{M} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

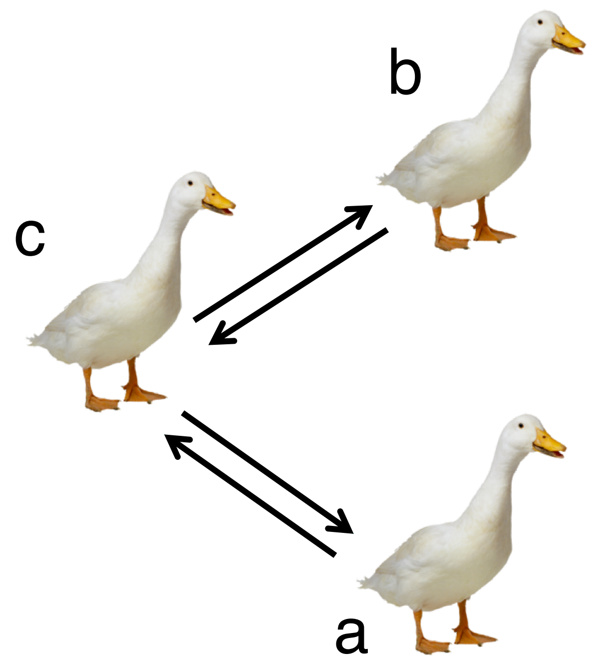
Due to stochastic rows,
each new state
in convex hull
of old states

Decentralized Optimization over Lossy Links

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := \underbrace{\begin{pmatrix} 3/4 & 0 & 1/4 \\ 0 & 3/4 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}}_{\mathrm{M}} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \mathrm{M} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$b = 3b/4 + c/4$

$c = a/4 + b/4 + c/2$



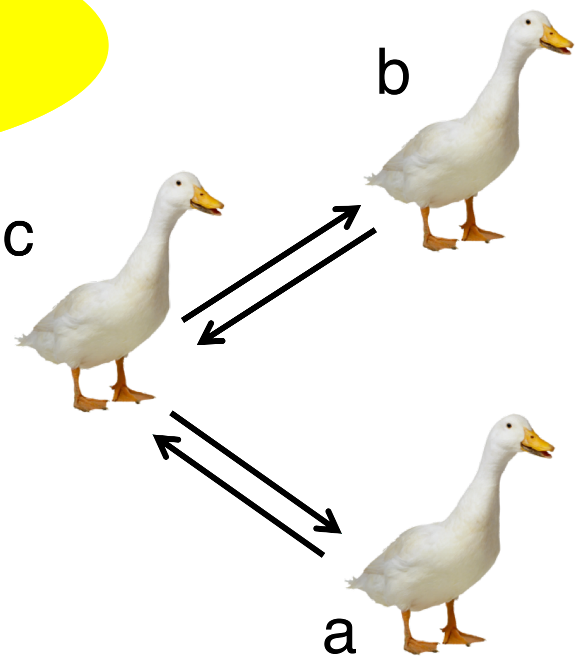
$a = 3a/4 + c/4$

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} := \mathrm{M}^{k} \begin{pmatrix} a \\ b \\ c \end{pmatrix} \rightarrow \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Doubly stochastic M

$b = 3b/4 + c/4$

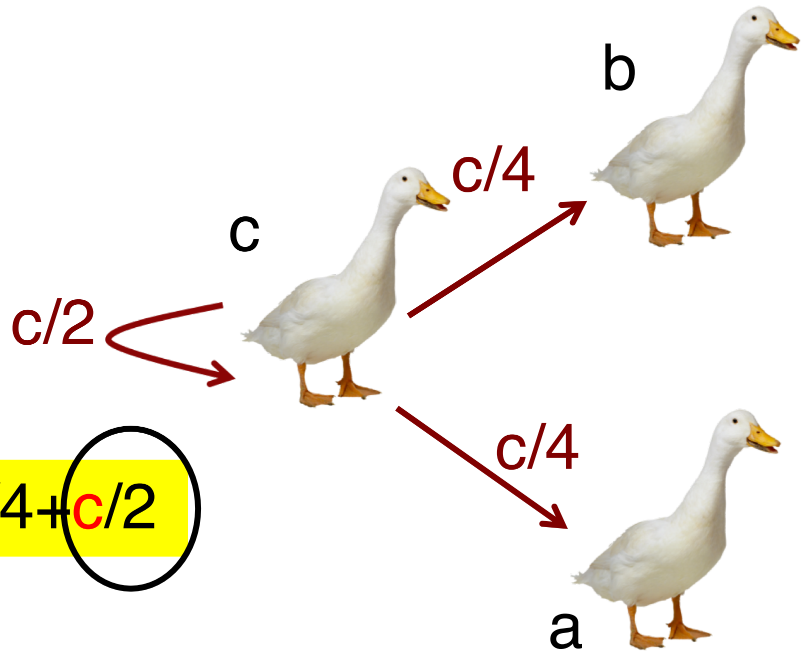
$c = a/4 + b/4 + c/2$



$a = 3a/4 + c/4$

Mass Transfer + Accumulation
An Alternate View

- Each node "transfers mass" to neighbors via messages
- Next state = Total received mass



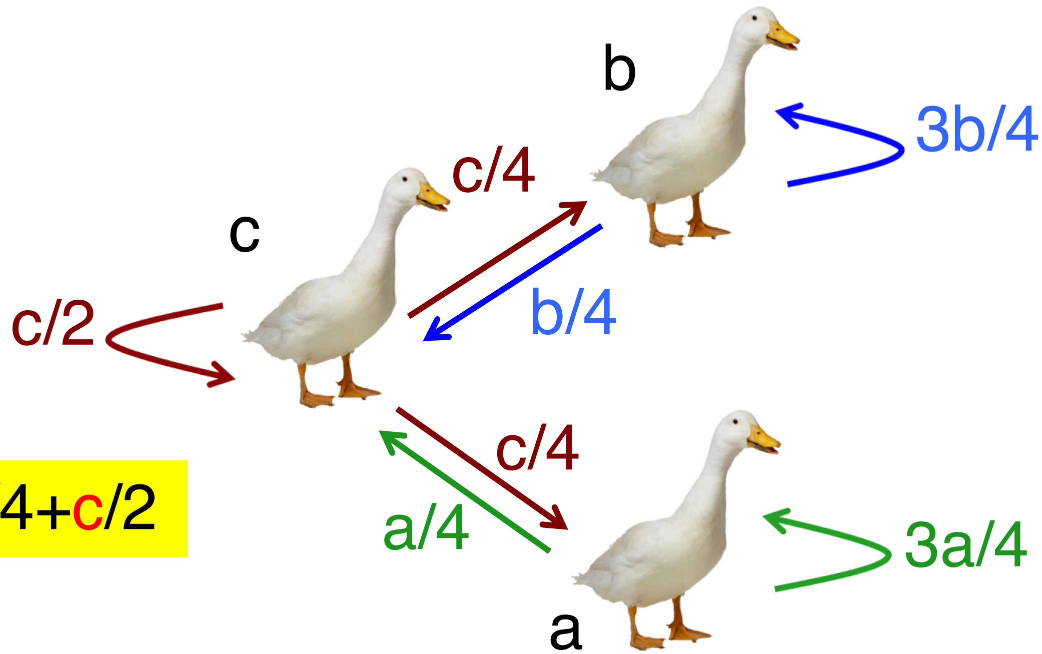
$b = 3b/4 + c/4$

$c = a/4 + b/4 + c/2$

$a = 3a/4 + c/4$

Mass Transfer + Accumulation
An Alternate View

- Each node "transfers mass" to neighbors via messages
- Next state = Total received mass



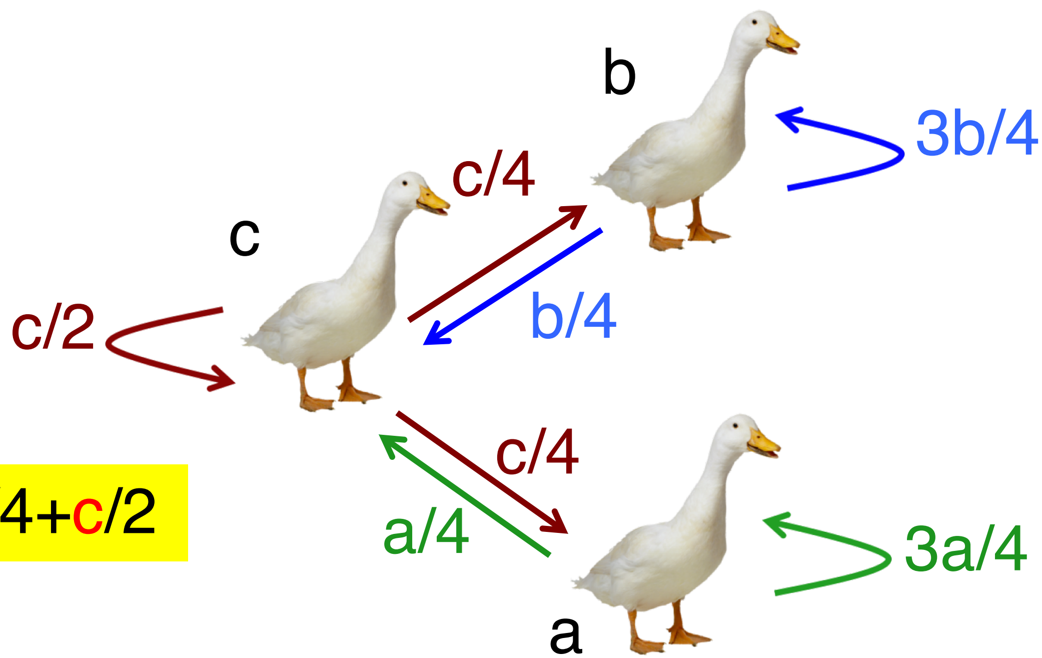
$b = 3b/4 + c/4$

$c = a/4 + b/4 + c/2$

$a = 3a/4 + c/4$

Conservation of Mass

- a+b+c constant after each iteration



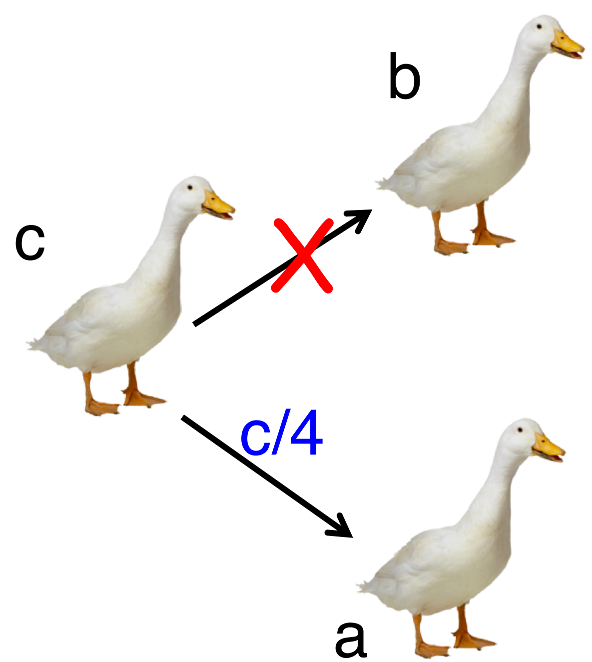
$b = 3b/4 + c/4$

$c = a/4 + b/4 + c/2$

$a = 3a/4 + c/4$

Wireless Transmissions Unreliable

$c = a/4 + b/4 + c/2$



$b = 3b/4 + \cancel{c/4}$

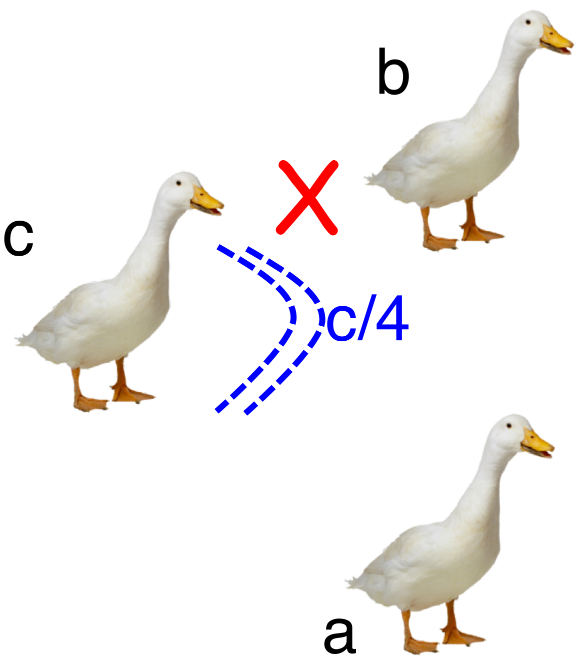
$a = 3a/4 + c/4$

Impact of Unreliability

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 3/4 & 0 & 1/4 \\ 0 & 3/4 & 0 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$b = 3b/4 + \cancel{c/4}$

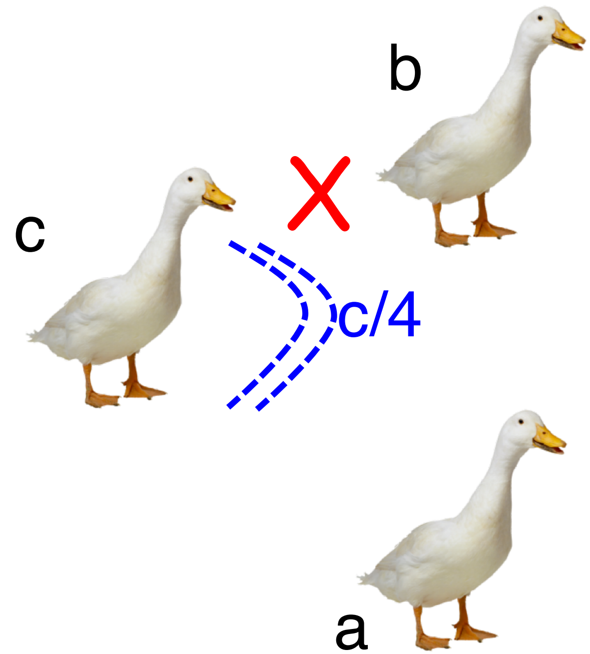
$c = a/4 + b/4 + c/2$



$a = 3a/4 + c/4$

Conservation of Mass

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 3/4 & 0 & 1/4 \\ 0 & 3/4 & 0 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$



$b = 3b/4 + \cancel{c/4}$

$c = a/4 + b/4 + c/2$

$a = 3a/4 + c/4$

Average consensus over lossy links ?

Potential Solution?

Assume that

transmitter KNOWS

when a message is not delivered

Potential Solution?

When mass not transferred to neighbor,

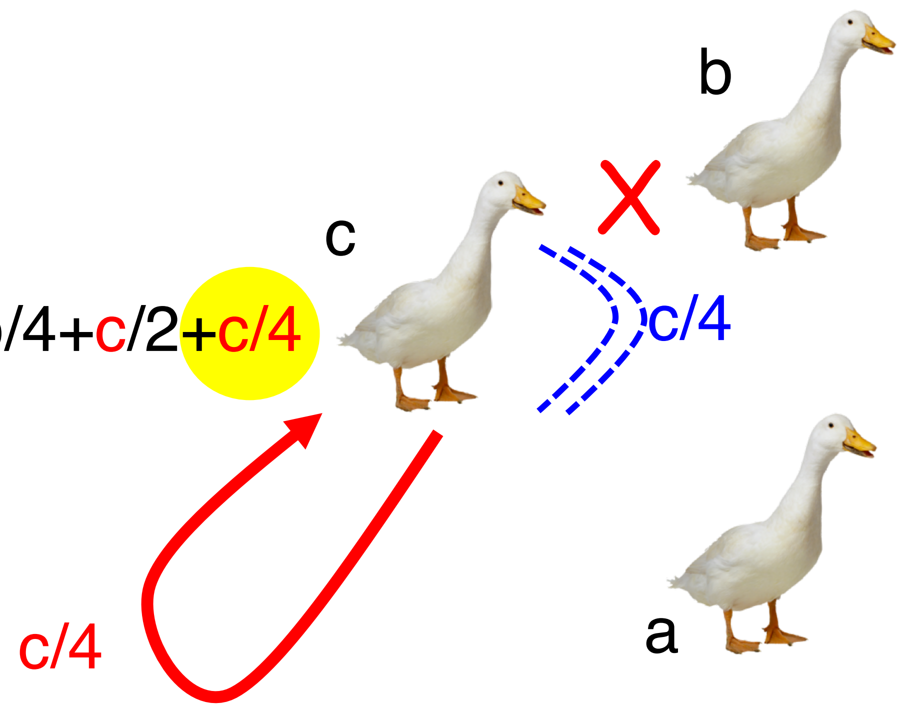
keep it to yourself

Convergence … if nodes intermittently connected

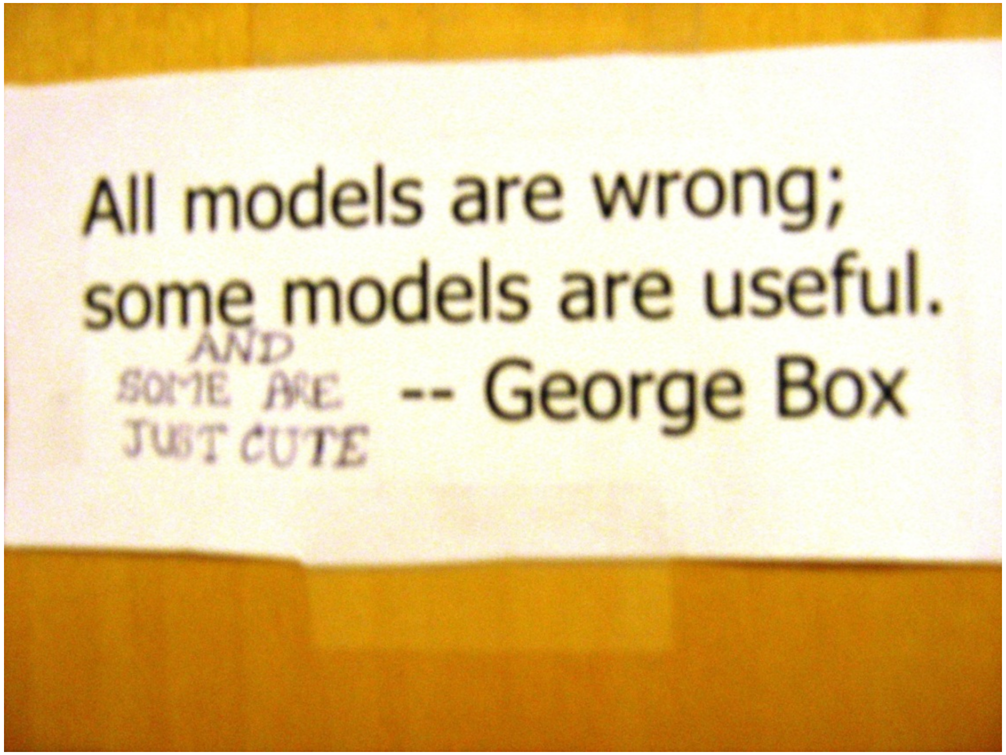
$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 3/4 & 0 & 1/4 \\ 0 & 3/4 & 0 \\ 1/4 & 1/4 & 3/4 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$b = 3b/4 + \cancel{c/4}$

$c = a/4+b/4+c/2+c/4$



$a = 3a/4 + c/4$


All models are wrong;
some models are useful.
AND SOME ARE JUST CUTE
-- George Box

Loss Model

Assume that

transmitter ~~KNOWS~~

when a message is not delivered

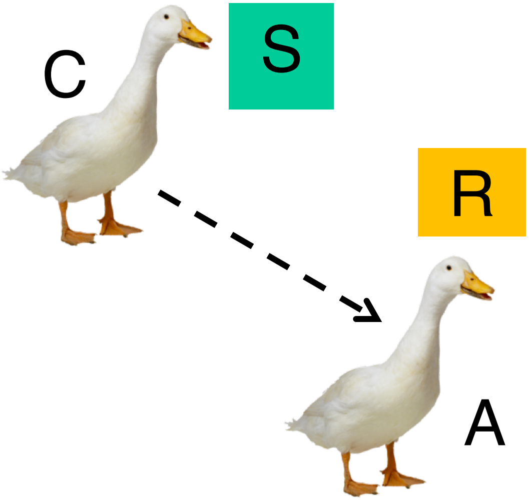
Better Model ?

No common knowledge regarding message delivery

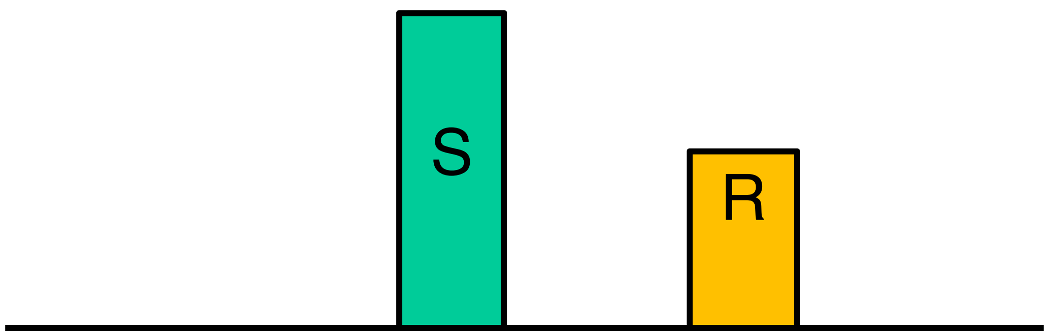
Solution

- Introduce memory

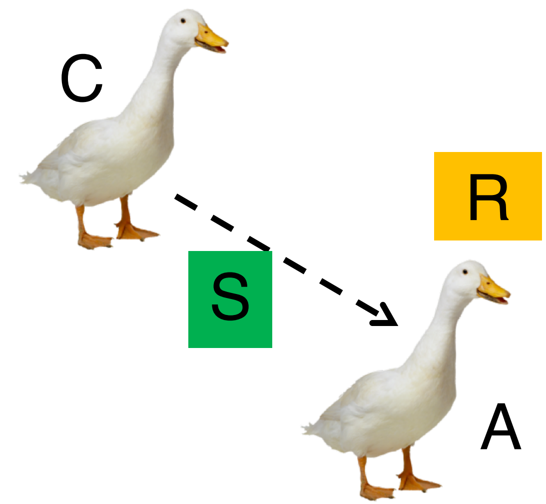
Solution Sketch



- S = mass C wanted to transfer to node A in total so far
- R = mass A has received from node C in total so far



Solution Sketch

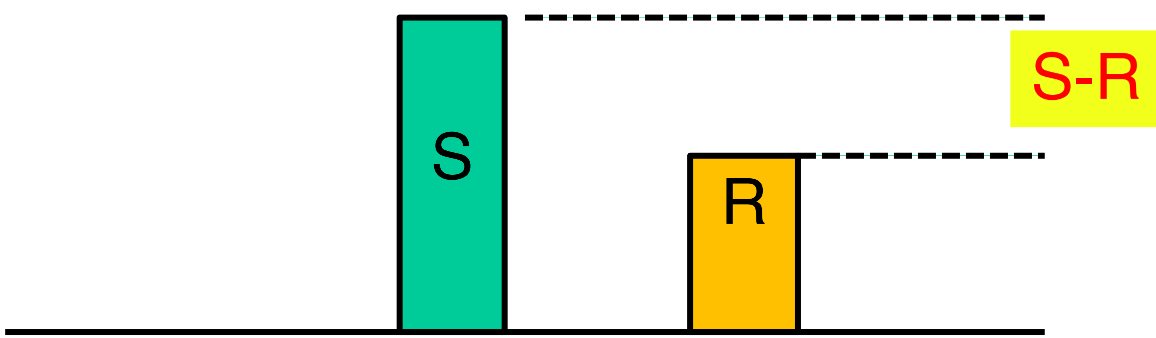


- Node C transmits quantity S

  …. message may be lost

- When it is received,

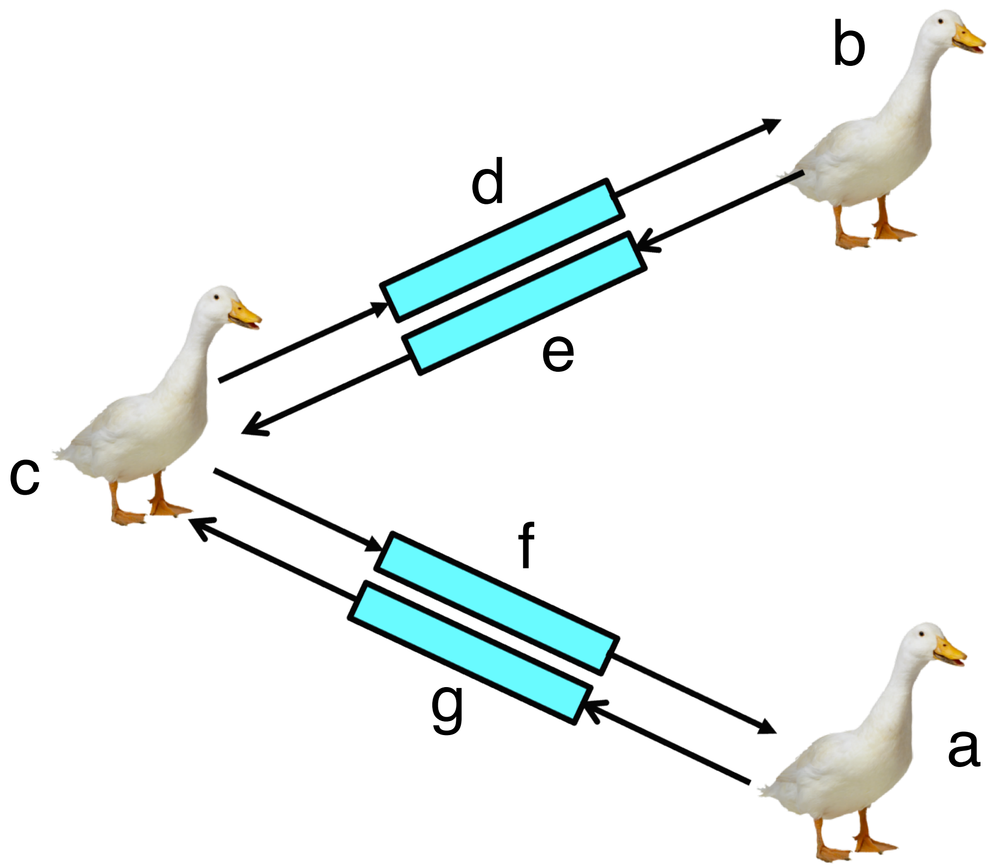
  node A accumulates (S-R)



What Does That Do ?

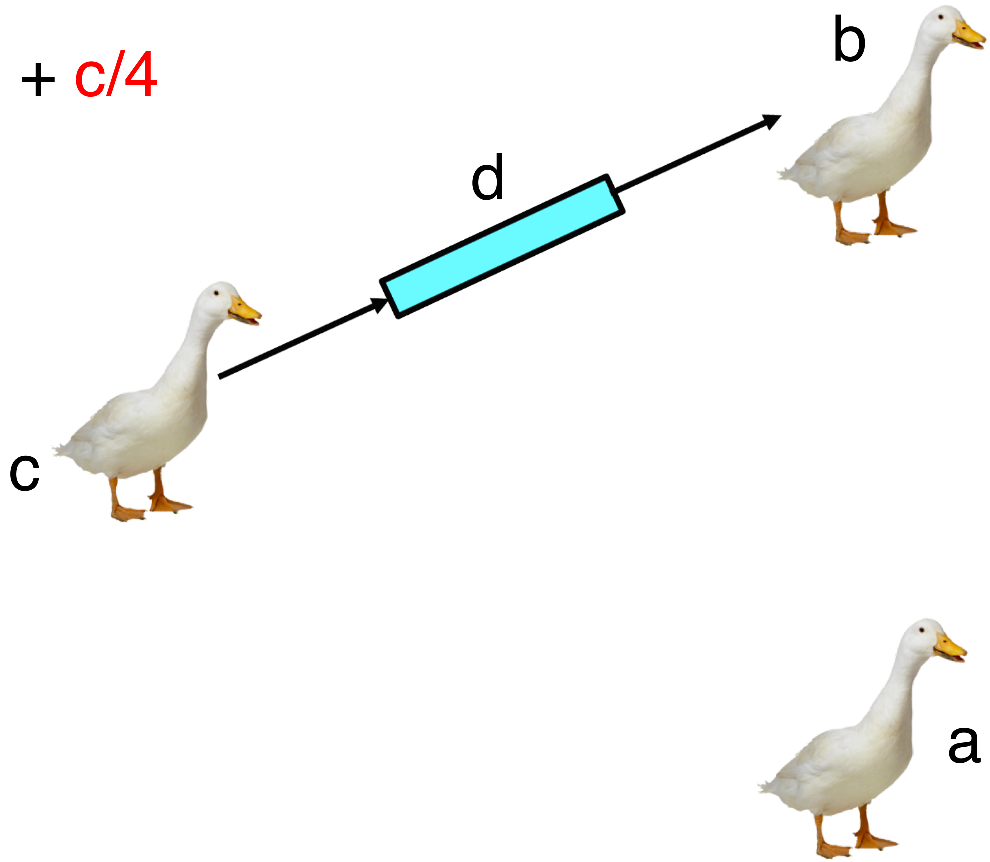
What Does That Do ?

- Implements virtual buffers



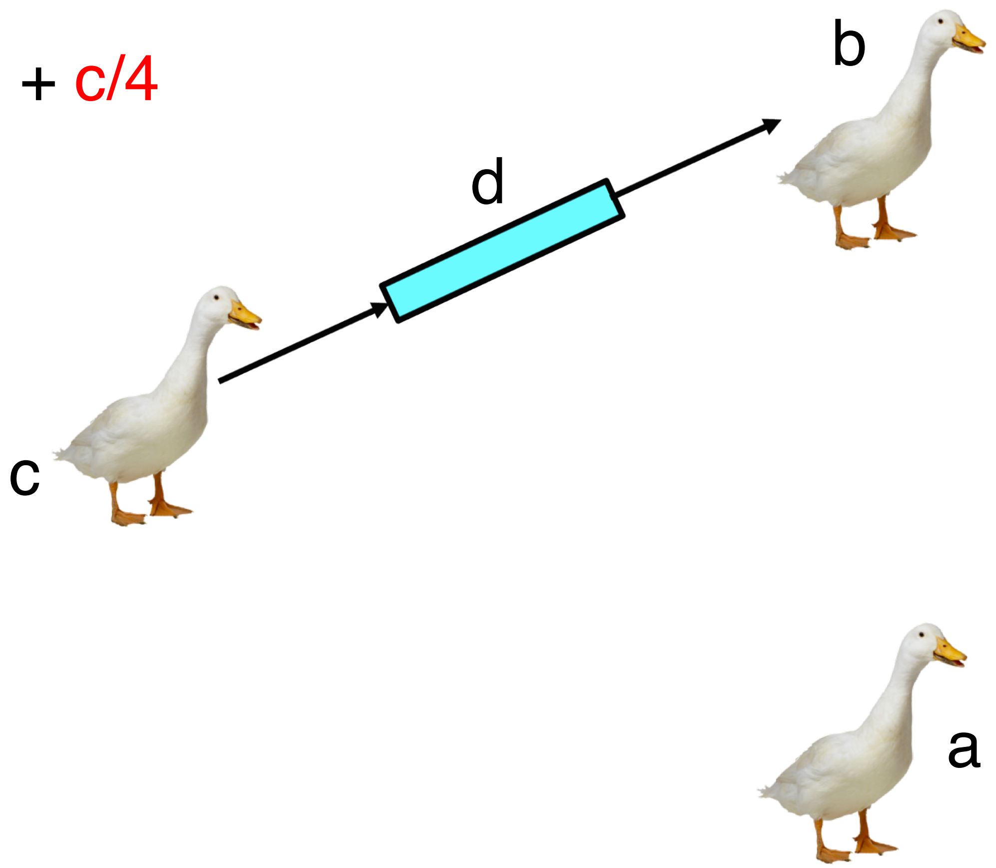
Dynamic Topology

- When C→B transmission unreliable, mass transferred to buffer (d)
- d = d + c/4



Dynamic Topology

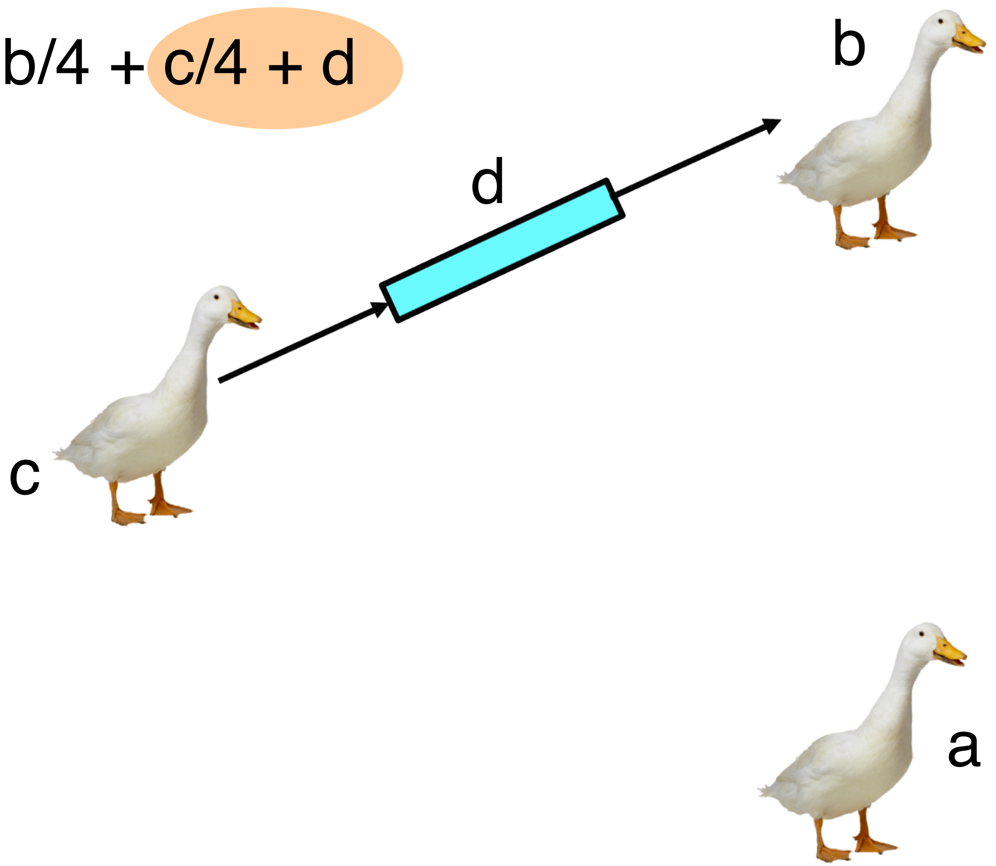
- When C→B transmission unreliable, mass transferred to buffer (d)
- d = d + c/4



No loss of mass even with message loss

Dynamic Topology

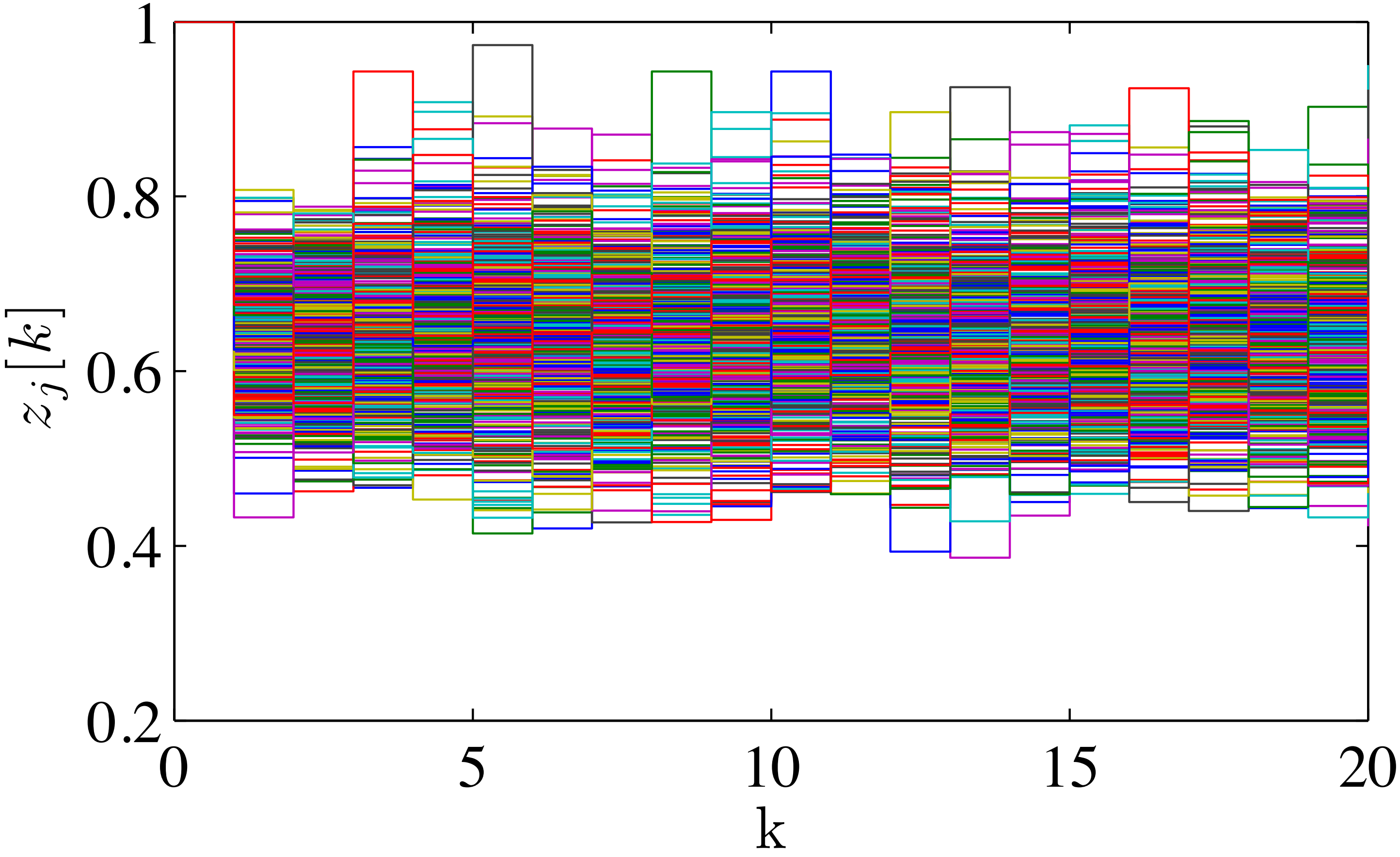
- When C→B transmission reliable, mass transferred to b
- $b = 3b/4 + c/4 + d$



No loss of mass even with message loss

Does This Work ?

Does This Work ?



Time-Varying Column Stochastic Matrix

- Mass is conserved
- Time-varying network

  ➔ Matrix varies over iterations

Matrix M_i for i-th iteration

State Transitions

- x = state vector = $\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{pmatrix}$

- x[0] = initial state vector

- x[t] = iteration t

State Transitions

- $x[1] = M_1\, x[0]$
- $x[2] = M_2\, x[1] = M_2\, M_1\, x[0]$

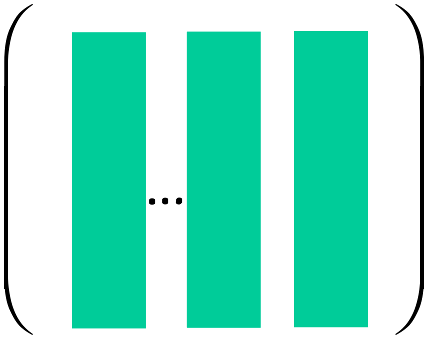
…

- $x[t] = M_k\, M_{k-1} \ldots M_2\, M_1\, x[0]$

State Transitions

- $x[t] = M_k \, M_{k-1} \, \ldots \, M_2 \, M_1 \, x[0]$

Matrix product converges to
column stochastic matrix with <u>identical</u> columns



State Transition

After k iterations

$$\begin{pmatrix} p & & p & p \\ q & \cdots & q & q \\ r & & r & r \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{pmatrix}$$

---

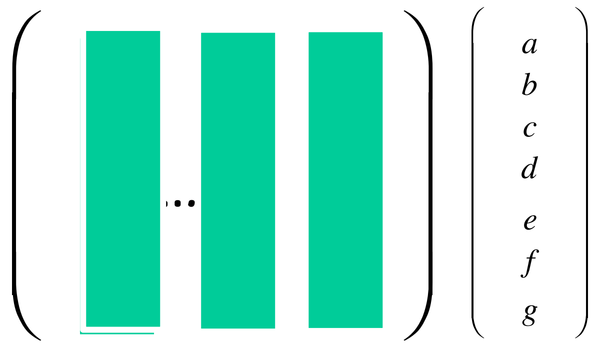
k+1

$$M_{k+1} \begin{pmatrix} & \cdots & & \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{pmatrix}$$

$$= \begin{pmatrix} z & & z & z \\ & \cdots & & \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{pmatrix}$$

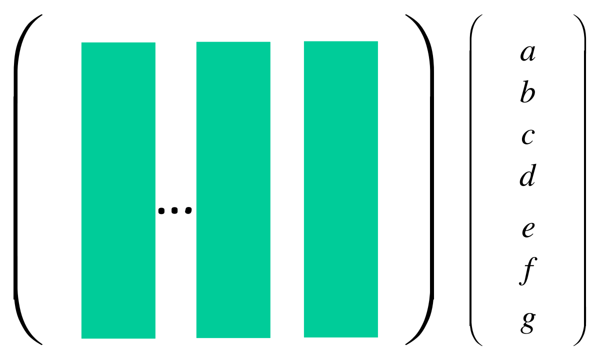
State Transition

After k iterations

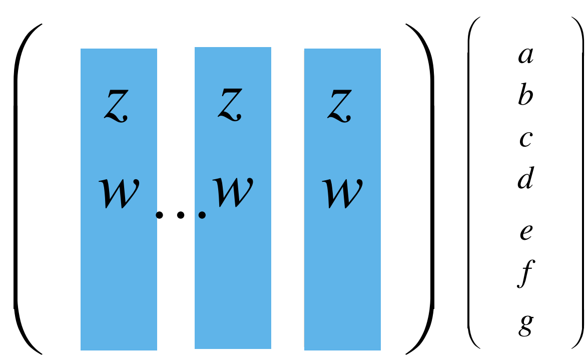


k+1

$\mathbf{M}_{k+1}$



=



z * sum

w * sum

State Transitions

- After k iterations, state of first node has the form

  z(k) * sum of inputs

  where z(k) changes each iteration (k)

- Does <u>not</u> converge to average

Solution

- Run two iterations in parallel
  - First : original inputs
  - Second : input = 1

Solution

- Run two iterations in parallel
  - First : original inputs
  - Second : input = 1
- After k iterations …

first algorithm: $z(k)$ * sum of inputs
second algorithm: $z(k)$ * number of nodes

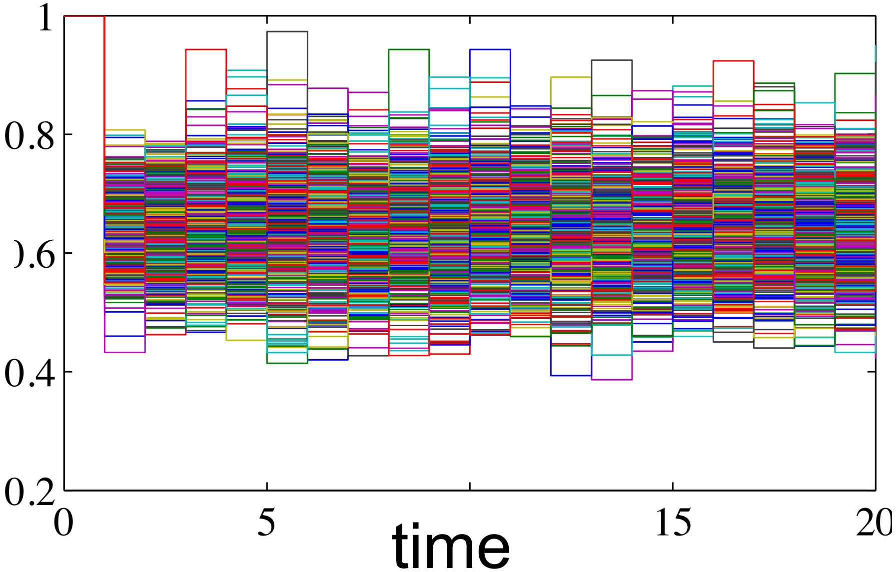
Solution

- Run two iterations in parallel
  - First : original inputs
  - Second : input = 1
- After k iterations …

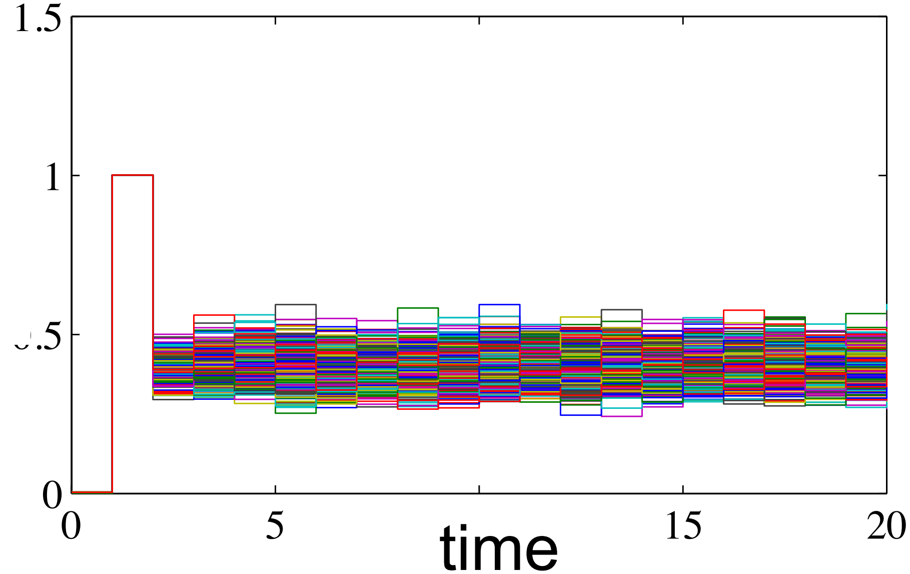
$$\frac{\text{first algorithm: } z(k) * \text{sum of inputs}}{\text{second algorithm: } z(k) * \text{number of nodes}}$$

ratio = **average**

numerator



denominator



ratio