A Journey from
Wireless Networks to Distributed Optimization

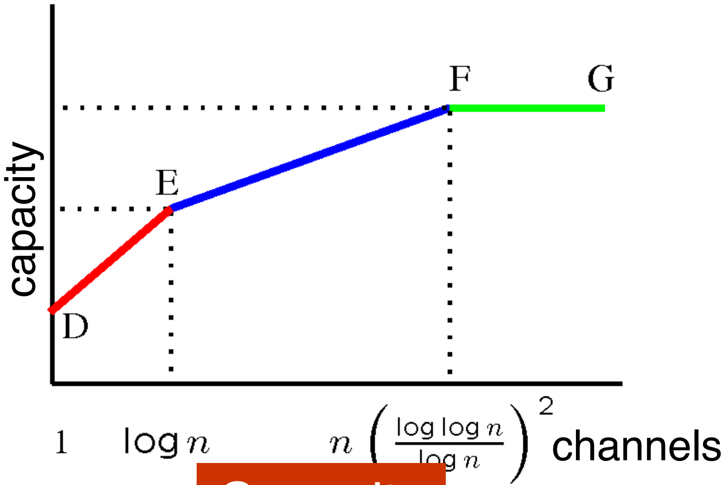
Nitin Vaidya
Georgetown University

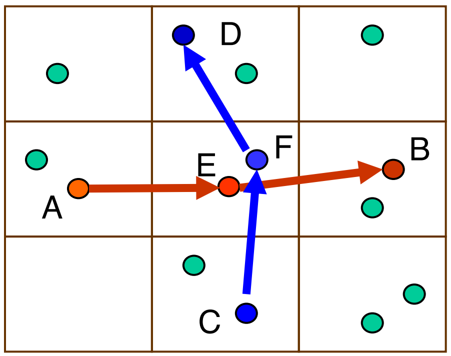
Net-X

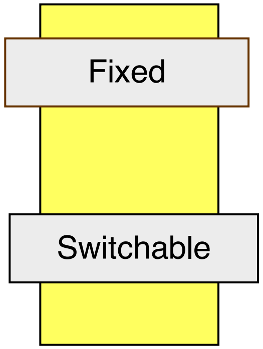
Multi-Channel Mesh

Theory to Practice

(2006)







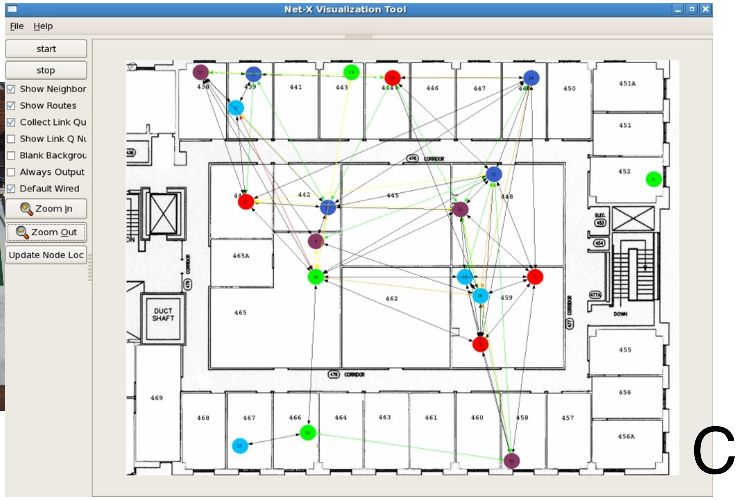
Capacity bounds

Insights on protocol design

OS improvements Software architecture

Net-X testbed

Linux box



Net-X Visualization Tool

File Help

start

stop

☑ Show Neighbor

☑ Show Routes

☑ Collect Link Qu

□ Show Link Q Nu

□ Blank Backgrou

□ Always Output

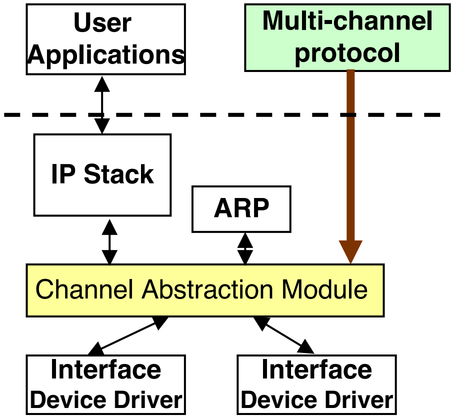
☑ Default Wired

Zoom In

Zoom Out

Update Node Loc

CSL



More Recently …

Fault-Tolerance in Distributed Optimization: The Case of Redundancy

Preserving Statistical Privacy in Distributed Optimization

Byzantine Consensus with Local Multicast Channels

From there to here …

- Through a few short-term, somewhat accidental, interactions

- I will discuss one example

Takeaway …

Moral of the Story #1

Natural, unanswered questions at the intersection of previously explored problem spaces

Picture from Wikipedia

Moral of the Story #2

Academia lets you work on things for which you may have no competence

Make the best use of the freedom

A Journey from
Wireless Networks to Distributed Optimization

Consensus

Consensus, consensus, everywhere …

Consensus, consensus …

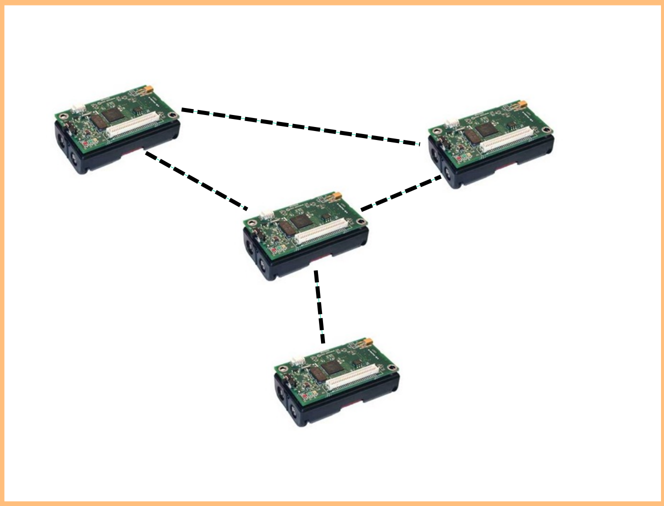
- Commit or abort ?

- Network of databases …

agree on a common action

Consensus, consensus …

- What is the temperature?



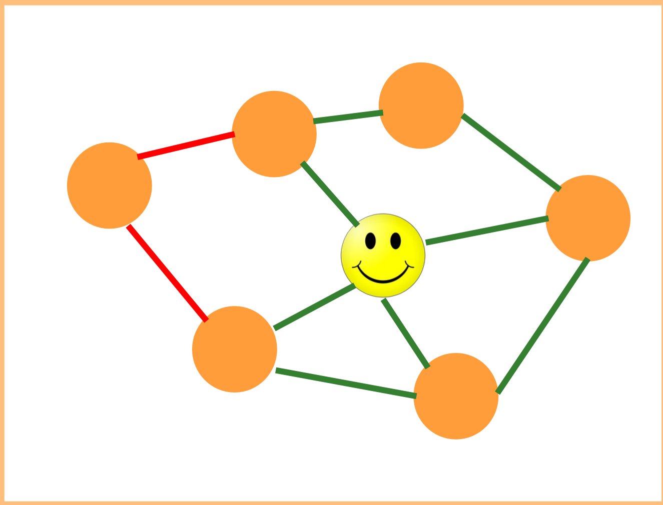
- Network of sensors …

agree on current temperature

Consensus, consensus …

- Should we trust ☺ ?

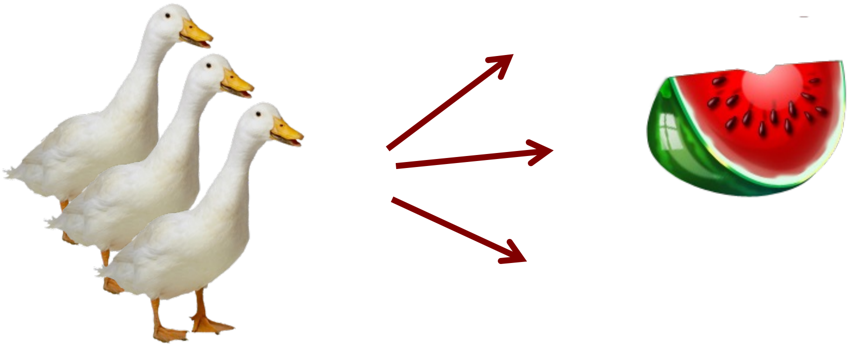
- Web of trust …



agree whether ☺ is good or evil

Consensus, consensus …

- Which way?

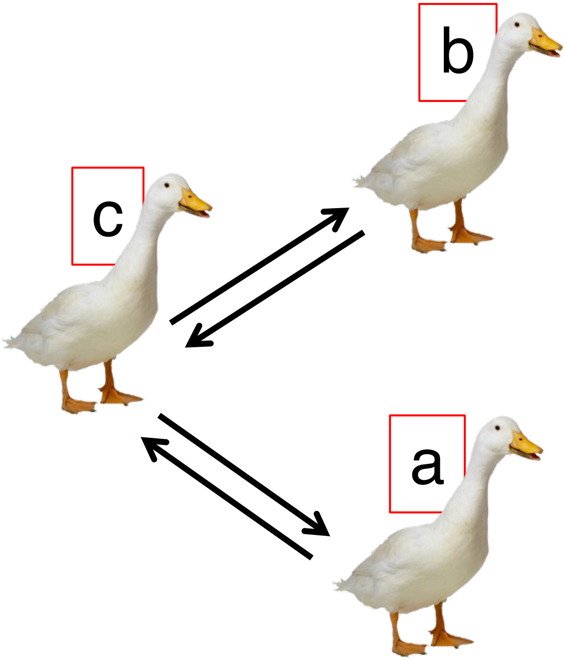


Consensus

“Local” Algorithms

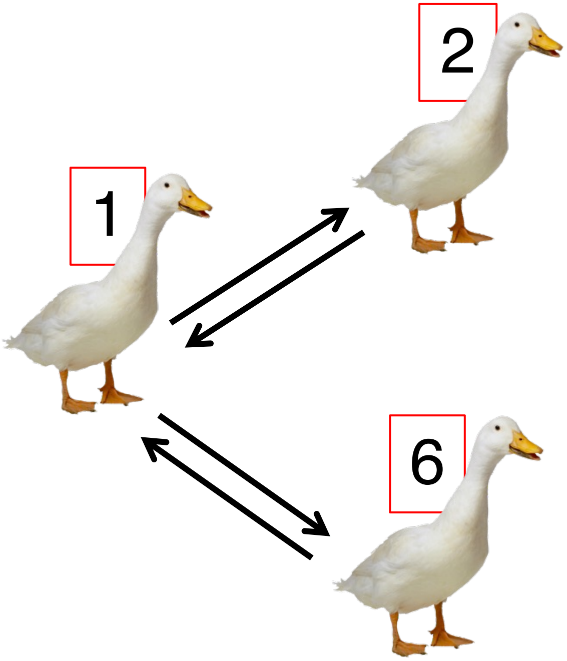
Consensus … Local Averaging

Initially, state = input



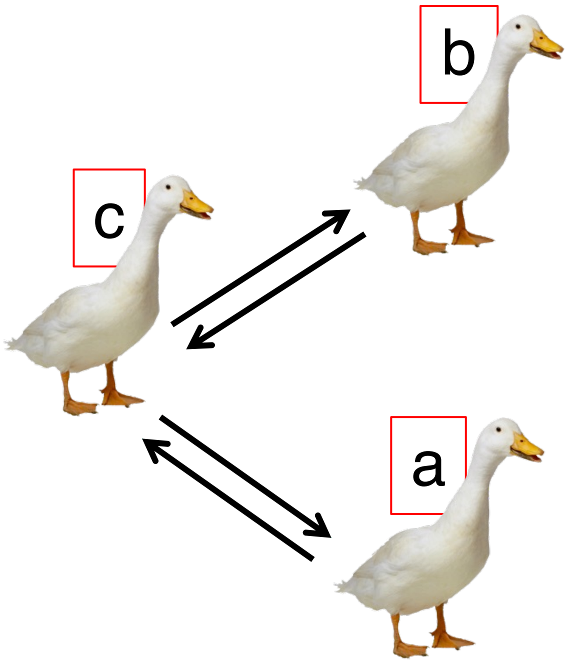
Consensus … Local Averaging

Initially, state = input



Consensus … Local Averaging

Initially, state = input

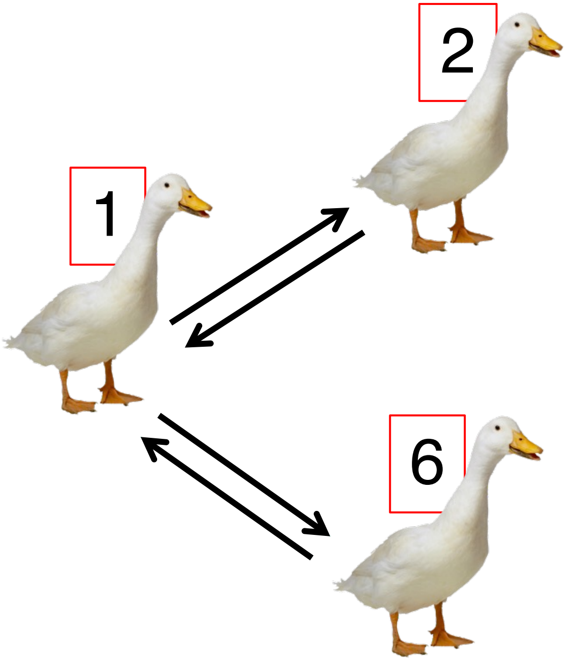


$b = (b+c)/2$

$c = (a+b+c)/3$

$a = (a+c)/2$

Consensus … Local Averaging

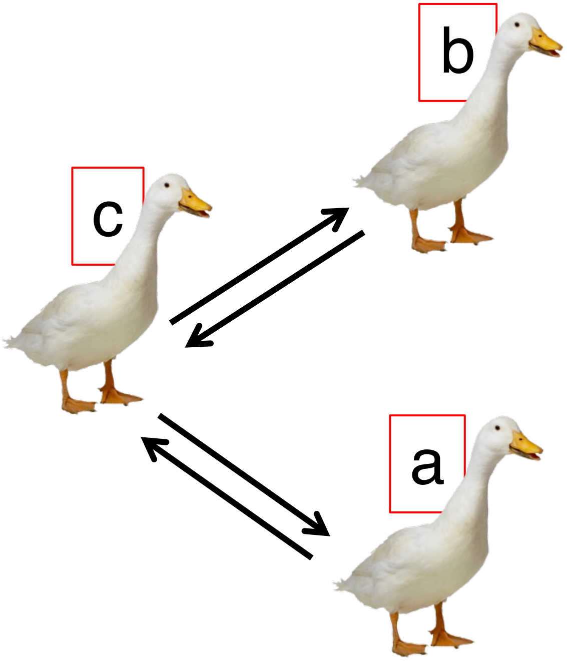


b = (1+2)/2 = 3/2

c = (1+2+6)/3 = 3

a = (1+6)/2 = 7/2

$$\underset{\text{new values}}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}} = \underset{\text{M}}{\begin{pmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}} \underset{\text{old values}}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}$$



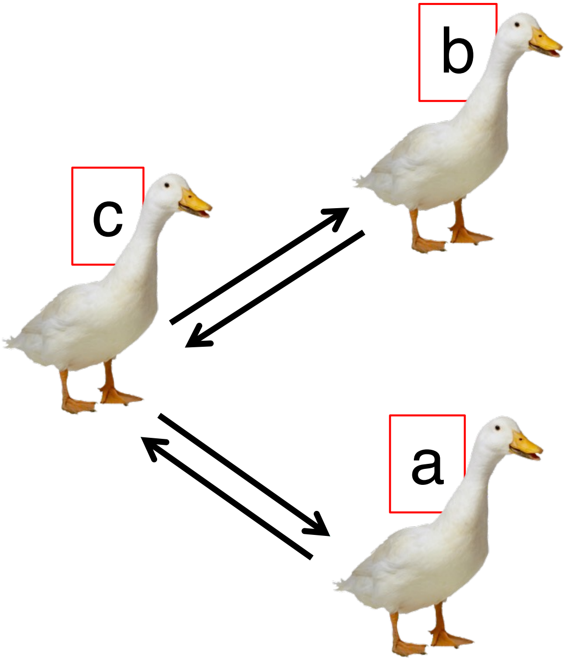
b = (b+c)/2

c = (a+b+c)/3

a = (a+c)/2

$$\underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\text{new values}} = \underbrace{\begin{pmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}}_{M} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = M \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\text{old values}}$$

$c = (a+b+c)/3$



$b = (b+c)/2$

$a = (a+c)/2$

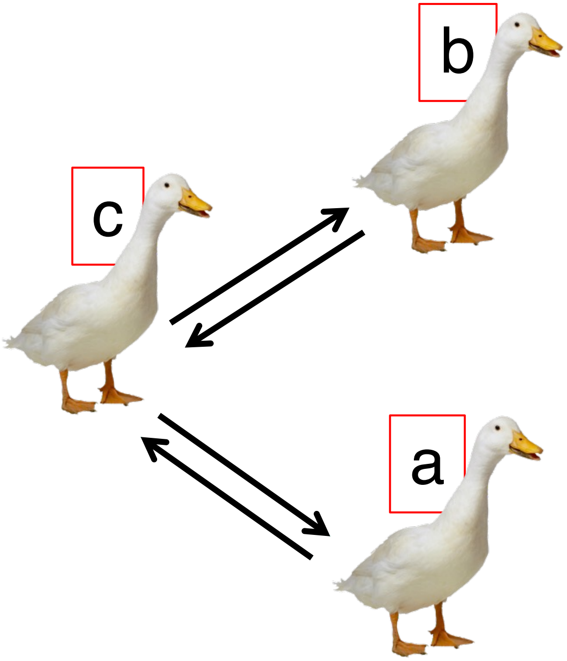
after 2 iterations

after 1 iteration

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

new values

old values



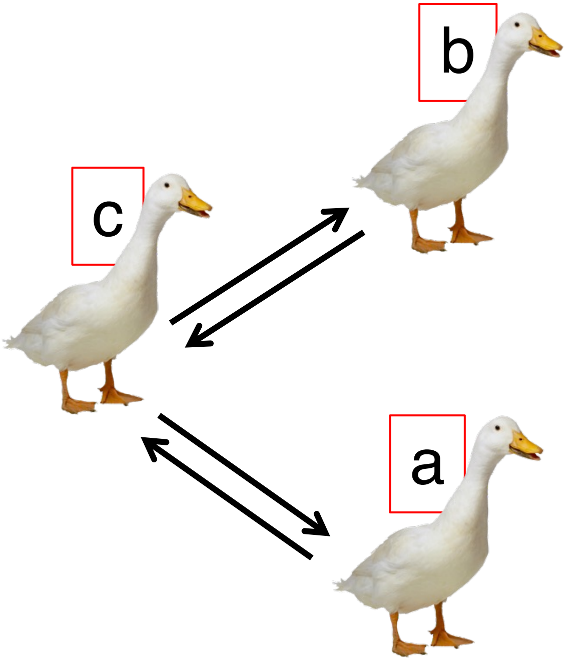
b = (b+c)/2

c = (a+b+c)/3

a = (a+c)/2

after k iterations

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



b = (b+c)/2

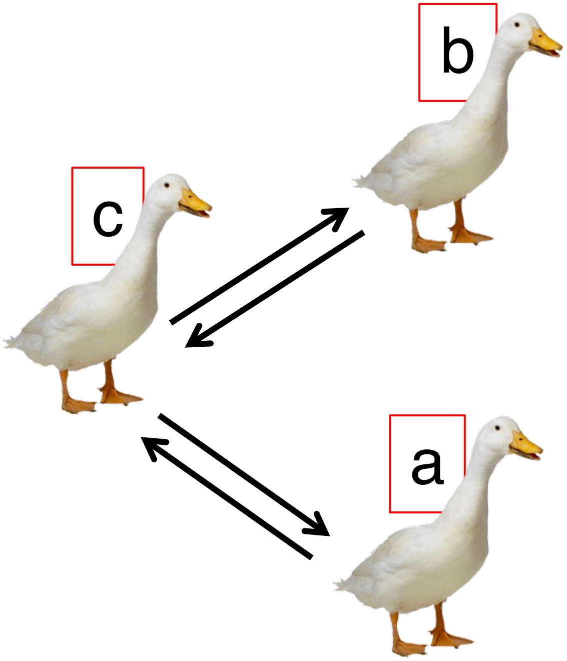
c = (a+b+c)/3

a = (a+c)/2

after k iterations

k → ∞

$$\underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\text{final}} = M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix} \longrightarrow \begin{pmatrix} .285 & .285 & .43 \\ .285 & .285 & .43 \\ .285 & .285 & .43 \end{pmatrix} \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\text{initial}}$$



b = (b+c)/2

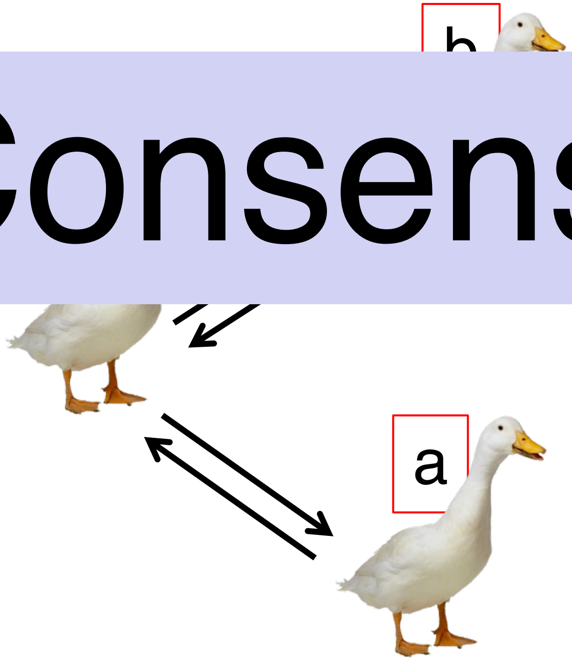
c = (a+b+c)/3

a = (a+c)/2

after k iterations

k → ∞

$$\underset{\text{final}}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}} = M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix} \longrightarrow \begin{pmatrix} .285 & .285 & .43 \\ .285 & .285 & .43 \\ .285 & .285 & .43 \end{pmatrix} \underset{\text{initial}}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}$$



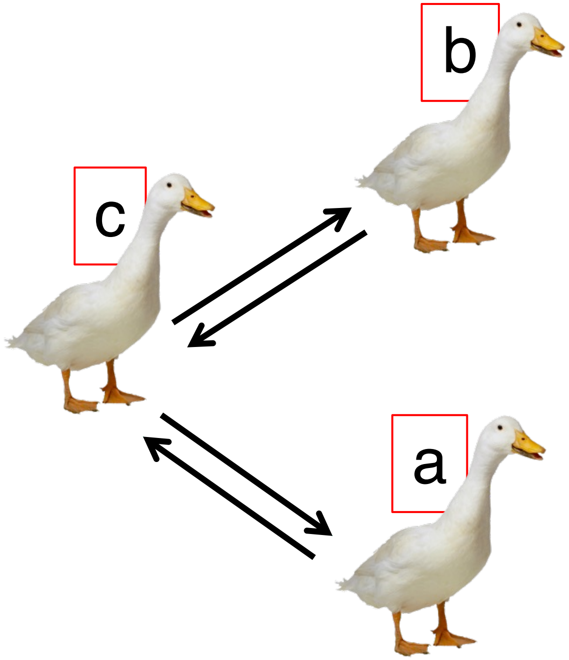
Consensus !

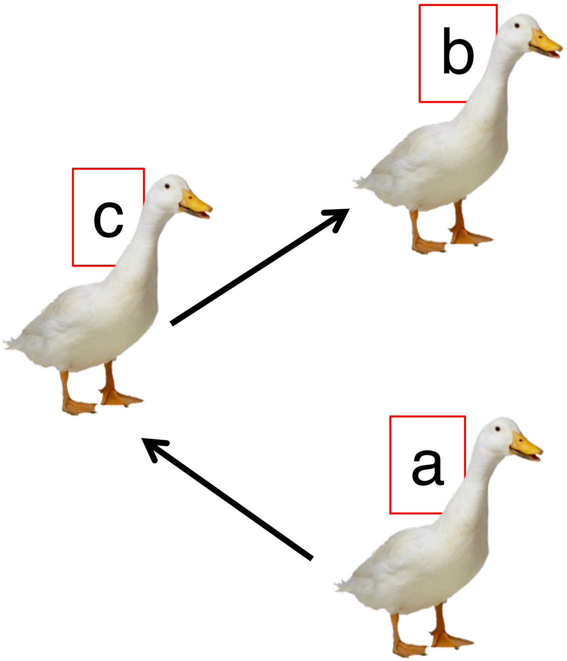
c = (a+b+c)/3

a = (a+c)/2

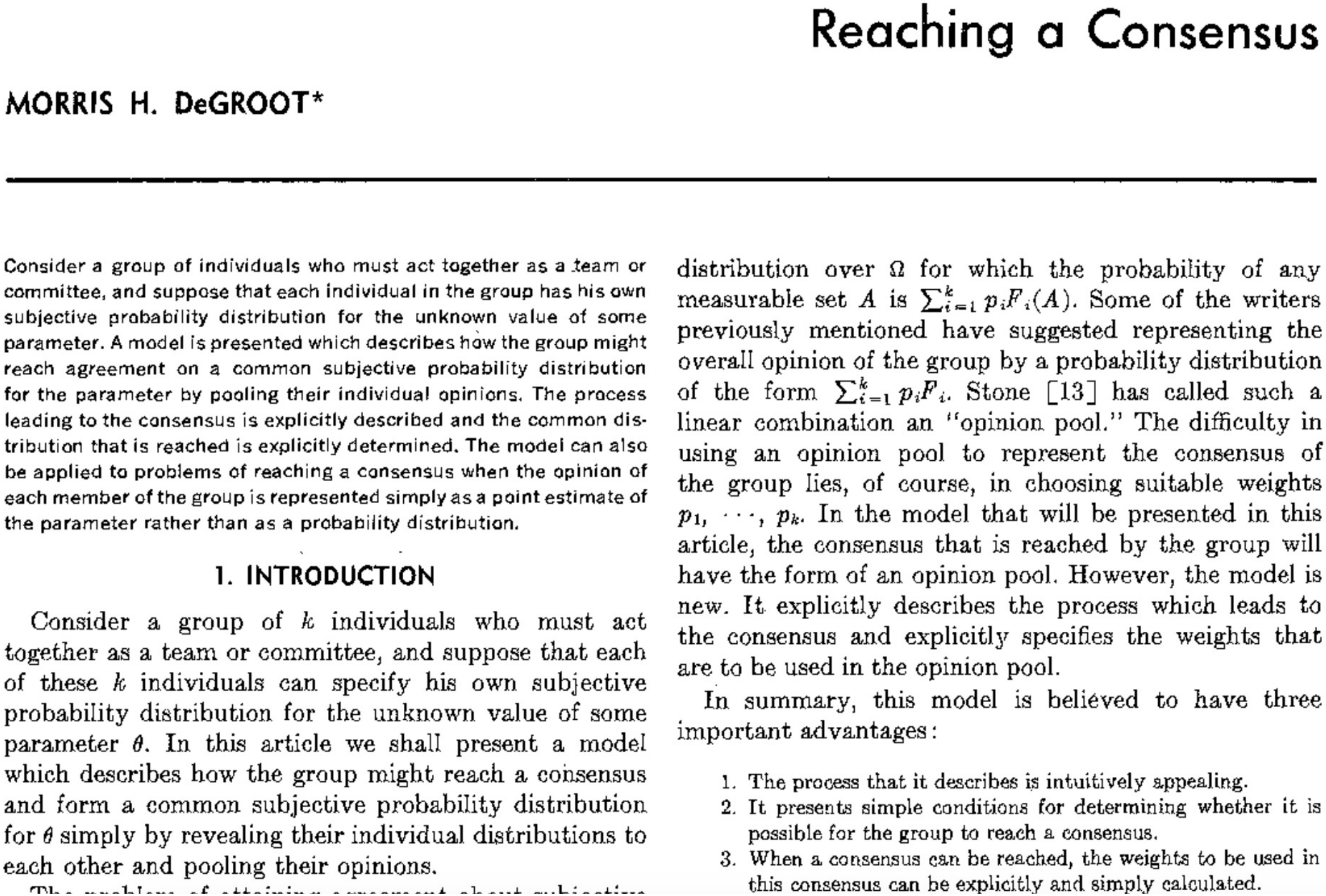
Graph Condition for Consensus

- At least one node must be able to influence all nodes





1974



Reaching a Consensus

MORRIS H. DeGROOT*

Consider a group of individuals who must act together as a team or committee, and suppose that each individual in the group has his own subjective probability distribution for the unknown value of some parameter. A model is presented which describes how the group might reach agreement on a common subjective probability distribution for the parameter by pooling their individual opinions. The process leading to the consensus is explicitly described and the common distribution that is reached is explicitly determined. The model can also be applied to problems of reaching a consensus when the opinion of each member of the group is represented simply as a point estimate of the parameter rather than as a probability distribution.

1. INTRODUCTION

Consider a group of k individuals who must act together as a team or committee, and suppose that each of these k individuals can specify his own subjective probability distribution for the unknown value of some parameter θ. In this article we shall present a model which describes how the group might reach a consensus and form a common subjective probability distribution for θ simply by revealing their individual distributions to each other and pooling their opinions.

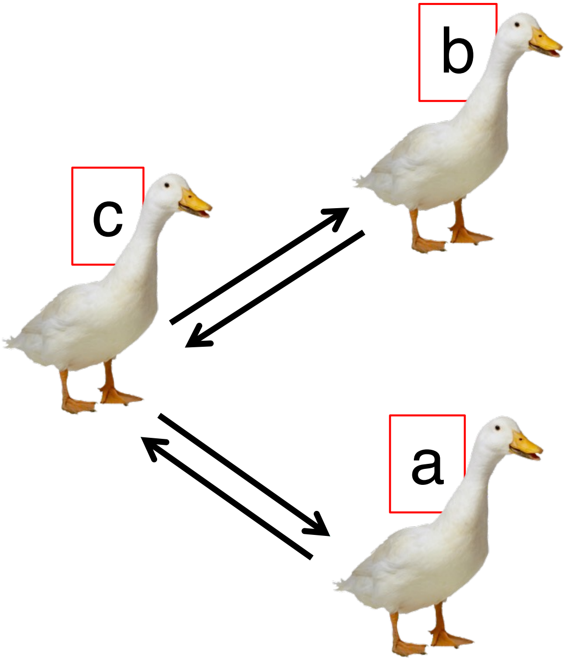
The problem of attaining agreement about subjective

distribution over Ω for which the probability of any measurable set A is $\sum_{i=1}^{k} p_i F_i(A)$. Some of the writers previously mentioned have suggested representing the overall opinion of the group by a probability distribution of the form $\sum_{i=1}^{k} p_i F_i$. Stone [13] has called such a linear combination an "opinion pool." The difficulty in using an opinion pool to represent the consensus of the group lies, of course, in choosing suitable weights $p_1, \cdots, p_k$. In the model that will be presented in this article, the consensus that is reached by the group will have the form of an opinion pool. However, the model is new. It explicitly describes the process which leads to the consensus and explicitly specifies the weights that are to be used in the opinion pool.

In summary, this model is believed to have three important advantages:

1. The process that it describes is intuitively appealing.
2. It presents simple conditions for determining whether it is possible for the group to reach a consensus.
3. When a consensus can be reached, the weights to be used in this consensus can be explicitly and simply calculated.

Change of Weights

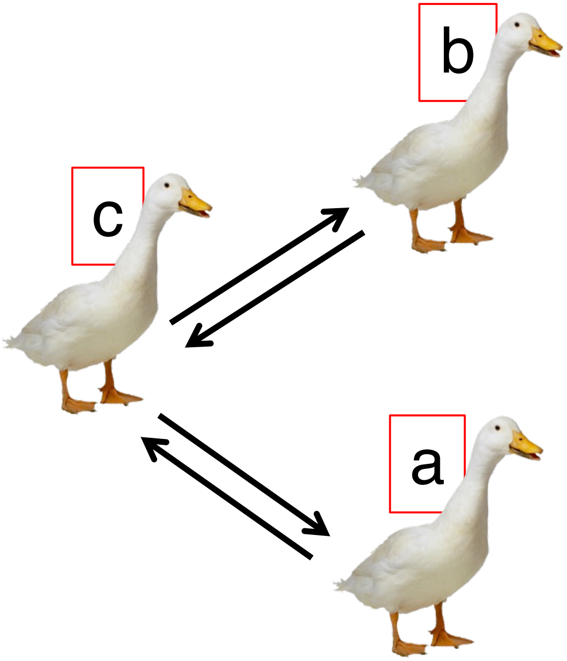


b = 3b/4+ c/4

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

a = 3a/4+ c/4

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



b = 3b/4+ c/4

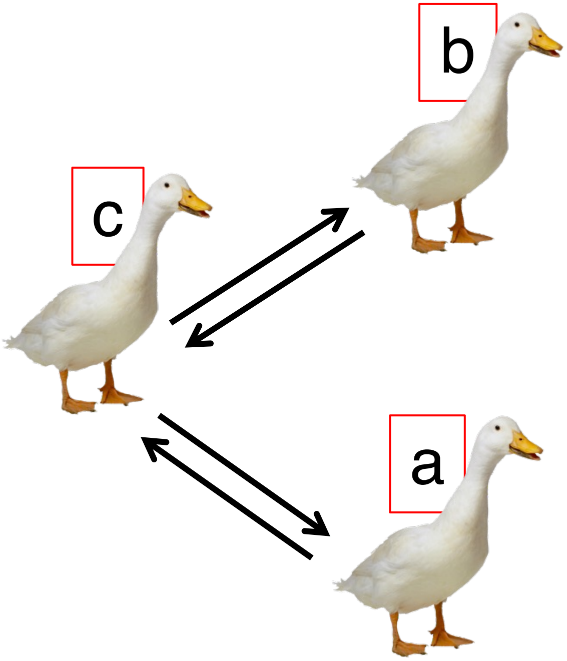
c = a/4+b/4+c/2

a = 3a/4+ c/4

after k iterations

k → ∞

$$\underset{\text{final}}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}} = M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix} \rightarrow \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix} \underset{\text{initial}}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}$$



b = 3b/4+ c/4

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

a = 3a/4+ c/4

after k iterations

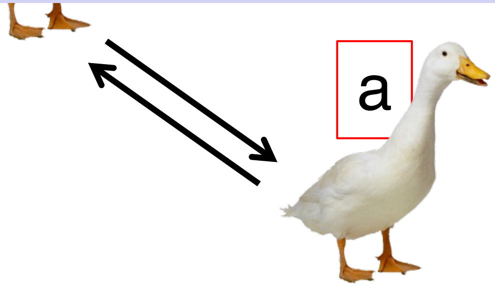
k → ∞

$$\underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\text{final}} = M^k \begin{pmatrix} a \\ b \\ c \end{pmatrix} \rightarrow \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{pmatrix} \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\text{initial}}$$

Average Consensus

3b/4+ c/4

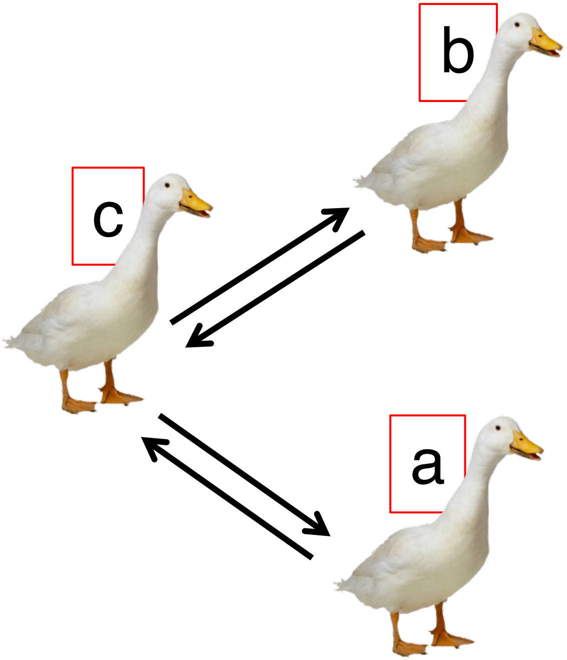
c = a/4+b/4+c/2

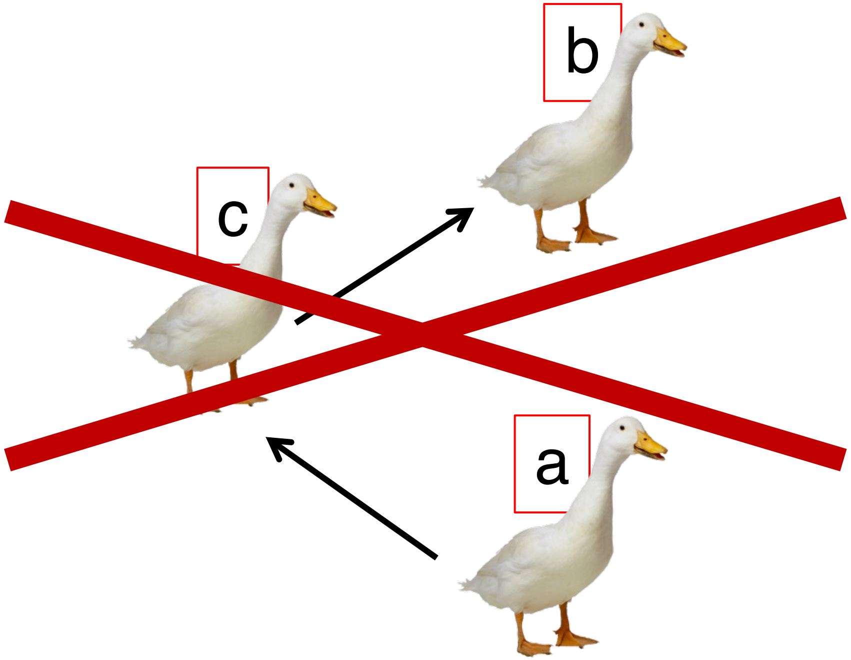


a = 3a/4+ c/4

Graph Condition for Average Consensus

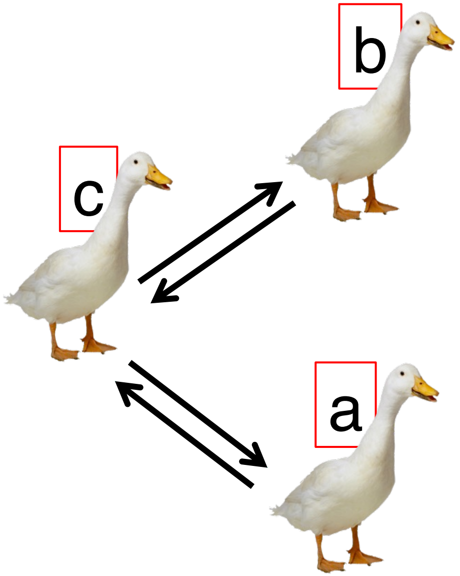
- Every node must be able to influence all others
  - Strong connectivity





Lossy Wireless Links

(2012)



Average consensus over lossy wireless links?

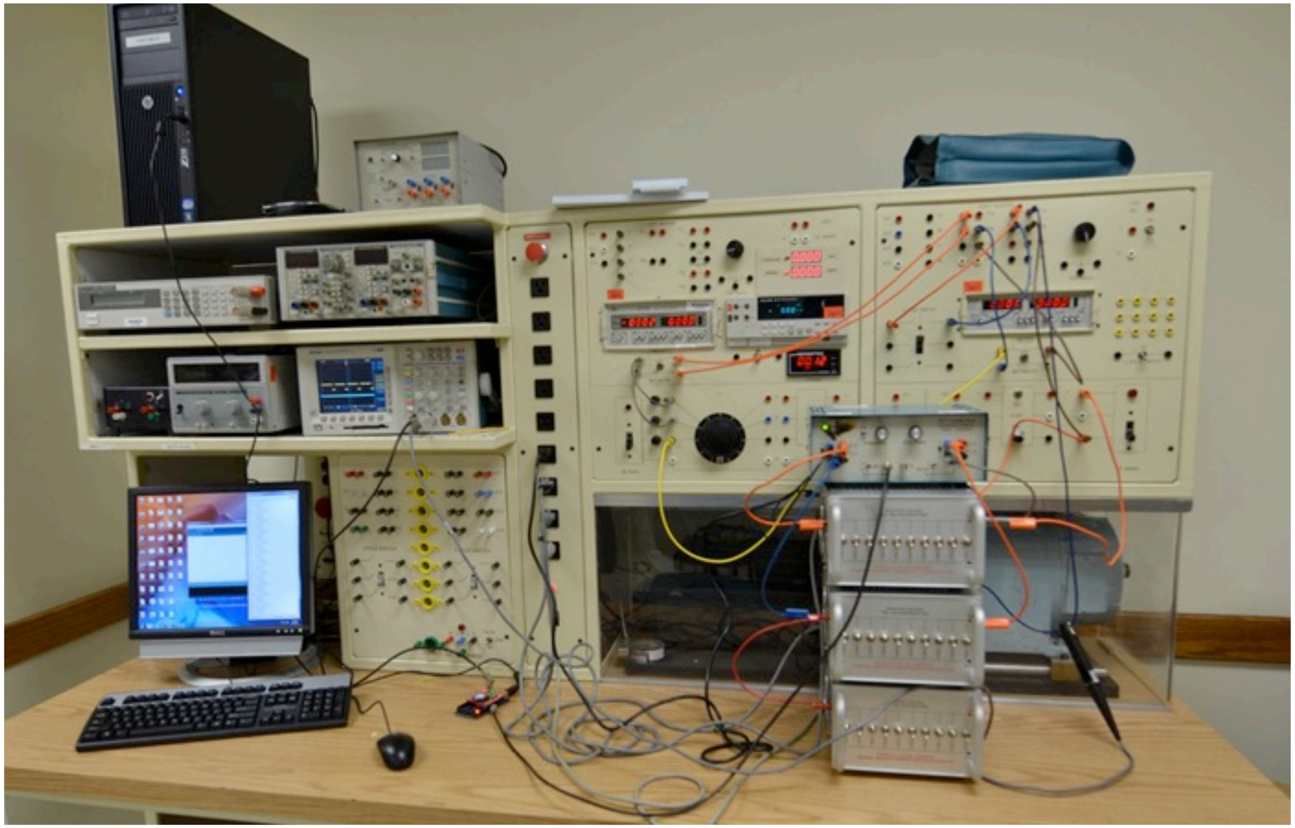
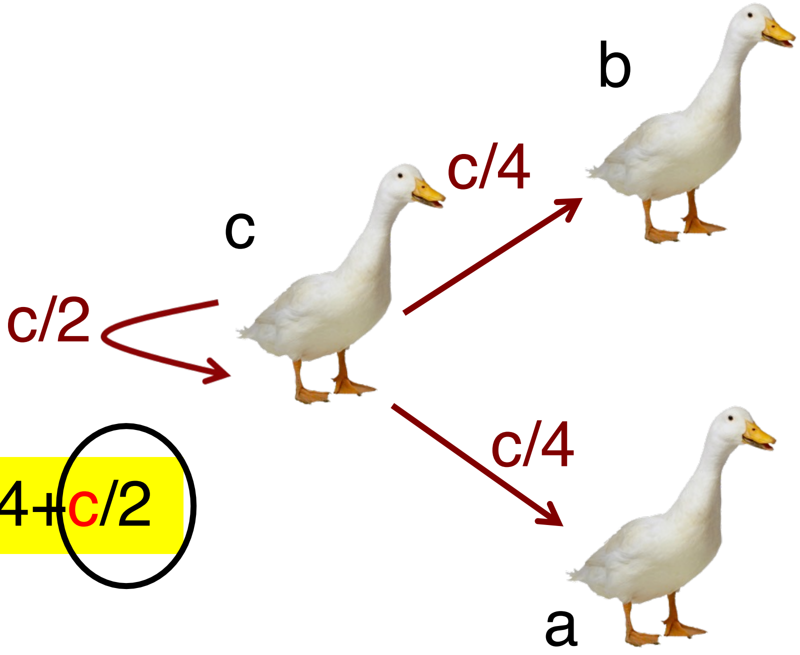


Photo courtesy Alejandro Dominguez-Garcia

Implementation

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



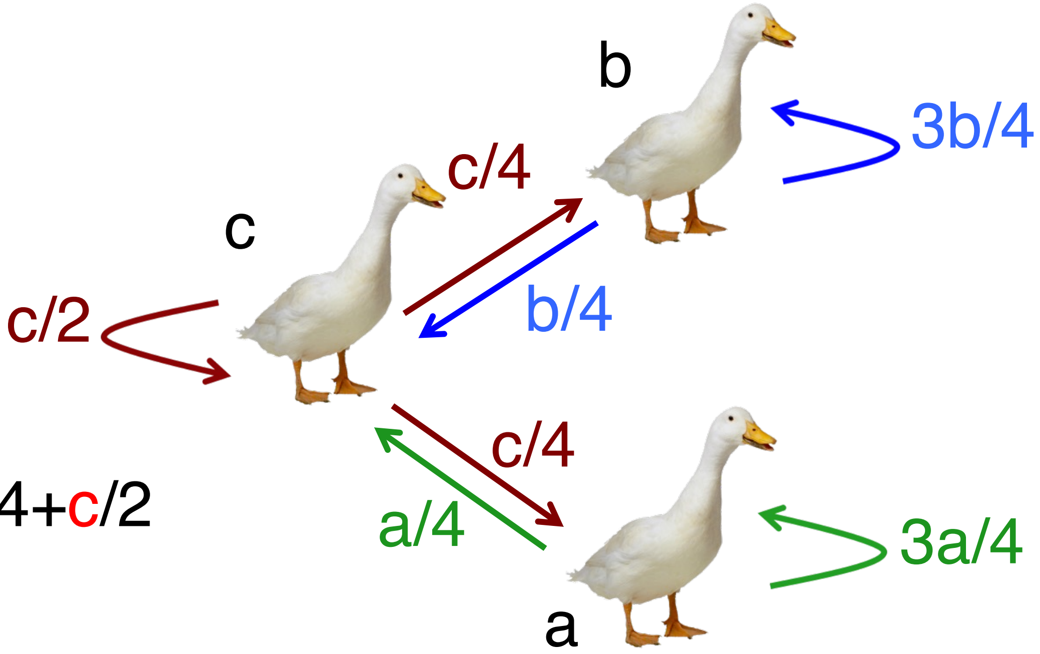
$b = 3b/4 + c/4$

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

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

Implementation

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



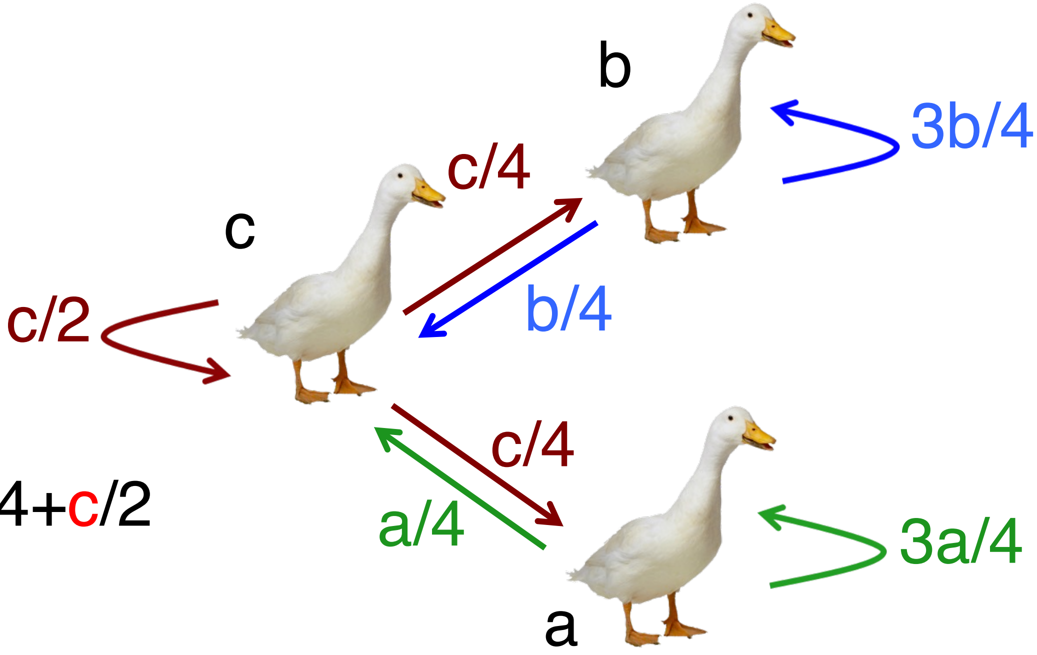
b = 3b/4+ c/4

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

a = 3a/4+ c/4

Conservation of Mass

- a+b+c constant after each iteration



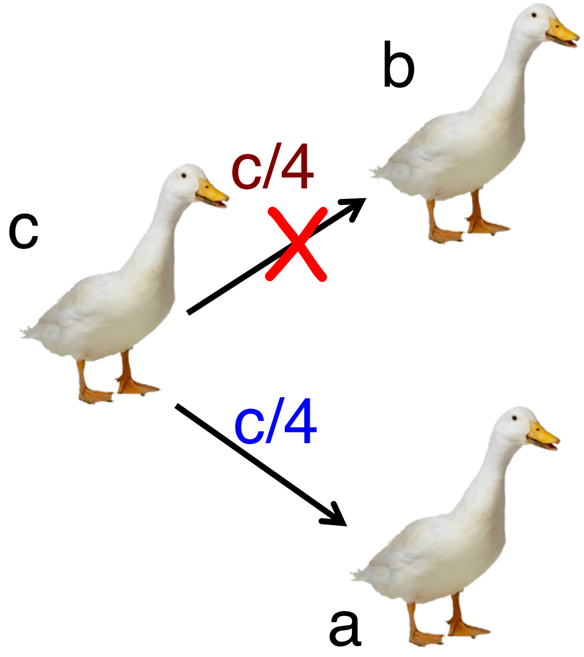
$b = 3b/4 + c/4$

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

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

Wireless Transmissions Lossy

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



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

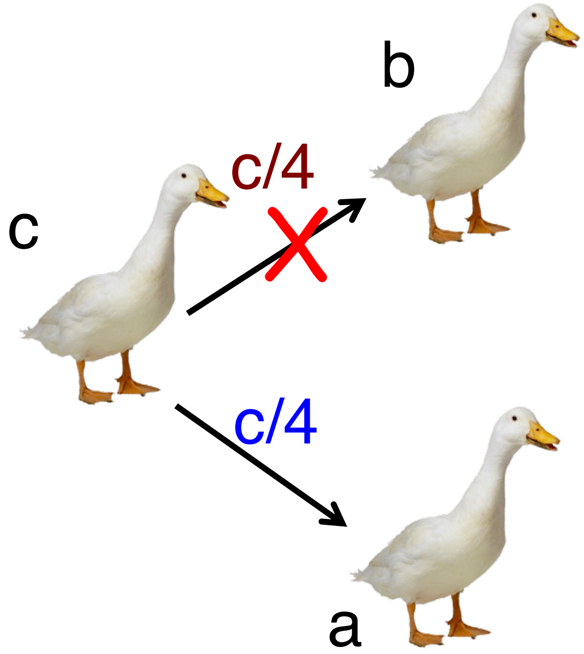
$a = 3a/4+ c/4$

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

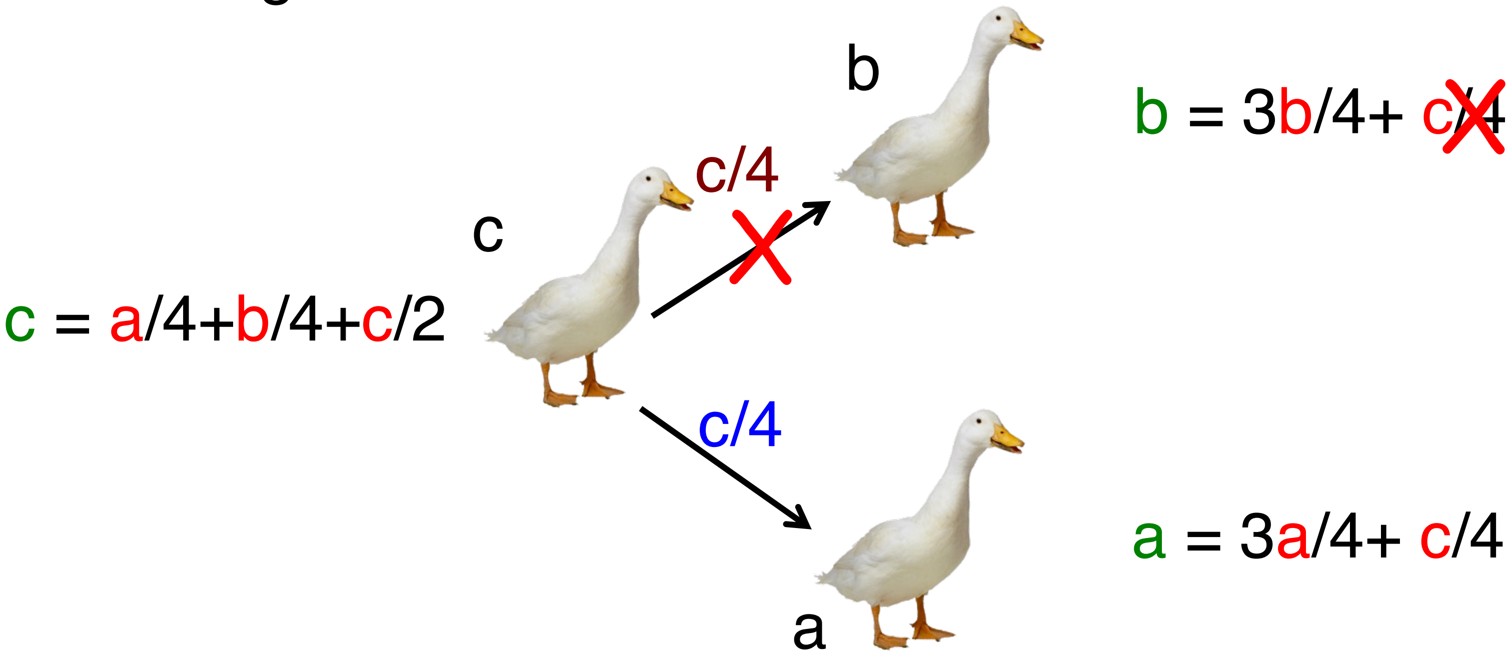


a = 3a/4+ c/4

Average Consensus over Lossy Links

- Sender and receiver views potentially inconsistent
  … message delivered or not?

- Average consensus fails



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

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

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

Average Consensus over Lossy Links (2012)

- Solution …

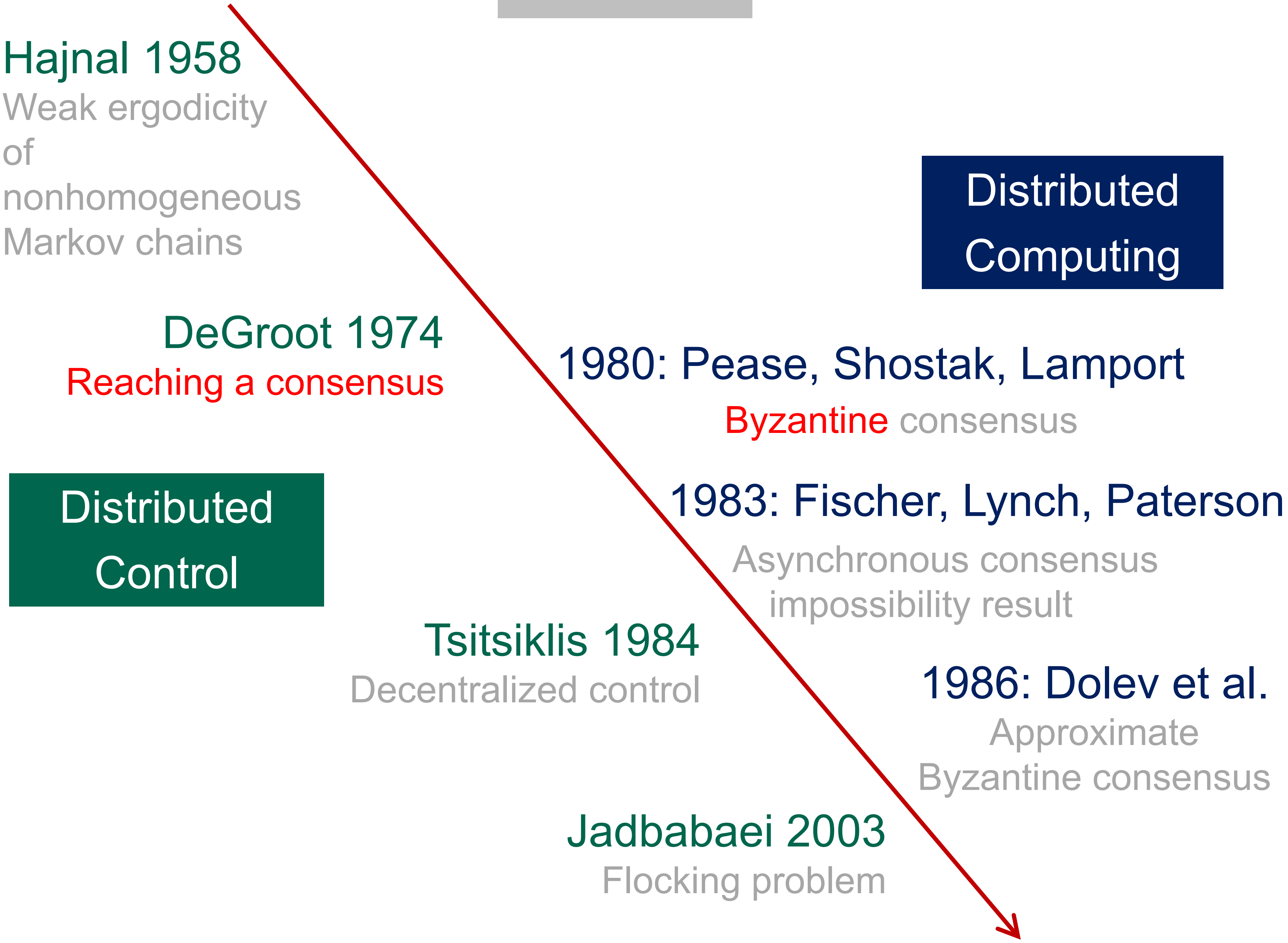
  a different algorithm that can
  tolerate lossy links
  without explicit knowledge of lost messages

Long-term benefit for me …

- Exposure to a new class of problems
- New mathematical tools for analyzing algorithms

➔ Impacted a large fraction of my work since

Consensus



Hajnal 1958
Weak ergodicity of nonhomogeneous Markov chains

Distributed Computing

DeGroot 1974
Reaching a consensus

1980: Pease, Shostak, Lamport
Byzantine consensus

Distributed Control

1983: Fischer, Lynch, Paterson
Asynchronous consensus impossibility result

Tsitsiklis 1984
Decentralized control

1986: Dolev et al.
Approximate Byzantine consensus

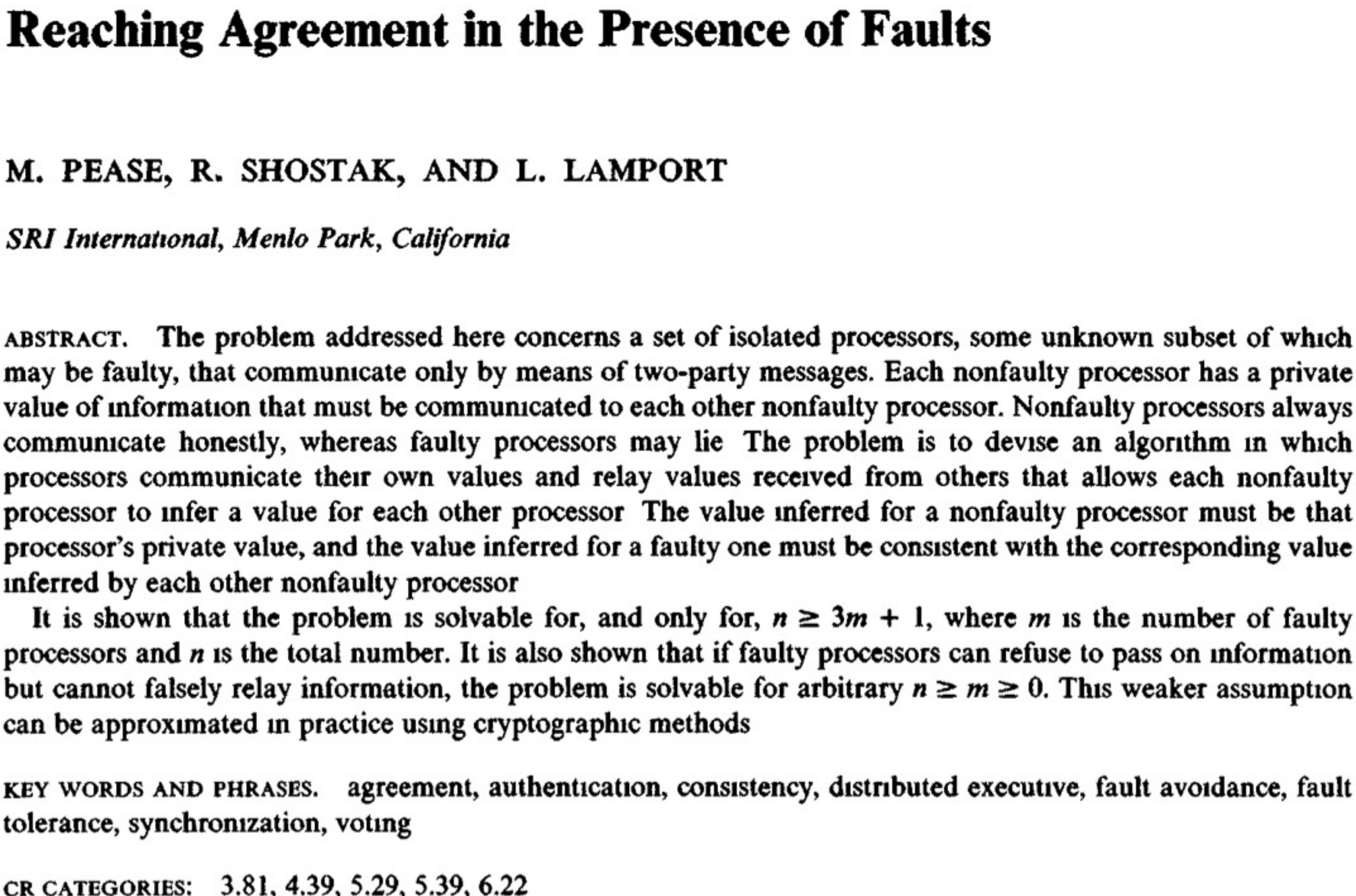
Jadbabaei 2003
Flocking problem

Consensus

Distributed Computing

- Faults
- Scalar inputs
- Undirected graphs, often complete
- *Global* algorithms
- Exact consensus in synchronous systems

1980



Reaching Agreement in the Presence of Faults

M. PEASE, R. SHOSTAK, AND L. LAMPORT

SRI International, Menlo Park, California

ABSTRACT. The problem addressed here concerns a set of isolated processors, some unknown subset of which may be faulty, that communicate only by means of two-party messages. Each nonfaulty processor has a private value of information that must be communicated to each other nonfaulty processor. Nonfaulty processors always communicate honestly, whereas faulty processors may lie The problem is to devise an algorithm in which processors communicate their own values and relay values received from others that allows each nonfaulty processor to infer a value for each other processor The value inferred for a nonfaulty processor must be that processor's private value, and the value inferred for a faulty one must be consistent with the corresponding value inferred by each other nonfaulty processor

It is shown that the problem is solvable for, and only for, $n \geq 3m + 1$, where m is the number of faulty processors and n is the total number. It is also shown that if faulty processors can refuse to pass on information but cannot falsely relay information, the problem is solvable for arbitrary $n \geq m \geq 0$. This weaker assumption can be approximated in practice using cryptographic methods

KEY WORDS AND PHRASES. agreement, authentication, consistency, distributed executive, fault avoidance, fault tolerance, synchronization, voting

CR CATEGORIES: 3.81, 4.39, 5.29, 5.39, 6.22

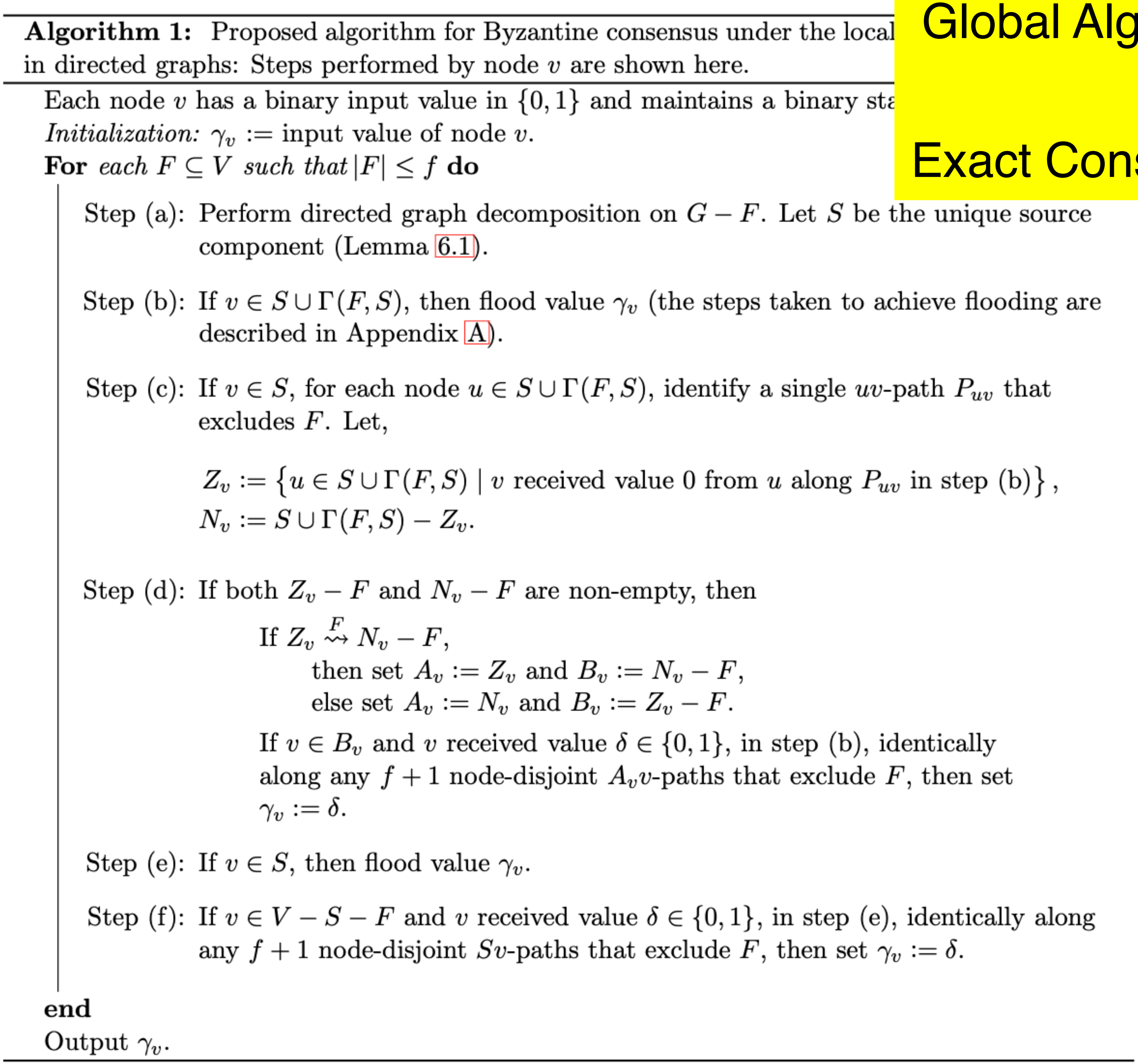
Consensus

Distributed Computing

- Faults
- Scalar inputs
- Undirected graphs, often complete
- *Global* algorithms
- Exact consensus in synchronous systems

Global Algorithm

Exact Consensus



Algorithm 1: Proposed algorithm for Byzantine consensus under the local
in directed graphs: Steps performed by node v are shown here.

```
Each node $v$ has a binary input value in $\{0,1\}$ and maintains a binary sta
Initialization: $\gamma_v :=$ input value of node $v$.
For each $F \subseteq V$ such that $|F| \leq f$ do
    Step (a): Perform directed graph decomposition on $G - F$. Let $S$ be the unique source
              component (Lemma 6.1).
    Step (b): If $v \in S \cup \Gamma(F,S)$, then flood value $\gamma_v$ (the steps taken to achieve flooding are
              described in Appendix A).
    Step (c): If $v \in S$, for each node $u \in S \cup \Gamma(F,S)$, identify a single $uv$-path $P_{uv}$ that
              excludes $F$. Let,
              $Z_v := \{u \in S \cup \Gamma(F,S) \mid v$ received value 0 from $u$ along $P_{uv}$ in step (b)$\}$,
              $N_v := S \cup \Gamma(F,S) - Z_v$.
    Step (d): If both $Z_v - F$ and $N_v - F$ are non-empty, then
                  If $Z_v \overset{F}{\rightsquigarrow} N_v - F$,
                      then set $A_v := Z_v$ and $B_v := N_v - F$,
                      else set $A_v := N_v$ and $B_v := Z_v - F$.
                  If $v \in B_v$ and $v$ received value $\delta \in \{0,1\}$, in step (b), identically
                  along any $f+1$ node-disjoint $A_v v$-paths that exclude $F$, then set
                  $\gamma_v := \delta$.
    Step (e): If $v \in S$, then flood value $\gamma_v$.
    Step (f): If $v \in V - S - F$ and $v$ received value $\delta \in \{0,1\}$, in step (e), identically along
              any $f+1$ node-disjoint $Sv$-paths that exclude $F$, then set $\gamma_v := \delta$.
end
Output $\gamma_v$.
```

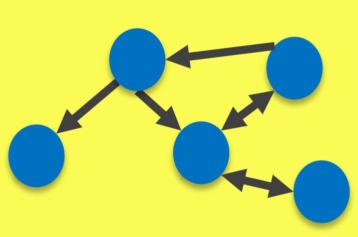
Consensus

Distributed Computing

- Faults
- Scalar inputs
- Undirected graphs, often complete
- *Global* algorithms
- Exact consensus in synchronous systems

Distributed Control

- No faults
- Vector inputs
- Incomplete (directed) graphs



- *Local* algorithms
- Approximate consensus

Many problems should have been solved decades ago … but were not

- Borrowing assumptions from the other domain

- New network models

Many problems should have been solved decades ago … but were not

- Local algorithms
  - Average consensus over lossy links
  - Byzantine consensus over point-to-point channels
  - Byzantine consensus over broadcast channels

- Global algorithm
  - Byzantine consensus over directed graphs
  - Byzantine consensus over broadcast channels

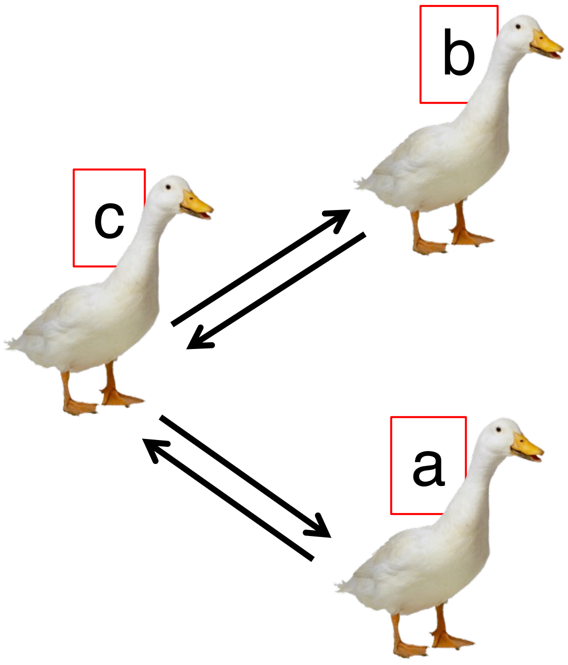
and more …

Fault-Tolerant Consensus

with Local Algorithms

Local Averaging

Initially, state = input



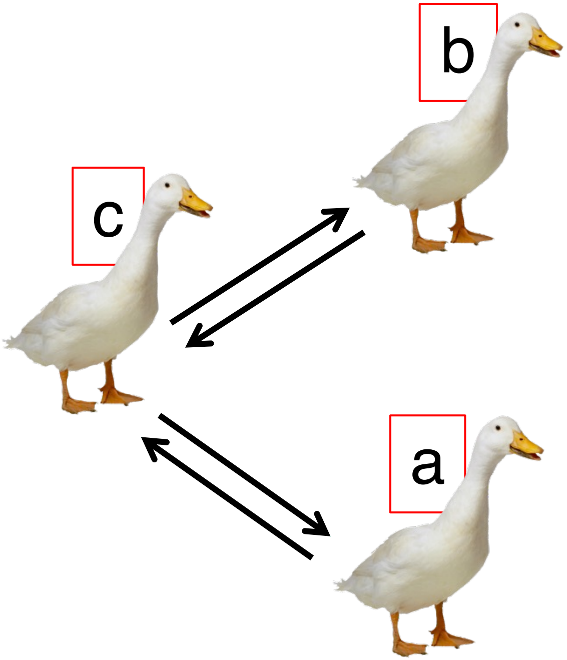
$b = (b+c)/2$

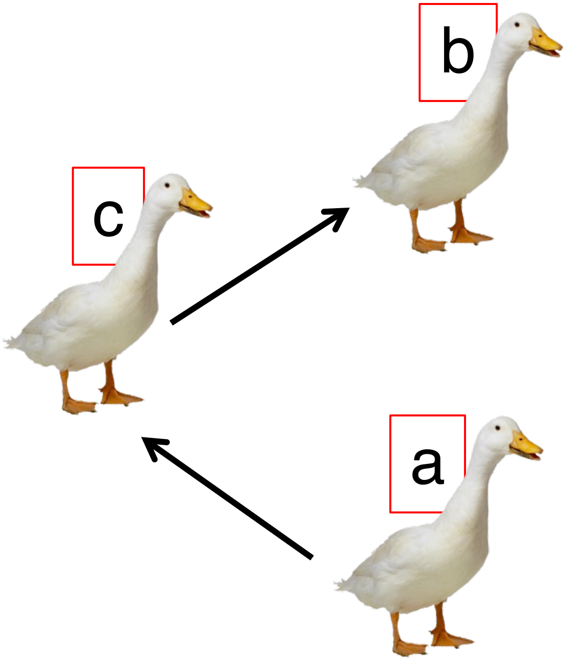
$c = (a+b+c)/3$

$a = (a+c)/2$

Graph Condition for Consensus

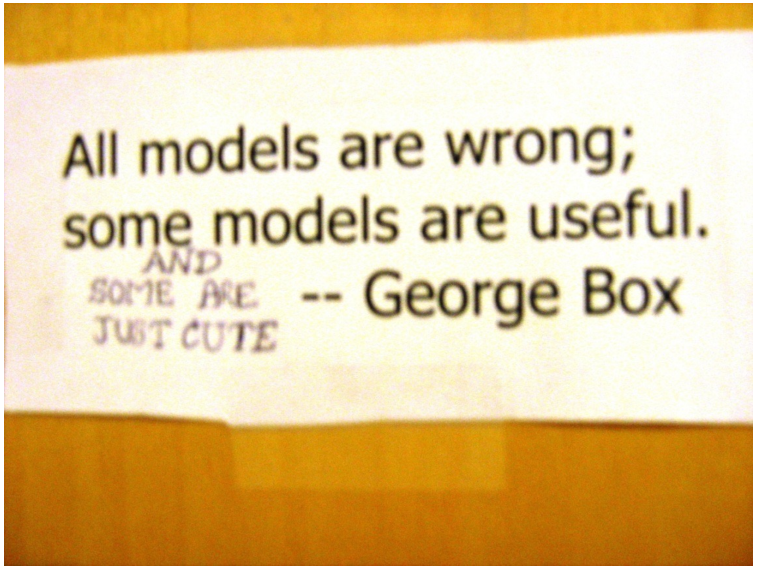
- At least one node must be able to influence all nodes





Byzantine Fault Model

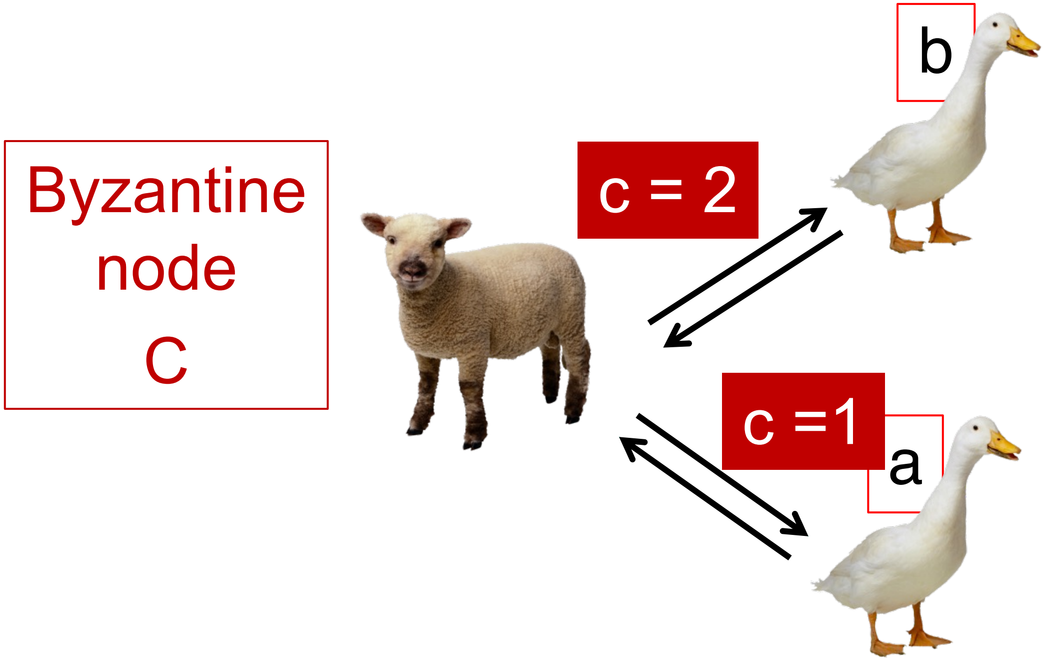
- No constraint on misbehavior of faulty agents



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

Local Averaging

Initially, state = input



Byzantine node C

c = 2

c =1

b

a

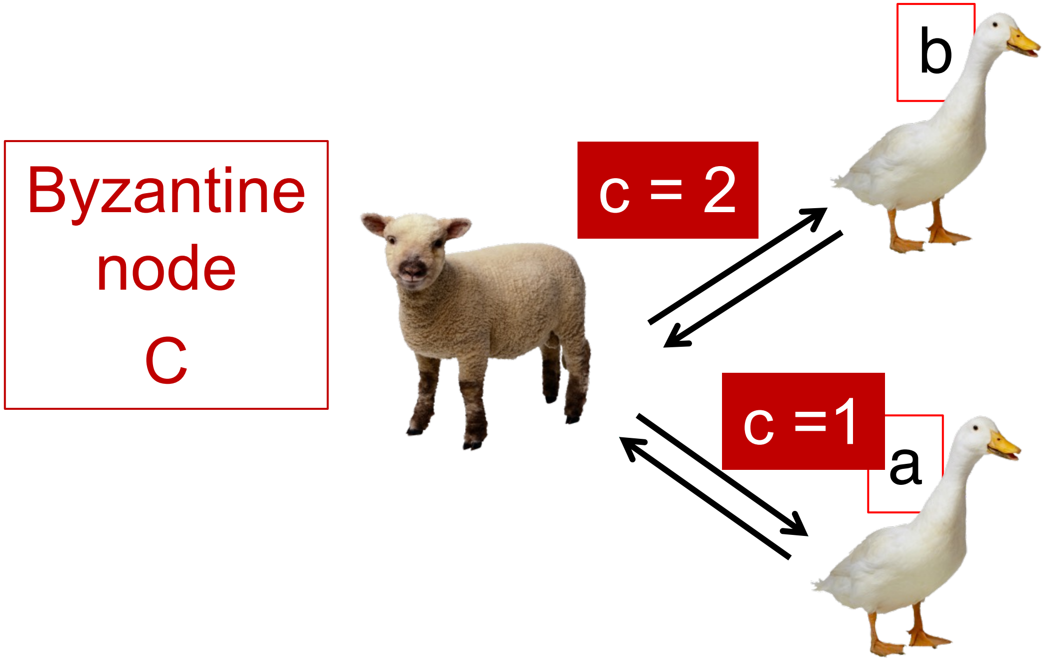
$b = (b+c)/2$

$a = (a+c)/2$

Local Averaging

Initially, state = input

No consensus !



Byzantine node C

c = 2

c =1

b

a

$b = (b+c)/2$

$a = (a+c)/2$

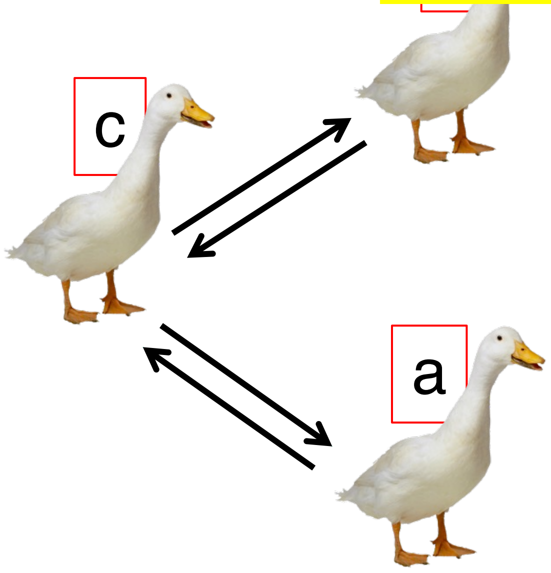
Graph Condition for Consensus with Byzantine faults

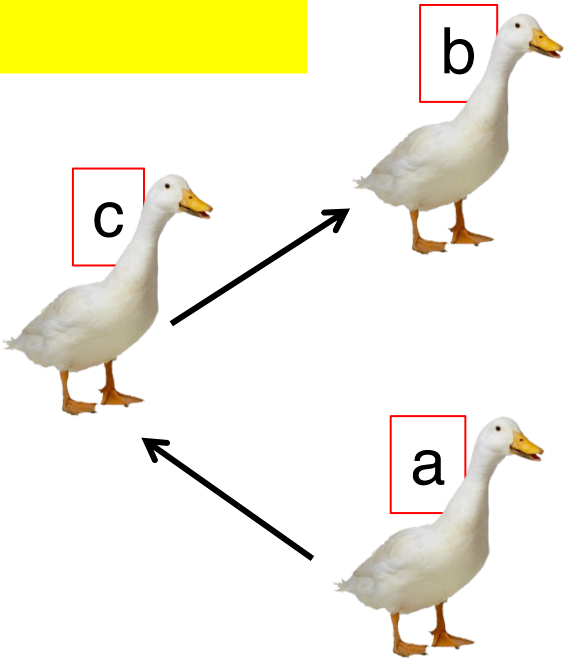
- Goal is to achieve consensus among the non-faulty agents

Graph Condition for Consensus
with Byzantine faults

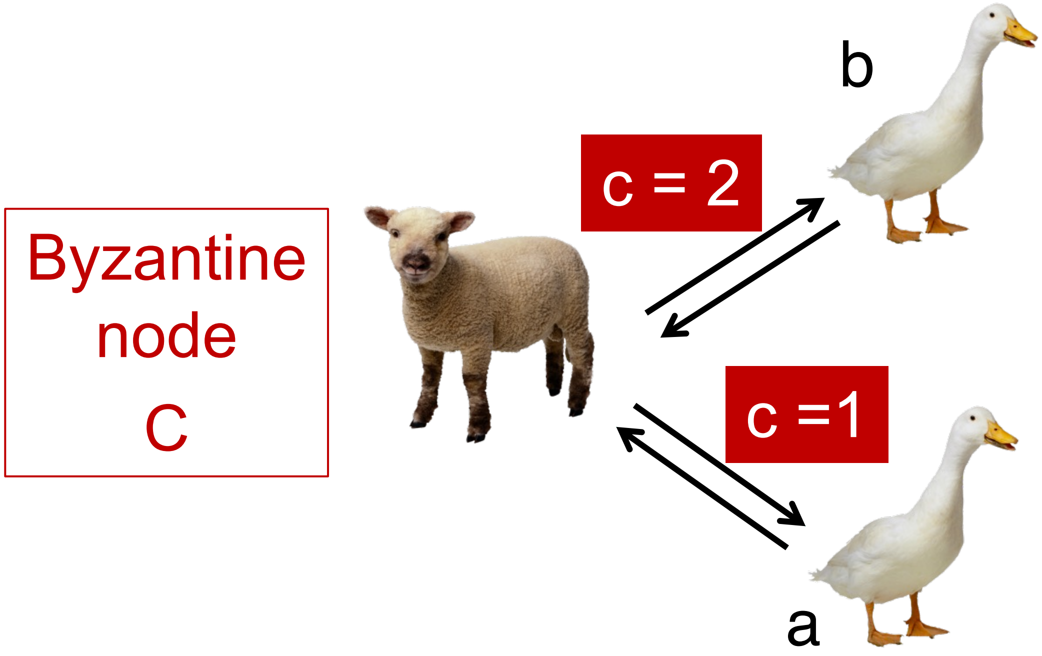
- At least one node must be able to influence all nodes

Insufficient with faults





Graph condition for local Byzantine consensus algorithms is now known (2012)



Byzantine node C

c = 2

c =1

b

a

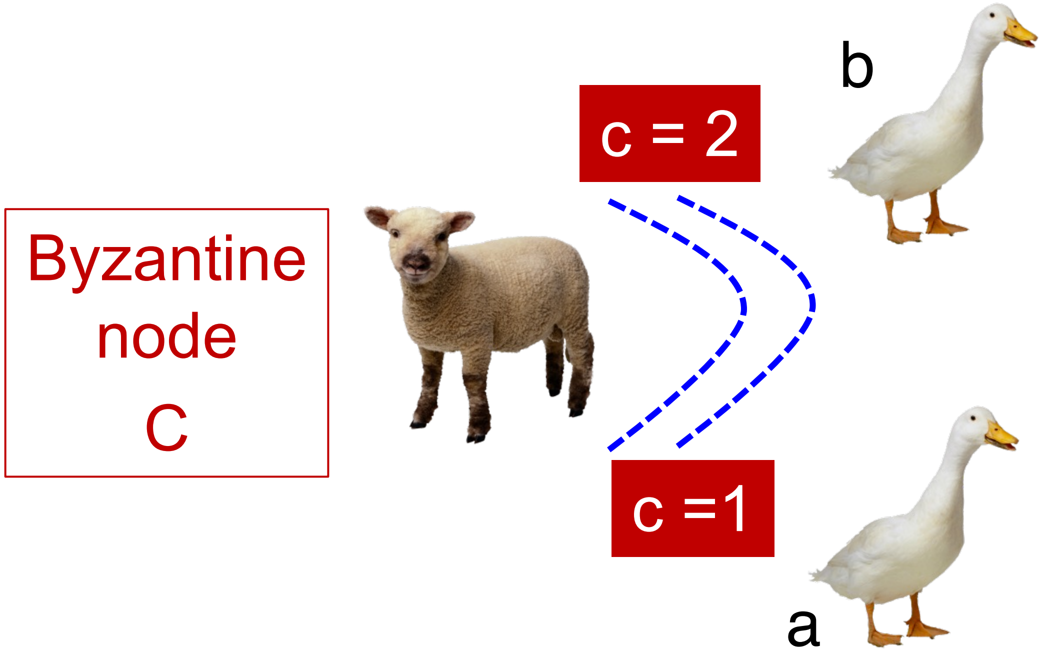
Fault-Tolerant Consensus

over Broadcast Channels

Wireless Broadcast

- Wireless transmissions (potentially) received by all neighbors of the sender
- Does this benefit Byzantine consensus?

Wireless Broadcast



Byzantine node C

c = 2

c =1

b

a

Misbehavior can be detected

Graph Condition for Byzantine Consensus
Global Algorithms

- Point-to-point networks (1982)

  2f + 1 connectivity

  3f + 1 nodes

f faulty

nodes

Graph Condition for Byzantine Consensus
Global Algorithms

- Point-to-point networks (1982)

  2f + 1 connectivity

  3f + 1 nodes

f faulty nodes

- Broadcast channels (2018)

  3f/2 + 1 connectivity

  2f node degree

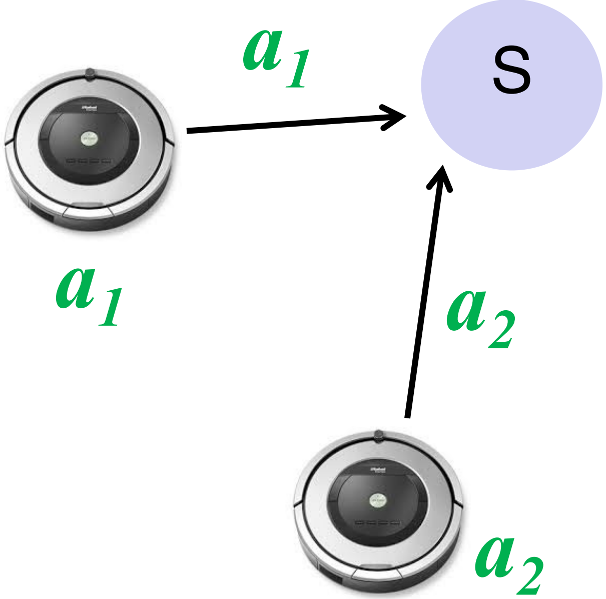
Fault-Tolerant

Distributed Optimization

Averaging

- Input of node i = a_i

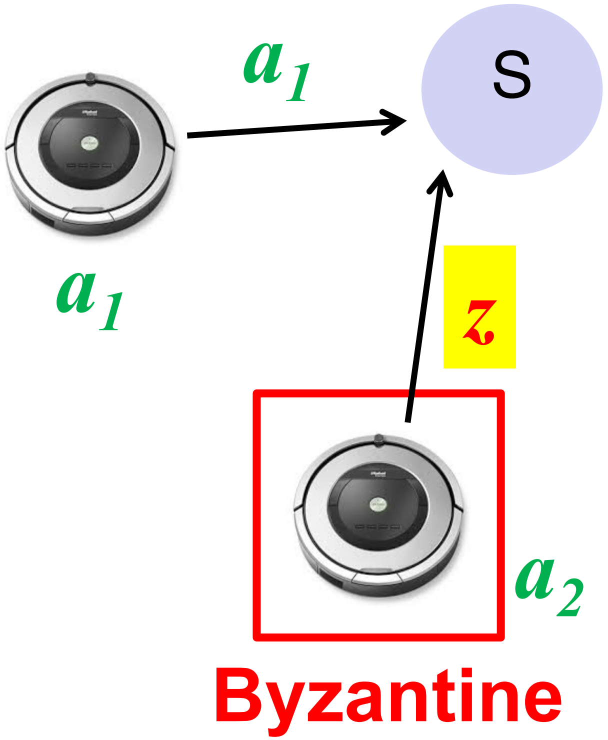
- Compute average of $a_i's$



S is a trusted server

Fault-Tolerant Averaging

- What to do if some nodes send bogus values?



Averaging ➔ Optimization

- Input of node i = a_i

- Average of $a_i's$ = $$argmin \sum_i f_i(x)$$

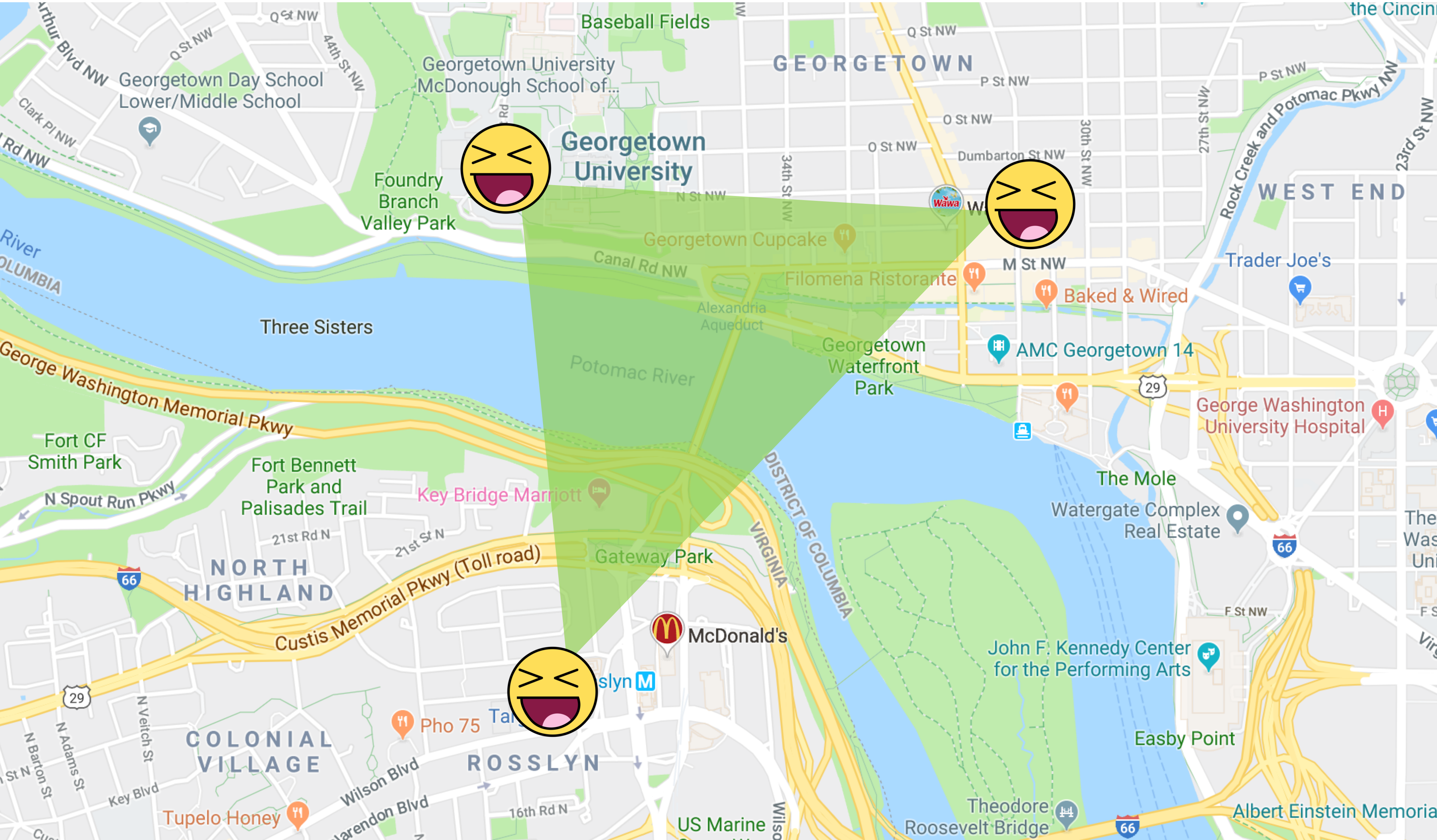
where

$$f_i(x) = (x - a_i)^2$$

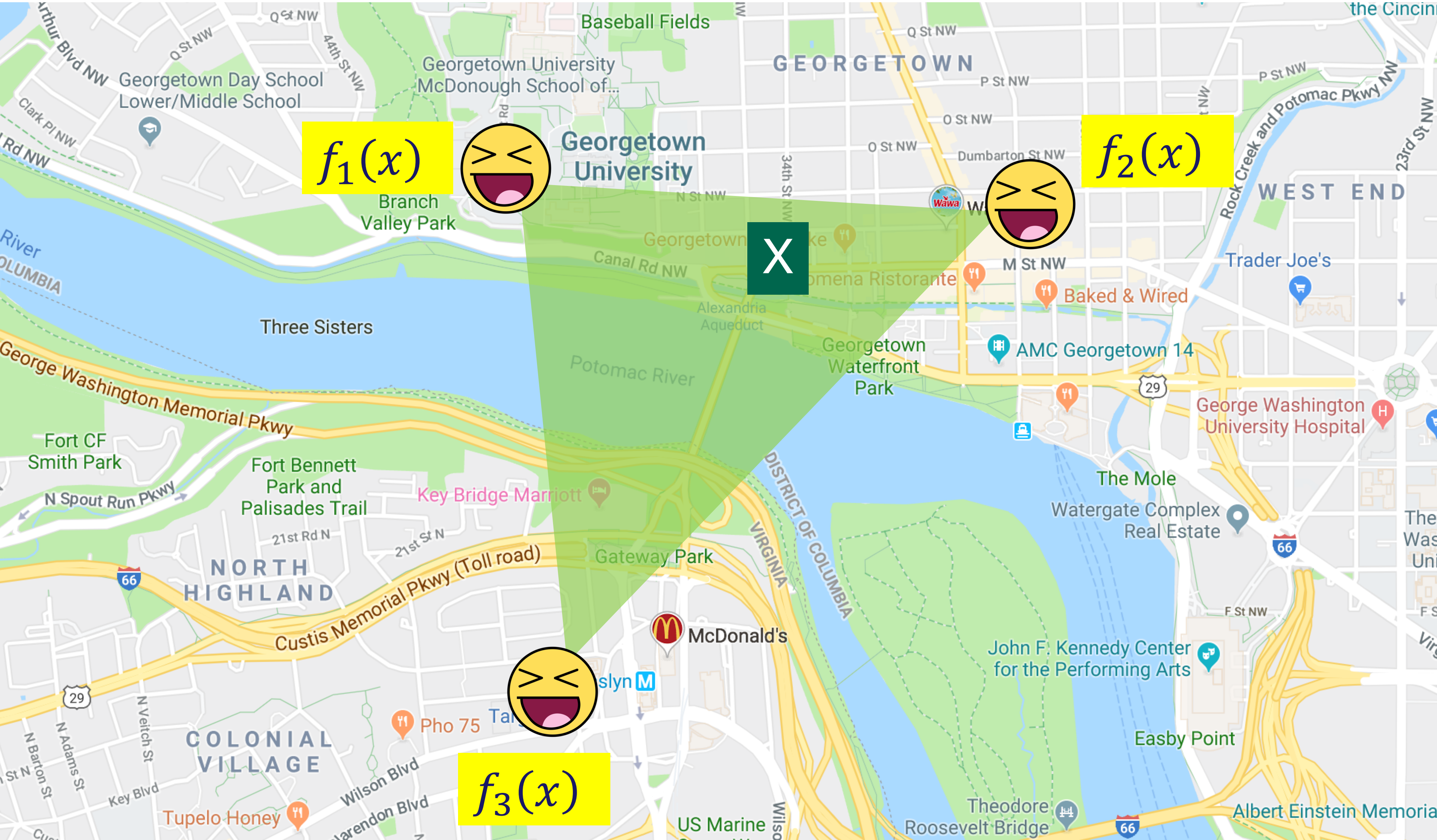
$$\operatorname{argmin} \sum f_i(x)$$

Many Applications

Rendezvous


Baseball Fields
GEORGETOWN
Georgetown University McDonough School of...
Georgetown Day School Lower/Middle School
Georgetown University
Foundry Branch Valley Park
Georgetown Cupcake
Canal Rd NW
Filomena Ristorante
Baked & Wired
M St NW
Trader Joe's
WEST END
Alexandria Aqueduct
Three Sisters
Potomac River
Georgetown Waterfront Park
AMC Georgetown 14
George Washington Memorial Pkwy
George Washington University Hospital
Fort CF Smith Park
Fort Bennett Park and Palisades Trail
Key Bridge Marriott
The Mole
Watergate Complex Real Estate
N Spout Run Pkwy
NORTH HIGHLAND
Gateway Park
Custis Memorial Pkwy (Toll road)
DISTRICT OF COLUMBIA
VIRGINIA
McDonald's
John F. Kennedy Center for the Performing Arts
Pho 75
COLONIAL VILLAGE
ROSSLYN
Wilson Blvd
Clarendon Blvd
Tupelo Honey
Easby Point
Theodore Roosevelt Bridge
Albert Einstein Memoria
US Marine
Rock Creek and Potomac Pkwy NW
Q St NW
P St NW
O St NW
N St NW
Dumbarton St NW
34th St NW
30th St NW
27th St NW
23rd St NW
44th St NW
21st Rd N
21st St N
16th Rd N
F St NW
Key Blvd
N Veitch St
N Adams St
N Barton St
29
66

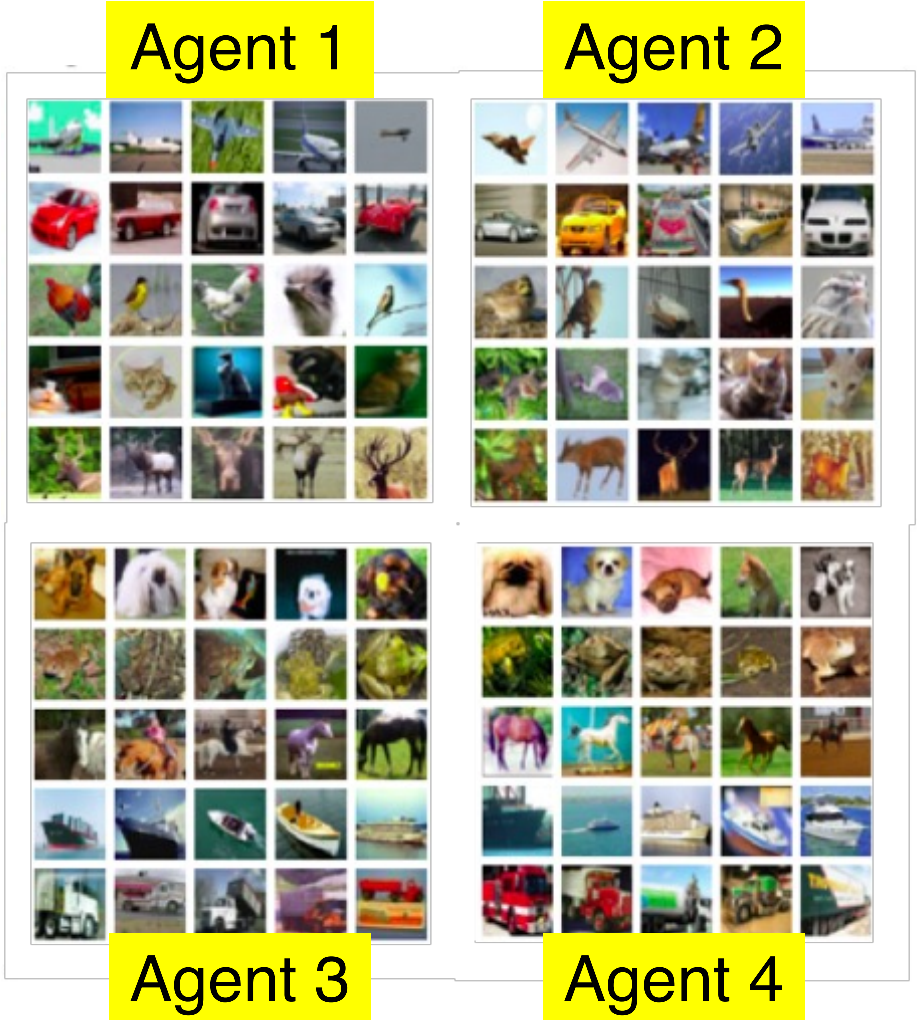
Rendezvous


$f_1(x)$
$f_2(x)$
$f_3(x)$
X
Georgetown University
GEORGETOWN
WEST END
Potomac River
Three Sisters
George Washington Memorial Pkwy
NORTH HIGHLAND
COLONIAL VILLAGE
Custis Memorial Pkwy (Toll road)
Key Bridge Marriott
Gateway Park
McDonald's
Georgetown Waterfront Park
AMC Georgetown 14
Trader Joe's
Baked & Wired
M St NW
George Washington University Hospital
The Mole
Watergate Complex Real Estate
John F. Kennedy Center for the Performing Arts
Theodore Roosevelt Bridge
Albert Einstein Memoria
Branch Valley Park
Baseball Fields
Georgetown University McDonough School of...
Georgetown Day School Lower/Middle School
Fort CF Smith Park
Fort Bennett Park and Palisades Trail
Canal Rd NW
Alexandria Aqueduct
DISTRICT OF COLUMBIA
VIRGINIA
Pho 75
Tupelo Honey
Easby Point
US Marine
Wilson Blvd
Clarendon Blvd

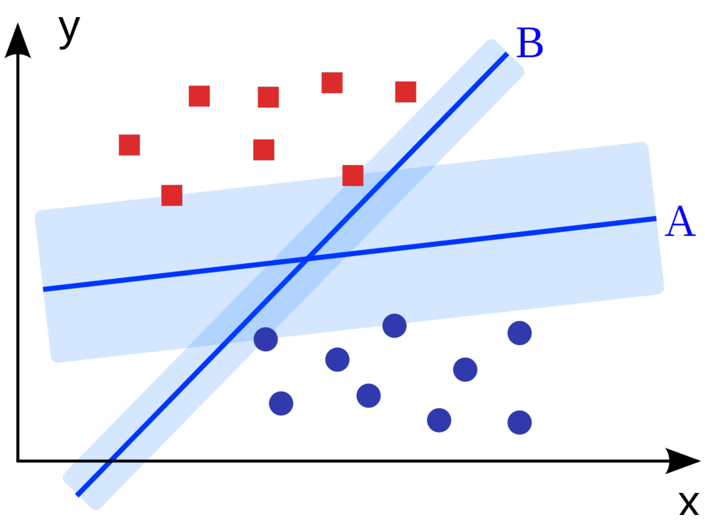
$$\operatorname{argmin} \sum f_i(x)$$

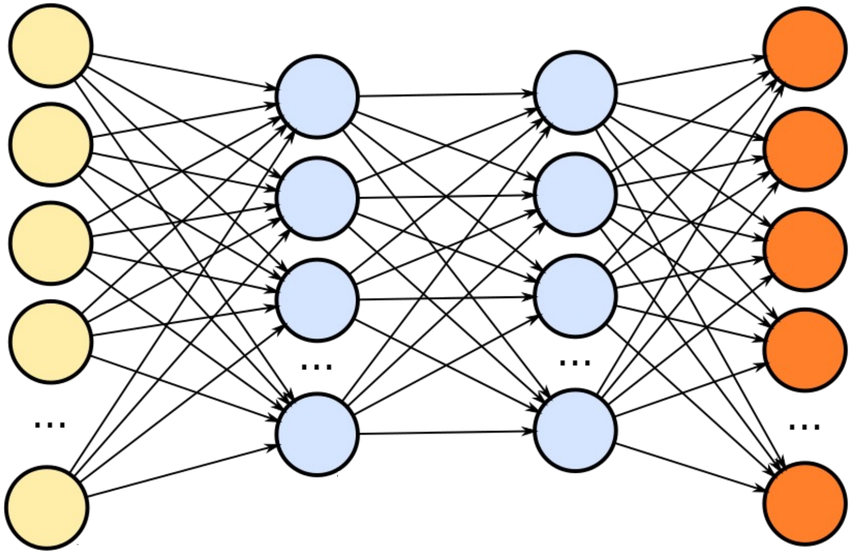
Machine Learning

- Data is distributed across different agents



- Data is distributed across different agents → Collaborate to learn





Machine Learning

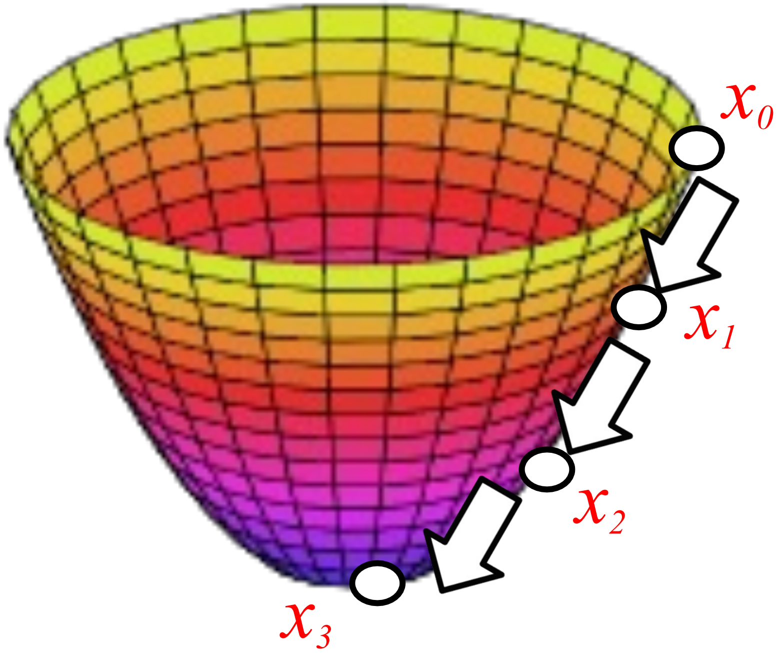
Minimize global loss

$$\sum f_i(x)$$

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

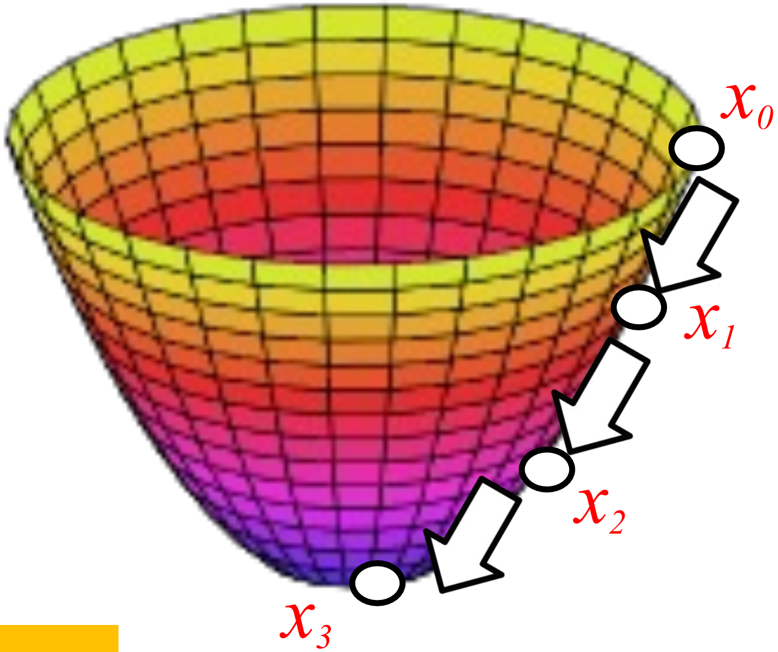
Gradient Method

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



Gradient Method

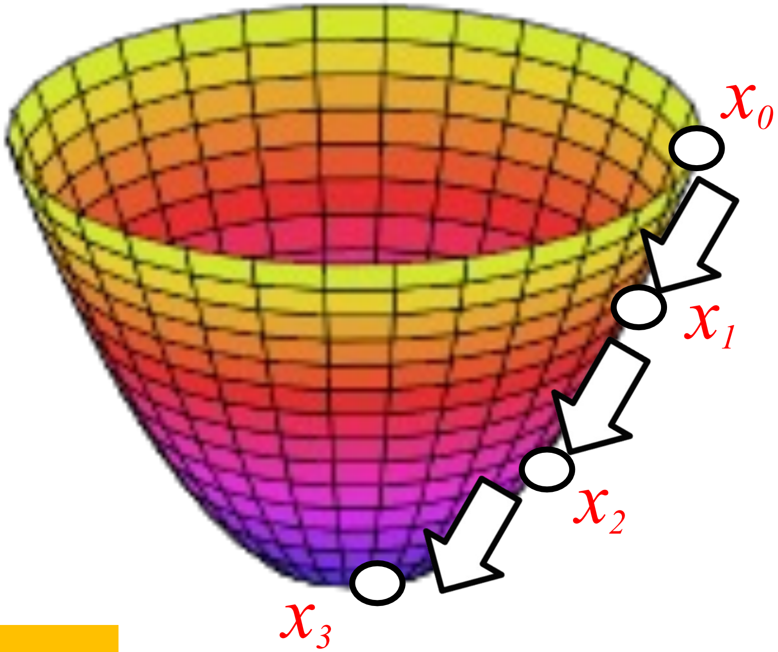
$$f(x) = \sum f_i(x)$$



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

Gradient Method

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



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

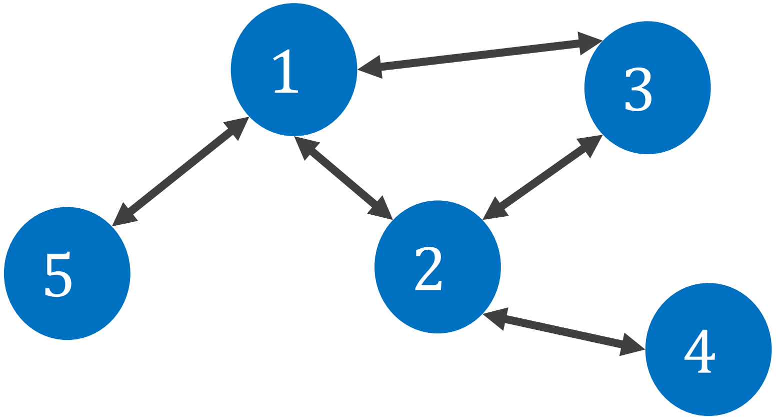
Distributed Optimization

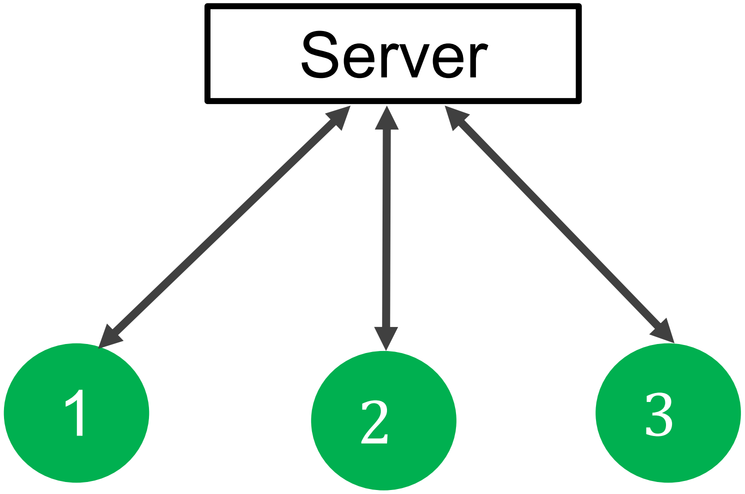
- Each agent i knows own cost function $f_i(x)$ and it can compute $\nabla f_i(x)$

- Need to cooperate to minimize $\sum f_i(x)$

➔ Distributed algorithms

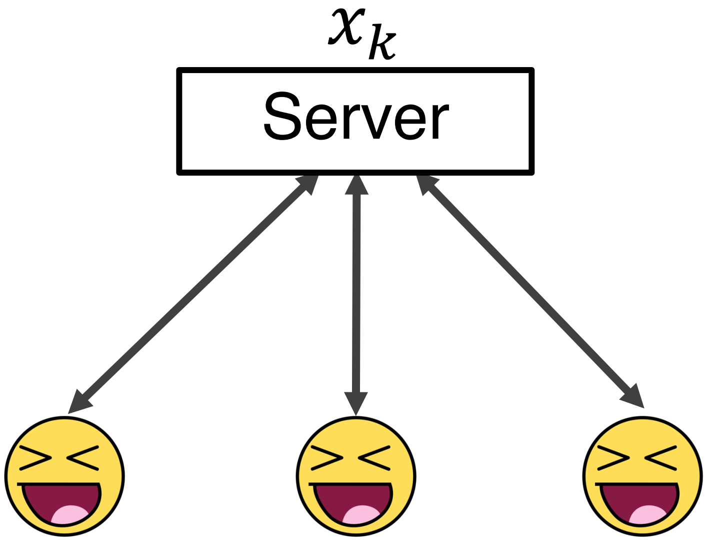
Architectures





Parameter Server

- Server maintains estimate x_k

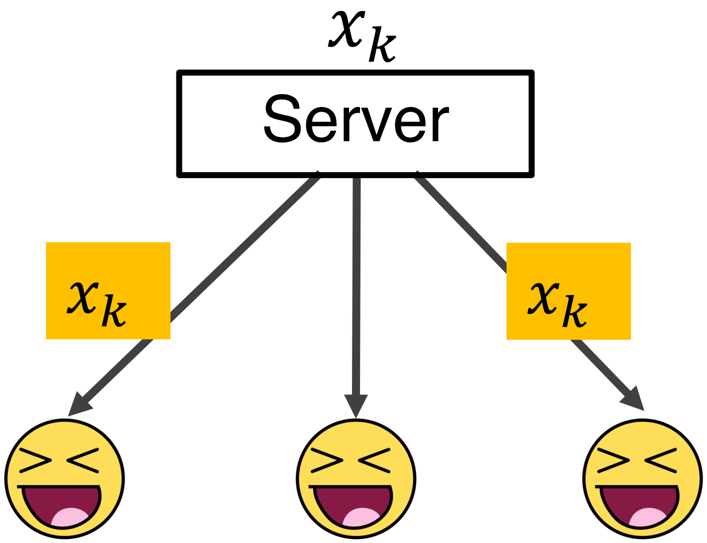


Parameter Server

- Server maintains estimate x_k

In each iteration

- Agent i
  - Receives x_k from server

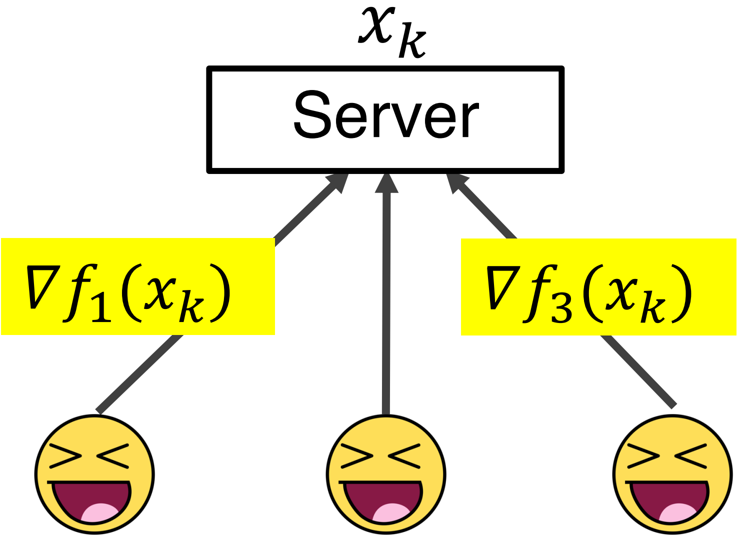


Parameter Server

- Server maintains estimate x_k

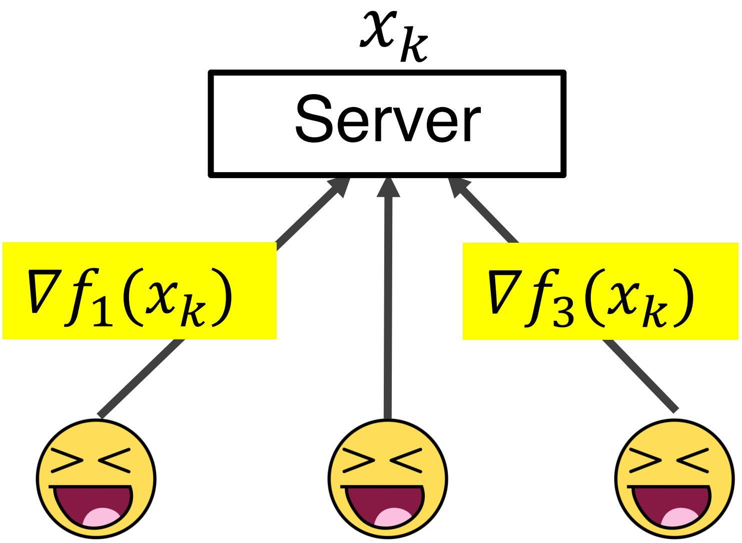
In each iteration

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



Parameter Server

- Server maintains estimate x_k



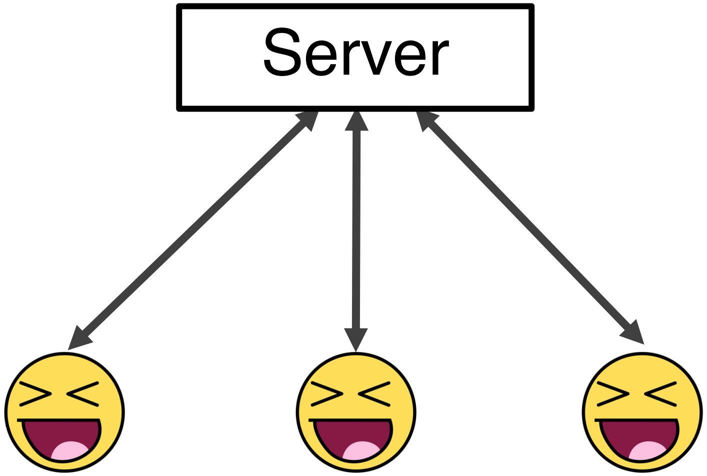
<u>In each iteration</u>

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

- Server updates estimate

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

Many Variations



… stochastic optimization

… asynchronous

… gradient compression

… acceleration

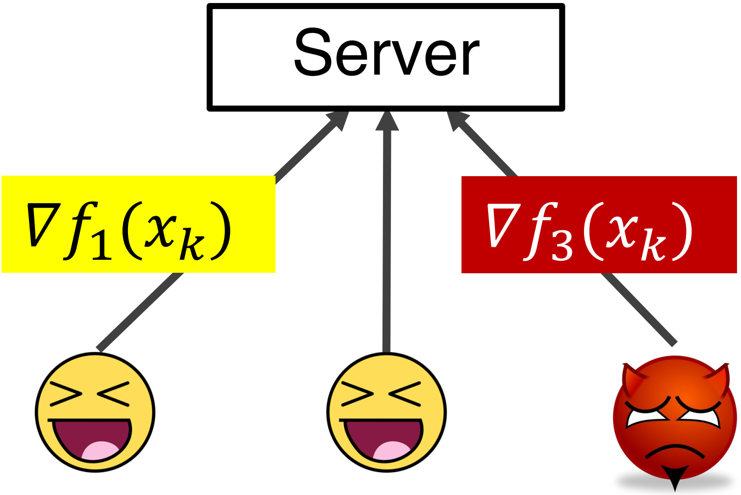
… shared memory

Adversarial Agents

- Fault-tolerant distributed optimization

$$f_1(x) + f_2(x) + f_3(x)$$

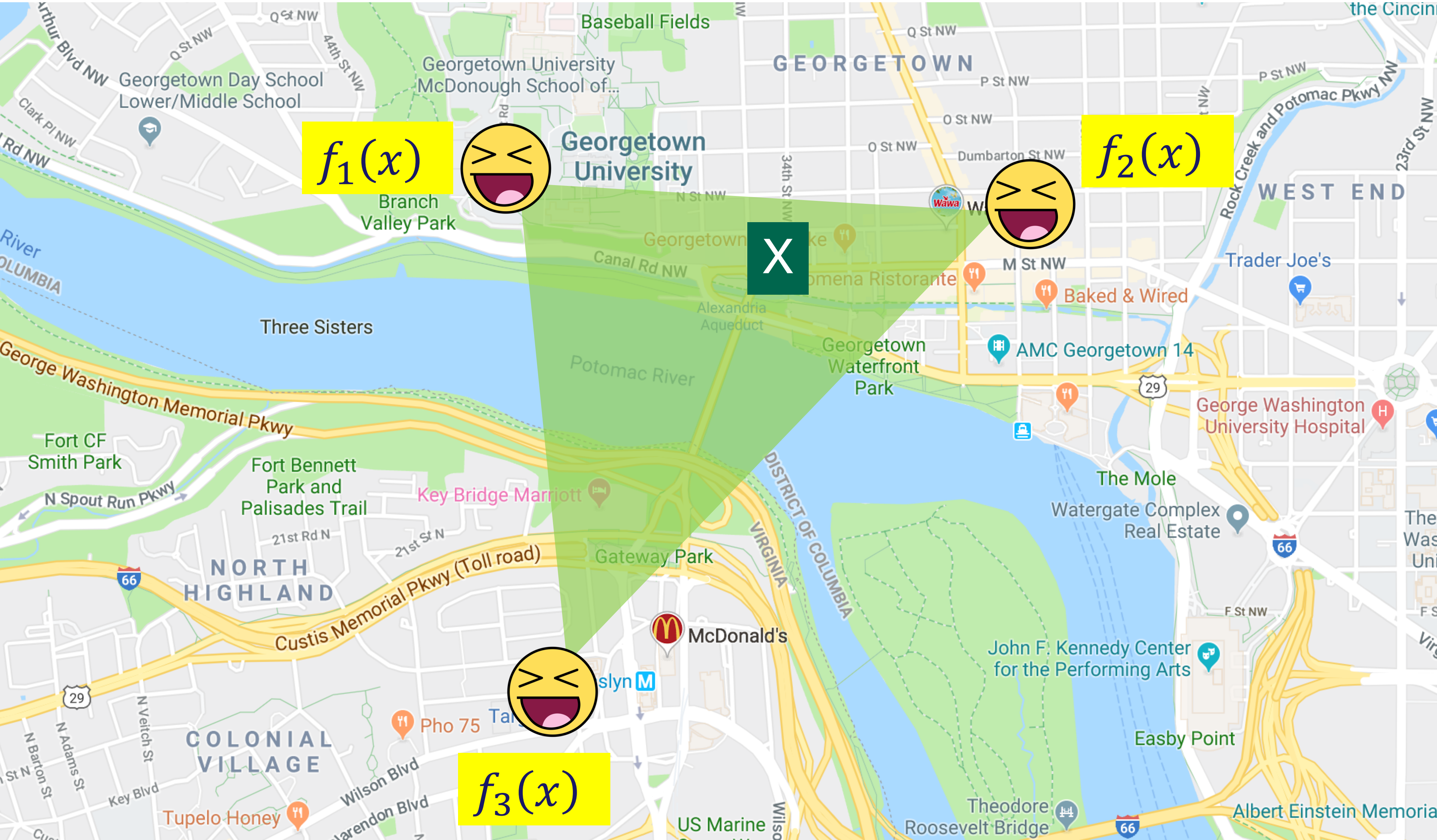
How to optimize if agents inject bogus information?



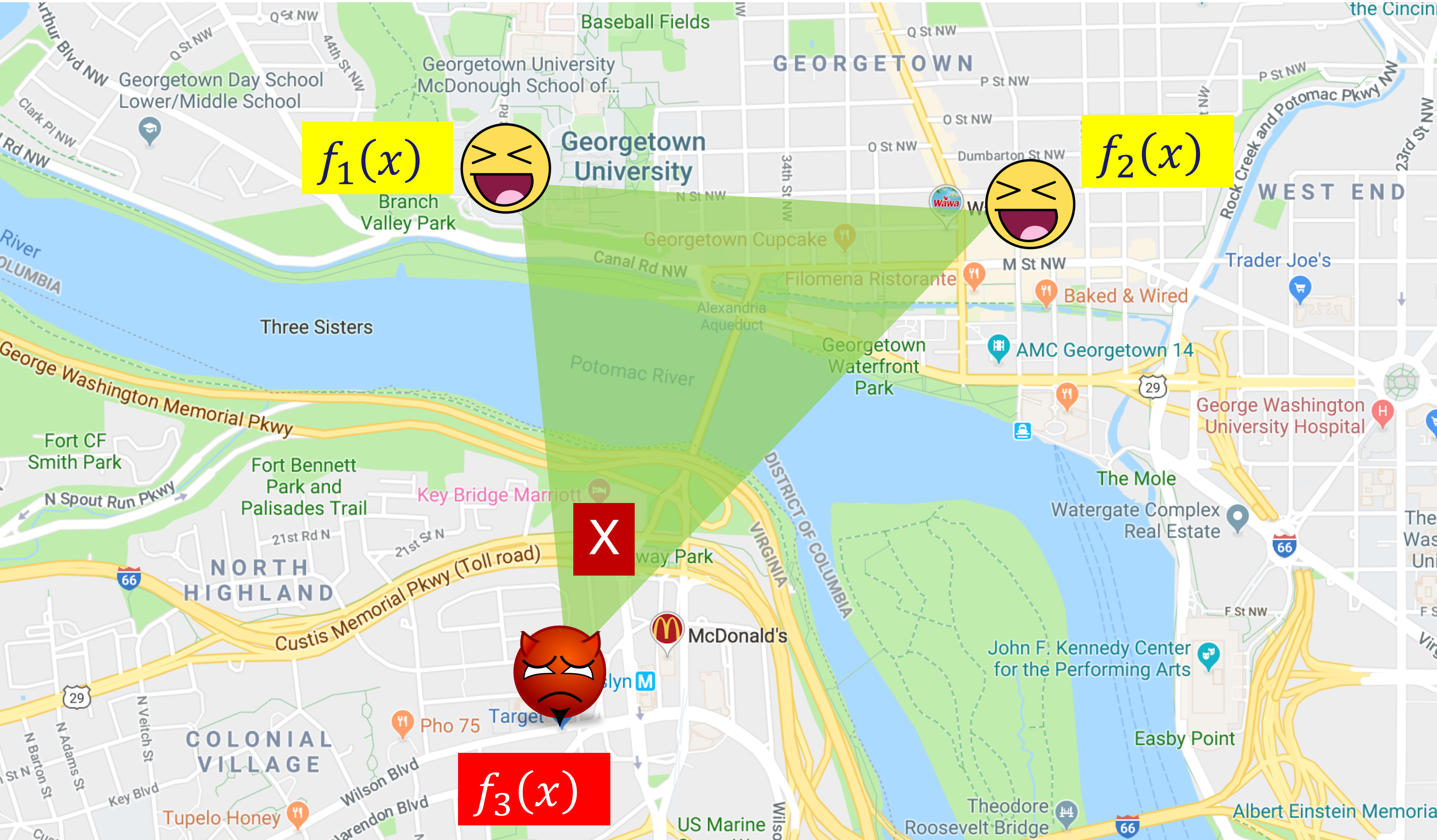
Fault-Tolerant Optimization

2015 …

Rendezvous


$f_1(x)$
$f_2(x)$
$f_3(x)$
X
Baseball Fields
GEORGETOWN
Georgetown University McDonough School of...
Georgetown Day School Lower/Middle School
Georgetown University
Branch Valley Park
WEST END
Trader Joe's
M St NW
Baked & Wired
Canal Rd NW
Alexandria Aqueduct
Three Sisters
Potomac River
Georgetown Waterfront Park
AMC Georgetown 14
George Washington Memorial Pkwy
George Washington University Hospital
Fort CF Smith Park
Fort Bennett Park and Palisades Trail
Key Bridge Marriott
The Mole
Watergate Complex Real Estate
NORTH HIGHLAND
Custis Memorial Pkwy (Toll road)
Gateway Park
DISTRICT OF COLUMBIA
VIRGINIA
McDonald's
John F. Kennedy Center for the Performing Arts
COLONIAL VILLAGE
Pho 75
Tupelo Honey
Wilson Blvd
Clarendon Blvd
Easby Point
Theodore Roosevelt Bridge
Albert Einstein Memoria
US Marine
Rock Creek and Potomac Pkwy NW
N Spout Run Pkwy
Key Blvd
21st Rd N
21st St N

Rendezvous


$f_1(x)$
$f_2(x)$
X
$f_3(x)$
Baseball Fields
Q St NW
GEORGETOWN
Georgetown University McDonough School of...
Georgetown Day School Lower/Middle School
Georgetown University
Branch Valley Park
P St NW
O St NW
Dumbarton St NW
N St NW
34th St NW
WEST END
Rock Creek and Potomac Pkwy NW
23rd St NW
Georgetown Cupcake
Canal Rd NW
Filomena Ristorante
M St NW
Baked & Wired
Trader Joe's
Alexandria Aqueduct
Georgetown Waterfront Park
AMC Georgetown 14
29
George Washington University Hospital
Three Sisters
Potomac River
George Washington Memorial Pkwy
Fort CF Smith Park
Fort Bennett Park and Palisades Trail
N Spout Run Pkwy
Key Bridge Marriott
21st Rd N
21st St N
NORTH HIGHLAND
66
Custis Memorial Pkwy (Toll road)
DISTRICT OF COLUMBIA
VIRGINIA
The Mole
Watergate Complex Real Estate
F St NW
McDonald's
John F. Kennedy Center for the Performing Arts
Pho 75
Target
COLONIAL VILLAGE
Wilson Blvd
Clarendon Blvd
Key Blvd
Tupelo Honey
N Veitch St
N Adams St
N Barton St
US Marine
Easby Point
Theodore Roosevelt Bridge
Albert Einstein Memoria
the Cincin
River
OLUMBIA
Clark Pl NW
44th St NW
rthur Blvd NW

Machine Learning

Faulty agent can adversely affect model parameters

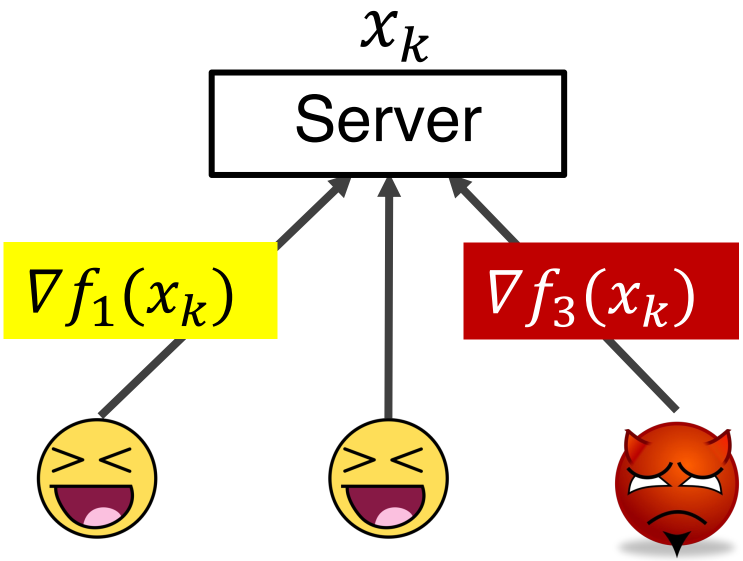
→

Minimize global loss

$$\sum f_i(x)$$

Parameter Server

- Server maintains estimate x_k



In each iteration

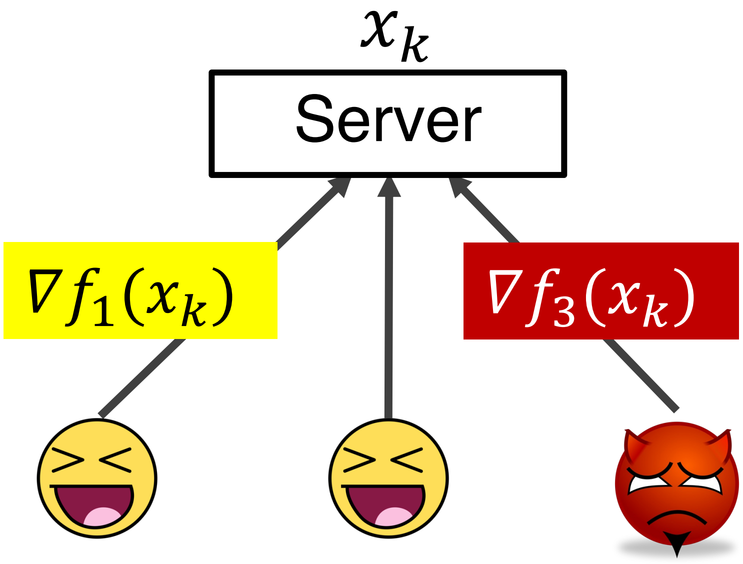
- Agent i
  - Downloads x_k from server
  - Uploads gradient $\nabla f_i(x_k)$

- Server updates estimate

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

Parameter Server

- Server maintains estimate x_k



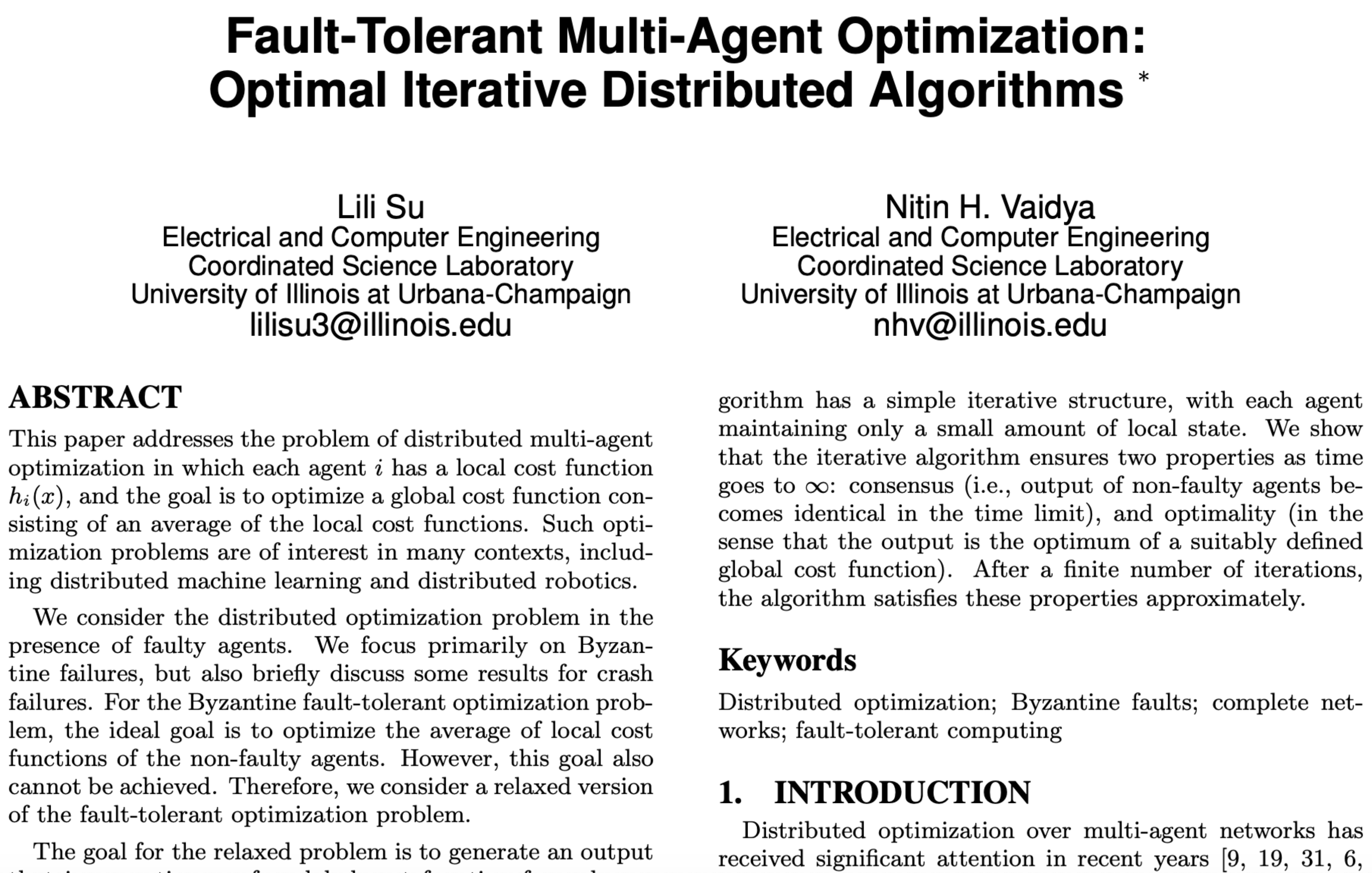
In each iteration

- Agent i
  - Downloads x_k from server
  - Uploads gradient $\nabla f_i(x_k)$

- Server updates estimate

$$x_{k+1} \longleftarrow x_k - \lambda \, \text{Filtered–Gradient}$$

2016



Fault-Tolerant Multi-Agent Optimization: Optimal Iterative Distributed Algorithms *

Lili Su
Electrical and Computer Engineering
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
lilisu3@illinois.edu

Nitin H. Vaidya
Electrical and Computer Engineering
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@illinois.edu

ABSTRACT

This paper addresses the problem of distributed multi-agent optimization in which each agent i has a local cost function $h_i(x)$, and the goal is to optimize a global cost function consisting of an average of the local cost functions. Such optimization problems are of interest in many contexts, including distributed machine learning and distributed robotics.

We consider the distributed optimization problem in the presence of faulty agents. We focus primarily on Byzantine failures, but also briefly discuss some results for crash failures. For the Byzantine fault-tolerant optimization problem, the ideal goal is to optimize the average of local cost functions of the non-faulty agents. However, this goal also cannot be achieved. Therefore, we consider a relaxed version of the fault-tolerant optimization problem.

The goal for the relaxed problem is to generate an output

gorithm has a simple iterative structure, with each agent maintaining only a small amount of local state. We show that the iterative algorithm ensures two properties as time goes to ∞: consensus (i.e., output of non-faulty agents becomes identical in the time limit), and optimality (in the sense that the output is the optimum of a suitably defined global cost function). After a finite number of iterations, the algorithm satisfies these properties approximately.

Keywords

Distributed optimization; Byzantine faults; complete networks; fault-tolerant computing

1. INTRODUCTION

Distributed optimization over multi-agent networks has received significant attention in recent years [9, 19, 31, 6, But what do we mean by fault-tolerance?

Fault-Tolerance

- Optimize over only good agents … set G

Fault-Tolerance

- Optimize over only good agents … set *G*

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Is this achievable?

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Is this achievable?

It Depends

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Is this achievable?

It Depends

Independent functions

"Enough" redundancy

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

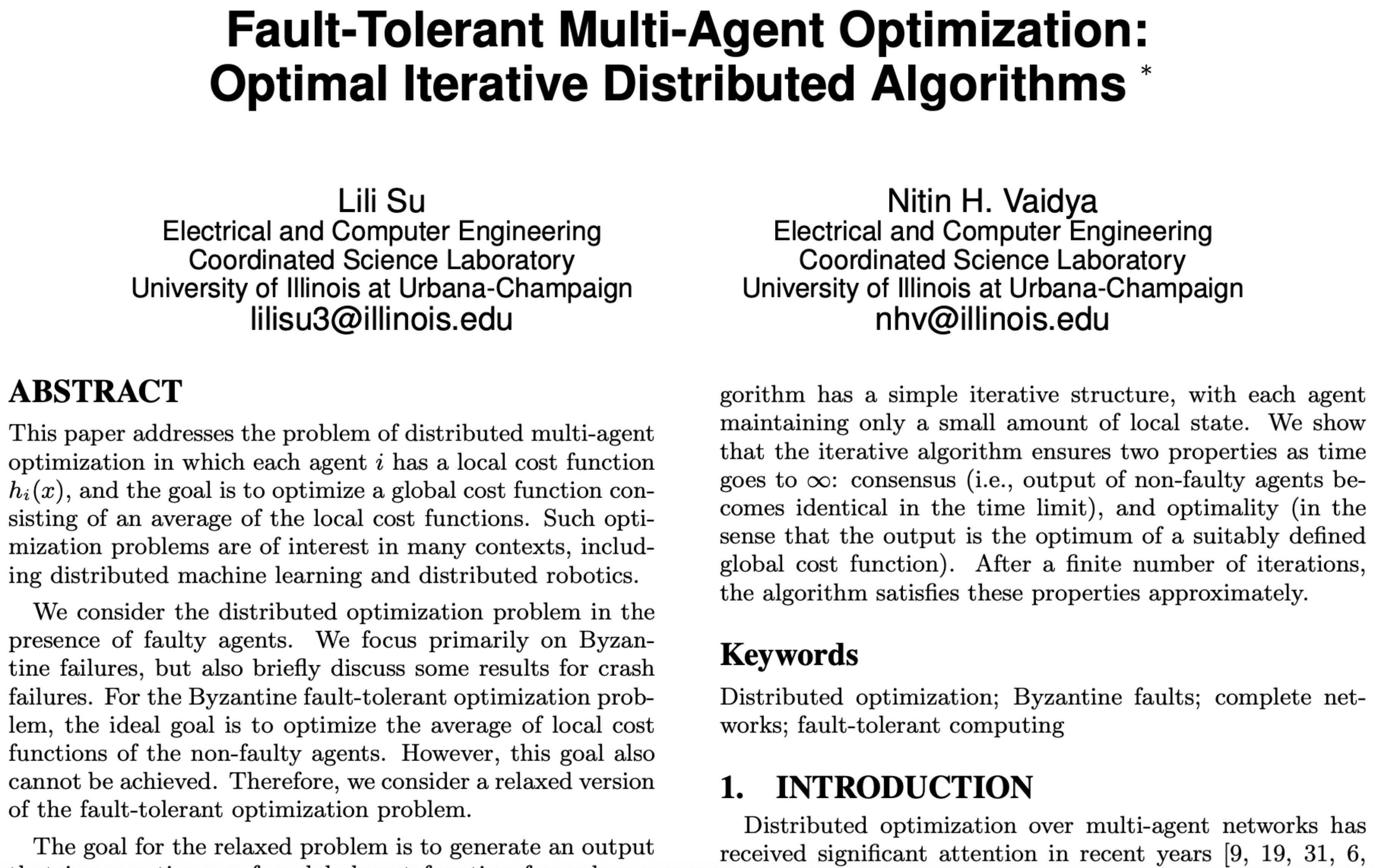
Is this achievable?

Independent functions

“Enough” redundancy

Approximate

2016



Fault-Tolerant Multi-Agent Optimization: Optimal Iterative Distributed Algorithms *

Lili Su
Electrical and Computer Engineering
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
lilisu3@illinois.edu

Nitin H. Vaidya
Electrical and Computer Engineering
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@illinois.edu

ABSTRACT

This paper addresses the problem of distributed multi-agent optimization in which each agent i has a local cost function $h_i(x)$, and the goal is to optimize a global cost function consisting of an average of the local cost functions. Such optimization problems are of interest in many contexts, including distributed machine learning and distributed robotics.

We consider the distributed optimization problem in the presence of faulty agents. We focus primarily on Byzantine failures, but also briefly discuss some results for crash failures. For the Byzantine fault-tolerant optimization problem, the ideal goal is to optimize the average of local cost functions of the non-faulty agents. However, this goal also cannot be achieved. Therefore, we consider a relaxed version of the fault-tolerant optimization problem.

The goal for the relaxed problem is to generate an output

gorithm has a simple iterative structure, with each agent maintaining only a small amount of local state. We show that the iterative algorithm ensures two properties as time goes to ∞: consensus (i.e., output of non-faulty agents becomes identical in the time limit), and optimality (in the sense that the output is the optimum of a suitably defined global cost function). After a finite number of iterations, the algorithm satisfies these properties approximately.

Keywords

Distributed optimization; Byzantine faults; complete networks; fault-tolerant computing

1. INTRODUCTION

Distributed optimization over multi-agent networks has received significant attention in recent years [9, 19, 31, 6,

Approximation

$$\operatorname{argmin} \textstyle\sum_{i \in G} f_i(x) = \operatorname{argmin} \textstyle\sum_{i \in G} \frac{1}{|G|} f_i(x)$$

Approximation

$$\operatorname{argmin} \sum_{i \in G} f_i(x) = \operatorname{argmin} \sum_{i \in G} \frac{1}{|G|} f_i(x)$$

$$\operatorname{argmin} \sum_{i \in G} \alpha_i \, f_i(x)$$

without necessarily knowing α_i’s

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Is this achievable?

Independent functions

“Enough” redundancy

Approximate

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Is this achievable?

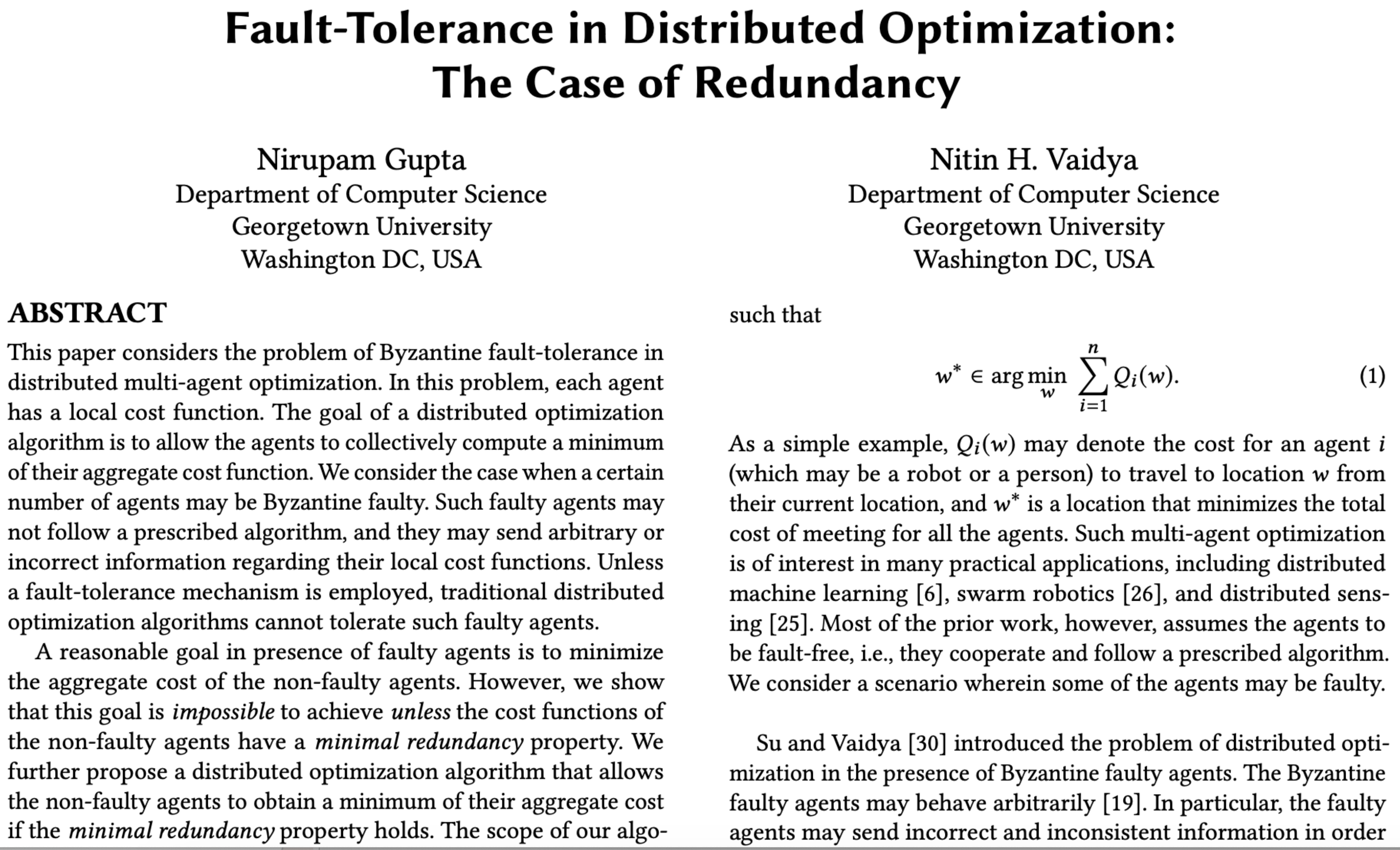
Independent functions

“Enough” redundancy

Approximate

Exact

(2020)



Fault-Tolerance in Distributed Optimization: The Case of Redundancy

Nirupam Gupta
Department of Computer Science
Georgetown University
Washington DC, USA

Nitin H. Vaidya
Department of Computer Science
Georgetown University
Washington DC, USA

ABSTRACT

This paper considers the problem of Byzantine fault-tolerance in distributed multi-agent optimization. In this problem, each agent has a local cost function. The goal of a distributed optimization algorithm is to allow the agents to collectively compute a minimum of their aggregate cost function. We consider the case when a certain number of agents may be Byzantine faulty. Such faulty agents may not follow a prescribed algorithm, and they may send arbitrary or incorrect information regarding their local cost functions. Unless a fault-tolerance mechanism is employed, traditional distributed optimization algorithms cannot tolerate such faulty agents.

A reasonable goal in presence of faulty agents is to minimize the aggregate cost of the non-faulty agents. However, we show that this goal is *impossible* to achieve *unless* the cost functions of the non-faulty agents have a *minimal redundancy* property. We further propose a distributed optimization algorithm that allows the non-faulty agents to obtain a minimum of their aggregate cost if the *minimal redundancy* property holds. The scope of our algo-

such that

$$w^* \in \arg\min_{w} \sum_{i=1}^{n} Q_i(w). \tag{1}$$

As a simple example, $Q_i(w)$ may denote the cost for an agent i (which may be a robot or a person) to travel to location w from their current location, and w^* is a location that minimizes the total cost of meeting for all the agents. Such multi-agent optimization is of interest in many practical applications, including distributed machine learning [6], swarm robotics [26], and distributed sensing [25]. Most of the prior work, however, assumes the agents to be fault-free, i.e., they cooperate and follow a prescribed algorithm. We consider a scenario wherein some of the agents may be faulty.

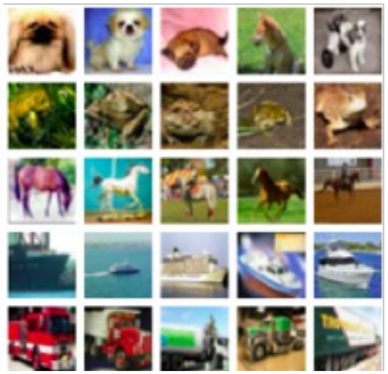
Su and Vaidya [30] introduced the problem of distributed optimization in the presence of Byzantine faulty agents. The Byzantine faulty agents may behave arbitrarily [19]. In particular, the faulty agents may send incorrect and inconsistent information in order

An “Extreme” Example

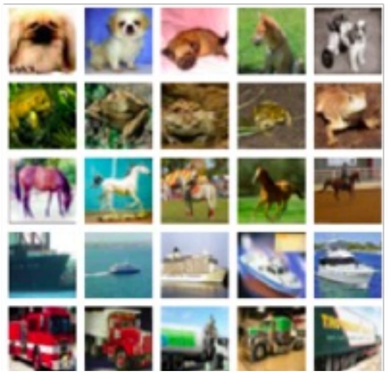
Stochastic machine learning

- Agents draw samples from identical data distribution

Agent 1



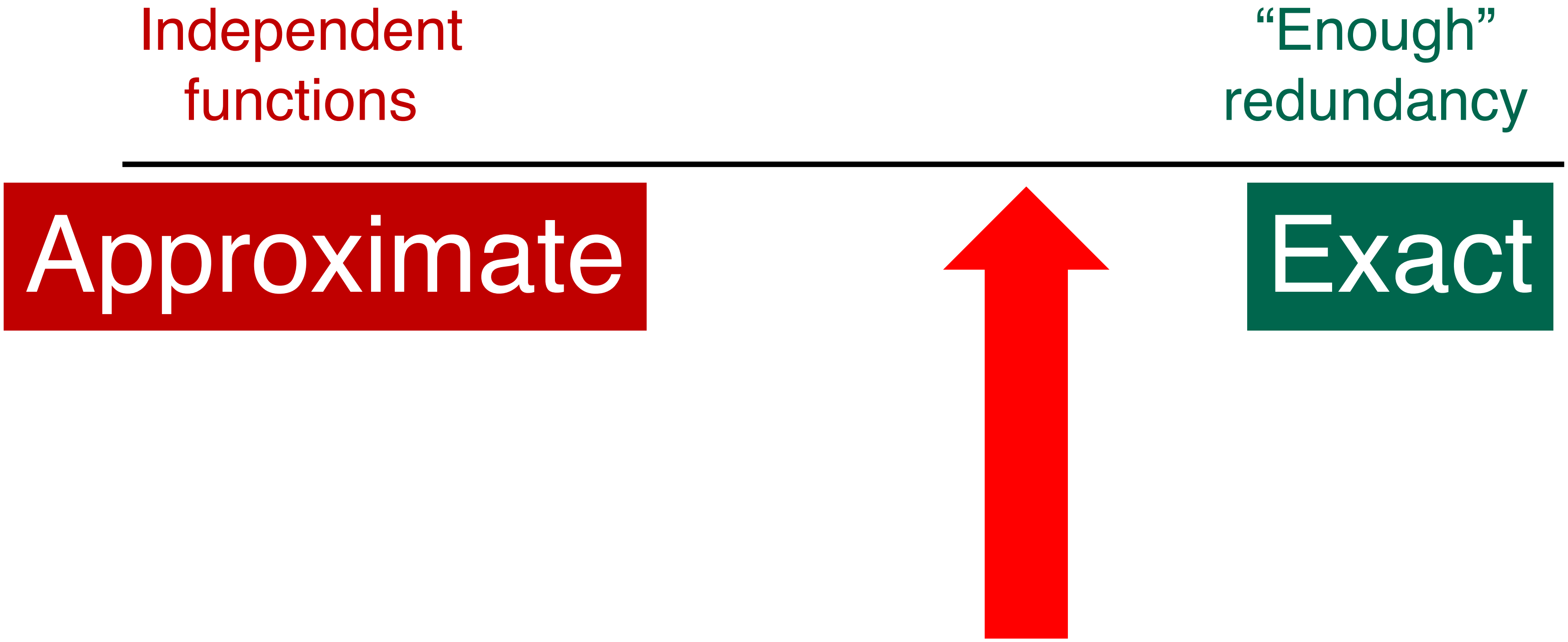
Agent 2



- Filter on stochastic gradients

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Is this achievable?



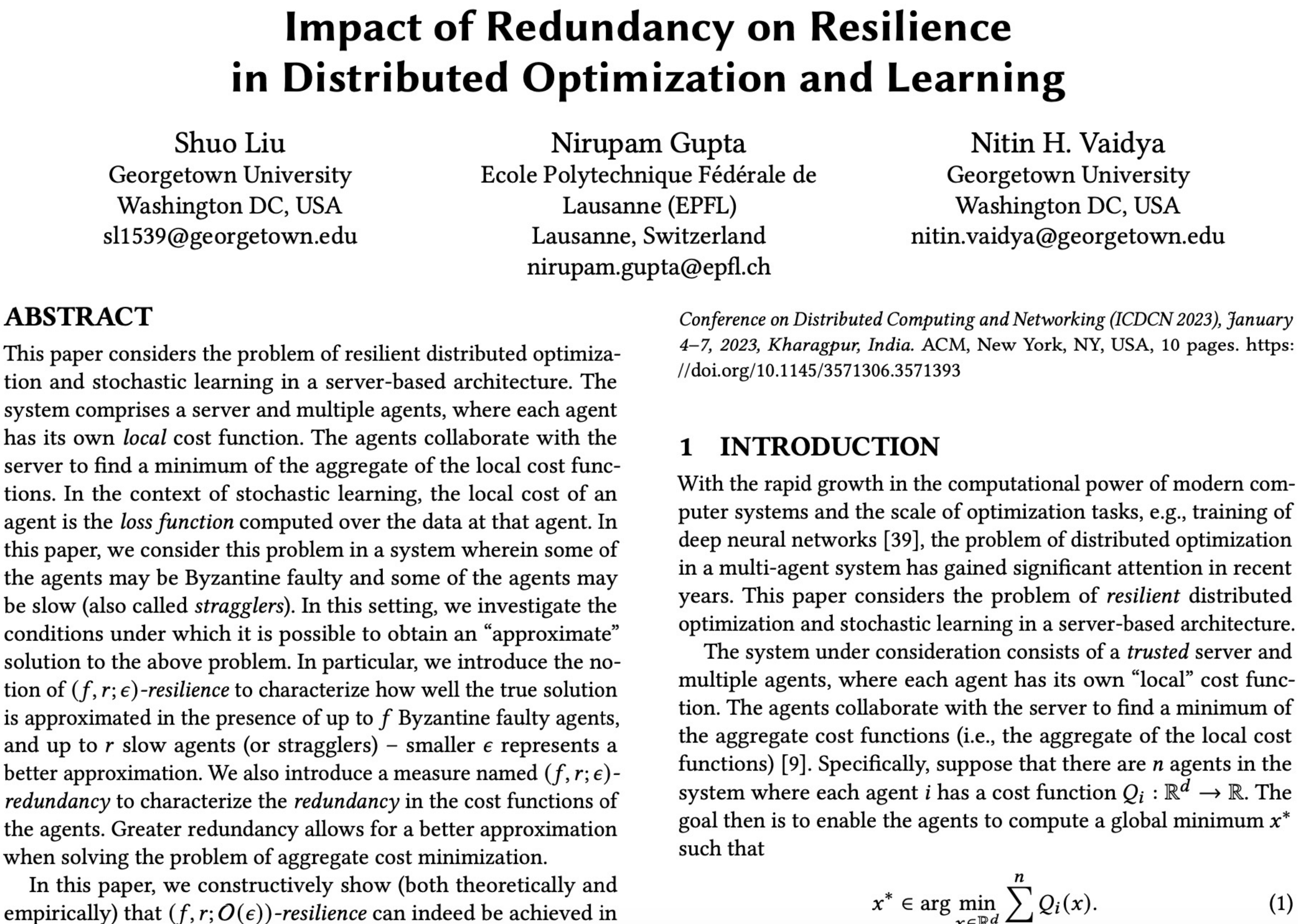
Independent functions

“Enough” redundancy

Approximate

Exact

(2023)



Impact of Redundancy on Resilience in Distributed Optimization and Learning

Shuo Liu
Georgetown University
Washington DC, USA
sl1539@georgetown.edu

Nirupam Gupta
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
nirupam.gupta@epfl.ch

Nitin H. Vaidya
Georgetown University
Washington DC, USA
nitin.vaidya@georgetown.edu

ABSTRACT

This paper considers the problem of resilient distributed optimization and stochastic learning in a server-based architecture. The system comprises a server and multiple agents, where each agent has its own *local* cost function. The agents collaborate with the server to find a minimum of the aggregate of the local cost functions. In the context of stochastic learning, the local cost of an agent is the *loss function* computed over the data at that agent. In this paper, we consider this problem in a system wherein some of the agents may be Byzantine faulty and some of the agents may be slow (also called *stragglers*). In this setting, we investigate the conditions under which it is possible to obtain an "approximate" solution to the above problem. In particular, we introduce the notion of $(f, r; \epsilon)$-*resilience* to characterize how well the true solution is approximated in the presence of up to f Byzantine faulty agents, and up to r slow agents (or stragglers) – smaller ϵ represents a better approximation. We also introduce a measure named $(f, r; \epsilon)$-*redundancy* to characterize the *redundancy* in the cost functions of the agents. Greater redundancy allows for a better approximation when solving the problem of aggregate cost minimization.

In this paper, we constructively show (both theoretically and empirically) that $(f, r; \mathcal{O}(\epsilon))$-*resilience* can indeed be achieved in

Conference on Distributed Computing and Networking (ICDCN 2023), January 4–7, 2023, Kharagpur, India. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3571306.3571393

1 INTRODUCTION

With the rapid growth in the computational power of modern computer systems and the scale of optimization tasks, e.g., training of deep neural networks [39], the problem of distributed optimization in a multi-agent system has gained significant attention in recent years. This paper considers the problem of *resilient* distributed optimization and stochastic learning in a server-based architecture.

The system under consideration consists of a *trusted* server and multiple agents, where each agent has its own "local" cost function. The agents collaborate with the server to find a minimum of the aggregate cost functions (i.e., the aggregate of the local cost functions) [9]. Specifically, suppose that there are n agents in the system where each agent i has a cost function $Q_i : \mathbb{R}^d \to \mathbb{R}$. The goal then is to enable the agents to compute a global minimum x^* such that

$$x^* \in \arg\min_{x\in\mathbb{R}^d} \sum^{n} Q_i(x). \quad (1)$$

Status

- Many papers, various groups

  ... particularly fault-tolerant stochastic learning

  - Various filters for gradients
  - Variations on underlying assumptions

- A tutorial available from my website

- A survey to be available soon (from another group)

Moral of the Story #1

Natural, unanswered questions at the intersection of previously explored problem spaces

Picture from Wikipedia

Moral of the Story #2

Academia lets you work on things for which you may have no competence

Make the best use of the freedom

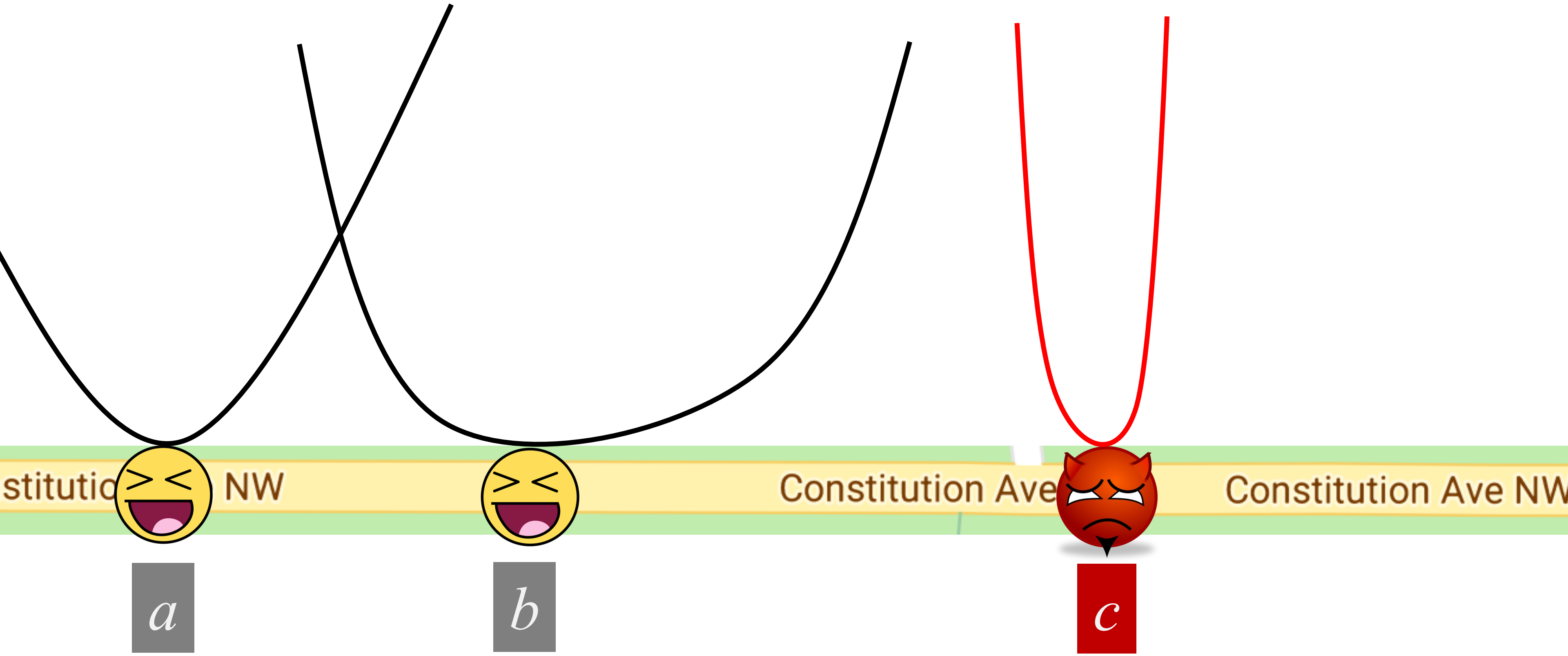
Thanks!

disc.georgetown.domains

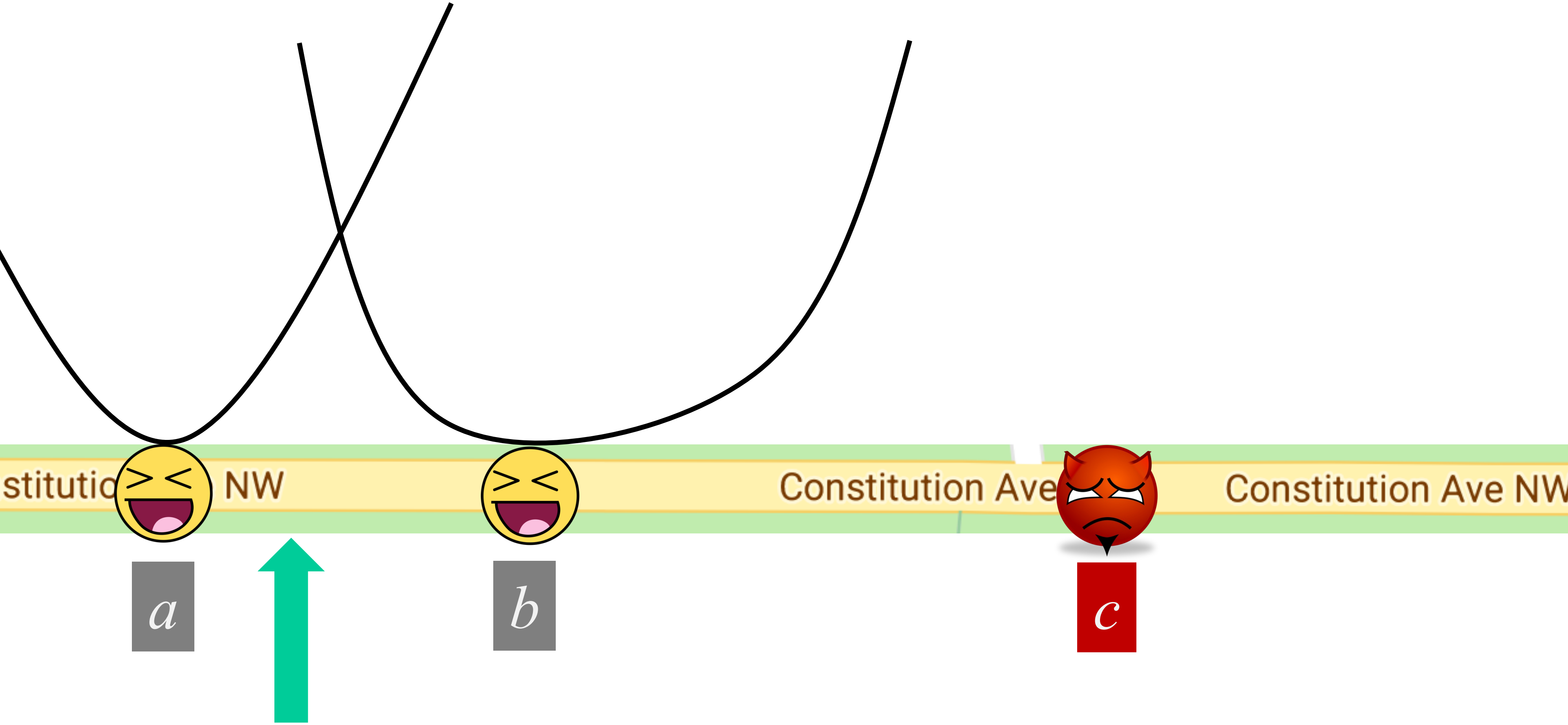
Thanks!

disc.georgetown.domains

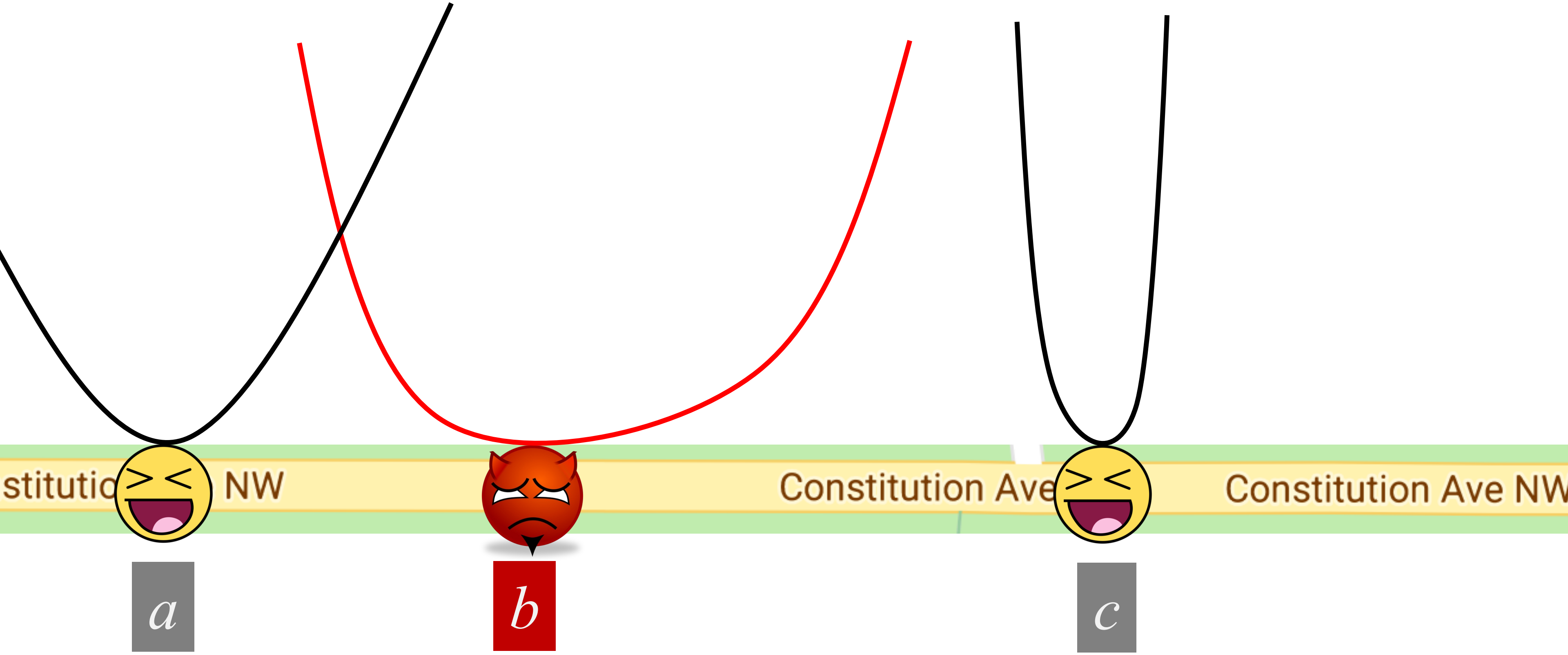
Independent Functions



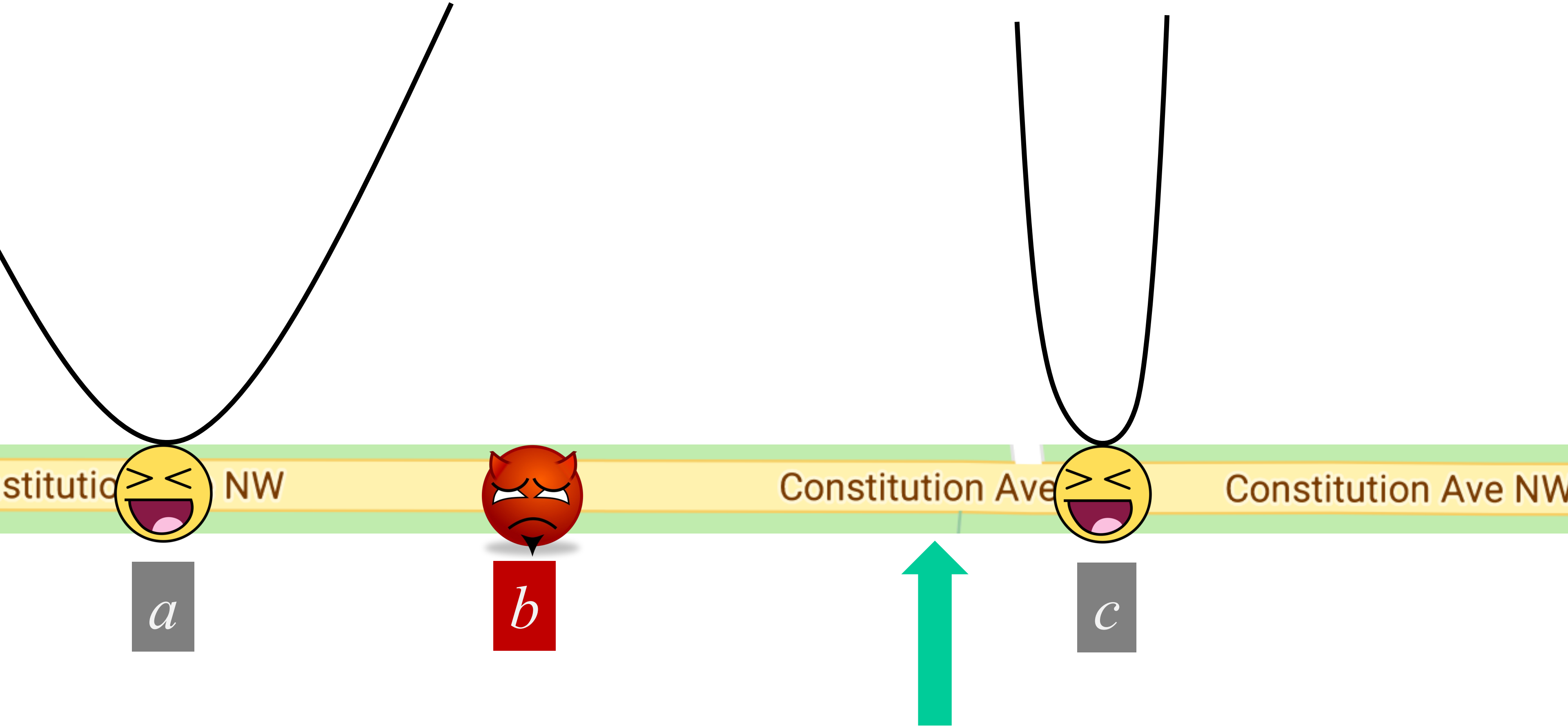
Independent Functions



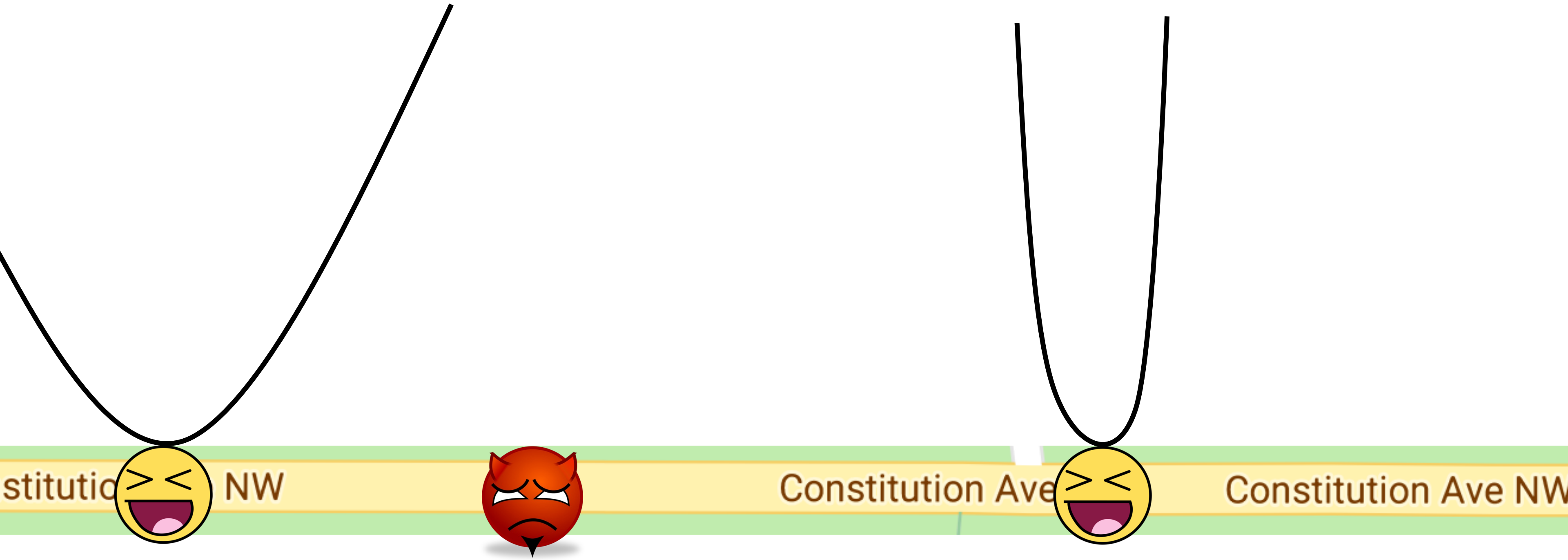
Independent Functions



Independent Functions



Independent Functions



Provably impossible to compute

$$\operatorname{argmin} \sum_{i \in G} f_i(x)$$

Norm Filter

- Clip the largest t norms to equal $t + 1$th norm

$$|\nabla f_1(x_k)| = 1$$

$$|\nabla f_2(x_k)| = 3$$

$$|\nabla f_3(x_k)| = 2$$

Norm Filter

- Clip the largest t norms to equal $t + 1$th norm

$$|\nabla f_1(x_k)| = 1$$

$$|\nabla f_2(x_k)| = 3$$

$$|\nabla f_3(x_k)| = 2$$

Filtered gradient = $\nabla f_1(x_k) \ + \frac{2}{3} \nabla f_2(x_k) \ + \nabla f_3(x_k)$

Norm Filter

- Clip the largest t norms to equal $t+1^{\text{th}}$ norm

$$|\nabla f_1(x_k)| = 1$$

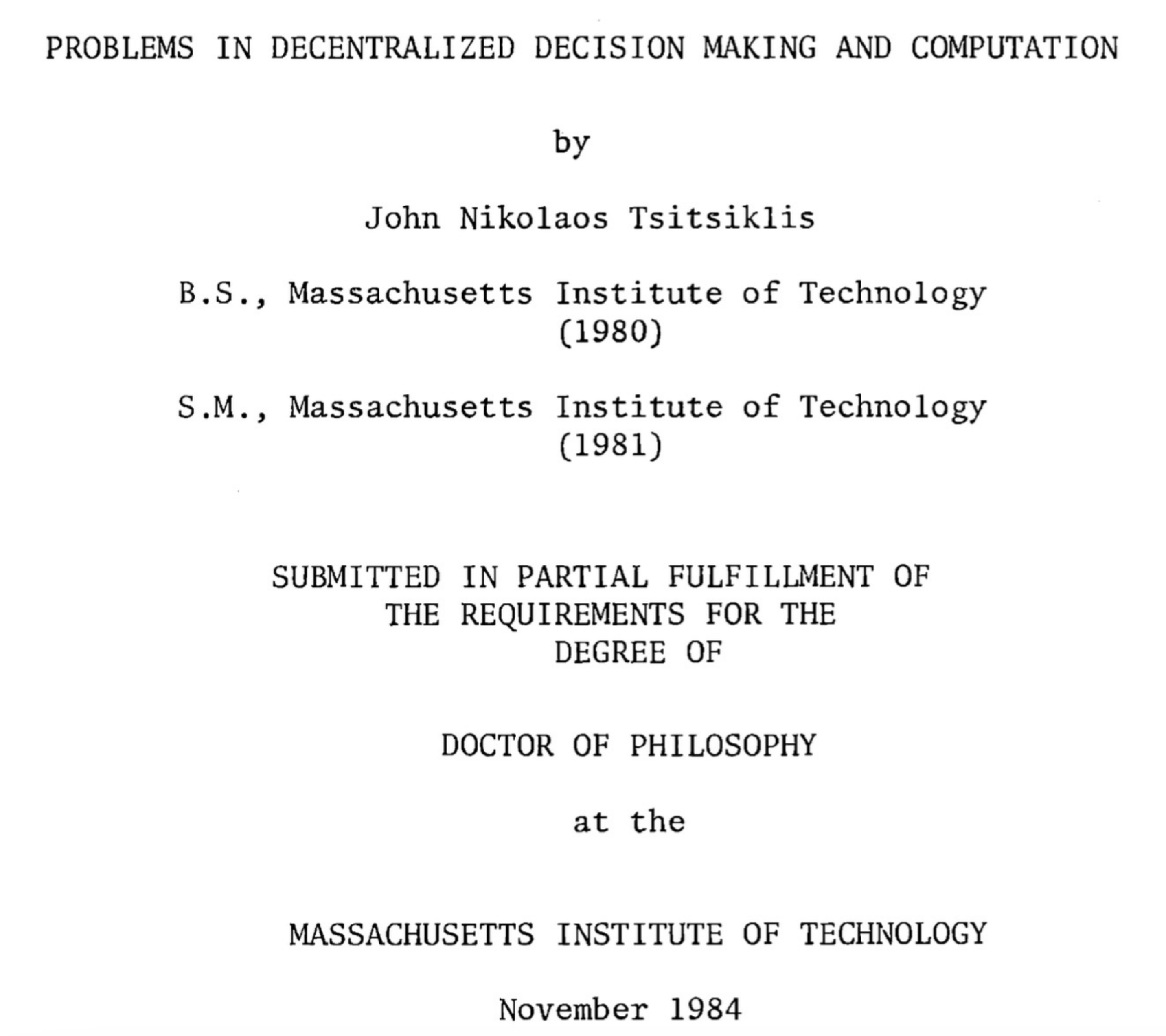
$$|\nabla f_2(x_k)| = 3$$

$$|\nabla f_3(x_k)| = 2$$

Filtered gradient = $\nabla f_1(x_k) + \frac{2}{3}\nabla f_2(x_k) + \nabla f_3(x_k)$

Exact optimum computed despite faulty agents

1984



PROBLEMS IN DECENTRALIZED DECISION MAKING AND COMPUTATION

by

John Nikolaos Tsitsiklis

B.S., Massachusetts Institute of Technology
(1980)

S.M., Massachusetts Institute of Technology
(1981)

SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE
DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

November 1984

An Example of Redundancy

n agents

f bad agents

- Aggregate cost of ANY $n-2f$ agents has argmin identical to desired $\mathrm{argmin} \sum_{i \in G} f_i(x)$

Another Approximation

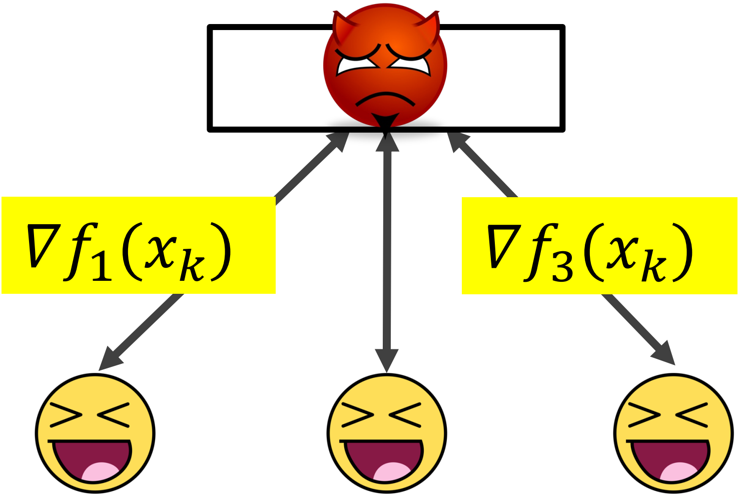
- Relax the notion of "enough redundancy"
- Produce output within distance ε of "true" minimum

$$\mathrm{argmin} \sum_{i \in G} f_i(x)$$

Challenges

- Privacy-preserving distributed optimization

How to collaborate without revealing own cost function?


$\nabla f_1(x_k)$
$\nabla f_3(x_k)$