Fault-tolerant Consensus in Directed Networks

Lewis Tseng

Boston College

Oct. 13, 2017

(joint work with Nitin H. Vaidya)

Fault-tolerant Consensus

Each node has an input

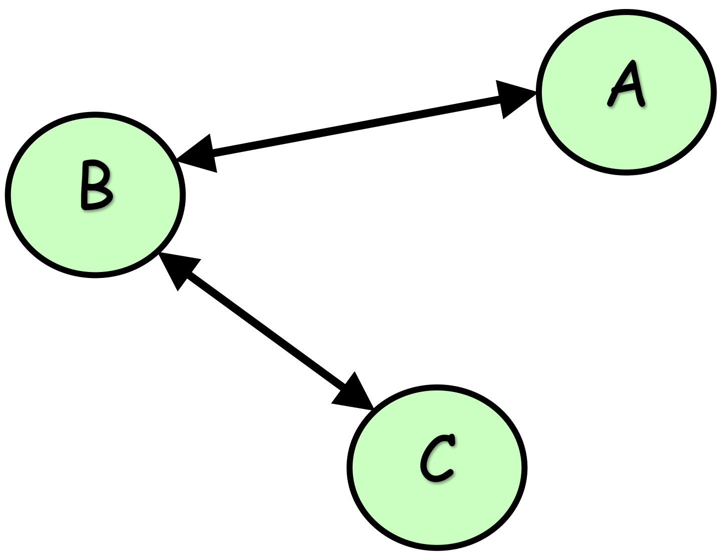
- Agreement: good nodes must agree
- Validity: some constraints on output

Exact vs. Approximate

- Termination

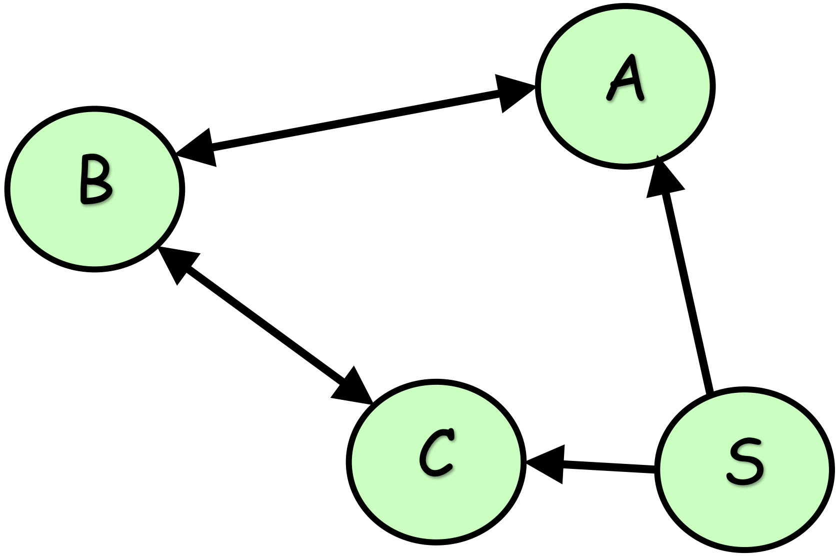
Message-Passing Communication

undirected graph



Message-Passing Communication

<u>directed</u> graph



- Partially connected
- links may *not* be bi-directional

Goal

Precise characterization of networks that can solve consensus

- Known for undirected graphs
- Unknown for directed graphs

Consensus

Fault Model	System/Output		Graph	Results
Crash	Synchronous	Exact		
		Approximate		
	Asynchronous			
Byzantine	Synchronous	Exact		
		Approximate		
	Asynchronous			

Consensus

Fault Model	System/Output		Graph	Results
Crash	Synchronous	Exact	Undirected	Well-known Results
			Directed	
		Approximate	Undirected	Decentralized Control (e.g., [Tsitsiklis ‘84] [Jadbabaei ‘03])
			Directed	
	Asynchronous		Undirected	
			Directed	
Byzantine	Synchronous	Exact	Undirected	[PSL ‘80] [FLM ‘85]
			Directed	
		Approximate	Undirected	[Dolev ‘83] [FLM ‘85]
			Directed	
	Asynchronous		Undirected	[Dolev ‘83] [FLM ‘85]
			Directed	

Consensus

Fault Model	System/Output		Graph	Results
Crash	Synchronous	Exact	Undirected	Well-known Results
			Directed	**PODC '15**
		Approximate	Undirected	Decentralized Control (e.g., [Tsitsiklis '84] [Jadbabaei '03])
			Directed	
	Asynchronous		Undirected	
			Directed	**PODC '15**
Byzantine	Synchronous	Exact	Undirected	[PSL '80] [FLM '85]
			Directed	**PODC '15**
		Approximate	Undirected	[Dolev '83] [FLM '85]
			Directed	**PODC '12**
	Asynchronous		Undirected	[Dolev '83] [FLM '85]
			Directed	**Open**

Why Directed Networks?

- Motivated by properties of wireless links
- Better understanding of network requirements for consensus
- Directed networks considered in several related contexts

Past Work on Directed Networks

- Decentralized control

  [Tsitsiklis ‘84],[Bertsekas, Tsitsiklis ‘97],[Jadbabaei et al. ‘03]

- Malicious fault model

  [Zhang et al. ‘12], [LeBlanc et al. ‘13]

- Different problems

  [Desmedt, Wang ‘02], [Bansal et al. ‘11], [Biely et al. ‘12], [Pagourtzis et al. ‘14], [Maurer et al. ‘14], [Biely et al. ‘14]

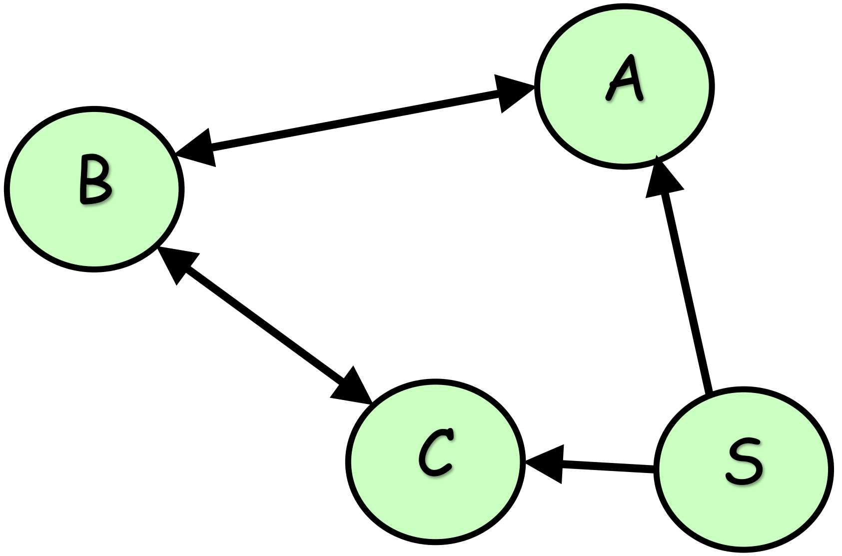
Algorithms

General Algorithm

- topology information

Iterative Algorithm

- local computation



This Talk: Exact Consensus

General Algorithm:

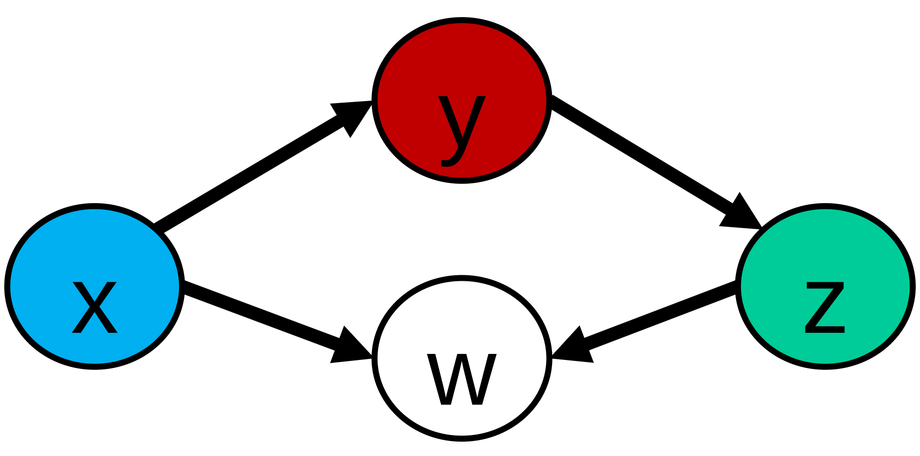
- Crash + Synchronous
- Byzantine + Synchronous

[Tseng and Vaidya, PODC ‘15]

Intuition

- Remove some nodes

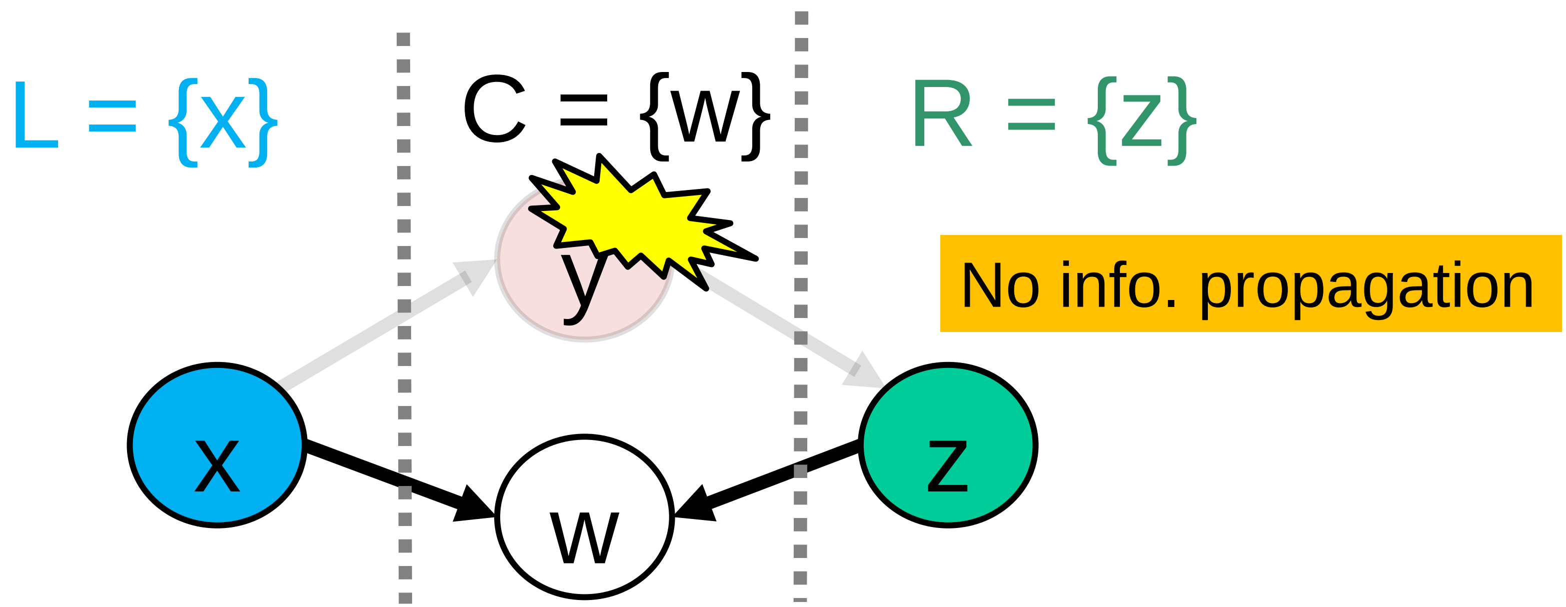
- For any node partition L, C, R,
  either L or R has <u>enough neighbors</u> from outside



Intuition

- Remove some nodes

- For any node partition L, C, R,
  either L or R has <u>enough neighbors</u> from outside

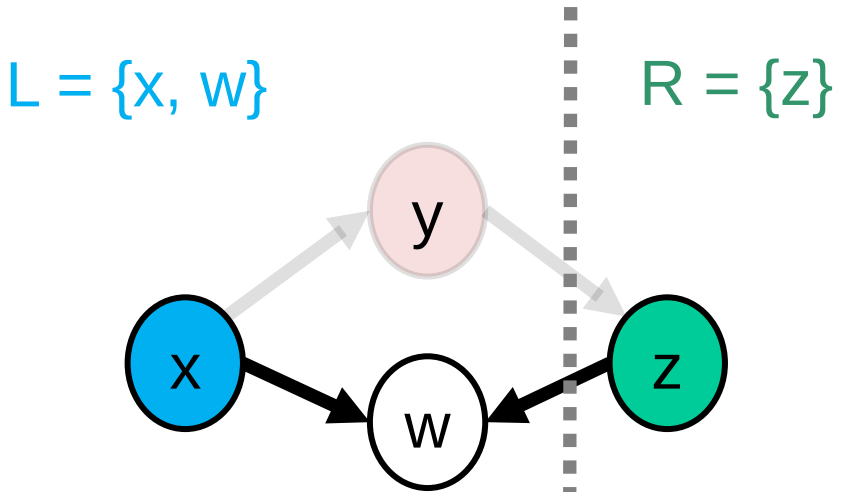


Why L, C, R?

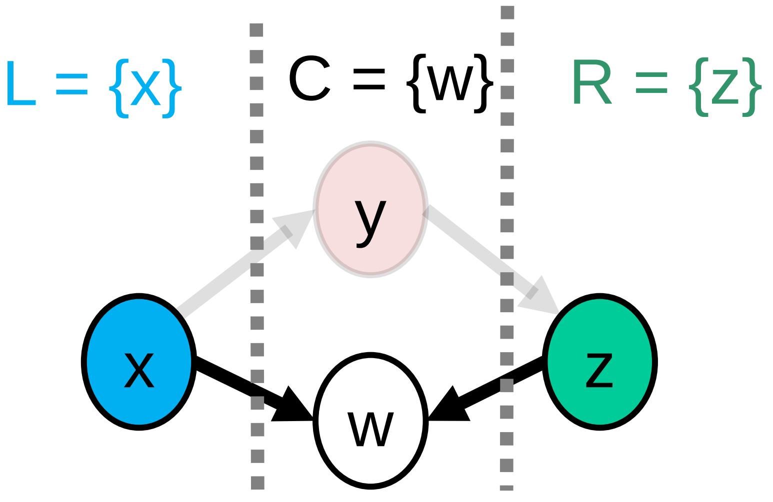
- Remove some nodes

- For any node partition L, C, R,
  either L or R has <u>enough neighbors</u> from outside

One-way info. propagation



No info. propagation



Crash Failures + Synchrony

Exact Consensus

Each node has a binary input

- Agreement: Good nodes agree on an *exact* value
- Validity: Agreed value is an input at some node
- Termination

Crash in Undirected Graphs

Known result:

$n > f$ and connectivity $> f$

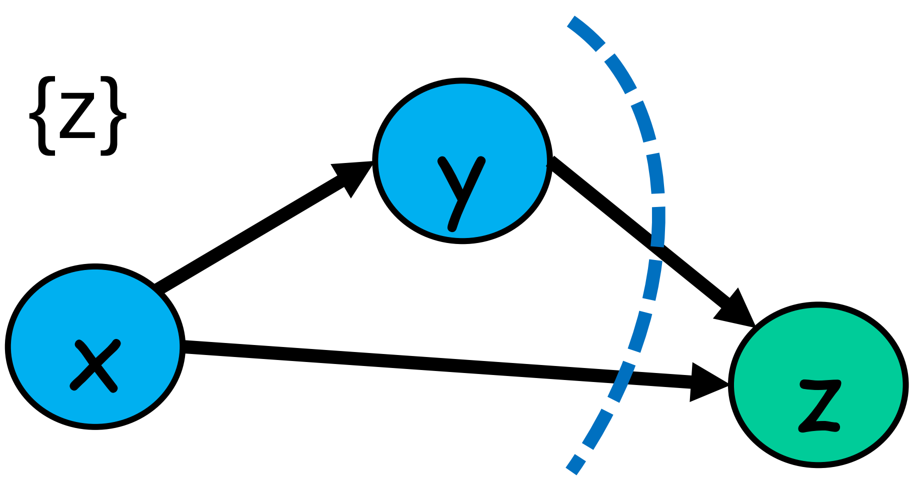
necessary and sufficient

n nodes, up to **f** failures

k-propagate

$A \xRightarrow{k} B$ if at least k distinct nodes in set A have links to nodes in set B

$\{x, y\} \xRightarrow{2} \{z\}$

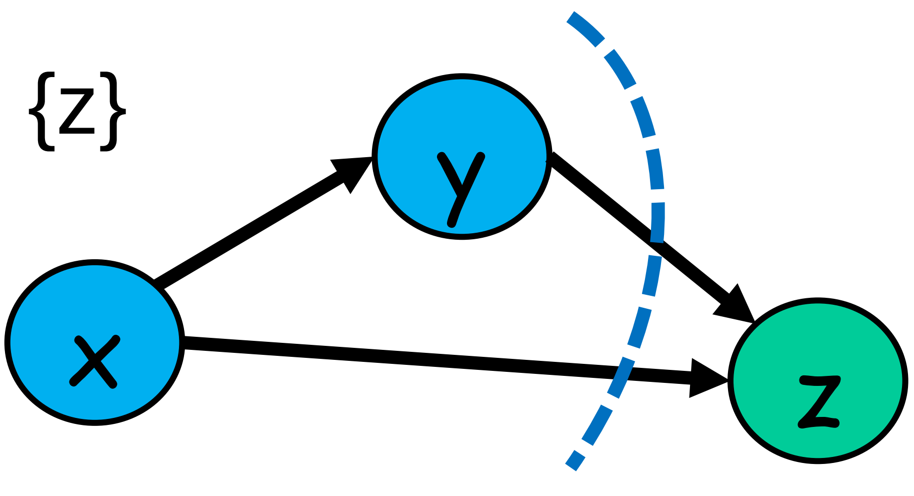


k-propagate

$A \xRightarrow{k} B$ if at least k distinct nodes in set A have links to nodes in set B

Whether set B has enough neighbors from outside?

$\{x, y\} \xRightarrow{2} \{z\}$



Crash Failures + Synchrony

Exact consensus possible iff

For any node partition L, C, R, F with

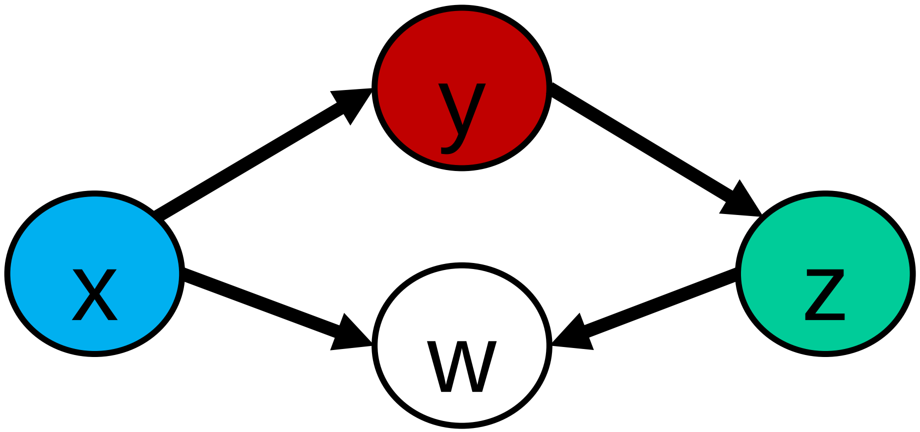
L, R non-empty <u>and</u> $|F| \leq f$,

$$L \cup C \xrightarrow{1} R$$

or

$$R \cup C \xrightarrow{1} L$$

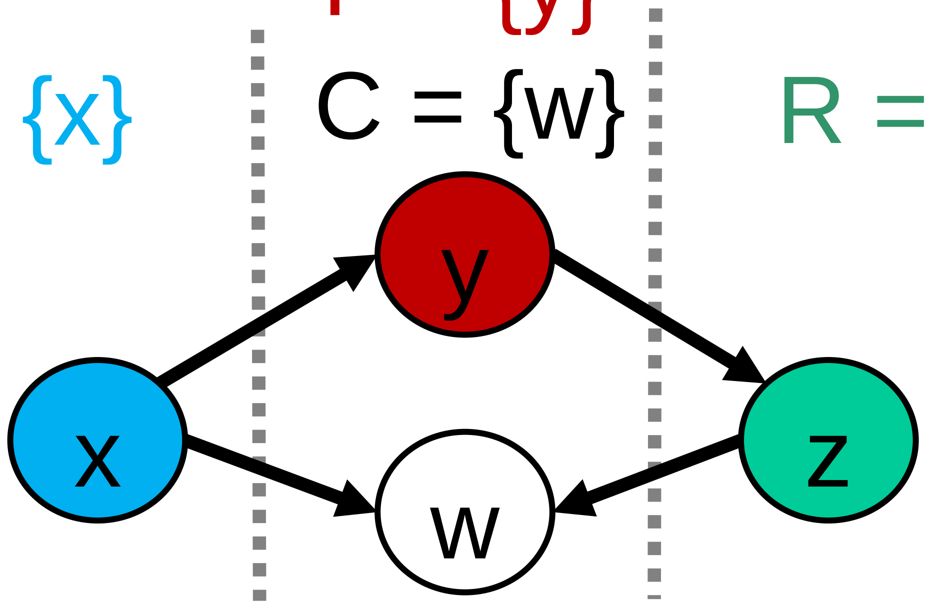
Example



Example

$F = \{y\}$

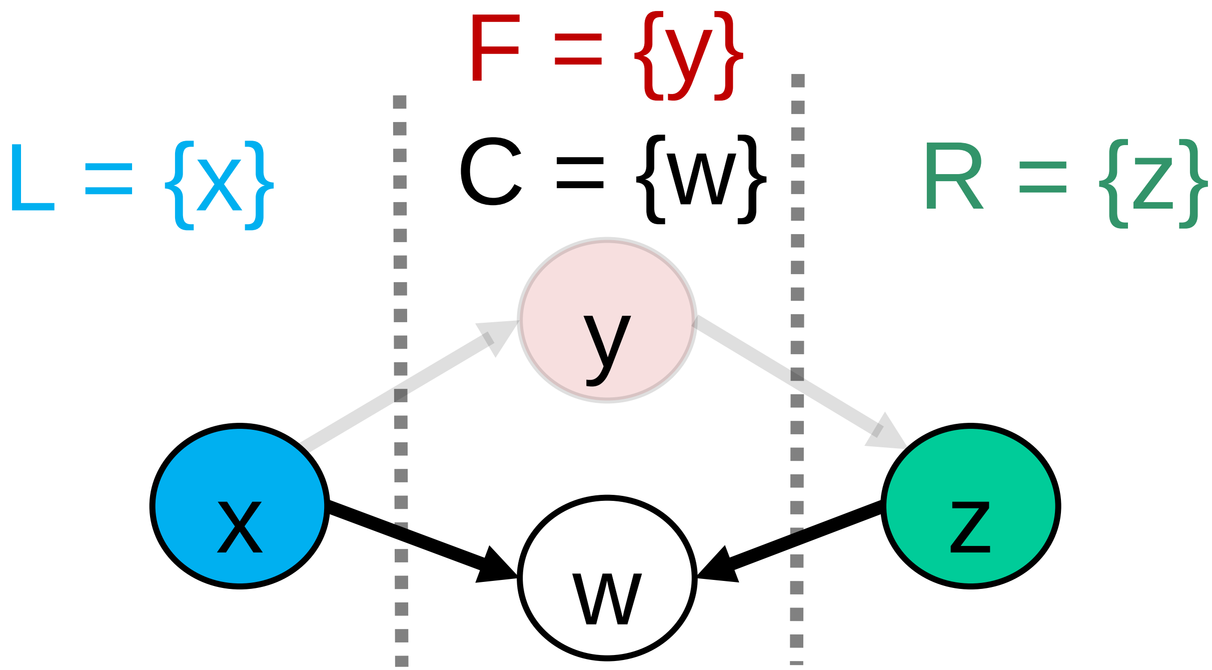
$L = \{x\}$ $C = \{w\}$ $R = \{z\}$



$$L \cup C \;\overset{1}{\not\Rightarrow}\; R$$

$$R \cup C \;\overset{1}{\not\Rightarrow}\; L$$

Example



$$L \cup C \overset{1}{\not\Rightarrow} R$$

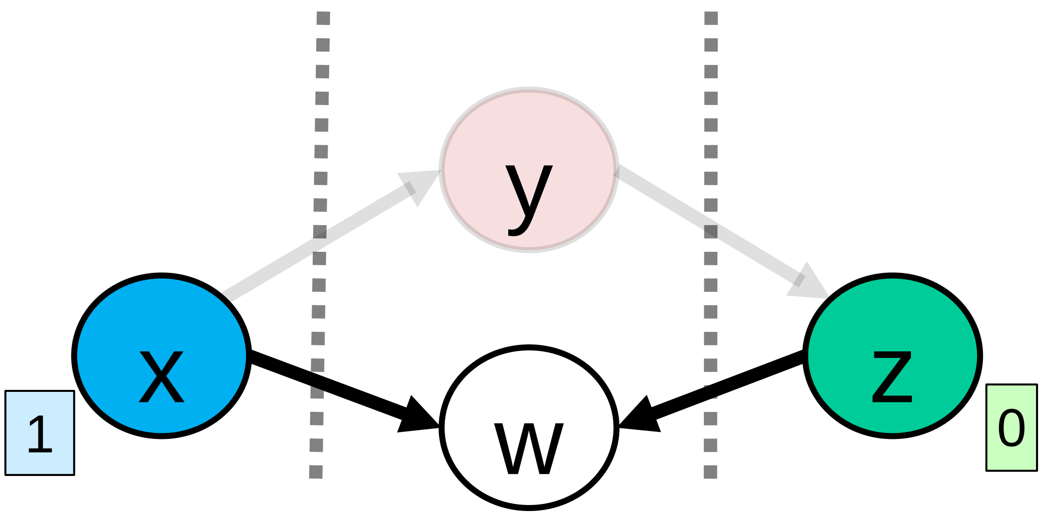
$$R \cup C \overset{1}{\not\Rightarrow} L$$

Cannot tolerate
1 crash fault

Necessity: Intuition

If condition is not true,

- all removed nodes crash
- two groups of nodes cannot communicate with each other

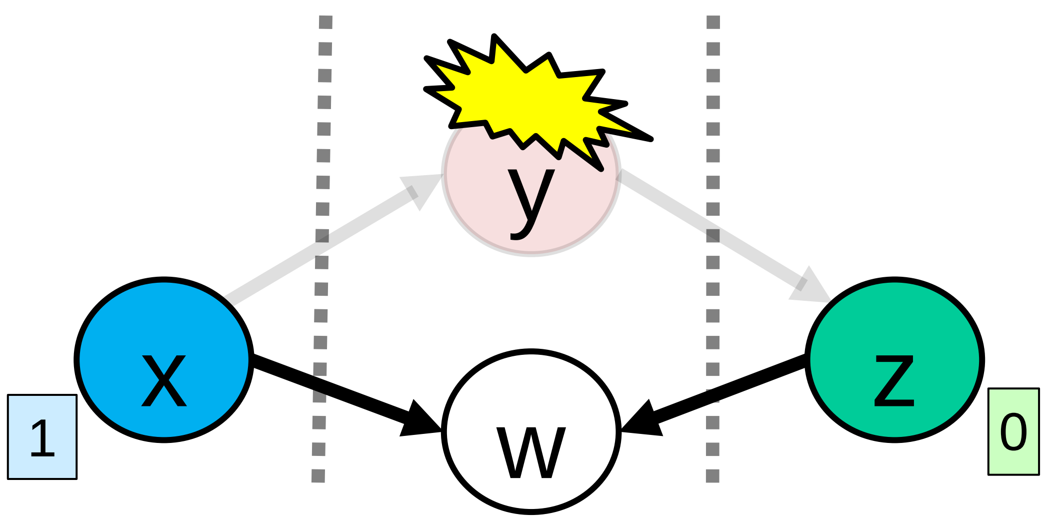


f=1

Necessity: Intuition

If condition is not true,

- all removed nodes crash
- two groups of nodes cannot communicate with each other



f=1

Equivalent Condition

Removing up to f nodes,

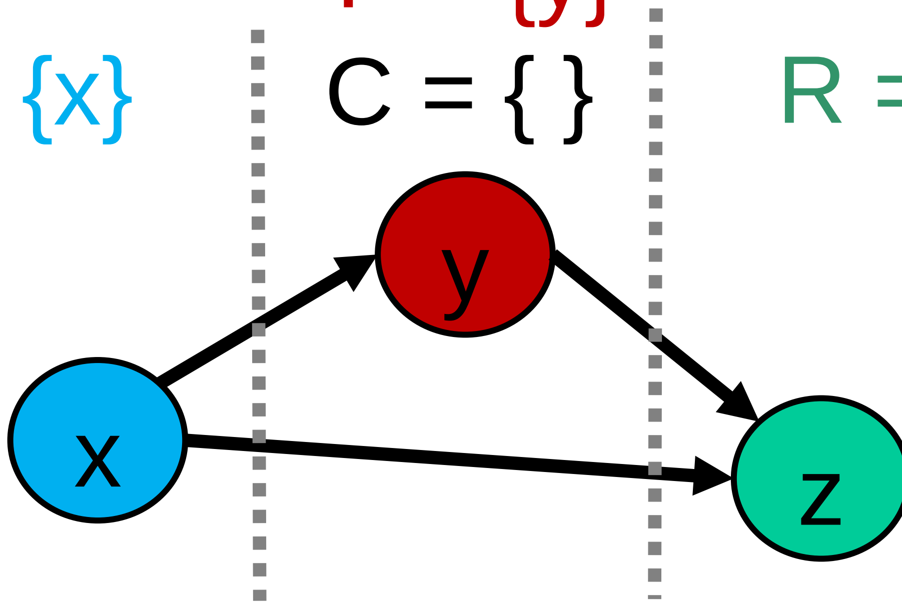
the remaining graph contains a

directed rooted spanning tree

Example

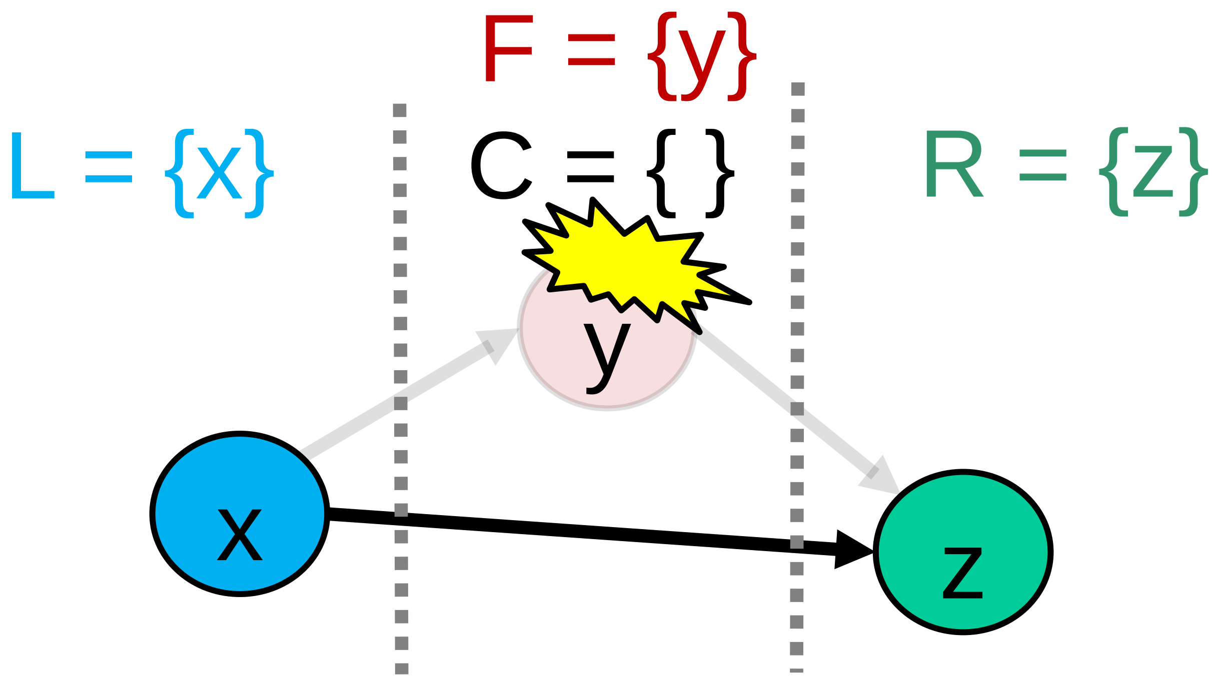
F = {y}

L = {x}   C = { }   R = {z}



$$L \cup C \overset{1}{\Longrightarrow} R$$

Example

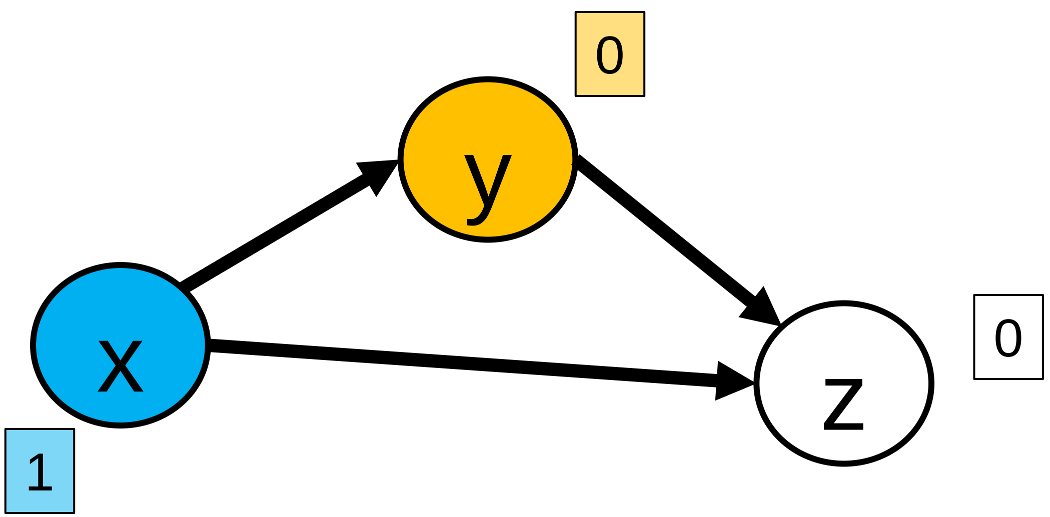


$$L \cup C \overset{1}{\Longrightarrow} R$$

Can tolerate 1 crash fault

Sufficiency: Intuition

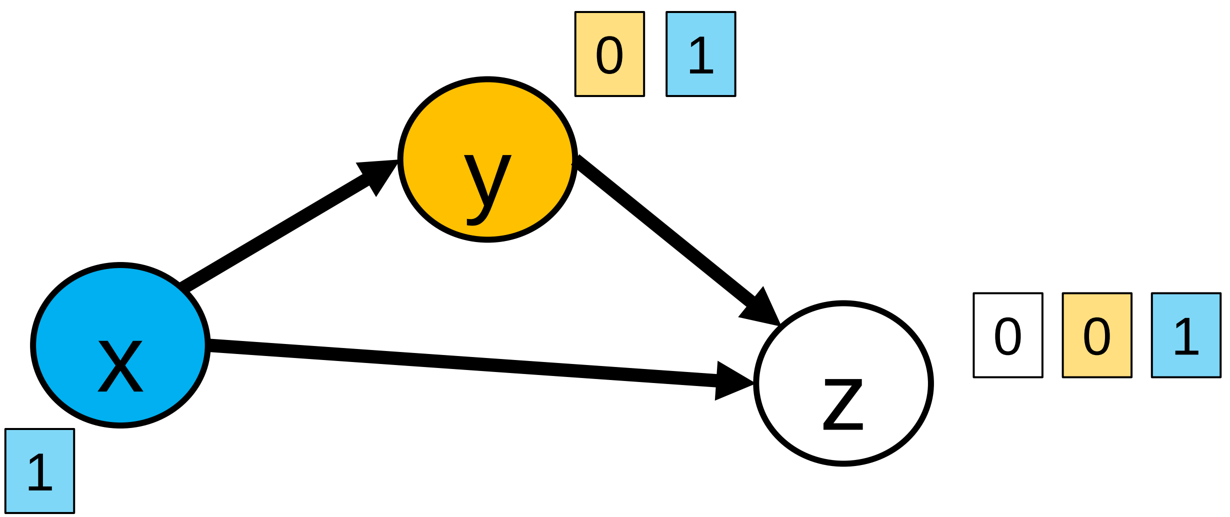
Source(s) can propagate its state



f=1

Sufficiency: Intuition

Source(s) can propagate its state



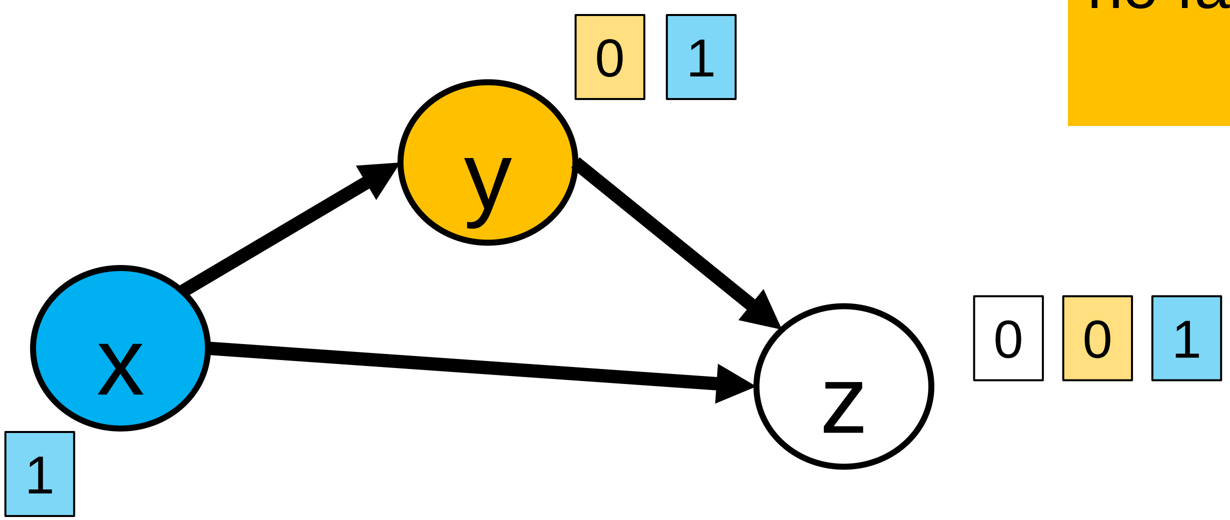
f=1

Sufficiency: Intuition

Source(s) can propagate its state

- Which source is fault-free?

no failure: pick 1



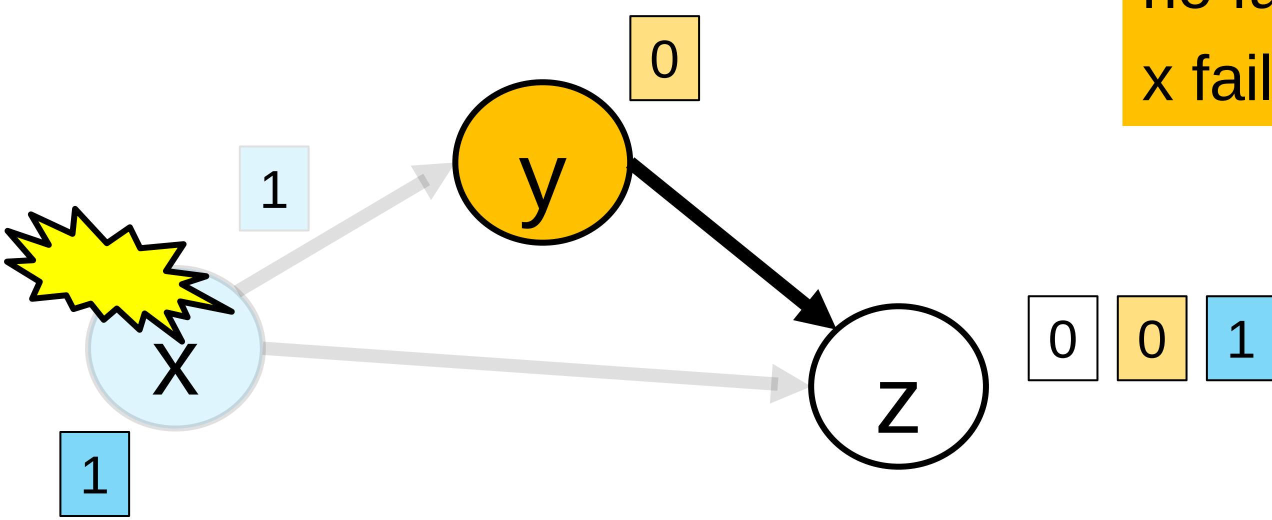
f=1

Sufficiency: Intuition

Source(s) can propagate its state

- Which source is fault-free?

no failure: pick 1
x fails:

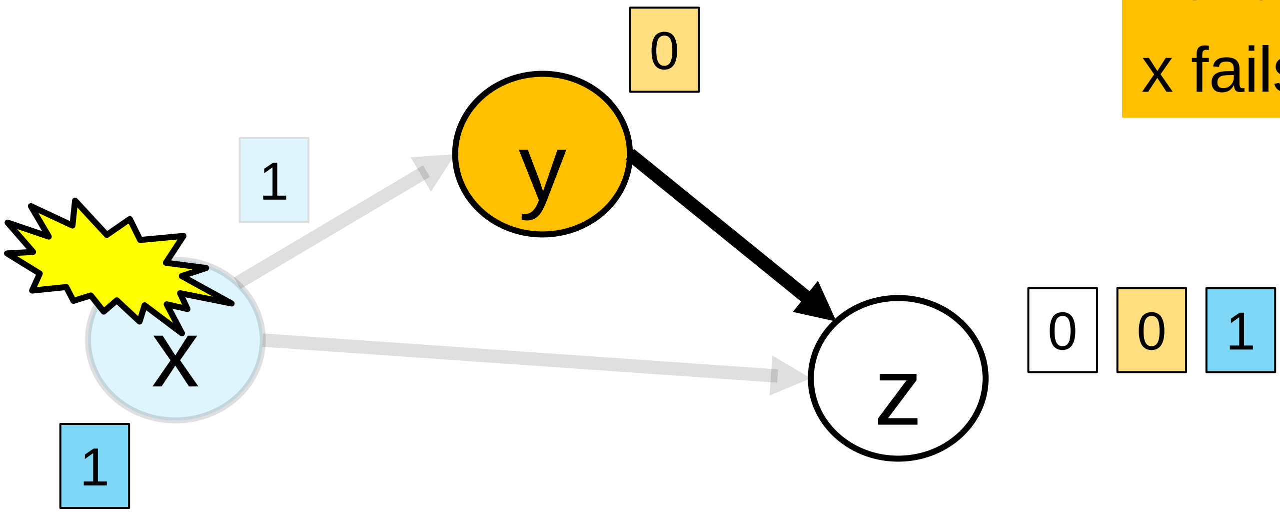


f=1

Sufficiency: Intuition

Source(s) can propagate its state

- Which source is fault-free?



no failure: pick 1
x fails: pick 0

f=1

Algorithm Min-Max

binary consensus

v_i = input

Phase p = 1 to 2f+2

Flood v_i

Receive set of values R_i

if p is even

$v_i = \min(R_i)$ (Min Phase)

else

$v_i = \max(R_i)$ (Max Phase)

Output v_i after 2f+2 phases

Algorithm Min-Max

binary consensus

v_i = input

Phase p = 1 to 2f+2

Flood v_i

Receive set of values R_i

if p is even

v_i = min(R_i) (Min Phase)

else

v_i = max(R_i) (Max Phase)

Output v_i after 2f+2 phases

Algorithm Min-Max

v_i = input

Phase p = 1 to 2f+2

Flood v_i

Receive set of values R_i

if p is even

v_i = min(R_i) (Min Phase)

else

v_i = max(R_i) (Max Phase)

Output v_i after 2f+2 phases

Algorithm Min-Max

v_i = input

Phase p = 1 to 2f+2

Flood v_i

Receive set of values R_i

if p is even

$v_i = \min(R_i)$ (Min Phase)

else

$v_i = \max(R_i)$ (Max Phase)

Output v_i after 2f+2 phases

Algorithm Min-Max

v_i = input

Phase p = 1 to 2f+2

Flood v_i

Receive set of values R_i

if p is even

v_i = min(R_i) (Min Phase)

else

v_i = max(R_i) (Max Phase)

Output v_i after 2f+2 phases

Algorithm Min-Max

v_i = input

Phase p = 1 to 2f+2

Flood v_i

Receive set of values R_i

if p is even

$v_i = \min(R_i)$ (Min Phase)

else

$v_i = \max(R_i)$ (Max Phase)

Output v_i after 2f+2 phases

Algorithm Min-Max

v_i = input

Phase p = 1 to 2f+2

two consecutive fault-free phases

Flood v_i

Receive set of values R_i

if p is even

$v_i = \min(R_i)$ (Min Phase)

else

$v_i = \max(R_i)$ (Max Phase)

Output v_i after 2f+2 phases

Sufficiency: Correctness

Two consecutive <u>fault-free</u> phases p and p'

Suppose p = min phase and

p' = max phase

If any <u>source</u> in phase p has 0, then done

Otherwise, the source(s) can propagate 1 in phase p'

Sufficiency: Correctness

Two consecutive fault-free phases p and p'

Suppose p = min phase and

p' = max phase

If any source in phase p has 0, then done

Otherwise, the source(s) can propagate 1 in phase p'

Necessary condition:

there exists a directed rooted spanning tree

Byzantine Failures + Synchrony

Exact Byzantine Consensus

Each node has a binary input

- Agreement: Good nodes agree on an *exact* value
- Validity: Agreed value is an input at some *good* node
- Termination

Byzantine Failures + Synchrony

Exact consensus possible iff

For any node partition L, C, R, F with

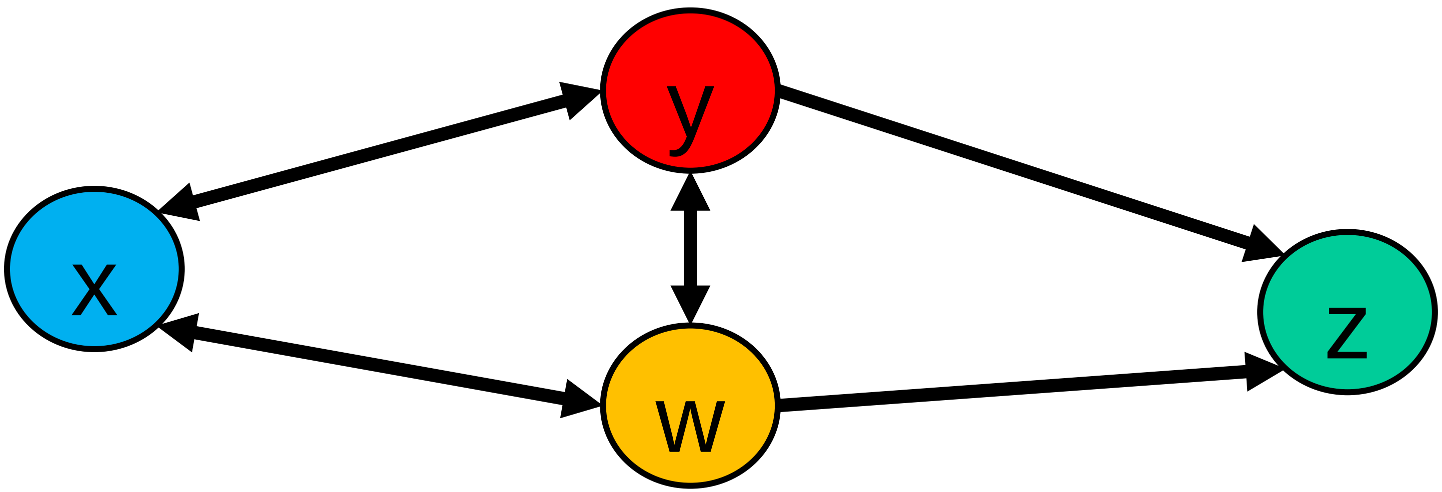
L, R non-empty, and $|F| \leq f$

$$L \cup C \xrightarrow{f+1} R$$

or

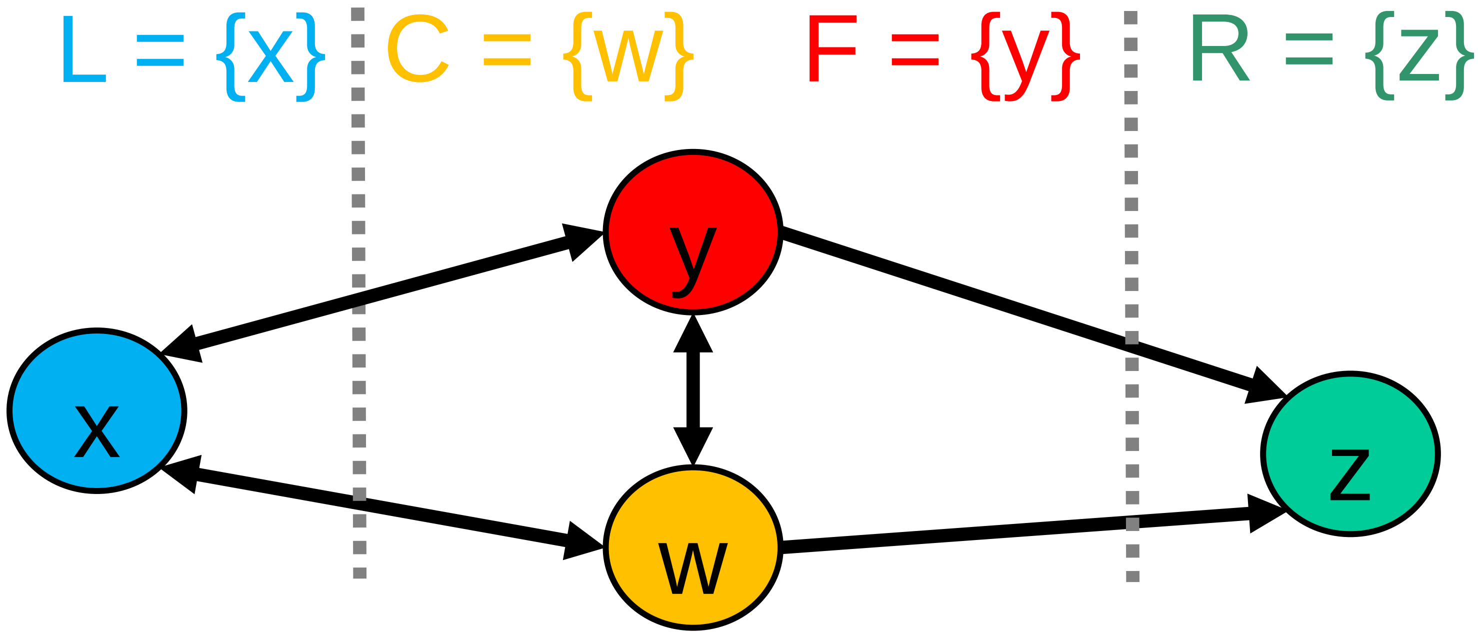
$$R \cup C \xrightarrow{f+1} L$$

Example



Tolerate 1 crash fault

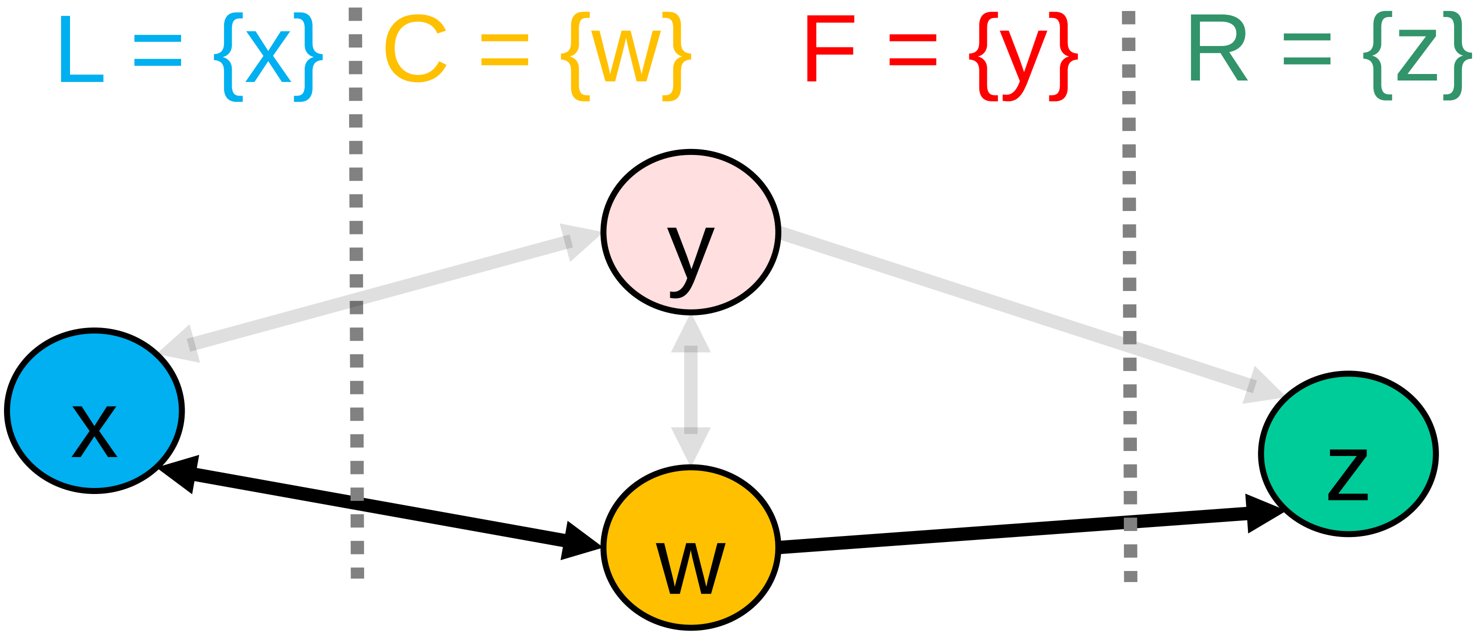
Example



$$L \cup C \overset{2}{\not\Rightarrow} R$$

$$R \cup C \overset{2}{\not\Rightarrow} L$$

Example

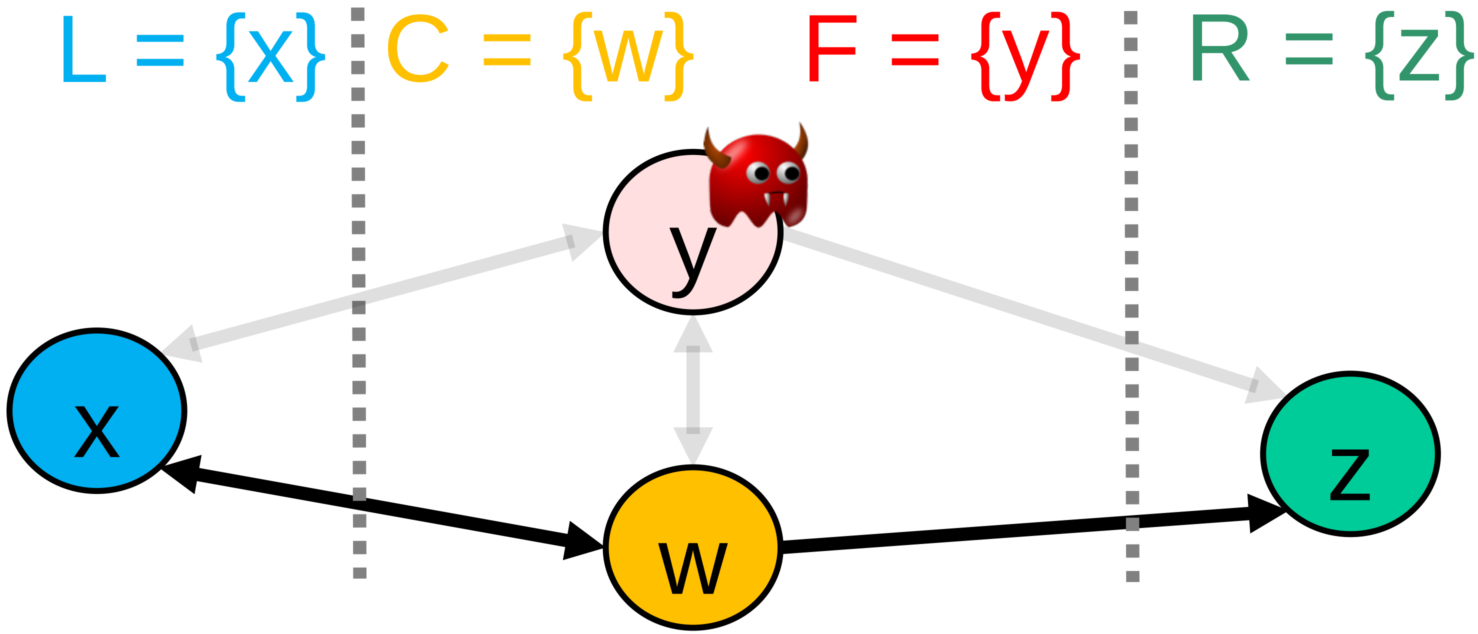


$$L \cup C \overset{2}{\not\Rightarrow} R$$

$$R \cup C \overset{2}{\not\Rightarrow} L$$

Cannot tolerate
1 Byzantine fault

Necessity: Intuition

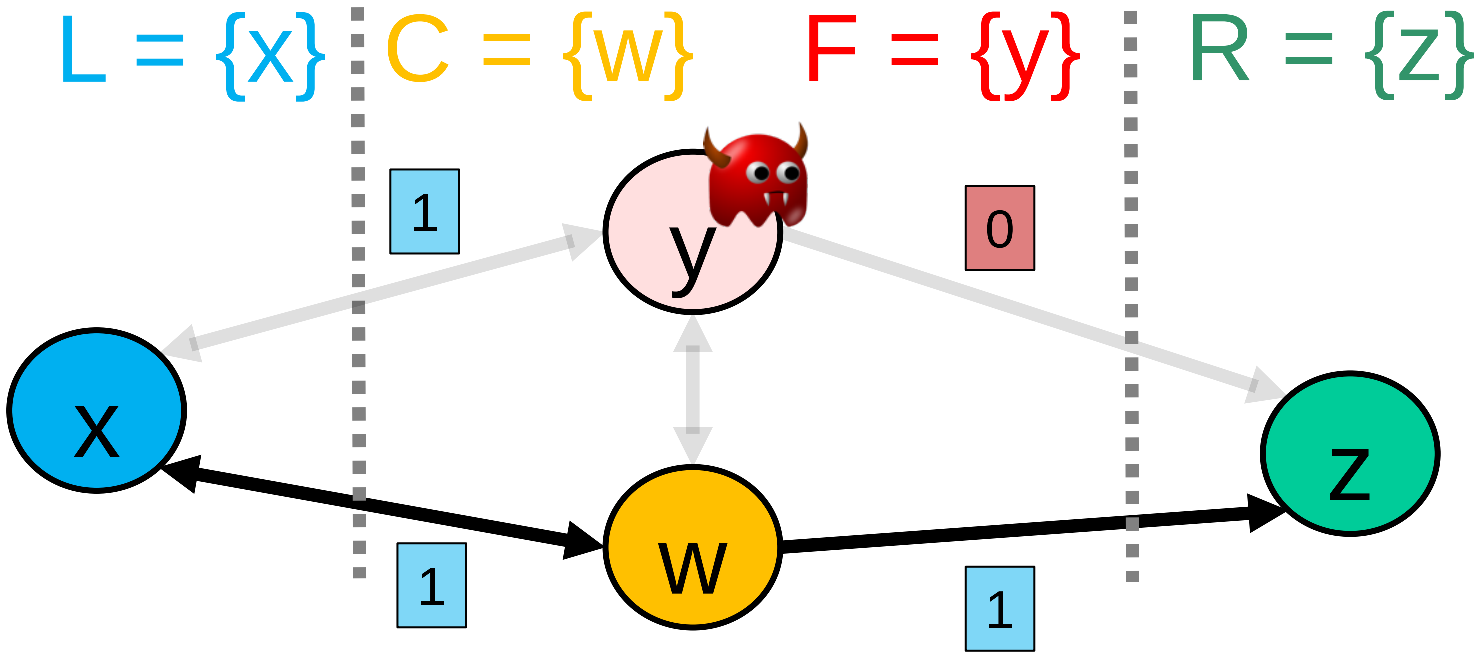


$$L \cup C \overset{2}{\not\Rightarrow} R$$

$$R \cup C \overset{2}{\not\Rightarrow} L$$

Cannot tolerate
1 Byzantine fault

Necessity: Intuition



- x cannot hear from z
- z cannot receive x's msg <u>reliably</u>

$$L \cup C \overset{2}{\nRightarrow} R$$

$$R \cup C \overset{2}{\nRightarrow} L$$

Cannot tolerate 1 Byzantine fault

Equivalent Condition

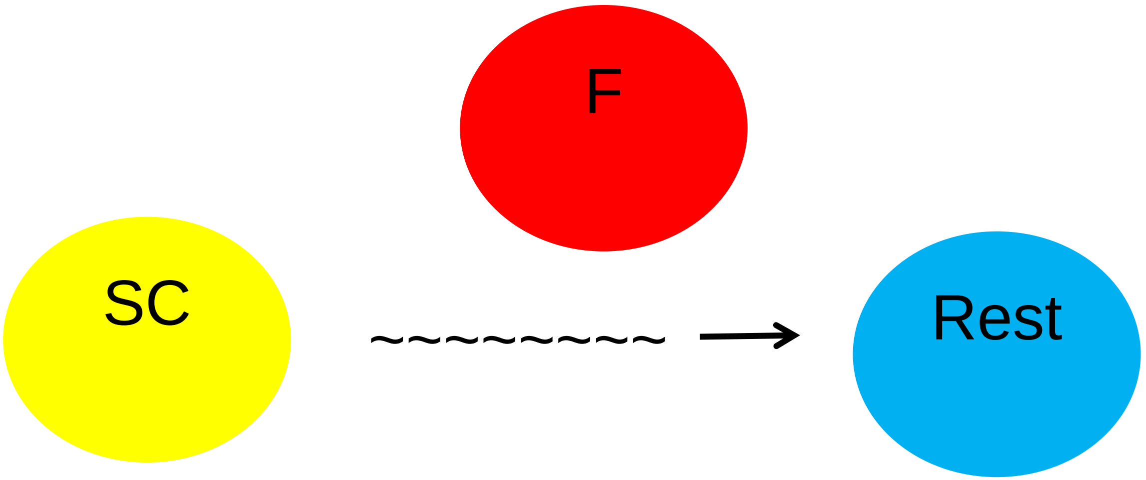
1. Remove F   ($|F| \leq f$)
2. Remove outgoing links of F1   ($|F1| \leq f$)

Then, the remaining graph contains a

directed rooted spanning tree

Key Properties

In the graph:



- strongly connected (of size > f)
- f+1 paths excluding F to the rest of the graph

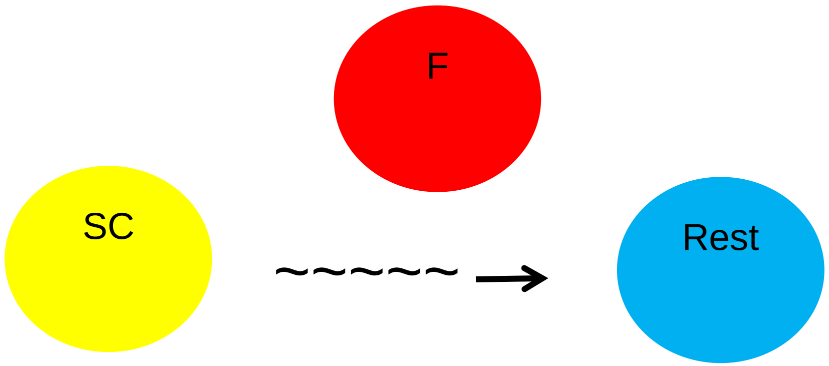
Sufficiency: Algorithm BC

OUTER Loop: enumerating over all possible F

INNER Loop: enumerating over all partitions

SC propagates values

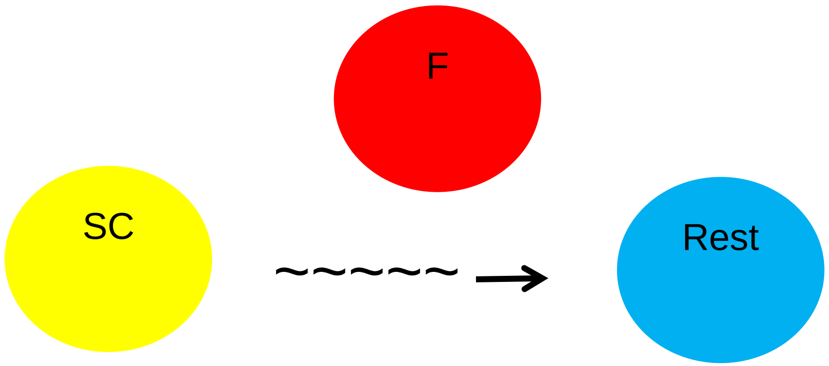
Propagation



- SC: using **f+1** paths excluding F to send values
- Rest: if same received values,

  state := value

Propagation



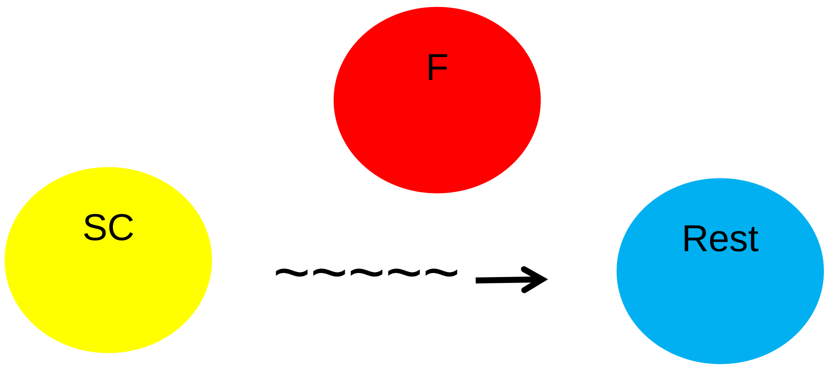
- SC: check if states are the same,

  use **f+1** paths excluding F to send state v

- Rest: if same received values,

  state := value

Propagation



States stay valid

Agreement is achieved:

- F = actual fault set
- nodes in SC have same state

- SC: check if states are the same,

  use **f+1** paths excluding F to send state v

- Rest: if same received values,

  state := value

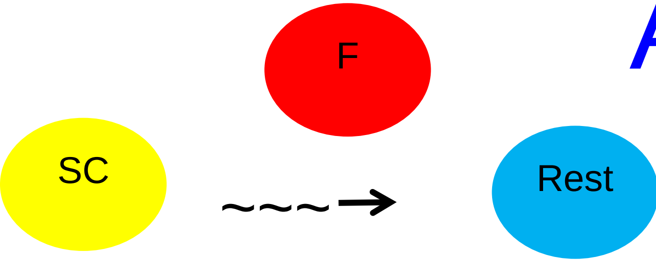
Algorithm BC

OUTER Loop: enumerating over all possible F

INNER Loop: enumerating over all partitions

SC propagates values

Algorithm BC

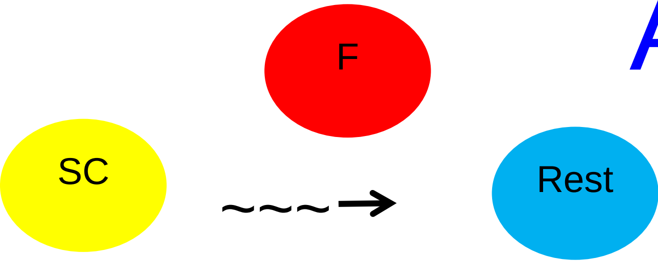


OUTER Loop: enumerating over all possible F

INNER Loop: enumerating over all partitions

SC propagates values

Algorithm BC



OUTER Loop: enumerating over all possible F

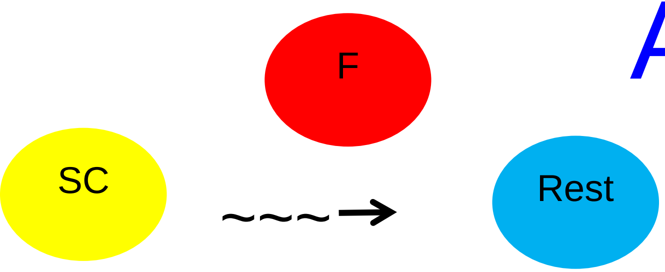
INNER Loop: enumerating over all partitions

SC propagates values

Propagation:

values stay valid

Algorithm BC



F = actual fault set

OUTER Loop: enumerating over all possible F

INNER Loop: enumerating over all partitions

SC propagates values

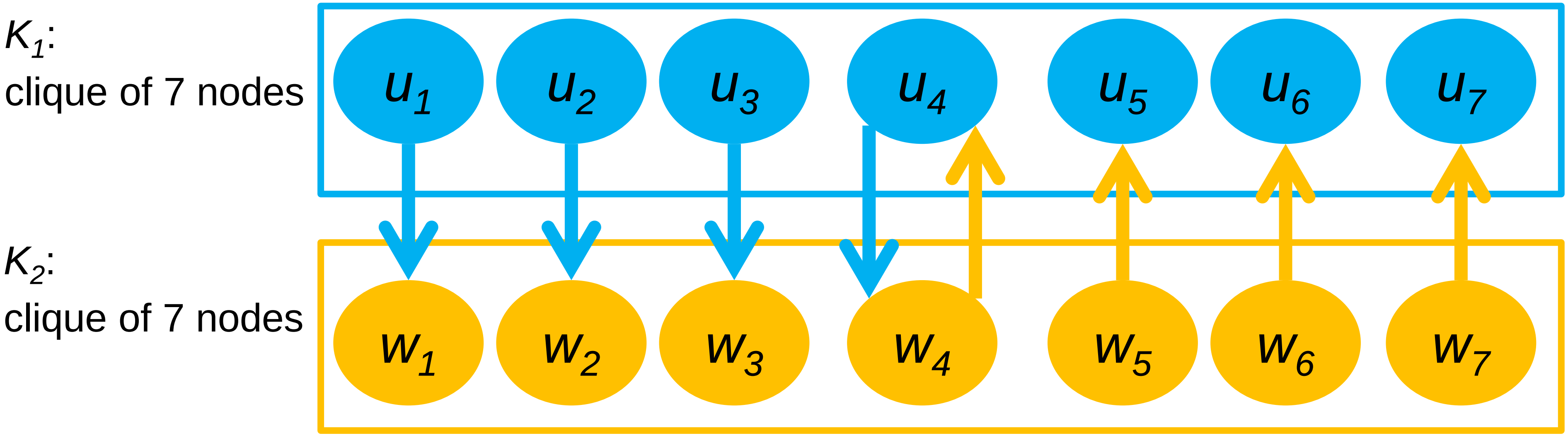
Propagation:

values stay valid

Agreement is achieved

when nodes in SC have the same value

2-clique Network

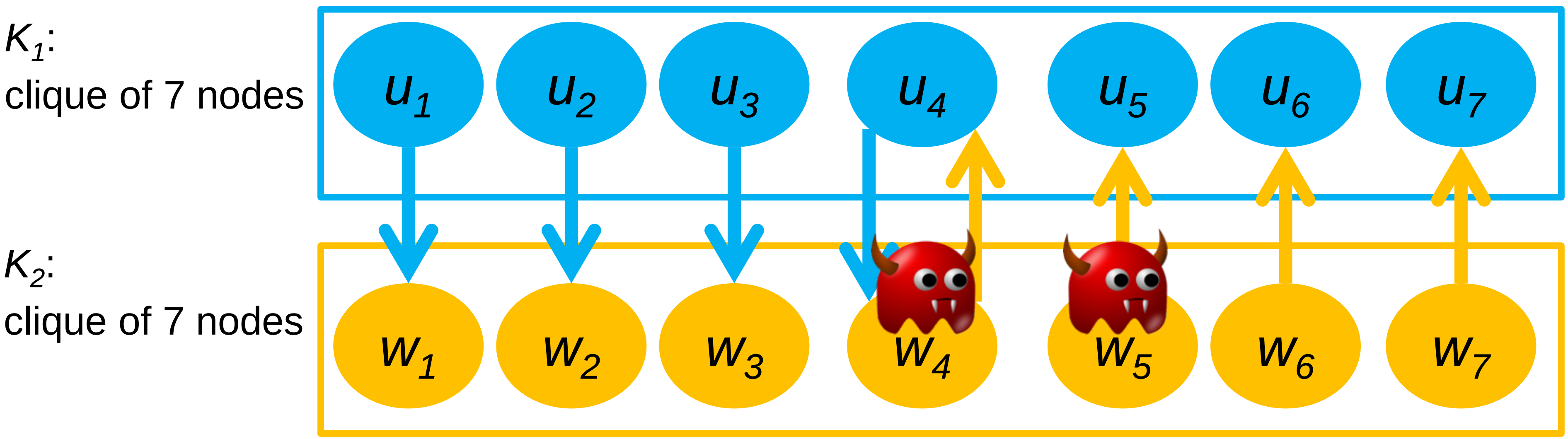


<u>4 directed links</u>
in each direction
between the cliques

Can tolerate 2
Byzantine faults

2-clique Network

K_1 and K_2 <u>cannot</u> talk reliably with each other?



<u>4 directed links</u>
in each direction
between the cliques

Can tolerate 2 Byzantine faults

Consensus

Fault Model	System/Output		Graph	Results
Crash	Synchronous	Exact	Undirected	Well-known Results
			Directed	**PODC ‘15**
		Approximate	Undirected	Decentralized Control (e.g., [Tsitsiklis ‘84] [Jadbabaei ‘03])
			Directed	
	Asynchronous		Undirected	
			Directed	**PODC ‘15**
Byzantine	Synchronous	Exact	Undirected	[PSL ‘80] [FLM ‘85]
			Directed	**PODC ‘15**
		Approximate	Undirected	[Dolev ‘83] [FLM ‘85]
			Directed	**PODC ‘12**
	Asynchronous		Undirected	[Dolev ‘83] [FLM ‘85]
			Directed	**Open**

Our Other Work

Byzantine + Synchronous + Approximate + Iterative:

Fault Model	Results
up to f failures	[Vaidya, Tseng, Liang, PODC ‘12]
Generalized	[Tseng, Vaidya, ICDCN ‘13]
Link failures	[Tseng, Vaidya, NETYS ‘14]
Mobile faults	[Tseng, SSS ‘17]

Byzantine Broadcast:

- [Tseng, Vaidya, Bhandari, IPL ’16], [Tseng NCA ‘17]

Convex Hull Consensus:

- [Tseng, Vaidya, PODC ‘14]

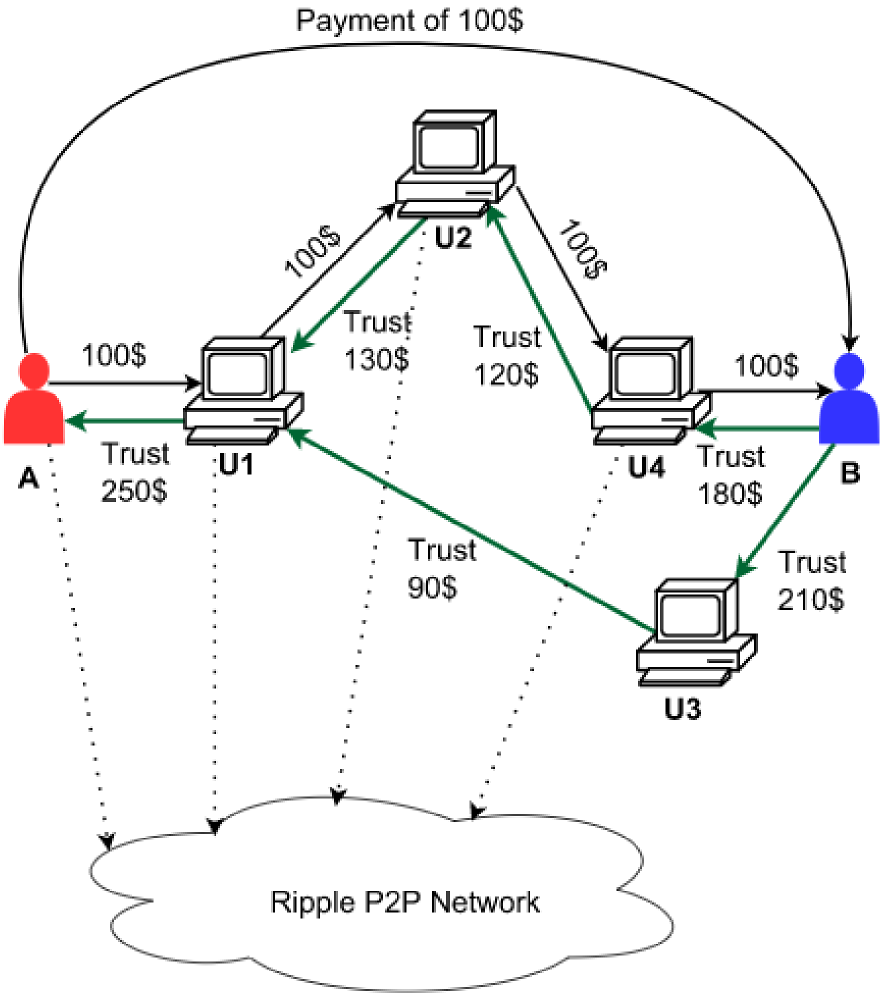
Open Problems

- Graph property for Asynchrony + Byzantine
- More efficient algorithms
- Lower bound on time complexity
- Given G, find the maximum number of faults that can be tolerated

Open Problems

- Other types of consensus
  - k-consensus
  - different fault models
  - different validity conditions
- Other types of networks
  - time-varying network
  - Different network interpretation, e.g., network of trusts

Network of Trusts



From Ripple Consensus White Paper [2014]

Thanks!