# Multiparty Communications
# CS 118
# Computer Network Fundamentals

Peter Reiher

# Outline

- Extending 2-party model to N-party

  instead of having just sender and communicator, we look at models where we have N parties

- A party has multiple receivers (other end)
- A party has multiple senders (local end)
- Multiples of information

# Shannon Channel

- Two preselected parties
  - Homogenous endpoints

- Unidirectional channel
  - Preselected sender, preselected receiver
- ***One predetermined sender, one predetermined receiver***

# Shannon 2-party communication

- We began by knowing:
  - Participating endpoints
  - Communication channel
- We didn't know, but fixed:
  - *When* the endpoints share state
    - So we need a handshake
    - Including "when they want to be active" vs. idle
  - Whether something is *lost* loss of information
    - So we need timers

# Decoupling party from channel

general communication: other kinds of parties

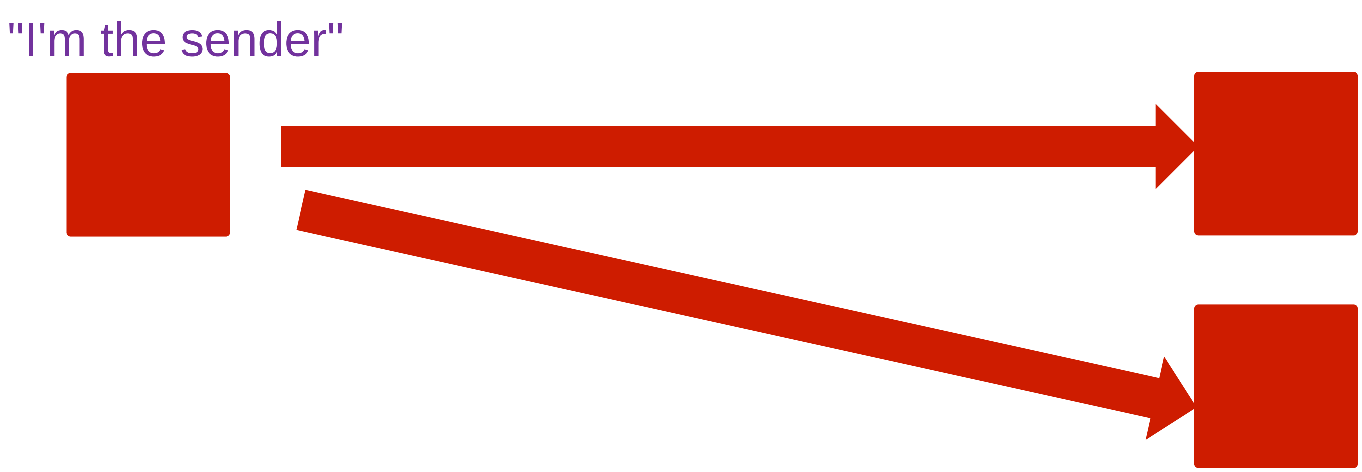
- What if we want to talk to different parties?
  - Sometimes we communicate with Twitter
  - Sometimes we communicate with eBay
  - Sometimes we communicate with Wikipedia
- Don’t want a permanent, always-on channel to each of them
- How can we do better?
  - “Detach” channel end from party

# Channel vs. party

- Shannon channel theoretical concept: we can't change who we are communicating with
  - Integrated with the endpoint (party)
  - No choices – all information sent/received uses the only channel there is

# Separating the two

- Need to treat what happens in the endpoint (state to share) from the channel (because there might be more than one)



# Abstract network components

- Endpoint source of information
  - (“party”)
  - Source or sink of state (“information”)

- Link move info from one place to another
  - (“channel”)
  - Action at a distance (“symbol transfer”)

# Components

**Shannon**

- Party
- Channel
- Information
- 2-party interaction

**Multiparty, modern terms**

- Endpoint, node, host
- Link, hop
- State, data
- N-party interaction

# Multiparty extensions


- Which party you're talking to
  - Need to differentiate the receivers
  - Names

- How talk to multiple parties at once
  - Juggling multiple "senders"
  - Sockets learn socket.io

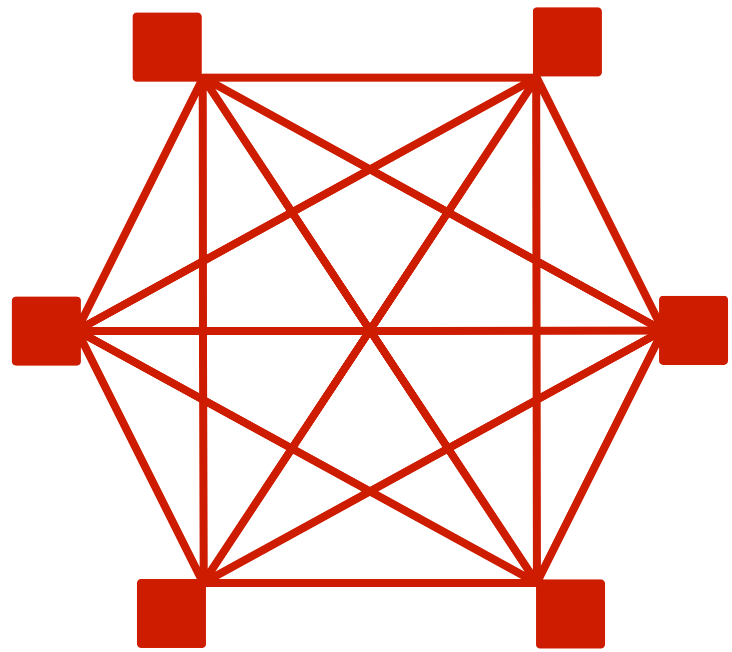
    operating system entity - CS 111 connects to CS 118
    each processes can specify which party to comm. with

- How to say the same thing multiple times
  - Broadcast and multicast

    at system level, we need some kind of abstraction
    multicast - send to limited people
    broadcast - same thing; we want to allow everyone in the same network to hear what the one person is saying

# Multiparty

- Multiple endpoints
  - All connected
  - By separate 2-party channels
  - Using a single protocol



# Multiparty assumptions

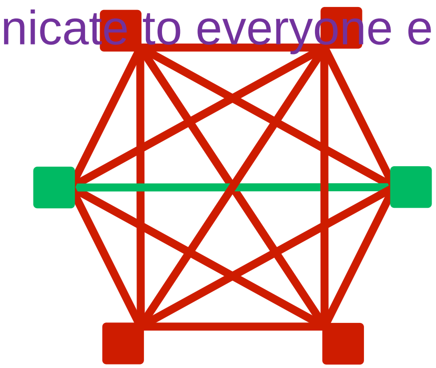
- Multiple parties
- Using ONE common protocol
- Connected by direct 2-party channels
  – I.e., fully-connected topology
  – Each channel disjoint from the others
    - In state
    - In inputs and outputs

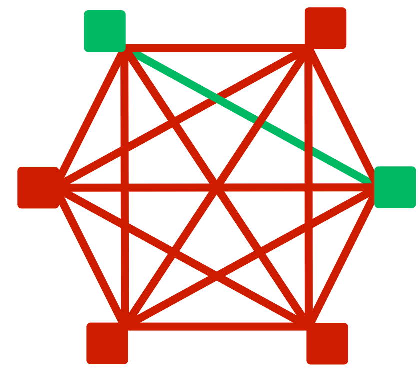
# Why is this *networking*?

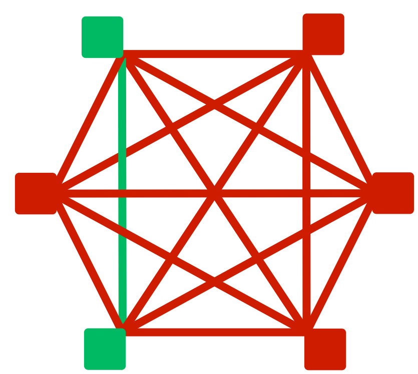
no longer simple shannon connection
anyone of the N parties can communicate to everyone else!

- Networking
  - Methods to enable communication between ***varying sets*** of indirectly connected parties that don't share a single protocol
- A small increment
  - ONE protocol for now
  - Direct 2-party channels for now
  - (we'll get to the other parts later…)

we are only doing a 2-party channel







# Importance of multiparty

- Varying participants
  - Pairs communicating change

    we get a lot more complexity in terms of deciding where a message is going to go;
- Varying view of state
  - Subsets of state, potential overlap, etc.

    does this mean I have one large state, or does it mean I have N different sub-states
- More power
  - Can share with more than one other party

# The need for names

direct channel for each N-1 receivers

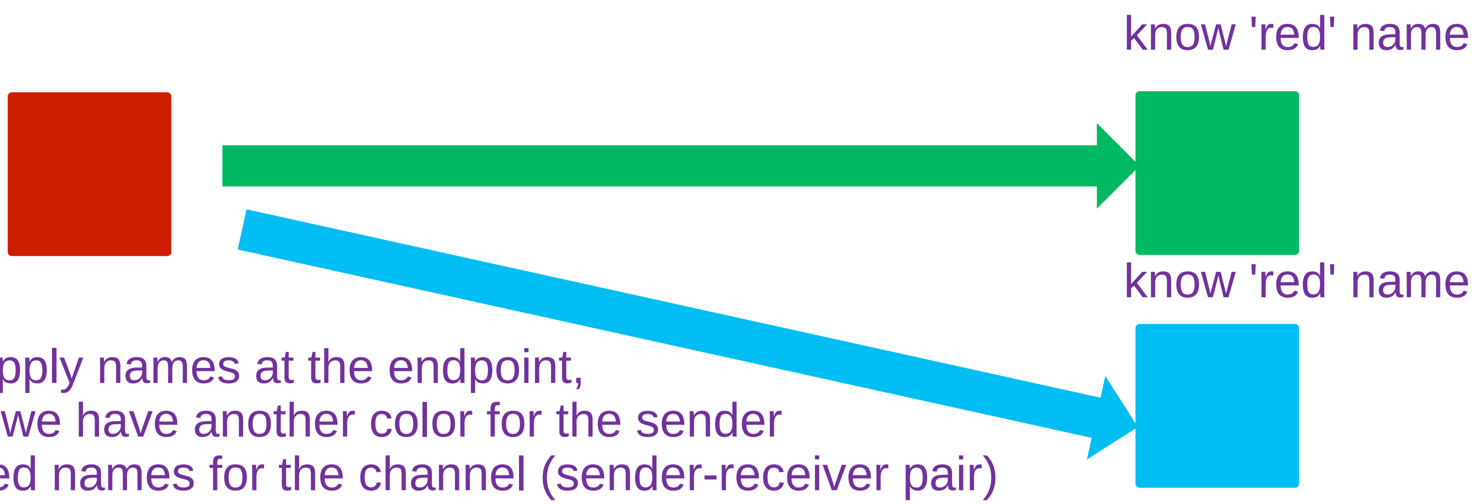
- Each source can interact with N-1 receivers
  - How are receivers differentiated?
    - Each uses a different channel
    - But how do we specify which channel is which?


*Need some sort of identifier to indicate which channel (indicating which receiver)*

# A simple case

identify which of the two receivers?

- One sender
- How do we identify one of the two possible receivers?

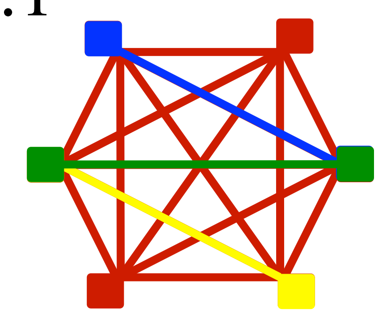



if we apply names at the endpoint,
then if we have another color for the sender
we need names for the channel (sender-receiver pair)
the sender iteself or the receiver, are general

# What can the name apply to?

- Identifier can mean several things at once:
  - Channels
  - Endpoints


- WHY?
  - Consider a fully-connected network
  - For each source, channel:endpoint is 1:1



# Names for receivers

- *Index*
  – A number that corresponds to the channel/endpoint
- *Port*
  – An OS-centric type of name specifying what the OS should connect the channel to

something that you send to on another computer
send to port '25', the other guy knows what you are talking about....

- *Channel*
  – Used more generically
- *Socket*

a channel-like thing, a big data structure from OS

  – Originally (1974 TCP) meant one end of 2-party
  – Unix/BSD copied the term (1983)
  – Now means a LOT more

sockets known by sending and receiving OS or process

    - Large data structure with many parts
    - A "socket descriptor", i.e., a pointer to that structure

process that owns the socket can refer to the socket and say I want to send message on this socket

# Receiver naming requirements

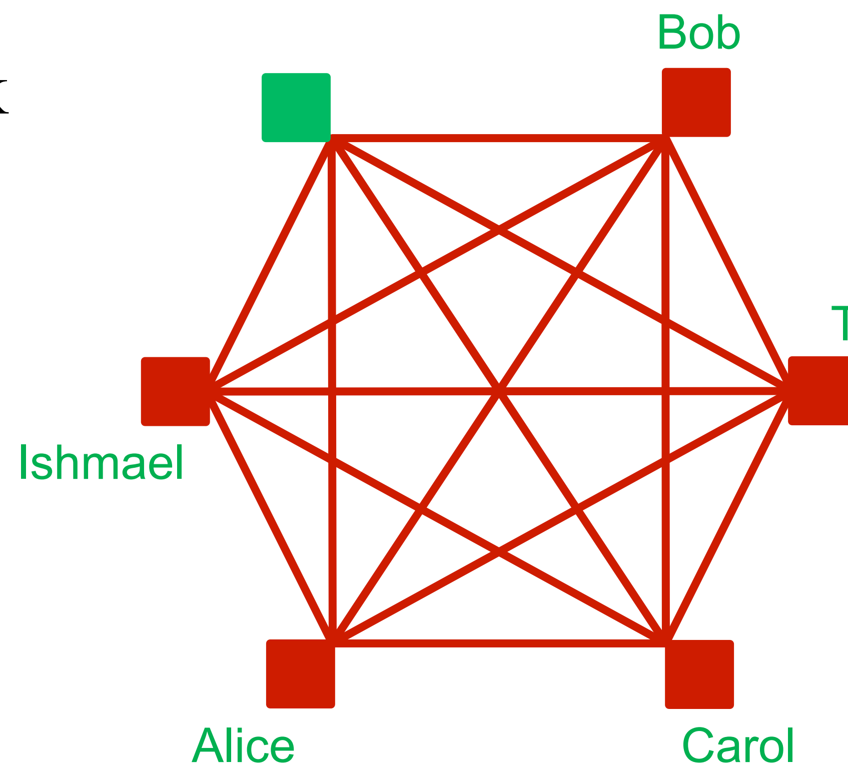
- How unique? direct communication - one sender does not care about the socket number for another socket
  - Each party needs to differentiate N-1 receivers

    need a bunch of names that are unique for each individual sender
  - Names need to be unique within that set

    ronly care about yourself and how you call other people
  - NO need (yet) for names to be unique within the set of all parties
    - You can call me Ray,
      or you can call me J,
      or you can call be Ray J,
      or you can call me RJ, …

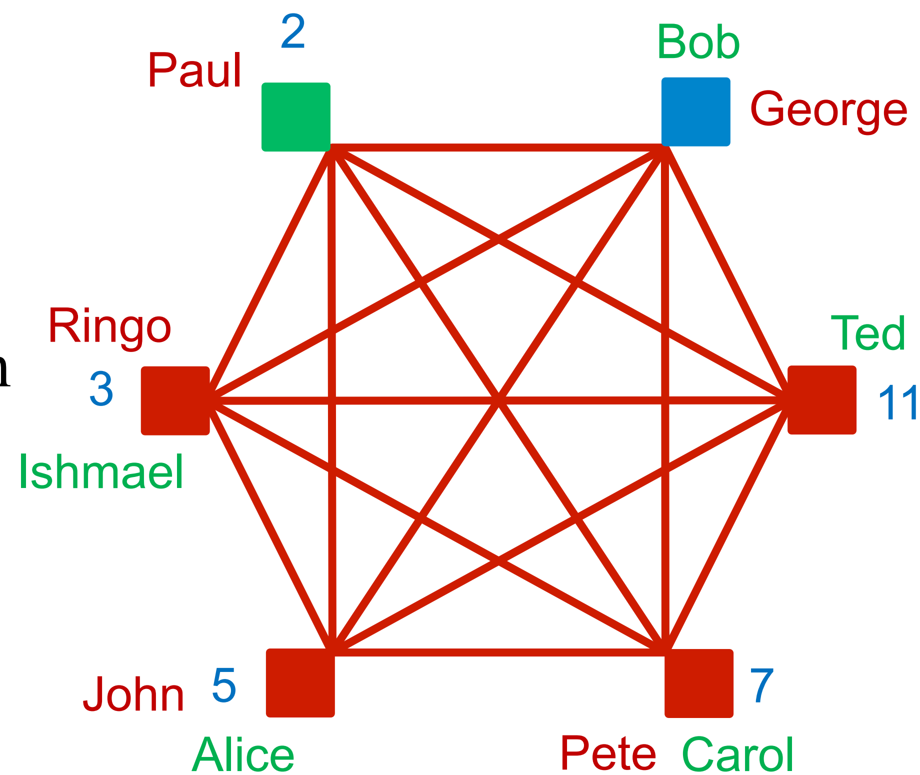
# Receiver name examples

- One sender can name the other ends it can talk to



# Receiver name examples

- Another sender can do the same thing
  - But possibly with different names
  - Its names need not match anyone else's
- Names are local
  - To the sender and receiver



# Multiple senders

- A party can have multiple senders (local end)
- Like my computer talking to multiple web sites

# Concurrency

- How does a party deal with multiple communications?
  - The channels – need to “keep ‘em separated”
  - Need to decouple the channel from the party itself

channels need to be kept separate

separate state machine for every channel

- Socket
  - A “disembodied” communication endpoint within a party

socket describes an endpoint within a party

# What's inside the party?

- Originate/terminate communication
  - State to be shared
- Where's that state?
  - Part of finite state machine (a process) within the party
  - Outside the party
    - We can treat this as output/input of a FSM that relays that info to the channel

deal with input and output

# How many machines are there?

FSM can be complicated if there are many different states

- Strictly, one
  - Multiple FSMs can be modeled as one FSM
- Simpler to think of them as independent
  - A set of FSMs, running concurrently
    - Multiprocessing
  - And/or running ***as if*** concurrent with each other
    - Multiprogramming
  - And/or having ***internal*** concurrent components
    - Multitasking / multithreading

# So what else do we have to name?

- On the machine (or state)
  - Process/thread identifier
  - State identifiers

port is one way to tie something to a process

- Why?
  - Need to know which portion of the party's state interacts with a given channel

else, it could interfere with other FSM

networking and OS connect, they are not disjoint

# Internal naming requirements

- How unique?
  - Each party needs to differentiate some number of "FSMs" (sets of states)
  - Names need to be unique within that set
  - NO need for names to be unique within the set of all parties
    - Will there ever be such a need?
    - State is always local to the endpoint

unique within one machine - does not need to be unique across all machines

# Summary of multiparty naming

- Need a way to pick an outgoing channel/ receiver
  - An internal channel index
- A way to pick a subset of internal state/ machine
  - An internal machine index

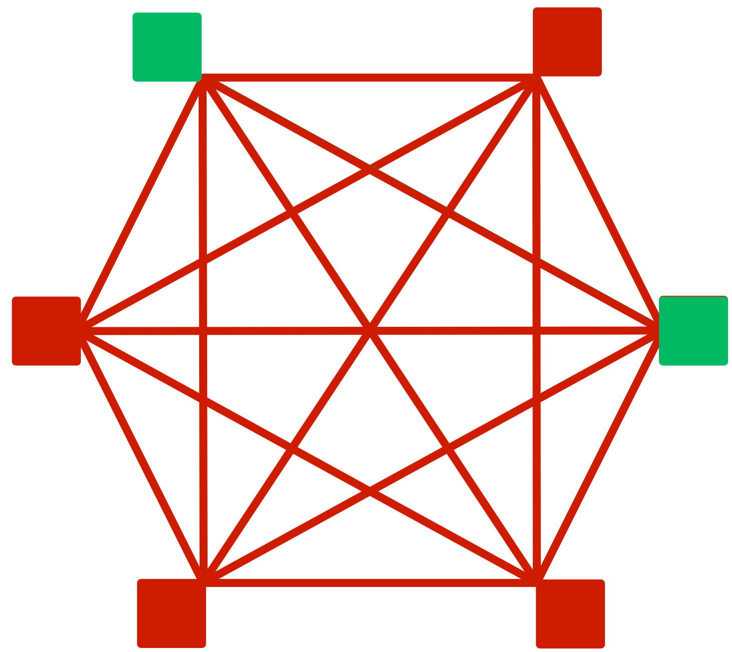
*BOTH ARE INTERNAL ONLY*

# Multiples of communications

- Each party usually wants to communicate to multiple other parties
- Sometimes 1-to-1
- Sometimes same info to many others

# Shannon channel

- Unicast
  - 1:1
- Two parties share state
  - Pick which two
  - Just communicate
- State now shared!



sender and receiver are the only ones involved

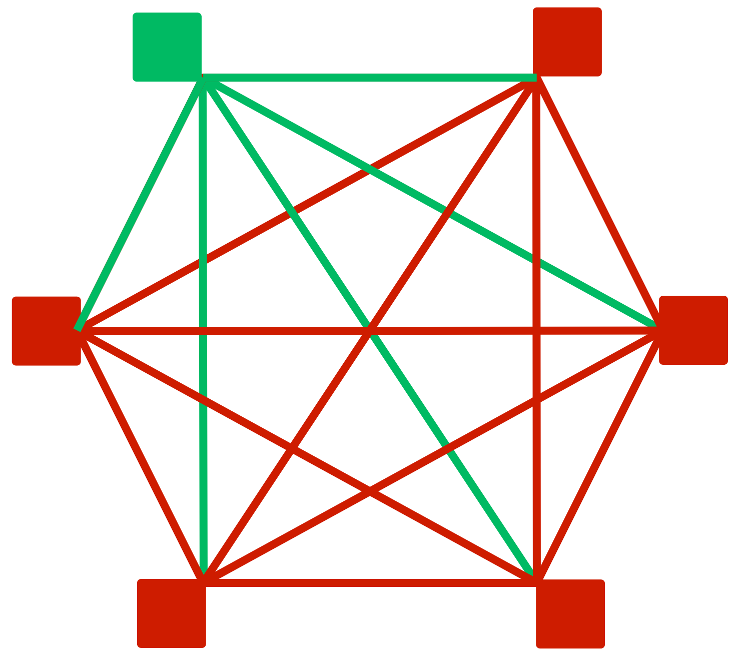
# Multiple receivers

**make sure to know difference between the two here**

- Broadcast (1:N)
  - Send same info. on all channels
  - Every party in the network has the same info.
- Multicast (1:M)
  - Broadcast on a subset of channels

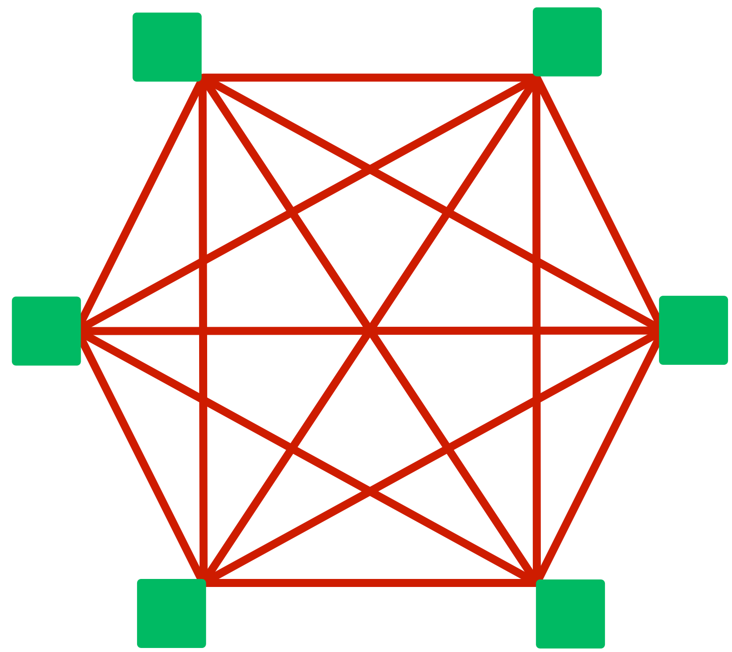
# Broadcast

- Share state everywhere
  - No need to pick
  - Need to replicate
    - Multiple communication
    - Multiple information



# Broadcast

- State now shared
  - When?
    Need to coordinate
  - How to coordinate?
    - Three-way handshake
    - Chang's "Echo alg."



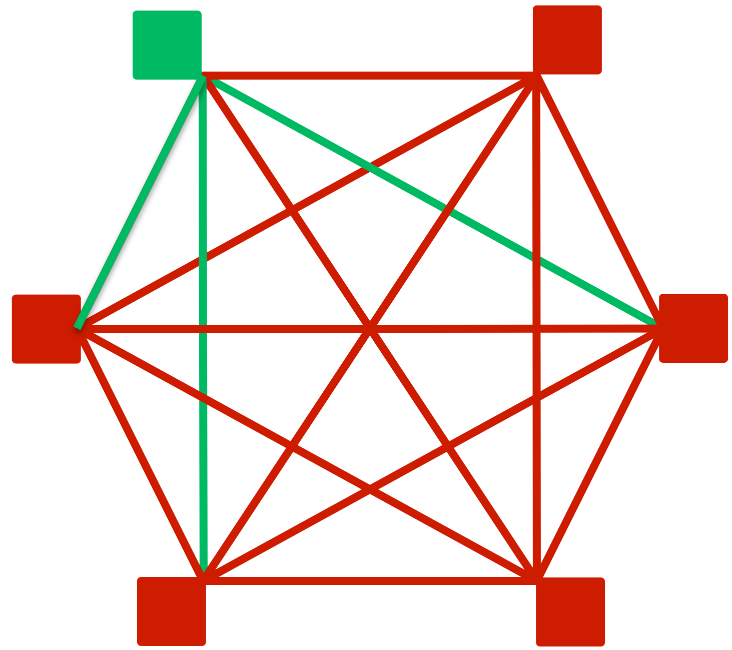
# Complexities of communications copying

- Atomicity  distributed problem
  – Losses don't correlate across channels
  – Might link "all-or-none" behavior
- Synchrony
  – Knowing all the receivers have the info at the same time
  – Having them know that
  – Having you know that
- Efficiency
  – Send one to each receiver? Can we do better?

# Multicast

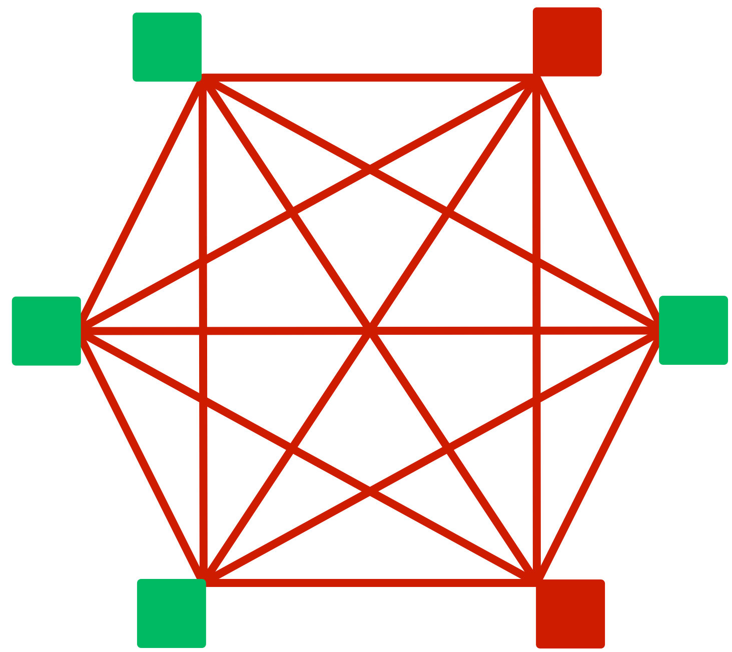
does sender pick or does receiver pick?

- Share with a subset
  - How to pick?
  - Who picks?
- Similar to broadcast
  - Need to replicate
  - Need to coordinate



# Multicast

- Things get worse…
  - Subset can change
    - Add parties
    - Remove parties



# Multicast complexities

- Group selection
  - How do you indicate the subset desired?
  - Who picks? Sender or receivers?
- Changes in group
  - Members join
  - Members leave

# Full pairwise connectivity

- One topology
  - Full, 1-hop connectivity
  - Simple to understand
- Expensive to maintain and use
- Hard to add new members

fully connected graph; every node has 1-hop connectivity
directly connected to everyone else

# Problems with this picture

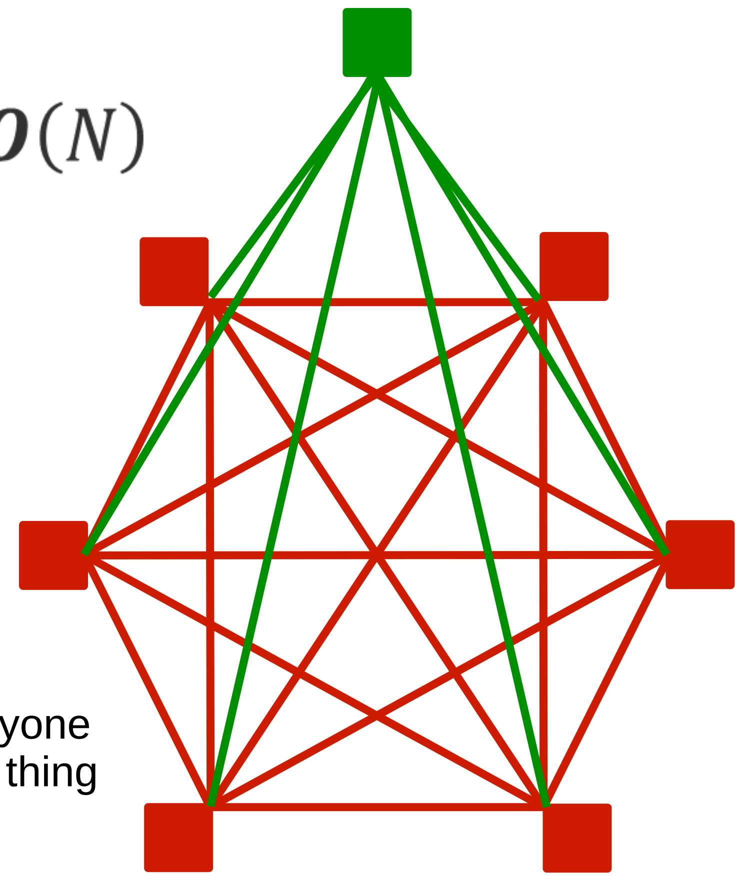
- Fully connected
  - Cost to add one node = $\boldsymbol{O}(N)$
  - Total cost = $\boldsymbol{O}(N^2)$

this is too much

*Solution: sharing*

some stuff have to be shared

N:1 is more complex - easier to broadcast to everyone rather than having everyone communicate to one thing



# What can we share?

- Endpoints different processes running on one endpoint
  - We're already doing that
  - Multiprocessing, multiprogramming, etc.
  - The rest is for CS 111 (Operating Systems)
    - Virtualization (abstraction!)
    - Resource sharing within a FSM
- Channels channel sharing?
  - Let's explore…

# Sharing a channel

- Sharing in different directions
  - Full-duplex

channel goes in two directions!

- Shared outgoing destination
  - A way to support broadcast/multicast
- Shared incoming source
  - To gather information from multiple sources

# The big reason

**Scale**

scale is important in computer networking

# Scale

- A relationship between two variables and their ratio
relationship between variables
  - An independent variable that changes arbitrarily
  - A dependent variable that is expressed in terms of the independent one

    $y = f(x)$

- The ratio grows in some way: $\frac{y}{x} = \frac{f(x)}{x}$

  $\frac{f(x)}{x} \leq c * g(x)$ where *c* is a positive constant

- We say *f(x)* is bounded by *O(g(x))*

this is an upper bound

# Scale magnitude

- Growth is bounded
  - No increase
    - Unlimited messaging at no extra cost
  - Logarithmic increase $y = c \log_k x$
    - Phone numbers –one digit gets 10x more numbers
  - Linear increase $y = cx$
    - 6 phones cost roughly 6x one phone
  - Polynomial increase $y = cx^k$
    - Every new person in the room adds N possible pairings
  - Exponential increase $y = ckc^x$
    - Not as bounded!
  - Beyond exponential increase $y = cx^{cx}$
    - Even worse, like factorial

# Why do we care?

- Metcalfe's law
  - Value of a network is related to the number of pairwise opportunities

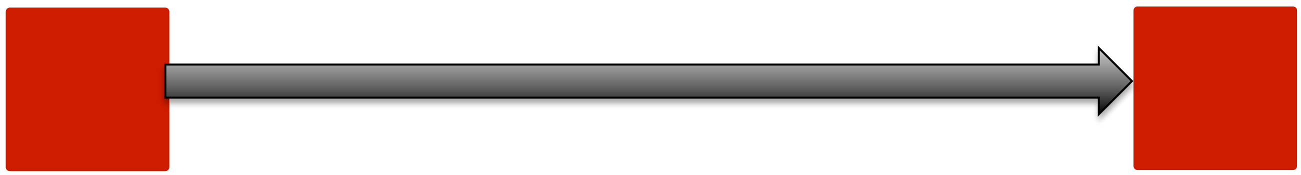
    the more pairwise opportunities, the better the value of network

  - I.e., for N nodes, value is $N^2$
  - In a fully-connected network, cost is $N^2$
  - Ratio of value to cost is 1:1
    - Can we do better?
    - Can we make the network cost grow more slowly than the increase in value?

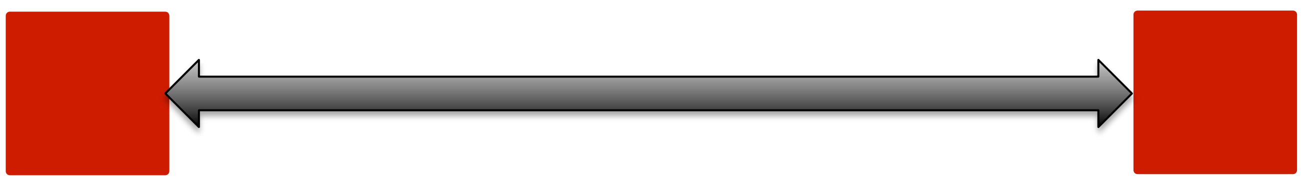
      ^ that is what our goal is

# 2-party sharing

- 2-party channel



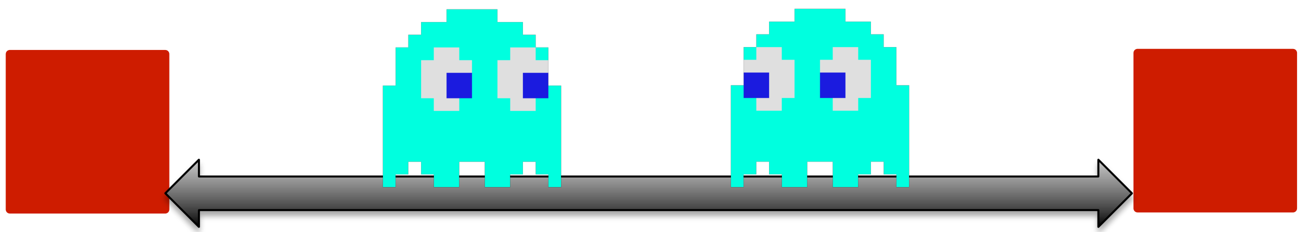
- Let’s make it two way:



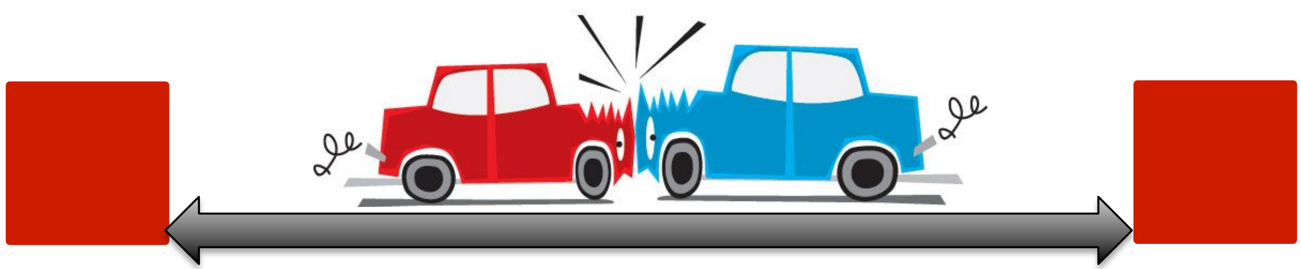
- How?

# Signals in different directions

- Some types of particles don’t interfere
  - Bosons: pass right through each other

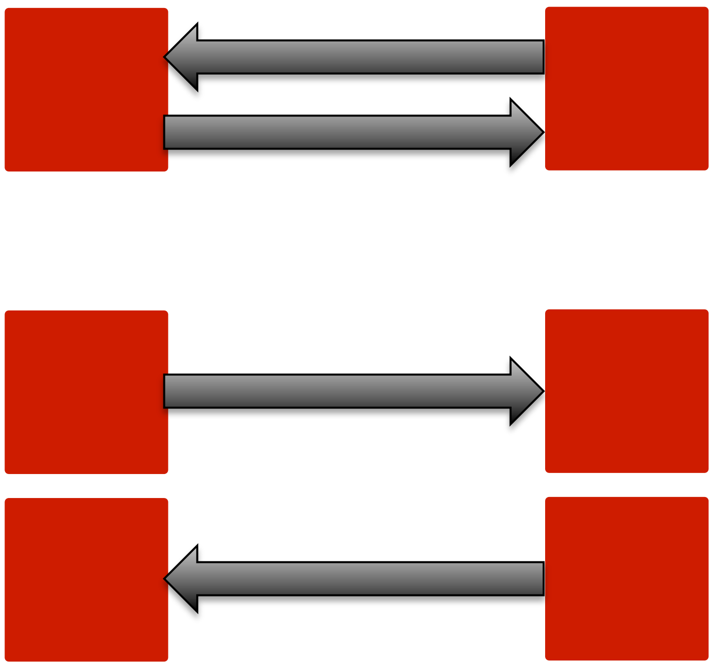


- Others do interfere
  - Fermions: collide (Pauli exclusion principle)



# For those that interfere,

- Keep them separated
- By space
  - Two simplex channels
  - Back where we started!
- By time
  - “Timesharing”
  - Time-division



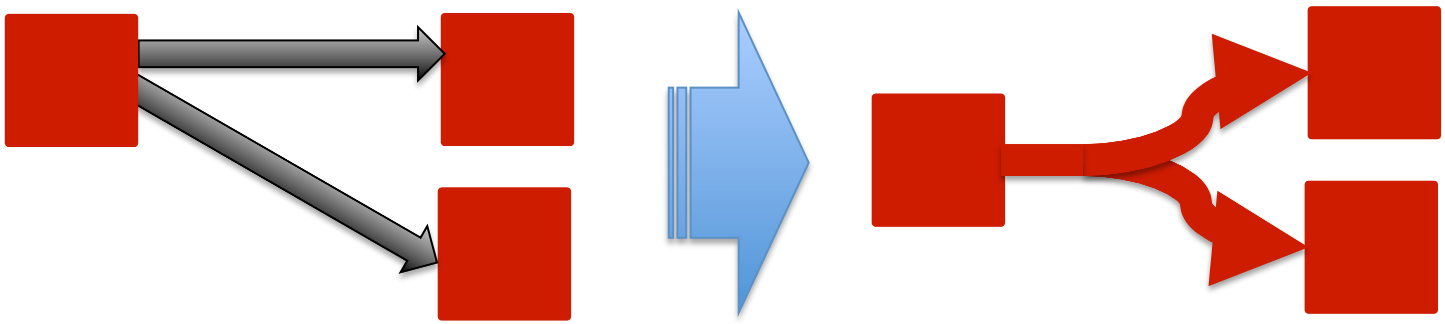
# Time sharing control

- Prior agreement
  - I.e., embedded in the protocol description
  - Requires a common time event (synchronization)
- Central controller
  - One side controls the communication

*We'll see more general cases later*

# N-party sharing: 1 to N

- Share an outgoing channel
- One channel to several destinations



What's the difference between these two?

# 1:N – How?

- Receivers *all* see what transmitter sent
  - "Non-destructive" reads

receivers all see what the transmitter sent

- Which receivers accept the symbols?
  - *All of them* ("native" multicast/broadcast)

multicast: subset get some signal

# Non-destructive reads

- Read by one receiver doesn’t affect others
  - Typical case
- Two ways:
  - Groups of identical symbols (e.g., particles)
  - Perfect copies (measurement doesn’t alter value)
- Allows sharing to assume broadcast messages
  - Can simplify the sharing protocol

# Destructive reads

- Read by one (or a subset) of receivers
  - Rare
- How?
  - Observer effect (read affects value)
  - E.g., quantum state, collect majority of particles, etc.
- Usually considered undesirable
  - Non-determinism – can’t control which receiver reads
  - Prevents using broadcast for sharing protocol
- Can be useful for security
  - Tamper evidence if expect only one receiver
  - Quantum cryptography, e.g.

quantum to communicate so that only one gets information

# Limiting 1:N transmissions

- How can a sender control which receiver gets the message?
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets ("languages")
  - Label the transmission <u>*destination*</u> (names)

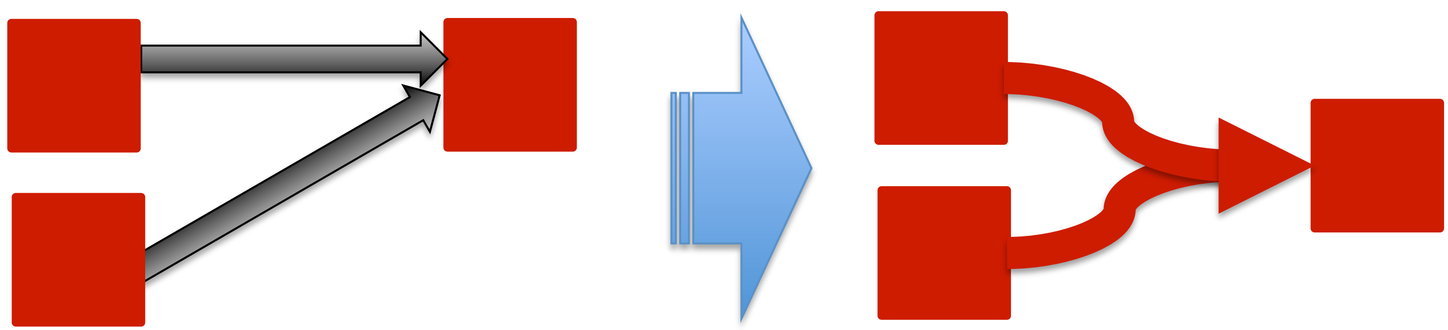
encode the information - write encoding and decoding to get

All can be internal to the source
I.e., this is the easy part

multicast in broadcast medium

# N-party sharing: N to 1

- Share an incoming channel
- One channel from several sources



N sources to 1 receiver

# N:1 – How?

- Receiver sees what *all* transmitters sent
  - Technically difficult, at the particle level
  - Collisions between particles
  - Or confusion of who sent which particle
  - *One* of them
    - But which one?

# Limiting N:1 transmissions

- How can transmitters avoid collisions?
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets (“languages”)
- How can a receiver determine transmitter?
  - (all of the above)
  - Label the transmission <u>*source*</u> (names)

Why is this harder than 1:N?

# N:1 is harder than 1:N

- 1:N
  - Coordinate use internal to the source
    - Time, symbol set
  - Naming needs to be coordinated with receiver
    - Need to use IDs the receiver recognizes
    - But each set is unique in the context of that sender

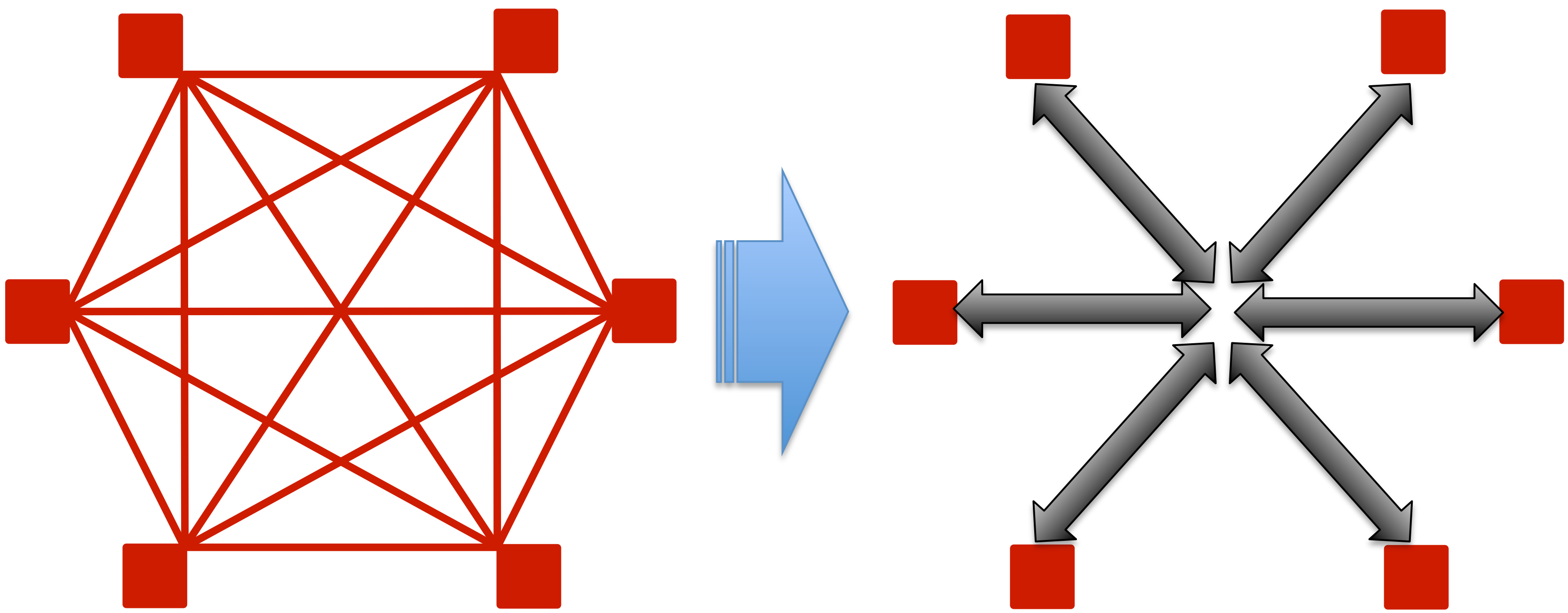
    use ID that is recognizable

- N:1
  - Coordinate use *between* sources
    - Time, symbol set
  - Coordinate naming
    - Converse of 1:N naming, but name attached by sender
    - How does sender know it has a unique name?

  multiple sources, each one is introducing info into the channel

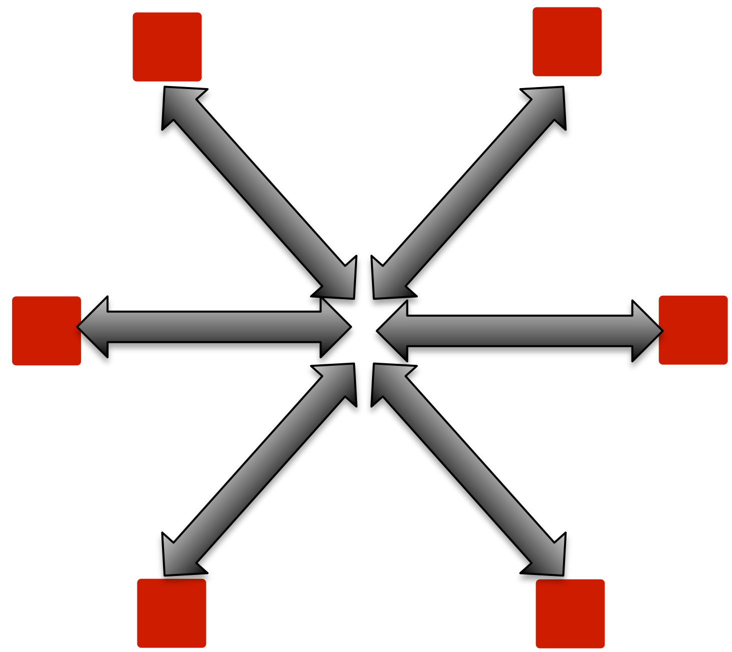
# N-party sharing: N to N

- Instead of $N^2$ links
- Use just $N$



# The ultimate shared channel

- One channel
  - All parties transmit on
  - All parties receive from
- Minimizes link cost
  - One link to add one node



# Single shared channel examples

- Freespace
  - Diffuse infrared
  - Omidirectional RF
- Wired
  - Bus
  - Ethernet
- Fiber
  - Individual fibers to a passive coupler

# N:N – combine rules

- 1:N – control receiver
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets (“languages”)
  - Label the ***destination***

a lot of people using
this one shared communication
mechanism.

- N:1 – avoid collision (control transmitter)
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets (“languages”)
- N:1 – identify source
  - (all of the above)
  - Label the ***source***

Shannon's result in a noisy channel: if you sent 0, 50/50 it is 1 or 0
Go through the calculations in the reading haha

# Summary

multi-party communications

- Channel sharing affects network size
  - Distance, number of parties

    share channels

    the fewer the channels, the lower the cost
- Shared channels requires shared namespaces
  - Networking required internal names

    some requirements
  - Sharing requires coordinated names

    limitations:
    -distance
    -
- Sharing requires mechanism
  - Protocols to manage the network, not just to share endpoint state