# 61A Lecture 31

November 14th 2011

# Parallel and Distributed Computing

Coordinating groups of computers

# So far

functions

data structures

objects

abstraction

interpretation

evaluation

## So far

functions

data structures

objects

abstraction

interpretation

evaluation

One program
One machine
One computer

# Parallel and Distributed Computing

## Distributed Computing

Groups of computers communicating and exchanging data with a shared goal.

- Communication networks
- Data storage
- Large scale computing

## Parallel Computing

One computer with many processes collaborating to execute the same program faster.

- Speeding up computation

# Lecture plan

## Today

`Distributed computing`

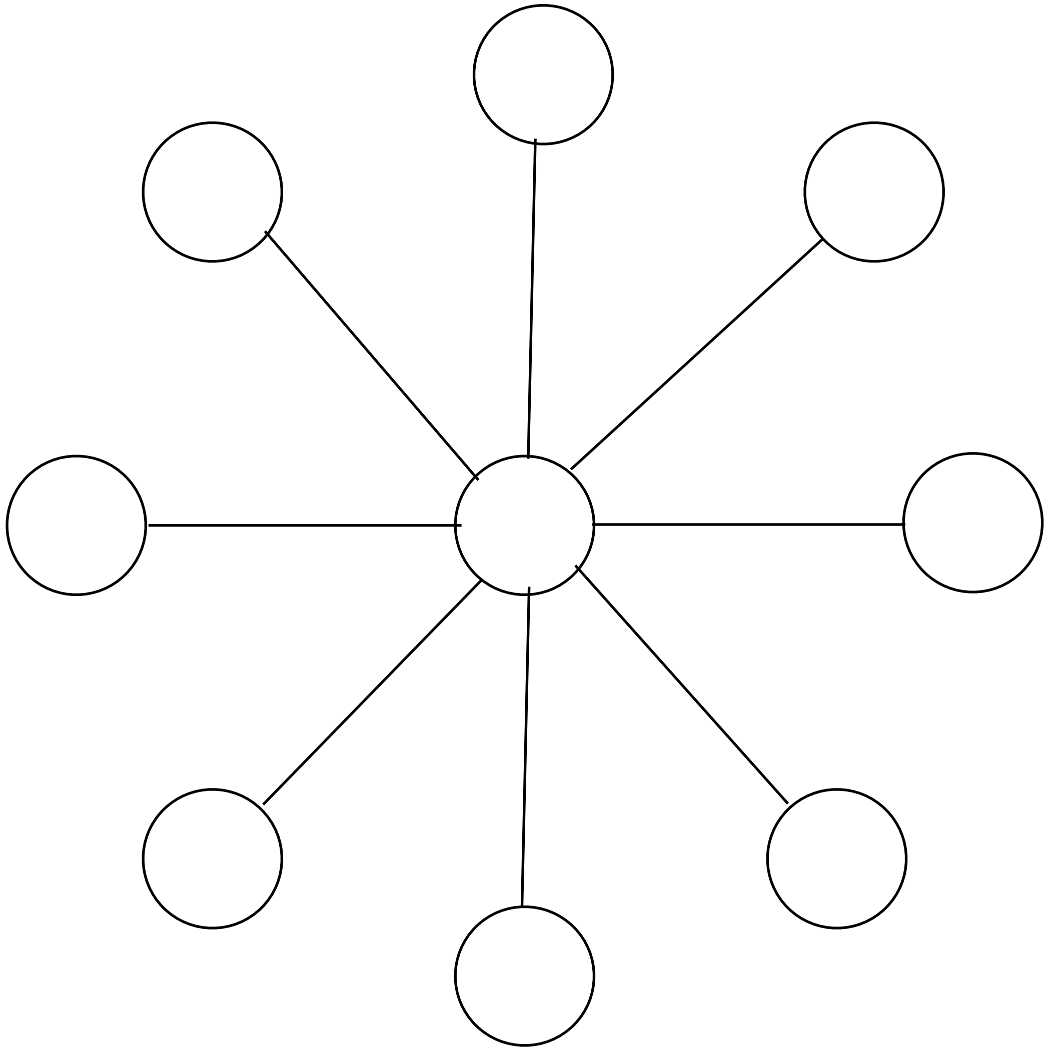
## Wednesday

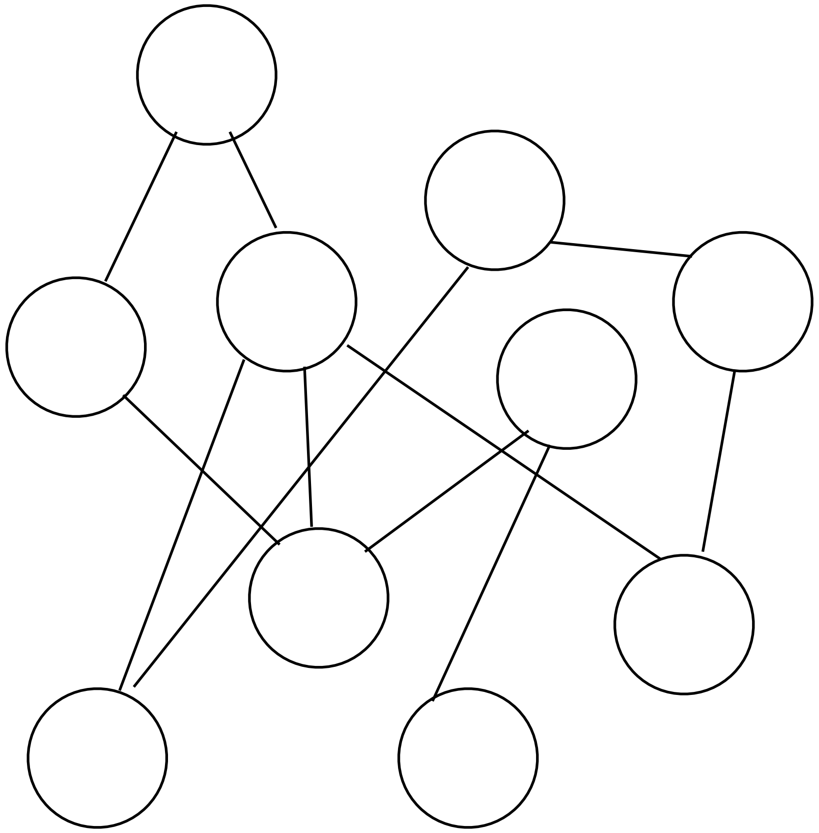
`Parallel computing: problems`

## Friday

`Parallel computing: solutions`

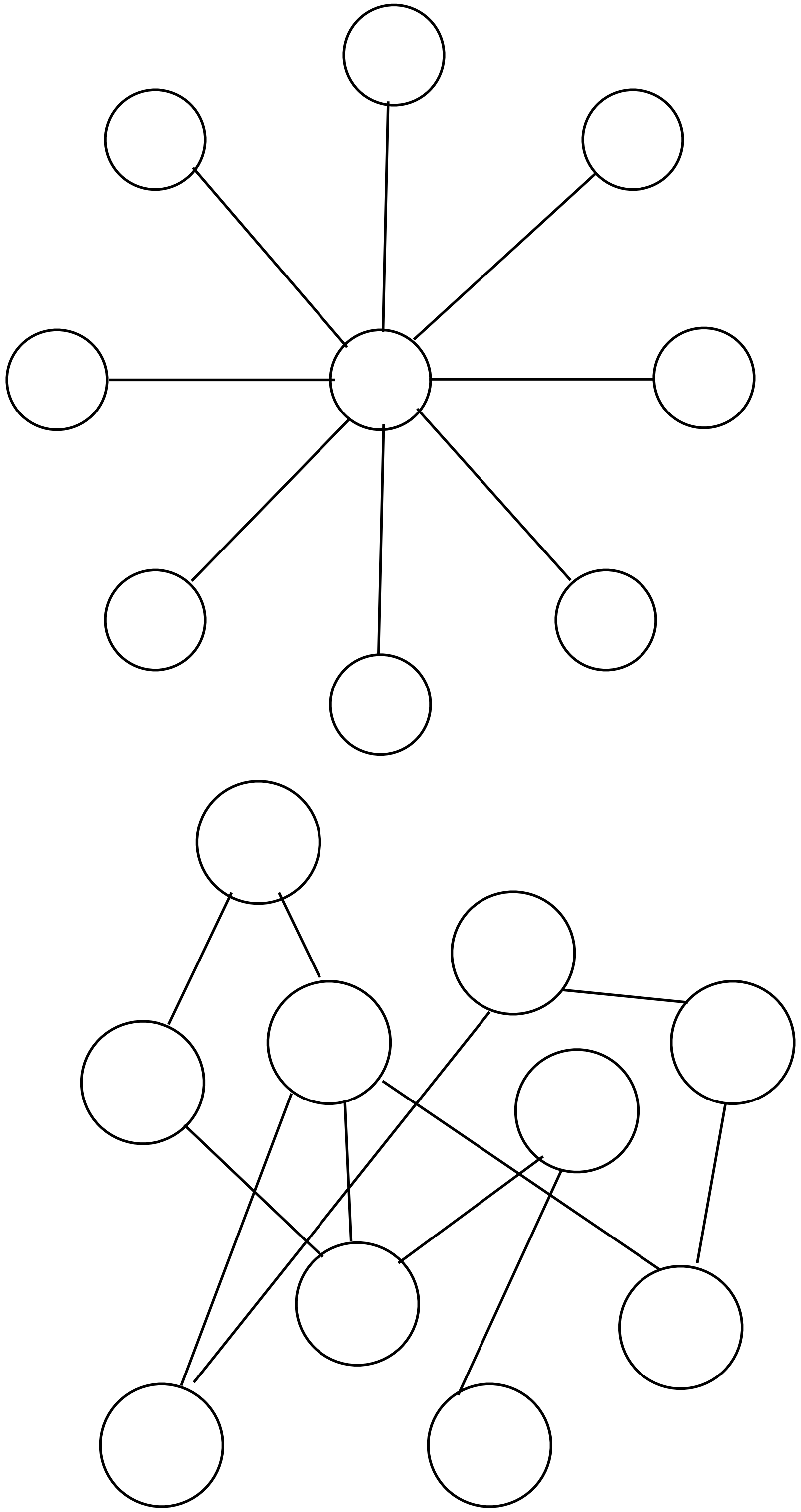
# Distributed Computer Systems





Interconnected groups of independent computers that collaborate to get work done.

# Characteristics of distributed systems



1. Independent computers

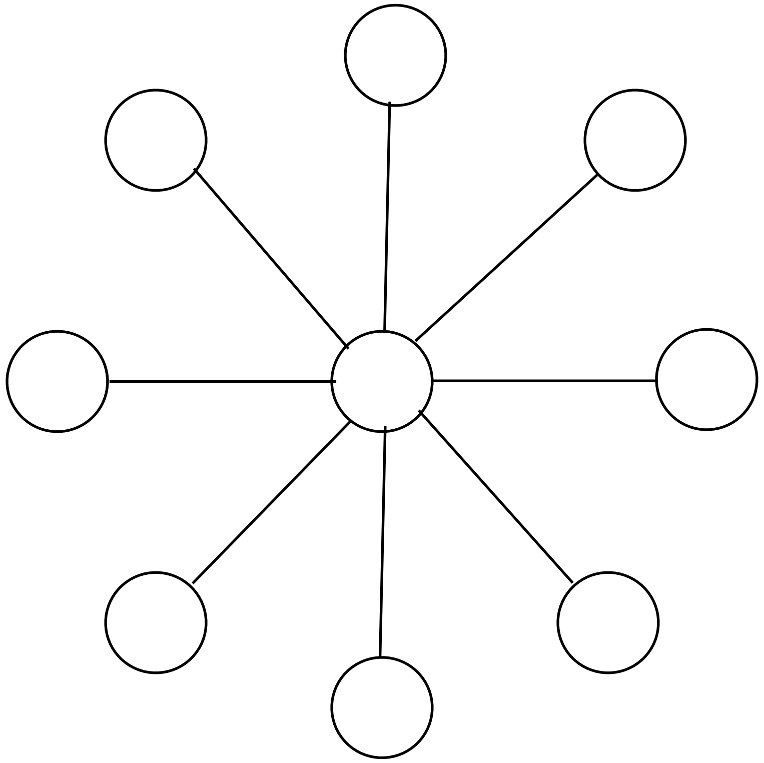
2. (Often) In different locations

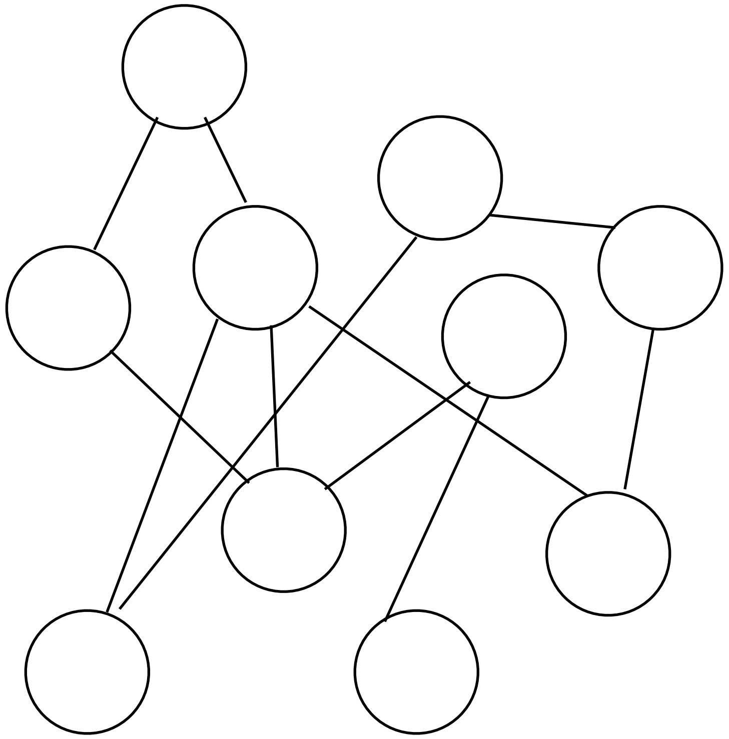
3. Connected by a network

4. Communicate by passing messages to each other

5. A shared computational goal.

# Examples of distributed systems





**Information sharing & communication**

Telephone networks, cellular networks

The world wide web

Skype, IM,

Xbox/PlayStation and other online multiplayer systems

**Large scale computation**

“Cloud computing” – Amazon and Microsoft

MapReduce – later in this course

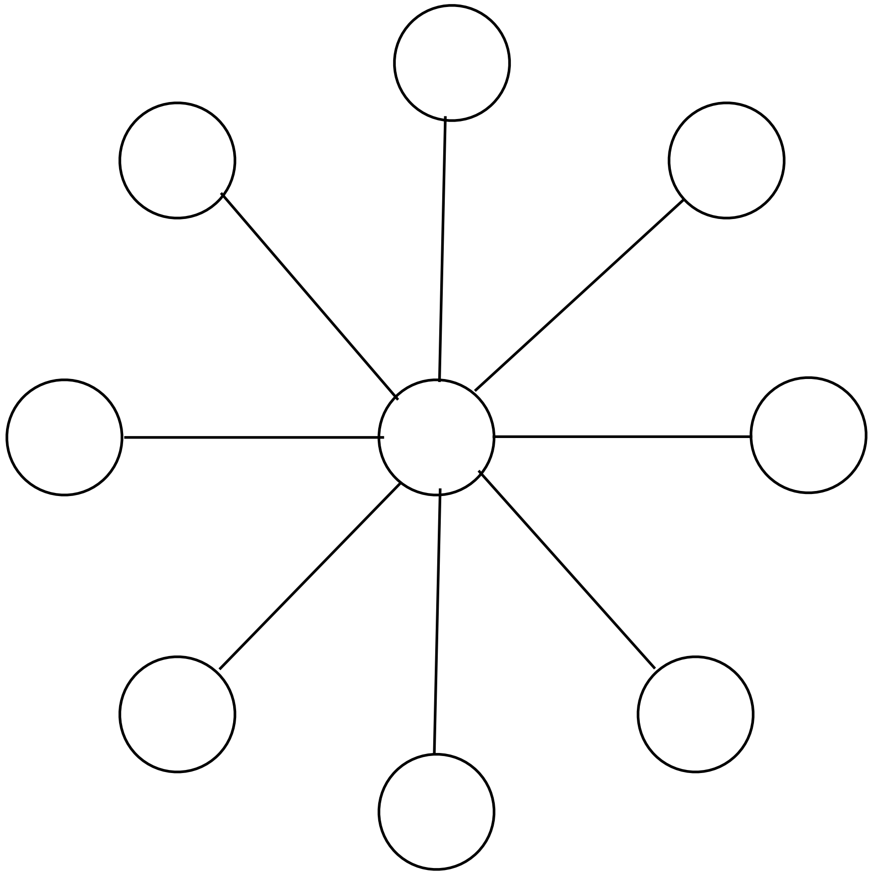
# Topics in Distributed Systems

- Architectures
  - Client-server
  - Peer-to-peer
- Message passing
- Design principles
  - Modularity
  - Interfaces

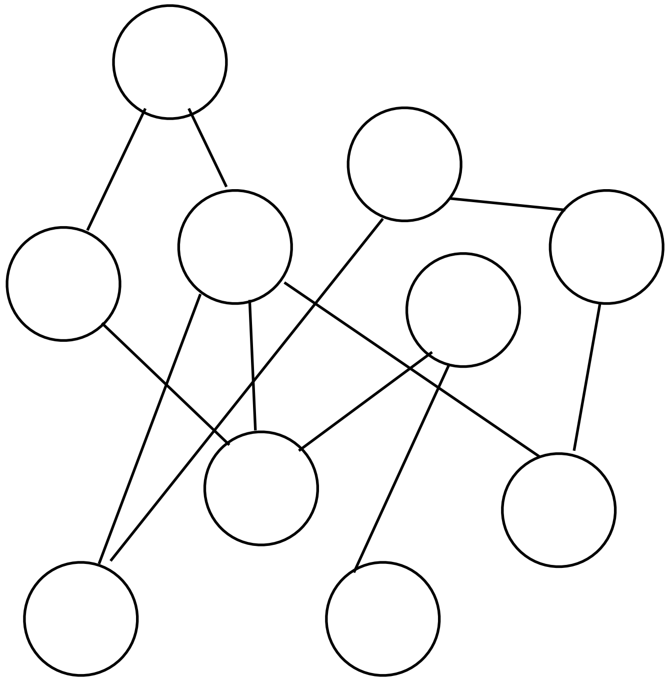
# Architecture

Computers in a distributed system can have different roles depending on the goal of the system.

The network of computers can be structured in different ways.

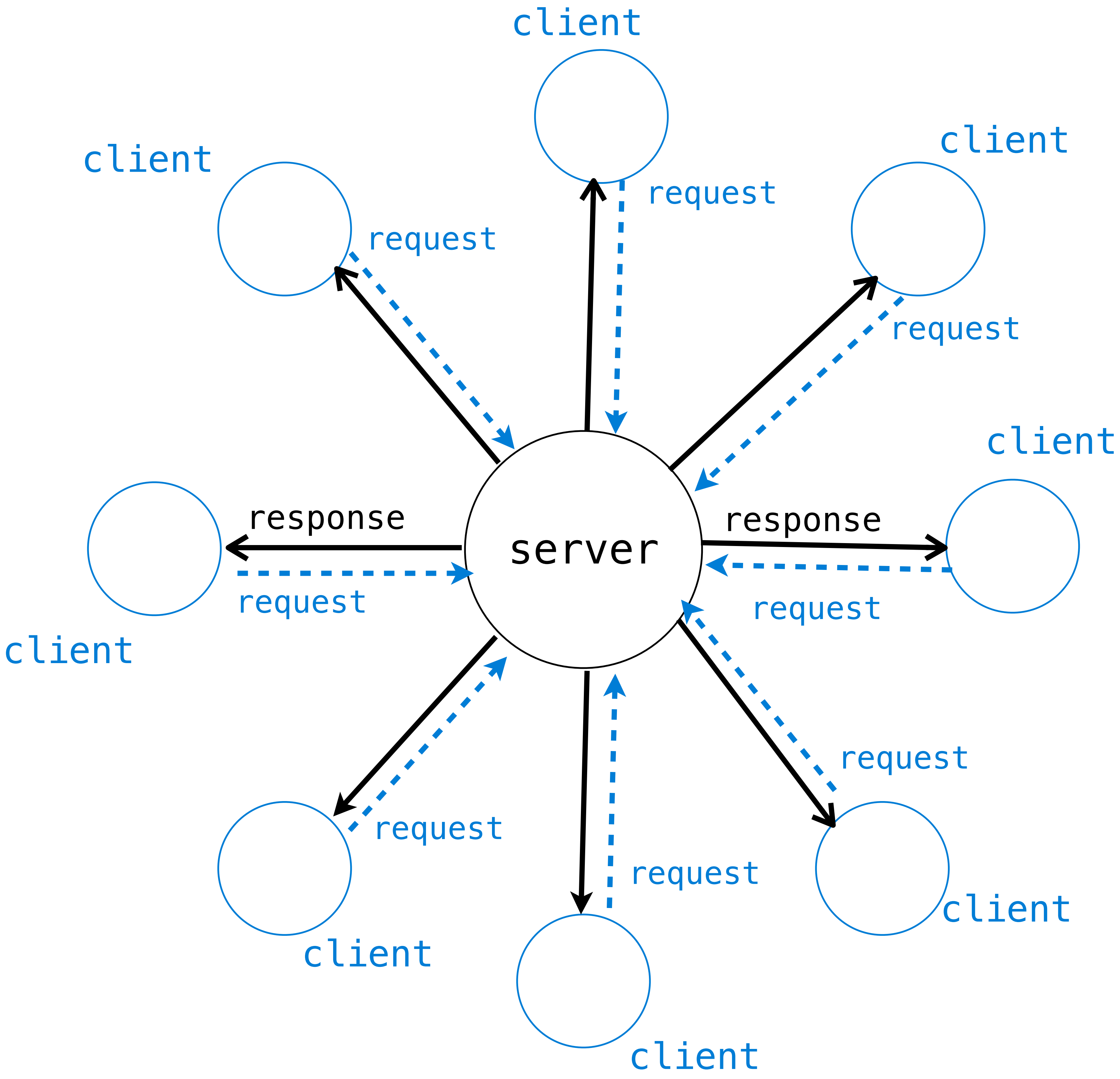


Client-server



Peer-to-peer

# Client-Server Architecture



Good for dispensing a service

2 roles

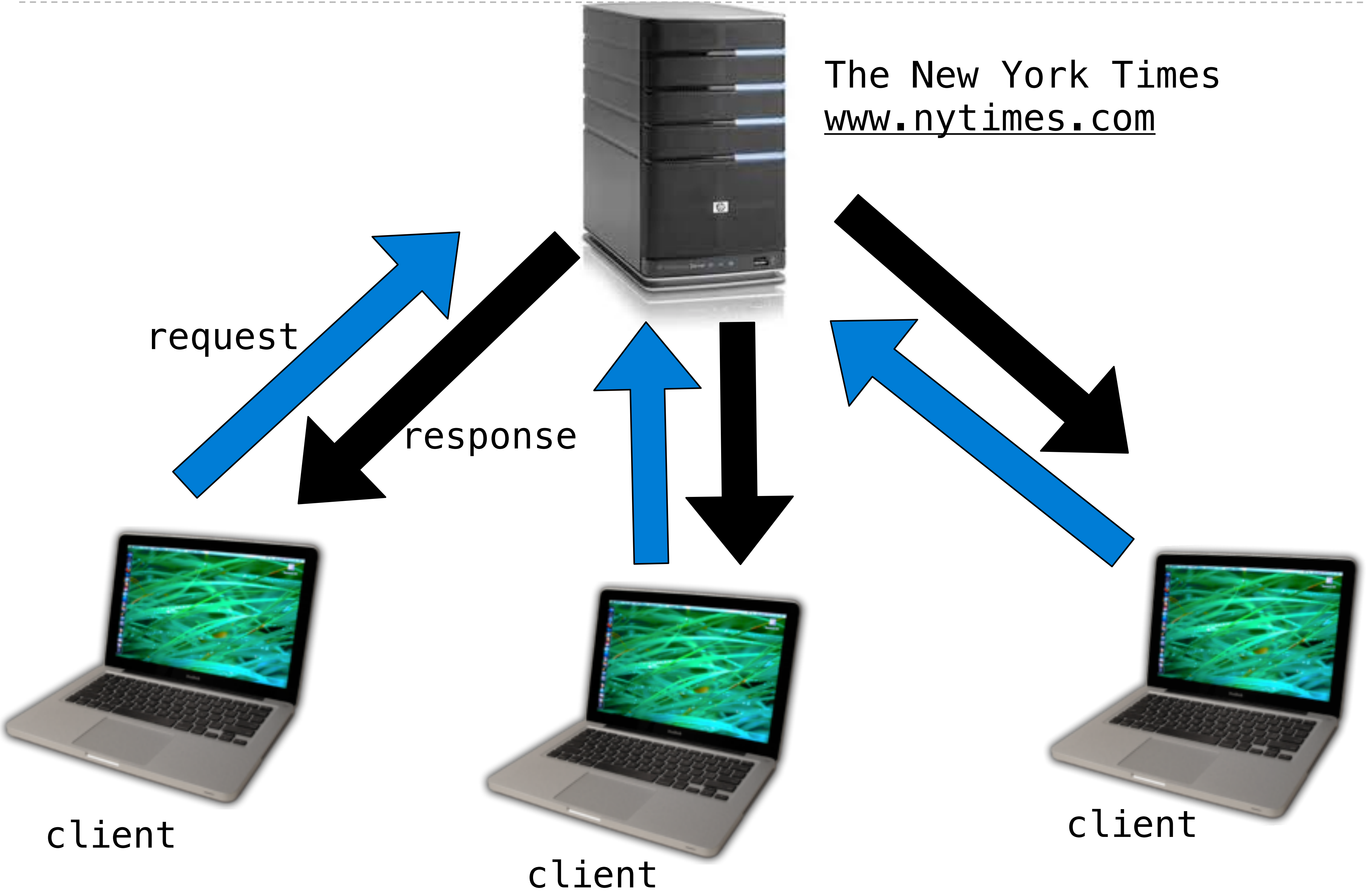
Clients: make requests from server

Server: listens for requests and responds to them.

Many clients

Only 1 server.

# Example: world wide web



# Example: world wide web

## Server’s job

Listen for requests

Calculate front page

- ads
- personalized content

Send web page back to correct browser

## Client’s job

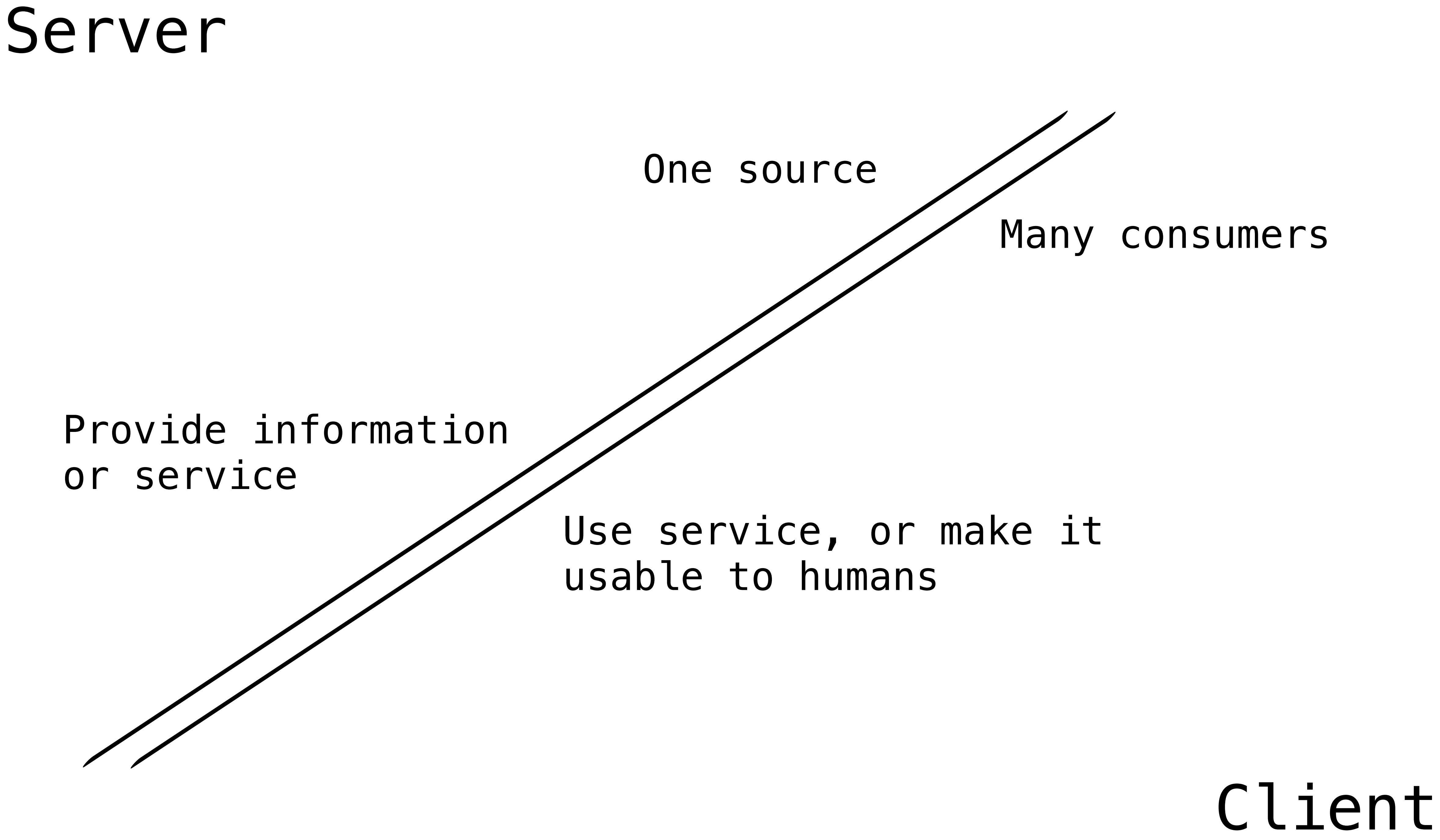
Send correct request to server based on user input

Display received web page

- fonts & colors
- images
- interactivity

Send further requests

# Division of labor



Server

One source

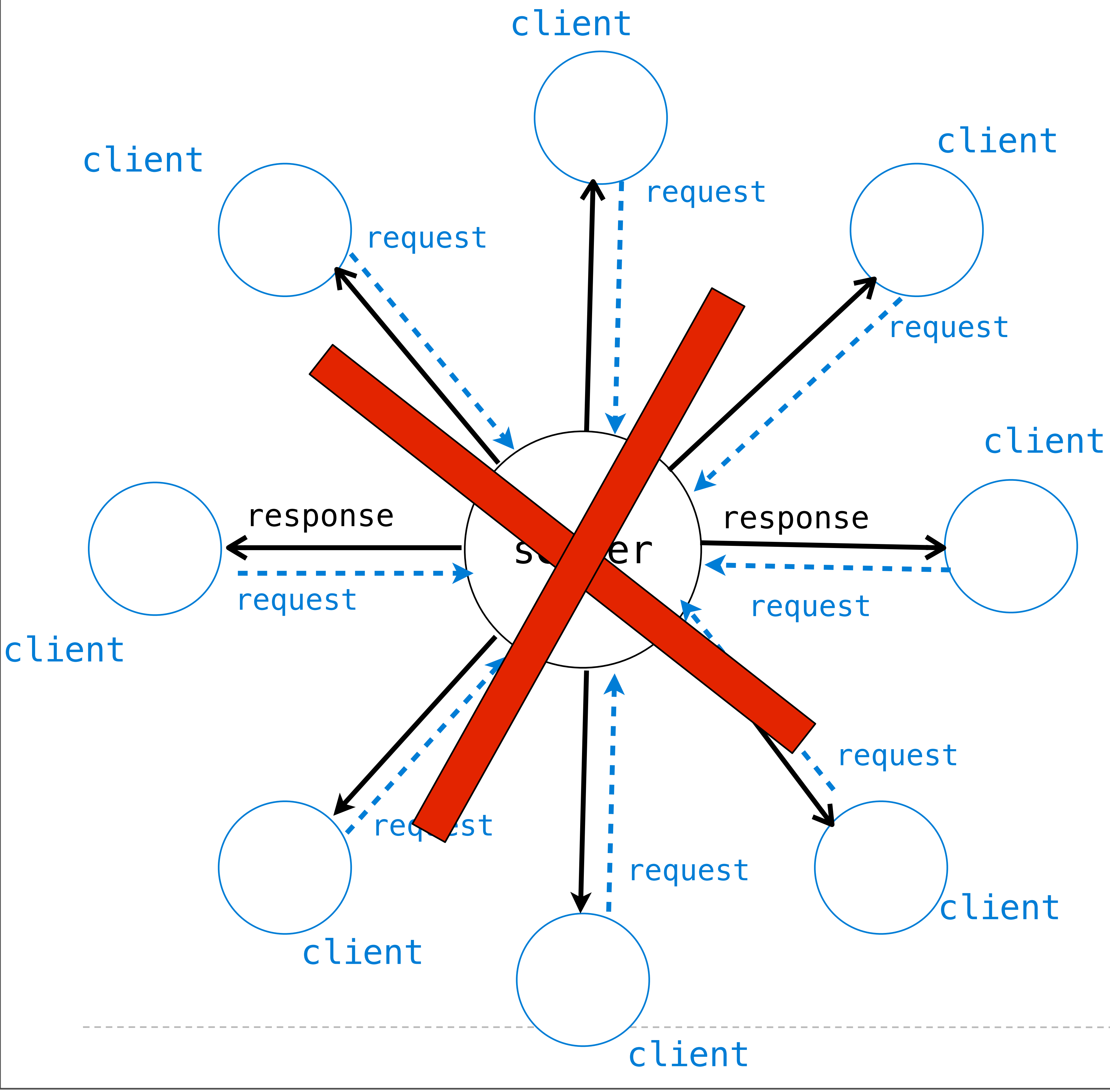
Many consumers

Provide information
or service

Use service, or make it
usable to humans

Client

# Single point of failure

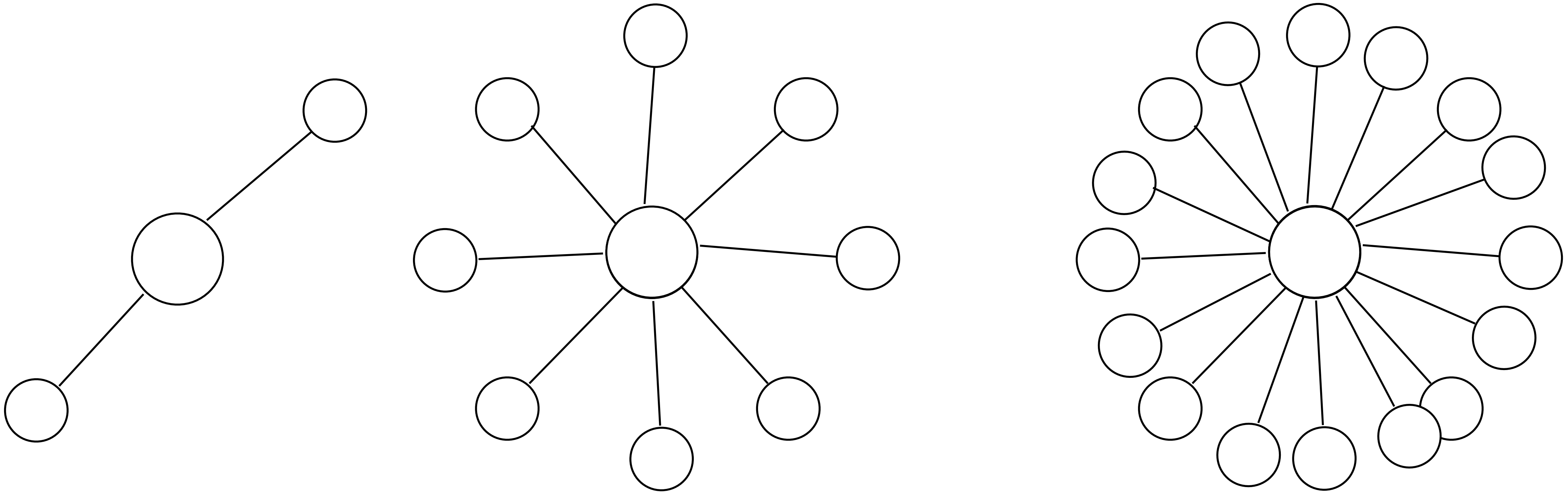


Server is a single point of failure.

System stops working if server goes down.

If client goes down, only that client is affected.

# Disadvantage: performance degrades under load

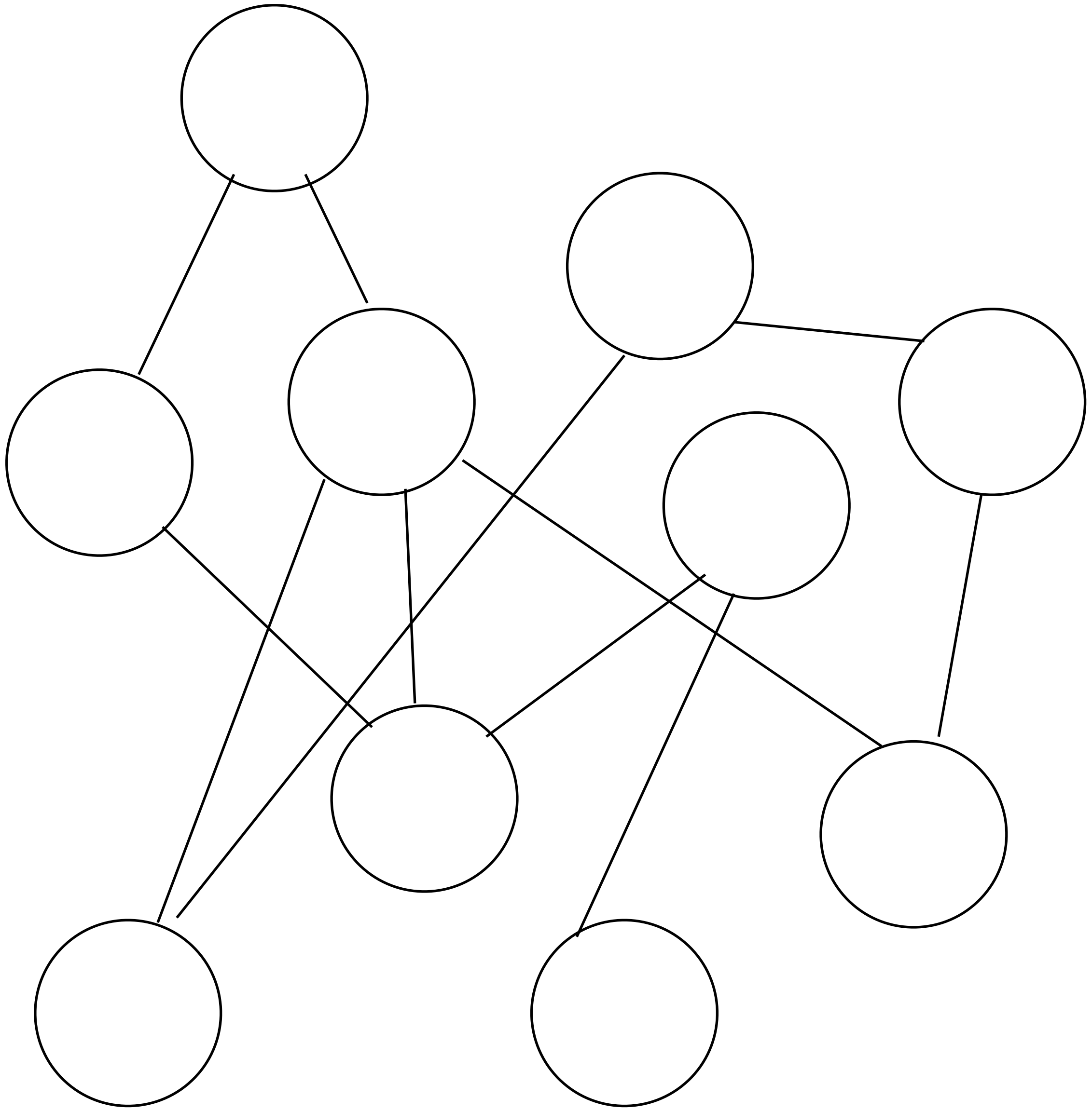


The more clients that want to use a server, the worse the server performs

- Connection speed becomes slow -- limited bandwidth
- Server becomes slow to respond -- limited processing power

Cannot shrink and grow with changing demand

# Peer to peer architecture



Division of labor among all computers

All computers send and receive data

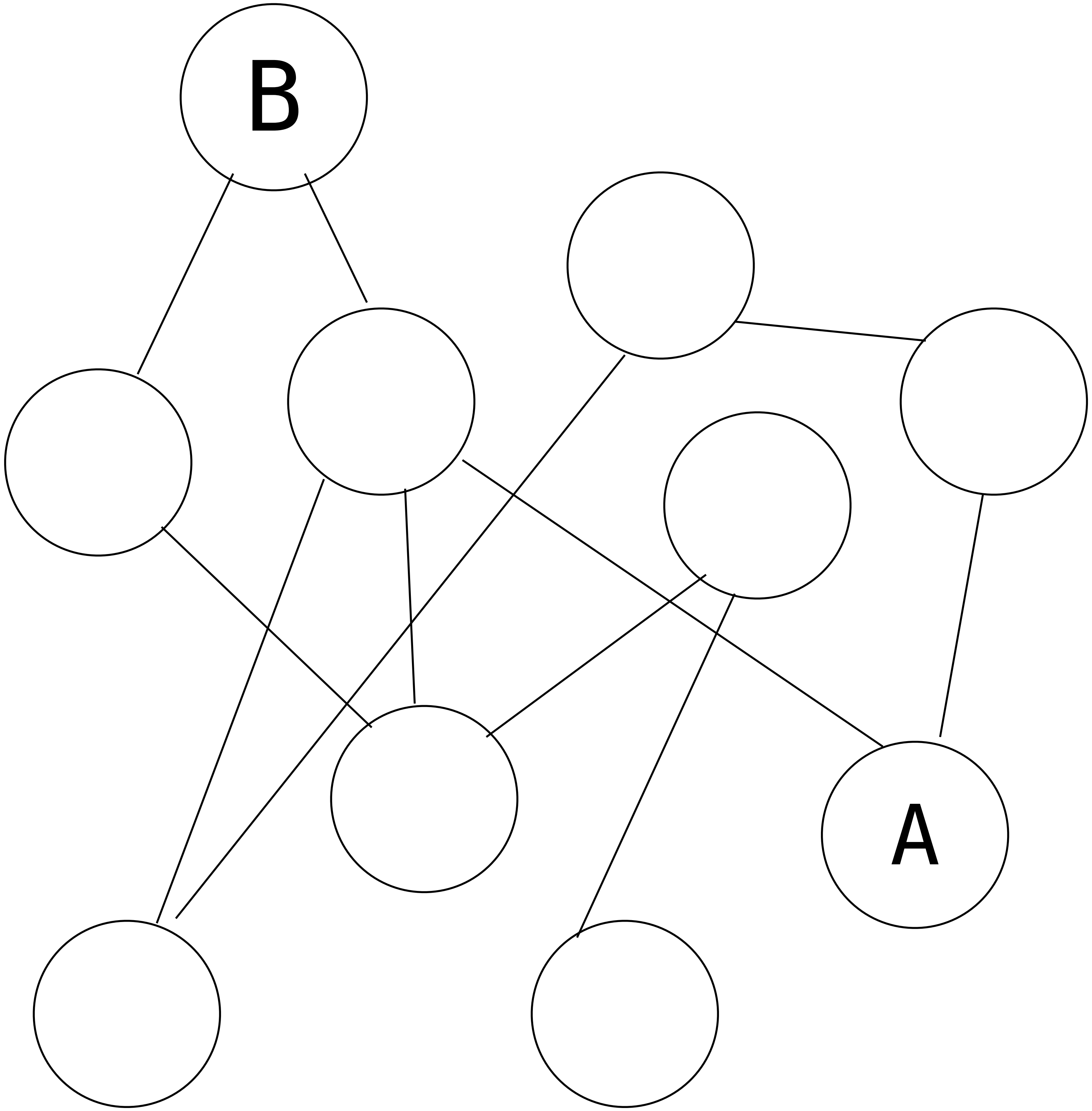
All computer contribute resources

- Disk space
- Memory
- Processing power

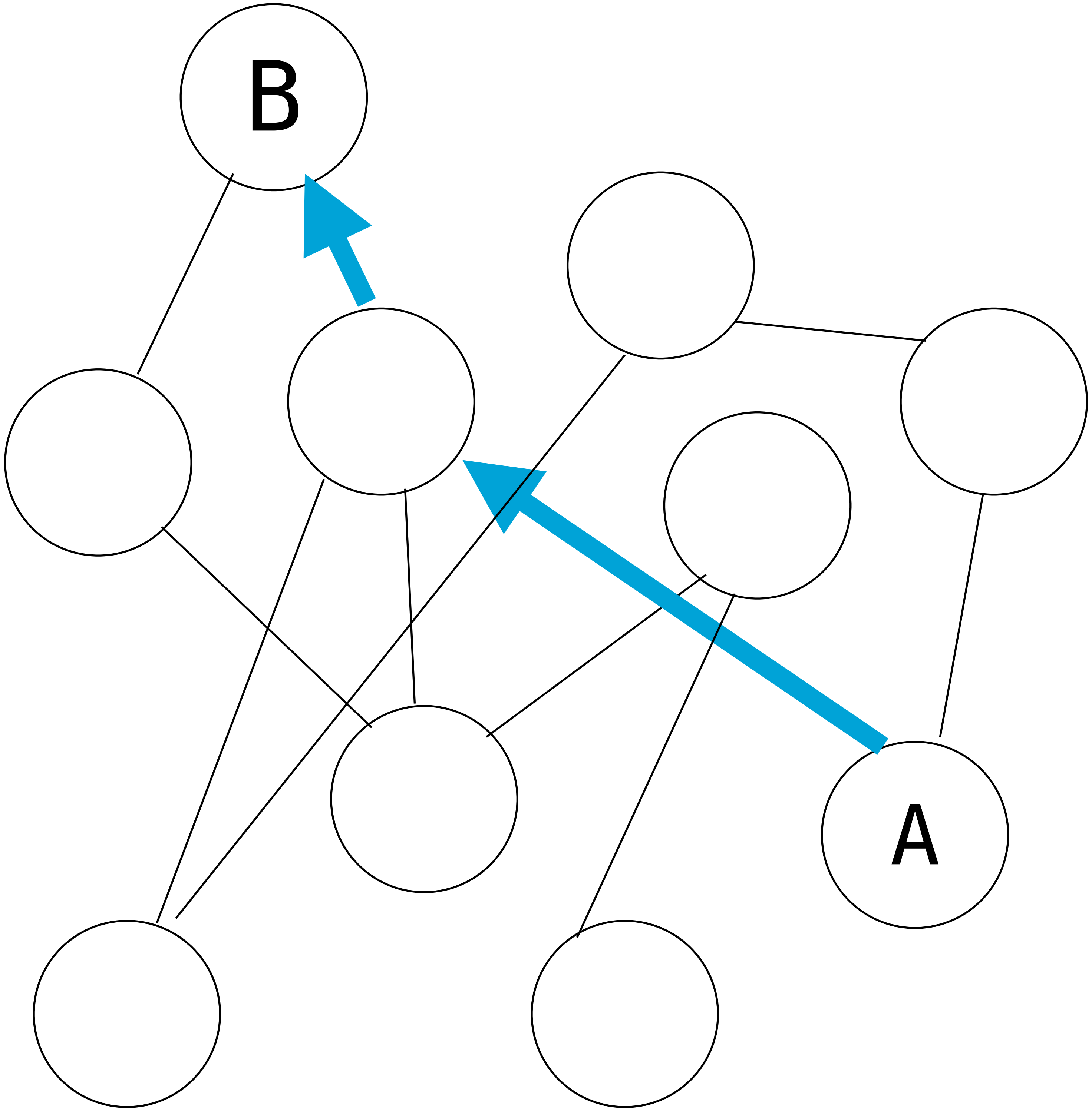
Applications

- Data storage
- Communication
- Large-scale computation

# The importance of an organized network structure

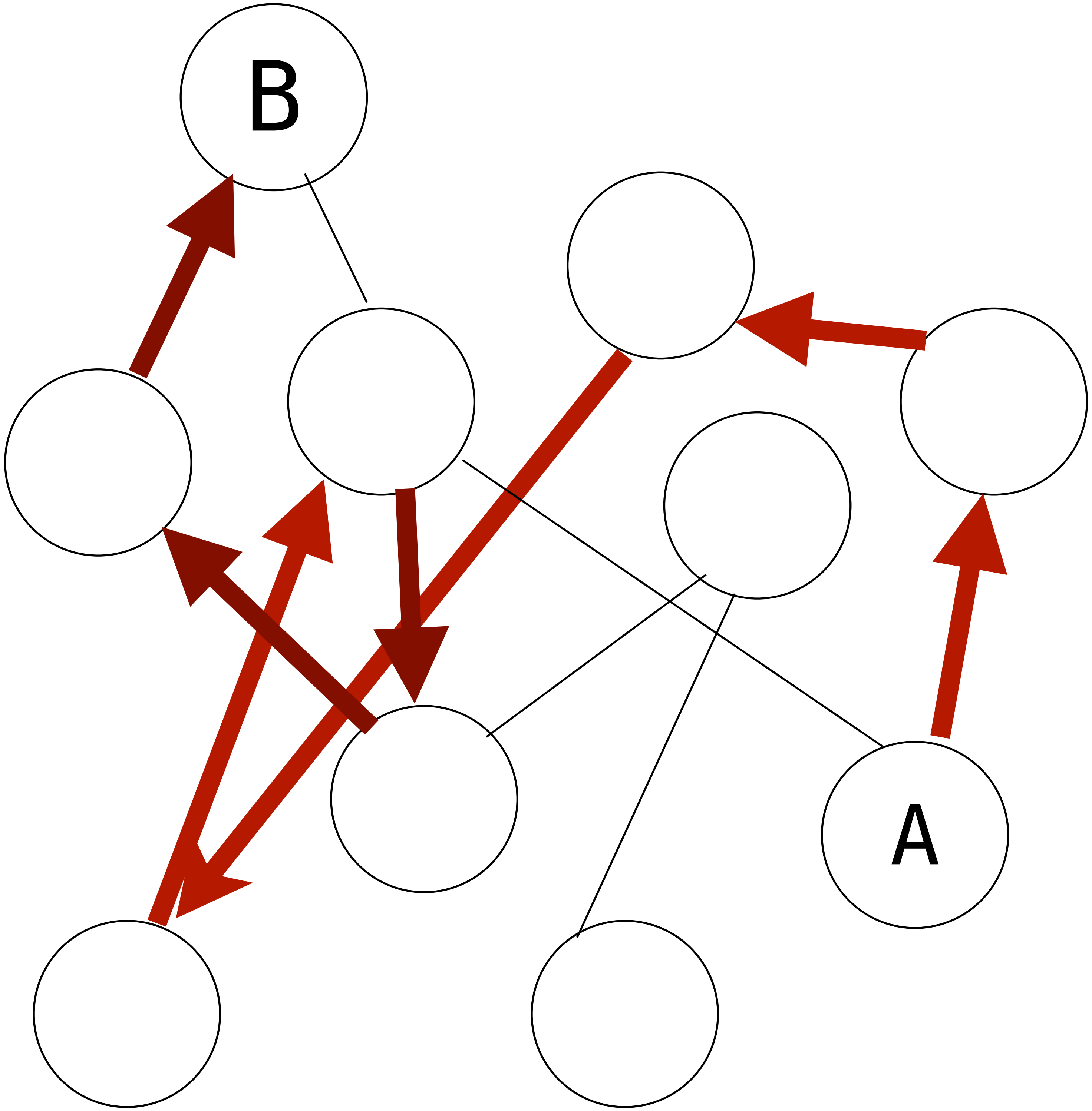


# The importance of an organized network structure



The shortest path for A to send a message to B.

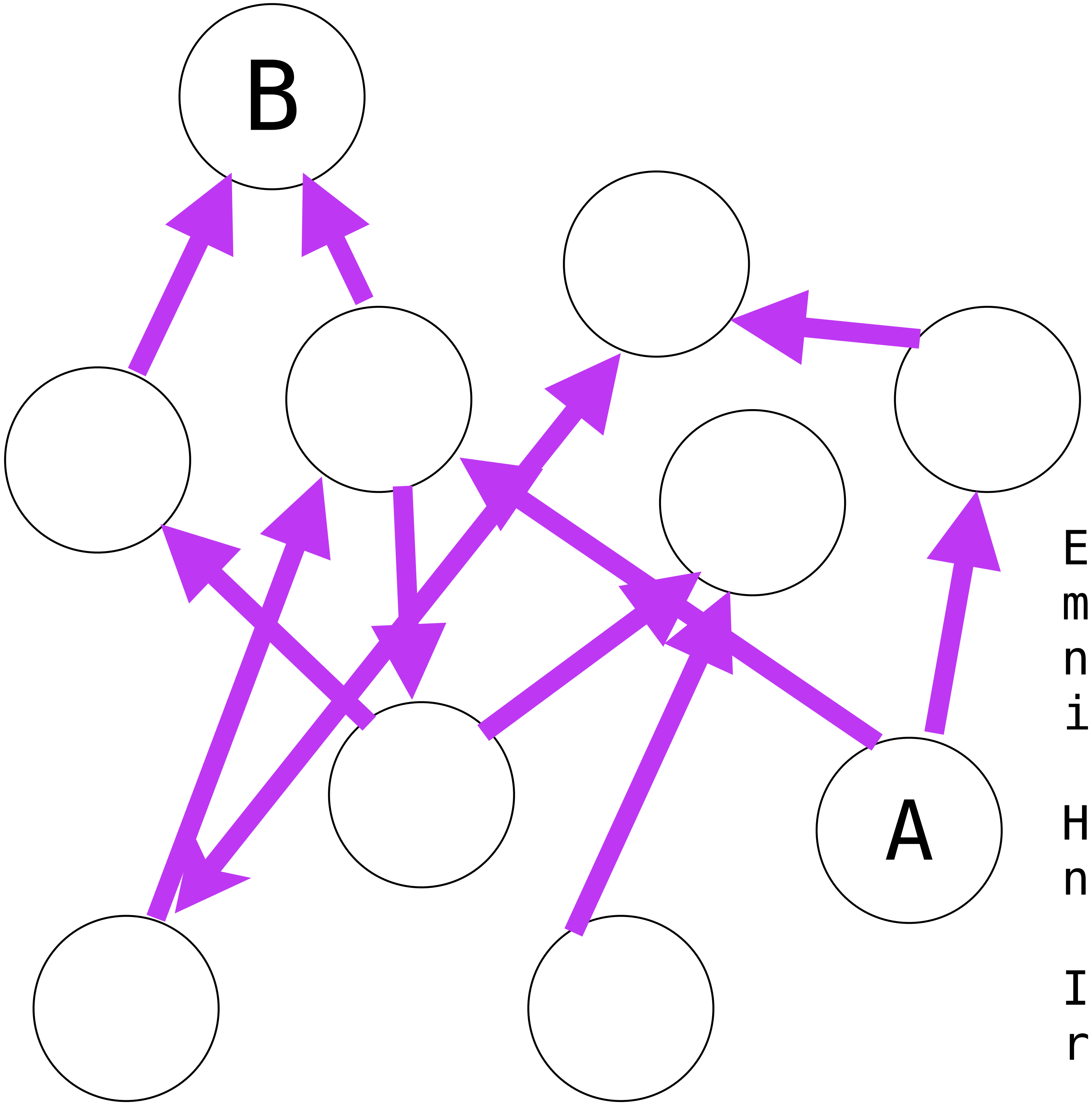
# The importance of an organized network structure



A roundabout path

Inefficient.

# The importance of an organized network structure

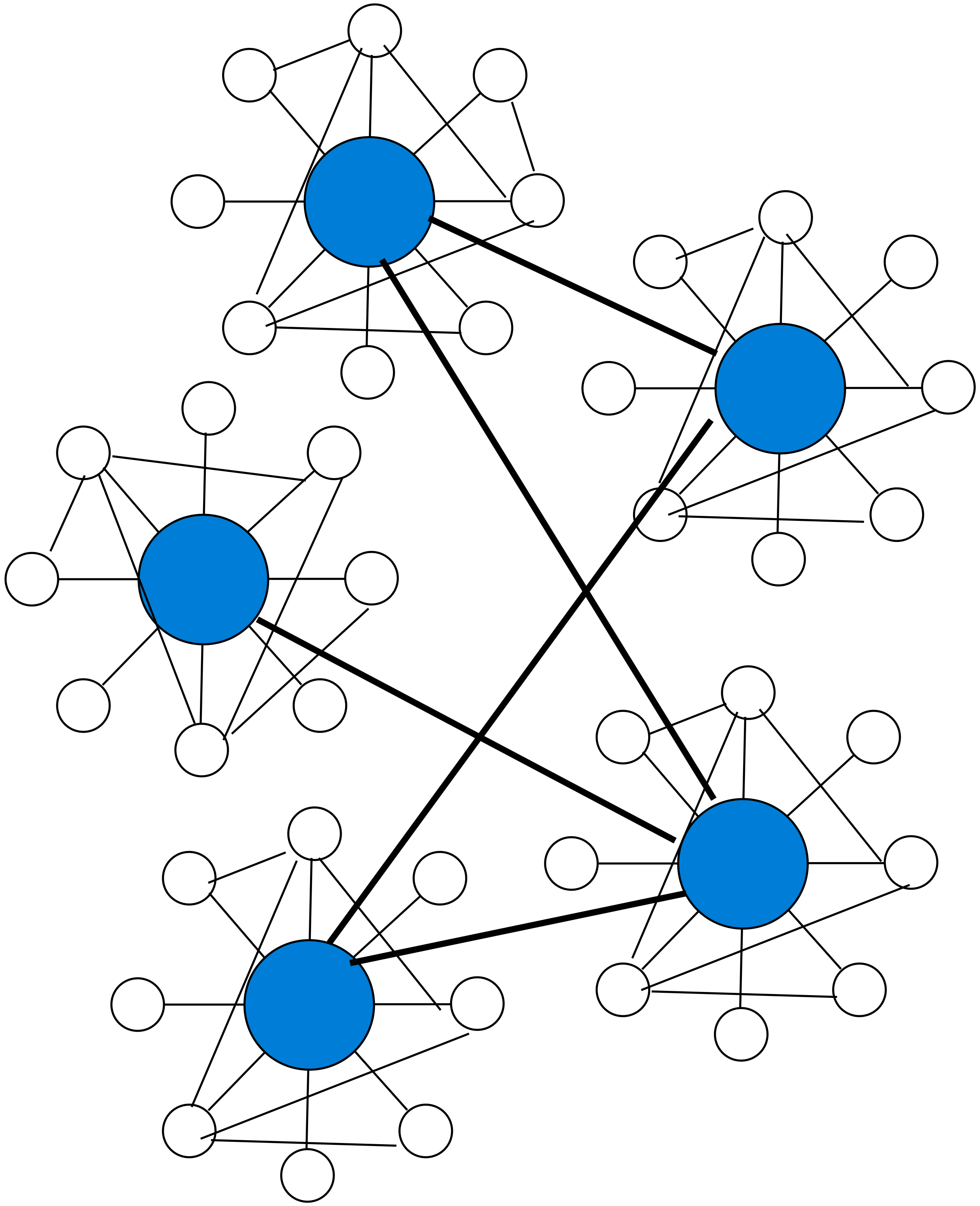


Everyone sends A’s message to their neighbors, until B is reached.

Huge load on network.

Inefficient use of resources

# Supernodes: keep track of network structure



Computers with a special function

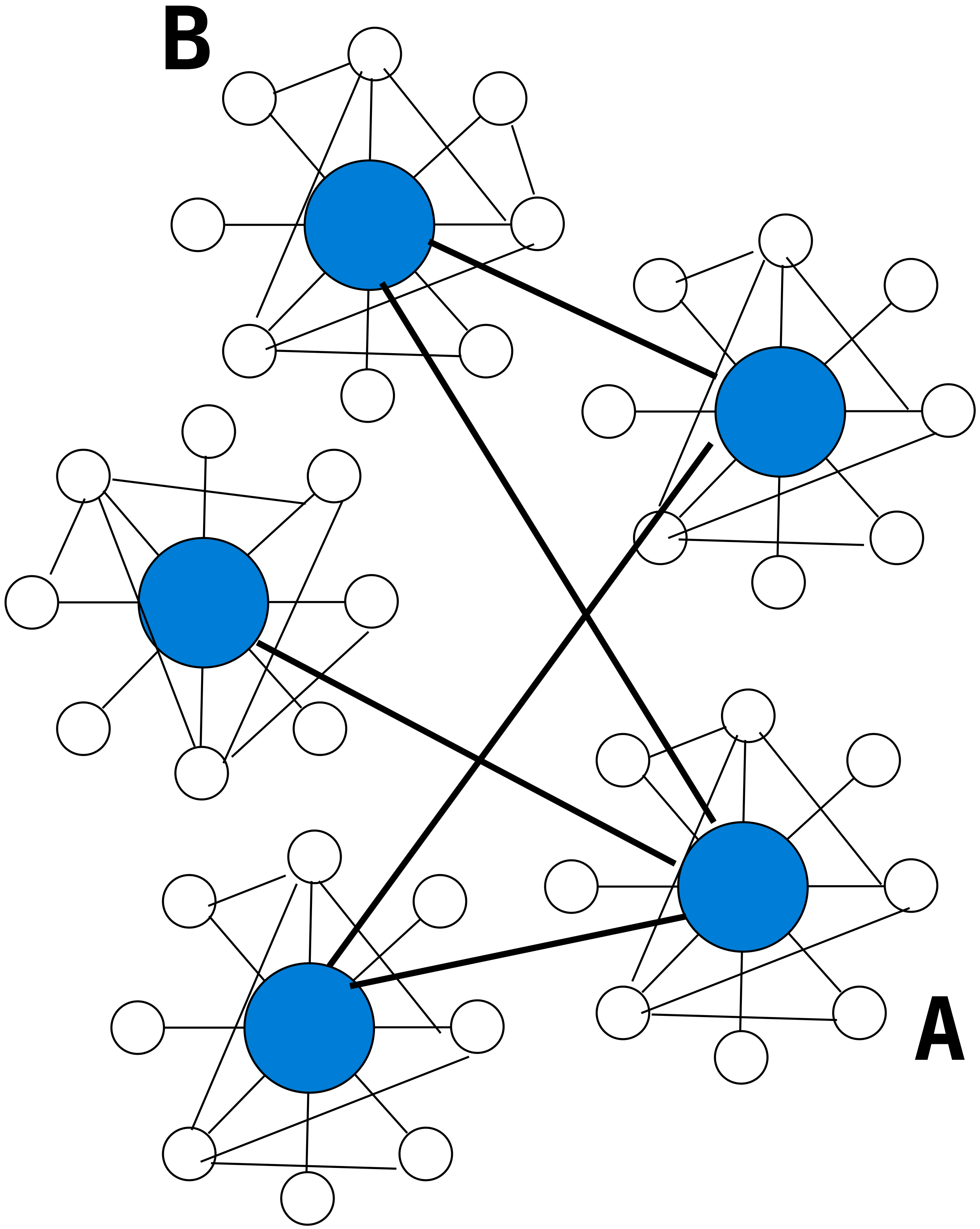
No longer “pure” peer-to-peer

Knows locations of other supernodes

Knows which computers are “under” it

Keeps track of newcomers and computers that leave

# Example: Skype



Peers: all computers running skype

Not a pure peer-to-peer network

Supernodes coordinate users and manage sign-ins and sign-outs.

## Disadvantages

Complex network structure

Inefficiency in communication

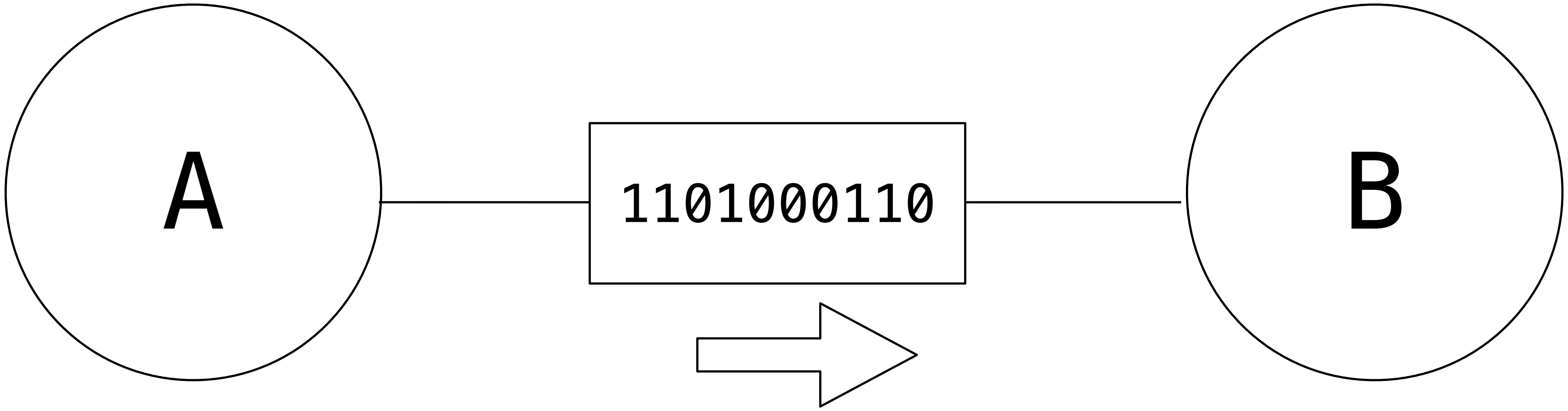
- Can take up a lot of traffic trying to route messages.

## Advantages

No single point of failure

Can grow and shrink with demand

# Messages



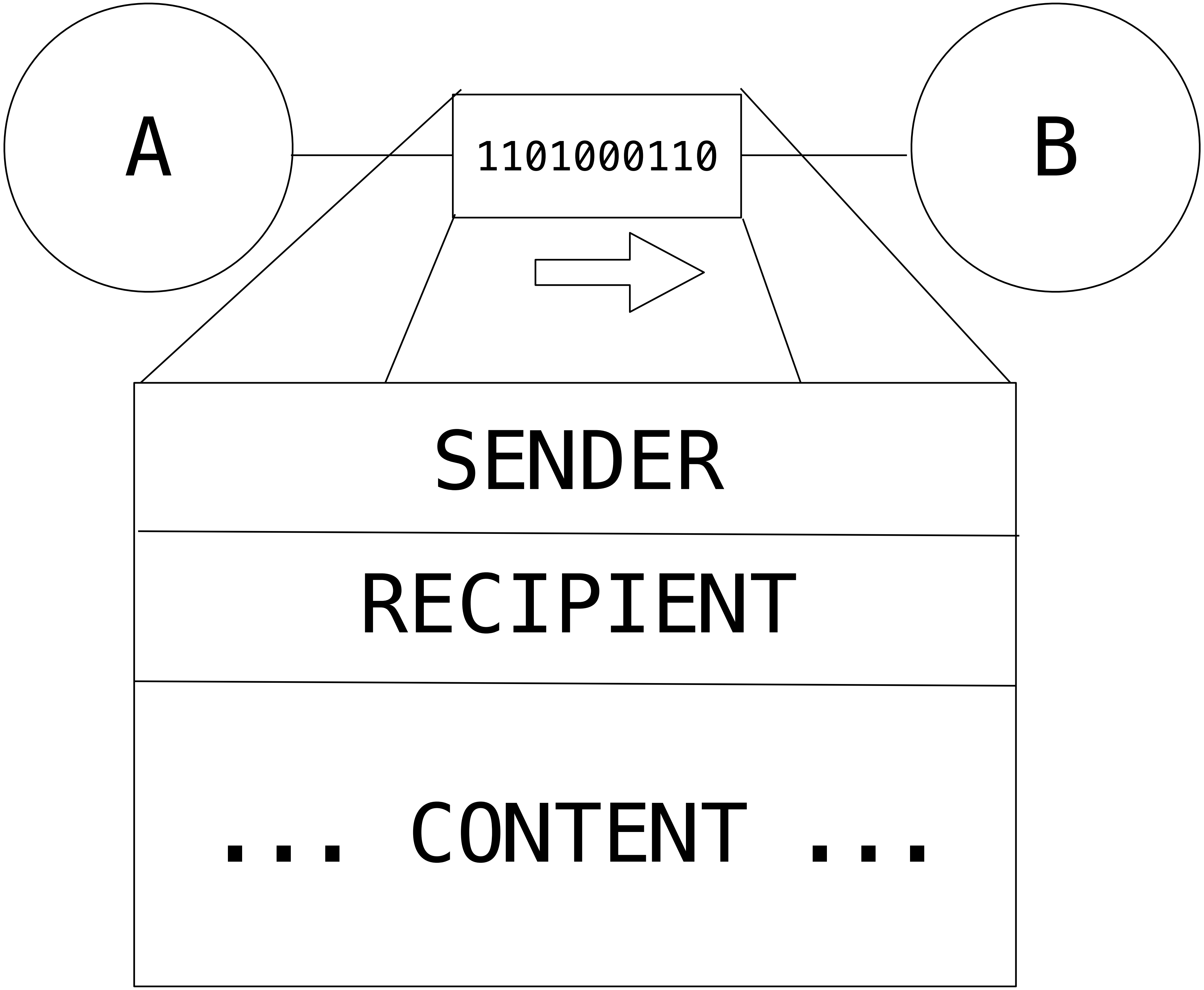
Used to coordinate behavior

Send or receive data

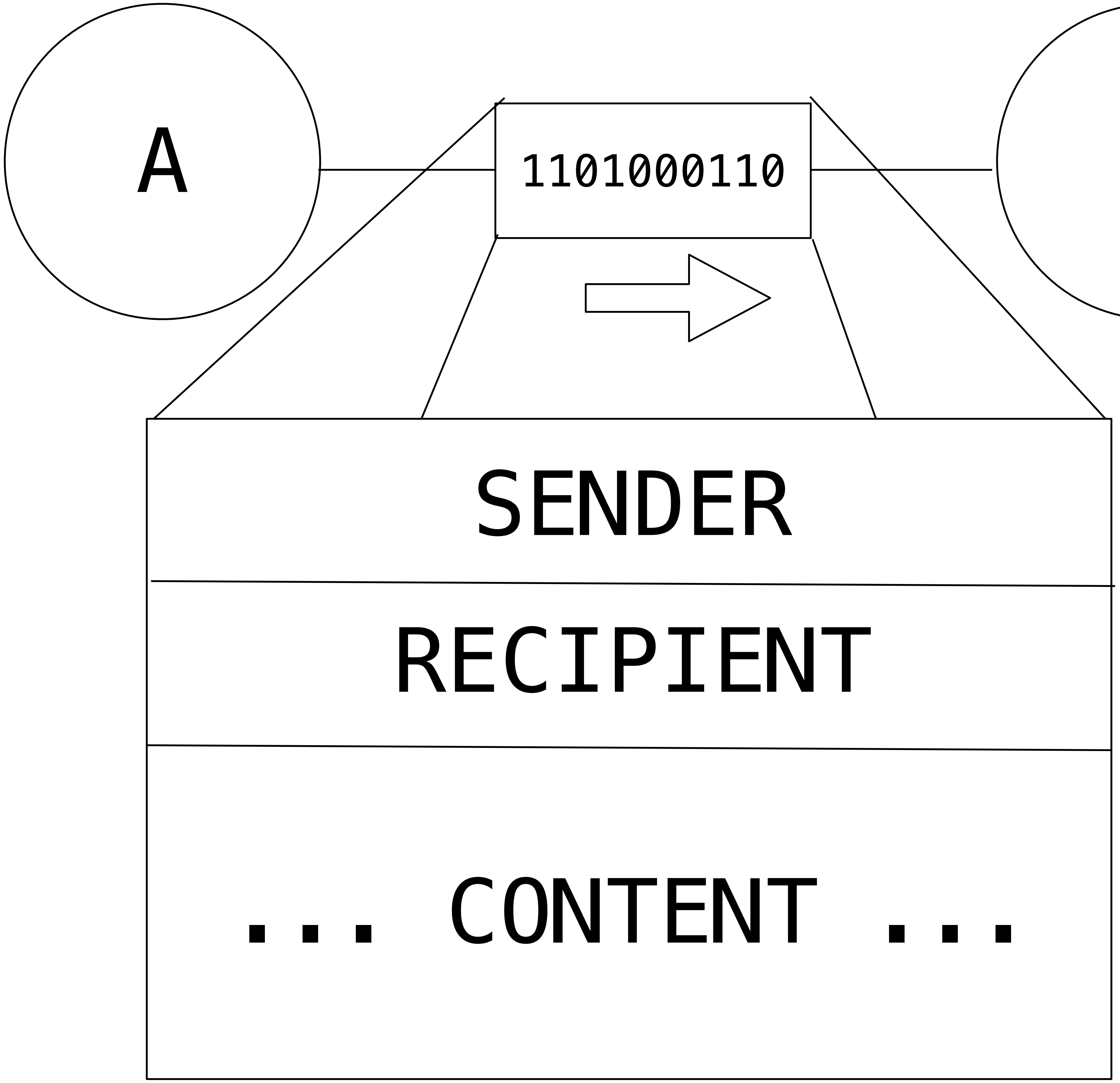
Request that a function be executed

Signal that a particular event has occurred

# Message Structure



# Message Structure



SENDER

RECIPIENT

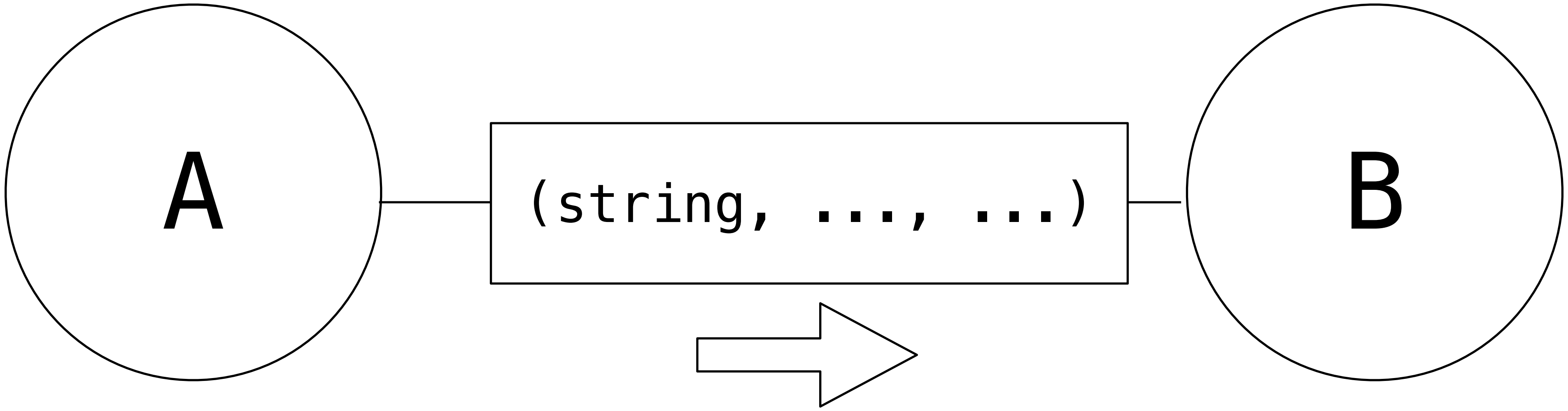
... CONTENT ...

So recipient knows where to send response

So message can be routed properly.

Data, remote procedure call, signal, encoded video, text, etc.

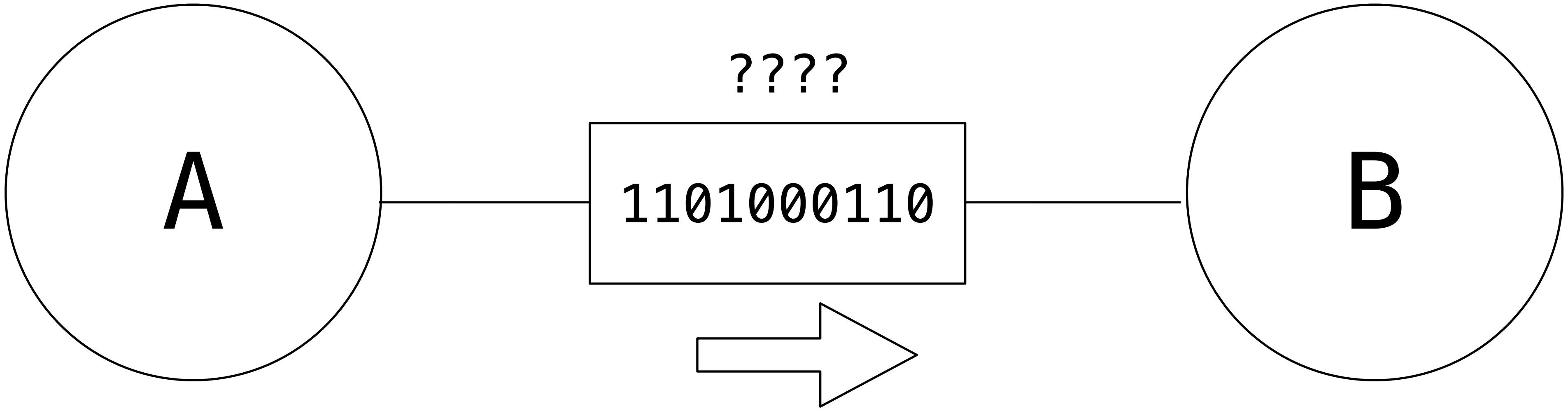
# Within a program



Dispatch procedure

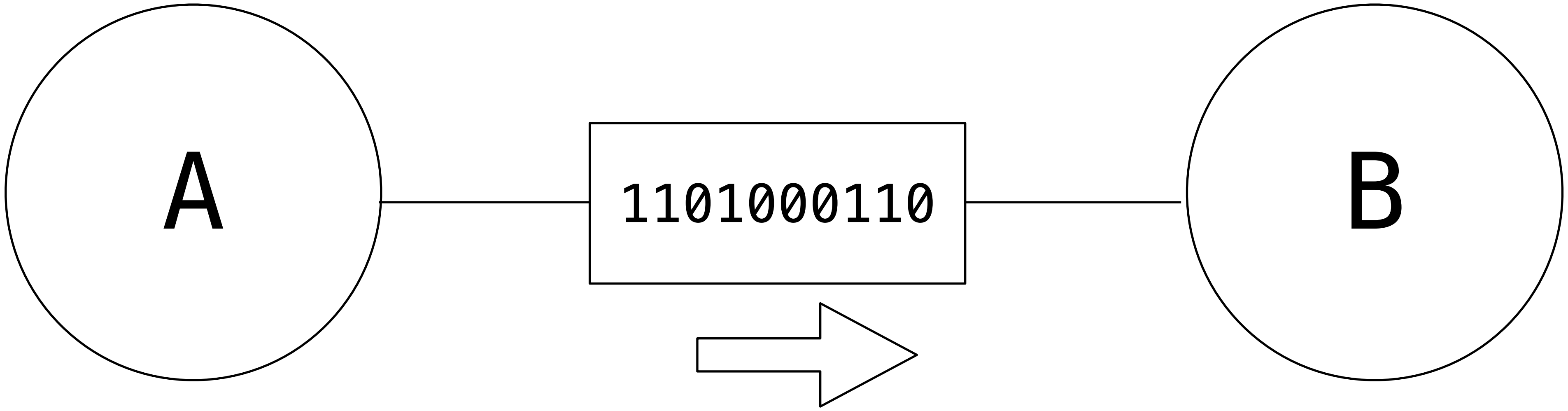
Sender and recipient implicit

Data sent using underlying data structures.



Over a network, there are no “underlying data structures”!

# Message protocol



A set of rules for encoding and decoding messages.

All computers in the system must obey the protocol when sending and receiving messages.

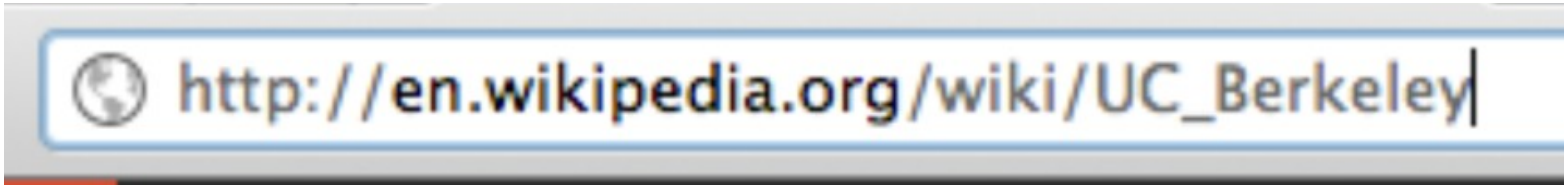
## Example

The first 3 bytes are the sender

The next 3 bytes are the recipient

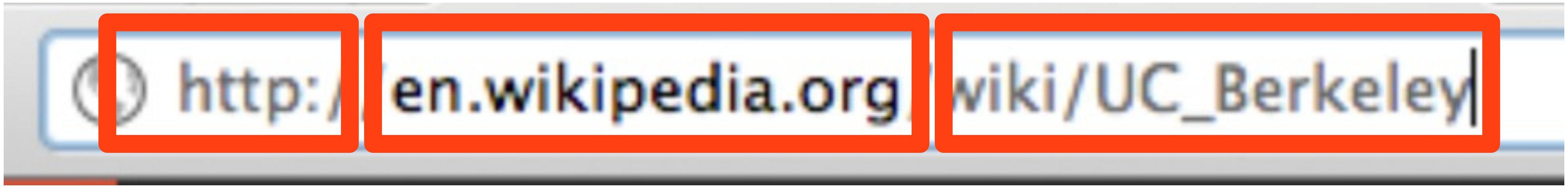
After that is the content, which is video, encoded according to... etc. etc.

# Example: messages on the world wide web



http://en.wikipedia.org/wiki/UC_Berkeley

# Example: messages on the world wide web



http://en.wikipedia.org/wiki/UC_Berkeley

Protocol name Server Requested page

Your IP address

en.wikipedia.org

GET wiki/UC_Berkeley HTTP 1.1

# Modularity

The components of a system should be black boxes with respect to each other.

The black boxes are required to hold up interfaces.

## Dispatch procedures

Interface =

List of messages that can be taken in

Responses that should be given to each message

## General Systems

Interface =

List of inputs that can be taken in

Outputs that should be given in response to inputs.

# Advantages of modularity

Easy to understand

=> Easy to change and expand

If something goes wrong, only defective component needs to be replaced

Easy to debug

- Compare real outputs to the supposed interface
- Defective component is the one that doesn’t hold up the interface any longer.