Communication of interacting processe

Communication of interacting processes

- One-way data flow
- Request/reply - client-server
- Heartbeat
- Probe/echo
- Broadcast/multicast/anycast/geocast
- Token passing
- Decentralized servers
- Replicated workers

One-way data flow

a special case and subtype of the Data-Flow Networks Family

So, **What is Data Flow Network?!**

Data Flow Network

- System components operate on large continuously available data stream
- Components are organized in an Arbitrary Topology
- Stream Data with Non-Transforming Connectors.

Data Flow Network

What happens when restrictions are placed on the topologies?

Answer: Subtypes

Subtypes:

1. Acyclic
2. Fanout
3. Pipeline
4. Unix Pipes and Filters

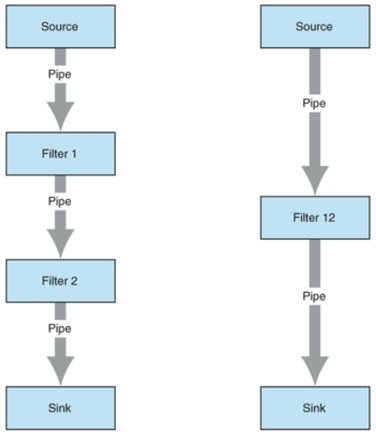
Table 1: Specializations of the dataflow network style

Style	Constituent parts		Control issues			Data issues				Ctrl/data interaction	
	Comp-onents	Conn-ectors	Topo-logy	Synch-ronicity	Bind-ing time	Topo-logy	Contin-uity	Mode	Bind-ing time	Isomor-phic shapes	Flow dir-ections
Data flow styles: Styles dominated by motion of data through the system, with no "upstream" content control by recipient											
Dataflow network [B+88]	trans-ducers	data stream	arbi-trary	asynch	i, r	arbi-trary	cont lvol or hvol	passed	i, r	yes	same
• Acyclic [A+95]			acyclic			acyclic					
• Fanout [A+95]			hier-archy			hier-archy					
• Pipeline [DG90, Se88, A+95]			linear			linear					
-Unix pipes and filters [Ba86a]		ascii stream			i				i		

Key to column entries

Synchronicity	asynch (asynchronous)
Binding time	i (invocation-time), r (run-time)
Continuity	cont (continuous), hvol (high-volume), lvol (low-volume)

One-way data flow



A piece of data enters the system and makes its way through a series of transformations, each transform accomplished by a separate process.

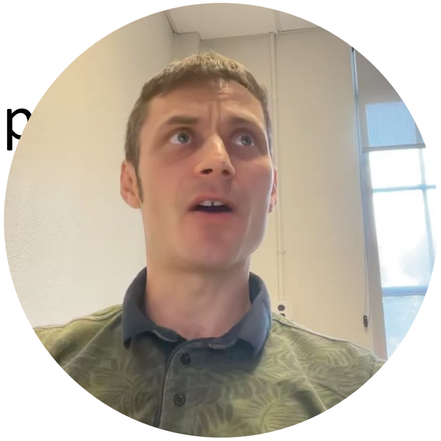
The **series need not be linear**; Andrews gives an example of a **tree of processes forming a sorting network**.

To cast it as a **specialization of dataflow networks, we**

(a) restrict its data and control topologies to one-way flows, and

(b) relax its data handling requirements from continuous to sporadic.

To cast it as a specialization of communicating processes we restrict its top
from **arbitrary to one-way** and its **synchronicity to asynchronous.**



One-way data flow

Example: React Libraries

Why is "one-way data flow" always listed in the "best practices" guides?

Think of Login Form which uses a form, username, password, and submit button.

You could implement this with one way data flow, and two ways data flow.

But, why not two-ways?

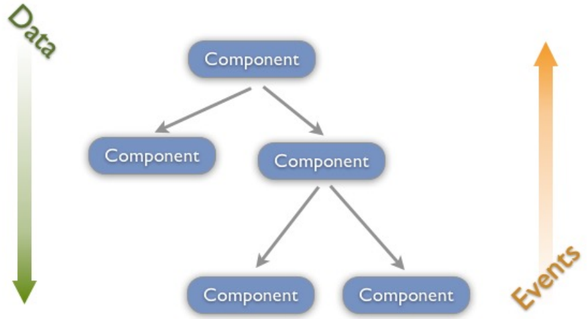
The short answer is that the cost of maintenance increases a lot.

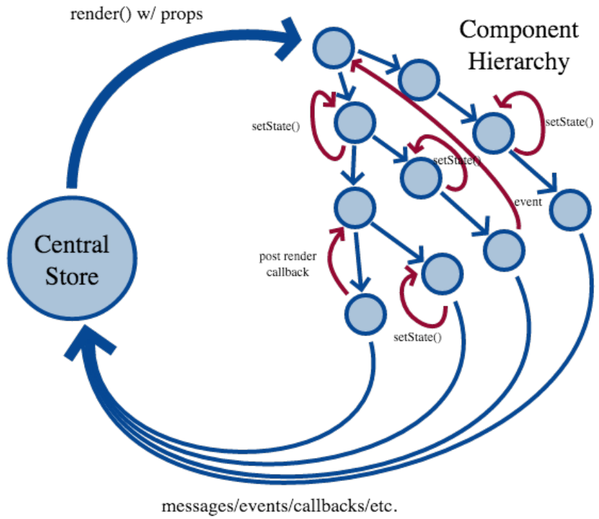
One-way data flow

React

Example: React Libraries

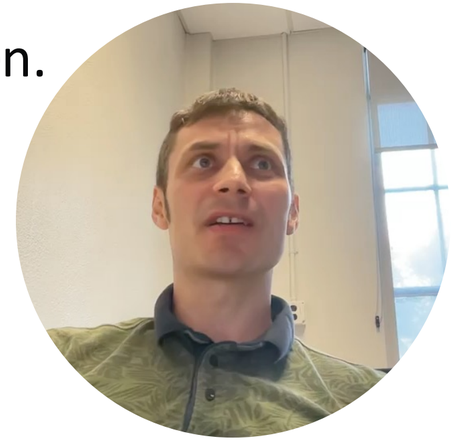
Why is "one-way data flow" Way going down on a component hierarchy





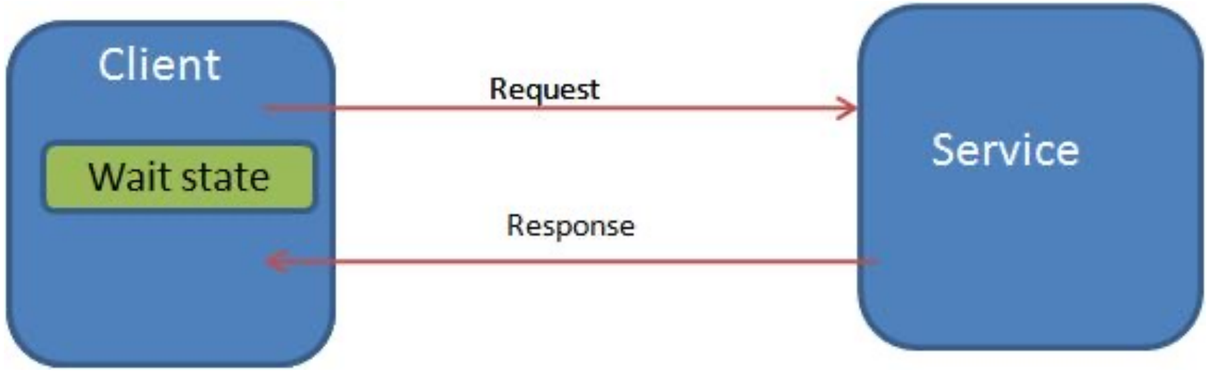
So then, why one-way is better in certain cases:

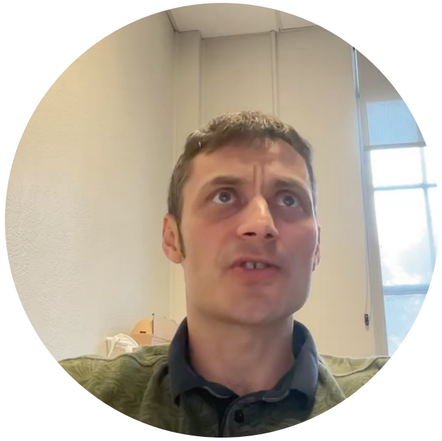
- Because is easier to maintain, understand/read/review, and so on. Basically because is in line with KISS (Keep it simple and stupid).
- One-way encourages the developers to keep their components simple,
  - following certain rules about state management.
- State should travel downwards (from parent component to children).
- State should be updated by the parent itself, reacting to events of its children.
- Performance issues in Two-Ways data bindings. Ex, Angular.
  - Keeping track of the data flow was arguably an even bigger problem



Request/reply - client-server

- The client–server model of computing is a distributed application structure
- Partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.





Request/reply - client-server

The client-server paradigm: The passive listener is the server, and the active initiator is the client.

(1) The server is the process that waits, offering a service.
(2) The client requests service but also performs its own computation.
(3) Large amounts of information can pass between client and server in both directions.
(4) Both client and server use a transport service like TCP/IP to communicate

Request/reply - client-server

To understand the communication between client and server, we need to know some simple topics:

Requests: are sent from the client in order to ask the server for some data like files or tell the server about things that happen.

Response: is sent from the server to the client and is the reaction of the server to a request of the client.

Service: is a specific task that the server provides for the client to use.

Request/reply - client-server

A thread for each request:

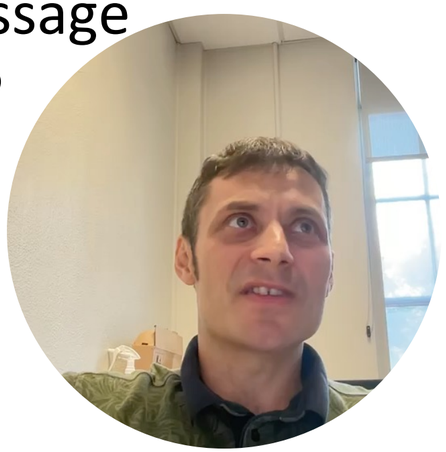
(1) In order to handle concurrency, the server-class computer uses “threads”
(2) Certain service associated with well-known ports.
(3) The main thread or listener is located at this port, as the request is made, a new thread is spun off for it.
(4) This allows the main listening thread to await the next client.

Request/reply - client-server

Types of transport protocols: the application services may be connection-oriented or connectionless.

(1) Connection-oriented: the applications must first establish the connection and then send the data across the connection. TCP requests the connection and once established the communication begins.

(2) Connectionless: The application sends a message to the destination at any time. The sending application needs to specify the destination with each message sends. The UDP is the connectionless support mechanism in the TCP/IP protocol.



Request/reply - client-server

Characteristics of clients and server.

The typical client:

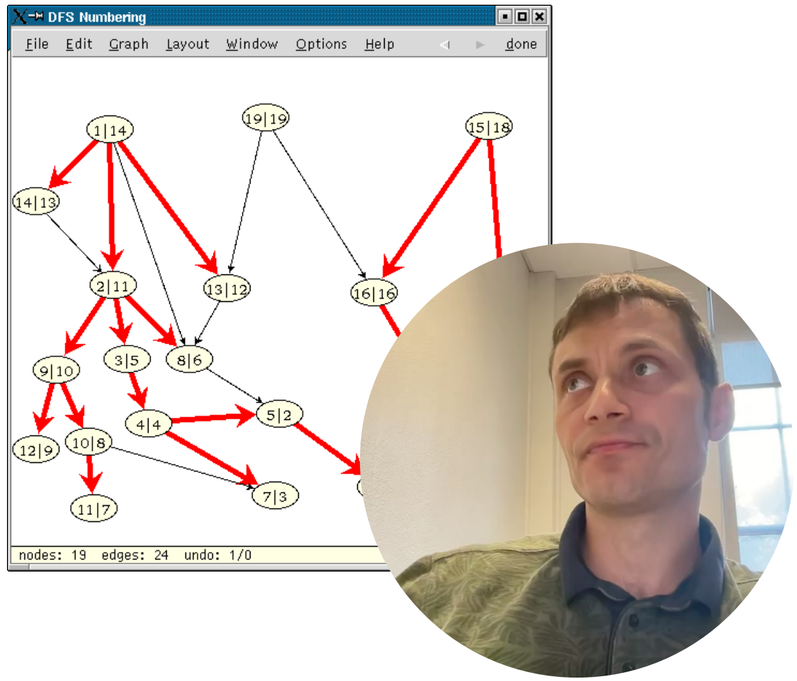
(1) Arbitrary process, temporary client.
(2) Invoked by user
(3) Runs on the user's PC
(4) Initiate contact with server
(5) Contact one server at a time.

The typical server:

(1) Dedicated to providing one service to multiple clients concurrently.
(2) Executes continually
(3) Waits passively for client conta
(4) Requires powerful hardware
sophisticated operating syst

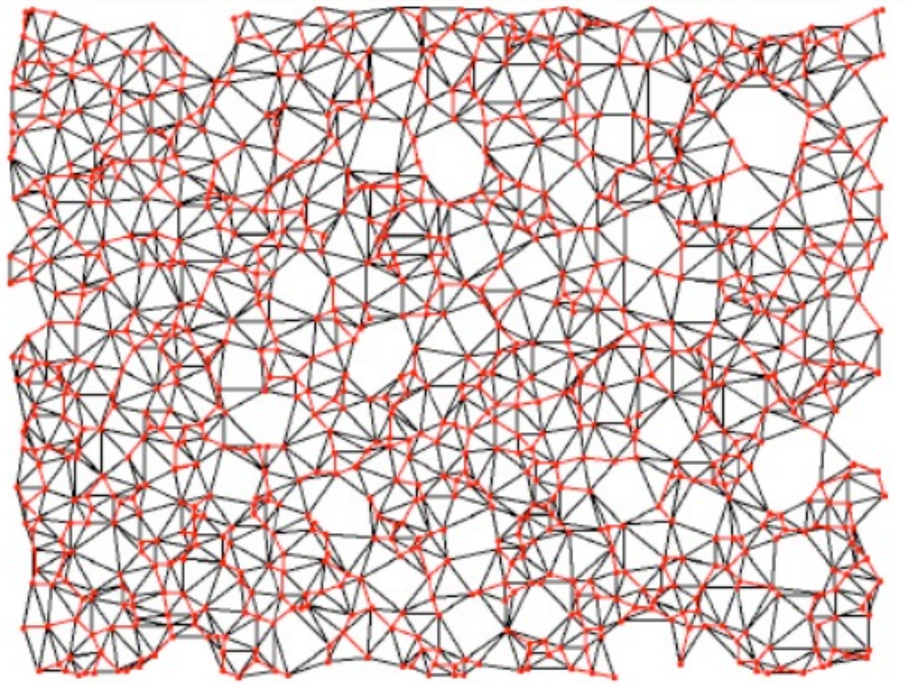
Probe a echo

- Gathering or distribution of data in graphs or trees
- No iterations
  - One node collects the entire topology from all other nodes by visiting them
  - Initiates a broadcast probe on all neighbors
  - Forwarding the probe to neighbors
  - Echo is the response to all probes
  - In the end, a broadcast of information to all nodes
- Parallel processing analogical to
  - Depth-First Search
  - Network topology discovery
  - Broadcast through neighbors



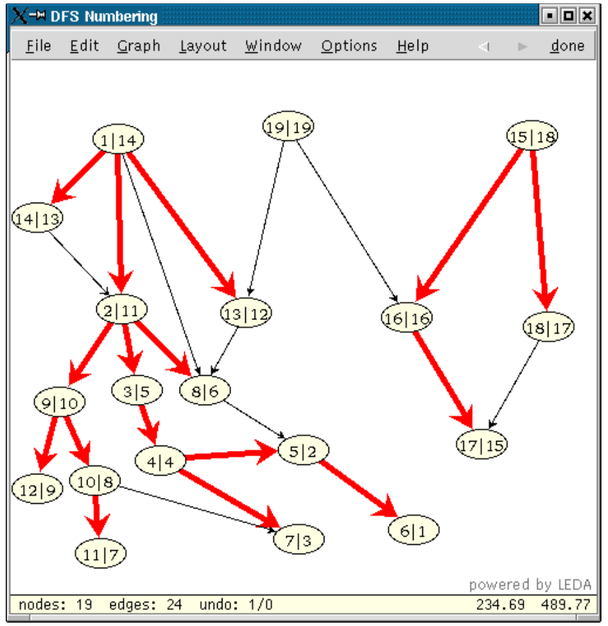
Probe a echo

- No information sharing – just messages receipt and submission
- Nodes send **probes to** their **neighbors** and **receive echoes** from them.

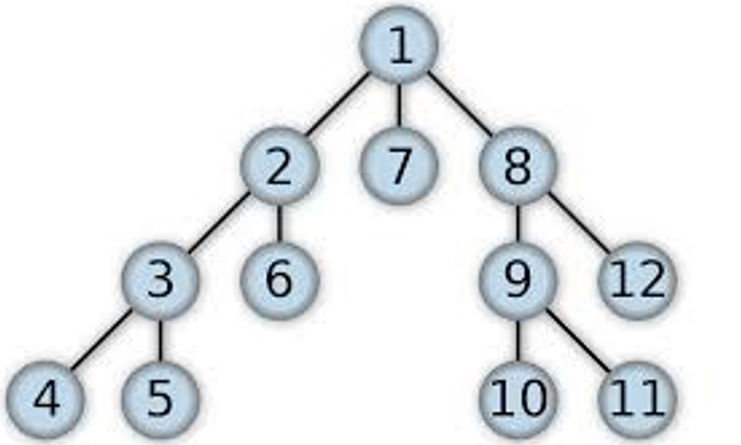


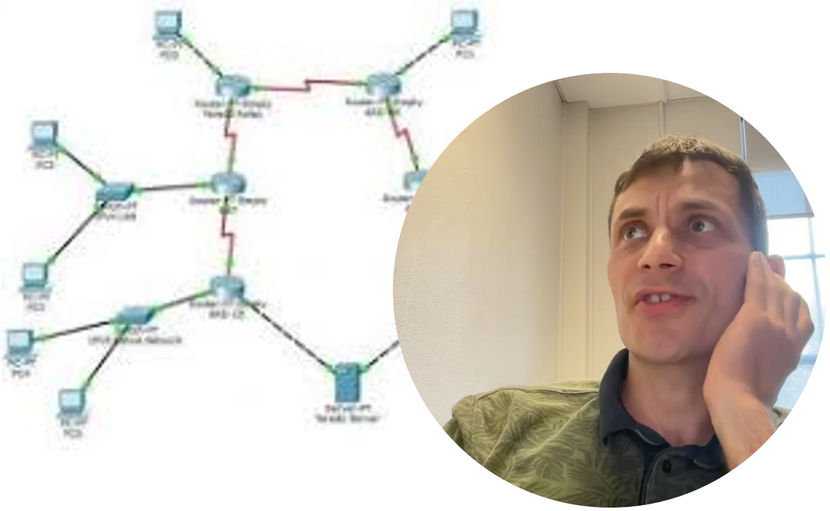
Probe/Echo

"How do I fill in the gaps of an incomplete graph?"



- Depth-First Search
- Network Topologies
- Broadcasting





What is a Probe?

- From process to successors
- Direct connections only

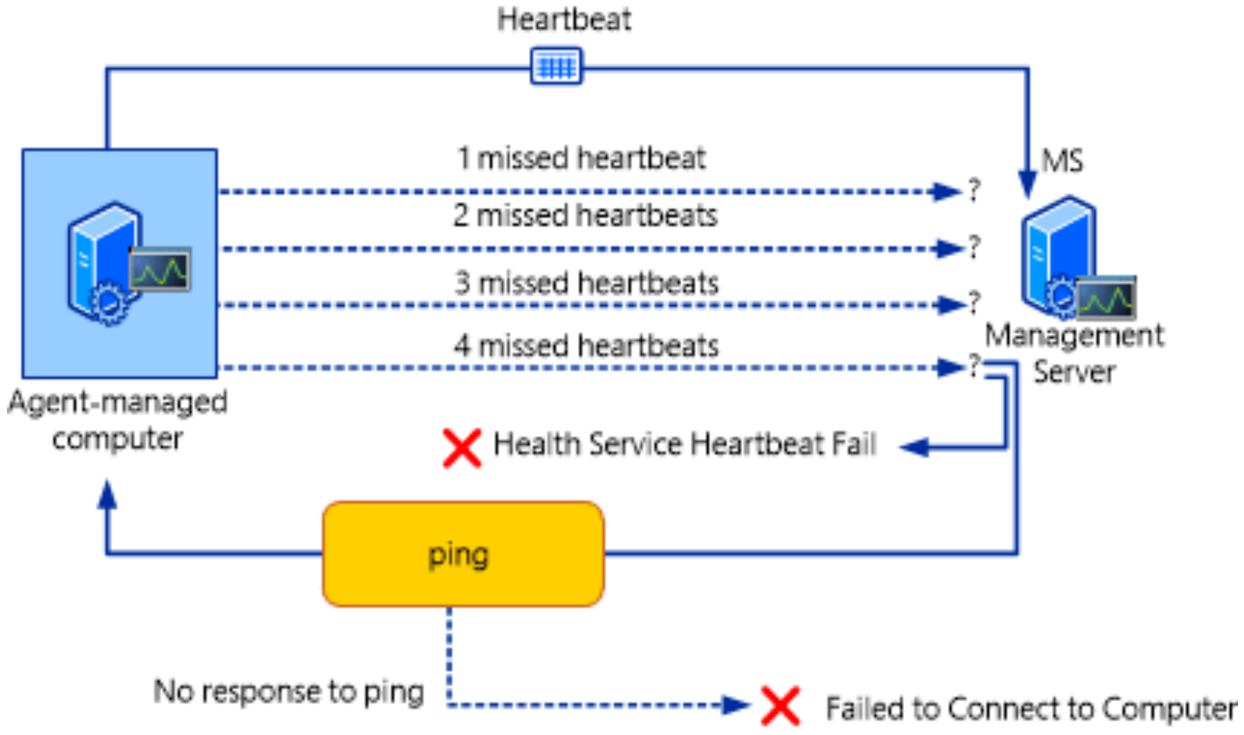
What is an Echo?

- Reply from probe
- Ping

Heartbeat = there and back : neighbor interaction

- In computer clusters, heartbeat network is a private network, which is shared only by the cluster nodes, and is not accessible from outside the cluster.
- It is used by cluster nodes in order to monitor each node's status and communicate with each other.
- Heartbeat is a periodic message sent to a central monitoring server other servers in the system to show that it is alive and functioning

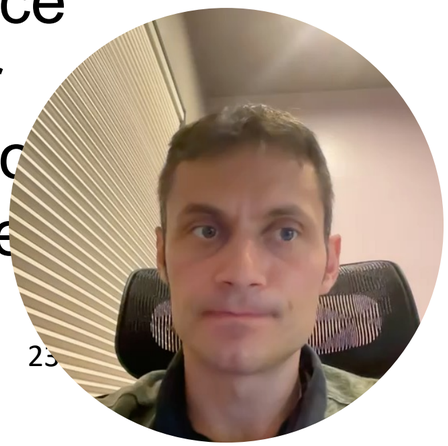
Heartbeat



- Availability Tactic.
- Fault Detection Mechanism.
- Discover Topology of

Heartbeat = MS System Center 2012

- Operations Manager uses heartbeats to **monitor communication channels** between an agent and the agent's primary management server.
- A heartbeat **is a packet of data sent** from the agent to the management server on a regular basis, by default **every 60 seconds**, using port 5723 (UDP).
- **When** an **agent fails to send a heartbeat 4 times**, a Health Service Heartbeat Failure **alert** is **generated** and **the management** server **attempts** to contact the computer by using **ping**. If the computer do **respond** to the ping, a Failed to Connect to Computer **alert** is gene



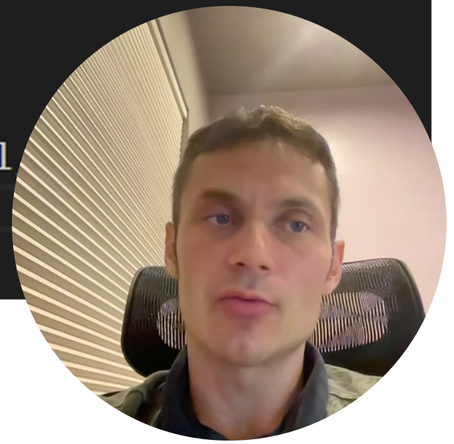
Heartbeat

```
class HeartBeatScheduler…

  public class HeartBeatScheduler implements Logging {
      private ScheduledThreadPoolExecutor executor = new ScheduledThreadPoolExecutor(1);

      private Runnable action;
      private Long heartBeatInterval;
      public HeartBeatScheduler(Runnable action, Long heartBeatIntervalMs) {
          this.action = action;
          this.heartBeatInterval = heartBeatIntervalMs;
      }

      private ScheduledFuture<?> scheduledTask;
      public void start() {
          scheduledTask = executor.scheduleWithFixedDelay(new HeartBeatTask(action), heartBeatInterval
          TimeUnit.MILLISECONDS);
      }
```



Heartbeat

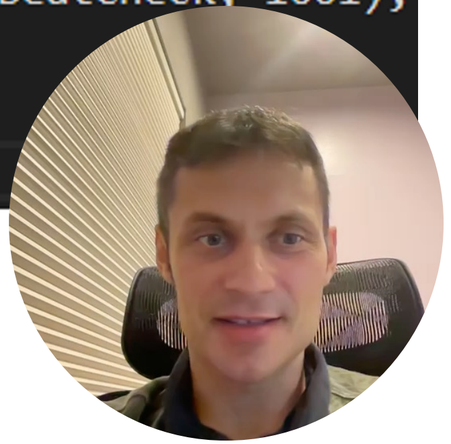
```
class SendingServer…

  private void sendHeartbeat() throws IOException {
      socketChannel.blockingSend(newHeartbeatRequest(serverId));
  }
```

```
class AbstractFailureDetector…

  private HeartBeatScheduler heartbeatScheduler = new HeartBeatScheduler(this::heartBeatCheck, 100l);

  abstract void heartBeatCheck();
  abstract void heartBeatReceived(T serverId);
```



Heartbeat

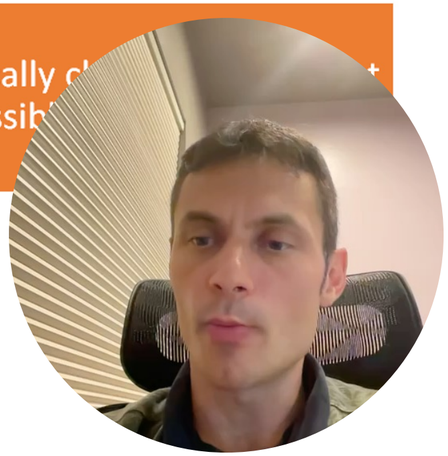
```
class ReceivingServer…

  private void handleRequest(Message<RequestOrResponse> request) {
      RequestOrResponse clientRequest = request.getRequest();
      if (isHeartbeatRequest(clientRequest)) {
          HeartbeatRequest heartbeatRequest = JsonSerDes.deserialize(
                                              clientRequest.getMessageBodyJson(),
                                              HeartbeatRequest.class);

          failureDetector.heartBeatReceived(heartbeatRequest.getServerId());
          sendResponse(request);
      } else {
          //processes other requests
      }
  }
```

A method to be called whenever the receiving server receives the heartbeat, to tell the failure detector that heartbeat is received

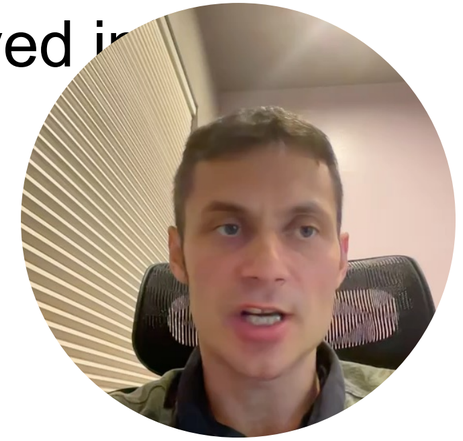
A method to periodically c… t status and detect possib…



Heartbeat : Distributed collection : Gossip

- Each node in the process graph does
  - Send information
  - Collect new information
  - Calculates matrix of neighbor nodes

- *Example. Discovery of network topology*
- *Each iteration*
  1. With each „beat,“ each node sends neighbors its matrix of neighbor nodes
  2. Between each heartbeat, calculate new matrix values using received i
  3. After „n“ iterations we know the topology of distance „n“
  4. Termination once we reach closing conditions
  - Network diameter, strict n, unchanged values



REDIS

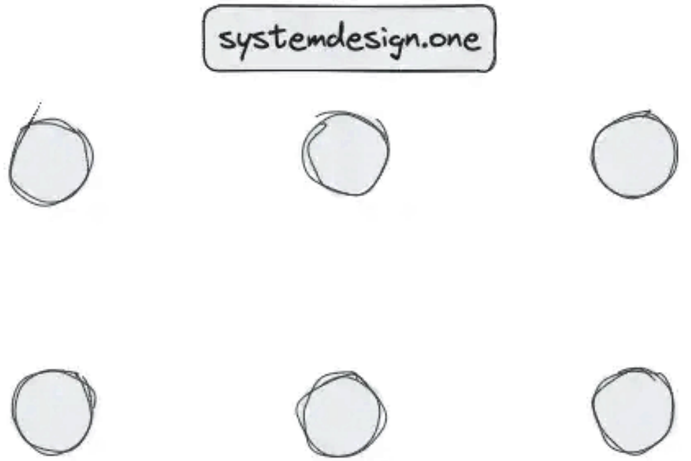
- http://cristian.regolo.cc/2015/09/05/life-in-a-redis-cluster.html

- Automatic data sharding across multiple nodes
- High degree of fault tolerance

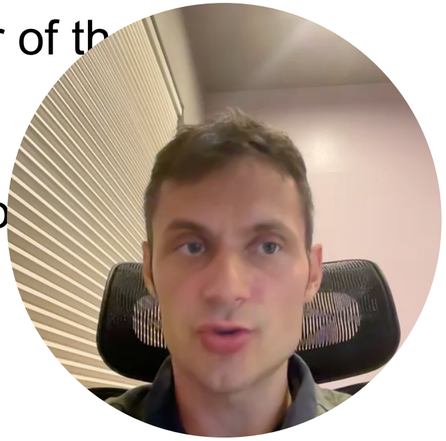
- Nodes in a Redis Cluster talk to each other using the Cluster Bus

- Nodes of the cluster exchange PING & PONG packets over the bus
- This packet flow constitutes the heartbeat mechanism of the cluster
- Each ping/pong packet carries important pieces of information about the state of the cluster from the point of view of the sender node
  - https://youtu.be/0BLmM73y3oo
- Redis Cluster uses a **Gossip protocol** in order to **quickly spread information through any connected node**.
- Heartbeat packets carry information independently, but they also con a special header for gossip data.

Gossip

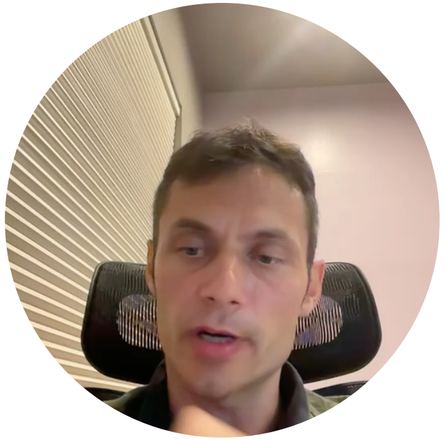


- Decentralized failure detection method.
- Process: Say we have 100 servers in a distributed environment
- 1. Each server stores the heartbeat of some (say:10 servers) from the pool of available servers. In our case, it‘s 100. Heartbeat is a counter incremented every unit of time, or simply it can be a timestamp.
- 2. Now, each server gossips. It shares its own heartbeat and the heartbeats of other servers; it has information about the other 10 random servers.
- 3. Other servers will compare the information they already have with the information provided. If already available information is most up-to-date, then they would inform the gossiping server of th latest one.
- 4. Now, say server 5 is at fault; it won't update its timestamp; when this stale timestamp is go by other servers (more than one), then server five will be marked as down.



Heartbeat vs Sonda a echo

- Heartbeat
  - Symmetric algorithm
  - Many messages
  - Parallel interaction
  - Resilient
- Probe-echo
  - Asymmetric algorithm
  - Fewer messages
  - More efficient
  - Let reliable

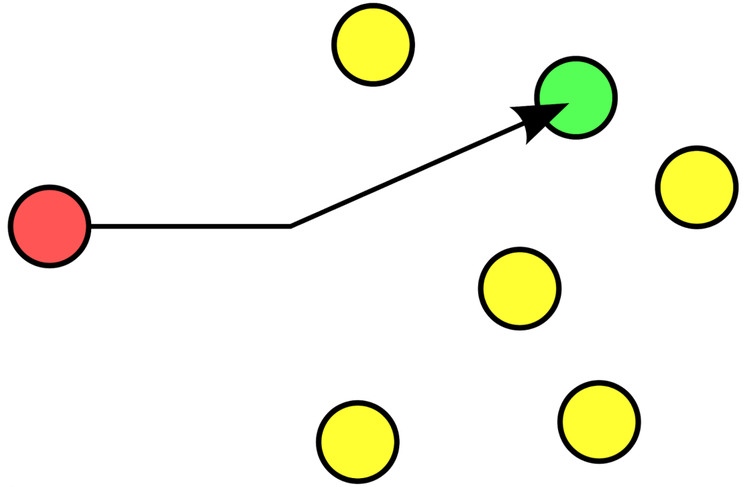


Broadcast/multicast/anycast/geocast

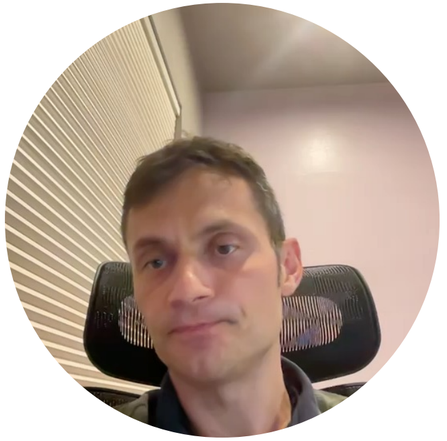
The **path** of communication and data transmission using any medium or any area network varies user to user.

How users are connected to the internet or computer networks is one of the main concepts of casting types.

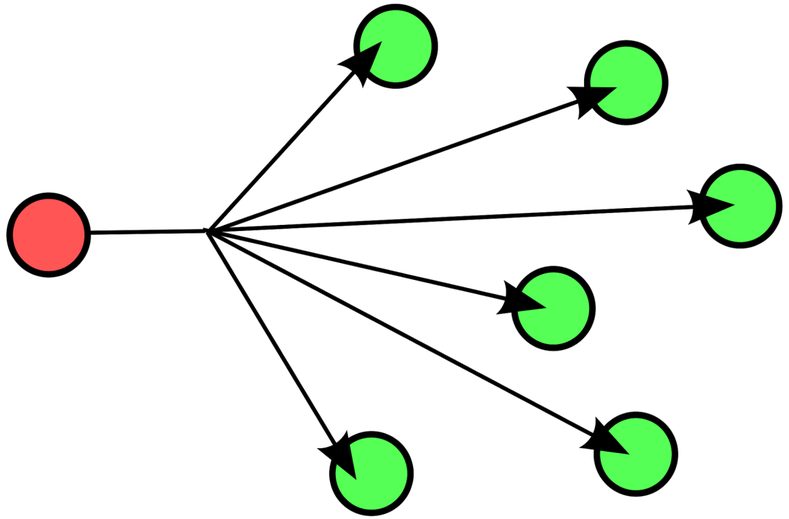
Unicast



- **Unicast** is the communication that there is only one receiver.
- This is **one-to-one** communication (one sender and one receiver).
- Only a single known recipient is addressed.
- Such as Transmission Control Protocol (**TCP**) in the Internet Protocol.

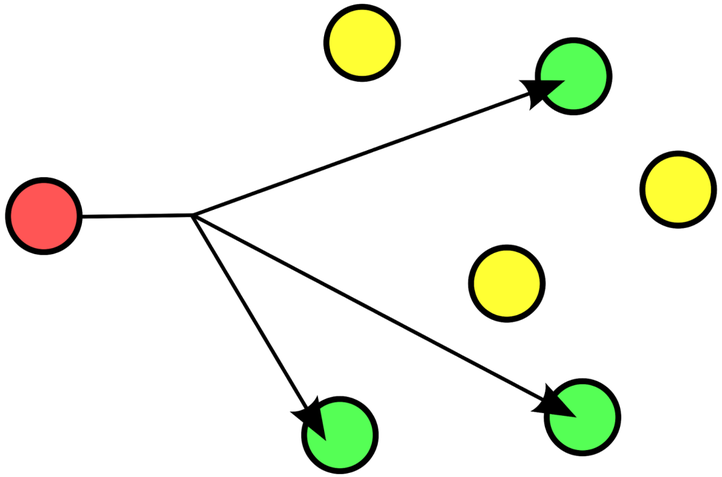


Broadcast

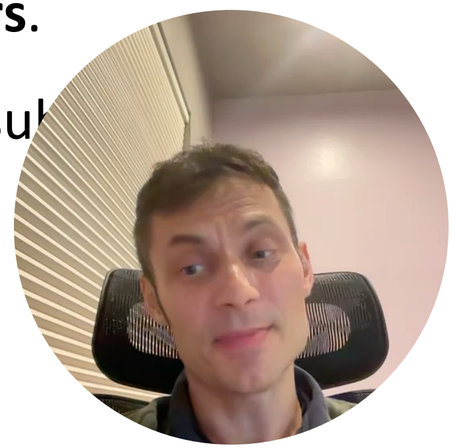


- Broadcasting is a method of transferring a message to all recipients simultaneously.
- This is **one-to-many** communication (one sender and many receivers).
- The broadcast differs from the unicast, such that the sender **does not** n
  indicate **recipient addresses.**

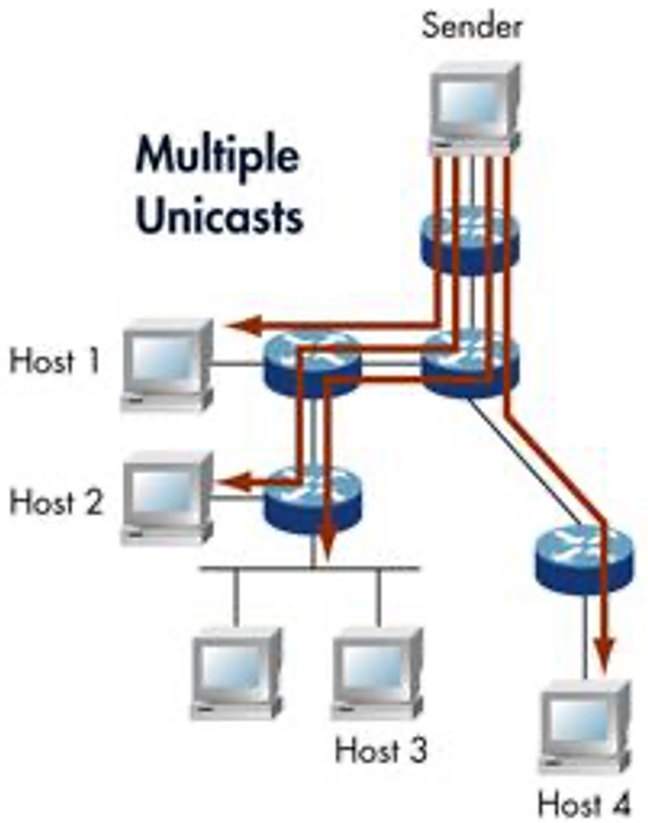
Multicast

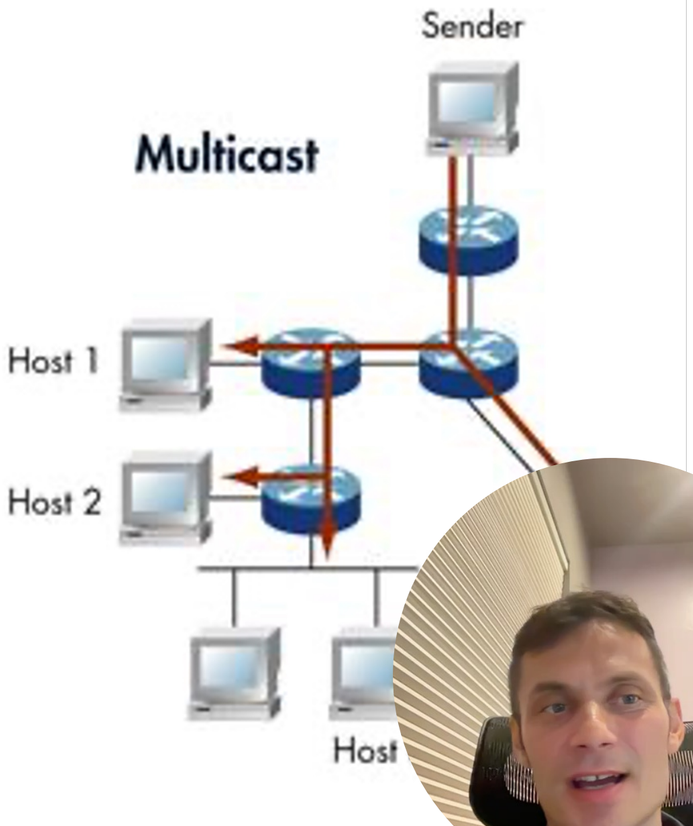


- **Multicast** is group communication where data transmission is addressed to a group of recipients simultaneously.
- This is **one-to-many-of-many** communication (one sender and many receivers).
- The system is capable of being scaled up to accommodate "**many-to-many-of-many**" type communications, with **multiple hosts transmitting to multiple receivers**.
- It differs from broadcast in that the destination address designates a sub **necessarily all**, of the accessible nodes.

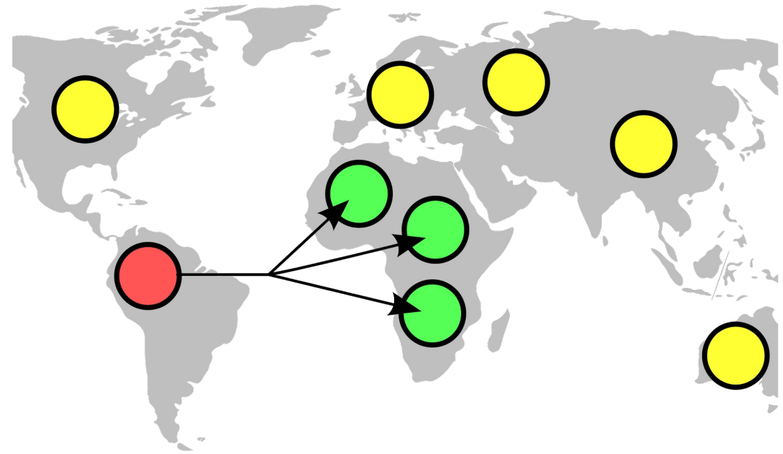


Multicast using Unicasts



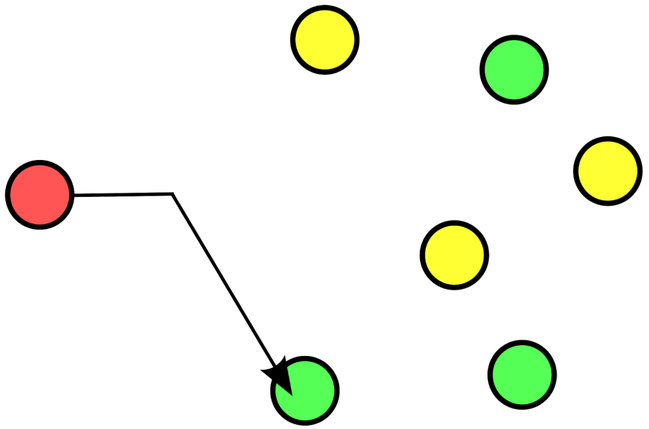


Geocast

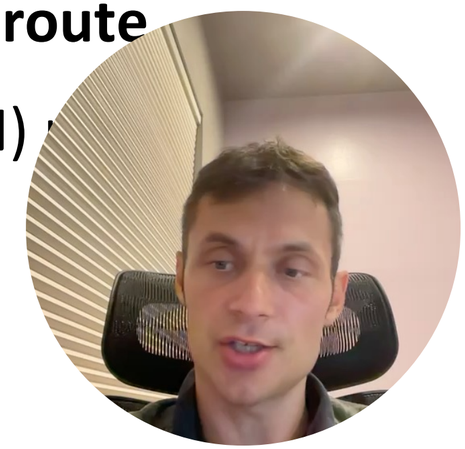


- It is a specialized form of **multicast**.
- **Geocast** refers to the delivery of information to a group of destinations identified by their geographical locations.

Anycast



- **Anycast** is a network addressing and routing methodology, in which a **single destination** address has **multiple routing paths** to two or more endpoint destinations.
- Routers will select the desired path on the basis of **number of hops**, **distance**, **lowest cost**, **latency measurements** or based on **the least congested route**.
- Anycast networks are widely used for **content delivery network** (CDN) to bring their content closer to the end user.

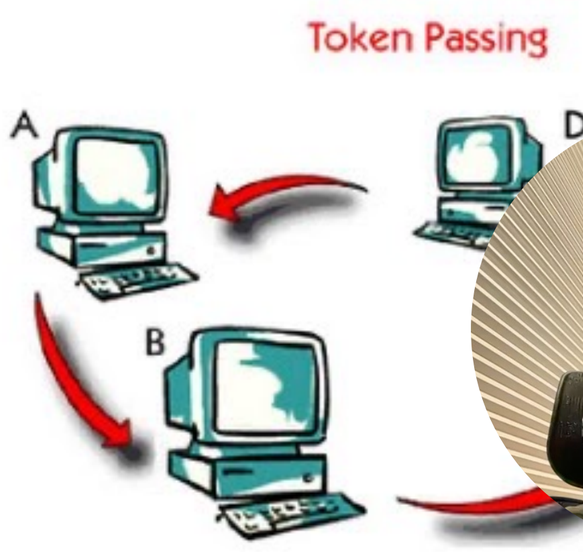


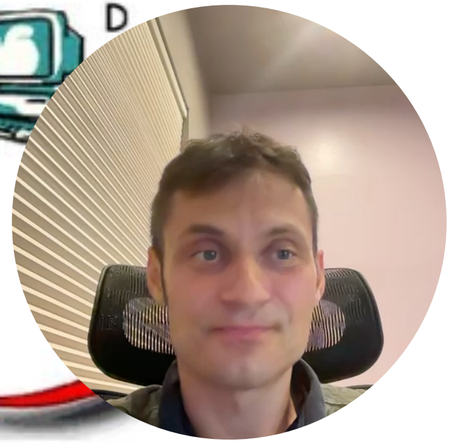
Token passing

In telecommunication, token passing is a channel access method where a signal called a **token** is passed between nodes that authorizes the node to communicate.

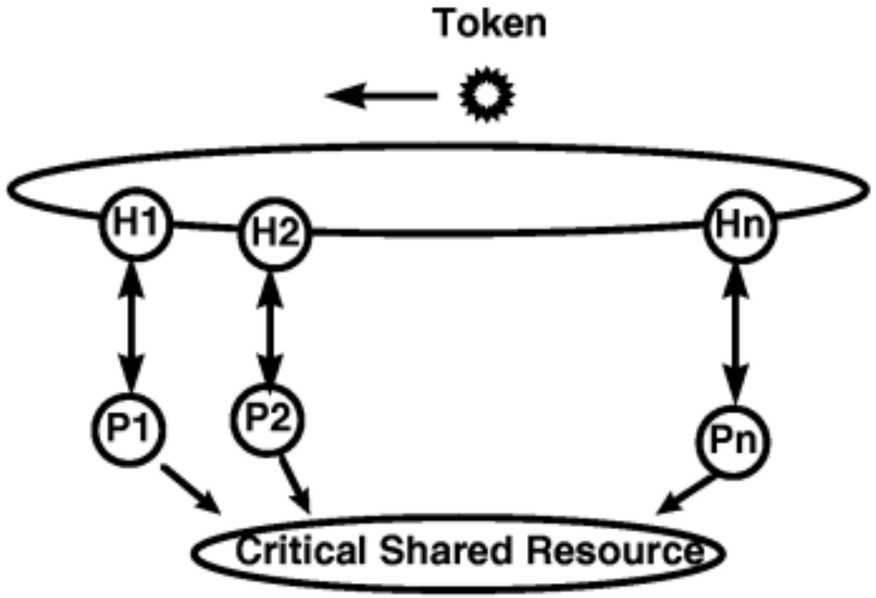
The most well-known examples are token ring and ARCNET.

- Token
  - Message Passing
    - Reliable vs Unreliable
    - Blocking vs Non-blocking
- Distributed coordination
  - Who has the token is authorized to access/control/perform.
  - Distributed State
- i.e. Global state of distributed system
- i.e. Distributed locks



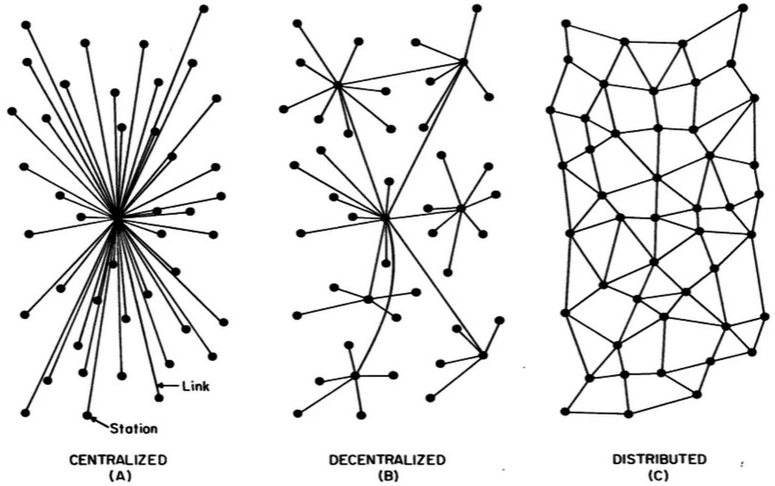


Token passing



- Detection of process termination
  - Tokens loop
  - A finished process produces and passes a token
  - Process receives a token, finishes tasks, and passes a token
  - If the process that passes a token receives the same token, we know the distributed assignment is finished.

Decentralized servers



Centralized:

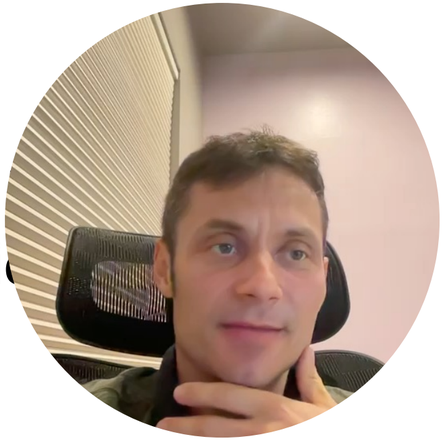
- In a centralized system, all users are connected to a central network owner or “server”.

Decentralized:

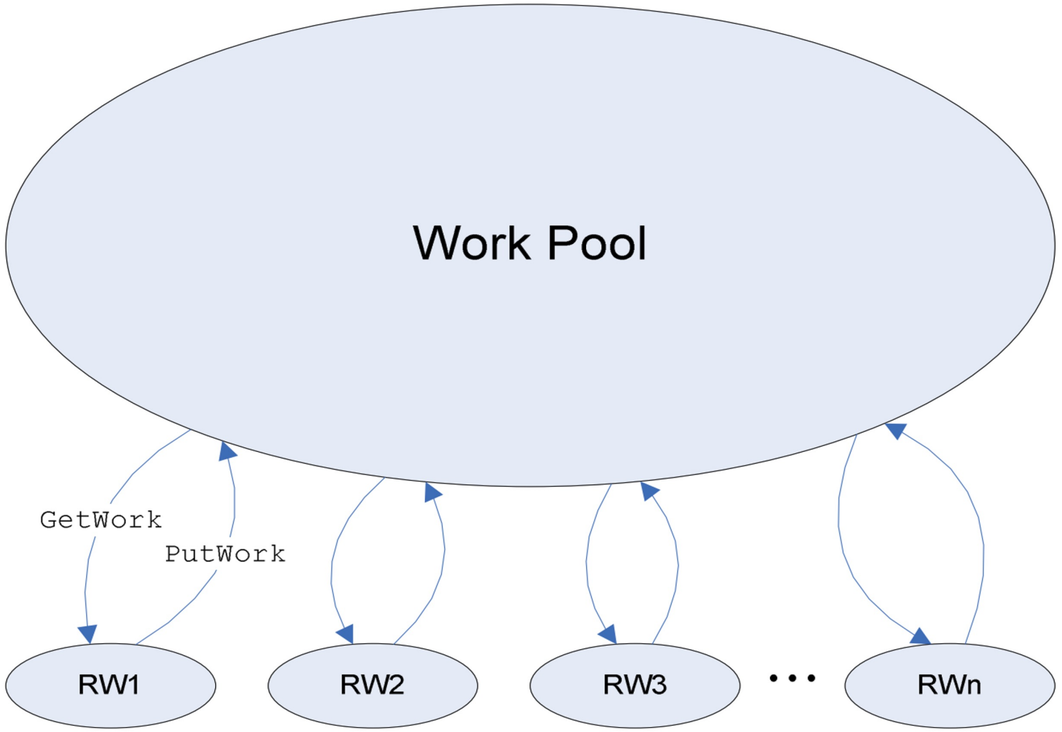
- Decentralized systems don’t have one central owner. Instead, they use multiple central owners, each of which usually stores a copy of the resources users can access.

Distributed:

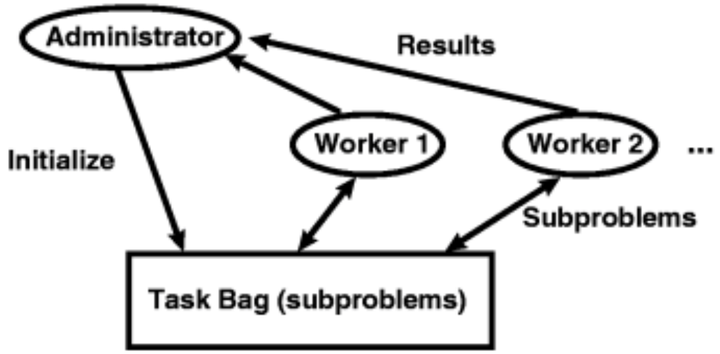
- A distributed system is similar to a decentralized one in that it doesn’t single central owner. But going a step further, it eliminates centralizati

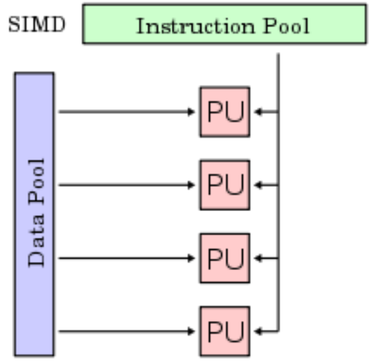


Replicated workers



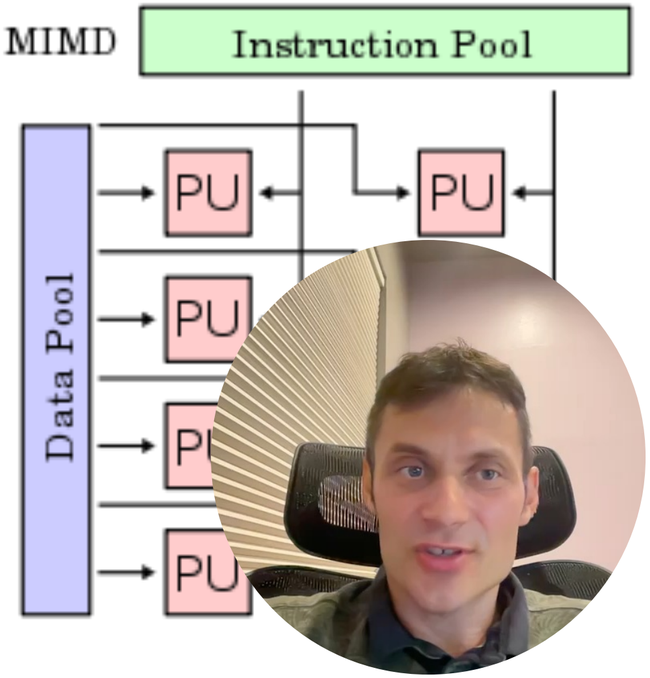
Replicated workers





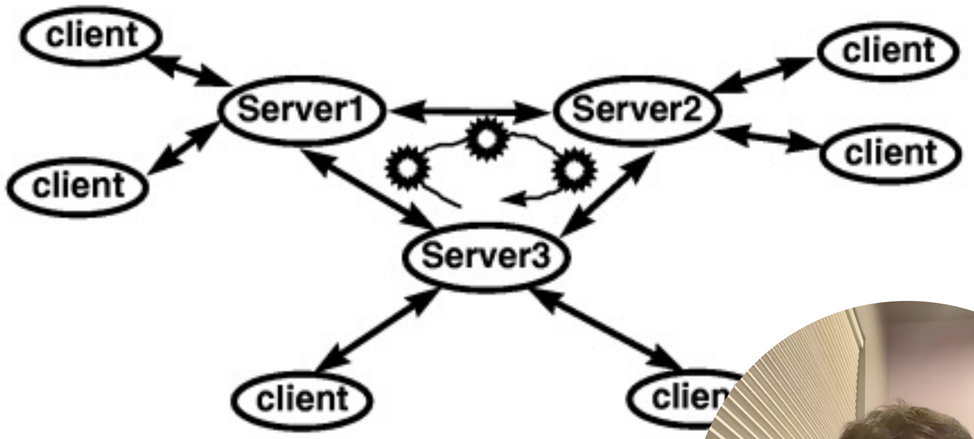
Definition: It is a distributed system architecture where different machines work as a part of the system and complete different sub-tasks of a big task.

- Parallel divide and conquer.
- Work pool.
- Multiple copies of computational elements.
- Managing tasks and communication between all workers.
- Sharding.



Replicated servers

- Replicated servers with identical functionality that interact
  - Resilience
  - Concurrency
  - Shared tasks
  - Parallel divide and conquer
- We can replicate identical data
  - Hard to change
- Availability of data

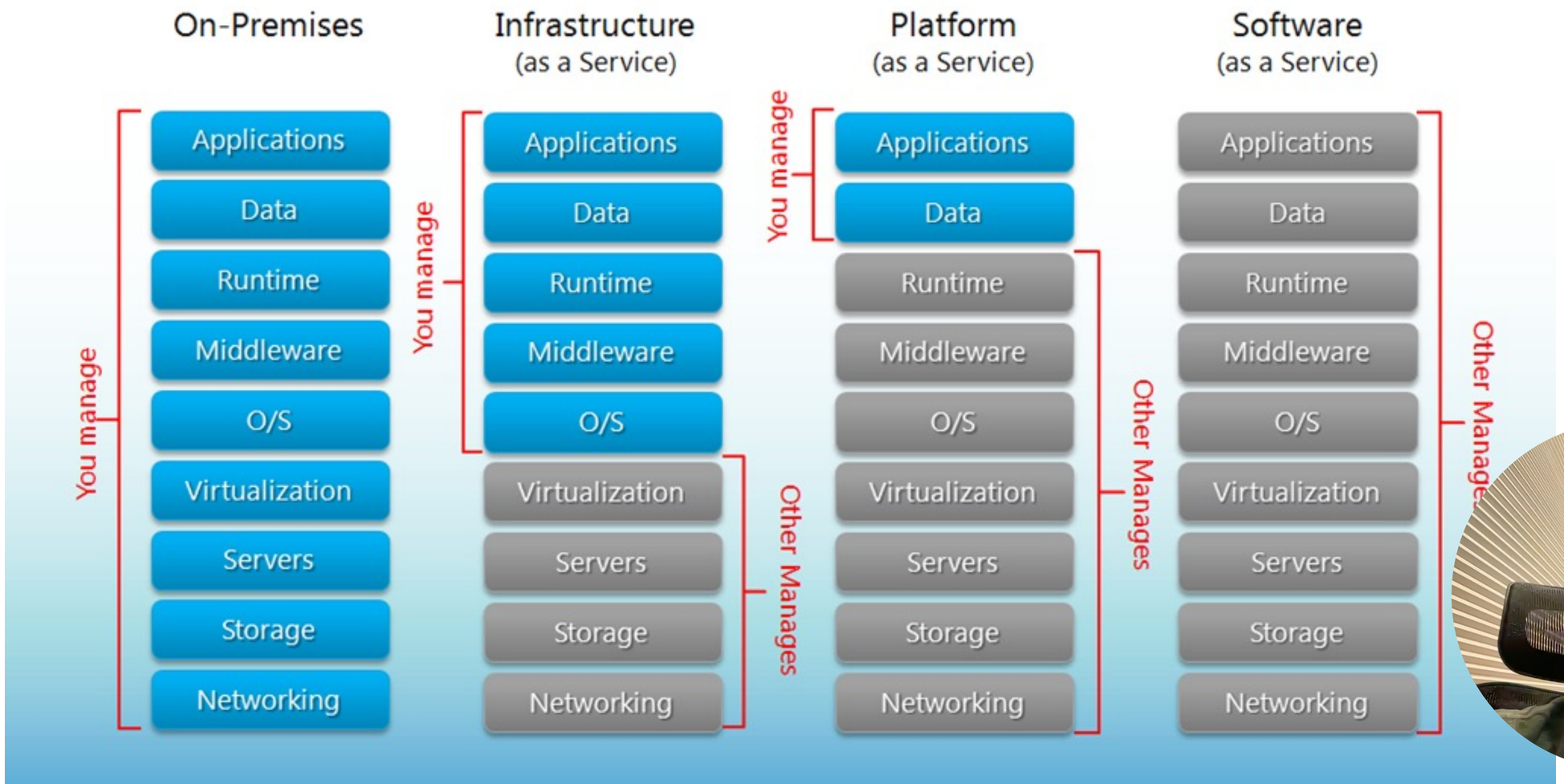


Cloud vs Cluster

- Platform as a service (PaaS) for cloud applications
  - Scalable cluster of services
- PaaS providers
  - Enable developers to deploy apps across clouds providing load balancing, scalability, high availability...
  - Without the huge costs of buying, managing, and maintaining their own environments.

IaaS / PaaS / SaaS

Separation of Responsibilities



Decentralization of servers vs replication of workers

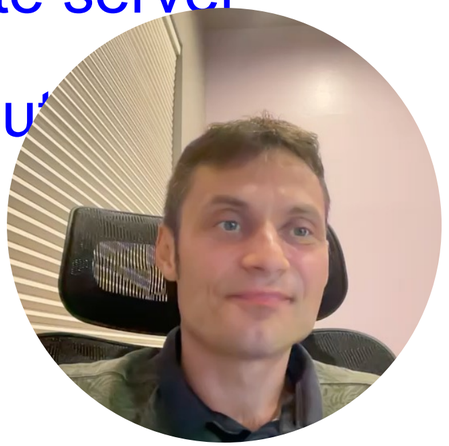
- **Replication**
  - Hierarchical and synchronous structure of control
  - Master node
    - Master-slave
    - Balancer
- **Decentralization**
  - Arbitrary topology of control, asynchronous
  - No master

Decentralized servers

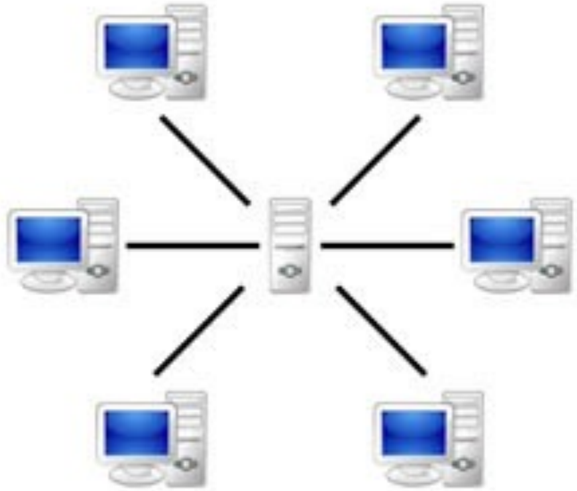
- **Cloud**
  - Service provided by a single vendor
  - Trust servers
  - Clients assume single endpoint interaction
- **Peer to peer**
  - Multiple vendors
    - Usually not certified
  - No trust at all
  - Dynamically changing environment

Decentralized servers

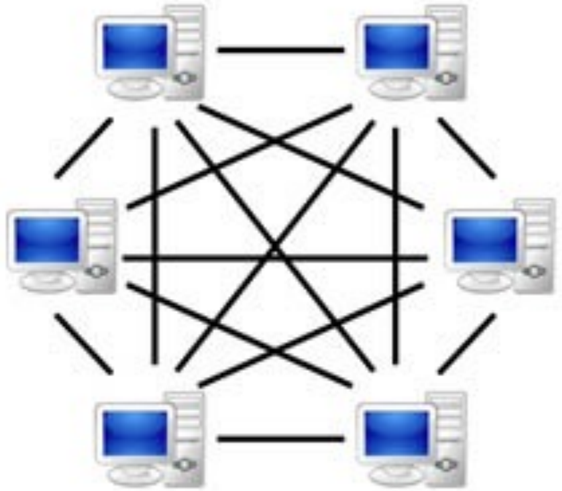
- **Cloud computing**
  - The practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer.
    - PaaS
- **Peer to peer**
  - In its simplest form, a P2P network is created when two or more PCs are connected and share resources without going through a separate server computer.
  - A P2P network can be an ad hoc connection - a couple of comput connected via a Universal Serial Bus to transfer files.



Decentralized servers



Server-based



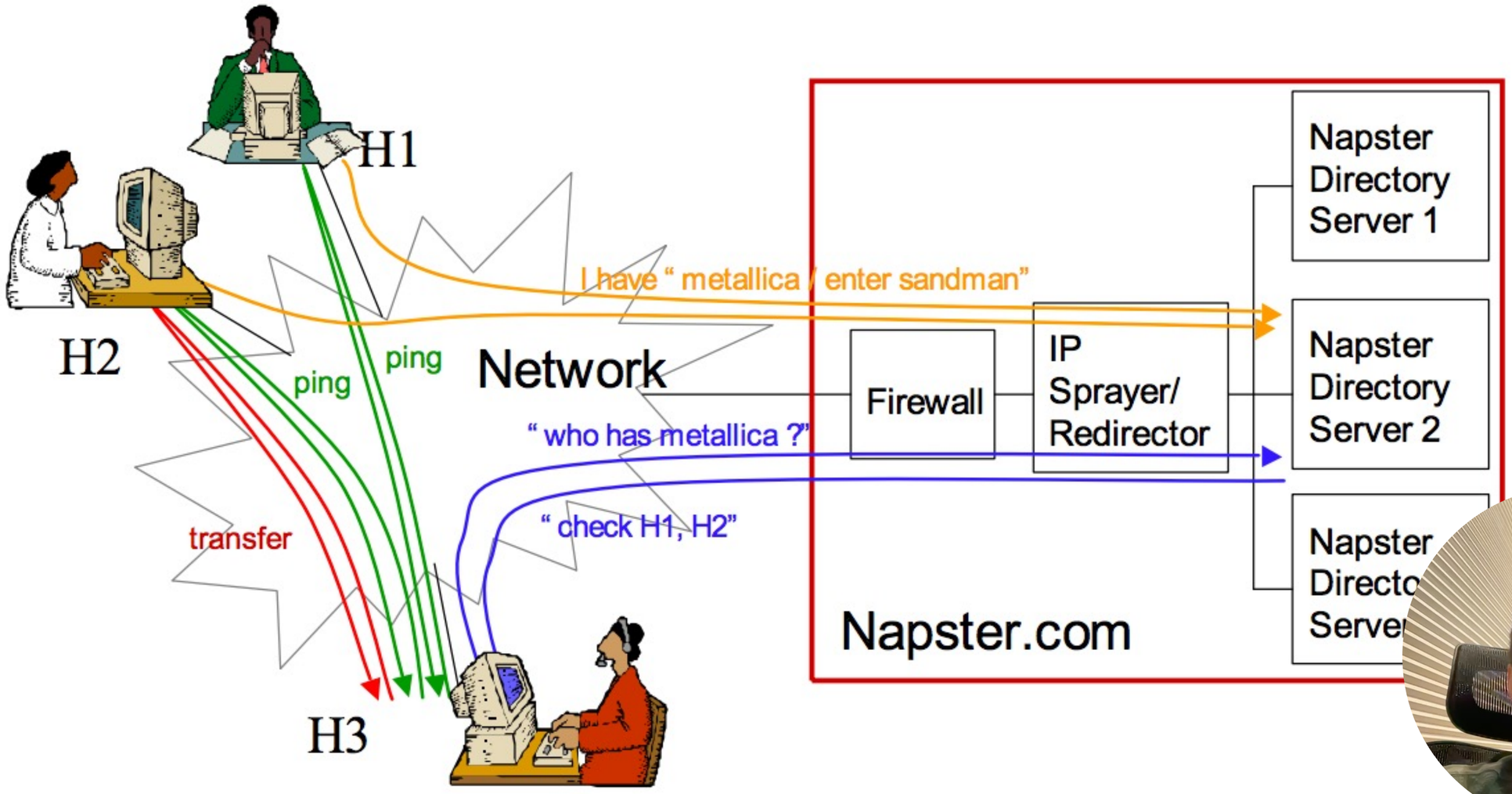
P2P-network

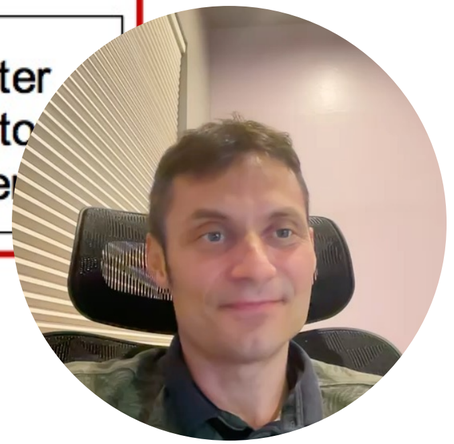
- Peer-to-peer

- Each node has properties of a client-server
  - - autonomous control
- Who can we trust?

Decentralized servers

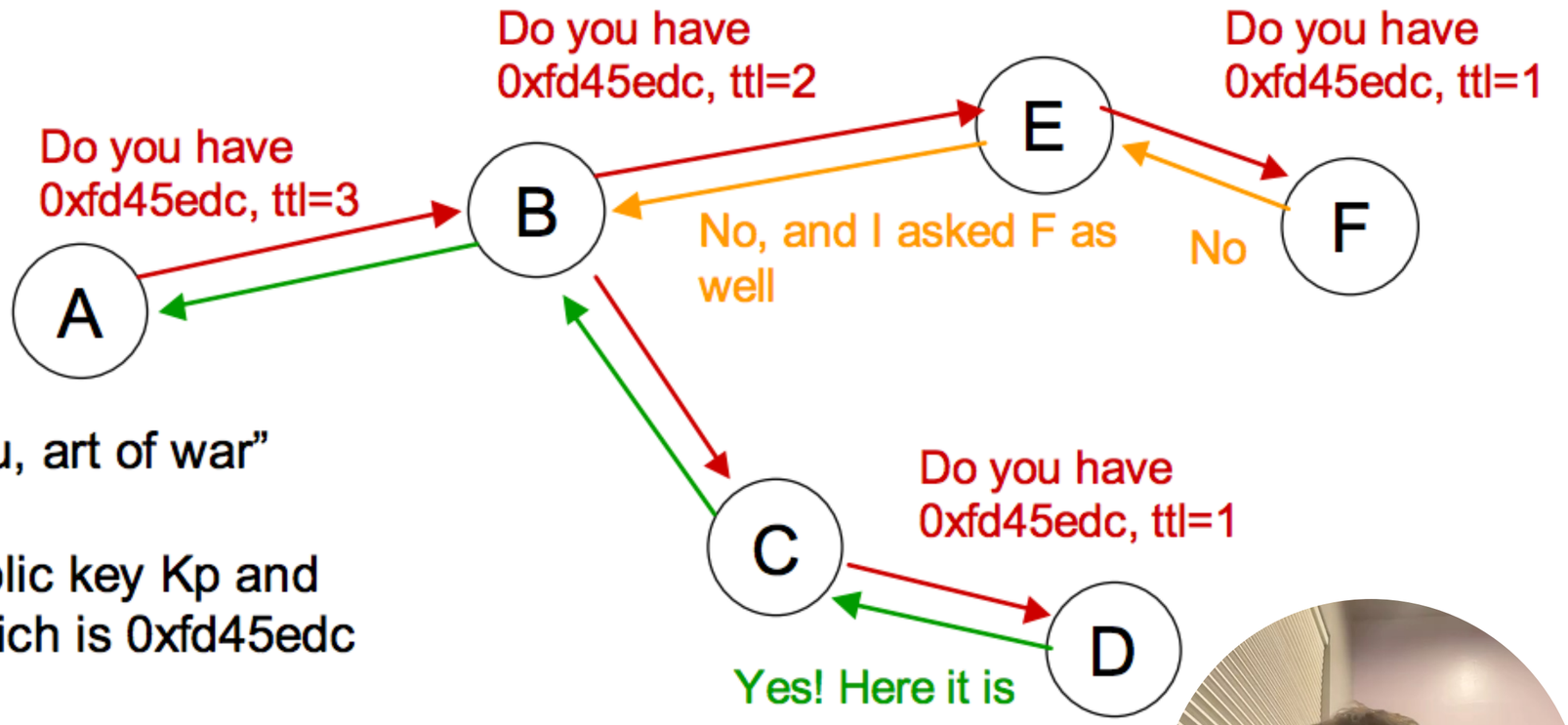
- Peer-to-peer – history connected with Napster

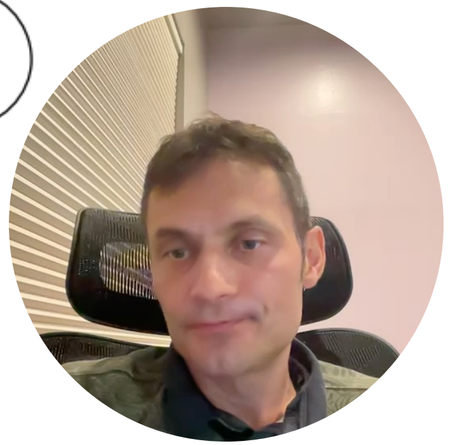




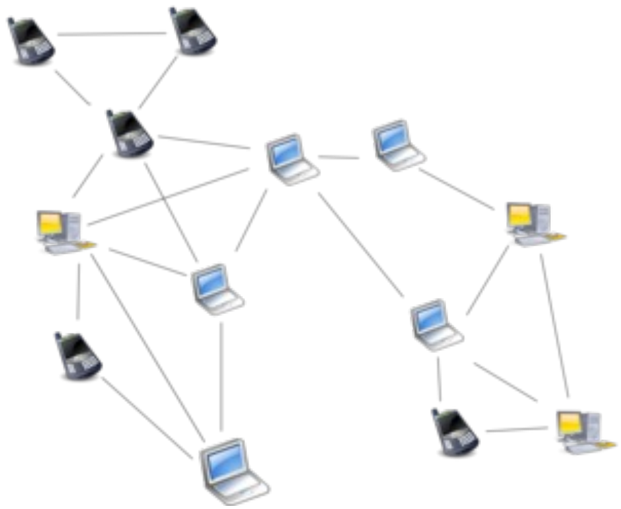
Decentralized servers

- Peer-to-peer – Solution with Freenet





Peer-to-peer Unstructured vs Structured networks



- **Unstructured**
  - Gossip
  - Gnutella
  - Kazaa
- **Structured**
  - Distributed Hash Table
  - Virtual overlays
    - Tree
  - Bittorent
  - Kademlia
  - Chord