GOSIM AI Paris 2025

Distributed Dataflows in Dora

Using Zenoh

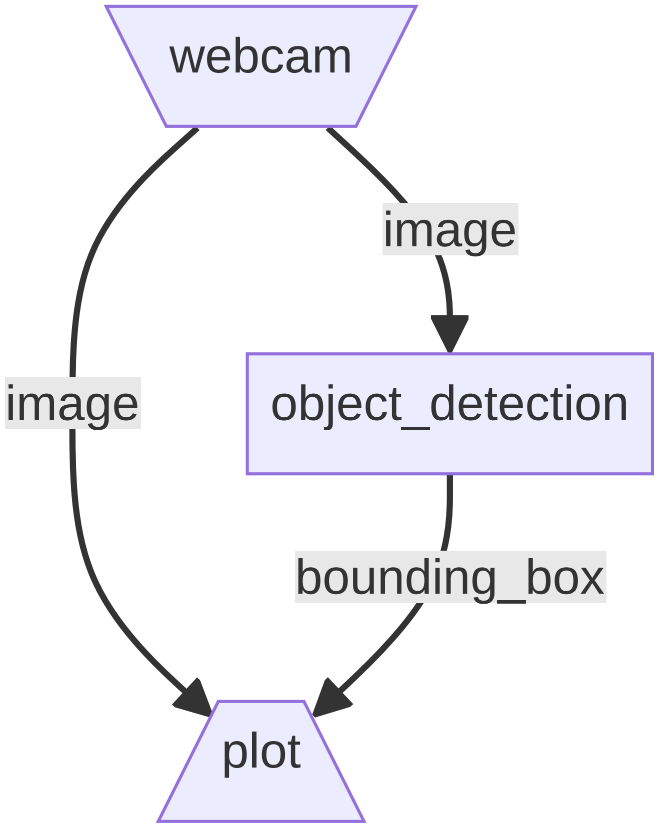
Philipp Oppermann
May 6, 2025

Agenda

- **Introduction to Dora**
- **Sending messages to remote receivers**
  - Challenges and possible solutions
- How we use **Zenoh**

The Dora Framework

- Framework for building **robotic and AI applications**
- Uses the **dataflow architecture**
  - Application are modeled as directed graph
  - Nodes represent operations
  - Data is sent along edges
- Advantages of dataflow design:
  - Isolation of components
  - Option to use multiple machines
  - Messages can be observed for debugging



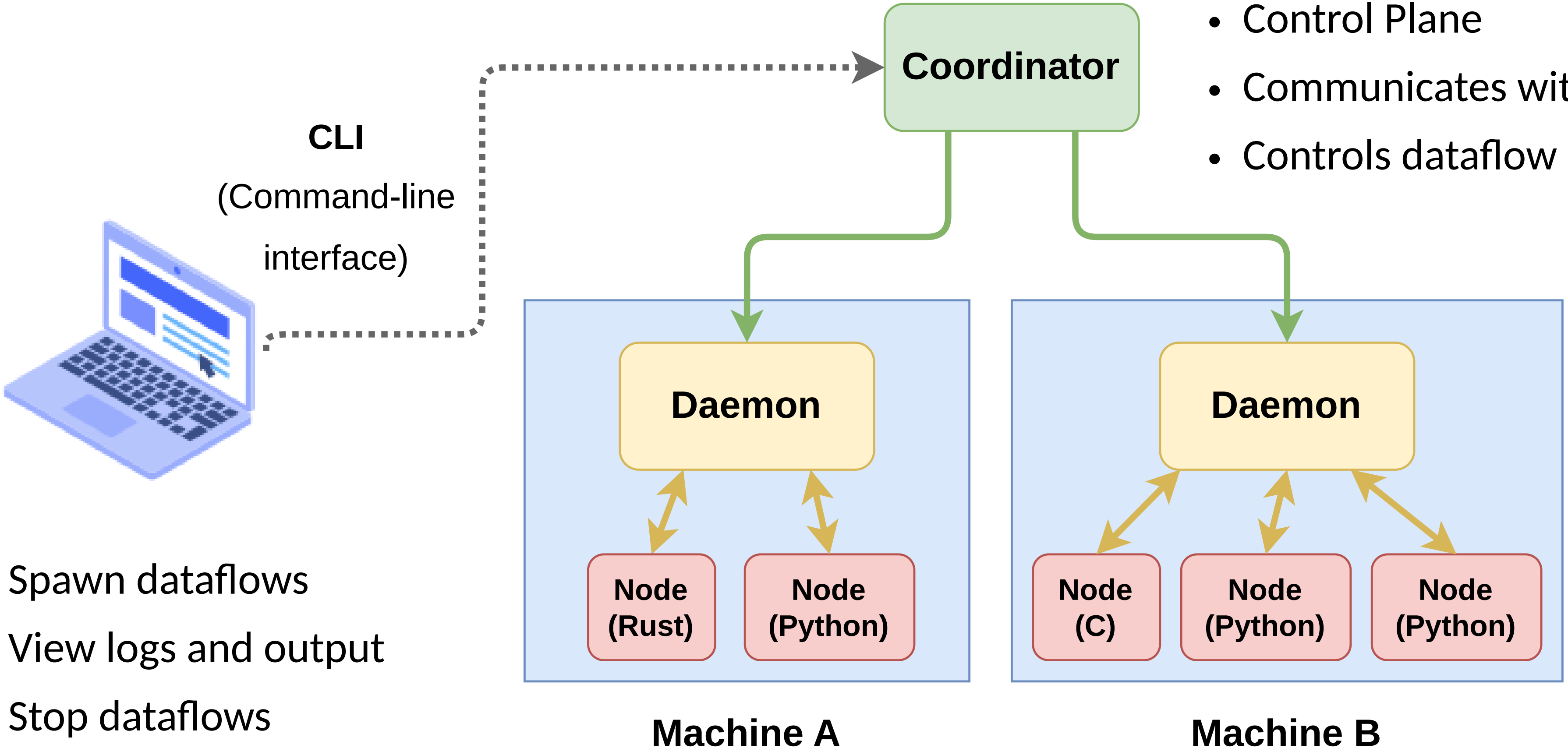
Dora: Motivation

GOSIM

- **Goal:** Make creation of robotic and AI applications **fast and simple**
- Support **various programming languages**
  - First class support for nodes written in Python and Rust
  - Also supports C and C++
- **Simple configuration and build system**
  - Define dataflow layout through a short YAML file → Example:
  - Use standard build systems and package managers
- **Node Hub** for reusing existing nodes
  - E.g. record data from microphone or webcam
  - Make it easy to use AI models (e.g. for object detection or speech recognition)

```yaml
nodes:
- id: node_1
  outputs:
    - some_output
- id: node_2
  inputs:
    foo: node_1/some_output
```

Dora: General Design



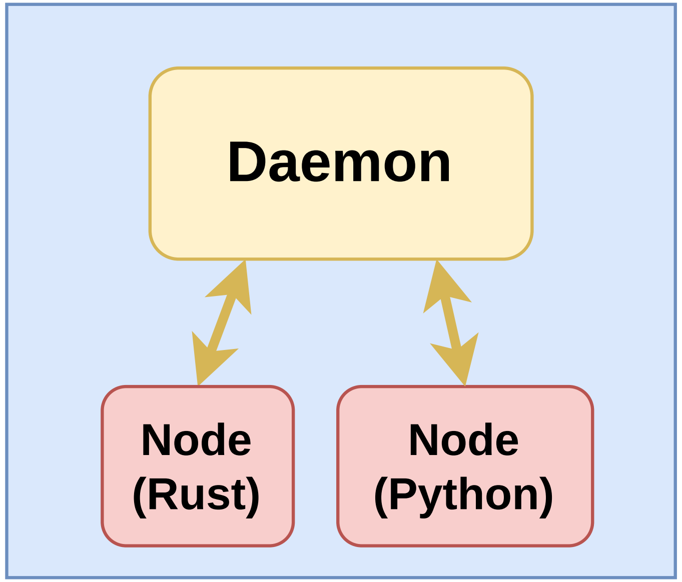
- Control Plane
- Communicates with Daemons
- Controls dataflow execution

- Spawn dataflows
- View logs and output
- Stop dataflows

Dora Daemon

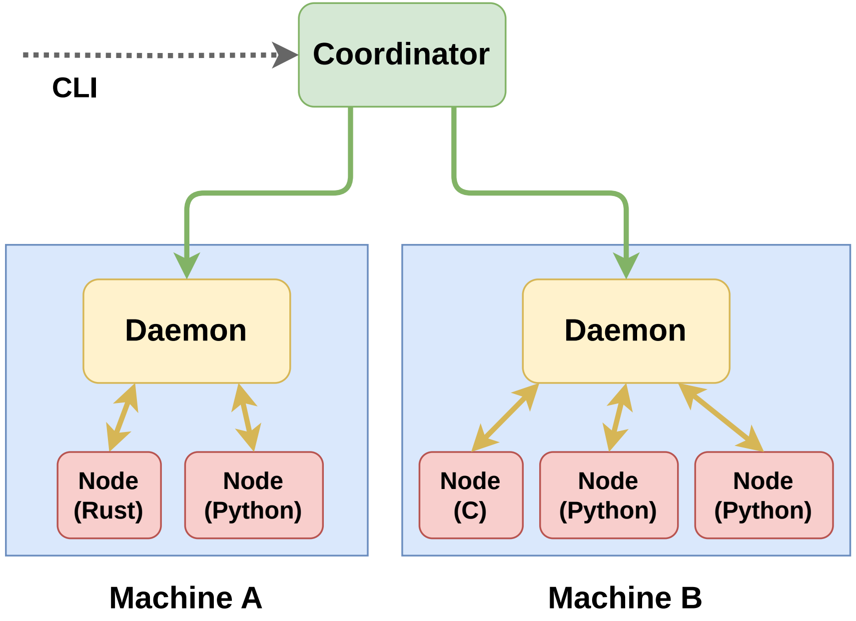
GOSIM

- One daemon per machine
- Handles **node building and spawning**
  - communicates with coordinator for synchronized spawning
  - nodes run as separate processes
- **Forwards messages** to all receivers
  - via shared memory for nodes on same machine
  - through network to nodes on other machines
- Controls execution of connected nodes
  - informs coordinator about finished nodes and errors
  - stops/kills nodes when requested by coordinator



Motivation for this Design

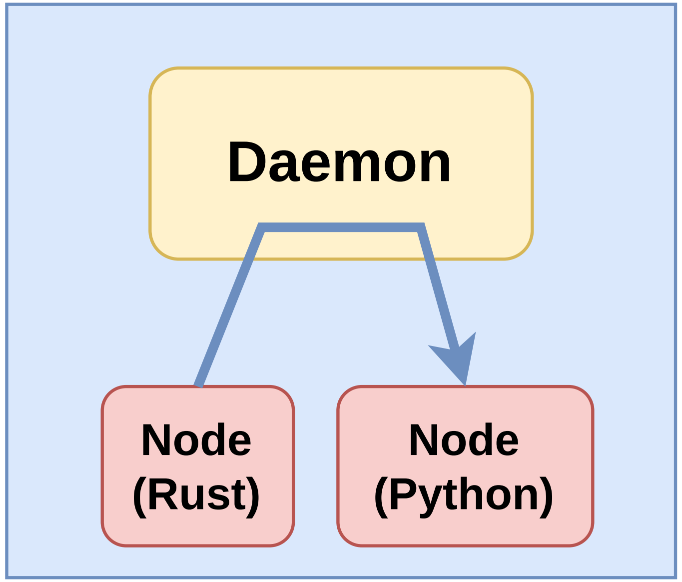
- Nodes only communicate with their local daemon
  - don't need to know about network topology
  - less dependencies for node API libraries (e.g. no SSL)
- Daemon can choose best way to pass messages
  - shared memory if receiver is on same machine
- Central Coordinator has full control of system
  - Avoids challenges of distributed systems
  - Enables synchronization across machines
  - CLI doesn't need to communicate with daemons
  - Drawback: Single point of failure



Messages to Local Receivers

GOSIM

- Messages might be big → copying them leads to overhead
- Use **shared memory** if receiver is on same machine
  - avoids the overhead of copying the message



Details:

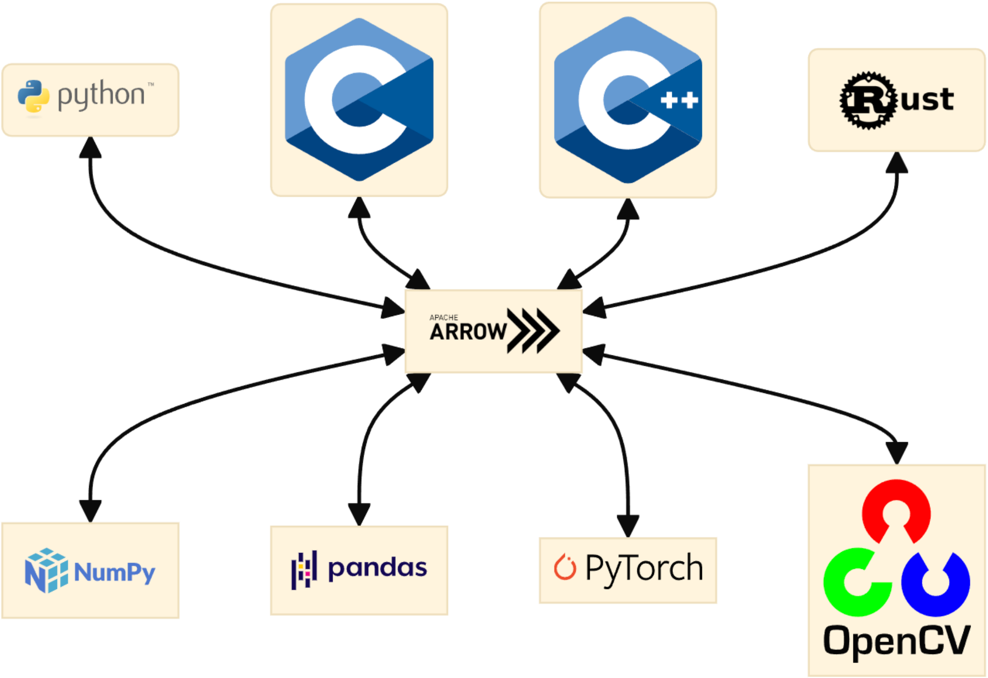
- Nodes prepare messages in shared memory
- Daemon forwards that shared memory to other local nodes
  - reference counting for cleanup
- receiving nodes can access original data without any copy

- **Challenge:** Different programming languages use **different data formats**
  - e.g. strings look different in C, Rust, and Python

Apache Arrow

GOSIM

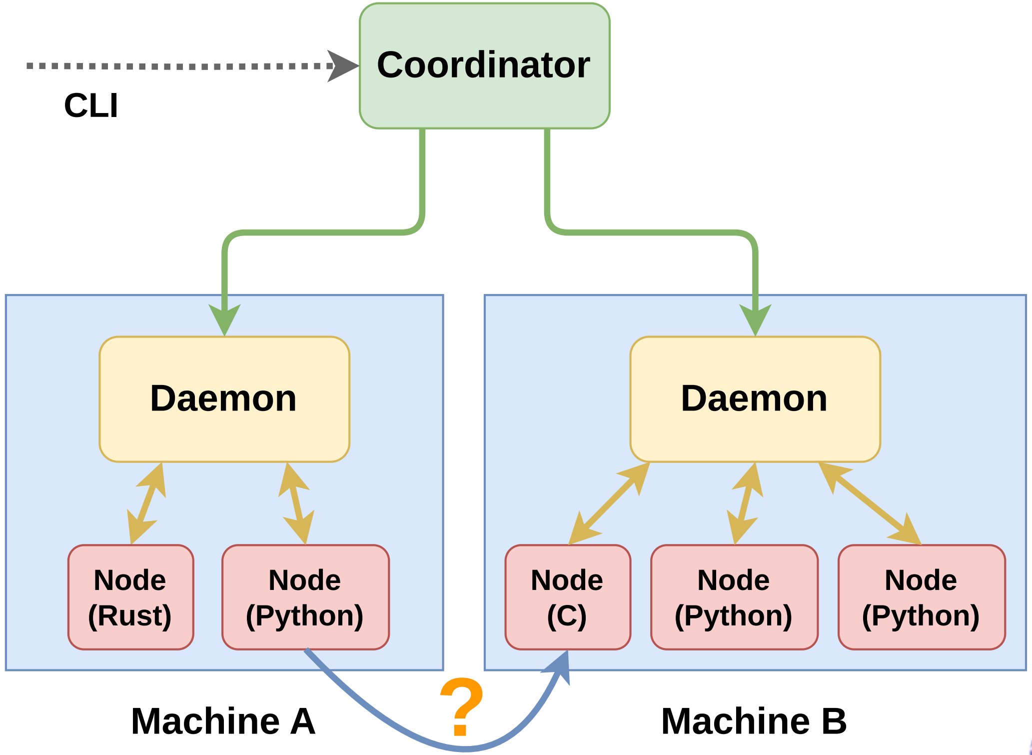
- Apache Arrow is a **cross-language data format**
- Provides bindings for Python, Rust, C, and many other languages
- Designed to enable data processing without additional copying
    - **immutable data** → no need to "backup" data
    - **lazy operations** → no intermediate copying
    - **compatible with numpy and pandas** → powerful processing without additional data conversion



→ Arrow-formatted messages in shared memory enable **zero-copy message transfer and processing**

Messages to Remote Receivers

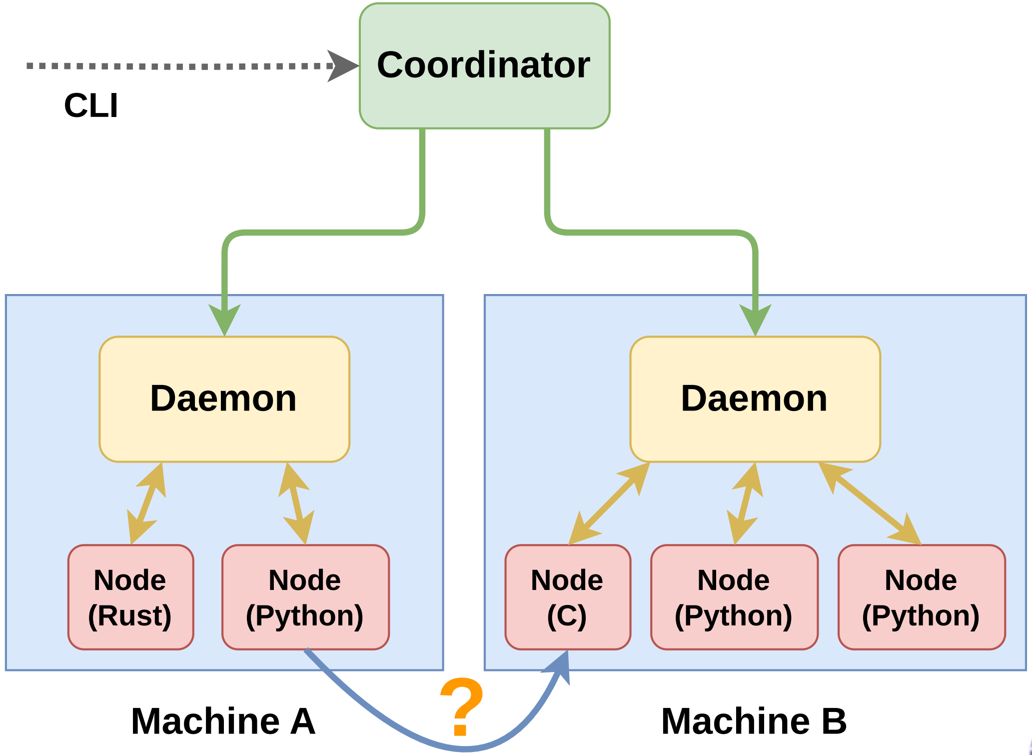
- What if receiver is on remote machine?
  - shared memory is not possible in this case
  - daemon needs to send the message **through a network connection**



Messages to Remote Receivers

GOSIM

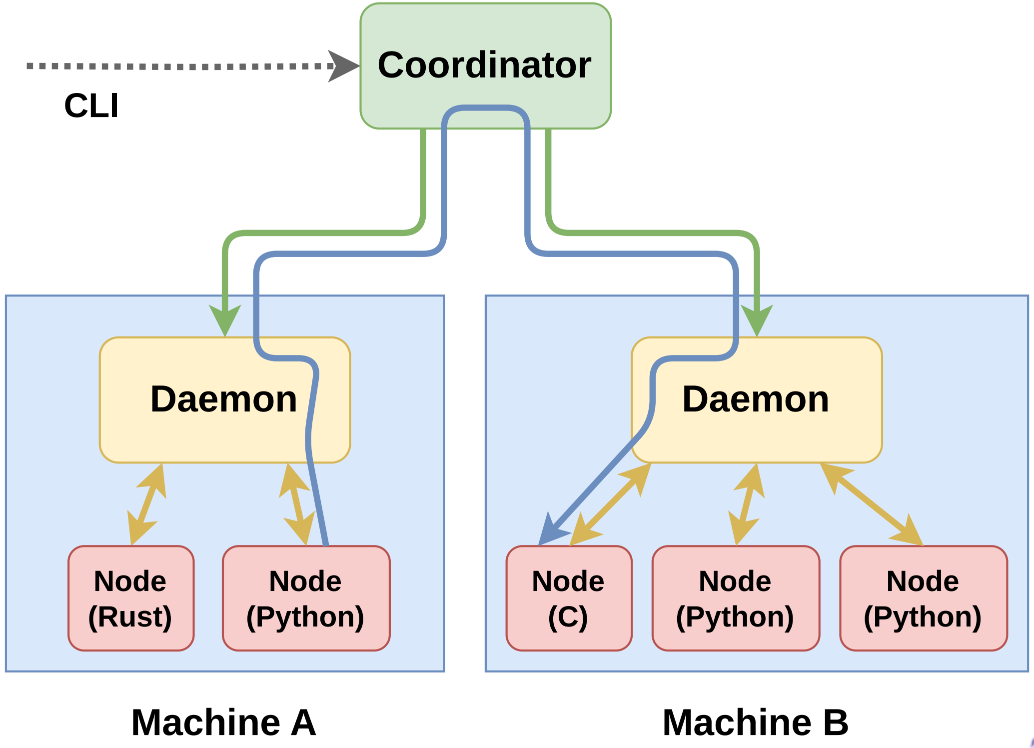
- What if receiver is on remote machine?
  - shared memory is not possible in this case
  - daemon needs to send the message **through a network connection**
- How to implement message passing?
  - Option 1: **Node opens network connection to receiving node**
    - lots of connections
    - all nodes need to be reachable via network
    - additional complexity in node API libraries



Messages to Remote Receivers

- What if receiver is on remote machine?
  - shared memory is not possible in this case
  - daemon needs to send the message **through a network connection**

- How to implement message passing?
  - ~~Option 1: Node to node connection~~
  - Option 2: **Send messages via coordinator**
    - Daemon sends messages to coordinator, which then forwards them to other daemon
    - Additional hop → slightly worse latency
    - A lot of work for the single coordinator → bottleneck

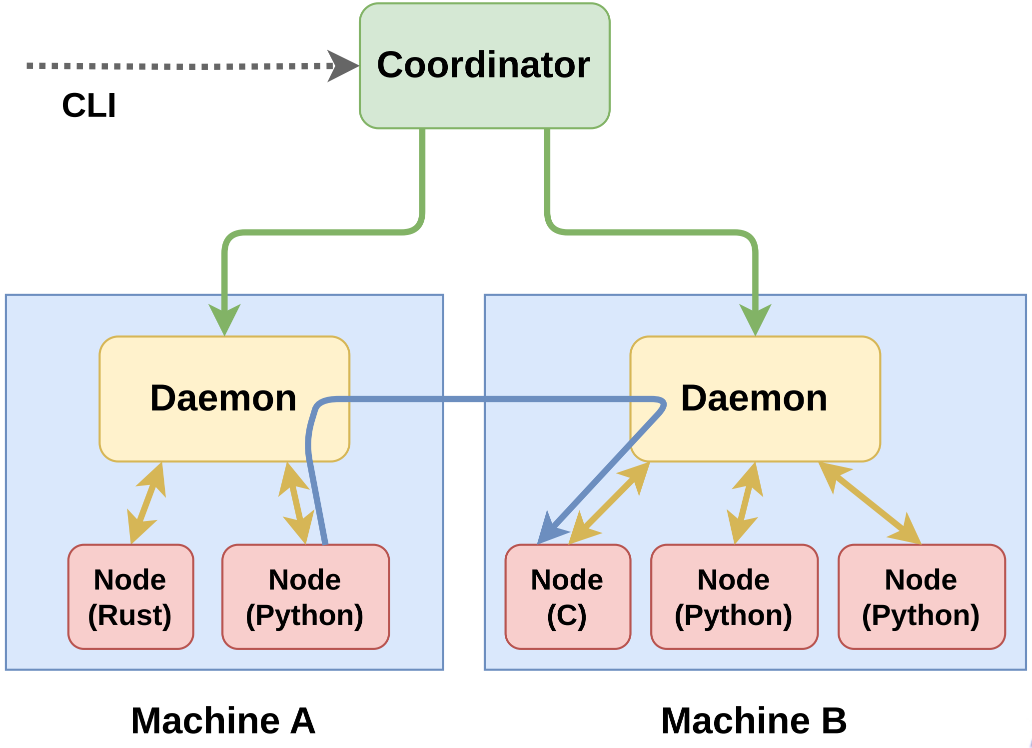


Messages to Remote Receivers

GOSIM

- What if receiver is on remote machine?
  - shared memory is not possible in this case
  - daemon needs to send the message **through a network connection**

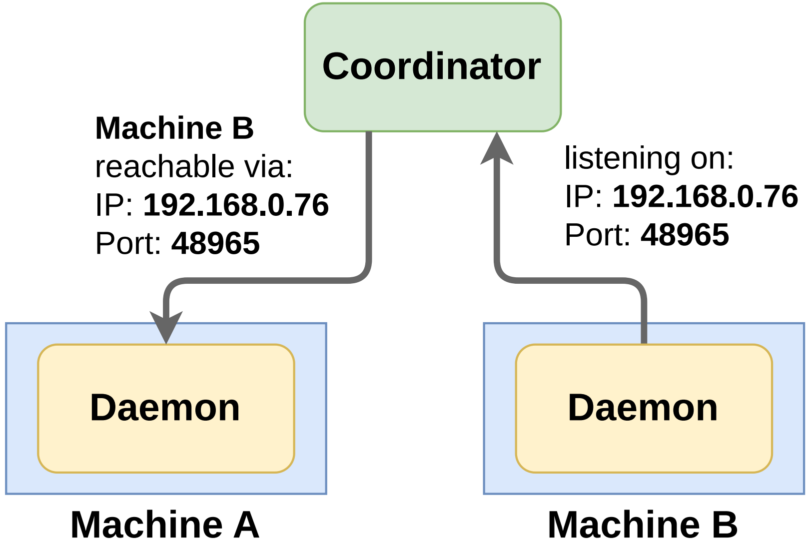
- How to implement message passing?
  - ~~Option 1: Node to node connection~~
  - ~~Option 2: Send messages via coordinator~~
  - Option 3: **Connect daemons to each other**
    - no extra work for coordinator
    - each daemon needs to be reachable by other daemons



Connecting Daemons

Step 1:

- Each daemon **listens on some local port**
- Daemons tells coordinator socket address
  - **IP address** and **port number**
- Coordinator distributes that information to other daemons



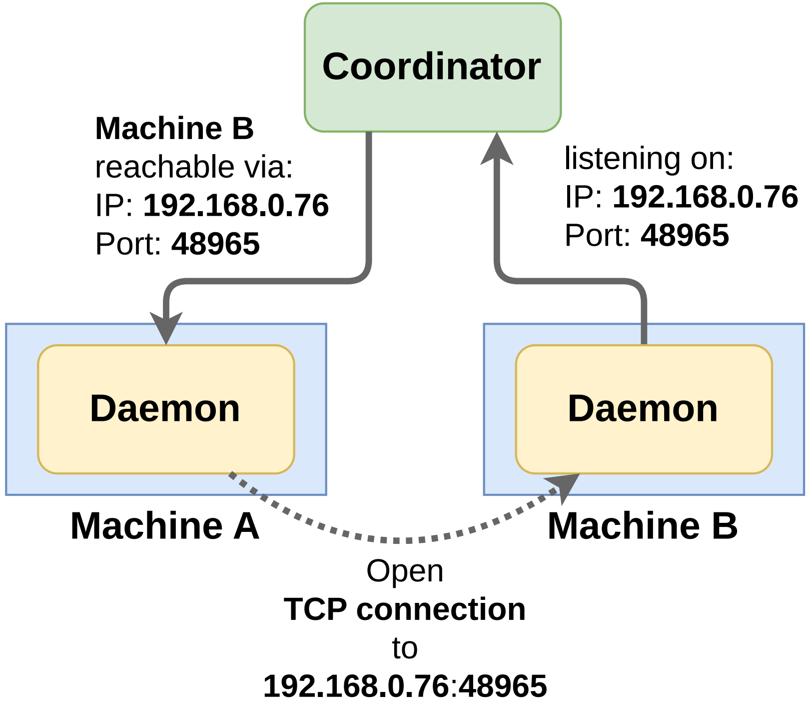
Connecting Daemons

Step 1:

- Each daemon **listens on some local port**
- Daemons tells coordinator socket address
  - **IP address** and **port number**
- Coordinator distributes that information to other daemons

Step 2:

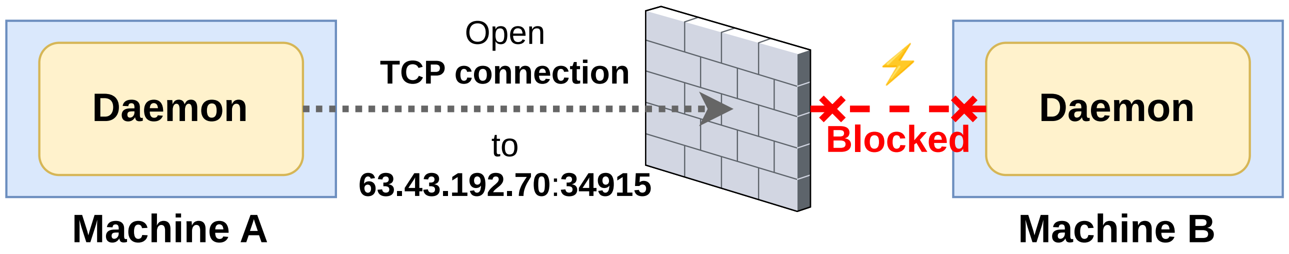
- Daemons can **open TCP connections** to other daemons
  - Use this connection to forward messages



Firewalls and NATs

What if daemons are **not in same local network**?

- Daemons need to communicate through the **internet**
- Most computers are not exposed to the internet directly → for security reasons
- **Firewalls** block traffic
  - Incoming connections are normally blocked
  - **Firewall exception required** to expose daemon listen port

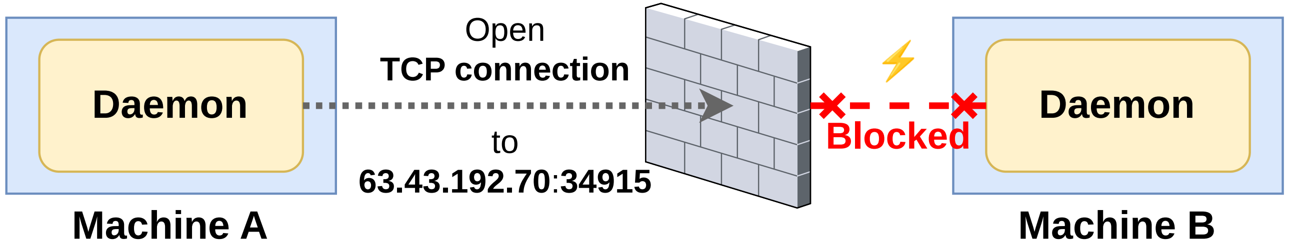


Firewalls and NATs

What if daemons are **not in same local network**?

- Daemons need to communicate through the **internet**
- Most computers are not exposed to the internet directly → for security reasons

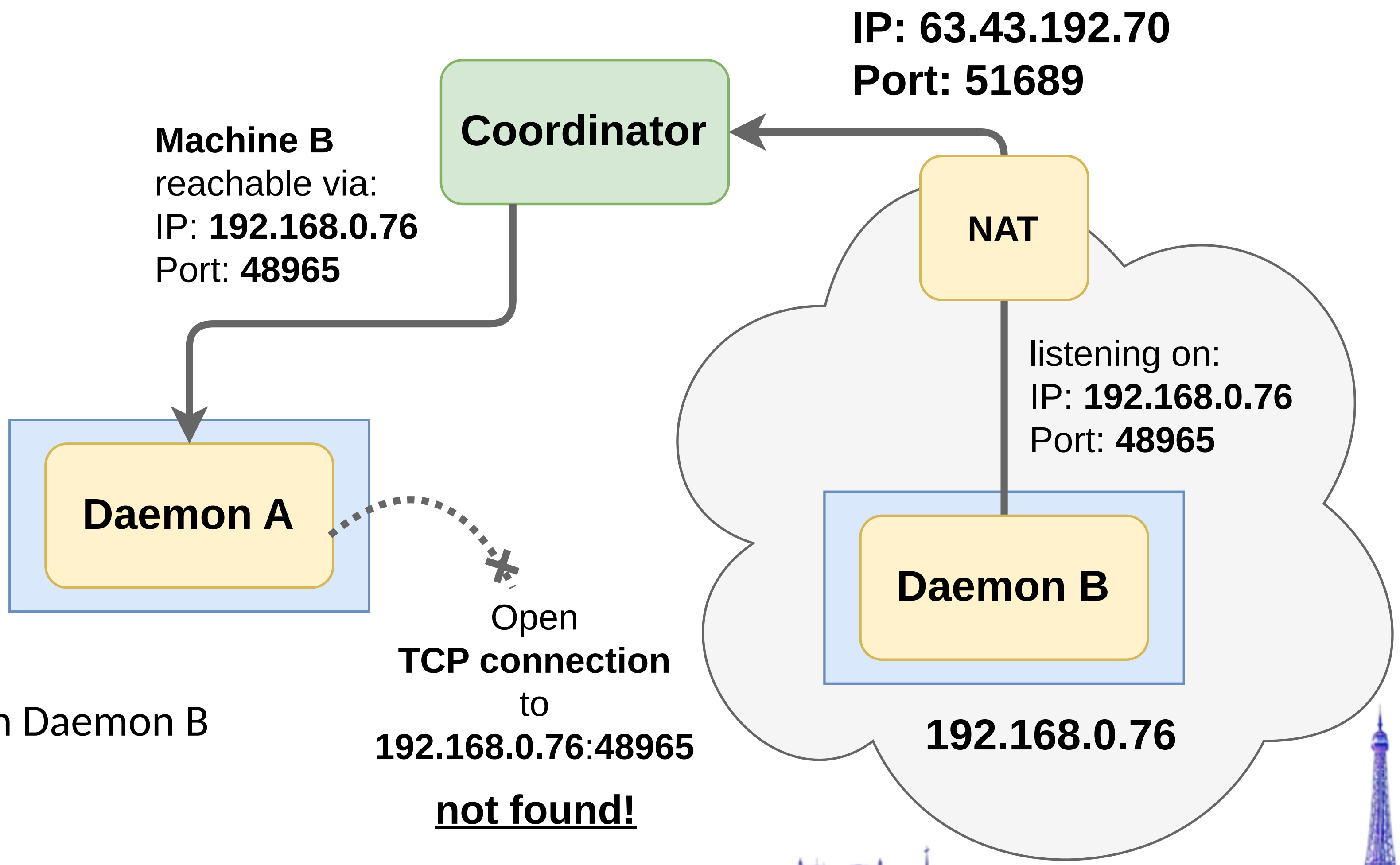
- **Firewalls** block traffic
  - Incoming connections are normally blocked
  - **Firewall exception required** to expose daemon listen port



- **NATs** change IP addresses and port numbers → stands for *network address translation*
  - common in consumer and cloud networks → often multiple levels
  - whole private network can share one public IP address

NAT Example

- NAT replaces IP address and port number in IP packets
- Daemon B only sees its internal IP `192.168.0.76`
  - Connection to coordinator uses external IP `63.43.192.70`
- Daemon A cannot reach Daemon B through internal IP



Bypass Firewalls and NATs

GOSIM

- Various hacks to bypass NATs
  - see "[Peer-to-Peer Communication Across Network Address Translators]" by Ford et al.
  - does not work for all NAT setups → not reliable enough for Dora

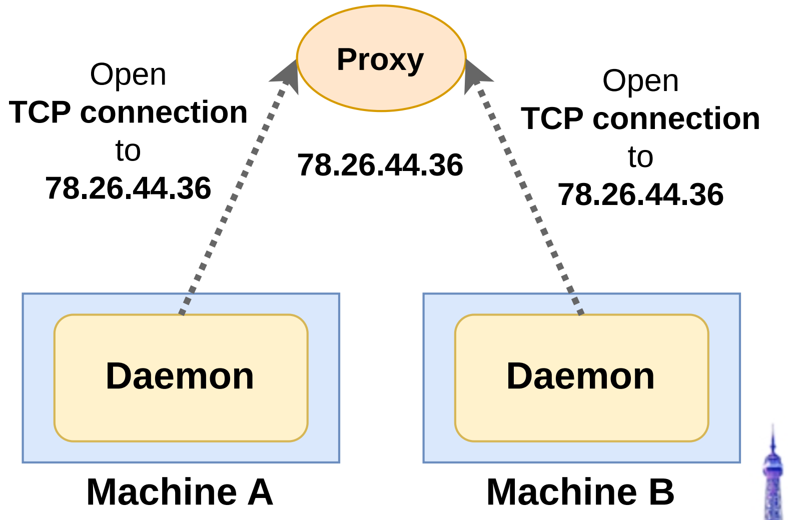
  → daemon-to-daemon connections are not practical

Bypass Firewalls and NATs

- Various hacks to bypass NATs
  - see "[Peer-to-Peer Communication Across Network Address Translators]" by Ford et al.
  - does not work for all NAT setups → not reliable enough for Dora

  → **daemon-to-daemon connections are not practical**

- **Communicate via proxy server** to bypass NATs and firewalls
  - outgoing connections are not affected by NATs and firewalls (usually)
  - *Step 1*: create connection to proxy

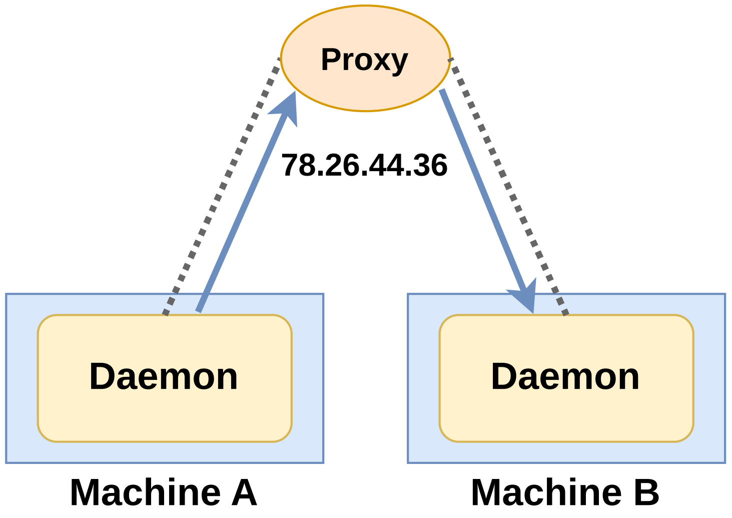


Bypass Firewalls and NATs

- Various hacks to bypass NATs
  - see "Peer-to-Peer Communication Across Network Address Translators" by Ford et al.
  - does not work for all NAT setups → not reliable enough for Dora

  → **daemon-to-daemon connections are not practical**

- **Communicate via proxy server** to bypass NATs and firewalls
  - outgoing connections are not affected by NATs and firewalls (usually)
  - *Step 1*: create connection to proxy
  - *Step 2*: **send message to proxy and let it forward it**
  - drawback: additional network hop
  - similar to sending messages via coordinator, but no additional work for coordinator + more scalable



Proxy Requirements

GOSIM

A good proxy server for Dora should be:

- Reliable → don't lose messages
- Scalable → avoid bottlenecks
- Fast
  - low latency
  - high throughput
- Support complex network topologies → from cloud to local networks

Zenoh

GOSIM

- Pub/Sub/Query protocol
- Supports **peer-to-peer** and **routed** communication
- High performance
- Open Source

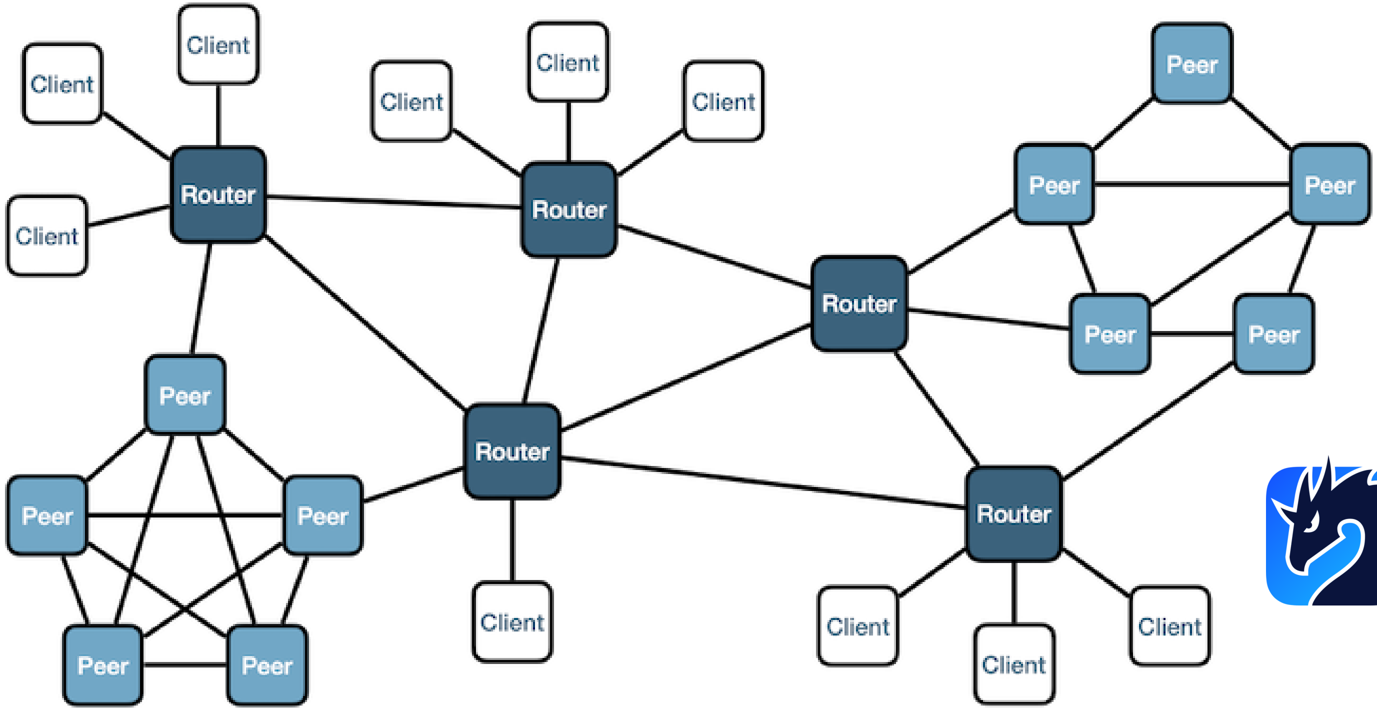
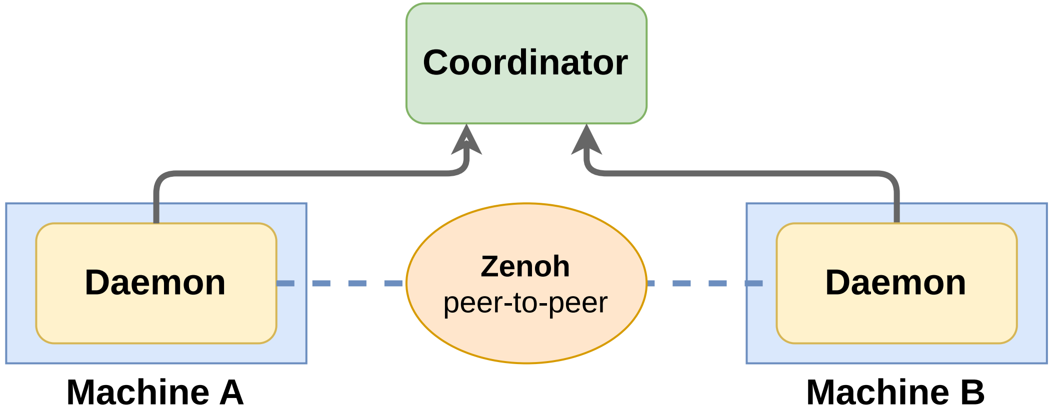
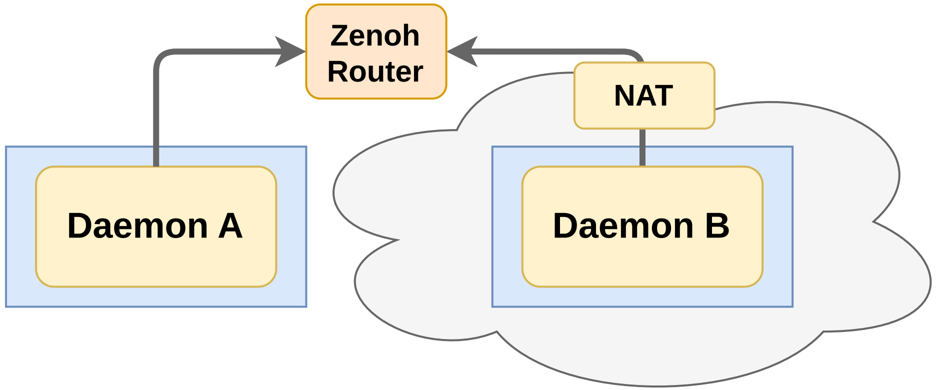


image source: https://zenoh.io

Forwarding Messages using Zenoh

- Keep existing TCP connection to coordinator (we plan to use GRPC for the control plane)
- Use Zenoh to connect daemons to each other
  - **peer-to-peer for local networks:**
  - **use routers to bypass firewalls + NATs:**





- Topic format: `dora/{network_id}/{dataflow_id}/output/{node_id}/{output_id}`

Distributed Dataflow Example

- Start `dora coordinator` on a machine with a public IP   *i.e. a machine reachable by all daemons*
- Start **`dora daemon`** instances on all machines
  - assign each daemon a **unique ID**
  - pass coordinator IP and listen port as arguments
- Specify **daemon ID** for each node in `dataflow.yml` file
  - syntax is still unstable
- Run `dora start dataflow.yml` to run the dataflow
  - messages to local receivers use **shared memory**
  - messages to remote machines are sent through **Zenoh**

```yaml
nodes:
  - id: numbers
    _unstable_deploy:
        machine: A
    outputs:
        - random
  - id: add-one
    _unstable_deploy:
        machine: A
    inputs:
        random_number: numbers/random
    outputs:
        - result
  - id: print-numbers
    _unstable_deploy:
        machine: B
    inputs:
        number: add-one/result
```

Zenoh Config

- Zenoh uses **IP multicast** packets for **autodiscovery in local networks**
  - peers can find each other without configuration
  - also: *gossip scouting* → peers tell each other about discovered peers/routers
- Distributed Zenoh networks manual endpoint configuration
  - Zenoh is configured through a JSON file
  - specify IP addresses of remote Zenoh routers
- Set `ZENOH_CONFIG` env variable with the path to the JSON config file
  - Dora will then use this config when initializing Zenoh
  - Modifying Zenoh config requires restart of Dora
    - no stable to change Zenoh config without reinitialization

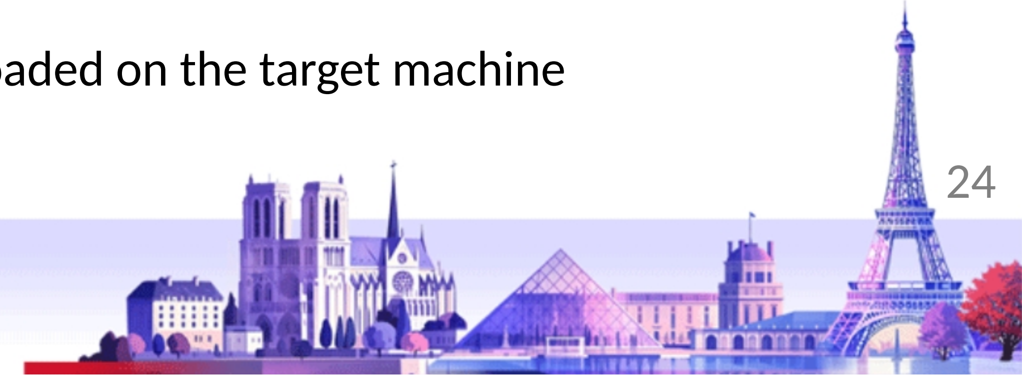
```
{
  connect: {
    endpoints: [
      "tcp/192.168.1.1:7447",
      "tcp/192.168.1.2:7447"
    ],
  },
}
```

Deployment

GOSIM

How to copy nodes from development machine to target machine?

- **Manual Copy**
  - copy and/or build the nodes on their target machines manually
  - just specify a `path` source for the nodes
- **Git Repo** (in development)
  - specify the repository URL and branch name for each node
  - Dora will automatically clone the repository (or pull on reuse)
  - you can also specify a `build` command that is executed before spawning
- **Docker** (planned)
  - Goal: specify a docker container that will be downloaded on the target machine



Future Plans

- Authentication + Traffic Encryption
  - use HTTPS instead of bare TCP/Zenoh
  - require credentials for connecting
- Deployment
  - Docker support
  - option to auto-select a suitable daemon (e.g. for load balancing)
- Simplify setup for distributed Dora
  - specify coordinator IP through environment variable or config file
  - auto-discovery of coordinator instances
  - better documentation
  - first-class support for common cloud platforms

GOSIM

Summary

GOSIM AI Paris 2025

Distributed Dataflows in Dora using Zenoh

- Dora runs **one daemon per machine**
- Daemons forward messages for nodes
  - using **shared memory for local receivers**
  - through **Zenoh for remote receivers**

- Zenoh supports **auto-discovery** within local networks
- Bypass firewalls and NATs using Zenoh routers
- Deploy nodes through `git` and `build` config

- Find us on **dora-rs.ai** and GitHub