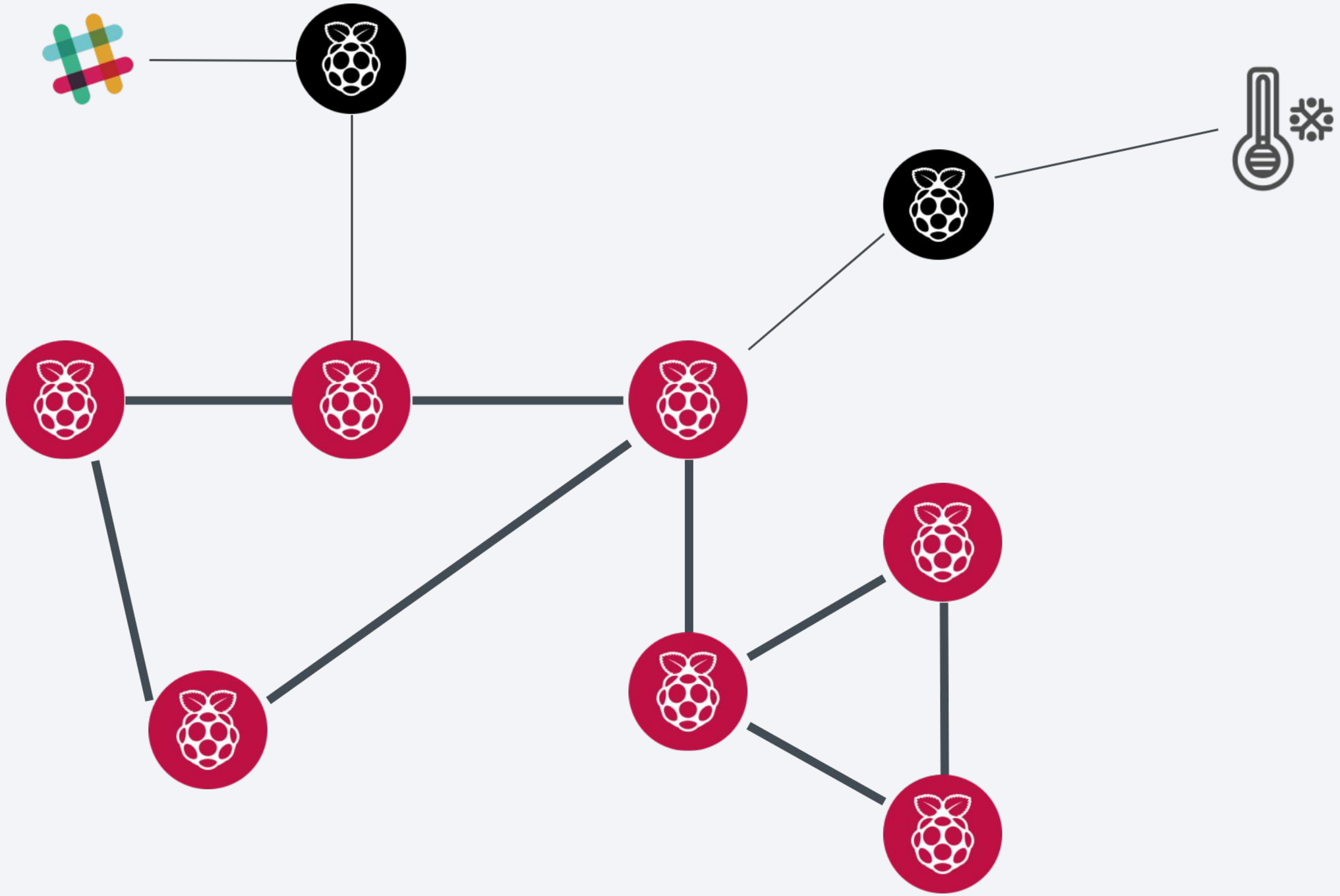
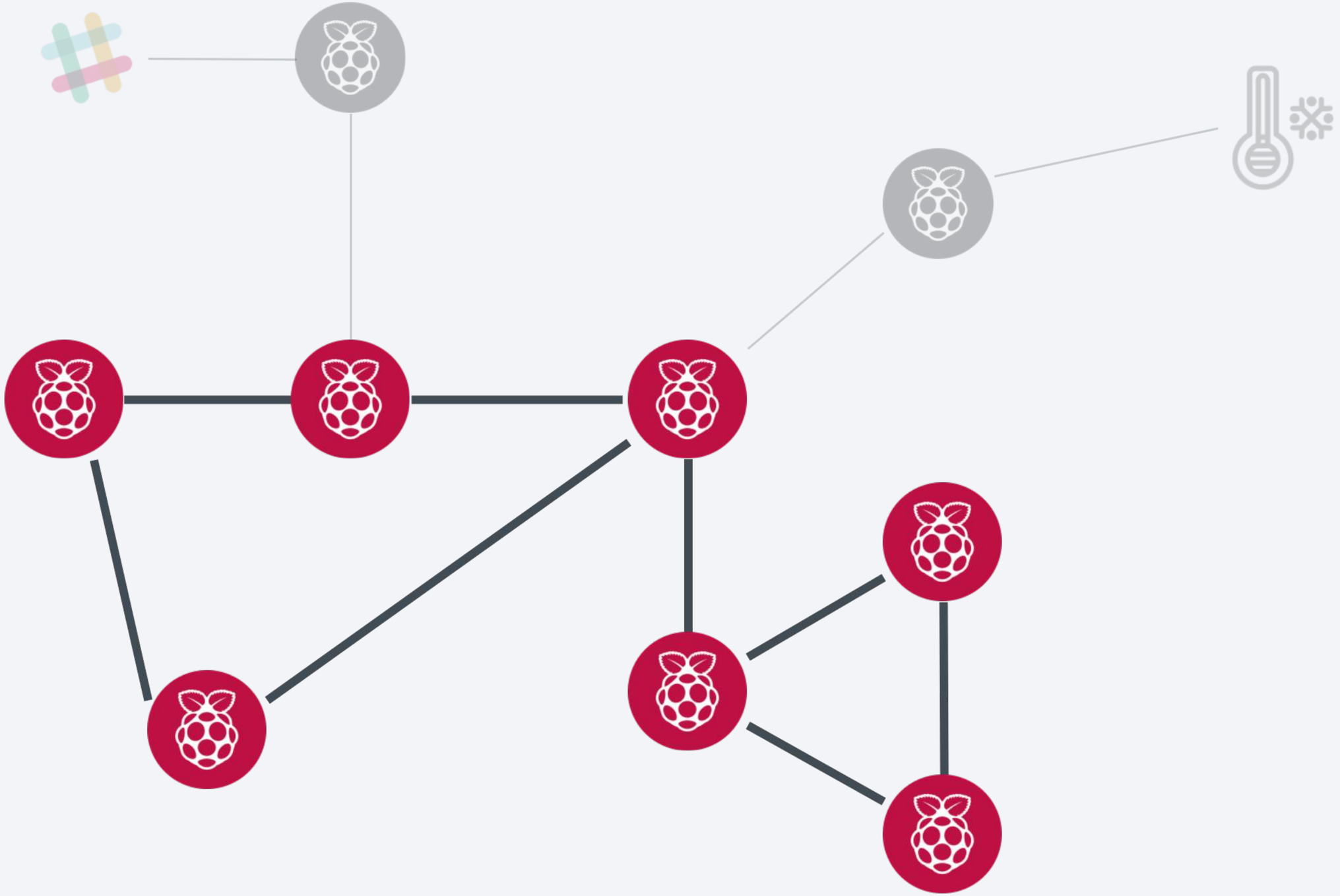
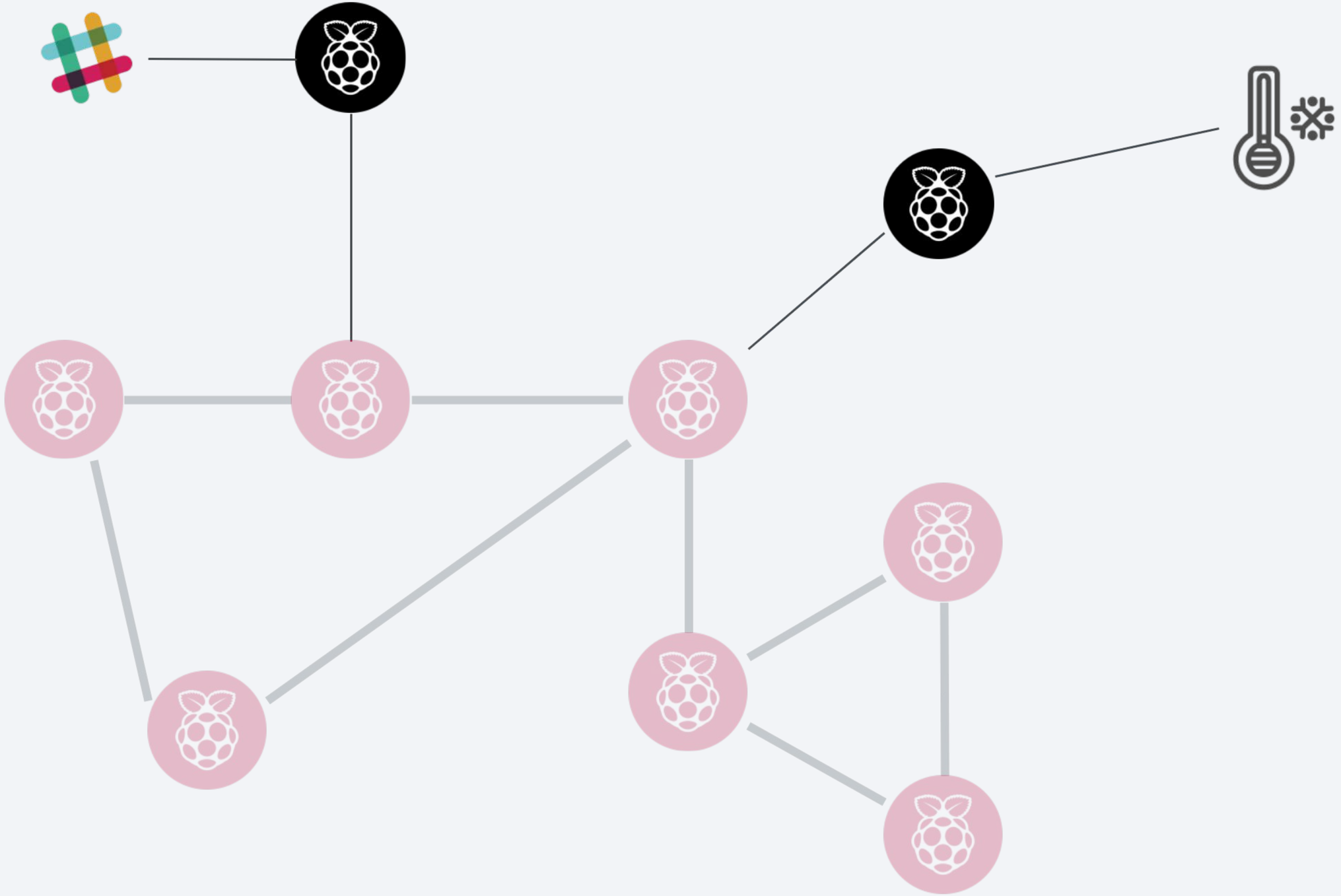
Knee-Deep Into P2P

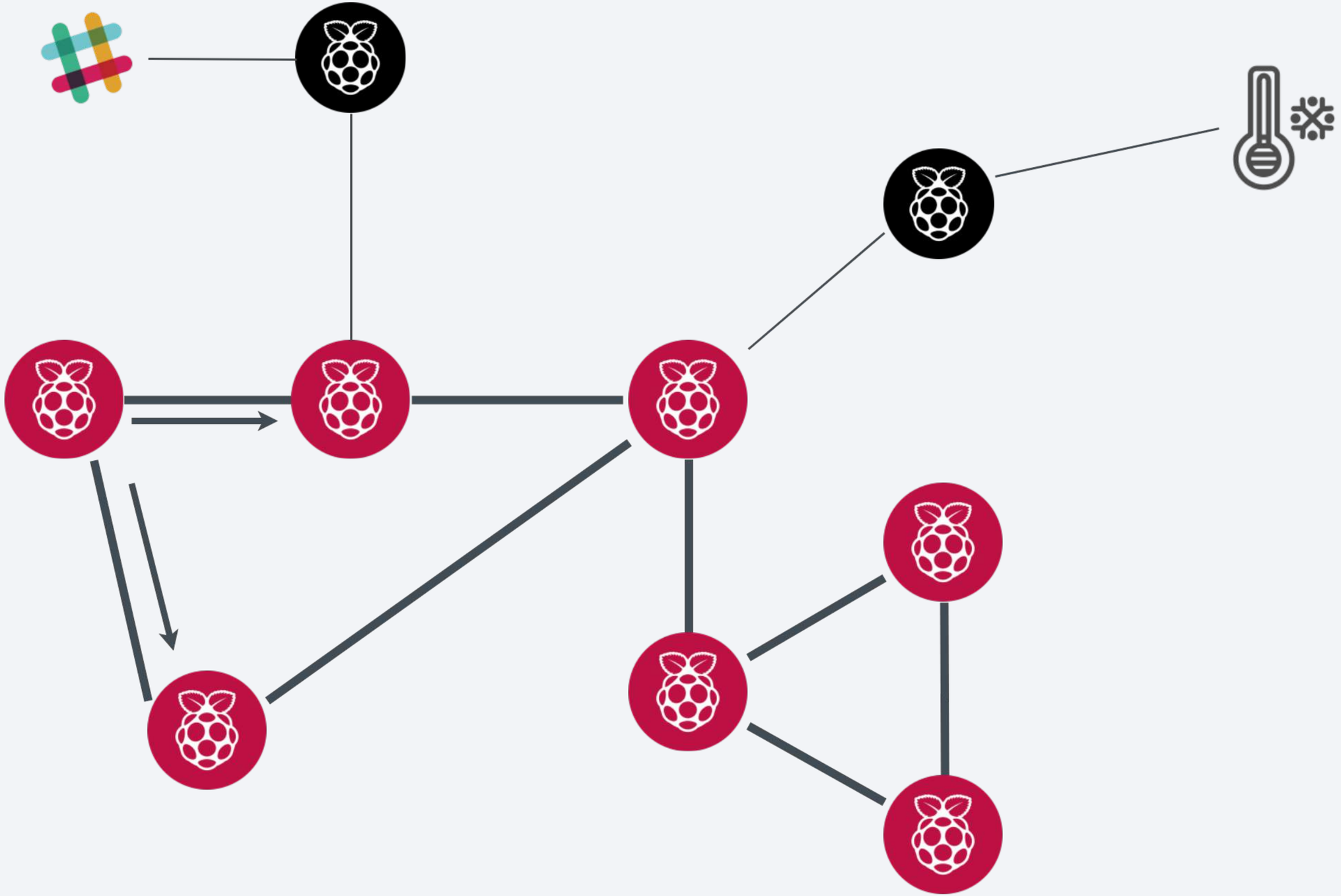
A Tale of Fail

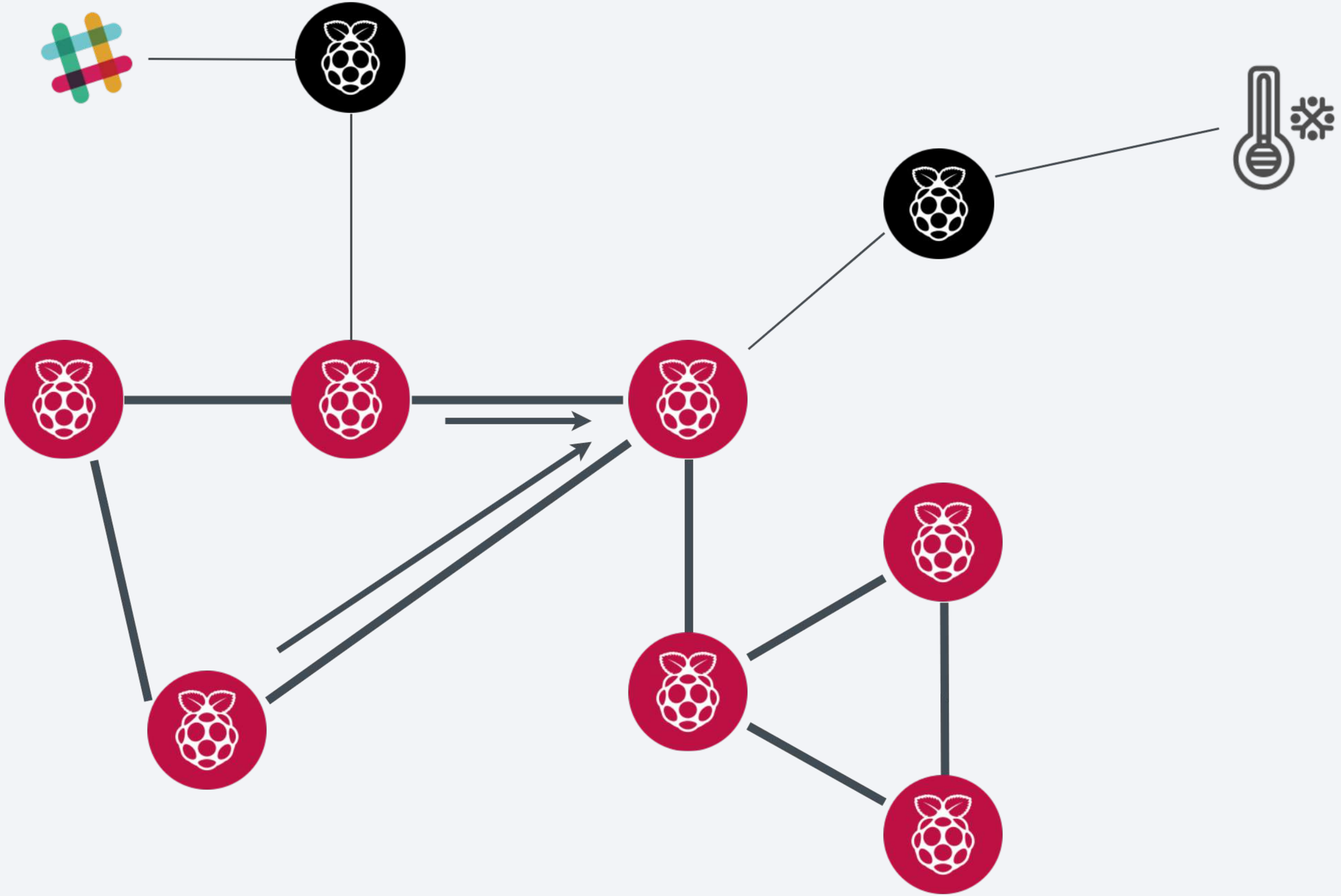
@fribmendes

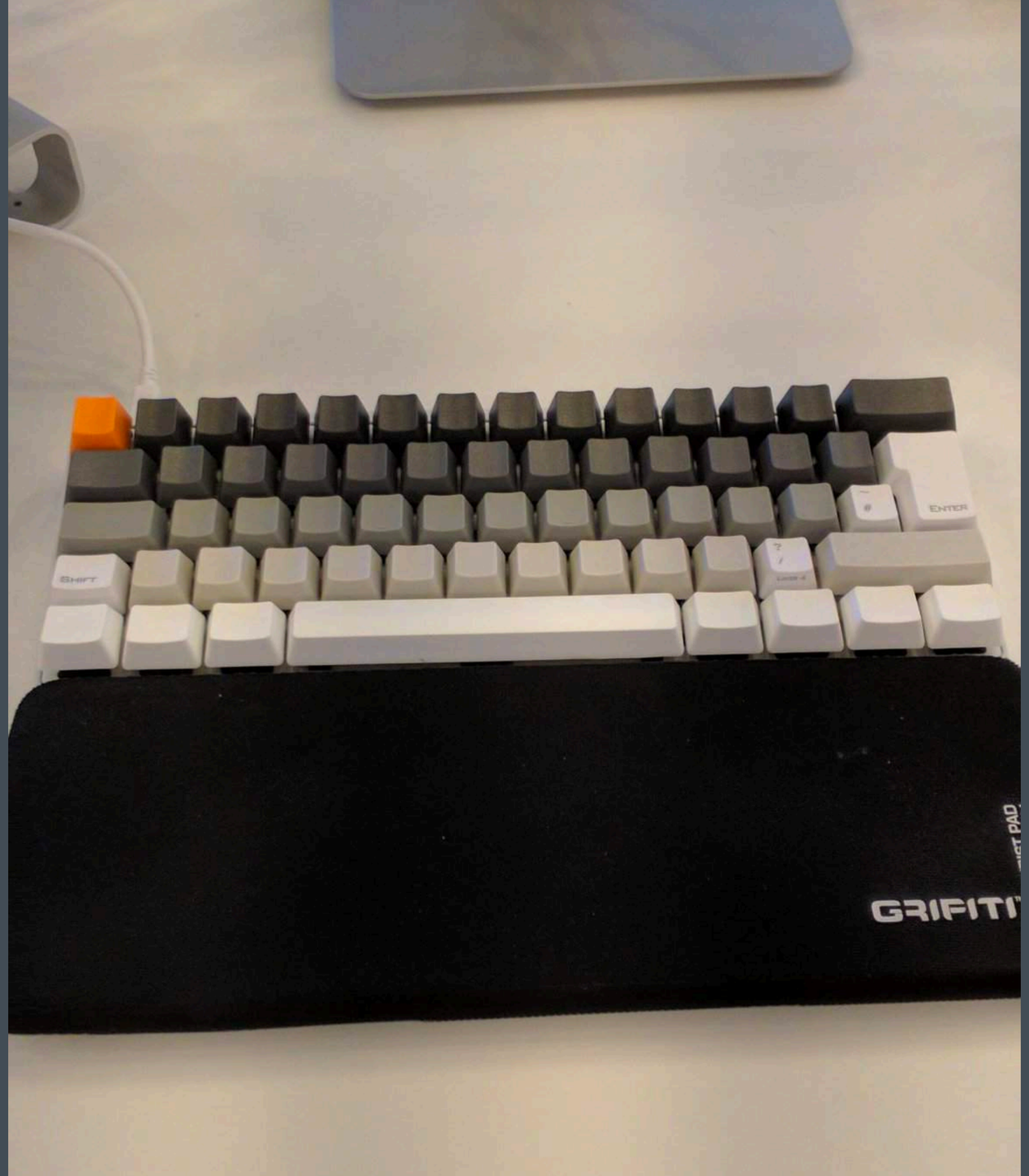











ENTER
SHIFT
GRIFITI

I don’t know how to smart office

I know how to web development

I know how to web development

... what now?

me failing at photoshop

@fribmendes

SUBVISUAL

SUBVISUAL

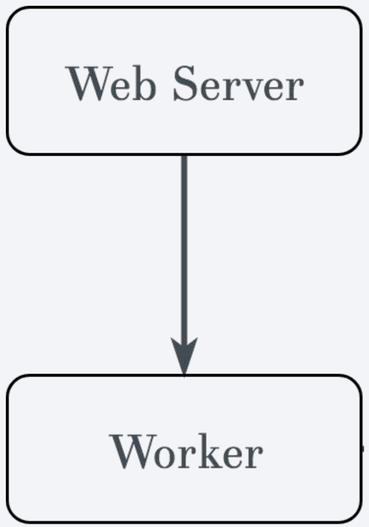
Mirror Conf

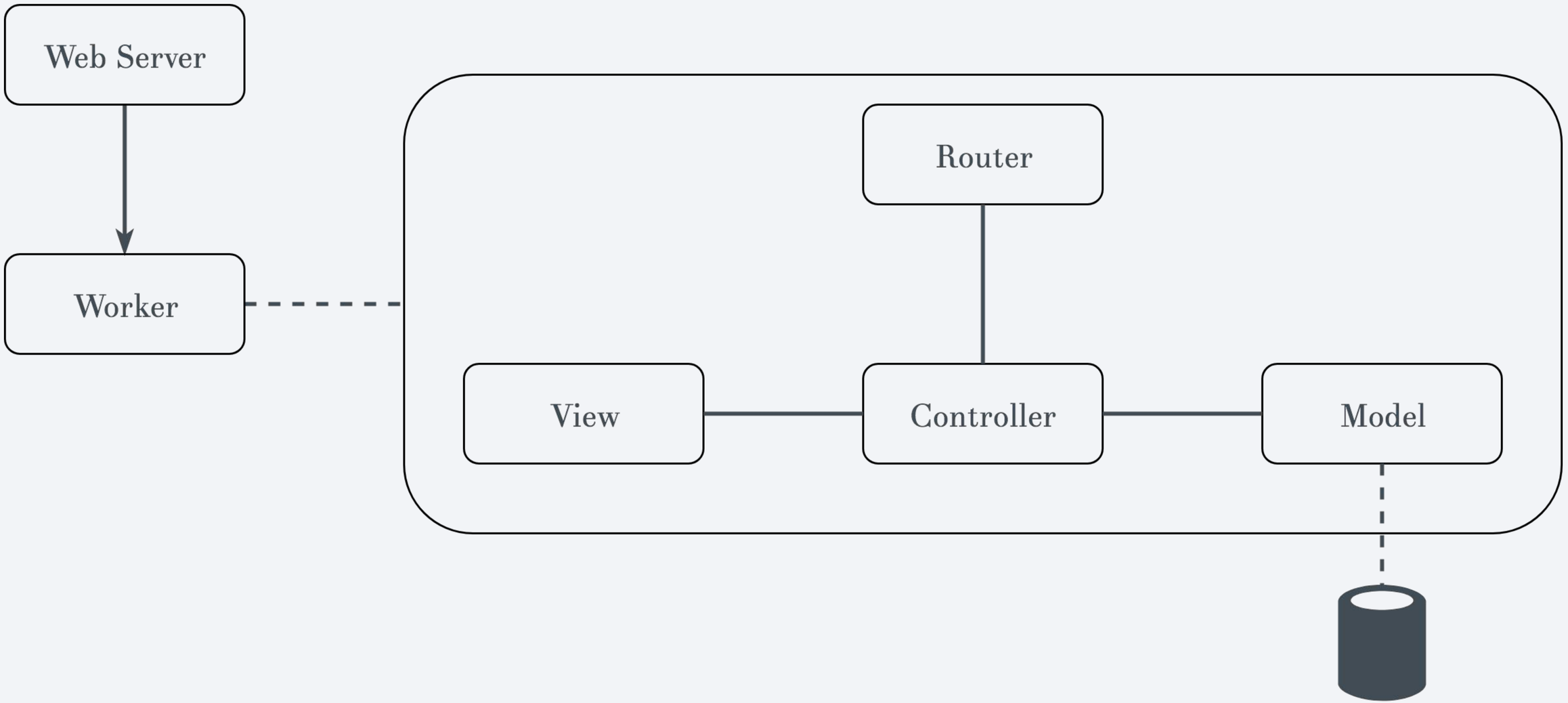
UTRUST

I know how to web development

... what now?


Web Server


Web Server
Worker


Web Server
Worker
Router
View
Controller
Model

Step 1: receive new connections

Step 1: receive new connections

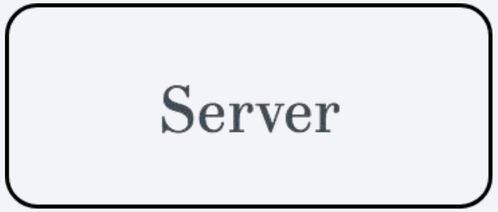
Step 2: accept and send messages

Step 1: receive new connections

Step 2: accept and send messages

Step 3: do a bunch of Steps 1 and 2

Step 1: receive new connections

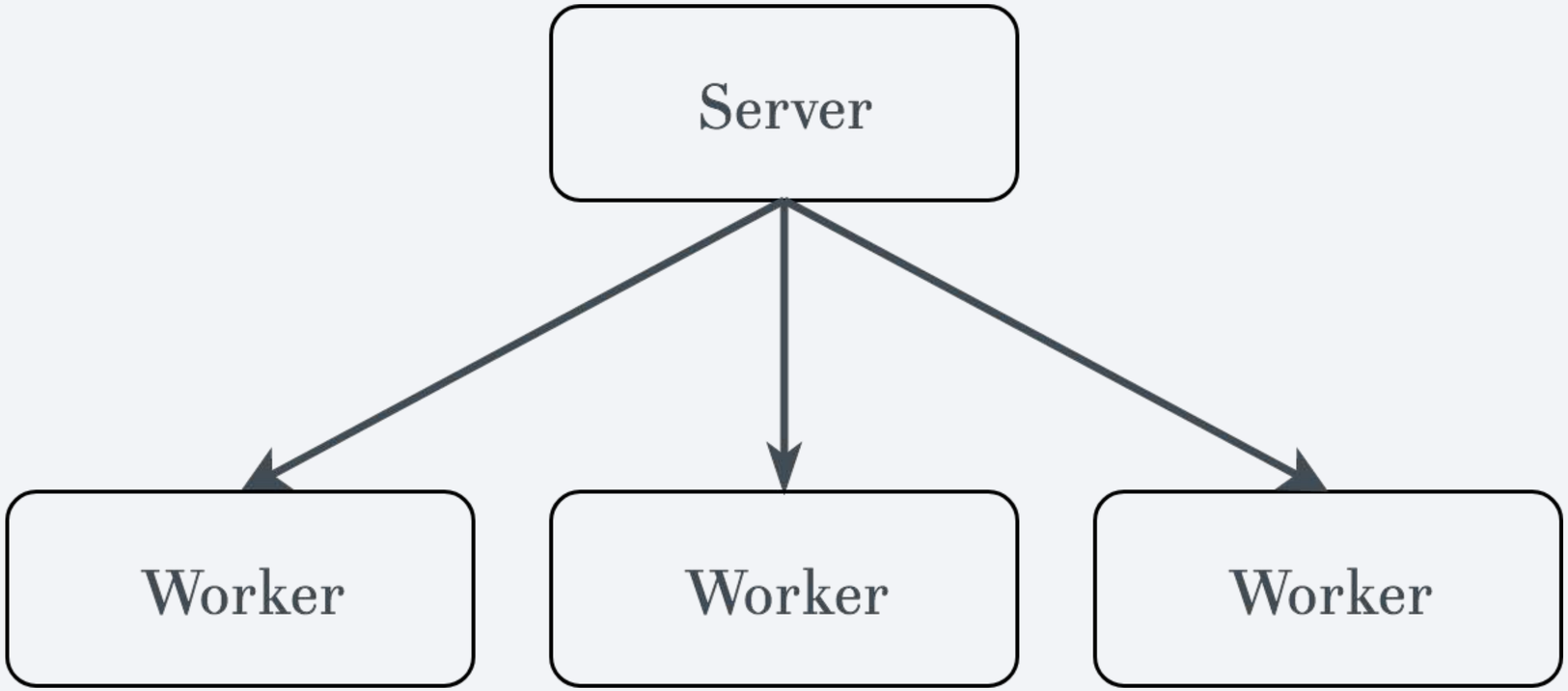

Server

```elixir
defp accept_loop(pid, server_socket) do
  {:ok, client} = :gen_tcp.accept(server_socket)
  :inet.setopts(client, [active: true])
  :gen_tcp.controlling_process(client, pid)

  Gossip.accept(pid, client)

  accept_loop(pid, server_socket)
end
```

```elixir
defp accept_loop(pid, server_socket) do
  {:ok, client} = :gen_tcp.accept(server_socket)
  :inet.setopts(client, [active: true])
  :gen_tcp.controlling_process(client, pid)

  Gossip.accept(pid, client)

  accept_loop(pid, server_socket)
end
```

Step 1: receive new connections

Step 2: accept and send messages


Server
Worker
Worker
Worker

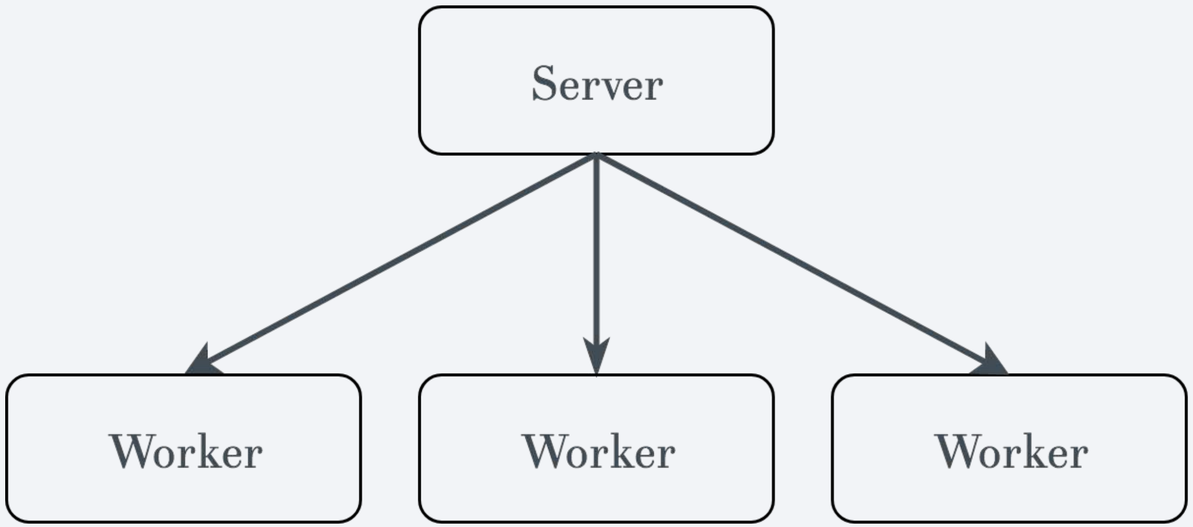
```elixir
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # process an incoming message

    {:tcp_closed, port} ->
      # close the sockets

    {:send, msg} ->
      # send an outgoing message
    end
  end
end
```

Step 1: receive new connections

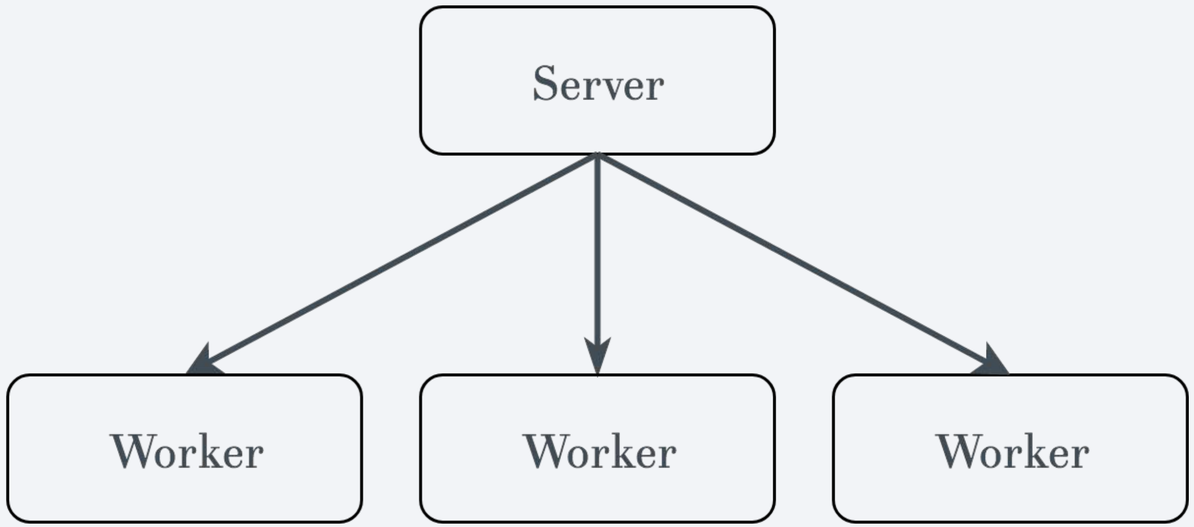
Step 2: accept and send messages

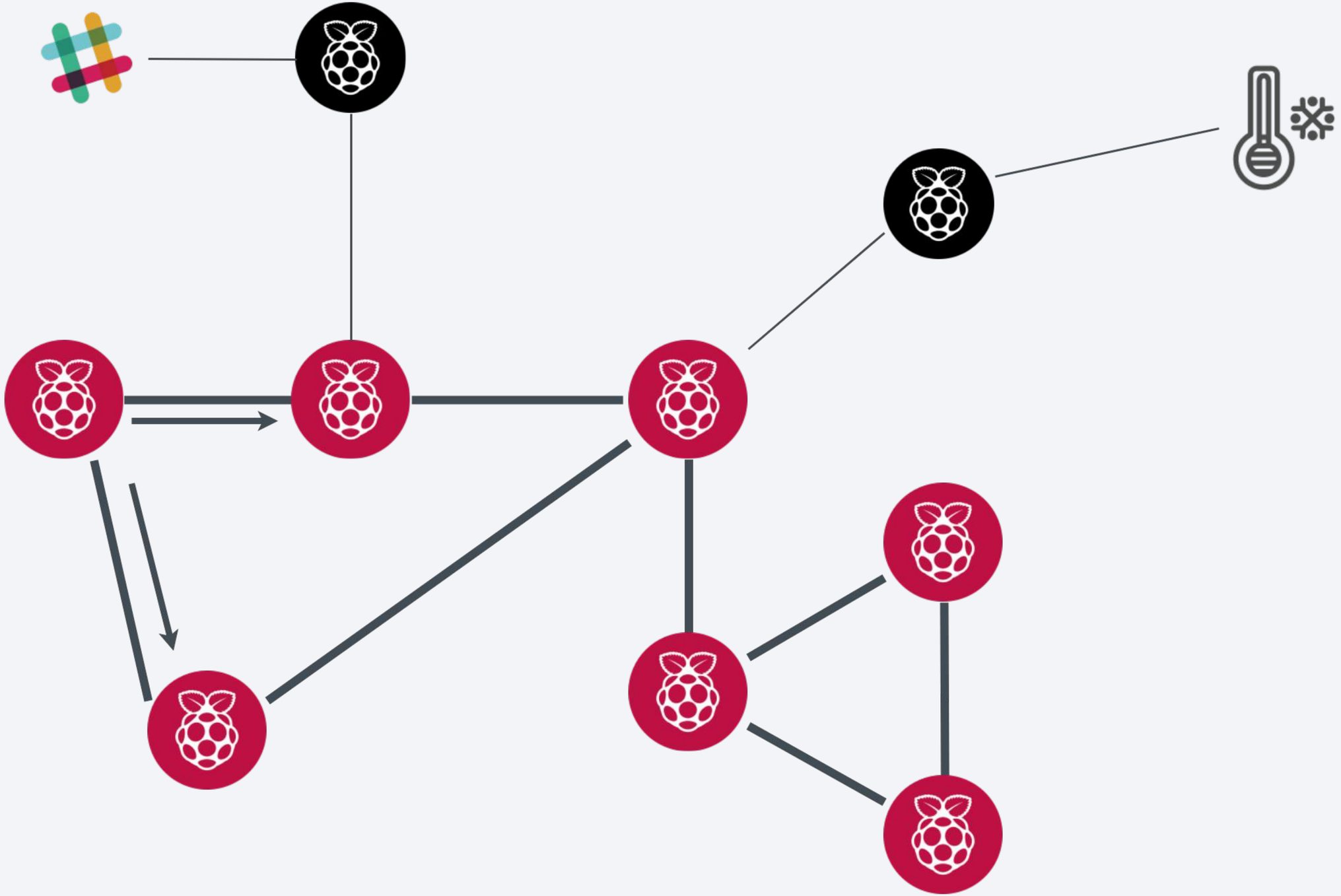
Step 3: do a bunch of Steps 1 and 2

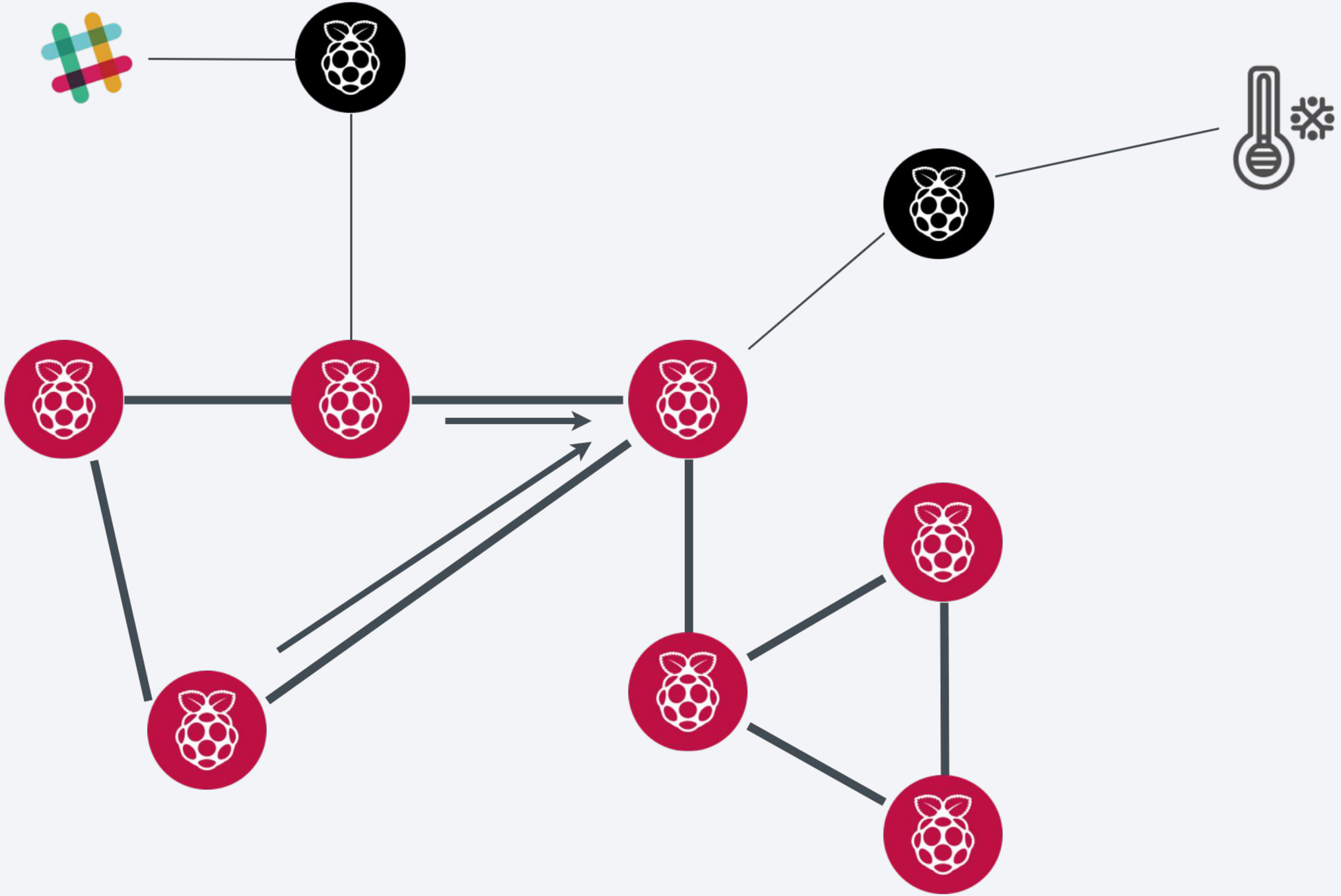
Raspberry Pi #1

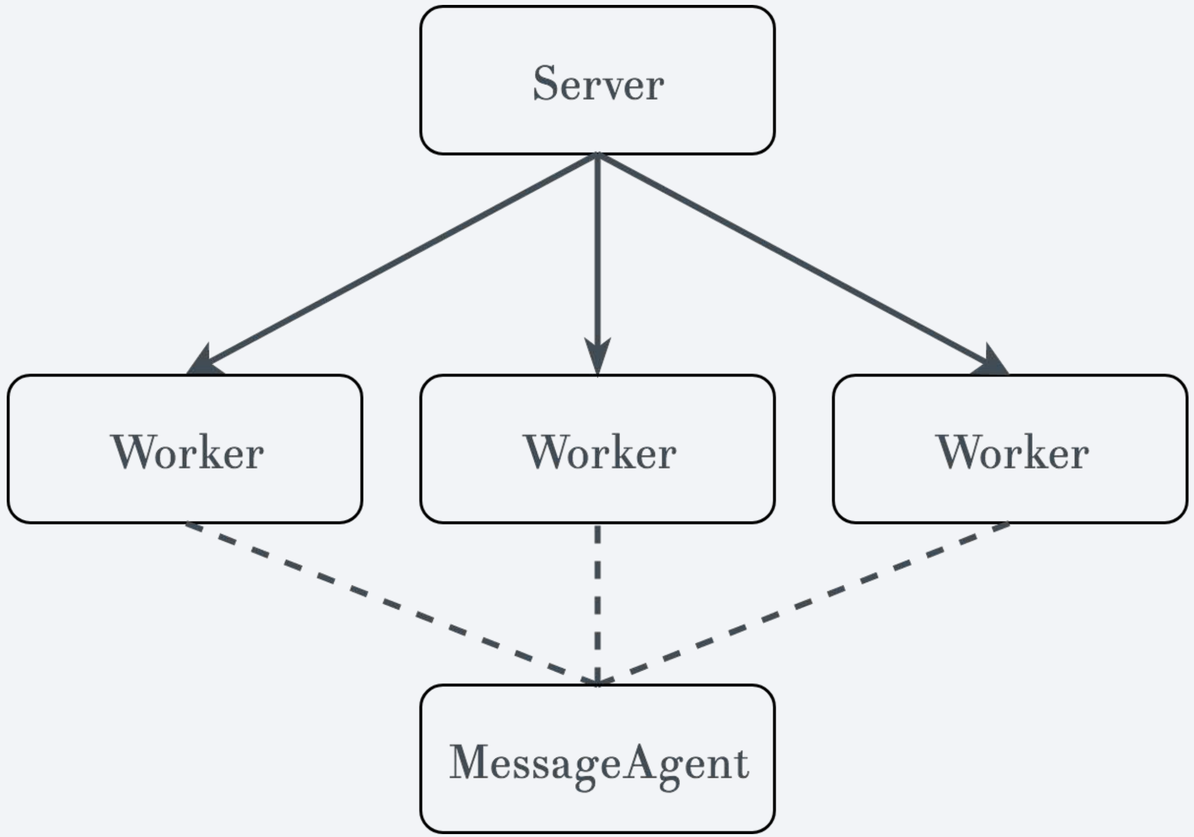


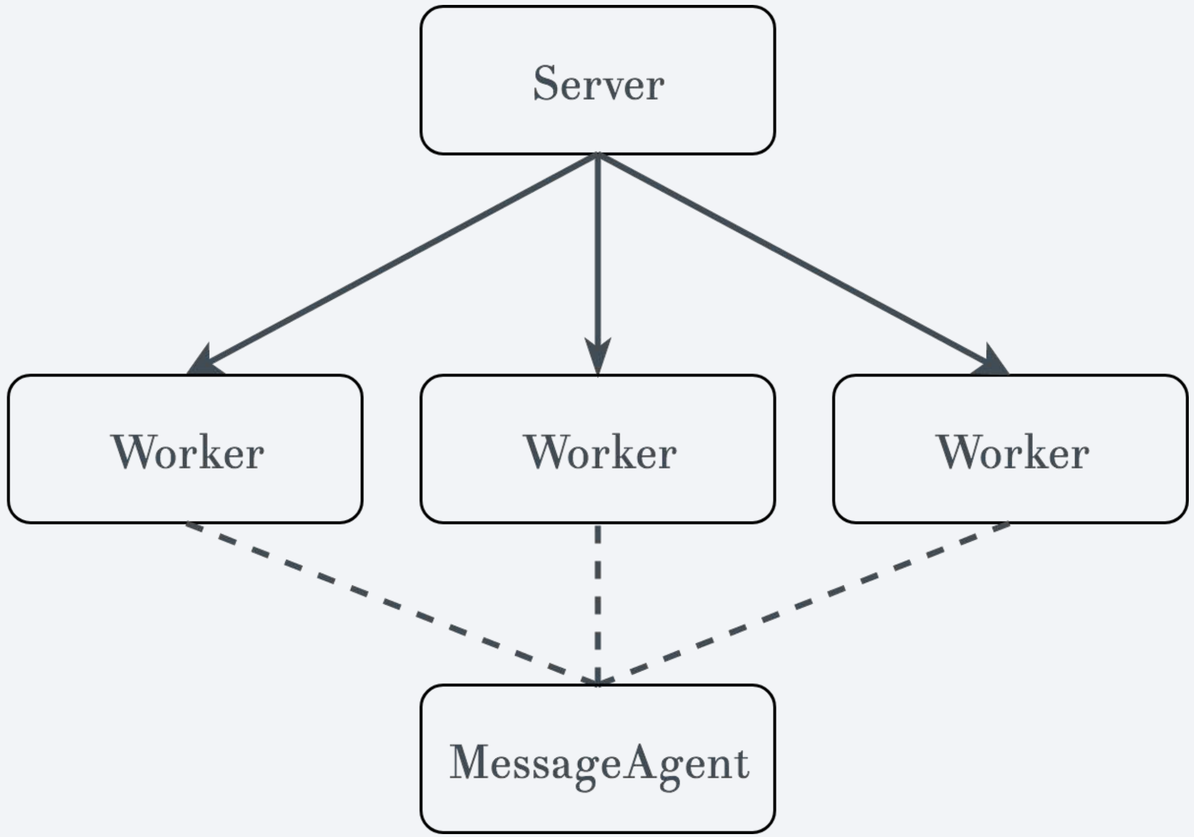
Raspberry Pi #2





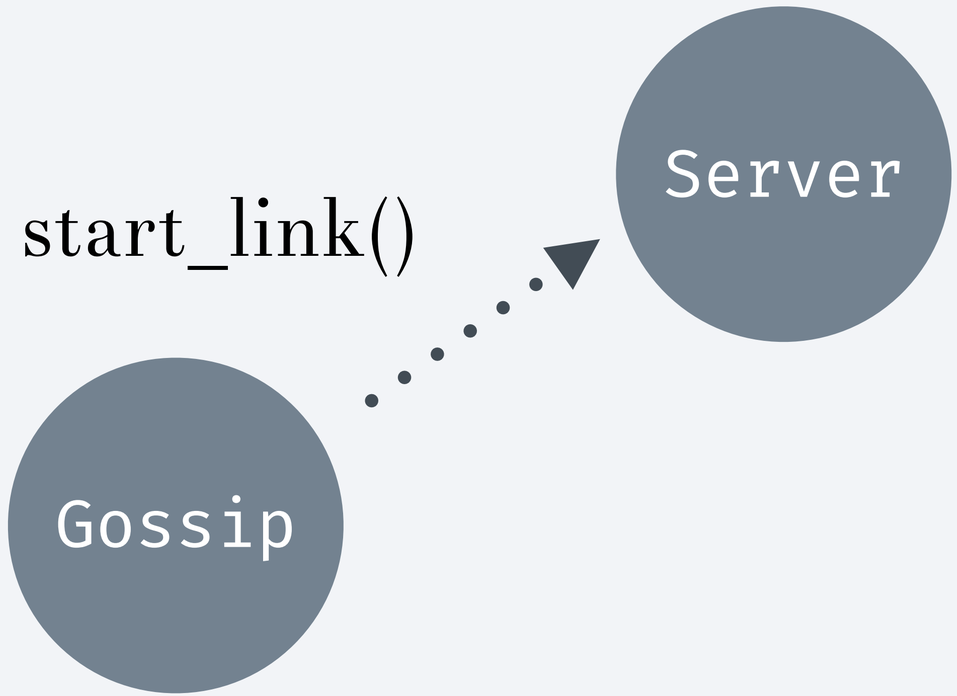



Server
Worker
Worker
Worker
MessageAgent

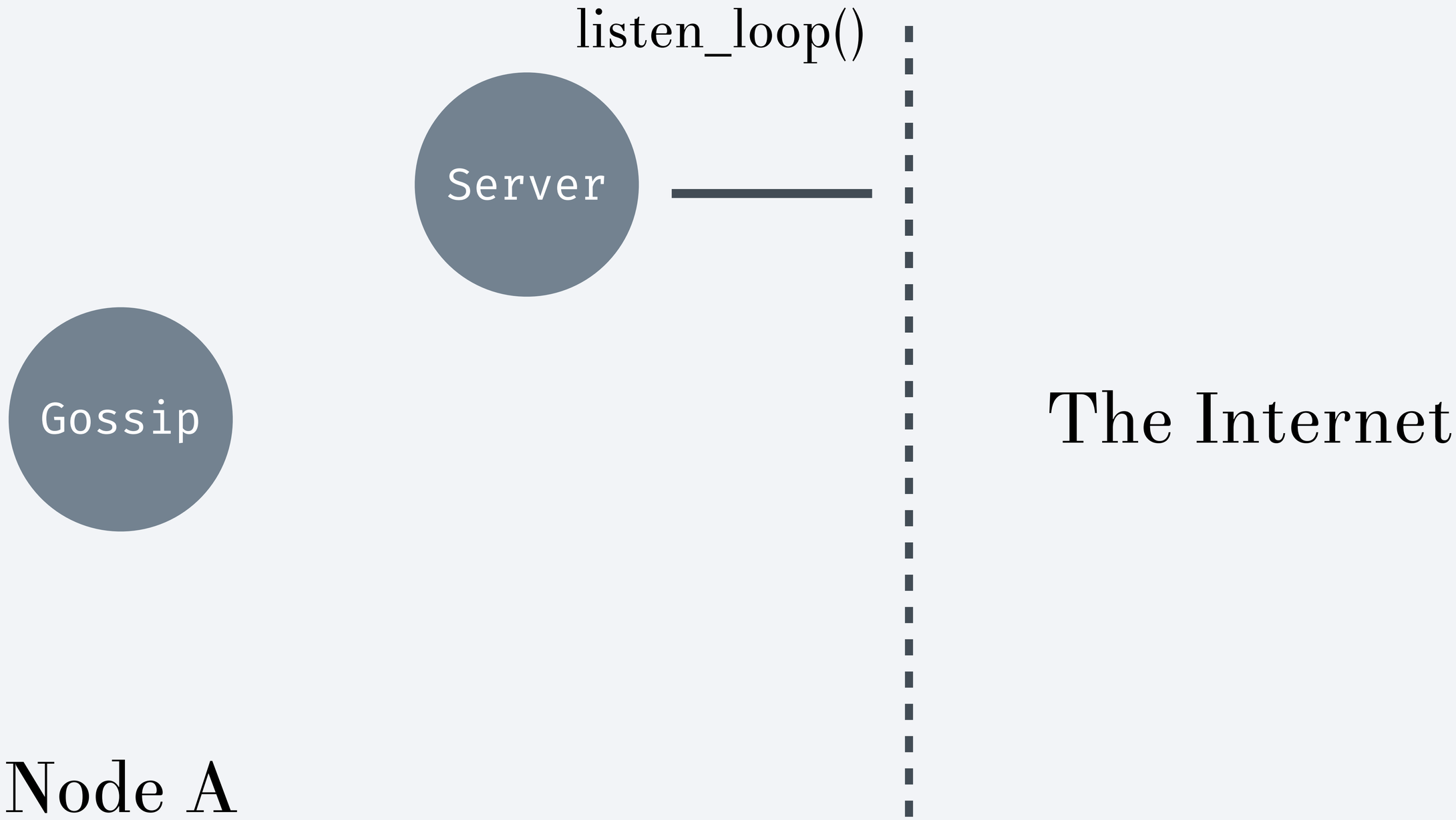

Server
Worker
Worker
Worker
MessageAgent

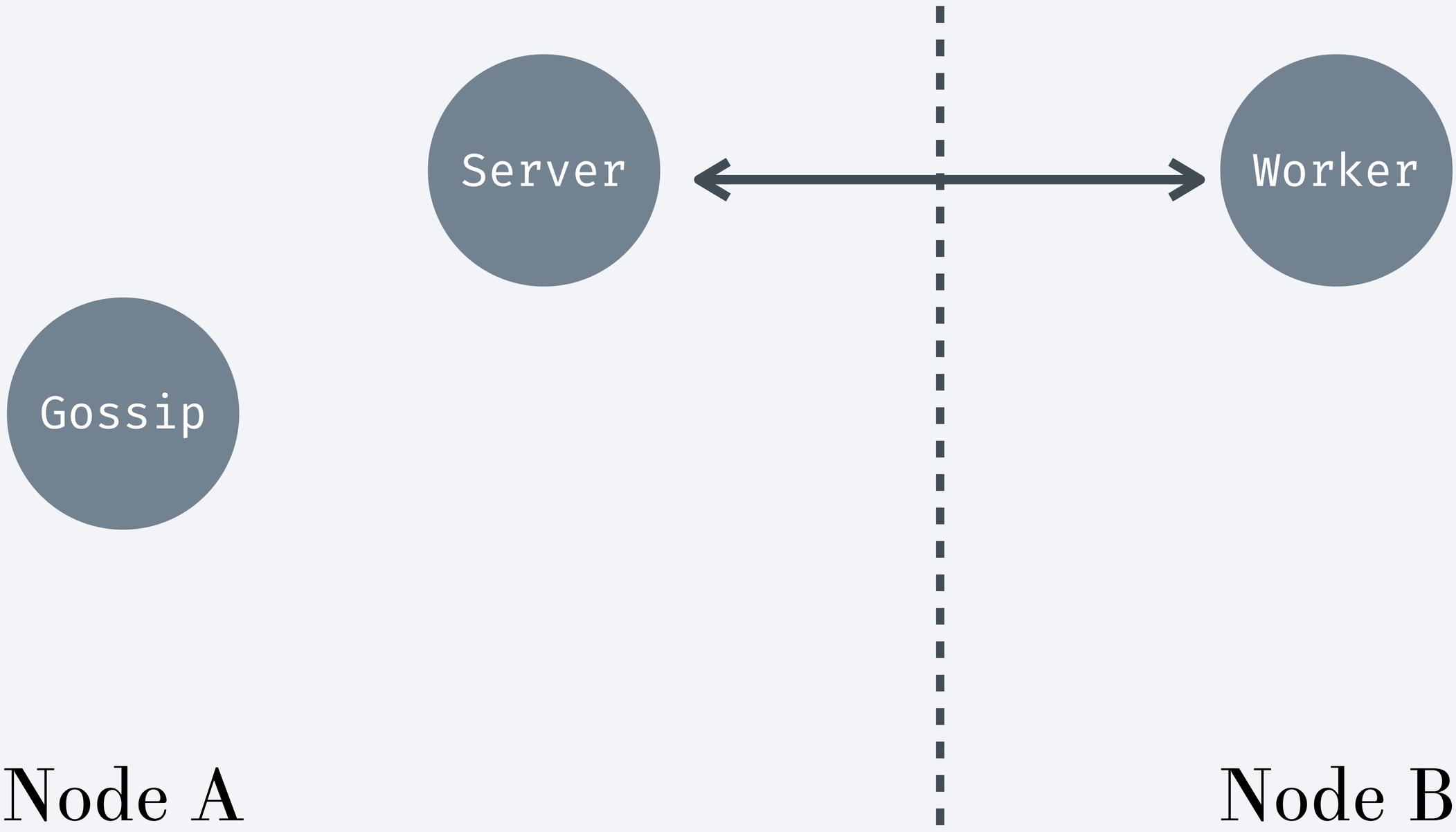
Testing

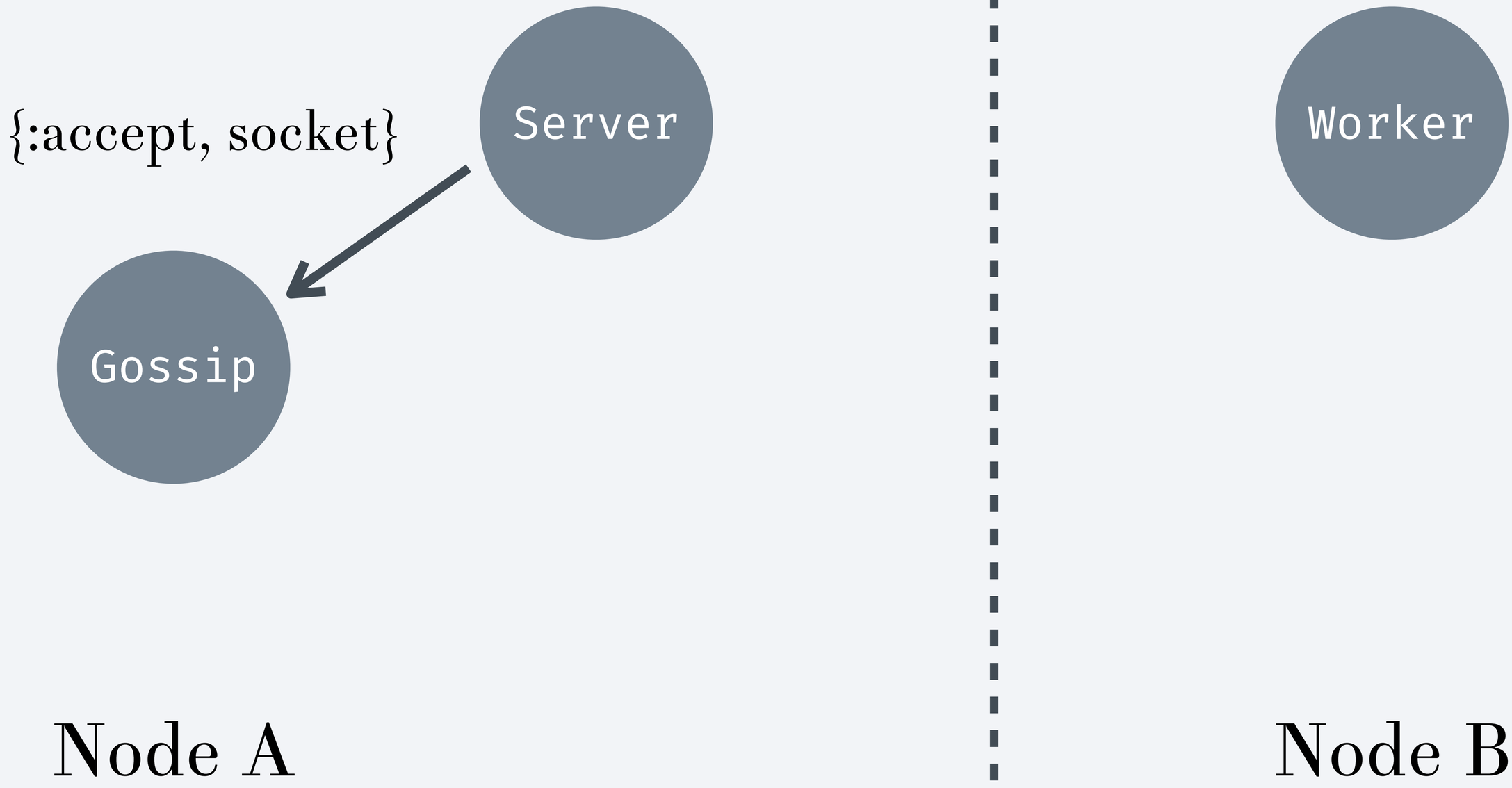
Node A

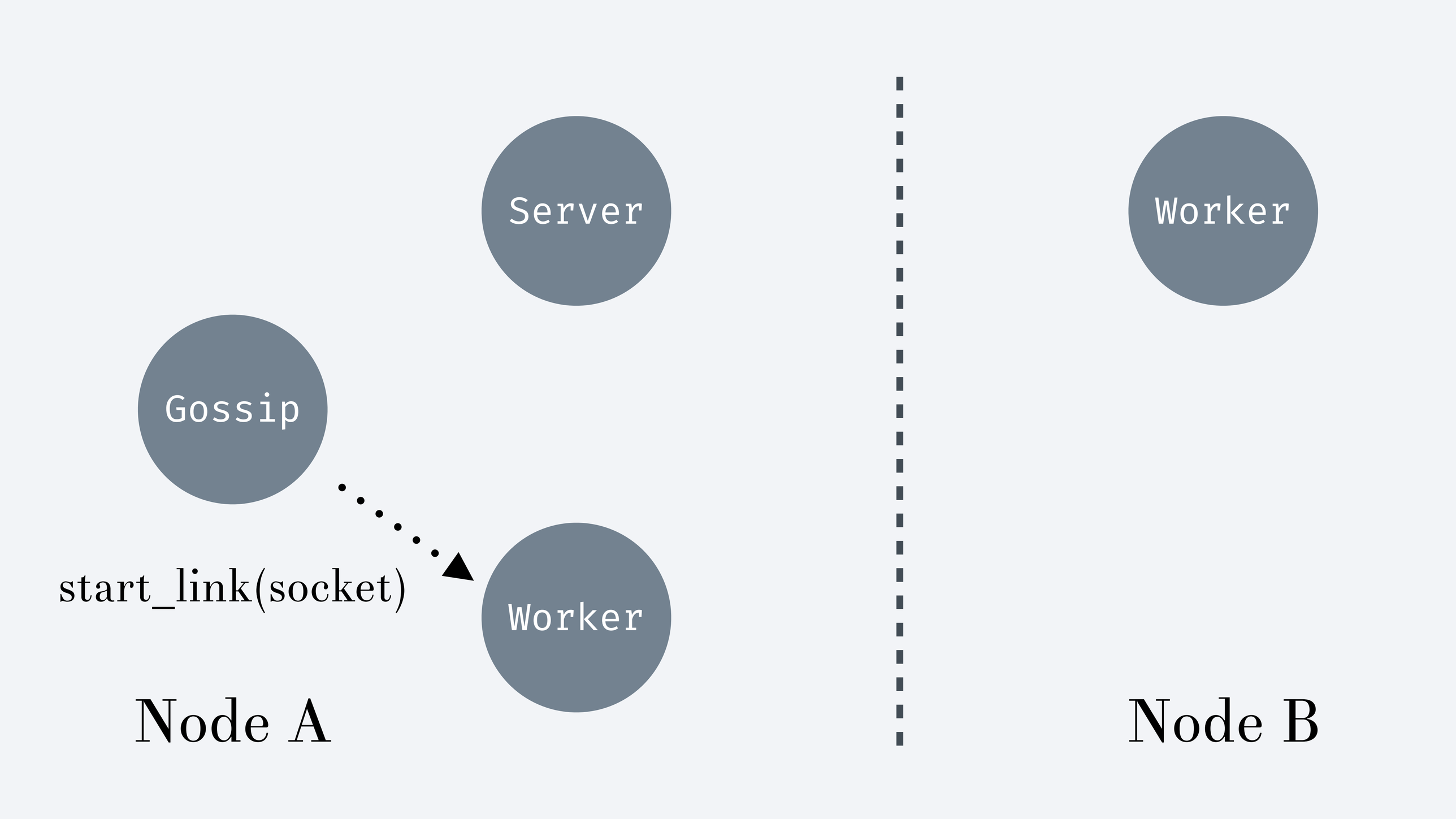


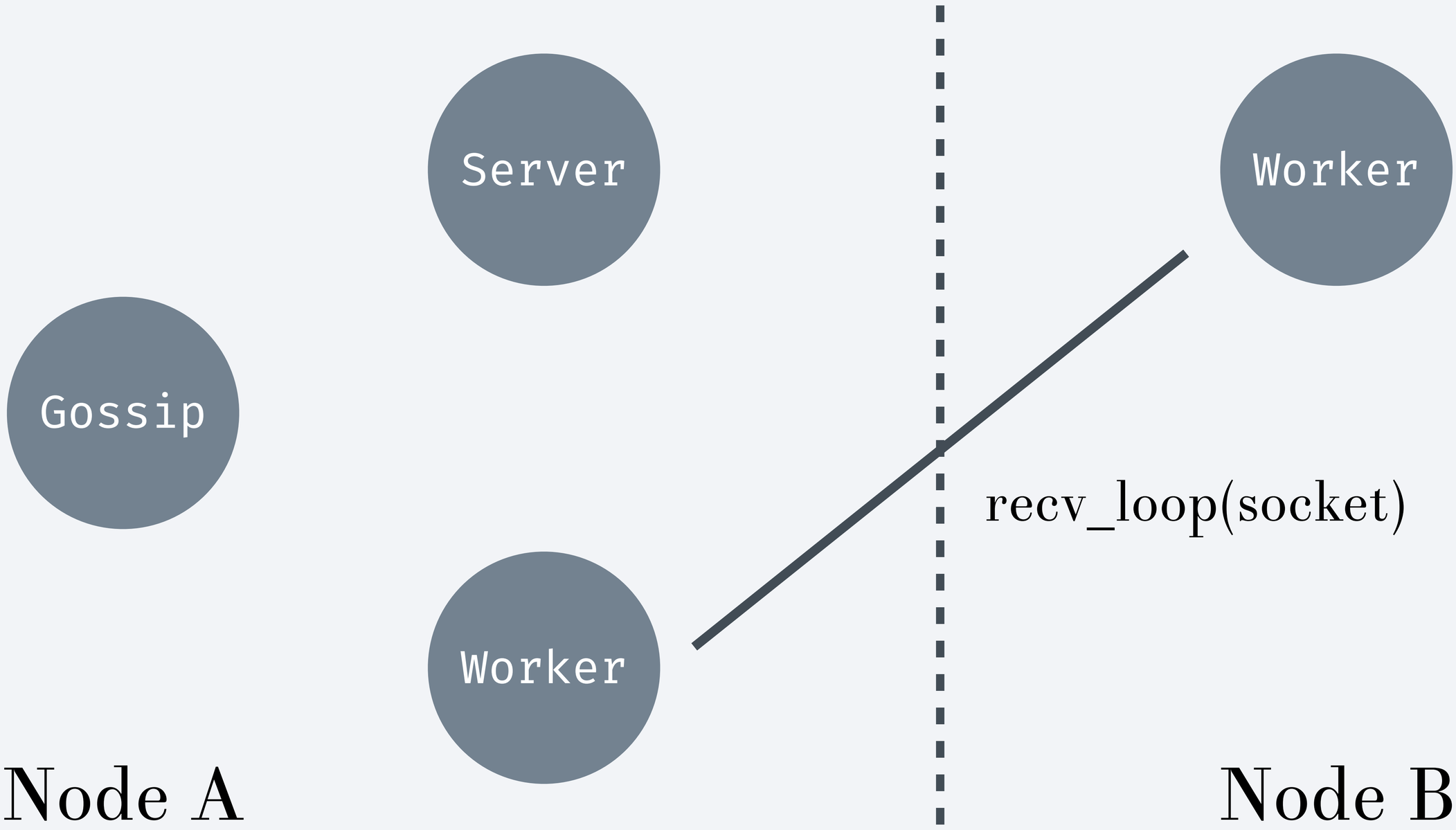
Node A

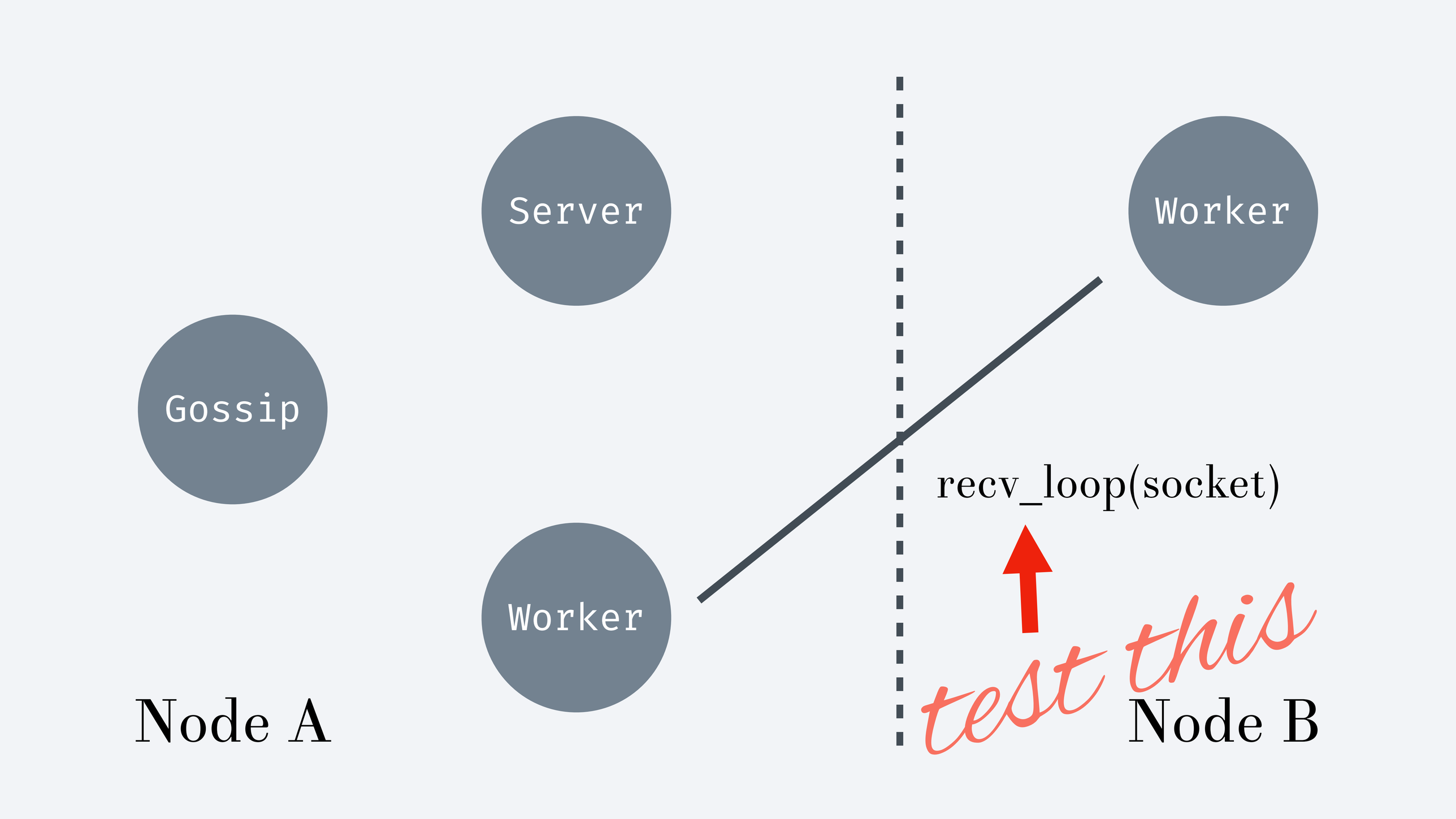

listen_loop()
Server
Gossip
The Internet
Node A


Server
Worker
Gossip
Node A
Node B


{:accept, socket}
Server
Gossip
Worker
Node A
Node B


Server
Worker
Gossip
start_link(socket)
Worker
Node A
Node B


Server
Worker
Gossip
recv_loop(socket)
Worker
Node A
Node B


Server
Worker
Gossip
recv_loop(socket)
Worker
test this
Node A
Node B

```elixir
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # echo the message

    {:tcp_closed, port} ->
      # close the sockets

    {:send, msg} ->
      # send the message
    end
  end
end
```

```
describe "recv_loop/2" do
  test "echoes :tcp messages" do
  end

  test "disconnects on :tcp_closed messages" do
  end

  test "sends a message on :send messages" do
  end
end
```

```
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # ...

    {:tcp_closed, port} ->
      # ...

    {:send, msg} ->
      # ...
    end
  end
end
```

gossip

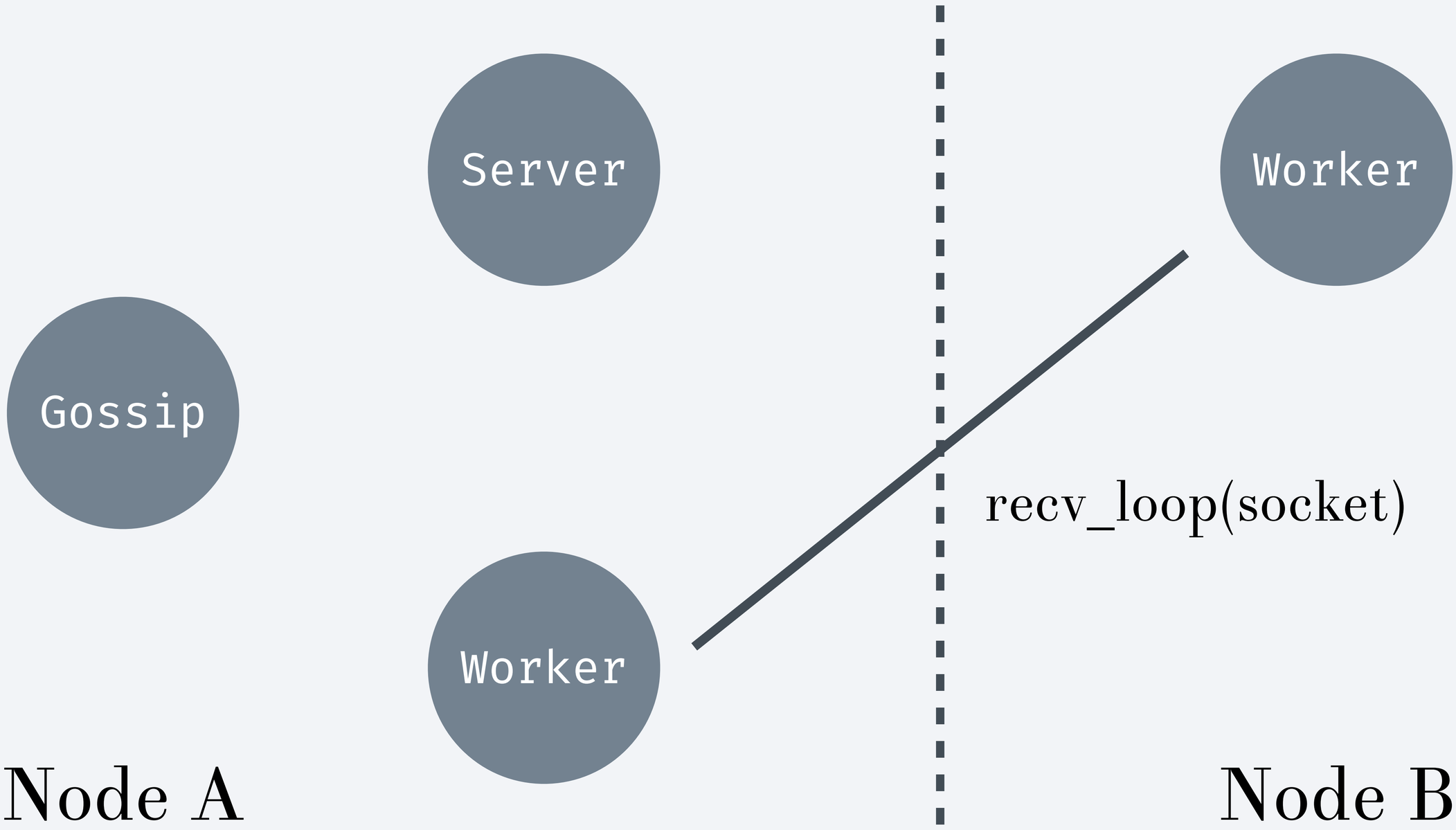
```
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # ...

    {:tcp_closed, port} ->
      # ...

    {:send, msg} ->
      # ...
    end
  end
end
```

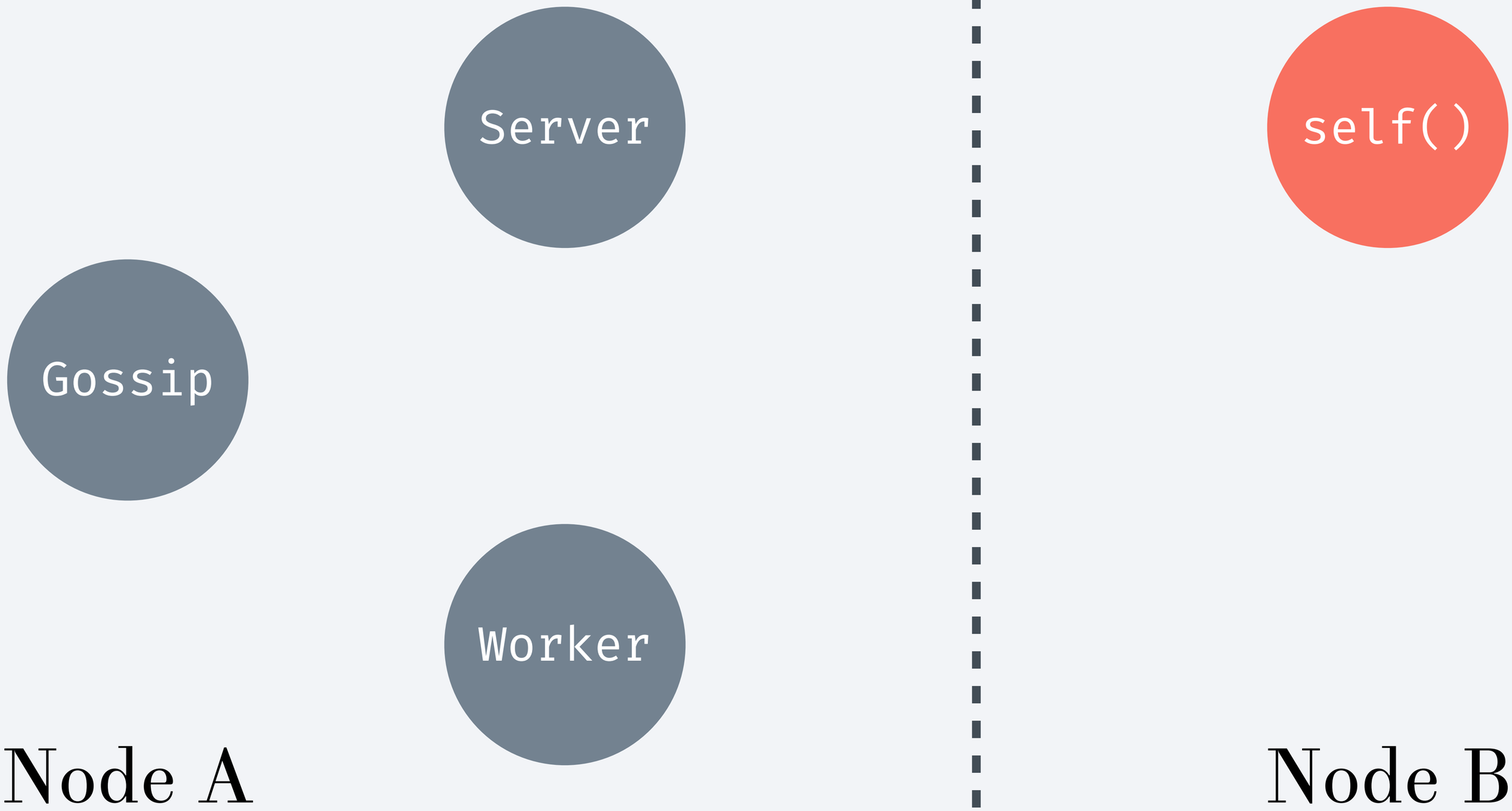
self()

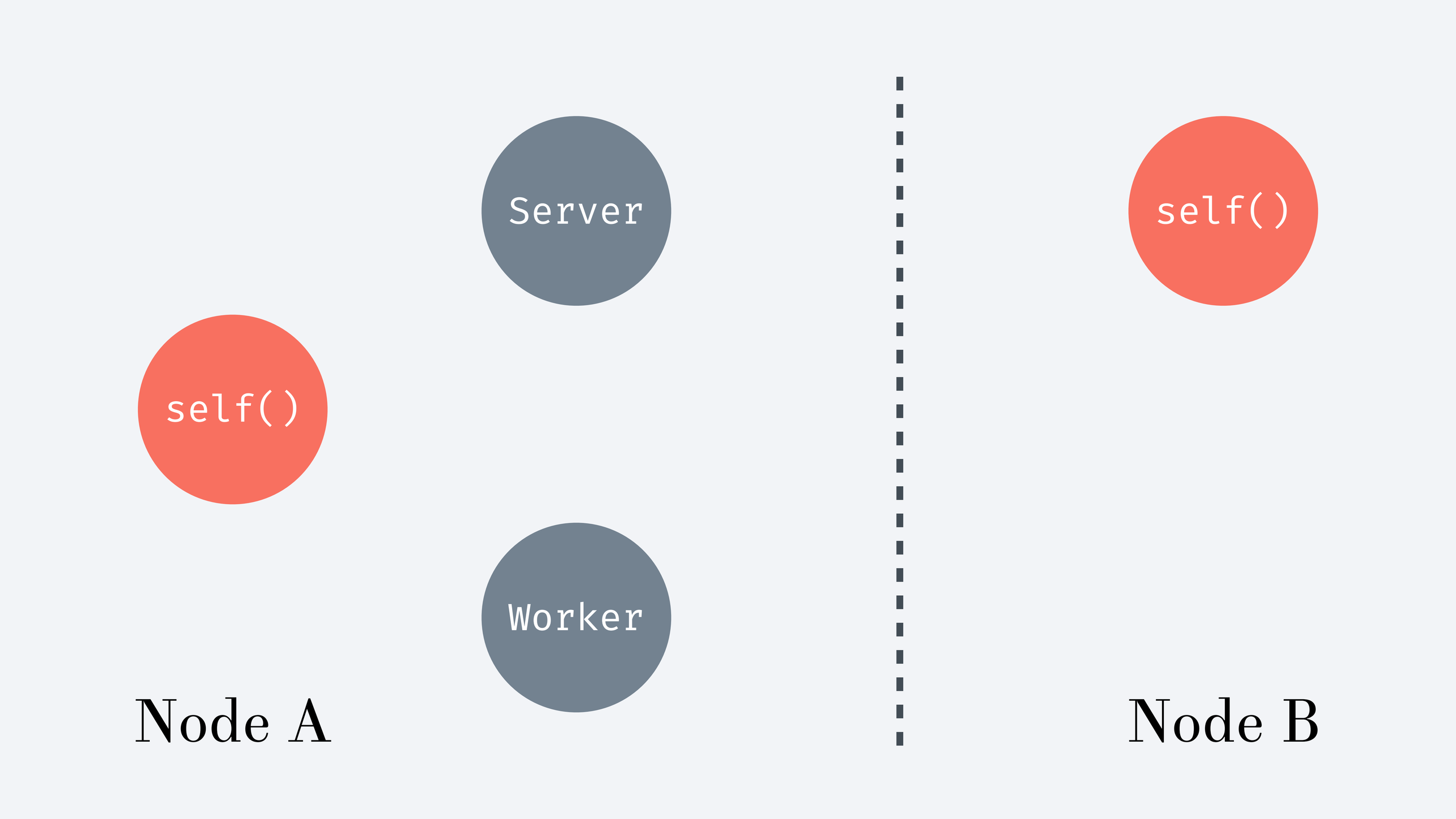
```
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # ...

    {:tcp_closed, port} ->
      # ...

    {:send, msg} ->
      # ...
    end
  end
end
```

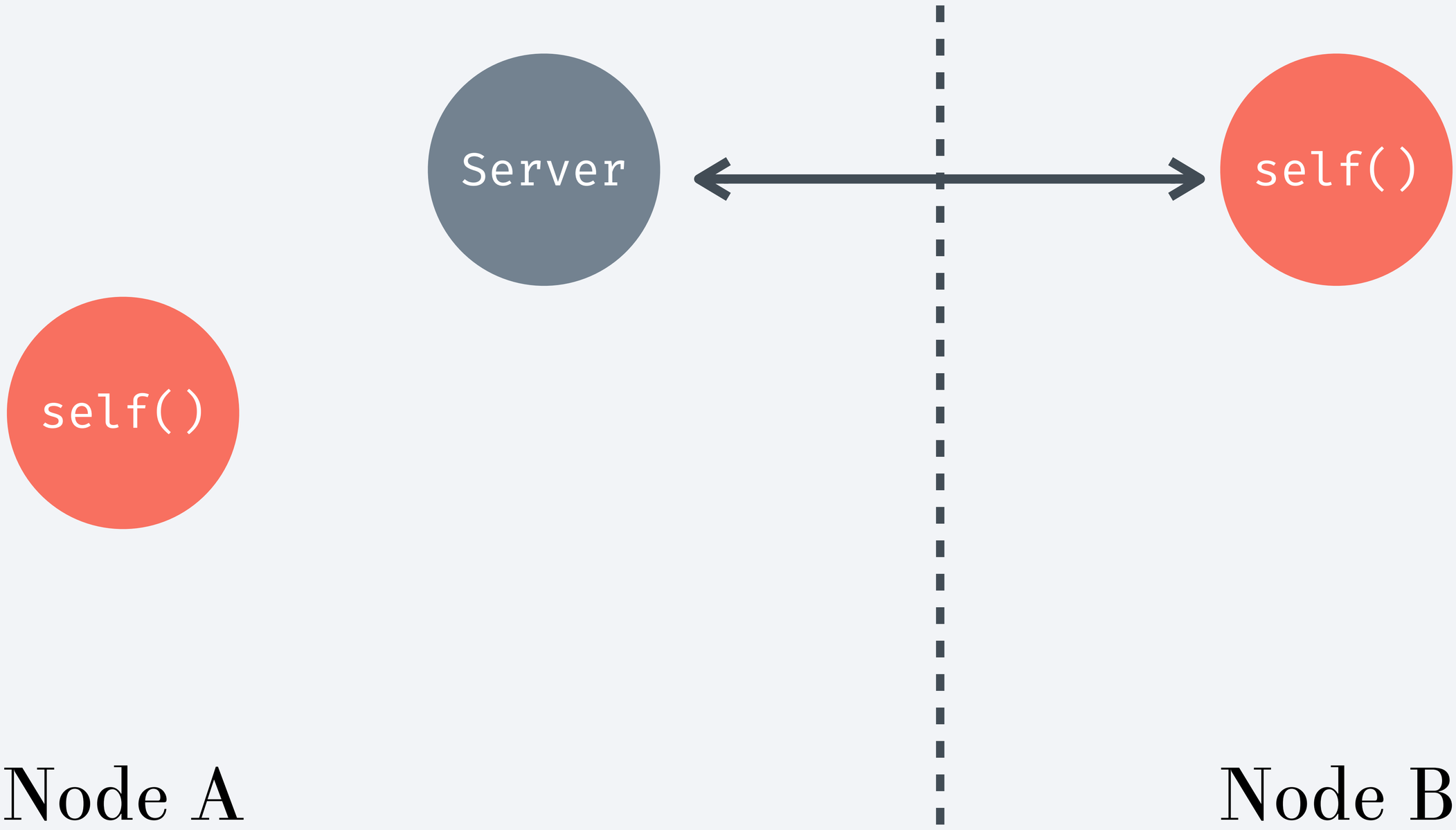
the test process

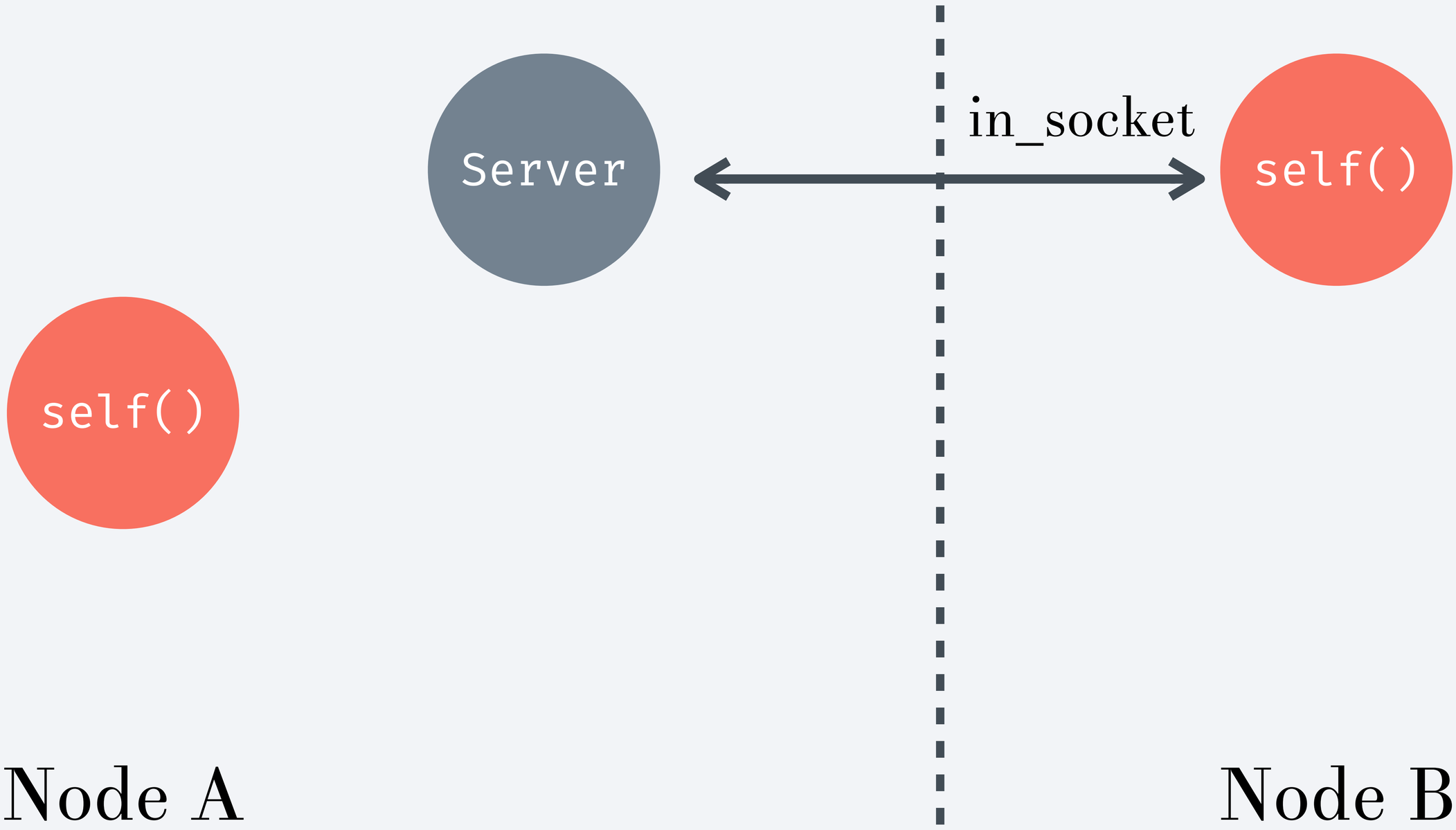
```
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # ...

    {:tcp_closed, port} ->
      # ...

    {:send, msg} ->
      # ...
    end
  end
end
```

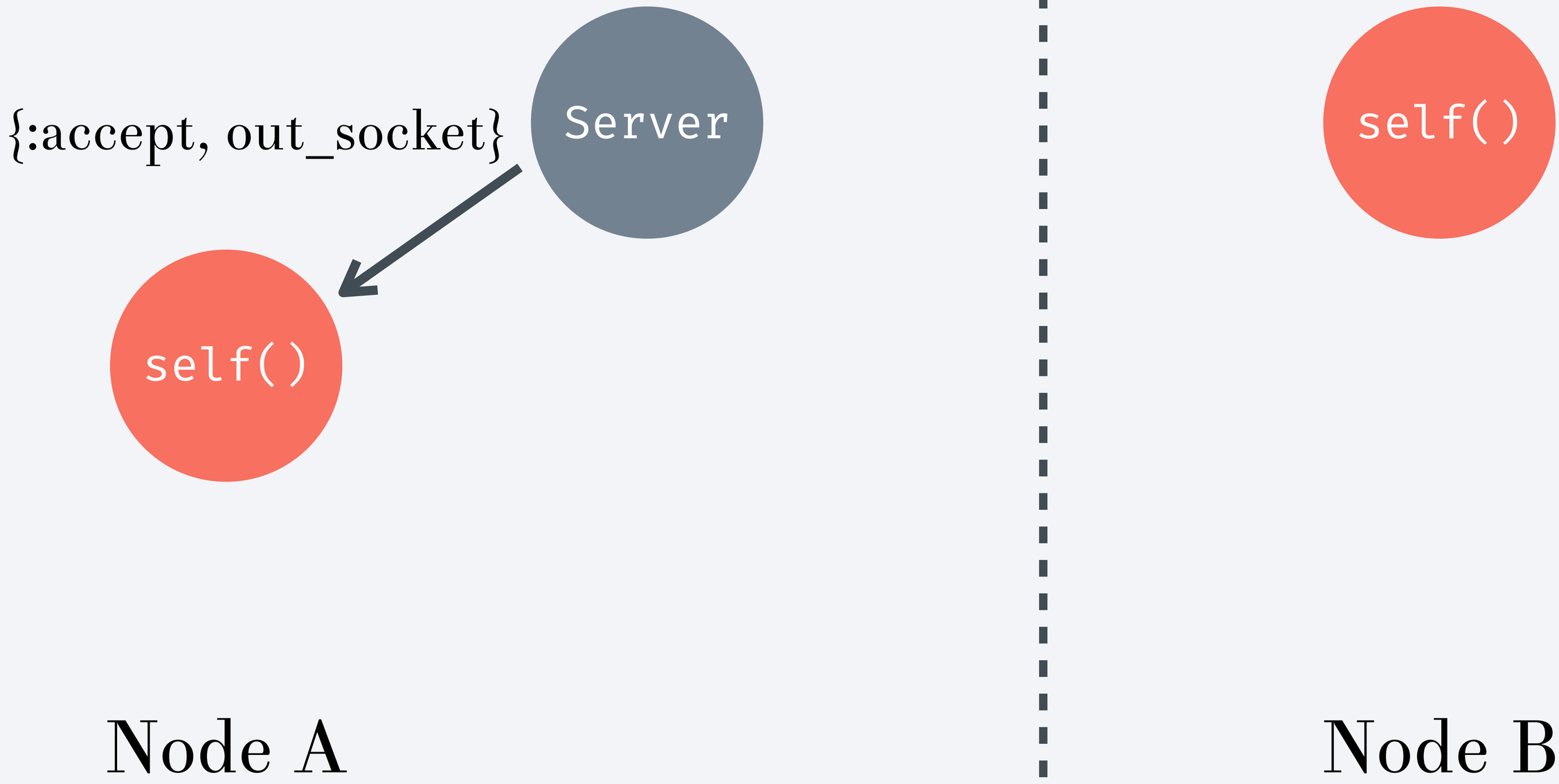

Server
Worker
Gossip
recv_loop(socket)
Worker
Node A
Node B

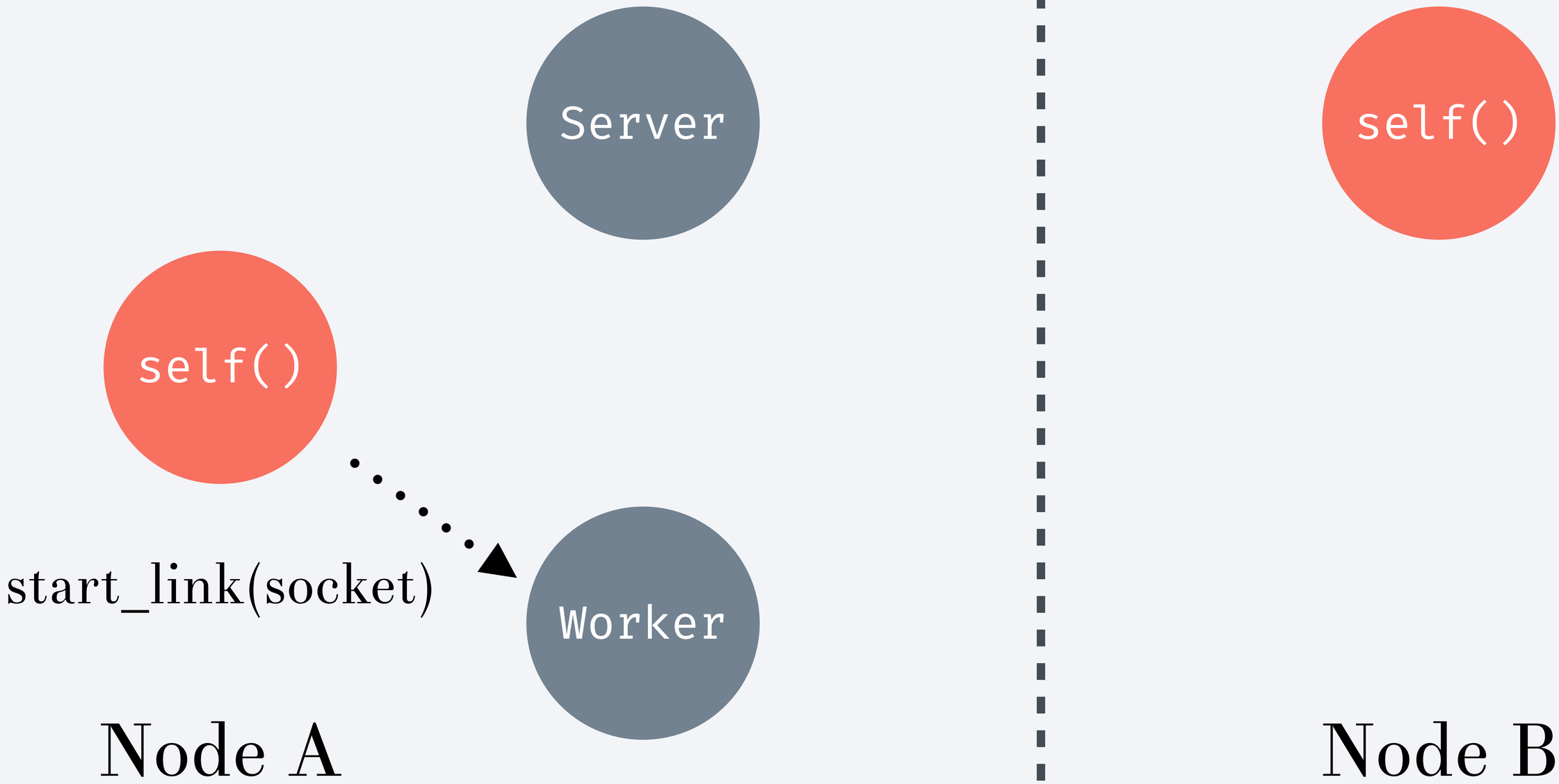

Server
self()
Gossip
Worker
Node A
Node B

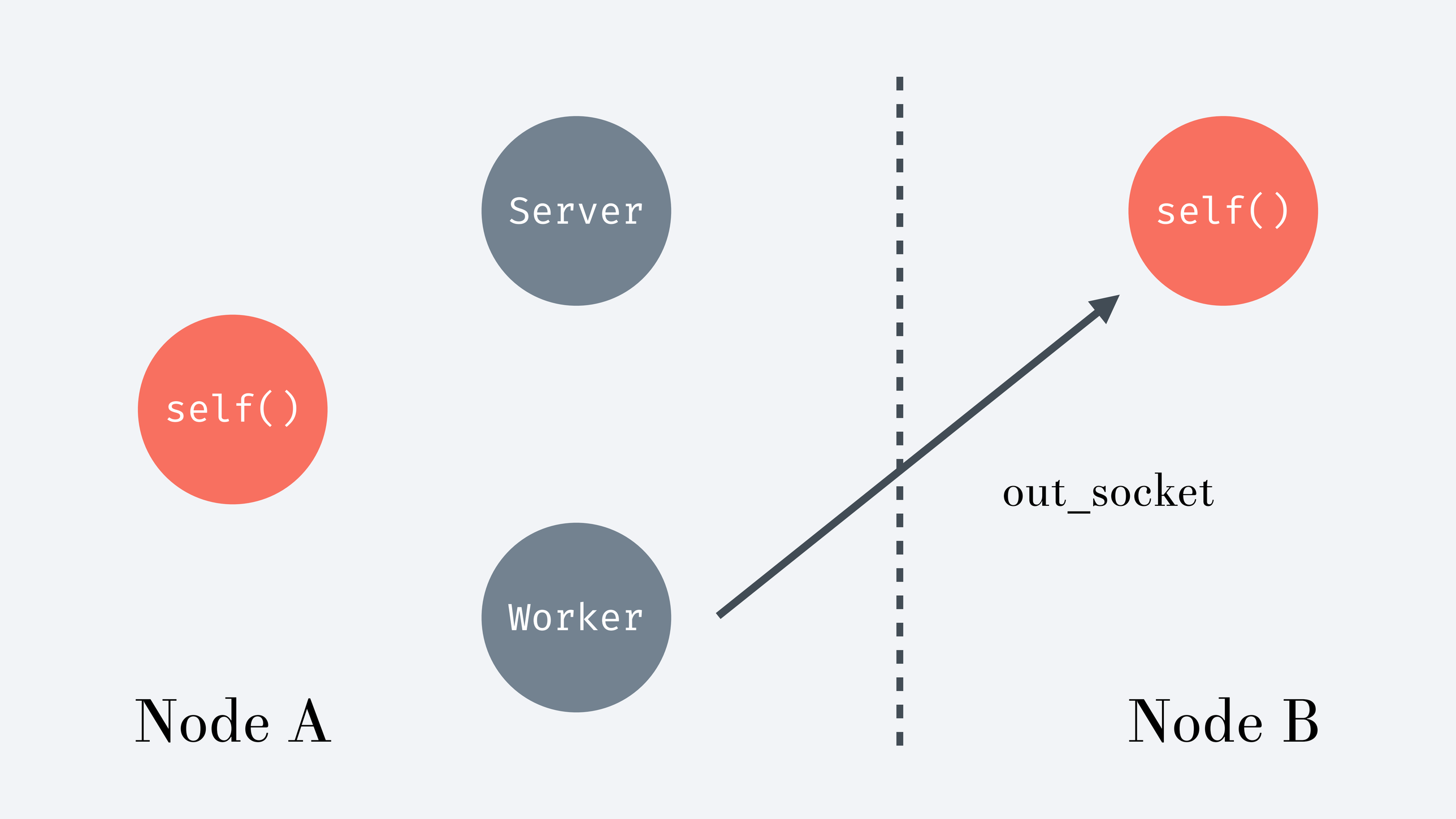

Server
self()
Worker
self()
Node A
Node B

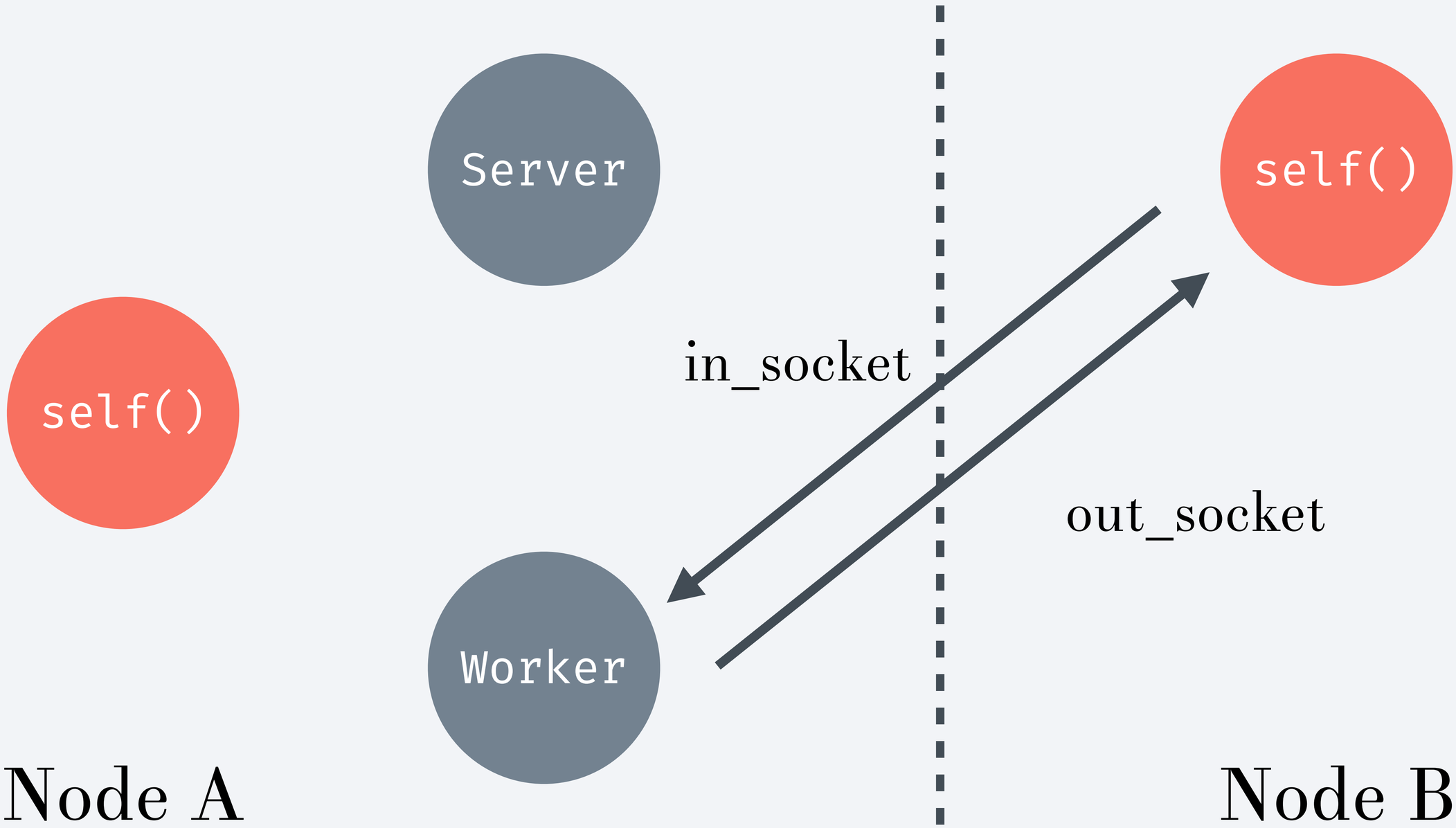

Server
self()
self()
Node A
Node B

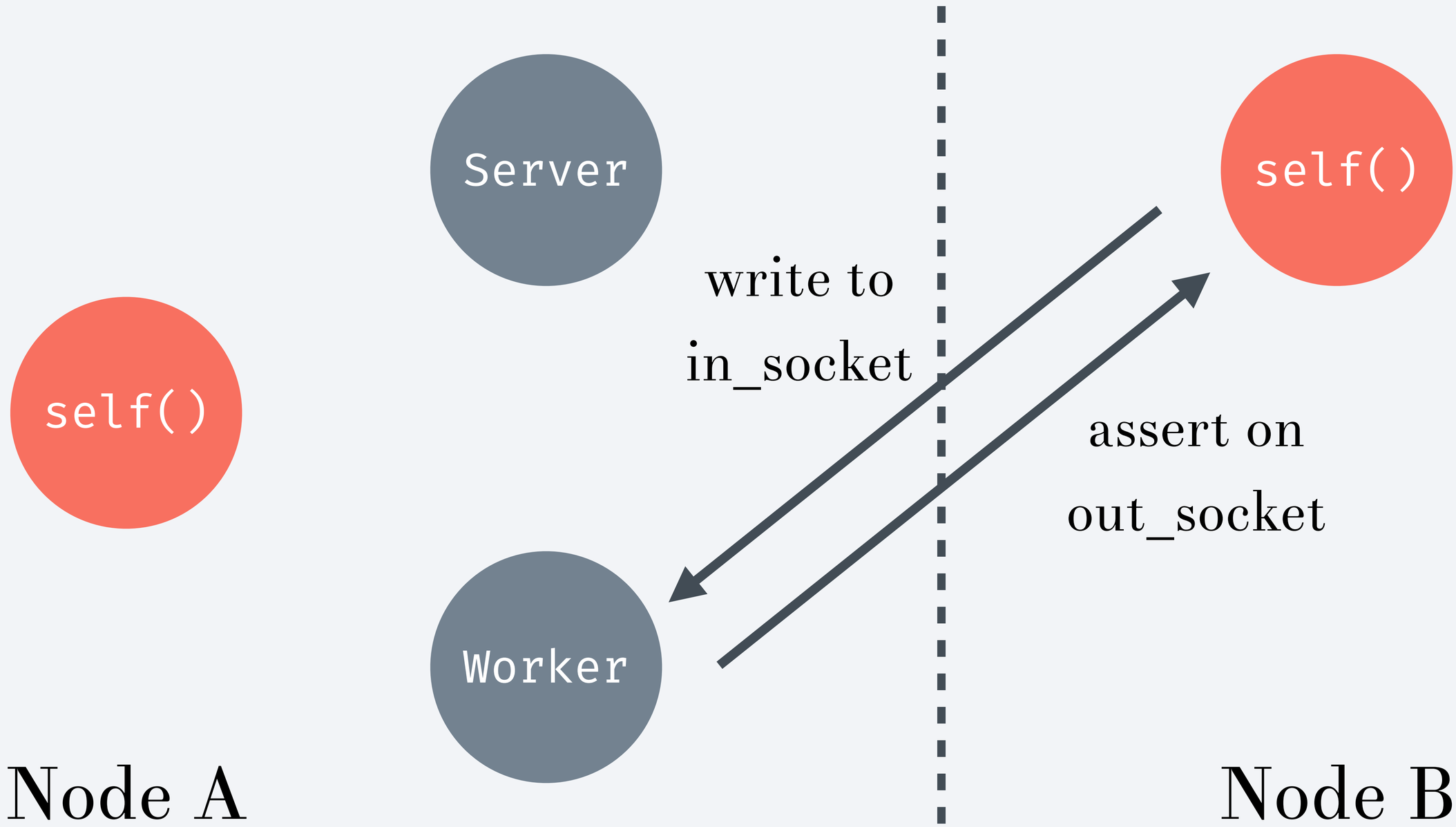

Server
in_socket
self()
self()
Node A
Node B


{:accept, out_socket}
Server
self()
self()
Node A
Node B


Server
self()
self()
start_link(socket)
Worker
Node A
Node B


Server
self()
self()
Worker
out_socket
Node A
Node B


Server
self()
self()
in_socket
out_socket
Worker
Node A
Node B


Server
self()
write to
in_socket
self()
assert on
out_socket
Worker
Node A
Node B

```elixir
def recv_loop(pid, socket) do
  receive do
    {:tcp, _port, msg} ->
      # ...

    {:tcp_closed, port} ->
      # ...

    {:send, msg} ->
      # ...
    end
  end
end
```

```
defp start_and_connect_to(port) do




end
```

```
defp start_and_connect_to(port) do
  Gossip.Server.start_link([self(), port])




end
```

```
defp start_and_connect_to(port) do
  Gossip.Server.start_link([self(), port])




end
```

```
defp start_and_connect_to(port) do
  Gossip.Server.start_link([self(), port])

  {:ok, in_socket} =
    :gen_tcp.connect('localhost', port, @socket_opts)



end
```

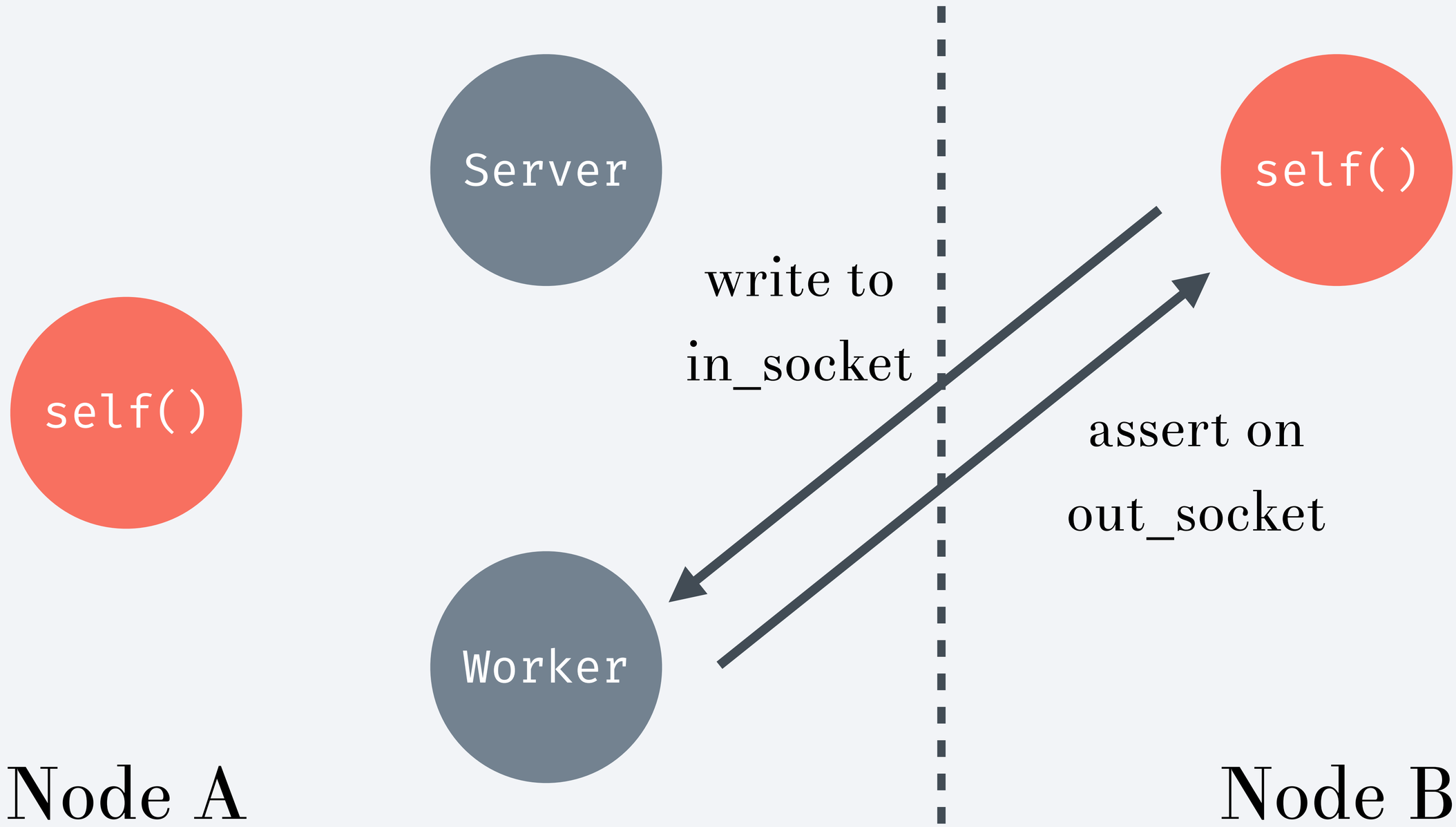
```
defp start_and_connect_to(port) do
  Gossip.Server.start_link([self(), port])

  {:ok, in_socket} =
    :gen_tcp.connect('localhost', port, @socket_opts)
  {:ok, out_socket} = receive_accept_msg()


end
```

```
defp receive_accept_msg do
  receive do
    {_, {:accept, out_socket}} -> {:ok, out_socket}
  after
    3_000 -> {:error, :timeout}
  end
end
```

```
defp start_and_connect_to(port) do
  Gossip.Server.start_link([self(), port])

  {:ok, in_socket} =
    :gen_tcp.connect('localhost', port, @socket_opts)
  {:ok, out_socket} = receive_accept_msg()


end
```

```
defp start_and_connect_to(port) do
  Gossip.Server.start_link([self(), port])

  {:ok, in_socket} =
    :gen_tcp.connect('localhost', port, @socket_opts)
  {:ok, out_socket} = receive_accept_msg()

  {in_socket, out_socket}
end
```

Mocking gives message control to your test process


Server
self()
write to
in_socket
self()
assert on
out_socket
Worker
Node A
Node B

```
describe "recv_loop/2" do
  test "echoes :tcp messages" do




  end
end
```

```
describe "recv_loop/2" do
  test "echoes :tcp messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)




  end
end
```

```
describe "recv_loop/2" do
  test "echoes :tcp messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)
    {:ok, worker} = start_worker(self(), out_socket)




  end
end
```

```elixir
describe "recv_loop/2" do
  test "echoes :tcp messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)
    {:ok, worker} = start_worker(self(), out_socket)

    send worker, {:tcp, in_socket, "hello"}


  end
end
```

```elixir
describe "recv_loop/2" do
  test "echoes :tcp messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)
    {:ok, worker} = start_worker(self(), out_socket)

    send worker, {:tcp, in_socket, “hello"}

    assert {:ok, "hello"} = :gen_tcp.recv(in_socket, 0)
  end
end
```

```
describe "recv_loop/2" do
  test "disconnects on :tcp_closed messages" do




  end
end
```

```
describe "recv_loop/2" do
  test "disconnects on :tcp_closed messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)
    {:ok, worker} = start_worker(self(), out_socket)

  end
end
```

```
describe "recv_loop/2" do
  test "disconnects on :tcp_closed messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)
    {:ok, worker} = start_worker(self(), out_socket)

    send worker, {:tcp_closed, out_socket}




  end
end
```

```elixir
describe "recv_loop/2" do
  test "disconnects on :tcp_closed messages" do
    {in_socket, out_socket} = start_and_connect_to(3000)
    {:ok, worker} = start_worker(self(), out_socket)

    send worker, {:tcp_closed, out_socket}

    # assert the sockets are closed
    assert {:error, :closed} = :gen_tcp.recv(in_socket, 0)
    assert {:error, :closed} = :gen_tcp.recv(out_socket, 0)
    assert_receive {_, {:disconnect, ^worker}}
  end
end
```

Avoid named processes

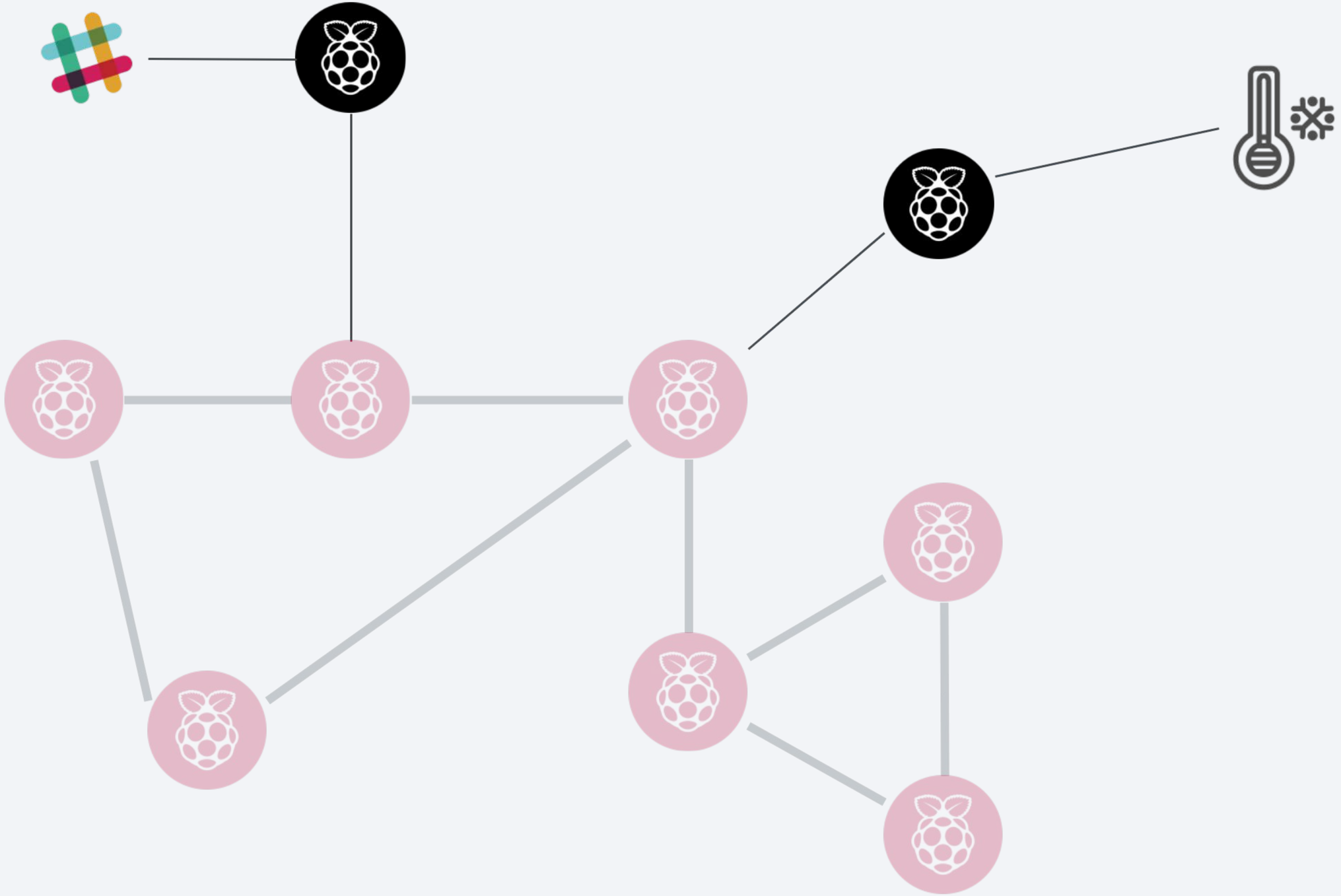
Inject self() into any functions that send messages

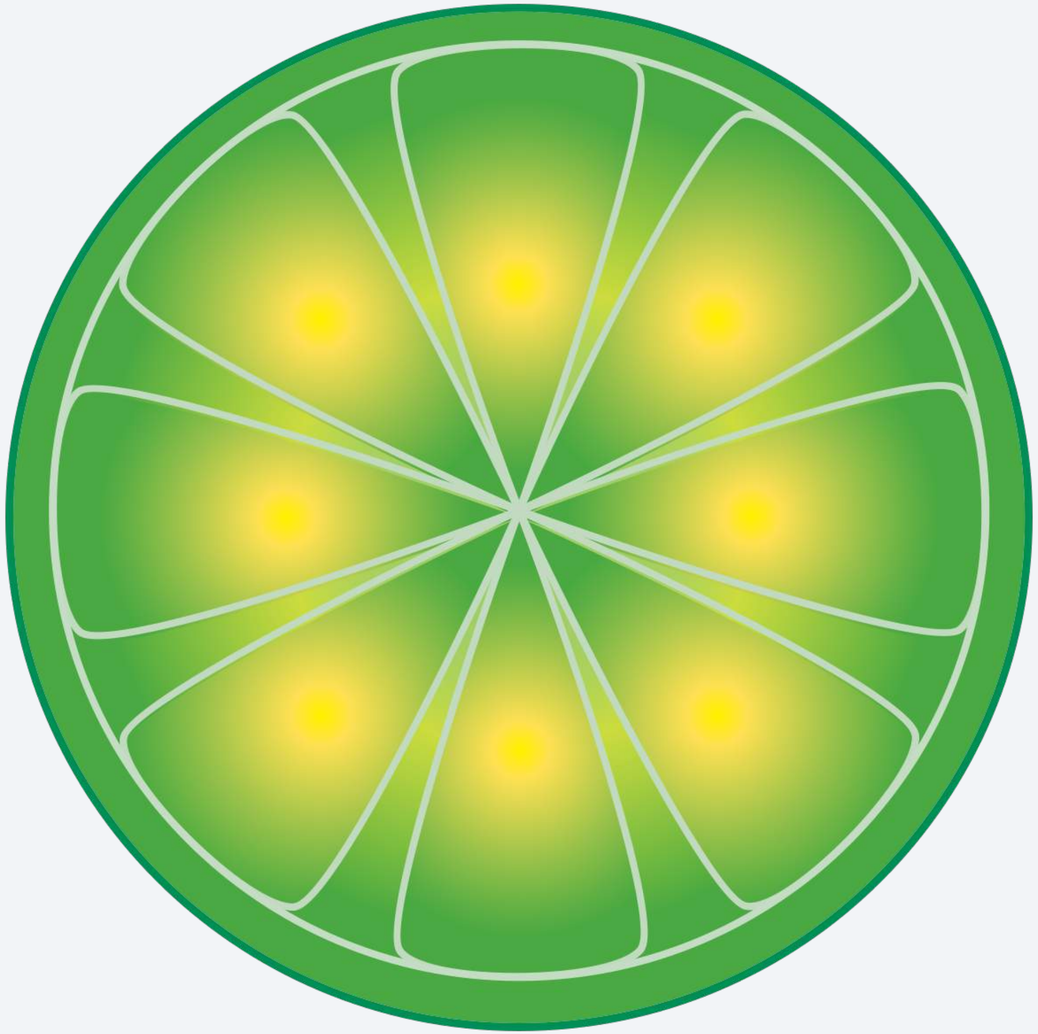
Test the invoked functions directly

Test the handle_* functions

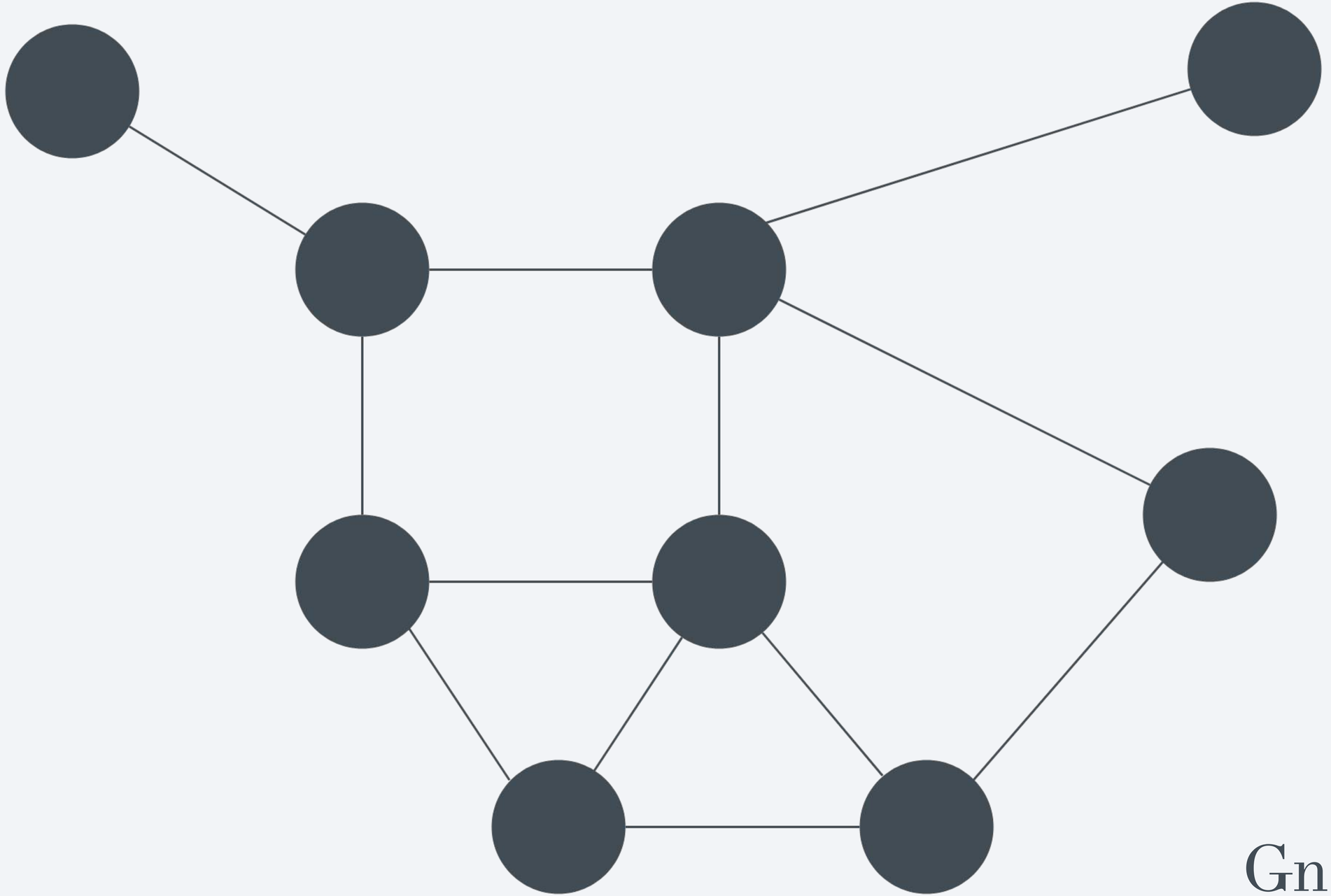
Play around with messages

“Does it scale?”

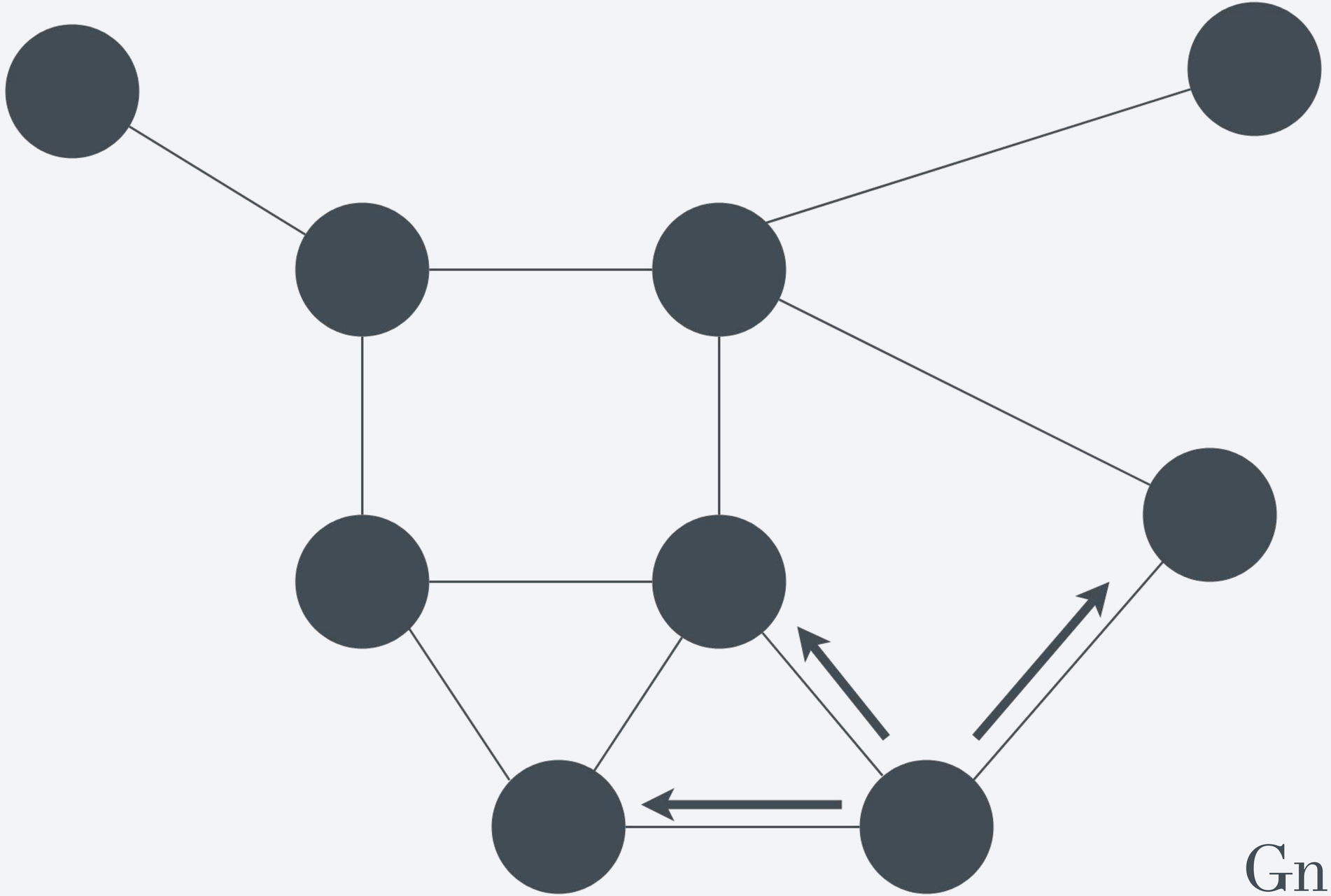




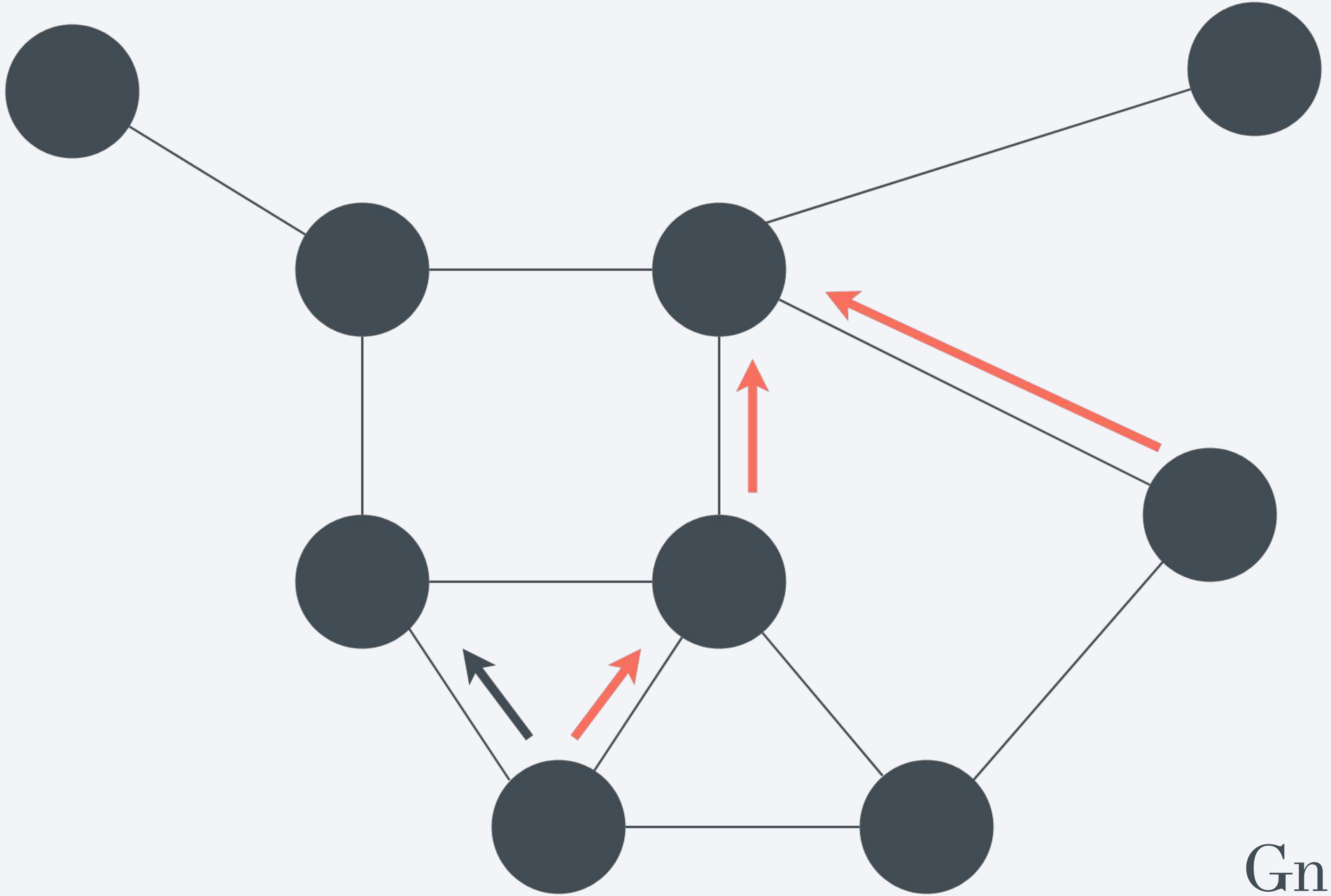
g



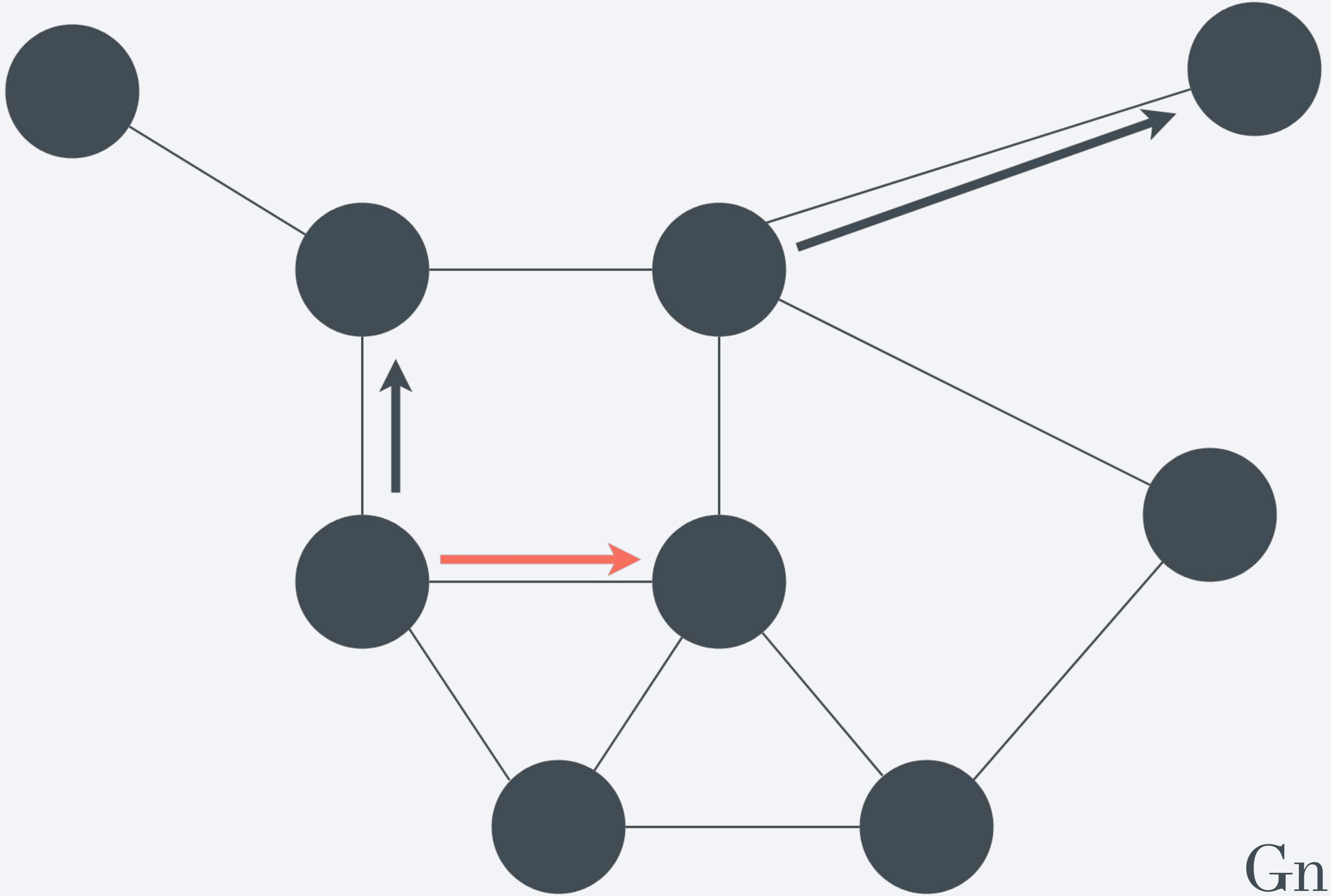
Gnutella



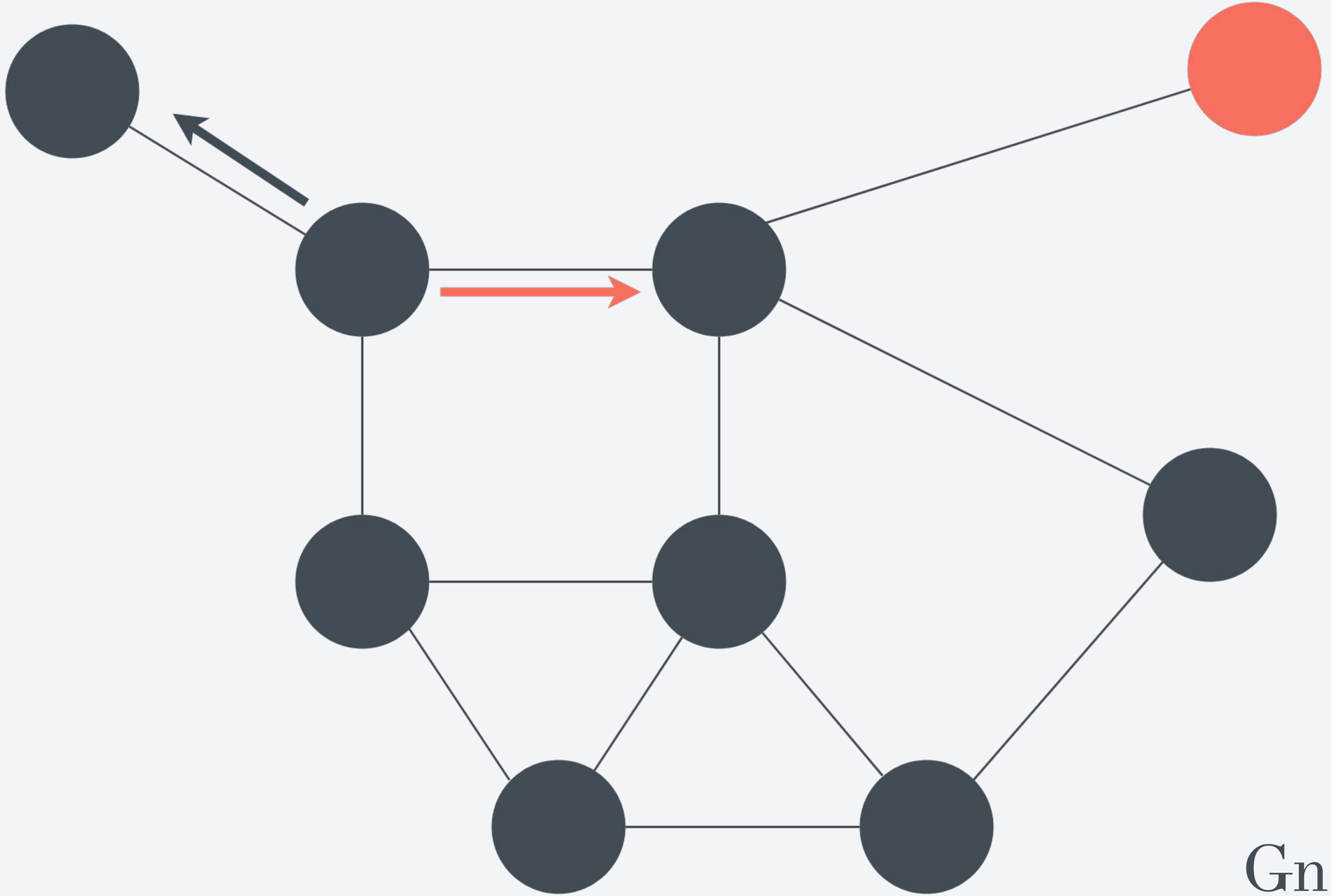
Gnutella



Gnutella



Gnutella

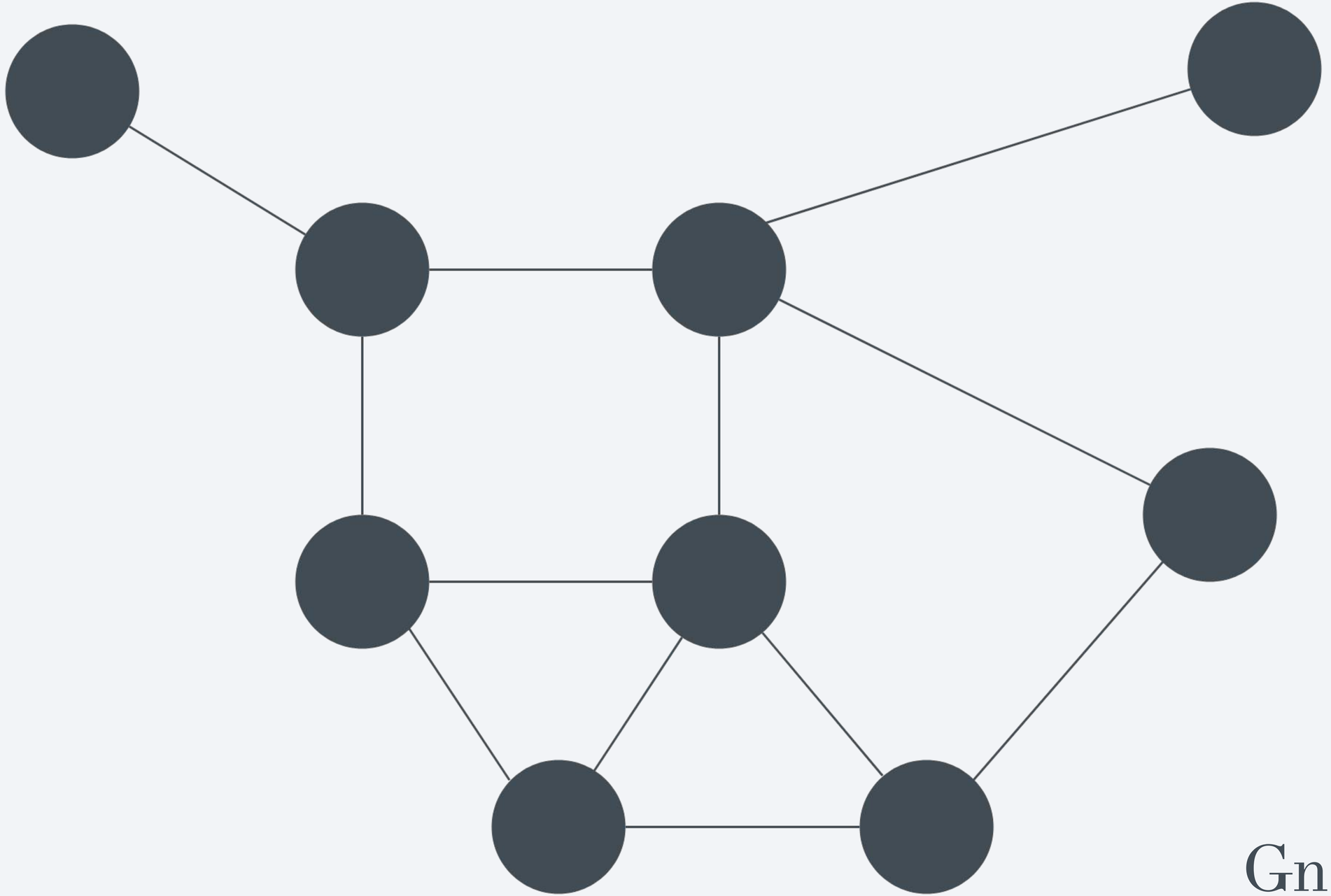


Gnutella

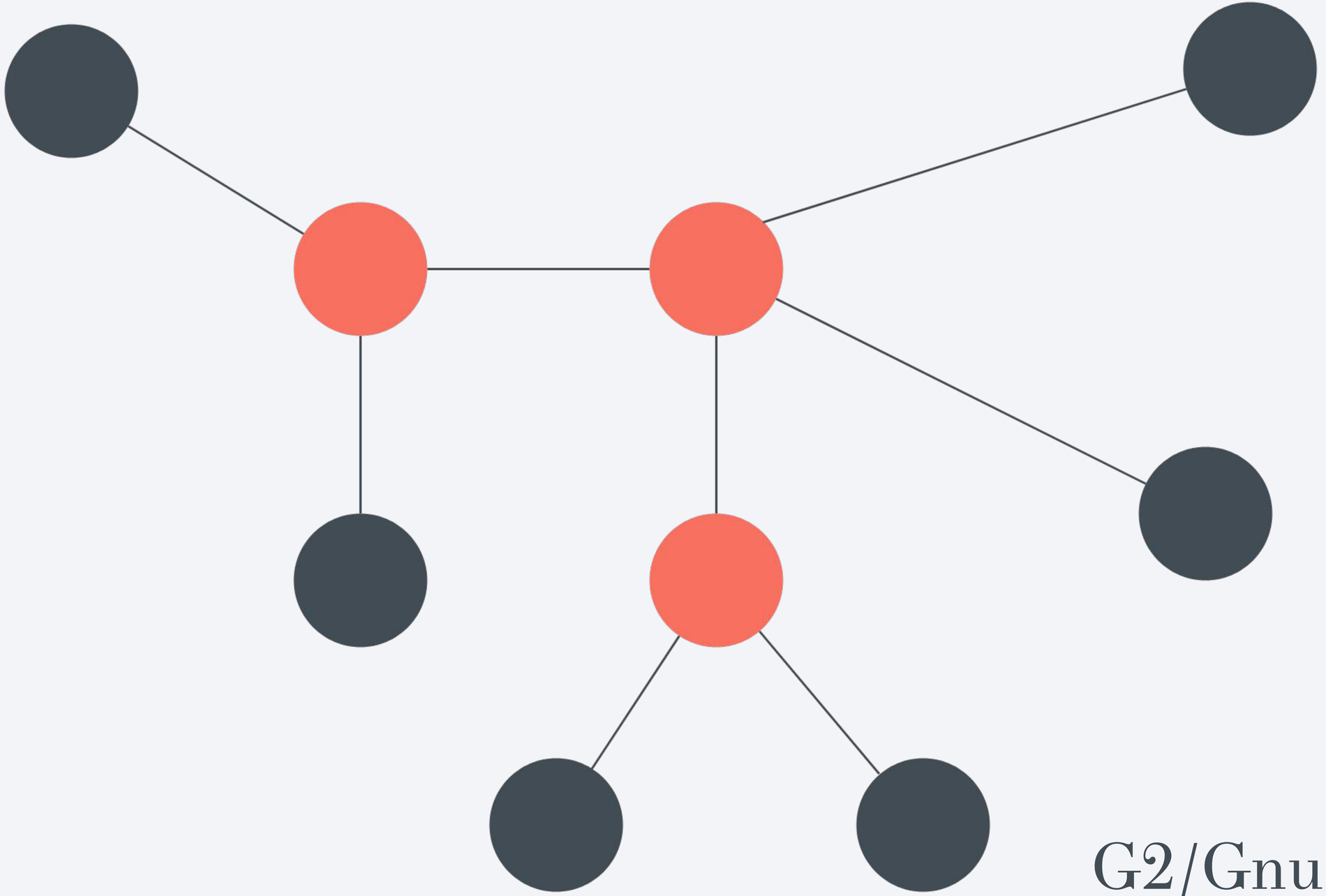
g

g 

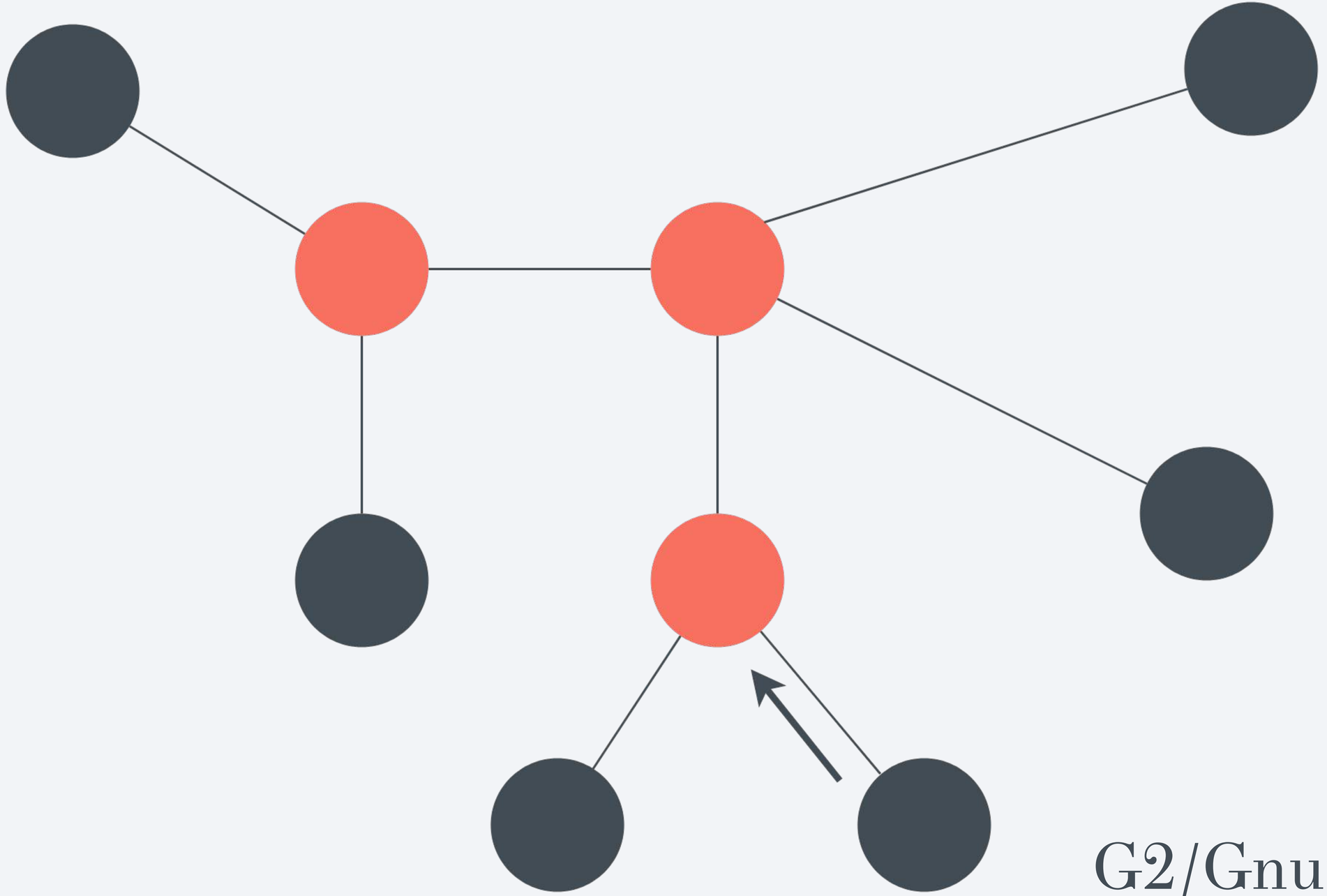
(gnutella2)



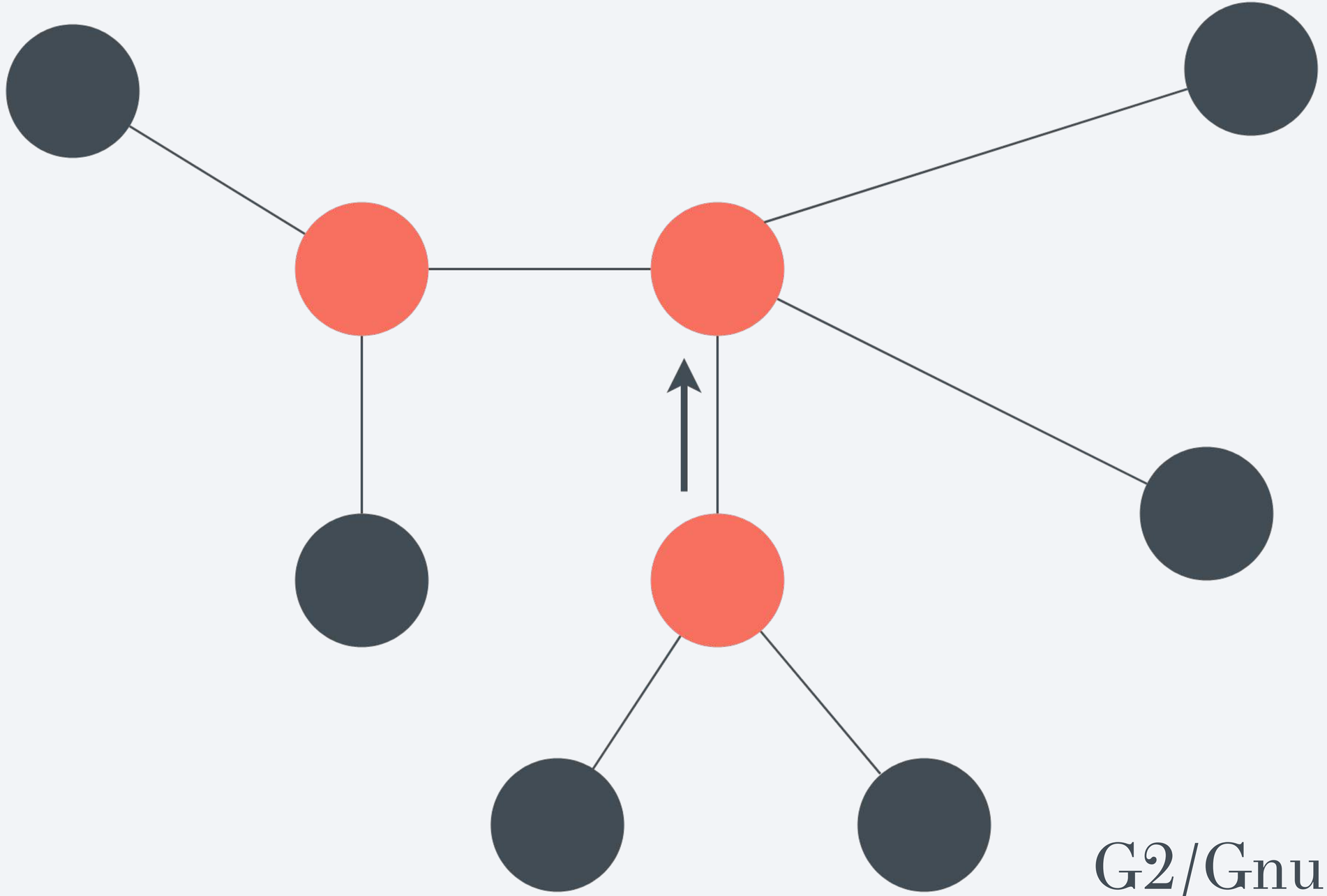
Gnutella



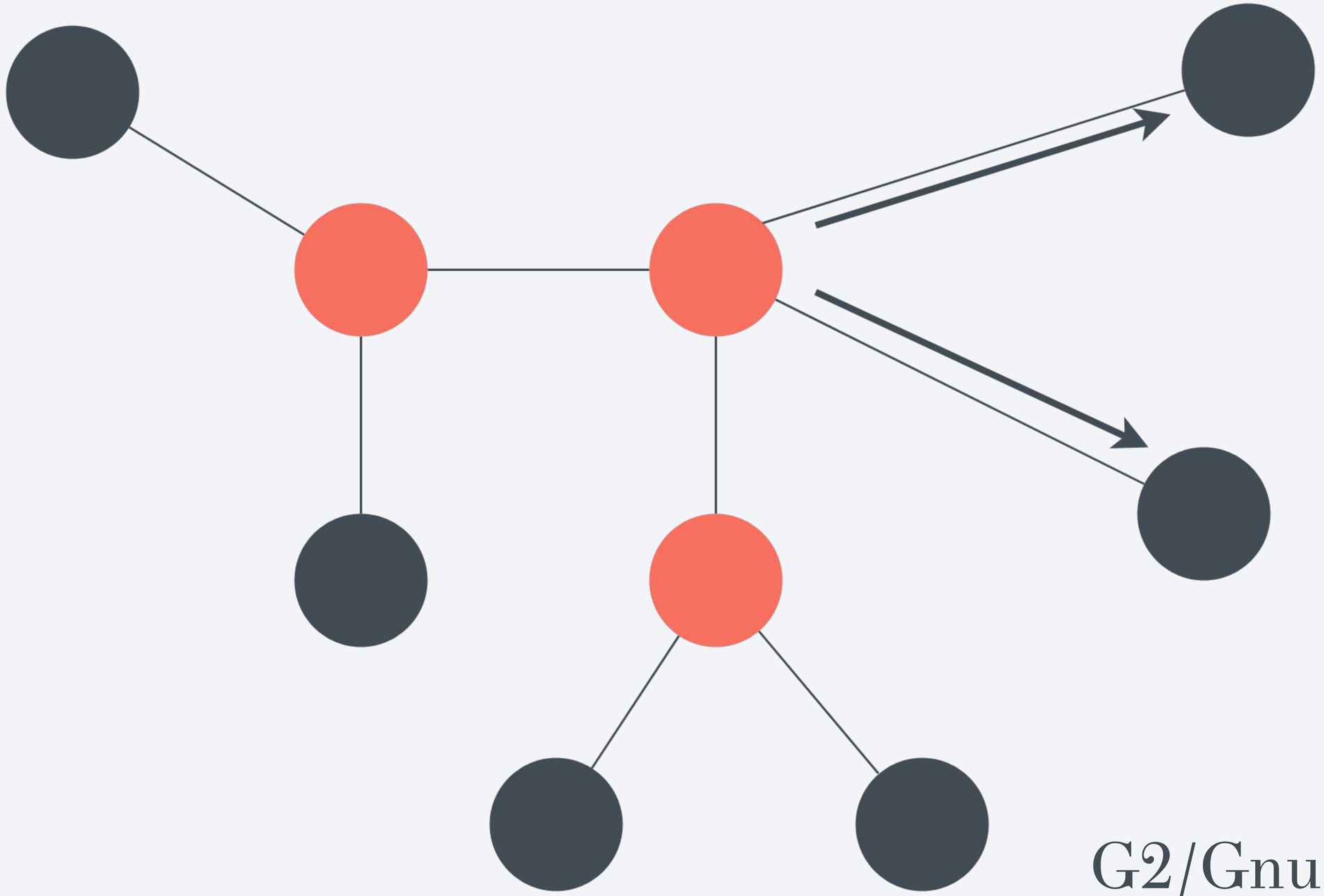
G2/Gnutella2



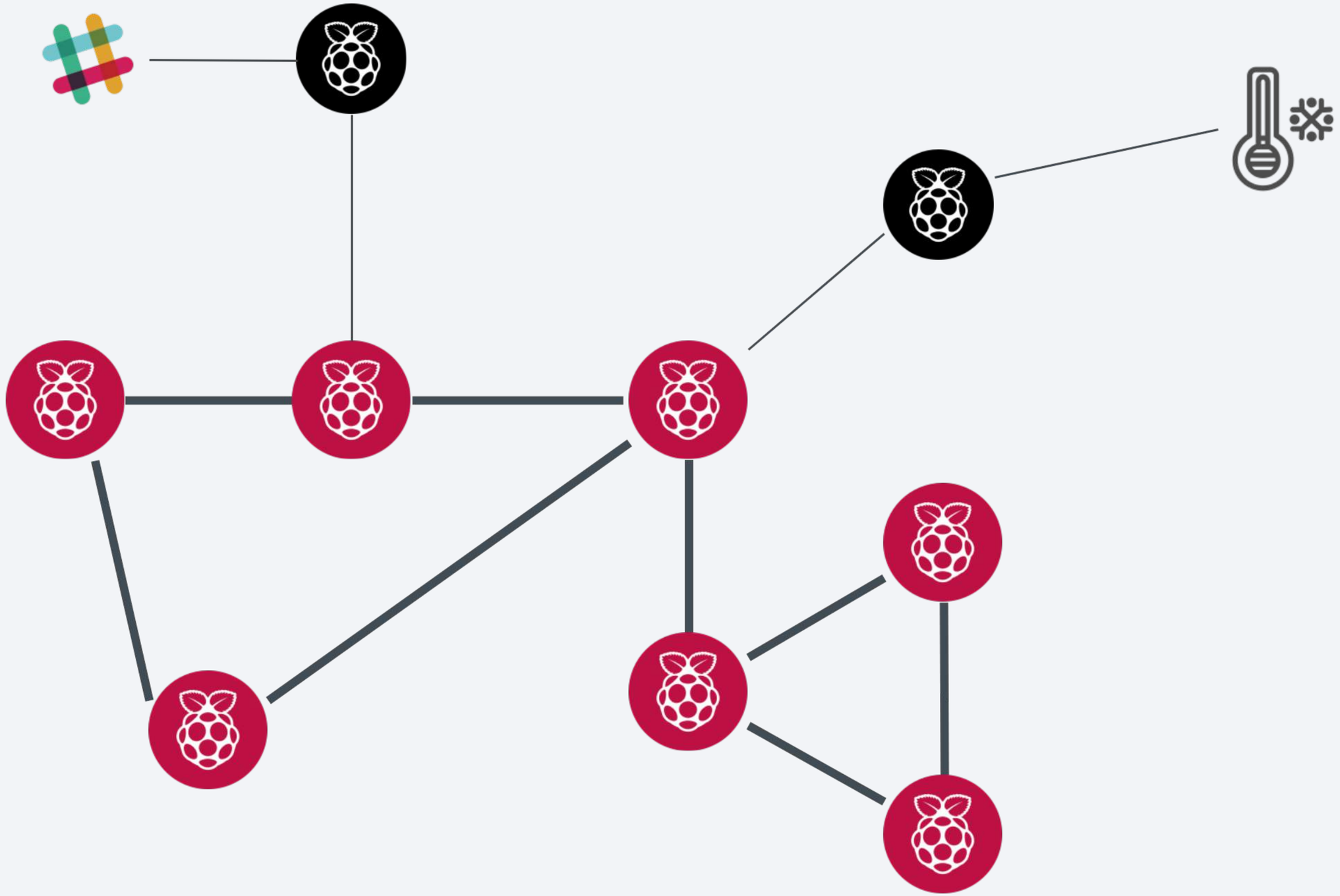
G2/Gnutella2

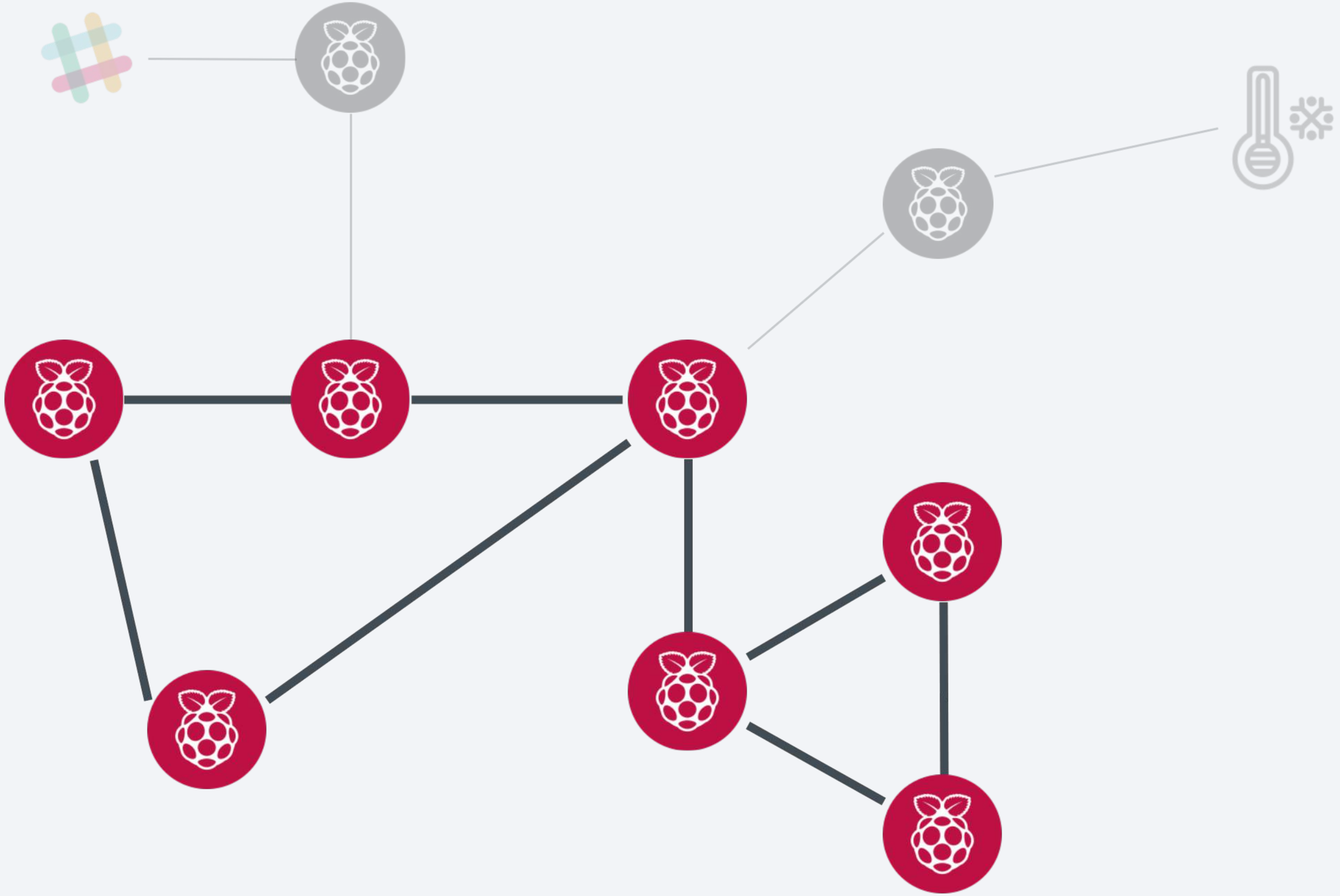


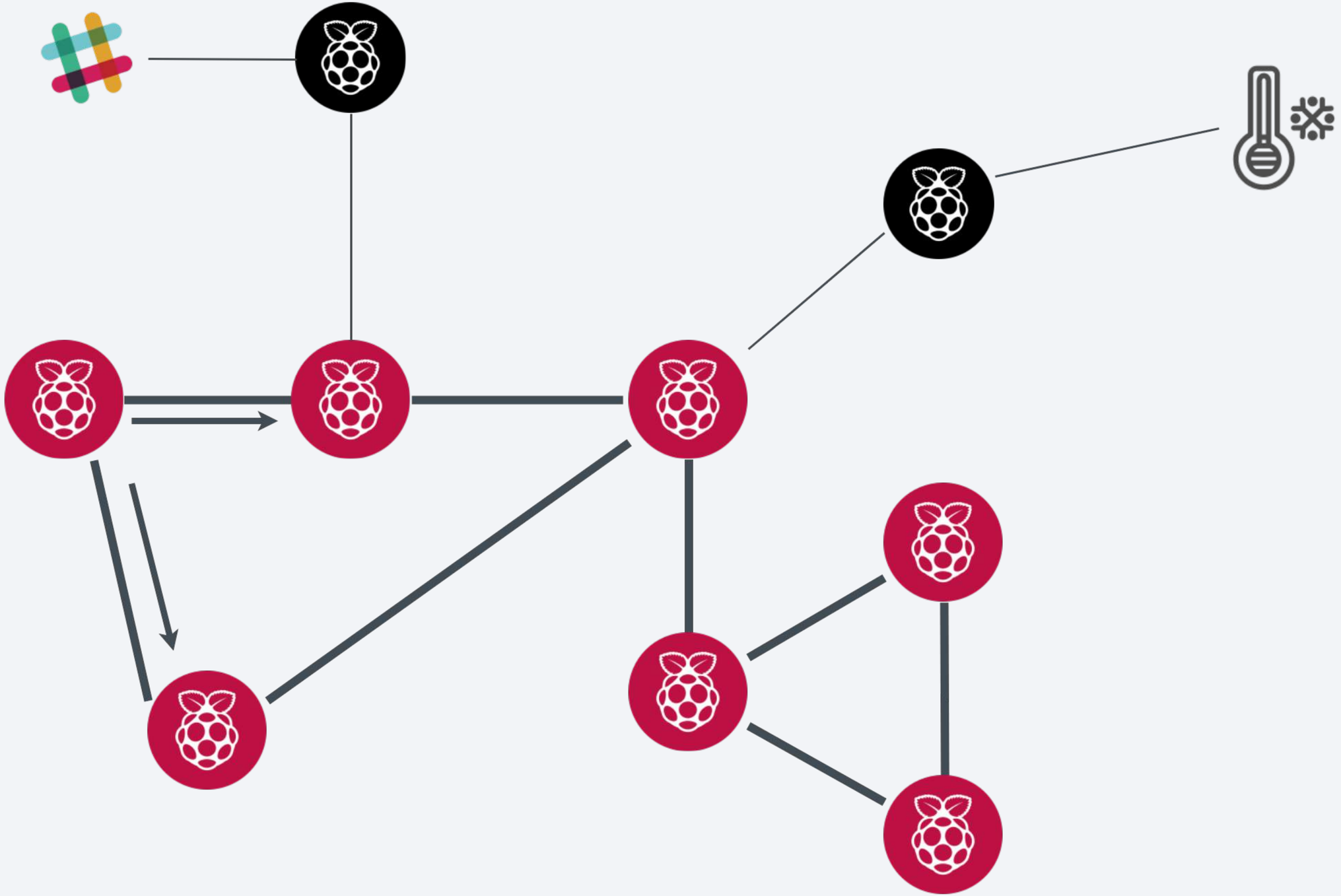
G2/Gnutella2

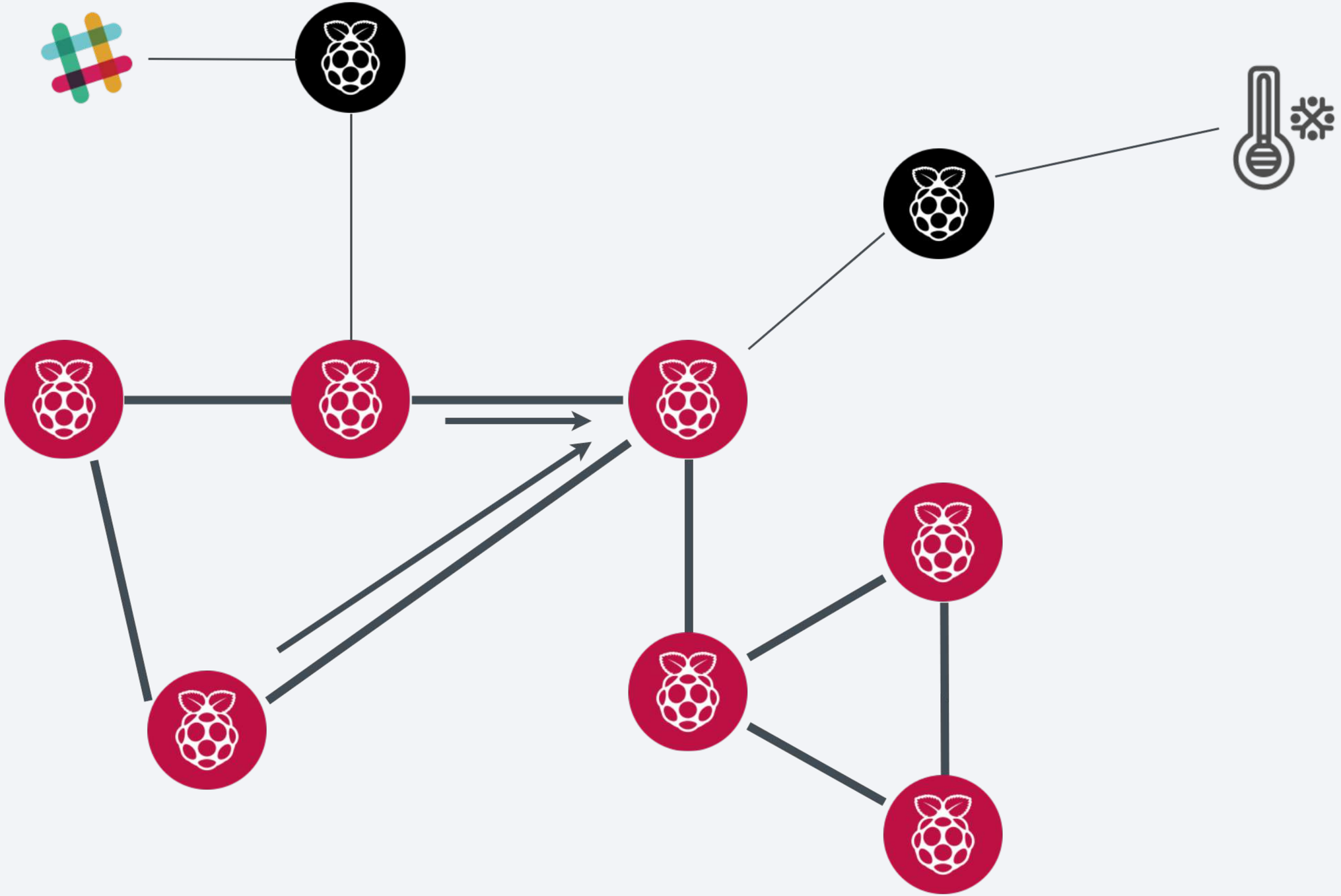


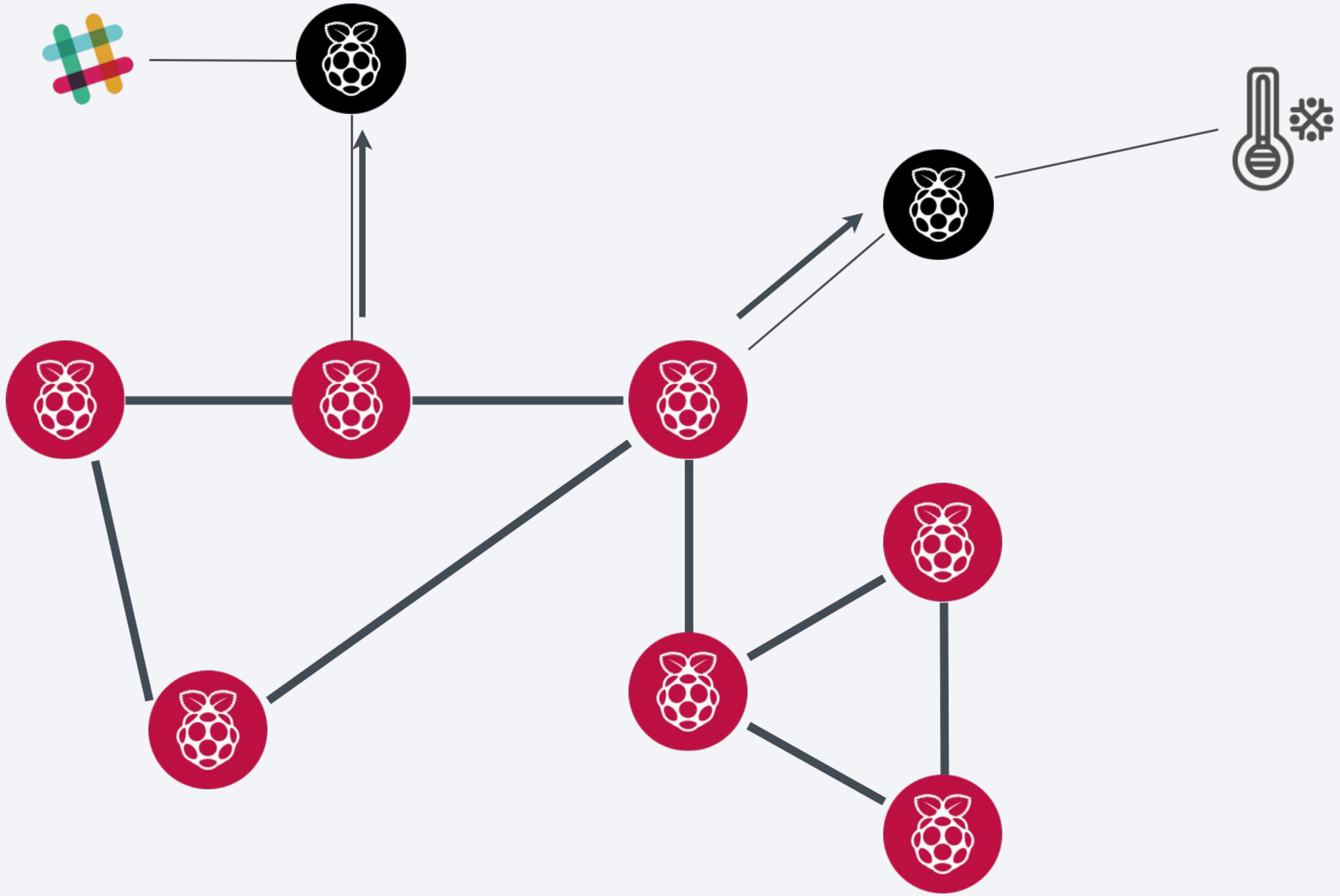
G2/Gnutella2



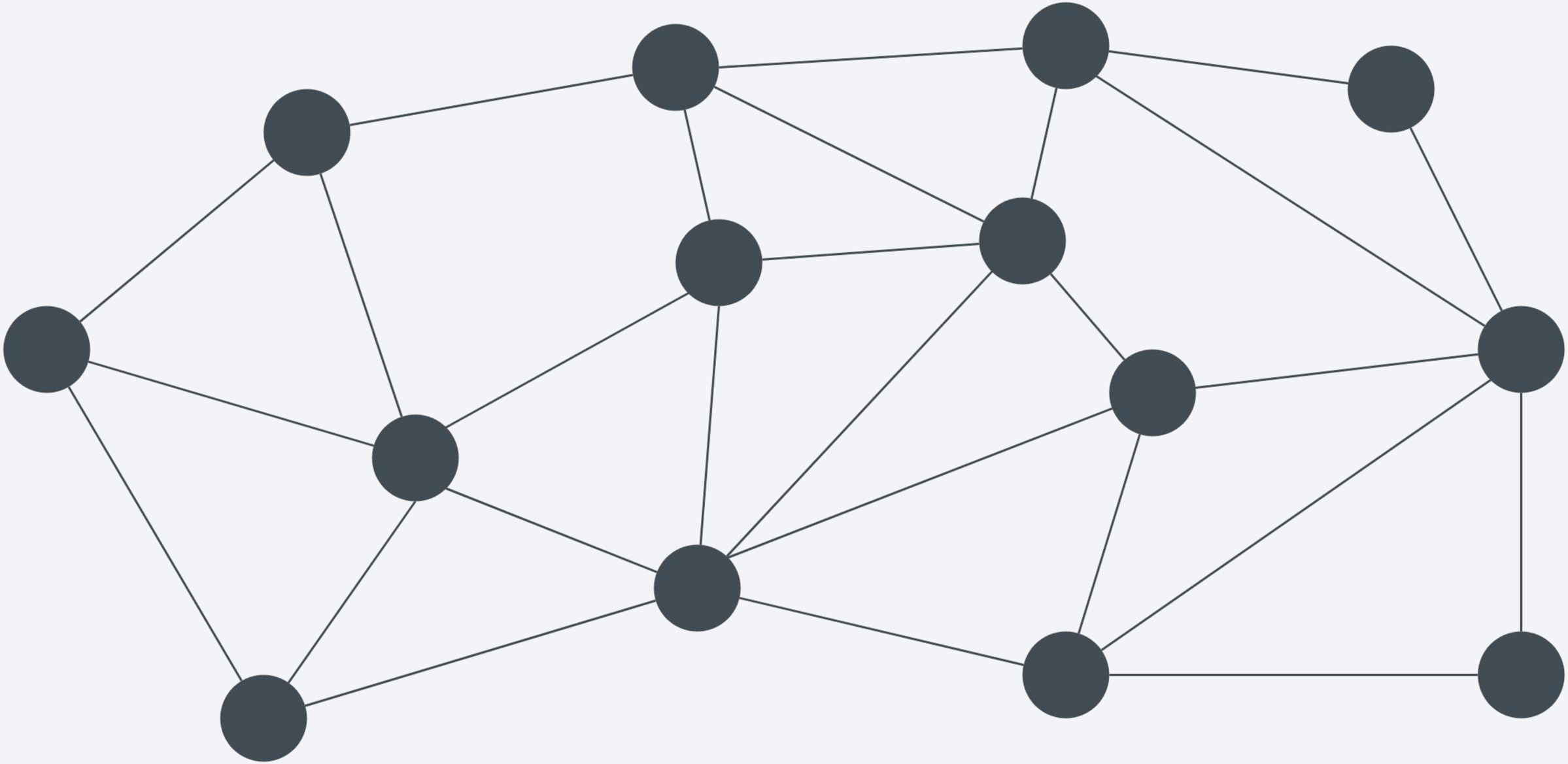


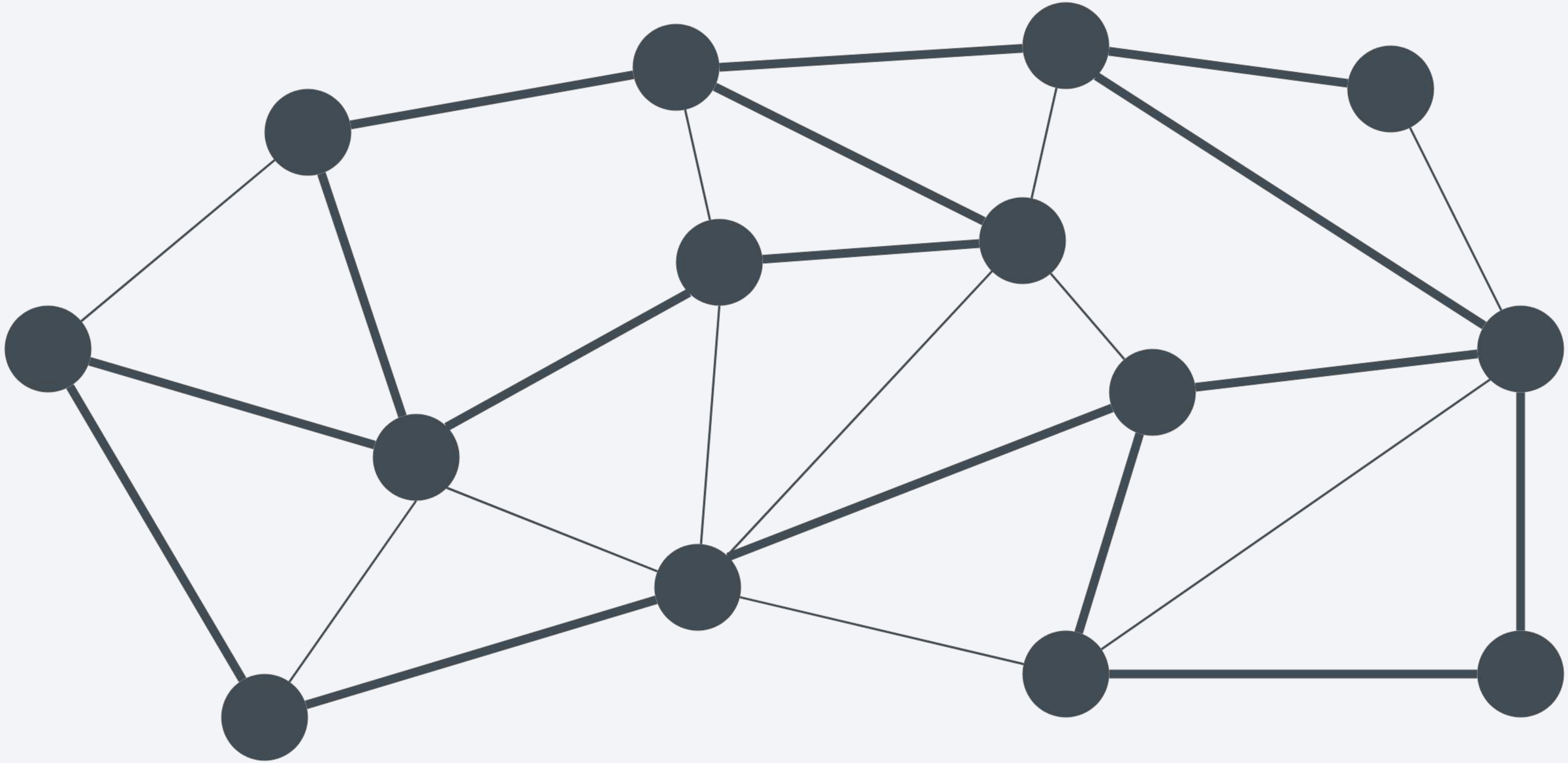


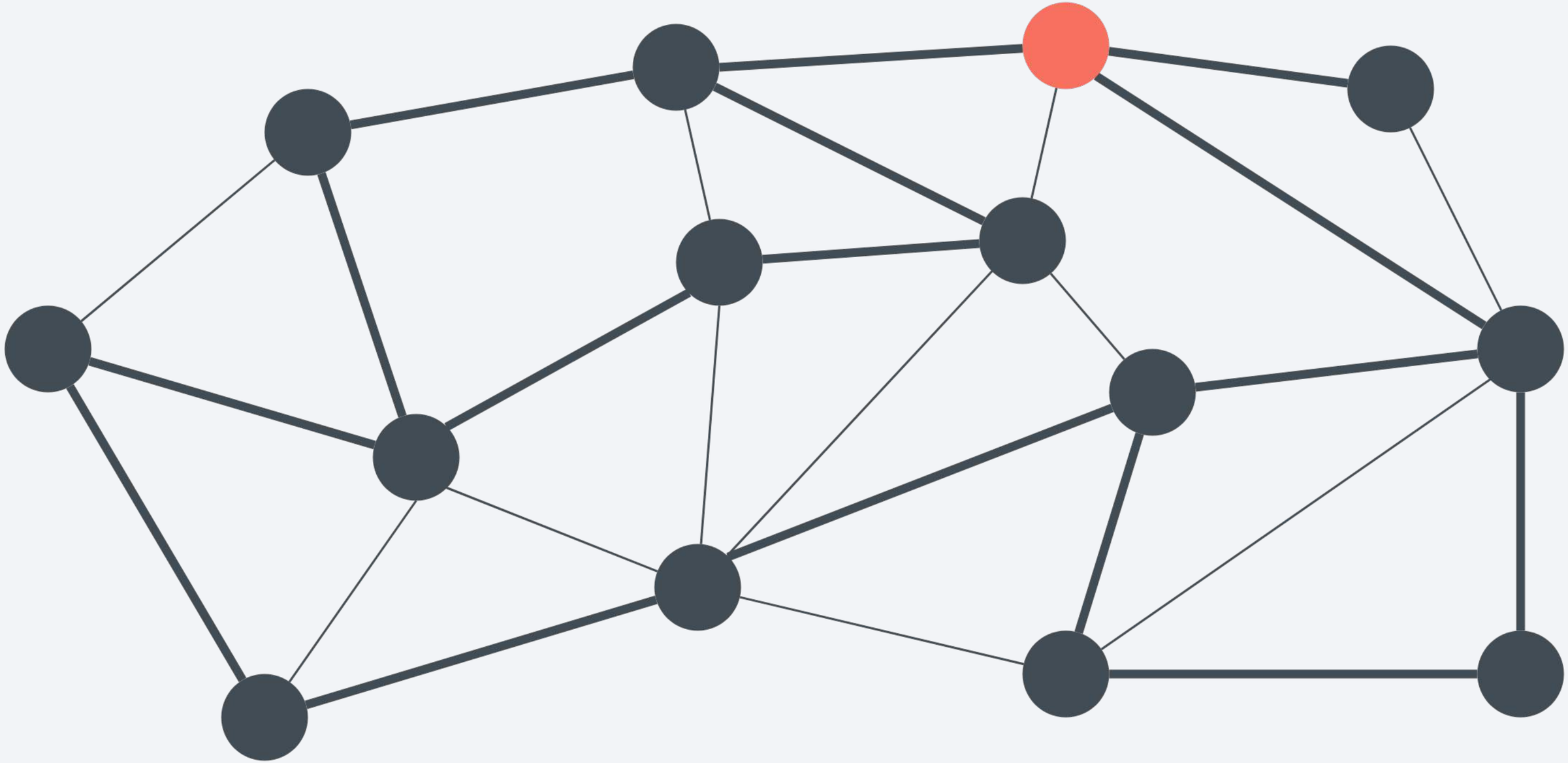


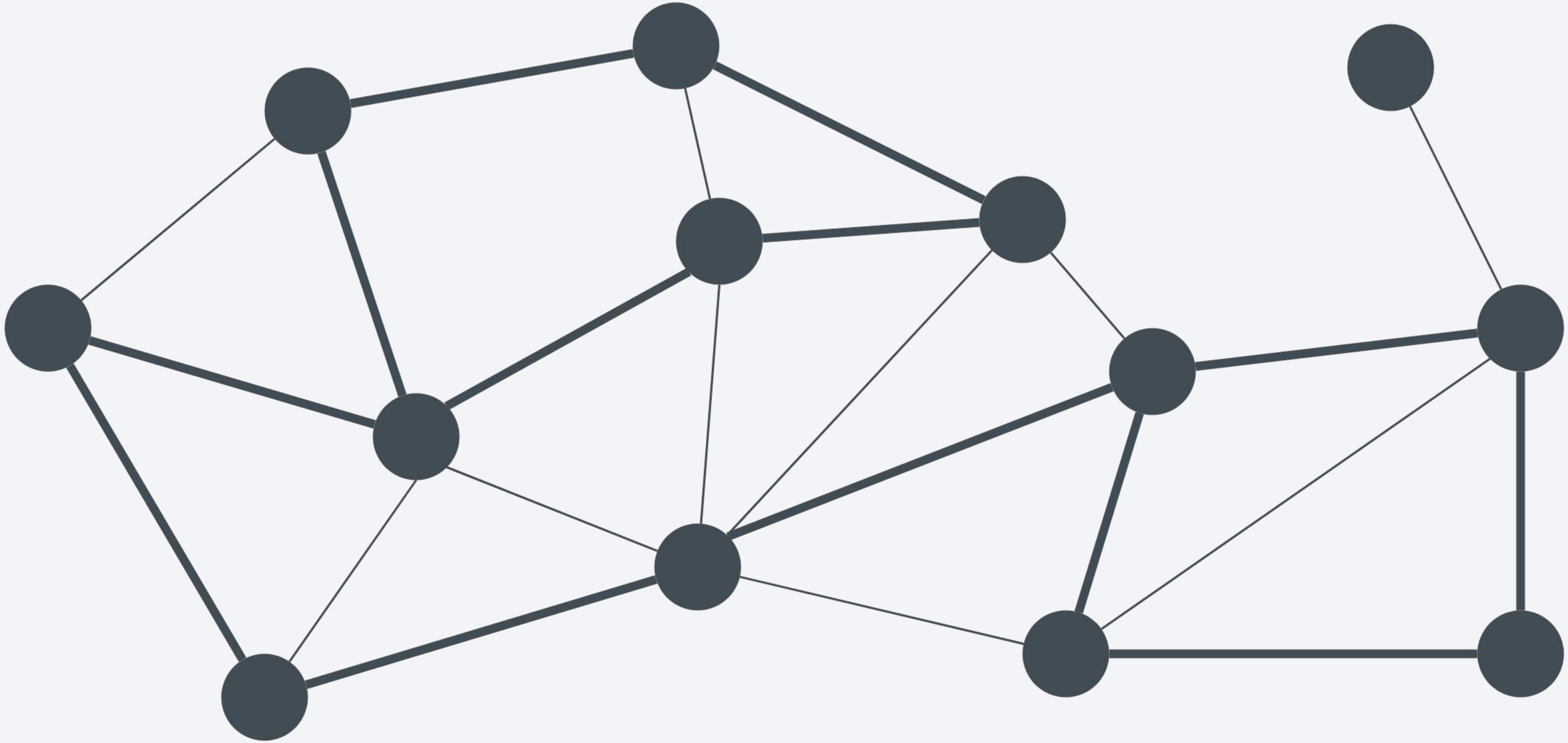


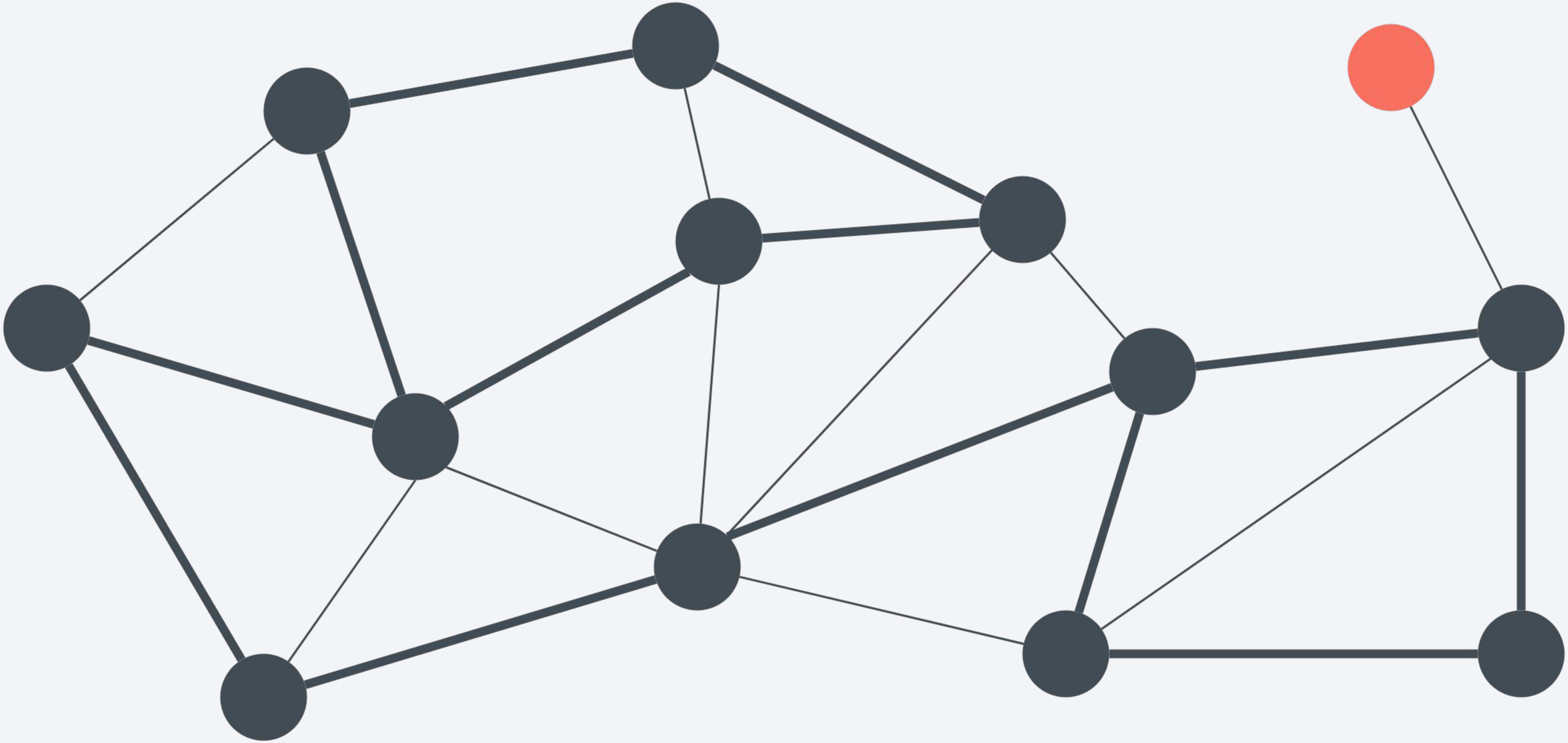
HyParView

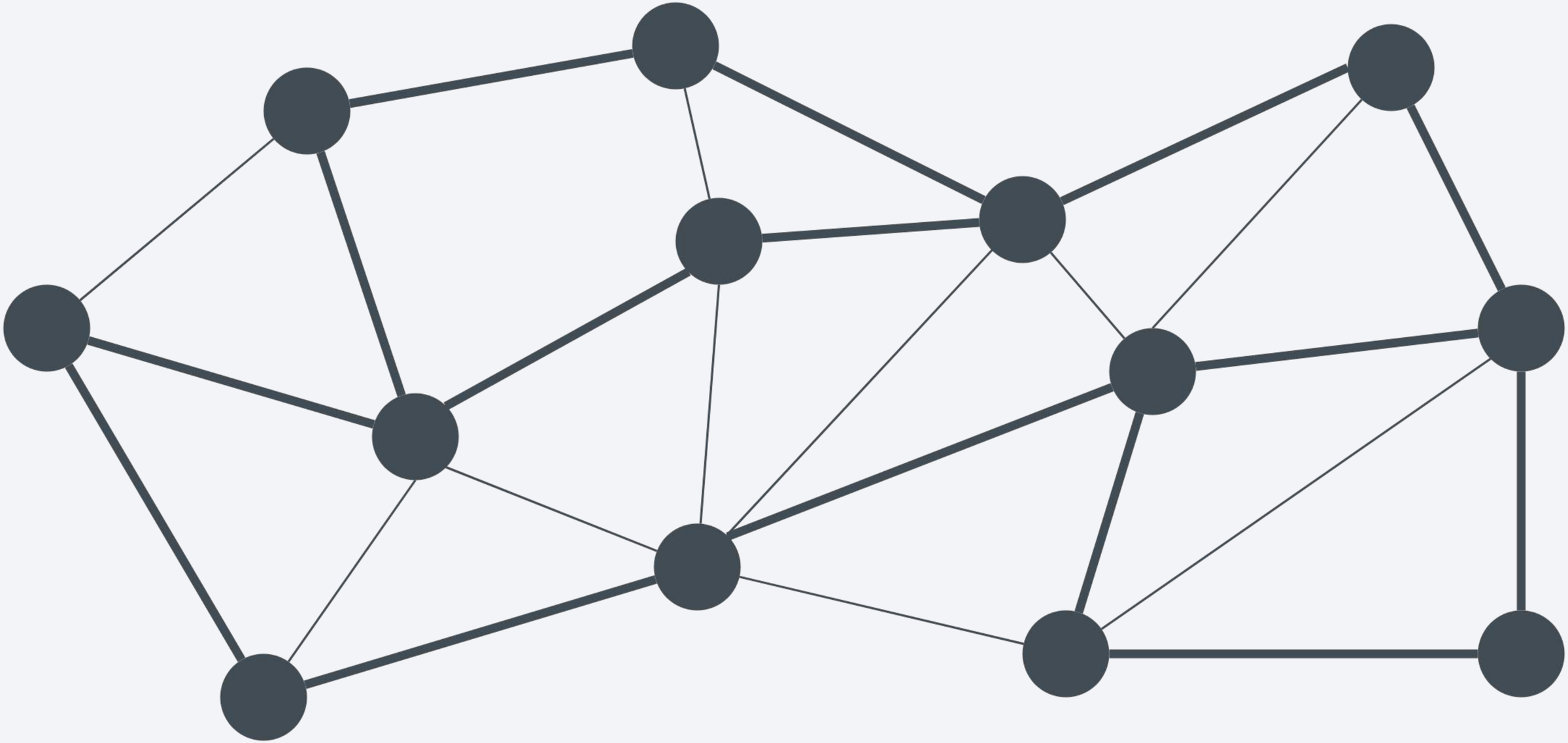












“Aha! It works on my computer!”

“Aha! It works on my computer!”

“Great but we need something to show”

“

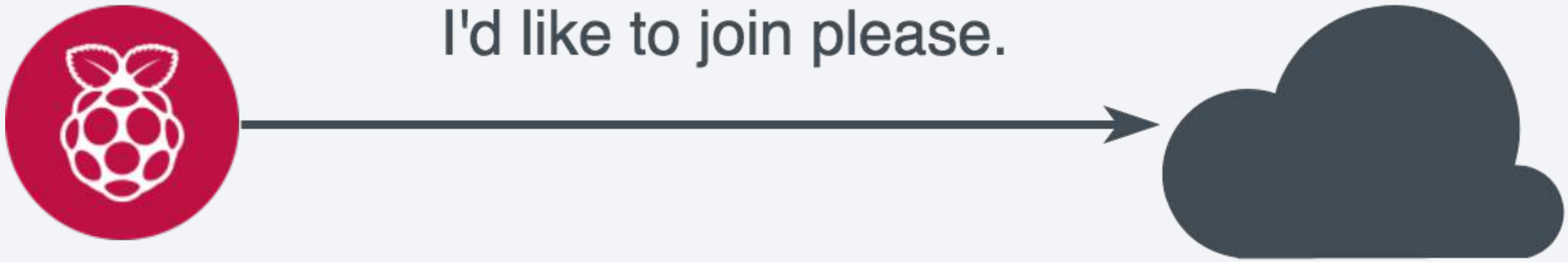
Great but we need something to show”

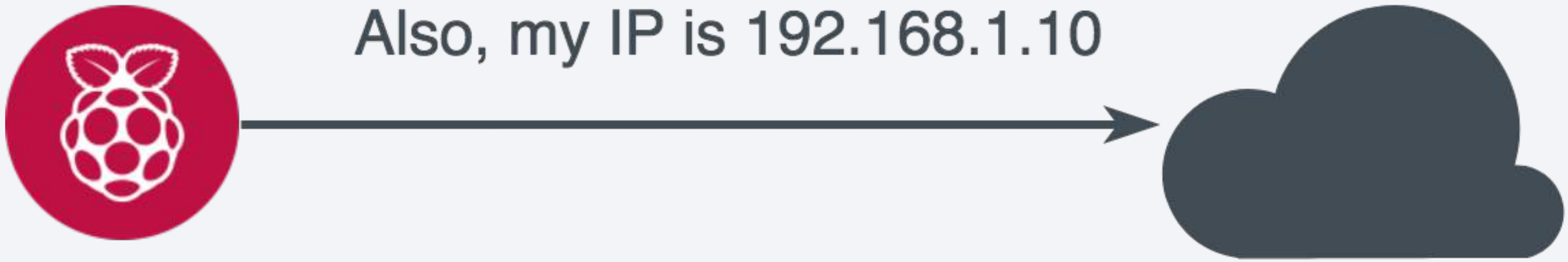
(aka Raspberry Pi time)

“Guys... Is this a bomb? Are we going to die?”

— @naps62

“Hey, I can borrow™ someone else’s code”


I'd like to join please.


Also, my IP is 192.168.1.10

you shall not pass!

Stick everything on Raspberry Pi's

Things running on one Raspberry Pi

Things running on one Raspberry Pi

✓BEAM

Things running on one Raspberry Pi

✓BEAM

✓thebox (sensors)

Things running on one Raspberry Pi

✓BEAM

✓thebox (sensors)

✓Phoenix app

Things running on one Raspberry Pi

✓BEAM (x2)

✓thebox (sensors)

✓Phoenix app

Things running on one Raspberry Pi

- ✓BEAM (x2)
- ✓thebox (sensors)
- ✓Phoenix app
- ✓Postgres

Things running on one Raspberry Pi

✓BEAM (x2)

✓thebox (sensors)

✓Phoenix app

✓Postgres

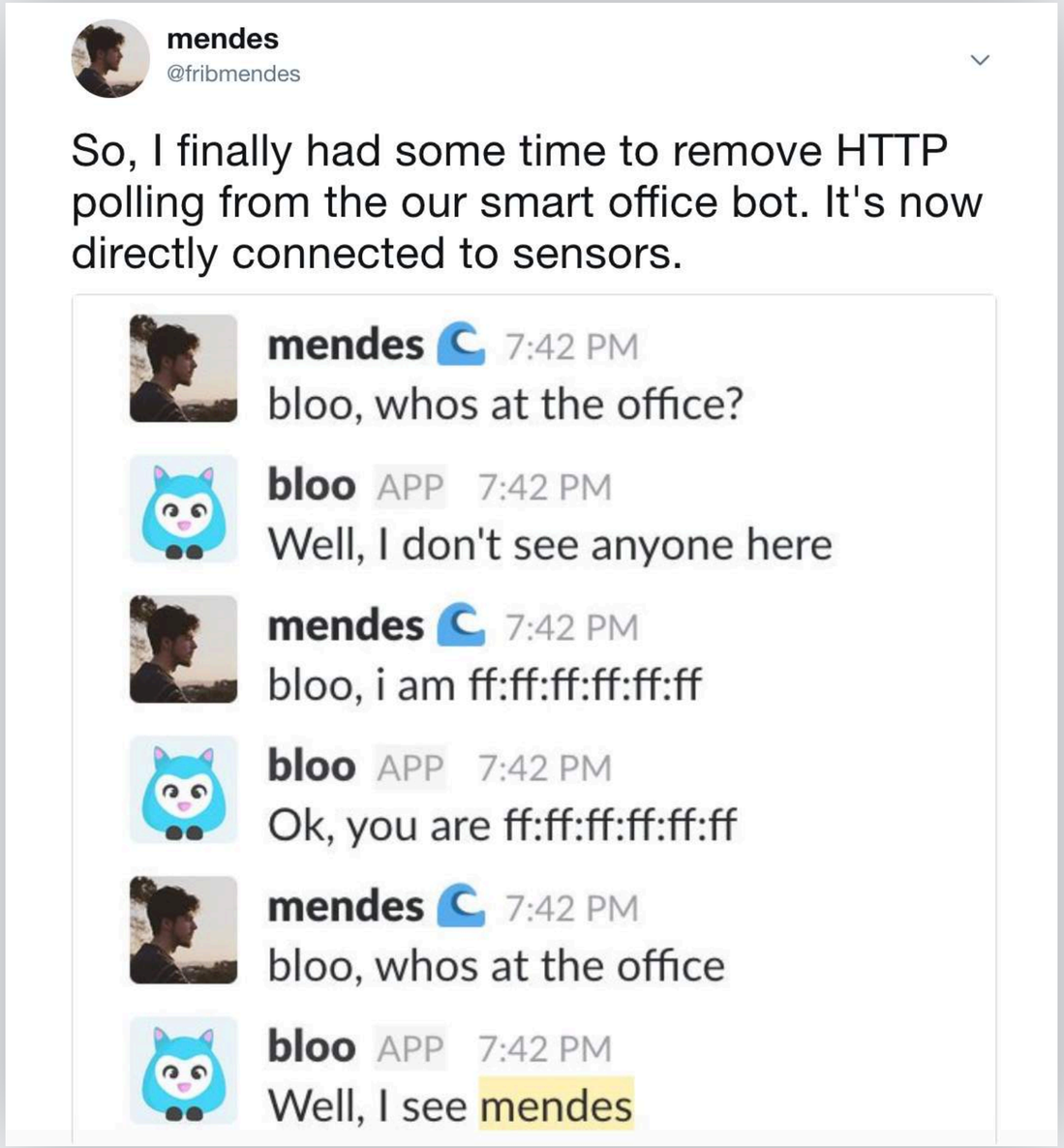
✓Cassandra

Things running on one Raspberry Pi

- ✓BEAM (x2)
- ✓thebox (sensors)
- ✓Phoenix app
- ✓Postgres
- ✓Cassandra

it works!


ONEGIF.COM


mendes
@fribmendes
So, I finally had some time to remove HTTP polling from the our smart office bot. It's now directly connected to sensors.
mendes 7:42 PM
bloo, whos at the office?
bloo APP 7:42 PM
Well, I don't see anyone here
mendes 7:42 PM
bloo, i am ff:ff:ff:ff:ff:ff
bloo APP 7:42 PM
Ok, you are ff:ff:ff:ff:ff:ff
mendes 7:42 PM
bloo, whos at the office
bloo APP 7:42 PM
Well, I see mendes

bloo APP 7:17 PM

guys, the open space temperature is at 24.0°C. You might want to turn the AC on.

“Looking good! Everything’s working!”

lol, nope

State of each node:

State of each node:

- Last sensor readings

State of each node:

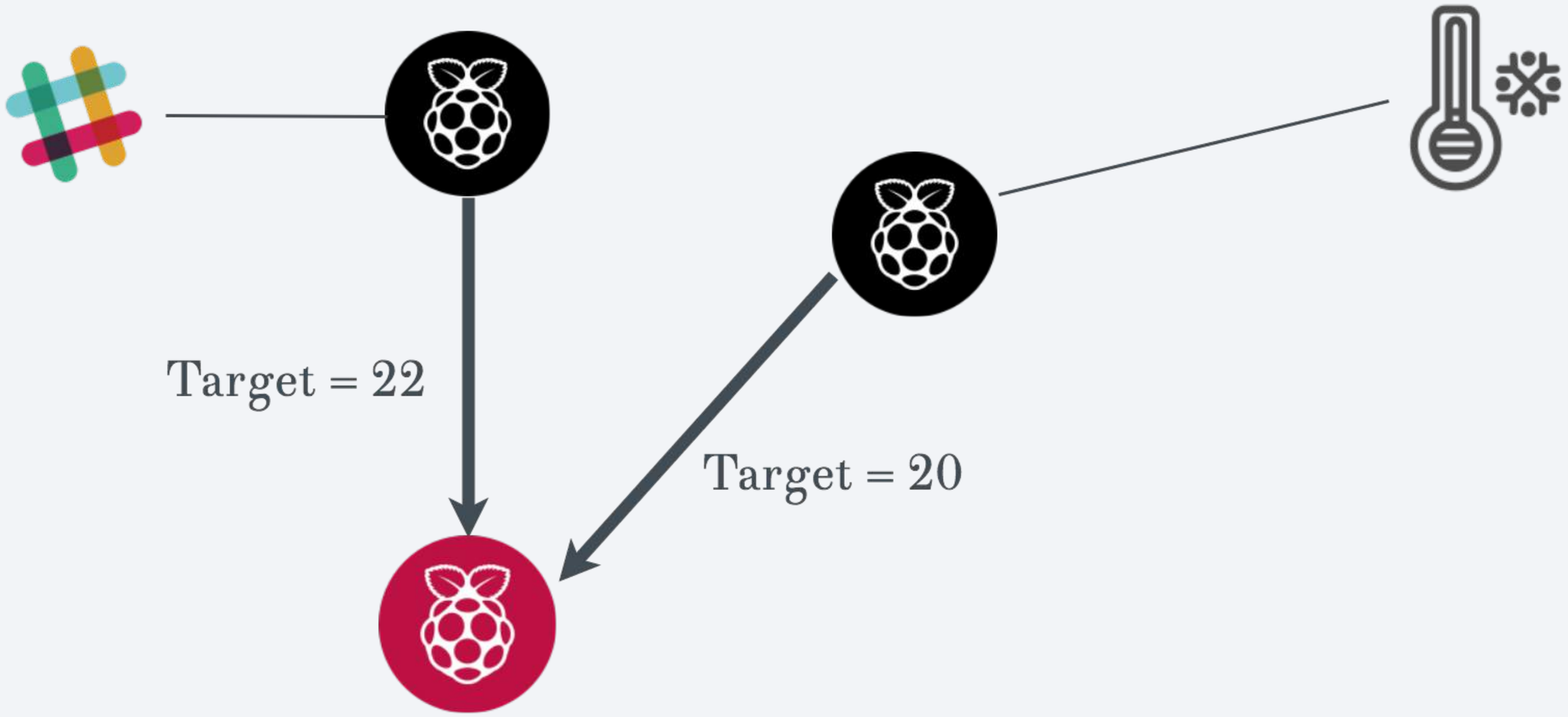
- Last sensor readings
- Network map (MAC-IP)

State of each node:

- Last sensor readings
- Network map (MAC-IP)
- Target values

State of each node:

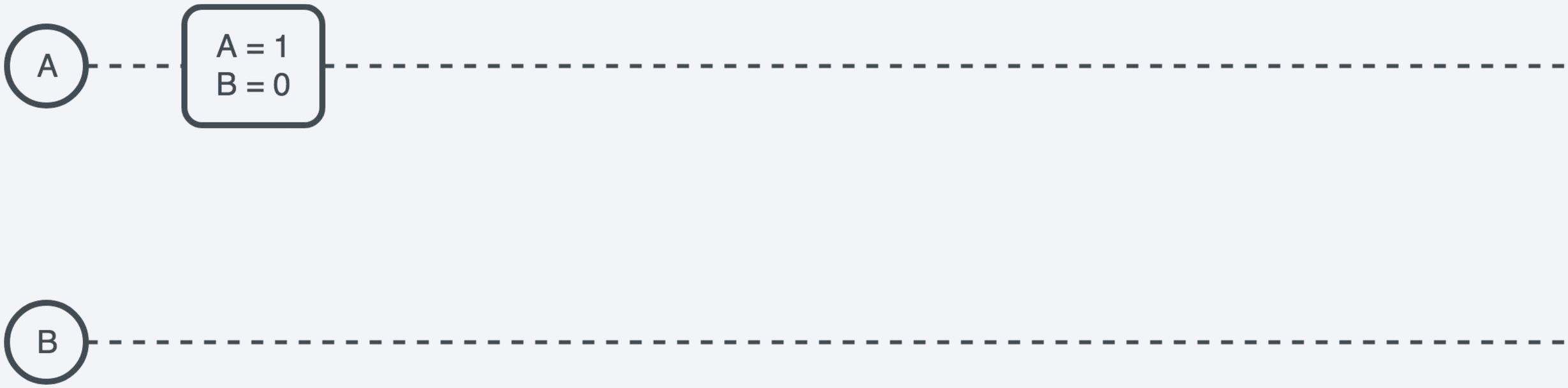
- Last sensor readings
- Network map (MAC-IP)
- Target values


Target = 22
Target = 20

How do we handle concurrency?

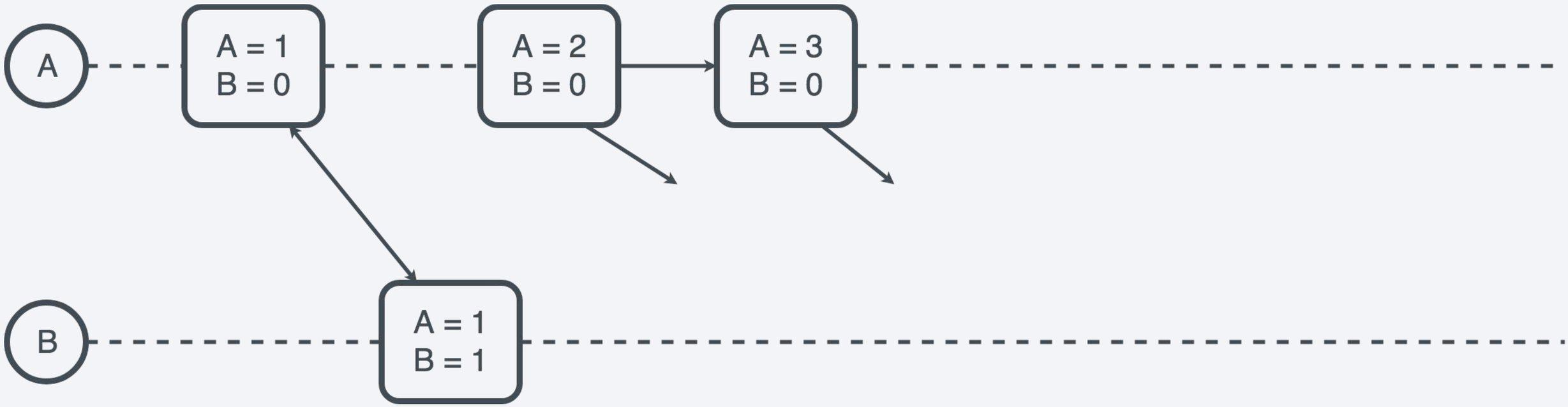
Vector Clocks

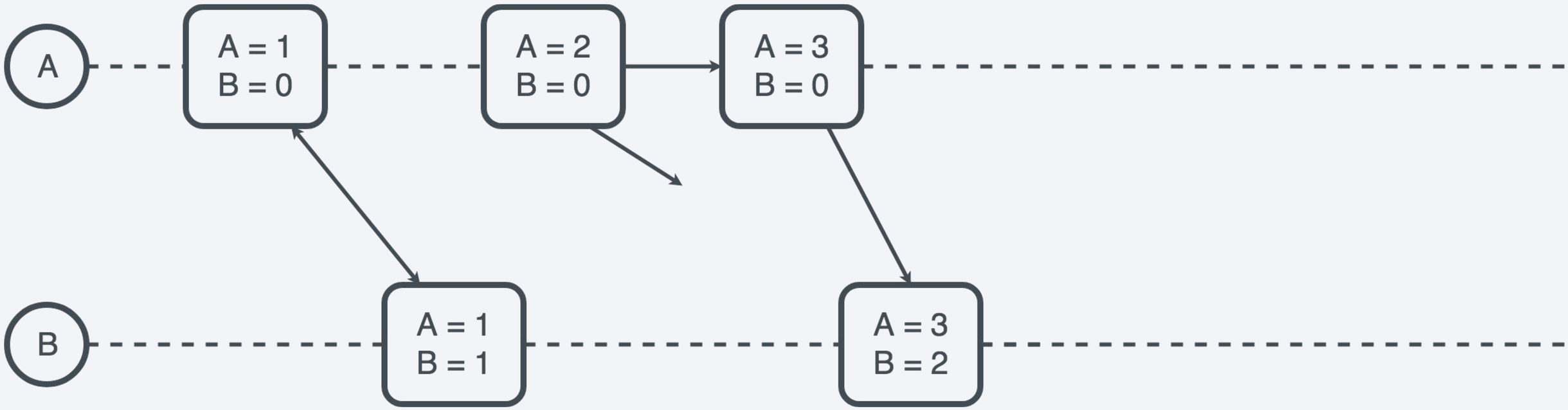

A
B

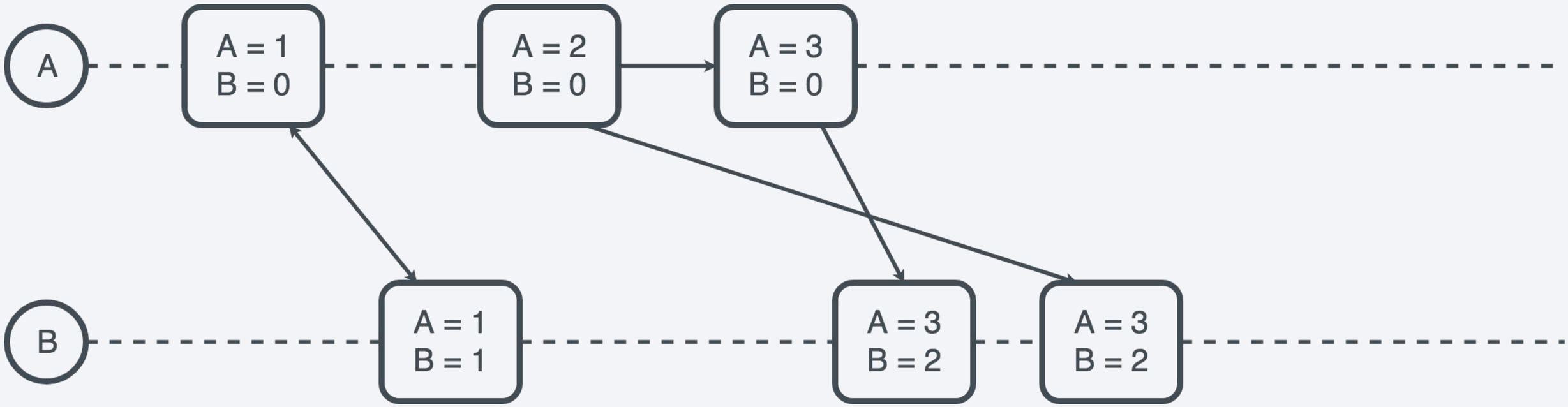

A
A = 1
B = 0
B

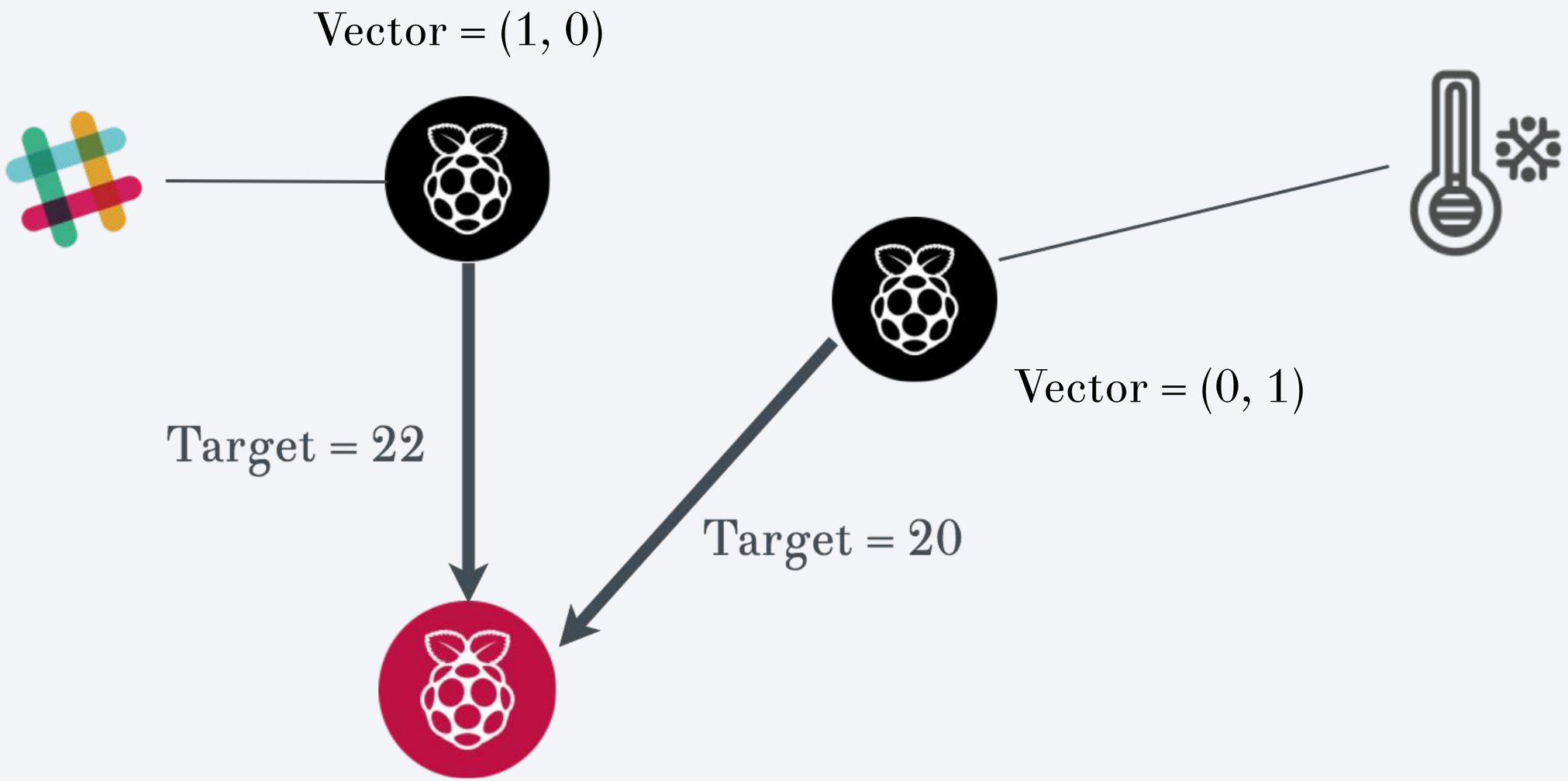

A
A = 1
B = 0
B
A = 1
B = 1


A
A = 1
B = 0
A = 2
B = 0
B
A = 1
B = 1


A
B
A = 1
B = 0
A = 2
B = 0
A = 3
B = 0
A = 1
B = 1


A
B
A = 1
B = 0
A = 2
B = 0
A = 3
B = 0
A = 1
B = 1
A = 3
B = 2


A
B
A = 1
B = 0
A = 2
B = 0
A = 3
B = 0
A = 1
B = 1
A = 3
B = 2
A = 3
B = 2


Vector = (1, 0)
Target = 22
Vector = (0, 1)
Target = 20

CAP Theorem

CAP Theorem

“you’re a programmer.
you can’t have nice things.”

consistency

availability

partitioning

consistency

availability

partitioning

Eventual Consistency

CRDTs

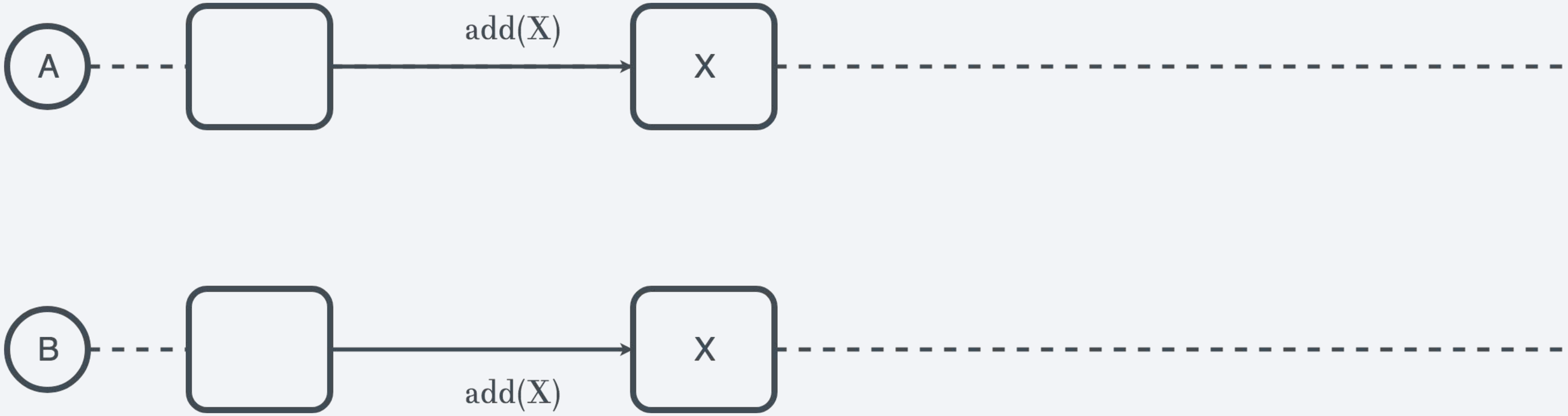
Operation-Based CRDT

Operation-Based CRDT

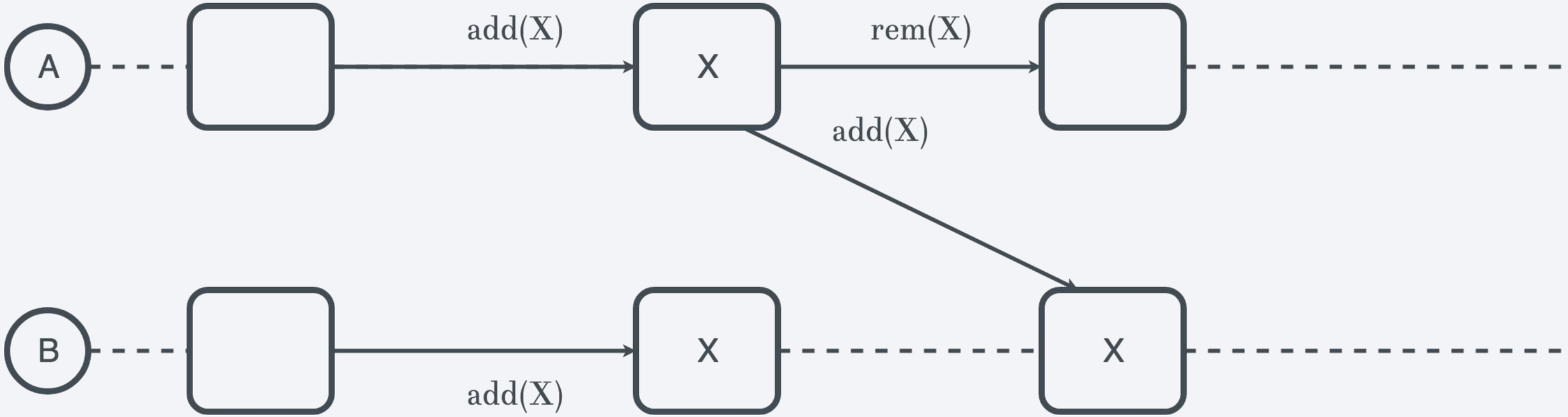
commutative but not idempotent

update exactly once

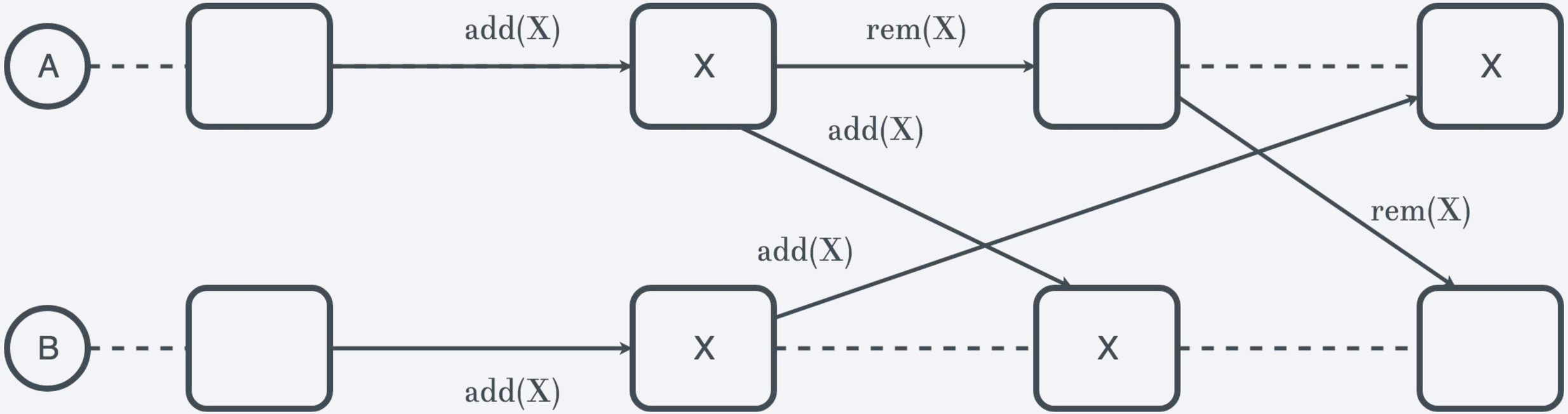
no CRDTs



no CRDTs

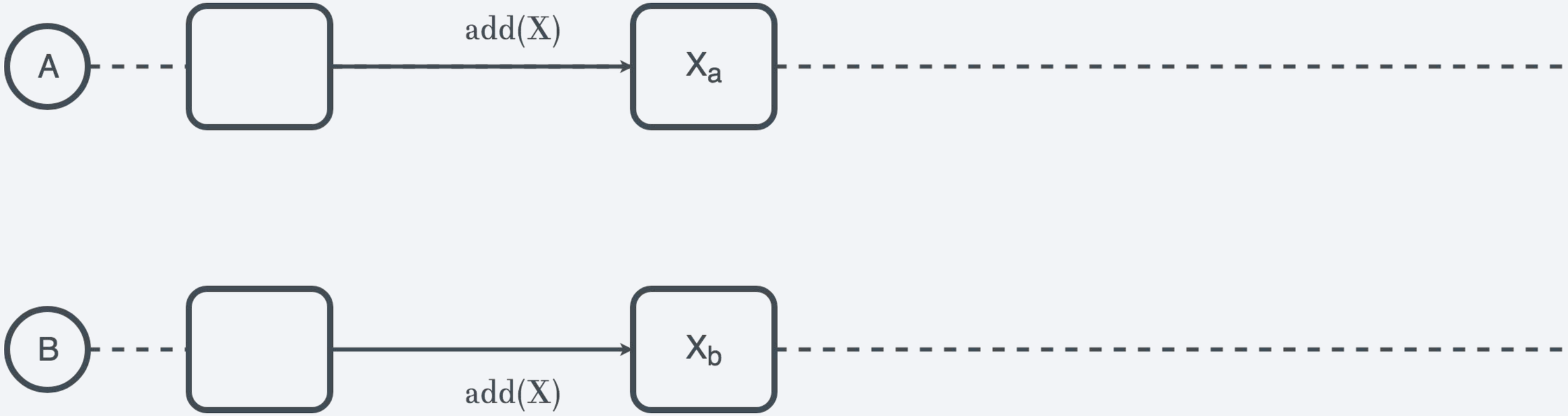


no CRDTs

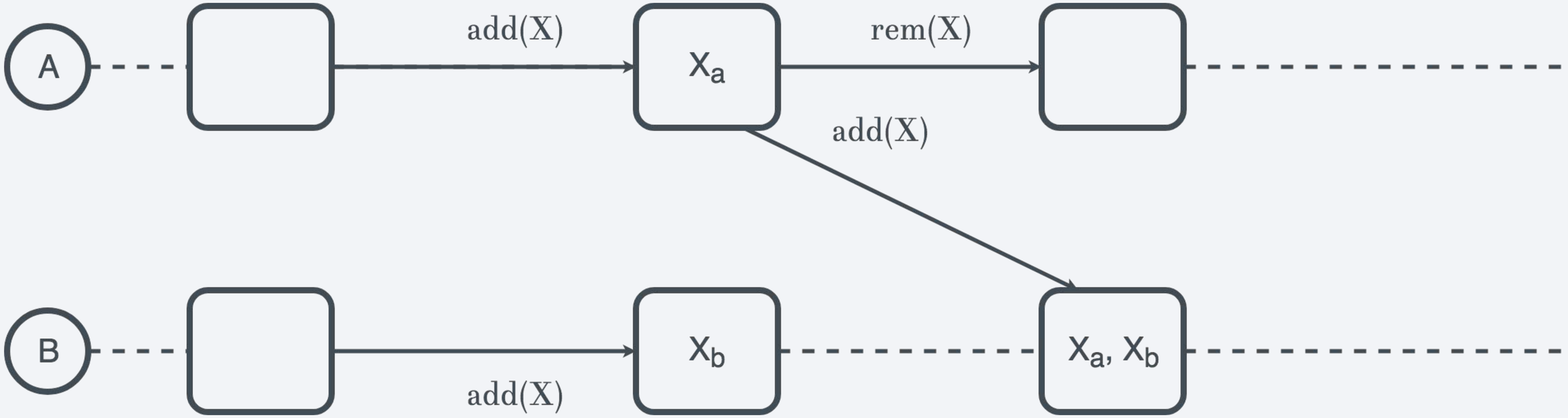


no CRDTs

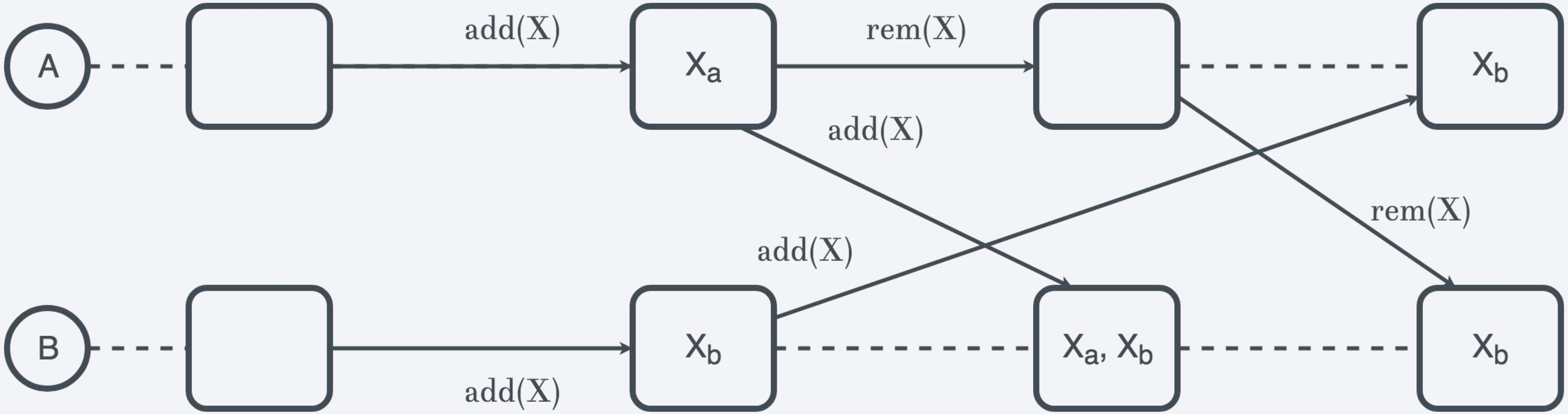
Op-based CRDTs



Op-based CRDTs



Op-based CRDTs



Op-based CRDTs

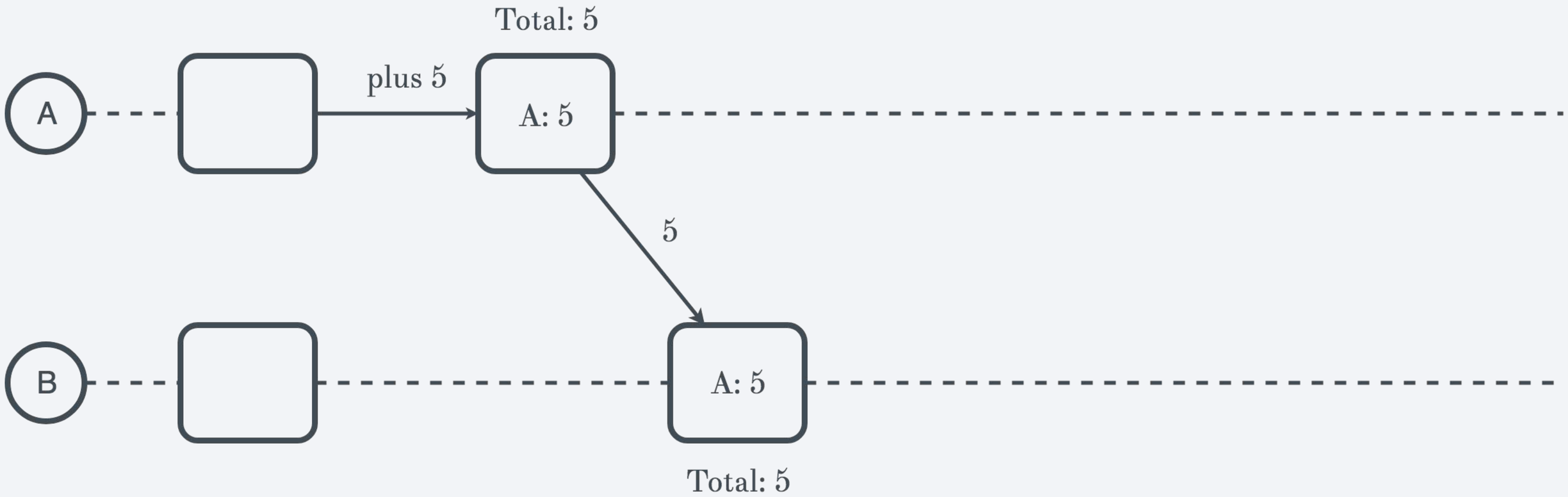
State-Based CRDT

State-Based CRDT

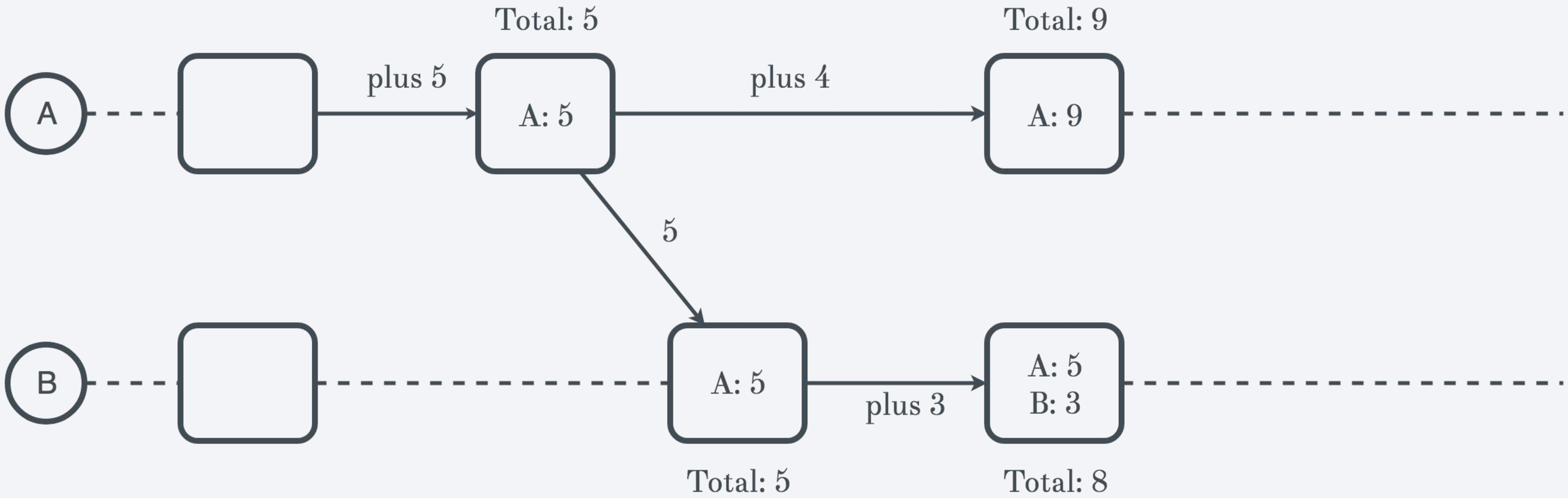
commutative and idempotent

heavier on the network

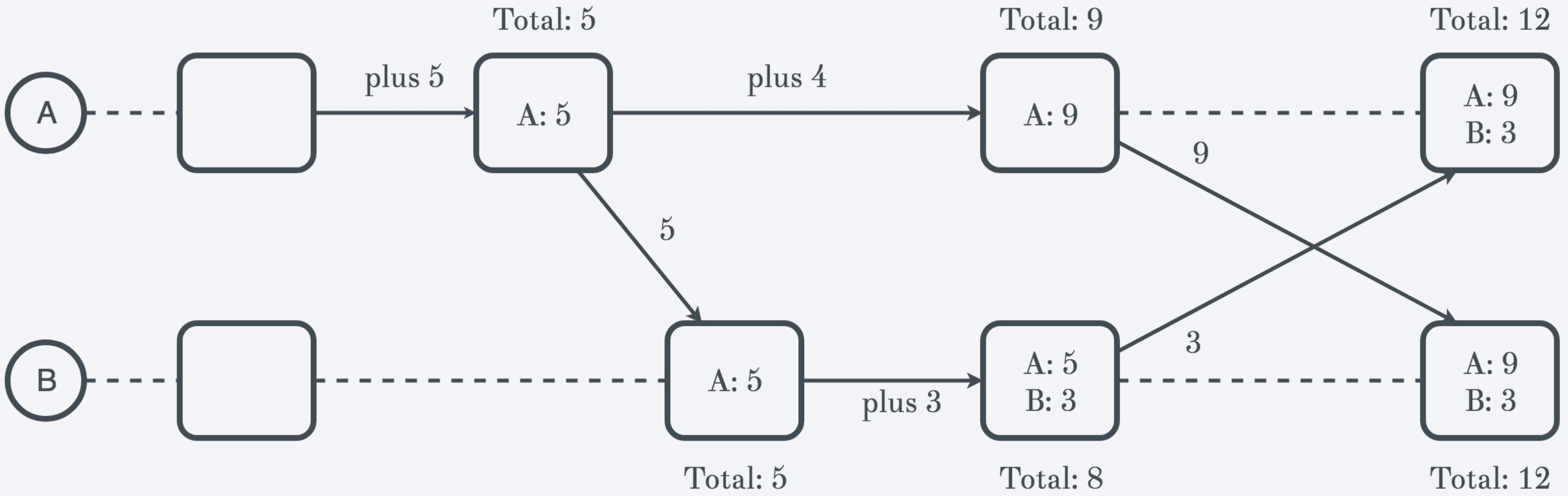
State-based CRDTs



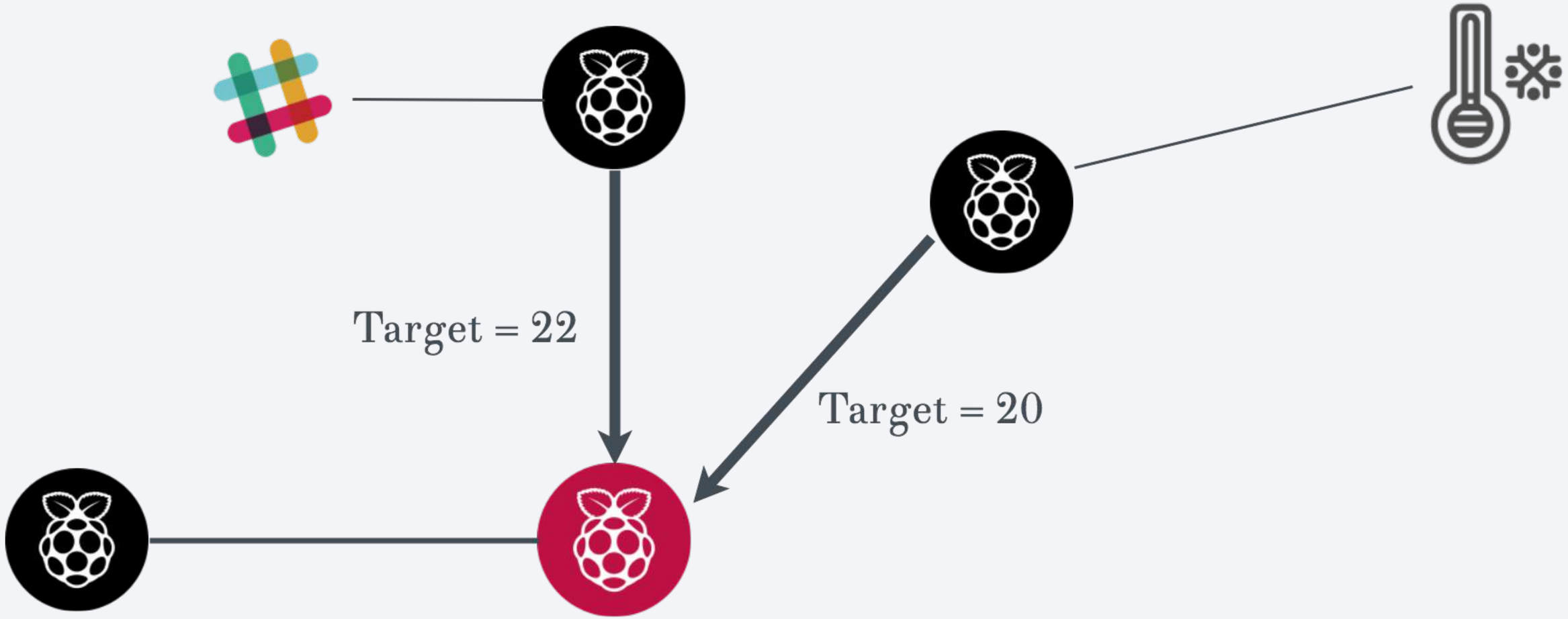
State-based CRDTs

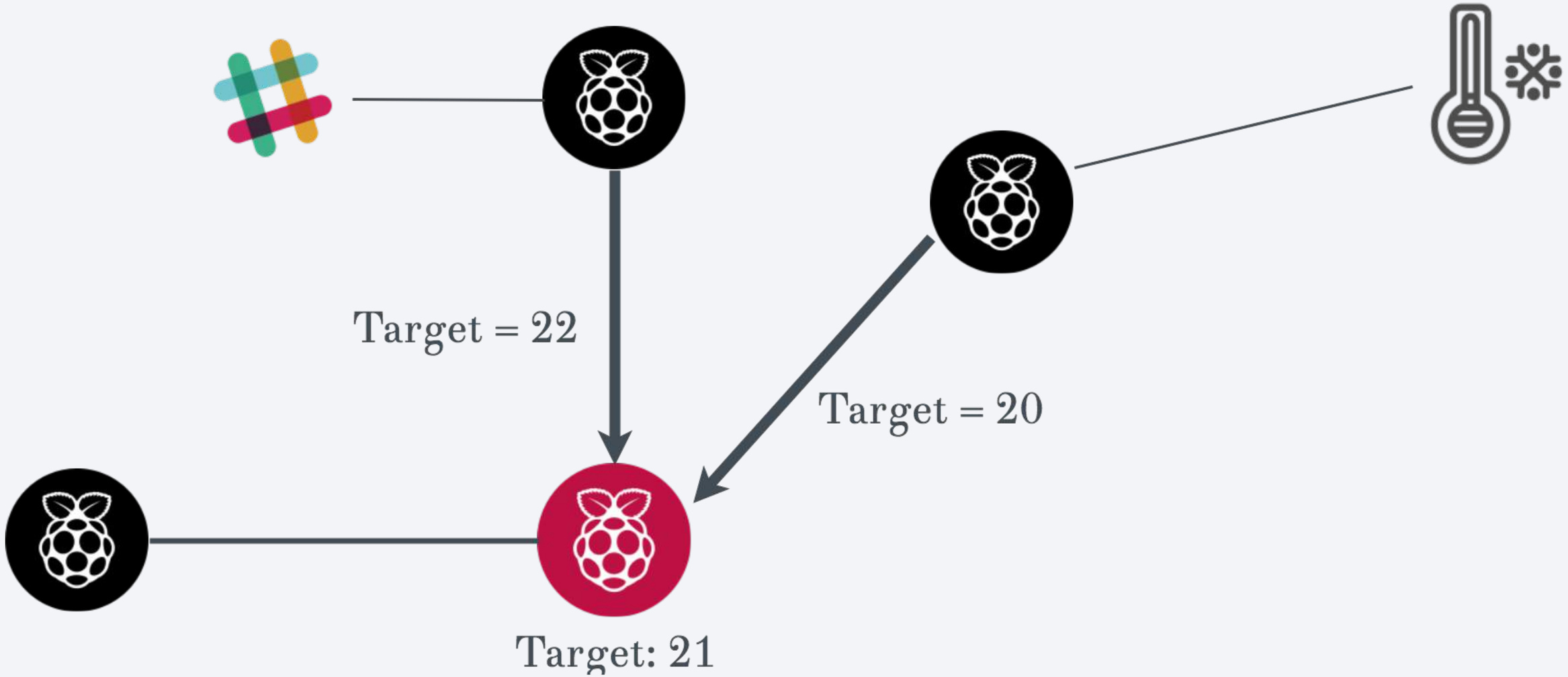


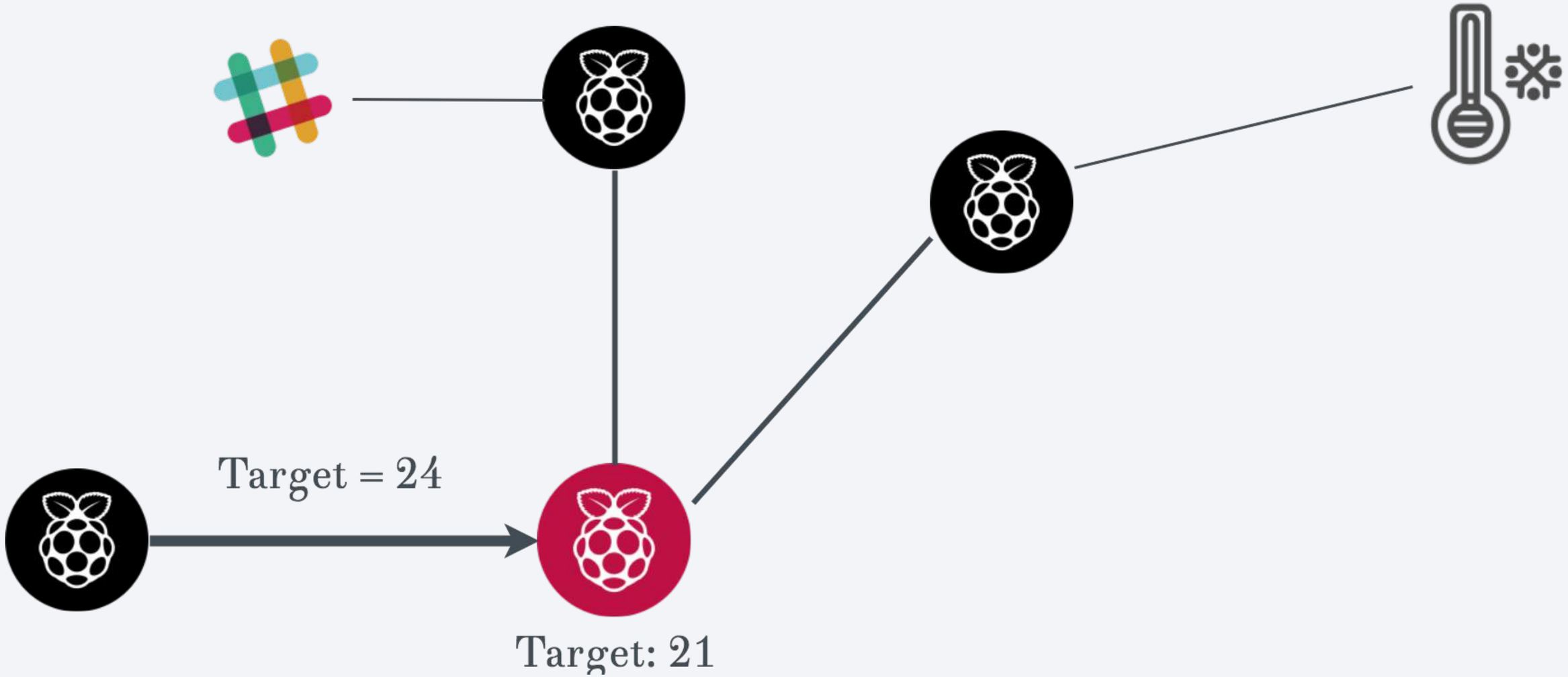
State-based CRDTs

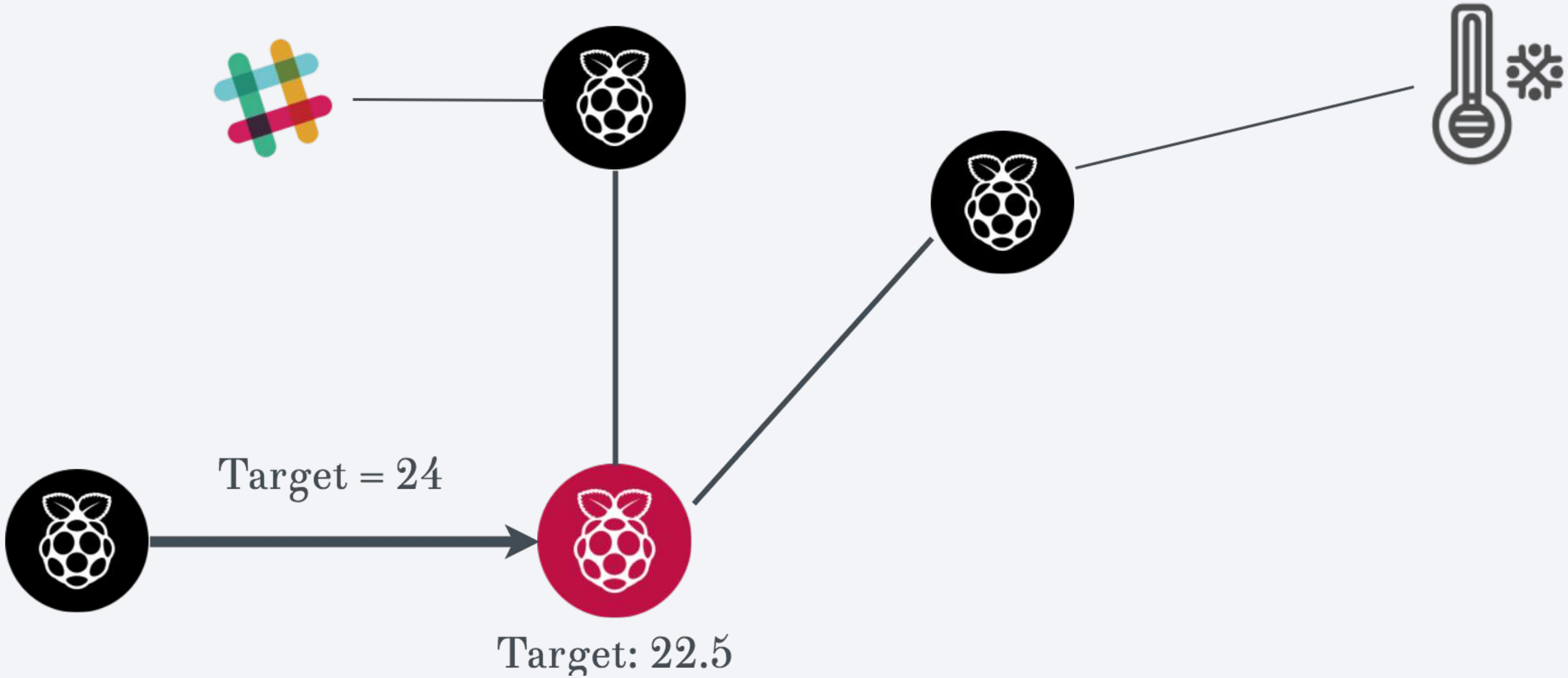


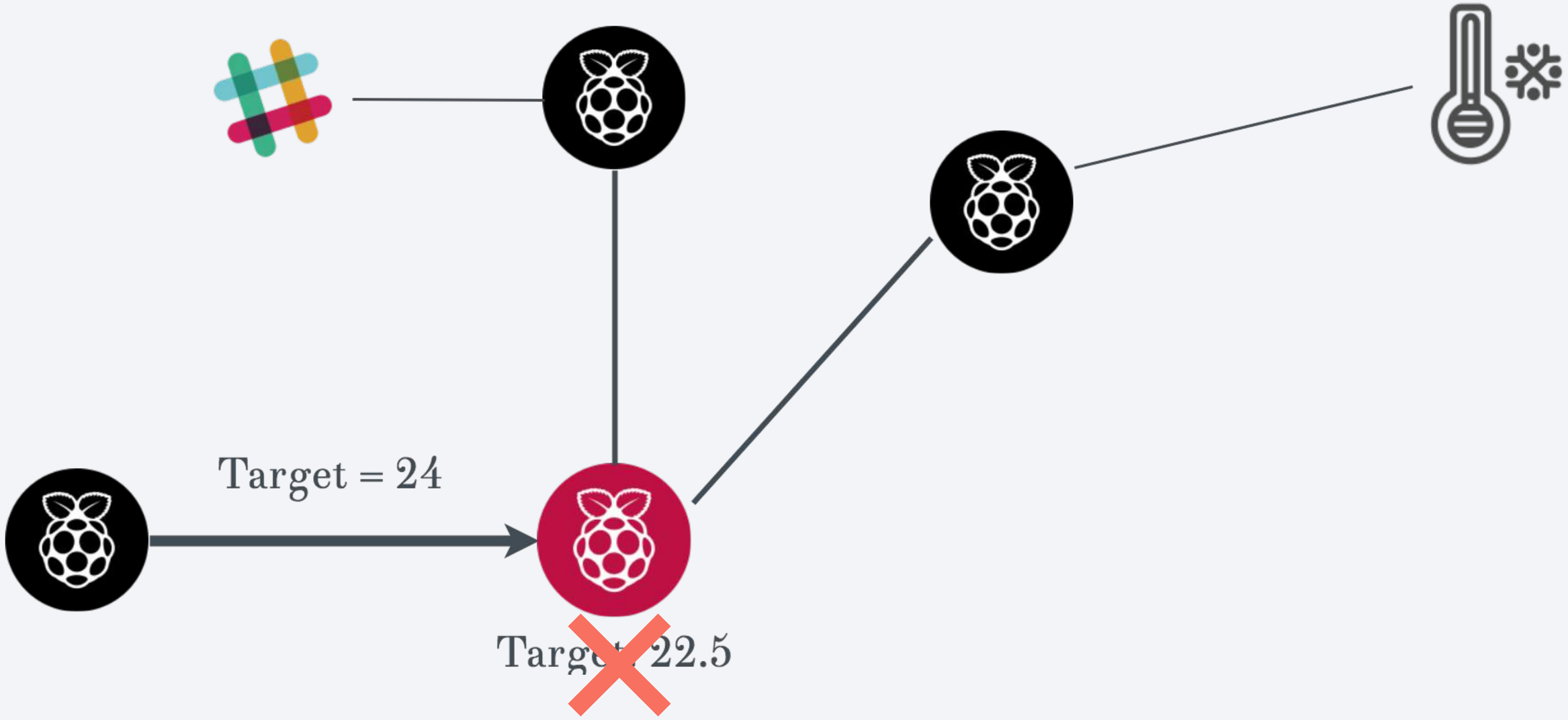
State-based CRDTs

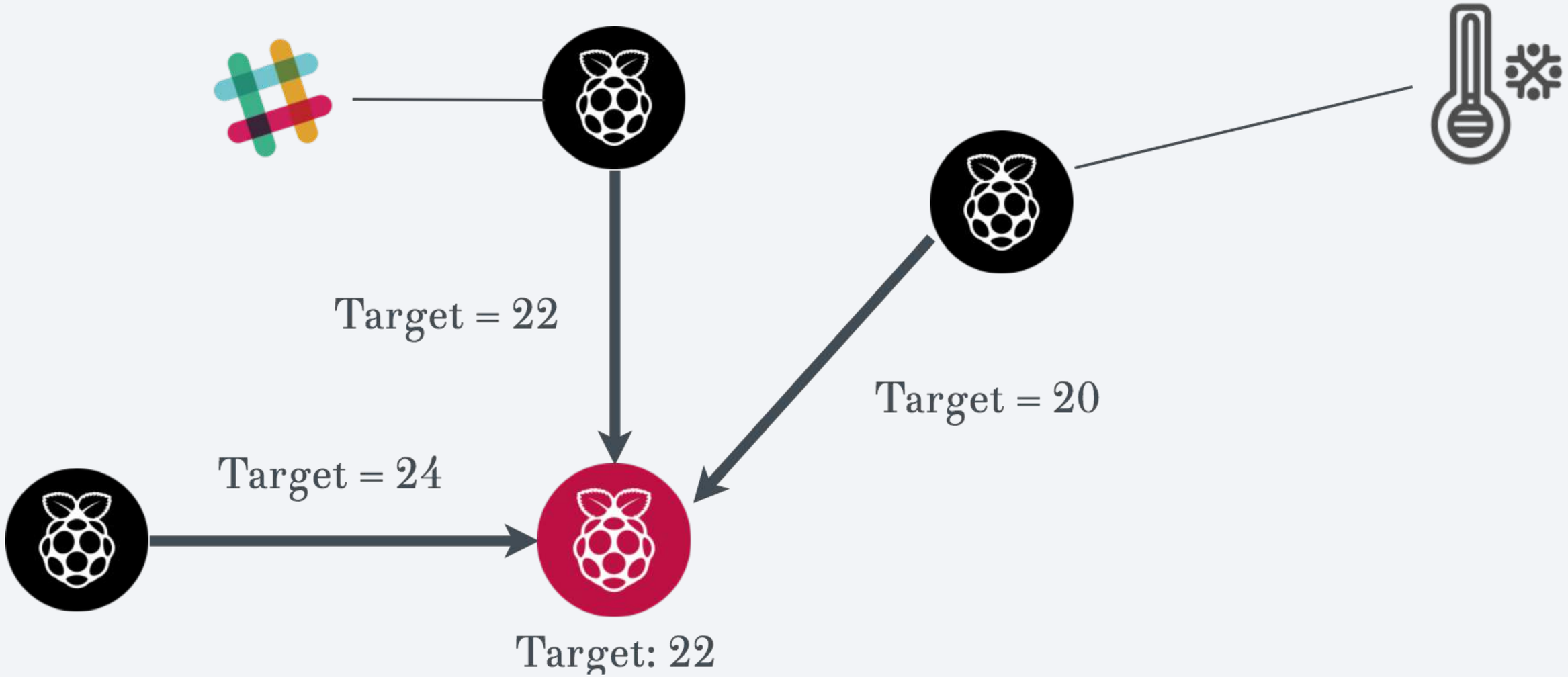

Target = 22
Target = 20

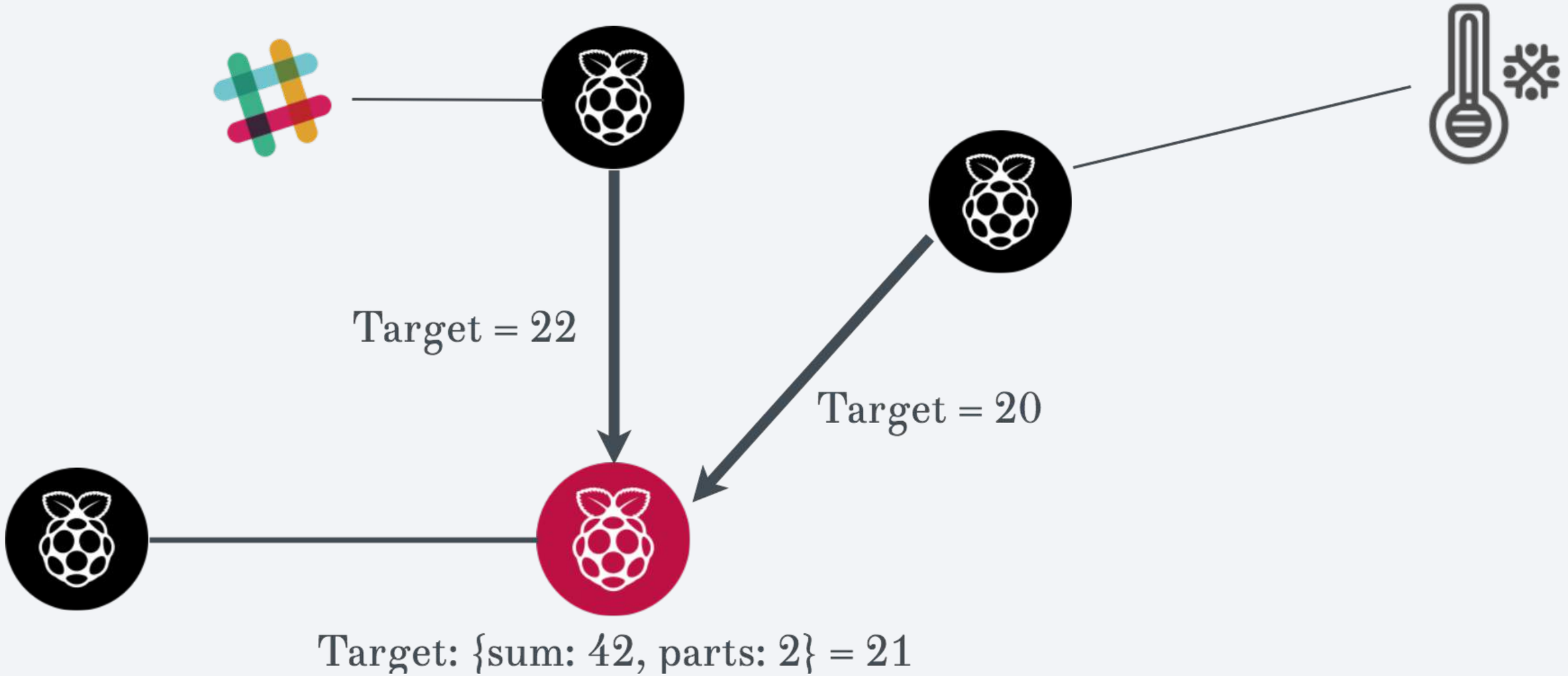

Target = 22
Target = 20
Target: 21

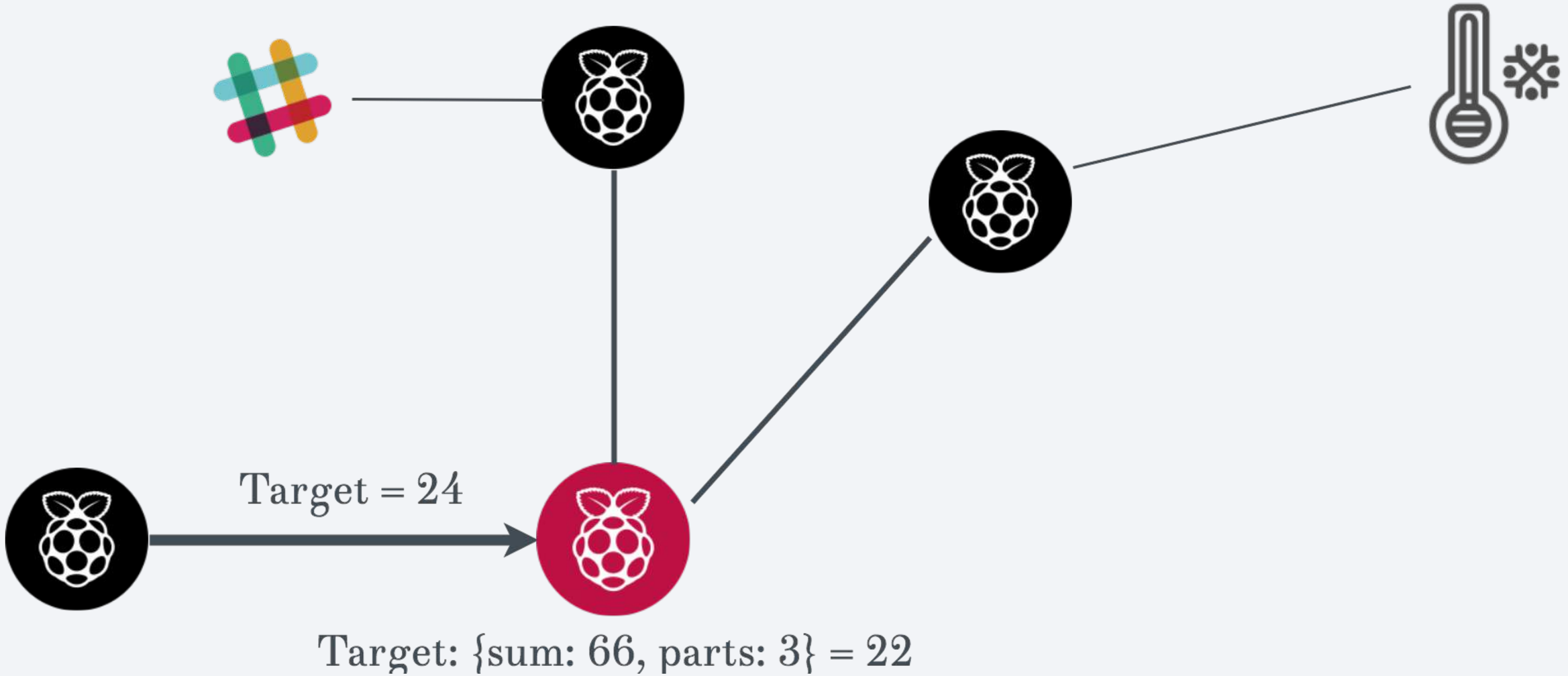

Target = 24
Target: 21


Target = 24
Target: 22.5


Target = 24
Target = 22.5


Target = 22
Target = 20
Target = 24
Target: 22


Target = 22
Target = 20
Target: {sum: 42, parts: 2} = 21


Target = 24
Target: {sum: 66, parts: 3} = 22

Wrapping up

System resources matter

System resources matter

your algorithms should account for them

There are models.

Use them.

Distributed System Checklist

Distributed System Checklist

- Is the number of processes known or finite?

Distributed System Checklist

- Is the number of processes known or finite?
- Is there a global notion of time?

Distributed System Checklist

- Is the number of processes known or finite?
- Is there a global notion of time?
- Is the network reliable?

Distributed System Checklist

- Is the number of processes known or finite?
- Is there a global notion of time?
- Is the network reliable?
- Is there full connectivity?

Distributed System Checklist

- Is the number of processes known or finite?
- Is there a global notion of time?
- Is the network reliable?
- Is there full connectivity?
- What happens when a process crashes?

It really doesn’t change that much

CRDTs aren’t a golden hammer

Reinventing the wheel is stupid

Fernando Mendes
@fribmendes

I spent one year working on a P2P smart office. Last Saturday I re-did the whole thing in one afternoon using IPFS Pub/Sub. Choose your tools wisely.

12:23 AM	**mendes**	stratos meeting room temp
12:23 AM	**stratos** APP	I'm sorry, I don't have that data.
12:23 AM	**mendes**	stratos meeting room temp
12:23 AM	**stratos** APP	temp is 22
12:23 AM	**mendes**	stratos meeting room hum
12:23 AM	**stratos** APP	hum is 30
12:24 AM	**mendes**	stratos meeting room sound
12:24 AM	**stratos** APP	I'm sorry. I don't have that data
12:24 AM	**mendes**	stratos what is love
12:24 AM	**stratos** APP	I'm sorry, I don't have that data.

Knee-Deep Into P2P

A Tale of Fail

@fribmendes