Koala Protocol

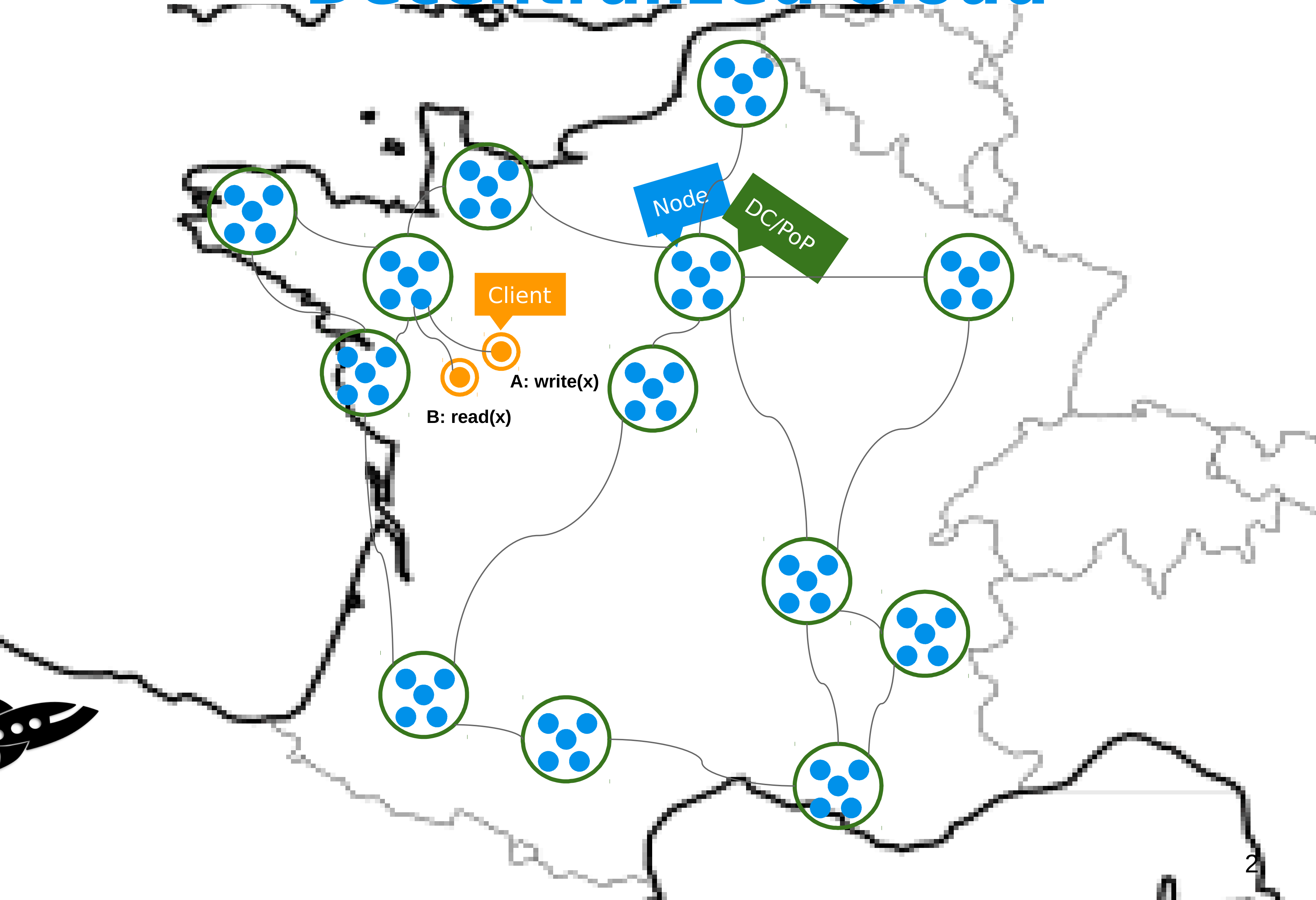
- A flat, lazy and topology-aware protocol for decentralized clouds

Genc Tato

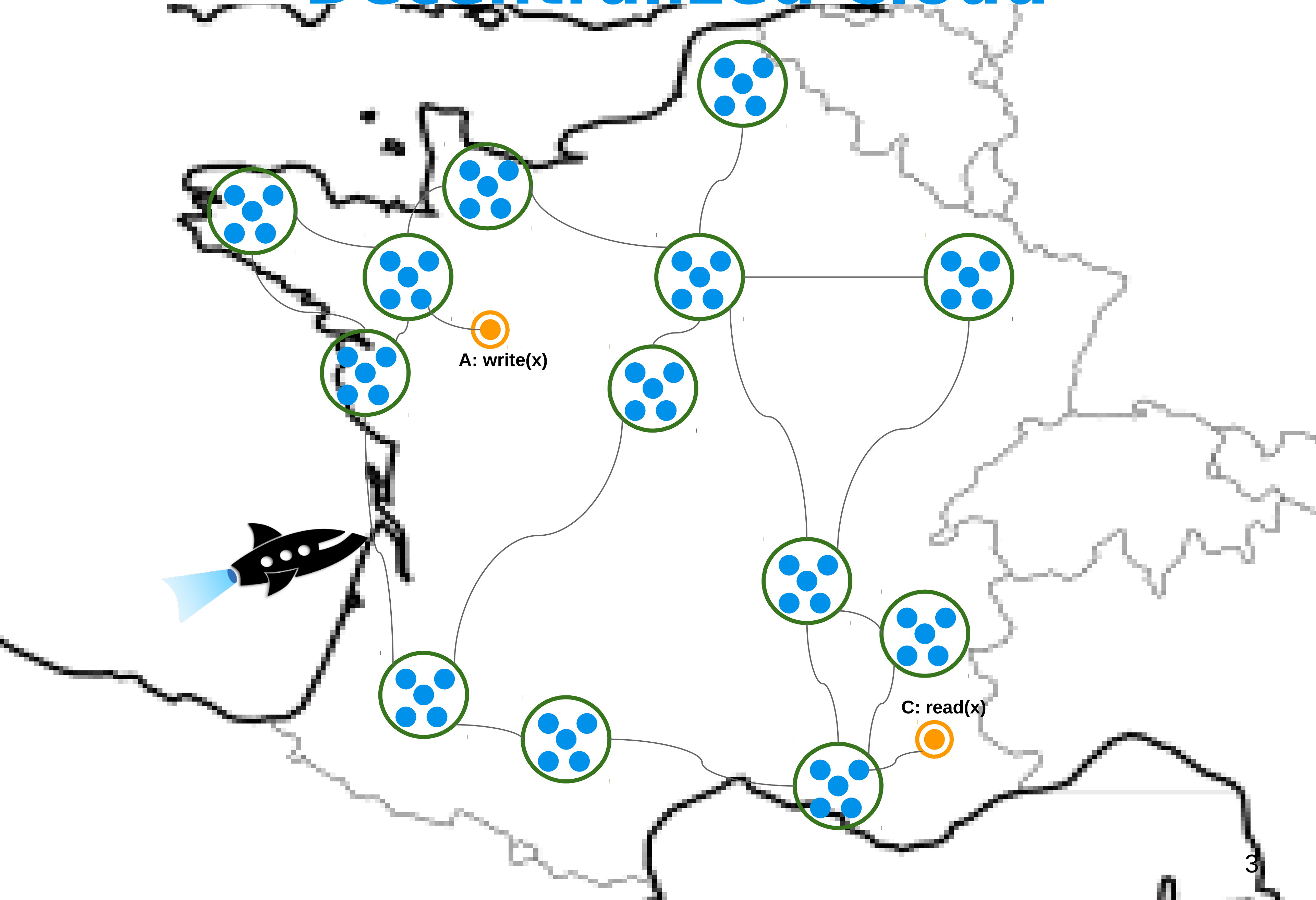
Supervised by:

Cédric Tedeschi
Marin Bertier

Decentralized Cloud



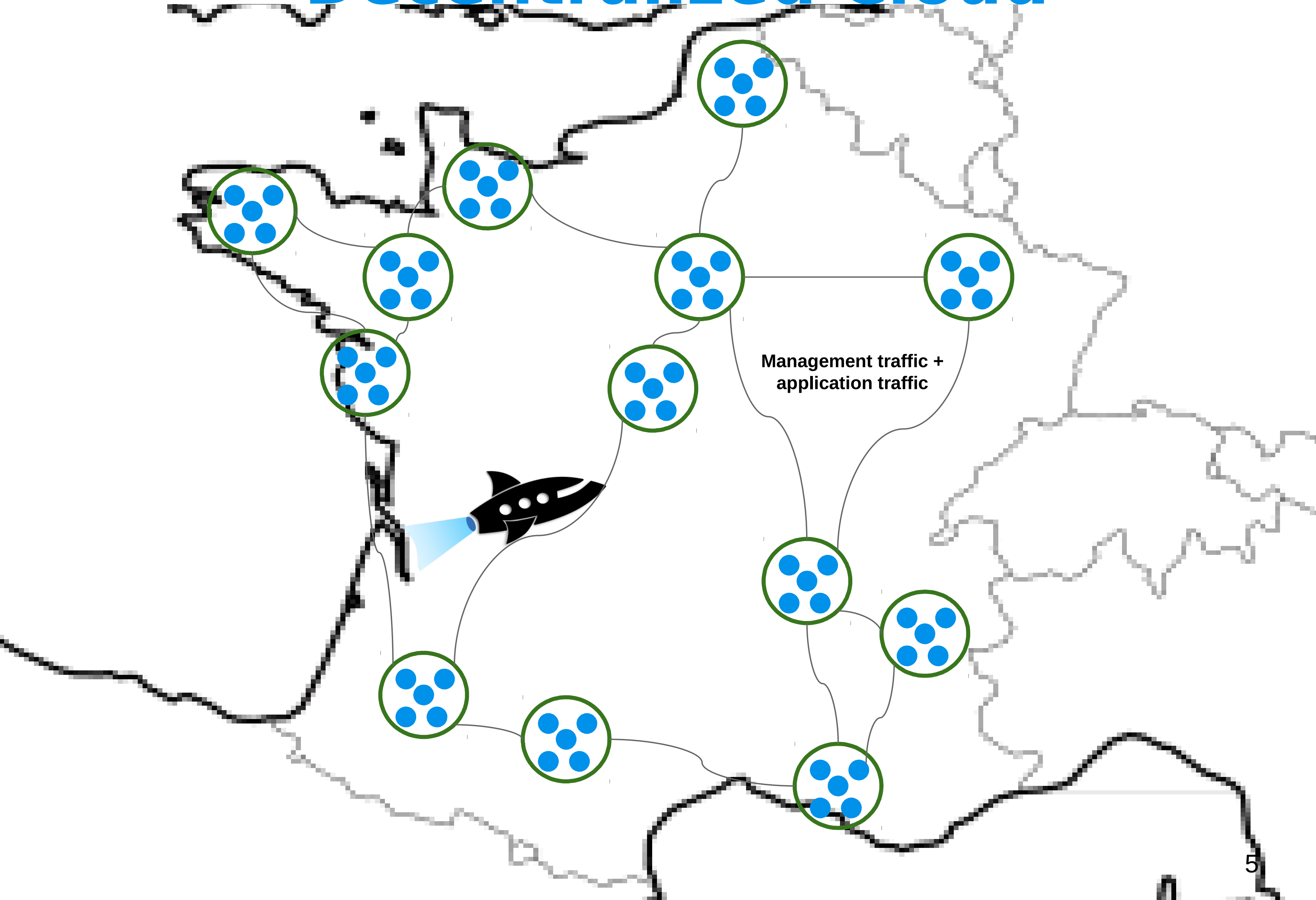
Decentralized Cloud



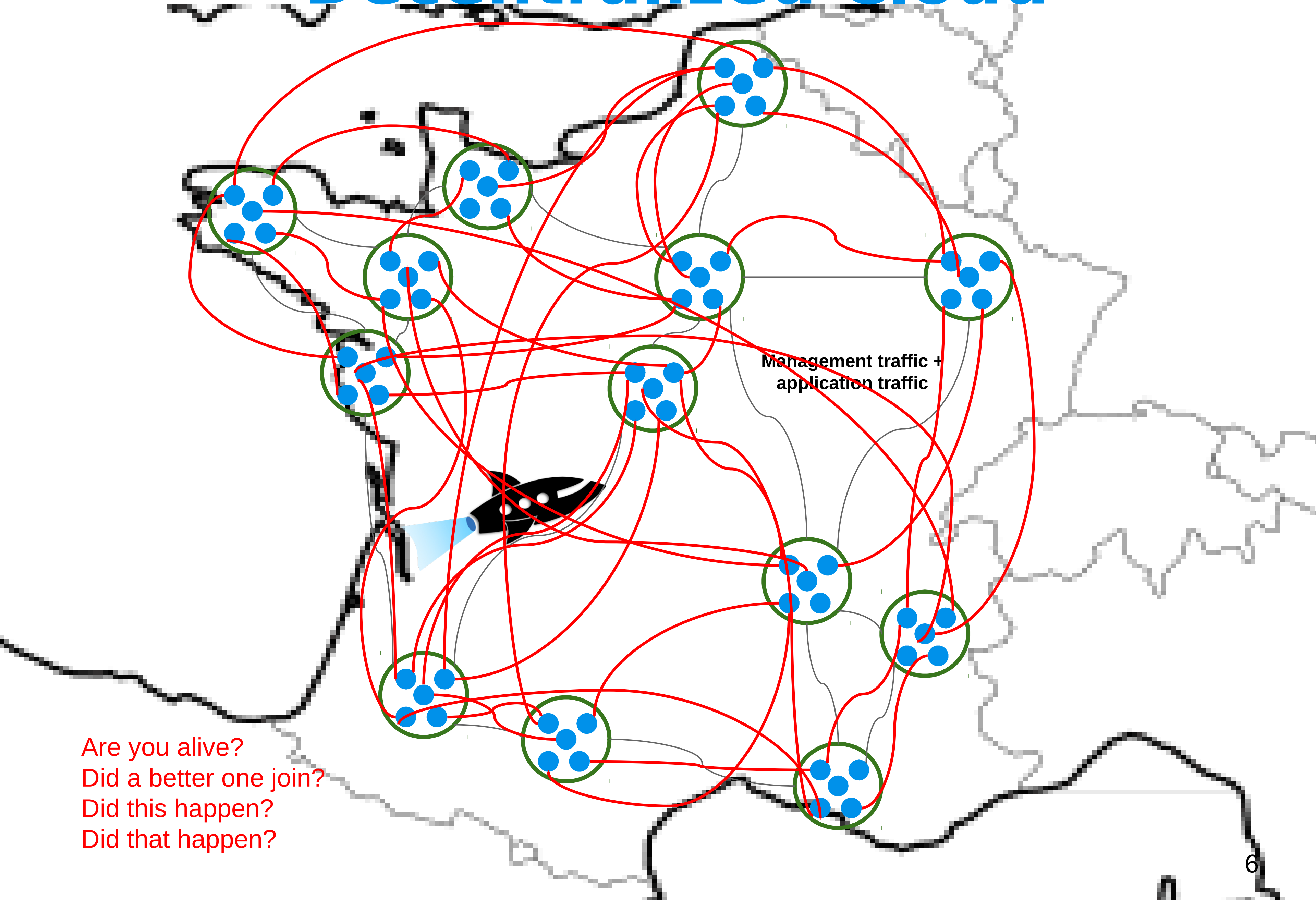
An Overlay Network

...but how should this overlay be?

Decentralized Cloud



Decentralized Cloud



Are you alive?
Did a better one join?
Did this happen?
Did that happen?

An Overlay Network

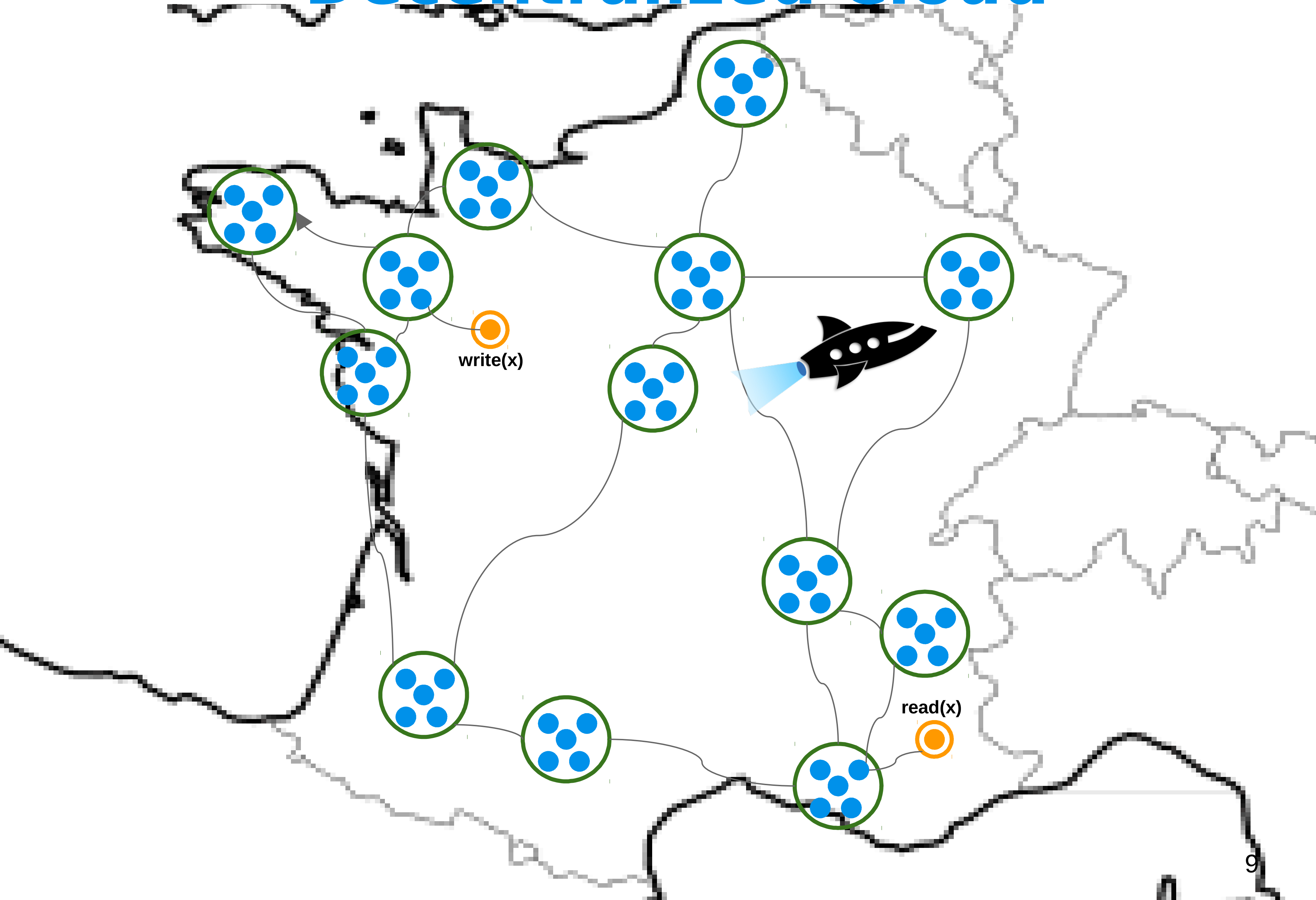
...but how should this overlay be?

An Overlay Network

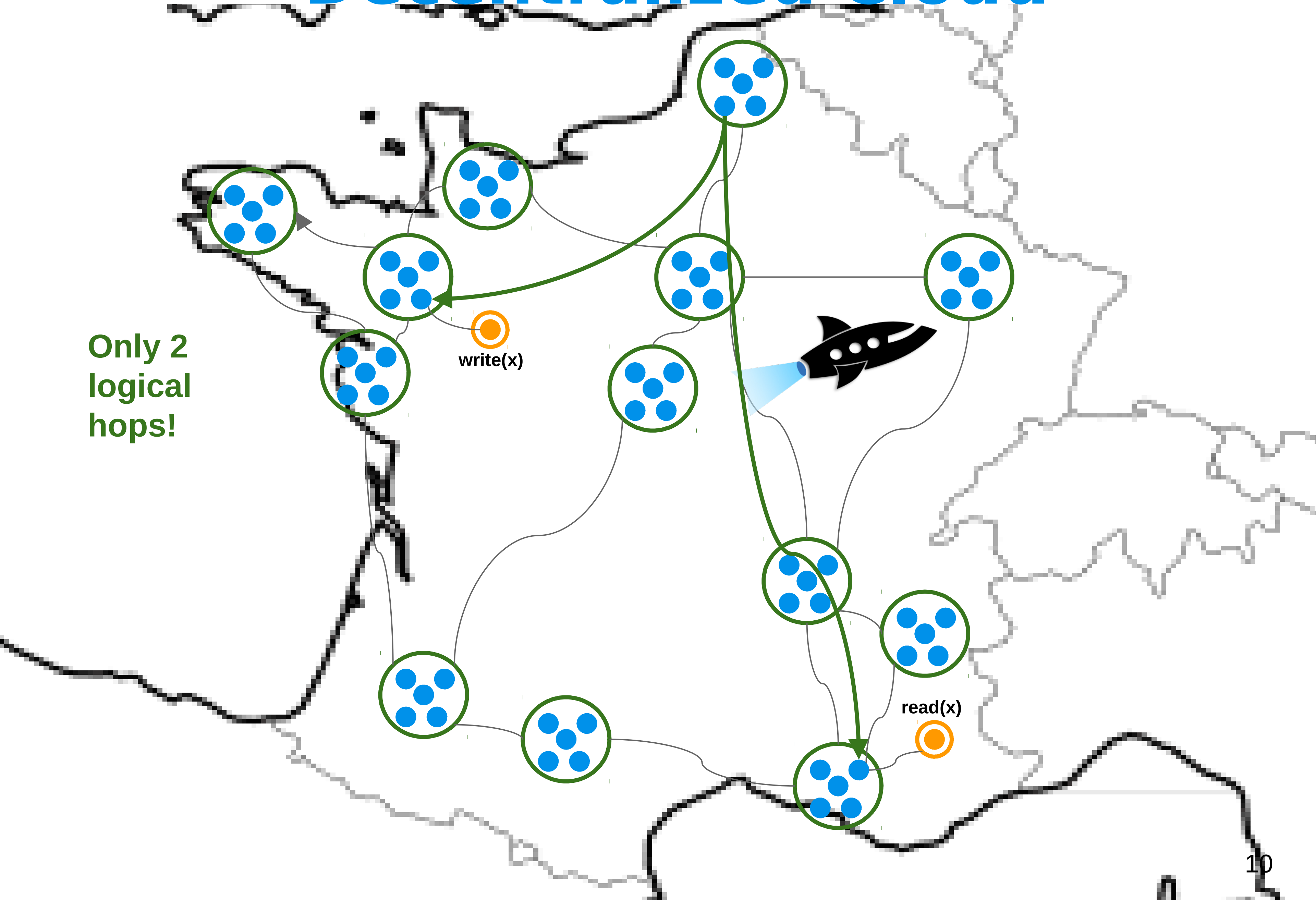
...but how should this overlay be?

1.Lazy (update overlay only when needed)

Decentralized Cloud

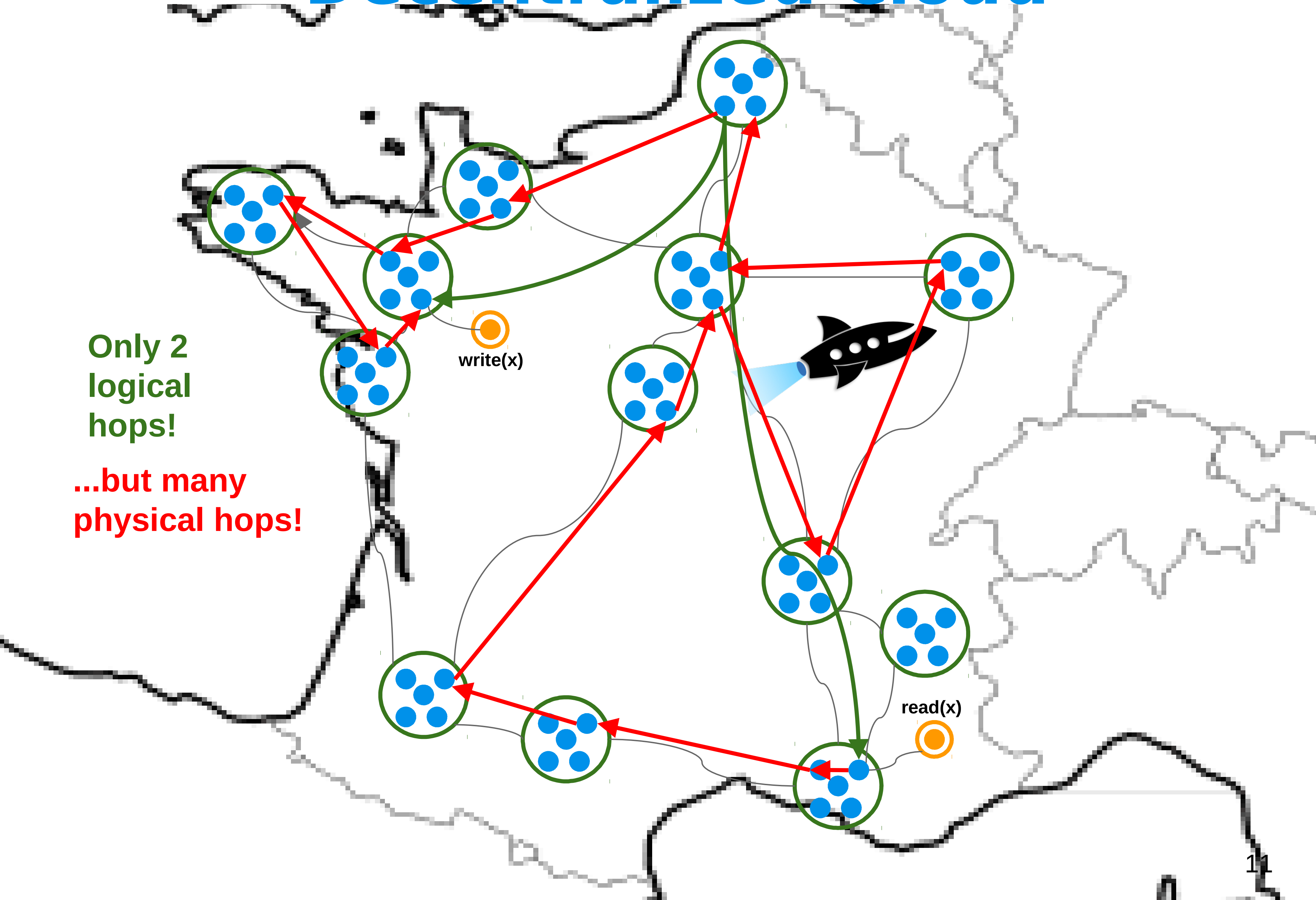


Decentralized Cloud



Only 2 logical hops!

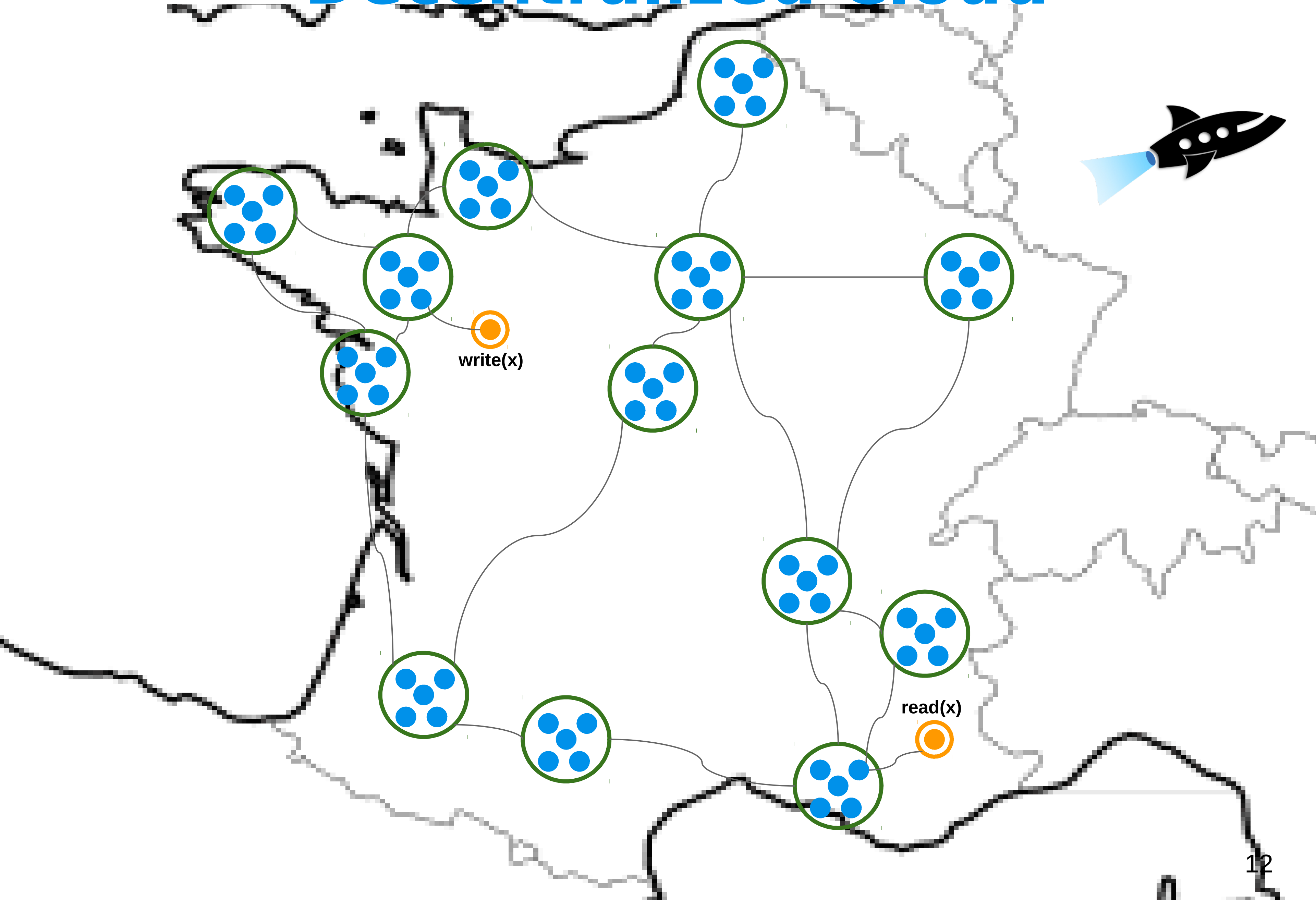
Decentralized Cloud



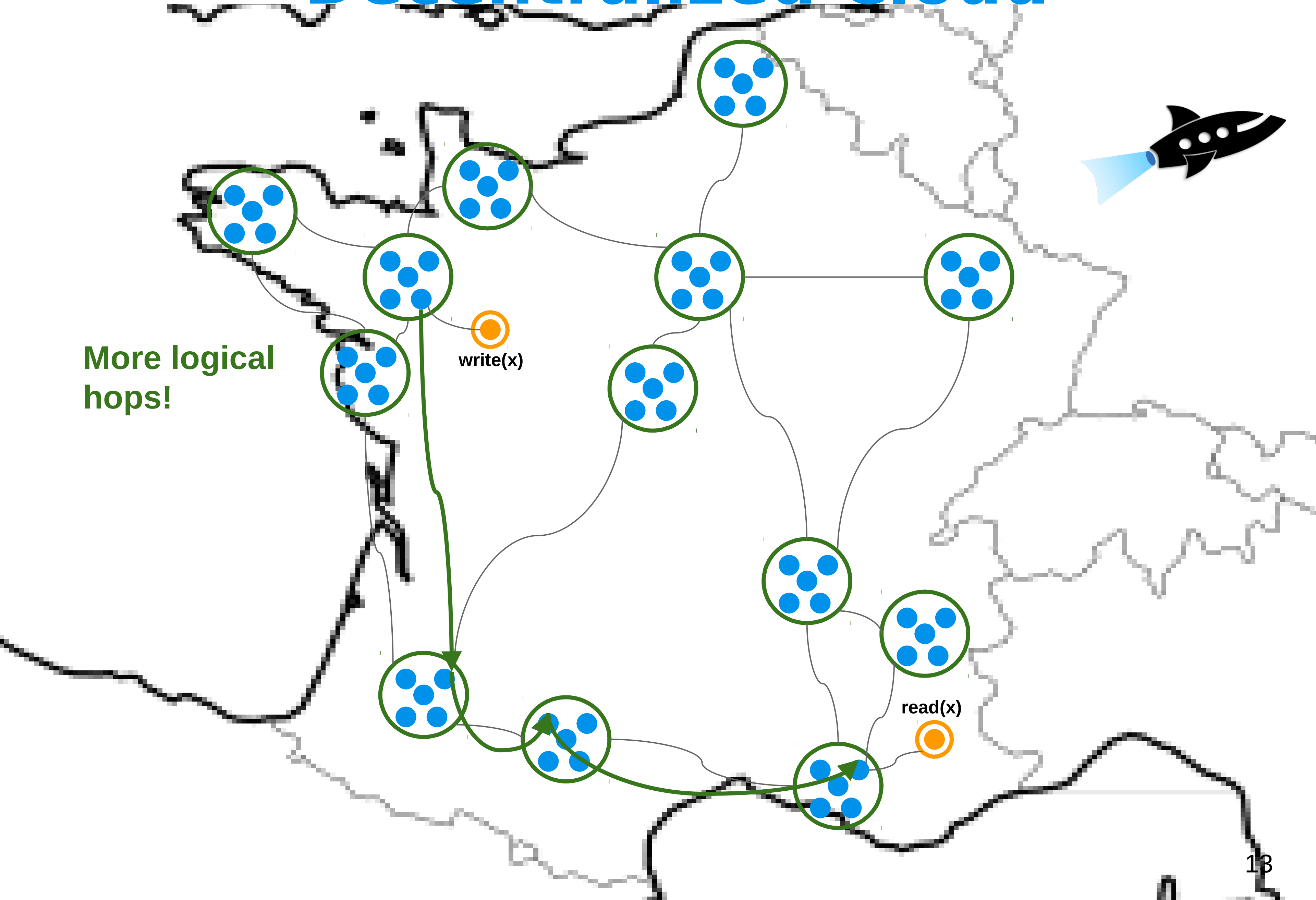
Only 2 logical hops!

...but many physical hops!

Decentralized Cloud

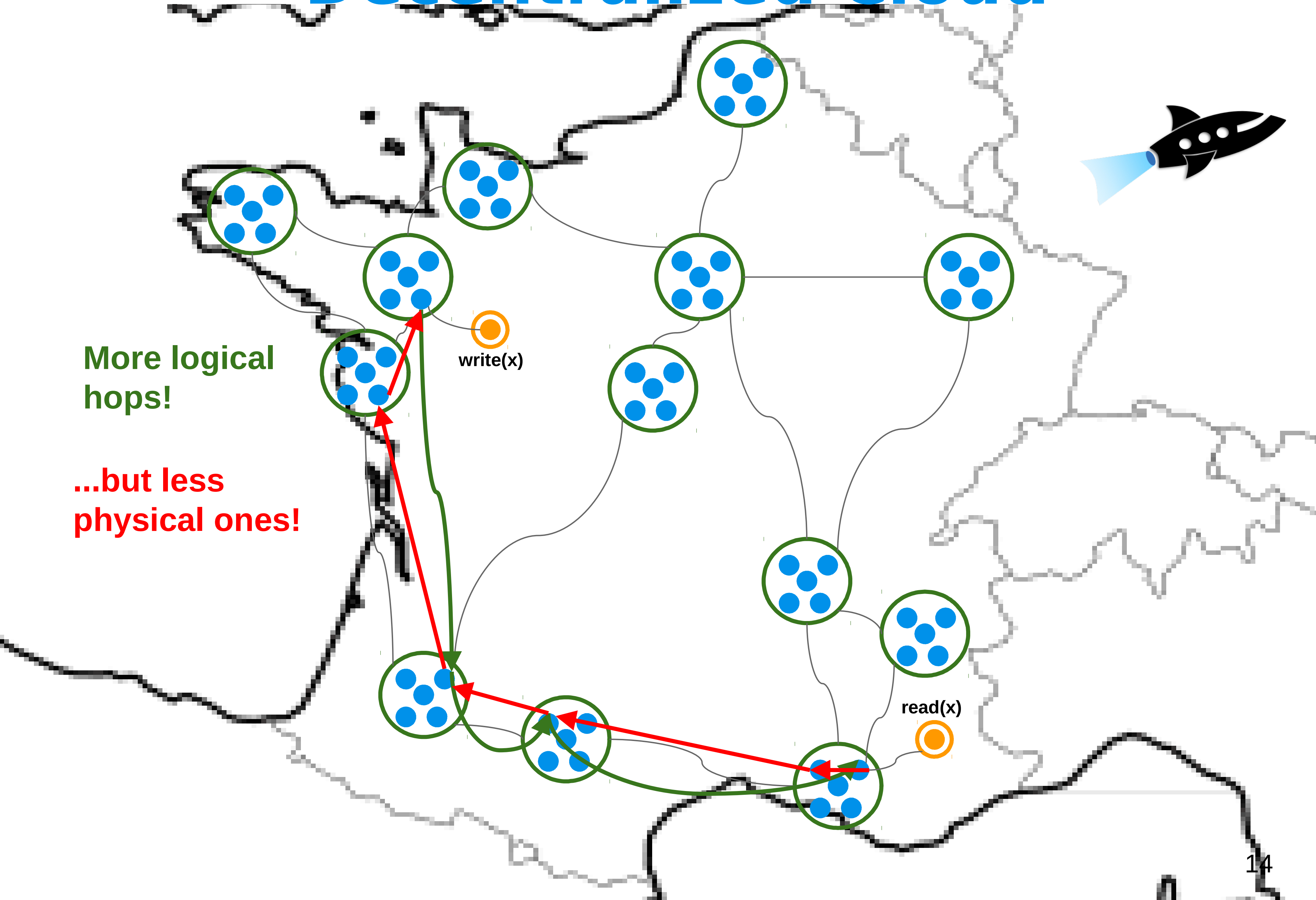


Decentralized Cloud



More logical hops!

Decentralized Cloud



More logical hops!

...but less physical ones!

An Overlay Network

...but how should this overlay be?

1. Lazy (update overlay only when needed)

An Overlay Network

...but how should this overlay be?

1.Lazy (update overlay only when needed)

2.Locality-aware

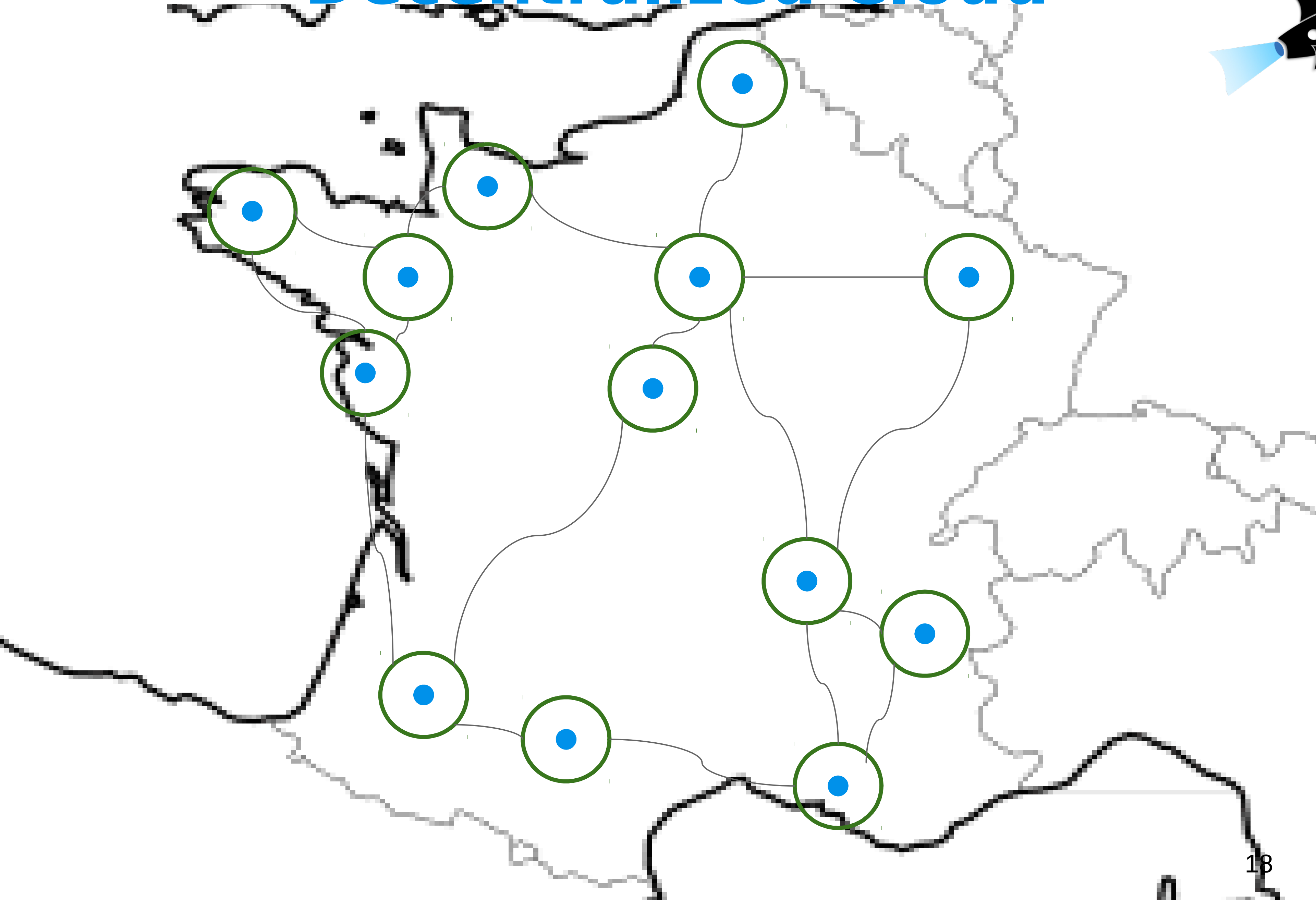
How to?

1. How do we implement laziness?
2. How do we integrate locality-awareness?

First, focus on communication between different PoPs.

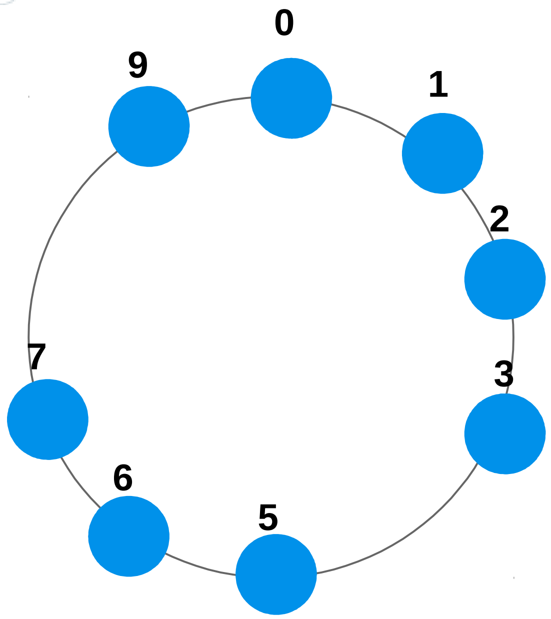
(nodes in same PoPs is discussed later)

Decentralized Cloud



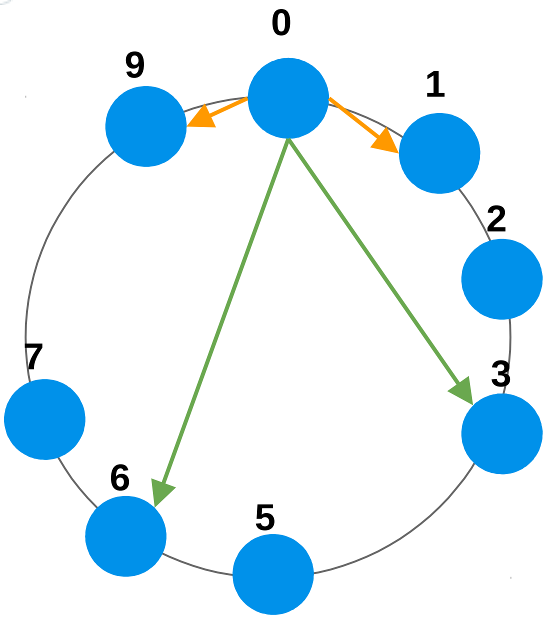
The basics

Nodes are organized in a ring and are identified by a circular id.



The basics

Nodes are organized in a ring and are identified by a circular id.



Routing table of node 0

ID	IP	RTT	Dist	IID
9	a.a.a.a	50	1	-
1	b.b.b.b	150	1	-
3	x.x.x.x	125	3	4
6	y.y.y.y	250	4	7

Neighbors

Long links

Ideal long links

Still want to be a **O(log N)** hops protocol

Ideal long links

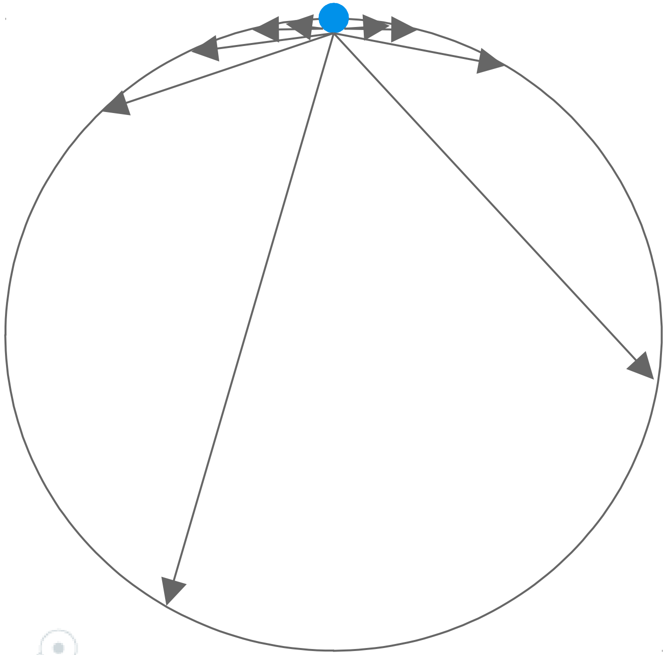
Still want to be a **O(log N)** hops protocol

Tell me who your long links are, I will tell you how efficient your routing is!

Ideal long links

Still want to be a **O(log N)** hops protocol

Tell me who your long links are, I will tell you how efficient your routing is!



Continuous Kleinberg distribution:

$$p(d) = 1/(d * \mathit{ln}\ N)$$

Where:

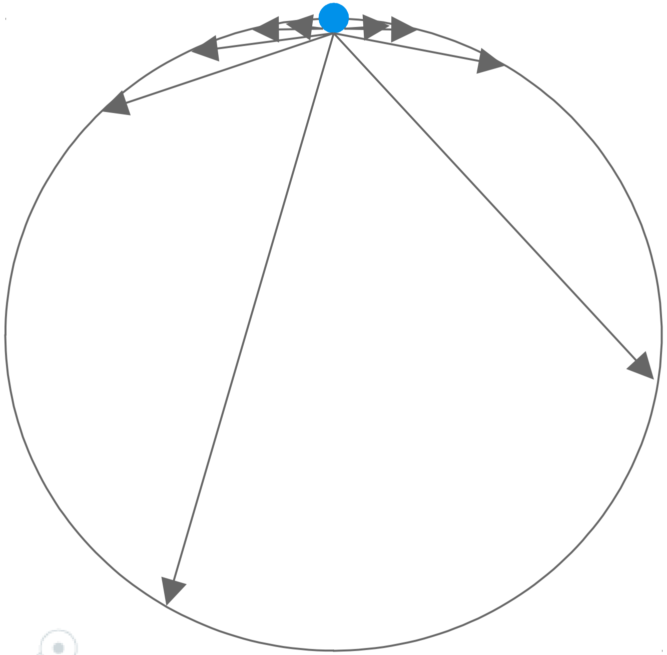
d = logical distance,

N = total nr. of nodes

Ideal long links

Still want to be a **O(log N)** hops protocol

Tell me who your long links are, I will tell you how efficient your routing is!



Continuous Kleinberg distribution:

p(d) = 1/(d**ln*** **N)**

Where:

d = logical distance,

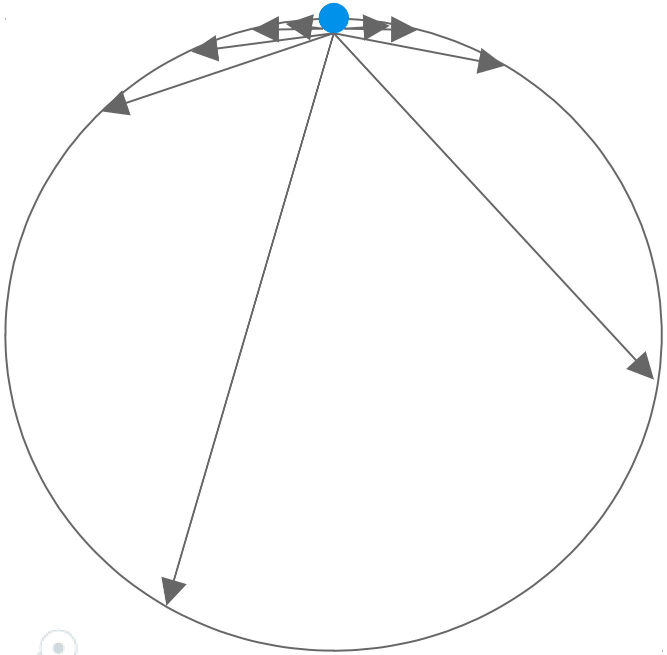
N = total nr. of nodes

Generate IDs using p(d) -> **Ideal IDs**

Ideal long links

Generating IDs does not mean contacting, it might not exist.

Lazy: do not search, wait to learn



Continuous Kleinberg distribution:

p(d) = 1/(d**ln***N)**

Where:

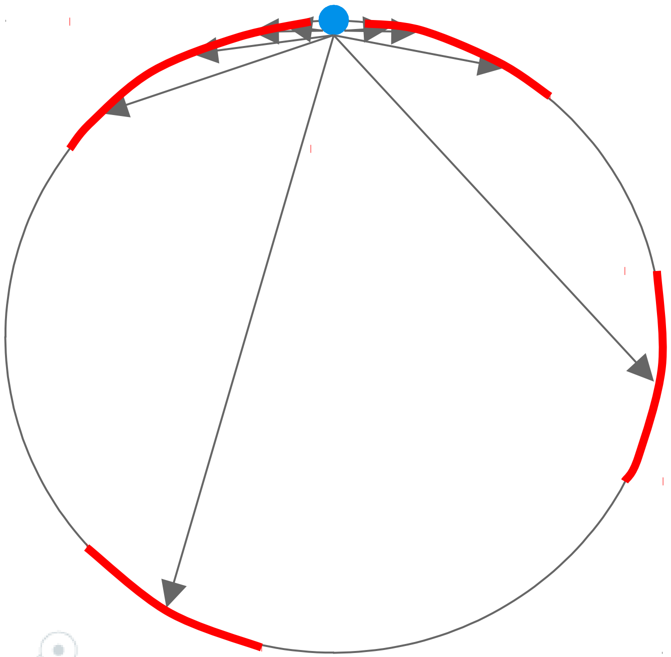
d = logical distance,

N = total nr. of nodes

Ideal long links

Generating IDs does not mean contacting, it might not exist.

Lazy: do not search, wait to learn



Continuous Kleinberg distribution:

$$p(d) = 1/(d*\mathit{ln}N)$$

Where:

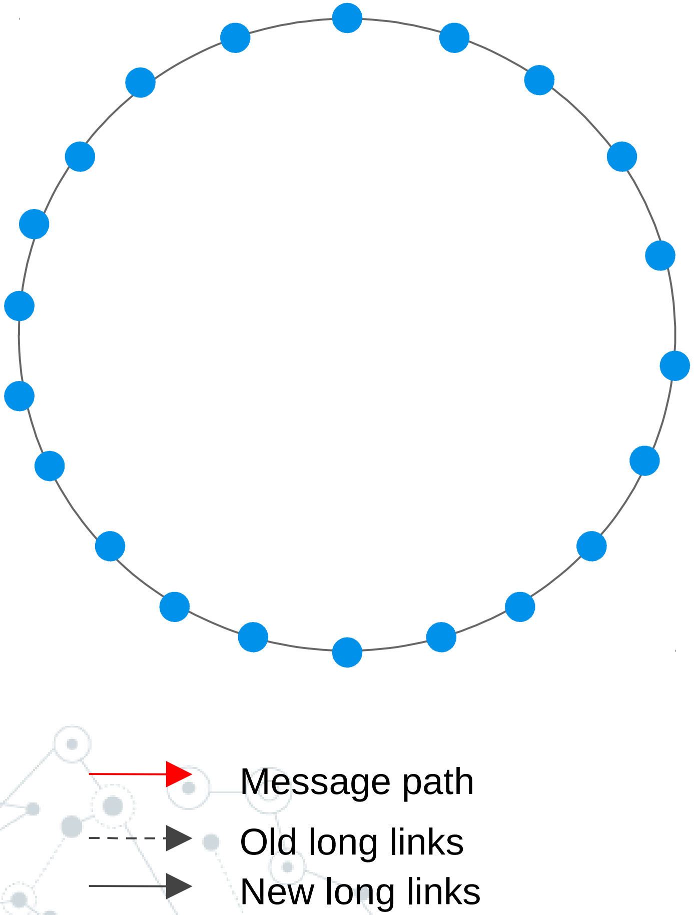
d = logical distance,

N = total nr. of nodes

Close to ideal is still ideal.

Laziness: Piggybacking

Embed information about nodes within the message



Message path

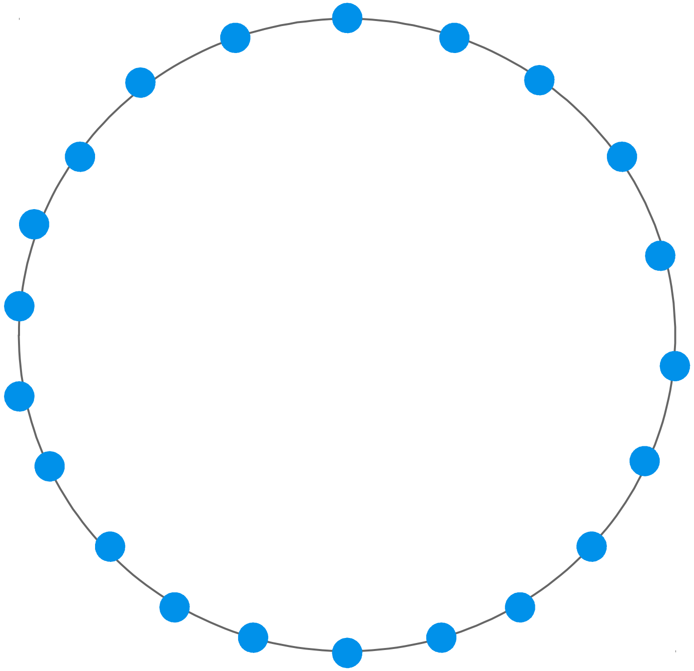
Old long links

New long links

2 sources:

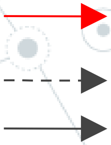
Laziness: Piggybacking

Embed information about nodes within the message



2 sources:

1. Nodes in the path



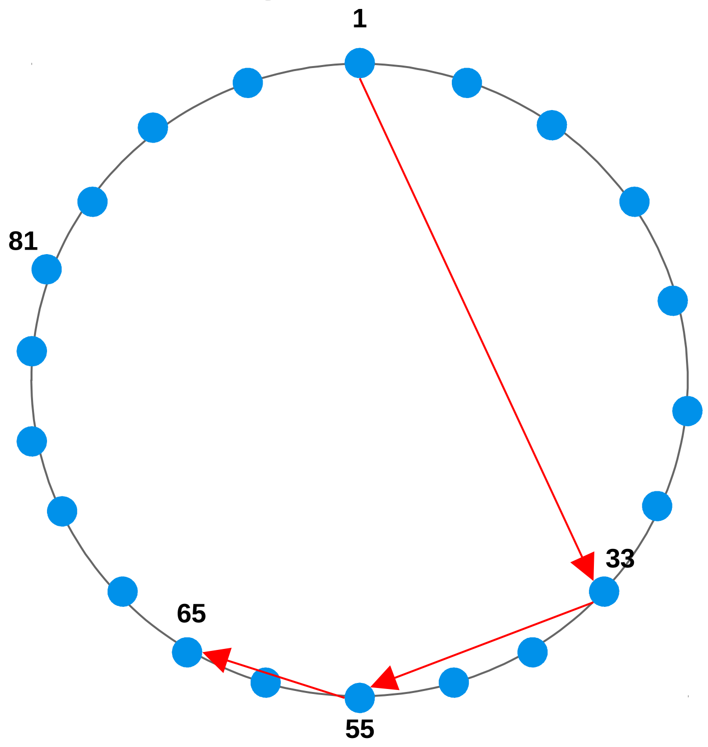
Message path

Old long links

New long links

Laziness: Piggybacking

Embed information about nodes within the message



2 sources:

1. Nodes in the path

Path: 1, 33, 55, 65

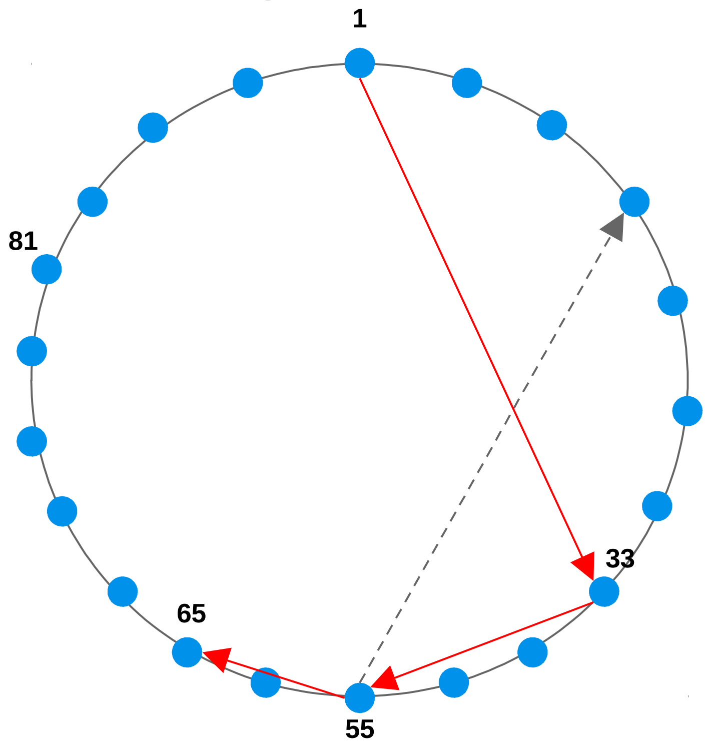
→ Message path

⇢ Old long links

→ New long links

Laziness: Piggybacking

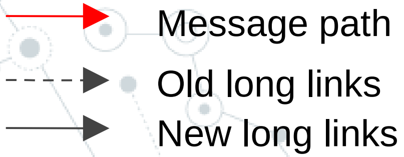
Embed information about nodes within the message



2 sources:

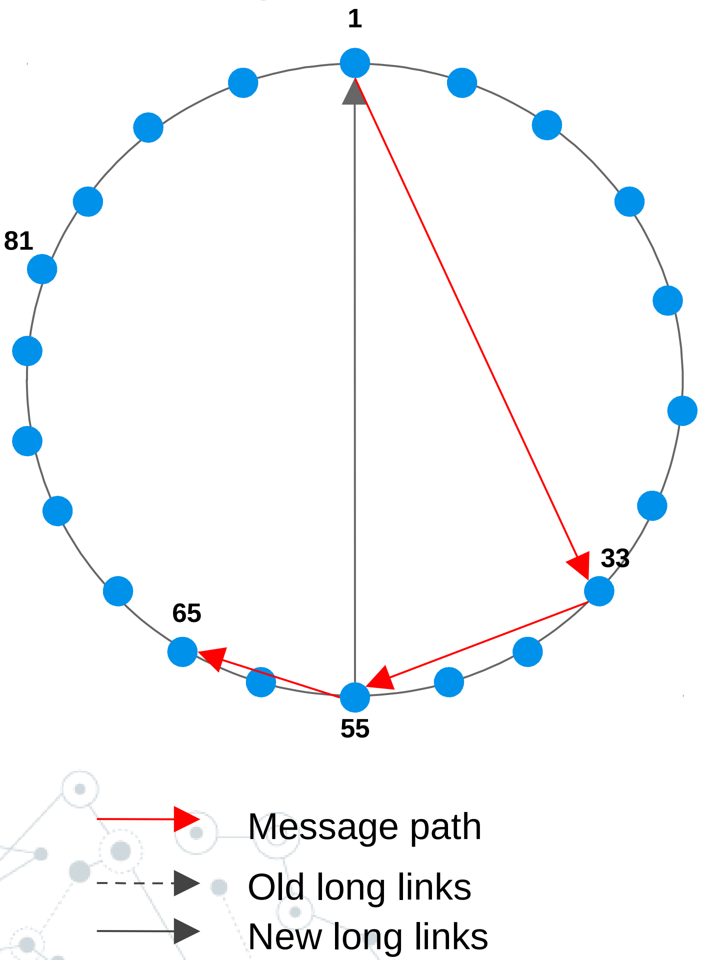
1. Nodes in the path

Path: 1, 33, 55, 65



Laziness: Piggybacking

Embed information about nodes within the message



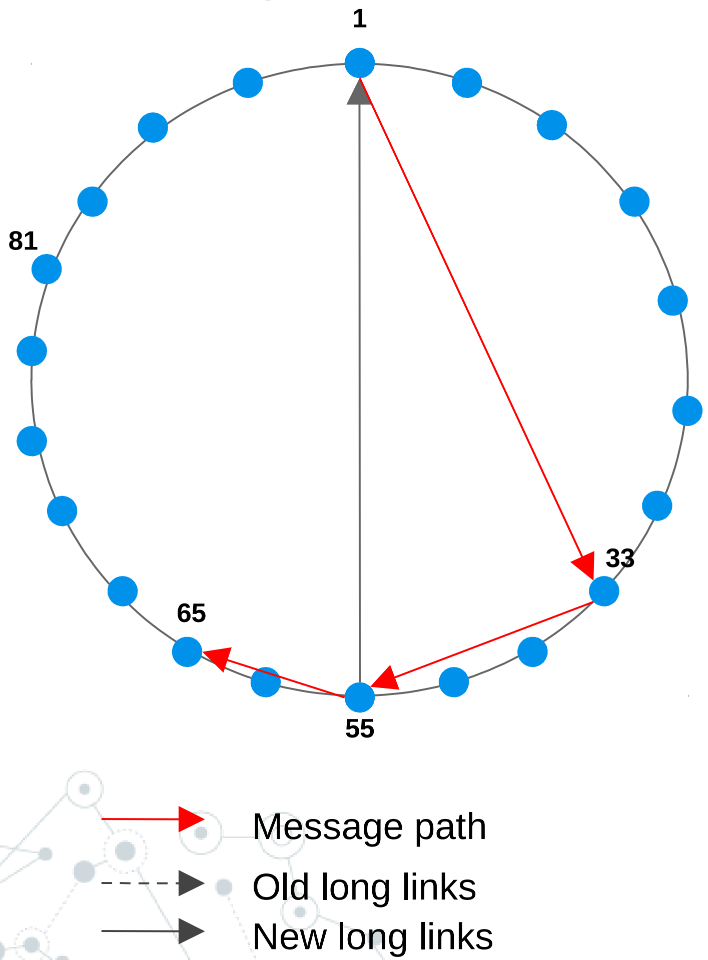
2 sources:

1. Nodes in the path

Path: 1, 33, 55, 65

Laziness: Piggybacking

Embed information about nodes within the message



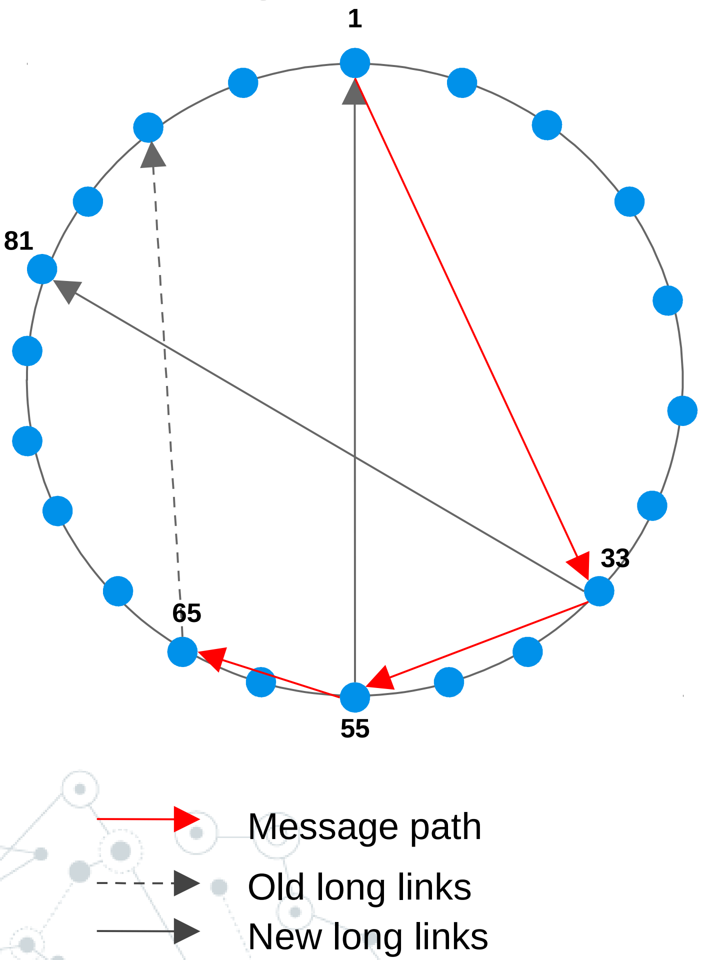
2 sources:

1. Nodes in the path
2. Nodes in the routing table of the nodes in the path

Path: 1, 33, 55, 65

Laziness: Piggybacking

Embed information about nodes within the message



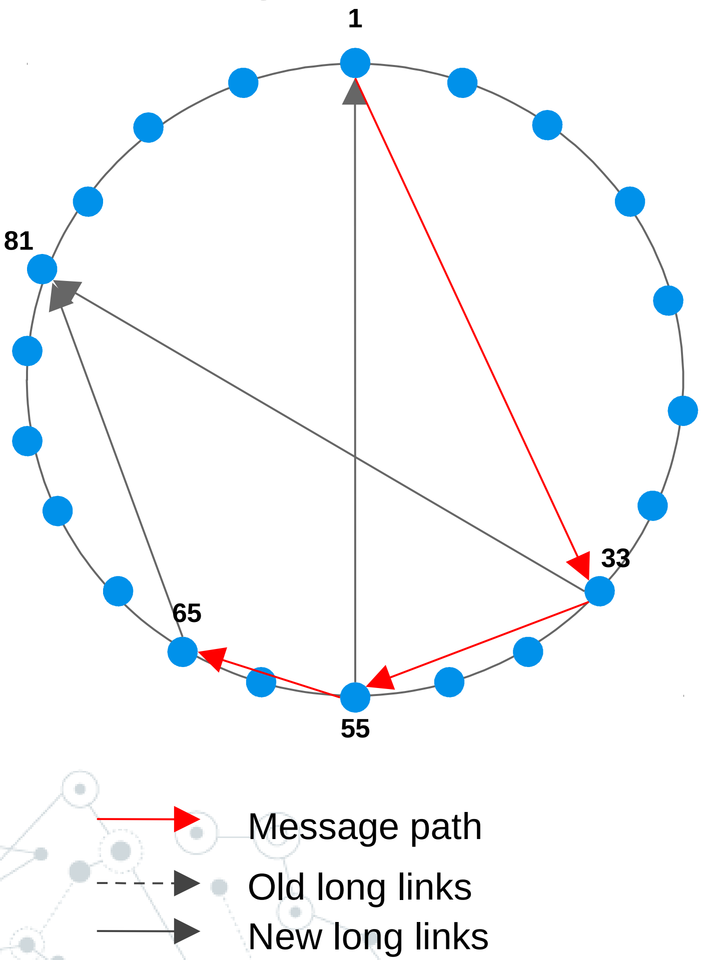
2 sources:

1. Nodes in the path
2. Nodes in the routing table of the nodes in the path

Path: 1, 33, 55, 65

Laziness: Piggybacking

Embed information about nodes within the message



2 sources:

1. Nodes in the path
2. Nodes in the routing table of the nodes in the path

Path: 1, 33, 55, 65

Locality-awareness

Find the cheapest logical path (latency-wise)

Locality-awareness

Find the cheapest logical path (latency-wise)

We can only choose the cheapest next hop!

Locality-awareness

Find the cheapest logical path (latency-wise)

We can only choose the cheapest next hop!

Keep right direction by reducing logical distance

Locality-awareness

Find the cheapest logical path (latency-wise)

We can only choose the cheapest next hop!

Keep right direction by reducing logical distance

A tradeoff between logical distance and latency

Locality-awareness

Find the cheapest logical path (latency-wise)

We can only choose the cheapest next hop!

Keep right direction by reducing logical distance

A tradeoff between logical distance and latency

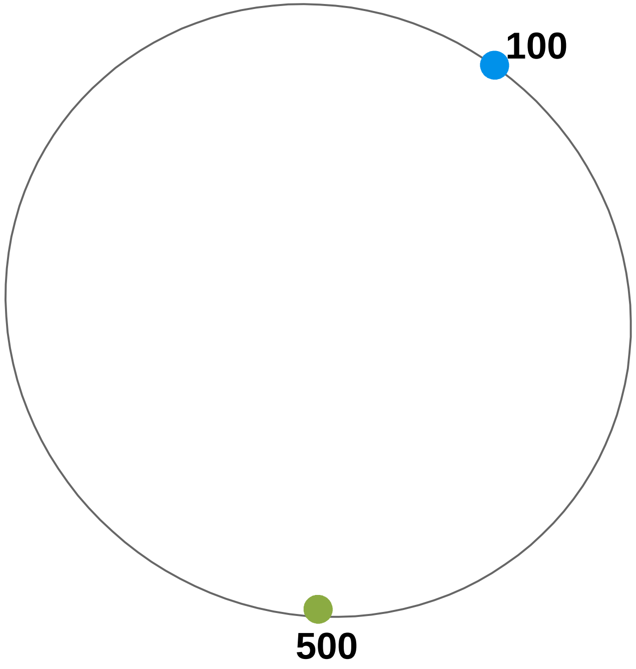
For each entry we calculate:

$$\mathbf{Q(re) = 1/(\alpha \times re.distance + (1-\alpha) \times norm(re.RTT))}$$

Locality-awareness: Routing

Route from **100** to **500**

N=1000

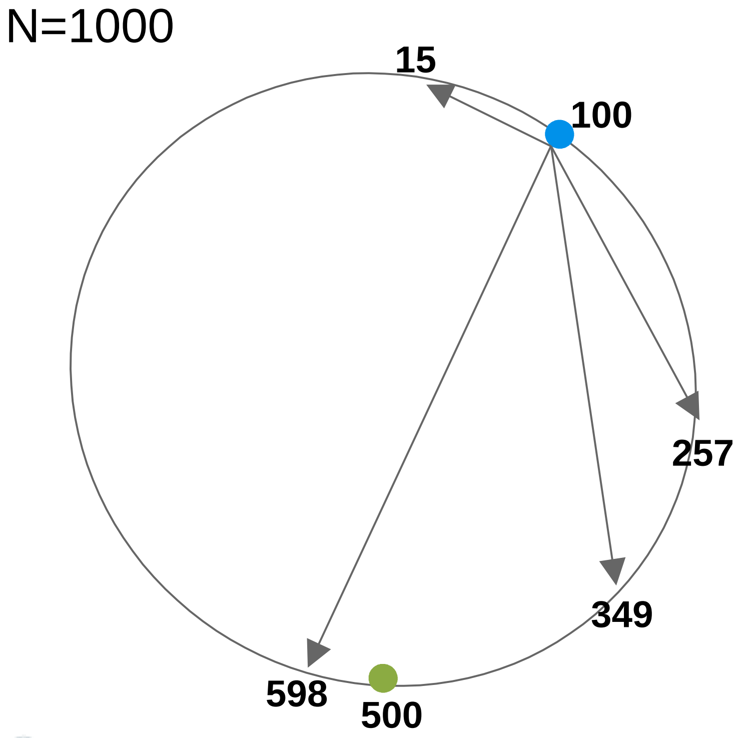


Routing table of node 100

Q(re) = 1/(α x re.distance + (1-α) x norm(re.RTT))

Locality-awareness: Routing

Route from **100** to **500**



Routing table of node 100

ID	RTT	DistToDest
15	340	485
257	105	243
349	194	151
598	1230	98

Q(re) = 1/(α x re.distance + (1-α) x norm(re.RTT))

Evaluation

Three criteria:

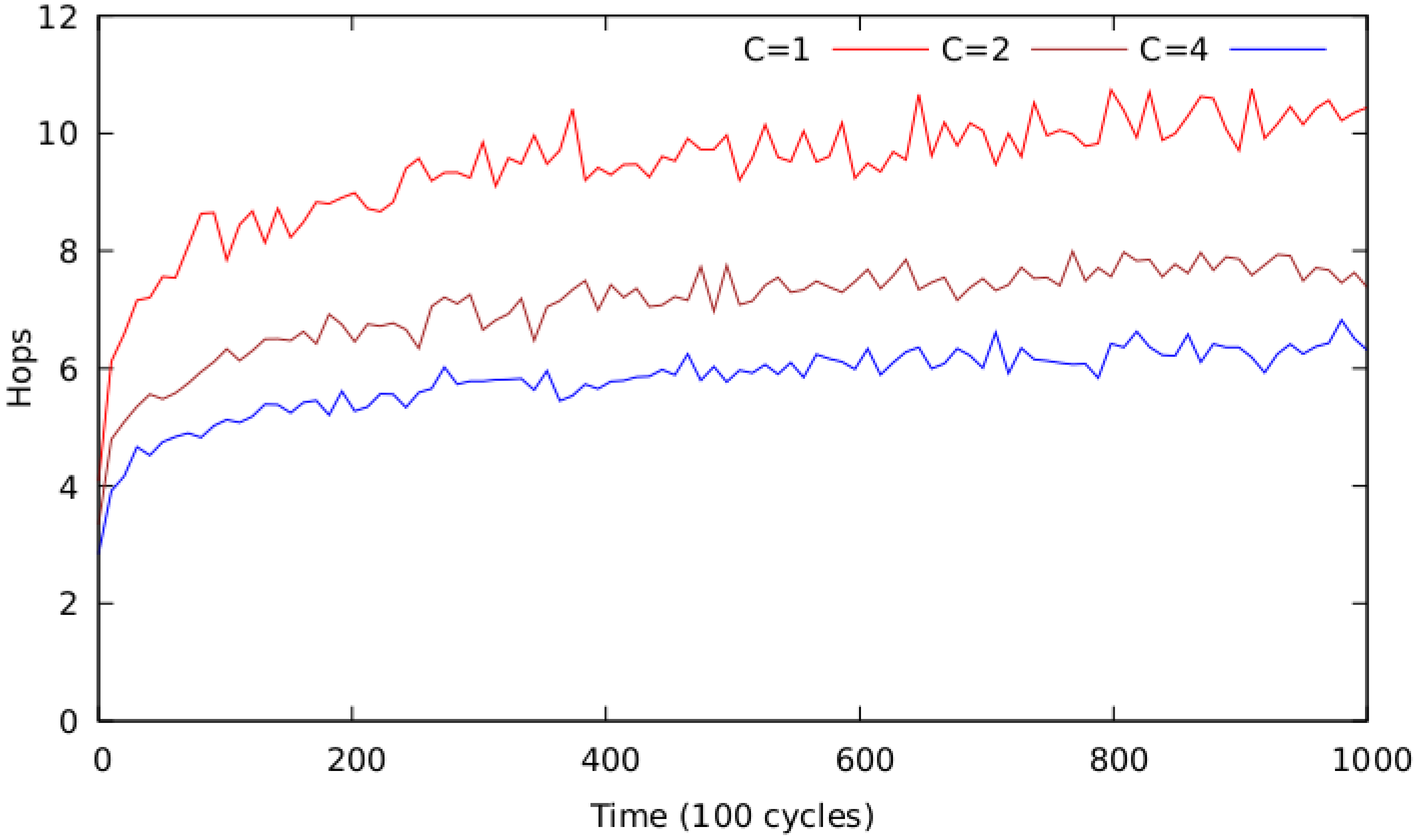
◎Scalability

◎Locality-awareness trade-off

◎Resilience

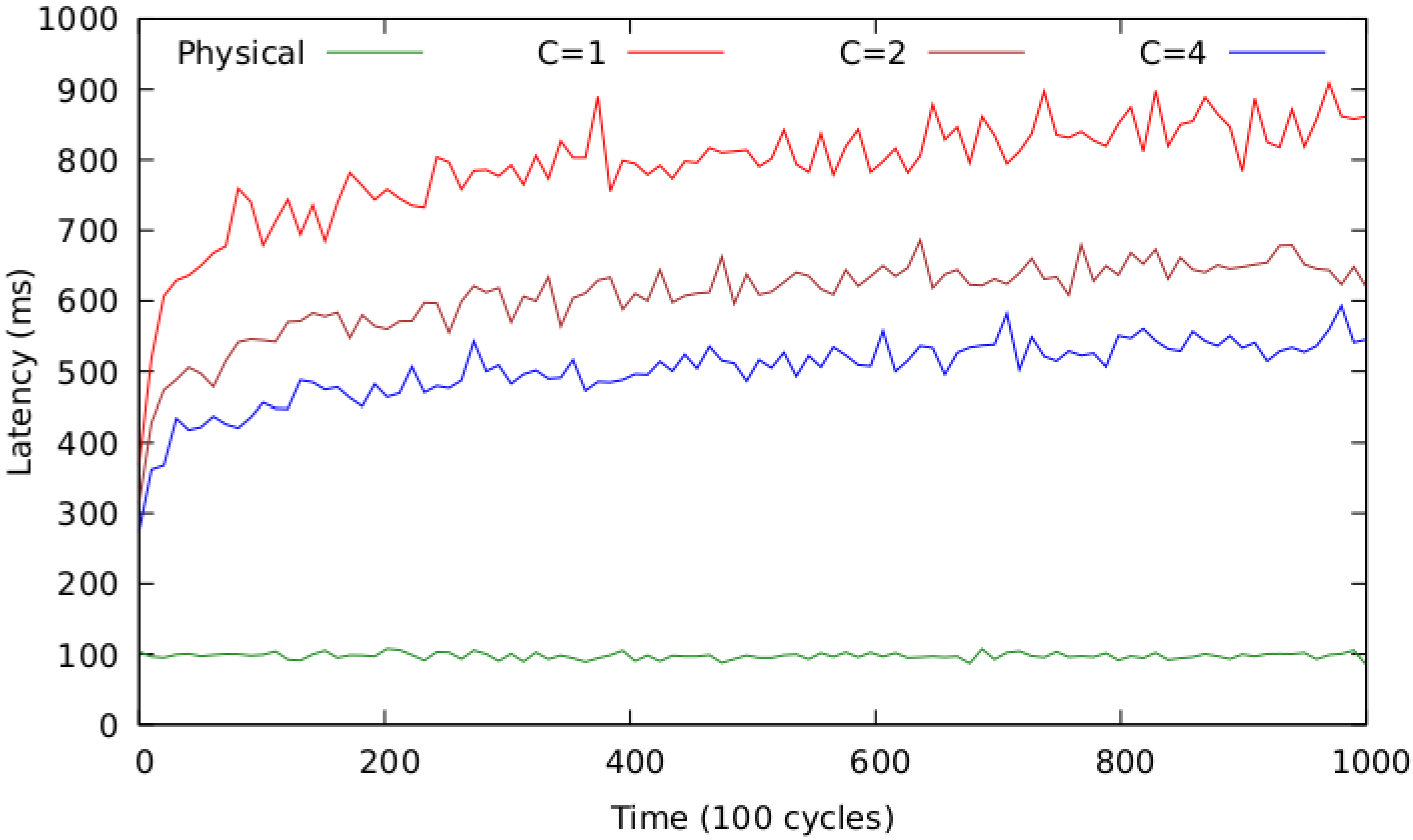
Scalability (hops)

Add nodes each cycle until 100K
# of long links = C * logN

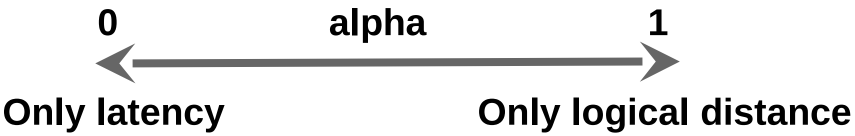


Scalability (latency)

Add nodes each cycle until 100K
# of long links = C * logN



Locality-aware vs Greedy (hops)

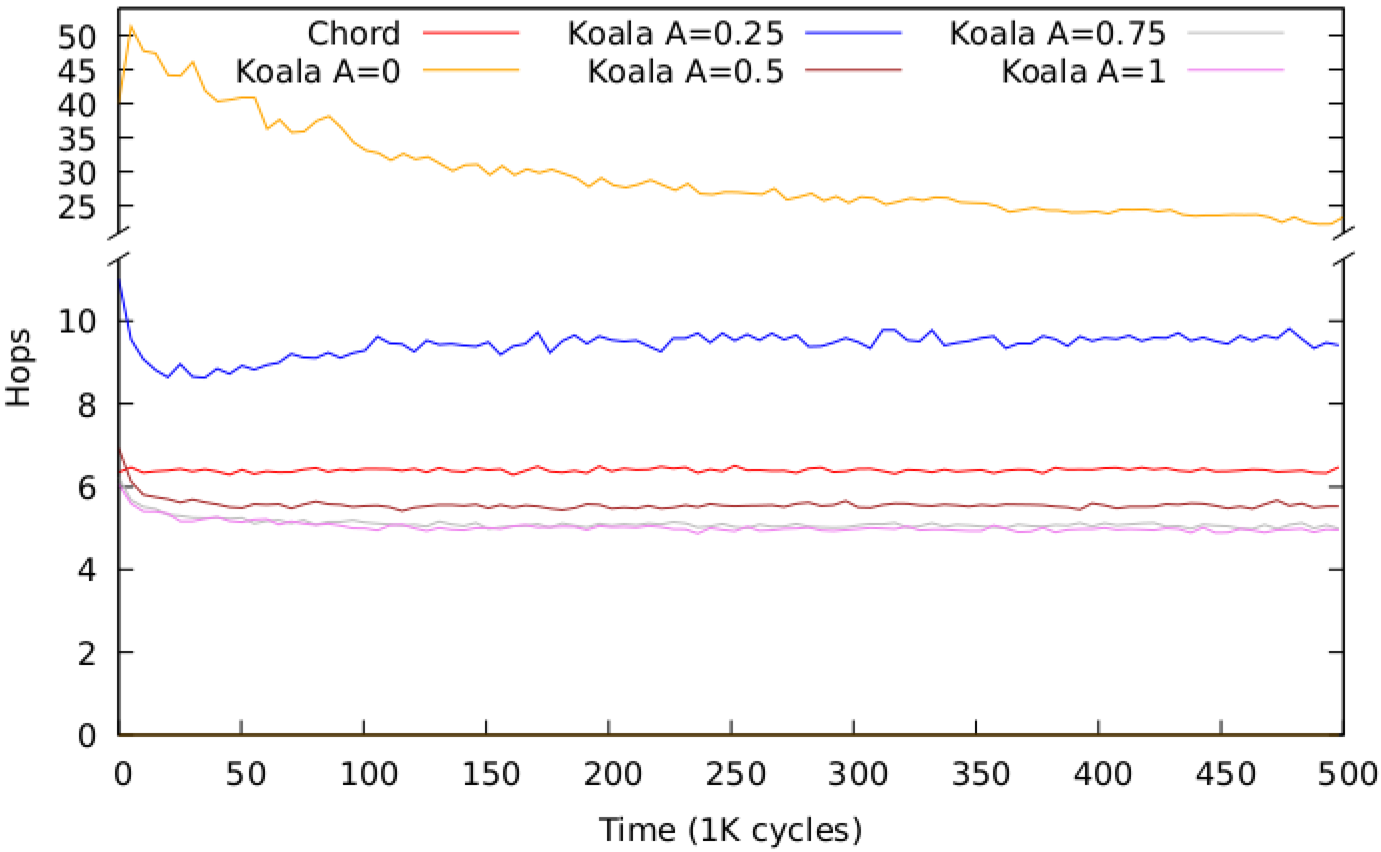

0

alpha

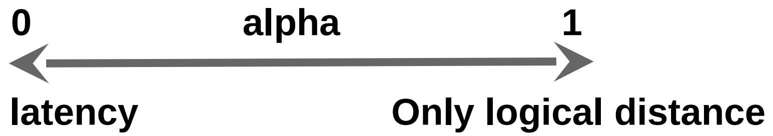
1

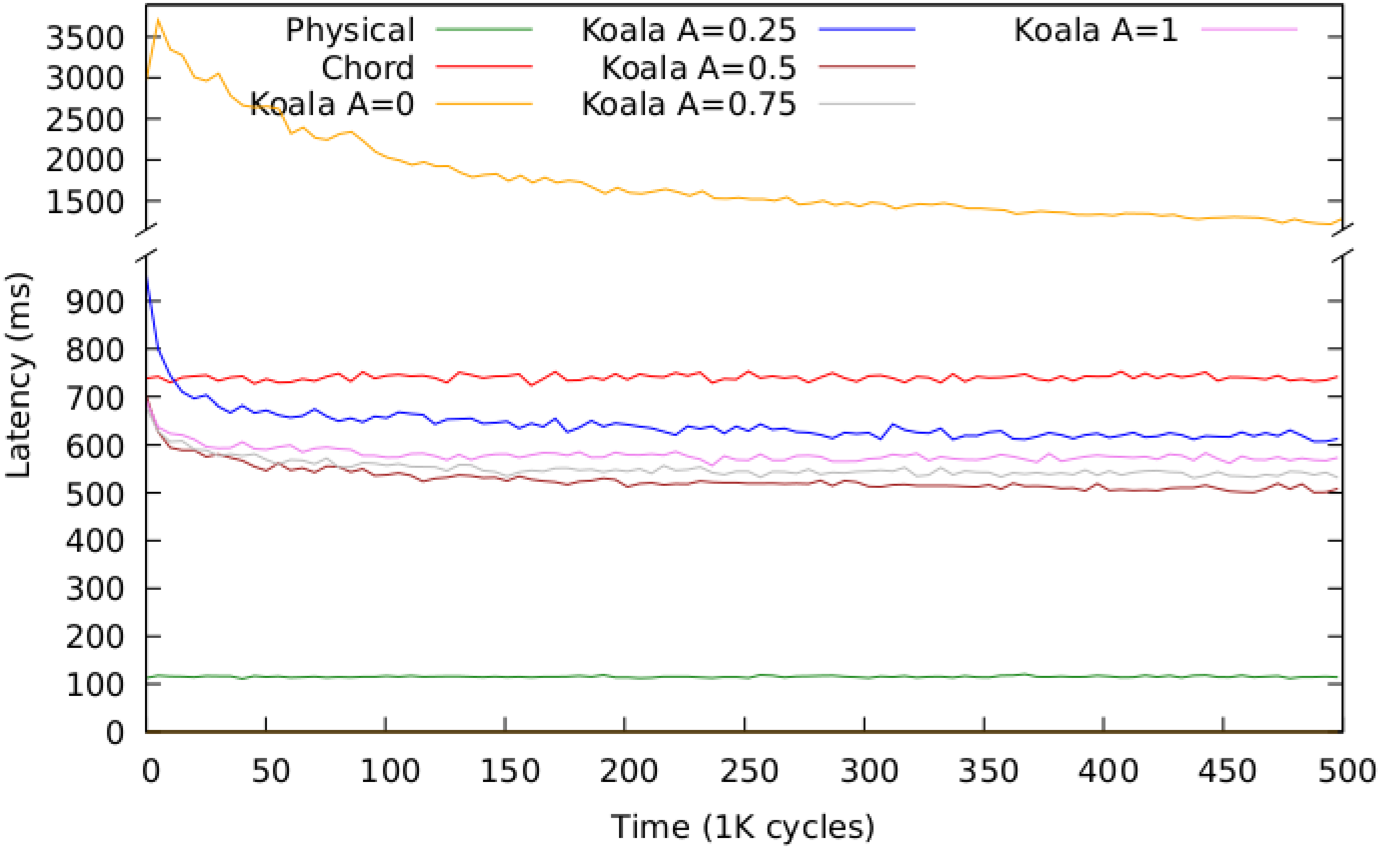
Only latency

Only logical distance



Locality-aware vs Greedy (latency)

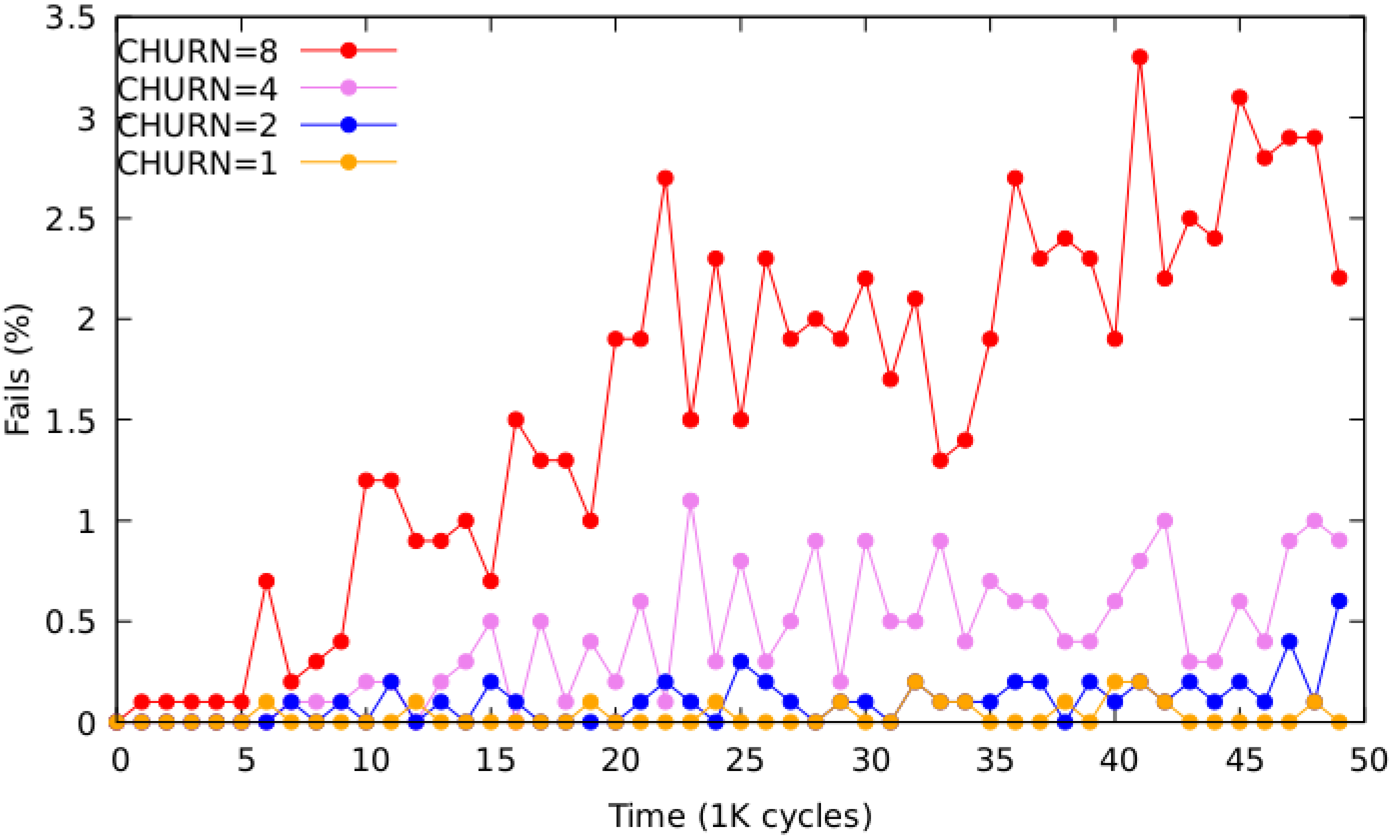




Resilience

N = 5K

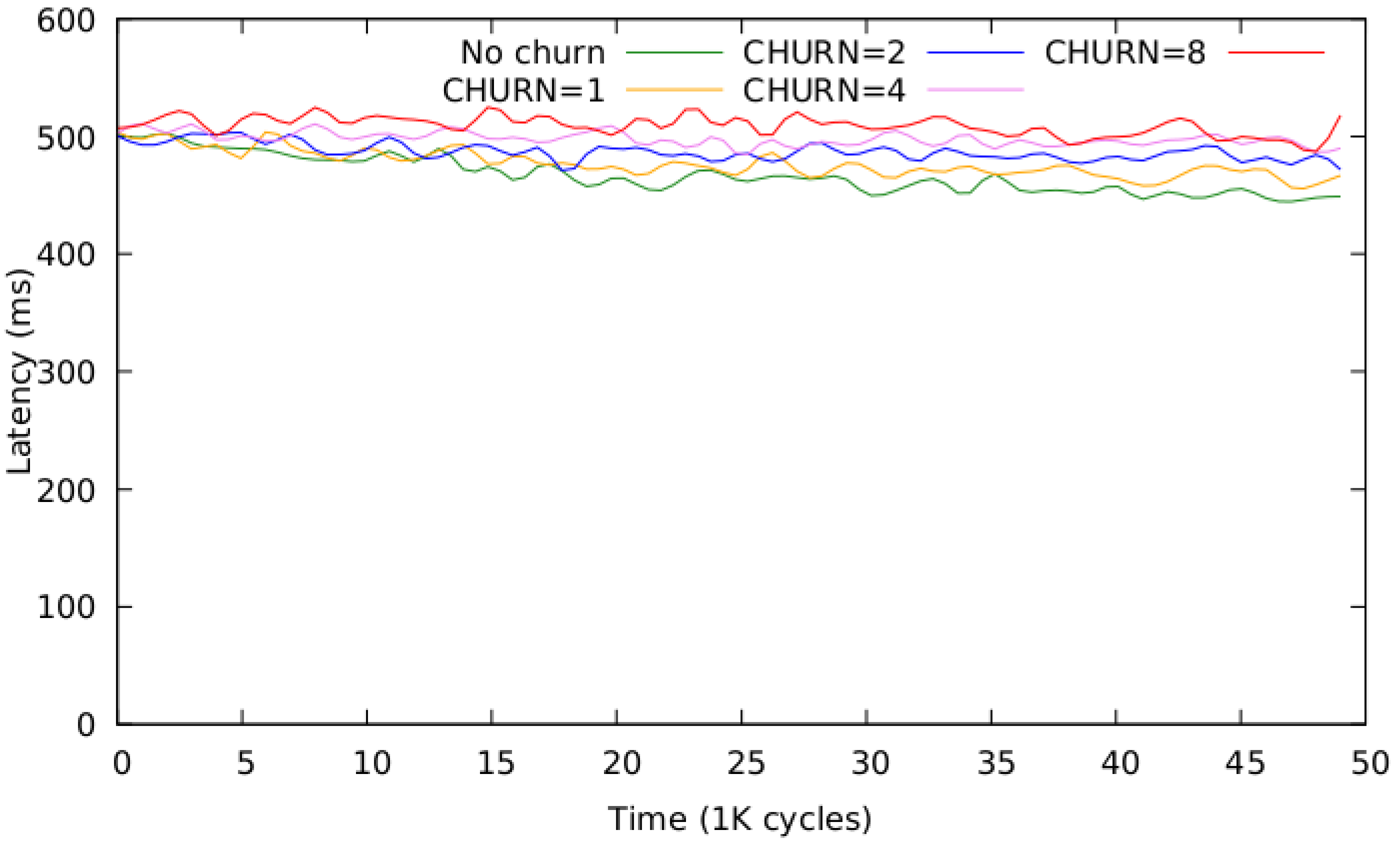
Every 80 cycles CHURN nodes leave and the same number joins



Resilience

N = 5K

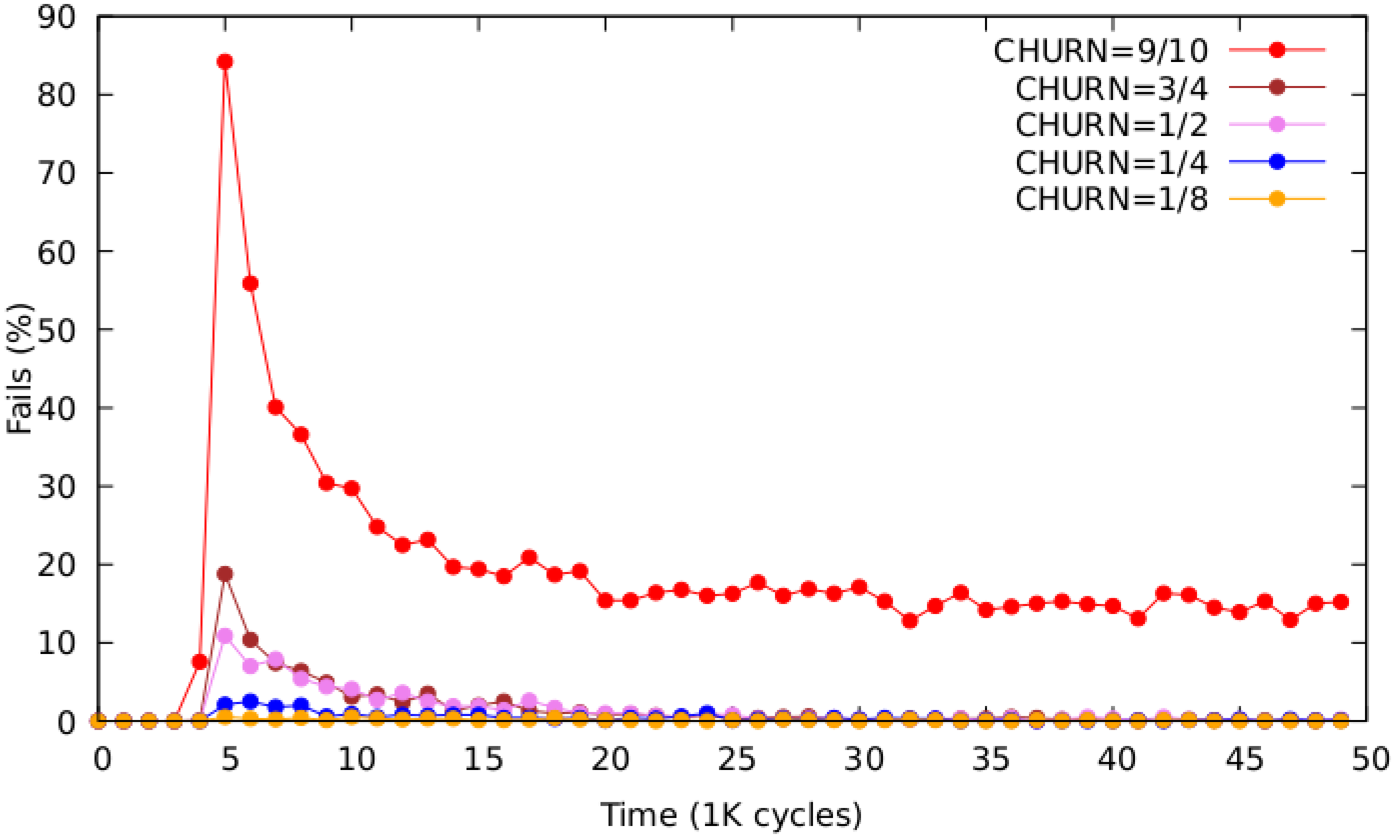
Every 80 cycles CHURN nodes leave and the same number joins



Recovery

N = 10K

At cycle 5K we introduce an unexpected failure



Results submitted to EuroPar 2017

2-level structure

...but so far we have assumed a PoP has only one server

Need to distinguish between local and global nodes.

An Overlay Network

…but how should this overlay be?

1. Lazy (update overlay only when needed)
2. Locality-aware

An Overlay Network

...but how should this overlay be?

1. Lazy (update overlay only when needed)
2. Locality-aware
3. **Consider 2-level structure**

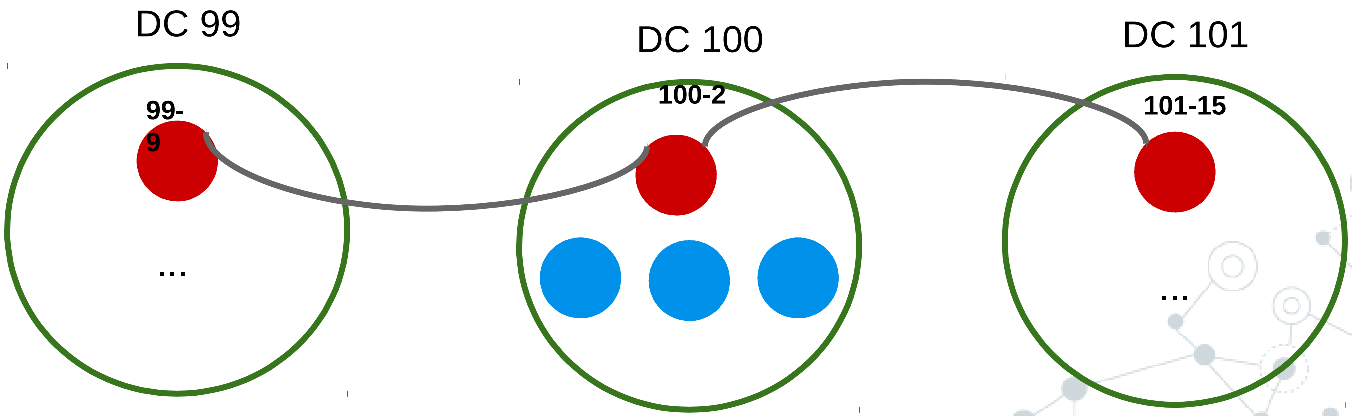
Hierarchical?

A leader in each PoP.

Responsible for global communication.

Upside: smaller global overlay

Downside: Needs replication and leader election protocols are cumbersome!



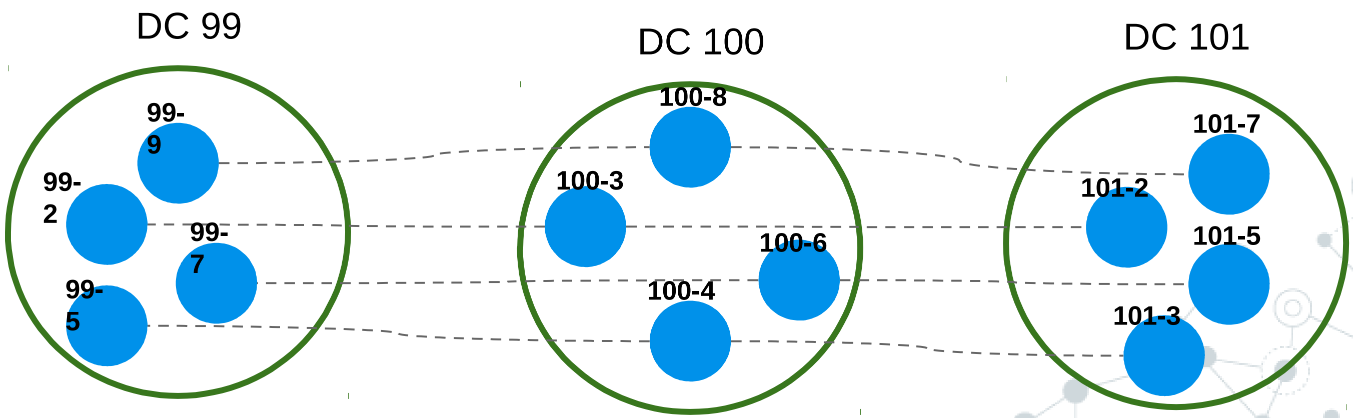
Flatten hierarchies

Nodes are still in a local group

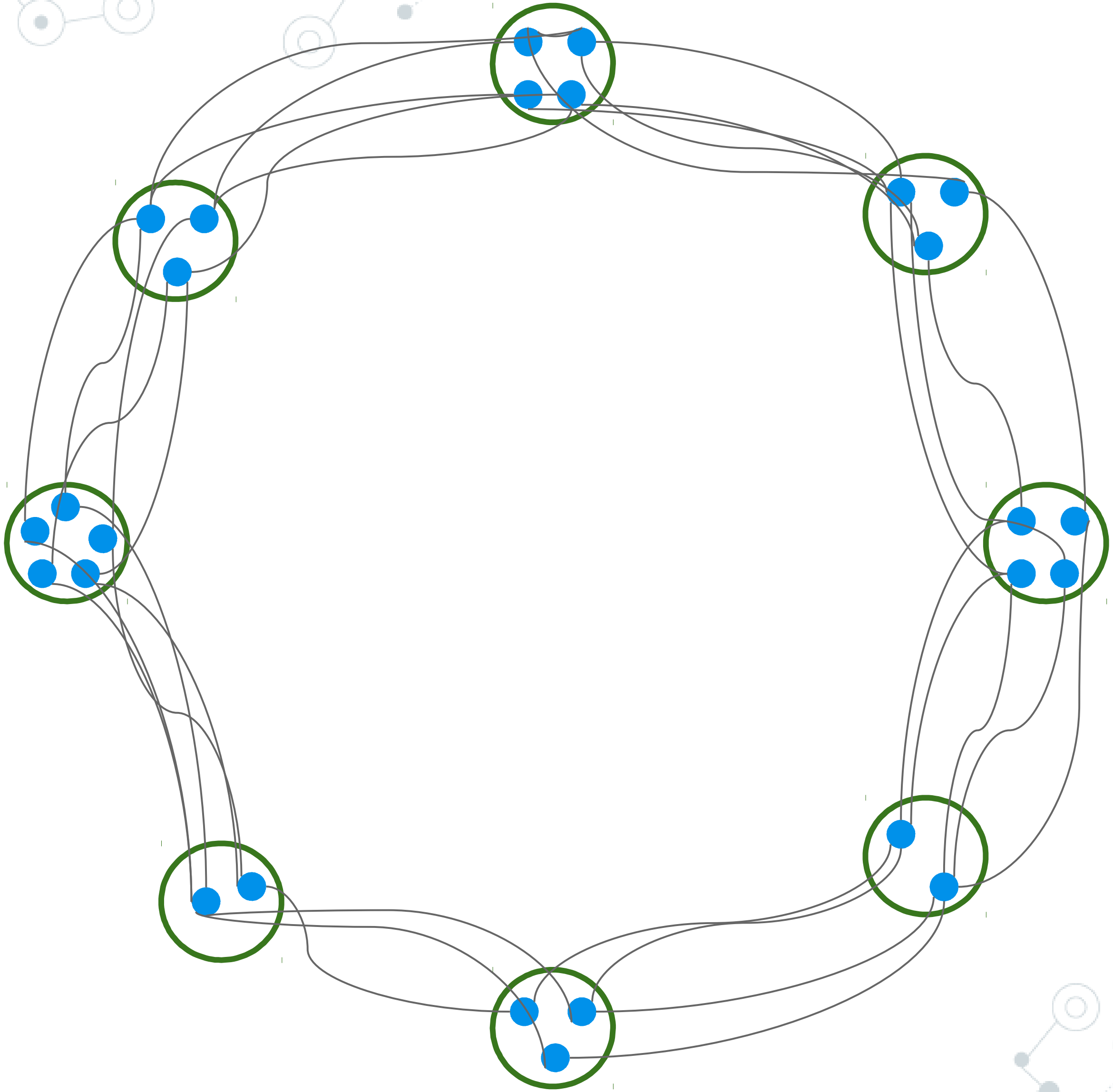
Every node has enough knowledge to route globally

Upside: Load balancing, resilience, comparable size of global system

Downside: Multiple rings



The multi-ring



Intra-PoP communication

We opt for a 0-hop DHT approach.

Since the number of servers is limited, everybody can now everybody else.

Consider ways to implement a lazy 0-hop DHT

Thank you!

Questions?