Sargun Dhillon, 2016

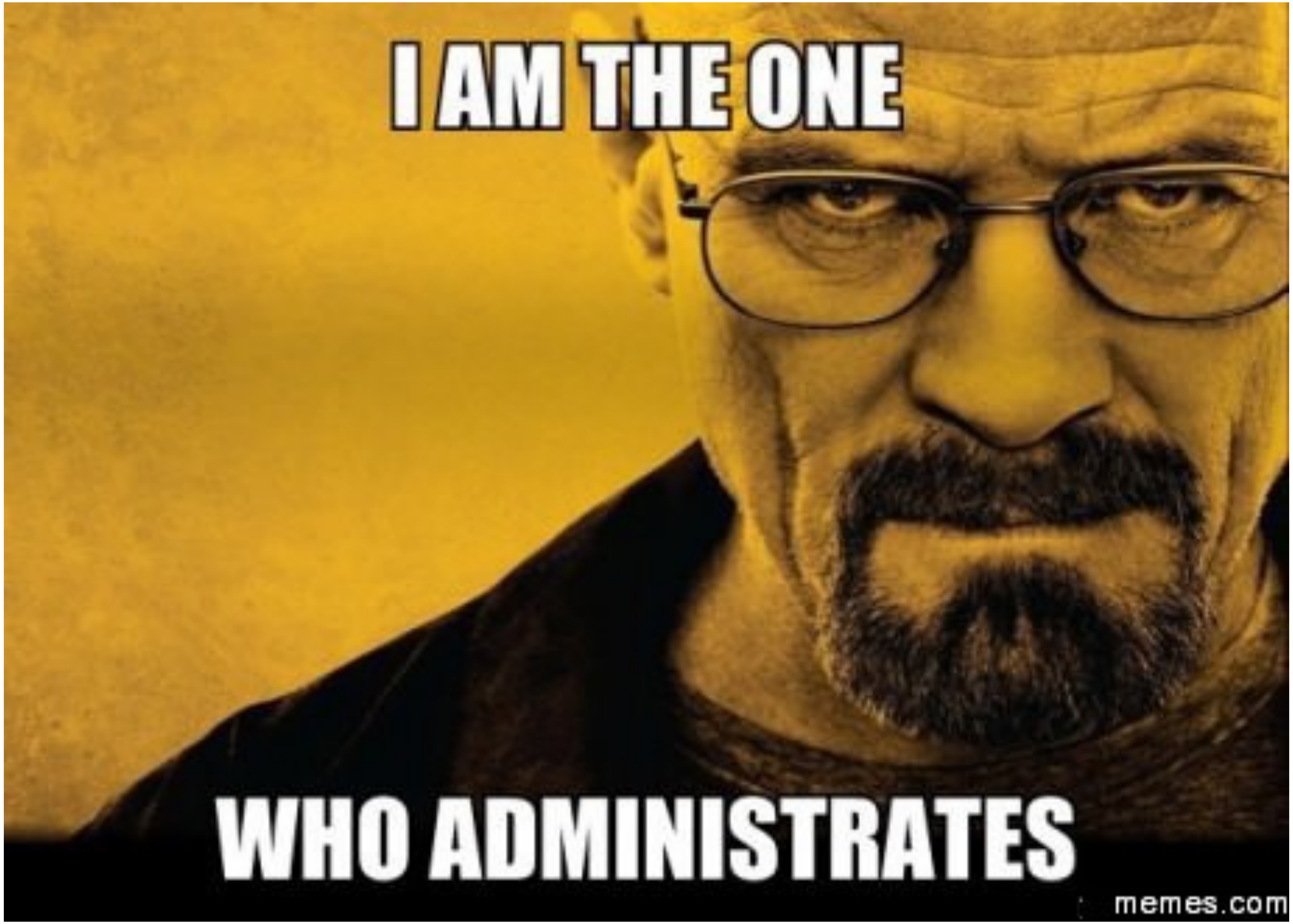
CONTAINER ORCHESTRATION AND SOFTWARE DEFINED NETWORK: A FIELD REPORT

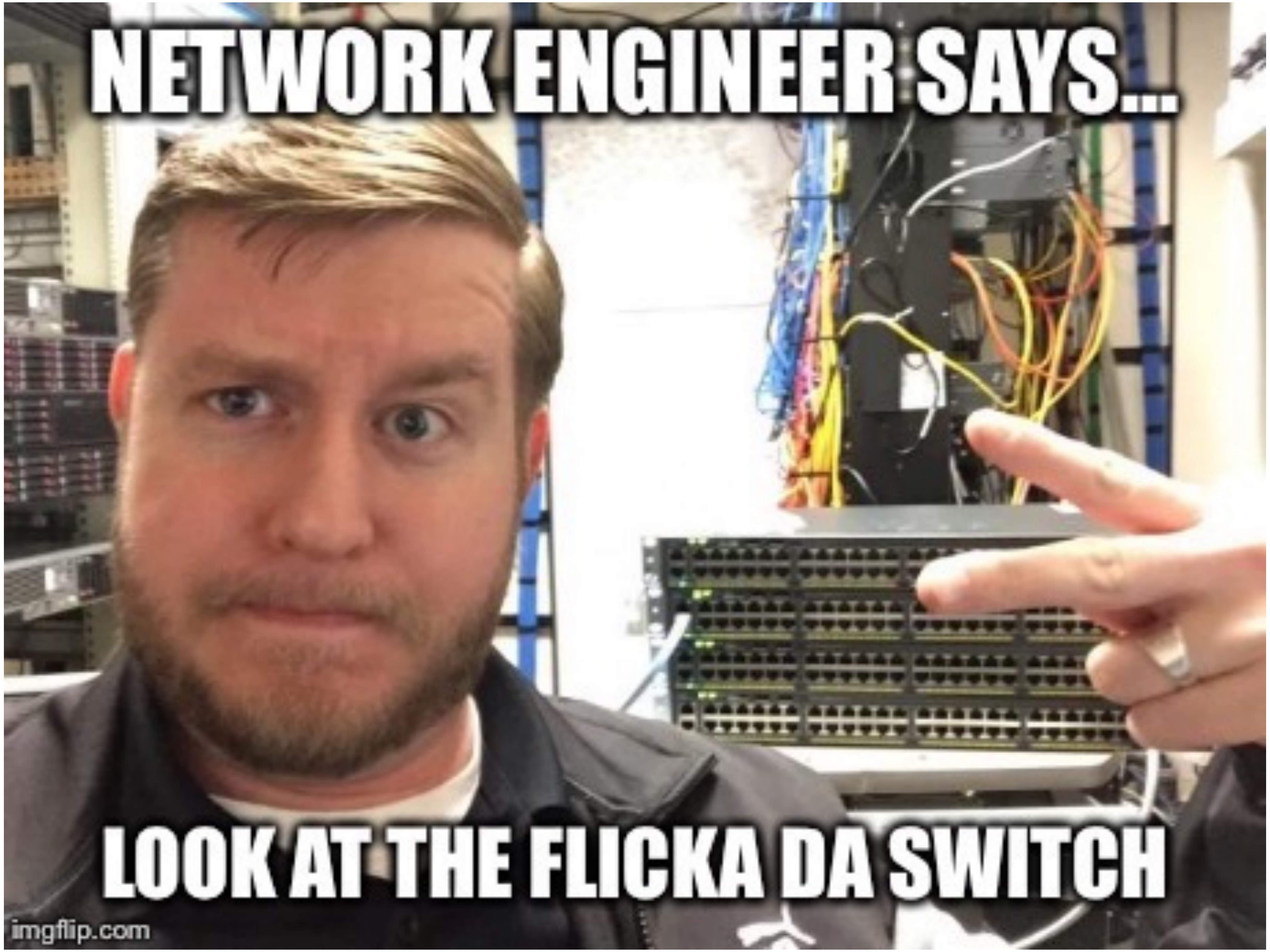
@SARGUN

MESOSPHERE

WHO AM I?

MESOSPHERE


I AM THE ONE
WHO ADMINISTRATES
memes.com


NETWORK ENGINEER SAYS...
LOOK AT THE FLICKA DA SWITCH
imgflip.com


I HAVE NO IDEA
WHAT I'M DOING
SOFTWARE ENGINEERING
quickmeme.com

My Mission

AGENDA

- A brief history of how we got here
- How we’re trying to make it better
- Some systems we’ve built to enable us to better ship
- How Erlang helped us get there

A BRIEF HISTORY OF NETWORKS IN THE DC

ORGANIZATIONS CIRCA 2007

- Before DevOps was first heard
- Clear differentiation of ownership
  - The datacenter was owned by a the NOC
  - Deployment of services was done by sysadmins in the operations group
  - Developers operated without access to production
  - Production deployments gated by QA, Operations

SOFTWARE CIRCA 2007

- Different services glued together via CORBA, XML-RPC, SOAP
  - No one was really consciously doing microservices
- Networks were static, giant layer 2 domains
  - Load Balancing provided by hardware
  - Firewall provided by hardware
- Everyone ran their own datacenter
  - EC2 in its infancy, only a year prior has the term “Cloud” began to become popular
- Systems statically partitioned

SaaS continued to grow at an incredible rate

There became a race to ship faster

We kept the software alive
By feeding it
With Sysadmins

We kept the machines alive
By feeding them
With Blood

This wasn’t working

ORGANIZATIONS 2008+

- We began seeing a gradual shift in the industry where lines between QA, Dev, and Ops were blurring
  - Devops term coined in 2008, first DevOpsDay in 2009
- Gradual adoption of the cloud, fewer organizations owning their own datacenters
  - Either networking was outsourced to the cloud, or typically remained in a small internal organization
- Needed to reduce ratio of operators to servers

SOFTWARE CIRCA 2008+

- Popularization of Open Source tooling to automate much of traditional operations, and QA
  - Jenkins / Hudson
  - Puppet / Chef
  - Capistrano
- Popularization of stacks requiring with more complex operational requirements
  - Nutch / Hadoop
  - NoSQLs
- Still statically partitioned machines
- Networks still sacred territory

CIRCA 2011

- Much of what's been happening for the past half-decade hits networking
  - Much of this falls under the term "SDN" (Software Defined Networking) or "NFV" (Network Function Virtualization)
  - Hastened by the adoption of VMs in the enterprise in the hype cycle
  - Openflow promises to fix everything
- Major adoptions of the cloud by startups as well as enterprise
- Virtualization begins to become mainstream as a mechanism of consolidating workloads
  - The invention of the "private cloud"
- DotCloud / Docker funded by Y-combinator a year earlier
- Term "Microservice" coined

CIRCA 2013

- Docker becomes instant hit and brings containers to the forefront
- Dynamic partitioning begins to make in-roads
  - Google releases Omega paper
  - Apache Aurora open sourced
  - Microservice counts explode, demanding collocation of workloads for efficiency
  - Mesosphere Founded
- Site Reliability Engineering begins to popularize and further blur the lines between Dev, Ops, and QA

Everything was changing

Why?

Business Value

100
FEDERAL RESERVE NOTE
BI 01207220 A
I9
UNITED STATES
FEDERAL RESERVE SYSTEM
THIS NOTE IS LEGAL TENDER
FOR ALL DEBTS, PUBLIC AND PRIVATE
SERIES 1999
Treasurer of the United States.
E2
100
FRANKLIN
THE UNITED STATES OF AMERICA
100
BI 01207220 A
Stan Dalone
Secretary of the Treasury.
E9
100
ONE HUNDRED DOLLARS

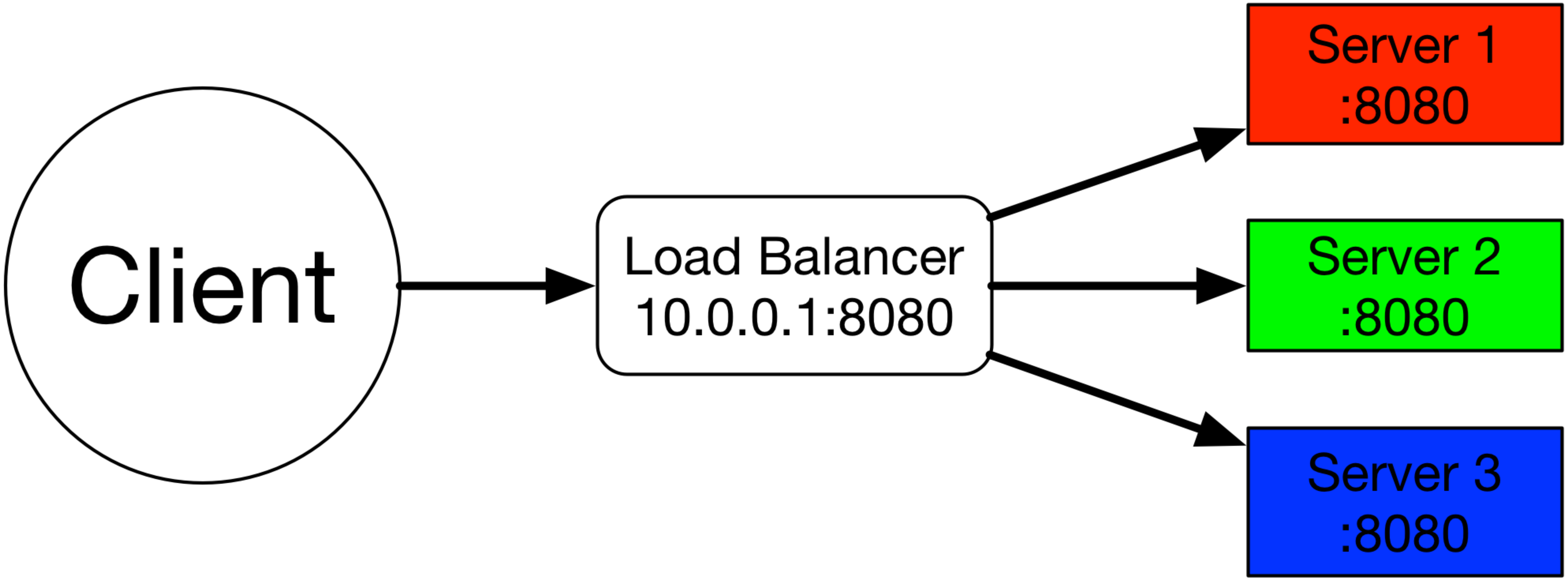
BENEFITS

- Reduction in cost of goods sold
  - Smaller engineer to server ratio
    - Linear, or super linear growth rate of engineering team to servers is unsustainable
  - Smaller engineer to capability ratio, where capability includes:
    - Features
    - Throughput
- Better User Experience
  - Better availability
  - Quicker release to features

But at what cost?

Complexity

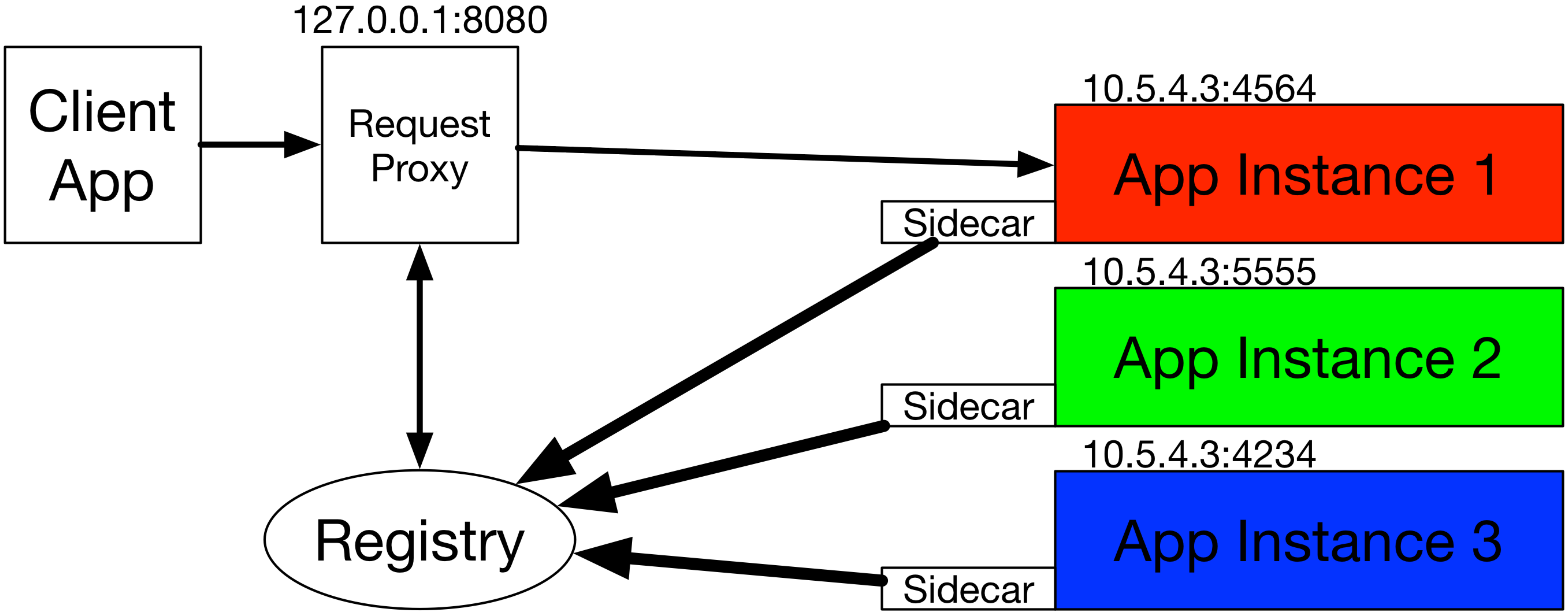
OLD WORLD



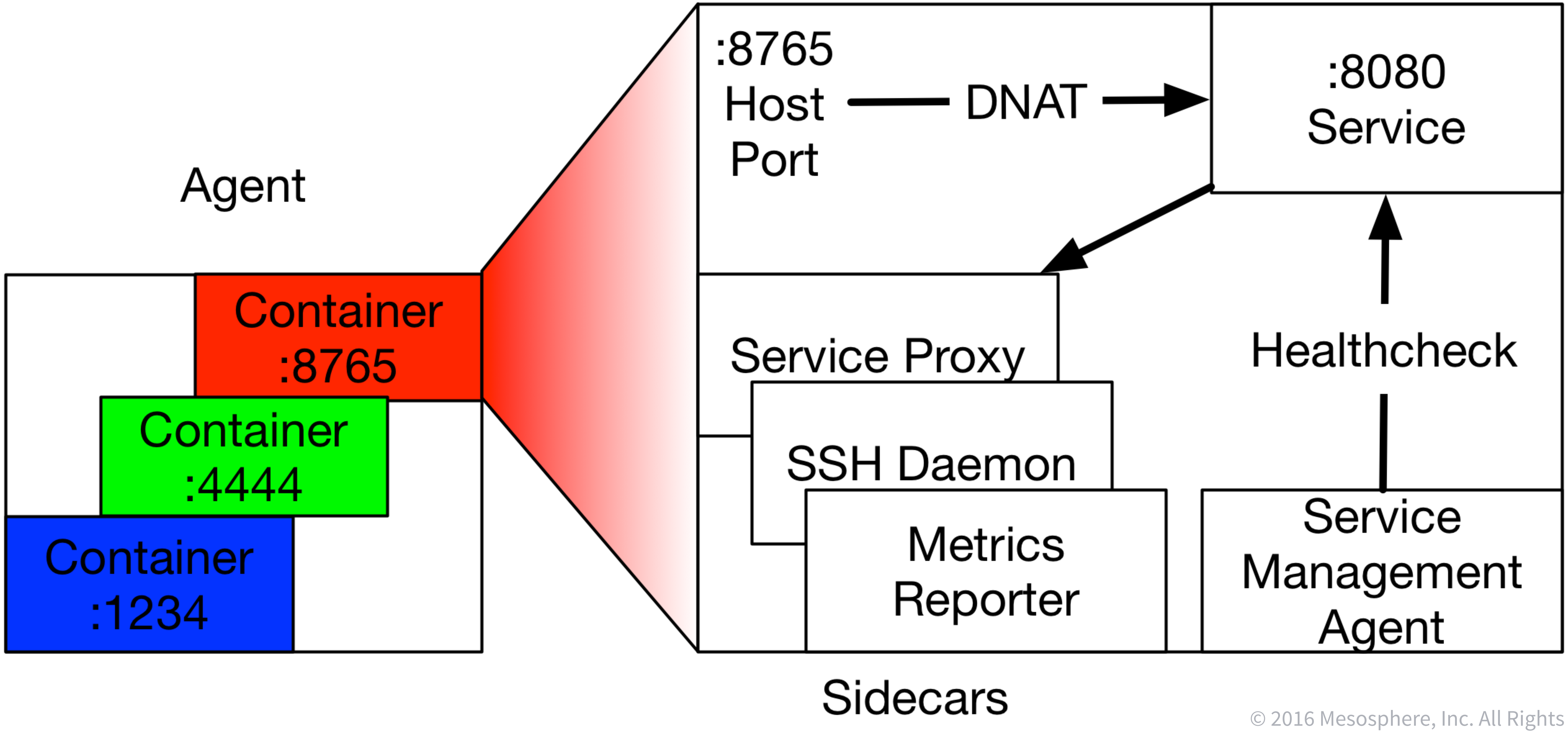
Client
Load Balancer
10.0.0.1:8080
Server 1
:8080
Server 2
:8080
Server 3
:8080

NEW WORLD

"Simple" Service Discovery



DIVING DEEPER



Etcd?

Pods?

Paxos?

VxLan?

Wat?

Ω Failure Detector?

Raft?

Sidecars?

Zookeeper?

It was time for something completely new

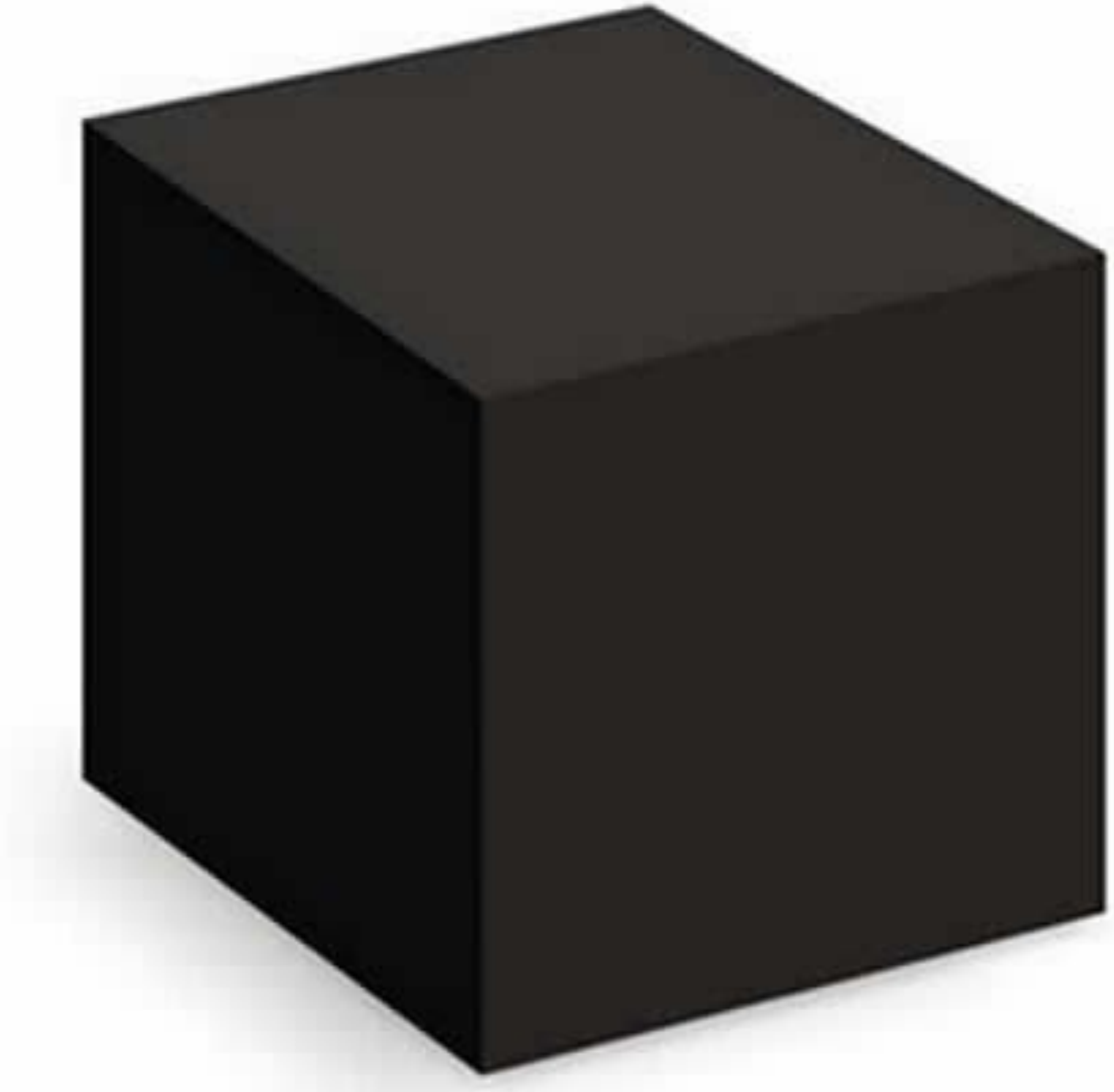
ENTER DC/OS

CONTAINERS WITH BATTERIES INCLUDED

- Includes everything you need to move away from 2007
- Networking
- Security
- Application deployment, and orchestration
- Stateful services
- Scheduling

In Your Datacenter

As a Black Box

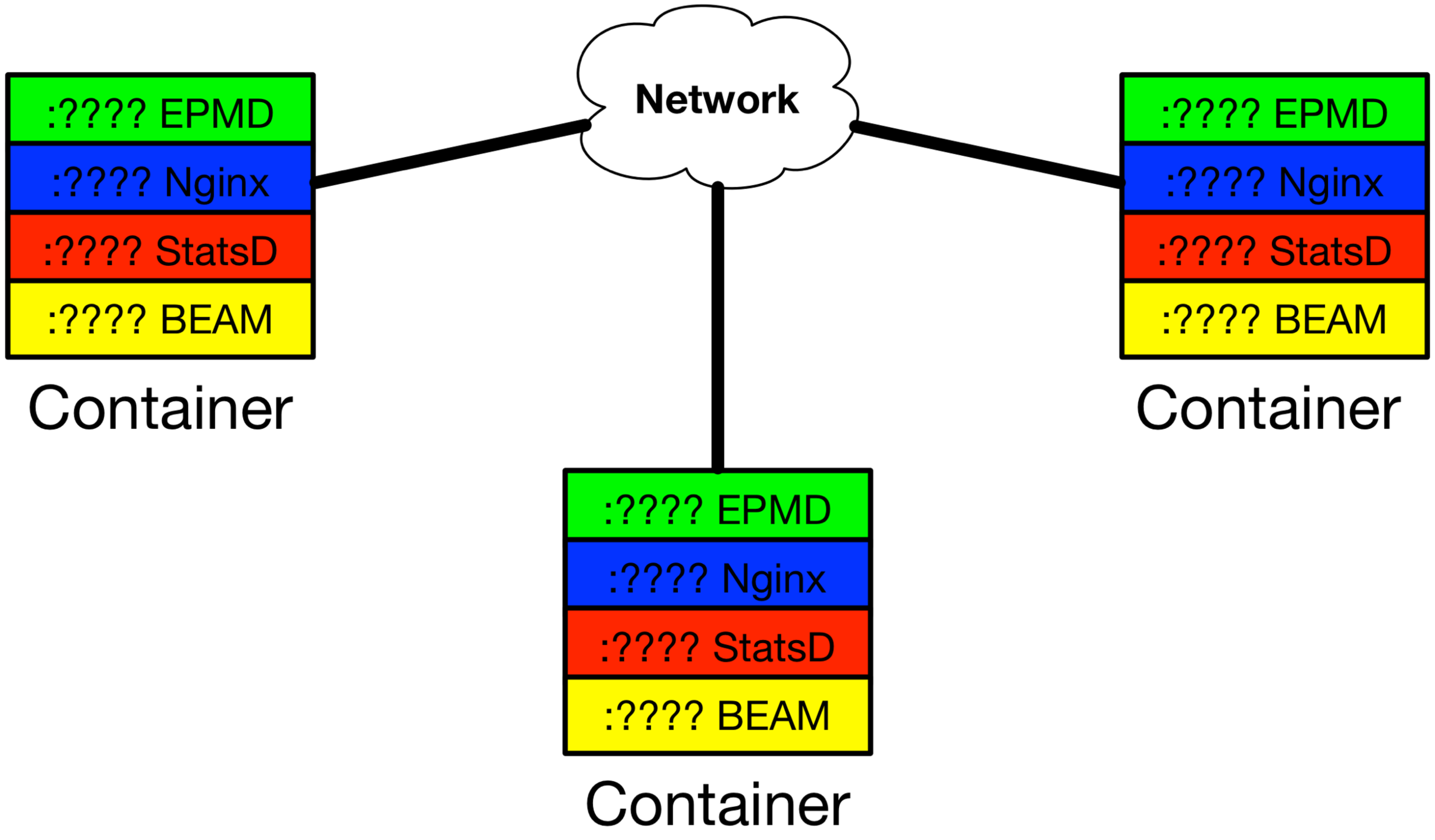


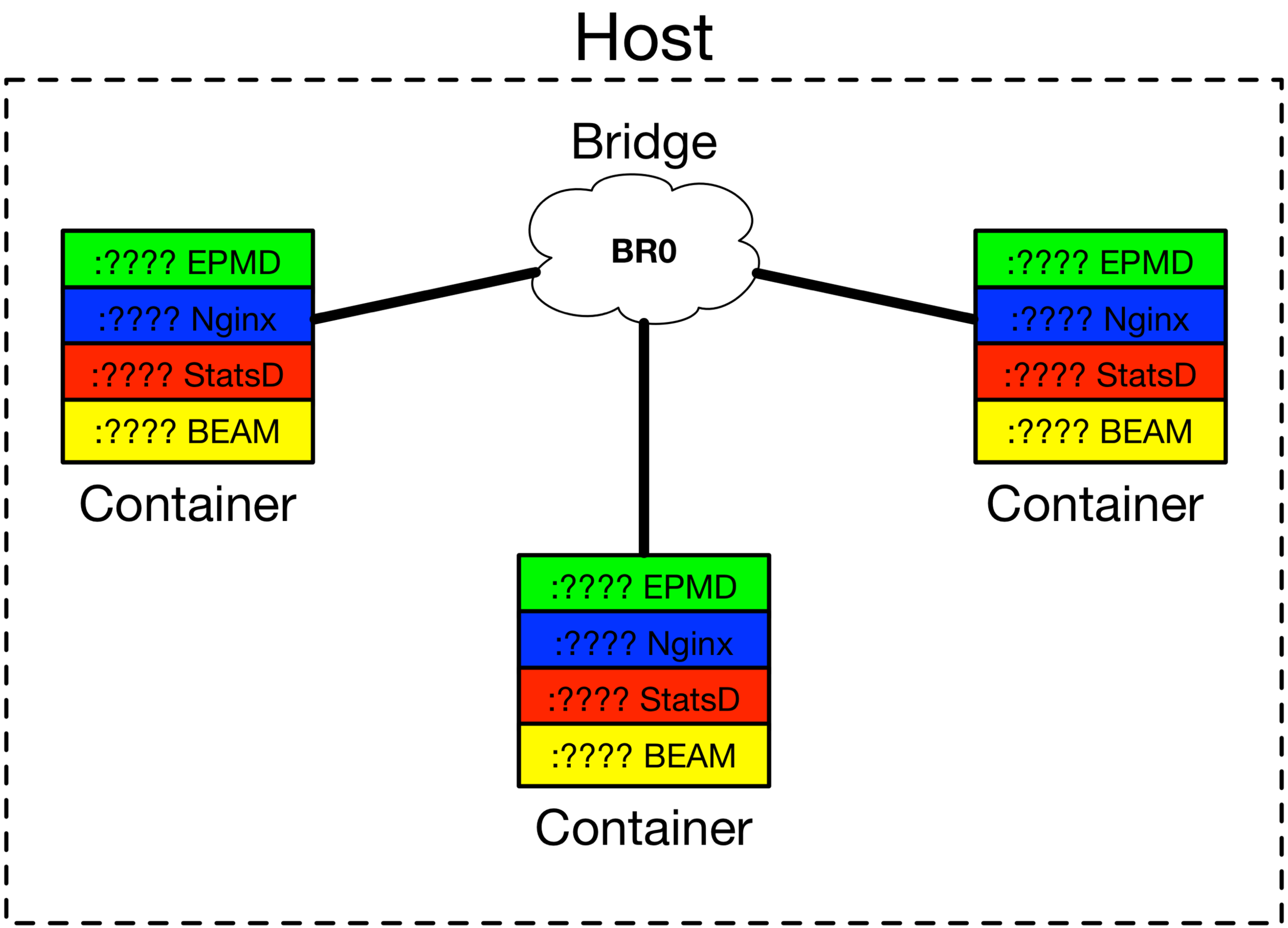
But, our story starts:
~November 2015
(~8 months ago)

Networking Features

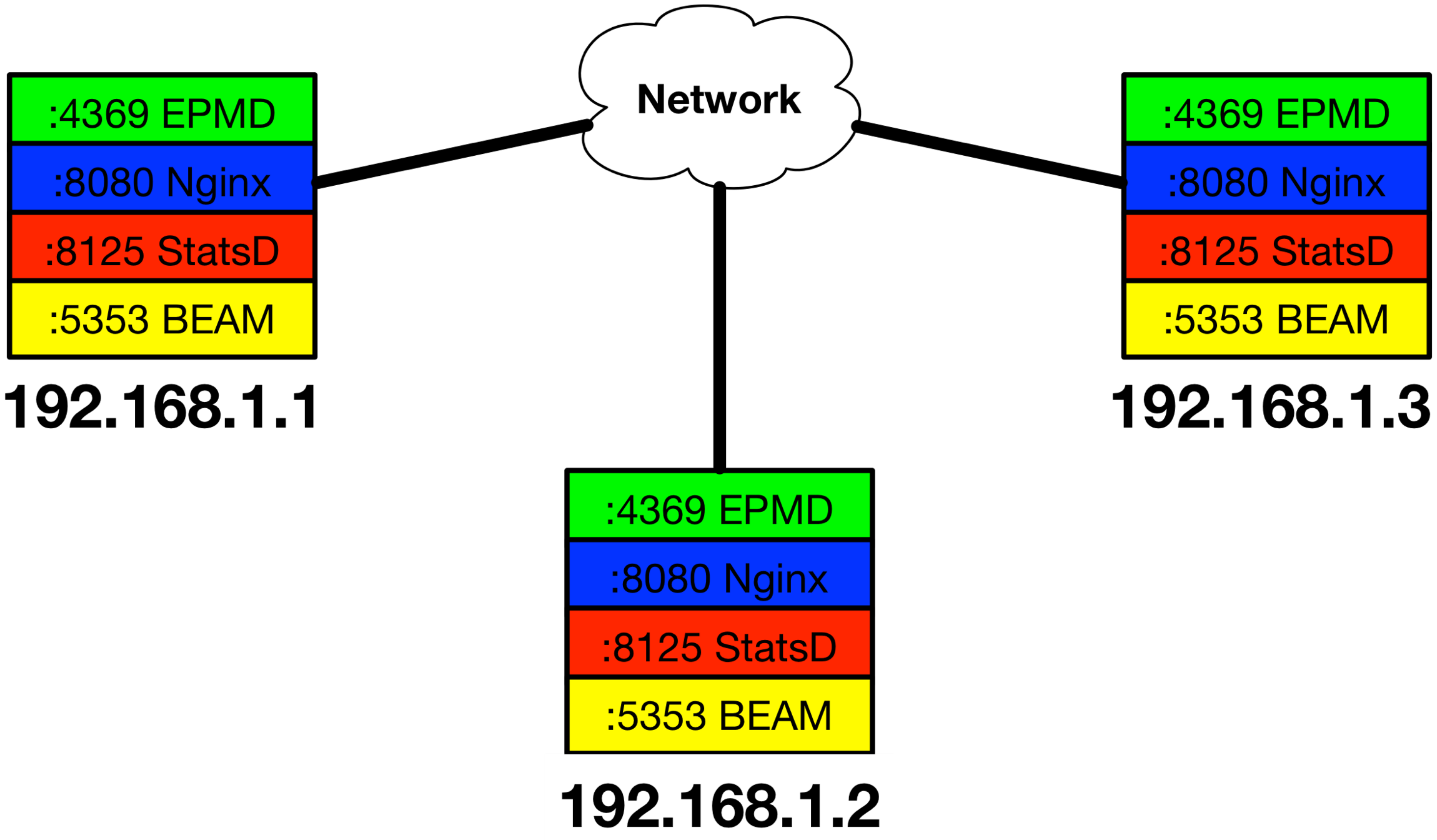
“Service Discovery”

Where are my apps running?




Host
Bridge
BR0
:???? EPMD
:???? Nginx
:???? StatsD
:???? BEAM
Container
:???? EPMD
:???? Nginx
:???? StatsD
:???? BEAM
Container
:???? EPMD
:???? Nginx
:???? StatsD
:???? BEAM
Container

The Old World



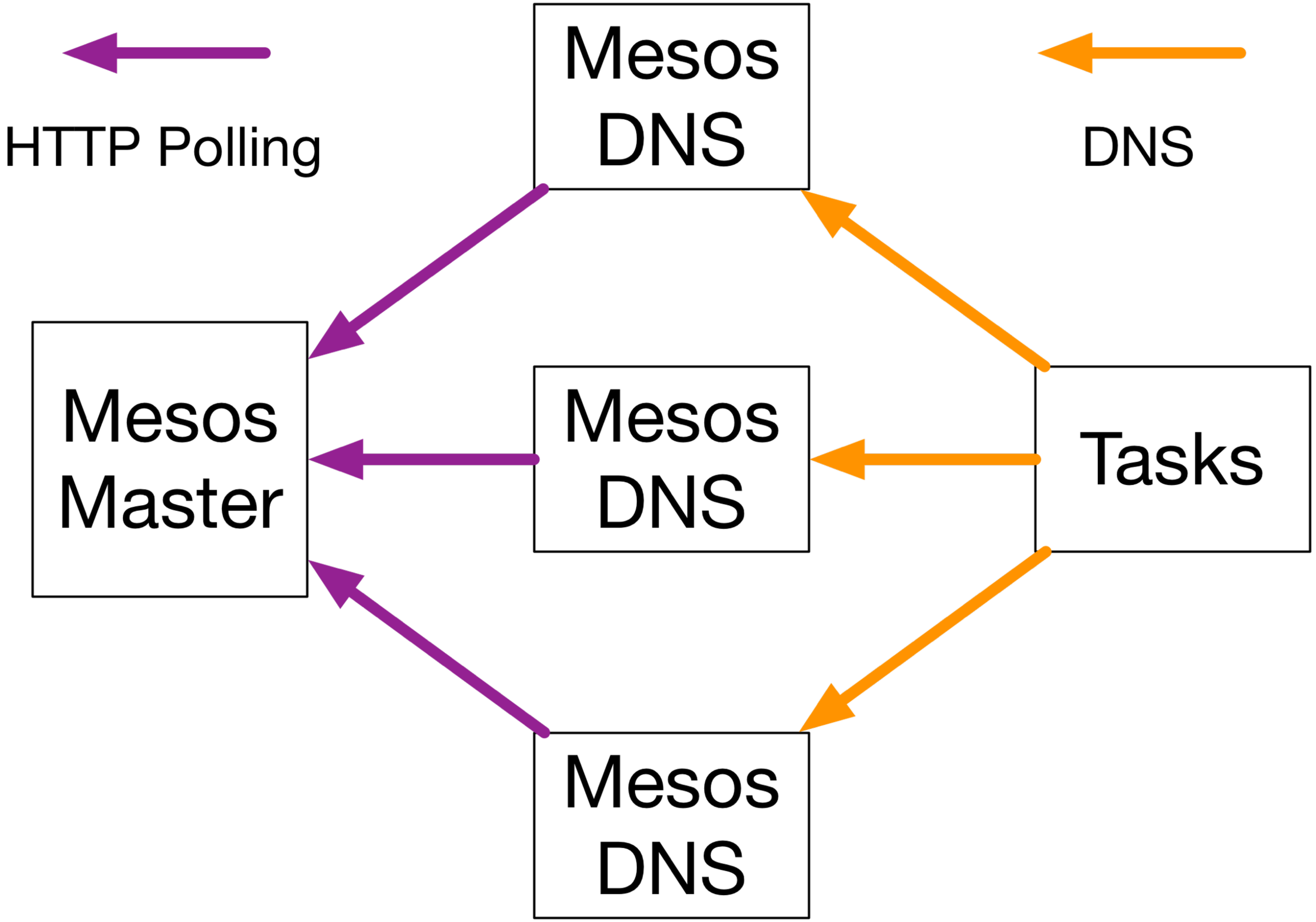
Let Mesos* Choose Ports

*The Scheduler

How do you find the tasks?

A Directory?

Mesos-DNS


HTTP Polling
Mesos DNS
DNS
Mesos Master
Mesos DNS
Tasks
Mesos DNS

…Ish

DNS SRV Based Service Discovery

Everyone has DNS right?

And GLibc even has a bug open for it!

Bug 2099 - Support for SRV records in getaddrinfo

...Opened in 2005

We needed a solution that actually worked

So, we performed an OODA loop

1. Observe
2. Orient
3. Decide
4. Act

Gathering our data about the field

OBSERVE: SERVICE DISCOVERY

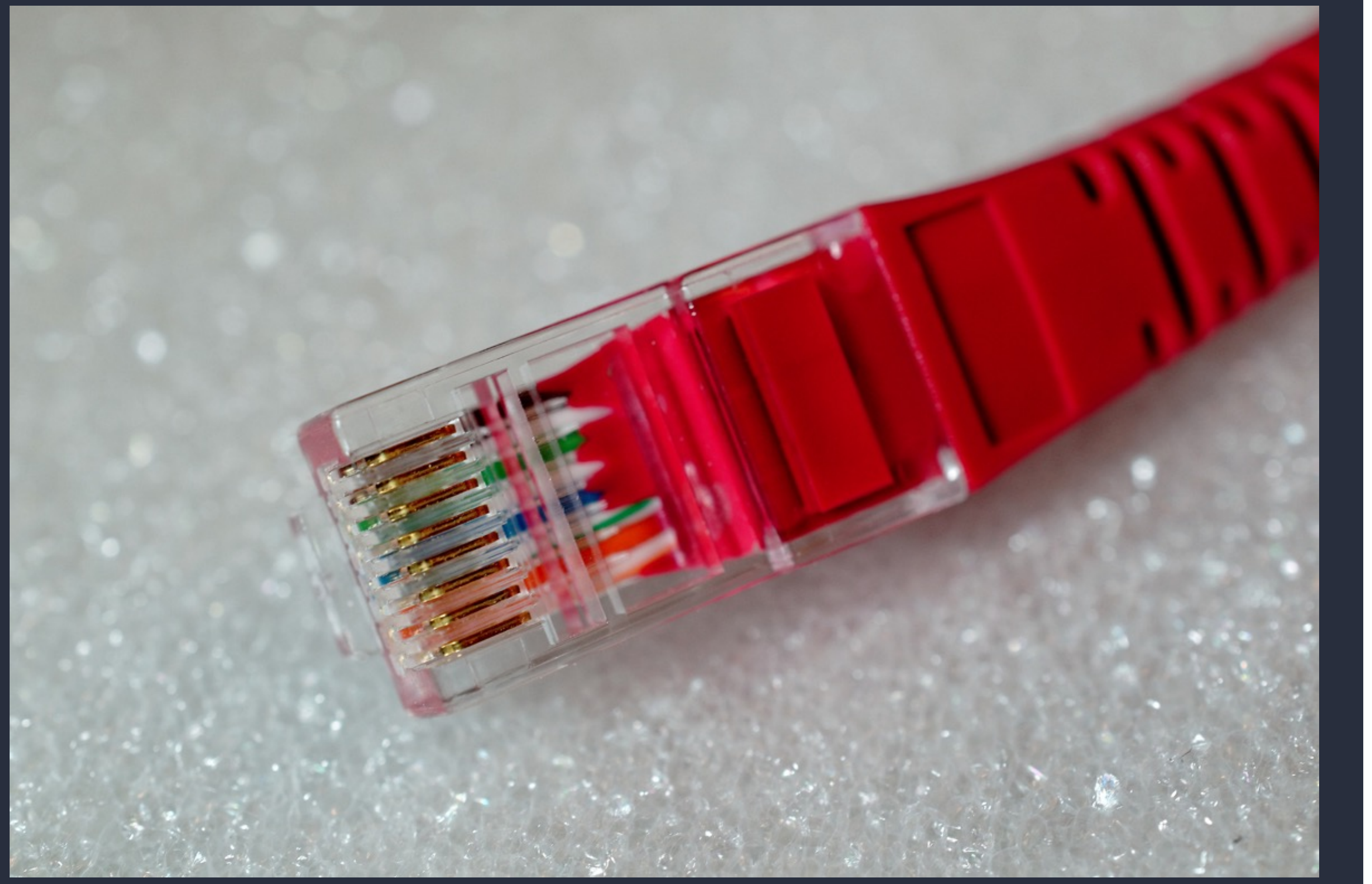
- Existing Dynamic Service Discovery Solutions:
    - Etcd
    - Finagle + Zookeeper
    - Consul
- Existing Static Service Discovery Solutions:
    - Amazon ELB
    - Hardware Load Balancers
- Service Discovery is an afterthought

Gathering our data about the field

OBSERVE: NETWORKING

- Everybody assumes IP per application instance
- Everybody assumes reliable DNS
- Some people want to be fast
- Some people want security
- Nobody wants to edit application code
- Nobody wants to talk to their network engineer

ORIENT: DC/OS NETWORKING CORE TENETS



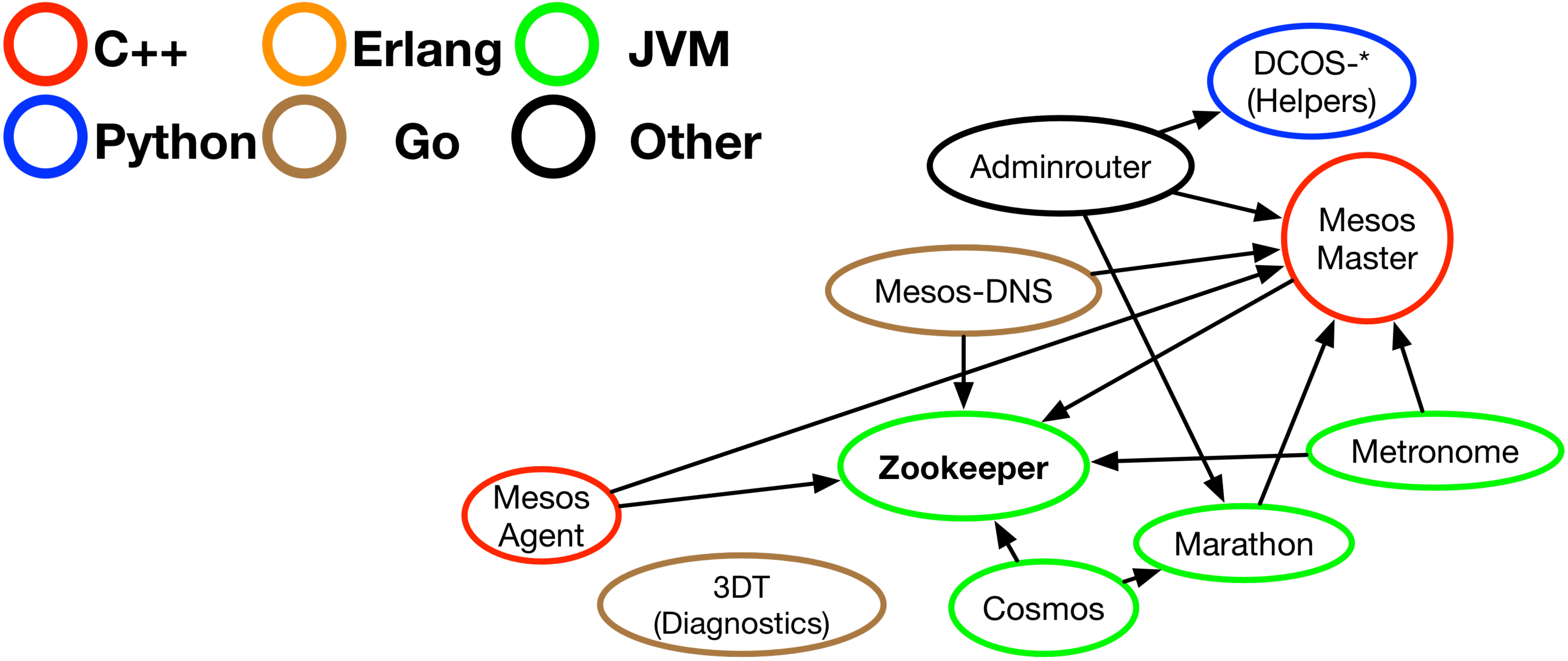
- DC/OS must be agnostic to the underlying environment
  - AWS / Azure / GCE / Softlayer as the lowest common denominators
- DC/OS should require no to minimal changes to the code in order to work
  - DC/OS should provide similar services to existing environments
    - Fixed load balancers
    - Security
    - IP/Container
- We do not want to require a change in organization procedures
- We want to be secure

What do we want to build?

DECIDE

- Load Balancing
- Seamless Service Discovery
- Reliable DNS
- External cluster Access
- Metrics and Discoverability
- IP Per Container

DC/OS ARCHITECTURE (~NOV '15): POLYGLOT MICROSERVICES

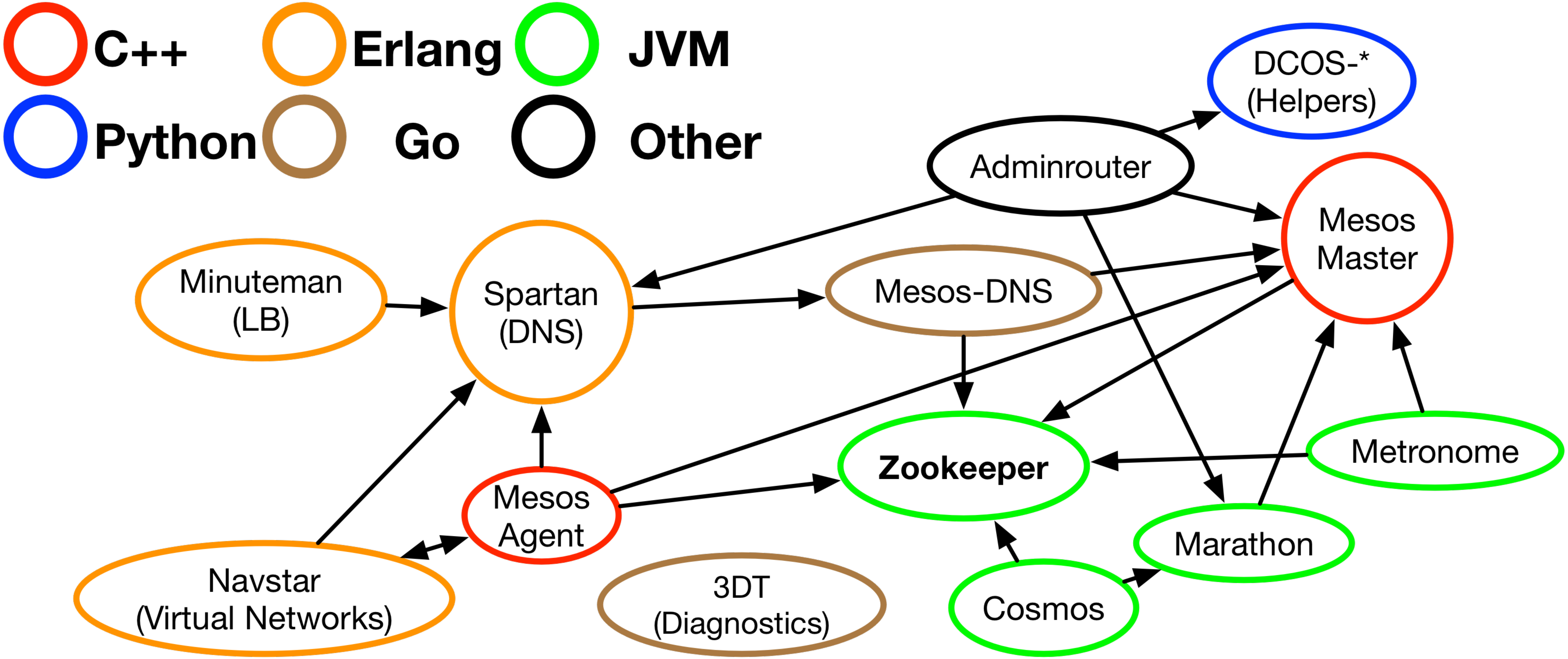


What Did We Build?

ACT: DEVELOPED NEW ERLANG SERVICES

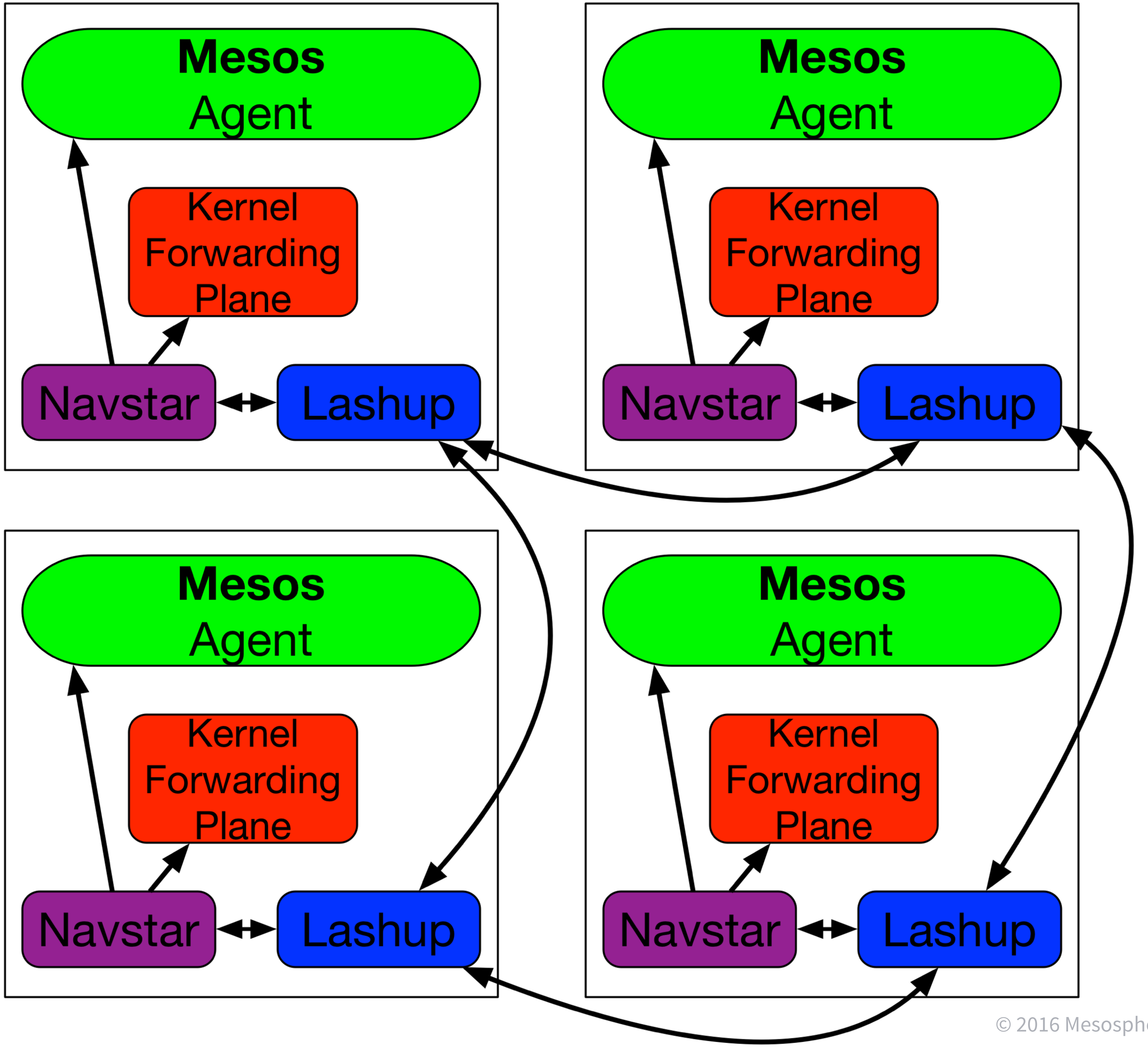
- Service Discovery
  - Navstar
  - Spartan
- Load Balancing
  - Minuteman
  - Networking API
  - Fishladder
- Control Plane
  - Lashup

DC/OS ARCHITECTURE (TODAY): POLYGLOT MICROSERVICES



Network Control Plane:

Navstar

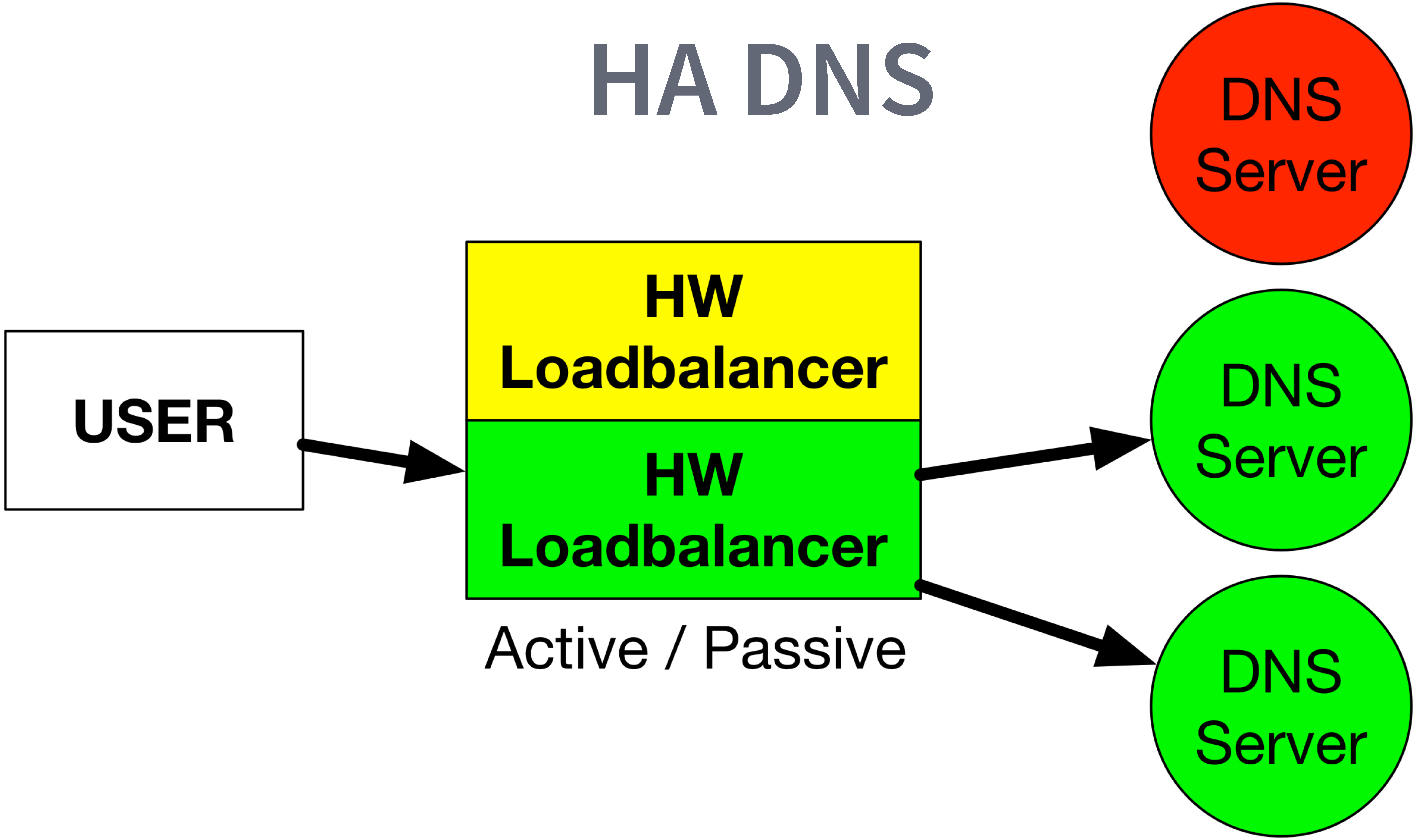

Mesos
Agent
Kernel
Forwarding
Plane
Navstar
Lashup
Mesos
Agent
Kernel
Forwarding
Plane
Navstar
Lashup
Mesos
Agent
Kernel
Forwarding
Plane
Navstar
Lashup
Mesos
Agent
Kernel
Forwarding
Plane
Navstar
Lashup

NAVSTAR: BENEFITS

- Fully decentralized control plane
- Scalable
- Extensible system to act as a building block
  - Began as network control plane

DNS:
Spartan

HA DNS



DNS Server
USER
HW Loadbalancer
HW Loadbalancer
Active / Passive
DNS Server
DNS Server

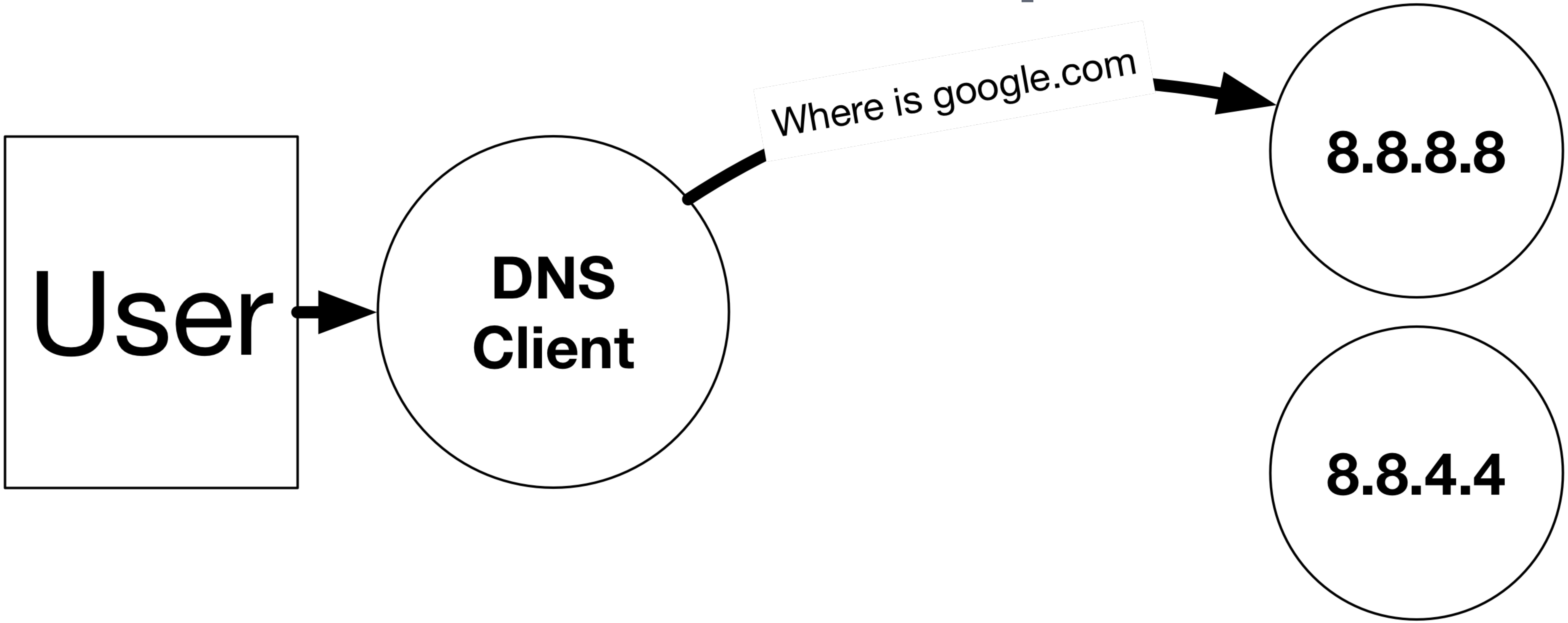
Normal DNS

/etc/resolv.conf

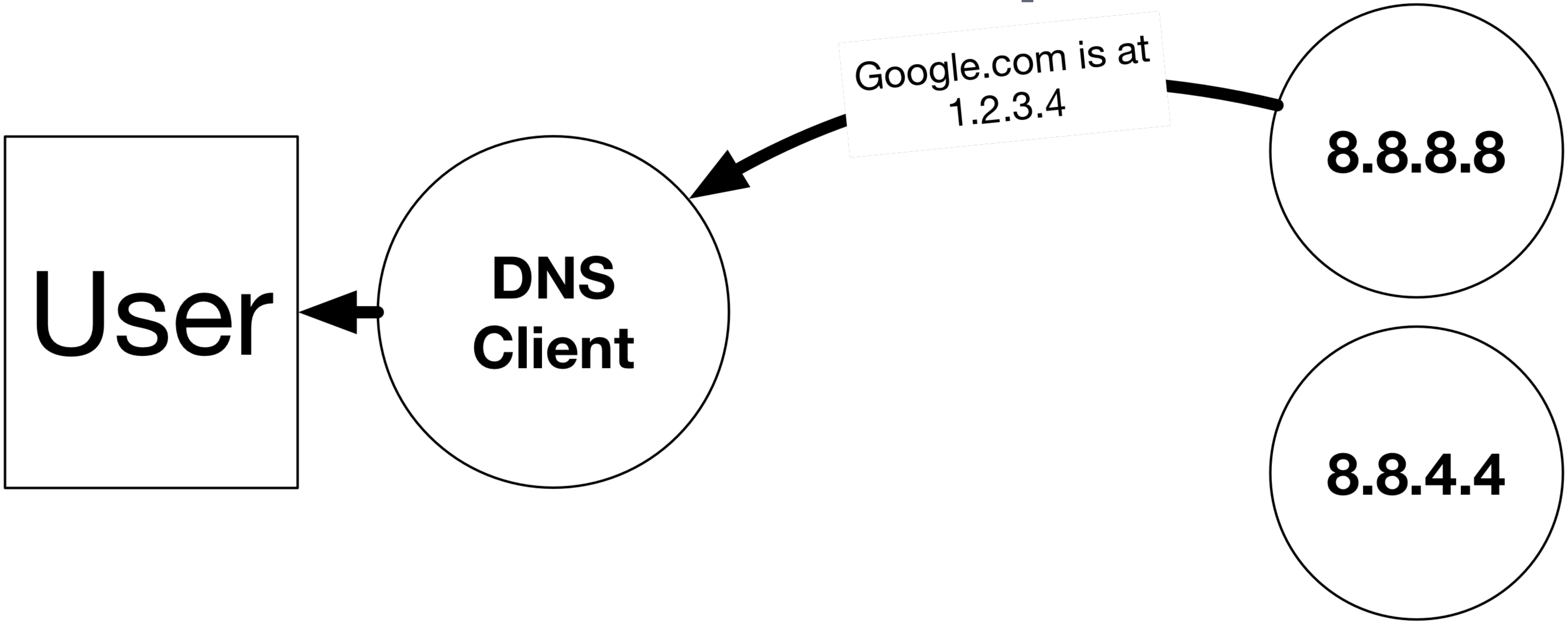
nameserver 8.8.8.8

nameserver 8.8.4.4

Ideal DNS Request

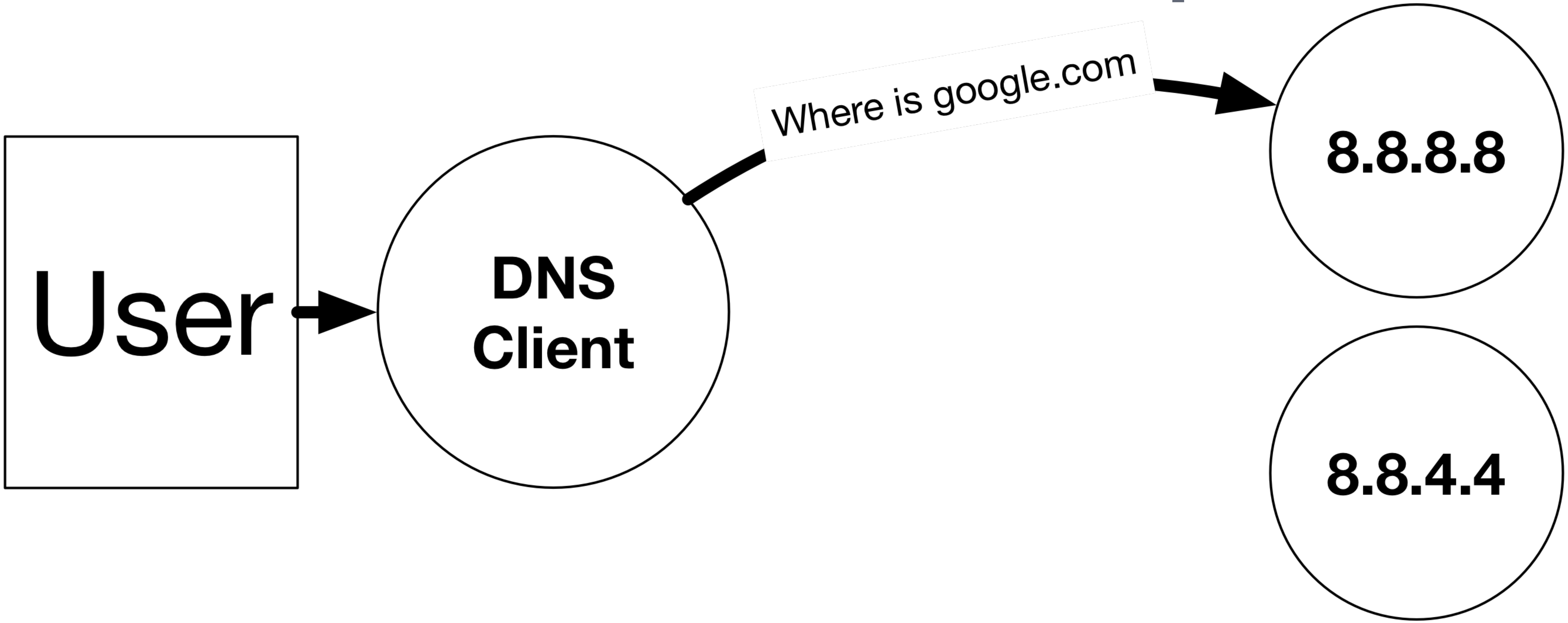


Ideal DNS Request



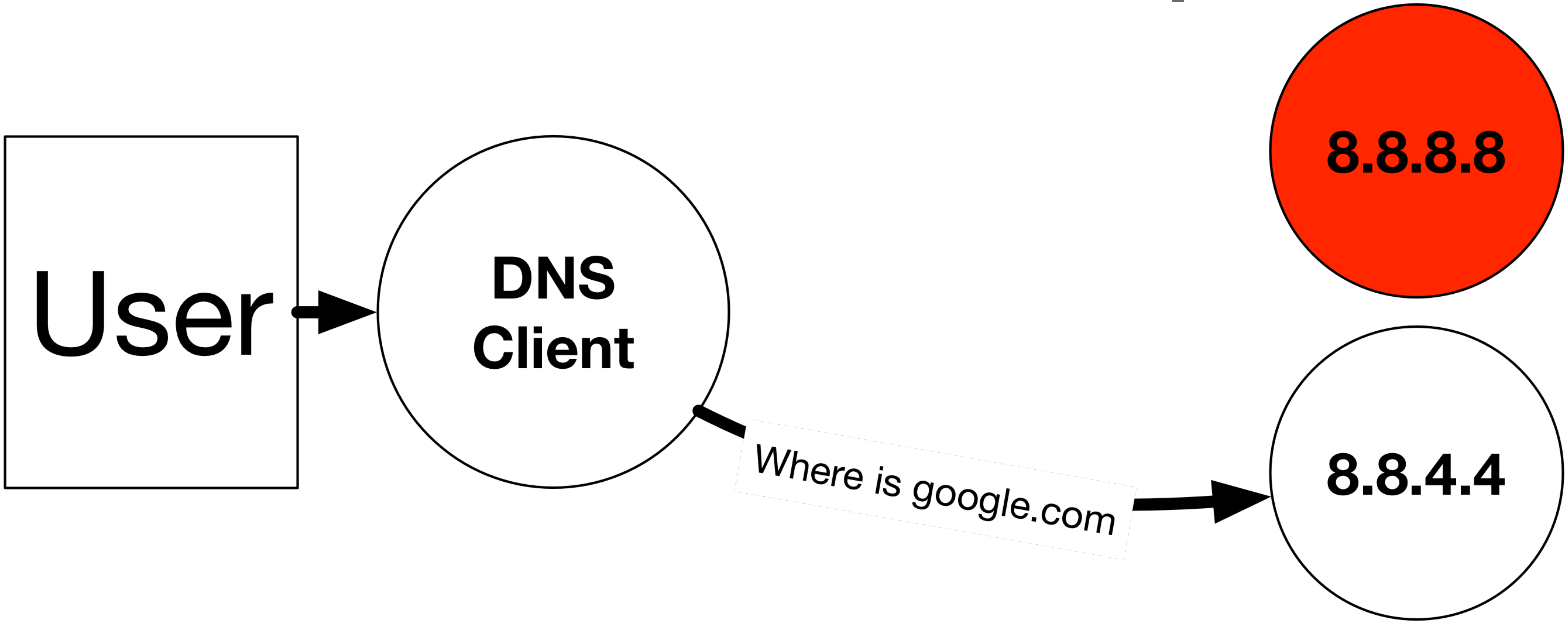
But, sometimes things go wrong

Not so Ideal DNS Request



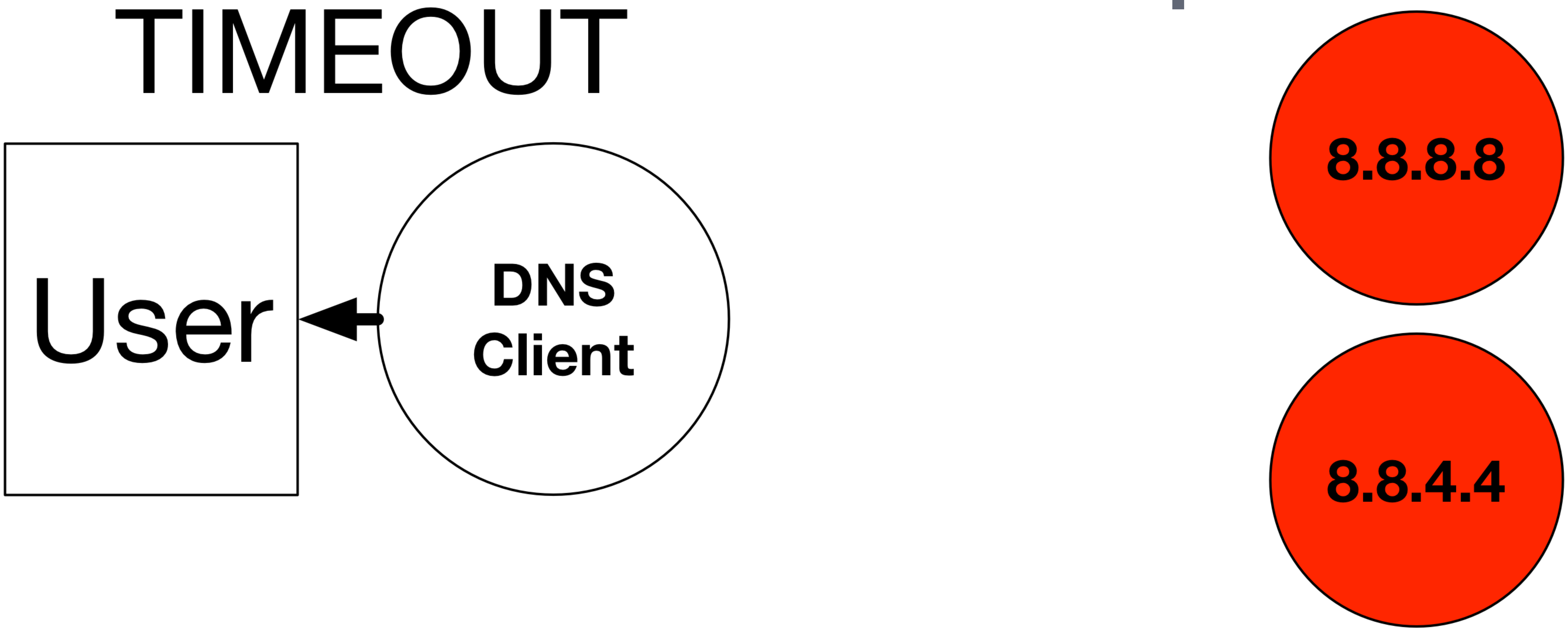
Wait at least 1 second

Not so Ideal DNS Request



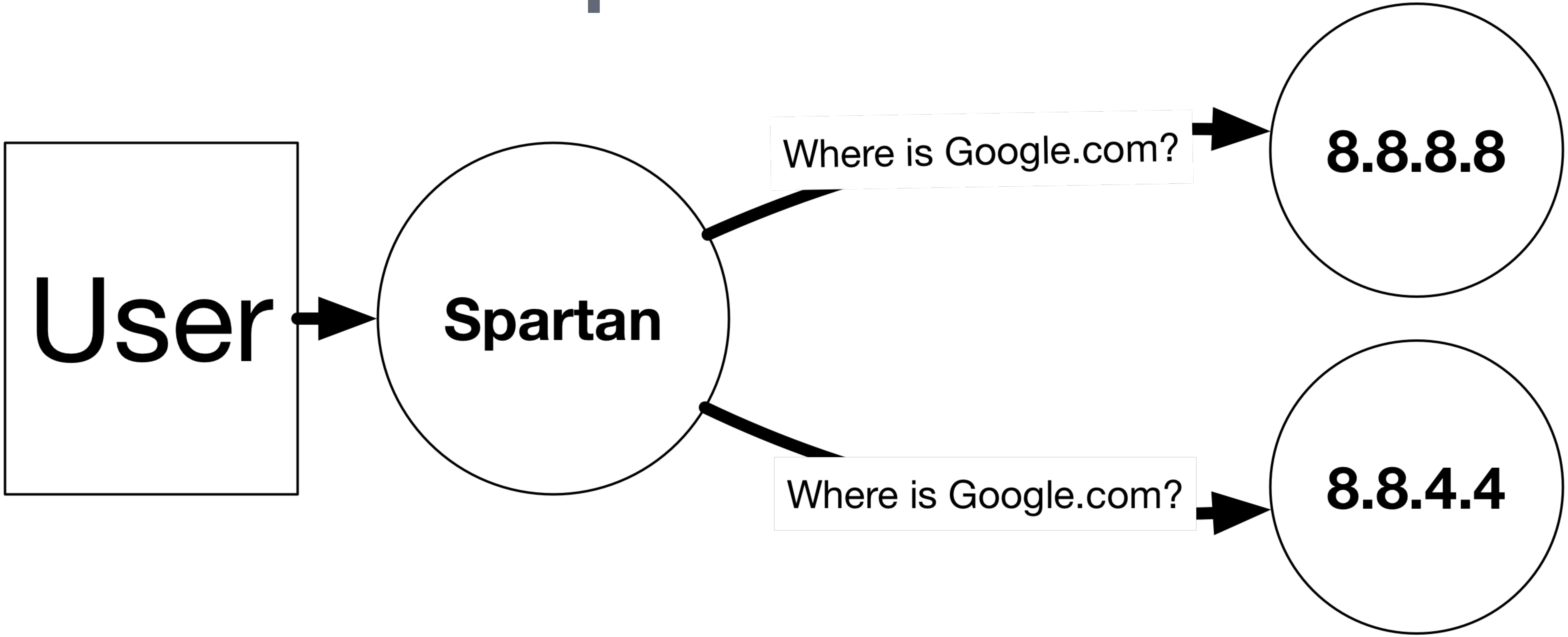
Wait at least 1 second

Not so Ideal DNS Request



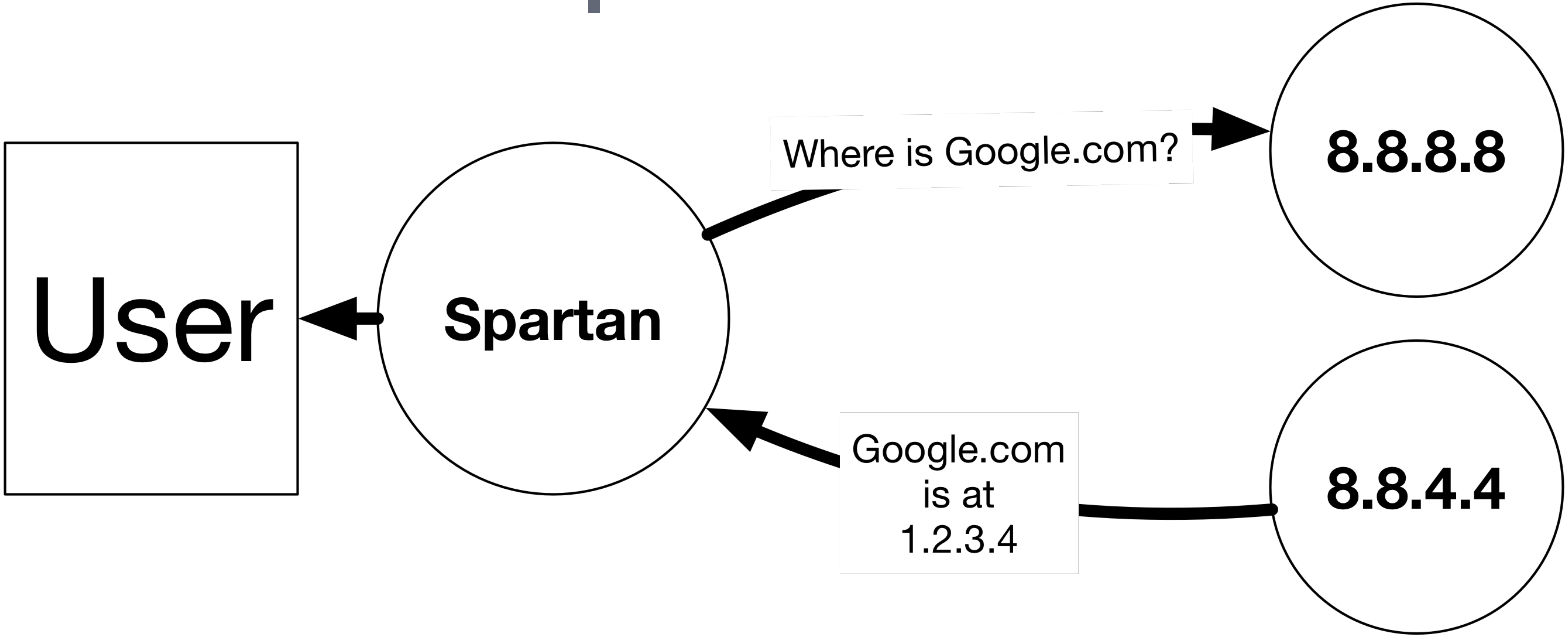
Wat?

Spartan DNS

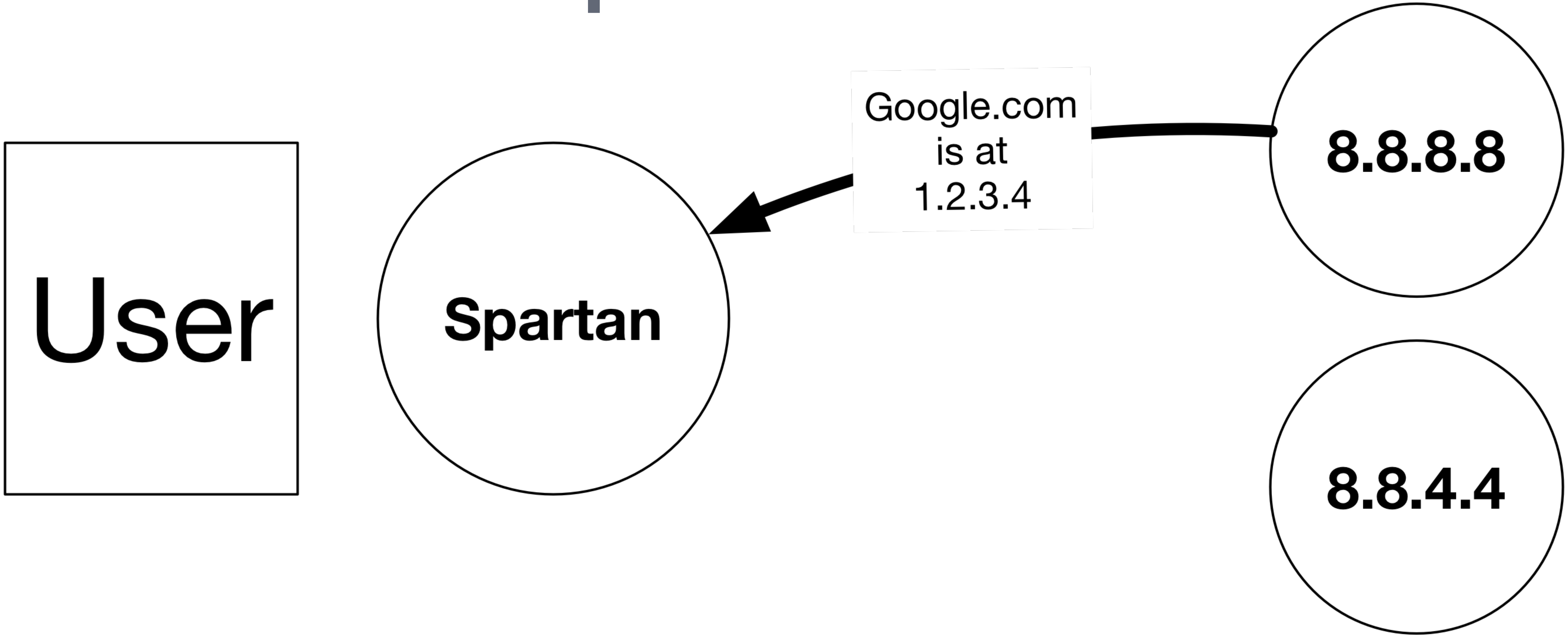


User
Spartan
Where is Google.com?
8.8.8.8
Where is Google.com?
8.8.4.4

Spartan DNS

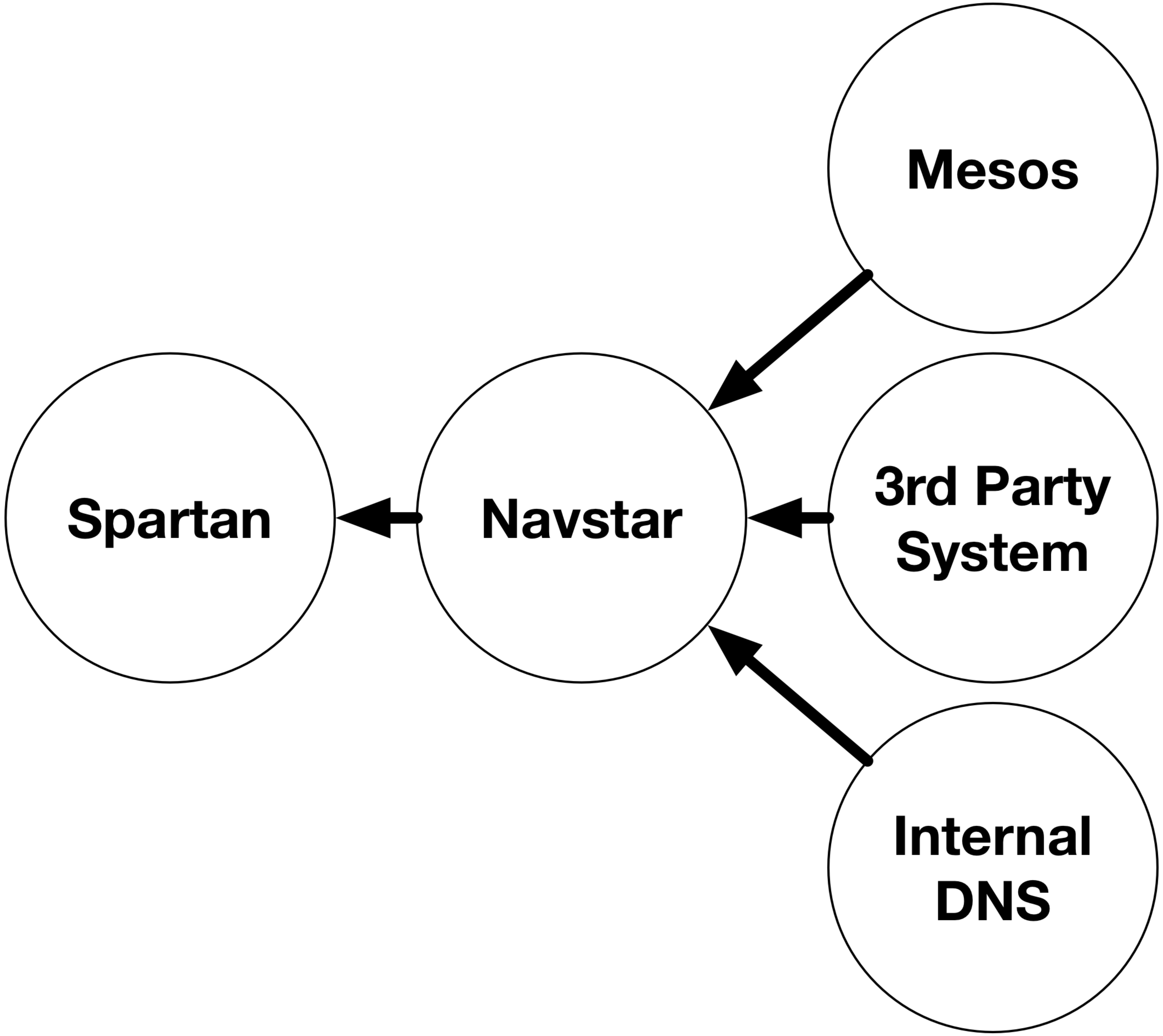

User
Spartan
Where is Google.com?
8.8.8.8
Google.com
is at
1.2.3.4
8.8.4.4

Spartan DNS



Spartan Controls Timeouts

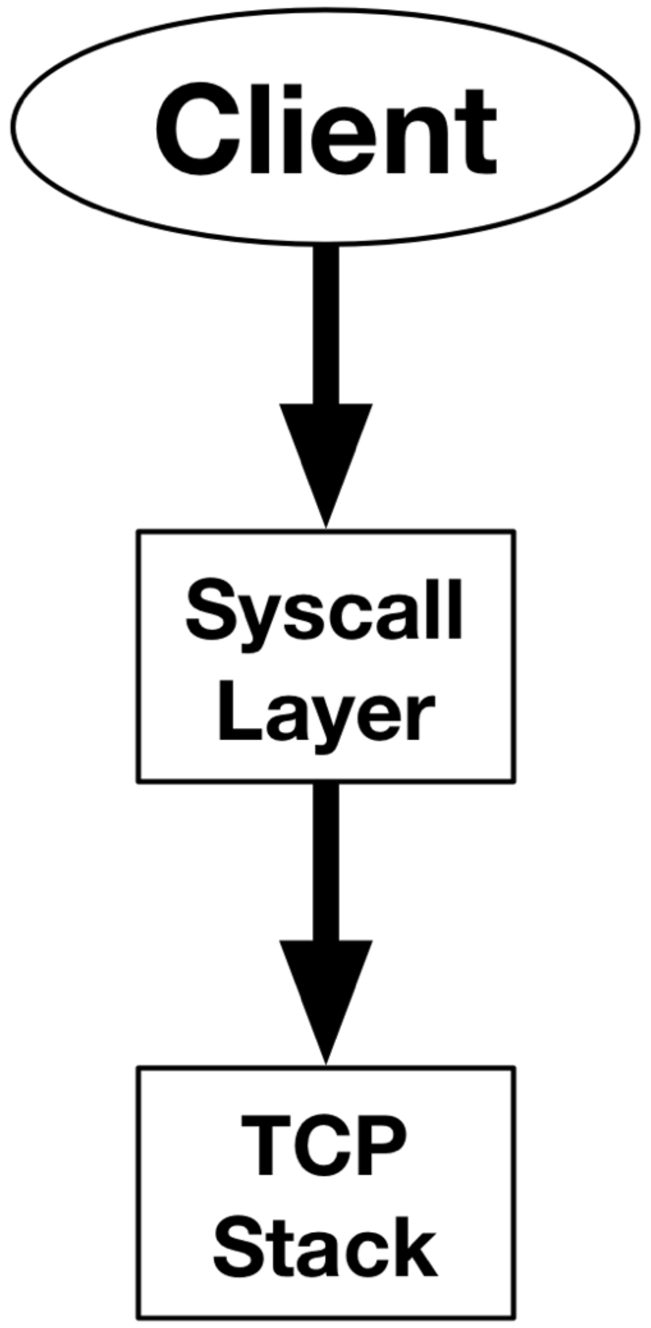
Edge DNS

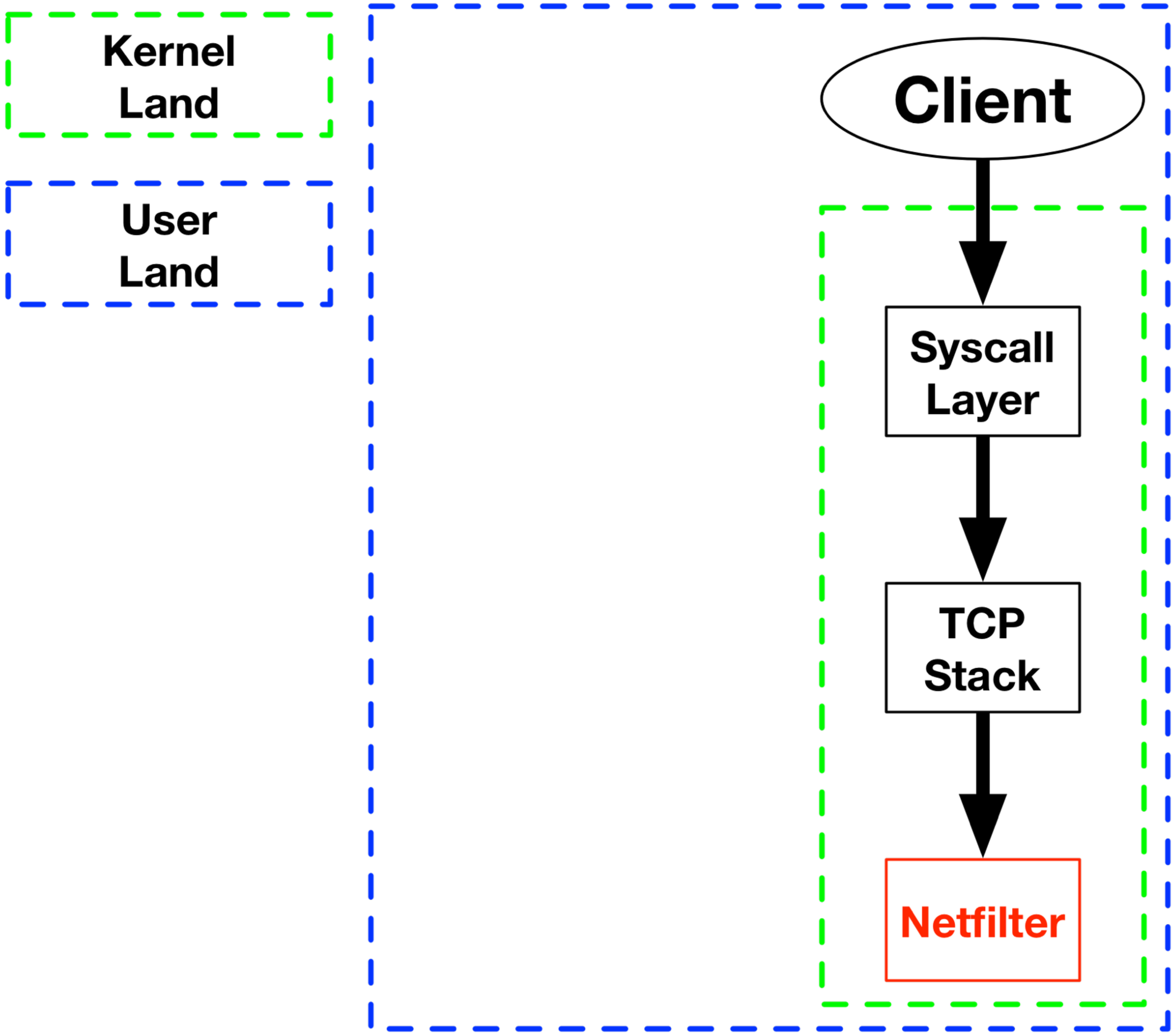

Mesos
Spartan
Navstar
3rd Party
System
Internal
DNS

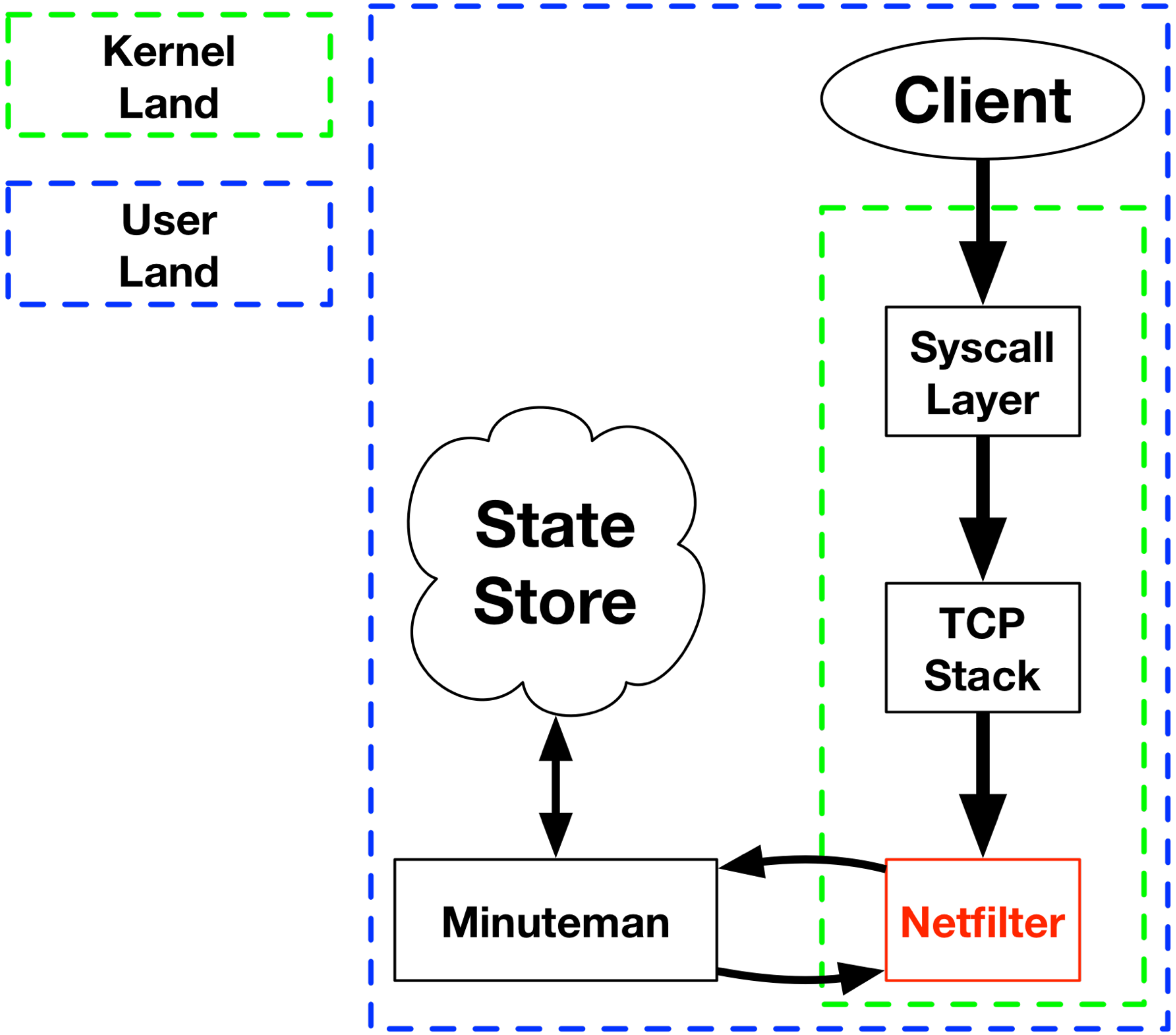
SPARTAN: BENEFITS

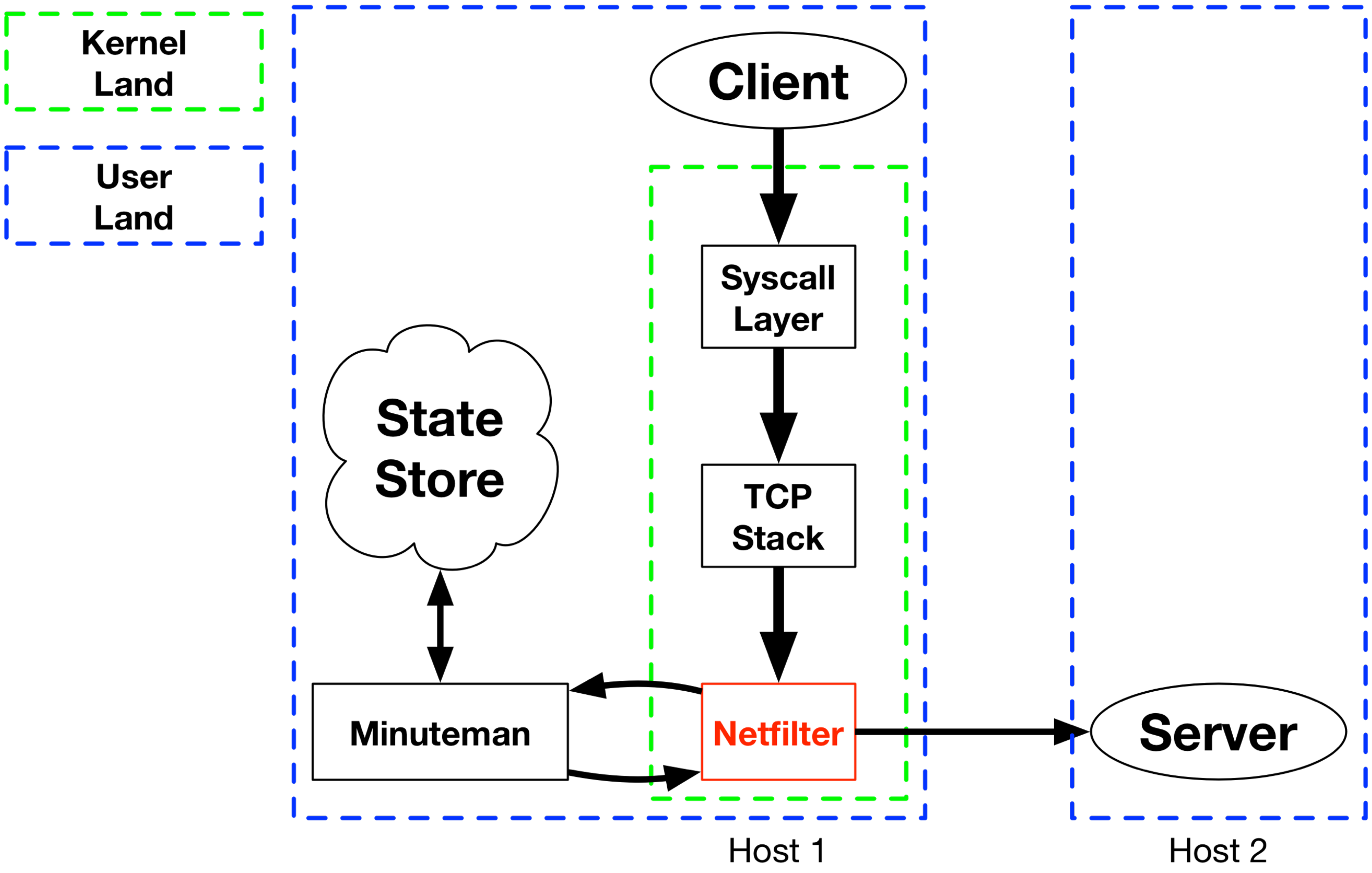
- Makes DNS highly available in light of failure
- Lowers Average Latency
- Can control timeouts to sub-second
- Edge DNS
    - Many DNS queries don’t need to go off-node

Load Balancing: Minuteman


Client
Syscall Layer
TCP Stack

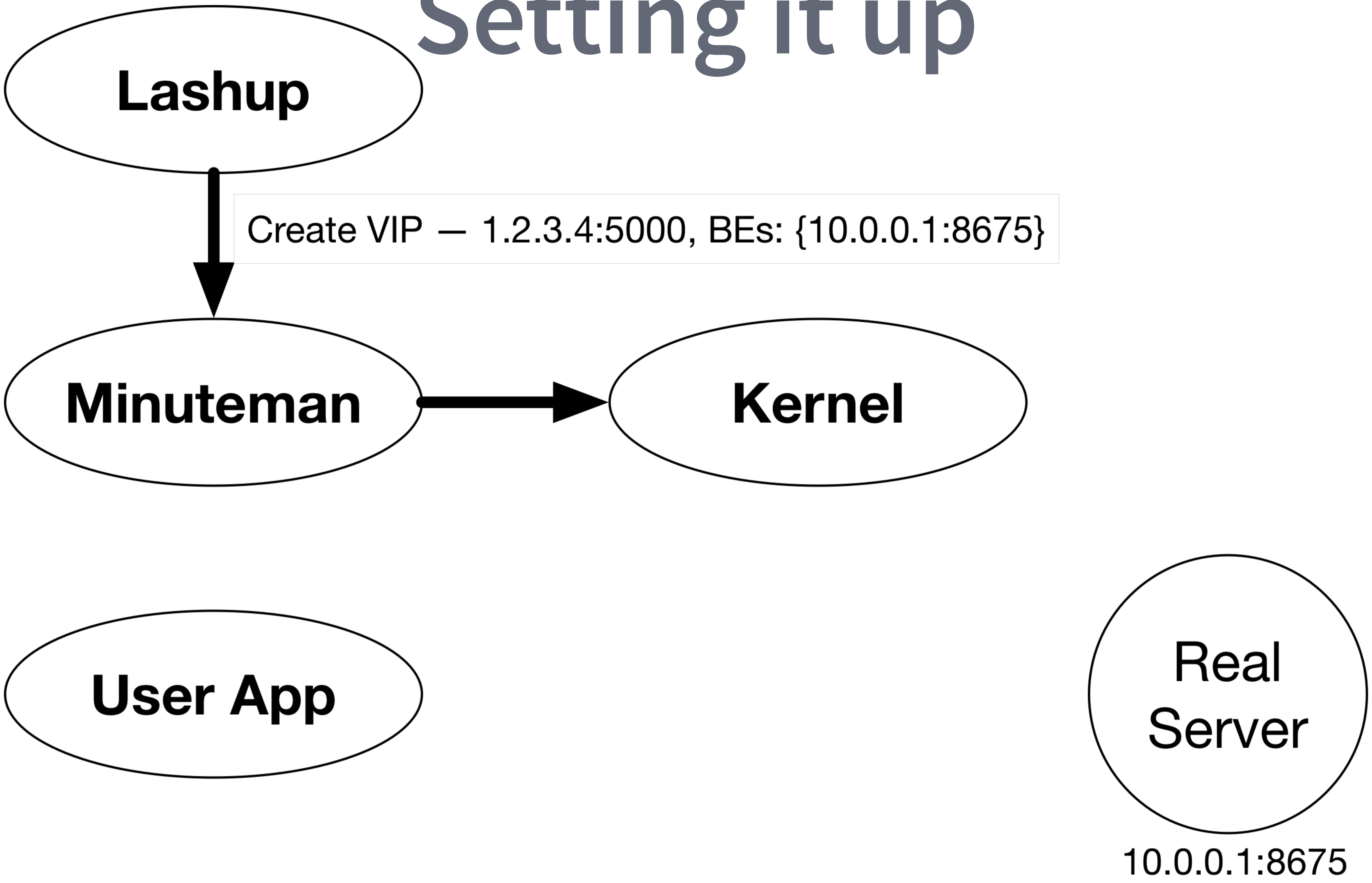

Kernel Land
User Land
Client
Syscall Layer
TCP Stack
Netfilter


Kernel Land
User Land
Client
Syscall Layer
TCP Stack
State Store
Minuteman
Netfilter

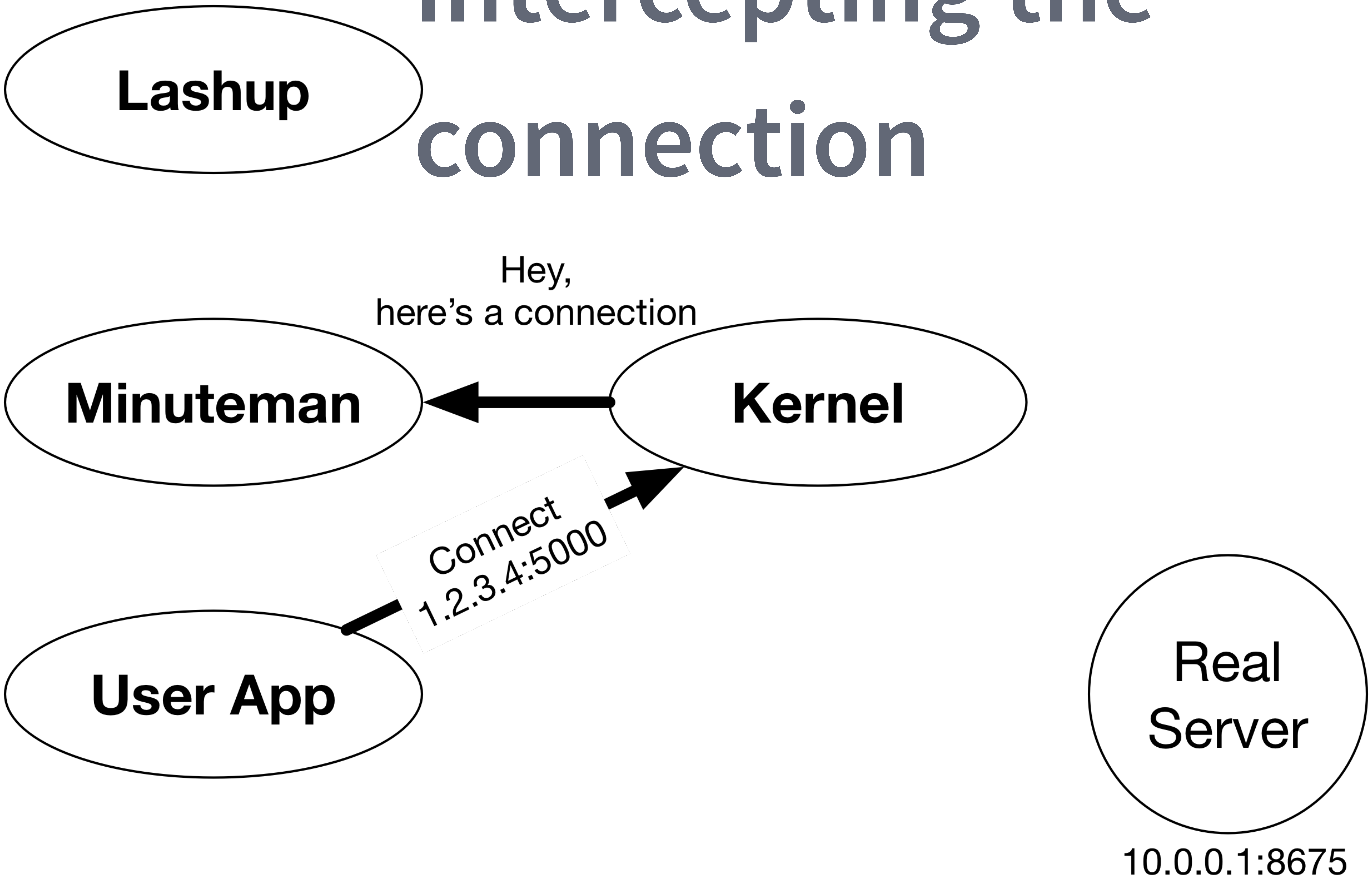

Kernel Land
User Land
Client
Syscall Layer
TCP Stack
State Store
Minuteman
Netfilter
Server
Host 1
Host 2

How does it work?

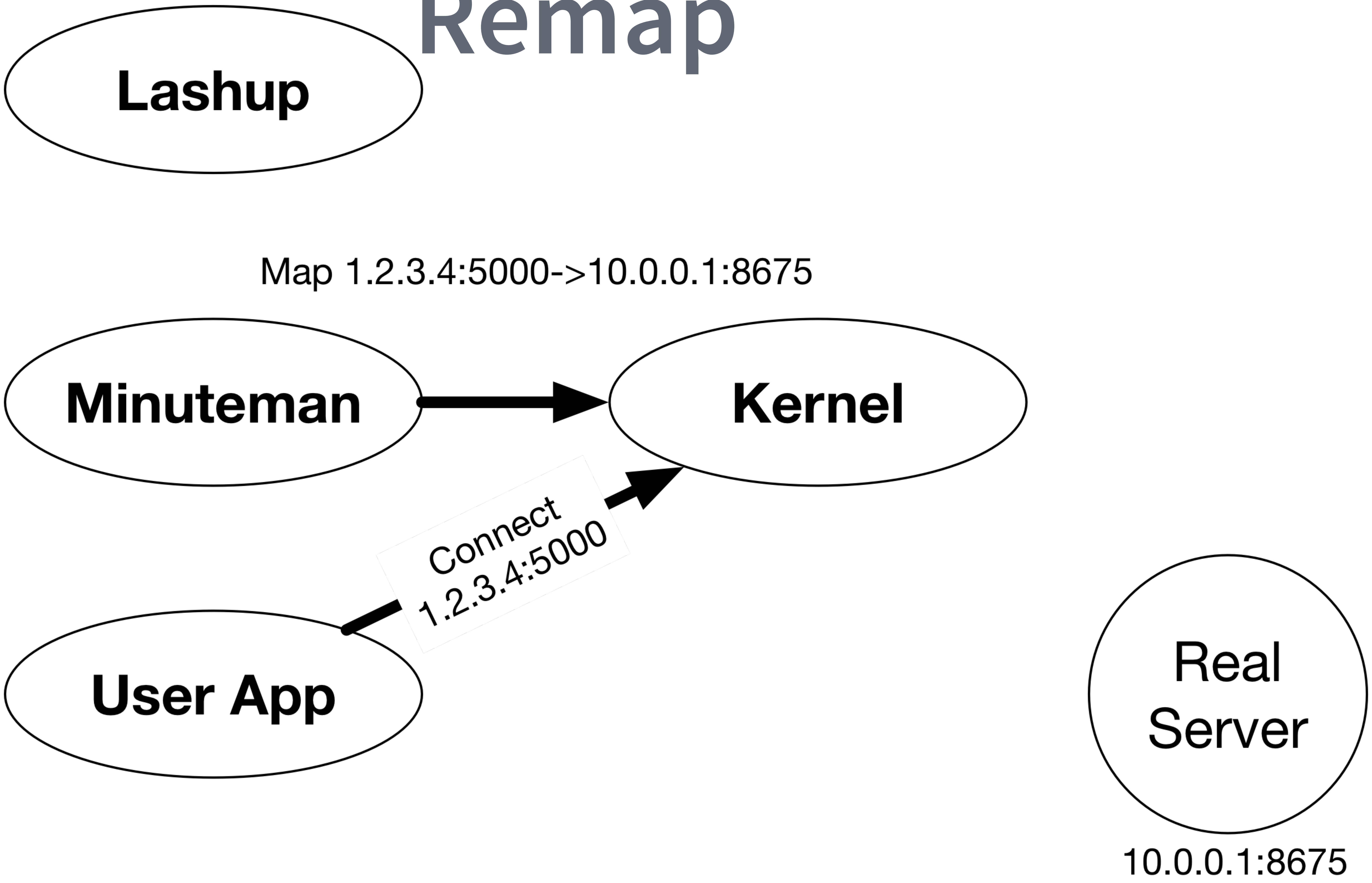
Setting it up



Intercepting the connection

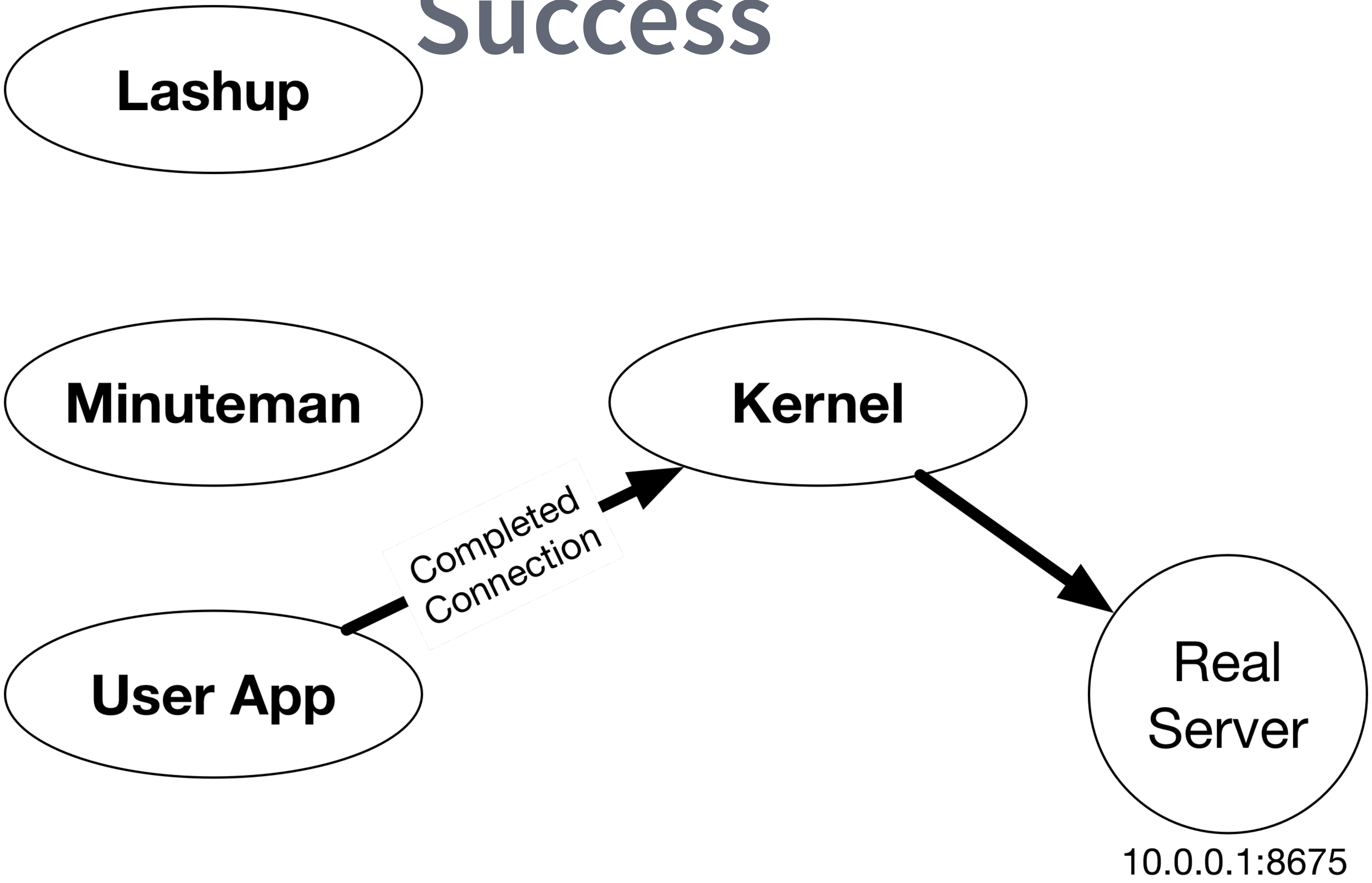


Remap



Lashup
Map 1.2.3.4:5000->10.0.0.1:8675
Minuteman
Kernel
Connect
1.2.3.4:5000
User App
Real
Server
10.0.0.1:8675

Success

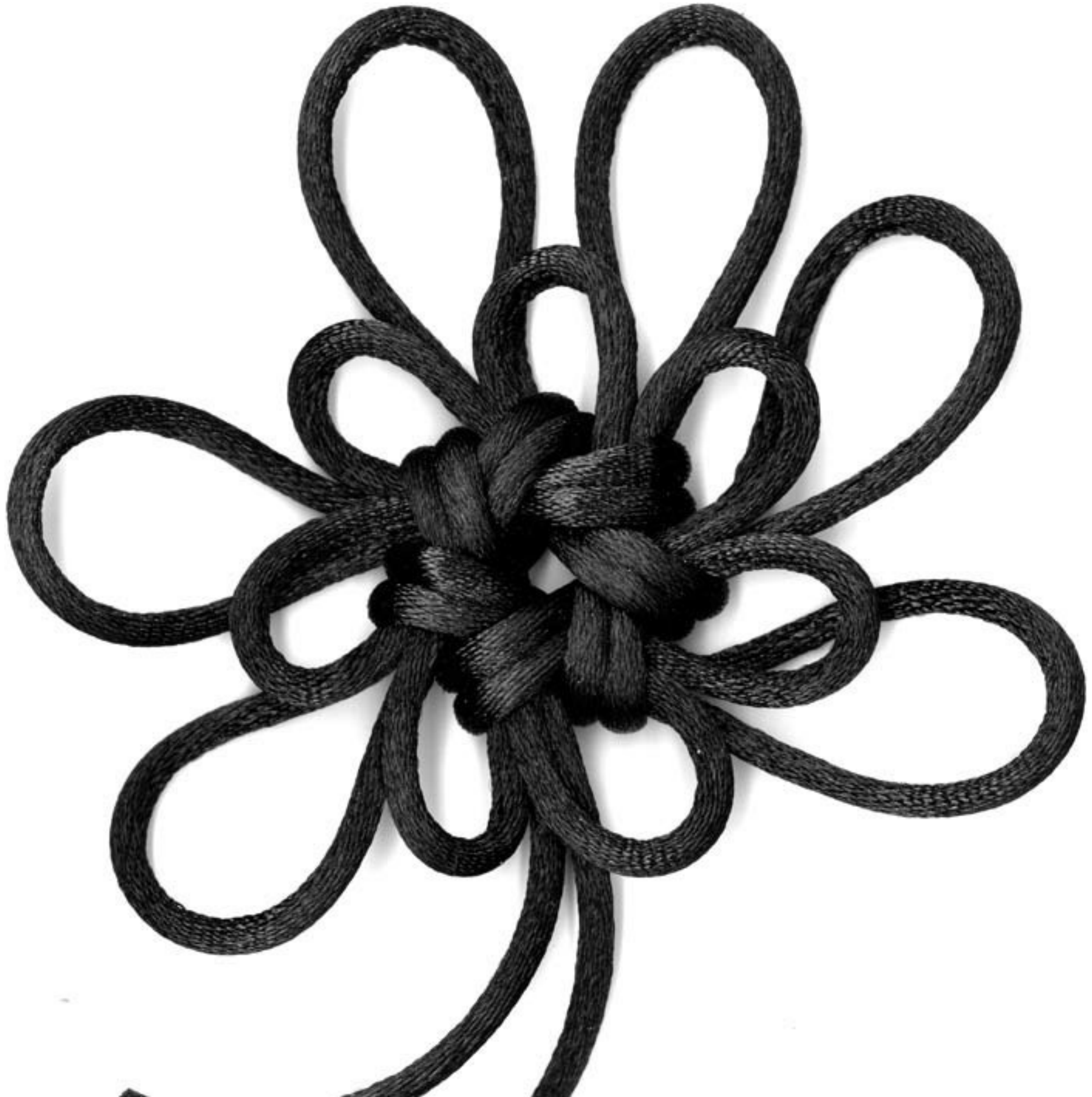


Lashup
Minuteman
Kernel
User App
Completed Connection
Real Server
10.0.0.1:8675

MINUTEMAN: BENEFITS

- Appearance of a fixed-load balancer
- Fully distributed
- Other than first packet, the entire lifetime is handled in kernel space
- Erlang allows us to maintain latency guarantees

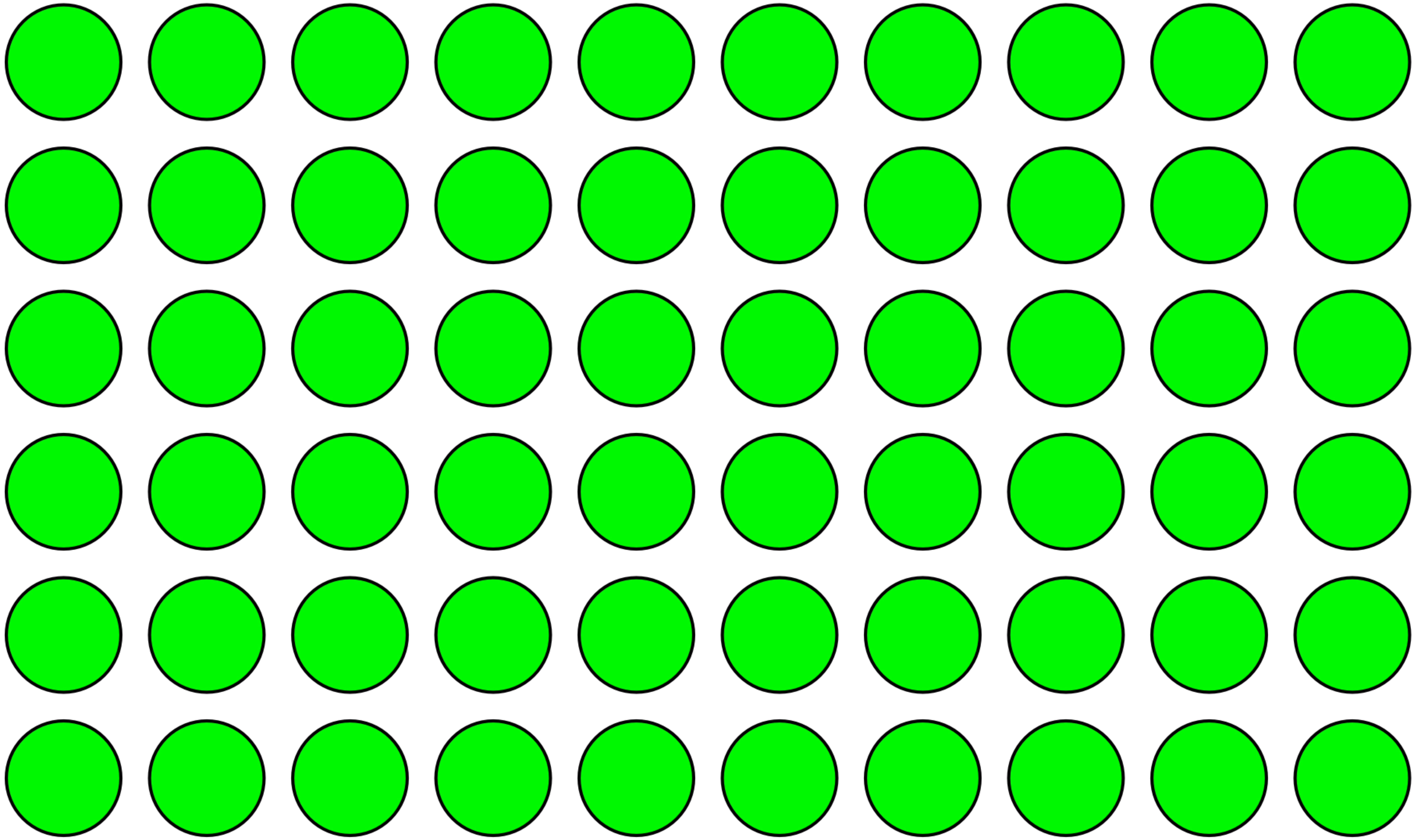
How do we tie it all together?



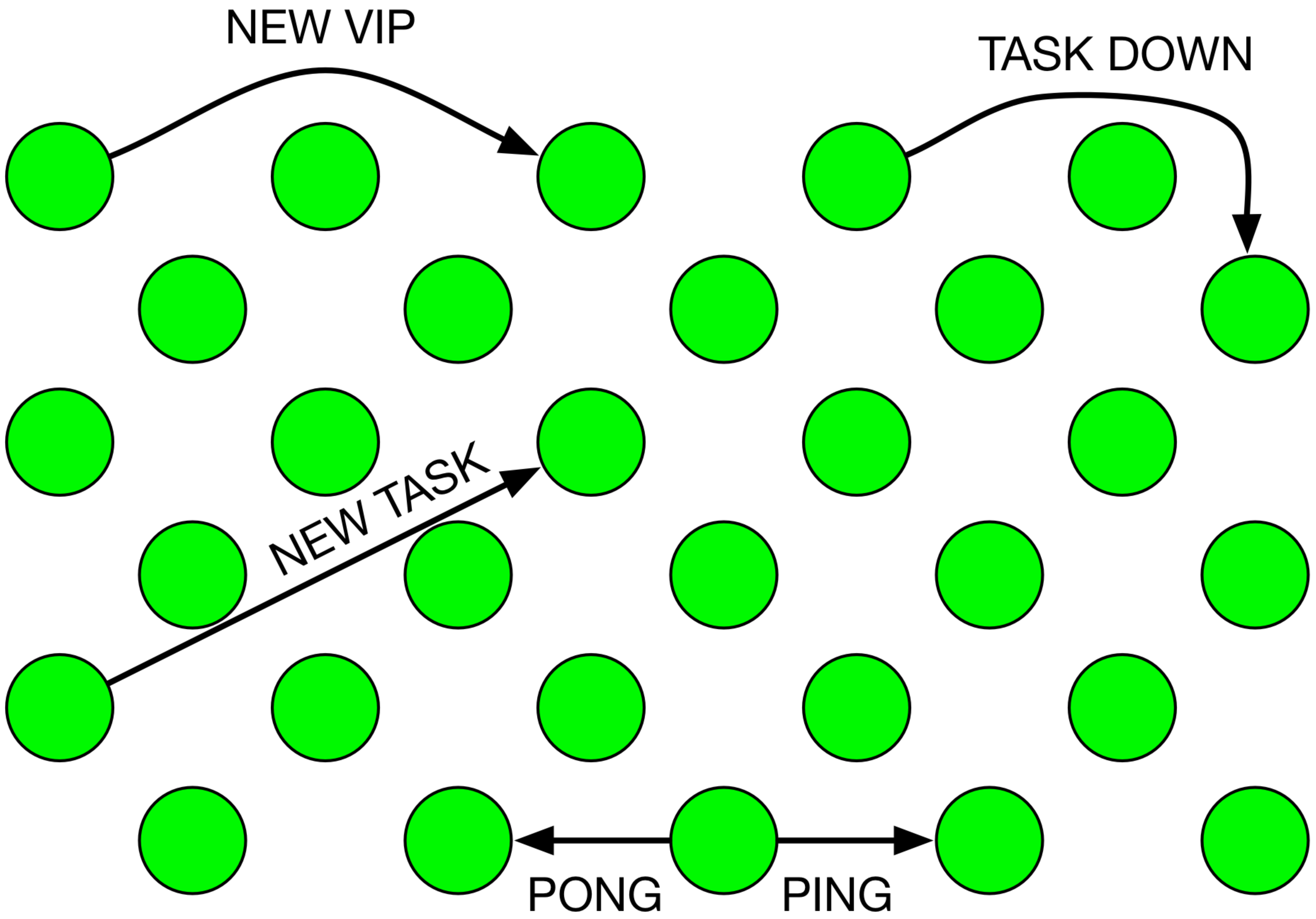
GLOBAL STATE

- Load Balancer Task Mapping
- DNS Zones
- Virtual Network Routing Tables
- Reachability
- Security ACLs

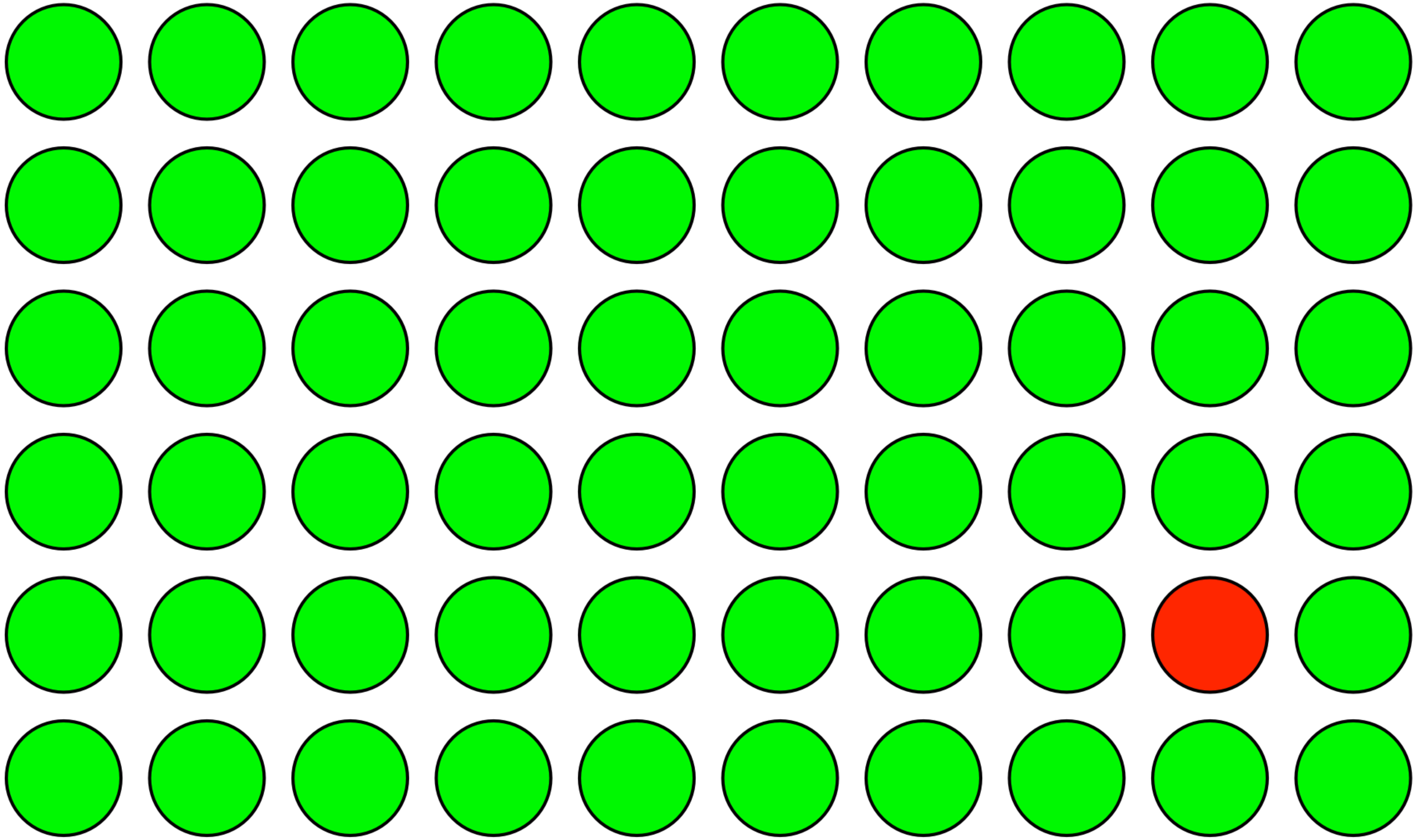
Computers



Constant Churn

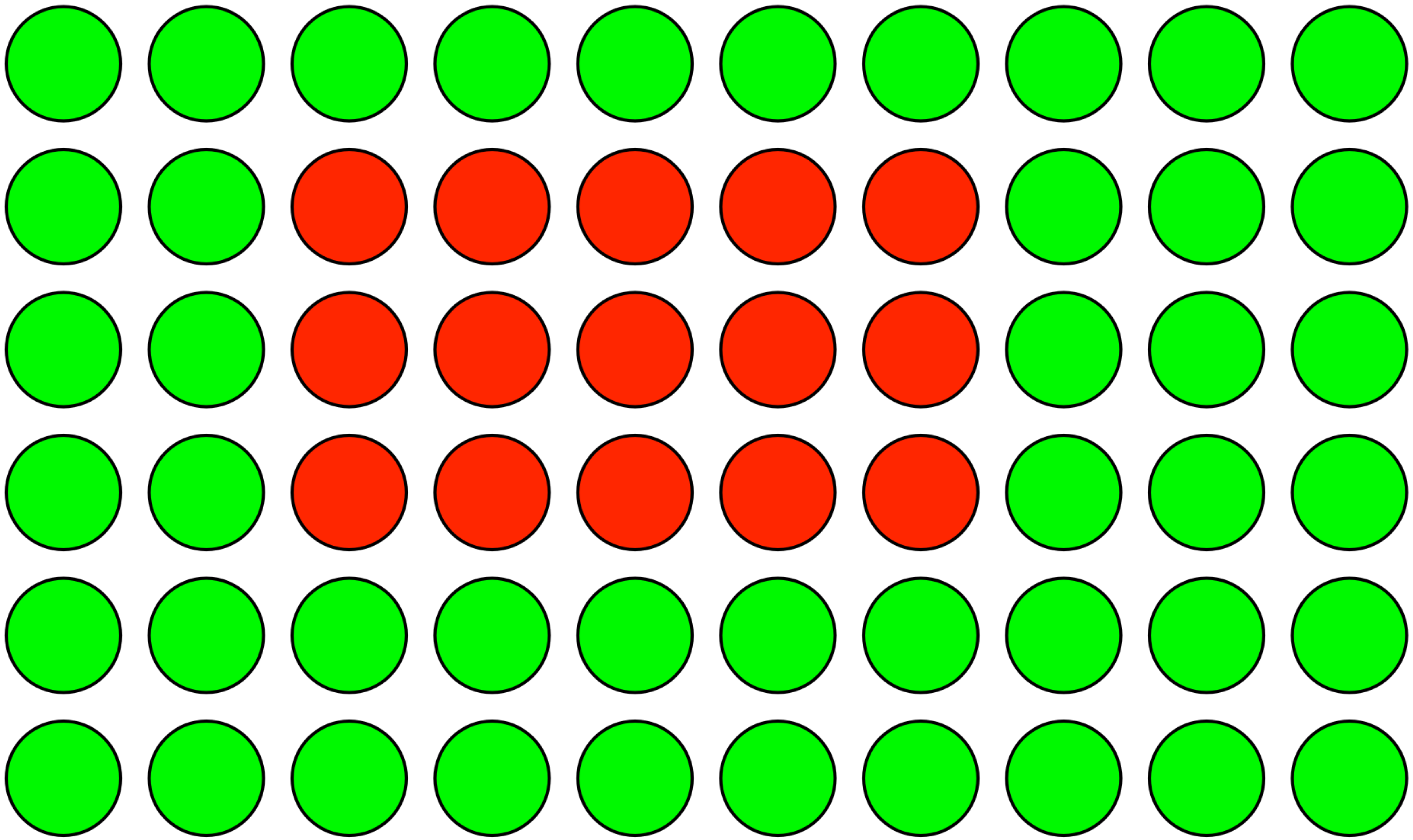


Sometimes



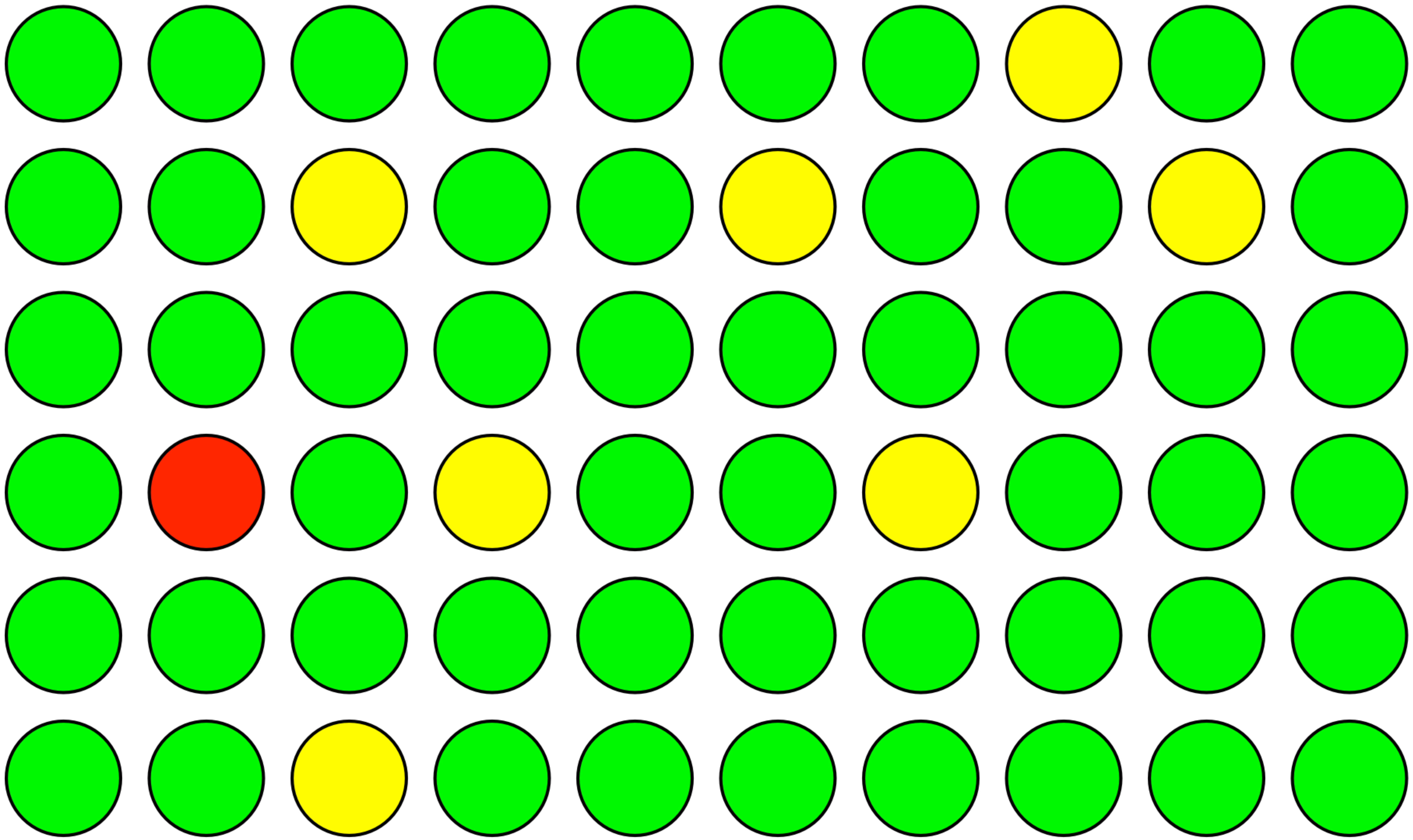
They Break

Sometimes



Many Break

Sometimes

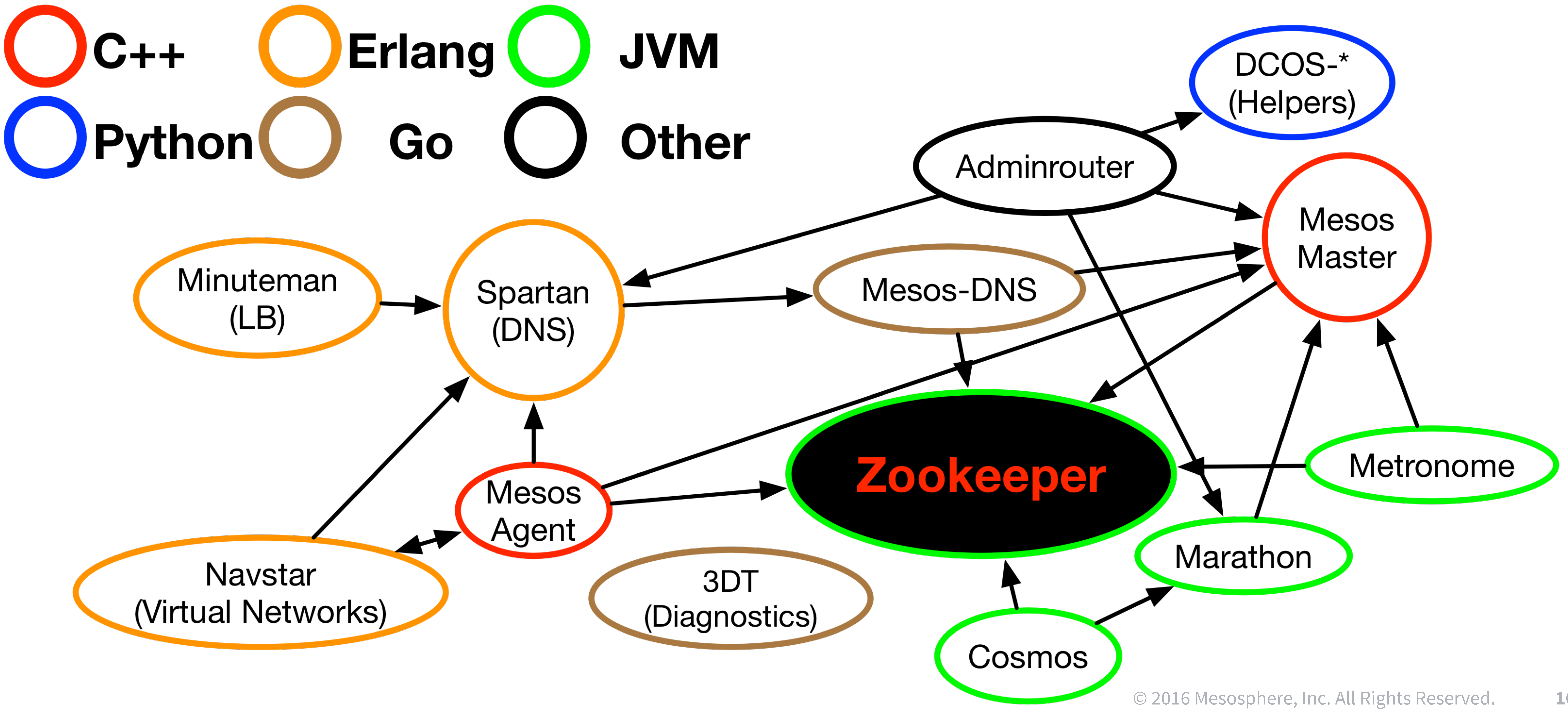


You don’t know

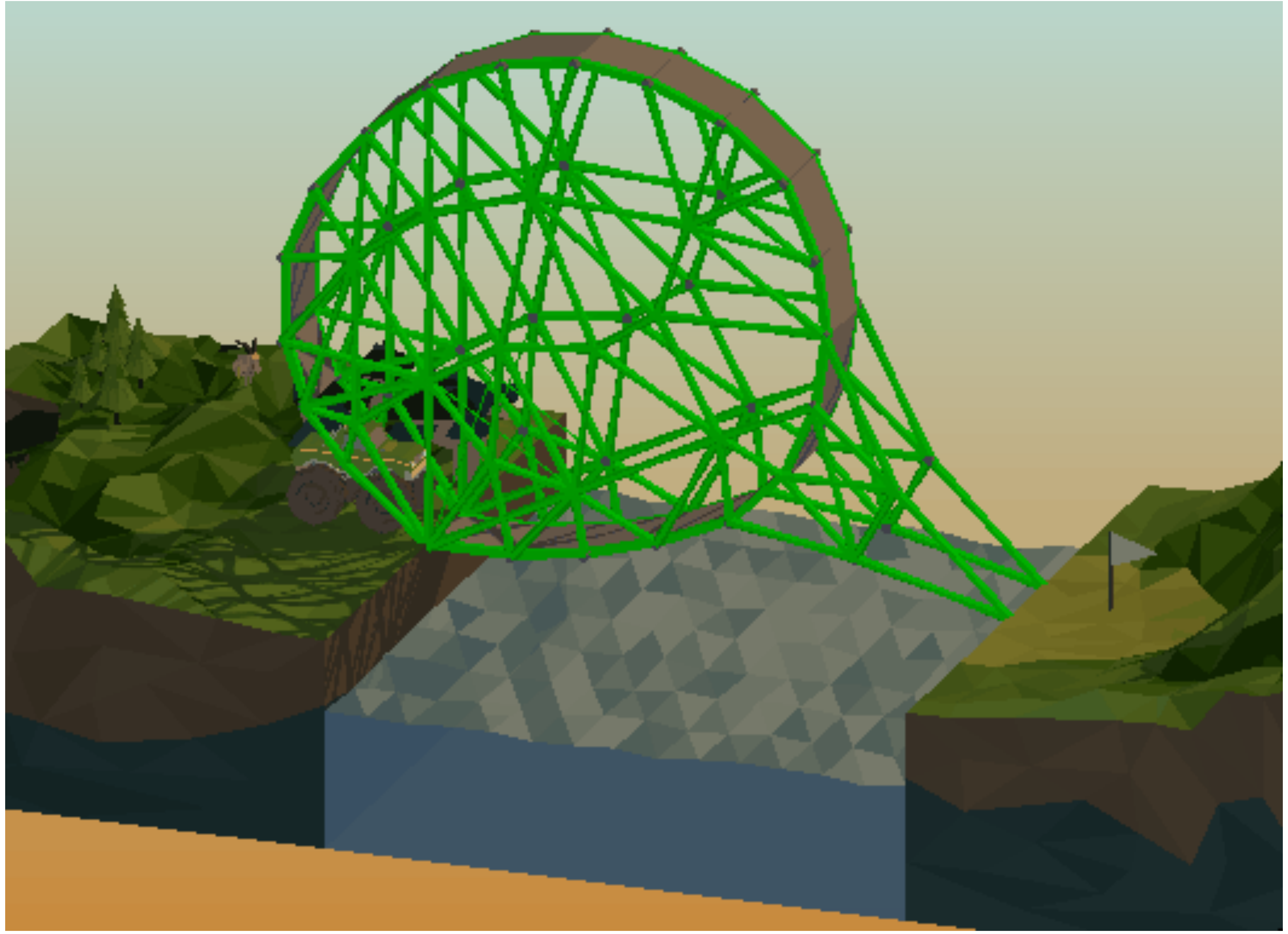
HOW DO YOU DO SIGNALING?

Nobody ever got fired for using Zookeeper

We know about Zookeeper



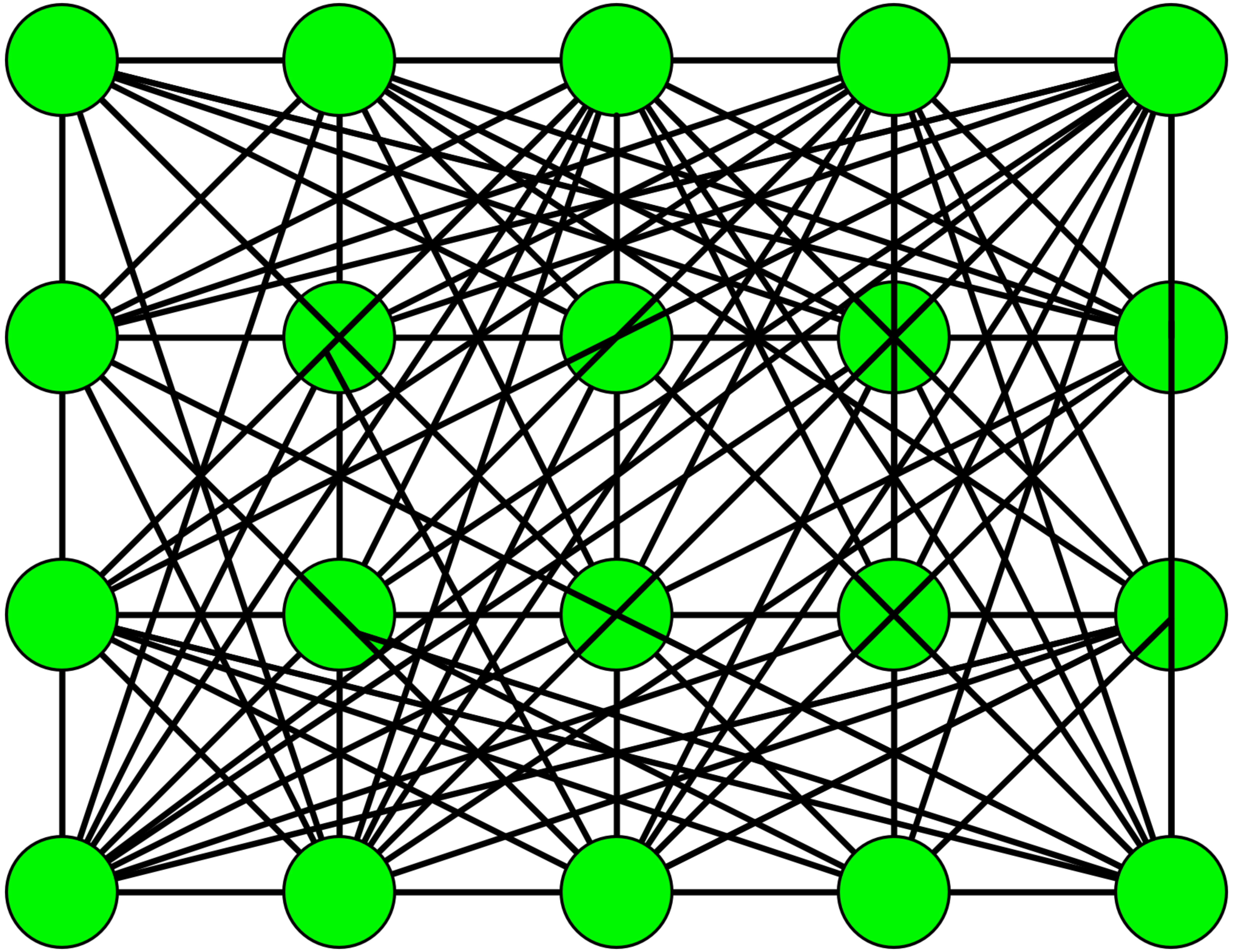
It works…usually



How else can we do this?

Naively

(See: Distributed Erlang)



Massive Amount of Information

Who else has solved this?

Academia

A Note on Two Problems in Connexion with Graphs

By

E. W. DIJKSTRA

HyParView: a membership protocol for reliable gossip-based broadcast

João Leitão
José Pereira
Luís Rodrigues

A Gossip-Style Failure Detection Service

Robbert van Renesse, Yaron Minsky, and Mark Hayden*

SWIM: *S*calable *W*eakly-consistent *I*nfection-style Process Group *M*embership Protocol

Abhinandan Das, Indranil Gupta, Ashish Motivala*
Dept. of Computer Science, Cornell University
Ithaca NY 14853 USA
{asdas,gupta,ashish}@cs.cornell.edu

EPIDEMIC ALGORITHMS FOR REPLICATED DATABASE MAINTENANCE

Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry

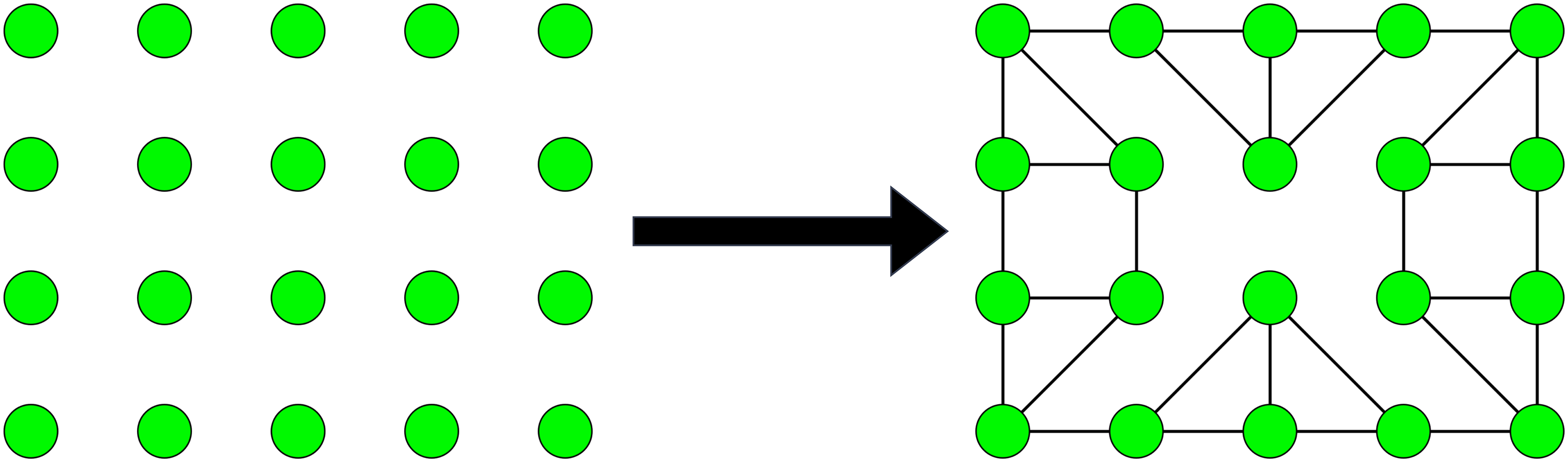
Xerox Palo Alto Research Center

Control Plane:
Lashup

How do we scale the naive approach?

All we need is a sparse, connected graph

But how?

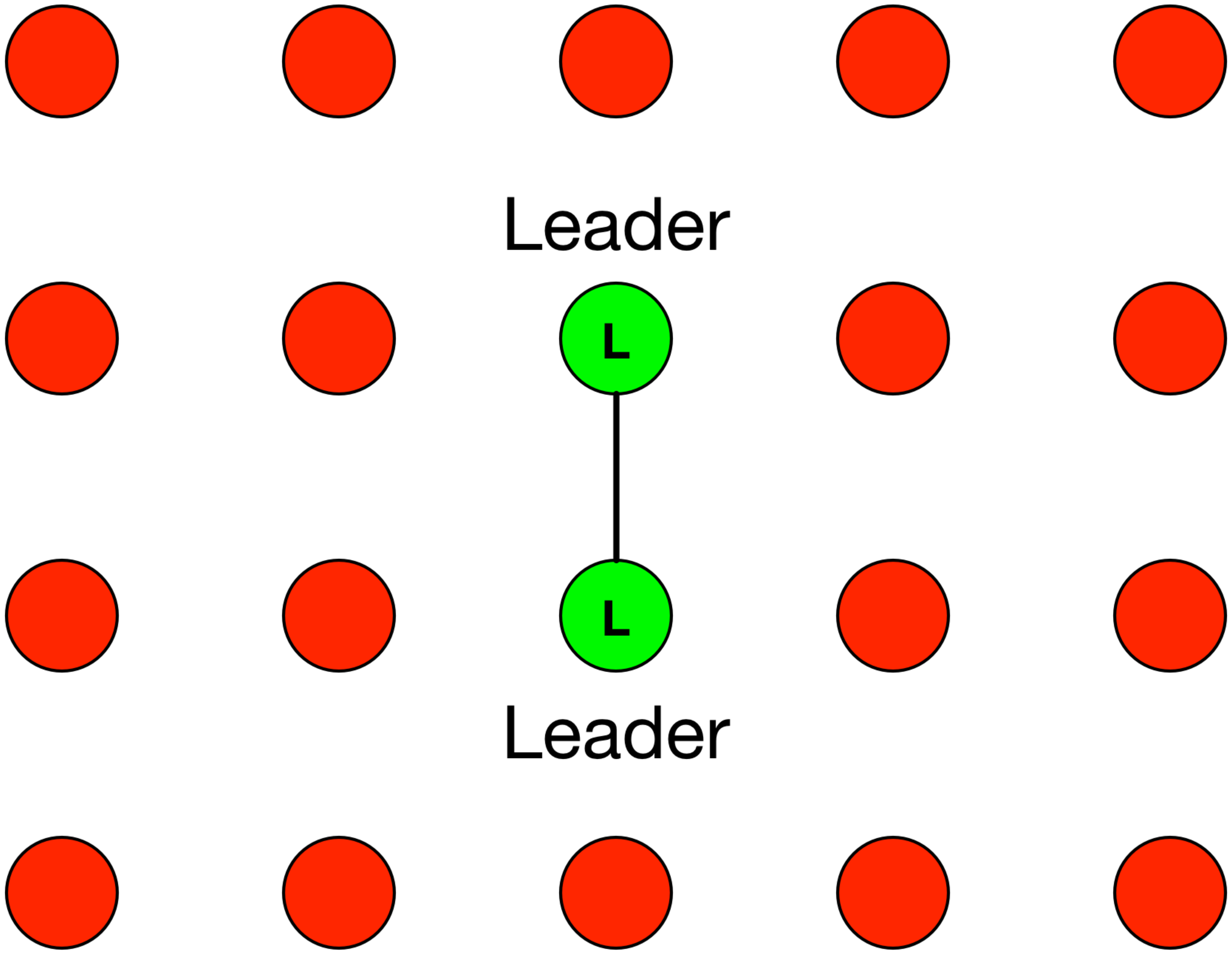


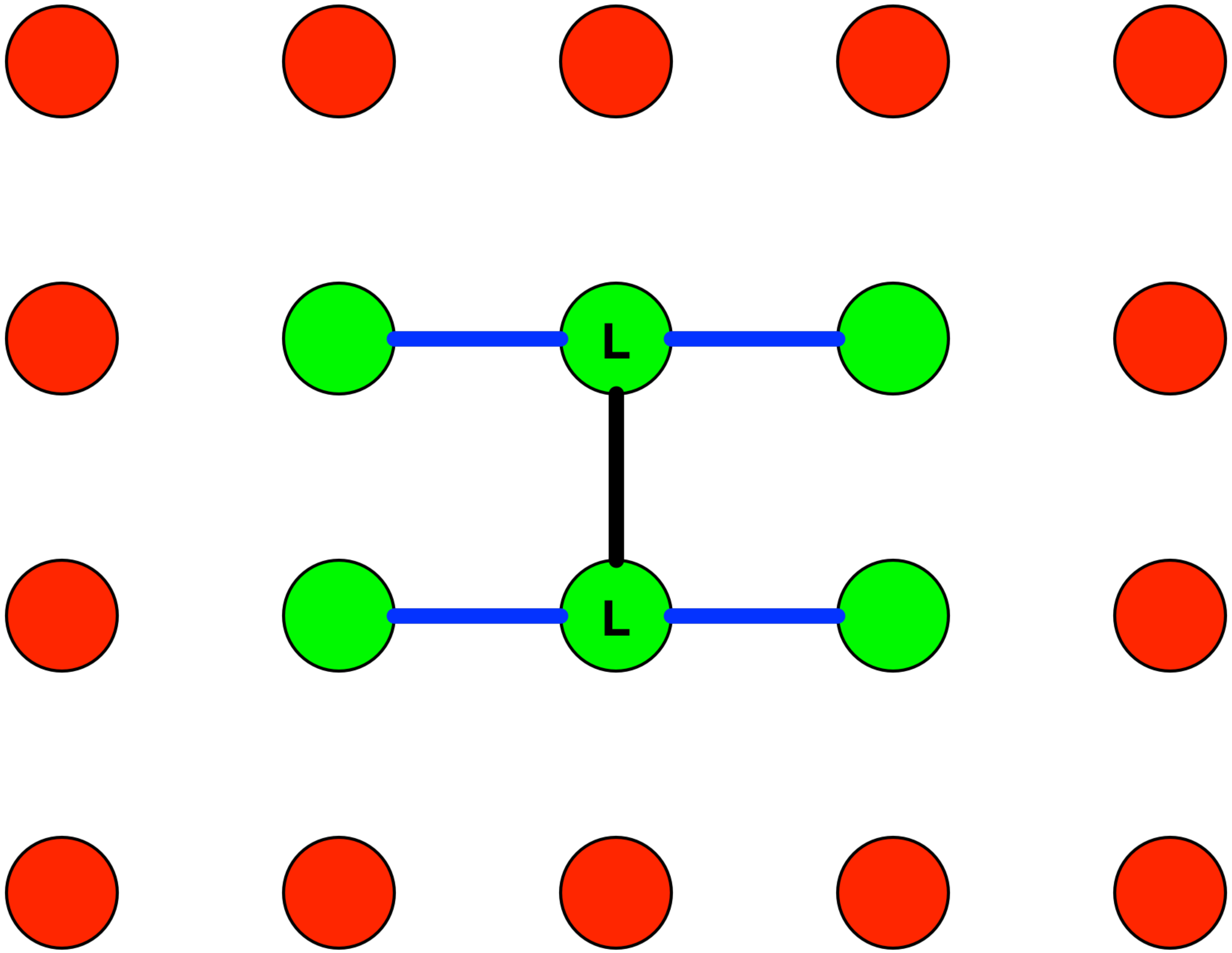
Enter: HyParView*

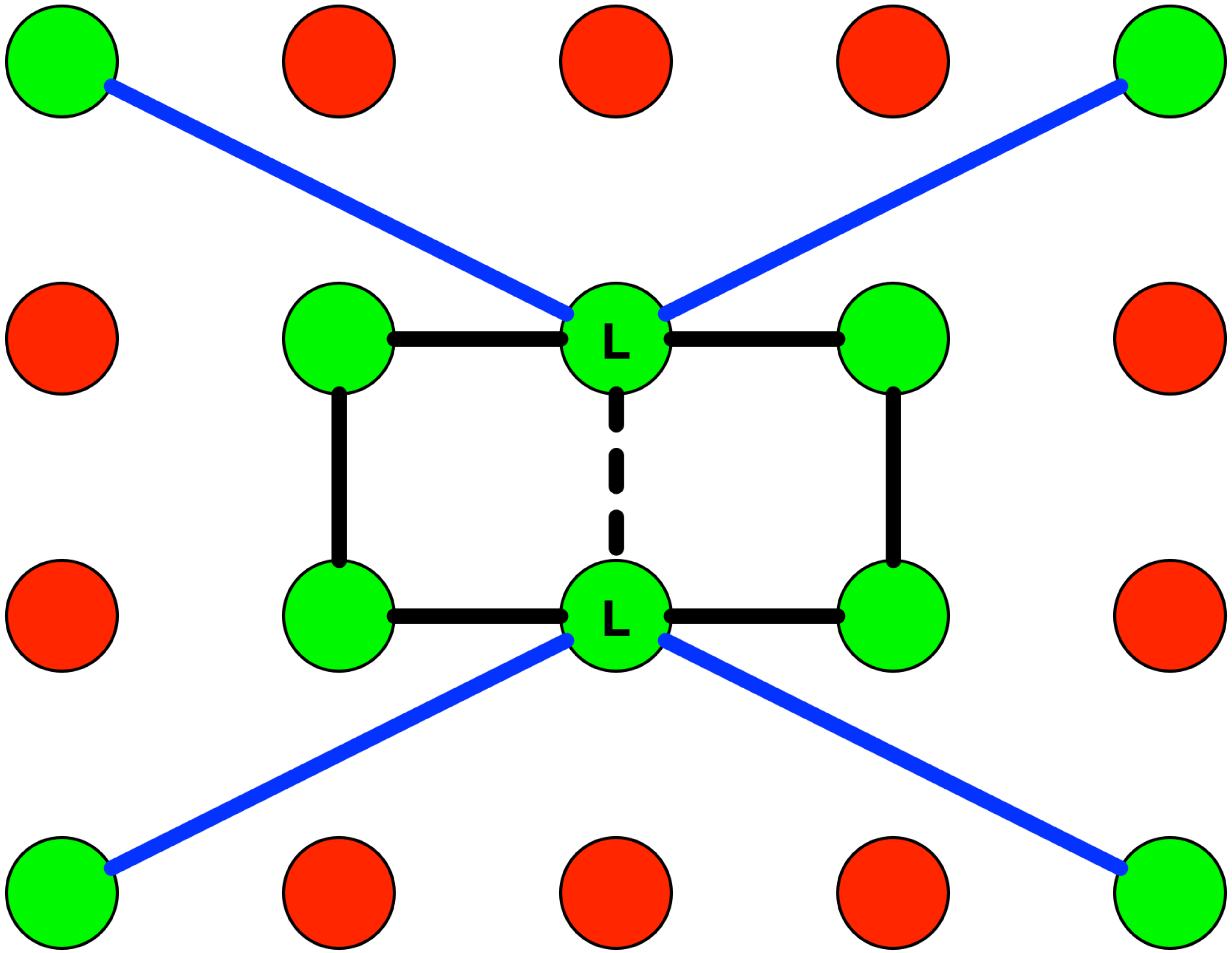
Builds a connected graph (overlay), where the degree is <=K

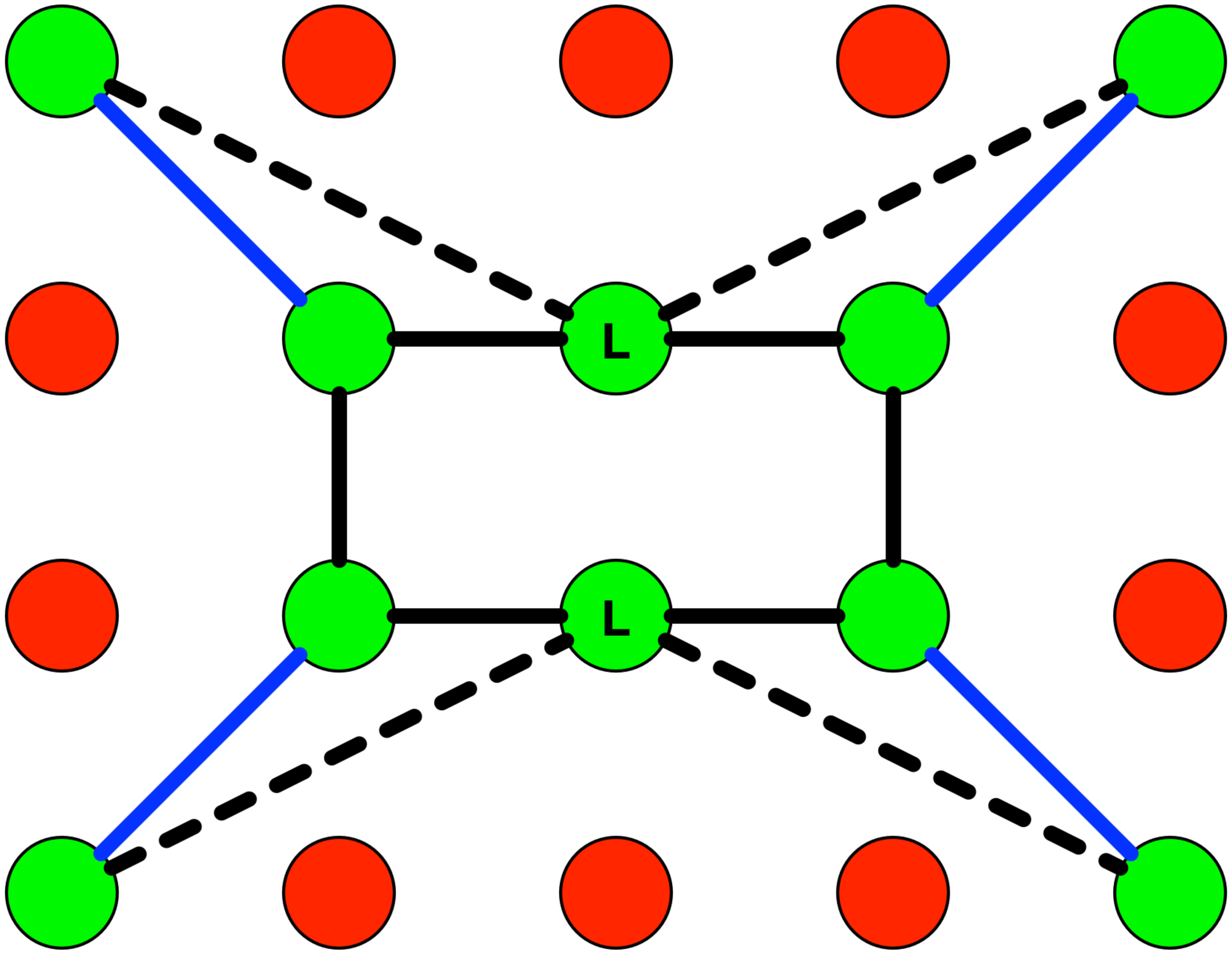
*Inspired Protocol

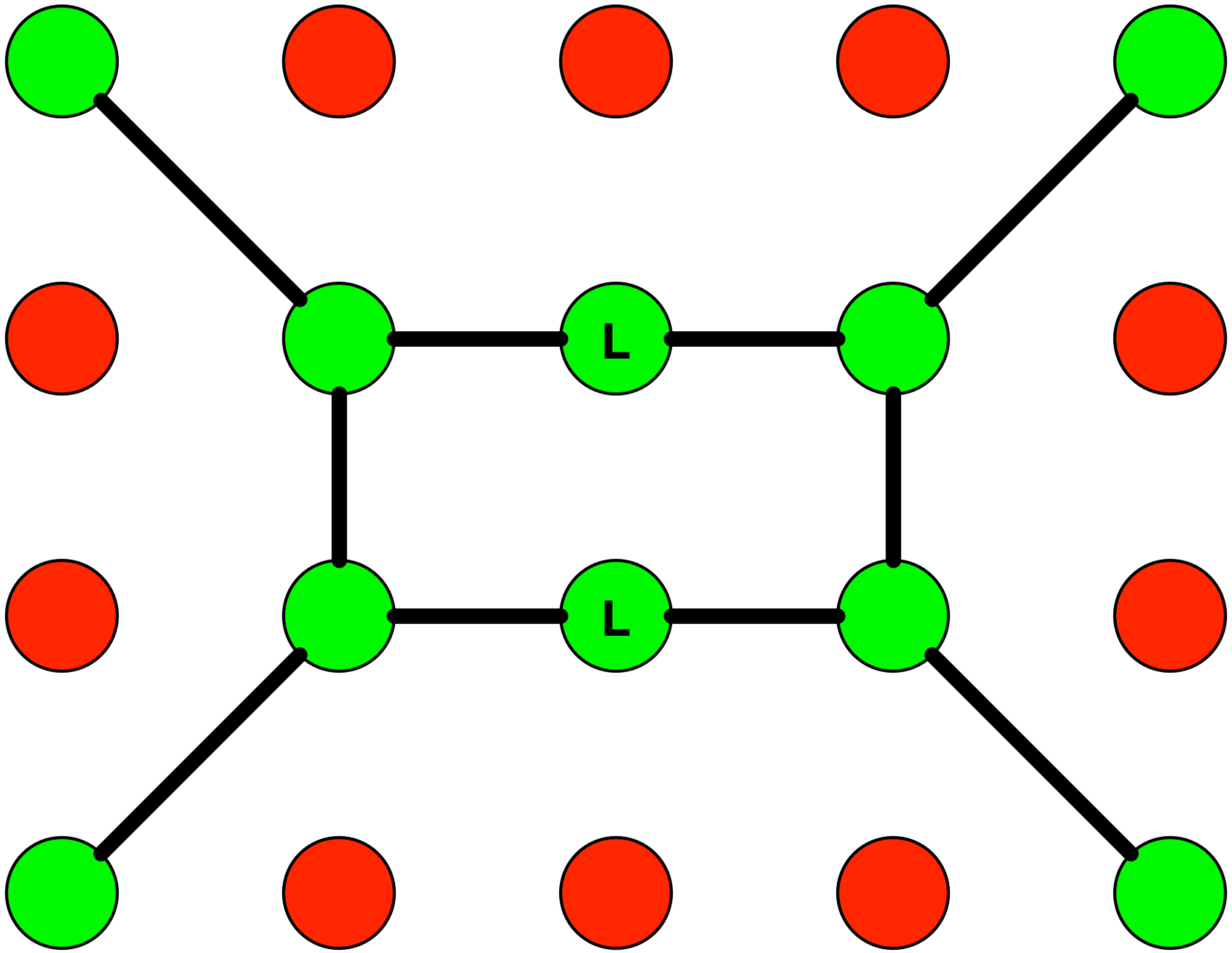
Boot Time

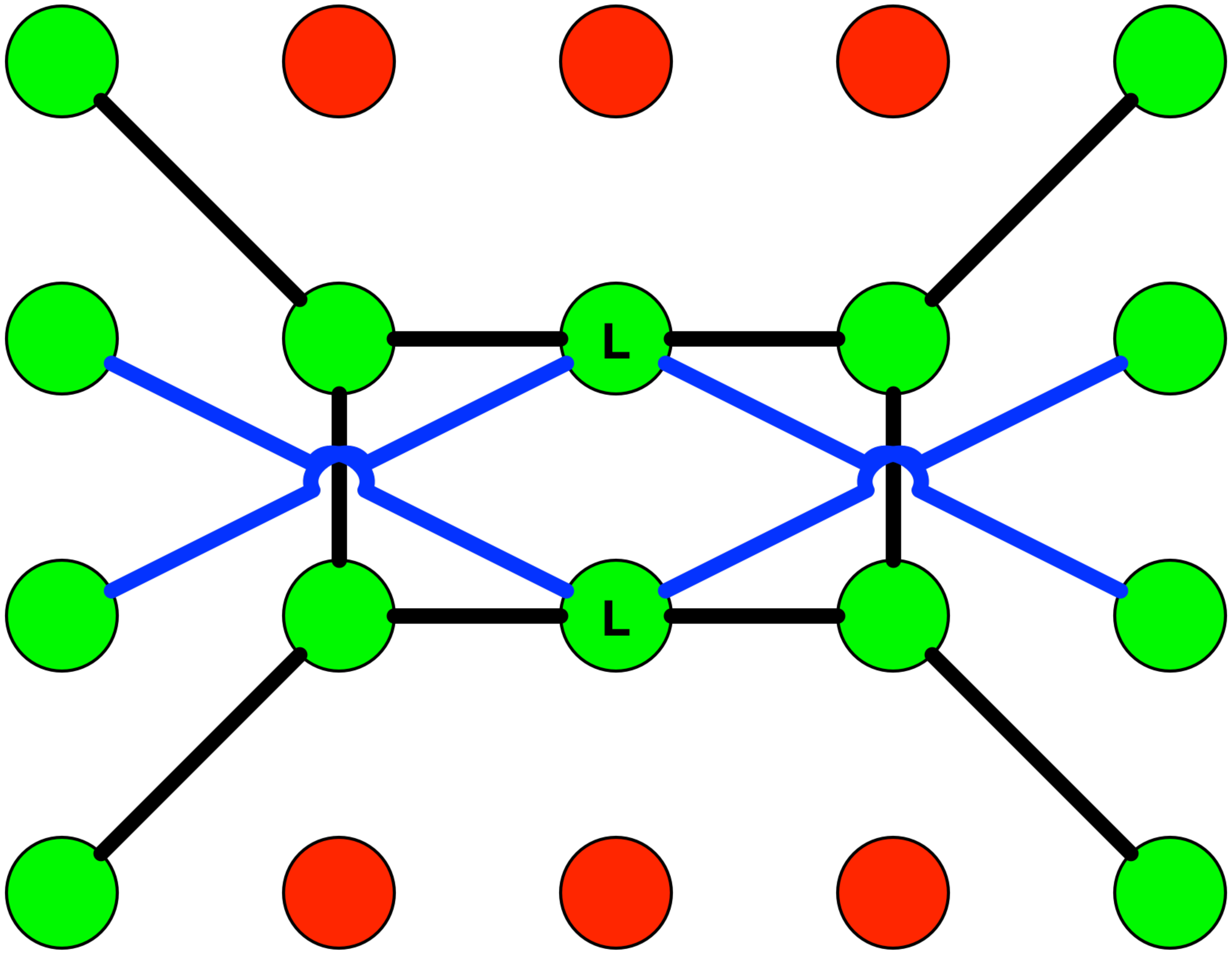


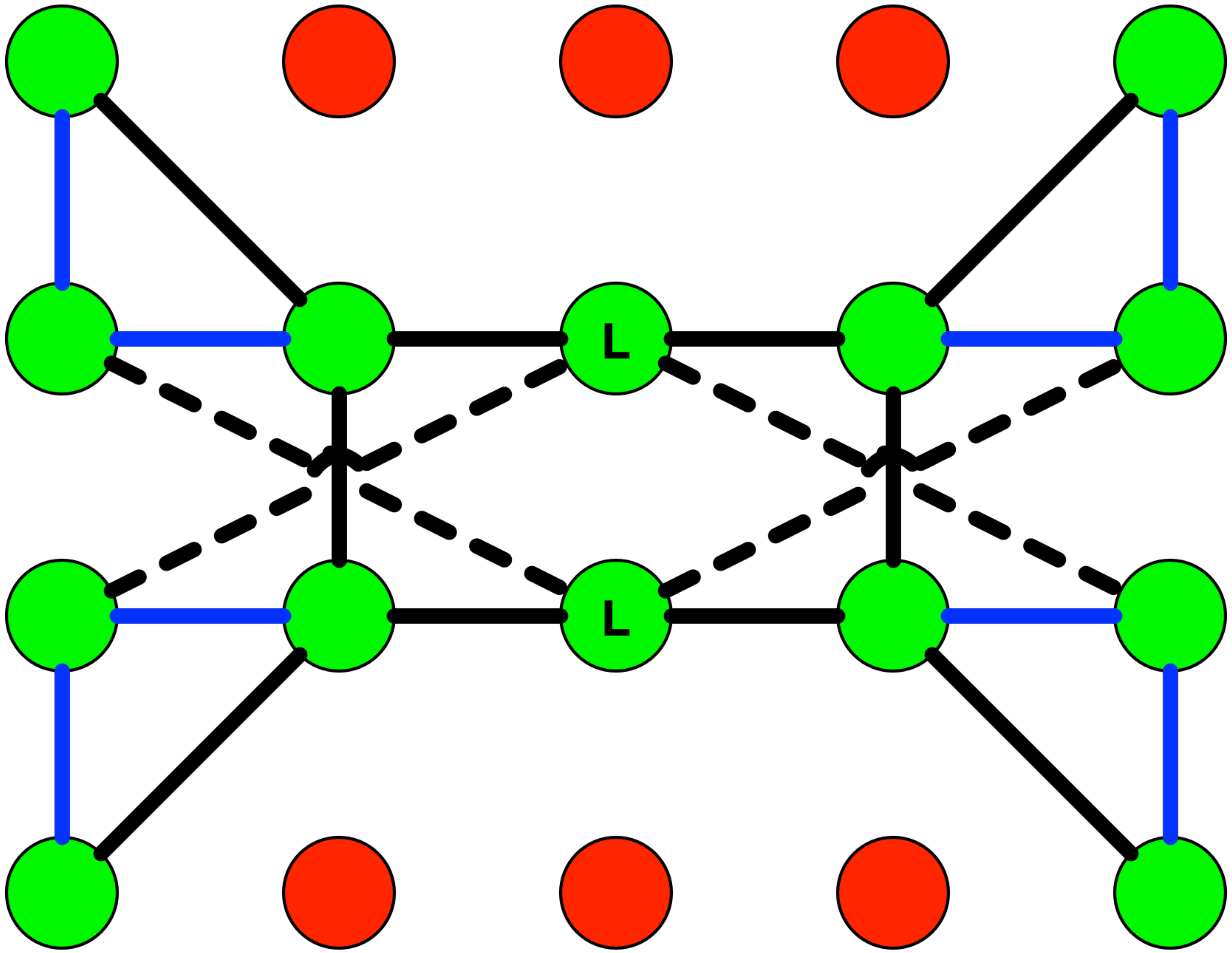

L
L

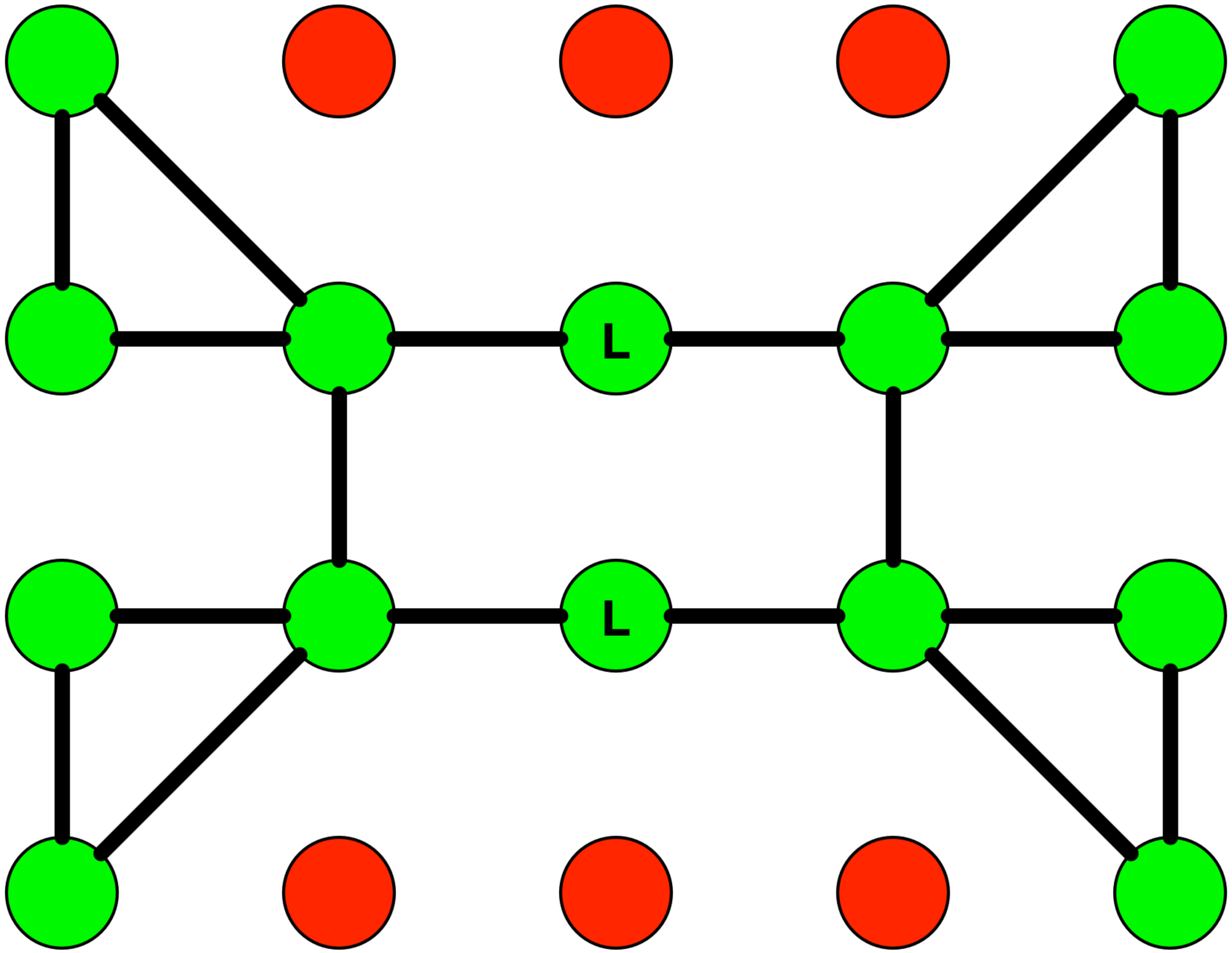

L
L

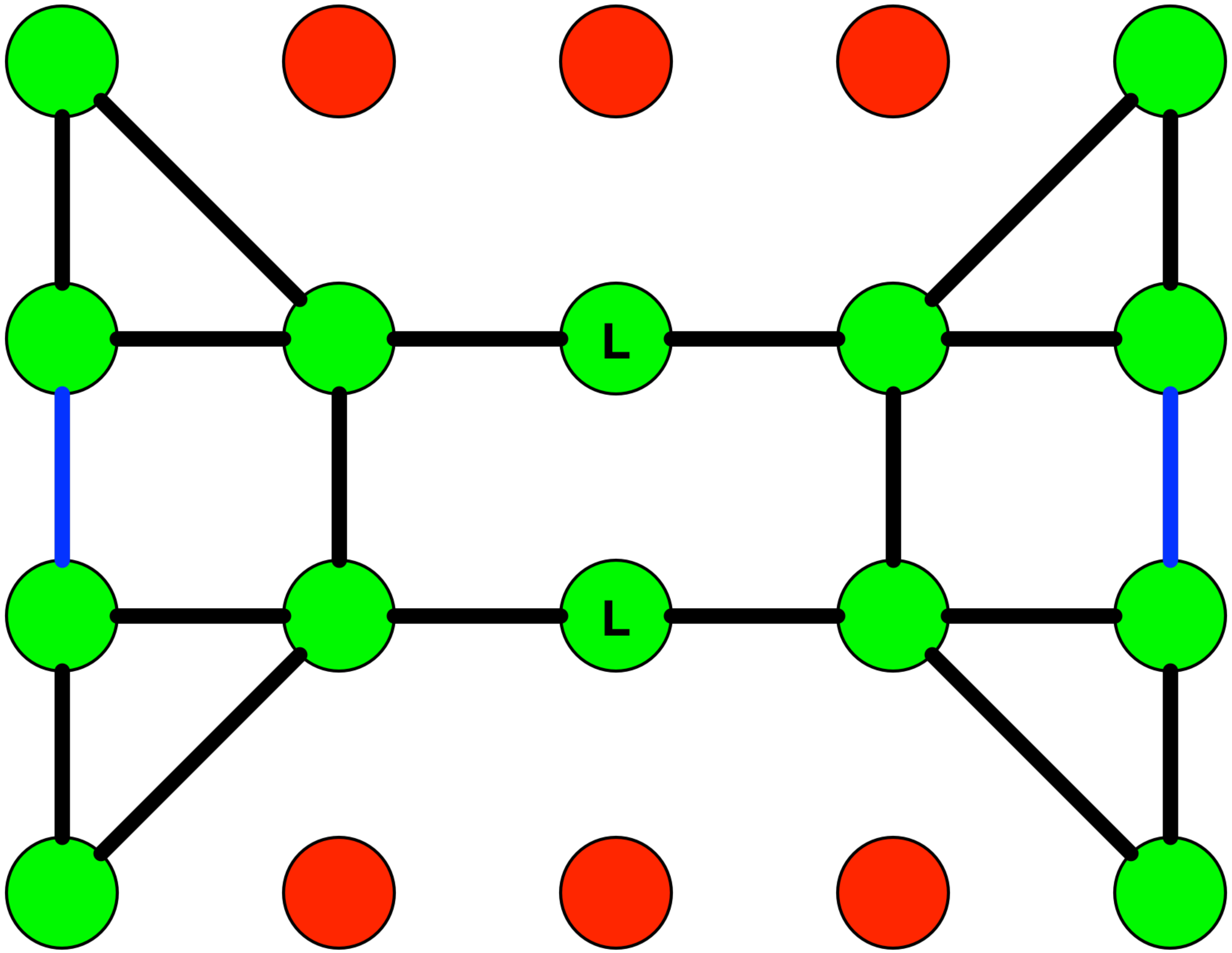

L
L

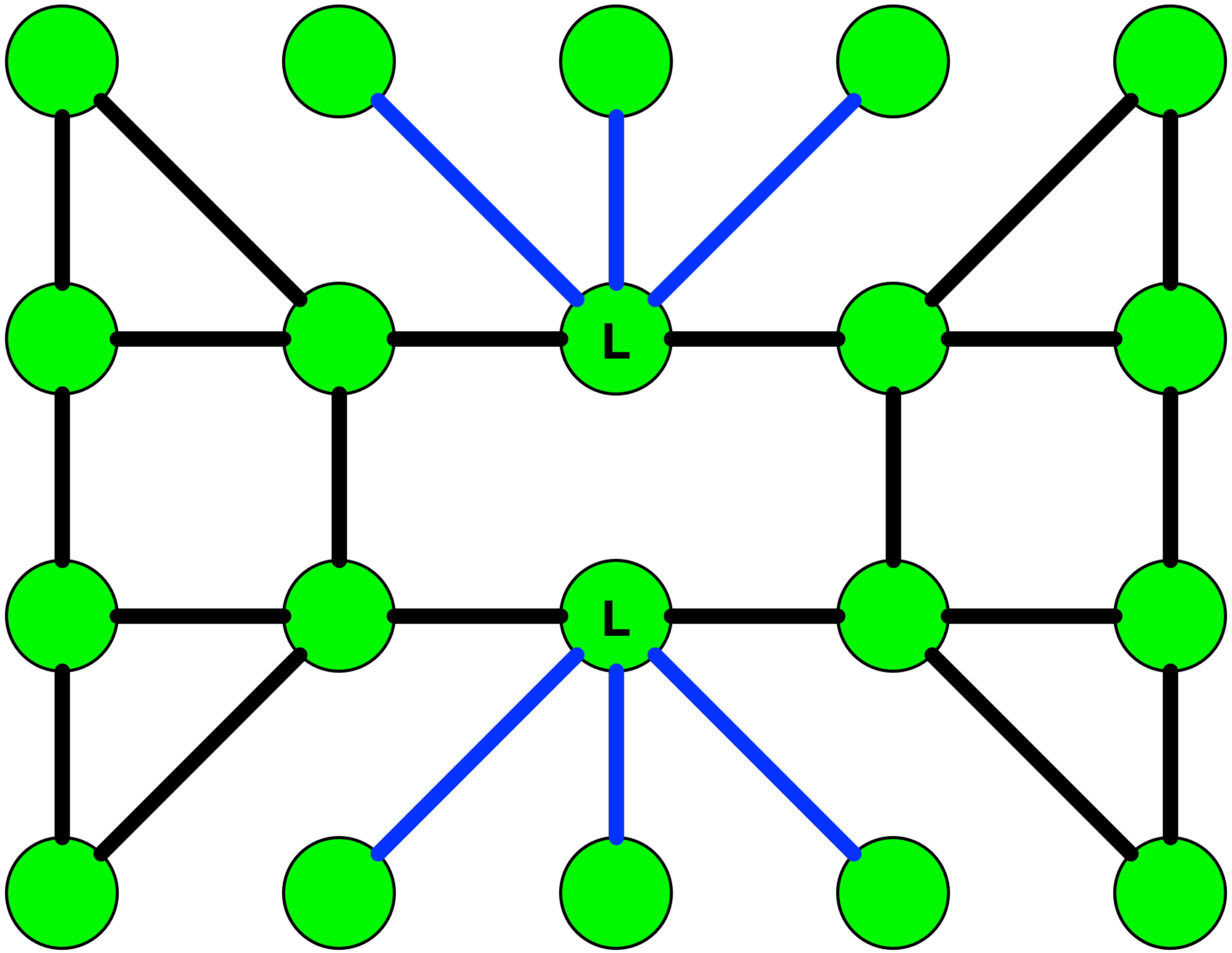

L
L

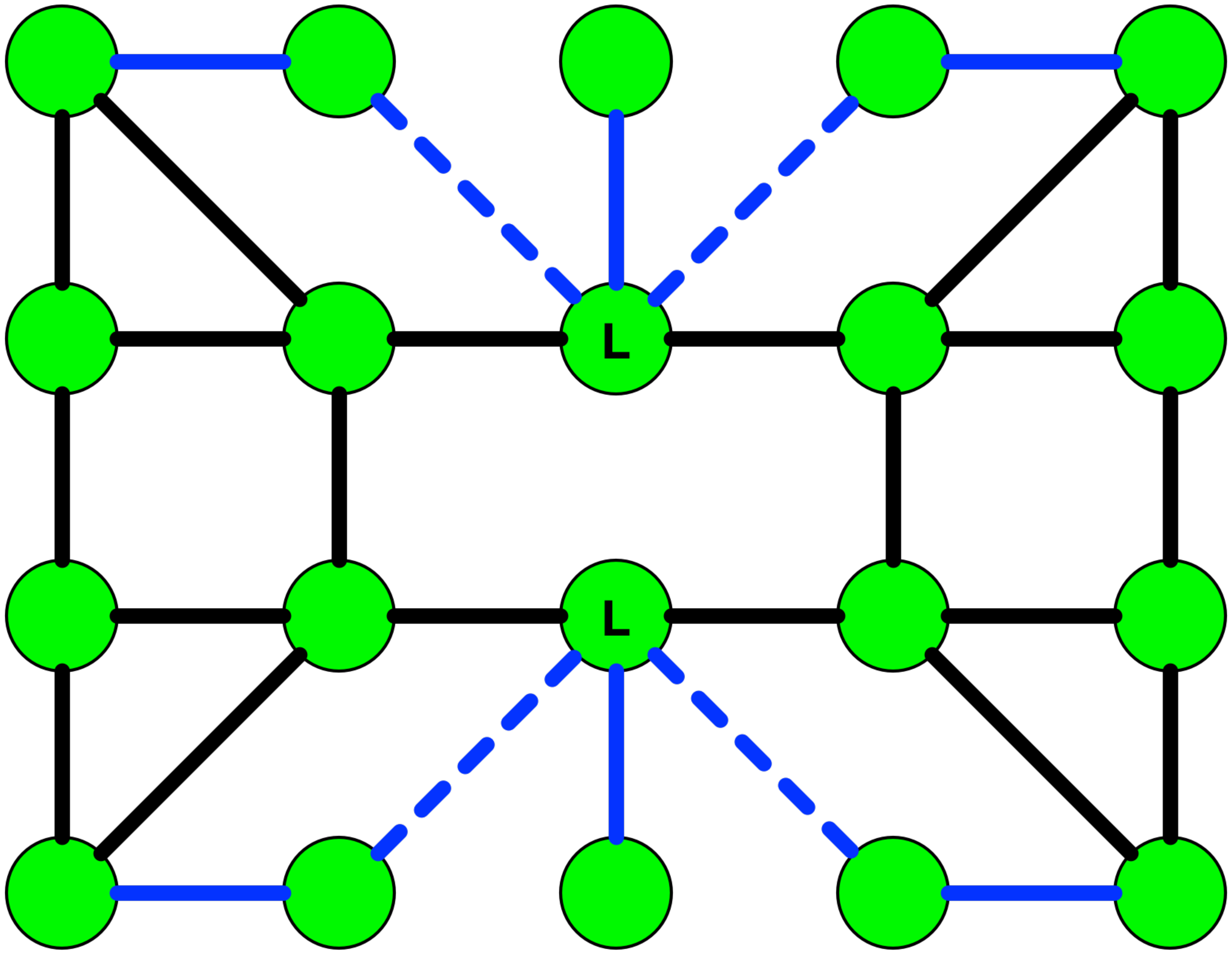

L
L

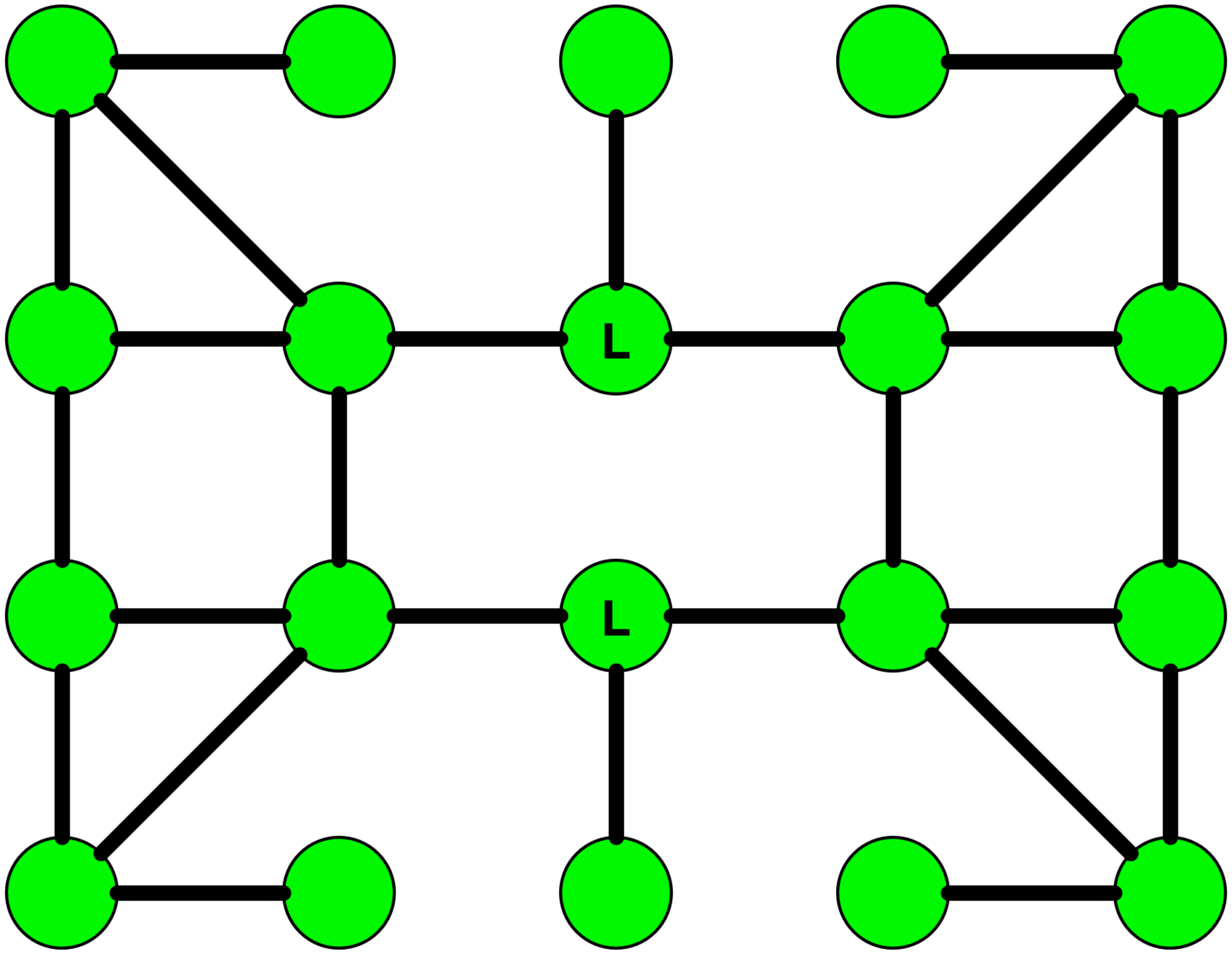

L
L

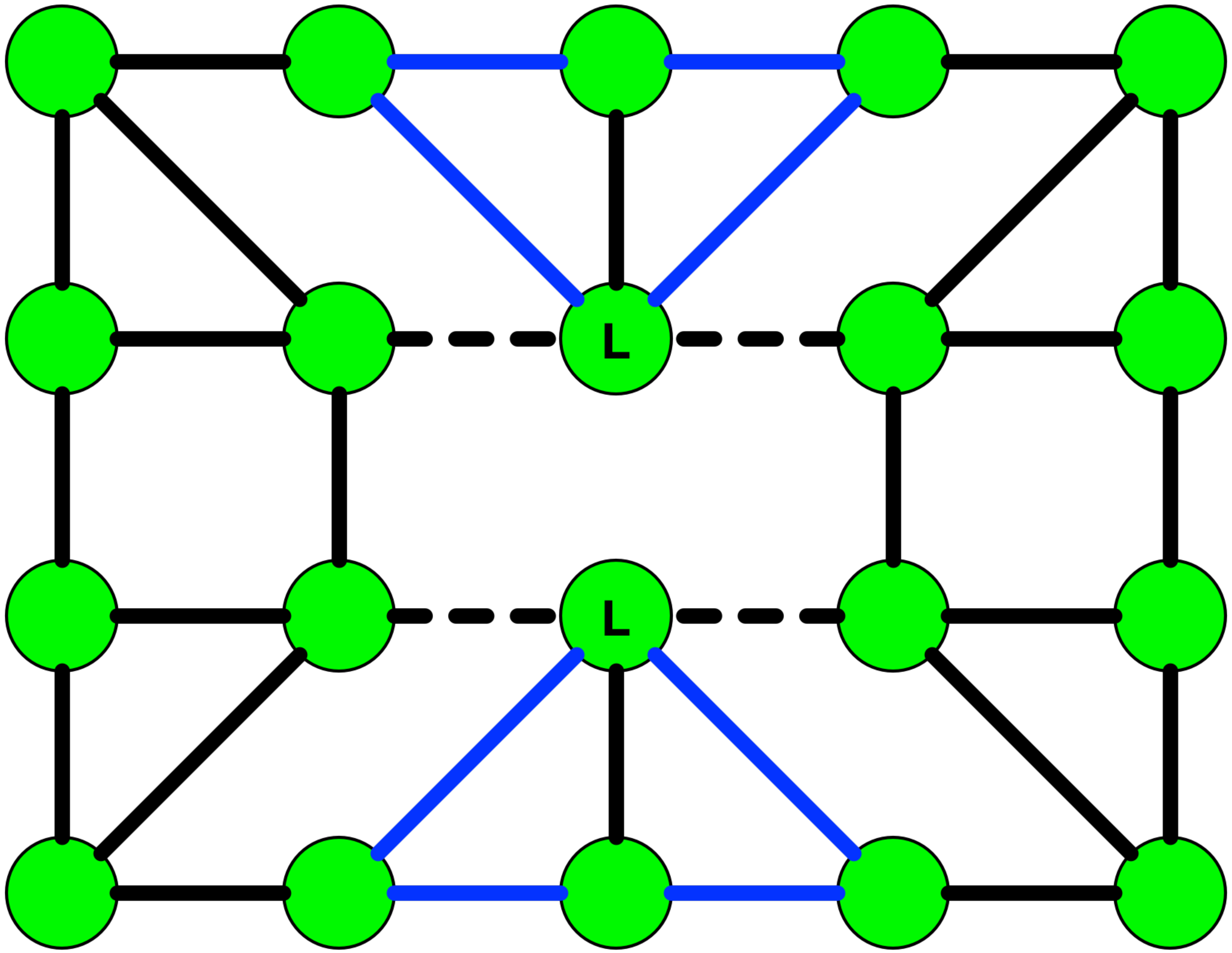

L
L

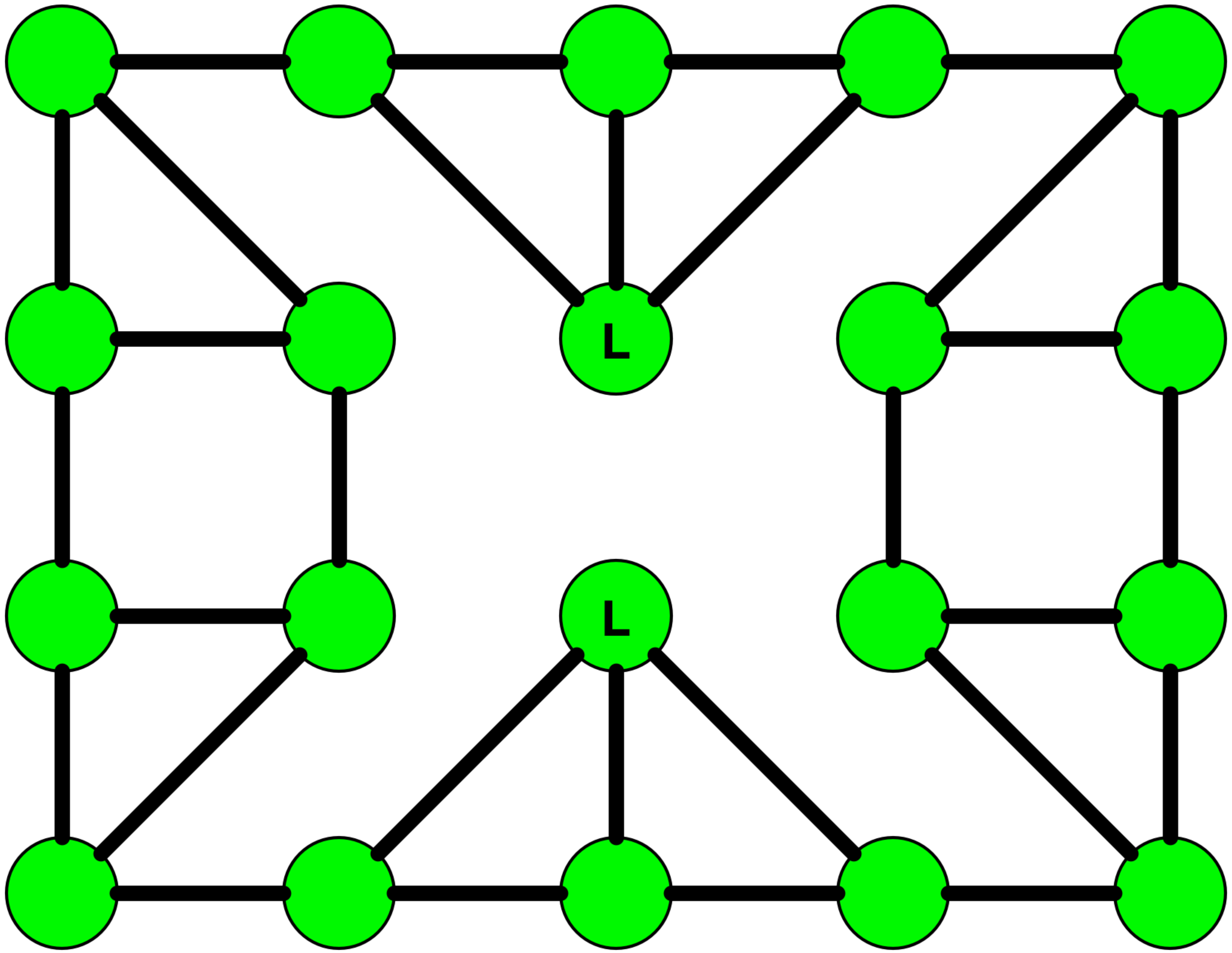

L
L


L
L

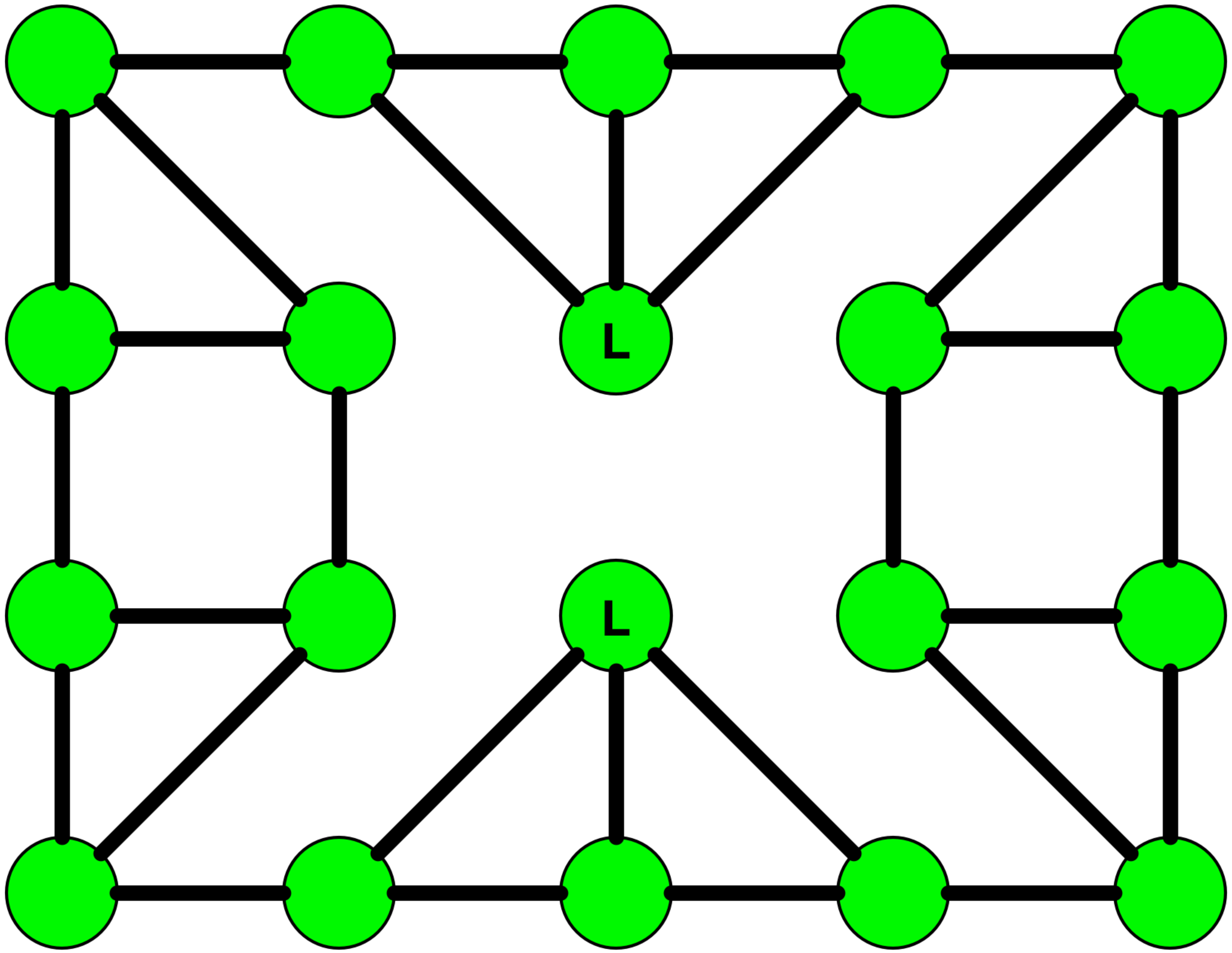

L
L


L
L


L
L


L
L

Connected Graph

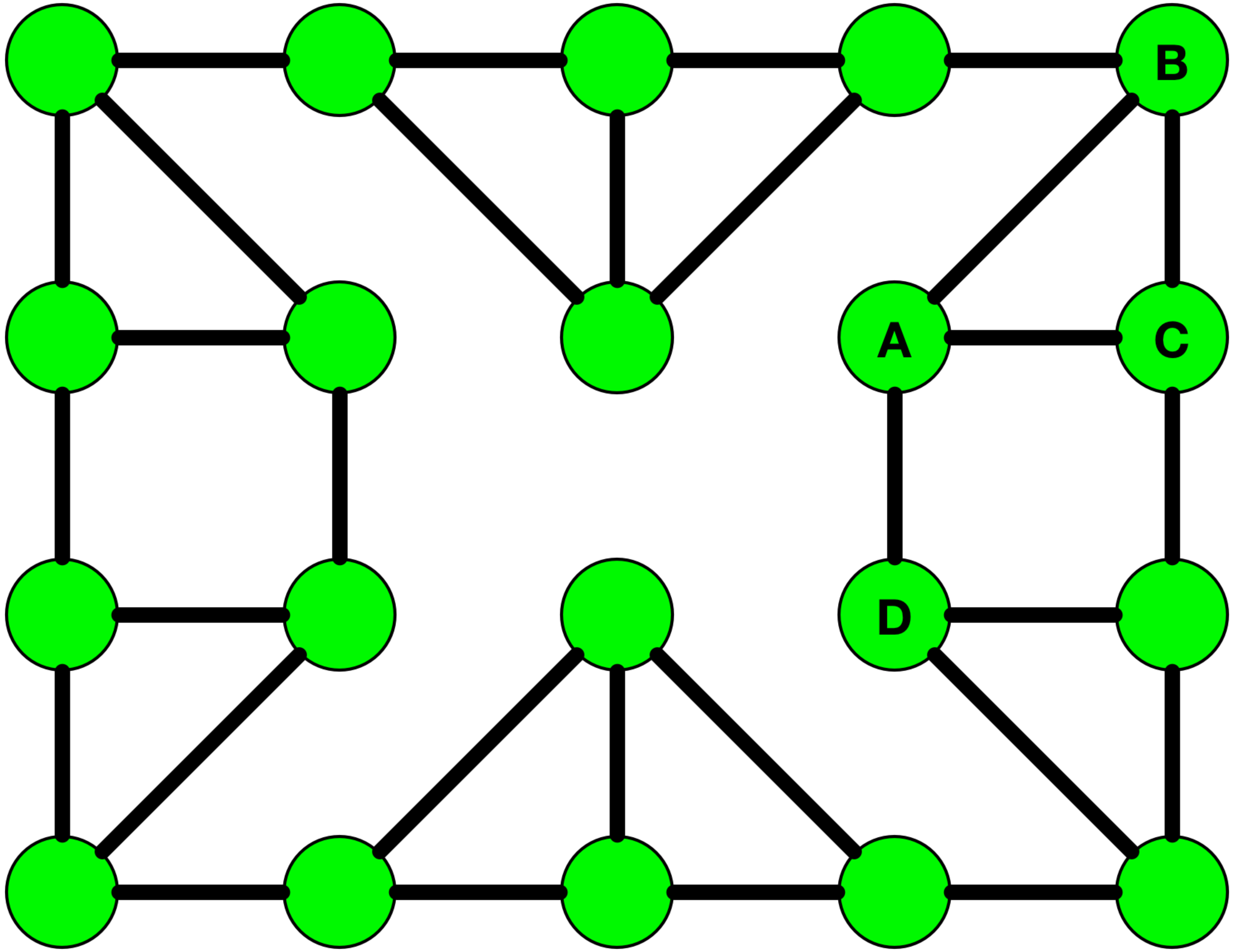


Hyparview

Constant Adaptive Health Checks*

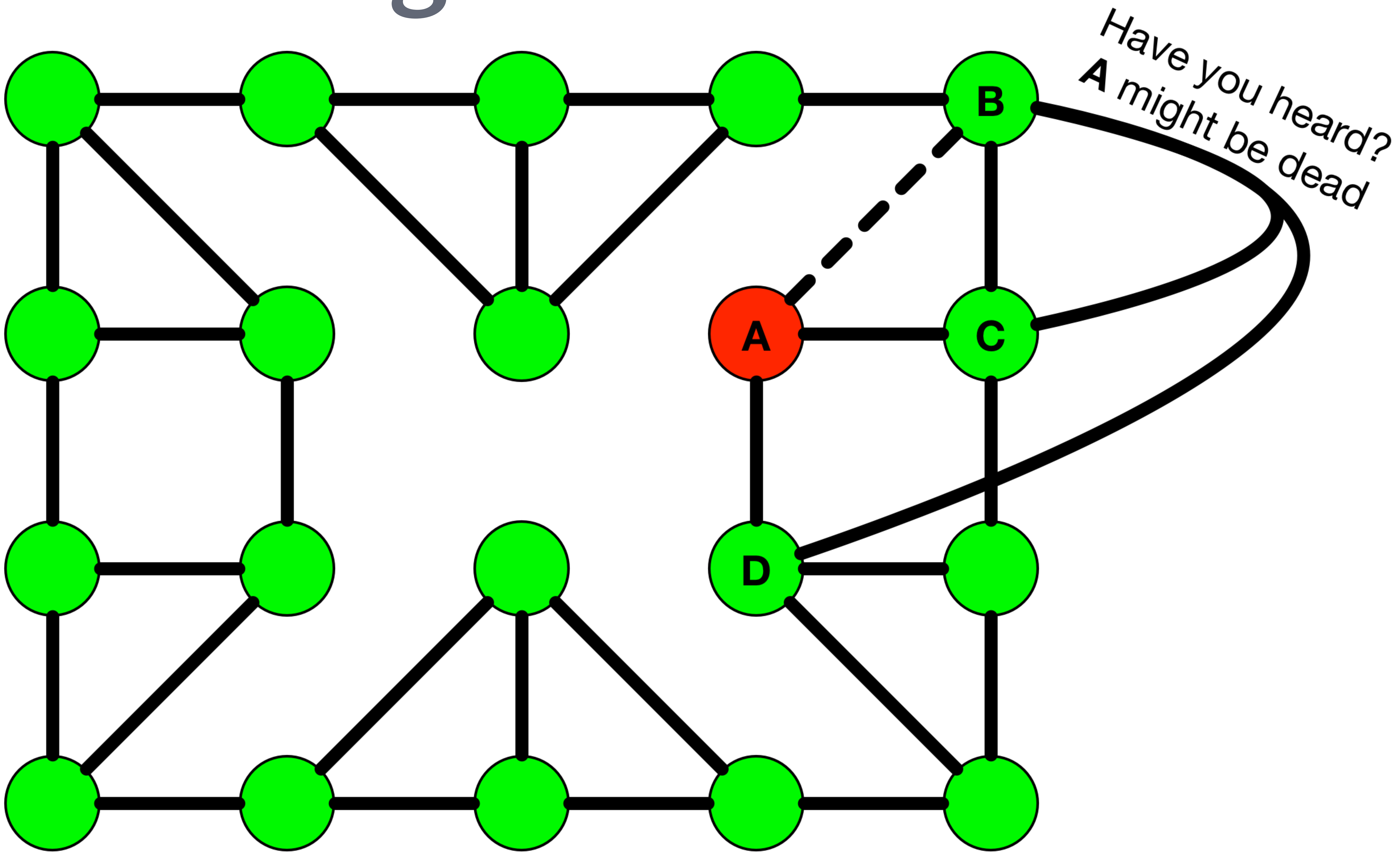
*Borrowed from SWIM, Gossip Style Failure Detector

Dealing with Failure



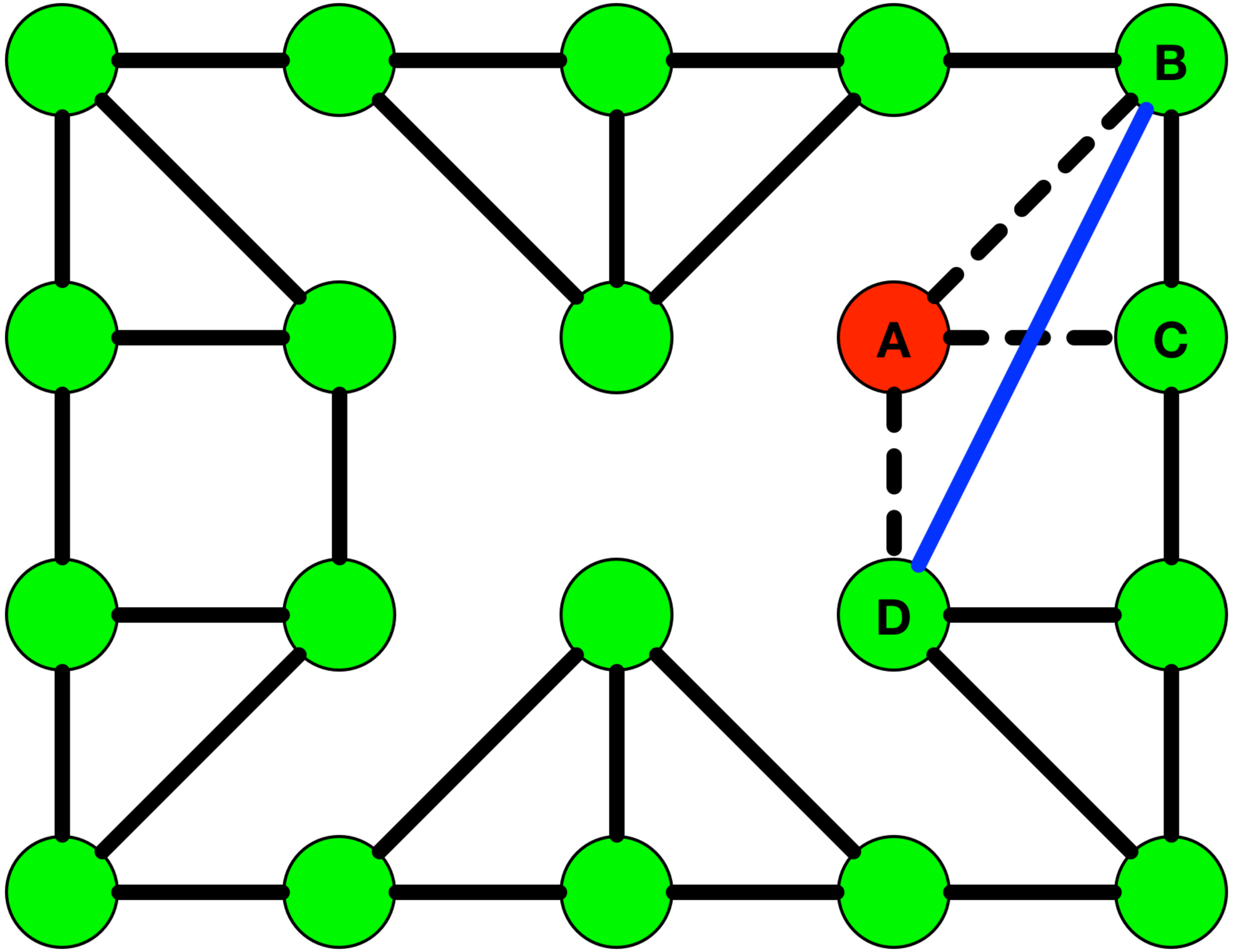
Hyparview

Dealing with Failure



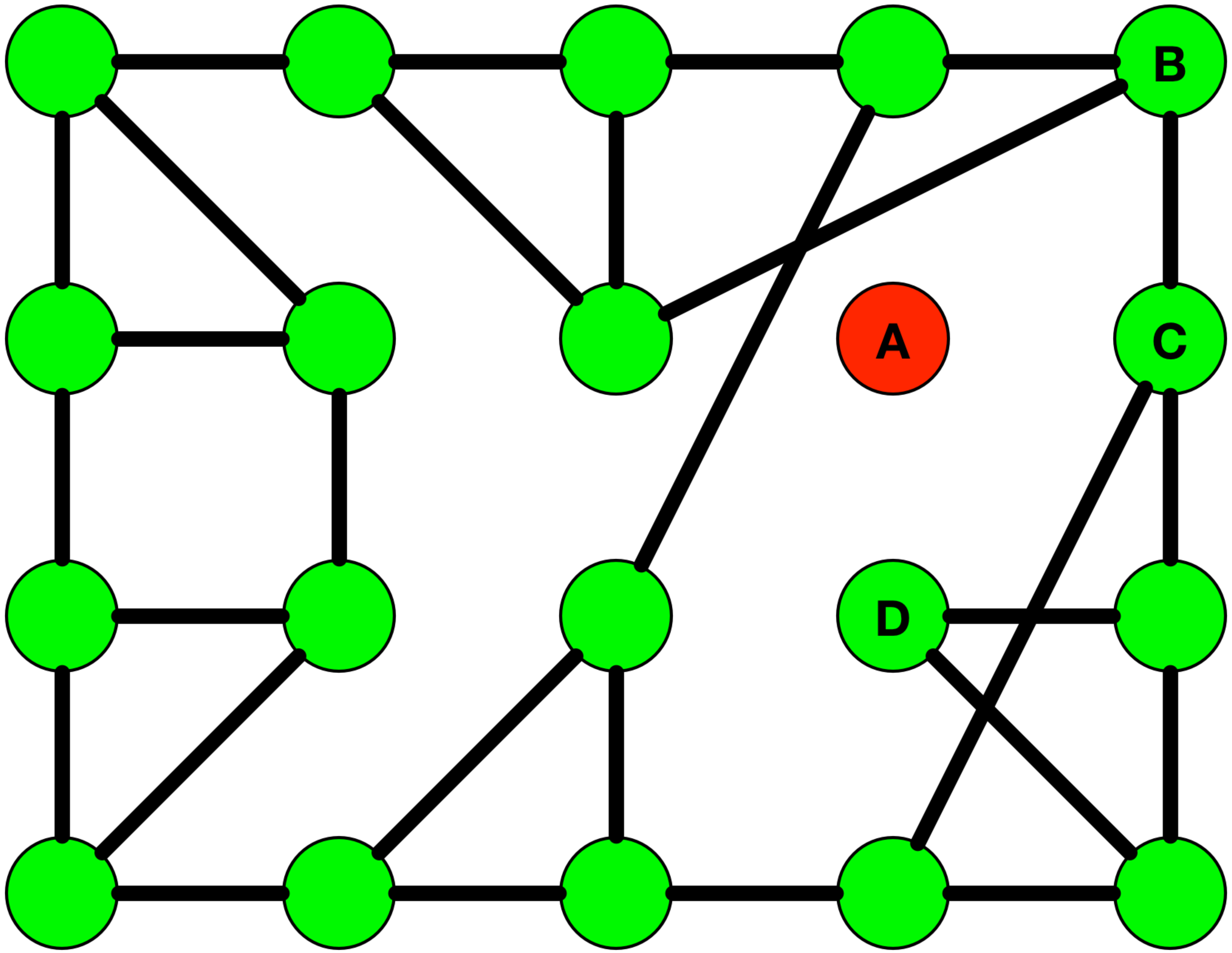
Hyparview

Dealing with Failure



Hyparview

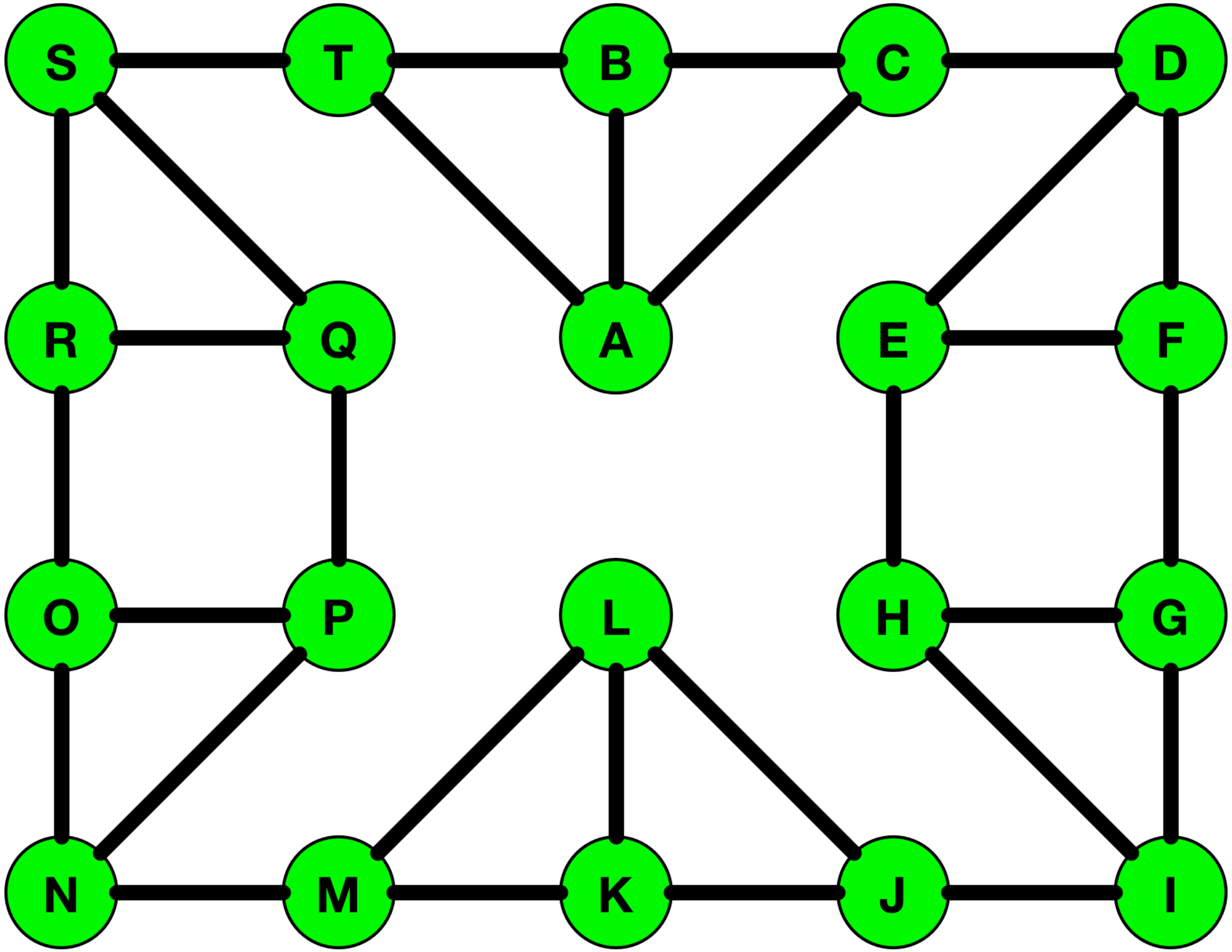
Dealing with Failure

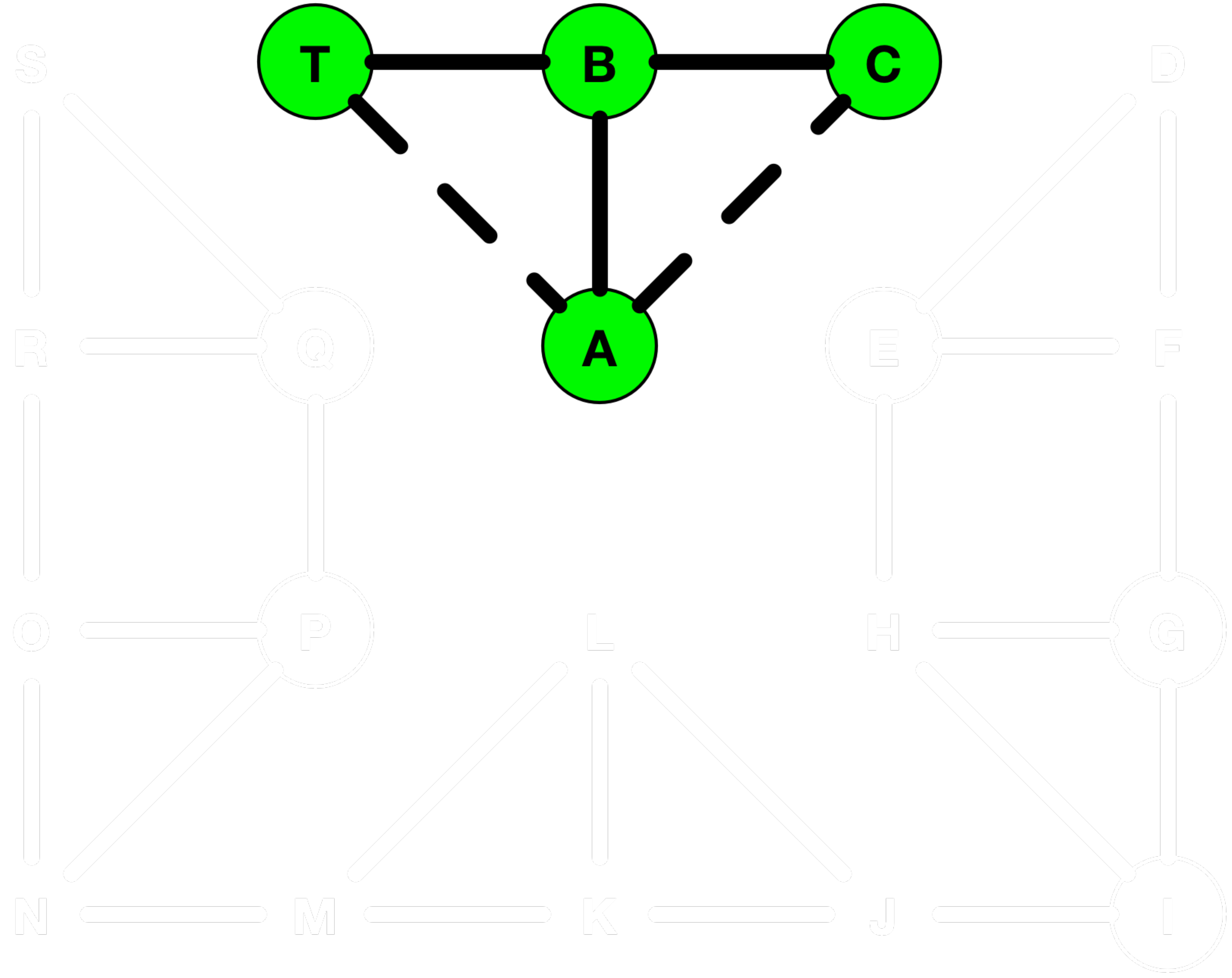


Hyparview

Routing*

*Borrowed from Dijkstra, OSPF, IS-IS

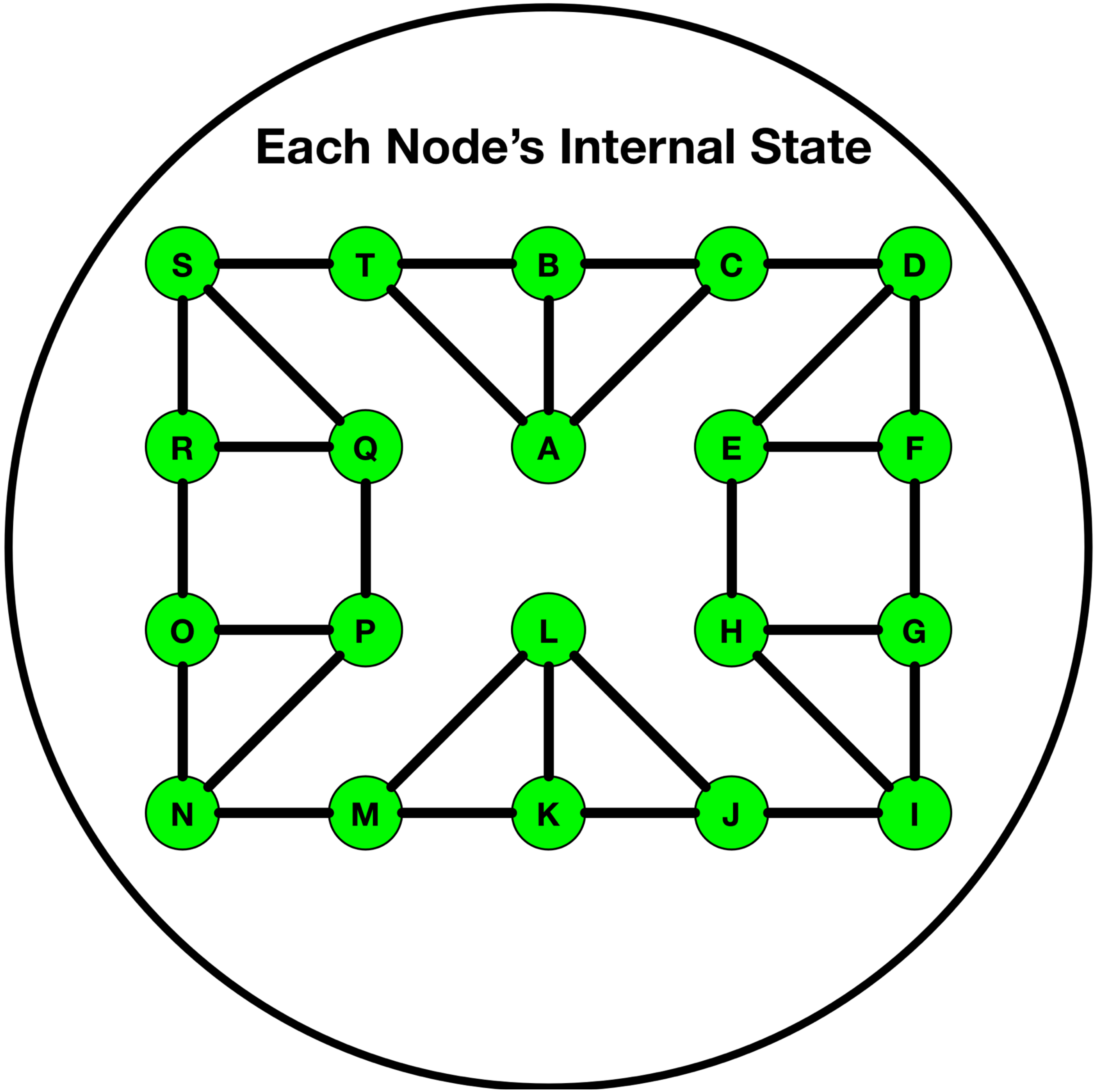

S
T
B
C
D
R
Q
A
E
F
O
P
L
H
G
N
M
K
J
I


T
B
C
A
S
D
R
Q
E
F
O
P
L
H
G
N
M
K
J
I

	Adjacent Node 1	Adjacent Node 2	Adjacent Node 3
A	T	B	C
B	T	A	C
C	B	A	D
D	C	E	F
E	D	F	H
F	D	E	G
G	F	H	I
H	E	G	I
I	G	H	J
J	I	L	K
K	M	L	J
L	M	K	J
M	L	K	N
N	P	O	M
O	R	P	N
P	Q	O	N
Q	S	R	O
R	O	Q	S
S	T	R	Q
T	B	A	S

View Changes Gossiped on Change

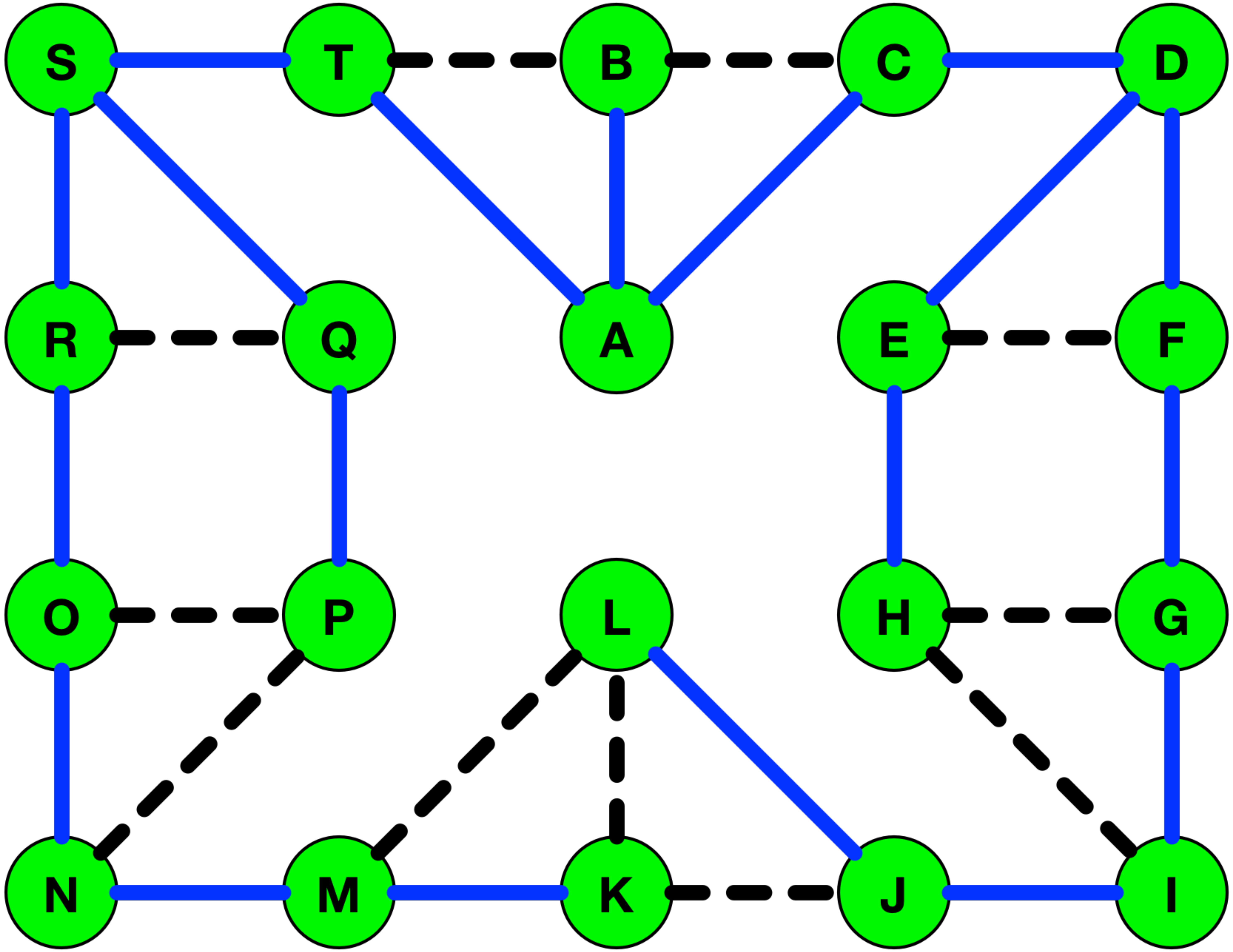
	Adjacent Node 1	Adjacent Node 2	Adjacent Node 3	Version (epoch)
A	T	B	C	1
B	T	A	C	5
C	B	A	D	2
D	C	E	F	7
E	D	F	H	94
F	D	E	G	1
G	F	H	I	415
H	E	G	I	15
I	G	H	J	1
J	I	L	K	3
K	M	L	J	5
L	M	K	J	7
M	L	K	N	88
N	P	O	M	1
O	R	P	N	3
P	Q	O	N	4
Q	S	R	O	6
R	O	Q	S	8
S	T	R	Q	1
T	B	A	S	3


Each Node’s Internal State
S
T
B
C
D
R
Q
A
E
F
O
P
L
H
G
N
M
K
J
I

Just Run Dijkstra*

*BFS / DFS

Minimum Spanning Tree, From Sender A

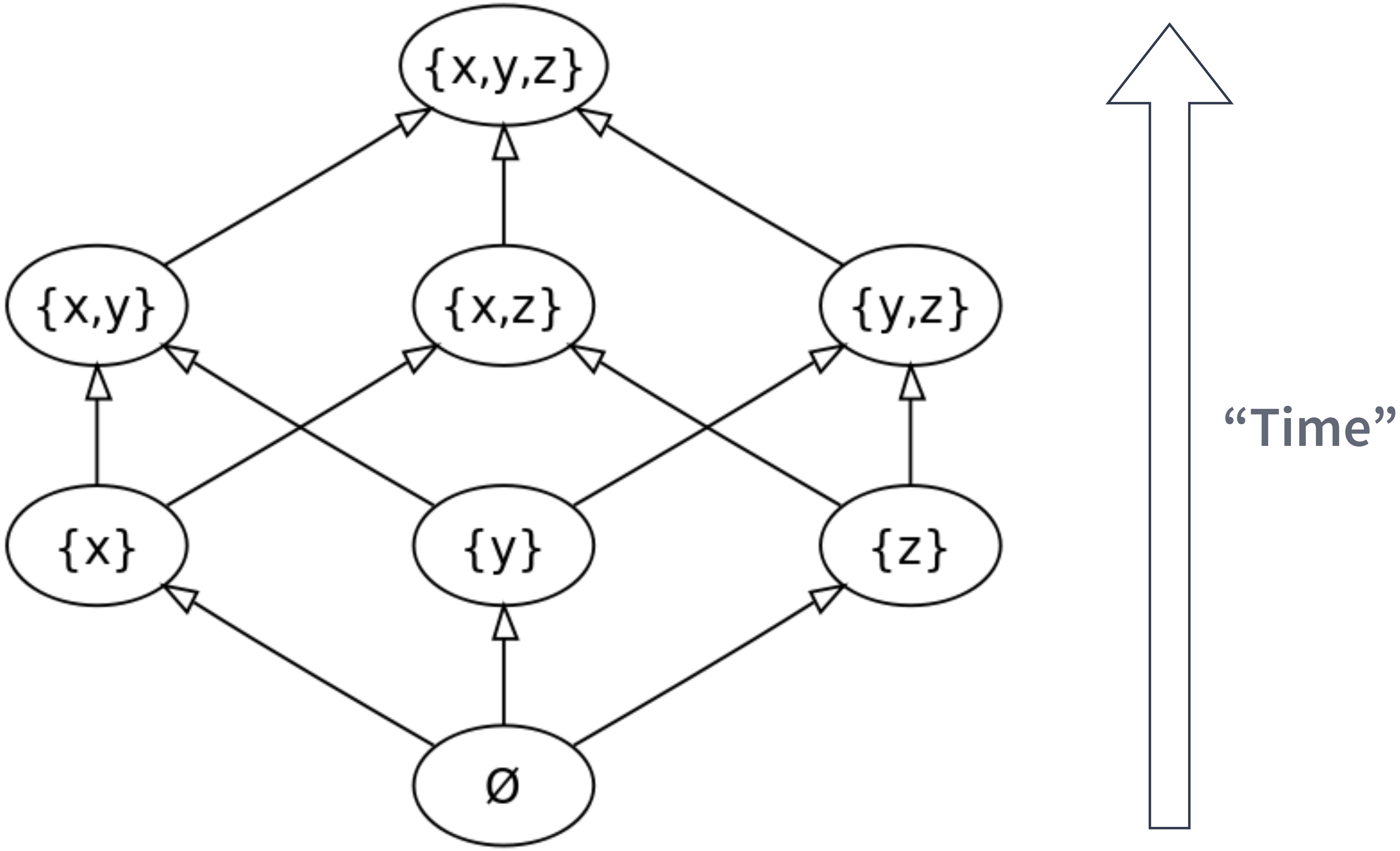


Database

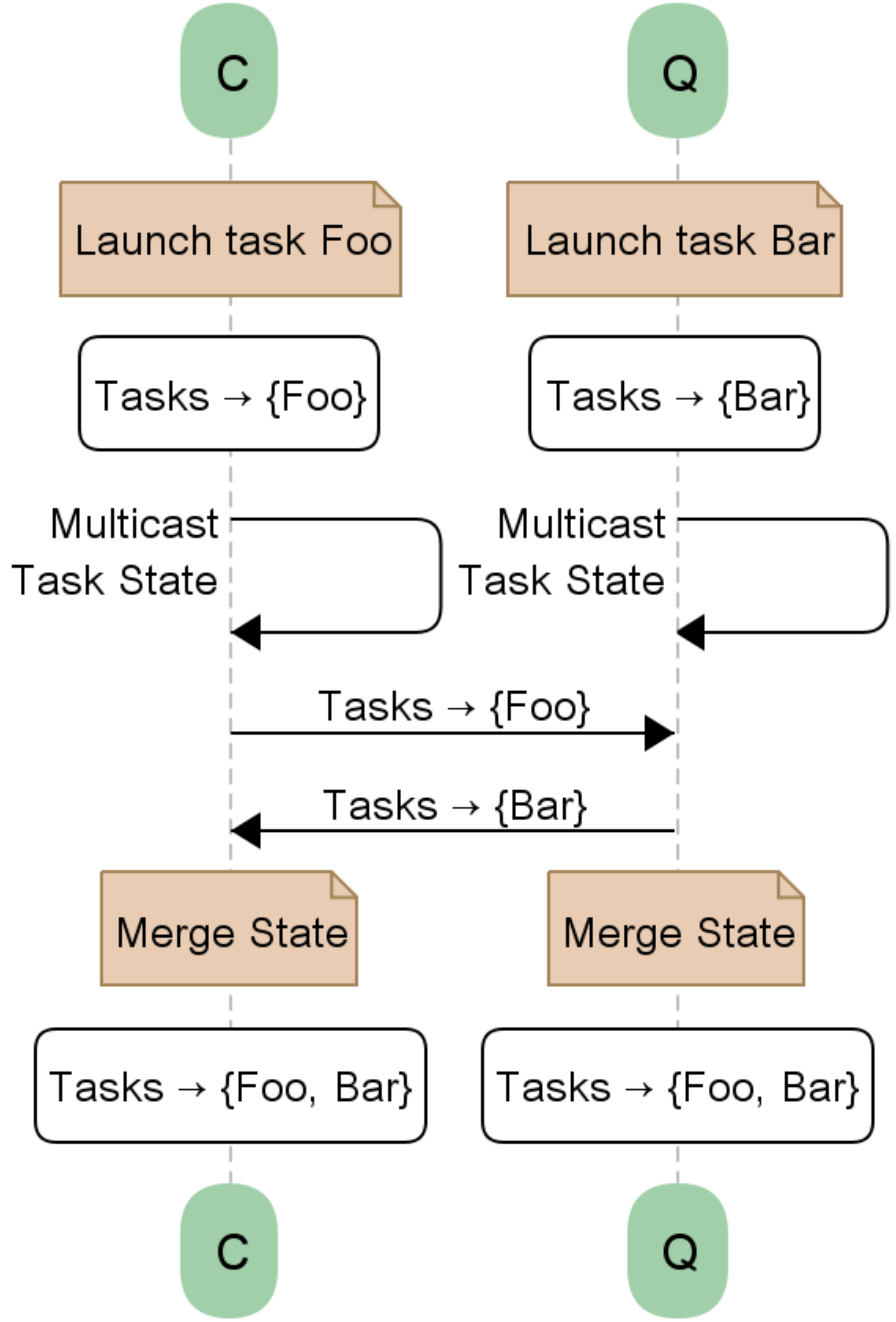
A Wild ~~Pokemon~~ Appears!*

*CRDT papers

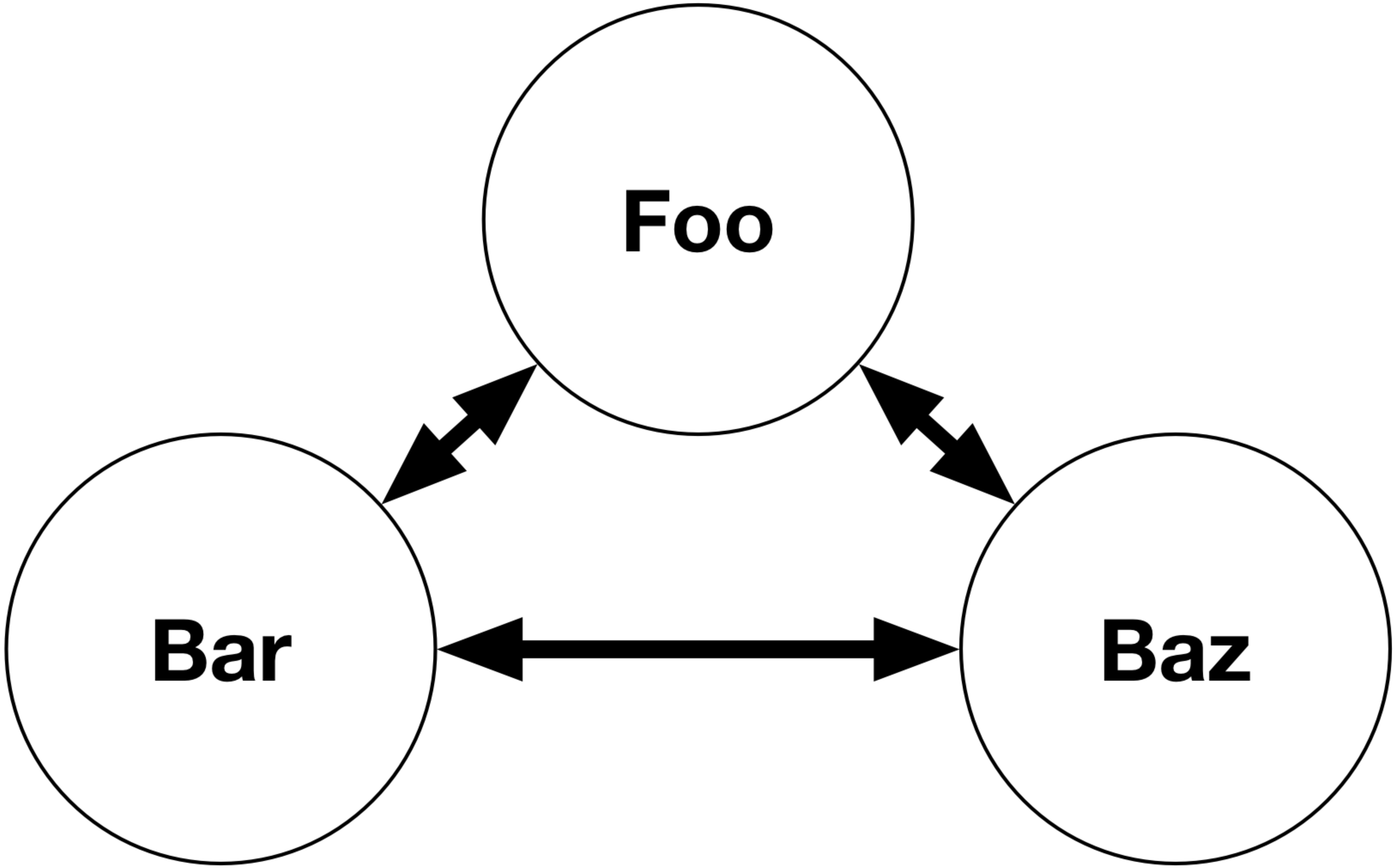
CRDTS: Semilattices



CRDT KV Store



“Demo”



“Demo”

```
(foo@3c075477e55e)10> nodes().
[baz@3c075477e55e,bar@3c075477e55e]
```

```
(bar@3c075477e55e)8> nodes().
[foo@3c075477e55e,baz@3c075477e55e]
```

```
(baz@3c075477e55e)7> nodes().
[foo@3c075477e55e,bar@3c075477e55e]
```

“Demo”

```
(foo@3c075477e55e)9> lashup_kv:value([erlang_users]).
[]
```

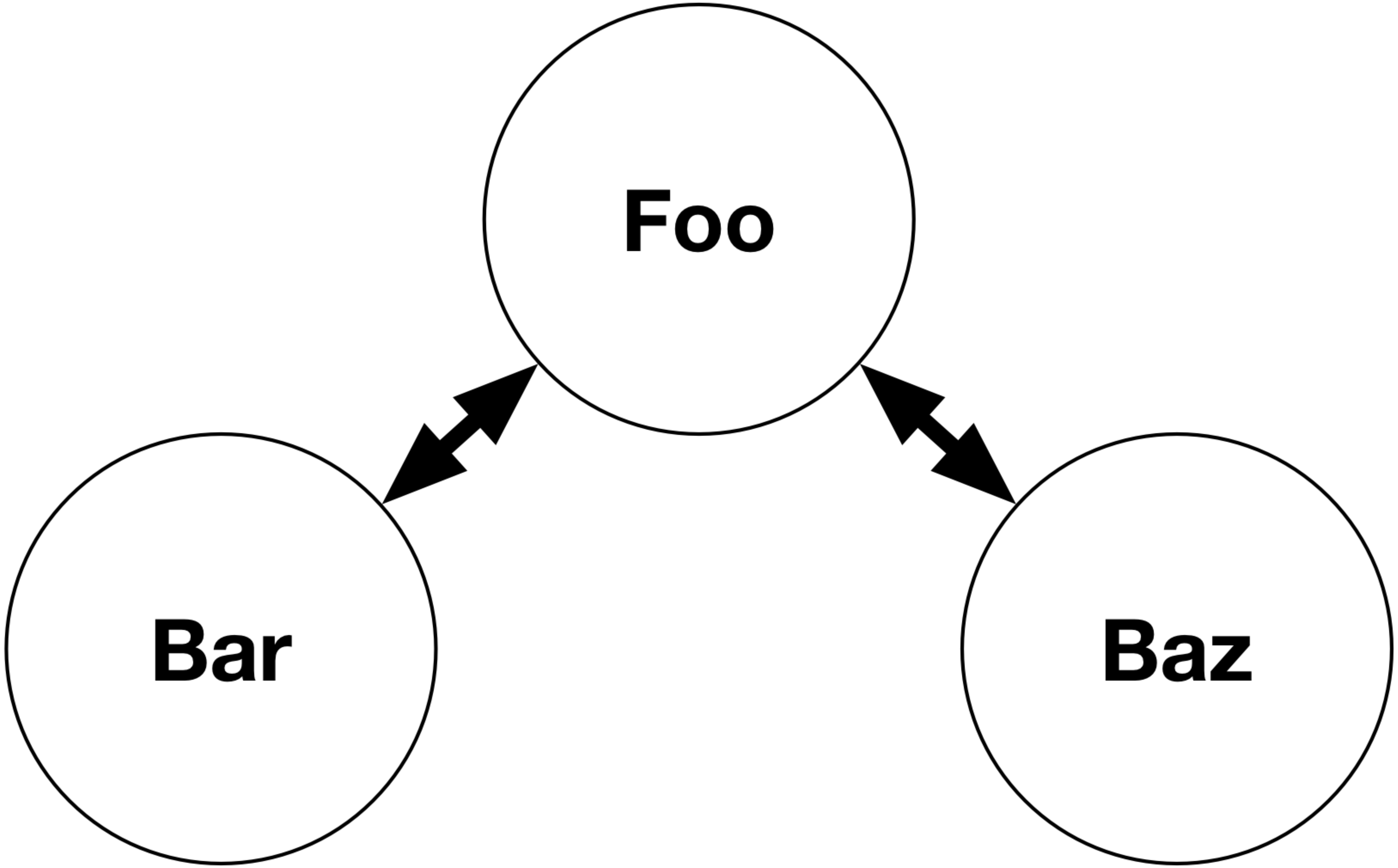
```
(bar@3c075477e55e)9> lashup_kv:value([erlang_users]).
[]
```

```
(baz@3c075477e55e)9> lashup_kv:value([erlang_users]).
[]
```

“Demo”

```
(baz@3c075477e55e)10>
net_kernel:allow([foo@3c075477e55e]).
ok
(baz@3c075477e55e)10>
net_kernel:disconnect('bar@3c075477e55e').
ok.
```

“Demo”



“Demo”

```
(bar@3c075477e55e)11>
lashup_kv:request_op([erlang_users], {update, [{update,
{number, riak_dt_pncounter}, {increment, 5}}]}).
{ok,[{{number,riak_dt_pncounter},5}]}
```

“Demo”

```
(foo@3c075477e55e)12> lashup_kv:value([erlang_users]).
[{{number,riak_dt_pncounter},5}]
```

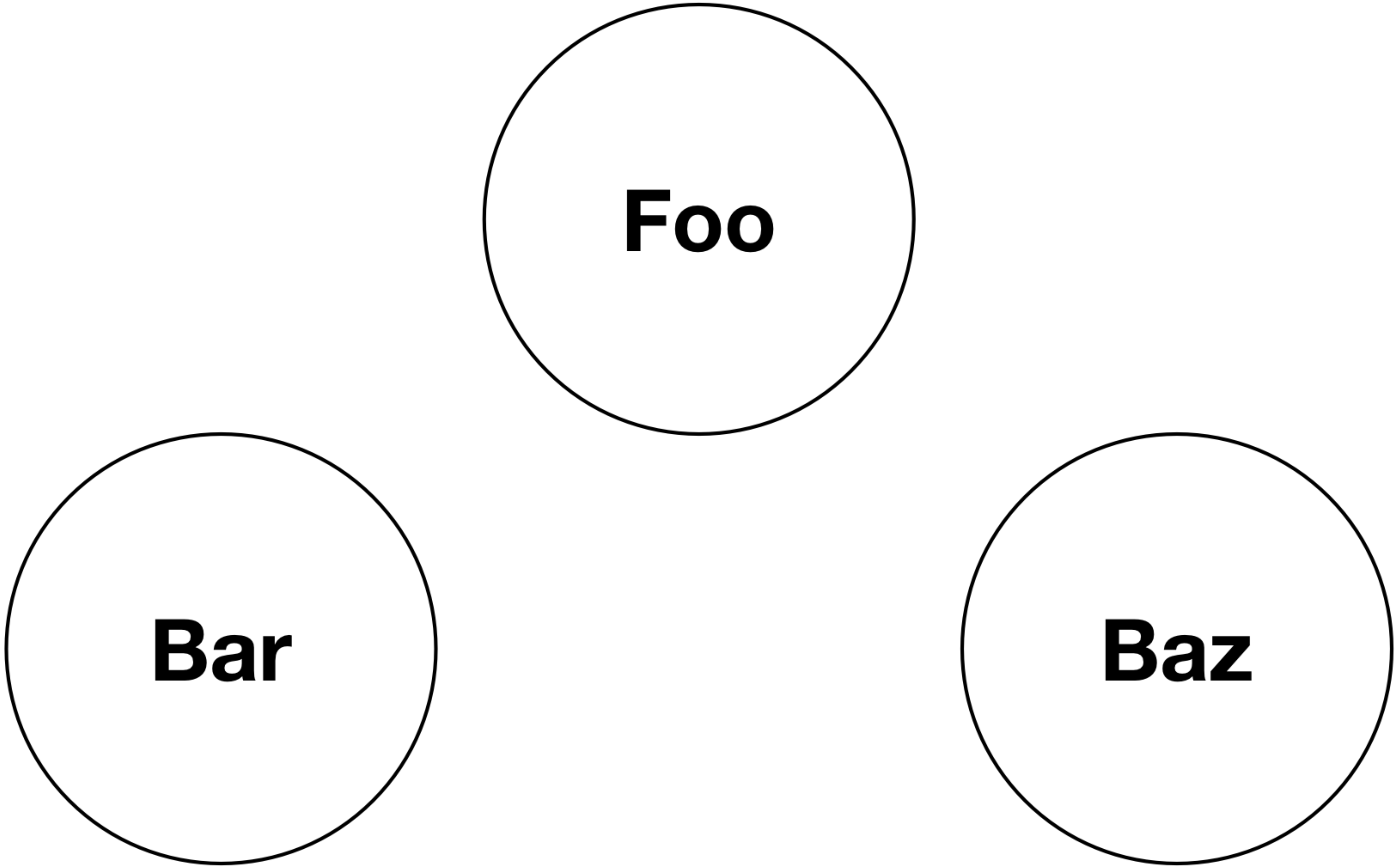
```
(bar@3c075477e55e)13> lashup_kv:value([erlang_users]).
[{{number,riak_dt_pncounter},5}]
```

```
(baz@3c075477e55e)15> lashup_kv:value([erlang_users]).
[{{number,riak_dt_pncounter},5}]
```

“Demo”

```
(foo@3c075477e55e)23> erlang:set_cookie(baz@3c075477e55e,
fake).
true
(foo@3c075477e55e)24> erlang:set_cookie(bar@3c075477e55e,
fake).
true
```

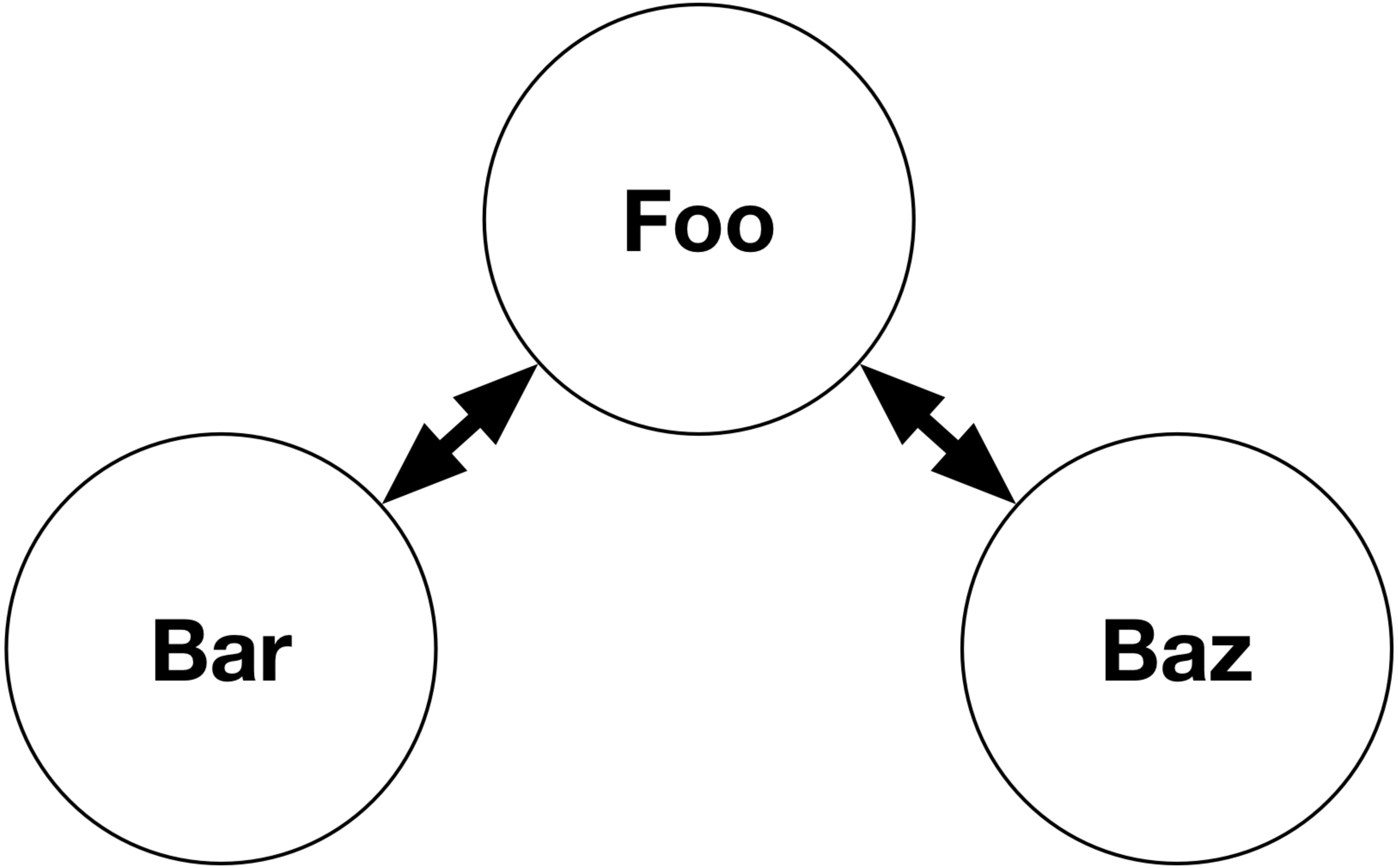
“Demo”



“Demo”

```
(foo@3c075477e55e)29> lashup_kv:request_op([erlang_users],
{update, [{update, {number, riak_dt_pncounter},
{increment, 3}}]}).
{ok,[{{number,riak_dt_pncounter},8}]}
(bar@3c075477e55e)17> lashup_kv:request_op([erlang_users],
{update, [{update, {number, riak_dt_pncounter},
{increment, 7}}]}).
{ok,[{{number,riak_dt_pncounter},12}]}
(baz@3c075477e55e)19> lashup_kv:request_op([erlang_users],
{update, [{update, {number, riak_dt_pncounter},
{increment, 11}}]}).
{ok,[{{number,riak_dt_pncounter},16}]}
```

“Demo”



“Demo”

```
(foo@3c075477e55e)1> lashup_kv:value([erlang_users]).
[{{number,riak_dt_pncounter},26}]
```

```
(bar@3c075477e55e)25> lashup_kv:value([erlang_users]).
[{{number,riak_dt_pncounter},26}]
```

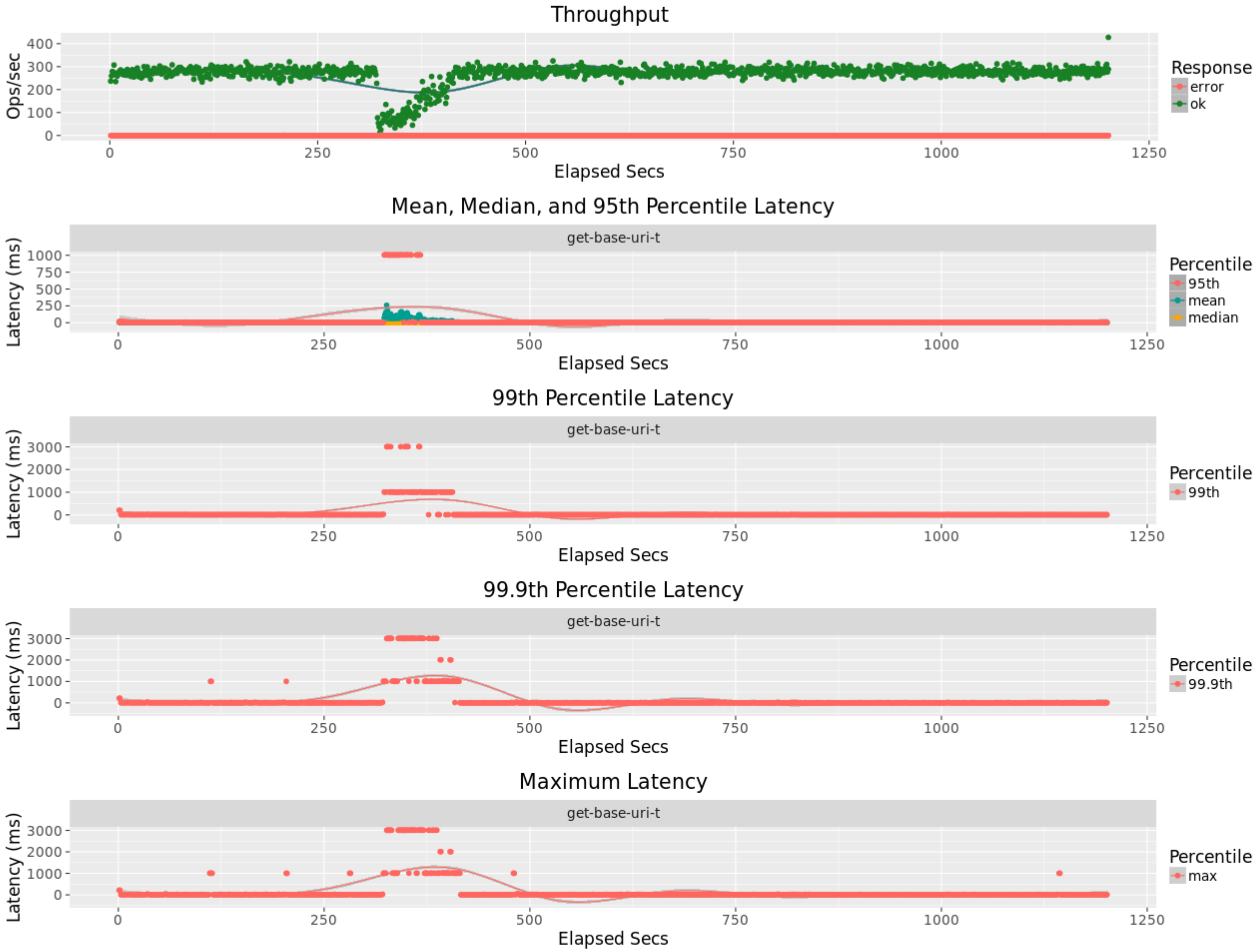
```
(baz@3c075477e55e)23> lashup_kv:value([erlang_users]).
[{{number,riak_dt_pncounter},26}]
```

LASHUP KV

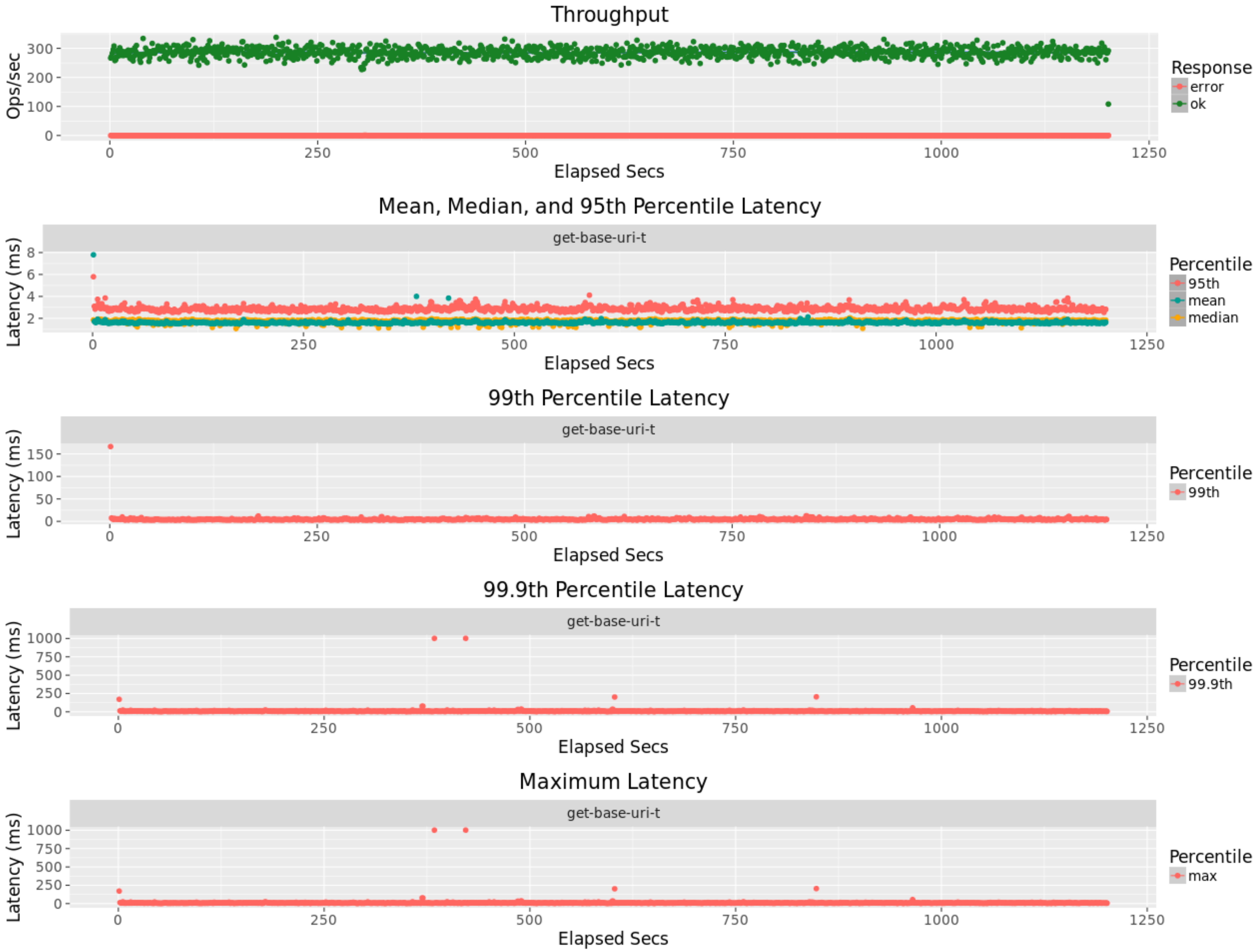
- Datatypes:
  - Maps
    - Composable
  - Sets
  - Flags
  - Counters
  - Registers
    - Last-write-wins

Does Lashup provide Business Value?

Prior to Lashup



After Lashup



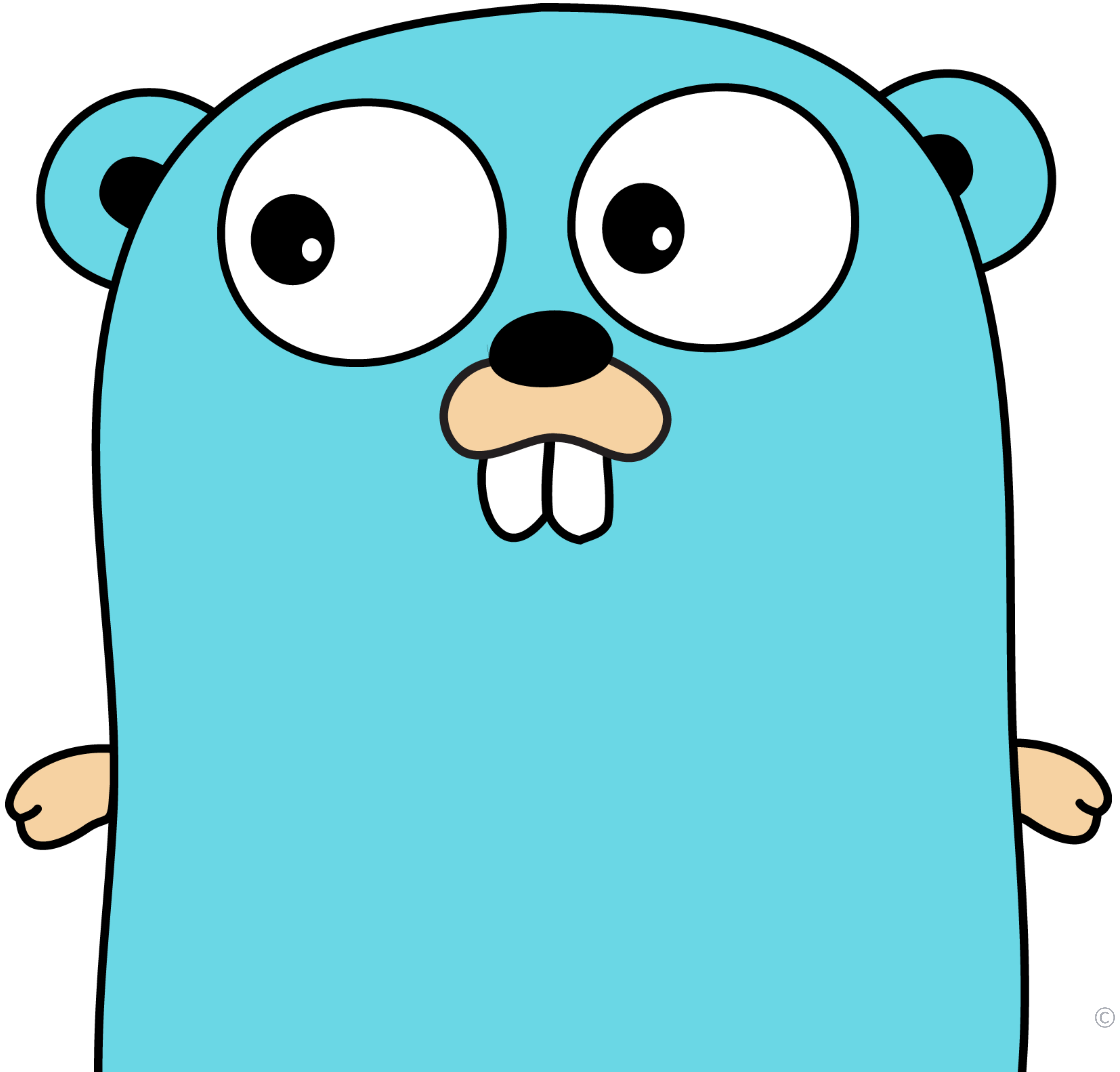
PROJECT LASHUP

- A novel distributed systems SDK that provides:
  - Membership
  - Multicast Delivery
  - Strongly-eventually consistent data storage
- Powers:
  - Minuteman VIP dissemination
  - Minuteman node liveness checks
  - Overlay routing
  - DNS Synchronization
- Open source: github.com/dcos/lashup

So,
Why Erlang/OTP?

Why Not?

(Go)



The First Version of Minuteman Was In Golang

Stand on the Shoulders of Giants

What's a good language with a healthy set of distributed systems, and networking libraries?

What’s a good language with a healthy set of distributed systems, and networking libraries?

With an ecosystem

What’s a good language with a healthy set of distributed systems, and networking libraries?

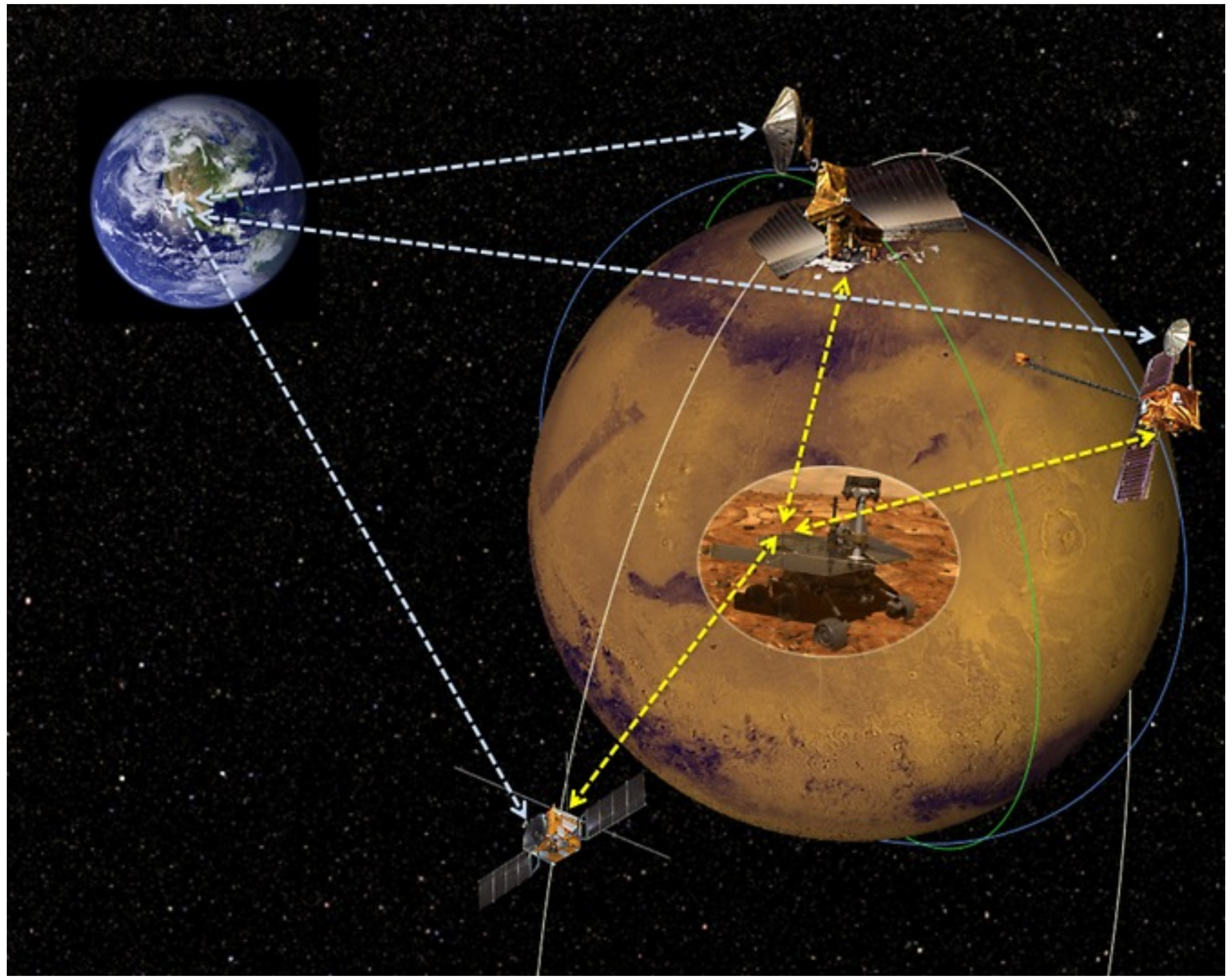
With an ecosystem

That’s battle tested

It’s a 3AM, who do you want picking up the phone?

Answer:

You make it so that your software so it doesn’t page you at 3 AM.



Upgrading Software on Mars is Easier

vs.

On-Prem

Erlang

LASHUP KV NEEDED A DURABLE STORE

ANSWER: MNESIA

- ACID, Transactional
- 20+ years old
- Battle Tested
- Comes built-in
- Software Transaction Memory

LASHUP KV NEEDED A CRDT LIBRARY

ANSWER: RIAK_DT

- Riak's CRDT Library
- Used in production by others
- Battle Tested
- Property tested

LASHUP NEEDED SECURE DISTRIBUTION

ANSWER: DISTERL

- SSL with mutual authentication is a matter of configuration
- Used in production by others
- Battle Tested
- Comes Built-in

SPARTAN NEEDED A DNS SERVER

ANSWER: ERL-DNS

- Runs DNSimple
- Used in production
- Battle Tested
- Some Statistics from DNSimple
  - Serves 20K+ domains
  - Does 20k queries/sec+

MINUTEMAN NEEDED A KERNEL API LIBRARY

ANSWER: GEN_NETLINK

- Liberal license
- Deployed by TravelPing
- Used in production
- Battle Tested

LASHUP NEEDED A GRAPH LIBRARY

ANSWER: DIGRAPH

- Built-in Library to Erlang/OTP
- Correctness verified
- Simple API
- Downsides:
  - Slow
  - Heavy

So, we built our own.

On Testing…

We verified lashup_gm_route against digraph

PropEr

A QuickCheck-Inspired Property-Based Testing Tool for Erlang

Property-Based Testing: PropEr

Automatically Generate Command Set

$C_{Set} = \{add\_node(e), add\_node(b), add\_edge(e,b), \ldots\}$

Verify Equality

Digraph		lashup_gm
add_node(e)		add_node(e)
	Check Equivalency	
add_node(b)		add_node(b)
	Check Equivalency	
add_edge(e, b)		add_edge(e, b)
	Check Equivalency	

…Tens of Thousands of Times

Common Test Makes Integration Testing A Breeze

Num	Module	Group	Case	Log	Time	Result	Comment
	lashup_hyparview_SUITE		init_per_suite	< >	0.024s	Ok	
1	lashup_hyparview_SUITE		hyparview_test	< >	34.484s	Ok	
2	lashup_hyparview_SUITE		hyparview_random_kill_test	< >	259.765s	Ok	
3	lashup_hyparview_SUITE		ping_test	< >	776.339s	Ok	
4	lashup_hyparview_SUITE		mc_test	< >	129.247s	Ok	
5	lashup_hyparview_SUITE		kv_test	< >	76.417s	Ok	
6	lashup_hyparview_SUITE		failure_test300	< >	375.089s	Ok	
	lashup_hyparview_SUITE		end_per_suite	< >	0.000s	Ok	
	lashup_kv_SUITE		init_per_suite	< >	0.283s	Ok	
7	lashup_kv_SUITE		kv_subscribe	< >	0.013s	Ok	
	lashup_kv_SUITE		end_per_suite	< >	0.001s	Ok	
	TOTAL				**1988.554s**	**Ok**	**7 Ok, 0 Failed of 7**

COMMON TEST: ON EACH CHECK-IN

(LASHUP)

- Spins up 25 virtual nodes
- Torture tests:
  - Partitions nodes randomly for 5 minutes to see if it can generate a permanent split
  - Kills nodes randomly to verify global health check convergence
  - Verifies multicast works even in poor network conditions
  - Writes divergent keys to all nodes, and reconnects them
- ~500 lines of tests, 80%+ code coverage

TAKE-AWAYS

- The container ecosystem is in is early stages
- Erlang is suitable for building modern, reliable distributed systems
    - These systems are tricky, so testing is key
- We have a healthy ecosystem
- There need to be alternatives to consensus

ERLANG & CONTAINER NETWORKING

@SARGUN

MESOSPHERE