PACEMAKER

OpenStack's PID 1

OpenStack Summit

May 21, 2015

David Vossel <dvossel@redhat.com>

David Vossel <dvossel@redhat.com>

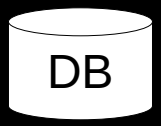
The Future of HA

And how Pacemaker Saved the Day

David Vossel <dvossel@redhat.com>

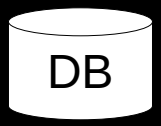
There once was a database

There once was a database



There once was a database

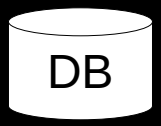
Not like the other databases



There once was a database

Not like the other databases

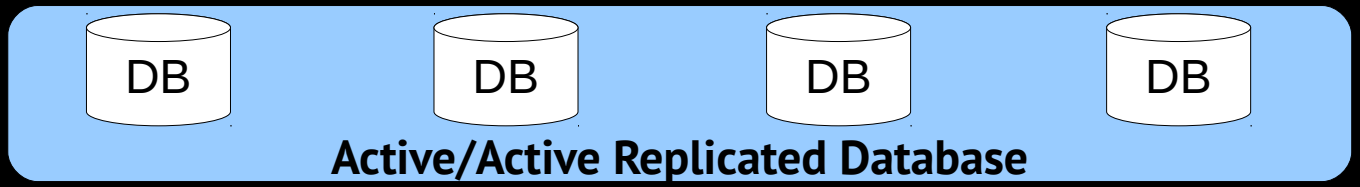
A distributed self replicating database



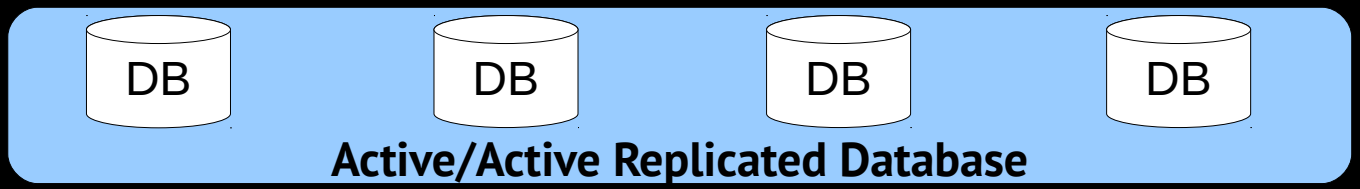
There once was a database

Not like the other databases

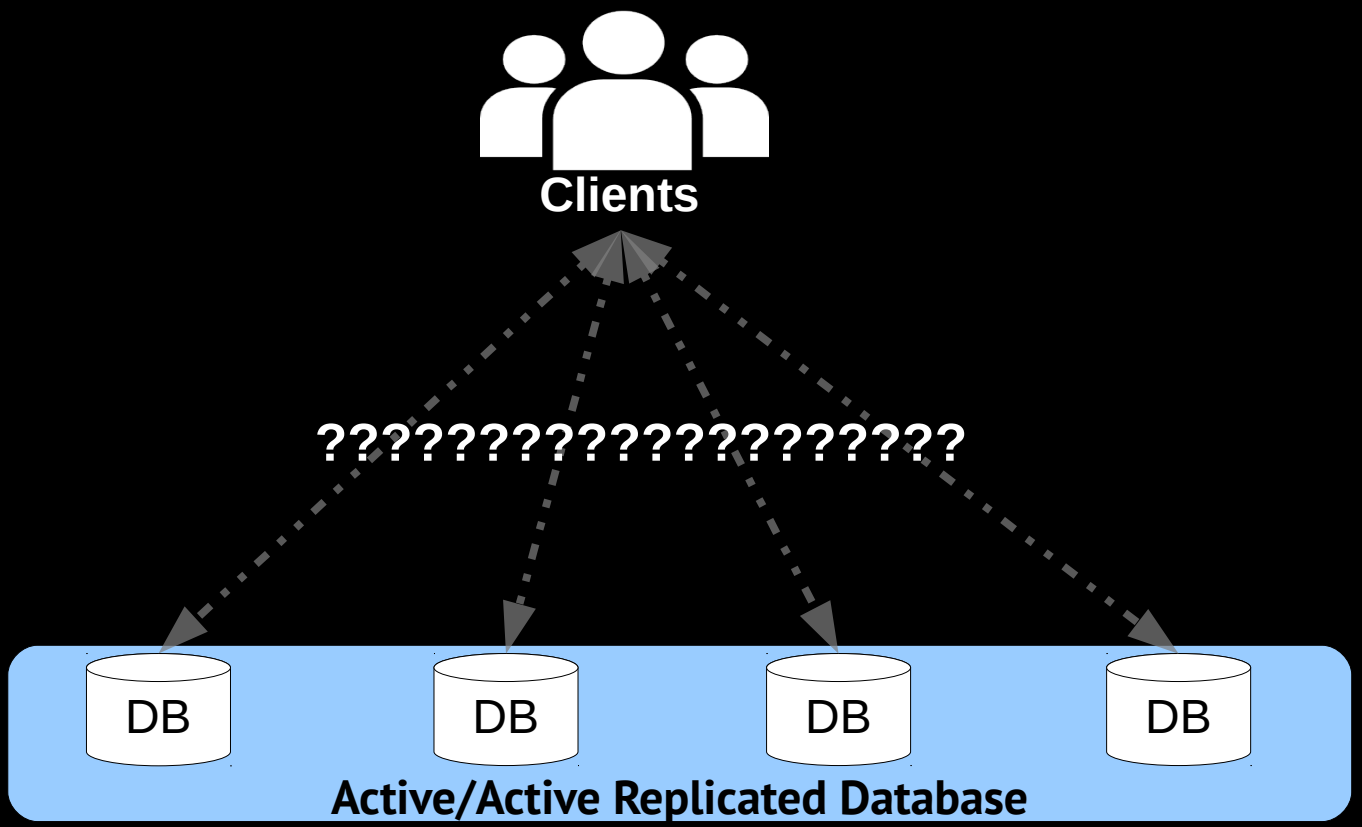
A distributed self replicating database



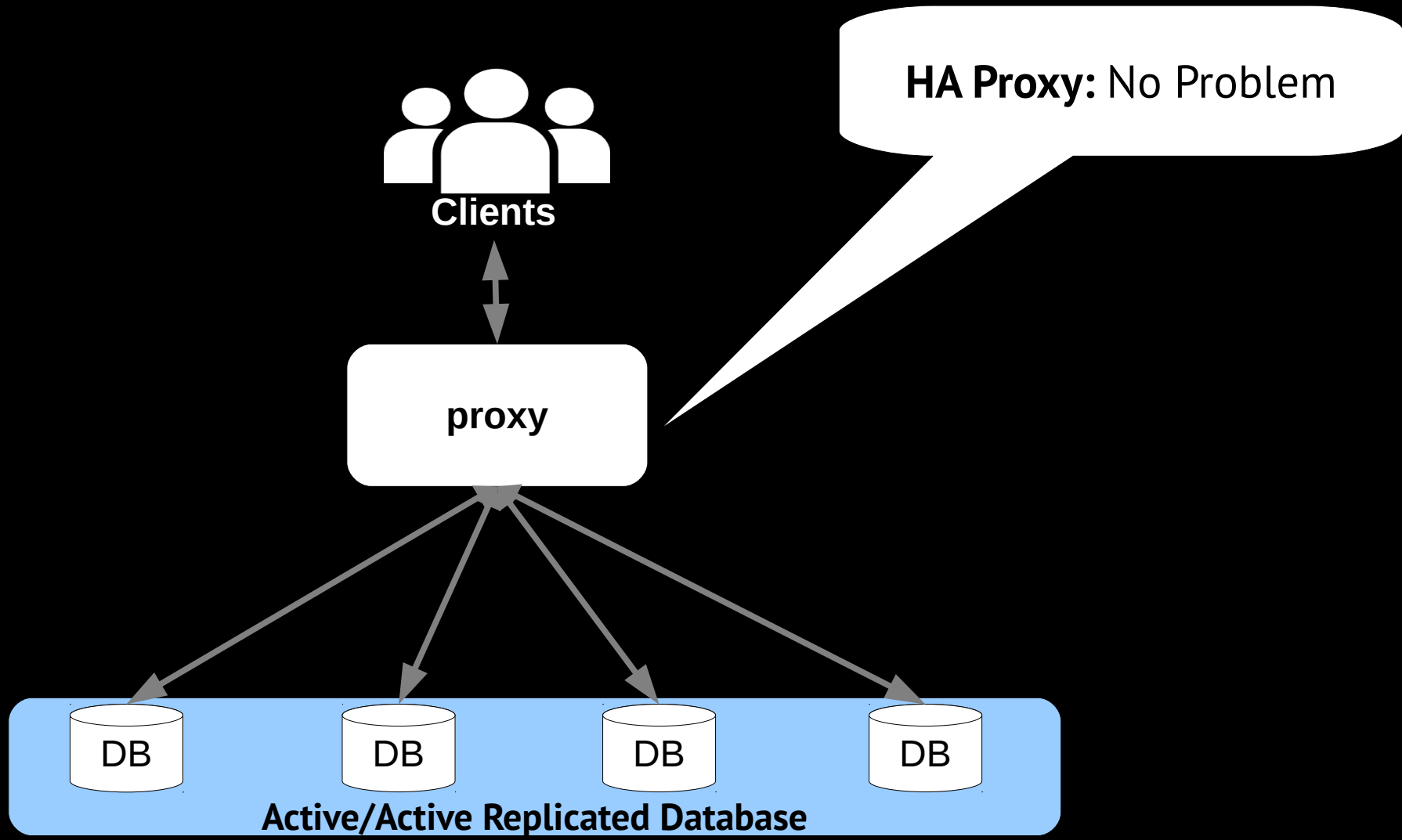
Everyone: HOORAY!

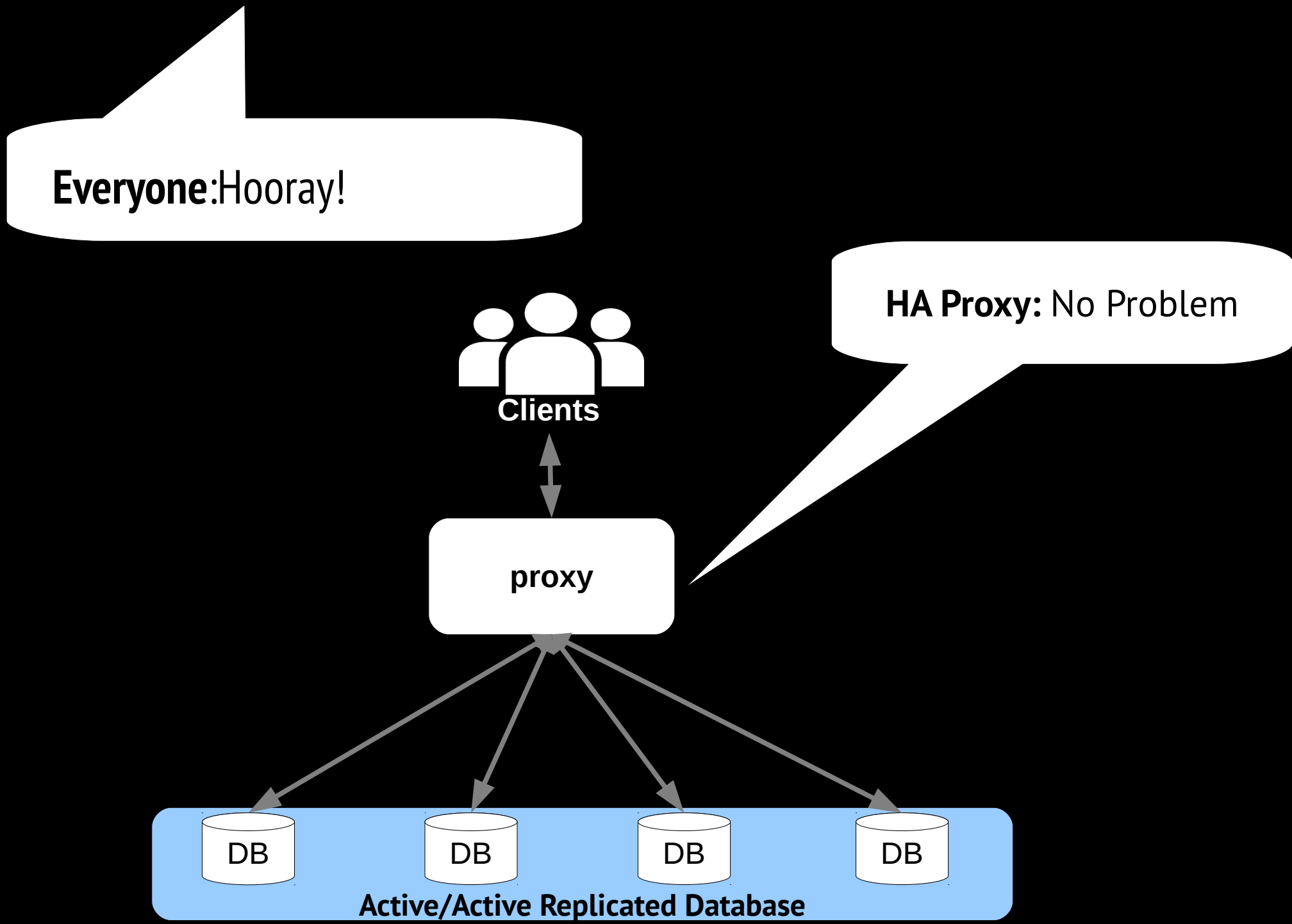


Everyone: Load Balance?

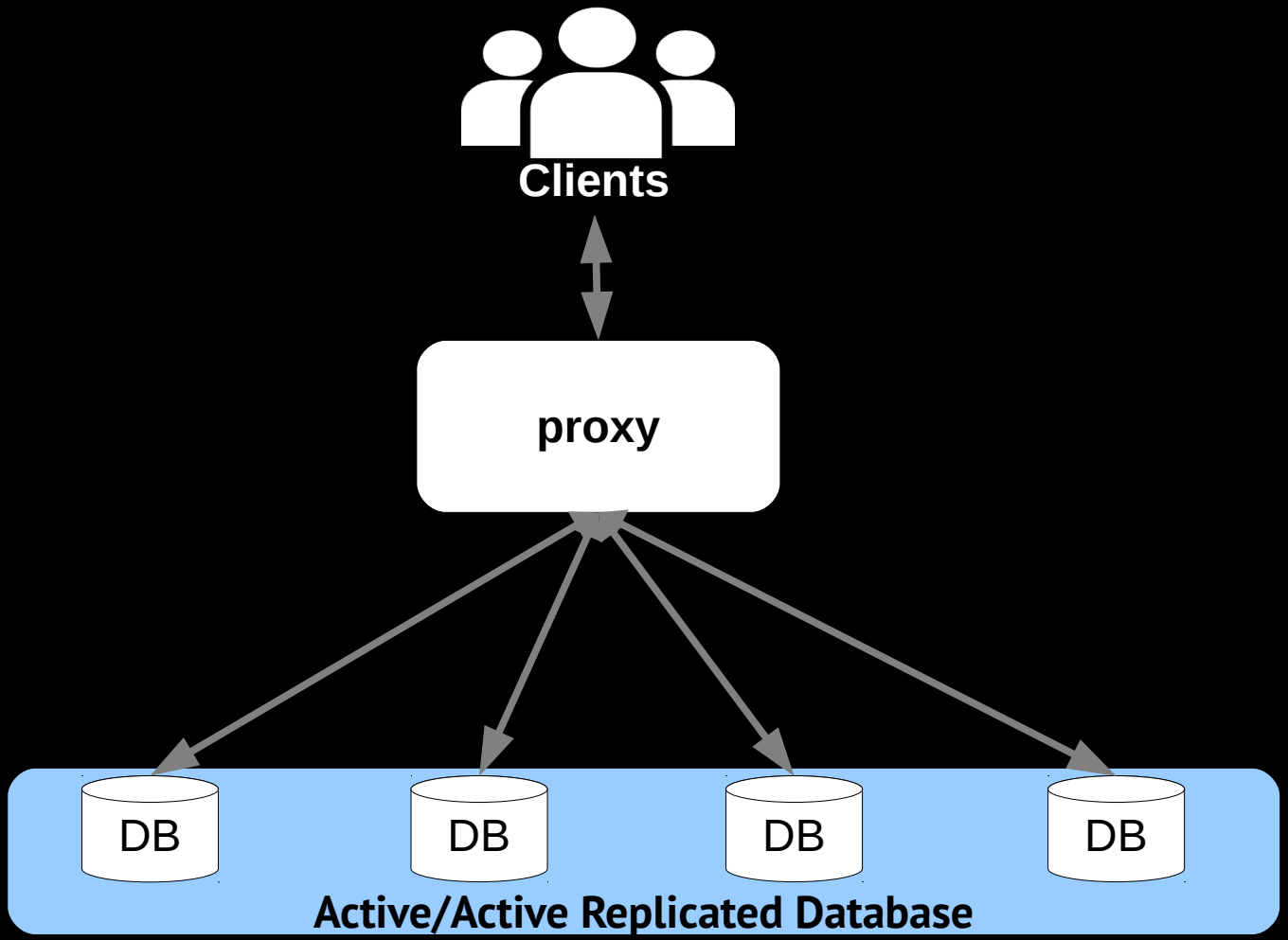


HAProxy enters the scene.

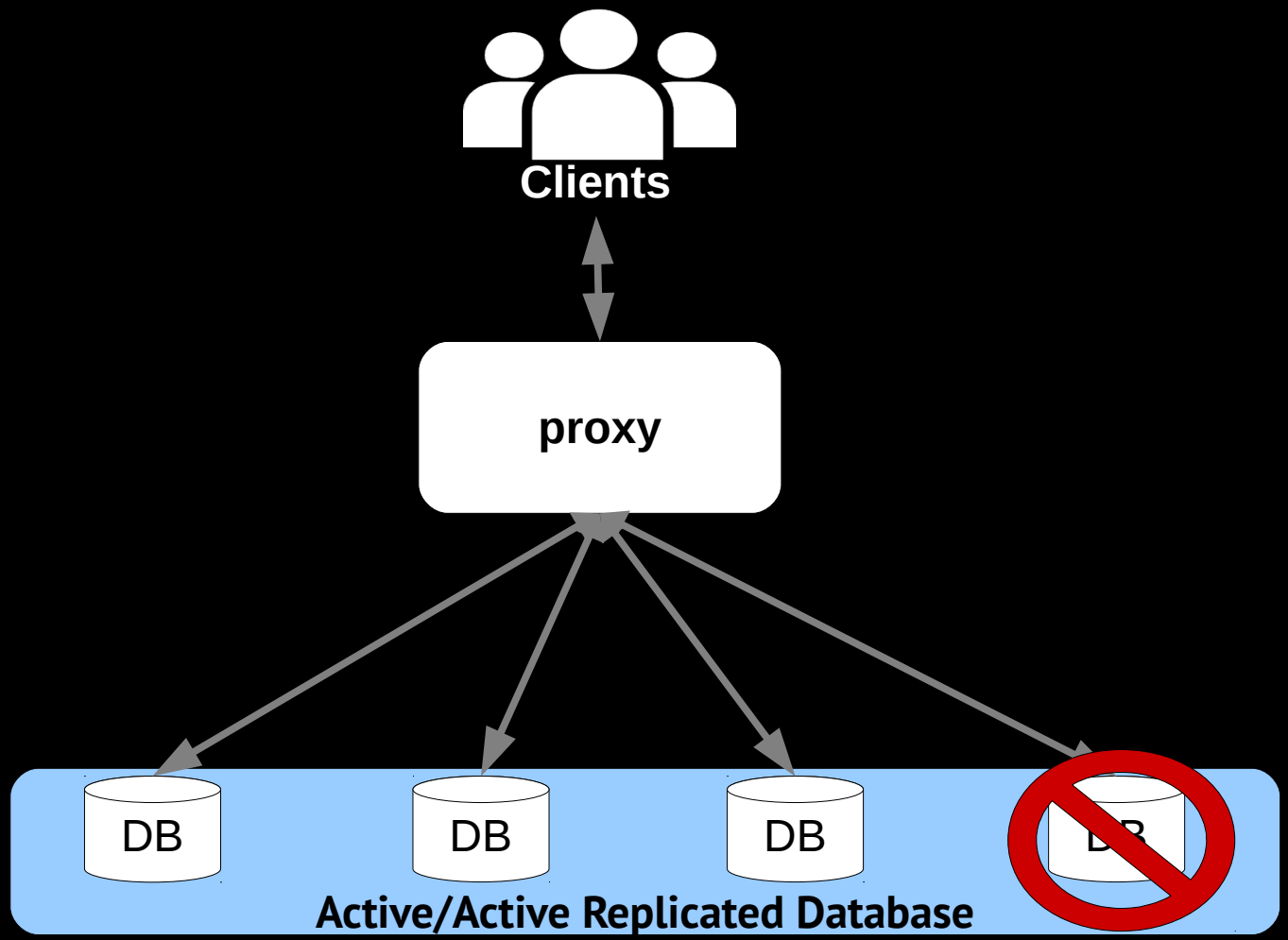

HA Proxy: No Problem
Clients
proxy
DB
DB
DB
DB
Active/Active Replicated Database


Everyone:Hooray!
HA Proxy: No Problem
Clients
proxy
DB
DB
DB
DB
Active/Active Replicated Database

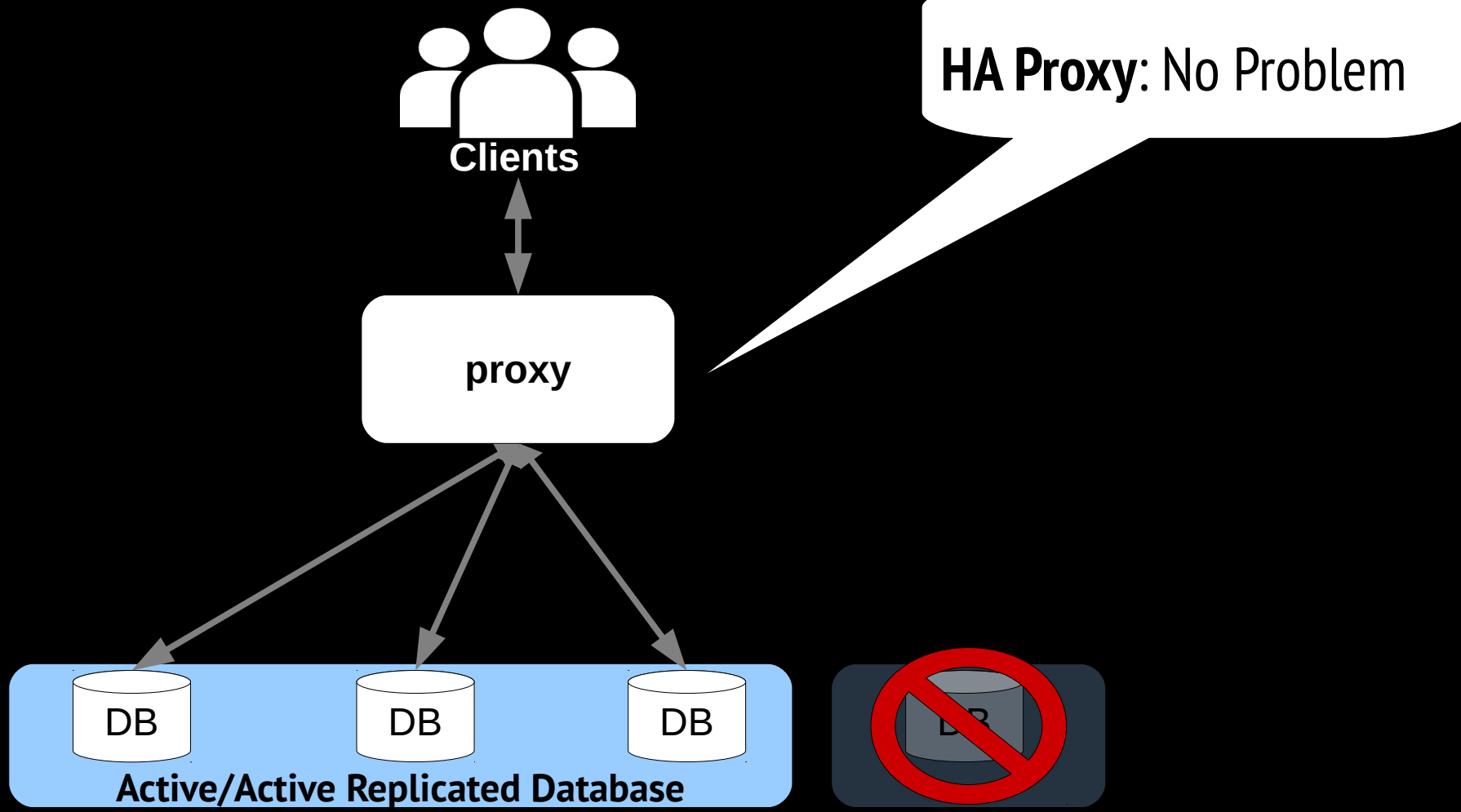
* **Everyone**: ummmmm



Everyone: What if a node dies?

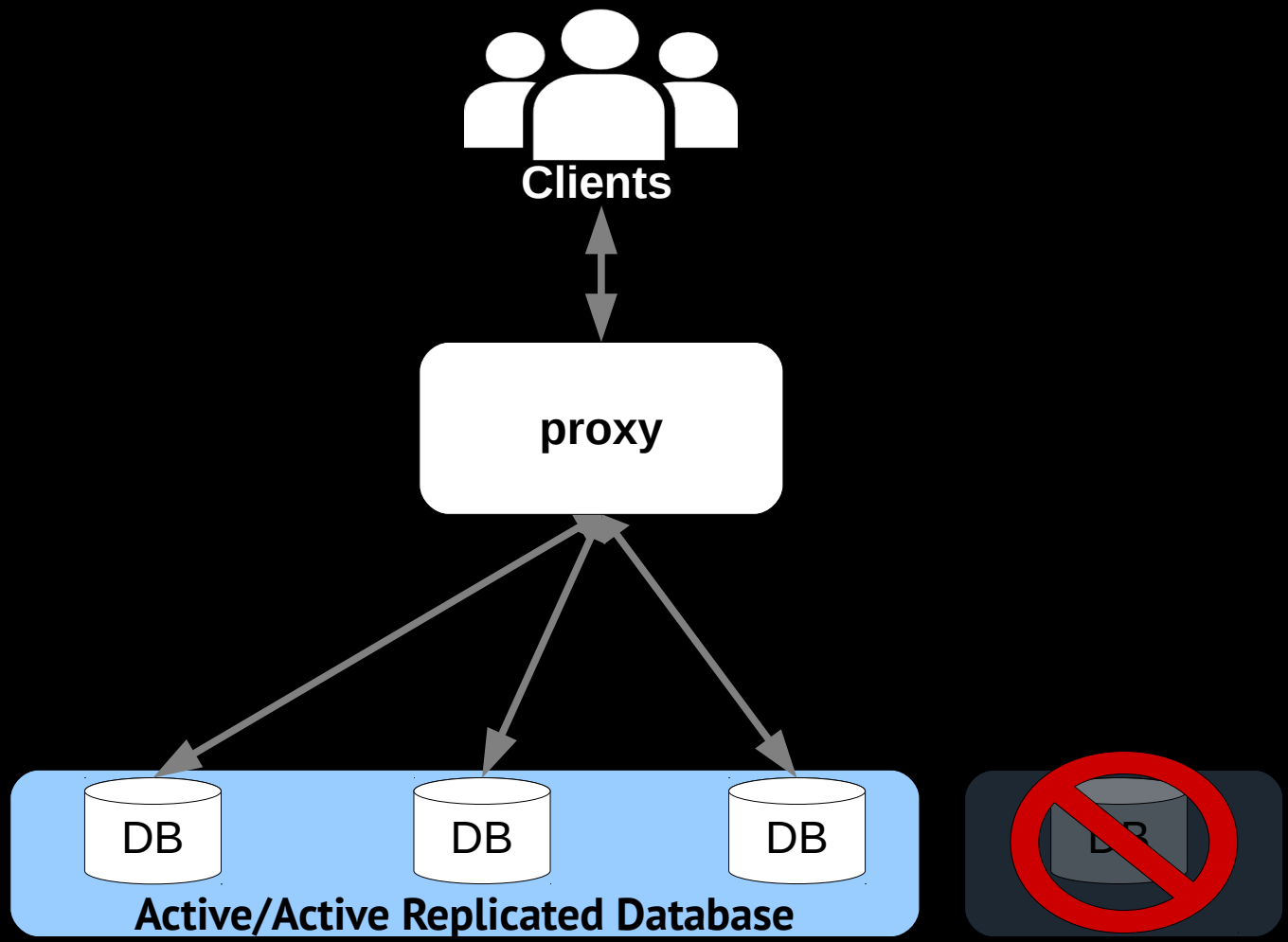


Everyone: What if a node dies?

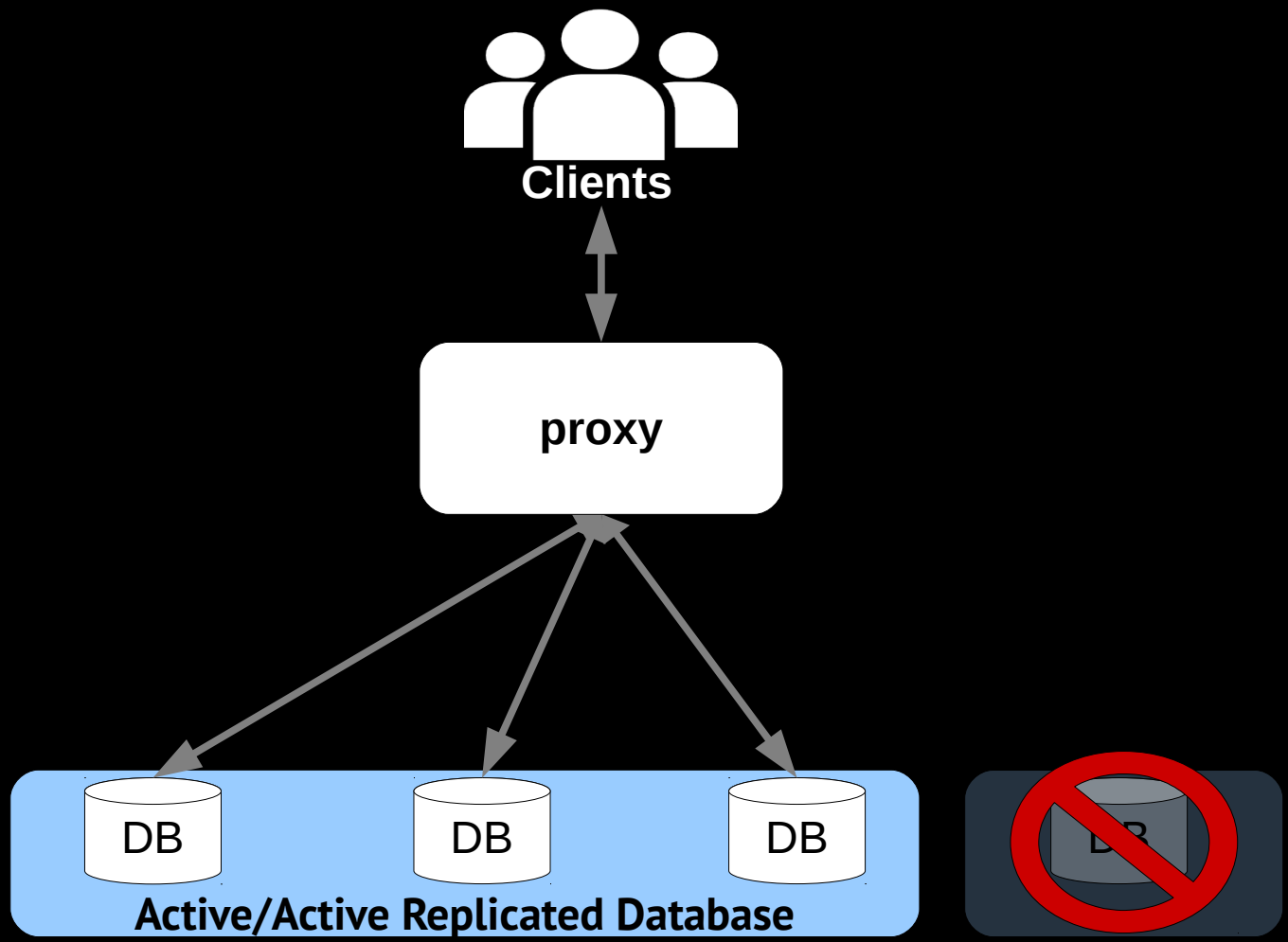


HA Proxy: No Problem

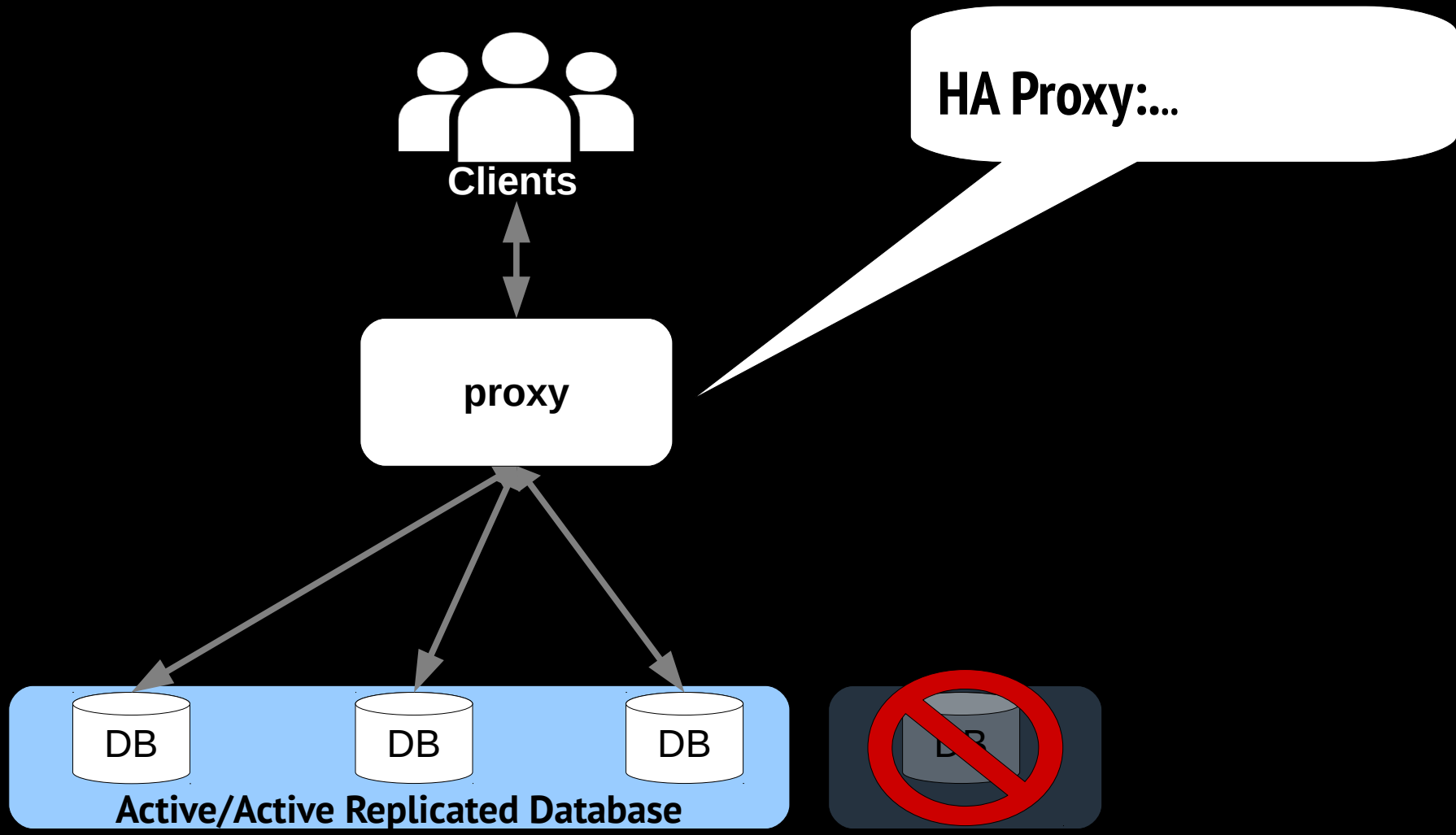
* **Everyone**: ummmmm



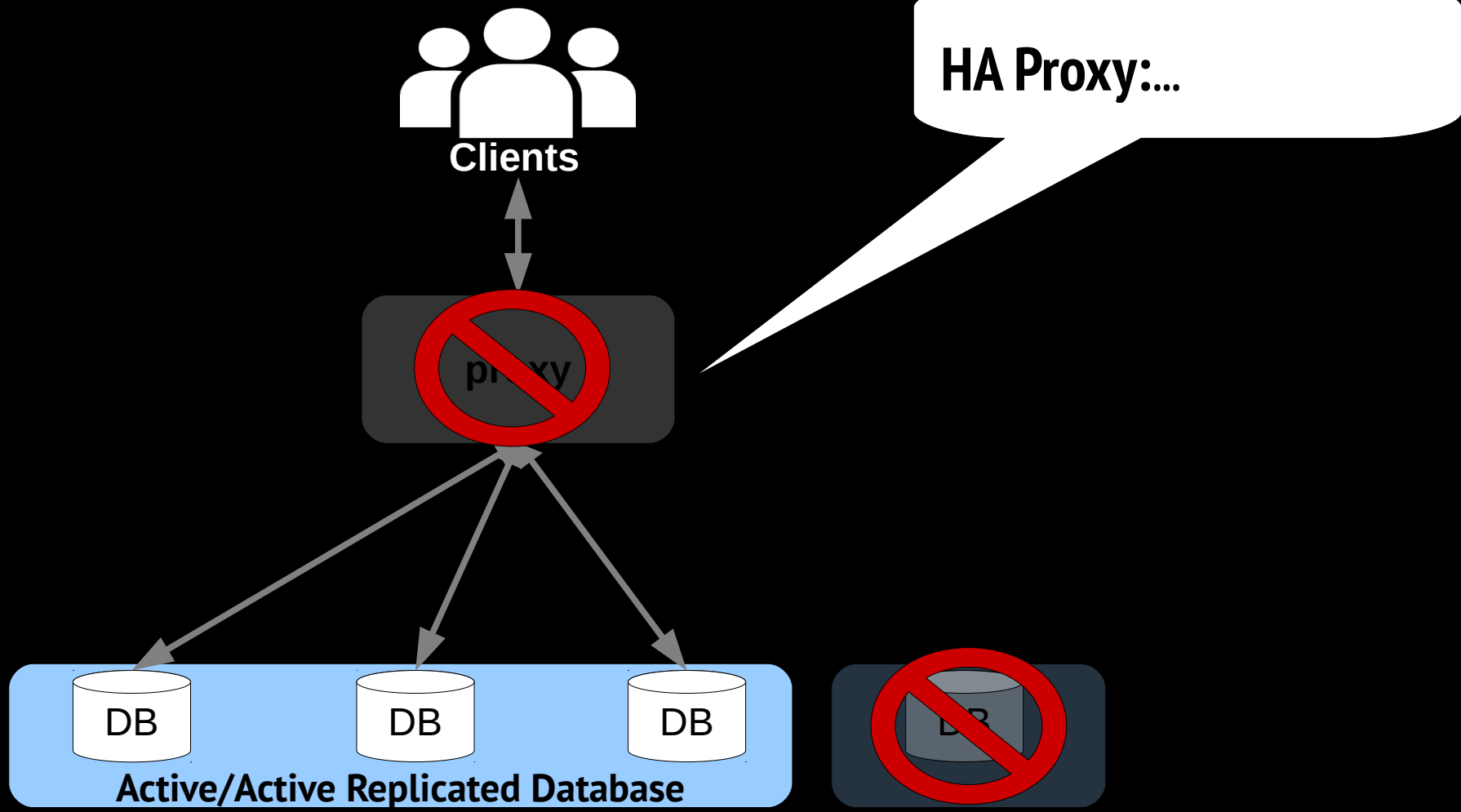
Everyone: But... What if **proxy** dies?



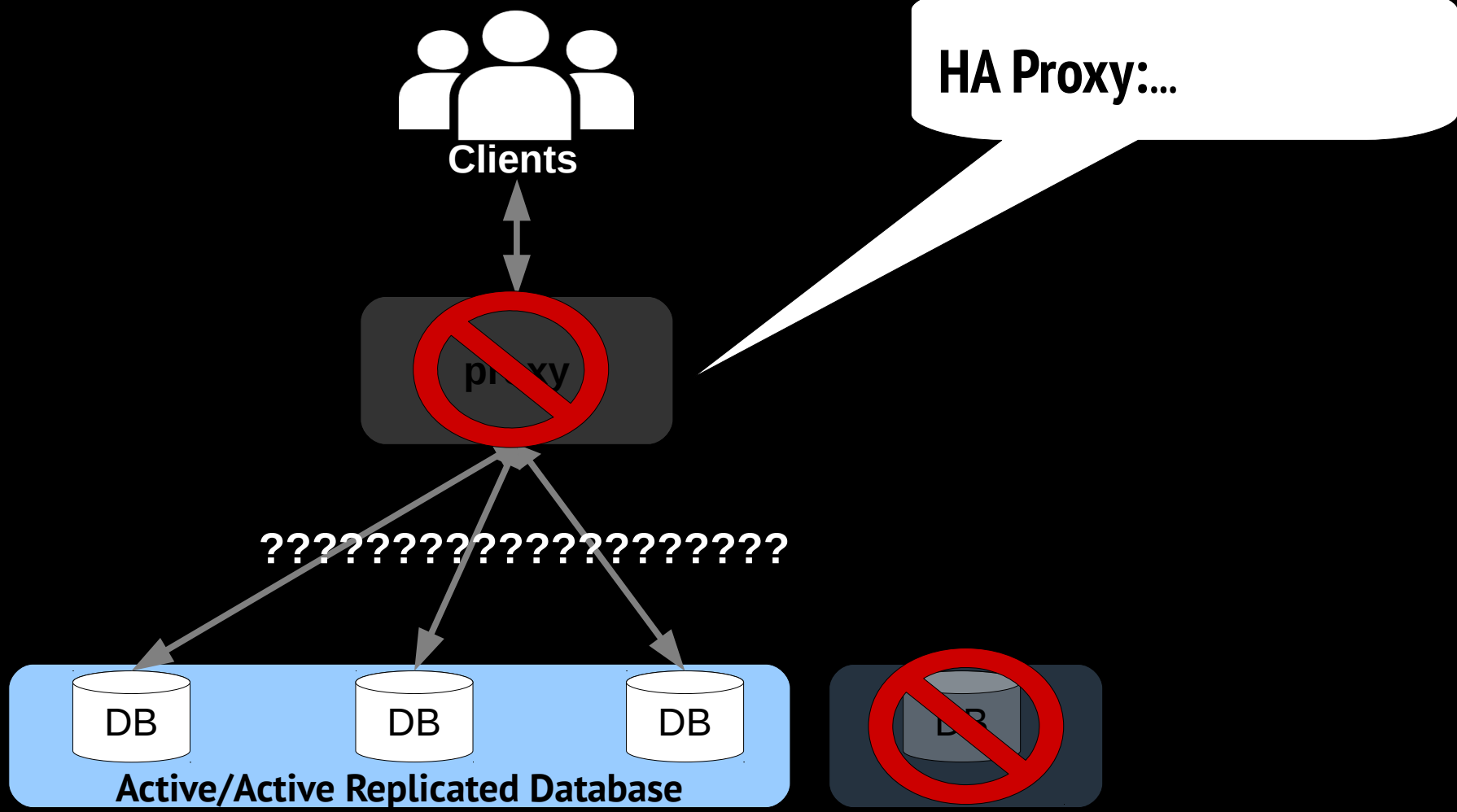
Everyone: But... What if **proxy** dies?



Everyone:What?????



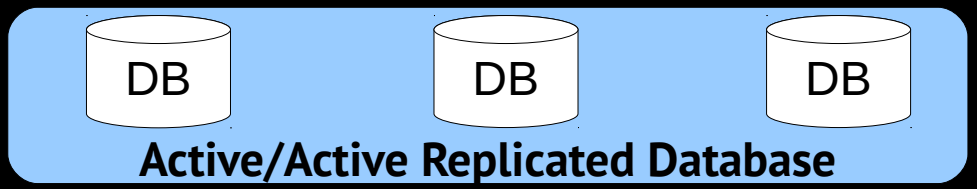
Everyone:Uhhh???!!!!??

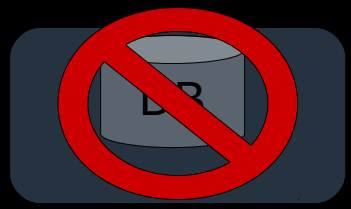


Everyone:Hello????

HA Proxy:...

??????????????????





Everyone:Anyone?

Clients

??????????????????

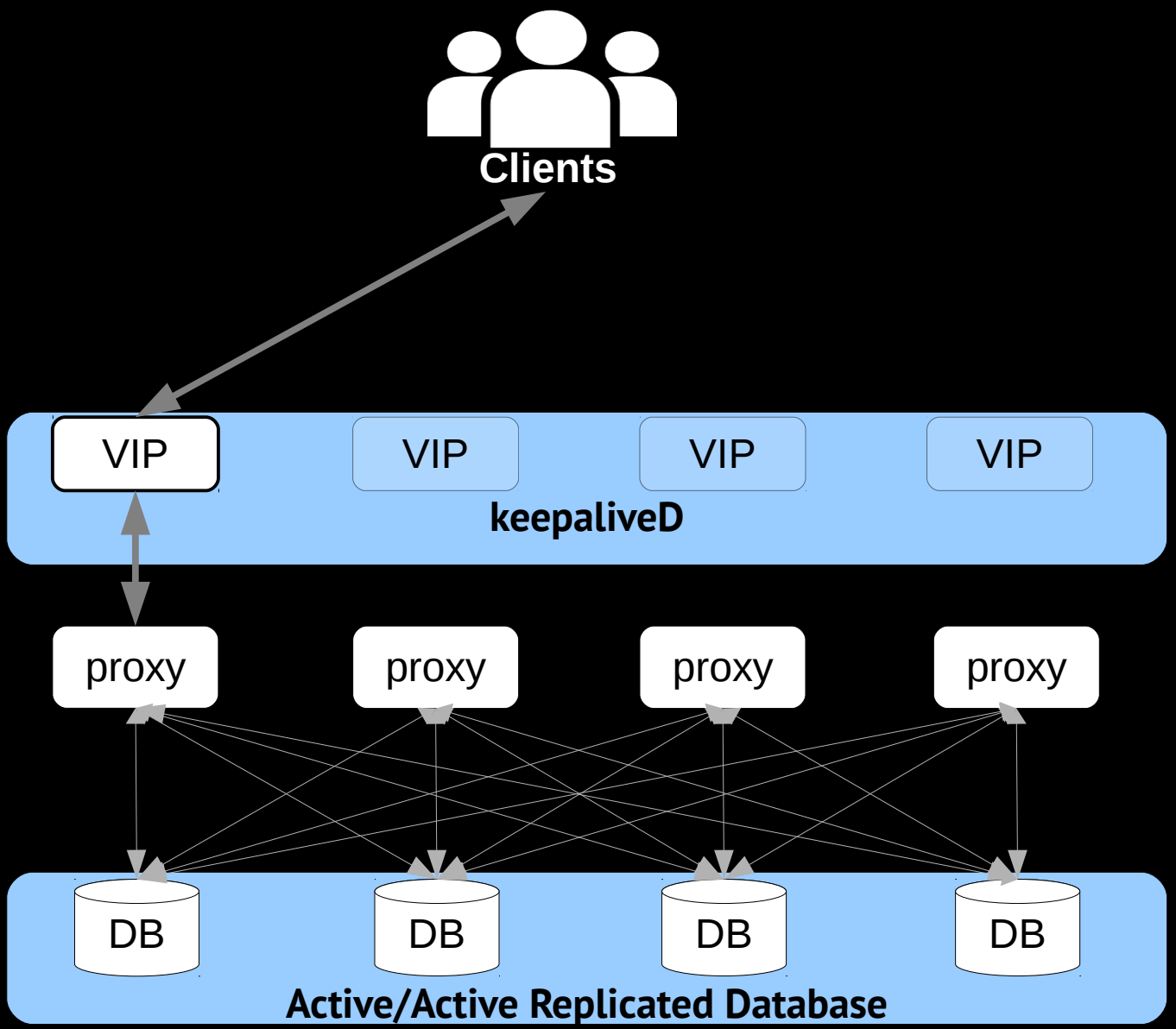
Everyone:I knew this cloud thing wouldn't work...

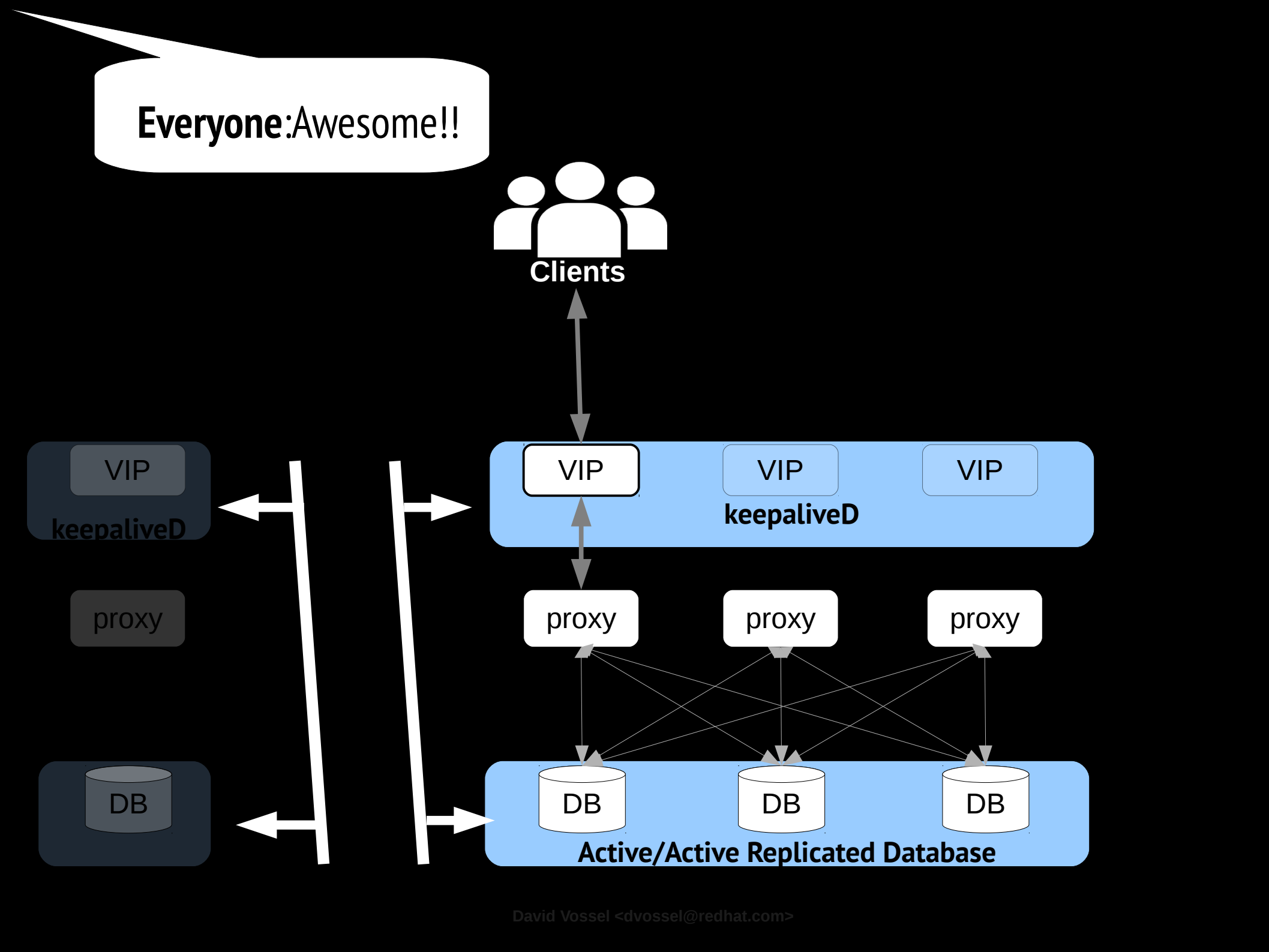
Everyone:I knew this cloud thing wouldn't work...

KeepaliveD: Wait! Guys... I've got an idea!!!

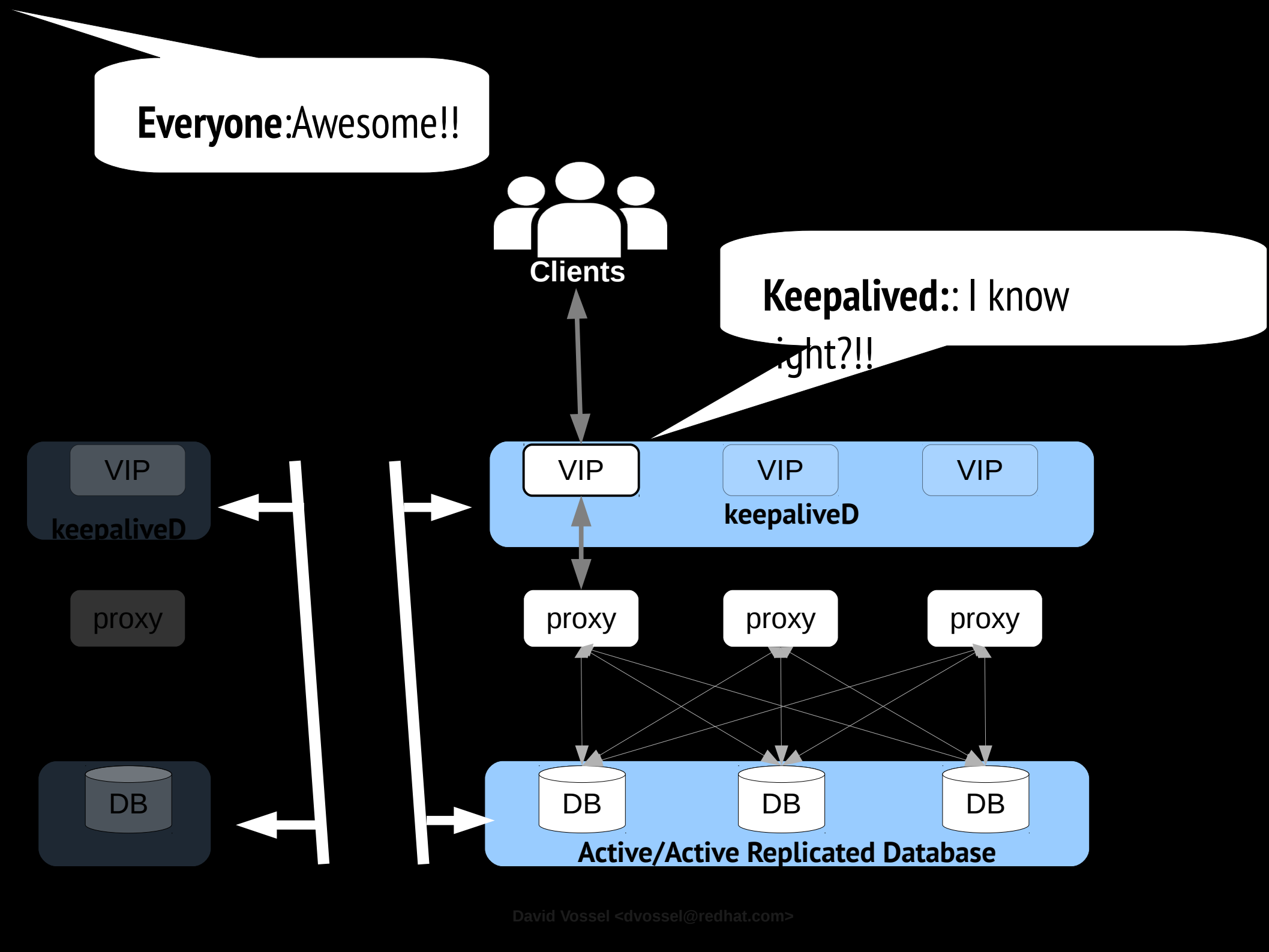
KeepaliveD enters the scene.

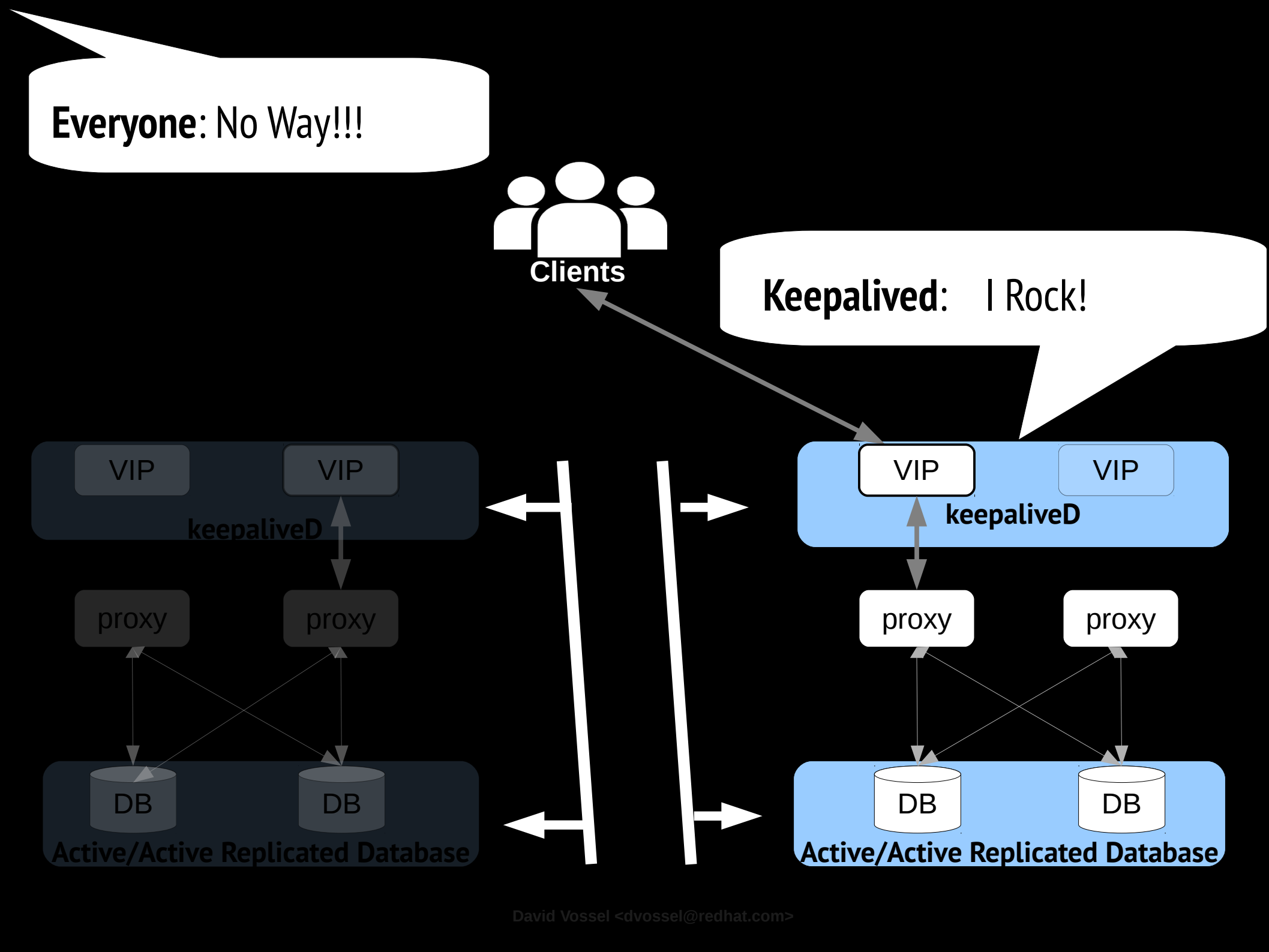
Everyone:Whoa, Proxy Failover!



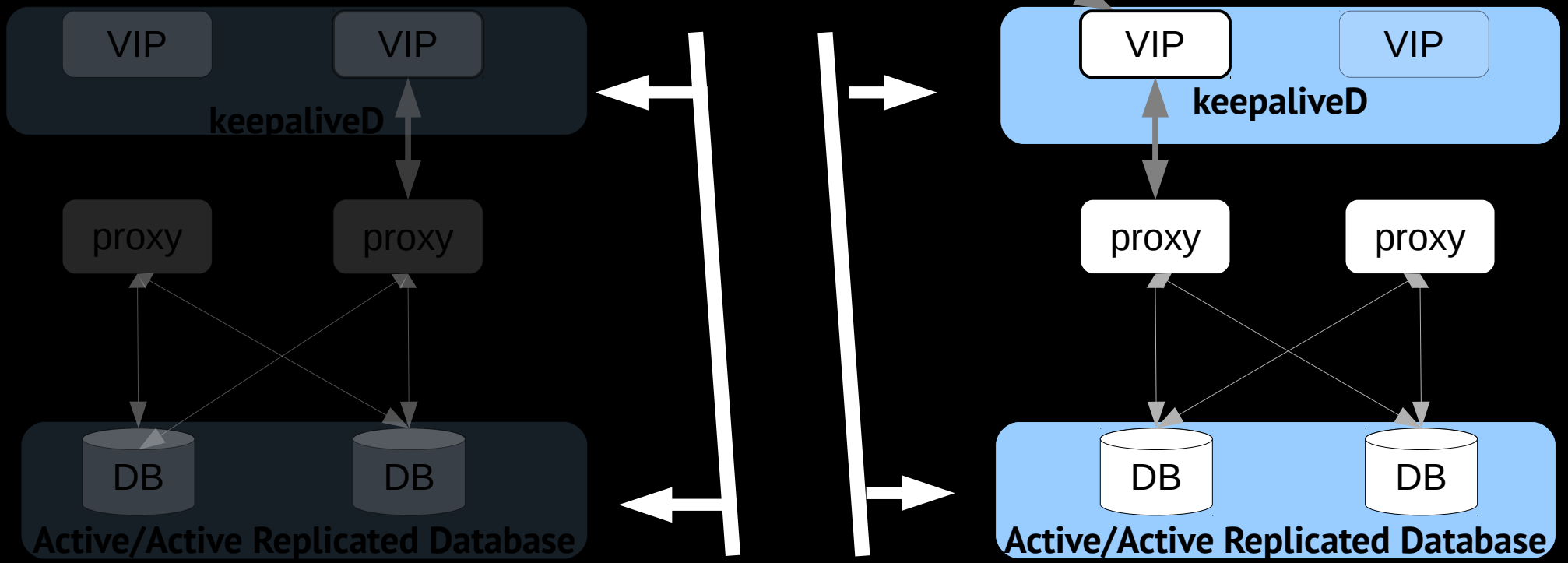


David Vossel <dvossel@redhat.com>

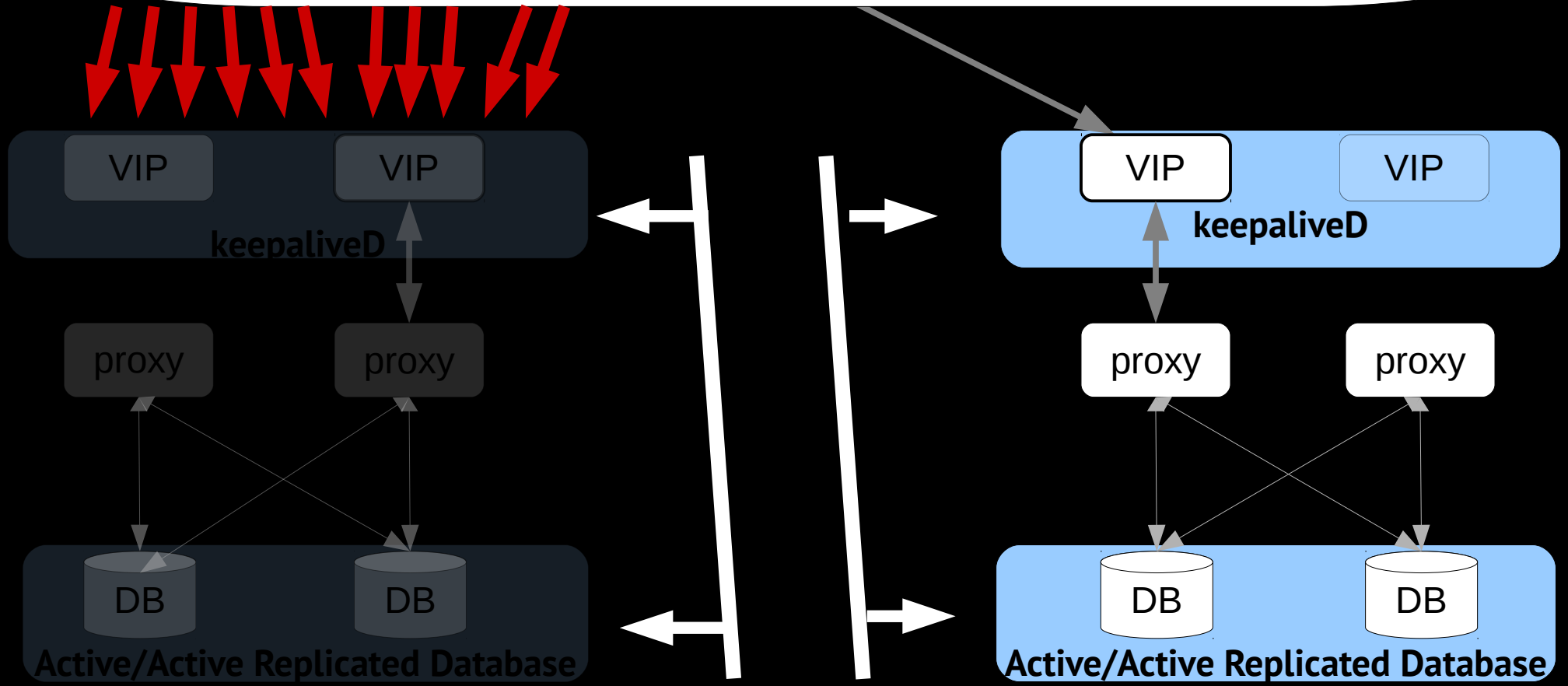

Everyone:Awesome!!
Clients
Keepalived:: I know
ight?!!
VIP
keepaliveD
proxy
DB
VIP
VIP
VIP
keepaliveD
proxy
proxy
proxy
DB
DB
DB
Active/Active Replicated Database


Everyone: No Way!!!
Clients
Keepalived: I Rock!
VIP
VIP
keepaliveD
proxy
proxy
DB
DB
Active/Active Replicated Database
VIP
VIP
keepaliveD
proxy
proxy
DB
DB
Active/Active Replicated Database

Everyone: Wait... Hold up

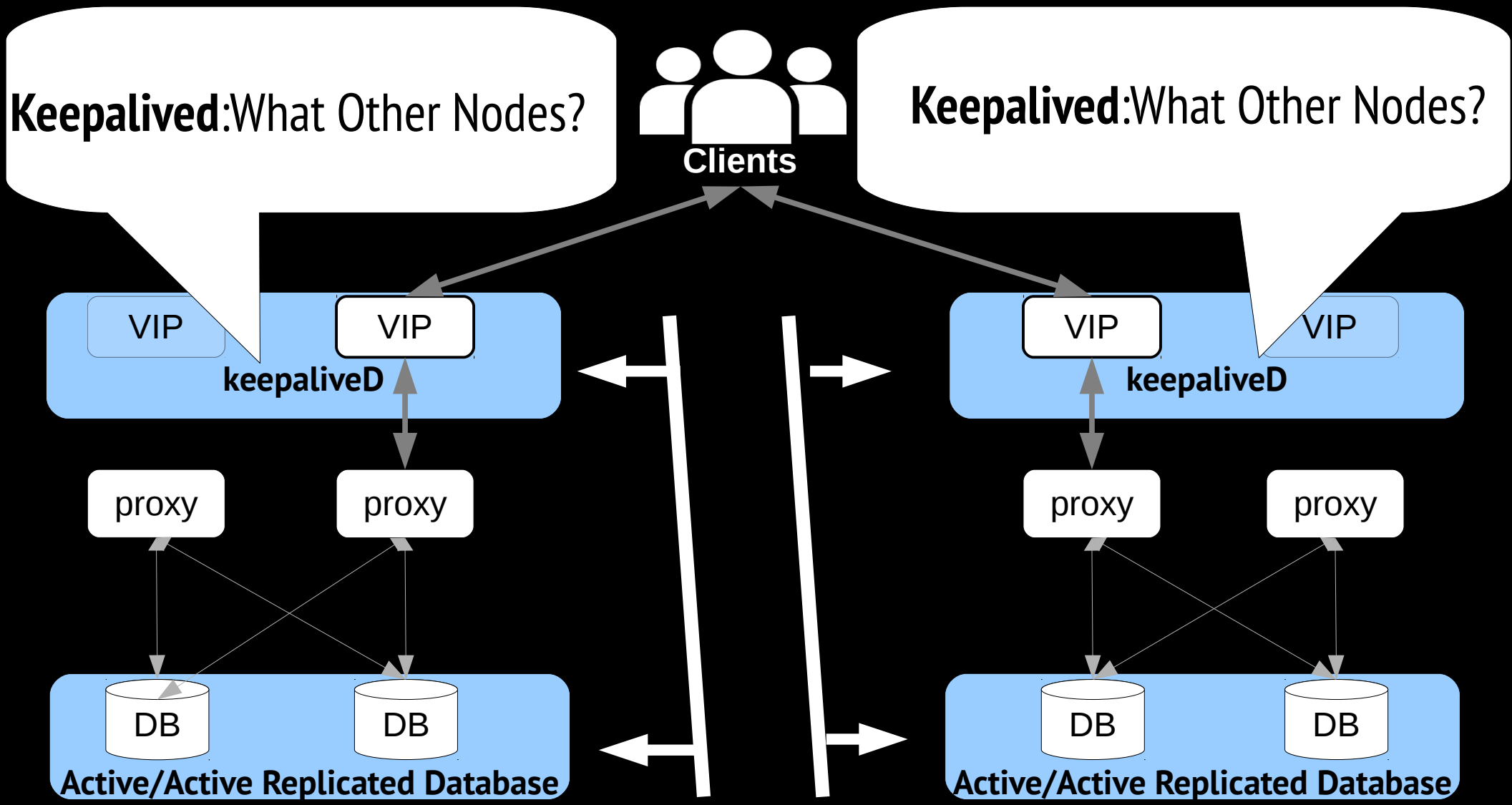


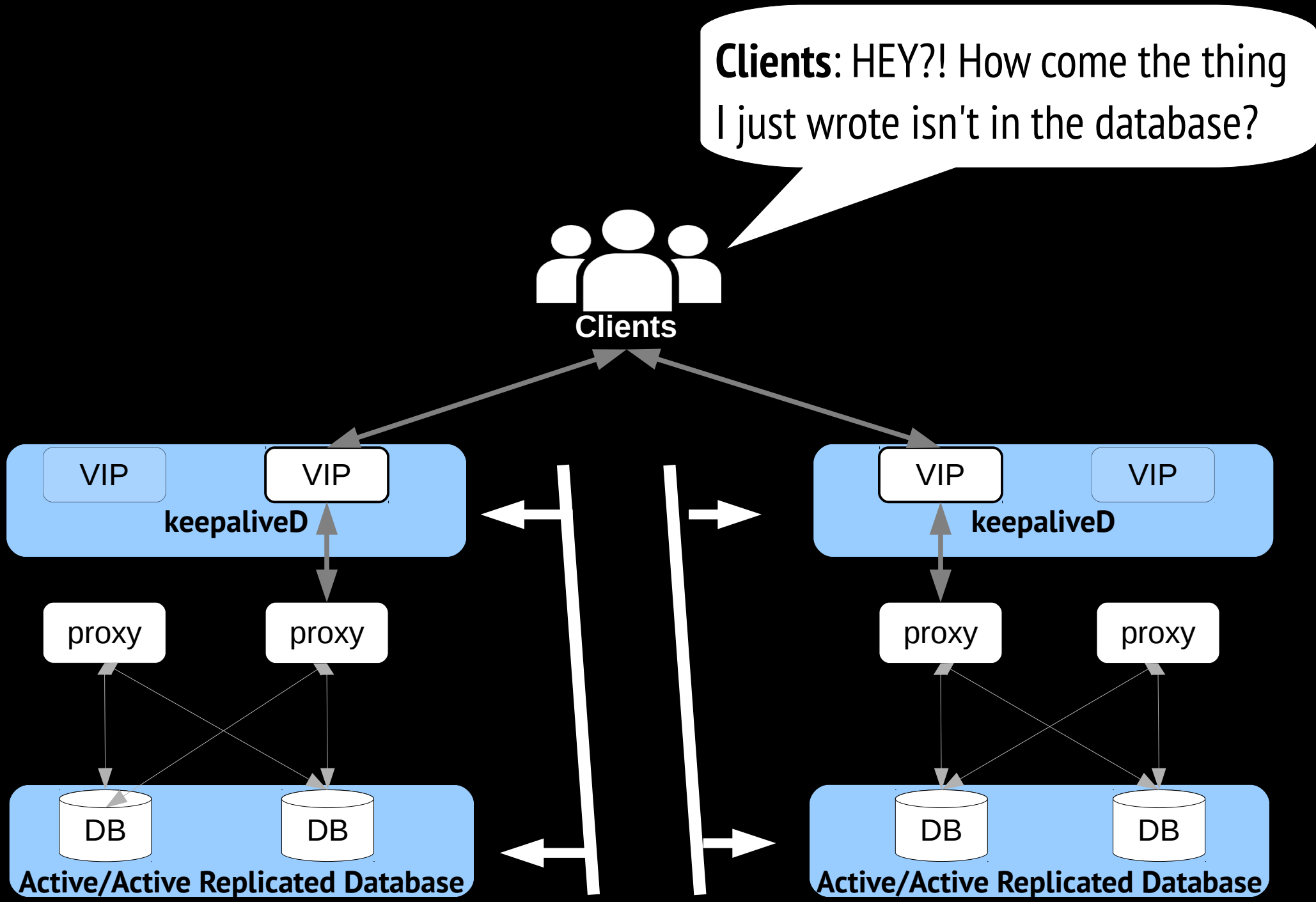
Everyone: But.... What actually happens to the “other” nodes?

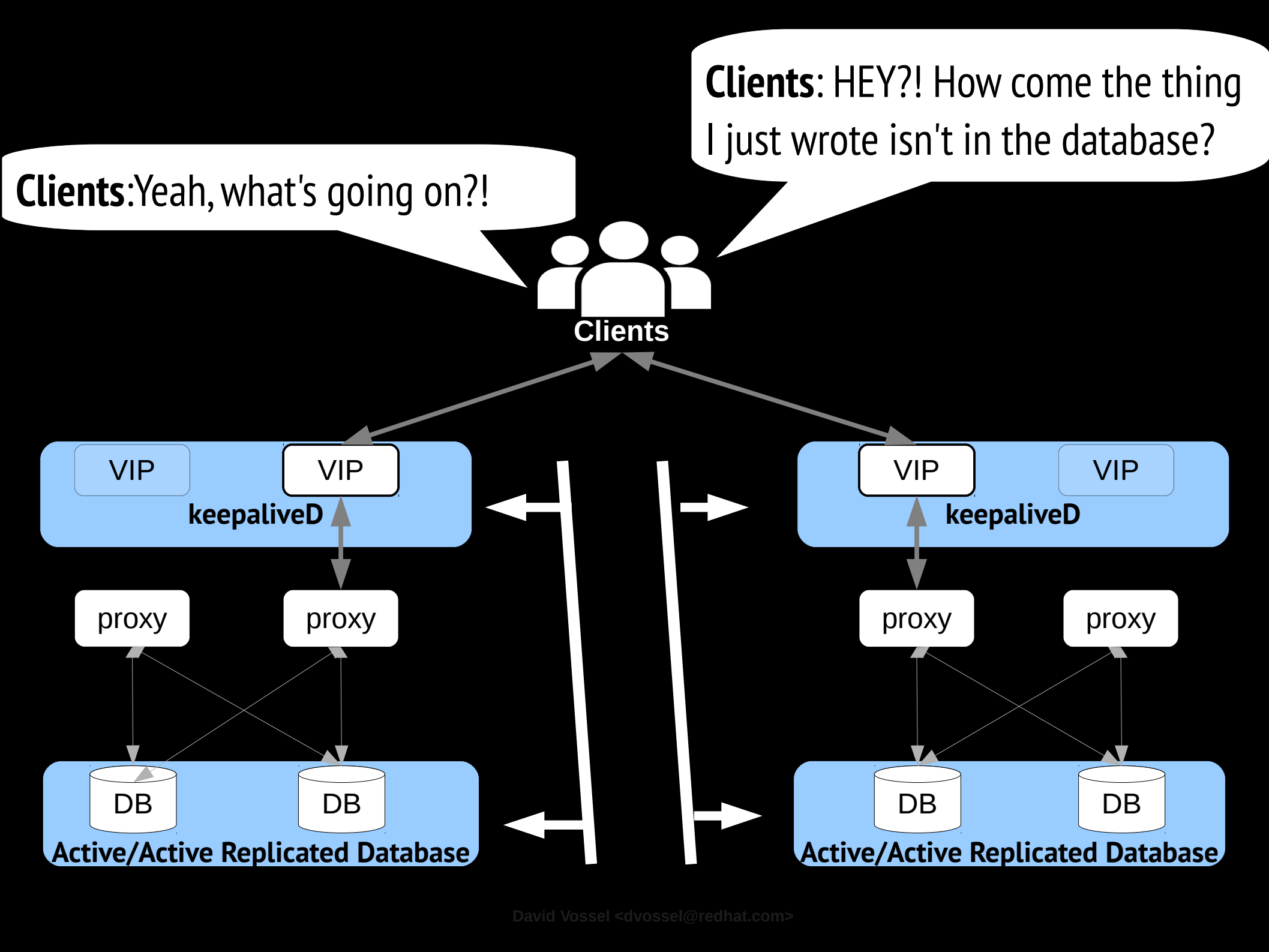


Peripeteia - per·i·pe·tei·a

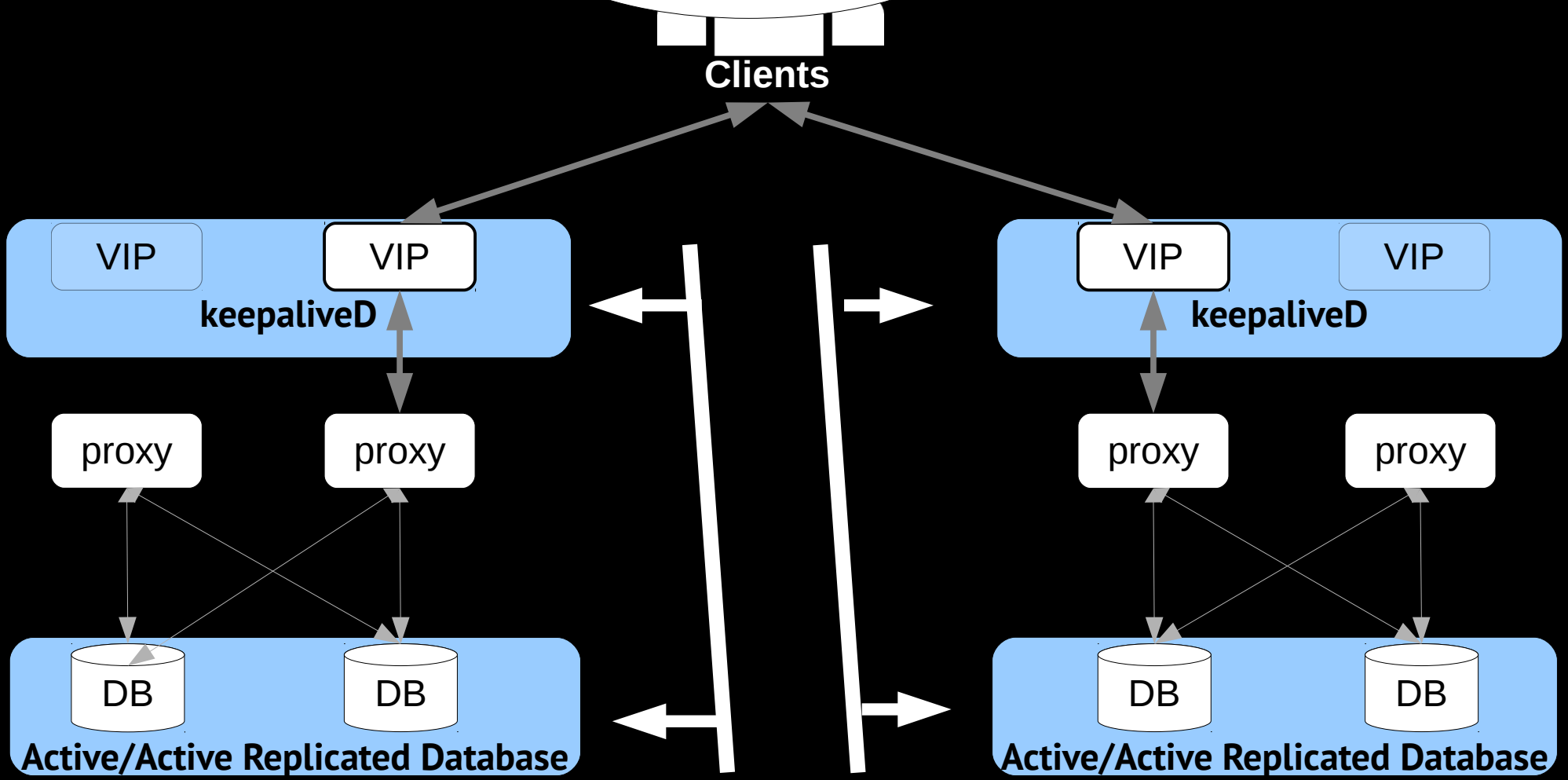
a sudden reversal of fortune or change in circumstances, especially in reference to fictional narrative.


Keepalived:What Other Nodes?
Clients
Keepalived:What Other Nodes?
VIP
VIP
keepaliveD
VIP
VIP
keepaliveD
proxy
proxy
proxy
proxy
DB
DB
Active/Active Replicated Database
DB
DB
Active/Active Replicated Database


Clients: HEY?! How come the thing I just wrote isn't in the database?
Clients
VIP
VIP
keepaliveD
VIP
VIP
keepaliveD
proxy
proxy
proxy
proxy
DB
DB
DB
DB
Active/Active Replicated Database
Active/Active Replicated Database


Clients: HEY?! How come the thing I just wrote isn't in the database?
Clients:Yeah, what's going on?!
Clients
VIP
VIP
keepaliveD
VIP
VIP
keepaliveD
proxy
proxy
proxy
proxy
DB
DB
DB
DB
Active/Active Replicated Database
Active/Active Replicated Database

* **Everyone**: I've made a huge mistake....

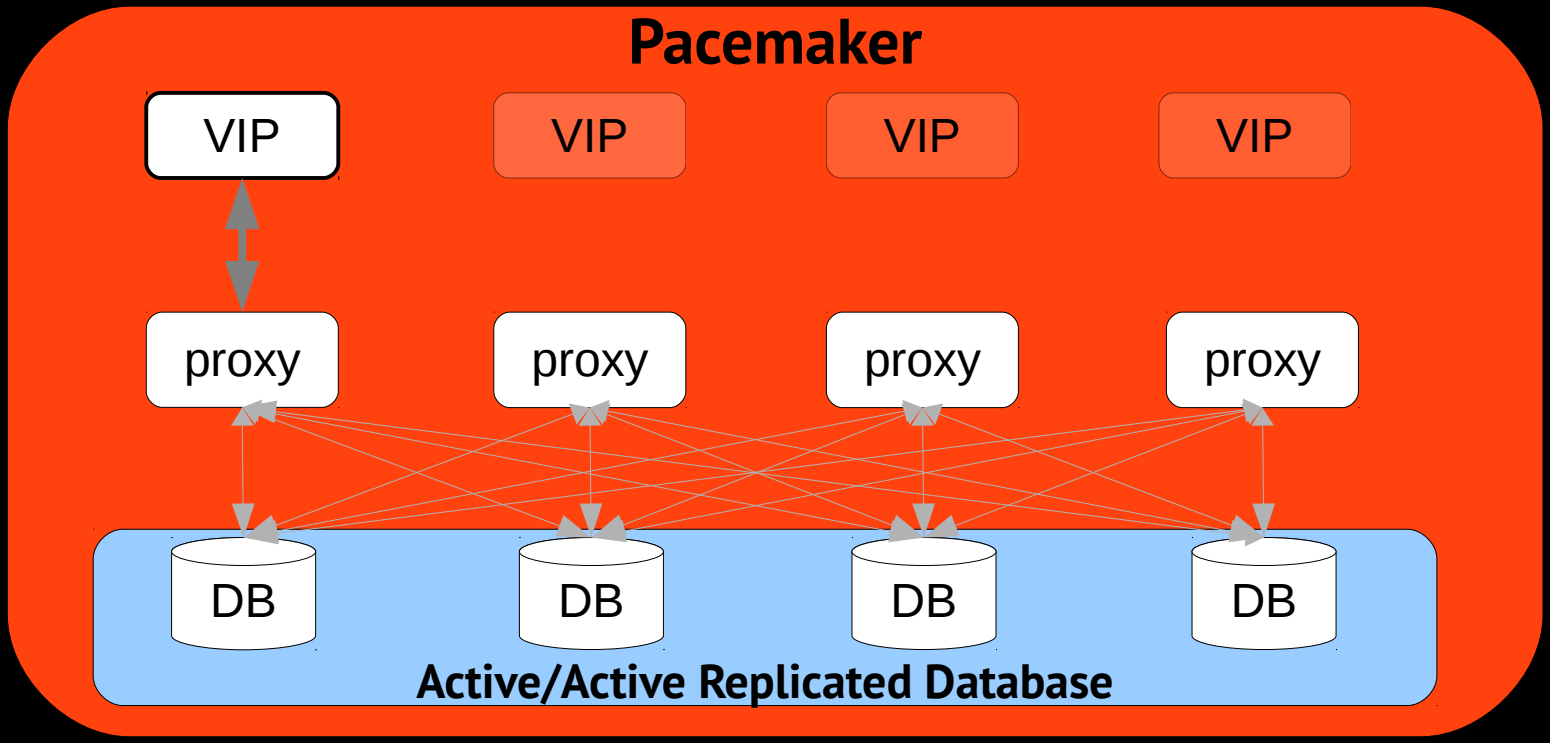


The Missing Piece?

David Vossel <dvossel@redhat.com>

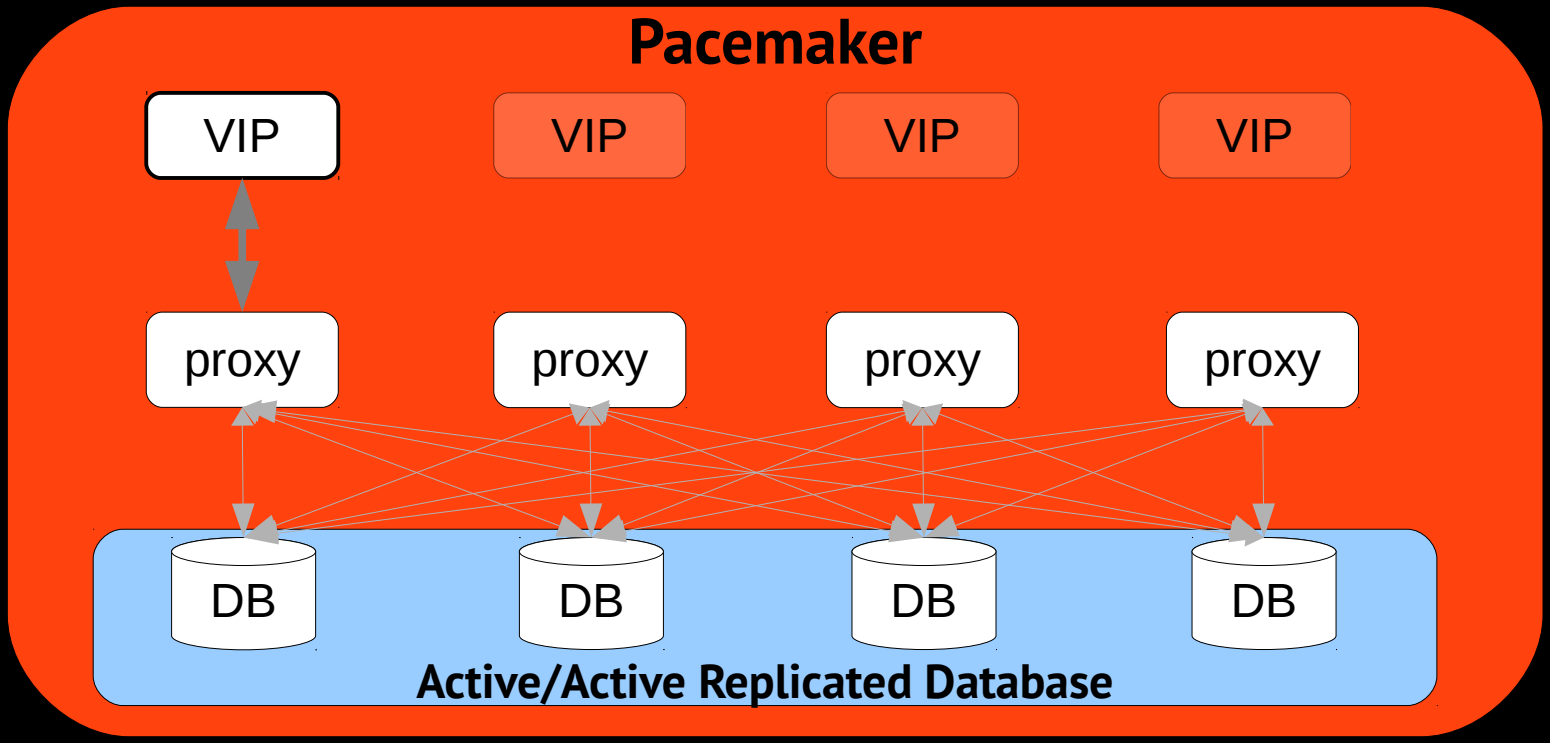
Pacemaker: System Level HA

- ***System level HA*** is holistic.



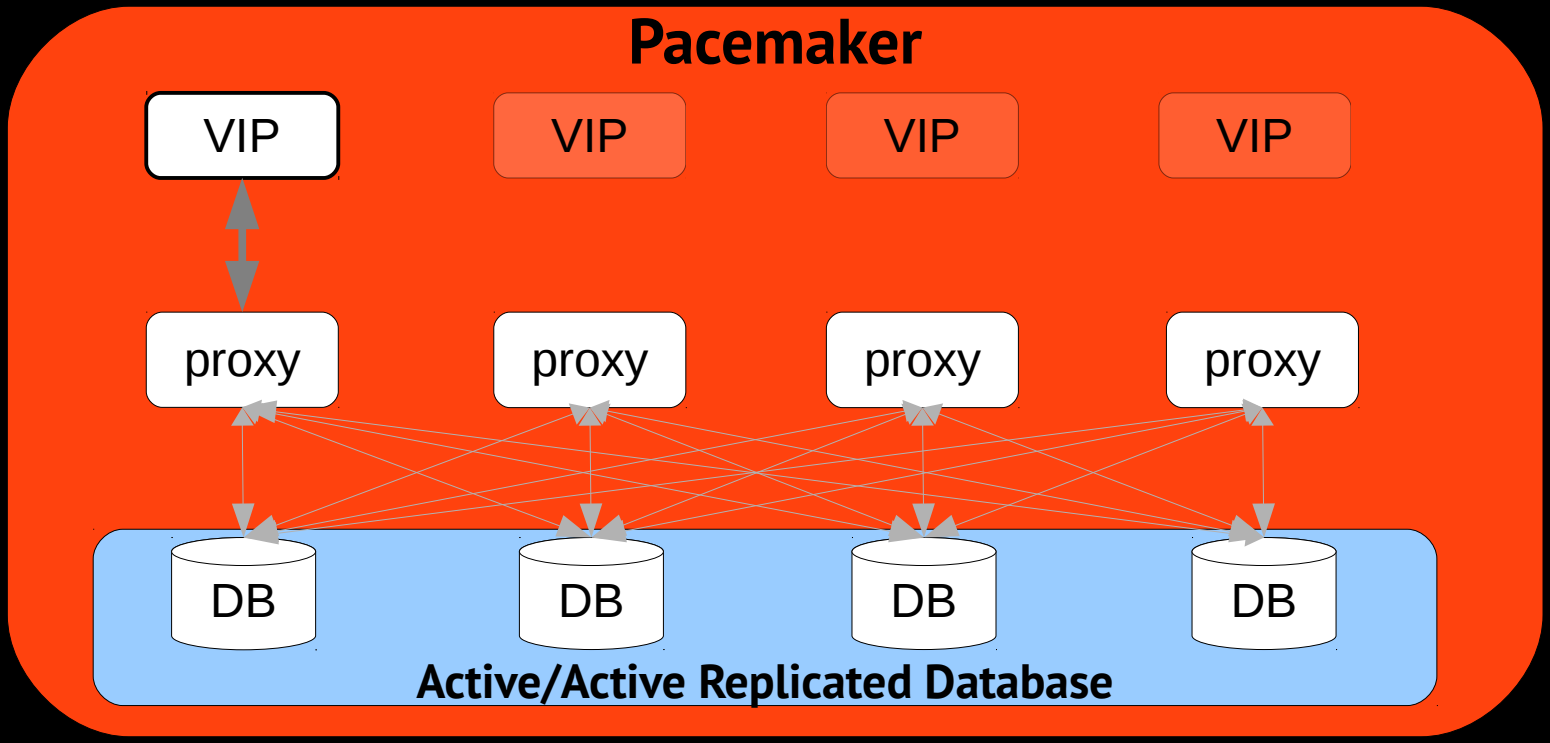
Pacemaker: System Level HA

- ***System level HA*** is holistic.
- **Defines the policy** of how to recover a set of applications



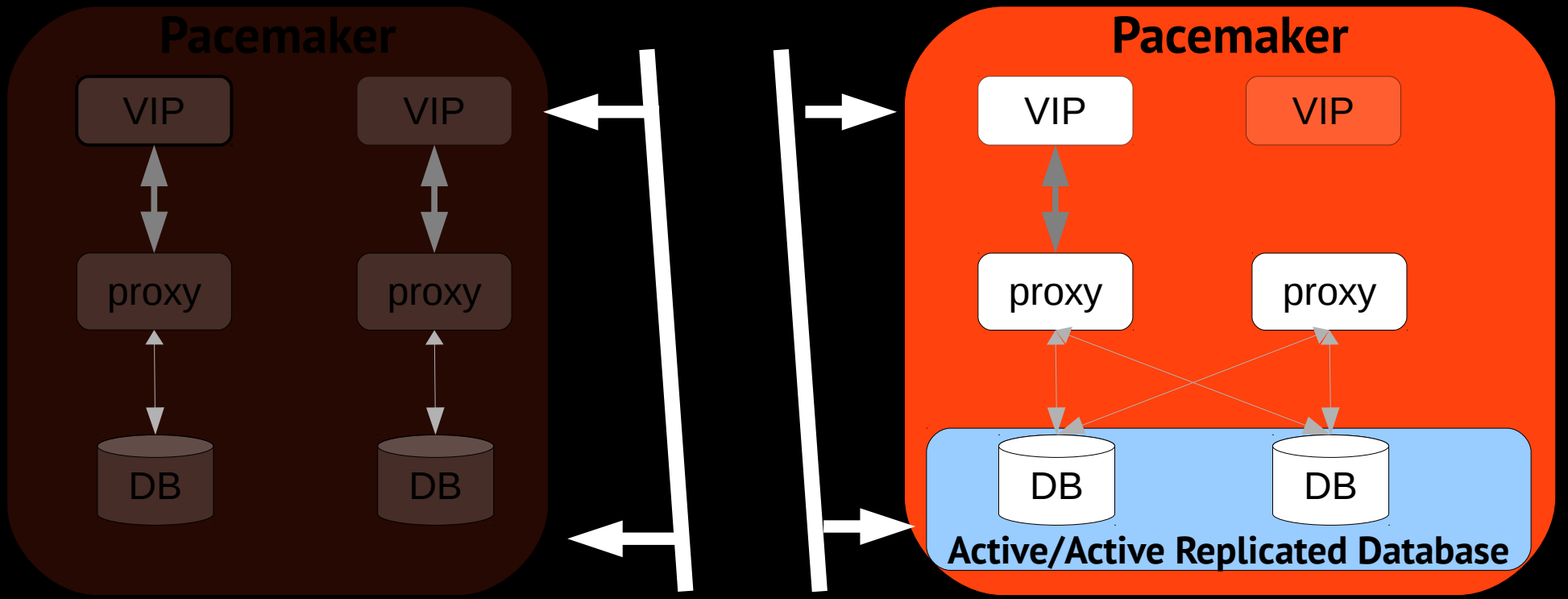
Pacemaker: System Level HA

- ***System level HA*** is holistic.
- **Defines the policy** of how to recover a set of applications
- **Enforces the policy** to achieve system wide deterministic behavior.



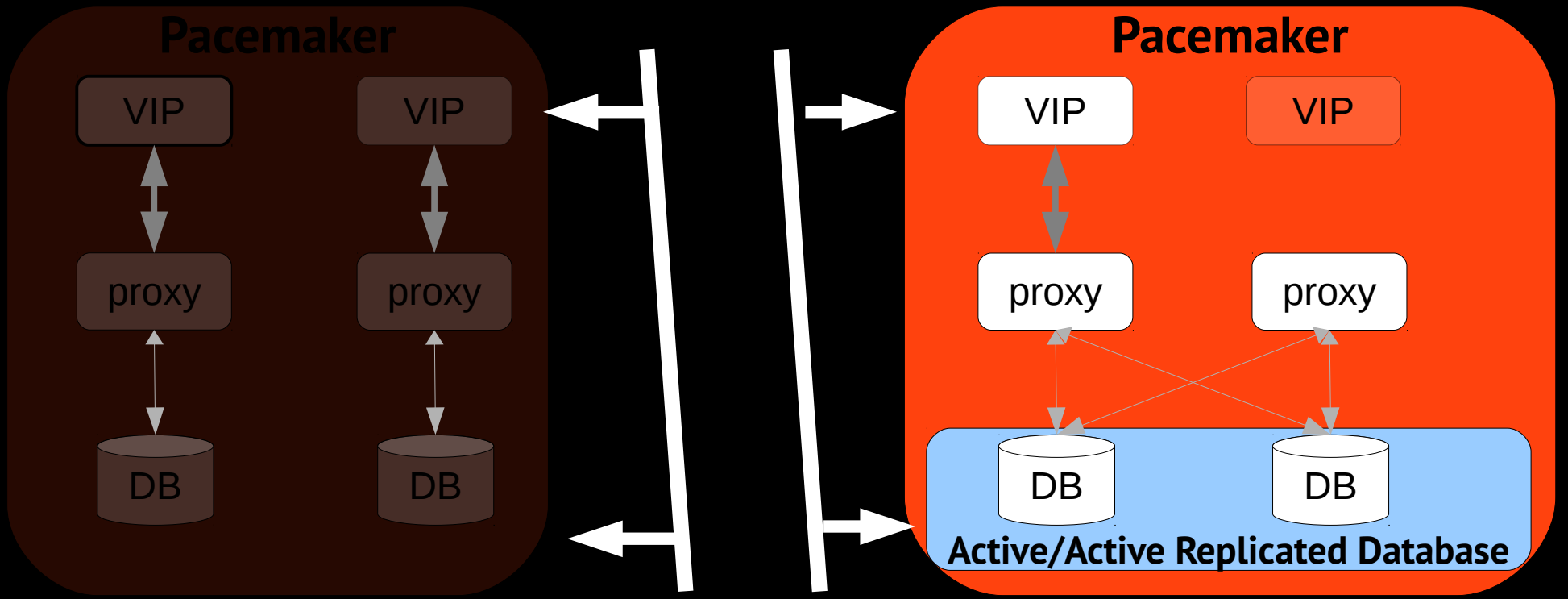
Back to the Story... How does Pacemaker Help?

- We don't question what happened to the other nodes... We know.

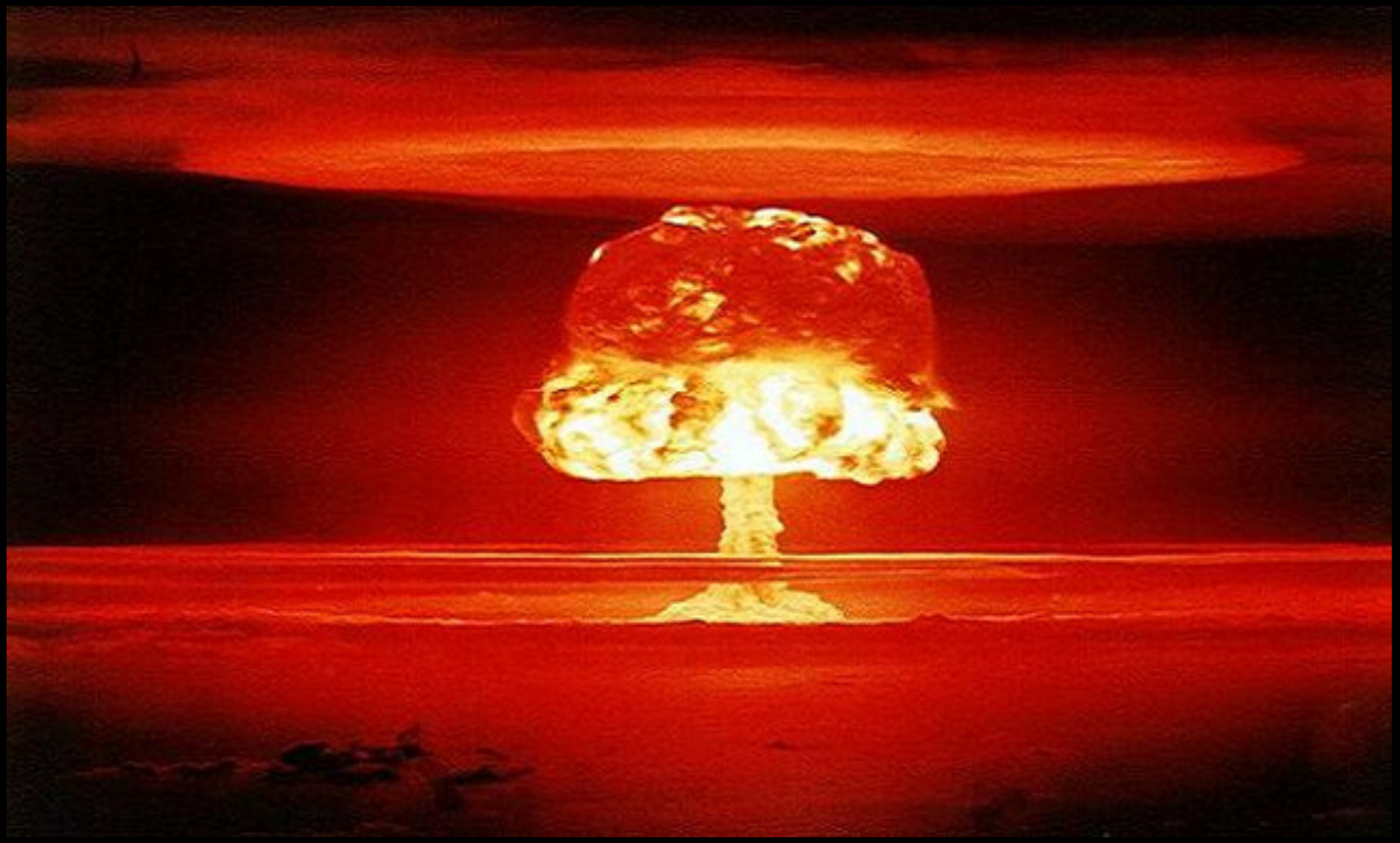


Back to the Story... How does Pacemaker Help?

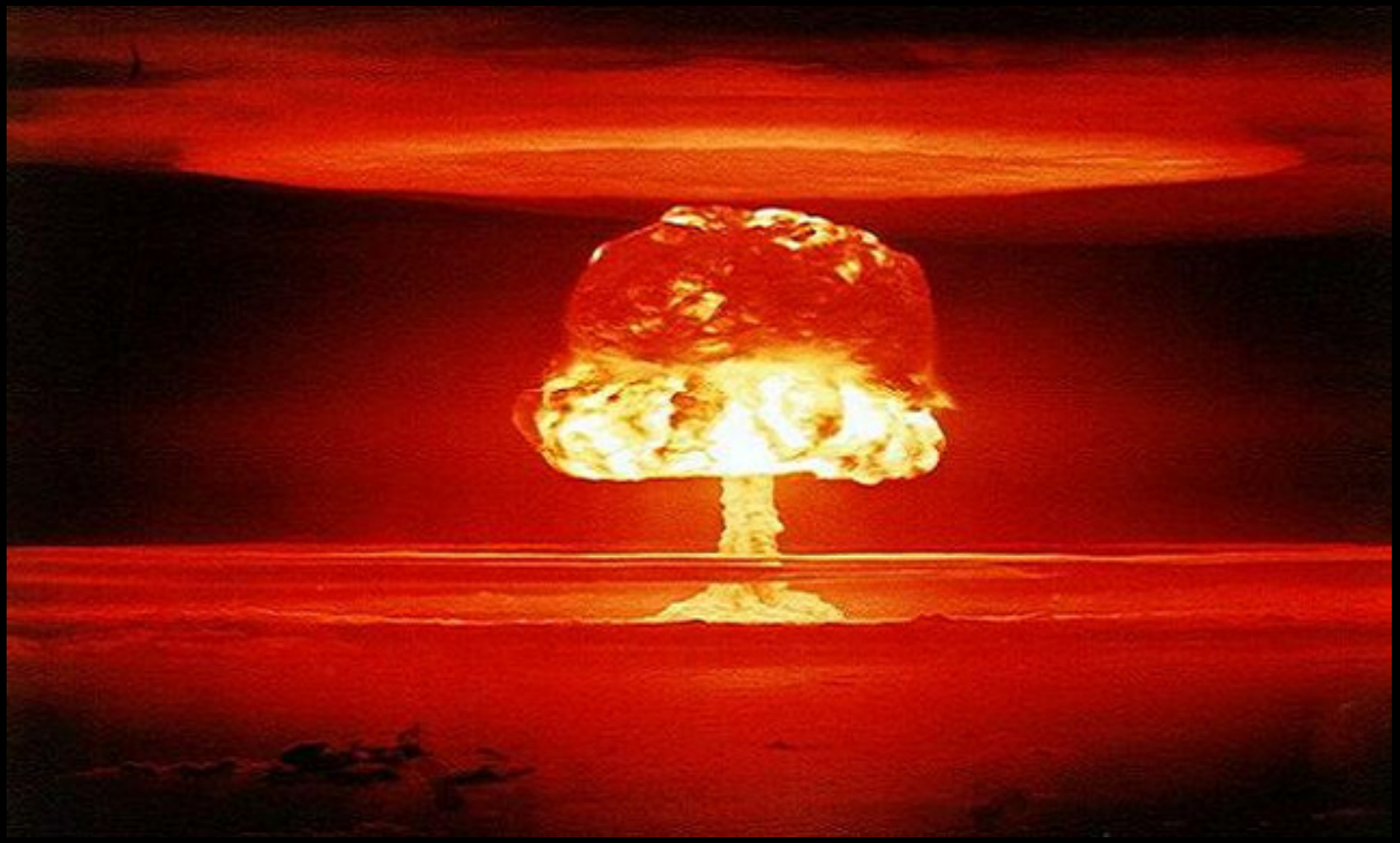
- We don't question what happened to the other nodes... We know.
- **And how do we know this?**



Introducing STONITH



Introducing STONITH

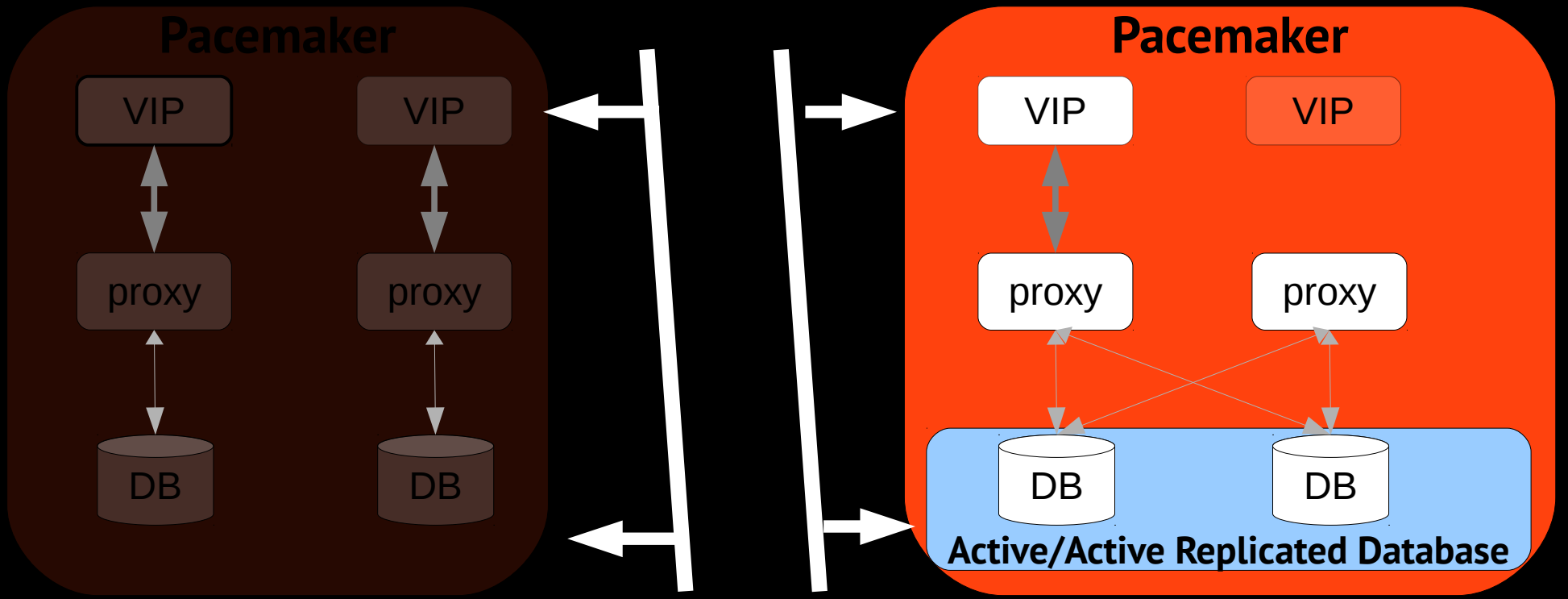


Shoot the Other Node in the Head

David Vossel <dvossel@redhat.com>

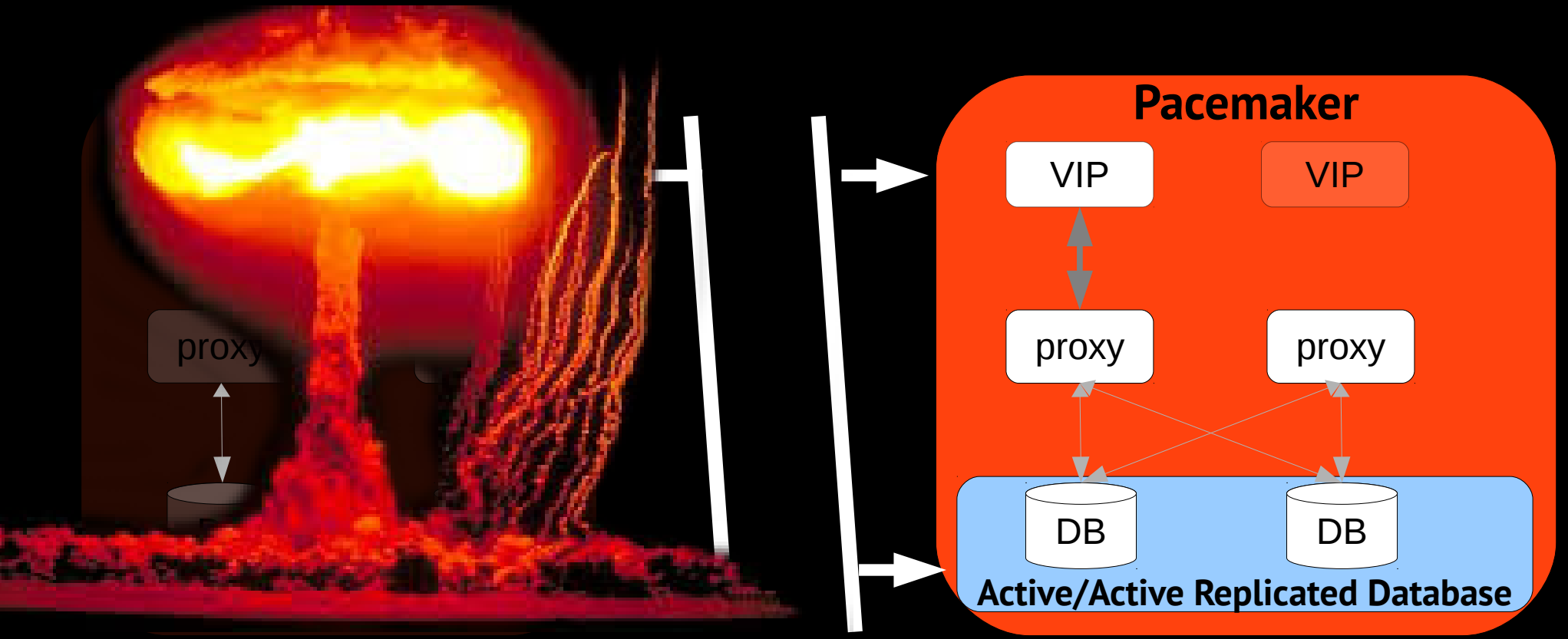
STONITH = Pacemaker's Fencing Daemon.

- Pacemaker knows the state of lost/misbehaving nodes



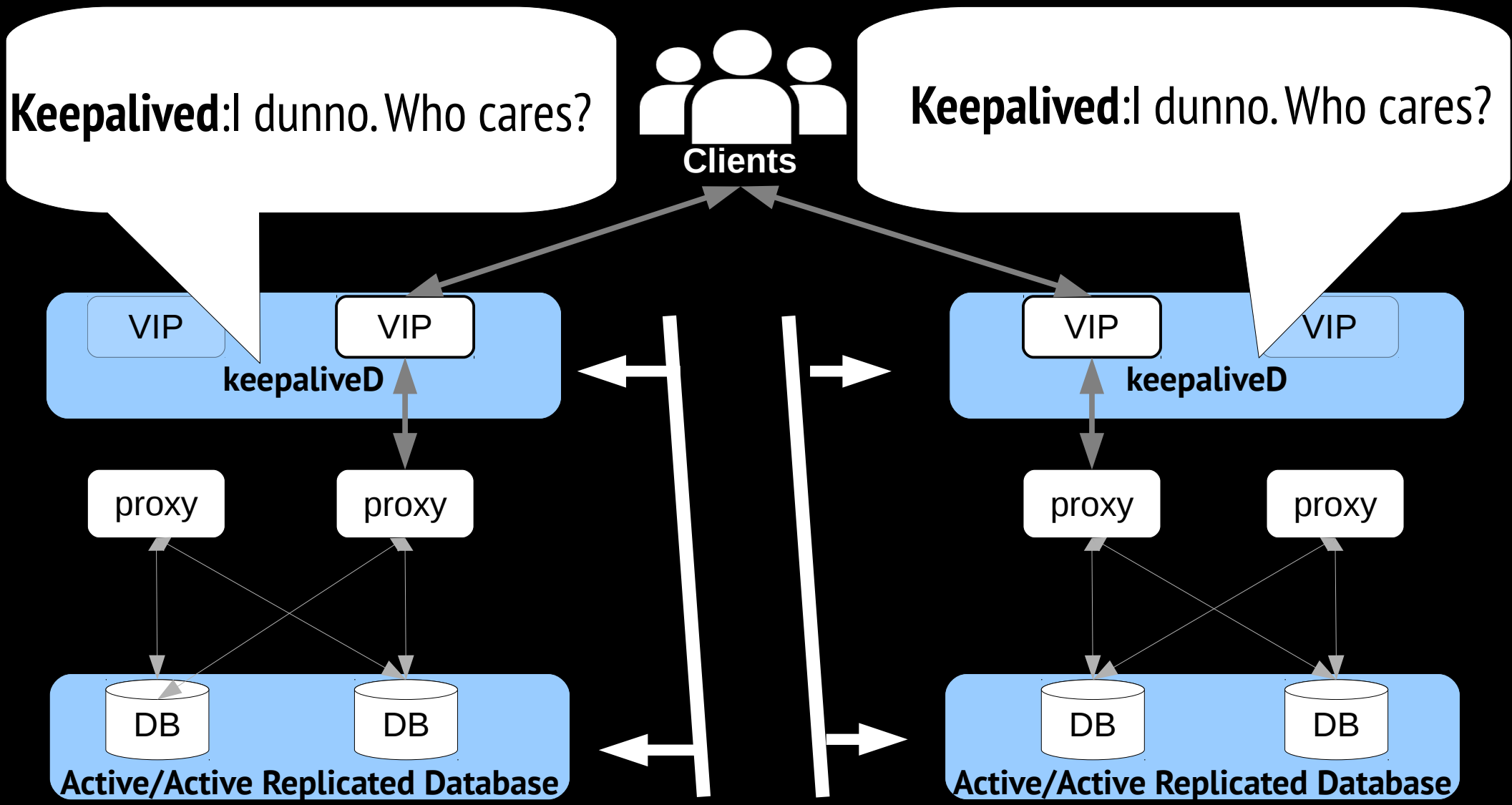
STONITH = Pacemaker's Fencing Daemon.

- Pacemaker knows the state of lost/misbehaving nodes
- Because with fencing via STONITH... that state is dead.

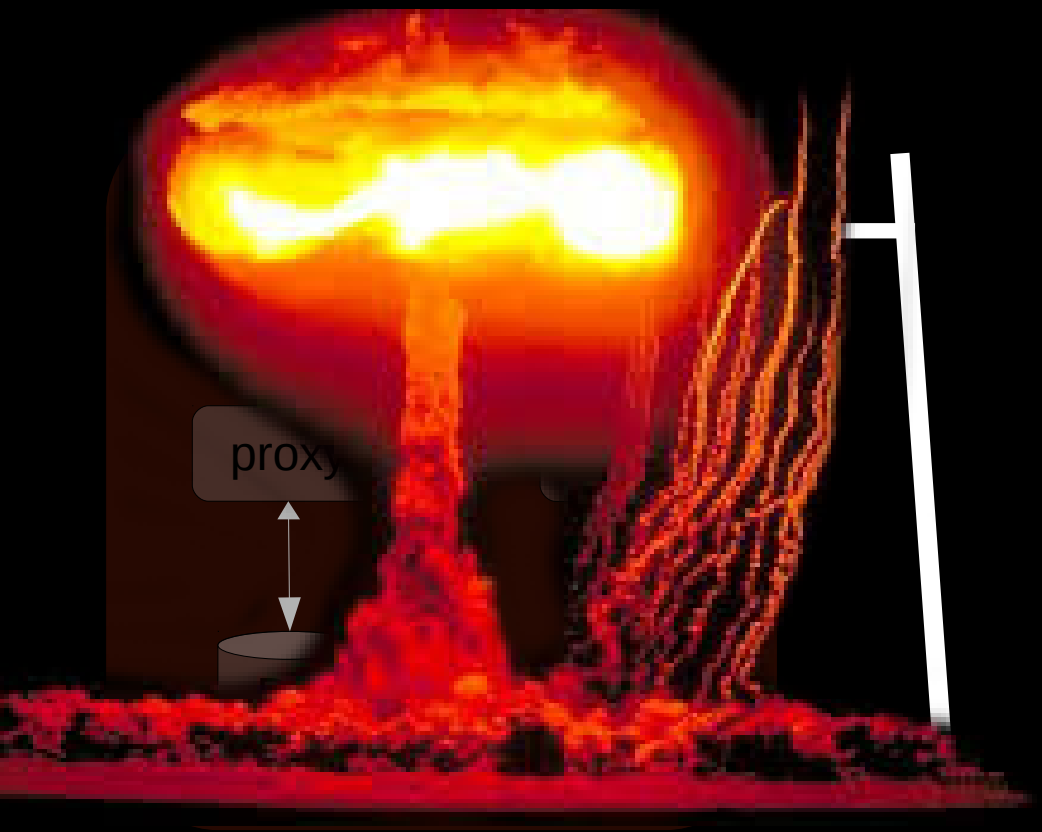


Quick Recap...

Without Pacemaker+STONITH

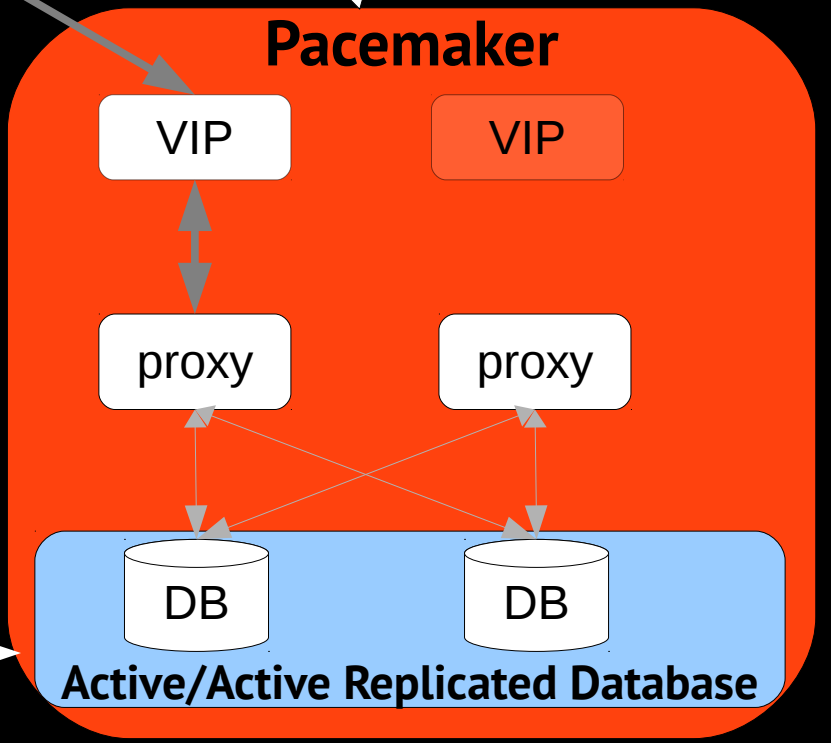


With Pacemaker+STONITH...



Clients

Pacemaker:They are dead because I killed them..



The Takeaway.

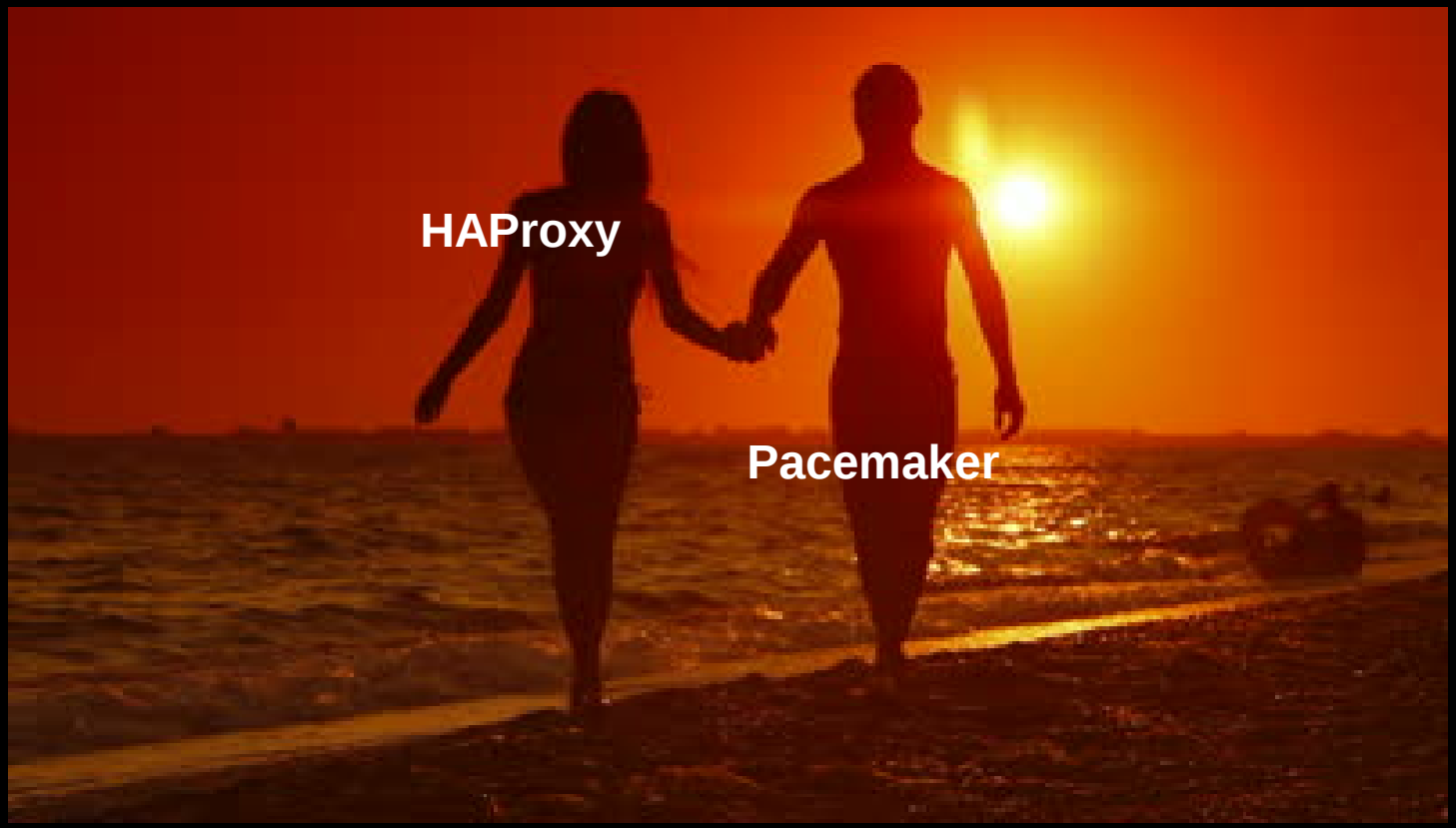
- **Pacemaker** and **load balancing** are **NOT** mutually exclusive.

The Takeaway.

- **Pacemaker** and **load balancing** are **NOT** mutually exclusive.
- **Pacemaker** and **HAProxy** are meant for one another.

The Takeaway.

- **Pacemaker** and **load balancing** are **NOT** mutually exclusive.
- **Pacemaker** and **HAProxy** are meant for one another.



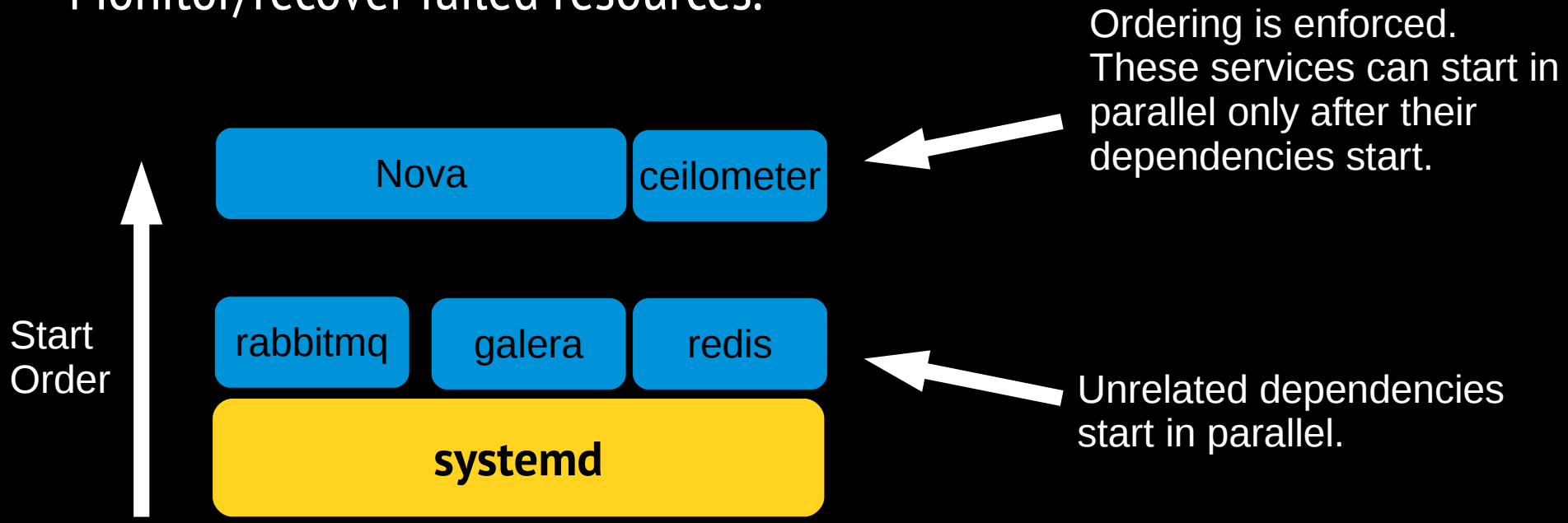
Pacemaker

The Distributed PID 1

David Vossel <dvossel@redhat.com>

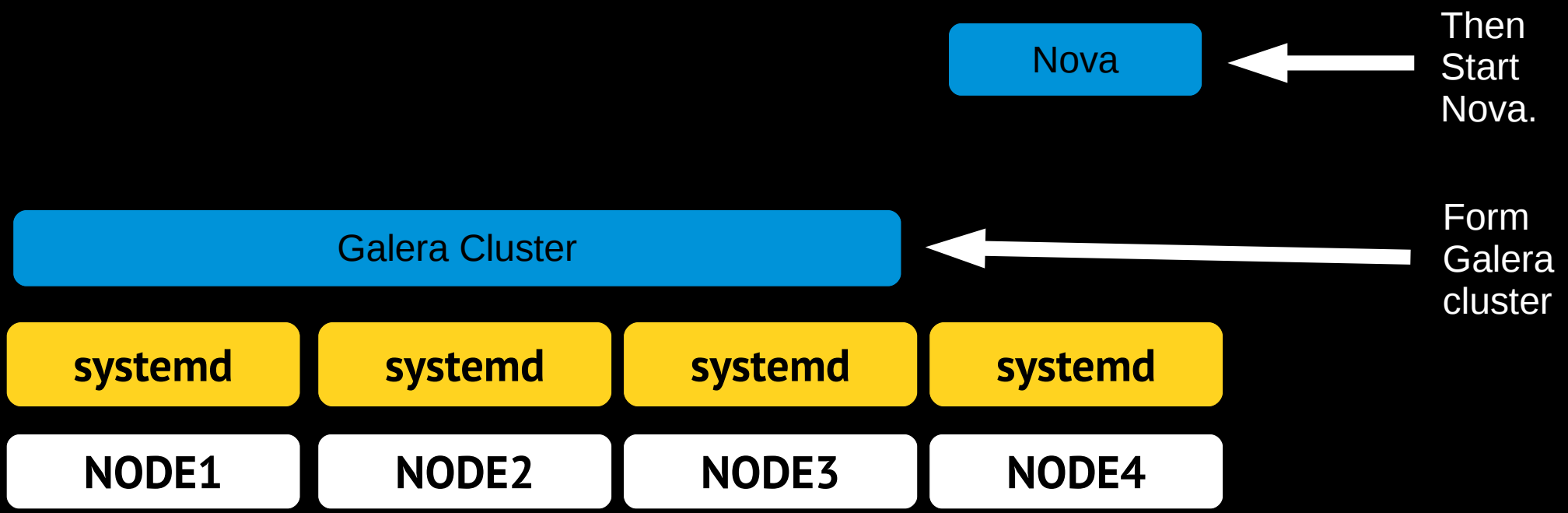
Modern PID 1's role.

- SystemD:
  - Launch services parallel
  - yet observe strict ordering between dependent services.
  - Monitor/recover failed resources.



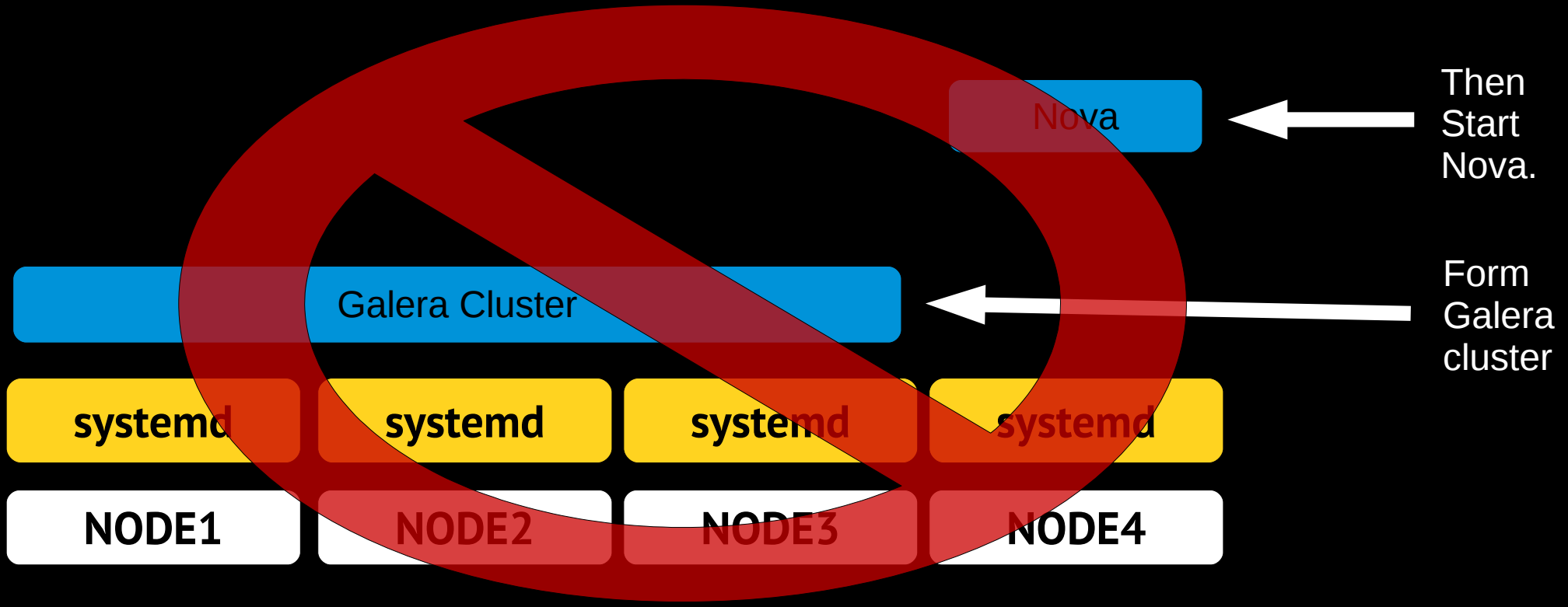
The Problem

- OpenStack services are not isolated to a local machine.



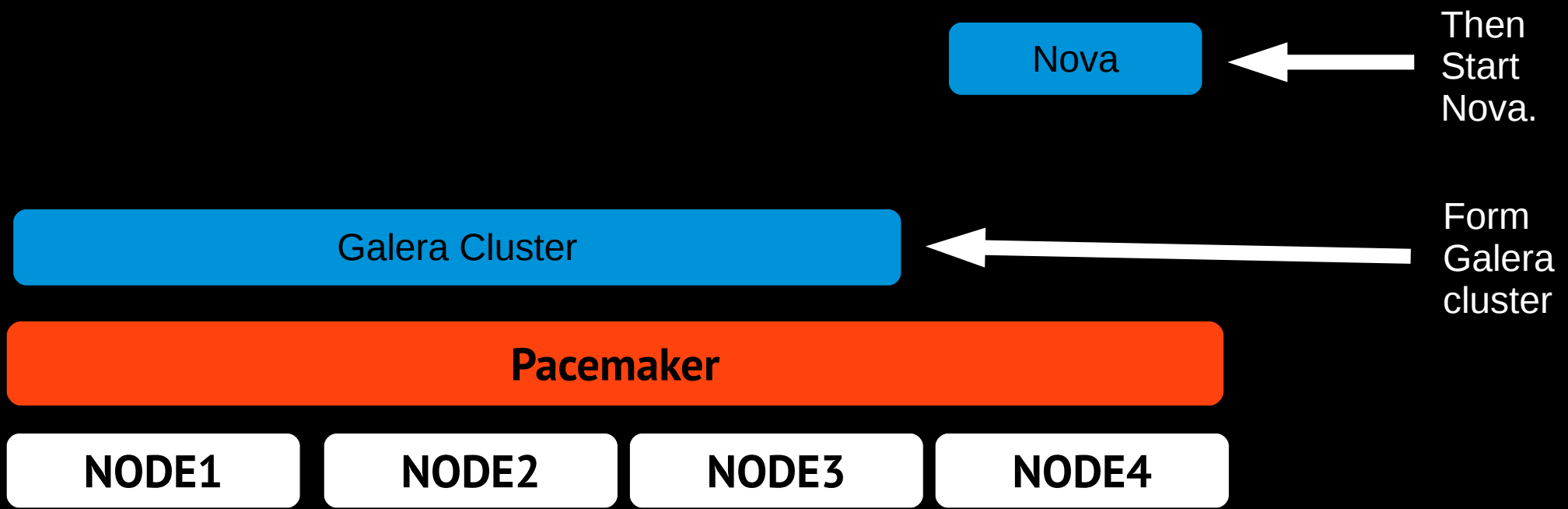
The Problem

- OpenStack services are not isolated to a local machine.
- SystemD Can't coordinate this.



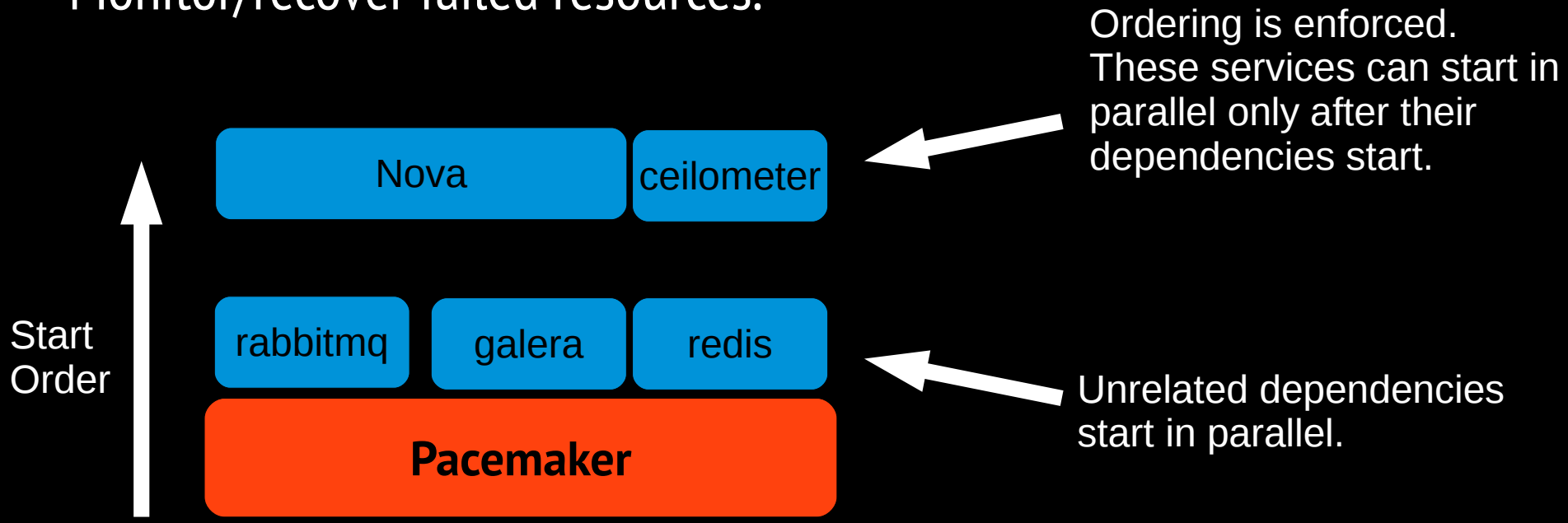
The Fix.

- But Pacemaker can
- because Pacemaker is distributed.



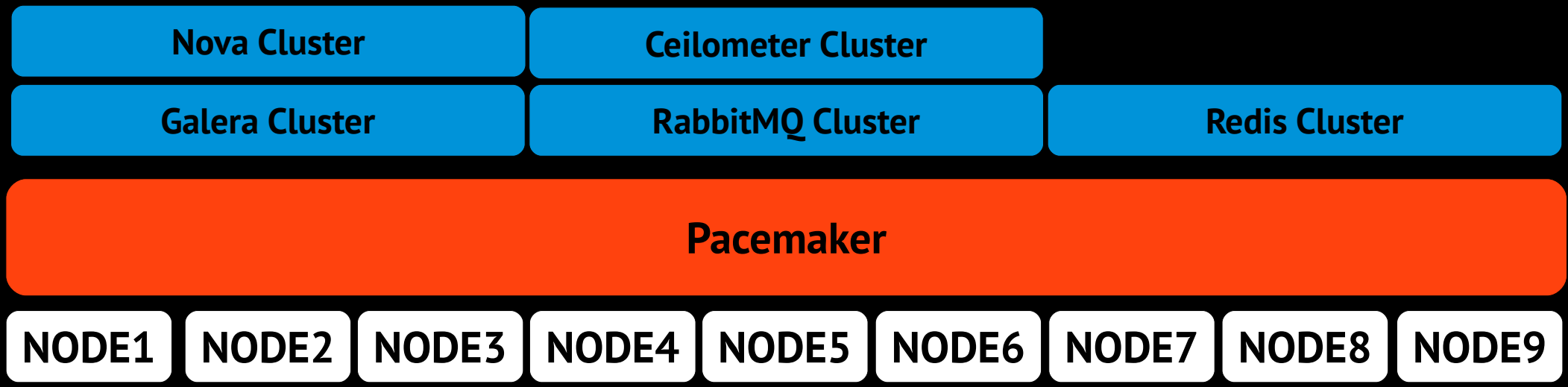
The Fix.

- Pacemaker, just like systemd, can...
  - Launch services parallel
  - yet observe strict ordering between dependent services.
  - Monitor/recover failed resources.



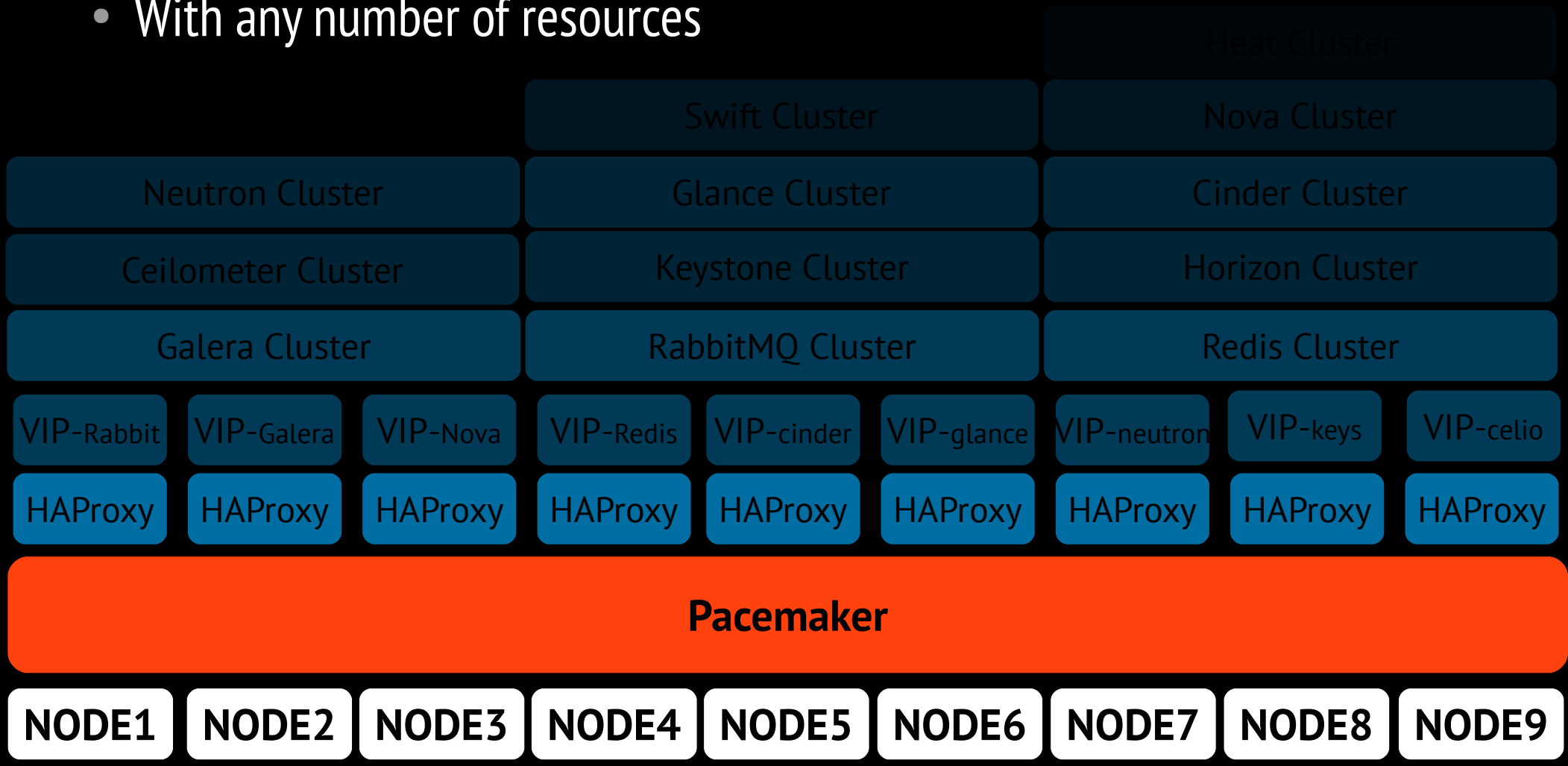
The Fix.

- Except pacemaker can coordinate this across any number of nodes.



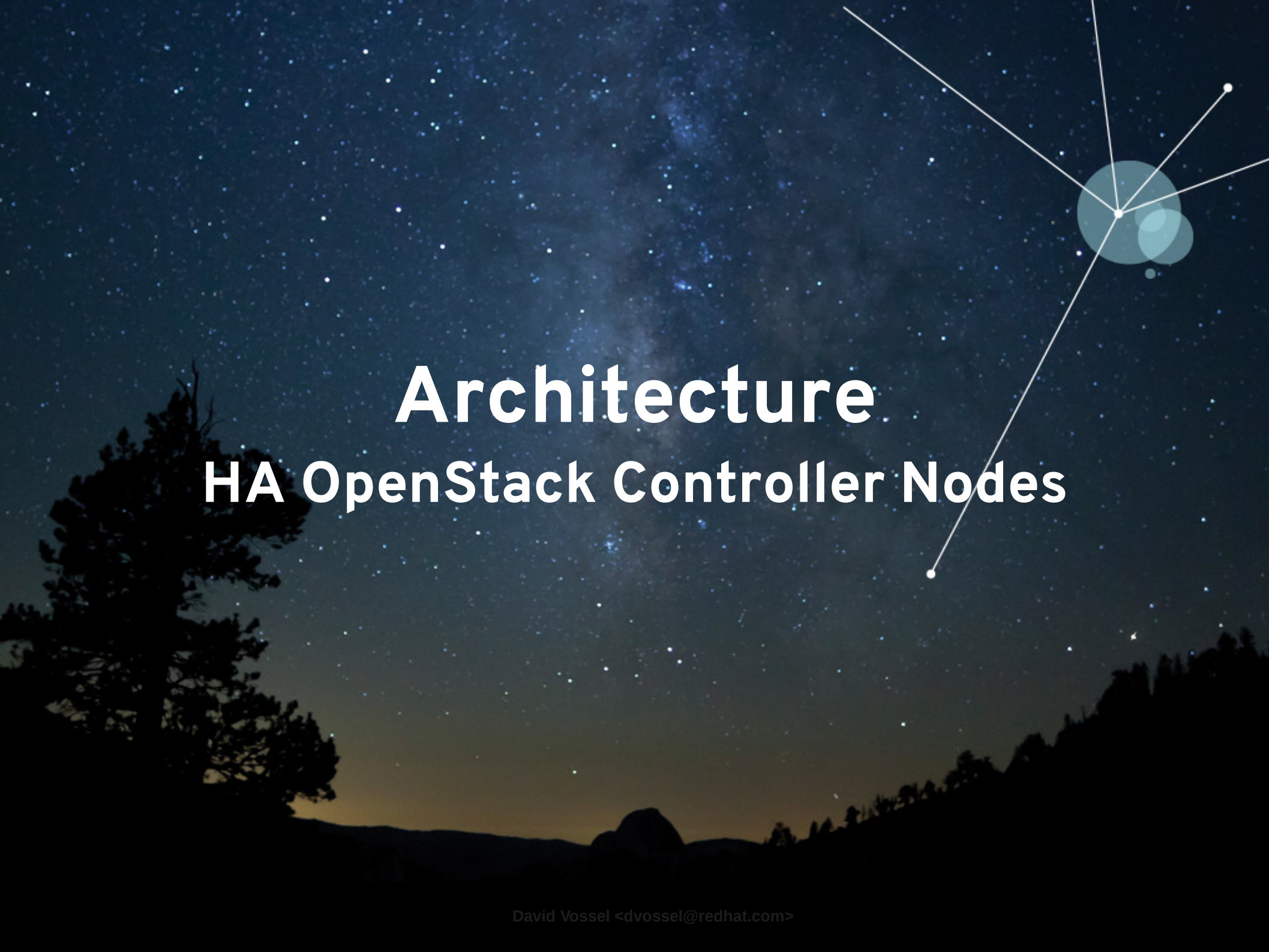
The Fix.

- Except pacemaker can coordinate this across any number of nodes.
- With any number of resources



Resource Constraints

- Pacemaker has unique capabilities for managing resources and modeling complex resource dependencies.
- Examples:
  - Start resource X then start resource Y
  - Colocate resource X with resource Y
  - Resource X prefers node A over node B
  - Resource X prefers node A between 8am-5pm



Architecture

HA OpenStack Controller Nodes

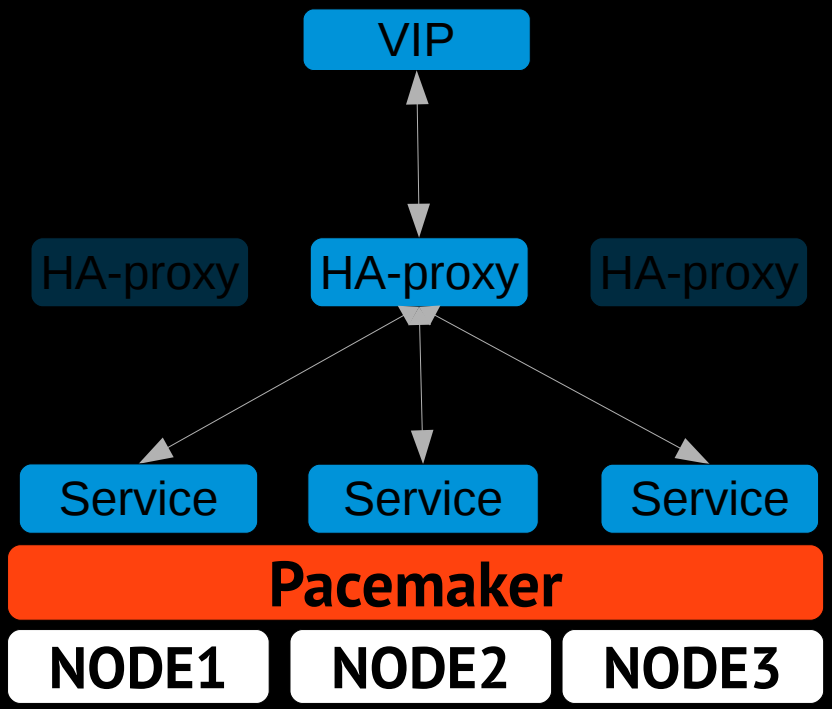
David Vossel <dvossel@redhat.com>

How it works... in one sentence

Pacemaker managing Virtual IPs + Load Balancers + Controller services to maximize the availability of OpenStack APIs

How it works... in one sentence

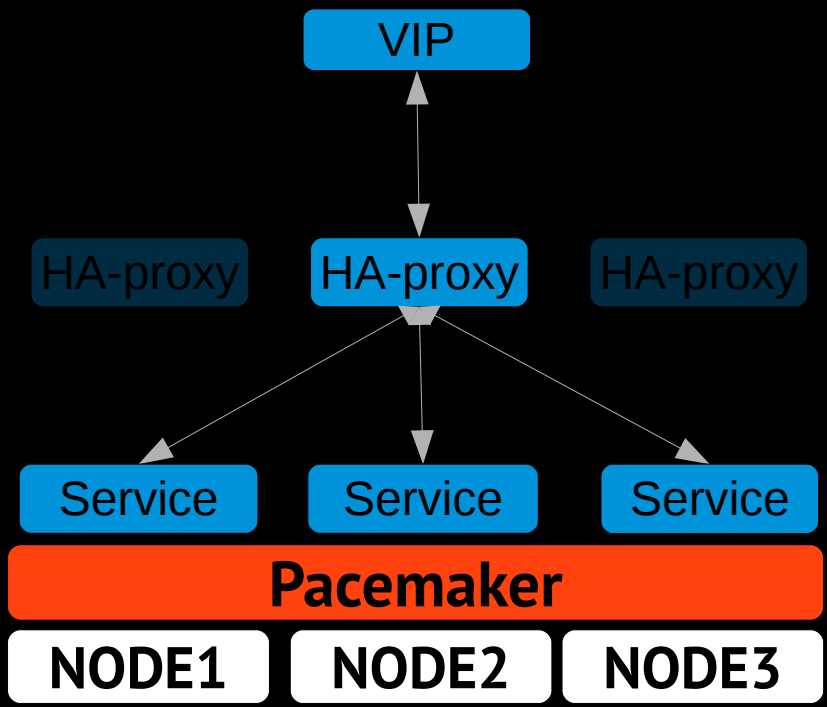
Pacemaker managing Virtual IPs + Load Balancers + Controller services to maximize the availability of OpenStack APIs



Each distributed service has a front end Virtual IP tied to a Load Balancer.

How it works... in one sentence

Pacemaker managing Virtual IPs + Load Balancers + Controller services to maximize the availability of OpenStack APIs

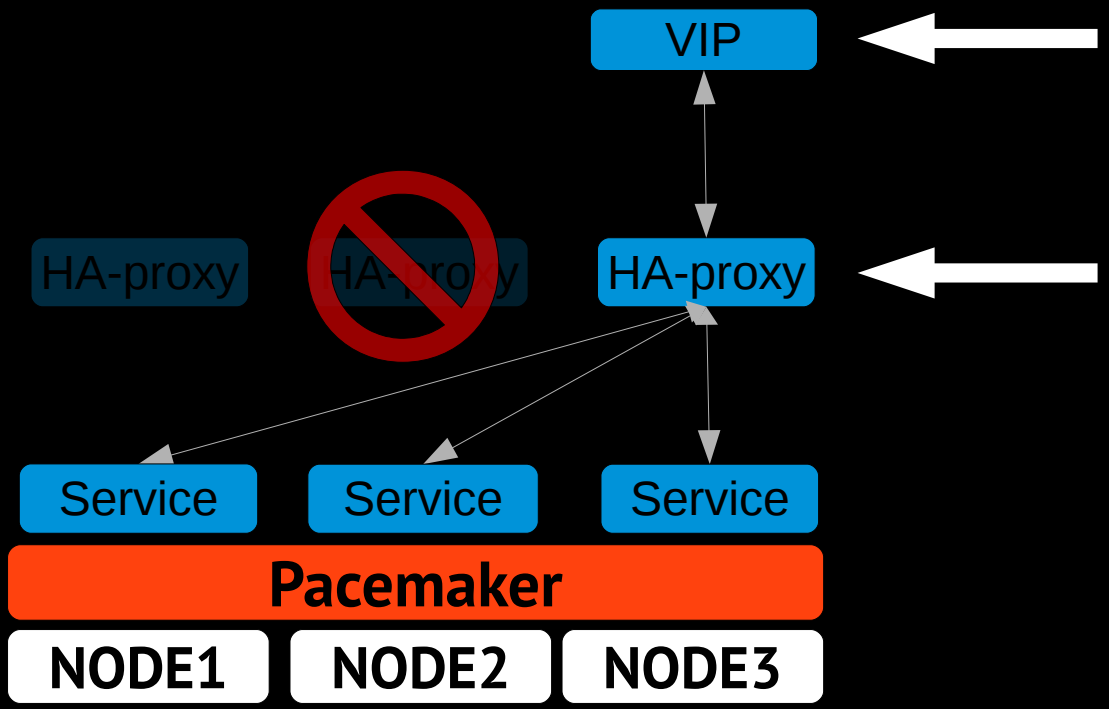


Each distributed service has a front end Virtual IP tied to a Load Balancer.

Each Load Balancer routes VIP traffic to its respective Active/Active service instances

How it works... in one sentence

Pacemaker managing Virtual IPs + Load Balancers + Controller services to maximize the availability of OpenStack APIs

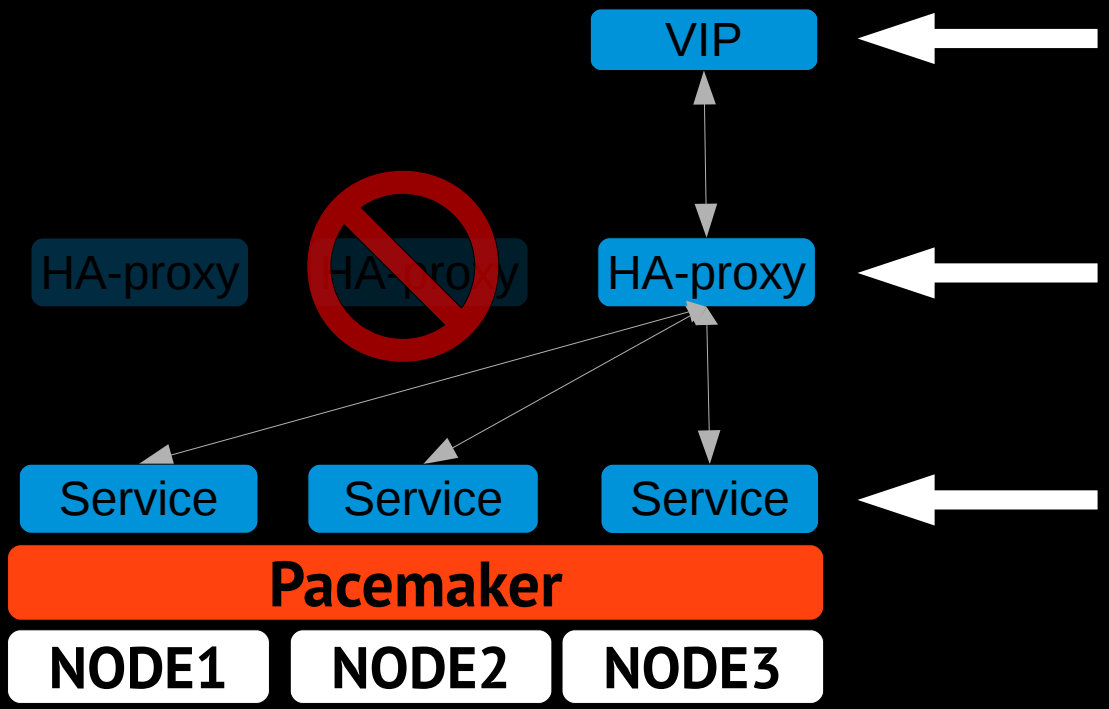


Each distributed service has a front end Virtual IP tied to a Load Balancer.

Each Load Balancer routes VIP traffic to its respective Active/Active service instances

How it works... in one sentence

Pacemaker managing Virtual IPs + Load Balancers + Controller services to maximize the availability of OpenStack APIs



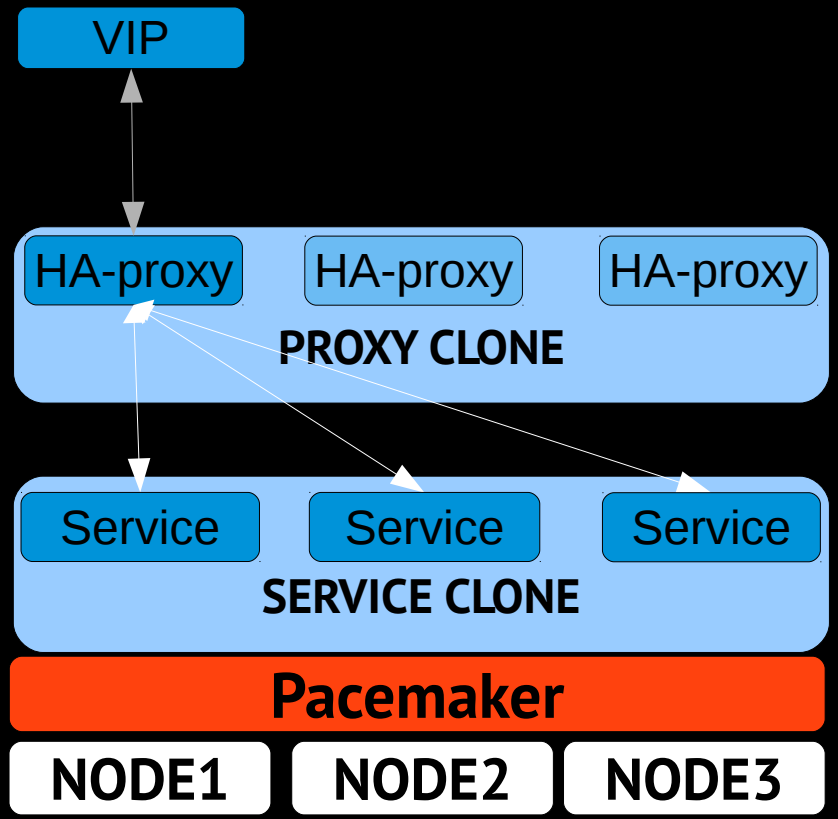
Each distributed service has a front end Virtual IP tied to a Load Balancer.

Each Load Balancer routes VIP traffic to its respective Active/Active service instances

Active/Active OpenStack Controller service... like Nova, Glance, Keystone, Galera, Redis, Rabbitmq, ect...

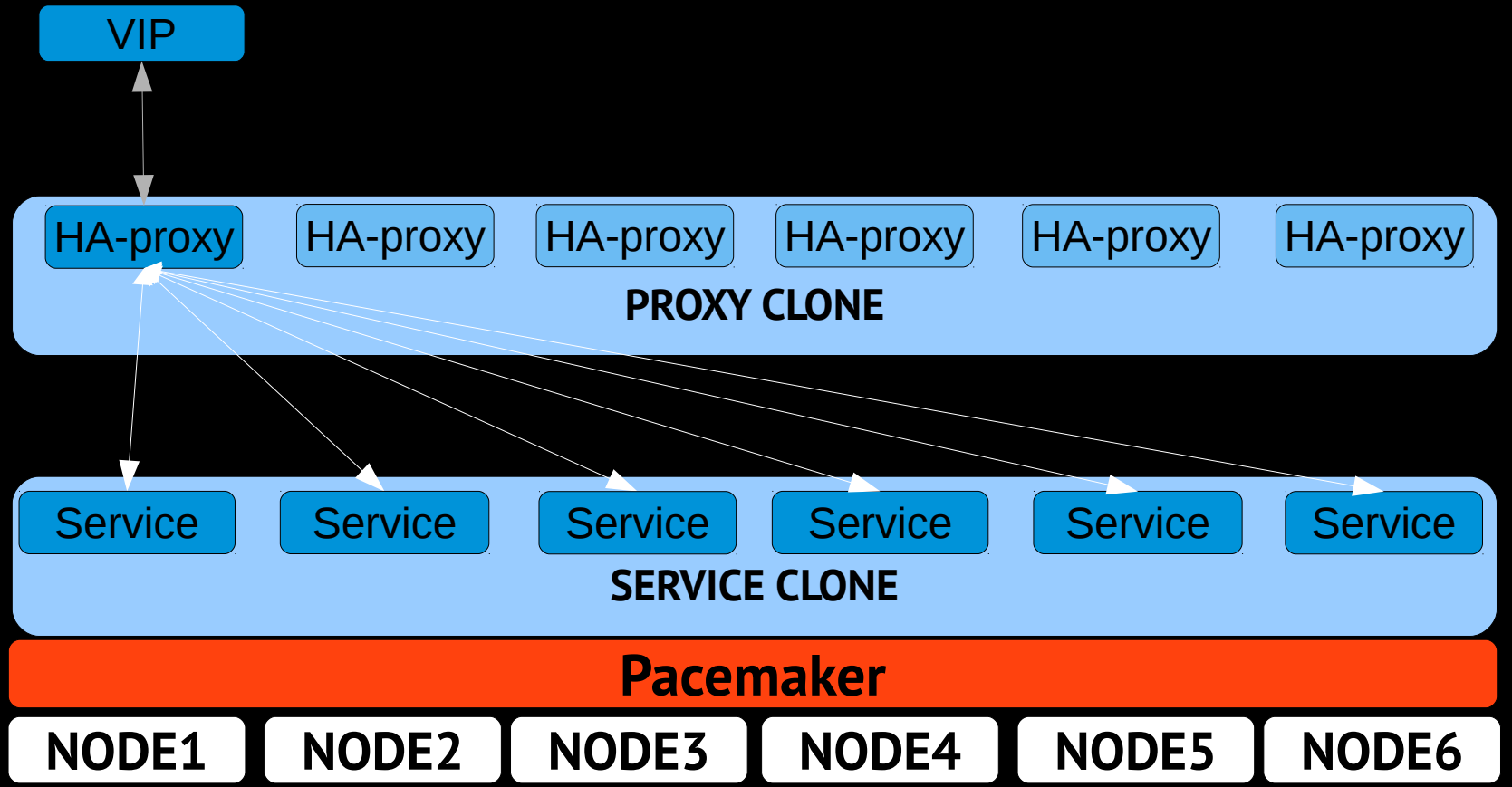
Scaling with Resource Clones

- Pacemaker's ability to clone services makes scaling trivial.
- Want more instance?



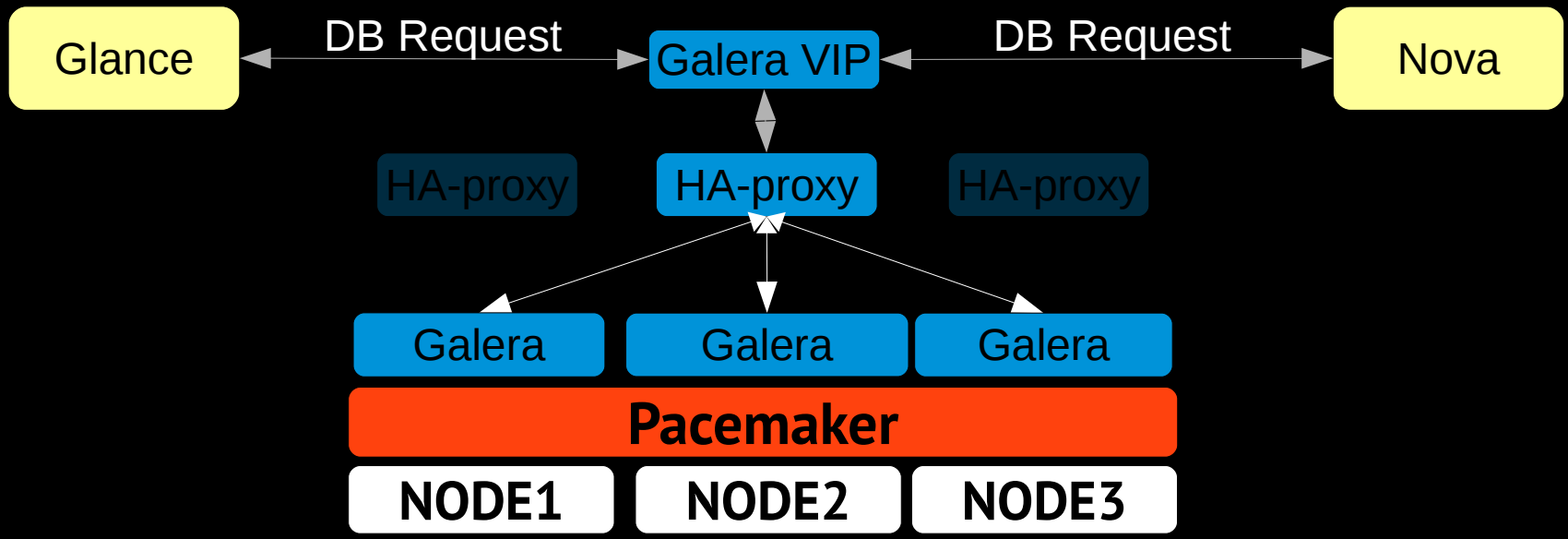
Scaling with Resource Clones

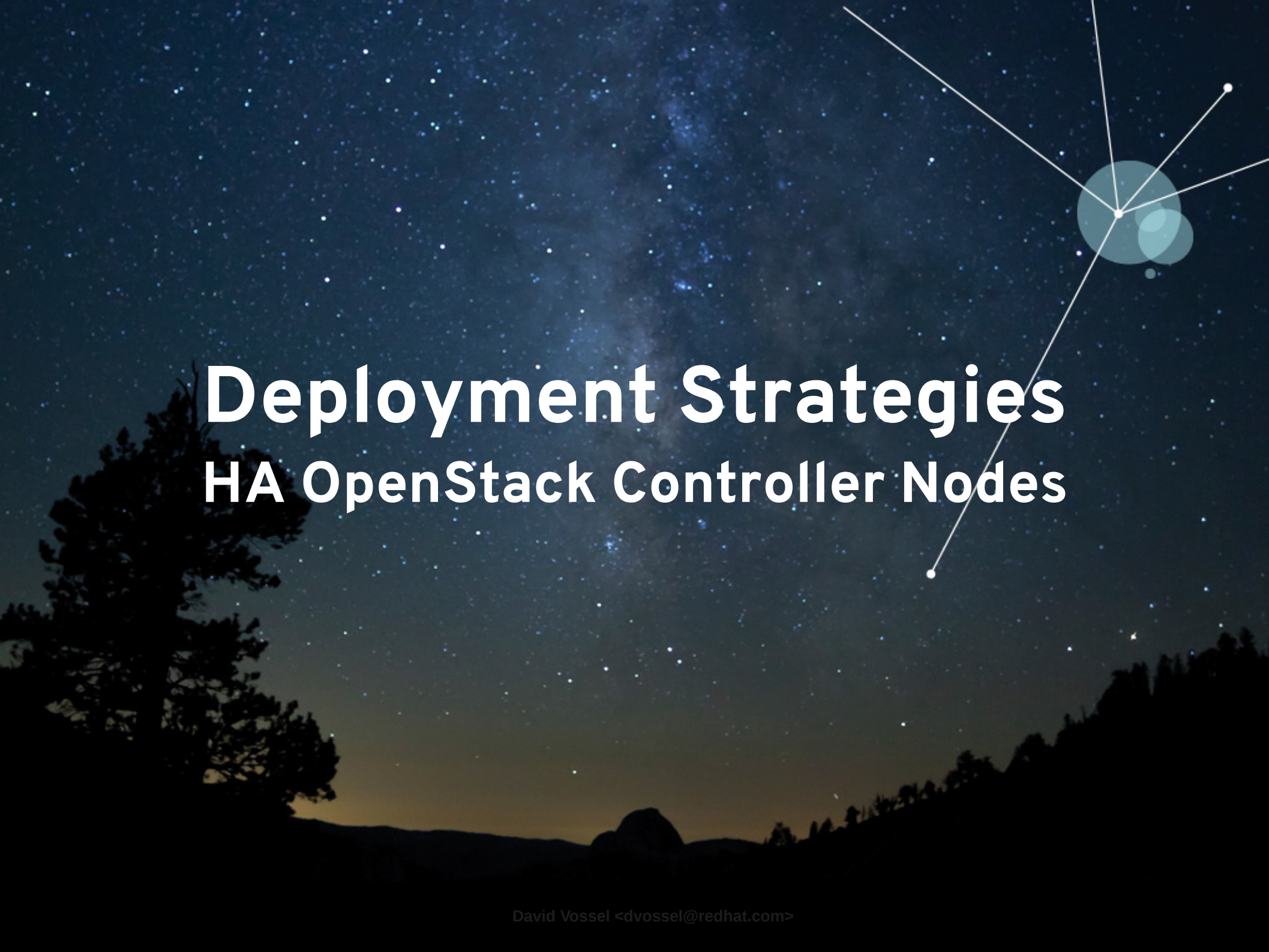
- Increment the number of clone instances pacemaker is allowed to run for a service to scale service instances.



How it works, continued...

- Services interact with one another using each service's **Virtual IP**
- Example: Both Glance and Nova need access to Galera... Galera is accessed via the **front end Virtual IP** and those requests are distributed to the **backend galera cluster.**



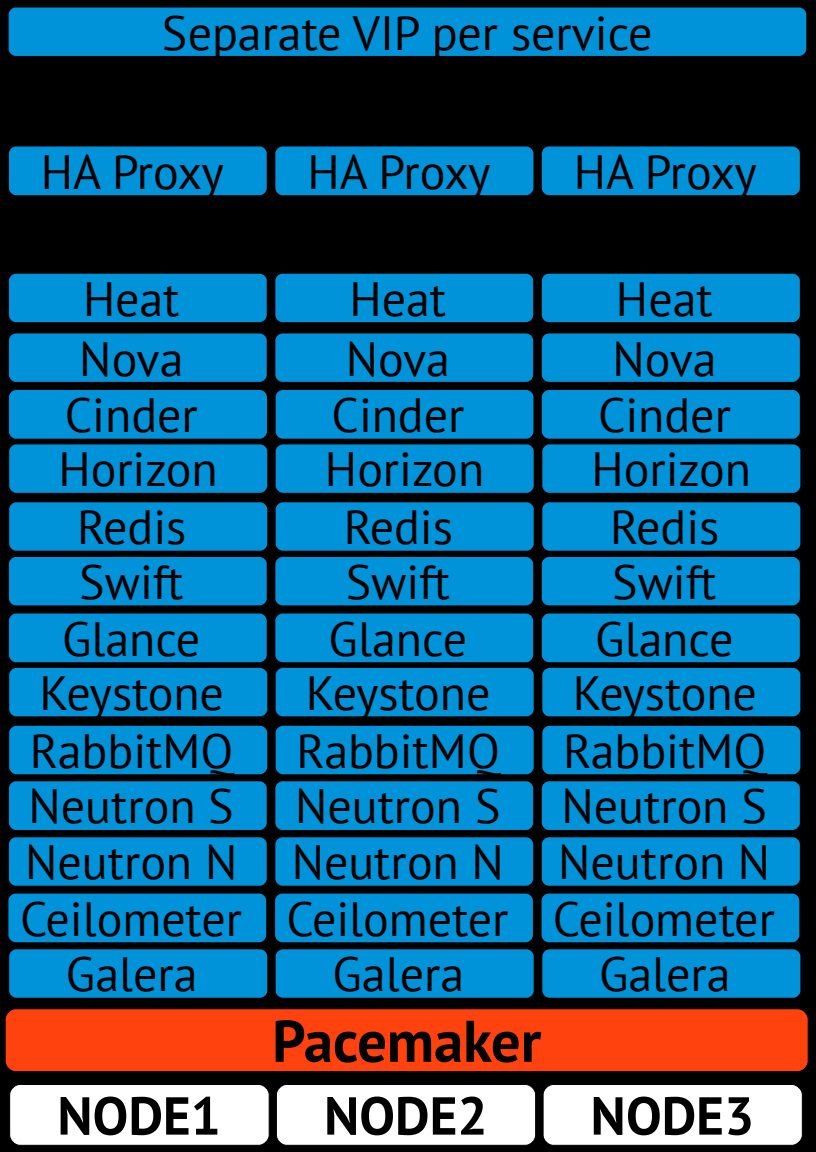


Deployment Strategies

HA OpenStack Controller Nodes

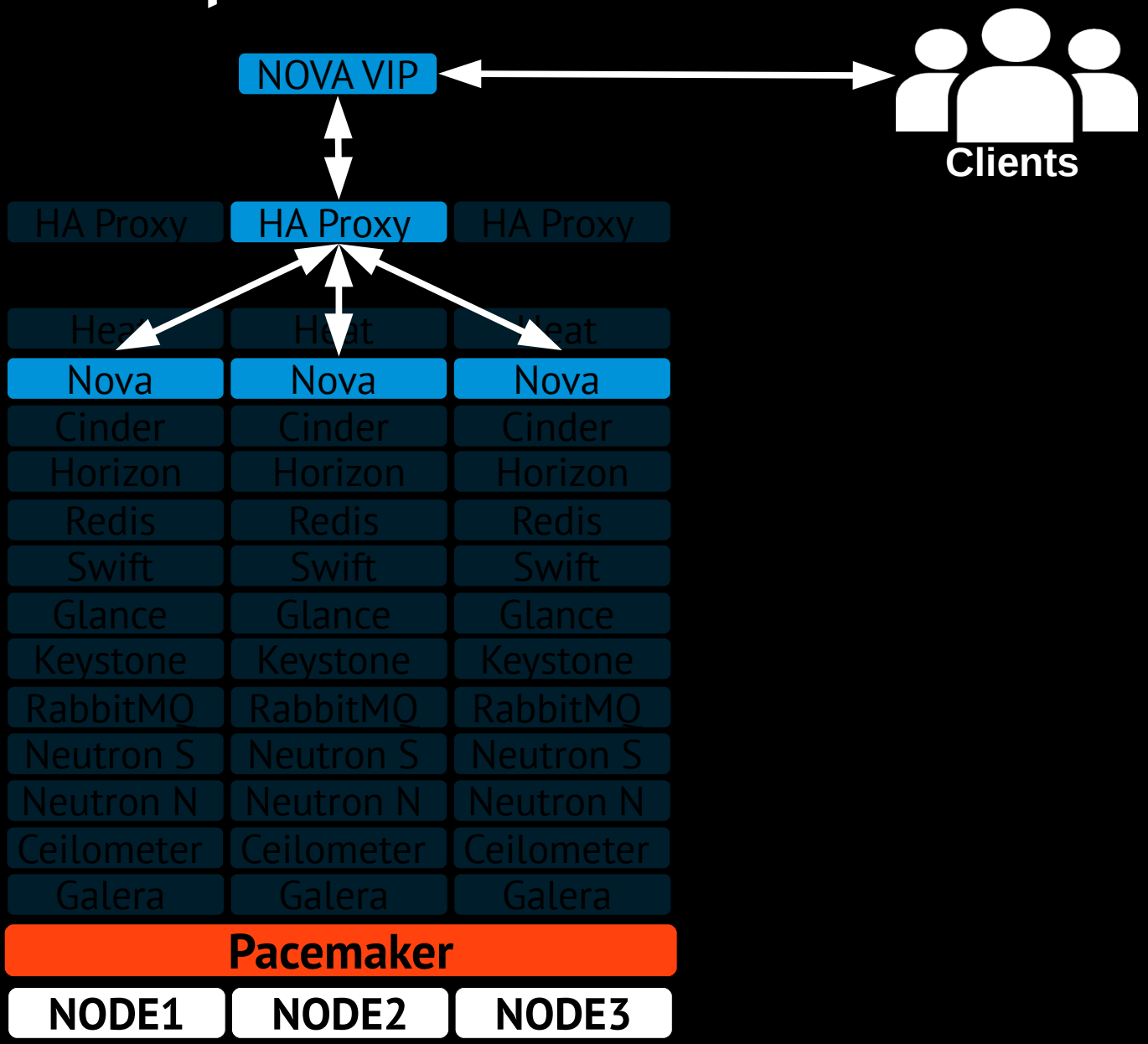
David Vossel <dvossel@redhat.com>

Collapsed Architecture

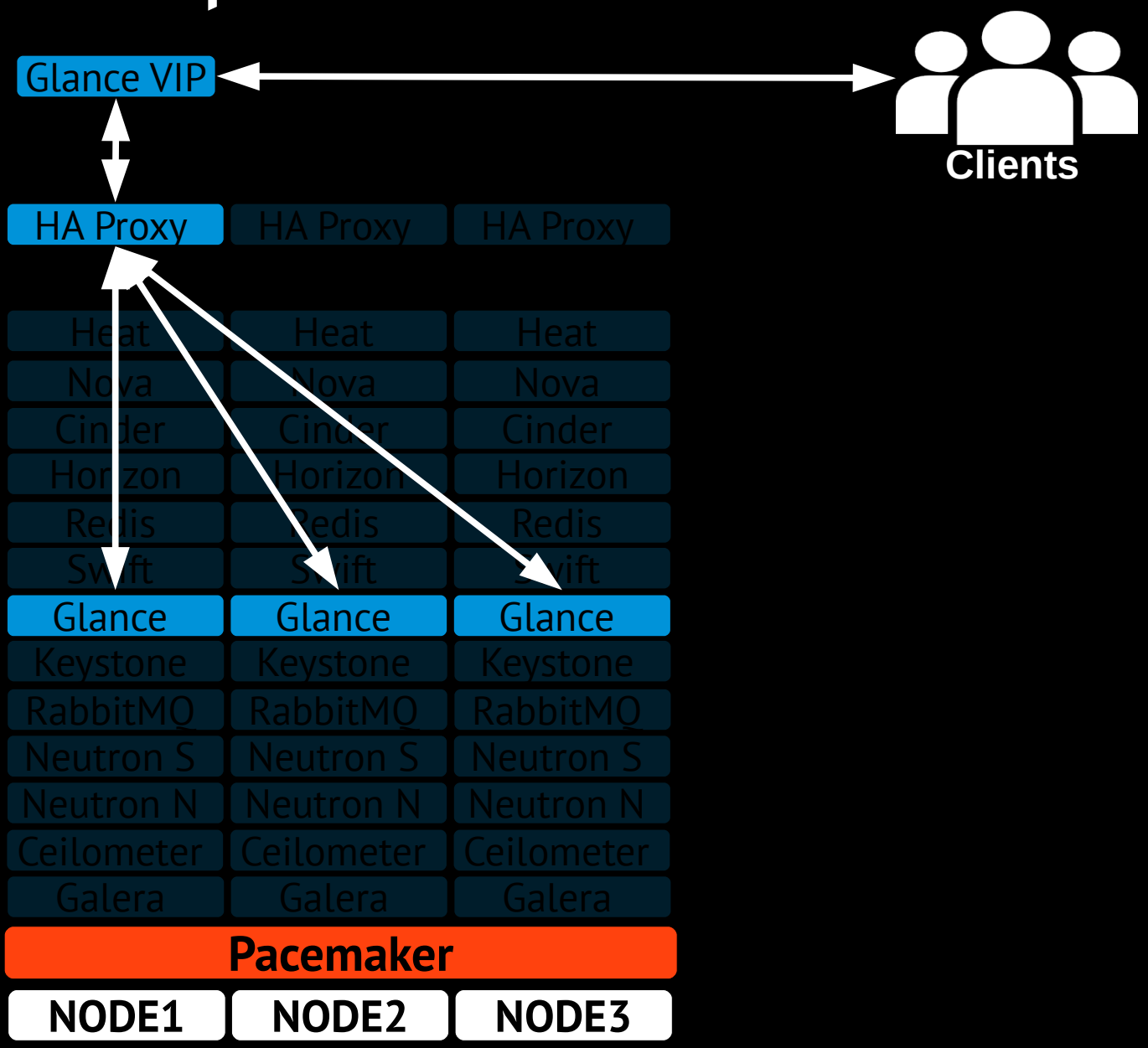


- All controller nodes run the same services.
- VIP+load balancers distribute access to APIs across cloned nodes.

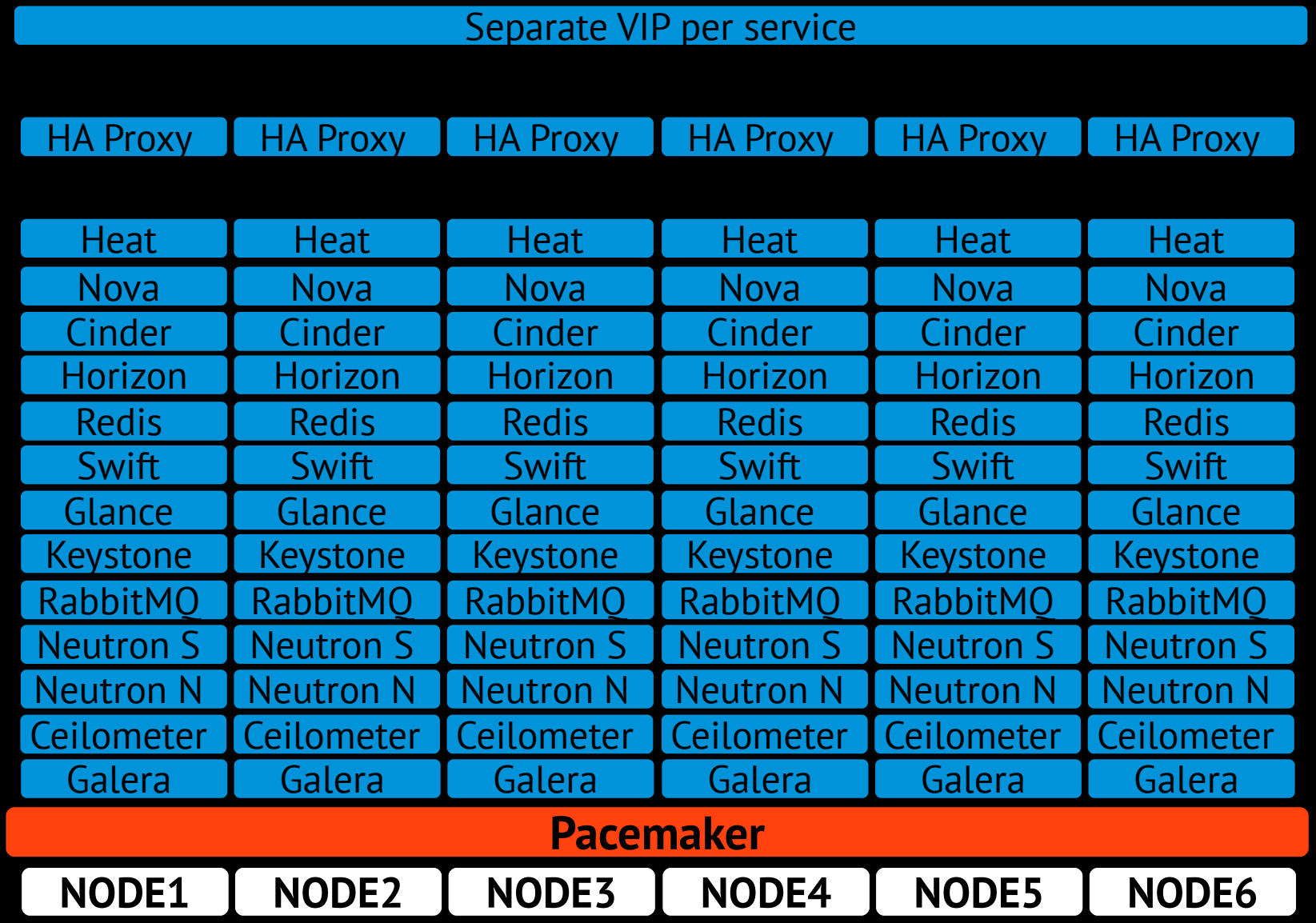
Collapsed Architecture



Collapsed Architecture



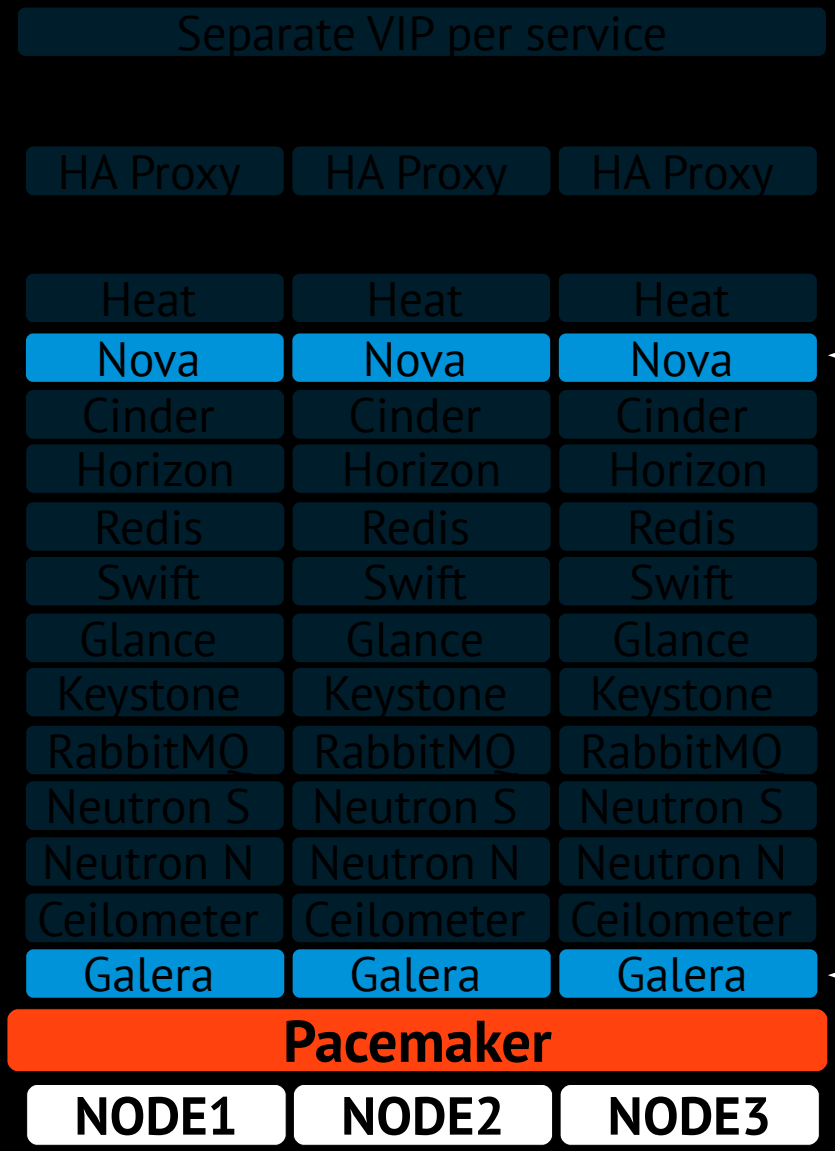
Collapsed Architecture Scaling



Startup Ordering Revisited.

Bootstrap and start Start Galera Cluster across multiple nodes.

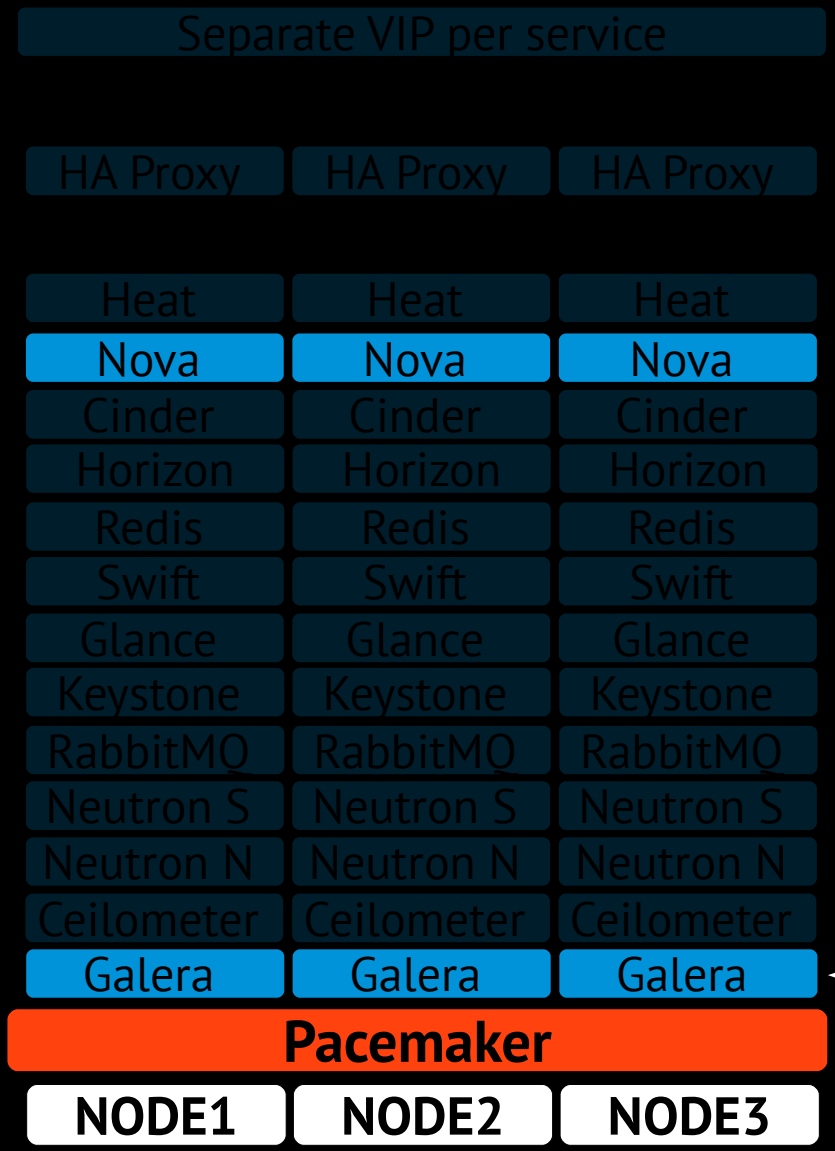
Startup Ordering Revisited.



Then Start Nova instances, which depend on an active Galera cluster.

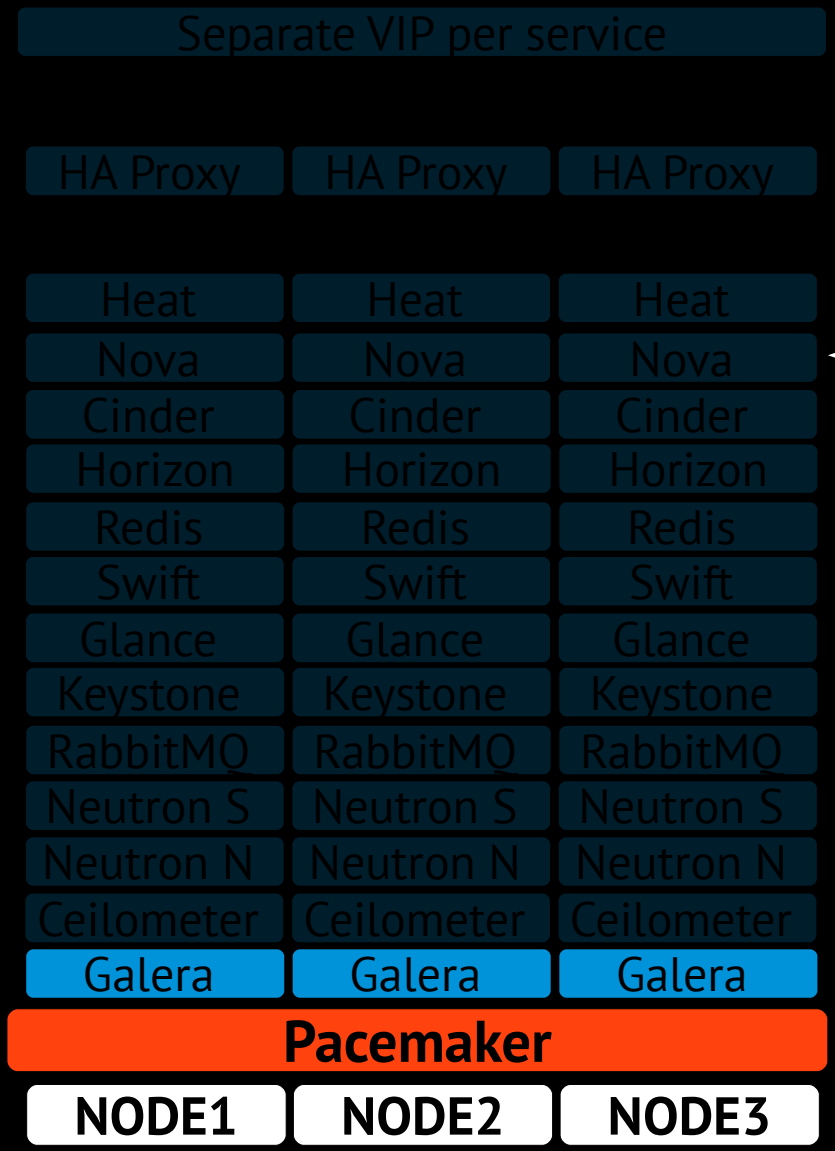
Bootstrap and start Start Galera Cluster across multiple nodes.

Stop ordering Ordering Revisited.



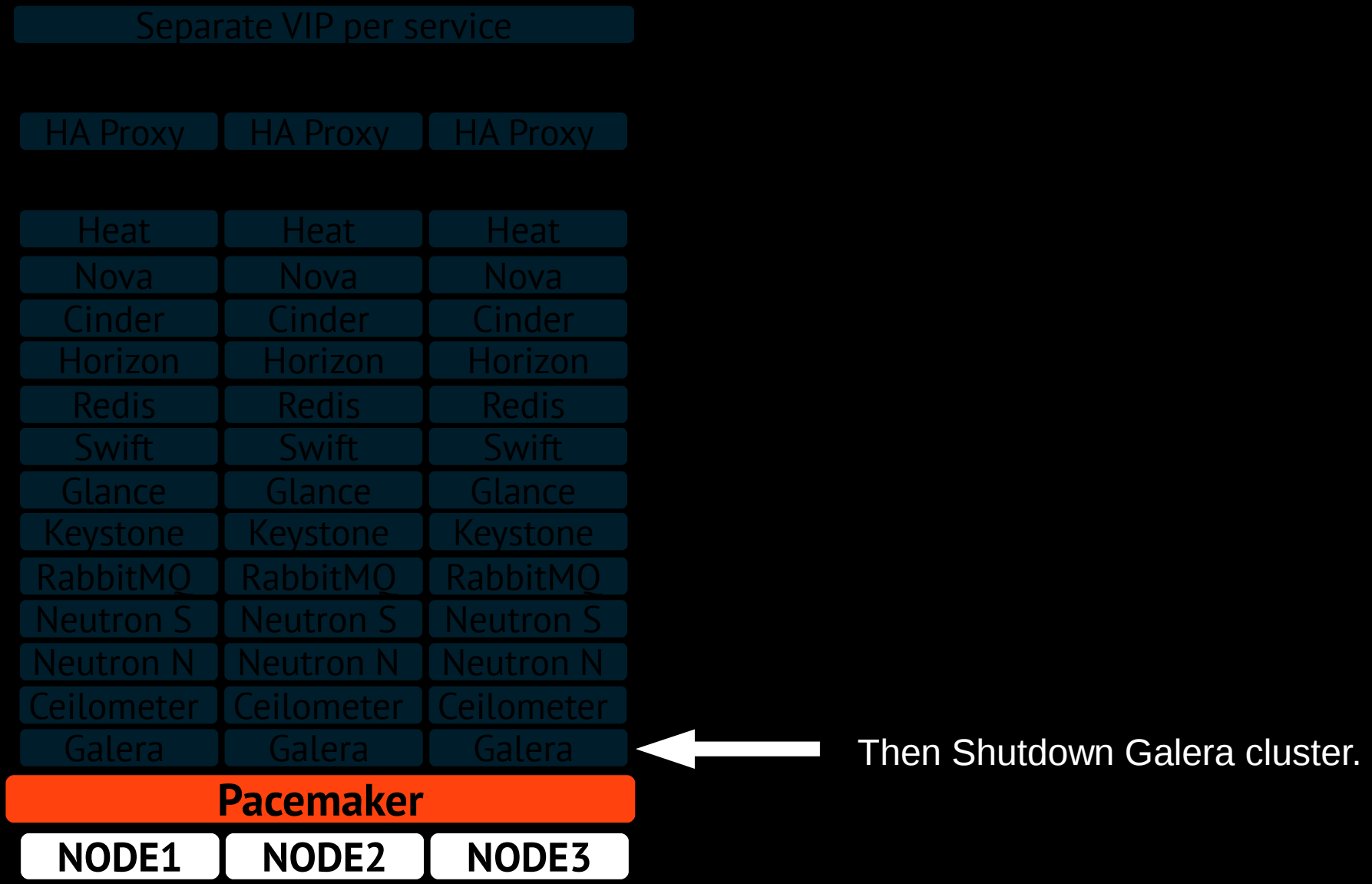
If we're shutting down galera cluster

Stop ordering Ordering Revisited.



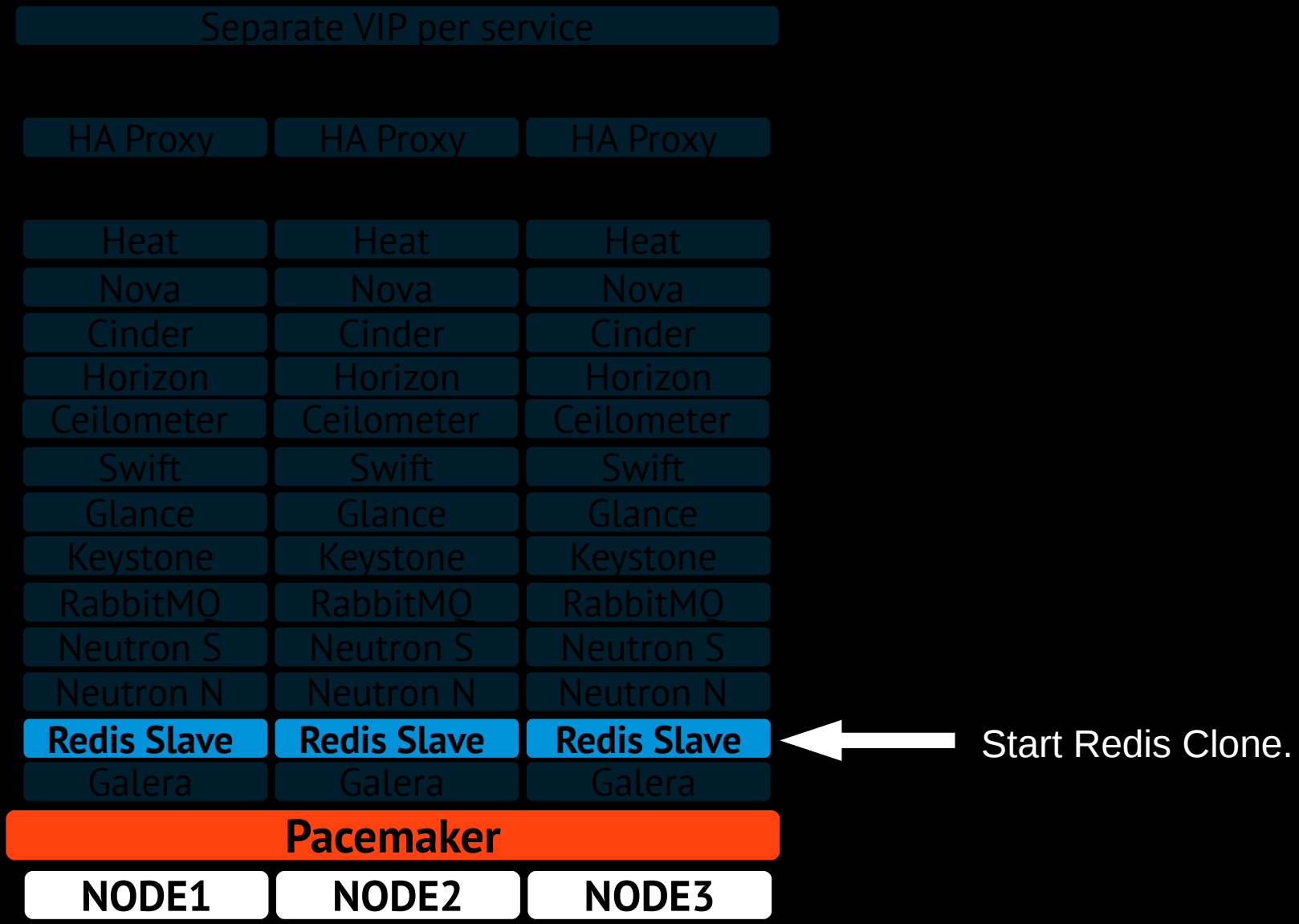
Stop everything that depends on Galera.
Like Nova...

Stop ordering Ordering Revisited.

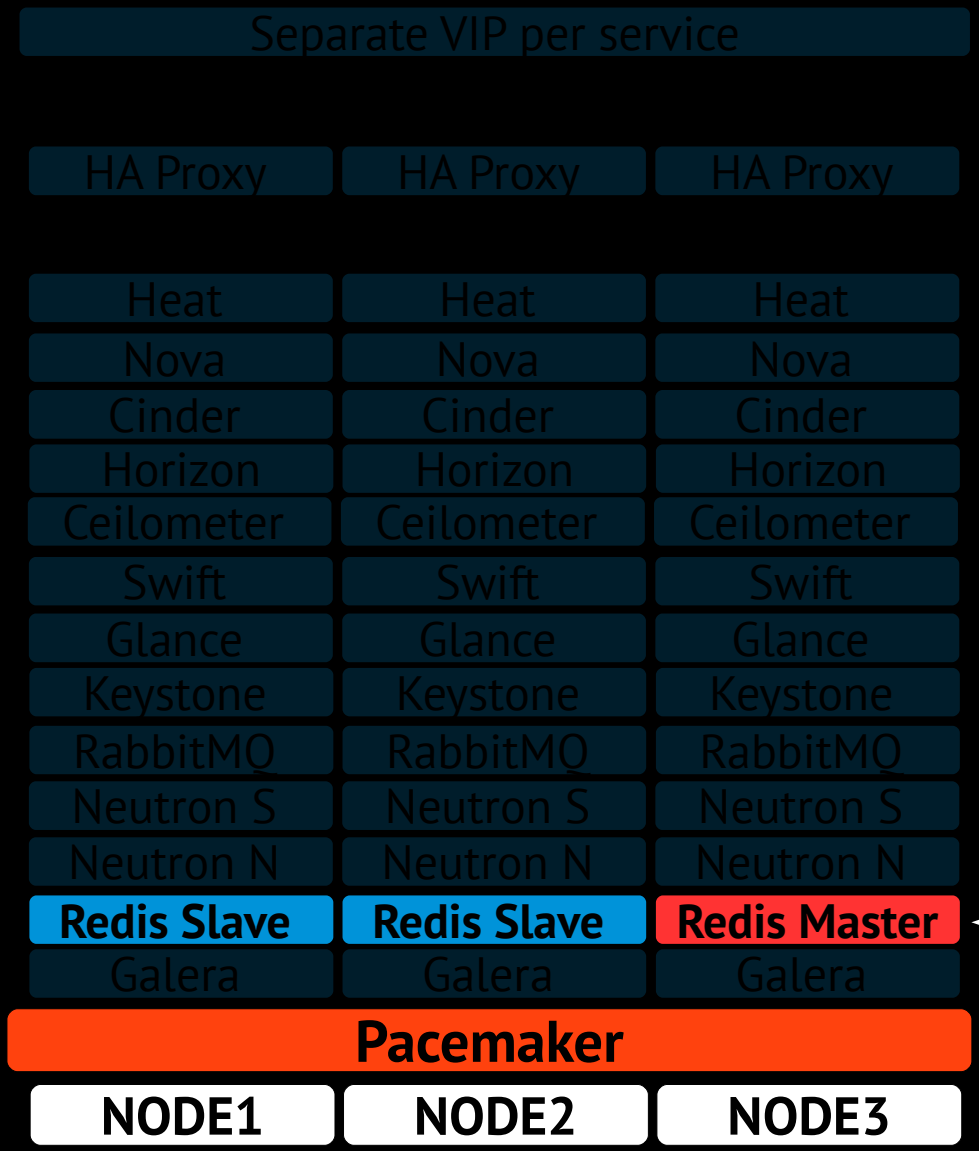


Then Shutdown Galera cluster.

Complex Startup Ordering

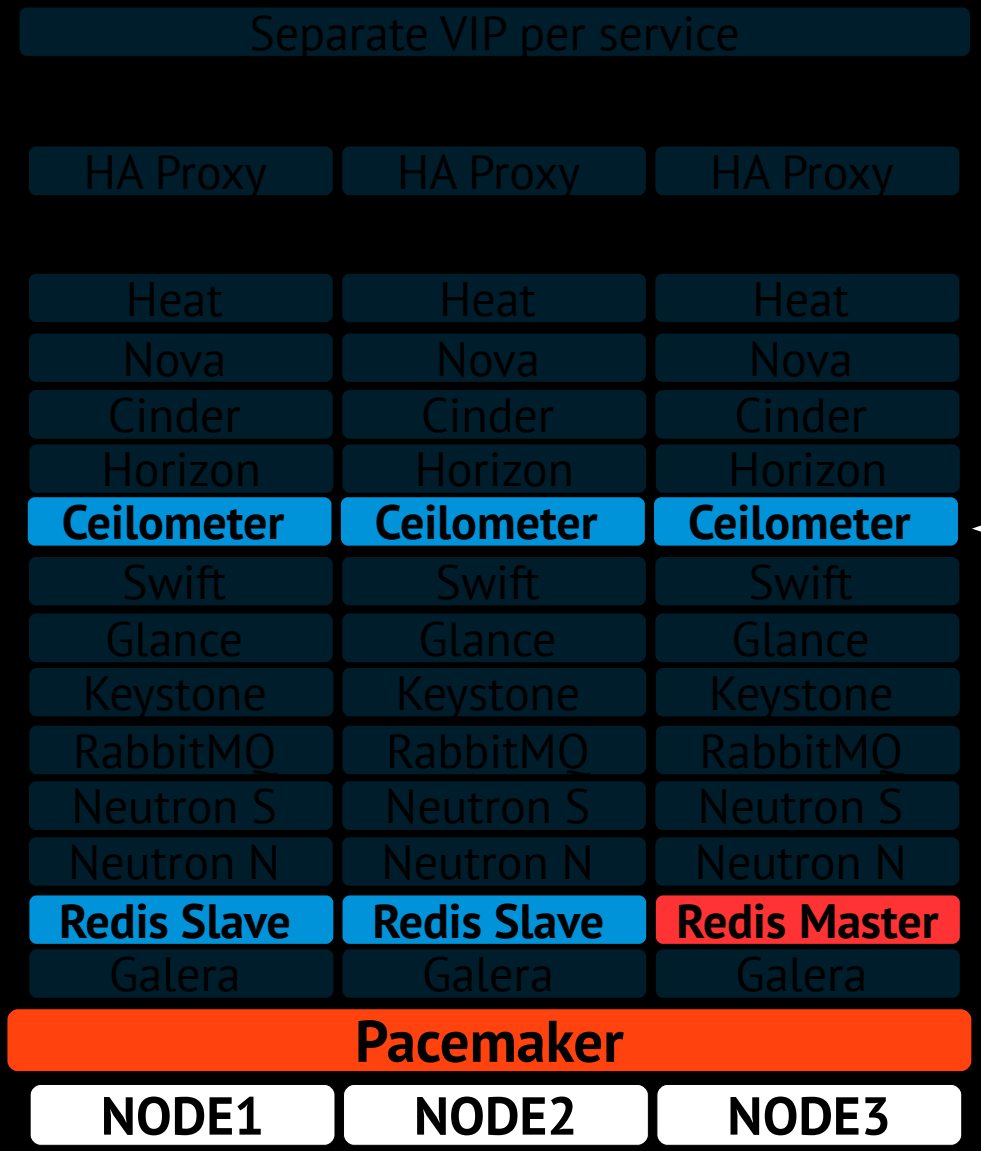


Complex Startup Ordering



Promote one instance of Redis clone to be **Master** Instance.

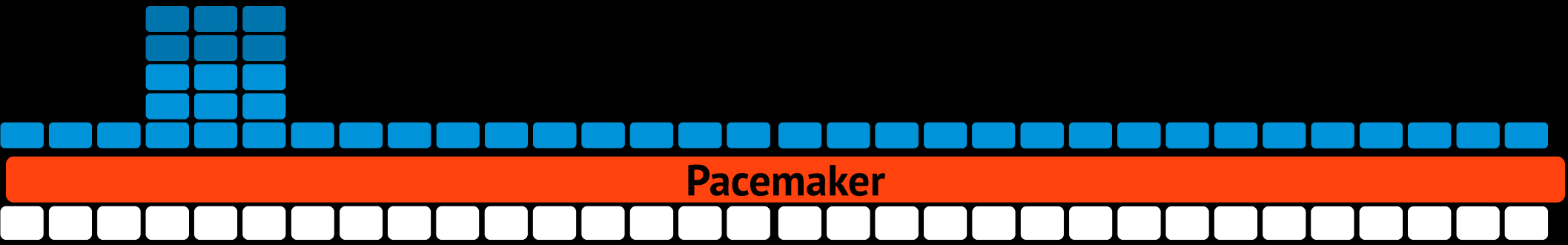
Complex Startup Ordering



Then start Ceilometer cluster which depends on Redis

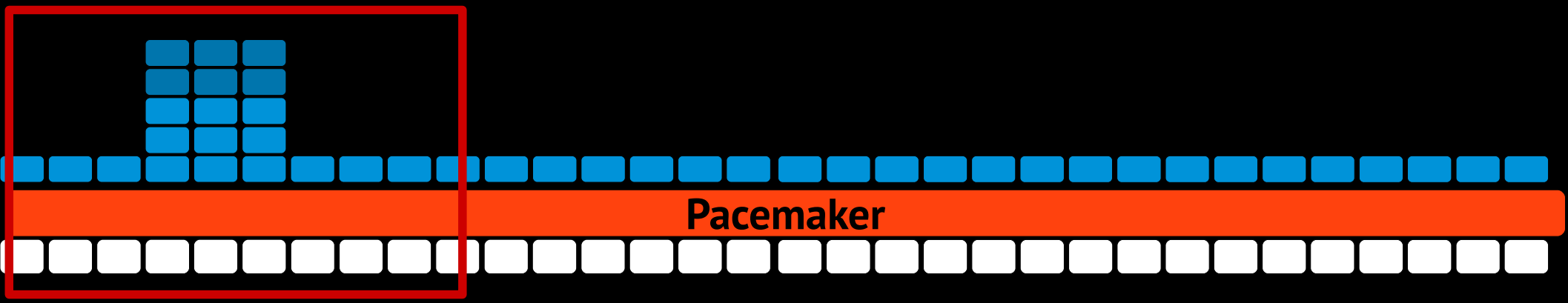
Segregated Architecture

- Each service runs on its own dedicated hardware.
- Scales much further
- Add capacity where capacity makes sense.
- Requires lots and lots of nodes.



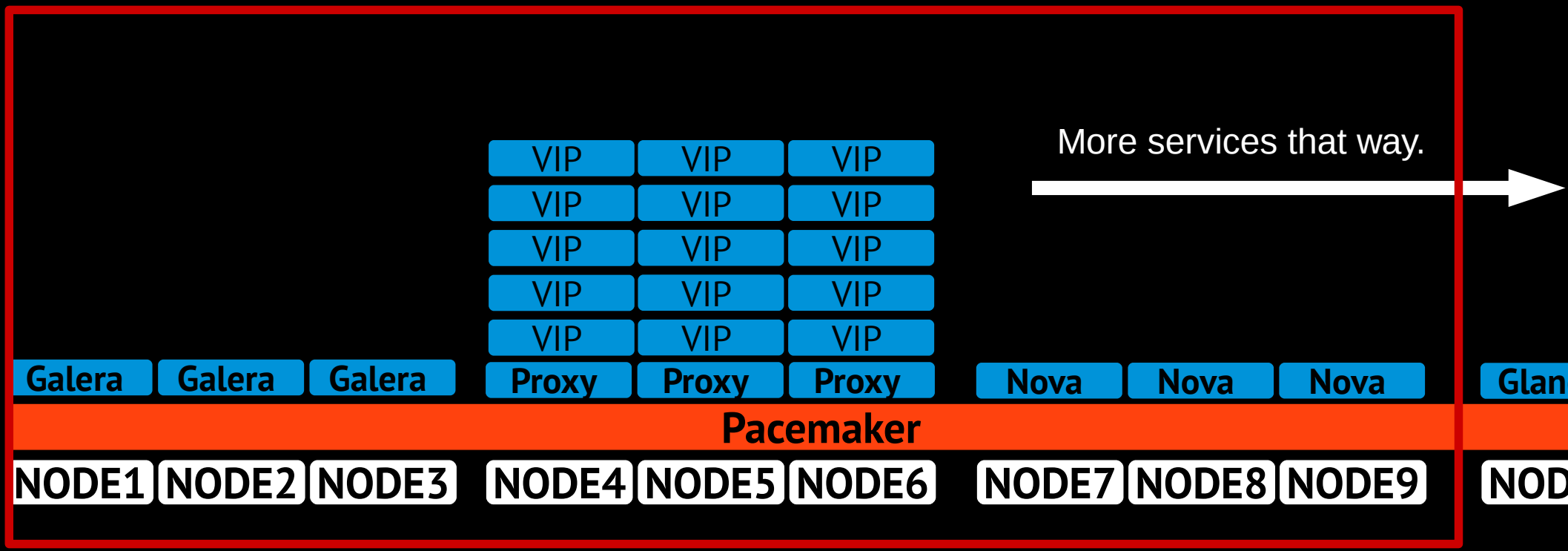
Segregated Architecture

- Take a closer look.



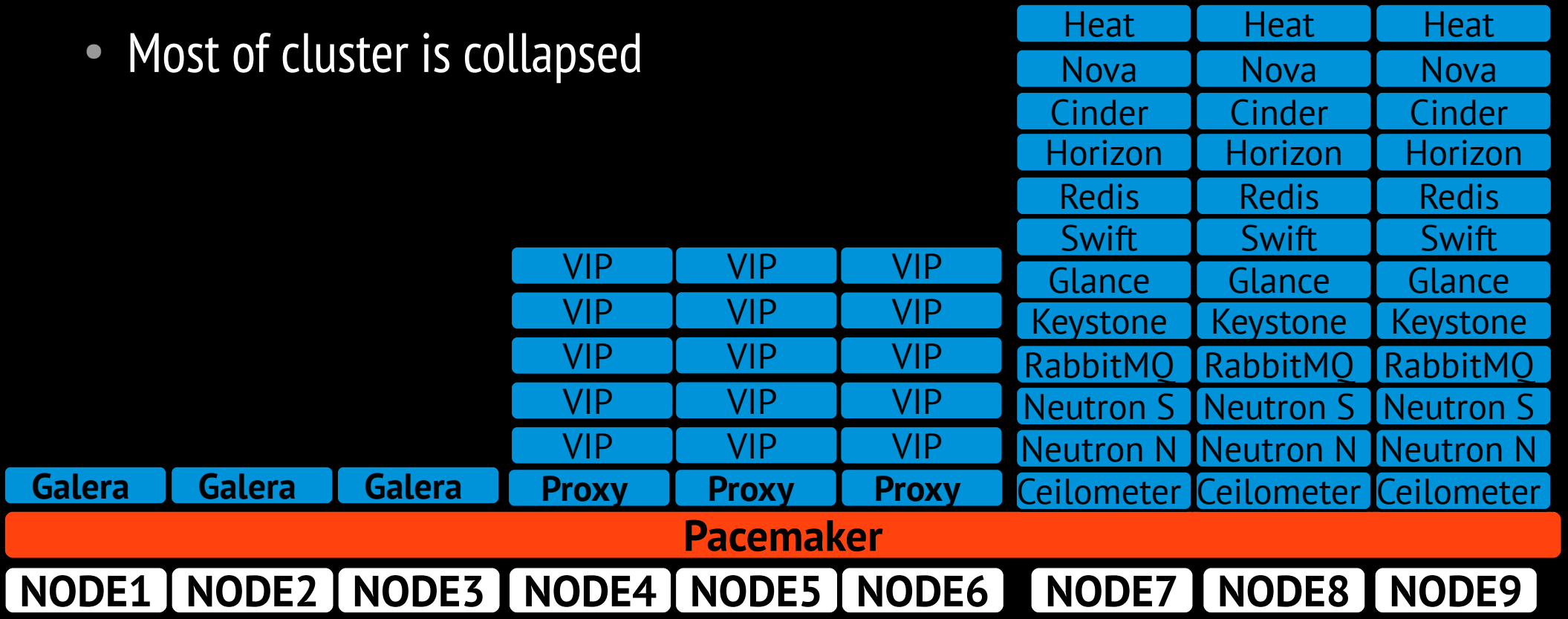
Segregated Architecture

- Possible to have have an entire set of nodes just for Load balancing
- Dedicated cluster for galera



Mixed Architecture

- Mixture of collapsed and segregated.
- Break some components into separate hardware
- Most of cluster is collapsed

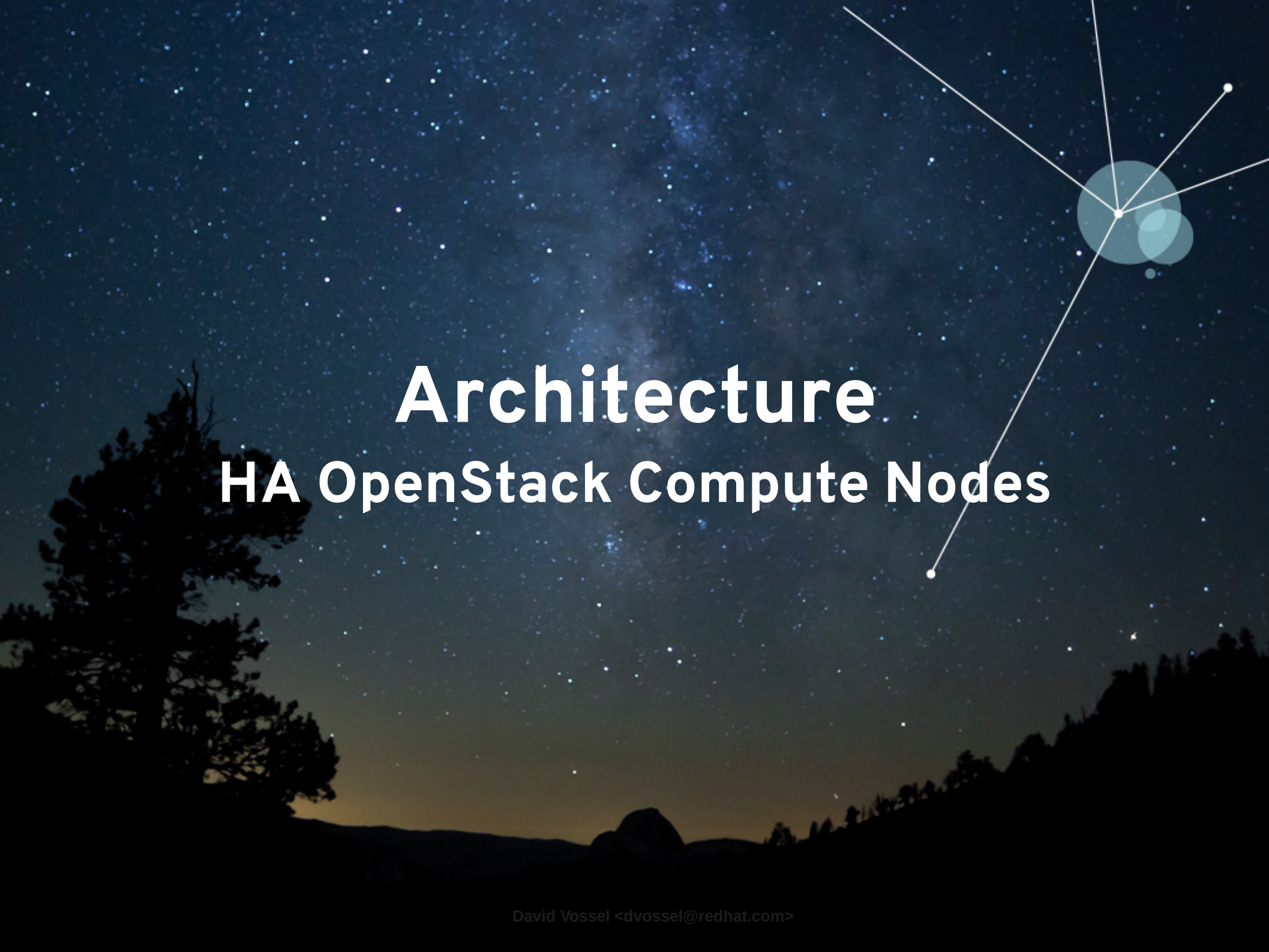


Pacemaker Advantages

- Automates bootstrap of services that previously required hand holding.
- Example: Automate Galera bootstrap
    1. Find out which galera instance is most up-to-date
    2. Bootstrap most current galera instance first.
    3. Then sync other galera instances

Pacemaker Advantages. Continued...

- start/stop distributed services in a graceful ordered manner.
- Gracefully and controller node into standby for maintenance.
- Dynamically grow capacity by adding more pacemaker nodes
- Centralized view of distributed service state.



Architecture

HA OpenStack Compute Nodes

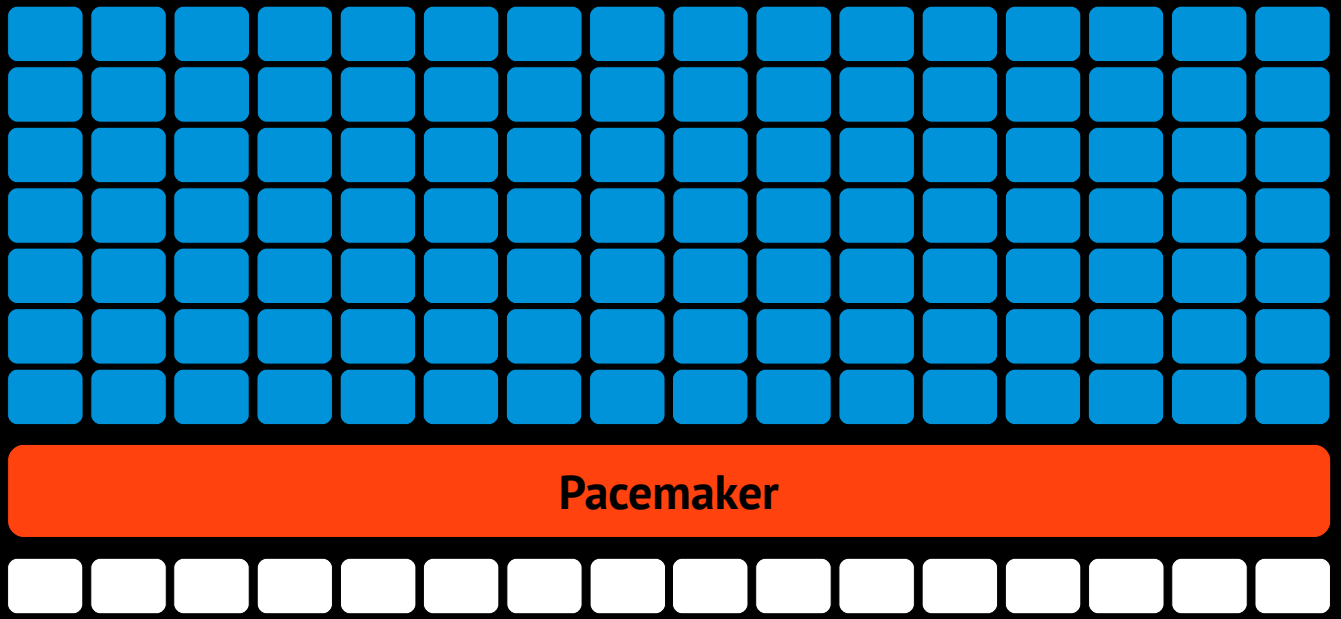
David Vossel <dvossel@redhat.com>

HA for Cattle

- Both Pets **and** Cattle **need High Availability.**
- Recognize the techniques used for each are different.

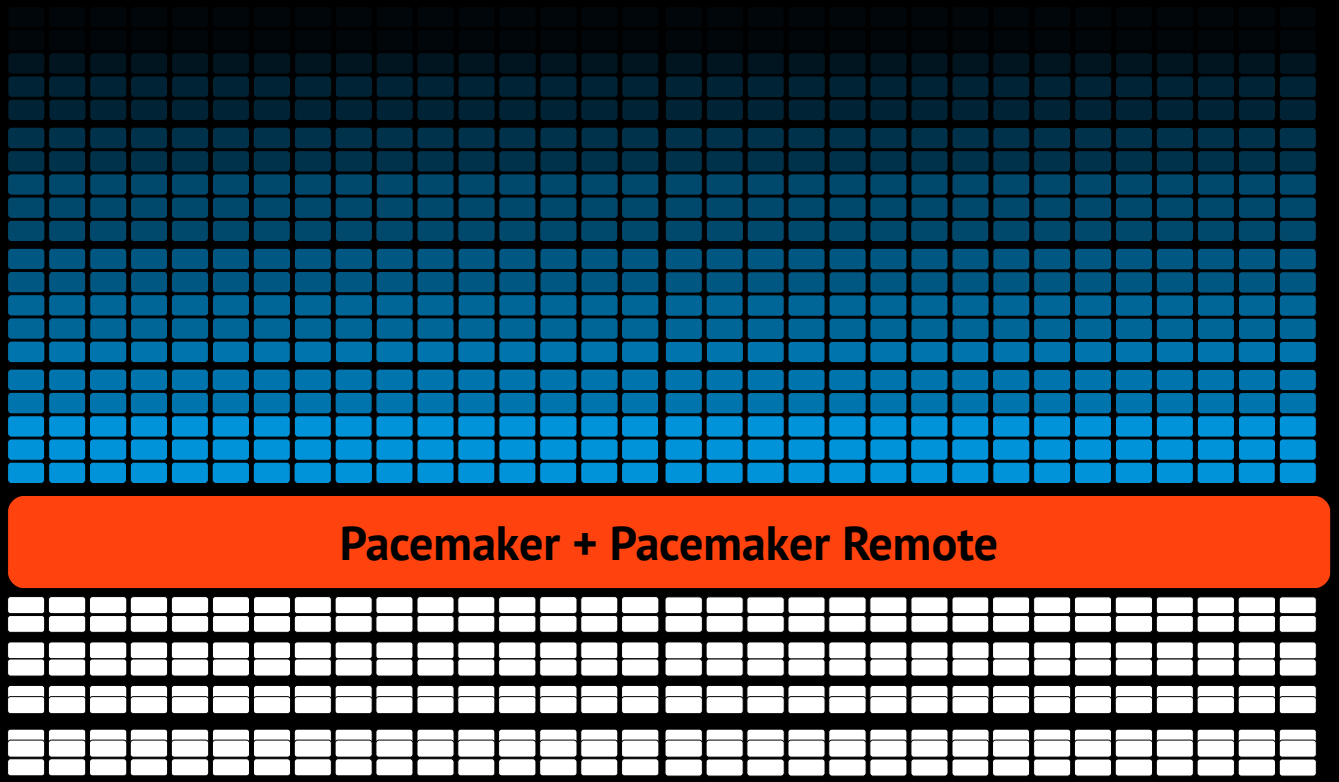
Pacemaker for Pets and small Herds.

- No limits in the number of resources.
- Pacemaker supports “n-node” clusters.
- Cluster are limited by the Corosync messaging layer to 16 nodes.



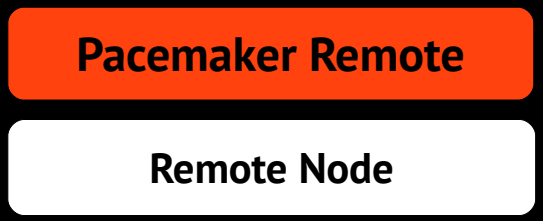
Pacemaker Remote for the Cattle.

- Pacemaker Remote allows clusters to scale beyond corosync membership layer limitations.
- Pacemaker Remote can scale clusters to 100s possibly 1000s of nodes.



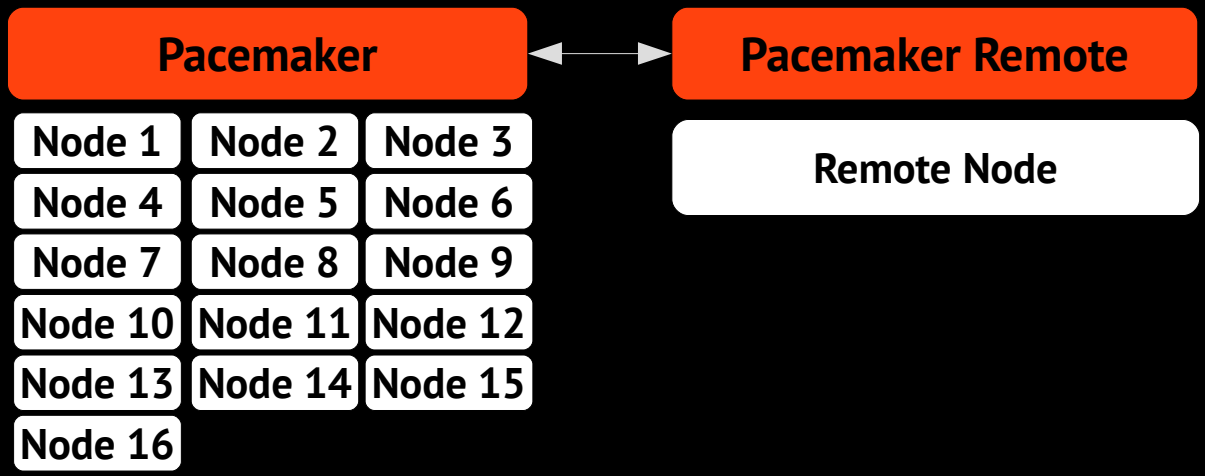
The Solution: Pacemaker Remote

- Pacemaker Remote is a single daemon, **pacemaker_remoted**



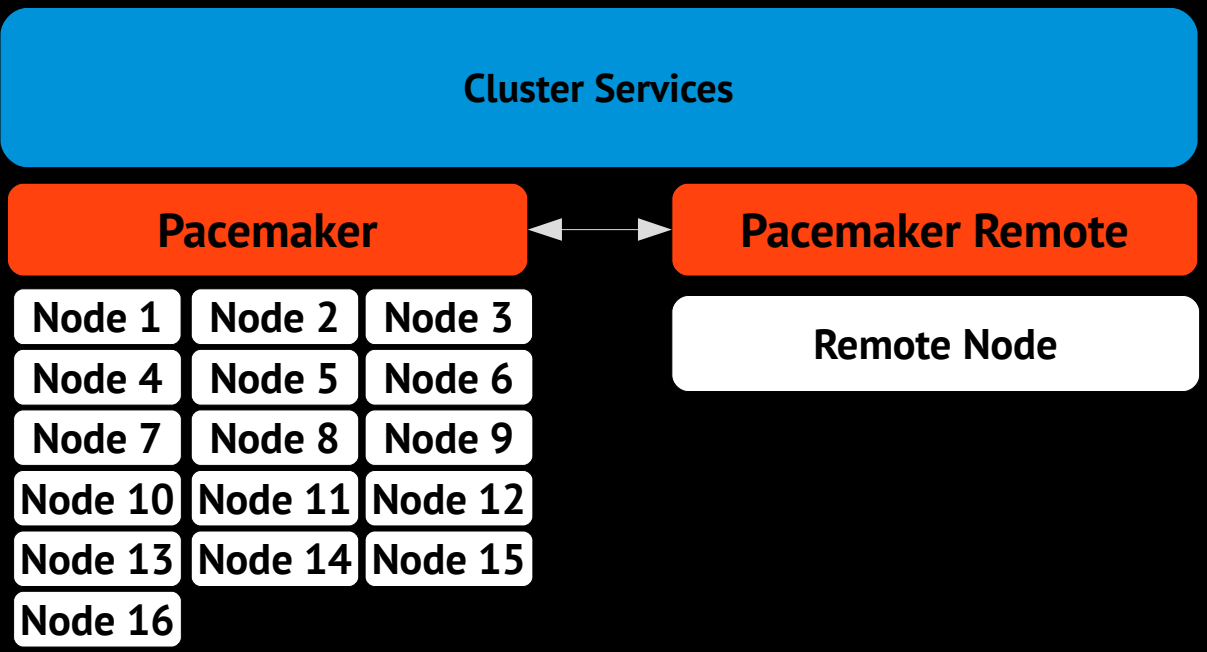
The Solution: Pacemaker Remote

- Pacemaker Remote is a single daemon, **pacemaker_remoted**
- This daemon is a lightweight way of integrating nodes into the cluster.



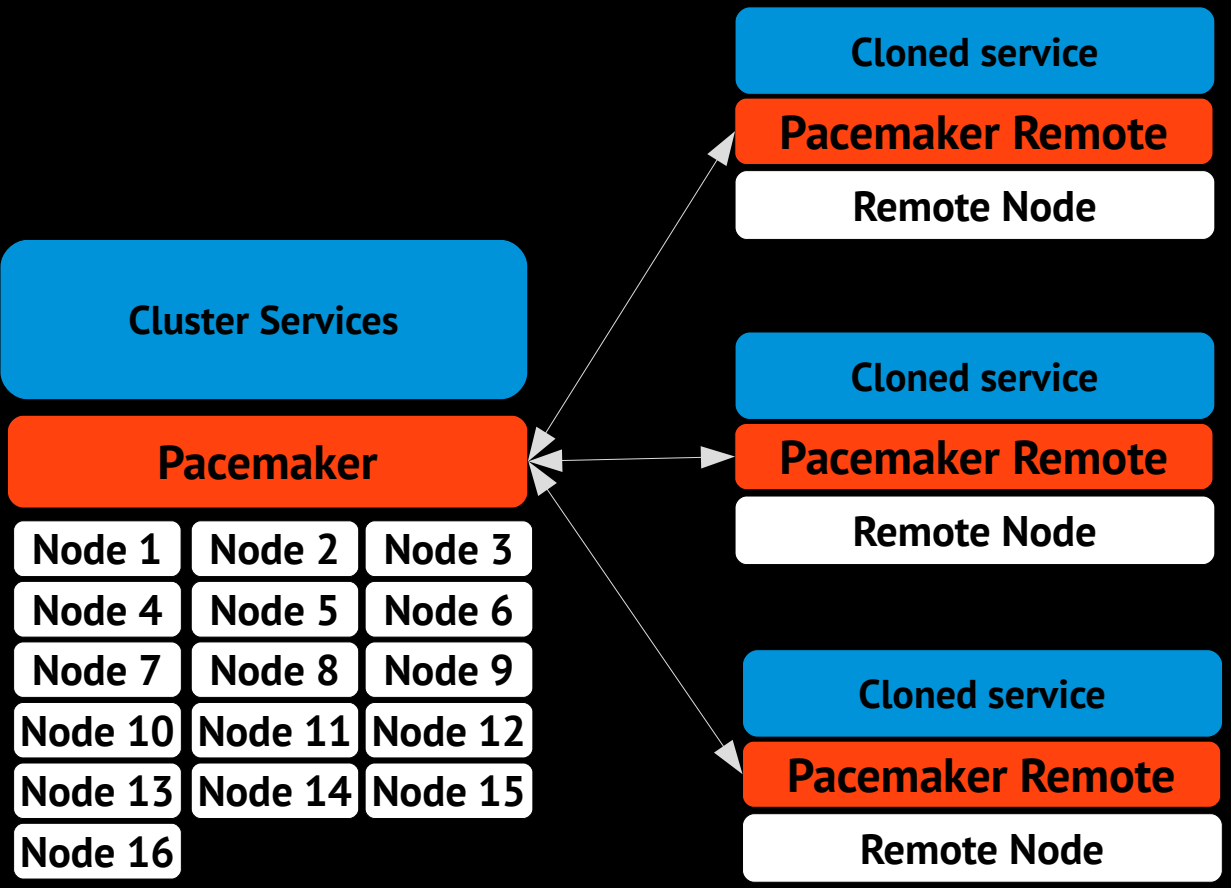
The Solution: Pacemaker Remote

- Pacemaker Remote is a single daemon, **pacemaker_remoted**
- This daemon is a lightweight way of integrating nodes into the cluster.
- Cluster services spread out across pacemaker and pacemaker_remote nodes as a single cluster partition.



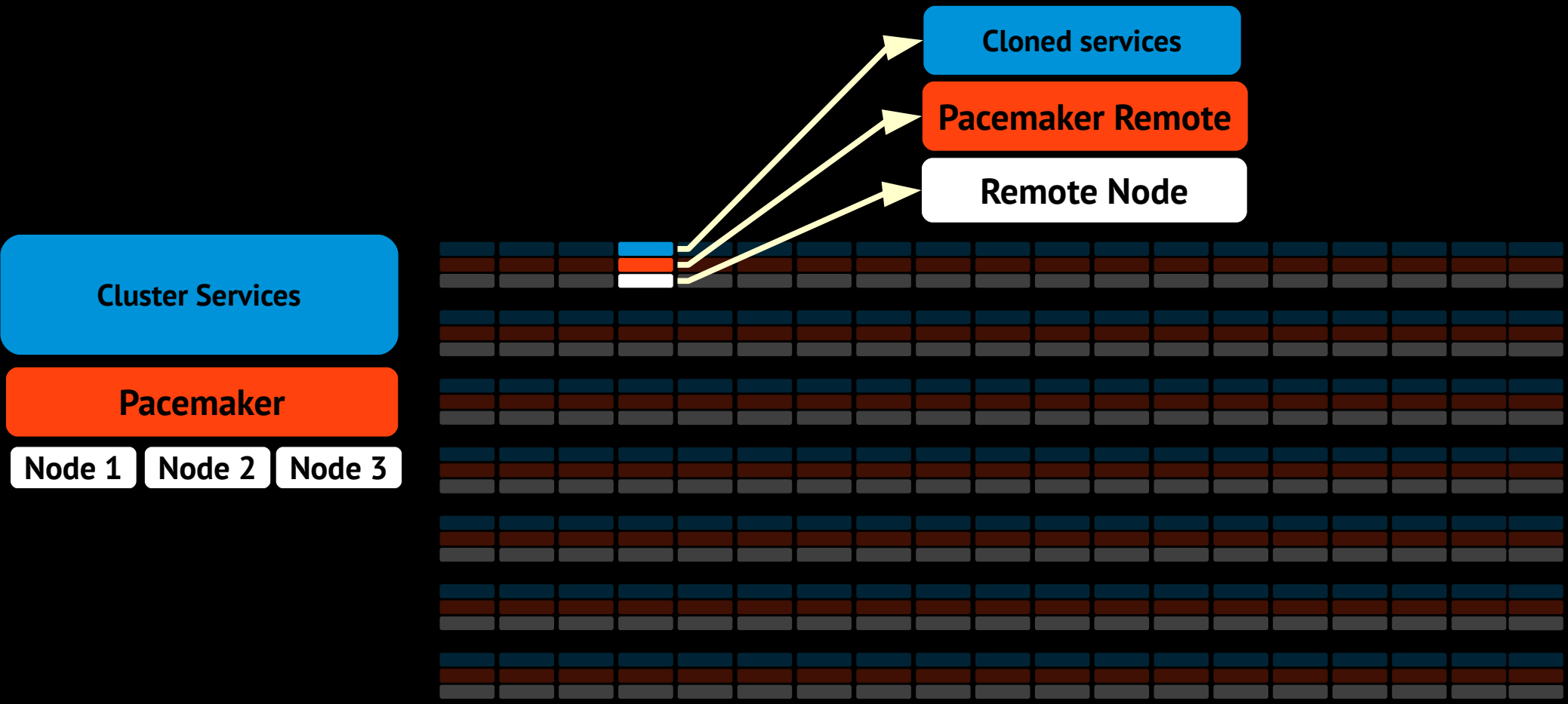
Pacemaker Remote use-case

- Management of services on Pacemaker Remote can work just like Pacemaker
- But thrives in the cattle **use case** where **every remote instance is identical.**

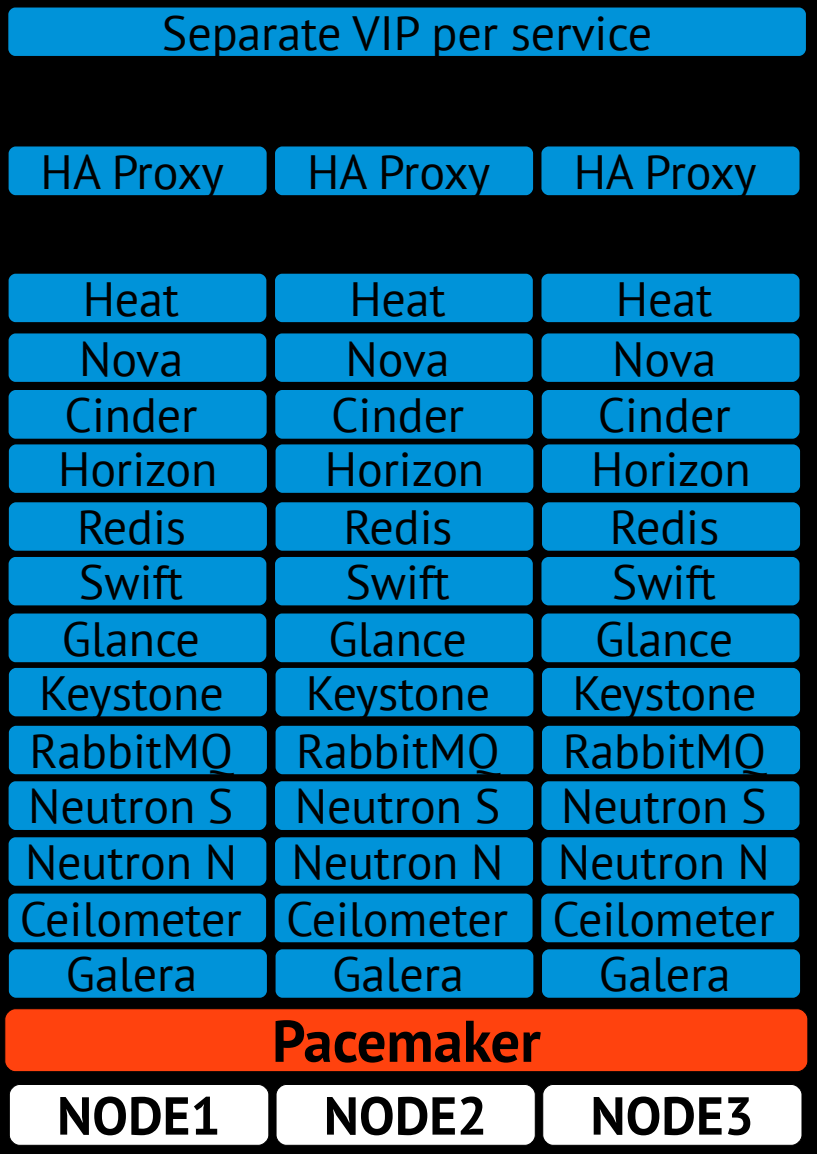


Pacemaker Remote use case

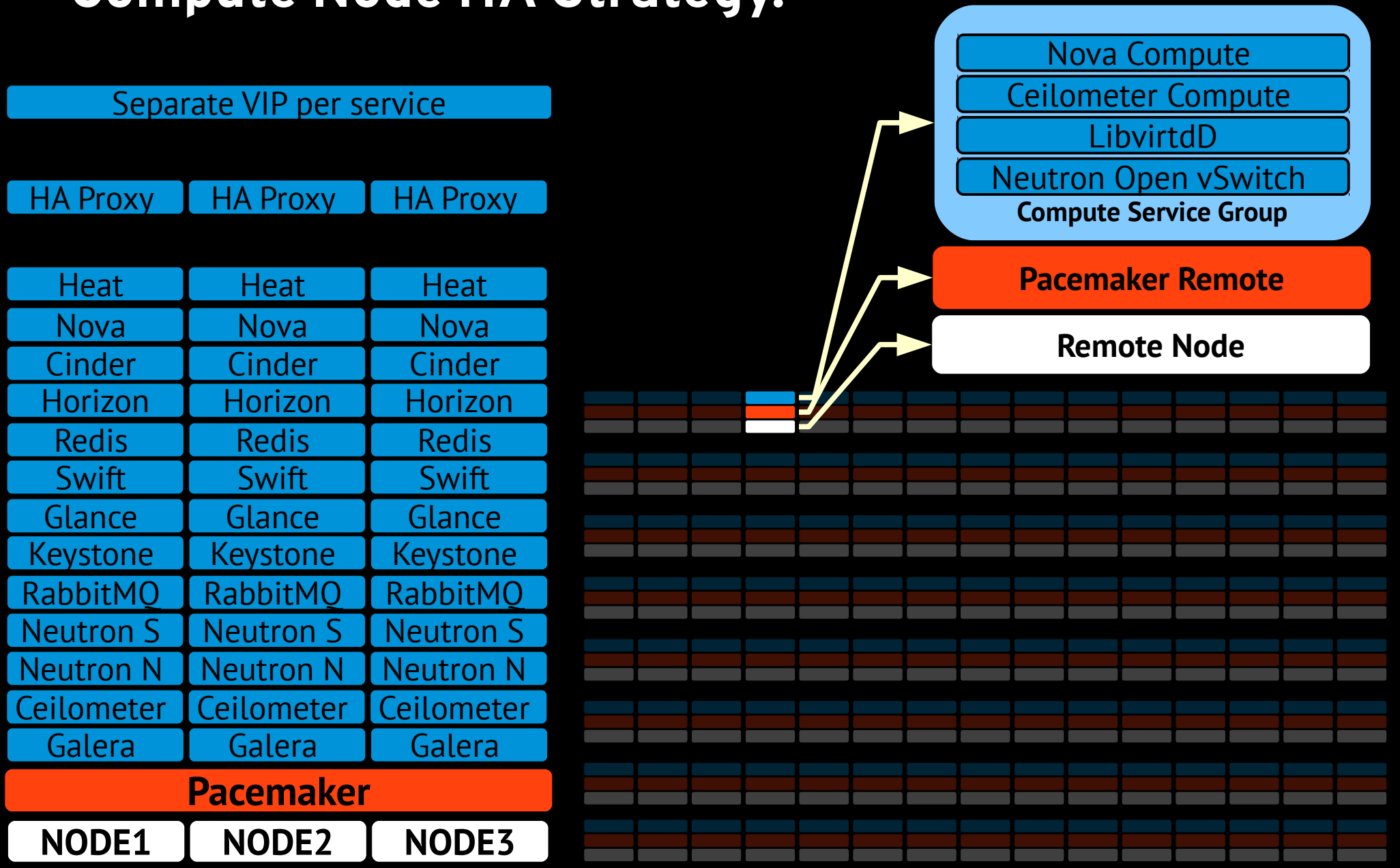
- Cloned services scale quite well on pacemaker remote



Compute Node HA Strategy.



Compute Node HA Strategy.



David Vossel <dvossel@redhat.com>

Why HA Compute Nodes?

- Maximize the Availability of Compute Instances.
  - detection of dead Cattle instances
  - Automate recovery of Cattle instances

Why HA Compute Nodes?

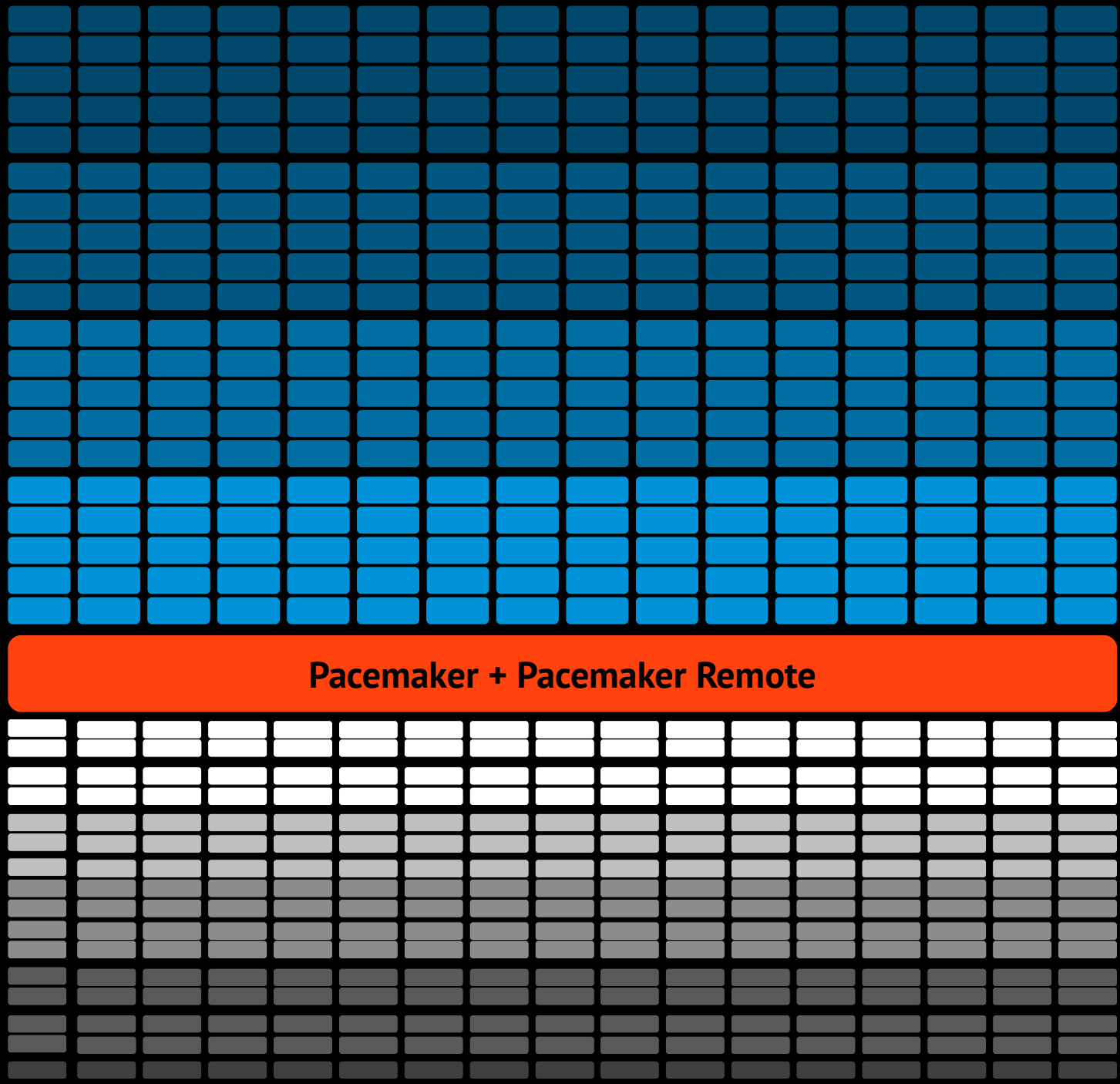
- Maximize the Availability of Compute Instances.
  - detection of dead Cattle instances
  - Automate recovery of Cattle instances
- Pacemaker Remote also has a secret weapon.

STONITH + Pacemaker Remote.

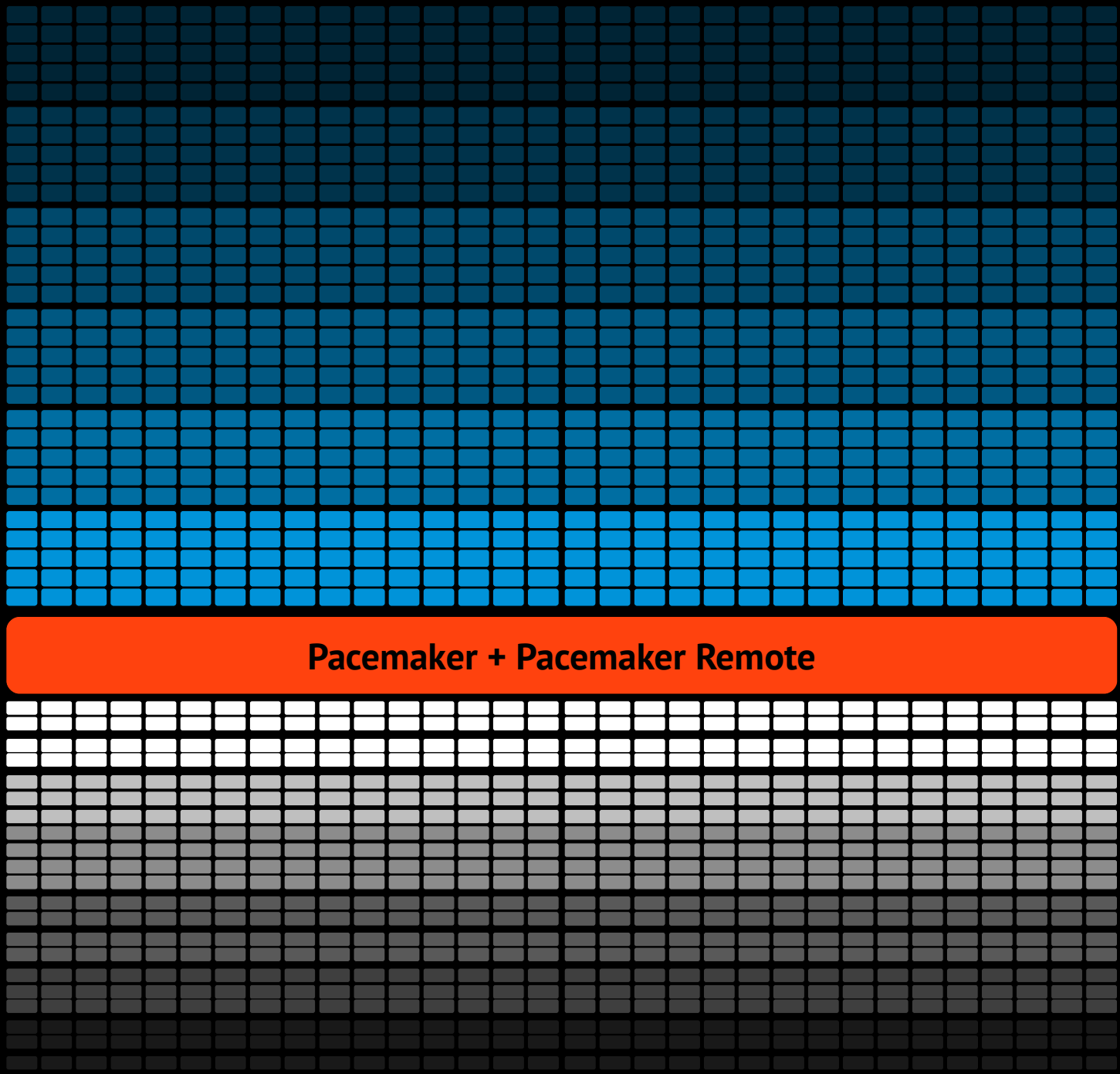
David Vossel <dvossel@redhat.com>

David Vossel <dvossel@redhat.com>

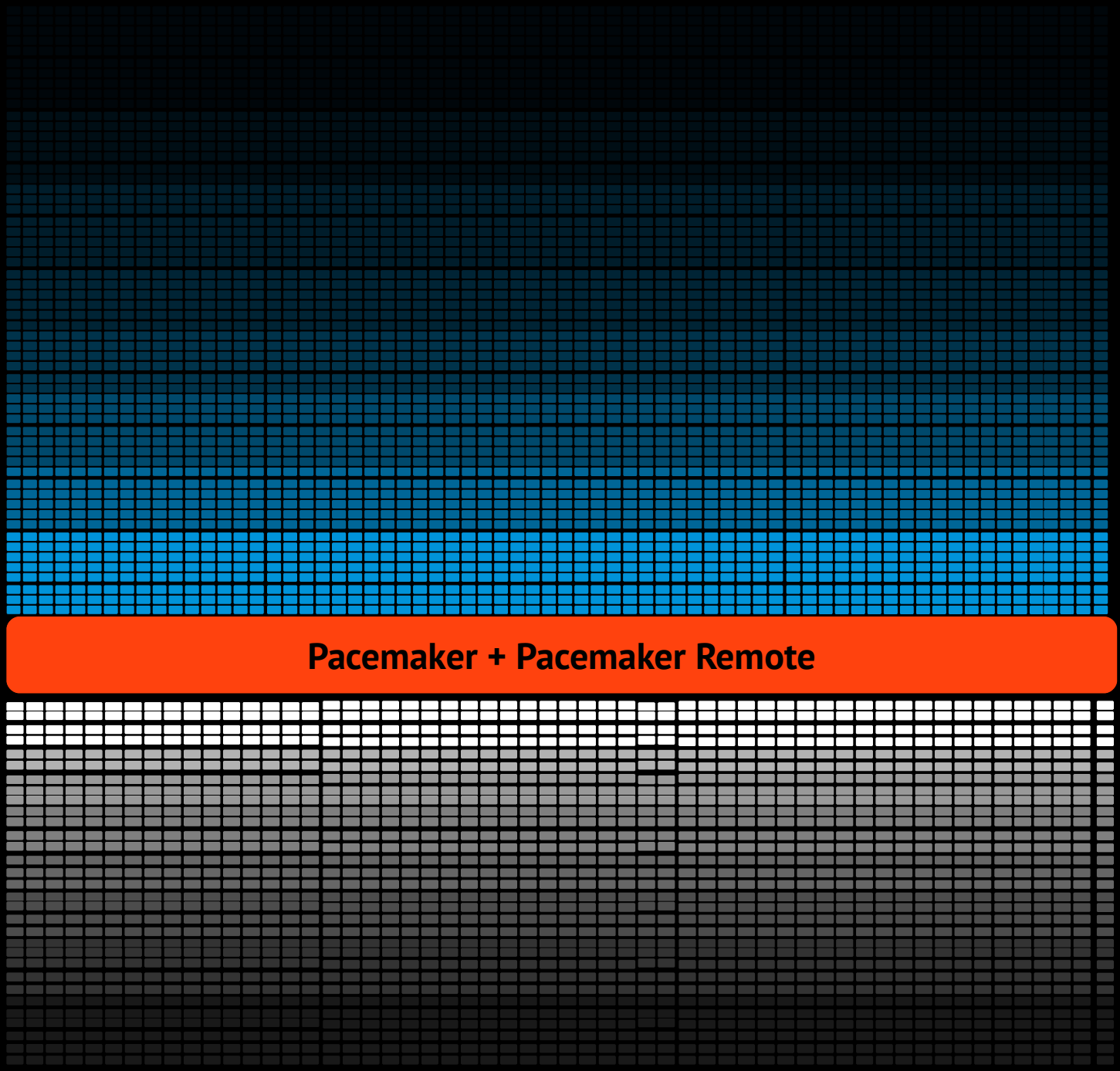
Limits?



How many services?



How many services?



How many nodes?

David Vossel <dvossel@redhat.com>

- MY NAME
- Work at Red Hat
- and today I want to talk about. Pacemaker
 pacemaker to power Openstack High Availability.

First off. Title. May be vague. Hope it piqued interest

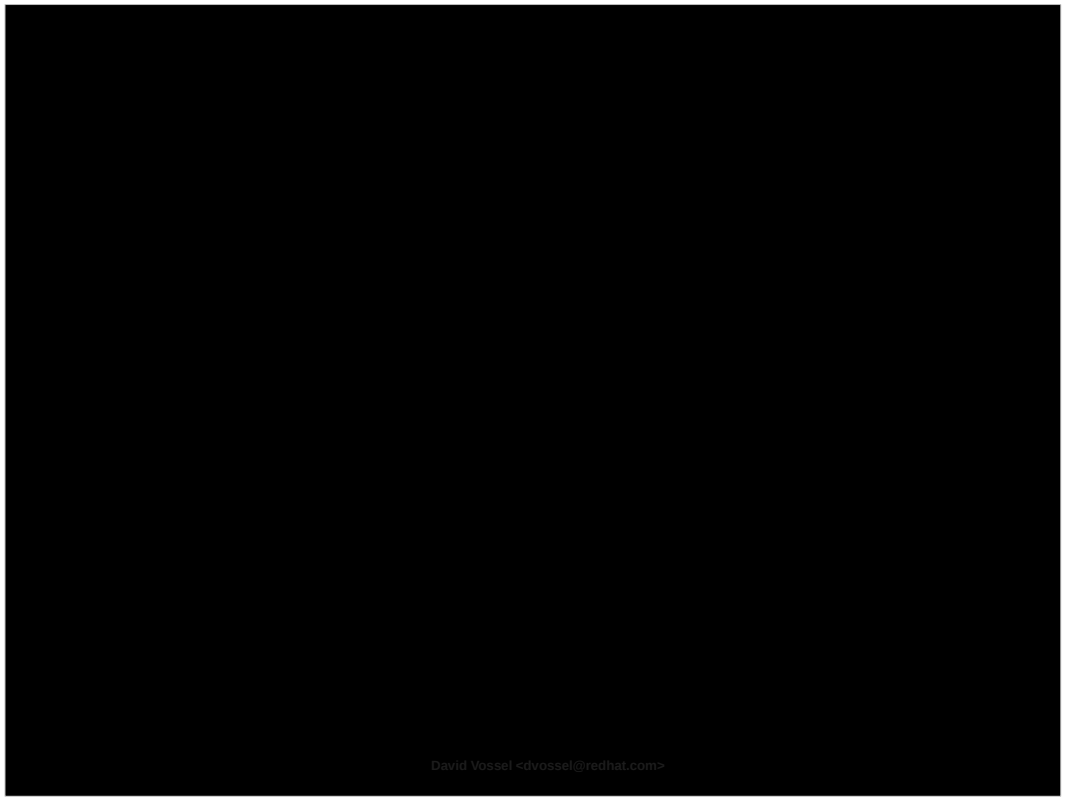
But clear on purpose.

1. openstack high availabilty
2. how pacemaker is the cornerstone that makes all of this possible.

- before discussing OpenStack HA
- want to do something different.

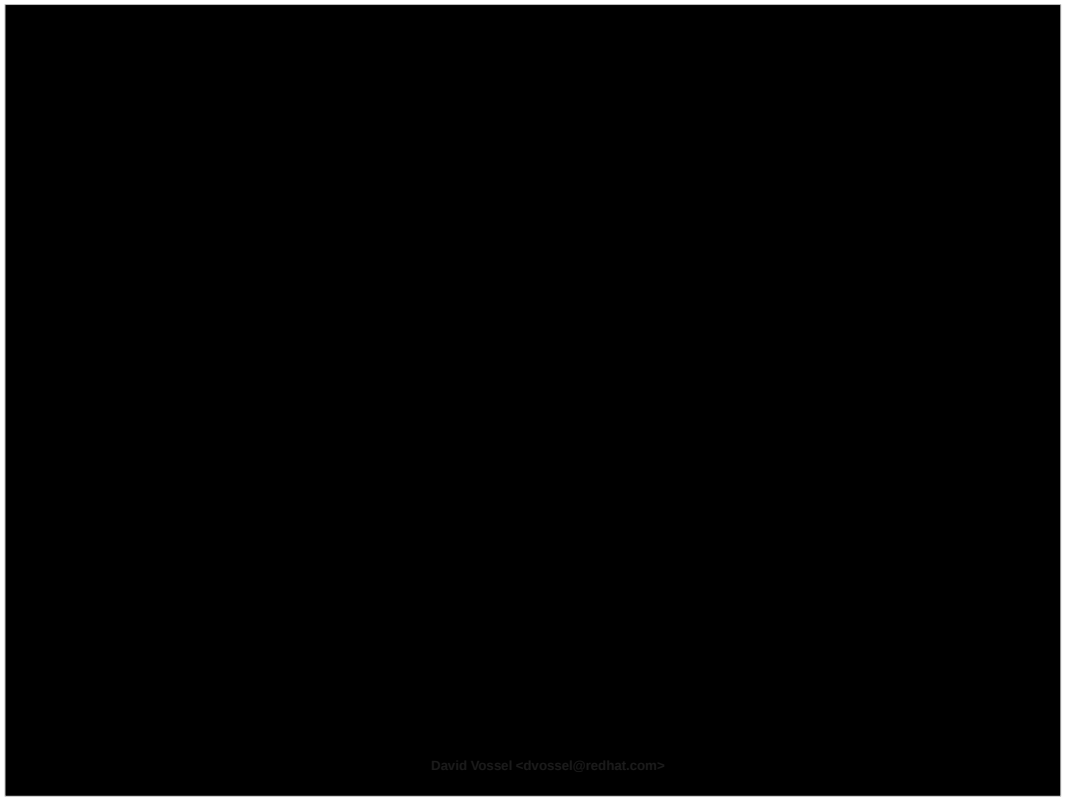
- Easy to discuss cool HA Ach
- but without background, makes little sense.
- Questions more important that solution.

- illustrate where we've come from and where we are
- I want to tell you a story.



So the title of our story here is.

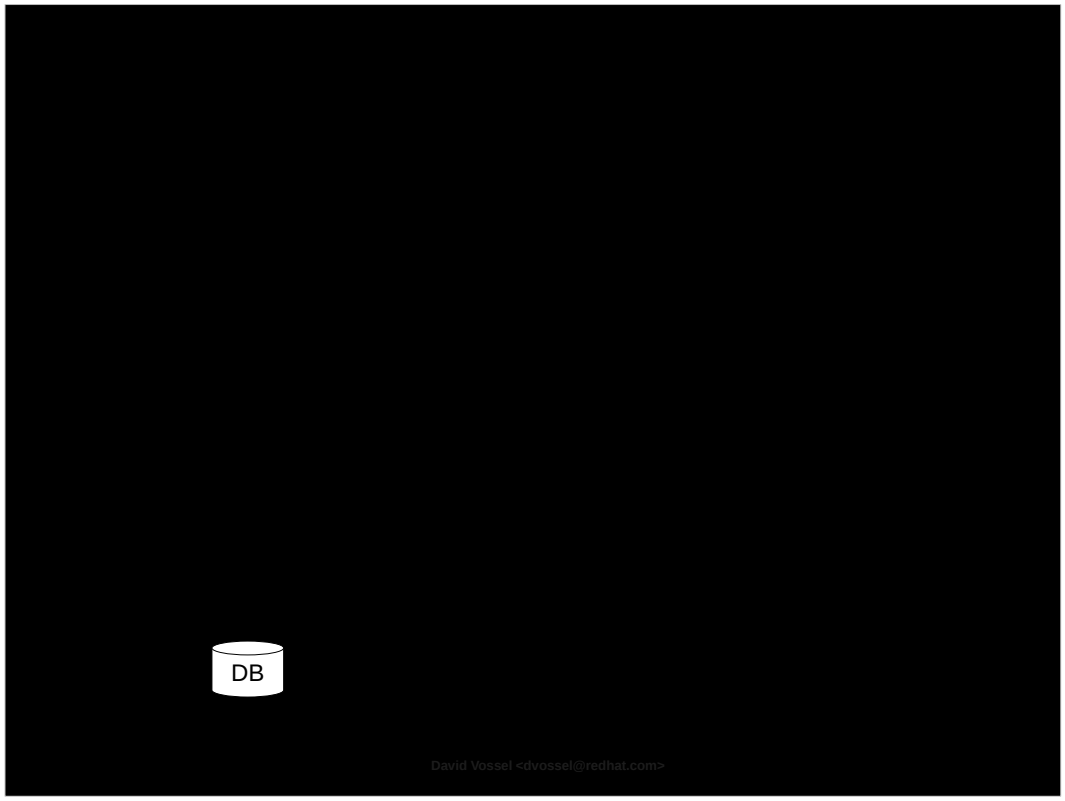
Our journey into our discovery that pacemaker makes a good fit for openstack



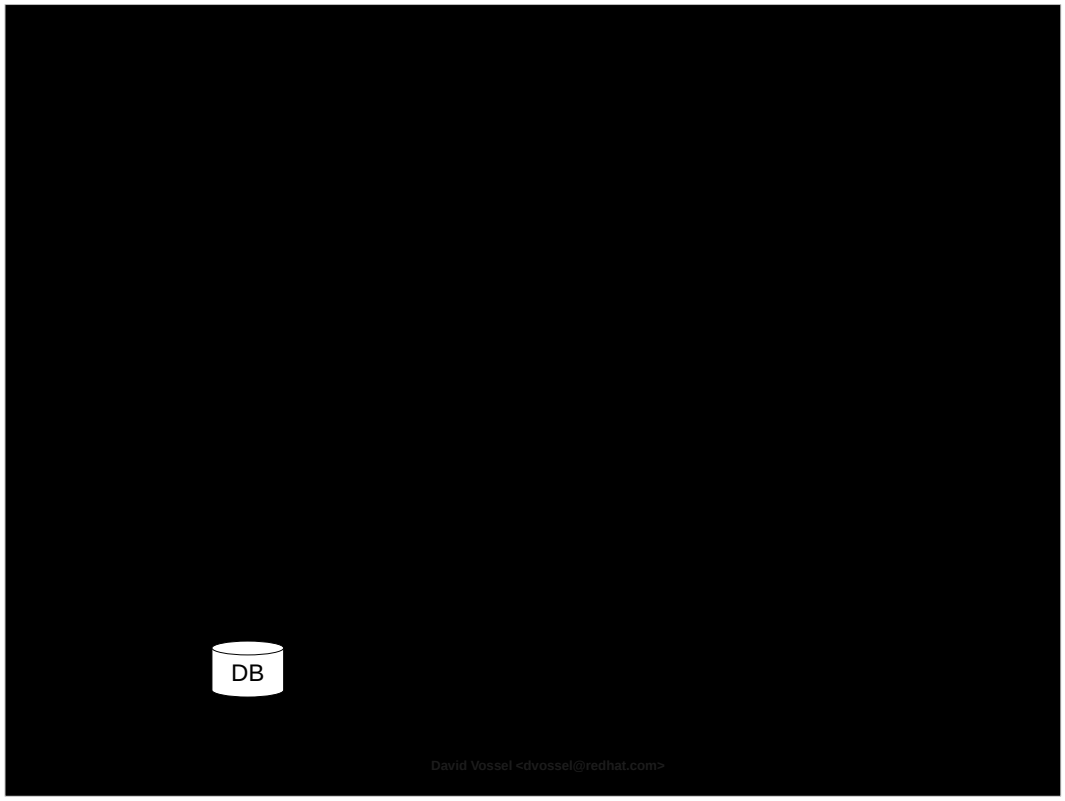
so. Story goes like this....

Once there was a database.

That looked like this.

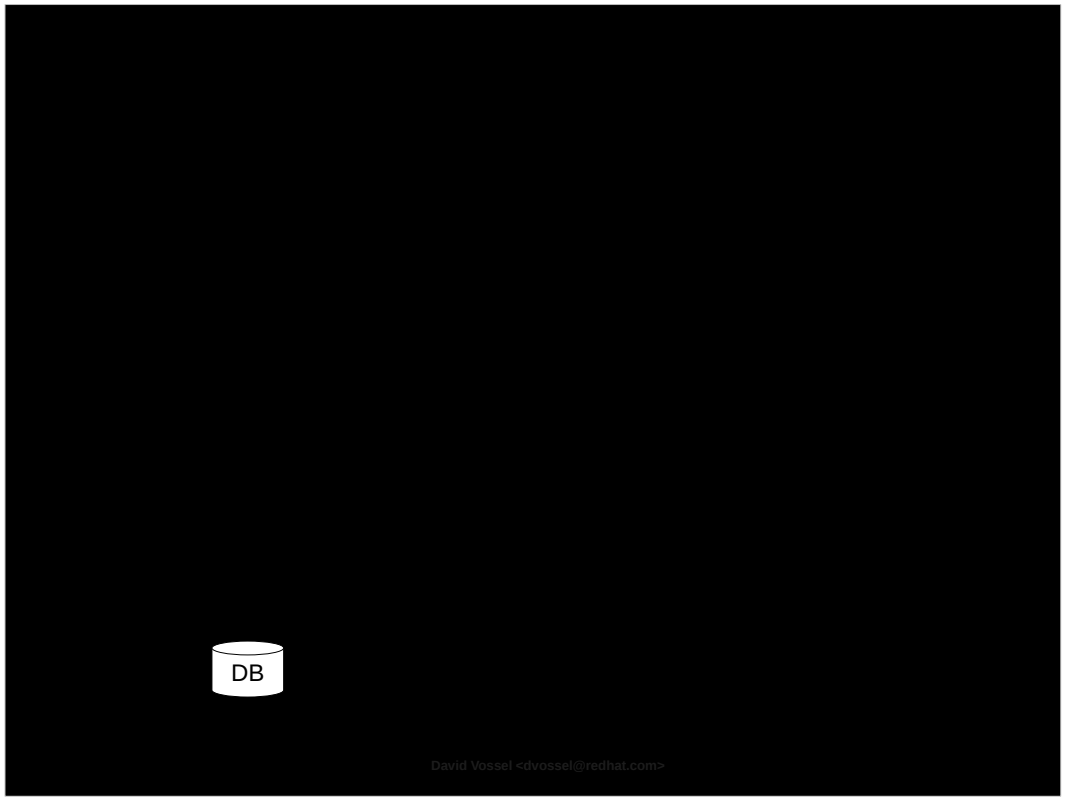


Skip next.

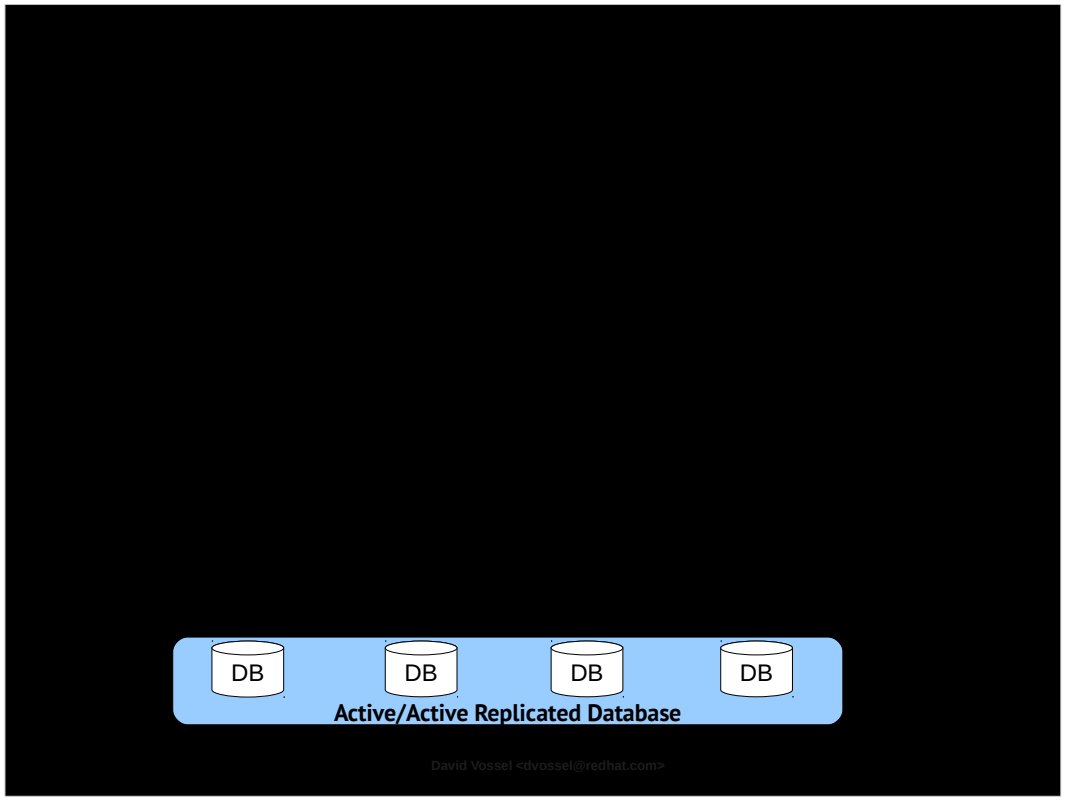


But This database was special.

It wasn't like all the other databases.



This was a distributed self replicating database



Everyone saw this and said

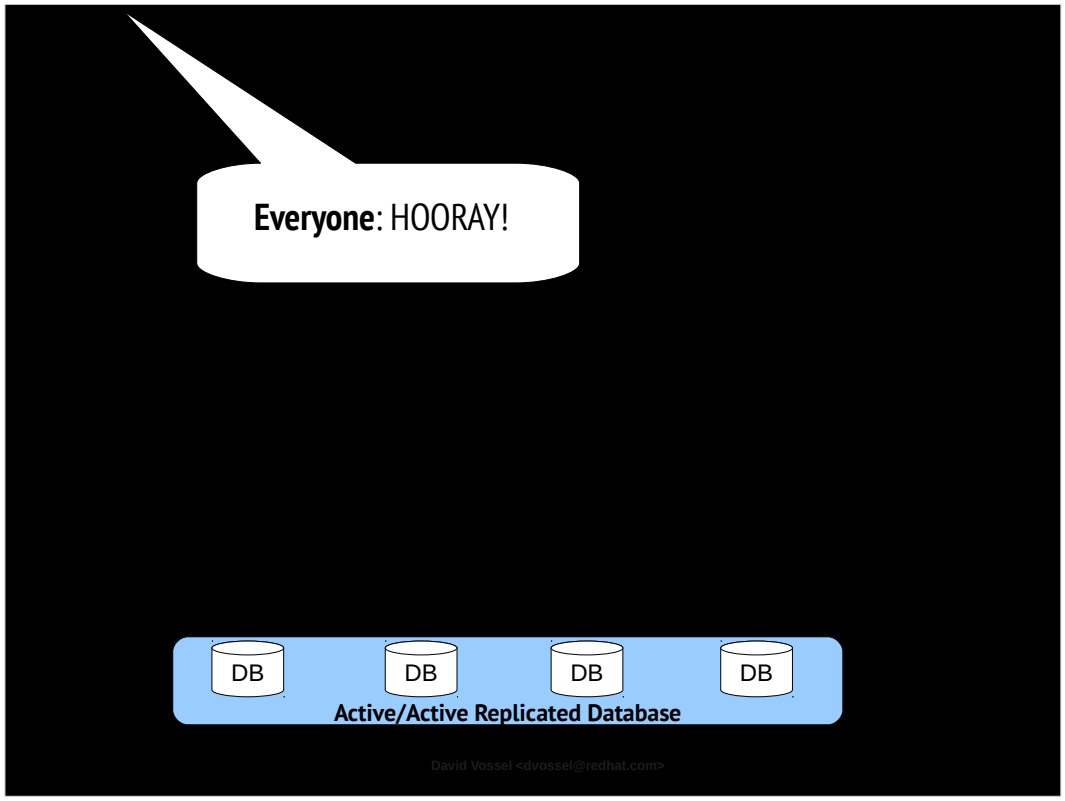
“a distributed self replicating database!!!”
“This is just want we need”

Fault tolerance is build right **into the application** itself.

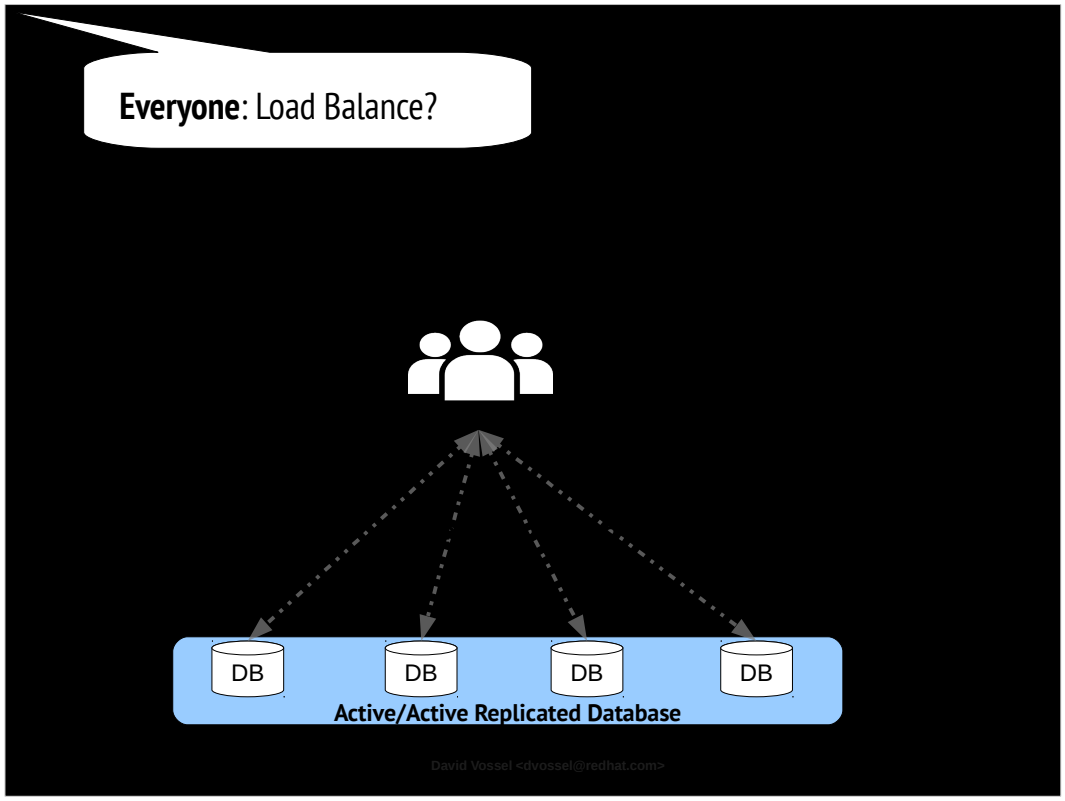
node goes down, cluster **picks up the slack.**

no expensive shared storage

It's just magic!



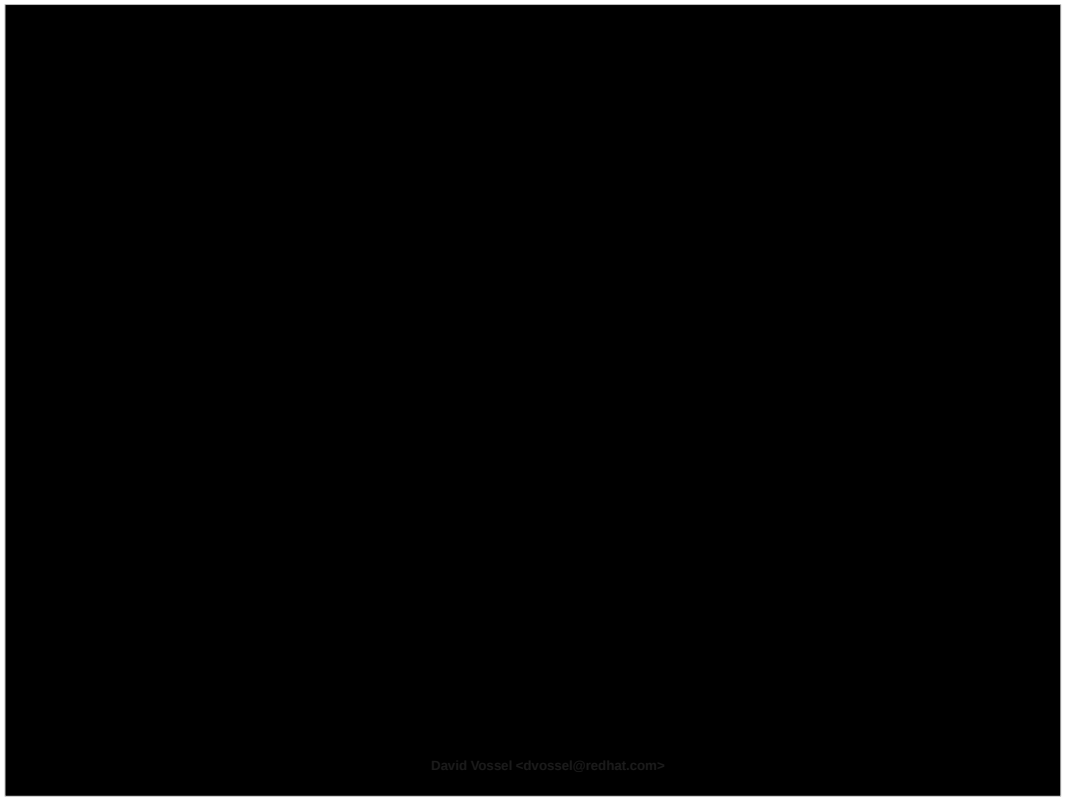
And everyone was happy!



But then they began to think.

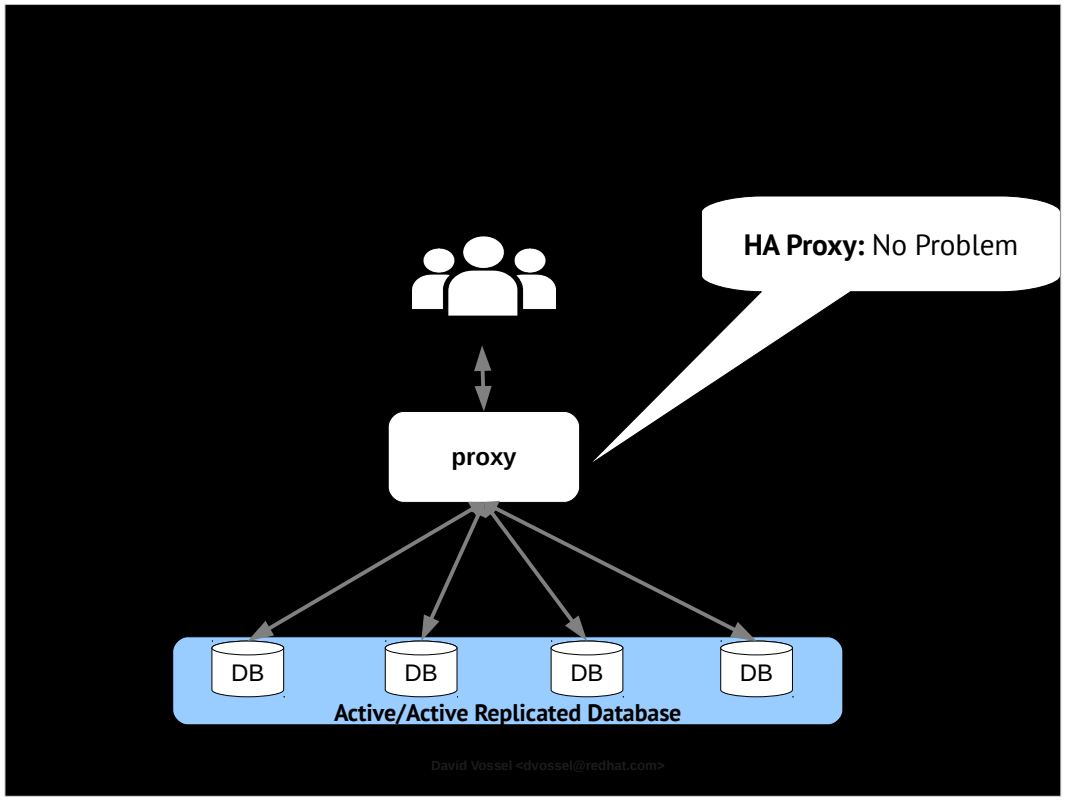
This is really neat

But how do we load balance this thing?



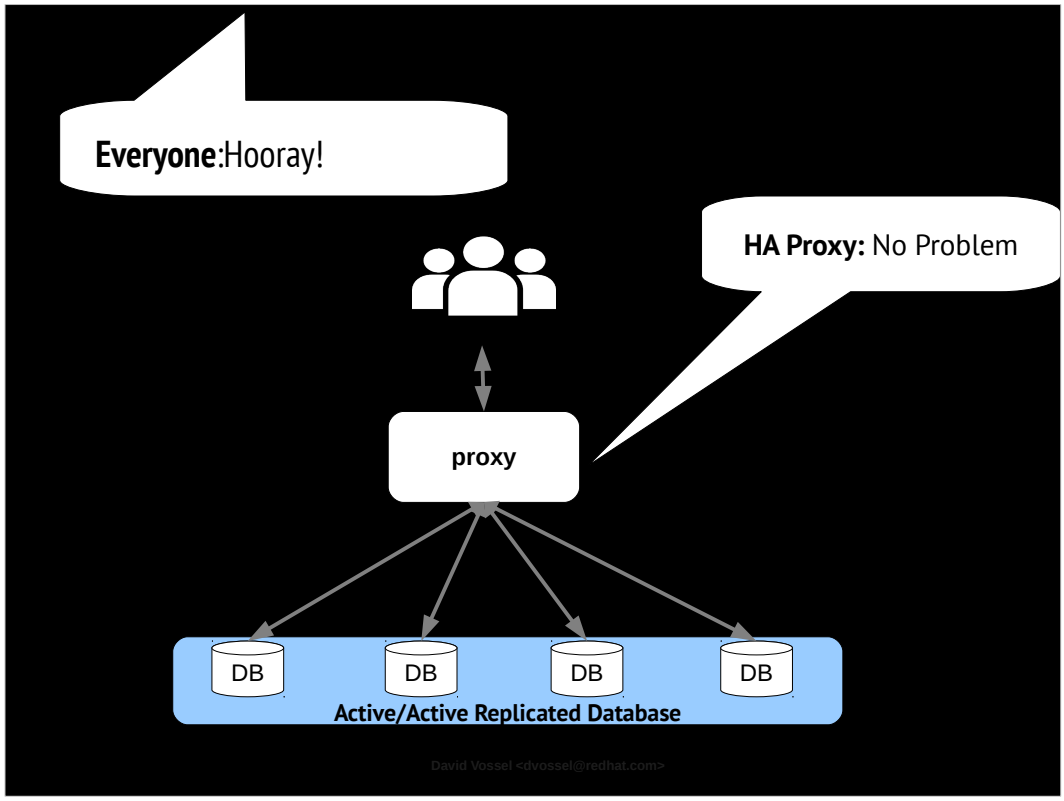
I wonder if HAProxy could help?

I mean, HAProxy is a load balancer. That's kind of what it does right?

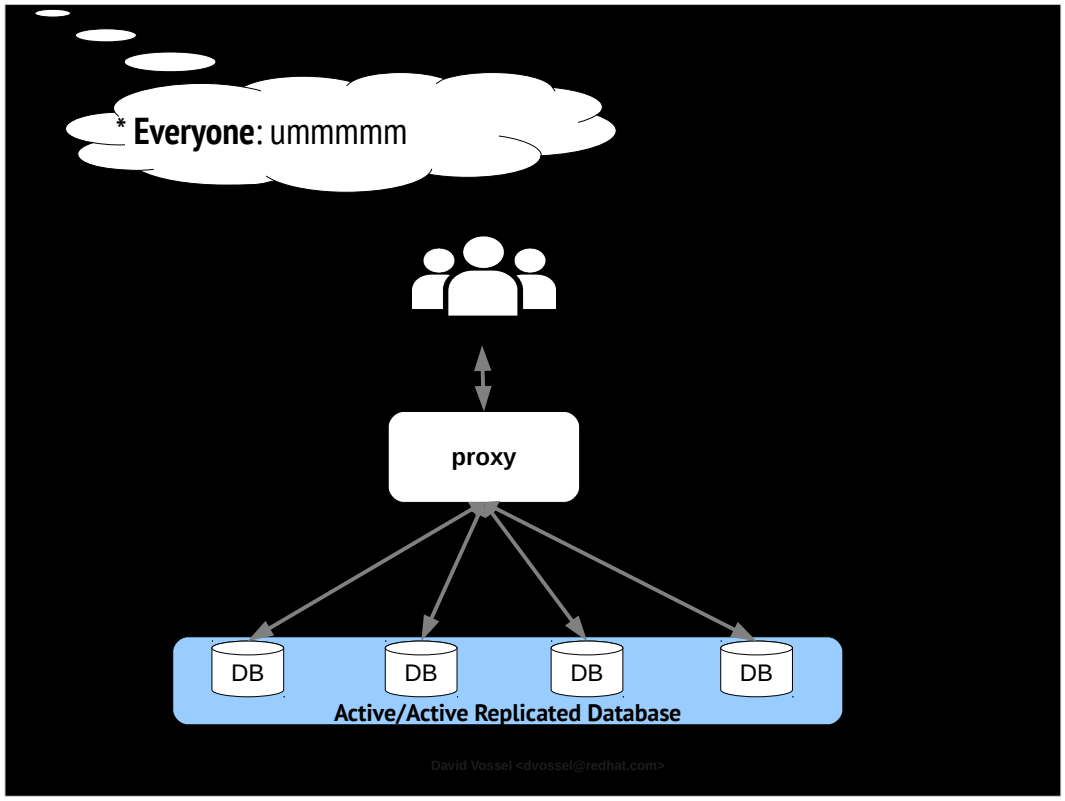


And sure enough. HAPROXY said

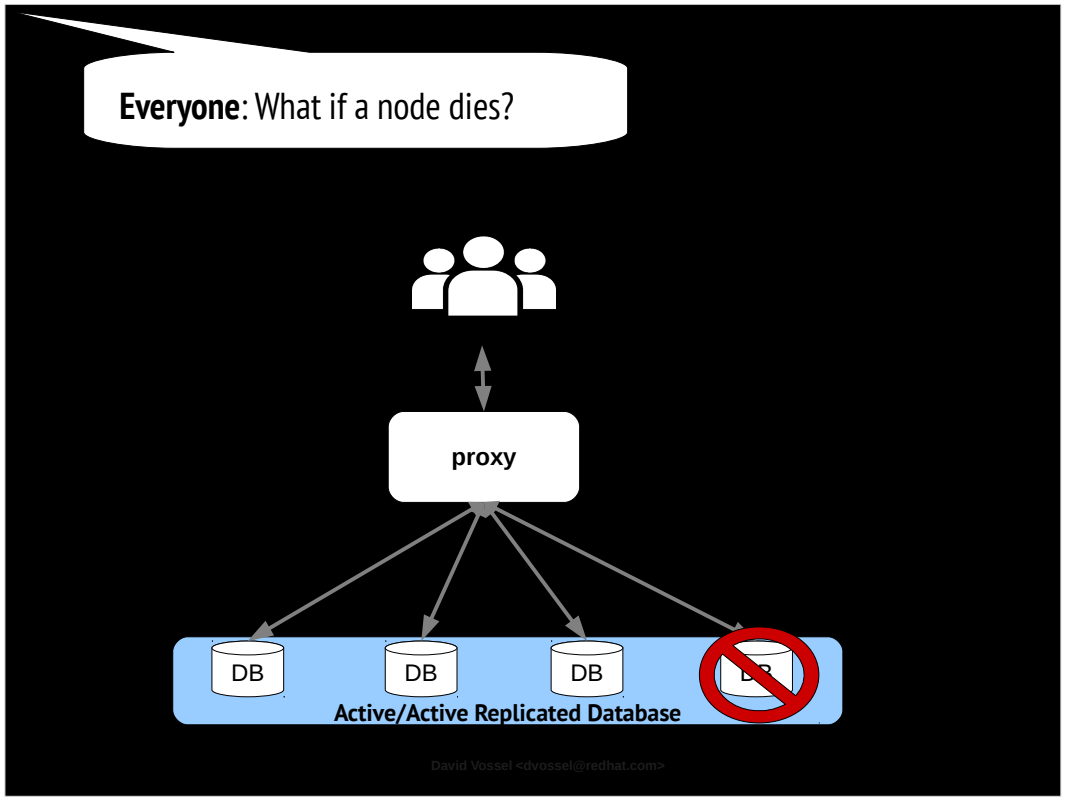
NO Problem.. i got this. load balancing is what I do!



And everyone was happy.



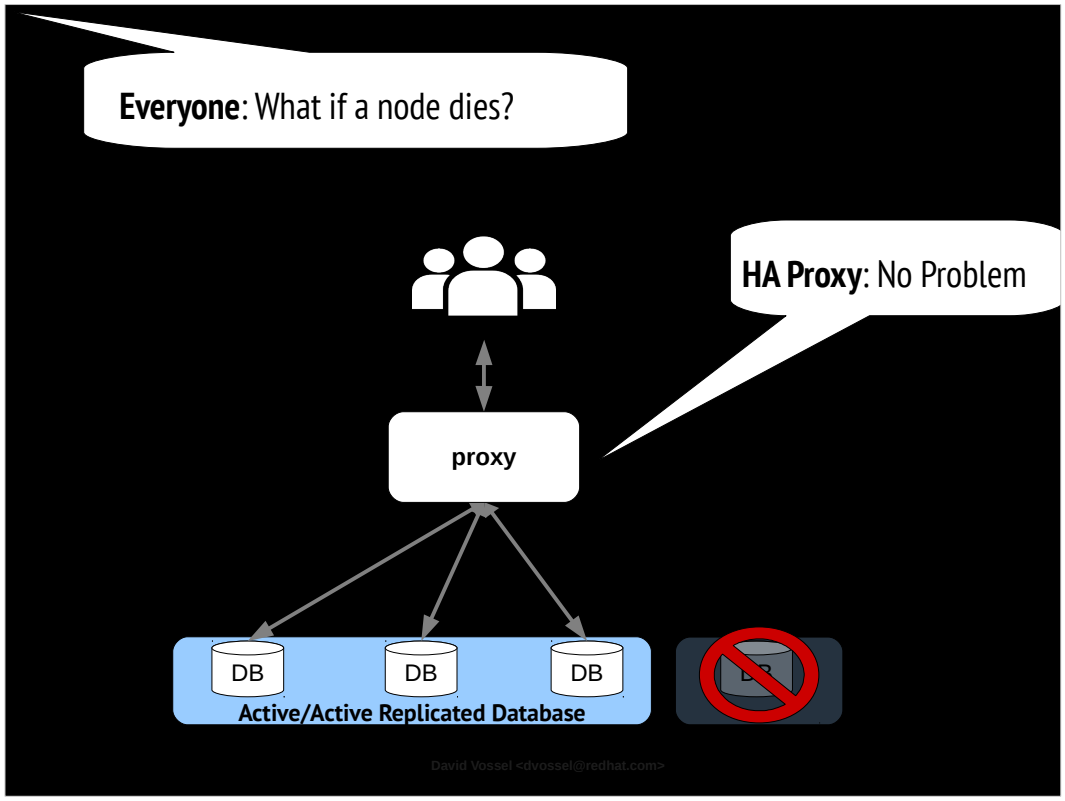
Then everyone began thinking again.



So, What happens if a **database node dies?**

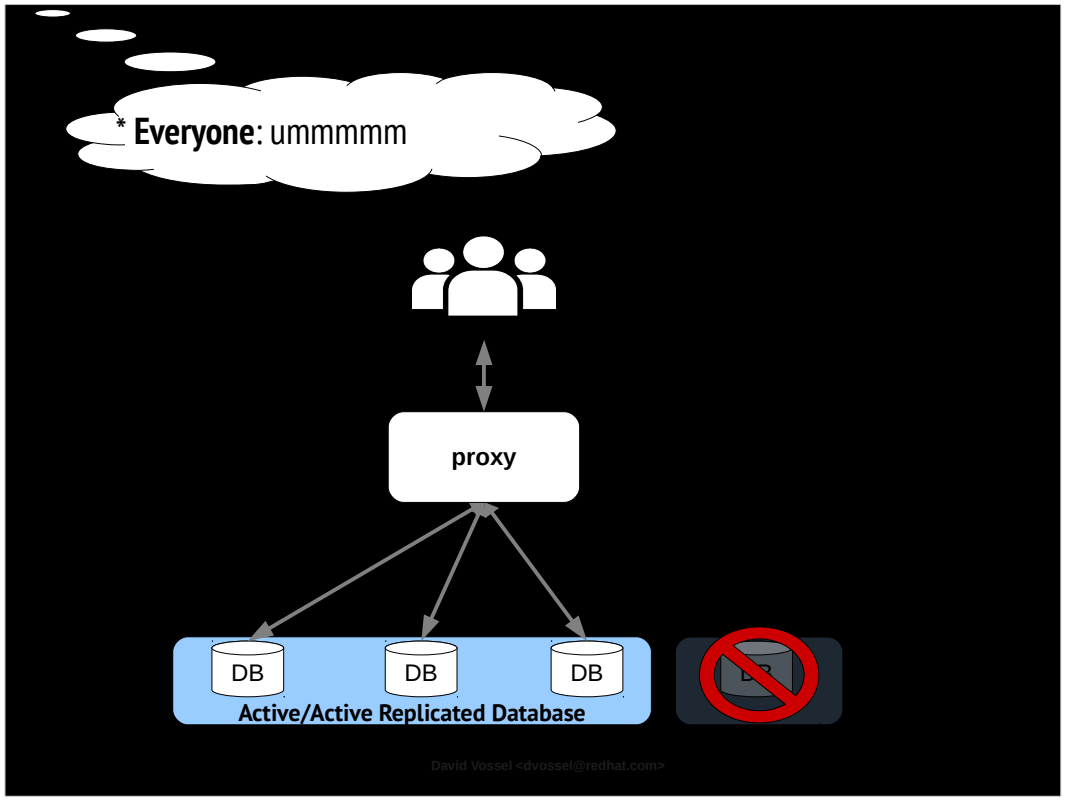
Does that mean some client requests just stop working or something?

And to this HAPROXY said...



No problem. I got this.

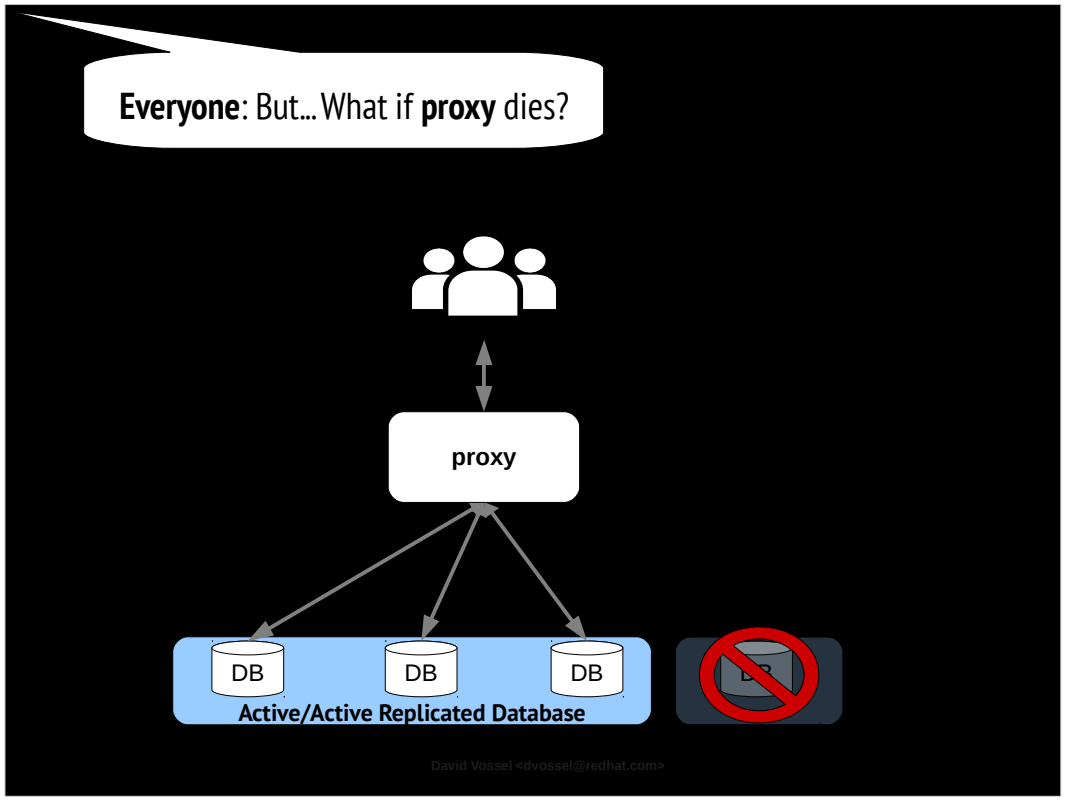
Database instance disappears, we **stop routing** clients to it.



But everyone wasn't quite convinced.

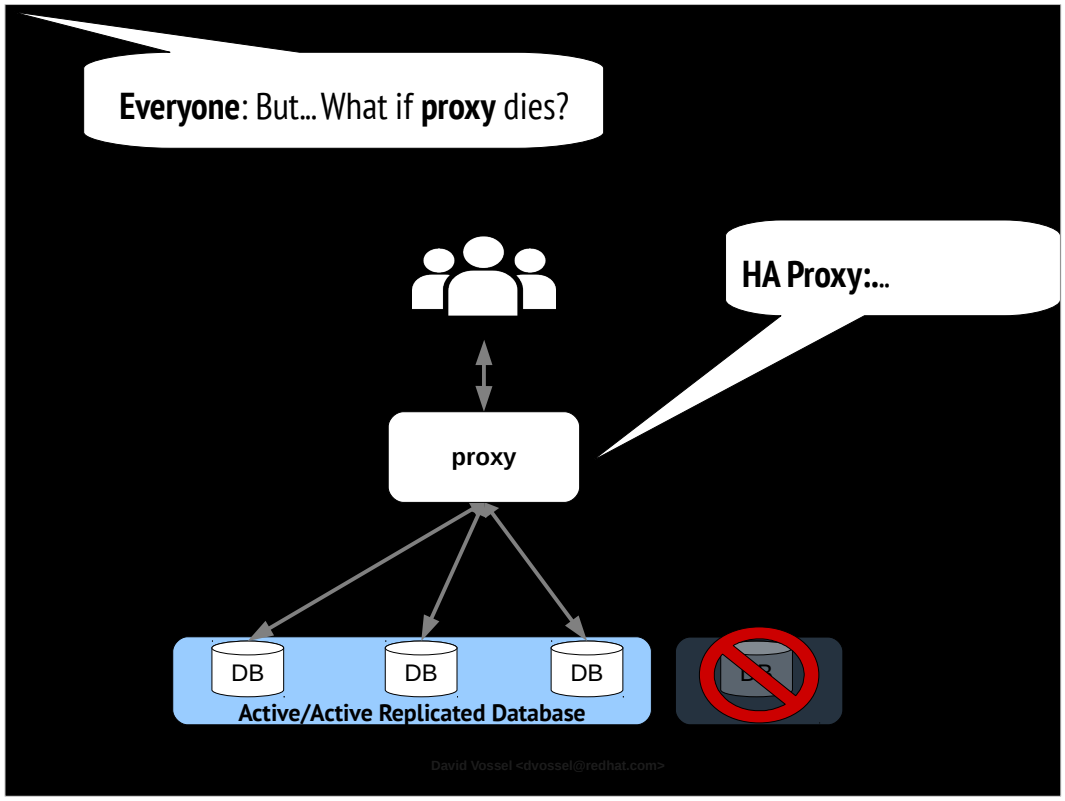
Thinking about availability

Asks HAProxy another question.



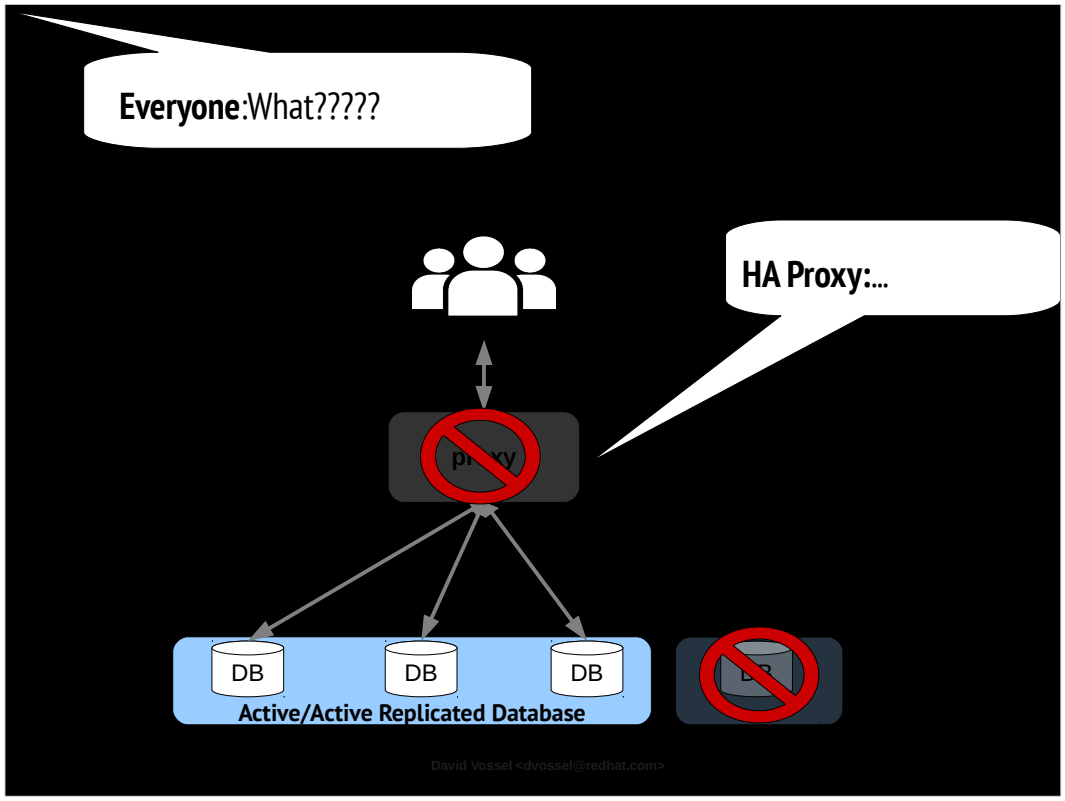
So, what happens if the **load balancer dies?**

And to this HAProxy says.

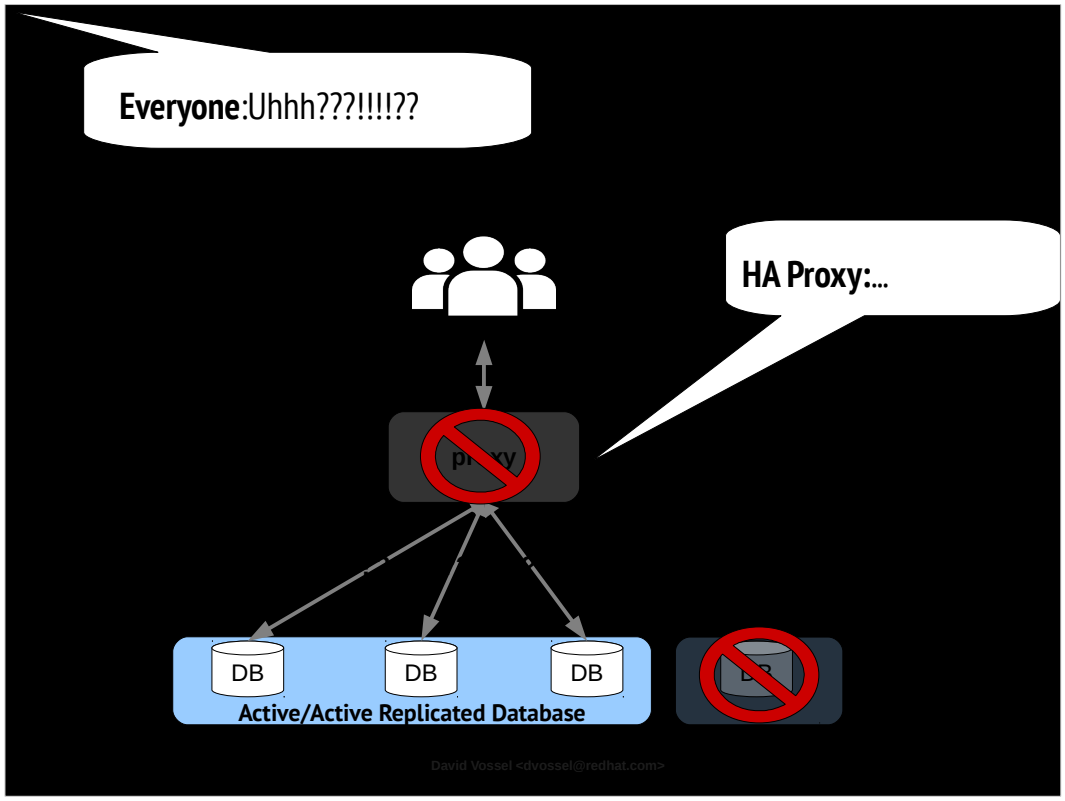


Nothing.

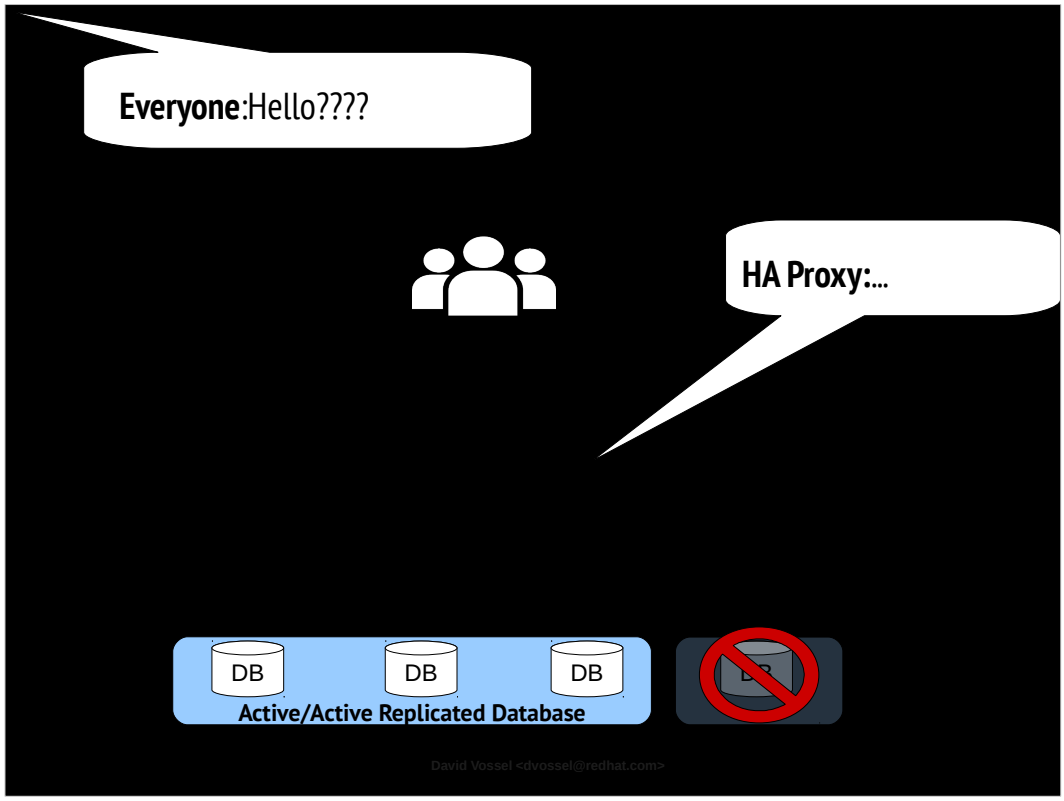
Skip next



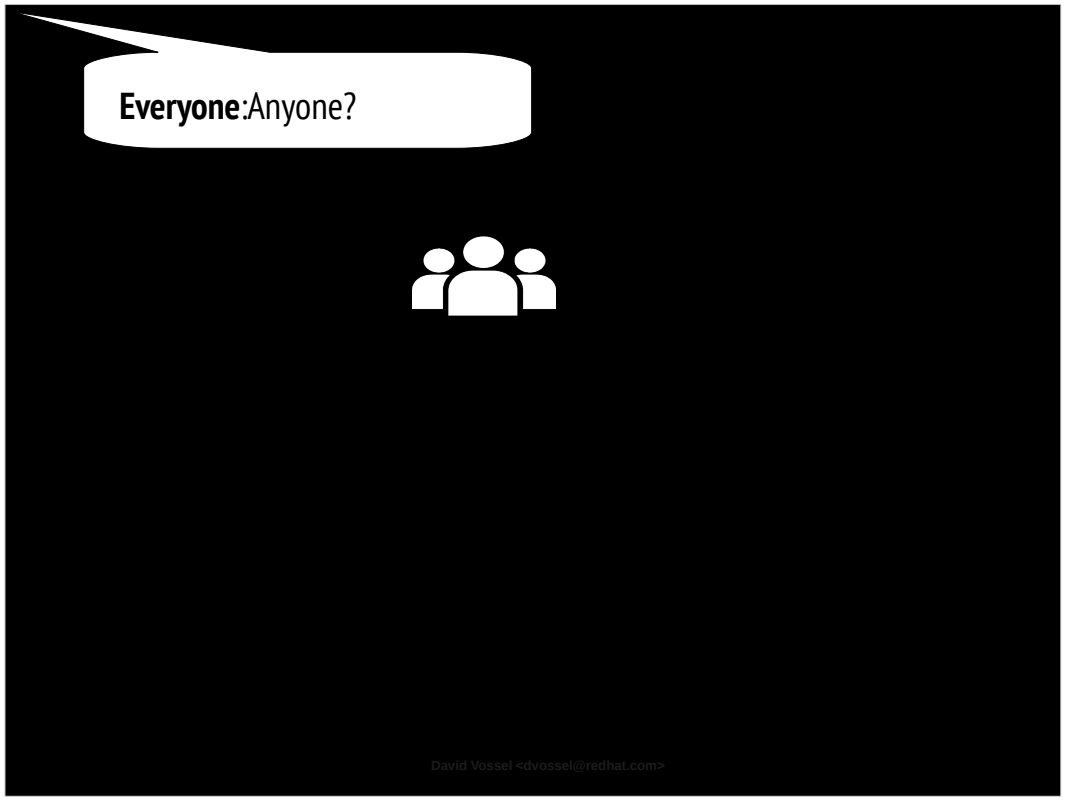
And **because HA proxy was unresponsive** everyone got to see **exactly what happens** when the **load balancer dies.**



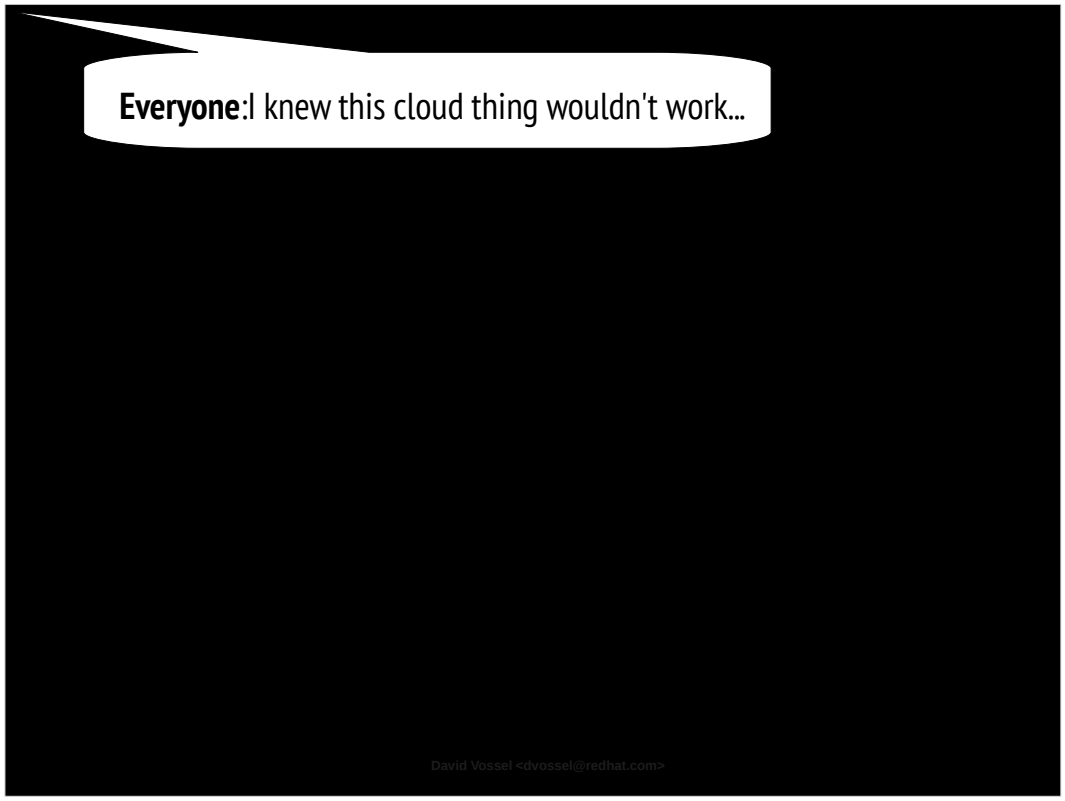
Clients are no longer connected to the database.

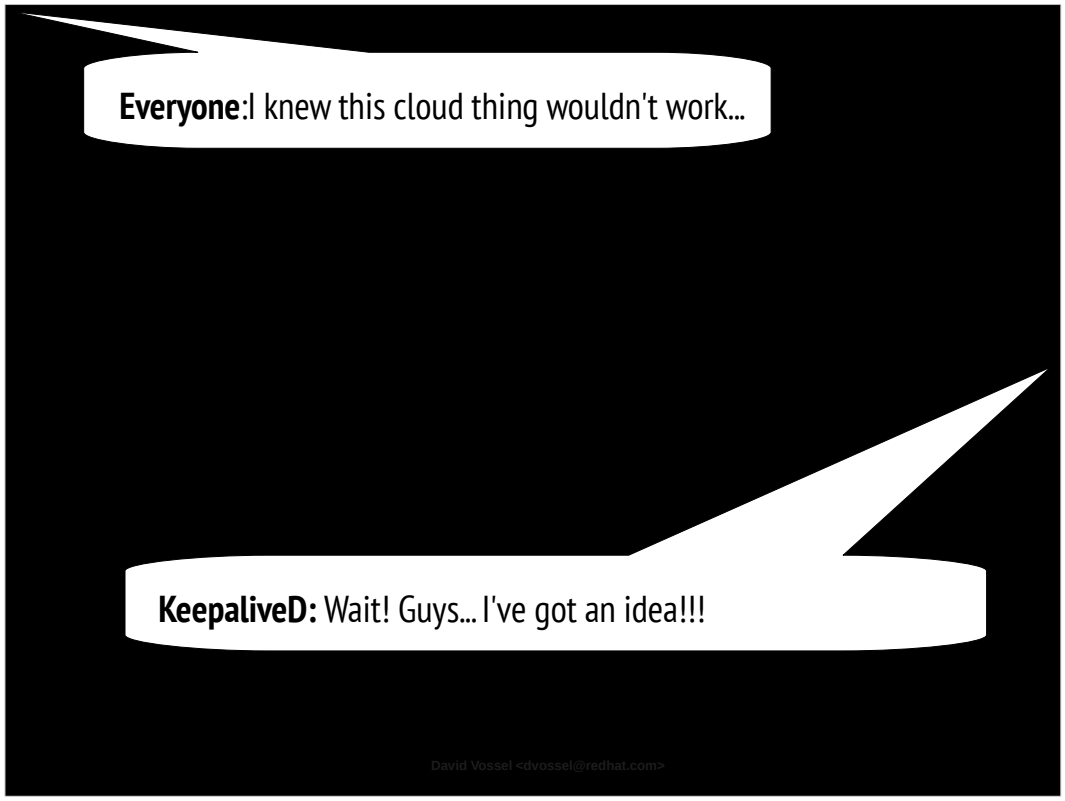


Everything grinds to a halt



And its not exactly what everyone had in mind for high availability.

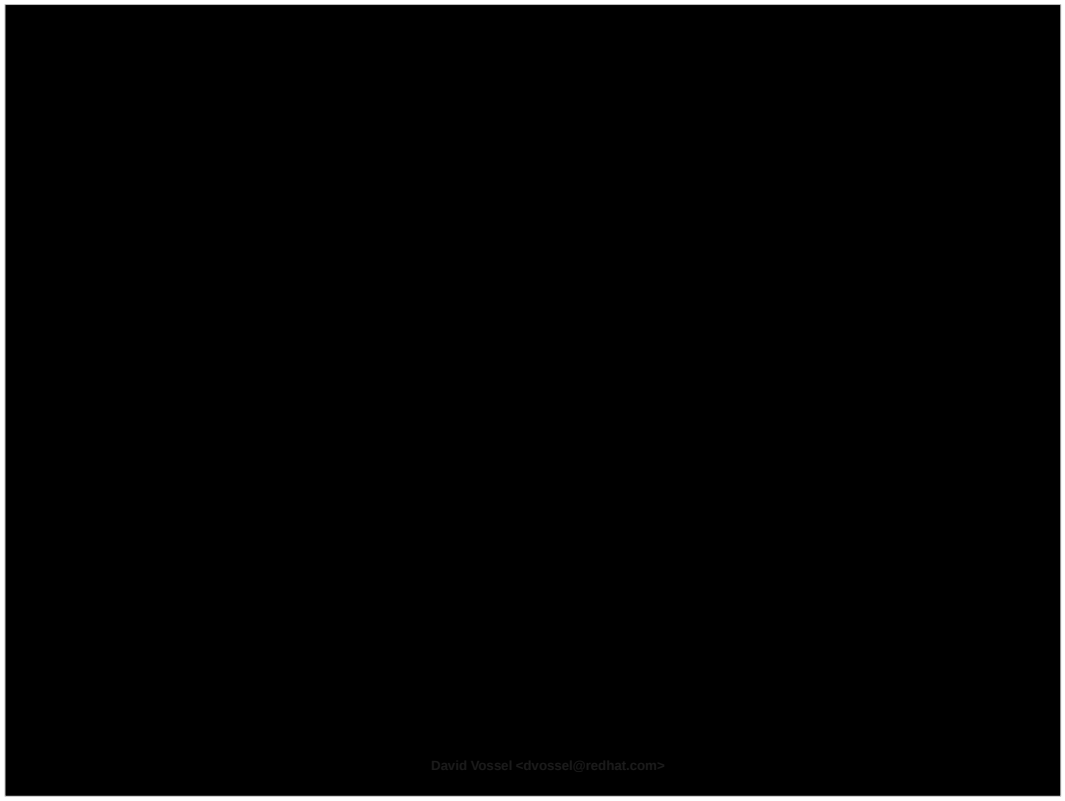

Everyone:I knew this cloud thing wouldn't work...
David Vossel <dvossel@redhat.com>



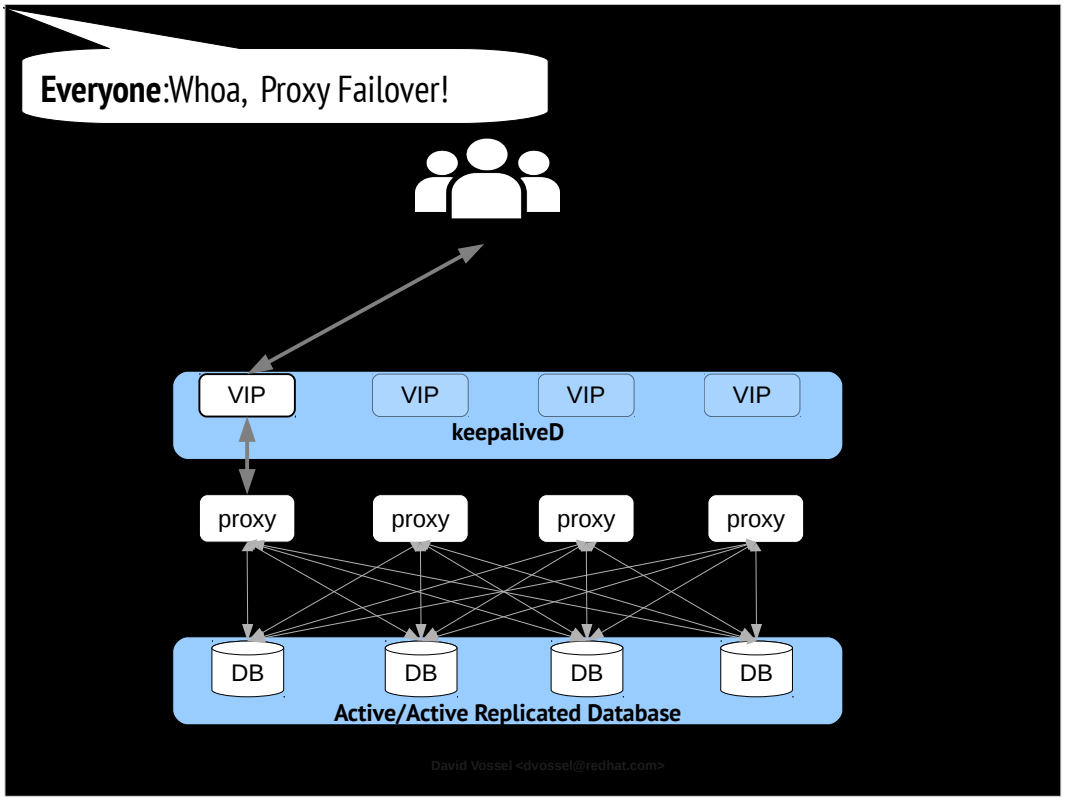
So, the obvious problem is

how do we make the **load balancer Highly Available?**

The database is fault tolerate in itself, but the system as a whole is not fault tolerant.

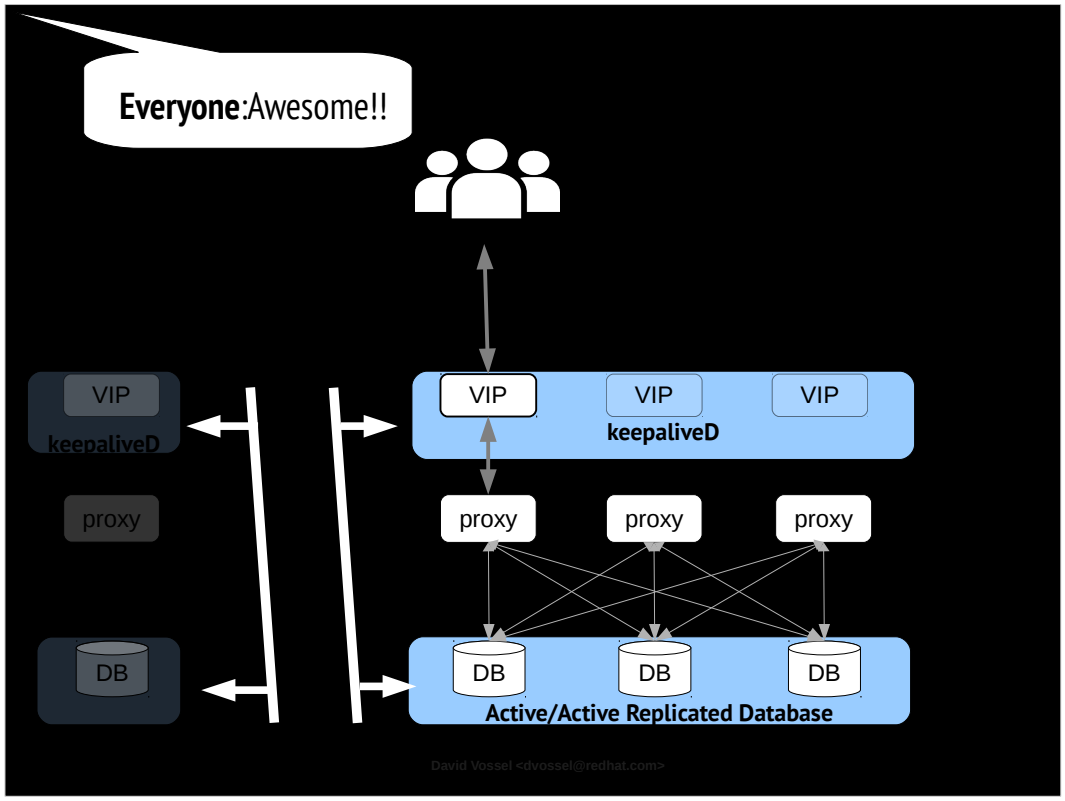


For this. KeepaliveD enters the scene



With keepaliveD, we Now have redundant proxies

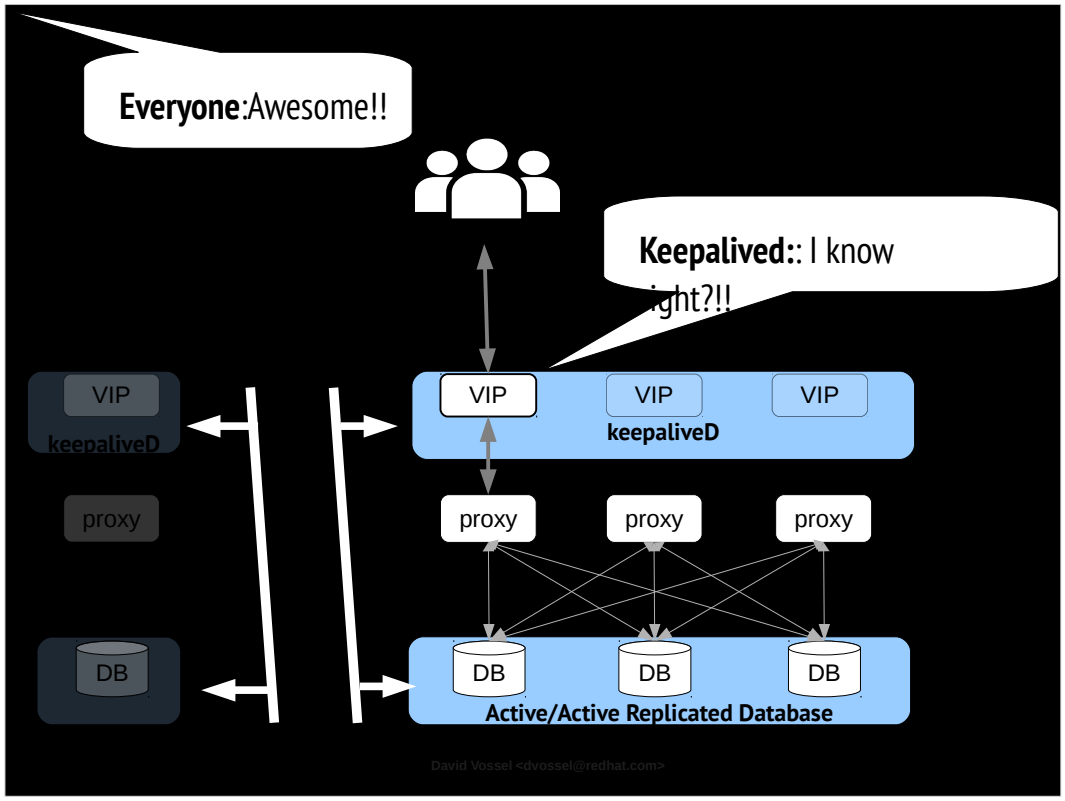
Keepalived works by migrating a front end Virtual IP address between redudant HA proxy instances.



So,

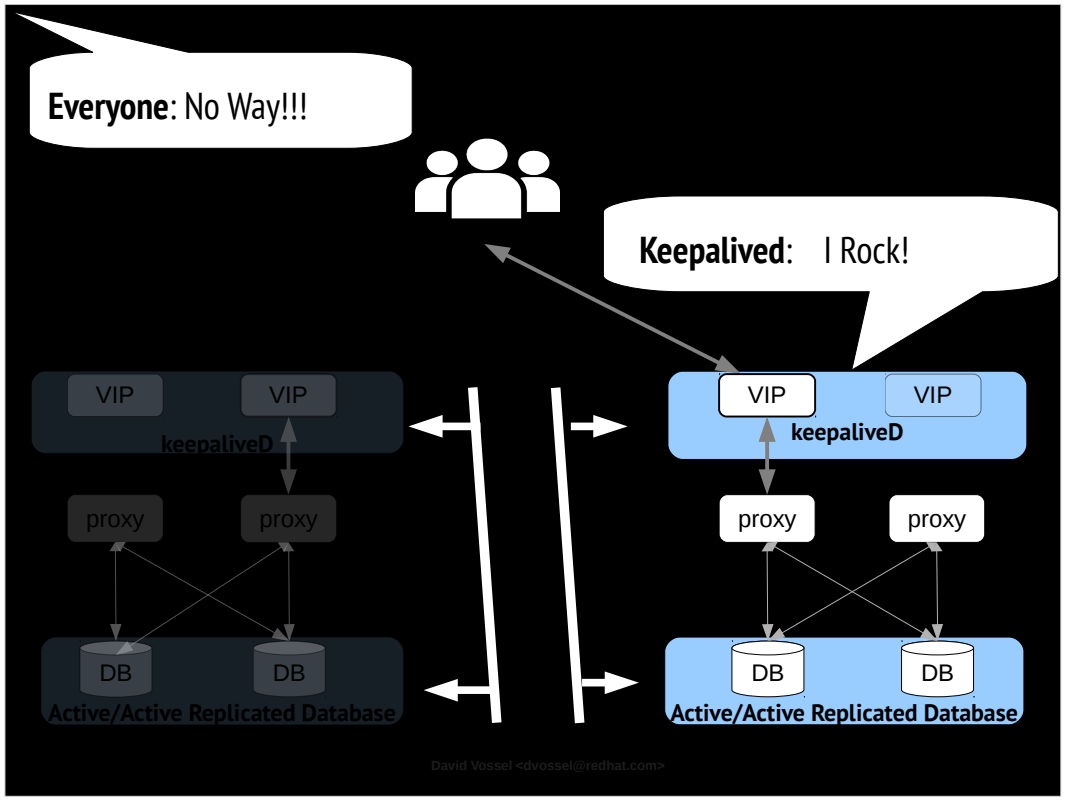
If **one Haproxy instance dies**

KeepaliveD **Moves VIP to another instance**



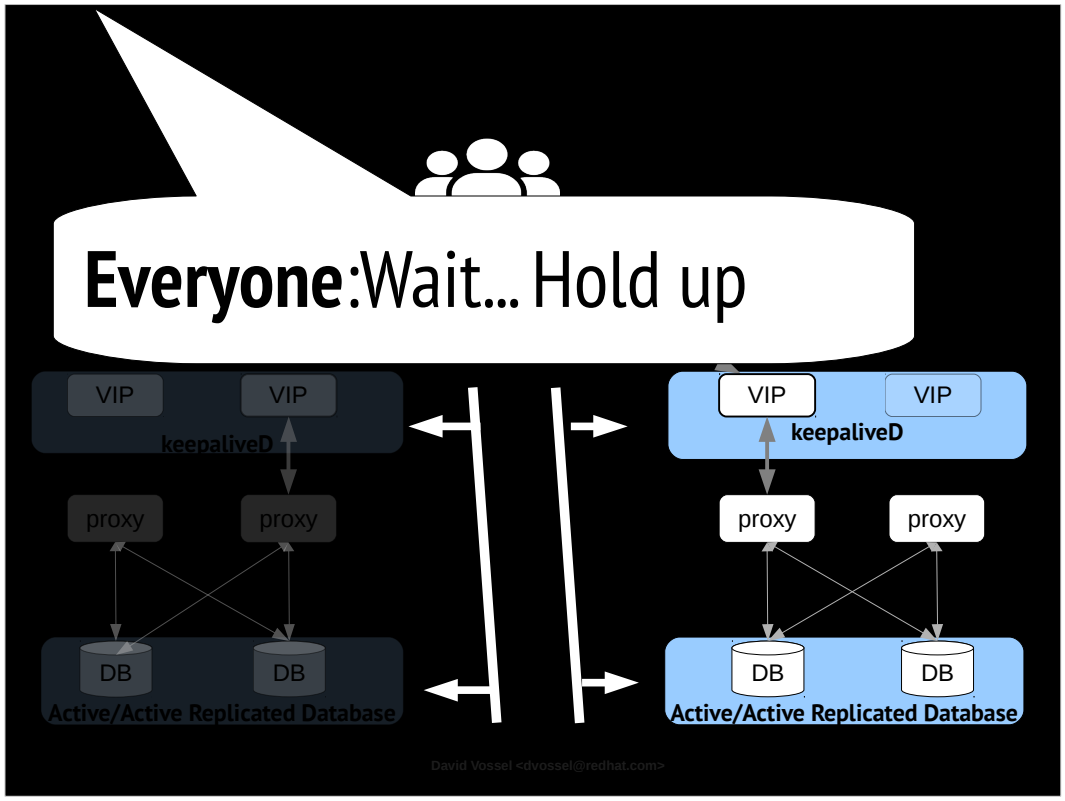
Clients requests **keep getting processed.**

They **don't even notice** that database **instances** and load balancer instances are **going on and off line.**



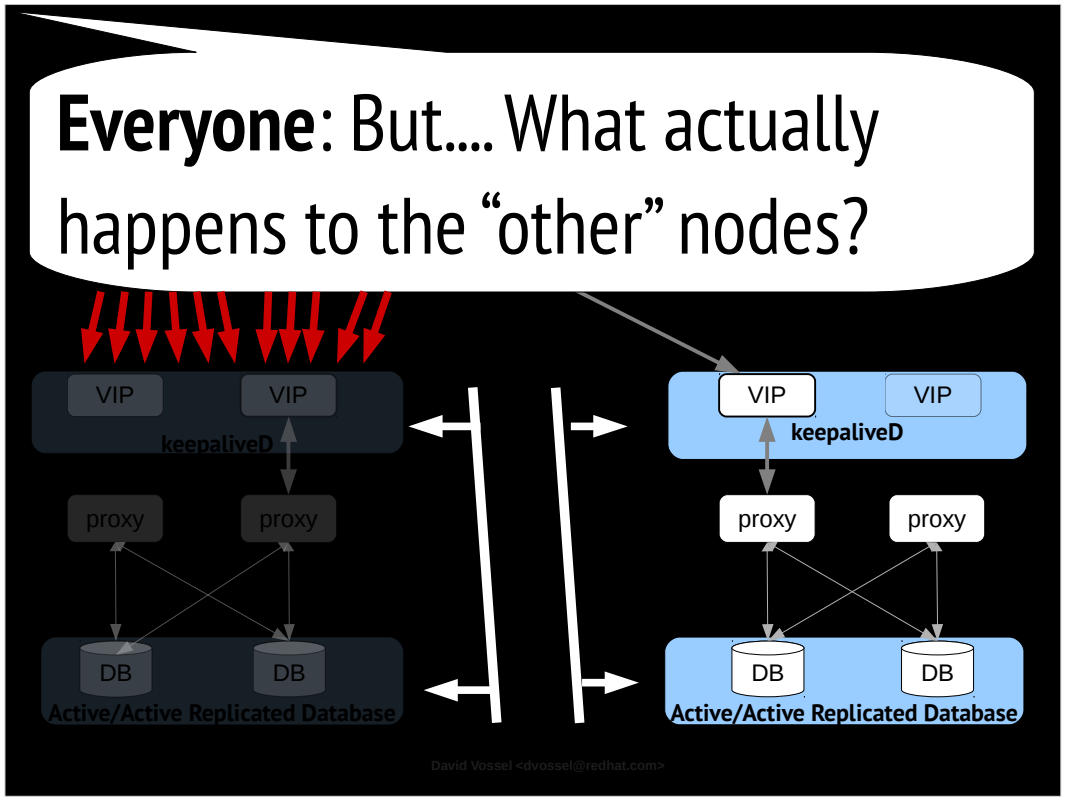
Everything just kind of works for them

Which granted, is kind of Awesome...



But there's a problem.

Skip next.



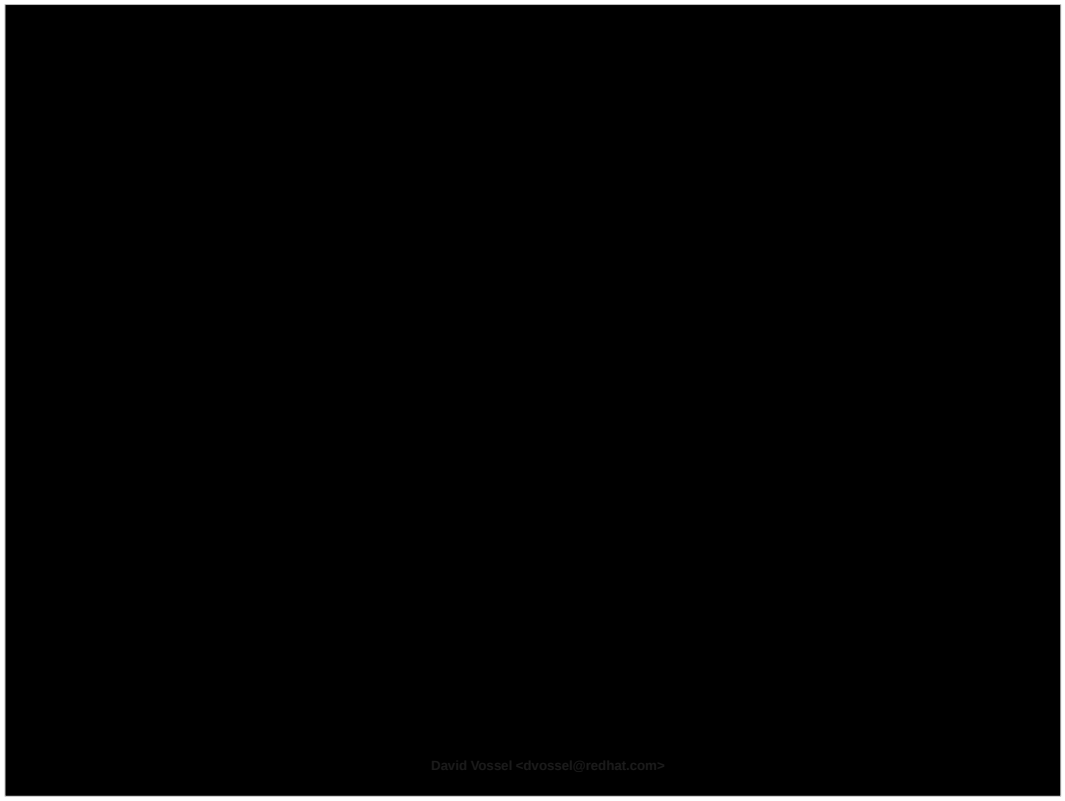
Everyone asks one LAST VERY important question.

But.. whta actually happens to the other nodes...

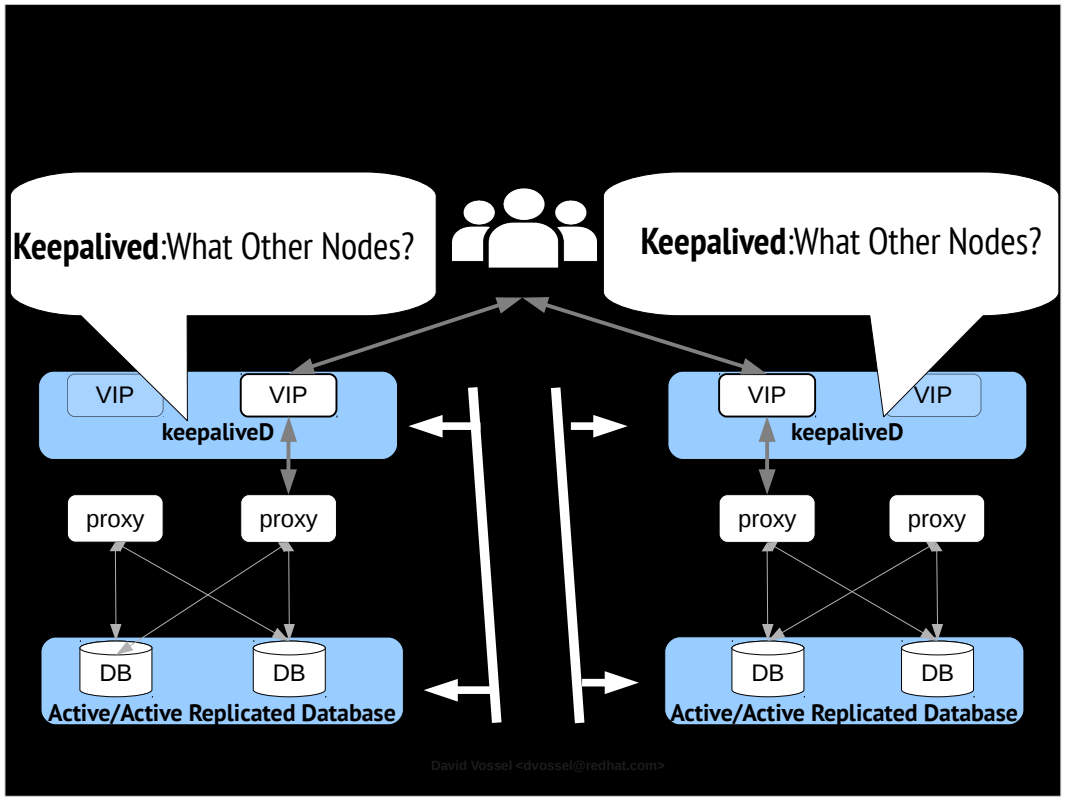
The other nodes that are greyed out.

The ones you are telling me are unresponsive.

Next slide (peripeteia)



And **In response** to **the fate** of the other nodes **KeepaliveD says.**



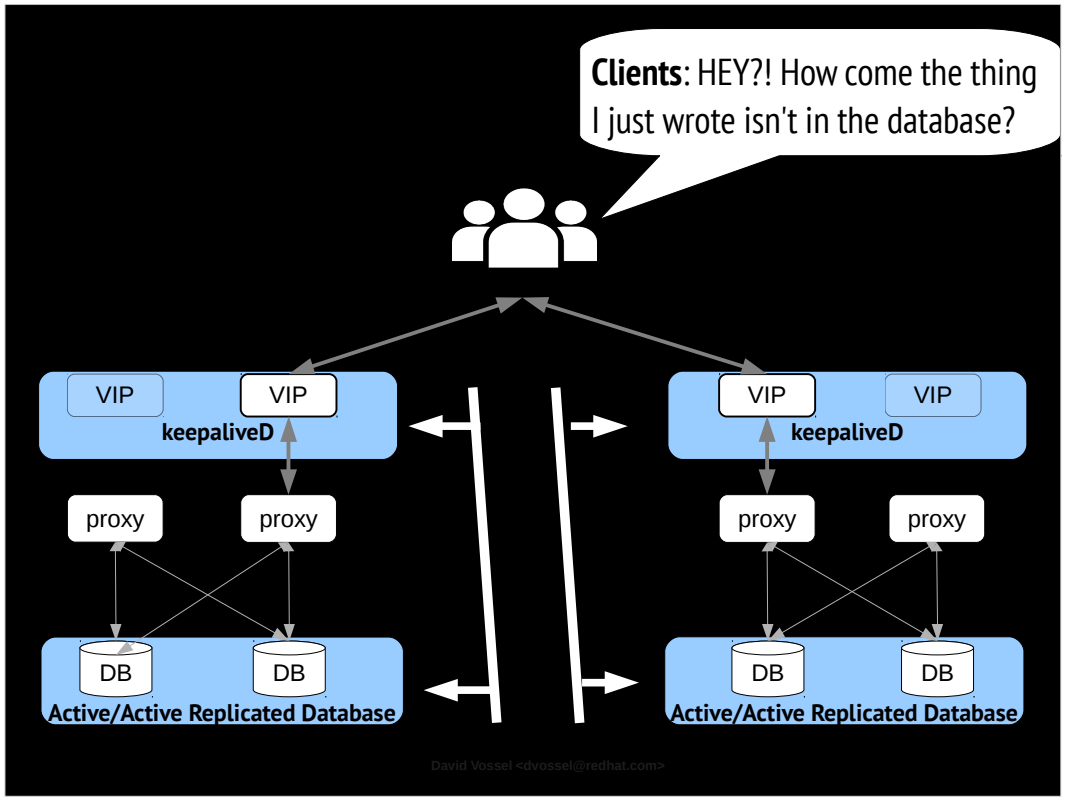
What other nodes?

Except the answer came as an echo.

Because the reality is the **other instances weren't unresponsive at all.**

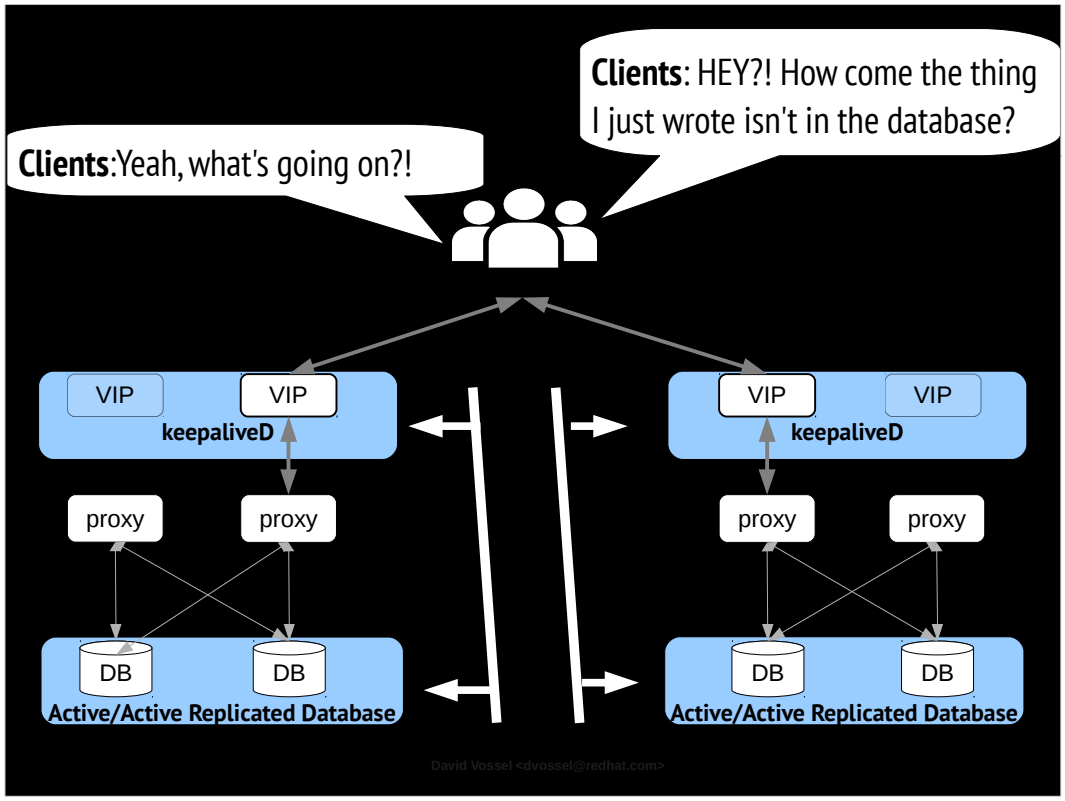
There was a **network split**

And **two separate partitions had formed**

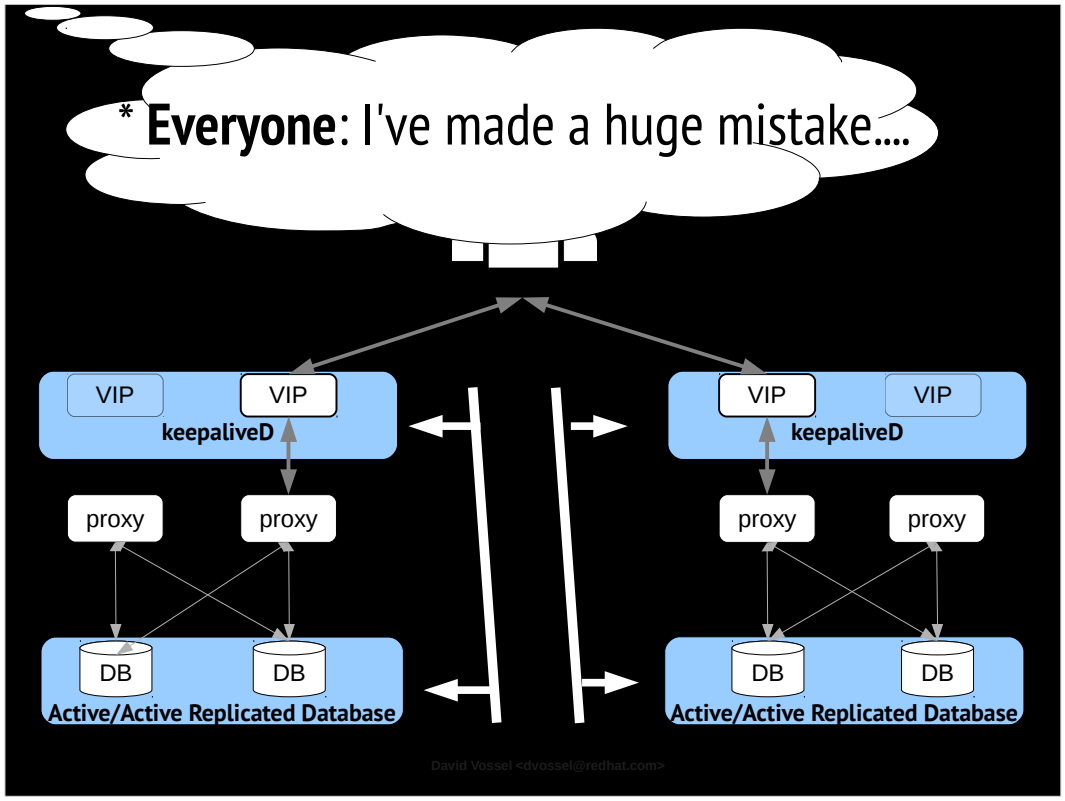


Clients start seeing weird things.

They **write things to the database**, but can't read the values back.

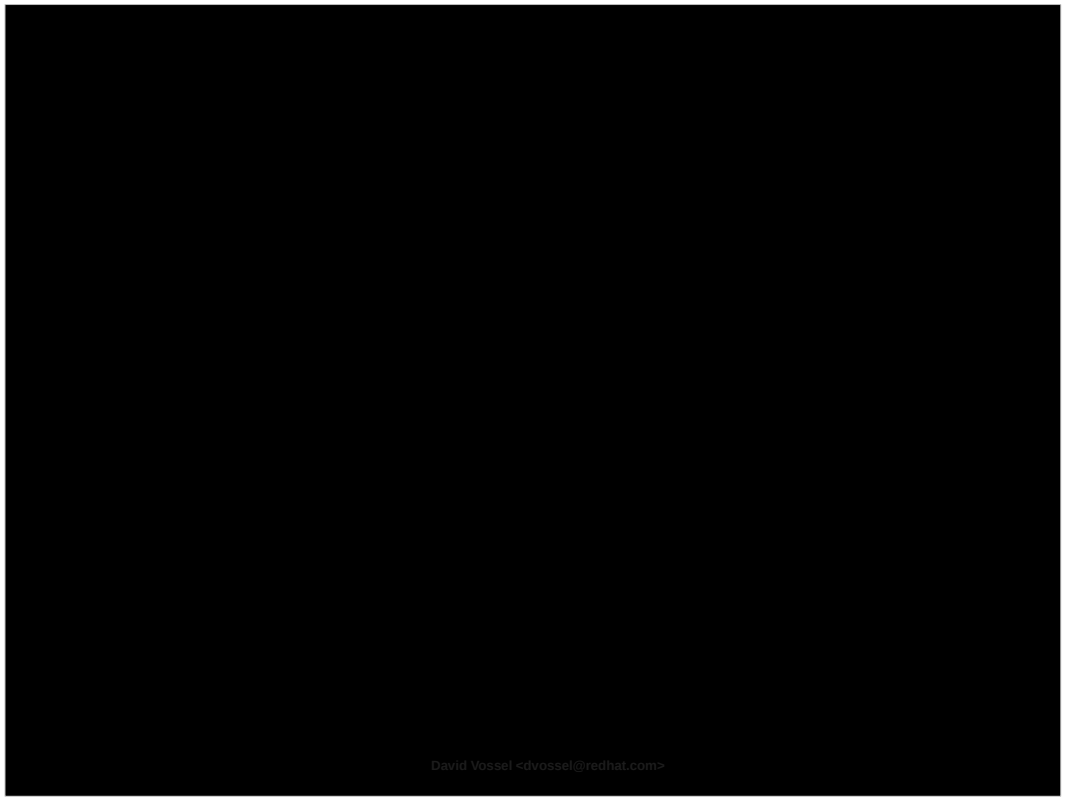


Other clients are seeing even weirder things.



Eveyrone hung their head in shame and declared they've made **a huge mistake.**

But they couldn't figure out where they went wrong.



They had Distributed fault tolerate database

They had Redundant Load Balancer

But they're having **data consistency issues.**

How can we fix this?

For us solution was....

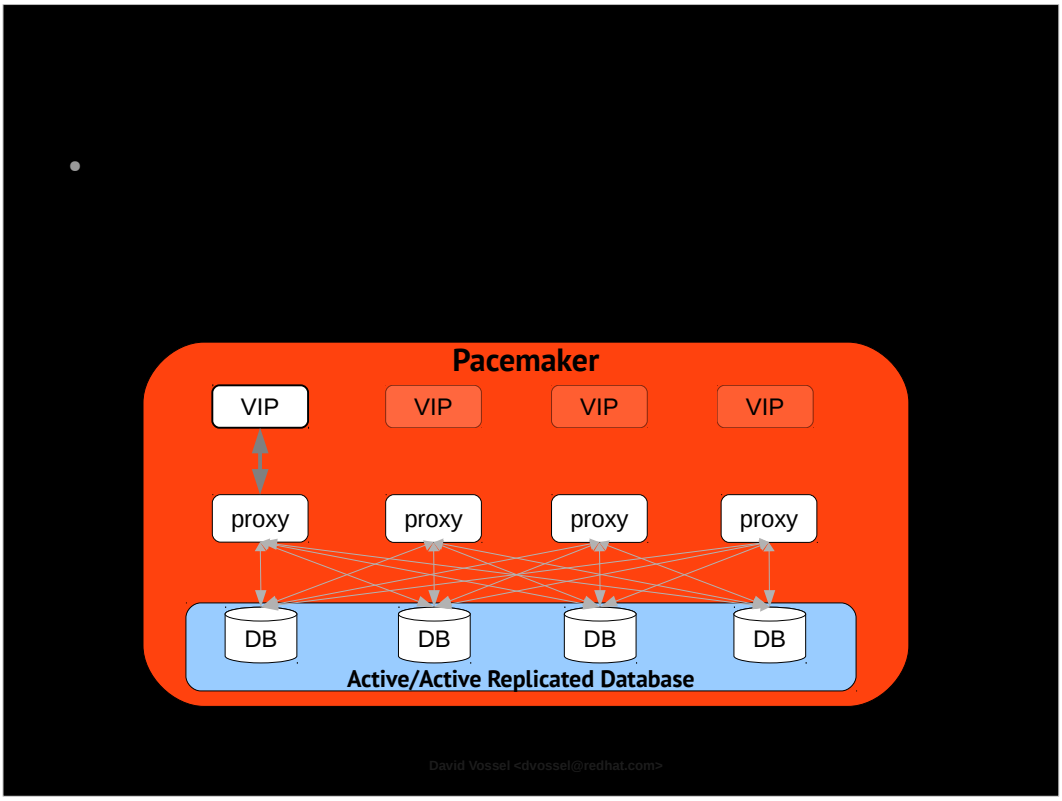
Pacemaker.

Handles prevention of split partitions very well.

Very sophisticated techniques for handling this scenario are native to how pacemaker works

I'll get into this in a second.

But there's quite a bit more that makes pacemaker valuable here than just fixing this scenario.



Pacemaker gives us concept of **System level HA**

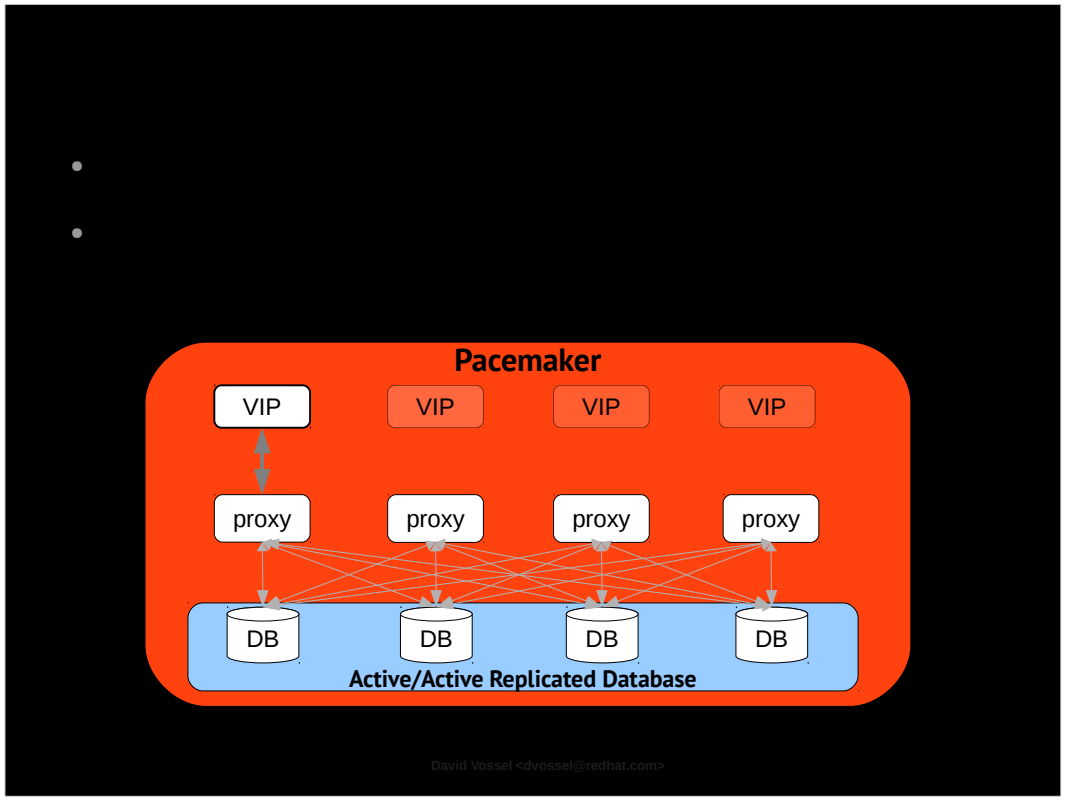
Holistic approach

Which eradicates need for **keekaliveD** at all.

With Pacemaker, every aspect of the **Service Management** is **centralized.**

Pacemaker controls
- Front end Ip Address paired HAProxy
- and controls **recovery and monitoring** of every element in the cluster.

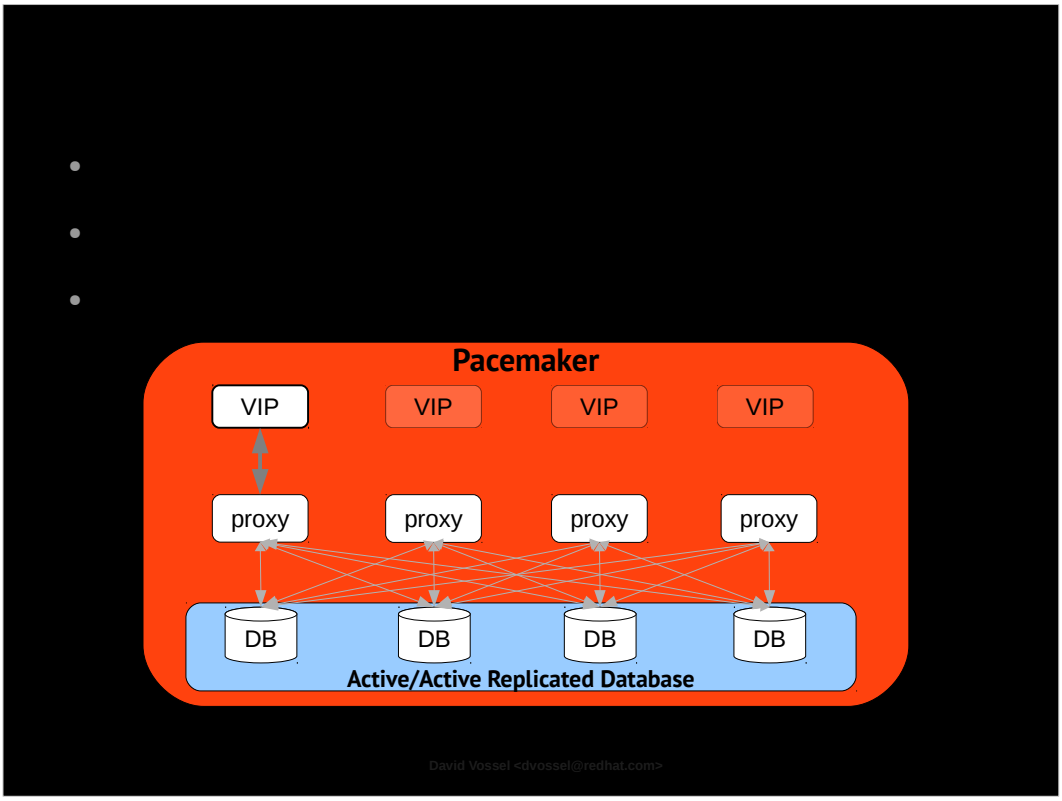
No sitting back and watching. Take action.



Pacemaker works by letting users define a **POLICY** for **managing services** across a **distributed set of nodes.**

Define what services pacemaker should run

And where those services should run



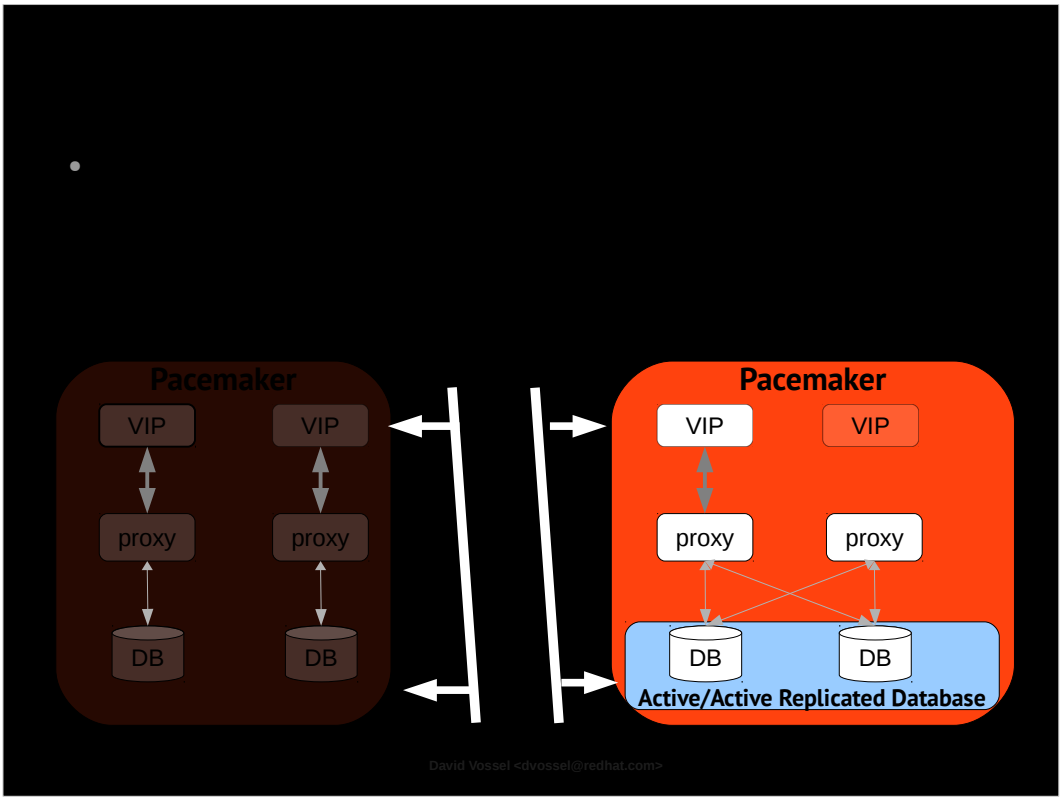
After defining policy.

pacemaker has the ability to **ENFORCE that policy** to achieve **system wide deterministic behavior.**

No unknown states.

Don't question what happens if a service fails.

We can simulate that failure and know exactly



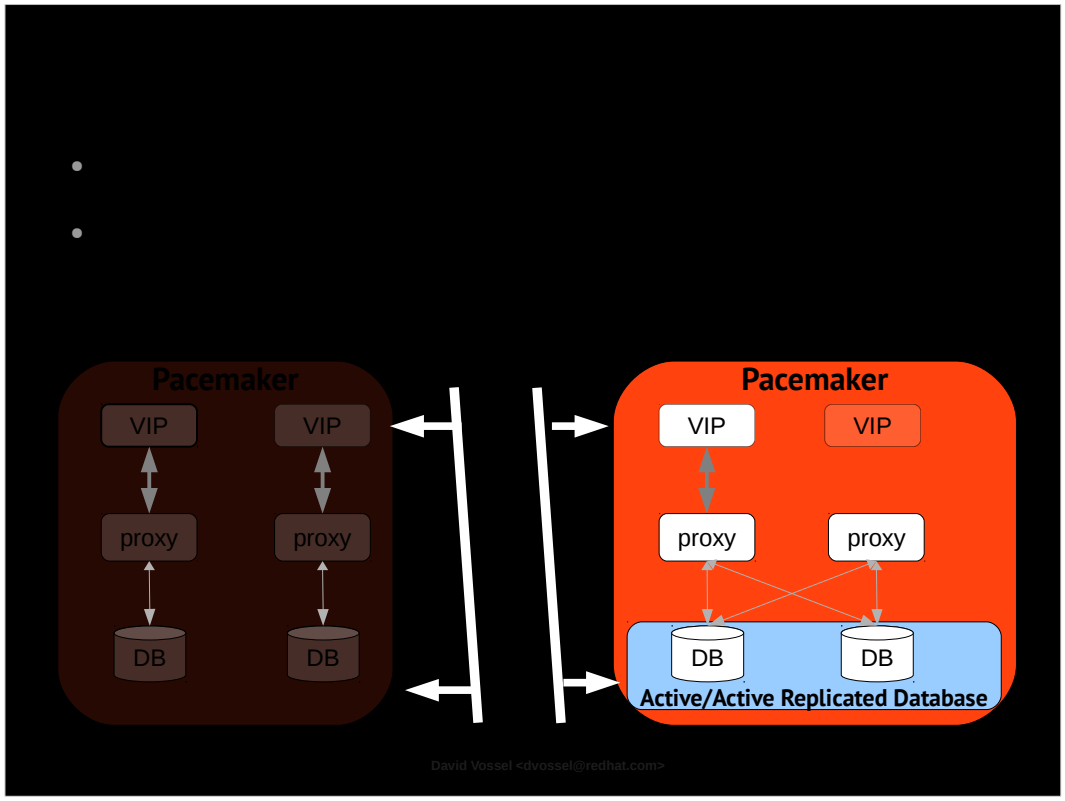
Back to the story...

With Pacemaker we don't question this scenario.

Pacemaker is very capable of handling partition splits.

We know exactly what happened to the unresponsive nodes.

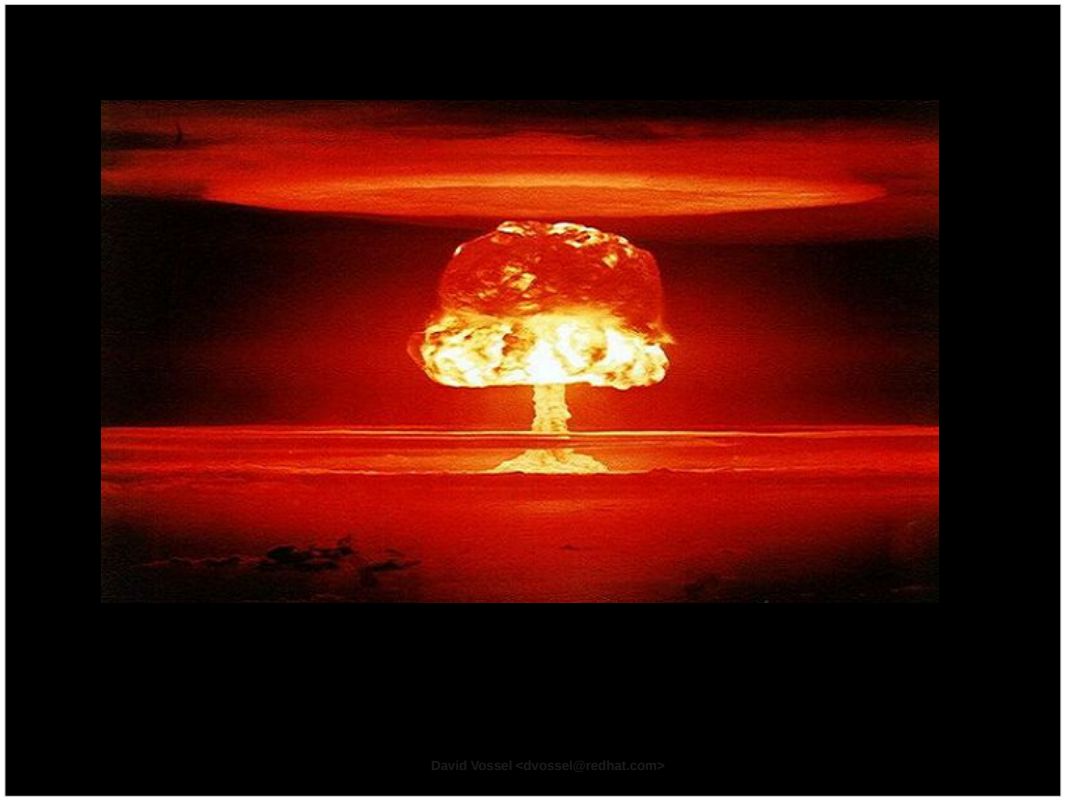
Next slide...



And how do we know this?

1. our quorum provider is super smart and knows how to handle tie breakers.

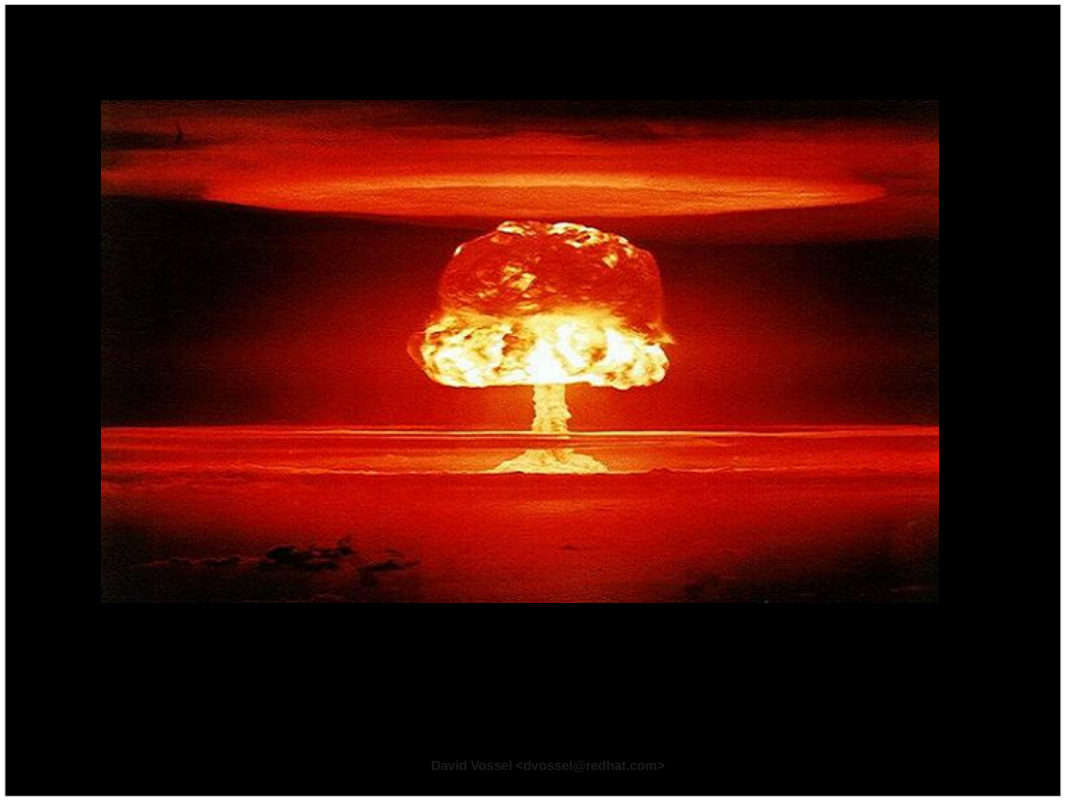
2. <next slide>



Stonith.

Stonith is pacemaker's fencing daemon.

Stonith is a large part of what makes pacemaker capable of enforcing deterministic behavior.



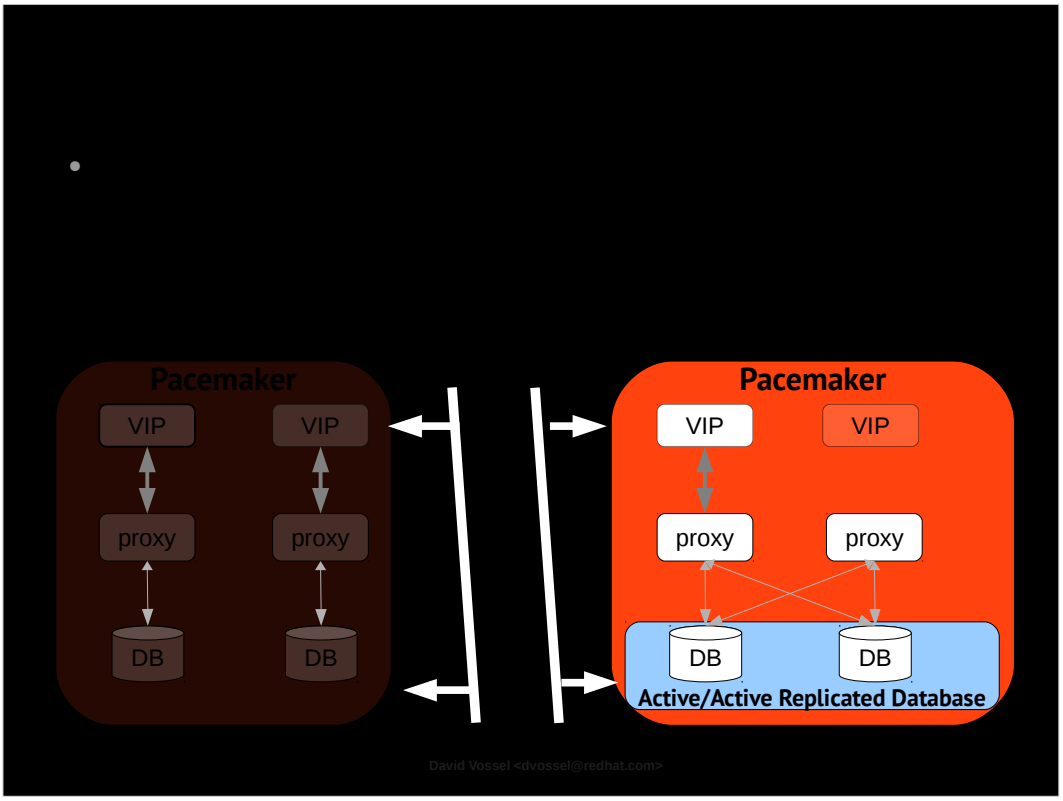
Acronym for. Shoot the other node in the head.

If pacemaker is ever in doubt about the state of a node. That node is fenced.

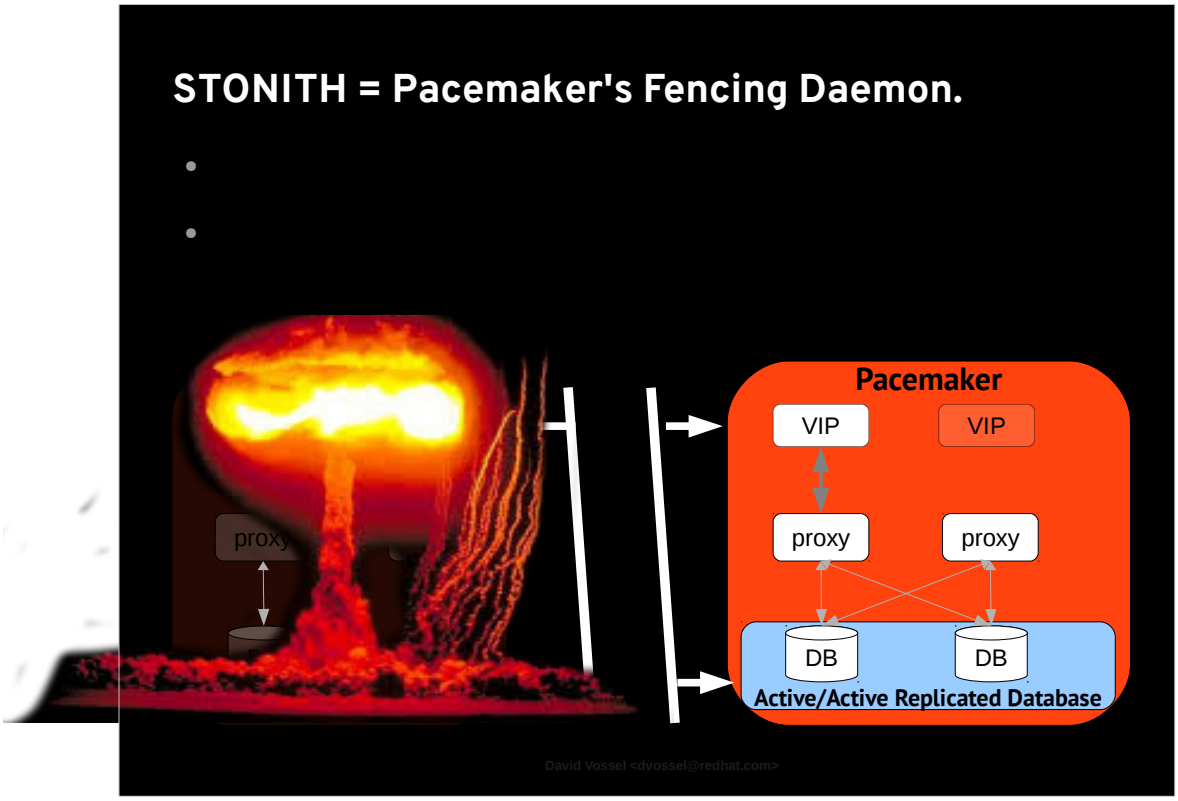
Unfamiliar with fencing? Detach node from partition

Typically Power fencing
at power switch physically kills the power.

This daemon is smart.. really smart.

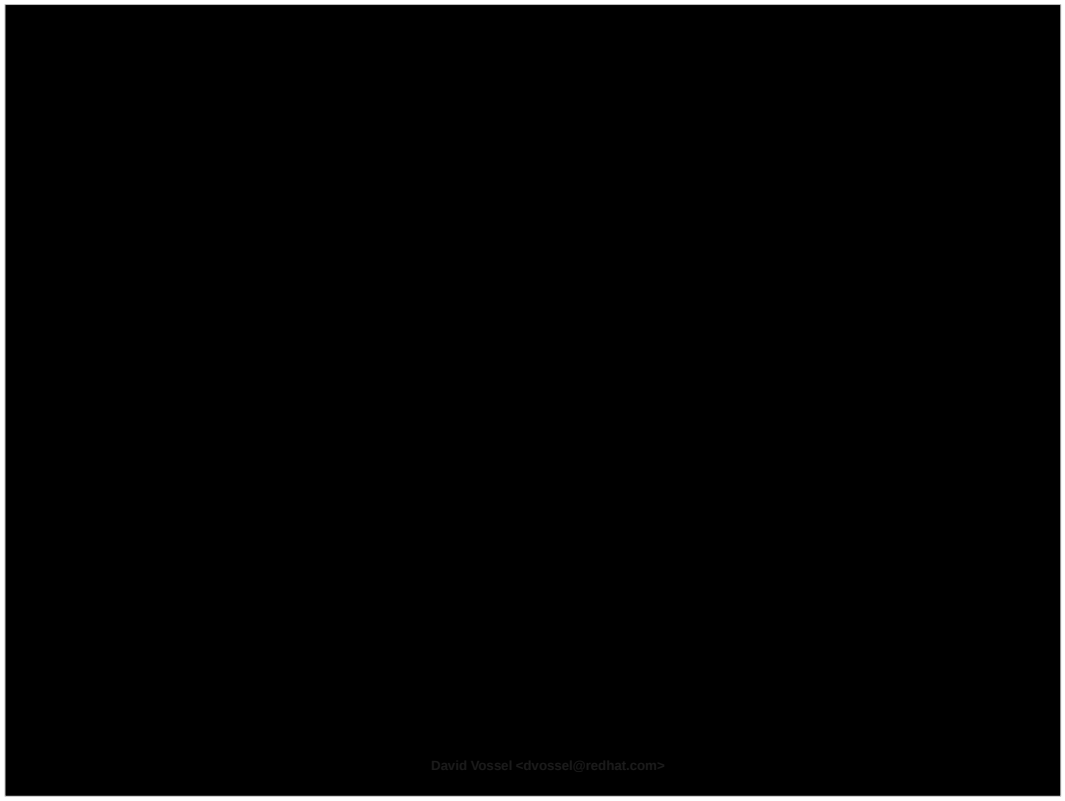


So with STONTIH, we know the state of misbehaving nodes and that state is..... NEXT SLIDE.



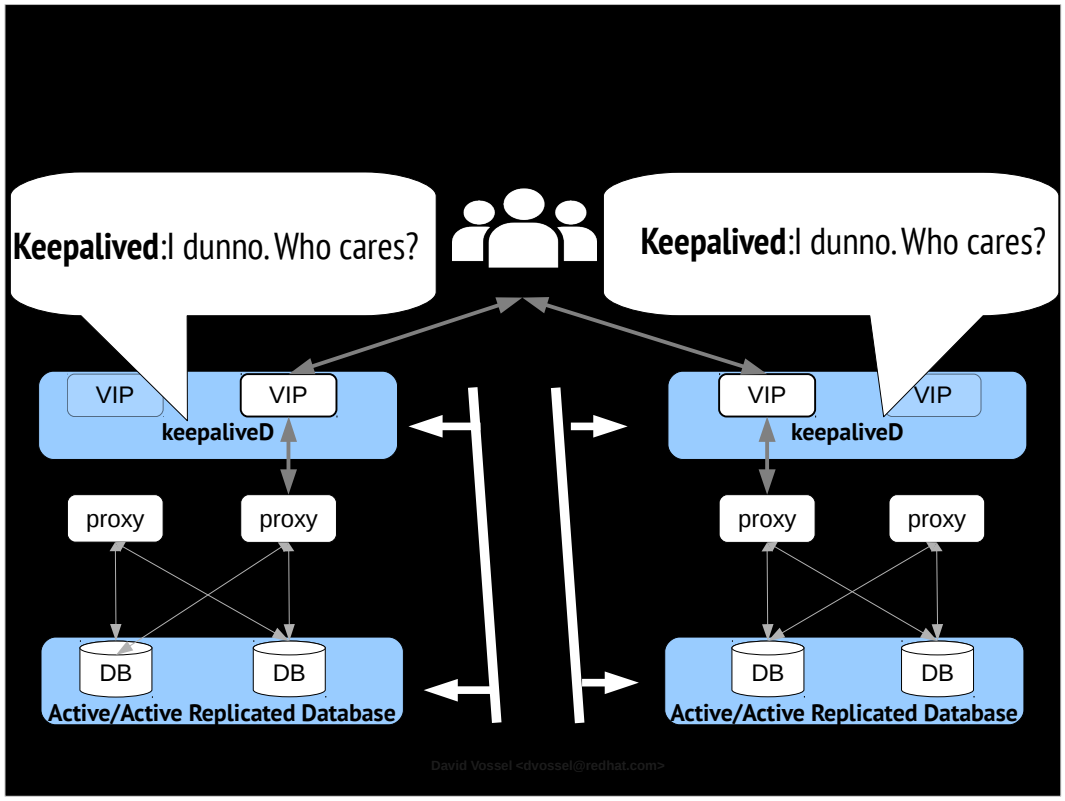
Dead

There's no separate partition forming on dead nodes



Okay,

Lets do a quick recap.



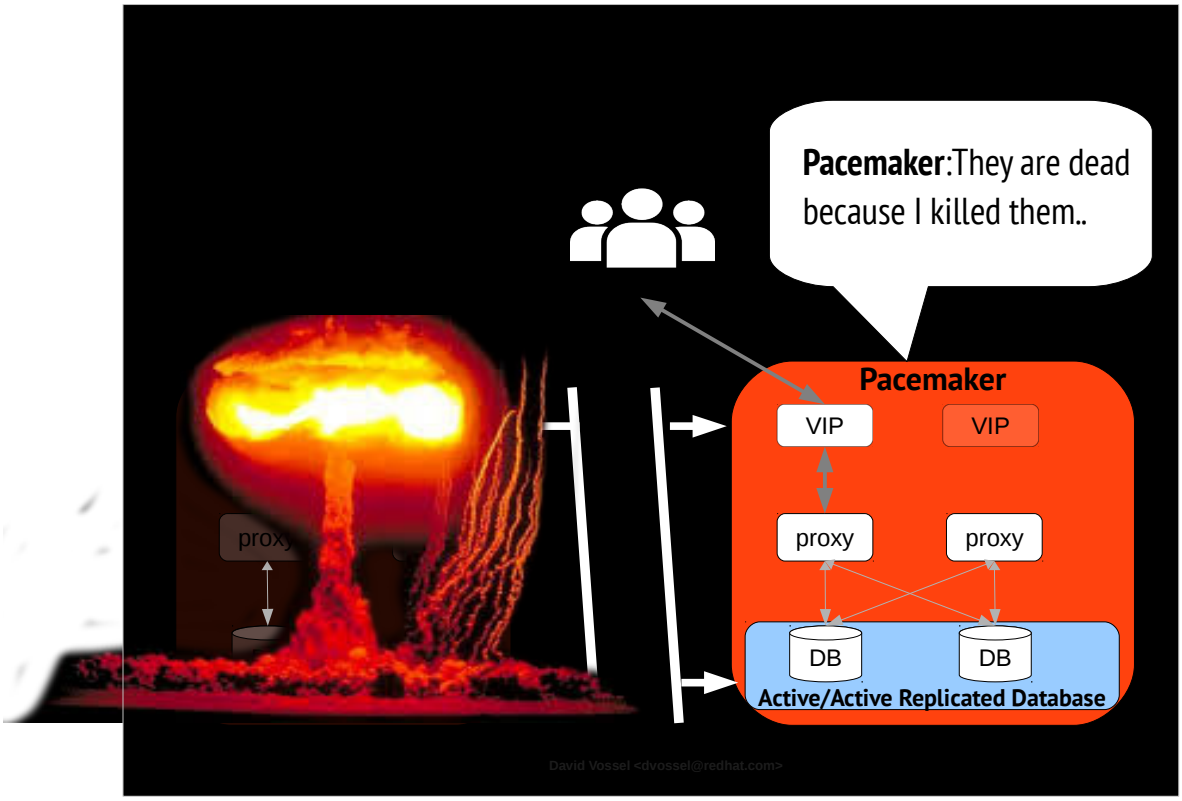
Without pacemaker and stonith

We run the potential to have distributed cluster partition splits

And

We have no centralized way of recovering resources when they fail.

While it's possible. You'd have to work very hard to get keepalived to handle this situation safely.

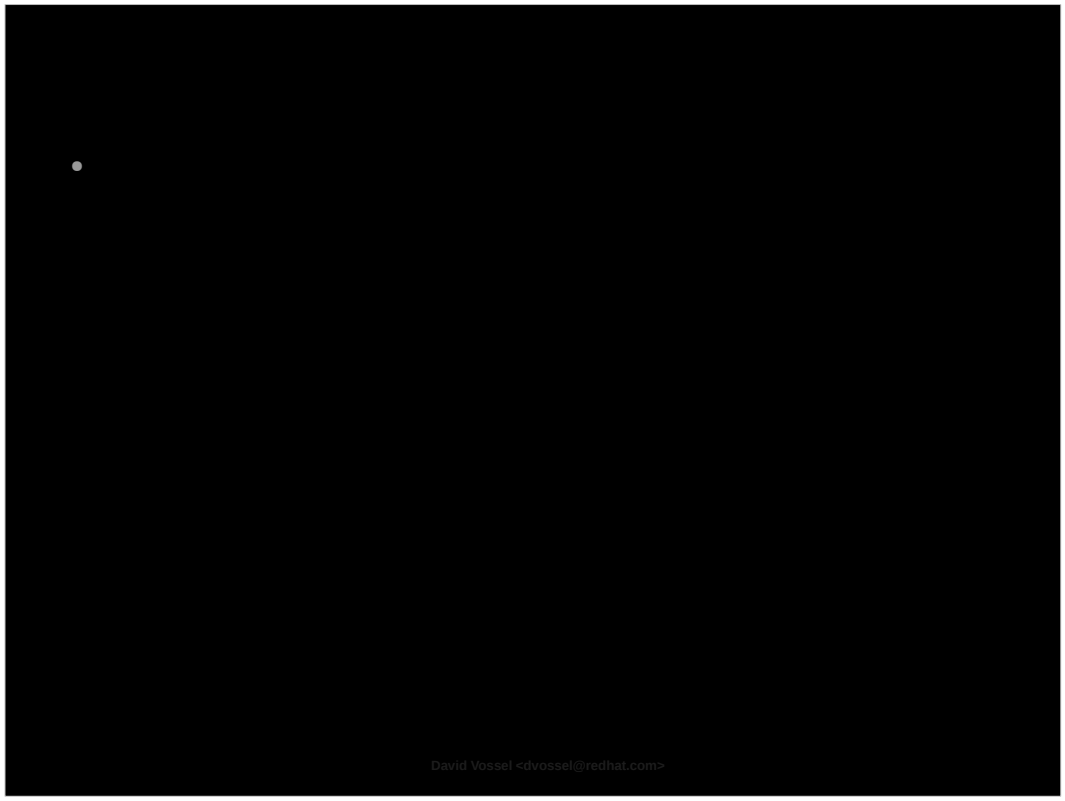


With Pacemaker

We **don't have to worry about partition splits** because **pacemaker's quorum provider and fencing** prevent multiple partitions from existing.

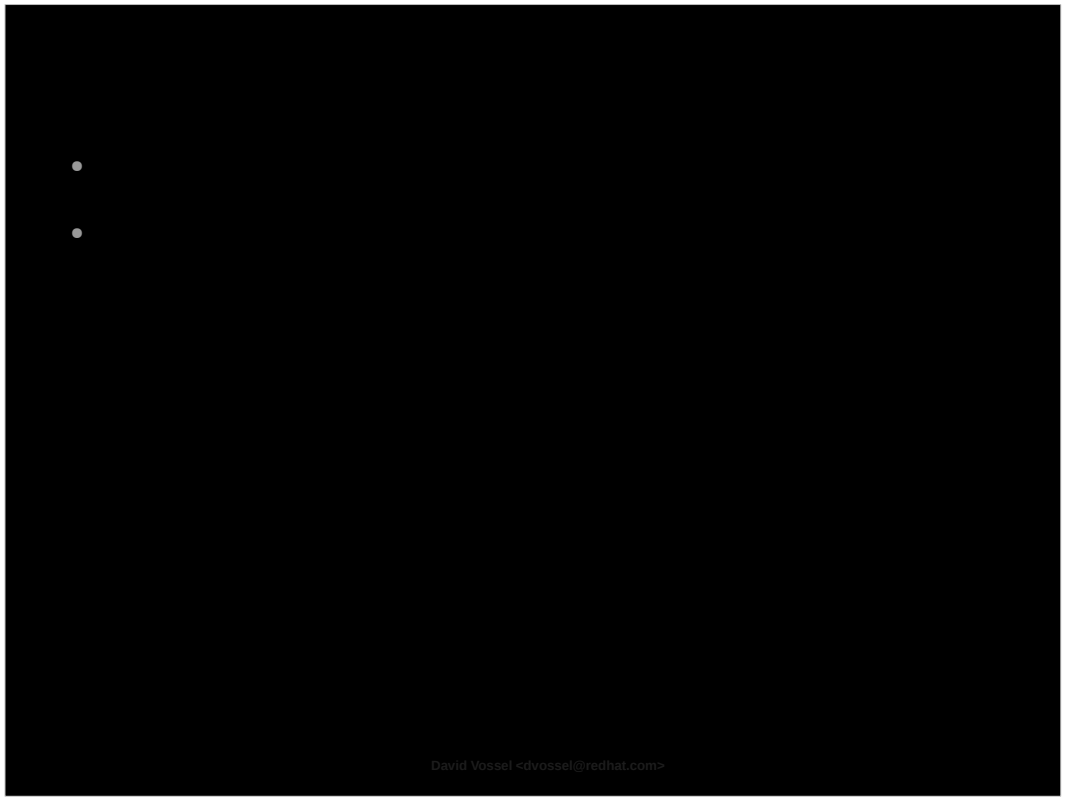
And

We get centralized management and recovery of **all the services in the cluster.**



There are a **couple of things to takeaway**

1. pacemaker and load balancing are not mutually exclusive.

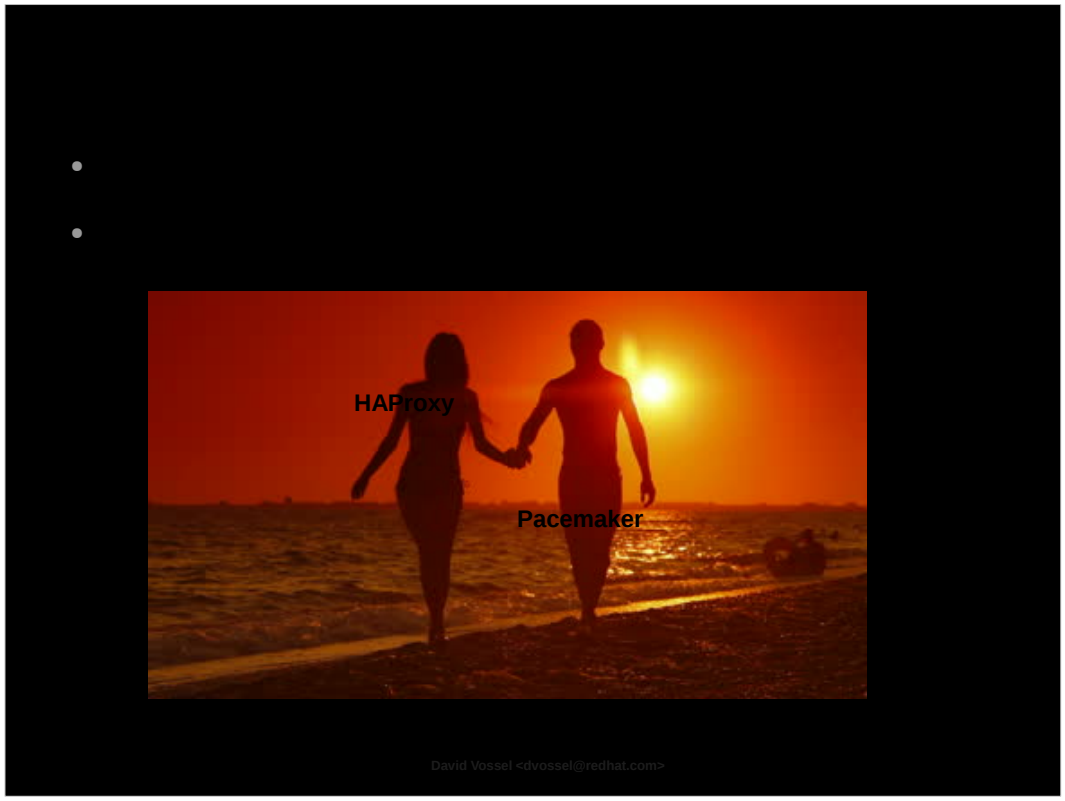


meant for each other.

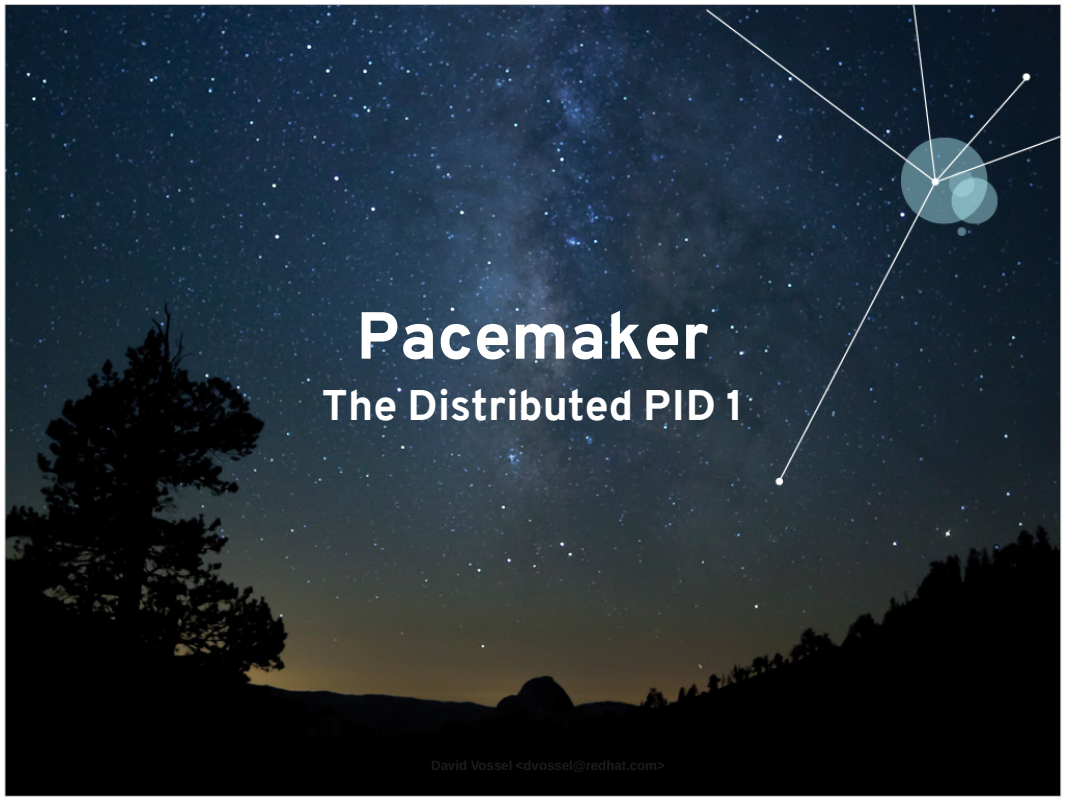
Pacemaker handles service management and recovery.

Load balancer handles routing traffic to active instances.

They work together to provide a holistic approach to HA.



David. You should drink some water or something before you pass out.



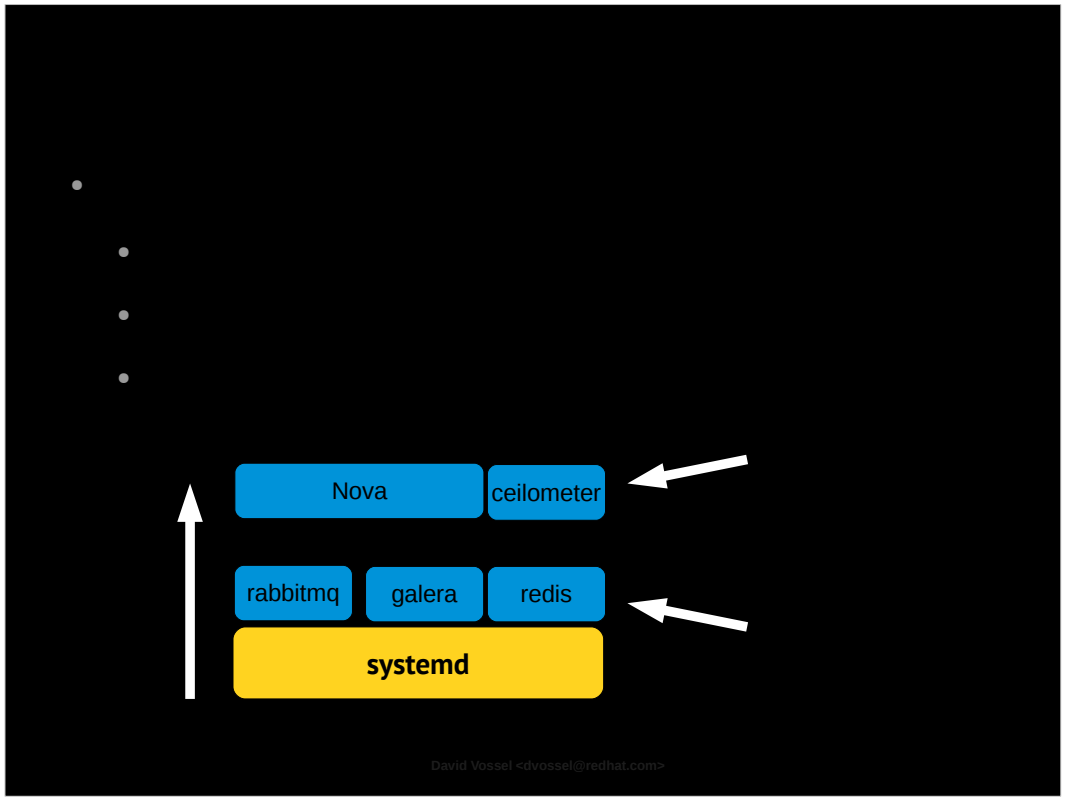
Title of presentation.

This is where I finally get to talk about the good stuff

understand why I told you that story to begin with.

Before we get into Opestack Specifics, I want to talk a little bit about what pacemaker is and how it works.

The analogy I'm using to describe pacemaker is it is a distributed PID 1... or a distributed init daemon.

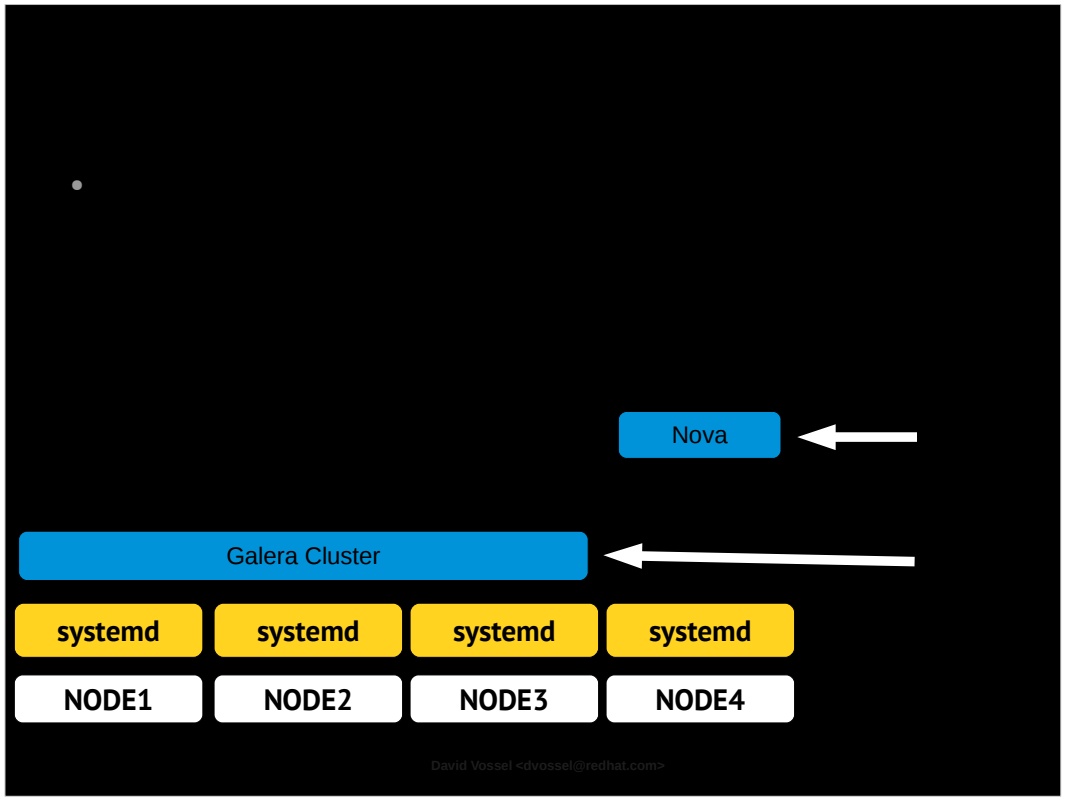


To understand what I mean by this. Lets take a look at what a modern PID 1's role.

So what are the fundamentals of what something like Systemd performs when acting as pid 1?

(look at slide and talk about points)

All really cool stuff, but there's a problem for openstack



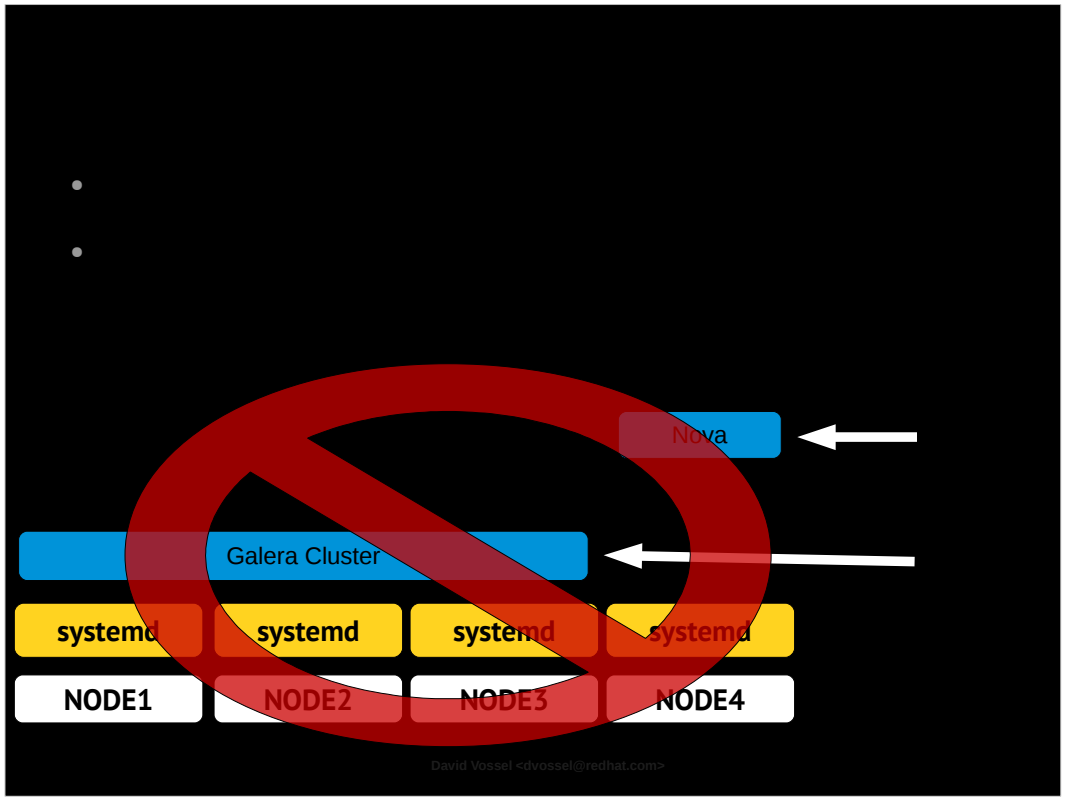
Openstack services are not isolated to a single machine

Openstack services are distributed.

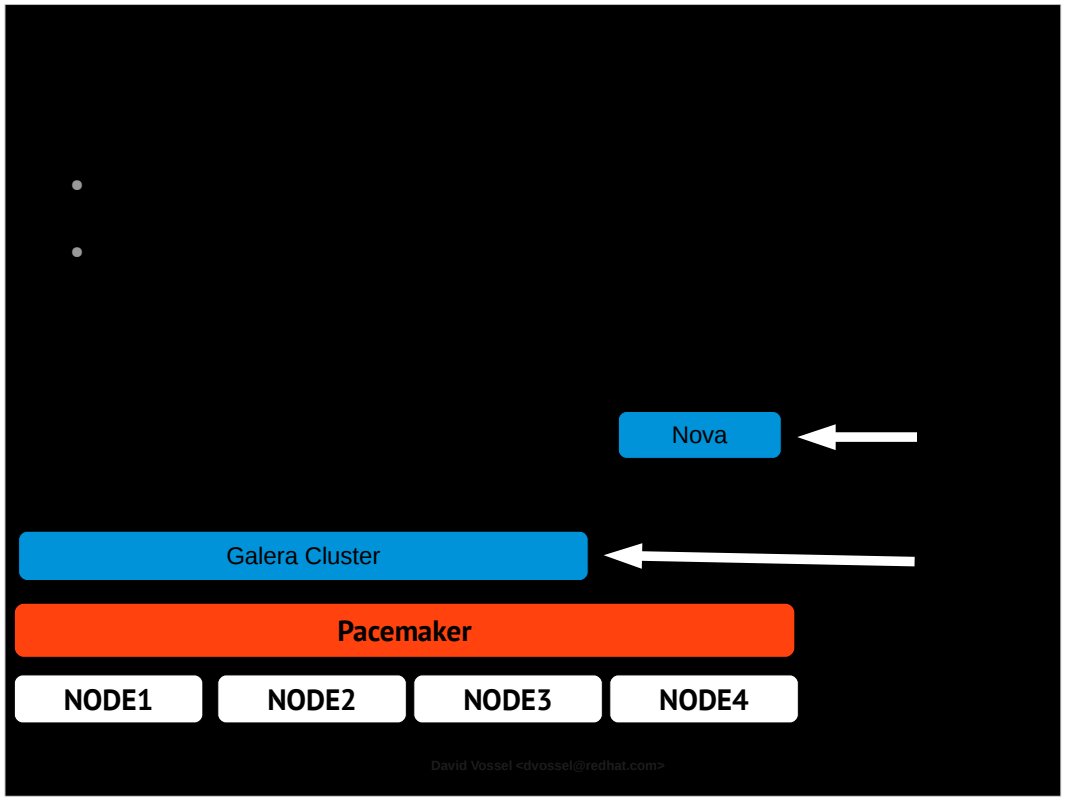
We really need a way to coordinate resource ordering across a distributed set of nodes.

(talk about slide)

Systemd is not distributed. Systemd can only coordinate services within a local machine.



Systemd. Can't coordinate order of distributed resources.



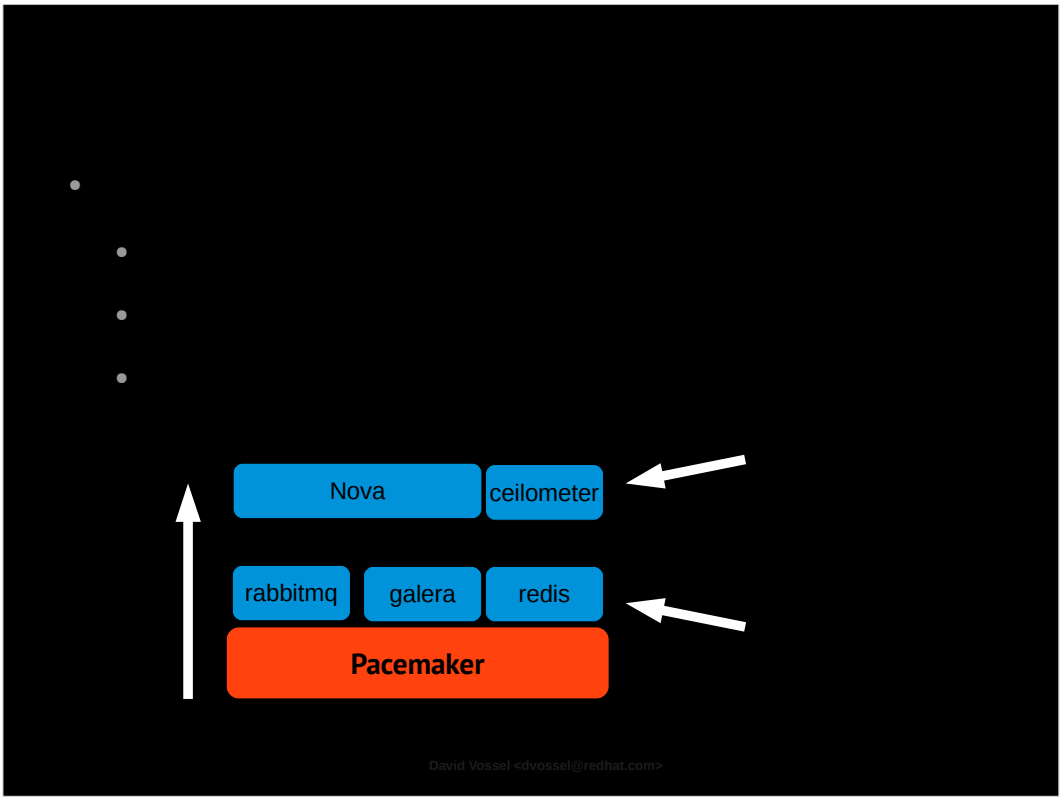
But pacemaker can.

Pacemaker was designed for this exact purpose.

We can say, start galera across 3 nodes.

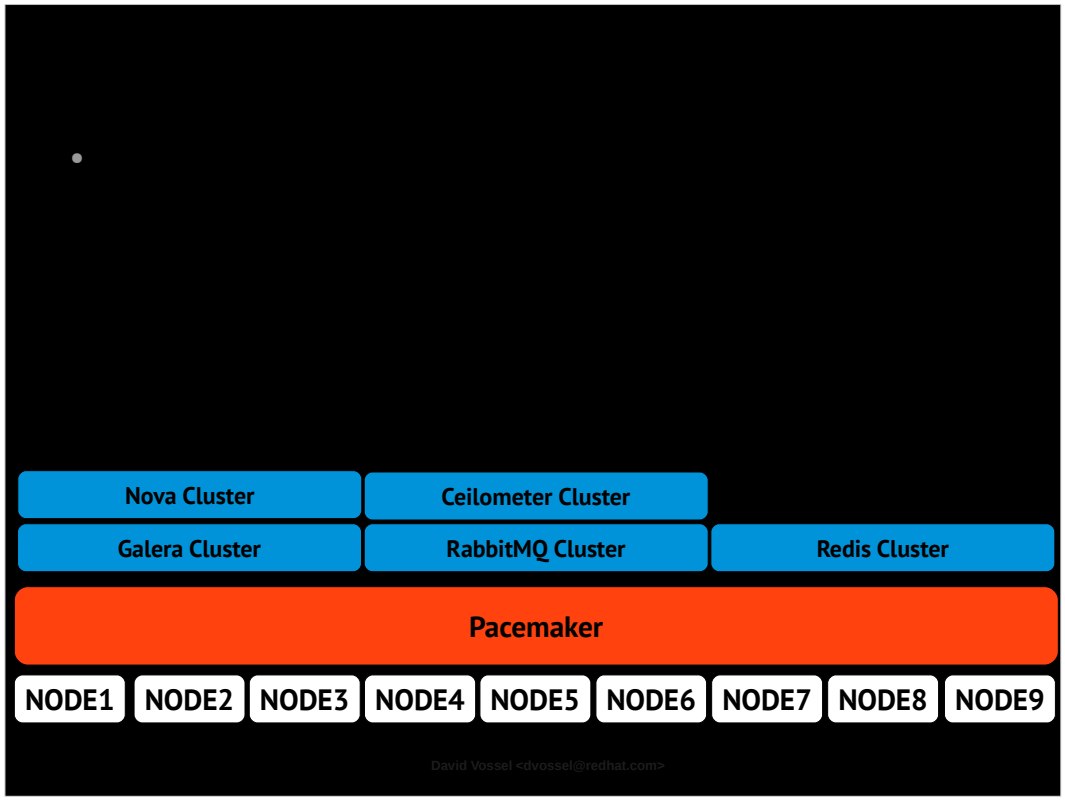
Then

Start Nova once galera cluster has formed.

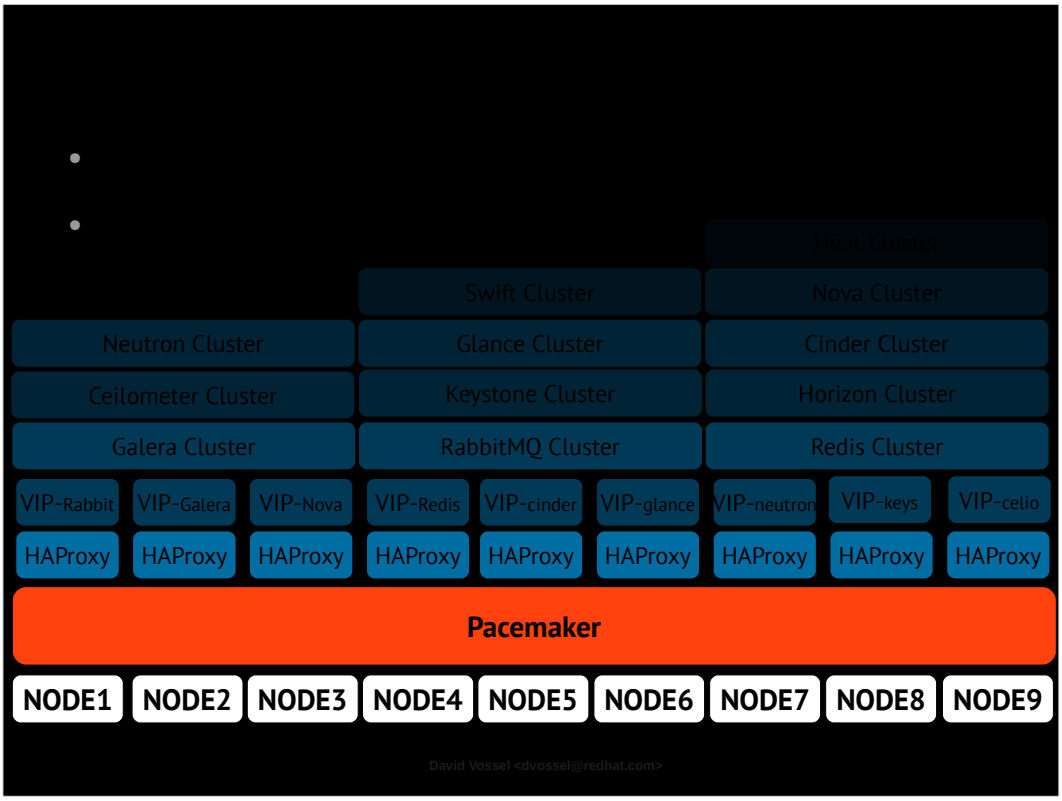


Go back and look at what a modern pid 1 does.

We can see pacemaker can do the exact same thing.



The big difference here is that pacemaker coordinate this across any number of nodes.

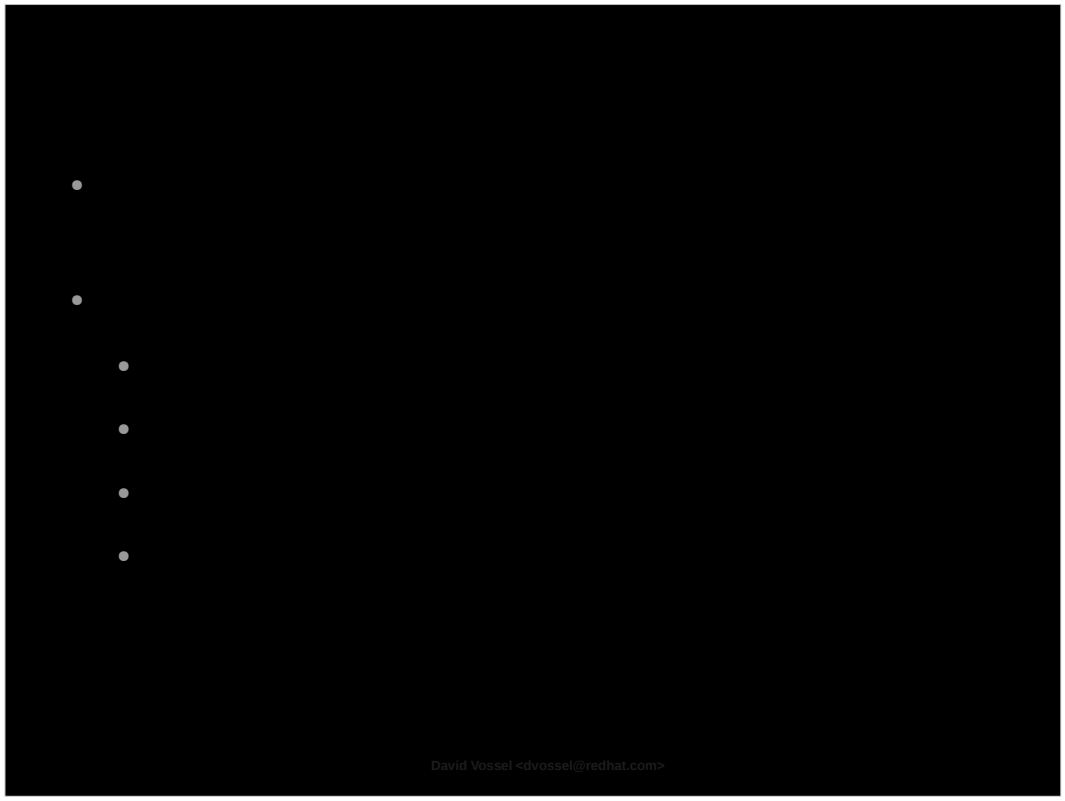


With any number of resources being involved.

Conceptually. Pacemaker is a distributed pid 1.

This is important to us because it allows us to coordinate how openstack services start in stop in a very flexible and powerful way.

So how does pacemaker do this?



Theres One important aspect of pacemaker's configuration Makes this possible.

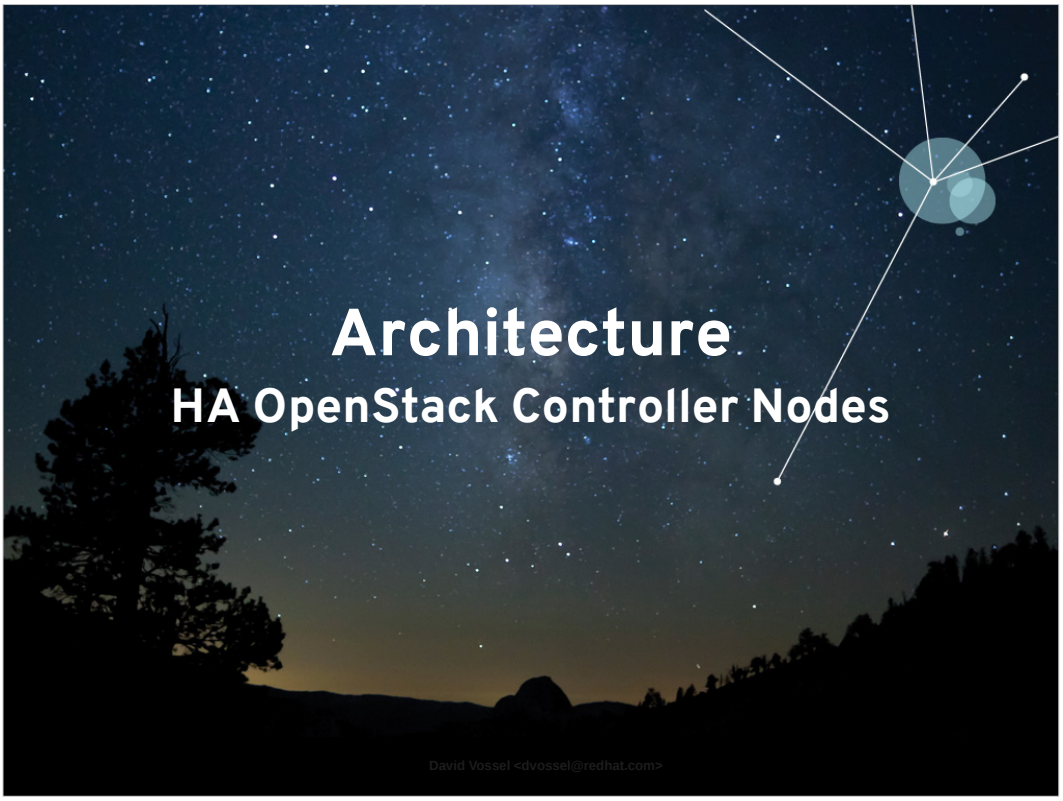
Resource constraints.

You tell pacemaker what services to run

Then define a set of relationships between services and nodes using constraints.

- order constraints
- colocation constraints

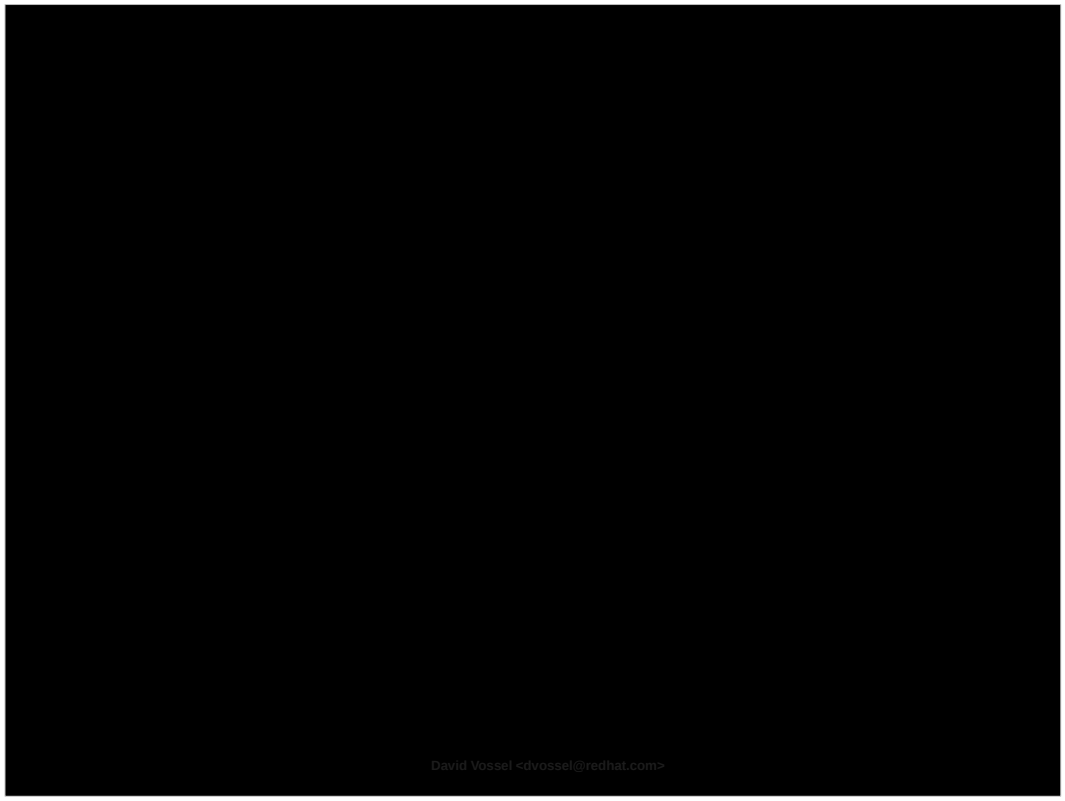
Refer to haproxy VIP for colocation.



On the topic of OpenStack services.

Lets dive into how we're leveraging the power of **Pacemaker to coordinate OpenStack Controller Node services.**

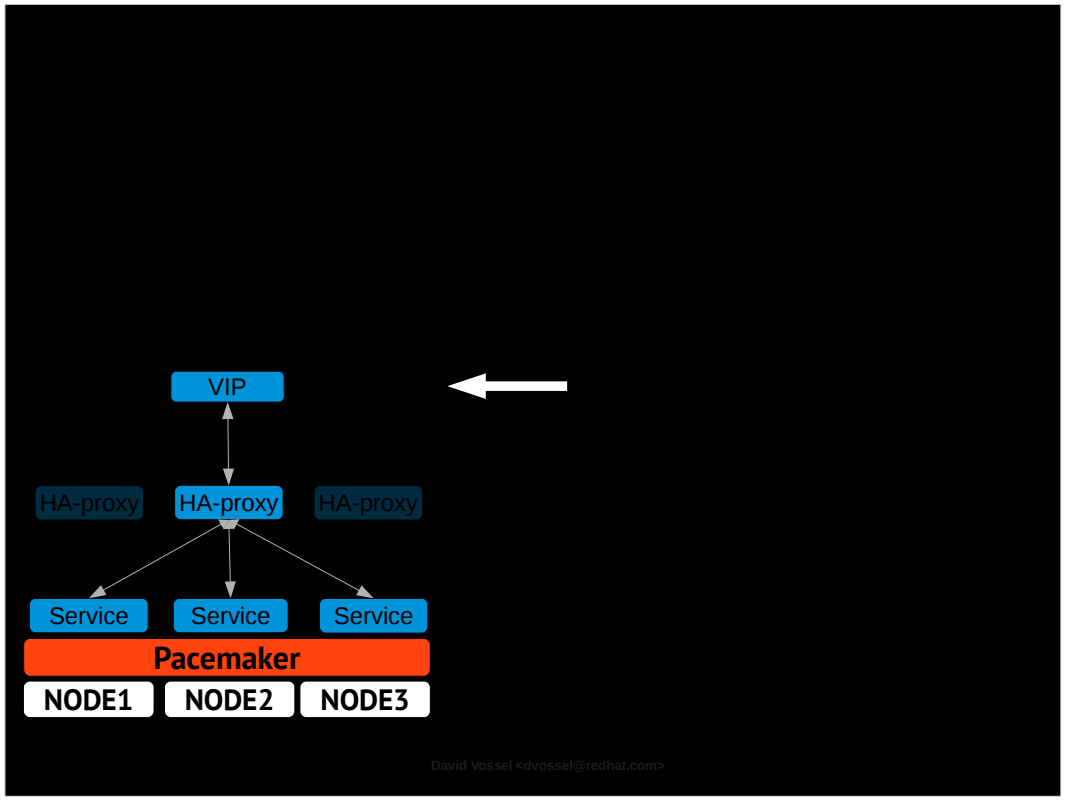
And make those service Highly Available.



break down how our OpenStack High Availability architecture works in one sentence.

It would look like this.

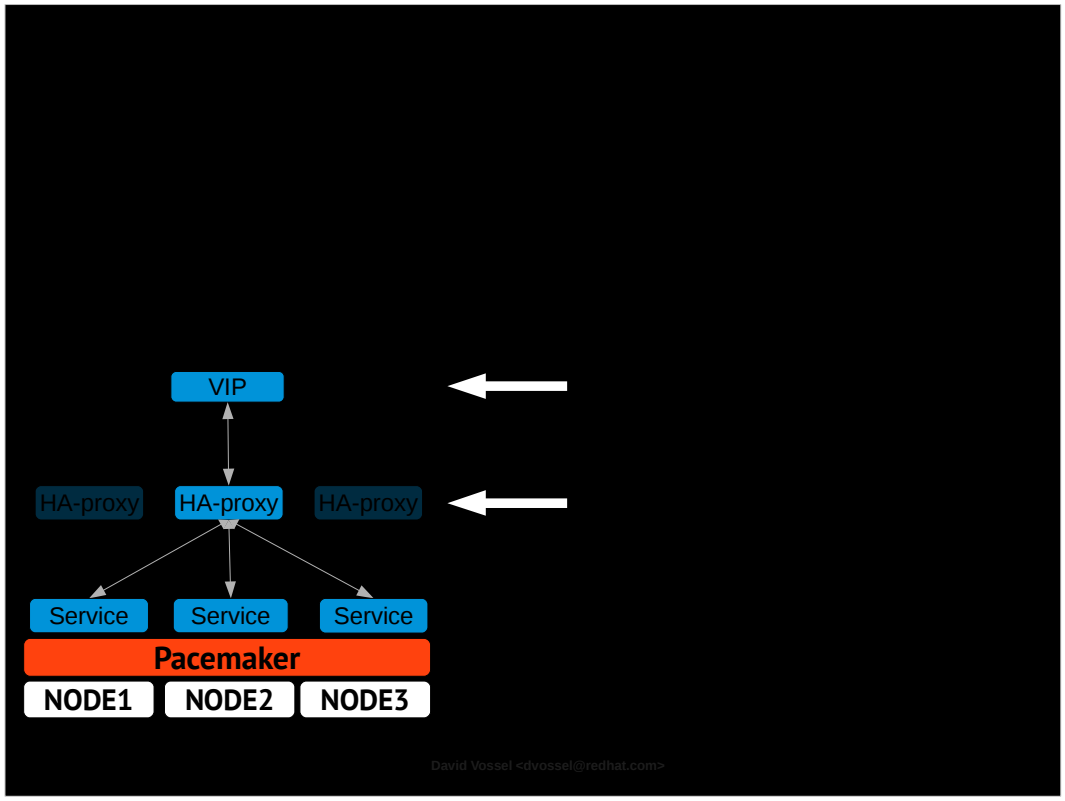
Pacemaker managing Virtual IPs + Load Balancers + Controller services to maximize the availability of OpenStack APIs



Pretty much every Openstack service is deployed with a variation of this pattern.

1. We have a Front End Virtual IP for each service

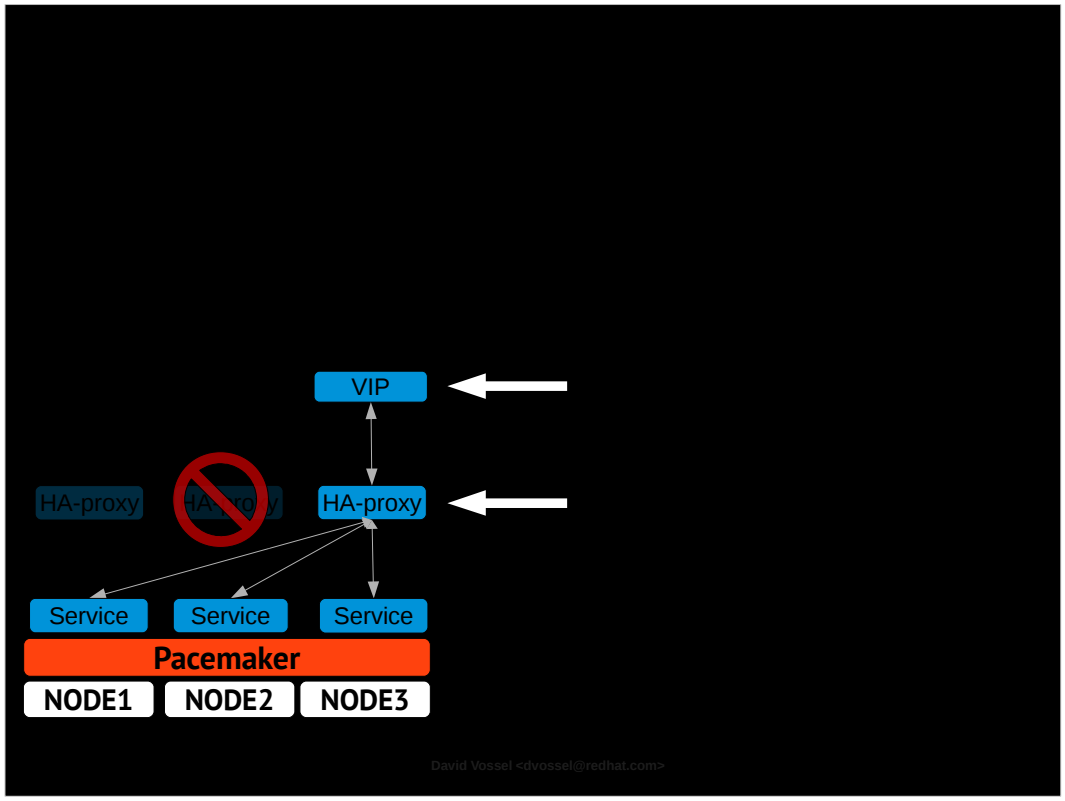
Anything trying to access a service's API talks to the VIP.



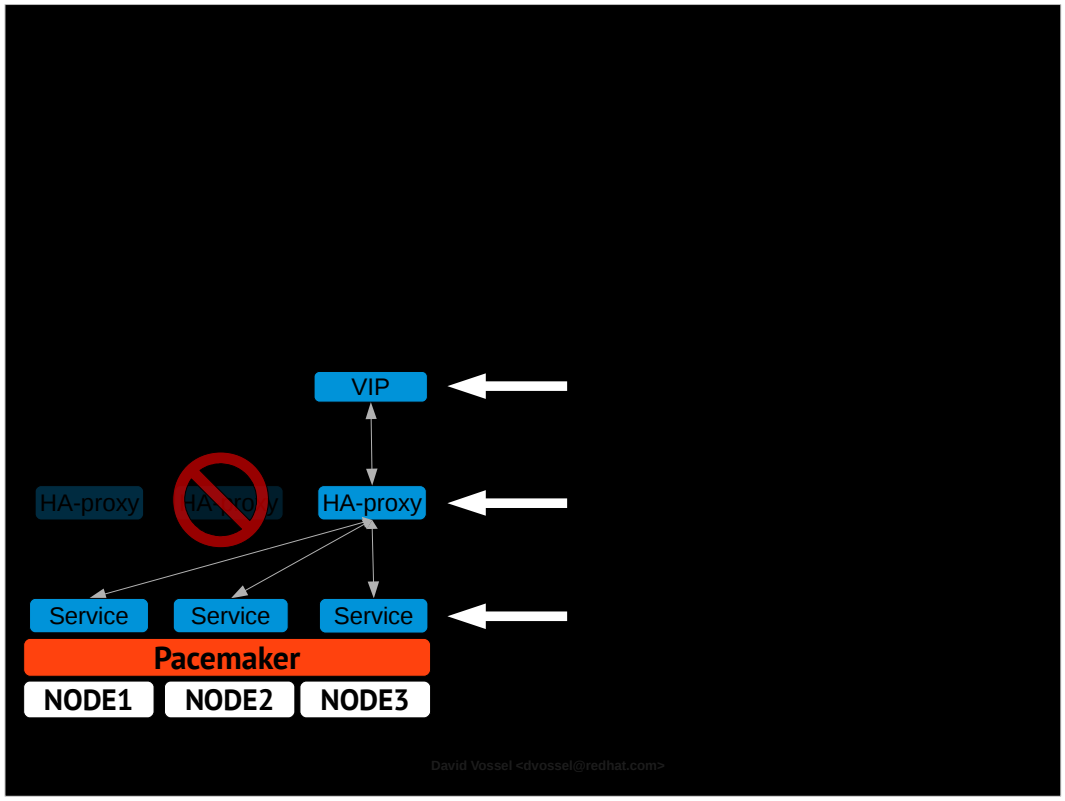
Each Virtual IP Address is paired with an active load balancer

A virtual IP address must be tied to an active load balancer instance.

Wherever HAProxy goes... (next slide.)



the Virtual IP goes.



And from there.

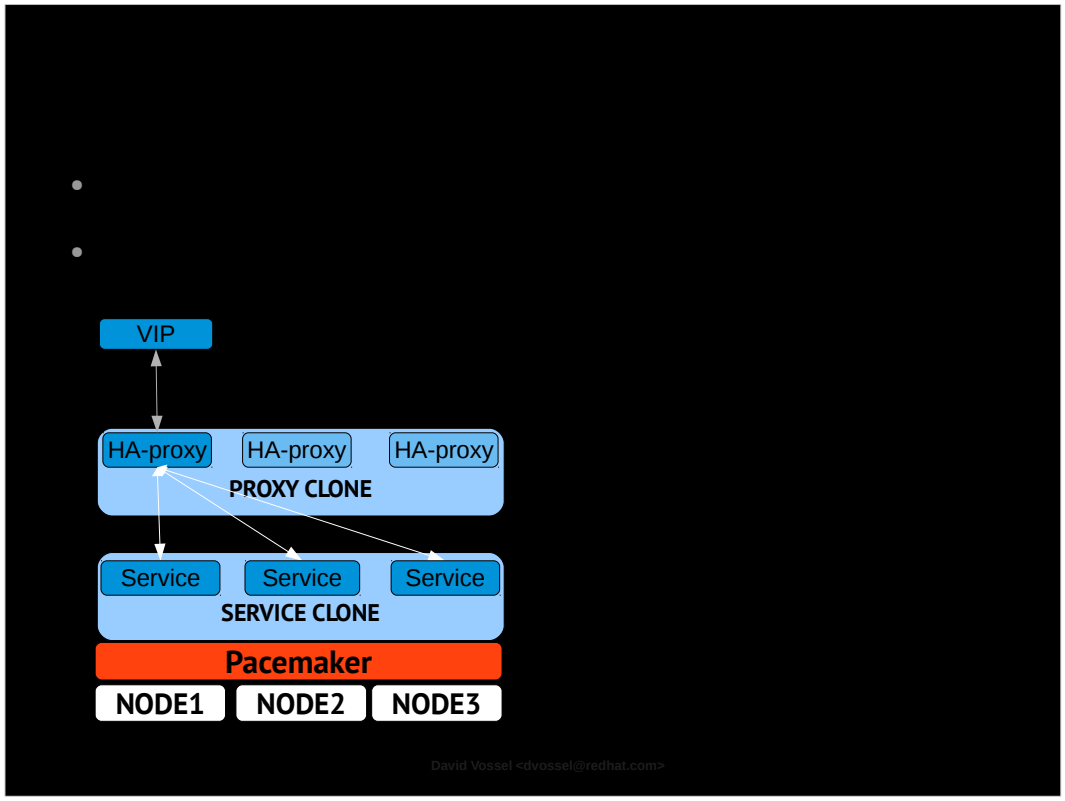
load balancer distributes traffic to our backend service cluster.

Most services are Active/Active

Some services are still Active/Passive.. but we're working on that.

These services are things like Nova, Glance, Keys..

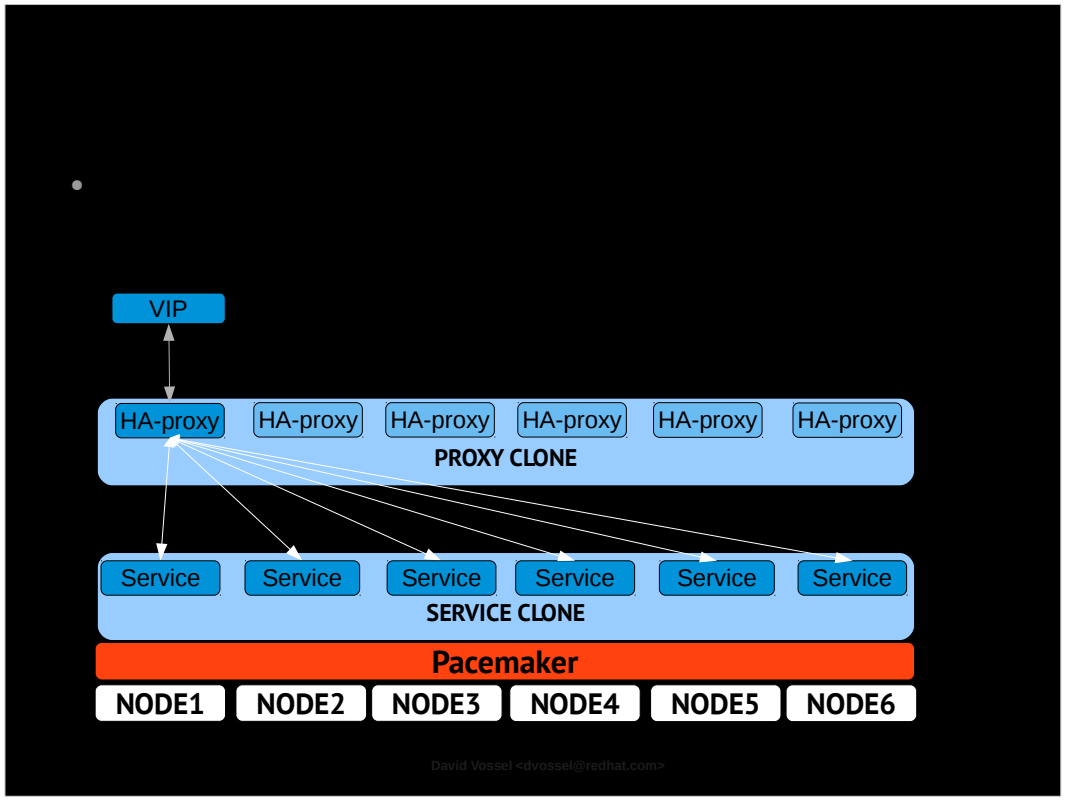
Remember the story from earlier? Look familiar???



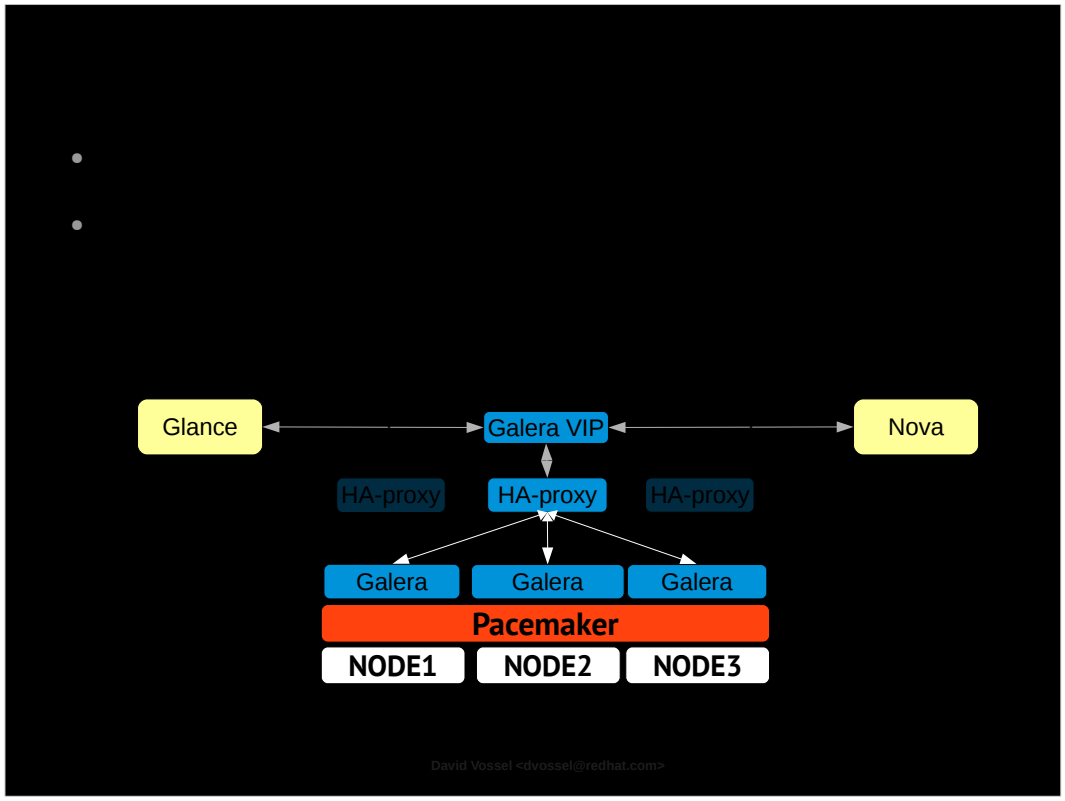
One thing I want to point out here is our use of pacemaker's Cloned resource type for the backend Controller Node services

With clones, we are telling pacemaker to replicate an identical service across multiple nodes.

Powerful for scaling from pacemaker;s perspective.



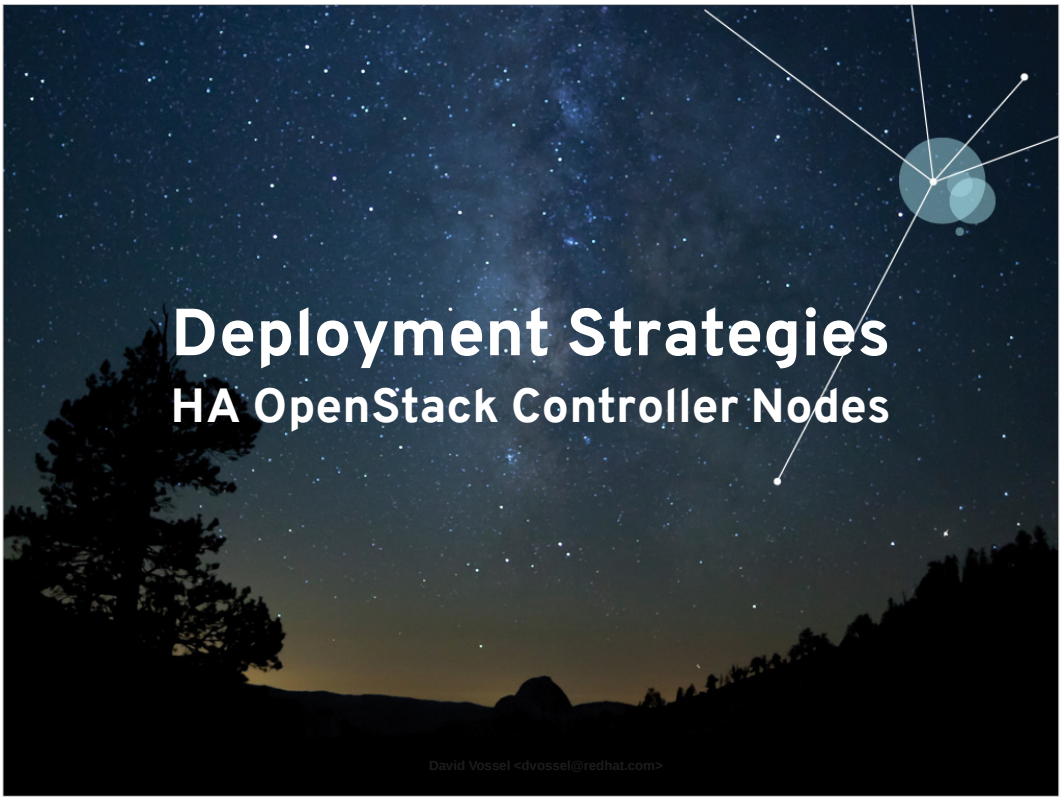
If we want more instances of a particular service, we just increment the number of cloned services pacemaker is allowed to run for that service.



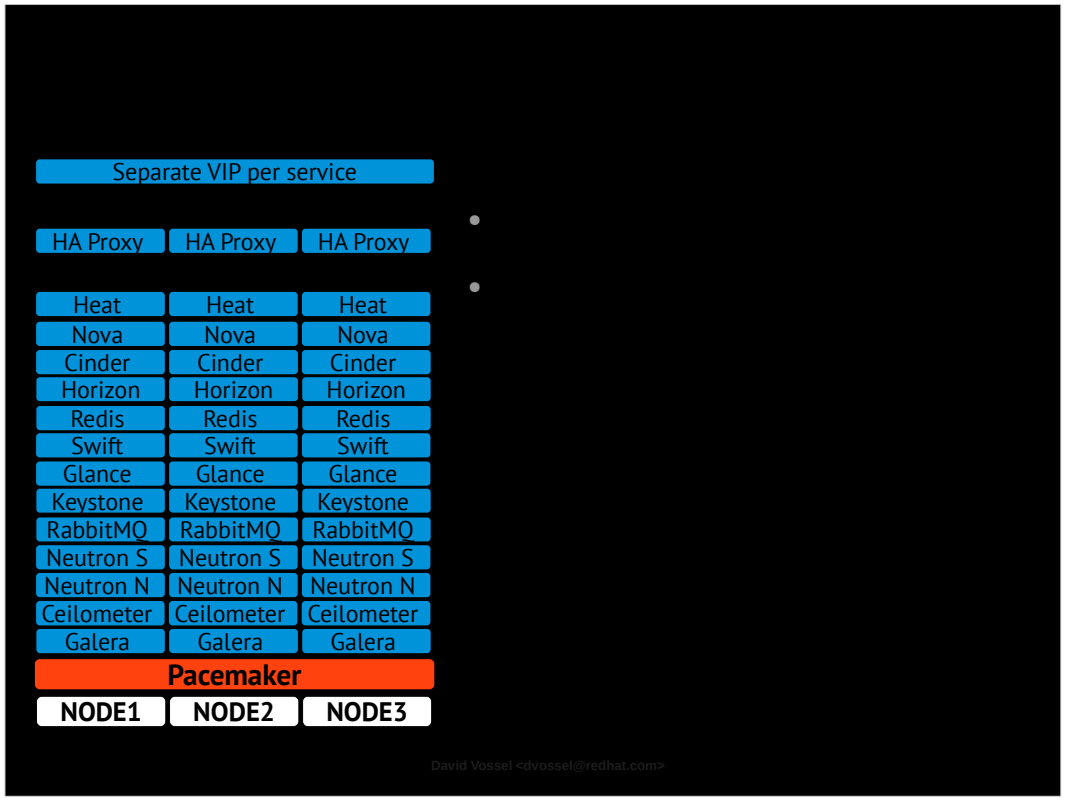
So, going back to how this all works.

OpenStack controller node services interact with one another's APIs using the front end VIP pacemaker is managing.

Look at slide.



So lets look at some deployment strategies for this pattern.

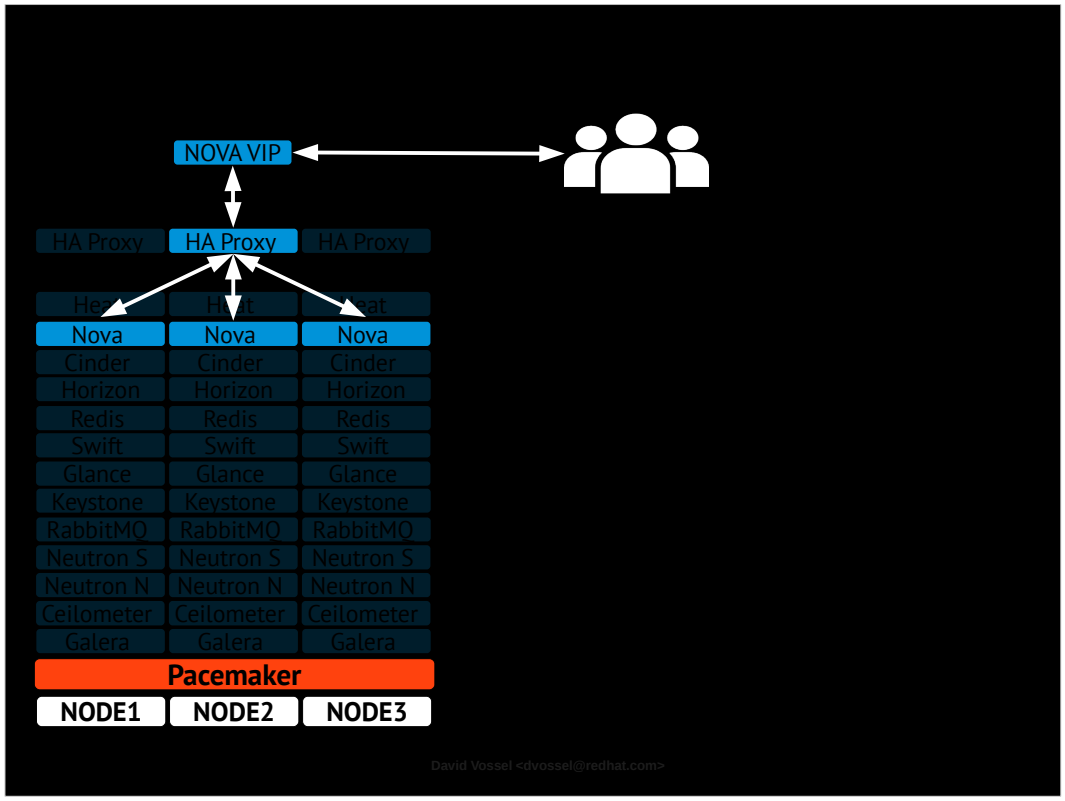


Collapsed.

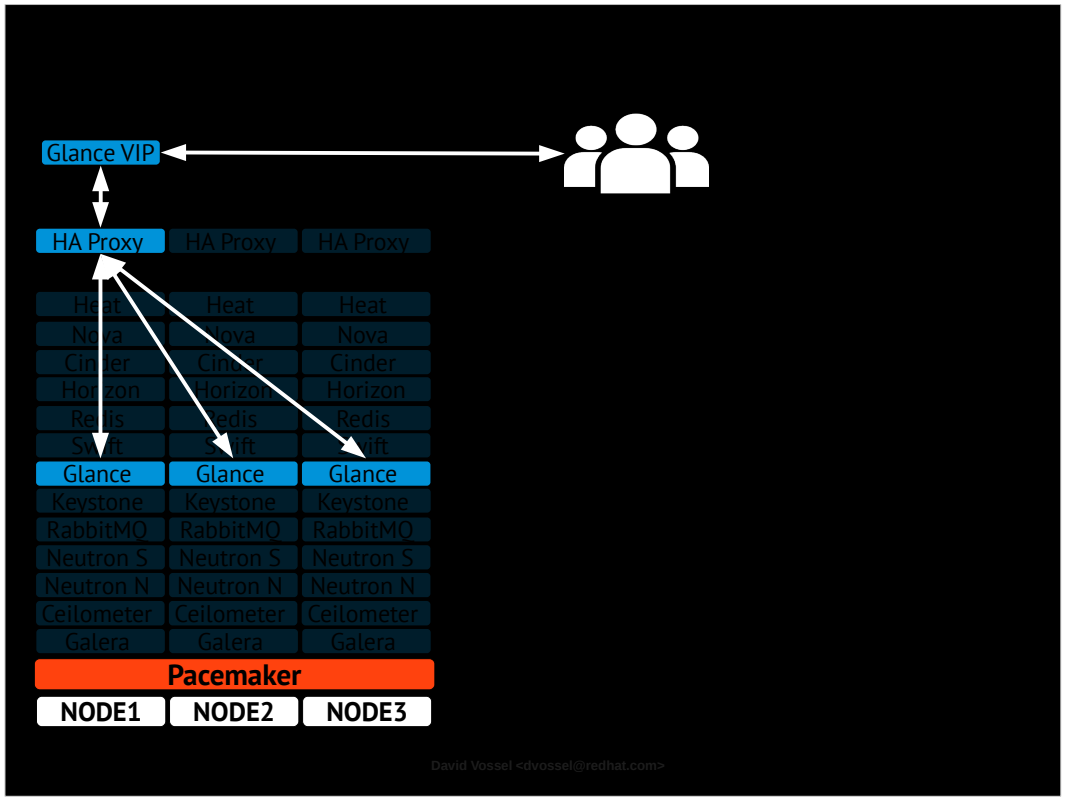
All controller nodes are identical to one another.

Requires powerful hardware... but is easy to deploy.

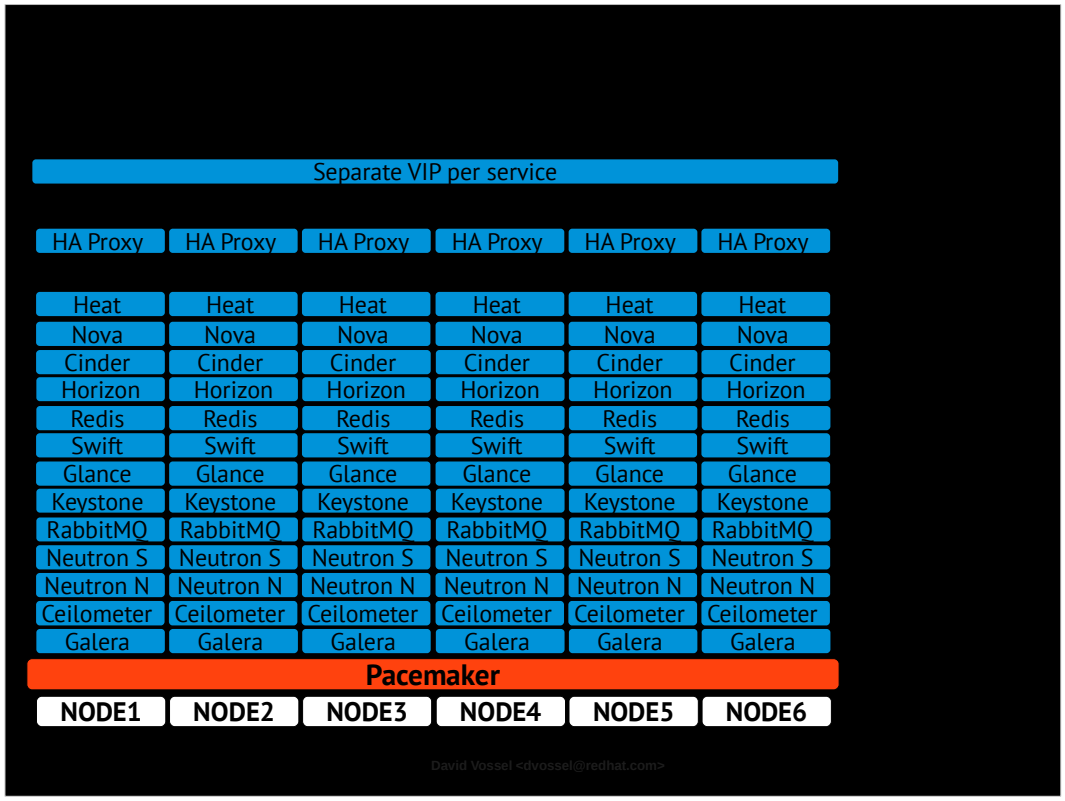
Easier to deploy because everything is a clone.



We can see our pattern all throughout the collapsed architecture



Here's another example of the pattern.

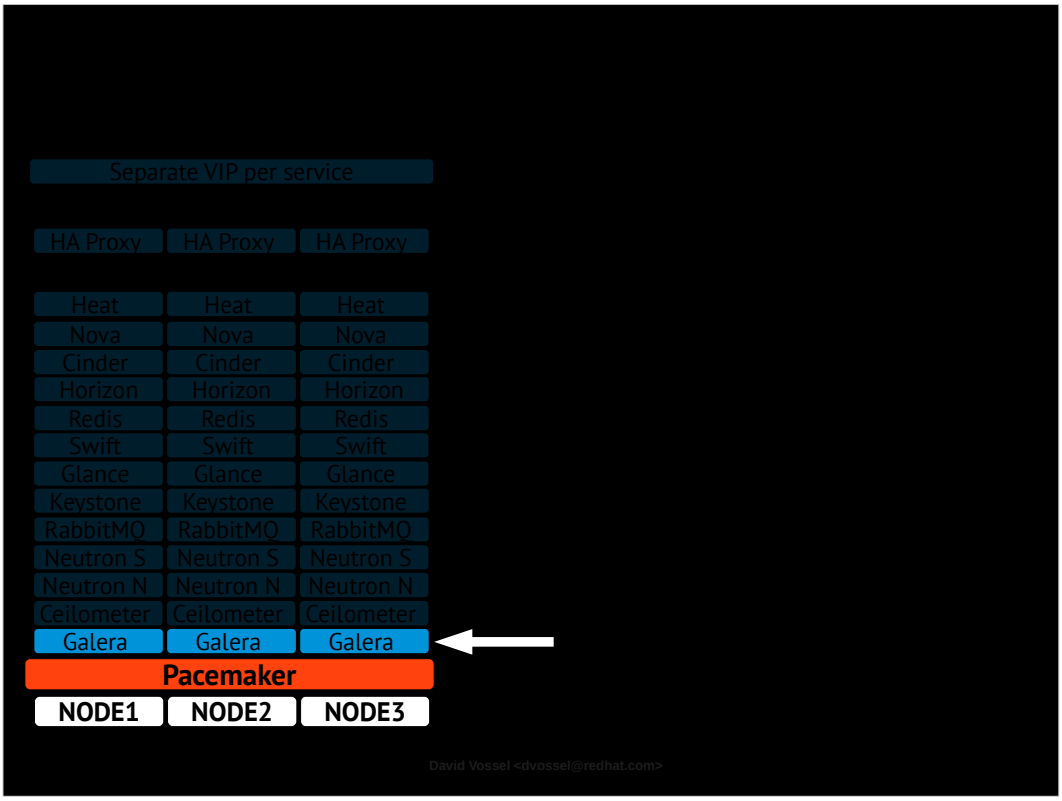


Scaling the collapsed architecture approach is simple.

Every node is identical,

To add more OpenStack controller node capacity, you just add more nodes.

Pacemaker knows all the nodes run an identical services, so its easy for us to integrate new nodes.



Remember what pacemaker is giving us here.

1. reliable load balancing through managing both the HAProxy instances and the VIP.

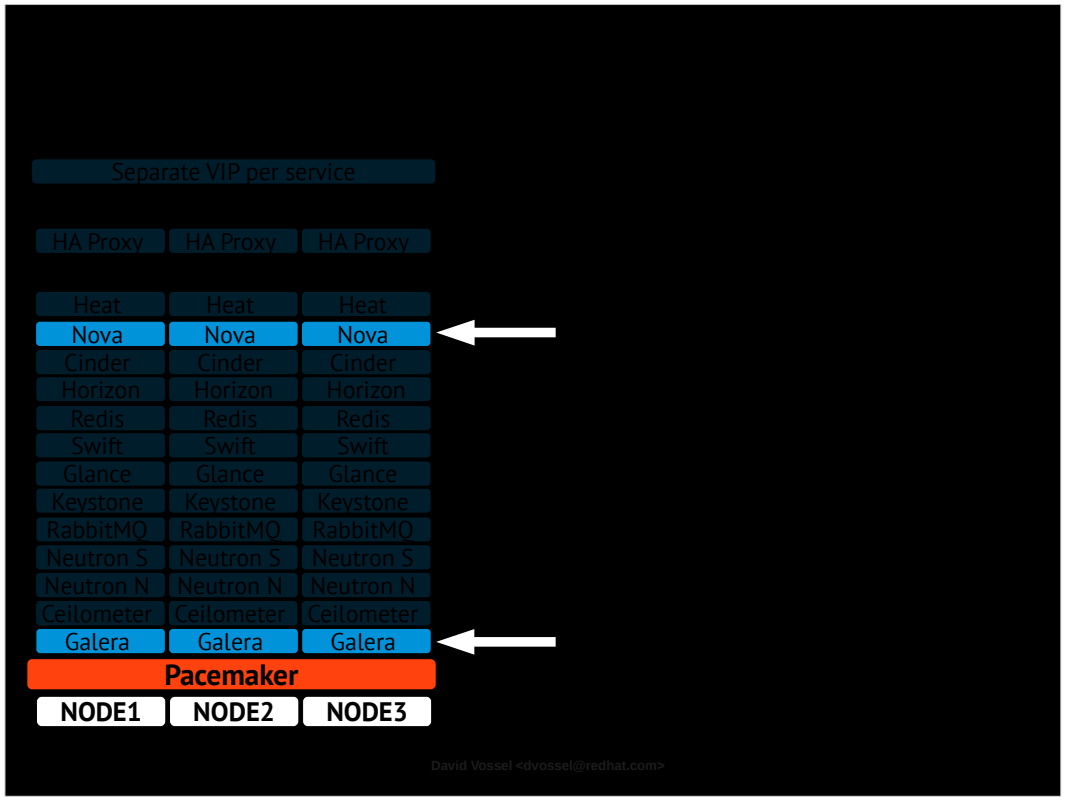
2. cluster wide startup ordering.

We can express complex dependencies that spread across multiple nodes.

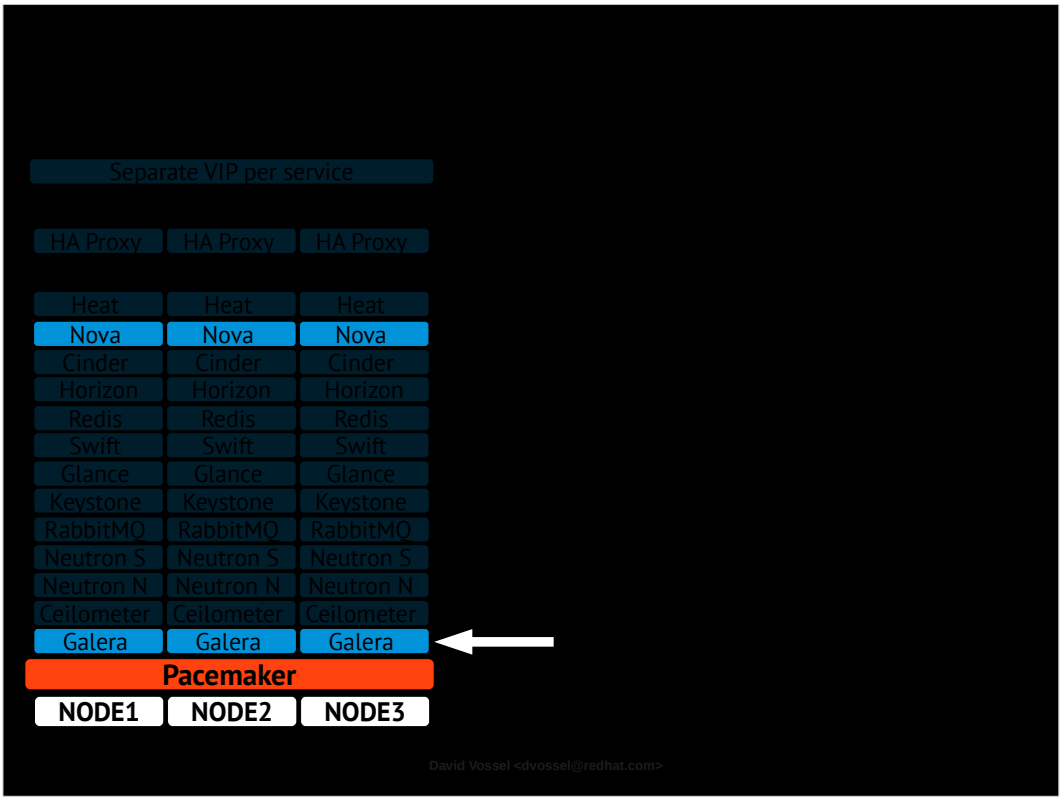
Give detail on one.

bootstrap and start the Galera cluster.

Not trivial

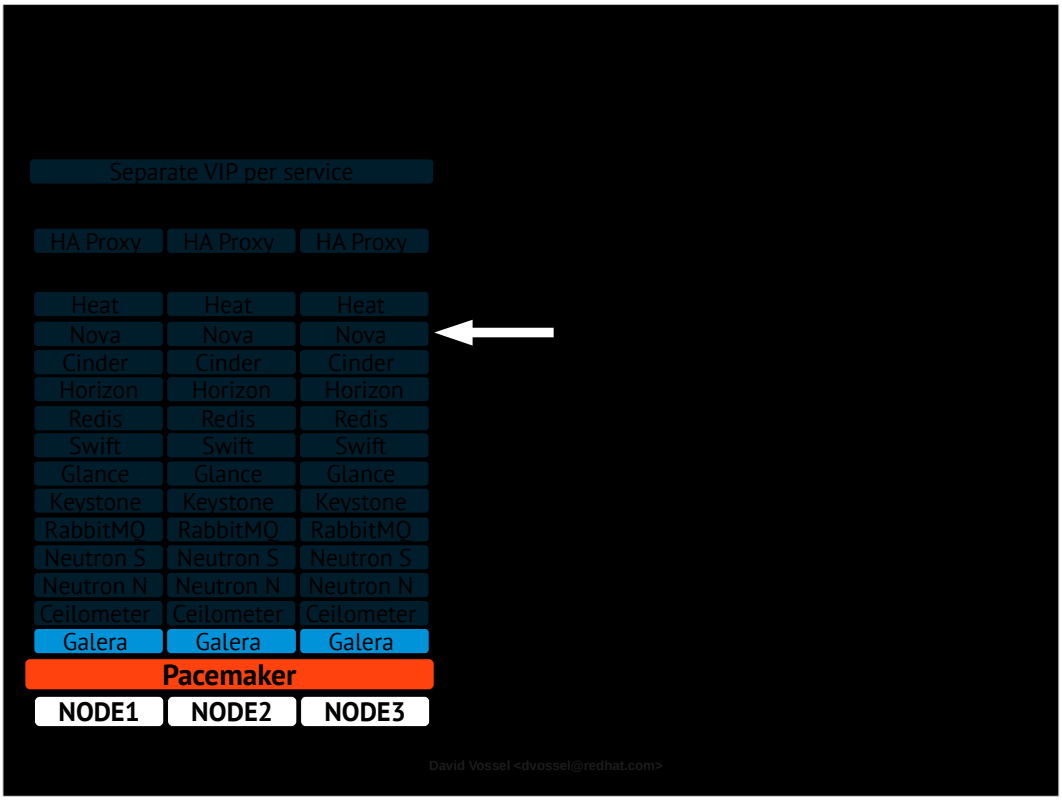


Now once that is complete allow the Nova instances to start.

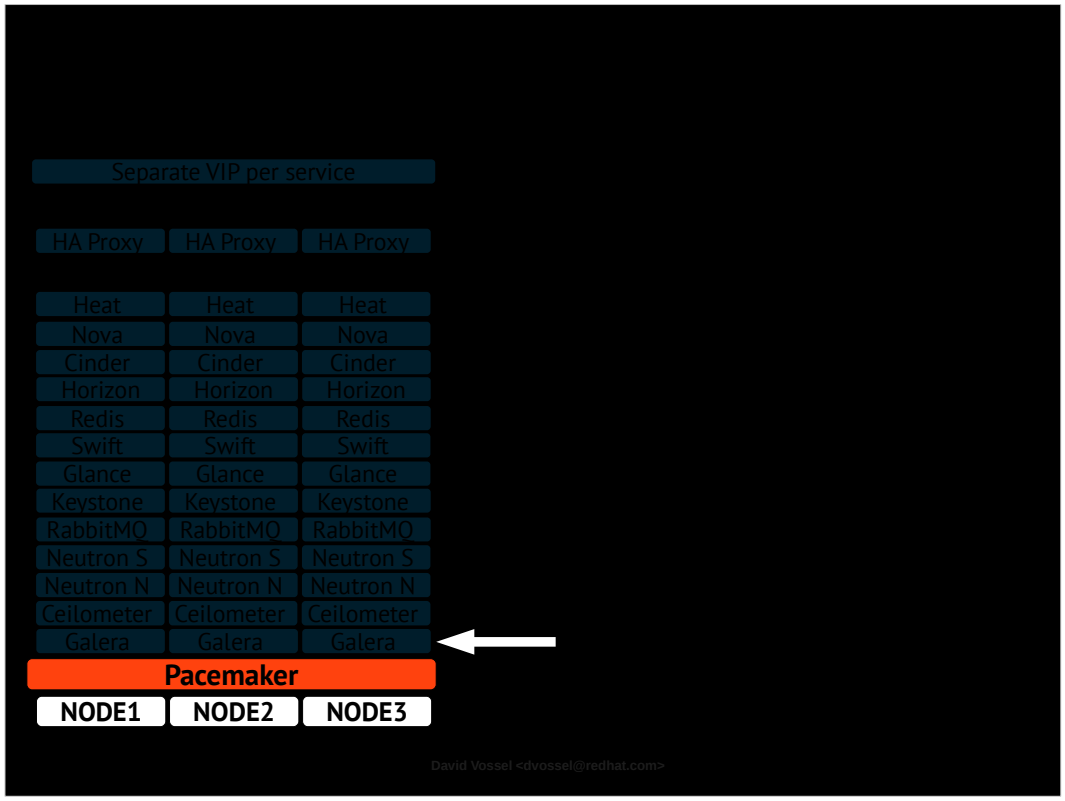


Likewise, pacemaker enforces cluster wide stop ordering.

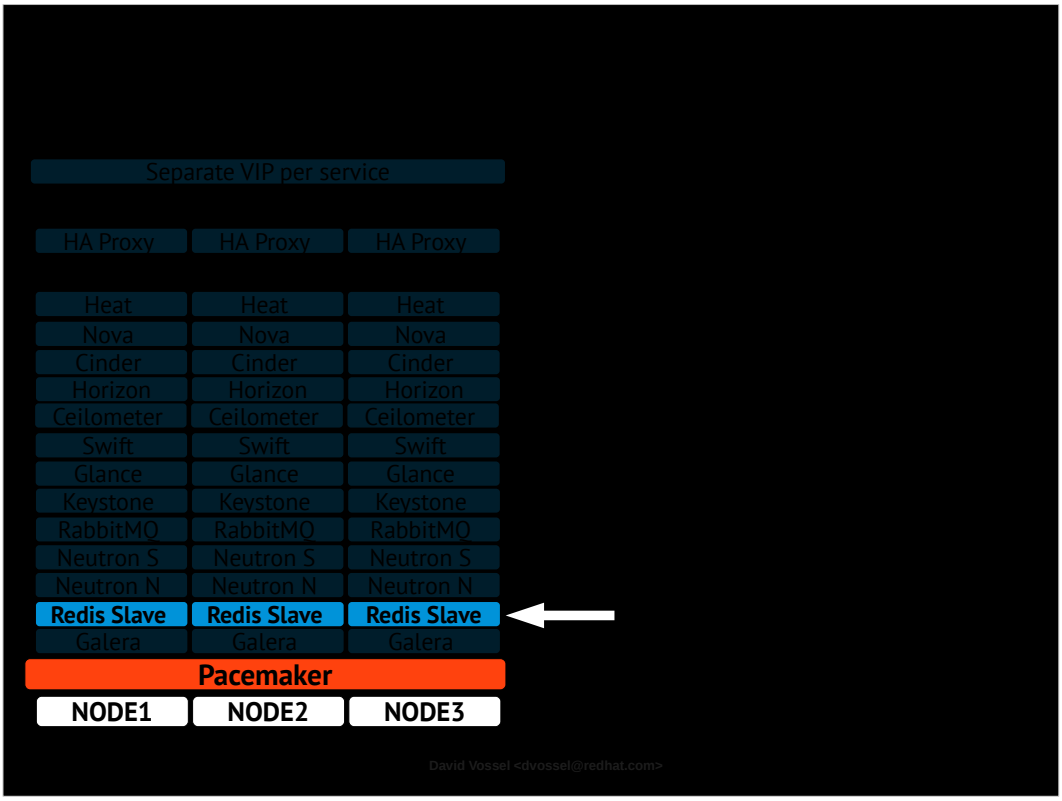
Say for instance we wanted to shutdown galera to do an upgrade.



Pacemaker knows to first shutdown everything in the cluster that depends on galera.



Then, galera can be gracefully shutdown.



Remember what pacemaker is giving us here.

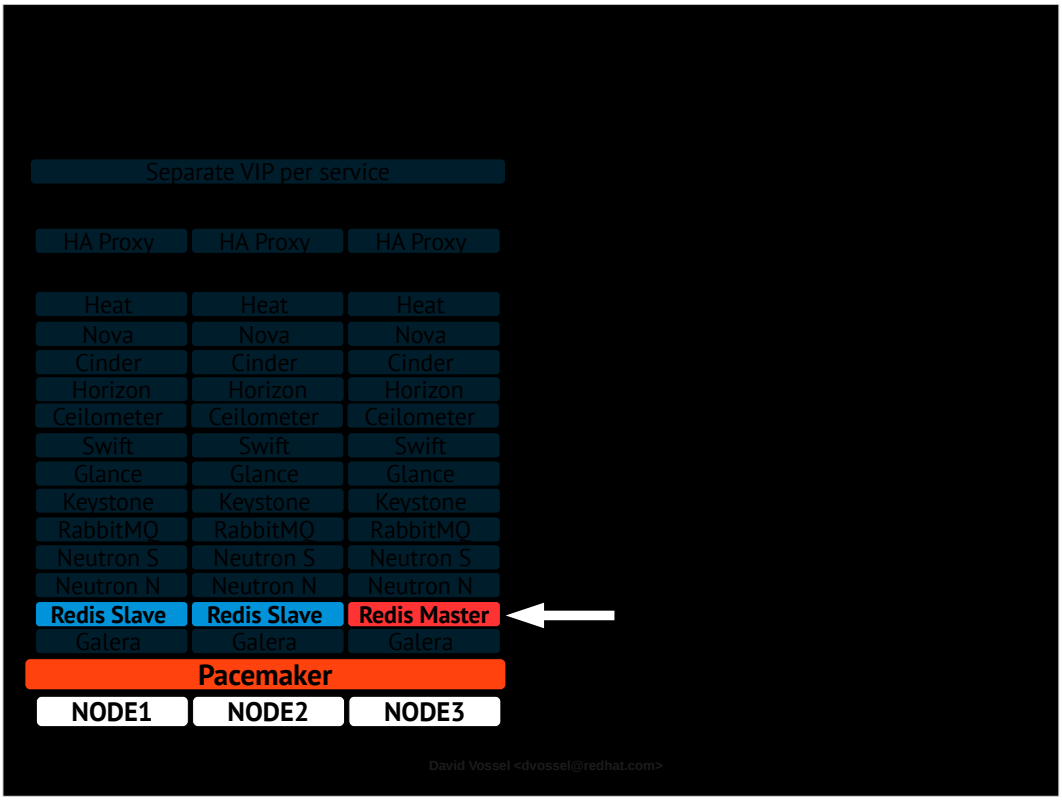
1. reliable load balancing through managing both the HAProxy instances and the VIP.

2. cluster wide startup ordering.

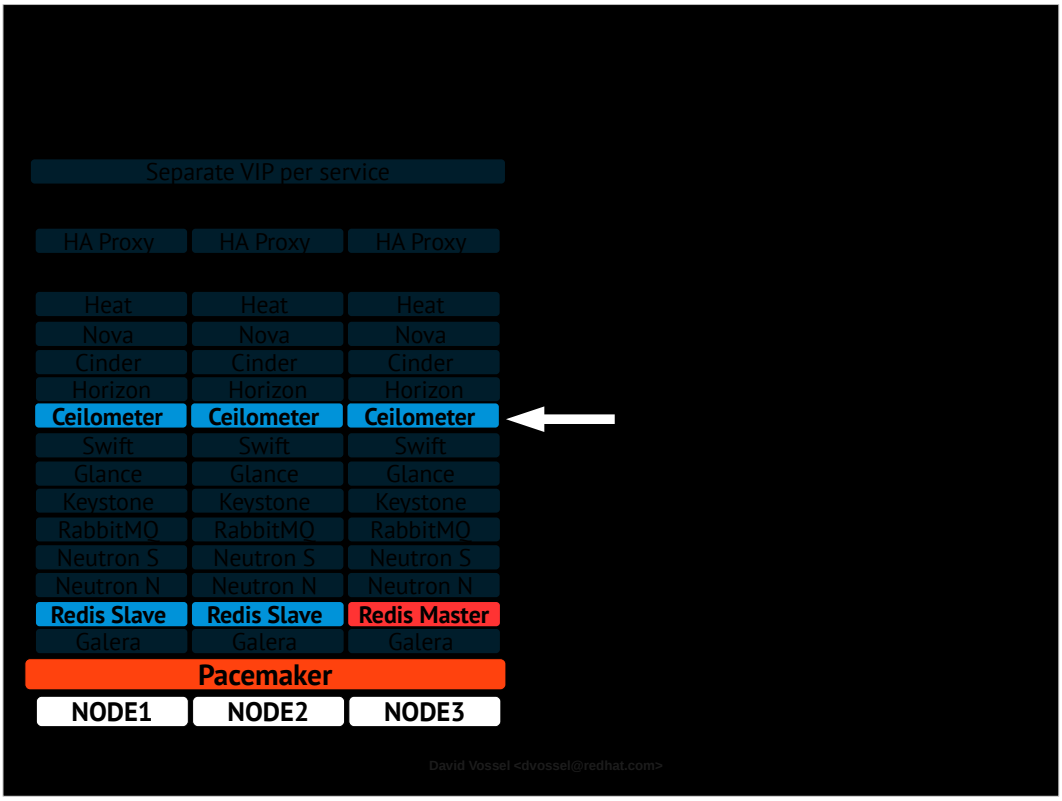
We can express complex dependencies that spread across multiple nodes.

Like bootstrap and start the Galera cluster.

Next slide...



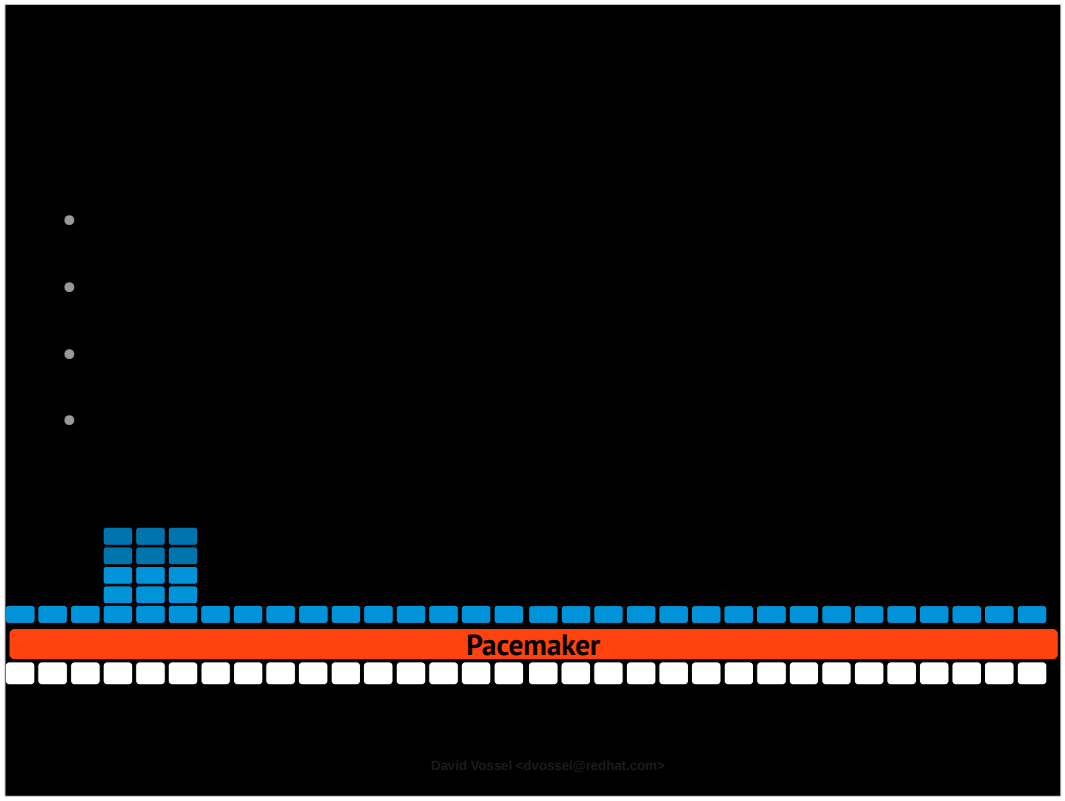
Here's another example of a complex ordering



Once master instance is up.

Allow ceilometer to start.

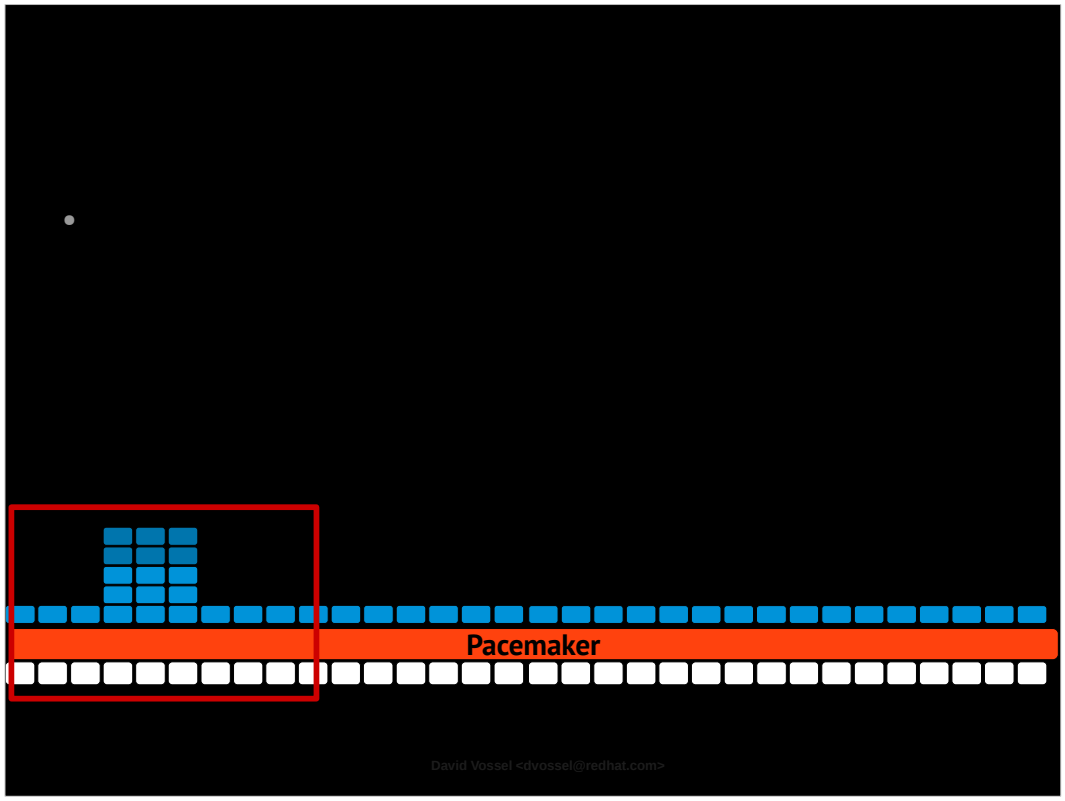
... pause... about to go to segregated arch.



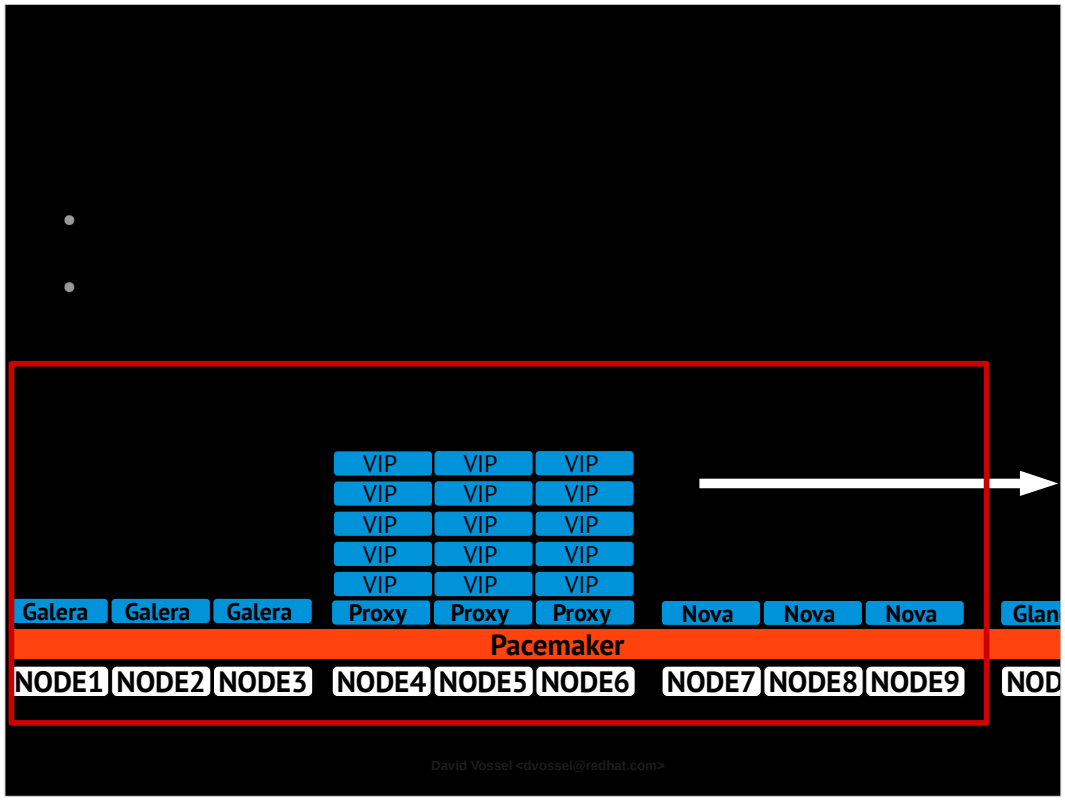
Our second deployment
segregated approach.

Each service get's it's own dedicated hardware.

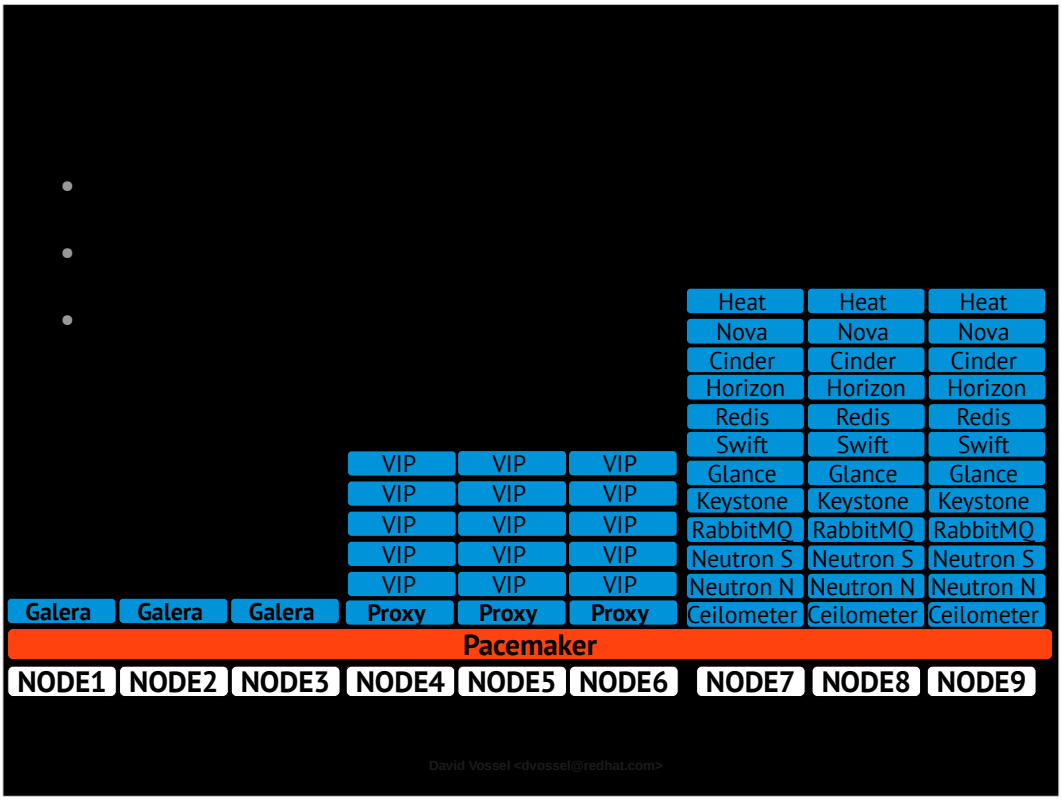
TALK ABOUT SLIDE.



Look a little closer at what's going on here.



Talk about slide

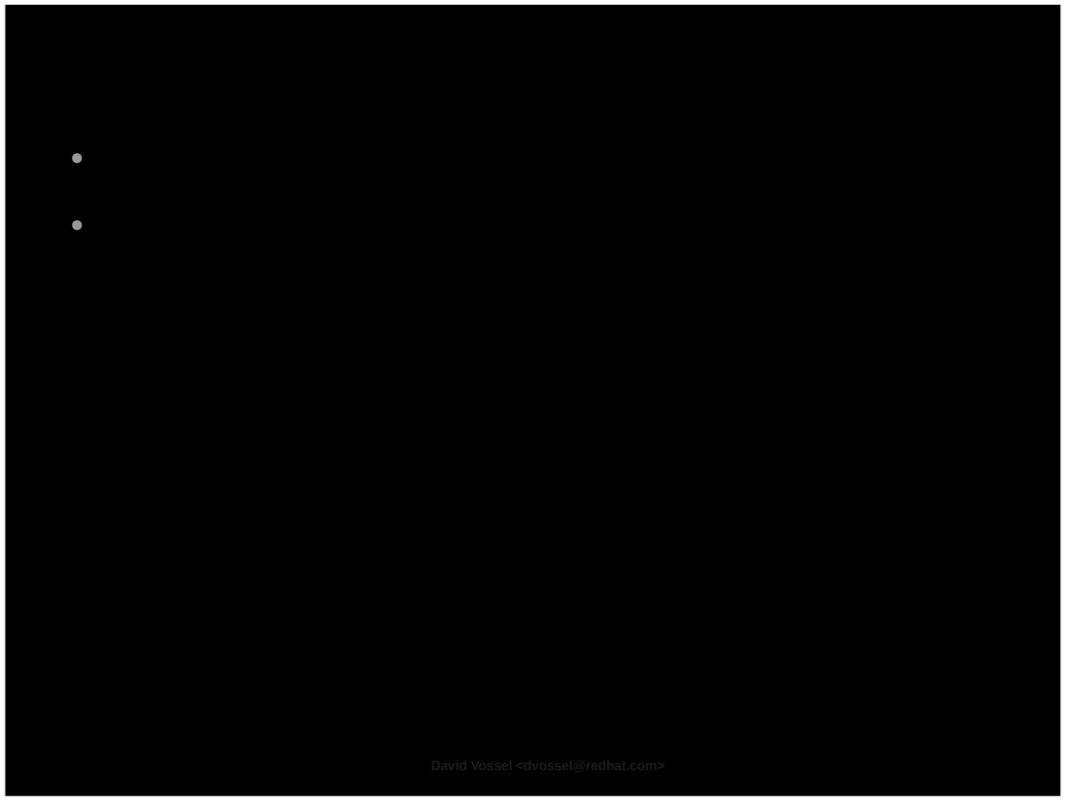


Mixed usage of both collapsed and segregated approaches are possible too.

Every service can be isolated to its own set of nodes.

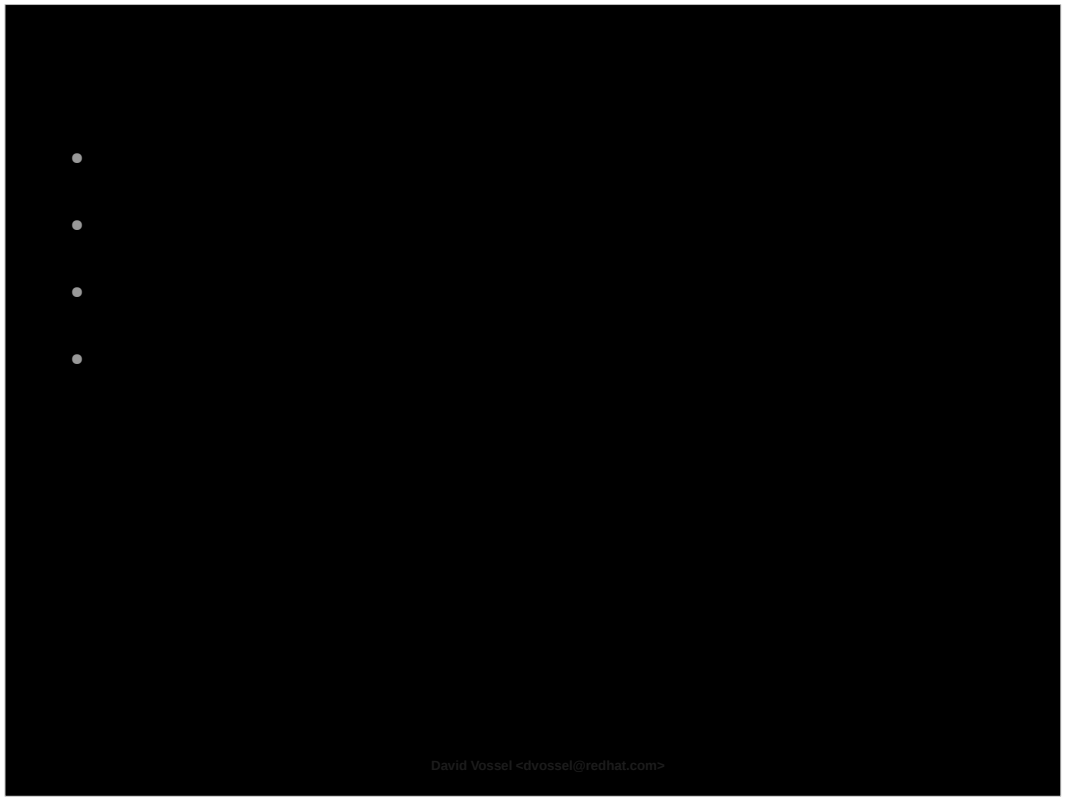
Or combined with other services.

Talk about slide....



Lets review some of the advantages of pacemaker managing controller services.

Advantages are on top of pacemaker's solid quorum and fencing features.

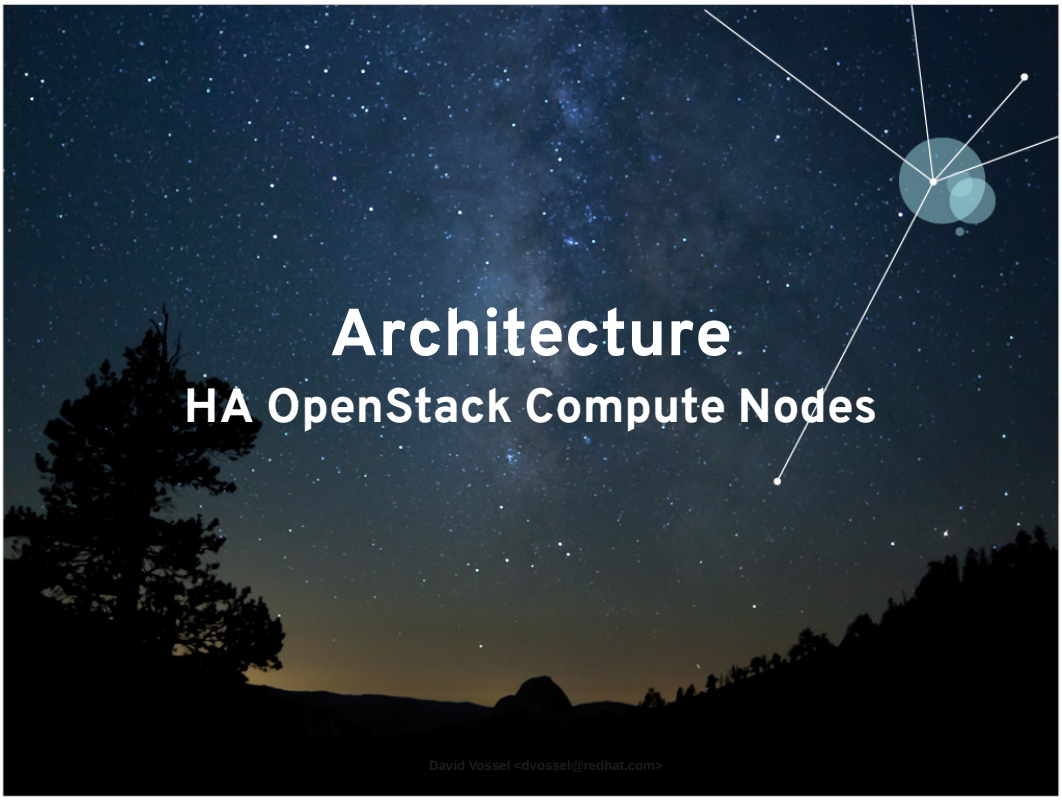


Graceful distributed start and stop ordering.
- shut down a single service and all it's dependencies
- shutdown openstack services on a single node

- which should make server maintenance and software upgrades easier.

Dynamically add capacity.. cloned resources help.

Resource ordering remains the same despite number of nodes.



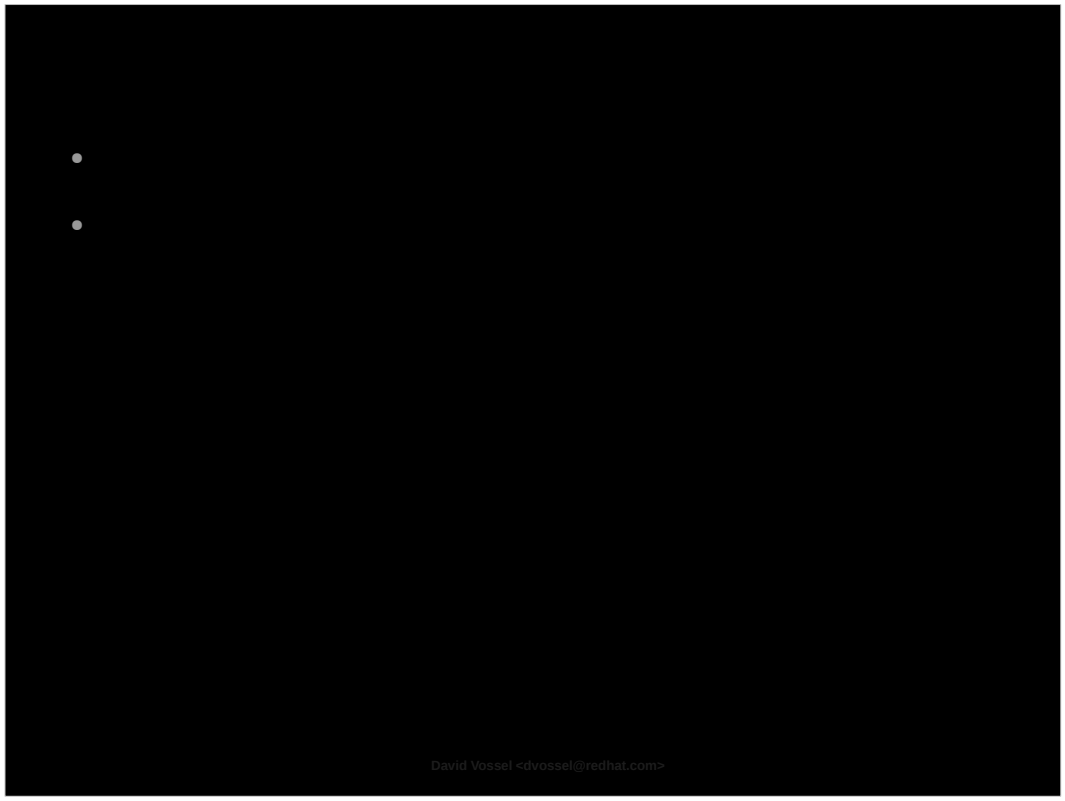
So far we've been openstack control plane highly available

But I want to talk about something different.

Compute nodes HA Ideological differences

i'll explain. i'm sure you all have heard the pets vs cattle analogy.

Pet services are unique.
- you only have a couple of them.
- You name them
- spend lots of time and energy maintaining them



Here's the problem.

somehow we've told ourselves that HA is only for the pet scenario.

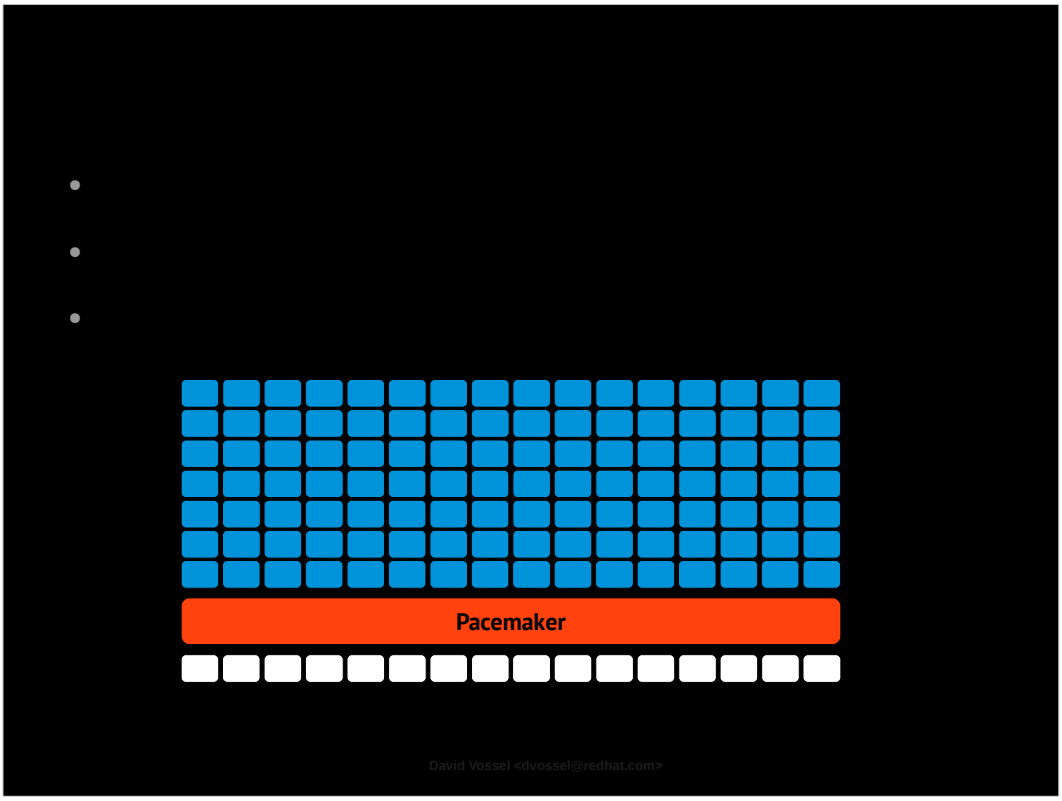
Somehow the herd saves us from needing HA...

That's not true.

Certainly HA for pets is different that cattle.

Don't get free pass for CATTLE HA

Still want to detect and automate the recovery of those cattle instances to **maximize availability.**



Pacemaker bridges the gap between the old HA world and the new.

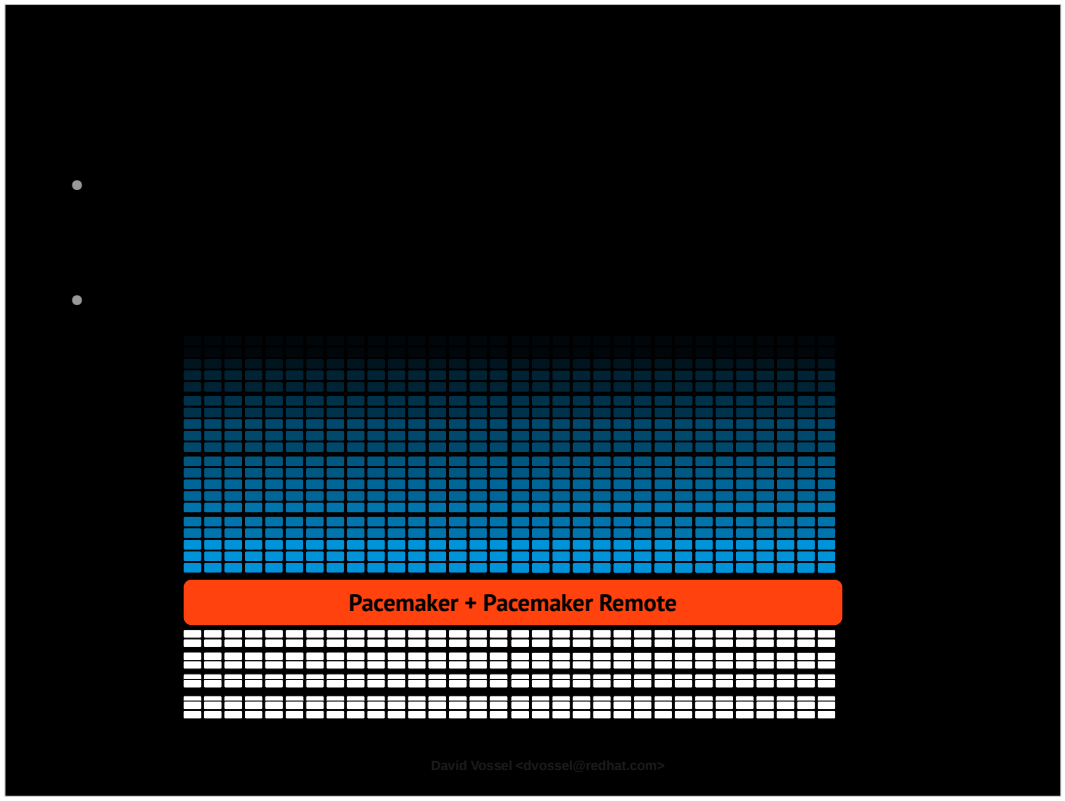
We can see this in how we scale.

The core pacemaker infrastructure has traditionally been limited to 16 nodes.

This is a limitation in the corosync messaging layer.

For pets, this is fine... even for some small herds this is fine.. but for cattle. 16 nodes is a joke.

So we did something about it.



We recognized that Pacemaker's policy engine.

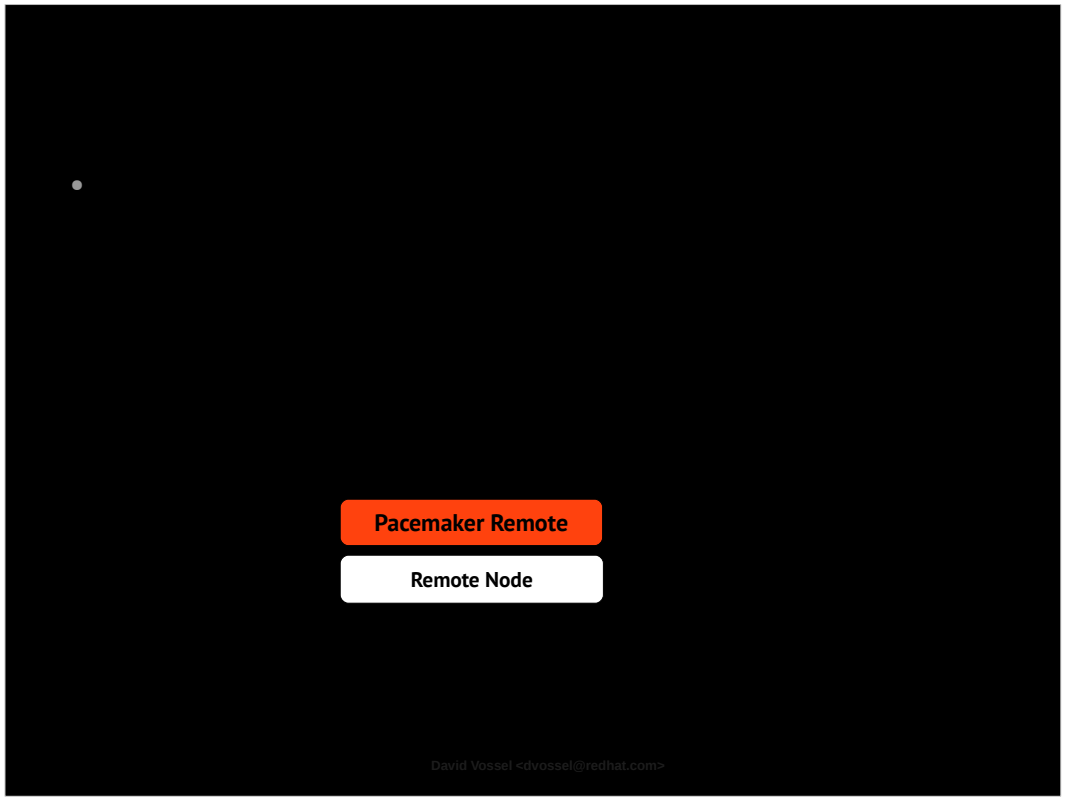
The Finite State Machine, is our most valuable asset.

How can we scale the ability to enforce the policy engine outside this artificial 16 node limit?

Pacemaker remote.

Pacemaker Remote is not bound by any messaging limitations.

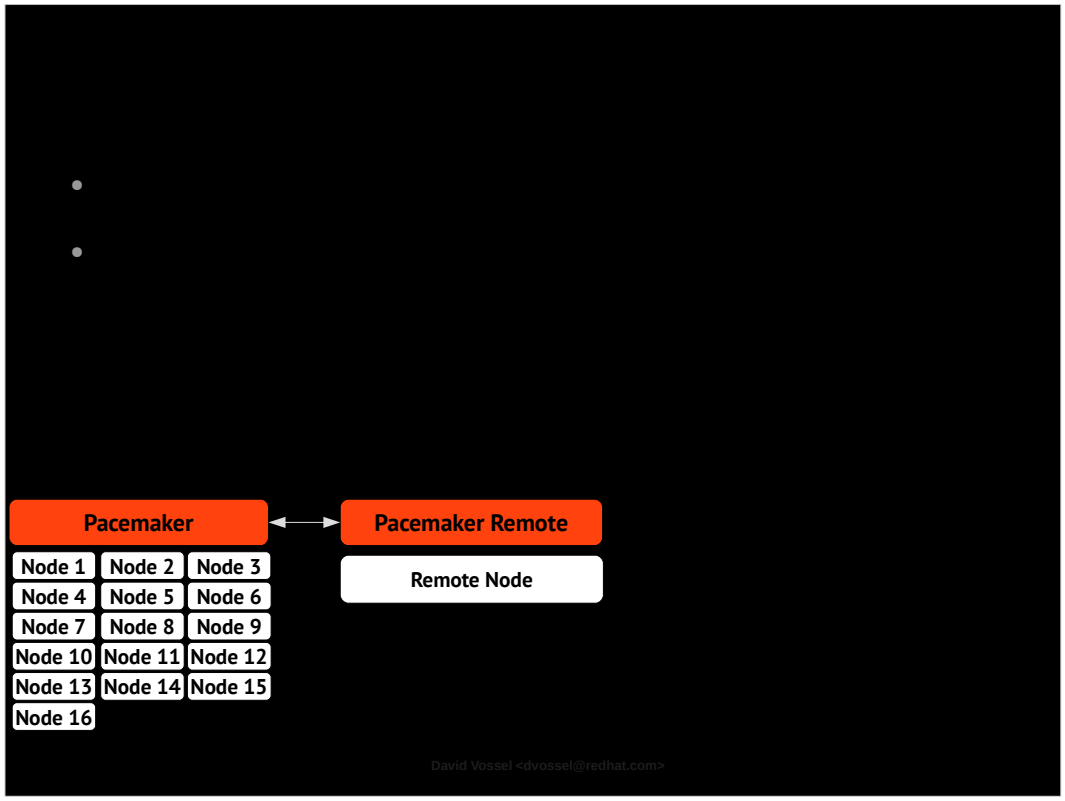
With pacemaker remote, we can scale pacemaker to 100s and 1000s of nodes.



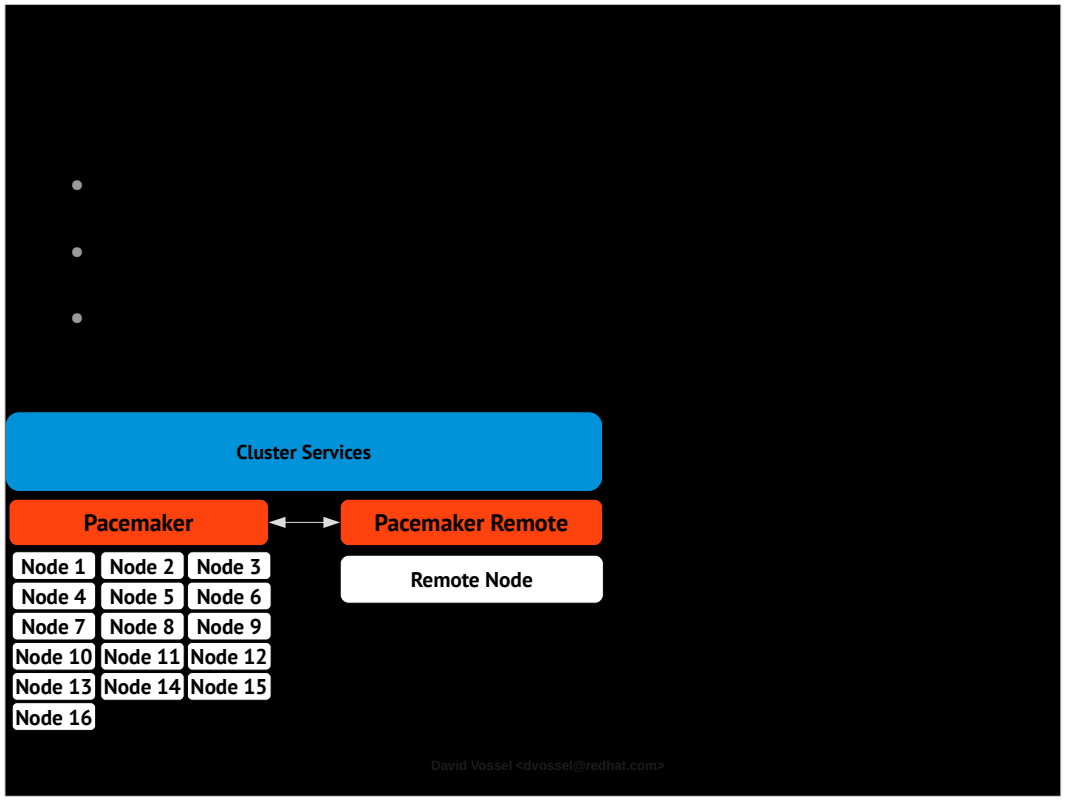
So what is pacemaker remote?

- Pacemaker remote is a daemon.

- This daemon is a lightweight way of integrating nodes into a pacemaker cluster.



- If we're really going to get down to it. Pacemaker remote is a way to integrate nodes into the cluster without requiring corosync.



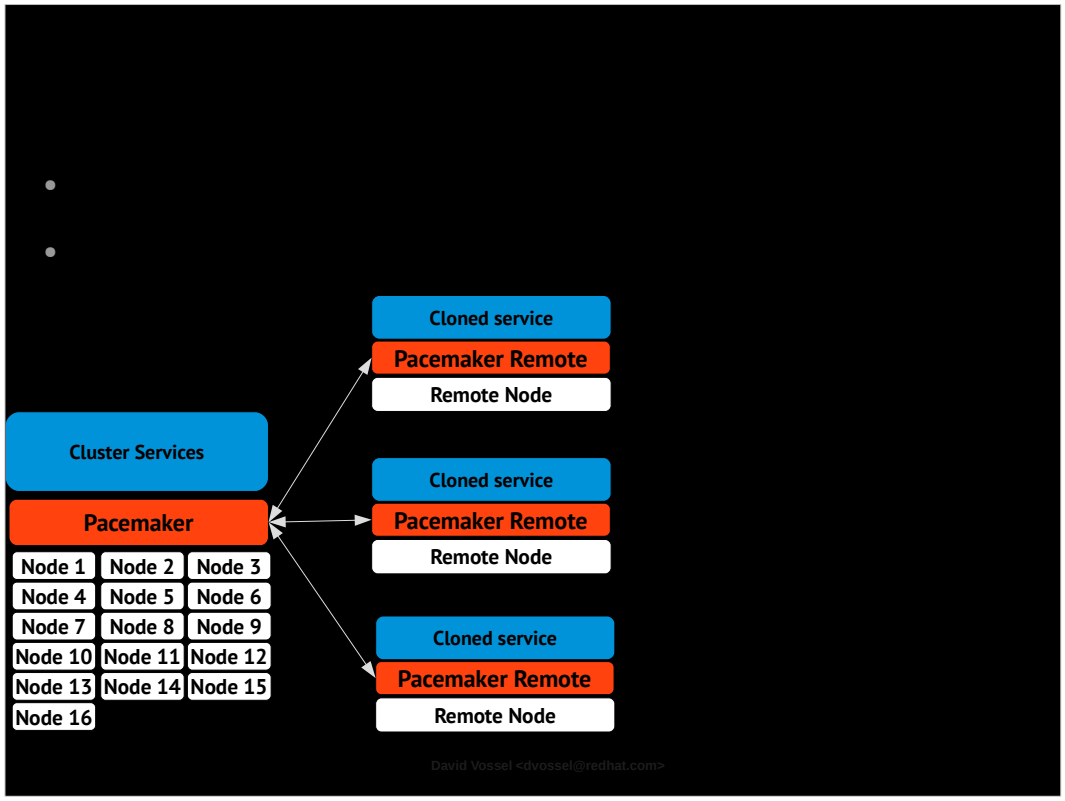
With pacemaker remote, cluster services

(the things pacemaker manages)

Act the exact same across both pacemaker and pacemaker remote nodes.

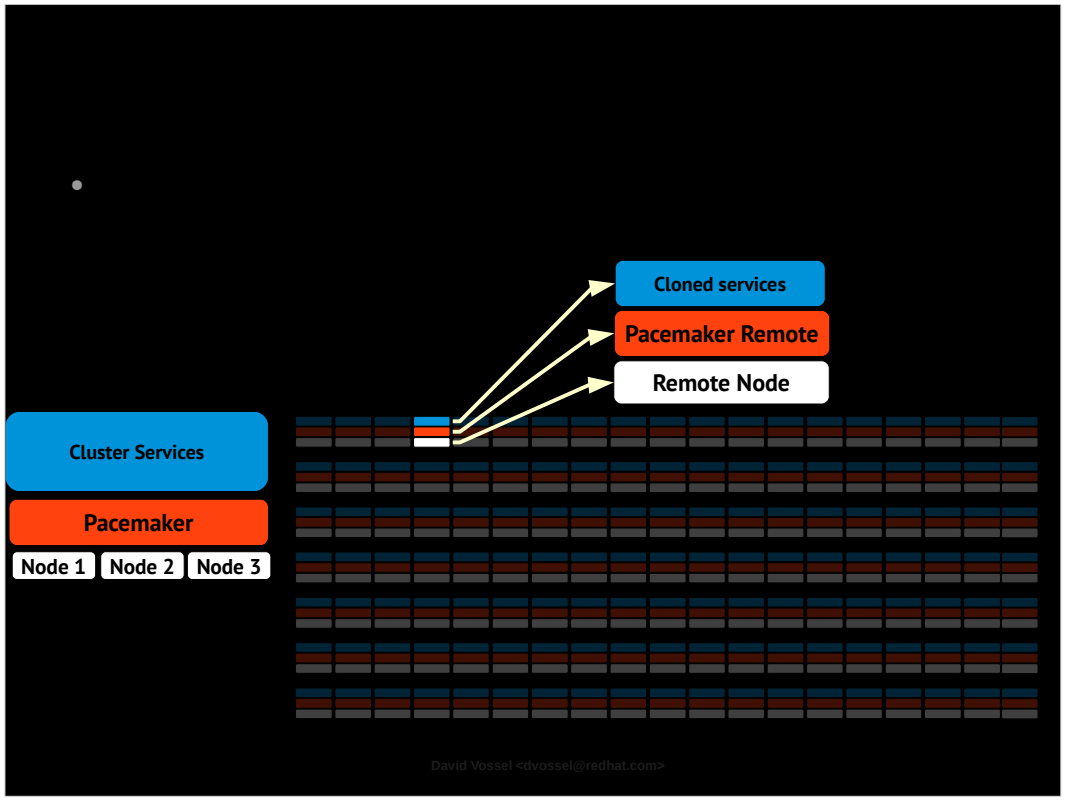
It doesn't matter what kind of node it is.

The cluster looks like a single partition to pacemaker.



While pacemaker and pacemaker remote can both run the same kinds of services.

For scaling, pacemaker remote really thrives when we use cloned resources.



requires a small pacemaker cluster for operation.

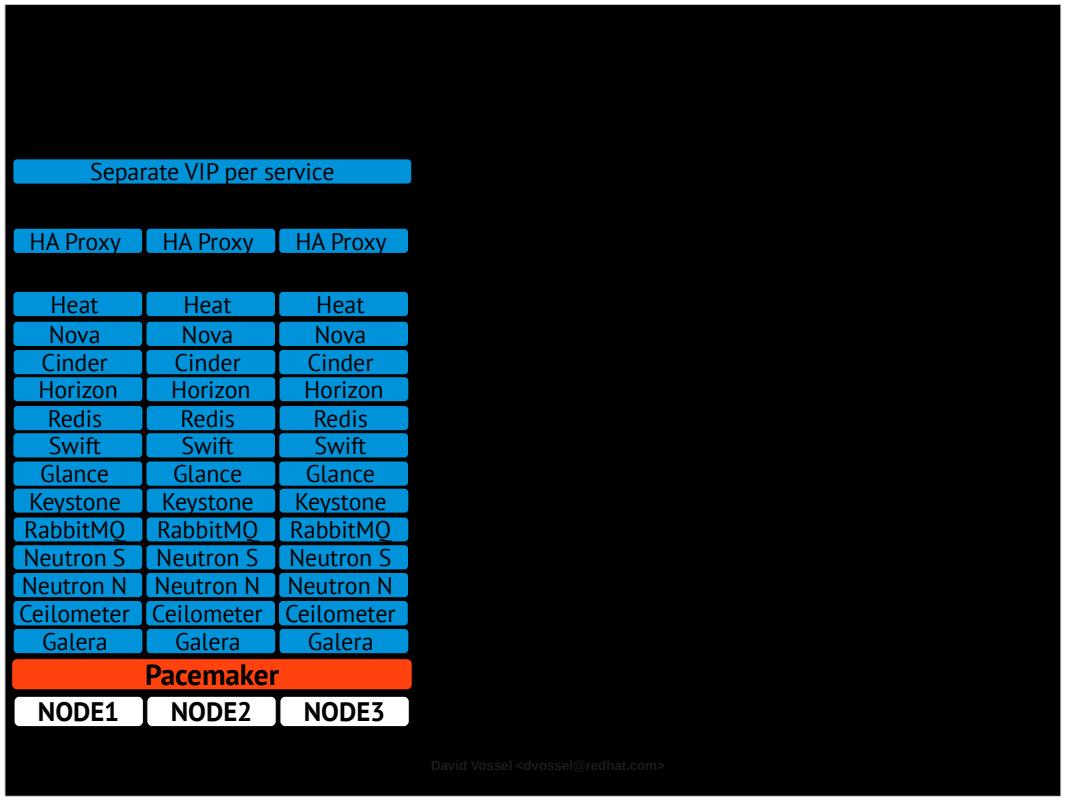
The pacemaker cluster is making decisions and executing decisions on remote nodes.

pacemaker remote is monitoring all the services on it's node and sending events back to pacemaker.

 use case I'm envisioning leveraged best in.

A small pacemaker cluster. servicing a larger set pacemaker remote which can scale horizontally

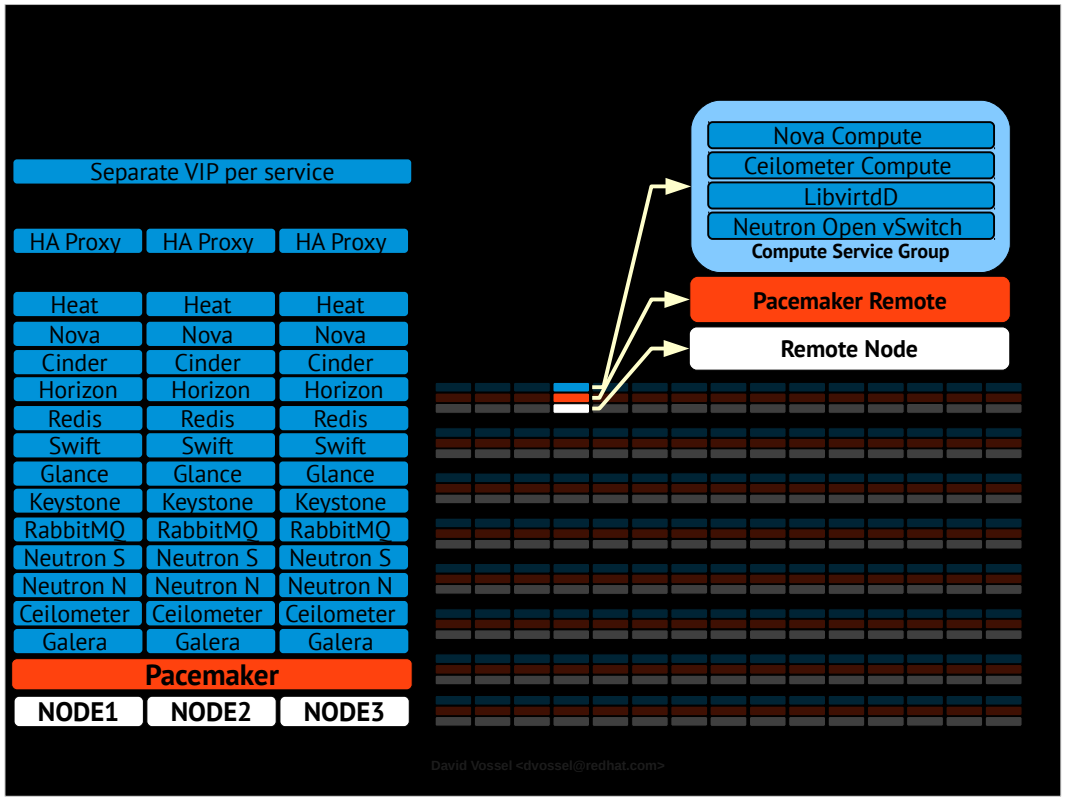
SO getting into our OpenStack compute node architecture, this actually fits us quite well.



Looking back at our collapsed controller node example from earlier.

We already have a small pacemaker cluster running all our Controller Node services.

Utilize pacemaker cluster to power a much larger set of Compute Nodes with pacemaker remote

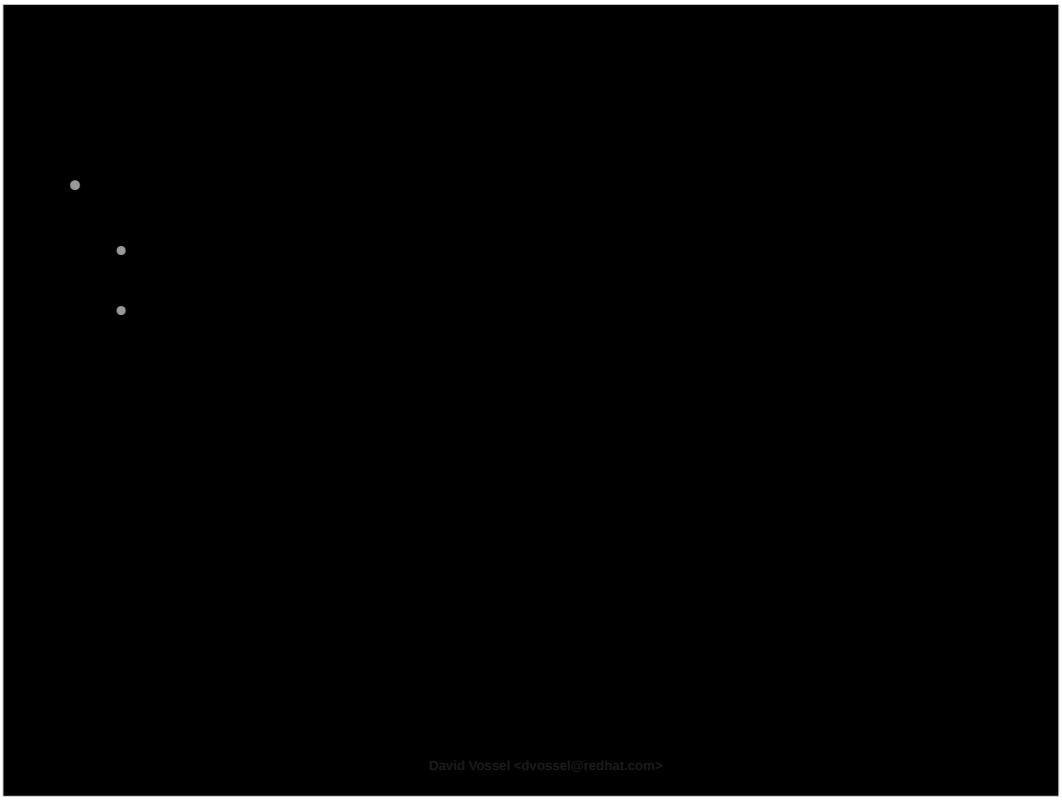


Compute services are identical for every Compute node,

every instance of pacemaker remote is running the exact same set of services.

<talk about cloned group>

Cloned services is where pacemaker remote works best, so this is really in many ways a perfect fit for us.



So what does this solution give us.

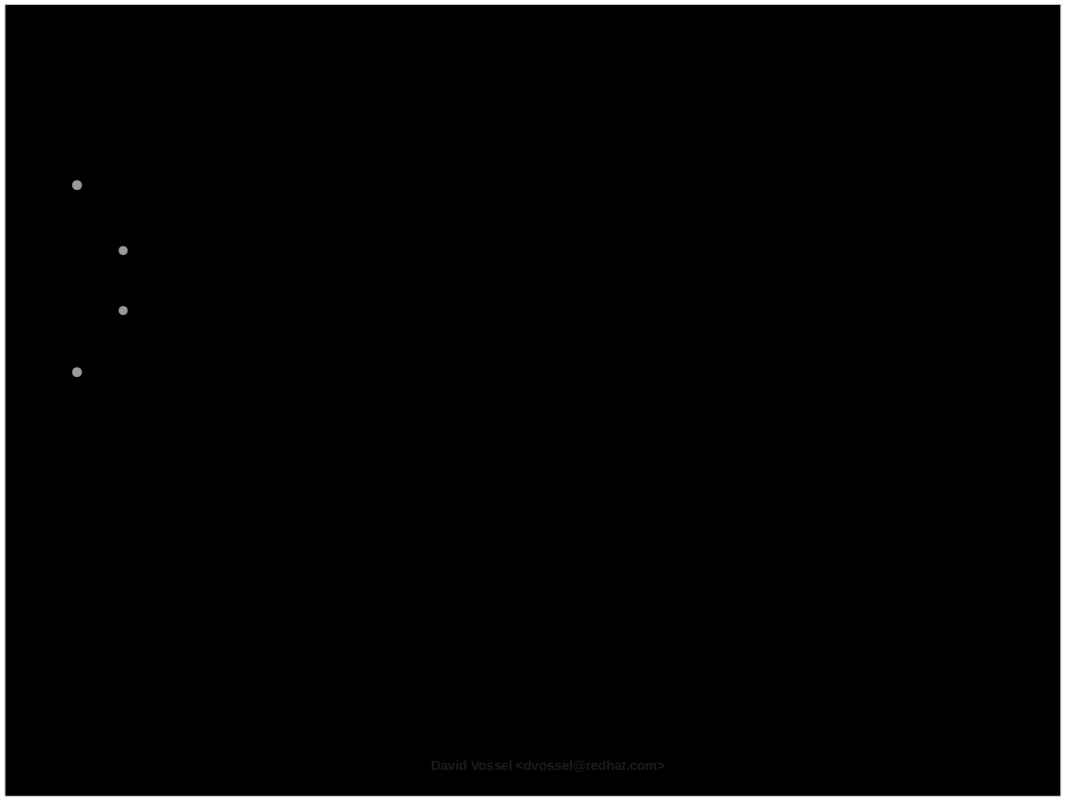
Maximize the availability of compute instances.

Not only can pacemaker recovery Compute node services in the event of a failure.

Pacemaker has the ability to help evacuate compute instances from failing Compute Nodes.

Pacemaker is actually coordinating with nova to do this

We monitor... trigger migration during failure.



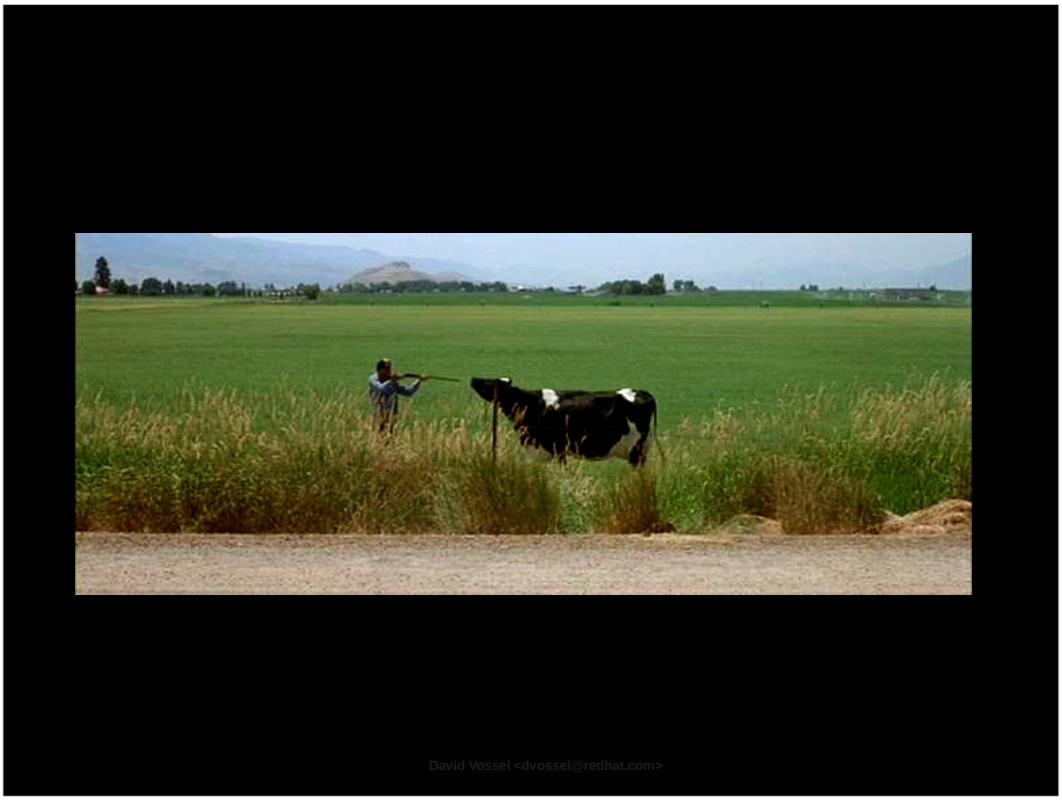
But If pacemaker detects a compute node has gone bad.

Something that is beyond repair.

In this case, we don't want compute resources assigned to a failing node... a node we know has issu

We want to evacuate whatever compute instances are on that node and isolate that node from the rest of the herd.

To do this Pacemaker remote has a secret weapon.



Stonith!

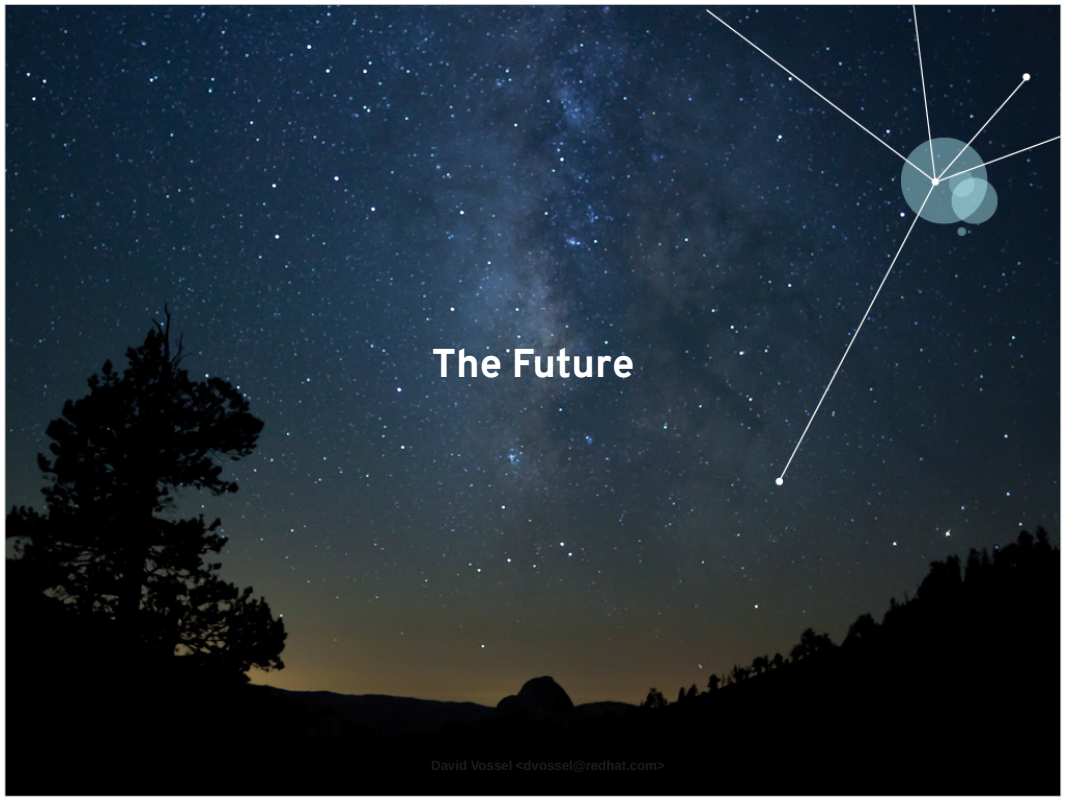
Remember, this is still pacemaker

stonith works for pacemaker remotes the exact same way it does for normal pacemaker nodes.

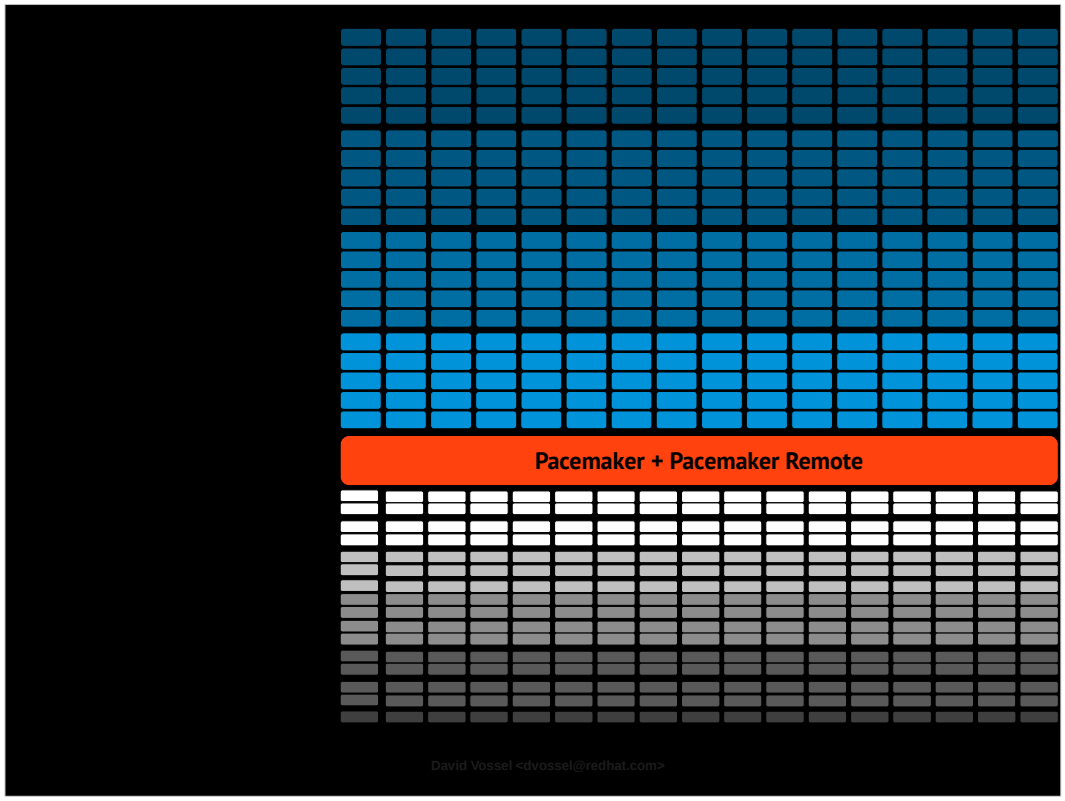
If cattle start doing really bad things pacemaker doesn't just sit back and watch. Has the ability to physically kill the bad cattle using fencing.

With STONITH We can shoot bad compute nodes and trigger evacuation of VM instances

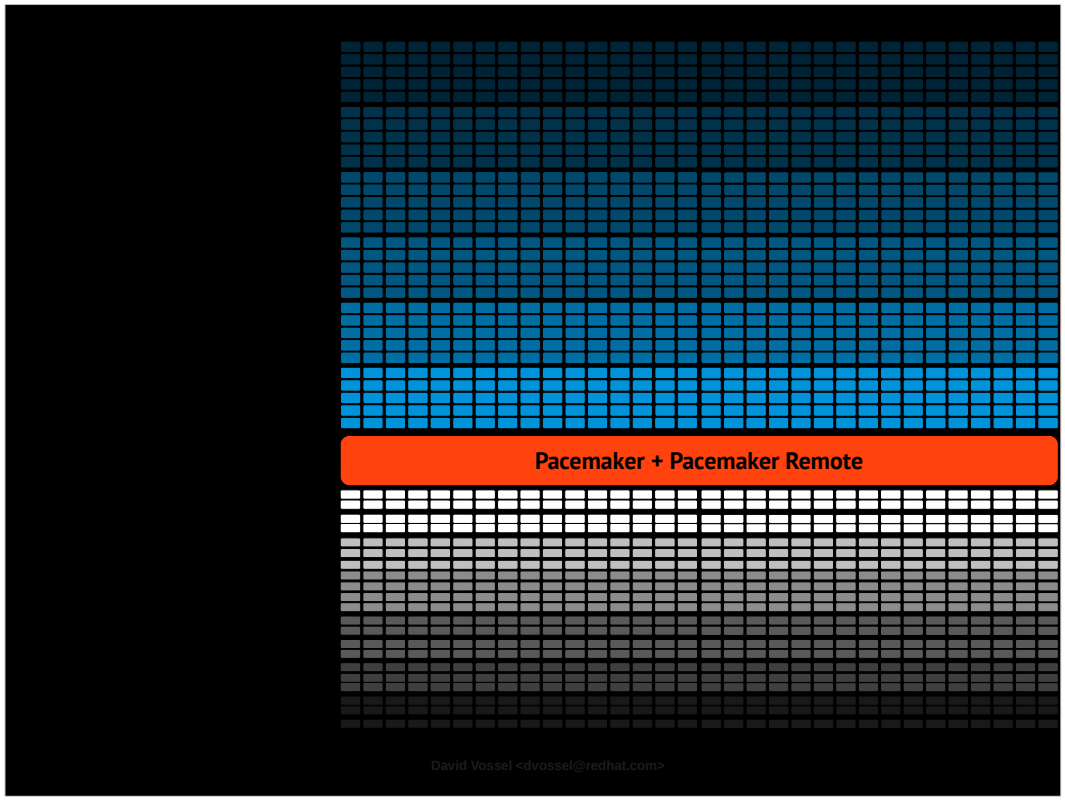
pacemaker_remote is heartless. And that's kind of what we're looking for here i suppose.



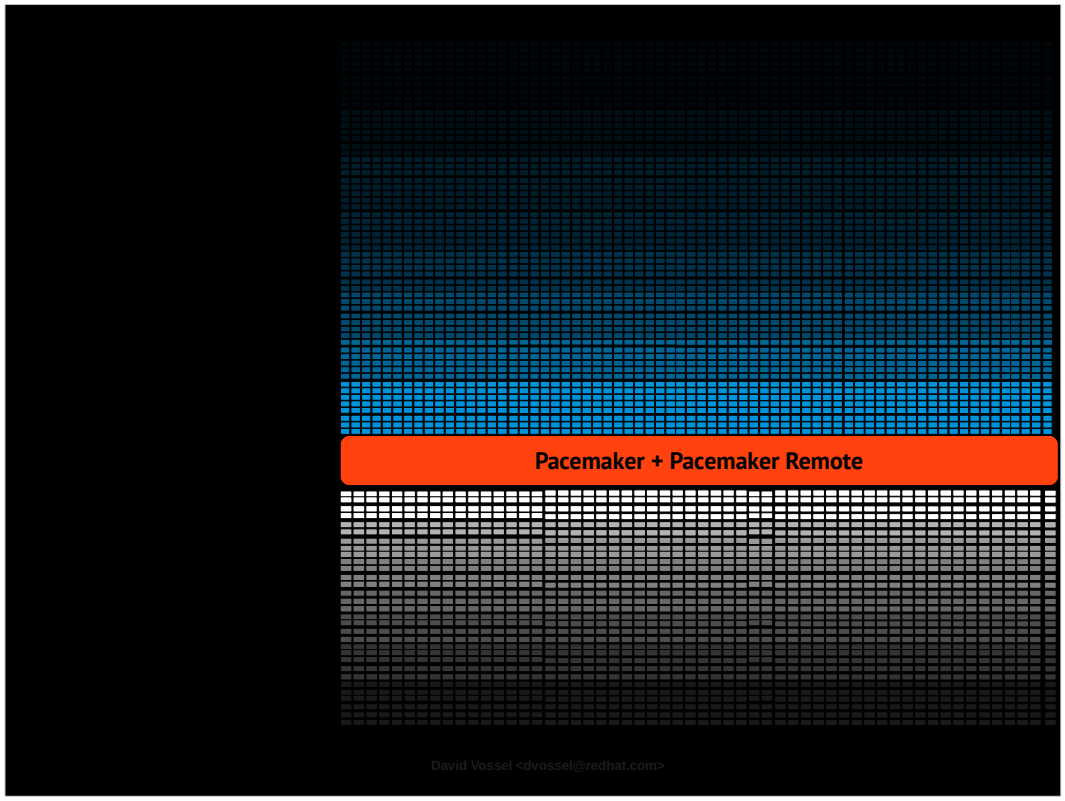
So, what's the future for pacemaker?



The problems i'm most intrested in involving determining how far we can scale pacemaker with pacemaker remote.



How man services can we run?



How many nodes?

A few years ago pacemaker was used primarily for 2 node clusters.

We understand the power of pacemaker combined with OpenStack and it's been incredibly rewarding to unlock all of this untapped potential pacemaker had to offer.

Pacemaker is a solid foundation for OpenStack HA and I'm excited to see where all this goes.


Questions?
Visit us at
clusterlabs.org
David Vossel <dvossel@redhat.com>