Resin

Lightweight, Open Source Java EE Application Server

resin®

CAUCHO

Resin Overview

- high-performance, cloud-optimized Java® application server
- Resin® cloud support
  - third generation clustering,
  - optimized to run in ec2, IaaS environments
- First java app server to support java ee WebProfile®
  - supports JMS, servlets, JPA, CDI and more
- DevOps friendly and cloud ready
  - CLI and REST controllable cloud platform
  - PaaS simplicity with IaaS control

resin®

CAUCHO

Resin Basics

- Older than Tomcat
- Used by big companies like Toronto Stock Exchange, Salesforce.com and Conde Nast
- Known for its speed, ease-of-use, and small footprint
- Java EE Web Profile Certified
- Resin Open Source – fast open source, Java EE WebProfile Java Application Server
- Resin Pro – adds world class support, health system, clustering, cloud

resin®

CAUCHO

Quotes

"It sounds silly and over simplistic, but Resin just works! It is easy to install, it is easy to get started, it is easy to configure, it is easy to get into a cluster, it is easy to work with, it shows at every level that Caucho focused on the target of a compact, efficient, easy JEE engine and delivered."
– Nathan Tableman / Sr. Manager of Development / Conde Nast Digital (Vogue, GQ, Wired)

"Salesforce.com has relied on Resin to run our market-leading CRM services for years. After evaluating competing products, we found Resin to be the most reliable, scalable and cost-effective solution."

"We chose Resin because we wanted a secure, high performance Java Servlet engine that supports clustering and failover at a reasonable price."
–Evan Simeone / Product Manager / eMeta

"Caucho's Resin application server has achieved Java EE 6 Web Profile Compliance. Congratulations to the team on reaching this milestone!"
Oracle GlassFish Blog, Alexis Moussine-Pouchkine

resin®

CAUCHO

Resin Cloud and Clustering

Lightweight, Open Source Java EE App Server

resin®

CAUCHO

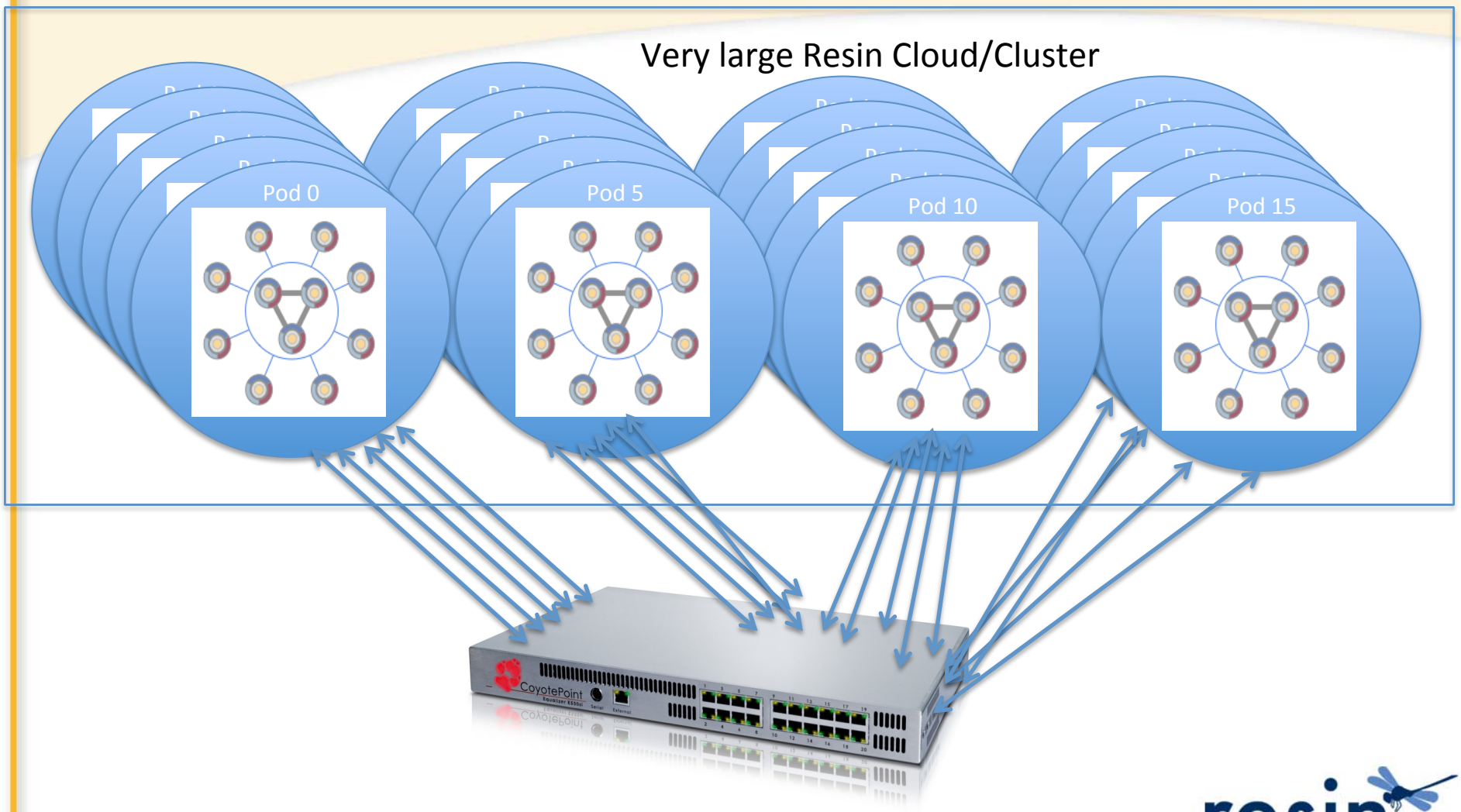
Cloud Support

- Cloud support is Resin's 3rd generation clustering
- Designed for elastic cluster
  - Ability to add and remove servers from a live cluster

- Resin clustering uses Triad server model
  - Triad is a triple redundant hub
  - Hub and spoke network
  - Triad Responsible for load-balancing clustered services
  - Triad servers also service regular requests

- Clustered Services
  - Load Balancing, Caching, JMS, Clustered Deployment
  - Added servers enroll automatically in services

Very Large Cloud



resin®

CAUCHO

Resin Clusters

- Every server in the cluster serves up requests for the same set of applications
- Servers can be load balanced
- Servers share queue, session, and object cache data
- Cluster Data is replicated for fault tolerance
- Load balancer can add/remove servers to distribute the load at runtime (dynamically)

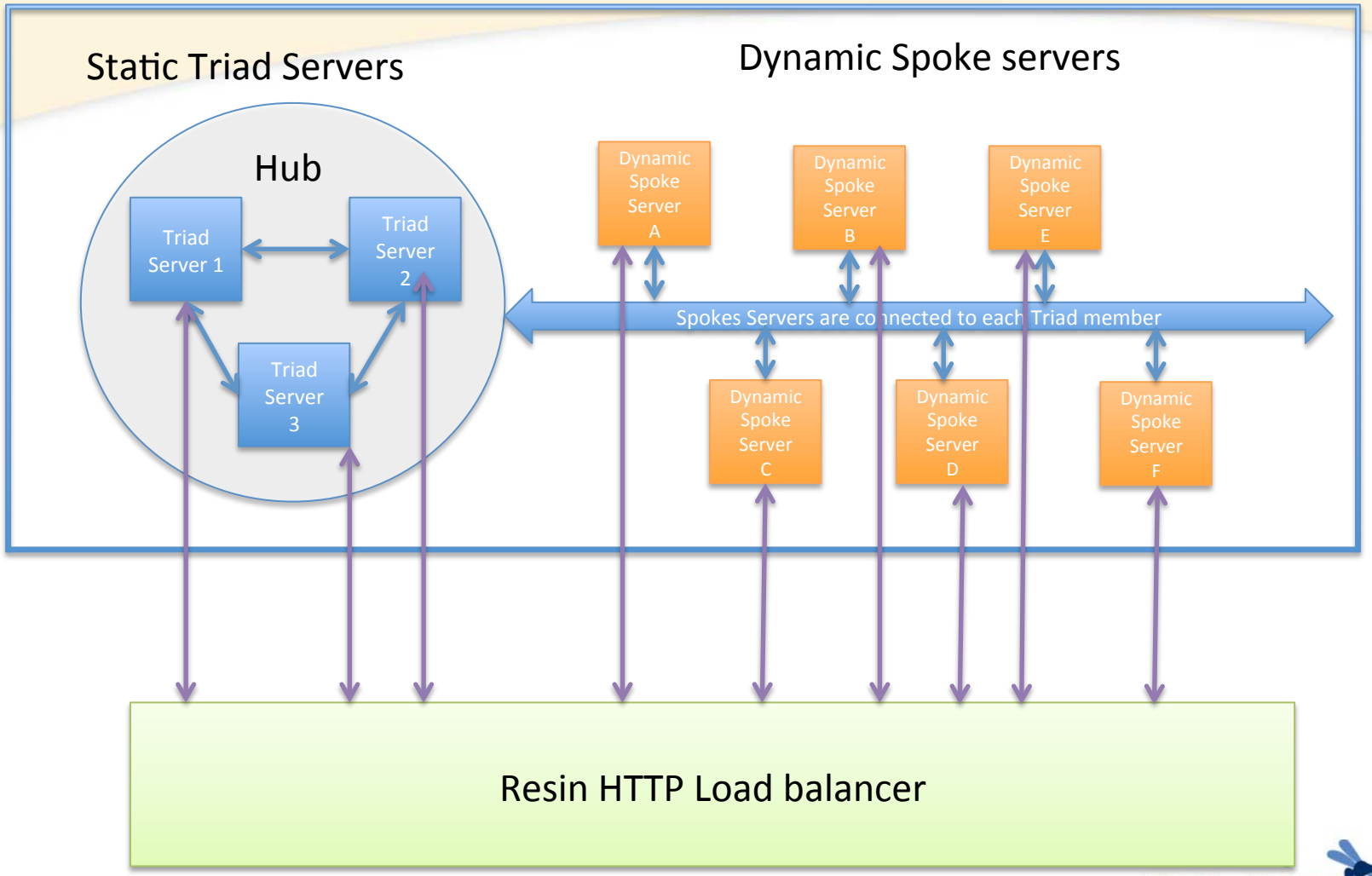
Clustering Overview

- Straight-forward elastic-server private cloud
  - Servers can join and leave
- Designed to avoid a lot of headaches associated with other clustering solutions
  - No massive data shuffling problems
  - No group splits, simple and reliable
  - Not overly chatty
  - Easy to understand and configure, don't need an army of consultants
- Resin prides itself on ***Operational Predictability***

Clustering Feature Overview

- To support elastic-server cloud, Resin maintains the following when servers are added and removed:
  - Data redundancy with a triad hub
  - Application deployment through a clustered transactional repository
  - Distributed caching and store for both servlet sessions and JCache caching (and Memcached)
  - Load balancing to the elastic spoke servers
  - Distributed management, sensing, and health checks
  - Elastic JMS message queues

Resin Simple Cluster



resin®

CAUCHO

Clustering always there

- Clustering is always enabled
- Even if you have single server, that server belongs to a cluster
- As you add more servers, the servers are added to the cluster
- Added servers automatically gain benefits like
  - clustered health monitoring,
  - heartbeats,
  - distributed management,
  - triple redundancy,
  - and distributed versioned deployment

Advantages of Clustering

- HTTP Load balancing and failover
- Elastic servers: adding and removing servers dynamically
- Distributed deployment and versioning of applications.
- A triple-redundant triad as the reliable cluster hub
- Heartbeat monitoring of all servers in the cluster
- Health sensors, metering and statistics across the cluster
- Clustered JMS queues and topics
- Distributed JMX management
- Clustered caches and distributed sessions

Why you need clustering

- most web applications start with a single server, at least during development
- Later severs get added for increased performance and/or reliability
- Developers are familiar with single server deployment
- As web application usage grows
  - Single server can have hardware limitations especially if more than one app is hosted on app server
  - Hardware limits can occur like chronic high CPU and memory usage etc.
  - Other resources can be adversely impacted

Server load balancing

- Server load-balancing solves scaling problem
  - lets you deploy a single web app to more than one physical machine
  - Machines share the web application traffic
  - Reducing total workload on a single machine and providing better performance from the perspective of the user
- load-balancing achieved by redirecting network traffic across multiple machines via a hardware or software load-balancer
- ***Resin supports software load balancer as well as hardware load balancers***
- Load-balancing increases reliability/up-time
  - if one or more servers go down for maintenance or due to failure,
  - other servers can still continue to handle traffic
- With a single server application, any down-time is directly visible to the user, drastically decreasing reliability

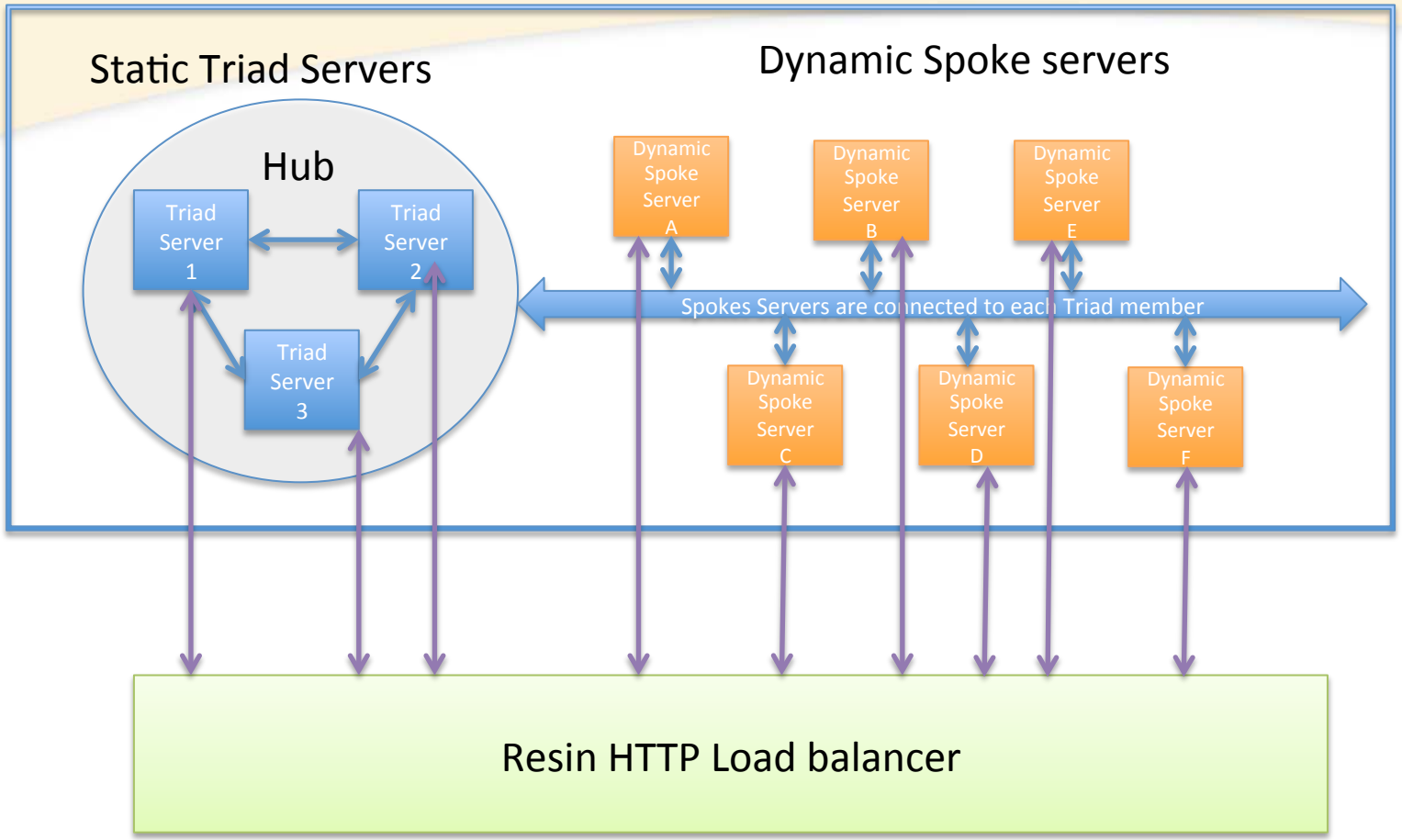
Clustered Deployment

- Cluster deployment
  - Resin's allows deploying web applications to the entire cluster
  - Allows updating new versions of your application to all servers in cluster
- Cluster deployment versioning
  - Verify that all servers in the cluster are running the same version
  - When a new version deploys, all servers have an identical copy, then the new app is validated and if successful new app is available on every server in the cluster
  - clustered deployment is reliable across failures because its built on a transactional version control system (Git)
- New servers checks with the cluster (redundant triad) for latest version of applications version
  - Whenever a server comes up it checks for latest version of applications
  - Application versions consistent across cluster

Stateful Web Applications

- If web applications were stateless, load-balancing alone do the trick for scalability
- Many web applications are relatively heavily stateful.
  - Even very simple web applications use the HTTP session to keep track of current login, shopping-cart-like functionality and so on.
  - JSF and Wicket heavily use HTTP session a lot
  - CDI and Seam also have a lot of conversational state stored in HTTP session
- When stateful web applications are load-balanced, HTTP sessions are shared across application servers
- You can use a sticky session to pin a session to a server
  - Session state still needs to be shared (replicated) among multiple servers for failover so users don't lose their session data
- With failover of a stateful application, session has to be moved to new server picked by load balancer
- Without failover, the user would lose a session when a load-balanced server experiences down-time.

Failover



resin®

CAUCHO

Triad Spoke and Hub

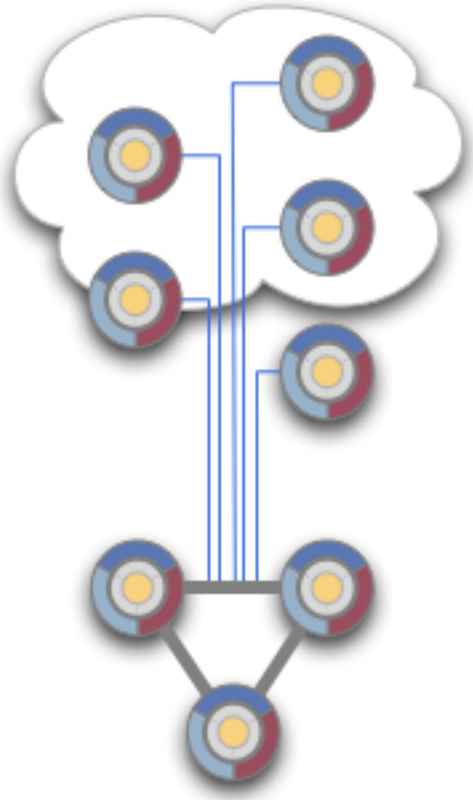
- Triad based on lessons learned from 13 years of clustering support
- Triad is the hub
- Triple Redundancy so you perform maintenance on one box while two remain for fault tolerance
  - Load increases 50% for each remaining server instead of increasing 200% if just using a backup model
- Important persistent data is stored and replicated on Triad servers (easy to understand)

Triad Model Advantages

- Easier to understand, no magic
  - Complex models very hard to support with Ops
  - Triad gives you ***operational predictability***
  - Triad makes private cloud configuration much simpler
- Best combination of centralization, reliability and speed
  - Centralized responsibilities in hub (called Triad)
  - Reliability for upgrading and maintenance

Triad Model

- Based on years of experience from developing distributed clustering for a wide variety of user configurations
- Refined to a hub-and-spoke model
- Interlocking benefits for the dynamically scaling server
- When you bring down one server for maintenance, ***triple redundancy continues to maintain reliability***
- ***Elastically*** add and remove dynamic servers safely without affecting the redundant state
- Network processing load-balanced across three servers, eliminating a single point of failure: ***fault resistant***
- Your servers scale evenly across the cluster because each new server only needs to speak to the triad, not all servers: ***reduced chatter*** and ***operational predictability***
- Your large site can split cluster into independent many "pods" of up to 64-servers to maintain ***scalability***



resin®

CAUCHO

Elastic Spoke Servers

- Non-Triad servers are spokes in hub-and-spoke model
  - 'spoke' servers not needed for reliability
- Spoke servers don't store
  - primary cache values,
  - primary application deployment repository,
  - queue storage (JMS)
- Spoke servers can be added at will
  - add and remove them without affecting the primary system
- Optimized to reduce excessive data shuffling if a server is removed

Triad Servers

- Important data that needs to be replicated is always stored in the Triad
- Master Queue data, master cache data, master deployment files
- Other servers just use Triad
- This model is easy to understand
  - No complex data sharing and copying scheme
  - Non-Triad servers are more expendable
  - Triad servers are worth additional reliability attention

Server Types

- Dynamic server
  - Servers that are elastic
  - Also described as ***elastic servers***
  - Does not need a fixed IP
  - Easy to bring up and down
- Static server
  - Servers that are configured by id and ip address in the resin.xml
- First three ***Static Servers*** configured are Triad servers
- Triad servers also handle normal requests

Elastic Servers with Elastic Cluster Service

- Add and remove servers dynamically to the system,
- Automatically adjusts the cluster to manage those servers
- Deployed applications get pushed to the new server automatically
- Distributed caching is notified of the new server
- Load balancer updates itself to include the new server
- JMS queues update with the new server so the new server can get messages
- Cluster administration adds the new server so the new server can for example get health heartbeats from the Triad

resin®

CAUCHO

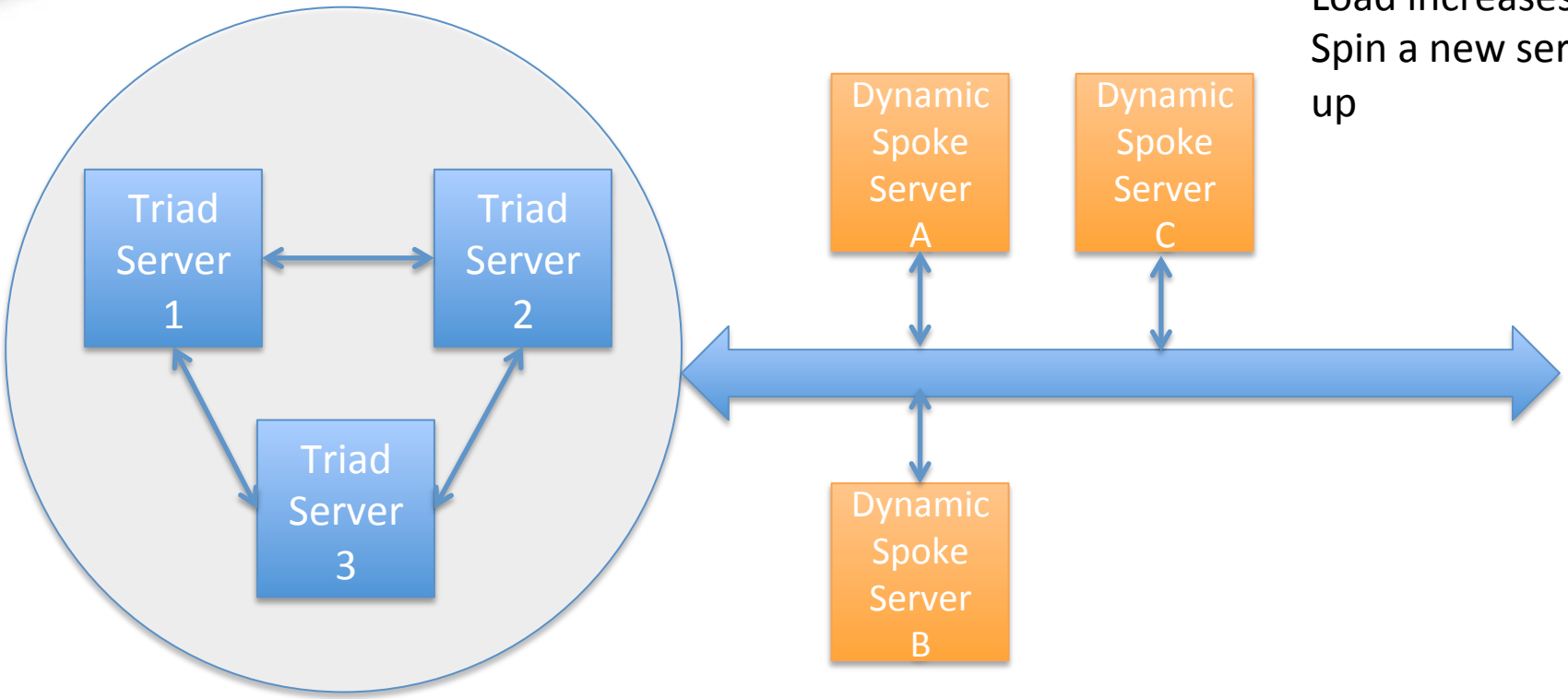
Baseline config

- Define 3 static servers in resin.xml
  - First three are always Triad
  - Triad servers are always static
- Copy resin.xml file to $RESIN_HOME/conf across all server boxes
- Attach new servers from command-line
  - As many as you need
  - These are Dynamic Spoke Servers

Baseline cluster setup

1. Define a resin.xml with three static servers as the triad.
2. Start the triad servers as normal.
3. Deploy .wars to the triad with the command-line 'deploy' to enable automatic deployment to the new servers.
4. Install a machine (or VM image) with Resin, your licenses, and the resin.xml.
5. Start the new Resin server with a --join-cluster command-line option.
6. The new server will load the applications from the triad and join in any clustered services like load-balancing and caching.
7. When load drops, stop the new Resin server as usual.

Basic Configuration



Load increases.
Spin a new server up

resin®

CAUCHO

Triad in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">
...
<cluster id="my-cluster">
  <server id="a" address="192.168.1.10" port="6800"/>
  <server id="b" address="192.168.1.11" port="6800"/>
  <server id="c" address="192.168.1.12" port="6800"/>
```

Define Triad

Spin up a new VM instance and add it to the cluster. It talks to the triad to join.

```
unix> bin/resin.sh --join-cluster my-cluster start
```

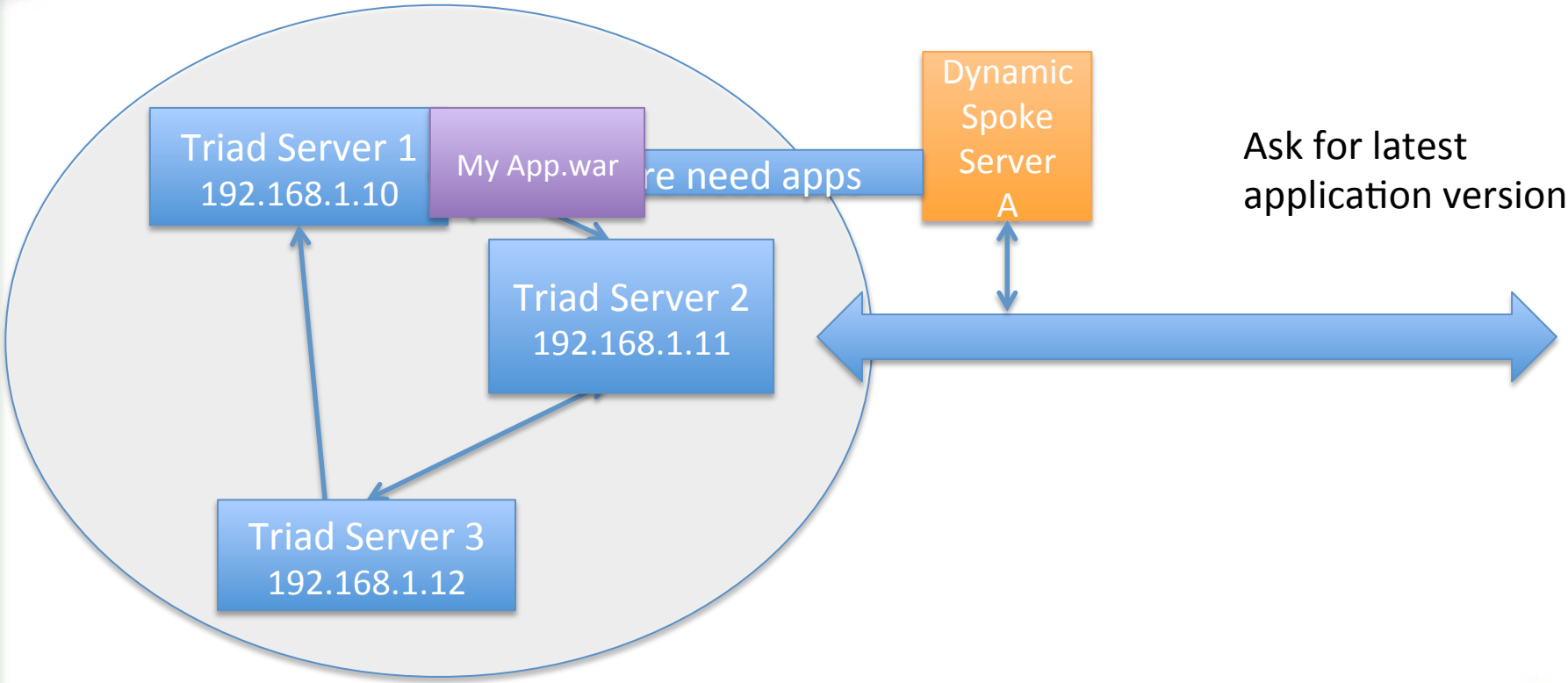
Spin up a new VM instance and add it to the cluster. It talks to the triad to join.

```
unix> bin/resin.sh --join-cluster my-cluster start
```

New server dynamically joining cloud

Spin up a new VM instance and add it to the cluster. It talks to the triad to join.

```
unix> bin/resin.sh --join-cluster my-cluster start
```



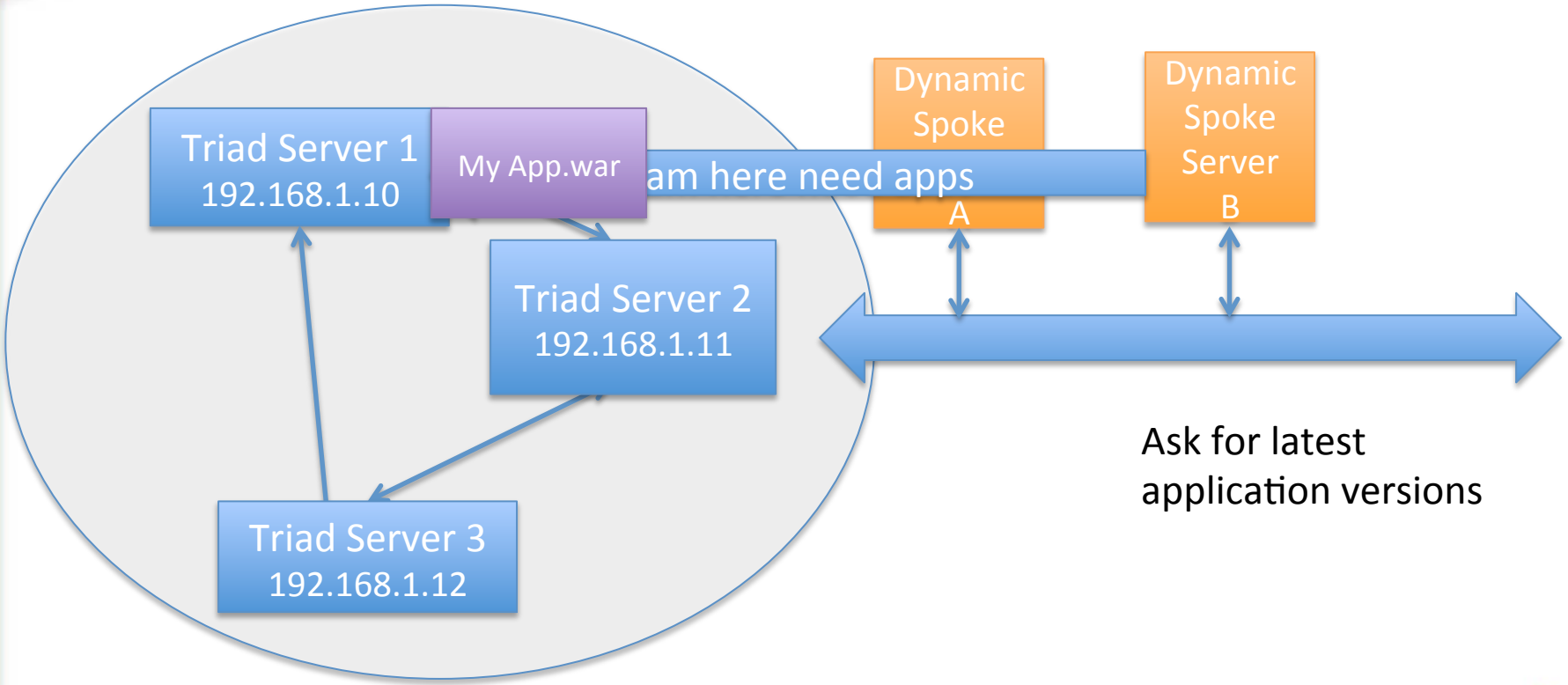
resin®

CAUCHO

New server dynamically joining cloud

Spin up a new VM instance and add it to the cluster. It talks to the triad to join.

```
unix> bin/resin.sh --join-cluster my-cluster start
```

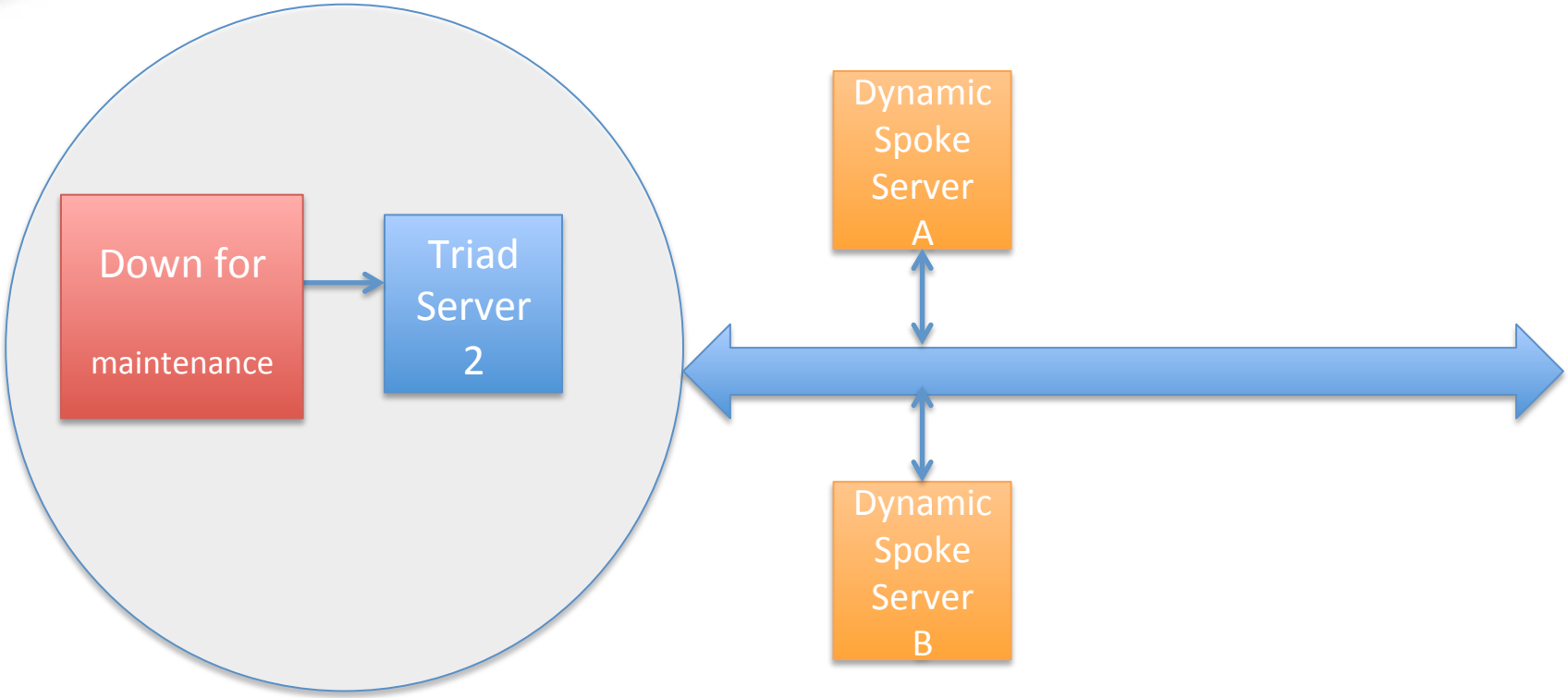


Triad with less than three

- You can have only ***one*** Static server in resin.xml
  - Configured Server is Triad, but there is no failover, if server goes down, cluster is down
  - Triad only has one server in it
- You can have two Static Servers in resin.xml
  - Both servers are Triad members
  - Triad only has two servers in it
  - You get failover but not as robust as three
- Three is the most reliable

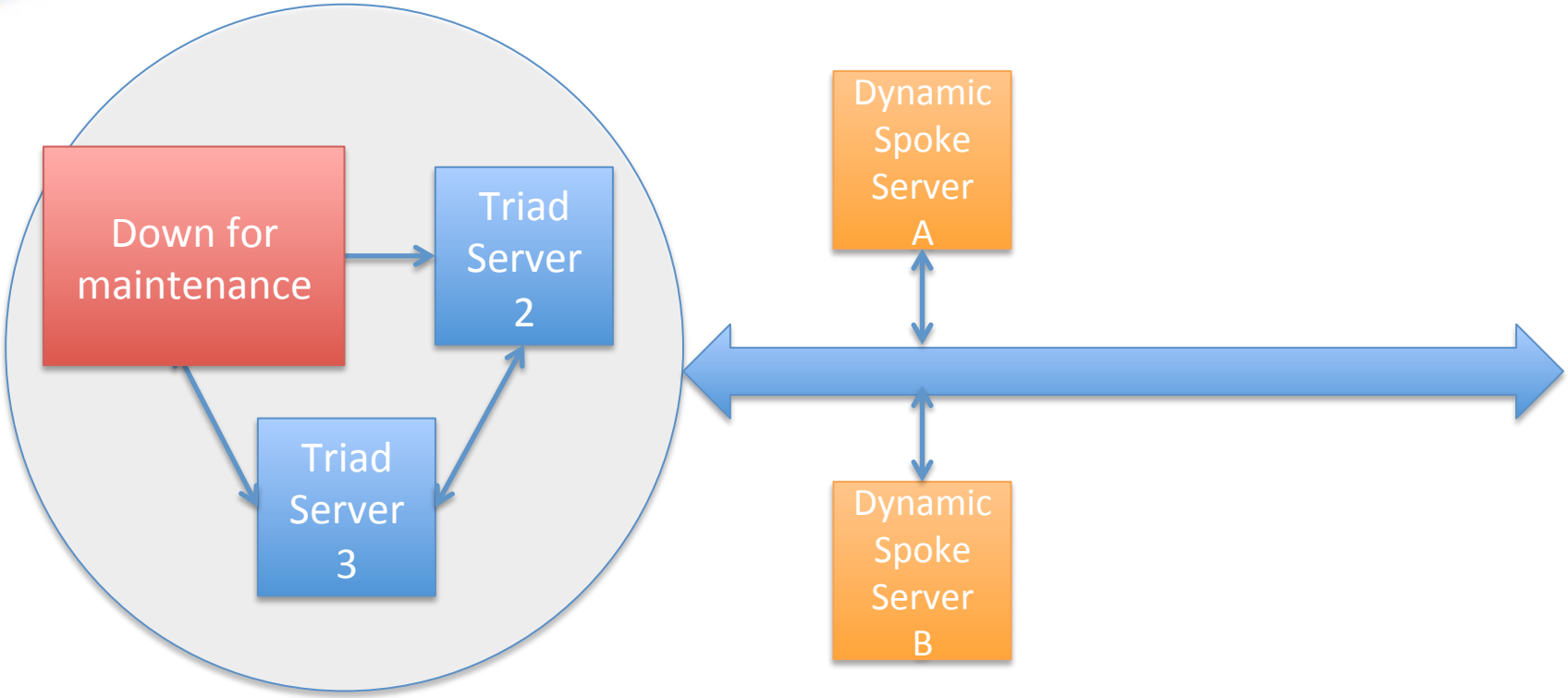
Two Triad Server Hub, problem with load

Triad responsibilities of Triad Server 2 goes up 200%



Basic Configuration 1 Triad Server down

Triad responsibility load increases by 50%
For Triad Server 2 and Triad Server 3

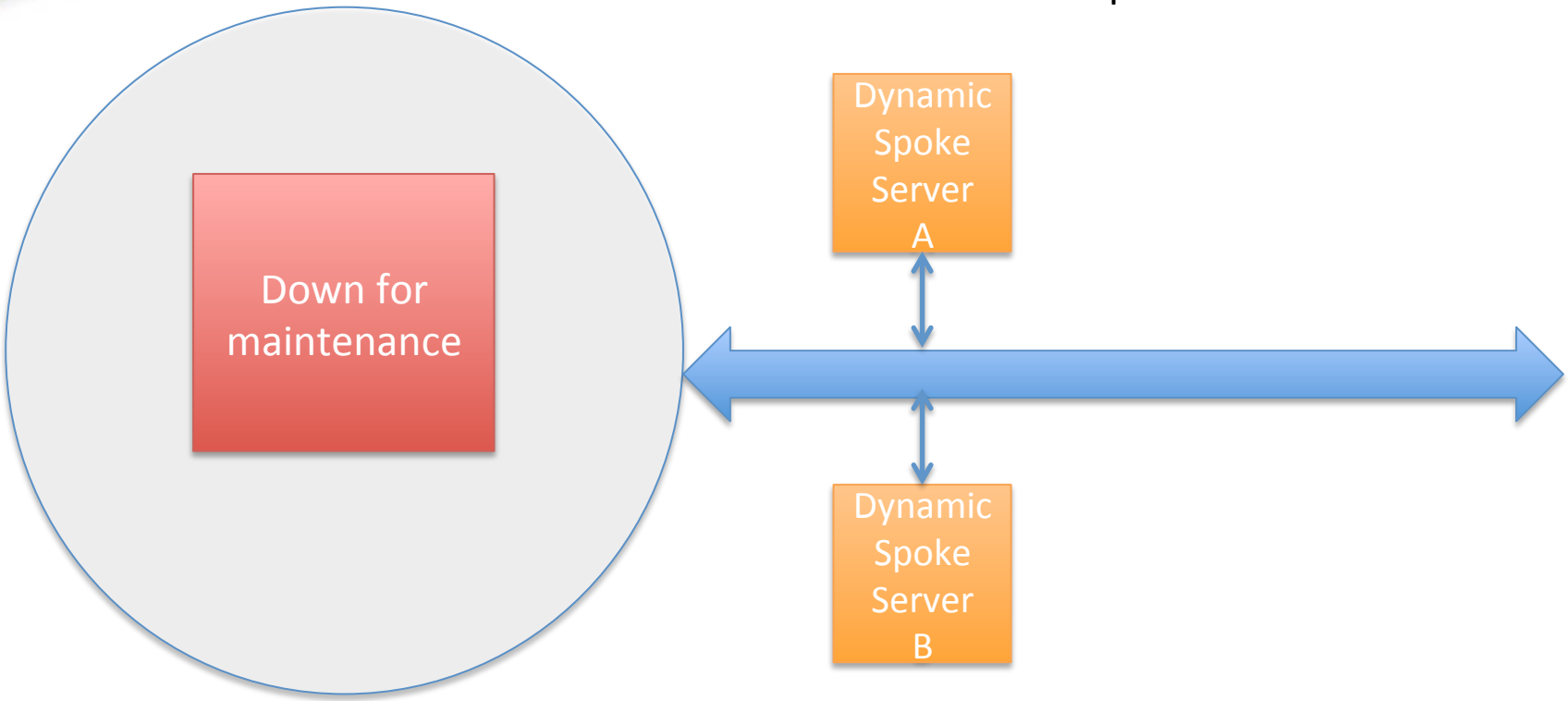


resin®

CAUCHO

One Triad Server Hub

Spoke servers in operation continue to work.
Clustering services stop. No new spoke servers can be
Added until Triad Server is back up



resin®

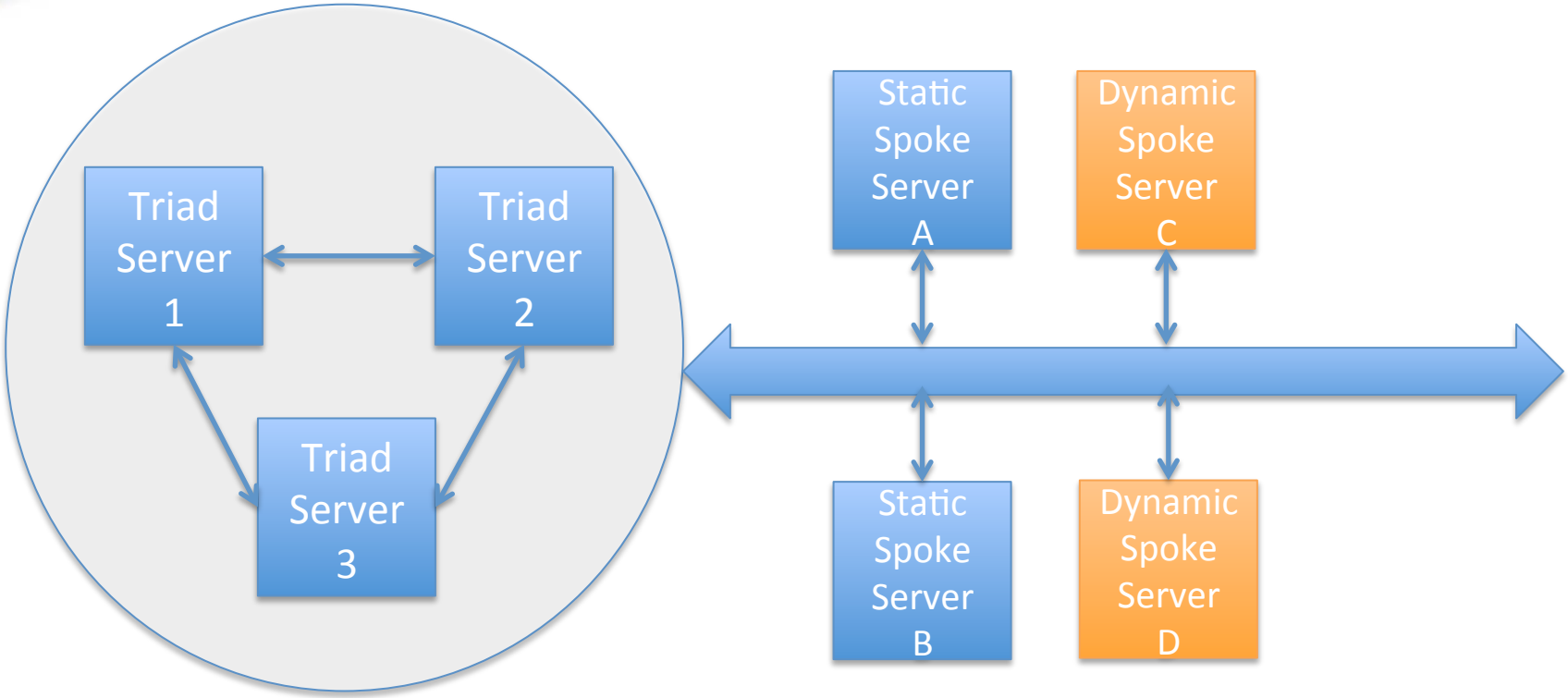
CAUCHO

More than 3 static servers

- You can have more than 3 static servers
- XML file can document all of the static servers ahead of time
- clustering administration (cluster heartbeats) let's you track up time and down time of all static servers
- Static servers show up in Topology

Static servers outside of Triad

Static servers can be spoke servers



resin®

CAUCHO

Web Tier full featured

- Dynamic load balancer that is cluster/cloud aware
  - Added servers automatically take load
  - Features similar to expensive Level 7 Hardware load balancer,
  - Level 7 is ADC - Application Delivery Controller
- HTTP proxy caching similar to Squid
- HTTP stack faster than Apache HTTPD

resin®

CAUCHO

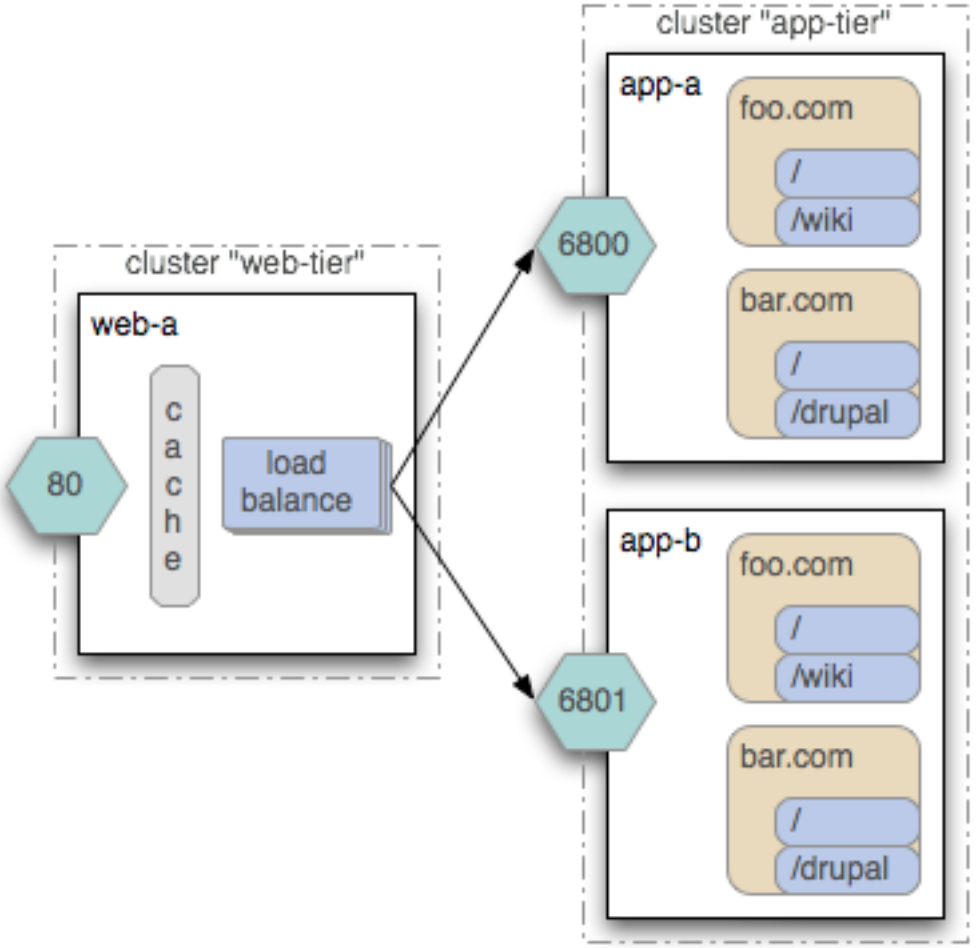
HTTP Load Balancing Easy to add

- If traffic increases beyond capacity 1 app server,
  - Add HTTP load balancer as well as the second application server
  - The load balancer is in the Resin web-tier used for HTTP load-balancing and HTTP proxy caching
  - Now you have load balancing and fault tolerance
- Two tier topology, with web-tier load balancer and app server tier
  - Easily system becomes two tier system
  - Resin can function as web-tier load balancer and app-tier app servers
  - Easy to setup, same config file as before, just a few XML entries and viola you have a HTTP load balancer
  - Just add <resin:LoadBalance> tag to forward request to app-tier
- A load balanced system in Resin
  - define the servers to use as the application-tier,
  - define the servers in the web-tier,
  - and select which requests are forwarded (proxied) to the backend app-tier servers.

HTTP Load Balancing Easy to Scale Out

- Small to medium level sites (under 64 servers per application) can use Resin's load balancer which is cloud/cluster aware
- Beyond 64 servers Resin works with hardware load balancers
  - Divide servers into cluster pods of up to 64 servers
  - hardware load balancer load balances to Resin load balancers
  - In effect, hardware load balancer load balances to Resin cluster pods

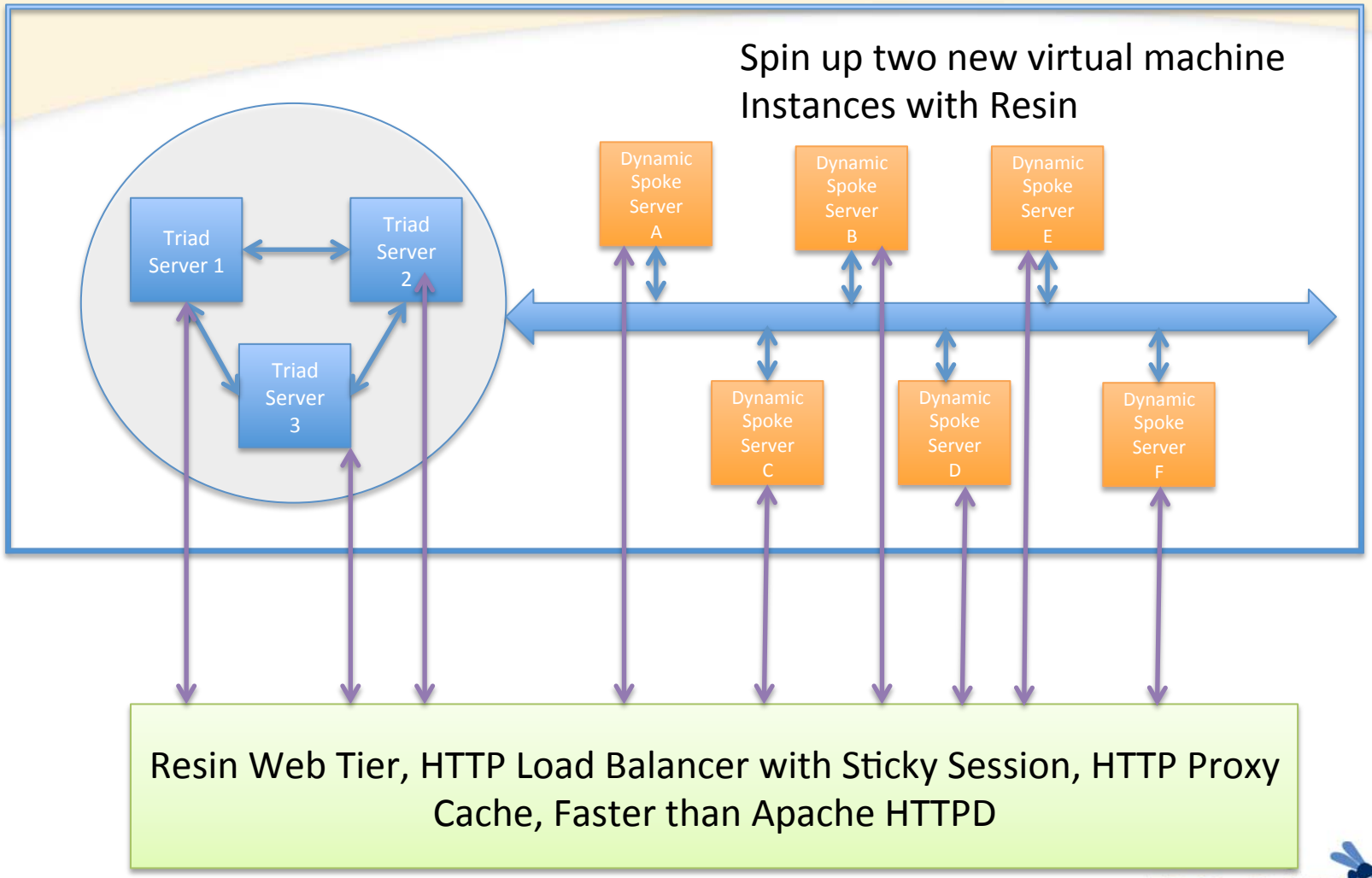
Load Balancer



resin®

CAUCHO

Resin Cloud/Cluster Aware Load Balancer



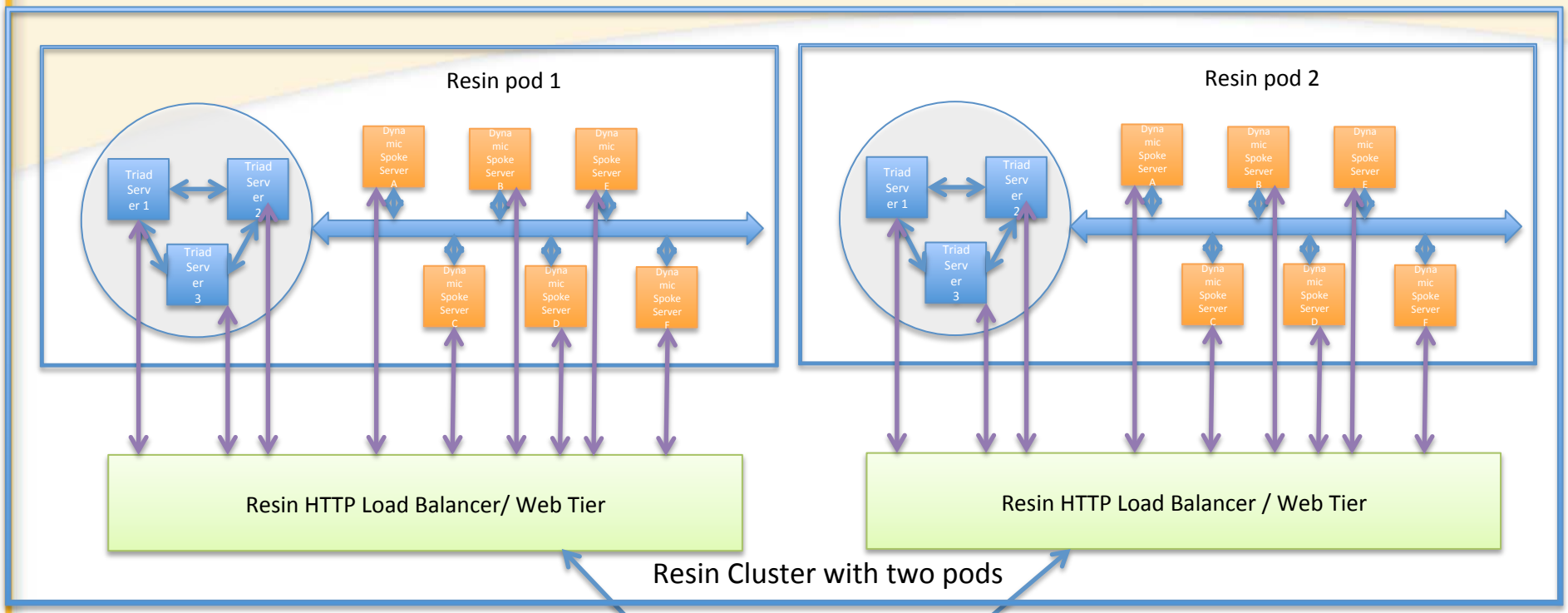
resin®

CAUCHO

Pods for Mass scale out

- If you never plan on using more than 64 servers, don't think about pods
- By default a cluster contains 1 pod
- A pod is a collection of up to 64 servers
- A pod has 1 triad (of up to 3 servers) and up to 61 to 63 static or dynamic servers
- Once you go beyond 64, you have to use this configurations <cluster …><pod><server…> … </pod>
  - Essentially divide servers up
- Below 64 you use this <cluster …><server> … </cluster>
- This is for cluster communicatoin responsiveness.
- Each cluster can have up to 64 pods

Resin Cloud Aware Load Balancer Massive Scale Out



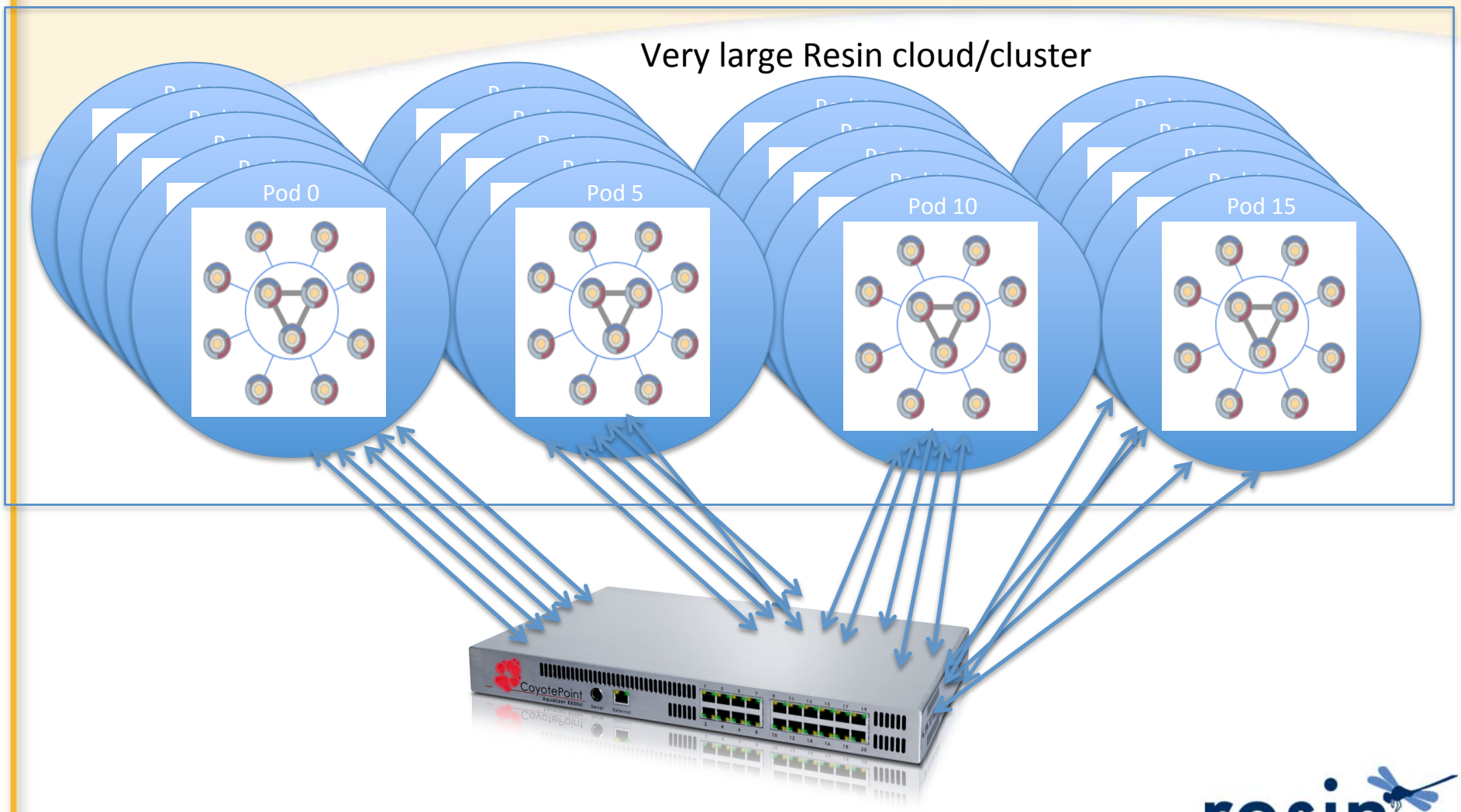
Hardware load balancer
With sticky session support

If Hardware Load Balancers
Supports 16 Servers.
And, Each Server can be LB for
Cluster with 64 servers
Then 16 * 64 = 1024 servers
Limitation of Cluster 4096 servers 64 * 64

resin®

CAUCHO

Massive scale out: a 20 pod system



resin®

CAUCHO

Load Balancer basics

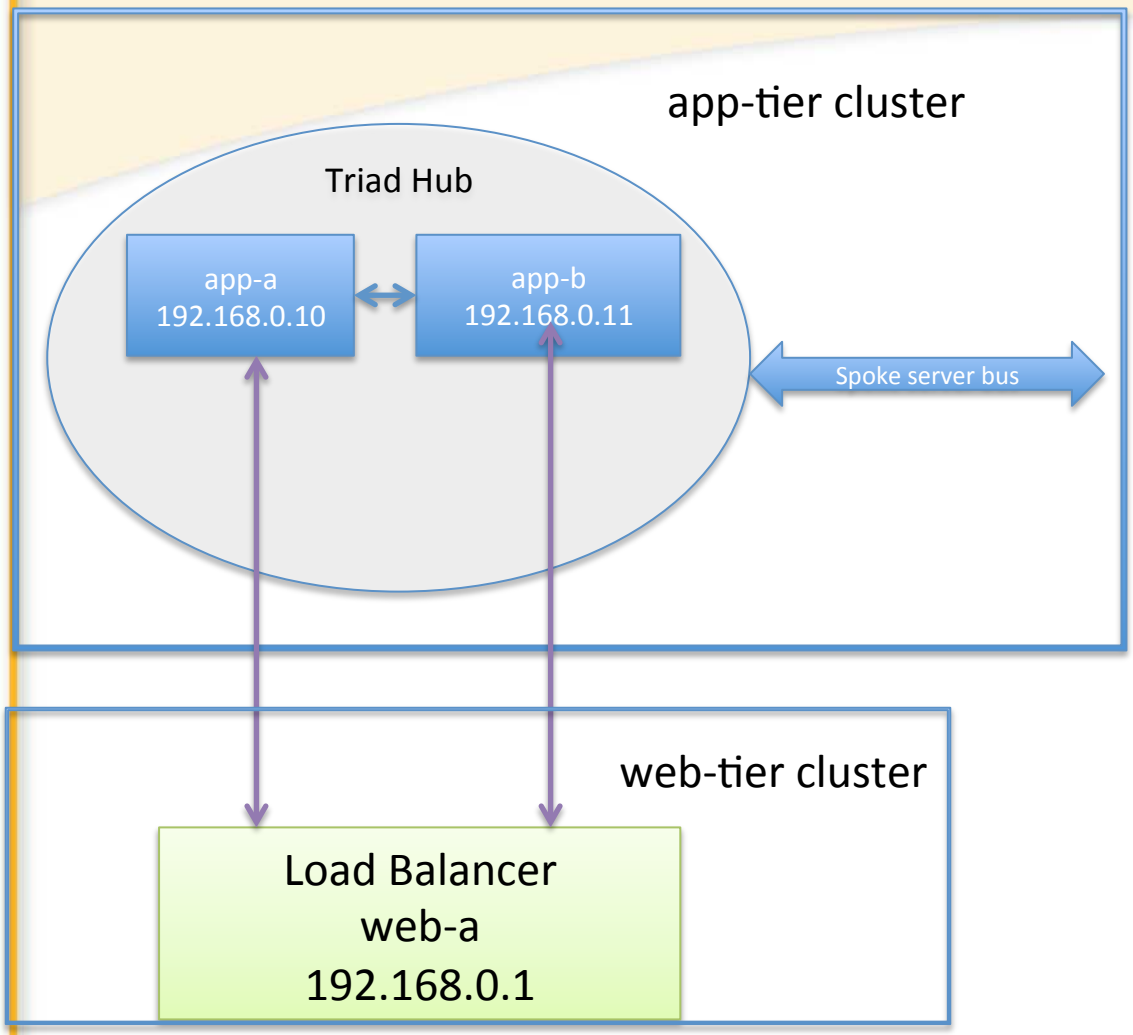
- Define servers to use as application-tier
- Define servers in web-tier, and select which requests are forwarded (proxied) to the backend app-tier servers
- Since clustering configuration already defined, only one additional configuration used forward request to app tier

<resin:LoadBalance>.

Sample Load Balancer Config

```
<cluster id="app-tier">
  <server id="app-a" address="192.168.0.10" port="6800"/>
  <server id="app-b" address="192.168.0.11" port="6800"/>
</cluster>

<cluster id="web-tier">
  <server id="web-a" address="192.168.0.1" port="6800">
    <http port="80"/>
  </server>

  <proxy-cache memory-size="256M"/>

  <host id="">

    <resin:LoadBalance regexp="" cluster="app-tier"/>

  </host>
</cluster>
```

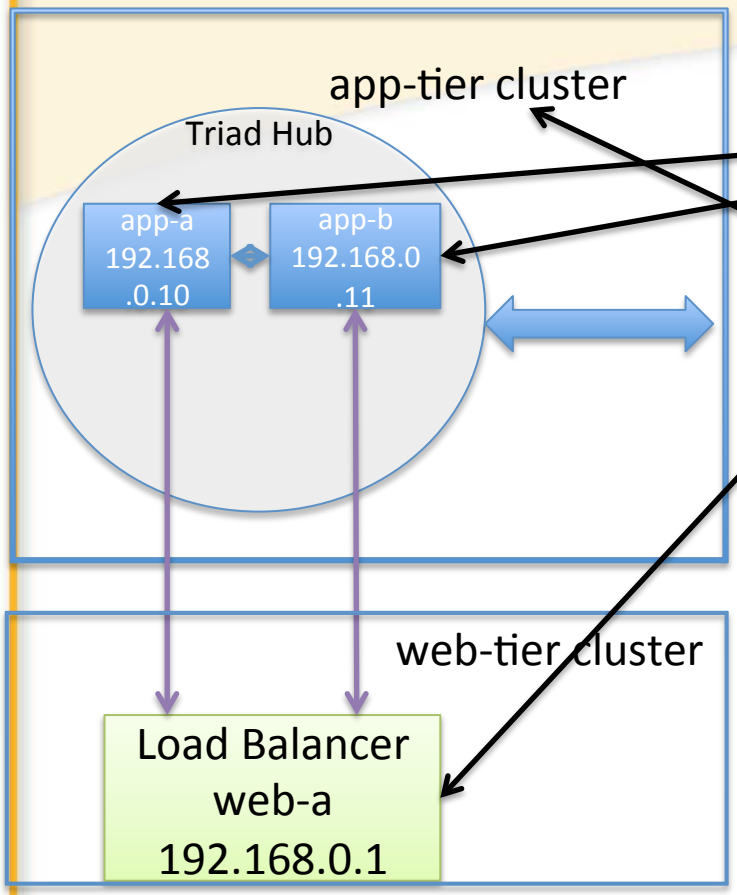
Load Balancer Setup



resin®

CAUCHO

Load Balancer Setup




```
<cluster id="app-tier">
  <server id="app-a" address="192.168.0.10" port="6800"/>
  <server id="app-b" address="192.168.0.11" port="6800"/>
</cluster>

<cluster id="web-tier">
  <server id="web-a" address="192.168.0.1" port="6800">
    <http port="80"/>
  </server>

  <proxy-cache memory-size="256M"/>

  <host id="">

    <resin:LoadBalance regexp="" cluster="app-tier"/>

  </host>
</cluster>
```

```
192.168.0.10> bin/resin.sh -server app-a start
192.168.0.11> bin/resin.sh -server app-b start
192.168.0.1> bin/resin.sh -server web-a start
```

resin®

CAUCHO

Sticky/Persistent Sessions

- Sticky session means load balancer has session affinity per server, i.e., if you start your session on one server, session stays on that server
- Resin does not replicate sessions by default
- With no sticky affinity, load balancer would pick a server pseudo randomly for each request (round robin), if it did, and you logged into server-b on second request you could log into server-c and lose your login from the first request
- With session affinity, user will go to server-b as long as long as server-b did not go down, if server-b goes down, then all users on server-b would lose their session unless session replication was turned on
- With session affinity and replicated session, users would go to server-b as long as it was up, if it were down, user would get directed to new server, the new server would request the session data for that user from the Triad
- Without session affinity and with replicated sessions, sessions would be constantly copied from one server to another

Clustering sessions

- load balancing, maintenance and failover should be invisible to your users
- Resin replicate user's session data across the cluster
- When the load-balancer fails over a request to a backup server, or when you dynamically remove a server, the backup can grab the session and continue processing
- From the user's perspective, there was no failure
- To make this process fast and reliable, Resin uses the triad servers as a triplicate backup for the user's session.

Clustered session setup

```
<web-app xmlns="http://caucho.com/ns/resin">
  <session-config>
    <use-persistent-store="true"/>
  </session-config>
</web-app>
```

Clustered Session Setup for all webapps

```
<resin xmlns="http://caucho.com/ns/resin">
  <cluster>
    <server id="a" address="192.168.0.1" port="6800"/>
    <server id="b" address="192.168.0.2" port="6800"/>

    <web-app-default>
      <session-config use-persistent-store="true"/>
    </web-app-default>
  </cluster>
</resin>
```

Sessions saved

- Sessions can be backed up on every request
- Or, Sessions can be backed up when a change is detected (default)
- Sessions are not locked for replication

Cloud Optimized, DevOps Friendly

"... we initially tried Tomcat. After a series of problems, including class loading, deployment and failed shutdowns we turned to Resin Pro. ...it resolved all the issues we had with Tomcat ... we found the administration, health checking and advanced deployment options in Resin Pro (Java EE WebProfile, Java Application Server) have greatly simplified our DevOps."

-John Bruce | Director | Kinross Group Ltd

resin®

CAUCHO