Katamari

Roll Your Own Reliable, Distributed API

Katamari

- Reliability
- Routing
- Scalability
- Easily!

Motivations:

Powerset

Picture It: Powerset

- You're a small startup with a big charter.
  - "It's just redefining search as the industry knows it."
- You have the Press's attention, but how long will that last?
- We're experimenting and iterating quickly.
- Release soon!

Picture It: Powerset

- Your code does amazing things but:
  - Much of it was written before "threads" or "exceptions" were in common use.
  - Some parses require unbounded memory. (It's NP-complete, you know?)
  - Most of it is not designed to work in an embedded library mode.

You'd Better Be Able To

- Scale as you get more hardware!
- Detect misbehaving services and restart them.
- Enforce global timeouts for requests that cause long waits.
- Host multiple versions of the API for A→B testing.

Motivations:

mog.com

Picture It: mog.com

- You've got a prototype on your laptop...
  - ... but you want it on a cluster ASAP!
- You've got it on a cluster...
  - ... but now you need to move to EC2!
- You've got a great idea...
  - ... let's hope Digg.com doesn't kill you.

I've been there, and in response, I wrote **Katamari**.

What is a

Katamari?

- Distributed Service Proxy, aggregating almost anything.
- Integration layer for any kind of functional service.
- Simple Network & Local Self-Healing
- A response to 3 years of frustration that something like it didn't exist!

- Written in Erlang
- But only for the network and reliability parts, you can use any language to talk to Katamari!
- Powerset uses it for their query analysis & reformulation on their live site.

How Does It Do That?

- Katamari starts local instances of single-threaded response-loop processes on worker machines, talking to them locally using Erlang Binary Protocol.
- Katamari interrogates these processes and aggregates them into homogenous groups, then dispatches requests to them in a round-robin fashion.

Katamari Architecture

- Main Components:
  - Masters
    - Resource Pools
  - Faceplates
  - Resource Managers
    - Chassis

Masters

- The “Root” of Clusters
- Each one is an erlang node.
- They contain the cluster’s state:
  - All resources available to the cluster
  - All resources currently in use

Master Set

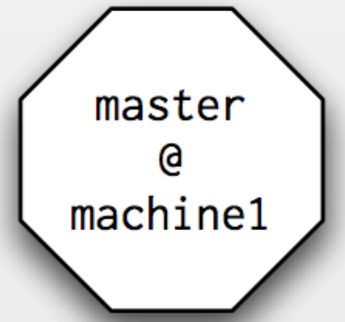
- Each Cluster has a “set” of masters.
- There isn’t just one!
- `['master@host1', 'master@host2']`
- Order matters!

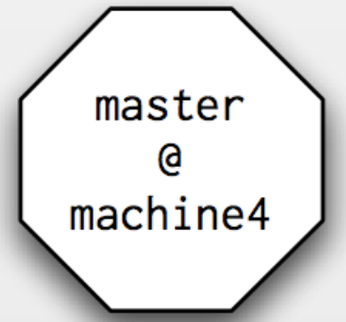
Masters

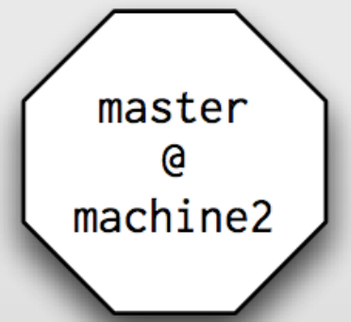
- Masters act like the “edge” of the cluster.
- Everything connects to the cluster (in both the erlang sense and the Katamari sense) by connecting to a Master.

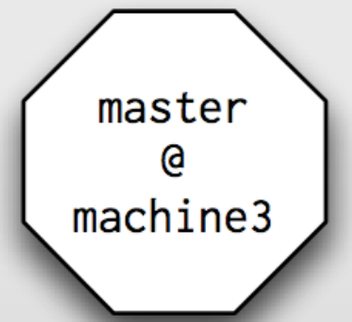
Master Election

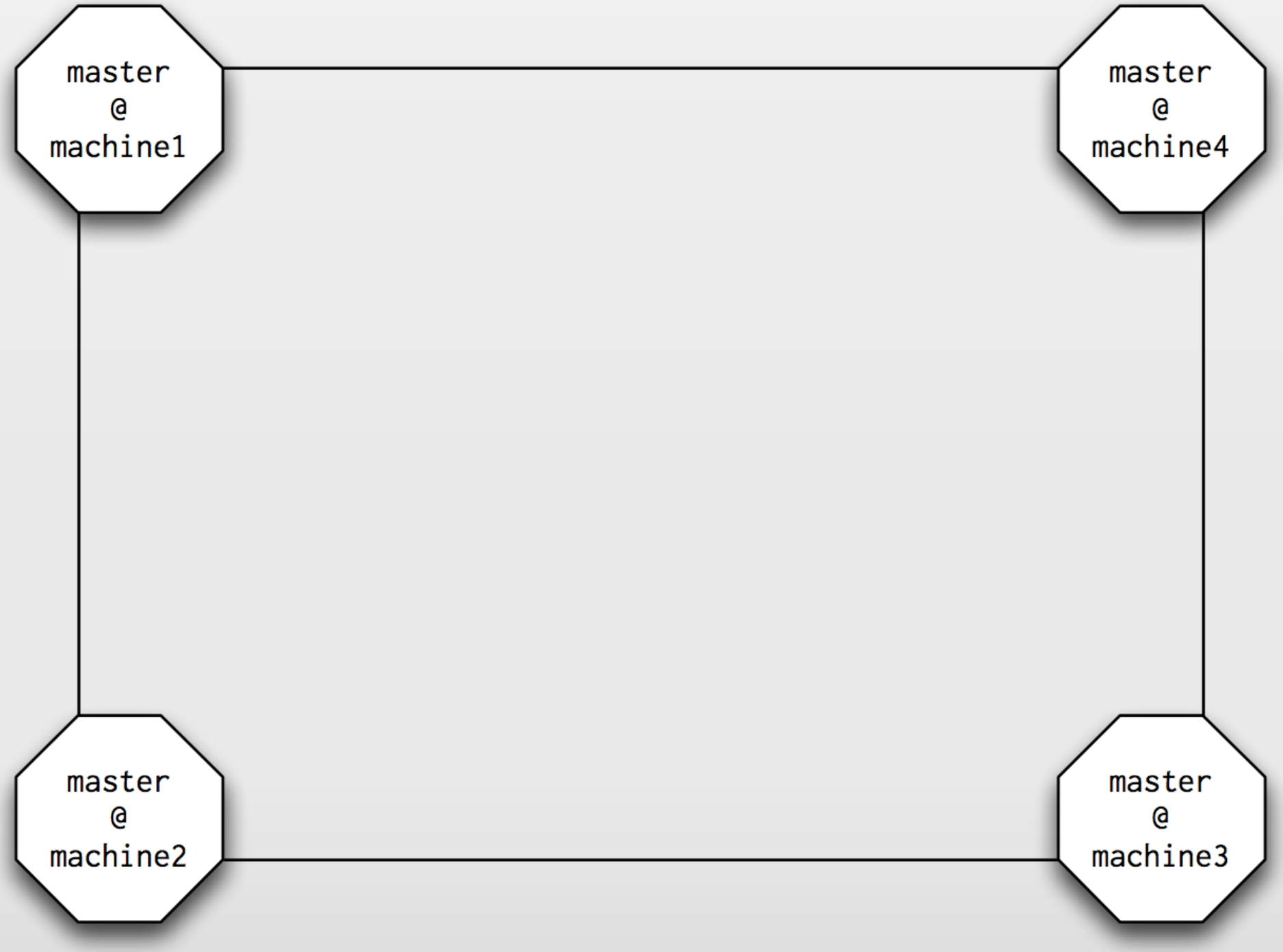
- One master is the “leader”.
- We use gen_leader2 to hold elections.
- If the master fails, an election is held to determine who should take up responsibility.
  - Priority is determine by Master List
  - Primary master has total authority.

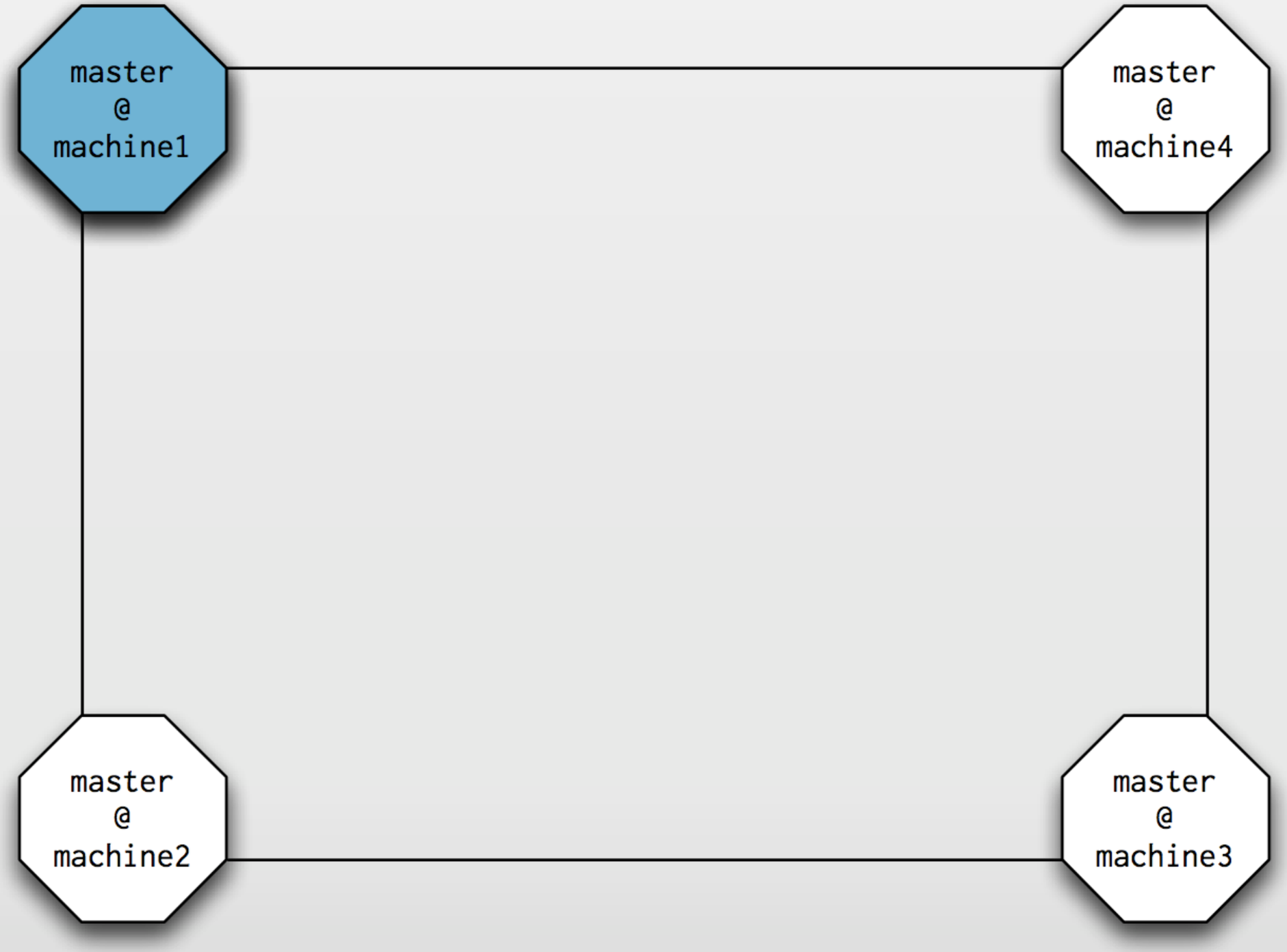


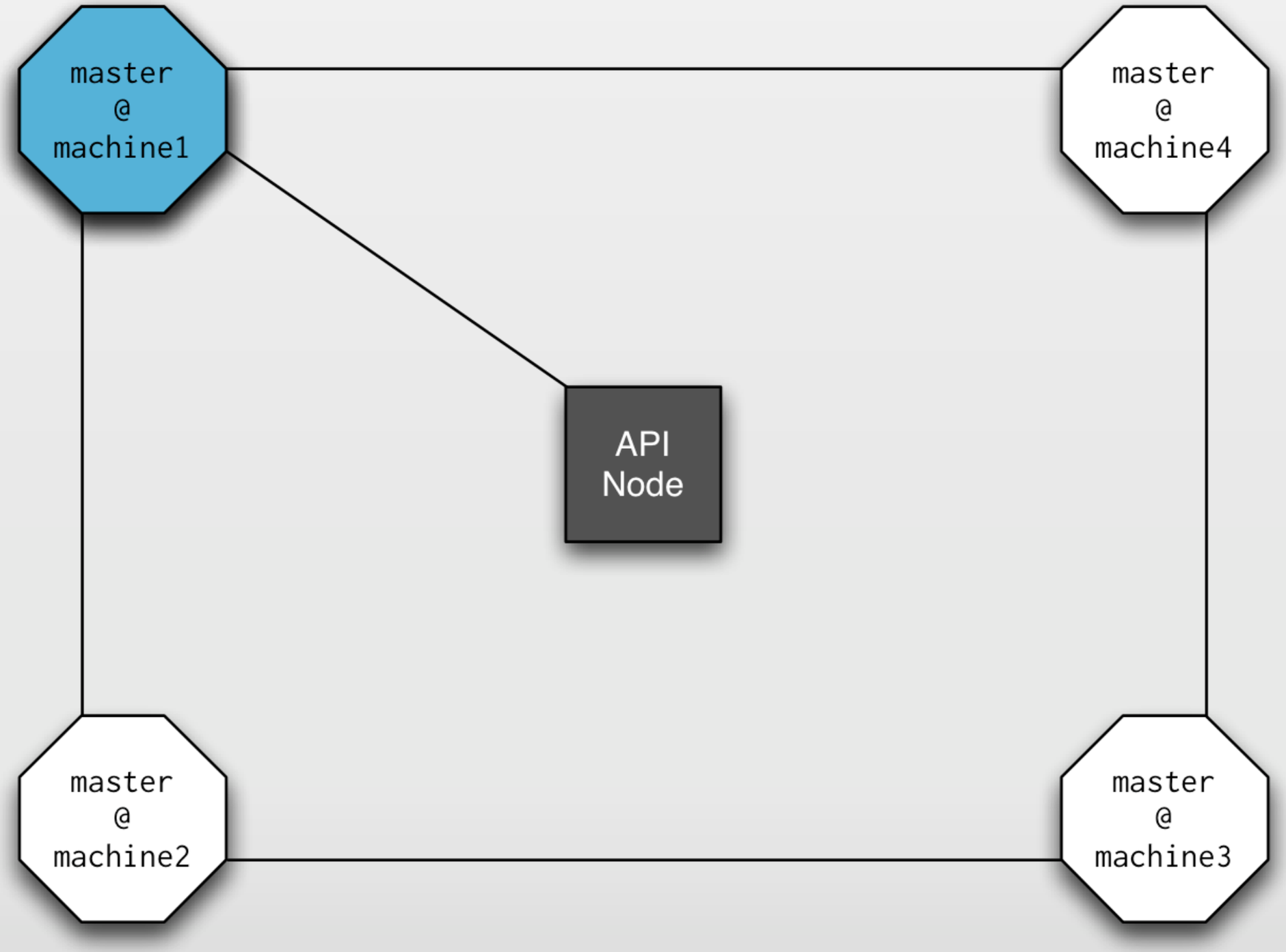


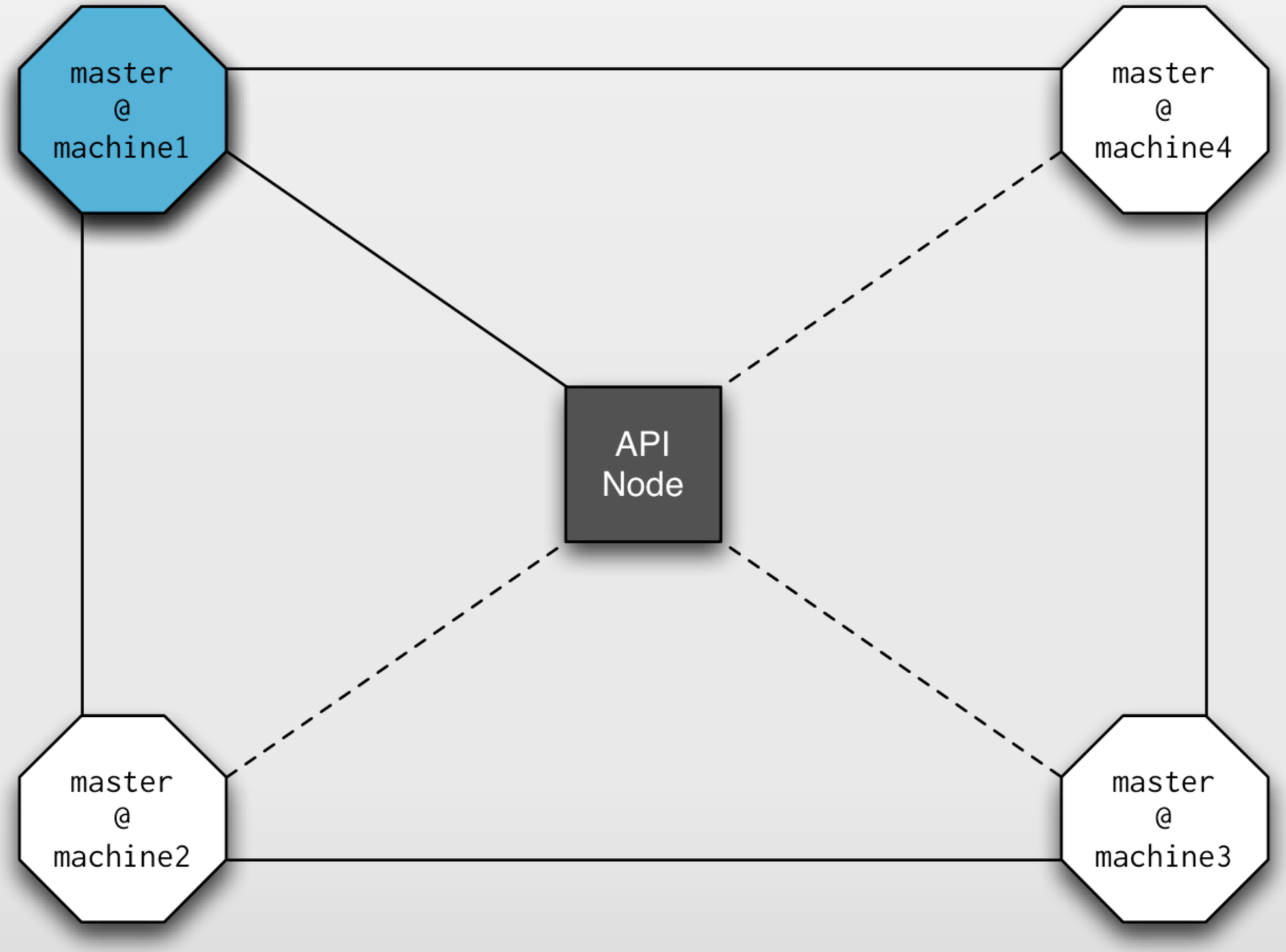












Resource Pools

- Master's pool worker nodes into homogenous pools.
- Each Master keeps an identical copy of the entire poolset.

Masters

Faceplates

- Provide a way to talk to the cluster
- References master data, but does not store it.
  - It is an observer to the gen_leader2 process.
- Any kind of server imaginable, from JSON-RPC to Thrift to raw TCP/IP.

Faceplates

- Faceplates are the endpoint of all requests into the resource cloud.
- More on this later.

Faceplates

Resource Manager

- Runs on worker machines
- Invokes your Chassis processes and introduces them to the cloud.
- Ensures that if a process becomes unresponsive or crashes, it is restarted and reintroduced.

Resource Manager: Chassis

- Ruby and C++ library for linking to Resource Manager.
- EBP Event Loop
- Trying to make it easy to incorporate any code into a Katamari cluster.

Chassis Example

```
class FakeHandler < Chassis
  kind "fake"

  handle(:echo, :textsy, :options) do |args|
    "You said: " + args[:textsy] + ' ' + args[:options].inspect
  end

  handle(:failure_to_launch) do |args|
    exit
  end

  handle(:test_arguments, :cowcat, true) do |args|
    "Arg structure was: #{args.inspect}"
  end

  handle(:cry_havok) do |args|
    return_and_exit("Let slip the dogs of war")
  end
end
```

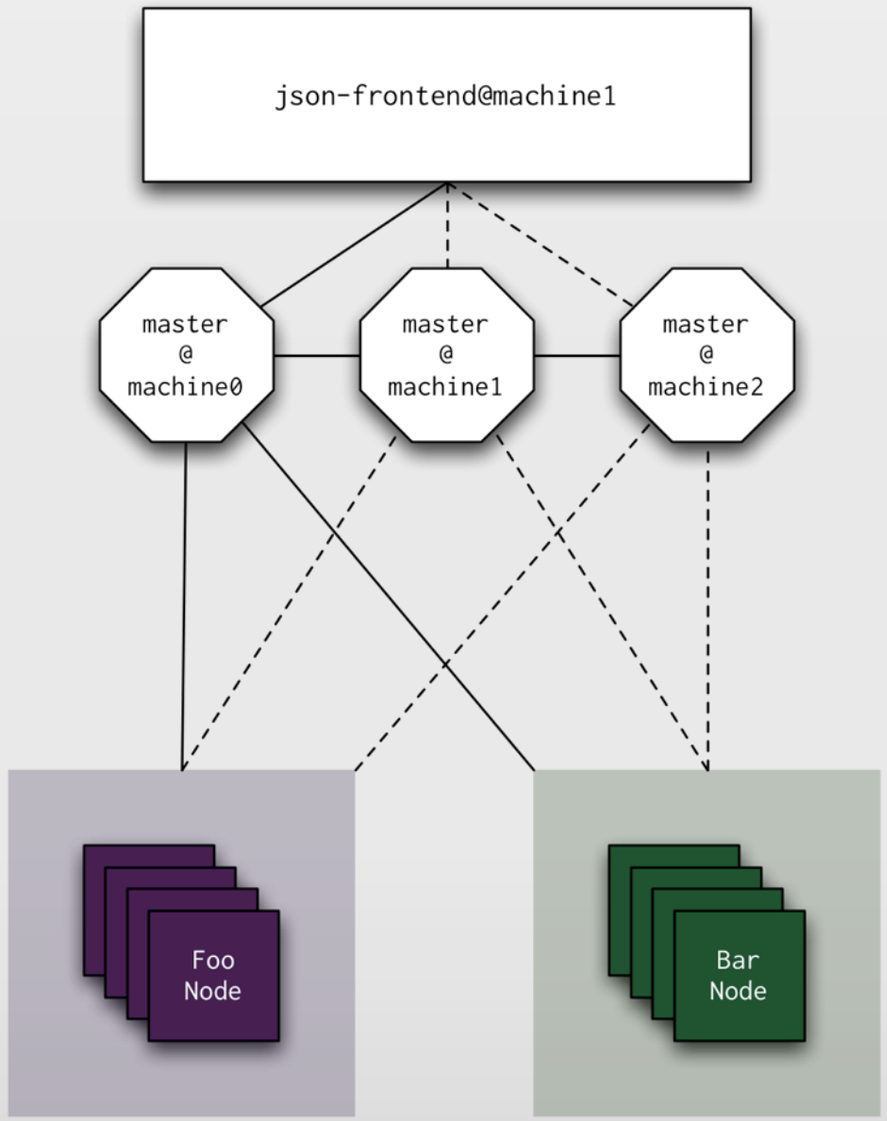
Resource

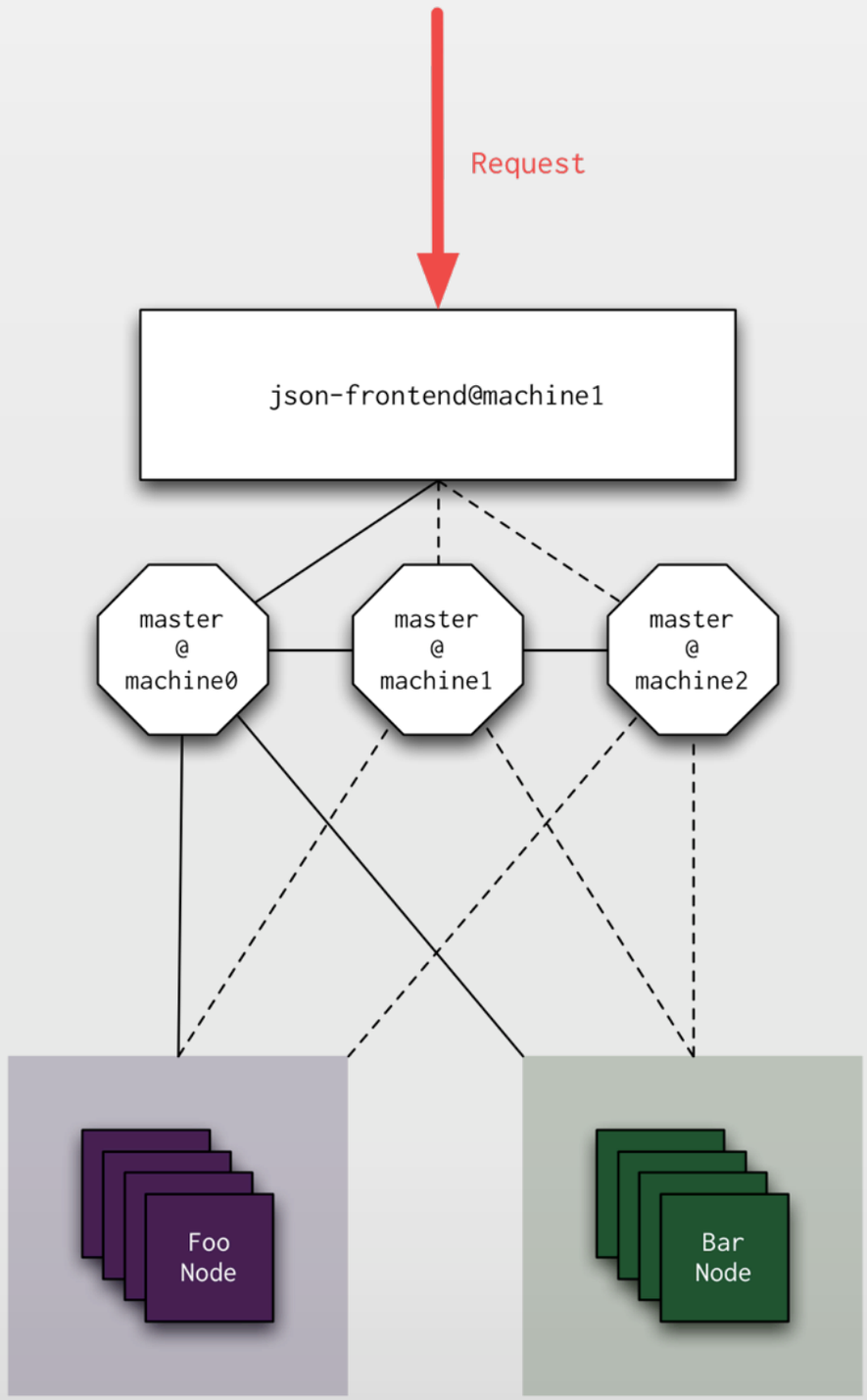
Managers

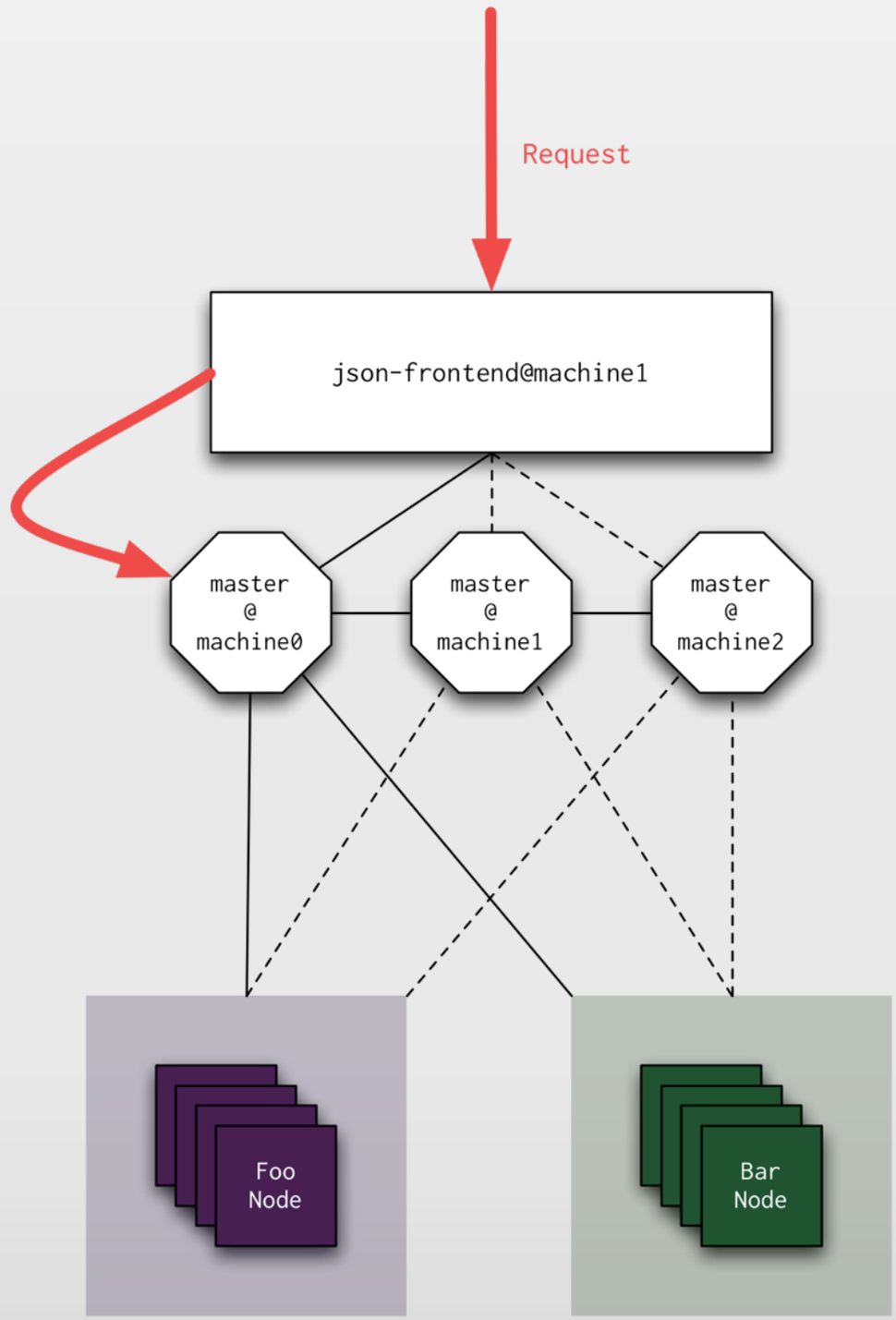
Using Katamari

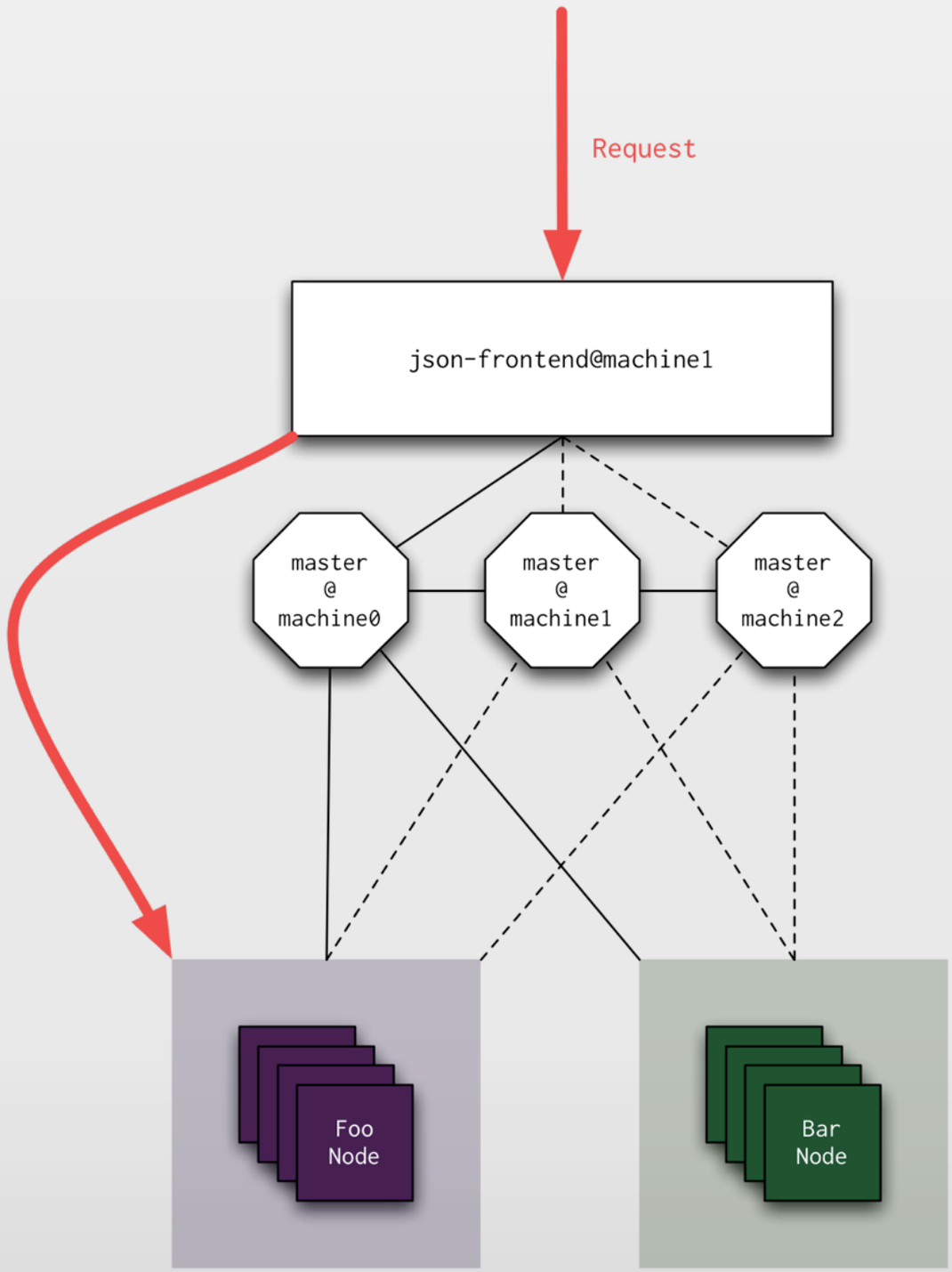
- JSON RPC faceplate.
- Thrift Faceplate
- Stats faceplate.

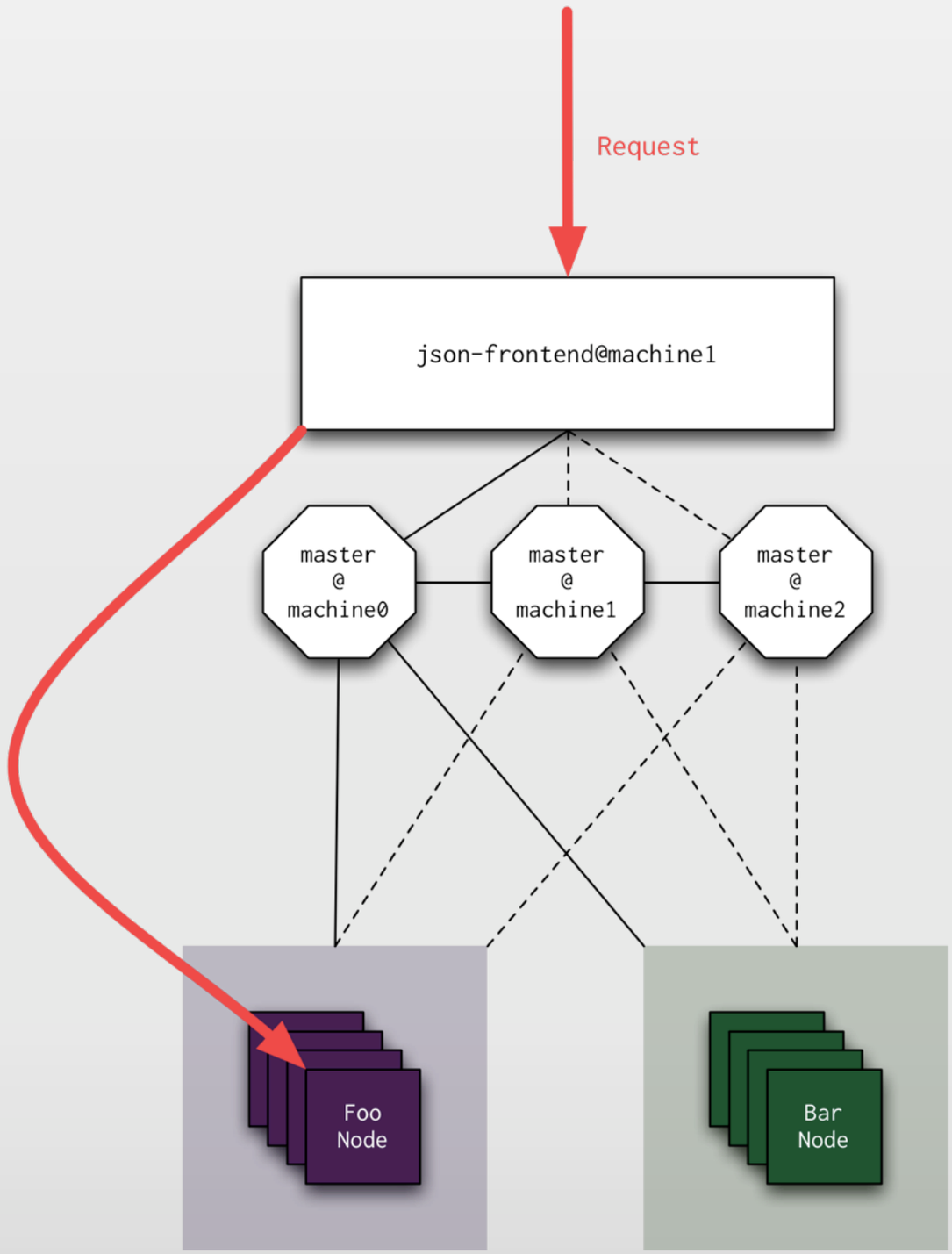
Making Requests

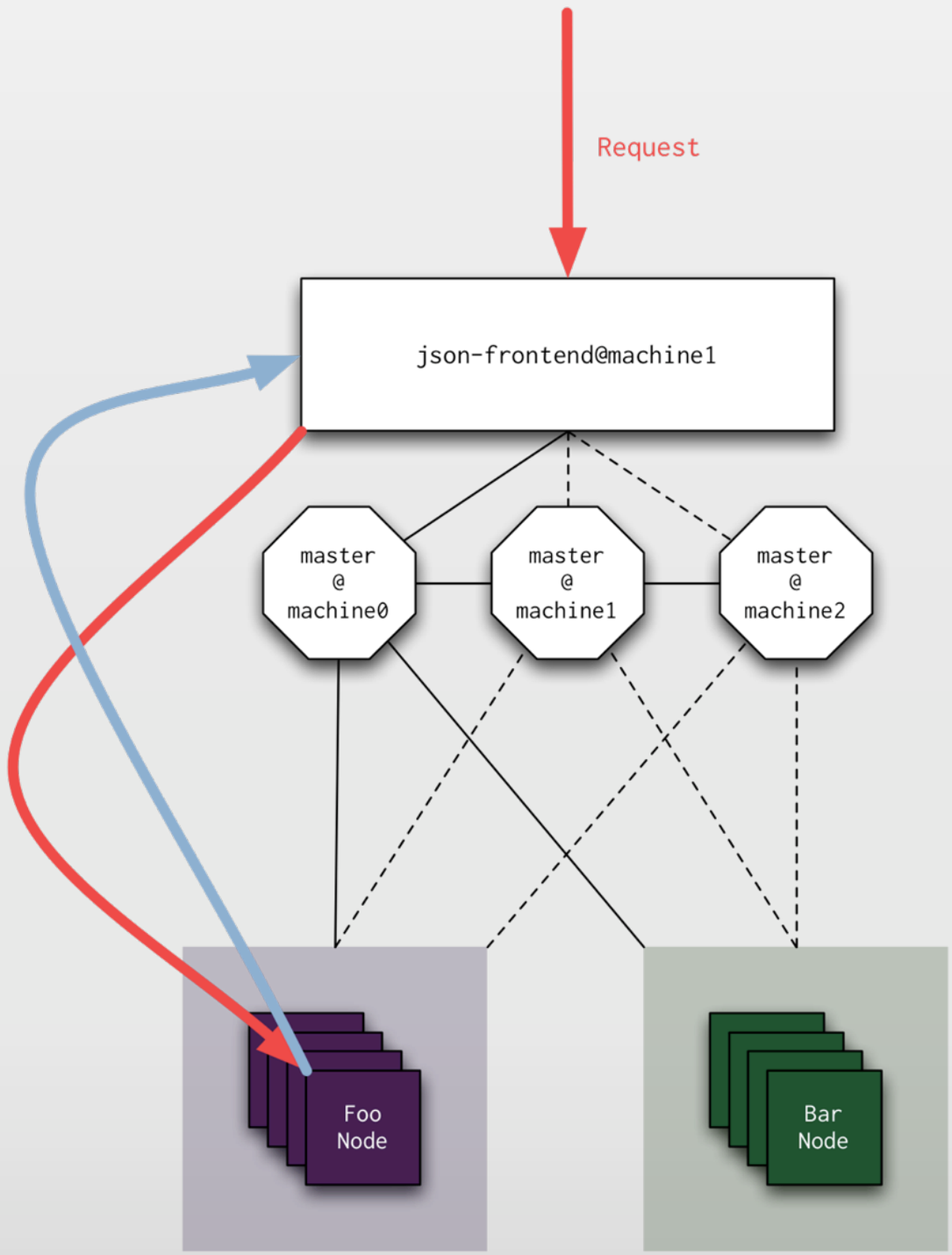


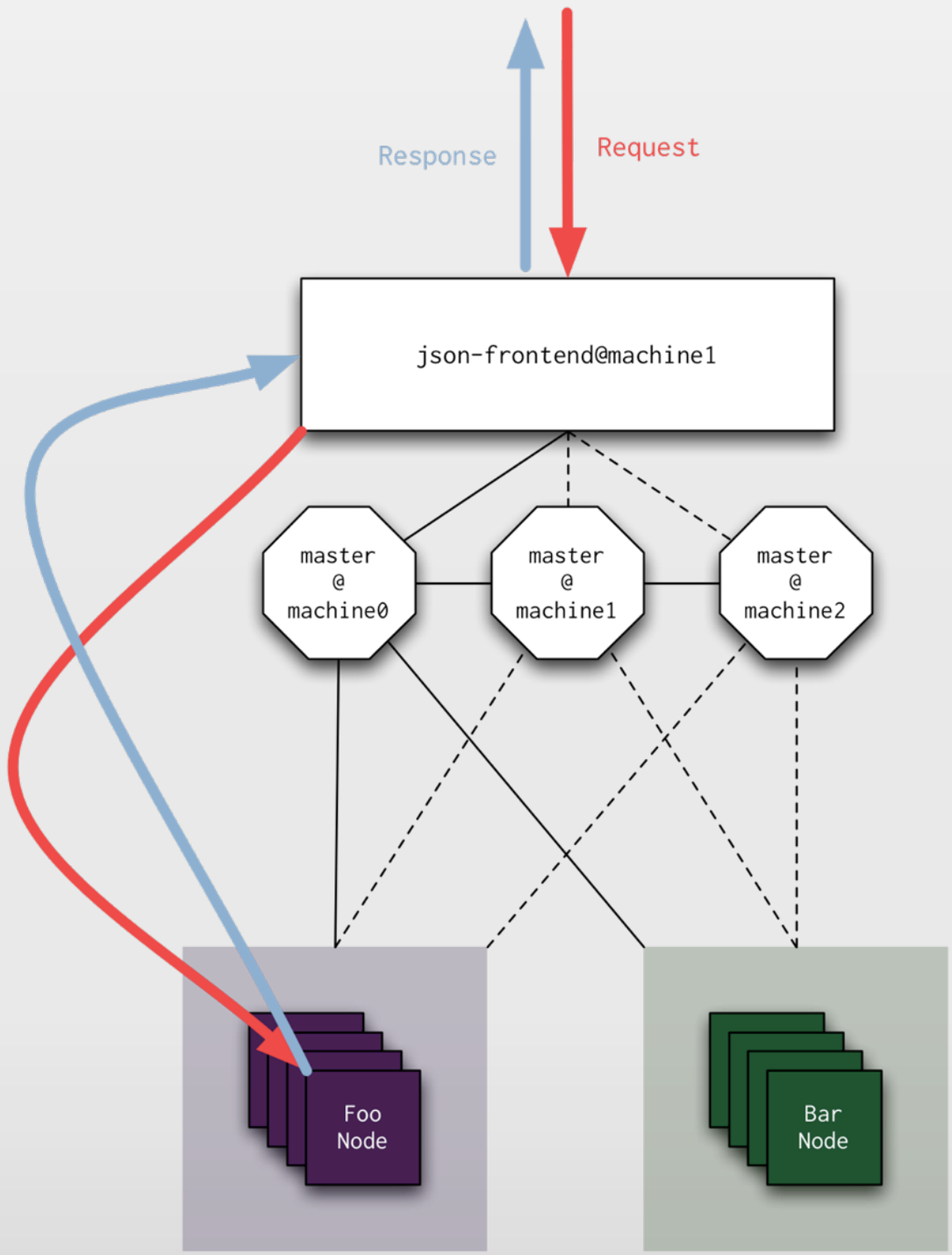












But What About…?

- What if there is no pool with details matching those of the request?
  - We signal an error.
- What happens if the pool is busy and there are no resources to use?
  - Your request waits. There is a tunable global timeout that may ultimately be reached.

Katamari In Action

- Currently In Use At Powerset
- Does all query reformulation
  - This means, we do multiple operations every time the user hits “enter” in the text box.

Katamari In Action

- Each User Query generates multiple requests to our Katamari
- (some Katamari services actually talk to Katamari to do their work)

Katamari In Action

- Katamari works well for this kind of task.
  - Multiple versions can be deployed.
  - Rack failures don't bring down the site.
  - Failure boxes brought back online tend to rejoin the cluster without instruction.

Katamari In Action

- Almost No Downtime in over a year!
  - Only real downtime was a hardware failure incident before we had multiple-masters.
  - Cluster promptly self-healed once the box was repaired.

Future Work

- Tunable Robustness via Majority Requests.
- Service to react to resource depletion.
- Automatic Provisioning!

Future Work

- Inclusion of a key-value data store.
  - Caching
  - Reliable storage for metrics
  - Intermediate Storage For...

Future Work

- Map-Reduce system.
  - “Erlojel”

Open Source

- http://github.com/KirinDave/fuzed

Open Source

- Working on making Katamari's development transparent and Open Source.
- Just a matter of protocol.

Want to know more?

- Come talk to us.
- dfayram@microsoft.com / abhay.kumar@microsoft.com
- @KirinDave / @abhaykumar on Twitter

Thank You.