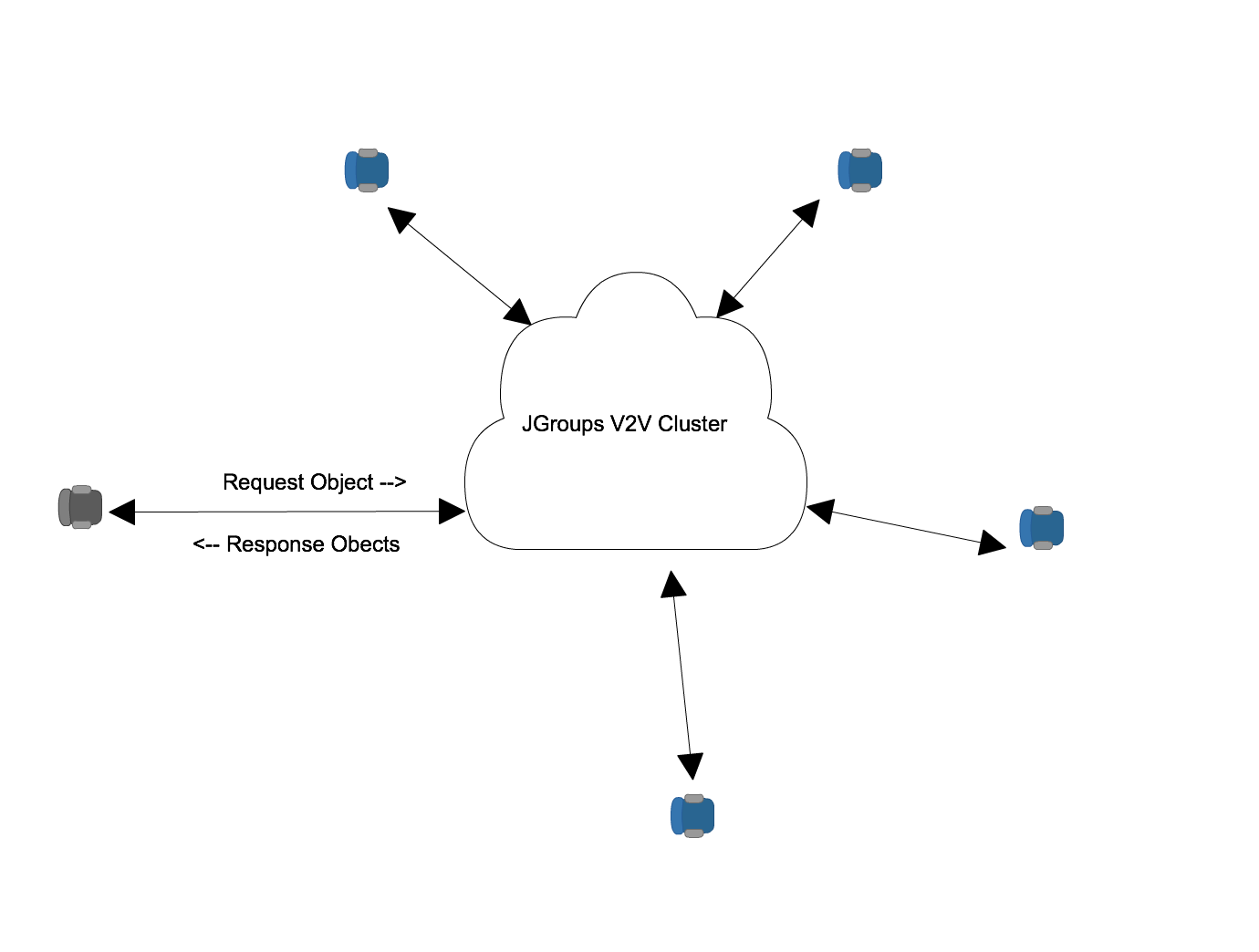
**System Design**

Key to design

* Distributed Querying API
* Distributed Processing API
* Platform independent- runs on a JVM
* APIs can be individually deployed as they use different stacks

**Distributed Querying API Design**

* Enable querying of vehicles within proximity for information
* Use-cases such as in-vehicle entertainment, emergency response and traffic safety.



* Querying mode  
  Primary use-case and useful when a vehicle need to get information from vehicles around it and then process it
* Informing mode  
  Faster for appropriate response  
  Inform vehicles around a vehicle of its status   
  Example “I had an accident and I have stopped at this location”

**Implementation**

**Jgroups**

* Toolkit for reliable messaging
* Create clusters of nodes
* Detection of new and departed/crashed nodes
* Point-to-multipoint and point-to-point
* Supports UDP(IP Multicasting) or TCP

**Starting the demo cluster**

*JgroupsRpc jgroupsRpc = JgroupsRpc.getInstance()  
jgroupsRpc.start();*

**Querying the cluster**

*RspList<Vehicle> rsp\_list=jrpc.dispatch(ResponseMode.****GET\_ALL****, 5000,* ***new Vehicle()****, Vehicle.****class****);*

*List<Vehicle> it= rsp\_list.getResults();*

**Informing the Cluster**

*Vehicle vehicleWithAccident = new Vehicle();  
vh.setHasAccident(true);  
jrpc.dispatch(ResponseMode.****GET\_ALL****, 5000,* ***vehicleWithAccident****, Vehicle.****class****);*

**Constraints**

* Applicability depends on the dynamics of the cloud
* Response time depends on the processing capabilities
* Response time can vary depending on number of vehicles in the cloud so there should be a manager