Global Reference & Global Garbage Collection in the Dripcast

Ikuo Nakagawa
Osaka University
Intec, Inc.
Transparent Cloud Computing Consortium

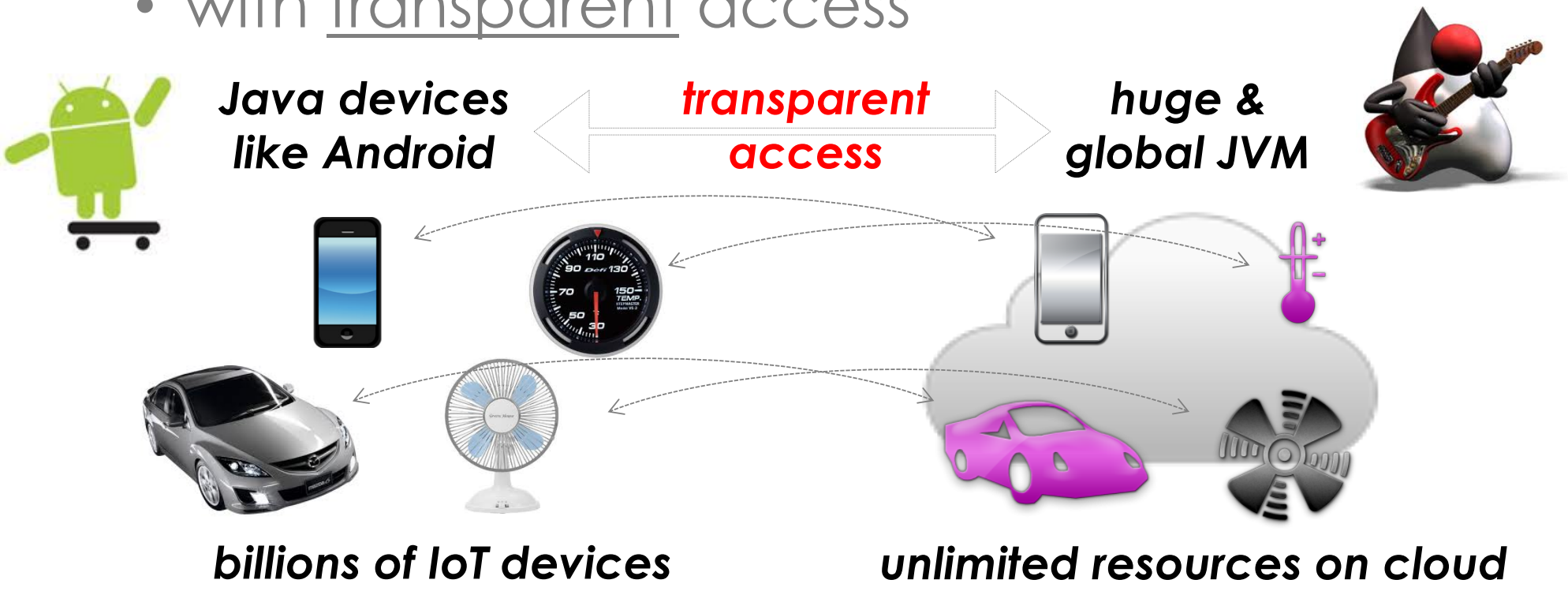
T-Cloud
Transparent Cloud Computing Consortium

Dripcast :
transparent framework for IoT apps

T-Cloud
Transparent Cloud Computing Consortium

Goal of the ***Dripcast*** project

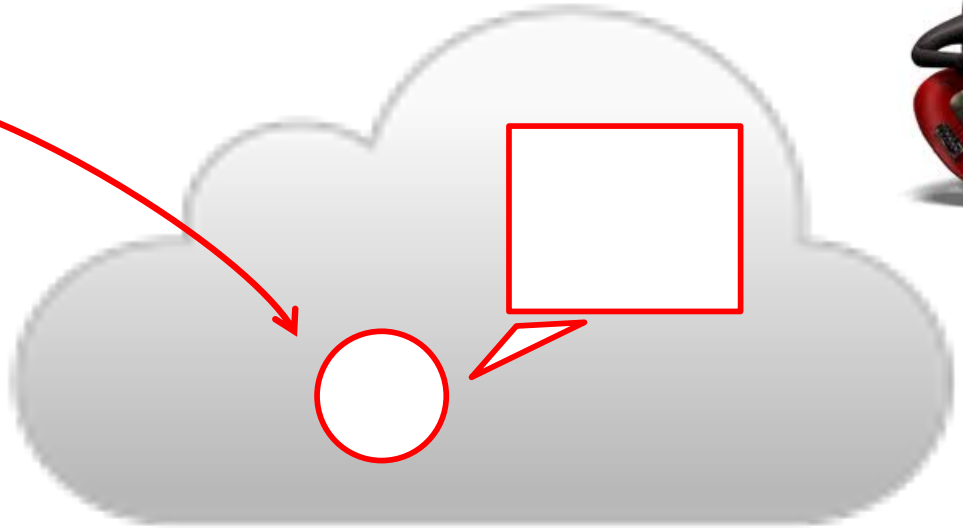
- is providing
  - cloud platform for billions of IoT devices
  - a huge global JVM (Java Virtual Machine)
  - with transparent access



Typical IoT apps.

run small app.

store & process data



Dripcast : Magic Spell

App.

```
// an instance we want to handle.
Gadget a = new GadgetImpl();




// store out value to the object, at first.
a.setValue(31);

// now, verify input against our value.
String s = in.readLine();
if (a.verifyValue(Integer.parseInt(s))) {
  System.out.println("Great! Lucky guess!");
} else {
  System.out.println("No match. Sorry.");
}
```

Interface

```
interface Gadget {
  void setValue(int val);
  boolean verifyValue(int val);
}
```

```
class GadgetImpl implements Gadget {
  private int value = 0;
  public void setValue(int val) {
    value = val;
  }
  public boolean verifyValue(int val) {
    return value == val;
  }
}
```

Data + Method

Let's make the object to be cloud enabled

Dripcast : Magic Spell (Cont'd)

App.

Try, 3 minutes hacking...

Interface

```
// an instance we want to handle.
Gadget a = new GadgetImpl();

// Biddidi-Bobbidi-Boo.
a = d.create(Gadget.class, a);


// store out value to the object, at first.
a.setValue(31);

// now, verify input against our value.
String s = in.readLine();
if (a.verifyValue(Integer.parseInt(s))) {
  System.out.println("Great! Lucky guess!");
} else {
  System.out.println("No match. Sorry.");
}
```

Store into the cloud

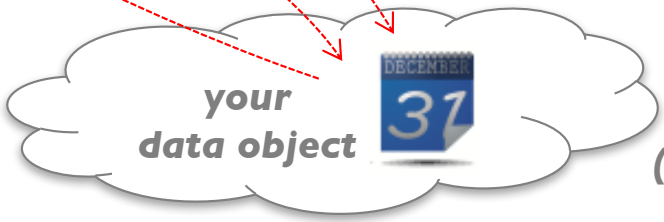
RPC (Invocation) and get result

```
interface Gadget {
  void setValue(int val);
  boolean verifyValue(int val);
}
```

```
class GadgetImpl implements Gadget {
  private int value = 0;
  public void setValue(int val) {
    value = val;
  }
  public boolean verifyValue(int val) {
    return value == val;
  }
}
```

Data + Method

Android device (application)

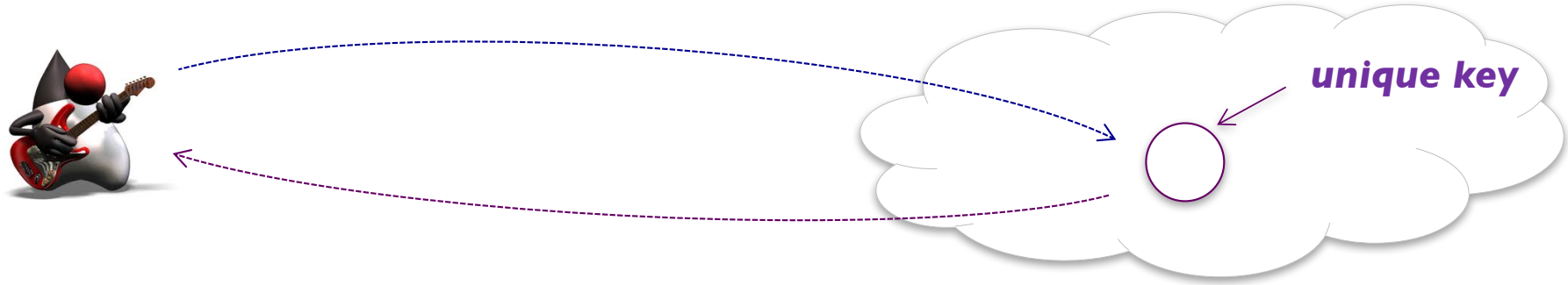


Cloud (platform)

Very easy, No care about communication, server-side nor RDB

What we do, in ***Dripcast***

**unique "Key" to identify the Java object on cloud,
and use "Proxy" for transparent method calls**



Client side Java code :

On method calls, create associated ***job*** object,
with { ***key***, ***method***, ***args*** },
transmit the ***job*** to the cloud,
and await for the ***result***
return the ***result***

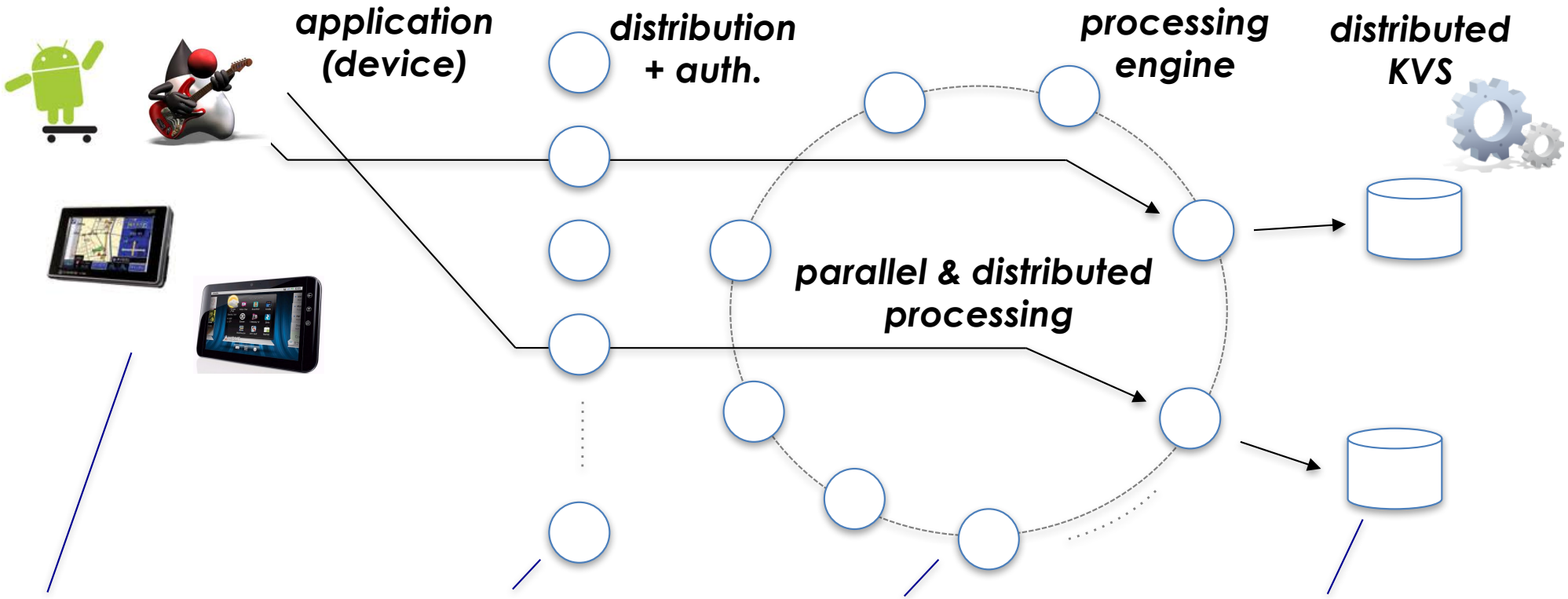
Cloud side Java invocation :

when cloud gets ***job***
lookup ***target*** object from ***key***,
lookup method for ***method*** with ***args***,
invoke the method with ***args***, and
return the ***result***

**attached Proxy object
in client for the key**

**real Java object on cloud
with unique key**

Dripcast : End-to-end Architecture

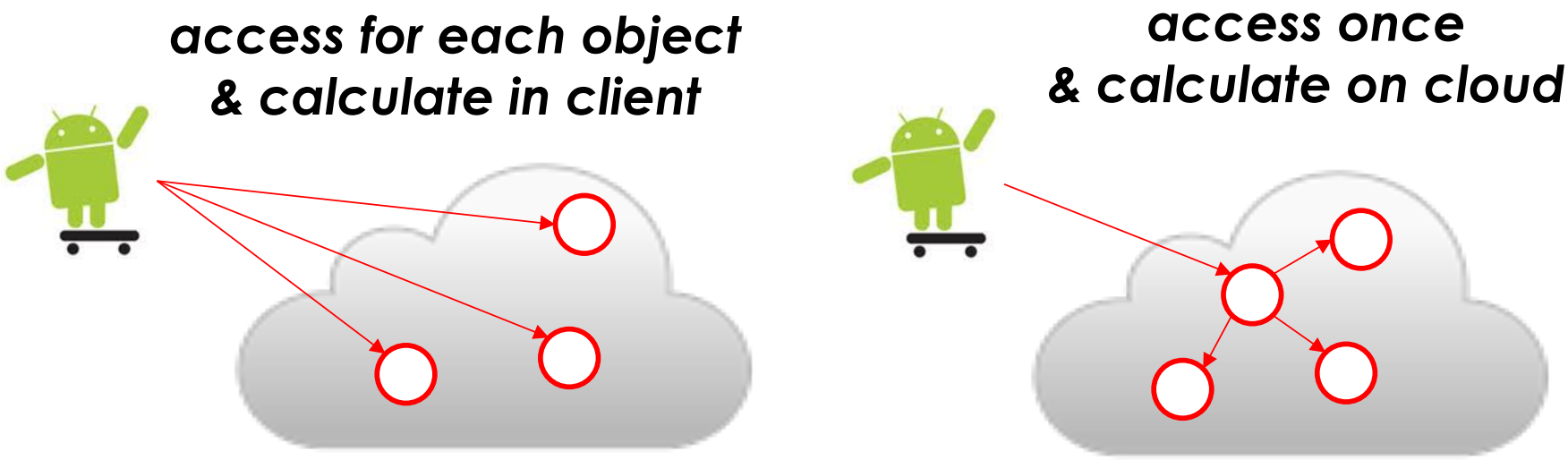


Client / Device	***Relay***	***Engine***	***Store***
App. on devices with SDK	Deliver. with auth. or SSO	Method invocation, on Java VM	Into physical disks, w/ replication
object-key and method call info.	***deliver to engines by object-keys***	***invoke Java method on a cloud***	***store Java object into backend***

Extensions:
Global Reference

What we talk about, today – 1/2

- Global Reference
  - which enables ***Dripcast*** to be global JVM
  - for not only client devices
  - but also for cloud objects

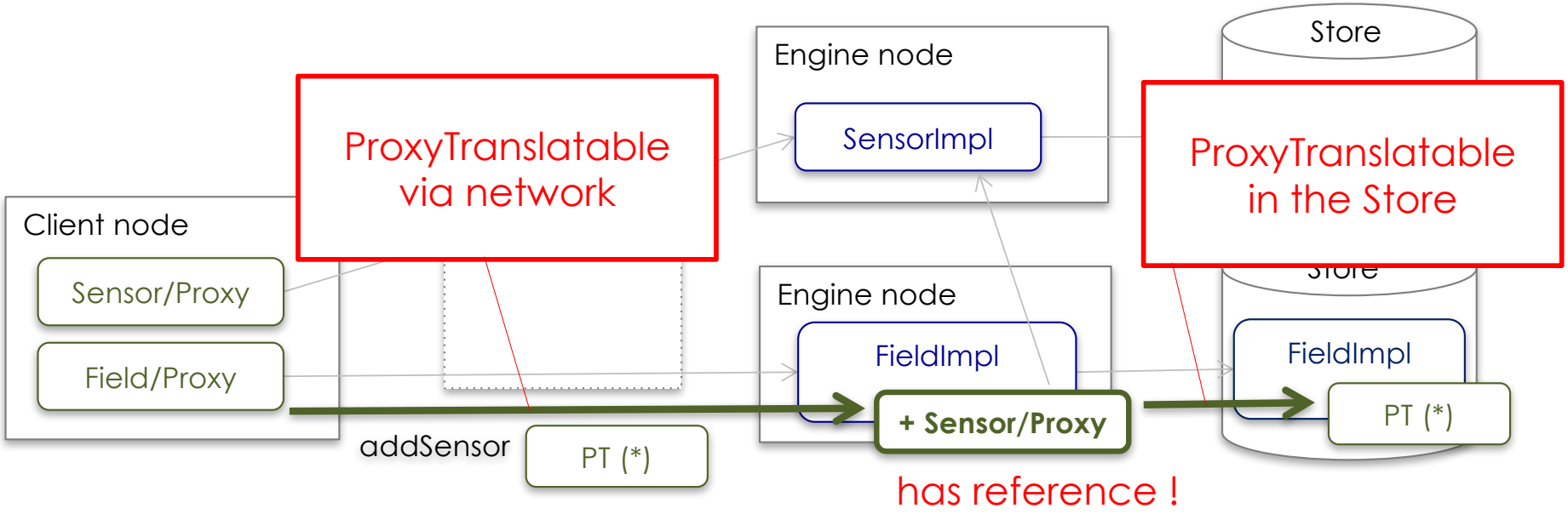


Global Reference : implementation

References in the ***Dripcast*** are represented as

Proxy in Client (was described as in-client reference)

ProxyTranslatable on network or in the Store (global reference)



(*) use writeReplace and readResolve translations

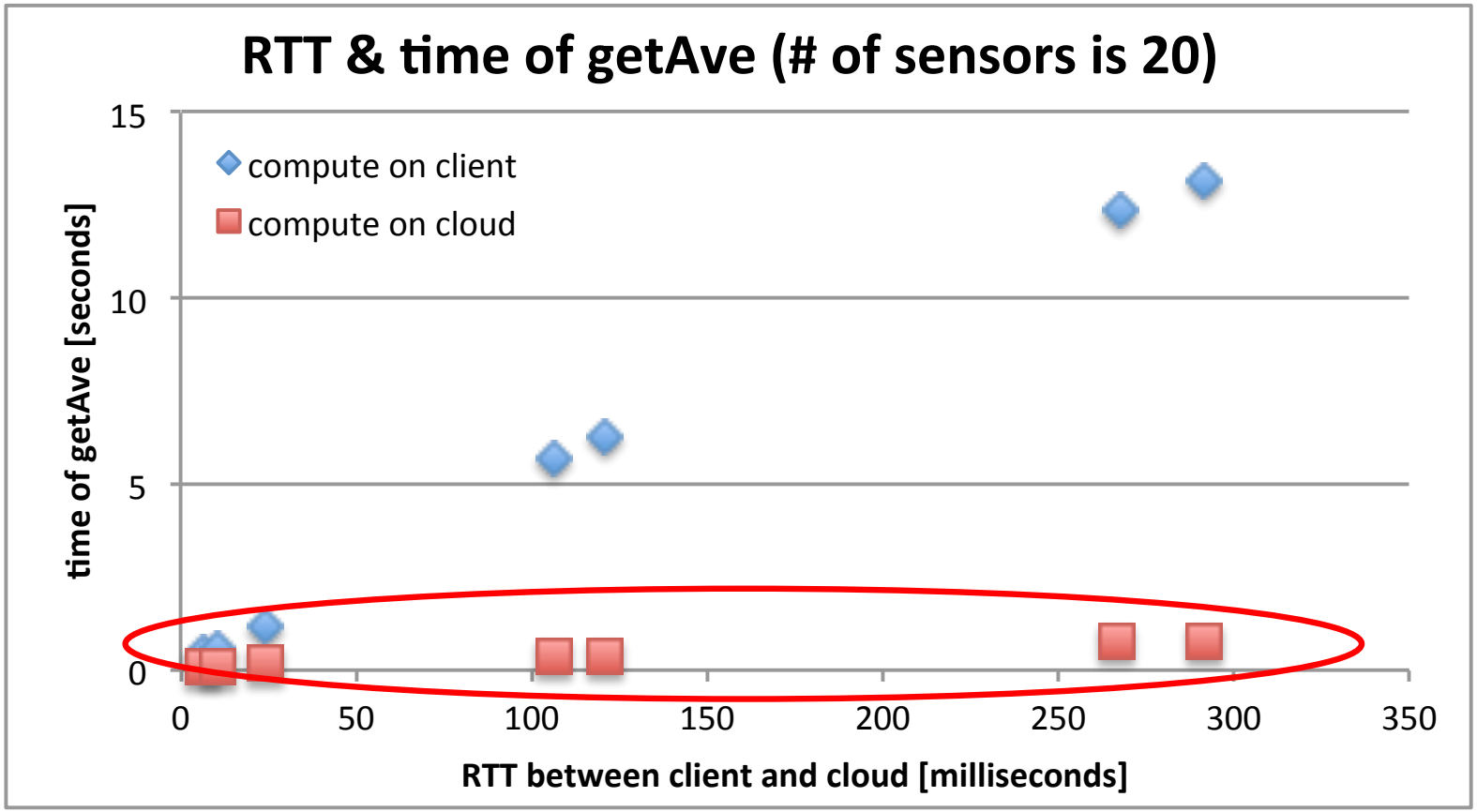
Benefit of Global Reference

- Compute on cloud
  - use computer resources on cloud
  - communication within cloud

- Scalability of computation power
  - O(N) where N is the number of Engine
  - with scale-out style computation model

- Transparency
  - Java code is still simple, for IoT apps
  - no REST/HTTP, server-side code nor RDB

Evaluation



Computation time is linear to RTT

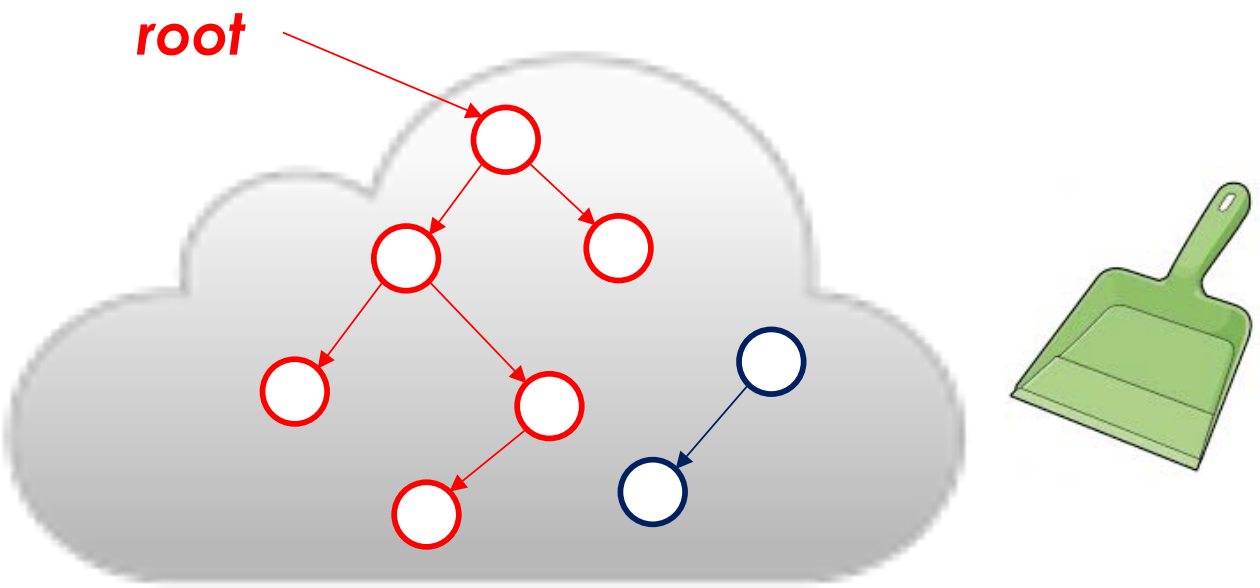
RTT is dominant factor, for "compute on client"

RTT has no impact, for "compute on cloud"

Extensions:
Global Garbage Collection

What we talk about, today – 2/2

- Global GC (Garbage Collection)
  - which is basic feature for JVM
  - was difficult to design & implement
  - in distributed computing environment

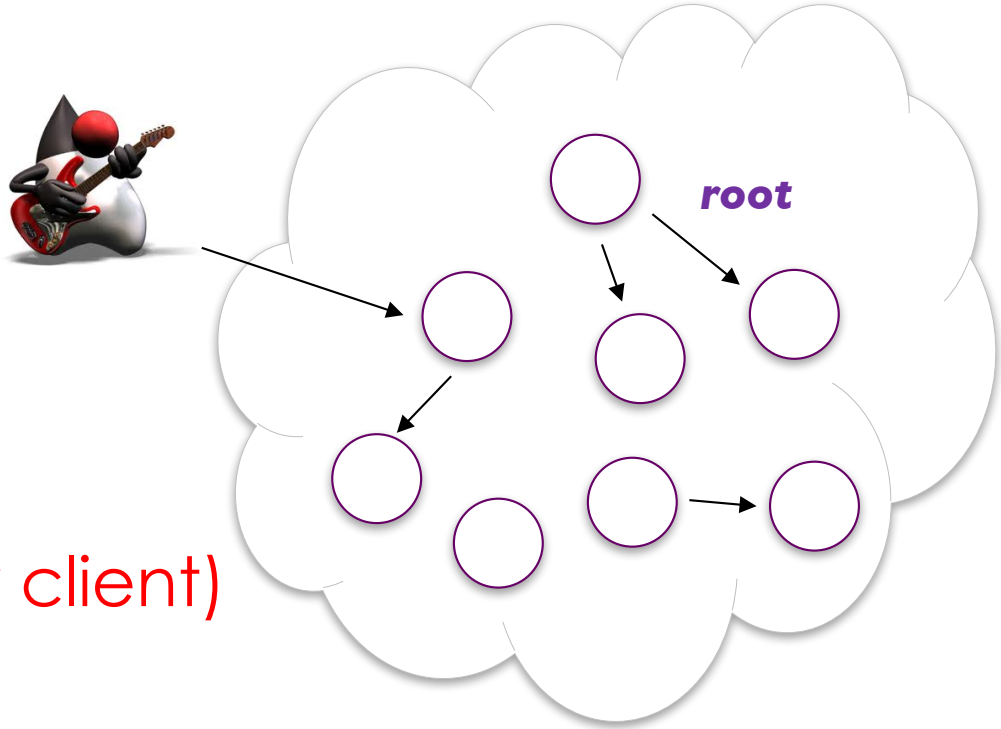


Assumption & mark rules

active if an object is referred by

the root object

⇨ mark all objects referred by root



a client

⇨ mark all fresh objects (recently accessed by client)

an active object

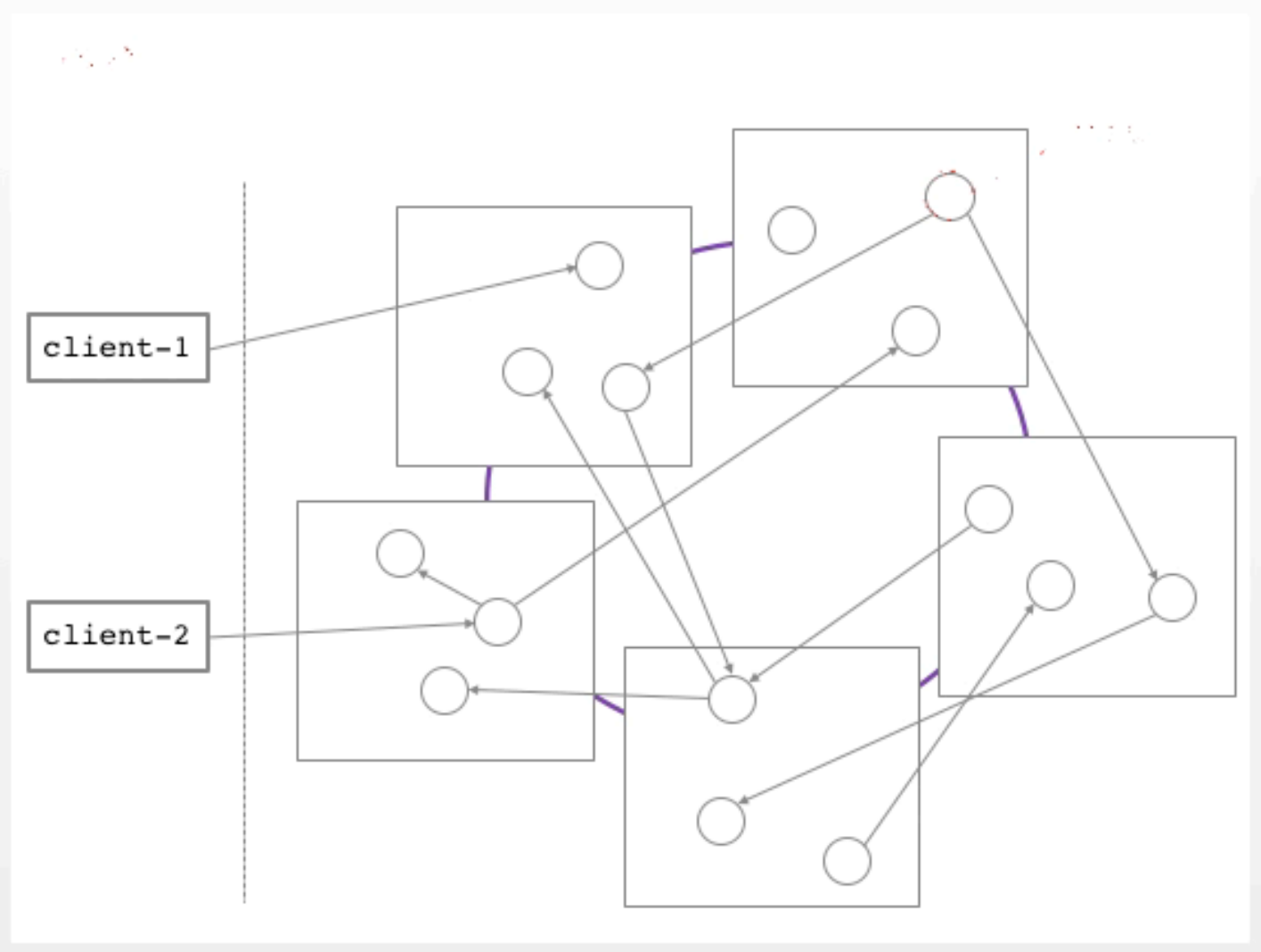
⇨ mark all objects referred by active objects

GGC algorithm, in parallel

let E := session-timeout & T_s := now

	# of threads
<u>mark</u>	
objects referred by root	1 in global
fresh objects ($t + E < T_s$) t = referred, modified by client	N in each engine
objects referred by marked objects	N in each engine
<u>sweep</u>	
remove objects without mark or last-mark-time < T_s	N in each engine

Very very very short demo of GGC



Conclusion

T-Cloud
Transparent Cloud Computing Consortium

Conclusion

We proposed extensions for the *Dripcast*,

Global References, so that
Dripcast object may have reference to others.
It enables "compute on cloud" model.

Global Garbage Collection, so that
Dripcast would remove unreferred objects
based on original mark & sweep model.

which are very important features so that

The ***Dripcast*** provides a huge global JVM (Java VM)
using unlimited computer resources on cloud.


T-Cloud
Transparent Cloud Computing Consortium