

# ConnectJS: A cross mobile platform actor library for multi-networked mobile applications

---

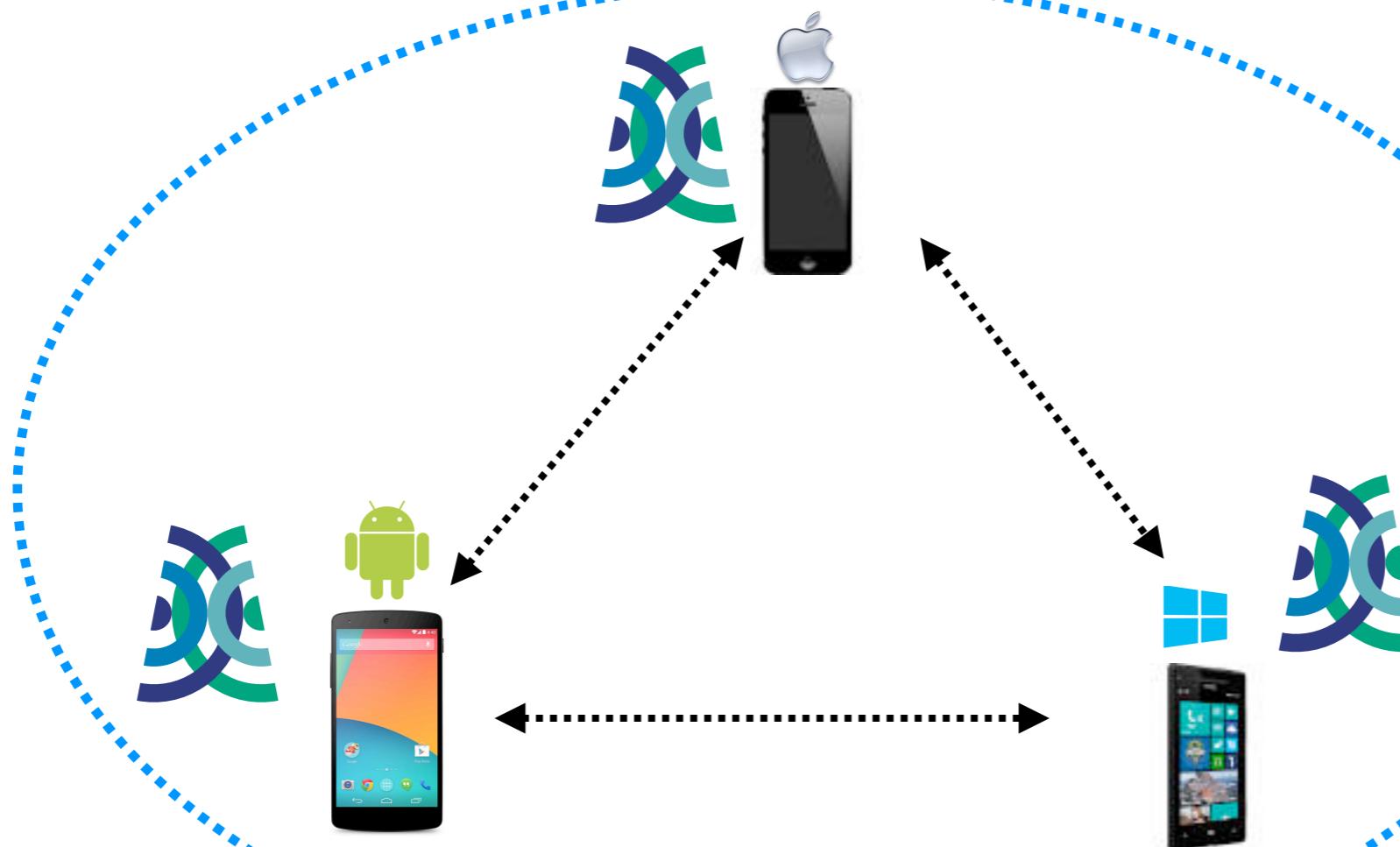
Elisa Gonzalez Boix, Christophe Scholliers, Nicolas Larrea, Wolfgang De Meuter





# Mobile Wireless Applications

Applications running on mobile ad hoc networks



wireless communication  
with volatile connections

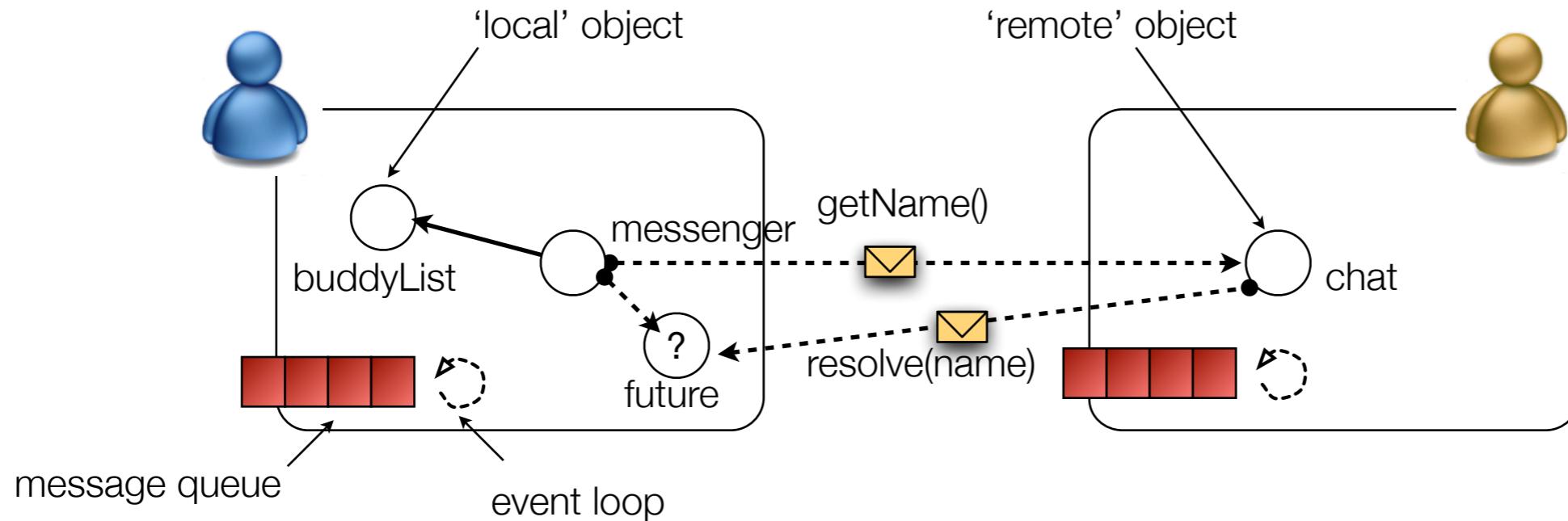


spontaneous interactions  
with little infrastructure



# Ambient-oriented Programming in AmbientTalk

- Programs are event loops that communicate **asynchronously**.
- Far references **tolerate** network failures.

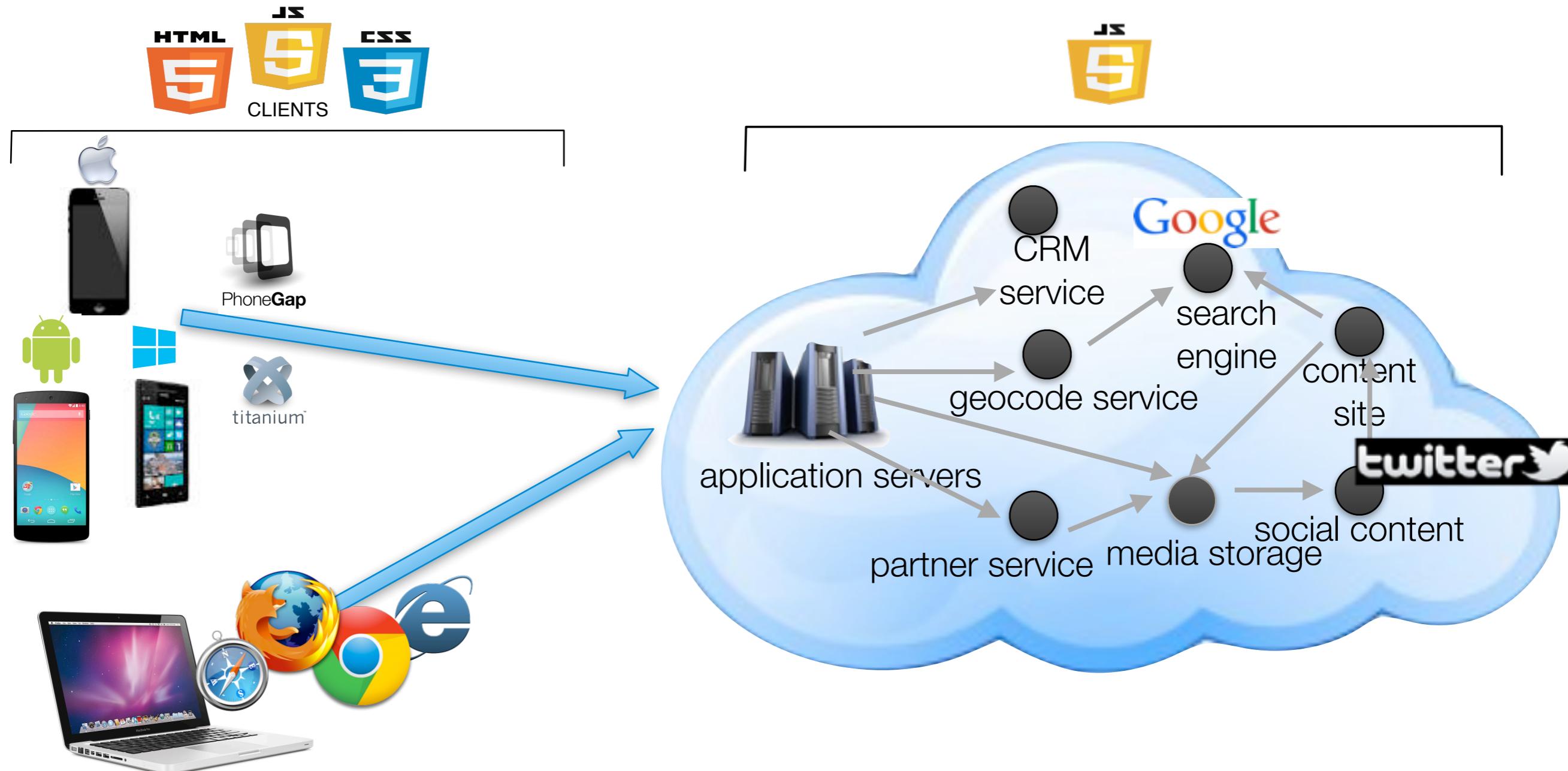


```
deftype InstantMessenger;
whenever: InstantMessenger discovered: {
    |messenger|
    def future := messenger->getName();
    when: future becomes: {
        |name|
        buddyList.put(name, messenger);
        display("Added buddy:" + name);
    }
}
```

```
deftype InstantMessenger;
export: chat as: InstantMessenger;
```

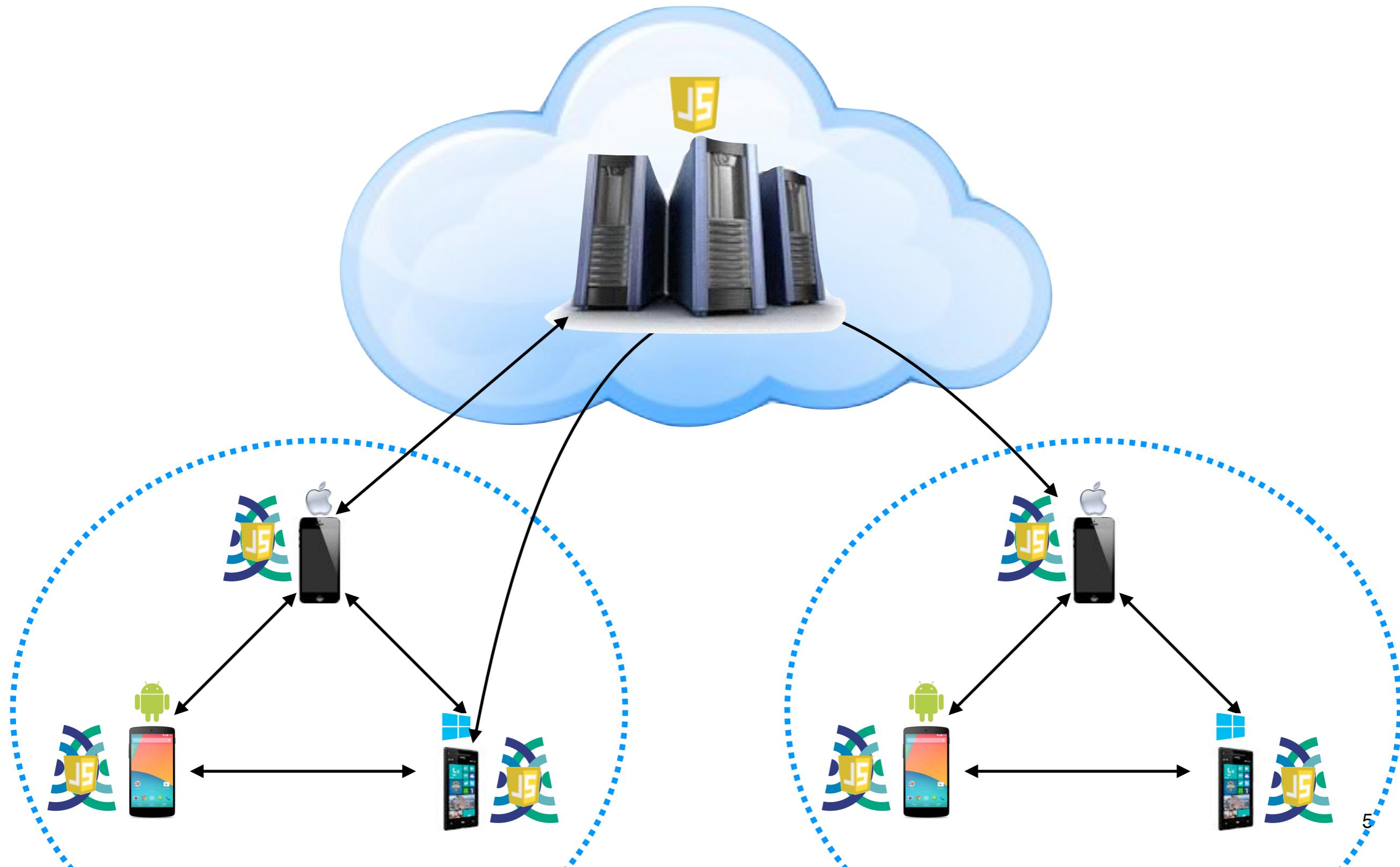


# Cloud-based Applications





# Rich Mobile Applications





# Programming Rich Mobile Applications

---

- How to ease the development of mobile applications that communicate over multiple networking interfaces?
- How to alleviate the pyramid of doom?



# When the web meets mobile computing ...

- makes it possible to create mobile applications using a single code base ( i.e. JavaScript) instead of device-specific languages

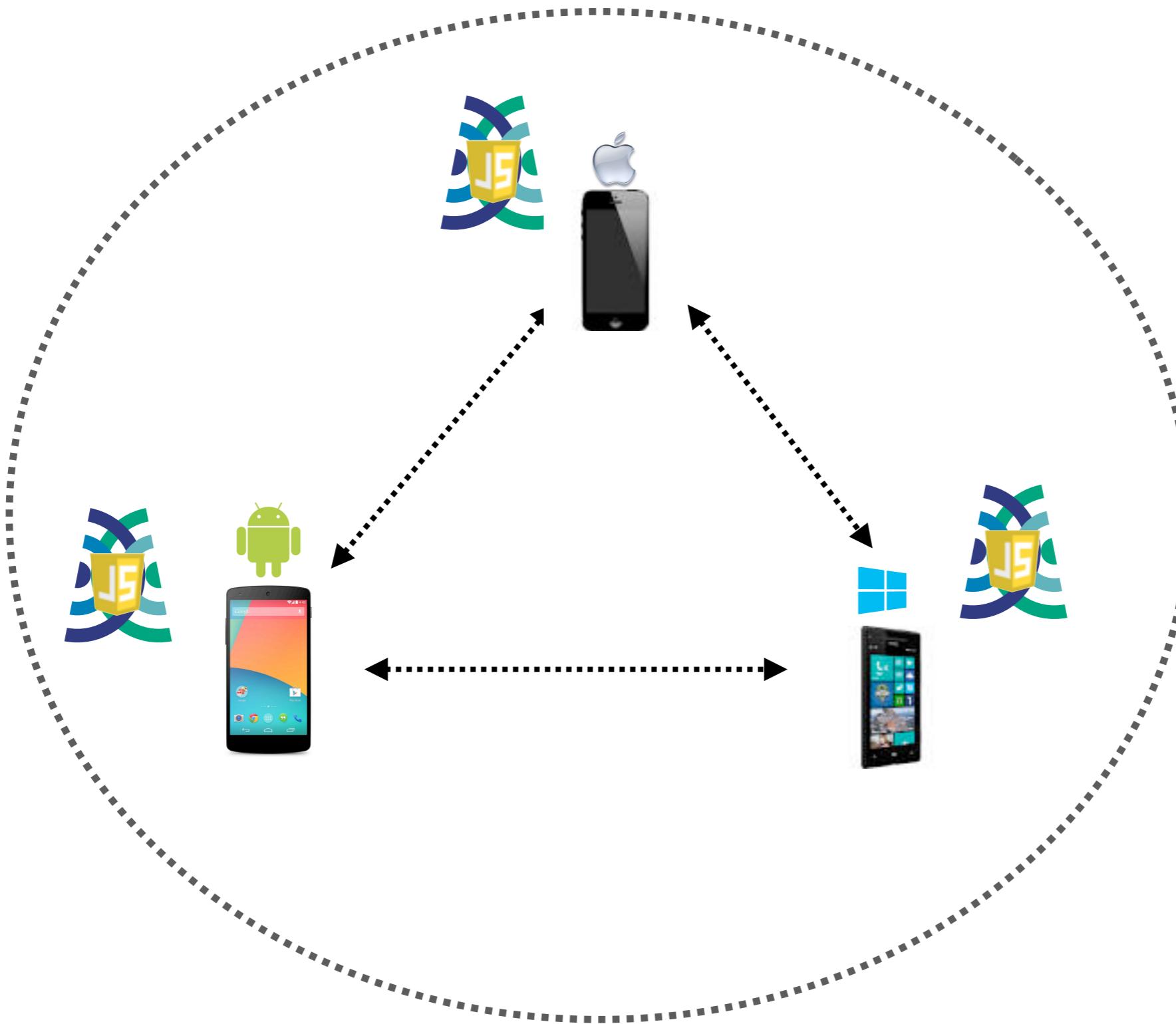


and run natively on multiple platforms



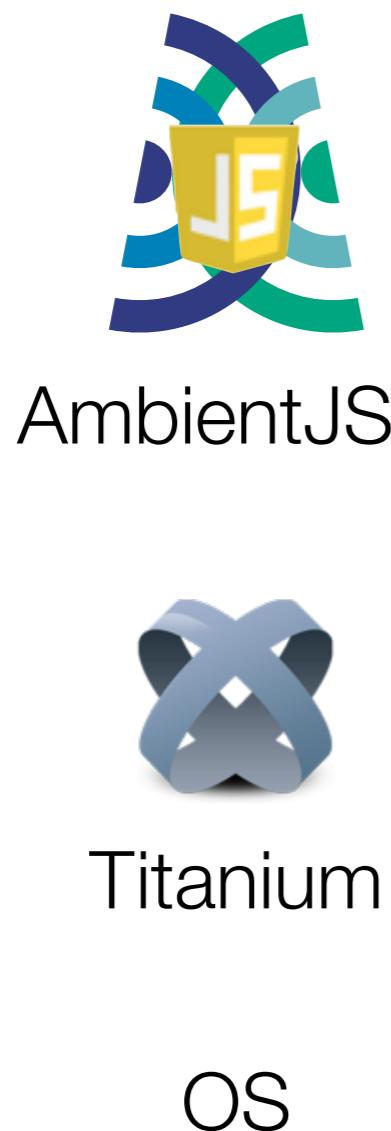
# but what about mobile wireless applications?

---





# Cross-platform ambient-oriented library





# Ambient-oriented Programming in AmbientJS

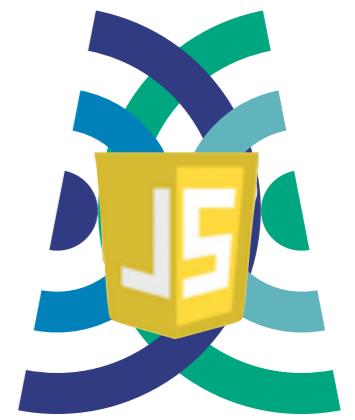
```
var Ambient = require('js/ambient/ambient');
var buddyList = {};

function initializeMessenger(name) {
  var remoteInterface = Ambient.createObject({
    "getName" : function () { return myName; },
    "talkTo" : function (msg) { displayMessage(msg); }
  });

  Ambient.exportAs(remoteInterface, "MESSENGER");
  Ambient.wheneverDiscovered("MESSENGER", function(reference) {
    var msg = Ambient.createMessage("getName", []);
    var future = reference.asyncSend(msg, "twoway");
    future.whenBecomes(function(reply) {
      buddyList[reply] = reference;
    });
  });
}

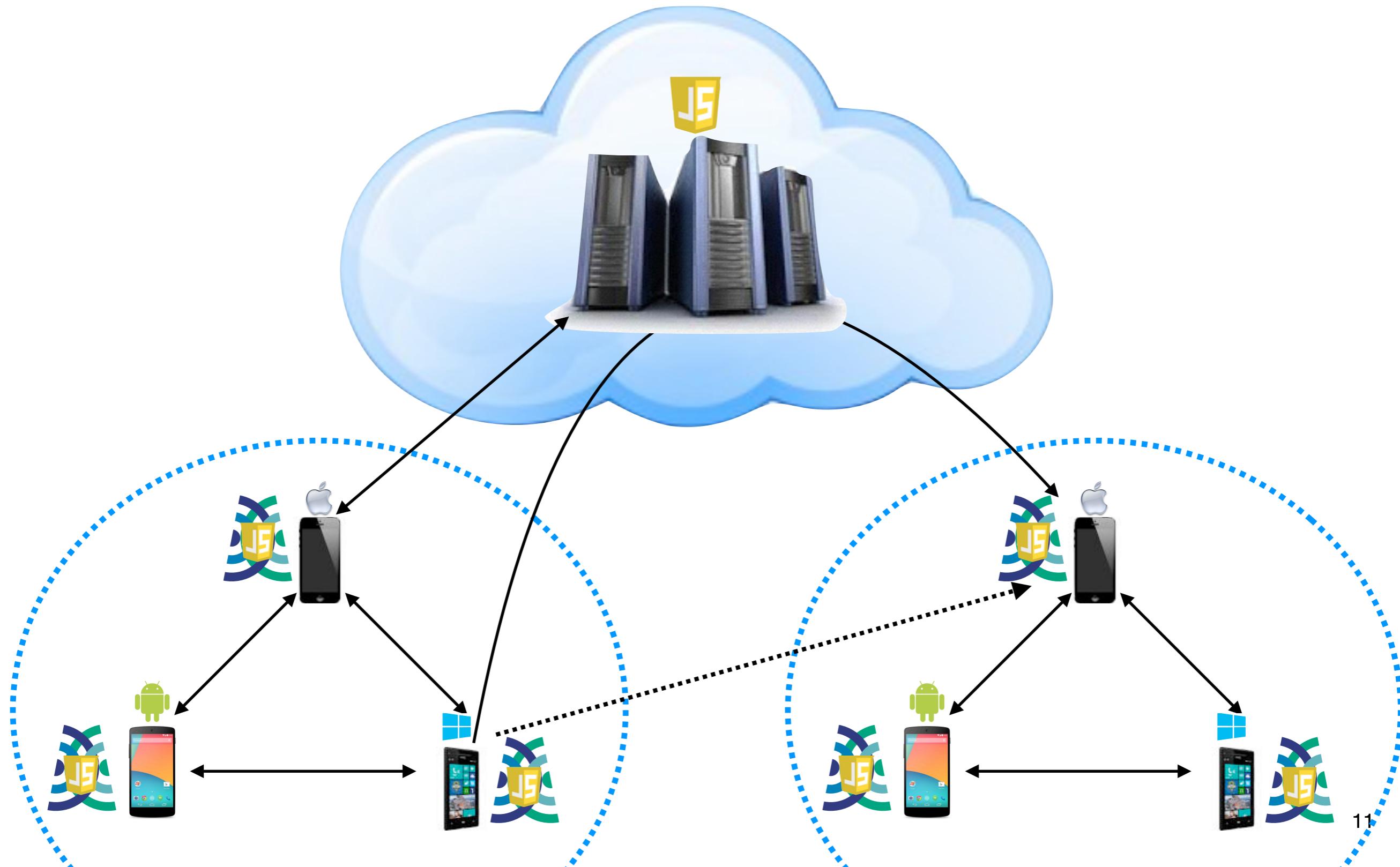
Ambient.online();
}

function sendMessage(text, buddyName) {
  var msg = Ambient.createMessage('talkTo', [myName + ":" + text]);
  buddyList[buddyName].asyncSend(msg);
}
```





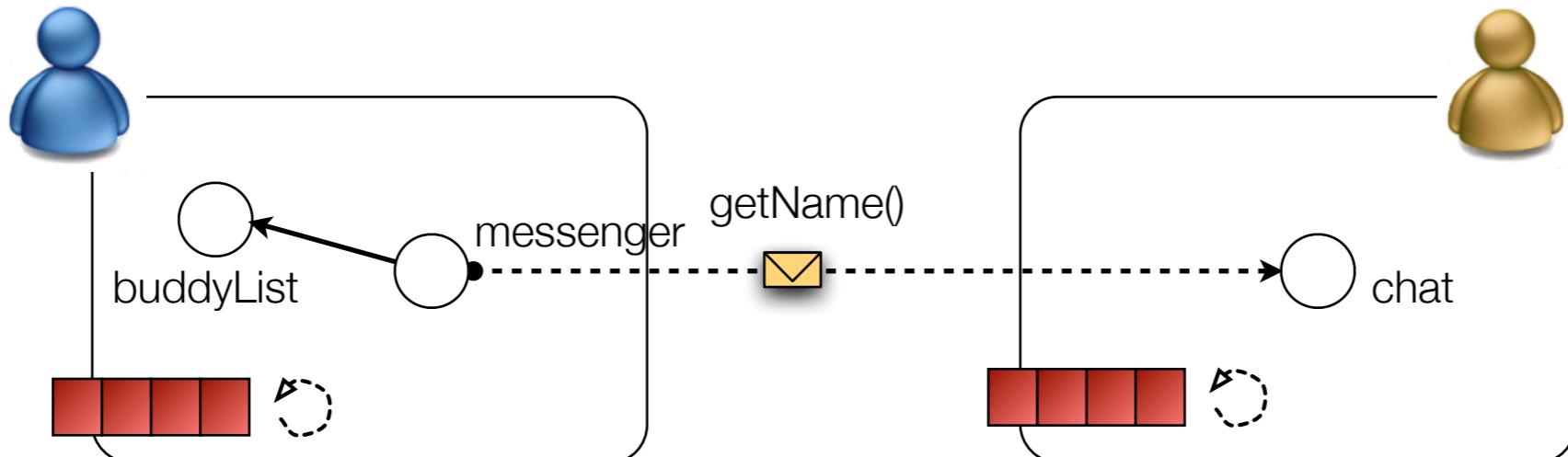
# How to reconcile P2P and client-server styles?





# Network-transparent References

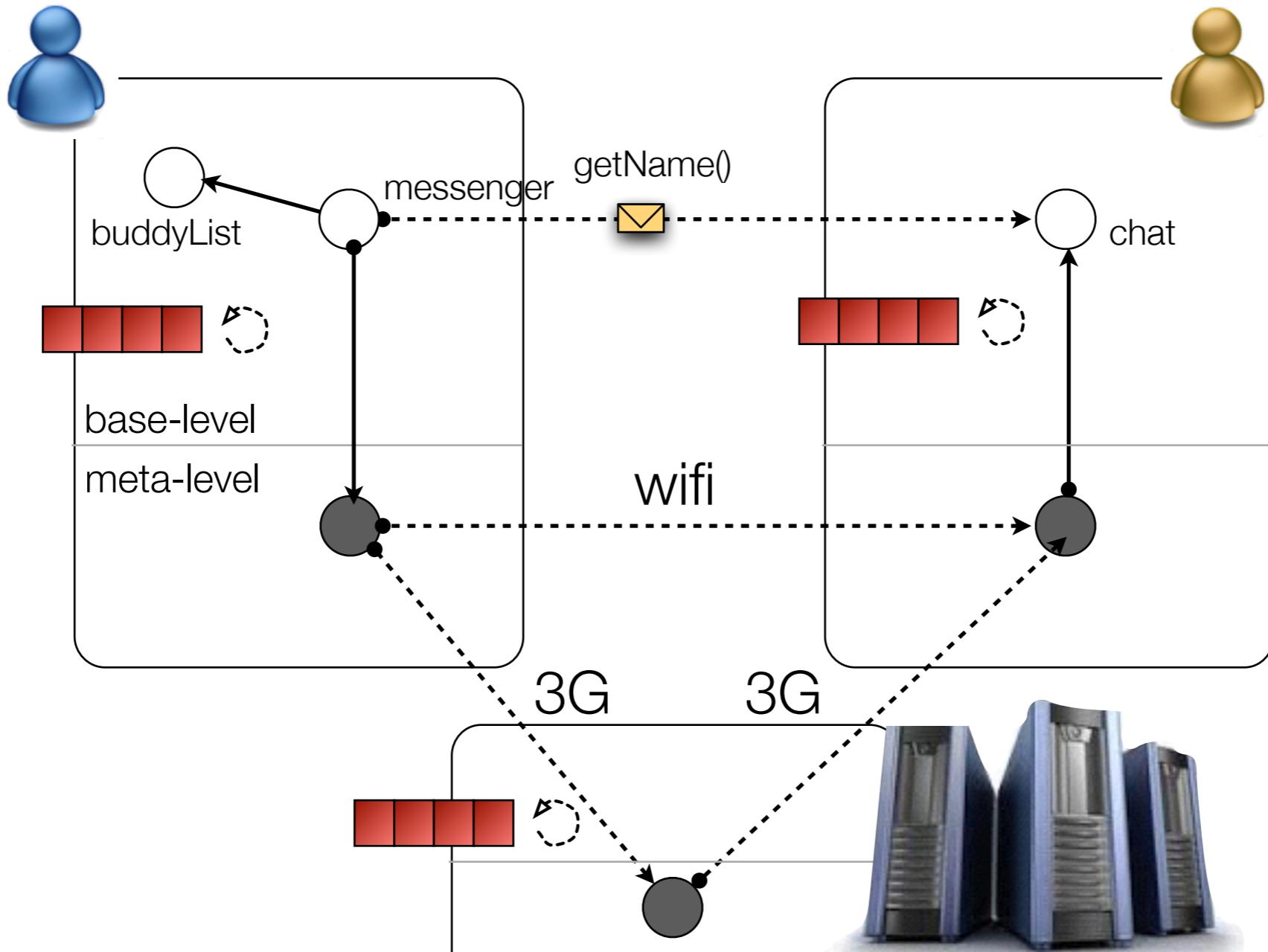
- Far references that abstract over underlying networking technology and provide at most once delivery guarantees.





# Network-transparent References (NTRs)

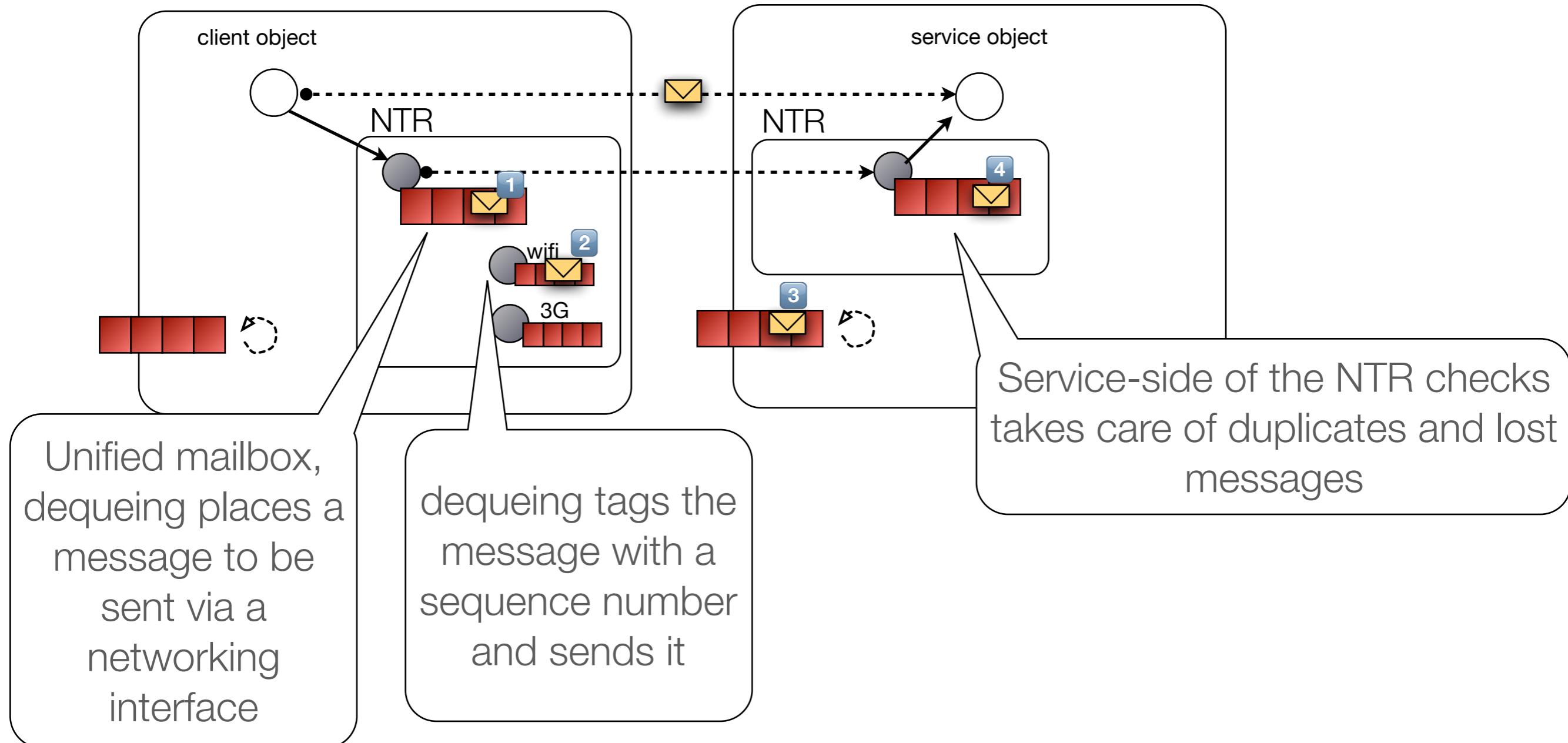
- Far references that abstract over underlying networking technology and provide at most once delivery guarantees.





# Network-transparent References (NTRs) 2

- Designates a **single unique object** over multiple networking interfaces





# How to alleviate the Pyramid of Doom?



```
varUrls = [  
    'http://api.openweathermap.org/data/2.5/weather?q=Pittsburg',  
    'http://api.events.org/data/2.5/date?q=25-10-2014'];  
  
functionrecommend(weatherinfo, events) { ... };  
  
httpGet(Urls[0]).then(function(weatherinfo) {  
    httpGet(Urls[1]).then(function(events) {  
        recommend(weatherinfo, events);  
    });  
});
```

Future Combinators



```
Future.of(recommend)  
    .ap(httpGet(Urls[0]))  
    .ap(httpGet(Urls[1]))
```

F (json->json->string)
F (json)->F(json->string)
F (json)-> F(string)



# ConnectJS

---

- Cross-platform ambient-oriented library which explores:
  - **network transparent references** for seamlessly communication over the cloud or an infrastructureless mobile networking technology.
  - **future combinator**s for structuring asynchronous code.

<http://soft.vub.ac.be/ambientJS/>