Introduction of Mobile Cloud Computing

Motivation

Motivation

- Mobile devices (e.g., smartphone, tablet pcs, etc) are increasingly becoming an essential part of human life,
- Dream of "**Information at your fingertips anywhere anytime",**
- Mobile devices still lack in resources compared to a conventional information processing device such as PCs and laptops

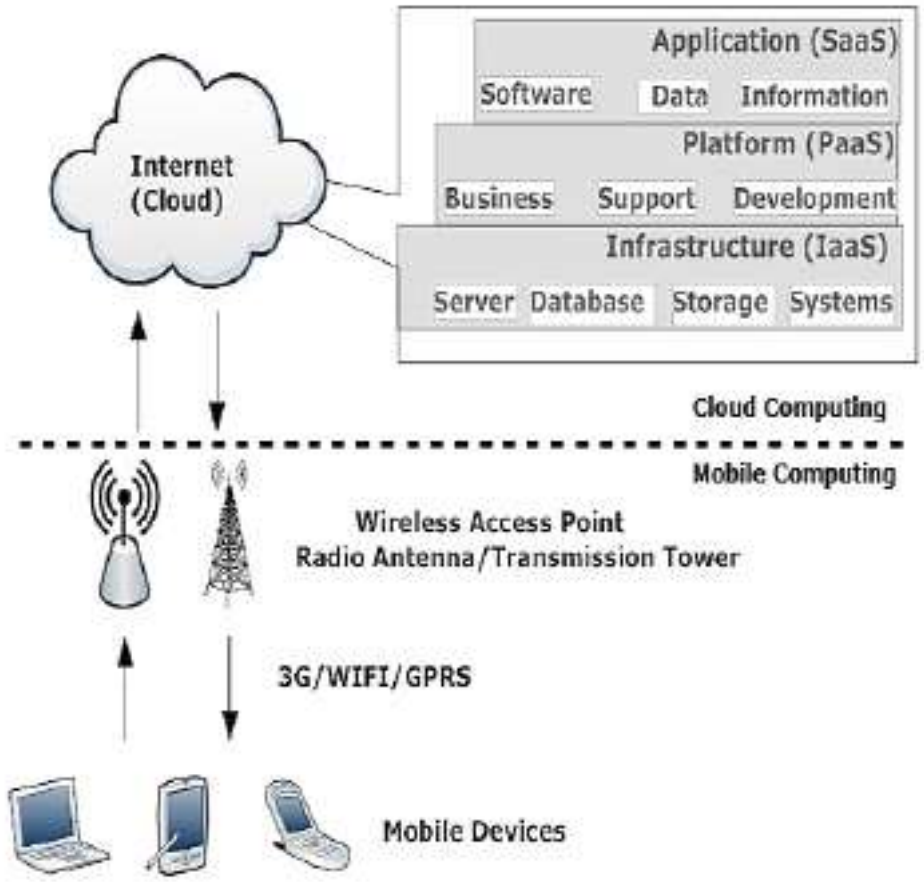
Soultion

› Mobile Cloud Computing (MCC)

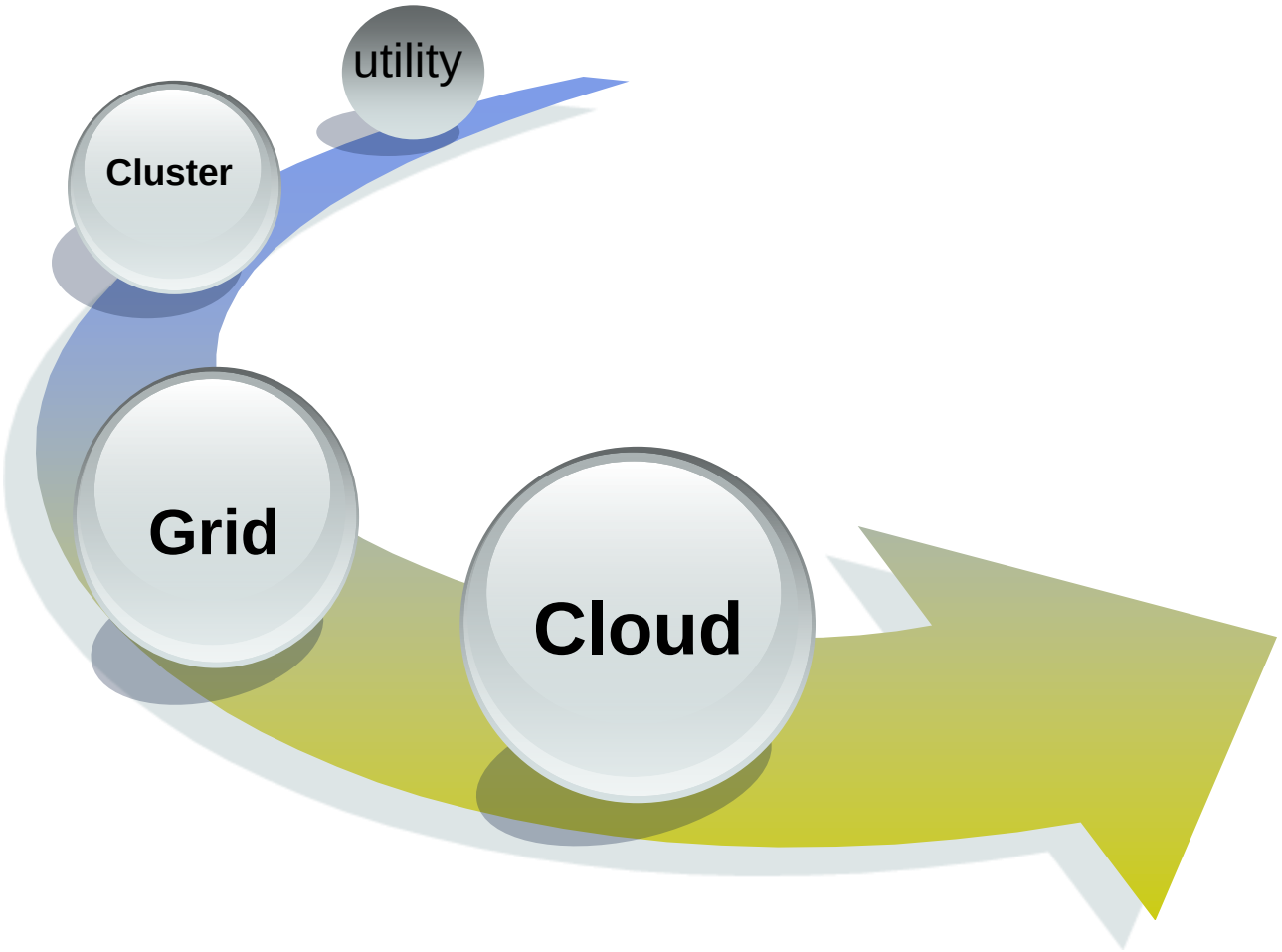
Definitions

- Cloud Computing

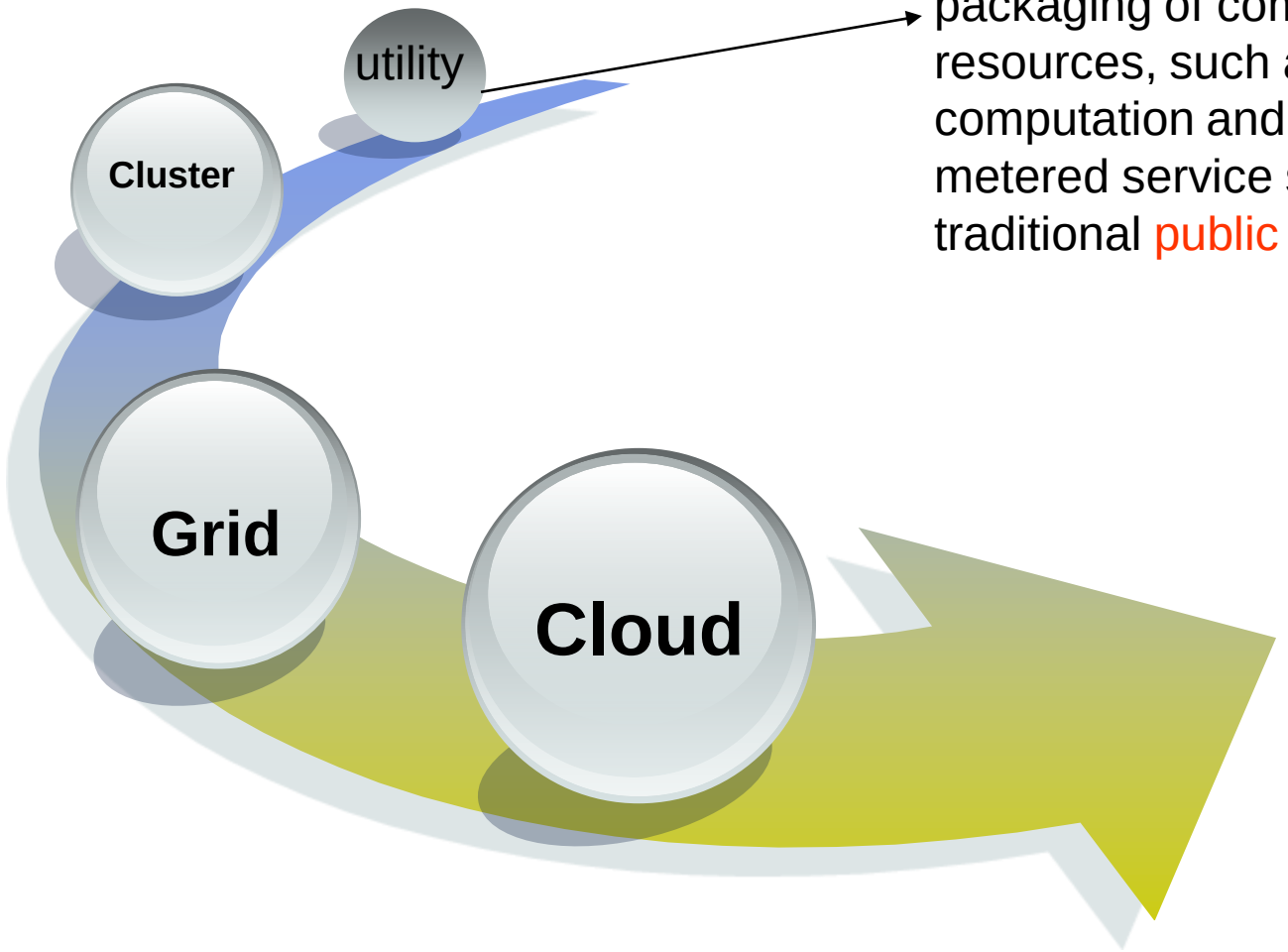
- Mobile Network



Definitions

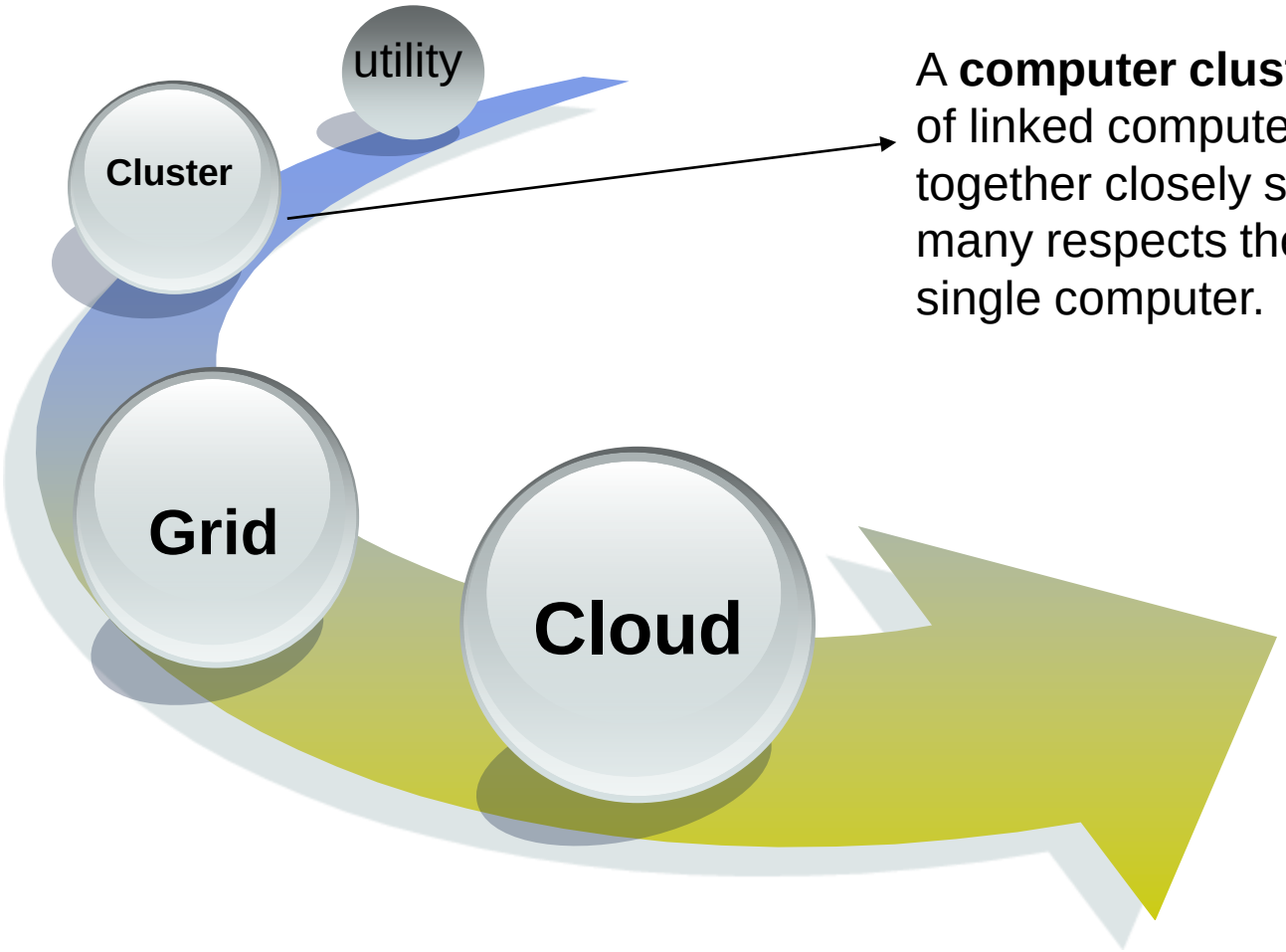


Definitions



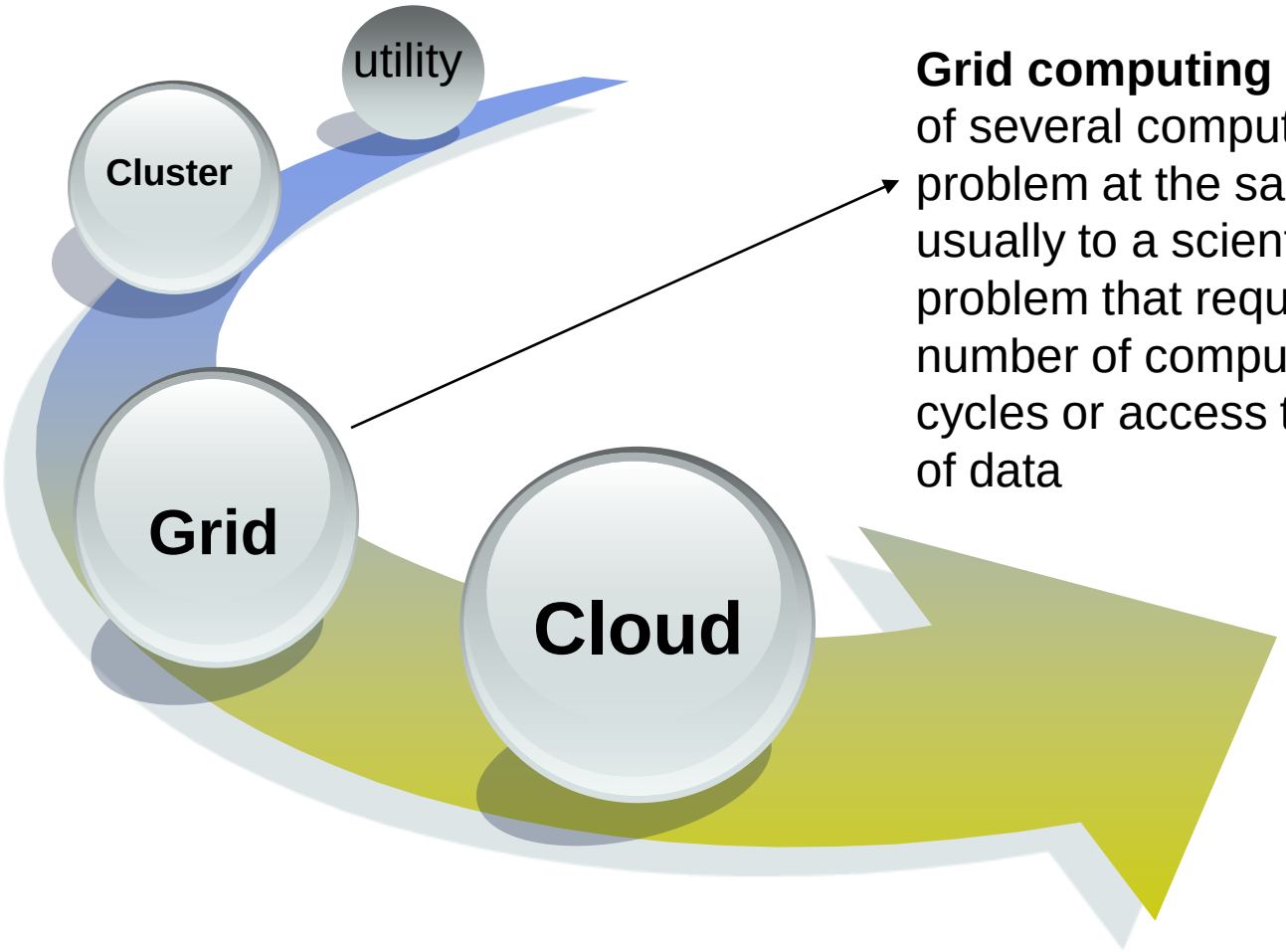
Utility computing is the packaging of computing resources, such as computation and storage, as a metered service similar to a traditional public utility

Definitions



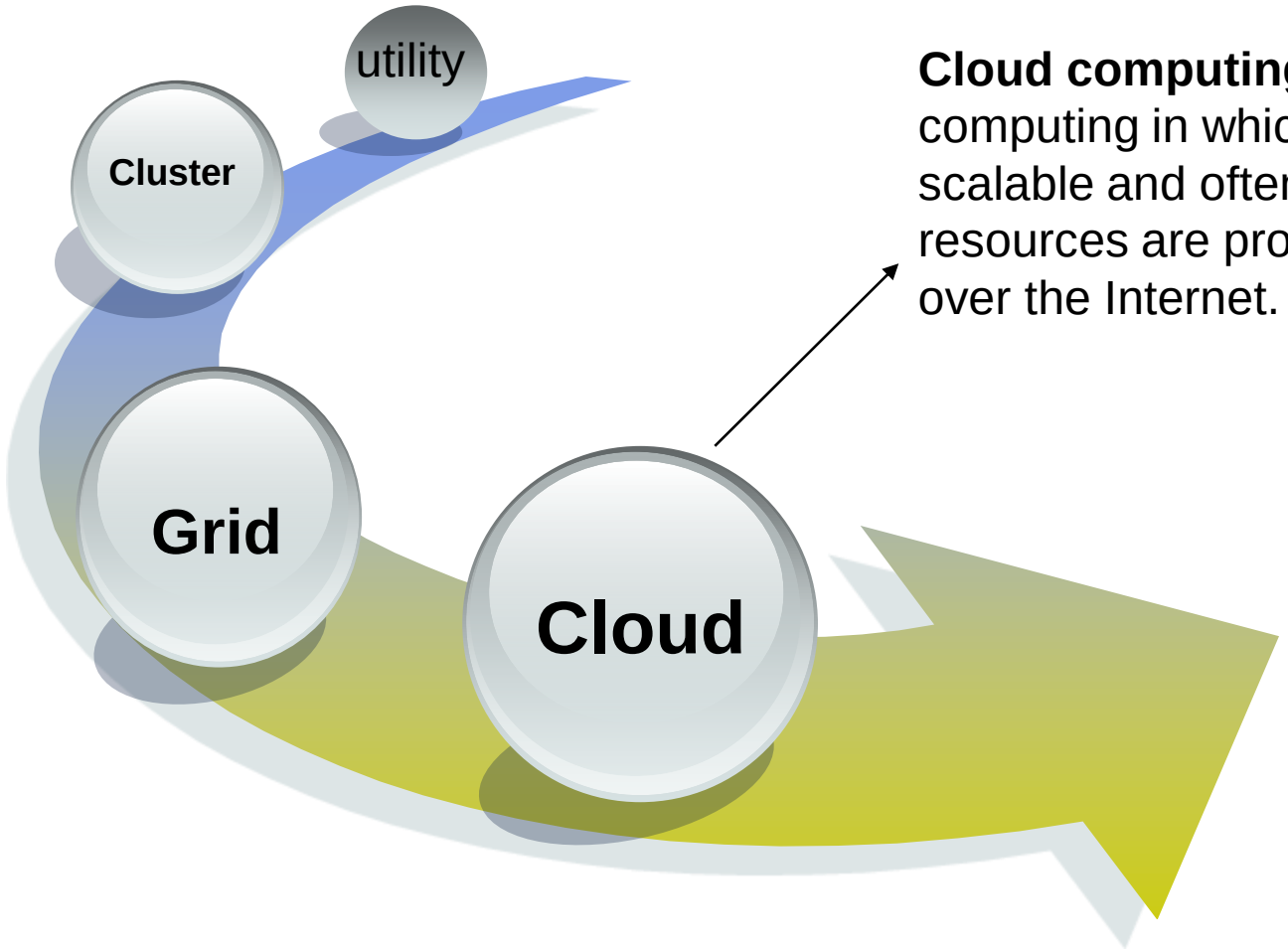
A **computer cluster** is a group of linked computers, working together closely so that in many respects they form a single computer.

Definitions



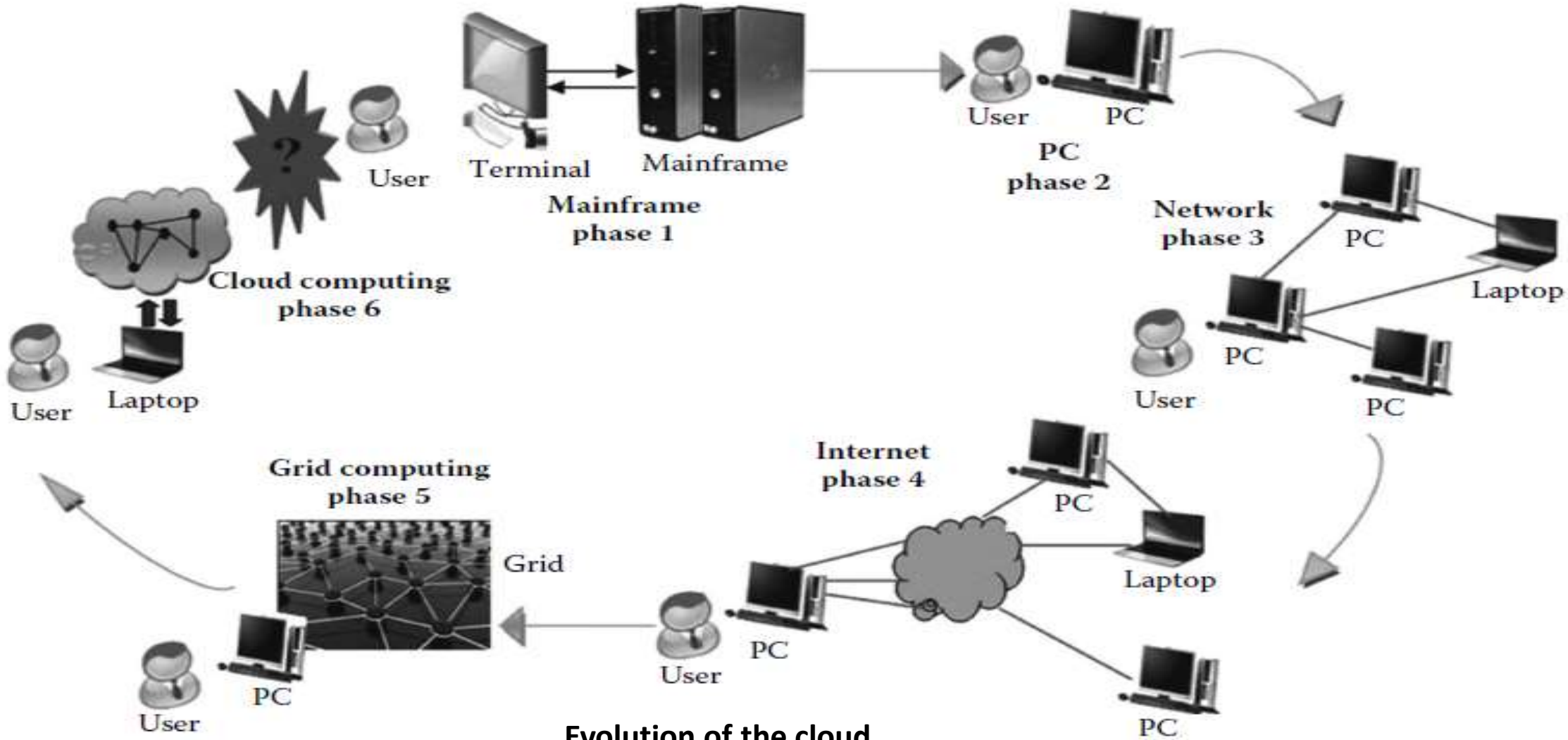
Grid computing is the application of several computers to a single problem at the same time — usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data

Definitions



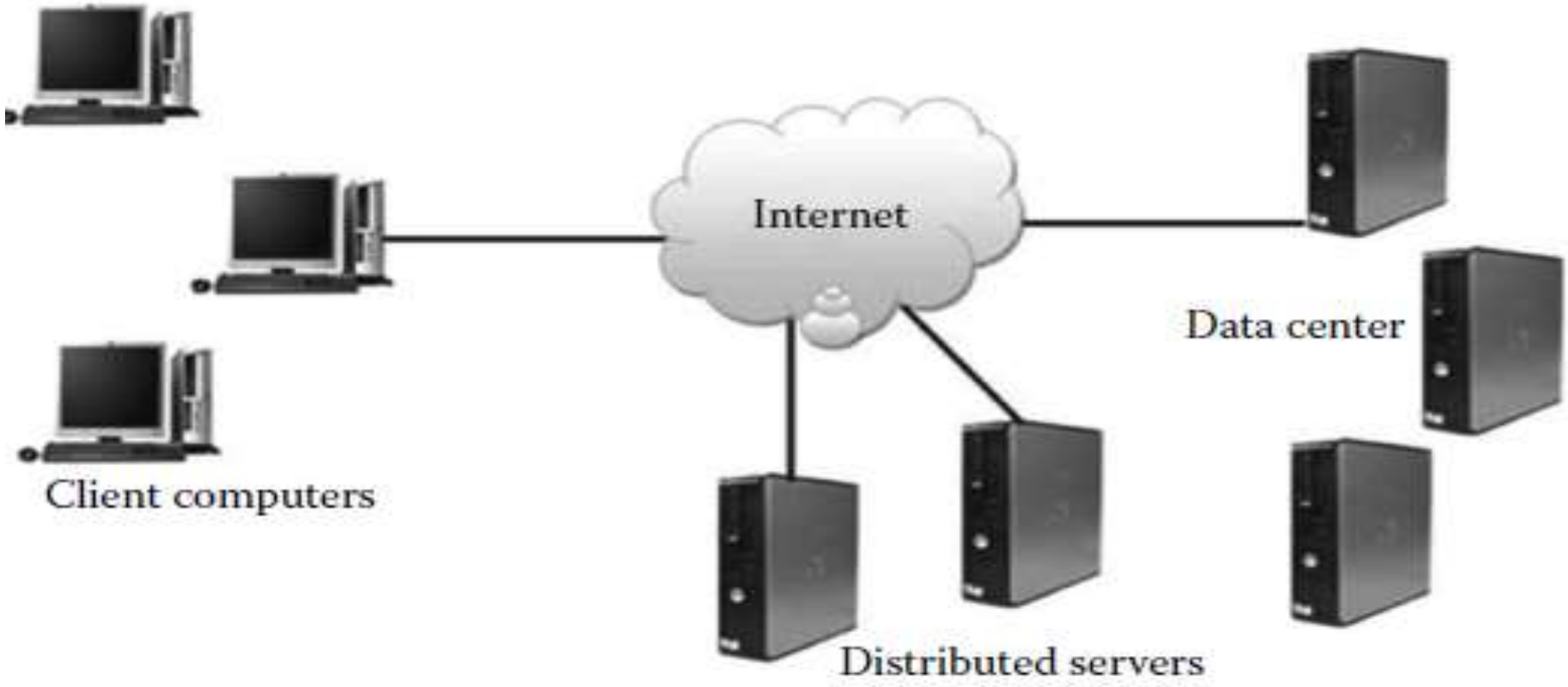
Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet.

WHAT IS CLOUD COMPUTING?

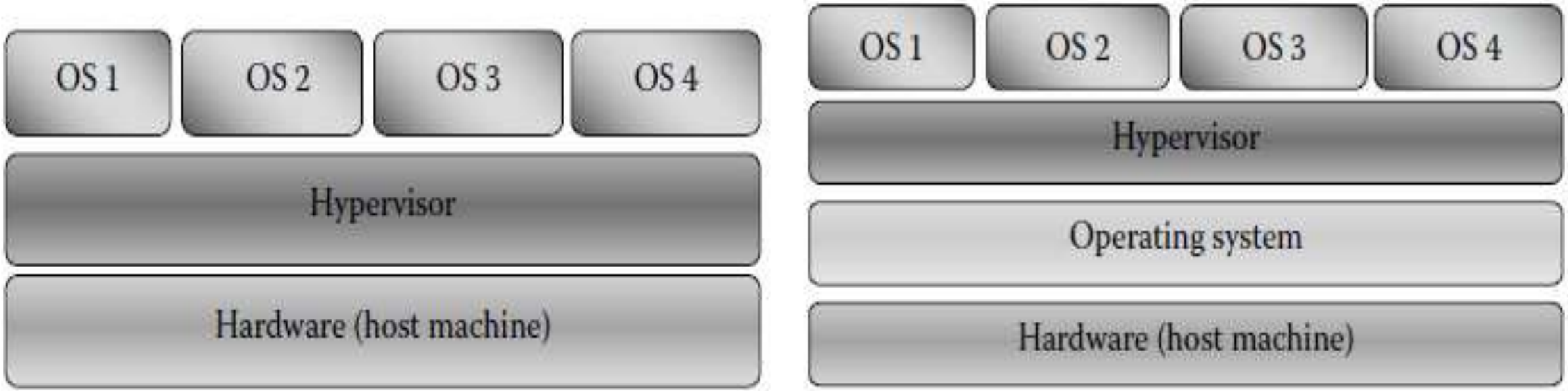


Evolution of the cloud

WHAT IS CLOUD COMPUTING?



WHAT IS CLOUD COMPUTING?

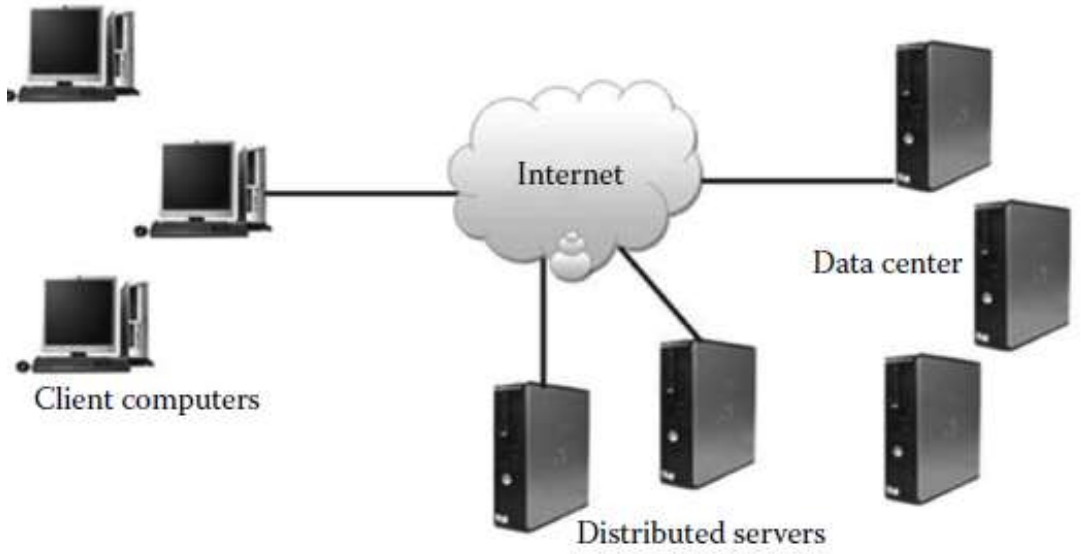


Virtualization type 1 and type 2

WHAT IS CLOUD COMPUTING?

NIST Definition

"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"



Cloud computing is a style of computing in which **dynamically scalable** and often **virtualized** resources are provided as a service over the Internet.

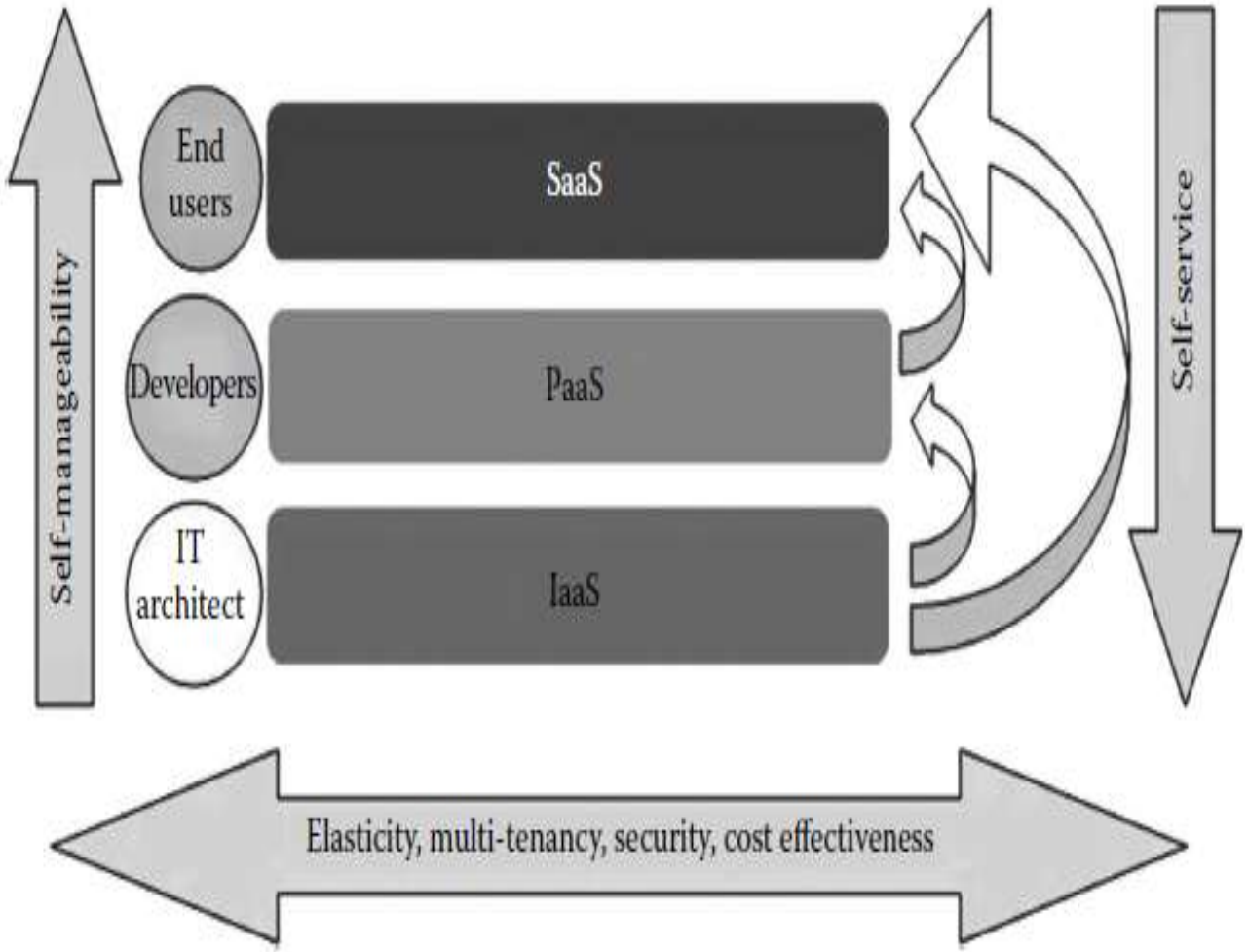
Major Types of cloud service

SaaS : Software as a Service

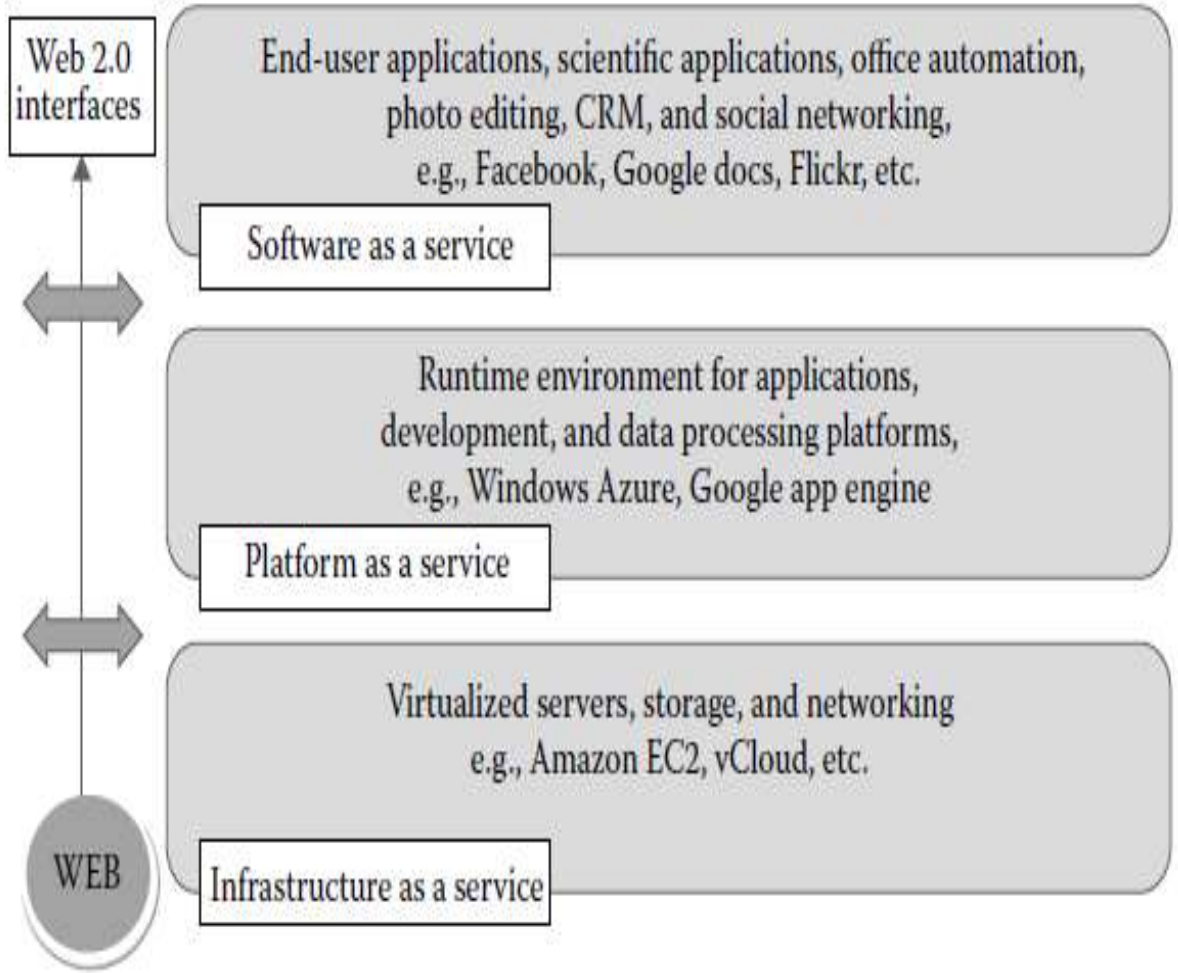
PaaS: Platform as a Service

IaaS: Infrastructure as a Service

Major Types of cloud service



Different Cloud services



Cloud computing

From Wikipedia, the free encyclopedia

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with a user's data, software and computation.

There are many types of public cloud computing:[1]

- Infrastructure as a service (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS)
- Storage as a service (STaaS)
- Security as a service (SECaaS)
- Data as a service (DaaS)
- Test environment as a service (TEaaS)
- Desktop as a service (DaaS)
- API as a service (APIaaS)
- Backend as a service (Baas)

The business model, using software as a service, users also rent application software and databases. The cloud providers manage the infrastructure and platforms on which the applications run.

Service Delivery Model Examples

Products and companies shown for illustrative purposes only and should not be construed as an endorsement

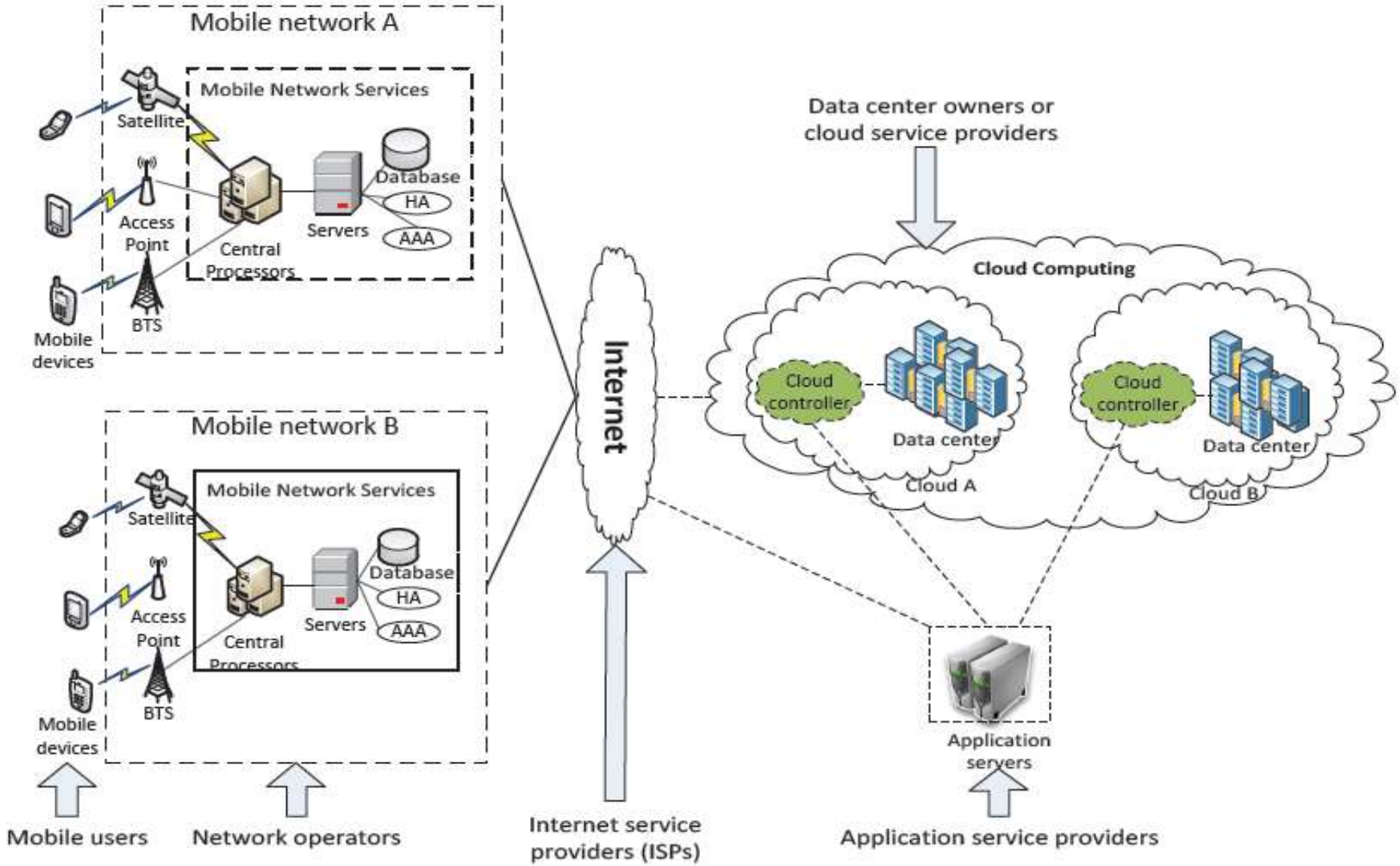
Cloud Deployment Models

- Private cloud
  - -Enterprise owned or leased
- Community cloud
  - -Shared infrastructure for specific community
- Public cloud
  - -Sold to the public, mega-scale infrastructure
- Hybrid cloud
  - -composition of two or more clouds

Mobile clould Computing

MCC

Architectures of MCC

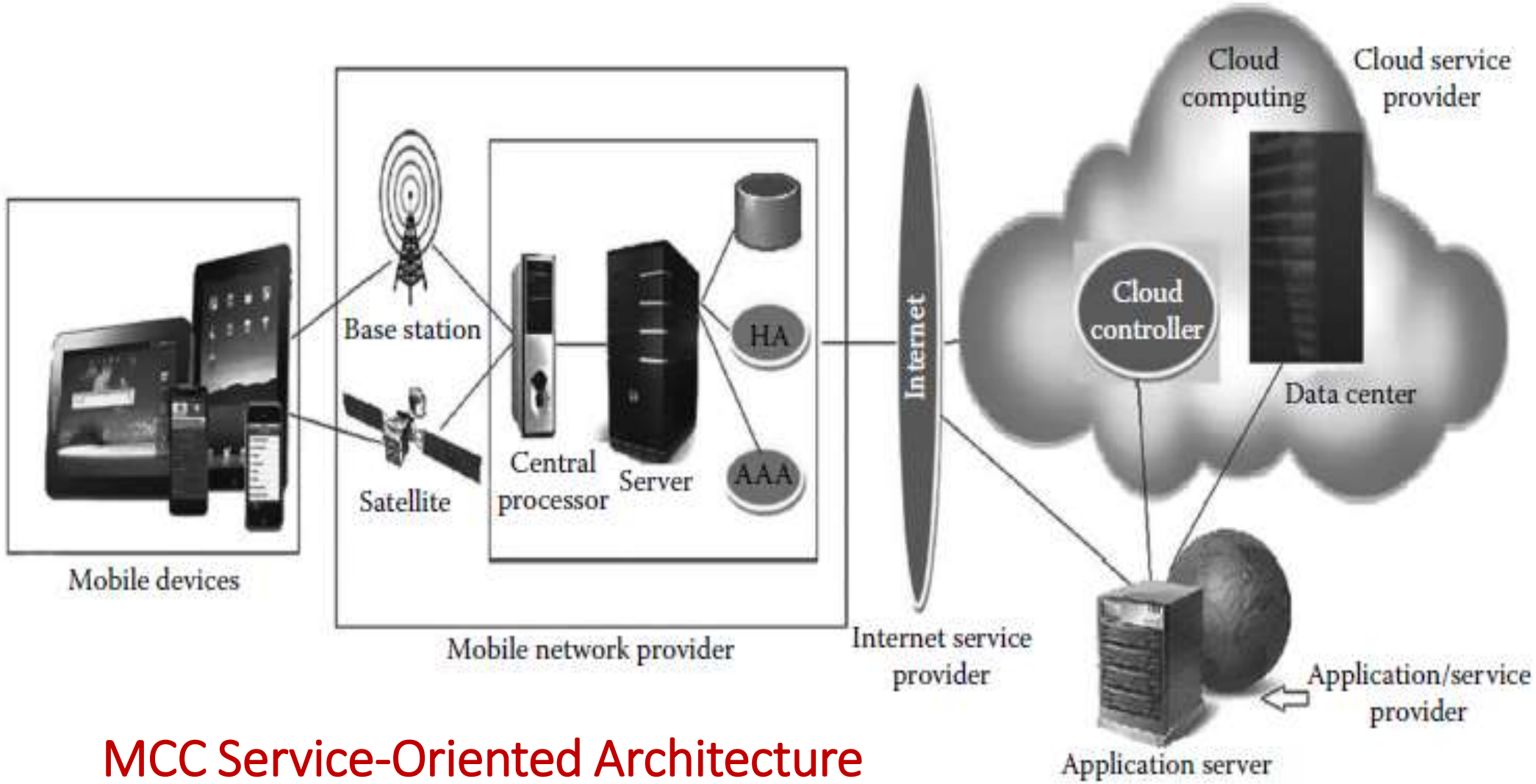


Where is the MCC?

Definition

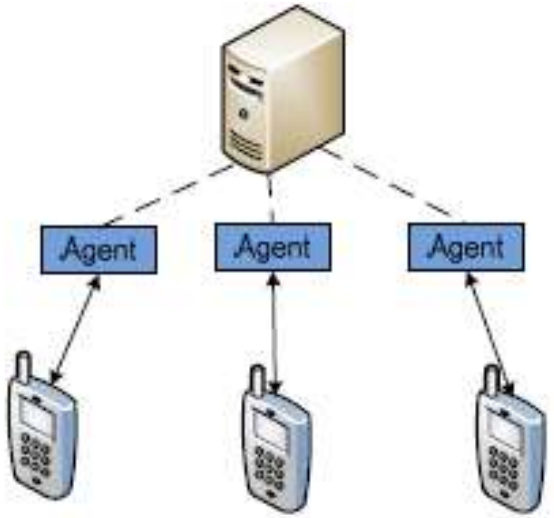
- *Mobile Cloud Computing (MCC) at its simplest, refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers”*

Mobile cloud computing Architecture

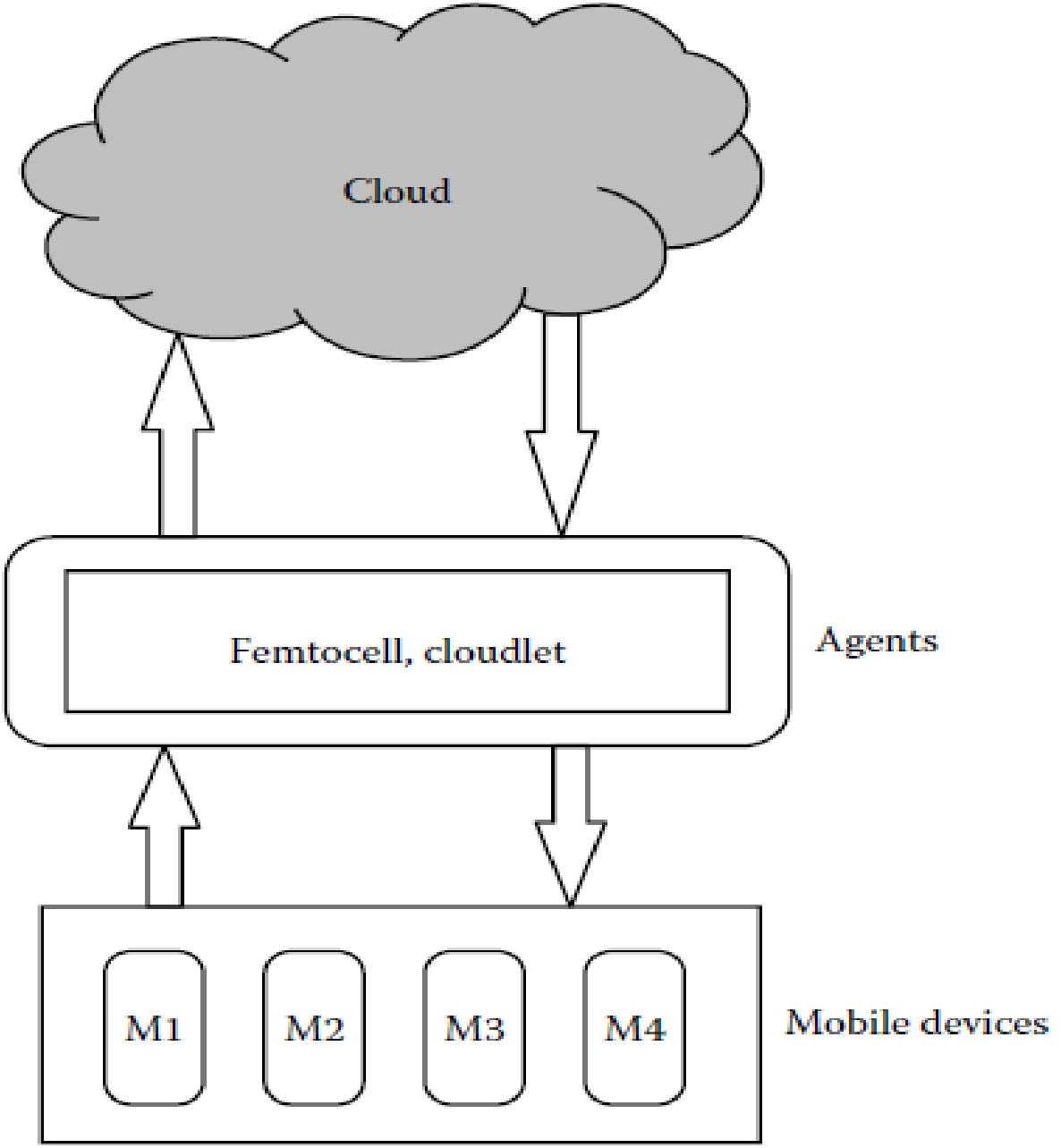


MCC Service-Oriented Architecture

Agent-client scheme

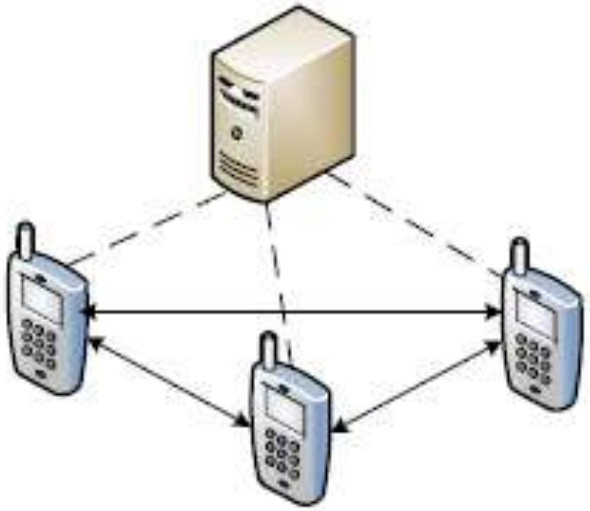


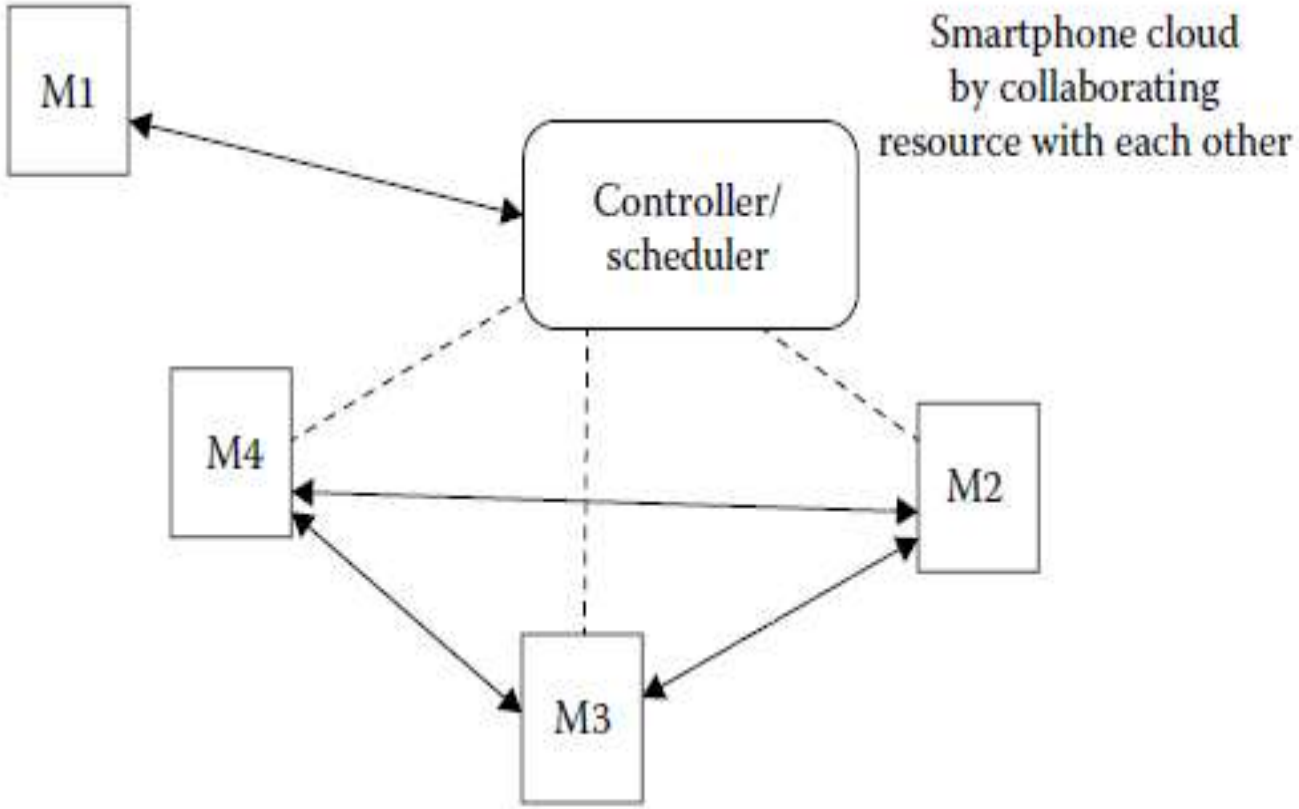
Client-agent Architecture of MCC



Collaborative Architecture of MCC

Collaborated scheme





Offloading

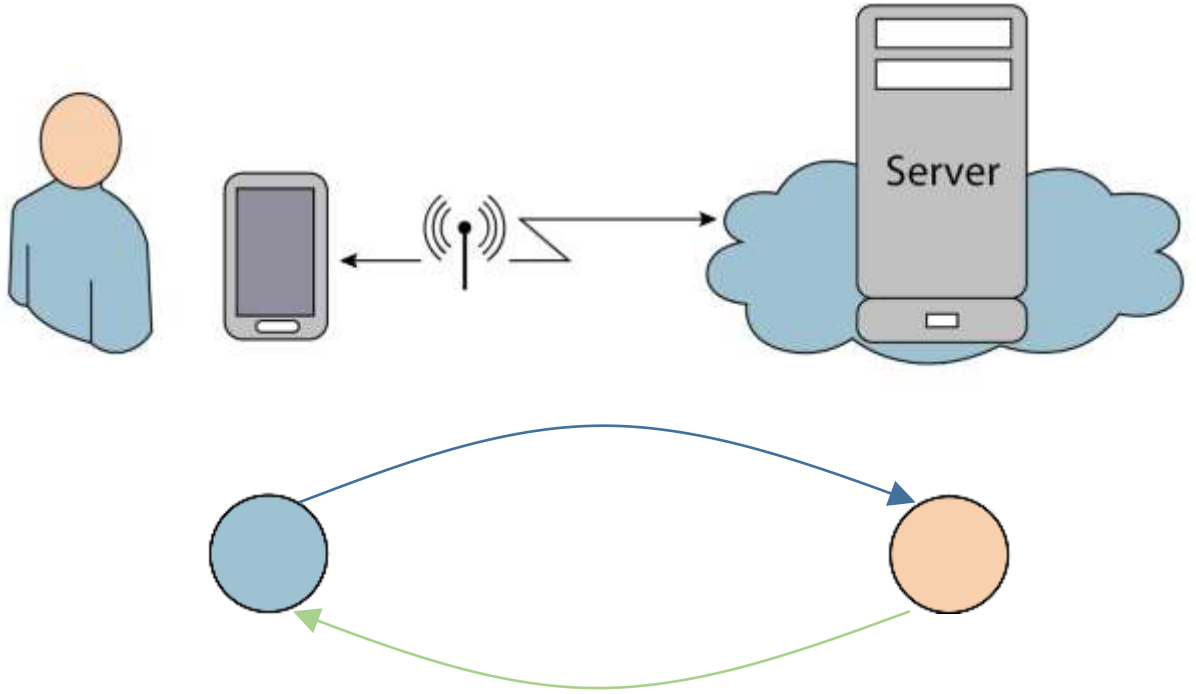
Application partition and offloading technology play an important role for the implementation of elastic applications.

Application partition decompose complex workload to atomic ones, thus can be processed concurrently.

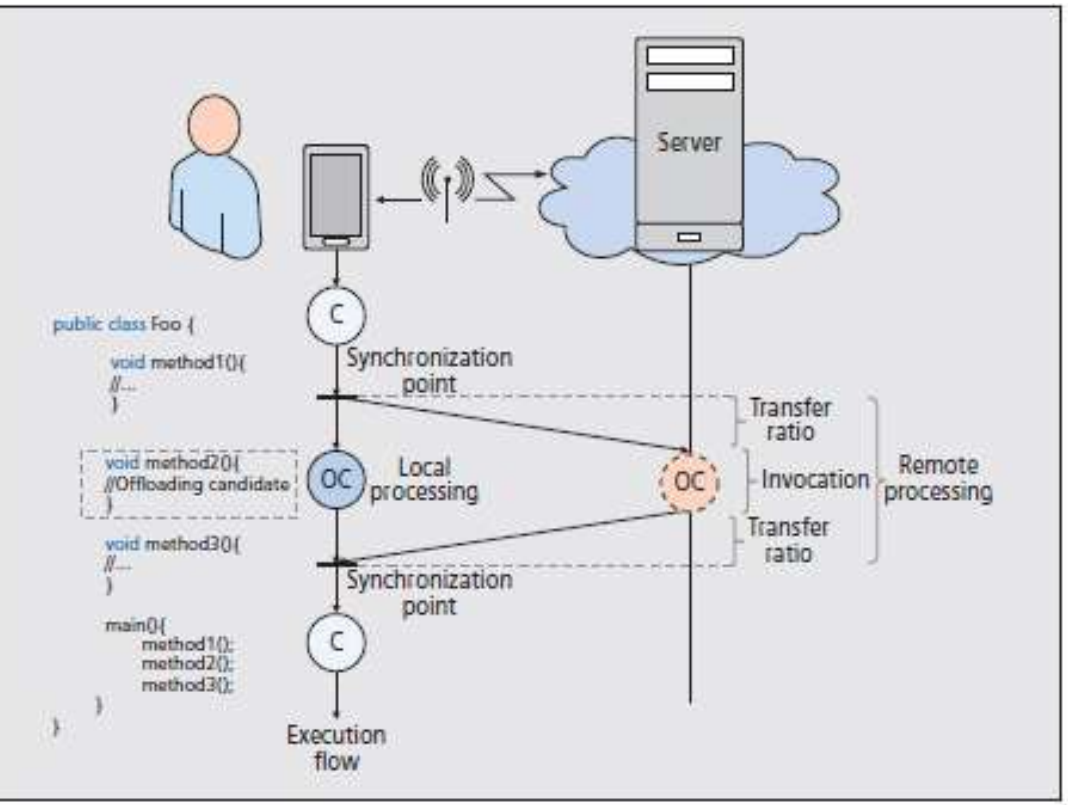
Offloading application can free burden of mobile devices.

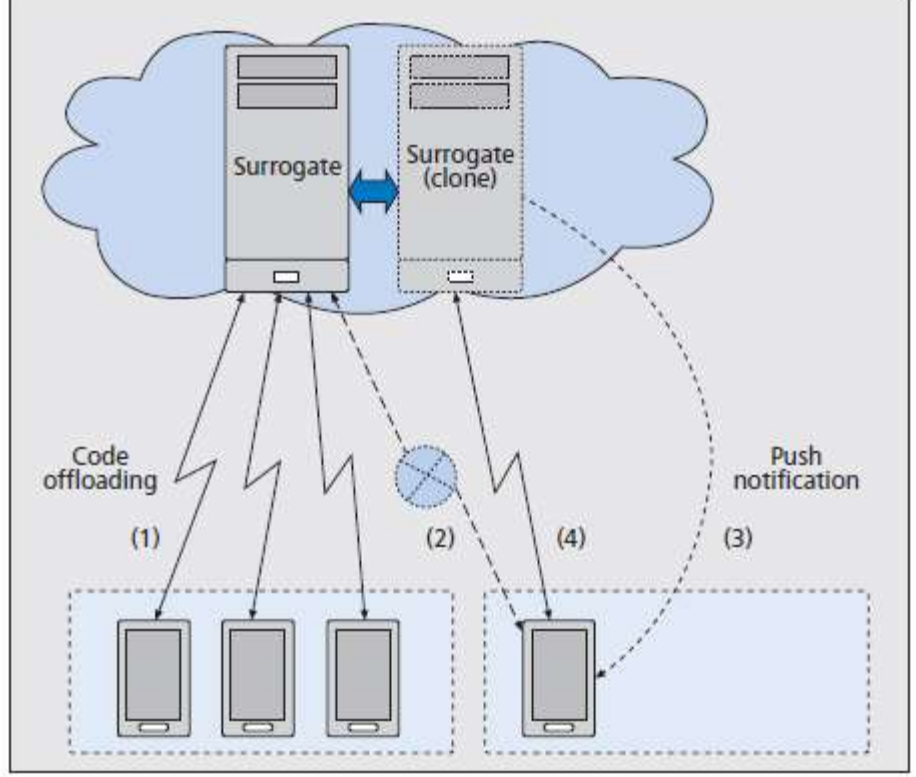
Opportunistic augmentation of resources

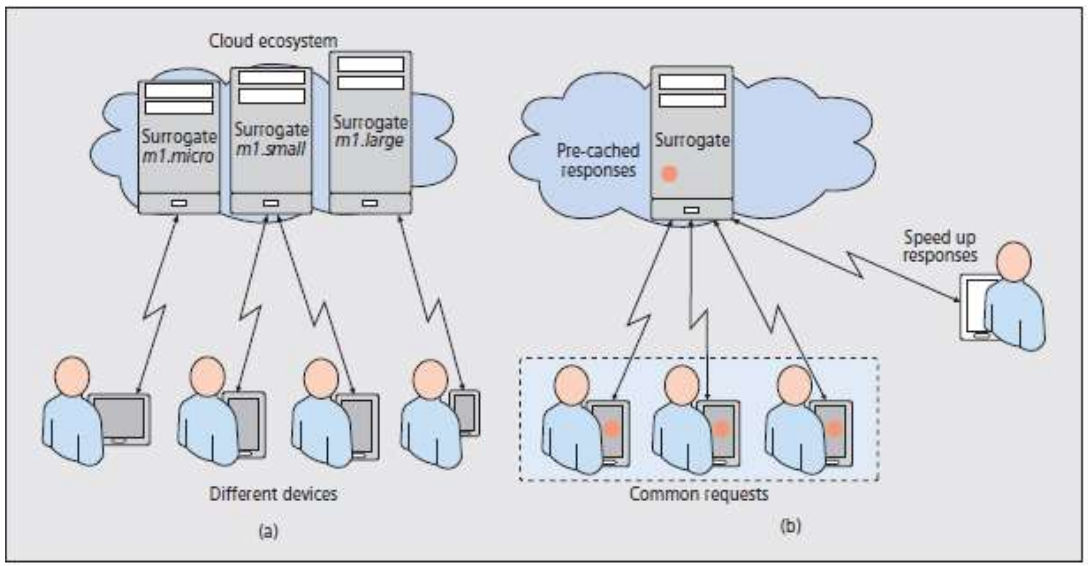
- *Processing*
- *Storage*
- *etc…*



Offloading

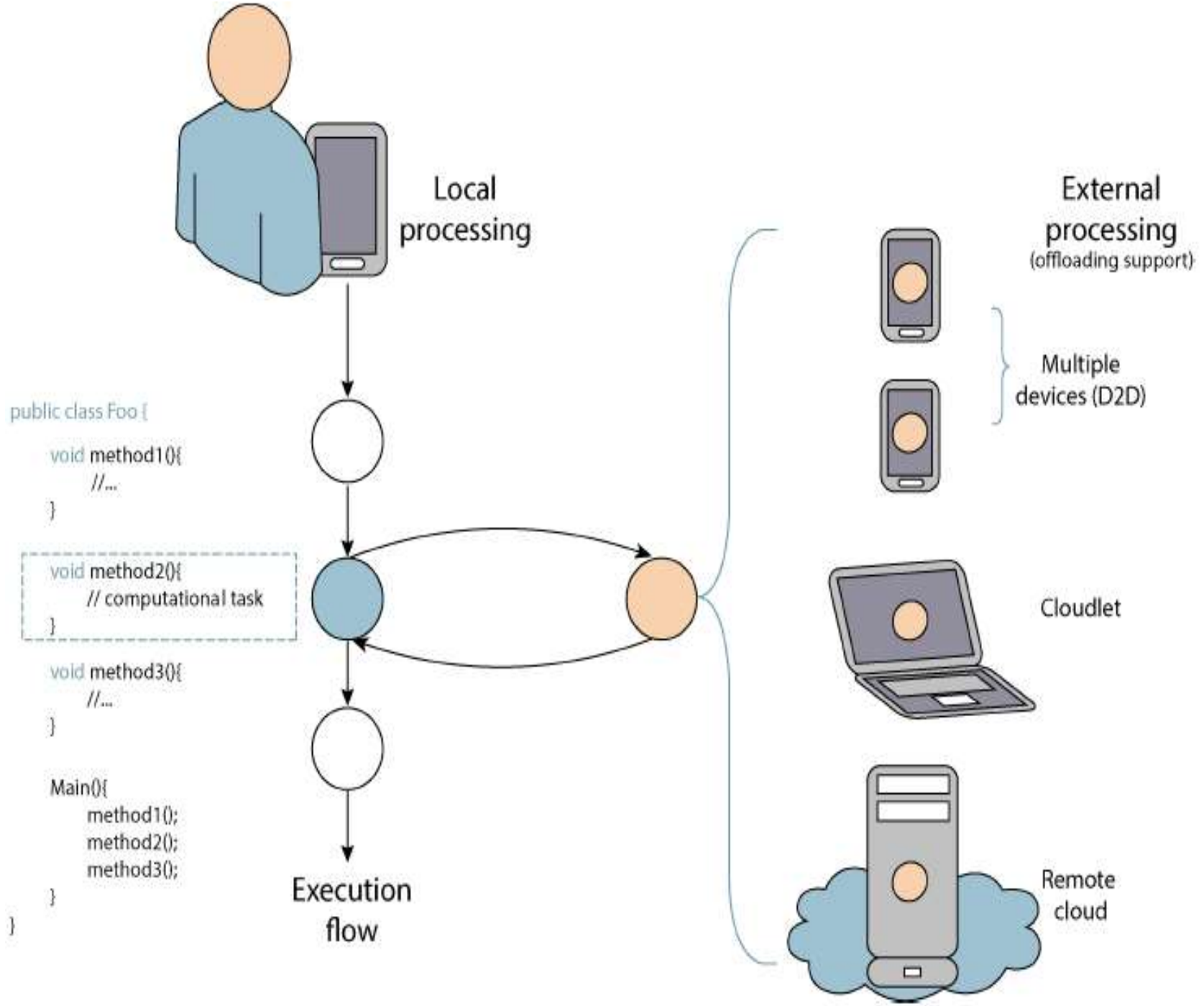

Server
public class Foo {
void method1(){
//...
}
void method2(){
//Offloading candidate
}
void method3(){
//...
}
main(){
method1();
method2();
method3();
}
}
C
Synchronization point
OC
Local processing
OC
Transfer ratio
Invocation
Transfer ratio
Remote processing
Synchronization point
C
Execution flow


Surrogate
Surrogate (clone)
Code offloading
(1)
(2)
(4)
(3)
Push notification

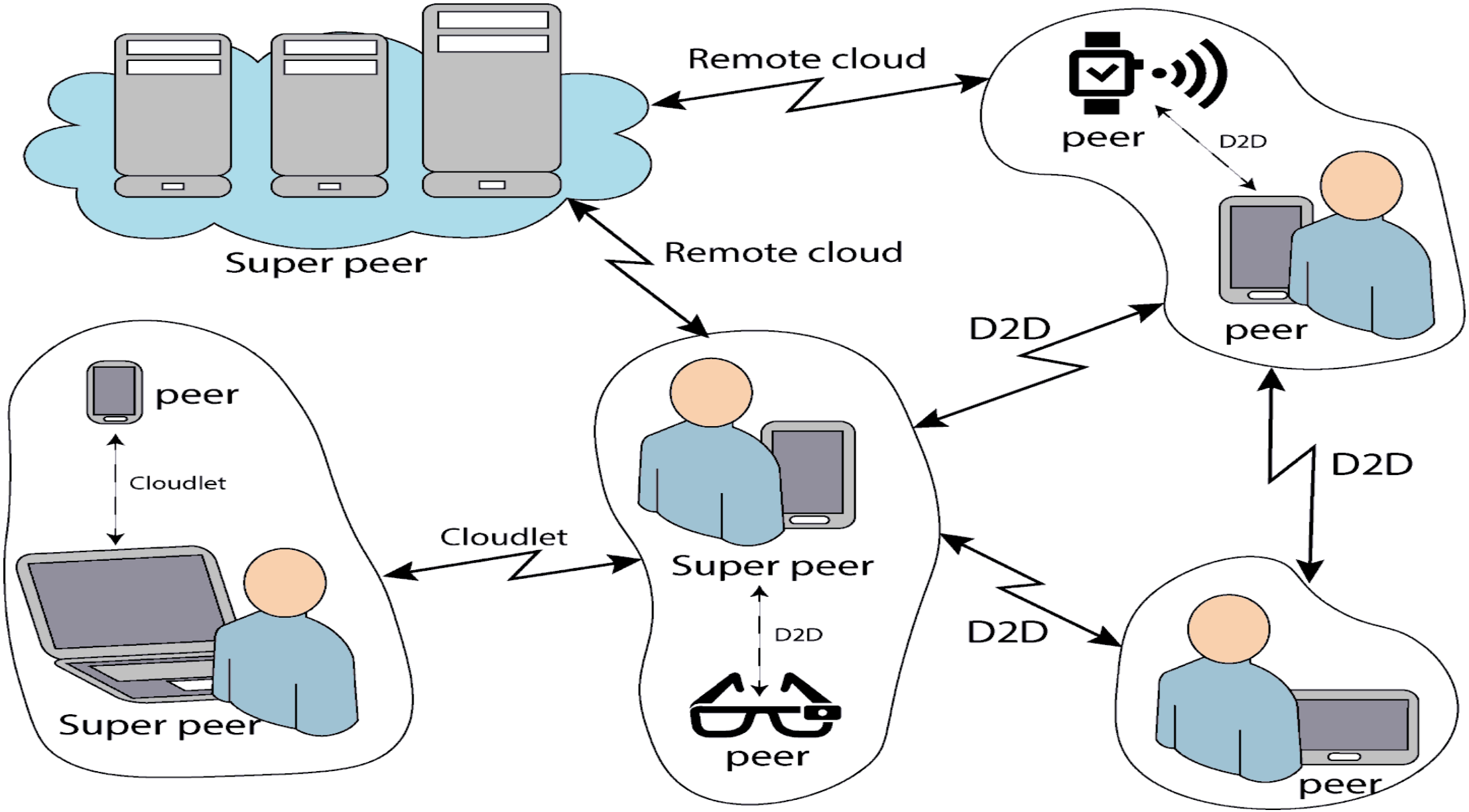

Cloud ecosystem
Surrogate m1.micro
Surrogate m1.small
Surrogate m1.large
Different devices
(a)
Pre-cached responses
Surrogate
Speed up responses
Common requests
(b)

Hybrid offloading systems

- Cloudlet
  - Scalability
- Remote cloud
  - Latency in the communication
- Device-to-Device (D2D)
  - Social participation



Social-aware hybrid offloading



Advantages of MCC

- **Extending battery lifetime**

- **Improving data storage capacity and processing power**

- **Improving reliability**

How MCC Can Extend Battery Lifetime?

Challenges:

- Battery is one of the main concerns for mobile devices,
- Traditional approaches need to **changes the structure of mobile devices**.
- The additional cost for the end mobile users is not appealing in wireless networks.

MCC's solution:

› **Computation offloading technique:**
  – Immigrate the large computations and complex processing from resource-limited devices (i.e., mobile devices) to resourceful machines (i.e., servers in clouds).

› This avoids taking a **long application execution time** on mobile devices which results in **large amount of power consumption.**

How MCC Can Improve Storage Capacity?

Challenges

- Users need more and more capacity for saving the essential information on mobile devices,
- Need to change the device,
- More capacity, more weight

MCC's solution

- MCC is developed to enable mobile users to store/access the large data on the cloud through wireless networks,
- Examples of existing services:
  - Amazon Simple Storage Service (Amazon S3),
  - Image Exchange,
  - Flickr, ShoZu.

How MCC Can Improve Reliability?

Challenges

- Users need reliable backup for their information,
- Lack of data security model for both service providers and users in existing mobile users,

MCC's solution

- Storing data or running applications on clouds is an effective way to improve the reliability since the data and application are stored and backed up on a number of computers.

Applications of MCC

- Mobile commerce,
- Mobile healthcare,
- Mobile learning,
- Mobile Gaming.

Mobile Commerce

- Mobile commerce (m-commerce) is a business model for commerce using mobile devices.

Mobile Commerce

- Some categories of M-commerce:
  - Finance,
  - Advertising,
  - Shopping.

Application Classes	Type	Examples
Mobile Financial application	B2C (Business to Customer), B2B (Business to Business)	Banks, brokage firms, mobile-user fees
Mobile Advertising	B2C	Sending Custom made advertisement according to users' physical location
Mobile Shopping	B2C, B2B	Locator/order certain products a mobile terminal

Mobile Learning (M-LEARNING) = (E-LEARNING) + Mobility

Traditional m-learning applications have limitations in terms of

1- High cost of devices and network,

2- Low network transmission rate,

3- Limited educational resources

Cloud-based m-learning applications are introduced to solve these limitations.

For example, utilizing a cloud with the large storage capacity and powerful processing ability, the applications provide learners with much richer services in terms of data (information) size, faster processing speed, and longer battery life.

Mobile-healthcare

Comprehensive health monitoring services,
Intelligent emergency management system

- Health-aware mobile devices detect pulse-rate,
- Pervasive access to healthcare information,

- Pervasive lifestyle incentive.

Mobile Gaming

- Mobile game (m-game) is a potential market generating revenues for service providers.
- M-game can completely offload game engine requiring large computing resource (e.g., graphic rendering) to the server in the cloud, and gamers only interact with the screen interface on their devices.

Other applications on MCC

- Keyword based searching
- Voice based searching
- Tag- Based searching

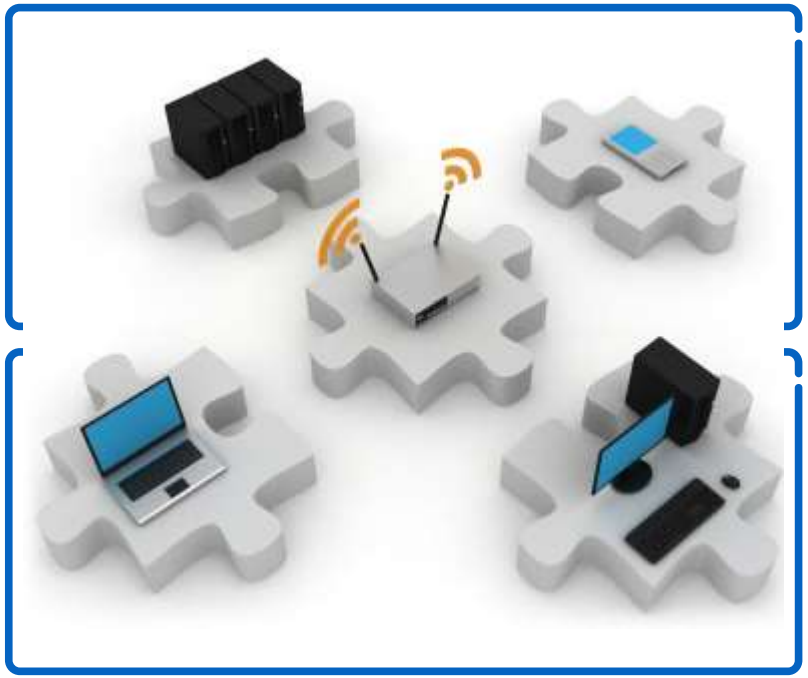
• ISSUES AND APPROACHES OF MCC

Due to the integration of two different fields, i.e., **cloud computing** and **mobile networks**,

MCC has to face **many technical challenges**.

Issues in Mobile Communication Side

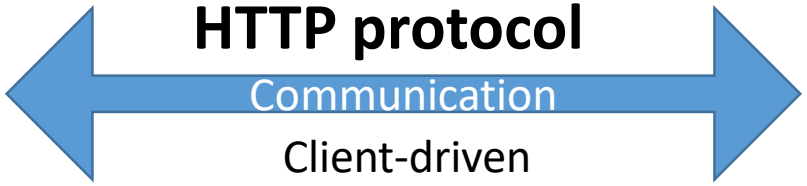
- Network latency and limited bandwidth
- Availability
- Heterogeneity



Thank you

HTTP

Mobile device **HTTP protocol** **Cloud**



Ressource request (service)/ Client initiates communication

Retrieve new data / Client decides of data updates checks on the server

==>

No way for clients to know others' clients data update

<== Server cannot iniate communication (cannot *push data* to client)

Standard request/response model of communication

How to design around?

HTTP

Mobile device **HTTP protocol** **Cloud**

User interaction with the app

Pull down to refresh UI ==> fetch data from the server

Update on view opened / on App launch

==> over user refresh may consume bandwidth when there is nothing to update

However, the user will stop refreshing at some point.

HTTP *Polling*

Mobile device **HTTP protocol** **Cloud**

Client continuously request data at regular interval of *t. (ex: 3 sec)*

Client doesn't know if there are updates or not.

- Wasting ressource on the server (ressource allocation, process request, check availaible data for the client, send response back)

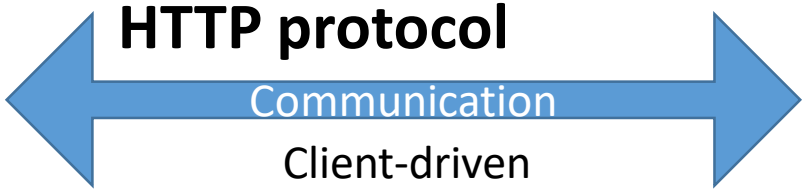
- Wasting network ressources, memory and CPU

Lots of clients ==> multiply ressources waste and effects on the network

Conclusion : *significant cost* in getting updates *with Polling*

Adaptive HTTP *Polling*

Mobile device **HTTP protocol** **Cloud**



Data updating rapidly in the server every t seconds (short time interval)

==> client should get a response for each request sent

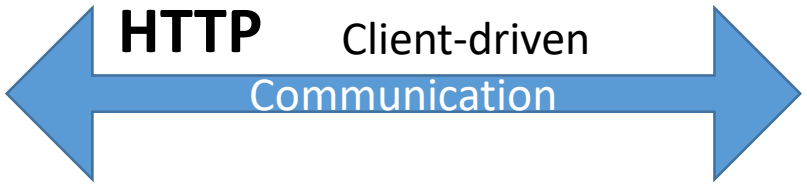
whithin t seconds time interval

if the client doesn't get the response after *n* requests, it adapts its polling rate. *2t, 4t, ..., 2kt* (client : maybe i am wasting ressources on the server)

2kt a very long polling interval ==> low update because of client polling policy.

switch back to short polling rate after a new update

Adaptive HTTP *Polling*

Mobile device  **Cloud**

Data updating rapidly in the server every *t* seconds (short time interval)

==> client should get a response for each request sent

whithin *t* seconds time interval

if the client doesn't get the response after *n* requests, it adapts its polling rate.

exponential backoff $2^K t$

(client : maybe i am wasting ressources on the server)

$2^K t$ a very long polling interval ==> low update because of client polling policy.

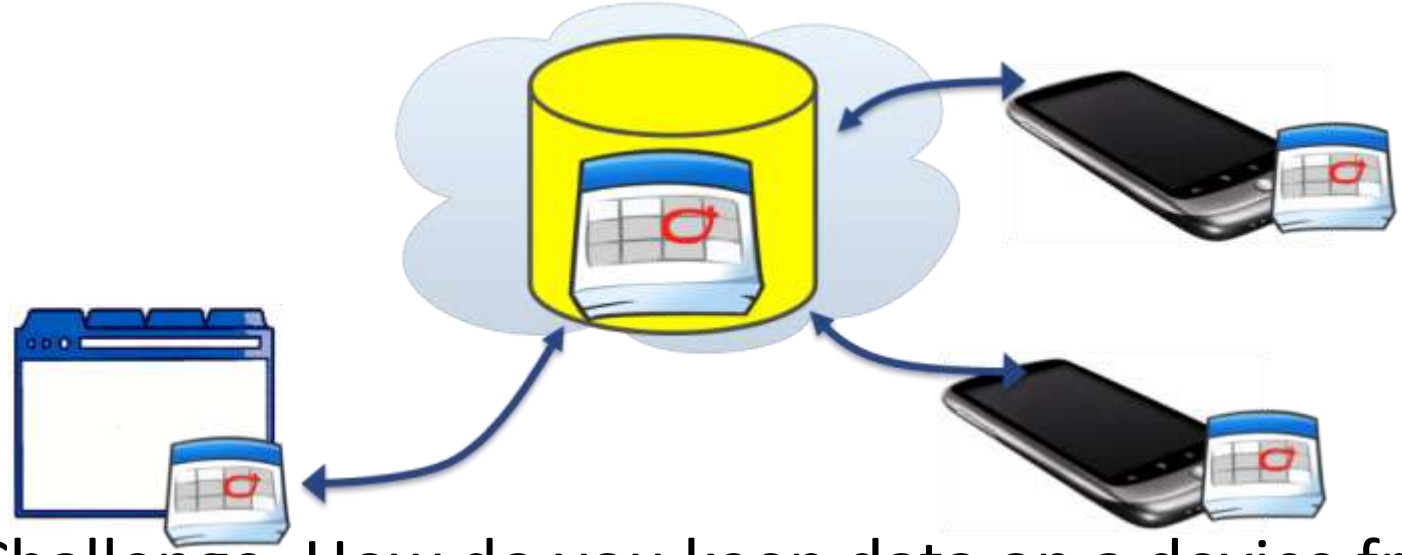
switch back to short polling rate after a new update

Solution

Mobile-Cloud Services (Push notifications)

Recap: Accessing Data in Cloud

- A typical design pattern is that a device updates/receives data in the cloud
  - Cloud as a rendezvous point



- Challenge: How do you keep data on a device fresh?

Recap: Solution Space

- Mobile poll

- Cloud push
  - ❒Each app push
  - ❒Shared (infrastructure) push

Shared Push Service

- A single persistent connection from device to a cloud push service provider
- Multiple application providers push to the service provider
- Service provider pushes to a device using the persistent connection

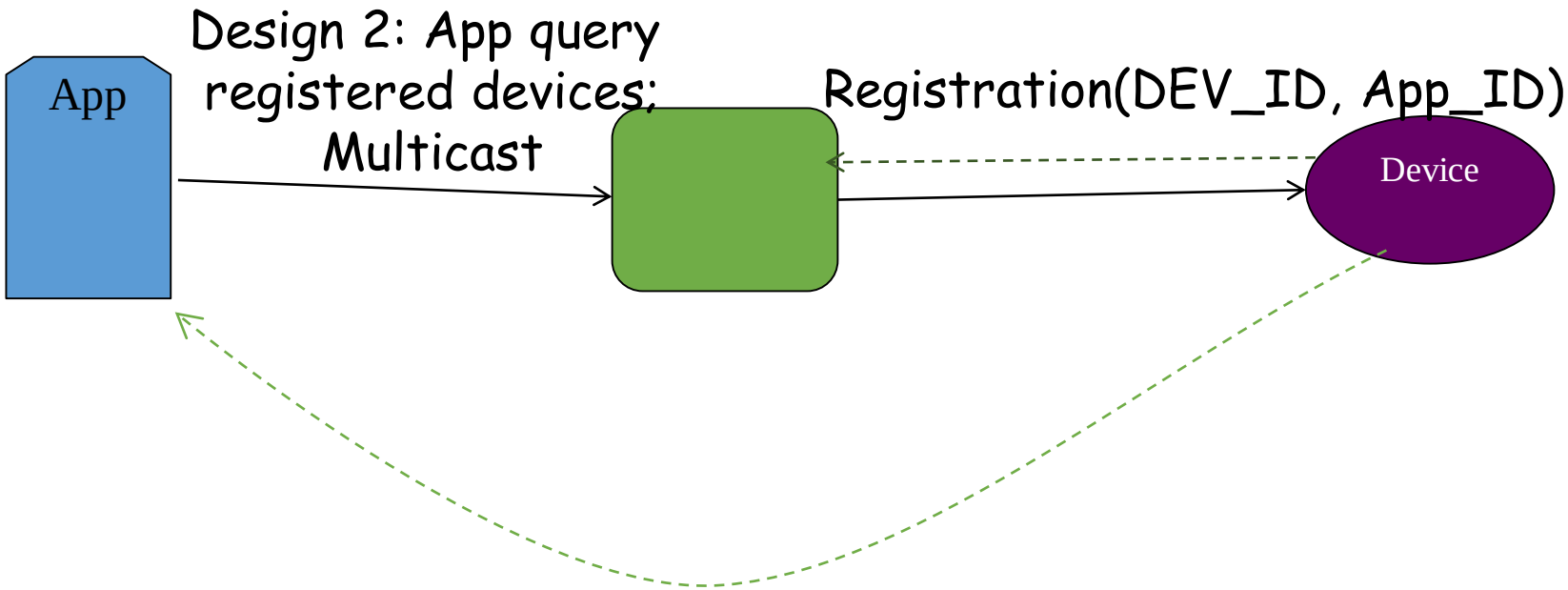
- Two examples
  - Apple Push Notification Service (APNS)
  - Google Cloud Messaging (GCM)

Design Requirements of a Shared Push Service

- Security/Authorization
  - Do not allow arbitrary app to push to a device
- Scalability
  - A large scale system may have millions of clients
- Fault tolerance
  - Client/device, push servers all can fail
- Generality
  - Can be used by diverse applications

Design Point: Authorization

Design 1: App does not know registered devices. Broadcast to all.

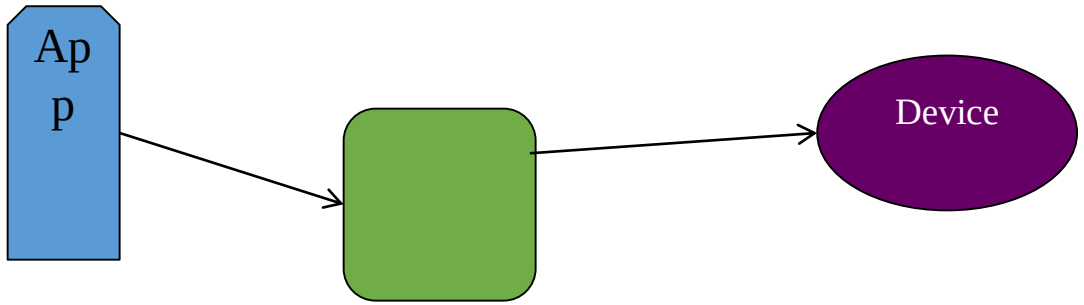


Design 3: Device notifies registration ID to its server;

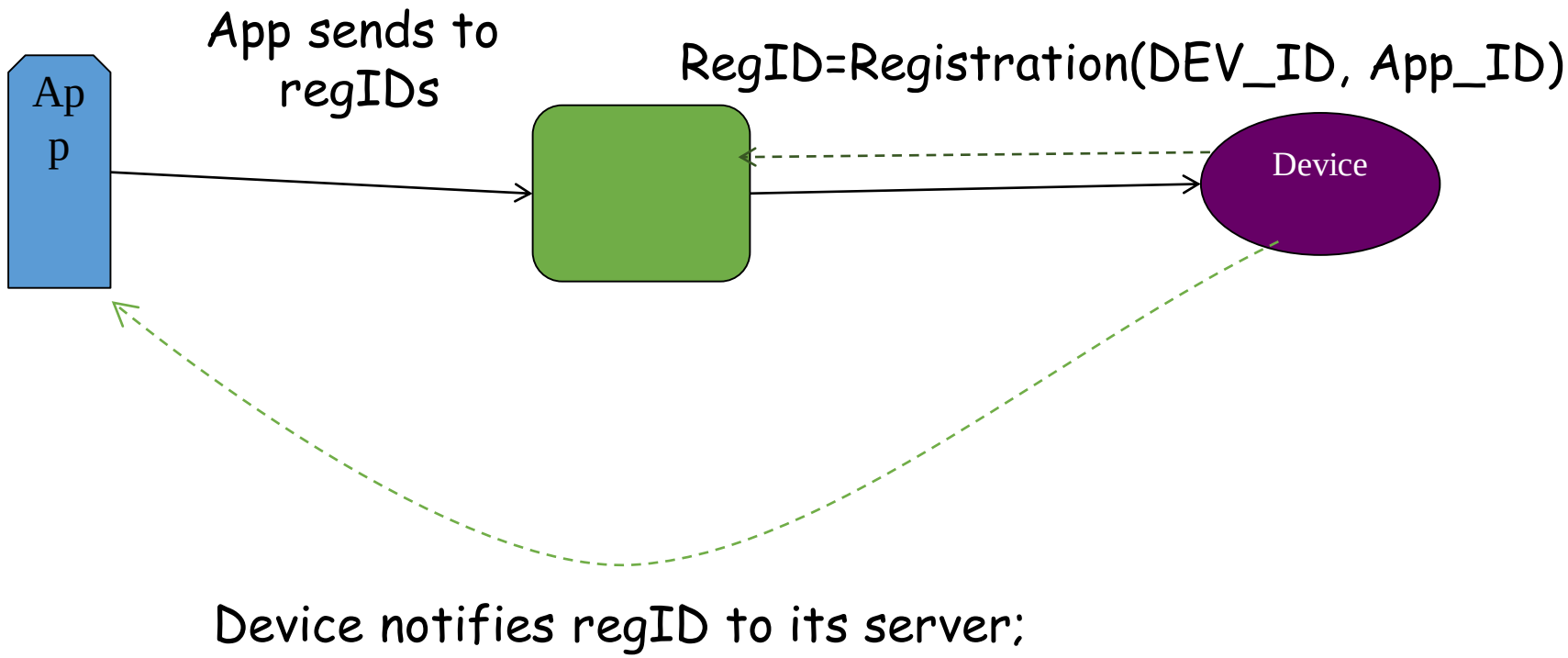
Design Point: What to Push?

- Option 1: Just push signal (data available) to devices and then devices fetch from app servers

- Option 2: push app data

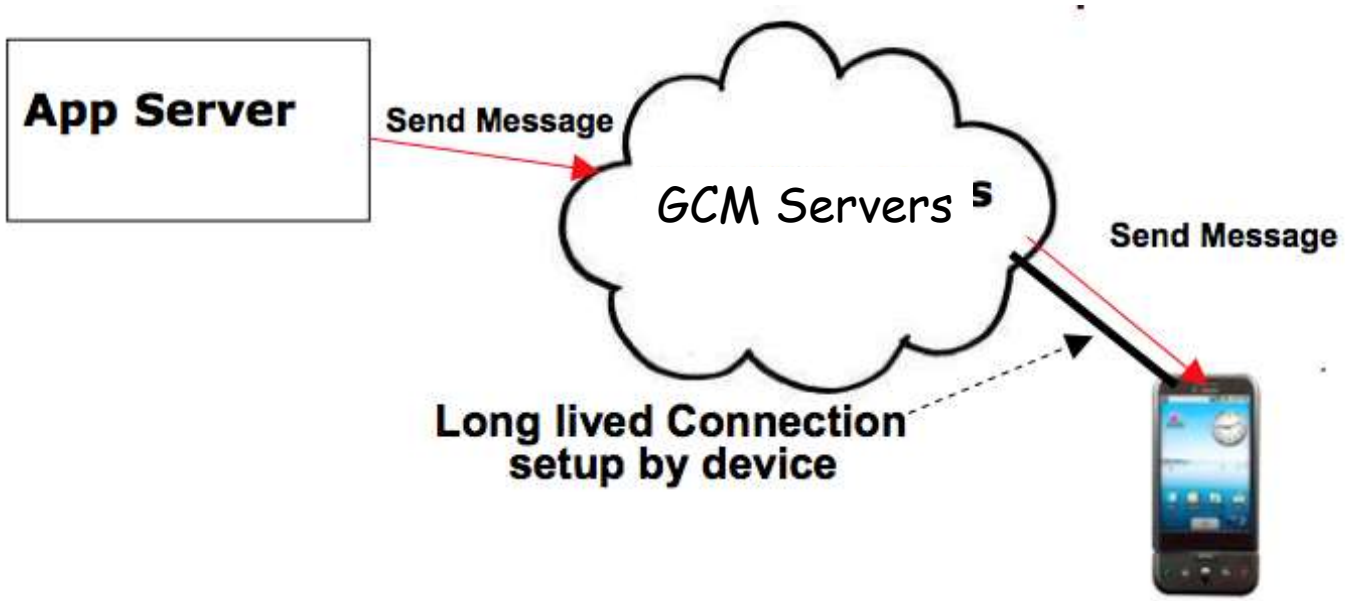


Design Point: Reliability (What Can Go Wrong)



Google Cloud Messaging

- Very similar to APNS



See http://developer.android.com/guide/google/gcm/gs.html
for detailed steps

GCM Flow: App Developer Registration

- App developer registers a project at Google
  - ❒Open API console: https://code.google.com/apis/console/

  - ❒After Cr

Project ID; Sender ID

https://code.google.com/apis/console/#project:4815162342

Summary: GCM Flow

- Enabling cloud to device messaging
  - App (on device) registers with Google to get registration ID
  - App sends registration ID to its App Server
- Per message
  - App Server sends (authenticated) message to Google
  - Google sends message to device, which sends to app
- Disabling cloud to device messaging
  - App can unregister ID, e.g., when user no longer wants push

Google cloud messaging
GCM

Problem

You want to get "push" notifications sent asynchronously from a server, without setting you your own complex infrastructure. This can be used to send short data (up to about 4KB), or to send a "ping" notification which will cause the app to download new data from your server.

Solution

Consider using Google Cloud Messaging (GCM)

Google cloud messaging
GCM

GCM is a free service offered to Android developers to deliver small messages direct to your application running on an Android device.

This avoids your application having to poll a server, which would either;

be not very responsive; why? it takes time to poll the server.

or

very bad for battery life. why? handling the operation poll context drains battery.

Google cloud messaging
GCM

The basic operation of GCM is:

1) The app *registers with GCM* to recieve messages;

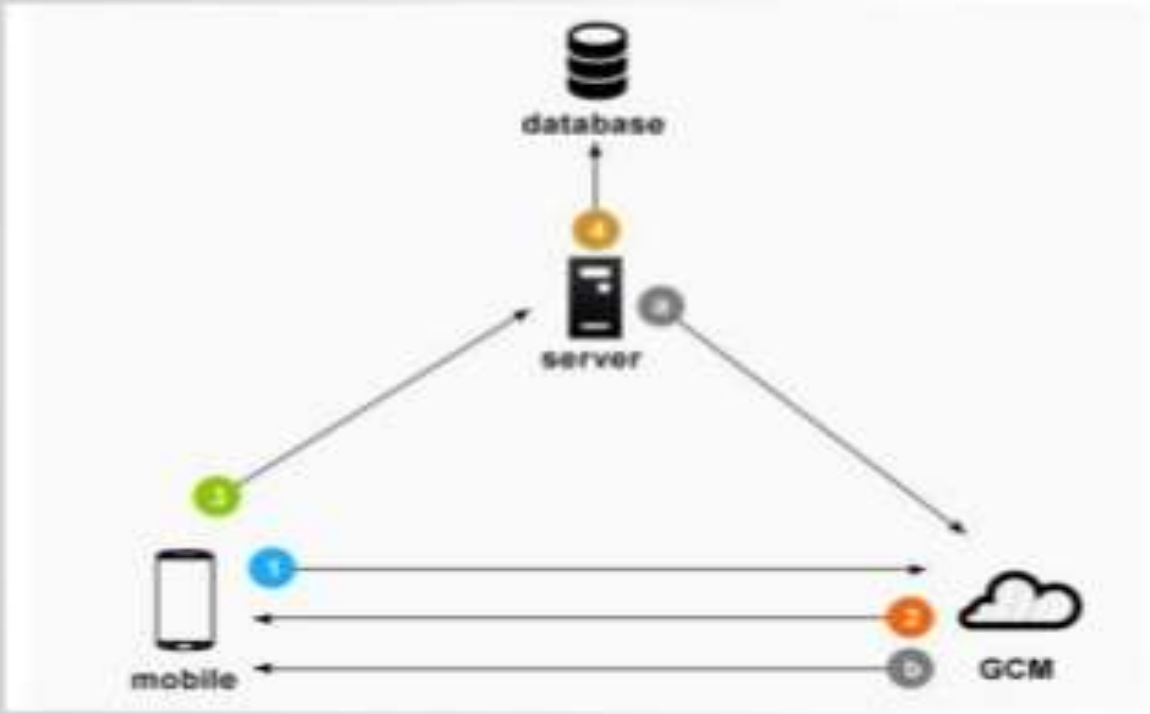
2) Your *server detects* some *condition that requires notifying* the particular *user's device* (eg., new or changed data available) and consequently *sends a message to* the *GCM* server.

3) The *GCM* server *sends* the message *to* your *user's device*, where it is *passed to* your app at a *BroadcastReceiver*

4) You do something with the information.

Google cloud messaging
GCM

GCM workflow



1. Sends sender ID, app ID to GCM server
2. GCM server sends registration ID to device
3. Device will send registration ID to app server
4. Server stores registration ID to DB

a. When push notification is needed, server sends push msg to GCM server with ID

b. GCM server will deliver that message to target mobile device using the device registration ID

Google cloud messaging
GCM

The basic steps are:

1) Sign up with Google to use GCM
2) Configure your client's AndroidManifest.xml
3) Create a BroadcastReceiver to handle the incoming notifications
...
4) Configure your back-end server to notify the GCM server when it has data to send
or push2sync:

send a notice to tell the client to download new data, a form of distributed MVC or push to synchronize model of update or notification.

Google cloud messaging
GCM

==== Sign up with Google to use GCM

Assume that you have a Google developer account (if not, Sign up).

Go to your Developer Console @ :

https://console.firebase.google.com/

*** Create a Project.**

*** Note down: the *Server key* and the *Sender ID***

You can find them later @ :

https://console.firebase.google.com/project***/PROJECTNAME/***settings/cloudmessaging

Google cloud messaging
Exemple Application
Downstream messaging

Downstream messaging **scenario:**

From server -----> to -----> client *via* GCM

App = Client + Buisness Server + GCM server

Client = Activity + Broadcast receiver + Service

Buisness Server = Java class = *our server*

Downstream messaging GCM

Client

Activity

Register with GCM

```
public class GcmMainActivity extends Activity {
        // Everything you already know
        final String registrationId
        private GoogleCloudMessaging gcm;
    protected void onCreate(Bundle savedInstanceState) {
    // Everything you already know
    gcm = GoogleCloudMessaging.getInstance(context);
    registrationId = gcm.register(SENDER_ID);
}
```

Downstream messaging GCM

Client Activity

Register with GCM

```
public class GcmMainActivity extends Activity {
        // Everything you already know
            final String registrationId
            private GoogleCloudMessaging gcm;
        protected void onCreate(Bundle savedInstanceState) {
        // Everything you already know here.....
        gcm = GoogleCloudMessaging.getInstance(context);
        registrationId = gcm.register(SENDER_ID);

        // Blocking operation  Why?
}
```

Downstream messaging GCM

Client

Activity

Register with GCM

gcm.register(SENDER_ID)

is **Blocking**

it happens *in the cloud*

==>

Asynctask to the rescue

Downstream messaging GCM

Client

Activity

Register with GCM

```
  private void registerWithGCMInBackground() {
   new AsyncTask() {
      @Override
      protected String doInBackground(Void... params) {
                gcm = GoogleCloudMessaging.getInstance(context);
                registrationId = gcm.register(SENDER_ID);
                sendRegistrationIdToBackend(); // your own implementation
         }.execute(null, null, null);
   ...
}
```

Downstream messaging GCM

Client

Activity

Register with GCM

```
public class GcmMainActivity extends Activity {
        // Everything you already know

            final String registrationId
            private GoogleCloudMessaging gcm;

        protected void onCreate(Bundle savedInstanceState) {
        // Everything you already know here.....

        registerWithGCMInBackground();
}
```

Downstream messaging GCM

Client

Activity

Register with GCM

GCM needs play services availaible

In your app's startup code (e.g., in onCreate() or onResume()), check that Google Play Services are available, with the static method call :

GooglePlayservicesUtil.isGooglePlayServicesAvailable(Context ctx);

Downstream messaging GCM

Client

Activity

Register with GCM

GCM needs play services availaible

```
boolean isPlayServicesAvailable() {
    int ret =

GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);

            if (ConnectionResult.SUCCESS == ret)
            return true;
            return false;
}
```

Downstream messaging GCM

Client Activity

Register with GCM

```
public class GcmMainActivity extends Activity {

        // Everything you already know

            final String registrationId

            private GoogleCloudMessaging gcm;

    protected void onCreate(Bundle savedInstanceState) {

    // Everything you already know here.....

if (isPlayServicesAvailable())

        registerWithGCMInBackground();

else txtView.setText(“ Check for play services' availabality failed”);

}
```

Downstream messaging GCM

Client

Activity

Register with GCM

```
public class GcmMainActivity extends Activity {
    @Override
    protected void onResume() {
        super.onResume();

        if (  ! isPlayServicesAvailable())

txtView.setText(“ Check for play services' availabality failed”);
    }
```

Downstream messaging GCM

Client

BroadcastReceiver

Receive from GCM

```
public class GcmReceiver extends WakefulBroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
        // Recycle "intent" into an explicit intent for the handler
    ComponentName comp =
        new ComponentName(context.getPackageName(), GcmService.class.getName());
    intent.setComponent(comp);
        startWakefulService(context, intent);
        setResultCode(Activity.RESULT_OK);
}
}
```

Downstream messaging GCM

Client

Service

Handle GCM message

```
public class GcmService extends IntentService {
        public GcmService() {
        super(GcmService.class.getSimpleName());
        }
        @Override
        protected void onHandleIntent(Intent intent) {
                String gcmMessage = intent.getExtras().getString("message");
                processGCM_Message(gcmMessage); // your implimentation here
                GcmReceiver.completeWakefulIntent(intent);
        }
}
```

Downstream messaging GCM

Client

AndroidManifest.xml

Downstream messaging GCM

Server (Buisness server)

```
public class GcmBusinessServer {

        /** Confidential Server API key gotten from the Google Dev
Console -> Credentials -> Public API Access -> Key for Android Apps */

        final static String AUTH_KEY = .....;


        final static String regIdFromClientApp = ......;


        final static String POST_URL =
"https://android.googleapis.com/gcm/send";
```

Downstream messaging GCM

Server (Buisness server)

```
public static void main(String[] args) throws Exception {

        final  String[][]  MESSAGE_HEADERS = {
                { "Content-Type", "application/json"},
                { "Authorization", "key=" + AUTH_KEY}
        };
```

Downstream messaging GCM

Server (Buisness server)

```
 /*
the sent json message form: destination + data
{
   "to" : {"regIdFromClientApp"},
   "data" : {
      "Master": "IAO"
      "Module": "Mobile and Cloud"
      "Date":   "02 February 2016"
      "Message":"Prepare for exam! GCM wishes you good luck."
   }
}
*/
```

Downstream messaging GCM

Server (Buisness server)

```
String jsonMessage =
    "{\n" +
    "    \"to\" : {\""+ "regIdFromClientApp" + "\"},\n" +
    "    \"data\" : {\n" +
    "        \"Master\": \"IAO\"\n" +
    "        \"Module\": \"Mobile and Cloud\"\n" +
    "        \"Date\":   \"02 February 2016\"\n" +
    "        \"Message\":\"Prepare for exam! GCM wishes you good luck.\"\n" +
    "    }\n" +
    "}\n";
sendMessage(POST_URL, MESSAGE_HEADERS, jsonMessage);
  }// end of main method
```

Downstream messaging GCM

Server (Buisness server)

```
private static void sendMessage(String postUrl, String[][] messageHeaders,
    String jsonMessage) throws IOException {


  HttpURLConnection conn = (HttpURLConnection) new
URL(postUrl).openConnection();


  for (String[] h : messageHeaders) {
    conn.setRequestProperty(h[0], h[1]);
  }
```

Downstream messaging GCM

Server (Buisness server)

```
System.out.println("Connected to " + postUrl);
    conn.setDoOutput(true);
    conn.setDoInput(true);
    conn.setUseCaches(false);      // ensure response always from server

PrintWriter pw = new PrintWriter(
        new OutputStreamWriter(conn.getOutputStream()));
```

Downstream messaging GCM
Server (Buisness server)

```
 pw.print(jsonMessage);
      pw.close();

      System.out.println("Connection status code " + conn.getResponseCode());
   } //end sendMessage method
}//end GcmBusinessServer class
```

for details see:
https://developer.android.com/reference/java/net/HttpURLConnection