

# Environnement logiciels pour l'informatique mobile

*Mobile applications & Cloud Computing*

Nicolas Ferry (SINTEF)

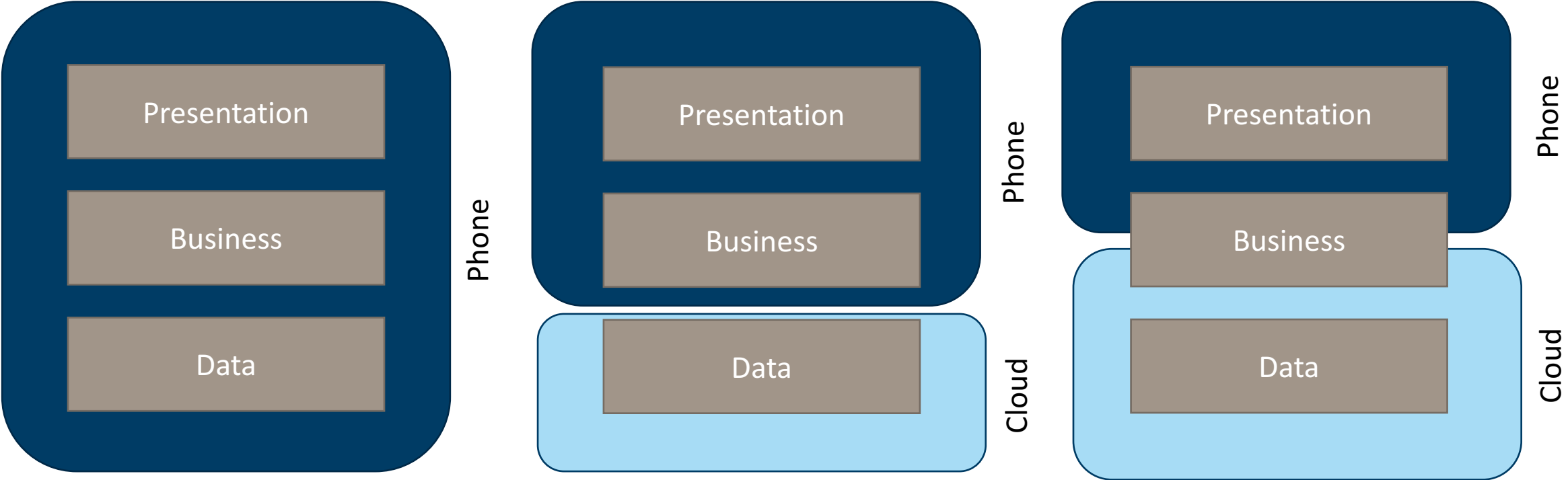
10<sup>th</sup> January 2018



- Trondheim & Oslo, Norway
- Largest research organization in Scandinavia



# Mobile apps & Cloud Computing



# Cloud computing

*“A computing model enabling ubiquitous network access to a shared and virtualised pool of computing capabilities (e.g., network, storage, processing, and memory) that can be rapidly provisioned with minimal management effort”*

--source: NIST



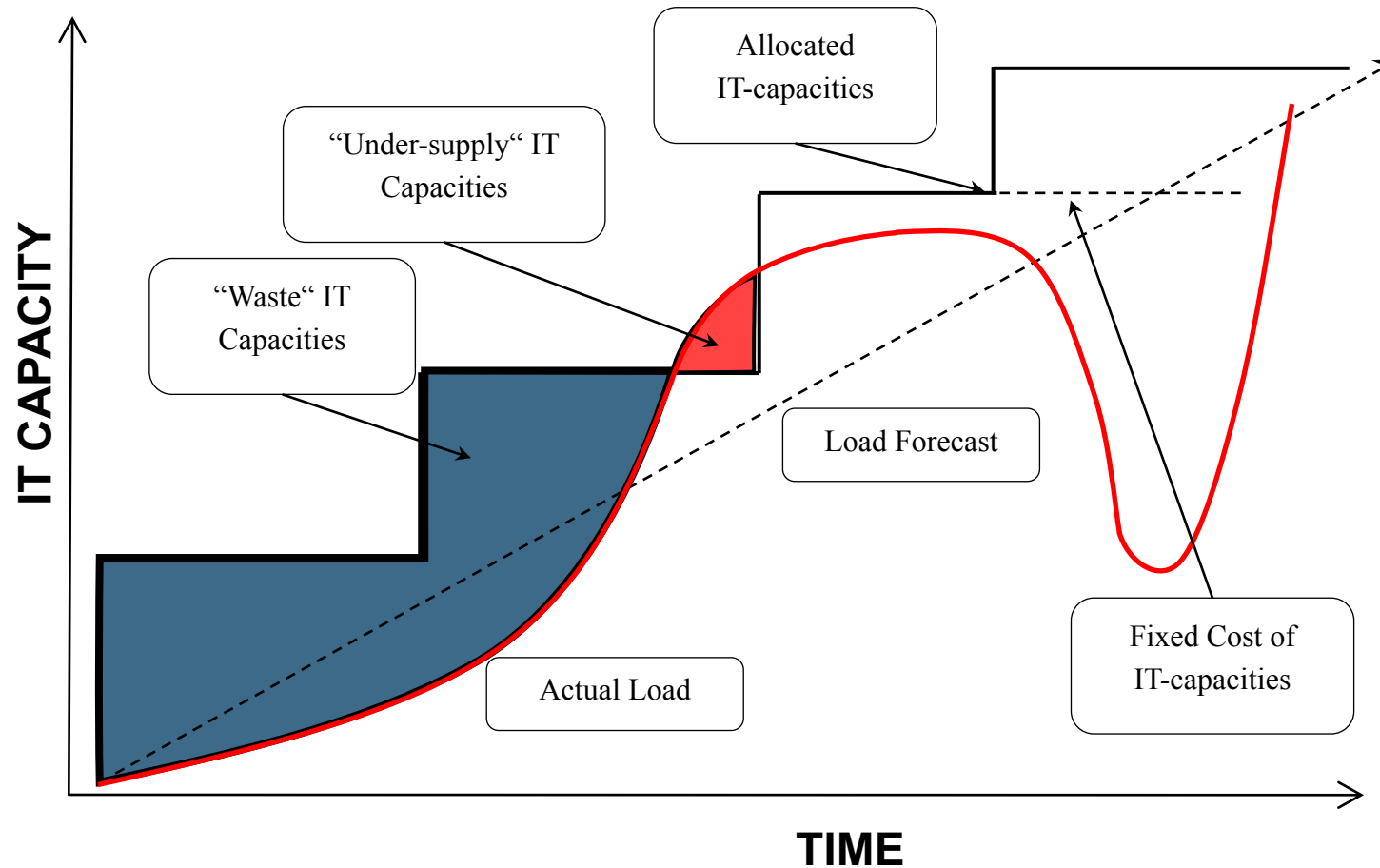
<http://youtu.be/QJncFirhjPg>

# Cloud computing in short

---

- **Large-scale and accessible on demand resources**
  - Network
  - Storage
  - Compute
  - Software
- **Available** via Web service calls **through** the **Internet**
- Short- or long-term access on a **pay per use** basis

# Optimize IT capacity to the load



# Elasticity and Scalability

---

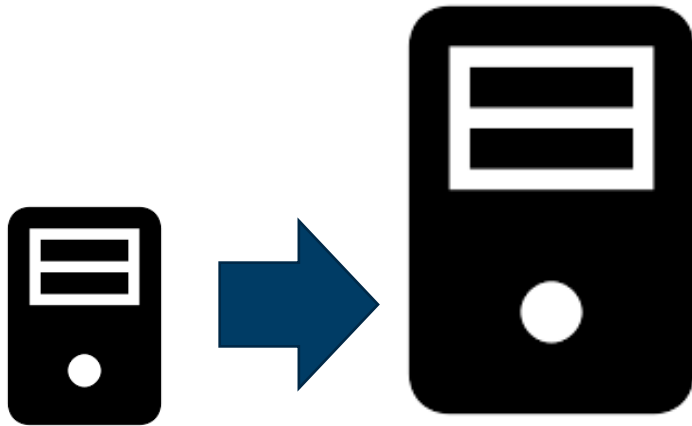
- **Scalability:** the ability of a service to sustain variable workload while fulfilling quality of service (QoS) requirements, possibly by consuming a variable amount of underlying resources.
- **Elasticity:** the ability of a service to rapidly provision and deprovision underlying resources on the fly.

**One does not guarantee the other!**

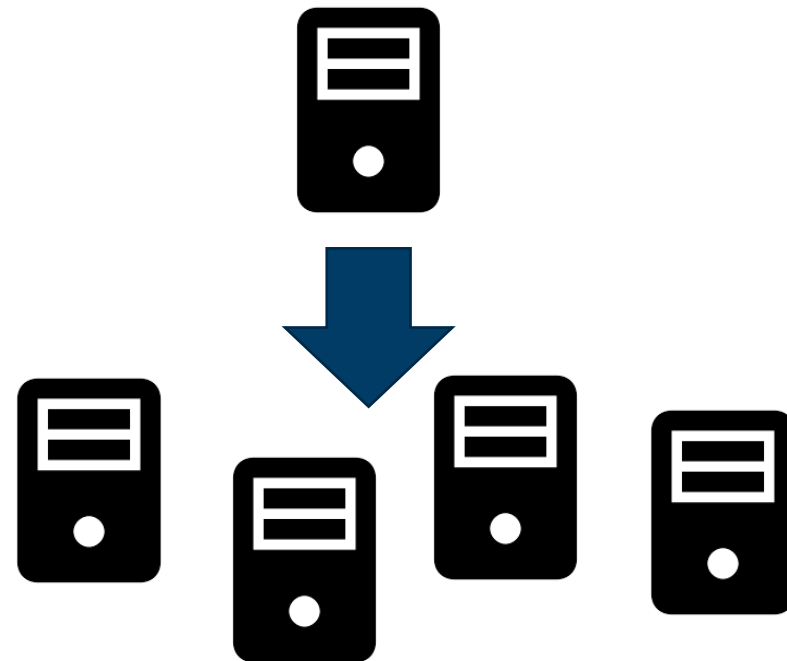
# Scalability

---

Vertical



Horizontal





# Benefits and challenges

---

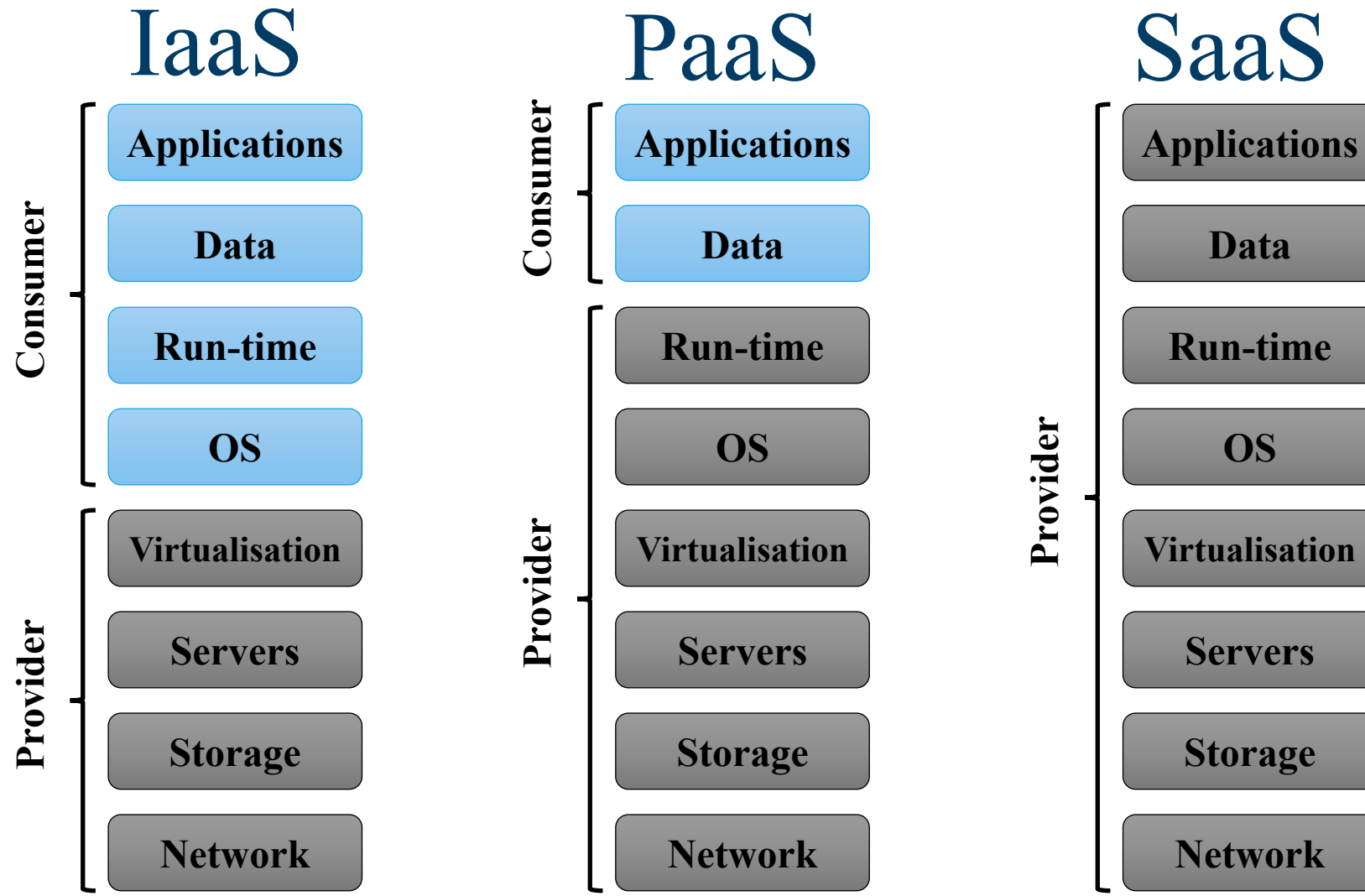
## Benefits:

- Scalability
- Performances
- Availability
- Cost?

## Challenges:

- Interoperability
- Vendor lock-in
- Legal aspects (e.g., data location, ownership etc.)
- Predictability
- Self-adaptation

# The cloud computing stack



# Deployment model

---

- **Private Cloud**

- Owned by the organization. Said to be more secure as the storage and processing stays under the organization control
- E.g., OpenStack, Cloud Foundry

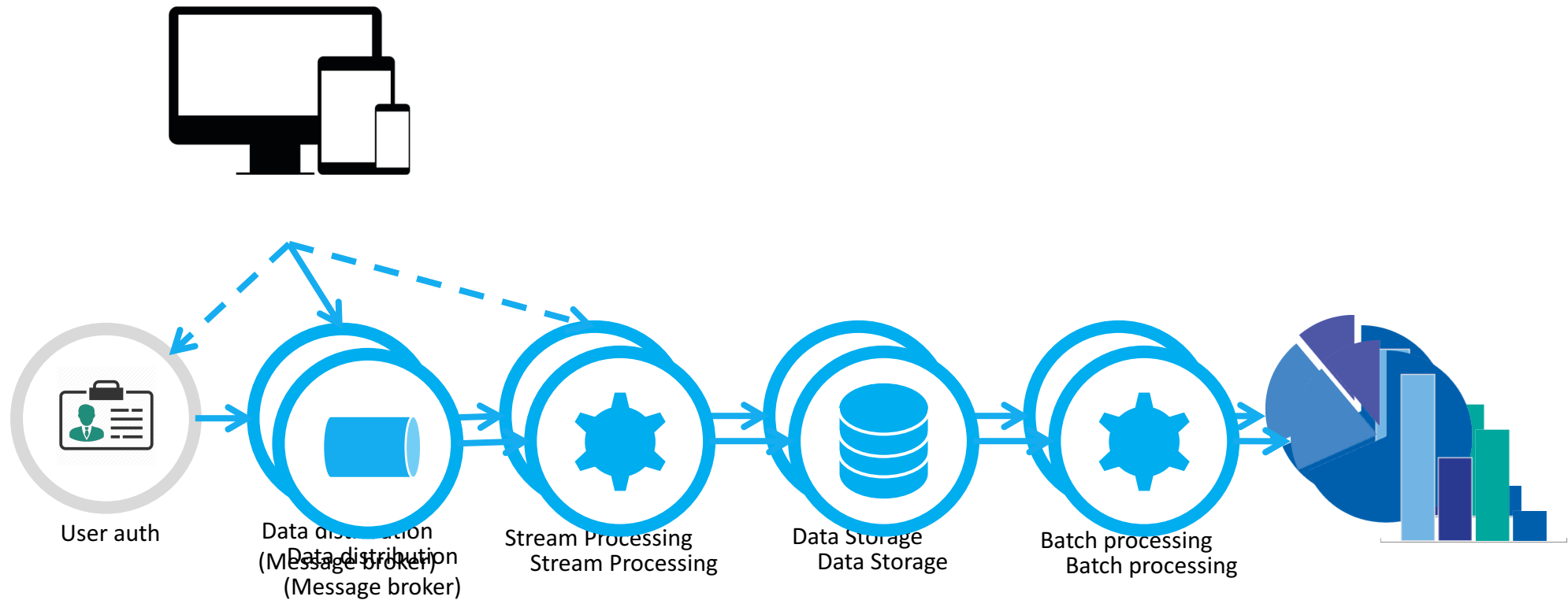
- **Public Cloud**

- Hosted at the provider premises, who is in charge of its maintenance and management
- E.g., AWS, Azure

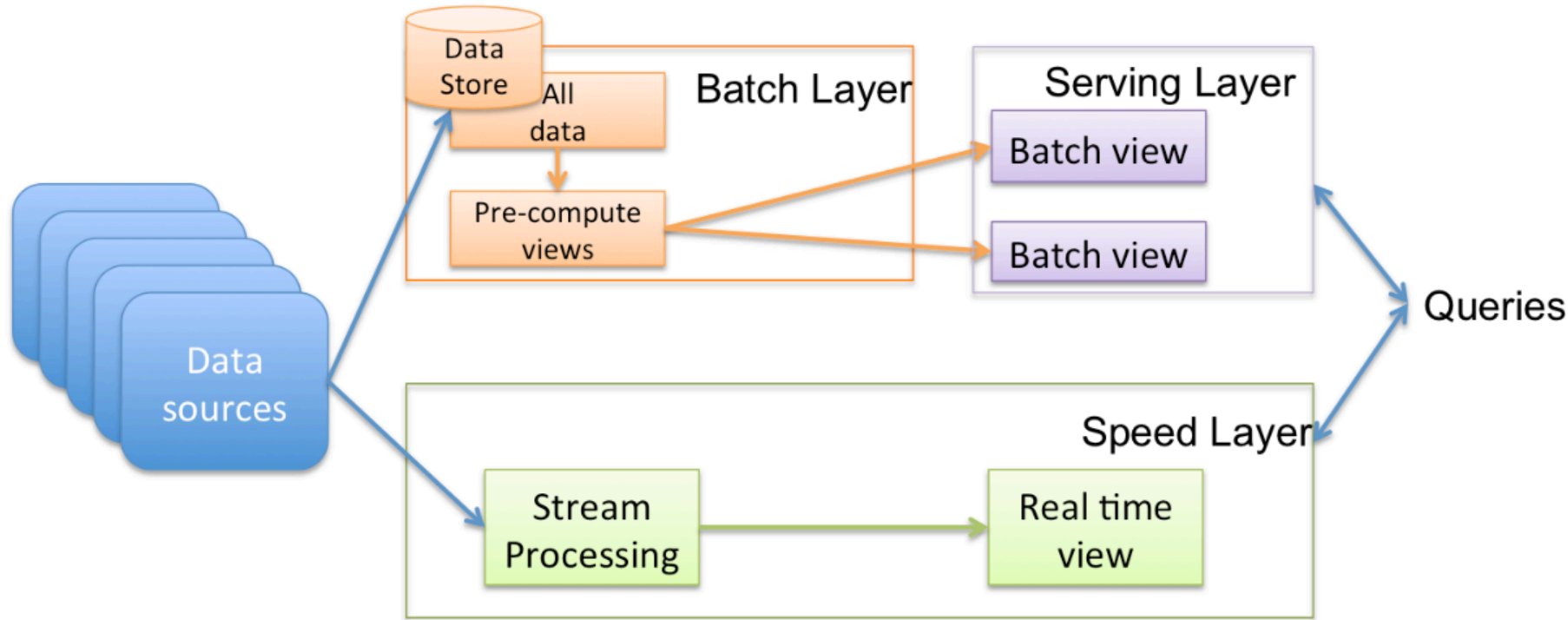
- **Hybrid Cloud**

- Composition of two or more public and private cloud

# Typical Pipeline

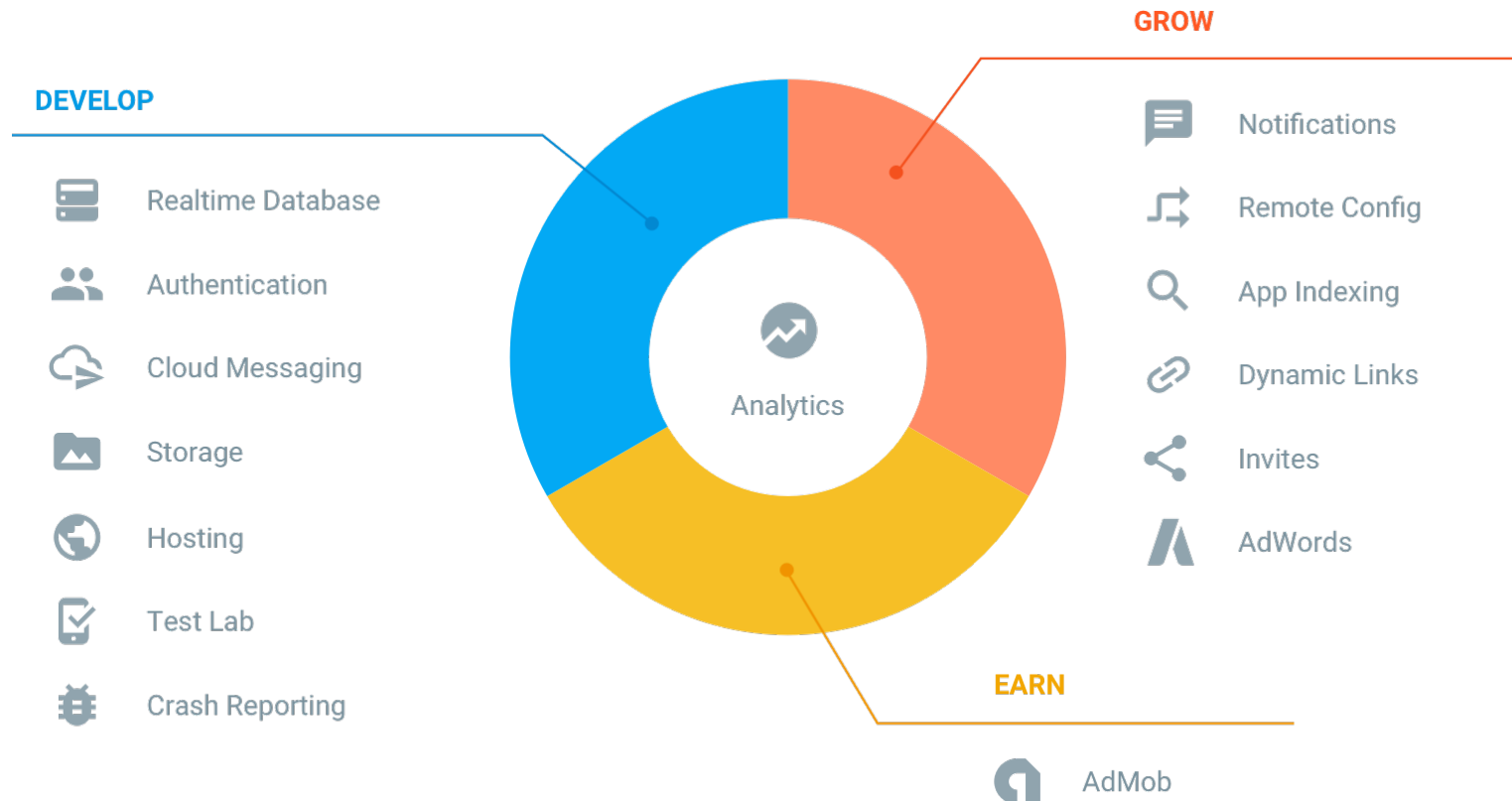


# Classical architecture



# Some cloud services

- Firebase(<https://codelabs.developers.google.com/codelabs/firebase-android/#0>)

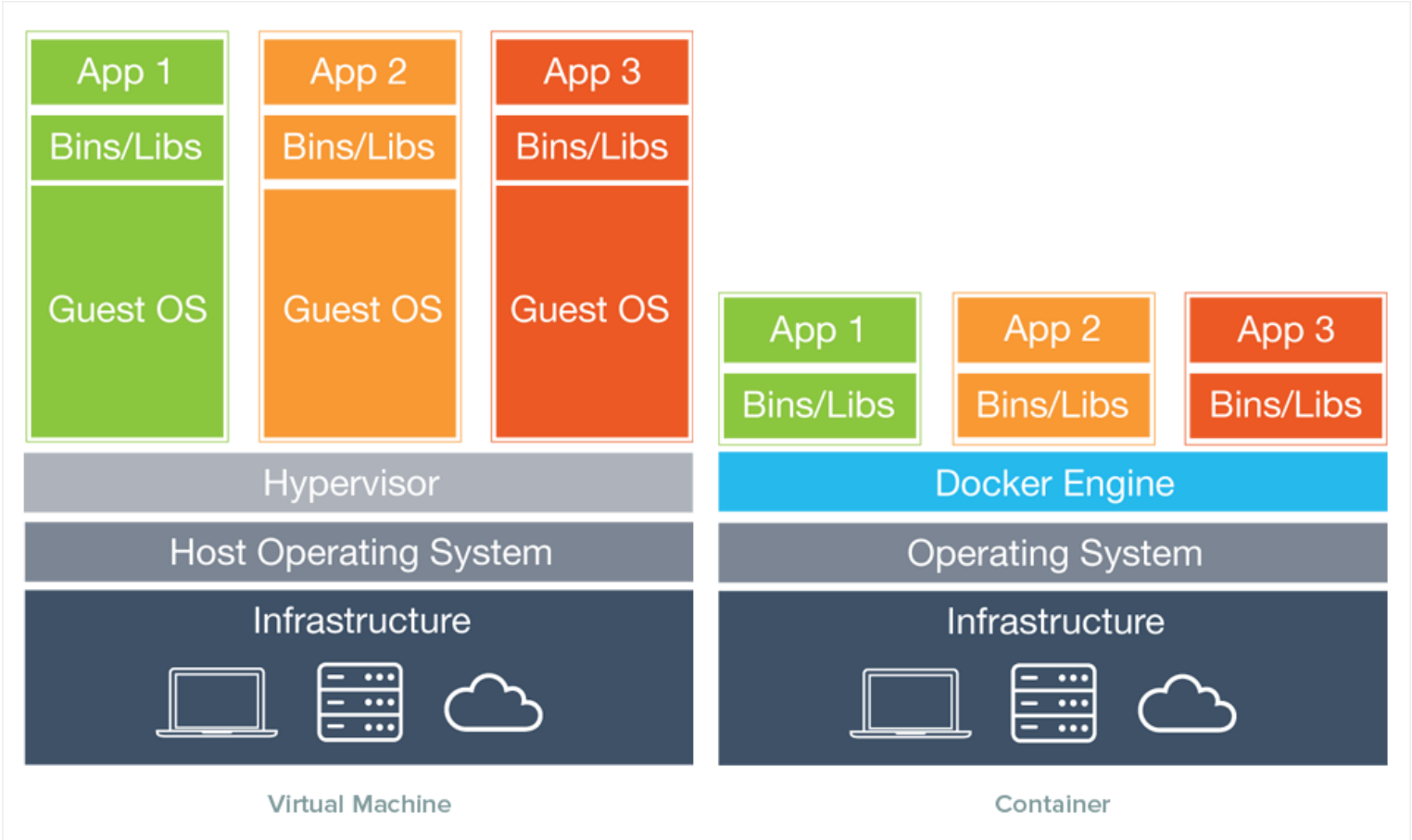


# Some cloud services

---

- **AWS Device Farm**
  - Test your app against mobile devices in the cloud!
- **AWS S3**
  - File storage
- **AWS DynamoDB**
  - No-SQL database
- **AWS Pinpoint**
  - Push notifications
- **AWS Cognito**
  - Authentication service
- **AWS IoT**
  - Software suite for building IoT apps

# Containers

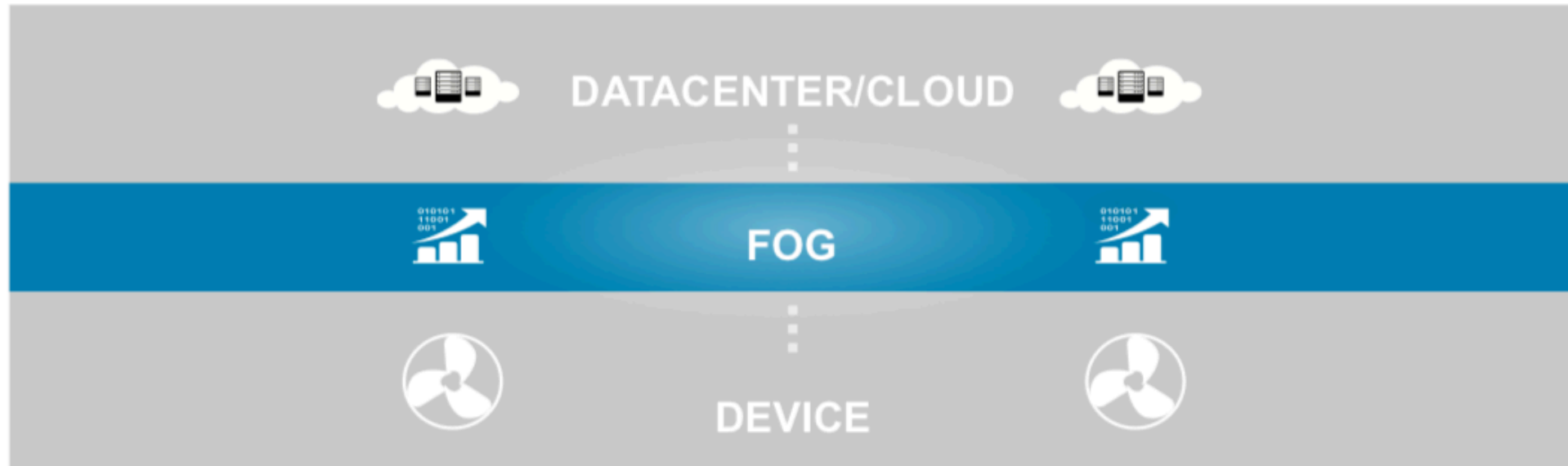


<http://blog.infinet.sh/the-missing-piece-in-containers-storage/>



# Fog Computing

---



[Cisco] Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are

# Why fog computing?

---

- Location aware
- Geographical distribution
- Mobility
- Large number of nodes
- Low latency

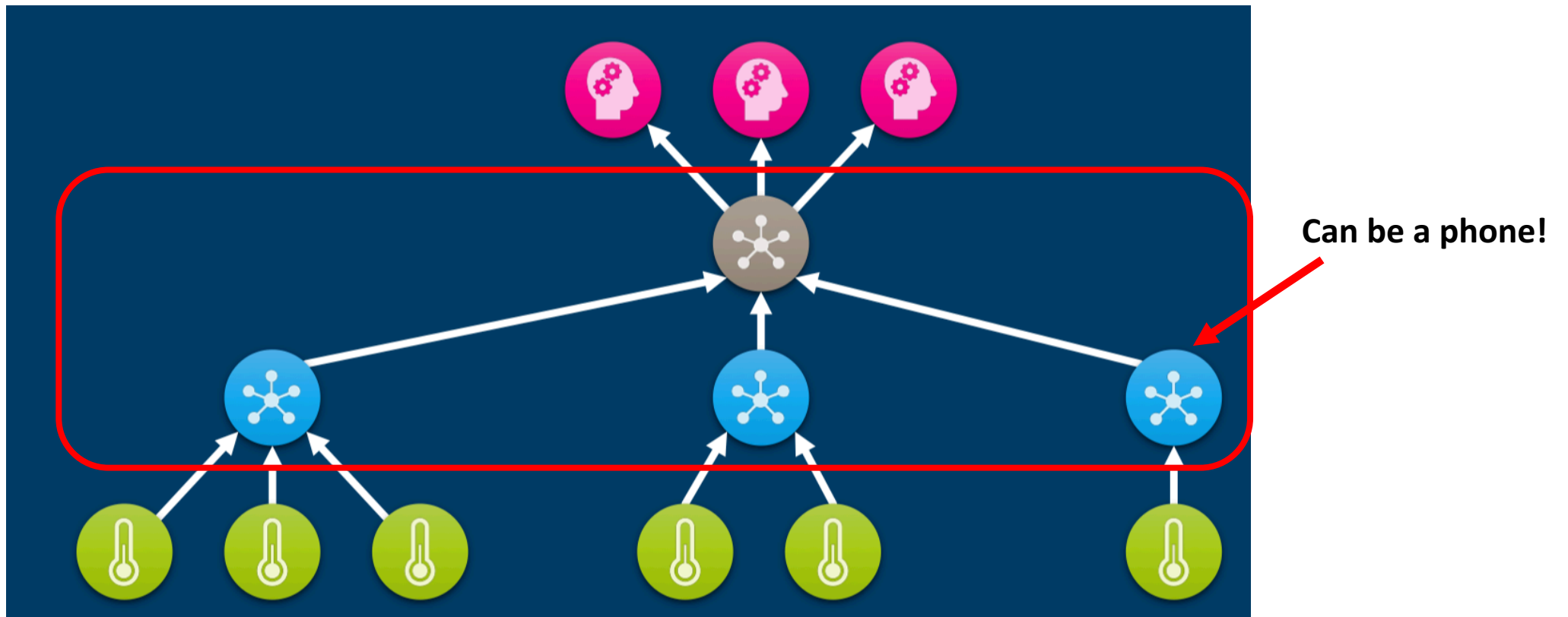
## When to Consider Fog Computing

- Data is collected at the extreme edge: vehicles, ships, factory floors, roadways, railways, etc.
- Thousands or millions of things across a large geographic area are generating data.
- It is necessary to analyze and act on the data in less than a second.

*--source: cisco*

# Smartphones?

---



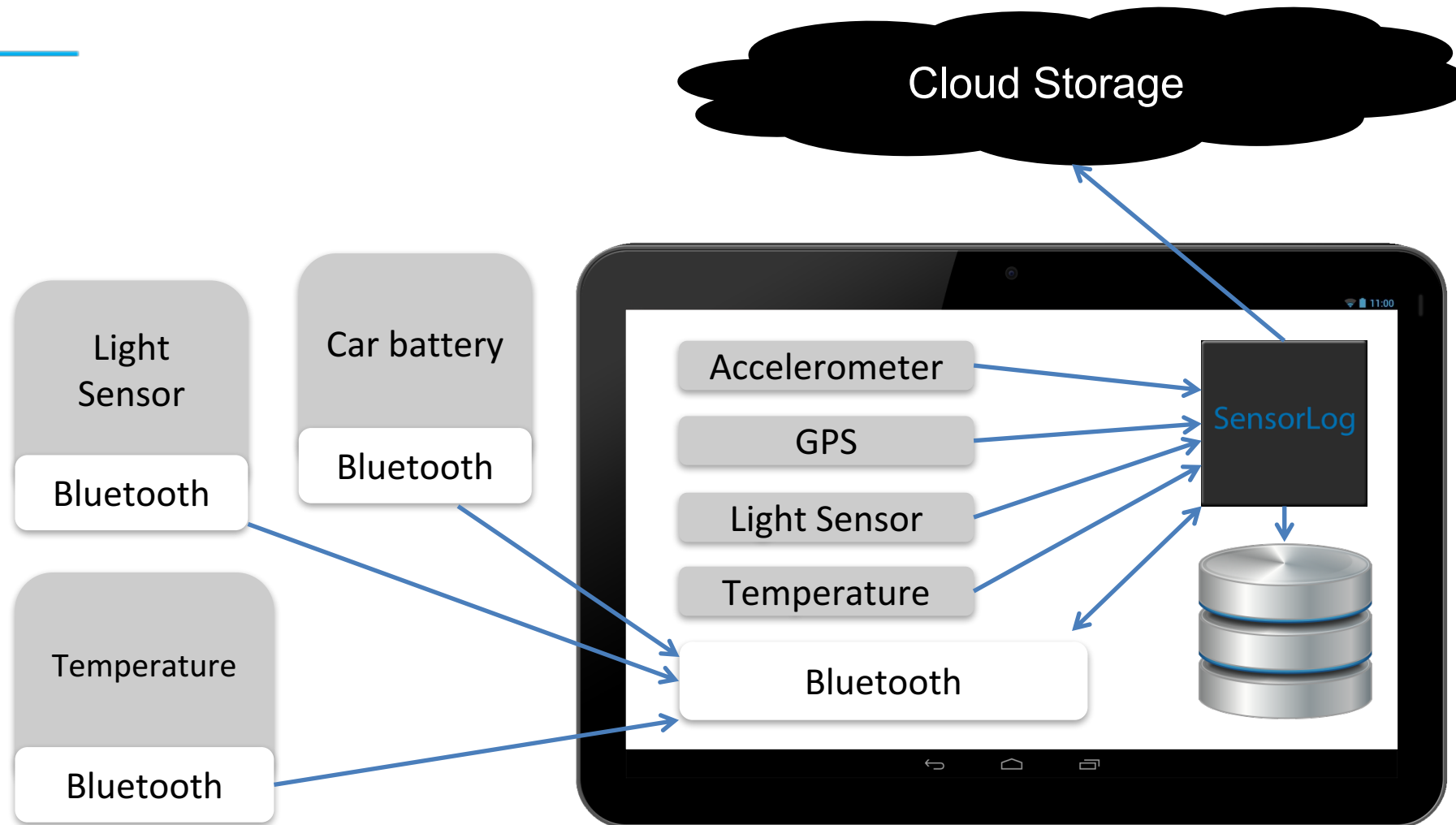
# Crowd sourcing: CITI-SENSE

---



- **FP7 EU project**  
(<http://www.citi-sense.eu>)
- **Objective:** Build sensor-based observatory communities for Improving quality of life in cities
- **One scenario:**
  - Equip people with sensors and use their smartphones as a gateway to upload the measurements in the cloud for analysis
  - Publish observations (their perception of the environment)

# Architecture

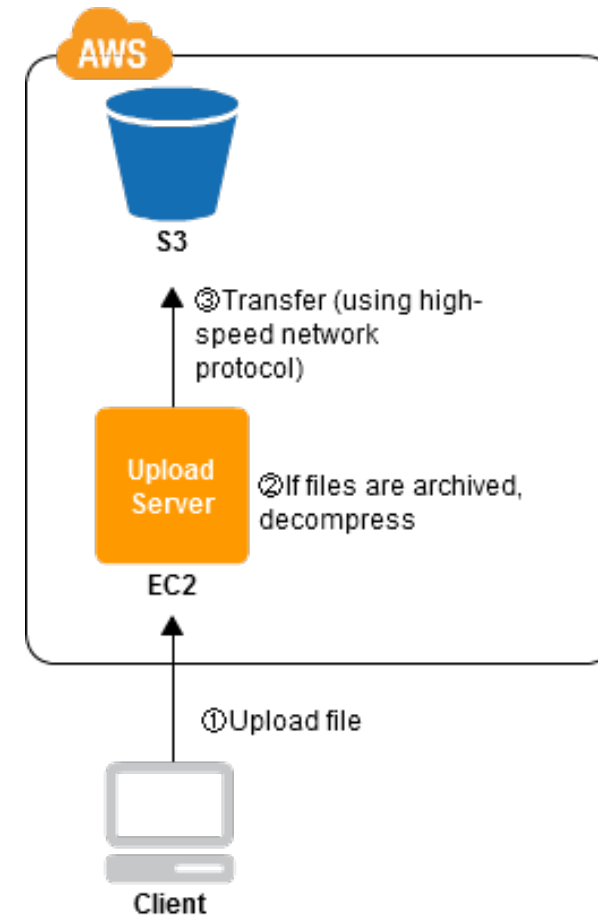


# Server side patterns for improving data uploading performances

---

# Write Proxy Pattern

- **Problem:** Some cloud services require the usage of specific protocols (e.g., HTTP as a communication protocol).  
As a result writing speed can be slow
- **Solution:** pass the data to a proxy first.



# Patterns for data synchronization and storage

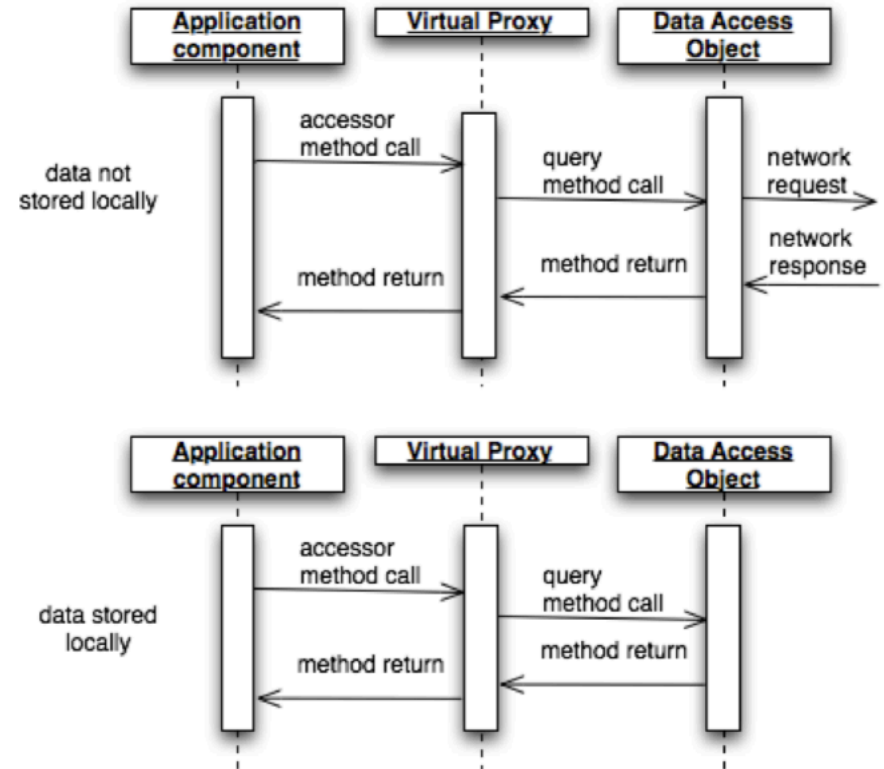
---

- Storage
  - **Partial storage**
  - Complete storage
  
- Synchronization
  - **Asynchronous Data Synchronization**
  - Synchronous Data Synchronization



# Partial Storage

- **Problem:** Network bandwidth and storage space are two vital concerns for mobile application design. Synchronize and store data only as needed to optimize network bandwidth and storage space usage.
- **Solution:** Data is synchronized dynamically “on-demand” by triggers in the application, most typically using a variant of the *Virtual Proxy* pattern.



# Asynchronous data synchronization

---

- **Problem:** Synchronous calls might introduce latency and degrade user experience.

- What might happen on Android:

- Android performs a callback,
- which takes 30s to get a reply,
- before that Android proposes to kill the app

Since version 3.0 d'Android (API 11), it is forbidden to perform network calls from the main thread.

- **Solution:** Perform asynchronous calls

- On Android:
  - AsyncTask
  - Runnable

# AsyncTask

---

- 1. onPreExecute:** Initialisation.
- 2. doInBackground:** This method is started in a Thread and is supposed to be the one that actually does the job. In our context: *open the connection, send the request, retrieve the result and close the connection.*
- 3. onProgressUpdate:** Update about the progress of the job – e.g., to update a progress bar.
- 4. onPostExecute:** This method is called once the AsyncTask is done. In our context: *do something with the result of the request and/or transmit it.*

# AsyncTask

---

- A generic class defined by three generic types, which are Classes! (no -> void, int).
- AsyncTask<Params, Progress, Result>
  - **Params**: Type of parameter of doInBackground
  - **Progress**: Type of parameter of onProgressUpdate
  - **Result**: Type of parameter of onPostExecute
- To be used via sub-classes.

# AsyncTask

---

- To start a task:

```
new myAsyncTask(something).execute("url1.com", "url2.com")
```

- Execute returns void, thus, if the result is not meant to be processed in the `onPostExecute` method, one should use a listener.

# Mixing partial storage and asynchronous transfer for uploading high velocity data

---

- E.g., Smartphone as a gateway that gather data from sensors (internal + external) and upload it on a cloud-based data store.

## In background:

1. Manage connections to sensors
2. Retrieve data and store it locally
3. Provide access to recent data
4. On a regular basis upload chunk of data to the cloud
5. If upload successful remove data from local storage

# TP

---

## 1. Store Files on AWS S3

- Bucket: lecture-epu
- Cognito id:eu-west-1:0cab6e17-78a7-4d97-ac68-77b543f21caa

## 2. In parallel store bulks of sensor data on Firebase realtime db

## 3. In parallel store bulks of sensor data on CouchDB:

<http://XXX:5984>

1. GPS location, plus others
2. Feel free to select the sensor you want

## 4. Add markers on Google Map

- Activate Android Map

# AWS S3

- **Concepts:** Store files in Buckets deployed in a specific region
- Manifest.xml

```
<service android:name="com.amazonaws.mobileconnectors.s3.  
transferutility.TransferService" android:enabled="true" />
```

- Create S3 client & Transfer tool

```
AmazonS3 s3 = new AmazonS3Client(credentialsProvider);  
  
TransferUtility transferUtility = new TransferUtility(s3, APPLICATION_CONTEXT);
```

- Upload/download

```
TransferObserver observer = transferUtility.upload(  
    MY_BUCKET,    /* The bucket to upload to */  
    OBJECT_KEY,  /* The key for the uploaded object */  
    MY_FILE      /* The file where the data to upload exists */  
);
```



# AWS S3

- Check status of Download/Upload

```
transferObserv
```

```
@Override
```

```
public voi
```

```
// do
```

```
}
```

```
@Override
```

```
public voi
```

```
int pe
```

```
//Disp
```

```
}
```

```
@Override
```

```
public void onError(int id, Exception ex) {
```

```
// do something
```

```
}
```

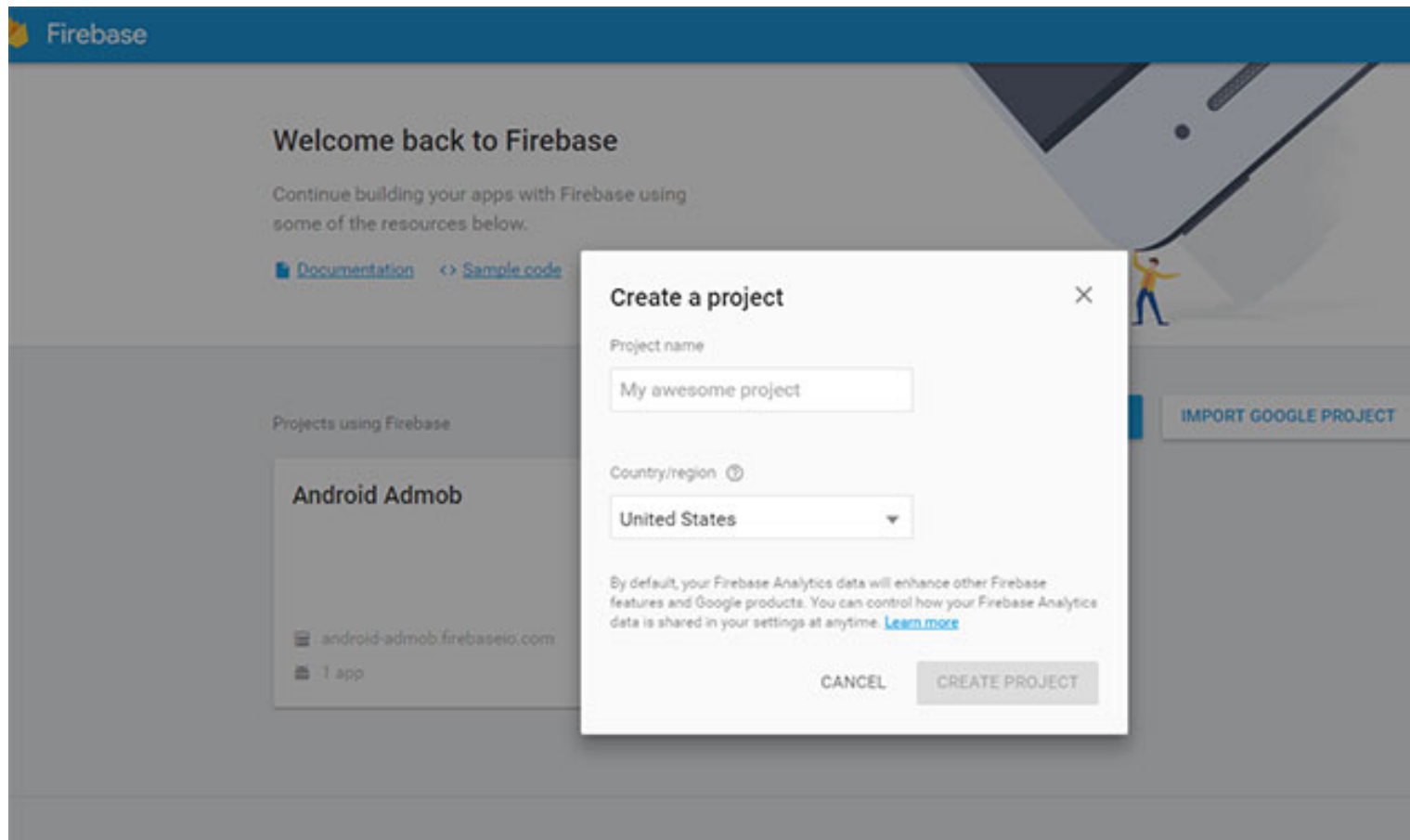
```
});
```

More details on:

<http://docs.aws.amazon.com/mobile/sdkforandroid/developerguide/s3transferutility.html>

# Firebase

---



# Firebase

---

Welcome to Firebase! Get started here.



Add Firebase to  
your Android app



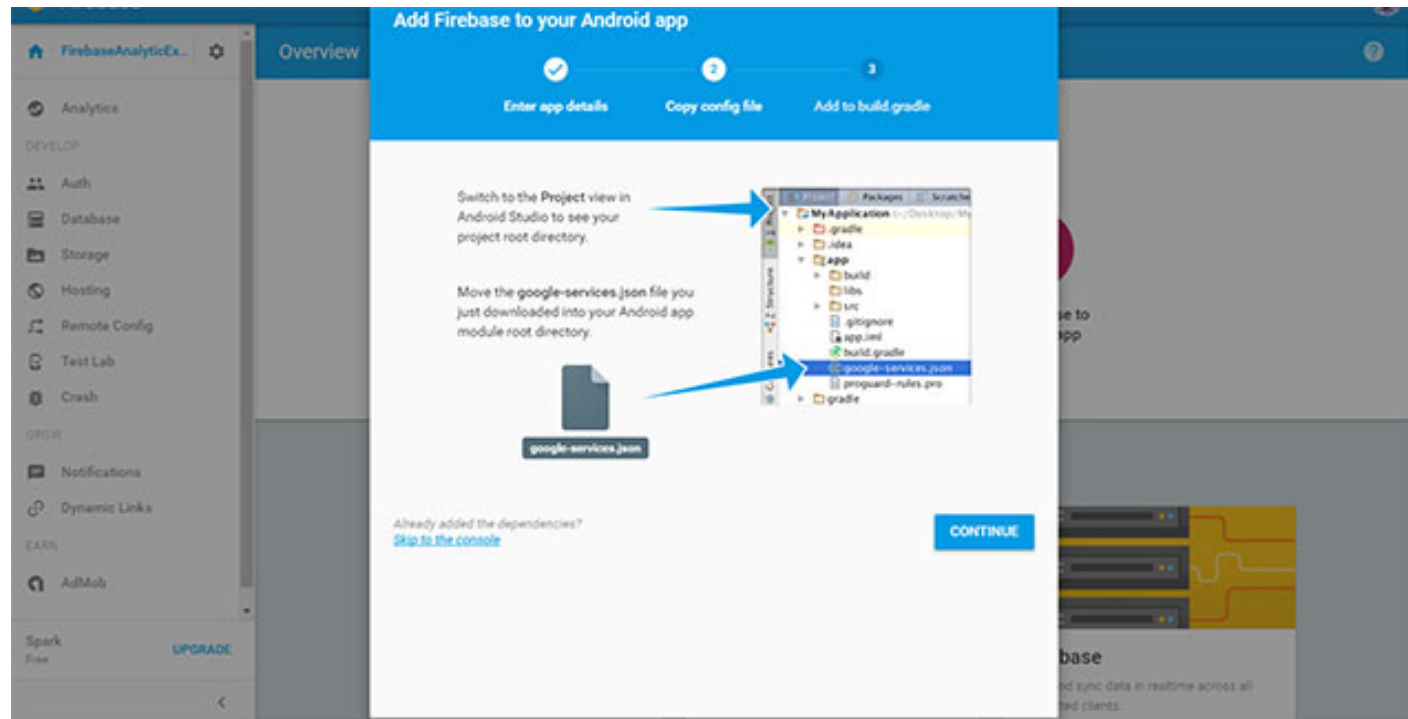
Add Firebase to  
your iOS app



Add Firebase to  
your web app

# Firestore

---



# Firebase

---

**Add Firebase to your Android app**

Enter app details   Copy config file   **Add to build.gradle**

The Google services plugin for [Gradle](#) loads the `google-services.json` file you just downloaded. Modify your build.gradle files to use the plugin.

1. **Project-level build.gradle** (`<project>/build.gradle`):

```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:3.0.0'
  }
}
```
2. **App-level build.gradle** (`<project>/app/build.gradle`):

```
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
includes Firebase Analytics by default
```
3. Finally, press "Sync now" in the bar that appears in the IDE:

Gradle files have changed since last sync. **Sync now**

**FINISH**

# Firestore

---

<https://firebase.google.com/docs/database>

# Google map

- Manifest.xml

```
<uses-feature
  android:glEsVersion="0x00020000"
  android:required="true"
/>

<meta-data
  android:name="com.google.android.maps.v2.API_KEY"
  android:value="AIzaSyDajV01xlQeTCchInMYpvtnIBoGOf8iFM"
/>

<meta-data
  android:name="com.google.android.gms.version"
  android:value="@integer/google_play_services_version"
/>
```

- Get SHA1

- Mac OS/Linux

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

- Windows

```
keytool -list -v -keystore "%USERPROFILE%\android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```