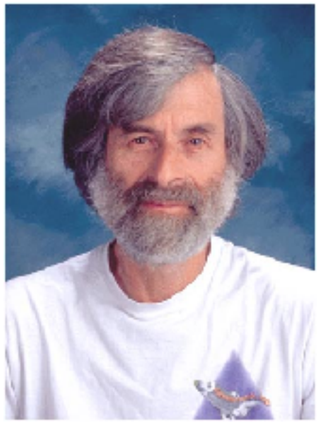
WHAT IS DISTRIBUTION ?

Slides adopted from lectures by Elisa Gonzalez Boix, Wolfgang De Meuter & Tom Van Cutsem

WHAT IS DISTRIBUTION ?

A distributed system is a collection of *independent computers* that appears to its users as a single coherent system.



“A distributed system is a system where I can’t get my work done because a computer has failed that I’ve never even heard of.”
[Leslie Lamport]

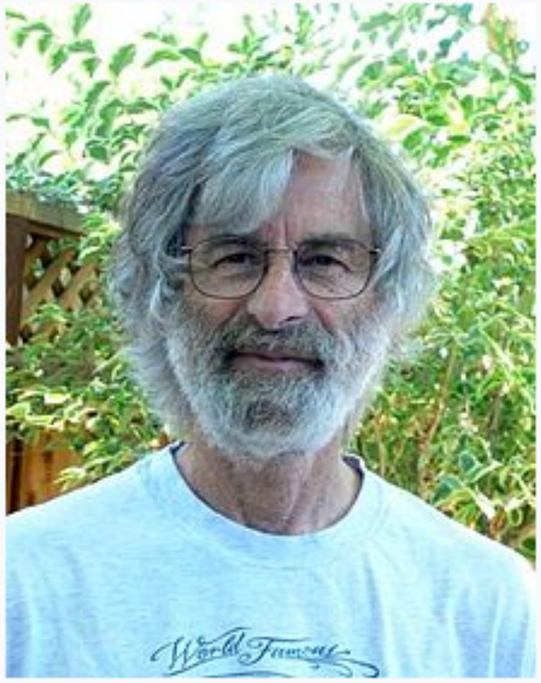
LESLIE LAMPORT

Leslie Lamport

From Wikipedia, the free encyclopedia

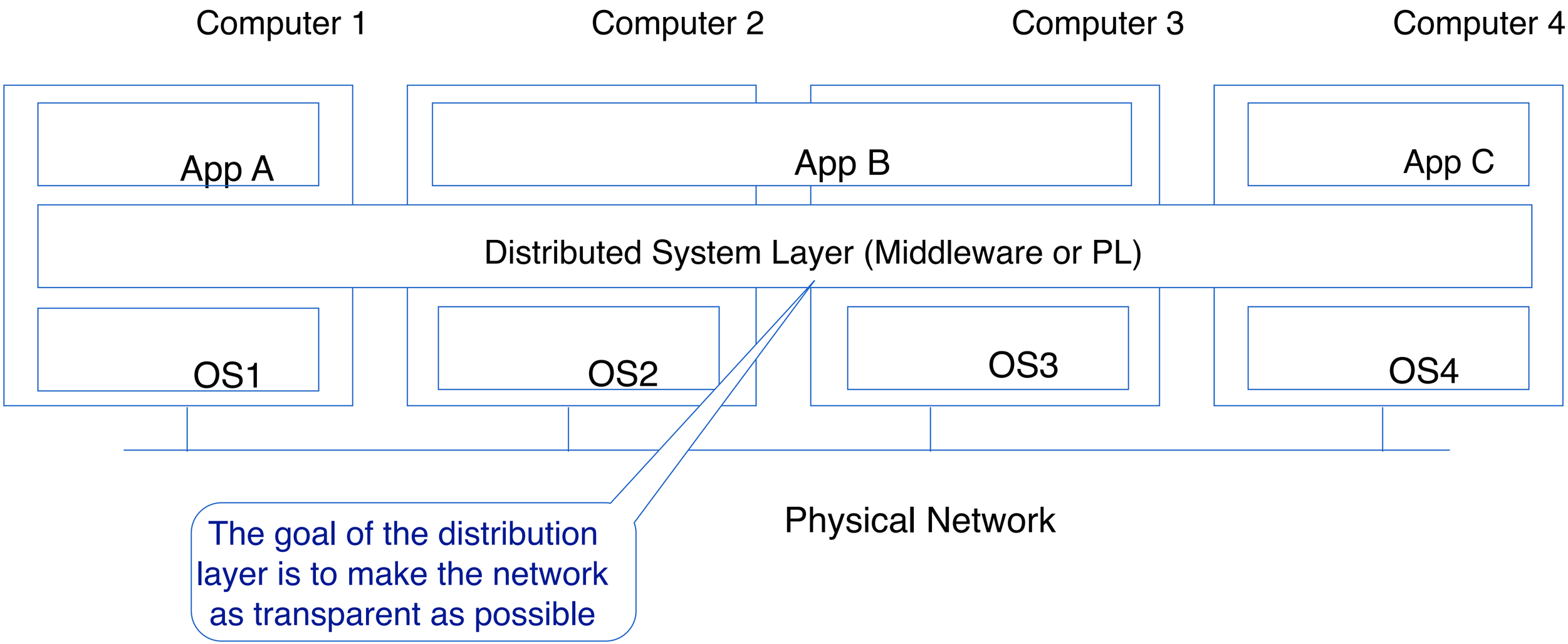
Leslie B. Lamport (born February 7, 1941) is an American computer scientist. Lamport is best known for his seminal work in distributed systems and as the initial developer of the document preparation system LaTeX.[2] Leslie Lamport was the winner of the 2013 Turing Award[3] for imposing clear, well-defined coherence on the seemingly chaotic behavior of distributed computing systems, in which several autonomous computers communicate with each other by passing messages. He devised important algorithms and developed formal modeling and verification protocols that improve the quality of real distributed systems. These contributions have resulted in improved correctness, performance, and reliability of computer systems.[4][5][6][7][8]

Leslie Lamport



Born	February 7, 1941 (age 75) New York City, New York

DISTRIBUTION VS. NETWORKING



UNDERLYING NETWORK: 8 FALLACIES

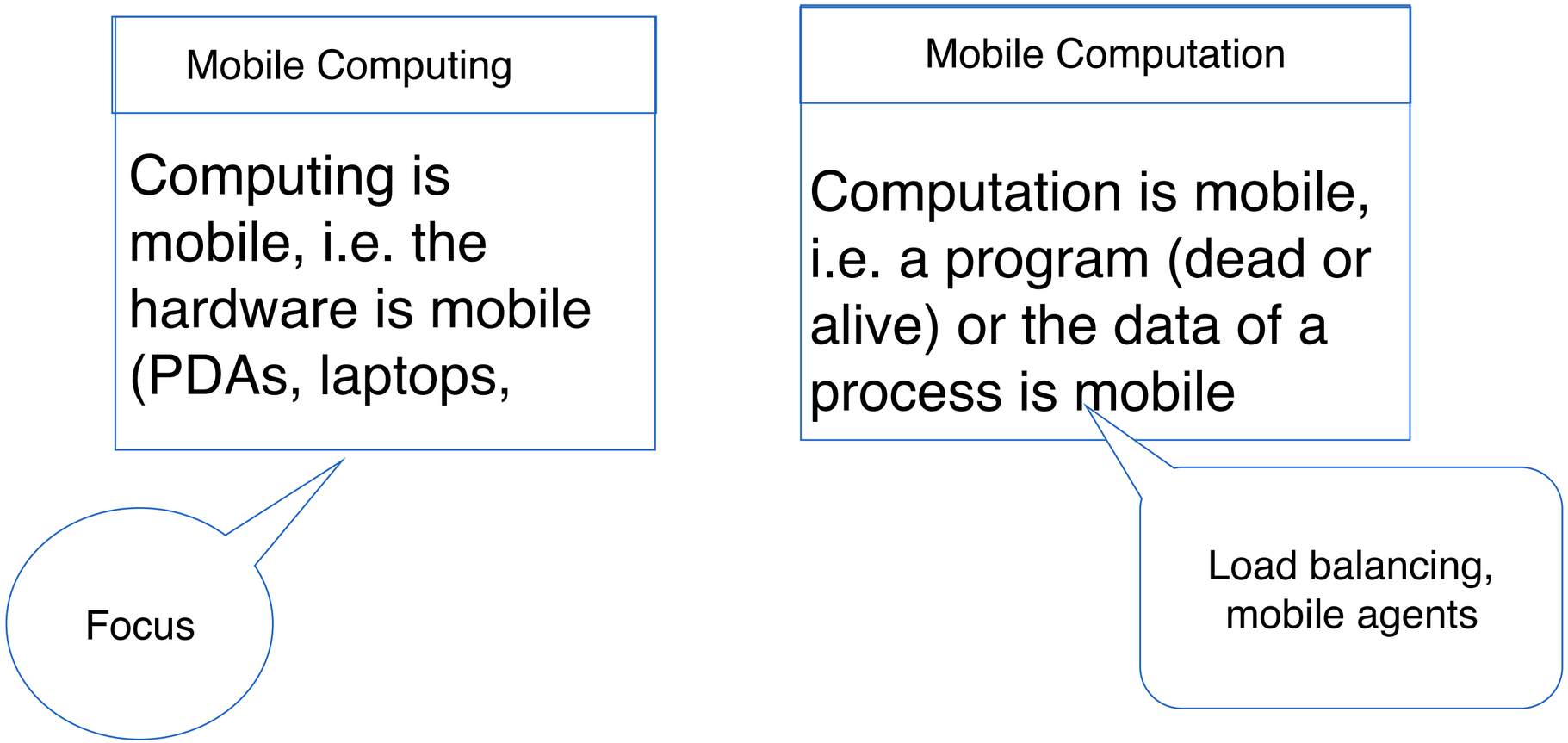
"Misvattingen"

[Peter Deutsch]

1. The network is reliable
2. The network is secure
3. The network is homogeneous
4. The topology does not change
5. Latency is zero
6. Bandwidth is infinite
7. Transport cost is zero
8. There is one administrator

All these assumption are ultimately proven wrong !

DISTRIBUTION VS. MOBILITY



OTHER FORMS OF DISTRIBUTION

Cluster Computing = collection of similar work stations or PCs, closely connected by means of a high-speed LAN and running the same OS

Build your own supercomputer

Grid Computing = federation of computer systems, where each system may fall under a different administrative domain and may be very different w.r.t hardware, software and network topology

E.g. LCG

Cloud Computing = deliver computing as a metered service over some network, e.g. the internet (Software as a Service - SaaS)

E.g. Amazon Web Services

TARGETED DISTRIBUTED SYSTEMS

A mundane distributed system will (probably):

- be wirelessly connected

In your granny's pocket

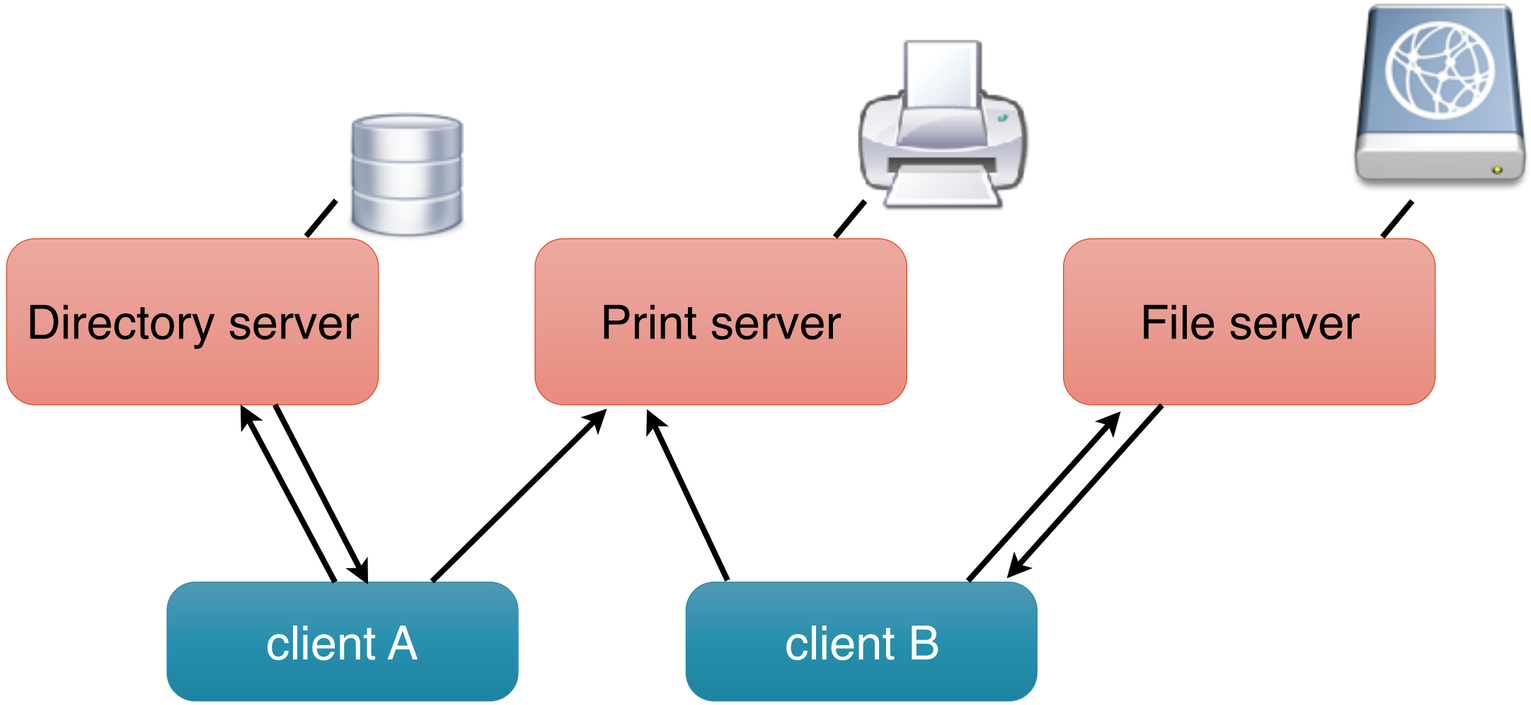
- be mobile
- be more and more collaborative (peer2peer vs. client-server)
- store data in a decentralized way
- have to deal with many many nodes

TOPICS IN DISTRIBUTION

ARCHITECTURAL MODELS

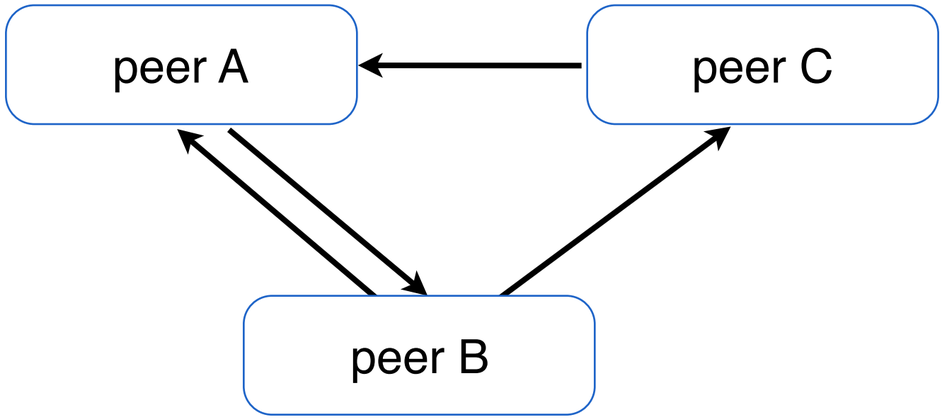
CLIENT-SERVER MODEL

- Environment consists of **clients** and **servers**
  - **Service**: task a machine can perform
  - **Server**: machine that performs the task
  - **Client**: machine that is requesting the service



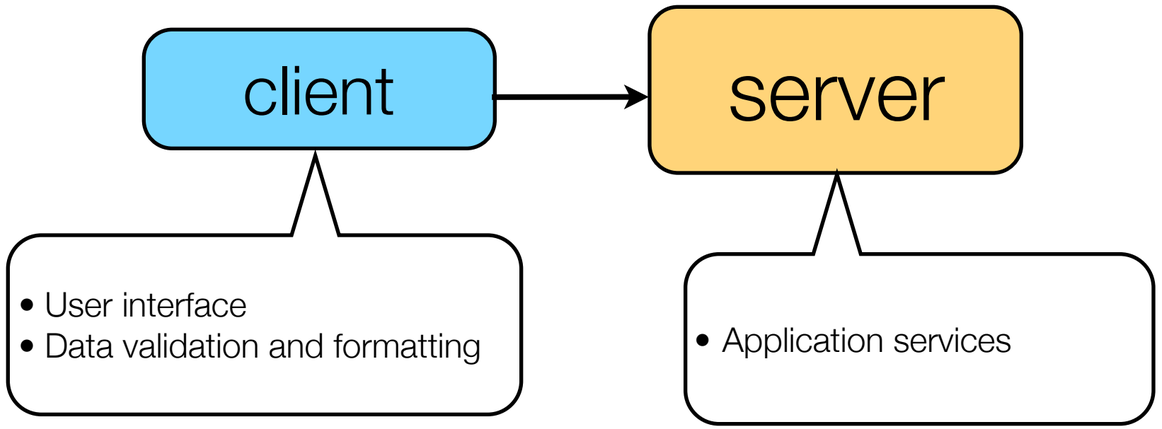
PEER-TO-PEER MODEL

- Each machine in the network has (roughly) equivalent capabilities
- Machines both request and provide services
- E.g. file sharing, bittorrent, bitcoin

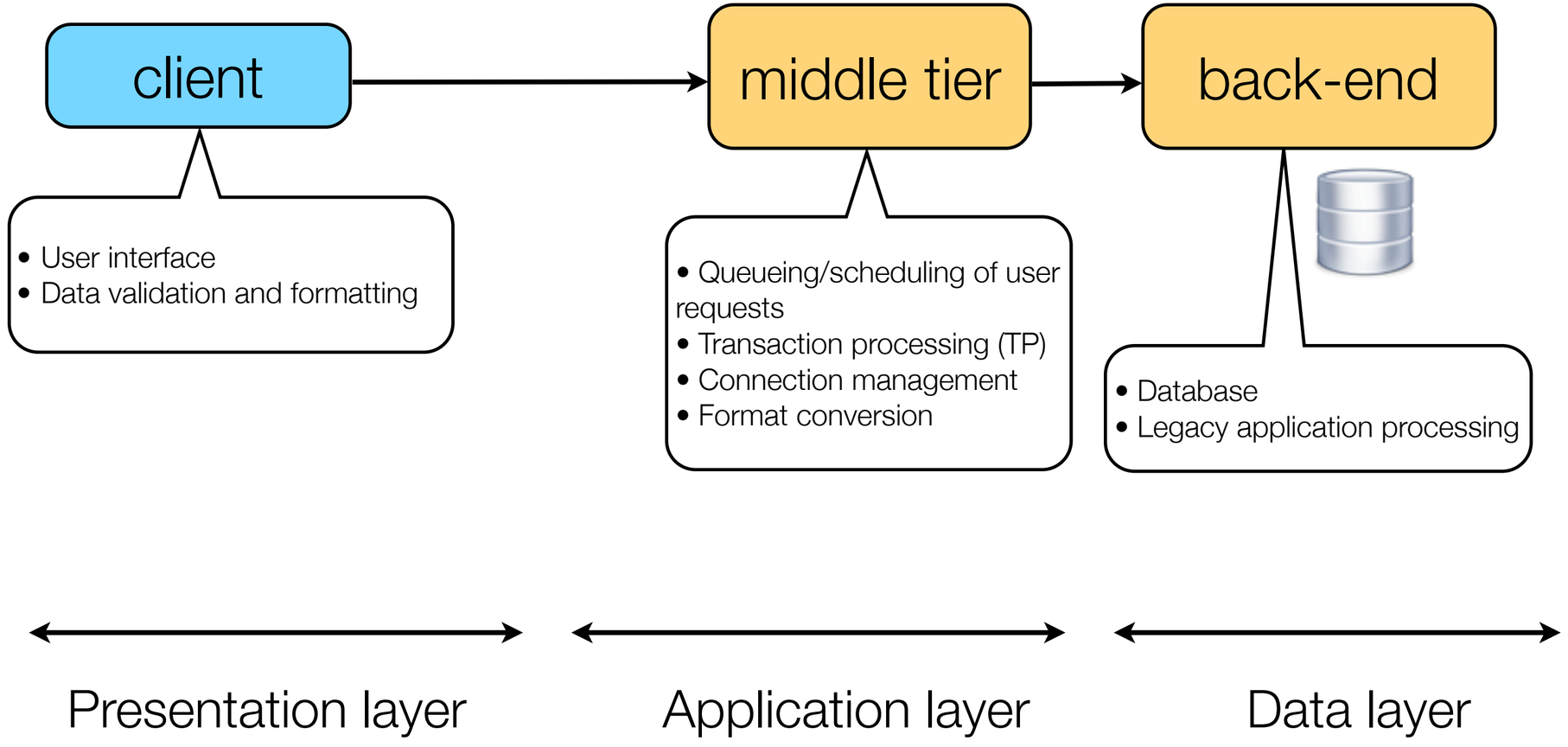


TWO-TIER ARCHITECTURE

- Common on the desktop from mid ‘80s to early ‘90s
- Common architecture today on the Web (client = webpage with Javascript)

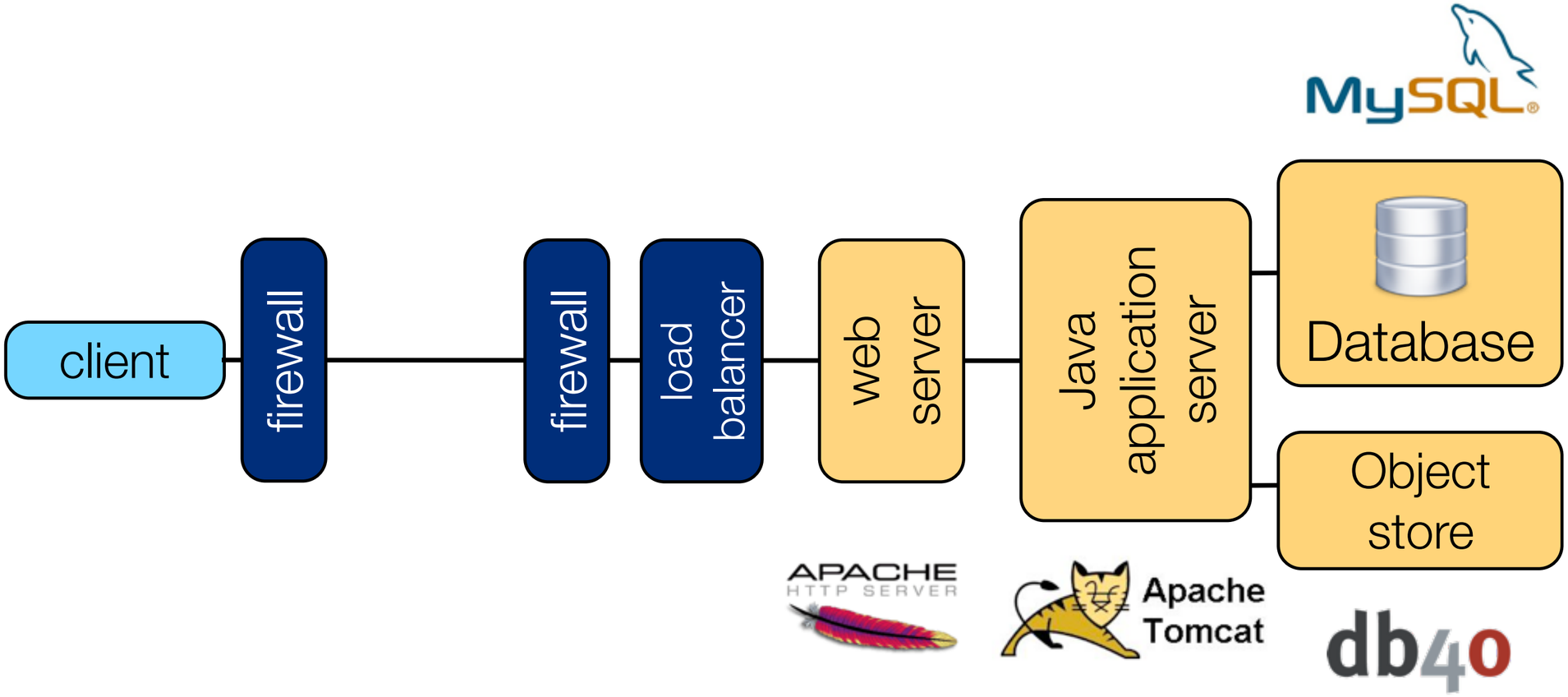


THREE-TIER ARCHITECTURE



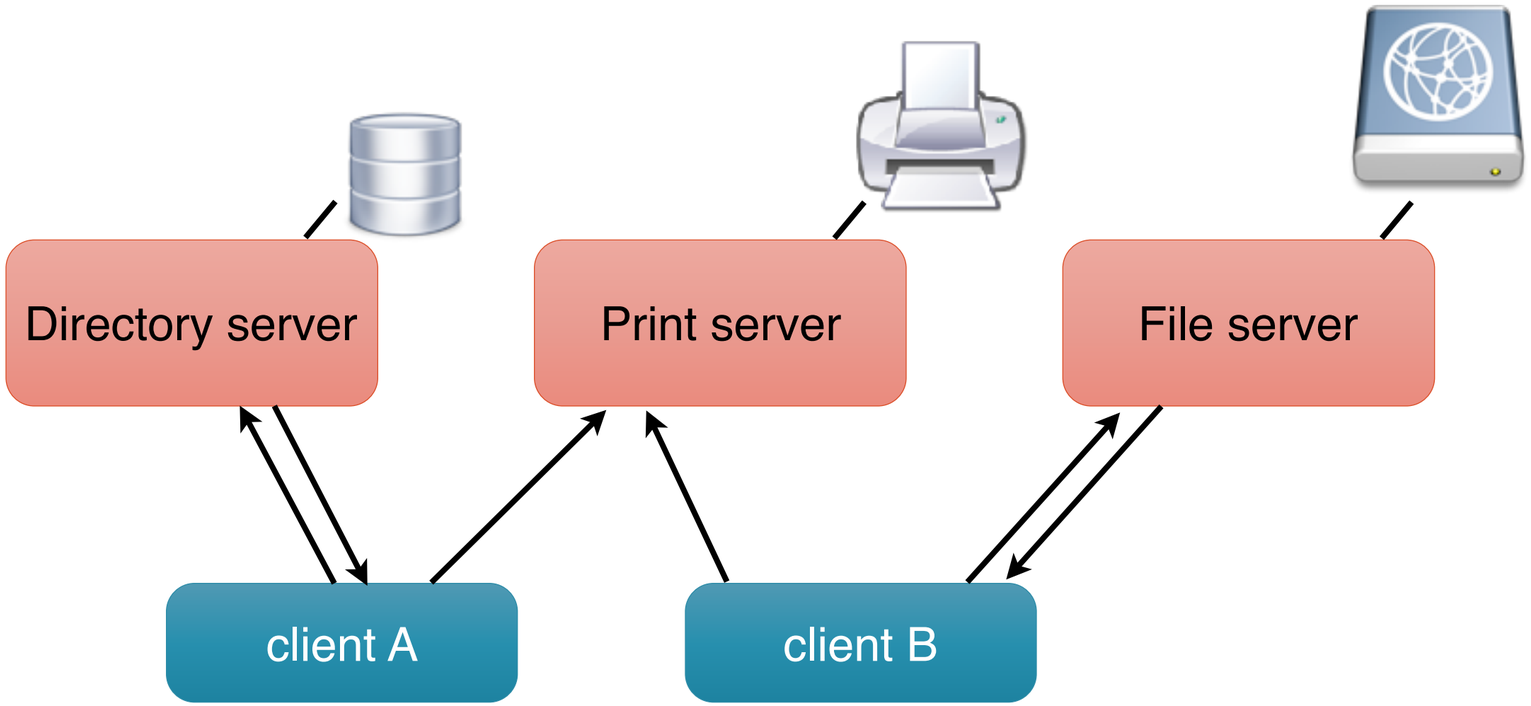
BEYOND THREE TIERS

– Most architectures are multi-tiered



PARALLELISM AND CONCURRENCY

DISTRIBUTION IMPLIES PARALLELISM



CONCURRENCY

Correctly and efficiently manage a shared resource

PARALLELISM

Use extra resources to solve a problem faster.

SHARED MEMORY MODEL

The model we will assume is shared memory with explicit threads

Old story: A running program has

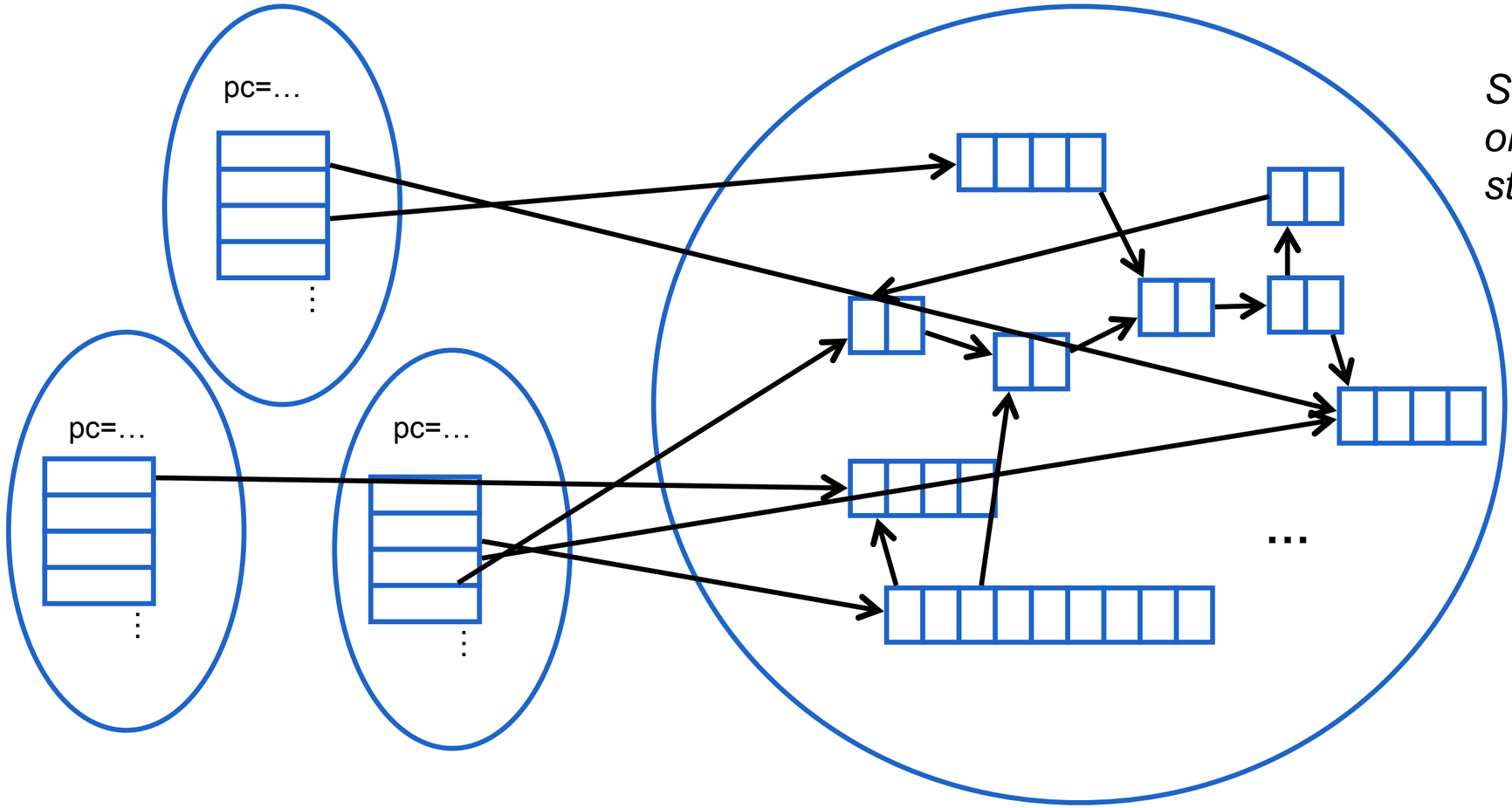
- One *call stack* (with each *stack frame* holding local variables)
- One *program counter* (current statement executing)
- Static fields
- Objects (created by **`new`**) in the *heap* (nothing to do with heap data structure)

New story:

- A set of *threads*, each with its own call stack & program counter
  - No access to another thread's local variables
- Threads can (implicitly) share static fields / objects
  - To *communicate*, write somewhere another thread reads

SHARED MEMORY MODEL

Unshared: locals and control



Shared: objects and static fields

JAVA THREADING BASICS

To get a new thread running:

1. Define a subclass `C` of **`java.lang.Thread`**, overriding **`run`**
2. Create an object of class `C`
3. Call that object's **`start`** method
   - Not **`run`**, which would just be a normal method call
   - **`start`** sets off a new thread, using **`run`** as its "main"

THREADS HELLO WORLD

```
public class ThreadExample  extends Thread {

    @Override
    public void run() {
        super.run();
        System.out.println("Hello from the spawned thread my name is:  " +
                           Thread.currentThread() );
    }

    public static void main(String... args) {
        ThreadExample te = new ThreadExample();
        System.out.println("Hello from the main thread my name is: " +
                           Thread.currentThread());
        te.start();
        System.out.println("Hello again from the main thread, my name is still : "
                           + Thread.currentThread() );
    }

}
```

1

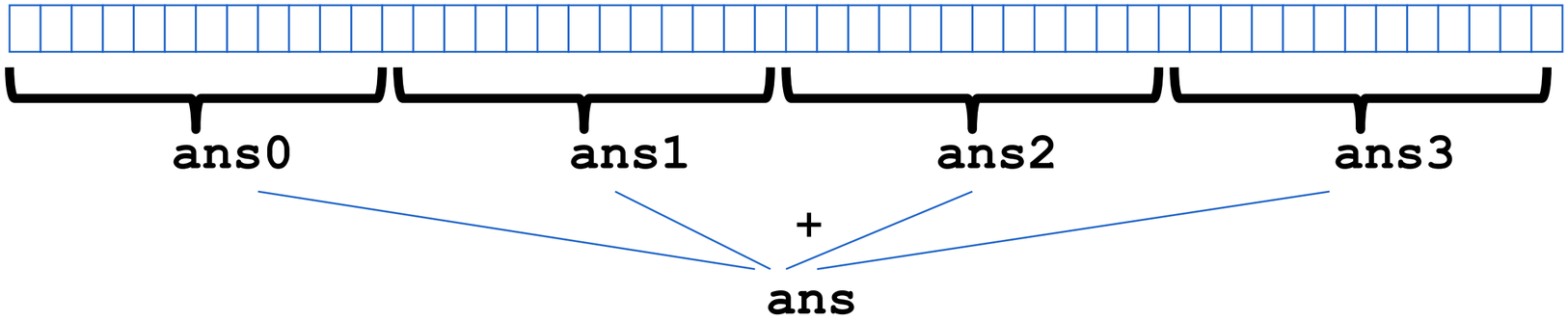
2

3

```
Hello from the main thread my name is: Thread[main,5,main]
Hello again from the main thread, my name is still : Thread[main,5,main]
Hello from the spawned thread my name is:  Thread[Thread-0,5,main]
```

SUM OF ELEMENTS

- Example: Sum elements of a large array
- Idea Have 4 threads simultaneously sum 1/4 of the array
  - *Warning: there exist better ways to calculate the sum in parallel*



  - Create 4 *thread objects*, each given a portion of the work
  - Call **`start()`** on each thread object to actually *run* it in parallel
  - *Wait* for threads to finish using **`join()`**
  - Add together their 4 answers for the *final result*

IMPLEMENTATION

```
class SumThread extends java.lang.Thread {
  int lo, hi;
  int[] arr;
  int ans = 0;

  SumThread(int[] a, int l, int h) {
      this.lo = l;
      this.hi = h;
      this.arr = a;
  }

  public void run() {
      ans = 0;
      for (int i = lo; i < hi; i++) {
        ans += arr[i];
    }
  }
}
```

```
static int sum(int[] arr) throws InterruptedException  {
      int len = arr.length;
      int ans = 0;
      SumThread[] ts = new SumThread[4];
      for(int i=0; i < 4; i++){//do parallel computations
          ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
          ts[i].start();
      }
      for(int i=0; i < 4; i++) { // combine results
          ts[i].join(); // wait for helper to finish!
          ans += ts[i].ans;
      }
      return ans;
  }
```

COMMON PITFALLS

```
static int sum(int[] arr) throws InterruptedException  {
        int len = arr.length;
        int ans = 0;
        SumThread[] ts = new SumThread[4];
        for(int i=0; i < 4; i++){//do parallel computations
            ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
            ts[i].start();
        }
        for(int i=0; i < 4; i++) { // combine results
            ts[i].join(); // wait for helper to finish!
            ans += ts[i].ans;
        }
        return ans;
    }
```

SHARED MEMORY?

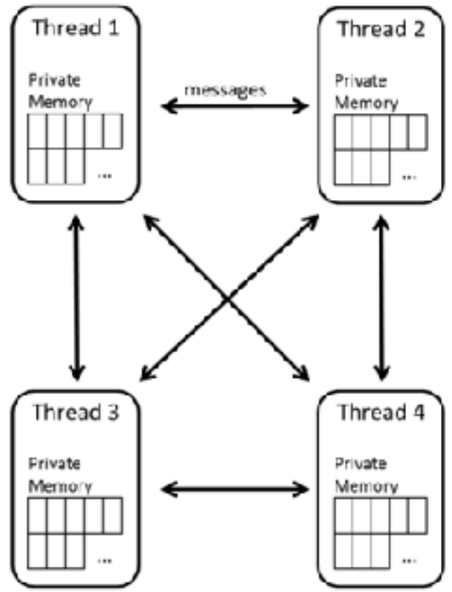
- Fork-join programs (thankfully) does not require much focus on sharing memory among threads

- But in languages like Java, there is memory being shared.
- In our example:
  - **`lo`**, **`hi`**, **`arr`** fields written by “main” thread, read by helper thread
  - **`ans`** field written by helper thread, read by “main” thread

- When using shared memory, you must avoid race conditions
  - **`join`** is one way to synchronize threads

MESSAGE PASSING

- Natural programming model for distributed memory
- Each thread has its own collection of objects, i.e. threads do not share objects.
- Communication is via explicit notion of messages, which sends a copy of some data to its recipient.

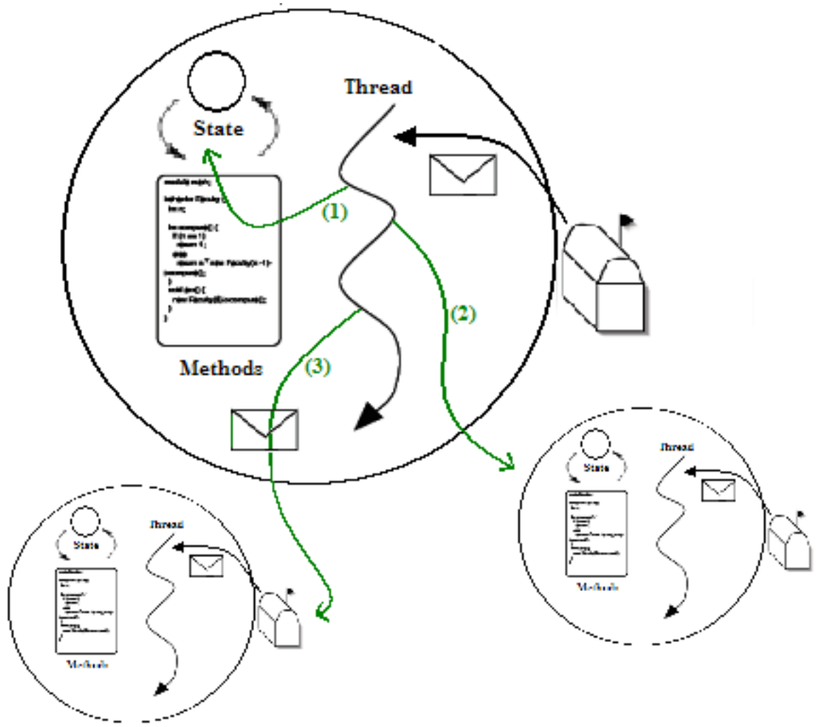


THE ACTOR MODEL

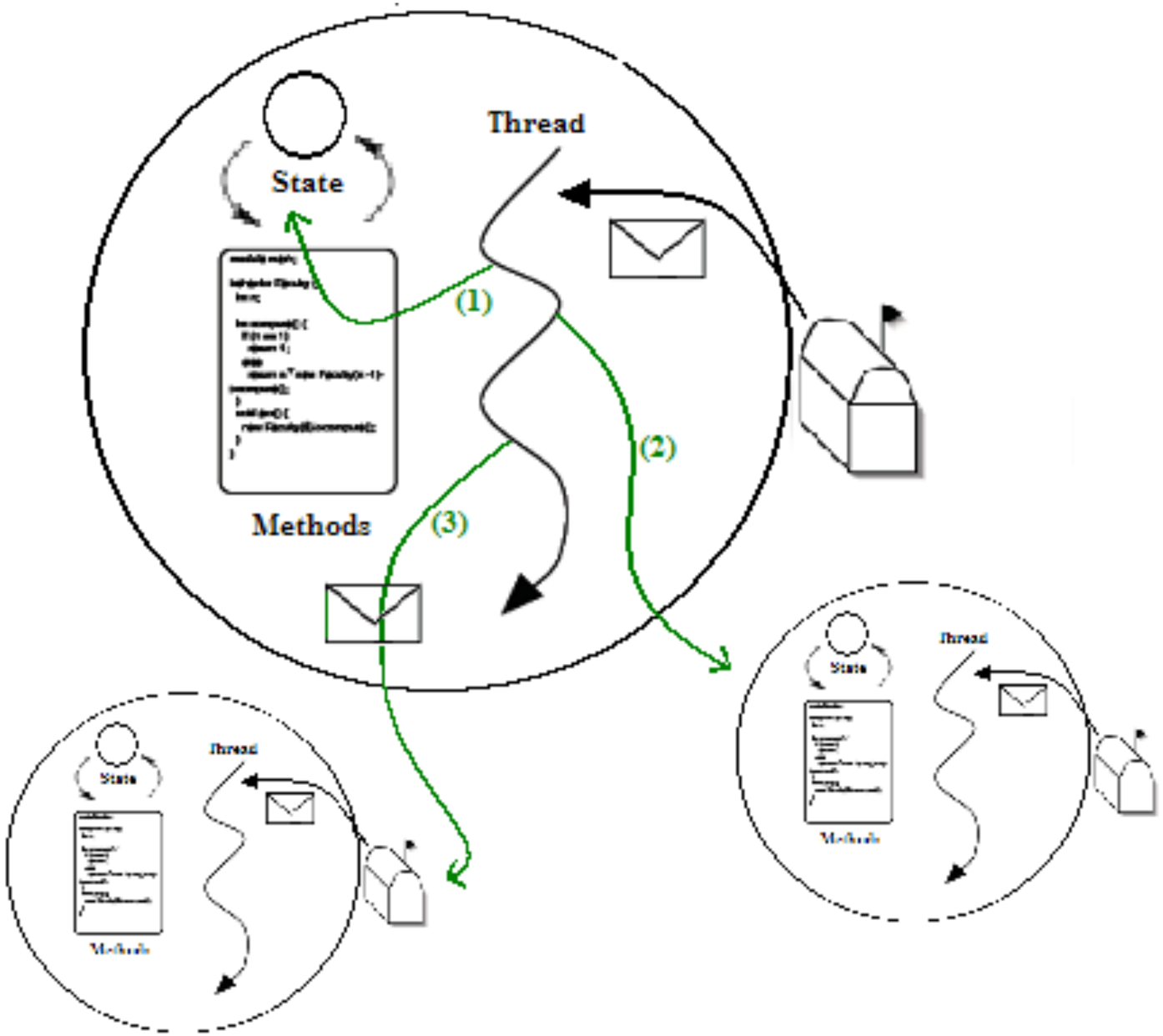
- First proposed by Carl Hewitt (1973)
- Gull Agha (1985) proposed the actor model as a general-purpose concurrent object-oriented programming model
- By now, many actor programming languages, some of them quite successful: Acttalk, Act1, ABCL, Erlang, Scala, AmbientTalk

WHAT ARE ACTORS

- Actors are independent concurrent objects, i.e. an object with its own thread of execution, that communicate by asynchronous message passing.
- Each actor has a mail address and a behaviour (expressed by fields and methods).
- When an actor receives a message it can:
  1. Modify its own state, possibly by changing its own behaviour
  2. Send messages to other actors
  3. Create new actors



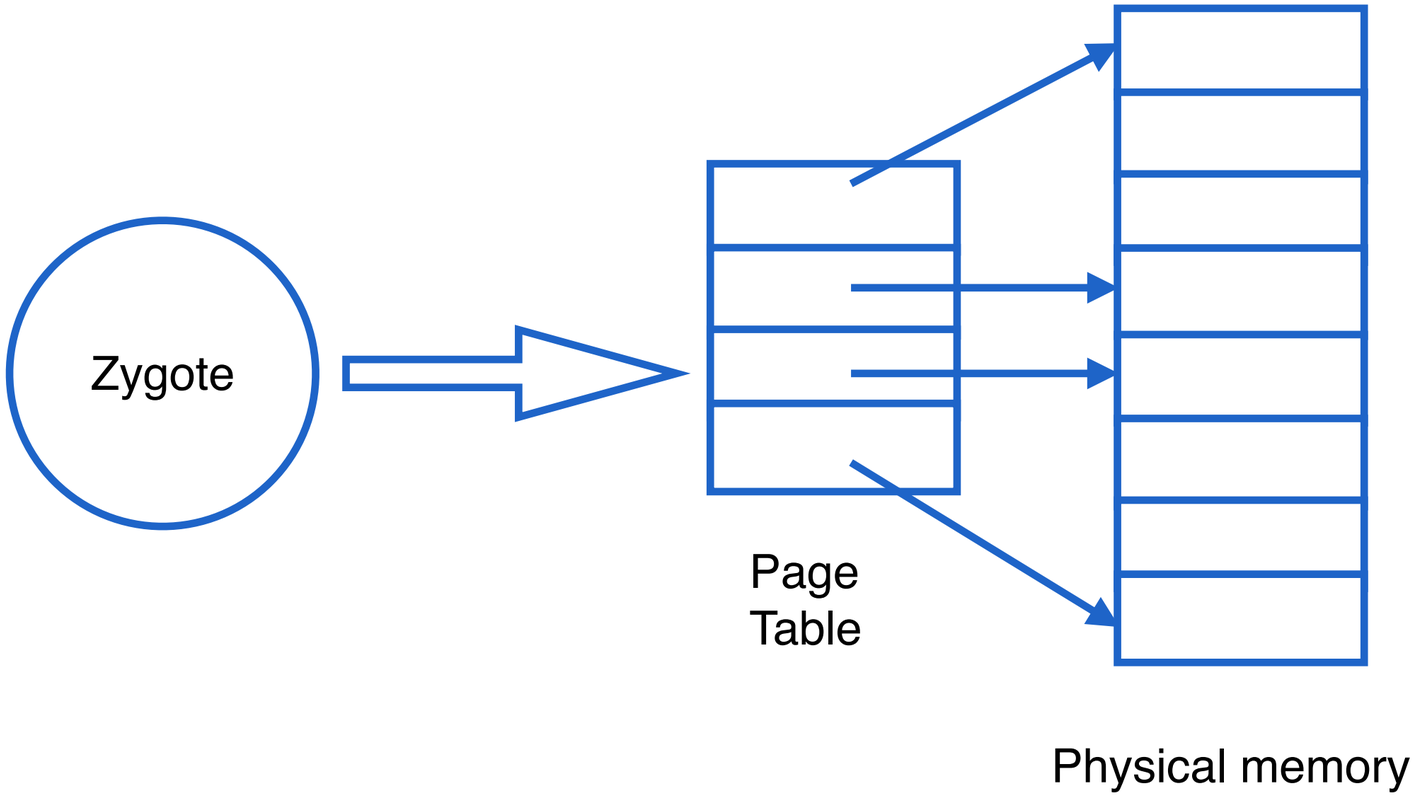
WHAT ARE ACTORS



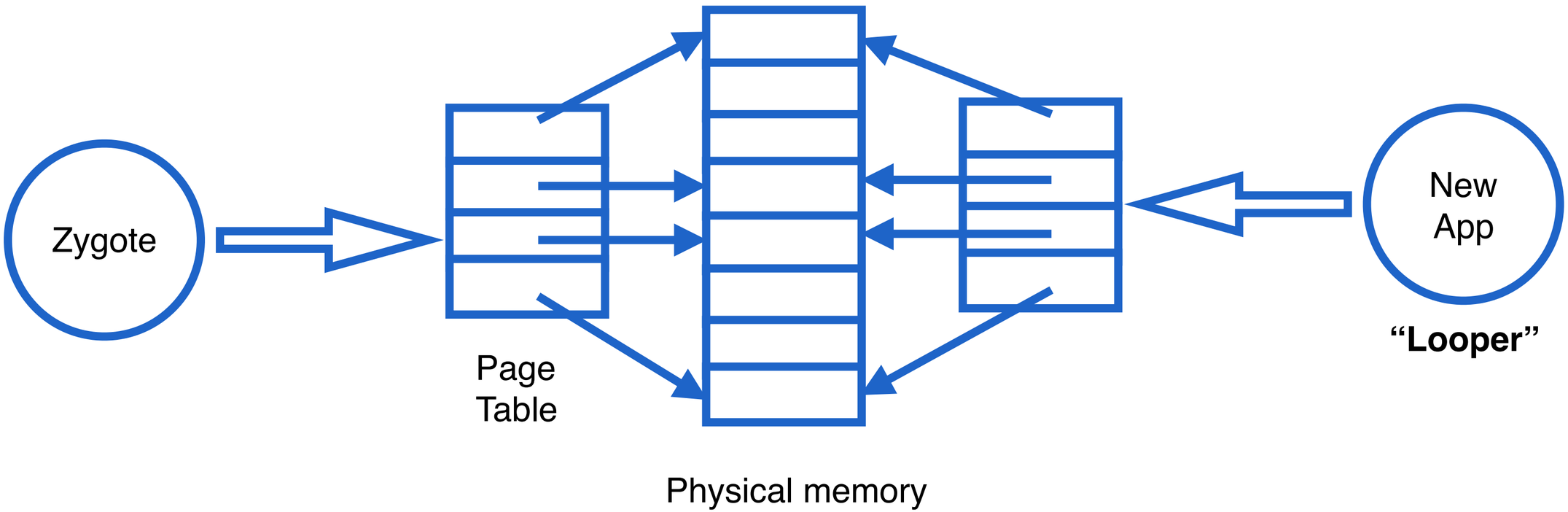
THE ACTOR MODEL

- Strengths:
  - Since actors do not share state => **no race conditions.**
  - Actor never suspends its execution to wait for another actor to finish a computation, rather they communicate via asynchronous messages => **no deadlocks.**
  - Actors support both shared and distributed memory architectures, suitable for distributed programming:
    - Geographical scale
    - Fault-tolerant
  - Weaknesses:
    - Still suffer from lack of progress issues
    - No direct support for parallelism

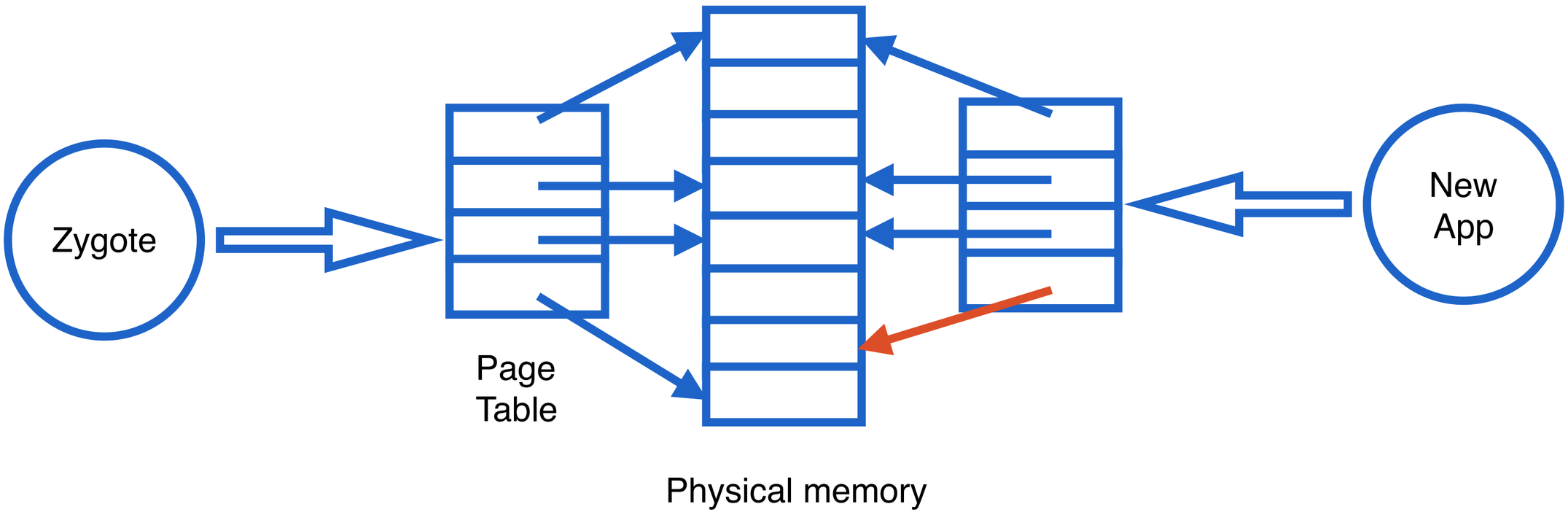
ANDROID SYSTEM



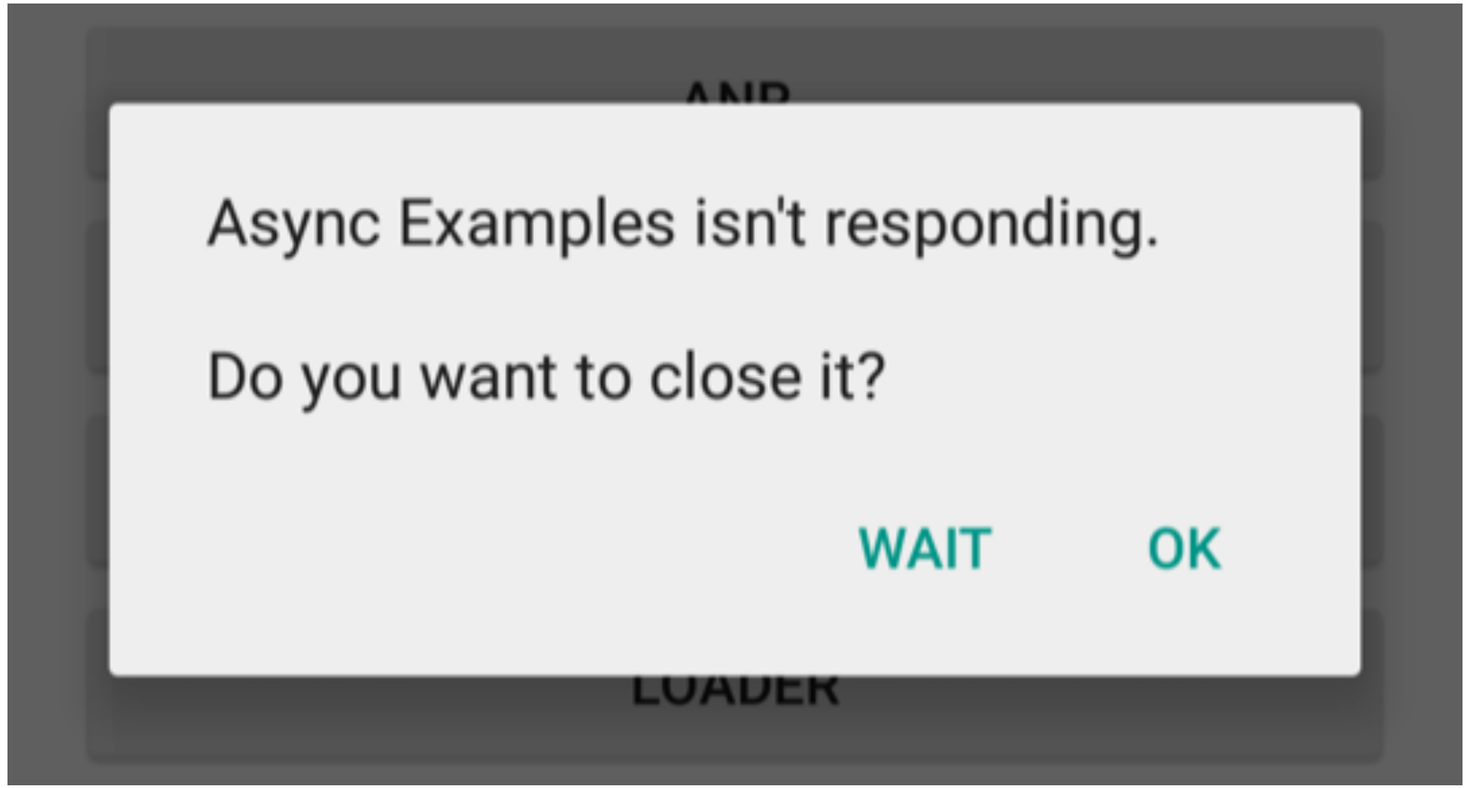
ANDROID SYSTEM



ANDROID COPY ON WRITE

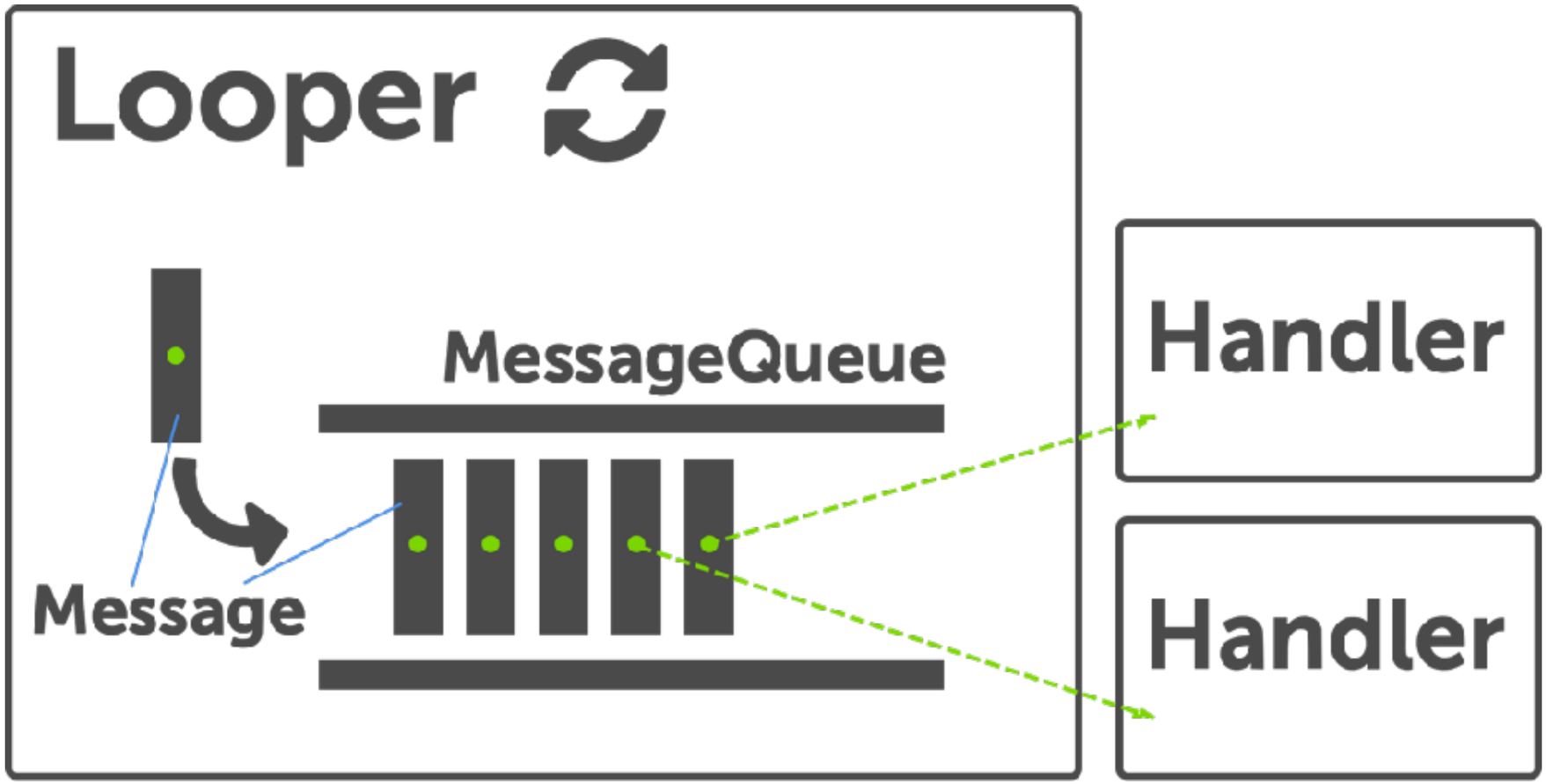


MAIN THREAD / UITHREAD



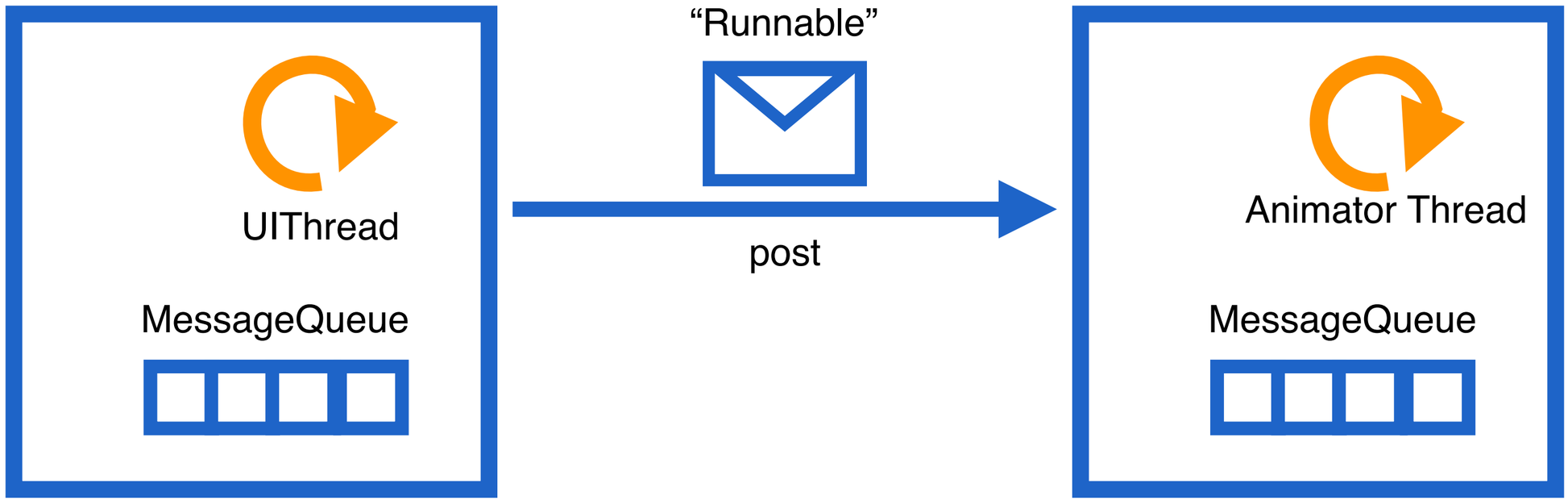
In any situation in which your app performs a potentially lengthy operation, you should not perform the work on the UI thread, but instead create a worker thread and do most of the work there. This keeps the UI thread (which drives the user interface event loop) running and prevents the system from concluding that your code has frozen.

LOOPER MODEL



http://blog.coldflake.com/posts/Android-style-Message-Passing/

PROGRAMMING WITH MULTIPLE LOOPERS



HELLO WORLD LOOPER

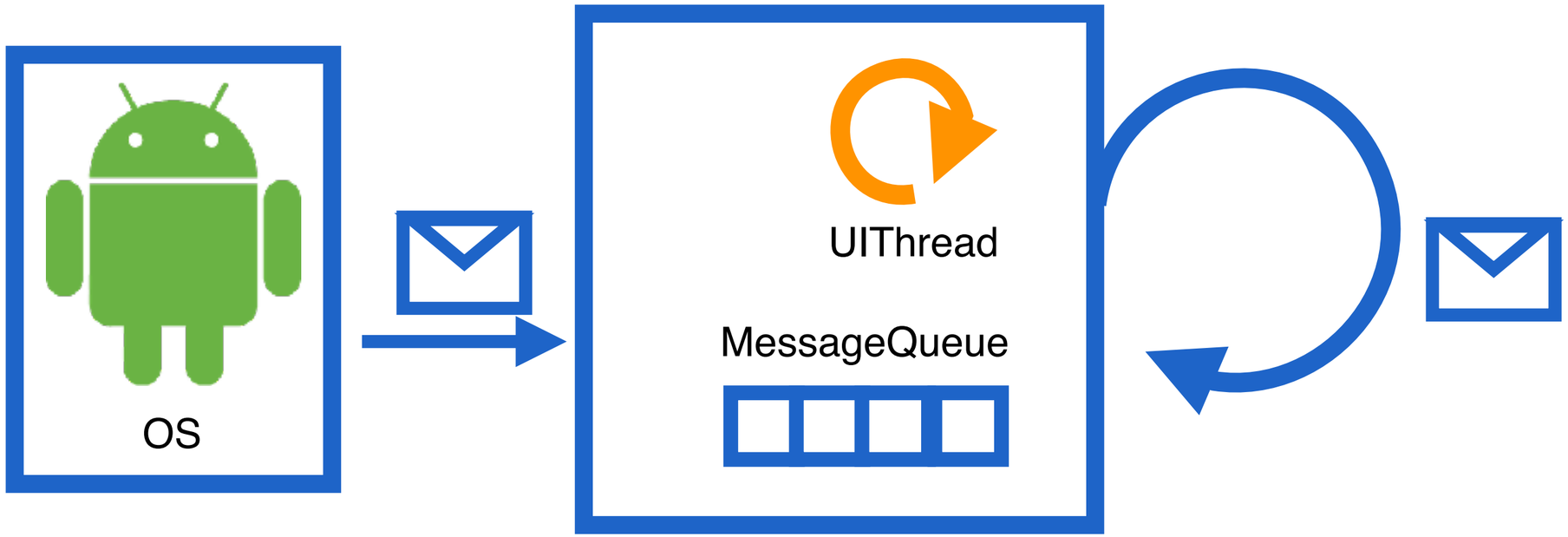
```
public class Animator extends Thread {
    public Handler mHandler;

    public void run() {
        Log.d("LOOPER","Started in a new Thread: " + Thread.currentThread() );
        Looper.prepare();
        mHandler = new Handler();
        Looper.loop();
    }

    public void printMessage() {
        Log.d("LOOPER","I am executing in thread: " + Thread.currentThread() );
    }

}
```

HELLO WORLD

```
public class Animation extends AppCompatActivity {
    Animator animator;

     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_animation);

          Log.d("MainLoper","Hello from UIThread: " + Thread.currentThread() );
          //Create a new Animator object
          animator = new Animator();
          //Start the thread
          animator.start();
          //wait until the mHandler is initialised !!
          // …
         //Post a task to the new Looper
          animator.mHandler.post( () ->  animator.printMessage()  );
     }

}
```

```
1163-1163/topl.ugent.edu.looperexample D/MainLoper: Hello from UIThread: Thread[main,5,main]
1163-1181/topl.ugent.edu.looperexample D/LOOPER: Started in a new Thread: Thread[Thread-2,5,main]
1163-1181/topl.ugent.edu.looperexample D/LOOPER: I am executing in thread: Thread[Thread-2,5,main]
```

CONCURRENCY



OPTIONALS / MAYBE

java.util.Optional<T>

of(T value)

map(**Function**<? super **T**,? extends U> mapper)

Functor

If a value is present, apply the provided mapping function to it, and if the result is non-null, return an Optional describing the result.

flatMap(**Function**<? super **T**,**Optional**<U>> mapper)

Monad

If a value is present, apply the provided Optional-bearing mapping function to it, return that result, otherwise return an empty Optional.

CONCURRENCY WITH LOOPERS

```
public class ConcurrentAnimation extends AppCompatActivity {
    private Handler  self;
    private Optional<Runnable> message;
    private TextView label;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_concurrent_animation);
        label = (TextView) findViewById(R.id.Label);
        self = new Handler();

        Animator an = new Animator(this,"World");
        message = Optional.of(an);
        self.postDelayed(an,3000);

    }

    @Override
    protected void onPause() {
        super.onPause();
           message.flatMap(
              (m) ->  { self.removeCallbacks(m);
                        return Optional.empty();
                      }
           );

    }

    public void setText(String s) {
        this.label.setText(s);
    }
```

```
class Animator implements Runnable {
    private String   newValue;
    private ConcurrentAnimation activity;

    Animator(ConcurrentAnimation activity,
                            String newValue) {
        this.activity = activity;
        this.newValue = newValue;
    }

    @Override
    public void run() {
        activity.setText(newValue);
    }

}
```

REMOVE MESSAGES

```
protected void onPause() {
     super.onPause();
     message.flatMap( (m) ->  { self.removeCallbacks(m); return Optional.empty(); } );
 }
```

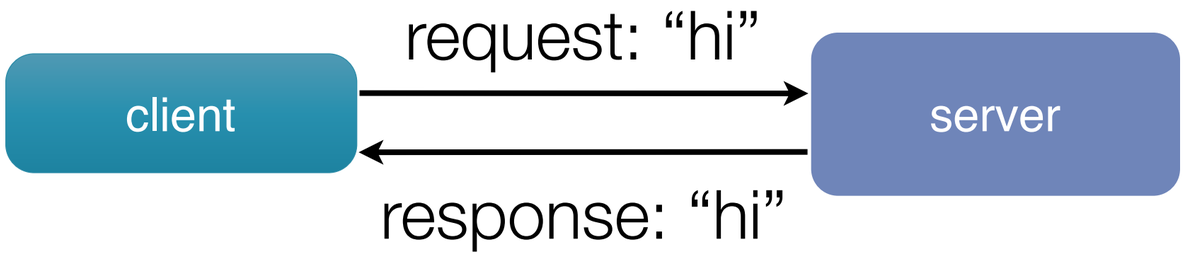
LOOPERS ARE NOT ACTORS

- Data can be shared between loopers
- UI elements are mostly “protected” i.e. there is a check at runtime to avoid UI elements to be accessed.
- Loopers and threads in the same system can still lead to deadlocks.

SOCKETS

EXAMPLE: ECHO SERVER

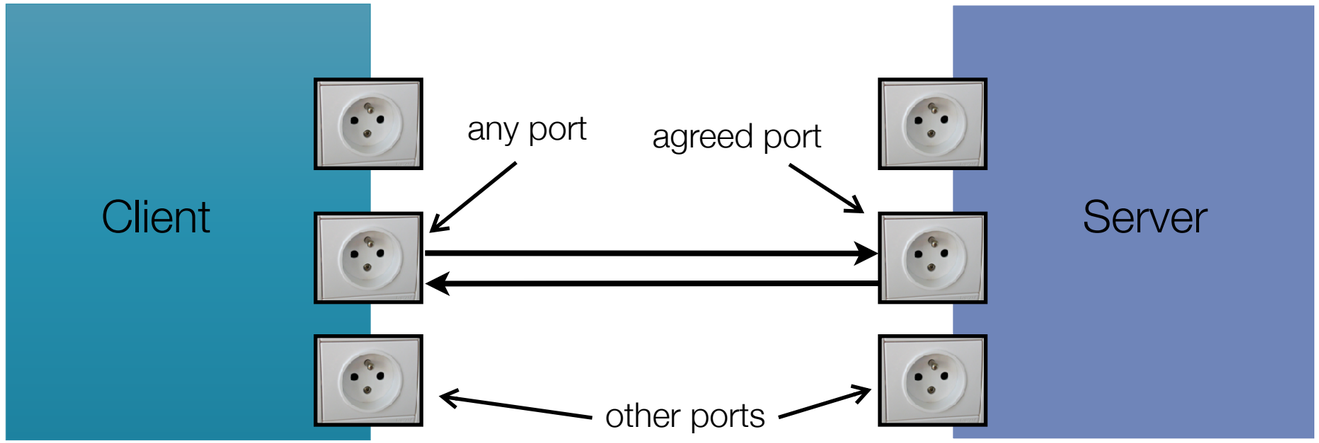
- The “hello world” of distributed computing
- Client sends a request carrying a string s
- Server responds to only a single type of request:
  expects a string s and sends back the received string



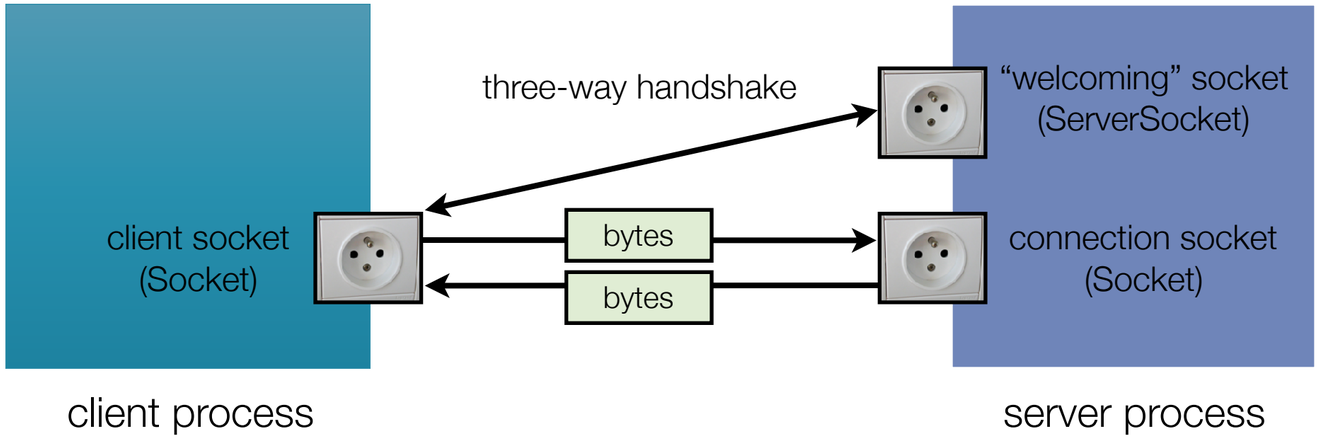
EXAMPLE: ECHO SERVER

- Example in Java using TCP/IP sockets
  - Start the server on a particular *port*
  - Client connects to server ip:port

Socket



TCP SOCKETS: ESTABLISHING A CONNECTION



ECHO SERVER

```
ServerSocket s = new ServerSocket(SERVER_PORT);
```

“welcoming” socket (ServerSocket)

```
Socket clientConnection  = s.accept();
```

connection socket (Socket)

```
OutputStream        o     = clientConnection.getOutputStream();
InputStream         i     = clientConnection.getInputStream();
```

STREAMS

java.io.Input/OutputStream

OutputStream

Bytes

public void write(byte[] b)
throws IOException

public void flush()
throws IOException

public void close()
throws IOException

InputStream

Bytes

public abstract int read()
throws IOException

public int available()
throws IOException

public void close()
throws IOException

STREAMS

java.io.Input/OutputStream

BufferedWriter

Strings

public void write(String str)
throws IOException

public void close()
throws IOException

BufferedReader

Strings

public String readLine()
throws IOException

public void close()
throws IOException

ECHO SERVER

```
ServerSocket s = new ServerSocket(SERVER_PORT);
```

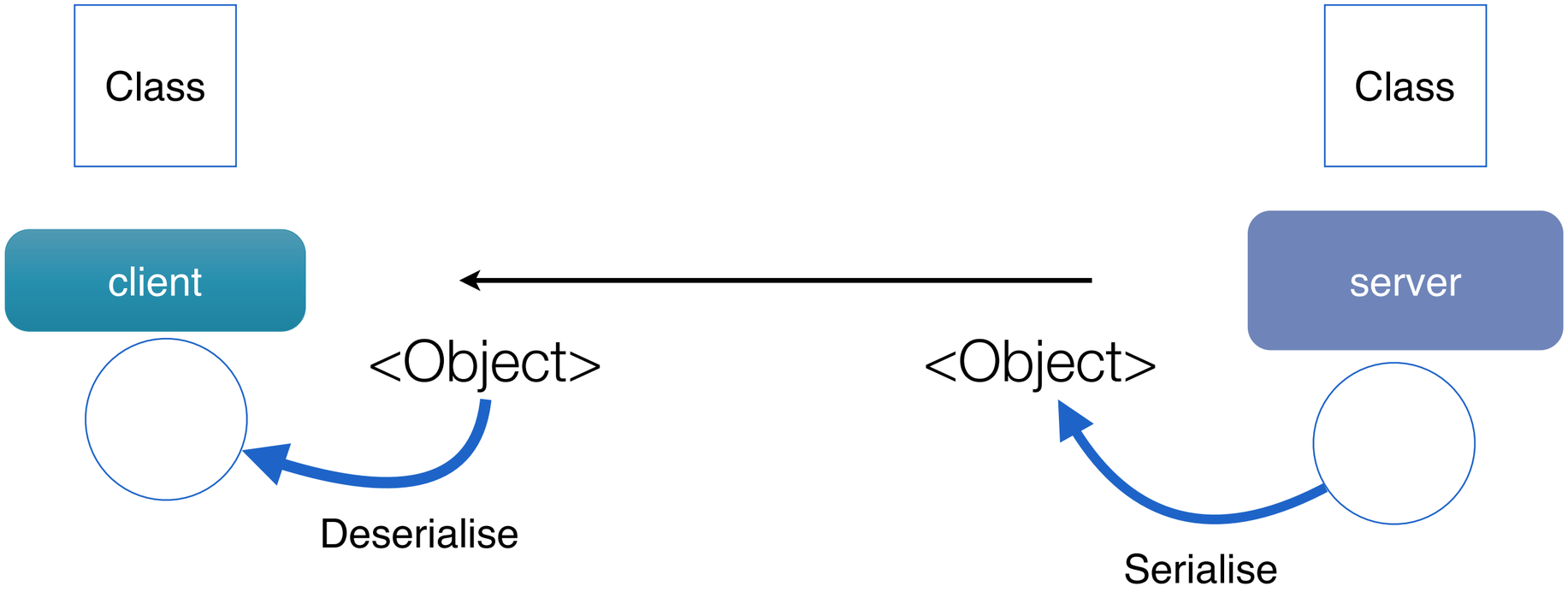
"welcoming" socket (ServerSocket)

```
Socket clientConnection  = s.accept();
```

connection socket (Socket)

```
OutputStream        o     = clientConnection.getOutputStream();
InputStream         i     = clientConnection.getInputStream();

BufferedWriter      bw    = new BufferedWriter(new OutputStreamWriter(o));
BufferedReader      br    = new BufferedReader(new InputStreamReader(i));

String              input = br.readLine();

bw.write(input);
bw.close();
br.close();
clientConnection.close();
```

SENDING OBJECTS



SERIALISATION

```
import java.io.Serializable;

public class Message implements Serializable {

    private static final long serialVersionUID = 729220062193520625OL;
    private String message;

    public Message(String msg) {
        this.message = msg;
    }

    public void print() {
        System.out.println(this.message);
    }

}
```

SERVER (OBJECT)

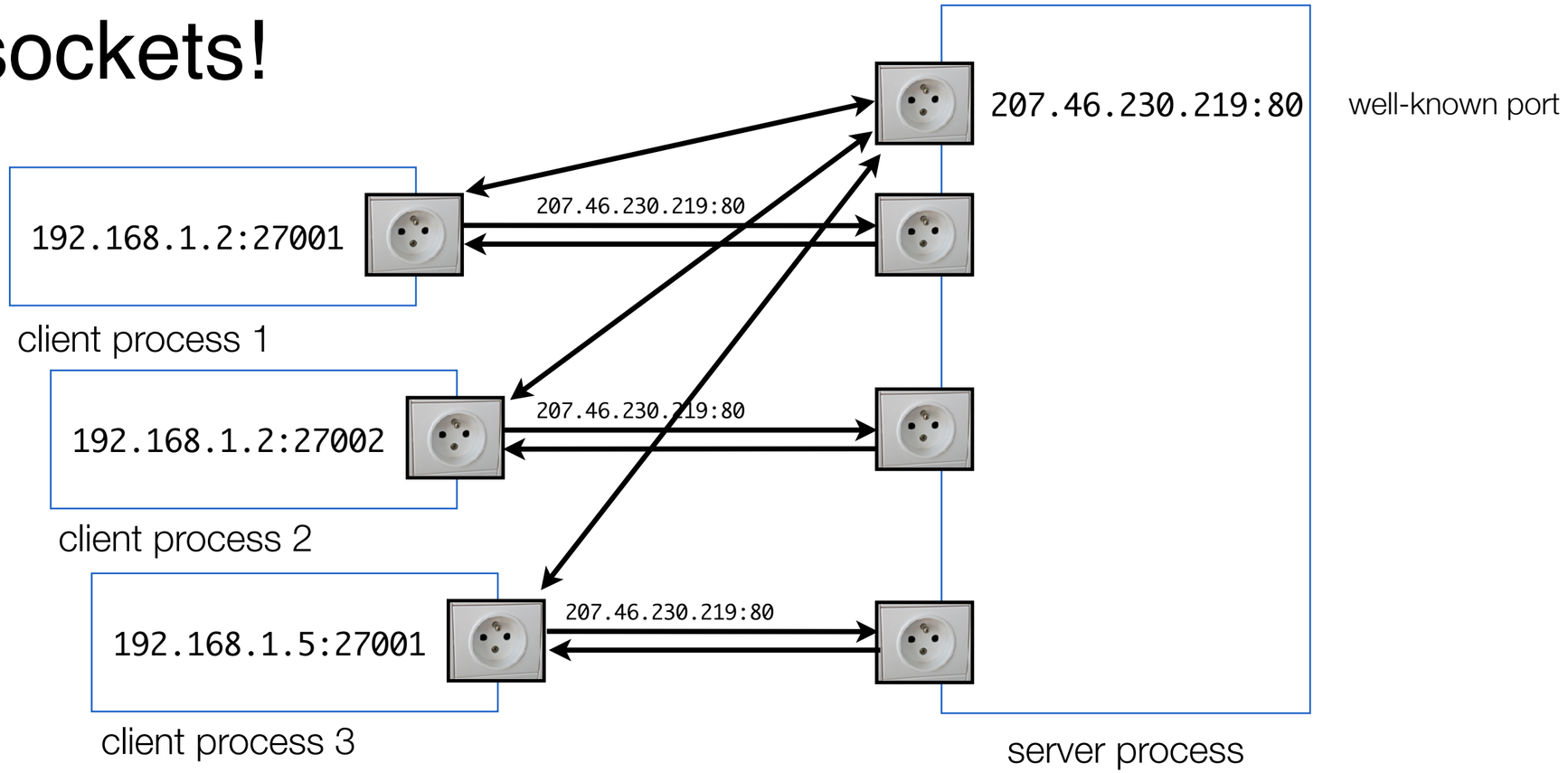
```
Message m                = new Message("Hello World!");
ServerSocket s           = new ServerSocket(SERVER_PORT);
Socket clientConnection = s.accept();
OutputStream        o    = clientConnection.getOutputStream();
ObjectOutputStream out   = new ObjectOutputStream(o);

out.writeObject(m);
clientConnection.close();
```

CLIENT (OBJECT)

```
Socket                s = new Socket(HOST,PORT);
InputStream          is = s.getInputStream();
ObjectInputStream isr = new ObjectInputStream(is);

Message m = (Message) isr.readObject();
m.print();
```

TCP SOCKETS: MULTIPLE CLIENT CONNECTIONS

- accept() returns a new socket for each connecting client
- Destination address and port are the same for all of these sockets!



ACCEPTING MULTIPLE CLIENTS

```
while(running) {
            Socket clientSocket;
            try {
                clientSocket       = serverSocket.accept();
                ClientConnection c = new ClientConnection(clientSocket,this) ;
                clients.add(c);
                c.start();
            } catch (java.net.SocketException e) {
                System.out.println( "Server finished closing the connections");
            } catch (Exception e) {
                System.err.println( "Error while waiting for client");
                e.printStackTrace();
        }
}
```

This is just one approach !

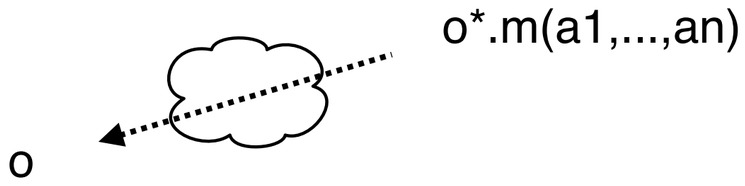
APPROACHES TO DEAL WITH DISTRIBUTION

“NAIVE” APPROACH: FULL TRANSPARENCY

Full transparency: use objects as the unit of distribution and **hide** the distribution in the message passing operator. Messages that happen to cross machine boundaries are automatically transformed into remote method invocations.

Hide complexity from the programmer

Earliest research



Remote object reference

o*.m(a1,...,an)
=
1. rmi(“m”,o*, marshall(a1,...,an))
2. await(answer) while o.m(a1*,...,an*) runs
3. unmarshall(answer)

CRITIQUE ON FULL TRANSPARENCY

Jim Waldo

The hard problem in distributed computing is **not** the problem of getting things on and off the wire !

–A note on distributed computing (1994)

THE FUNDAMENTAL PROBLEMS

Jim Waldo

#1 LATENCY

#2 DIFFERENCE IN MEMORY ACCESS

#3 PARTIAL FAILURE

#4 CONCURRENCY

–A note on distributed computing (1994)

#1 LATENCY

Remote invocation is 5 to 6 orders of magnitude slower and this difference will only increase.

Some transparent designs will require large amount of communications and will therefore be inherently slow.

#2 DIFFERENCE IN MEMORY ACCESS

A pointer/variable in one address space has a different meaning in another address space:

- either: accessing = reading $\Longrightarrow$ duplicating
- or: accessing = remote $\Longrightarrow$ message sending

#3 PARTIAL FAILURE

• In local computing, failure is total. In DC, an independent component (node, link) can fail while the others continue; i.e. **failure is partial**.

• There is no common agent that is able to determine what component has failed and inform the other components of that failure.

• The failure of a network link (or latency) is indistinguishable from the failure of a node.

#4 CONCURRENCY

- To cooperate with each other, processes in a distributed application need not only to communicate, but also to **synchronize** their actions. Distributed objects by nature must handle concurrent method invocations.

- Automatic parallellisation efforts have failed.

- Parallel programming must explicitly be taken into account.

LEAVING FULL TRANSPARENCY

Jim Waldo

Rachid Guerraoui

The four aforementioned problems show that distribution cannot be added "after the facts". Distribution is not just an implementation detail.

Distribution
Transparency is a myth that is misleading and dangerous

–OO Distributed Programming is NOT Distributed OO Programming (1999)

LEAVING FULL TRANSPARENCY

Jim Waldo

"Distributed objects are different from local objects and keeping that difference visible will keep the programmer from forgetting the difference and making mistakes."

Invent New Abstractions

2 Approaches: Languages vs. Middleware

PROGRAMMING LANGUAGE APPROACH

aka “Integrative Approach”

- Think of new concepts
- Turn them into a language feature and add it to a language
- Bury the dirty details in the interpreter/VM/compiler
- (Redesign that language for better integration)

MIDDLEWARE APPROACH

aka “Library Approach”

- Think of new concepts
- Implement them as a class-library in your favorite language (usually Java or C#).
- Use the abstraction/reuse-facilities of that language to make them reusable&generic

class Connection
{ ... }

inheritance, encapsulation, composition, templates, ...

EXAMPLE

Send a message “`ping`” to an object `o` on the wireless network whose name is “`MyPrinter`”

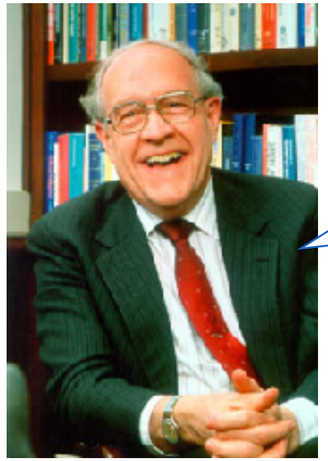
```
when: MyPrinter discovered: { |o|
    o <- ping();
 }
```

Language Approach

Middleware Approach

```
                    NameServer n = NameServer.instance();
                        Object o = n.lookup(“MyPrinter”);
Message m = new Message(o, “ping”, new Object[{ a });
                                                m.send();
```

CRITIQUE ON MIDDLEWARE



Frederic Brooks

Essential Complexity is tackled by adding Accidental Complexity

"What does a high-level language accomplish? It frees a program from much of its accidental complexity.
.... it eliminates a whole level of complexity that was never inherent in the program at all."

–No Silver Bullet — Essence and Accidents of Software Engineering (1986)

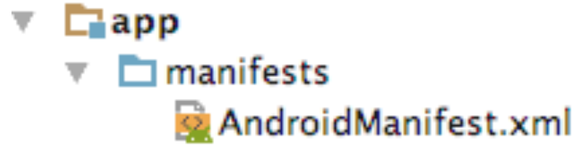
HTTP NETWORK OPERATIONS

STEPS TO RETRIEVE DATA OVER HTTP

- Set the permissions of your application
- Open a data stream
- Convert the data stream into meaningful data
- Close the stream

- Check the network availability
- Check user settings

PERMISSIONS

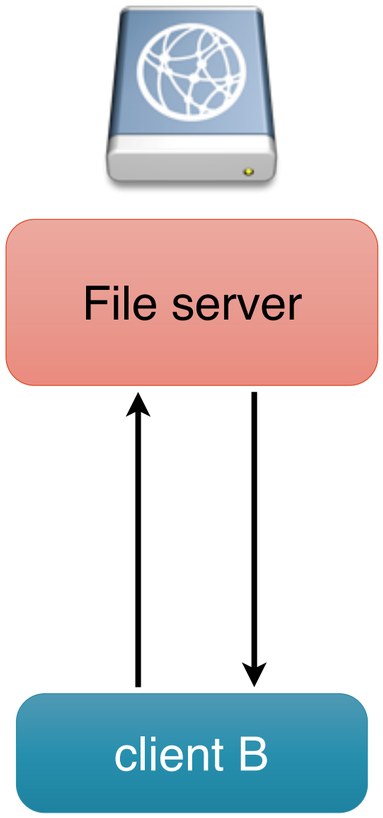


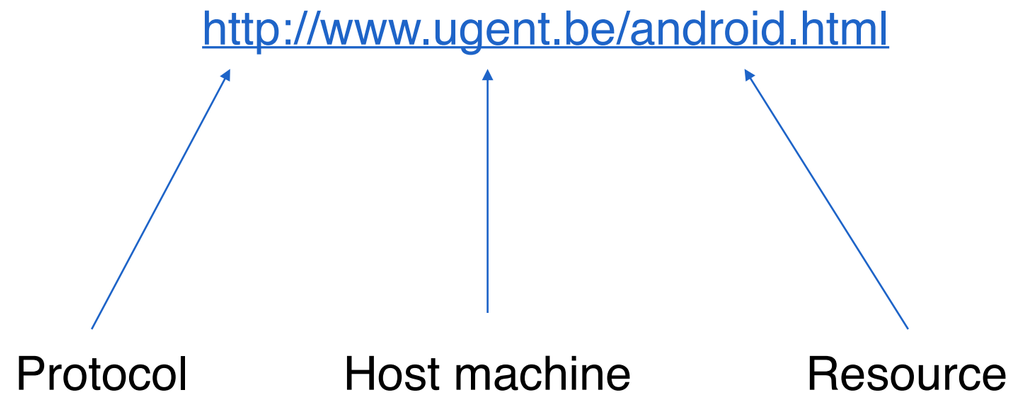
Access to the network operations

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Access to the network state

UNIFORM RESOURCE LOCATOR






```
Uri.Builder builder = new Uri.Builder();

builder.scheme("http")
       .authority("www.ugent.be")
       .appendPath("android.html")
       .build();
```

Use URI builder for complex urls !

HTTPURLCONNECTION

java.net.HttpURLConnection

1 Obtain a new `HttpURLConnection` by calling `URL.openConnection()` and casting the result to `HttpURLConnection`.

2 Prepare the request. The primary property of a request is its URI. Request headers may also include metadata such as credentials, preferred content types, and session cookies.

3 Optionally upload a request body. Instances must be configured with `setDoOutput(true)` if they include a request body. Transmit data by writing to the stream returned by `getOutputStream()`.

4 Read the response. Response headers typically include metadata such as the response body's content type and length, modified dates and session cookies. The response body may be read from the stream returned by `getInputStream()`. If the response has no body, that method returns an empty stream.

5 Disconnect. Once the response body has been read, the `HttpURLConnection` should be closed by calling `disconnect()`. Disconnecting releases the resources held by a connection so they may be closed or reused.

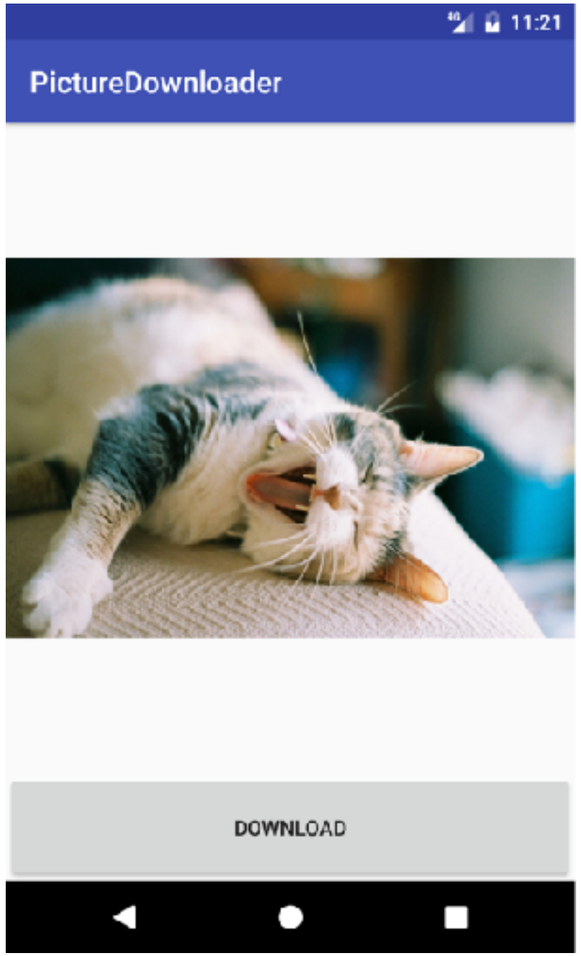
READ

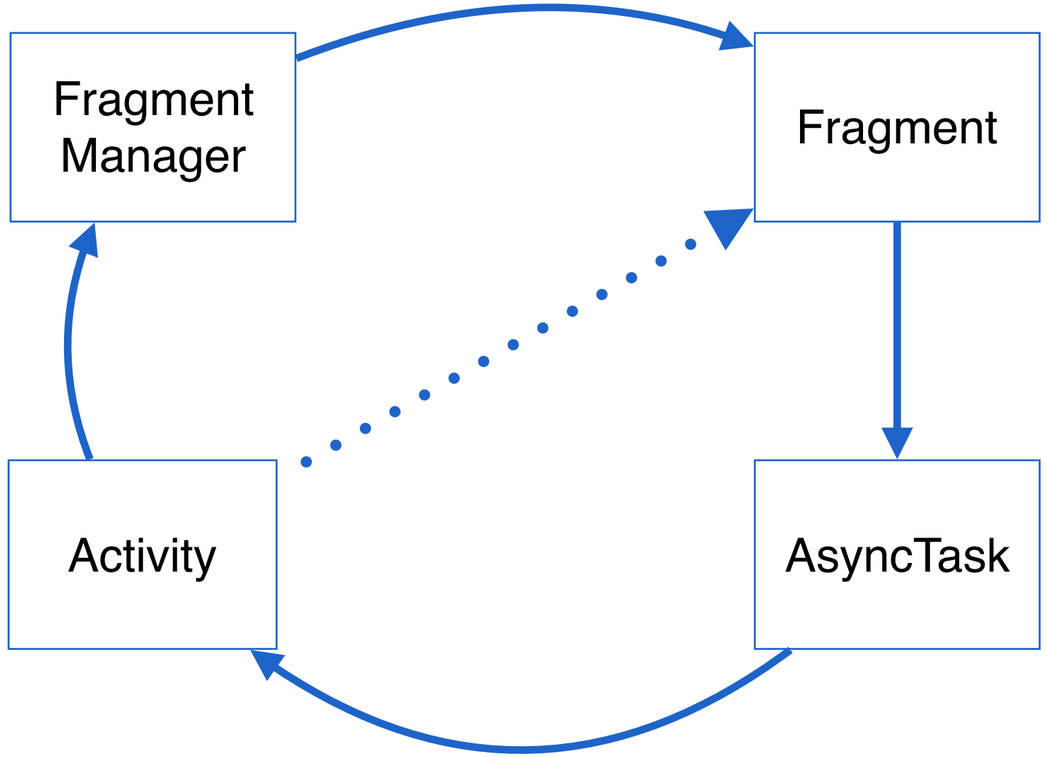
```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try {
  InputStream in = new BufferedInputStream(urlConnection.getInputStream());
  readStream(in);
} finally {
  urlConnection.disconnect();
}
```

WRITE

```
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
  try {
    urlConnection.setDoOutput(true);
    urlConnection.setChunkedStreamingMode(0);

    OutputStream out = new BufferedOutputStream(urlConnection.getOutputStream());
    writeStream(out);

    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
    readStream(in);
  } finally {
    urlConnection.disconnect();
  }
```

NETWORK OPERATIONS ARE NOT ALLOWED ON THE MAIN TRHEAD !





See asynchronous programming Chapter 4

EXAMPLE DOWNLOADING PICTURES

```
protected Bitmap doInBackground(String[] params) {
      URL url = new URL(Constants.CAT_URL);
      HttpURLConnection = (HttpURLConnection) url.openConnection();
      InputStream in = new BufferedInputStream(urlConnection.getInputStream());
      return readStream(in);
 }

  private Bitmap readStream(InputStream in) {
      return  BitmapFactory.decodeStream(in);
  }

  protected void onPostExecute(Bitmap result) {
         //show result
  }
```

THINGS MIGHT GO WRONG !

```
protected Bitmap doInBackground(String[] params) {
      URL url = null;
      try {
          url = new URL(Constants.CAT_URL);
      } catch (MalformedURLException e) {
          e.printStackTrace();
          return null;
      }

      HttpURLConnection urlConnection = null;

      try {
          urlConnection = (HttpURLConnection) url.openConnection();
      } catch (IOException e) {
          e.printStackTrace();
          return null;
      }

      try {
          InputStream in = new BufferedInputStream(urlConnection.getInputStream());
          return readStream(in);
      } catch (IOException e) {
          e.printStackTrace();
          return null;
      } finally {
          urlConnection.disconnect();
      }
  }

  private Bitmap readStream(InputStream in) {
      return  BitmapFactory.decodeStream(in);
  }

  protected void onPostExecute(Bitmap result) {
      if(result != null) {
           //show result
      } else {
           //show error
     }
     //cleanup fragment
  }
```

NETWORK STATUS

CONNECTIVITY MANAGER (SYSTEM SERVICE)

android.net.ConnectivityManager

1. Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
2. Send broadcast intents when network connectivity changes
3. Attempt to "fail over" to another network when connectivity to a network is lost
4. Provide an API that allows applications to query the coarse-grained or fine-grained state of the available networks
5. Provide an API that allows applications to request and select networks for their data traffic

GETTING INFORMATION ABOUT THE NETWORK STATE

android.net.ConnectivityManager

```
private static final String DEBUG_TAG = "NetworkStatusExample";
...
ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();
networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

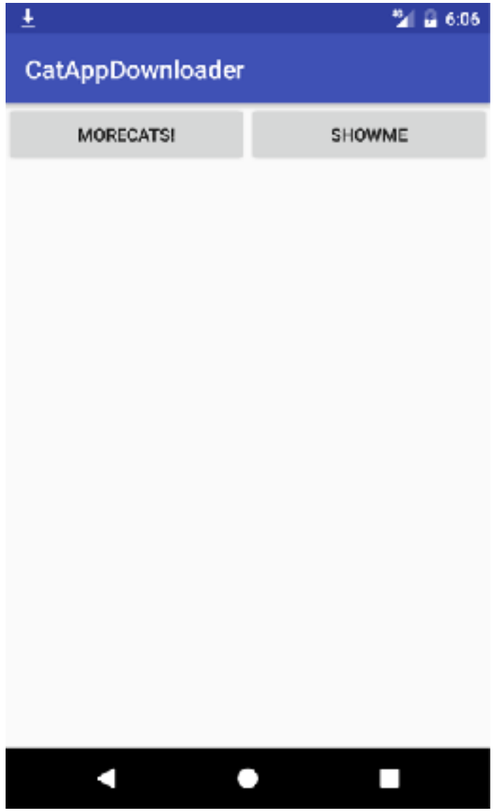
ONLINE ?

android.net.ConnectivityManager

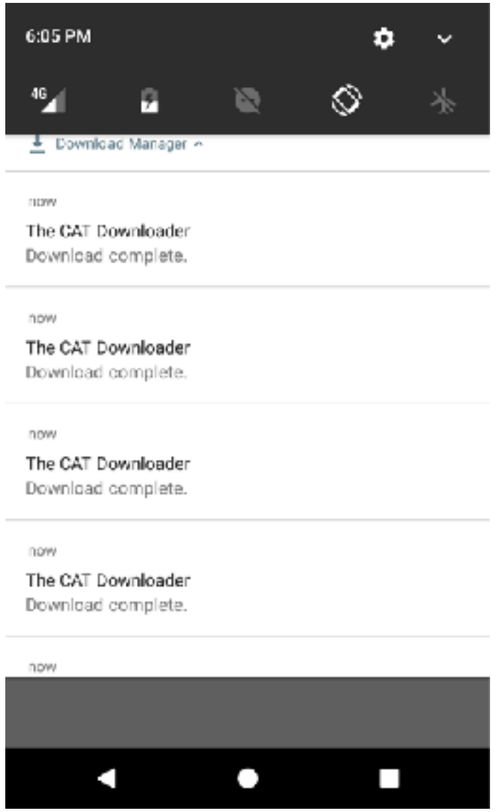
```
public boolean isOnline() {
    ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.isConnected());
}
```

DOWNLOAD MANAGER

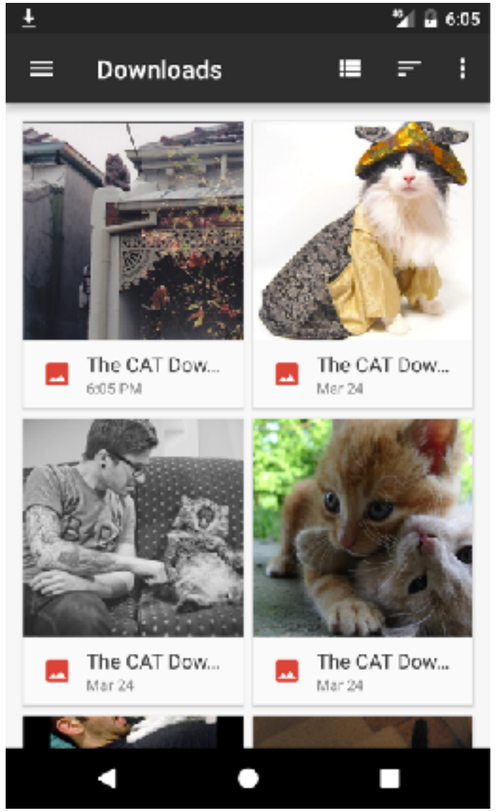
DOWNLOADING FILES



Start Screen



Download Status



Show the pictures

DOWNLOAD MANAGER

android.app.DownloadManager

- The download manager is a system service that handles long-running HTTP downloads. Clients may request that a URI be downloaded to a particular destination file.
- The download manager will conduct the download in the background, taking care of HTTP interactions and **retrying downloads after failures or across connectivity changes and system reboots**. Instances of this class should be obtained through getSystemService(String) by passing DOWNLOAD_SERVICE.
- Apps that request downloads through this API should register a broadcast receiver for ACTION_NOTIFICATION_CLICKED to appropriately handle when the user clicks on a running download in a notification or from the downloads UI.
- Note that the application must have the INTERNET permission to use this class.

DOWNLOADING A FILE

android.app.DownloadManager

1
```
downloadManager = (DownloadManager) getSystemService(this.DOWNLOAD_SERVICE);
```

2
```
DownloadManager.Request request = new DownloadManager.Request(uri);
```

3
```
request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED);
request.setTitle("The CAT Downloader");
request.setDescription("This is going to be awesome !”);
```

4
```
long reference = downloadManager.enqueue(request);
```

Show progress in the notification bar.

1.Get a reference to the download manager
2.Create a request
3.Set the request parameters
4.Enqueue the request into the downloadManager

RESTRICTING NETWORK UTILISATION

android.app.DownloadManager

```
request.setAllowedNetworkTypes(DownloadManager.Request.NETWORK_WIFI);
```

```
request.setAllowedNetworkTypes(DownloadManager.Request.NETWORK_MOBILE);
```

REACTION TO DOWNLOAD COMPLETION

android.app.DownloadManager

```
BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(context,"Got you a new cat !",Toast.LENGTH_LONG).show();
        }
    };


IntentFilter filter = new IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE);
registerReceiver(receiver,filter);
```

DETAILS ABOUT THE COMPLETED DOWNLOADS

android.app.DownloadManager

```
public void onReceive(Context context, Intent intent) {
        long reference = intent.getLongExtra(DownloadManager.EXTRA_DOWNLOAD_ID, -1);
        if (reference == myDownloadReference) {
            DownloadManager.Query myDownloadQuery = new DownloadManager.Query();
            myDownloadQuery.setFilterById(reference);
            Cursor myDownload = downloadManager.query(myDownloadQuery);
            if (myDownload.moveToFirst()) {
                int fileUriIdx  = myDownload.getColumnIndex(DownloadManager.COLUMN_LOCAL_URI);
                String fileUri  = myDownload.getString(fileUriIdx);
                // Do something with the file.
            }
        }
}
```

CLEANUP !

```
protected void onPause() {
        super.onPause();
        if (receiver != null) {
            unregisterReceiver(receiver);
        }
}
```

REACTION TO DOWNLOAD CLICKS

android.app.DownloadManager

```
private BroadcastReceiver onClickreceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String id = DownloadManager.EXTRA_NOTIFICATION_CLICK_DOWNLOAD_IDS;
            long[] references = intent.getLongArrayExtra(id);
            for (long reference : references) {
                //Do something with the download reference ...
            }
        }
};

IntentFilter filterClick = new IntentFilter(DownloadManager.ACTION_NOTIFICATION_CLICKED);
registerReceiver(onClickreceiver, filter);
```

SHOWING THE DOWNLOAD FOLDER

```
public void showMeTheCats(View v) {
        Intent i = new Intent();
        i.setAction(DownloadManager.ACTION_VIEW_DOWNLOADS);
        startActivity(i);
}
```

MIDDLEWARE VOLLEY

ADDING VOLLEY TO YOUR PROJECT

```
dependencies {
    ...
    compile 'com.android.volley:volley:1.0.0'
}
```

SENDING A SIMPLE REQUEST

```
// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Display the first 500 characters of the response string.
        mTextView.setText("Response is: "+ response.substring(0,500));
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        mTextView.setText("That didn't work!");
    }
});
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

NETWORK IMAGE VIEW

```
NetworkImageView avatar = (NetworkImageView)view.findViewById(R.id.twitter_avatar);
avatar.setImageUrl("http://someurl.com/image.png",mImageLoader);
```

```
mImageLoader = new ImageLoader(mRequestQueue, new ImageLoader.ImageCache() {

            private final LruCache<String, Bitmap> cache = new LruCache<String, Bitmap>(20);

            @Override
            public Bitmap getBitmap(String url) {
                return cache.get(url);
            }

            @Override
            public void putBitmap(String url, Bitmap bitmap) {
                cache.put(url, bitmap);
            }
        });
```

SINGLETON PATTERN

```
public class MySingleton {
    private static MySingleton mInstance;
    private RequestQueue mRequestQueue;
    private ImageLoader mImageLoader;
    private static Context mCtx;

    private MySingleton(Context context) {
        mCtx = context;
        mRequestQueue = getRequestQueue();

        mImageLoader = new ImageLoader(mRequestQueue,
                new ImageLoader.ImageCache() {
            private final LruCache<String, Bitmap>
                    cache = new LruCache<String, Bitmap>(20);

            @Override
            public Bitmap getBitmap(String url) {
                return cache.get(url);
            }

            @Override
            public void putBitmap(String url, Bitmap bitmap) {
                cache.put(url, bitmap);
            }
        });
    }
```

SINGLETON PATTERN

```
 public static synchronized MySingleton getInstance(Context context) {
        if (mInstance == null) {
            mInstance = new MySingleton(context);
        }
        return mInstance;
    }

    public RequestQueue getRequestQueue() {
        if (mRequestQueue == null) {
            // getApplicationContext() is key, it keeps you from leaking the
            // Activity or BroadcastReceiver if someone passes one in.
            mRequestQueue = Volley.newRequestQueue(mCtx.getApplicationContext());
        }
        return mRequestQueue;
    }

    public <T> void addToRequestQueue(Request<T> req) {
        getRequestQueue().add(req);
    }

    public ImageLoader getImageLoader() {
        return mImageLoader;
    }
}
```

USING THE SINGLETON

```
// Get a RequestQueue
RequestQueue queue = MySingleton.getInstance(this.getApplicationContext()).
                     getRequestQueue();

// ...

// Add a request (in this example, called stringRequest) to your RequestQueue.
MySingleton.getInstance(this).addToRequestQueue(stringRequest);
```

PEER2PEER DISCOVERY

HOW TO ESTABLISH A CONNECTION ?

HOW TO ESTABLISH A CONNECTION ?

ChatService

Name
Type

Name
Type

ChatService

HOW TO ESTABLISH A CONNECTION ?

ChatService

Name
Type
Port

Name
Type
Port

ChatService

HOW TO ESTABLISH A CONNECTION ?

ChatService

Name
Type
Port

Name
Type
Port

ChatService

P2P NETWORK OPERATIONS IN ANDROID

Network Service Discovery (NSD)

Register

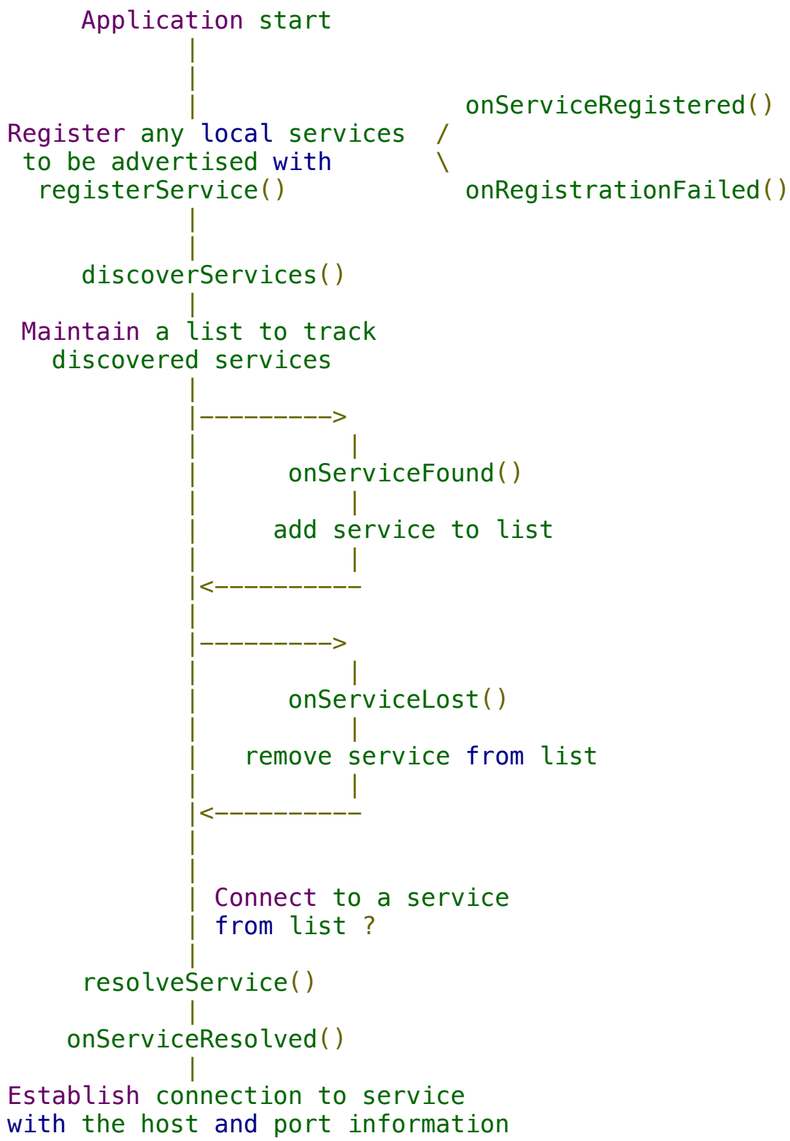
Discovery

Connecting to the network service

Unregister the network service

NSDMANAGER

android.net.nsd.NsdManager



REGISTERING A SERVICE

- Create a `NsdServiceInfo` object
- Create a `RegistrationListener` object
- Get a reference to the `NsdManager`
- Register the service with the service info object and the registration listener.

SERVICE REGISTRATION

android.net.nsd.NsdServiceInfo

```
NsdServiceInfo serviceInfo  = new NsdServiceInfo();
serviceInfo.setServiceName("NsdChat");
serviceInfo.setServiceType("_http._tcp.");
serviceInfo.setPort(port);
```

Service Name

Service Type

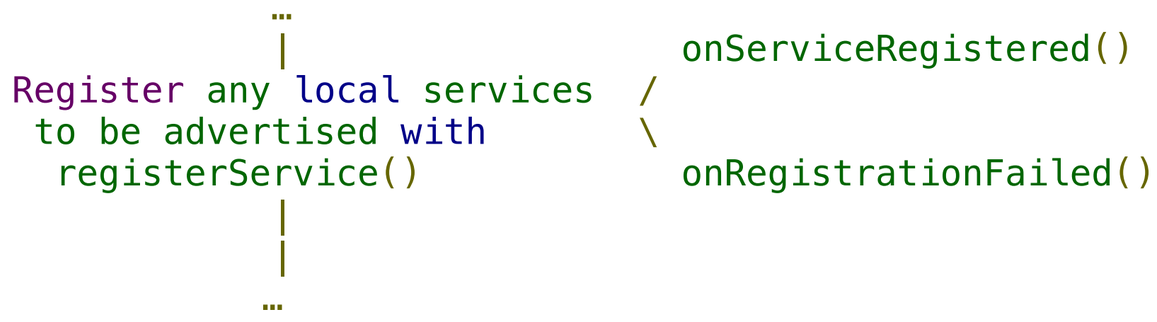
Service Port

SERVICE REGISTRATION

android.net.nsd.NsdManager.RegistrationListener

```
public void initializeRegistrationListener() {
    mRegistrationListener = new NsdManager.RegistrationListener() {

        @Override
        public void onServiceRegistered(NsdServiceInfo NsdServiceInfo) {
            // Save the service name.  Android may have changed it in order to
            // resolve a conflict, so update the name you initially requested
            // with the name Android actually used.
            mServiceName = NsdServiceInfo.getServiceName();
        }

        @Override
        public void onRegistrationFailed(NsdServiceInfo serviceInfo, int errorCode) {
            // Registration failed!  Put debugging code here to determine why.
        }

        @Override
        public void onServiceUnregistered(NsdServiceInfo arg0) {
            // Service has been unregistered.  This only happens when you call
            // NsdManager.unregisterService() and pass in this listener.
        }

        @Override
        public void onUnregistrationFailed(NsdServiceInfo serviceInfo, int errorCode) {
            // Unregistration failed.  Put debugging code here to determine why.
        }
    };
}
```

SERVICE REGISTRATION




```
public void registerService(int port) {
    NsdServiceInfo serviceInfo  = new NsdServiceInfo();
    serviceInfo.setServiceName("NsdChat");
    serviceInfo.setServiceType("_http._tcp.");
    serviceInfo.setPort(port);

    mNsdManager = Context.getSystemService(Context.NSD_SERVICE);

    mNsdManager.registerService(
            serviceInfo, NsdManager.PROTOCOL_DNS_SD, mRegistrationListener);
}
```

ACCEPTING INCOMING CONNECTIONS

```
public void initializeServerSocket() {
    // Initialize a server socket on the next available port.
    mServerSocket = new ServerSocket(0);

    // Store the chosen port.
    mLocalPort =  mServerSocket.getLocalPort();
    ...
}
```

DISCOVER SERVICES

```
public void initializeDiscoveryListener() {

    // Instantiate a new DiscoveryListener
    mDiscoveryListener = new NsdManager.DiscoveryListener() {

        //  Called as soon as service discovery begins.
        @Override
        public void onDiscoveryStarted(String regType) {
            Log.d(TAG, "Service discovery started");
        }

        @Override
        public void onServiceFound(NsdServiceInfo service) {
            // A service was found!  Do something with it.
            Log.d(TAG, "Service discovery success" + service);
        }
```

Service, attribute, host and port details not yet retrieved

DISCOVER SERVICES

```
        @Override
        public void onServiceLost(NsdServiceInfo service) {
            // When the network service is no longer available.
            // Internal bookkeeping code goes here.
            Log.e(TAG, "service lost" + service);
        }

        @Override
        public void onDiscoveryStopped(String serviceType) {
            Log.i(TAG, "Discovery stopped: " + serviceType);
        }

        @Override
        public void onStartDiscoveryFailed(String serviceType, int errorCode) {
            Log.e(TAG, "Discovery failed: Error code:" + errorCode);
            mNsdManager.stopServiceDiscovery(this);
        }

        @Override
        public void onStopDiscoveryFailed(String serviceType, int errorCode) {
            Log.e(TAG, "Discovery failed: Error code:" + errorCode);
            mNsdManager.stopServiceDiscovery(this);
        }
    };

}
```

DISCOVER SERVICES

```
mNsdManager.discoverServices(SERVICE_TYPE, NsdManager.PROTOCOL_DNS_SD, mDiscoveryListener);
```

SERVICE RESOLVING

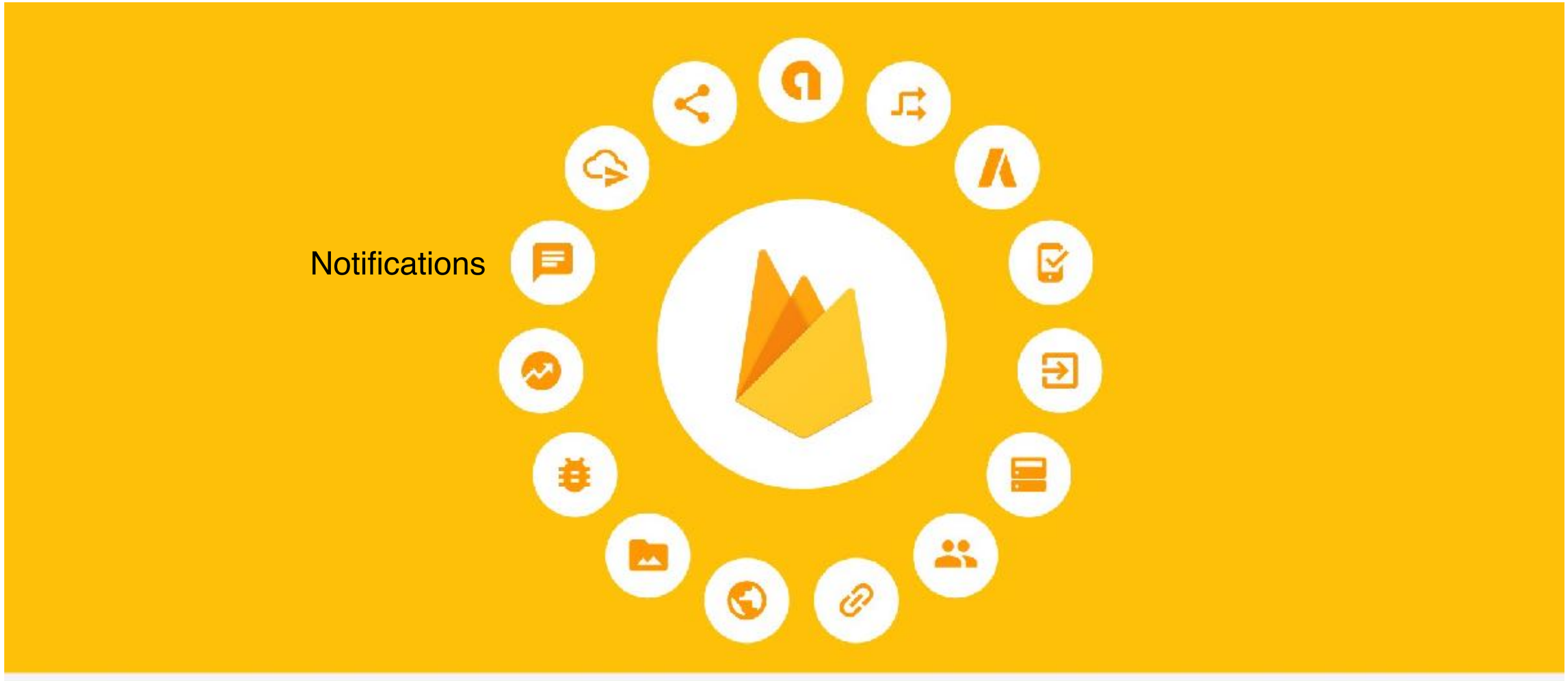
```
public void onServiceFound(NsdServiceInfo service) {
    // A service was found!  Do something with it.
    Log.d(TAG, "Service discovery success" + service);
    if (!service.getServiceType().equals(SERVICE_TYPE)) {
        // Service type is the string containing the protocol and
        // transport layer for this service.
        Log.d(TAG, "Unknown Service Type: " + service.getServiceType());
    } else if (service.getServiceName().equals(mServiceName)) {
        // The name of the service tells the user what they'd be
        // connecting to. It could be "Bob's Chat App".
        Log.d(TAG, "Same machine: " + mServiceName);
    } else if (service.getServiceName().contains("NsdChat")){
        mNsdManager.resolveService(service, mResolveListener);
    }
}
```

SERVICE RESOLVING

```
mResolveListener = new NsdManager.ResolveListener() {

        @Override
        public void onResolveFailed(NsdServiceInfo serviceInfo, int errorCode) {
            // Called when the resolve fails.  Use the error code to debug.
            Log.e(TAG, "Resolve failed" + errorCode);
        }

        @Override
        public void onServiceResolved(NsdServiceInfo serviceInfo) {
            Log.e(TAG, "Resolve Succeeded. " + serviceInfo);

            if (serviceInfo.getServiceName().equals(mServiceName)) {
                Log.d(TAG, "Same IP.");
                return;
            }
            mService = serviceInfo;
            int port = mService.getPort();
            InetAddress host = mService.getHost();
        }
    };
```

PUSH NOTIFICATIONS

FIREBASE



https://firebase.google.com/

FIREBASE

```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.0.0'
    }
}
```

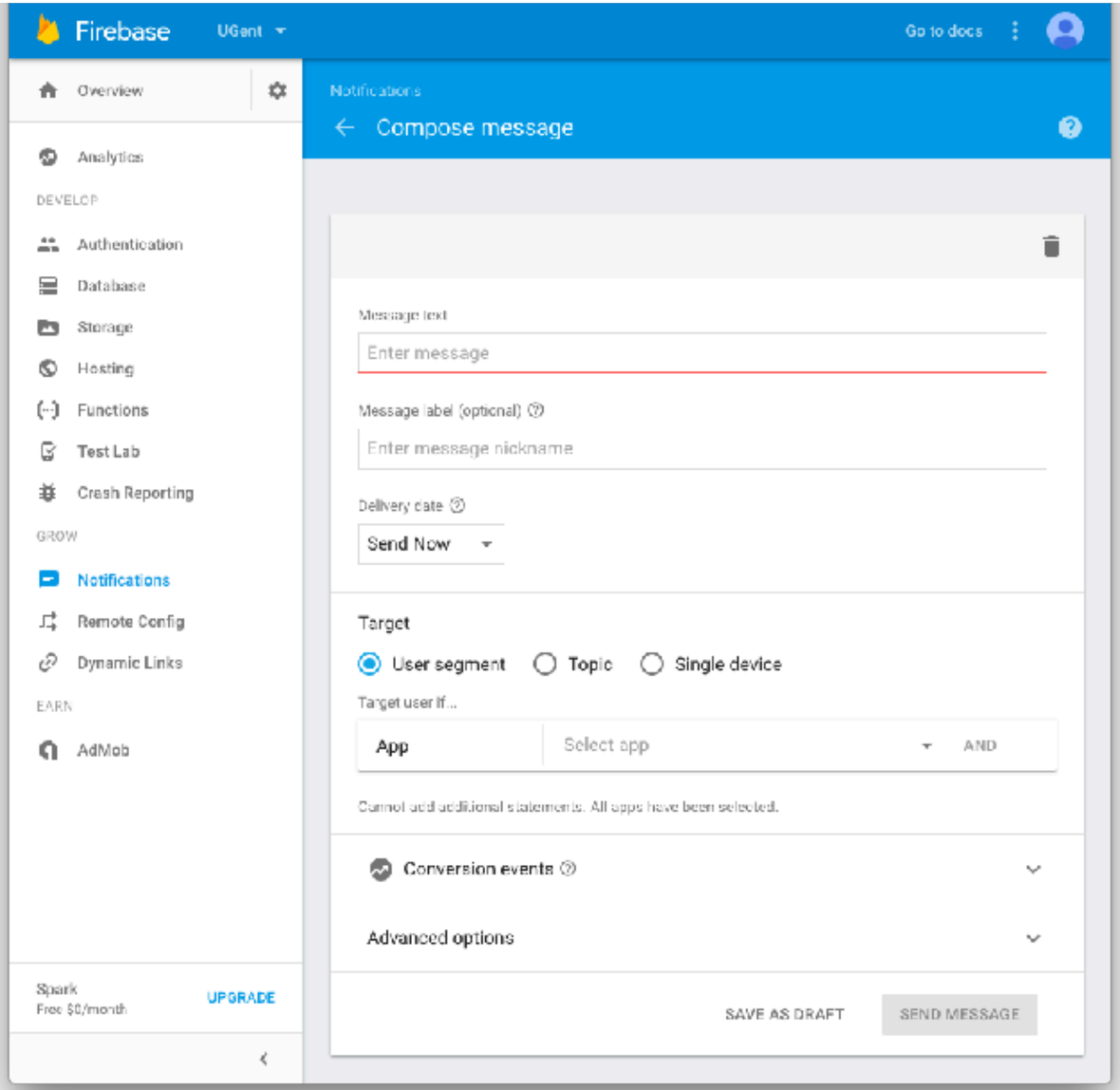
FIREBASE

```
apply plugin: 'com.android.application'

android {
  // ...
}

dependencies {
  // ...
  compile 'com.google.firebase:firebase-core:10.2.1'

  // Getting a "Could not find" error? Make sure you have
  // the latest Google Repository in the Android SDK manager
}
```

// ADD THIS AT THE BOTTOM

```
apply plugin: 'com.google.gms.google-services'
```

SEND A NOTIFICATION ON THE CONSOLE



INTERCEPTING NOTIFICATIONS

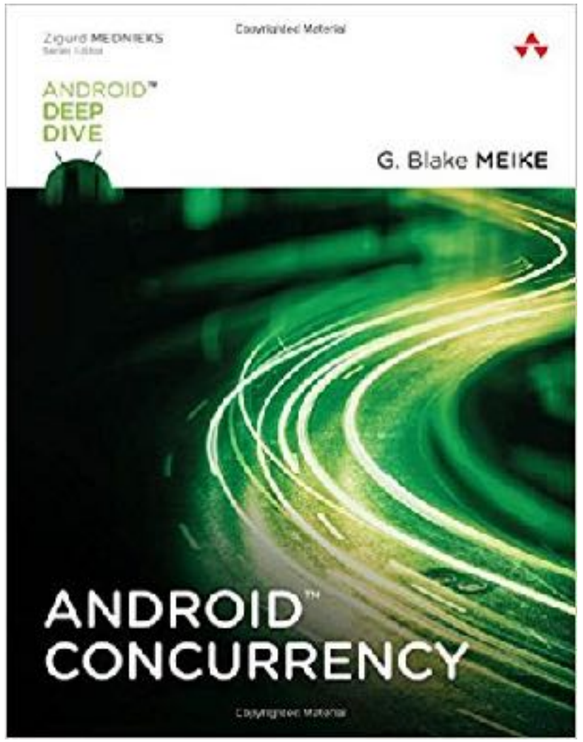
```
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "FCM Service";
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        //Handle messages here.
        Log.d(TAG, "From: " + remoteMessage.getFrom());
        Log.d(TAG, "Notification Message Body: " + remoteMessage.getNotification().getBody());
    }
}
```

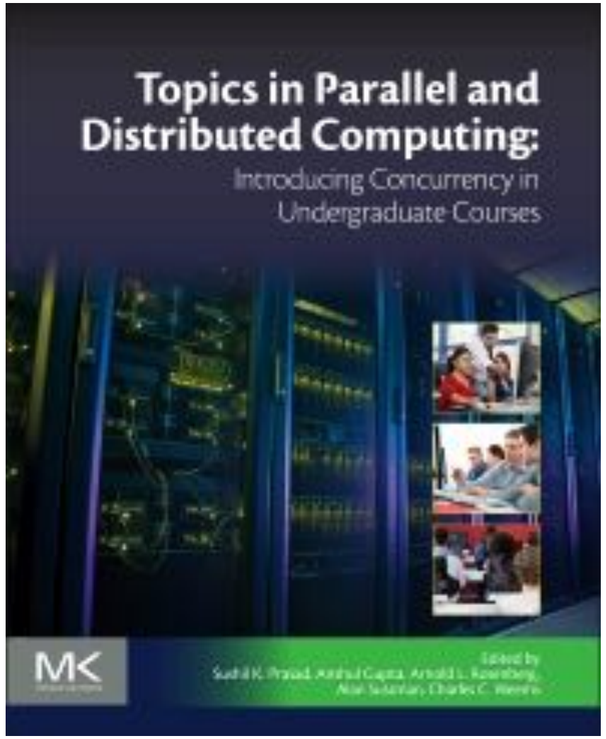
FIREBASE TOKEN ID

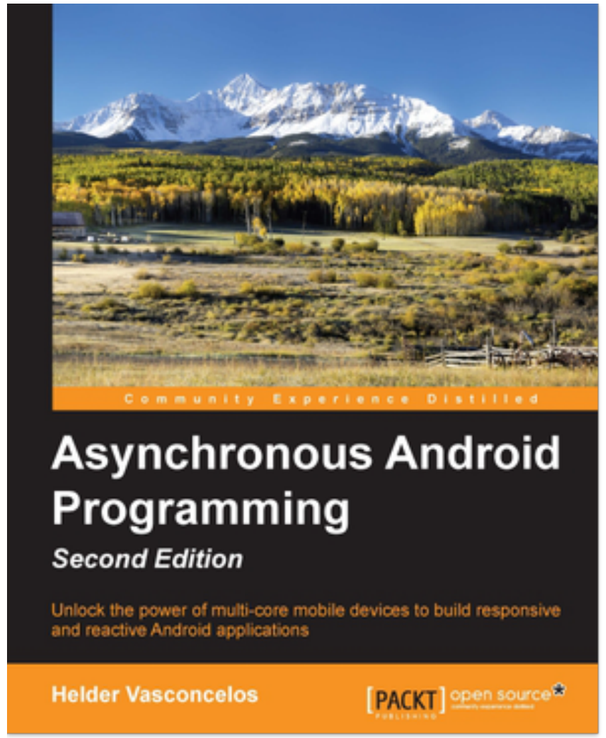
```
public class FirebaseIDService extends FirebaseInstanceIdService {
    private static final String TAG = "FirebaseIDService";

    @Override
    public void onTokenRefresh() {
        // Get updated InstanceID token.
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();
        Log.d(TAG, "Refreshed token: " + refreshedToken);
        sendRegistrationToServer(refreshedToken);
    }

    private void sendRegistrationToServer(String token) {
        // Add custom implementation, as needed.
    }
}
```

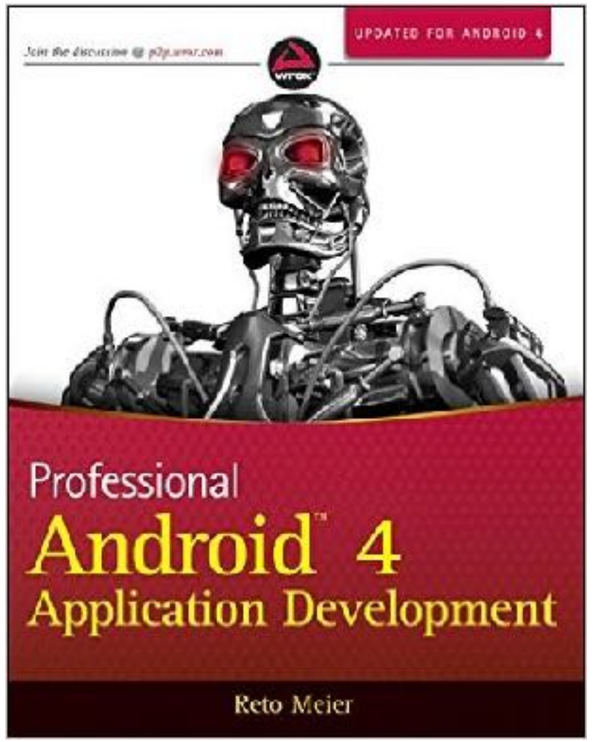
USED RESOURCES

BOOKS









LOT'S OF EXPLANATION WITH THE CODE EXAMPLES

RESEARCH ARTICLES

Fallacies

A note on distributed computing

No Silver Bullet

OO distributed programming ...

INTERNET RESOURCES

https://developer.android.com/training/building-connectivity.html

Easy to follow tutorials ! However, concurrency and synchronicity is not always explained well!

https://developer.android.com/guide/index.html

Read the API’s they are really good !

LANGUAGES BASED ON THE ACTOR MODEL