SISTEMAS DISTRIBUÍDOS

Aula 3

Descoberta de serviços

Tratamento de falhas

SUMMARY

Service discovery

Handling faults

- Communication errors
- Detecting the failure of a component

WHY DISCOVER WHERE A SERVICE IS RUNNING?

```
String hostname = "localhost:8080";
if( args.length > 0)
        hostname = args[0];
```

The User has to provide the address of the server

```
ClientConfig config = new ClientConfig();
Client client = ClientBuilder.newClient(config);
```

```
URI baseURI = UriBuilder.fromUri("http://" + hostname +
"/").build();
```

```
WebTarget target = client.target(baseURI);

Endpoint[] endpoints = target.path("/contacts")
                             .request()
                             .accept(MediaType.APPLICATION_JSON)
                             .get(Endpoint[].class);
```

How to perform service discovery?

Think about a distributed system, running in a local network, with multiple machines, where each machine might provide multiple web services, and these web services might some times have to be migrated between machines, and maybe change ports…

ChatWebService

FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

1st Alternative: Server Initiated

ChatWebService

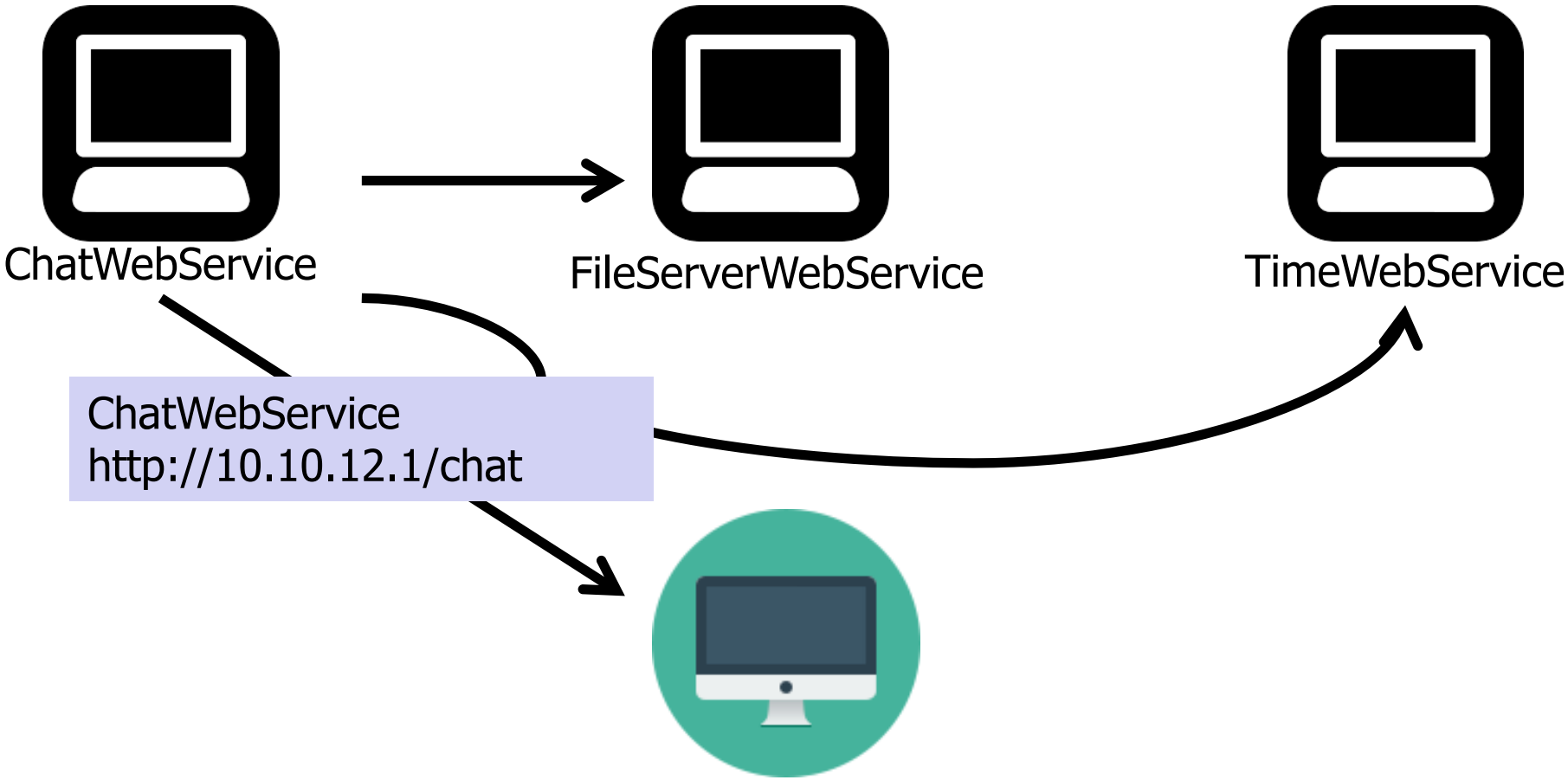
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

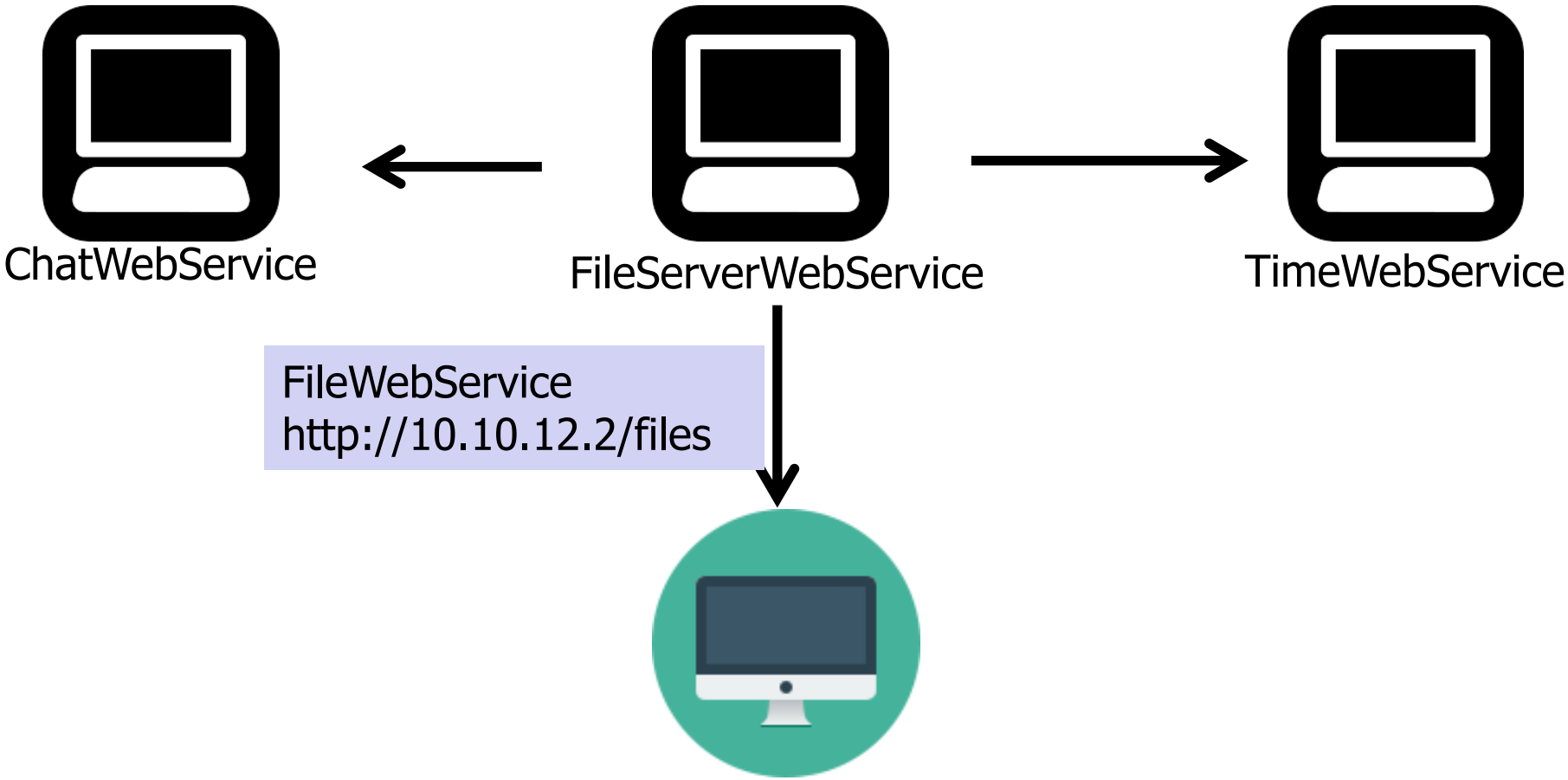
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

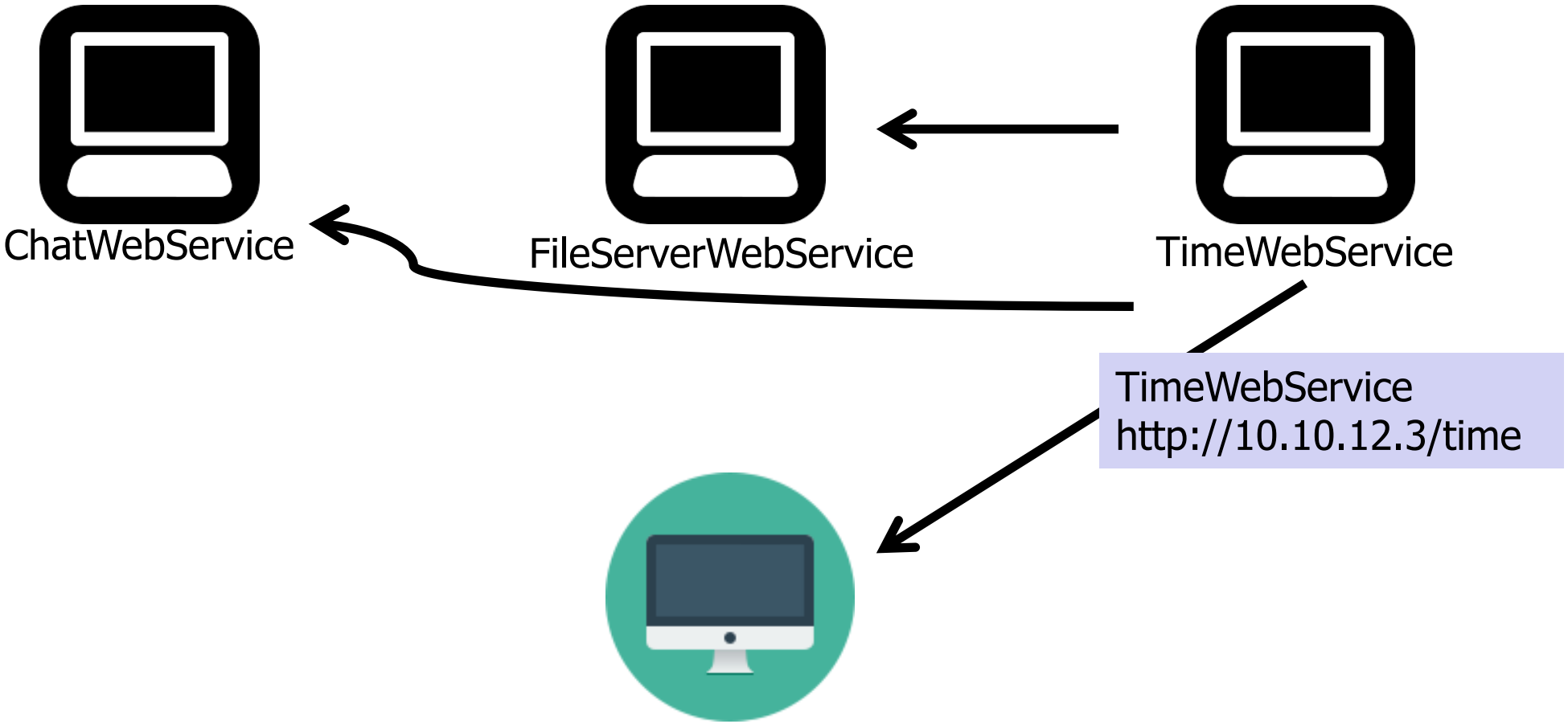
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

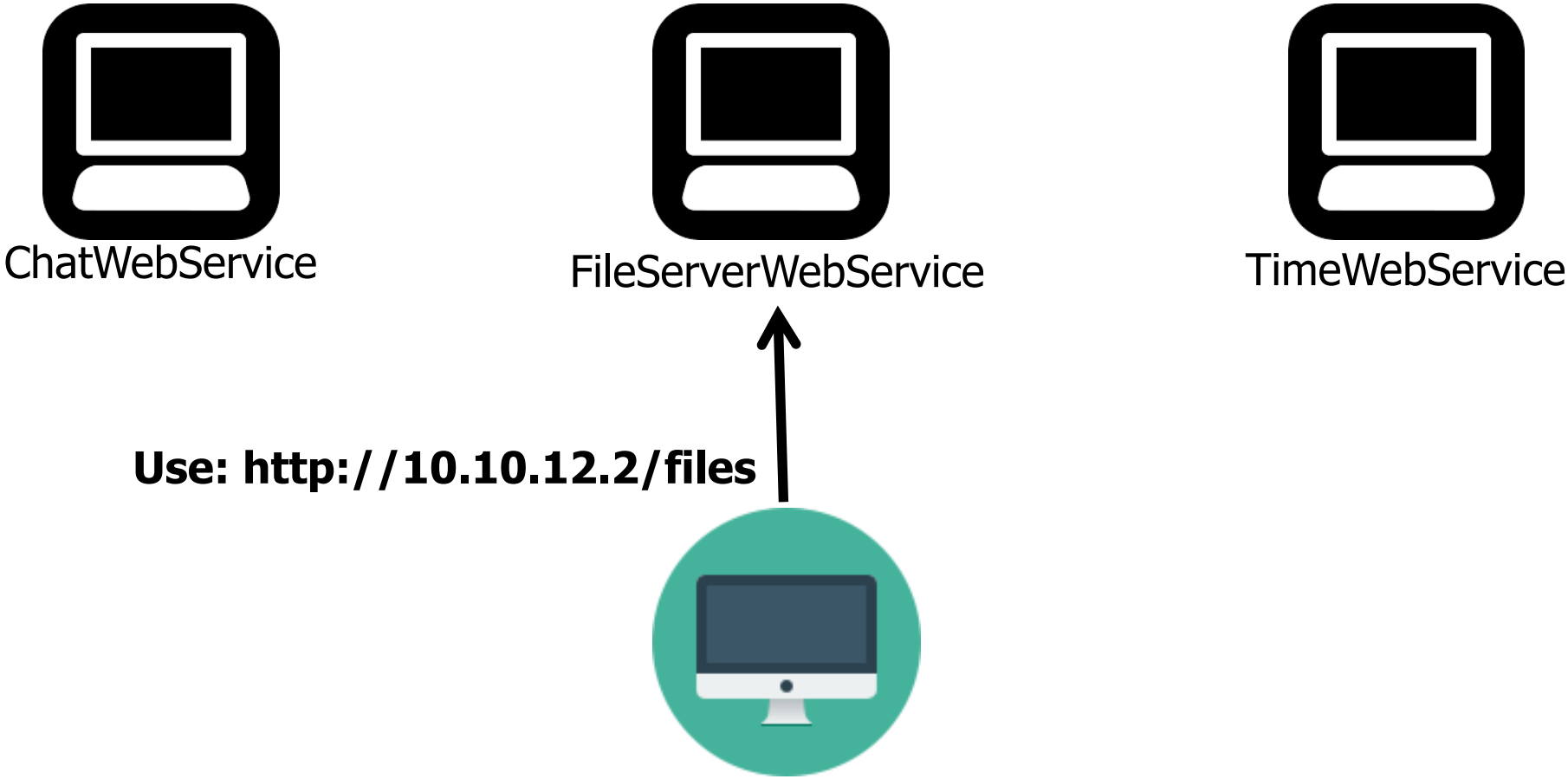
1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

1st Alternative: Server Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

2nd Alternative: Client Initiated

ChatWebService

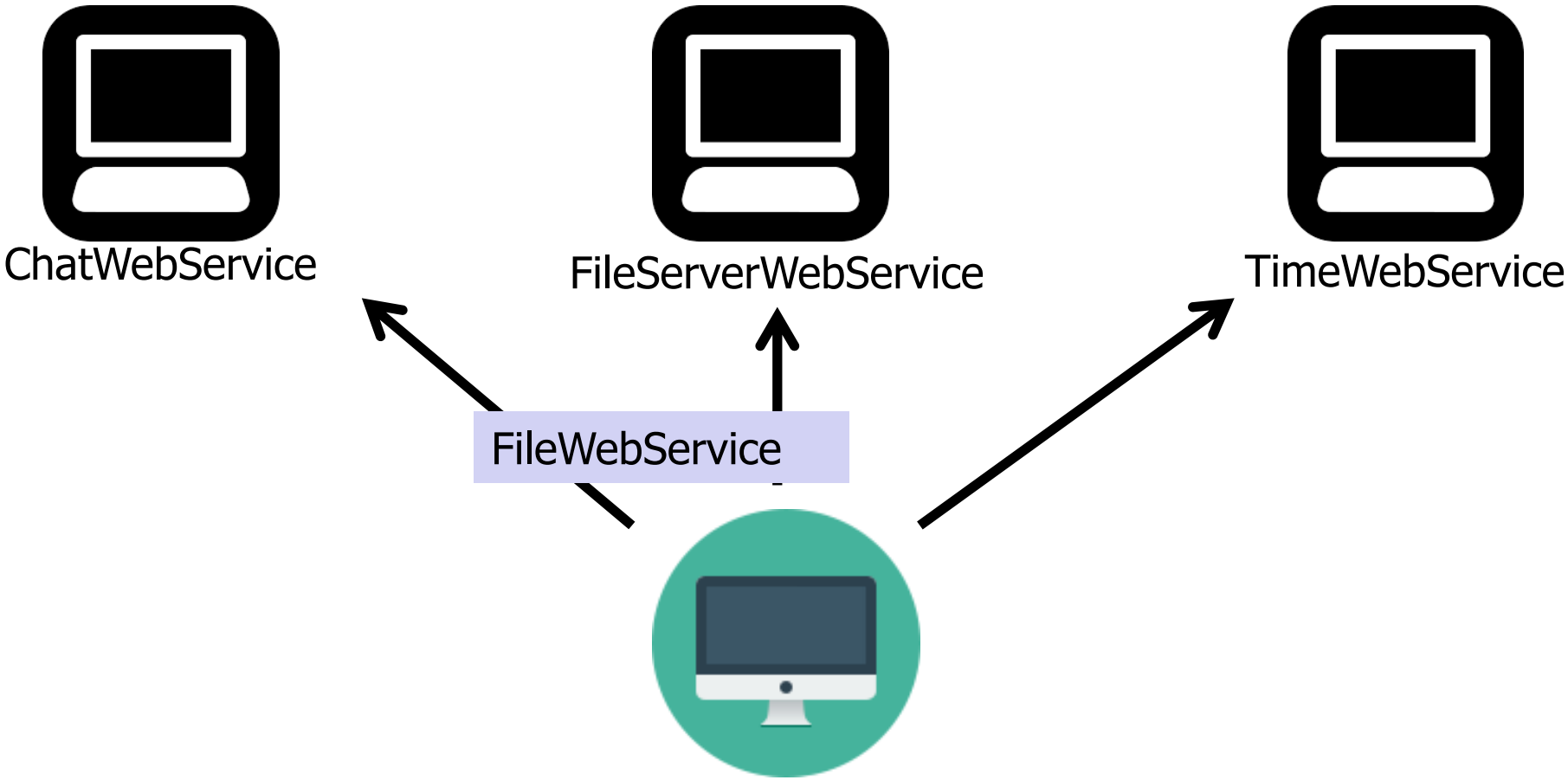
FileServerWebService

TimeWebService

How to perform service discovery?

One solution is to use **IP Multicast**

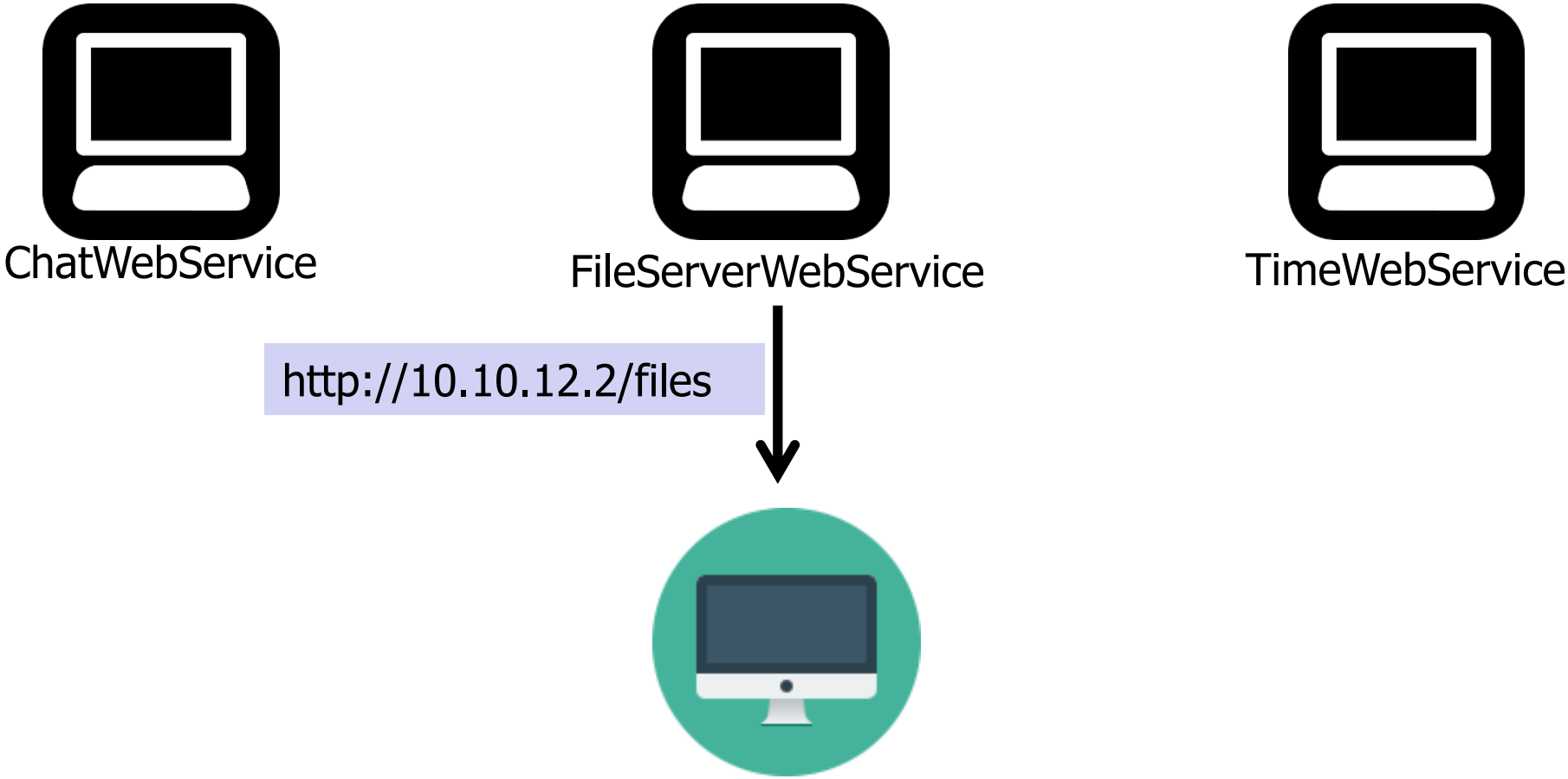
2nd Alternative: Client Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

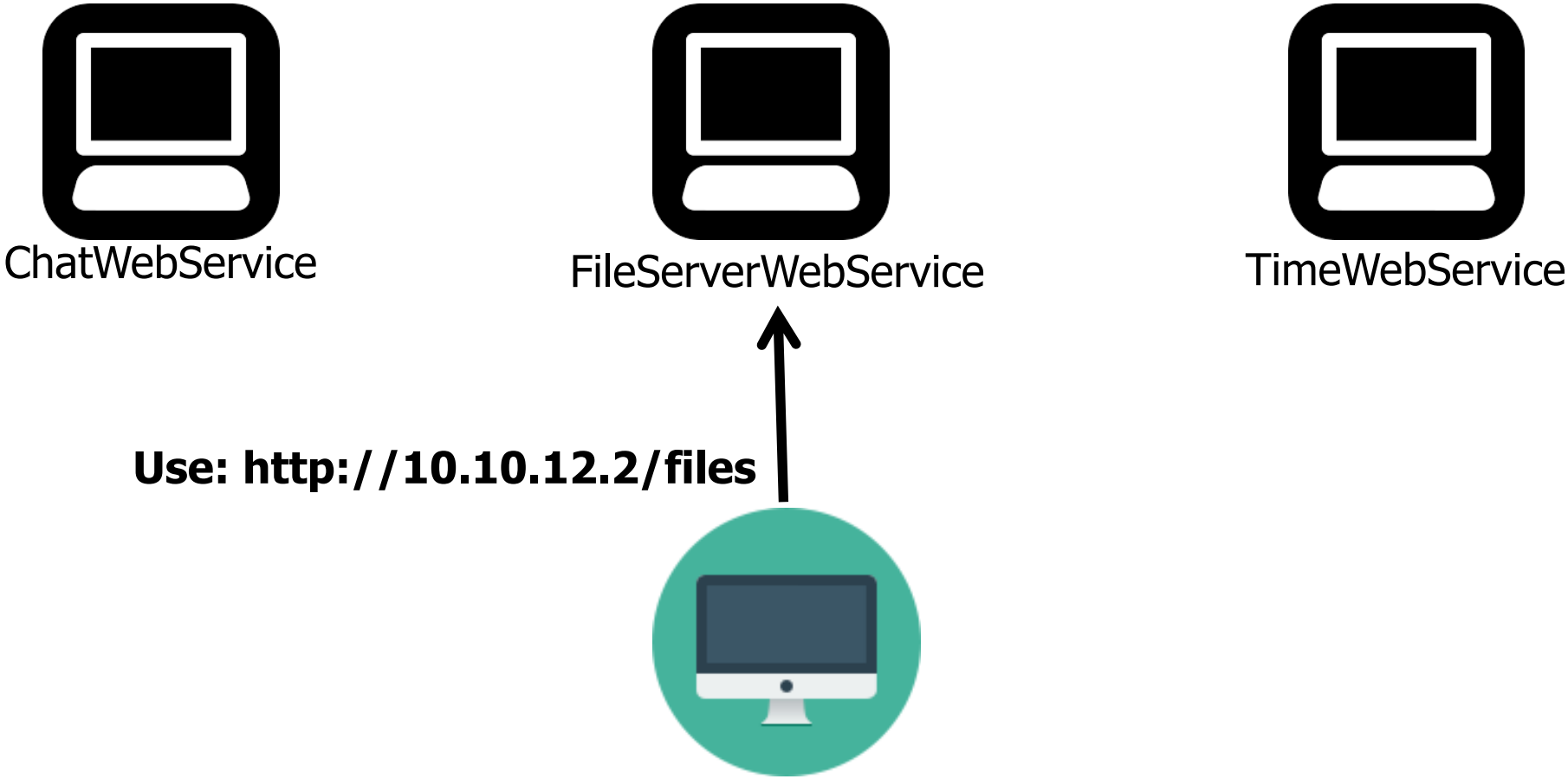
2nd Alternative: Client Initiated



How to perform service discovery?

One solution is to use **IP Multicast**

2nd Alternative: Client Initiated



WHERE TO PERFORM SERVICE DISCOVERY? (SERVER)

```
public class RendezVousServer {
      public static void main(String[] args) throws Exception {
        int port = 8080;
        if( args.length > 0)
                port = Integer.parseInt(args[0]);

        URI baseUri =
             UriBuilder.fromUri("http://0.0.0.0/").port(port).build();
        ResourceConfig config = new ResourceConfig();
        config.register( new RendezVousResources() );

        JdkHttpServerFactory.createHttpServer(baseUri, config);

        System.err.println("REST RendezVous Server ready @ " + baseUri
+ " : local IP = " + InetAddress.getLocalHost().getHostAddress());
        }
}
```

Where to perform Service Discovery? (Server)

```
public class RendezVousServer {
      public static void main(String[]
        int port = 8080;
        if( args.length > 0)
              port = Integer.parseInt(

        URI baseUri =
             UriBuilder.fromUri("http:/
        ResourceConfig config = new Res
        config.register( new RendezVous

        JdkHttpServerFactory.createHttp

        System.err.println("REST Rendezvous Server ready @ " + baseUri
+ " : local IP = " + InetAddress.getLocalHost().getHostAddress());
        }
}
```

What to do here?

1. Create a Multicast Socket
2. Join a Multicast Group (to receive multicast messages)
3. Wait for a discovery request
4. When receive a discovery request reply with the URL of the service
5. Continue waiting for other discovery request

Where to perform Service Discovery? (Client)

```
String hostname = "localhost:8080";
if( args.length > 0)
        hostname = args[0];

ClientConfig config = new ClientConfig();
Client client = ClientBuilder.newClient(config);

URI baseURI = UriBuilder.fromUri("http://" + hostname +
"/").build();

WebTarget target = client.target(baseURI);

Endpoint[] endpoints = target.path("/contacts")
                             .request()
                             .accept(MediaType.APPLICATION_JSON)
                             .get(Endpoint[].class);
```

Where to perform Service Discovery? (Client)

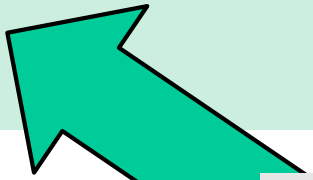
```
String hostname = "localhost:8080";
if( args.length > 0)
        hostname = args[0];

ClientConfig config = new Client
Client client = ClientBuilder.newCl

URI baseURI = UriBuilder.fromUri("h
"/").build();

WebTarget target = client.target(ba

Endpoint[] endpoints = target.path(
                              .request(
                              .accept(M
                              .get(Endp
```



What to do here?

1. Create a Multicast Socket
2. Send a discovery request to the correct multicast address (and port)
3. Wait for a discovery reply (*)
4. Extract the URL from the reply and use it to instantiate the WebTarget

(*) recall UDP is best effort, the request may need to be retransmitted after some timeout without a reply.

MULTICAST: SERVER SIDE (EXAMPLE)

java.net / Sockets Multicast / Servidor

```
package aula0;

import java.io.* ;
import java.net.* ;

public class MulticastServer {
   public static void main(String[] args) throws Exception {
      final InetAddress address = InetAddress.getByName( args[0] ) ;
      if( ! address.isMulticastAddress()) {
        System.out.println( "Use range : 224.0.0.0 -- 239.255.255.255");
        System.exit( 1);
      }
      MulticastSocket socket = new MulticastSocket( 9000 ) ;
      socket.joinGroup( address);
      while( true ) {
         byte[] buffer = new byte[65536] ;
         DatagramPacket packet = new DatagramPacket( buffer, buffer.length ) ;
         socket.receive( packet ) ;
         System.out.write( packet.getData(), 0, packet.getLength() ) ;
      }
   }
}
```

MULTICAST: CLIENT SIDE (EXAMPLE)

java.net / Sockets Multicast / Cliente

```
package aula0;

import java.io.* ;
import java.net.* ;

public class MulticastClient {
   public static void main(String[] args) throws Exception {
      final int port = 9000 ;
      final InetAddress address = InetAddress.getByName( args[0] ) ;
      if( ! address.isMulticastAddress()) {
        System.out.println( "Use range : 224.0.0.0 -- 239.255.255.255");
      }
      MulticastSocket socket = new MulticastSocket() ;
      do {
         byte[] input = (readLine() + "\n").getBytes();
         DatagramPacket packet = new DatagramPacket( input, input.length ) ;
         packet.setAddress( address ) ;
         packet.setPort( port ) ;
         socket.send( packet ) ;
      } while( ! input.equals(".\n") ) ;
      socket.close() ;
   }
   public static String readLine() throws Exception {
      return new BufferedReader( new InputStreamReader( System.in ) ).readLine() ;
   }
```

SUMMARY

Service discovery

Handling faults

- Communication errors
- Detecting the failure of a component

COMMUNICATION ERRORS IN REST

Communication errors may occur when invoking a REST server

- Server may be unavailable
- The server may fail while processing the request

When an error occurs, a **javax.ws.rs.ProcessingException** is thrown

Programs must catch these exceptions

COMMUNICATION ERRORS IN WEB SERVICES (2)

```
ClientConfig config = new ClientConfig();
Client client = ClientBuilder.newClient(config);
URI baseURI = UriBuilder.fromUri("http://localhost:8080/").build();
WebTarget target = client.target(baseURI);

Endpoint[] endpoints = null;
try {
     endpoints = target.path("/contacts")
                .request()
                .accept(MediaType.APPLICATION_JSON)
                .get(Endpoint[].class);
    System.err.println("as array: " + Arrays.asList(endpoints));
} catch( ProcessingException e) {
    // an error has occurred
}
```

What to do when an error occurs...

- Application-specific
  - Try to invoke the method again
  - Use another server
  - Handle error in application

RETRY EXECUTING A METHOD

```
boolean executed = false;

for( int i = 0; ! executed && i < 3; i++) {    // number of tries
    try {
        server.someop( someargs);
        executed = true;
    } catch( RuntimeException e) {
        if( i < 2) {
            try {     // wait some time
                Thread.sleep( 5000);
            } catch( InterruptedException e1) {
                // do nothing
            }
        }
    }
}
```

SUMMARY

Service discovery

Handling faults

- Communication errors
- Detecting the failure of a component

FAILURES

It is impossible to reliable detect the failure of a component, but it is possible to suspect the event with high probability

Detecting failed/inactive components in a distributed system

- KeepAlive (Ping)
- Heartbeat

KEEP-ALIVE

Process A periodically sends a request to process B asking if it is still alive

If process B does not reply to several consecutive requests, it is assumed to be inactive

HEARBEAT

Process A periodically sends an heartbeat message notifying that it is alive

Process B that receives the message registers the time when the message is received

Process B considers A is inactive if it has not received an heartbeat after some time (with this time depending on how frequent messages are sent)

SUMMARY

Automatic discovery of the rendez-vous server [2 points]
It should be possible to automatically find the rendez-vous server. (…)

Service discovery

Handling faults

- Communication errors
- Detecting the failure of a component

Up-to date rendez-vous server [2 points]
The rendez-vous server must maintain up-to-date information about indexing servers. To this end, the information about an indexing server must be discarded if the servers stops.