DISTRIBUTED SYSTEMS

Project 2

Nuno Preguiça, Sérgio Duarte, Dina Borrego, João Vilalonga

TRABALHO PRÁTICO – SEGURANÇA

A segurança do sistema é garantida por:

1. Utilização de canais seguros com TLS (https://...)
2. Utilização de passwords para autenticas os clientes.

TRABALHO PRÁTICO – SEGURANÇA (CONT.)

Como garantir que um cliente não pode chamar métodos adicionais criados para comunicação entre servidores?

Utilização dum segredo partilhado entre servidores. Envio desse segredo nas operações usadas entre servidores.

Como deve ser passado esse segredo?

- Várias alternativas: parâmetro quando se lança o programa.

Como não deve ser passado esse segredo?

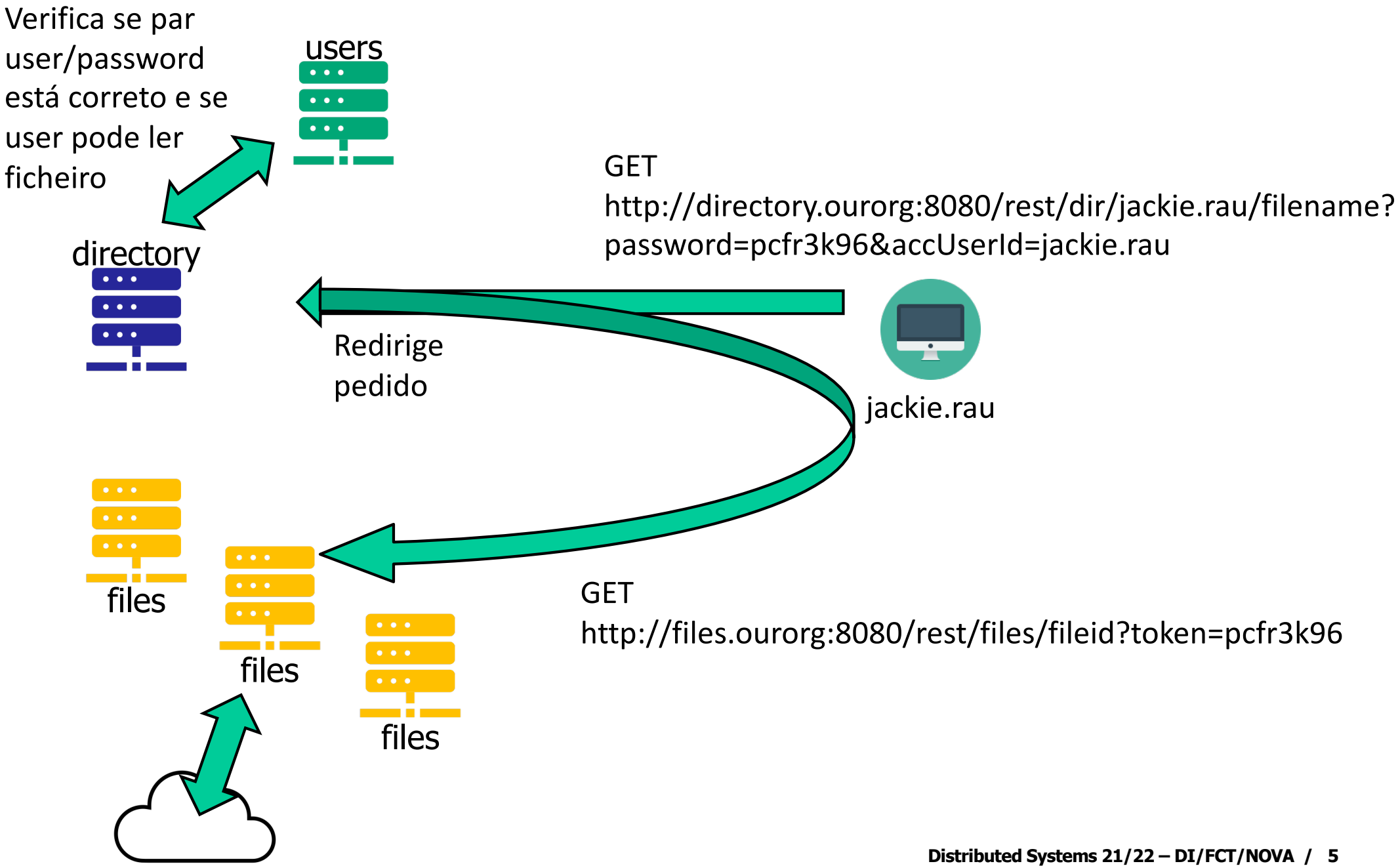
- Codificado como constante no código do programa!!!

TRABALHO PRÁTICO – SEGURANÇA – CONTROLO DE ACESSOS FICHEIROS (CONT.)

Como garantir o controlo de acessos no servidor de ficheiros? Várias alternativas:

1. Clientes não acedem diretamente ao servidor de utilizadores (nem são redirigidos para o servidor de ficheiros) – servidor usam segredo partilhado (ver slide anterior).
2. Utilização dum token, gerado e assinado pelo servidor de diretório, que indica permissão para aceder ao ficheiro. Como? (nas aulas de segurança)

EXECUÇÃO: LER FICHEIRO

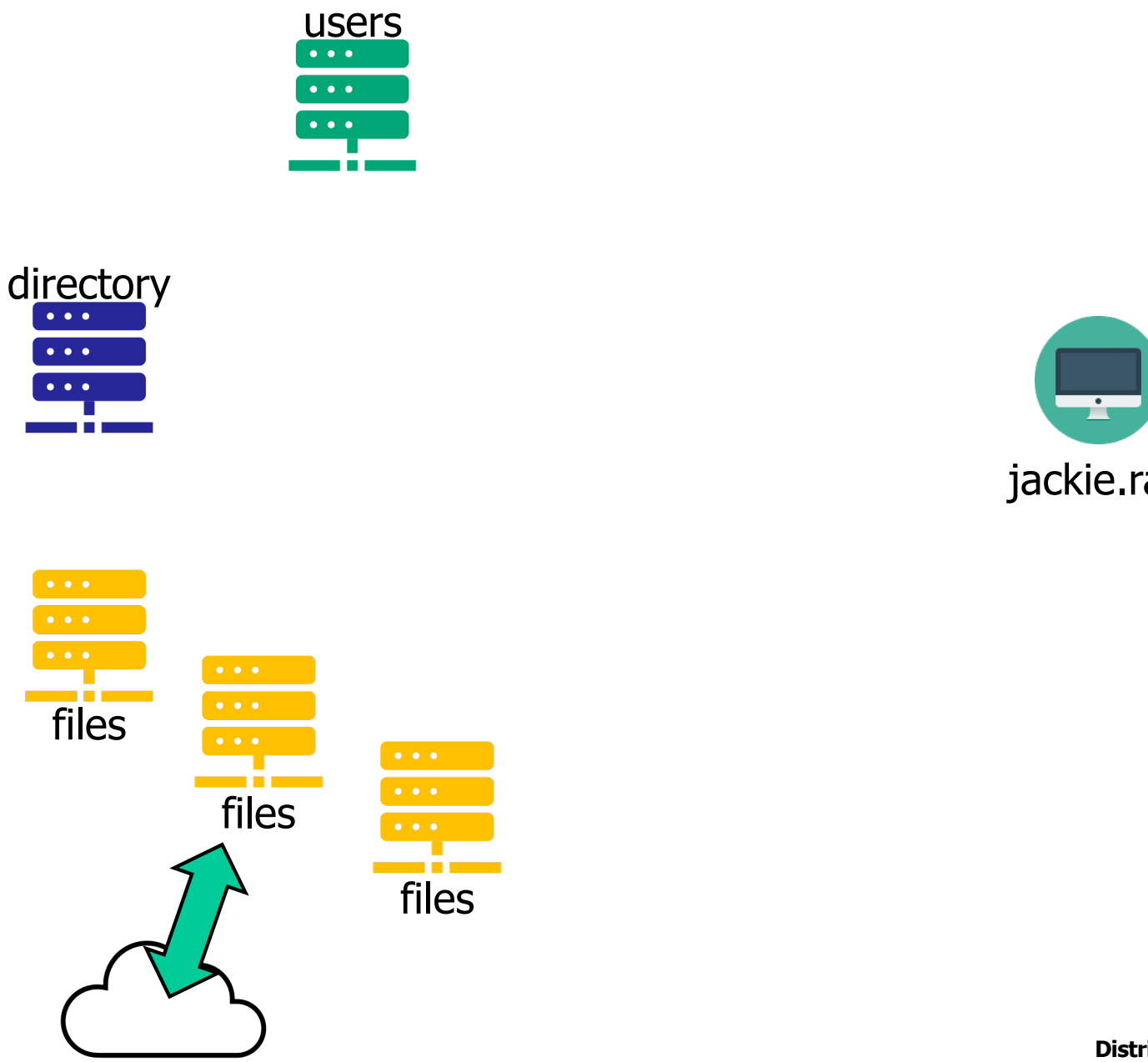


Trabalho prático – Interação com serviço externo

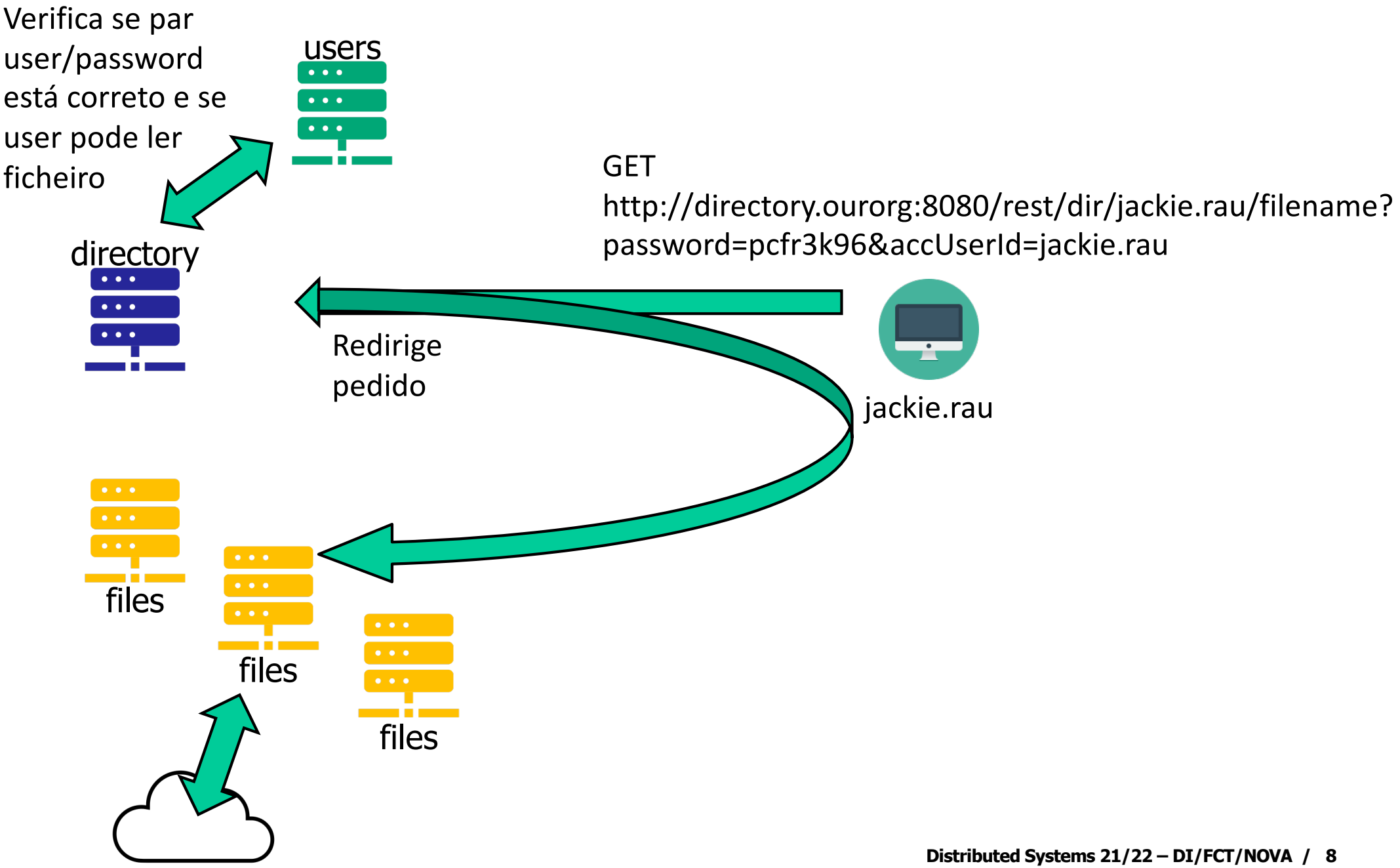
Servidor de ficheiros que funciona como proxy para um serviço externo (e.g. Dropbox).

Operações de leitura e escrita devem ser executadas no serviço externo.

TRABALHO PRÁTICO – INTERAÇÃO COM SERVIÇO EXTERNO



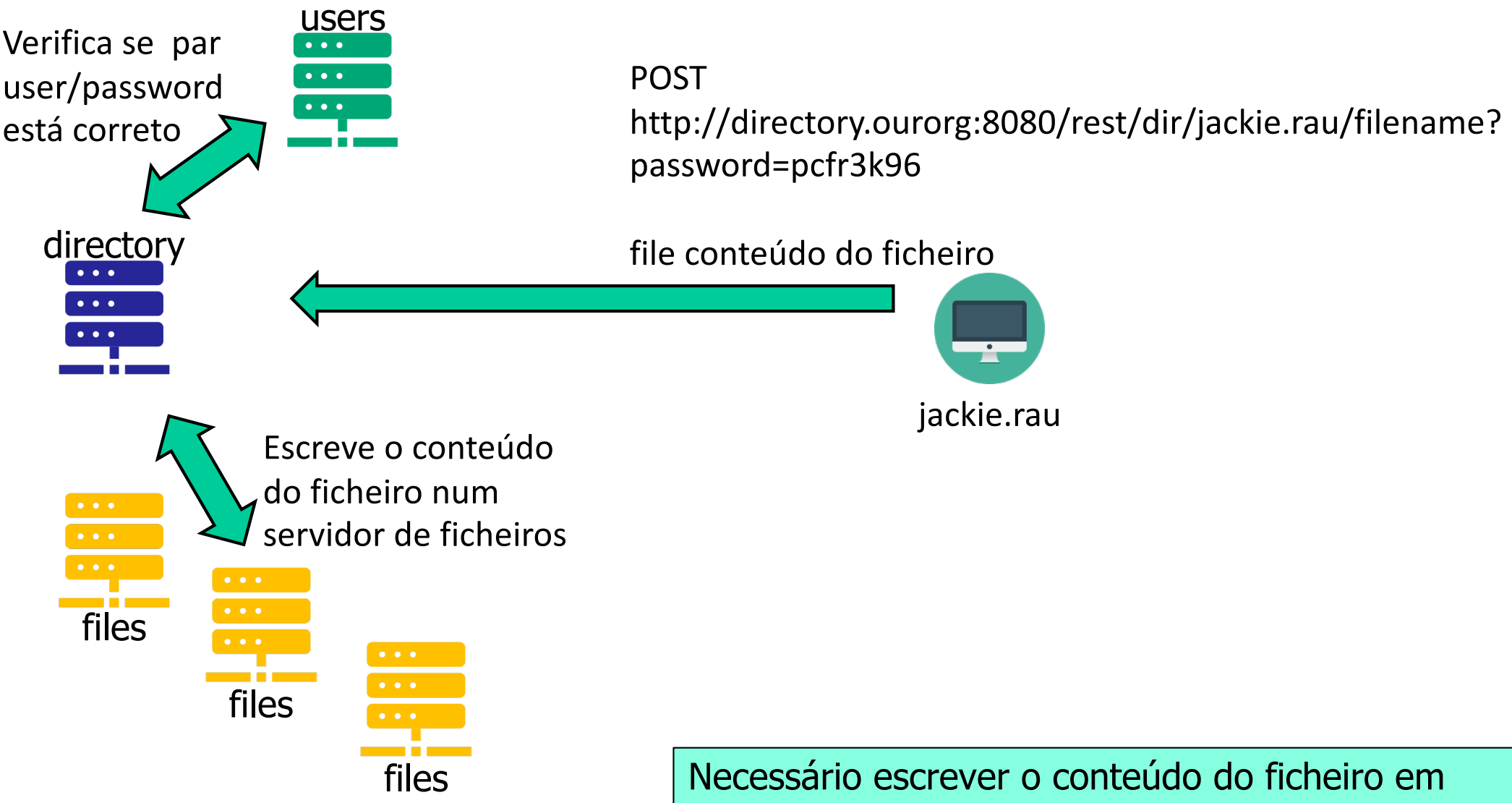
EXECUÇÃO: LER FICHEIRO



TRABALHO PRÁTICO – TOLERÂNCIA A FALHAS DO SERVIDOR DE FICHEIROS

O sistema deve tolerar a falha de 1 servidor de ficheiros.

EXECUÇÃO: ESCREVER FICHEIRO

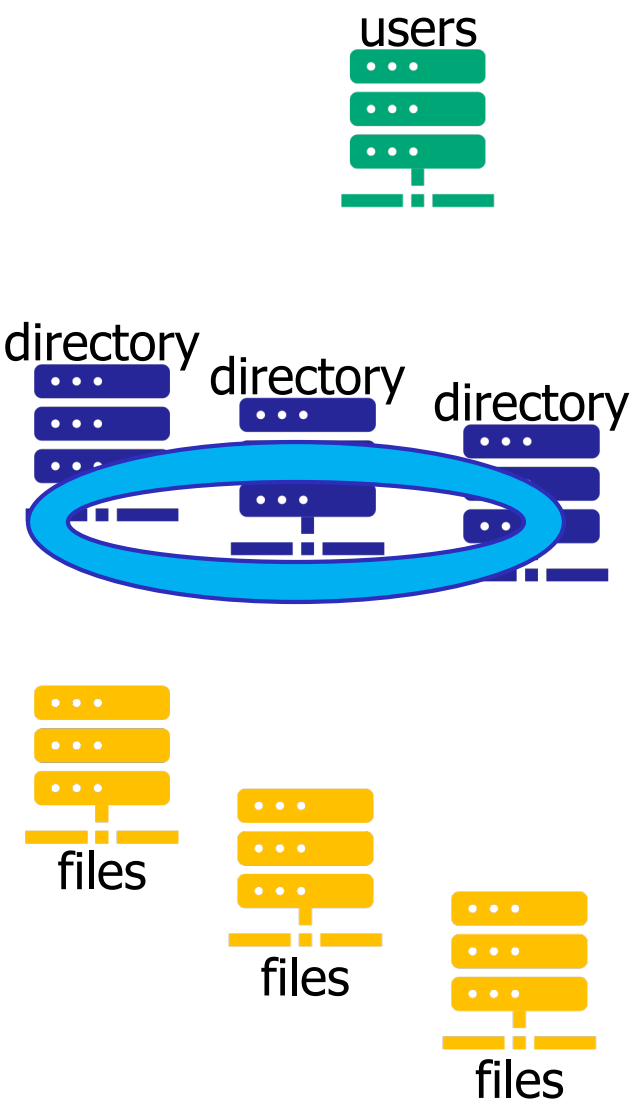


Necessário escrever o conteúdo do ficheiro em mais do que um servidor e manter informação sobre onde o ficheiro está gravado.

TRABALHO PRÁTICO – TOLERÂNCIA A FALHAS DO SERVIDOR DE DIRETÓRIO

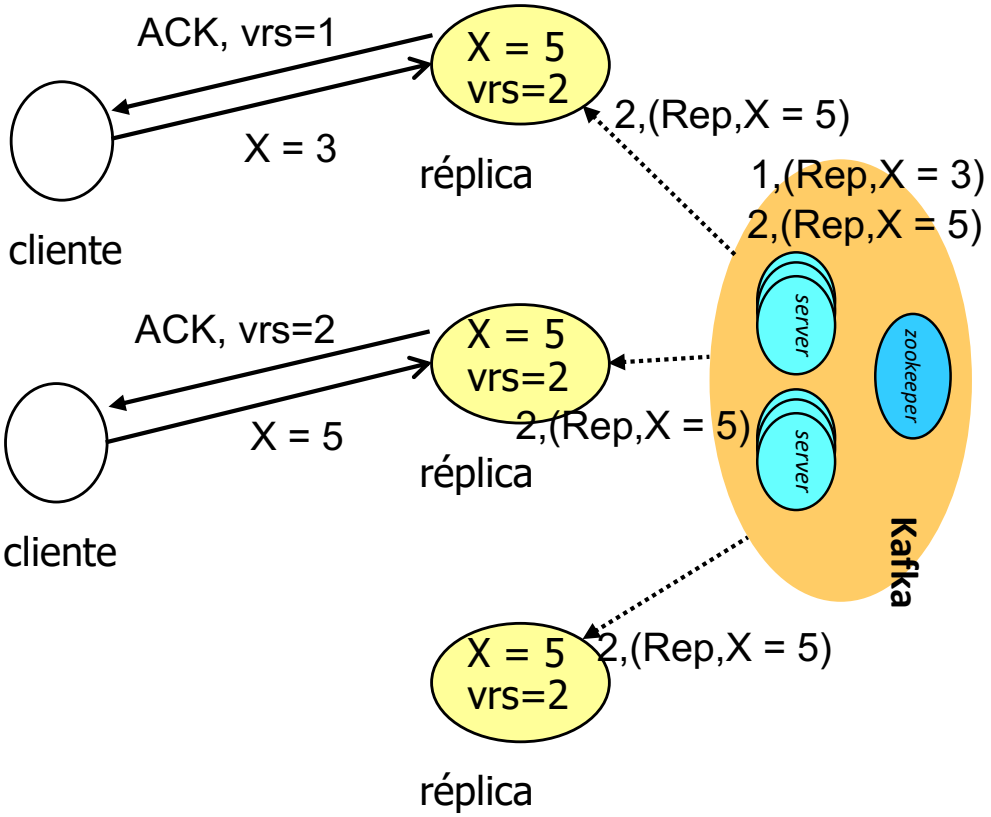
Necessário implementar um mecanismo de replicação do servidor de diretórios.

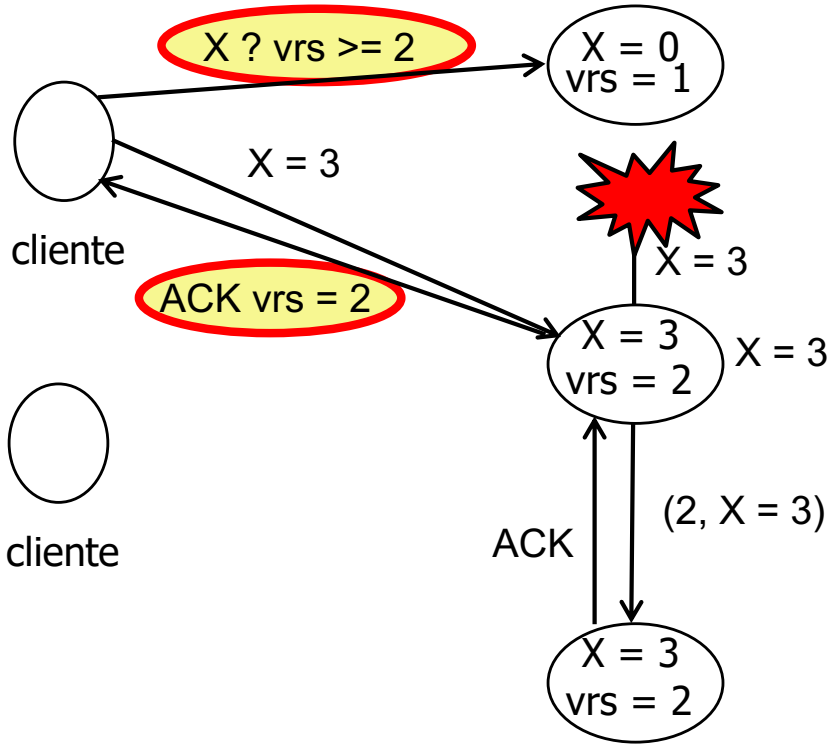
TRABALHO PRÁTICO – REPLICAÇÃO DO SERVIDOR DE DIRETÓRIO



TRABALHO PRÁTICO 2 – COMO ACEDER A VERSÃO TÃO ATUAL COMO A ACEDIDA ANTERIORMENTE

Necessário enviar número de versão nos pedidos e nos resultados das operações.





ENVIAR HEADER

Pode-se adicionar no pedido com o método header.

```
WebTarget target = client.target(serverUrl).path(RestDirectory.PATH);
Response r = target.request().
                    header(RestDirectory.HEADER_VERSION, 1).
                    accept(MediaType.APPLICATION_JSON).
                    get();
```

OBTER VALOR DUM HEADER NUM MÉTODO REST

Pode-se usar a anotação @HeaderParam. Se o header não estiver no pedido, o valor será null.

```
@Path(RestDirectory.PATH)
public interface RestDirectory {

    public static final String PATH="/dir";
    public static final String HEADER_VERSION = "X-DFS-version";

    @POST
    @Path("/{userId}/{filename}")
    @Consumes(MediaType.APPLICATION_OCTET_STREAM)
    @Produces(MediaType.APPLICATION_JSON)
    FileInfo writeFile(@HeaderParam(HEADER_VERSION) Long version,
            @PathParam("filename") String filename, byte []data,
            @PathParam("userId") String userId, @QueryParam("password") String password);
```

ENVIAR O VALOR DO HEADER NO RESULTADO – ALT. 1

Pode-se registar um filtro para adicionar o header. O ReplicationManager seria uma classe que manteria o número da versão do servidor (como o SyncPoint no exemplo a seguir).

```
@Provider
public class VersionFilter implements ContainerResponseFilter {
    ReplicationManager repManager;

    VersionFilter( ReplicationManager repManager) {
        this.repManager = repManager;
    }

    @Override
    public void filter(ContainerRequestContext request,
                       ContainerResponseContext response)
            throws IOException {
        response.getHeaders().add(RestDirectory.HEADER_VERSION,
                                  repManager.getCurrentVersion());
    }
}
```

```
ReplicationManager manager = null;

ResourceConfig config = new ResourceConfig();
config.register(RestDirectoryResource.class);
config.register(new VersionFilter( manager));
```

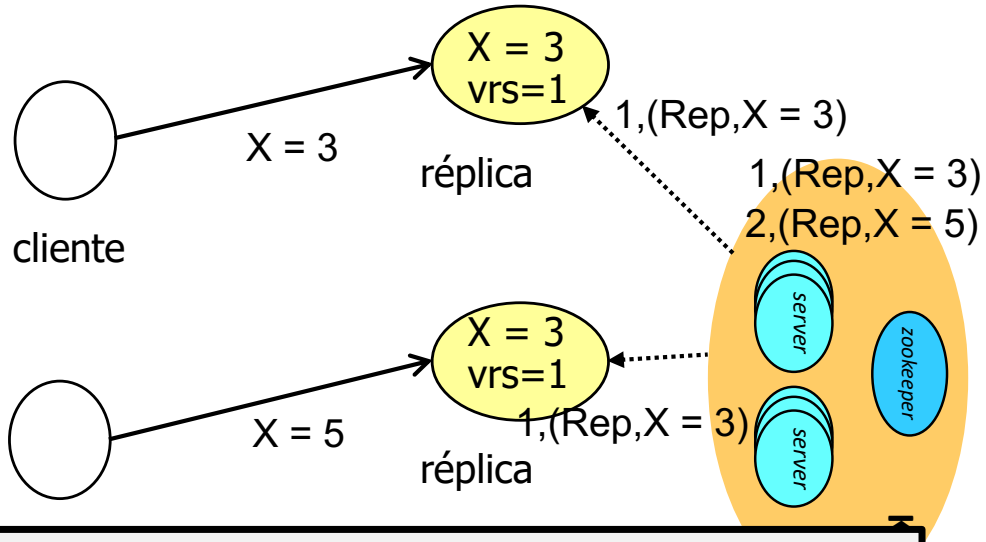
ENVIAR O VALOR DO HEADER NO RESULTADO – ALT. 2

Alternativamente, pode-se enviar o header no momento do envio do resultado.

```
class RestDirectoryResource implements RestDirectory {

    @Override
    public FileInfo writeFile(Long version, String filename,
                              byte[] data, String userId, String password) {
        Log.info( "WriteFile " + filename + " ; version = " + version);

        //...

        throw new WebApplicationException( Response.ok().
                header( RestDirectory.HEADER_VERSION,version).
                entity(fileInfo).build());
    }
```

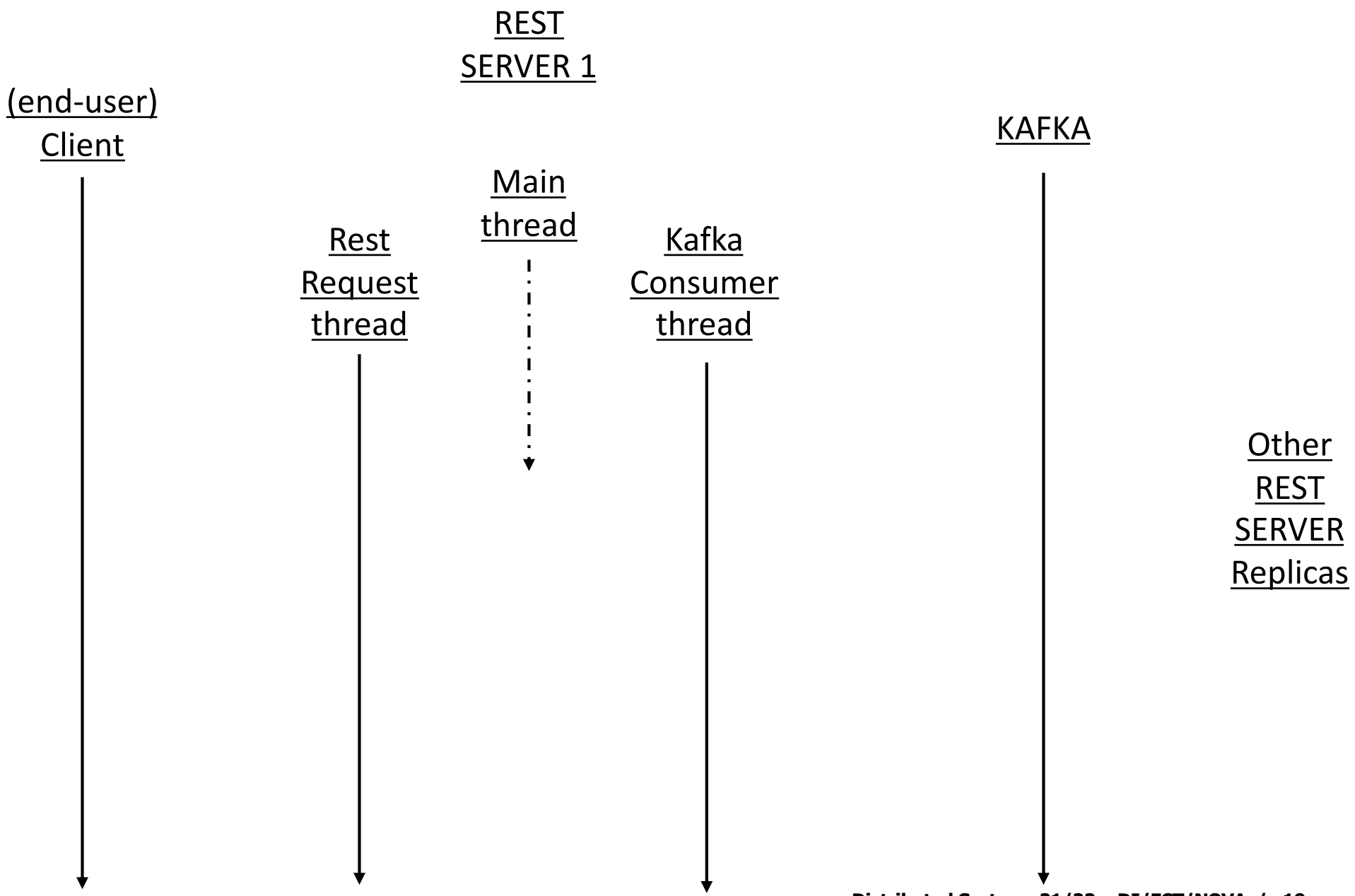
TRABALHO PRÁTICO 2 – REPLICAÇÃO

Na execução dum método é necessário esperar que um processo em background processe uma dada mensagem para continuar.

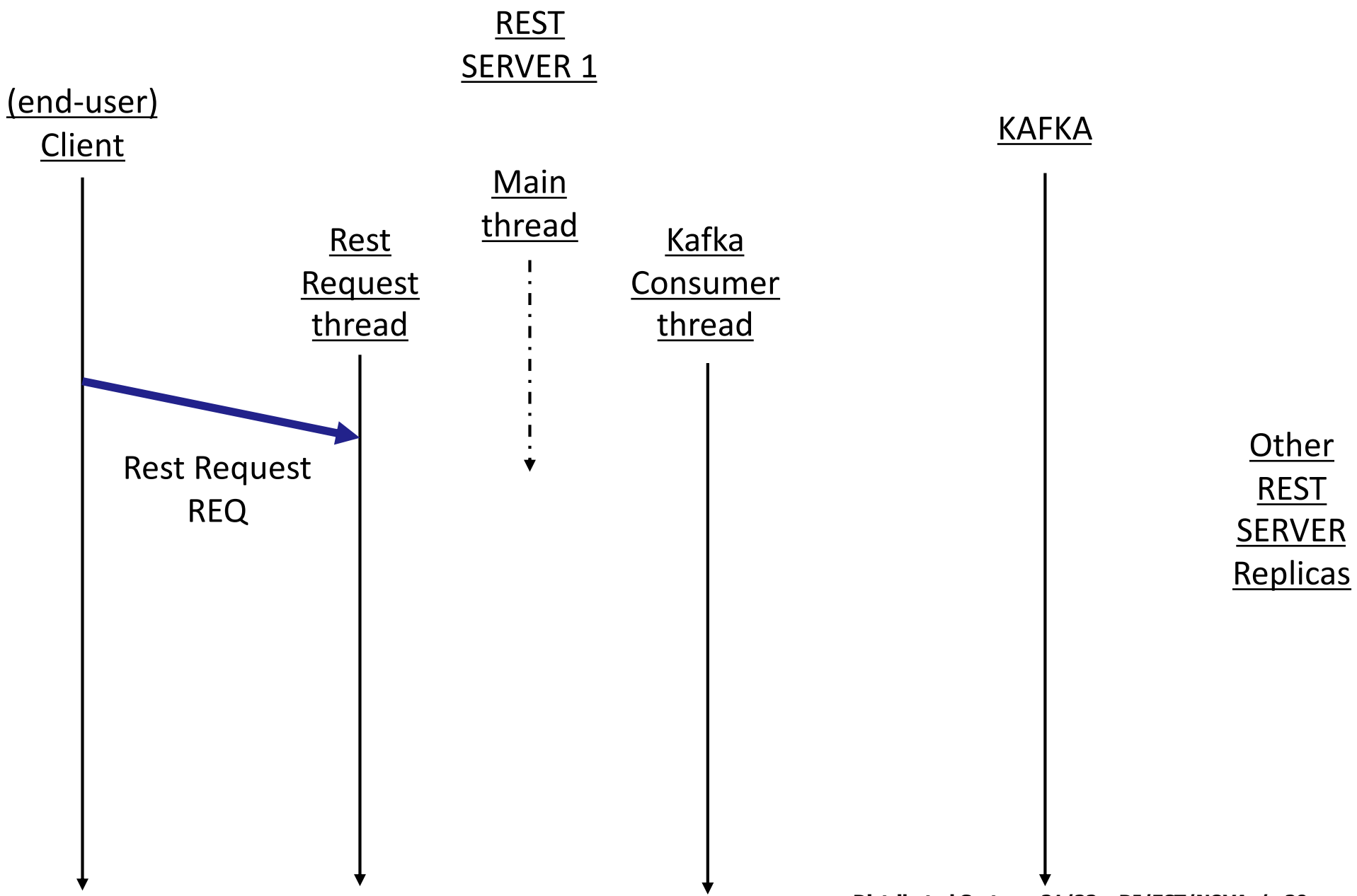



```
ConsumerRecords<String, String> records = consumer.poll(1000);
records.forEach( r -> {
        // Process operation
        System.err.println(r.offset() + "->" + r.topic() + "/" + r.value());
});
```

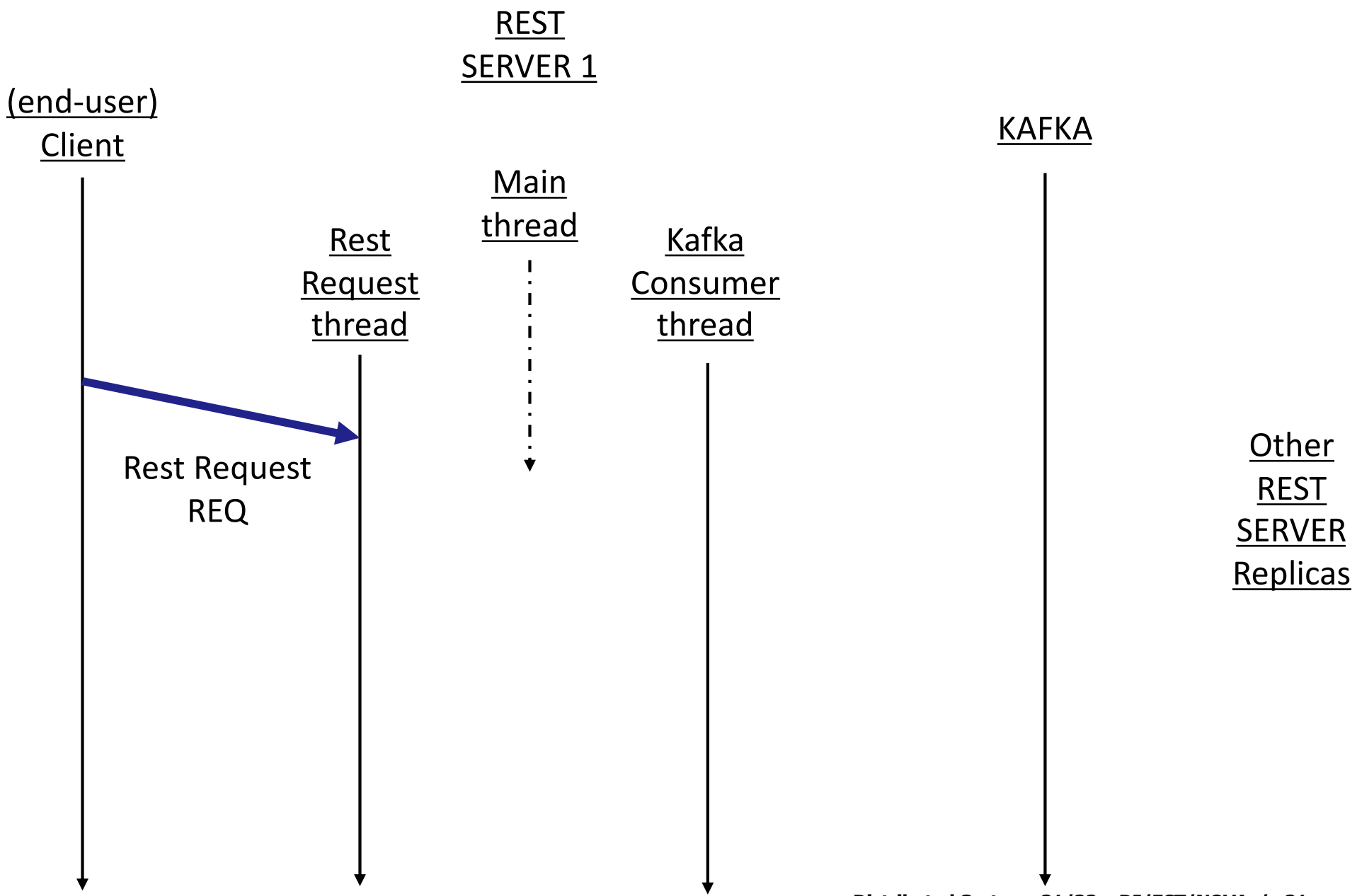
KAFKA: ASYNCHRONY AND REST SERVERS



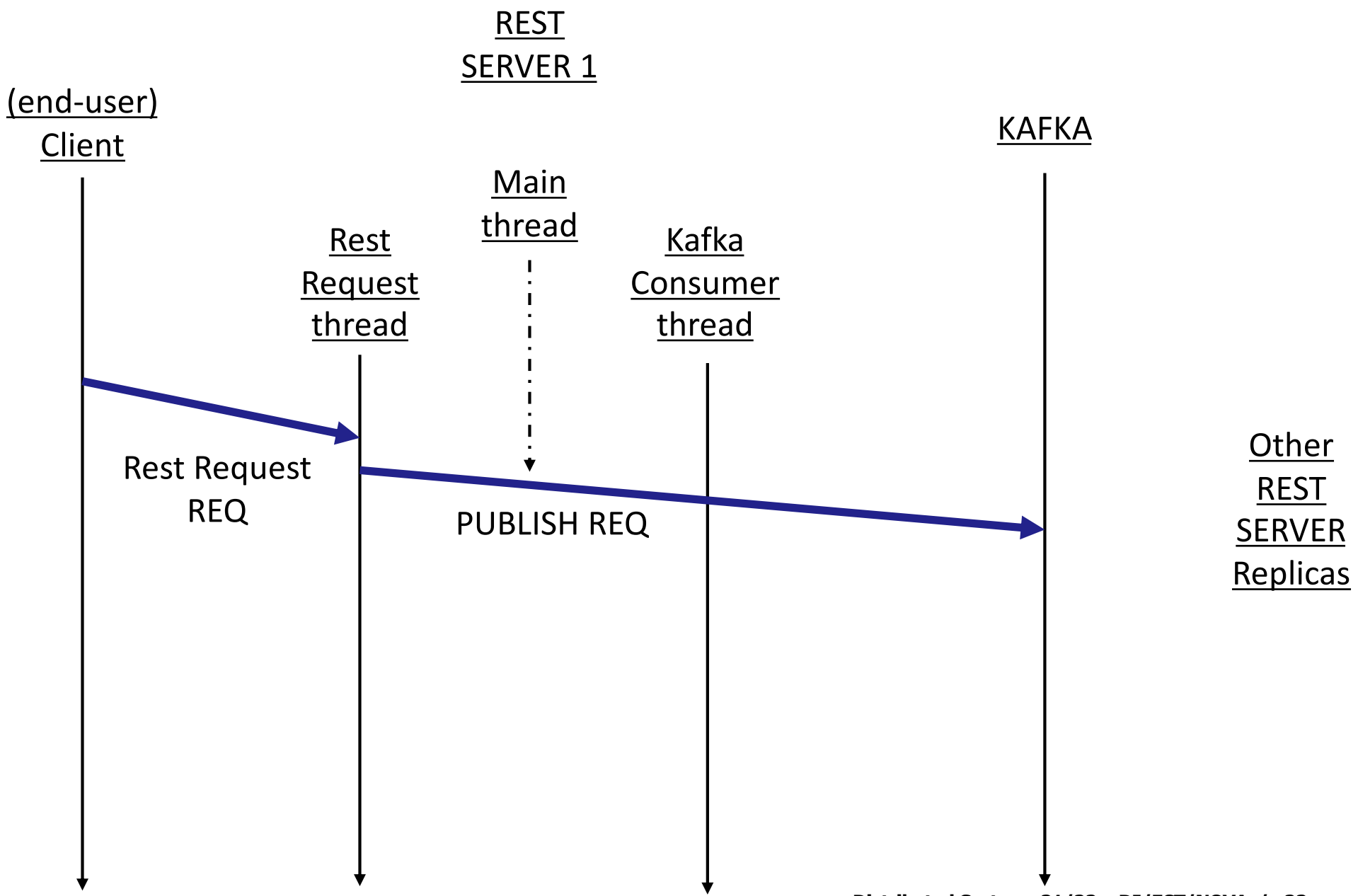
KAFKA: ASYNCHRONY AND REST SERVERS



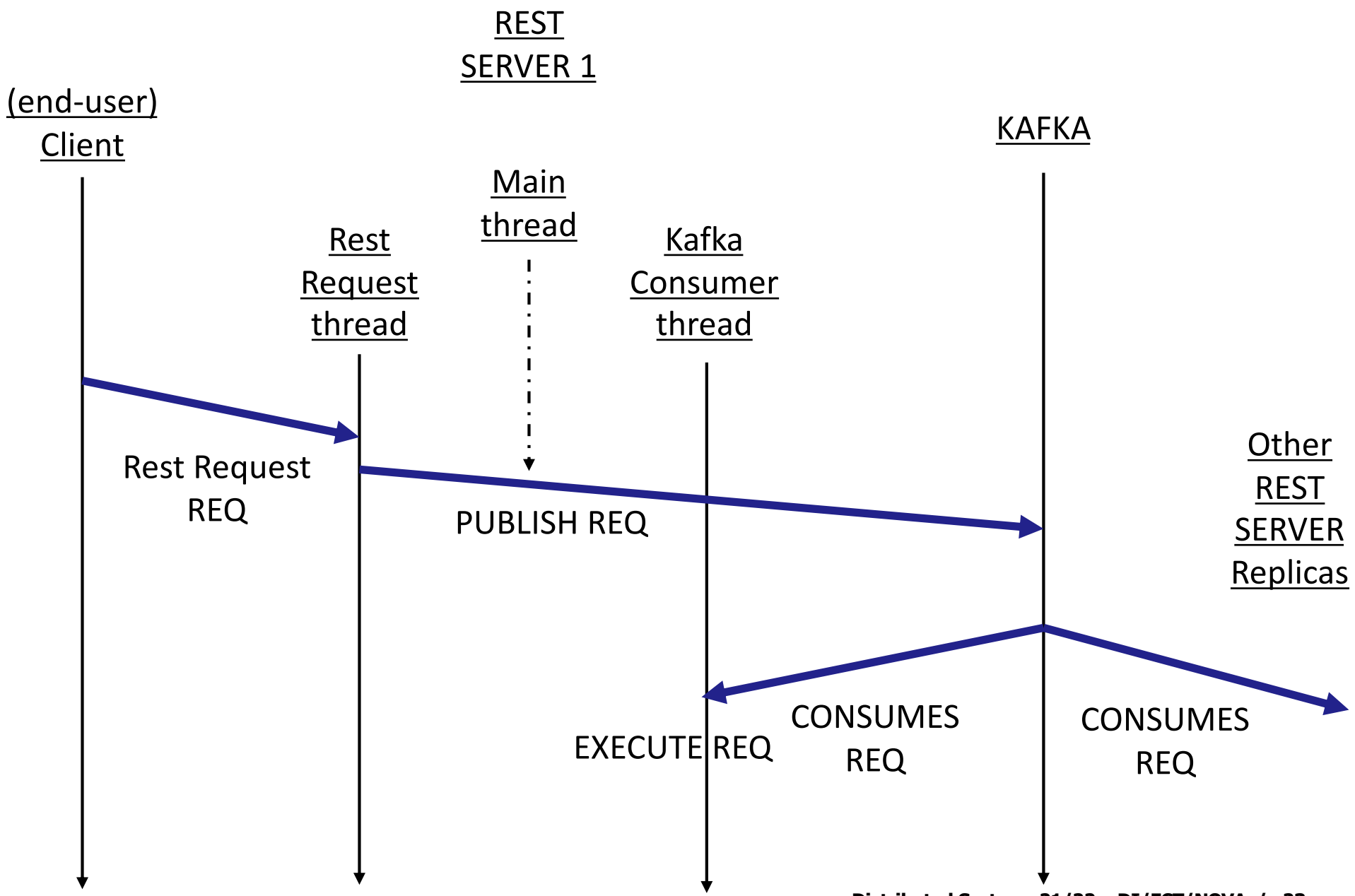
KAFKA: ASYNCHRONY AND REST SERVERS



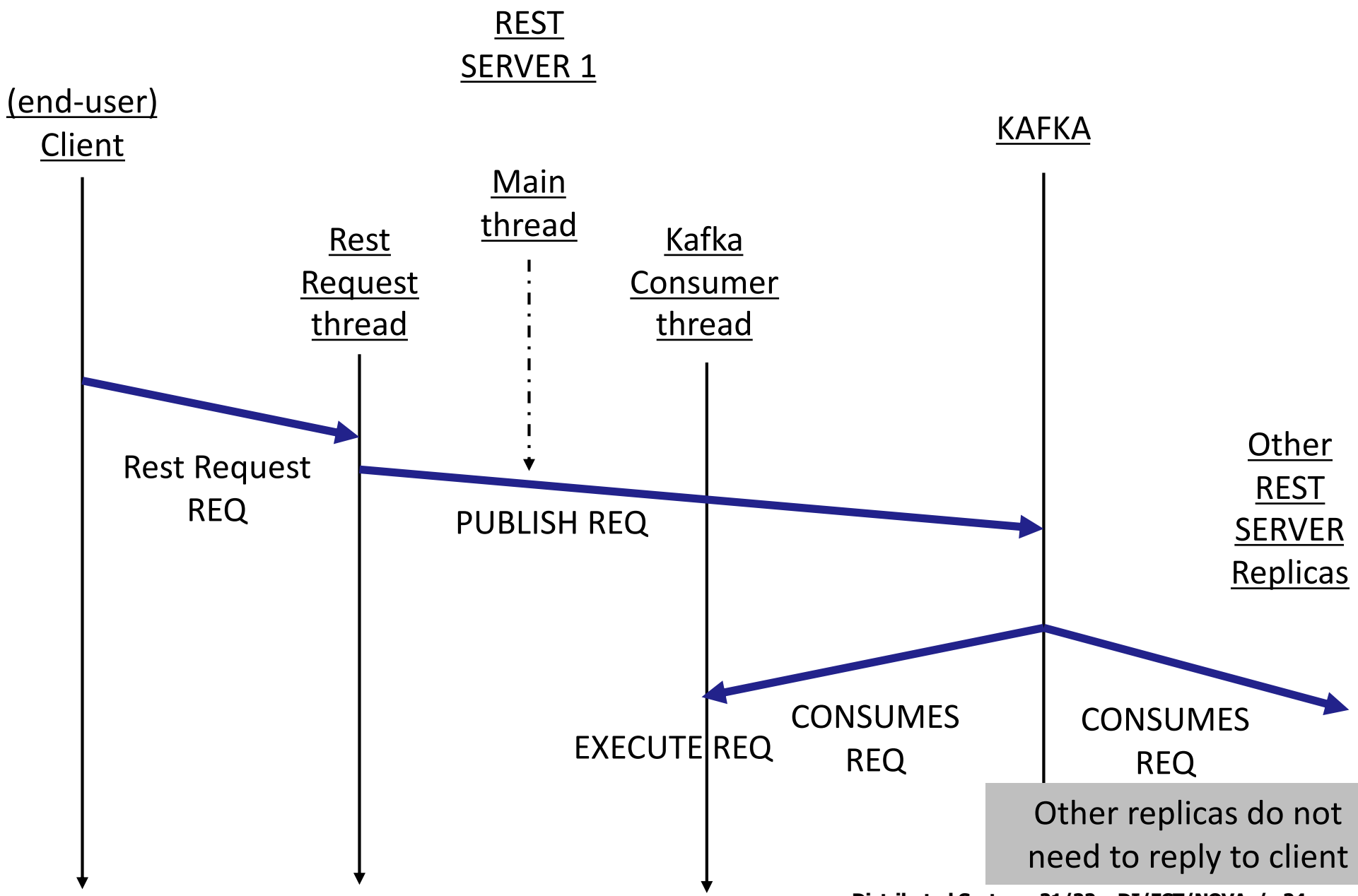
KAFKA: ASYNCHRONY AND REST SERVERS



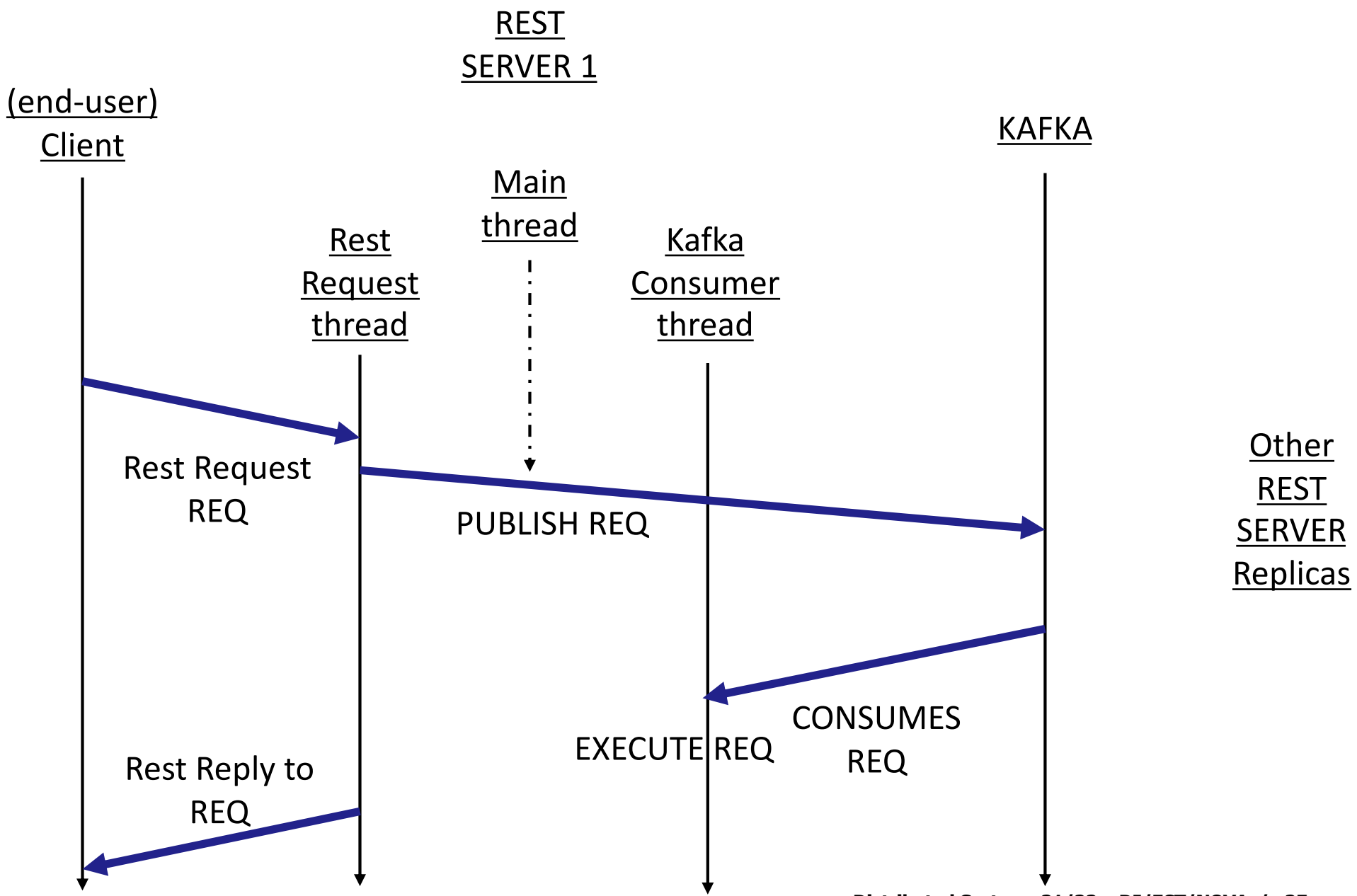
KAFKA: ASYNCHRONY AND REST SERVERS



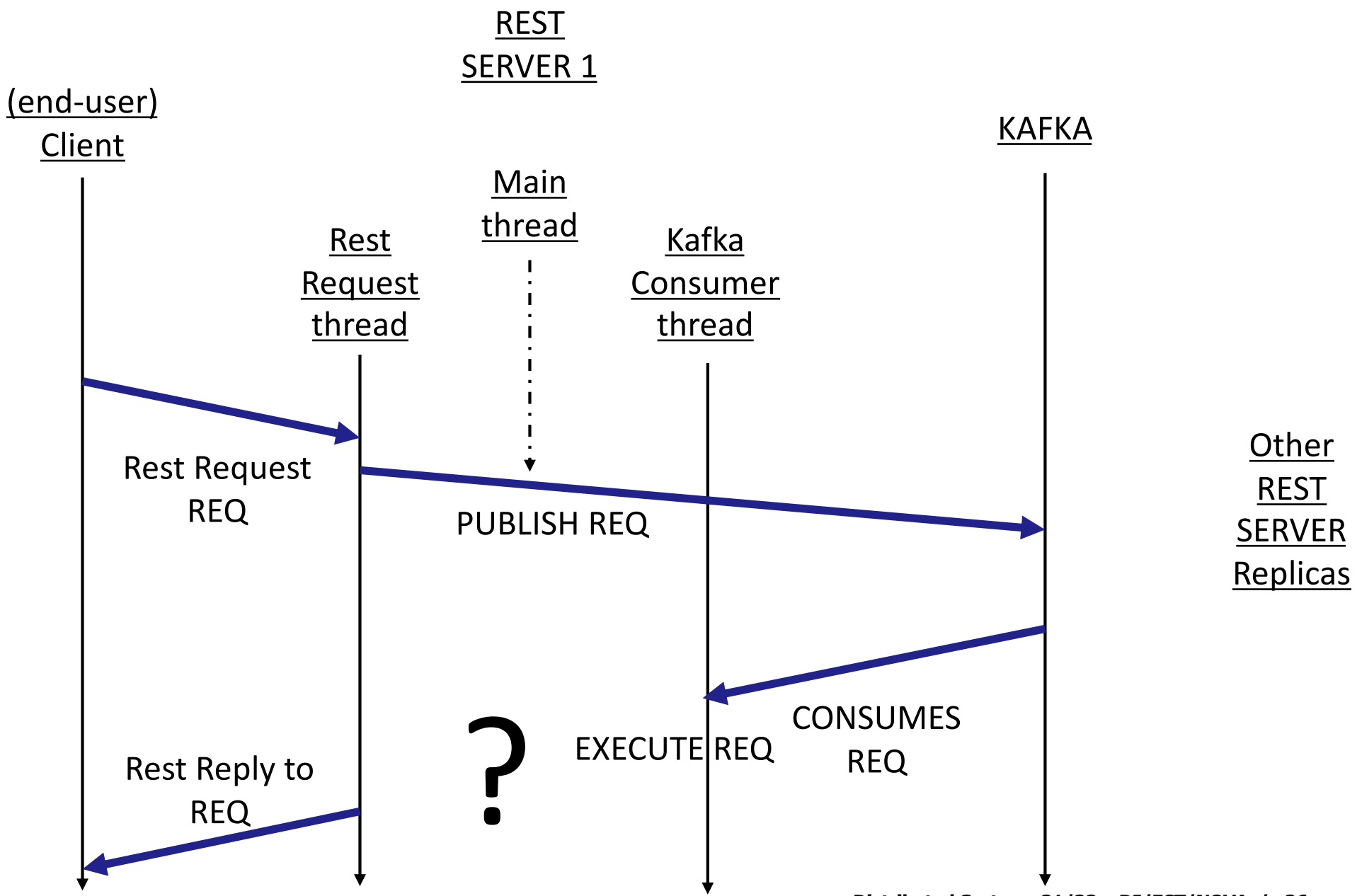
KAFKA: ASYNCHRONY AND REST SERVERS



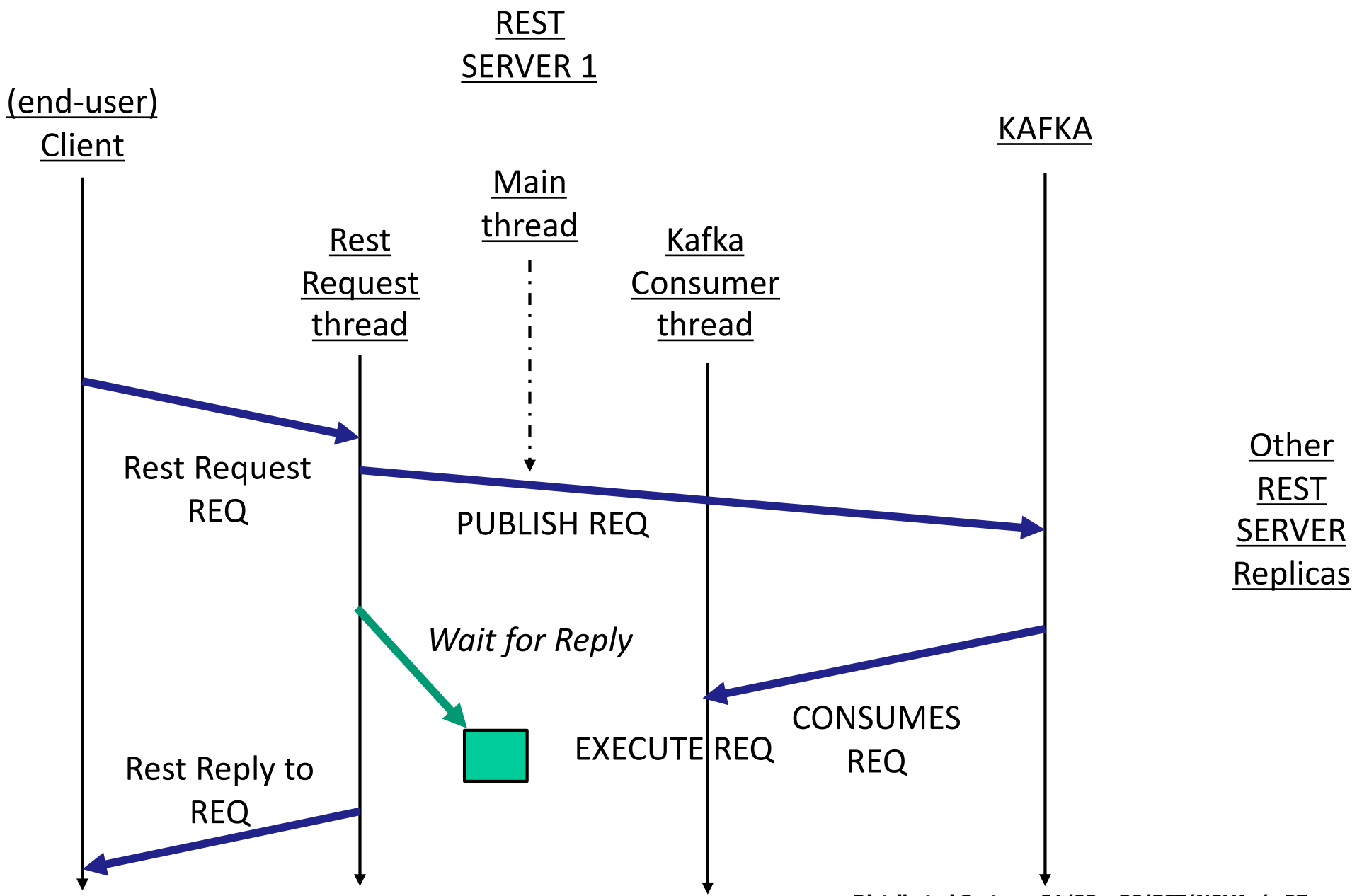
KAFKA: ASYNCHRONY AND REST SERVERS



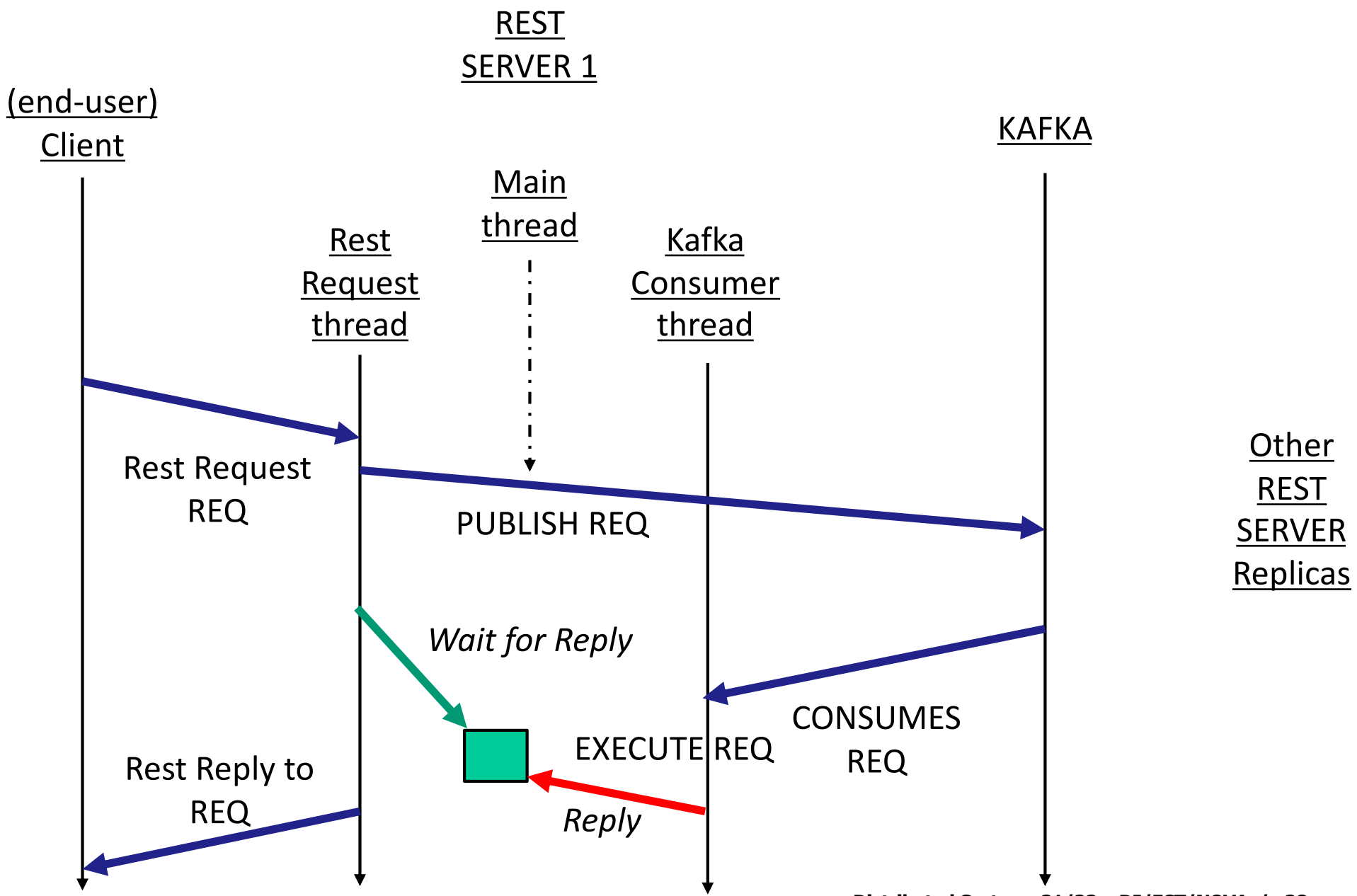
KAFKA: ASYNCHRONY AND REST SERVERS



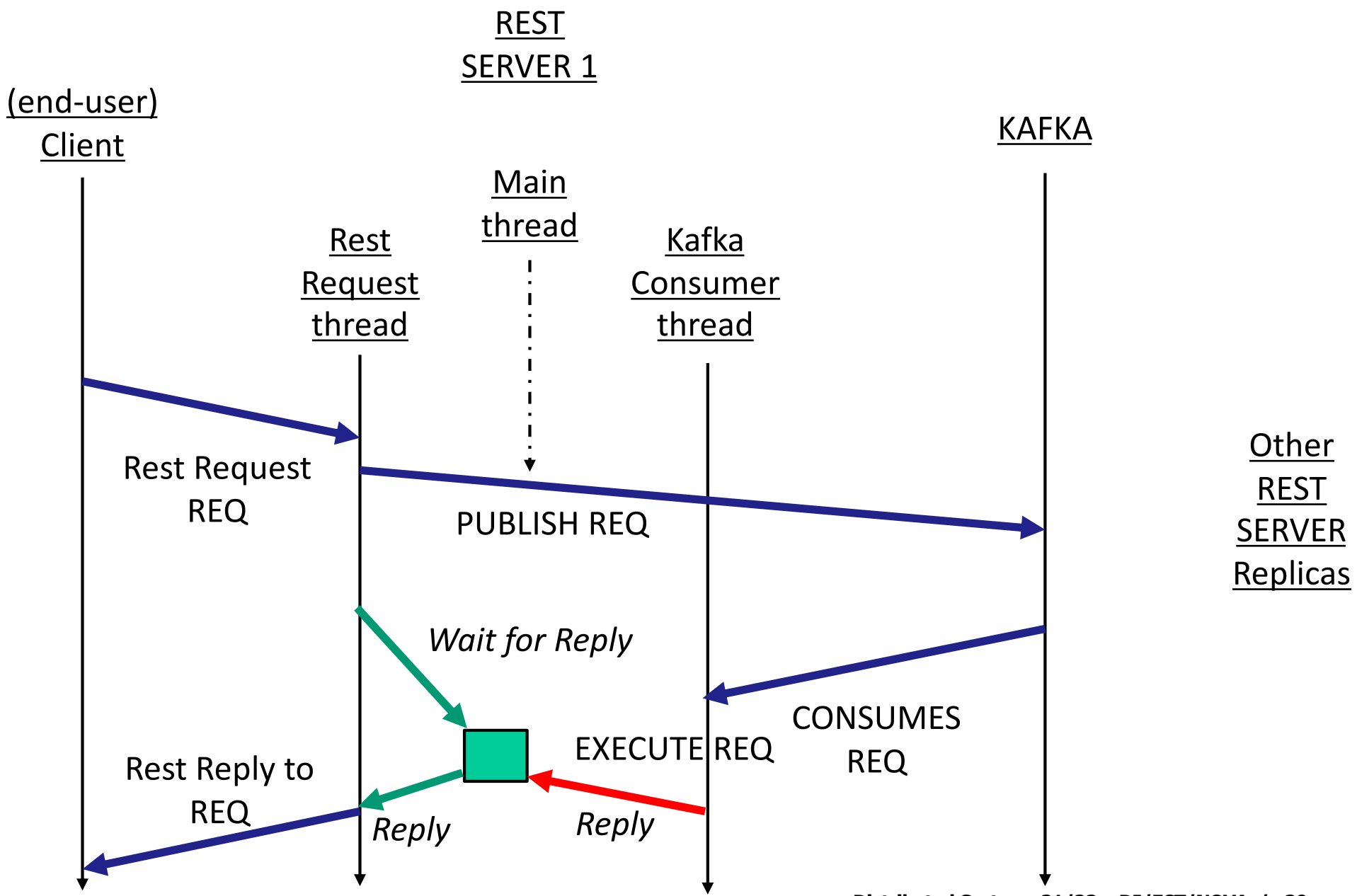
KAFKA: ASYNCHRONY AND REST SERVERS



KAFKA: ASYNCHRONY AND REST SERVERS



KAFKA: ASYNCHRONY AND REST SERVERS



KAFKA: ASYNCHRONY AND REST SERVERS

To simplify this interaction, we provide in the Kafka project that supports this Lab a **SyncPoint class** that can be used by the thread that receives the client request to wait for the reply, and by the Kafka consumer thread to deliver the result of the operation to the waiting thread.

This class also offers mechanisms that allow to ensure that clients do not interact with a replica that are outdated in relation to what the client has previously observed from the system. This will be discussed in more detail on the theoretical lectures.

Like the Discovery, this class will have to be accessible both to the REST Resources class and the Kafka Consumer class (you can instantiate it in the Main of your Server(s), and pass as an argument in the constructor of classes that need access to it)

KAFKA: ASYNCHRONY AND REST SERVERS

Class SyncPoint (relevant methods)

```
/**
 * Assuming that results are added sequentially, returns null if the result is not available.
 */
public synchronized String waitForResult( long n) {
    while( version < n) {
        try {
            wait();
        } catch (InterruptedException e) {
            // do nothing
        }
    }
    return result.remove(n);
}

/**
 * Updates the version and stores the associated result
 */
public synchronized void setResult( long n, String res) {
    if( res != null)
        result.put(n, res);
    version = n;
    notifyAll();
}
```

KAFKA: ASYNCHRONY AND REST SERVERS

Class SyncPoint (relevant methods)

```
/**
 * Assuming that results are added sequentially, returns null if the result is not available.
 */
public synchronized String waitForResult( long n) {
    while( version < n) {
        try {
            wait();
        } catch (InterruptedException e) {
            // do nothing
        }
    }
    return result.remove(n);
}

/**
 * Updates the version and stores the associated result
 */
public synchronized void setResult( long n, String res) {
    if( res != null)
        result.put(n, res);
    version = n;
    notifyAll();
}
```

This method should be used by the thread that received the request from the client to wait for the operation to be ordered and delivered by kafka (and executed by the Consumer) to obtain the reply that should be sent to to the client.

The argument is the sequence number of the record sent to Kafka as shown in the previous example.

KAFKA: ASYNCHRONY AND REST SERVERS

Class SyncPoint (relevant methods)

```
/**
 * Assuming that results are added sequentially,
 */
public synchronized String waitForResult( long n
    while( version < n) {
        try {
            wait();
        } catch (InterruptedException e) {
            // do nothing
        }
    }
    return result.remove(n);
}
```

This method is used by the KafkaConsumer to store the result of an operation received from Kafka (and executed by it).

The long n parameter is the sequence number of the record received from Kafka. The String res should encode the HTTP response code to send back to the client (depending on the outcome of the execution of the operation)

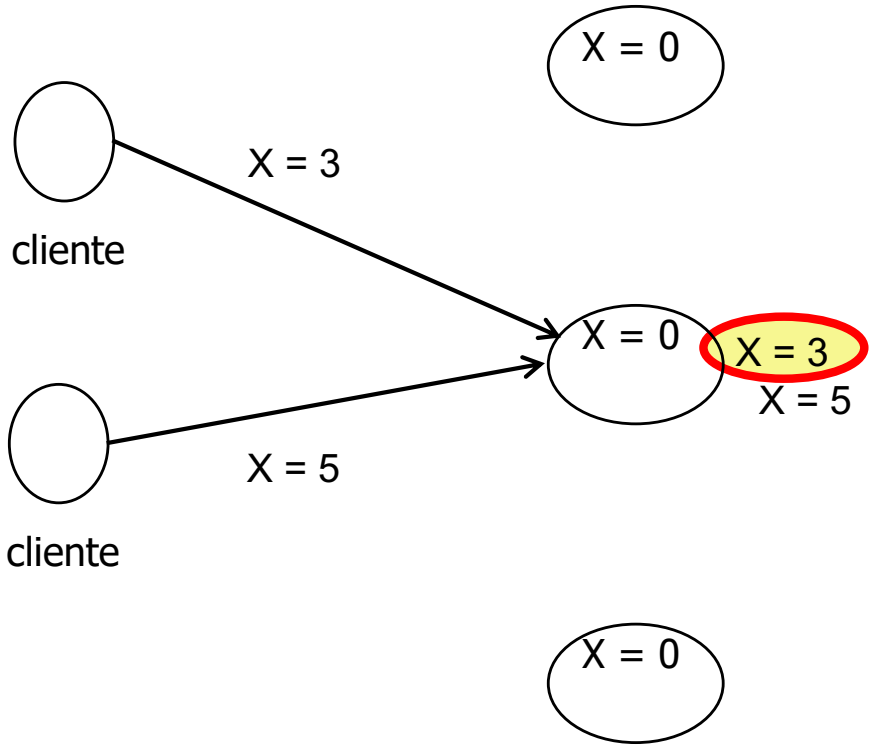
```
/**
 * Updates the version and stores the associated result
 */
public synchronized void setResult( long n, String res) {
    if( res != null)
        result.put(n, res);
    version = n;
    notifyAll();
}
```

TRABALHO PRÁTICO 2 – REPLICAÇÃO PRIMÁRIO/SECUNDÁRIO – SERIALIZAR OPERAÇÕES

Necessário serializar operações.

Escritas enviadas para o primário

Primário **serializa as escritas** e executa-as ordenadamente



Trabalho prático 2 – Replicação primário/ secundário – serializar operações (cont.)

Necessário serializar operações no primário.

Precisamos duma classe que serialize (e armazene) as operações.

1. Definir objeto para representar operação.
2. Definir método que atribua um novo número de sequência à operação – esta parte tem de ser synchronized – e guarde a operação (caso seja necessário enviá-la mais tarde novamente).
3. Envio para os secundários pode ser feito por apenas um thread ou por múltiplos threads – desde que seja enviado o número de sequência da operação, os secundários (e o primário) sabem quando executar.

Trabalho prático 2 – Replicação primário/ secundário – serializar operações (cont.)

No secundário, uma operação recebida do primário só deve ser executada depois de completer a execução de todas as operações anteriores.

O secundário tem também de ir guardando as operações que executa – notem que o secundário pode ter de passar a primário.