Symbian OS

Client-Server Framework

Disclaimer

- **These slides are provided free of charge at http://www.symbianresources.com and are used during Symbian OS courses at the University of Applied Sciences in Hagenberg, Austria (http://www.fh-hagenberg.at/)**
- **Respecting the copyright laws, you are allowed to use them:**
  - **for your own, personal, non-commercial use**
  - **in the academic environment**
- **In all other cases (e.g. for commercial training), please contact andreas.jakl@fh-hagenberg.at**
- The correctness of the contents of these materials cannot be guaranteed. Andreas Jakl is not liable for incorrect information or damage that may arise from using the materials.
- Parts of these materials are based on information from Symbian Press-books published by John Wiley & Sons, Ltd. This document contains copyright materials which are proprietary to Symbian, UIQ, Nokia and SonyEricsson. “S60™” is a trademark of Nokia. “UIQ™” is a trademark of UIQ Technology. Pictures of mobile phones or applications are copyright their respective manufacturers / developers. “Symbian ™”, “Symbian OS ™” and all other Symbian-based marks and logos are trademarks of Symbian Software Limited and are used under license. © Symbian Software Limited 2006.

Contents

- Client-Server Framework
  - Architecture
  - Message Passing
  - Servers and Sessions

Colour coding used in these slides:

Client

Server

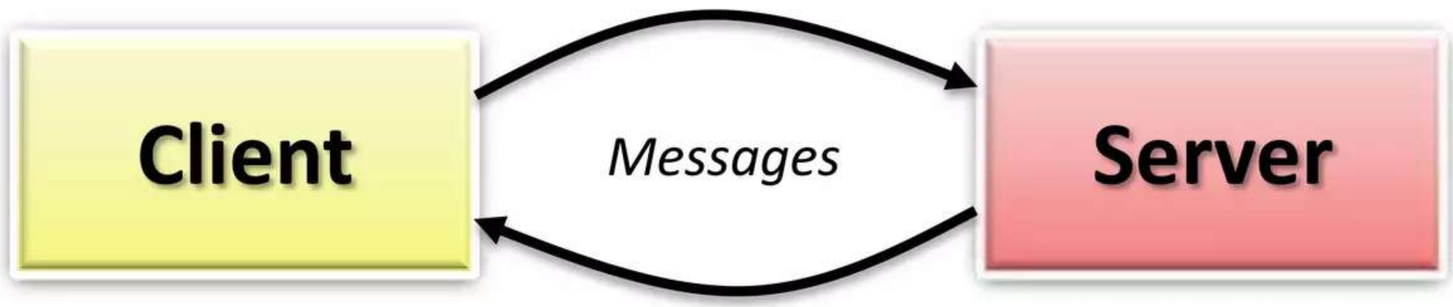
Kernel

Theory of the …

Client-Server Framework

Client-Server Pattern

- **Client** makes use of services provided by a server
- **Server:**
  - Receives request **messages** from its clients
  - Handles requests (synchronously or asynchronously)

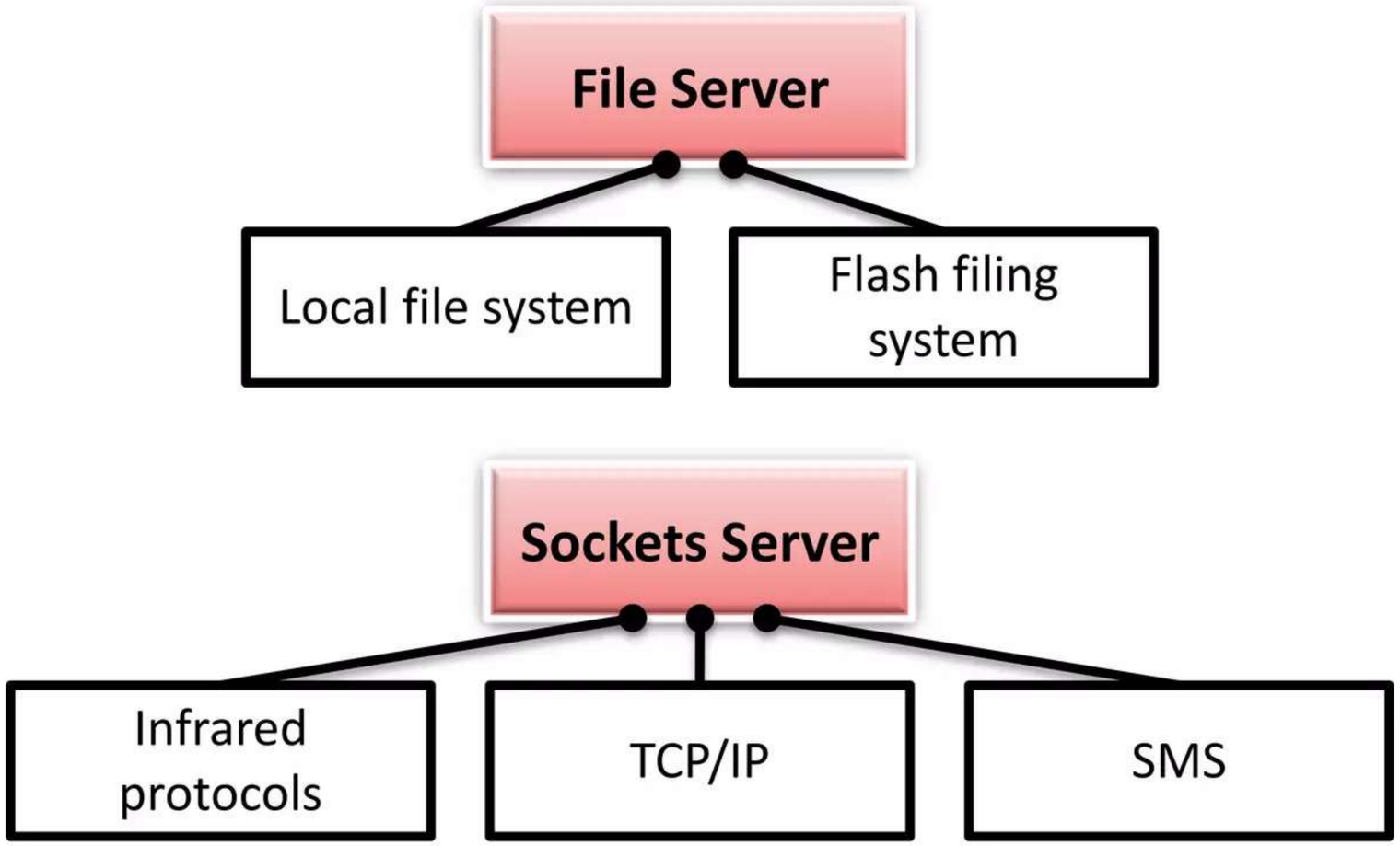


Client-Server Architecture

- Most of the system services in Symbian OS are client-server-based
- **Examples:**
  - Window Server (UI, keypad)
  - File Server
  - Socket Server
  - Telephony Server

Advantages: Plugins

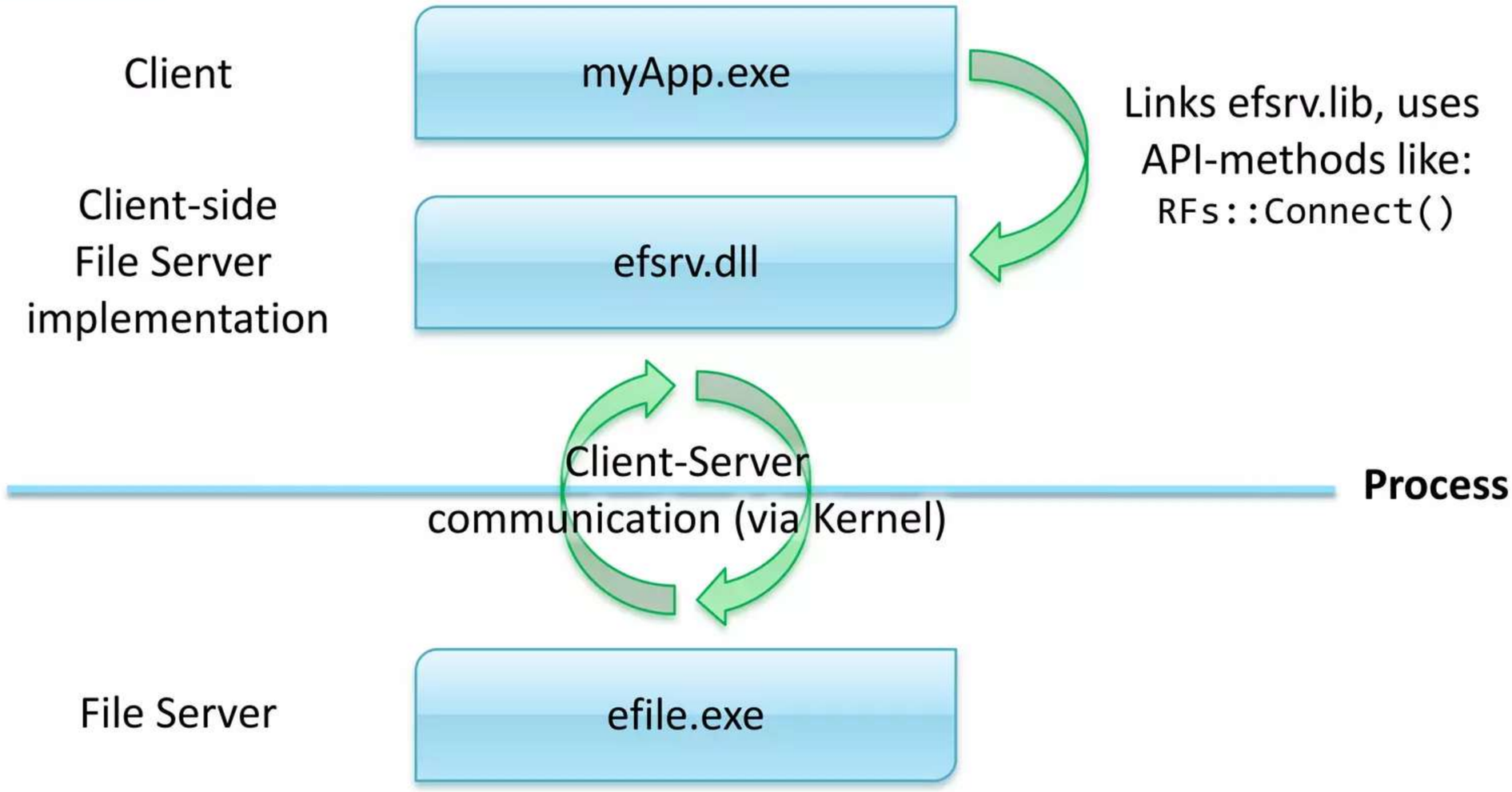
- **Extensibility:** Plug-in modules for globally available new object types (e.g. new multimedia codecs, …)



Advantages

- **Efficiency**
  - Multiple clients use a single server
- **Security**
  - Server/Client in different threads (usually also different processes).
  - Communication through messages → client can not “crash” server
- **Asynchronous**
  - Server uses Active Objects for return messages. No polling needed → good for our battery!

Structure – Example



R-type Classes

- For client-side implementation
- **R**: Owns external **R**esource handle (e.g. handle to a server session)
- **Resource allocation:** usually Open(), Create() or Initialize()
- **Deallocation:** usually Close() or Reset()
  - The destructor doesn't do it – most of the times, there is no destructor
  - App. should release the resource as soon as possible
- Can be instantiated on the stack (automatic variable) or on the heap (instance variable)

Example code

```
// Establish connection to file-server
RFs fs;
User::LeaveIfError(fs.Connect());
CleanupClosePushL(fs);

// Submit a request to the server
_LIT(KIniFile, „c:\\myini.ini“);
User::LeaveIfError(fs.Delete(KIniFile));

// Create a subsession
RFile file;
User::LeaveIfError(file.Create(fs, KIniFile, EFileRead | EFileWrite |EFileShareExclusive));
CleanupClosePushL(file);

// Submit a request using the subsession
TBuf8<32> buf;
User::LeaveIfError(file.Read(buf));

// Cleanup
CleanupStack::PopAndDestroy(2, &fs);        // file, fs
```

When deleting: close connection to the server (call Close())

Wrapper for RSubSessionBase::SendReceive()

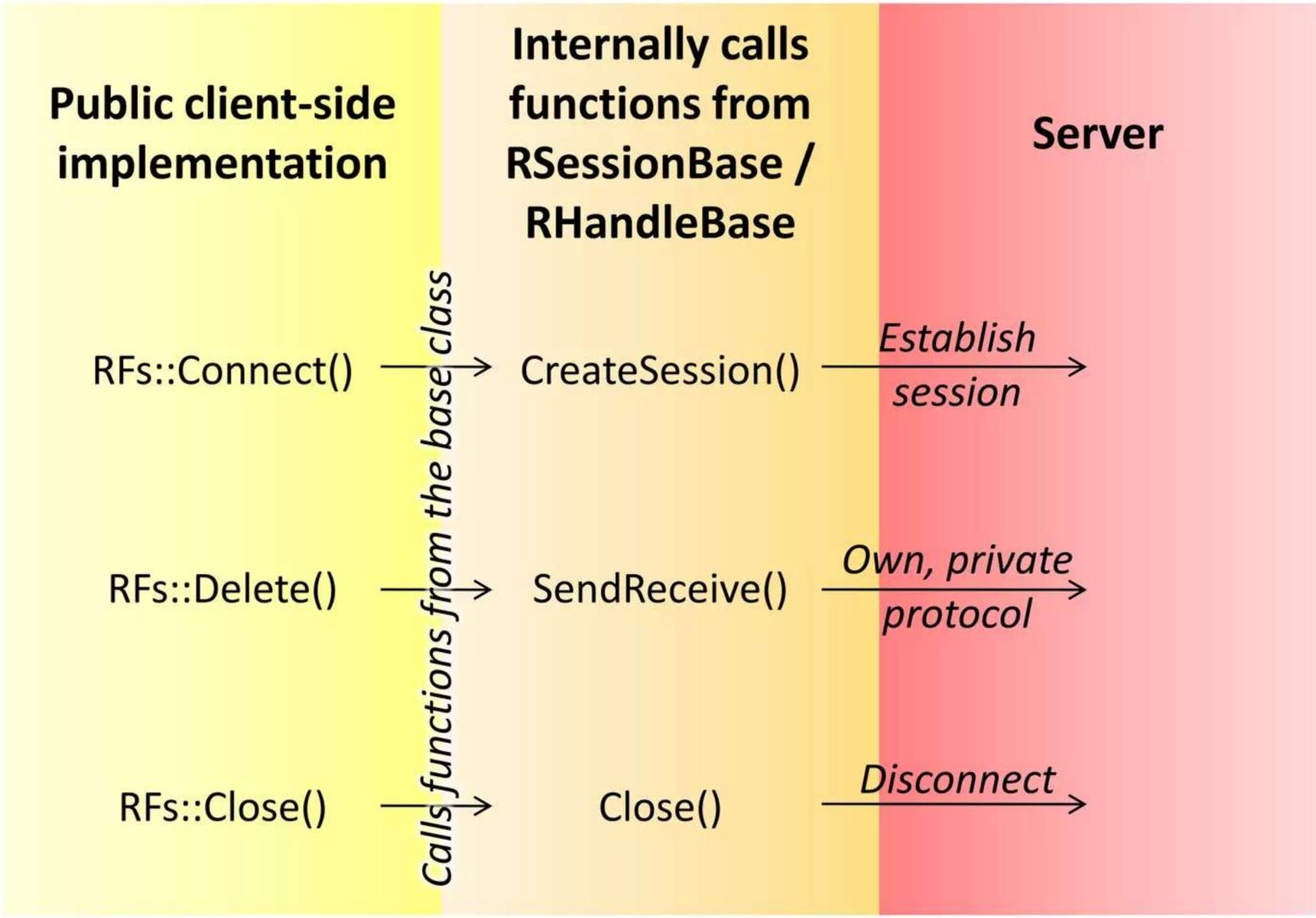
Session

- **Session = Communication channel between client and server**
  - Initiated by a client
  - Kernel creates server-side representation
  - Messages between client / server mediated by kernel
- Only **one synchronous** request may be submitted at a time
- ... but **multiple asynchronous** requests may be outstanding

Client-side Representation

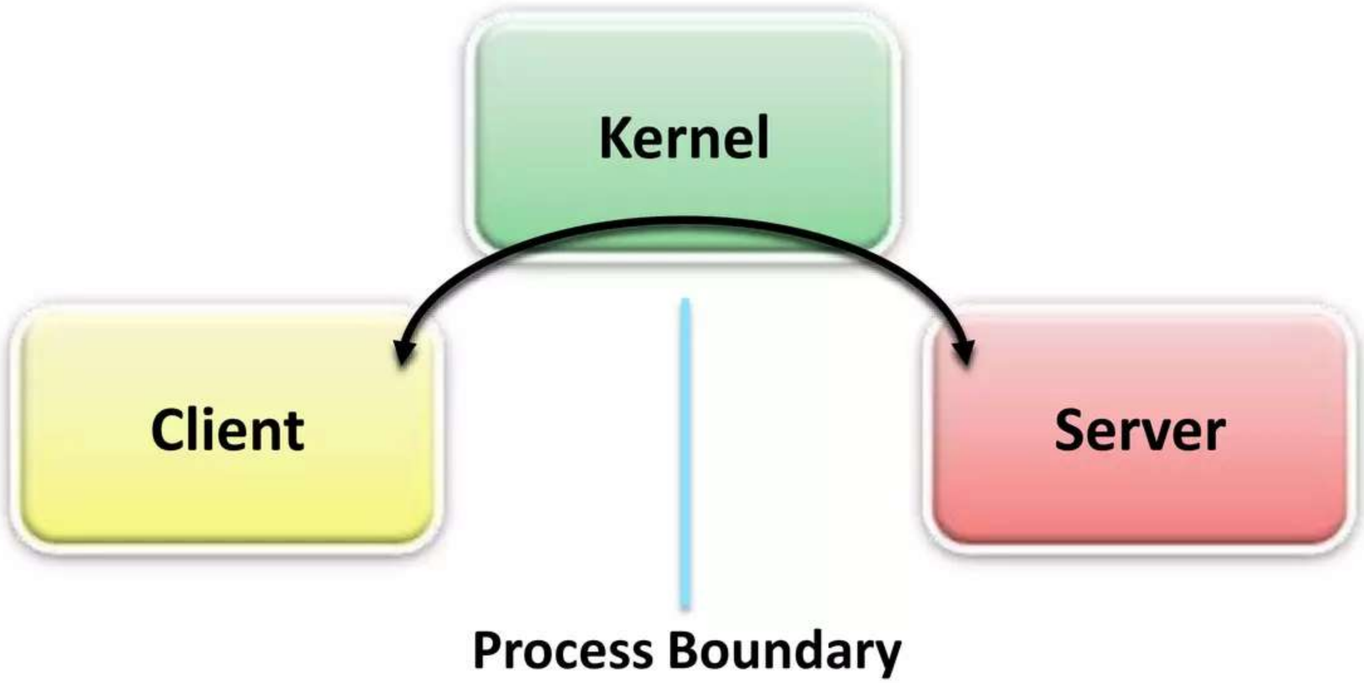
- Server is usually accessible through a **public client class**
  - Handles details of establishing a session with its server
  - Sends commands to the server
  - Handles receiving responses
  - Derives from **RSessionBase**
- → **Hides implementation details** of the private client-server communication protocol
- **Example:**
  - Simple functions RFs::Delete() or RFs::GetDir() communicate with the file server and hide the more complex client-server communication protocol

Communication



Communication

- Send data between processes:
  - **Message Passing:** Used in all cases
  - **Inter-process communication (IPC):** For large data packets, if needed



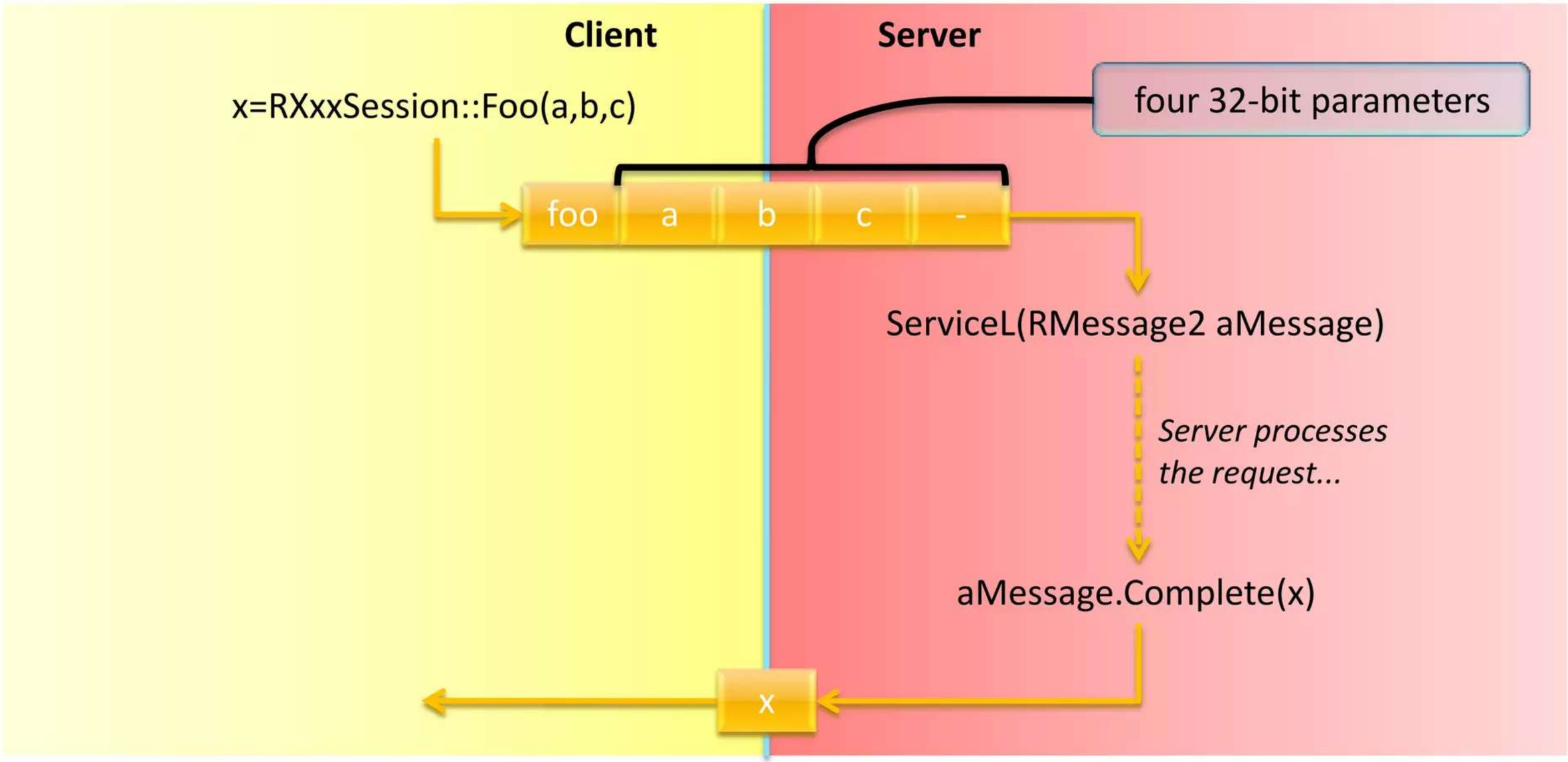
Messages [1]

- **Clients send messages to servers**
- Each message is a **request for a service**
  - Server processes request
  - Performs work on behalf of the client
  - Optionally sends a reply to the client
- Messages can be sent:
  - synchronously, or
  - asynchronously

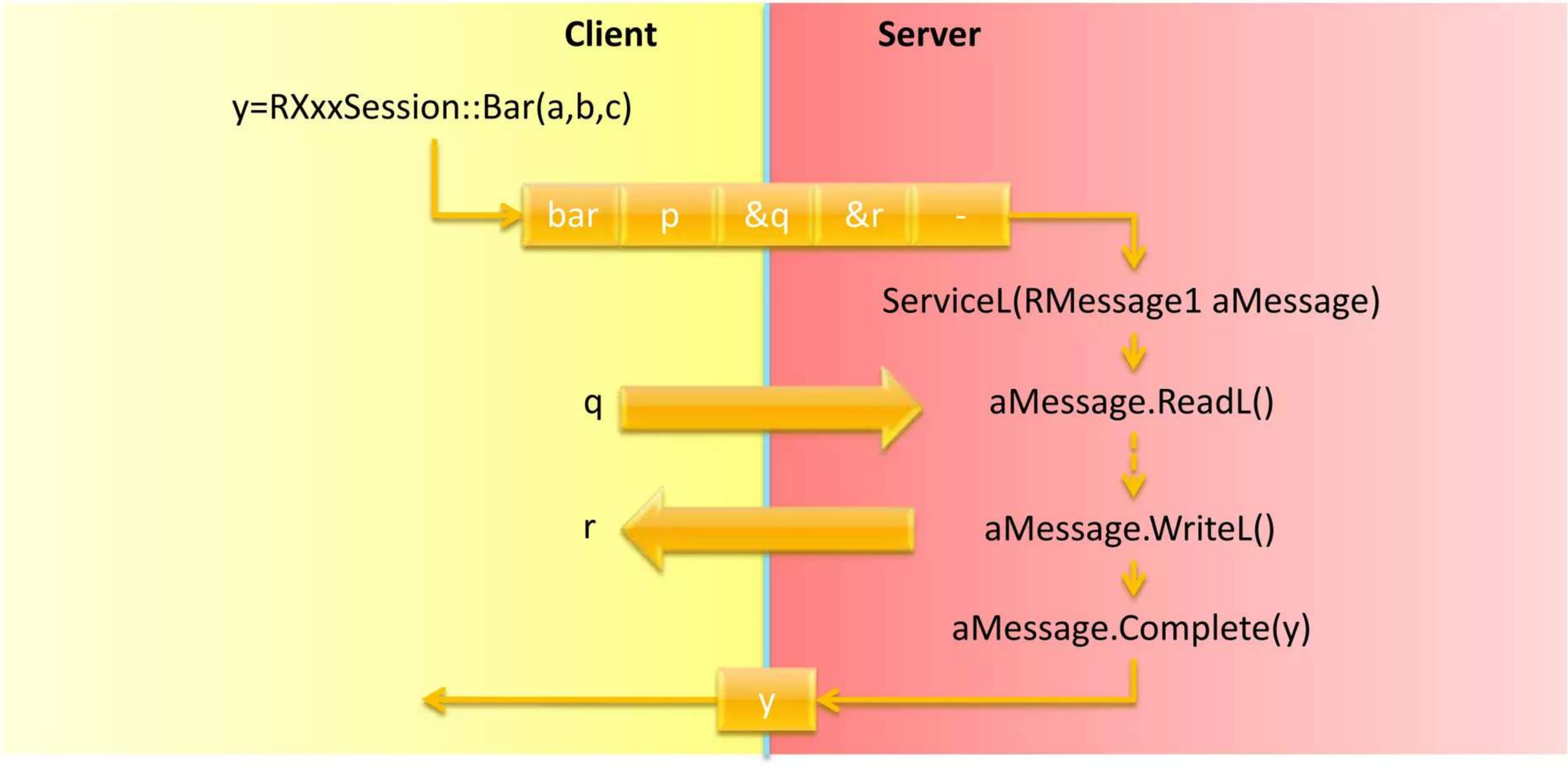
Messages [2]

- **Request message**: Consists of five 32-bit numbers
  - **Function number**
  - **Four 32-bit parameters**
- Each of the four parameters can be:
  - An integer (TInt)
  - A pointer to a flat data structure, wrapped as a Symbian OS descriptor ($\rightarrow$ read using IPC)

Message Passing



Inter-Process Communication



Request-Handling

"2" – Symbian OS 9: Capability-Support

Client

RSessionBase
SendReceive()
Functions for the client, converts requests to parameters

Server

CSession2
ServiceL()
Handling of requests, represents session server-side

CServer2 (Active Obj)
RunL()
Receives requests, forward to corresponding session

RMessage2
Complete()
For client/server data transfer

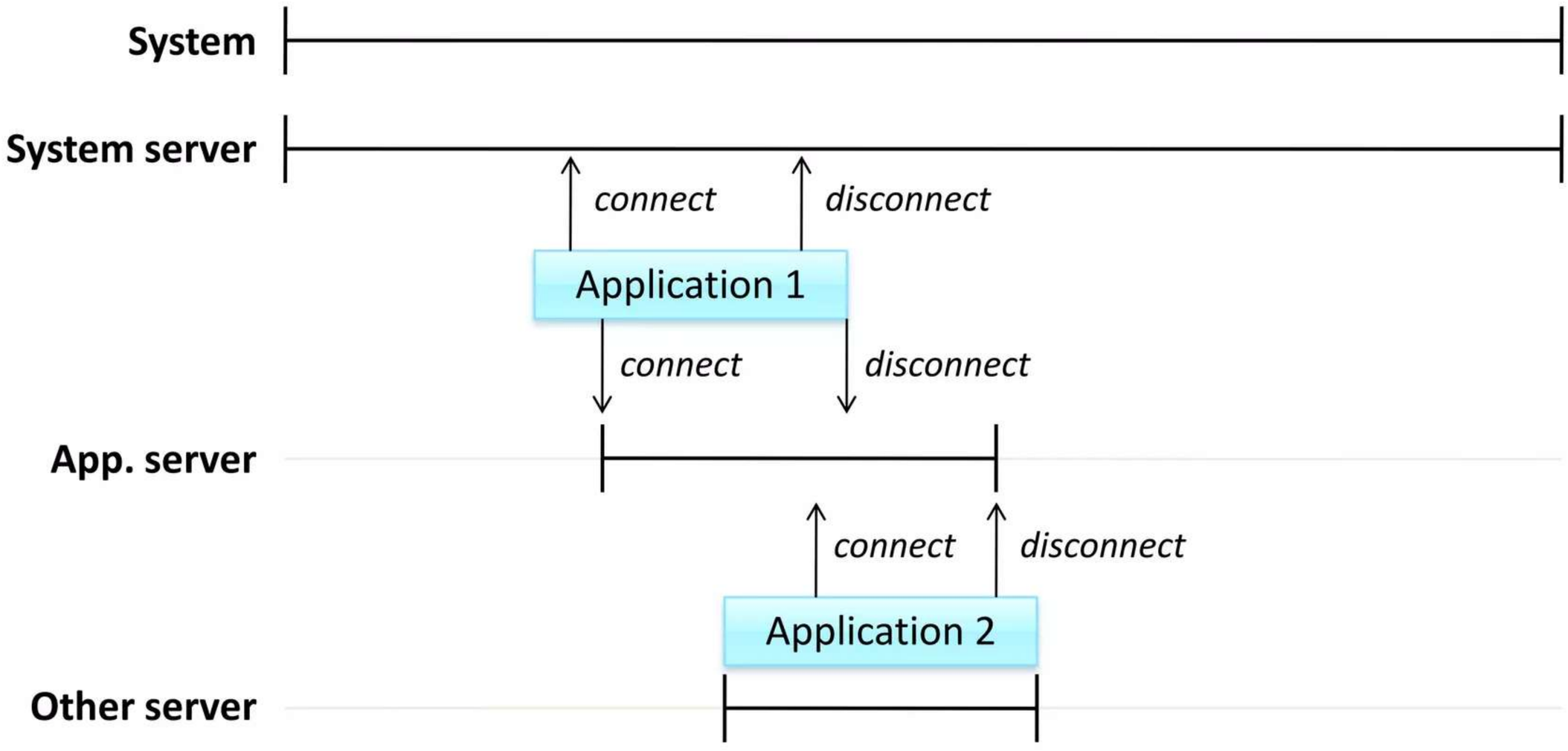
DSession
Links Client/Server, Cleanup

Kernel

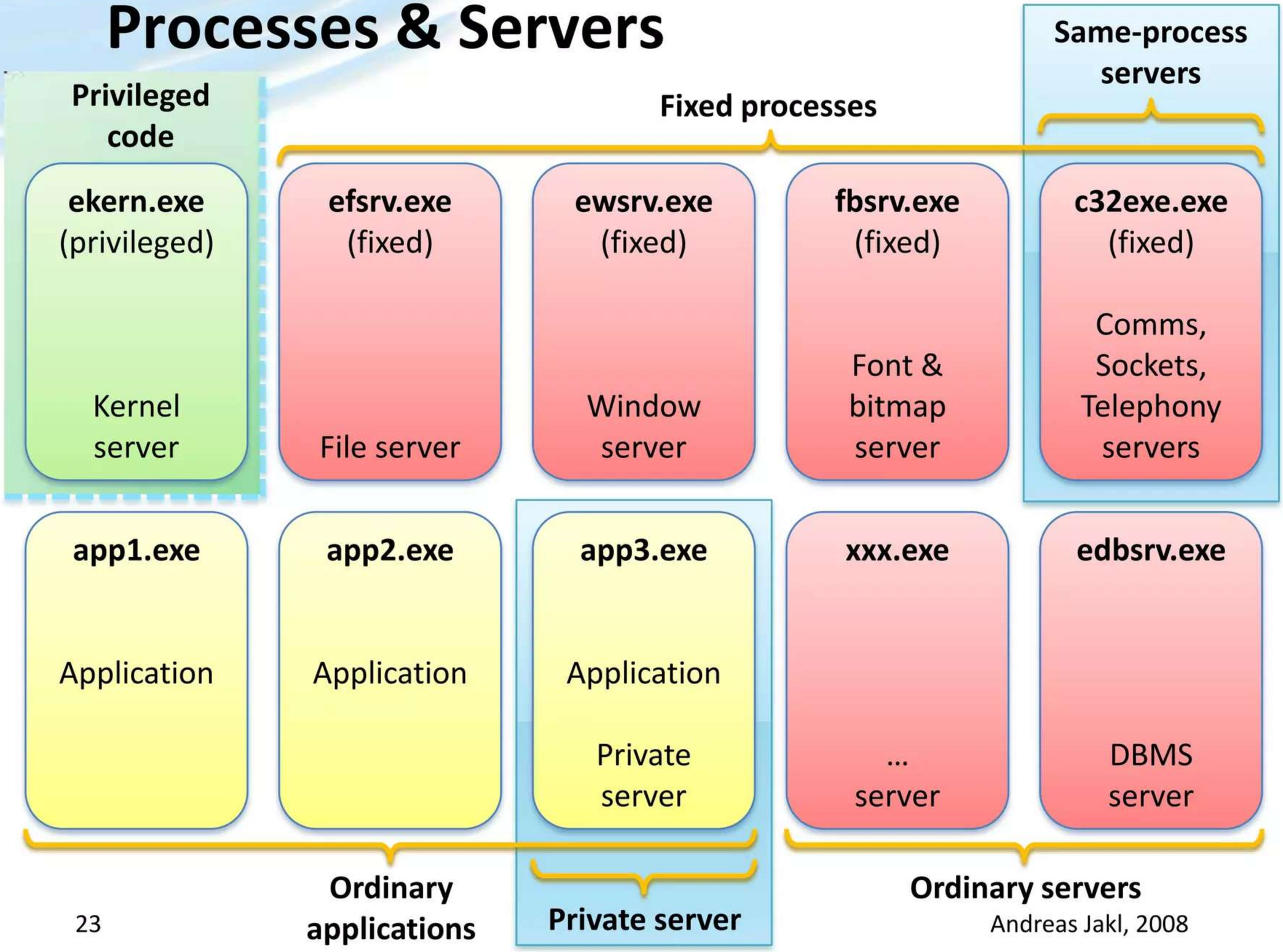
Server Lifetime

- Three options:
  1. **System Server:** Started with the mobile phone, essential for the OS (e.g. File Server, Window Server). Unexpected termination usually forces phone reboot.
  2. **Application / Transient Server:** Started on request of an application, closed when last client disconnects. If a second client starts a new server, no error is produced, existing server instance is used.
  3. **Other Servers:** Started and closed with an individual application. If used by multiple applications: each app. has its own private instance; e.g. POSIX server.

Server Lifetime

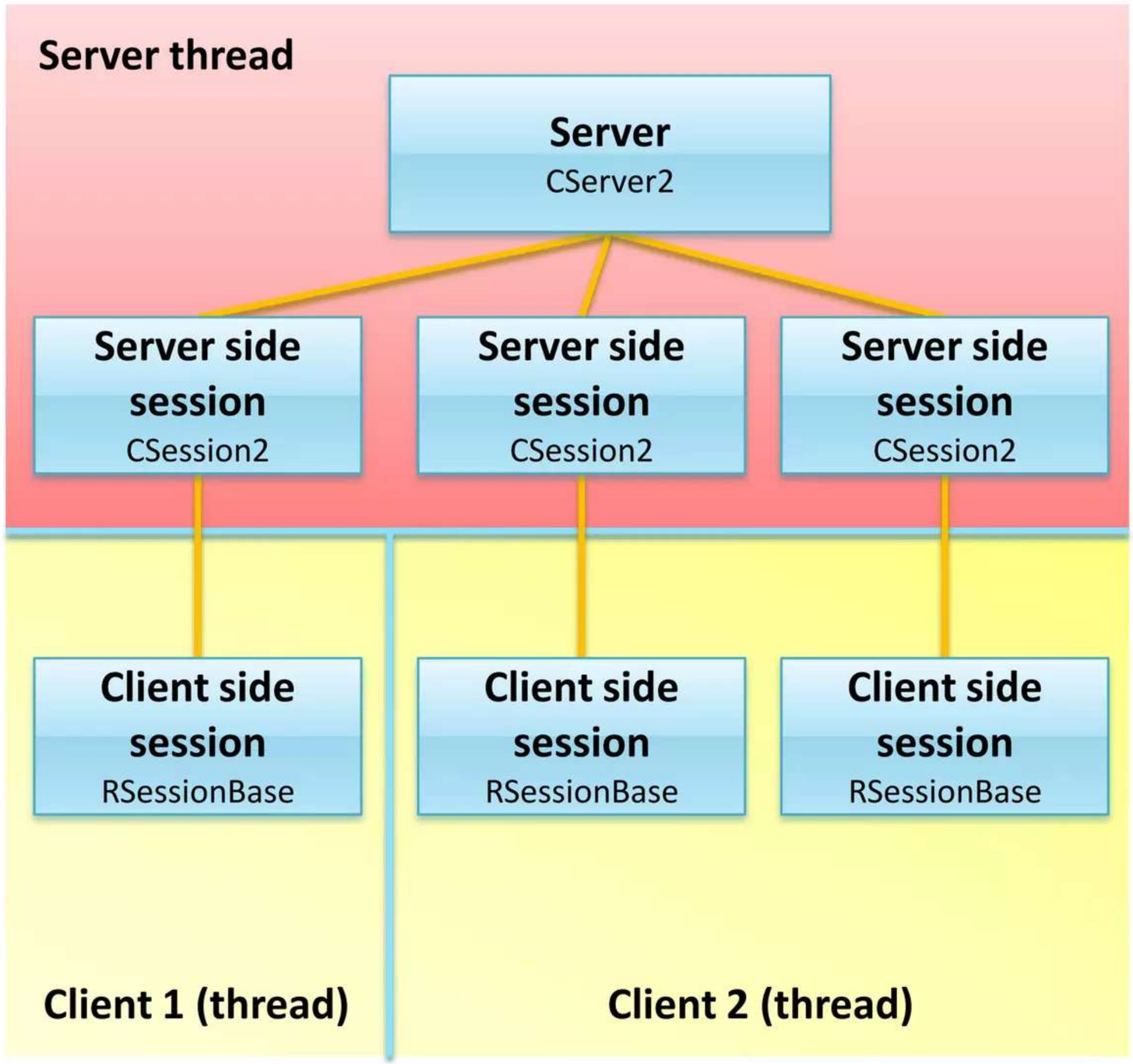


Processes & Servers



Sessions

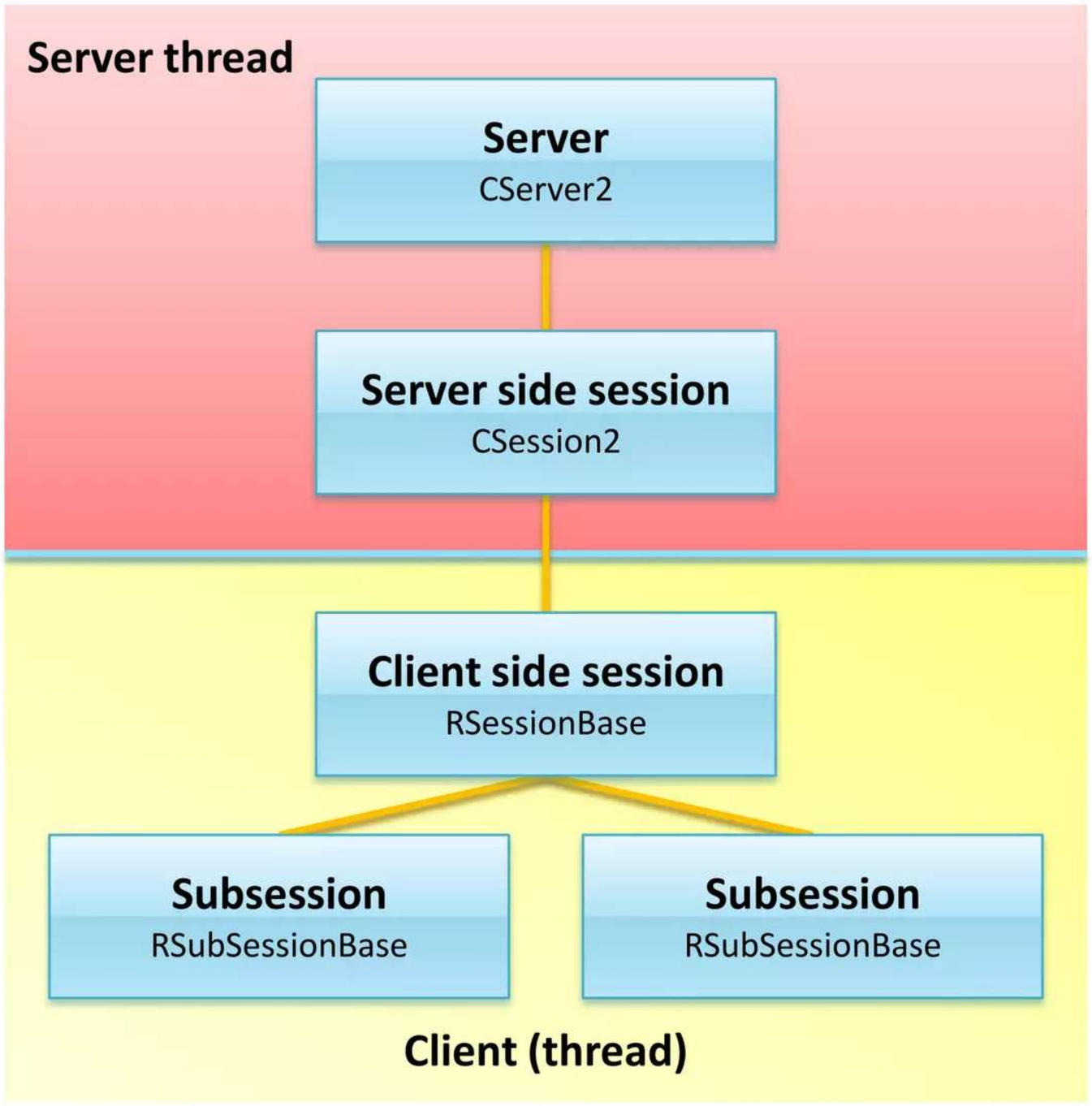
- Server can serve multiple clients
- A client can have multiple sessions with a server
- Each session requires resources



Subsessions

- **Solution:** SubSessions
- **e.g. File Server:** One Session per Client (RFs), multiple files represent SubSessions (`RFile`)

→ no new session for every single file!



API Example

```
class RFs : public RSessionBase {...}
class RFsBase : public RSubSessionBase {...}
class RFile : public RFsBase
{
public:
   IMPORT_C TInt Open(RFs& aFs, const TDesC& aName, TUint aFileMode);
   IMPORT_C TInt Read(TDes8& aDes) const;
   IMPORT_C void Read(TDes8& aDes, TRequestStatus& aStatus) const;
   IMPORT_C TInt Write(const TDesC8& aDes);
   IMPORT_C void Write(const TDesC8& aDes, TRequestStatus& aStatus);
   [...]
};
```

Example code (Revisited)

```
// Establish connection to file-server
RFs fs;
User::LeaveIfError(fs.Connect());
CleanupClosePushL(fs);

// Submit a request to the server
_LIT(KIniFile, „c:\\myini.ini“);
User::LeaveIfError(fs.Delete(KIniFile));

// Create a subsession
RFile file;
User::LeaveIfError(file.Create(fs, KIniFile, EFileRead | EFileWrite |EFileShareExclusive));
CleanupClosePushL(file);

// Submit a request using the subsession
TBuf8<32> buf;
User::LeaveIfError(file.Read(buf));

// Cleanup
CleanupStack::PopAndDestroy(2, &fs);        // file, fs
```

When deleting: close connection to the server (call Close())

Wrapper for RSubSessionBase::SendReceive()

Performance

- **Communication:**
  - **Overhead** through thread / process **context switches**
  - Inter-process data transfer requires mapping client data area to server's virtual address space
- **Best practice:**
  - **Transfer a large amount of data** in a single transaction instead of performing a number of server accesses
  - **Example:** Most Symbian OS components don't use RFile::Read() / Write() directly, but instead use **buffered streams** (RReadStream() / RWriteStream())

Is a Server Required?

- Need to share functionality or resources across a number of threads or processes?
- Require isolation between service provider and service user?
- Pre-emptive asynchronous processing desirable?
- Gains outweigh costs?

ASD-like Question – Easy

- **Which of the following statements correctly describe the client-server model in Symbian OS?**
  - **A.** A client and server always execute in separate threads.
  - **B.** A client and server always execute in separate processes.
  - **C.** A client initiates a server request by passing an integer to identify the request. It then sends any “payload” data separately.
  - **D.** To return data to a client, the server writes directly into the client’s address space.
  - **E.** At any time, there can only be one outstanding synchronous client request to a server.

Solution

- **A.** Correct.
- **B.** Incorrect. Most servers are executed in an own process, but it is not required.
- **C.** Incorrect. The data is sent together with the function request as a parameter.
- **D.** Incorrect. The return values are sent using a message object.
- **E.** Correct.

ASD-like Question – Medium

- **For which of the following scenarios would a developer choose to implement a client and server on Symbian OS?**
  - **A.** To provide utility code, such as a 3D graphics library.
  - **B.** To implement a system-wide “broadcast” notification system.
  - **C.** To manage access to a hardware resource, such as the camera.
  - **D.** To implement a Bluetooth game where the host player runs as a server and all other players are clients.
  - **E.** To implement a user interface application for displaying weather information.

Solution

- **A.** Incorrect. The performance overhead would be too high.
- **B.** Correct.
- **C.** Correct.
- **D.** Incorrect. This has nothing to do with the client/server communication within Symbian OS.
- **E.** Incorrect. UI and application logic should be separated, but rather by using DLLs instead of implementing client/server communication.

That’s it!

Thanks for your attention