Infineon

Flash File System

Presentation FFS

- Presented by Erik Christensen, SW Designer, IFWD

never stop thinking

Infineon technologies

Revision: 1.0

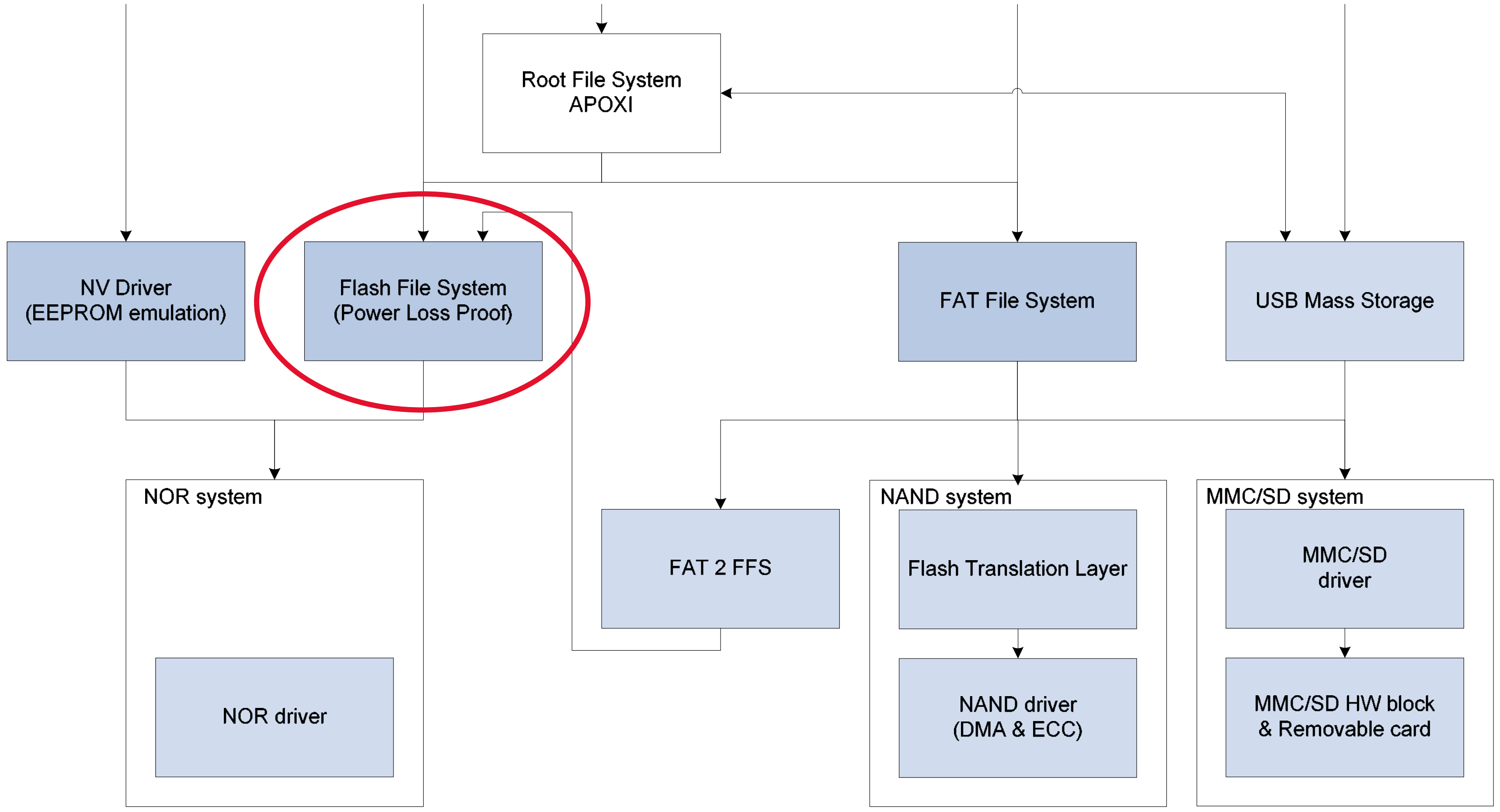
Confidential

Agenda

- Overview
- Features
- Architecture
- API
- Questions

Overview

- The Flash File System is a part of the NOR system in the overall storage system.



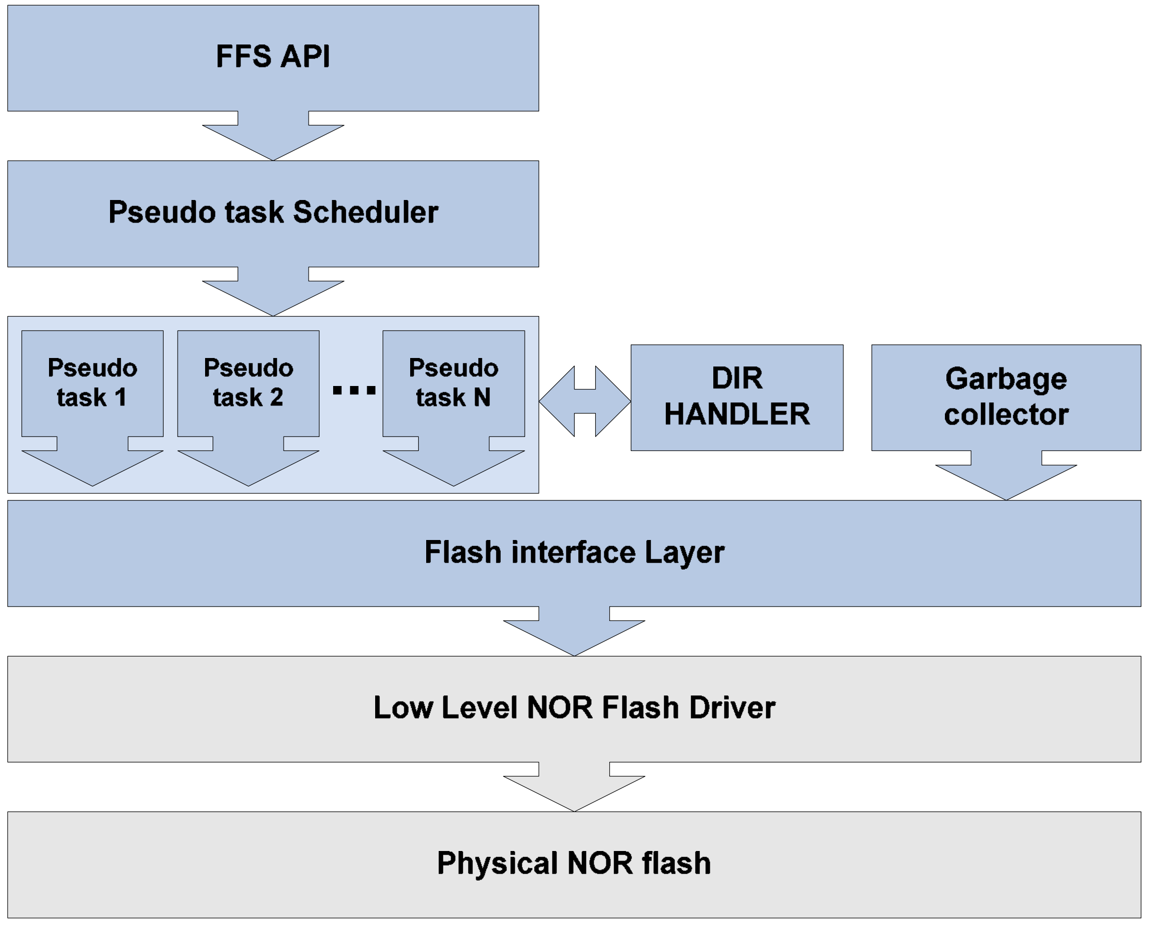
Features

- Proprietary file-system.
- Power Loss Proof.
- Redundancy during file-updates prevents corrupted files due to power loss.
- Re-entrant. Multiple prioritized tasks with FFS-access simultaneously.
- Unlimited number of files opened for read/write simultaneously.
- Unique file identification using filename or file-Id.
- Directory support.
- Dedicated memory pools to reserve space for specific file types.
- API: Sync. and async. interface + streaming option.
- Dynamic and static(read-only) partitions. Linked files for factory restore.
- Wear leveling (dynamic + static)

Features - limitations

Max. file size:	4 Gbytes.
Max. size FFS storage:	4 Gbytes.
Max. number of files:	0-65536 (configurable).
Max. number of filetypes:	0-256 (configurable).
Max. Filename length:	Configurable (currently 48 chars)
Max. Path length:	Configurable (currently 128 chars)

Architecture (main components)



Architecture – main components

FFS API

The operation requests are received through the FFS API.

The requests are validated, and all valid requests are past on to the pseudo task scheduler.

Pseudo task scheduler

The component is scheduling the operation-requests based on the priority of the FFS-users. Operations requested by high priority user will cause low priority operation to be put hold.

Pseudo tasks

Each pseudo task holds the state of one individual file-operation.

Architecture – main components

Garbage collector

The task for the Garbage Collector is to clean up and reclaim obsolete parts of the flash used for the FFS.

Whenever a file is deleted, the Garbage Collector has to clean up (erase) the area used by the file, and make this area available for new data.

Flash Interface Layer

In this layer high level adaptions for the actual used flash device are made (e.g. program buffer size and bank switching).

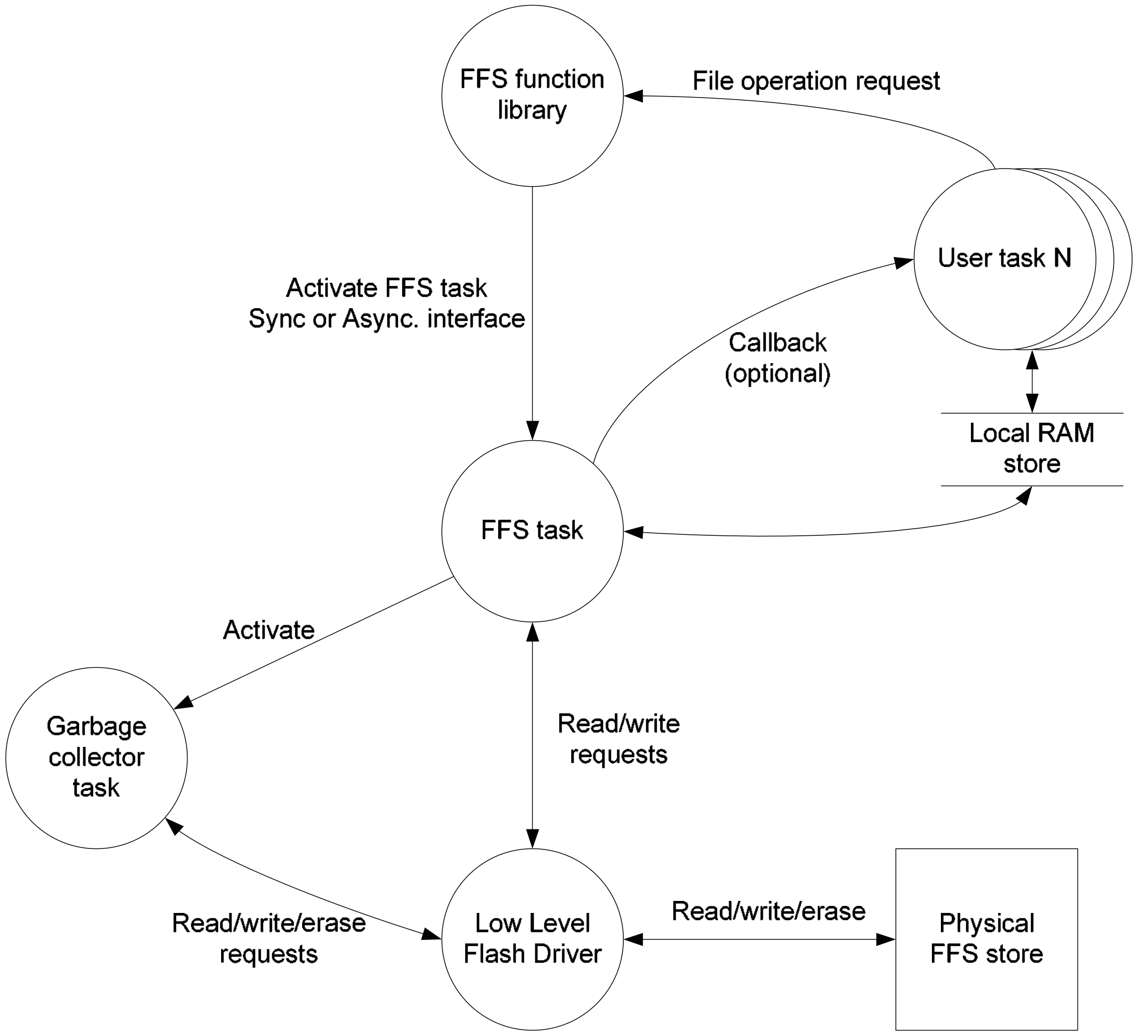
Low Level Flash Driver

This driver is performing the physical updates in the flash media.

Flash program and erase commands is always carried out from here.

This driver is normally located in RAM.

Architecture – flow diagram



Forced garbage collection

Normally garbage collection is performed from the garbage collector, as a background process running on low RTOS priority.

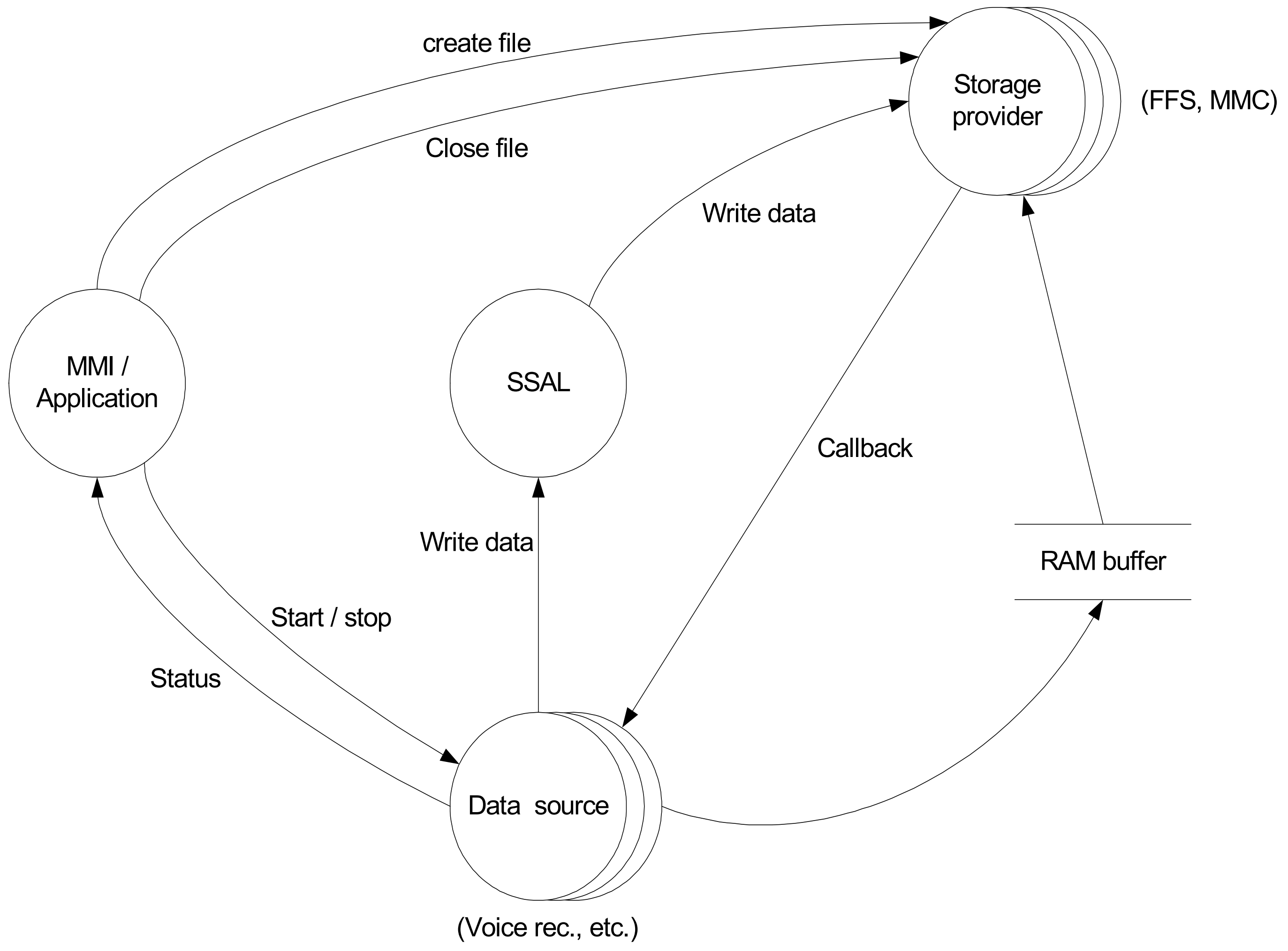
In special scenarios with high CPU load and massive FFS activity, it might be that the garbage collector task is not scheduled by the RTOS. If garbage generating operations are performed in this situation, garbage will accumulate in the file system.

If the file system gets full due to uncleaned garbage in the system, this could block file-operations which need to allocate data in the system.

To overcome this problem, forced garbage collection can started from FFS task level. Forced GC will be activated if garbage needs to be cleaned to be able to carry out an requested file-operation.

Only the memory needed to carry out requested operations are cleaned by forced garbage collection.

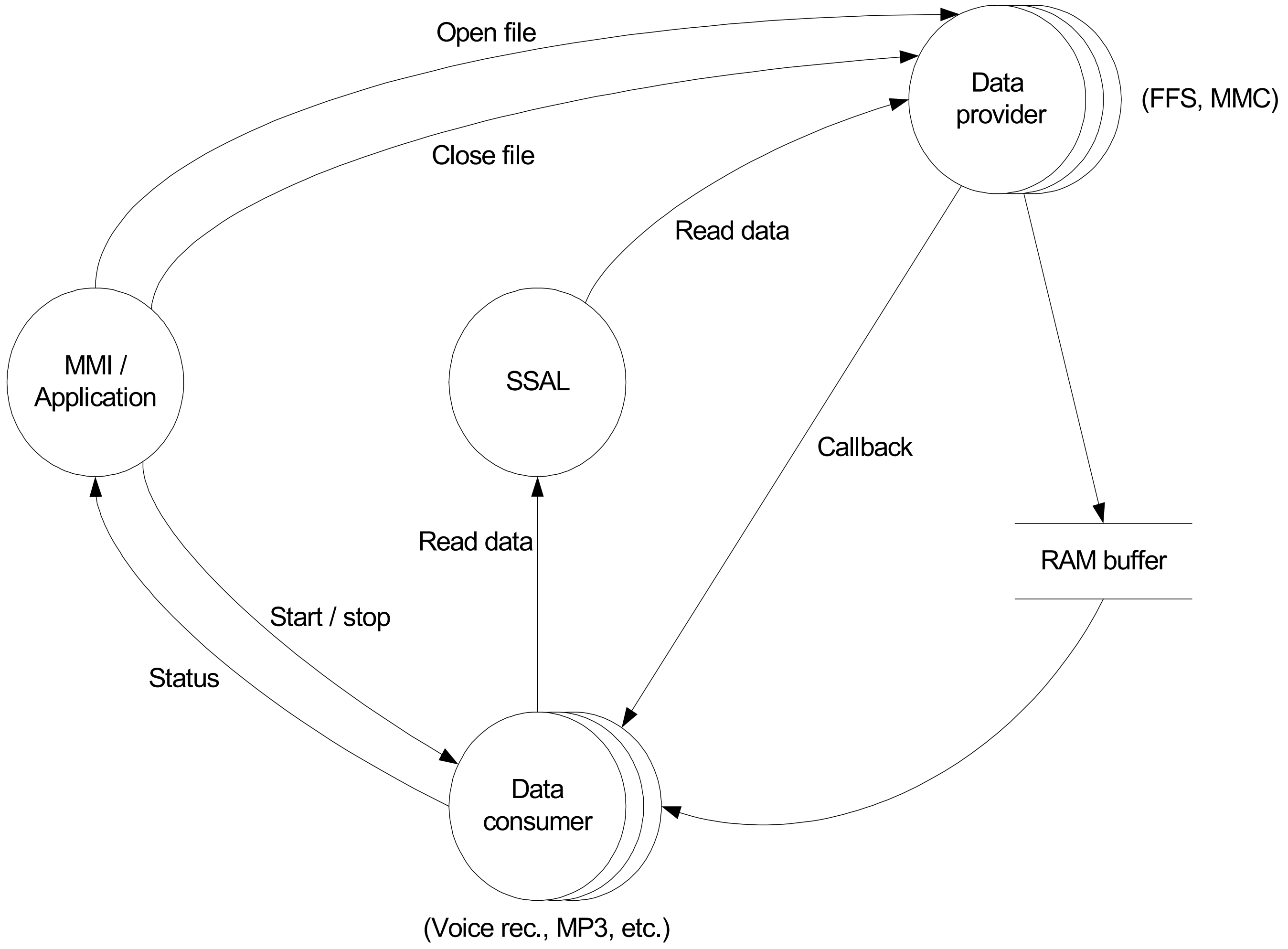
Streaming recording



Streaming recording sequence

- Create file
- Start data source
- Send write request (via Streaming Storage Abstraction Layer)
- Data is copied from RAM buffer to FFS
- Callback function is called on completion of the write operation
- New write operations can be requested
- The streaming is stopped by de-activating the data consumer
- The file is closed

Streaming playback



Streaming playback sequence

- Open file
- Start data consumer
- Send read request (via Streaming Storage Abstraction Layer)
- Data is copied from FFS to RAM buffer
- Callback function is called on completion of the read operation
- The data is now available for the data consumer, and new read operations can be requested
- The streaming is stopped by de-activating the data consumer
- The file is closed

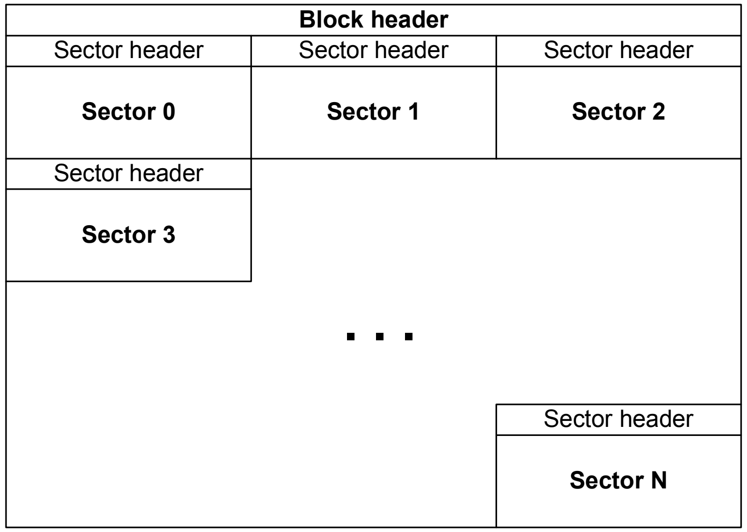
Physical format

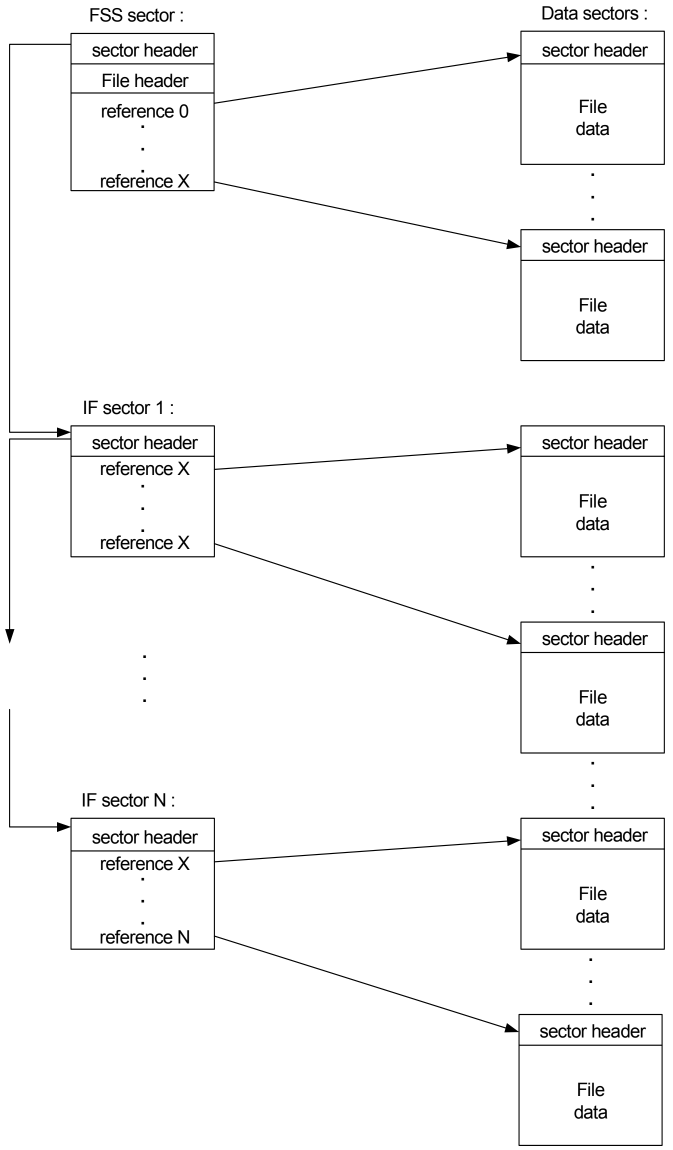
The FFS consists of N flash-blocks (e.g. 128 Kbytes each).

Each block is divided into N logical sectors.
A sector is the smallest allocation unit.

A file consists of A File Start Sector, (some Index Sectors) and N Data Sectors.

The figures shows the logical layout. In the physical layout, the sector headers are physically separated from the sectors payload data.





Confidential

Power loss recovery

The Power Loss Recovery algorithms provides a very important feature that ensures consistency in the file-system.

If the mobile station looses the power by accident, or the battery is removed while the flash is being modified by the FFS-task or the Garbage collector, it will be detected by the power loss recovery algorithms at the next power up. The recovery algorithms will then perform the required clean-up or finish operations in progress (if possible).

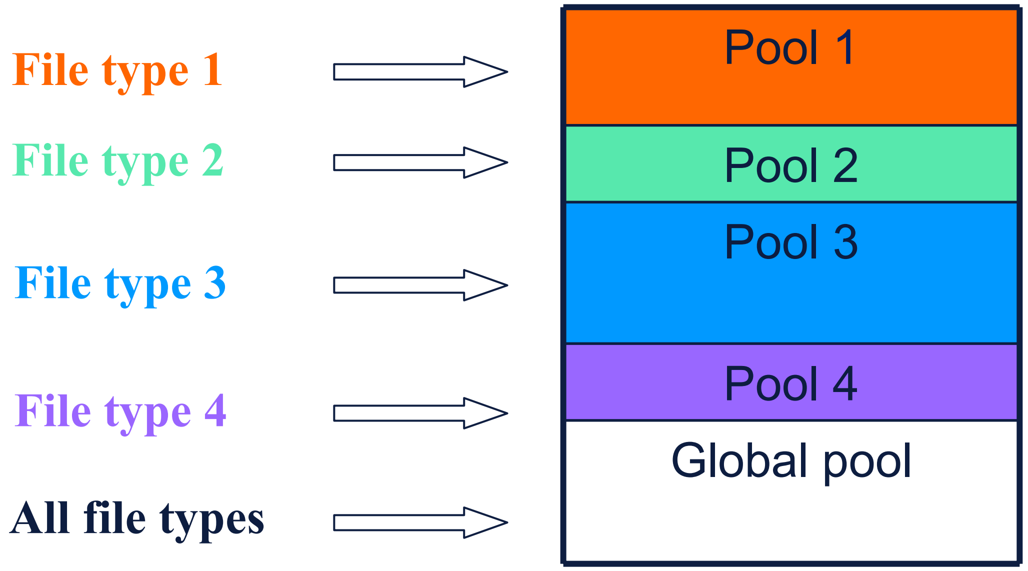
So, the worst thing that can happen if the power is lost, is that requested modifications which are not yet updated in the flash, will be lost.

File types and memory pools

N different file types can be declared in FFS.H, and for each file type a dedicated memory pool can be reserved.

The dedicated memory pools ensures that a minimum of memory always will be available for the corresponding file type.

Besides the dedicated pools there's a global pool for general use. Memory for a specific file type can not be allocated in the global pool, before the dedicated pool is full.



API – File identification

File ID:

To be able to make unique identifications of the different files in the FFS file ID's are used.

A file ID is a number (0-65535). The ID must be used whenever files are accessed.

To create a new file a ID must be reserved and used.

Pre-assigned ID table:

It's possible to pre-assign ID's to files already at compile-time.

Elements added to the Pre-assigned ID table will reserve an ID which is known by the user at compile-time.

File Name:

As an alternative to the ID identification a filename can be used. Internally the FFS converts this name to an ID.

API – User identification

User ID table:

To allow the FFS to recognize the FFS-users, all users most be declared in the User ID Table.

An FFS user is a process which accesses files in the FFS.

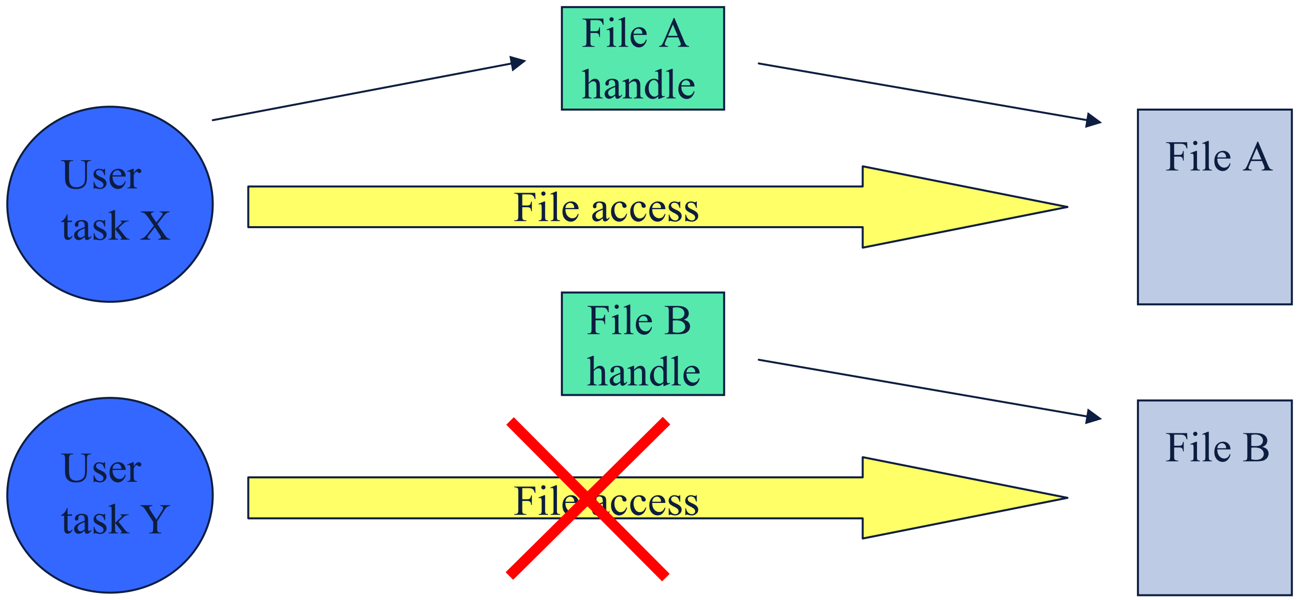
For each user in the User ID Table a user-priority is defined. This priority is used when multiple users are requesting FFS-operations simultaneously. This allows an active FFS-operation requested by a low-priority user to be interrupted by an operation requested by a high-priority user.

API – File handles

To control access to the files in the FFS, and to prevent more processes to operate on a file simultaneously, file handles are used.

A file handle is a pointer to a logical representation of the file in the FFS. When requesting the Create and Open operations a file handle is returned.

To be able to operate on a file the corresponding file handle is required when requesting operations.



API – Basic interface functions

File operations:

Create	Open
Close	Read
Write	Append
Modify	Rename
Delete	Truncate
Copy	

Directory operations:

Make dir	Remove dir
Rename dir	Is Dir
Set cwd	Get cwd

Service functions:

File Exists	Get File Info
Find First	Find Next
Get Free Space	Get Number Of Files

Never stop thinking

Questions

? ? ?