PJFS (Partitioning, Journaling File System): For Embedded Systems

Ada-Europe

6-June-2006

Greg Gicca

Why File Systems Are Important

- **Most computer systems today, even deeply embedded, have ample storage media**
  - It is hard to find any computer without flash
  - ATA/IDE also very common (CompactFlash, disk)
- **Files used to store**
  - Configuration information
  - Applications to load at run-time
  - Parts of the operating system
  - Application-specific data
    - Mission/Science
    - Diagnostics
    - Sensitive information
- **In some systems, file system operation is critical to safety, security, and/or system reliability**

Legacy File Systems

- **Legacy File Systems**
  - Windows DOS/FAT/NTFS
  - UNIX VFS/FFS
  - MacOS HFS
- **File system and file data integrity can not be guaranteed**
  - Sudden power loss
  - File system bugs
  - Other hardware/software failures
- **If you're lucky, some data is lost, but file system can still come back up**
  - If you're unlucky, the entire file system can be lost

Legacy File Systems

- **No quality of service guarantees**
  - One client can exhaust the disk/flash
  - One client can write a file while another is reading/writing
  - Performance of client affected by other clients
- **File system start up takes too long**
  - Boot that follows a sudden interruption
  - Time proportional to size of file system storage
- **File systems can not be assured to the highest levels of reliability**
  - DO-178B Level A, Common Criteria EAL-6+, IEC-61508 SIL3/4
  - Tens of thousands of lines of code not uncommon
- **File systems have huge footprints**
  - Not suitable for embedded

Journaled File Systems

- **User's "write" call translates into several file system operations**
  - Sudden interruption during this sequence is what causes corruption
  - Sometimes recoverable by walking the file system to correct inconsistencies
- **Journal is used to log sequence of related operations**
  - High level "write" is "atomically" committed from the log
  - Lose only uncommitted operations on sudden interruption
  - File system integrity is guaranteed
  - Start-up time is very fast
- **In a few years, it will be hard to find new computers running a non-journaled file system**

Legacy Journaled File Systems

- **Examples**
  - Linux ext3
  - IBM JFS
  - Namesys ReiserFS
  - Datalight Reliance
  - SGI/Linux XFS
- **Many of the same problems as legacy non-journaled**
  - Not designed for high assurance
  - Not designed for embedded
  - No provision for preventing client interference
- **Some only journal the “metadata”**
  - File system integrity guaranteed, but data loss can occur
  - ext3, ReiserFS, XFS, JFS

Green Hills SOFTWARE, INC.

Introducing: PJFS

- **PJFS is the only commercial**
  - high assurance
  - crash-safe
  - file system
  - *in the world*
- **PJFS implements *full* journaling**
  - metadata and data
  - fast boot
- **POSIX API**
  - lightweight non-blocking read/write extensions

Green Hills SOFTWARE, INC.

Introducing: PJFS

- **PJFS is designed for high assurance**
  - A couple thousand lines of code
    - Contrast: ReiserFS is over 100K SLOC
  - Using same High Integrity Development Process as INTEGRITY RTOS
- **PJFS small footprint is great for embedded**
  - Yet scales to workstation environment
- **PJFS is a "MILS file system"**
  - Uses the separation kernel to implement its own separation policies

Green Hills SOFTWARE, INC.

Introducing: PJFS

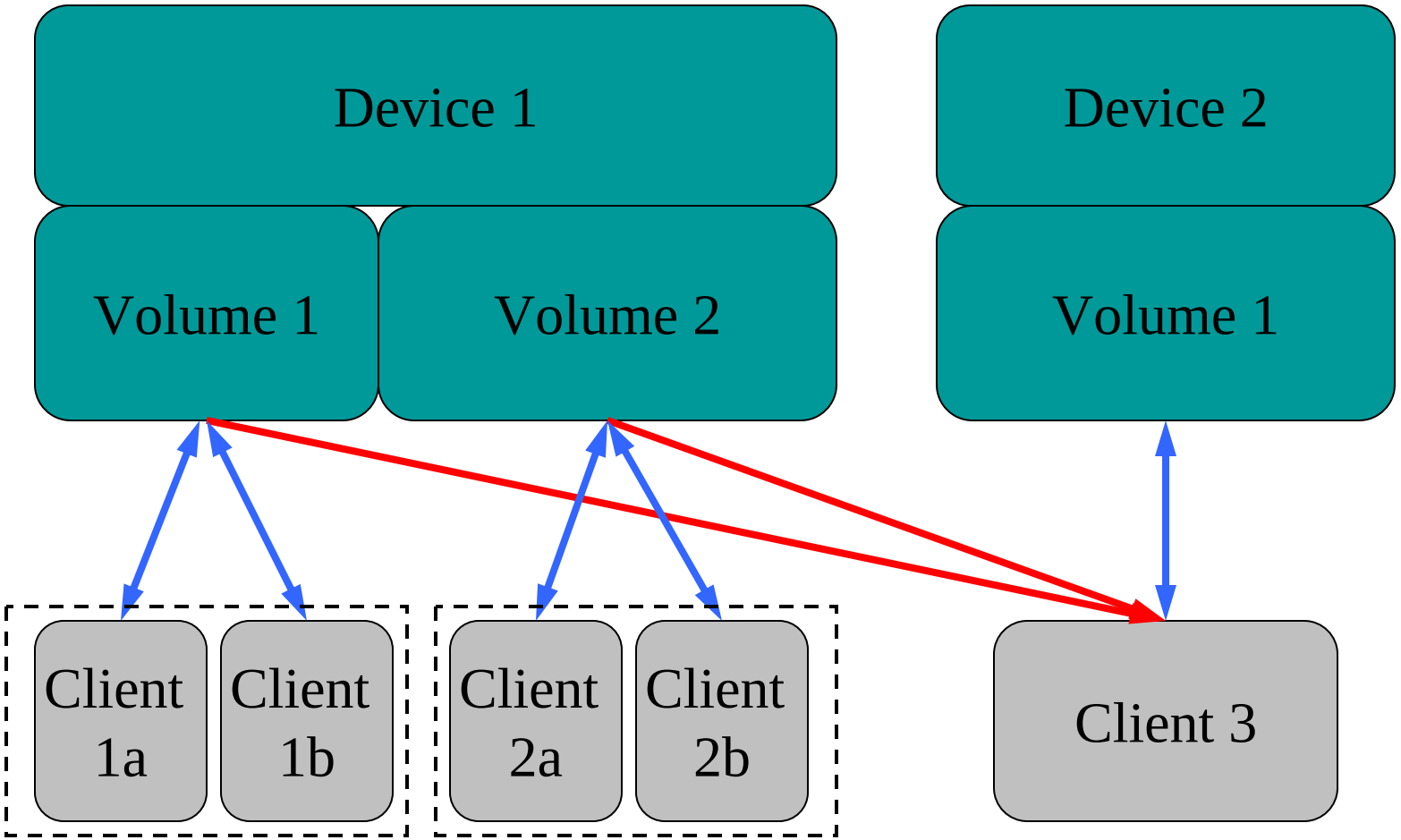
- **PJFS has a unique architecture for quality of service guarantees**
  - Patent-pending
  - A reader always has a consistent view, even if another client is simultaneously writing to the same file
  - Read time is unaffected by write operations
  - Media can be divided into partitions (aka volumes) that have guaranteed storage allocation
    - Extends the INTEGRITY concept of GRA to the external file storage (e.g. flash, disk)

Introducing: PJFS

- **PJFS has a unique architecture for quality of service**
  - Each volume can be limited to one “owner/writer”
    - No risk of unintended write interference
  - Each volume has an independent file system cache
    - Performance/CPU time independence
  - Each volume has an independent journal
  - Each volume has an independent set of worker threads

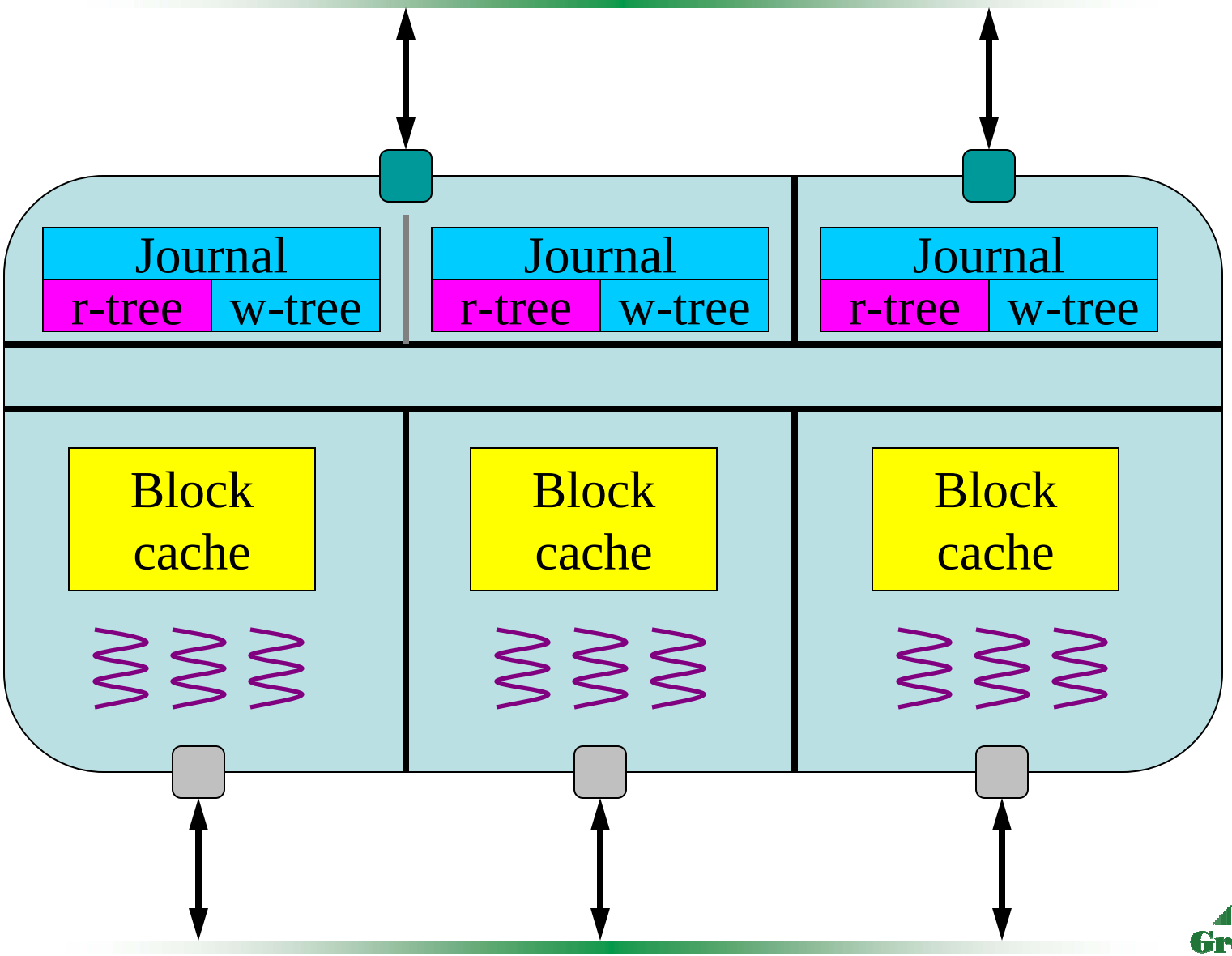
Green Hills SOFTWARE, INC.

PJFS System Logical View



Green Hills SOFTWARE, INC.

PJFS Server Architecture View



ARINC-653 On File Systems

- **Defines the File System to conform to the POSIX standard**
  - IEEE 1003.1-1988 and later variants as well as parts of international standard ISO-IEC 9945
- **ARINC-653-2 Part 2 (draft)**
- **On Partitioning:**
  - The ARINC 653 File System should not violate time and space partitioning requirements. Space partitioning considerations include prevention of modification of file data not owned by the partition. The file system can be used by partitions to write data to a file and read data from a file. Files can be read by multiple partitions but written to by only one partition.
- **On Journaling:**
  - On a halt or cold or warm restart of a partition **...**
    It is implementation dependent whether the partition's write requests received by the file system are completed, but if they occur, the writes complete atomically.

ARINC-653 On File Systems

- **Thus ARINC-653 requires a PJFS**

- **It requires a POSIX file system API and semantics**
- **It requires partition safe operation**
- **It requires atomic writes in the case of failure**
- **The standard then defines full APIs for the Ada and C languages**
- **ARINC-653-2 part 2 updated the Ada API to Ada95**
  - Thus ARINC is still tracking and supporting the Ada language
  - Updated to the current ISO standard
  - Bad Luck that it will be finalized around when Ada2005 will !