Developing Scalable and Portable CIFS Server

Single solution for any CIFS application

Plan

- Introduction
- The Challenge
- Architecture In General
- Decompositions
- Modeling
- Porting & Integration
- Q & A

Introduction

Who We Are

Company

- Visuality Systems Ltd.
- Embedded/Mobile CIFS
- CIFS Acceleration

Speaker

- Mark Rabinovich – Project Manager

Experience To Share

- NQ – Client/Server CIFS solution
- Above 80 customers in more than 15 industries
- Portable solution
- Currently suites embedded and mobile markets

Today's Theme

- We will discuss the CIFS Server – the CIFS Client not covered
- We will analyze requirements introduced by various and, sometimes, contradicting markets
- We will see how to satisfy those requirements in a framework of a single solution

The Challenge

Dimensions of Flexibility

- Industries…

  introduce various and, usually, contradicting requirements

- Operating Systems…

  not only have different APIs but require very polar approaches sometimes

- CPU…

  not as crucial as the OS, but might introduce certain challenge

- Compiler…

  might apply additional difficulties

Industries

- Consumer products: media players, set-top boxes, printers, scanners
- Network appliances: home routers
- Medical equipment
- Military
- Aerospace
- Mobile telephony
- …and more

More Industries

And of course – the traditional CIFS consumers

- PDAs
- Desktops and notebooks
- Servers
- NAS

Operating Systems

A partial list of operation systems to be considered:

- VxWorks
- Linux (numerous flavors)
- Integrity
- ThreadX
- Windows CE
- ITRON (numerous flavors)
- Mobile OS (Symbian, SE-OS, etc.)
- Others...

Operating Systems (cont.)

- Posix versus Object-Oriented
  - Native API may be either functional (Linux, VxWorks, etc.) or object-oriented (Symbian)
- Where is my "printf"?
  - Basic OS services may differ in syntax and even in semantics
- "Select" or not "select"?
  - Main mechanisms may have incompatible models: like select-controlled sockets versus event-driven sockets
  - Fortunately, most of operating systems feature comparable File System models although APIs may differ.

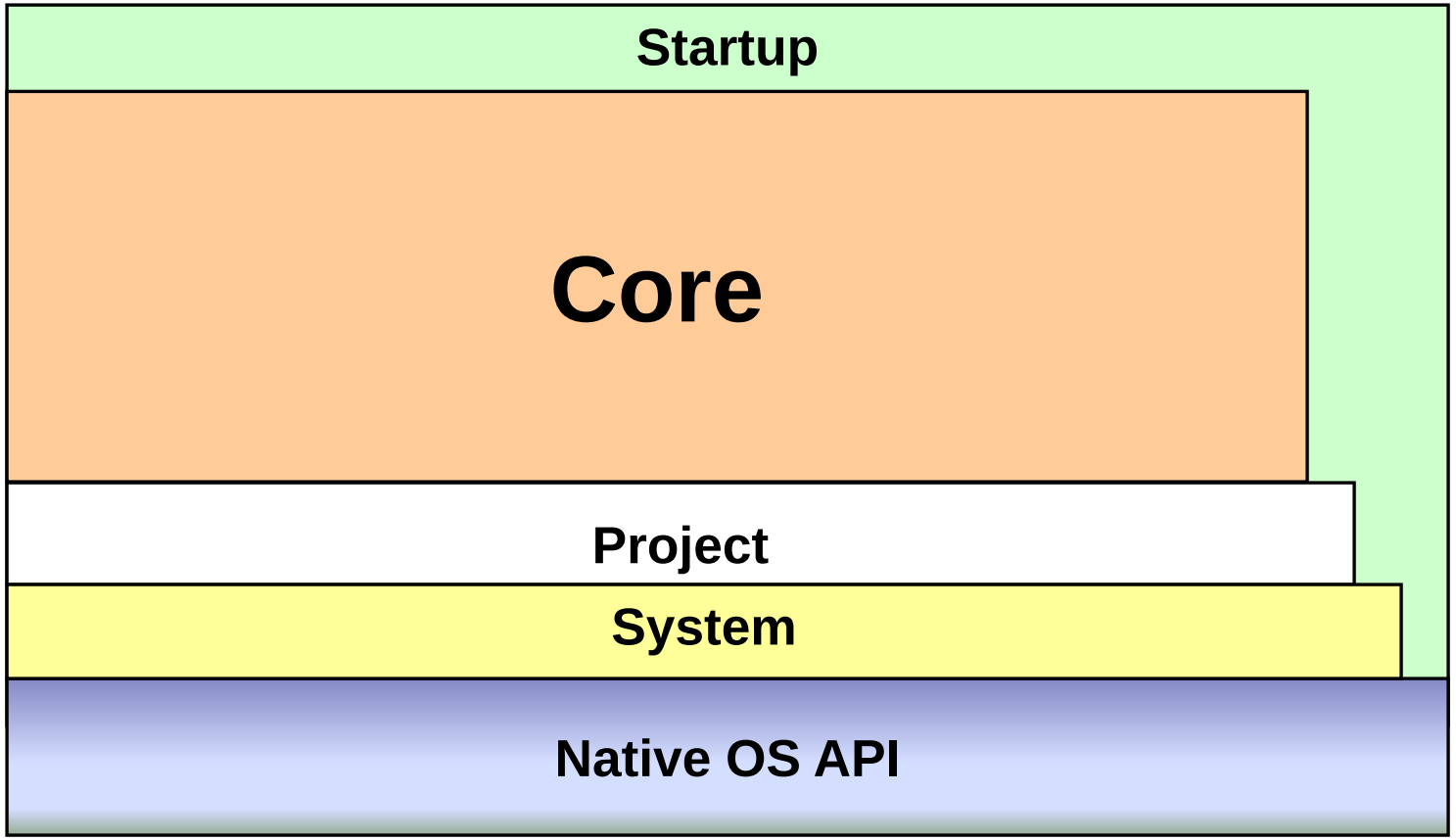
CPU + Compiler

Hardware and compiler introduce more challenges:

- Byte order:
  - Big Endian or Little Endian?
- Alignment:
  - bet Odd or bet Even?
- Structure packing
- Stack size

Architecture In General

Architecture



Layers

- **Core** – shared by all solutions
- **Startup** – system-dependent, may be modified per project
- **System** – reusable by projects sharing the same platform
- **Project** – per a project

API between the Core layer and System/Project layers is an abstraction layer API

Goals

- The **Core** component should cover as much code as possible (at least 90% expected)
- It should be configurable to suite different applications/industries
- High level of abstractions in APIs between layers

Decomposition

Application Types

- Embedded

  Low-scale. Provide CIFS services "as is" regardless of the performance. A couple of client connections. An integrated service.

- Mobile

  As above but runs as a loadable application.

- Desktop

  Assumes limited number of client connections with medium-to-high performance.

- Servers and NAS

  Upper-scale. Serves numerous client connections with high performance.

Resources Per Application

The numbers are for an SMB1 Server. SMB2 Server requires more open file and search resources. The numbers are for consideration only and they do not necessarily reflect each application.

	Embedded	Mobile	Desktop	Servers
Shares	1-3	1-2	Unlimited	Unlimited
Concurrent clients	1-3	1-2	10	Unlimited
Tree connections	2-6	2-4	20-30	Unlimited
Files	20	10	100	Unlimited
Open files	20	10	150	Unlimited
Searches	5	3	20	Unlimited

Features

- Memory usage
  - **Static** = preallocated
  - **Dynamic** = heap
- Resources
  - **Limited** = fixed size tables
  - **Medium** = limited allocation
  - **Unlimited** = free allocation
- Performance:
  - **Low** – "as is" SMB Server
  - **Medium** – emphasis on performance without applying extra mechanisms
  - **High** – the maximum performance

Mechanisms

- Multithreading
  - Thread per client connection
  - Thread pool
- Pipelining
  - Transferring payload between protocol layers: SMB/Transaction/RPC
  - Happens because of different buffer limits
- Bulk data transfer
  - Socket-to-file
  - File-to-socket

Features Per Application

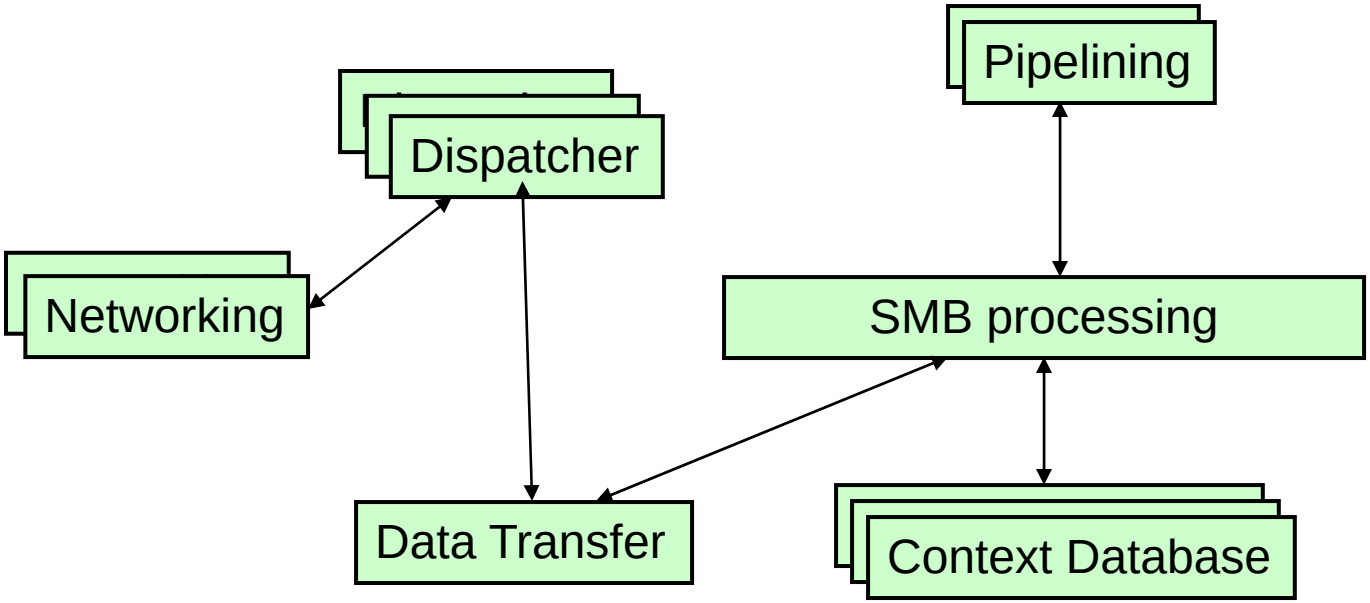
	Embedded	Mobile	Desktop	Servers
Memory usage	Static	Dynamic	Dynamic	Dynamic
Resources	Limited	Limited	Medium	Unlimited
Performance	Low	Low	Medium	High
Multithreading	-	-	V	V
Thread pool	-	-	-	V
Pipelining	-	-	V	V
Bulk transfer	-	-	-	V

Modeling

The Approach

- We have to put at least 90% of the code into the "Core" component.
- The trade-off: use a set of alternative (replaceable) solutions to:
  - better abstract the API between the Core layer and system-and-project-dependent layers;
  - better suite conflicting application requirements

Assembling The LEGO



This model is suitable for any application but Server/NAS

Some components have optional solutions

Networking

- Select()-based approach
- Event-driven approach

Why two solutions?

Implementing Select()-based API on top of event-driven sockets involves writing too much system-dependent code.
… and vise versa.

See also Porting below

Packet Dispatcher

- Single-threaded
  - Easy to implement and maintain
- Thread-per-connection
  - For a limited number of connections
  - Assumes light-weight threads
- Thread pool
  - For numerous connections

Context Database

Keeps the context of SMB conversation

Part of the context may be external to SMB.

The following data types are likely to be internal:

- Connections
- Shares
- Tree connections
- Users

Some data types can be external and shared with NFS, FTP, etc. This is suitable for Server/NAS applications

- Files
- Open files
- Searches (directory scans)

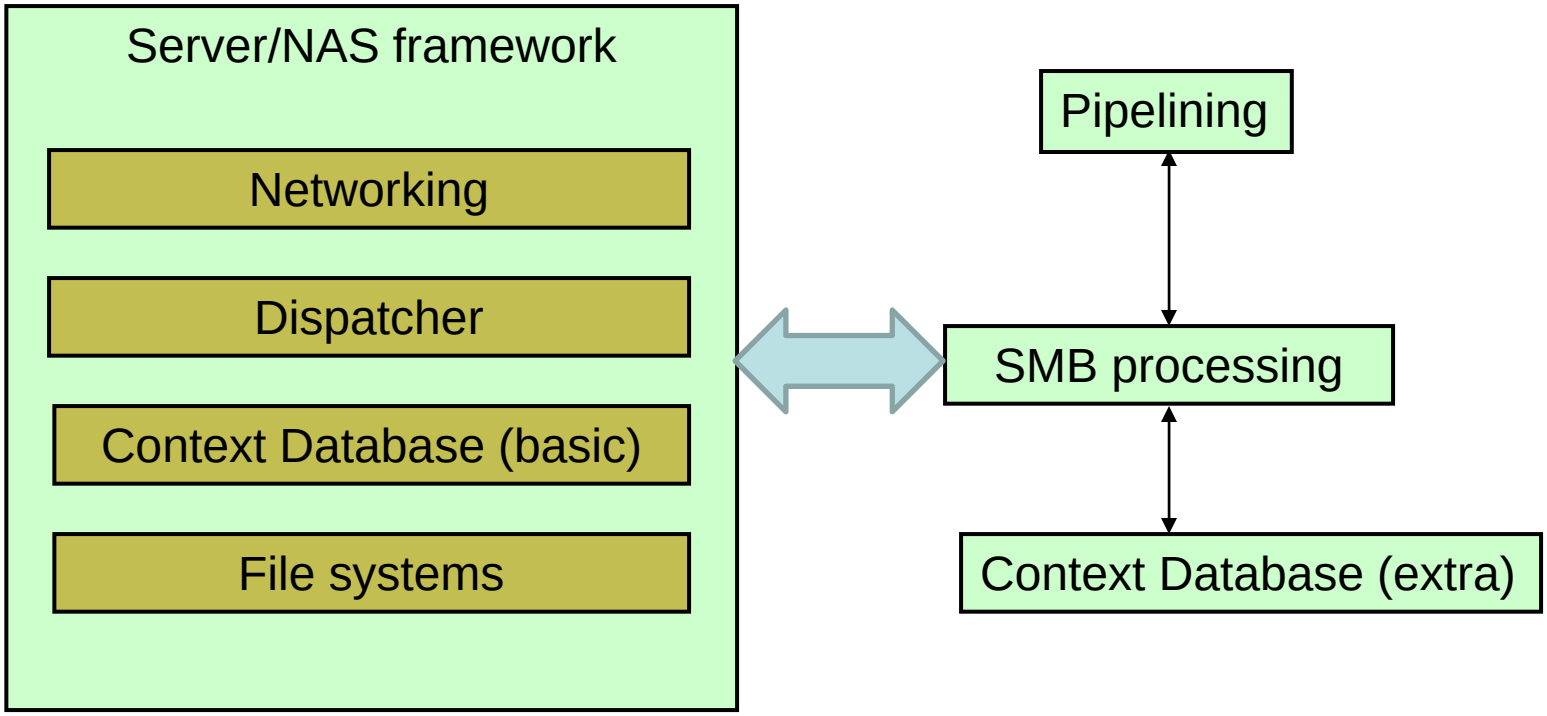
Context Database (cont.)

- Optional solutions:
  - Fixed size tables
  - Dynamic allocation with pre-allocation and limits
  - Unlimited dynamic allocation

Pipelining

- Pseudo-pipelining – limited functionality
  - The same buffer is used
  - Processing happens in the same thread
  - Limited Transaction/Transaction2
  - SPOOLSS not available
- Full pipelining – requires threads for:
  - Transaction/Transaction2/NtTransact
  - RPC

Server/NAS



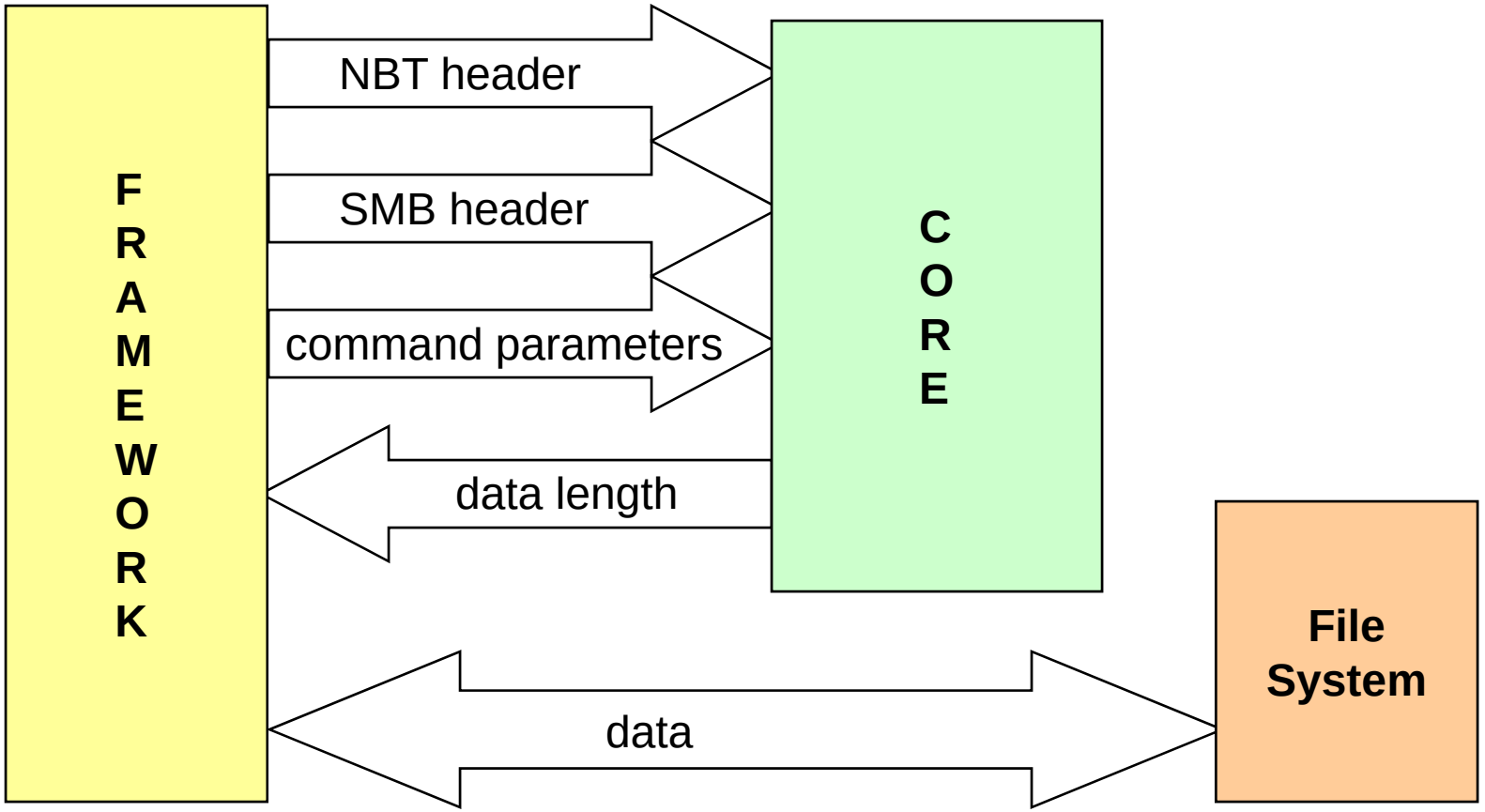
Networking, Dispatcher, etc., when implemented outside SMB may be shared between SMB/FTP/NFS/etc.

Context Database may be fully/partially implemented inside SMB

Data Transfer

- Suites Server/NAS applications
- Can be compared to DMA
- Separates between SMB processing and data transfer:
  - Socket → SMB processing → File
  - File → SMB processing → socket
- May be internally implemented as yet another Core component, or …
- May be implemented outside SMB as a bulk transfer

Data Transfer Outside SMB



Data Transfer API

This is an conceptual example of Data Transfer API

Framework calls:

smbParseRequest(callback);

callback->receive(buffer, 4);

callback->receive(buffer, HEADERSIZE);

callback->receive(buffer, commandSize);

…on Write(WriteAndX) calls:

callback->transferDownstream(fileID, dataSize);

callback->send(nbtHeader, 4);

callback->send(smbHeader, HEADERSIZE);

callback->send(commandPaylod, commandLength);

…on Read(ReadAndX) calls:

callback->transferUpstreams(fileID, size);

Porting and Integration

Porting and Integration

"Porting" stands for modifying the CIFS solution to suite yet another platform – a combination of:

- OS
- Compiler
- Hardware
- Project-dependent solutions

"Integration" stands for modifying the CIFS solution for yet another project on the same platform

Issues and Goals

- It is virtually impossible to support some 20 OS solutions.
- Modular approach to share solutions between OS-es
- Inherit yet another solution from sample solutions
  - Linux
  - VxWorks
- Separate between System Layer and Project Layer

Decomposition

System Layer

- File System
- Networking
- General services
- CPU + Compiler
- Trace log

Project Layer

- Users and passwords
- Shares
- Home domain

Q & A

Thank you