Notas ER

Slide 2:

Android Application entry points:

- Launch activity
- Services (background)
- Receivers (receive intents)
- Exported components (debug)

AndroidManifest.xml:

- Contains Permissions, Intents and start classes
- Exposes public interfaces and data sources.

META/MANIFEST.MF:

- Contains hashes for all files.

Classes.dex

- Contains all Dalvik bytecode
- Includes app code and some libraries

JVM:

- Stack Based Machine (Stack that pushes and pops values)

Android Environment:

- Runs linux
- Each application executes in a independent VM instance
- Crashes are limited to 1 app
- Data isolation is ensured by independent execution

Dalvik VM:

- Before execution, files are optimized.
- odex file is an optimized files where the functions are resolved in to the

vtable
- Bytecode is processes using JIT (real time during execution)

Dex (dalvik executables) can be converted to java and vice versa.

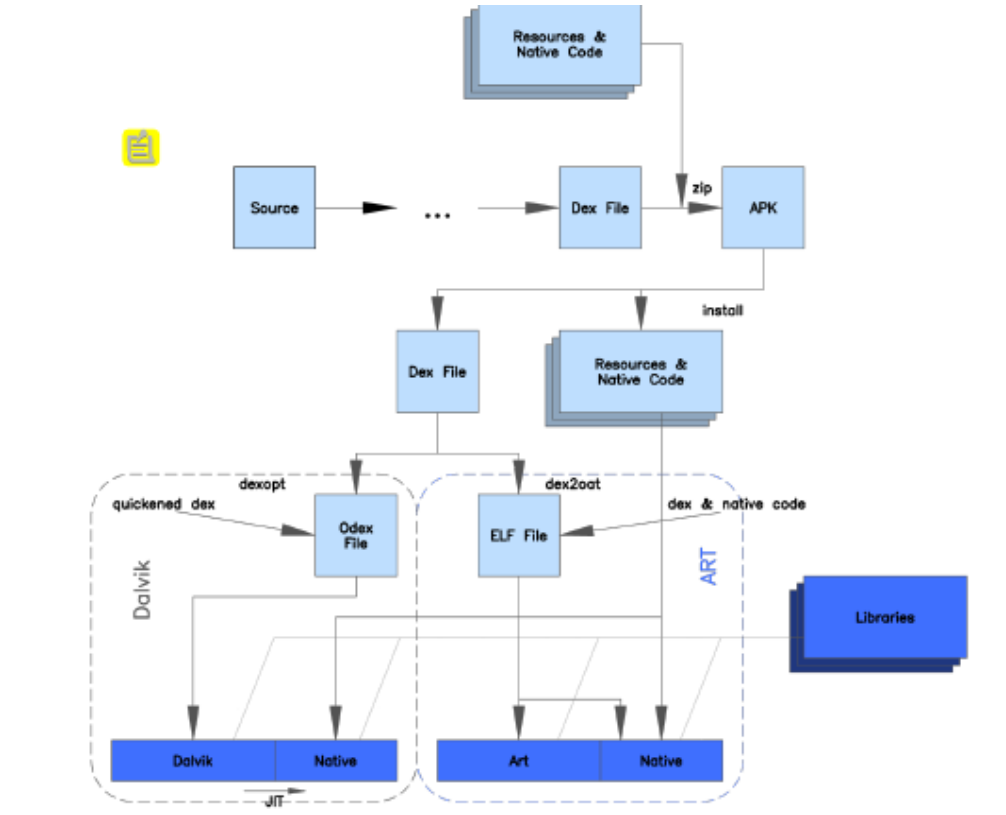
Android RunTime (ART):
- Optimized execution path
- Runs OAT files
- Ahead of time compilation

.oat - Elf containing oat data (ART).
.odex - .oat containing precompiled apps

Java methods in DEX are mirrored in C++.
java.lang.string -> art::mirror::String

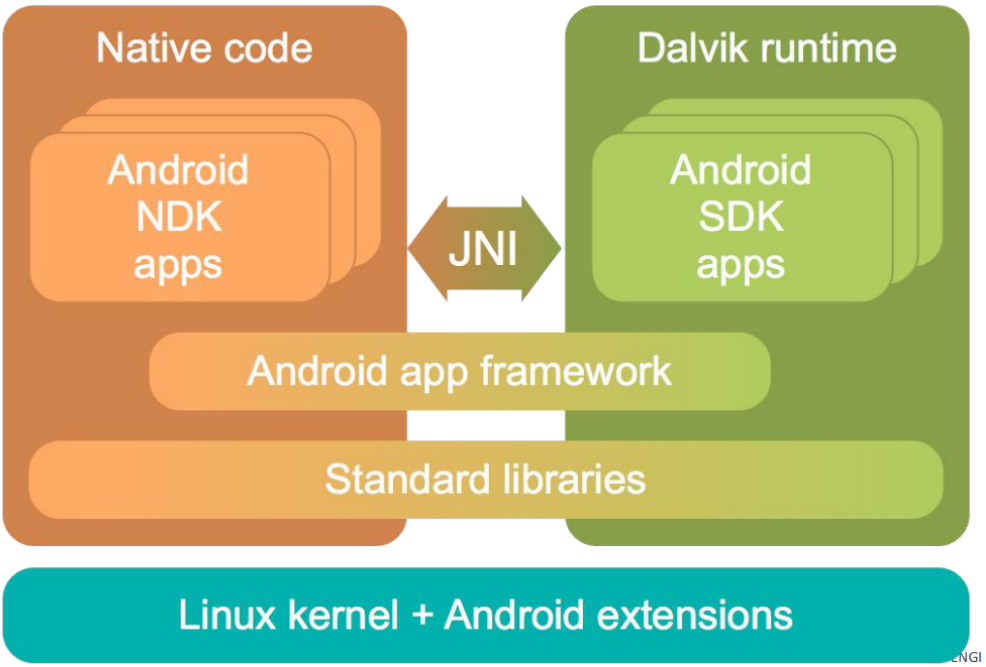
Smali: assembly of dex bytecode

Dex obfuscation objectives:
- Code shrinking
- Resource shrinking
- Obfuscation
- Optimization



Slide 3:

Java Native Interface: allows the definition of Java methods, whose implementation is present in native code



Android Native Development Kit (NDK): Android provides a somewhat rich Development Kit allowing C/C++ applications to access Android resources. One for each architecture. (libraries).

Before the binary libraries can be used, Java must load them with System.loadLibrary

Native methods support arguments from Java code
Arguments are pointers to Java structures
Native methods can also call Java methods, and classes though JNIEnv
Native methods are declared with native and no implementation.

JNI Dynamic Linking:

- Schema:

```
Java_pt_ua_deti_hello_Worker_doWork()
```

magic | Package name | Class name | method

JNI Static linking:

- Manually by the binary code.

With static linking, reverse engineering is to find the JNI_LOAD Method and find RegisterNatives on the native library.

Native apps are not great, as they have low code reusability, require more maintenance etc,

Web apps: Use standard web tech (HTML,CSS,JS)
Looks like a normal app
Standalone Mobile Web Browser

Hybrid apps combine both native and web.
Web for the interface, java for the backend, custom interface connecting both levels.

Slide 4:

Android Dynamic Analysis: Observe the application white it is running.

What can be analyzed:

- Message exchanged with external servers
- Intents send ou received
- Logs
- Files accessed / created
- Memory content

Android logs are considered dynamic analysis
MIM with proxy for external API's
For HTTPS traffic you have to install a CA certificate on the device
Because of certificate pinning, the CA certificate might still not work as the application might ignore it. To solve this, patch the application or run the application on a VM.

FRIDA allows:

- Tracing network communications at the method level

- Understand how the application behaves
- Manipulate the methods called, arguments and return codes

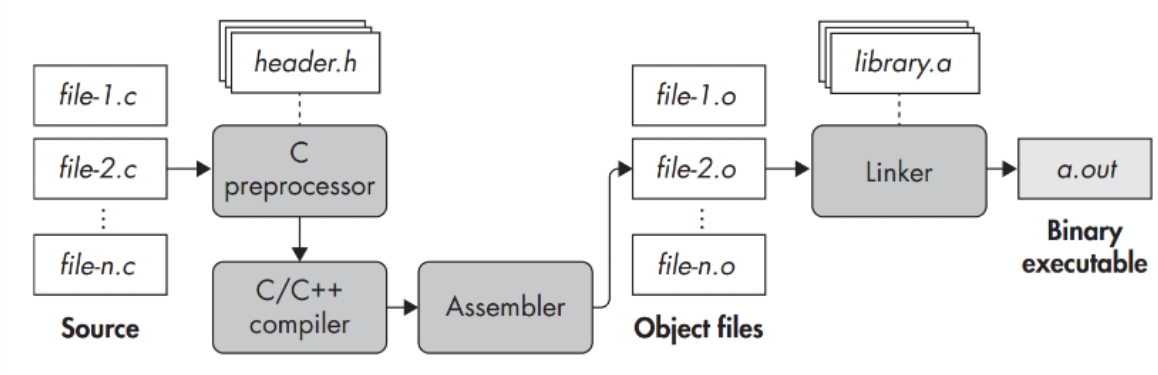
The application needs to be patched to execute frida as a binary.
A frida server can be created on the android with root and without google services.
Can change a function implementation:

```
Java.perform(function(){
  Java.use("com.re.lab1.b").checkAppSignature.implementation = function(a) {
     console.log("Signature will fail");
     return 0;
  };
});
```

Can also trigger events for when functions are called, for example.

Slide 5:

Binary Files



Preprocessor expands macros and validates code structure e agrega os ficheiros.
C compiler turns code to assembly
Assemblers transform assembly into machine code.

Linker some references to libraries get resolved. Creates the executable with the code and multiple dependencies.

Symbol tables:

- dynsym: symbols which will be allocated to memory when the program loads
- .symtab: contains all symbols used for linking and debugging

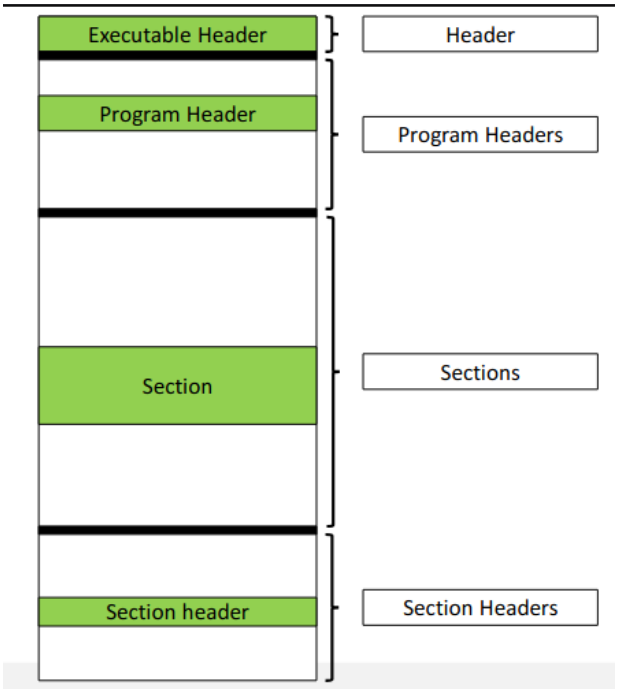
Only dynsym is required.

stripping a binary removes all unused symbols and code from the binary. Only dynsym is kept.

Object file:
- rodata (read only data)
- got (global offset table)
- plt (procedure linkage table)
- bss (block starting symbol)
- .dynsym (list of symbols)

File is split according to existing sections

ELF files:
- Container for executable files, object files, shared libraries etc.



Executable header:
- Magic number

- Architecture
- Entry point etc.

ELF Sections

- .init and .fini -> Contains code required to execute before/after the library entry point is executed.
- .text contains mais program code (where the entry point exists)
- .rodata (read only data)
- .data (info to initialize variables)
- .bss (uninitialized variables)
- .plt
- .got

Lazy Binding:

- call external function
- Go to the plt correspondent to external function
- Jumps to the address in the correspondent got entry
- If it's not there yet it will go to the .plt and this will place the real address in the GOT.
- Runs again, but this time the address is there

Dynamic Linker:

- Loading the process
- Communicated with LD_
- LD_LIBRARY_PATH: list of directories to search for libraries.
- LD_BIND_NOW: resolving symbols instantly
- LD_PRELOAD: Load libs first

Slide 6:

How to analyze elf files:

- File analysis
- Static Analysis
- Behavioral Analysis
- Dynamic Analysis

File analysis:

- Try to find the type of file, architecture, starting address etc...
- used commands: file, strings, xxd, ldd, nm

Static analysis:

- Ghidra: disassembler
- The disassembly process involves analyzing the binary, converting binary code to assembly but might not be perfect, linear and recursive disassembly can be used.

Linear disassembly:

- Iterate all code and disassemble when opcode is found.
- Start on some address as the entry point might not be known.
- Used in x86 as data is together normally, everytime data is found, disassembly becomes desynchronized.
- Good for finding hidden code

Recursive disassembly:

- Disassembles code since the initial point, while following the control flow.
- It syncs fast and flow can be fully recovered.

Function detection:

- functions include prolog and epilogues
- prolog sets up stack
- epilog optionally check canaries and release stack

Calling conventions:

- cdecl, arguments are passed in the stack in inverted order.
- stdcall, arguments from right to left.

Common logic structures:

- if else: conditional branches with control-flow instructions, can jump to hardcoded code, to a place stored in memory or based on a condition
- switch: jump table
- for: while and for loops are the same, and jumps to the top of the loop

Slide 7:

Dynamic Binary Analysis:

- Captures behavior that depends on external input

Load the binary and execute instructions of the target binary
Allow some interaction with the binary while it is running, this includes inspecting and changing memory.

Passive analysis: observation
Active analysis: modification

Considerations:

- Stability: reversing is more difficult when the execution is unstable (multi thread)
- Save and Replaying: Moving back in time to previous program states.
- Safety: The target binary can be malicious (Isolate with VM)
- Support of Heterogeneous Architectures: Frameworks must be extensible in order to support a wide range of architectures
- Support of Peripherals and external entities: Need to recreate the set of external devices/entities.
- Context manipulation: Only some paths according to the program execution can be analyzed, only with forced conditional branches is possible to analyze the full coverage.
- Patch files and binaries.
- The program can try to defend against analysis.

Tracers:

- Log information about system calls when executing a binary.
- Tracers add hooks

Debugging:

- Control a target executing binary.
- Extensive, interactive control over a process execution flow.
- Can be detected, for example VM detection, Ptrace TRACEME.

Sandboxing:

- An agent monitors interactions of the application inside the environment and may allow instrumentation

Emulators:

- Sandbox Backend: VMS

Remote debugging with emulators:

- GDB

Dynamic Binary Instrumentation:

- Qiling e dá para dar override a funções nsei como.

Slide 8:

Aims at hardening the process of reverse engineering

Static obfuscation techniques:

- Transforms code before execution

Dynamis obfuscation techniques:

- Transforms code during execution

Dissimulate one file type as another file type.

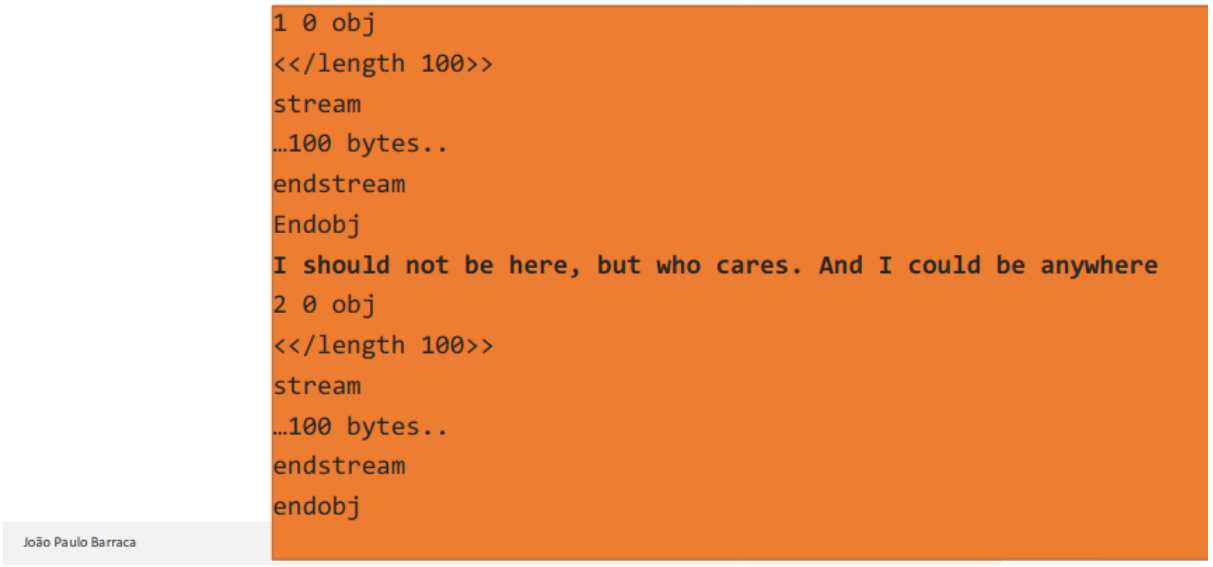
- This means that the file explorers will present a file based on the extension, but the environment might execute it differently.
- Windows hides real extension

Polyglots:

- A file that has different types simultaneously, which may bypass filters and avoid security countermeasures
- Simple Polyglot file has different types and depending how it is accessed.
- Schizophrenic file: Interpreted differently depending on the parser.
- Empty space can be used to inject crafted content.

A simple bash-pdf polyglot

Two objects and something else that is not parsed



Code obfuscation:

- hiding how source code is structured.
- removing debug info
- removing comments
- removing spaces
- stripping a binary
- Insert dummy instructions
- dead code
- inline functions etc.
- Encode data contents with XOR
- Control obfuscation with conditions that will neve happen
-

Slide 9:

Comunicação paralelo:

- N bits por n linhas de cada vez
- So funciona para distâncias curtas
- Fichas grandes
- Alto custo

Comunicação em série:

- 1 bit de cada vez
- debido elevado
- Distancias elevadas
- Diminui custos

Série funciona com shift registers, um de cada lado
1 bit por cada ciclo de relógio
Ou há relógio sincronizado através da transmissão de relógio do transmissor para o receptor, ou tem que estar sincronizado.

Transmissão síncrona: Precisa de relógio sincronizado através de um sinal adicional ou implícito nos dados.

Transmissão assíncrona: É preciso start e stop bit para sincronizar relógios, não é usado nenhuma transmissão de relógio.

Clock Stretching:

- O recetor mete a 0 o sinal enquanto não estiver disponível para receber dados.

Relógio Codificado:

- codificado com codificação manchester nos dados

Relógio Implícito:

- Os relógios são locais e existe ressincronização às vezes.

Transmissão orientada ao byte:

- 1 byte de cada vez RS232C

Transmissão orientada ao bit:

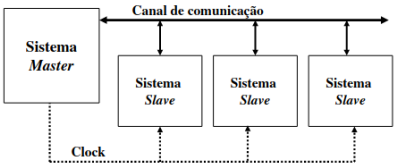
- tramas com várias partes de diferentes tamanhos separados por identificadores.

Half Duplex: Um transmite, Um recebe

Full Duplex: Ambos transmitem e recebem

Comunicação bidirecional multiponto

- Canal de comunicação partilhado
- *Half-duplex*



Slide 10:

I2C:

- Master-slave e multi-master

Transferência bidirecional half-duplex
SDA e SCL
Cada slave tem um endereço de dispositivo.
Endereços de 7 bits
Master: assegurar que se vários masters tentarem controlar o barramento, apenas um é permitido continuar.

O master controla o SCL, Inicia e termina a transferência de dados.

O slave é endereçado pelo master, e condiciona o SCL.

Transmissor/Recetor:
Um deles.

0 é o bit dominante.
Quando SCL = 1 os dados são validos

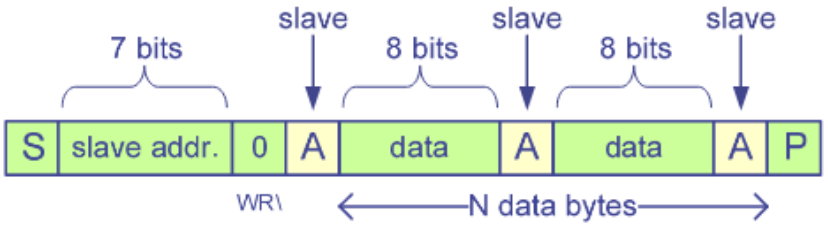
Endereço do slave +1 o bit de read write. Quando 0 o master e transmissor, quando 1 o master e recetor

Start : 1 para 0
Stop: 0 para 1

Transferência de dados

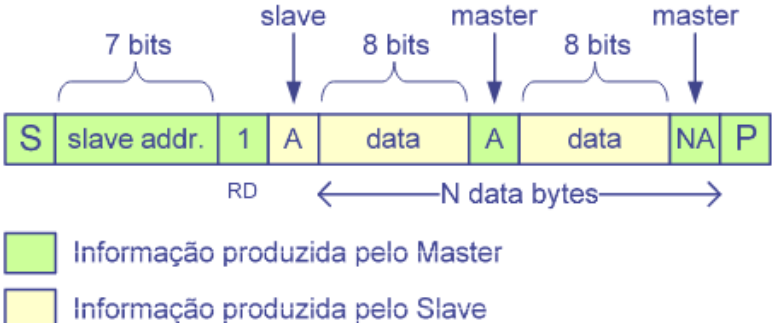
- A transferência é orientada ao byte (8 bits) sendo transmitido, em primeiro lugar, o bit mais significativo (MSbit)
- No início de uma transferência, o *master*:
  - Envia um **START** (S)
  - Em seguida envia o **endereço** do *slave* (7 bits) e o bit de qualificação da operação (Read / Write\ - R/W\)
- Após o 8º bit (o LSbit, correspondente ao bit RD/WR\), o *slave* endereçado gera um *acknowledge* (**ACK**) na linha SDA, **sob a forma de um bit dominante** (0)
- De seguida o transmissor (*master* ou *slave*) envia o byte de dados
- Após o 8º bit (o LSbit), o recetor gera um *acknowledge* (**ACK**) na linha SDA, **sob a forma de um bit dominante** (0)
- Este ciclo de 9 bits repete-se para cada byte de dados que é transferido

- Operação de **Escrita** (*master* é transmissor, *slave* é recetor)



Leitura:

- Operação de **Leitura** (*master* é recetor, *slave* é transmissor)



- Dois (ou mais) *masters* podem iniciar uma transmissão num barramento livre (isto é, após um STOP) ao mesmo tempo
- Tem de estar previsto um método para decidir qual dos *masters* toma o controlo do barramento e completa a transmissão – arbitragem de acesso ao barramento
- O *master* que perde o processo de arbitragem retira-se e só tenta novo acesso ao barramento na próxima situação de "barramento livre"

Se começarem ao mesmo tempo, aquele que descobrir que o seu bit n esta na linha cala-se

Slide 10:

RS-232C - UART

TX,RX,GND

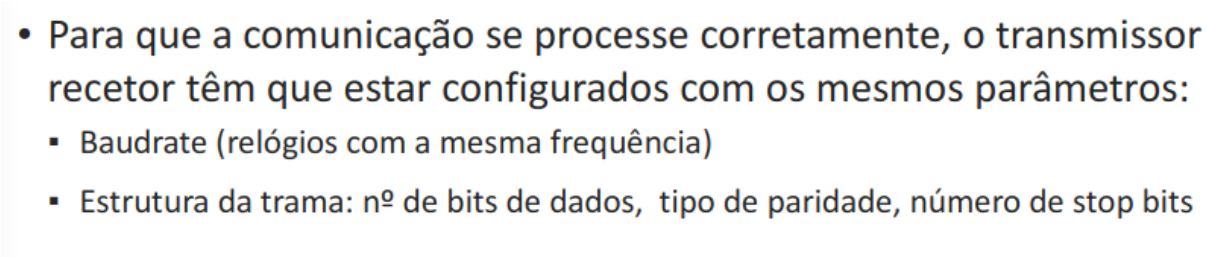
Funciona com drivers de linha

7 ou 8 bits de dados, um parity bit ou não, 1 ou 2 stop bits

Começa de 1 para 0, acaba de 0 para 1

Baud Rate = Frequência porque 1 símbolo é 1 bit

Comunicação assíncrona, não há linha de clock.
O relógio é sincronizado no primeiro bit start.



Oversampling no receptor para conseguir sincronizar e validar de x em x ciclos de relógio. que corresponde a 1 ciclo de relógio do original.

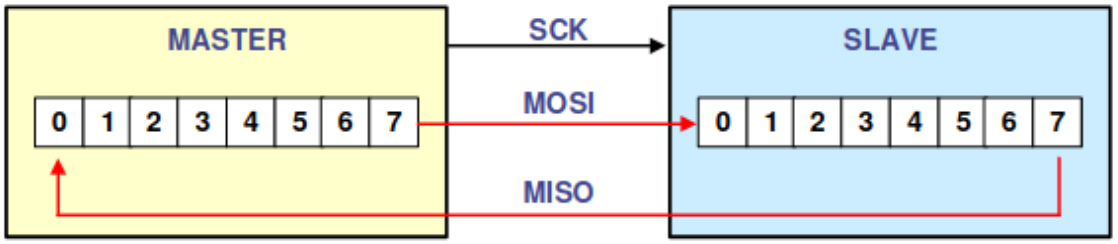
Slide 11:

Ligações a curtas distâncias.
Master-slave ponto a ponto.
Full-duplex.

Relógio explícito de master para todos os slaves.

Só pode ter 1 master.

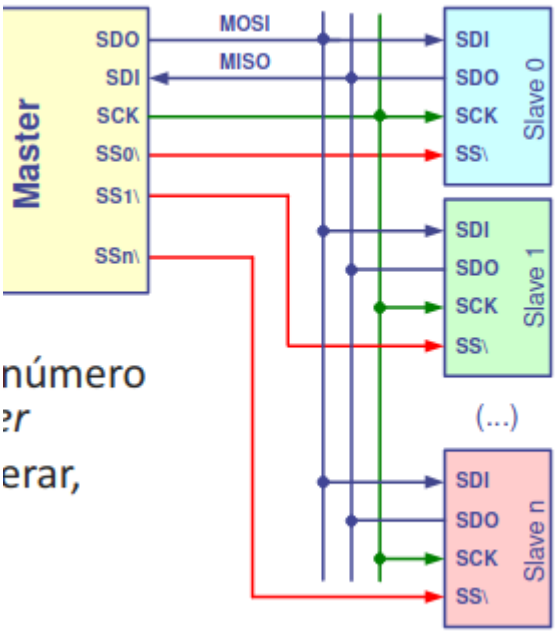
SCK, MOSI,MISO,SS
o SS é active-low



Relogio tem duty-cycle de 50%
Flanco negativo para colocarem o bit na linha
Flanco positivo para lerem o bit

Operação

- O *master* configura o relógio para uma frequência igual ou inferior à suportada pelo *slave* com quem vai comunicar
- O *master* ativa a linha SS\ do *slave* com que vai comunicar
- Em cada ciclo do relógio, por exemplo na transição positiva
  - O *master* coloca na linha MOSI um bit de informação que é lido pelo *slave* na transição de relógio oposta seguinte
  - O *slave* coloca na linha MISO um bit de informação que é lido pelo *master* na transição de relógio oposta seguinte
- O *master* desativa a linha SS\ e desativa o relógio (que fica estável, por exemplo, no nível lógico 1)
  - Só há relógio durante o tempo em que se processa a transferência
- No final, o *master* e o *slave* trocaram o conteúdo dos seus *shift-registers*



número
er
erar,

- Antes de iniciar a transferência há algumas configurações que sã efetuadas no *master* para adequar os parâmetros que definem a comunicação às características do *slave* com que vai comunicar:

1. Configurar a frequência de relógio
2. Especificar qual o flanco do relógio usado para a transmissão (a receção é efetuada no flanco oposto). Esta configuração é feita em função das características do *slave* com o qual o *master* vai comunicar:
   - Transmissão no flanco ascendente (consequentemente, a receção é feita no flanco descendente)
   - Transmissão no flanco descendente (consequentemente, a receção é feita no flanco ascendente)