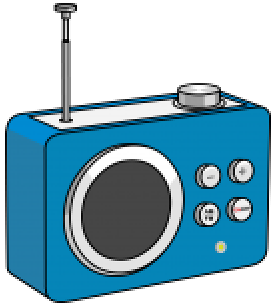
1.5. I/O

Serial Communication



Simplex

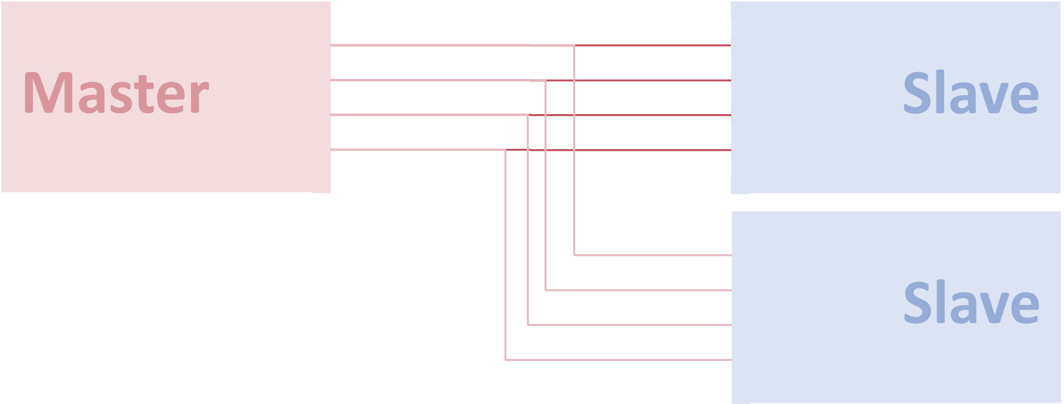
Half-Duplex

Duplex

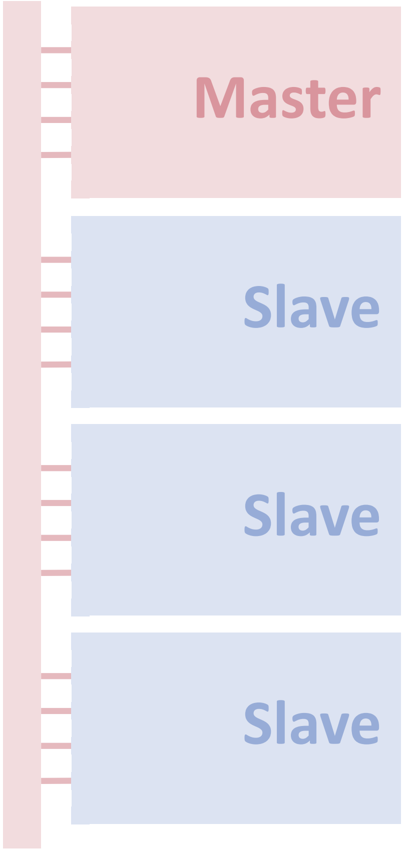
Serial Communication

Master-Slave

Master-Multi-Slave

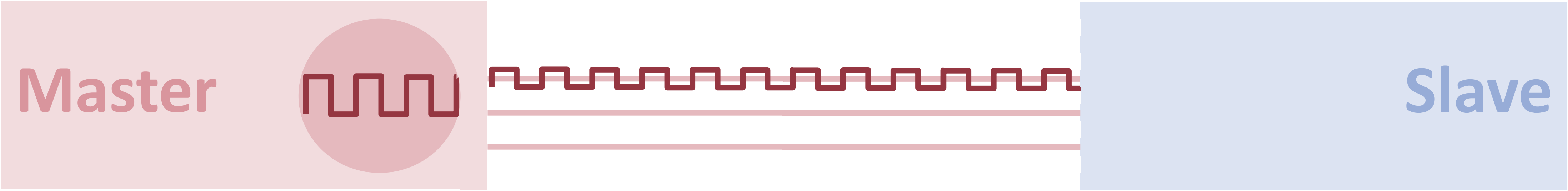


(Multi-)Master Multi-Slave

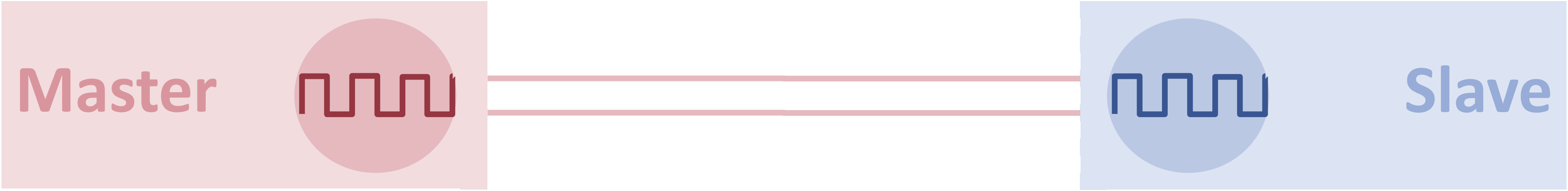


Serial Communication

Synchronous



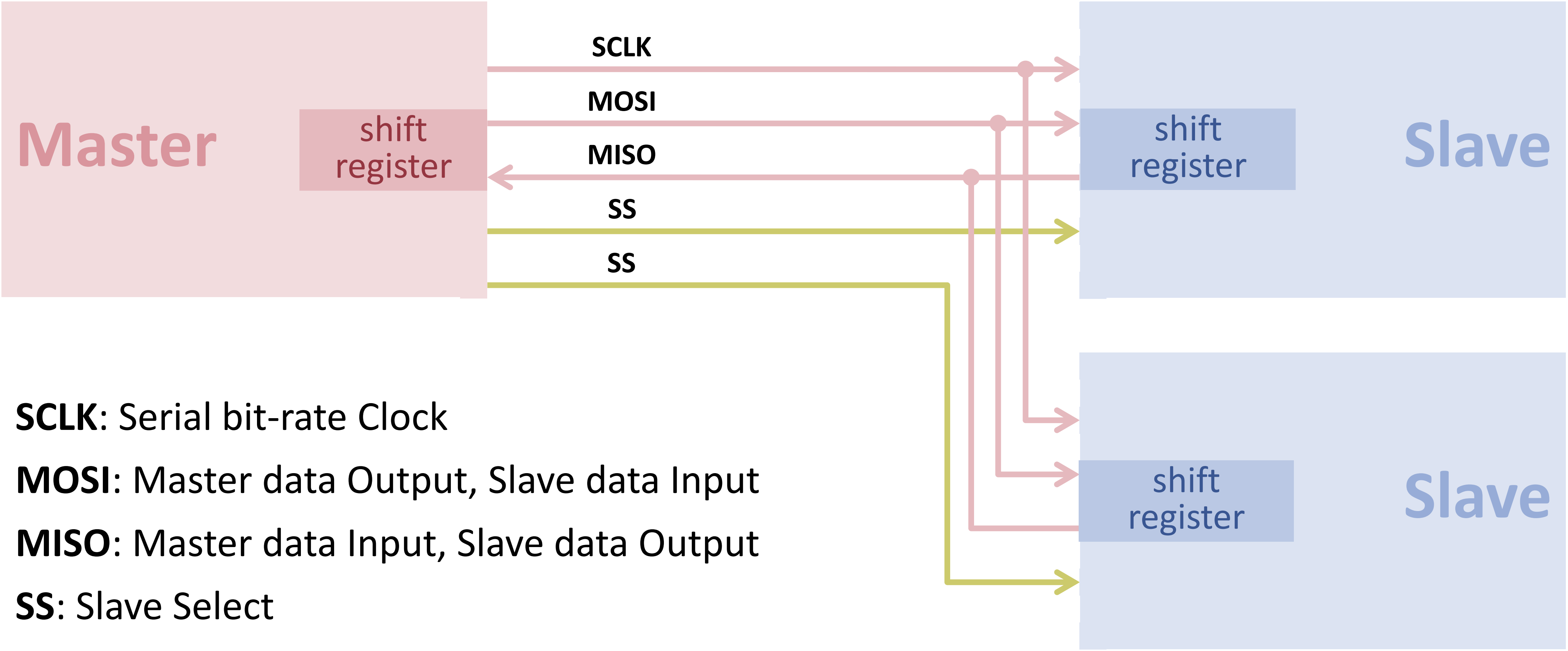
Asynchronous



Some Bus Types

	Wires (+Gnd)	Directionality	Synchrony	Distance typ.	Speed typ.	Remarks
RS-232	3/5 –8	full duplex	asynchronous +synchronous	10 m	115kbps / 1Mbps	Point-to-Point Interference prone
RS-485	3/5	half/full duplex	asynchronous	1000 m	Mbps	Differential Signalling
SPI [aka SSP, Microwire]	4	full duplex	synchronous	few cm	10 Mbps	Master-Multi-Slave with Slave select
I^2C [SMBus]	2	half duplex	synchronous	few m	100kbps-3Mbps	Addressed Multi-Master
1-Wire	1	half duplex	time-slot based, synchronous	tens of m	15kbps/ 125kbps	Master-Multi-Slave Parasitic power
USB 2.0	3/5	half-duplex	asynchronous	few m	12Mbits/ 480 MBits	isochronous/ bulk/ interrupt transfers
USB 3.0	5	full-duplex	asynchronous	few m	5/10 GBits	

SPI



SCLK: Serial bit-rate Clock

MOSI: Master data Output, Slave data Input

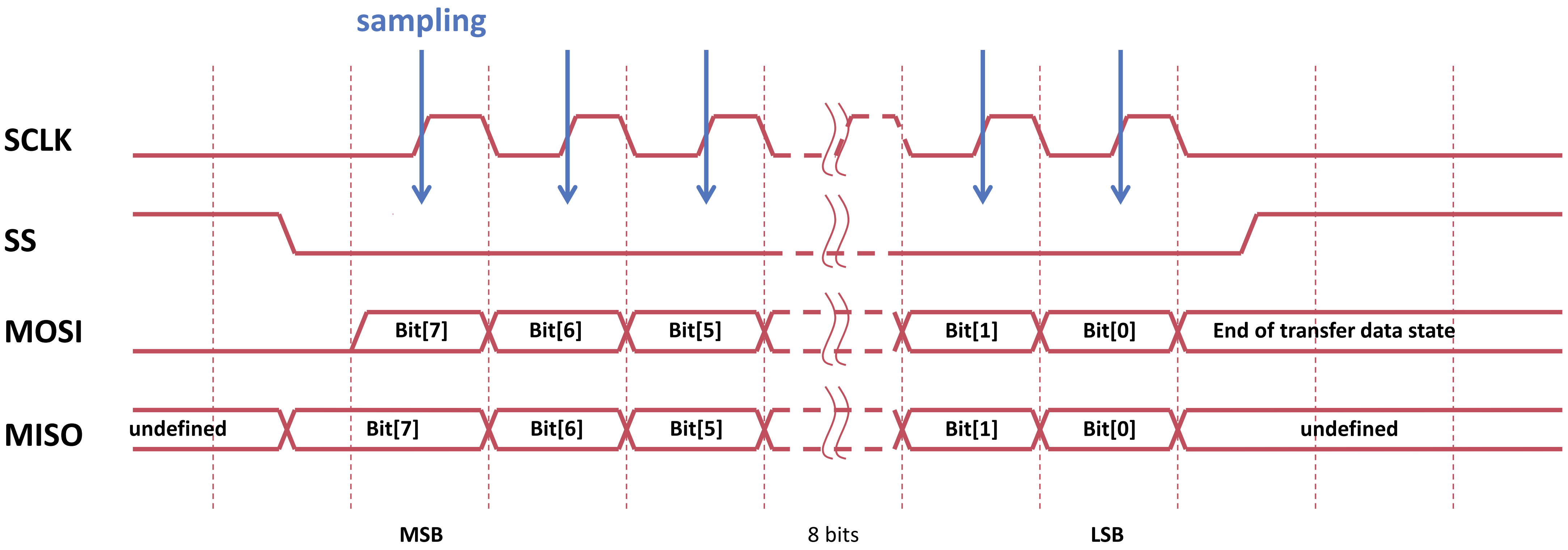
MISO: Master data Input, Slave data Output

SS: Slave Select

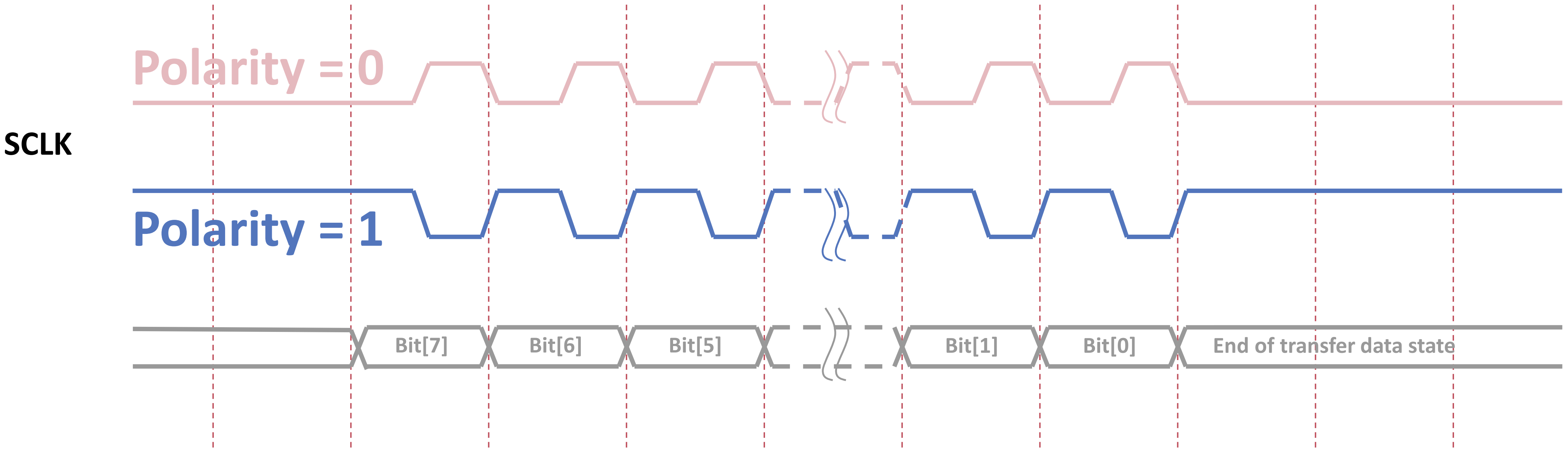
SPI

- Four wire serial bus invented / named by Motorola
- Serial connection between two or more devices (microprocessors, D/A converters)
- Configurations
  - 1 Master, 1 Slave (single slave mode)
  - 1 Master, N Slaves (multiple slave mode)
- Synchronous bidirectional data transfer
- Data transfer initiated by Master
- Bandwidth some KBits/s up to several MBits/s
- Simple implementation in software
- Used in a variety of devices, such as memory (flash, EEPROM), LCD displays and in all MMC / SD cards

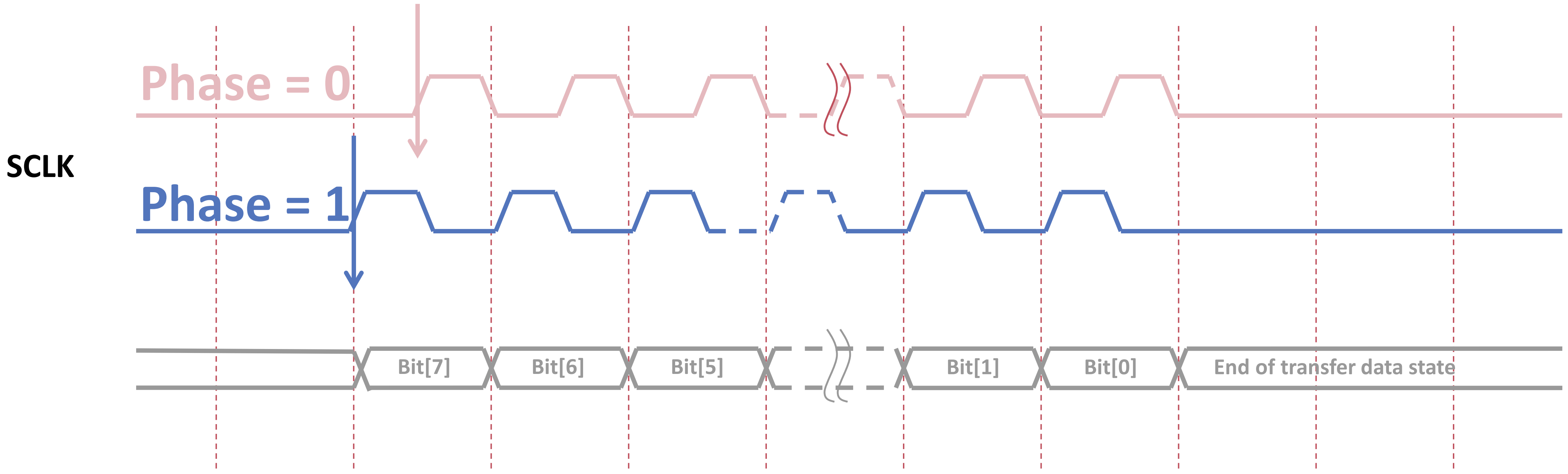
Communication



Polarity


Polarity = 0
SCLK
Polarity = 1
Bit[7]
Bit[6]
Bit[5]
Bit[1]
Bit[0]
End of transfer data state

Phase


Phase = 0
SCLK
Phase = 1
Bit[7]
Bit[6]
Bit[5]
Bit[1]
Bit[0]
End of transfer data state

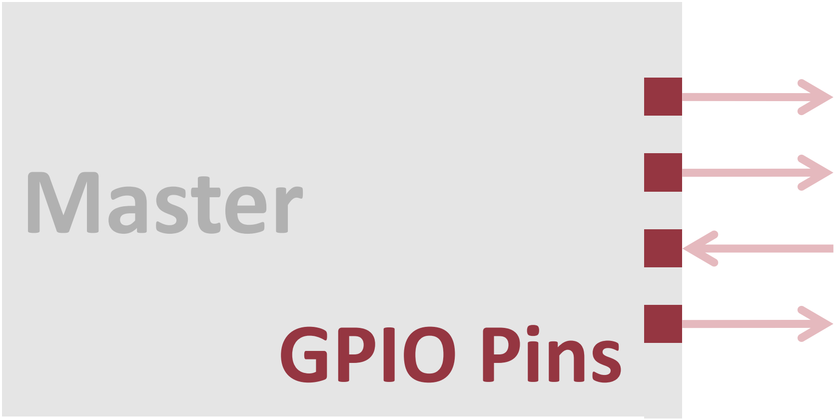
SPI – Data Transfer

- Master configures the clock
- Master selects slave (SS), followed by waiting period (if required by slave)
- Full duplex data transmission in each cycle
  - Master sends bit over MOSI line, slave reads bit
  - Slave sends bit over MISO line, master reads bit
- Two shift registers, one in slave, one in master for transfer
- When no data is to be transmitted any more, master stops toggling the clock

- **No acknowledgement mechanism**
- **No device interrupts**

Programming SPI

1. Bit-Banging



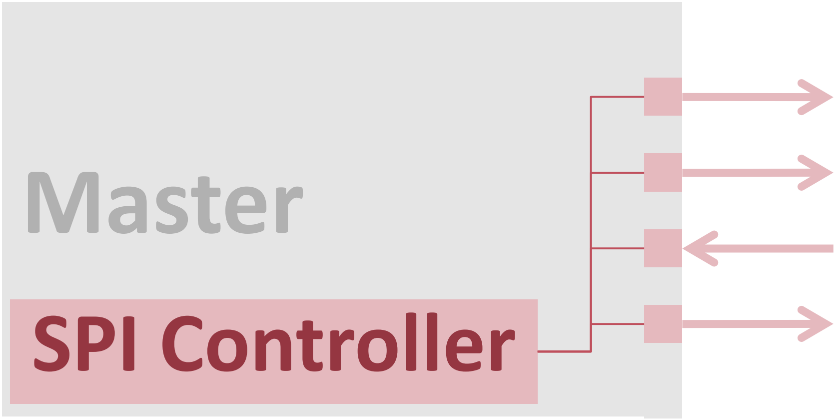
Programming SPI

1. Bit-Banging

```
FOR i := 7 TO 0 BY -1 DO
 IF ODD(ASH(data,-i)) THEN
  Platform.WriteBits(Platform.GPSET0, MOSI);
 ELSE
  Platform.WriteBits(Platform.GPCLR0, MOSI);
 END;
 Kernel.MicroWait(HalfClock);
 Platform.WriteBits(Platform.GPSET0, CLOCK);
 Kernel.MicroWait(HalfClock);
 Platform.WriteBits(Platform.GPCLR0, CLOCK);
END;
```

Programming SPI

2. Using a Controller



Programming SPI

2. Using a Controller

```
(* start transition *)
Platform.SetBits(Platform.SPI_CS, {TA});

REPEAT UNTIL TXD IN Platform.ReadBits(Platform.SPI_CS);

Platform.WriteWord(Platform.SPI_FIFO, data);
junk := Platform.ReadWord(Platform.SPI_FIFO);

REPEAT UNTIL DONE IN Platform.ReadBits(Platform.SPI_CS);

(* transfer inactive *)
Platform.ClearBits(Platform.SPI_CS, {TA});
```

BCM 2835 Registers

CS -- Control and Status
Chip Select
FIFO Status
Transfer Progress
Interrupts
Polarity & Phase

FIFO Register
Data

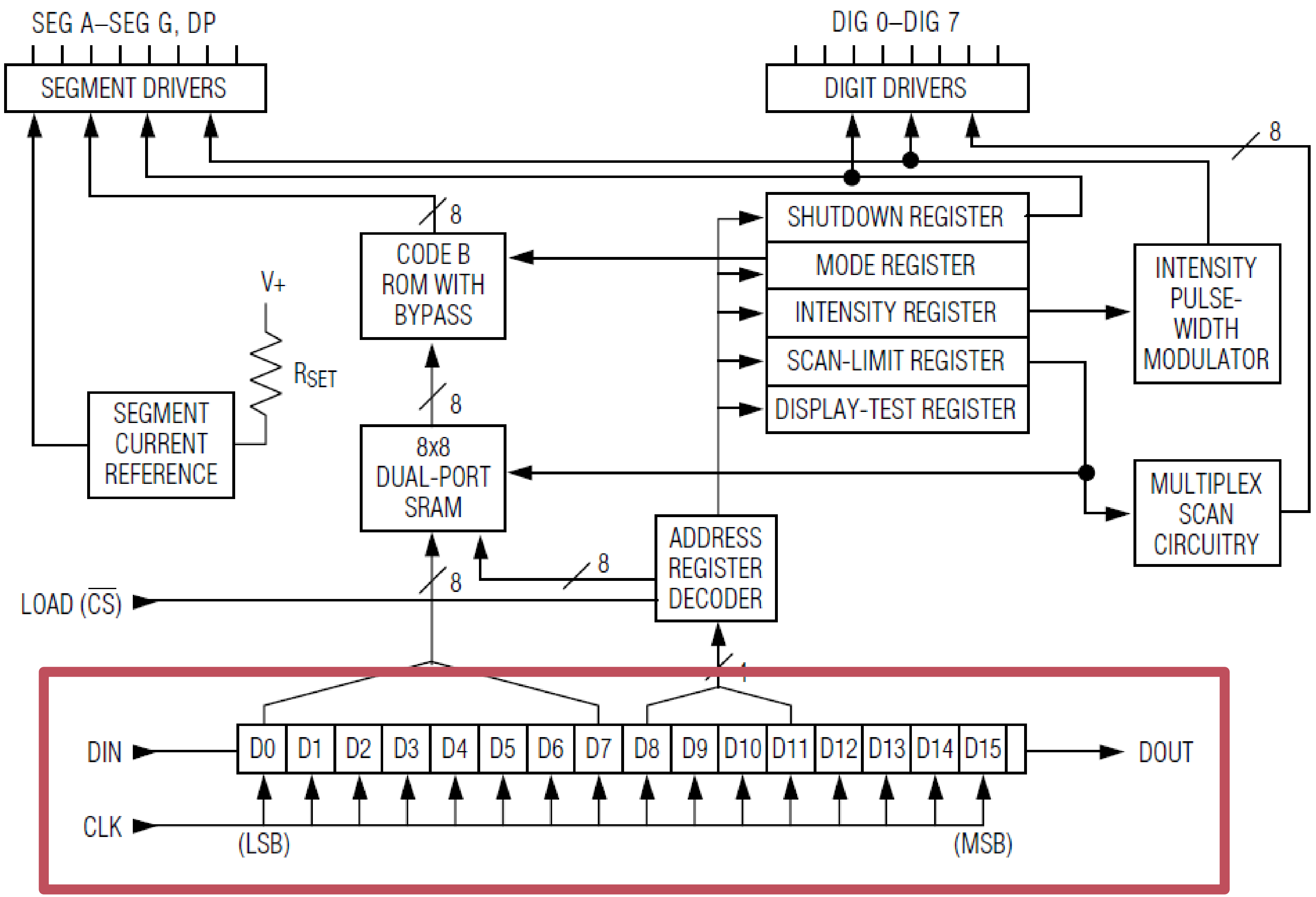
Write:
TX Fifo

Read:
RX Fifo

CLK
Clock Divider

Other
DMA Control
Special Mode Control

MAX7219 8-Digit LED Display Driver



Max7219 Specification, p.5

MAX7219 8-Digit LED Display Driver

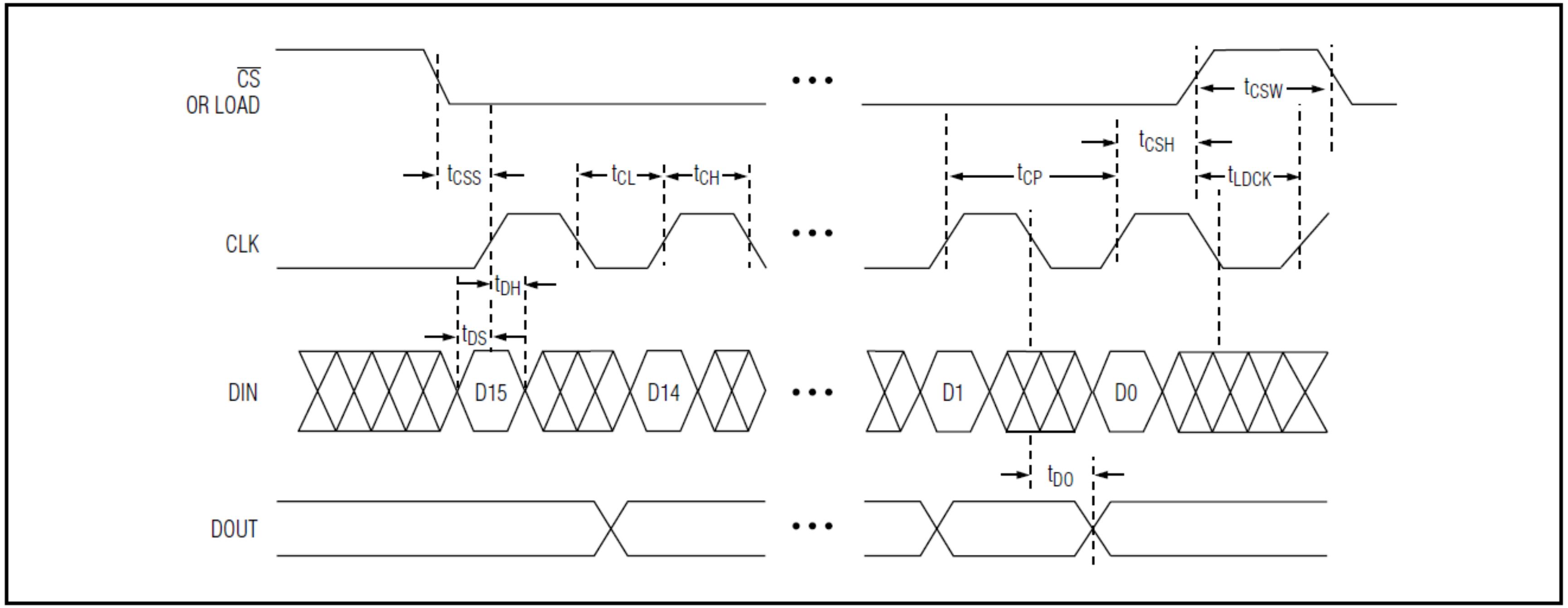


Figure 1. Timing Diagram

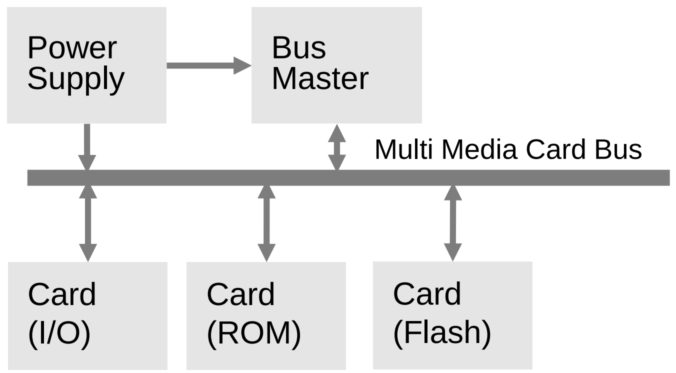
Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB			DATA				LSB

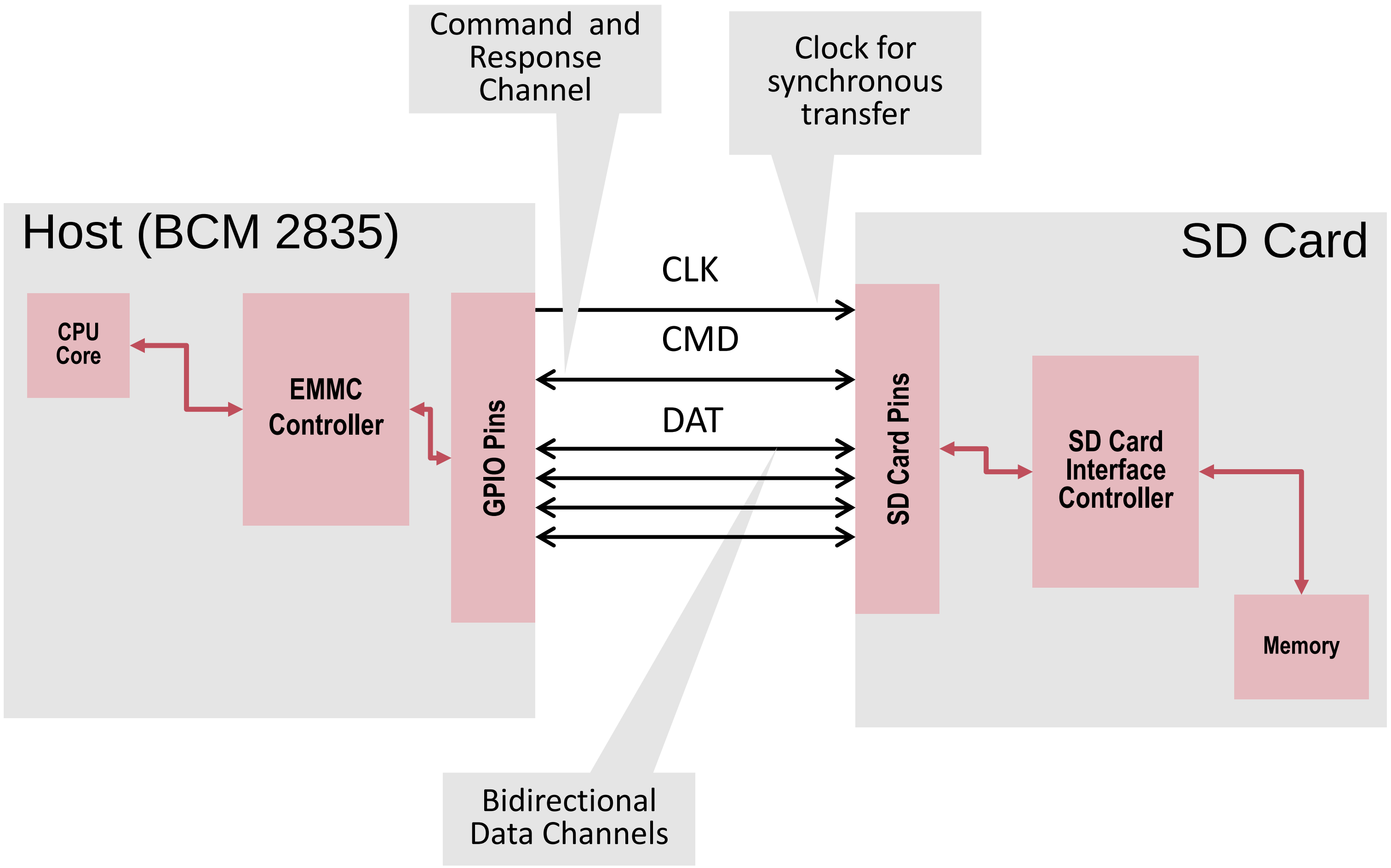
Max7219 Specification, p.6

MMC and SD Cards

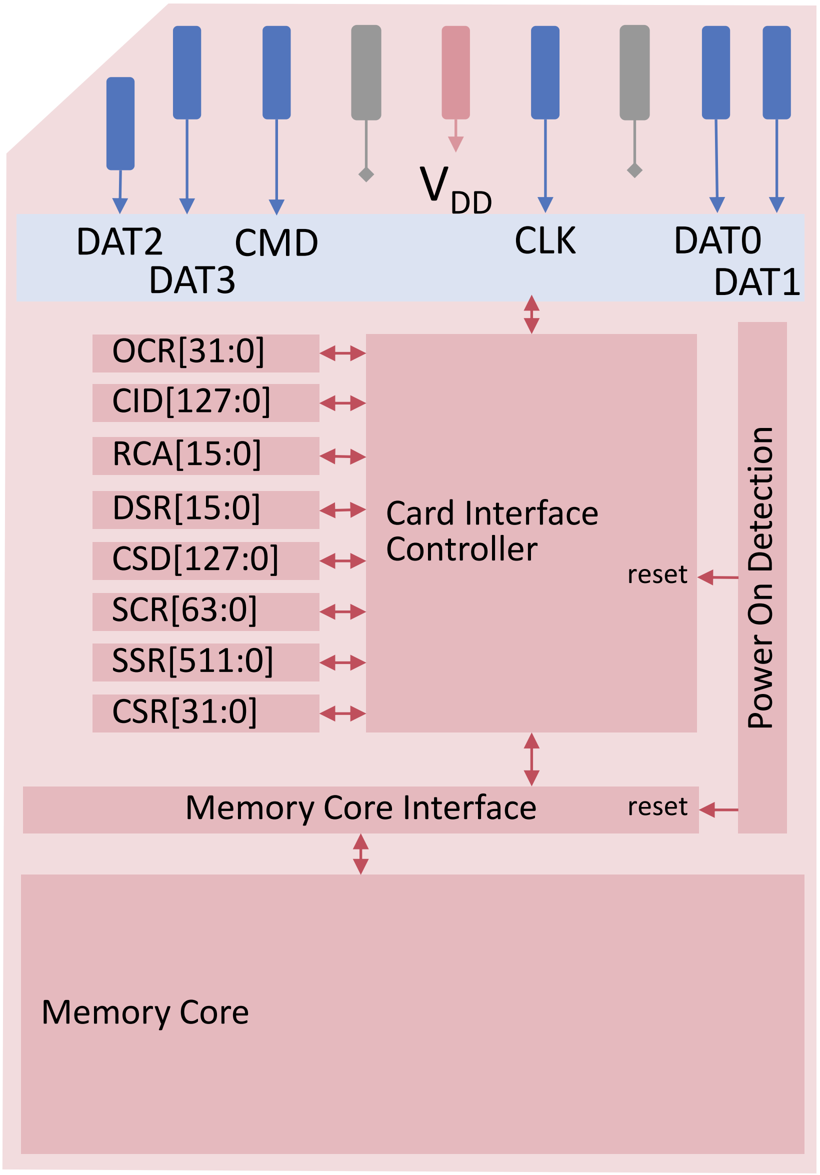
- Low cost memory system for persistent data on „solid state mass storage“ (for example flash memory cards)
- Separate bus system
  - 1 master, N slaves (cards)
  - typically 1 master for one card
- Serial & synchronous transfer of commands and data
  - Sequential read/ write
  - Block read/ write



MMC System Interaction

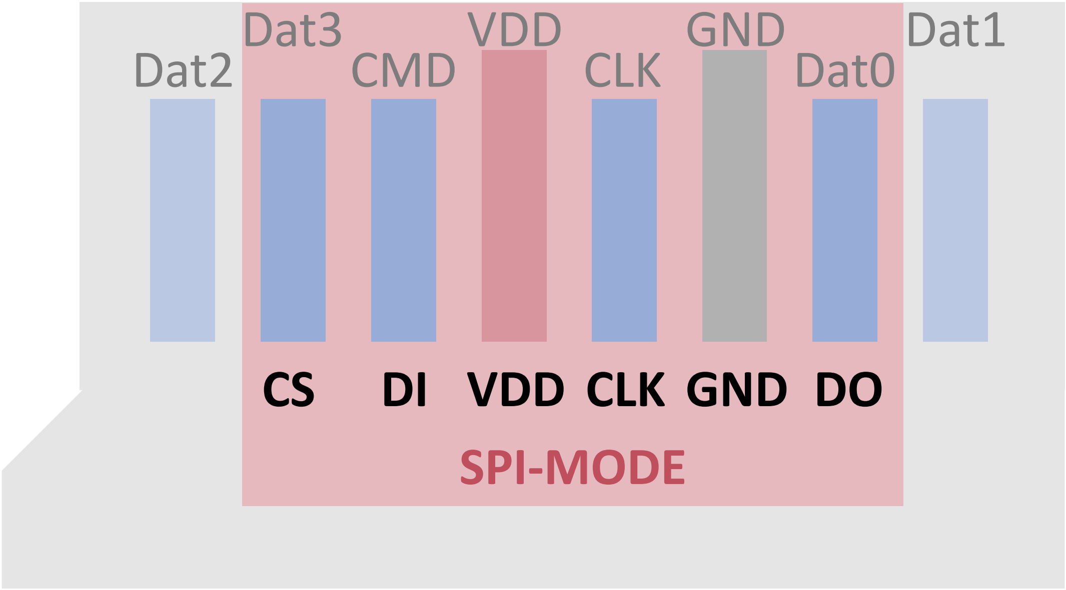


SD Card



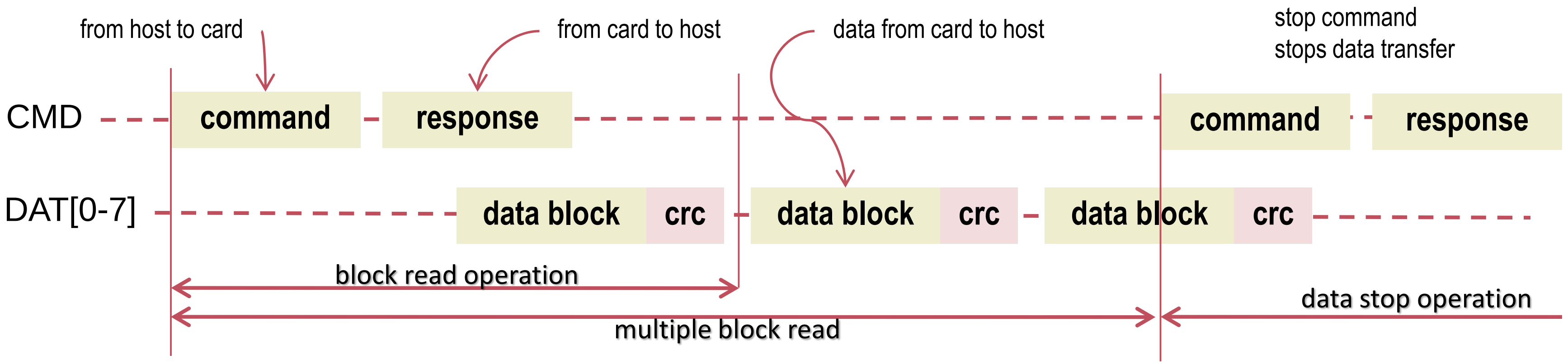
SD Physical Layer Spec. P. 12

SD Mode vs SPI Mode

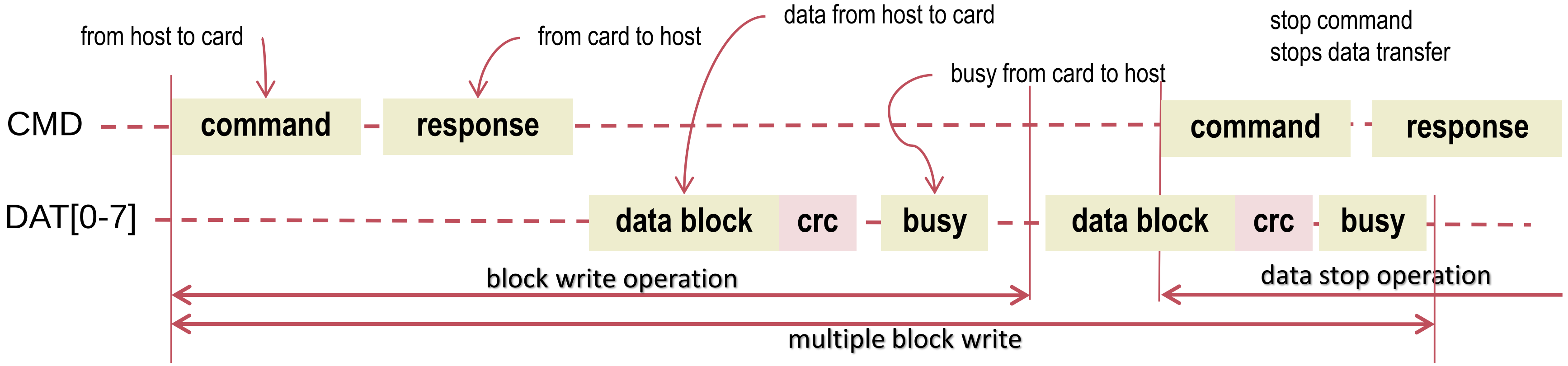


Block Read/ Write Operation

- Read



- Write

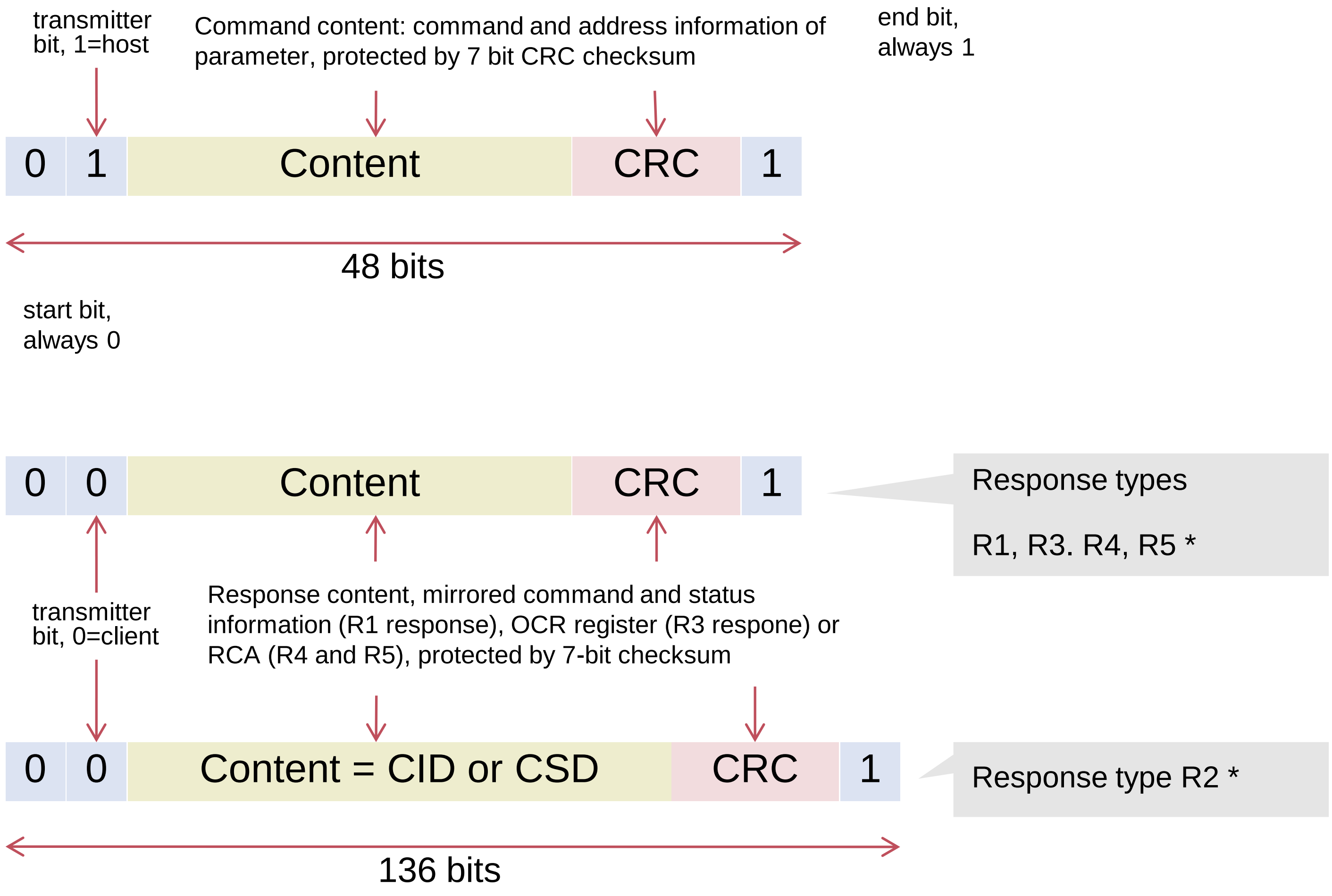


Packet Formats

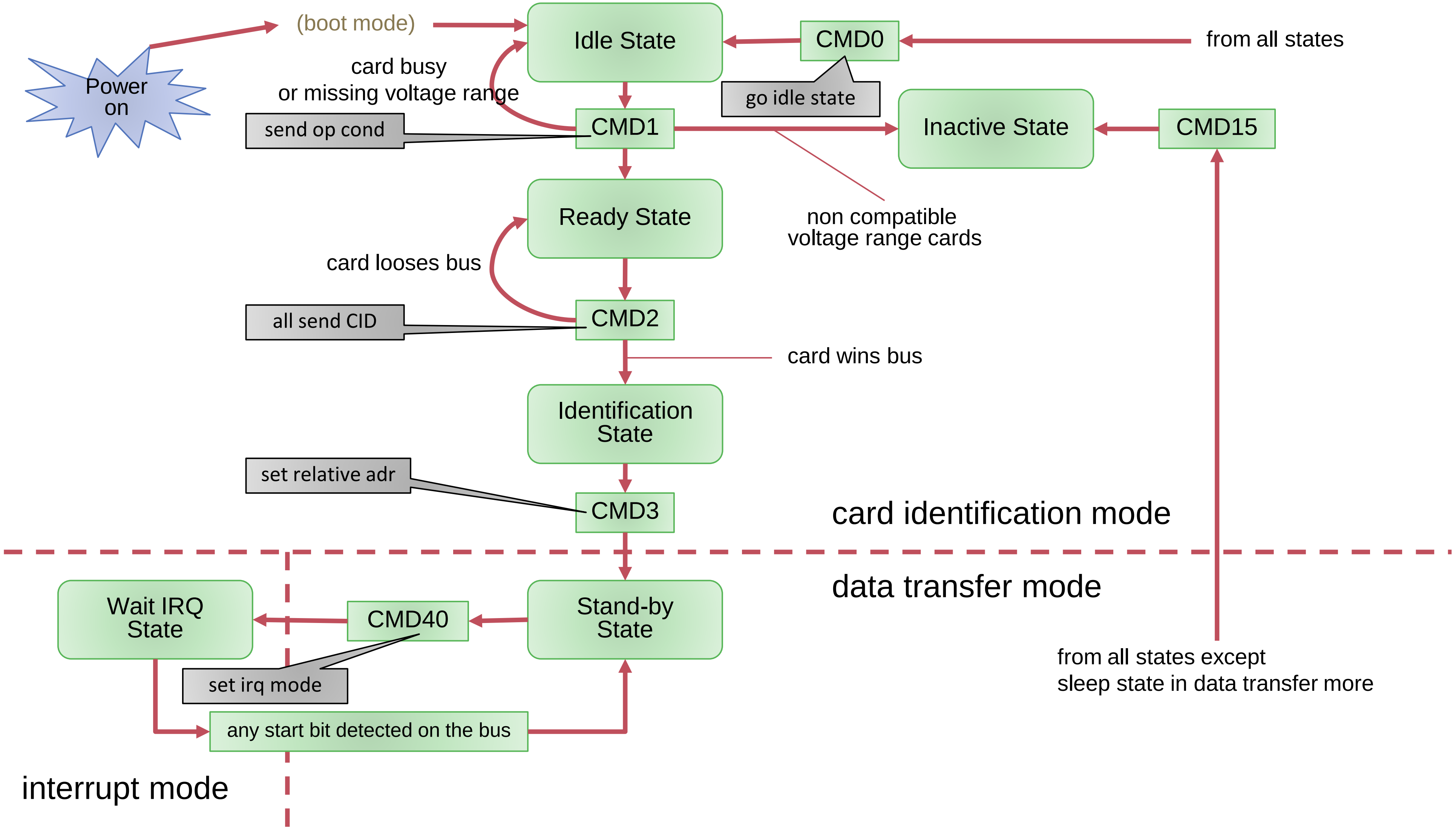
Command and Response

- Command Token

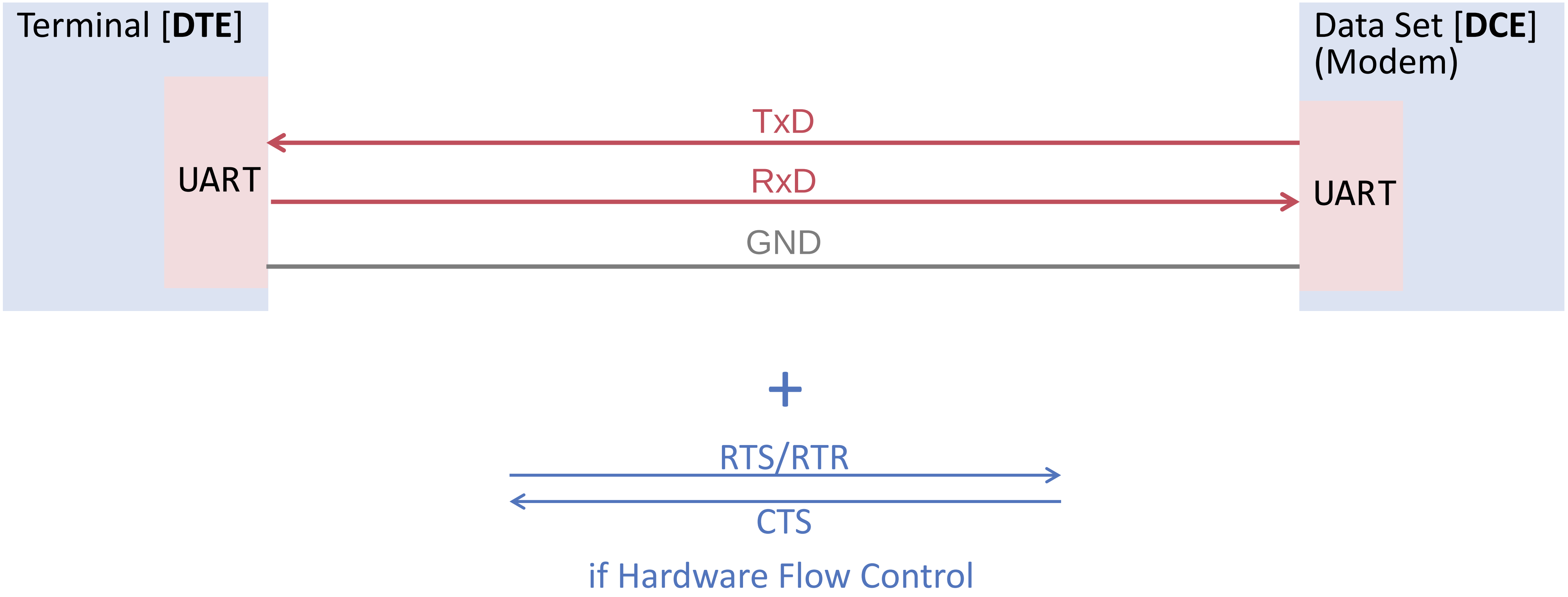
- Response Tokens



Example: MMC Memory Card State Diagram

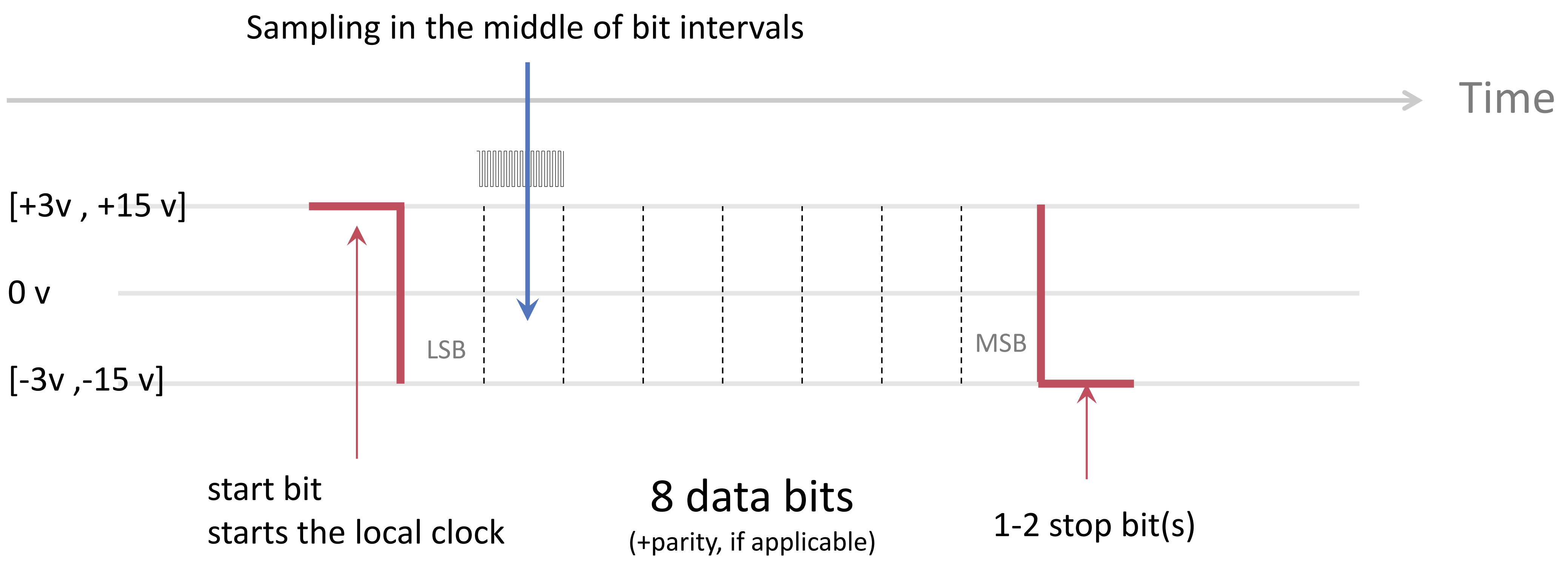


RS232



Terminal [DTE]
UART
Data Set [DCE]
(Modem)
UART
TxD
RxD
GND
+
RTS/RTR
CTS
if Hardware Flow Control

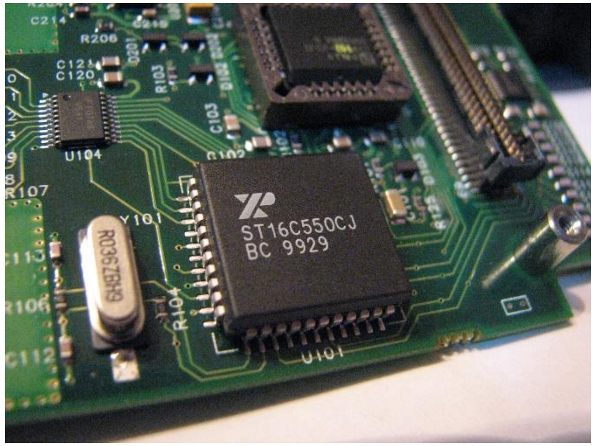
RS232 Signalling



UART

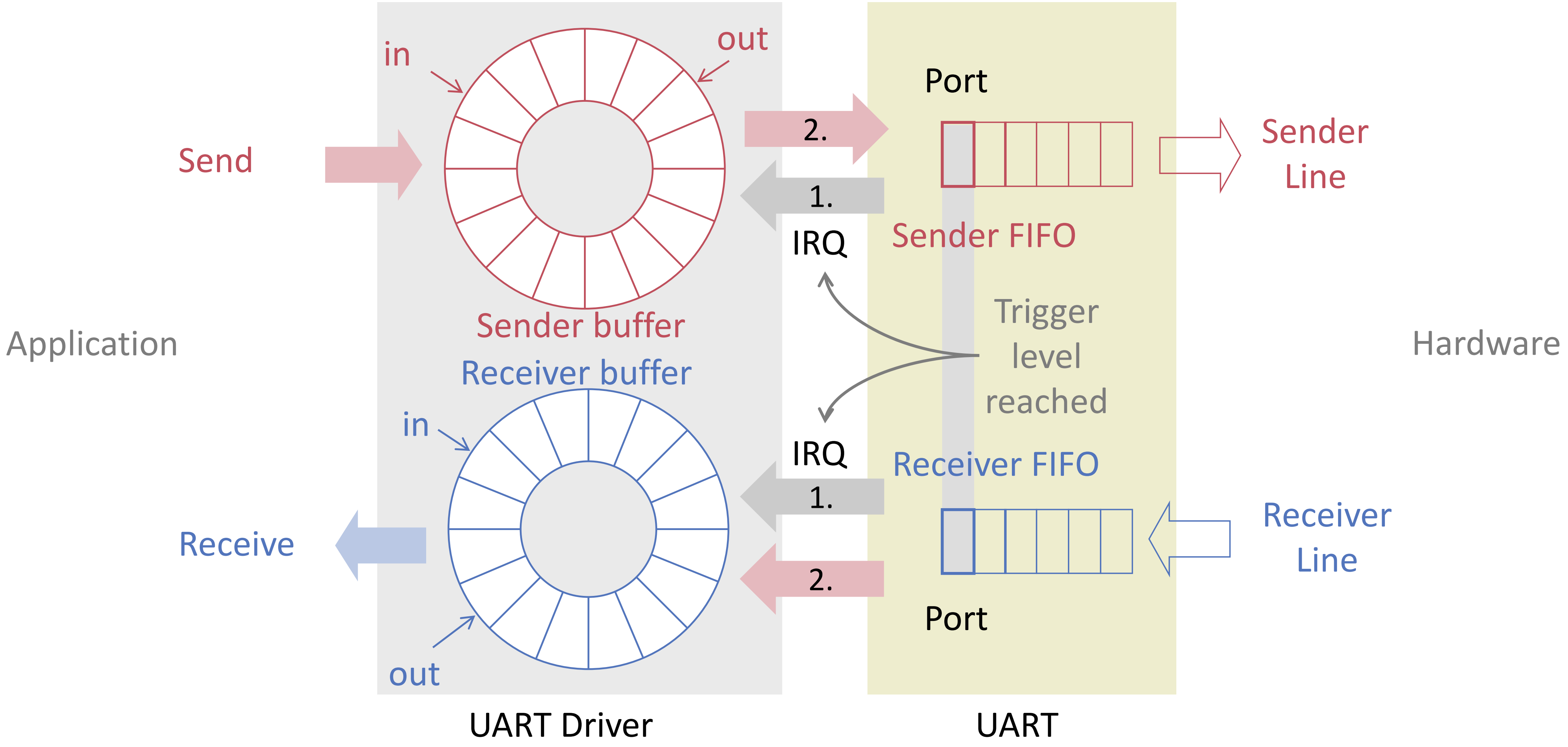
Universal Asynchronous Receiver/ Transmitter

- Serial transmission of individual bits in byte packets (lowest significant bit first)
- Configurable
  - Number of data bits per byte: 5, 6, 7, 8
  - Parity: odd, even, none
  - Number of stop bits: 1, 1.5, 2
  - Transfer rate in bps (bits per second): 75, 110, 300,... , 115200



source: Wikipedia

Implementation



Producer Consumer Implementation

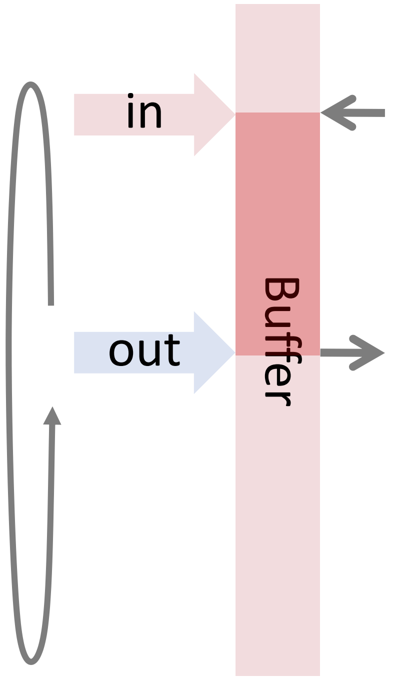
Assumption: one consumer and one producer

Producer

```
WHILE (in+1) MOD bufferSize = out DO END;
buffer[in] := produced;
in = (in+1) MOD bufferSize;
```

Consumer

```
WHILE in = out DO END;
consumed := buffer[out];
out := (out+1) MOD bufferSize;
```



Driver

- Method *Send*
  - Put data in sender buffer;
    Update *in (sender)*
- Method *Receive*
  - Get data from receiver buffer;
    Update *out (receiver)*
- Sender-Interrupt
  - Shift data from sender buffer to sender FIFO;
    Update *out* (sender)
- Receiver Interrupt
  - Shift data from receiver FIFO to receiver buffer;
    Update *in (receiver)*

not covered in class

1.6. FILE SYSTEM

Modular Structure

not covered in class

Files

Sequential files
Files as byte sequences
Riders for reading/writing
Buffering

Flat name space
B-tree representation
node ⇔ block

FileDir

BlockDevice

Block Sequences
Block allocation
Read/ write block

API

not covered in class

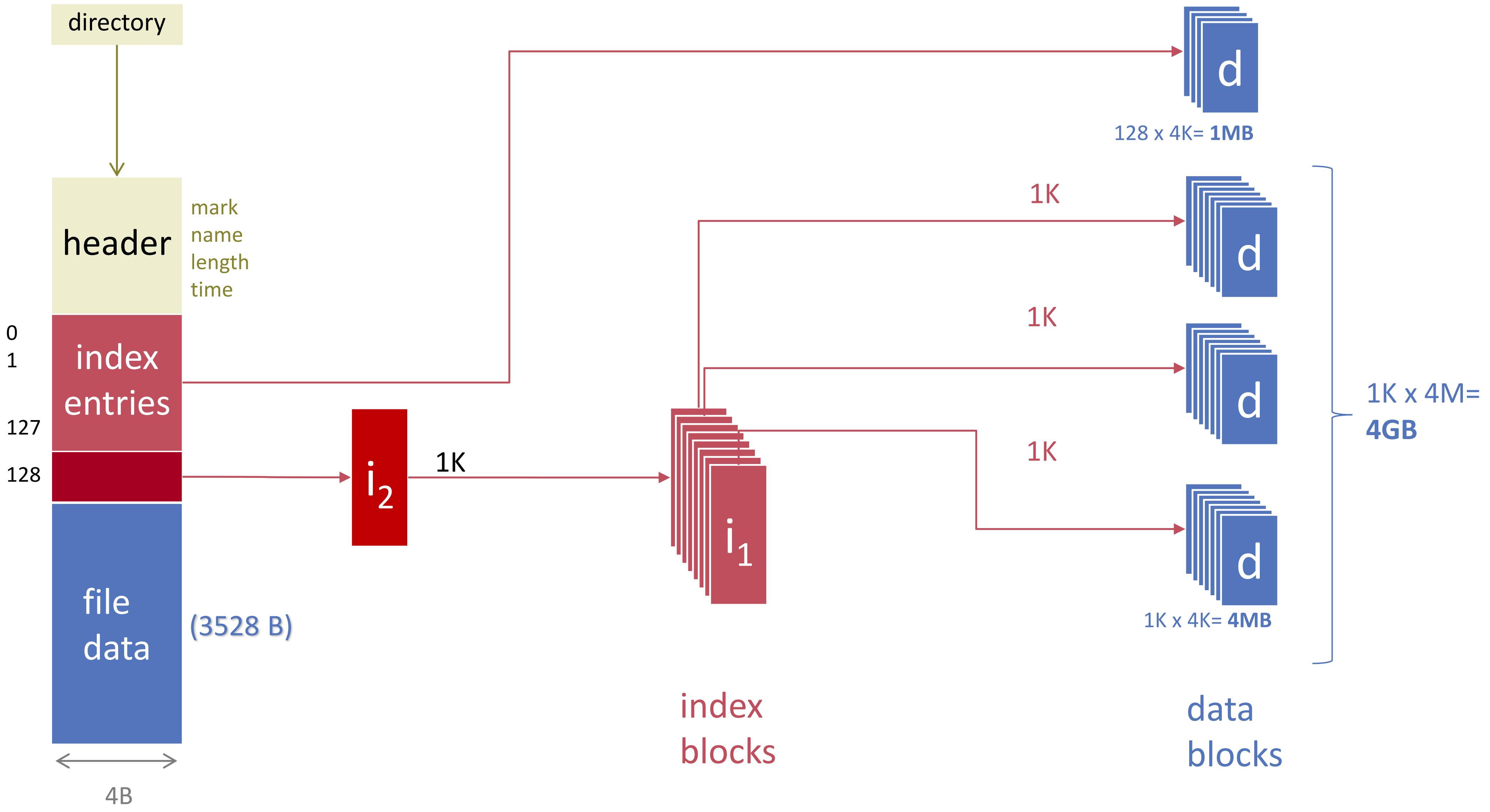
- Abstract data types *File, Rider*
- Open file (new or via name)
- Close file
- Position rider in file
- Read next byte via rider
- Write next byte via rider

```
File* = POINTER TO FileDesc;
FileDesc* = RECORD … END;

Rider* = RECORD
   eof*: BOOLEAN;
   …
   hint*: Buffer;
   file*: File;
END;
```

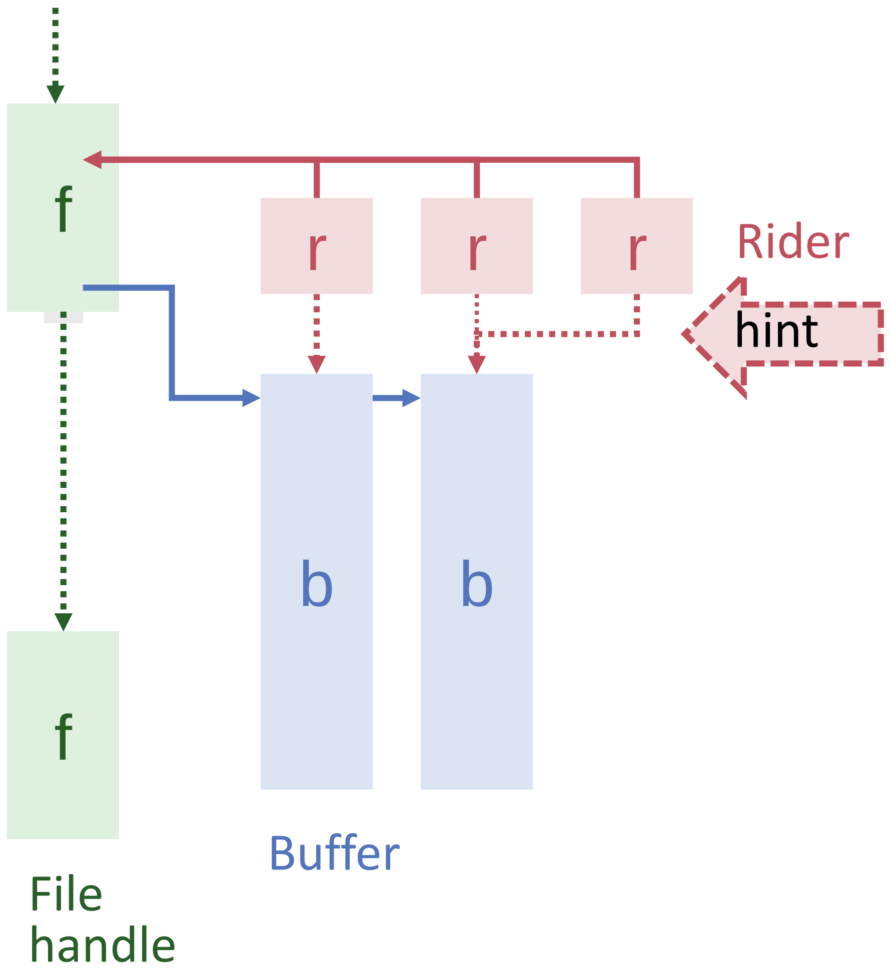
Block Structure of Files

not covered in class



Internal Data Structure

not covered in class



Rider for accessing files

- Positioning
- Sequential reading
- Sequential writing

Buffer

caching pages around the current focus to minimize disk accesses

Read from Buffered Rider

not covered in class

```
PROCEDURE Read*(VAR r: Rider; VAR x: CHAR);
VAR buf: Buffer; f: File;
BEGIN
 buf := r.hint(Buffer); f := r.file;
 IF r.apos # buf.apos THEN
  buf := GetBuf(f, r.apos); r.hint := buf
 END;
 IF r.bpos < buf.lim THEN
  x := buf.data.B[r.bpos]; INC(r.bpos)
 ELSIF r.apos < f.aleng THEN
  Search buffer in file buffers.
  If no buffer at r.apos then use r.hint, flush if modified and read
   x := buf.data.B[0]; r.bpos := 1
  ELSE x := 0X; r.eof := TRUE
 END
END Read;
```

Block Allocation Table

not covered in class