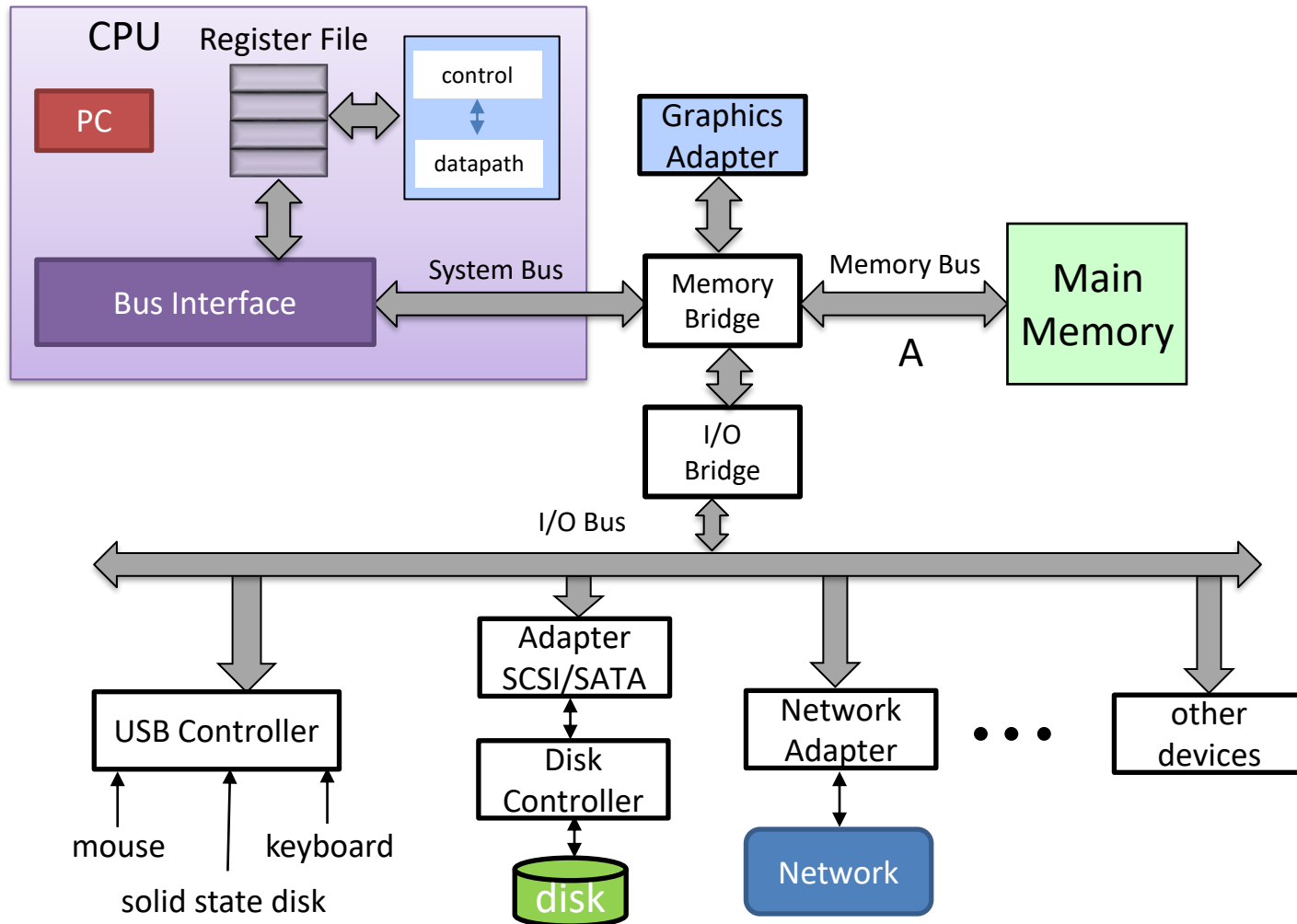
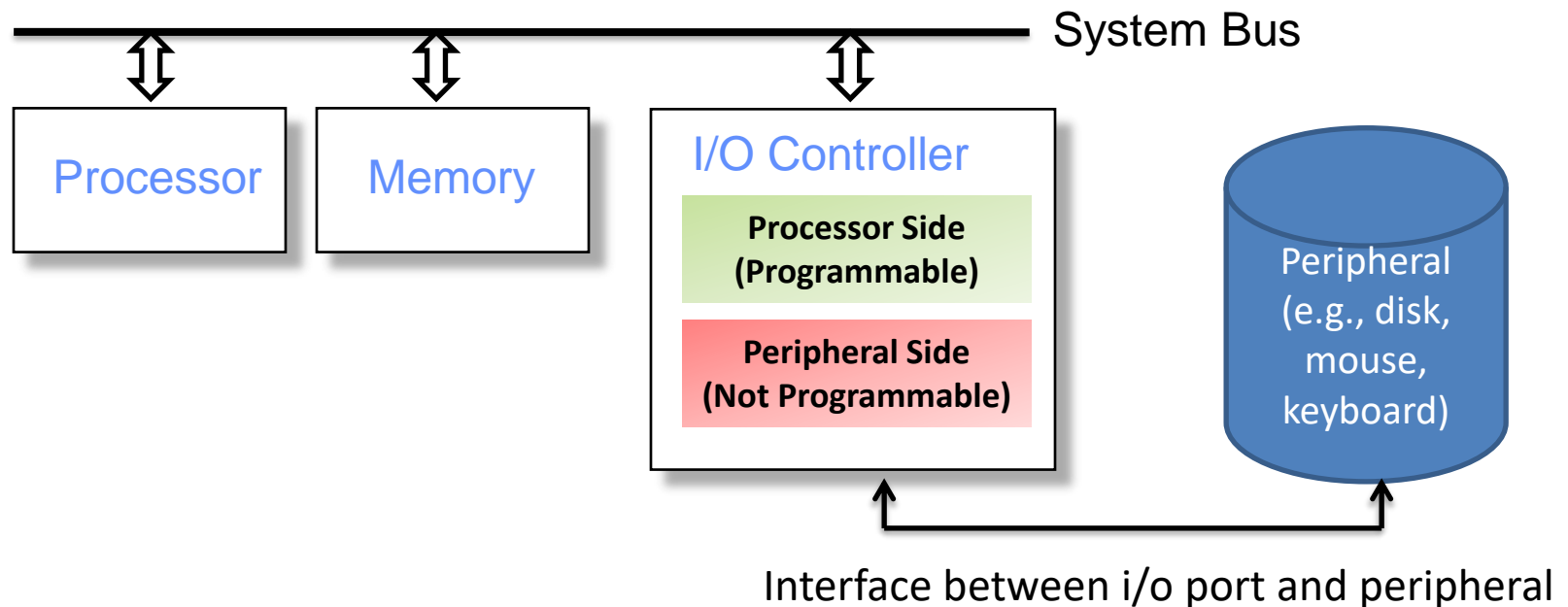


Logical View of a Traditional Computer



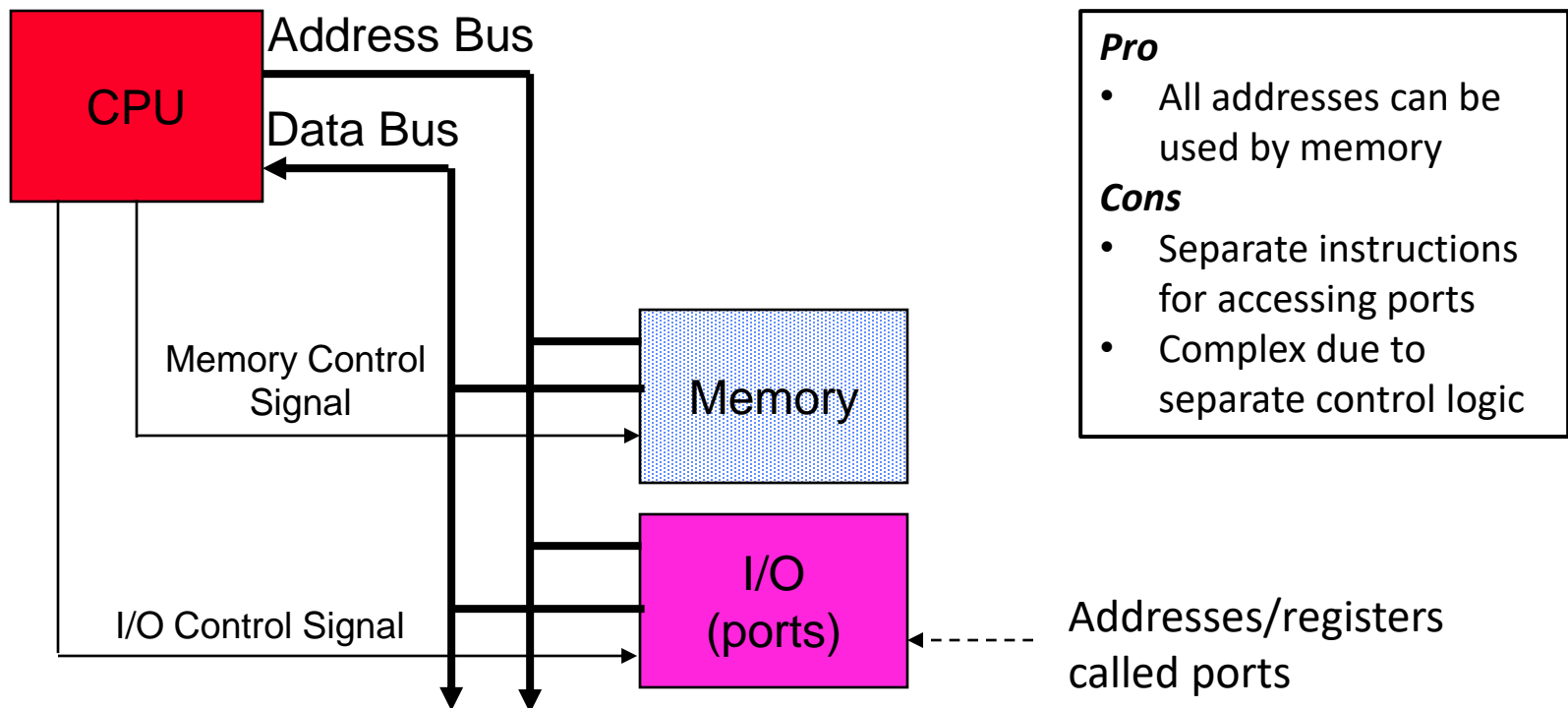
Peripherals and I/O Device Controllers

- Controller – interface between I/O device and system bus
- Contains registers, and possibly memory, processor, clock
- Organized into two halves
 - Processor – device driver is part of OS
 - Peripheral – external to computer



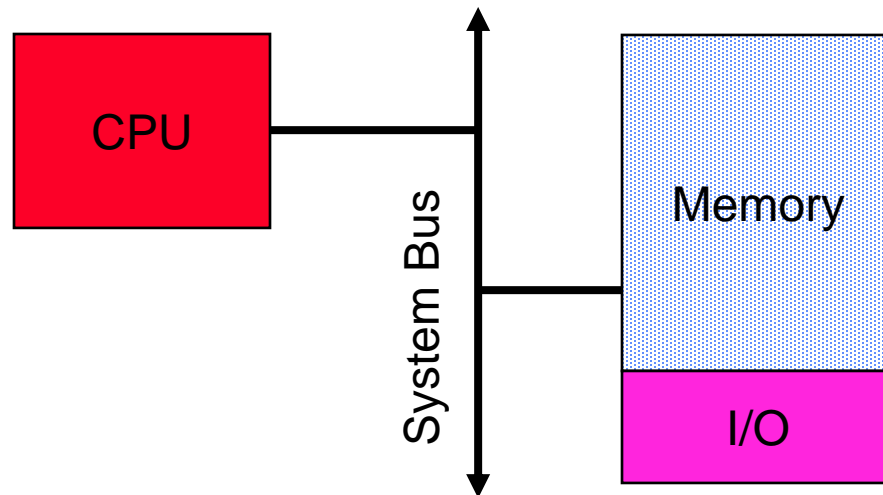
Isolated I/O

- Shared address and data bus, separate control buses for I/O and memory

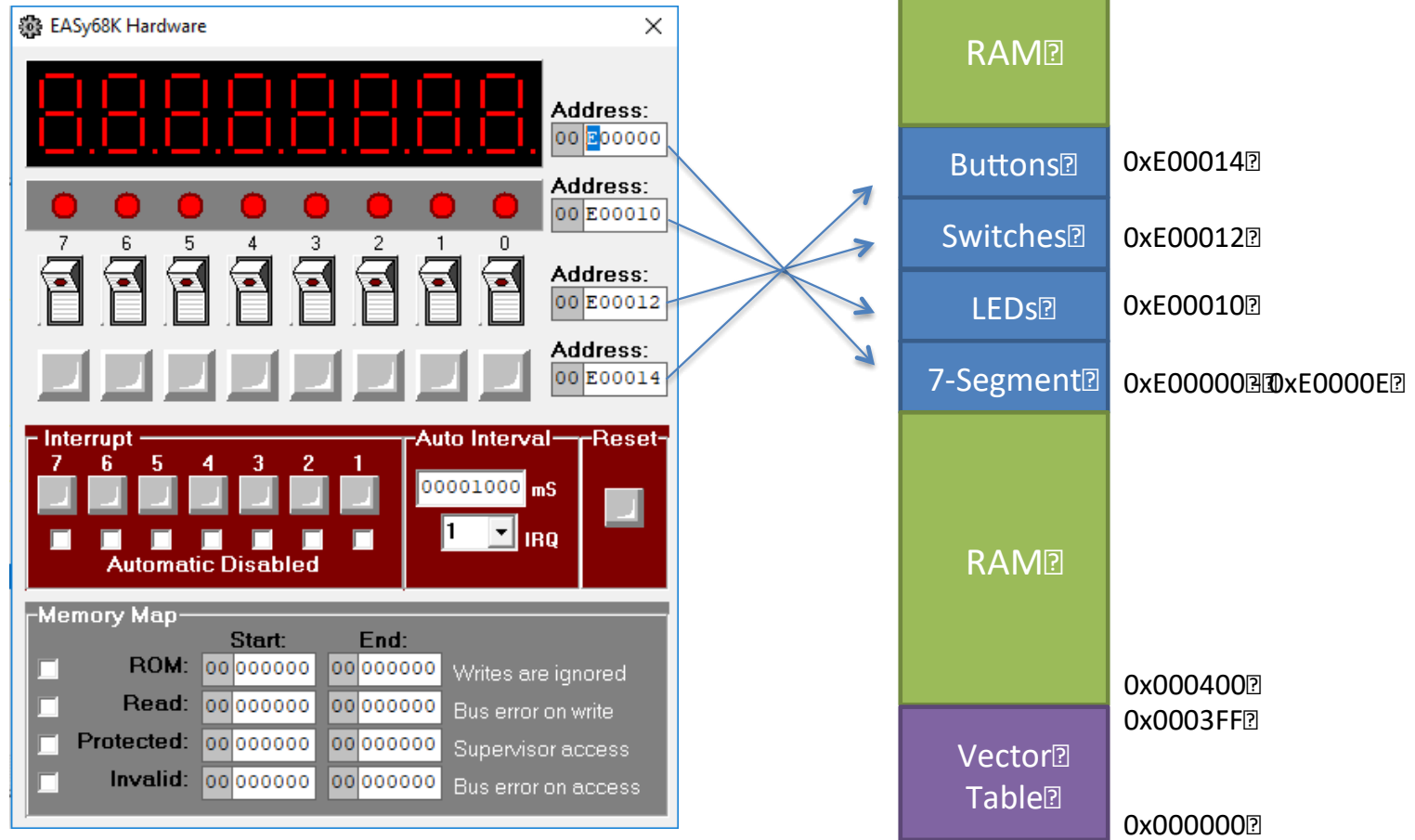


Memory-Mapped I/O

- Memory and I/O devices share the same system *bus* and same *address space*
- Data-transfer instructions can be used to move data to and from I/O device registers
 - A **load** operation moves data from an I/O device to a CPU register
 - A **store** operation moves data from a CPU register to an I/O device register

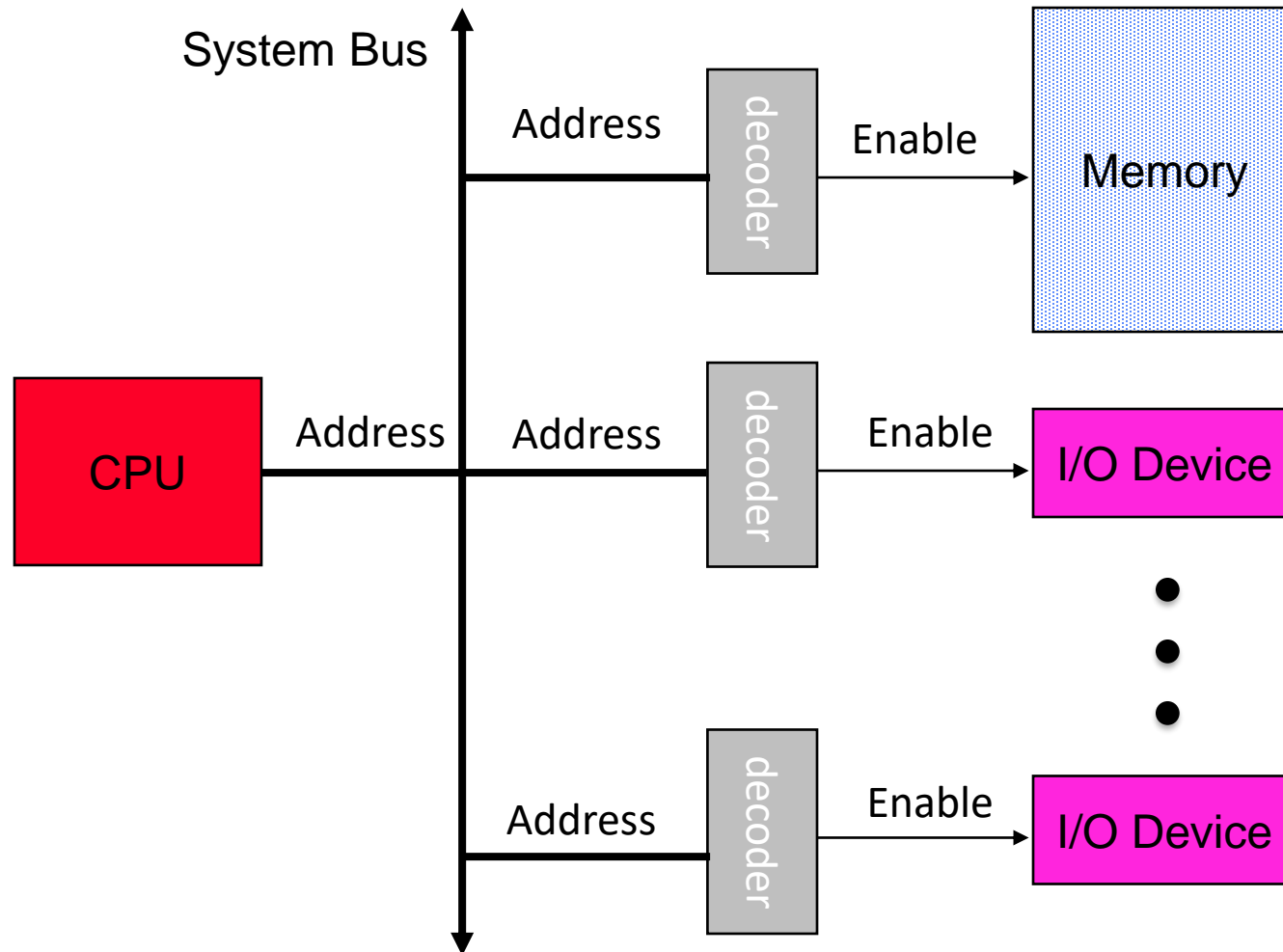


Example of Memory-Mapped I/O from Easy68K

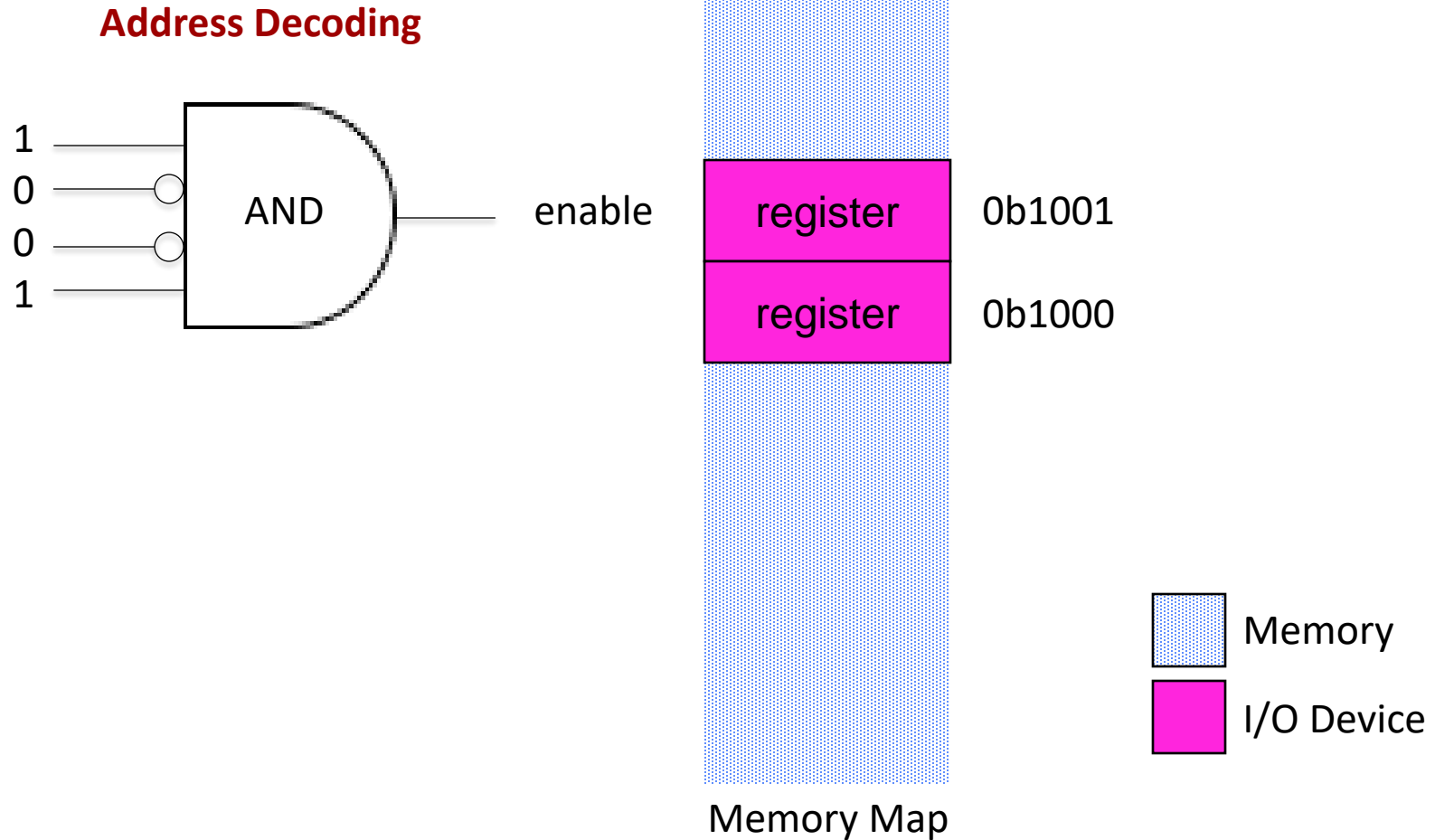


Hardware Devices Memory Map

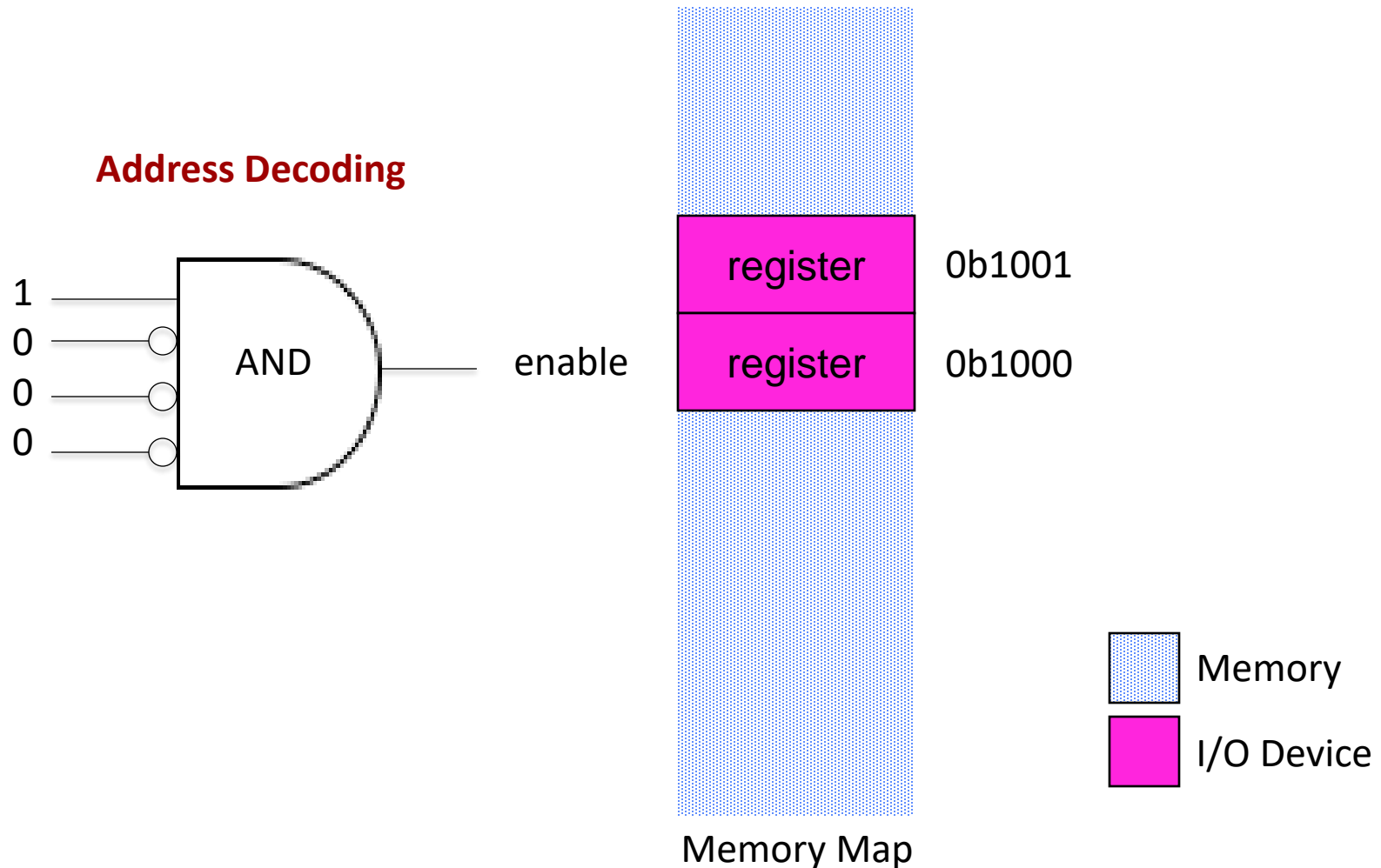
Mapping Registers into Addresses Requires Logic



Mapping Registers into Addresses Requires Logic

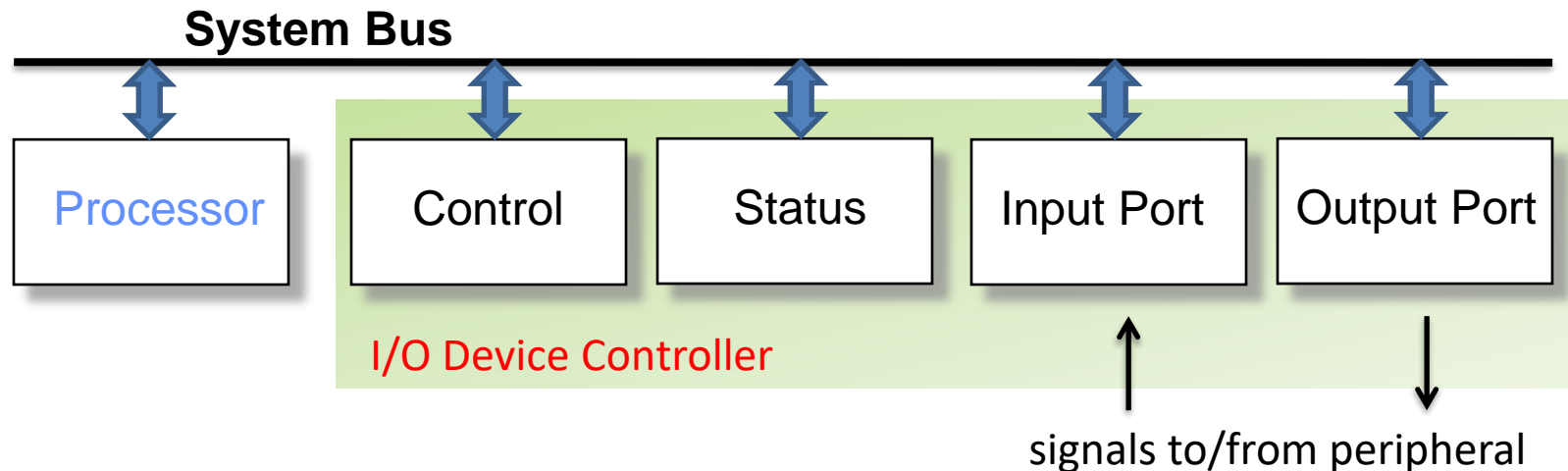


Mapping Registers into Addresses Requires Logic



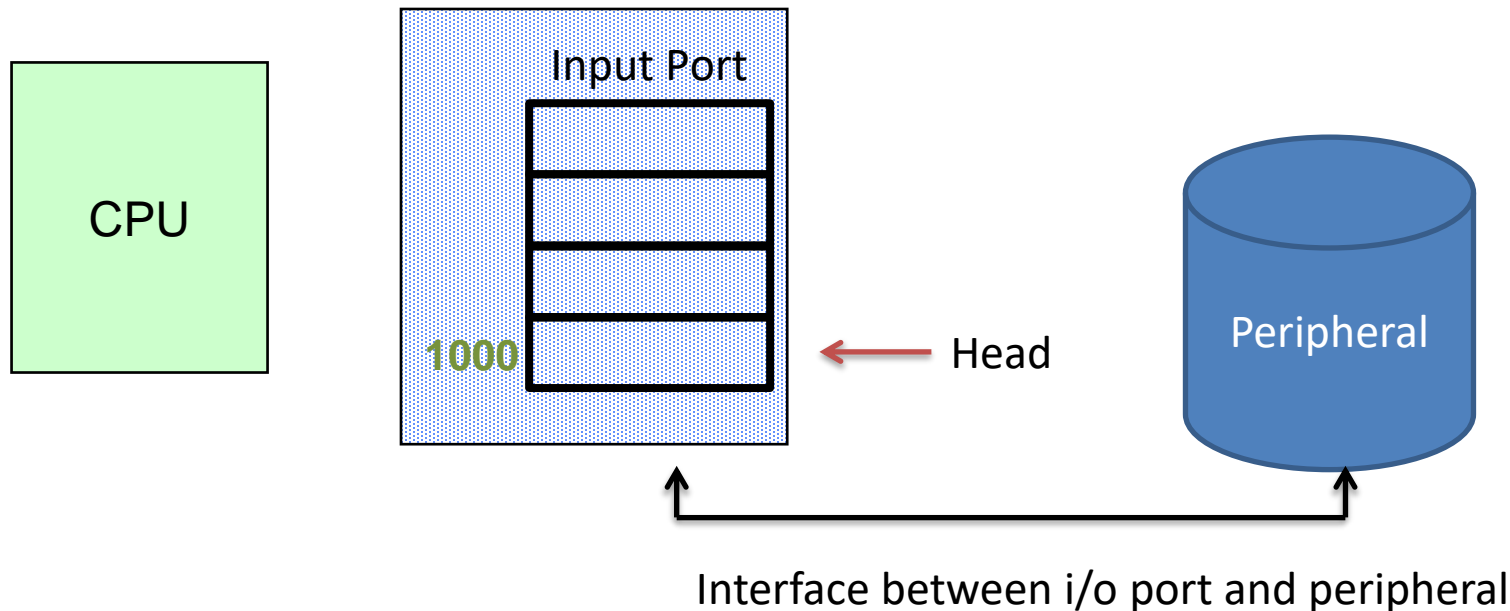
Role of Registers in I/O Controller

- Control
 - Programmed (by device driver in OS) to control the operation of the peripheral
- Status register
 - Queried (by device driver in OS) to learn things about the peripheral
- Input and Output ports
 - Temporarily hold data that has been received or is about to be transmitted



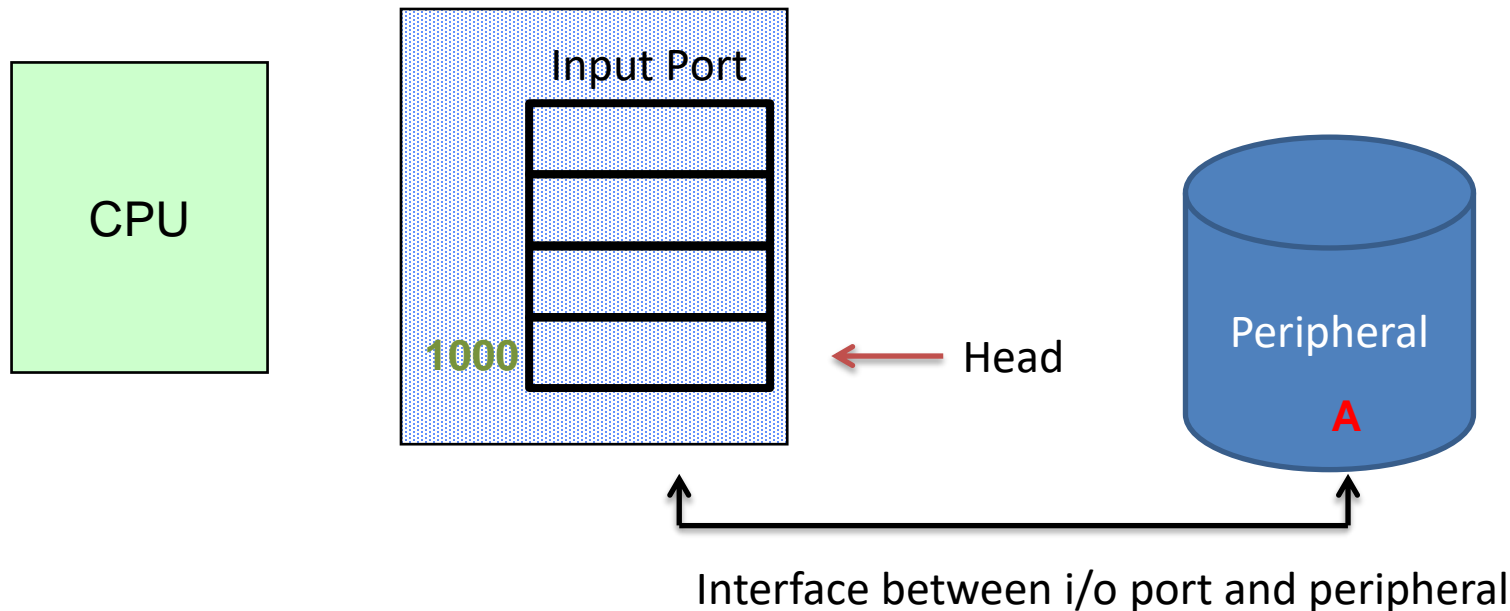
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



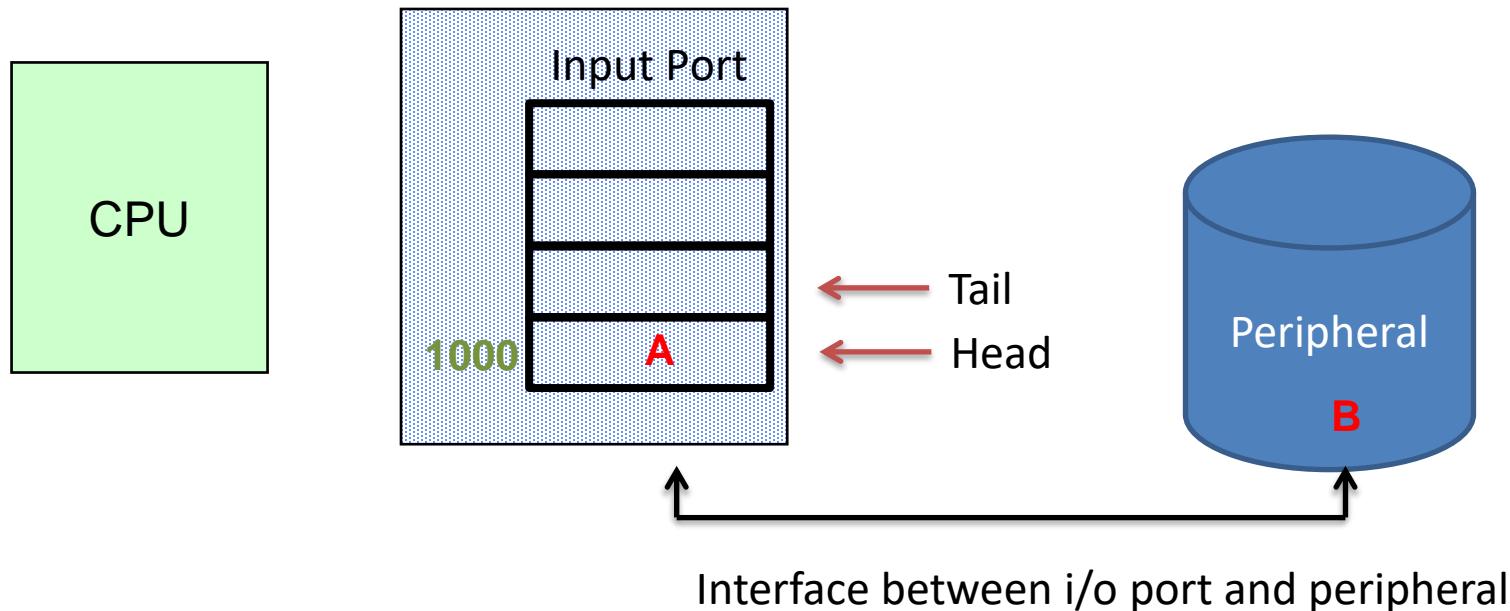
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



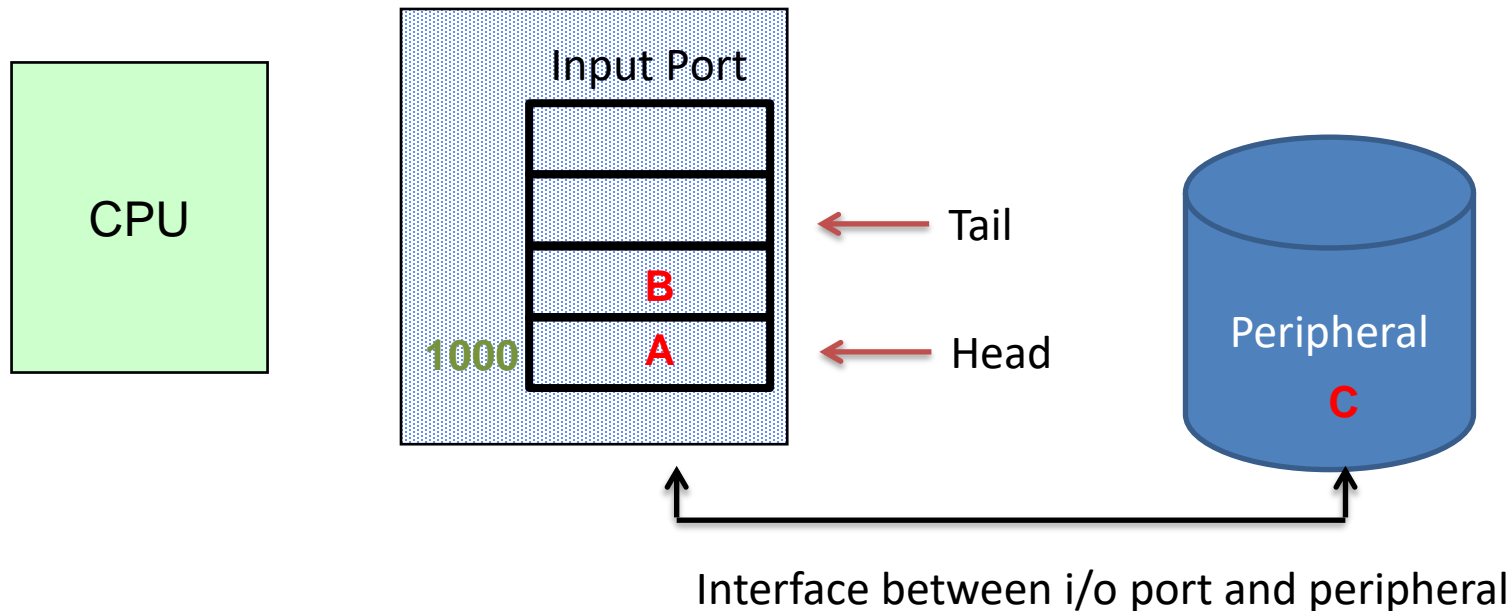
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



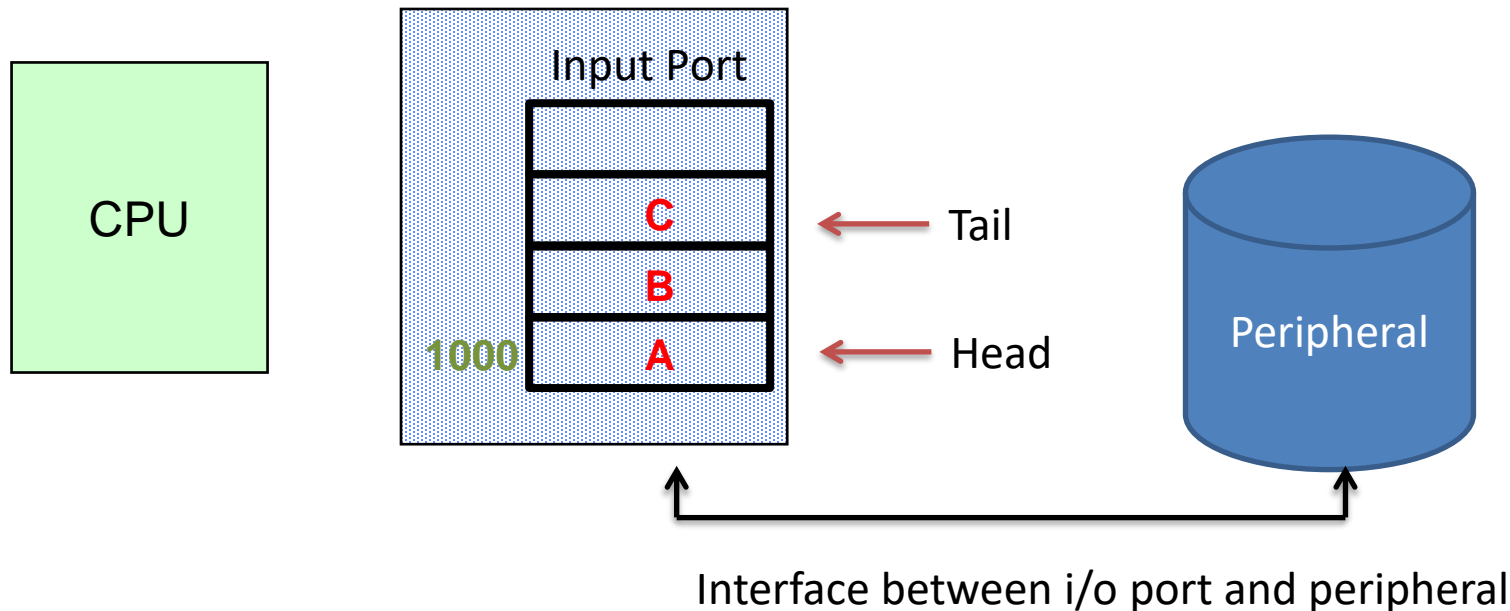
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



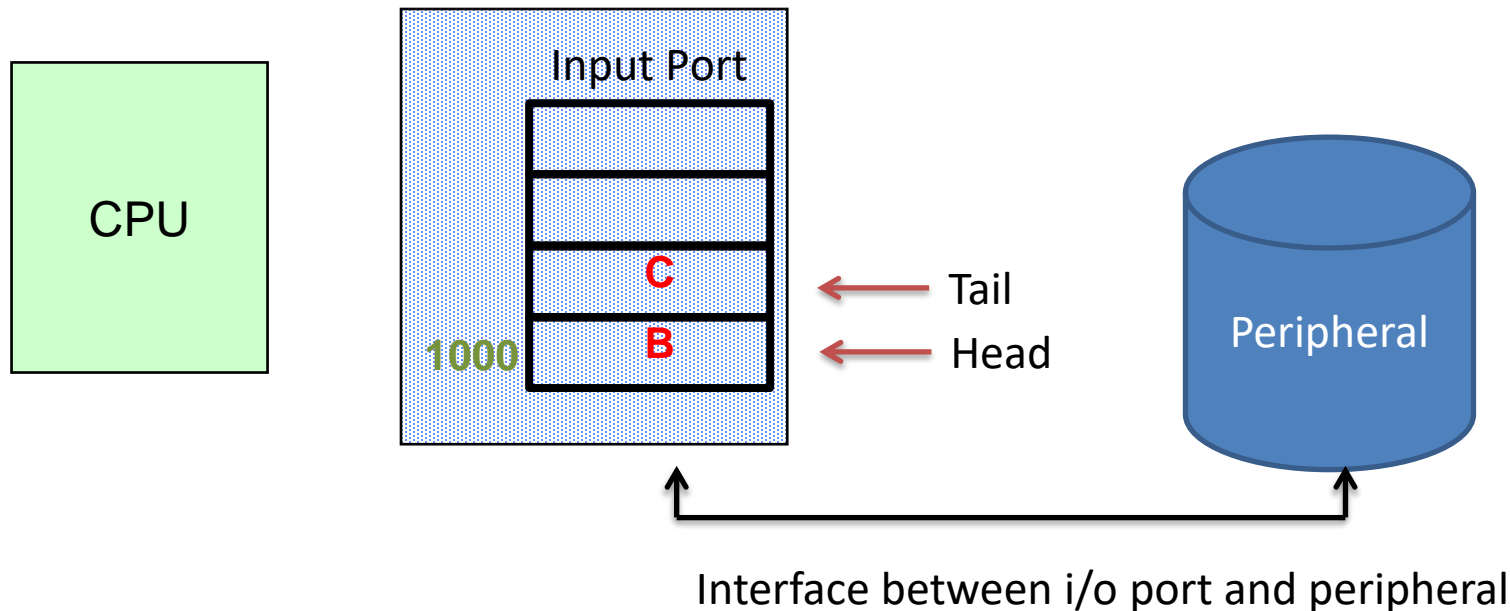
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



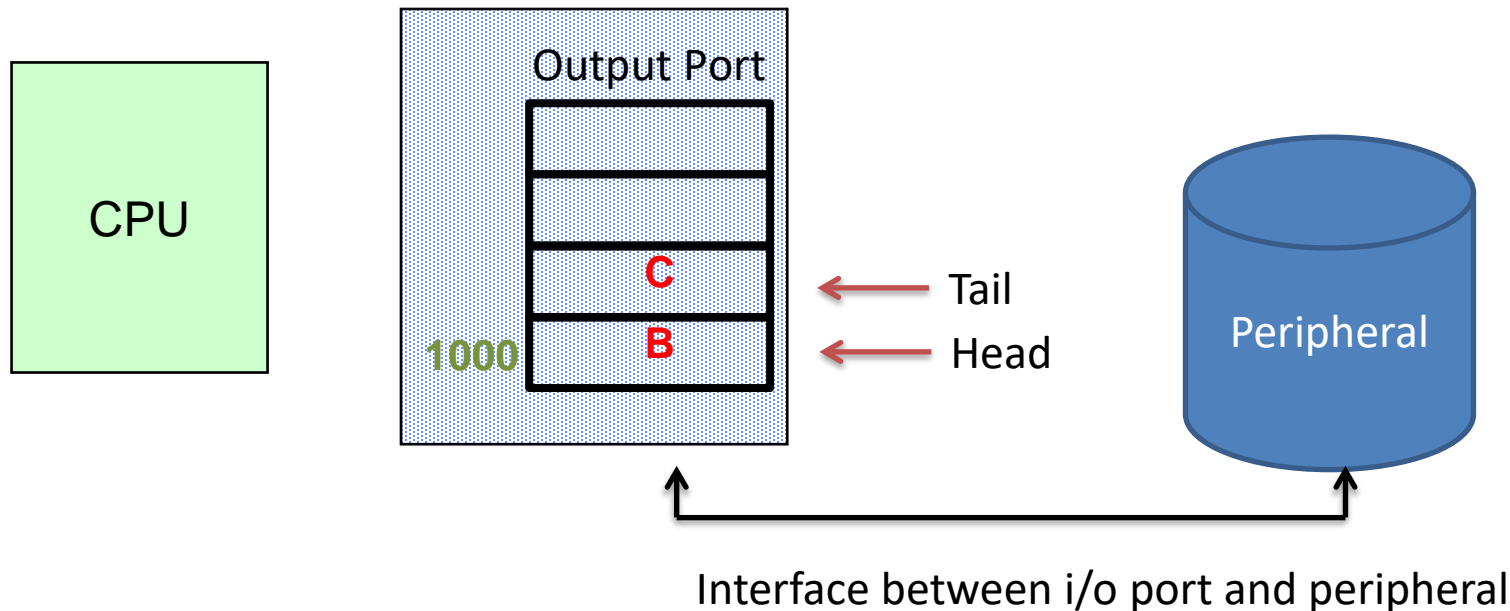
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



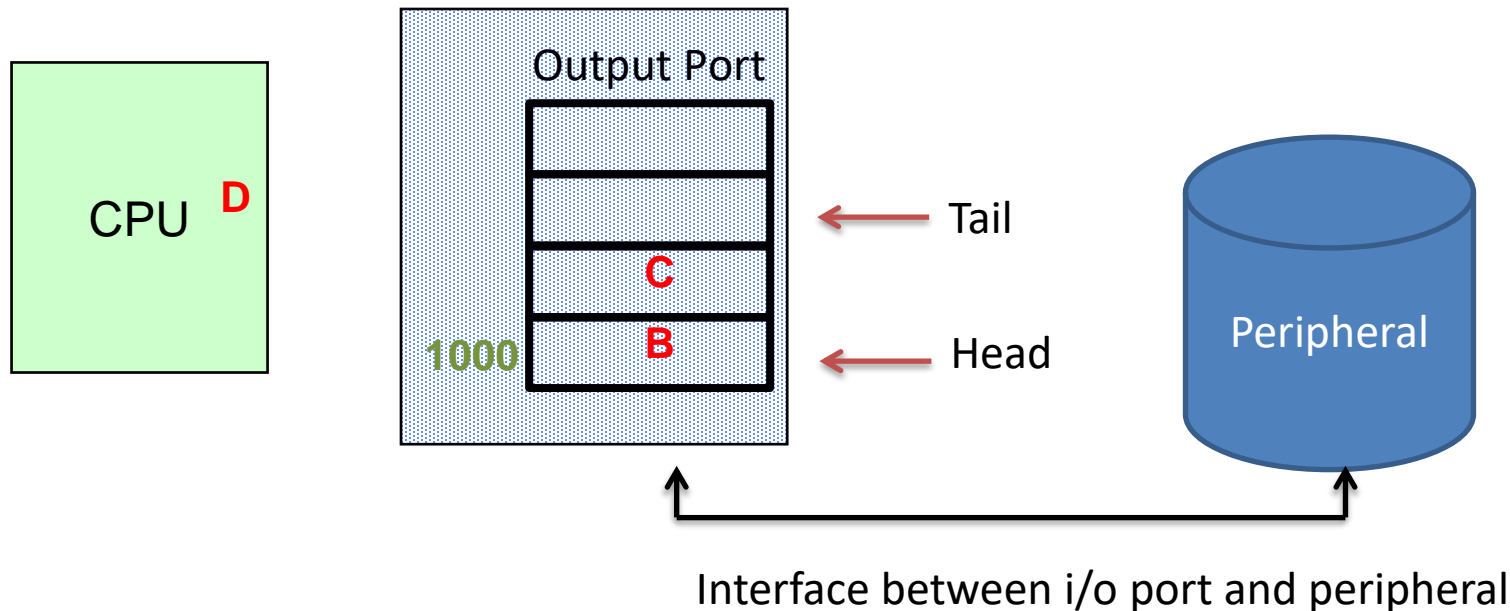
Input and Output Port Buffering

- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port

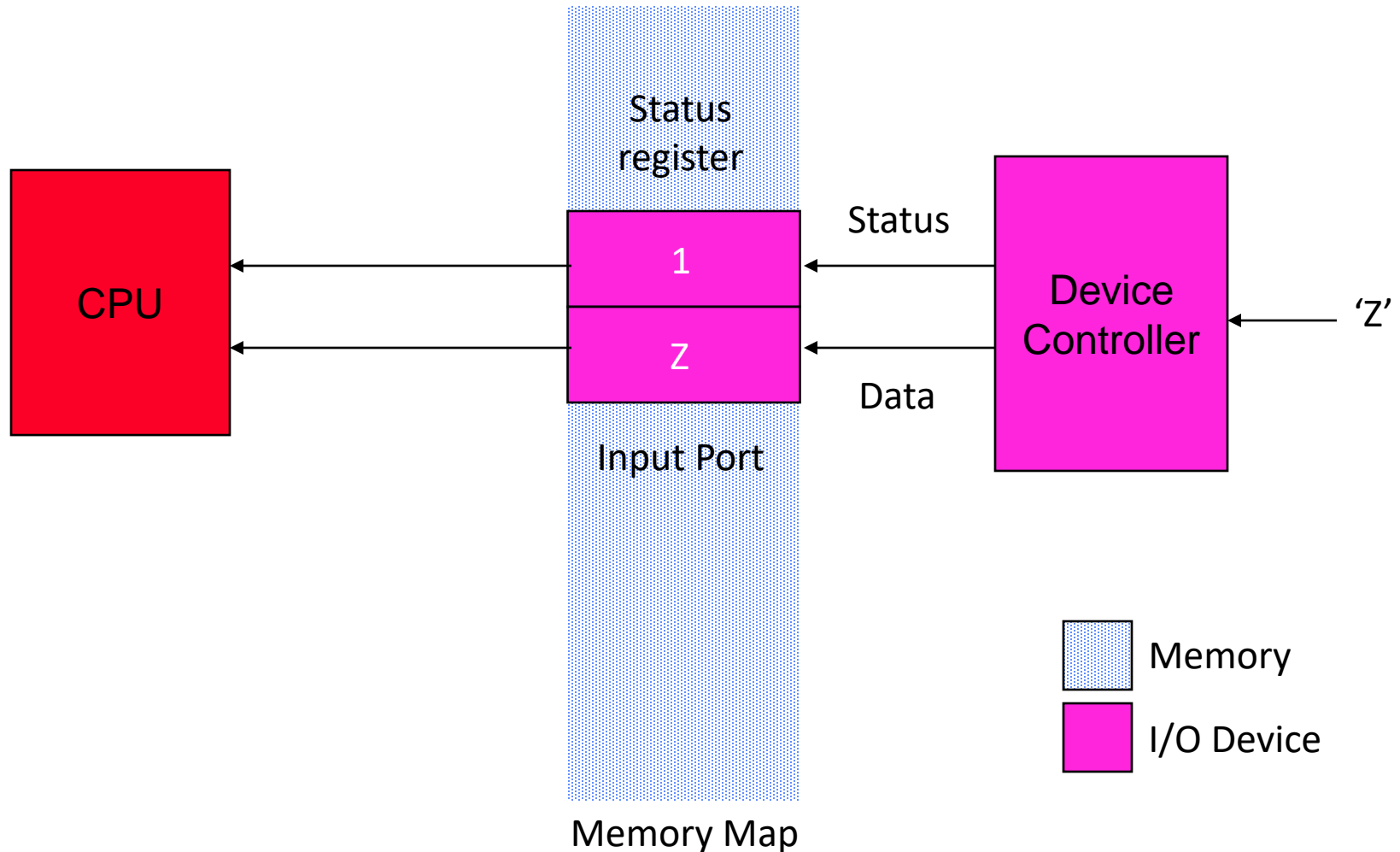


Input and Output Port Buffering

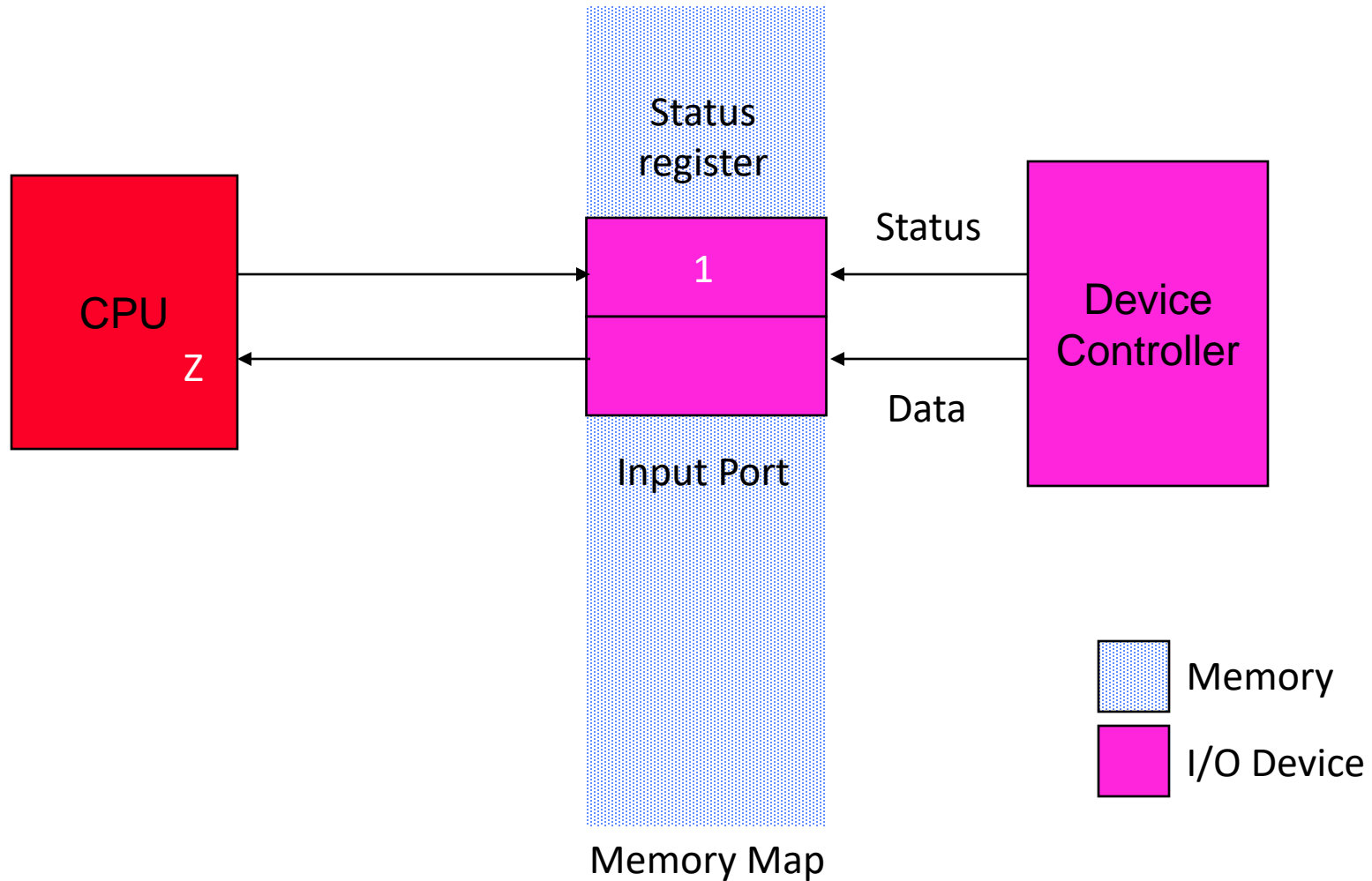
- Input and output ports are usually buffered
 - Typically implemented as a FIFO queue
 - Avoids loss of data at input port
 - Avoids processor having to wait at output port



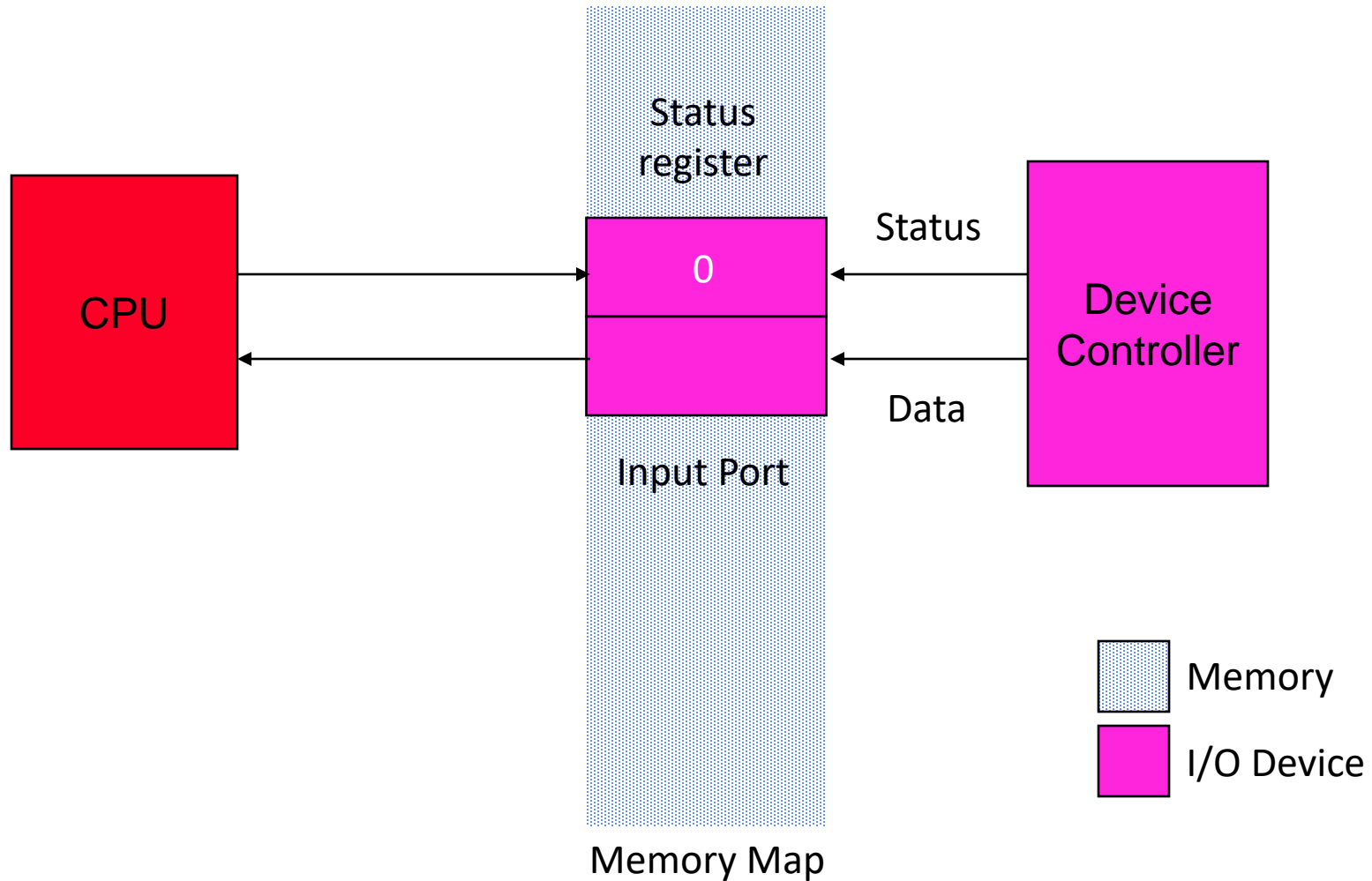
Simple I/O Involving Single Character



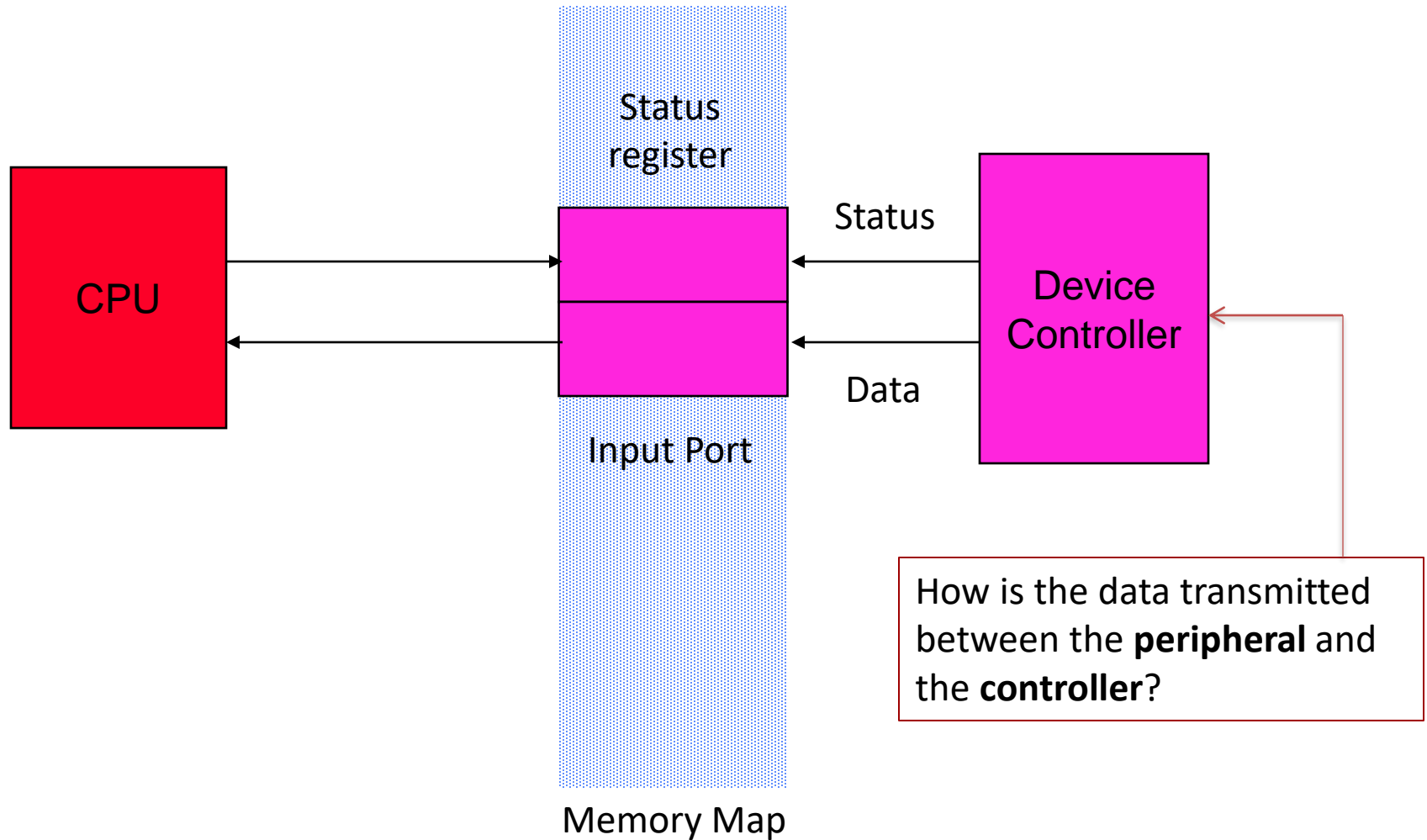
Simple I/O Involving Single Character



Simple I/O Involving Single Character



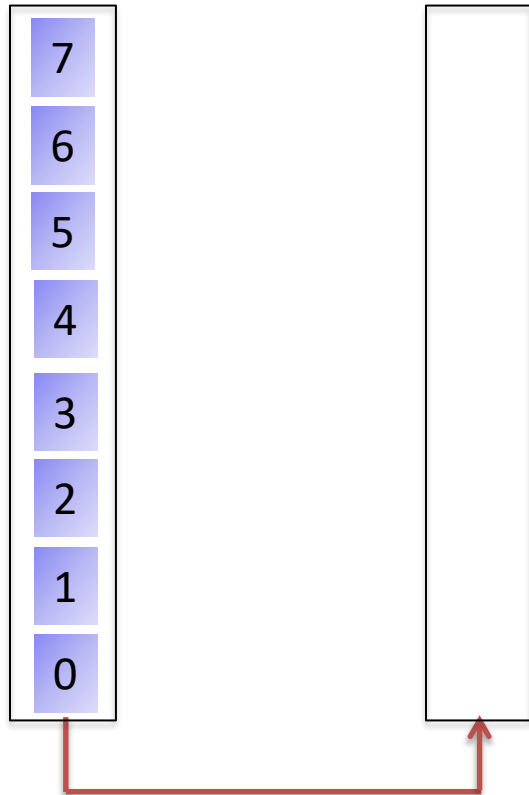
Simple I/O Involving Single Character



Serial/Parallel Transmission

Sending Device

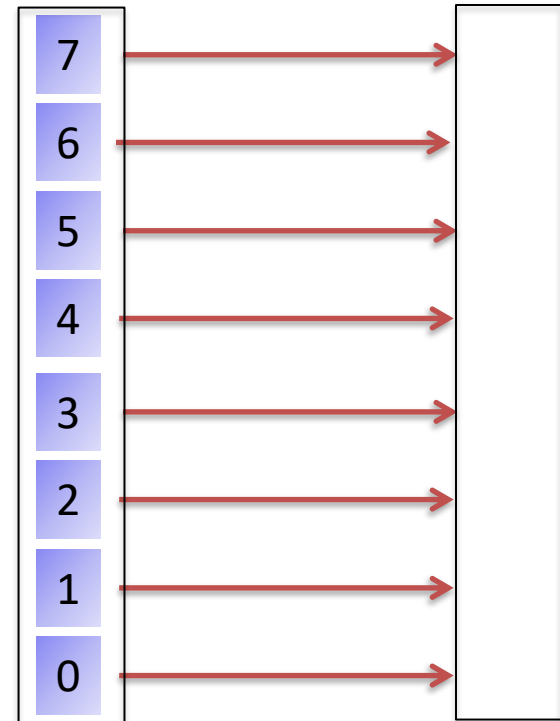
Receiving Device



Serial Transmission (Single Line)

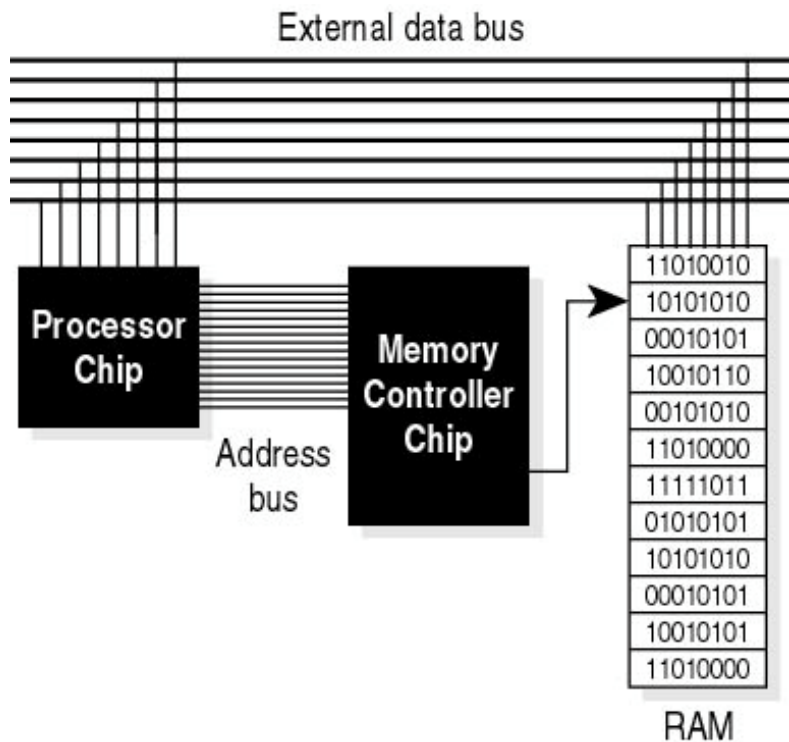
Sending Device

Receiving Device



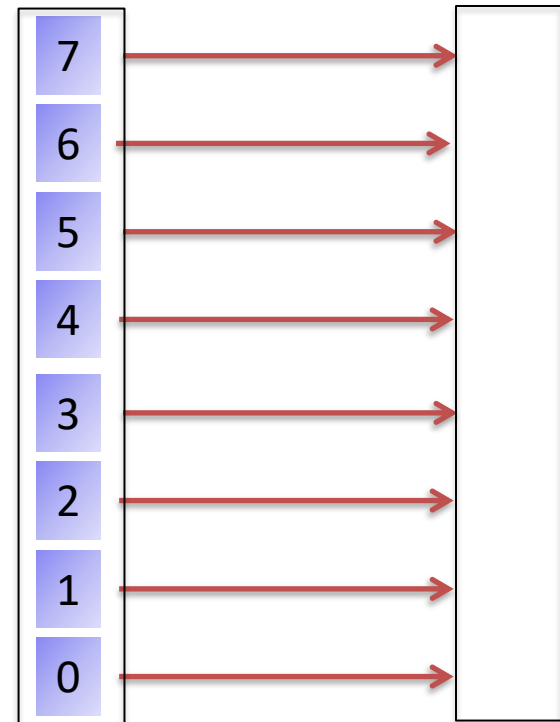
Parallel Transmission (Multiple Lines)

Parallel Transmission - Internal



Sending Device

Receiving Device

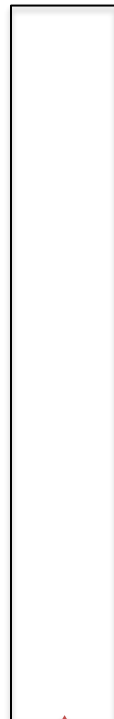
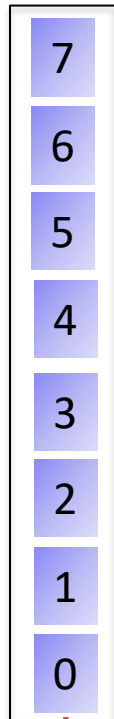


Parallel Transmission (Multiple Lines)

Serial transmission - External

Sending Device

Receiving Device

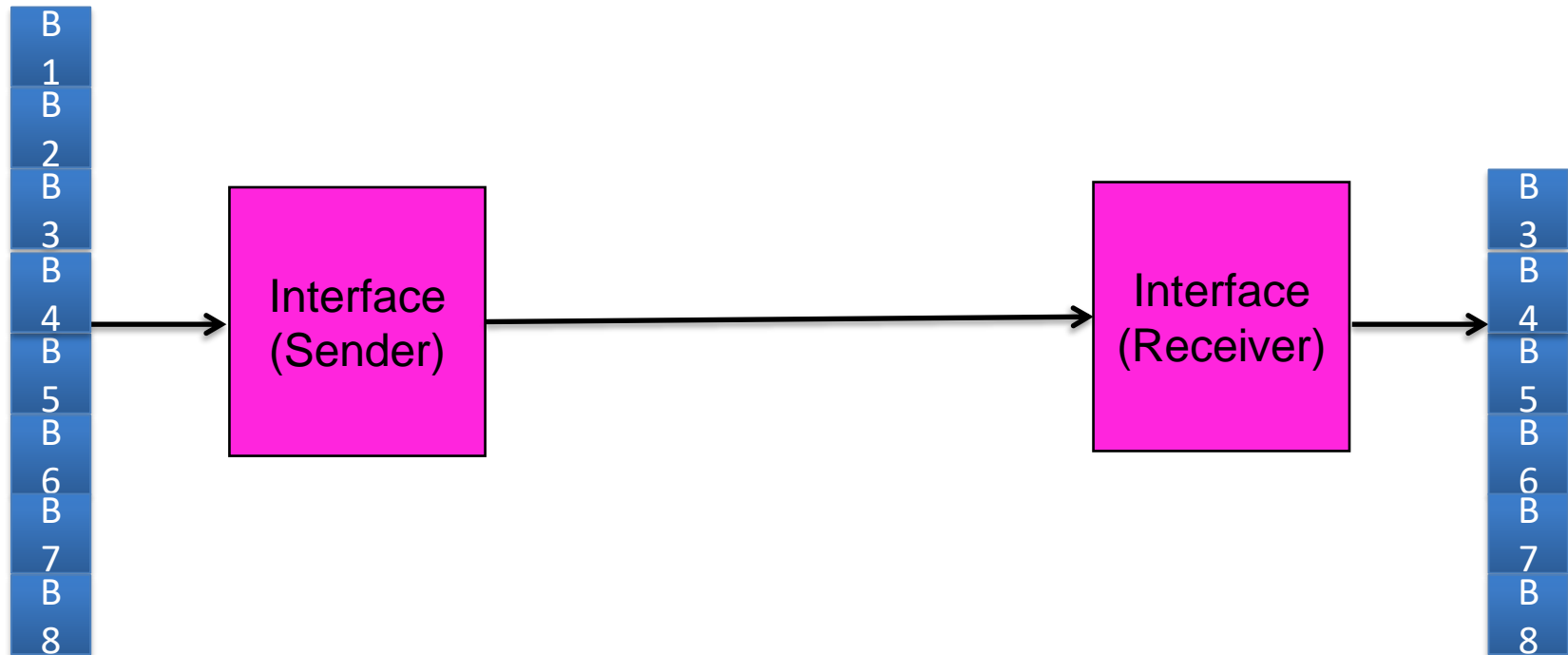


Serial Transmission (Single Line)

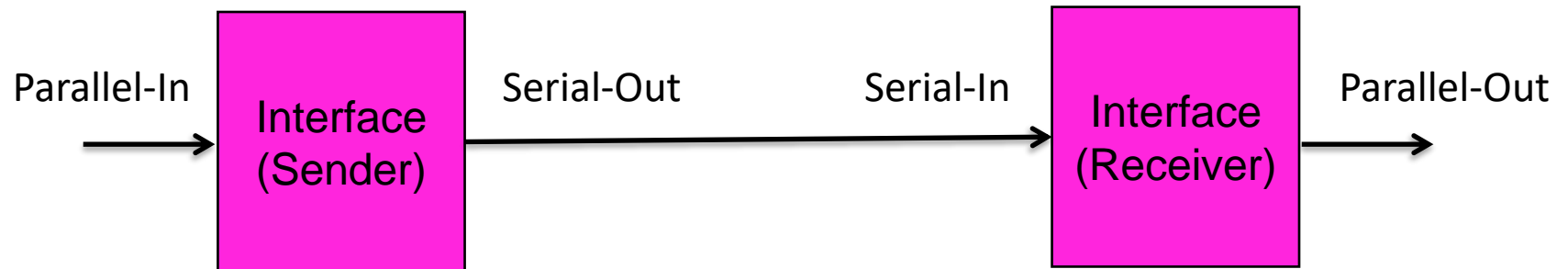


Universal Serial Bus	Speed
USB 1.0 (low speed)	1.5 MBs
USB 1.0 (high speed)	12 MBs
USB 2.0 (high speed)	480 MBs
USB 3.0 (super speed)	5 GBs
USB 3.1 (super speed)	10 GBs
USB4 (coming soon)	40 GBs

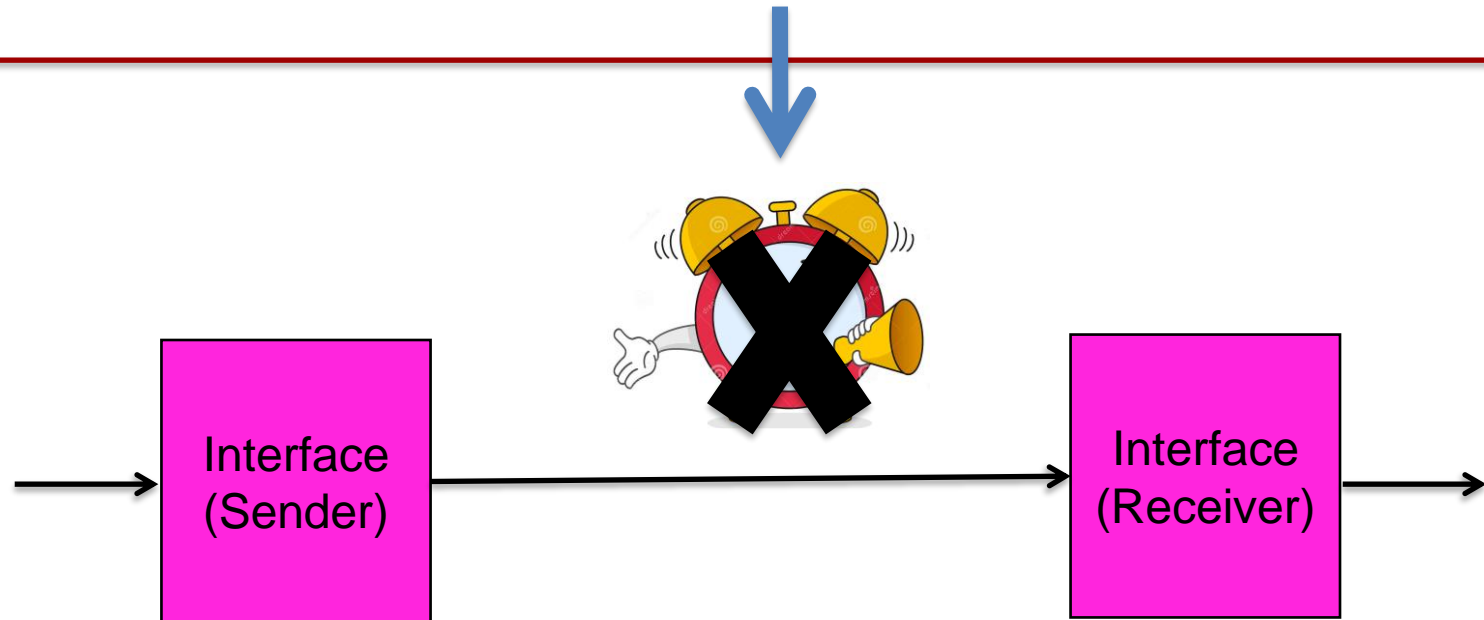
Case Study: Asynchronous Serial Link



Case Study: Asynchronous Serial Link

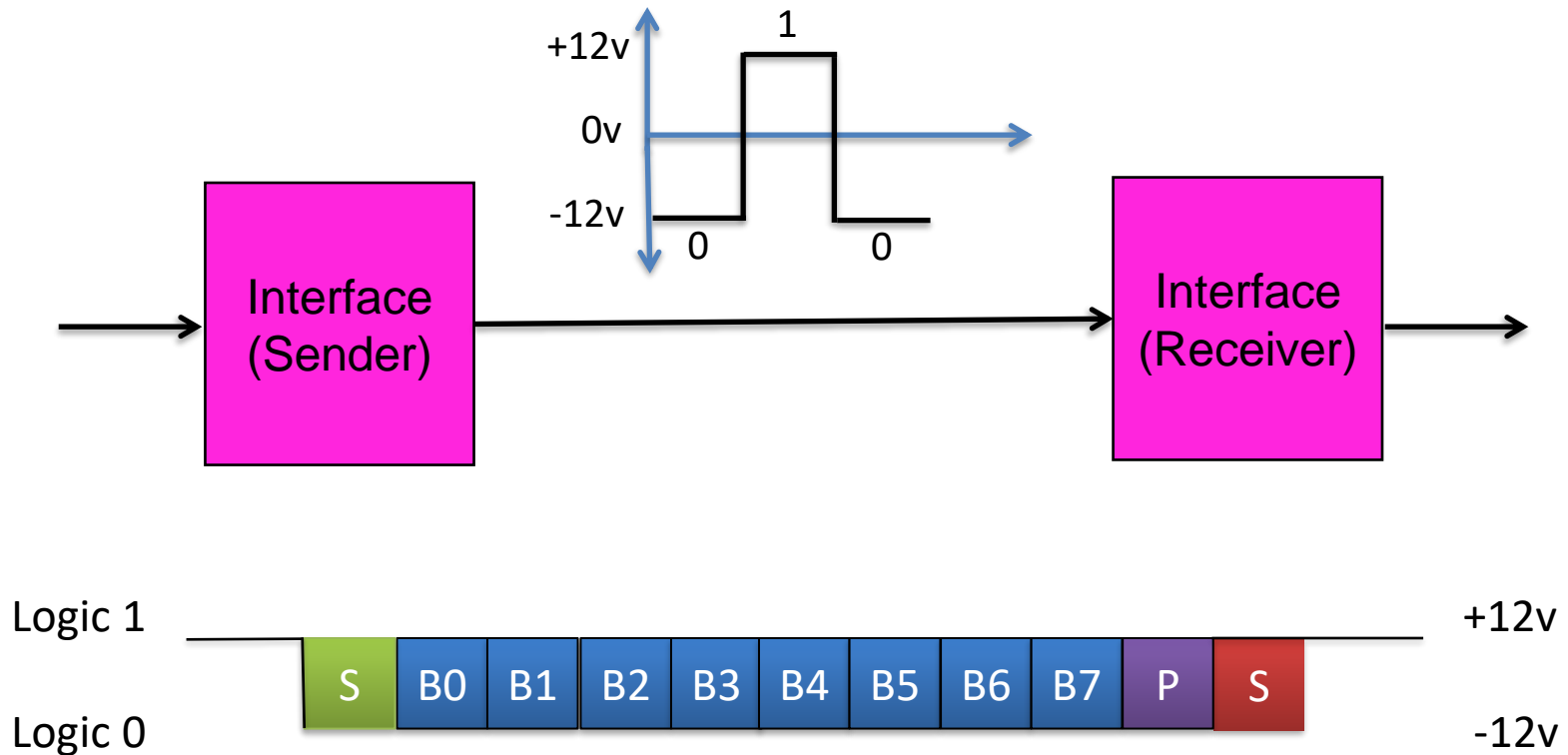


Case Study: Asynchronous Serial Link

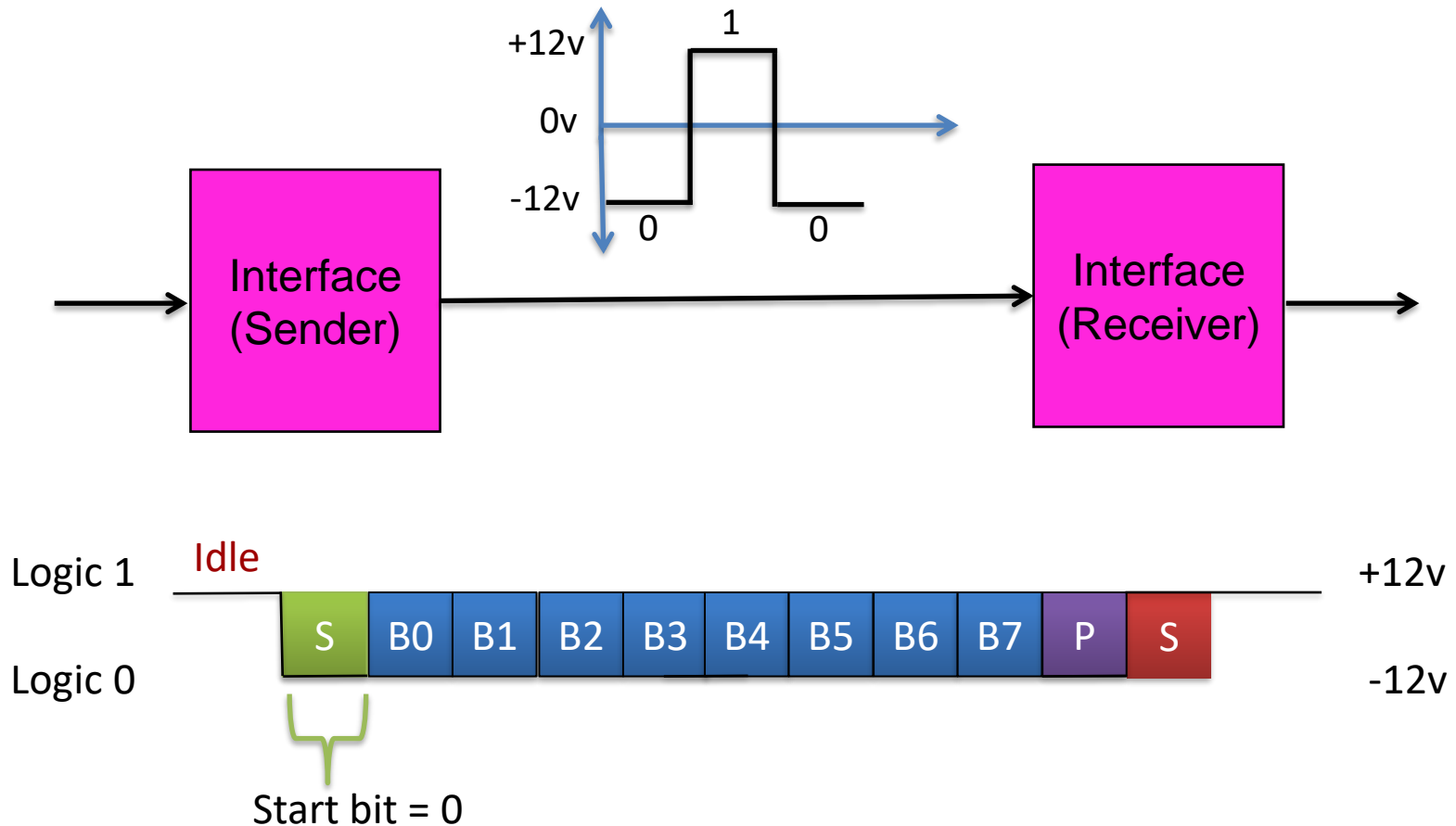


- Data must be formatted so that the receiver can recognize the *beginning* and *end* of the data
- Both the sender and receiver must agree on the formatting parameters *prior* to transmission

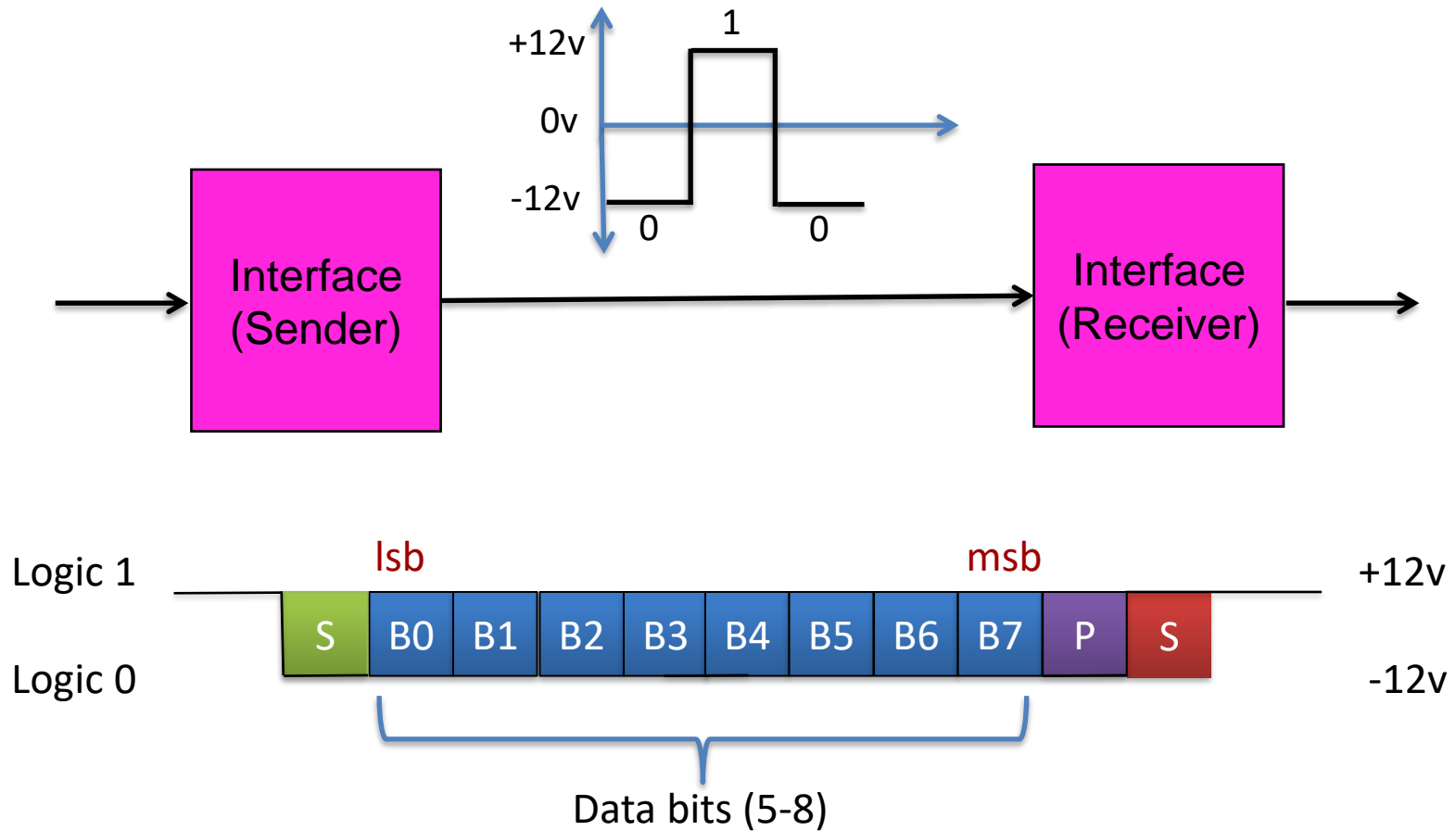
Case Study: Formatting



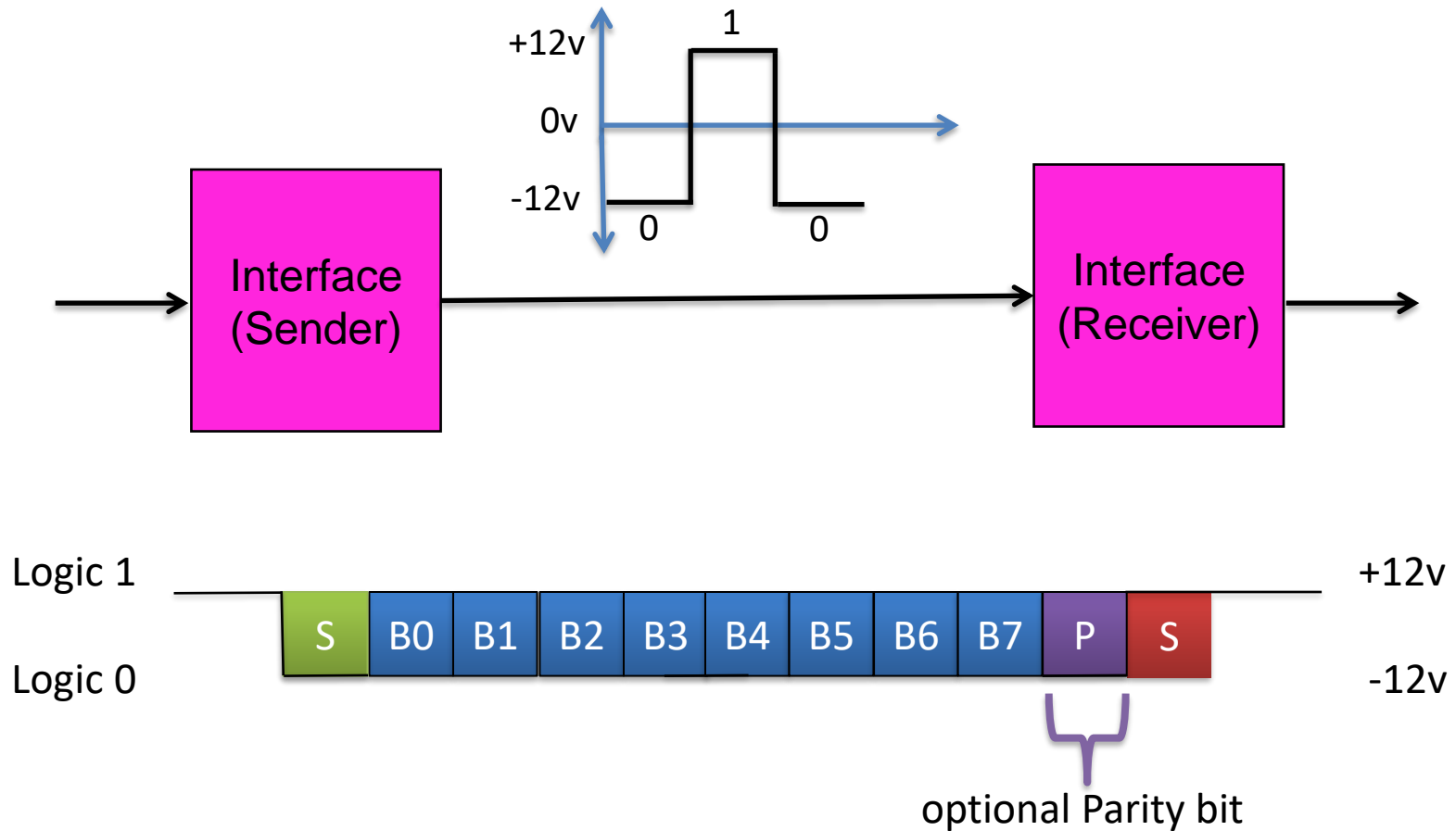
Case Study: Formatting



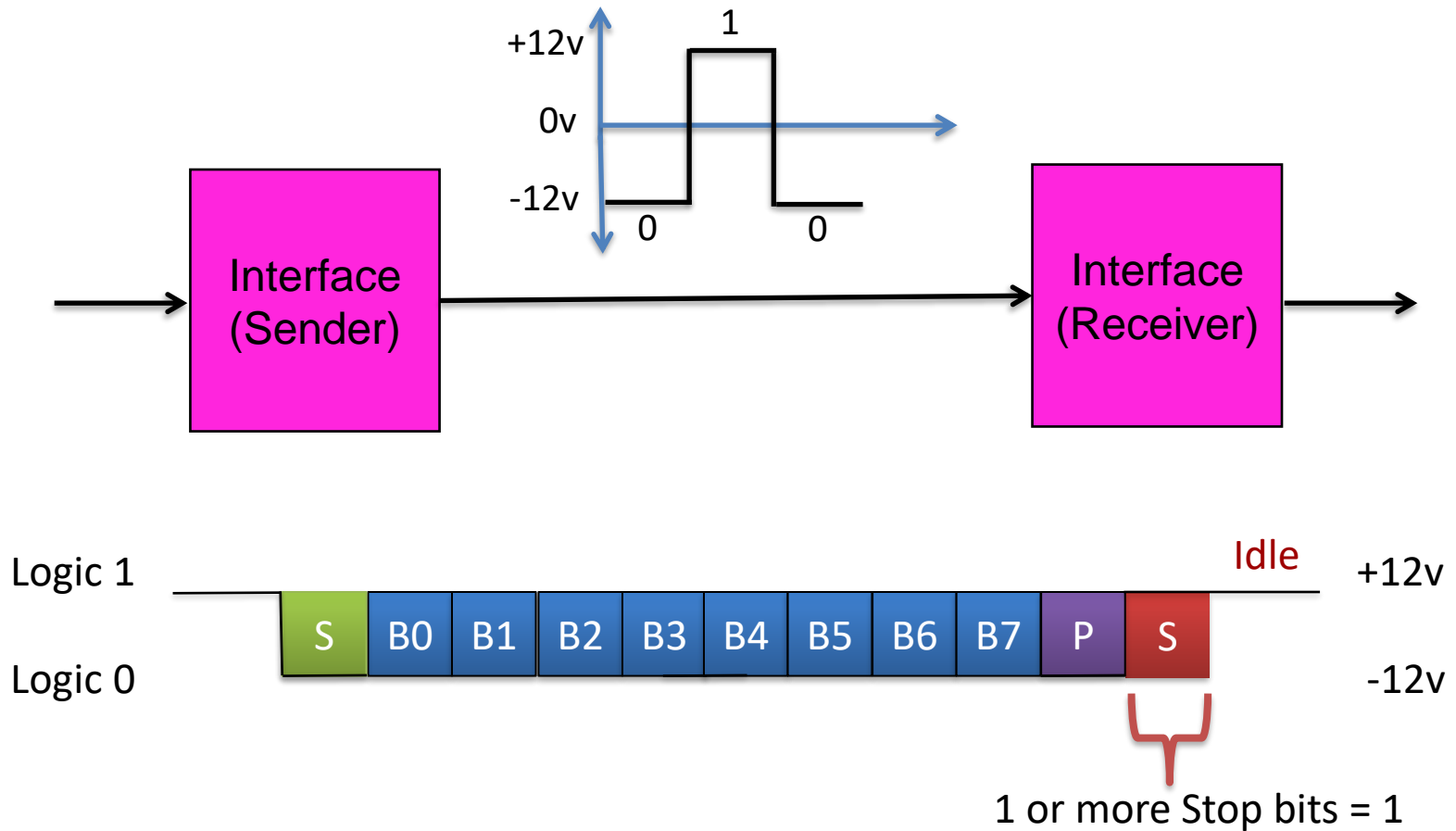
Case Study: Formatting



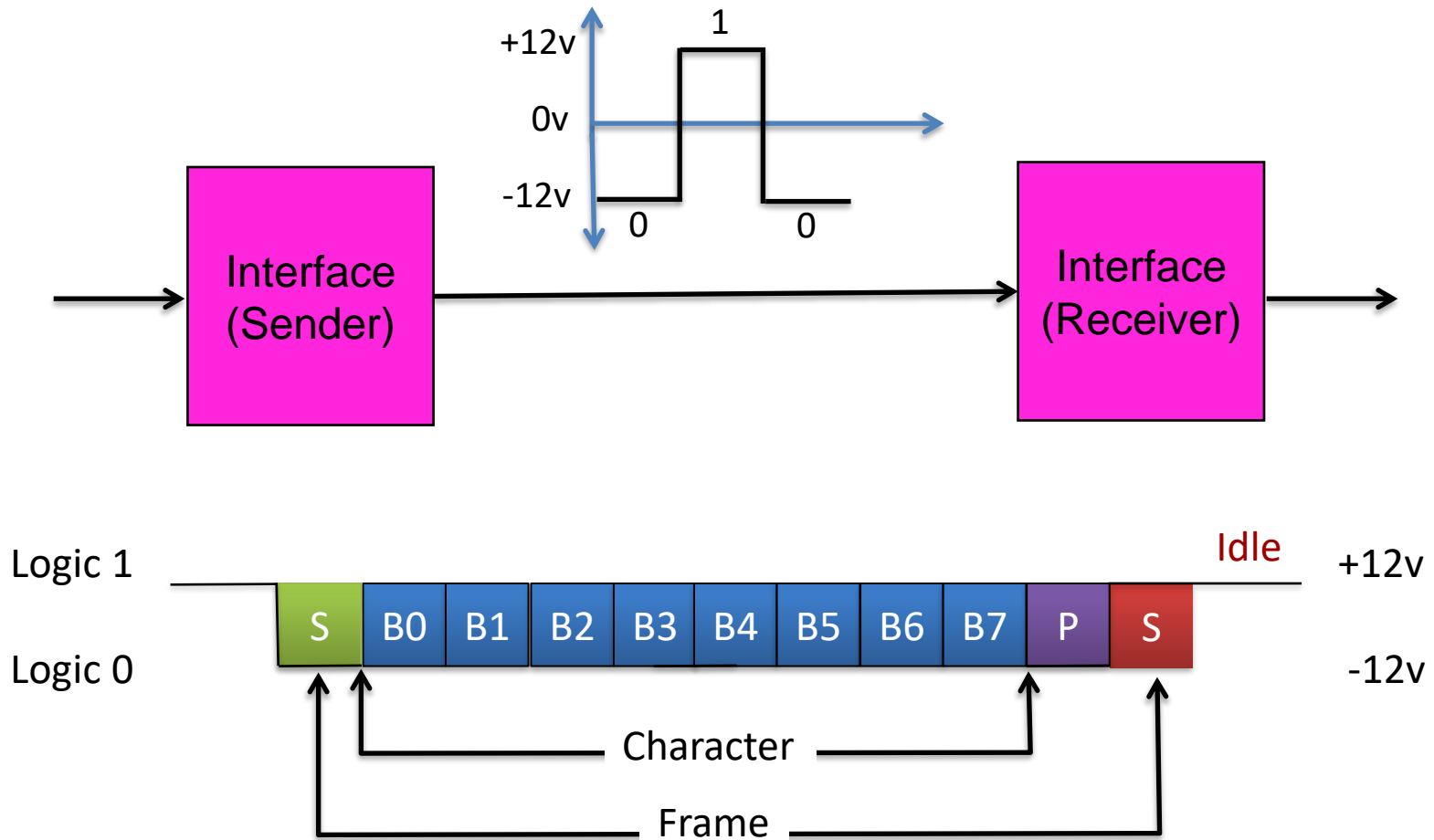
Case Study: Formatting



Case Study: Formatting



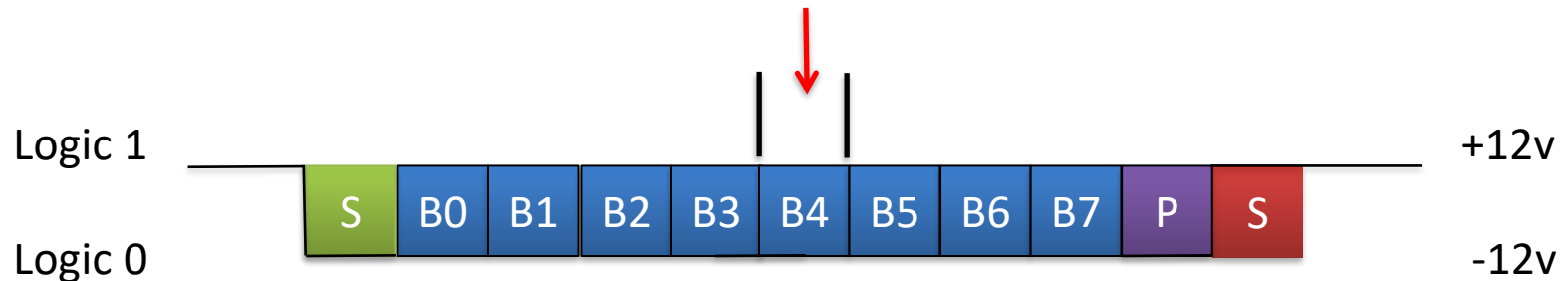
Case Study: Framing of a Character



Case Study: How Fast is the Serial Bus?

- The speed (or frequency) of a serial bus is defined as the **baud rate** and is measured in bits per second.

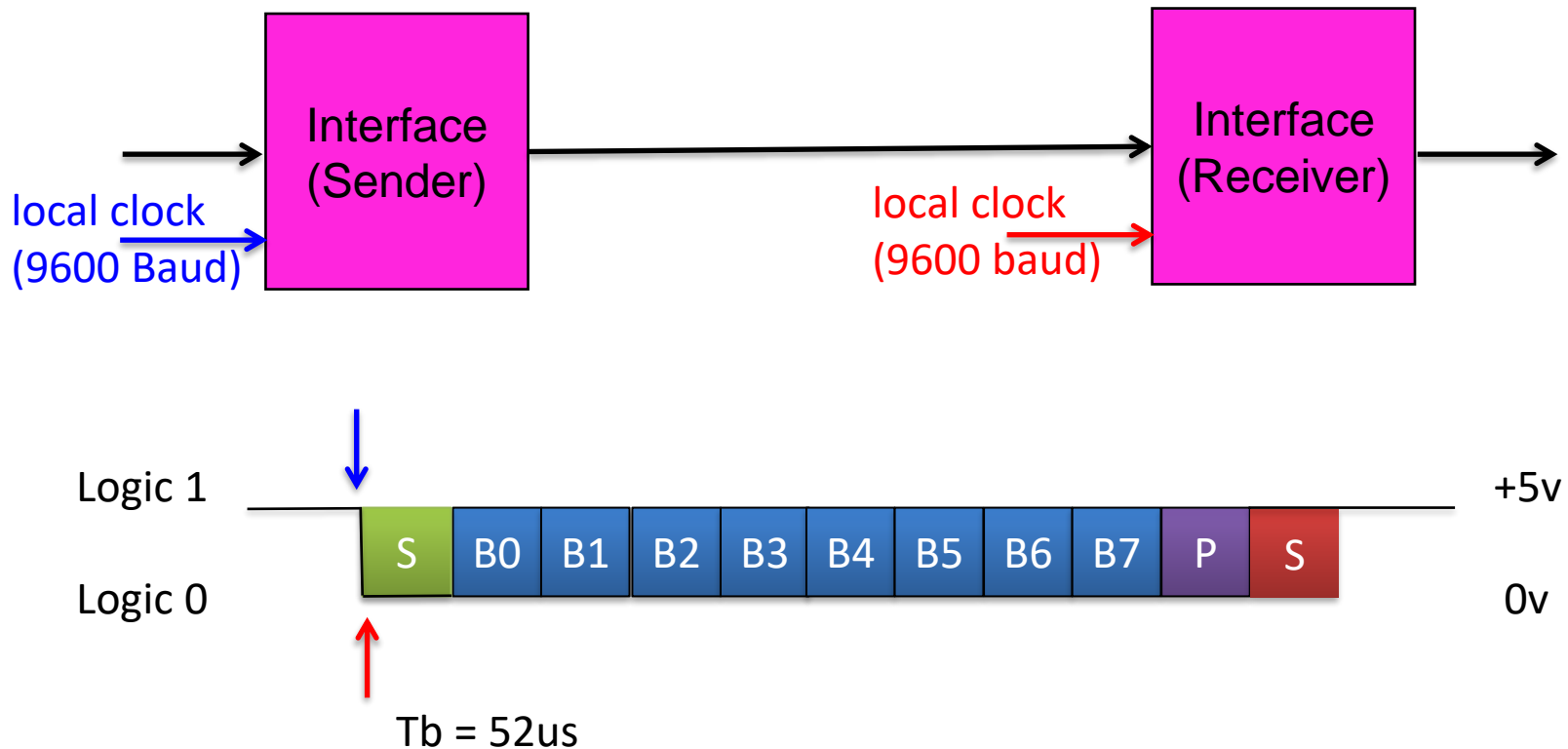
Baud Rates (bps)	Time-Per-Bit ($T_b = 1 / \text{Baud Rate}$)
1200	833 microseconds
2400	416 microseconds
4800	208 microseconds
9600	104 microseconds
115,200 (max)	8.6 microseconds



Case Study: Local Clocks

Baud Rate = 9600 bps

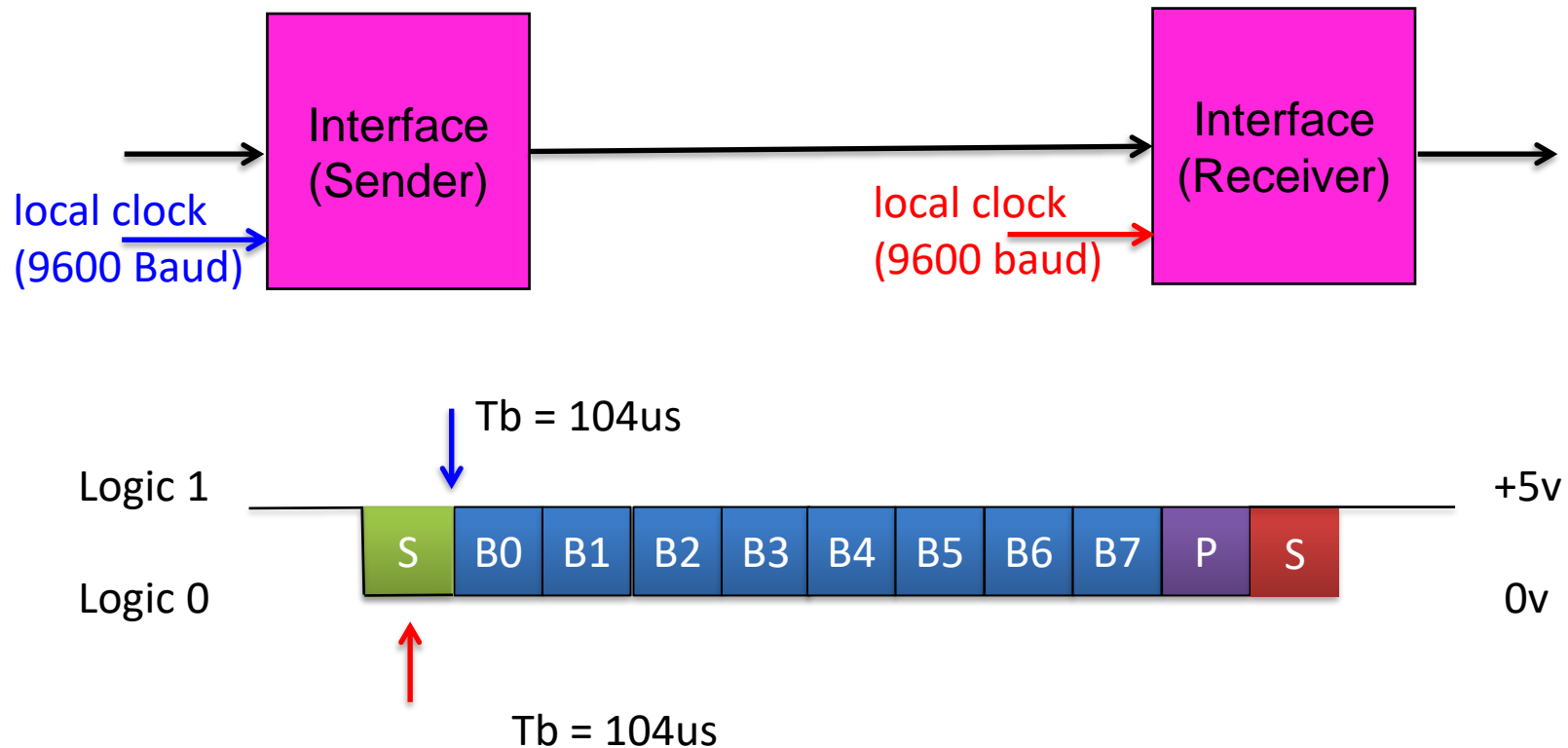
Time per bit (T_b) = 104 microseconds



Case Study: Local Clocks

Baud Rate = 9600 bps

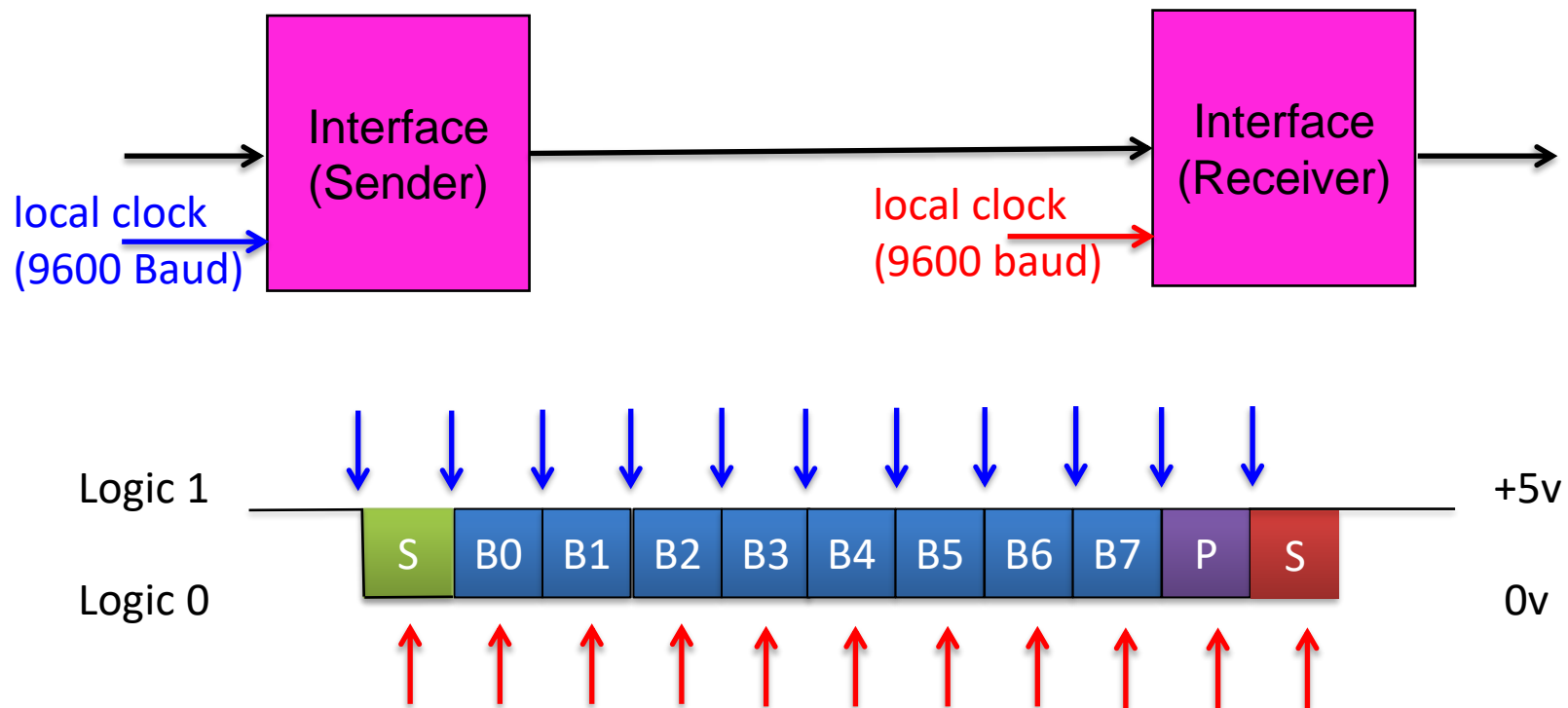
Time per bit (T_b) = 104 microseconds



Case Study: Local Clocks

Baud Rate = 9600 bps

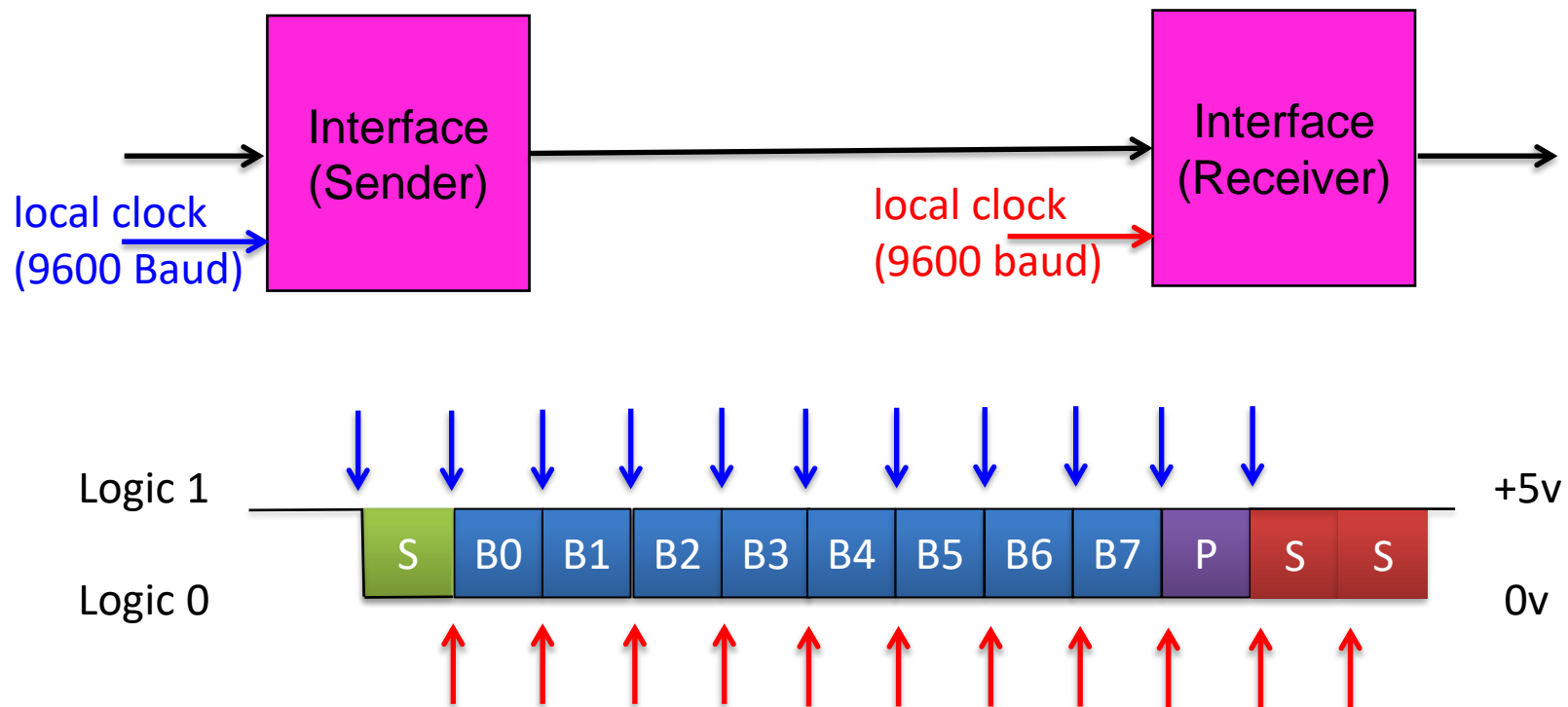
Time per bit (T_b) = 104 microseconds



Case Study: Local Clocks

Baud Rate = 9600 bps

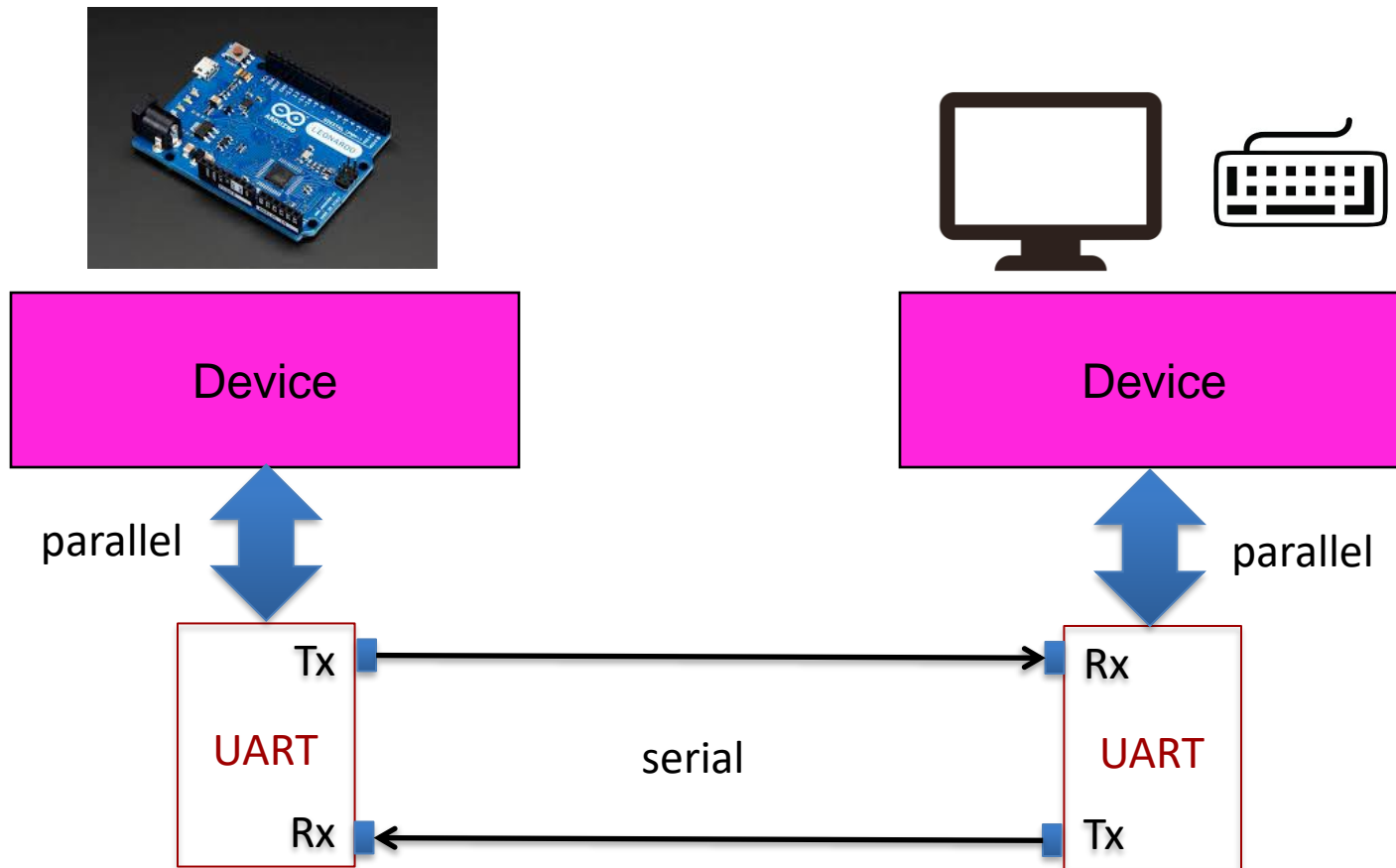
Time per bit (T_b) = 104 microseconds



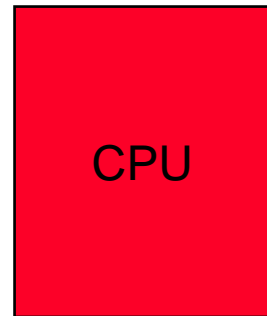
In-Class Example

- An asynchronous serial interface is configured to operate as follows
 - 9600 baud
 - 1 start bit
 - 7 data bits
 - 1 parity bit
 - 2 stop bits
- How much time would it take to transmit 100 ASCII characters?

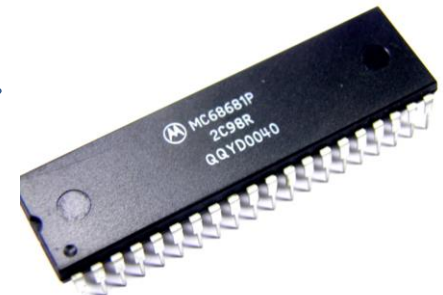
Universal Asynchronous Receiver Transmitter (UART)



MC68681 DUART



8-bit registers



\$FFFFFFE			\$FFFFFFF
	.	.	
	.	.	
	.	.	
\$00C006	Byte	Reg 3	\$00C007
\$00C004	Byte	Reg 2	\$00C005
\$00C002	Byte	Reg 1	\$00C003
\$00C000	Byte	Reg 0	\$00C001

Base Address
of DUART



DUART's Internal Registers

	Read Operation		Write Operation	
Address	Mnemonic	Full Name	Mnemonic	Full Name
C001	MR1A, MR2A	Mode Register A	MR1A, MR2A	Mode Register A
C003	SRA	Status Register A	CSRA	Clock-Select Register A
C005	N/A	Do not access	CRA	Command Register A
C007	RBA	Receive Buffer A	TBA	Transmit Buffer A
C009	IPCR	Input Port Change Register	ACR	Auxiliary Control Register
C00B	ISR	Interrupt Status Register	IMR	Interrupt Mask Register
C00D	CUR	Counter Upper Register	CTUR	Counter/Timer Upper Register
C00F	CLR	Counter Lower Register	CTLR	Counter/Timer Lower Register

DUART's Internal Registers Continued

	Read Operation		Write Operation	
Address	Mnemonic	Full Name	Mnemonic	Full Name
C011	MR1B, MR2B	Mode Register B	MR1B, MR2B	Mode Register B
C013	SRB	Status Register B	CSRB	Clock-Select Register B
C015	N/A	Do not access	CRB	Command Register B
C017	RBB	Receive Buffer B	TBB	Transmit Buffer B
C019	IVR	Interrupt Vector Register	IVR	Interrupt Vector Register
C01B	IP	Input port	OPCR	Output Port Configuration Register
C01D	START	Start Counter Command	OPRSET	Output Port Register Bit Set
C01F	STOP	Stop Counter Command	CTLR	Output Port Register Bit Clear

Channel A Mode Register 1 (MR1A)

- Defines some of the basic operating parameters of the DUART

Other Parameters			Parity Mode		Parity Type	Bits/Character	
7	6	5	4	3	2	1	0
0	0	0	00 = with parity		0 = even	10 = 7	
Assume			01 = no parity		1 = odd	11 = 8	

- Configure the serial port on the DUART to operate with 7 data bits and odd parity
 - Memory-mapped address: \$C001

Channel A Mode Register 2 (MR2A)

- Also defines some of the basic operating parameters of the DUART

Channel Mode		Other parameters		Stop-bit Length			
7	6	5	4	3	2	1	0
0	0	0	0	0 1 1 1 = 1 bit			
Normal		Assume		1 1 1 1 = 2 bits			

- Configure the serial port on the DUART to operate normally with 2 stop bits
 - Memory-mapped address: \$C001

Clock-Select Register for Channel A (CSRA)

- The clock-select register is used to specify the transmitter and receiver baud rates

Receiver Clock				Transmitter Clock			
7	6	5	4	3	2	1	0
0100 300				0100 300			
0101 600				0101 600			
0110 1200				0110 1200			
1000 2400				1000 2400			
1001 4800				1001 4800			
1011 9600				1011 9600			
1100 19200				1100 19200			
Baud Rate				Baud Rate			

- Configure the serial port on the DUART to operate at 9600 baud
 - Memory-mapped address: \$C003

Channel A Command Register (CRA)

- The clock-select register is used to control the DUART's transmitter and receiver

Other Commands				Transmitter Commands		Receiver Commands	
7	6	5	4	2	3	1	0
0	0	0	0	00 = no action 01 = enable 10 = disable		00 = no action 01 = enable 10 = disable	
Assume							

- Enable the DUART's transmitter and receiver
 - Memory-mapped address: \$C005

Initializing the DUART

- Write an instruction sequence to initialize the DUART's channel A serial interface to operate with 1 start bit, 7 data bits, odd parity, and 2 stop bits.


```
duart    equ    $C001                ;base (1st) register
mr1a     equ    0
mr2a     equ    0
csra     equ    2
cra      equ    4

        lea     duart,a0              ;a0 points to duart
        move.b  #$06,mr1a(a0)         ;7 data bits, odd parity
        move.b  #$0f,mr2a(a0)         ;2 stop bits
        move.b  #$bb,csra(a0)         ;baud rate = 9600
        move.b  #$05,cra(a0)          ;enable tx/rx
```

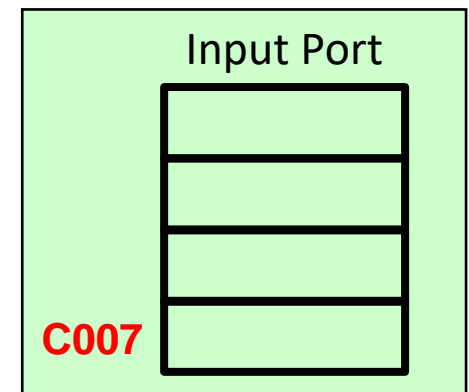
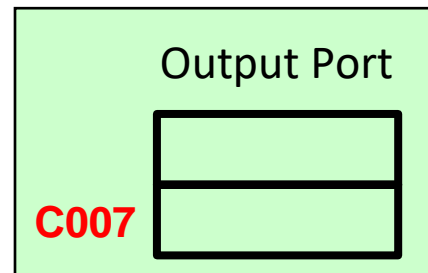

Channel A Status Register

- The status register provides information about the DUART's receiver and transmitter.

Received Break	Framing Error	Parity Error	Overrun Error	TxEMPTY	TxRDY	FFULL	RxRDY
7	6	5	4	3	2	1	0
0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes



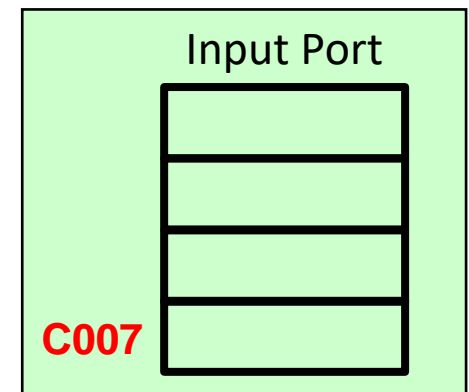
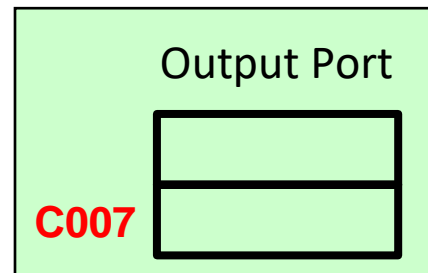

- Memory-mapped address: **\$C003**



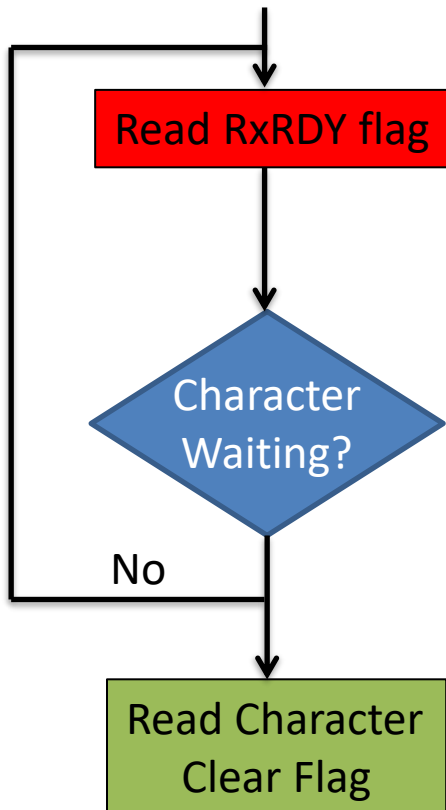
Channel A Status Register

- The status register provides information about the DUART's receiver and transmitter.

Received Break	Framing Error	Parity Error	Overrun Error	TxEMPTY	TxRDY	FFULL	RxRDY
7	6	5	4	3	2	1	0
0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes



Using a Polling Strategy to Read a Character

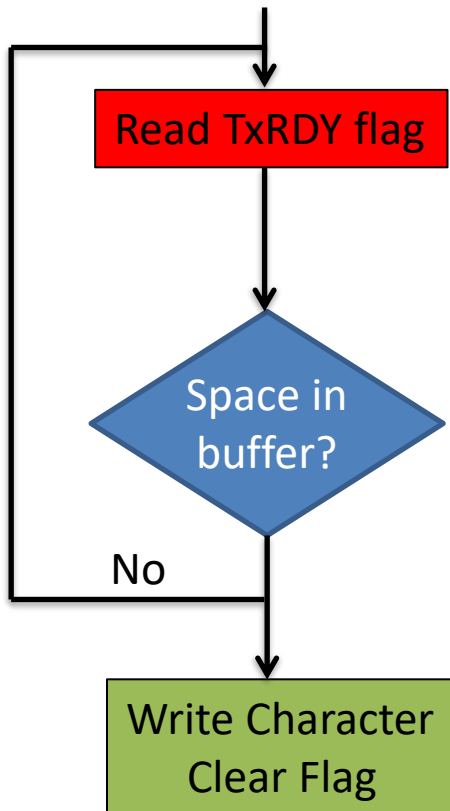


Received Break	Frame Error	Parity Error	Overrun Error	TxEMPTY	TxRDY	FFULL	RxRDY
7	6	5	4	3	2	1	0
0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

```
duart    equ    $C001
sra      equ    2
rba      equ    6

loop     lea      duart,a0
         move.b   sra(a0),d7
         andi.b   #1,d7
         beq      loop
         move.b   rba(a0),d0
```

Using a Polling Strategy to Write a Character



Received Break	Frame Error	Parity Error	Overrun Error	TxEMPTY	TxRDY	FFULL	RxRDY
7	6	5	4	3	2	1	0
0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

```
duart    equ    $C001
sra      equ    2
tba      equ    6

loop     lea     duart,a0
         move.b  sra(a0),d7
         andi.b  #4,d7
         beq     loop
         move.b  d0,tba(a0)
```

In-Class Example

- Write a program to display the message “Assembly-language Programming is fun” on the terminal attached to the DUART’s serial port A.
 - Display the message 10 times on 10 separate lines