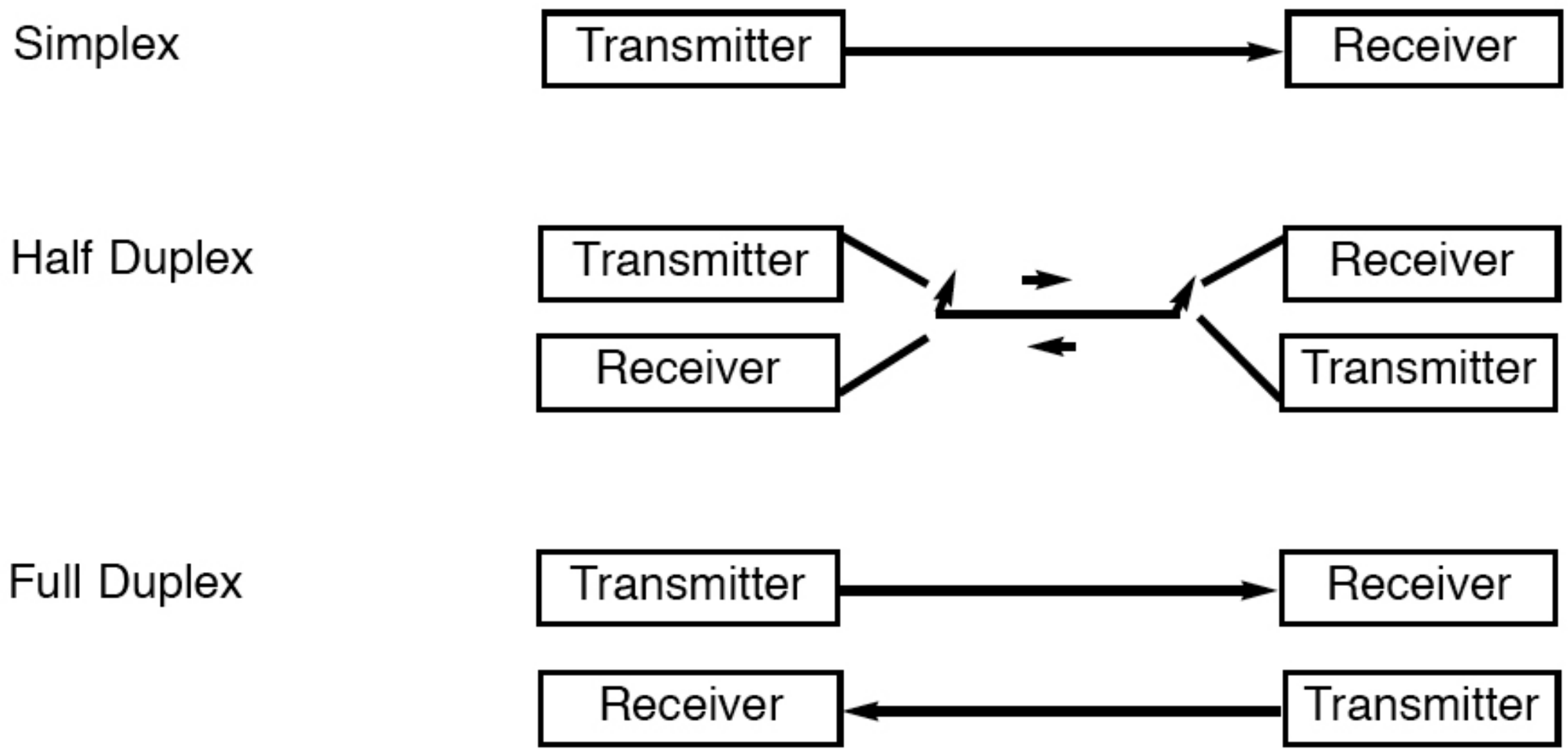
Serial I/O I

ECE 3710

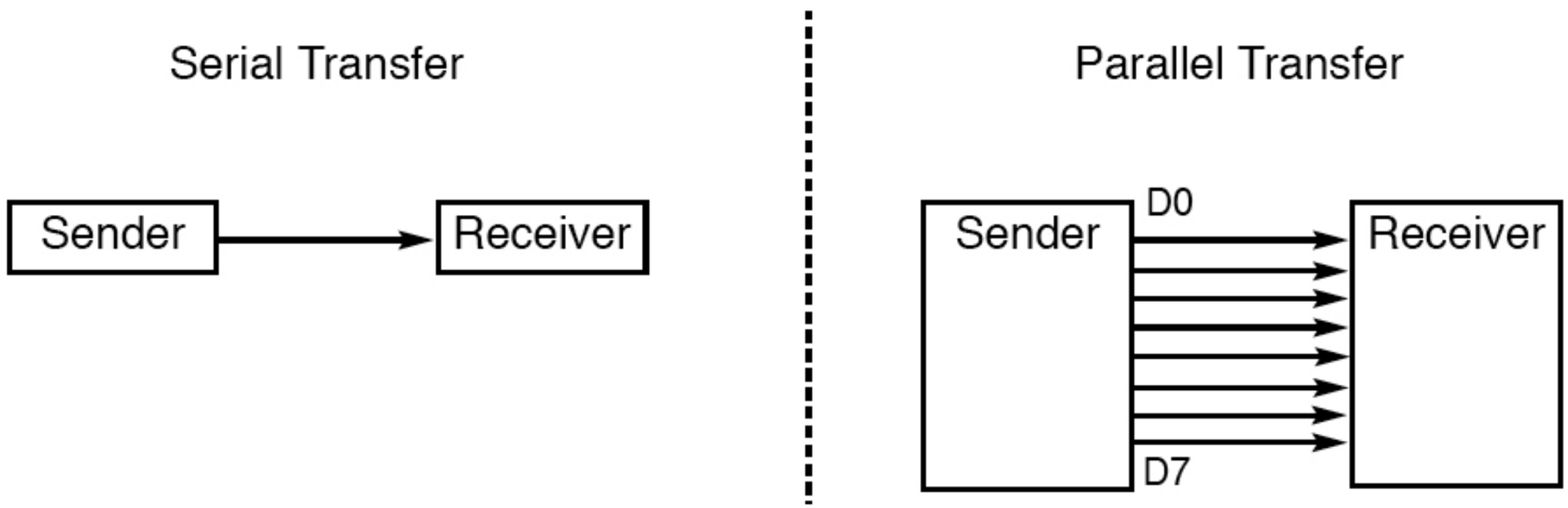
When someone is impatient and says, "I haven't got all day," I always wonder, How can that be? How can you not have all day?

- George Carlin

communication systems



we have assumed that sender/receiver sync'd

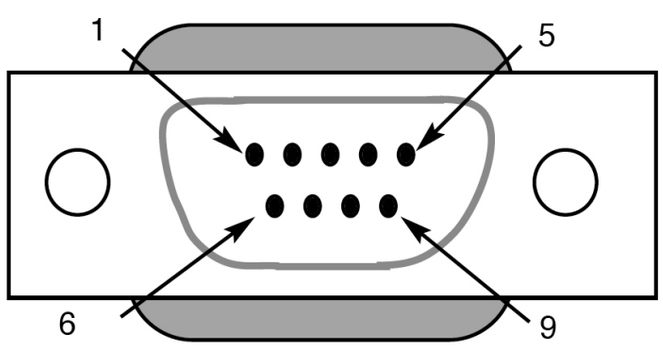


synchronous: sync'd clocks
asynchronous: clock in data stream

let someone else
figure it out

universal asynchronous
receiver-transmitter (UART):

1. rate
2. sending
3. receiving



DB-9

pins:

1. TxD
2. RxD
3. Grnd

full-duplex

(transitions per second)

data rate:

1. baud rate (Hz)
2. bit rate (bits per second)

baud rate vs. bit rate

baud rate:

rate at which we transmit a symbol
(one symbol per baud)

single bit or multiple bits

common baud rates

110
150
300
600
1200
2400
4800
9600
19200

Note: Some of the Baud rates supported by 486/ Pentium IBM PC BIOS.

ex:

9600 baud rate

2-bits per transition

= 9600 * 2

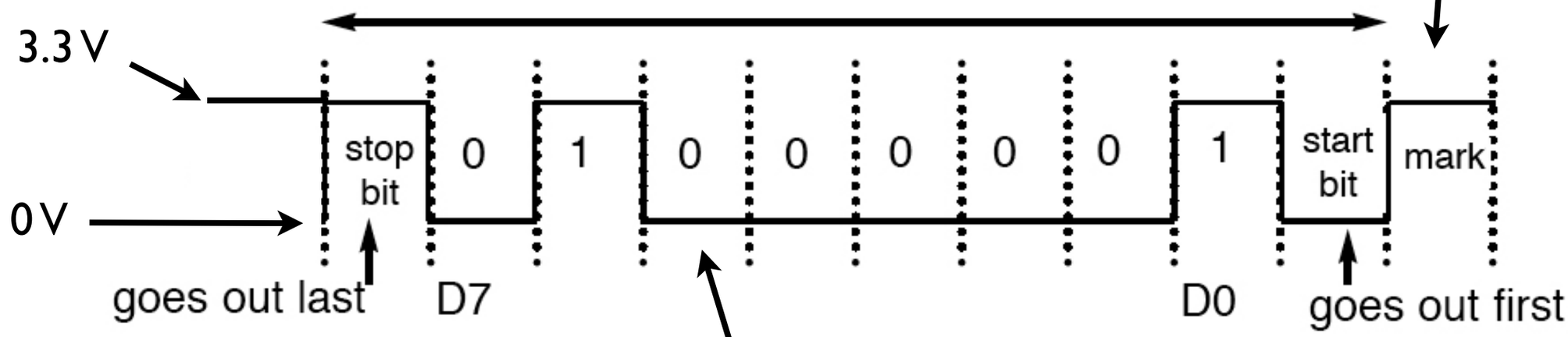
= 19,200 bps

UART: transmit ‘A’ (9600 baud)

(always transmit a byte)

idle: logic high

a frame



one bit every 1/9600 s

transmission mode:

8-bits data, one bit (low) start, one bit (high) stop, no parity

serial communication overview

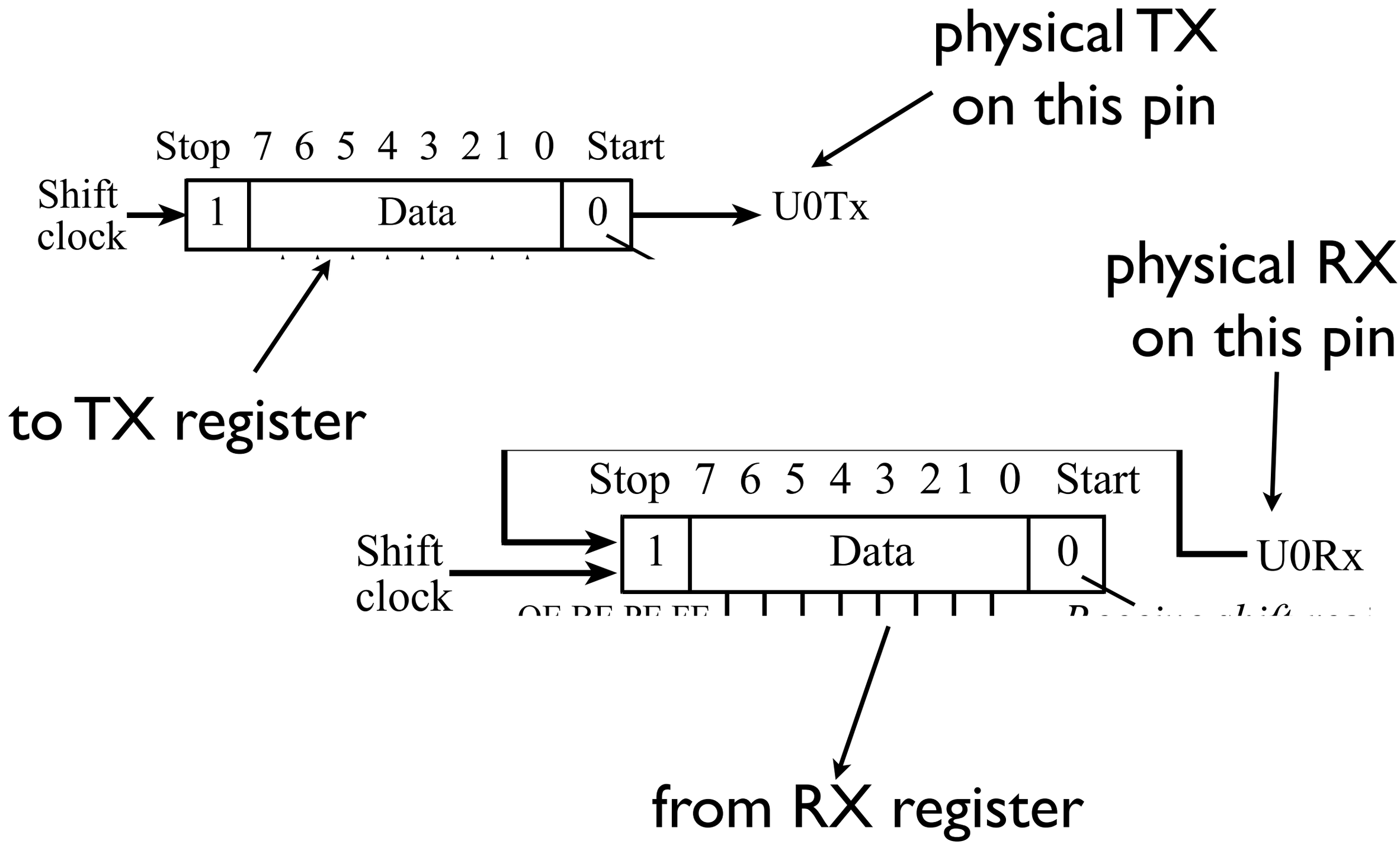
0. set speed (baud rate) at which TX/RX occurs
(the same; assuming one UART)

1. TX:
a. move data to TX register
b. wait for TX to finish
c. goto 1a.

presumably a bit flip, somewhere

2. RX:
a. wait for RX to finish
b. move data from RX register
c. goto 2a.

the how of serial TX/RX



this seems suspiciously easy...

...can we find a way to make it more complicated?

complicate it:

1. fixed-point arithmetic for baud rate
2. TX/RX register (just one)
3. hardware buffering

next time

complications:

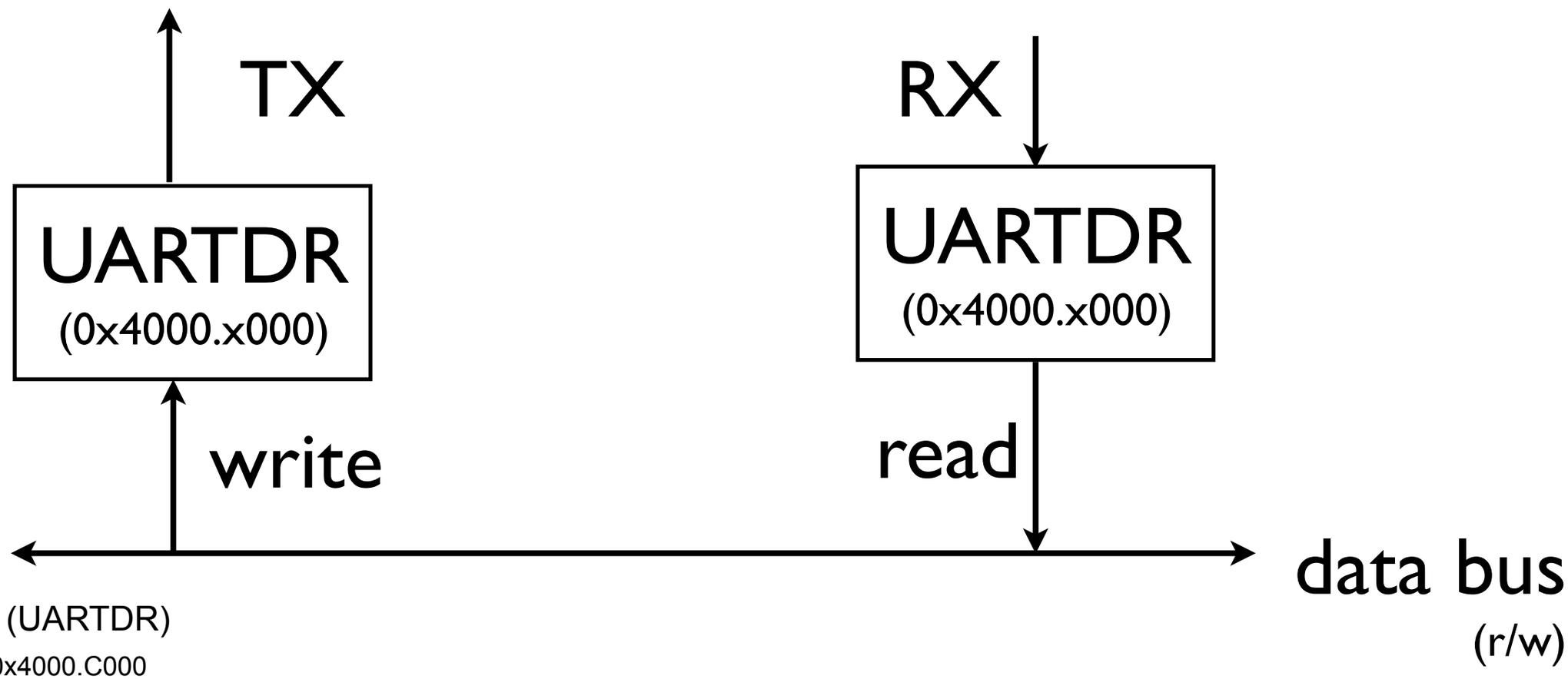
they actually make life easier...in the long run

(remember Keynes: 'In the long run, we are all dead')

2. TX/RX register

just one?
(uC specific)

write to DR: TX
read from DR: RX



UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
Offset 0x000
Type R/W, reset 0x0000.0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				OE	BE	PE	FE	DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

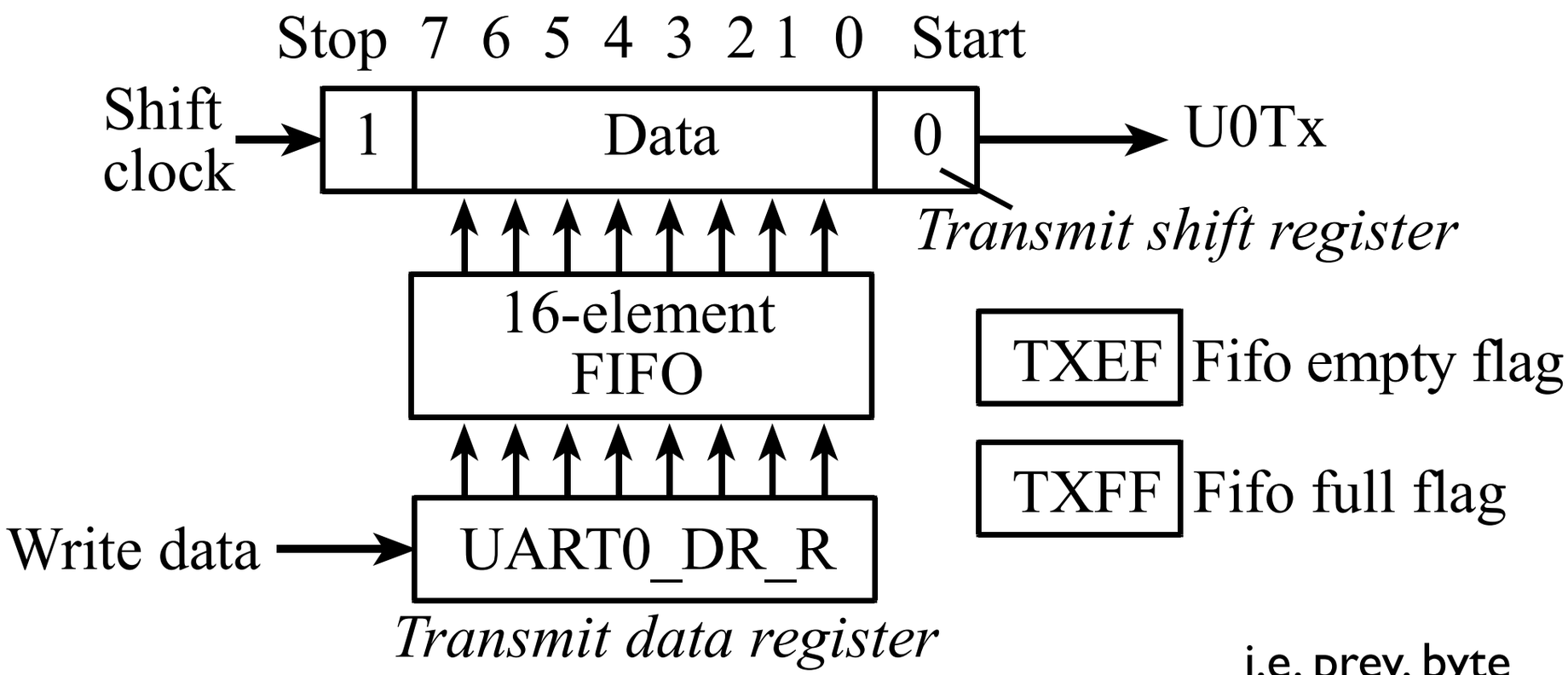
3. hardware buffering

problem:

1. data comes in spurts (nothing...then a lot)
2. uC faster than serial:
   a. give the serial lots of data then do something else
   b. get lots of data while doing something else

solution: hardware buffering

TX FIFO buffer

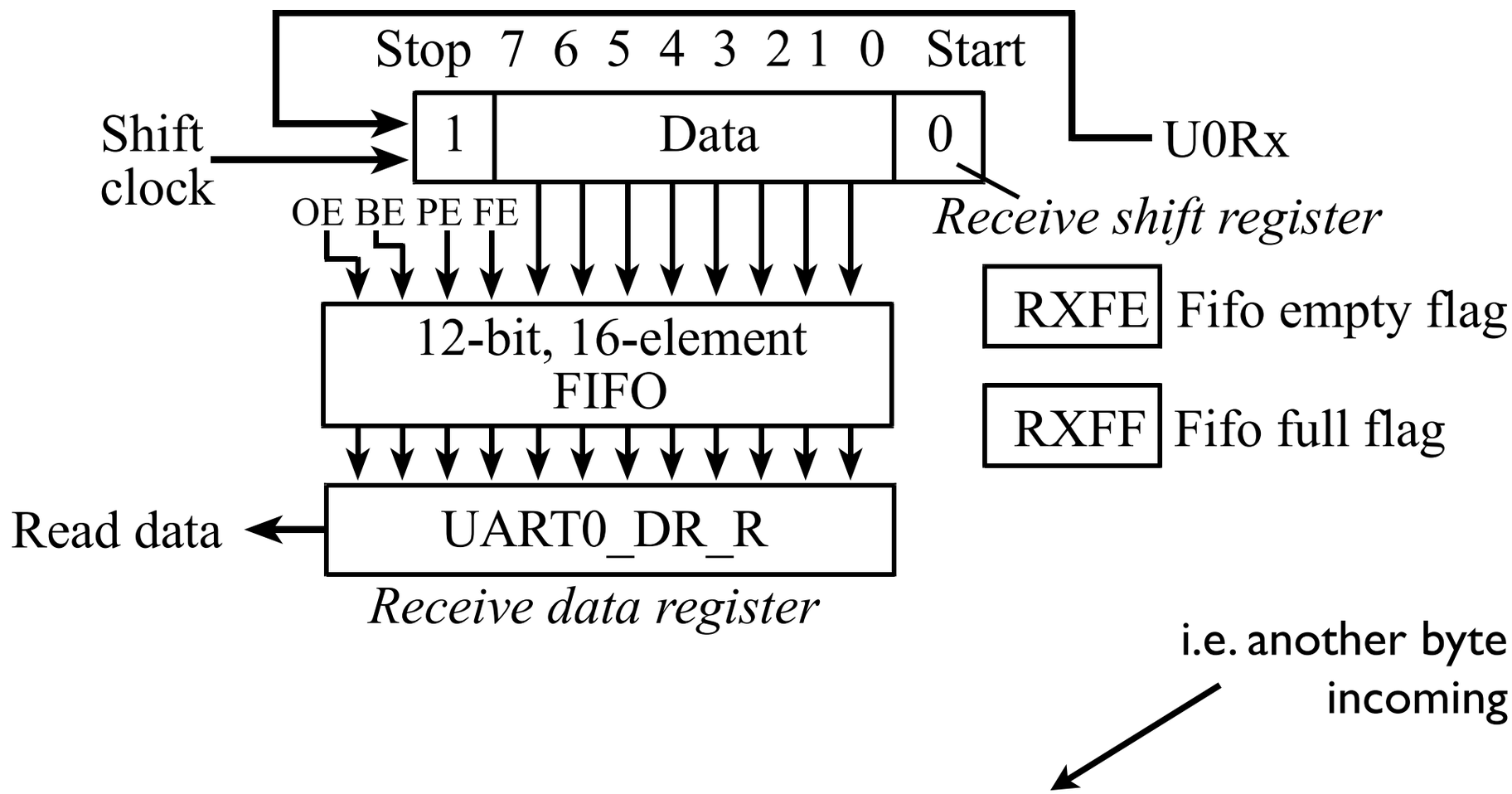


i.e. prev. byte not finished

write to DR register during TX:

1. new data put in first-in first-out queue
2. can store up 16 bytes (can throw 16 bytes at LM3S1968 UARTs)
3. each tx automatically adjusts queue

RX FIFO buffer



read from DR register during RX:

1. new data put in first-in first-out queue
2. can store up 16 bytes (receive 16 bytes at LM3S1968 UARTs)
3. each read automatically adjusts queue

UART Setup

(LM3S1968)

0. setup SysClk
1. enable clock on UART and GPIO peripherals
2. enable alt. func. and pin for GPIO pin
3. disable UART
4. set baud rate divisor
5. set serial parameters
   (length of frame, start/stop/parity bits, FIFO buffer)
6. enable TX/RX and UART

serial i/o: knowing when it's done

1:

a. TX register empty

b. TX buffer empty

UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
Offset 0x018
Type RO, reset 0x0000.0090

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								TXFE	RXFF	TXFF	RXFE	BUSY	reserved		
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

1:

a. TX register full

b. TX buffer full

1:

TXing stuff

serial i/o: knowing when you've got it

1:

a. RX register empty

b. RX buffer empty

UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
Offset 0x018
Type RO, reset 0x0000.0090

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								TXFE	RXFF	TXFF	RXFE	BUSY	reserved		
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

1:

a. RX register full

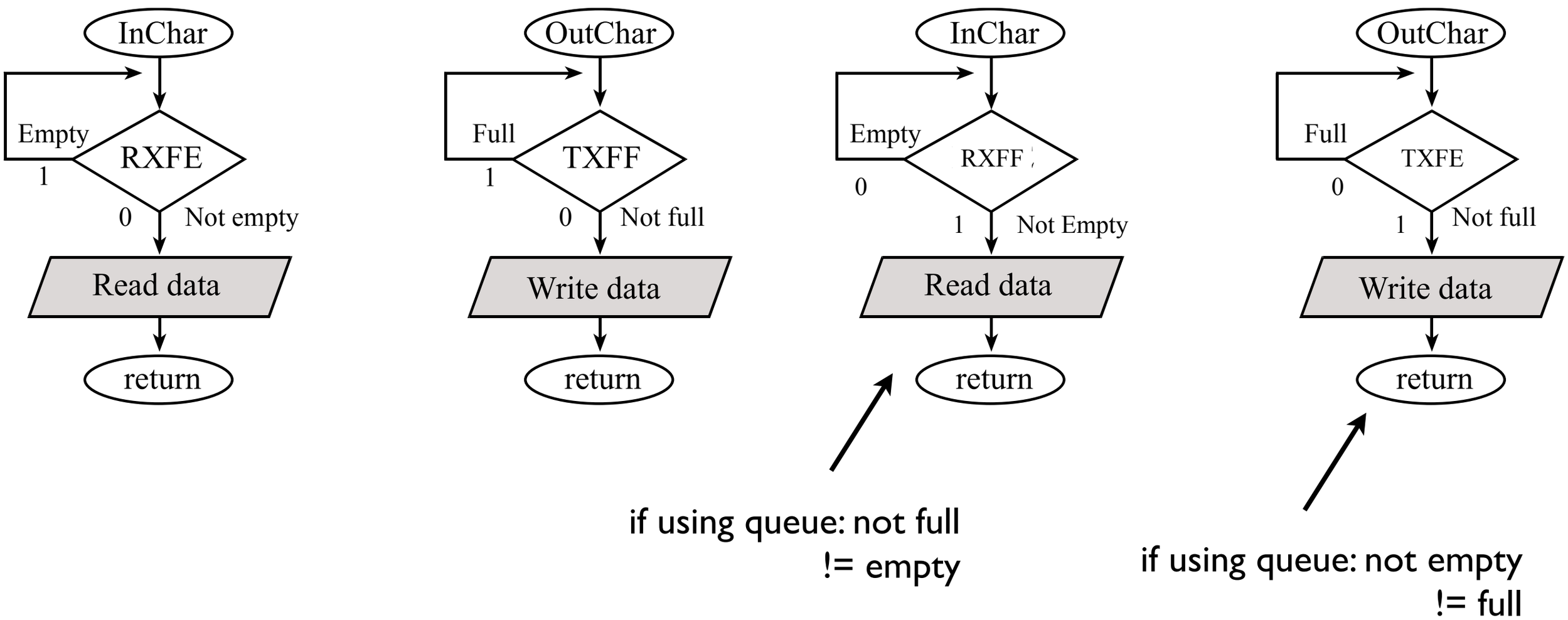
b. RX buffer full

1:

TXing stuff

serial i/o:

1. knowing when you've got it
2. knowing when it's done



polling approach: watch these bits

(dangerous to use `BUSY` for TX)

i.e. depth = 1

note: FIFO queues disabled by default...

but

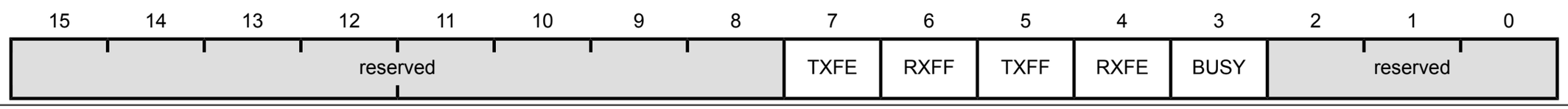
bugs in simulator:

1. `BUSY` bit not set during TX
2. byte being TX'd not in queue

(effectively: FIFO queue depth=2)

example: TX 10101010 then 01010101 by continuously shifting 32-bit value

```
  ;assume UART0 configure for TX
  mov R0,#0xAAAAAAAA ;what we'll TX
TX
  strb R0,[R1,#0x0] ;TX first byte of R0
  ror R0,#1 ;may as well rotate
            ;as TX will take a while
  ; need to wait for uart to
  ; finish TX: tx register empty
wait
  ldr R2,[R1,#0x18] ;get status
  ands R2,#0x80 ;set Z=1 if result is zero
  beq wait      ;branch if Z=1 (TXFE == 1)
  b TX
```



Q: modify wait routine to use TXFF bit

i.e. wait while register/buffer is full