I2C 1

ECE 3710

I busted a mirror and got seven years bad luck, but my lawyer thinks he can get me five.

- Steven Wright

SPI:

3 wires for half-duplexTX/RX

one2one communication

(more wires for CS on shared bus)

still serial

Inter-integrated Circuit Interface (I2C):

2 wires for half-duplex

one2many

(no more wires)

more complex

this is the general trend

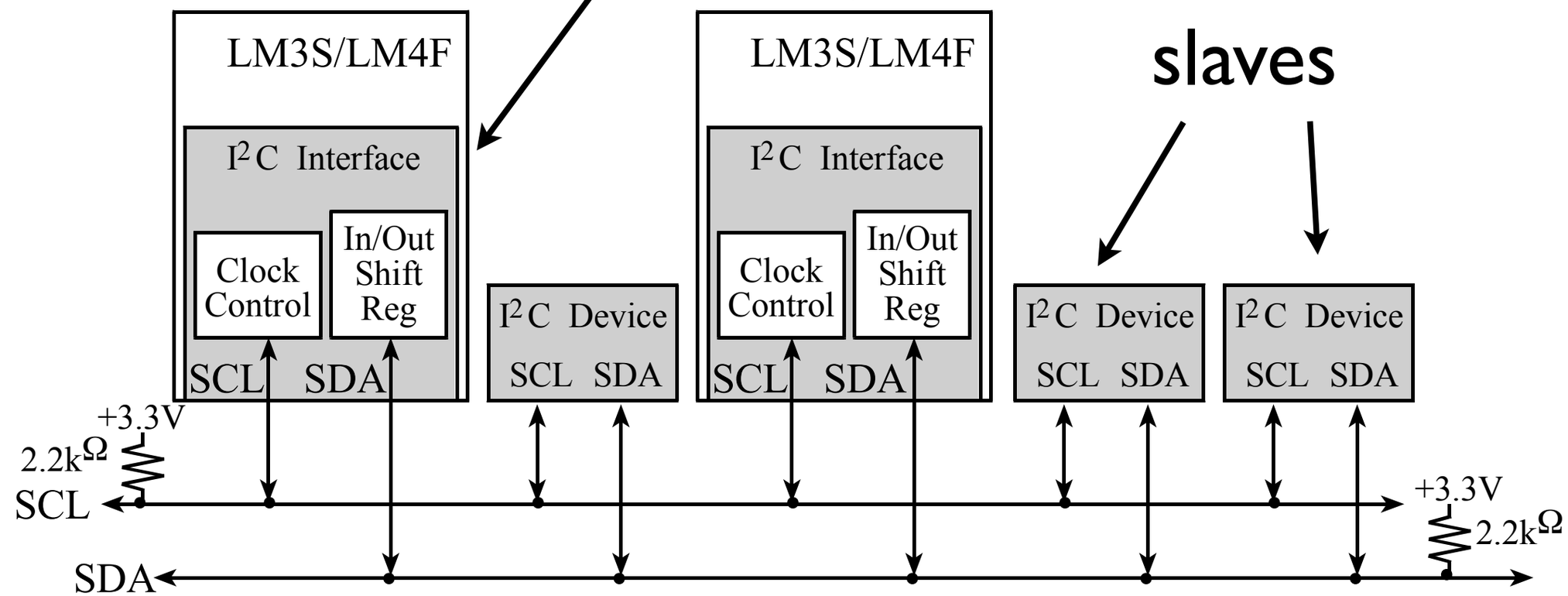
I2C architecture

multiple devices on same bus:

(no CS wires)

masters

slaves



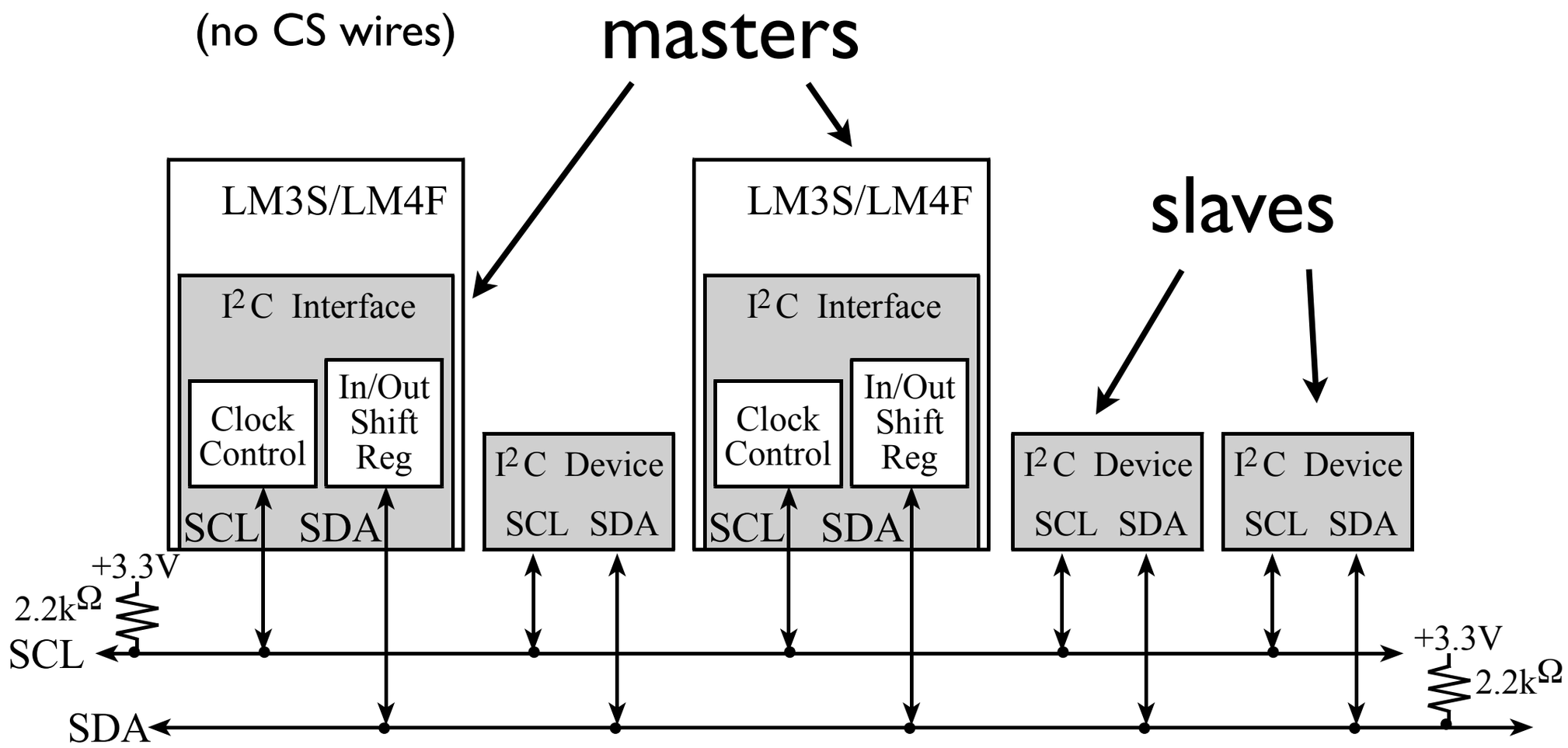
can have more than one → master(s):

1. control clock (generate)
2. control bus (initiate data TX/RX)

I2C architecture

multiple devices on same bus:

(no CS wires)



master-centric:

slaves:

1. do not initiate comm
2. no direct comm. between slaves

1. each has unique address
2. controlled by master

I2C architecture

shared bus:

open-drain
w/pull-up

clock & data

1. default is logic one
2. any device (master or slave) can bring low

stays low until released

note: devices open drain

I2C protocol:

1. initiate comm
(master)
2a. send addr
(which device to comm with)
2b. read or write
(direction of comm: master2slave or slave2master [TX or RX])
3. acknowledge request
(slave)
4. TX/RX data
5. acknowledge data
(recipient of data: master or slave)
6. 4--5 until all data TX'd/RX'd
7. stop/restart
(master)

all communication:

1. 8-bit frames
2. MSB first

I2C protocol

1. initiate comm

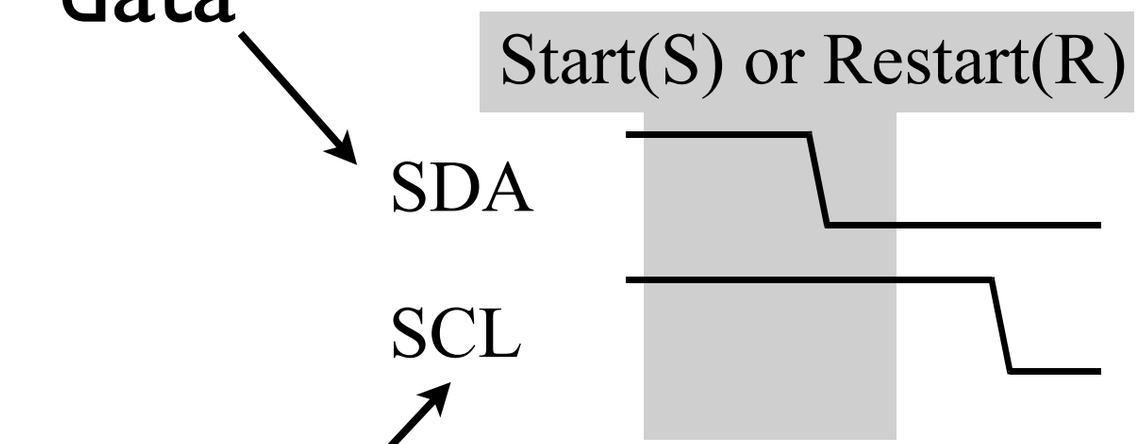
(master)

bring data low

bring clock low

data

clock



I2C protocol

2a. send addr

(which device to comm with)

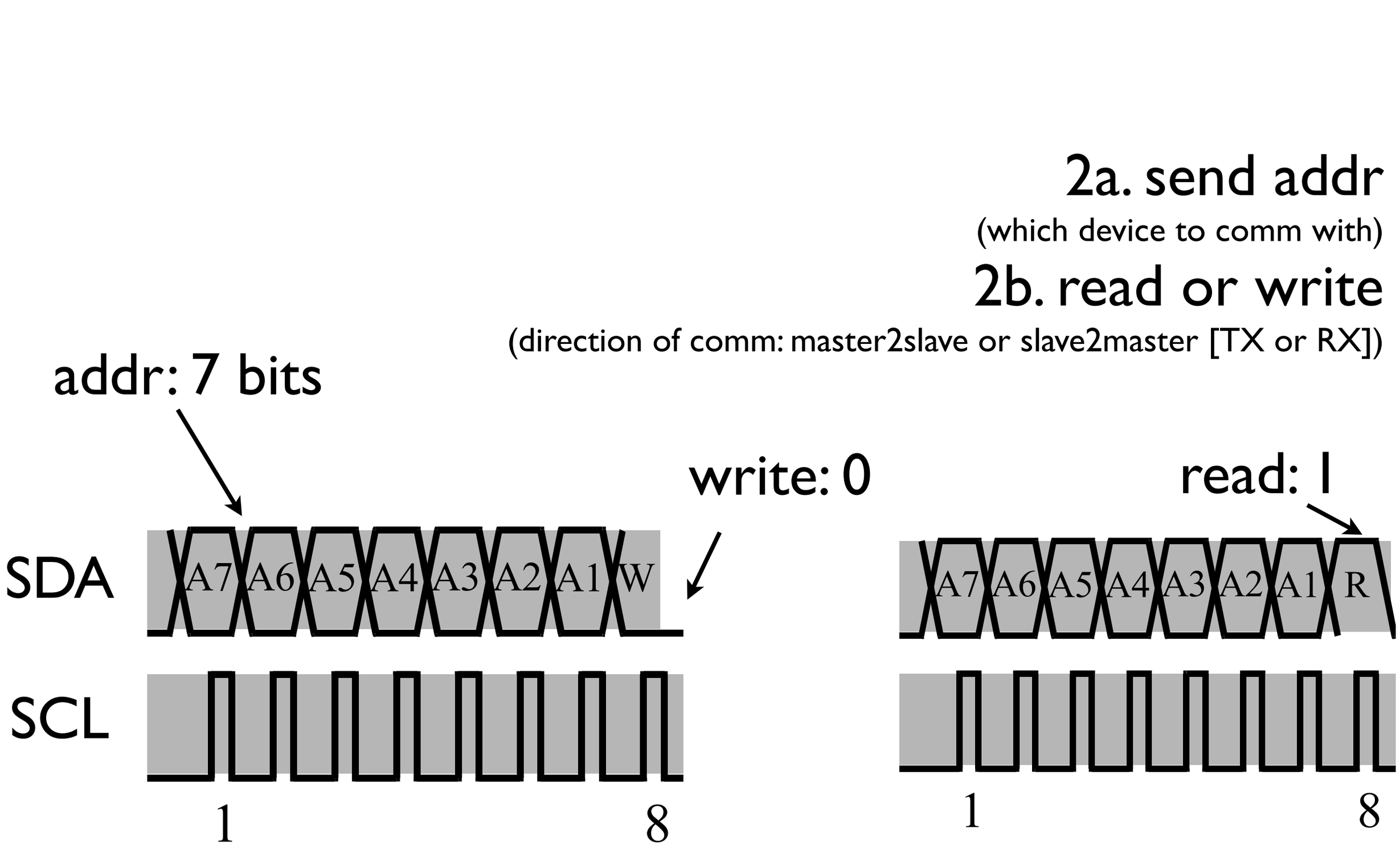
2b. read or write

(direction of comm: master2slave or slave2master [TX or RX])

addr: 7 bits

write: 0

read: 1



addr/data: read when clock is high

note: shaded means master action

I2C protocol

3. acknowledge request
(slave)

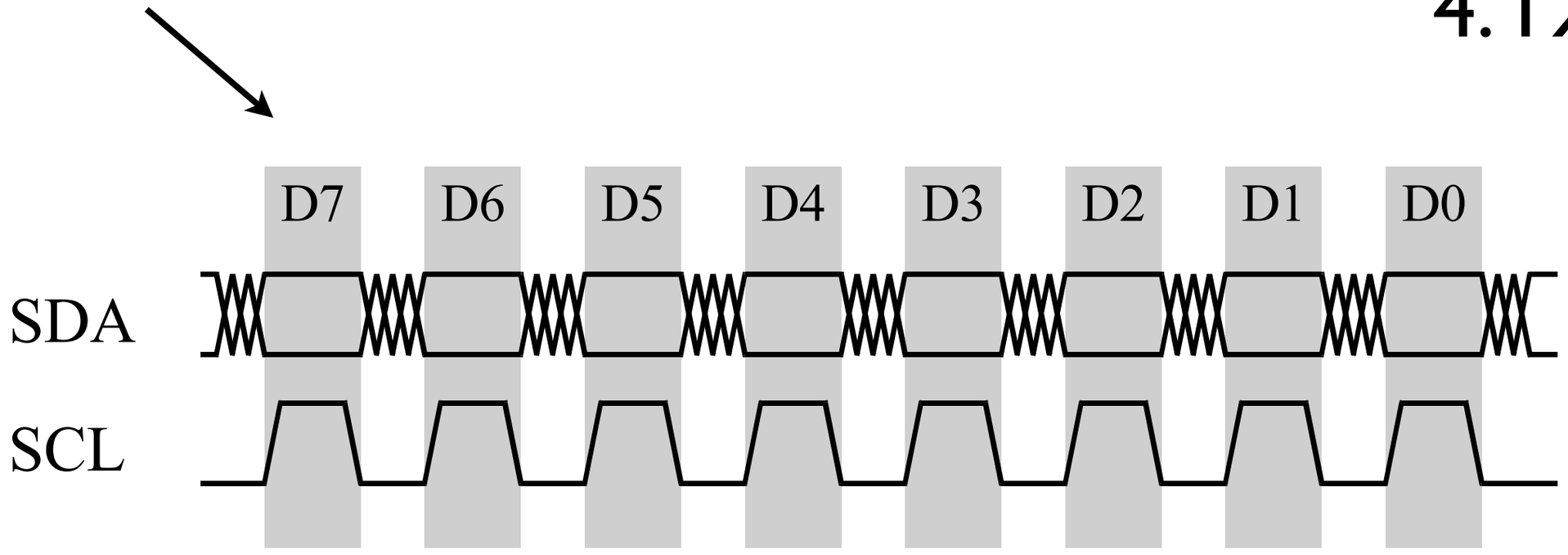
bring data low
leave clock high

if received, master:
1. issue restart
2. send out addr/direction

I2C protocol

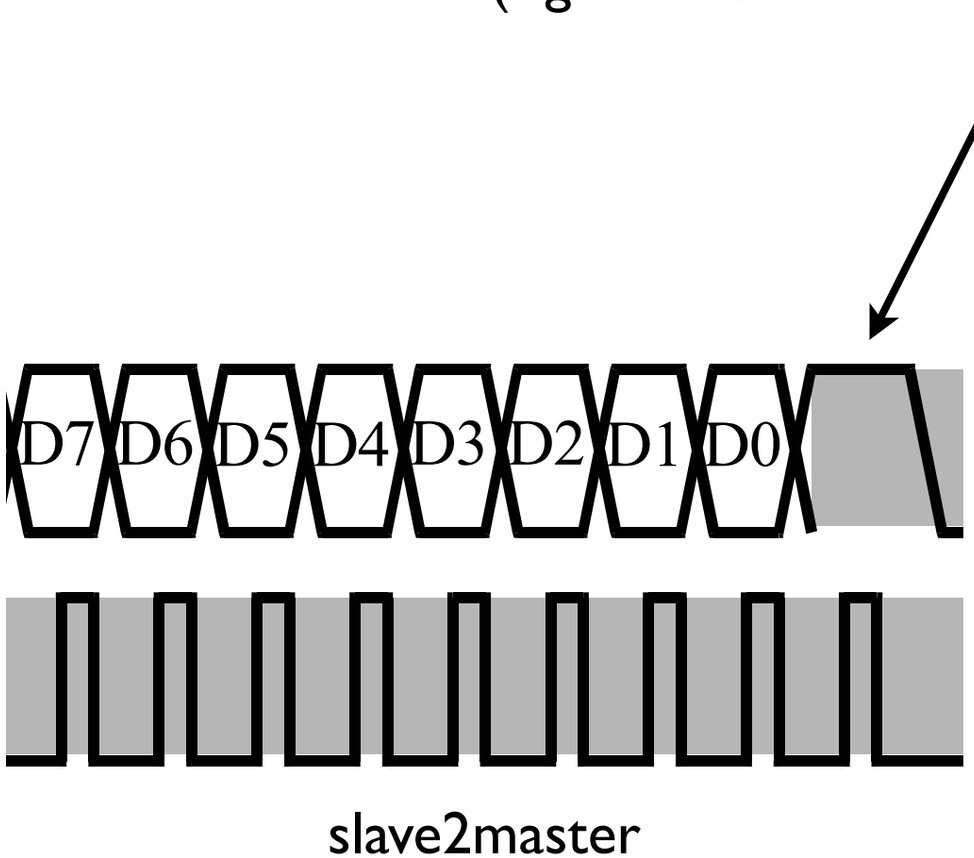
MSB sent first

4. TX data



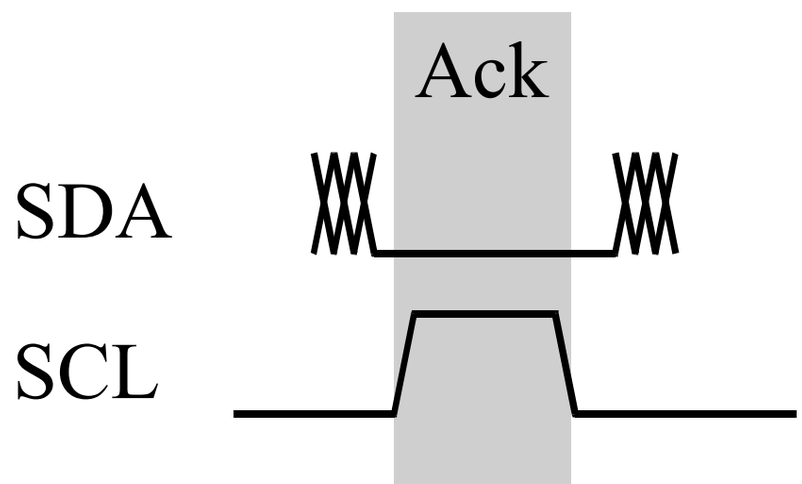
I2C protocol

note: when slave TX last byte,
master does not ack
(signal to slave: end of data)



slave2master

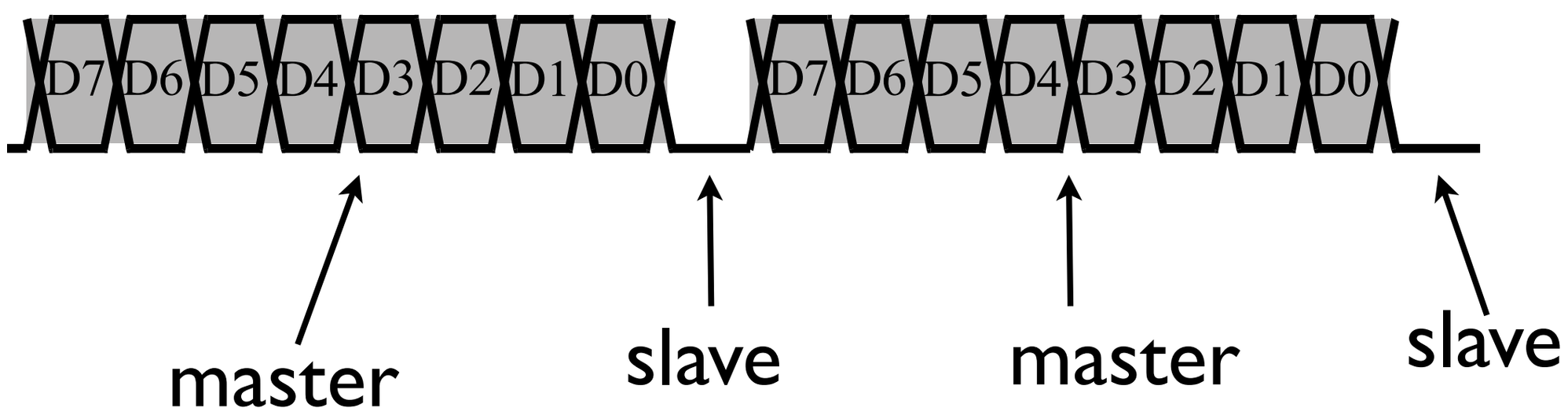
5. acknowledge data
(recipient of data: master or slave)



I2C protocol

TX of multiple bytes:
(master2slave)

6. 4--5 until all data TX'd



I2C protocol

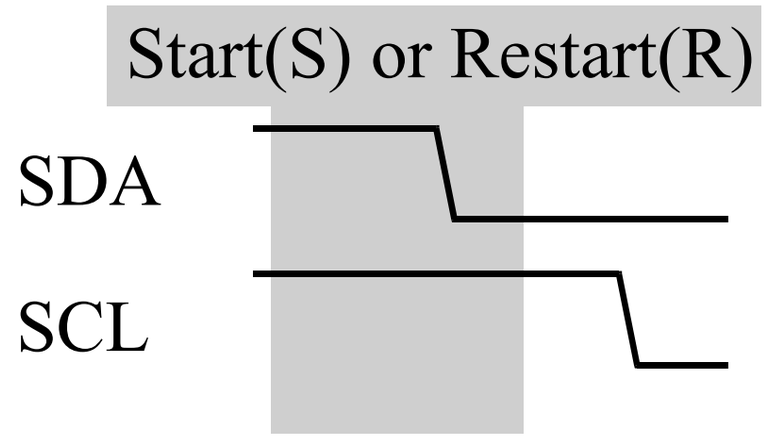
7. stop
(master)

bring clock high
bring data high

I2C protocol

note: if master receiving,
RESTART issued for additional TX/RX:

w/o STOP



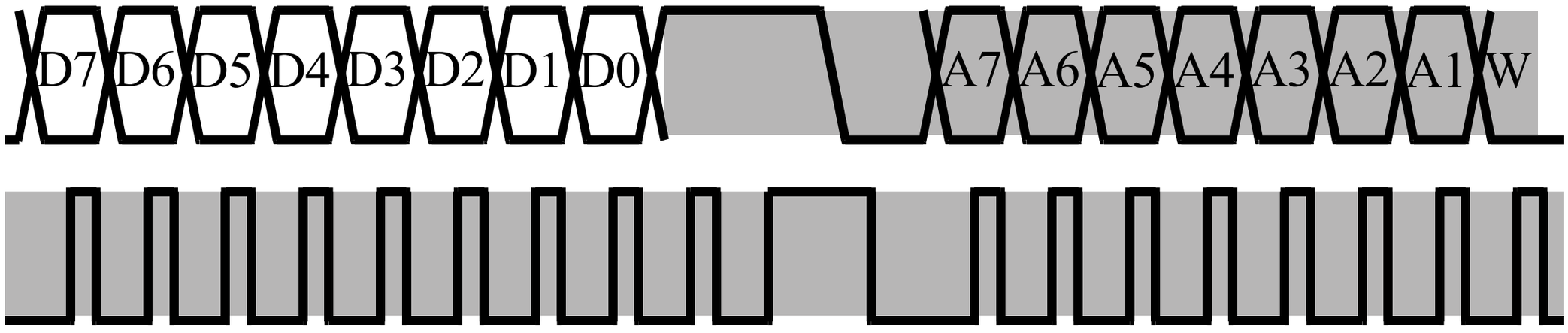
7. restart
(master)

data RX

NACK

restart

addr for data TX

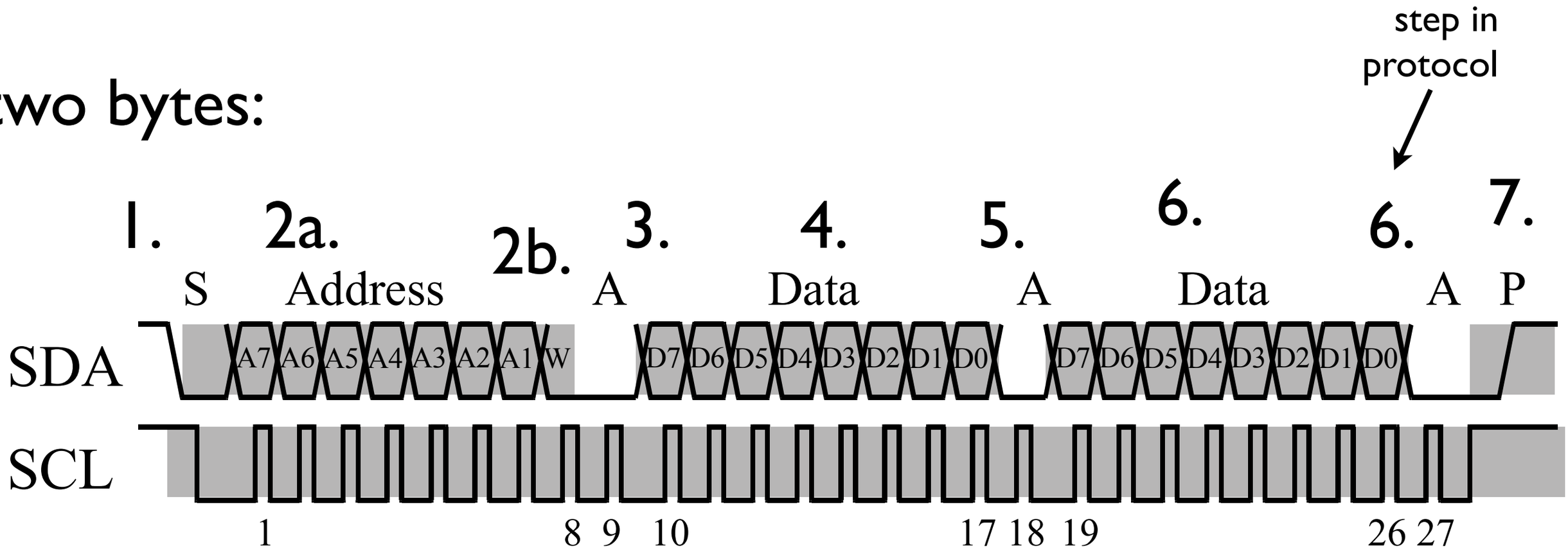


slave2master
then
master2slave

ex: master2slave

(master shaded, slave in white)

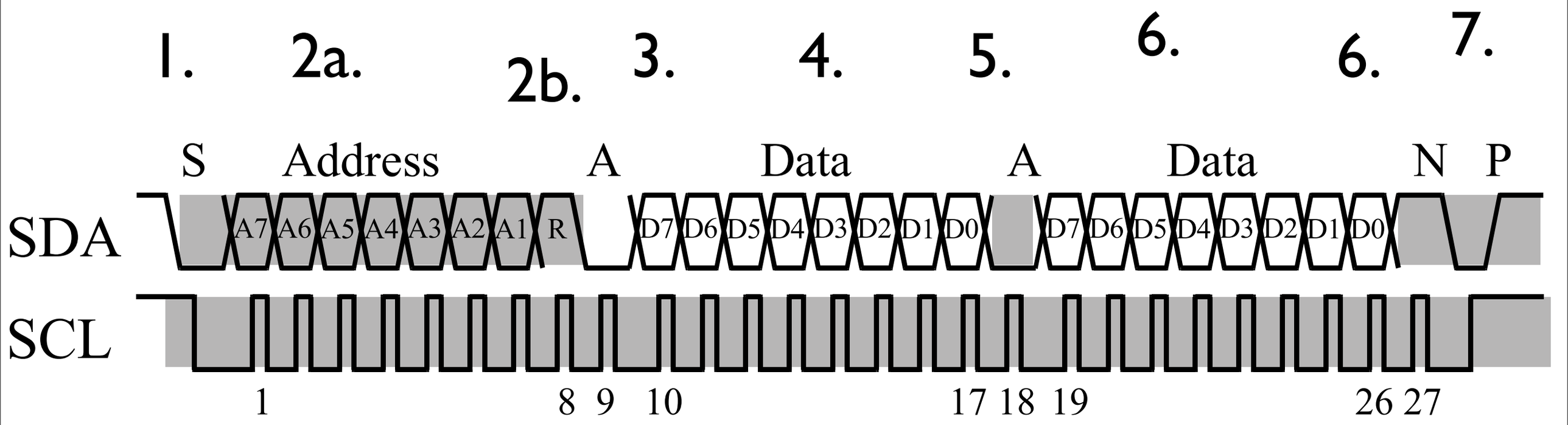
two bytes:



ex: slave2master

(master shaded, slave in white)

two bytes:

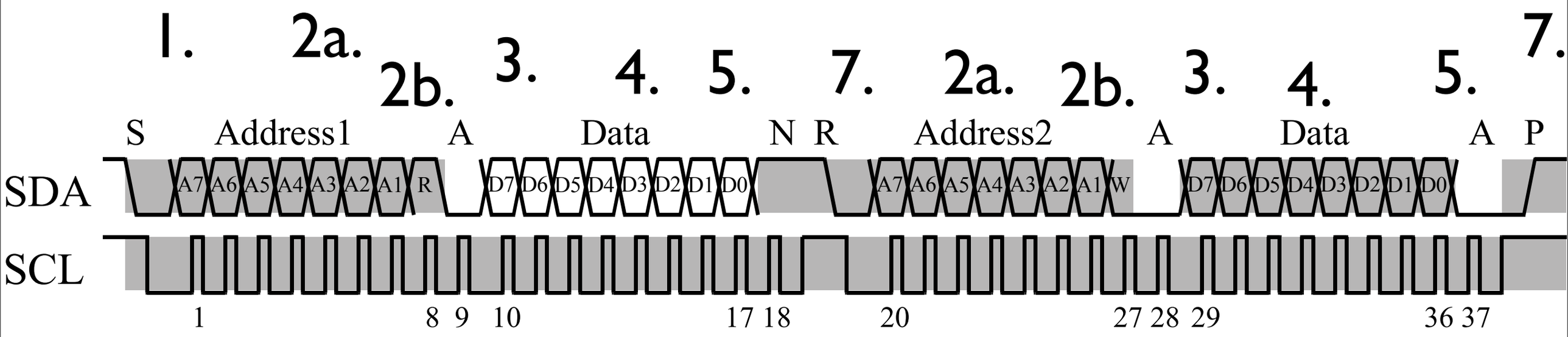


notice NACK after byte two

ex: slave2master then master2slave

(master shaded, slave in white)

two bytes:



notice NACK and RESTART after byte one

(no STOP)

Q:

1. clock too fast for slave?
2. multiple masters sending at same time?

1. clock stretching

slave holds clock line low
until data TX/RX

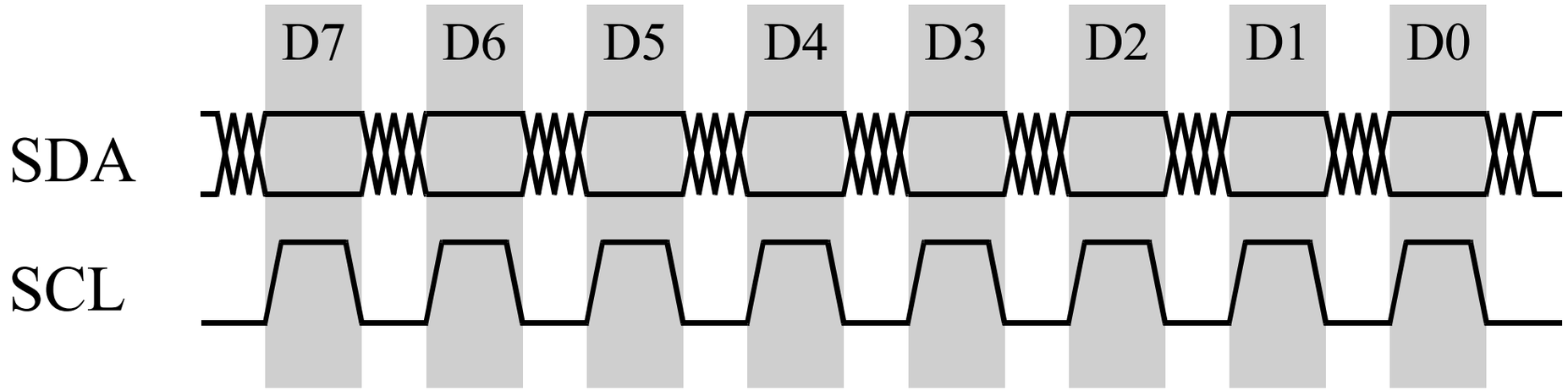
master sets clock high (lets float)
but must wait for it

remember: open drain

(any device can keep lines low w/o consent of others)

1. clock stretching

slave holds clock line low
until data TX/RX
(delay 0->1 only)



master sequence (TX):

1. bring clock low
2. set data
3. wait
4. set clock high
5. wait for clock high
6. wait
7. repeat/stop waiting if clock low

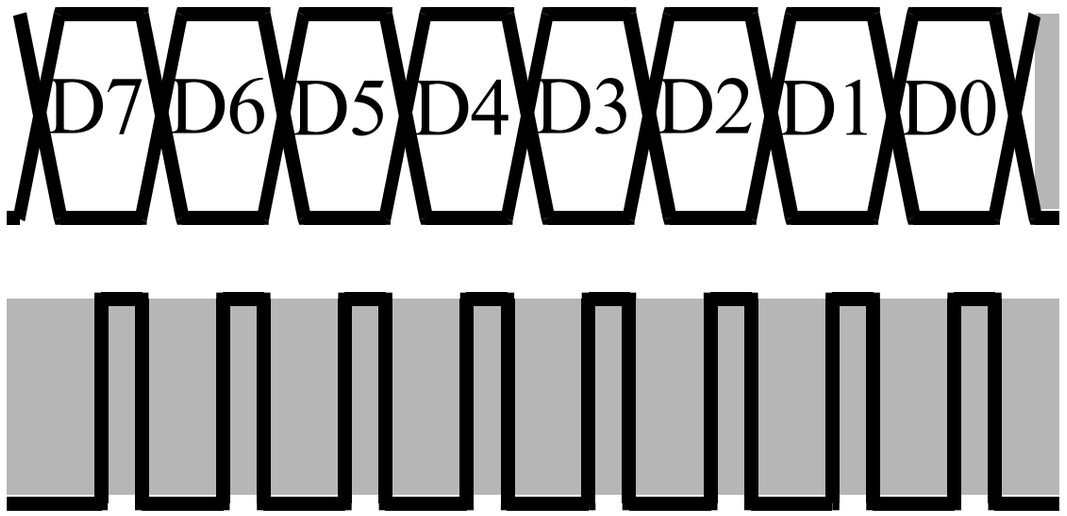
i.e. release clock

this is where slave can hold low

e.g. another master grabs bus

1. clock stretching

slave holds clock line low
until data TX/RX
(delay 0->1 only)



master sequence (RX):

1. clock low
2. wait
4. clock high
5. wait for clock high
6. get data
7. wait
8. repeat/stop waiting clock low

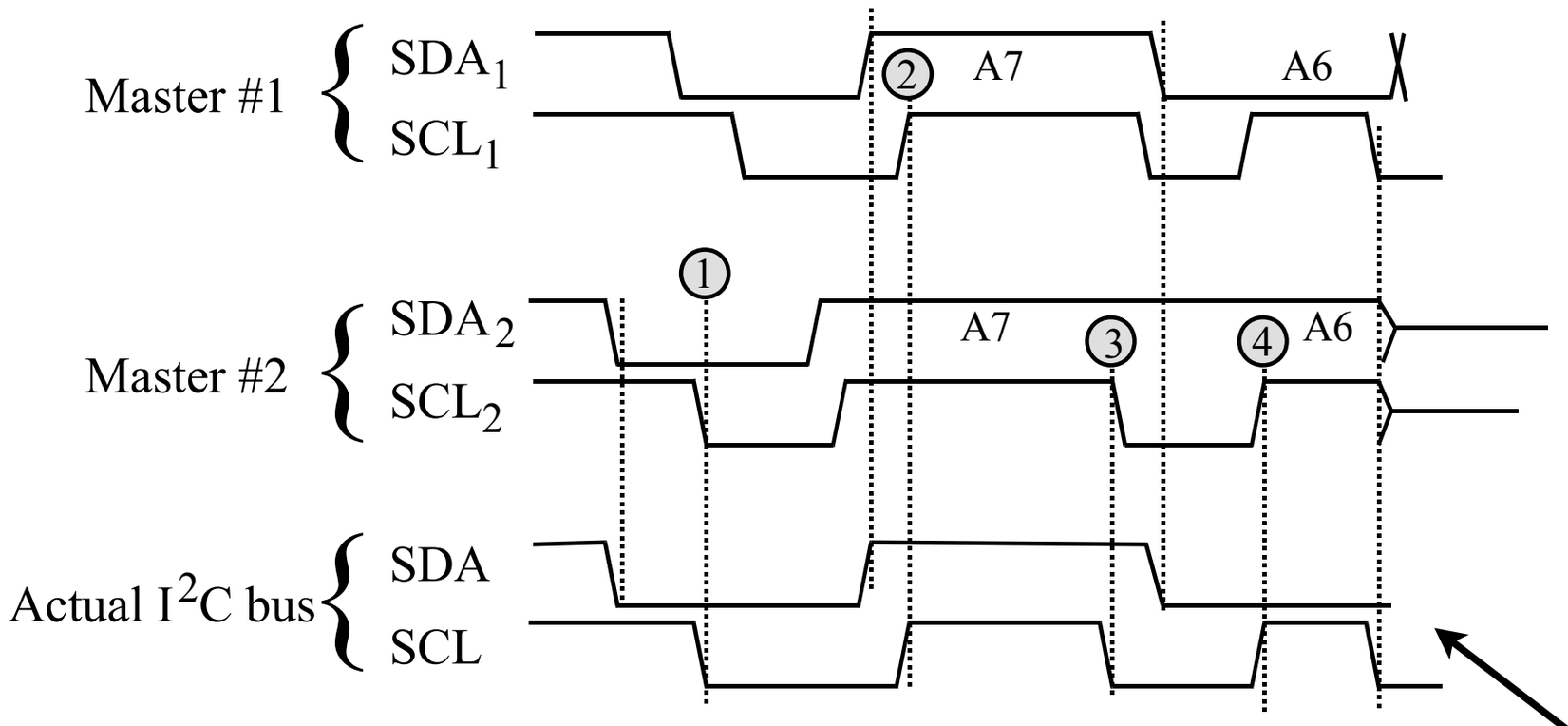
this is where
slave can hold low

2. multiple masters

1. different clocks
2. line control

everything gets AND'd:

(beginning of comm)



START

lowest addr wins
(master one)

note: master must check
to see that it's still in control