

Lab 3: Talking DNS

Checkpoint due: Monday, October 19, 11:59 PM ET

Due: Monday, October 26, 11:59 PM ET

Let's look at what we are implementing...

Difference between Lab 1&2 and Lab 3

1. You can use C or Python. You cannot use functions that simplify DNS or hide details of socket programming. For example, do not make calls to `gethostname()` or `getaddrinfo()` – these do DNS lookups. If you're unsure if you can use a function, ask.
2. You will use UDP sockets instead of TCP sockets
3. DNS is a binary protocol. You cannot use string functions to look at any part of the DNS. Wireshark will be helpful for this!

Program overview

- Query types
 - Should support Type A and Type MX if passed the `-m` flag
 - You may assume NS records in authority section have matching A records in additional section
- Query status message
 - Program should print short message about its intermediate steps
- Query timeout
 - After waiting for 5 seconds with no response to your query, you should timeout
- You should never ask a DNS server to perform a recursive query for you

Program workflow

1. Check the arguments to determine the type of query
2. Build a query [You will have to read the DNS RFC for this]
3. UDP socket calls to send request using port 53
4. Receive response per request (use `struct.pack`)
5. Parse response message (most of the work is here) [You will have to read the DNS RFC for this]
 - If you don't have an answer, you need to query the next NS and the next until you do
 - In Python have to use `struct.unpack`
6. Continue process down hierarchy until you get an A record with answer or an error
7. Return resolved IP address for an A record

Week 1 Goal:

Steps 1-6:

Hostname-to-IP resolution: Successfully navigate the DNS hierarchy, parse NS records, and reach the authoritative DNS name server

How to read DNS RFC 1035

- Look for Section 4: Message format
 - The message format is the same for requests and responses, but what will be in the message will be different and the header will tell us what to expect in the message
 - The header tells you what to expect in the rest of the reply
 - **Look particularly at 4.1.4 Message compression** when trying to parse reply
- Look for Section 3.2: Resource Record formats
- Octet = 1 byte = 8 bits

Making DNS request...

Header:

- Set ID to some 16-bit value
- Set all flags to 0
- Set QDCOUNT to 1

In Question Section:

- Add a question

That's it!

Byte ordering

If you have two bytes: $0x0120 = 01\ 20$
which one do we send first?

Little endian: 20 01

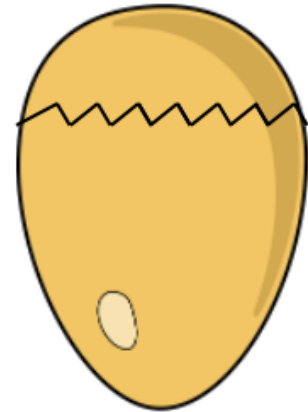
Big endian: 01 20

`struct_unpack` will
put data in the right
order for you

See starter code `struct_example.py`



Little Endian: Host byte order
Data representation on x86 chips
LSB -> MSB



Big Endian: Network byte order
Data representation sent/arriving
over the network
MSB -> LSB

How to use dig

```
dig . ns
```

```
dig +noredc diveintosystems.org @a.root-servers.net
```

```
dig +noredc diveintosystems.org @a2.org.afiliat-nst.info
```

```
dig +noredc diveintosystems.org @ns-1274.awsdns-31.org
```

How to use Wireshark...

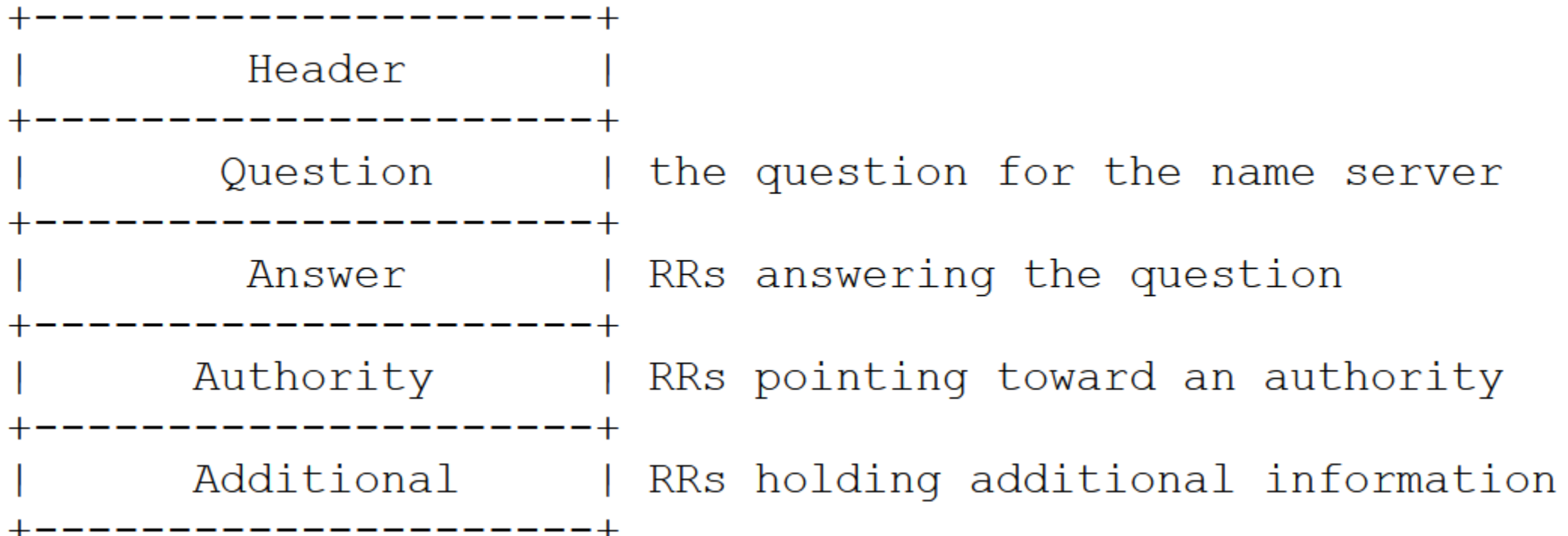
- It will show all the traffic going through the network
- Set a filter in the filter bar to focus on DNS
 - “dns” or “dns.qry.name contains "diveintosystems”” to look only at DNS queries for a specific domain
- You can look at the headers and see where bytes are set

Parsing DNS responses...

- ID will be the same as the request
- QDCOUNT will be the same
- Number of answer, authority, additional records will vary depending on the kind of server. Root servers for example will not have any answers and will have some number of authority and additional records
- The most challenging part is you'll need to deal with message compression...

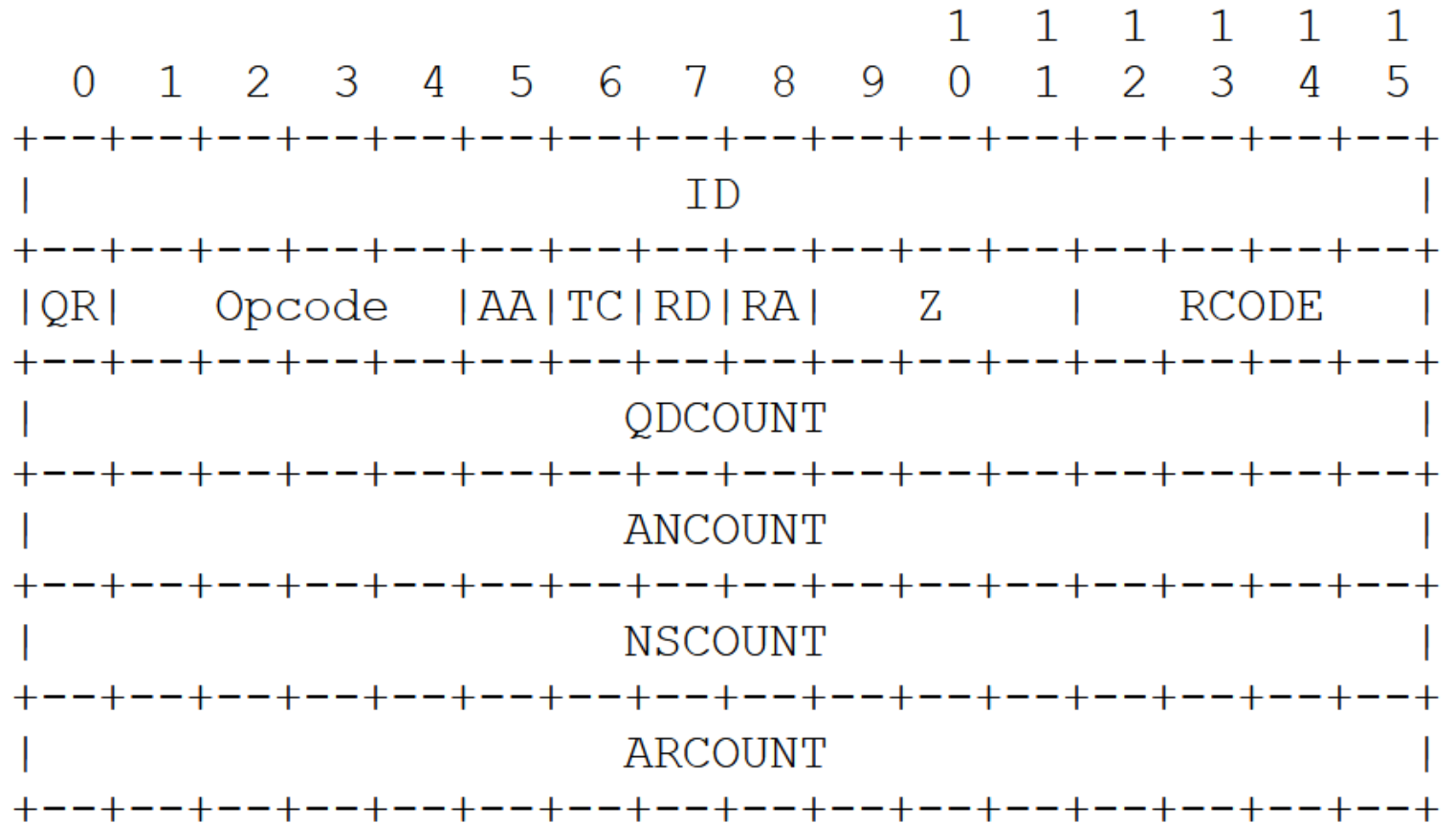
Parsing Responses

- So, you have a bunch of bytes...



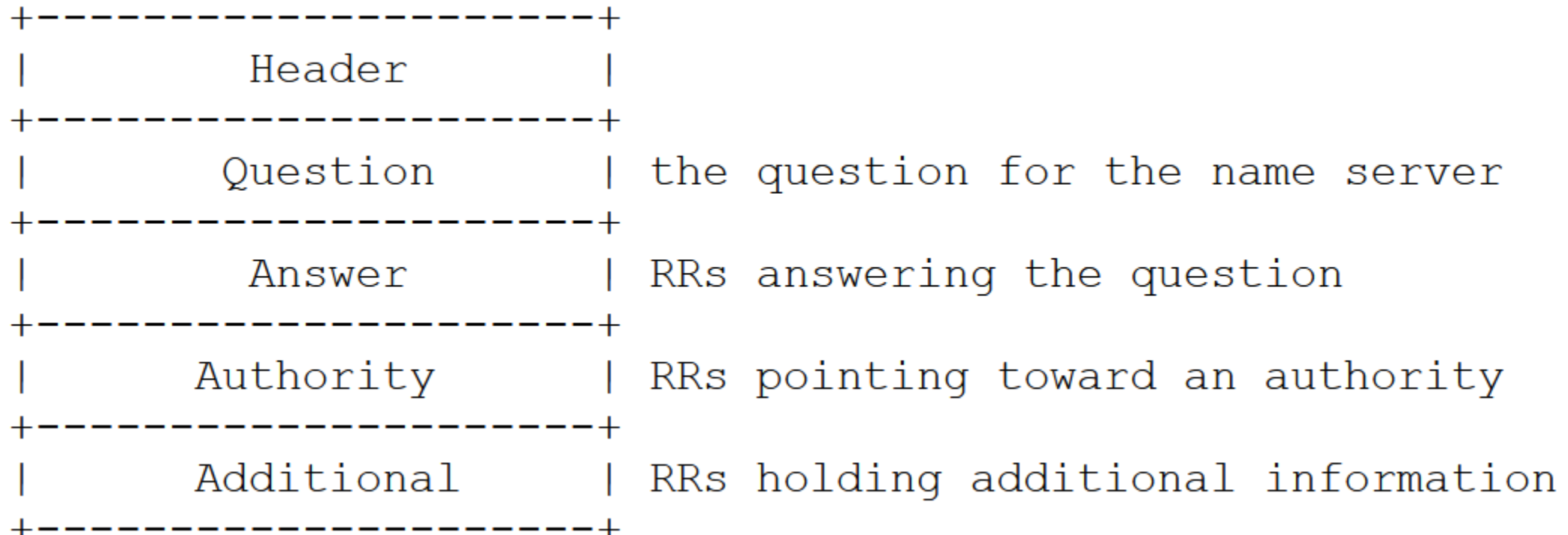
Parsing Header

- Same as the header you built, in reverse
- If you want to detect errors early, look at RCODE. Also ok to just look at the type of answer you get.



Parsing Responses

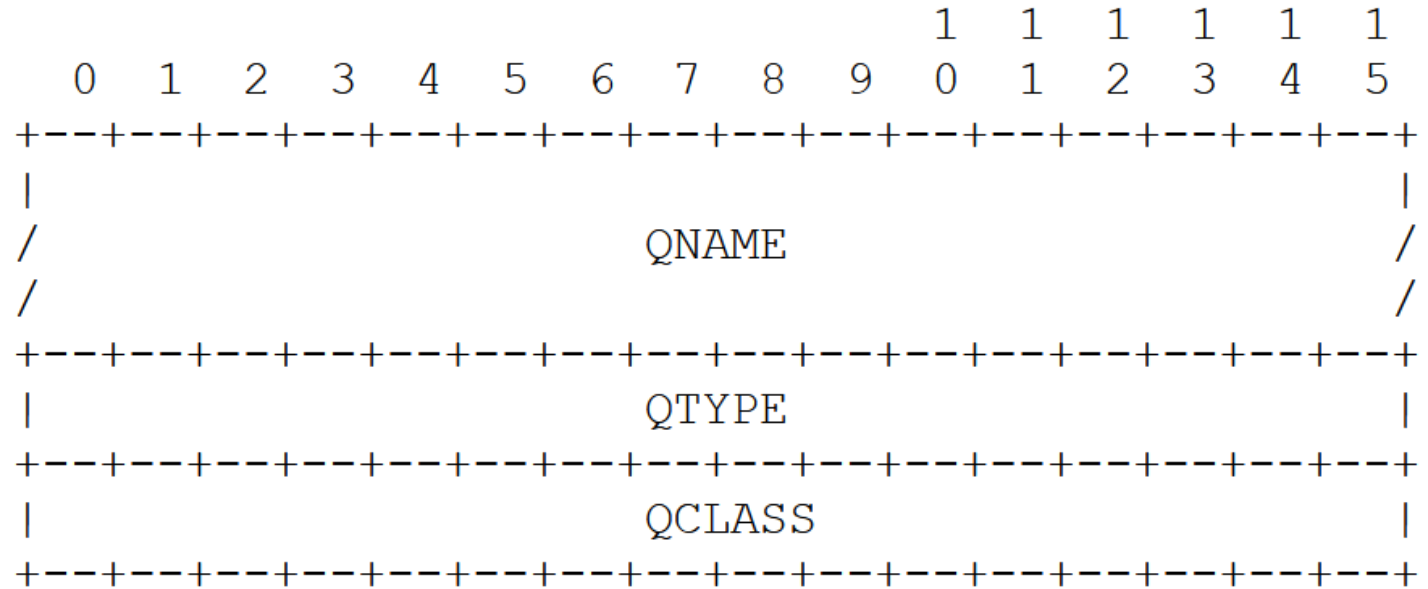
- So, you have a bunch of bytes...



Parsing Questions

- Note the // on the border of QNAME – variable length

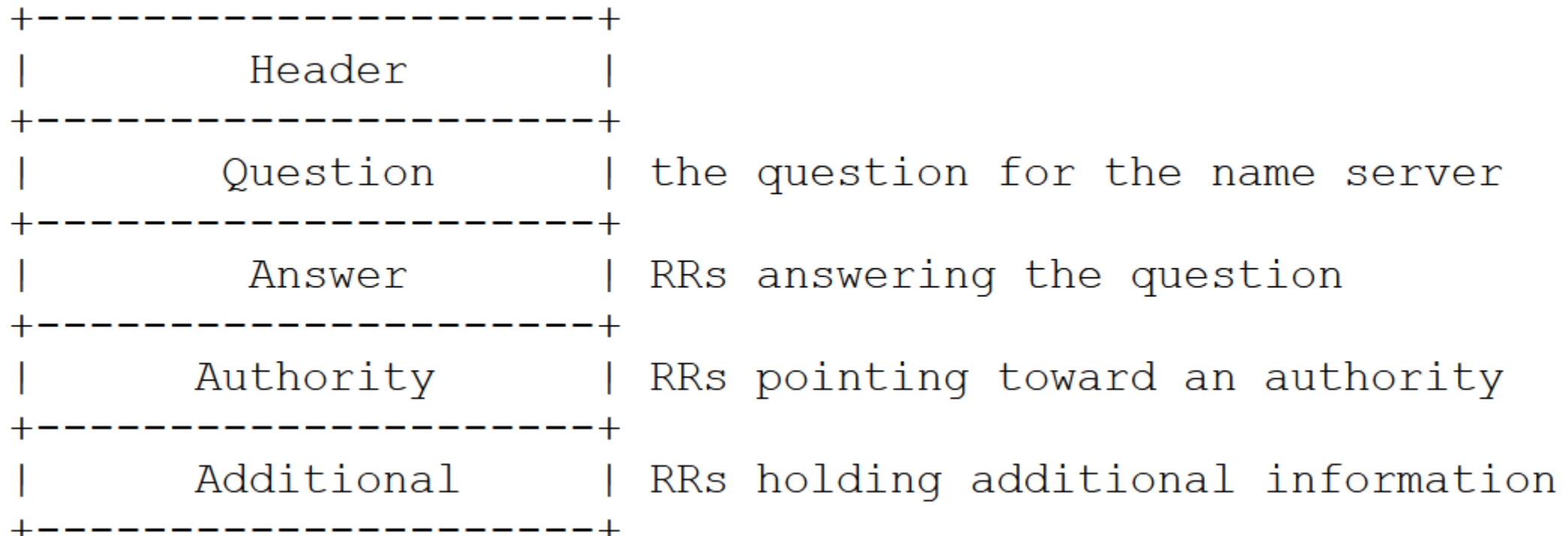
- You don't really care about what the question contains



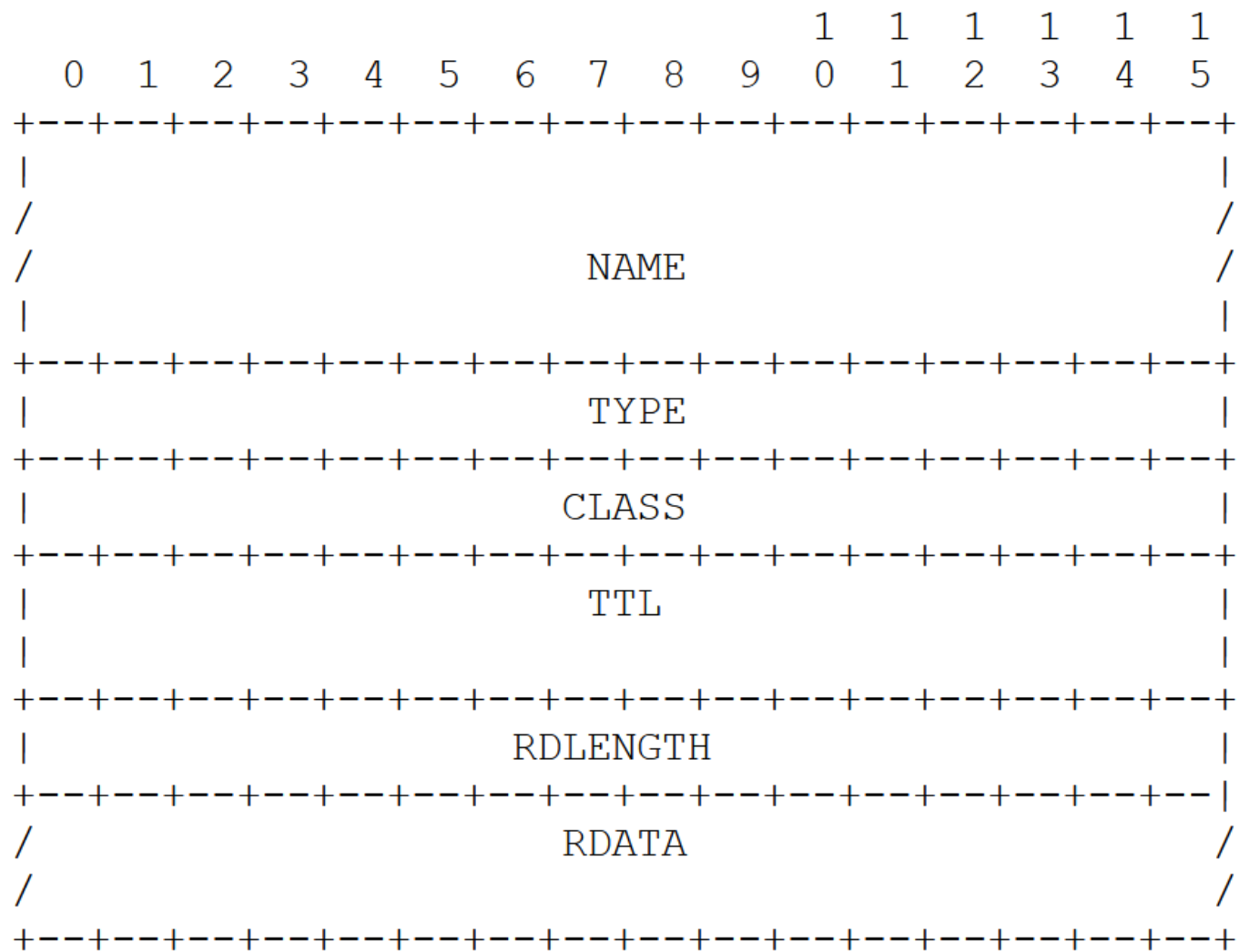
- You MUST know how many bytes are in the question to move beyond it

Parsing Responses

- So, you have a bunch of bytes...



Parsing Resource Records



Names

4.1.4. Message compression

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurrence of the same name.

The pointer takes the form of a two octet sequence:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 1  1|                                OFFSET                          |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less. (The 10 and 01 combinations are reserved for future use.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.

The compression scheme allows a domain name in a message to be represented as either:

- a sequence of labels ending in a zero octet
- a pointer
- a sequence of labels ending with a pointer

- The following is a real example of the response that I got from running:

```
dig @dns.cs.swarthmore.edu demo.cs.swarthmore.edu
```

Parsing Names (no pointers)

 Binary

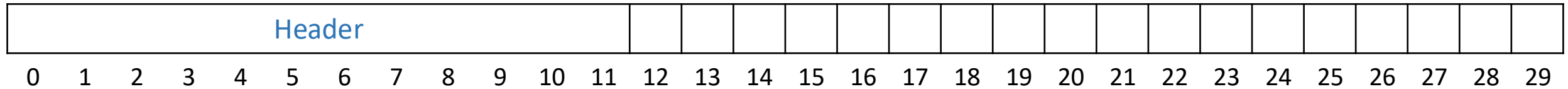
ASCII

[illegible]

Parsing Names (no pointers)

 Binary

ASCII

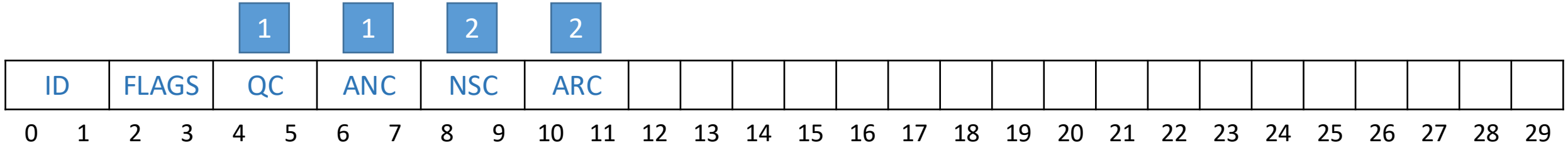


Parsing Names (no pointers)



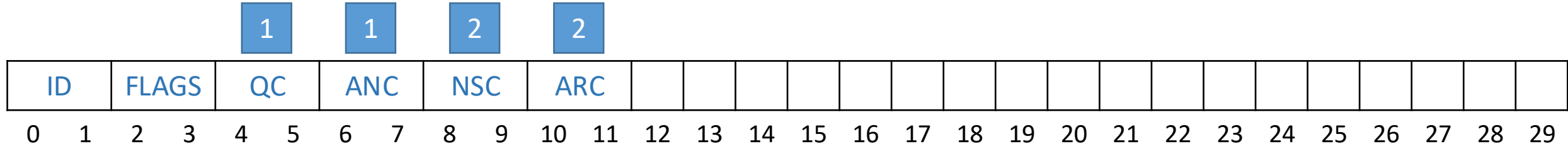
ID		FLAGS		QC		ANC		NSC		ARC																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Parsing Names (no pointers)



```
parse_question(response, 12)
```

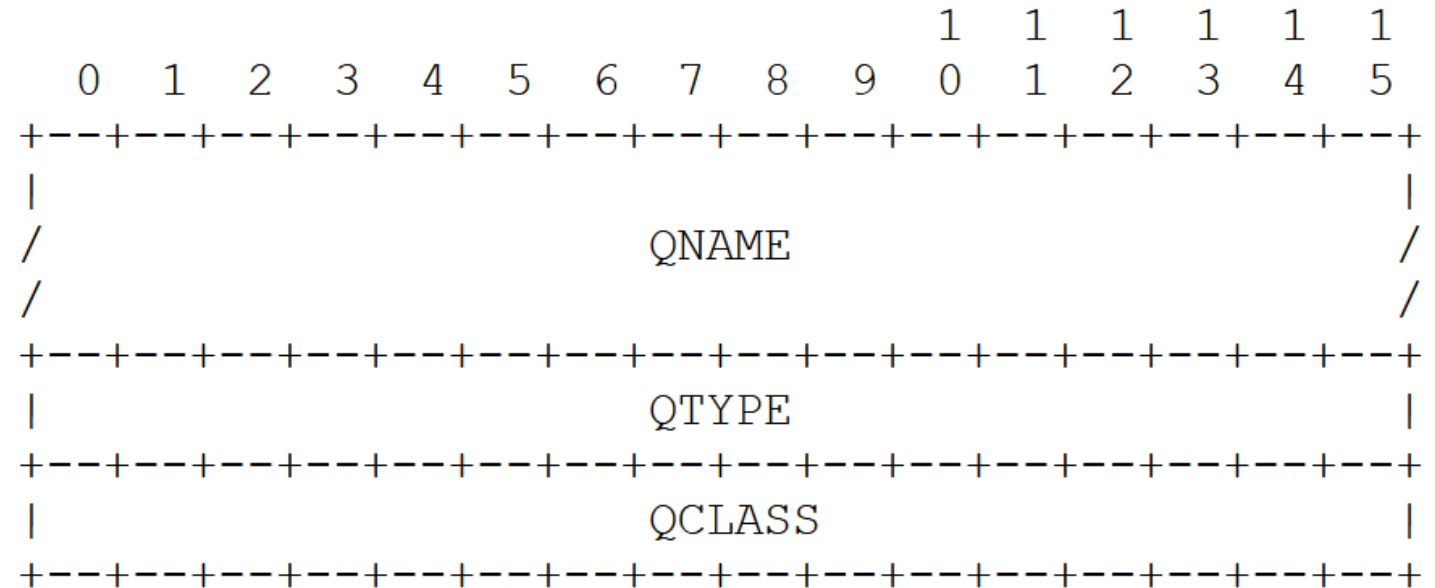
Parsing Names (no pointers)



parse_question(response, 12)

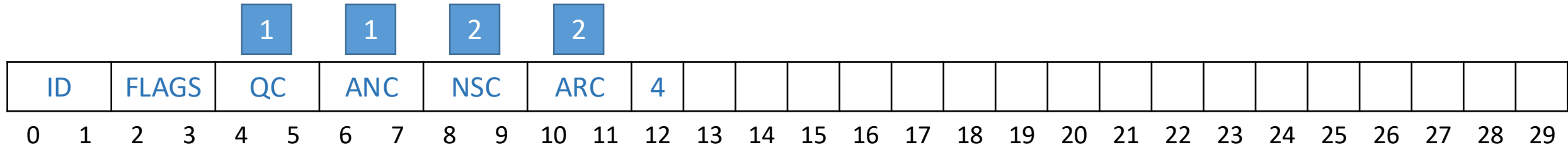
parse_name(response, 12)

Name so far:



Parsing Names (no pointers)

 Binary
 ASCII



Unpack one-byte, unsigned integer: !B

Does this byte have the two high order bits set to 1?

If yes, pointer

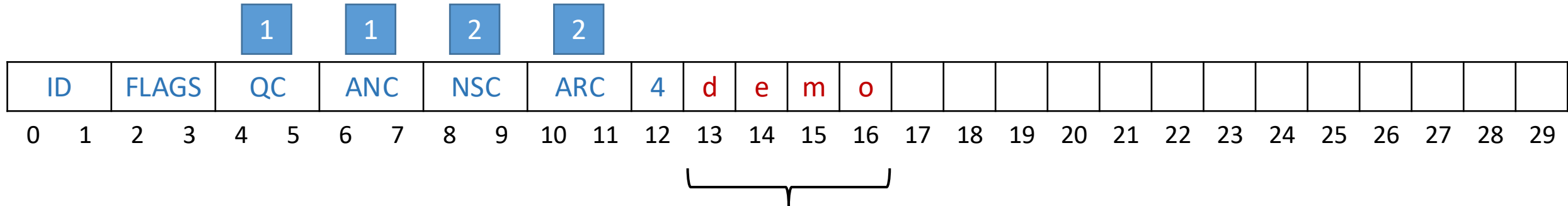
This case: no, not a pointer

Is it 0? If so, end of name

Name so far:

Parsing Names (no pointers)

 Binary
 ASCII

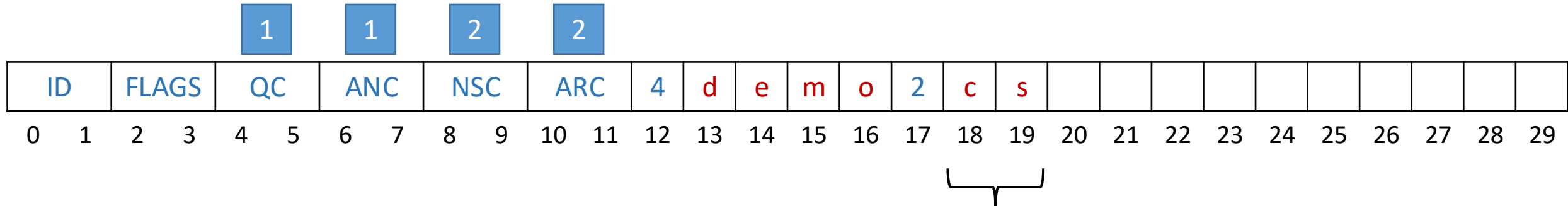


Next 4 characters are one
"label" or chunk of the name.

Name so far: demo

Parsing Names (no pointers)

 Binary
 ASCII



Name so far: demo.cs

Parsing Names (no pointers)

 Binary
 ASCII

ID		FLAGS		QC		ANC		NSC		ARC		4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

e																														
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	

Name so far: demo.cs.swarthmore

Parsing Names (no pointers)

 Binary
 ASCII

ID		FLAGS		QC		ANC		NSC		ARC		4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

e	3	e	d	u																									
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

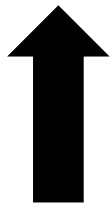
Name so far: demo.cs.swarthmore.edu

Parsing Names (no pointers)

 Binary
 ASCII

				1			1			2			2																
ID	FLAGS	QC	ANC	NSC	ARC	4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

e	3	e	d	u	0																								
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59



Binary 0 means end of name

Name so far: demo.cs.swarthmore.edu

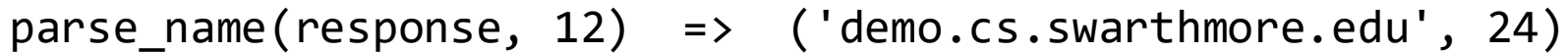
Parsing Names (no pointers)

 Binary
 ASCII

				1			1			2			2																
ID	FLAGS	QC	ANC	NSC	ARC	4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
e	3	e	d	u	0																								
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

`parse_name(response, 12) => ('demo.cs.swarthmore.edu', 24)`

- Binary
- ASCII



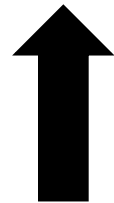
```
parse_question(response, 12) => 28
```

Parsing Names (no pointers)

 Binary
 ASCII

				1			1			2			2																
ID	FLAGS	QC	ANC	NSC	ARC	4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

e	3	e	d	u	0	Q	T	Y	P	Q	C	L	S																		
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59		



This must be the start of an answer resource record (RR).

Parsing Names (with pointers)

 Binary

ASCII

				1		1		2		2			
ID		FLAGS		QC		ANC		NSC		ARC		4	d
0	1	2	3	4	5	6	7	8	9	10	11	12	13

e	3	e	d	u	0	Q T Y P	Q C L S						
30	31	32	33	34	35	36	37	38	39	40	41	42	43

```
parse_resource(response, 40)
```

[illegible]

Parsing Names (with pointers)

 Binary

 ASCII

				1		1		2		2			
ID		FLAGS		QC		ANC		NSC		ARC		4	d
0	1	2	3	4	5	6	7	8	9	10	11	12	13
e		3	e	d	u	0	Q	T	Y	P	Q	C	L
30	31	32	33	34	35	36	37	38	39	40	41	42	43

```
parse_resource(response, 40)
```

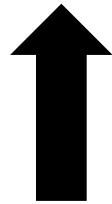
```
parse_name(response, 40)
```

[illegible]

Parsing Names (with pointers)



				1		1		2		2																					
ID		FLAGS		QC		ANC		NSC		ARC		4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
e	3	e	d	u	0	Q TYP		QCLS		192																					
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59		

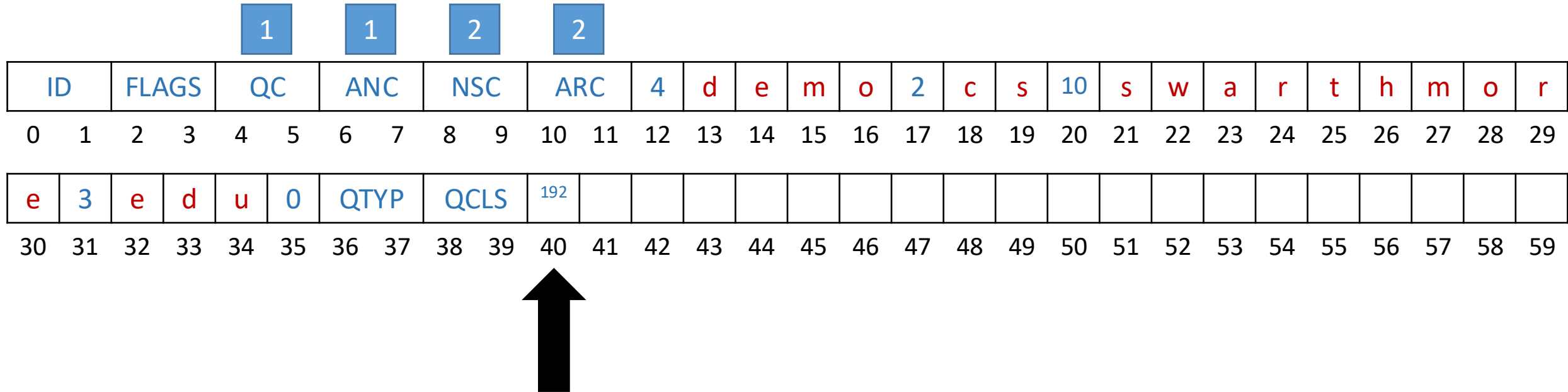


Unpack one-byte, unsigned integer: !B

Does this byte have the two high
order bits set to 1?

Name so far:

Parsing Names (with pointers)



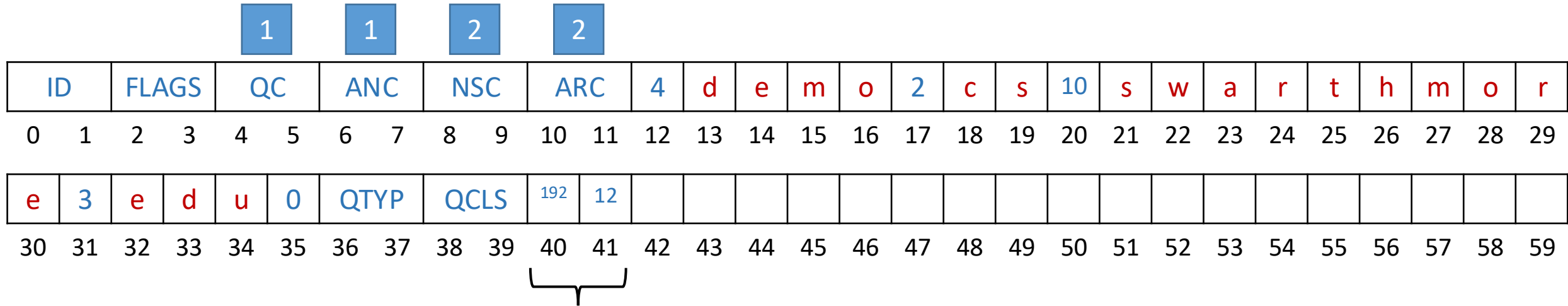
Unpack one-byte, unsigned integer: !B

Does this byte have the two high order bits set to 1?

192 in binary: 11000000

Name so far:

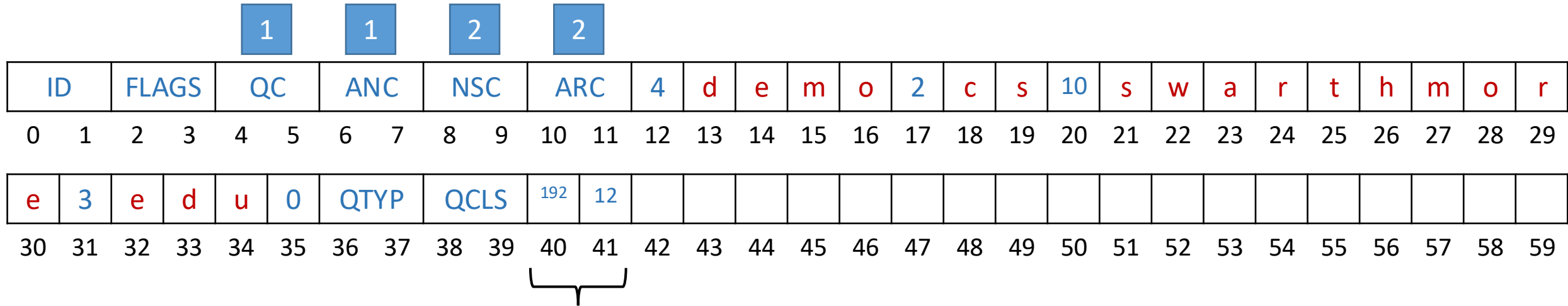
Parsing Names (with pointers)



Take the first type (with high-order 11 bits) and the next byte and treat them as a two-byte value: unpack them with !H

Name so far:

Parsing Names (with pointers)



Take the first type (with high-order 11 bits) and the next byte and treat them as a two-byte value: unpack them with !H

Together, these bytes are: 11000000 00001100

Because the first two bits indicate "pointer", we ignore them: ~~00~~000000 00001100

Name so far:

Parsing Names (with pointers)



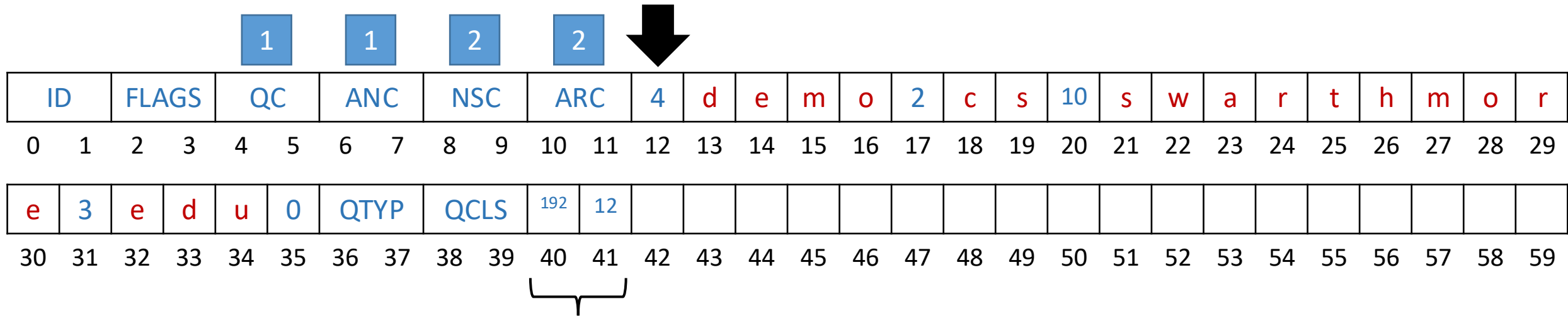
				1				1				2				2													
ID		FLAGS		QC		ANC		NSC		ARC		4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
e	3	e	d	u	0	QTYP		QCLS		192	12																		
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

Because the first two bits indicate "pointer", we ignore them: ~~00~~000000 00001100

Thus, the pointer is pointing to offset 12, which means "resume building the name you're looking for at offset 12 of the message".

Name so far:

Parsing Names (with pointers)

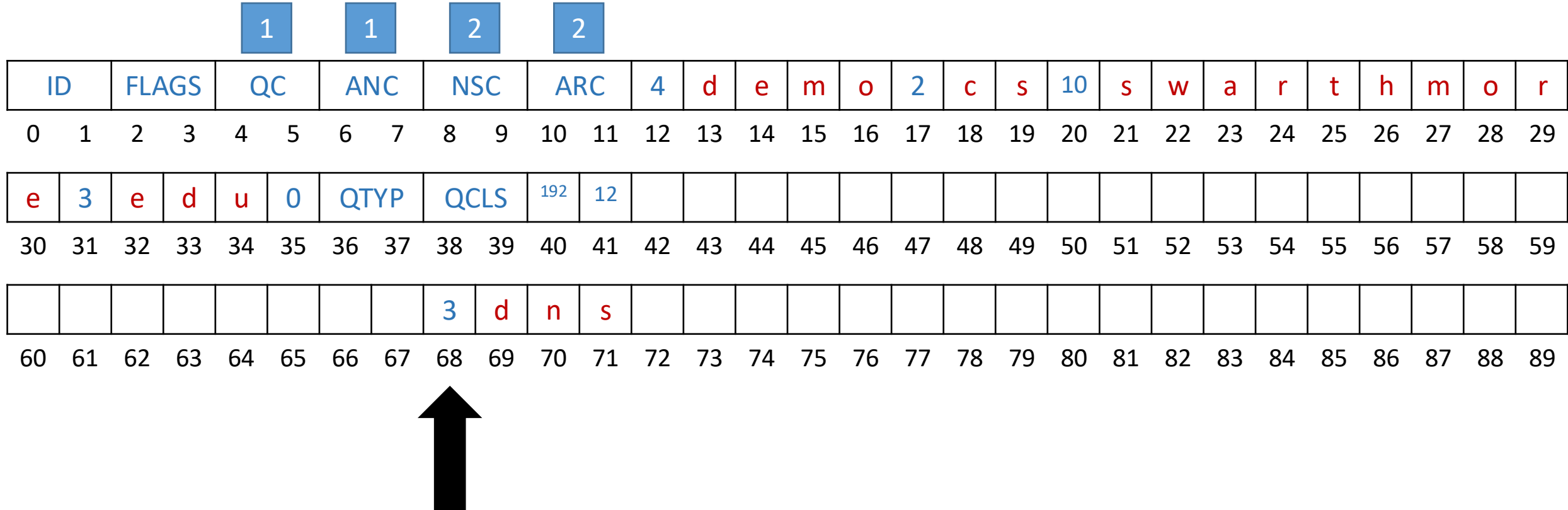


Because the first two bits indicate "pointer", we ignore them: ~~00~~000000 00001100

Thus, the pointer is pointing to offset 12, which means "resume building the name you're looking for at offset 12 of the message".

Name so far:

Parsing Names (with pointers)

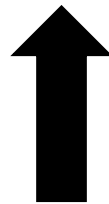


Name so far: dns

Parsing Names (with pointers)



				1		1		2		2																			
ID		FLAGS		QC		ANC		NSC		ARC		4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
e	3	e	d	u	0	Q TYP		QCLS		192	12																		
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
								3	d	n	s	192	17																
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89



Name so far: dns

Parsing Names (with pointers)

 Binary
 ASCII

				1			1			2			2																
ID	FLAGS			QC		ANC		NSC		ARC		4	d	e	m	o	2	c	s	10	s	w	a	r	t	h	m	o	r
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
e	3	e	d	u	0	QTP		QCLS		192	12																		
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
										3	d	n	s	192	17														
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89

Name so far: dns + (whatever we find at offset 17) => dns.cs.swarthmore.edu

Important things to consider:

- You may assume you'll have IP for root servers in `root-servers.txt` file
- You may assume that the additional records section will contain the A records for any server names listed in the NS records of the authority record section.
- Structuring your code with recursion (a function that calls itself) may be easier if you're comfortable with recursion. An iterative solution using loops can work too.
- Timeout after 5 seconds if you get no reply. See `timeout.c` and `timeout.py` in starter code
- Use helper functions and test small bits of code as you go.
- DNS is a binary protocol. Get comfortable using Wireshark.

