Enet

TCP/IP in Pure Erlang

Geoff Cant <nem@erlang.geek.nz>

Enet

~~TCP~~ UDP/IP in Pure(ish) Erlang

Geoff Cant <nem@erlang.geek.nz>

Who?

- Erlang hacker
- archaelus on irc/stackoverflow/erlang-questions/github
- Wellington -> Paris -> San Francisco

Also from NZ

My flows don’t glow like phosphorous.

What is an Enet?

- **http://github.com/archaelus/enet**
- A port program for using a TAP device
- A suite of packet encoders/decoders
- A collection of network interface functions
- A primitive IP stack

Why would you do that?

- Funsies
- To learn how the IP world works
- To build a library of IP modules for future work
- To get more control over the network stack
- To migrate a live TCP socket

Outline

- Port programs
- Binary syntax story time
- Tcpdump in Erlang
- Slowest loopback in the west
- Where to from here?

Port programs

Keeping my C code out of your VM.

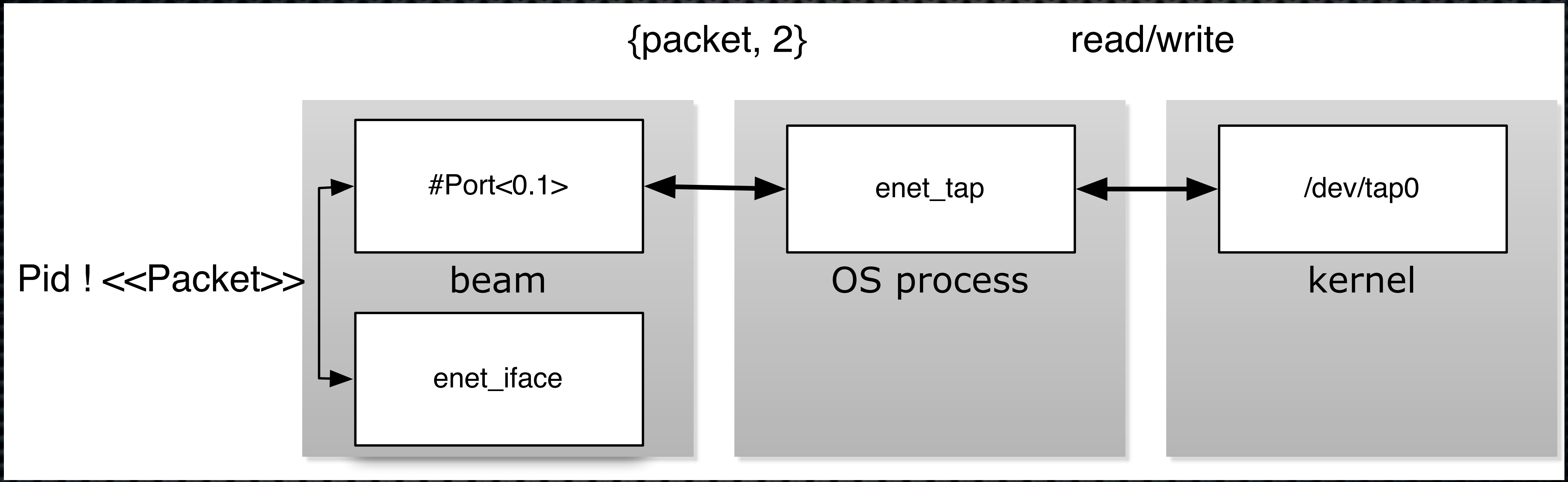
Playing well with others

- Port programs
- Port drivers
- NIFs
- Shell commands
- Network servers
- FFI (EEP-7)

Port programs

- Stable
    - Run as separate OS processes
    - My C code is not good. Keep it out of your VM.
- Simple structure
    - Communicate with erlang on stdin/out
    - Communicate with external resources via API

Port Communication



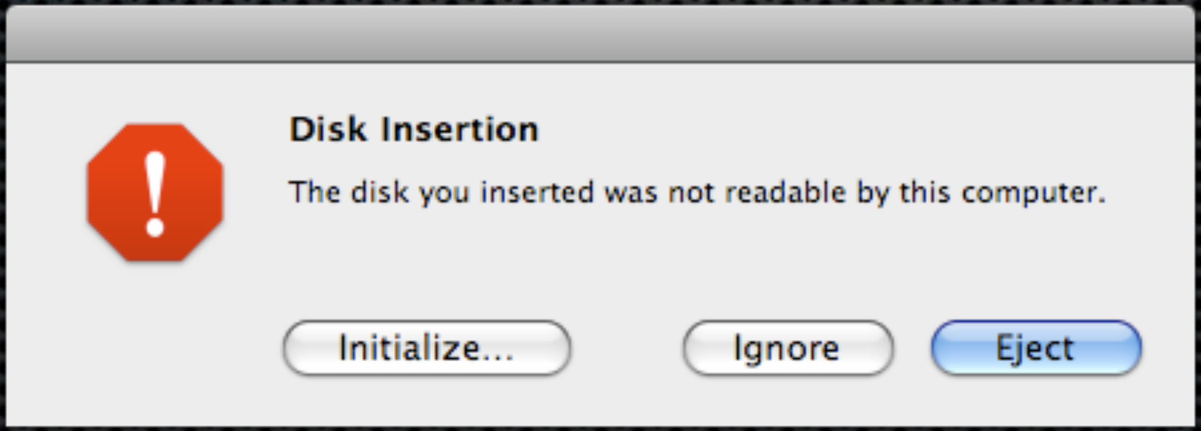
libevent

- Tie bufferevents to stdin to run code when erlang sends us data
- Tie an event to the TAP fd to run code when we receive packets from the network
- Queue output tasks in the input handlers

Harddisk recovery

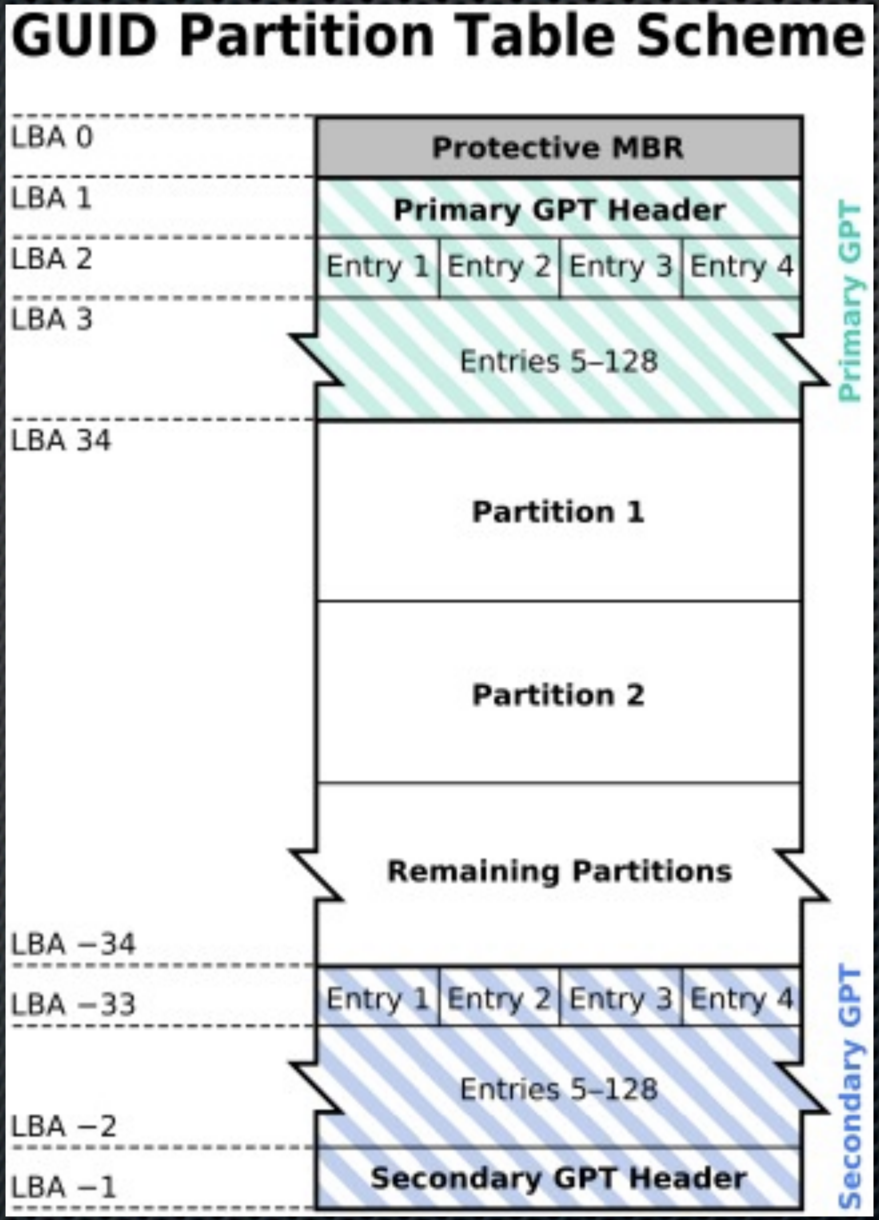
Binary syntax saves my ~~bacon~~ data

A sad tale...

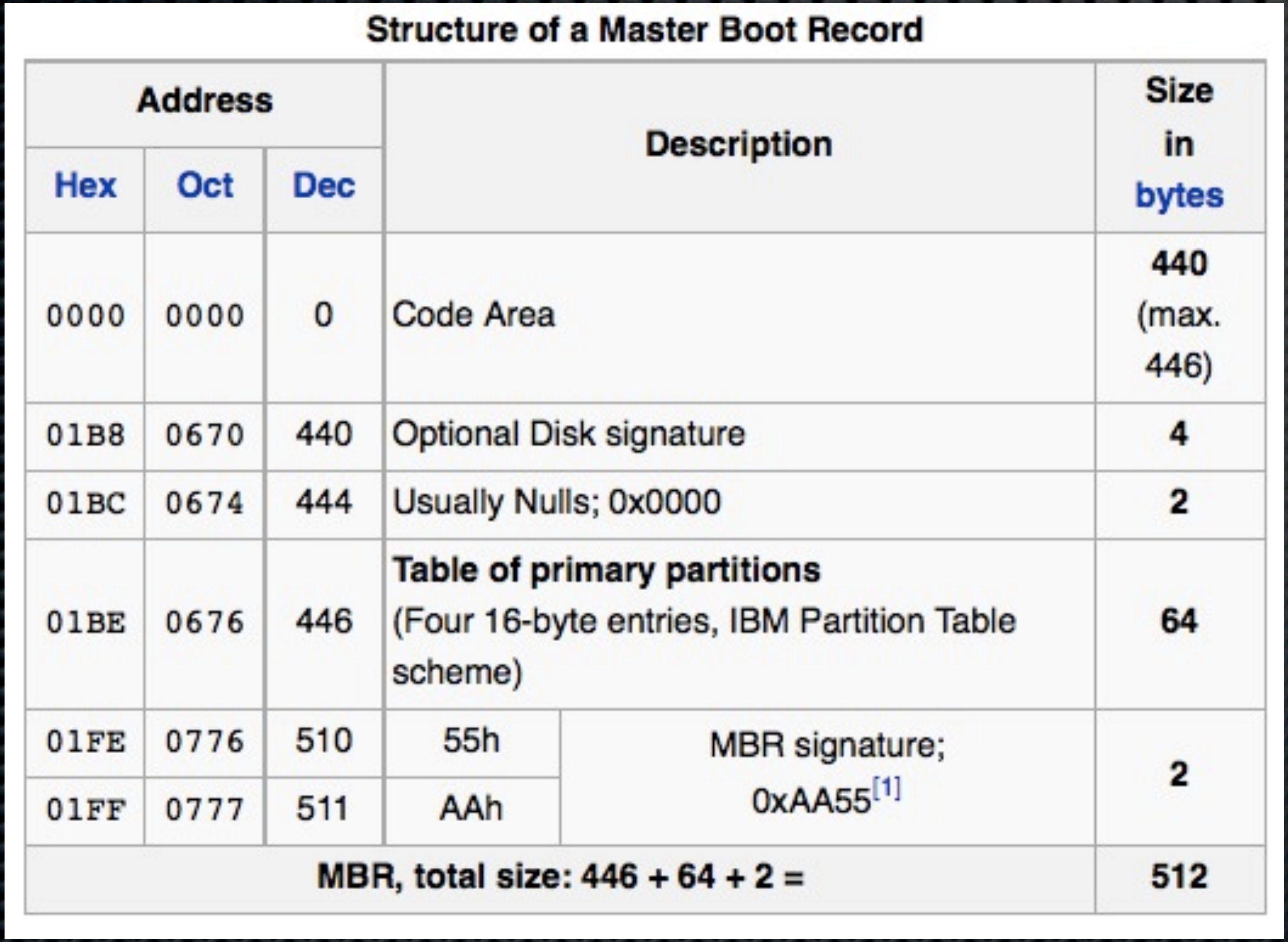

Disk Insertion

The disk you inserted was not readable by this computer.

A plan begins to form

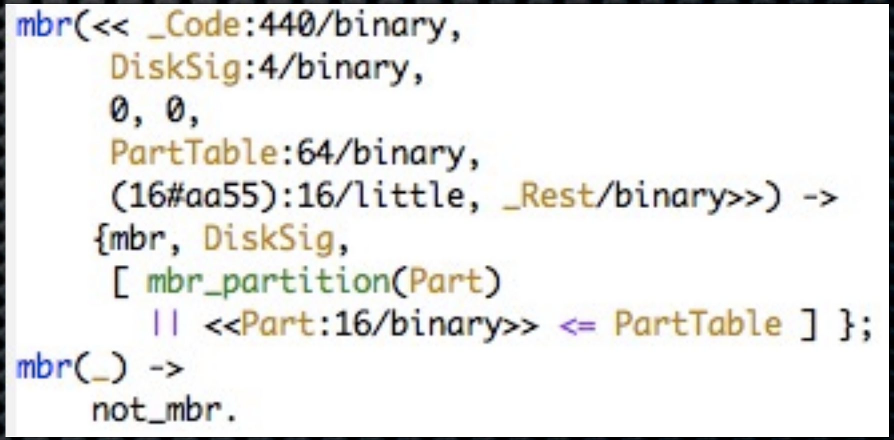


MBR



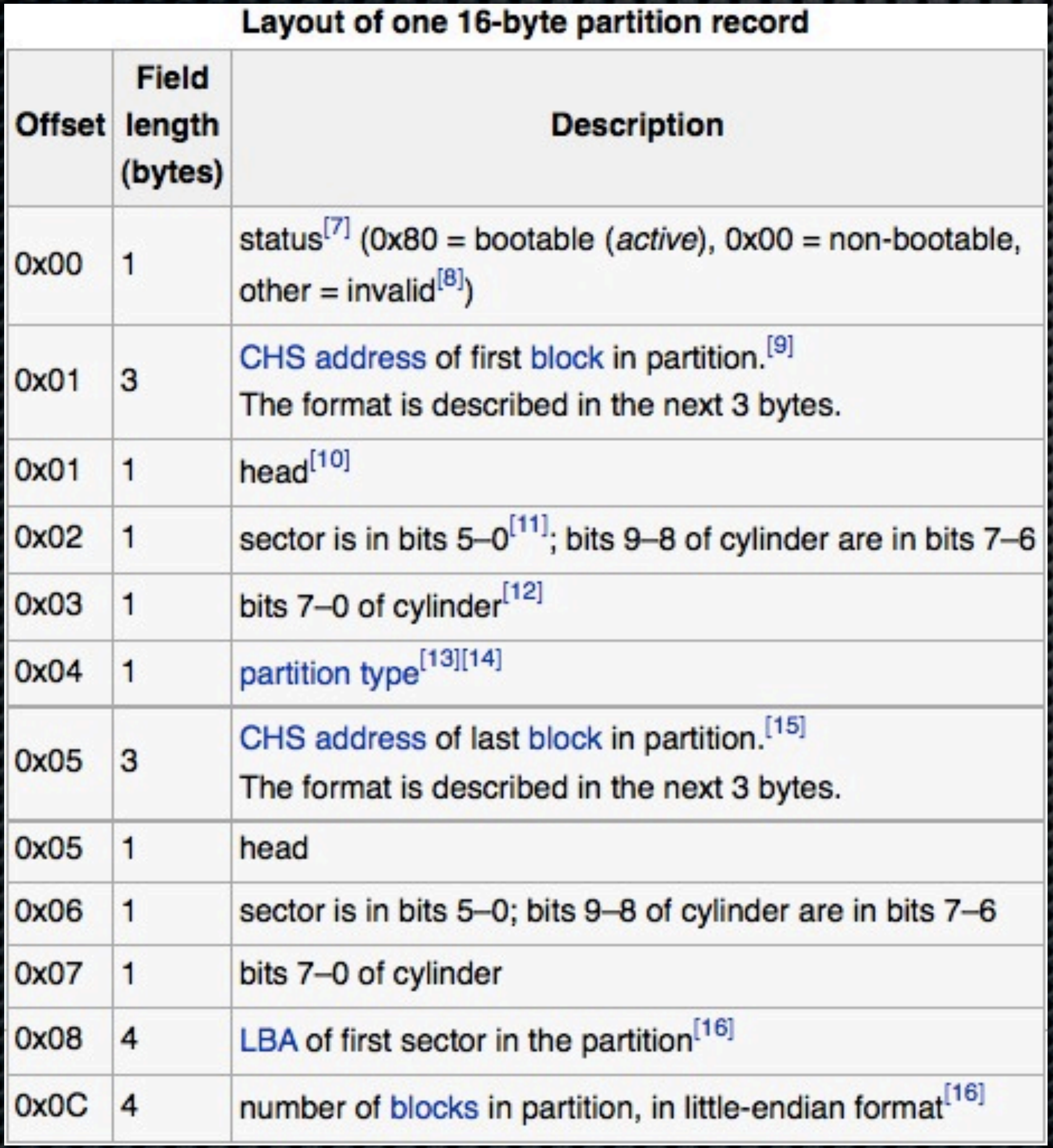
Structure of a Master Boot Record

Address			Description		Size in bytes
Hex	Oct	Dec			
0000	0000	0	Code Area		**440** (max. 446)
01B8	0670	440	Optional Disk signature		**4**
01BC	0674	444	Usually Nulls; 0x0000		**2**
01BE	0676	446	**Table of primary partitions** (Four 16-byte entries, IBM Partition Table scheme)		**64**
01FE	0776	510	55h	MBR signature; 0xAA55[1]	**2**
01FF	0777	511	AAh		
MBR, total size: 446 + 64 + 2 =					**512**



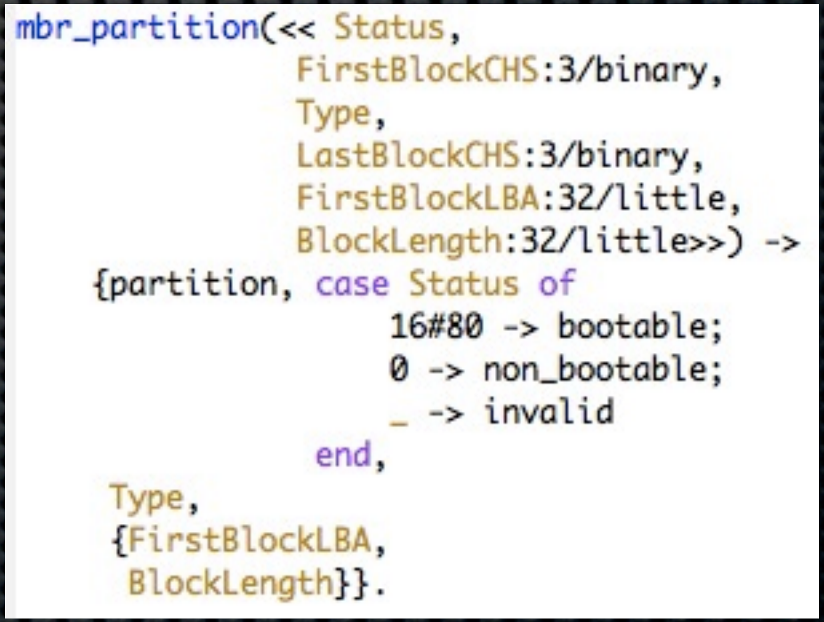
```
mbr(<< _Code:440/binary,
     DiskSig:4/binary,
     0, 0,
     PartTable:64/binary,
     (16#aa55):16/little, _Rest/binary>>) ->
    {mbr, DiskSig,
     [ mbr_partition(Part)
       || <<Part:16/binary>> <= PartTable ] };
mbr(_) ->
    not_mbr.
```

MBR Partitions



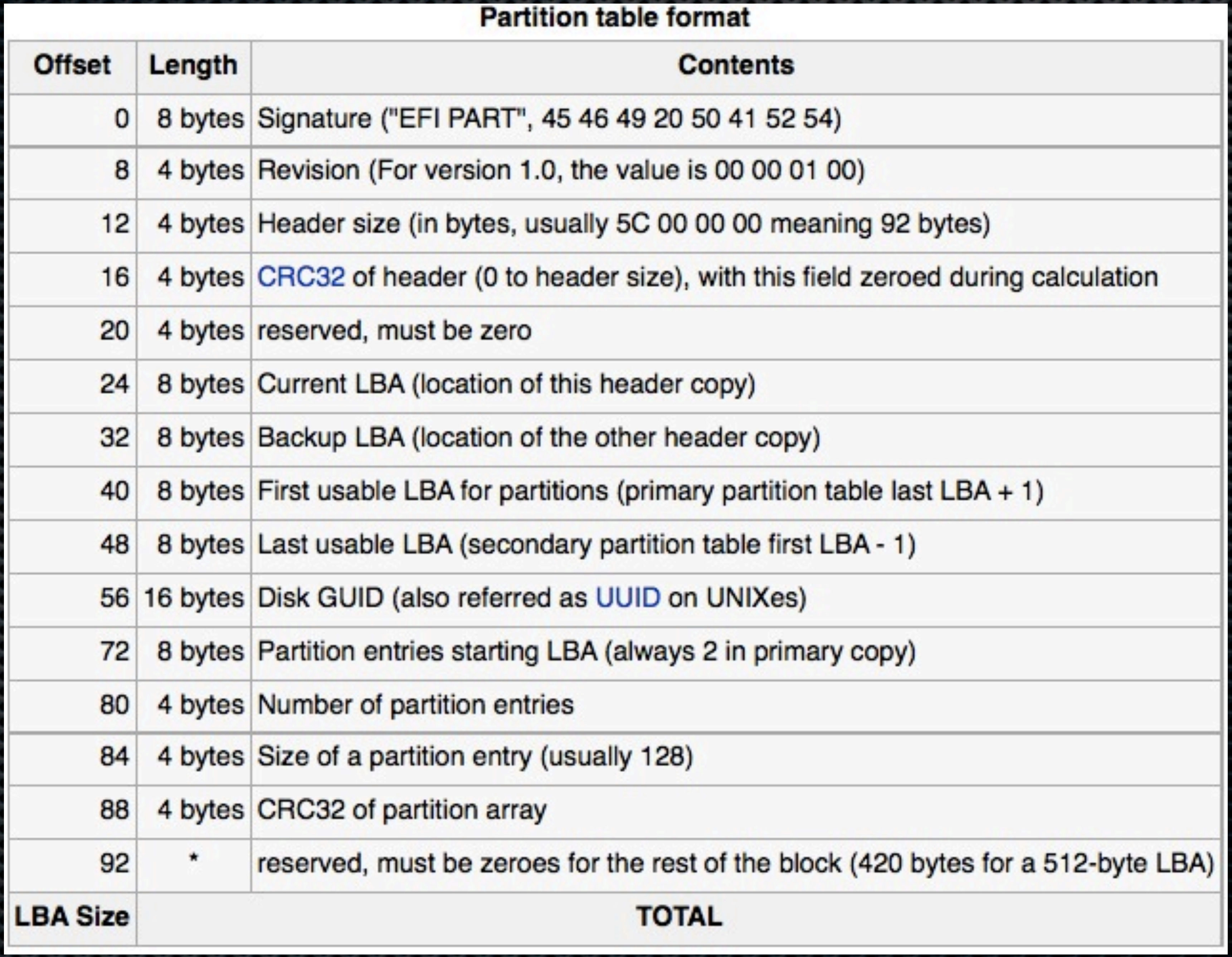
Layout of one 16-byte partition record

Offset	Field length (bytes)	Description
0x00	1	status[7] (0x80 = bootable (*active*), 0x00 = non-bootable, other = invalid[8])
0x01	3	CHS address of first block in partition.[9] The format is described in the next 3 bytes.
0x01	1	head[10]
0x02	1	sector is in bits 5–0[11]; bits 9–8 of cylinder are in bits 7–6
0x03	1	bits 7–0 of cylinder[12]
0x04	1	partition type[13][14]
0x05	3	CHS address of last block in partition.[15] The format is described in the next 3 bytes.
0x05	1	head
0x06	1	sector is in bits 5–0; bits 9–8 of cylinder are in bits 7–6
0x07	1	bits 7–0 of cylinder
0x08	4	LBA of first sector in the partition[16]
0x0C	4	number of blocks in partition, in little-endian format[16]



```
mbr_partition(<< Status,
                 FirstBlockCHS:3/binary,
                 Type,
                 LastBlockCHS:3/binary,
                 FirstBlockLBA:32/little,
                 BlockLength:32/little>>) ->
    {partition, case Status of
                    16#80 -> bootable;
                    0 -> non_bootable;
                    _ -> invalid
                end,
     Type,
     {FirstBlockLBA,
      BlockLength}}.
```

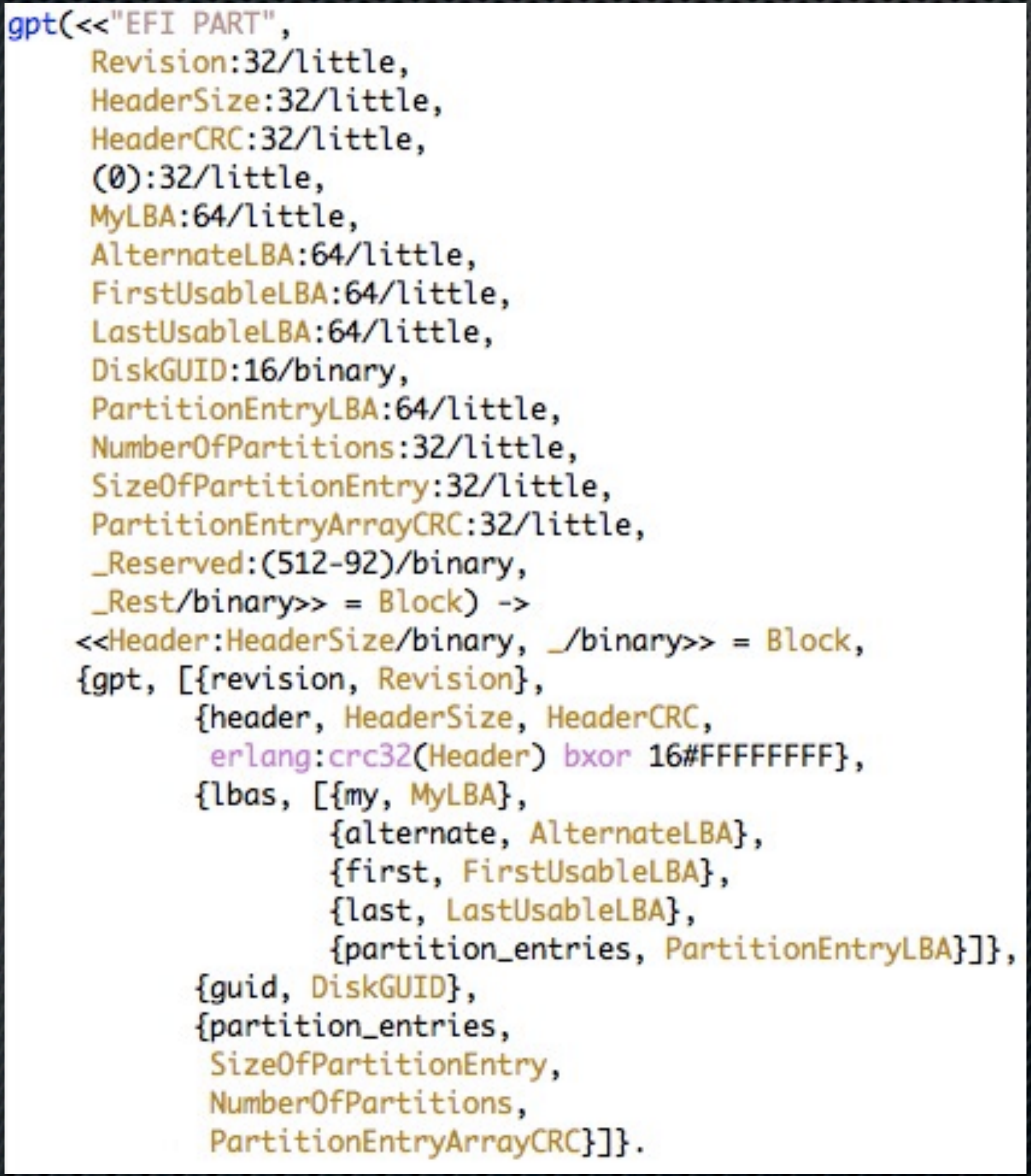
GPT Header



Partition table format

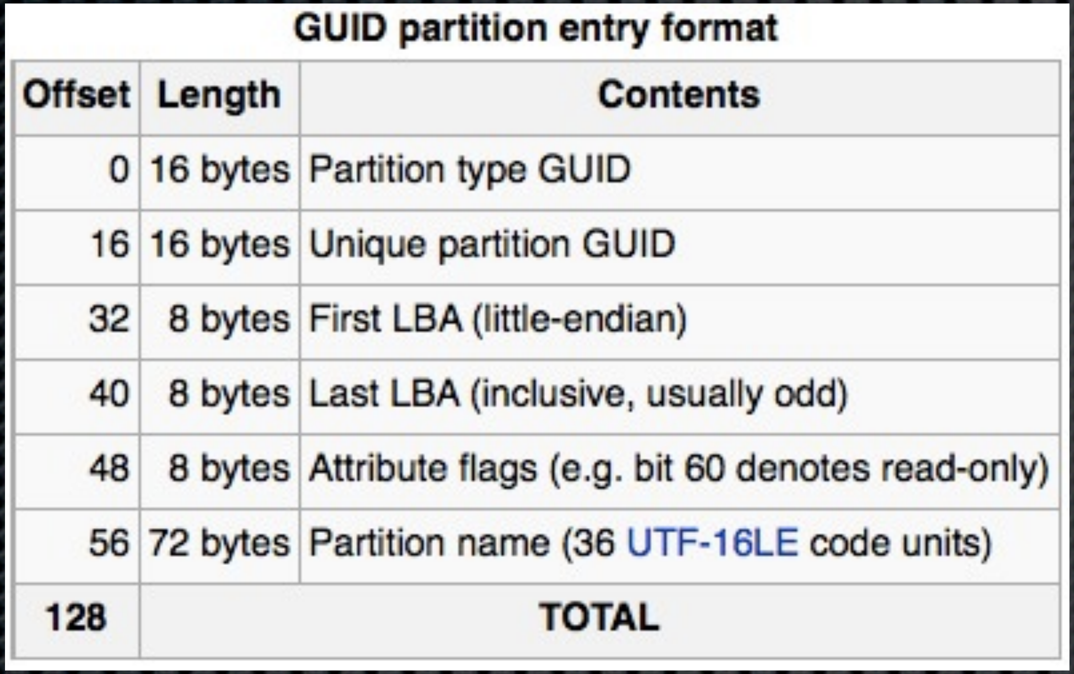
Offset	Length	Contents
0	8 bytes	Signature ("EFI PART", 45 46 49 20 50 41 52 54)
8	4 bytes	Revision (For version 1.0, the value is 00 00 01 00)
12	4 bytes	Header size (in bytes, usually 5C 00 00 00 meaning 92 bytes)
16	4 bytes	CRC32 of header (0 to header size), with this field zeroed during calculation
20	4 bytes	reserved, must be zero
24	8 bytes	Current LBA (location of this header copy)
32	8 bytes	Backup LBA (location of the other header copy)
40	8 bytes	First usable LBA for partitions (primary partition table last LBA + 1)
48	8 bytes	Last usable LBA (secondary partition table first LBA - 1)
56	16 bytes	Disk GUID (also referred as UUID on UNIXes)
72	8 bytes	Partition entries starting LBA (always 2 in primary copy)
80	4 bytes	Number of partition entries
84	4 bytes	Size of a partition entry (usually 128)
88	4 bytes	CRC32 of partition array
92	*	reserved, must be zeroes for the rest of the block (420 bytes for a 512-byte LBA)
LBA Size	**TOTAL**	

GPT Header



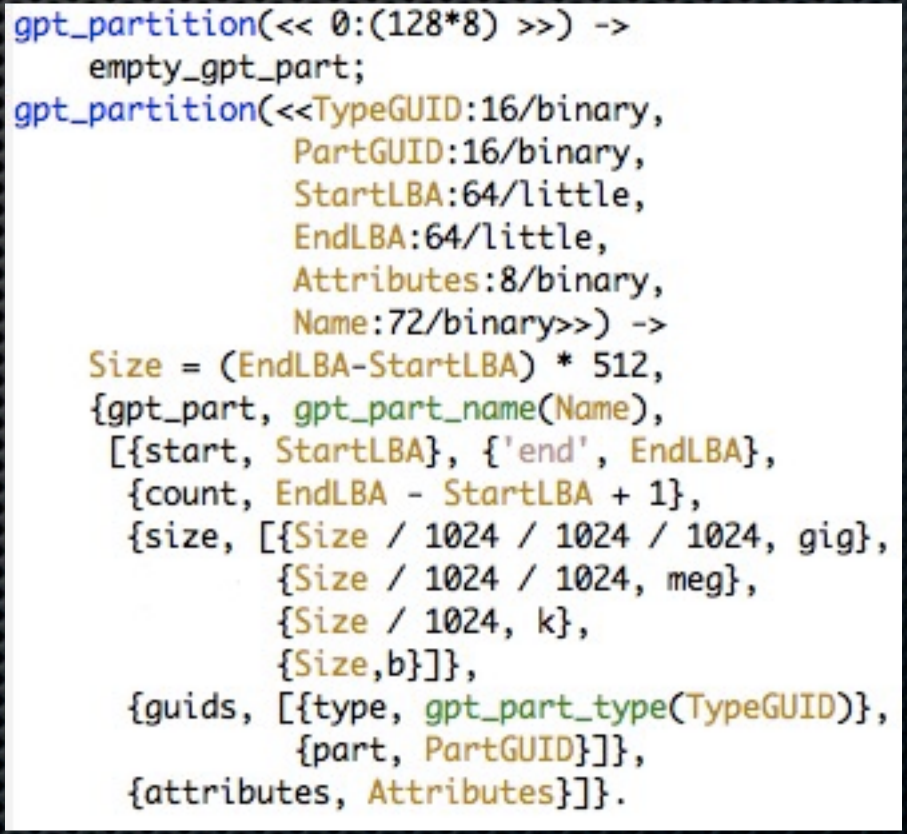
```
gpt(<<"EFI PART",
     Revision:32/little,
     HeaderSize:32/little,
     HeaderCRC:32/little,
     (0):32/little,
     MyLBA:64/little,
     AlternateLBA:64/little,
     FirstUsableLBA:64/little,
     LastUsableLBA:64/little,
     DiskGUID:16/binary,
     PartitionEntryLBA:64/little,
     NumberOfPartitions:32/little,
     SizeOfPartitionEntry:32/little,
     PartitionEntryArrayCRC:32/little,
     _Reserved:(512-92)/binary,
     _Rest/binary>> = Block) ->
    <<Header:HeaderSize/binary, _/binary>> = Block,
    {gpt, [{revision, Revision},
           {header, HeaderSize, HeaderCRC,
            erlang:crc32(Header) bxor 16#FFFFFFFF},
           {lbas, [{my, MyLBA},
                   {alternate, AlternateLBA},
                   {first, FirstUsableLBA},
                   {last, LastUsableLBA},
                   {partition_entries, PartitionEntryLBA}]},
           {guid, DiskGUID},
           {partition_entries,
            SizeOfPartitionEntry,
            NumberOfPartitions,
            PartitionEntryArrayCRC}]}.
```

GPT Partition



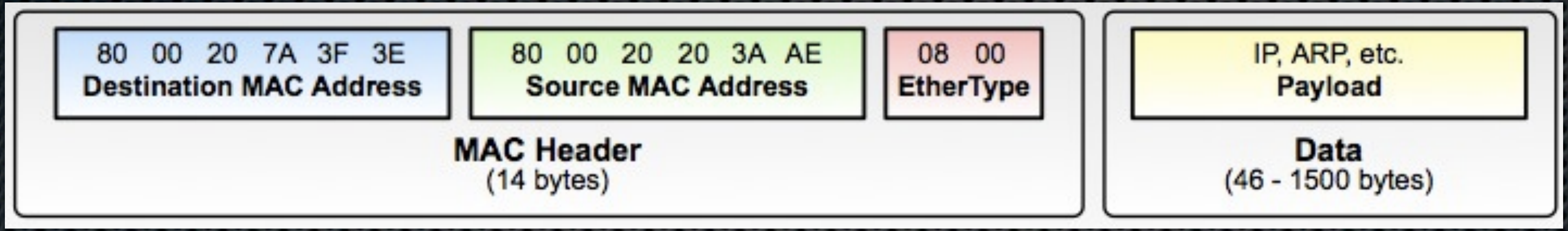
GUID partition entry format

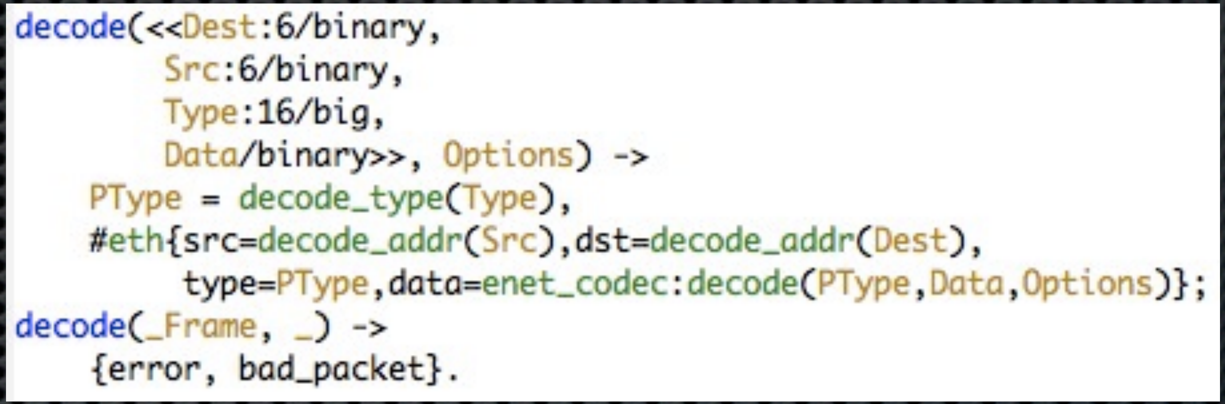
Offset	Length	Contents
0	16 bytes	Partition type GUID
16	16 bytes	Unique partition GUID
32	8 bytes	First LBA (little-endian)
40	8 bytes	Last LBA (inclusive, usually odd)
48	8 bytes	Attribute flags (e.g. bit 60 denotes read-only)
56	72 bytes	Partition name (36 UTF-16LE code units)
128	TOTAL	



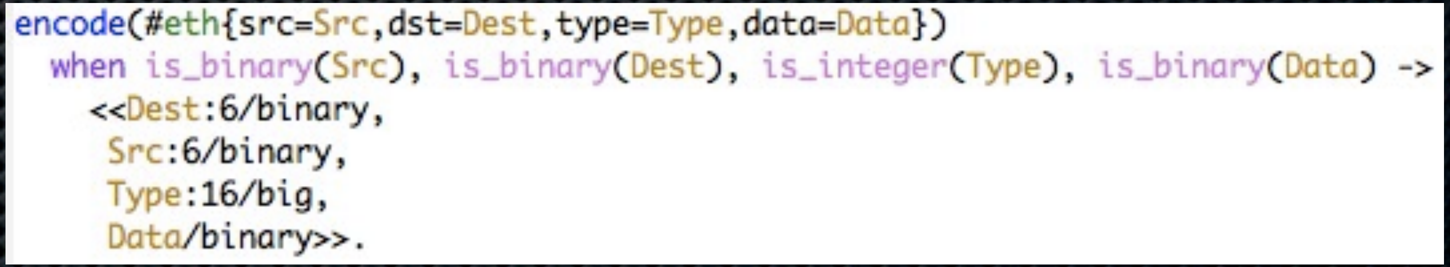
```
gpt_partition(<< 0:(128*8) >>) ->
    empty_gpt_part;
gpt_partition(<<TypeGUID:16/binary,
                PartGUID:16/binary,
                StartLBA:64/little,
                EndLBA:64/little,
                Attributes:8/binary,
                Name:72/binary>>) ->
    Size = (EndLBA-StartLBA) * 512,
    {gpt_part, gpt_part_name(Name),
     [{start, StartLBA}, {'end', EndLBA},
      {count, EndLBA - StartLBA + 1},
      {size, [{Size / 1024 / 1024 / 1024, gig},
              {Size / 1024 / 1024, meg},
              {Size / 1024, k},
              {Size,b}]},
      {guids, [{type, gpt_part_type(TypeGUID)},
               {part, PartGUID}]},
      {attributes, Attributes}]}.
```

Ethernet





```
decode(<<Dest:6/binary,
        Src:6/binary,
        Type:16/big,
        Data/binary>>, Options) ->
    PType = decode_type(Type),
    #eth{src=decode_addr(Src),dst=decode_addr(Dest),
         type=PType,data=enet_codec:decode(PType,Data,Options)};
decode(_Frame, _) ->
    {error, bad_packet}.
```

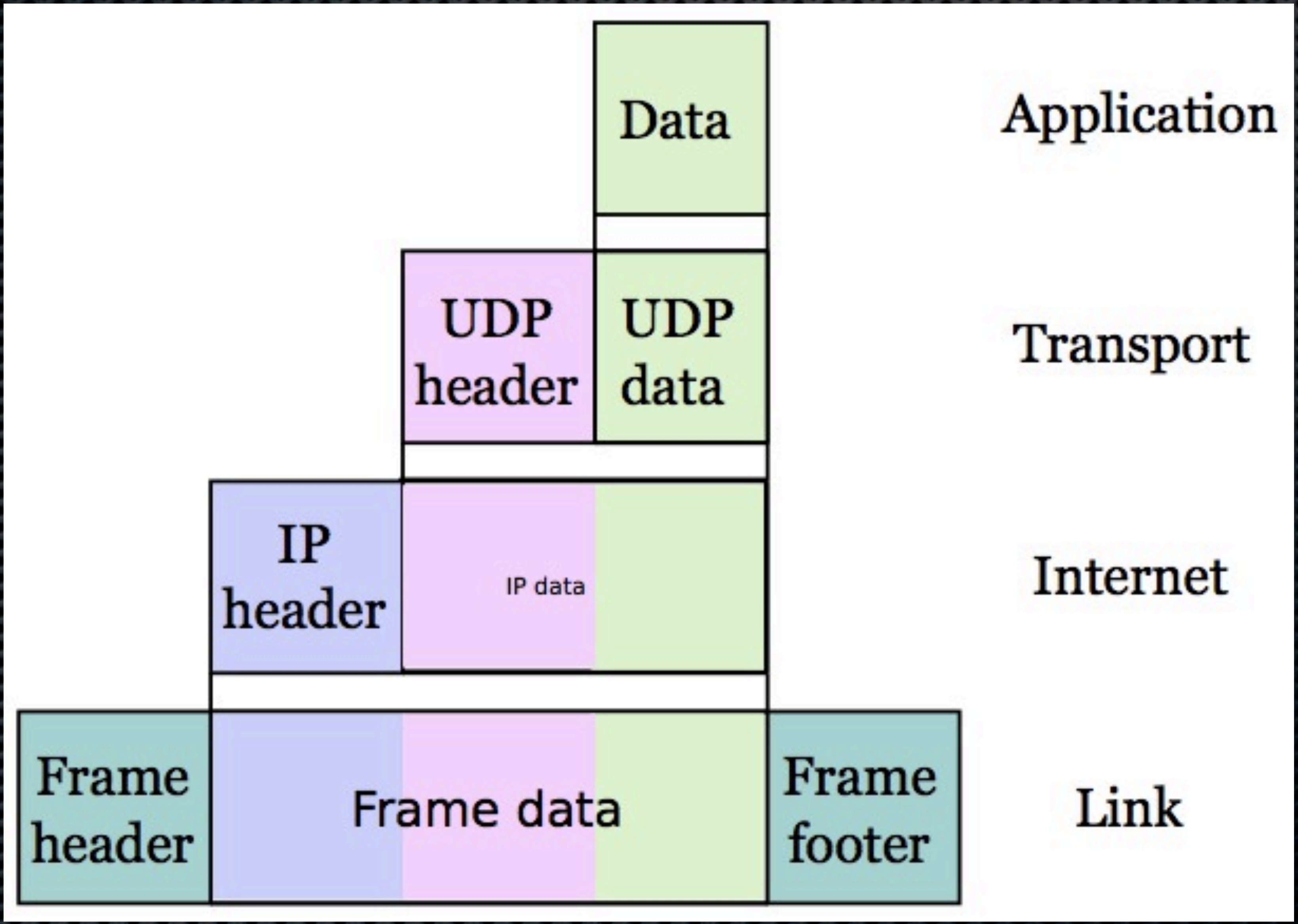


```
encode(#eth{src=Src,dst=Dest,type=Type,data=Data})
  when is_binary(Src), is_binary(Dest), is_integer(Type), is_binary(Data) ->
    <<Dest:6/binary,
     Src:6/binary,
     Type:16/big,
     Data/binary>>.
```

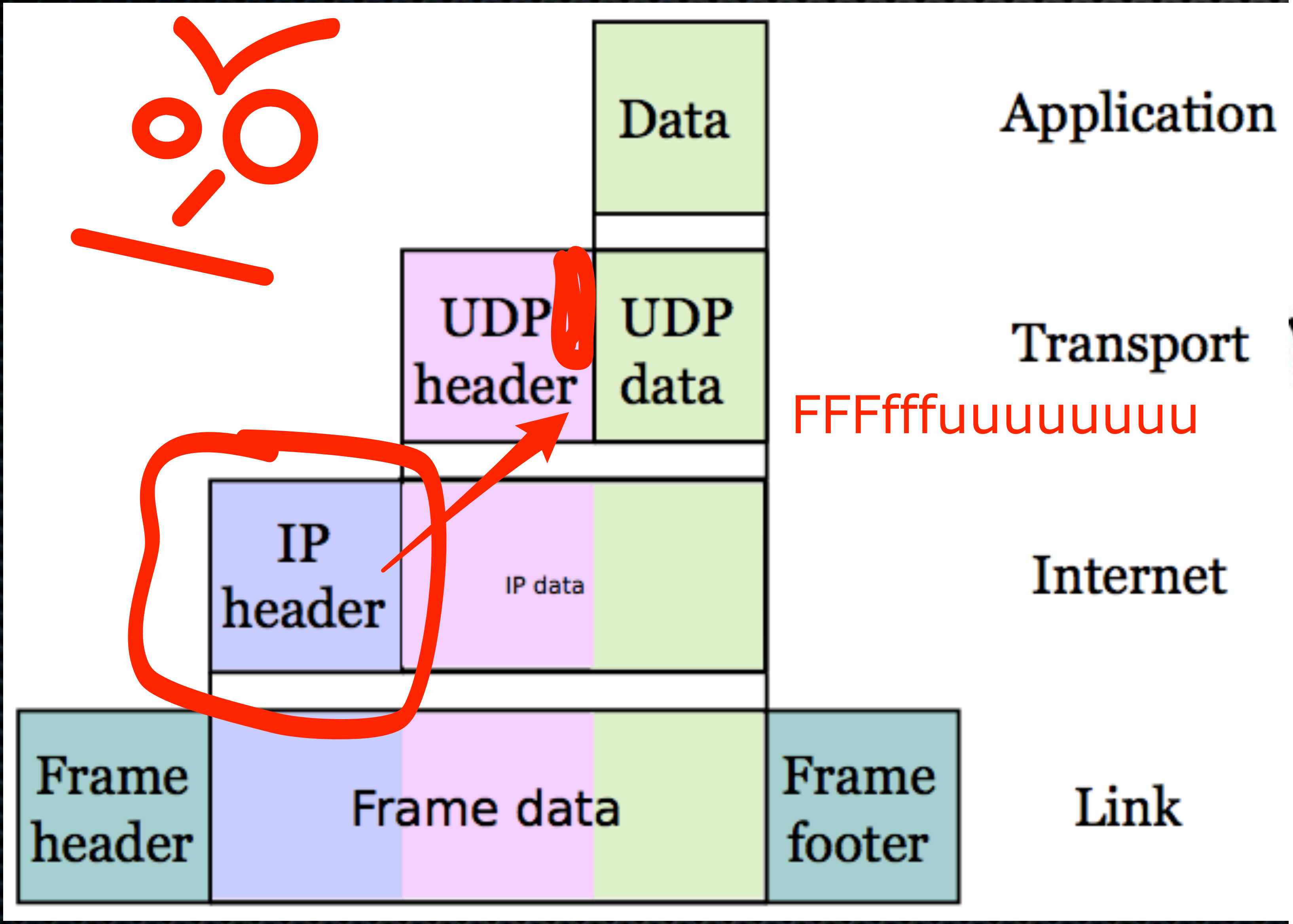
And so on...

- ethernet
- ipv4
- ipv6
- icmp4
- udp4
- tcp4
- icmp6
- udp6
- tcp6
- arp4
- dns

Network Stacks...

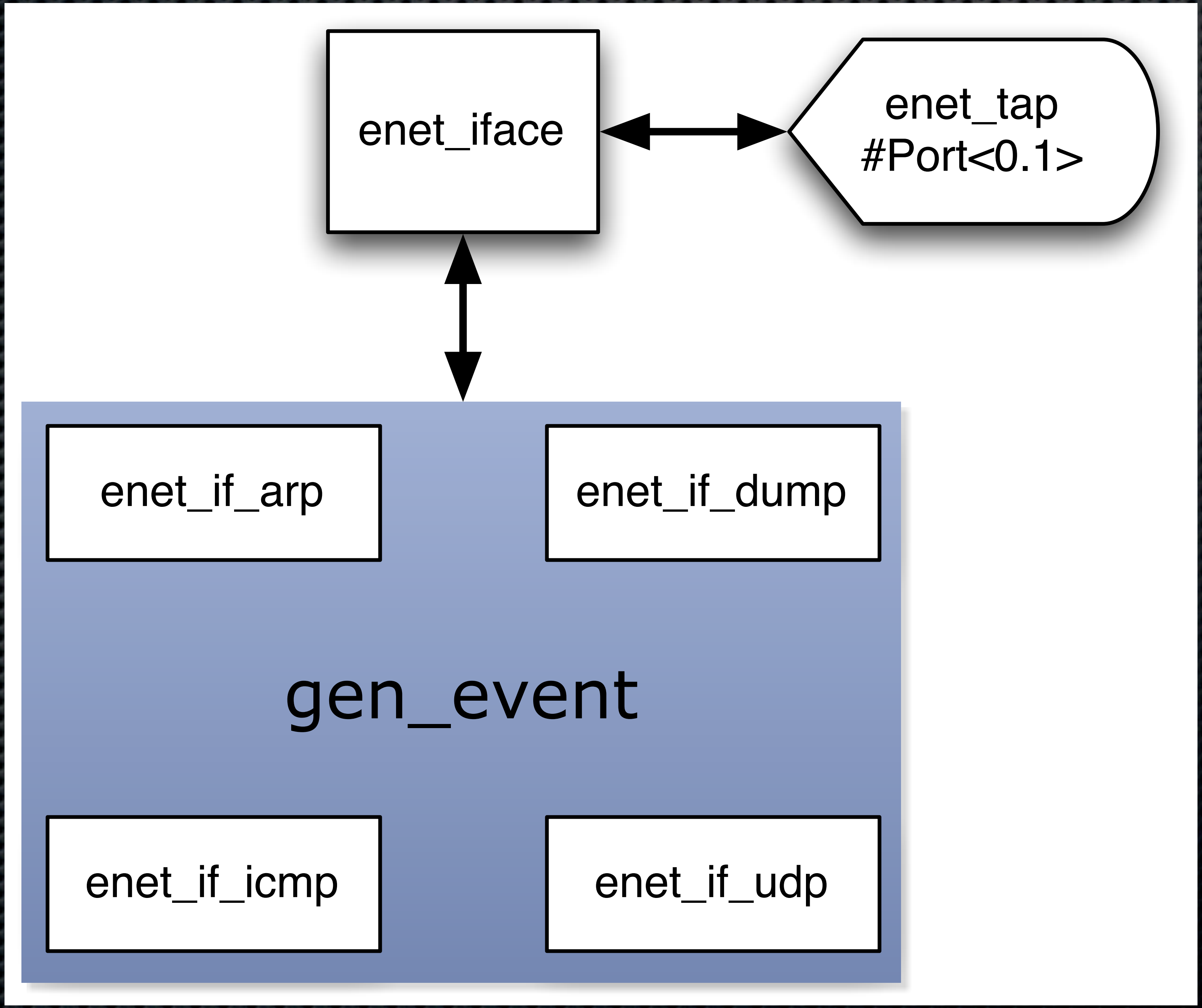


All filthy lies



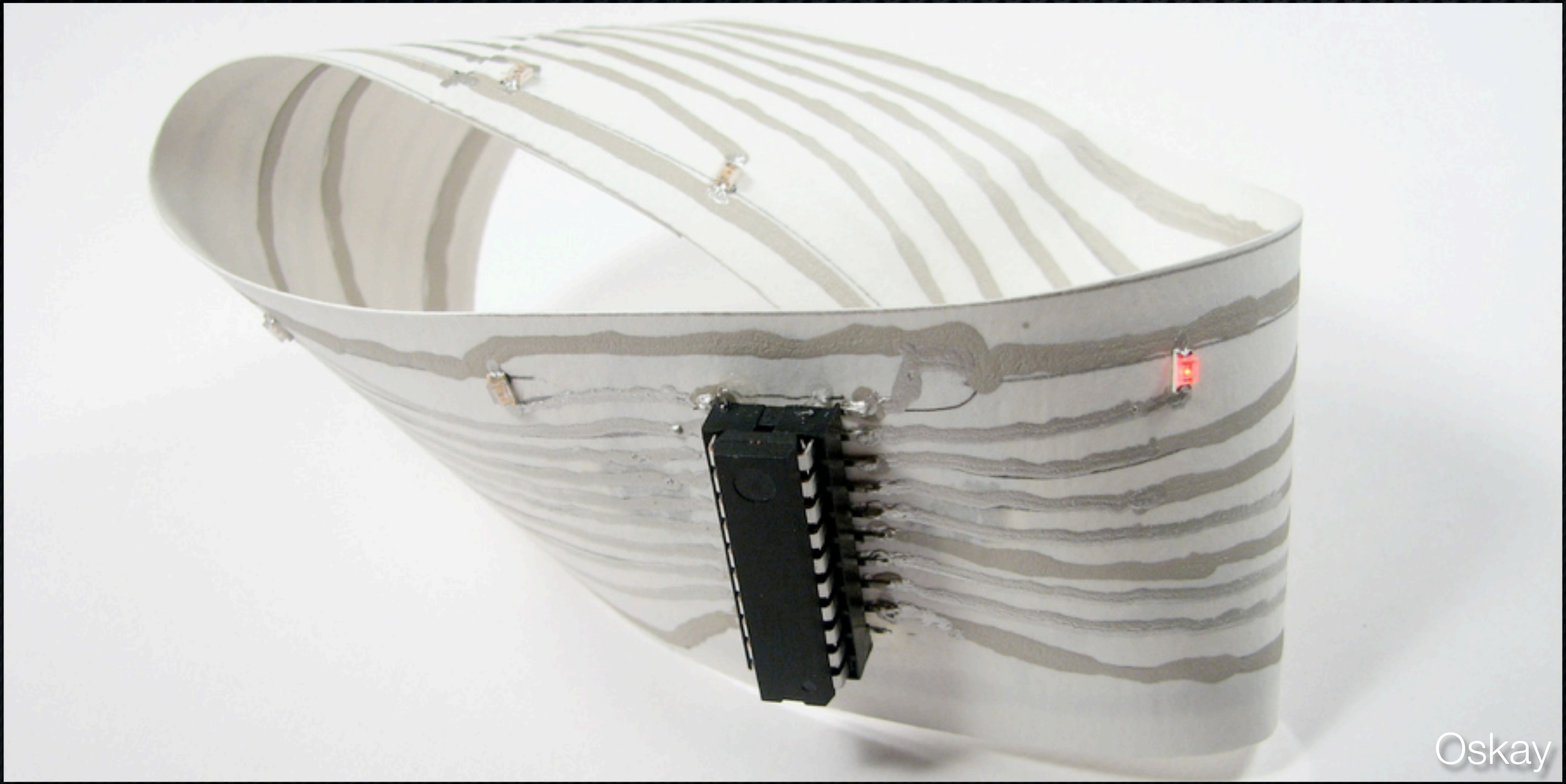
Tcpdump in Erlang

Enet Interface



Demo

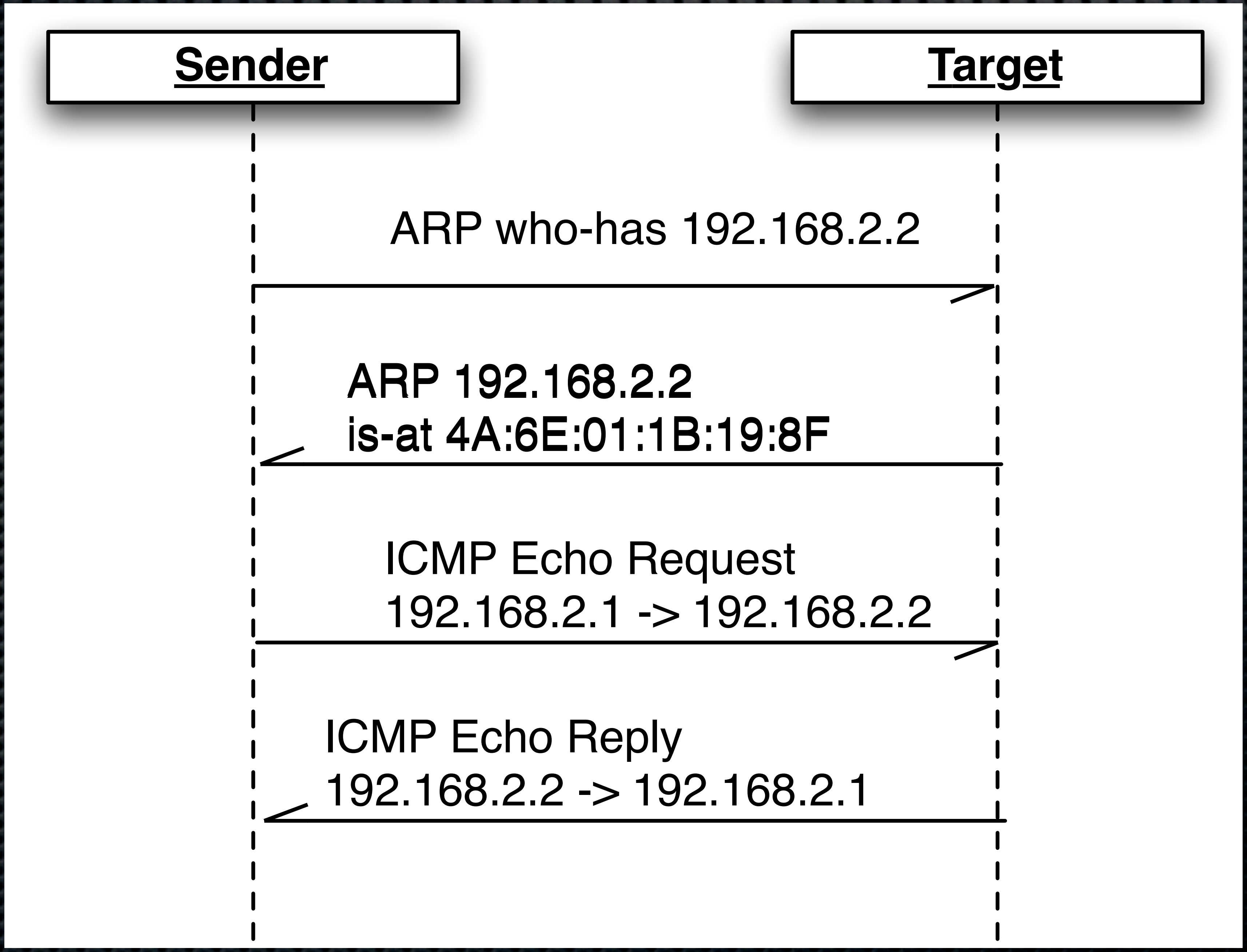
TCPdump



Erloopback

Dialup speed *without* an acoustic coupler

Ping over ethernet



Demo

Ping

Problems, TODO

- Erlangy pubsub
- TCP fsm
- Rewrite and flesh out the interface code
- SCTP
- Socket migration