SW sits on top of the HW.

prevailing SW to implement communication is TCP/IP

requires source destination address.

IP address 32 bits (IPv4)

human readable form

a.b.c.d

128.105.140.15

dotted decimal notation

try
hostname -i
try \% hostname

We like words we can read, so host names form a set of domain names.

Tree structure with unnamed root:

```
    edu
   |___ com
   |     |___ mil
   |     |     |___ org
   |     |     /...
   |______
  /     
 cs
 /       
 turunmaa
```

Karen's desktop is

turunmaa.cs.wisc.edu
So, we need a mapping from host + domain names to IP addresses. 

DNS

a distributed database

try
godig

useful functions:

```c
struct hostent * gethostbyname
  (const char * name);
```

```c
struct hostent * gethostbynameaddr
  (const void * addr,
   int len,
   int type);
```

note: old functions
New: getaddrinfo() getnameinfo()
hostent structure

struct hostent {
    char *h_name;      /* official name of host */
    char **h_aliases;  /* alias list */
    int h_addrtype;    /* host address type */
    int h_length;      /* length of address */
    char **h_addr_list; /* list of addresses */
};

null-terminated array of
pointers to in-addr
structs

in-addr struct is an unsigned
int representing an
IP address in network
byte order
Back to the original goal: a way for a program running on computer A to exchange messages with a program running on computer B.

connection
point-to-point
g full duplex
g simplex
g half duplex

IP:port

A

B
Sockets provide connections.

common standard:

Berkeley sockets (BSD)
POSIX sockets

a socket is an endpoint of the connection.

It has an address:

IP address & port number

1. ephemeral
   used by client
   assigned by kernel

2. well known
   used by services, so clients know how to contact server.

53 DNS
80 HTTP
143 IMAP(email)
using sockets...  

*client

create a socket

int socket(int domain,  
SOCK_STREAM (TCP)  
SOCK_DGRAM (UDP)

int type,  
AF_INET (for IPv4)  
AF_INET6 (for IPv6)  
AF_UNIX (local)

int protocol);  
which transport protocol to use

0 - use defaults for chosen domain + type

returns file descriptor or -1 if error
connect with a server

Kernel assigns local port (TCP) connects to server

```c
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addr_len);
```

returns

0  success
-1  error occurred (no connection)
socket structures

#include <socket.h>
#include <netinet/in.h>

/* generic socket structure */
struct sockaddr {
    unsigned short   sa_family;    /* protocol family */
    char             sa_data[14];
};

/* socket structure for internet */
struct sockaddr_in {
    unsigned short   sin_family;    /* address family AF_INET */
    unsigned short   sin_port;      /* port number in network byte order */
    struct in_addr   sin_addr;      /* IP address in network byte order */
    unsigned char    sin_zero[8];   /* extra bytes to match size of
                          struct sockaddr */
}

struct in_addr {
    unsigned int   s_addr;    /* network byte order */
}
Different architectures use big/little endian byte ordering. So constructing packets and frames is problematic.

We define network byte order, and put all things (outgoing) into that order. System dependent library functions that do the work:

```c
#include <arpa/inet.h>

unsigned long int htonl (uint32_t hostlong);
uint16_t htons (hostshort);
uint32_t ntohl (netlong);
uint16_t ntohs (netshort);
```
interact with server

use system calls such as

```c
#include <unistd.h>

ssize_t read(int fd, void* buf,
              size_t count);
```

read up to count bytes from fd into buffer starting at buf

```c
ssize_t write(int fd,
              const void* buf,
              size_t count);
```

write up to count bytes from buf to file specified by fd
when done, terminate the connection

#include <unistd.h>

int close(int fd);

0 success
-1 error
* client summary *

socket
connect
read/write
close
server

create a socket

just like client

assign an address to socket

int bind (int sockfd,
    const struct sockaddr *addr,
    socklen_t addrlen);

associate address in addr with the sockfd
prepare for connections

needed for socket connections.

```c
#include <sys/types.h>
#include <sys/socket.h>

int listen (int sockfd,
            int backlog);

0 success
-1 error

how many connection requests to queue.

tell the kernel to prepare to queue connections for this socket, as server will be using accept().
```
Since all 3: 
socket()
bind()
listen()

need to happen for sockets in the server's code, they are often grouped and placed into a function, like textbook example, Figure 11.17 page 906.
Common paradigm:

Server waits for a connection (a request), handles the request, closes the connection.

\[
\text{for Tcp}
\]

\[
\text{int accept (int sockfd, struct sockaddr *cliaddr,}
\]

\[
\text{socklen_t *addrlen);
\]

the fd of the accepted connection or -1 error

client's address info.