Lecture 05
Building a Beowulf Cluster

Dr. Ehtesham Zahoor

Assignment-02

The consensus problem in distributed systems is the problem of getting a set of nodes to agree on something – it might be a value, a course of action or a decision. This problem has been at the core of distributed systems research for over last few decades. A Simple solution, which can work under some constraints, is called two-phase commit, or 2PC.

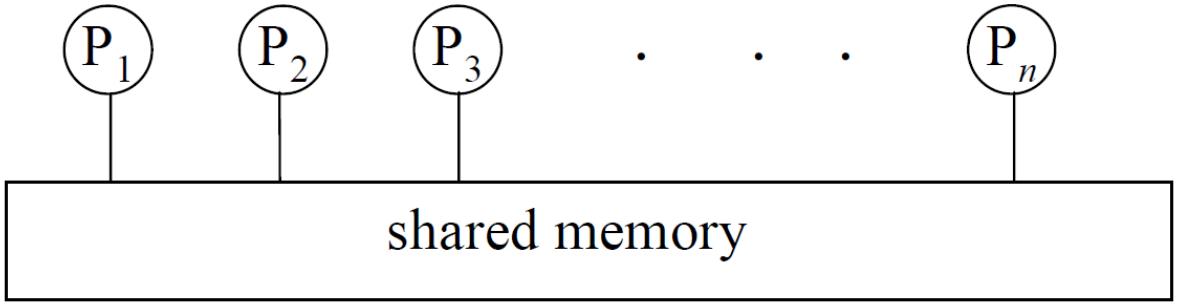
Think about how you would solve a real-world consensus problem – perhaps trying to arrange a graduate party! You'd call all your friends and suggest a date and a time. If that date/time is good for everybody you have to ring again and confirm, but if someone can't make it you need to call everybody again and tell all your friends that the party is off. More specifically, we can identify two steps in the process:
First, the first proposal phase involves proposing a value to every participant in the system and gathering responses. In the next phase, if everyone agrees, there is need to contact every participant again to let him or her know. Otherwise, contact every participant to abort the consensus.

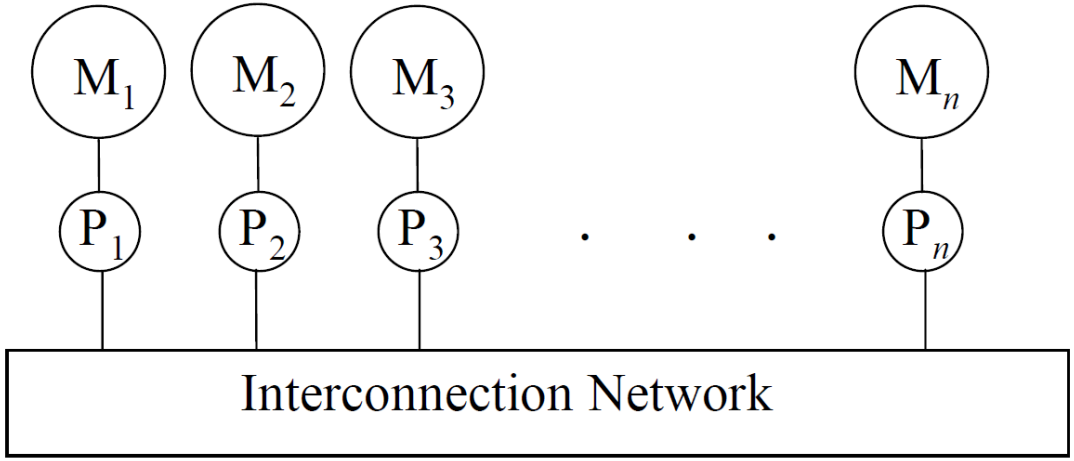
For this question, you need to ignore the part to contact every participant again, thread share variables and this can be implicit.

Your code should be efficient (one criteria is LOC) and concise and the assignment needs to be done using openMP and not pthreads.

Parallel and Distributed Computing

- The difference includes whether the processes communicate using shared or a distributed memory.



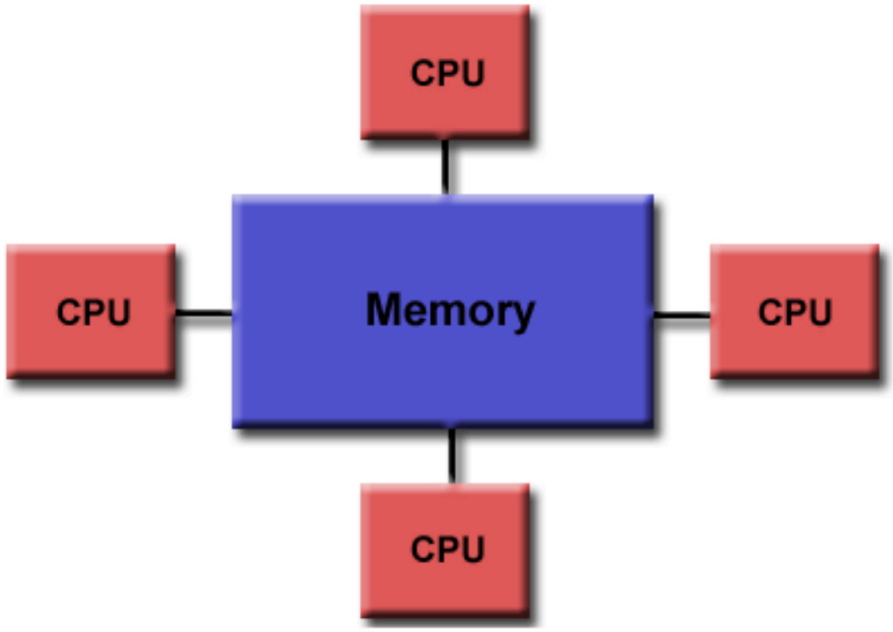


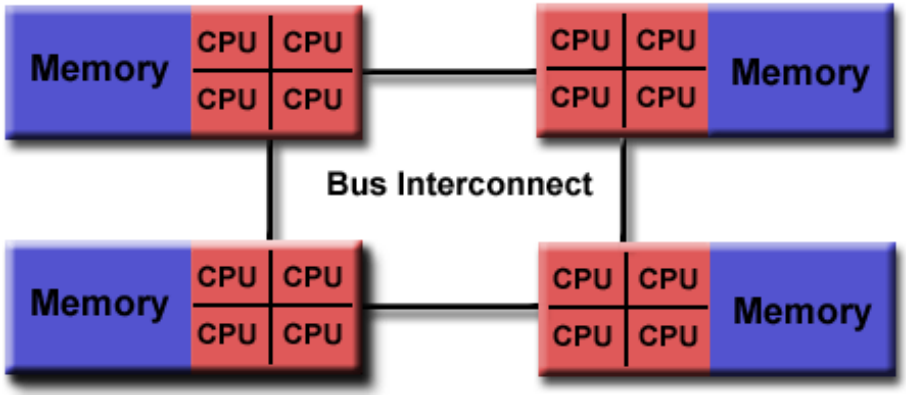
OpenMP

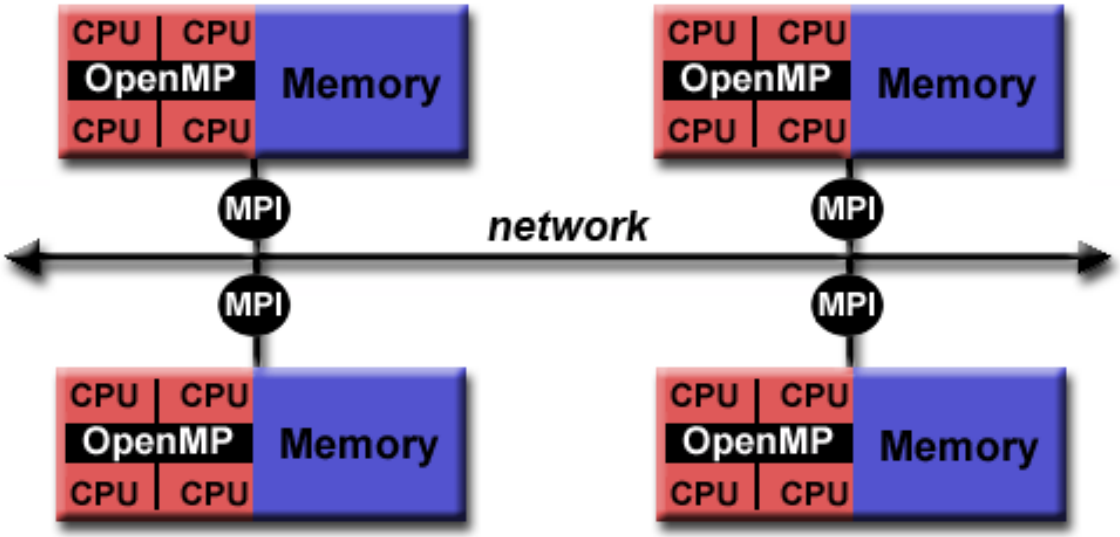
- Programming of shared memory systems
- An API for Fortran and C/C++
  - Directives
  - Runtime routines
  - Environment variables

This Lecture is based on https://computing.llnl.gov/tutorials/openMP/

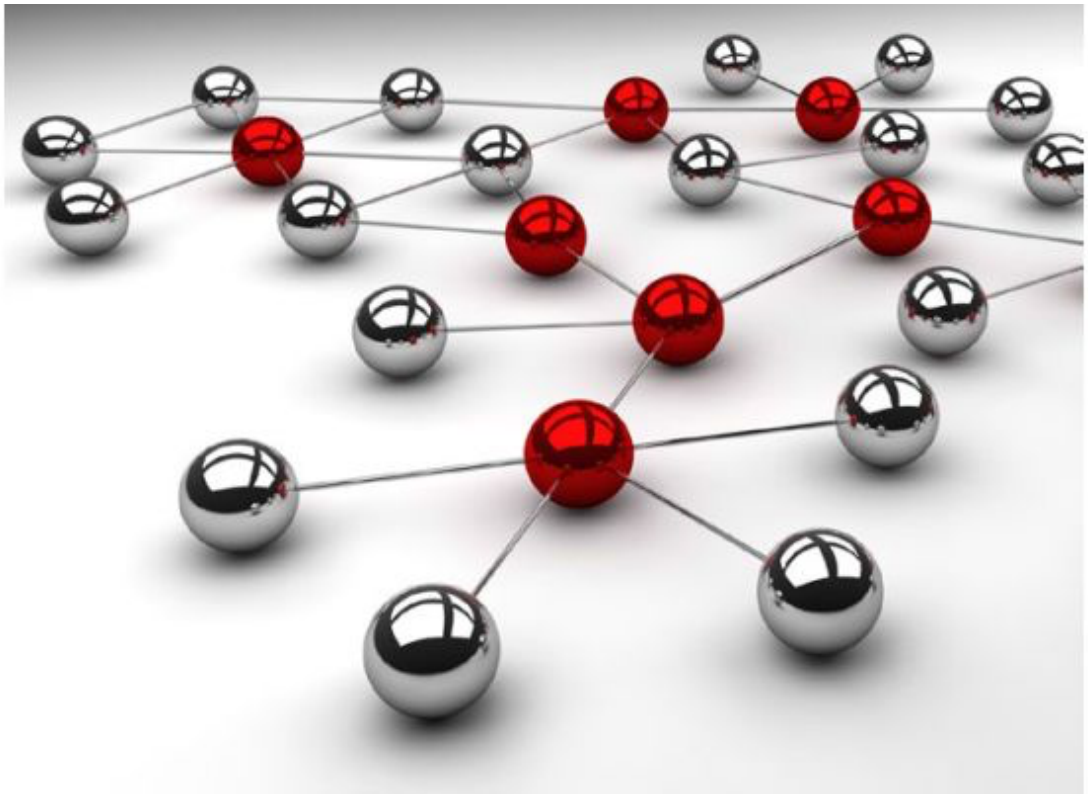
Lecture 05: Building a Beowulf Cluster







- Lets build a *virtual* cluster



Beowulf Cluster

- A Beowulf cluster is a computer cluster of what are normally identical, commodity-grade computers networked into a small local area network with libraries and programs installed which allow processing to be shared among them.

- The result is a high-performance parallel computing cluster from inexpensive personal computer hardware.

Beowulf Cluster

- The name *Beowulf* originally referred to a specific computer built in 1994 at NASA.

- No particular piece of software defines a cluster as a Beowulf.

But we dont have machines …

- We can use virtual machines!!

VMs or Docker containers?

- For simplicity, we will use VMs
- For the *actual* Cluster, it doesnt matter if we are using the *actual* or physical machines, the steps remain exactly the same.

Beowulf Cluster - Step 0: Setup

- Install Oracle VirtualBox
  - Open source, can be downloaded from https://www.virtualbox.org/wiki/Downloads
- Download Ubuntu server / desktop
  - We are using ubuntu-14.04.4-desktop-amd64
  - Other versions should also be fine
- Create a new VM on VirtualBox
  - Linux – Ubuntu 64-bit
  - We name it master, use default settings

Beowulf Cluster - Step 0

- Start the new VM, master, using VirtualBox
- Once asked, select the Ubuntu installation file

- Go through the steps to install the Ubuntu, not much complicated, mostly choose default options
  - Choose easy to remember username/password
  - Choose correct region

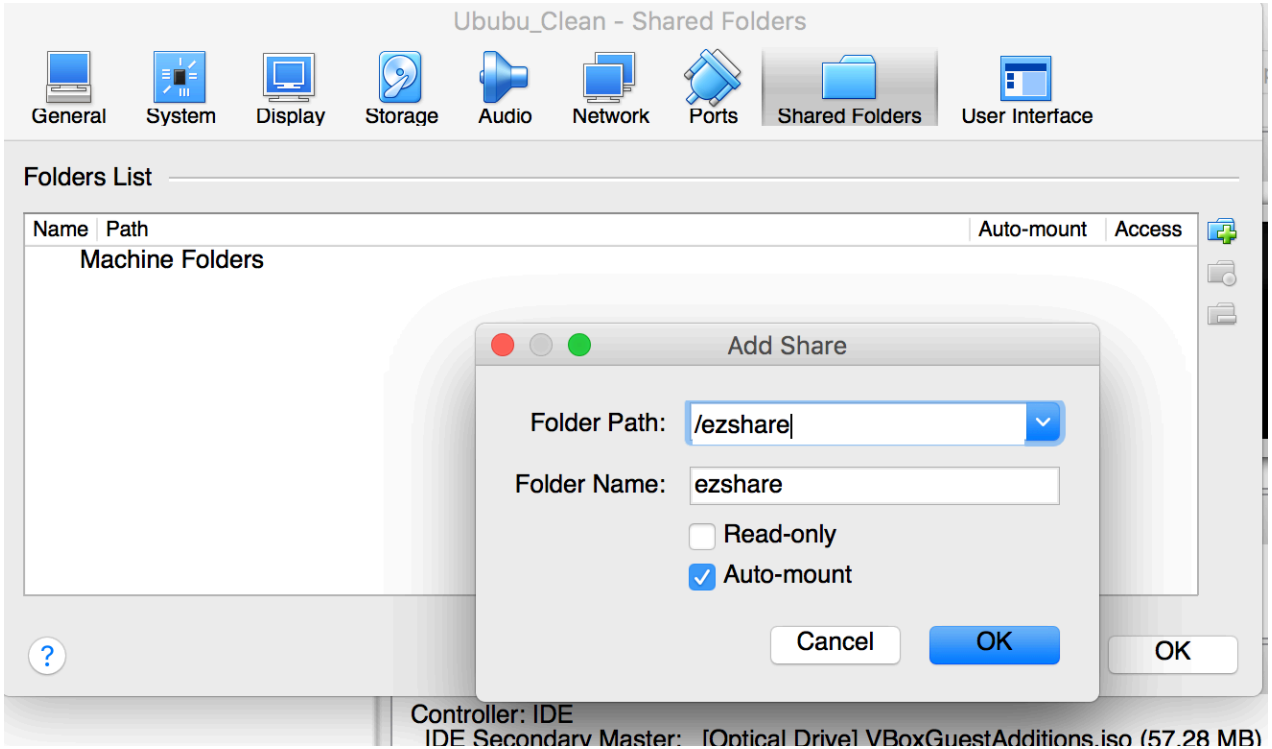
Beowulf Cluster - Step 0

- Once the Installation is complete you would be able to log into your system

- *If the screen size doesnt scale on maximize, click devices --- Insert Guest Additions CD Image and install the executable in mounted CD …*

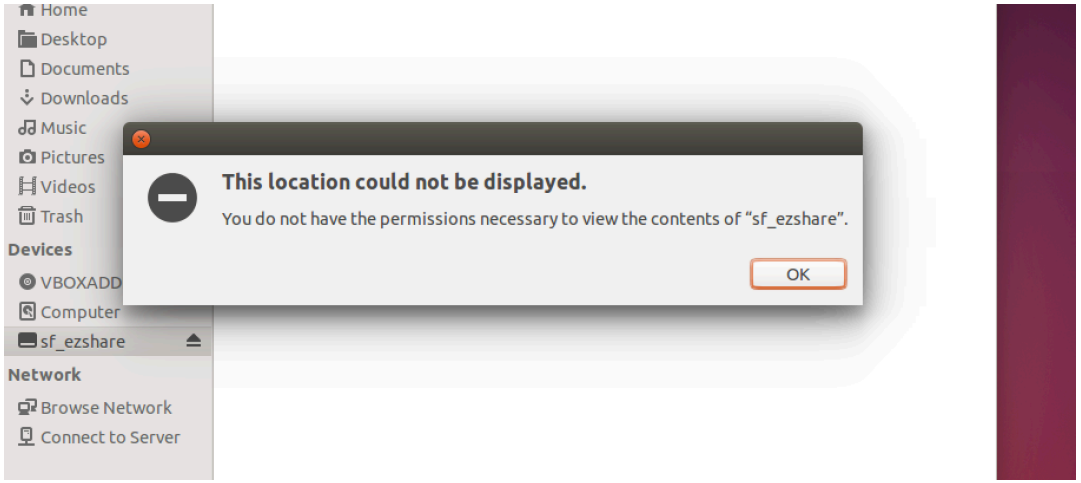
Beowulf Cluster - Step 0

- In order to share a Host folder



Beowulf Cluster - Step 0

- If it gives permission denied, open terminal on your VM and type



  - sudo adduser *user-name* vboxsf
  - where *user-name* is your user account name. Log out and log back in to Ubuntu.

Beowulf Cluster - Step 0

- One final thing to complete the Step 0.

- Stop the *master* VM and right click to open the VirtualBox settings for our VM.
  - Click the *Network* tab
  - Make sure *Enable Network Adapter* is checked
  - From *Attached to: drop down* choose *Bridged Adapter*

- What does it mean?

Some networking review

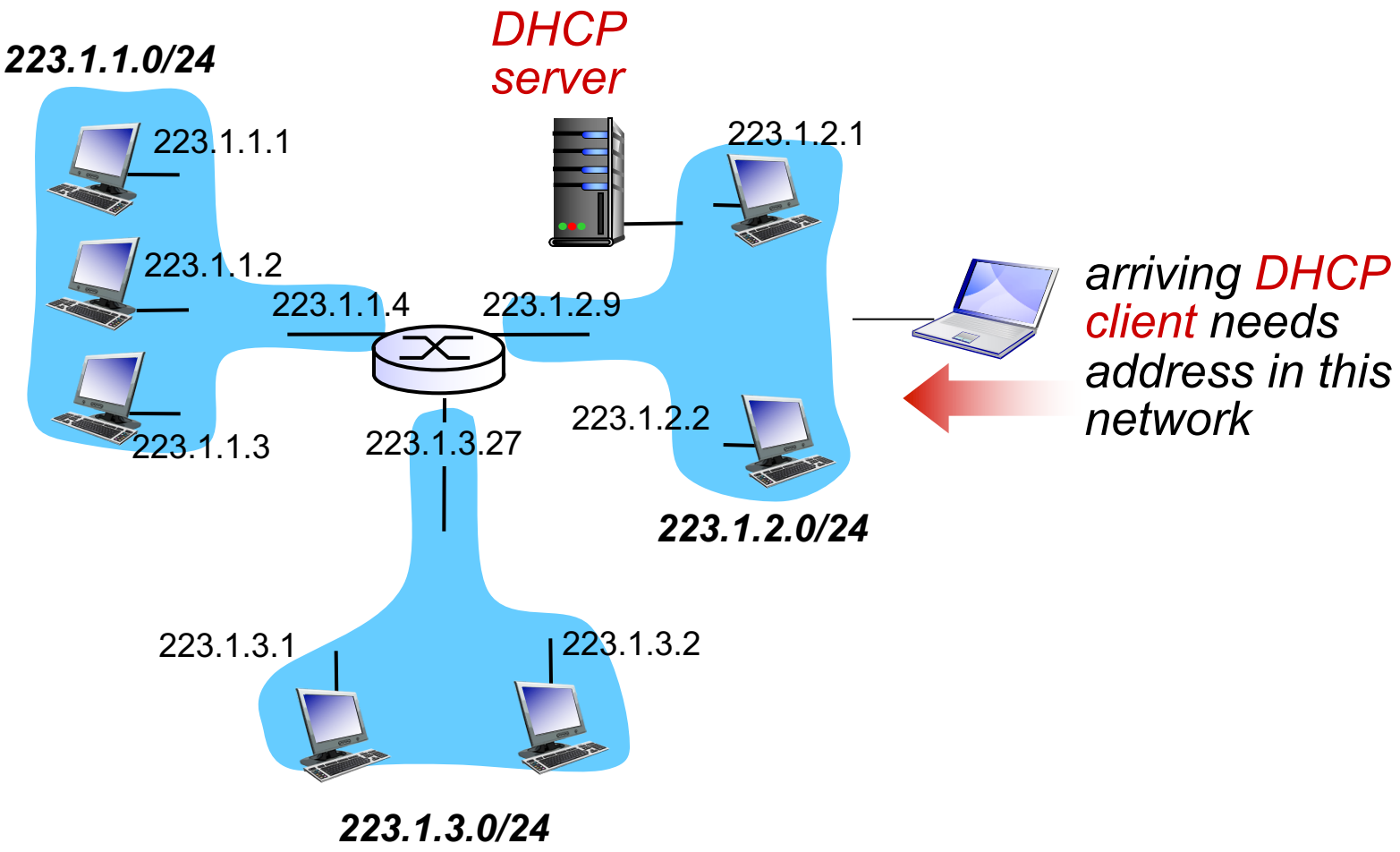
- We will get back to networking options in VirtualBox, but first some quick review about some basic concepts

DHCP: Dynamic Host Configuration Protocol

goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/"on")

DHCP client-server scenario

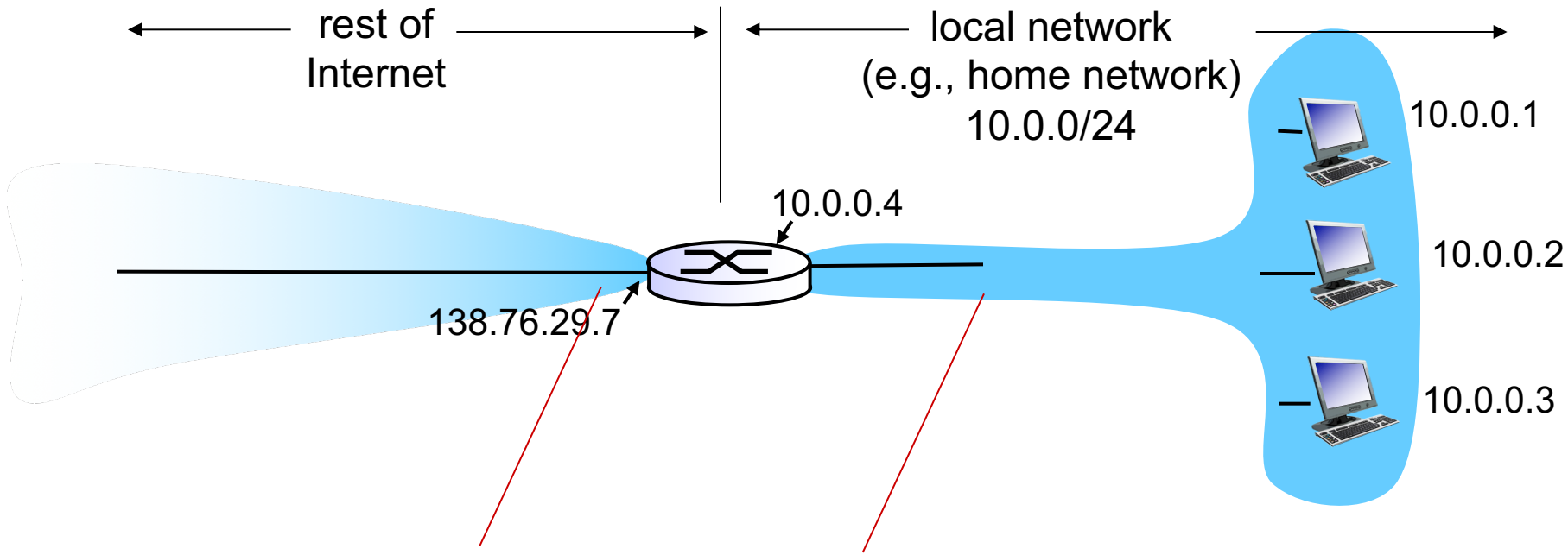


DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

NAT: network address translation



all datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7,different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: network address translation

motivation: local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

Networking in VirtualBox

- VirtualBox allows you to configure how the NIC operates with respect to your Host's physical networking. The main modes are:
    - Network Address Translation (NAT)
    - Bridged networking
    - …

Network Address Translation (NAT)

- Default mode – works best when the Guest is a "client" type of vm. (i.e. most network connections are outbound).
  - In this mode, every vm is assigned the same IP address because each vm thinks they are on their own isolated network.
  - And when they send their traffic via the gateway VirtualBox rewrites the packets to make them appear as though they originated from the Host, rather than the Guest (running inside the Host).

Bridged Networking

- Bridged Networking allows your VM to be a full network citizen, i.e. to be equal to your host machine on the network.
- The effect of this is that each VM has access to the physical network in the same way as your host.

- As we are expecting both incoming and outgoing connections, our VMs cam be configured to use this mode

Internal Networking

- In this mode, VirtualBox ensures that all traffic on that network stays within the host and is only visible to vm's on that virtual network

- VirtualBox provides no "convenience" services such as DHCP

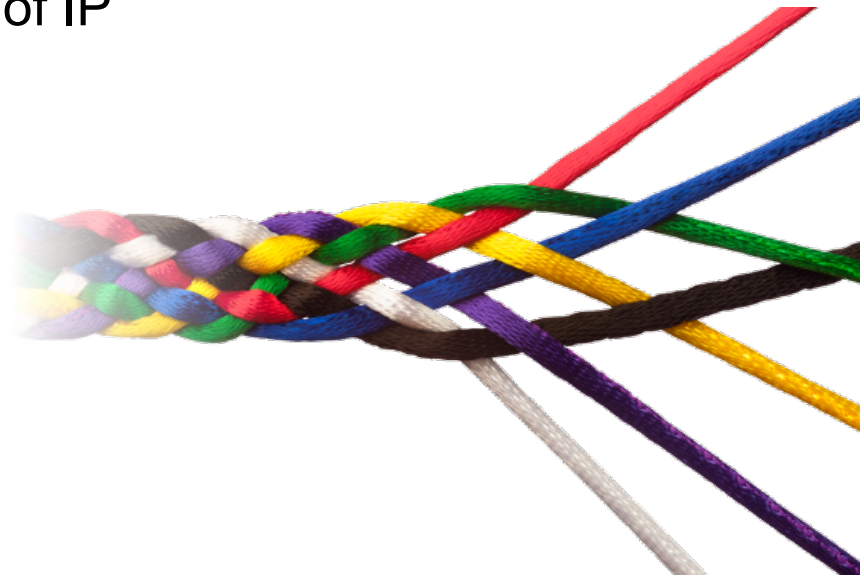
Beowulf Cluster - Step 0: Completion

- At the end of step-0, you should have:
  - a working VM named master
  - with Ubuntu Desktop installed
  - that is able to connect to internet using *bridged adapter* mode

- You may want to *clone (right click VM in VirtualBox and click Clone)* to be used later.

Beowulf Cluster – Step 1: Connectivity

- What we want to achieve in this step
  - Assign a static-ip to your VM named *master*
  - Clone you *master* node to create a *slave* node
  - Allow your *master* to communicate with *slave* nodes by name instead of IP



Beowulf Cluster – Step 1: Static IP

- If you have installed the Desktop version of Ubuntu, there is no need to modify any files and you can directly change IP using Network Settings.

Beowulf Cluster – Step 1: Static IP

- By default your VM uses DHCP to obtain an IP address at startup.
  - What does it mean?

- We want our VM to have the same IP even if it restarts.
- How it can be done? Do not use DHCP ☺

Beowulf Cluster – Step 1: Add a Slave

- It is now time to introduce a Slave.
  - This is easy – just *clone (right click master VM in VirtualBox and click Clone)* the existing VM and we are done.
  - Name the new node Slave1

- Slave1 is exact replica of master node, everything is inherited including static IP!

Beowulf Cluster – Step 1: Add a Slave

- Change the IP address using the GUI
- To test your configuration, you should be able to **ping** from master to slave1 and vice-versa.

- In my configuration, master has 192.168.8.109 and I assigned 192.168.8.110 to my slave1 (192.168.8.111 to slave2 and so on).

Beowulf Cluster – Step 1: Add a Slave

- To test your configuration, you should be able to **ping** from master to slave1 and vice-versa.

- On master `ping 192.168.8.110 -c 3` should be success, substitute the slave1 IP in your case.

Beowulf Cluster - Step 1: Access by name

- It would be even better if our nodes can access each other by name instead of IP.
- For this we need to edit the hosts file and add names (whatever we like) corresponding to IP addresses for our host.

```
sudo gedit /etc/hosts
```

Beowulf Cluster – Step 1: Access by name

- The /etc/hosts file for my nodes looks like this, you can just modify IPs and copy-paste in both *master* and *slave1* nodes.

```
127.0.0.1       localhost
192.168.8.109   master
192.168.8.110   slave1
192.168.8.111   slave2
192.168.8.112   slave3
```

- As I intend to use two other slaves by cloning slave1, so I have added their infor in advance.

Beowulf Cluster - Step 1: Access by name

- To test your configuration, you should be able to **ping by name** from master to slave1 and vice-versa.

- On master `ping 192.168.8.110 -c 3` should be success, substitute the slave1 IP in your case.

Beowulf Cluster – Step 1: Hostname

- Change /etc/hostname for master and slave1 and update the name accordingly.

Beowulf Cluster – Step 1: Completion

- At the end of step 1, you should have
  - At least two nodes, we call them master and slave1
  - They have been assigned static IPs
  - You are able to ping from each machine to other, by using their names and/or IP addresses

Beowulf Cluster – Step 2: NFS

- What do we want to achieve in this step
  - For our cluster to work, different nodes should be able to share content, primarily the program to be executed on individual machines.
  - How it is done in a single shared memory system?

- We will configure and use The Network File System on our nodes. Just follow the steps.

Beowulf Cluster – Step 2: NFS

- What is NFS?
  - Need for speed? Yes! but in our context, it stands for Network File System which allows the client to automatically mount remote file systems and therefore transparently provide an access to it as if the file system is local.

- In our scenario we are going to export a file directory from our *master* node and mount it on the *slave1* node.

Beowulf Cluster - Step 2: NFS

- To get started, on the *master* node you need to install NFS server

```
sudo apt-get install nfs-server
```

- On the slave1 node, you need to install the NFS client

```
sudo apt-get install nfs-client
```

Beowulf Cluster - Step 2: NFS

- Make a folder in all nodes, we'll store our data and programs in this folder.

```
sudo mkdir /mirror
```

- We can now share the contents of this folder located on the master node to all the other nodes.
- For this we first edit the /etc/exports file on the master node

Beowulf Cluster - Step 2: NFS *master*

- Here are some common NFS export techniques and options:

/home/nfs/ 10.1.1.55(rw,sync)	export /home/nfs directory for host with an IP address 10.1.1.55 with read, write permissions, and synchronized mode
/home/nfs/ 10.1.1.0/24(ro,sync)	export /home/nfs directory for network 10.1.1.0 with netmask 255.255.255.0 with read only permissions and synchronized mode
/home/nfs/ *(ro,sync)	export /home/nfs directory for any host with read only permissions and synchronized mode

- We want our /mirror folder in *master* node to be shared with slave nodes in read-write mode, what should we do ?

Beowulf Cluster - Step 2: NFS *master*

- In order to do this we first edit the /etc/exports file on the ***master*** node to contain the additional line

```
/mirror *(rw,sync)
```

- This can be done using a text editor as before or by issuing this command:

```
echo "/mirror *(rw,sync)" | sudo tee -a /etc/exports
```

Beowulf Cluster - Step 2: NFS *master*

- We need to restart the nfs service on the master node to parse this configuration once again.

```
sudo service nfs-kernel-server restart
```

Beowulf Cluster - Step 2: NFS *master*

- To later test our setup, lets create a blank file in the /mirror directory of the master node

```
sudo touch /mirror/sampleFileForMySlaves
```

- We are done with NFS configuration on the master node, lets move on to the *slave1* node.

Beowulf Cluster - Step 2: NFS *slave1*

- On the *slave1* node all we need to do is to mount the shared folder

```
sudo mount master:/mirror /mirror
```

- But it's better to change fstab in order to mount it on every boot. We do this by editing /etc/fstab and adding this line:

```
sudo gedit /etc/fstab
```

```
add the line
master:/mirror    /mirror      nfs
```

Beowulf Cluster – Step 2: Completion

- At the end of step 2, you should have
  - At least two nodes, we call them master and slave1, which have been assigned static IPs
  - You are able to ping from each machine to other, by using their names and/or IP addresses
  - The nodes can share files using NFS, /mirror on master node is shared with /mirror on slave1

Beowulf Cluster – Step 3:SSH setup

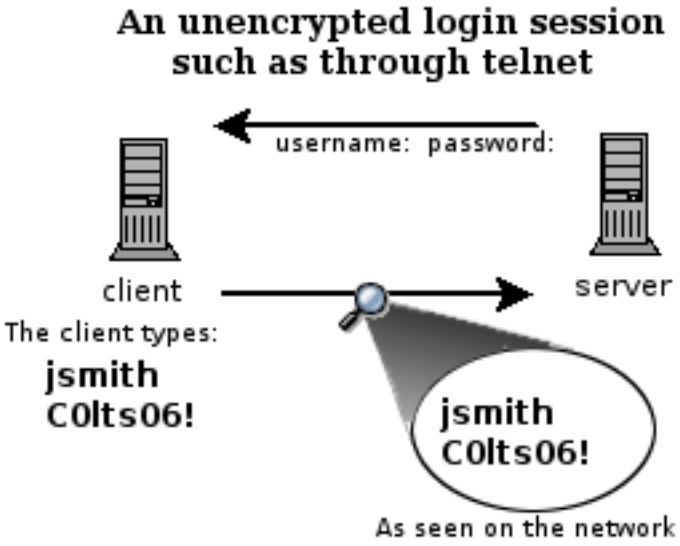
- For the cluster to work, the master node needs to be able to communicate with the slave nodes, and vice versa.
- Secure Shell (SSH) is usually used for secure remote access.

- By setting up password-less SSH between the nodes, the master node is able to run commands on the compute nodes.

- But what is this Secure Shell anyway?

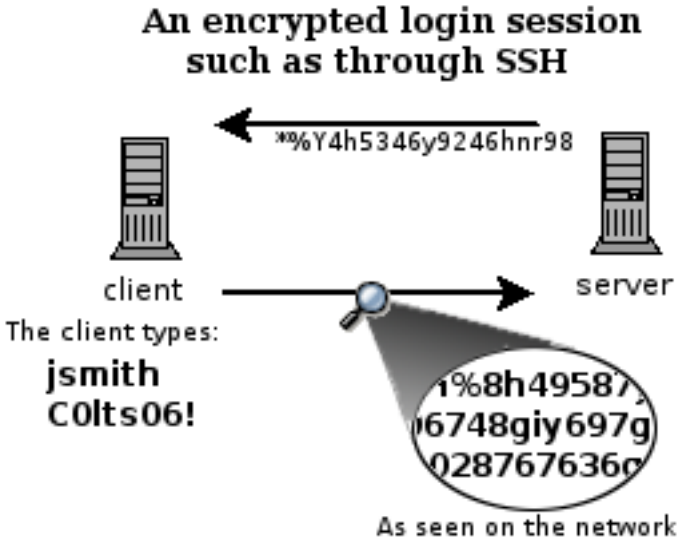
Secure Shell -SSH

- There are a couple of ways that you can access a shell (command line) remotely on most Linux/Unix systems.
  - Telnet - everything that you send or receive over that telnet session is visible in plain text on the network,
  - We know how to sniff, remember Wireshark, so not that secure



Secure Shell - SSH

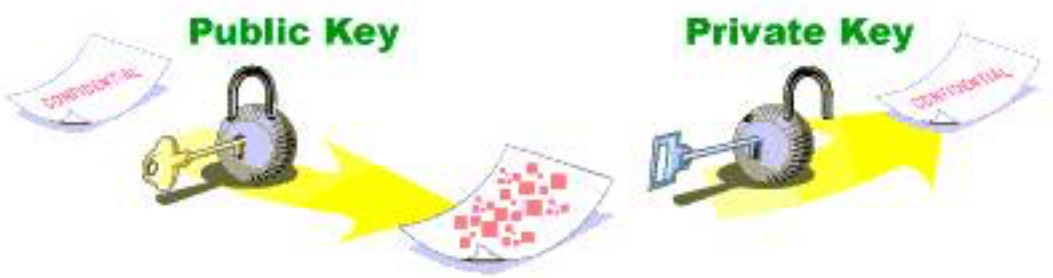
- There are a couple of ways that you can access a shell (command line) remotely on most Linux/Unix systems.
  - SSH, which is an acronym for Secure SHell, was designed and created to provide the best security when accessing another computer remotely.



Secure Shell, SSH - Key Generation

- The basic idea is to use public/private keys pair
  - What does it mean?

- Some network security review !!



There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: A lot

- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* "take over" ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

Network security

- field of network security:
  - how bad guys can attack computer networks
  - how we can defend networks against attacks
  - how to design architectures that are immune to attacks

What is network security?

Confidentiality: only sender, intended receiver should "understand" message contents

- sender encrypts message
- receiver decrypts message

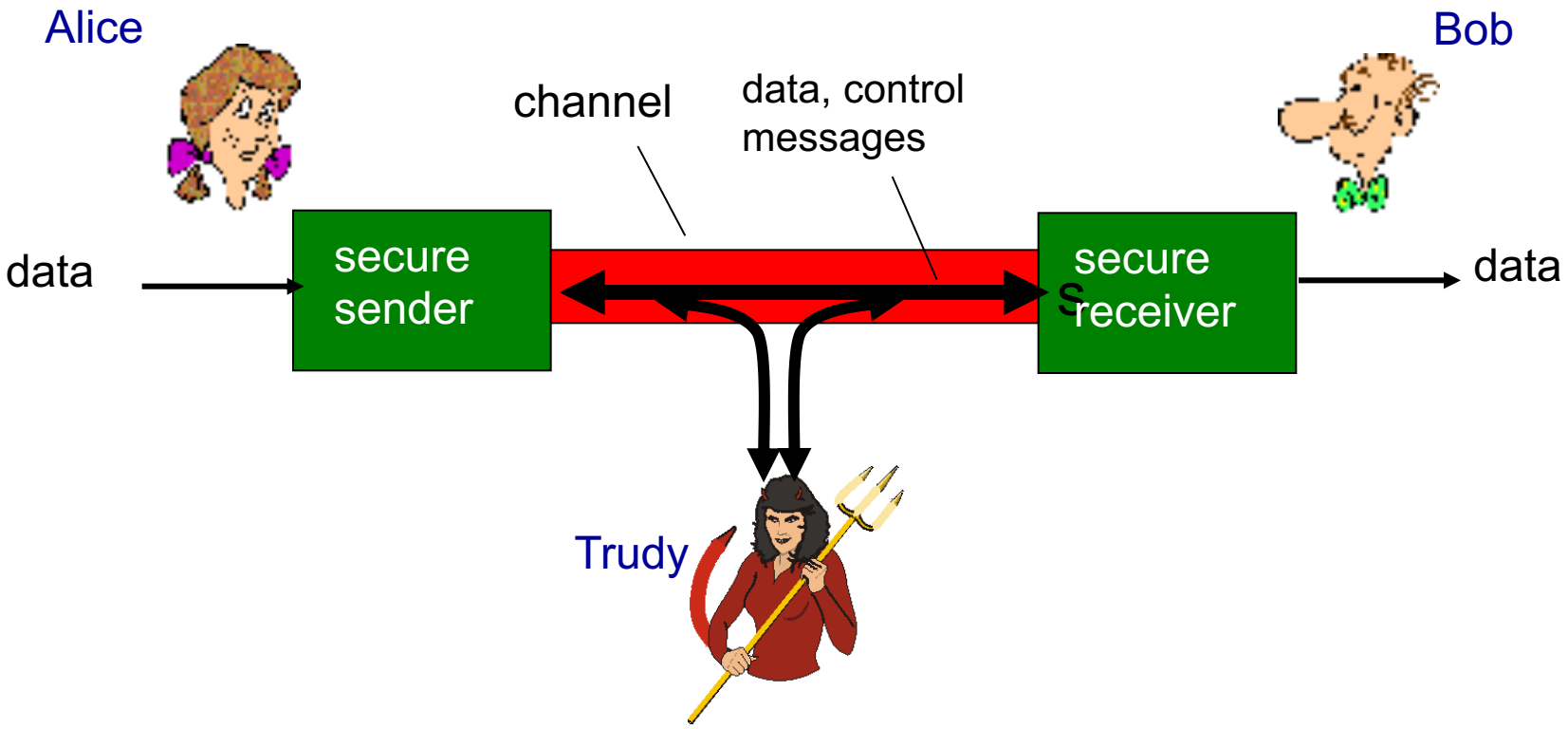
Authentication: sender, receiver want to confirm identity of each other

Message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Access and Availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

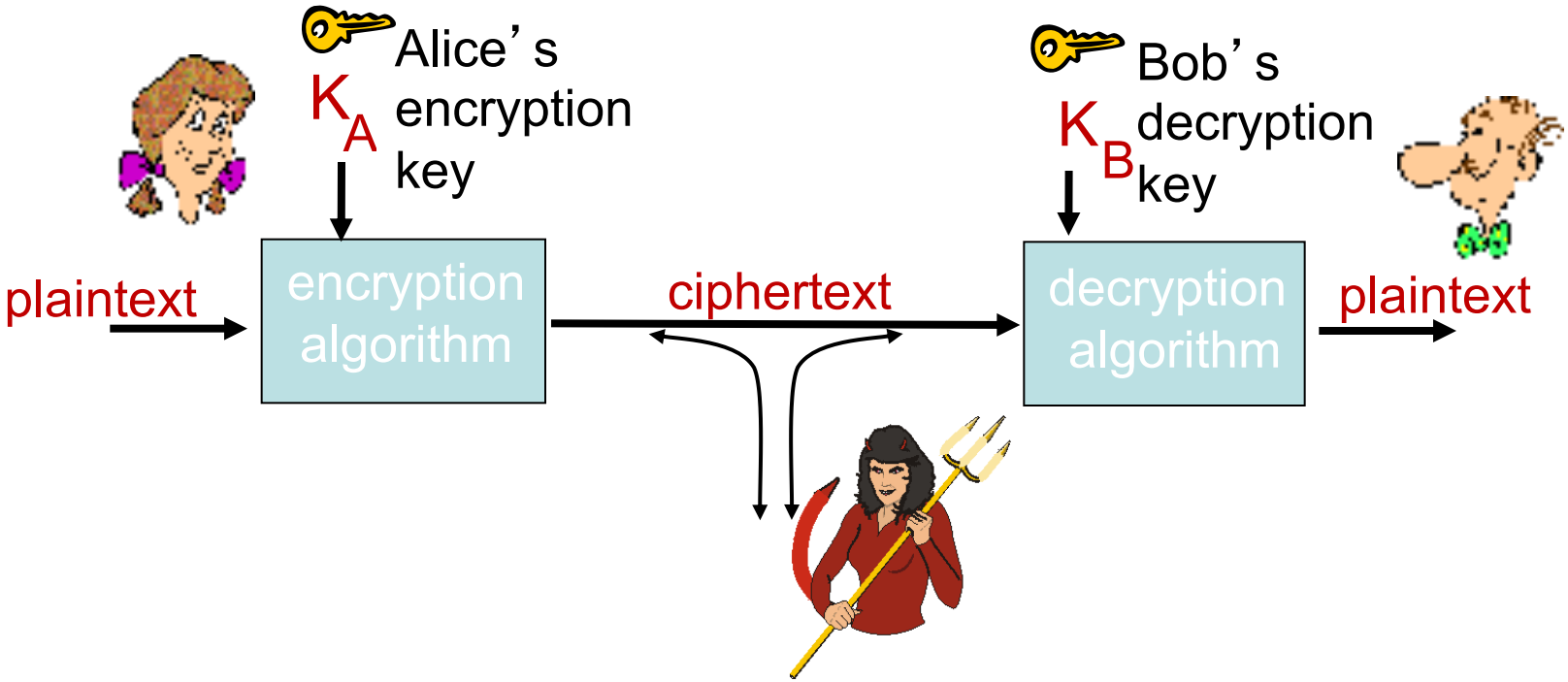
- well-known in network security world
- Bob, Alice (lovers!) want to communicate "securely"
- Trudy (intruder) may intercept, delete, add messages



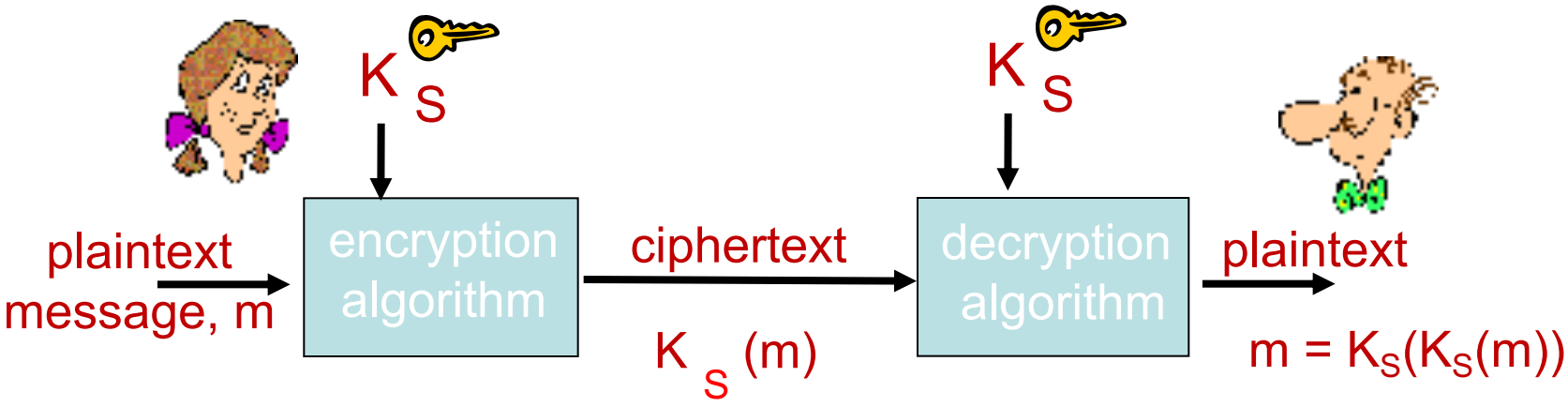
Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- …

The language of cryptography



Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

Q: how do Bob and Alice agree on key value?

Public Key Cryptography

symmetric key crypto

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key crypto

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

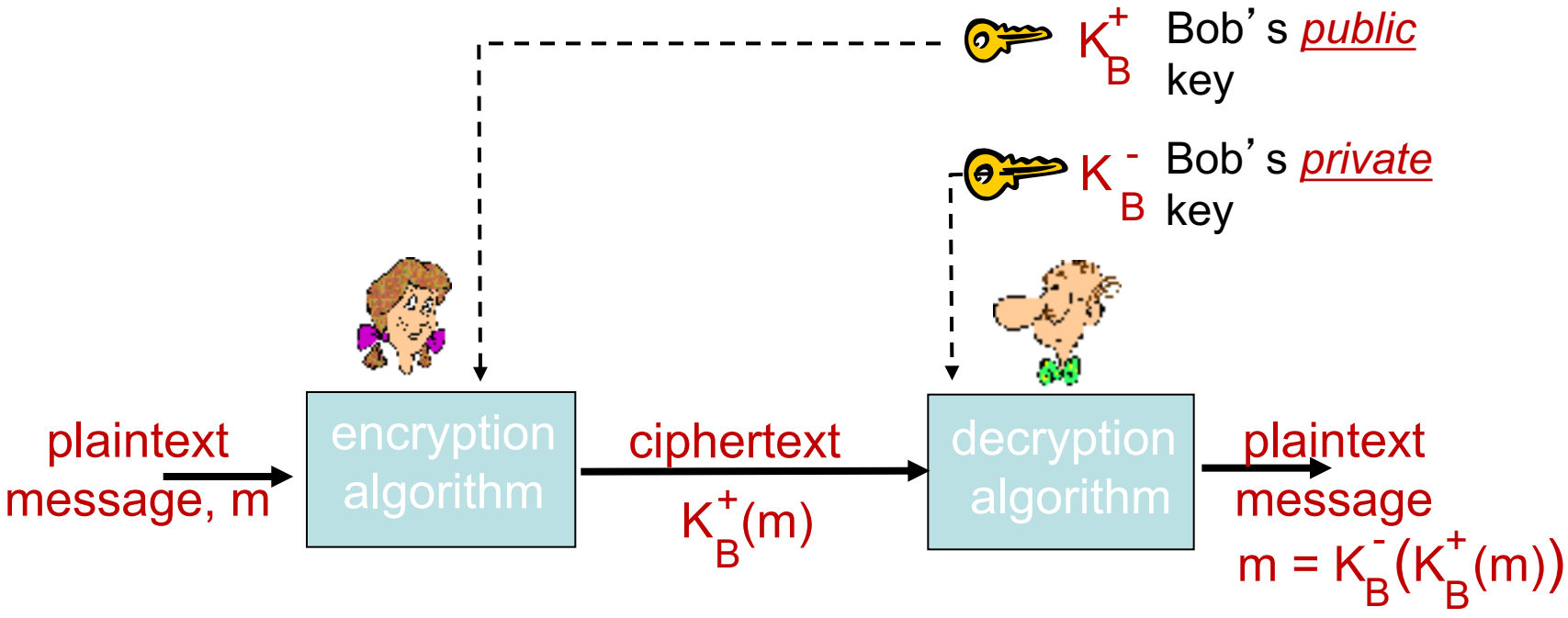
Public and Private Keys

- The Public and Private key pair comprise of two uniquely related cryptographic keys (basically long random numbers).

- Below is an example of a Public Key:

3048 0241 00C9 18FA CF8D EB2D EFD5 FD37 89B9 E069 EA97 FC20 5E35 F577
EE31 C4FB C6E4 4811 7D86 BC8F BAFA 362F 922B F01B 2F40 C744 2654 C0DD
2881 D673 CA2B 4003 C266 E2CD CB02 0301 0001

Public key cryptography



Message Integrity

Alice receives msg from Bob, wants to ensure:

- message originally came from Bob
- message not changed since sent by Bob

Digital signatures

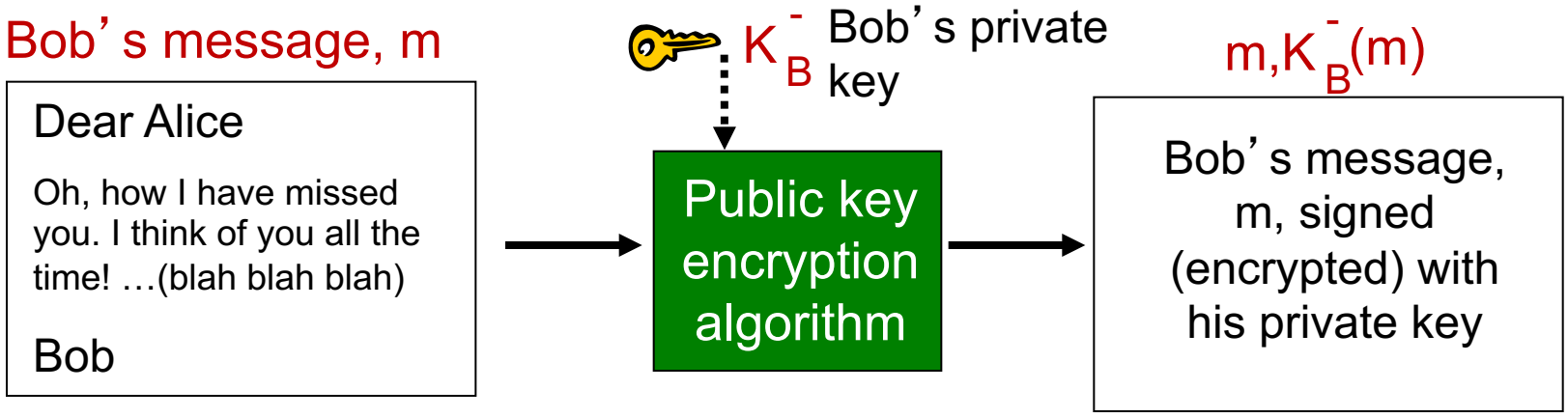
cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital signatures

simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Digital signatures

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key

Alice thus verifies that:

- ✓ Bob signed m
- ✓ no one else signed m
- ✓ Bob signed m and not m '

non-repudiation:

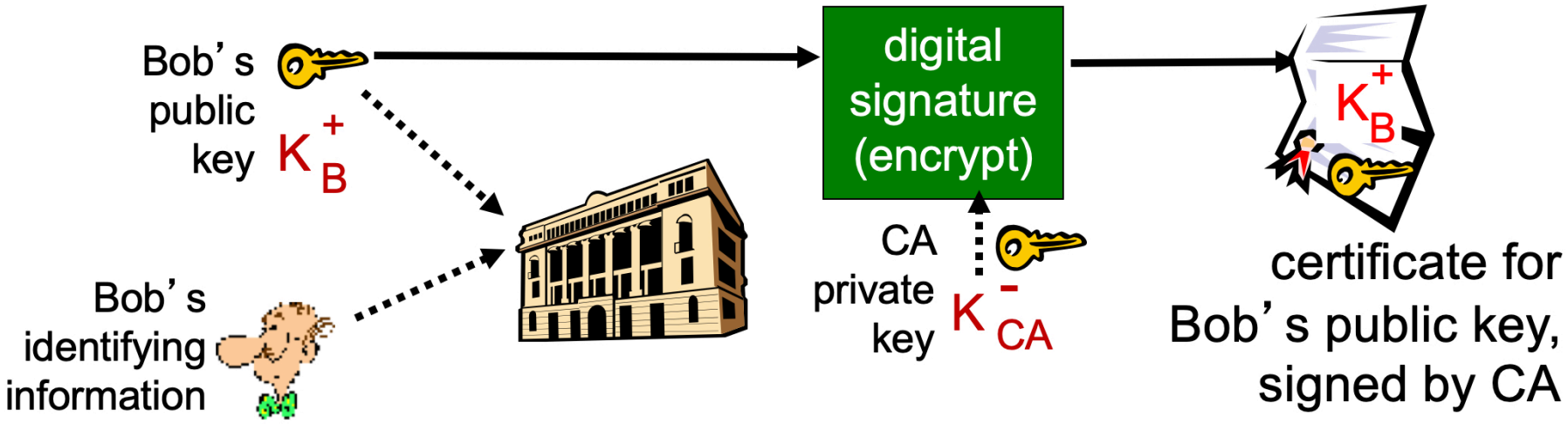
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Public-key certification

- motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four cheese pizzas.
    Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four cheesepizzas to Bob
  - Bob doesn't even like cheese

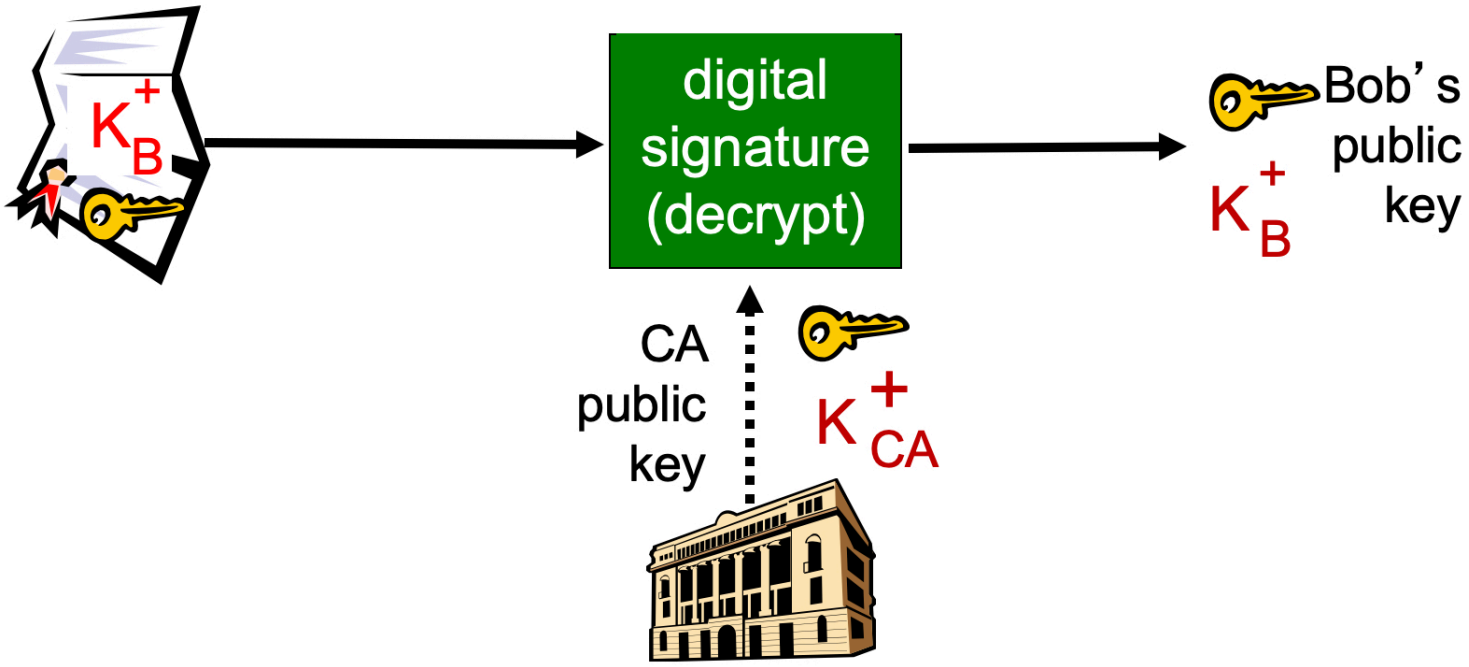
Certification authorities

- *certification authority (CA):* binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



Certification authorities

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



Lecture 05: Building a Beowulf Cluster

Secure Shell - SSH Installation

- The version of SSH that you will use on Linux is called OpenSSH
- Use apt-get to install it on ALL NODES

```
sudo apt-get install openssh-server
```

Beowulf Cluster - Step 3: Create User

- We define a user with same name and same userid in **all nodes** with a home directory in /mirror. We name it mpiuser, this can be done by using the following commands

```
sudo useradd -d /mirror mpiuser
sudo passwd mpiuser
```

- Choose an easy to remember password, same for all nodes.

Beowulf Cluster - Step 3: Create User

- You need to also change ownership for /mirror

```
sudo chown mpiuser /mirror
```

- NOTE: On Slave, you need to first unmount before chown. Use these commands

```
cd ~
sudo umount /mirror
sudo mount master:/mirror /mirror
```

Beowulf Cluster – Step 3: SSH Key Generation master

- So lets start with the key generation for the user we created, called mpiuser, on master node.
  - First switch users using su and typing password

```
su – mpiuser , make sure once switched your home
directory is really /mirror, type pwd to test
```

  - Then generate the public/private keys using

```
ssh-keygen -t rsa
```

- Use default path i.e probably /mirror/.ssh ….
- Do use empty passphrase, not recommended in general but this would be our little hack to enable password less connection ☺

Beowulf Cluster – Step 3: SSH Key Usage

- Now, whenever you connect via ssh to a host that has your public key loaded in the authorized_keys file, it will use your private key and public key to determine if you should be granted access

- So our master has the public key, what about others?
  - In which folder you generated the keys? /mirror/.ssh/..
  - So? As /mirror is shared by nfs. All our nodes have the keys ☺

Beowulf Cluster – Step 3: SSH Key Usage

- What is all we need to do now is that the public SSH key of the *master* node needs to be added to the list of known hosts (this is usually a file ~/.ssh/authorized_keys) of all compute nodes.

- But this is easy, since all SSH key data is stored in one location: /mirror/.ssh/ on the master node. So instead of having to copy master's public SSH key to all compute nodes separately, we just have to copy it to master's own authorized_keys file.

Beowulf Cluster – Step 3: SSH Key Setup

- This is what we need to do only on the master node

```
cd .ssh
cat id_rsa.pub >> authorized_keys
```

- To test SSH, run ssh slave1, you should be able to ssh into the slave1. Use exit to end your session.

For the first time, you may be presented an option to permanently add the host to a list of known_hosts, choose yes.

Beowulf Cluster – Step 3: Completion

- At the end of step 3, you should have
  - At least two nodes, we call them master and slave1, which have been assigned static IPs and can be ping-ed by name.
  - The nodes can share files using NFS, /mirror on master node is shared with /mirror on slave1
  - The nodes can be SSHed from each other using mpiuser and without using any password.

Beowulf Cluster - Step 4: MPICH

- To be able to run applications on our Cluster, we need the actual middleware, the message passing implementation that coordinates and manages processes

- We should also be able to compile code on our *master* node.
- You can get gcc and other necessary stuff by installing the build-essential package:

```
sudo apt-get install build-essential
```

Beowulf Cluster - Step 4: MPICH

- To be able to run applications on our Cluster, we need the actual middleware, the message passing implementation that coordinates and manages processes

- Now the last ingredient we need installed on all the machines is the MPI implementation. You can install MPICH using by typing:

```
sudo apt-get install mpich
```

Beowulf Cluster - Step 4: MPICH

- To test that the program did indeed install successfully enter this on all the machines:

```
which mpiexec
which mpirun
```

Beowulf Cluster - Step 4: Completion

Congratulations!! You are all done, you have a cluster platform now

But wait, we still need to test if it works ☺

Beowulf Cluster - Testing

- On master node, Create a file called "machinefile" in mpiuser's home directory (/mirror) with node names followed by a colon and a number of processes to spawn:

slave1:4 # this will spawn 4 processes on slave1
Master:4 # this will spawn 4 processes on master

Beowulf Cluster - Testing

- In the same directory /mirror in master, write this MPI helloworld program in a file mpi_hello.c

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
    int myrank, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    printf("Hello from processor %d of %d\n", myrank, nprocs);

    MPI_Finalize();
    return 0;
}
```

Beowulf Cluster - Testing

Compile the code using,

mpicc mpi_hello.c -o mpi_hello

and run it

mpiexec -n 8 -f machinefile ./mpi_hello

the parameter next to -n specifies the number of processes to spawn and distribute among nodes

Beowulf Cluster - Testing

You should see output as below,

Hello from processor 0 of 8
Hello from processor 1 of 8
Hello from processor 2 of 8
Hello from processor 3 of 8
Hello from processor 4 of 8
Hello from processor 5 of 8
Hello from processor 6 of 8
Hello from processor 7 of 8

Congratulations! You have a working MPI platform to build applications