Beowulf Cluster

(Setup & Configurations)*

(CS 3006)

Dr. Shujaat Hussain

Department of Computer Science,

National University of Computer & Emerging Sciences,

Islamabad Campus

Beowulf Cluster

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

- **Beowulf** is a **cluster** of **computers** (``**compute nodes**'') **interconnected** with a **network** with the **following typical characteristics**:
  - **dedicated homogeneous nodes**
  - **Off-the-shelf machines**
  - **Standard network**
  - **Open-source software**
  - **Linux platform**
  - One **special node** called **Head** or **Master node**

Slides credits: https://webhome.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/

But we dont have machines …

- **We can use virtual machines!!**

- For the ***actual* Cluster**, it **doesnt matter** if we are using the ***virtual*** or **physical** machines, the **steps remain exactly the same**

Beowulf Cluster – Step 0: Setup

- **Install Oracle VirtualBox**
  - **Open source,** 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 **<u>choose default options</u>**
  - **Choose easy to remember username/password**
  - **Choose correct region**

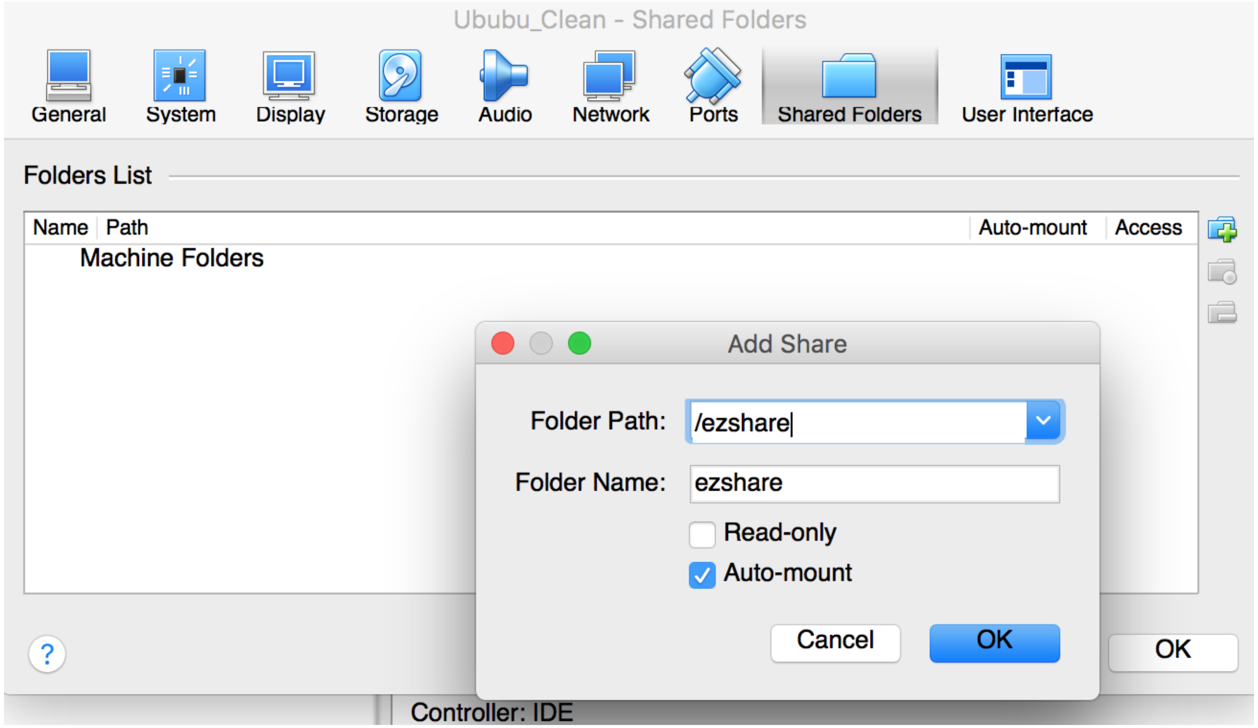
Beowulf Cluster – Step 0

- **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 …*

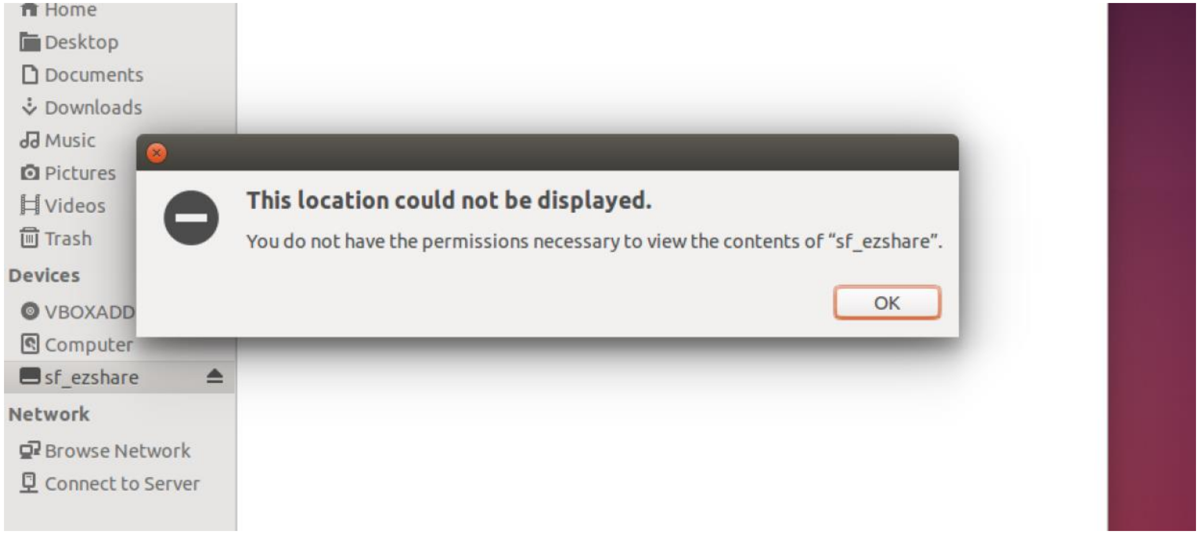
Beowulf Cluster – Step 0

- 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?**

Network Address Translation (NAT)

- **Default mode** - **works best when** the **Guest** is a "**client**" **type of VM**
  - i.e., **most network connections** are **out-bound**
  - **Every VM is assigned the same IP address** (because each VM thinks they are on their own isolated network)
  - **When** they **send their traffic via 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 VM** to be a **full network citizen**, i.e., **to be equal** to your **host machine**

- **Each VM** has **access** to the **physical network** in the **same way** as **your host**

- This **enables** both **incoming** and **outgoing connections**

Internal Networking

- **VirtualBox ensures** that all **traffic** on that **network stays within** the **host** and is only **visible** to VM's on that **virtual network**

- **VirtualBox do not provide DHCP**

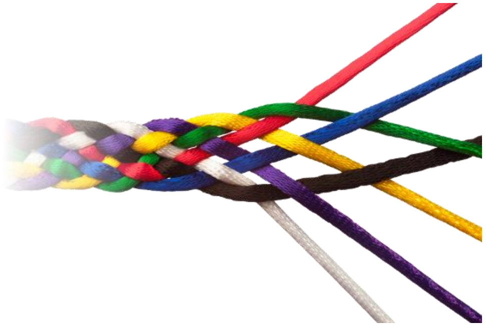
Beowulf Cluster – Step 0: Completion

- **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***
  - For later use

Beowulf Cluster – Step 1: Connectivity

- **What 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

- For **Ubuntu Desktop version**:
  - **No need** to **modify** any **files**
  - **Directly change IP** using **Network Settings**

- **Purpose**:
  - **VM** to have the **same IP even** if it **restarts**

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
```

`*do clone your VM befor this step`

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: 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
```

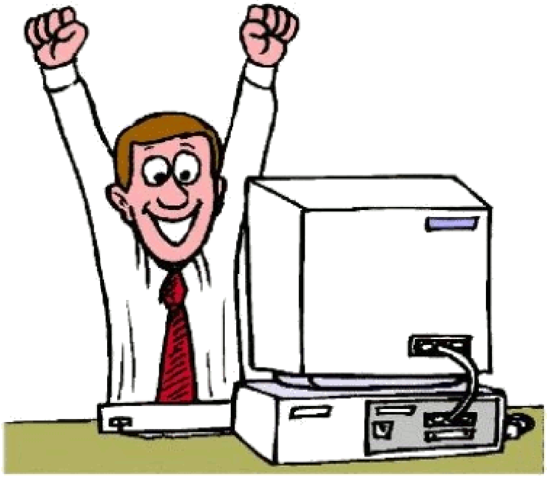
- But it's better to change fstab in order to mount it on every boot. We do this by editing /etc/fstab and adding this line:

```
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

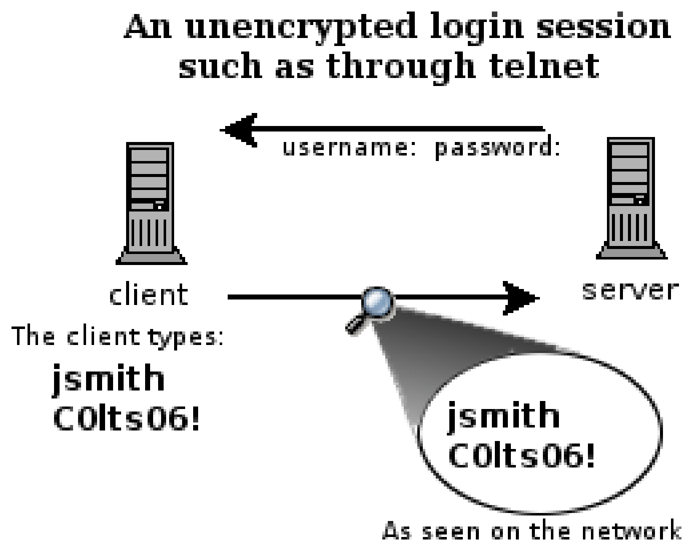
- For the cluster to work, the master node needs to be able to communicate with the slave nodes, and vice versa.
- Secure Shell (SSH) is usually used for secure remote access.

- 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?

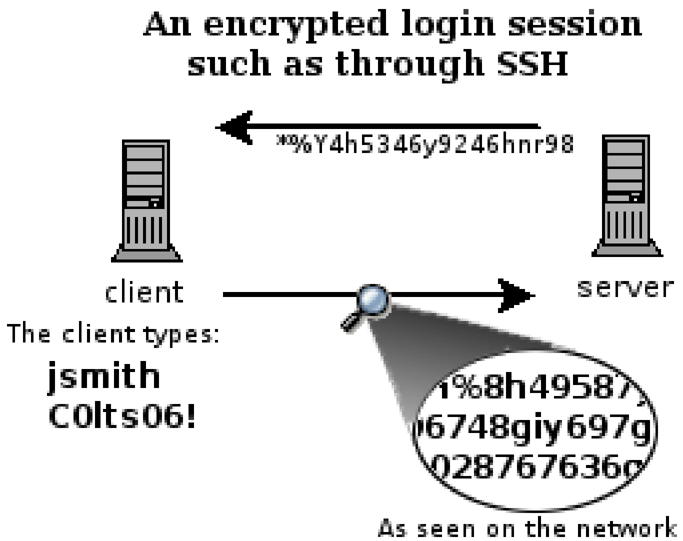
Secure Shell –SSH

- 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



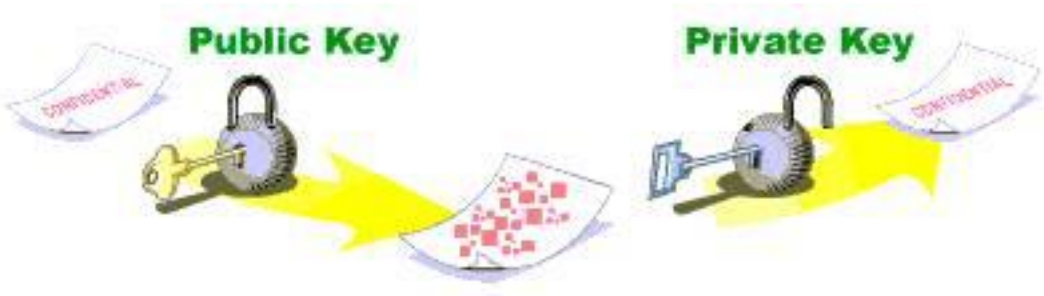
Secure Shell – SSH

- 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

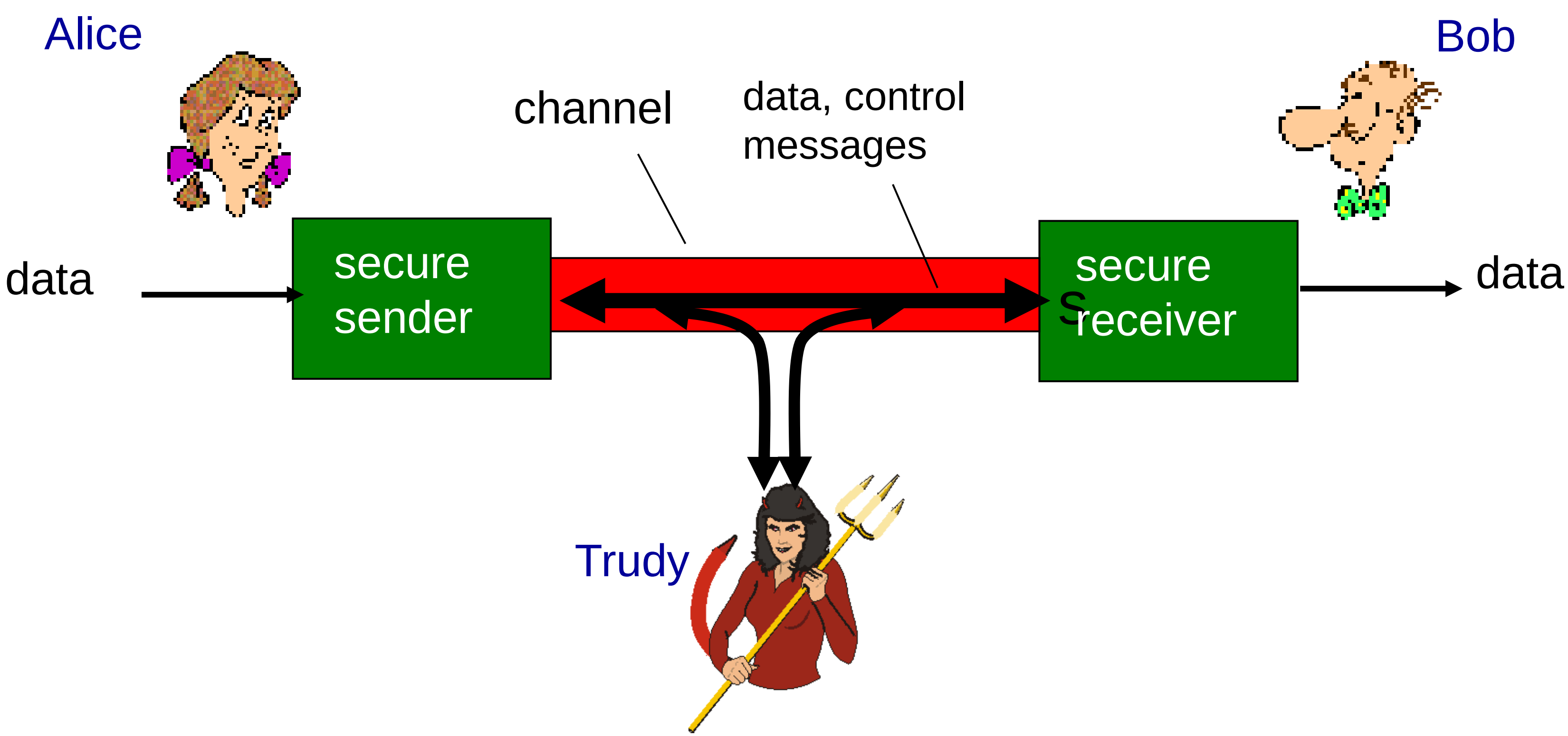
Authentication: sender, receiver want to confirm identity of each other

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

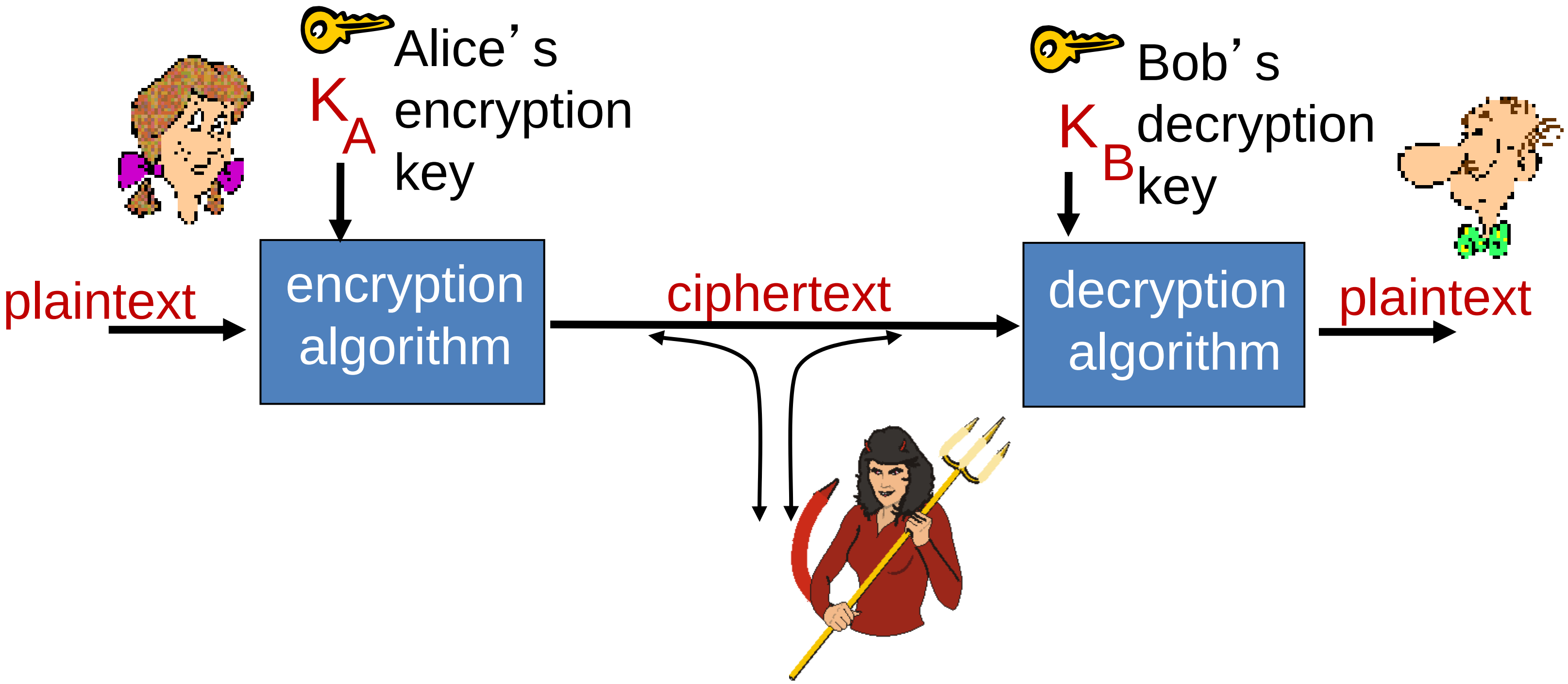
- well-known in network security world
- Bob, Alice (lovers!) want to communicate "securely"
- Trudy (intruder) may intercept, delete, add messages



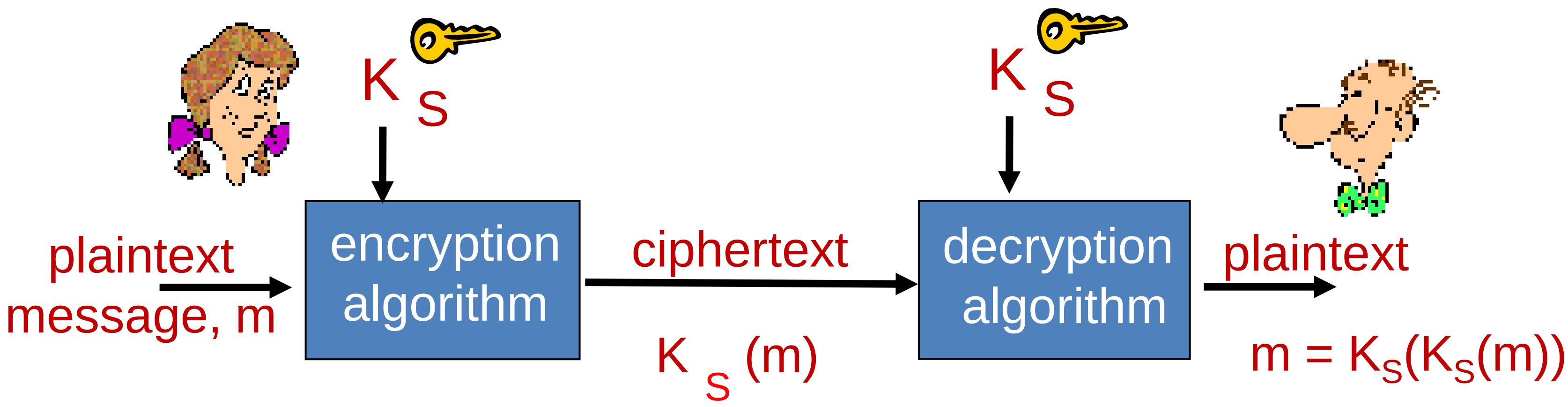
Who might Bob, Alice be?

- … 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



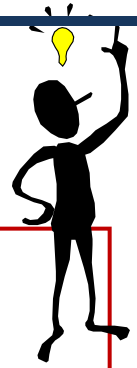
Symmetric key cryptography



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

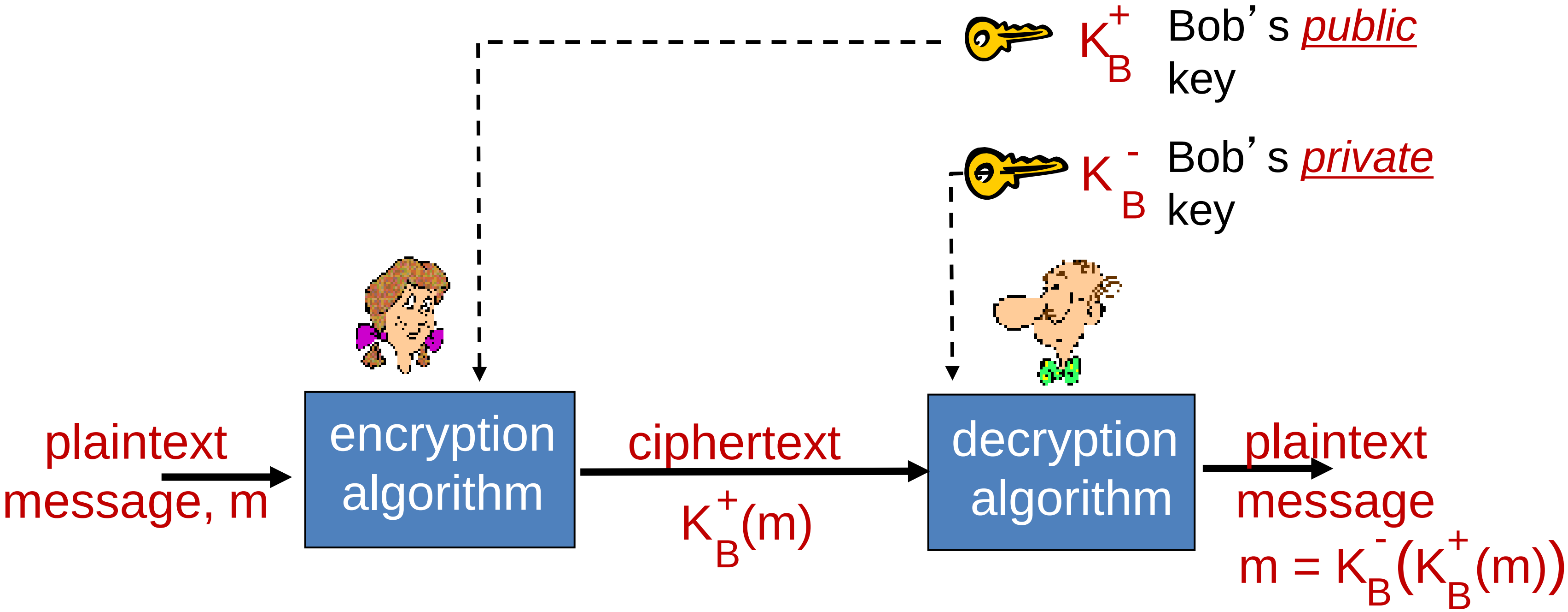
Public and Private Keys

- 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

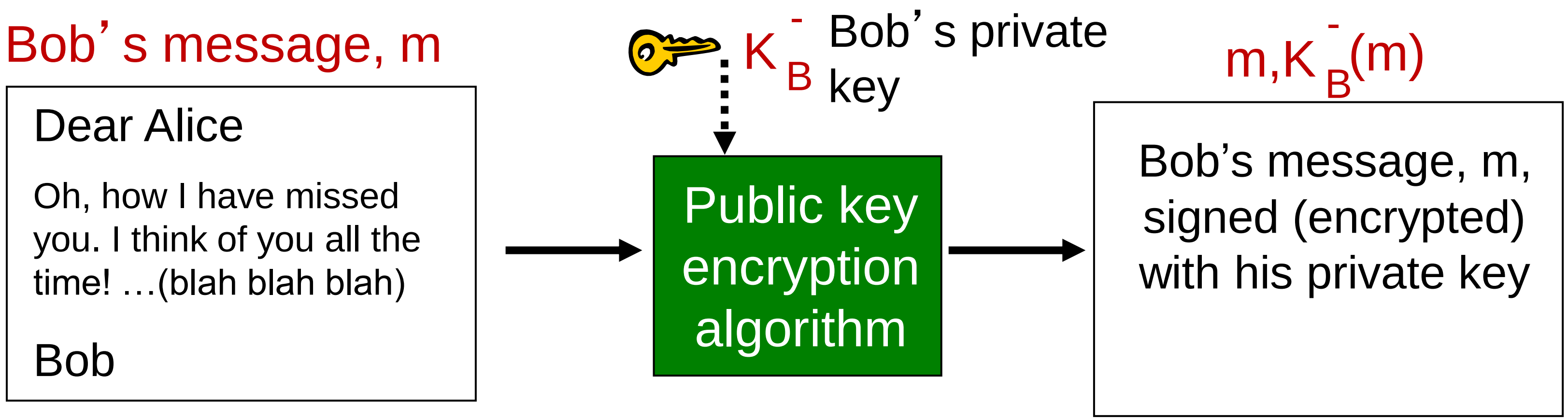
cryptographic technique analogous to hand-written signatures:

- 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

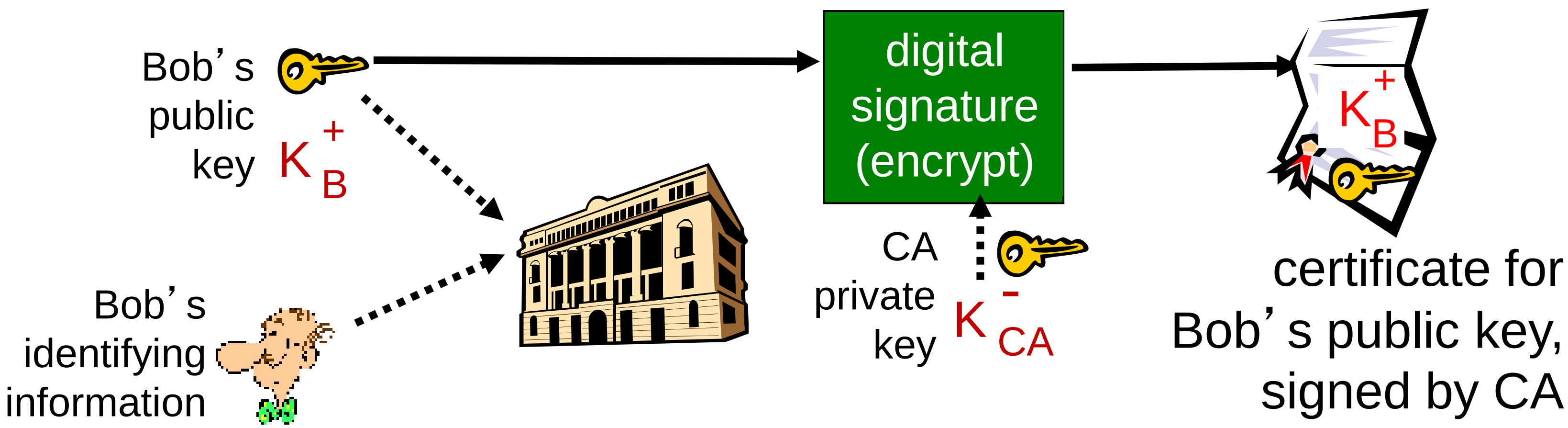
-

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

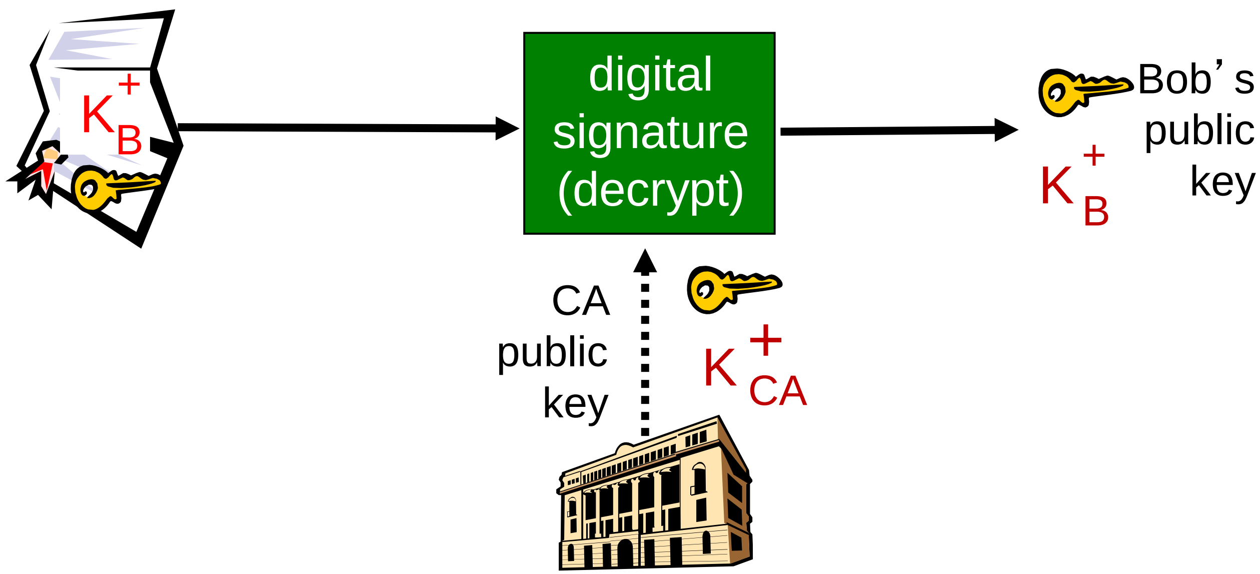
Certification authorities

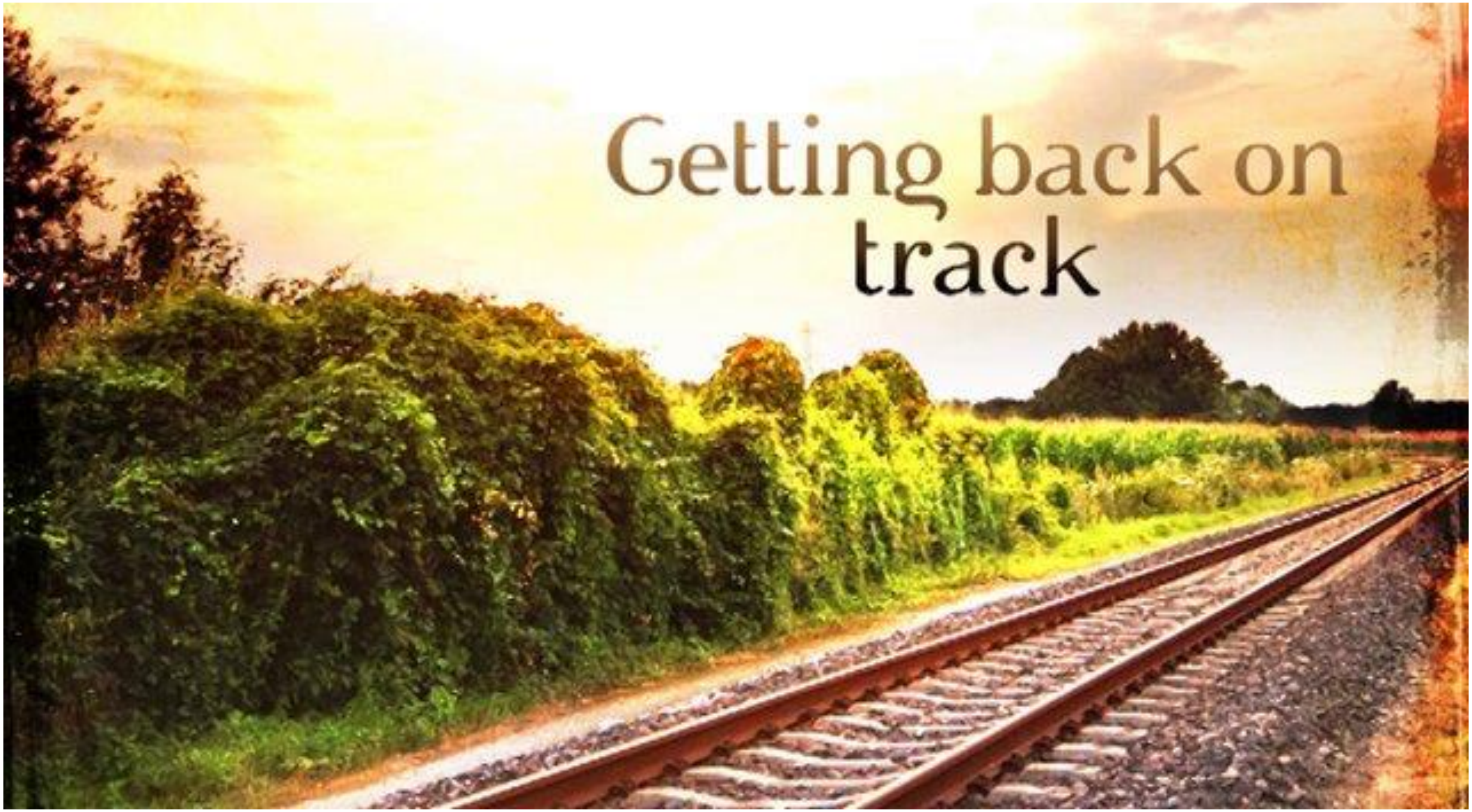
- *certification authority (CA):* binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says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




Getting back on
track

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
```

- Use ssh –V, to check for the installed version

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. You need to also change ownership for /mirror

```
sudo chown mpiuser /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

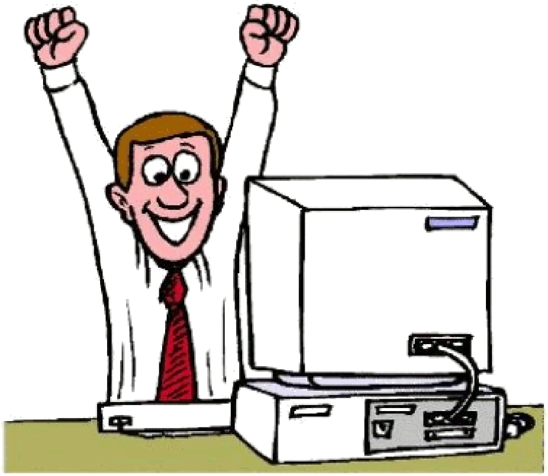
```
cd .ssh
cat id_rsa.pub >> authorized_keys
```

- 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: MPICH2

- 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: MPICH2

- 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 MPICH2 using by typing:

```
sudo apt-get install mpich2
```

Beowulf Cluster – Step 4: MPICH2

- 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