Languages and Compilers

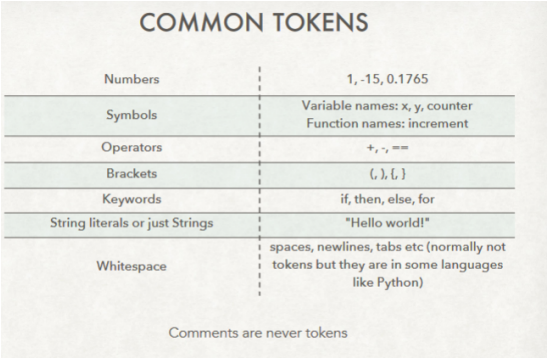
4 Phases of compilation Phase 1: Lexical Analysis
Groups characters into tokens
Phase 2: Parsing
Checks grammatical structure and builds internal representation of program
Phase 3: Semantic analysis and code generation
Analyses meaning; generates machine instructions
Phase 4: Code optimization
Improves efficiency of code in time or space required (less memory used)

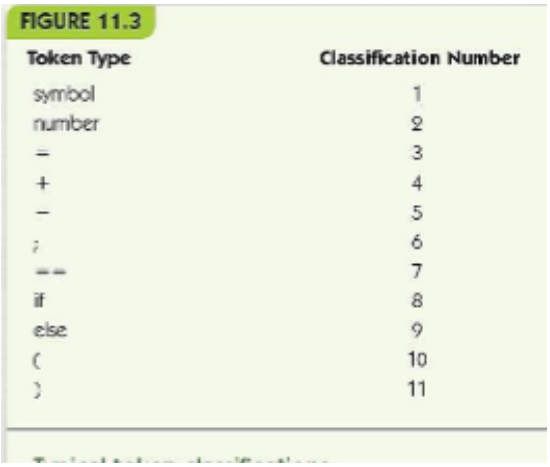
Phase 1

The input to a scanner is a high-level language statement from the source program
Its output is a list of all the tokens contained in that statement, as well as the classification of each token found

- A **lexical analyzer** or scanner or lexer
  - Groups input characters into tokens
  - Discards unnecessary characters
    - E.g. blanks, tabs, and comments
    - However, blanks and tabs are important to delimit tokens
  - Determines the type of each token
    - E.g. symbol, numbe, and left parenthesis
- Tokens are like the vocabulary of the source language

e.g. of splitting: *area = 3.14159 * radius * radius;* -> “area”, “=”, “3.14159”, “*”, “radius”, “radius”, “;"
A lexer opens the source file and reads each character in turn trying to match groups of characters into tokens. **There are rules for each language about what the tokens are**





Each token belongs to a class. Some classes have only one token (e.g. left parentheses) while others may be a collection of similar tokens (e.g. numbers) \

Parsing

A parser takes a list of classified tokens and:

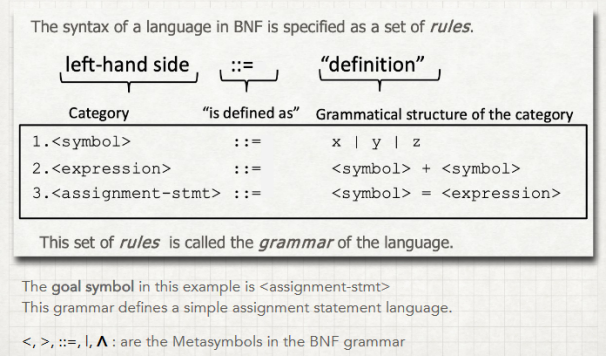
- Determines the grammatical structure
- Builds a parse tree
- Can tell us if the output from the lexer is a valid program (**syntactically correct**)

Syntax

- Grammatical structure
- Defined by rules (productions)
  - **BNF (Backus-Naur Form)** is one notation for describing rules
- A grammar is the set of rules that define a language

BNF Rules: left-hand side ::= right-hand side

- **Left-hand side** of a rule: grammatical catergory (a non-terminal)
- **Right-hand side** of a rule: pattern that captures the structure of caterogy
- **Terminals**: tokens from the lexical analyzer (sometimes written with <angle brackets> e.g. <number>)
- **Non-terminals**: grammatical catergories. Always written with <angle brackets>
  - There must be at least one rule with non-terminal on the left-hand side
- **Goal symbol**: a non-terminal which tells us if the program is valid
  - If the program produces a goal symbol it is syntactically correct

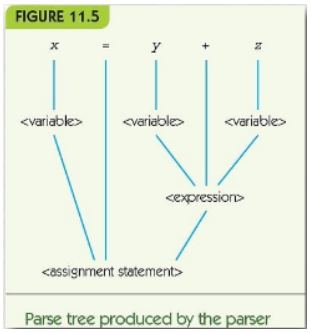


BNF has metasymbols (like language operators)

- <, >, these go around **non-terminal** (sometimes terminals)
- ::=, read as "is defined as"
- |, "or"
- ^, lambda and stands for empty string

The Parser

- Input to parser is a list of **classified tokens** from the lexer and **grammar** specifiying the syntax of the language
- Ouput is a **parse tree** that represents the **syntactic structure** of the program according to the grammar rules



The Parse Tree

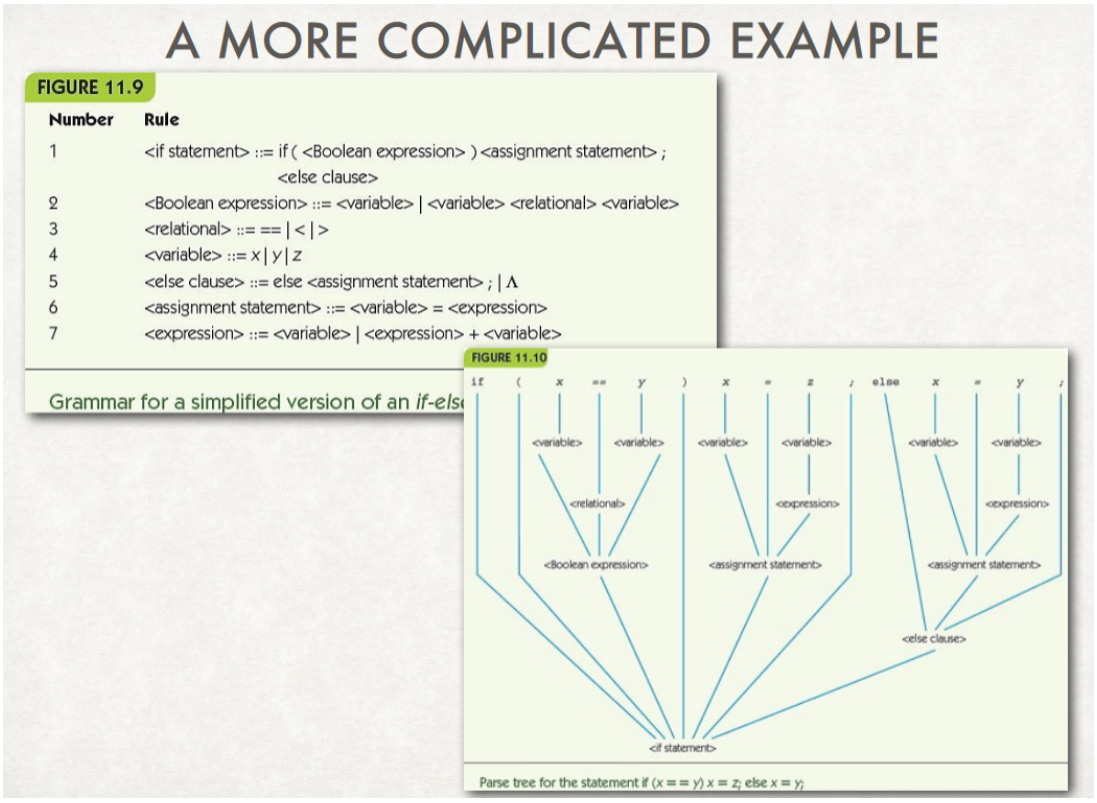
- Has its leaves and **tokens** returned by the lexer
- Uses the **rules of the grammar** to combine:
  - Leaves into branches,
  - Branches into "thicker" branches
- Each such combining step of leaves/branches into a single branch is called a **production**
- Eventually, there should be only one branch left: the root representing the **goal symbol**
- If such a tree cannot be built, it concludes that the code has a syntax error

Recursive Definition

A rule like "<**expression**> ::= <variable> | <**expression**> + <**expression**>" is a recursive definition.
The non-terminal of the left-hand side is defined in terms of itself
Sometimes we use lambda in recursive definitions e.g. "<lots of a> ::= a <lots of a> | ^".
This matches "a", "aa", "aaa", "aaaa", ...

Ambiguous Grammars

If the parser can produce **more than one parse tree** then the grammar is abiguous (not good)



Semantics and Code Generation

Semantic analysis checks if all the branches of the parse tree make sense
Semantic records store information about the values associated with non-terminals (also stored in symbol tables)

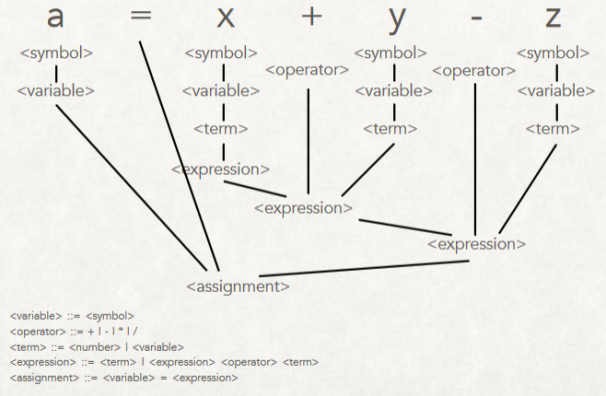
- <variable> came from a token “x” and its type was integer

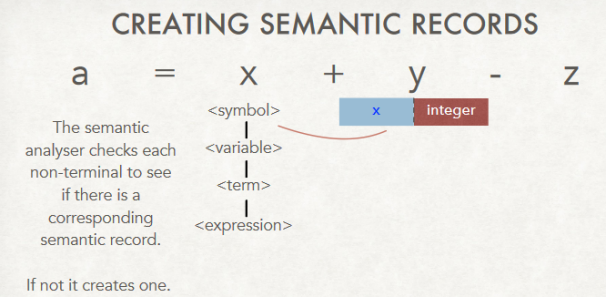
Code Generation

- At the same time as creating sematic records, we can use the information in the parse tree to also produce assembly code
  - Translate parse tree nodes into aseembly code
  - So semantic analysis is simultaneuous with code generation
  - Not all parts of the parse tree produce code (some parts are purely renaming and nothing extra is added)

Semantic records in parse tree

- Usually variables in the RHS of an assignment statement alreday have semantic records
- In expressions we have to check the types of the symbols being combined with operators to ensure they make sense





GENERATING CODE

a = x + y - z

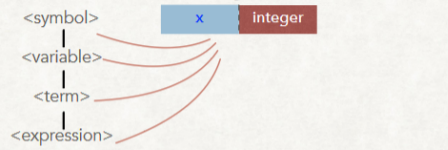
At the same time the code generator can produce the matching assembly code

```
x:   .DATA 0
```

The 0 is just a temporary value

CREATING SEMANTIC RECORDS

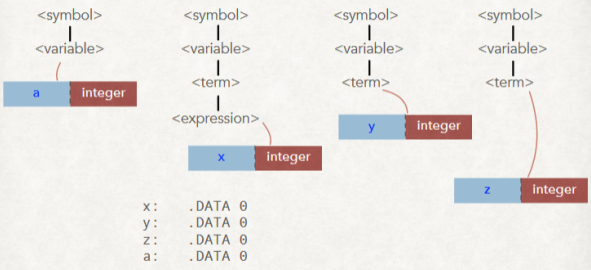
a = x + y - z



The remaining nodes as we move down the branches in this case do not need to create a new semantic record and don't generate any more code.

THEN WE CAN DO THE SAME THING

a = x + y - z

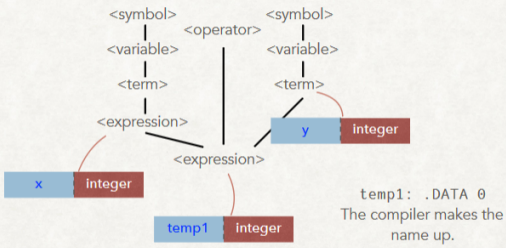



```
x:   .DATA 0
y:   .DATA 0
z:   .DATA 0
a:   .DATA 0
```

Generated code

WE NEED A NEW SEMANTIC RECORD FOR THE RESULT

a = x + y - z

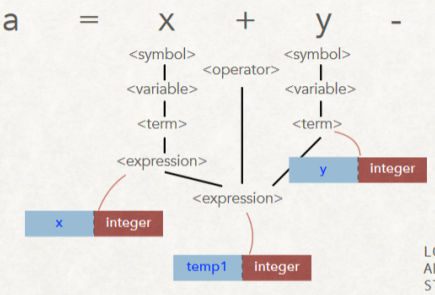



```
temp1: .DATA 0
```

The compiler makes the name up.

This is also the stage where the **meaning is checked**. In our case since we can add two integers together the semantics are correct so far.

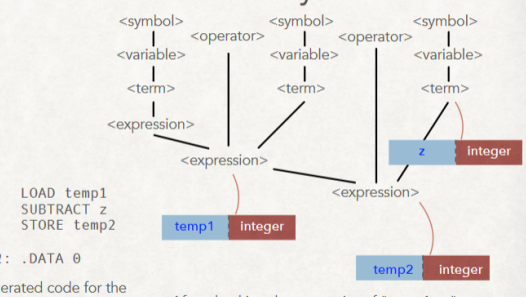
WE NEED A NEW SEMANTIC RECORD FOR THE RESULT




```
LOAD x
ADD y
STORE temp1
…
temp1: .DATA 0
```

The generated code for this expression.

AND AGAIN

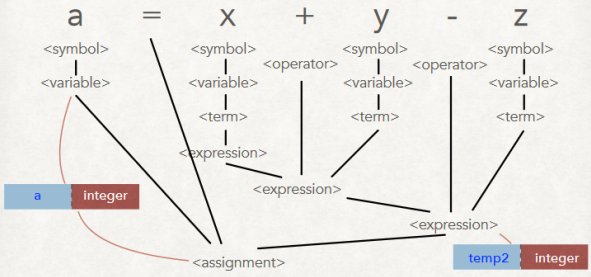



```
LOAD temp1
SUBTRACT z
STORE temp2
…
temp2: .DATA 0
```

The generated code for the second expression.

After checking the semantics of "temp1 - z".

AND FINALLY THE ASSIGNMENT



After checking the semantics of "a = temp2".

```
LOAD temp2
STORE a
```

The generated code for the assignment

THE COMPLETED ANALYSIS AND CODE

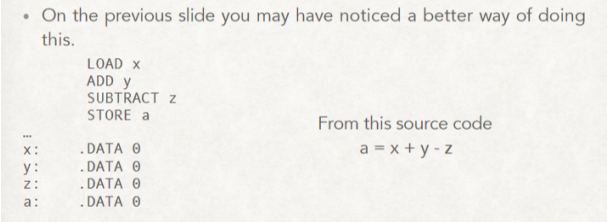
- The semantic analysis didn't find any errors and generated the following code.

```
        LOAD x
        ADD y
        STORE temp1
        LOAD temp1
        SUBTRACT z
        STORE temp2
        LOAD temp2
        STORE a
…
x:      .DATA 0
y:      .DATA 0
z:      .DATA 0
a:      .DATA 0
temp1:  .DATA 0
temp2:  .DATA 0
```

From this source code

a = x + y - z

Optimization



This is a very simple example of code optimization (we saved 4 instructions and 2 data values)
The compiler tries to improve the time or space efficiency of the generated machine code

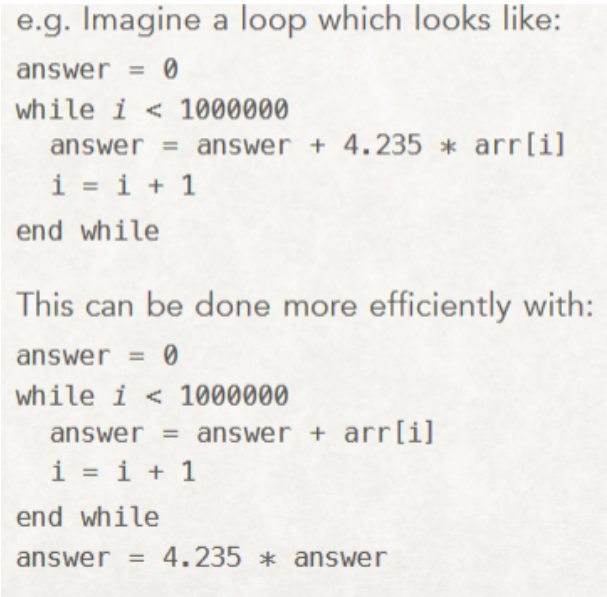
Local Optimization

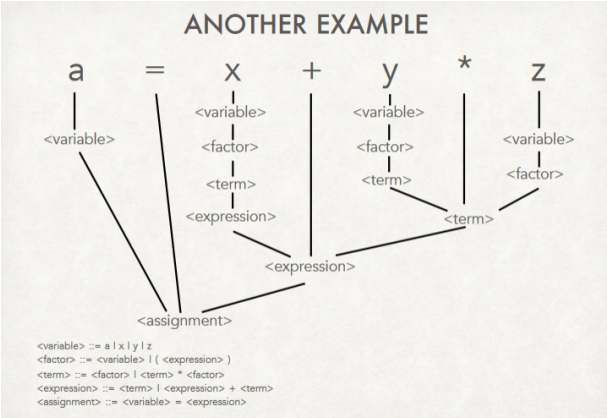
The previous example was local optimization. The compiler looks at a small number of instructions and **removed redundancies**. (e.g. remove immediate LOAD after STORE)

- Others include: **constant evaluation** - computer arithmetic expressions at compile time if possible
  - **Strength reduction** - uses faster arithmetic alternatives (2 * 2 === 2 + 2)

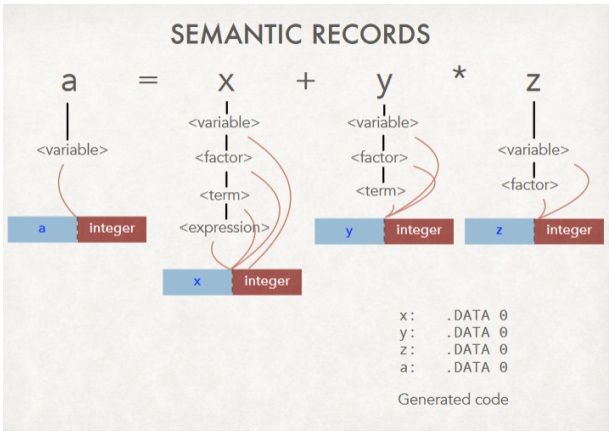
Global Optimization

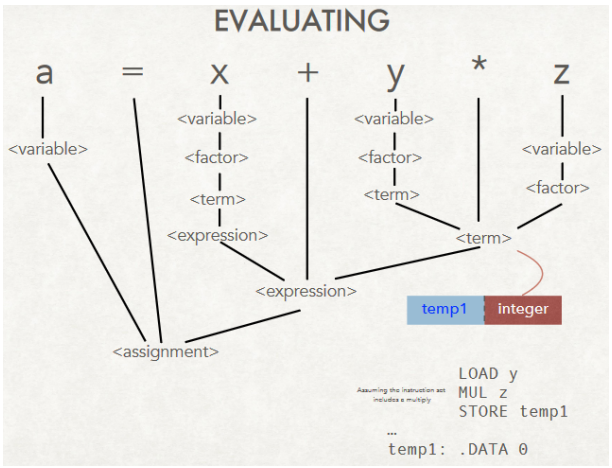
Much harder problem, in particular when the code executed inside loops

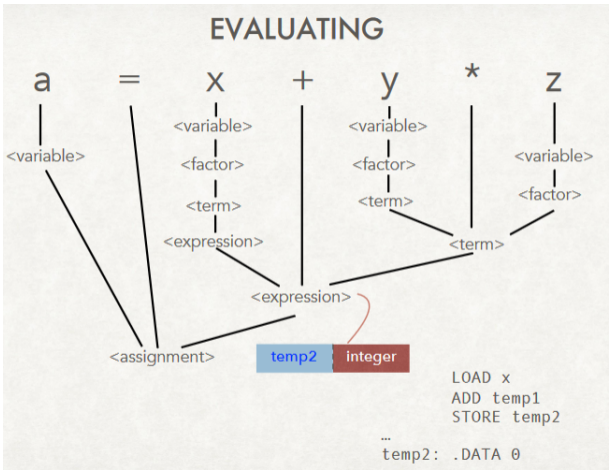


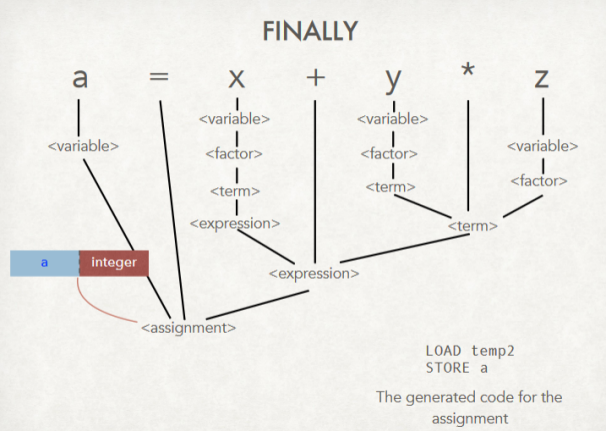


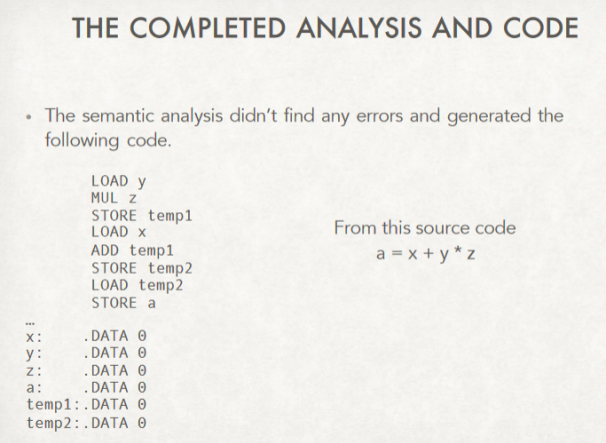
,


SEMANTIC RECORDS
a = x + y * z
<variable>
<factor>
<term>
<expression>
a integer
x integer
y integer
z integer
x: .DATA 0
y: .DATA 0
z: .DATA 0
a: .DATA 0
Generated code


EVALUATING
a = x + y * z
<variable>
<factor>
<term>
<expression>
<assignment>
temp1 integer
Assuming the instruction set includes a multiply
LOAD y
MUL z
STORE temp1
...
temp1: .DATA 0


EVALUATING
a = x + y * z
<variable>
<factor>
<term>
<expression>
<assignment>
temp2 integer
LOAD x
ADD temp1
STORE temp2
...
temp2: .DATA 0

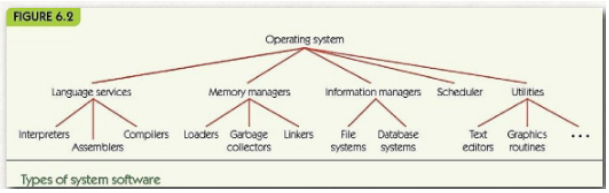




Operating Systems

An operating system is the most important collection of system software
It manages and controls the computer and provides the interface to you so you can control the computer
Some typical duties:

- Load and run programs
- Keep track of memory, allocating it when required
- Save information in a file or directory
- Deal with files, list, copy, move, delete
- Establish network connections
- Let the user set or change password
- Accept commands from users and programs



Main Components

Process manager: controls programs as they run
Memory manager: controls memory allocations
File manager: controls access to files and directories
Device manager: controls other devices (e.g. keyboard, screen)

All have to be **protected** with a security system

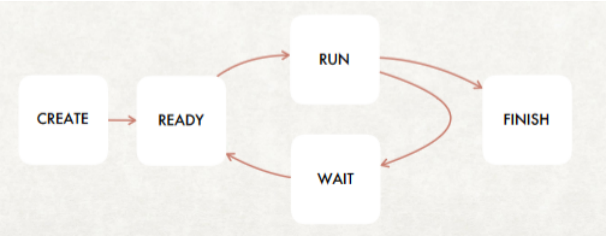
Process Manager
A process is a program which is running or part way through running
The process manager has to **keep track of all** the processes and the **resources** associated with them
Most system processes (and their users) are not allowed to access all resources

Most laptops have 4 to 8 cores (or processors) meaning they can really only run 4 to 8 processes at once.

- Most processes are not running
- A process starts when a person tells the OS to run a program (or another process)
- Most programs soon need data when they start running (from user, from file, from network connection etc)
- Until the data arrives, it has nothing to do (doesn't use core)

When a process needs to wait for input, it stops. The PM immediately selectes a process to replace it from the processes which don't need to wait (ready).

Scheduler
Chooses which process should be running next. A process is "read to run" when they have all the resouces they need to run
When a running process waits or finishes, the scheduler chooses the process which will run next
We have a **process staging diagram**:



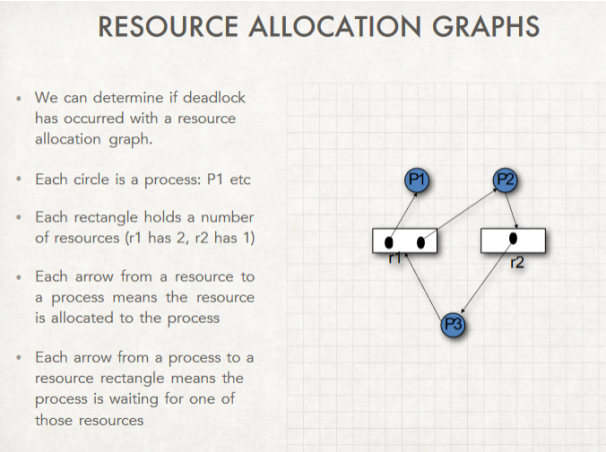
How the scheduler chooses the next process is called the **scheduling algorithm**

- There are many, most employ the concept of **process priority**. A process with better priority will be chosen over a process with worse priority
- The **First Come First Served (FCFC)** scheduling algorithm always chooses the one waiting the longest. Ready processes are kept in a queue and we take first off the queue.

- If a program doesn't need to wait but does a lot of calculation, it will hold up other processes which are ready to run. So, most scheduling algorithms **allocate a time-slice** to a process (e.g. 100ms).
  - If a process finishes its time-slice, it is moved to the back of the read queue and the next process is dispatched (made runnable)

Deadlock
Happens when a group of processes get stuck because the resources they need are held by other processes in the group (e.g. Process A has Resource X and is waiting for Resource Y and Process B

has Resource Y and is waiting for Resource X).
Both processes will never leave the waiting state.



A cycle in the graph means there is a deadlock.

Memory Management Unit (MMU)
Multitasking allows more than one program to run concurrently resulting in these programs being **loading into the memory simultaneously**
Requirements:

- More memory
- Memory spaces of the different processes need to be kept seperate
- Need to protect the memory of the OS from being modified by a **user** process

The hardware provides a system called the MNU which provide all these capabilities
The MMU allows each program its own memory area (**address space**) starting at 0 and going up to a max
Actually, the MMU is converting the addresses in the process into the real RAM addresses
The OS memory manager determines which sections (**pages**) of each program get stored where in the real RAM

- Works in conjuction with the MMU hardware
- Keeps track of available free memory, what memory has been allocated to what process, make sure memory is returned when a process no longer needs it, make sure the rules about what each process is allowewd to do with memory are observed (e.g. how much memory, access to which memory, read or write)

On many computers, we can use more memory than we have RAM for. The OSMM can allow this by using storage as a temporary memory location

- This is obviously slow but as long as it doesn't happen to often its very useful

Proccess which are witaing for resources for a long time might have all of their memory moved to a disk drive, thus freeing up RAM for processes which are still running
We talk about memory being **swapped out** and **swapped in**

The **file system** is the part of the OS which looks after storing and retreiving files from storage devices

- Keeps track of free space, keeps track of which parts of the disk are allocated to which file, controls access to files

The file system needs to store information about each file:

- File name

- File size
- Time it was most recently modified
- Owner of the file
- Where the data is stored on the disk
- Who is allowed to do what to this file

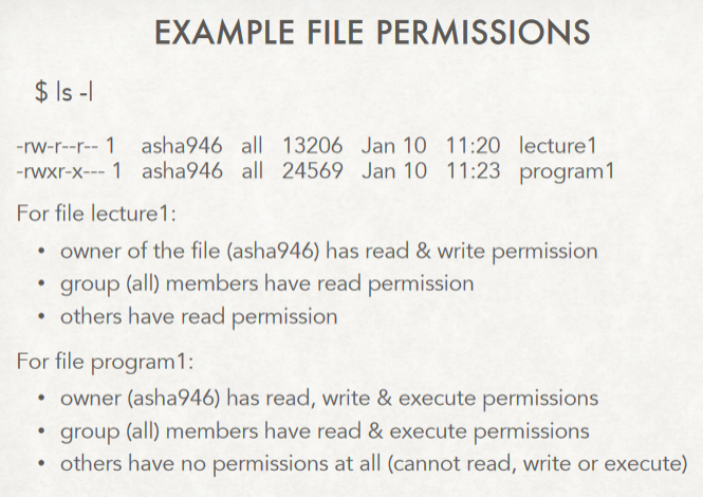
All of this information must be stored on the disk too. This is known as meta-data (data *about* the files)

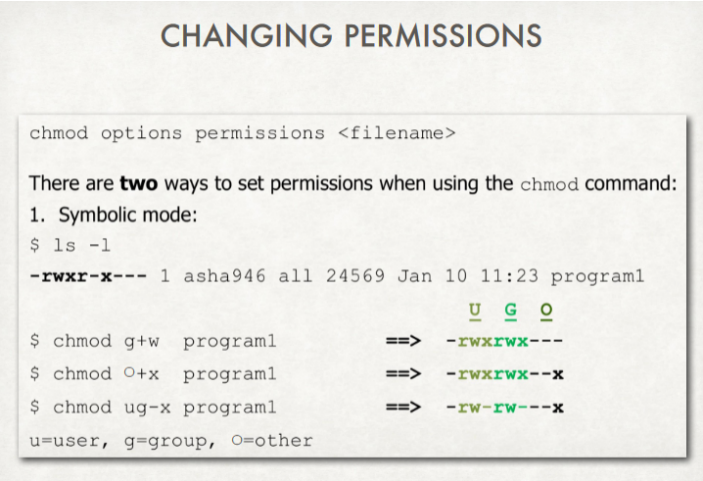
From the OS POV, only **authenticated** users should be allowed access to the system
- Authority to access the files is granted or denied by the OS based on the file permissions
- All files have an owner user, and resources such as disk space, printing, and even CPU time can be given per-user quotas
- Files also belonging to user *groups* or *roles* allow multiple users to read or write to the same files

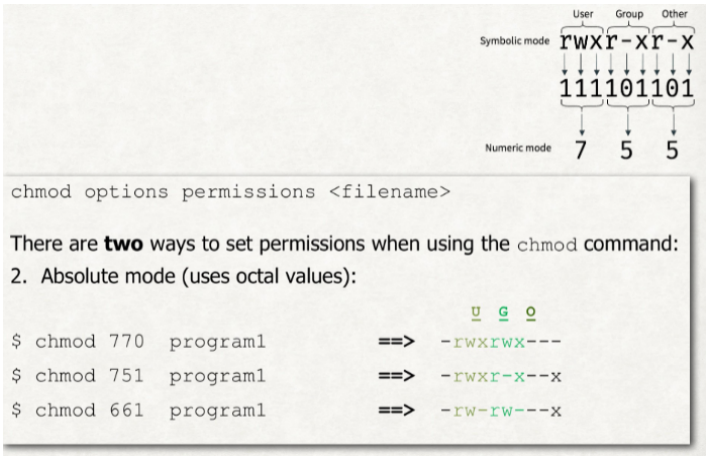
Unix File Permissions

Each file has an:
- Owner (normally the person who created the file) who can change privileges and do anything with it
- Group (collection of users)

Permissions are allocted to the ower, the group, and everyone else
- The permissions are read (r), write (w), and execute (x)







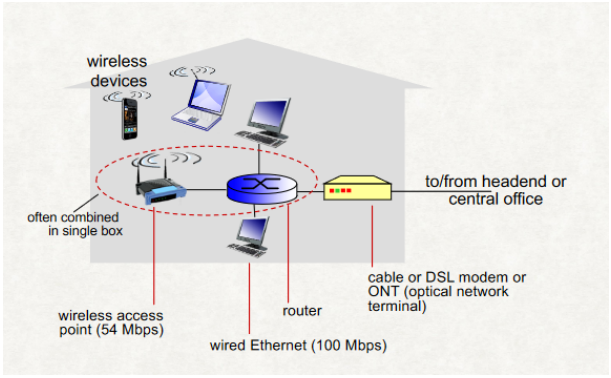
Computer Networks

A computer network is made up of computing devices (**nodes/hosts**) and interconnets for sharing information and resources. Networks may be wired (dial-up, broadband) or wireless (WLAN, WWAN, MAN, LAN, PAN)

Switched, dial-up lines are analog lines (voice oriented) and transmit digital data. The **modem** modulates carrier wave and the **bandwidth** is the capacity (rate at which info could be sent or received). Has speeds up to 56 Kbps

Broadband provides transmission rate >= 25 Mbps (typical home internet connections). They have asymmetric download/upload speeds.

- **Digital Subscriber Line (DSL)** uses phone lines, but sends signals on different frequencies than voice (download: 5-50 Mbps, upload: 1-5 Mbps)
- **Cable Modem** uses cable TV lines (download: up to 100 Mbps, upload: 3-5 Mbps)



Ethernet (1970′s): dedicated coaxial cable, operates at 10 Mbps

Fast Ethernet (early 1990′s): dedicated lines (coaxial, fiber-optic, or twisted-pair copper wire), operates at 100 Mbps

Gigabit Ethernet Standard (late 1990′s): from *gigabit networking* research project, IEEE standard, operates at 1000 Mbps/1 Gbps

Wireless Data Communication allows network communication without the need for cables. Radio, microwave, or infrared signals to mobile computers. **Mobile computing** delivers data regardless of location

Bluetooth is low power, close range (30-50 feet), connects devices like wireless mice, cameras, video games, earphones etc. Implements PAN (personal area network)

Wireless Local Area Network (WLAN): computers transmit wirelessly to a base station (wireless

router, access point) which has a *wired connection*. Range of 150-300 feet and transmission rate of 10-50 Mbps (down)
- **Wi-Fi**: term for wireless network communication
- **Wi-Fi hotspot**: library, campus, coffee shop, etc

Metropolitan Area Network (MAN): build-out wireless network that covers blocks or cities
Wireless Wide Are Network (WWAN): computers transmit wirelessly to a **remote base station** which has a *wired connection*. Cellular technology involves antennas on towers miles apart. Signal may be blocked when indoors, errors with data transmission can slow performance, wireless signals are easy to intercept (security concern)
Local Area Network (LAN): wired connection, all computers, printers, servers are in close proximity. Privately owned and operated. **Network topology**: how computers are connected affects how they communicated. Can be made up of different types of systems and OS installs
Wide Area Network (WAN): wired connection, connected computers located at great distances.
Dedicated point-to-point lines (mesh network): computers connect to others on individual lines, **store-and-forward, packet-switched**: packets go from node to node until reaching their destination. Long messages are broken down into packets and sent individually to the network.
- Routing of packets is determined dynamically
- Had redundant paths, fault tolerence, responsive to traffic load
- Multiple packets - each packet takes a different route

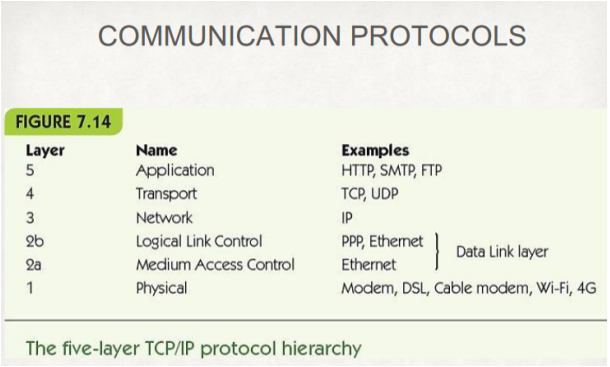
LAN Topology
- **Bus** topology: e.g. ethernet LAN's
  - Shared central cable
  - Devices take turn using the
  - Less cabling
  - Limited computers, little fault tolerence
- **Ring** topology:
  - Messages circulate until they reach the destination
  - Each installation, less cabling
- **Star** topology: e.g. high speed LAN's
  - All messages are sent to a central node, which routes their messages to their destinations (broadcast or unicast)
  - Easy to reconfigure

Ethernet LAN
- Ethernet LAN with *shared cable*. Uses bus technology and consists of a single cable over short distances and multiple cables over longer distances.
- **Repeater** amplifies and forwards the signal while the **bridge** routes messages only when necessary

- Ethernet LAN with *switch*. Uses bus technology and constsists of a shared cable inside the switch. Wiring contains switch and ports and ethernet jacks in rooms connect to the switch in the closet. Wireless base stations also connect to the switch in the closet.

Internet
Combination of LANs and WANs connected by **routers** that direct message traffic.
ISP provides access to the internet, DNS provide addressing information
- ISPs exist at multiple levels: local, regional, national, international (tier-1 network)

Communication Protocols
Protocol: standard set of rules for comminicating
Protocol hierarchy/protocol stack -TCP/IP: layers of protocols. Physical transmission to end application rules and standards
Internet Society: makes standards and promote research. Standards evolve over time
International agreements make the internet possible



Messaging originates with hosts. Hosts connected to local ISP. Local ISP has internal network that connects to other ISPs, eventually leading to a tier 1 ISP.

Physical Layer
Rules for exchanging binary data across a physical channel.
- How to know when a bit is present on the line
- How much time will the bit remain on the line
- Whether the bit is digital or analog in form
- What physical quantities represent 0 and 1
- Shapre of the connector between the computer and the trasmission line

Create an abstract “bit pipe” used by higher layers

Data Link Layer
Ensures reliable transmission of bits
Error detection and correction. Notices failures in the transmission and fix them
Framing determines which bits belong to one message
Two parts: Layer 2a - Medium Access Control, Layer 2b - Logical Link Control

Packets/Frame contains:
- Markers for start and end of packet (SOP and EOP)
- Sequence number for packet (e.g. 2 of 5)
- Packet data
- Error-checking bits
- Source and destination MAC addresses

Purpose of the data link layer is to create a *virtual error-free message pipe*
- Message go in one end and come out the other correct and in the right order

Medium Access Control Protocols
- Rules for communicating on shared lines - who has ownership
- *Contention-based* protocol - Ethernet
  - When a node wants to send a message
    - Listen to the line and wait until it is free
    - Begin transmitting as soon as it is free

    - If a collision results, wait a random amount of time
    - Repeat
- Advantage: **distributed**, no master bottleneck

Logical Link Control Protocols
- Rules for *detecting and correcting* errors
- **ARQ algorith** (automatic repeat request)
  - *Sender*
    - Transmit a packet and what for ACK or time out
    - If ACK received, go on to next packet
    - Otherwise, repeat on the current packet
  - *Receiver*
    - If no error, return **acknowledgement message (ACK)**
    - Otherwise, return nothing

Network Layer
- Sometimes called the *Internet Layer*
- Transmit messages across *multiple nodes* in a network and follows a good faith tranmission model
- Requirements:
  - Standard for *addressing* all network nodes
  - *Routing method* for finding a route from any node to any other node
- Provides end to end “network delivery service"
- Internet network layer: **IP (Internet Protocol)**

Addressing
- **Host name**: human-friendly name for node
- **IP address**: unique numerical address used by the computer
- **Domain Name Service (DNS)**: maps host names to IP addresses
  - Symbolic host name goes to a local DNS server
  - If it has no record, goes to remote servers until one has the host name and retrieves the IP address

Routing
- Picking a path through the network from source to destination
- Seeks the shortest/best path: fastest travel
- Massive network requires efficient path-seeking
- Networks are dynamic: nodes come online and go offline all the time - routing must adapt quickly

Transport Layer
- Application to applicatoin, reliable packet delivery
- **Port number**: unique identifier for a program/application
  - Application types have standard port numbers
    - Web server: port 80
    - Domain Name Service: port 42
    - SMTP, sending email: port 25
- **TCP (Transport Control Protocol)**
  - Ensures no errors (retransmit lost packets)
  - Establishes ordered delivery of packets
  - *Connection-oriented* - virtual, direct, quilaty connection between programs
  - Prioritze reliability over time (uses another version of ARQ algorith)
  - 20 bytes overhead in the header

    - e.g. web browsers, email, file transfer
- **UDP (User Datagram Protocol)**
    - *Connectionless*
    - Lacks error detection and flow control
    - Prioritize time over reliability (fast communication)
    - 8 bytes overhead in the header
    - e.g. video streaming, DNS, Voice over IP (VoIP)

Application Layer

- Handles formatted data transmitted between application programs

Acronym	Name	Application	Well-Known Port
HTTP	Hypertext Transfer Protocol	Accessing webpages	80
SMTP	Simple Mail Transfer Protocol	Sending email	25
POP3	Post Office Protocol	Receiving email	110
IMAP	Internet Message Access Protocol	Receiving email	143
FTP	File Transfer Protocol	Accessing remote files	21
TELNET	Terminal Emulation Protocol	Accessing remote terminals	23
DNS	Domain Name System	Translating symbolic host names to IP addresses	42

Some popular application protocols on the Internet

Hypertext Transfer Protocol (HTTP)

- Web page/service is identified by unique **URL (Uniform Resource Locator)**
    - protocol:
    - Multiple protocols: http, mailto, news, ftp
- Web browser uses TCP to send formatted messages to a web server, and vice versa - hence the connection is established
- Browser reads protocol (https), extracts host name (and requests IP address from the DNS server)
- Asks TCP to establish connection with port 80 of the host machine
- **Request message**: After connection is established, sends “Get” message with page information
- **Response message**: Server responds with message containing status code, page contents, and size and indicates the connection closes at the end of the message

Encapsulation

- Each layer adds information called a **header** to the data being passed to it
- This head contains information the layer needs to perform its job
- When the packet is received at its destination, the same process is repeated in reverse
- The packet is **de-encapsulated** and the headers stripped off when it is received by the intended target

Switch (layer 2)

- Ethernet packet is forwarded based on MAC address
- No involvement of the network layer
- IP packet remains completely untouched

Router (layer 3)

- The data link layer de-encapsulae the packet, passes the IP packet to network layer for routing
- IP packet is forwarded based on the IP address
- No involvment of the layers above the network layer
- The IP packet is encapsulated again to be transmitted by outgoing interface

MTU & MSS

MTU (maximum transmission unit) is the maximum amount of data that can be transmitted across the network. It does NOT include the ethernet headers required to transmit the packet on thernet. The common value of MTU on the internet is **1500 bytes**

MSS (maximum segment size) is only concerned with the size of the payload within each packet. It does NOT include the TCP header or the IP header.

Security

Threats: Threats online are often more dangerous than threats to physical items. Hackers prefer to attack easily accessible data
Defenses: Multiple deterrents to attacks

- **Authentication**: process that establishes the user's identity to the satisfaction of the system
- **Authorization**: governs what an authenticated user is allowed to do
- **Encryption**: purpose is to make information meaningless even if someone does manage to steal it

Malware: malicious software arriving from the network

- **Virus**: program embedded within another program or file, replicates itself and attacks other files (carried by infected host file)
- **Worm**: program that can send copies of itself to other nodes on the network (self-replicating)
- **Trojan Horse**: program that seems beneficial but hides malicious code within it.
  - Keystroke logger: records all keys types
  - Drive-by exploit/drive-by download: trojan horse downloaded by simply visiting and infected website
- **Denial-of-service (DoS) attack**: directs many computers to try access the same URL at the same time
  - Clogs the network, prevents legitimate access, and causes the server to crash
  - Distributed DoS uses thousands of computers
    - Uses a **zombie army (botnet)**: many innocent computers infected with malware
- **Phishing**: obtain sensitive information by impersonating legitimate sources
  - Many emails: just a few "bites" are enough

Types of Hackers

- *White hats*: security experts and those who work to help protect systems from attackers. Also called "ethical hackers"
- *Black hats*: individuals or groups who work toward geting around security to steal information, get money, or do other nefarious, immoral, and illegal acts
- *Grey hats*: do no have malicious intentions. Will find vulnerabilities like White Hats, but without permission to do so

Authentication: establishing identity. Requires username and password. OS secures the password file with a **hash function** (one-way encryption)
Password file security: no plaintext password stored
On login:

- Read username and password
- Look up entry for username in a password file
- Hash input password and compare

More secure method

- Keep password creation time
- Add creation time to password before hashing
- Identical passwords won't has to identical values

Other authentication methods:
- Answer personal information questions
- Biometric information (fingerprint or retinal scans)
- One-time password schemes
  - User enters ID and partial password
  - System or user device generates last half of the password
  - Last half of the password is good for only a few seconds
- **Dual authentication**: temporary code or password is sent to a trusted device

Salt
- Serves three purposes:
  - Prevents duplicate passwords from being visible in the password file
  - Greatly increases difficulty of offline *dictionary attacks*
  - Becomes nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them

Password Attacks
- Brute force (guess passwords)
  - Try common passwords
  - Try personal references
  - Try all possible passwords (computationally difficult)
- Steal password file and use **password-cracking software**
  - Tries words and word combinations, millions of password possibilites per second
- **Social engineering**: get a person to tell you their password

Authorization: set of permitted actions for each authorizated person

Operating system maintains **access control lists (or permissions)**
- Read access
- Write access
- Execute access
- Delete access

System admin or **superuser** has universal access and sets up authorization

Encryption
- One of the risks that data is exposed to in storage or in transit between hosts is eavesdropping
- **Encryption** and **decryption** convert from plaintext to cyphertext and back again
- All encryption algorithms use a **key** of some kind to convert the plain text to cypher text
- The intended recipient of the data uses the cyphertext and their key(s) to decrypt the encrypted data back to plaintext
- **Cryptography**: science of "secret writing"

Simple Encryption Algorithms
Caesar Cypher (or shift cypher)
- Map characters to others a fixed distance away in the alphabet
- **Stream cypher**: encode each character as it comes
- **Substitution cypher**: similar, but implement other mappings
- Pros: easy and fast, can do character by character.
- Cons: letter frequency, double letters, still pertain, make it easy to break (only 25 possible keys)

Weakness of the Monoalphabetic Replacement Cyphers
- The cypher text maintains the "fingerprint" of the language

- We can, therefore use a “Frequency Analysis” table to decypher the text

Block Cypher

- Block of plaintext encoded into a block of cyphertext
- Each plaintext character in the block contributes to multiple cypher text characters
- This *destroys the structure* of the plaintext making it hard to decrypt
- *Matrix-Based Block Cypher*
  - Groups characters into blocks of “n” characters long
  - Find invertible n by n matrix, M, and its inverse, M’, as keys. This property is waht allows M’ to **reverse the effect** of M.
  - Map characters to letts A -> 1, B -> 2, etc
  - Wrap values 26 and above back to 0: 26 -> 0, 27 -> 1, etc

Block cyphers produce **scattering (diffusion) of the plaintext** within the cypher text, which is advantageous.

Symettric encryption algorithm
- A secret key shared by the sender and the receiver
- Same key is used to encrypt and decrypt
- *Challenge*: to securely transmit the secret key

Asymetic encryption algorith (public key encryption)
- Uses two keys: public and private
- Use public key (generally known) to encrypt
- Use private key (known only to receiver) to decrypt

The three most important **symmetric block cyphers**
1. Data Encryption Standard (DES)
2. Triple DES (3DES)
   - Improves the security of DES. Requires 3 56-bit keys (which can be thought of as a 168-bit key length) and runs the DES algorithm three times; block size 64 bits
3. Advanced Encryption Standard (AES)
   - Uses a similar approach, (successive rounds of computations that mix up the data and the key). Longer keys (128, 192, or 256 bits), and larger block size (128 bits)

DES: Symmetric Encryption Algorithm

- Designed for digital data; plaintext (64 bit) is a binary string
  - Longer plaintext amounts are processed in 64-bit blocks
- Uses 64-bit binary key (56 bits actually used, remaining bits used for error checking)
  - From the original 56-bit key, 16 subkeys are generated, one of which is used for each round
- Sixteen rounds of the same series of manipulations
- Decryption uses the same algorithm; keys in reverse
- Fast and effective, but requires shared key
  - 56 bits is too small for modern technology
- Decryption uses the same algorithm but keys in reverse
  - Use K16 on the first iteration, K15 on the second, etc

In symmetic encryption systems, such as DES, the shared key must be protected from access by others. This strength of any symmetric cryptographic system rests with the key distribution

technique.
This is where *asymmetric* or *public-key cryptography* comes in handy

- Two most used public-key algorithms: RSA (Ron Rivest, Adi Shamir, and Len Adleman) and Diffie-Hellman key exchange

RSA Key Exchange Let P: plaintext, C: cyphertext, e: encryption key, d: decryption key
We can think of the encryption and decryption processes as being inverses of each other

Encrypt: $P^e \bmod n = C$
Decrypt: $C^d \bmod n = P$

Public key = (n, e), Private key = d. Here, $n = p \cdot q$; where p and q are two very large prime numbers.
In practice, n is as large as 2048 or 4096 bits
Thought n is part of the public key, it is computationally difficult to find two prime factorsr of n in finite time. This is the *strength of RSA*.

Steps

1. Pick 2 large prime numbers, p and q
2. Compute $n = p \cdot q$, and $m = (p-1) \cdot (q-1)$
3. Chose a large number e at random so that e and m are co-prime, where $1 < e < m$
4. This guarantees that there will be some multiple d of e, between 0 and m such that $(e \cdot d) \bmod m = 1$.

Digital Certificates
Challenge: there is still however, one problem in secret key distribution. How does the client know that the server's public key is actually the **server's** public key and not some *impostor's* public key?
This is where **digital certificates** come in
Certificates are issued by well-known *certificate authorities* (CA's), whose own certificates come pre-installed with most browsers, for example.

If the server presents the client with its certificate, then the client can use the certificate authority's public key to decrypt the server's public key, and therefore knows that the server's keys are trusted by the certificate issuer.

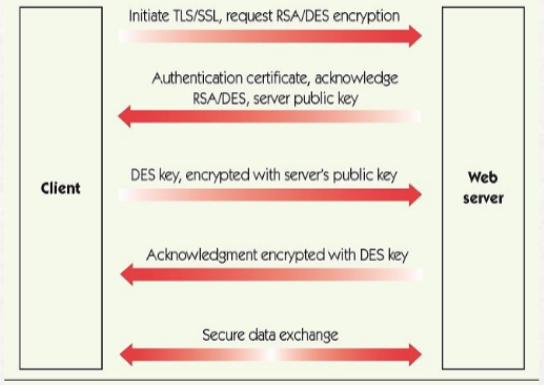
Web Transmission Security

- Ecommerce requires secure transmission of names, passowrds, and credit card numbers
- Web protocols: **SSL (Secure Sockets Layer)** and **TLS (Transport Layer Security)**
  - Client-server applications
  - Server provides certificate of authentication (digital certificate) and server's public key
  - *Digital Certificate*: issued by trusted third-party certificate authority
  - Client sends its DES key, encrypted using RSA
  - Data is send encrypted by the (now shared) DES key

The exchange of setup information between the client and the server, preparatory to exchaning data, is known as a *handshake*.

WEB TRANSMISSION SECURITY

FIGURE 8.4



A typical TLS/SSL session