Languages and Compilers

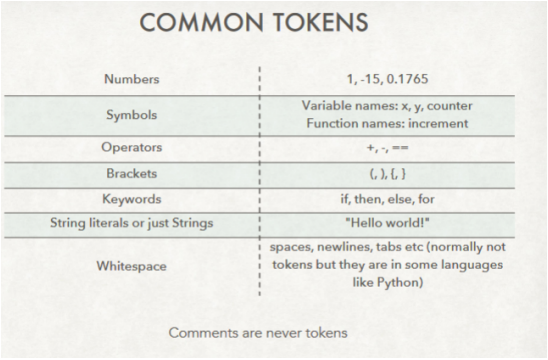
4 Phases of compilation Phase 1: Lexical Analysis
Groups characters into tokens
Phase 2: Parsing
Checks grammatical structure and builds internal representation of program
Phase 3: Semantic analysis and code generation
Analyses meaning; generates machine instructions
Phase 4: Code optimization
Improves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

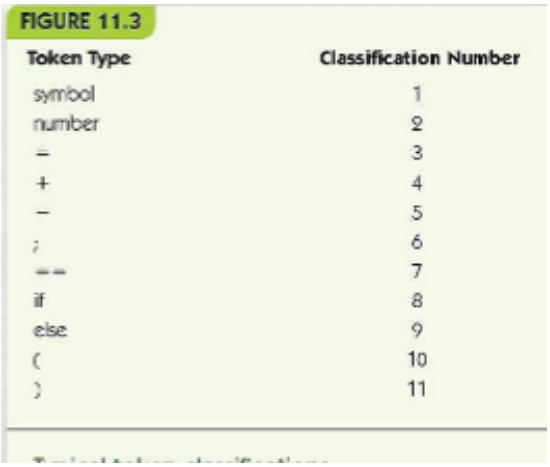
Phase 1

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:

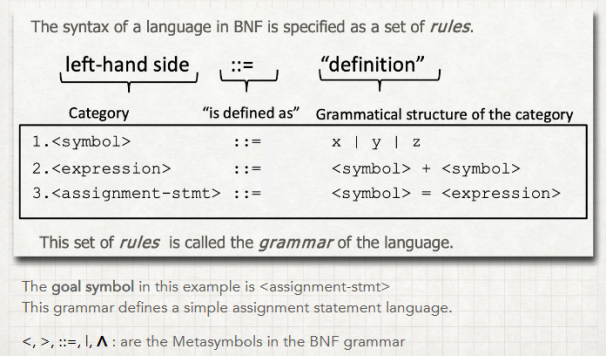
- Determines the grammatical structure
- Builds a parse tree
- Can tell us if the output from the lexer is a valid program (**syntactically correct**)

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

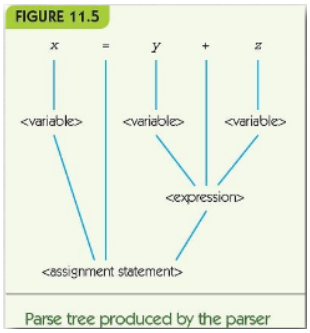


BNF has metasymbols (like language operators)

- <, >, 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

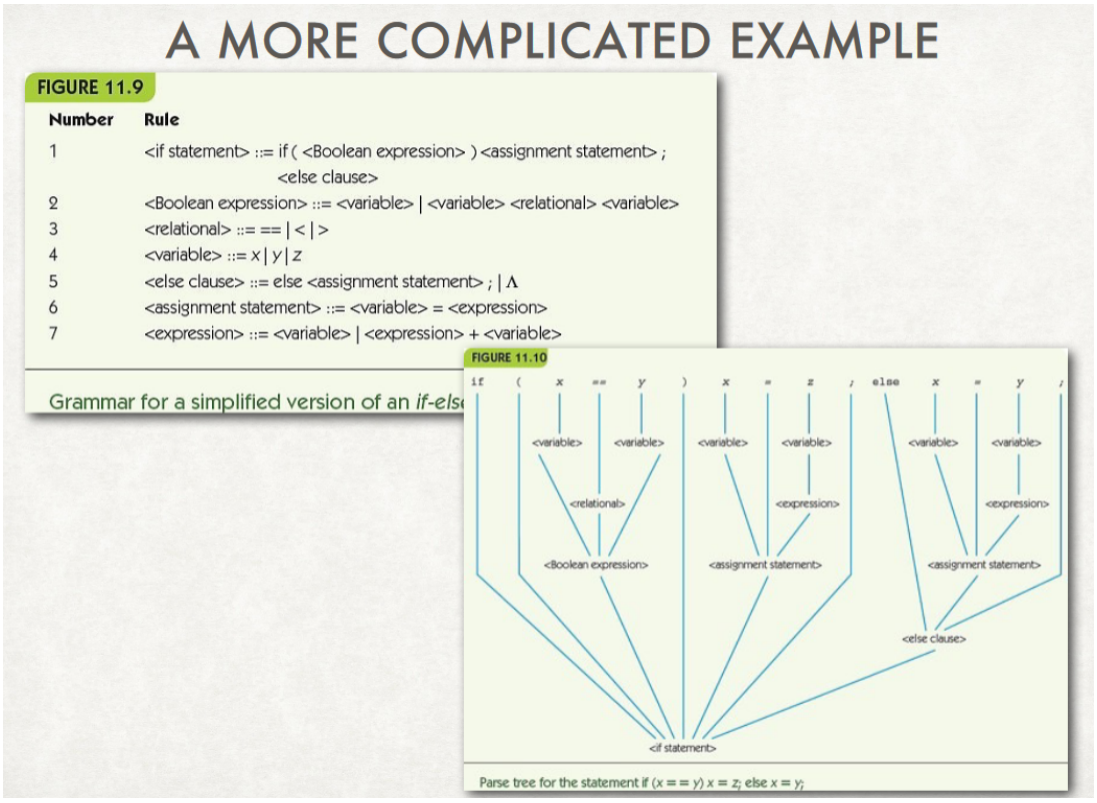
- Has its leaves and **tokens** returned by the lexer
- Uses the **rules of the grammar** to combine:
  - Leaves into branches,
  - Branches into "thicker" branches
- Each such combining step of leaves/branches into a single branch is called a **production**
- Eventually, there should be only one branch 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)

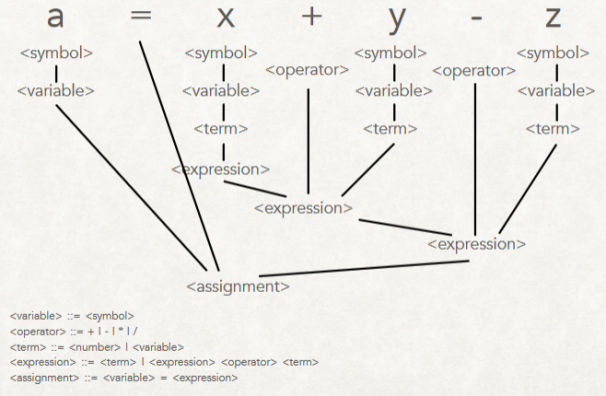
- <variable> came from a token “x” and its type was integer

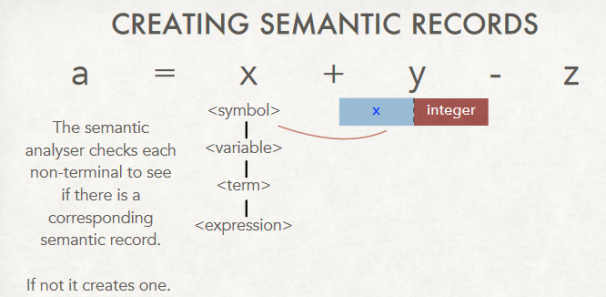
Code Generation

- 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

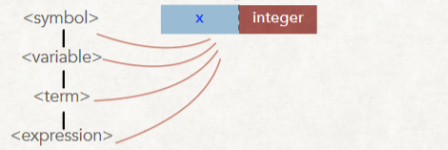
At the same time the code generator can produce the matching assembly code

```
x:   .DATA 0
```

The 0 is just a temporary value

CREATING SEMANTIC RECORDS

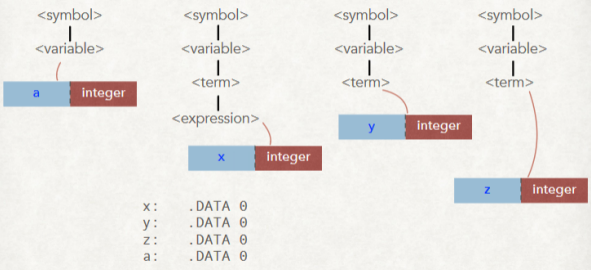
a = x + y - z



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

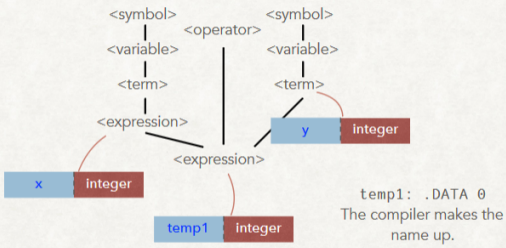



```
x:   .DATA 0
y:   .DATA 0
z:   .DATA 0
a:   .DATA 0
```

Generated code

WE NEED A NEW SEMANTIC RECORD FOR THE RESULT

a = x + y - z

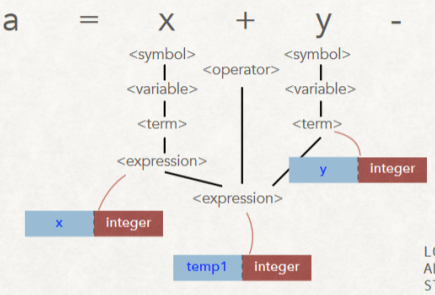



```
temp1: .DATA 0
```

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.

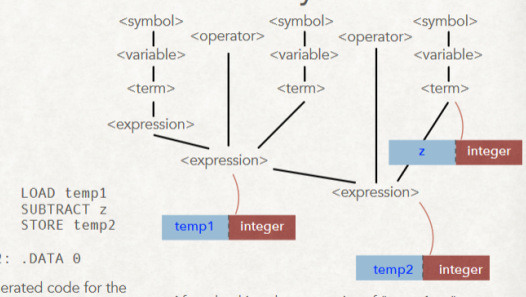
WE NEED A NEW SEMANTIC RECORD FOR THE RESULT




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

The generated code for this expression.

AND AGAIN

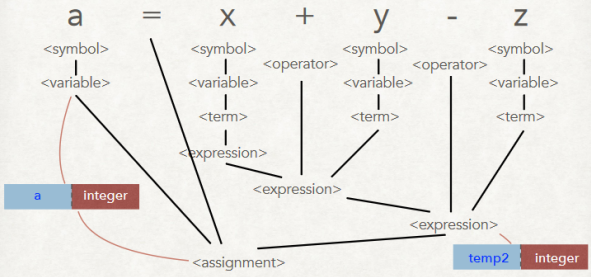



```
LOAD temp1
SUBTRACT z
STORE temp2
…
temp2: .DATA 0
```

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

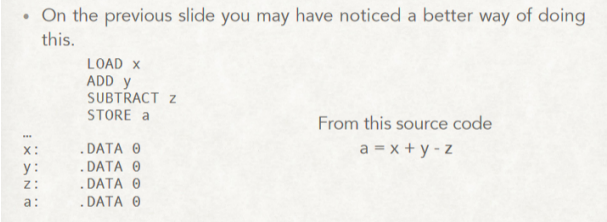
- The semantic analysis didn't find any errors and generated the following code.

```
        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

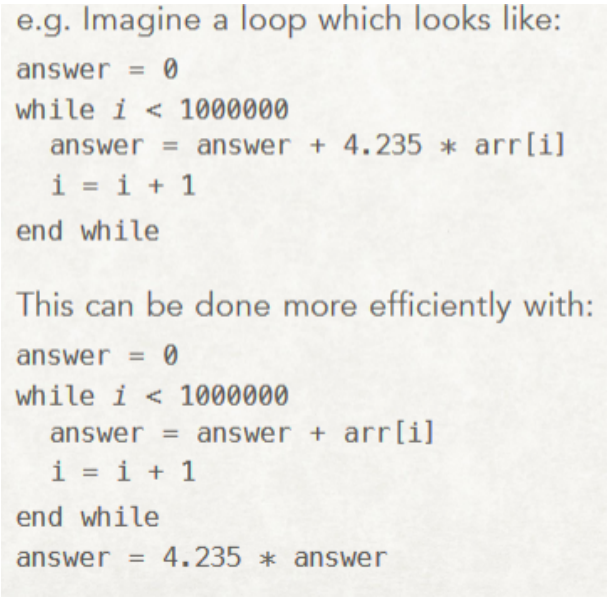
Local Optimization

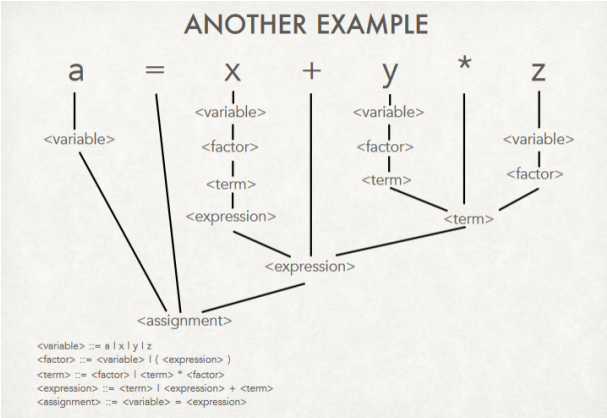
The previous example was local optimization. The compiler looks at a small number of instructions and **removed redundancies**. (e.g. remove immediate LOAD after STORE)

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

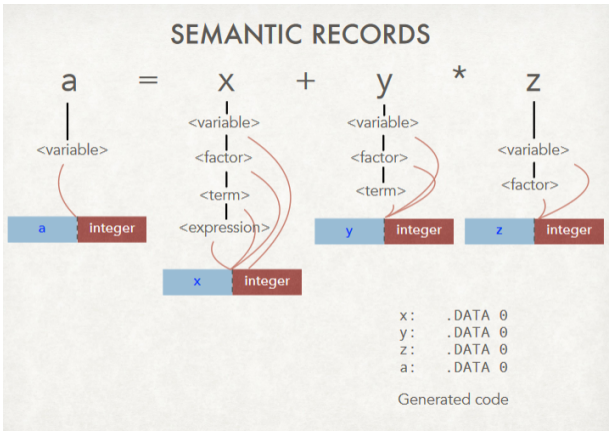
Global Optimization

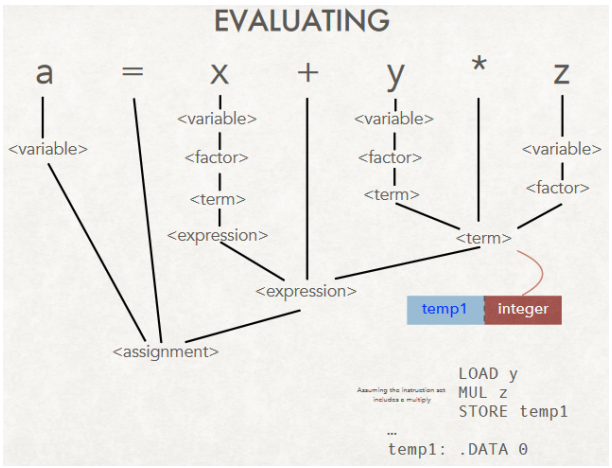
Much harder problem, in particular when the code executed inside loops

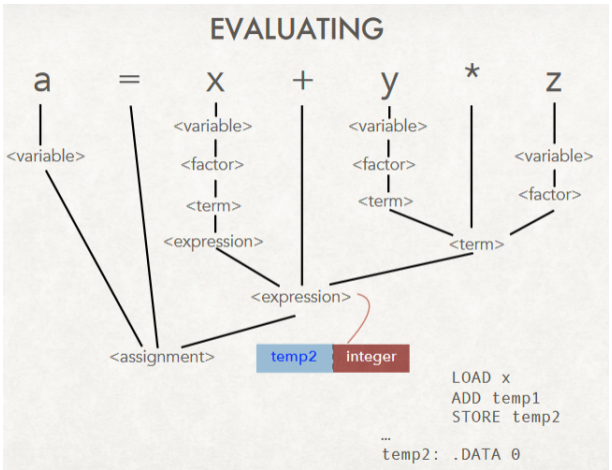


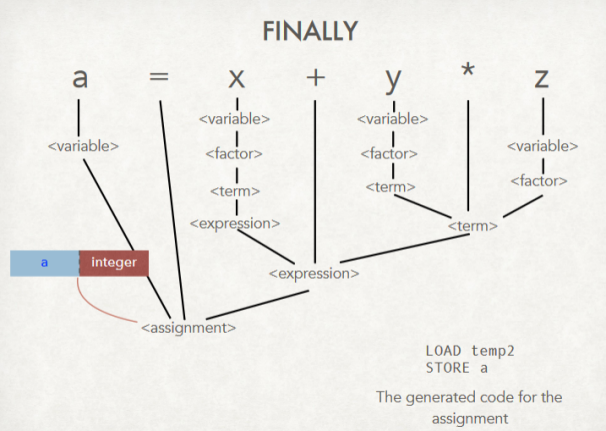


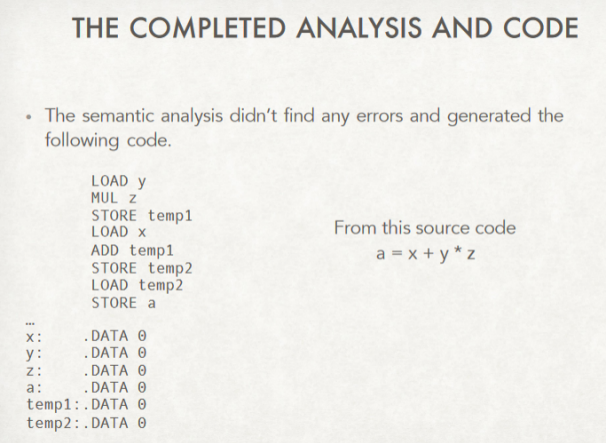
,


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

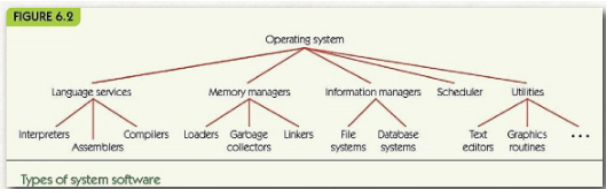




Operating Systems

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

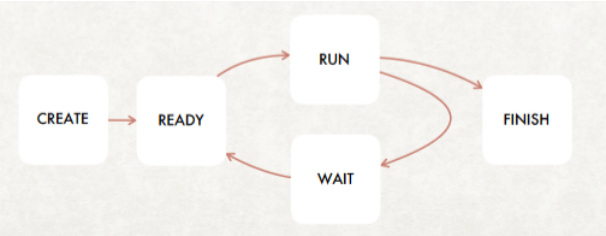
Process Manager
A process is a program which is running or part way through running
The process manager has to **keep track of all** the processes and the **resources** associated with them
Most system processes (and their users) are not allowed to access all resources

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**:



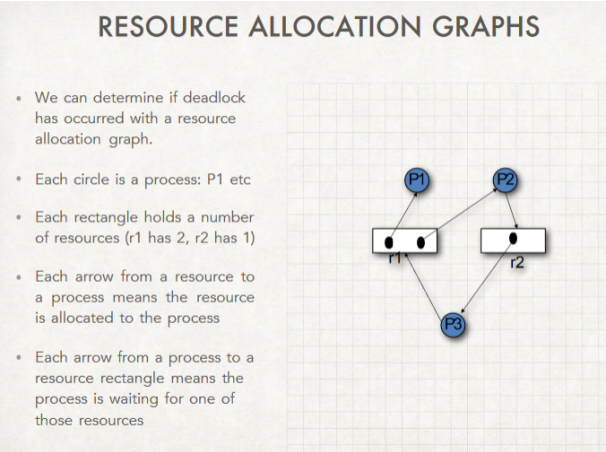
How the scheduler chooses the next process is called the **scheduling algorithm**

- 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

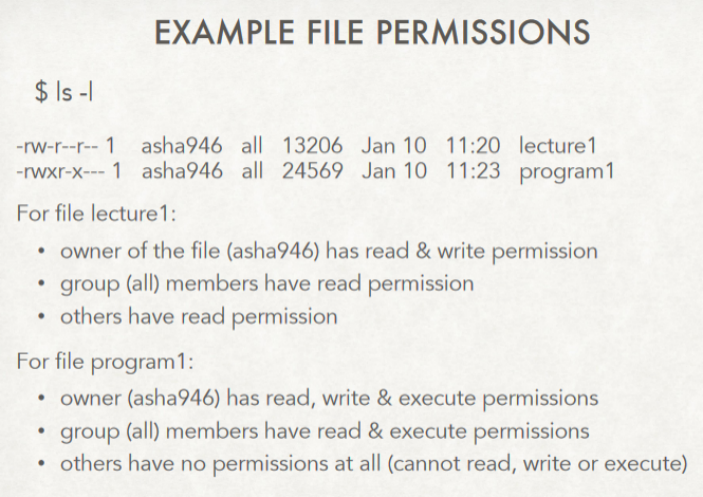
All of this information must be stored on the disk too. This is known as meta-data (data *about* the files)

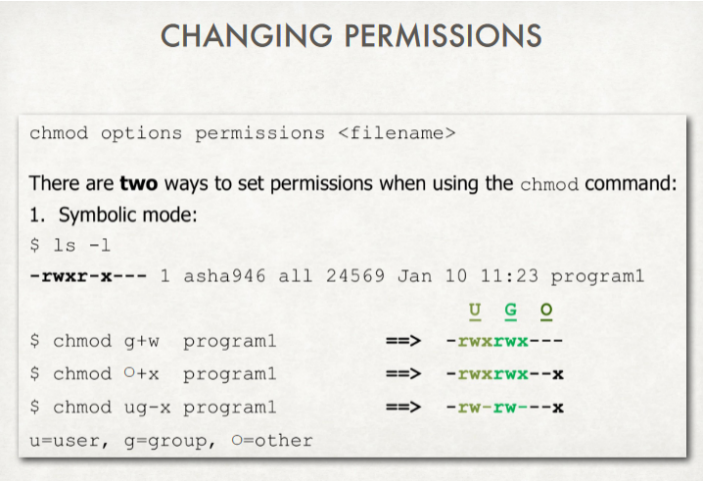
From the OS POV, only **authenticated** users should be allowed access to the system
- Authority to access the files is granted or denied by the OS based on the file permissions
- All files have an owner user, and resources such as disk space, printing, and even CPU time can be given per-user quotas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

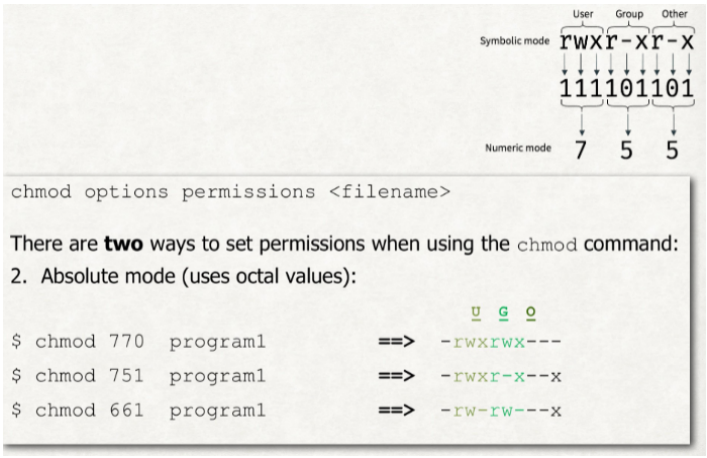
Unix File Permissions

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)







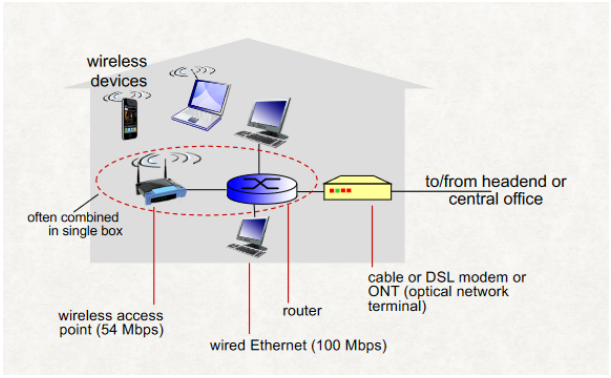
Computer Networks

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)