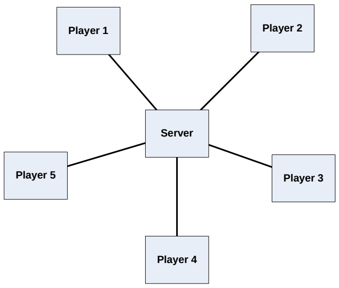
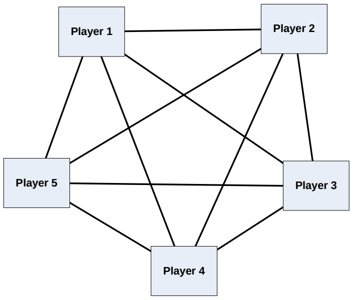
Network Games Part II

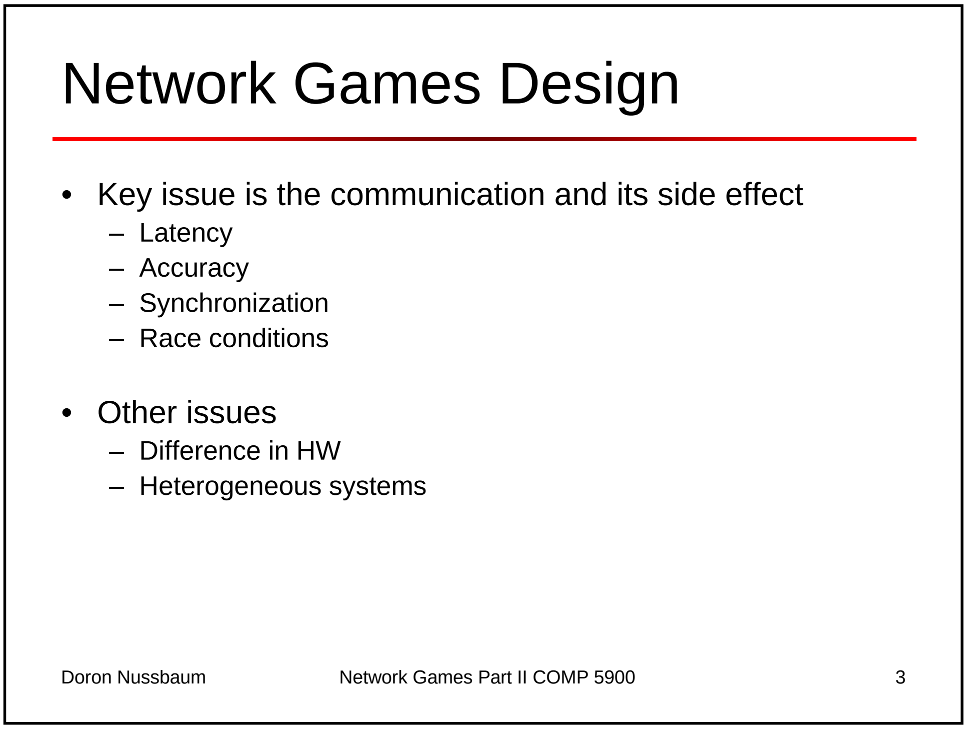
Architecture of Network Game

Client Server



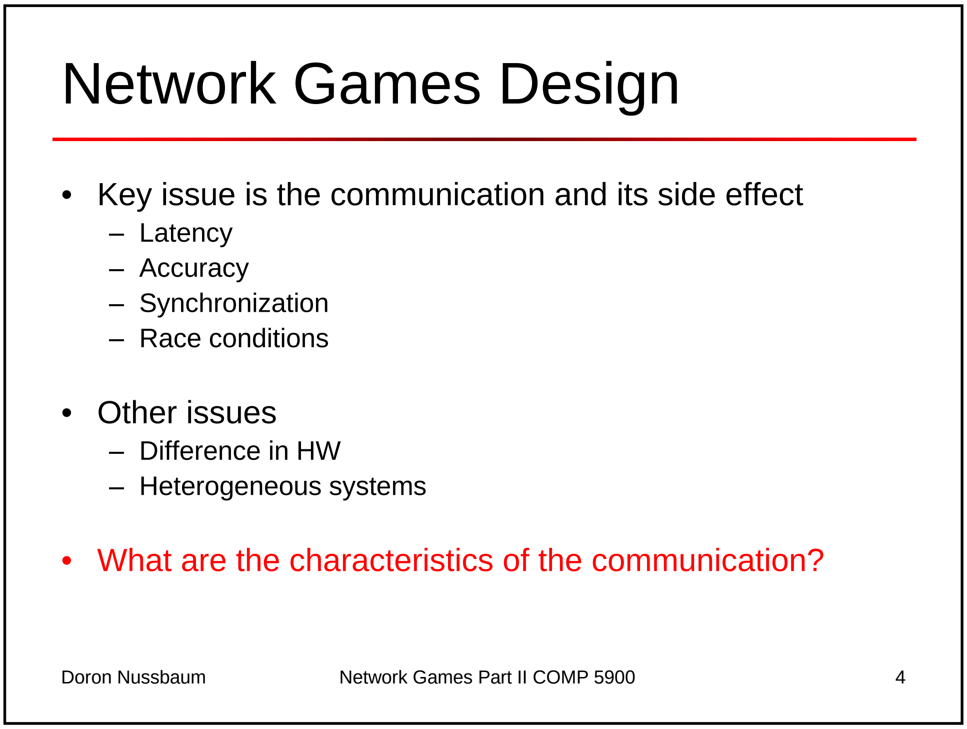
Peer to Peer





Network Games Design

- Key issue is the communication and its side effect
  - Latency
  - Accuracy
  - Synchronization
  - Race conditions
- Other issues
  - Difference in HW
  - Heterogeneous systems



Network Games Design

- Key issue is the communication and its side effect
  - Latency
  - Accuracy
  - Synchronization
  - Race conditions
- Other issues
  - Difference in HW
  - Heterogeneous systems
- What are the characteristics of the communication?

Comm. Behaviour of Architecture

Peer to Peer

- O(n^2) messages

Client Server

- O(n)-O(n^2) messages depending on the implementation
- O(n) implementation imposes tradeoffs

Comm. Behaviour of Architecture

Peer to Peer

- O(n^2) messages

Client Server

- O(n)-O(n^2) messages depending on the implementation
- O(n) implementation imposes tradeoffs

Which one is better?

Which one would have more latency?

What Can the Network Support?

d - message size (packet size)

f - frequency (in Hz)

n - number of of connections

C - network capacity

$$d * f * n \le C$$

d - 1500B=12,000b

f - 5Hz

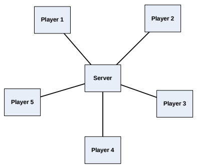
n - ?

C - 100MB=10^8

$$12{,}000 * 5 * n \le 10^8$$

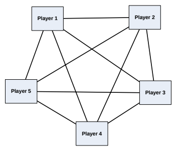
$$n = \frac{10^8}{5*12{,}000} = 1{,}666 \text{ connections}$$

Architecture of Network Game



Client Server

~1600 players?



Peer to Peer

57 players

Message sending pattern

- How many messages are sent and received?
- How much work does each processor handle?
- What are the tradeoffs?

Client Server

At each cycle

For each received message
sends *n* messages

Or

Receive *n* messages (one from each)
send *n* large messages

Pee to Peer

At each cycle

Receive *n* messages
Send *n* message

Sending a message

$$T_m = T_{preprocess} + T_{transmit} + T_{postprocess}$$

T_m - time of message cycle

$T_{preprocess}$ - time to prepare the message to be sent

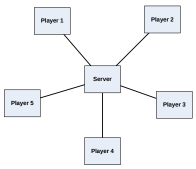
$T_{transmit}$ - time of transmit the message

$T_{postprocess}$ - time to process the message by receiever

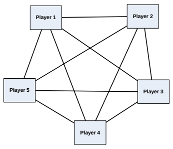
In our case we are interested in the sending part

$$T_m = T_{preprocess} + T_{transmit}$$

What does it mean?

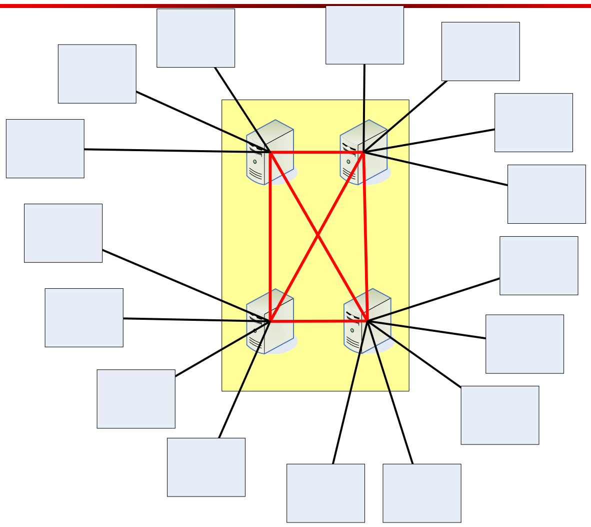


Client Server



Peer to Peer

Server Farm Communication

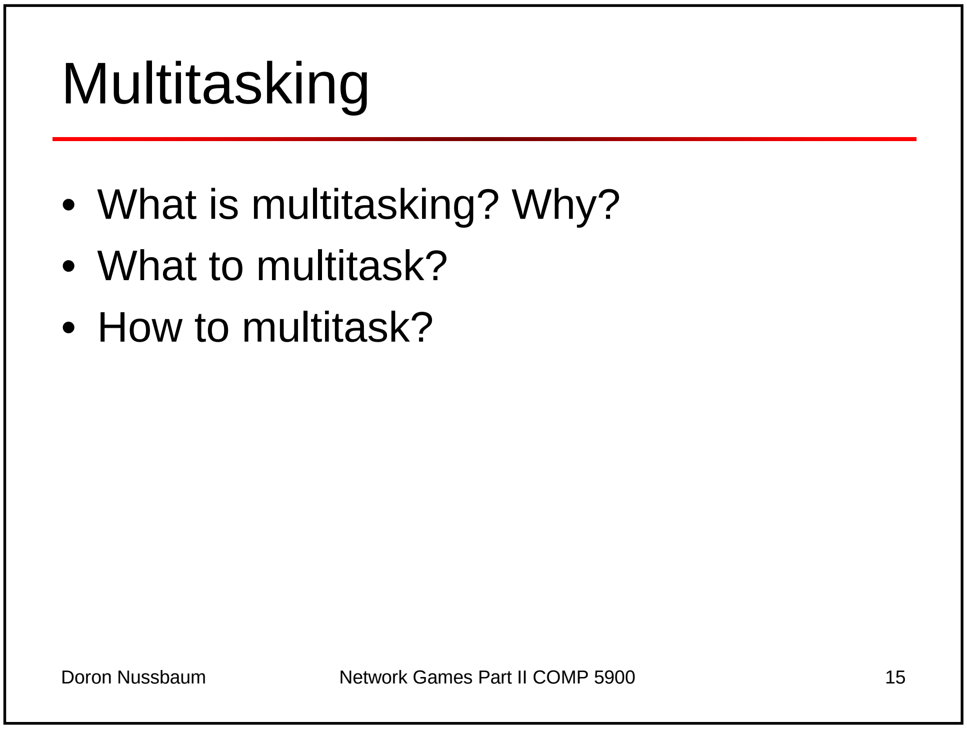


Server Farm

- A hierarchical solution
  - One network at the servers level
  - One network at the server/player level
- Commnication can be overlapped

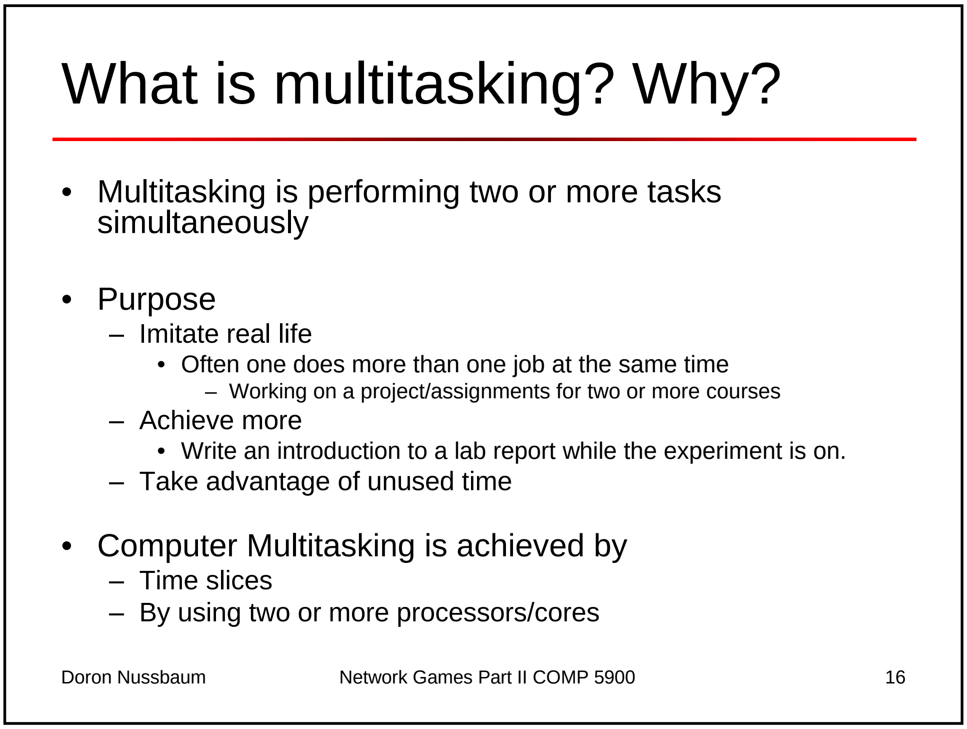
Play

P



Multitasking

- What is multitasking? Why?
- What to multitask?
- How to multitask?



What is multitasking? Why?

- Multitasking is performing two or more tasks simultaneously

- Purpose
  - Imitate real life
    - Often one does more than one job at the same time
      - Working on a project/assignments for two or more courses
  - Achieve more
    - Write an introduction to a lab report while the experiment is on.
  - Take advantage of unused time

- Computer Multitasking is achieved by
  - Time slices
  - By using two or more processors/cores

How is multitasking achieved?

- Providing capabilities to
  - Start multiple program at the same time
  - Ability to switch between the programs
  - If needed, enabling programs to interact with each other

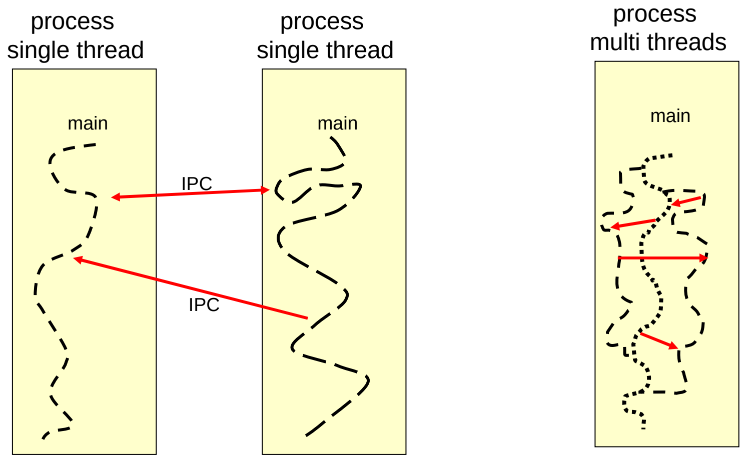
- Multitasking OS
  - An OS that enables multitasking on HW with one or more processors/cores

Processes and Threads

- Process
  - A process is a self contained program that is allocated system resources for its operation.
  - Consume resources - the system manage processors as a whole (e.g., memory, program counter and file descriptors).
  - Relatively “expensive”

- Threads
  - A thread is a distinct execution path within the processor
  - Imitation of the OS multitasking within a processor

How is multitasking achieved?



process
single thread

process
single thread

process
multi threads

main

main

main

IPC

IPC

How to multitask?

- Issues – coordination
  - Between tasks (race condition)
  - Accessing data

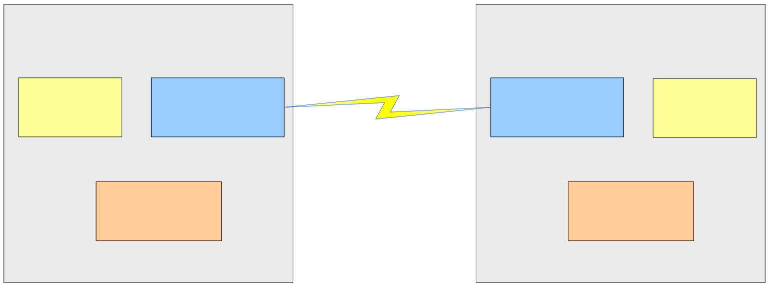
Task: Send/receive Messages

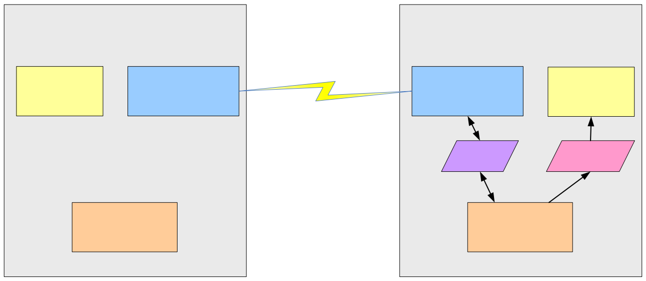
- Purpose
  - – Send messages to all players
  - – Receive messages from other players
- What needs to be done?
  - – Service many players (nodes)
  - – Interact with internal components

Servicing many clients

- Must assess the cohesion of tasks
  - – None!!
  - – Wait → request → response
- The show must go on!!!
  - – One cannot block and wait?

What must to be done?





Rendering Task

- Externally
  - – Use IPC – sockets
- Internally
  - – User threads

Non Blocking Sockets

- Instruct sockets not to block

```
u_long iMode;

iMode = 1; // non blocking is enabled
ioctlsocket(sockfd, FIONBIO, &iMode);
```

Non Blocking Sockets – collect sockets

Listen to connect messages

Add socket to a list of sockets to service

```
fd_set master_fds;
```

```
typedef struct fd_set {
   u_int fd_count;
   SOCKET fd_array[FD_SETSIZE];
 } fd_set;
```

```
clientAddLen = sizeof(clientAdd);
clientSock = accept(sockfd, (struct sockaddr *) &clientAdd, &clientAddLen)

if (clientSock == INVALID_SOCKET) {
    // error
    rc = GetLastError( );
    if (rc == WSAEWOULDBLOCK)  {
        // that's ok. There is nothing to accept
        Sleep (30);                          // sleep for a while if one can afford it
       } else {
          // error ….
      }
}

if (clientSock != INVALID_SOCKET) {
     if (FD_ISSET(clientSock, &master_fds) == 0) {
          // socket is not in the set
          FD_SET(clientSock, &master_fds);
     }
}
```

Non Blocking Sockets

Service requests

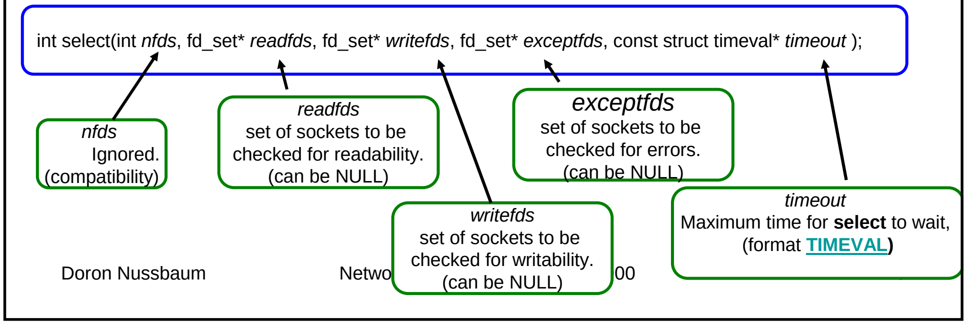
- Get message
- Process message
- If needed send reply on one or more

```
// master_fds contains all sockets

fd_set read_fds;

read_fds = master_fds;

rc = select(0,&read_fds,NULL,NULL,&timeout);
for (i = i < rc; i++) {
   // read information from the socket
```



int select(int *nfds*, fd_set* *readfds*, fd_set* *writefds*, fd_set* *exceptfds*, const struct timeval* *timeout*);

nfds
Ignored.
(compatibility)

readfds
set of sockets to be checked for readability.
(can be NULL)

writefds
set of sockets to be checked for writability.
(can be NULL)

exceptfds
set of sockets to be checked for errors.
(can be NULL)

timeout
Maximum time for **select** to wait,
(format **TIMEVAL)**

Sockets descriptors sets

Four macros

- **FD_CLR(*s*, **set*)** – removes the descriptor *s* from *set*.
- **FD_ISSET(*s*, **set*)** – nonzero if *s* is a member of the *set* else zero.
- **FD_SET(*s*, **set*)** – adds descriptor *s* to *set*.
- **FD_ZERO(**set*)** – Initializes the *set* to the null set.

- Note that FD_ISSET leaves only the sockets that are active in the set.
  - Use a master_list and copy it.
- One can change the number of sockets that are available by redefining FD_SETSIZE