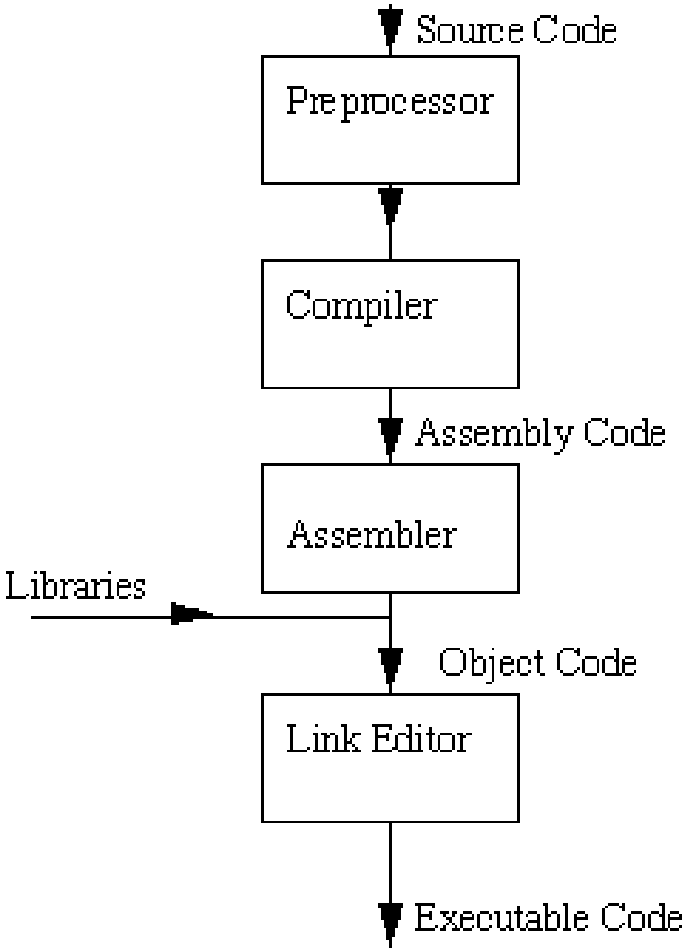
Intro:
C/C++ - Programming under Linux

Toolchain

- **Source Code Editor**
  - **Conventions: Extension .c or .cpp**
  - **E.g. vi myprog.c**
- **Compiler**
  - **Various compilers or integrated IDEs available**
  - **Linux Standard: GNU C/C++ compiler gcc, g++**
- **Debugging**
  - **Debugging Tools under Linux: gdb, ddd, …**
- **Excecuting the program**
  - **Conventions: no extension (.EXE)**
  - **./myprog**



gcc/g++

- **gcc/g++ are used for all steps from preprocessor to linker**
  - **Complex handling with countless options, flags and parameters**
  - **For details: man gcc, man g++**

- **gcc myprog.c**
  - **Creates an executable named a.out when no errors occur**
- **gcc -o myprog myprog.c**
  - **Creates an executable named myprog**
- **gcc -g -Wall -O –o myprog myprog.c**
  - **Enables all warnings, code optimization and symbols for debugging**
- **g++ -Wall -g -std=c++17 -o test test.cpp**
  - **Compile C++ with c++17 standard**
- **g++ Cheatsheet**
  - **https://bytes.usc.edu/cs104/wiki/gcc/**

gcc

- **Important options**

-c	Compile only without linking
-E	Preprocessor only
-S	Generate only assembly code
-o filename	name of executable
-g	Add symbols for debugging (debug build)
-O	Enable optimizations
-Dsymbol=value	#define symbol=value
-Wall	Show all warnings
-lbibliothek e.g. -lm	Link with library Link with math library
-I directory	Include directory in search path for header files
-L directory	Include directory in search path for libraries

make

- **Powerful tool to automate building software**
- **Reads instructions from a simple textfile (Makefile) how to compile and build a program and which dependencies exist**
- **Executes all steps for a build target automatically instead of manual steps from command line with lots of compiler/linker options**
- **Make checks dependencies and timestamps, which source files changed and compiles only not up-to-date files**

- **See also**
- **https://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap17-000.htm**

make

- **Syntax of Makefile**

```
# Comment
Target1: Dependencies
<tab>command 1
<tab>command 2
<tab>…
Target2: Dependencies
<tab>command 1
<tab>command 2
<tab>…
```

- **Invoke with:**
  - **make … builds first target in Makefile**
  - **make target … builds target**

make

- **Simple example**

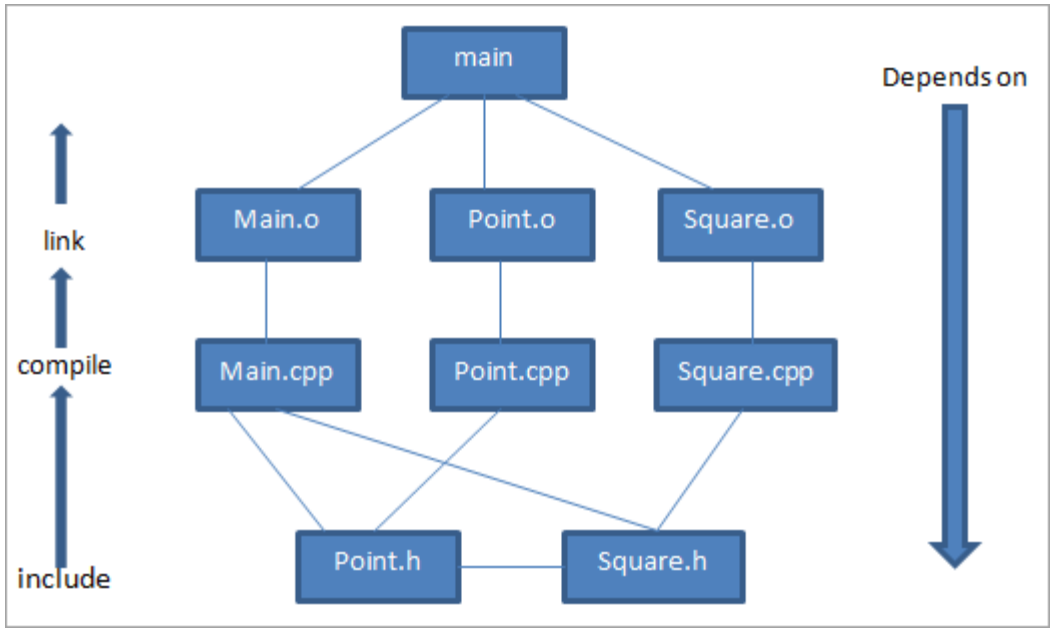
```
#Makefile für myprog.c
all: myprog

myprog: myprog.c
  gcc –g –Wall -0 –o myprog myprog.c

clean:
  rm –f myprog
```

- **make or make all builds program myprog**
  - **Target all depends on target myprog the executable**
  - **Target myprog depends on the source file myprog.c, so when the source is updated make will build the target and invoke the gcc command**
- **make clean deletes all generated files**
- **Convention: Targets all and clean should always be present and all should be first target!**

make

- **Example with variables**

```
# Makefile for project main
# ***********************************************
# Variables to control Makefile operation
CC = g++
CFLAGS = -Wall -g
# ***********************************************
all: main
main: Main.o Point.o Square.o
    $(CC) $(CFLAGS) -o main Main.o Point.o Square.o
Main.o: main.cpp Point.h Square.h
    $(CC) $(CFLAGS) -c Main.cpp
Point.o: Point.h
    $(CC) $(CFLAGS) -c Point.cpp
Square.o: Square.h Point.h
    $(CC) $(CFLAGS) -c Square.cpp
clean:
    rm -f main *.o
```



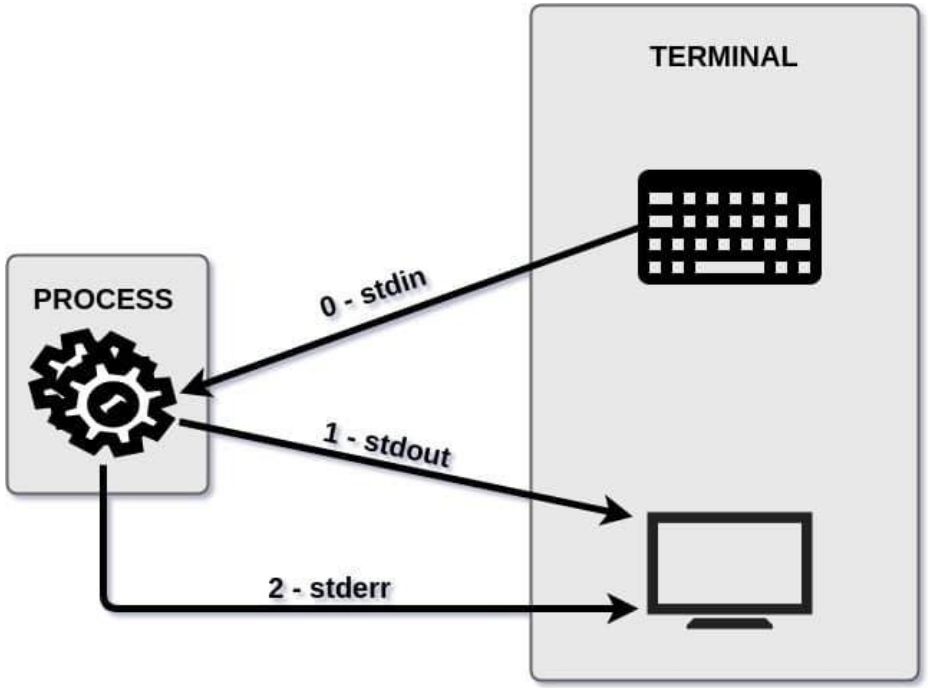
- **More examples**
  - https://makefiletutorial.com/

Standard Streams

- **Stdin (0), stdout (1) and stderr (2) are defined as filestreams in stdio.h**
  - **C++: cin, cout, cerr**

- **More info:**

https://www.putorius.net/linux-io-file-descriptors-and-redirection.html



General Conventions

- **Error Handling**
  - **Check return codes of functions**
  - **Write meaningful error-messages including the program name (stderr not stdout)**
    - C: fprintf(stderr,…) instead of printf
    - C++: cerr instead of cout
    - `fprintf(stderr,"%s: Error opening file %s\n",argv[0],filename);`
- **Compile with warnings enabled (-Wall)**
- **Release/Close your resources (avoid memory leaks)**
- **Don‘t use functions that crash in case of bad input (gets, scanf, atoi, atol) use safe versions instead (fgets, sscanf, strtol).**

Entry Point

- **int main(int argc, char* argv[])**
  - **defines the entry point to the application / process**
  - **return value represents an error code (EXIT_SUCCESS = 0, error > 0)**
  - **check return value in the shell using $?**
    - ./myprog
    - echo $?
  - **argc is the argument counter, always >=1**
  - **argv[0] .. argv[argc-1] is a string array containing the parameters**
    - separated by space
    - argv[0] always refers to the application name
  - **Examples**
    - ./myprog test argc=2
    - ./myprog -o test argc=3
    - ./myprog -o foo bar argc=4
    - ./myprog -ofoo bar argc=3

Argument parsing

- **Further args can be used as options (first) or arguments**
  - E.g. curl -X GET http://localhost:8080
  - Here -X is a short option and can also be used as --request (long option) before the URL argument
  - Multiple options can be used together like
    - **ls -ld which equals to ls -l -d**
  - Options can appear only once
  - arguments to options are possible (optional space) like
    - **gcc –o myprog myprog.c**
    - **gcc –omyprog myprog.c**

- **Use getopt function for simpler parsing**
  - getopt.h needs to be included
  - function needs argc, argv and a format string with valid options

Argument parsing

- **int getopt (int argc, char *const argv[], const char *optstring)**
  - **optstring**
    - a letter without colon afterwards defines an option without argument
    - a letter with colon afterwards defines an option with mandatory argument
  - **EOF is returned as the last argument after looping through the options**

  - **global variable optarg points to argument of an option**
  - **global variable optind shows the index of next element to be processed in argv array**

  - **further examples:**
    - https://manpages.ubuntu.com/manpages/jammy/en/man3/getopt_long_only.3.html

Argument parsing example

```c
int c;
int isAEnabled = 0;
while( (c = getopt(argc, argv, "af:")) != EOF ){
  switch( c ){
    case 'a': /* Option without argument */
      isAEnabled++;
      break;
    case 'f': /* Option with argument */
      printf("%s", optarg);
      break;
    case '?': /* illegal argument */
      print_usage();
    default: /* unmöglich */
      assert( 0 );
  }
}
```

FH University of Applied Sciences TECHNIKUM WIEN

Thread synchronization (linux)

Thread synchronization

- **Complex programs require**
  - **Shared data**
  - **Access in defined order**

- **So, it requires synchronization**

Thread synchronization

- **„critical section“**
  - **Code that manipulates shared data in parallel so that it can potentially become inconsistent**

- **Ensure:**
  while data is manipulated
  no other instance can access the data
  before the process is finalized.

Thread synchronization

- **Cooperative process**
  - **Read / write access must secure critical sections**
  - **Access to shared data must be synchronized (check for others, authorize)**
  - **Secure information exchange between threads must be guaranteed**

- **Use:**
  - **Mutex (MUTual EXclusive)**
    - Special semaphore implementation
  - **Condition Variables**

Mutex - Functions

- **pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;**

- **int pthread_mutex_init (pthread_mutex_t *mutex,**
  pthread_mutexattr_t *attr);

- **int pthread_mutex_destroy (pthread_mutex_t *mutex);**

example

```c
#include <pthread.h>

  /* declare a structure with a mutex */
  typedef struct my_struct_tag {
    pthread_mutex_t mutex;/*protected access*/
    int value; /*value to protect*/
  } my_struct_t;

  my_struct_t data = {PTHREAD_MUTEX_INITIALIZER, 0};
```

Lock and Unlock Mutexes

- **int pthread_mutex_lock (pthread_mutex_t *mutex);**
- **int pthread_mutex_trylock (pthread_mutex_t *mutex);**
- **int pthread_mutex_unlock (pthread_mutex_t *mutex);**

Example (1/3)

```c
#include <stdio.h>
#include <pthread.h>

  int a=1000; // balance of account A
  int b=500; // balance of account B
  int sumValue=0;
  pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

  void *move(void *arg) {
    int status;
    pthread_detach(pthread_self());

    // critical section
    status=pthread_mutex_lock(&mutex);
    a=a-*((int *)(arg));
    sleep(1);
    b=b+*((int *)(arg));
    status=pthread_mutex_unlock(&mutex);

    printf("transaction completed\n");
  }
```

Example (2/3)

```
1   void *sum(void *arg) {
2     int status;
3     pthread_detach(pthread_self());
4     sleep(1);
5
6     // critical section
7     status=pthread_mutex_lock pthread_mutex_lock(&mutex);
8     sumValue = a + b;
9     status=pthread_mutex_unlock(&mutex);
10
11    printf("Sum of accounts: %d\n", sumValue);
12  }
```

Example (3/3)

```
1  int main() {
2     pthread_t thread1, thread2;
3     int x=500;
4
5     printf("old balance: a:%d b:%d\n",a,b);
6
7     pthread_create(&thread1, NULL, move, &x);
8     pthread_create(&thread2, NULL, sum, NULL);
9
10    while (!sumValue) {sleep (1);};
11
12    printf("new balance: a:%d b:%d\n",a,b);
13
14    return 0;
15 }
```

Threads ab C++ 11

C++ 11

- **Threads added in the standard library**
  - **g++ -std=c++11 –pthread**

```cpp
#include <iostream>
#include <thread>

// This function will be called from a thread
void call_from_thread() {
  std::cout << "Hello, World" << std::endl;
}

int main() {
  // Launch a thread
  std::thread t1(call_from_thread);

  // Join the thread with the main thread
  t1.join();

  return 0;
}
```

https://baptiste-wicht.com/posts/2012/03/cpp11-concurrency-part1-start-threads.html

example with a group of threads

```cpp
#include <iostream>
#include <thread>

  static const int num_threads = 10;

  // This function will be called from a thread
  void call_from_thread(int tid) {
    std::cout << "Launched by thread " << tid << std::endl;
  }

  int main() {
    std::thread t[num_threads];

    // Launch a group of threads
    for (int i = 0; i < num_threads; ++i) {
      t[i] = std::thread(call_from_thread, i);
    }

    std::cout << "Launched from the main\n";

    // Join the threads with the main thread
    for (int i = 0; i < num_threads; ++i) { t[i].join(); }
    return 0;
  }
```

mutex

```
1 #include <iostream>
2 #include <thread>
3 #include <mutex>
4
5   static const int num_threads = 10;
6   std::mutex m;
7
8   // This function will be called from a thread
9   void call_from_thread(int tid) {
10    m.lock();
11    std::cout << "Launched by thread " << tid << std::endl;
12    m.unlock();
13  }
```

Co-Routines and Fibres

Coroutine (c++20)

- A coroutine is a function that can suspend execution to be resumed later.

- This allows for sequential code that executes asynchronously (e.g. to handle non-blocking I/O without explicit callbacks), and also supports algorithms on lazy-computed infinite sequences and other uses.
  - co_await, co_yield, co_return

https://en.cppreference.com/w/cpp/language/coroutines

co_await

- uses the co_await operator to suspend execution until resumed

```
task<> tcp_echo_server() {
  char data[1024];
  for (;;) {
    size_t n = co_await socket.async_read_some(buffer(data));
    co_await async_write(socket, buffer(data, n));
  }
}
```

https://www.jacksondunstan.com/articles/6311

Fiber

- **Fibers describe essentially the same concept as coroutines.**

- **The distinction is that**
  - **coroutines are a language-level construct, a form of control flow, while**
  - **fibers are a systems-level construct, viewed as threads that happen to not run in parallel.**

[https://en.wikipedia.org/wiki/Fiber_(computer_science)]

- **Further reading:**
  - **Distinguishing coroutines and fibers**

FH University of Applied Sciences TECHNIKUM WIEN

InterProcess communication (IPC)

IPC

- **Data exchange between processes**
  - **data is safe in virtual address space**
  - **No other process is allowed to access another process‘**

- **But sometimes…**
  - **more processes need to share data**
  - **processes depend on each other and need to synchronize their doing**
  - **output of one process needs to be handed over to another process**
  - **Coordinate the usage of system resources**

IPC

- **Sychronization of processes**
  - **Timing of data manipulation**
    - Explicitly: synchronisation mechanisms
    - Implicitly: based on blocking data flow

- **Possibilities of IPC**
  - **Signals, (Un)named Pipes, Message Queues**
  - **Shared Memory, Semaphore**

Signals

- **Asynchronous IPC between processes**
  - **e.g.: <Ctrl + C> in terminal (SIGINT)**
  - **e.g.: terminate a child process (SIGCHLD)**

- **Sending of signals (signal.h)**
  - **int kill(pid_t pid,int signal)**

https://openbook.rheinwerk-verlag.de/c_von_a_bis_z/020_c_headerdateien_007.htm

Signals

- **Signalhandler**
  - **routines for handling signals**
  - **Registered using signal()**
  - **Called asynchronously to normal flow when signal is called**

```c
void signal_handler(int sig) {
  // cleanup
}

int main(int argc, char *argv[]) {
  // ...
  (void) signal(SIGHUP, signal_handler);
  (void) signal(SIGINT, signal_handler);
  (void) signal(SIGQUIT, signal_handler);
  // ...
}
```

https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm

Implicit Synchronization

- **Synchronization**

- **Automatically by using IPC mechanisms**
  - **Send / Receive of messages**
  - **Writing in (named) pipes**
  - **Read in (named) pipes**

- **No explicit synchronization mechanisms required**

Message Queues vs. Named Pipes

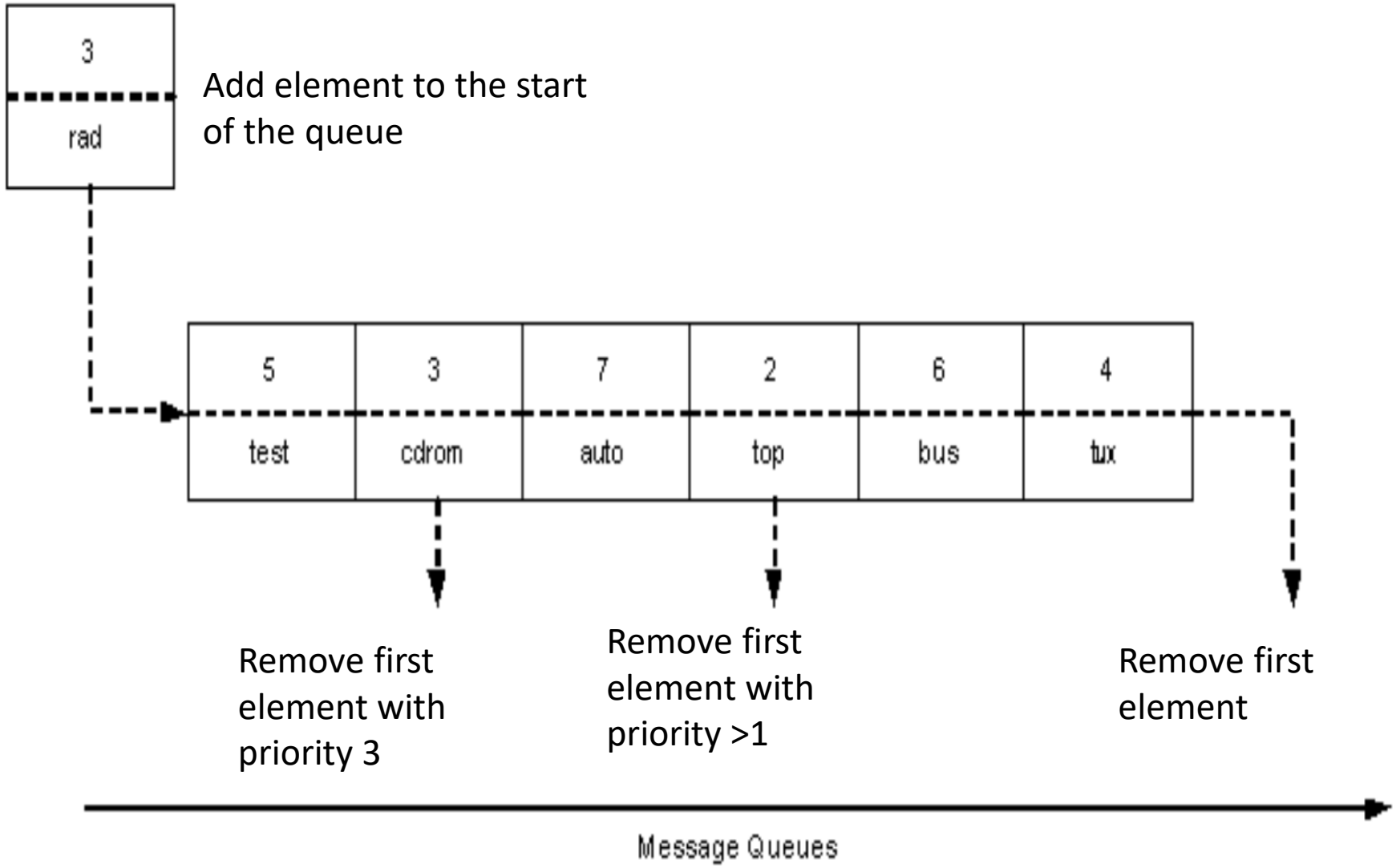
- **Message Queues**
  - **Structured communication**
  - **Exchange of messages via queues**
    - Messages == packet oriented
  - **Communication between multiple sender / receiver possible**
  - **Consuming receiver**

- **Named Pipe**
  - **FIFO Data transmission (First In – First Out)**
  - **Stream oriented**
    - Sequential reading only
  - **Problems with multiple sender / receiver**

Message Queues

- **Queues for messages to a process**
  - **each queue has a distinct identifier which is usable from other processes**
  - **Works like a mailbox for a given process**
    - Process can push data into a message queue which can be fetched by other processes
  - **Messages can be prioritized**
    - Fetch by FIFO or first message with defined priority

Message Queues



Message Queues

- **Operations**
  - **Create Message Queue**
  - **Send message**
  - **Receive message**
  - **Delete Message Queue**

- **Library functions in C available**
  - **#include <sys/types.h>**
  - **#include <sys/ipc.h>**
  - **#include <sys/msg.h>**

Create Message Queue

- **int msgget (key_t key, int flags);**
  - **Key**
    - system-wide unique ID
  - **Flags**
    - Permissions
      - **Similar to file permissions (0660 => r/w for group and owner)**
    - Creation Flags
      - **IPC_CREAT: creates queue if it does not exist**
      - **IPC_EXCL: returns an error if queue already exists**
      - **None: access to existing queue**
    - Permission and Creation Flags are “or”-ed using a pipe (e.g.: 0660|IPC_CREAT|IPC_EXCL)
- **Returns queue-id or -1 in case of an error**

example

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// ...

#define KEY 9825007
#define PERM 0666

  // ...

  int msgid;

  // ...

  if( (msgid = msgget(KEY,PERM|IPC_CREAT|IPC_EXCL ))==-1 ){
    /* error handling */
  }
```

messaging

- **Self-defined message structure**
  - **first element must be a long and defines the message type**
  - **the type (e.g.) defines the priority**
  - **Further structure can freely be defined**

```
1 typedef struct {
2   long mType;
3   char mText[MAX_DATA];
4 } message_t;
5
6 message_t msg;
```

Send Message

- **int msgsnd (int msgid, const void *msg, size_t msgsize, int flags);**
  - **msgid**
    - Queue id (result of msgget())
  - **msg**
    - pointer to the message
  - **msgsize**
    - Size of payload (without the first long)
    - Typically: sizeof(msg) - sizeof(long)
  - **flags**
    - Send-Mode
    - IPC_NOWAIT: error if queue is full
    - Else: blocks if queue is full
- **Returns 0 or -1 on error (errno)**

Receive message

- **int msgrcv (int msgid, void *msg, size_t msgsize, long msgtype, int flags);**
  - **msgid**
    - Queue id (result of msgget())
  - **msg**
    - pointer to the message
  - **msgsize**
    - Size of payload (without the first long)
    - Typically: sizeof(msg) - sizeof(long)
  - **msgtype**
    - 0: first message
    - x: first message with type == x
    - -x: first message with type <= x
  - **flags**
    - Send-Mode
    - IPC_NOWAIT: error if queue is full
    - Else: blocks if queue is full
- **Returns 0 or -1 on error (errno)**
- **Received message will be removed from queue**

Delete Message Queue

- **int msgctl (int msgid, int cmd, struct msqid_ds *buf);**
  - **msgid**
    - Queue id (result of msgget())
  - **cmd**
    - IPC_RMID: removes the queue
  - **buf**
    - Depends on cmd; IPC_RMID: NULL
- **Returns 0 or -1 on error (errno)**
- **Only 1 process should remove the queue.**

Message queue management from shell

- **ipcs -q**
  - **Displays the message queues**

- **ipcrm –q msgid**
  - **Deletes the message queue based on the message id**

- **ipcrm –Q key**
  - **Deletes the message queue based on the message key**

Named Pipes

- **Unstructured communication**
  - **stream based**
  - **No structured data**
  - **usage similar to file access**
  - **FIFO Message flow**

- **Consuming receiver**
  - **Reading deletes message from pipe**

- **Operations**
  - **Creation of a named pipe**
    - Also possible from the shell using mkfifo
  - **Deletion of a named pipe**
  - **Send and receive using stdio operations**

Named Pipes example

- **Often used in client / server applications**
  - **E.g.: Printer spooler**
  - **Client writes job in a named pipe**
  - **Server processes request (FIFO)**

Create Named Pipe

- **int mkfifo (const char *path, mode_t mode);**
  - **Path**
    - "filename"
  - **Mode**
    - Permission
- **Returns 0 or -1 on error (errno)**

```
1 if (mkfifo("foo", 0660) == -1) {
2   /* error */
3 }
```

Read / Write Named Pipe

- **Using stdio**

- **Open: fopen**
- **Read: fgets, fgetc**
- **Write: fputs, fputc, fprintf**
- **Close: fclose**

Remove Named Pipe

- **int remove (const char *path);**
  - **path**
    - “filename”
- **Returns 0 or -1 on error (errno)**
- **Only 1 process should remove the named pipe**

Named Pipe IPC

- **Open**
  - **A process using named pipes for reading is blocked until a process for writing is opened (vice versa).**

- **Read**
  - **If a pipe is opened for writing by at least one process, reading blocks in case of an empty pipe**
  - **If a pipe is not opened for writing, reading returns EOF (and does not block).**

Named Pipe IPC

- **Write**
  - **If a pipe is opened for reading by at least one process, writing blocks in case of a full pipe**
  - **If a pipe is not opened for reading, writing receives signal SIGPIPE (and does not block).**
- **Multiple Reader/Writer**
  - **No low-level synchronization of reading and writing**
  - **Chaos / data confusion / data jumble**

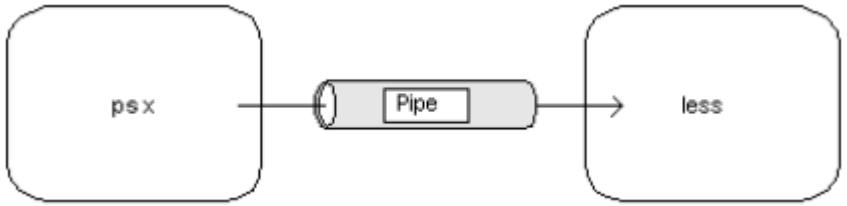
Comparison

- **Message Queues**
  - **message oriented**
  - **Structured data**
  - **Identified by numeric Key**
  - **Message selection possible**
  - **Read/write via API**
  - **Special store in OS**
  - **Concurrent writing into message queue is supported/managed by OS**
- **Named Pipes**
  - **stream oriented**
  - **Unstructured data**
  - **Identified by filename**
  - **Pure FIFO**
  - **Read/write via stdio-functions**
  - **Entry in filesystem**
  - **Application needs to handle concurrency**

Unnamed pipes

- **Communication between related processes**

- **Example: ps x | less**
  - **Ouput from ps x is forwarded as input of less**



- **Unidirectional connection (half-duplex)**
  - **Data transmission only in 1 direction**
  - **Bidirectional communication achieved by 2 pipes**

- **Synchronization between writing and reading process**
  - **Write blocks in case of a full pipe**
  - **Read blocks in case of an empty pipe**

Create Pipes

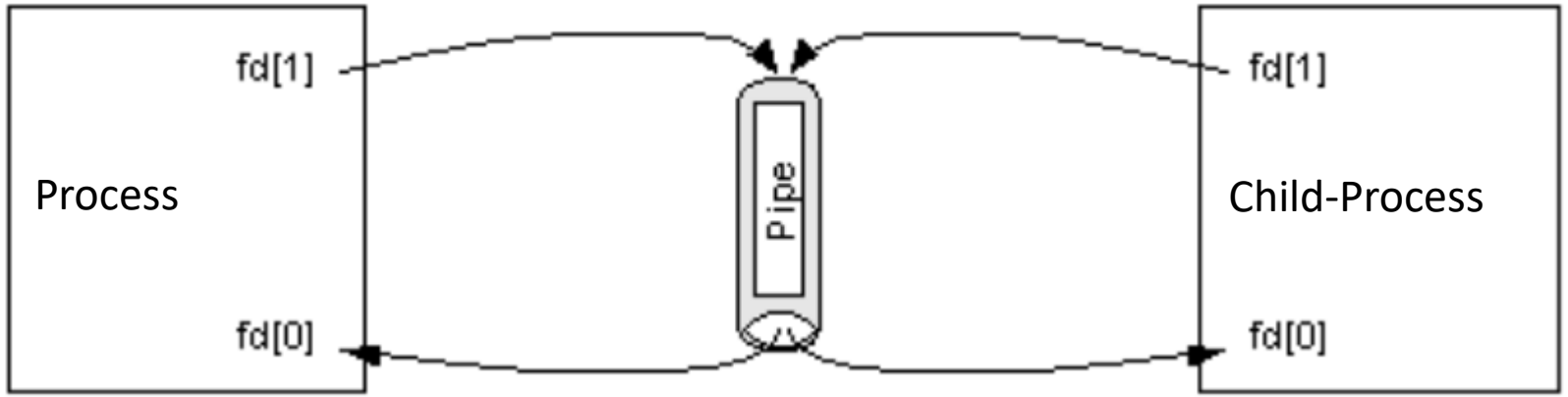
- **Pipes declared by a field of 2 integer Elements**
  - **int fd[2];**
    - fd[0] … for reading operations (compare stdin)
    - fd[1] … for writing operations (compare stdout)

- **Open an unnamed pipe using system calls:**

```c
#include <limits.h>
#include <unistd.h>

int fd[2];

// ...

if (pipe(fd)!=0) /* Error */
```

Pipes with fork()

- **The pipe is inherited into a child-prcess (fork())**



- **Afterwards the unused descriptor needs to be closed**
  - **Writing process closes read-descriptor (close(fd[0]);)**
  - **Reading process closes write-descriptor (close(fd[1]);)**
  - **Valid scenarios:**
    - Parent: reader, Child: writer
    - Parent: writer, Child: reader

Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

int main(void) {
  int fd[2], nbytes;
  pid_t childpid;
  char string[] = "Hello, world!\n";
  char readbuffer[80];
  pipe(fd);
  if((childpid = fork()) == -1) {
    perror("fork");
    exit(1);
  }

  if(childpid == 0) {
    /* Child process closes up input side of pipe */
    close(fd[0]);

    /* Send "string" through the output side of pipe */
    write(fd[1], string, strlen(string)+1);
    exit(0);
  } else {
    /* Parent process closes up output side of pipe */
    close(fd[1]);

    /* Read in a string from the pipe */
    nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
    printf("Received string: %s", readbuffer);
  }

  return(0);
} // end of main
```

Further usage of unnamed pipe

- **After the creation of the pipe the standard i/o functions can be used for further access.**
  - **Therefore, the file descriptor needs to be converted into a file pointer (using fdopen with file descriptor and mode).**

```
1 FILE *rd, *wr;
2
3 rd = fdopen(fd[0], "r");
4 wr = fdopen(fd[1], "w");
```

Redirection of descriptors

- **In case a child process using exec() calls an existing program the file descriptors for stdin (0) and stdout (1) can be redirected to a pipe using dup2().**

- **dup2 (int oldd, int newd)**
  - **The old descriptor will be duplicated and connected to the new descriptor.**

Example: connect stdin of a process to a pipe

```c
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

if (dup2 (fd[0], STDIN_FILENO) != STDIN_FILENO) {
  /* error using dup2 ... */
}

close (fd[0]); /* not required anymore */
```

popen

- **Merge of: pipe(), fork(), exec()**
  - **Opens a pipe, create a child-process, closes the not required descriptors and overloads the child-process with an exec function**

- **FILE *popen (char *command, char *type);**
  - **Command**
    - Filename of a program
  - **Type**
    - Pipe direction
    - „w“: written data is redirected to stdin of the command
    - „r“: output of the command is redirected to be available using the pipe

- **Use pclose() to close the opened file pointer**
  - **pclose waits for the command to finish**

example

```c
#include <stdio.h>
#define MAX 256

int main(int argc, char* argv[]) {
  FILE *fp;
  char puffer[MAX];

  if ((fp=popen("ls -l","r"))!=NULL) {
    while (fgets(puffer, MAX, fp) != NULL) { /* ... */}
  }

  pclose (fp);
  return 0;
}
```

FH University of Applied Sciences TECHNIKUM WIEN

Basics

Distributed Systems

- **In the early days of computers, we had huge monolithic systems**
  - **Computation was done on 1 machine**
  - **Logic Centralized**

- **→ Restructure huge complex system in smaller pieces**

Definition

- **A distributed system consists of a collection of autonomous computers linked by a computer network with software designed to produce an integrated computing facility.**

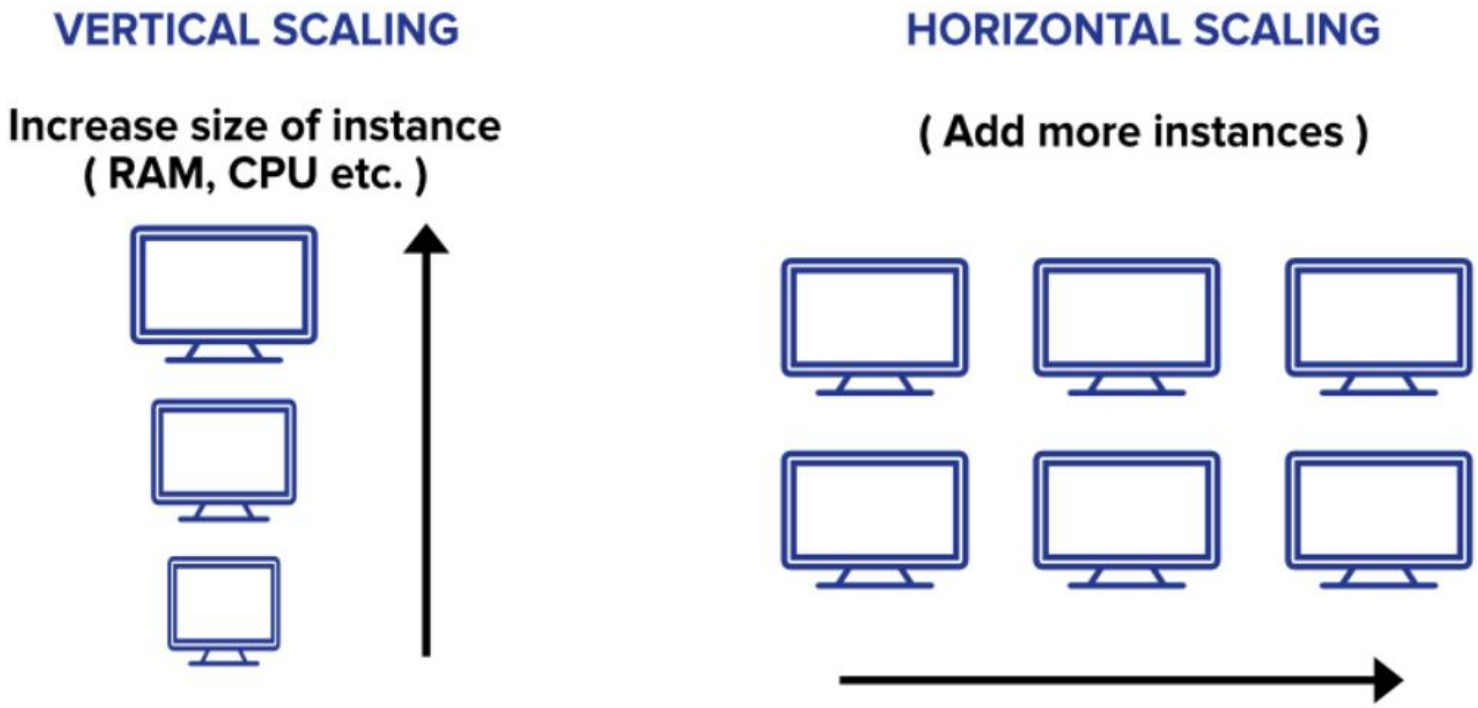
Examples

- **ATM**
- **Booking system**
- **Navigation System**
- **Distributed services (DNS, LDAP, Messaging services, …)**
- **Distributed file systems (NFS, AFS, SMB, …)**
- **Component based Systems (CORBA)**
- **Web-Services (SOAP)**
- **Restful Services (HTTP / JSON)**

Advantages

- **Good ratio price / performance**
- **Enables resource sharing (printer, file-server, internet access,…)**
- **Better throughput (parallel distributed execution, load balancing)**
- **High Availability**
- **Reliable because of redundancy / replication (Data, Process,…)**
- **Extendable**
- **Evolutionary (step by step) exchange of components**

Scaling



https://www.geeksforgeeks.org/system-design-horizontal-and-vertical-scaling/

Requirements

- **Resource Sharing**
  - **Hardware (Printer, Drives, Processors)**
  - **Data and services**
  - **Problem: consistency and synchronization (CAP theorem)**
  - **Architecture: Peer-to-peer, Client/Server**

Requirements

- **Openness**
  - **heterogeneous hardware, operating systems, environment (grid-computing)**
  - **Beneficial:**
    - Usage of open standards
    - Clearly defined interfaces
    - Consistent IPC on application level

Heterogeneous Systems - Motivation

- **Heterogeneity requires standardized interfaces**

"An open distributed system is essentially a system that offers components that can easily be used by, or integrated into other systems. At the same time, an open distributed system itself will often consist of components that originate from elsewhere.

To be open means that components should adhere to standard rules that describe the syntax and semantics of what those components have to offer (i.e., which service they provide)"

- M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017., p. 12

Heterogeneous Systems - Motivation

- *"**Interoperability** characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard."*
- *"**Portability** characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A."*

- M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017., p. 13

Heterogeneous Systems - Motivation

- **Applications and systems must be interoperable despite**
  - **… non-uniform hardware**
  - **… non-uniform operating systems**
  - **… non-uniform programming languages**
  - **… non-uniform communication protocols**

- **Advantages of heterogeneity:**
  - **Allows better adaptation to different requirements and environments**
  - **No dependency on a certain technology or vendor, and thus prevents monopolization**

Making Distribution Transparent

- **An important goal is to hide the fact that**
  - **Systems and services are heterogeneous**
  - **Processes and resources are physically distributed across multiple computers**
  - **Those computers may be separated by large distances**

- **Primary transparency attributes**

  Users and applications should not be aware of this distribution, it should be "transparent" (i.e., invisible) to them

Requirements

- **Location transparency:**
  users don‘t know about the location of a used resource
  - Access transparency: local or remote access identical for the process
  - Naming transparency: naming of a resource is identical on all nodes
- **Scaling transparency:**
  users don‘t know about an expansion of a used resource
  - Hardware
  - Started instances

Types of Distribution Transparency

Transparency	Description
Access	Provides a unified way of accessing objects that have different internal data representations
Location	User does not need to know where an object is physically located for accessing it
Relocation	Hides that an object may be moved to another location while in use
Migration	Moving an object to a new physical location does not change the way of accessing it
Replication	Hides that an object may be replicated
Concurrency	Multiple users may access the same object at the same time without noticing
Failure	Failure and recovery of a sub-component has no influence on the overall system
Latency	Response time is independent of traffic / server load

Access Transparency

- **Provide a unified way of accessing objects that have different internal data representations**

- **Example:**
  - **A distributed system may consist of computers with different operating systems and different file systems (e.g., Linux vs. Windows)**
  - **Resulting differences in file naming conventions, file operations or inter-process communication should be hidden to provide access transparency**

Location and Relocation Transparency

- **User does not need to know where an object is physically located for accessing it**
- **Hide that an object may be moved to another location while in use**
- **Naming is an important factor for (re-)location transparency. Using logical names helps with that (i.e., assigning names that do not encode the physical location of an object)**

- **Example: https://www.technikum-wien.at/ (URL)**
  - **No information of the actual server location encoded (location transparency)**
  - **Entire site may move to a different data center and the user won't notice (relocation transparency)**

Migration Transparency

- **Moving an object to a new physical location does not change the way of accessing it**
- **Relocation transparency refers to being *moved by the distributed system*, migration transparency supports mobility of processes and resources *initiated by users*, without affecting ongoing operations**

- **Example: Mobile phones**
  - **Two people may have a call while moving around without getting disconnected**

Replication Transparency

- **Hides that an object may be replicated**
- **For increasing availability or improving performance**

- **Examples:**
  - **RAID systems (Redundant Array of Independent Disks)**
  - **Data Mirroring**

Concurrency Transparency

- **Multiple users may access the same object at the same time without noticing**
- **Concurrent access should leave that object in a consistent state (e.g., through use of locking)**

- **Examples:**
  - **Two independent users access the same file on a file server or the same record in a database**
  - **Multiple users share a single printer**
  - **Thousand of users access the same webservice simultaneously**

Failure Transparency

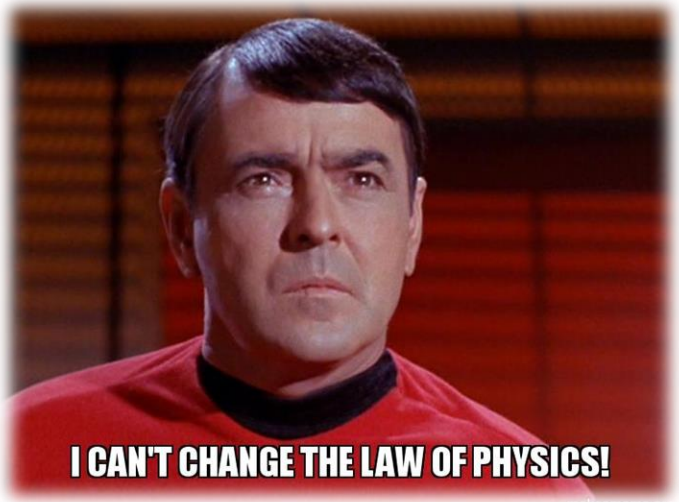
- **Failure and recovery of a sub-component has no influence on the overall system**
- **Masking failures is one of the hardest issues in distributed systems**
  - **Different types of failures need different (auto-)recovery strategies, automatically determining the correct type of failure might be tricky**
  - **E.g., a time-out when browsing a web page might have different reasons (high server load and thus slow response times vs. the server does not respond at all)**

- **Example:**
  - **Automatic switch to backup database server, when main database server fails (failover)**

Degree of Distribution Transparency

- **Being fully transparent might not always be preferable or possible**

  - **Messages may not travel faster than the speed of light, sending a message from San Francisco to Amsterdam will take at least 35 ms**
  - **Processing messages takes some amount of time**
  - **Trying to mask a server failure with retries may slow down the whole system, cancellation of a request might be the better solution**
  - **Requiring database replicas to be consistent at all times might slow down the whole system**



Degree of Distribution Transparency

- **Being fully transparent might not always be preferable or possible**
  - **Considering the current location of objects might actually be desirable, e.g., for location-based services**
  - **Making distribution explicit might improve the understanding of the system and its limitations**

“You know you have [a distributed system] when the crash of a computer you’ve never heard of stops you from getting any work done.”

- Leslie Lamport

Typical Exam Questions

- **Is the access of a website location transparent?**
- **Is the access of a network share (windows) access transparent?**
- **Which transparency attributes are key for a flight booking system?**

Requirements

- **Flexibility**
  - **Extensibility of components**
  - **No fixed monolithic structure**

- **Fault tolerance**
  - **Dependency to network**
  - **Unstable Transmission, Routing**
  - **Jitter (changing response times)**
  - **Time synchronization**
  - **High availability required**

- **Concurrency (Data Synchronization)**

Requirements

- **Scalability**
  - **Sizing of the system should be estimated roughly**
    - (10 users, 100 users, 1000 users,…)
  - **Scaling of components should be transparent**
    - No rewrite of code in case of required scaling to multiple instances (flexibility)
  - **No central components**
    - SPOF (single point of failure), bottlenecks
  - **No central data**
    - What happens if this database / component is down?
  - **No location dependent services / algorithms**
  - **Solution: Caching and Replication**
    - Backup or hot standby takes over the workload if primary server goes down. (active / passive cluster)
    - Swapping should be transparent for clients.

The 8 Fallacies of Distributed Computing (Peter Deutsch)

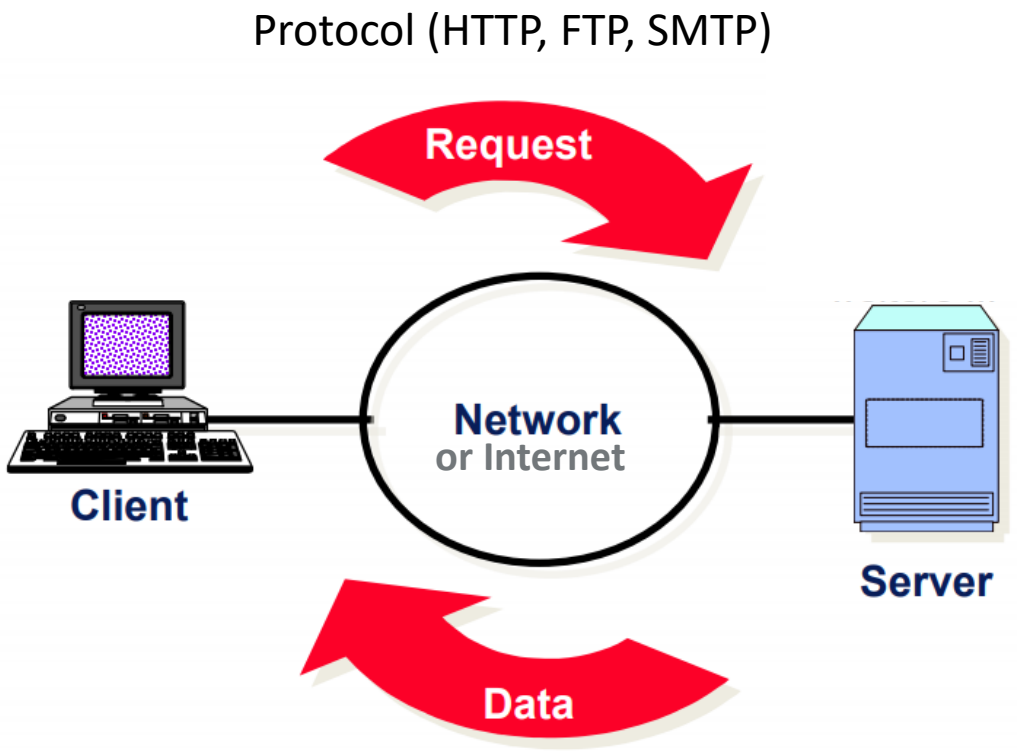
1. **The network is reliable**
2. **Latency is zero**
3. **Bandwidth is infinite**
4. **The network is secure**
5. **Topology doesn't change**
6. **There is one administrator**
7. **Transport cost is zero**
8. **The network is homogeneous**

Essentially everyone, when they first build a distributed application, makes the above eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences. (Peter Deutsch)

Client / Server Model

- **user uses client**
  - **Representation**

- **services provided on server**
  - **stores data**
  - **computation**

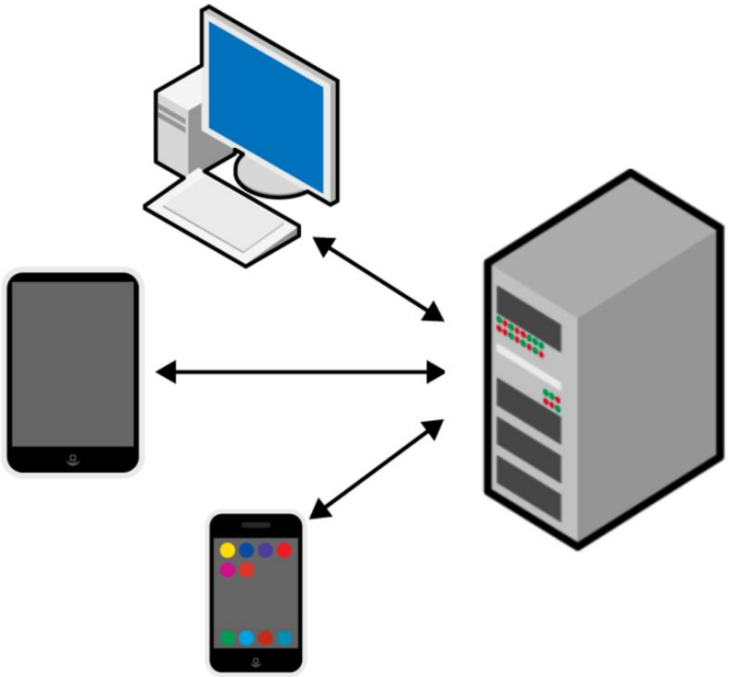


Tiers and Layers

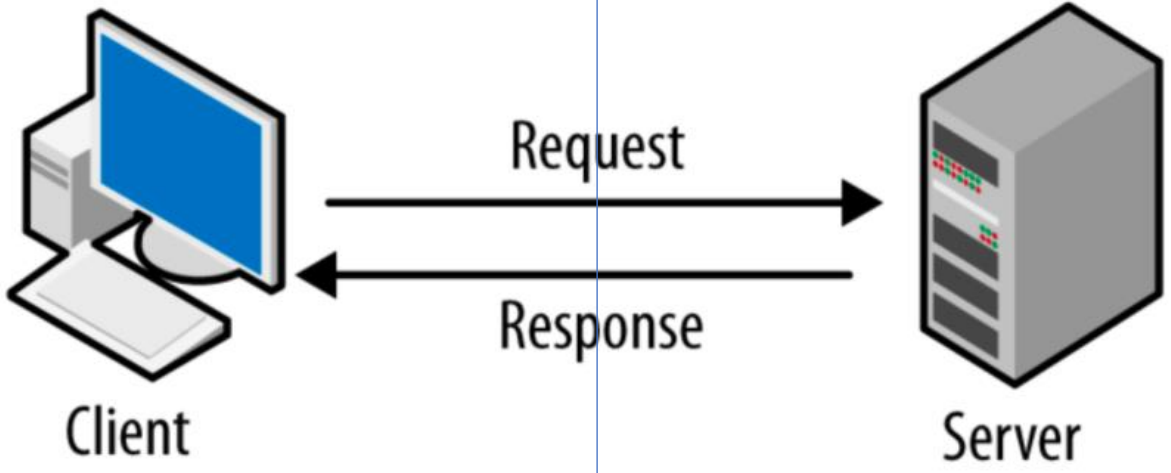
- **Layers describe the logical groupings of the functionality and components in an application**

- **Tiers describe the physical distribution of the functionality and components on separate servers, computers, networks, or remote locations.**

Client / Server Model



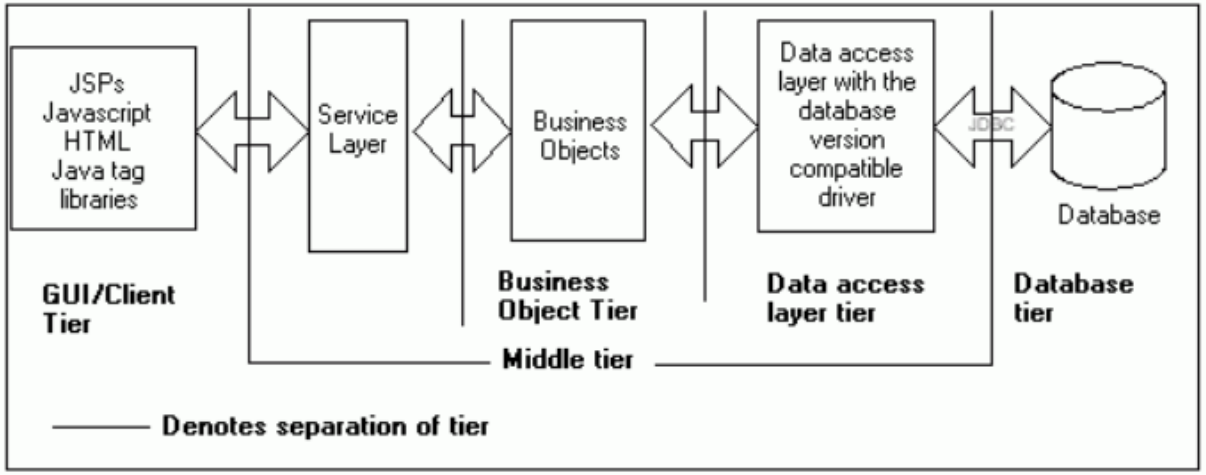
2 tier architecture

Presentation layer	logic layer
	
service user	service provider

https://madooei.github.io/cs421_sp20_homepage/client-server-app/

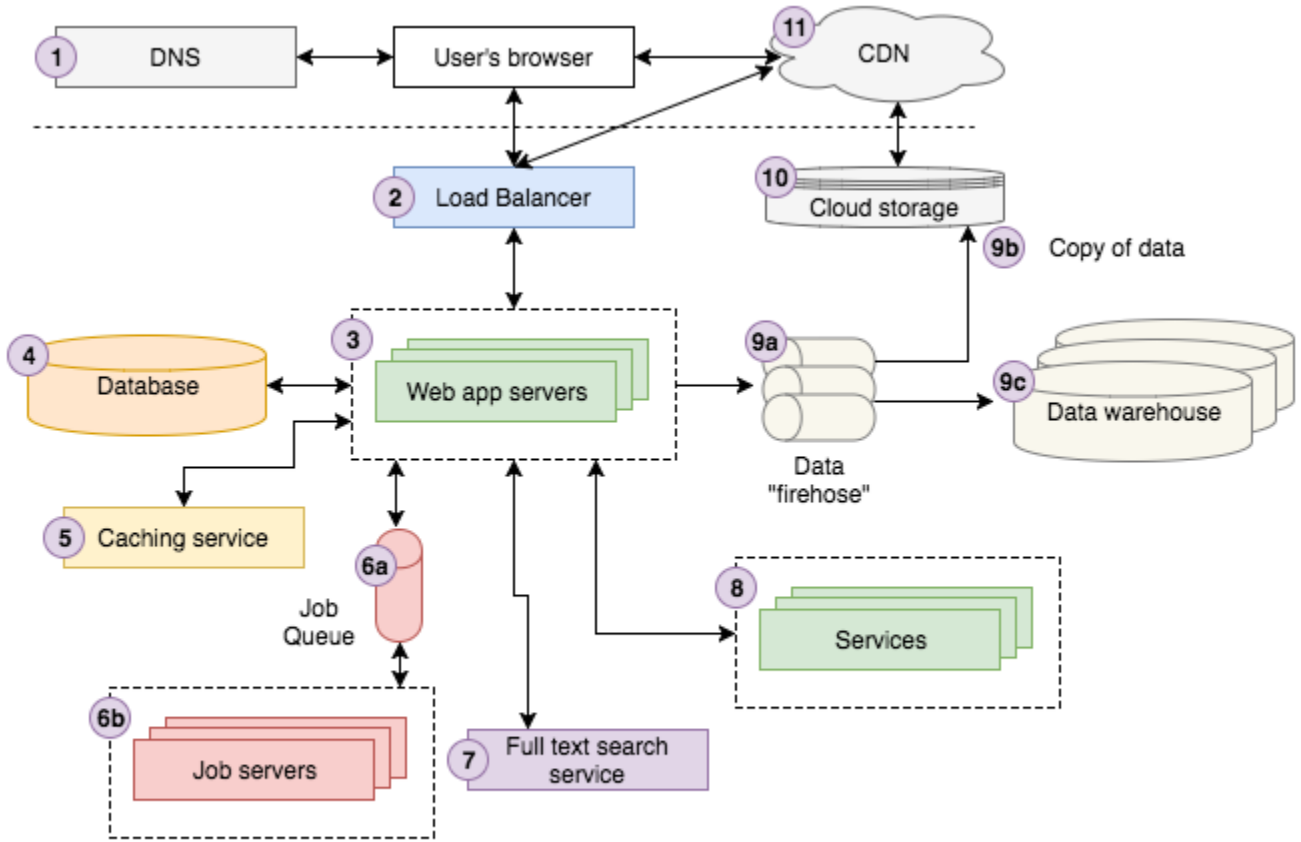
n tier (or multi tier) architecture

- **Persistence**
- **External services**
- **Data and Transactions**
- **Separation of Concerns**



https://docs.oracle.com/cd/E76467_01/alloc/pdf/1329/html/operations_guide/Chapter3.htm

Web Architecture (example)



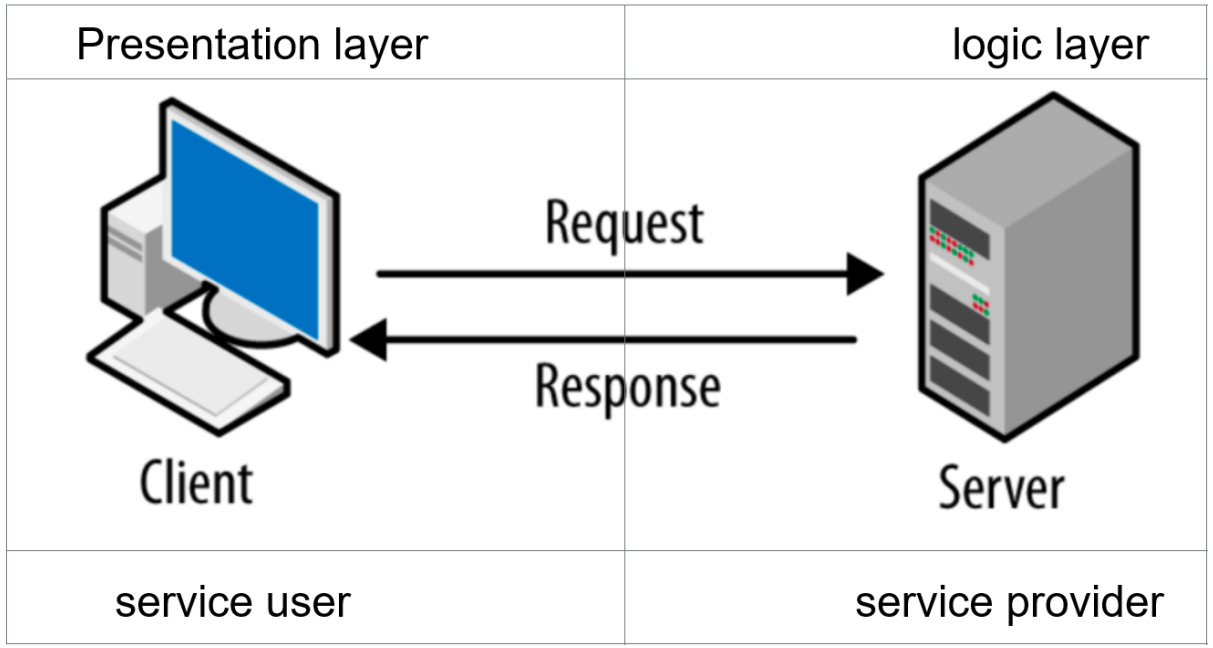
Modern web application architecture overview

https://medium.com/storyblocks-engineering/web-architecture-101-a3224e126947

Network – Development (Linux)

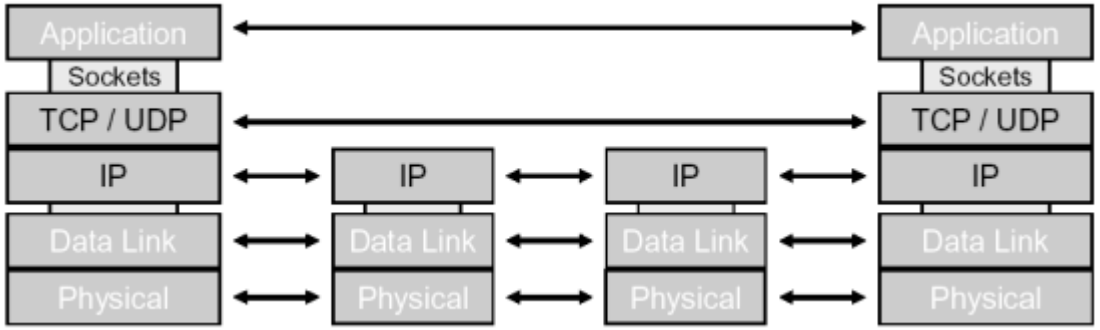
Client-Server

2 tier architecture



- **Socket – Communication**
  - **TCP – connection oriented (1:1)**
  - **UDP – connectionless (1:n)**

The socket

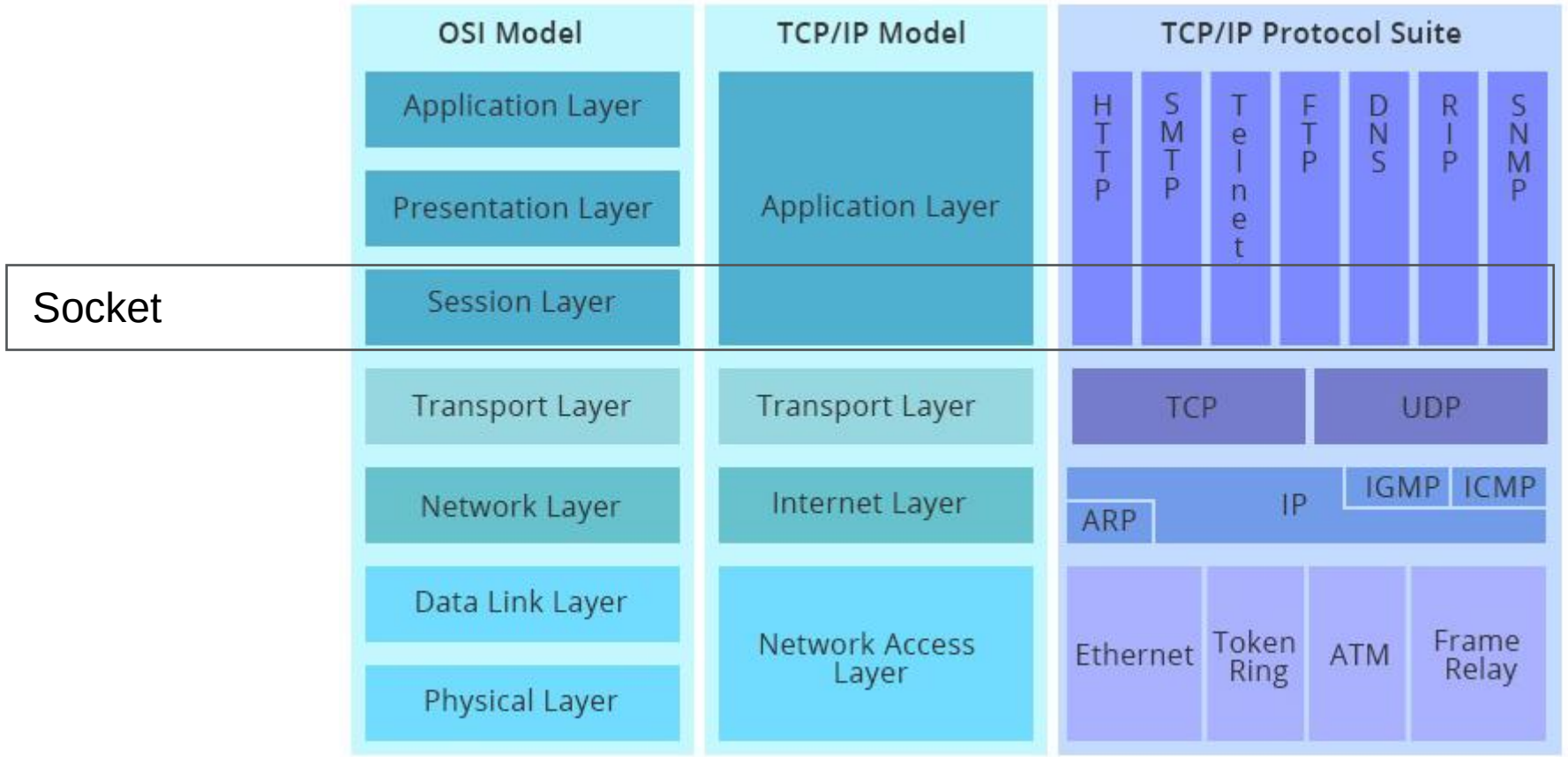


- **Developed in BSD Unix**
  - **available in Unix/Linux**
  - **Compliant Implementation in Windows (Winsock)**

Socket attributes

- Network applications „Create“, „Use“ and „Release“ sockets
- Create address on application level (using port numbers)
- Enable read and write of data
  - Internally addressed by using „descriptors“
  - Workflow very similar to file access (utilizing standard I/O operations)
- Enable buffering
- Implemented in the kernel (linux) and accessed by system calls

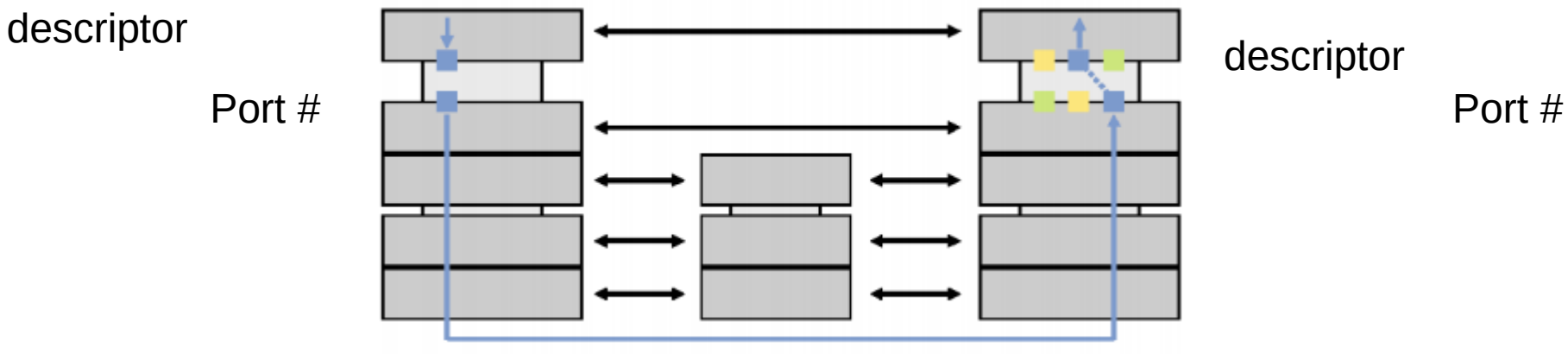
ISO / OSI model



https://community.fs.com/de/blog/tcpip-vs-osi-whats-the-difference-between-the-two-models.html

Ports

- **Why are they useful?**
  - **1 host can provide multiple services**
  - **each network-enabled service uses (at least) one socket**
  - **a socket enables the usage of the TCP/IP stack**
  - **clients connect to sockets using host address and socket**
  - **1 to 1 relationship between descriptors / sockets and port numbers**



Port attributes

- **16-bit number (0-65535 (2^16-1))**
- **Well known ports (0 – 1023)**
  can only be provided by privileged users (root)
  - **See list of well-known ports**
- **Further: Registered ports**

Port	TCP	UDP	Name	Beschreibung
1	✓	✓	tcpmux	TCP Port Multiplexer
5	✓	✓	rje	Remote Job Entry (Jobferneingabe)
7	✓	✓	echo	Echo-Service
9	✓	✓	discard	Null-Service für Prüfzwecke
11	✓	✓	systat	Systeminformationen
13	✓	✓	daytime	Zeit- und Datumsangaben
17	✓	✓	qotd	Sendet Zitat des Tages
18	✓	✓	msp	Übermittelt Textnachrichten
19	✓	✓	chargen	Sendet eine endlose Zeichenkette
20	✓		ftp-data	FTP-Datenübertragung
21	✓	✓	ftp	FTP-Verbindung
22	✓	✓	ssh	Secure Shell Service
23	✓		telnet	Telnet-Service
25	✓		smtp	Simple Mail Transfer Protocol
37	✓	✓	time	Maschinenlesbares Zeitprotokoll
39	✓	✓	rlp	Resource Location Protocol
42	✓	✓	nameserver	Name-Service
43	✓		nicname	WHOIS-Verzeichnisservice
49	✓	✓	tacacs	Terminal Access Controller Access Control System
50	✓	✓	re-mail-ck	Remote Mail Checking
53	✓	✓	domain	Namensauflösung per DNS
67		✓	bootps	Bootstrap Protocol Services
68		✓	bootpc	Bootstrap Client
69		✓	tftp	Trivial File Transfer Protocol
70	✓		gopher	Dokumentensuche
71	✓		genius	Geniusprotokoll
79	✓		finger	Liefert Kontaktinformationen von Benutzern
80	✓		http	Hypertext Transfer Protocol

https://www.ionos.at/digitalguide/server/knowhow/tcp-und-udp-ports/

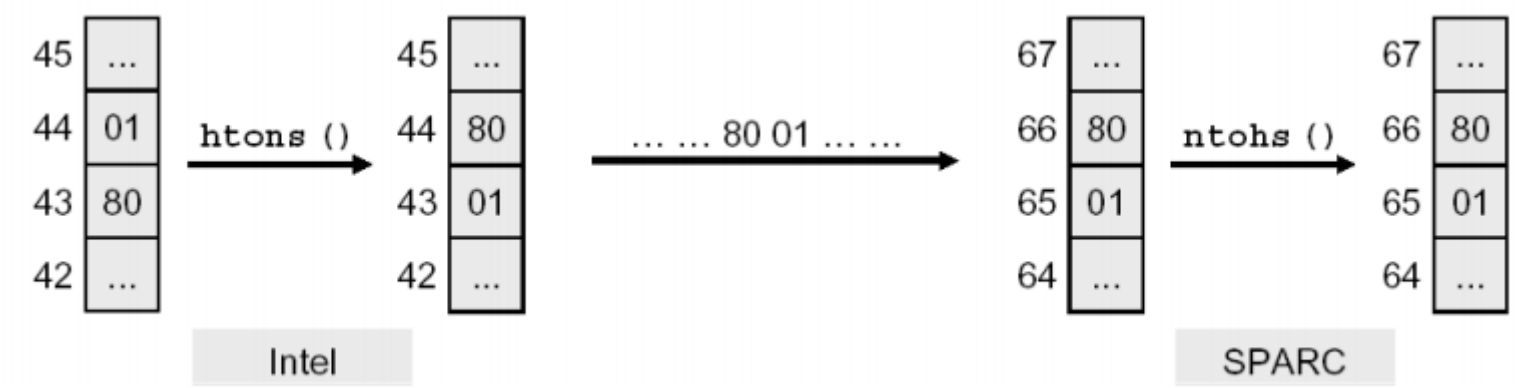
Connection

- **Defined by:**
  - **Source IP**
  - **Source Port**
  - **Destination IP**
  - **Destination Port**
  - **Transport Protocol (TCP, UDP)**

Connection attributes

- **Byte ordering**
  - **Big endian (bytes with highest value first; SPARC, Motorola)**
  - **Little endian (bytes with lowest value first; Intel)**

- **Network byte order**
  - **Global agreement how bytes should be ordered (big endian)**
  - **Conversion on application layer**

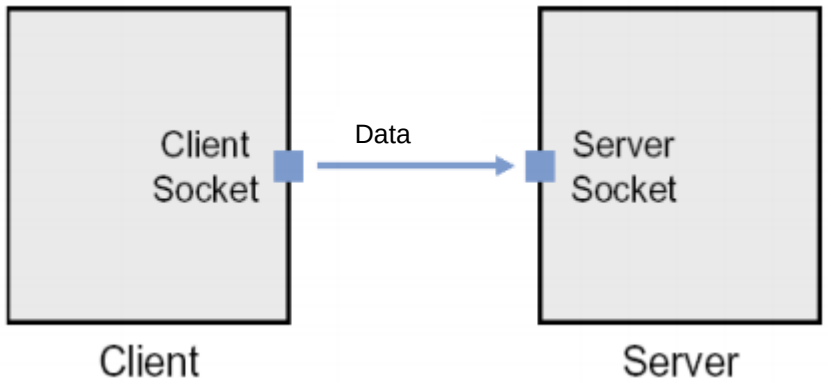


C – Network Programming

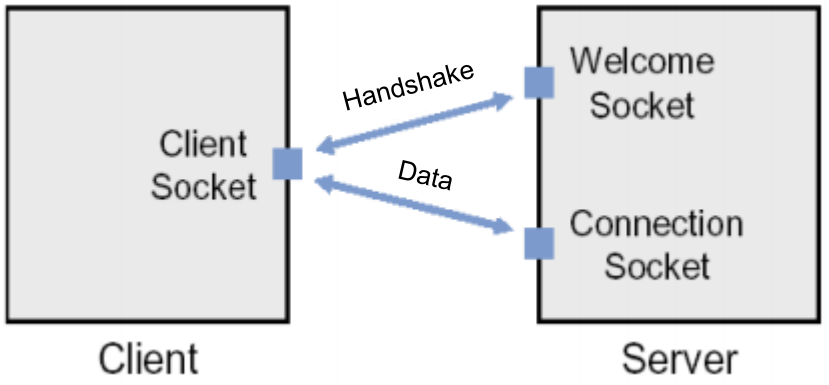
https://github.com/kienboec/ClientServerSampleInC/

Comparison UDP / TCP

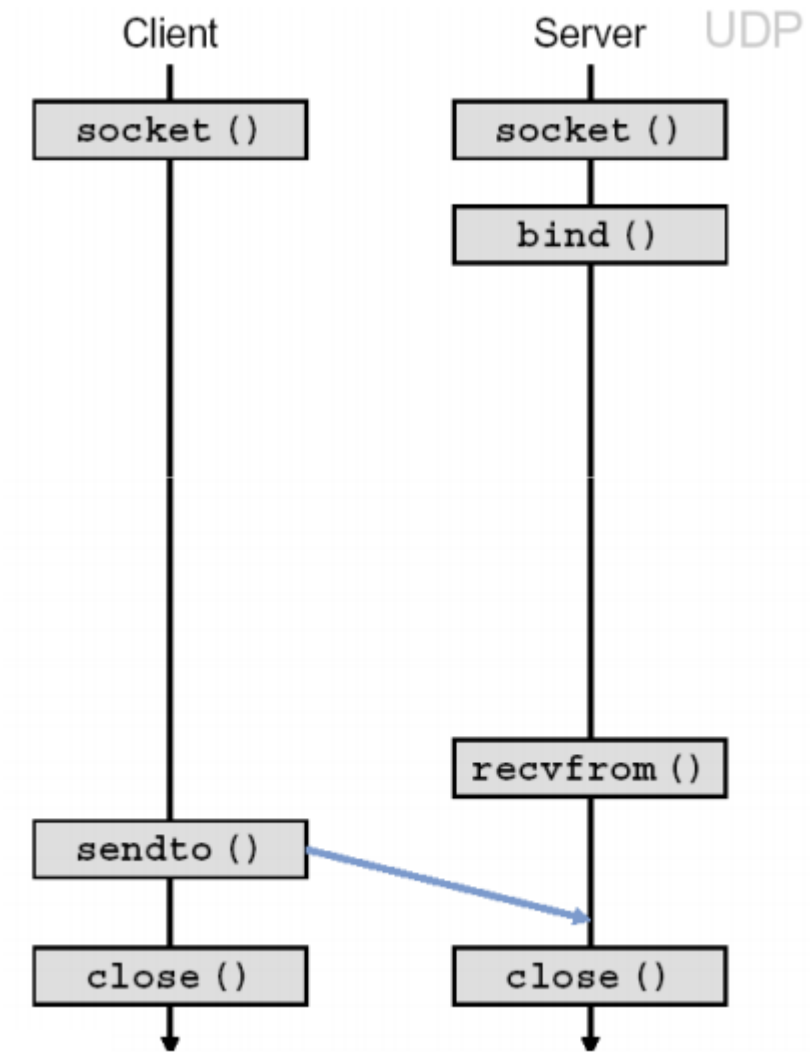
- **UDP**
  - **1 socket on the client**
  - **1 socket on the server**
  - **No connection setup**
  - **Unidirectional data transmission**

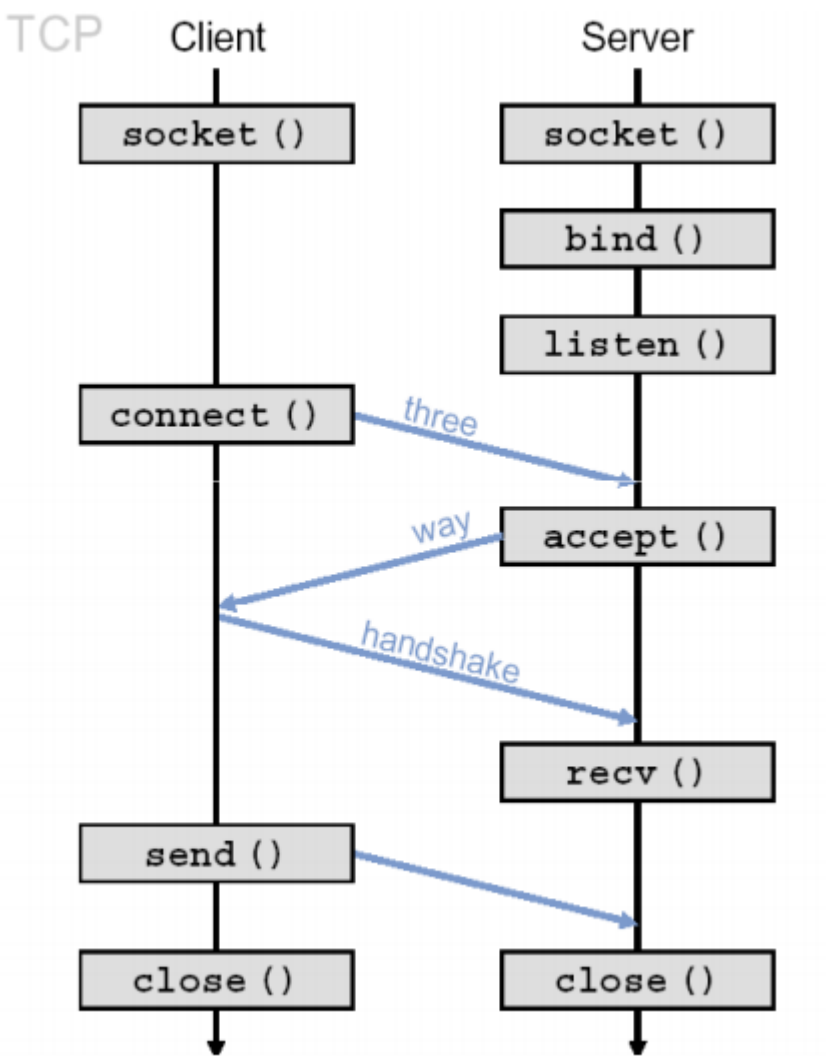


- **TCP**
  - **1 socket on the client**
  - **2 sockets on the server**
    - welcome or listen socket to setup connection (handshake)
    - connection socket for data transmission
  - **Bidirectional data transmission**



Comparison UDP / TCP (flow)





Connection related - functions

https://github.com/kienboec/ClientServerSampleInC/

Agenda

- Creating a socket
  - socket(domain, type, protocol)
- Assign an address
  - bind(socket, addr, addrlen)
- Setup a connection
  - connect(socket, addr, addrlen)
- Allow a client to establish a connection
  - listen(socket, backlog)
- Accept a client connection
  - accept(socket, addr, addrlen)
- Closing a connection
  - Close(socket)

Create Socket

- **int socket (int family, int type, int protocol)**
  - **family:**
    - AF_INET for IPv4
    - AF_INET6 for IPv6
    - AF_LOCAL for UNIX domain sockets
  - **type**
    - SOCK_STREAM for Stream Sockets (TCP)
    - SOCK_DGRAM for Datagram Sockets (UDP)
    - SOCK_RAW for Raw Sockets directly on IP
  - **protocol**
    - In general set to zero (used only for raw sockets)

network byte order - functions

- **Includes**

```
#include <sys/socket.h>    // socket(), bind(), ...
#include <netinet/in.h>    // struct sockaddr_in
#include <arpa/inet.h>     // inet_ntoa(), ...
#include <unistd.h>        // read(), write(), close()
#include <errno.h>         // global var errno
```

- **Conversion Functions (for shorts and longs)**

```
#include <netinet/in.h>
htons() host-to-network
ntohs() network-to-host
htonl() host-to-network
ntohl() network-to-host
```

Socket address - data structure

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

= IPv4

- https://man7.org/linux/man-pages/man7/ip.7.html
- https://github.com/kienboec/ClientServerSampleInC/blob/master/src/myserver.c

Assign Socket

- **int bind (int sd,**
  const struct sockaddr *myaddr,
  socklen_t myaddr_len)

  - **sd**
    - socket descriptor (result of the socket()-function)
  - **myaddr**
    - pointer to the address structure
  - **myaddr_len**
    - size of the address structure (in byte)

- **bind() associates descriptor with IP / port**
- **makes the port visible from outside**
- **used on the server-side (only)**

Simplified example

```
1   int socket_fd;
2   struct sockaddr_in my_addr;
3
4   socket_fd = socket (AF_INET, SOCK_DGRAM, 0);
5
6   memset(&my_addr,0,sizeof(my_addr));
7   my_addr.sin_family = AF_INET;
8   my_addr.sin_port = htons (6000 );
9   my_addr.sin_addr.s_addr = htonl (INADDR_ANY);
10
11  if ( bind( socket_fd,(struct sockaddr *) &my_addr, sizeof (my_addr)) == -1) {
12    perror("bind error");
13    exit(1);
14  }
```

Convert IP-string to address data structure

- int inet_aton (const char *cp, struct in_addr *inp)
  - cp: string containing an IP
  - inp: pointer to data structure
- Address already in network byte order
- reverse function: inet_ntoa

Setup a connection (TCP)

- **int connect (int sd,**
  const struct sockaddr *servaddr,
  socklen_t addrlen)
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **servaddr**
    - pointer to the address structure
  - **addrlen**
    - size of the address structure (in byte)

- **Establishes a connection to the server (TCP only)**
- **Starts the 3-way-handshake (SYN / ACK)**

Allow a client to establish a connection

- **int listen (int sd, int backlog)**
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **backlog**
    - count of queued connections allowed to wait for an accept
- **In case the queue is full the error „connection refused“ will be returned to the client.**

Accept a client connection

- **new_sd = accept (int sd,**
  struct sockaddr *cliaddr,
  socklen_t *addrlen)
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **cliaddr**
    - pointer to the address structure of the client
  - **addrlen**
    - size of the address structure (in byte)

- **returns a descriptor for further communication with the client**
- **blocks until a client connection is available**

Closing a connection

- **int close (int sd)**
  - **sd**
    - socket descriptor (result of the socket()-function)
- **no communication after the socket is closed**
- **frees the descriptor**

communication related - functions

https://github.com/kienboec/ClientServerSampleInC/

Agenda

- TCP read / write
  - send
  - recv
- UDP read / write
  - sendto
  - Recvfrom
- low-level I/O functions read and write can be used as well

Send to stream socket

- **int send(int sd, const void *msg, int len, int flags);**
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **msg**
    - pointer on data to be sent
  - **len**
    - Length of data (bytes)
  - **flags**
    - Mode; normally zero
- **send returns the count of bytes sent or -1 as error**
  - **Attention: returned size can be smaller than len!**
    - send small packages (e.g.: 1k)
    - use in a loop

(simple) send example

```
1   const char *string = "Hello!";
2   int len = strlen(string) + 1;
3   if (send(sd, string, len, 0) == -1)
4   {
5     /* error */
6   }
```

(complex) send example

```
1   int sendall(int sd, char *buf, int *len) {
2     int total = 0;        // total sent
3     int bytesleft = *len; // left to send
4     int n;                // currently sent
5
6     while(total < *len) {
7       n = send(sd, buf+total, bytesleft, 0);
8       if (n == -1) {
9         break;
10      }
11
12      total += n;
13      bytesleft -= n;
14    }
15
16    // return number actually sent here
17    *len = total;
18
19    // return -1 on failure, 0 on success
20    return n==-1 ? -1 : 0;
21  }
```

receive from socket stream

- **int recv(int sd, void *msg, int len, int flags);**
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **msg**
    - pointer on data to be received
  - **len**
    - Length of data (bytes)
  - **flags**
    - Mode; normally zero

- **blocking**
- **recv returns the count of bytes received, 0 if remote socket was closed or -1 as error**
  - **Attention: returned size can be smaller than len!**

(simple) receive example

```
1   char string[LEN];
2   int len=0;
3   if ((len=recv(sd, string, LEN-1, 0)) == -1){
4     /* error */
5   }
6
7   string[len] = '\0';
```

Alternative API-usage

- **Instead of send() and recv() the low-level API calls read() and write() can be used, because sockets are still descriptors under the hood.**
  - **Same parameters and syntax (except flags)**

  - **ssize_t read(int fd, void *buffer, size_t len);**
  - **ssize_t write(int fd, void *buffer, size_t len);**

Send to datagram socket

- **int sendto(int sd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);**
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **msg**
    - pointer on data to be sent
  - **len**
    - Length of data (bytes)
  - **flags**
    - Mode; normally zero
  - **To**
    - Pointer on receiver address
  - **Tolen**
    - Length of receiver address data in bytes
- **sendto returns the count of bytes sent or -1 as error**

receive from datagram stream

- **int recvfrom(int sd, void *msg, int len, int flags, struct sockaddr *from, int fromlen);**
  - **sd**
    - socket descriptor (result of the socket()-function)
  - **msg**
    - pointer on data to be received
  - **len**
    - Length of data (bytes)
  - **flags**
    - Mode; normally zero
  - **from**
    - Pointer to sender address data
  - **fromlen**
    - Length of sender address data in bytes
- **recv returns the count of bytes received or -1 as error**

Further useful functions

- Address of the other peer of a stream socket
  - int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
- Hostname
  - int gethostname(char *hostname,size_t size);
- DNS request
  - struct hostent *gethostbyname(const char *name);

Further useful tooling

- **netstat**
  Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships

- **tcpdump**
  dump traffic on a network

- **lsof**
  list open files
  (-i: this option selects the listing of all Internet and x.25 (HP-UX) network files.)

FH University of Applied Sciences TECHNIKUM WIEN

Client / Server Architecture

Design

- Connections can be
  - stateful or stateless
  - connectionless or connection oriented
  - Iterative or concurrent

Stateful server

- state is stored over multiple transmissions
- Problems:
  - Clients can unexpectedly be turned on / off
  - Network outage can lead to data loss or multiple data
- Implementation is complex

Stateless server

- Very common (especially on the internet)
- State is only available from connection start to end
- store state here is combined with high additional effort
  - e.g.: cookies

Connectionless server

- UDP
- low overhead / high performance
- no validation if data was truely received by the client
- reliability challenging (Application layer)
- Theoretically no limit of connected clients
  - all data received through the same socket descriptor
  - separation of clients needs to be done on application layer

Connection-oriented server

- typically on the Internet via TCP
- connection management taken over by the OS (operating system)
- overhead / bad performance
- reliability
- simple implementation
- 1 socket per connection (connection identifiable by descriptor)
- limited count of concurrent connections

Iterative Server

- server application works on 1 connection
- typical for simple TCP servers
- received requests are queued until open requests are closed
- blocking servers
- some requests might be refused

Concurrent Server

- the true server process
  - receives the requests
  - starts 1 child process per connection
  - hand over the communication data to the child process
- the child process
  - is reliable for any further communication
  - stops when the connection is closed

- high performance because of parallel execution
- typical for TCP servers with simultaneous connections
  - WWW, FTP,…
- each child process is allocating additional resources

Concurrent Server

Agenda

- forking server
- preforked server
- threaded server

forking server

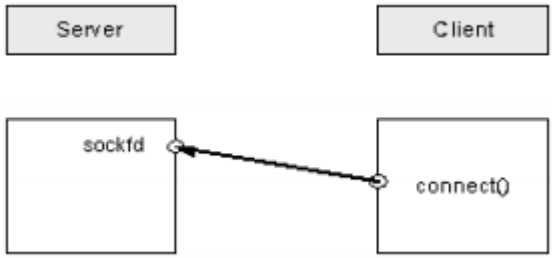
- Common unix solution
- Create a child process using fork()
- In TCP-based servers the fork() is called after accept()
- In UDP-based servers the fork() is called after recvfrom()
- fork() is not suitable for high performance

(simple) example

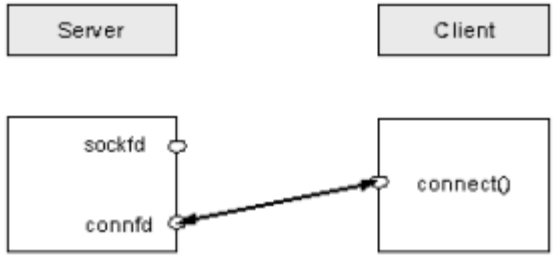
```
1   while(1) {
2     connfd = accept( sockfd,(struct sockaddr *)&adresse, &adrlaenge);
3     if( connfd < 0 ) {
4       if( errno == EINTR ) {
5         /* interrupted by a signal that was caught before a valid connection arrived */
6         continue;
7       } else {
8         printf("error in accept() ...\n");
9         exit(EXIT_FAILURE);
10      }
11    }
12
13    printf ("connection established ... ready for data\n");
14
15    /* create a child process */
16    if ((pid = fork ()) == 0) {
17      close (sockfd);
18
19      /* child process work here */
20
21      close(connfd);
22      exit(EXIT_SUCCESS);
23    }
24
25    /* parent process work here */
26    close (connfd);
27  }
```

Workflow

- **Setup connection through the client**

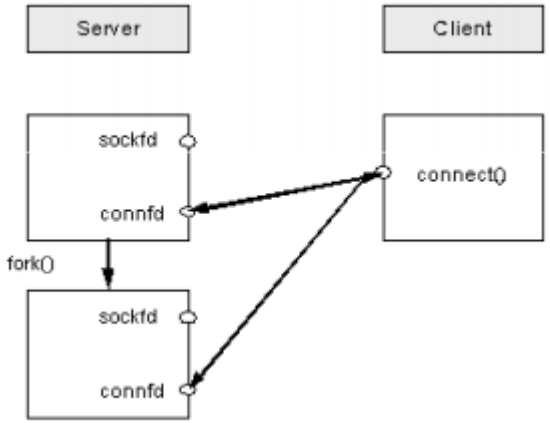


- **After 3 way handshake returning from accept()**



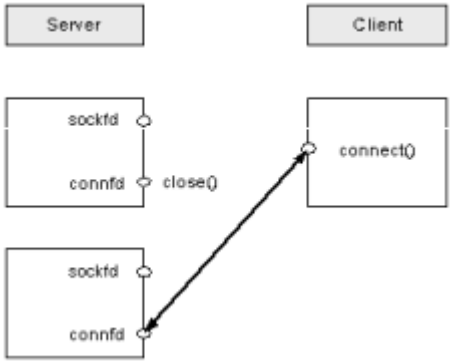
Workflow

- **After fork (child-process inherits open socket descriptors)**



Workflow

- **Unneeded socket-resources need to be closed**
  - **Child process closes listening-socket**
  - **Parent process closes connection-socket**



  - **After the connection is closed the child-process close()s the connection socket and terminates.**
  - **The parent process reacts on the termination to prevent a zombie process.**

Preventing zombie-processes

- **Signal-Handling in the parent process**
  - **Catch the SIGCHLD – signal**

```c
void no_zombie (int signr) {
  pid_t pid;
  int ret;
  while ((pid = waitpid (-1, &ret, WNOHANG)) > 0) {
    printf ("Child-Server with pid=%d stopped\n", pid);
  }
  return;
}

// ...

/* in main() */
(void) signal (SIGCHLD, no_zombie);
```

preforked server

- server creates a count of child-processes with same listening port
- each child-process blocks in accept()
- incoming requests are distributed child processes
- child-process not closed after the connection is closed
- in-case further child processes can be started (more instances)
- performance gain in comparison to forking server (reuse)

- Example: Apache Web-Server (1.x)

Threads

- 1 thread is started for each connection
- faster than fork(); less resource usage
- simpler exchange of data between different threads
- prethreaded server possible (see: Thread-Pooling)
- count of threads per process is limited
- Combination [pre]threaded / [pre]forked server possible

(simple) example

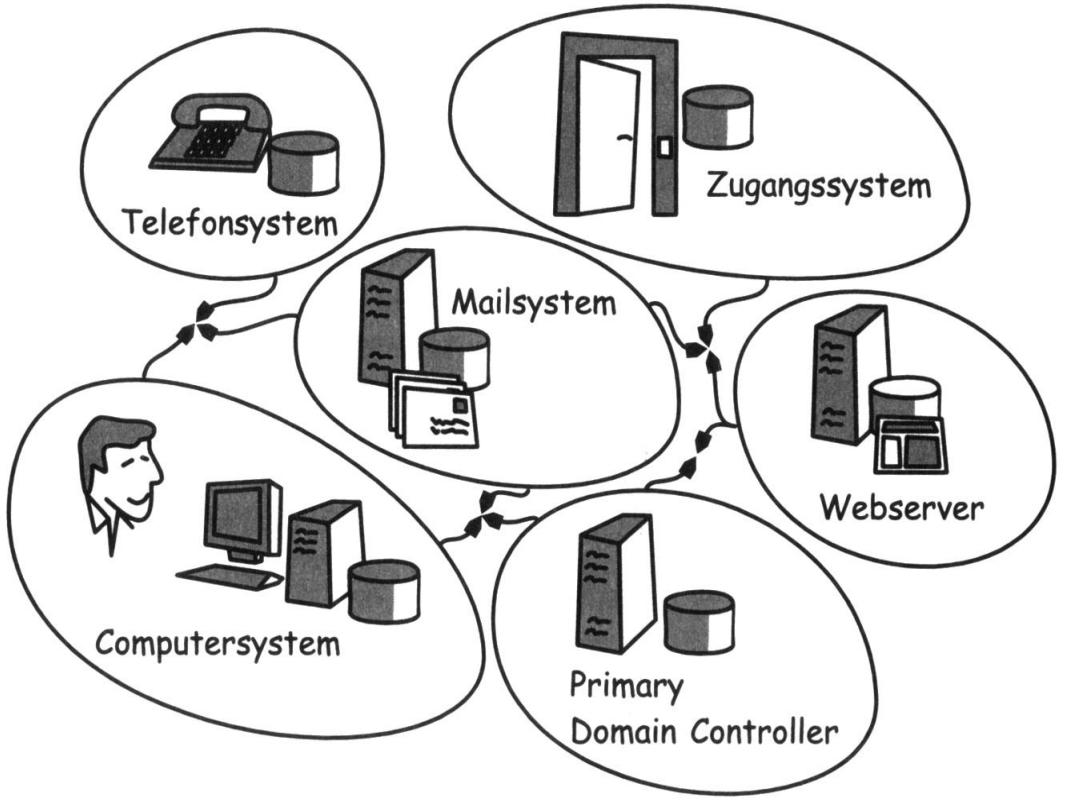
```
1  while(1) {
2    connfd = accept( sockfd,(struct sockaddr *)&adresse, &adrlaenge);
3    if( connfd < 0 ) {
4      if( errno == EINTR ) {
5        continue;
6      } else {
7        printf("error in accept() ...\n");
8        exit(EXIT_FAILURE);
9      }
10   }
11
12   printf ("connection established ... ready for data\n");
13
14   pthread_create(&th, NULL, threading_socket, (void *)&connfd);
15 }
16
17 void * threading_socket (void *arg) {
18   int clientfd = *((int *)arg);
19   pthread_detach (pthread_self ());
20
21   /* ready for client communication */
22
23   close (clientfd);
24   return NULL;
25 }
```

LDAP

FH University of Applied Sciences
TECHNIKUM WIEN

Motivation Directory Services

- Different services need similar information
  - names
  - telephone numbers
  - email addresses
  - account information (username, password)
  - …
- Redundant data storage increases probability of error
  - => central data storage
  - => accessible from different applications

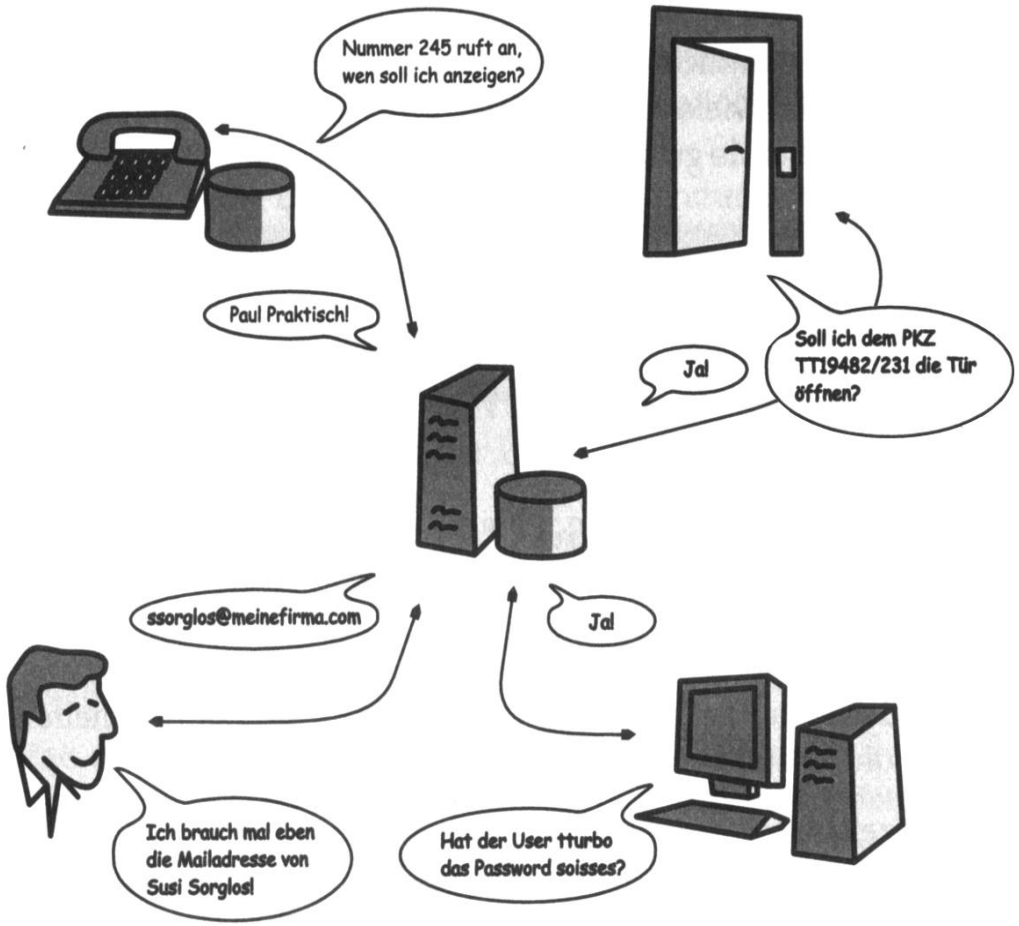


Motivation Directory Services

- **Advantages of a central data storage:**
  - **no redundancy in data**
  - **Single point of administration**
  - **Single account and password to remember**

in an enterprise environment for various applications like email, intranet, web services ,….

  - **prerequisite for single sign on systems**



Definition Directory Service

- **Directory or Name Service**
  - **maps the names of (network) resources to their respective addresses**
  - **shared information infrastructure for locating, managing, administering and organizing everyday items and network resources**
  - **specialized data store optimized for fast and simple lookup operations like a dictionary or a telephone book**

- **Directory Server**
  - **a server which provides such a service over a standardized interface**

- **Examples: DNS, NIS, X.500, LDAP, Active Directory**

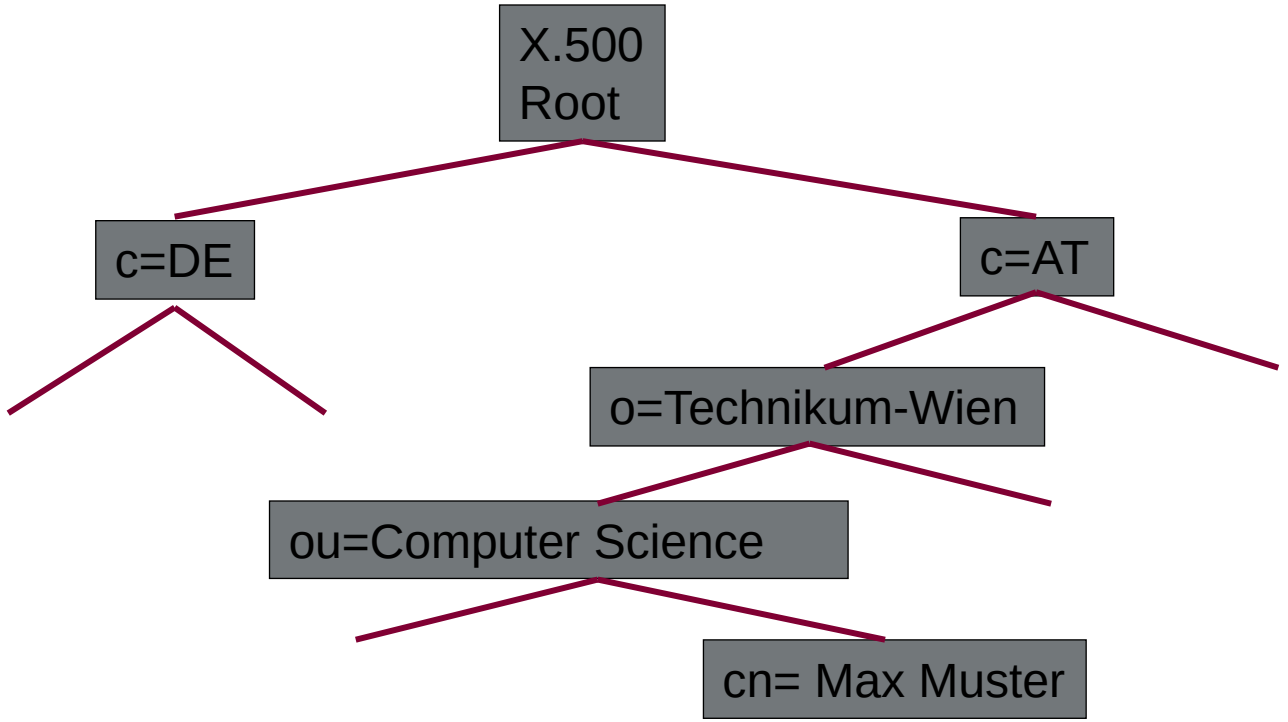
X.500

- Directory Access Protocol (DAP)
  - based on OSI model
  - open standard
  - hard to implement

- Distributed hierarchical database
  - similar to DNS, uses a global namespace
  - Directory Server Agents (DSA)

X.500 Namespace

- **Example**
  - **c = country, o = organization, ou = organizational unit, cn = common name**



LDAP

- **Lightweight Directory Access Protocol**
  - **developed at University of Michigan**
  - **originally planned as a frontend for X.500**
  - **uses the TCP/IP stack instead of the OSI Stack**
  - **Client-Server architecture**

- **LDAP Version 2, 1995, RFC 1777-1779**
  - **Internet Draft Standard**
  - **https://tools.ietf.org/html/rfc1777**

- **LDAP Version 3, 1997, RFC 4510-4519**
  - **https://tools.ietf.org/html/rfc4510**
  - **TLS(SSL) encryption**
  - **improved schema definitions and many more improvements…**
  - **only use v3 for security reasons!**
    - Needs to be enabled in clients since v2 still default for backwards compatibility!

LDAP

- Advantages of LDAP
  - Single Point of Administration
  - Single Sign On
  - Open standard (RFC)
  - platform independent
  - vendor support
  - standard APIs usable in every common programming language

LDAP Data model

➢Attribute

- Single name/value pairs
- Can be single or multi-valued e.g. attribute email can consist of more than one email address values
- Limited datatypes text or binary
- e.g. attribute name: surname, value: „Maier“

➢Entry

- some attributes combined to one entry similar to a database record
- e.g.: A Person entry with all attributes belonging to that person

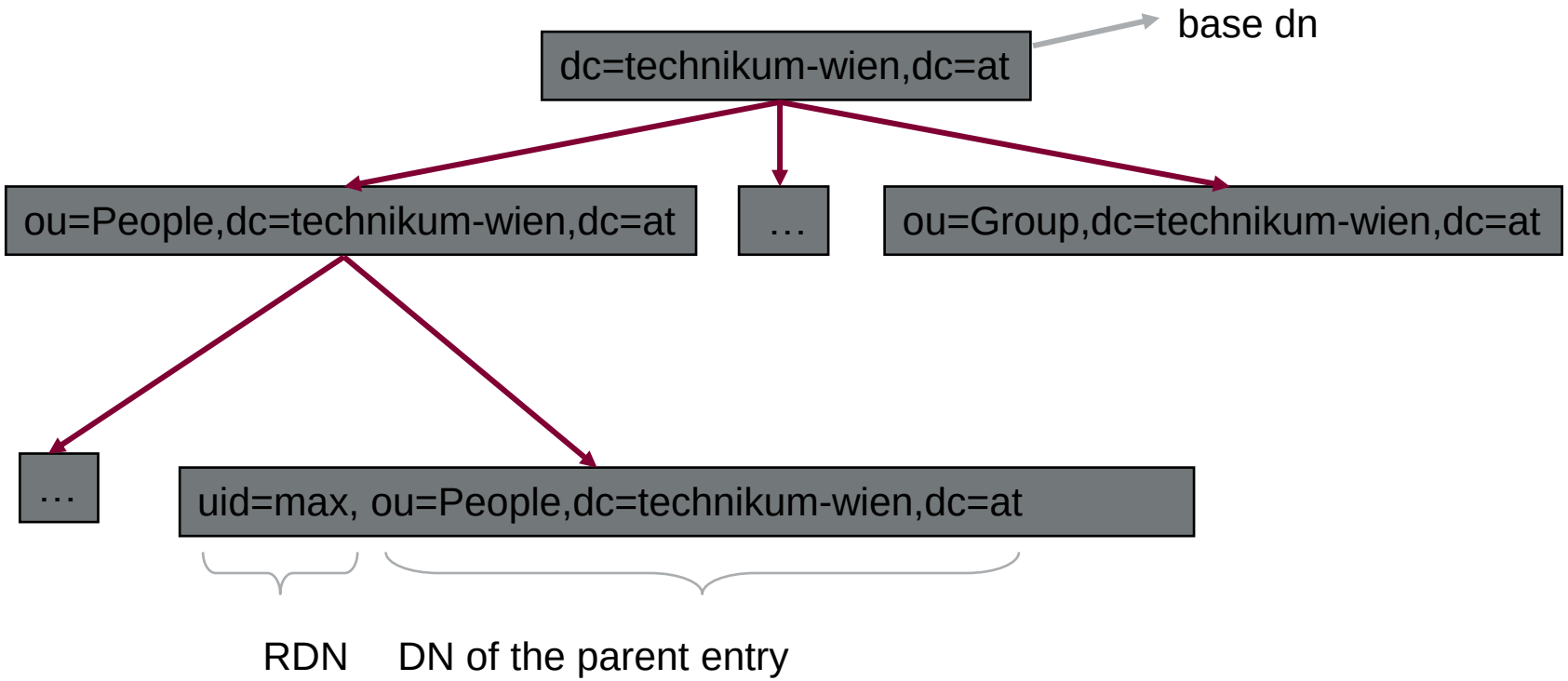
LDAP Data model

- **Namespace**
  - **hierarchical tree structure**
  - **can be organized in a similar way as the DNS namespace**
  - **LDAP attribute dc used for DNS domain components**
  - **RFC 2377**

  - **e.g. dc=technikum-wien,dc=at**

LDAP Data model

- **DNS namespace example**



LDAP Data model terminology

- **Distinguished Name (DN)**
  - identifies an entry unambigously
  - similar to a relational database primary key
  - consists of RDN and the DN of the parent entry
- **Relative Distinguished Name (RDN)**
  - Unique component of an entry (e.g. uid attribute)
- **Base DN**
  - root node of the directory
- **Directory Information Tree (DIT)**
  - tree structure of the directory

LDAP Data model

- **Object class**
  - **defines a set of attributes belonging to an entry**
  - **similar to a table in a relational database**
  - **can contain mandatory attributes (MUST) and optional attributes (MAY)**
  - **objectclass itself is a multi-valued attribute in an entry, so every entry can consist of one or more object classes**
  - **an object oriented model allows for inheritance of object classes**

  - **Example**
    - an object class person defines base attributes needed for persons like givenname, surname, …
    - The subclass inetorgperson extends person to allow for email attribute
    - A custom subclass mycompanyperson extends the possible attributes to allow also for optional attribute keycardID

LDAP Data model

- **Standard object classes**

top	Base class
person	Base class of persons
organization	Class for organizations
organizationalUnit	Class for organizational units
organizationalPerson	Person class for organizations
groupOfNames	Group object class
inetOrgPerson	RFC2798 internet person
dcObject	class for DNS components
alias	Alias Object, link to other DN
posixAccount	Account data for UNIX users

LDAP Data model

- **Standard attributes**

cn	common name
sn	surname
gn	givenname
o	name of the organization
ou	organizational unit
uniqueMember	member of a group
mail	email address
dc	DNS component
uid	user ID
userPassword	password

LDAP Data model

- LDAP schema
  - „blueprints“ of the LDAP server
  - Defines all available object classes and attributes with a standard naming convention based on IANA registered object identifiers (OIDs)

- A new entry consists of one or more object classed and needs to contain all MUST attributes of all used object classes and can also contain some of the available MAY attributes of those object classes
  - The LDAP server performs this so called schema-check for every ADD or MODIFY operations

LDAP Data model

- **Schema example objectclass**

```
# inetOrgPerson
# The inetOrgPerson represents people who are associated with an
# organization in some way.  It is a structural class and is derived
# from the organizationalPerson which is defined in X.521 [X521].
objectclass        ( 2.16.840.1.113730.3.2.2
    NAME 'inetOrgPerson'
         DESC 'RFC2798: Internet Organizational Person'
    SUP organizationalPerson
    STRUCTURAL
         MAY (
                  audio $ businessCategory $ carLicense $ departmentNumber $
                  displayName $ employeeNumber $ employeeType $ givenName $
                  homePhone $ homePostalAddress $ initials $ jpegPhoto $
                  labeledURI $ mail $ manager $ mobile $ o $ pager $
                  photo $ roomNumber $ secretary $ uid $ userCertificate $
                  x500uniqueIdentifier $ preferredLanguage $
                  userSMIMECertificate $ userPKCS12 )
         )
```

LDAP Data model

- **Schema example attribute**

```
# displayName
# When displaying an entry, especially within a one-line summary list, it
# is useful to be able to identify a name to be used.  Since other attri-
# bute types such as 'cn' are multivalued, an additional attribute type is
# needed.  Display name is defined for this purpose.
attributetype ( 2.16.840.1.113730.3.1.241
      NAME 'displayName'
      DESC 'RFC2798: preferred name to be used when displaying entries'
      EQUALITY caseIgnoreMatch
      SUBSTR caseIgnoreSubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
      SINGLE-VALUE )
```

Example of a schema check

- **A LDAP server has the following schema definitions stored:**

```
objectclass A
    MUST   a,b,c
    MAY    d,e,f
objectclass B
    MUST   a,d,x
    MAY    b,y
objectclass C SUP A
    MUST   d
    MAY    x,y,z
```

Legend: uppercase: object classes
lowercase: attributes
MUST: mandatory
MAY: optional
SUP: derived from

- **Which entries are valid, which are not?**

entry	implemented object classes	included attributes
1	A,B	a,b,d,f,e
2	C	a,b,c,d,e,f,x,y
3	B,C	a,b,c,d,x,y
4	A	a,c,b,g
5	A,B,C	a,x,d,c,b,y

LDIF

- LDAP Data Interchange Format (LDIF)
  - standardized data format for LDAP entries
  - Allows for data exchange with various clients and other LDAP servers
  - Plain-text storage of attributes and values
  - UTF-8 encoded (Unicode)
  - Base-64 encoding for binary data (photos, password hashes, …)
  - Historic reason to use this format instead of XMLor Json for example but mapping libraries exist

LDIF example

```
dn: uid=max,ou=People,dc=example,dc=com
objectclass: posixAccount
objectclass: inetOrgPerson
uid: max
loginshell: /bin/bash
uidnumber: 517
homedirectory: /home/max
cn: Max Mustermann
sn: Mustermann
givenname: Max
mail: max@example.com
gidnumber: 100
userpassword:: e2NyeXdnkrWfnjs*
```

Comparison LDAP-RDBMS

- Different Use Case
  - LDAP is optimized for fast and simple read and authorization requests
  - E.g. User account data is often searched but rarely modified
  - RDBMS are optimized for consistent and robust data manipulation (Inserts, Updates and Deletes) and implement complex transaction management
- Different data model
  - LDAP uses object-oriented hierachical model which makes it easy to modify and extend existing entries and add additional attribute
  - RDMBS have a rigid limited table design which is harder to alter after the initial database design
- Different query language
  - LDAP uses simple search filters instead of SQL

LDAP Protocol

- Client-Server protocol using TCP/IP
  - Standard Port 389 with STARTTLS or 636 with LDAPS (deprecated)

- Possible operations:

  - Binding
  - Searching
  - Compare Entries
  - Add Entry
  - Modify Entry
  - Remove Entry

LDAP Protocol

- Binding
  - Client needs to authenticate to the LDAP server

- 2 possible scenarios:
  - Anonymous binding (blocked by some servers)
  - Bind as a specific LDAP user with DN and credentials

- Access Control dependent on binding
  - Different privileges for different users

LDAP Protocol

- **Searching**
  - **Search operations by providing search filters**

  - **(cn=Max Mustermann)**
  - **(&(sn=Maier)(ou=Sales))**
  - **(&(givenname=Max*)(|(ou=Sales)(ou=Engineering)))**
  - **logical operators in prefix notation: & AND, | OR, ! NOT**
  - **wildcard operator: ***

- **Server responds with: DN + attributes of matching entries**

LDAP Protocol

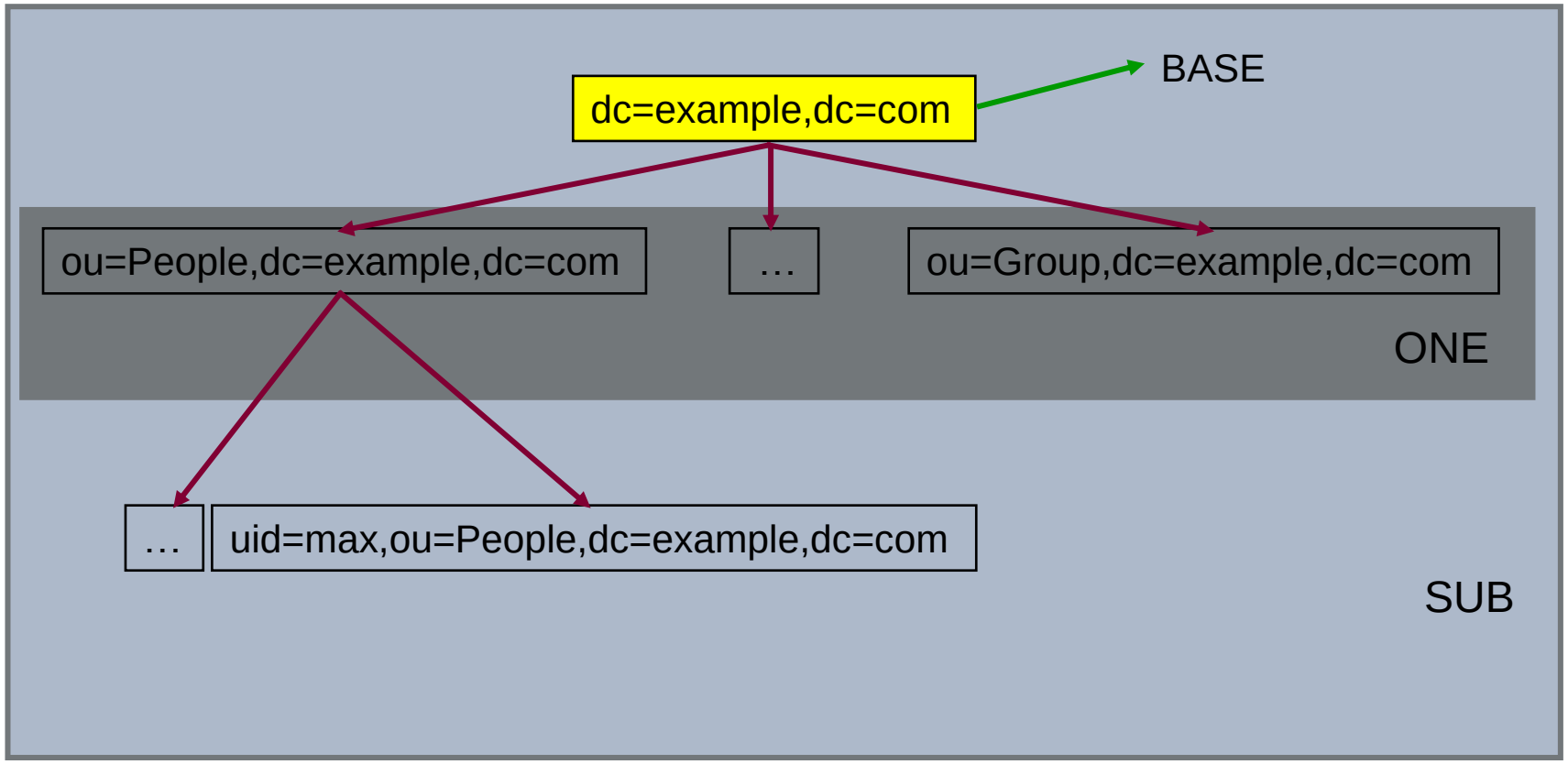
- **Search Base**
  - **Starting point of the search in the LDAP tree**
  - **defaults to Base DN (root node of the LDAP tree)**

- **Search Scope**
  - **Limits the search**
  - **3 possible categories:**

SUB	complete tree below the Search Base (recursive search, Default)
ONE	only one level below the Search Base
BASE	search Base only

LDAP Protocol

- **Search Scope Example**



LDAP Protocol

- **Compare Entries**
  - **compare attributes, rarely used**
- **Add Entry**
  - **add a new entry**
  - **provide object class(es) and attribute name/value pairs as LDIF**
- **Modify Entry**
  - **add new or modify existing attributes of an entry**
- **Remove Entry**
  - **delete an entry by providing its DN**

LDAP Servers

- University of Michigan LDAP Server
  - First implementation of the LDAP protocol
  - Open Source reference implementation
  - available for UNIX systems

- Commercial servers
  - Microsoft Active Directory
  - Novell eDirectory
  - Oracle (Oracle Internet Directory)
  - Red Hat 389 Directory Server

LDAP Servers

- **OpenLDAP**
  - **Open Source project**
  - **Developed based on the University of Michigan servers**
  - **Supports all Unix and Linux OS and Mac OS X**
  - **http://www.openldap.org**

  - **Historic release OpenLDAP 2.3**
    - Functional enhancements and improved scalability
    - Configuration Backend (cn=config)
    - Password Policy Overlay
    - Sync Provider Overlay
    - Delta-syncrepl support

  - **Current release**
    - OpenLDAP 2.6 (first released October 2021)
    - MirrorMode and MultiMaster replication
    - Proxy Sync replication
    - Expanded monitoring
    - Multiple new Overlays
    - LDAP transaction support
    - Load Balancer deamon

OpenLDAP

- **Components**
  - **Server Daemon slapd**
  - **Client Tools ldapsearch, ldapadd, ldapmodify, ldapdelete,…**
  - **APIs for C/C++**

- **Installation: 2 possibilities**
  - **Preferred way: Precompiled packages of your Linux distribution**
    - Available packages for server, client-tools und libraries
    - e.g. for Ubuntu: libldap-common, libldap-2.4.2, libldap2-dev, ldap-utils, slapd
  - **Or: compile from sources**
    - Relative complex, a lot of dependencies on other libraries

OpenLDAP Client Tools

- **ldapsearch**
  - **Client for LDAP search queries**
  - **Syntax:**

```
ldapsearch [Options] filter [Attributes]
```

filter … LDAP search filter (Default: objectclass=*)

  - **Returns all matching entries with all attributes**
  - **Possible to limit output of attributes with optional [Attributes]**

OpenLDAP Client Tools

- **ldapsearch most used parameters**

-D binddn	Authenticate as binddn (Default: anonymous)
-W	Ask for credentials
-w bindpw	password for binddn
-x	Simple Authentication (without SASL)
-L	Output as LDIF format with additional info
-LL	Output as LDIF without comments
-LLL	Default LDIF output without version
-h ldaphost	alternative LDAP Server
-p ldapport	alternative LDAP Port
-b searchbase	alternative Base DN for searching
-S attribute	Sorted output
-s scope	Search Scope: sub (Default), one, base
-ZZ	enforce STARTTLS encryption

OpenLDAP Client Tools

- ldapsearch examples

```
ldapsearch -x -LLL "(uid=dummy)" cn sn mail
```

```
ldapsearch -x -LLL "(&(sn=A*)(gidNumber=102))"
```

```
ldapsearch -x -LLL -D "uid=testuser,ou=People,dc=technikum-
wien,dc=at" -W "uid=testuser"
```

```
ldapsearch -x -LLL -s one "(ou=*)"
```

LDAP Linux Client Authentication

- **Network Information Services History**

  - **Local files**
    - /etc/passwd, /etc/shadow, /etc/group, /etc/hosts …
    - Only for single workstations
    - no synchronization

  - **NIS**
    - Client/Server architecture
    - distribution of local files
    - transparent for clients
    - limited scalabilty
    - no security mechanisms
    - limited extendability

LDAP Linux Client Authentication

- LDAP as recommended central Network Information System
  - open standard
  - centralized database
  - extendable
  - scalable
  - secure with SSL/TLS encryption

LDAP Linux Authentication

- **Schema mapping NIS -> LDAP**
  - **nis.schema (RFC 2307)**
  - **Mapping from local files to LDAP object classes**
    - passwd <-> posixAccount
    - shadow <-> shadowAccount
    - group <-> posixGroup

LDAP Linux Authentication

- **Example – Linux user account**
  - **/etc/passwd entry**

```
maxm:x:500:100:Max Muster:/home/maxm:/bin/bash
```

  - **/etc/shadow entry**

```
maxm:Password hash value:11858:0:99999:7:::
```

  - **mapped LDAP entry**

```
dn: uid=maxm,ou=People,dc=example,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
uid: maxm
cn: Max Muster
uidNumber: 500
gidNumber: 100
homeDirectory: /home/maxm
loginShell: /bin/bash
userPassword:: Password hash value base64 encoded
shadowLastChange: 11858
shadowMax: 99999
shadowWarning: 7
```

LDAP Linux Authentication

- **Example – Linux group**
  - **/etc/group entry**

```
admins:x:120:usera,userb,userc
```

  - **mapped LDAP entry**

```
dn: cn=admins,ou=Group,dc=example,dc=com
objectClass: top
objectClass: posixGroup
cn: admins
gidNumber: 120
memberUid: usera
memberUid: userb
memberUid: userc
```

LDAP Linux Authentication

- **Workflow and used libraries**
  - **Linux authentication is highly modular and configurable**
  - **Pluggable Authentication Module (PAM)**
    - integration of various auth providers into system services as login, passwd, su, ftp, ssh …
    - standard framework used in Linux und Solaris
  - **Name Service Switch (NSS)**
    - provides mapping and resolution of names <-> numerical values (e.g. uid <-> numerical uidnumber)
    - Standard API
    - needs access to passwd, group, hosts, aliases, … files
    - e.g. every ls –l command needs to resolve the usernames from the internal uidnumbers stored in the inode
    - configuration file /etc/nsswitch.conf where different name service providers can be defined

LDAP Linux Authentication

- **PAM module**
  - **Standard Library Calls**
  - **Auth mechanism independent of applications**
  - **PAM Module implemented as shared libraries**
  - **Configurations files in /etc/pam.d**
    - one file for each service (login,ssh,ftp,…)
  - **4 Module types**
    - auth – Login/password query
    - account – checks system ressources
    - session – session handling
    - password – changing passwords
  - **modules are stackable**

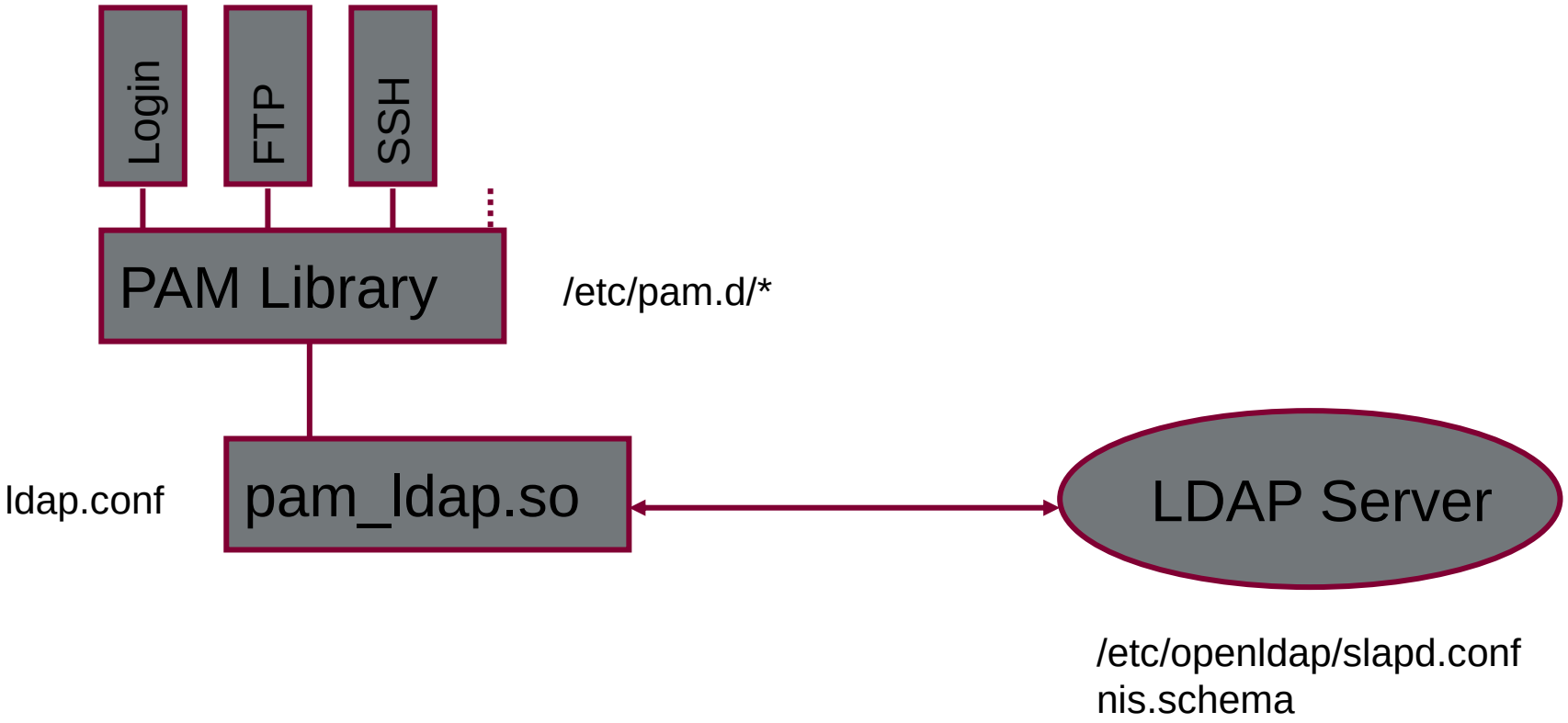
LDAP Linux Authentication

- **NSCD Service**
  - **NSS requests can be cached**
    - NSCD (Name Service Cache Daemon)
    - increases performance of lookups
  - **Configuration file /etc/nscd.conf**

- **Different client integrations available**
  - **NSLCD**
    - daemon for NSS and PAM lookups using LDAP (nss-pam-ldapd)
    - configuration file /etc/nslcd.conf
  - **SSSD**
    - system Security Services Daemon
    - can provide caching and offline support to the system
    - provides PAM and NSS modules
    - configuration file /etc/sssd/sssd.conf
    - need to disable NSCD caching

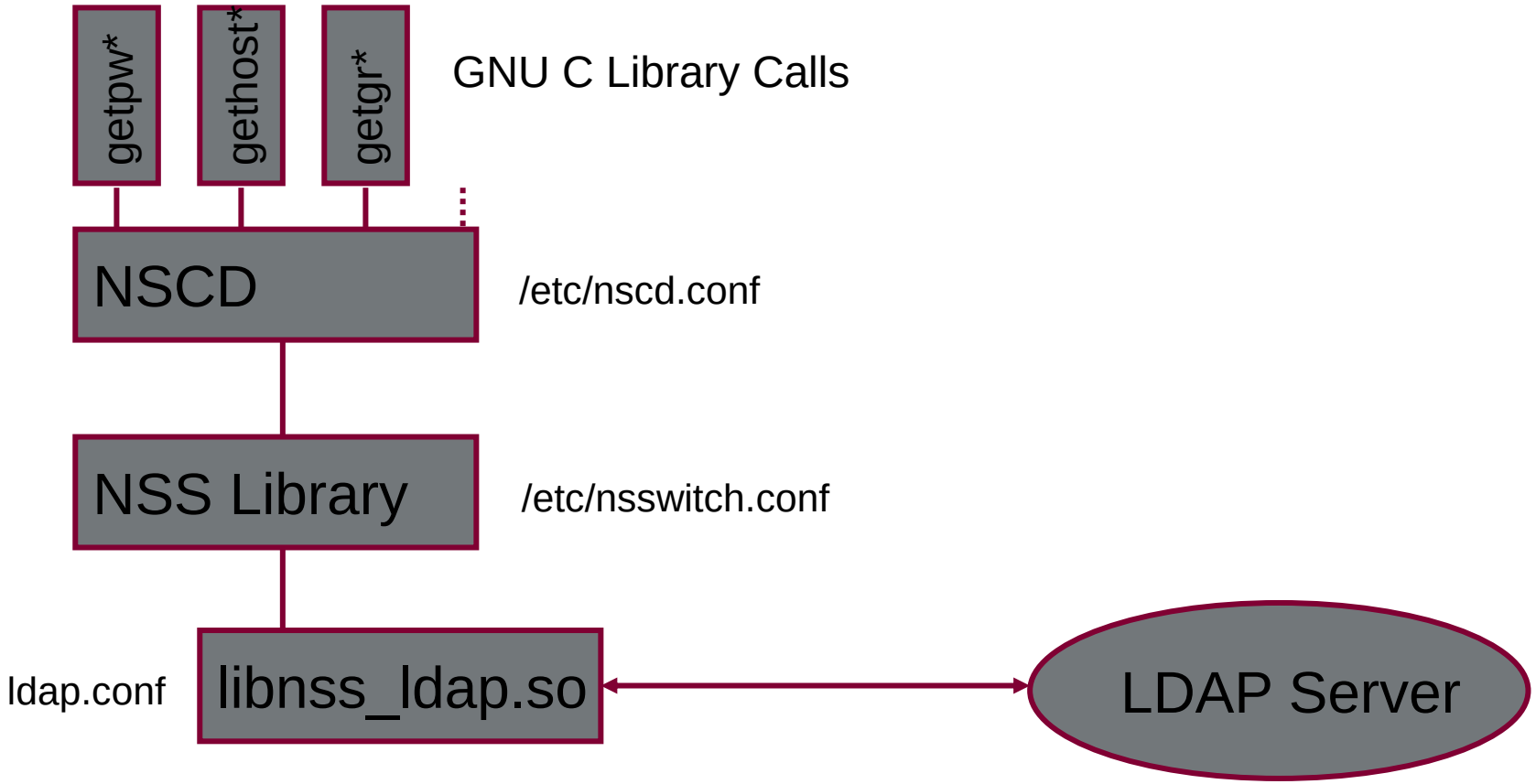
LDAP Linux Authentication

- **PAM LDAP layout**



LDAP Linux Authentication

- **NSS LDAP layout**



LDAP Linux Authentication

- **nss_ldap – configuration**
  - **Modification of /etc/nsswitch.conf**

```
passwd: files ldap
shadow: files ldap
group: files ldap
```

  - **Lookup of local users first, if not found then try LDAP**
  - **Recommendation:**
    - use local system accounts (root, daemon accounts)
    - only user accounts in LDAP

Bibliography

- **LDAP RFCs http://www.ietf.org/rfc**
- **Implementing LDAP, Mark Wilcox. Wrox Press, 1. Edition, 1999**
- **LDAP System Administration, Chris Carter, O‘Reilly, 2003**
- **http://www.openldap.org**
  - **Infos, Administrator Guide, FAQ**
- **LDAP-Howto**
  - http://www.tldp.org/HOWTO/LDAP-HOWTO.html
  - http://www.yolinux.com/TUTORIALS/LinuxTutorialLDAP-SoftwareDevelopment.html

FH University of Applied Sciences TECHNIKUM WIEN

Replication

Introduction

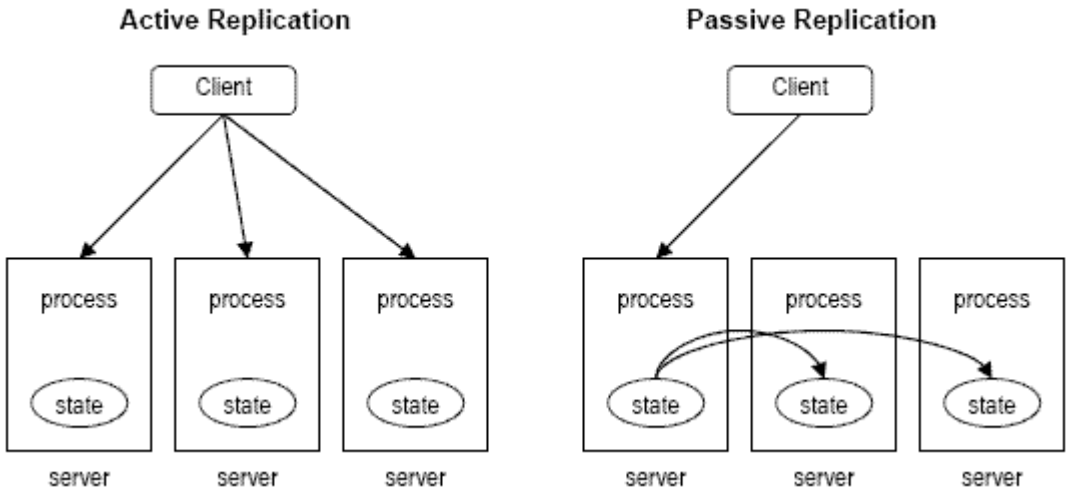
- Why Replication
  - Enhance reliability
  - Improve performance

- Two issues
  - Placement of replicas & how updates are propagated
  - How replicas are kept consistent

- References
  - https://www.distributed-systems.net/index.php/books/ds3/

Active vs Passive

- Active replication
  - This approach makes each server available for client queries and says that each client request is processed by every server, so their states should be the same.
- Passive replication
  - In this approach the servers have a leader, the server that actually processes the client requests, and multiple backup servers. On each request the leader updates the state of the other servers.



Content Distribution

- Consider only a client-server combination
  - Propagate only notification/invalidation of update (often used for caches)
  - Transfer data from one copy to another (distributed databases): passive replication
  - Propagate the update operation to other copies: active replication

- Note
  - No single approach is the best, but depends highly on available bandwidth and read-to-write ratio at replicas.

Replication Strategies

- Pull-based replication
  - periodically polls the provider for updates
- Push-based replication
  - consumer listens for updates that are sent in real time

- Pros/Cons
  - Issue: Time for updating, polling interval in pull-based replication?
  - Issue: list of client caches, state at server and need for persistent connections in push-based replication
  - Memory footprint, used bandwith
  - Consider resource usage (e.g. battery life time for mobile devices)

Replication Strategies

- We can dynamically switch between pulling and pushing using leases: A contract in which the server promises to push updates to the client until the lease expires.
- Make lease expiration time dependent on system's behavior (adaptive leases)
  - Age-based leases: An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
  - Renewal-frequency based leases: The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
  - State-based leases: The more loaded a server is, the shorter the expiration times become

Consistency Protocols

- describe a specific implementation of a consistency model

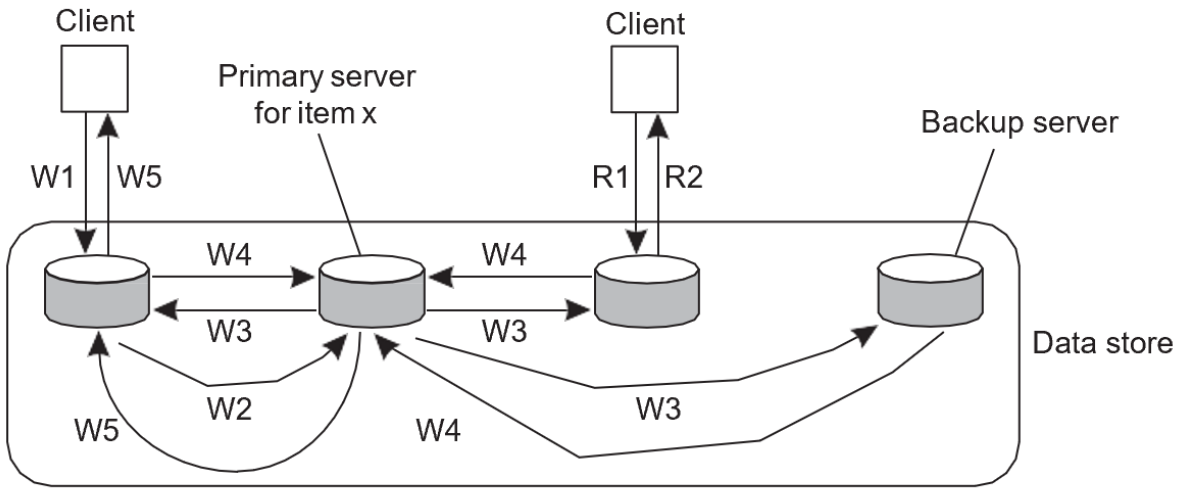
- Primary based protocols
  - Each data item is associated with a primary.
  - The primary is responsible for coordinating writes to the data item.

- Replicated write protocols
  - With these protocols, writes can be carried out at multiple replica, instead of only one

Primary based protocols

- Example primary-backup protocol
  - Traditionally applied in distributed databases and file systems that require a high degree of fault tolerance. Replicas are often placed on same LAN.

Primary-backup protocol

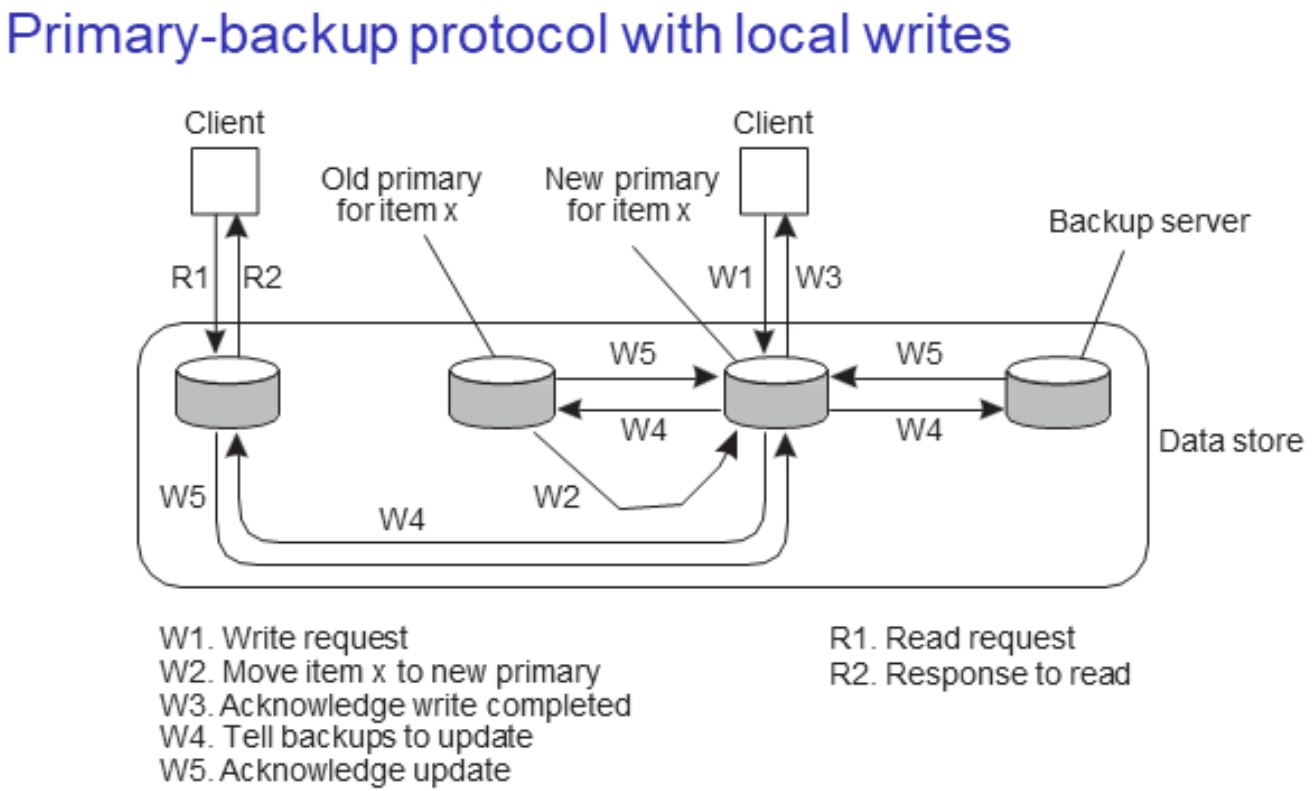


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

Primary based protocols

- Example primary-backup protocol with local writes
  - Mobile computing in disconnected mode (ship all relevant files to user before disconnecting, and update later on).



Replicated write protocols

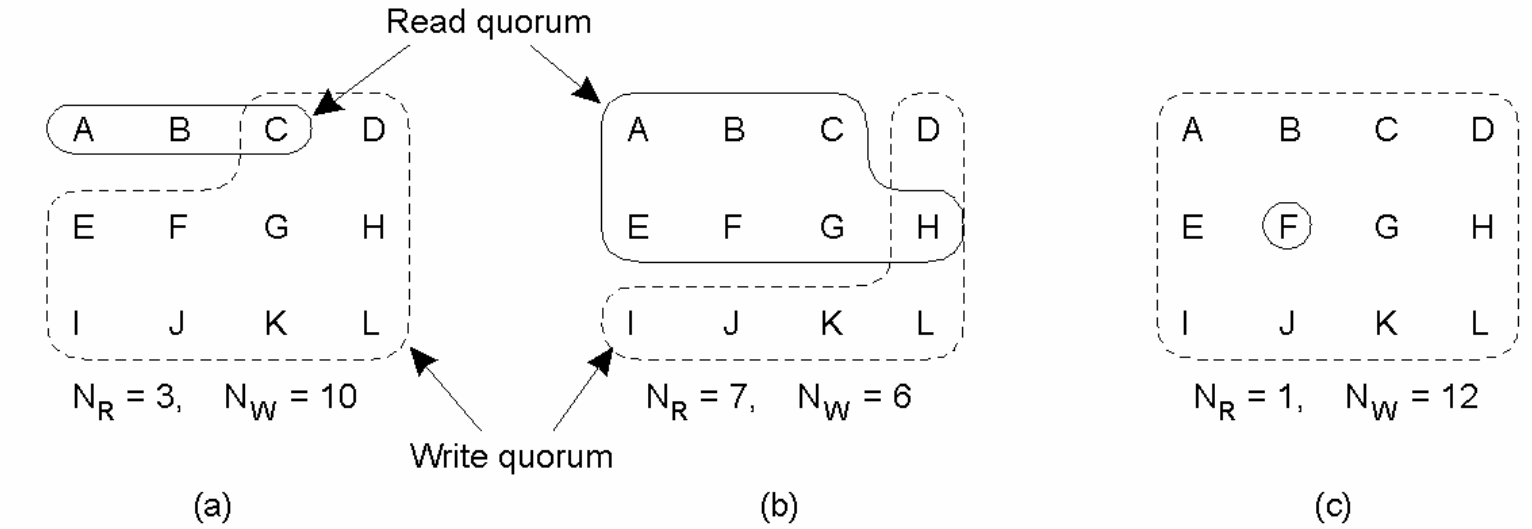
- Quorum-based protocols
  - Ensure that each operation is carried out in such a way that a majority vote is established
  - Clients must acquire permissions from multiple replicas before reading/writing a replicated data item.

- Example:
  - To update a file, a process must get approval from a majority of the replicas (> N/2)
  - These replicas also need to agree to perform the write.
  - To read, a process also contacts a majority of the replicas (> N/2)

- Generalization:
  - distinguish read quorum (N_R) and write quorum (N_W):
  - Must obey the following two rules

    $N_R + N_W > N$: avoid read-write conflicts
    $N_W > N/2$: avoid write-write conflicts

Quorum based protocols



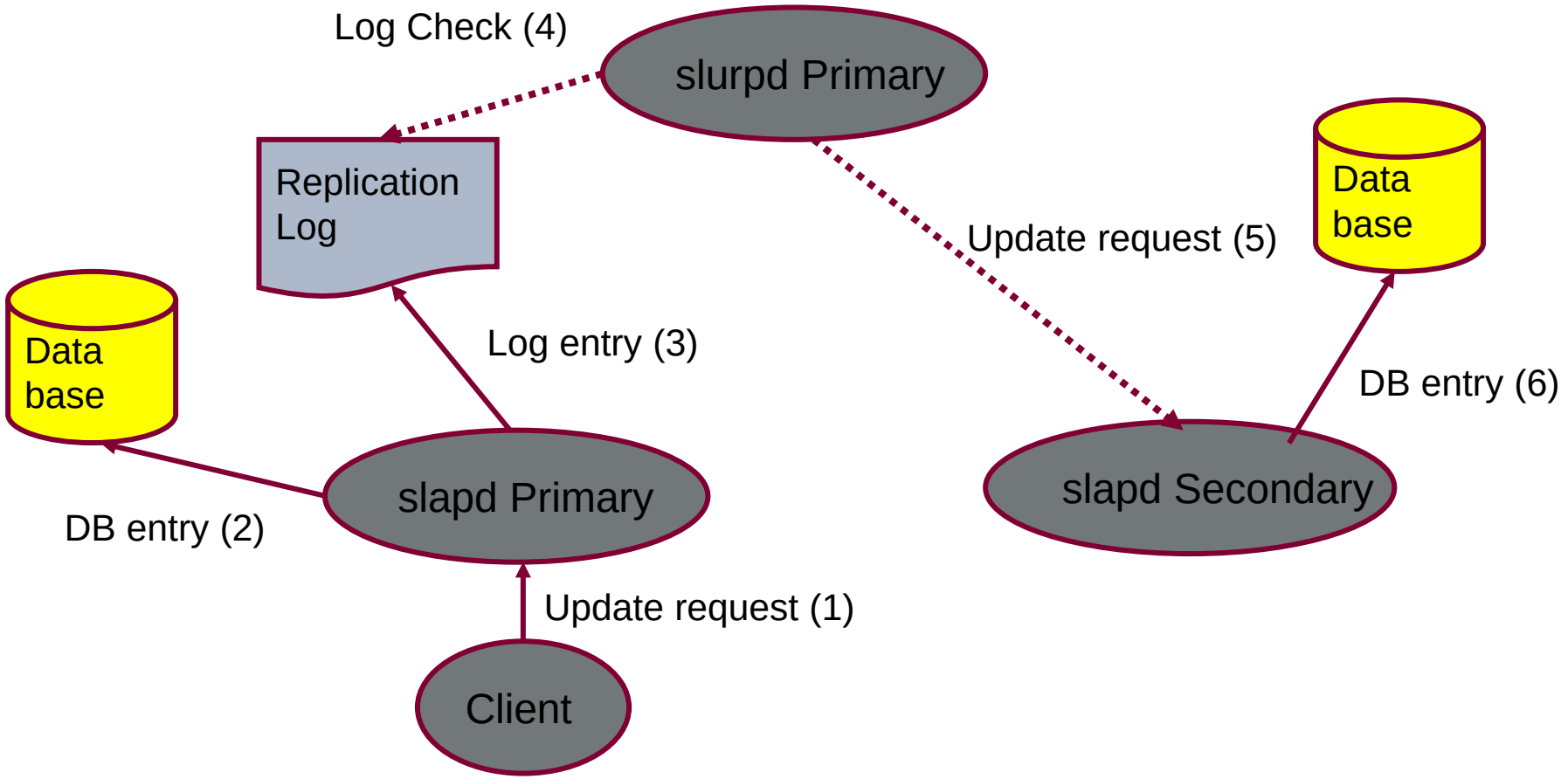
- Example (a): A correct choice of $N_R = 3$ and $N_W = 10$
  - most recent write quorum consisted of the 10 servers C-L. All of these get the new version number. Any subsequent read quorum of three servers (A-C in this example) will have to contain at least one member of this set and client will get the most recent update.
- Example (b): write-write conflict may occur because $N_W <= N/2$
  - e.g. one client chooses {A,B,C,E,F,G}, another client chooses {D,H,I,J,K, L} as its write set, the two updates will both be accepted without detecting the conflict.
- Example (c): A correct choice of $N_R = 1$
  - possible to read a replicated file by finding any copy and using it.
  - good read performance but write updates need to acquire all copies (N_W=N).
  - This scheme is generally referred to as Read-One, Write-All, (ROWA).

Example: OpenLDAP replication

- Distributing the LDAP data to different servers
  - increase availability
  - load balancing
  - improve fault tolerance
- Primary/Secondary Topology
  - one primary server (provider)
  - one or more secondary servers (consumer)
  - replicating data from provider to consumer servers
- Implementations
  - slurpd daemon (push based) but deprecated
  - syncrepl (from OpenLDAP 2.3, push and/or pull based)
  - multi-master replication (not recommended)

OpenLDAP replication

- SLURPD scenario (deprecated)

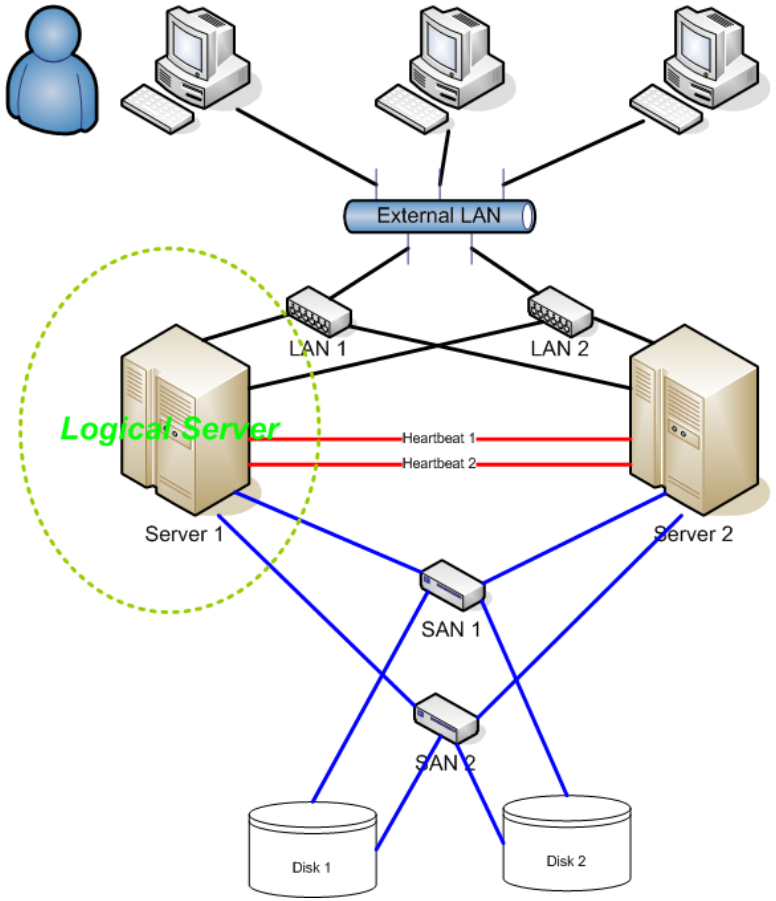


OpenLDAP Syncrepl

- LDAP Sync Replication engine
  - Consumer-side replication engine
  - uses the LDAP Content Synchronization protocol RFC4533
  - provides a stateful replication
  - supports both pull-based and push-based synchronization
  - tracks status of the replication content by maintaining and exchanging synchronization cookies as contextCSN (change sequence number)
  - Deployment strategies
    - Syncrepl
    - Delta-syncrepl
    - Syncrepl Proxy Mode
    - MirrorMode
    - N-Way Multi-Master
  - See details in OpenLDAP Replication strategies presentation

High Availability (HA) Cluster

- are groups of nodes that support server applications that can be reliably utilized with a minimum amount of down-time.
- HA clusters detect hardware/software faults, and immediately restart the application on another system without requiring administrative intervention (**failover**).
- HA clusters usually use a **heartbeat** private network connection which is used to monitor the health and status of each node in the cluster.



[https://en.wikipedia.org/wiki/High-availability_cluster]

HA Node Configurations [https://en.wikipedia.org/wiki/High-availability_cluster]

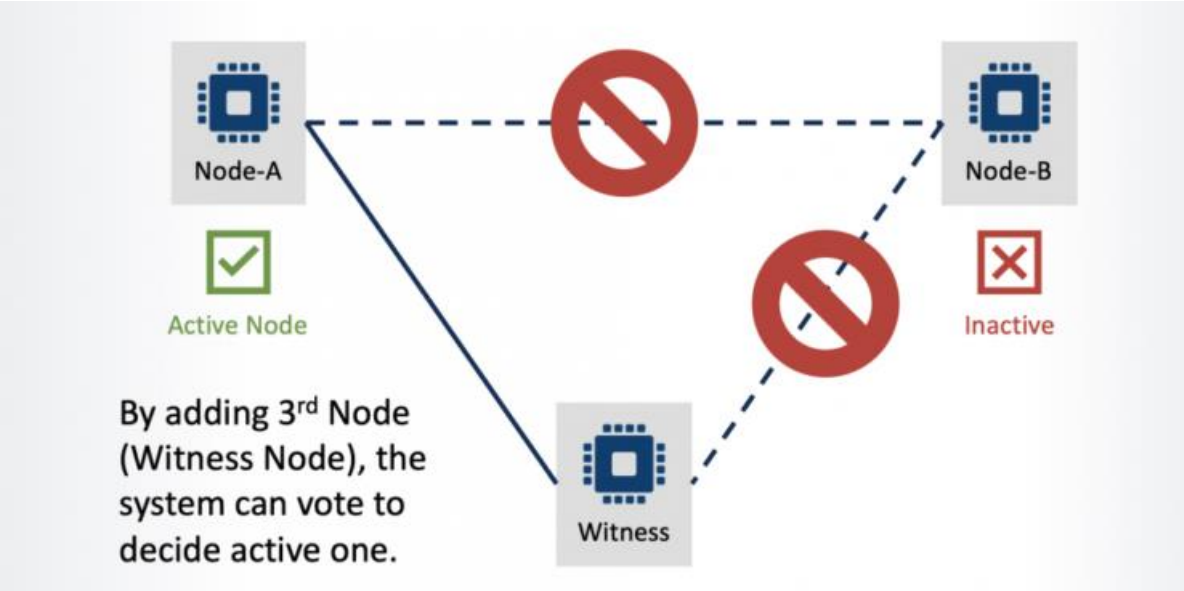
- Active/active
  - Traffic intended for the failed node is either passed onto an existing node or load balanced across the remaining nodes.
- Active/passive
  - Provides a fully redundant instance of each node, which is only brought online when its associated primary node fails.
- N+1
  - Provides a single extra node that is brought online to take over the role of the node that has failed. In the case of heterogeneous software configuration on each primary node, the extra node must be universally capable of assuming any of the roles of the primary nodes it is responsible for.
- N+M
  - In cases where a single cluster is managing many services, having only one dedicated failover node might not offer sufficient redundancy. In such cases, more than one (M) standby servers are included and available. The number of standby servers is a tradeoff between cost and reliability requirements.

Split-Brain

- occurs when all of the private links go down simultaneously, but the cluster nodes are still running
- each node in the cluster may mistakenly decide that every other node has gone down and believes it is the only one running
- Nodes attempt to start services that other nodes are still running
- duplicate instances may lead to data corruption or other data inconsistencies that might require manual intervention and cleanup.

Split-Brain

- HA clusters often use quorum witness storage (local or cloud) to avoid this scenario.
  - A witness device cannot be shared between two halves of a split cluster
  - When all cluster members cannot communicate with each other (e.g., failed heartbeat), if a member cannot access the witness, it cannot become active.



Time Synchronization
NTP

Motivation

Motivation

- **Why do you need to synchronize clocks/time in distributed systems?**
- **Possible scenarios where unsynchronized clocks might cause issues:**
  - **Timestamps of files in distributed file systems**
    - Distributed software development
    - Backups and archives
  - **Reproducibility of network attacks**
    - Chronological order of entries in different log files
  - **Performance-measurements within the network**
    - Tracing a request trough multiple servers/services
  - **Request caching**
    - Timestamp Last-Modified: set by the server, evaluated by the client
    - Timestamp Expires: set by the server, evaluated by the client

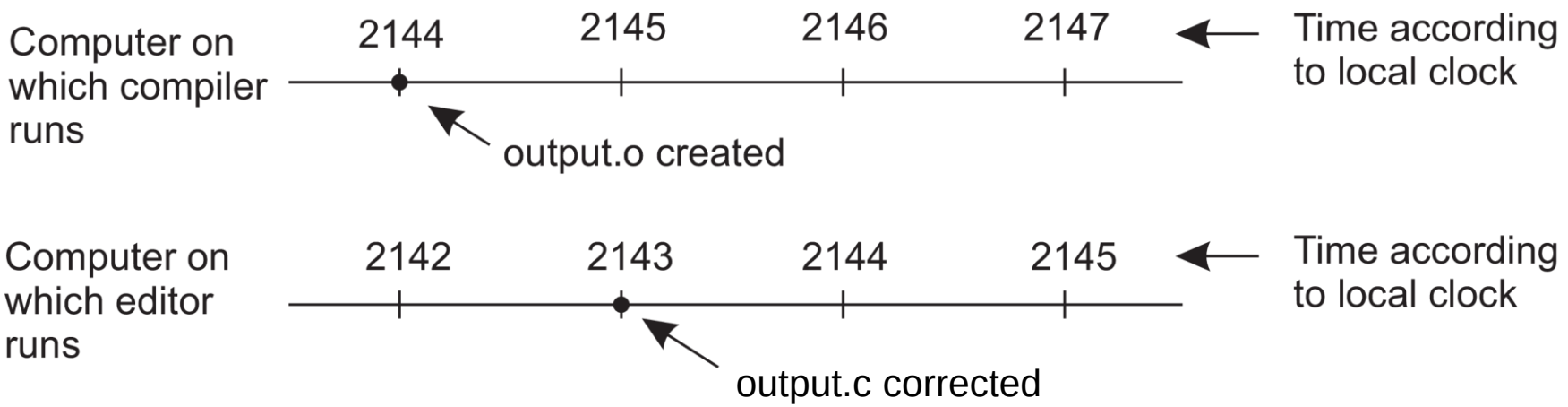
Motivation

- No two physical clocks are completely identical
  - Delayed initialization (constant offset)
  - Diverging clock speed (frequency error)
  - Different environmental influences (e.g., age of clock components, temperature, …)

- Without regular clock synchronizations, the error may grow over time!

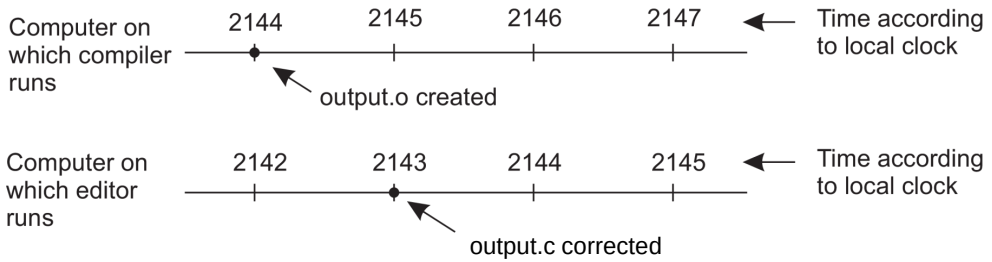
Motivation

- **Distributed systems require time as a means to express order, e.g.:**
  - **Monitoring**
  - **Log-file analysis**
  - **Distributed development**
- **Example: Compilation with `make`**
  - **`output.c` is edited on one computer, but compiled on another computer**
  - **These two computers do not have a synchronized clock**



Motivation

- **Example: Compilation with `make`**



- **make only rebuilds source files, if they have been created/changed since the last build**
- **In this example, `output.c` is modified to correct an error in the previous build and the modification time is set to 2143**
- **make on the other computer does not rebuild `output.c` to `output.o`, because it has timestamped (the actually outdated) `output.o` with 2144 due to a clock mismatch**

Unit of time: Second

- The second (s) is the base unit of time in the International System of Units (SI, from French *Système International d'unités*)

- Definition of a second until 1956
  - A second is 1/86400 of a mean solar day
  - However, the length of a mean solar day is not constant, as the period of the earth's rotation is not constant
  - Around 300 million years ago, a year had 400 days. Furthermore, there exist other factors that may lead to variations in the earth's rotation speed

Unit of time: Second

- **Definition of a second from 1956 to 1967**
  - **A second is 1/31,556,925.9747 of the tropical year in 1900**
  - **It is called the "ephemeris second" (ephemeris = itinerary of the trajectory of an astronomical object)**
  - **It is about $3*10^{-8}$ s shorter than today's universal time second.**

- **Definition of a second since 1967**
  - **Defined by the *Bureai International de l'Heure* (BIH)**
  - **A second is the time duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the fundamental unperturbed ground-state of the caesium-133 atom**

Time Systems

- GMT: Greenwich Mean Time
  - Mean solar time at the zero meridian
  - Used for deriving 24 time zones
  - About every 15° it changes by 1 hour, considering political and geographic borders

- TAI: Temps Atomique International
  - Based on the BIH definition of a second
  - A TAI-day is the mean of a day of about 250 caesium-133 clocks around the world
  - Does not correspond with astronomic circumstances
  - Currently 3 ms shorter than a mean solar day, and thus would cause a drift over time, rendering it unusable for public life

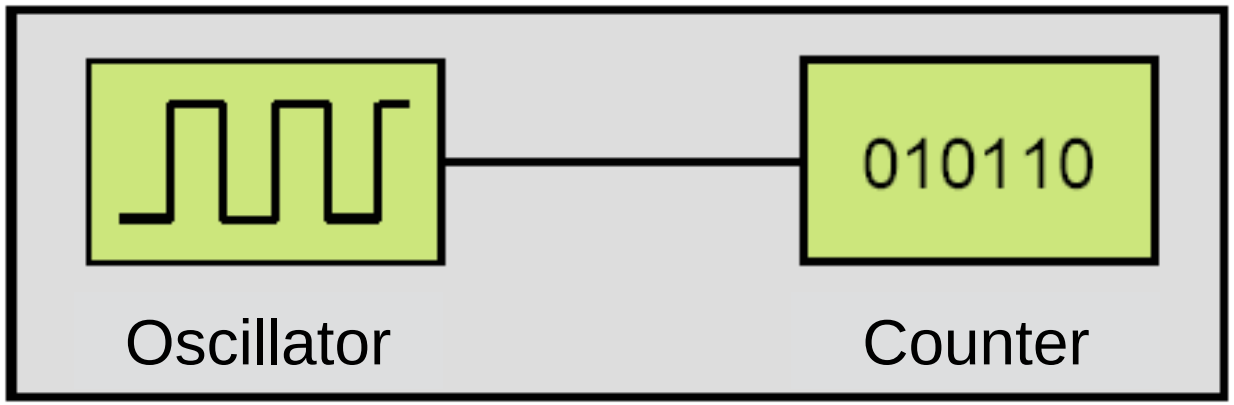
Time Systems

- **UT: Universal Time**
  - **Derived from astronomic sidereal time (= mean solar time at Greenwich meridian)**
  - **UT1 applies additional corrections for polar motion**

- **UTC: Universal Time Coordinated**
  - **Based on TAI, uses the same scale**
  - **Whenever the difference to UT1 is bigger than 900 ms, a leap second is added**
  - **Defined by the *International Earth Rotation and Reference Systems Service (IERS, https://www.iers.org/)***

Time Measurement Basics

- **Terms**
  - **Instant of time: A moment/point on a time scale, can be used for ordering events**
  - **Time interval: The difference between two instants of time**
  - **Oscillator: A generator that oscillates at a defined frequency**
  - **Clock: A device for measuring time. Allows for measuring time intervals, can also provide instants of time, if a reference point is given**



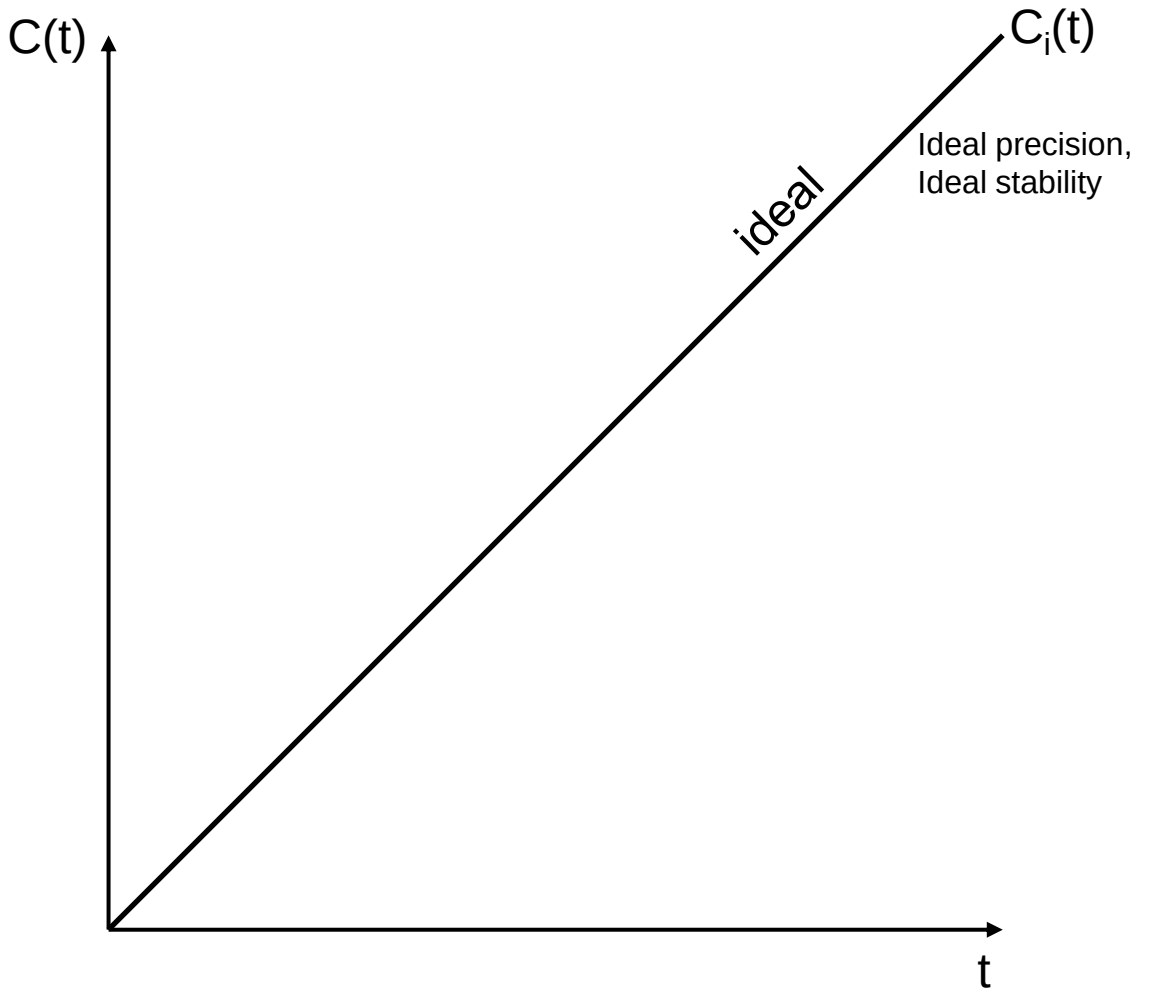
Time Measurement Basics

- **Measurement**
  - **Accuracy: Deviation of a clock from a reference time**
  - **Precision: Degree of time resolution**
  - **Stability: Measure for frequency fluctuations of a clock**
  - **Offset: Time difference between two clocks**
  - **Drift: Frequency deviation of two clocks**

Time Accuracy

- **Time plot**
  - t … reference time (e.g., UTC)
  - $C_p(t)$ … value of clock p at time t
  - ρ (Rho) … maximum accepted drift rate

- **Ideal clock $C_i(t)$**
  - Ideal precision
  - Ideal stability
  - i.e., no offset
  - $C_p(t) = t \; \forall \; p, t$ or $\frac{dC}{dt} = 1$

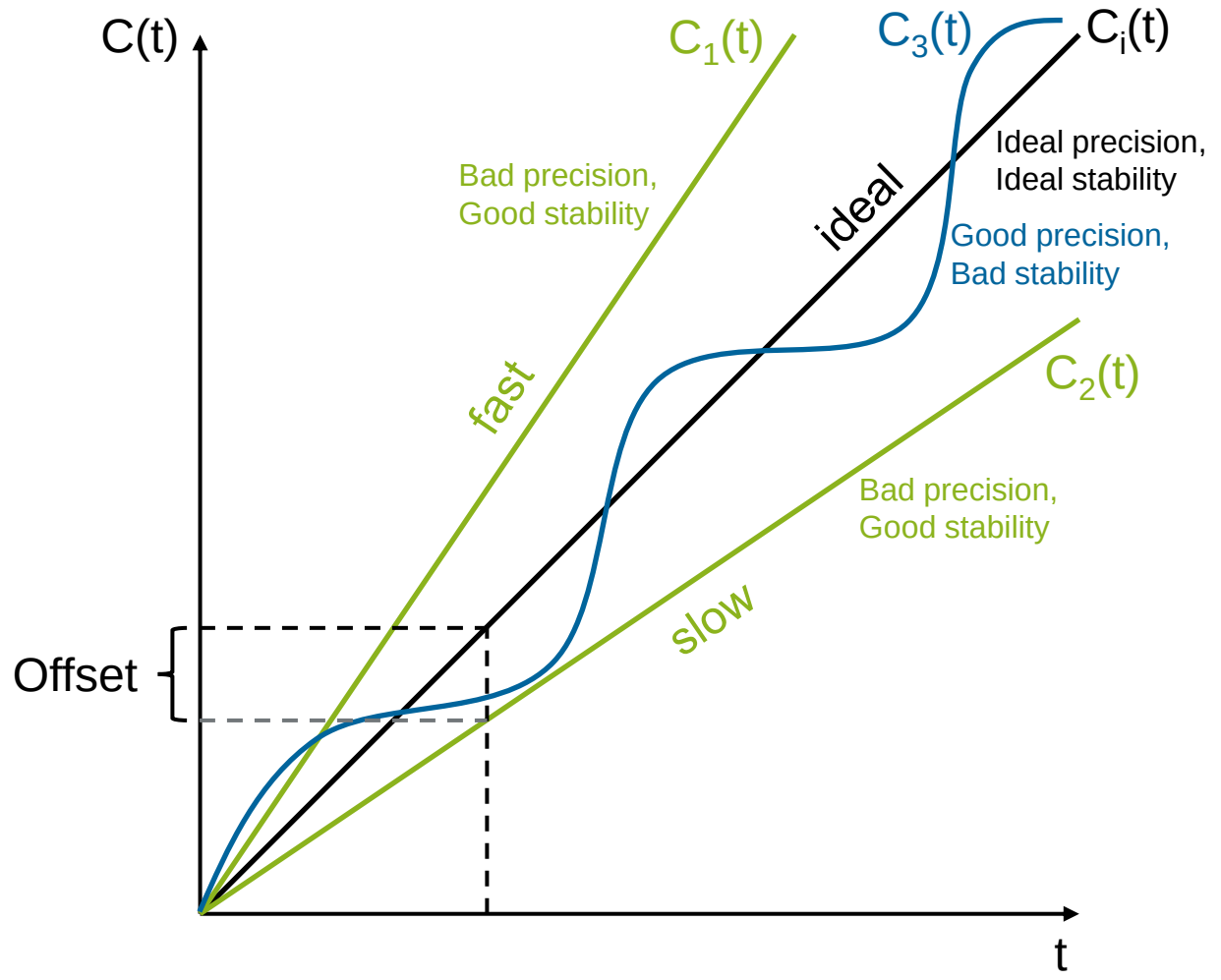


Time Accuracy

- **Real clock**
  - **Fluctuating precision**
  - **Fluctuating stability**
  - **i.e., fluctuating offset regarding $C_i(t)$**
  - **If there exists a constant ρ such that**

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$$

then the timer works within specification



Computer Clocks

- **Classic UNIX internal time representation**
  - **Computer counts seconds since January 1st 1970 (UNIX time)**
  - **Uses 32-bit values for most operations**
  - **Allows for calculating time until the year 2038**
  - **Allows for calculating time differences of 68 years max.**
  - **See also "Year 2038 problem": https://en.wikipedia.org/wiki/Year_2038_problem**
  - **The format `hh:mm:ss` is just a user-friendly way of representing the actual UNIX time**
  - **There are utility function for converting to this format (e.g., by using the `timezone` library)**

Clock Synchronization

- How to compensate for drifting clocks?

- Possible approach:
  - Periodically reset the clock to the reference time
  - Not a (huge) problem, when the clock is slow (behind the reference time)
  - A huge problem if the clock is fast (ahead of the reference time)
  - Not practical for application which require a highly precise clock

Clock Synchronization

- **Why is it a problem, if the clock is fast?**
  - **Time will be reset to a point in the “past”**
  - **The same instant of time might happen twice, i.e., the assumption of time monotonicity is violated**
  - **The correct order of events (e.g., syslog) cannot be guaranteed anymore**

- **Solution:**
  - **Slow down or accelerate the clock frequency to converge to the reference time**

Clock Synchronization in Distributed Systems

- Issues caused by networking
  - Non-deterministic travel-time of packets in IP-networks
  - Variable delays in router- and switch-queues
  - Asymmetric routing
  - Packet loss caused by queue overflows or bit-errors

- Issues caused by location
  - Computers might be located in different time zones

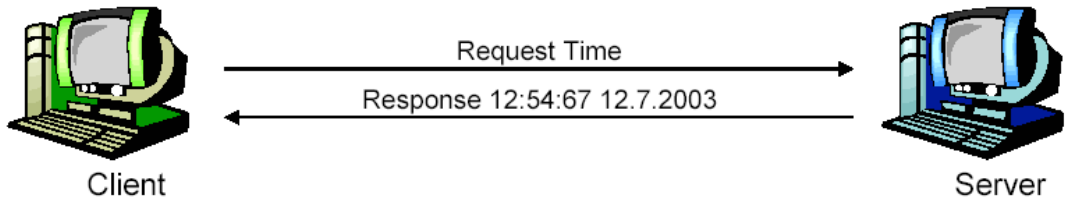
Clock Synchronization in Distributed Systems

- Issues caused by computers
  - Clock frequency depends on quartz oscillator
  - If the frequency is too high, the clock will be fast, if it is too low, it will be slow
  - The frequency is influenced by the environmental temperature
  - Limited resolution for determining the system time
  - High load might cause missed system clock interrupts

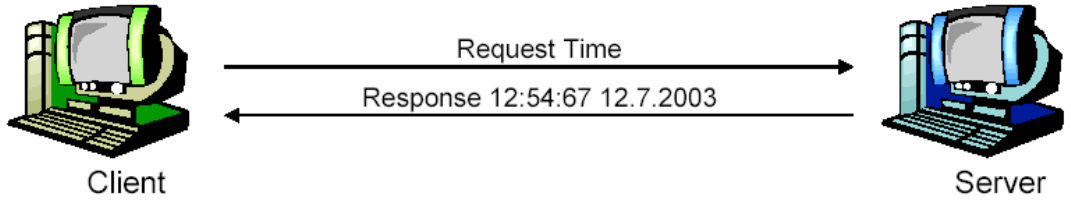
Clock Synchronization in Distributed Systems

- **Task**
  - **Synchronize the system time of a client with a server that has a precise clock (time server)**

- **Simple algorithm**
  - **Clients sends a packet to the time server requesting the current time**
  - **The time server responds with the current time**
  - **Client adjusts clock to the received time**



Clock Synchronization: Simple Algorithm

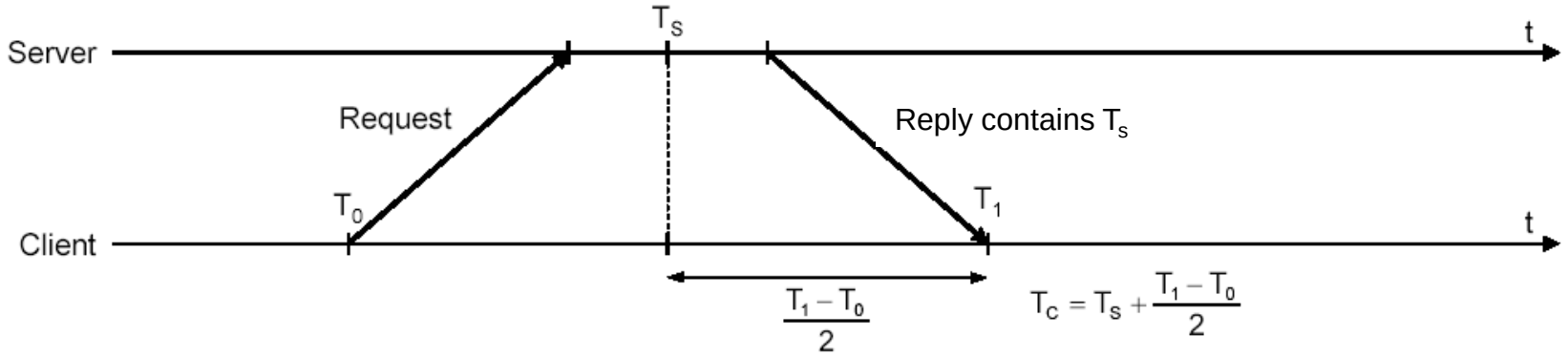


- **Advantage:**
  - **Trivial implementation**

- **Disadvantage:**
  - **Not precise**
  - **Does not consider server processing time**
  - **Does not consider message propagation time**

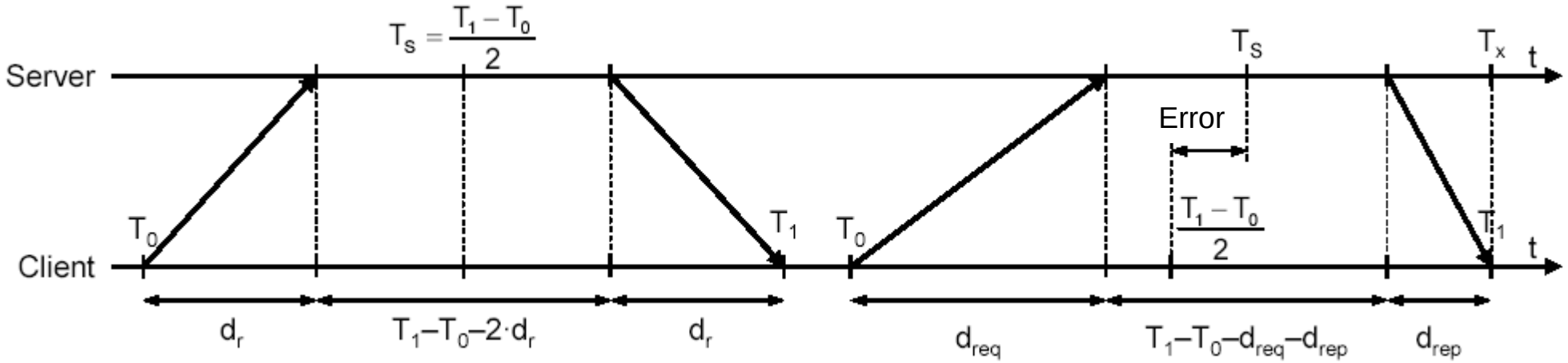
Clock Synchronization: Cristian's Algorithm

- **Properties**
  - **Decentralized algorithm, passive time server**
  - **Attempts to synchronize the system time of the client T_C with server T_S**
  - **Considers message propagation time for request and reply**
- **Approach**
  - **Compensates the propagation time (Round Trip Time RTT) by considering**
    - Time of request transmission T_0
    - Time of reply receipt T_1
  - **Assumes symmetric propagation time for request and reply**



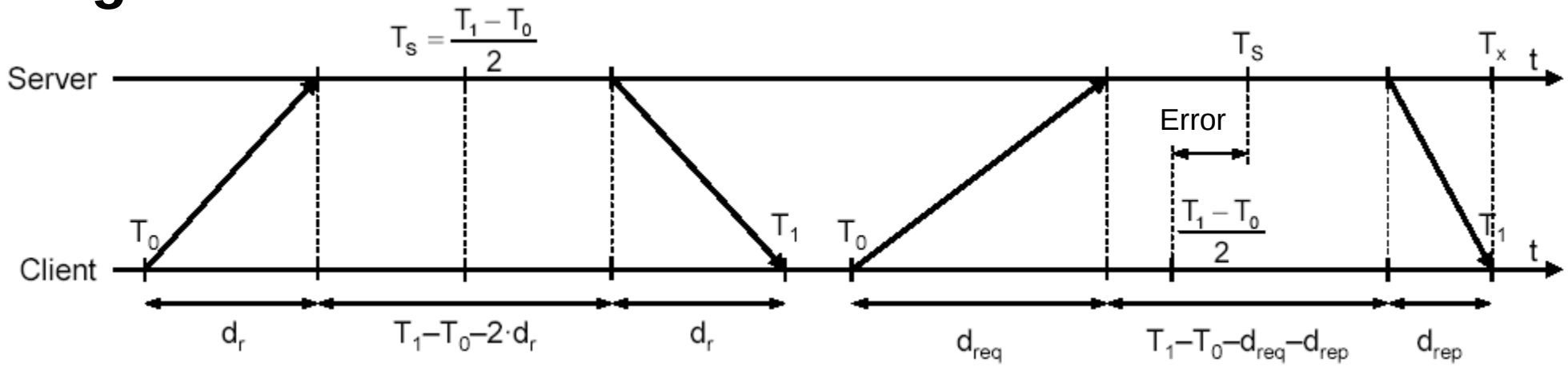
Clock Synchronization: Cristian's Algorithm

- **In reality, the message propagation time of request and reply are not symmetric**
  - **Ideal:** $d_{req} = d_{rep} = d_r$
  - **Real:** $d_{req} \neq d_{rep}$



Clock Synchronization: Cristian's Algorithm

- **Estimating the error**



- **For estimation, we assume minimal message propagation time d_{min}**
- **At time T_1 the current server time T_X is in the range $[T_S + d_{min}, T_S + RTT - d_{min}]$**
  - The server has set T_S at time $RTT - d_{min}$ at the earliest
  - The server has set T_S at time d_{min} at the latest
- **The error estimation is thus $\pm(\frac{RTT}{2} - d_{min}) = \pm(\frac{(T_1 - T_0)}{2} - d_{min})$**

Clock Synchronization: Berkeley Algorithm

- **Properties**
  - **Centralized algorithm, active server**
  - **Attempts to synchronize the system times of a group of machines**
  - **If the server fails, one of the clients take over**

- **Approach**
  - **The server polls the system time of all clients**
  - **It then calculates a time average**
  - **Outliers will be ignored in order to not skew the average**
  - **The server then calculates the time difference to the average (the offset) for each client and sends it to the respective client (careful: message propagation time fluctuations!)**
  - **Client adjusts clock to the received offset**

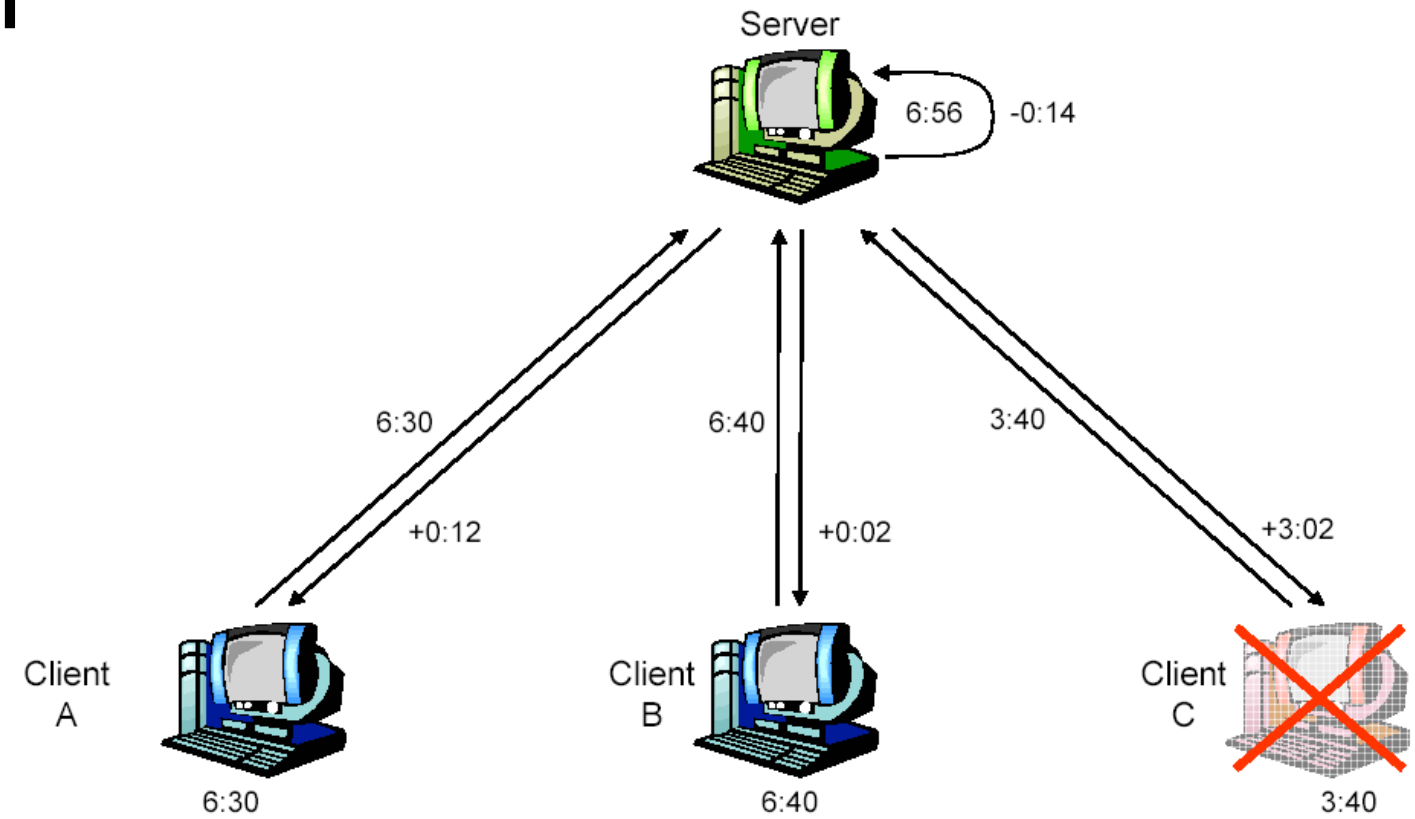
Clock Synchronization: Berkeley Algorithm

- Properties
  - Centralized algorithm, active server
  - Attempts to synchronize the system times of a group of machines
  - If the server fails, one of the clients take over
- Approach
  - The server polls the system time of all clients and calculates a time average
  - Outliers will be ignored in order to not skew the average
  - The server then calculates the time difference to the average (the offset) for each client and sends it to the respective client (careful: message propagation time fluctuations!)
  - Client adjusts clock to the received offset
- Disadvantages
  - Limited scalability with regards to number of clients
  - Different groups have different times

Clock Synchronization: Berkeley Algorithm

Example:

1. **Server fetches local time of all clients**
2. **Server discards outliers**
   - **Time of Client C**
3. **Server calculates average from remaining times**
   - $\frac{6:30+6:40+6:56}{3} = 6:42$
4. **Server sends respective offsets to all clients**
   - **Includes outlier clients**



NTP – The Network Time Protocol

- **Infos: https://www.ntp.org/**

- **History:**
  - **Developed since 1982 (NTP v1, RFC 10529), lead by David L. Mills**
  - **Since 1990: NTP v3, to some extend still in use**
  - **Since 1994: NTP v4, this is the current version**

- **Goal:**
  - **Synchronize computer clocks for computers that are connected via the internet**

David L. Mills

NTP – The Network Time Protocol

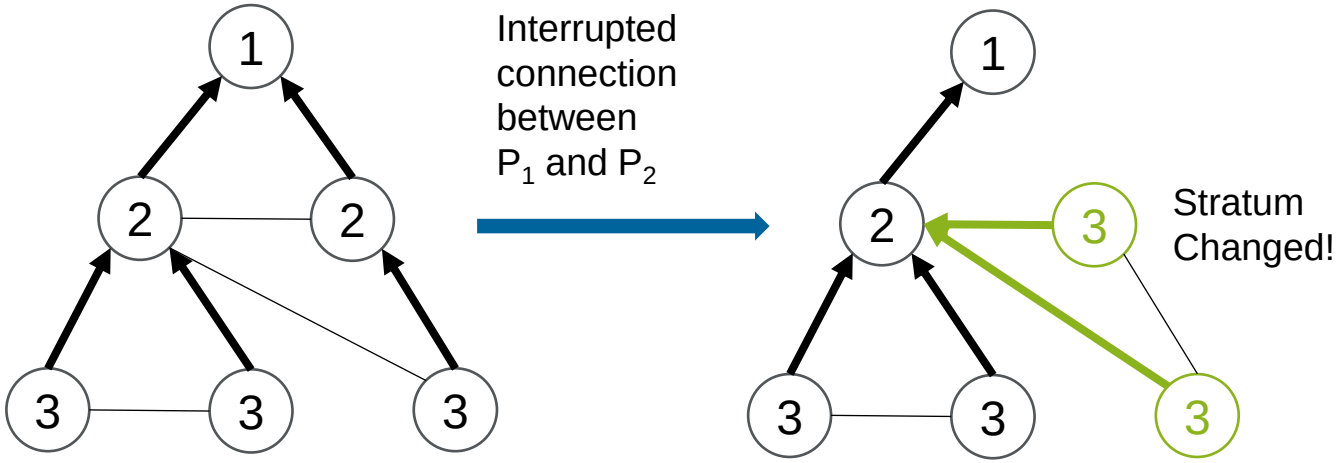
- Currently there are more than 100,000 NTP-nodes world-wide
  - List of public time servers: http://support.ntp.org/bin/view/Servers/WebHome
- Precision of up to 0.01 s in WANs, < 1 ms in LANs
- NTP-daemon is available of nearly every platform
  - Windows, Unix, Embedded Systems, …
  - UDP for transmitting time information
- Fault-tolerant

NTP Overview

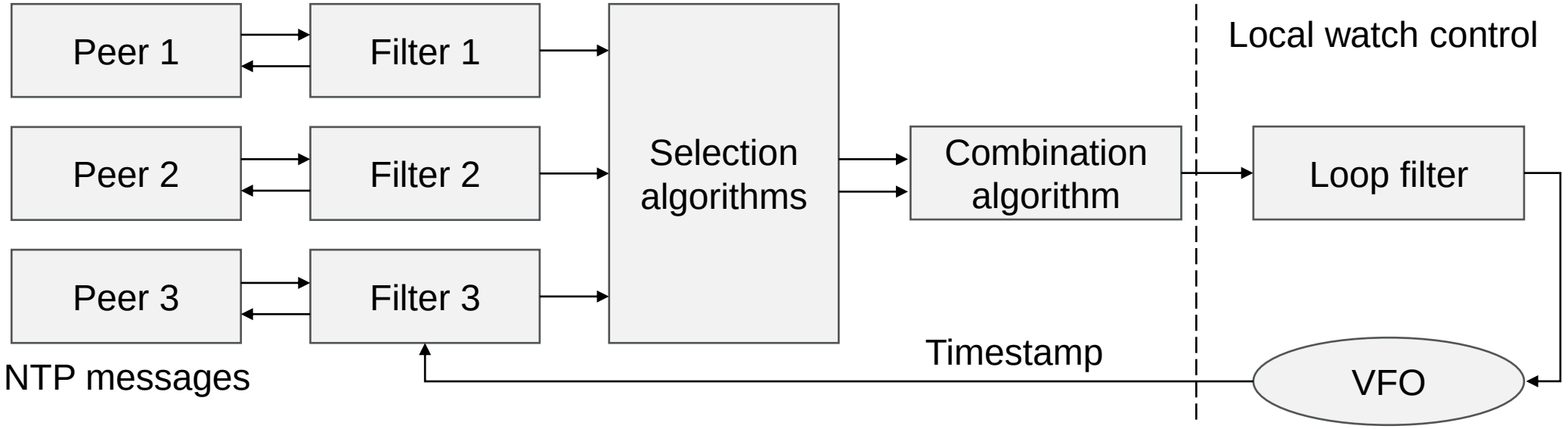
- **Primary servers (*Stratum 1*)**
  - **Synchronize with high-precision reference clocks via dedicated lines or radio**
- **Secondary servers and clients synchronize based on primary servers via a self-organizing, hierarchical network**
- **Different operating modes:**
  - **Primary/secondary, symmetric synchronization, multicast, …**
- **Reliability by use of redundant servers and network paths**
- **Optimized algorithms to reduce errors from jitter, changing reference clocks and erroneous servers**
- **Local clocks are adjusted in time and frequency by applying adaptive algorithms**

NTP Stratum

- **Stratum 1:**
  - **Primary time provider**
- **Stratum i:**
  - **i > 1**
  - **Synchronizes with time provider of stratum i – 1**
  - **Maximum stratum is 15, higher stratum means “unsynchronized”**
- **The stratum may change dynamically:**



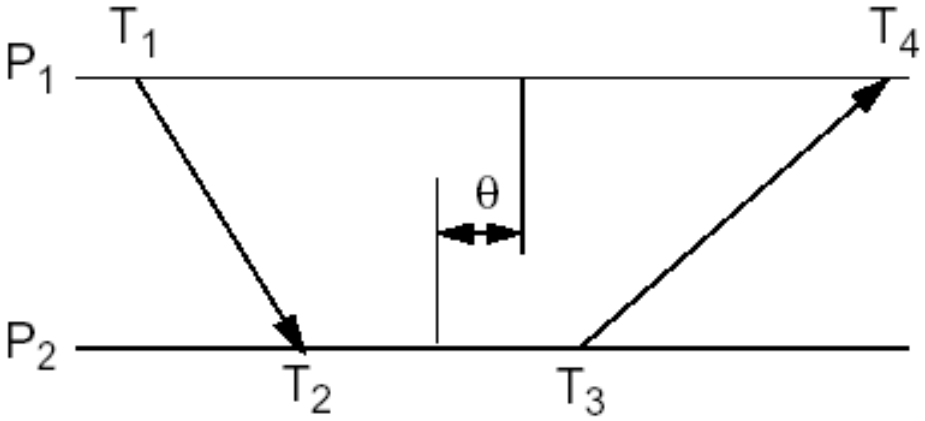
NTP Architecture



- **Multiple reference servers (peers) for redundancy and fault tolerance**
- **Peer-filters determine on a per-peer bases the best value from the last 8 offset measurements**
- **Selection algorithms try to detect correct clocks and filter incorrect clocks**
- **Combination algorithm calculated weighted mean**
- **Loop filter and variable frequency oscillator (VFO) control the local watch (frequency adaptation)**

NTP Offset Measuring

- Saves the 8 latest measurements
- Message propagation time (delta): $\delta = (T_4 - T_1) - (T_3 - T_2)$
- Estimated offset (theta): $\theta = \frac{T_2+T_3}{2} - \frac{T_1+T_4}{2}$
  - Exact value, if message propagation time is the same in both directions
  - Maximum error for asymmetric message propagation times: $\frac{\delta}{2}$



NTP Implementation in UNIX

- `xntpd` package
- NTP daemon `ntpd`
  - Configured via `/etc/ntp.conf`
- Query tool `ntpq`
- Tool for setting the time via NTP: `ntpdate`
- Tool for tracing the NTP strata: `ntptrace`

Clock Synchronization: Logical Clocks

- Idea
  - It is impossible to synchronize physical clocks absolutely
  - Many processes require knowledge about the order of events
  - However, they to not require a reference to the “real” time
- Requirement
  - One has to be able to correctly derive the order of event s
  - Relation to the real time is not required
- Definition of logical time
  - If two events happen in a process, then their ordering is based on their observation
  - If a process sends a message to another process, then the “send” event has to happen before the “receive” event (causality)

Clock Synchronization: Logical Clocks

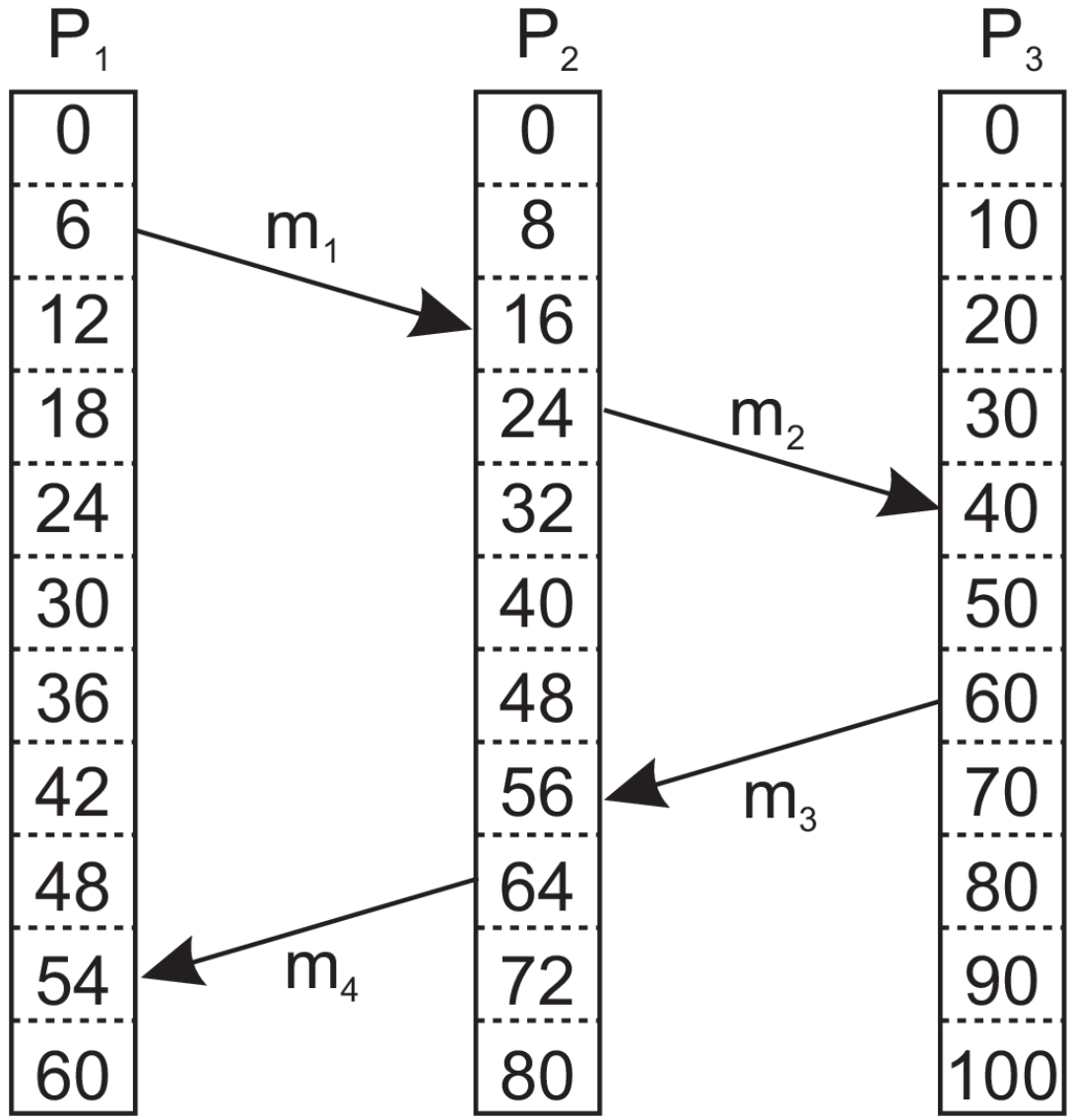
- Lamport defined a relation called “happens-before”
  - Written: $a \rightarrow b$
  - Read as: a happens before b
- Causality
  - If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Concurrent events
  - If events x and y happen in different processes that do not exchange messages, neither $x \rightarrow y$ nor $y \rightarrow x$ apply
  - The event are called concurrent, i.e., it is not possible to make assumptions on their temporal order: $x \,||\, y$ and $y \,|\quad|\, x$

Clock Synchronization: Logical Clocks

- **Definition of a logical clock C**
  - **When event a happens, the logical clock assigns it a time value $C(a)$, which is unique to the whole system**
  - **If events a and b happen with $a \rightarrow b$ then $C(a) < C(b)$**
  - **If a is sending a message and b is receiving this message, then $C(a) < C(b)$**
  - **C must increase monotonically**

Clock Synchronization: Lamport Algorithm

- **Assumption**
  - **Same clock frequencies**
  - **Different increments per interrupt**
    - P_1: increments by 6 per interrupt
    - P_2: increments by 8 per interrupt
    - P_3: increments by 10 per interrupt
- **Problem**
  - **P_1 and P_2 receive messages that have a higher timestamp than the local system time**
    - At t = 56 for P_2
    - At t = 54 for P_1
  - **P_1 and P_2 cannot correctly temporally order those messages**
  - **Order of events does not reflect the timestamps**



Clock Synchronization: Lamport Algorithm

- **Solution**
  - **Introduce logical time**
- **Approach**
  - **Each message header contains the timestamp when it has been sent**
  - **Each process checks this timestamp on receipt**
- **Consider two cases**
  - **Timestamp is lower than current system time → do nothing**
  - **Timestamp is higher than current system time → set system time to timestamp + 1**
  - **No violation of monotonicity**