Lecture 8: External Programs, NLP Pipeline

PCL II, CL, UZH
April 20, 2016

Contents

External Programs
The `subprocess` Module

```
>>> import subprocess
>>> subprocess.call(["touch", "file.txt"]) #prints 0
```

- the function `call` executes a command with arguments and returns its exit status
- command and arguments given as a list of strings
- command functionality executed
- meant mostly for simple commands

- **output** of the command **lost**

External Programs
The `subprocess` Module

```
>>> import subprocess
>>> subprocess.check_output(["ls", "-l"])
#prints a list of files with mod. dates, sizes, etc.
```

- the function `check_output` executes a command with arguments and returns its output as string
- exit status not lost -- if not 0, `CalledProcessError` thrown
- still meant mostly for simple commands

Contents

External Programs
Unix program communication

Standard streams:

- `stdin` -- standard input
  - program can read from it
  - commonly sent from keyboard
  - or can be directed from elsewhere
- `stdout` -- standard output
- `stderr` -- standard error output
  - program can write into them
  - commonly displayed on screen
  - or can be redirected elsewhere

External Programs
Redirection in Unix

- stdin redirection: `command < file`
- stdout redirection: `command > file`

  or `command >&2`
- stderr redirection: `command 2> file`

  or `command 2>&1`
- redirection between programs is done with pipes: `|`

```
command1 | command2
ls -l 2>ls-err.log | grep '\.txt' | cut -d . -f 1 >& log
```

External Programs Redirection in Python

- subprocess.Popen() enables more detailed control over the process and its standard streams:

```
proc = subprocess.Popen(['ls', '-l', '/dev/'])
print "done"
```

- process executed in the background

External Programs
Redirection in Python

- subprocess.Popen() enables more detailed control over the process and its standard streams:

```
proc = subprocess.Popen(['ls', '-l', '/dev/'])
proc.wait()
print "done"
```

- process executed in the background
- `processHandle.wait()` instructs the interpreter to wait for the command to complete

External Programs
Redirection in Python

```
proc = subprocess.Popen(['./test.py', 'arg'],
       stdin=subprocess.PIPE,
       stdout=subprocess.PIPE)

proc.stdin.write("hello\n")
proc.stdin.write("hi again\n")
proc.stdin.close() #must close for the process to continue

for outputLine in proc.stdout:
   print('OUT: ' + outputLine),

proc.wait()
```

- "Deadlock": both programs wait for each other
- Safer alternative: the `communicate` method

External Programs
The `communicate` method

- `proc.communicate(input=None)` overtakes the communication
- no argument = input from `stdin`
- returns a tuple `(stdoutOutput, stderrOutput)`
- waits automatically

External Programs Example

```
import subprocess

proc = subprocess.Popen(['tree-tagger-german'],
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE)

inputData = "\n".join(["Der", "Hund", "bellt", "laut",
"."])

(out, err) = proc.communicate(inputData)

print out,
```

External Programs
Process communication

using `communicate`:

- Pro's: simple, no deadlocks
- Con's: slower since all data is copied in memory

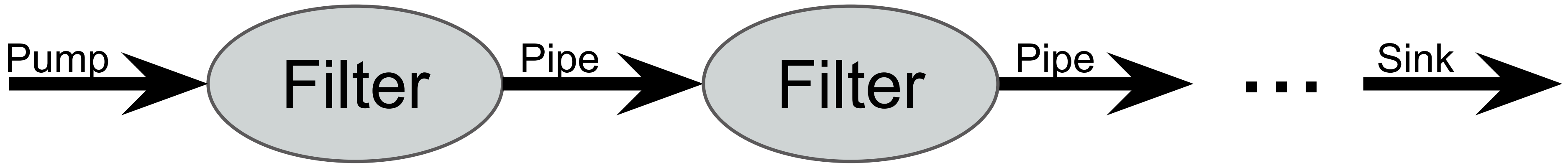
manually, via `read/write/`...:

- Pro's: fast
- Con's: complicated and deadlocks possible

Contents

Pipeline

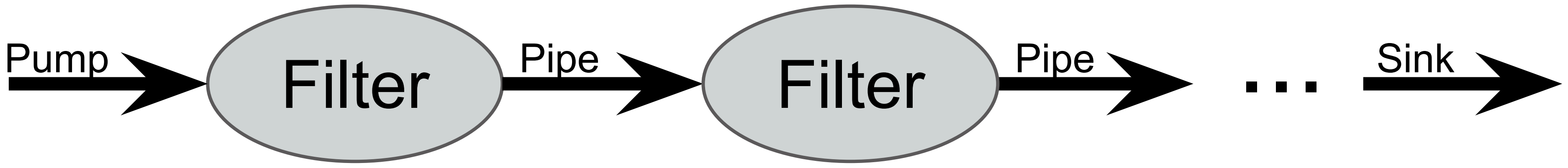
- pipes transfer data from the input (pump) through the filters to the output (sink)
- filters process the data, they are fully independent of each other



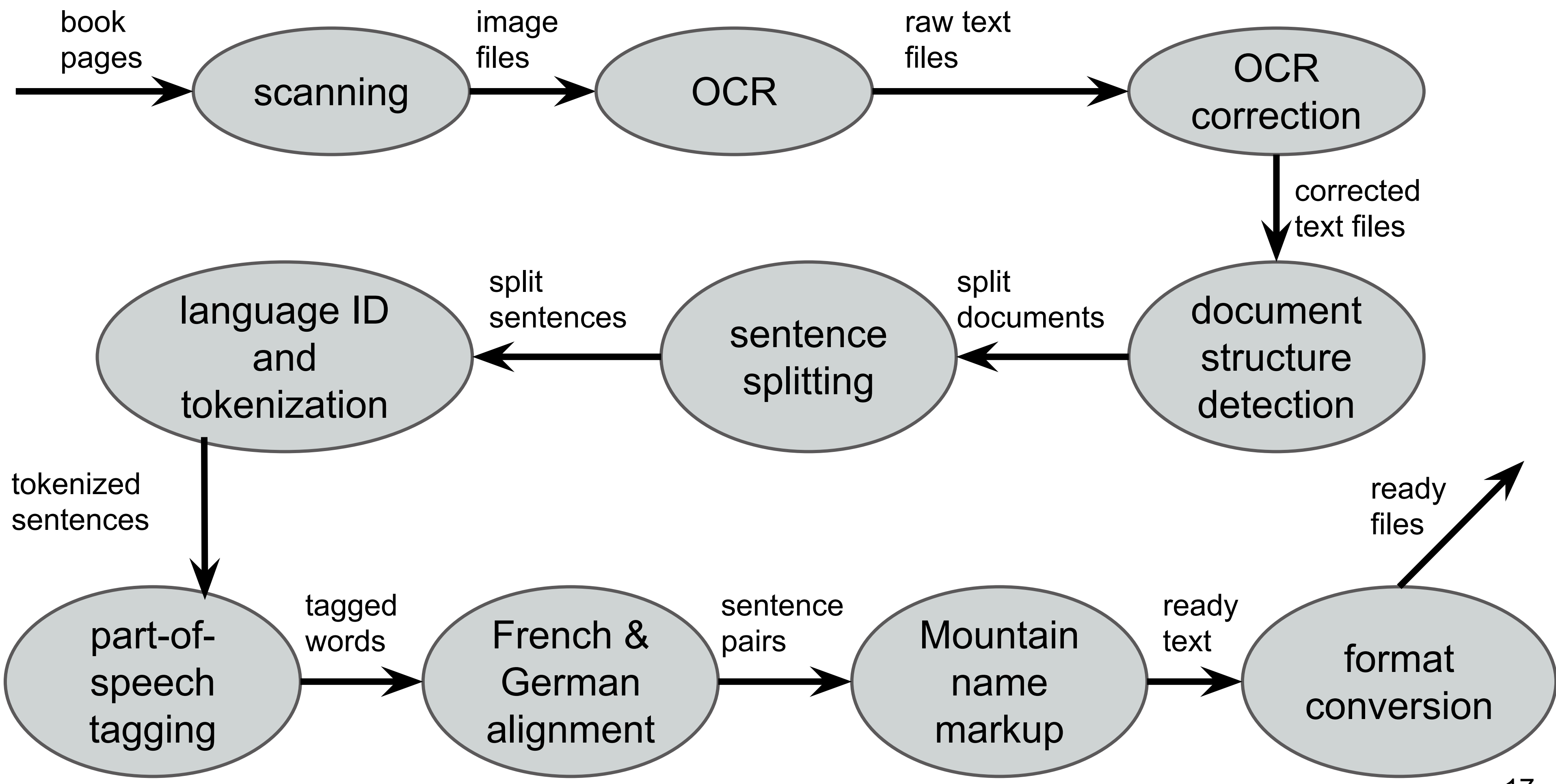
Pipeline

- pipes transfer data from the input (pump) through the filters to the output (sink)
- filters process the data, they are fully independent of each other

- Pro's: modular architecture, easy to understand in pieces
- Con's: errors on one step passed on to next steps



NLP Pipeline Example: Text+Berg



Lecture 8: External Programs, NLP Pipeline

PCL II, CL, UZH
April 20, 2016

Universität Zürich^UZH