Lecture 8: External Programs, NLP Pipeline

PCL II, CL, UZH
April 20, 2016

Universität Zürich[UZH]

Contents

External Programs
Usage

- A lot of tasks have been solved previously
  - no need to re-implement
    - unless it's for an educational purpose

- Some tasks could be solved more efficiently in a different programming language

External Programs
Usage

To consider:

- Platform
- State of the program
  - stable / beta / ...
- State of the code
  - clear / spaghetti / ...
- Support
- Documentation
- License

External Programs Licensing

Universität Zürich[UZH]

- Regulates the usage of a program
- Might allow/prohibit certain ways of using
  - Commercial
    - usage for a fee, one-time / annual
  - Free software
    - might be restricted, e.g. only for research and educational purposes
    - *e.g.* Tree-Tagger (PoS-tagging)
  - Open source
    - source code and compiled program available
    - might regulate inclusion of code into other programs
    - *e.g.* Moses (statistical machine translation)

External Programs
The subprocess Module

Universität Zürich[UZH]

```
>>> 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

Universität Zürich[UZH]

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

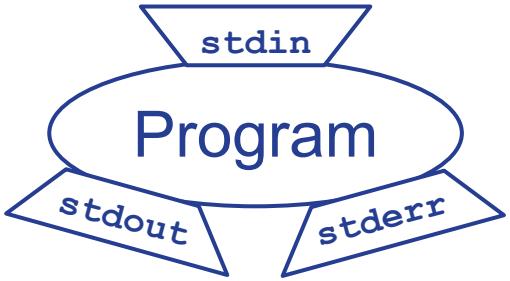
- the function `check_output` executes a command with arguments and returns its output as string
- exit status not lost -- if not 0, `CalledProcessError` thrown
- still meant mostly for simple commands

External Programs
Unix program communication

- Program arguments
- **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

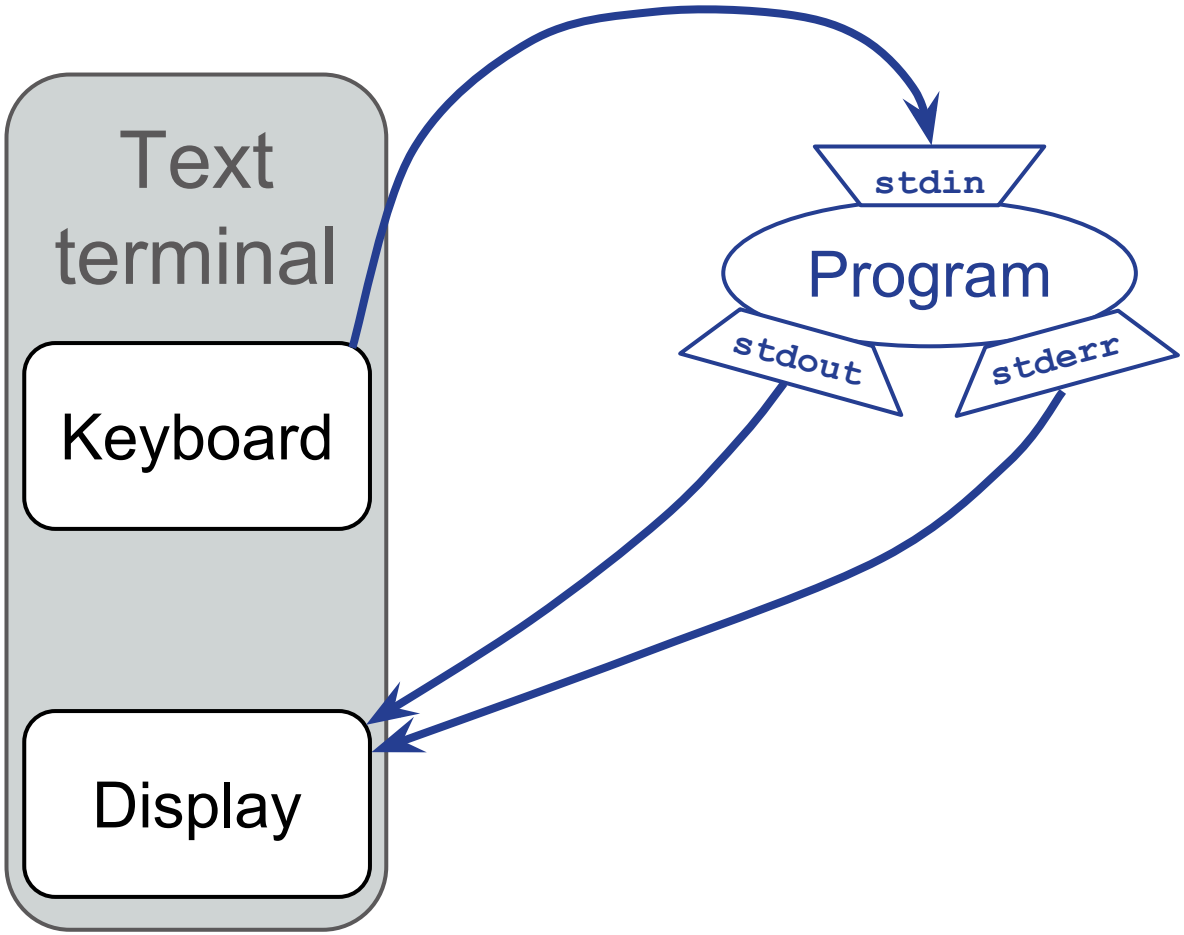
External Programs
Redirection

Universität Zürich[UZH]



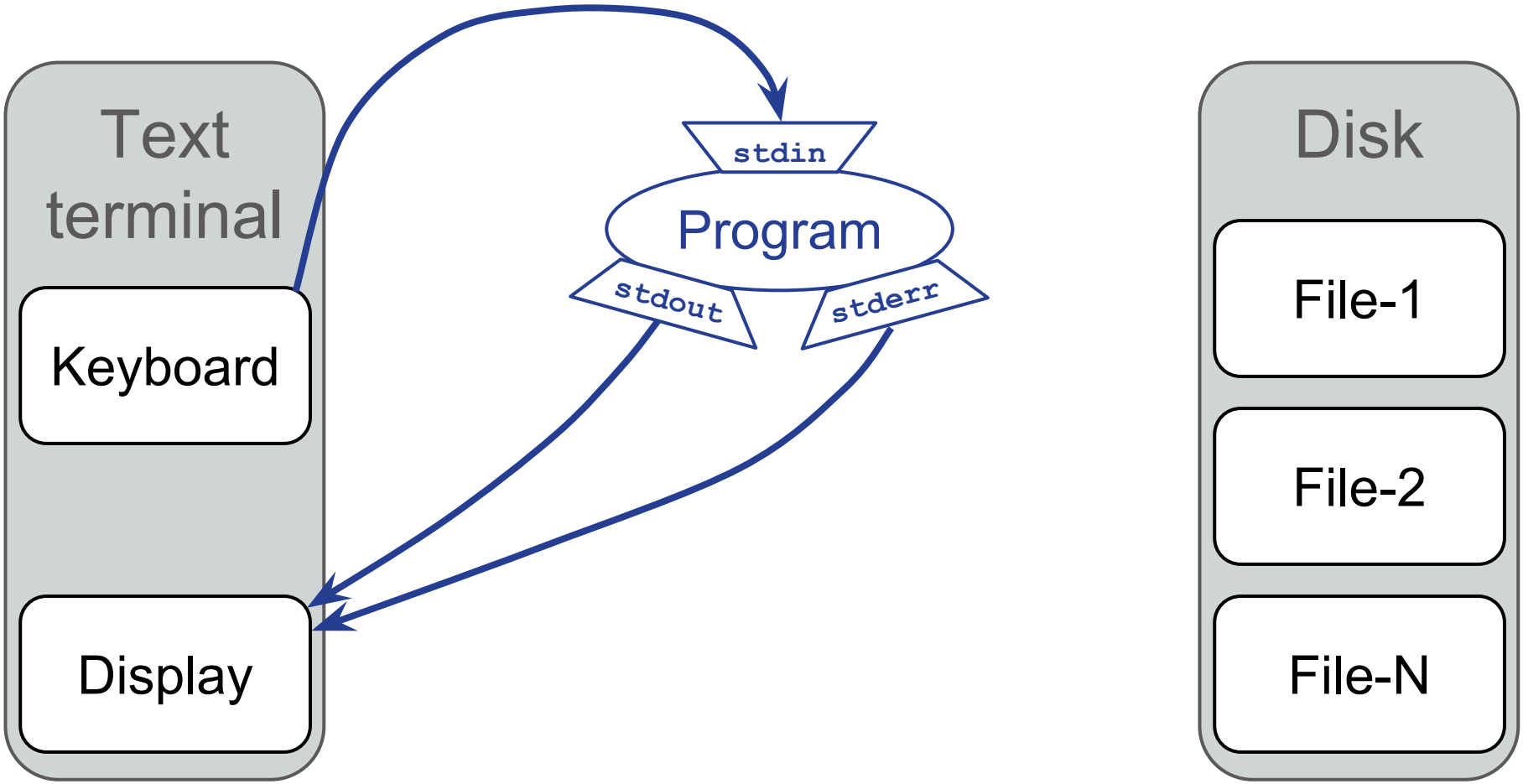
External Programs
Redirection

Universität Zürich[UZH]



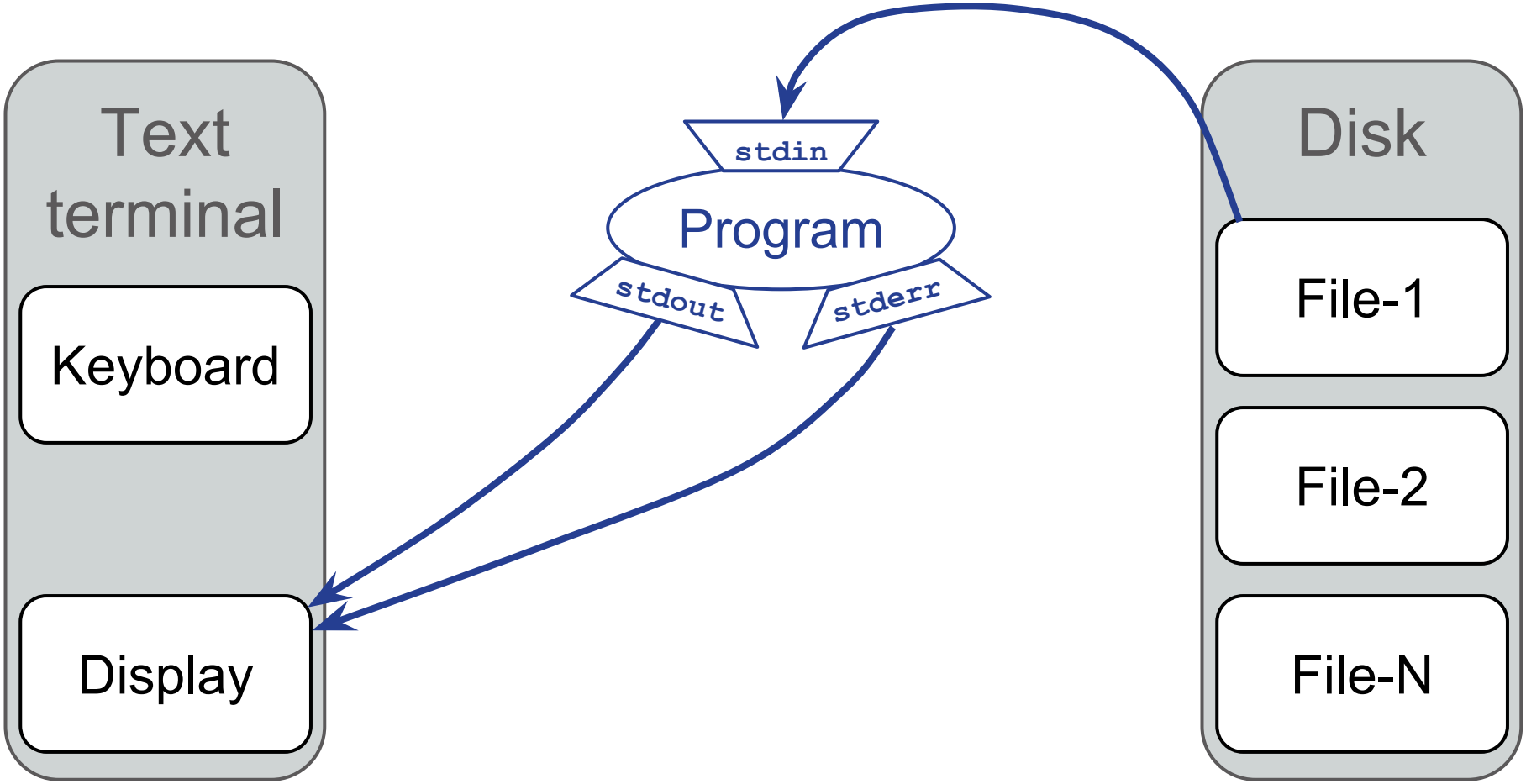
External Programs Redirection

Universität Zürich[UZH]



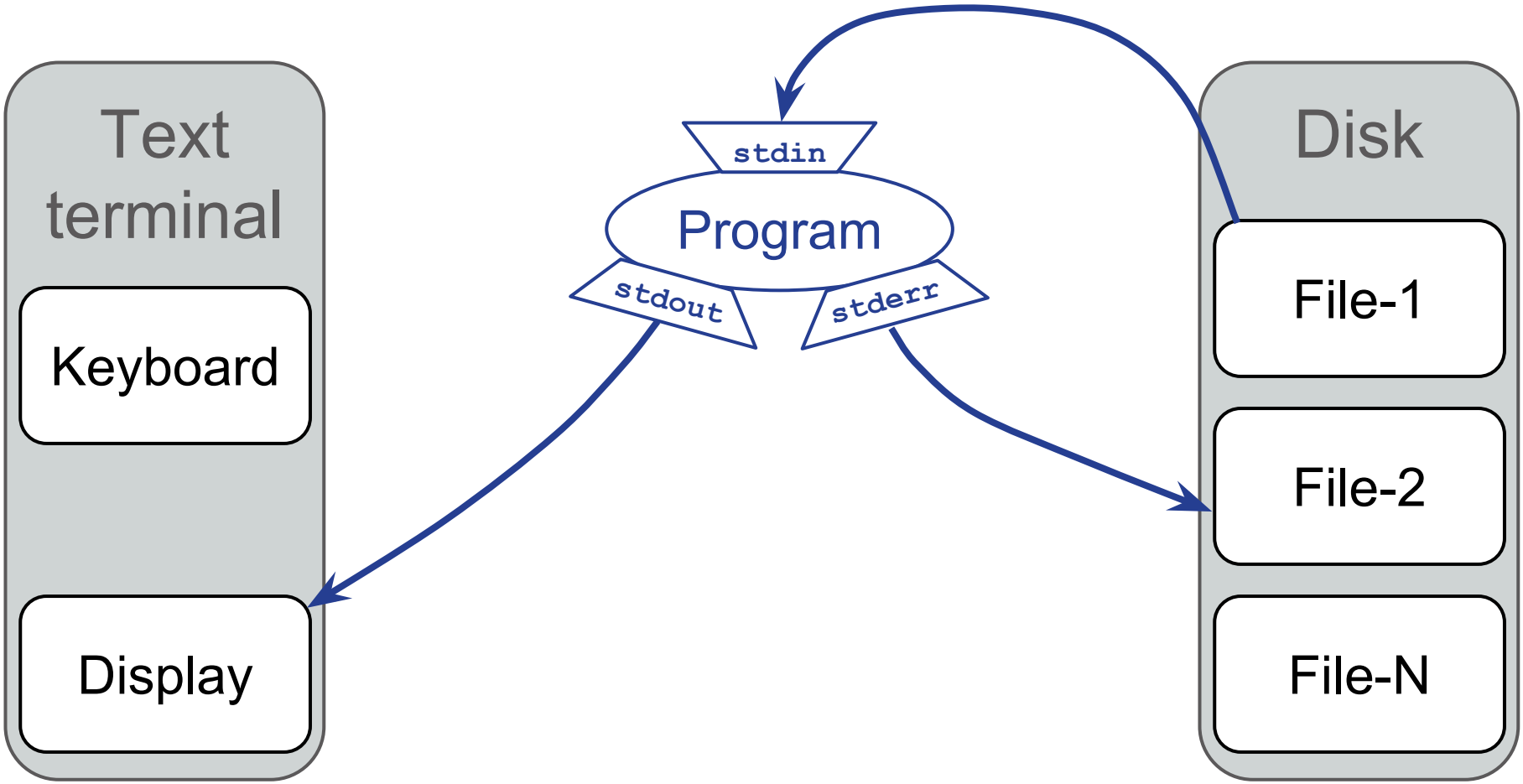
External Programs
Redirection

Universität Zürich[UZH]



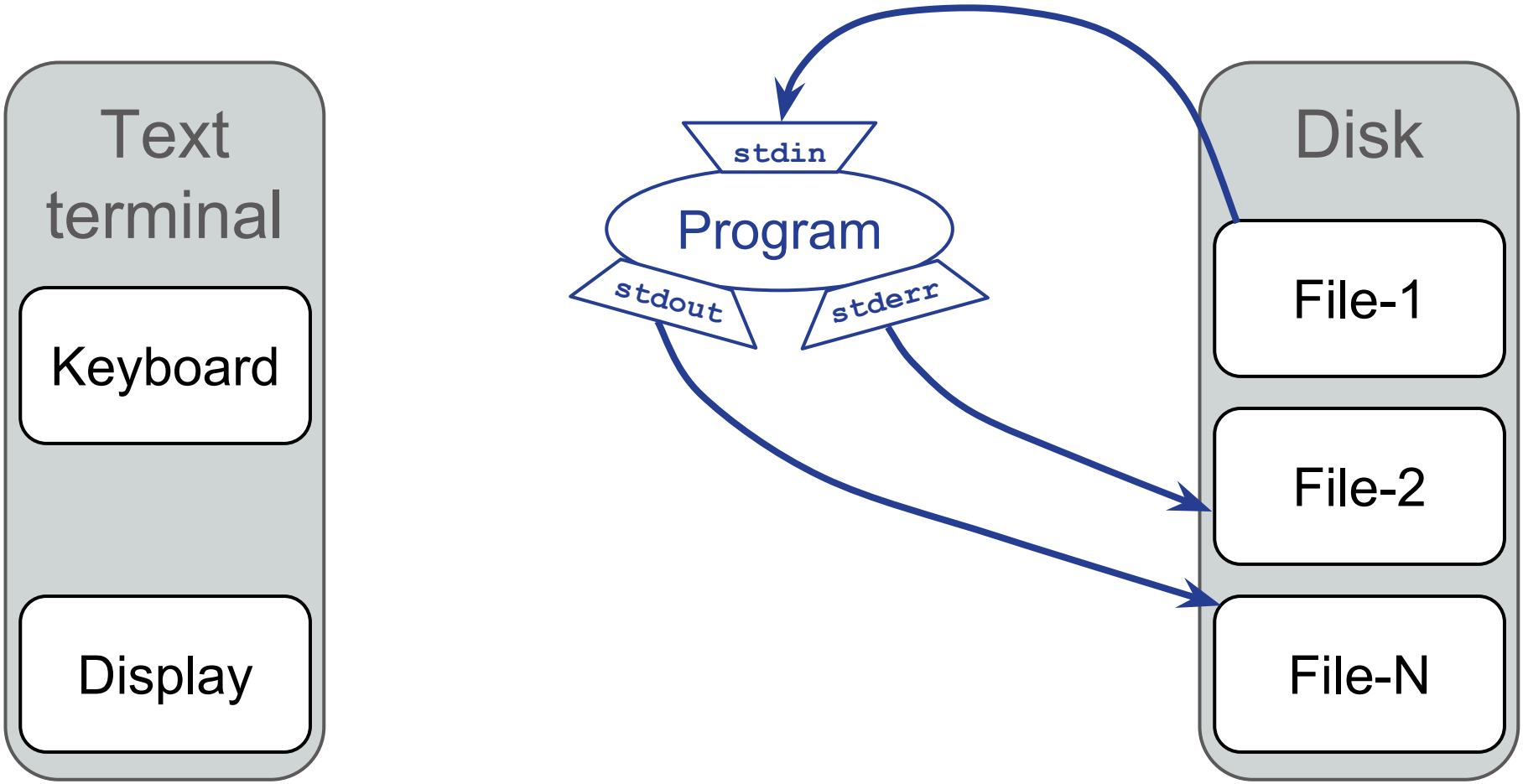
External Programs
Redirection

Universität Zürich[UZH]



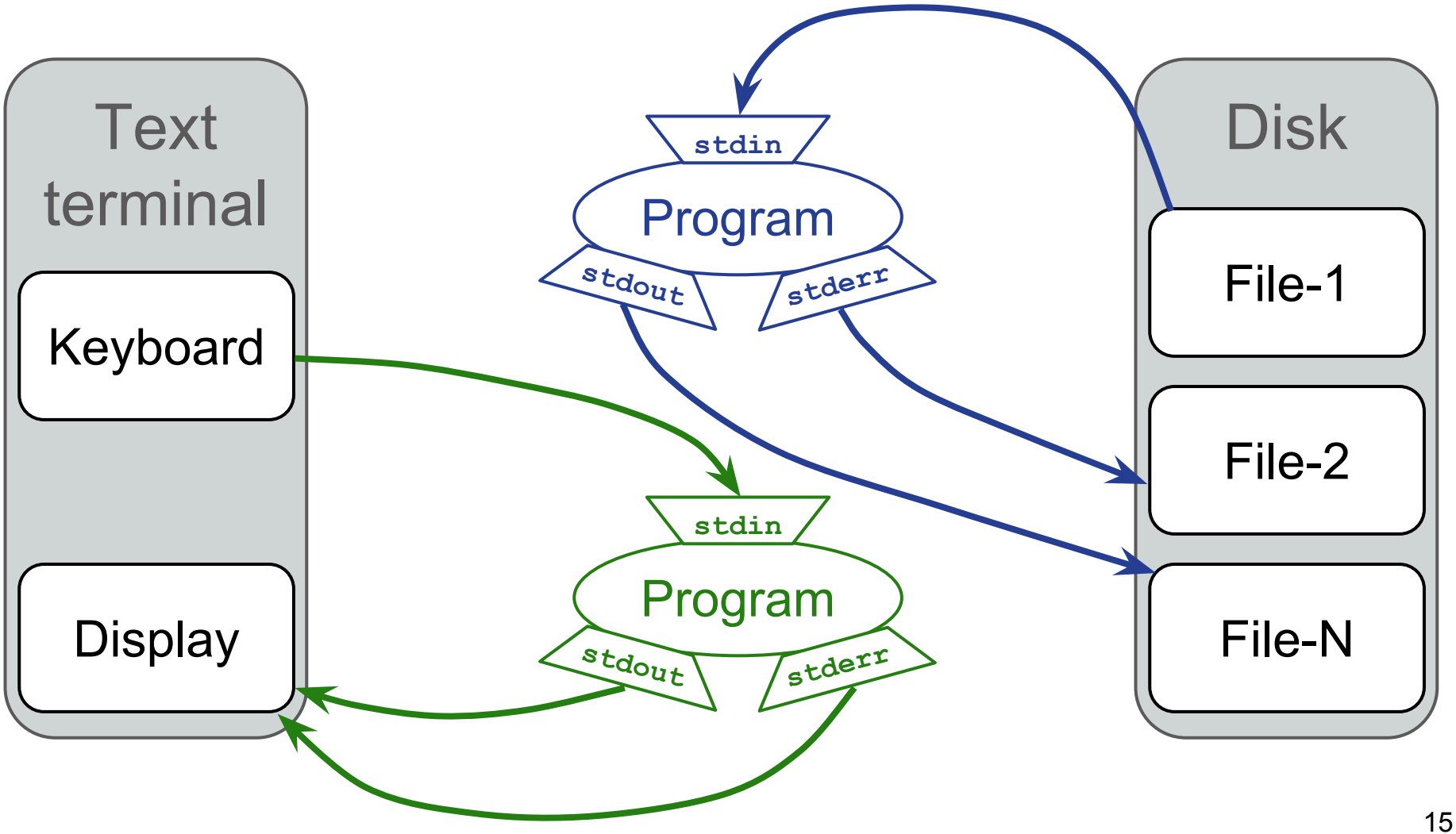
External Programs
Redirection

Universität Zürich[UZH]



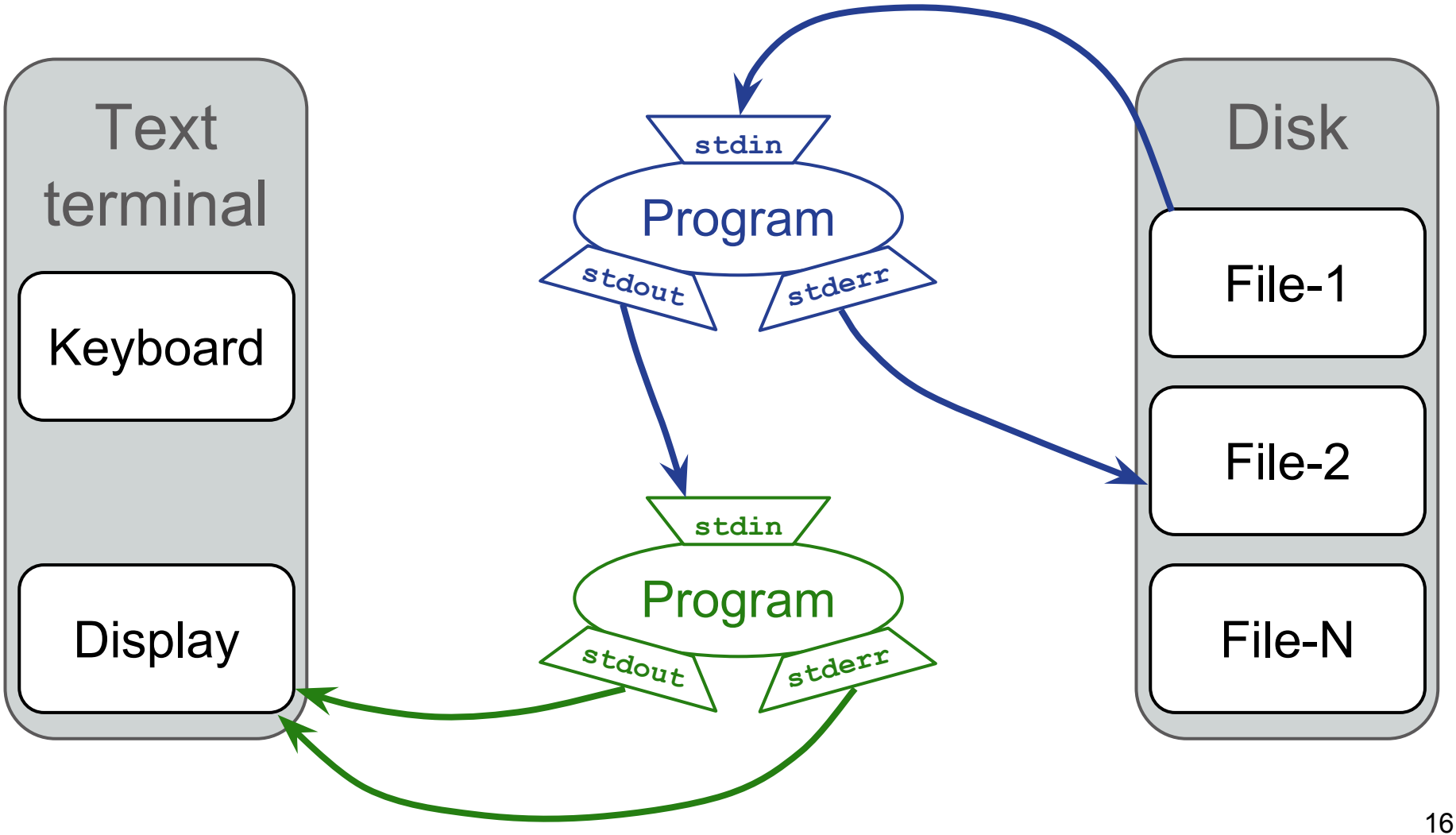
External Programs
Redirection

Universität Zürich[UZH]



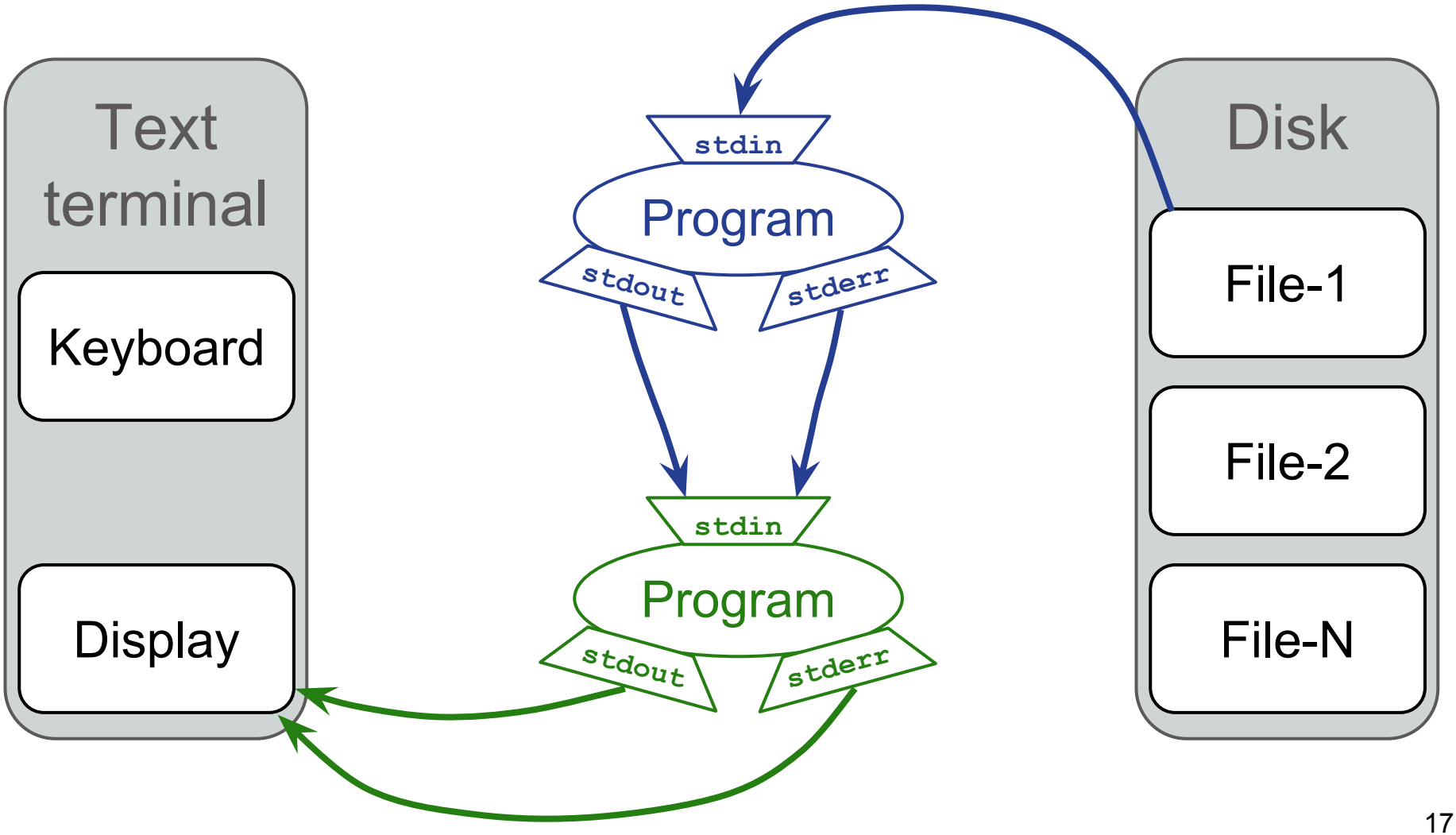
External Programs
Redirection

Universität Zürich[UZH]



External Programs Redirection

Universität Zürich[UZH]



External Programs
Redirection in Unix

Universität Zürich[UZH]

- stdin redirection: `command < file`
- stdout redirection: `command > file`
  or `command >&2`
- stderr redirection: `command 2> file`
  or `command 2>&1`

External Programs
Redirection in Unix

Universität Zürich[UZH]

- redirection between programs is done with pipes: `|`

```
command1 | command2
```

- redirects `stdout` of `command1` into `stdin` of `command2`
- `>`, `<`, `2>` and `|` can be combined

  e.g. to redirect `stderr` of `command1` into `stdin` of `command2`

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

External Programs
Redirection in Python

Universität Zürich[UZH]

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

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

External Programs
Redirection in Python

Universität Zürich[UZH]

- 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

Universität Zürich[UZH]

- 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

External Programs
Redirection in Python

Universität Zürich[UZH]

- 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

Universität Zürich[UZH]

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

```
inputFile = open("/etc/wgetrc", "r")

proc = subprocess.Popen(['./test.py', 'arg'],
       stdin=inputFile,
#optional
       stdout=open("/tmp/test-out.txt", "w"))
#optional
#process executed in background

proc.wait()
#wait for its completion
```

External Programs
Redirection in Python

Universität Zürich[UZH]

- pipe redirection:

```
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() #must wait -- why?
```

External Programs
Redirection in Python

Universität Zürich[UZH]

- `proc.stdin`, `proc.stdout`, `proc.stderr` are file handles
  - `read()`, `for line in handle`, etc. supported
- Must close stdin
  - forget it, and hang your python program
- Must wait after reading stdout/stderr
  - try calling wait() before stdout/stderr is done, and hang your program again

- "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

Universität Zürich[UZH]

```
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

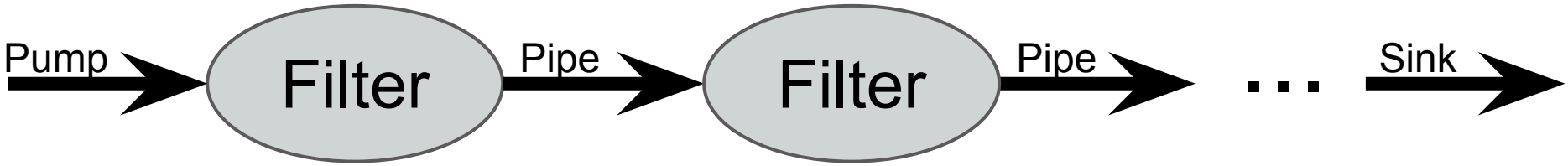
Contents

Pipeline

Universität Zürich[UZH]

- Architecture: structure or plan of some system
  e.g. your program or NLP project

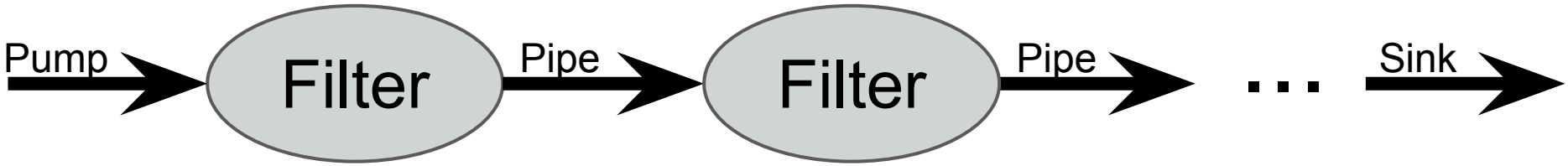
- Pipeline: one such architecture



Pipeline

Universität Zürich[UZH]

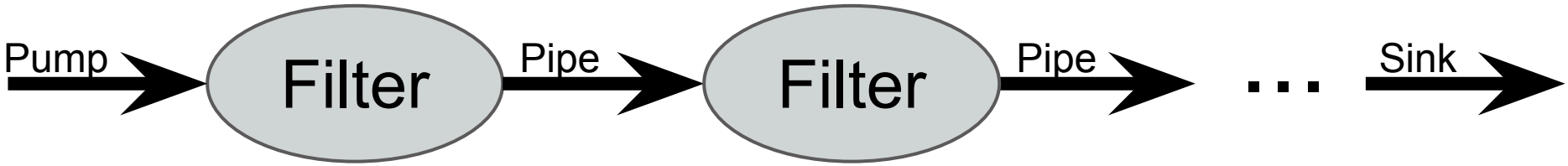
- pipes transfer data from the input (pump) through the filters to the output (sink)
- filters process the data, they are fully independent of each other



Pipeline

Universität Zürich[UZH]

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



NLP Pipeline

Universität Zürich[UZH]

Text processing pipeline:

- processing text can be viewed comfortably as a pipeline with independent consecutive steps
- some tools already have a pipeline infrastructure

  e.g. Stanford CoreNLP
- mostly incremental
  - next filter uses the output of the previous filter as input

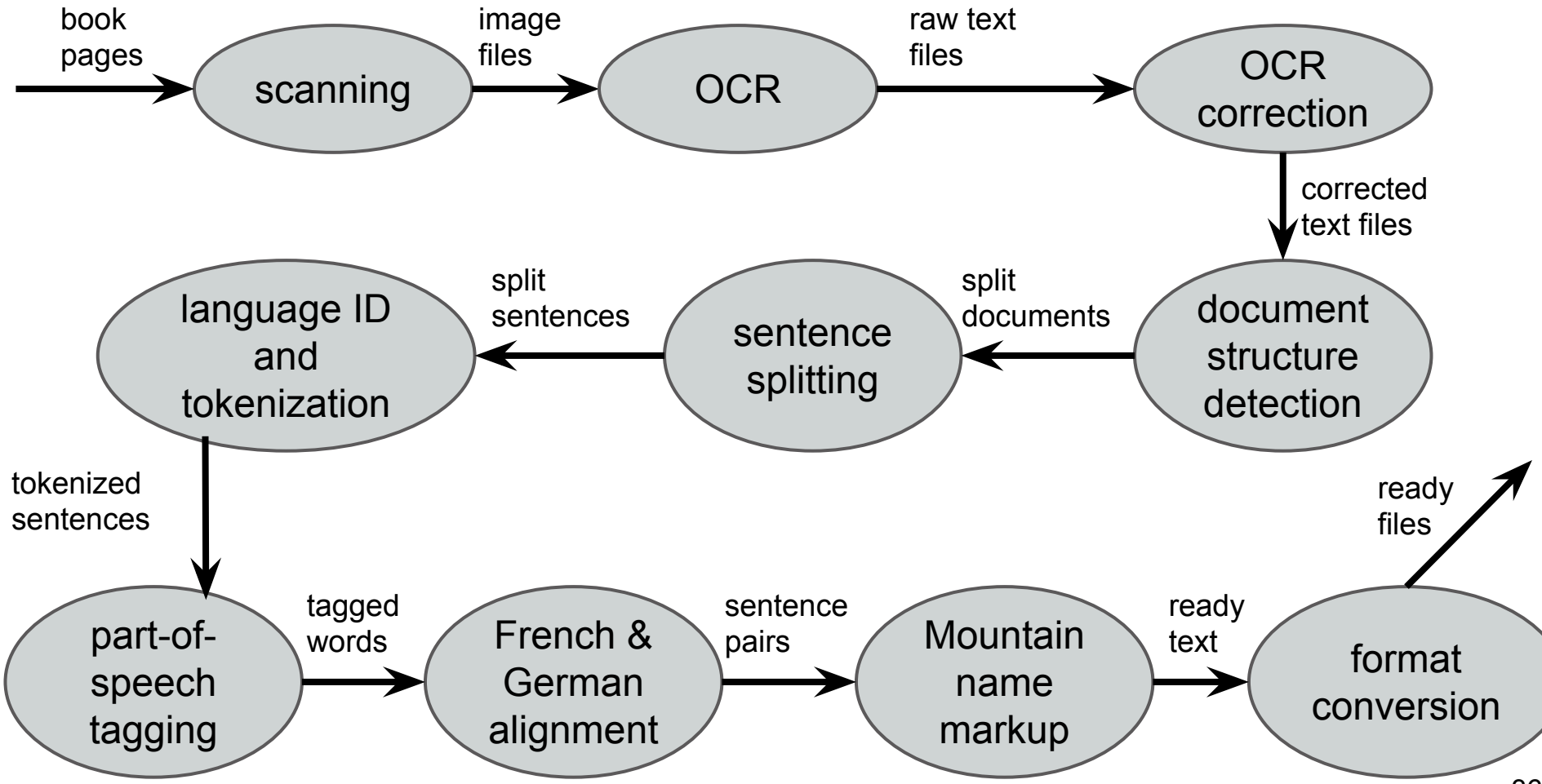
NLP Pipeline
Example: Text+Berg

Universität Zürich[UZH]

- Project aim: digitize the annual publication of the Swiss Alpine Club and make a corpus of it
- Corpus used for
  - German<->French translation
  - terminology search
  - named entity recognition
  - other annotations

NLP Pipeline
Example: Text+Berg

Universität Zürich[UZH]



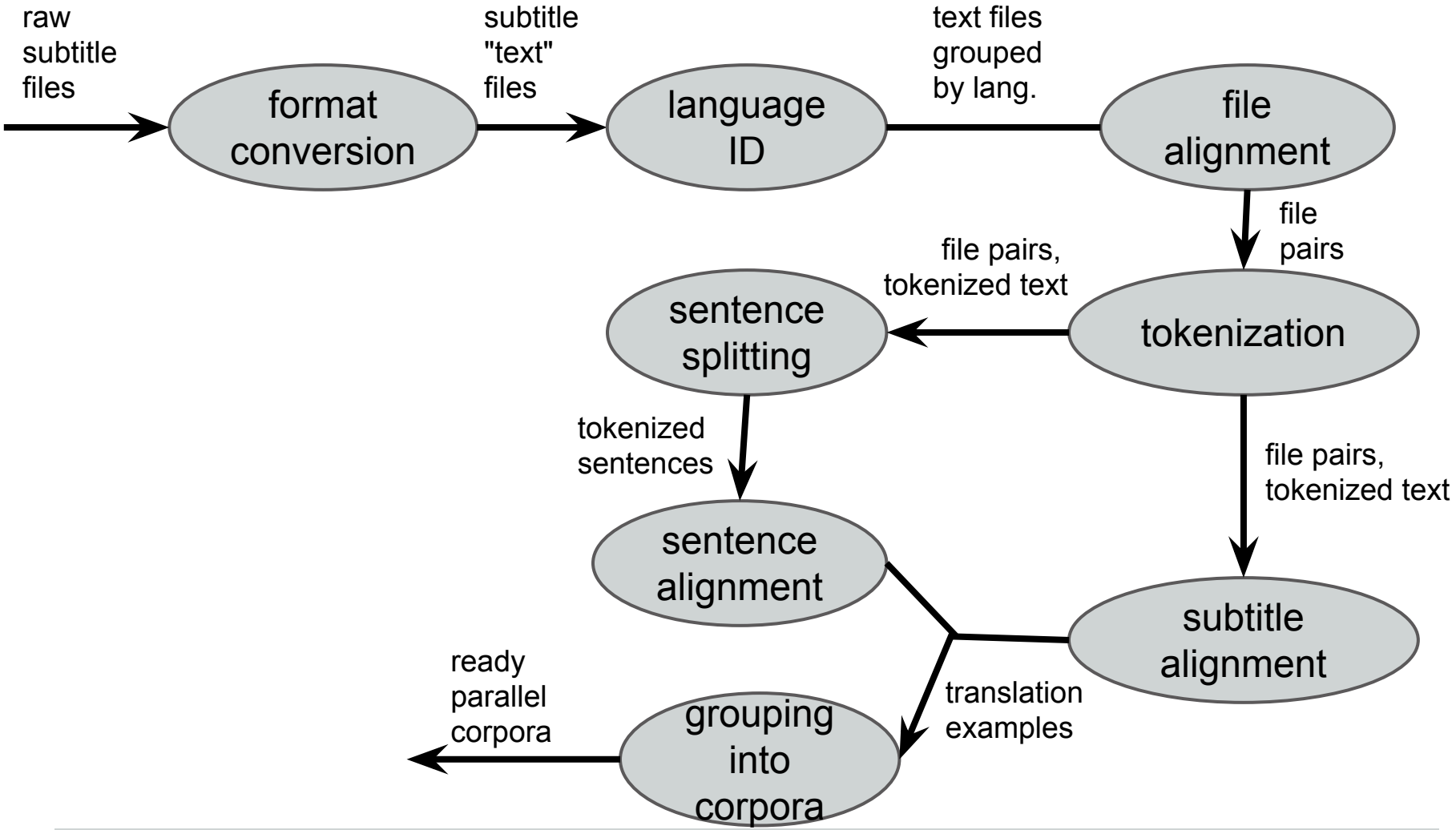
NLP Pipeline
Example: SUMAT

Universität Zürich[UZH]

- Project aim: create translation systems for subtitles
  - English<->
    - German, French, Spanish, Dutch, Portuguese, Swedish
  - Serbian<->Slovene
- Statistical machine translation
  - use translation examples to automatically learn to translate unseen input texts

NLP Pipeline
Example: SUMAT

Universität Zürich[UZH]



Lecture 8: External Programs, NLP Pipeline

PCL II, CL, UZH
April 20, 2016

Universität Zürich[UZH]