UNIX, C, C++

History, Philosophy, Patterns & Influences on “modern” Software Development

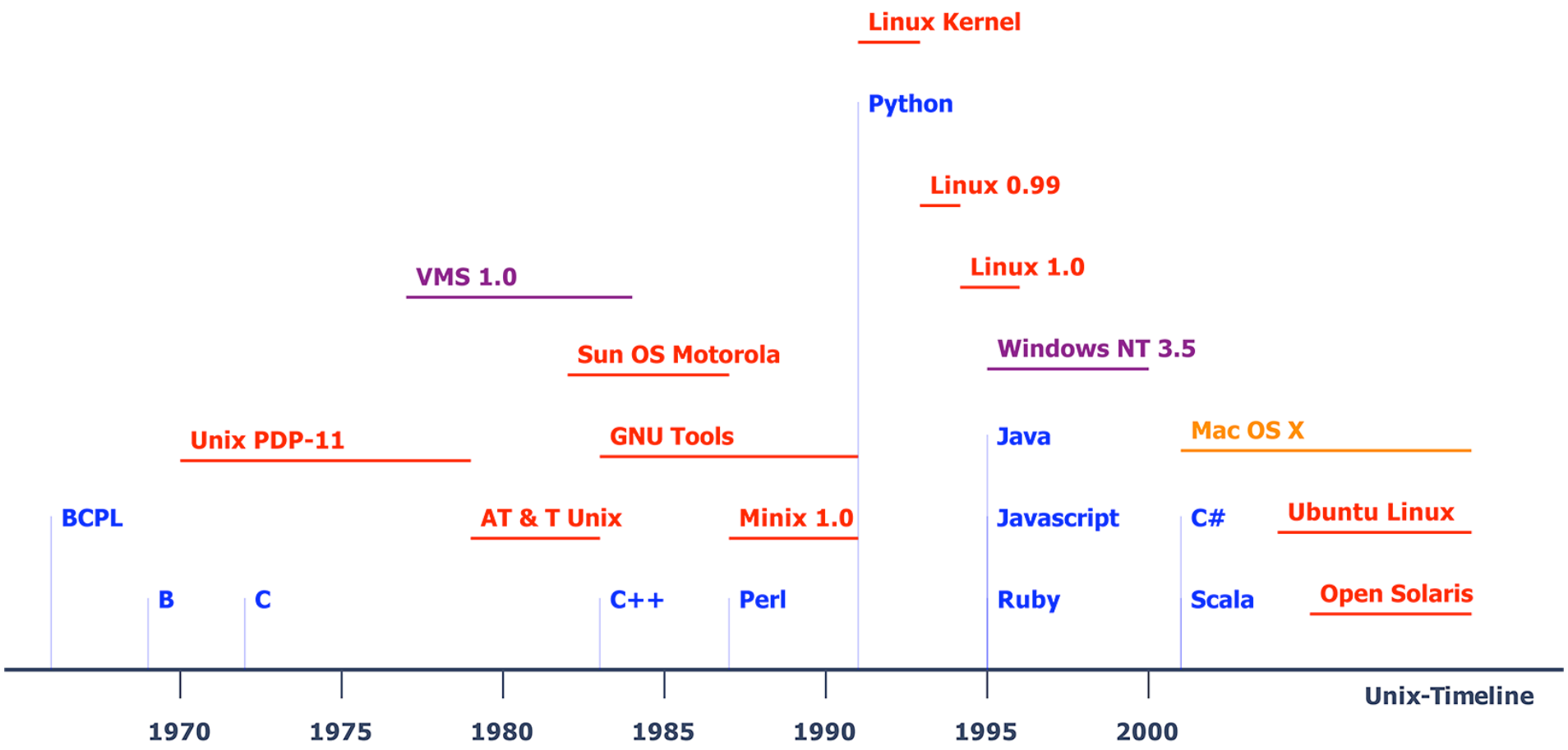
Alexander Schatten
www.schatten.info

November 2009

Agenda

- Timeline
- C and C++
- The Unix Philosophy
- Example: Unix and VMS styles
- Example: Unix Pipes
- Languages influenced by C / C++
- Modularisation in C, C++ and today
- “Test-Driven” Development
- Clarity and Simplicity vs. Expressiveness

Timeline


Linux Kernel
Python
Linux 0.99
Linux 1.0
VMS 1.0
Windows NT 3.5
Sun OS Motorola
Mac OS X
Unix PDP-11
GNU Tools
Java
BCPL
AT & T Unix
Minix 1.0
Javascript
C#
Ubuntu Linux
B
C
C++
Perl
Ruby
Scala
Open Solaris
Unix-Timeline
1970
1975
1980
1985
1990
1995
2000

B → C → C++

B (BCPL)

- B & BCPL were developed at Bell Labs
- Appeared 1969
- Not relevant any more
- “Curly Bracket” programming language
- Only one datatype: “word”
- Implementations for PDP-7 and PDP-11

C

- General purpose programing languange
- Introduced 1972 (Bell Labs) for PDP-11
- “B plus data-types and structure”
- Designed for Implementation of System-Software
- Low-level memory access
- Imperative (Procedural)
- Portability!
- Not object-oriented

B (BCPL) and C

PDP-7

C and C++

Dennis Ritchie and Ken Thomson designed C in 1972 in the Bell Labs

Photo from Flickr (Gubatron)

Bjarne Stroustrup: created C++ as enhancement of C starting 1979 (Bell Labs) "C with Classes" in 1983 renamed to C++

Software? Open Source?

Photo by Ben Franske

IBM System/360

Bill Gates (1977)

Unix and Open Source

GNU

Richard Stallman

GPL started 1989
Originated from similar licences in Emacs or GNU C Compiler

The UNIX vs. VMS Philosophy

- **VMS** was proprietary OS from DEC for VAX Minicomputers (Picture shows VAX 11/780 from late 70s, early 80s)
- Extremely **robust and stable** OS
- However, very **different design choices** compared to Unix
- **Windows NT** based on VMS principles
- First released **1978**

- Process Spawning in VMS expensive
- Rather monolithic large and complex applications
- Less CLI usage
- Using less, but more complex programs with more complex functions

Photo from Flickr (Astio)

The Unix Philosophy

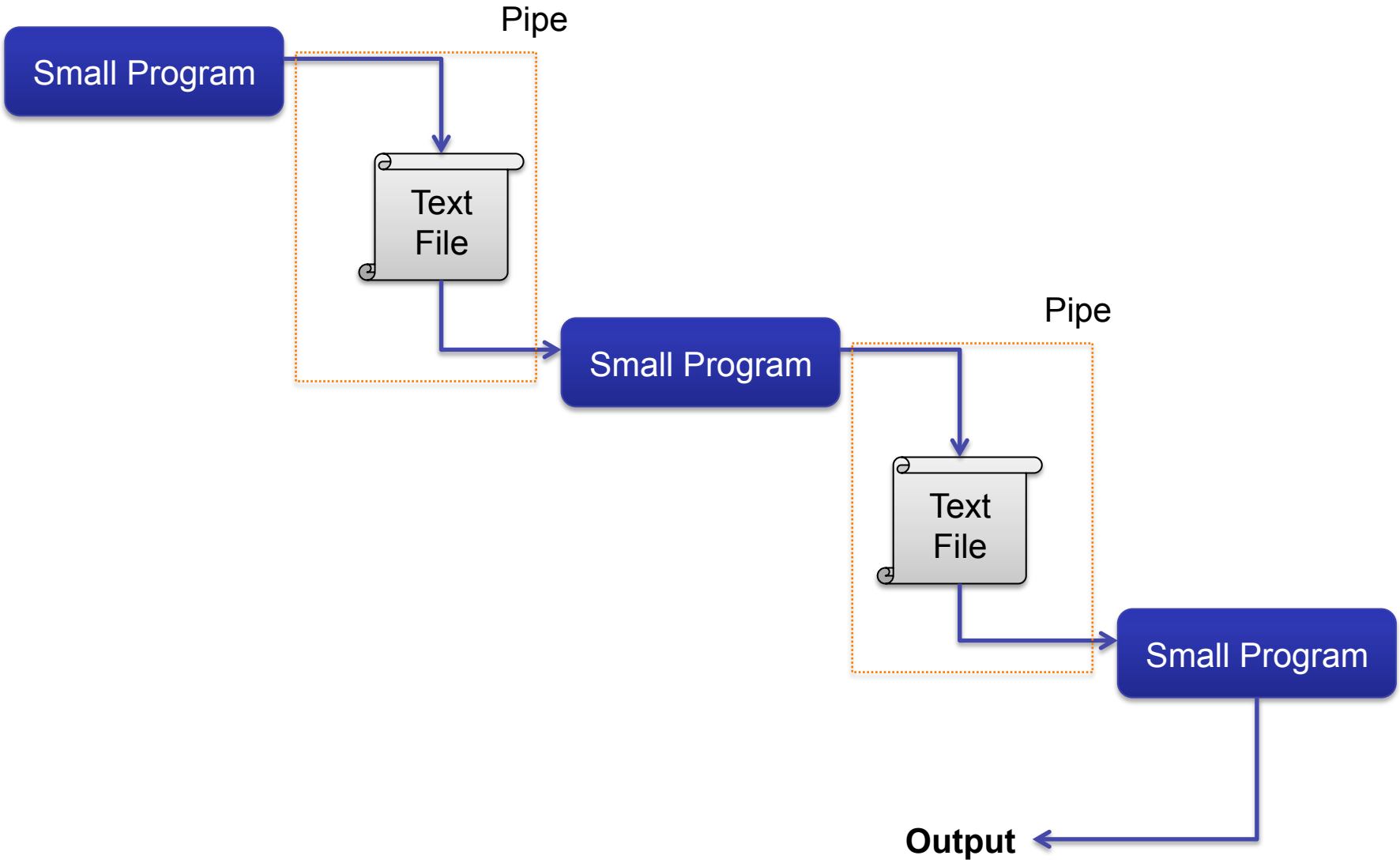
"Do one thing and do it well"

- **Modularity:** Simply parts connected by clean interfaces
- **Clarity** is better than cleverness
- **Simplicity**
- **Composition:** Write small programs to be connected
- **Representation:** Make data complex, not logic
- **Economy:** Developer time is expensive, machine time is not

- **Textual Data-Formats** (including e.g. GCC compile steps)
- **Command-Line Interfaces**
- **Library & Library Exercisers**

The Art of Unix Programing, Eric S. Raymond

Example: Unix Pipes 1



Wikipedia Example: Unix Pipes

```
curl "http://ANYURL" |
sed 's/[^a-zA-Z ]/ /g' |
tr 'A-Z ' 'a-z\n' |
grep '[a-z]' |
sort -u |
comm -23 - /usr/share/dict/words
```

1. Load File from URL
2. Remove Characters
3. Change uppercase to lowercase
4. Remove blank lines
5. Sort Lines
6. Compare List with dictionary

→ **Spellcheck of Webpage**

Languages influenced by C/C++

Java, Javascript, C#, PHP …

Example: Java

- Syntax very similar to C/C++
- No preprocessor, no multiple inheritance
- Higher level of abstraction (no direct system calls)
- Virtual Machine
- Primitive Types: standardised with VM
- Garbage Collection
- Type Safety
- **Cost and Benefit of Abstraction:** Rapidly growing standard library

Year	JDK Version	Packages	Classes
1999	1.2.1	21	358
2001	1.3.1	76	1840
2002	1.4.2	135	2723
2004	1.5	165	3279
2006	1.6	202	3777

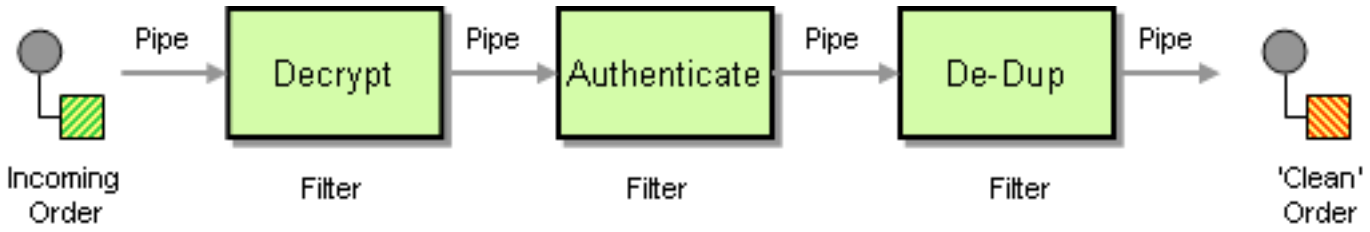
Modularisation in C / Unix

- **Conventions**, not Standards;
  Unix Compiler always works with one file
  - Per C File small number of functions
  - Header Files used to “expose” functionality
  - Grouping in “Packages”, i.e. Directories
- **Build-Scripts** using e.g. “make” to build whole projects

- **Configuration** using
  - textual config files
  - command-line parameters
- **Communication between applications via Text-(Files)**

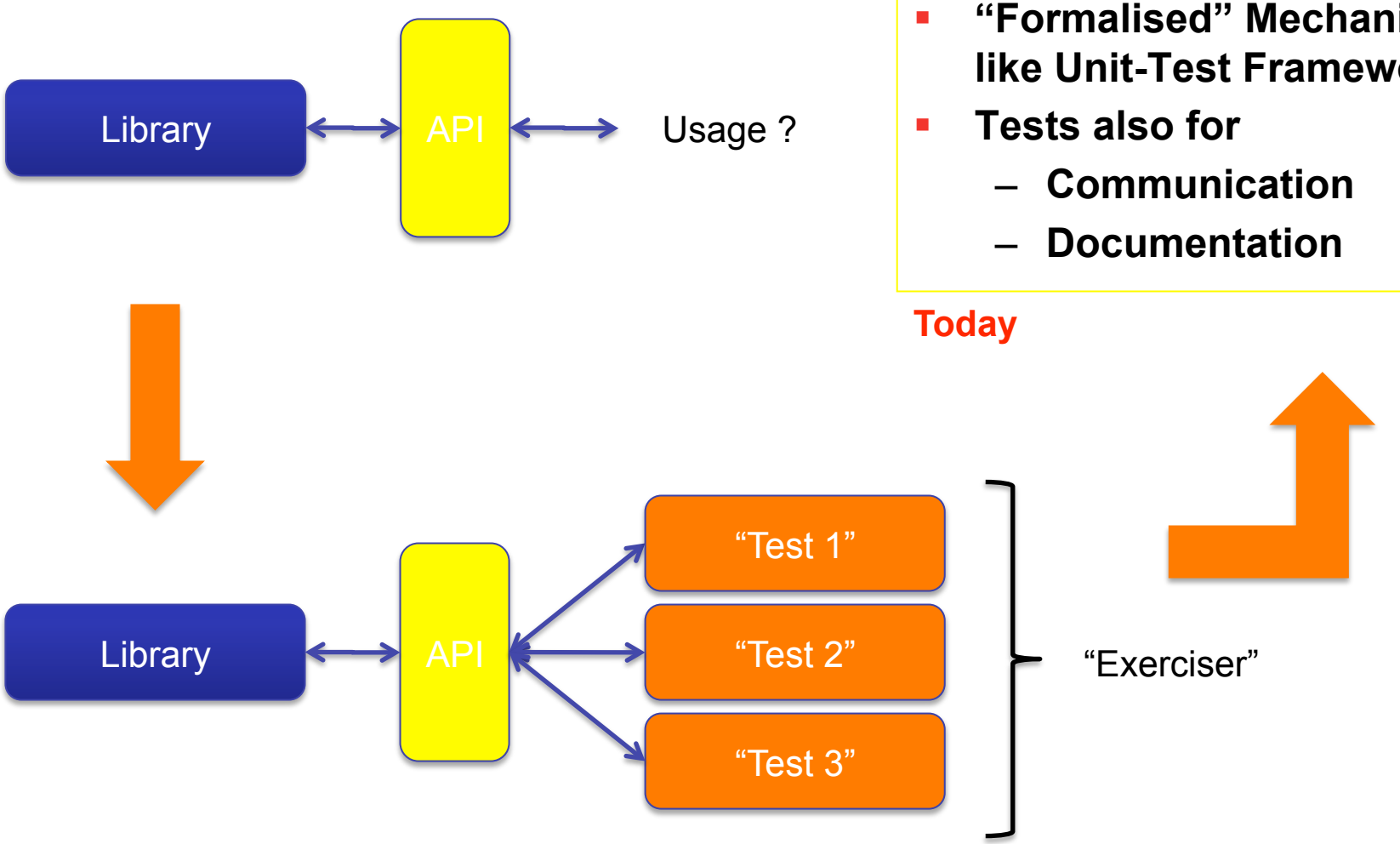
Modularisation Today

- Components → Services → Service Oriented Architecture
- Platform-specific RPC?
- Text(XML)-based interoperability
  - REST
  - Webservices (SOAP, WSDL)
  - Message oriented Middleware (e.g. JMS)
- Enterprise Integration Patterns
  - Pipes-and-Filters…Unix Pipes scaled up



"Pipes and Filters" Example from Gregor Hohpe: "Enterprise Integration Patterns"

From Library Exercisers to Test-Driven Development



- **Test-Driven Development**
- **"Formalised" Mechanisms like Unit-Test Frameworks**
- **Tests also for**
  - **Communication**
  - **Documentation**

Today

Clarity and Simplicity vs. Expressiveness

“Keep It Simple Stupid” ?

- Language Efficiency?
- Expressiveness of Languages
- Imperative vs. functional & “mixed” languages
- Learning Curve
- Long-Term vs. Short-Term Effects
- E.g. Scala vs. Java?
- Complex Frameworks?

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    int n = 0;

    while ( n < argc ) {
          printf("Command line argument
          %d is %s\n", (n+1), argv[n]);
          ++n;
    }

    return EXIT_SUCCESS;
}
```

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU / lreP rehtona tsuJ";sub p{
@p{"r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;($q*=2)+=$f=!fork;map{$P=$P[$f^ord
($p{$_})&6];$p{$_}=/ ^$P/ix?$P:close$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^[P.]/&&
close$_}%p;wait until$?;map{/^r/&&<$_>}%p;$_=$d[$q];sleep rand(2)if/\S/;print
```

Conclusion

- Unix Concepts from the 70s were extremely successful and influential until today
- Unix-Philosophy allows
  - complex yet robust global networks and
  - distributed software
- De-Coupling and Modularisation
- Testing
- Communication via Text-based protocols
- Power of Command-Line
- Open Source Movement

Dr. Alexander Schatten

Senior Researcher: Vienna University of Technology
IT Consulting

www.schatten.info | alexander@schatten.info | alex_buzz (Twitter)
best-practice-software-engineering.blogspot.com

Mantra

Keep it simple, stupid!