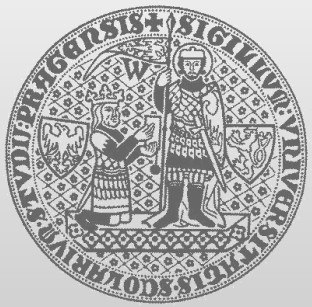
Software Building
(Sestavování aplikací)

http://d3s.mff.cuni.cz

Pavel Parízek

parizek@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Make

Make

- Standard build automation tool in the Unix and Linux world
- Used mainly for programs that use C/C++ and scripting languages (bash, Awk)
- Many derivatives exist
  - GNU Make, BSD Make, qmake, NMake, ...

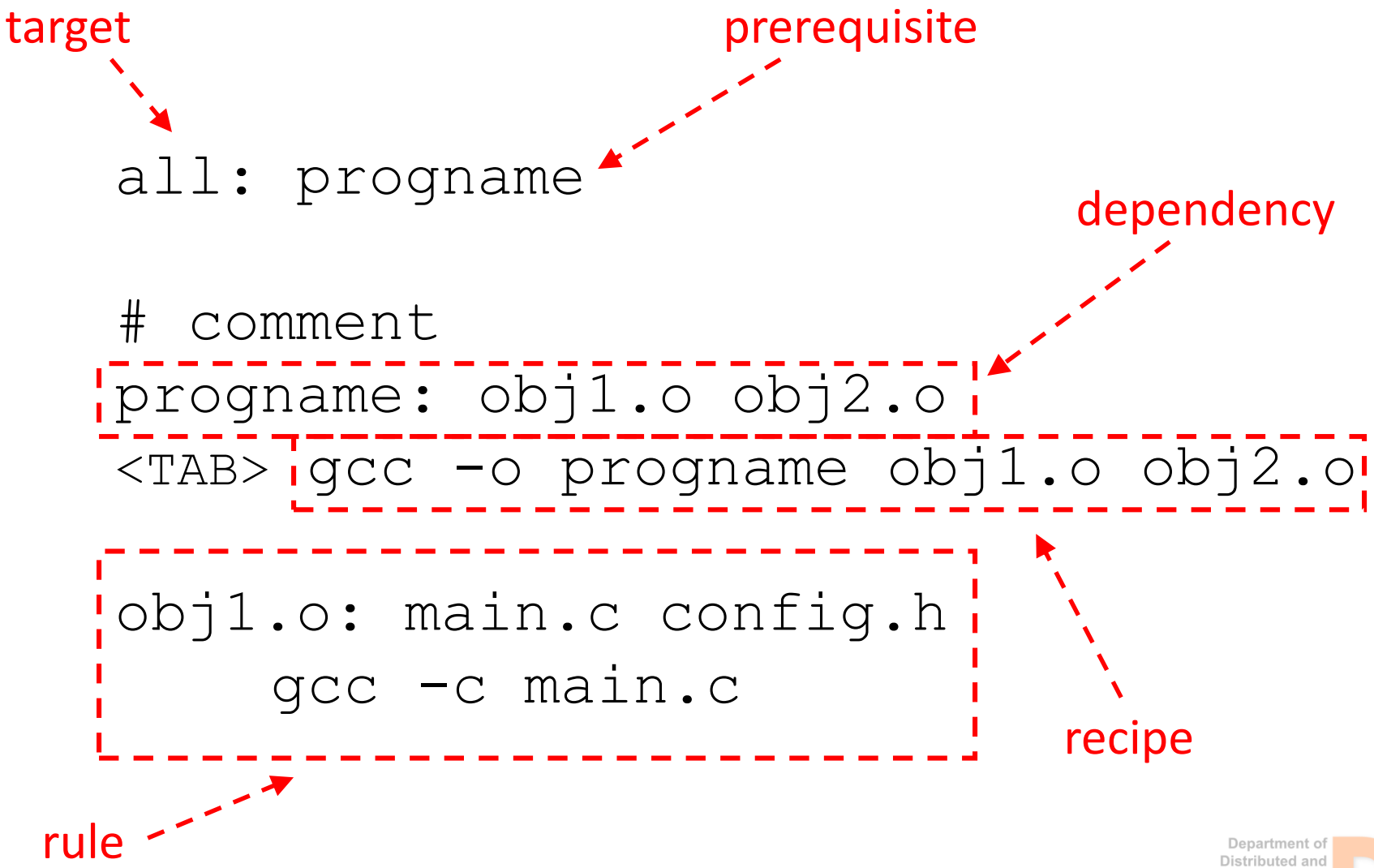
- Build script: `Makefile`

Key concepts

- Target
  - Entity to be built: executable program, object file (.o), distribution package (.tgz)
  - Action to be done: clean, build all, prepare something
- Prerequisite
  - Entity that must be **available** and **up-to-date** before the associated target is fulfilled during the build process
- Rules
  - Dependencies between targets and prerequisites
  - Commands that fulfill targets (build entities, ...)

Department of Distributed and Dependable Systems D3S

Makefile: example



Build process with Make

- Running
  - `make` *`target`*
  - `make` // default target
- Two steps
  - Construction of the build tree
    - Root node: target given by the user
    - Leaf nodes: available prerequisites
  - Processing rules in the tree

Task 1

- Download the “sockets” program
  - http://d3s.mff.cuni.cz/teaching/software_development_tools/files/sockets.tgz
  - Contains sources for a simple network client and server
  - Both have an UDP and TCP variant
    - Select using the parameter “`-u`”

- Write the `Makefile` for the “sockets” program
  - Useful targets: `all`, `clean`, program binaries, object files
  - Dependencies between entities (.o ➔ binary, .c ➔ .o, etc)
  - See the attached script `build.sh` for commands that can be used to compile source files
  - See `clean.sh` for commands to remove binaries and intermediate object files

Department of Distributed and Dependable Systems D3S

Variables

```
objects = obj1.o obj2.o main.o \
          utils.o network.o gui.o

all : progname

progname: $(objects)
    gcc -o prog $(objects) -lcommon
```

Note: wildcard expansion is quite tricky (manual, section 4.4)

Phony targets

- When the target does not represent any file

```
.PHONY : clean

clean :
	rm *.o
```

Guidelines

- Use built-in variables
  - `CC` // C compiler (gcc)
  - `CFLAGS` // C compiler flags
  - `CXX` // C++ compiler (g++)
  - `CXXFLAGS` // C++ compiler flags
  - ... and many more
- Use standard targets
  - `all`, `clean`, `distclean`, `install`

How to use built-in variables

- Define recipes properly

```
$(CC) $(CFLAGS) -c main.c
```

- Set flags when running Make

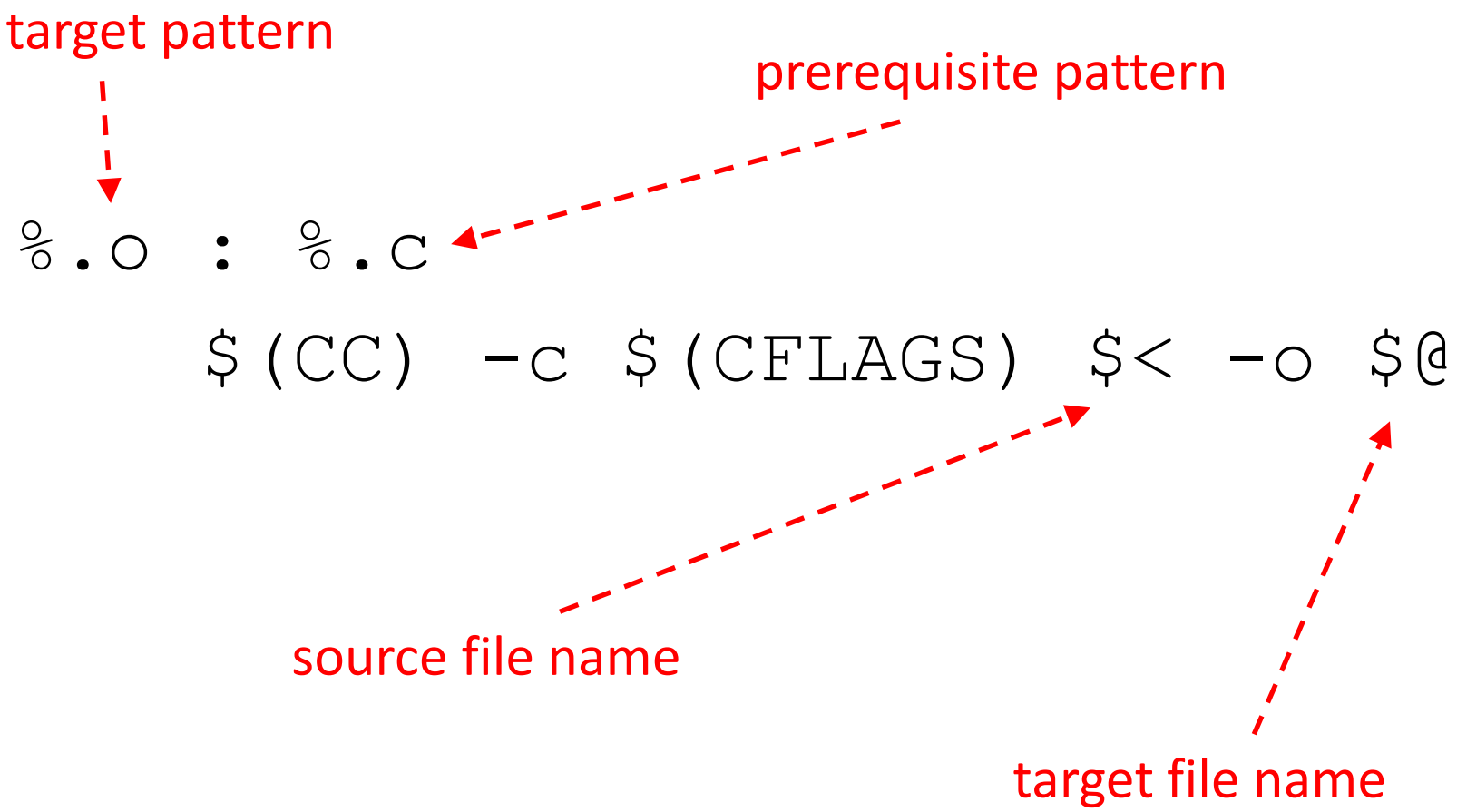
```
CFLAGS=-O2 make
```

Static pattern rules

```
objects = main.o util.o network.o

$(objects): %.o : %.c
	$(CC) -c $(CFLAGS) $< -o $@
```

Implicit rules



Task 2

- Rewrite the `Makefile` for “sockets”
- Eliminate duplication using these features:
  - Variables (built-in, custom)
  - Phony targets
  - Implicit rules
  - Static patterns
- Use dependencies between targets properly
- Respect common guidelines (best practices)

Recursive invocations (subdirectories)

```
SUBDIRS = src doc

.PHONY: subdirs $(SUBDIRS)

subdirs: $(SUBDIRS)

$(SUBDIRS):
	$(MAKE) -C $@
```

Department of Distributed and Dependable Systems

Two flavors of variables

- Recursively expanded

```
objects = $(core_objs) $(server_objs)
core_objs = tcp.o udp.o
server_objs = srv/main.o
```

- Simply expanded

```
$(core_objs) := tcp.o udp.o
objects := $(core_objs) srv/main.o
```

Substitutions

```
objects := main.o client.o server.o

sources := $(objects:.o=.c)
```

OR

```
sources := $(objects:%.o=%.c)
```

Operations with variables and values

- Appending

```
objects = main.o util.o
objects += network.o
```

- Functions

```
$(subst from,to,text)
$(patsubst pattern,replacement text)
$(filter pattern1 ... patternN,text)
$(dir path1 ... pathN)
$(basename path1 ... pathN)
$(suffix path1 ... pathN)
```

Automatic variables

- Target: `$@`
- First prerequisite: `$<`
- All prerequisites: `$^`

Task 3

- Try more advanced features
  - Recursive invocation
  - Pattern substitutions
  - Text processing functions
  - Automatic variables

Other advanced features

- Order-only prerequisites
- Automated generating of files that capture prerequisites (suffix `.d`)
  - Good support by compilers
  - Fallback: `makedepend` tool
- Parallel execution
- Conditional directives

- ... and many more

- See the documentation for GNU Make

Limitations

- Portability over different Unix-like systems
  - Library functions in C (issues with compatibility)
  - Environment: shell, utilities (Awk, sed, grep, …)
- Hard to maintain complex `Makefiles`
- Writing rules by hand can be tedious

- Solution: GNU build system (Autotools)
  - Tools: Autoconf, Automake, Libtool, gettext
  - De-facto standard in the open-source world

```
./configure ; make ; make install
```

Department of Distributed and Dependable Systems

Links

- http://www.gnu.org/software/make/
- http://www.gnu.org/software/make/manual/

- https://docs.microsoft.com/cs-cz/cpp/build/nmake-reference?view=vs-2017

Homework

- Assignment
  - http://d3s.mff.cuni.cz/~parizek/teaching/sdt/
- Deadline
  - 12.11.2018 / 13.11.2018