doit – automation tool

Bringing the power of build-tools to execute any kind of task

Eduardo Schettino

About me

- Brazil
- Beijing – China
- Exoweb http://www.exoweb.net

Talk overview

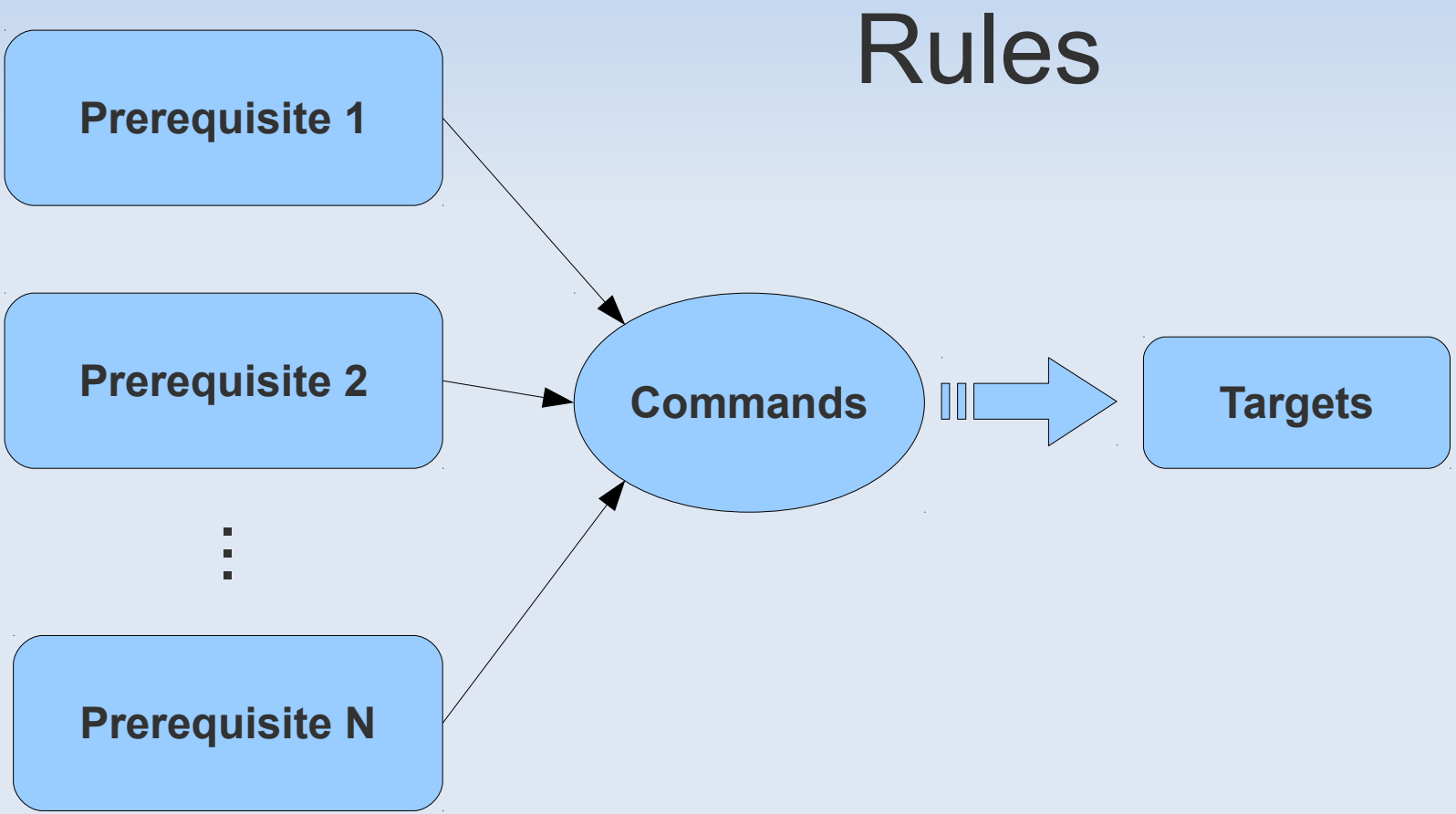
- build-tools (*make*)
- Who needs a build-tool?
- *doit*
- Questions & Answers

build-tools

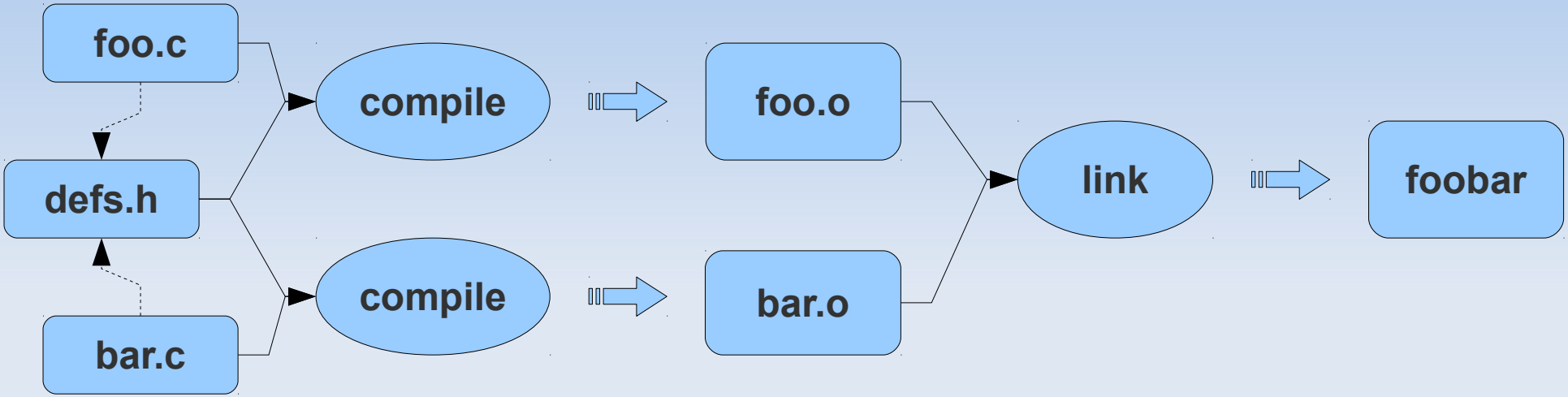
Anything worth repeating is worth automating

- manage repetitive tasks and their dependencies
- speed-up development (faster turn-around)
- 1977: Make (C & other compiled languages)

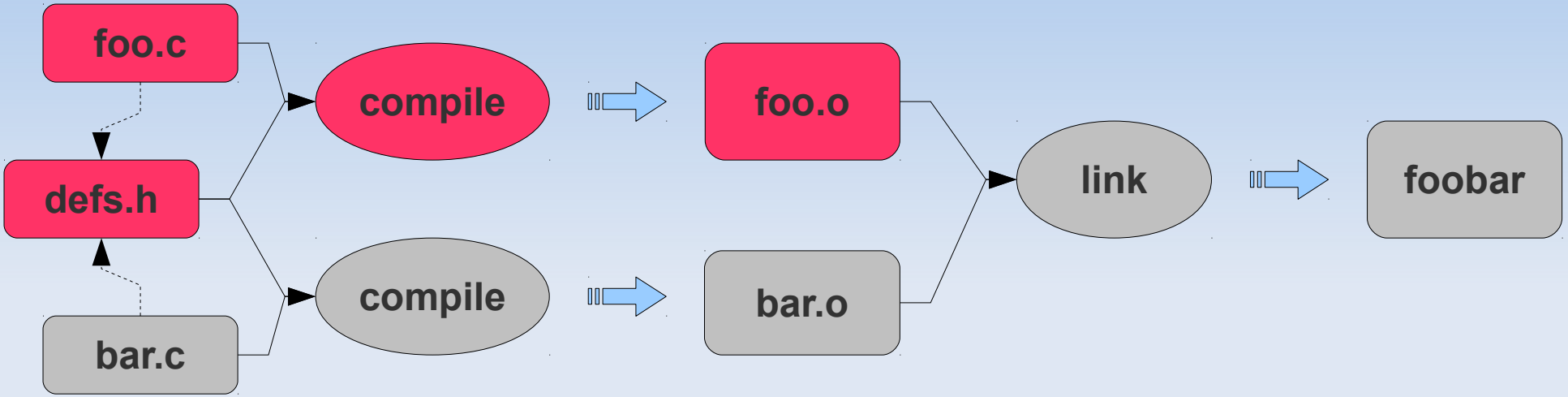
make model


Rules
Prerequisite 1
Prerequisite 2
Prerequisite N
Commands
Targets

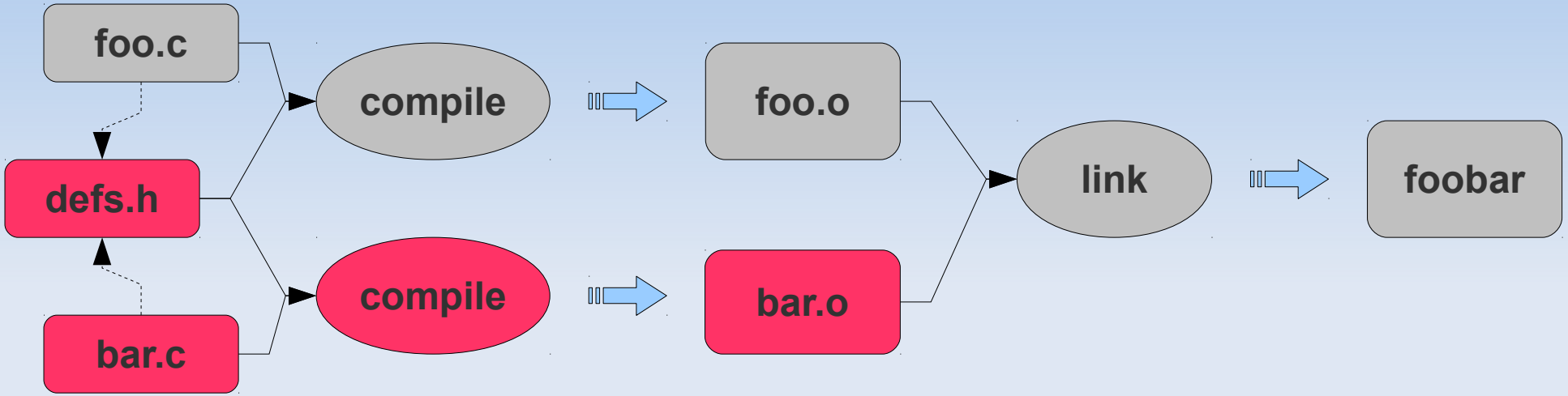
C project



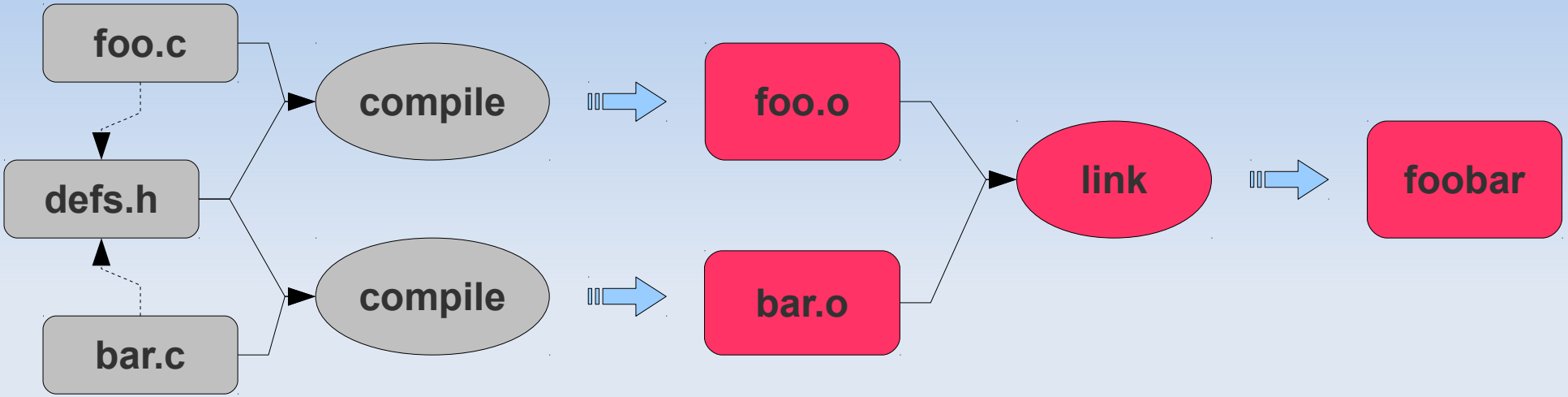
1 - compile foo



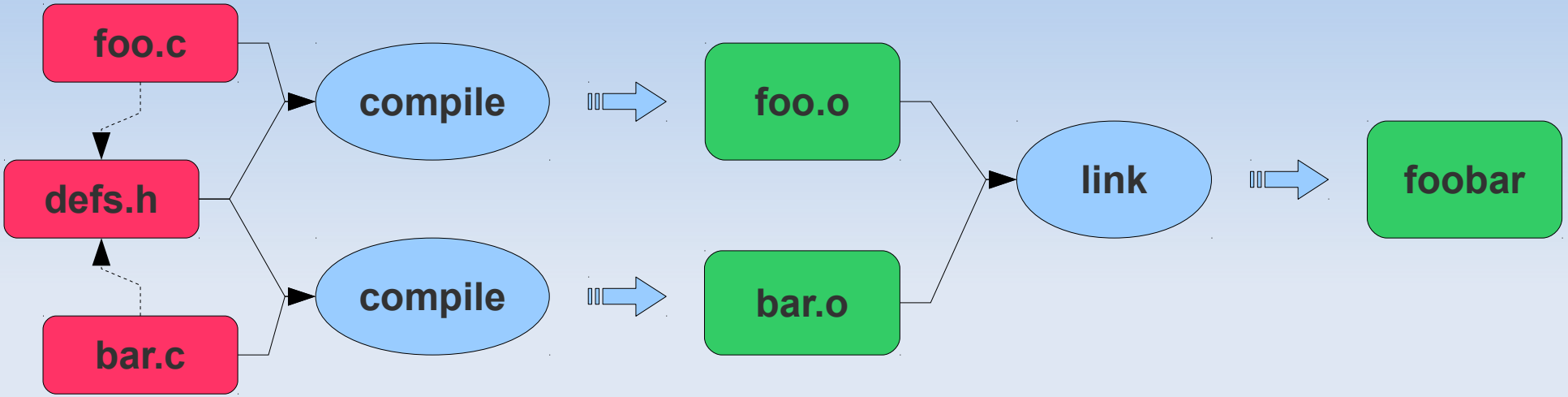
2 - compile bar



3 - foobar

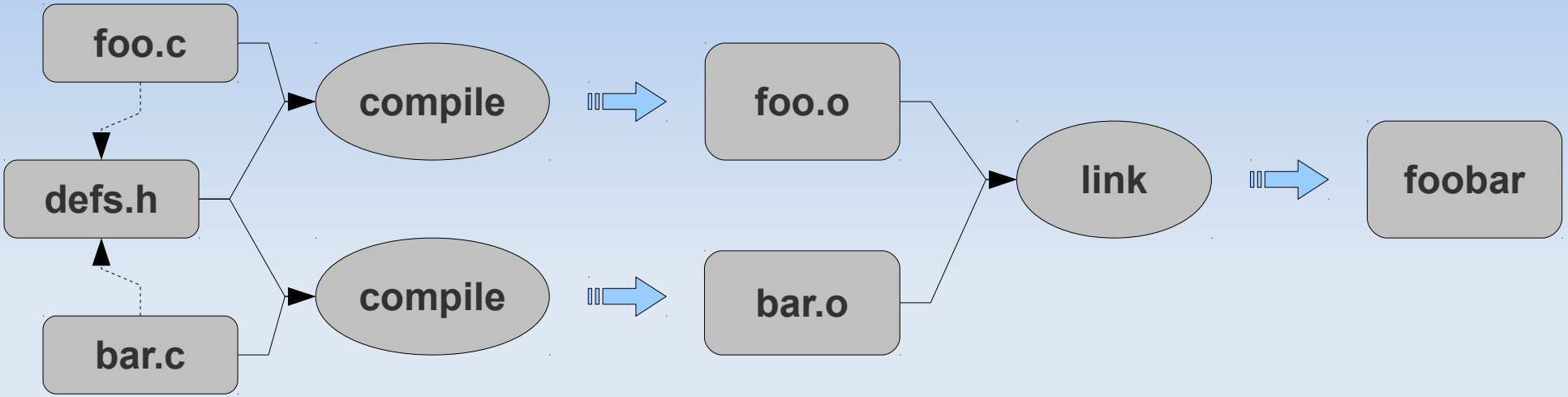


initial build



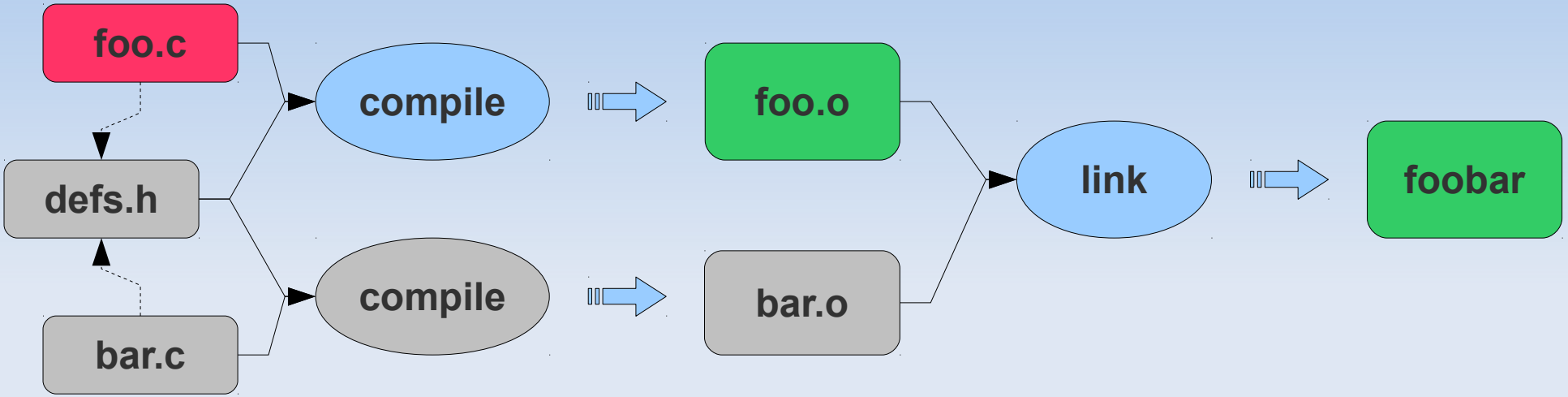
All 3 operations executed

no-op rebuild



No changes. No operation executed

incremental rebuild



foo.c changed. 2 operations executed

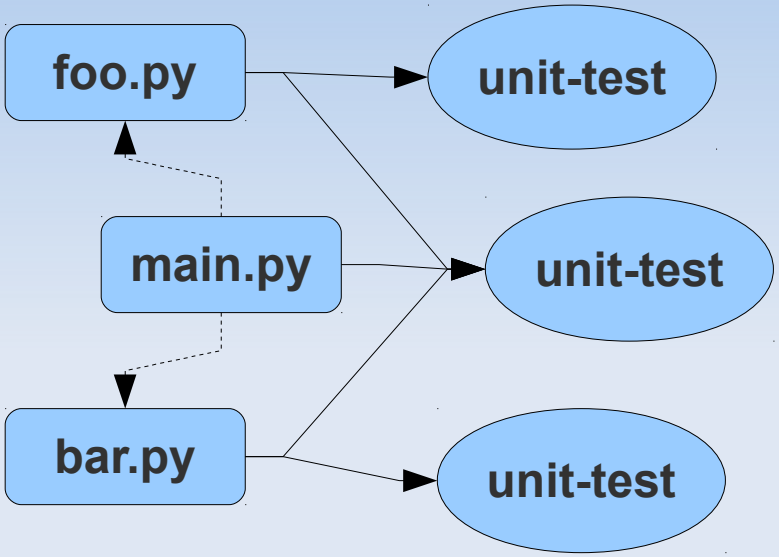
how it works?

- file time-stamp comparison
- if any of the prerequisite files were modified after last modification on target => re-execute
- if target is **up-to-date** => skip execution

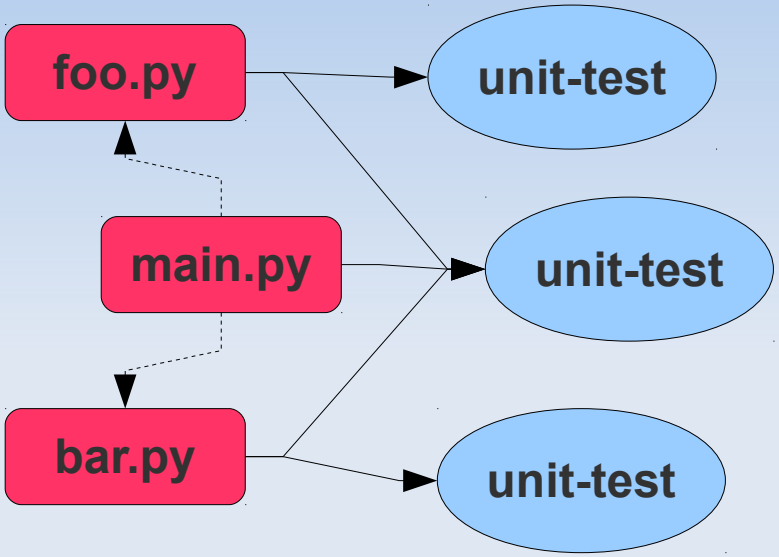
who needs a build-tool?

- dynamic language => no compilation => static checkers + unit-tests
- functional tests (DB + web) are slow
- maintenance tasks

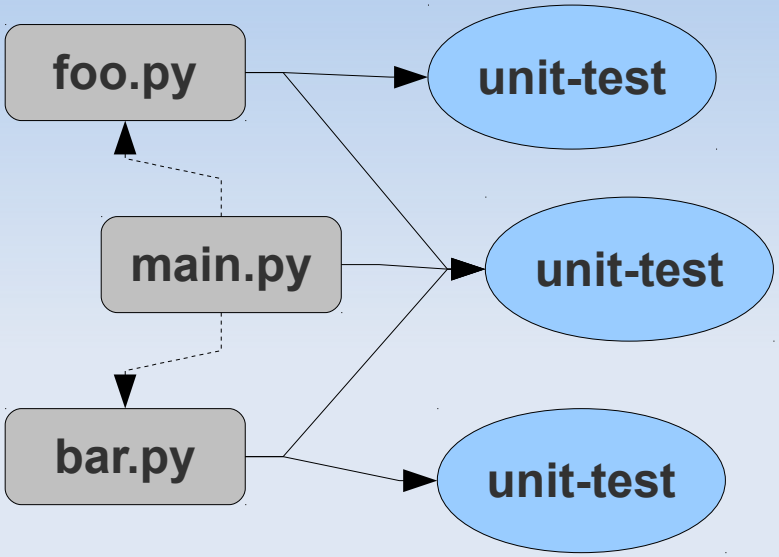
python project



first run



no-op run???



- no modifications on source code
- all tests are executed again!
- no targets to compare time-stamps...

make problems

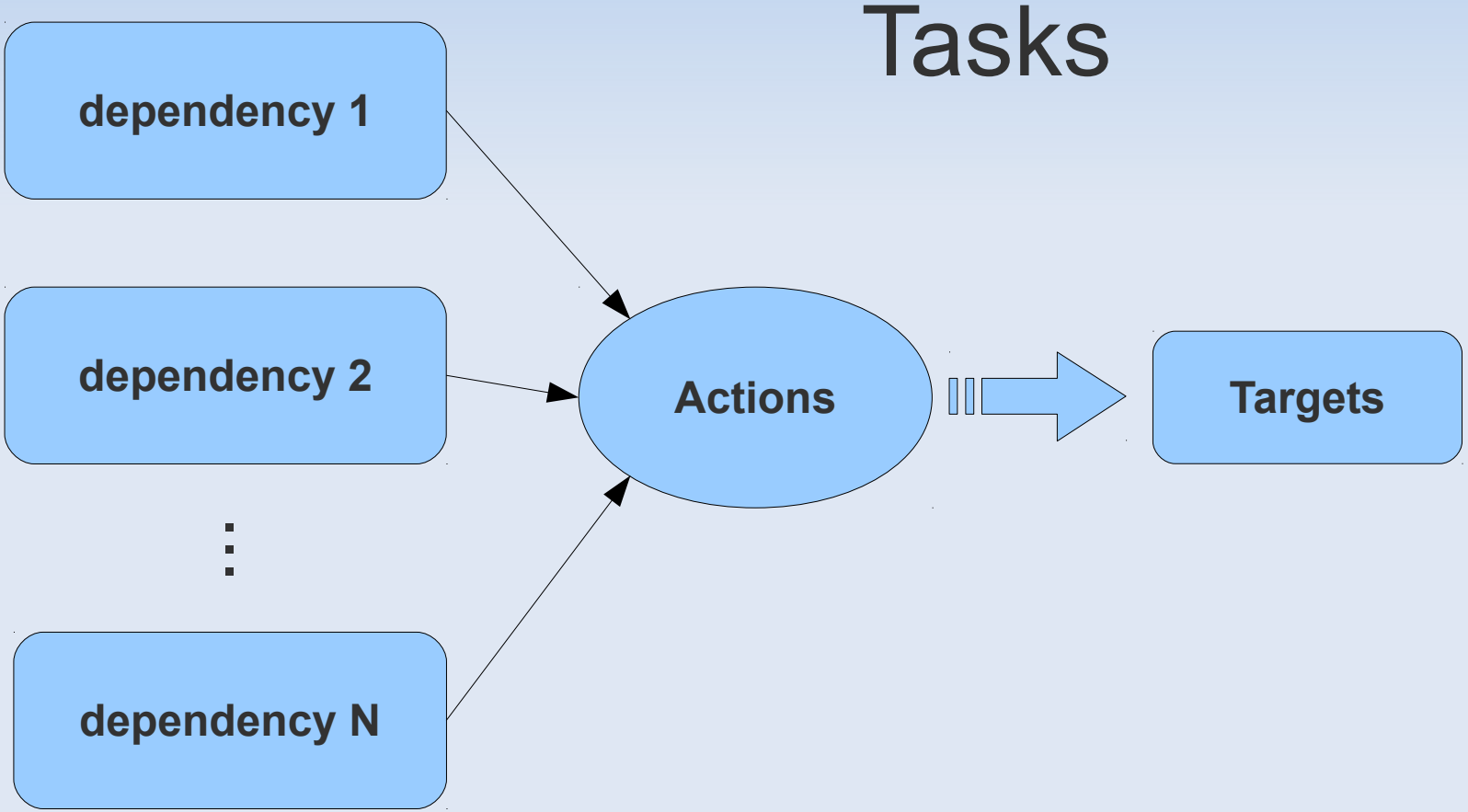
- ad-hoc language (hard to use, debug, ...)
- time-stamp based (fragile)
- restricted to operations that create files
- not good on dynamic creation of rules

doit goals

- more flexible model than traditional build-tools
- “real” language => python
- get out of your way

doit model

- based on “Tasks”, focus on “actions”



how it works?

- same principle but...
- use a “db” file to save info from successful execution
- tasks are defined on a “**dodo**” file, a plain python module
- i.e. does not require targets to check if task is **up-to-date**

Hello World

```
def task_hello():


    return {'actions':
       ['echo Hello World > hello.txt']}
```

Hello World

A task generator ...

```
def task_hello():


    return {'actions':
        ['echo Hello World > hello.txt']}
```

Hello World

A task generator ...

```
def task_hello():
```

... returns a dictionary with task meta-data

```
    return {'actions':
            ['echo Hello World > hello.txt']}
```

Hello World

A task generator ...

```
def task_hello():
```

... returns a dictionary with task meta-data

```
    return {'actions':
        ['echo Hello World > hello.txt']}
```

strings are shell commands

python actions

a plain python function

```
def pyhello():
    with open('hello.txt', 'w') as hello_file:
        hello_file.write("Hello Python\n")


def task_hello():
    return {'actions': [(pyhello,)]}
```

python actions

a plain python function

```
def pyhello():
    with open('hello.txt', 'w') as hello_file:
        hello_file.write("Hello Python\n")
```

tuple (callable, args, kwargs)

```
def task_hello():
    return {'actions': [(pyhello,)]}
```

compile

```
def task_compile():
    return {'actions': ["cc -c main.c"],
            'file_dep': ["main.c","defs.h"],
            'targets': ["main.o"]
            }
```

dependencies

targets

sub-tasks

define task meta-data

```
def task_create_file():
    for i in range(3):
        filename = "file%d.txt" % i


        yield {'name': filename,
               'actions':
                 ["touch %s" % filename]}
```

sub-tasks

define task meta-data

```
def task_create_file():
    for i in range(3):
        filename = "file%d.txt" % i


        yield {'name': filename,
               'actions':
                 ["touch %s" % filename]}
```

“yield” to create multiple tasks

sub-tasks

define task meta-data

```
def task_create_file():
    for i in range(3):
        filename = "file%d.txt" % i

        yield {'name': filename,
               'actions':
                 ["touch %s" % filename]}
```

required for sub-tasks

“yield” to create multiple tasks

sub-tasks

define task meta-data

```
def task_create_file():
    for i in range(3):
        filename = "file%d.txt" % i

        yield {'name': filename,
               'actions':
                 ["touch %s" % filename]}
```

required for sub-tasks

“yield” to create multiple tasks

task dependency

```
def task_foo():
    return {'actions': ["echo foo"]}
def task_bar():
    return {'actions': ["echo bar"]}

def task_mygroup():
    return {'actions': None,
            'task_dep': ['foo', 'bar']}
```

tasks that must be executed before this task

result dependency

```
def task_which_version():
   return {'actions':
 ['bzr version-info --custom –template="{revno}\n"']}
```

```
def task_do_something():
   return {'actions': ['echo "TODO: send an email"'],
           'result_dep': ['which_version']}
```

check result from another task instead of a file content

result dependency

```
def task_which_version():
   return {'actions':
 ['bzr version-info --custom –template="{revno}\n"']}
```

task “result” is string returned by action

```
def task_do_something():
   return {'actions': ['echo "TODO: send an email"'],
           'result_dep': ['which_version']}
```

check result from another task instead of a file content

execution

```
DOIT_CONFIG = {'default_tasks': ['t3']}
```

- by default all tasks are executed
- controlled by DOIT_CONFIG default_tasks

```
$ doit
.  task3
$ doit task2 task1:foo
.  task2
.  task1:foo
```

up-to-date ?

- no file dependencies were modified
- no result dependencies were modified
- all targets exist
- compares => timestamp, size, checksum

up-to-date output

```
$ doit
.  compile
```

. (dot) => executed

```
$ doit
-- compile
```

-- (dashes) => skipped

```
$ rm main.o
$ doit
.  compile
```

environment setup

```
def task_start_server():
    for name in ('serverX', 'serverY'):
        yield {'name': name,
               'actions': [(start, (name,))],
               'teardown': [(stop, (name,))],
               }



def task_test_A():
    return {'actions':['echo fun_test_a'],
            'setup': ['start_server:serverX'],
           }
```

test_A requires serverX to be running
start serverX only if test_A not up-to-date

environment setup

```
def task_start_server():
    for name in ('serverX', 'serverY'):
        yield {'name': name,
               'actions': [(start, (name,))],
               'teardown': [(stop, (name,))],
               }
```

stop serverX after all tasks finish running

```
def task_test_A():
    return {'actions':['echo fun_test_a'],
            'setup': ['start_server:serverX'],
           }
```

test_A requires serverX to be running
start serverX only if test_A not up-to-date

calculated dependency

```
def task_mod_deps():
    return {'actions': [(print_deps,)],
            'calc_dep': ["get_dep"],
           }
```

dependencies are calculated on another task

```
def get_dep(mod):
    return {'file_dep': ['a', 'b', 'c']}
def task_get_dep():
    return {'actions':[(get_dep,)]}
```

calculated dependency

```
def task_mod_deps():
    return {'actions': [(print_deps,)],
            'calc_dep': ["get_dep"],
           }
```

dependencies are calculated on another task

```
def get_dep(mod):
    return {'file_dep': ['a', 'b', 'c']}
def task_get_dep():
    return {'actions':[(get_dep,)]}
```

returns dictionary with same keys as task-dict
(file_dep, task_dep, result_dep, calc_dep)

other task features

- clean => clean actions ($doit clean <tasks>)
- doc => ($doit list)
- params => get parameters from command line
- getargs => get values computed in different tasks

other runner features

- verbosity => capture/display stdout/stderr
- title => controls output (task name)
- custom_reporters => complete output control
- use wildcard to select tasks

parallel execution

- parallel execution of tasks in multiple processes
- uses multiprocessing lib
- subject to same limitations...

auto execution

- long running process
- watches for file modifications and automatically re-execute outdated tasks
- works on linux and mac
- TDD, never leave the editor screen :)

thanks

- Questions ?

- website: http://python-doit.sourceforge.net