Операционные системы (6 семестр)

Лекция 3.1. Управление программным обеспечением

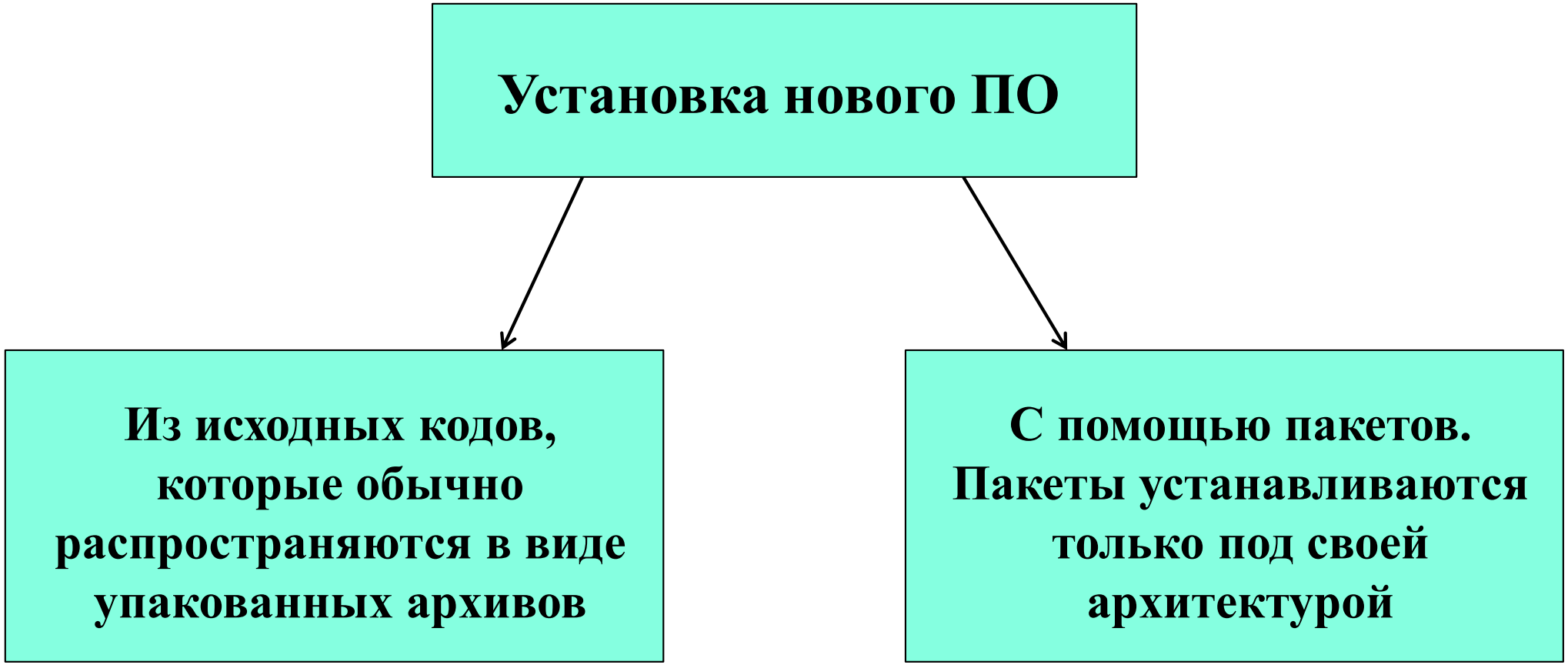
Ассистент, к.т.н. Митричев Иван Игоревич

Москва 2018

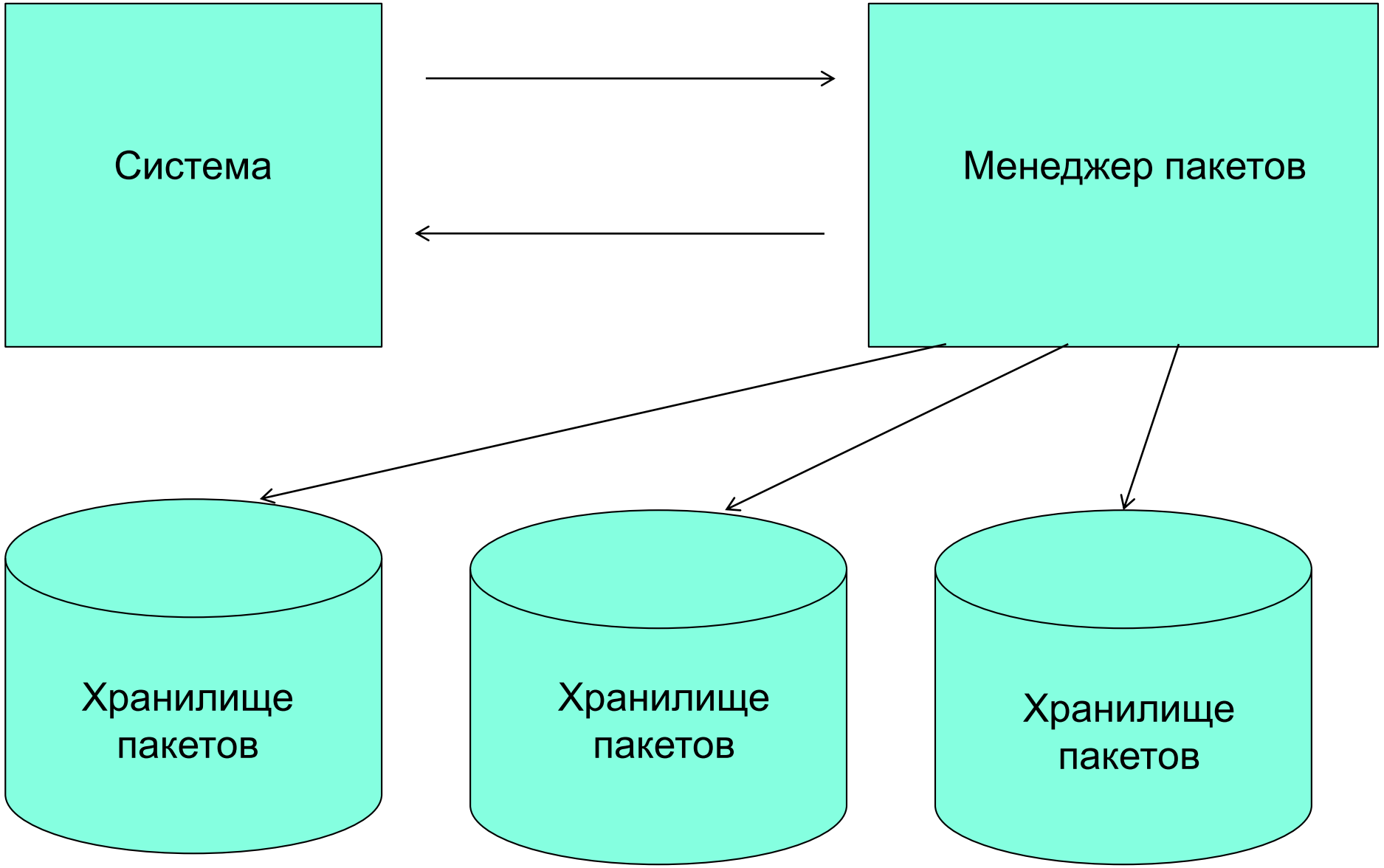
План лекции

- Управление программным обеспечением.
- Системы управления пакетами.
- Менеджеры пакетов rpm, yum, apt.
- Преимущества и недостатки системы управления.
- Стандартные расположения установки программ.
- Конфликт пакетов и пути его решения.
- Сборка программного обеспечения из архивов с исходным кодом.
- Утилиты configure, make и файл сборки Makefile.
- Частые ошибки при компиляции из исходных кодов.
- Управление библиотеками.

Управление программным обеспечением



Место менеджера пакетов



Системы управления пакетами (менеджеры)

- *RPM Package Manager (rpm)*
- *Synaptic (Ubuntu)*
- *Система управления пакетами apt (Ubuntu, Mint)*
- *Система управления пакетами yum (RedHat) и dnf (Fedora)*
- *Система управления пакетами Debian (dpkg)*
- *Система портов Gentoo, Portage*
- *Система управления пакетами Slackware*
- *RpmDrake*

Менеджер пакетов rpm

Синтаксис утилиты:
$rpm [режим опции] [пакет].rpm

Режимы работы:
-q – получение информации (запрос);
-i – установка;
-U –обновление;
-V – проверка пакетов;
-e – удаление.

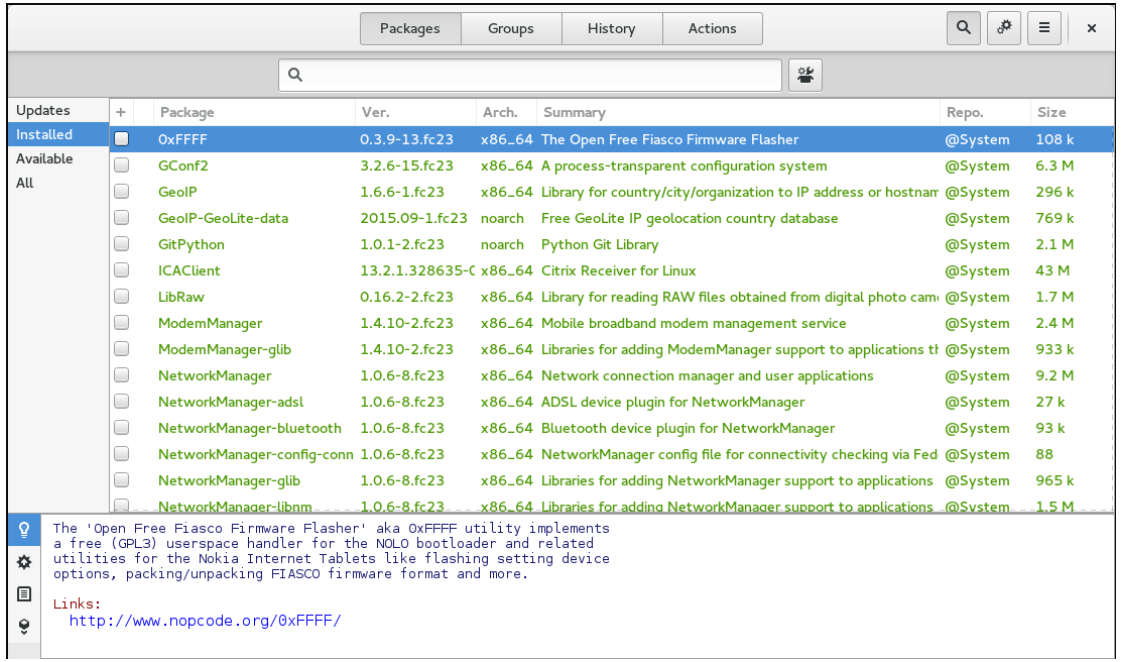
Основные опции:
-v – показать подробную информацию.
-i – получить информацию о пакете (**-qi**);
-l – список файлов пакета (**-ql**);
-c – список конфигурационных файлов пакета (**-qc**);
-a – список пакетов, установленных в системе;
--force – выполнять действие принудительно;
--nodeps – не проверять зависимости;
--test – проверить последствия удаления или установки пакета.

Один из существенных недостатков rpm – программа не может разрешать зависимости. Если в системе отсутствует необходимый пакет, то установка завершится ошибкой.

Менеджер пакетов yum

Основные команды **yum**:

- **#yum help** – список команд и опций;
- **#yum list** – список названий пакетов из репозиториев;
- **#yum list installed** – список установленных пакетов.
- **#yum repolist** – список подключенных репозиториев;
- **#yum install [пакет]** – установить пакет;
- **#yum remove [пакет]** – удалить пакет;
- **#yum update [пакет]** – обновить пакет;
- **#yum info [пакет]** – вывести информацию о пакете;
- **#yum search [строка]** – найти пакет по строке в описание.



Существует графический интерфейс для yum – **Yum Extender**.
http://www.yumex.dk

Менеджер пакетов apt

Синтаксис утилиты:

$ apt [опции] [команда] [пакет]

Полезные опции для apt:

- **-t** – версия релиза для которой устанавливать пакет;
- **-f** – выполнить операцию принудительно;
- **-o** – строка конфигурации.

Основные команды apt:

- **download** – скачать пакет без установки;
- **update** – обновление информации о пакетах в репозиториях;
- **upgrade** – обновление системы без удаления пакетов;
- **search** – поиск пакетов в локальной базе;
- **show** – просмотр информации о пакете;
- **install** – установка пакета;
- **remove** – удаление пакета;
- **purge** – полное удаление пакета (вместе с конфигурационными файлами).

Другие утилиты установки пакетов

•**dselect** – программа управления с графическим интерфейсом. Является старейшим фронтендом к **dpkg**. На сегодняшний момент практически полностью вытеснена **Advanced Packaging Tool**.

•**aptitude** – оболочка с графическим интерфейсом для **Advanced Packaging Tool**.

•**dpkg** – основное программное обеспечение системы управления пакетами **Debian**, а так же систем со схожей архитектурой.
•**$dpkg –i [пакет].deb** – установка пакета **.deb**.

•**synaptic** – графический интерфейс для системы управления пакетами **apt** или проектами **Debian**.

•**apt-get** – индивидуальное управление пакетами и источниками. Утилита устарела, в настоящее время используется **apt**.

Стандартное расположение установленных программ

Пользовательские программы обычно устанавливаются в директорию **/usr**, при этом программа делится на части следующим образом:

- **/usr/bin** – исполняемые файлы программ;
- **/usr/sbin** – исполняемые файлы запускаемые с правами администратора;
- **/usr/lib** – библиотека программы;
- **/usr/share** – остальные файлы программы.

ПО, собранное самостоятельно из исходных кодов устанавливается в **/usr/local**.

Сборка в среде Linux

Система сборки на **Linux** основана на **GNU**.

Обычно программы с исходным кодом распространяются в виде упакованных архивов (**tar.xz, tar.gz**). Первый шаг перед установкой – это **распаковка:**

$tar -Jxvf [пакет].tar.xz

$tar -zxvf [пакет].tar.gz

Дальше запускается файл конфигурации:

./configure

Скрипт оболочки **configure** будет пытаться подобрать правильные значения переменных, использующихся в процессе компиляции. Эти переменные необходимы для создания **Makefile** в каждом каталоге пакета, набор заголовочных файлов, содержащих системные зависимости.

Сборка в среде Linux (продолжение)

Для компиляции используется команда **make**.
Компиляцию можно производить в параллельном режиме, используя опцию **–j**, например:
make **-j4** – параллельная компиляция на 4 ядрах процессора.

Переходим к установке:
make install

Стандартное расположение установленных программ, собранных из исходных кодов:
/usr/local

Удаление:
make uninstall – удаление установленных файлов (обратно make install)
make clean – удаление скомпилированных файлов в месте сборки (обычно, - очистка каталога build)

Пример makefile

```
# объявление переменной
# -c - собрать объектный файл. -Wall - включить все предупреждения
CFLAGS=-c -Wall

# цель по умолчанию (выполняется, если команда make вызвана без указания цели)
all: project

# это - цель сборки. После двоеточия указывают, от чего она зависит.
# Отсутствие зависимости позволяет не перестраивать весь проект при изменении одного файла с кодом.
project: 1.o 2.o 3.o
# в следующей строке обязателен отступ (TAB). Это команды (компиляции)
	$(CC) 1.o 2.o 3.o -o project

1.o: 1.cpp
	$(CC) $(CFLAGS) 1.cpp

2.o: 2.cpp
	$(CC) $(CFLAGS) 2.cpp

3.o: 3.cpp
	$(CC) $(CFLAGS) 3.cpp

# цель для очистки проекта (удаление скомпилированных файлов)
clean:
	rm -rf *.o project
```

Ошибки при компиляции из исходных кодов

Часто при компиляции из исходных кодов конфигуратор (**configure**) не может найти ту или иную библиотеку. Иногда названия библиотеки может не совпадать с названием пакета в репозитории. В таком случае можно 1) найти недостающую библиотеку
apt-cache search ключевое_слово
2) воспользоваться системами управления с графическим интерфейсом, например, **synaptic**, который выполнит поиск нужной библиотеки внутри пакетов.

Ошибки могут возникнуть при **конфликте пакетов друг с другом**, тогда для решения проблемы придётся вручную удалять наименее важные пакеты и проверить целостность оставшихся.

Управление библиотеками

Исполняемые программы в **Linux** делятся на два типа:

1. Статически скомпонованные.

+уже содержат в себе все необходимые библиотечные функции.

+не требуется предварительная установка необходимых компонентов.

-занимают больше места по сравнению с динамически скомпонованными.

2. Динамически скомпонованные.

+занимают меньше места по сравнению со статически скомпонованными.

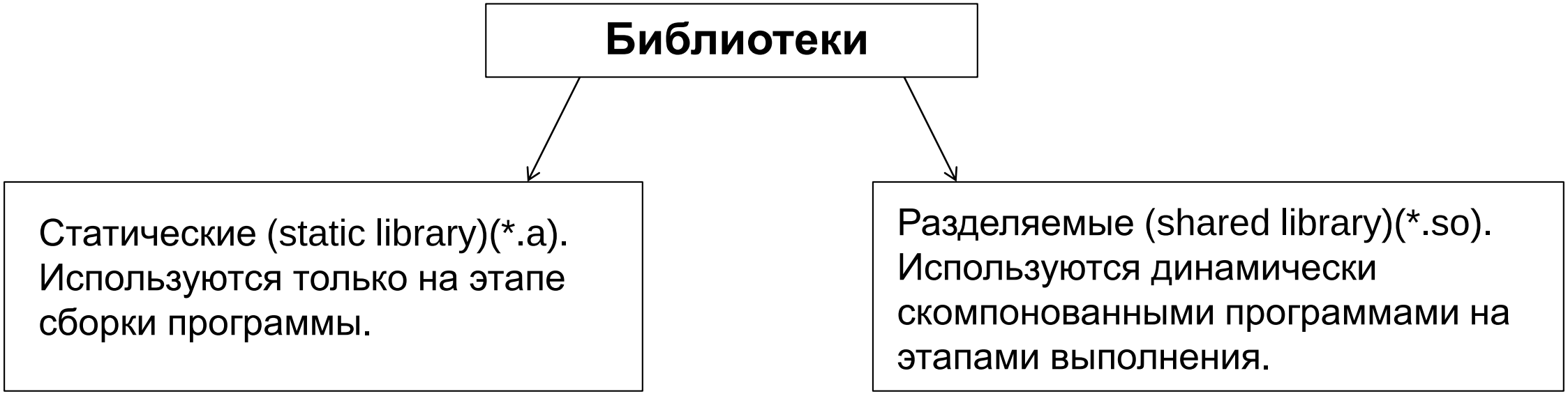
- требуют установку дополнительных компонентов.

Расположение библиотек в **Linux**:

/lib – 32-разрядные.

/lib64 – 64-разрядные

Типы библиотек



ldd [файл] – команда определит с какими библиотеками собран исполняемый файл.

Управлять версиями библиотек можно при помощи символичных ссылок **(ls –l /lib)**.

Ошибка запуска: отсутствие библиотек

Иногда ошибки могут возникать при запуске программ (отсутствие библиотек)
Не найдена разделяемая библиотека: Error loading shared library libopenblas.so.3: No such file or directory
Решение:

$ ldd «исполняемый файл, использующий разделяемые библиотеки»

В списке библиотек ненайденные помечены будут как «Not Found».

$ sudo apt install apt-file
$ apt-file update
$ apt-file find [название ненайденной библиотеки]
$ sudo apt install [название пакета]

Puppet

Puppet – кроссплатформенное клиент-серверное приложение, позволяющее централизованно управлять конфигурацией операционных систем и программ на нескольких компьютерах.