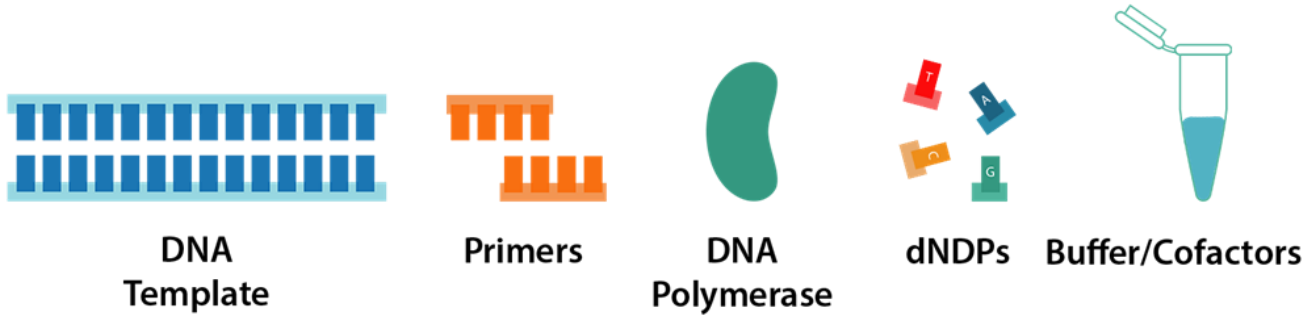
Базы данных

в биоинформатике

Полимеразная цепная реакция (ПЦР)

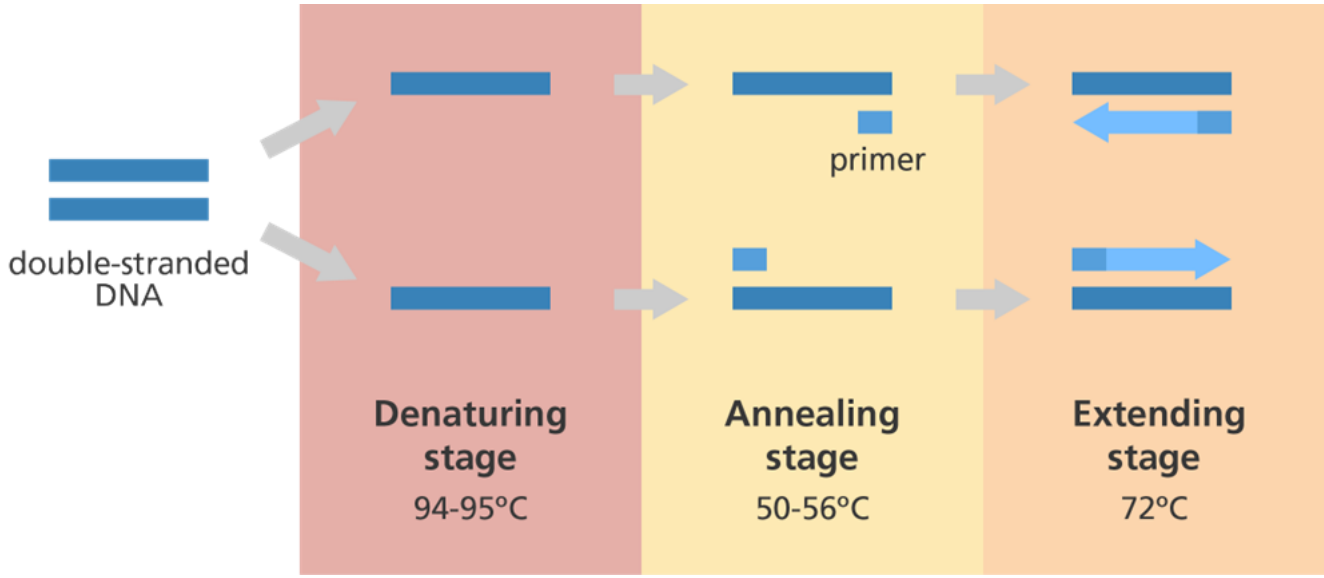
ПЦР - это экспериментальный метод молекулярной биологии, позволяющий добиться значительного увеличения определённых фрагментов нуклеиновой кислоты. ПЦР стоит в основе множества методик молекулярной диагностики, включая секвенирование. Основные компоненты:

1. Праймеры (15-30 п.о., комплементарны противоположным концам цепей искомого участка);
2. Термостабильная полимераза;
3. Смесь дезоксинуклеотидтрифосфатов (дНТФ);
4. Буфер;
5. Анализируемый образец.



Этапы ПЦР

1. Денатурация (разрыв водородных связей между цепями);
2. Отжиг (присоединение праймеров к мишени);
3. Элонгация (полимераза достраивает цепи ДНК с 3’-конца при помощи дНТФ).



Подбор праймеров

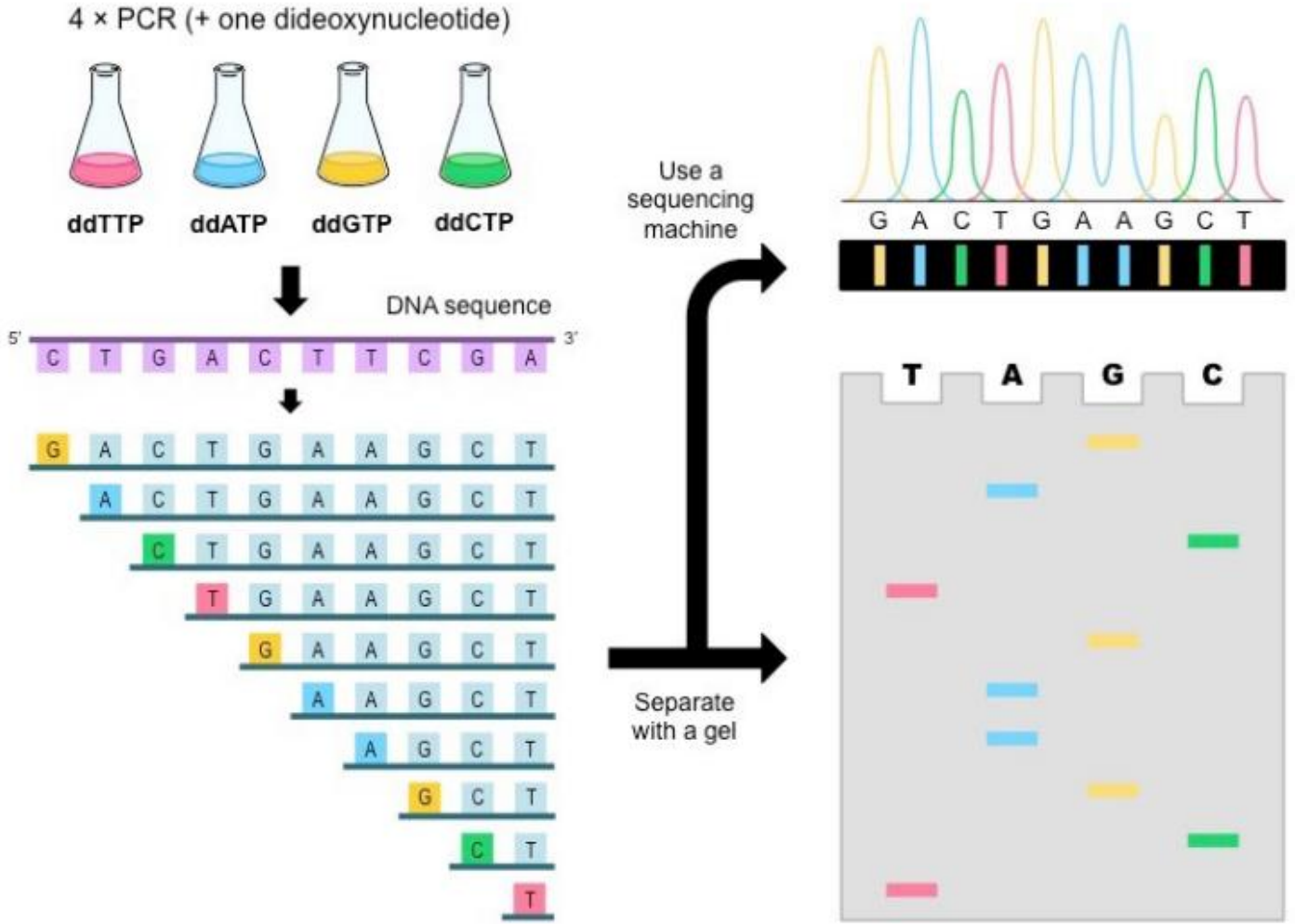
1. Длина праймера 18-22 п.о. (короче - страдает специфичность, длиннее - приходится делать температуру отжига слишком низкой);
2. Температура плавления праймеров должна мало отличаться, в пределах 52-65°C (температура отжига должна быть чуть ниже);
3. Доля GC 40-60% (в случае дальнейшего секвенирования - качество прочтения страдает; по три водородные связи);
4. Вторичные структуры (шпильки, димеры) не должны образовываться (желательно).

Для дизайна праймеров используют Primer3/NCBI Primer Designing Tool.

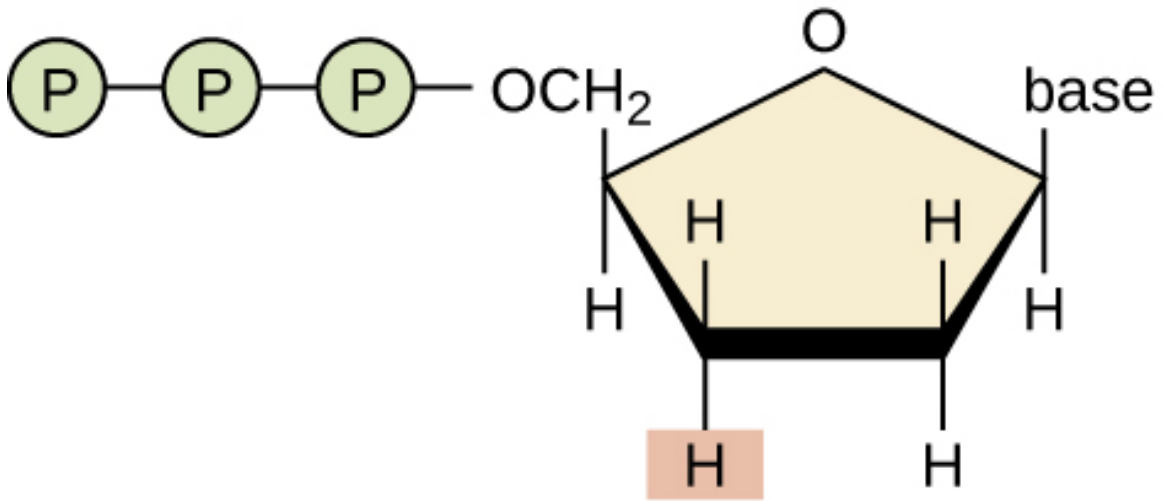
Секвенирование первого поколения

Реакция происходит в 4 “пробирках” в присутствии:

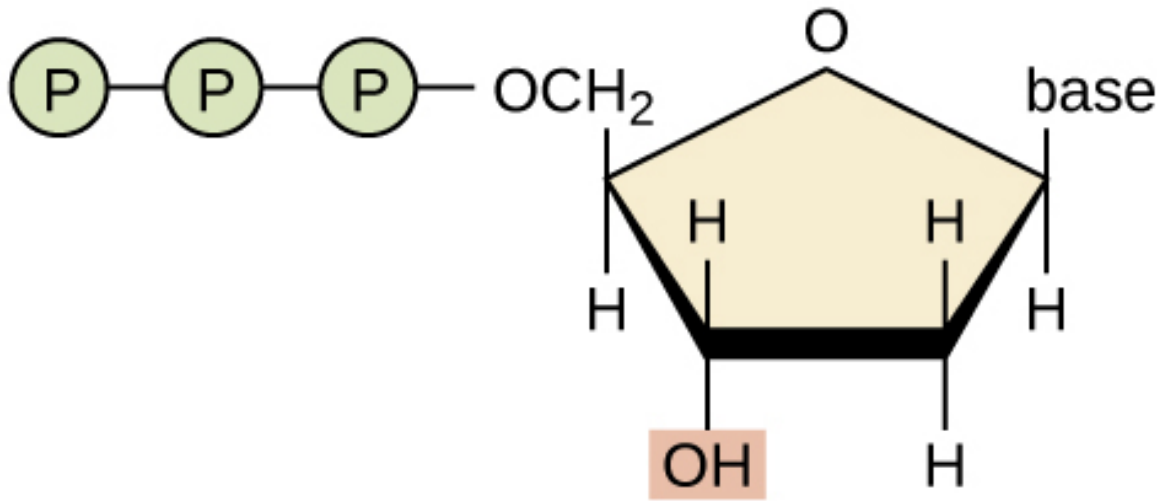
- ДНК-полимеразы,
- дНТФ (строительные блоки, все нуклеотиды),
- ди-дНТФ (блокируют синтез, в каждой пробирке - только один тип нуклеотидов).



Дезоксинуклеотиды и дидезоксинуклеотиды



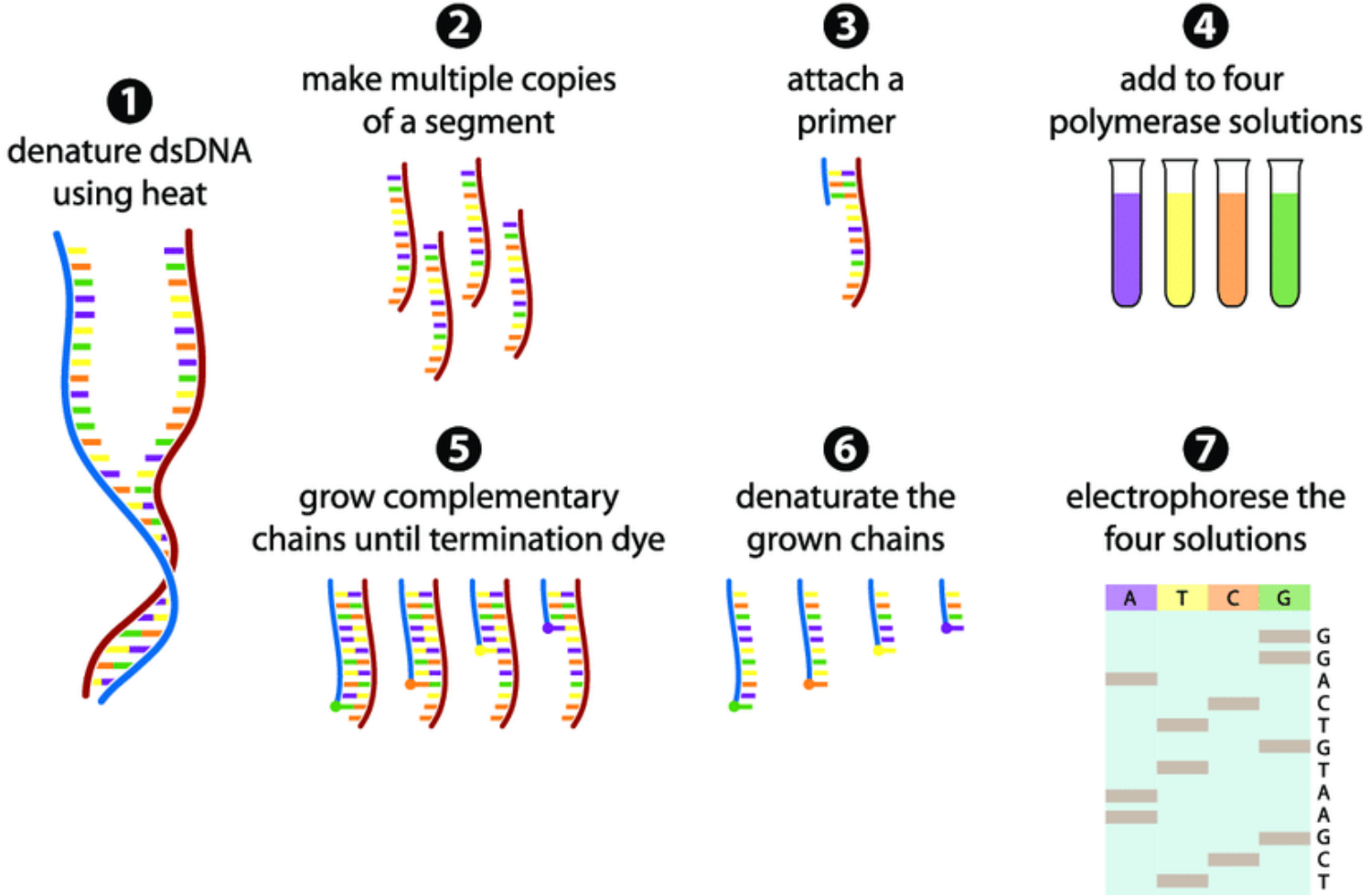
dideoxynucleotide (ddNTP)



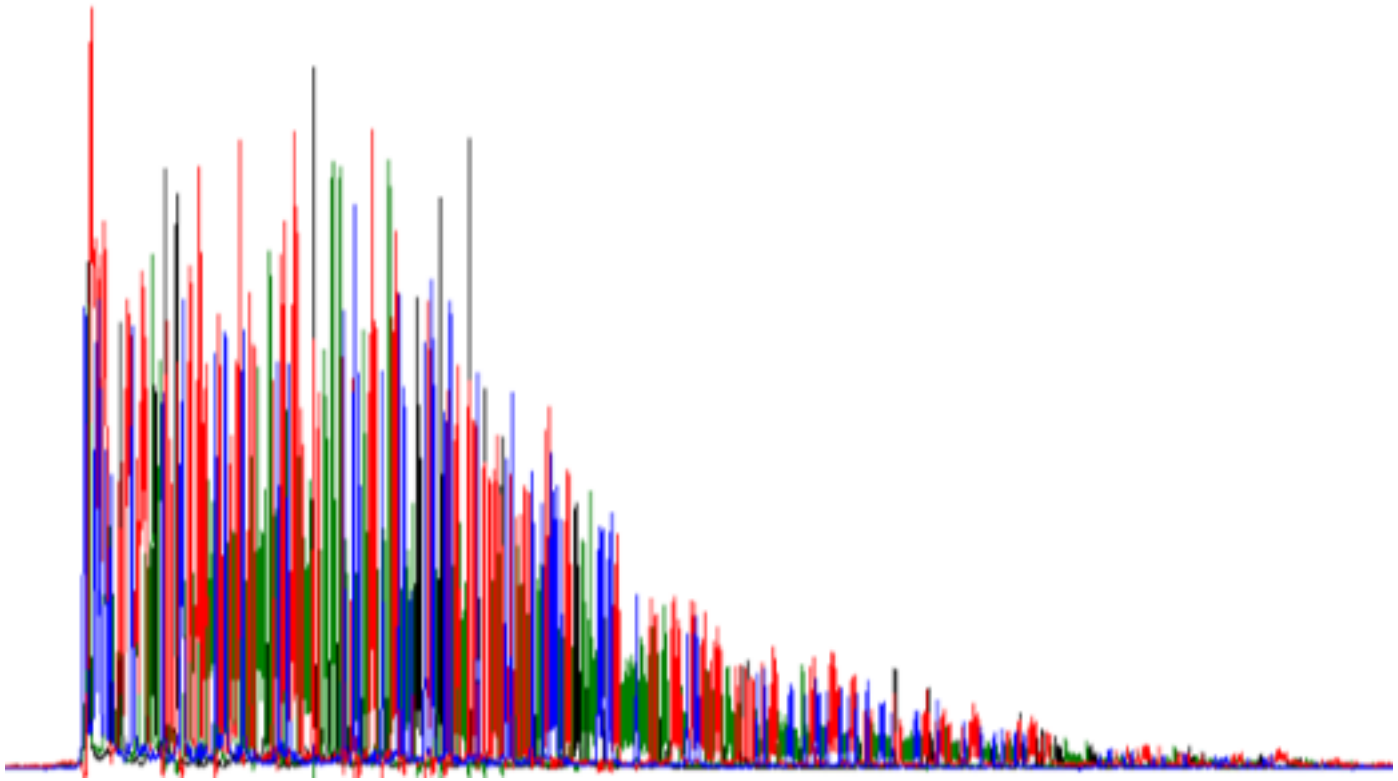
deoxynucleotide (dNTP)

Секвенирование первого поколения

Реакция происходит в 4 “пробирках”, в каждой из которых она останавливается на определённом нуклеотиде. Чем длиннее достроенная цепь, тем вероятнее остановка синтеза. Поэтому к “концу” прочтения сигнал угасает. <1000 нуклеотидов!

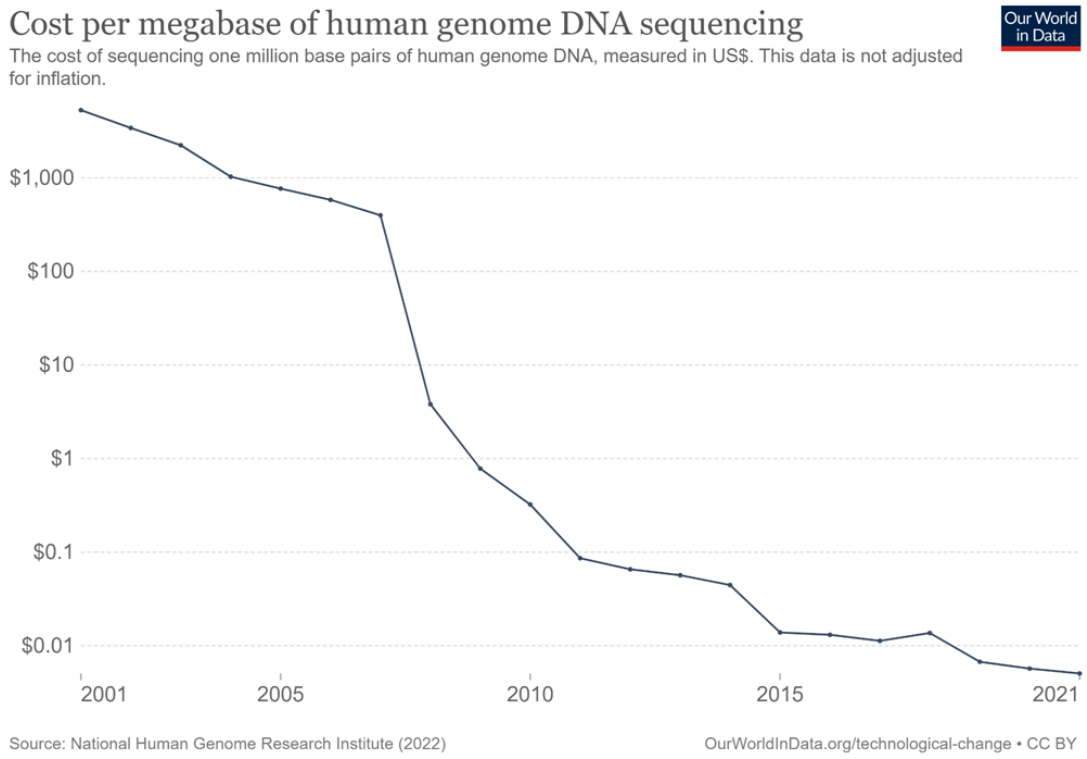


Угасание сигнала

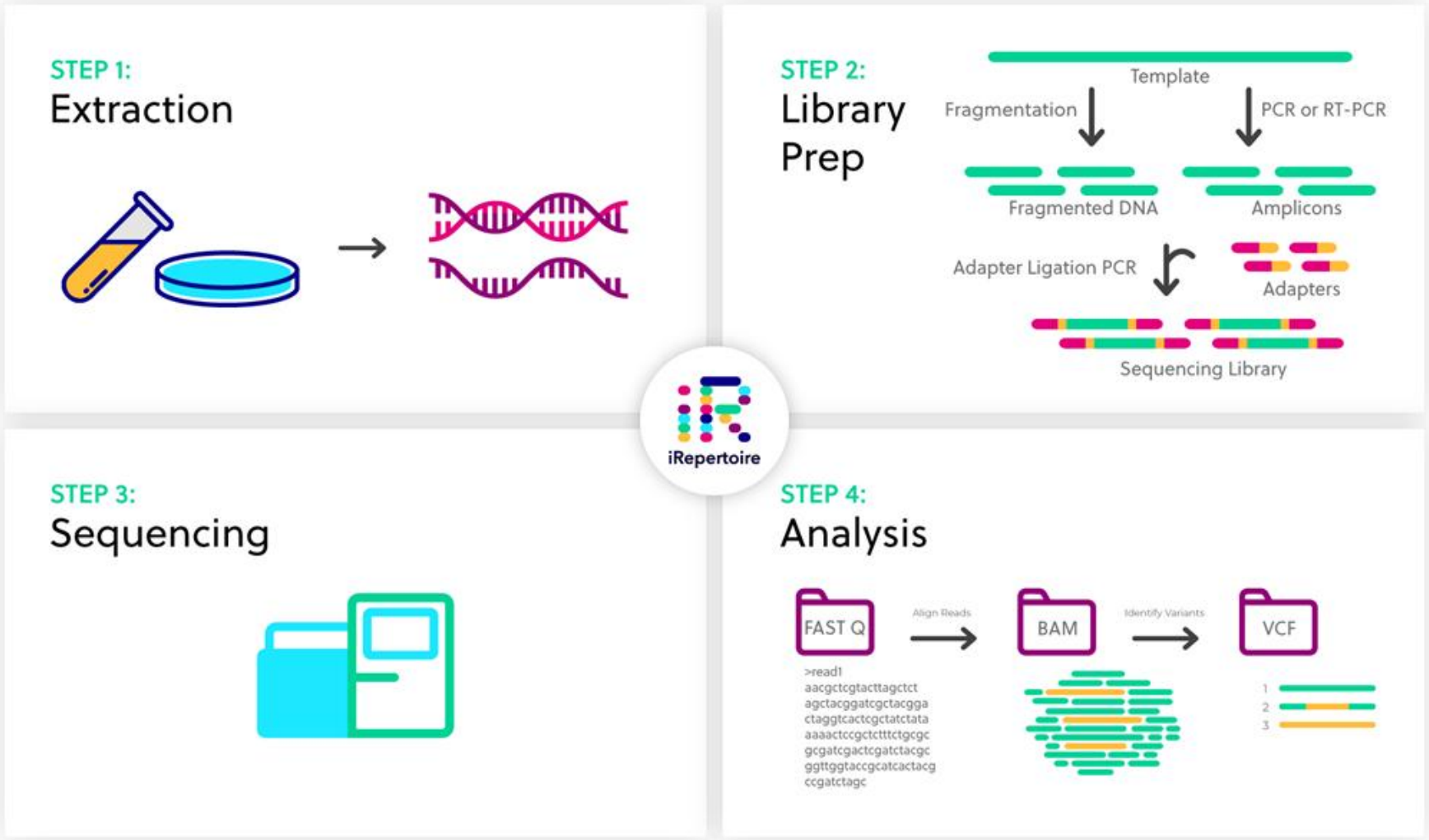


Секвенирование второго поколения

- Риды (прочтения) длиной в сотни нуклеотидов;
- Миллионы и миллиарды ридов считывается в течение одного анализа;
- Для анализа необходимы часы/дни;
- Стоимость секвенирования мегабазы сравнительно низка;
- Существуют различные технологии NGS, каждая из них характеризуется своей спецификой и характерными ошибками.

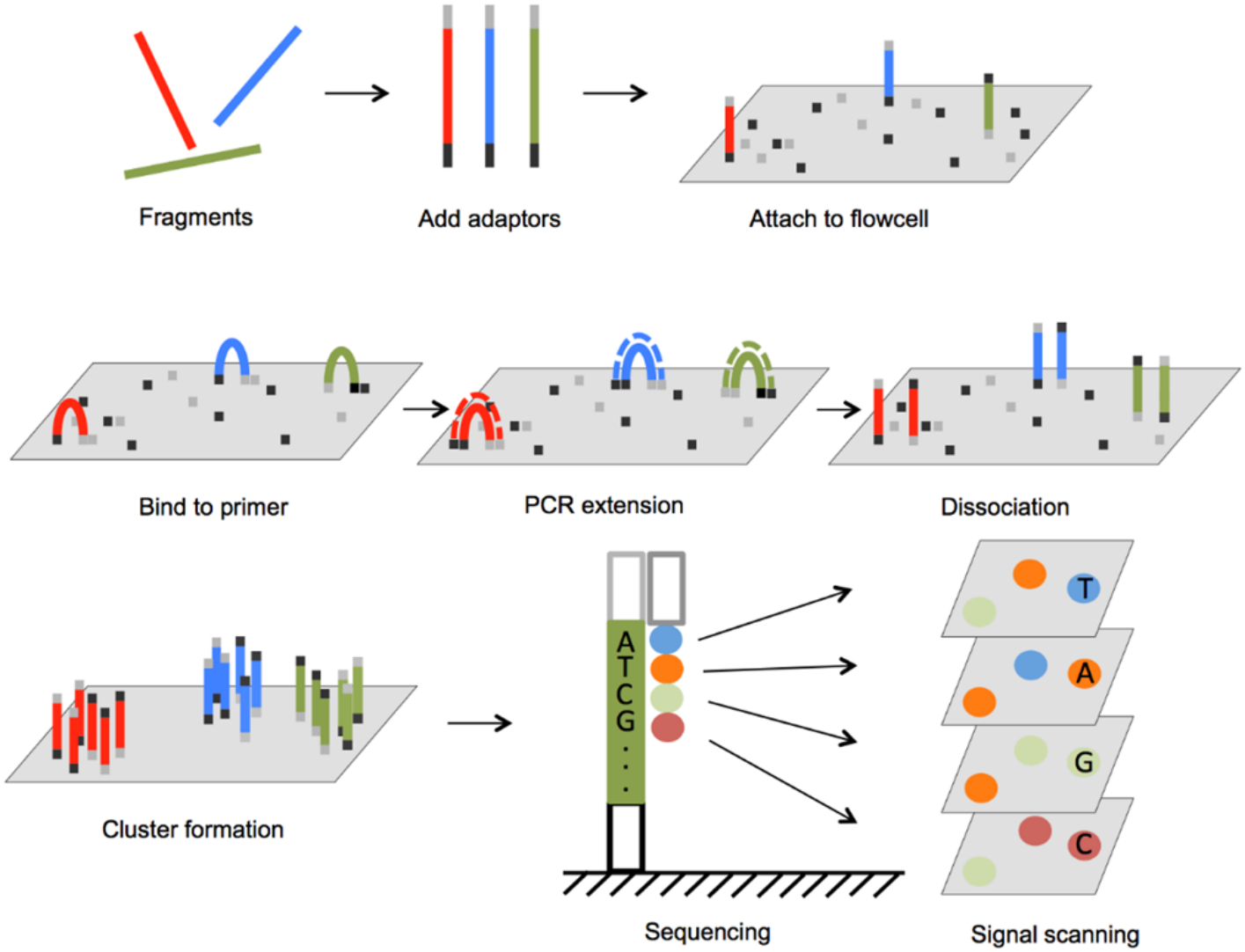


Секвенирование второго поколения

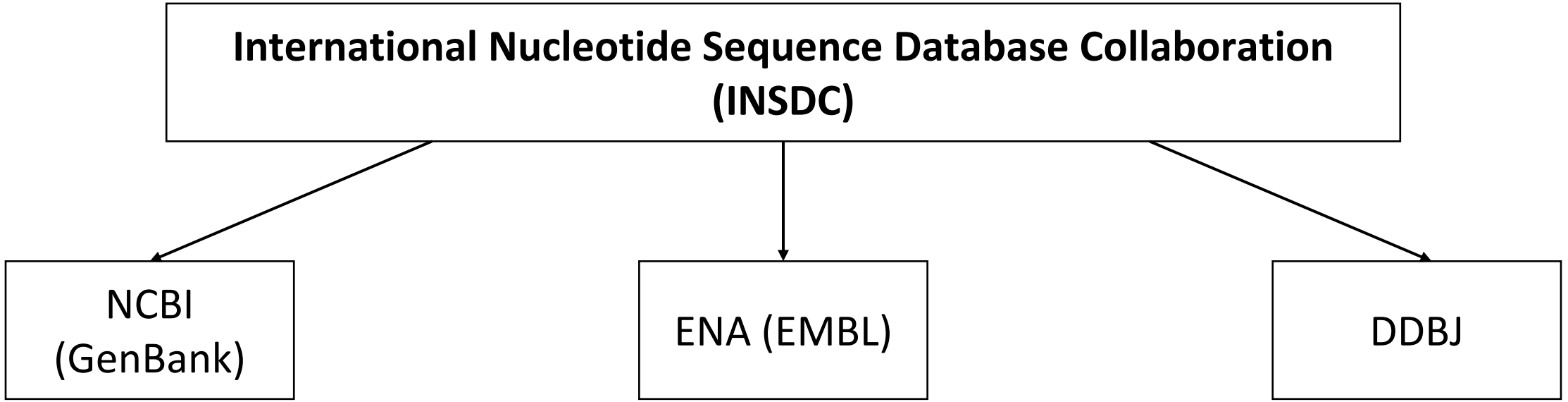


Технология Illumina: циклическая обратимая терминация

- Bridge amplification: обычные дНТФ, в конце все обратные цепи смываются;
- Sequencing by synthesis: меченый дНТФ блокирует дальнейший синтез; за один раунд присоединяется только один тип дНТФ; лазер => свечение; в начале следующего раунда блок синтеза устраняется (флюорофор смывается).



Основные хранилища (repositories)

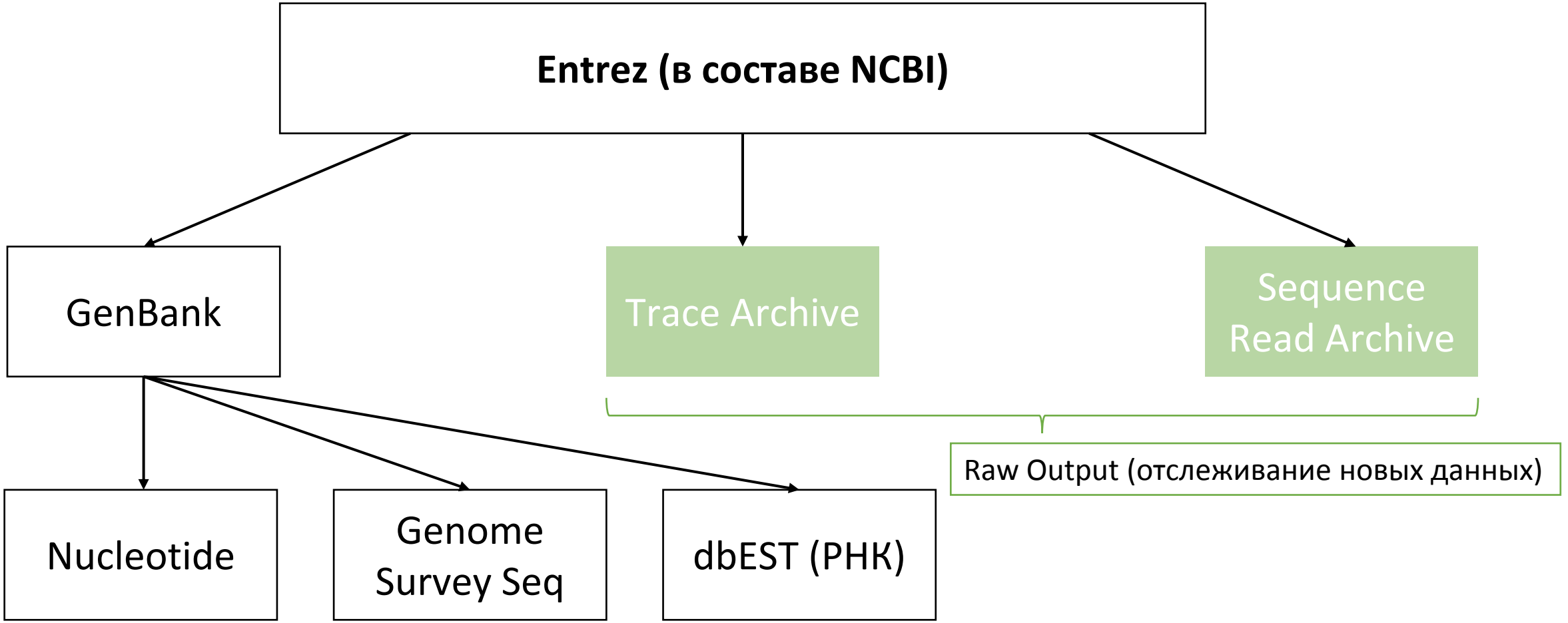


National Center for Biotechnology Information
NCBI

EMBL-EBI

DDBJ
DNA Data Bank of Japan

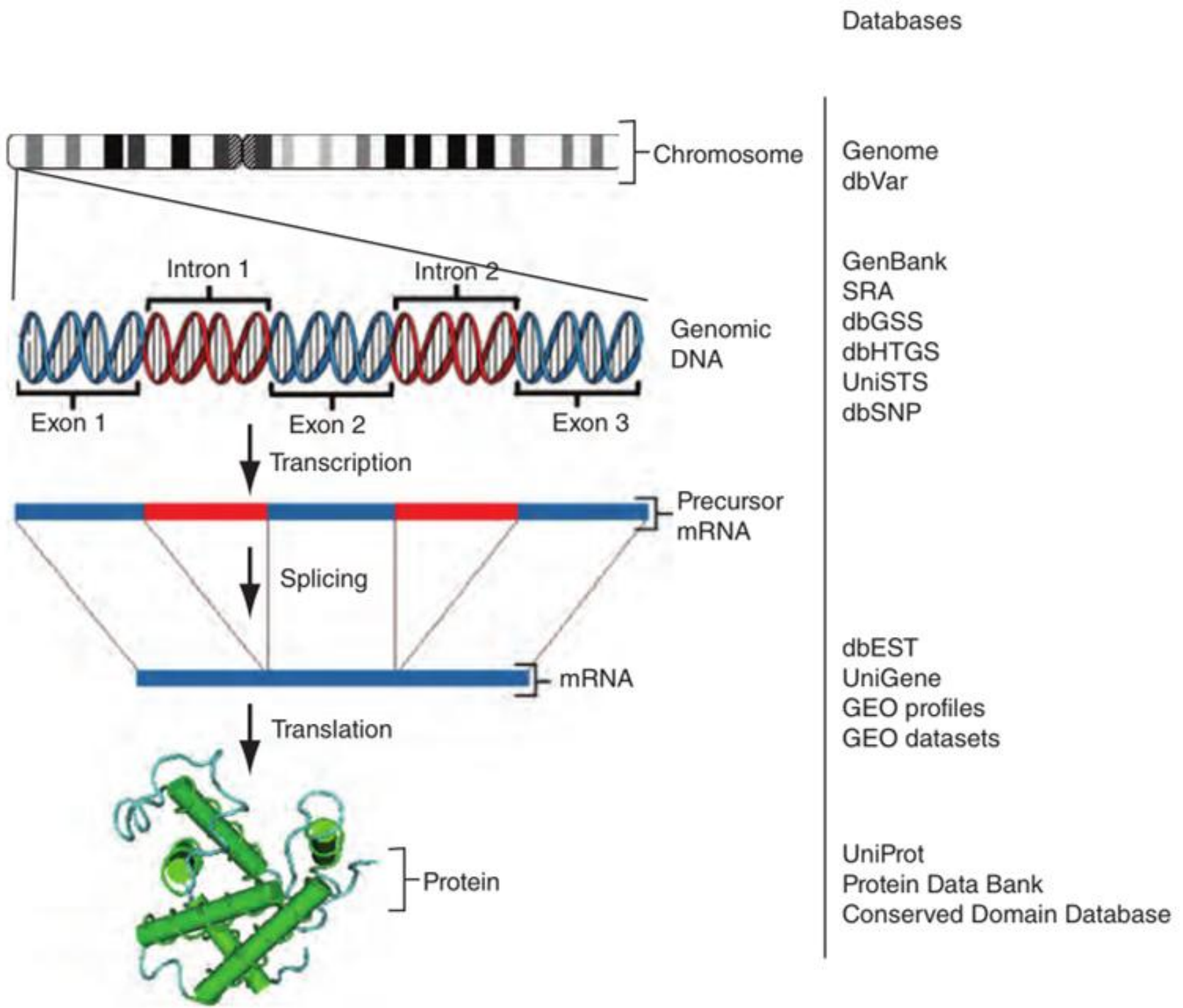
Структура хранилища (на примере Entrez)



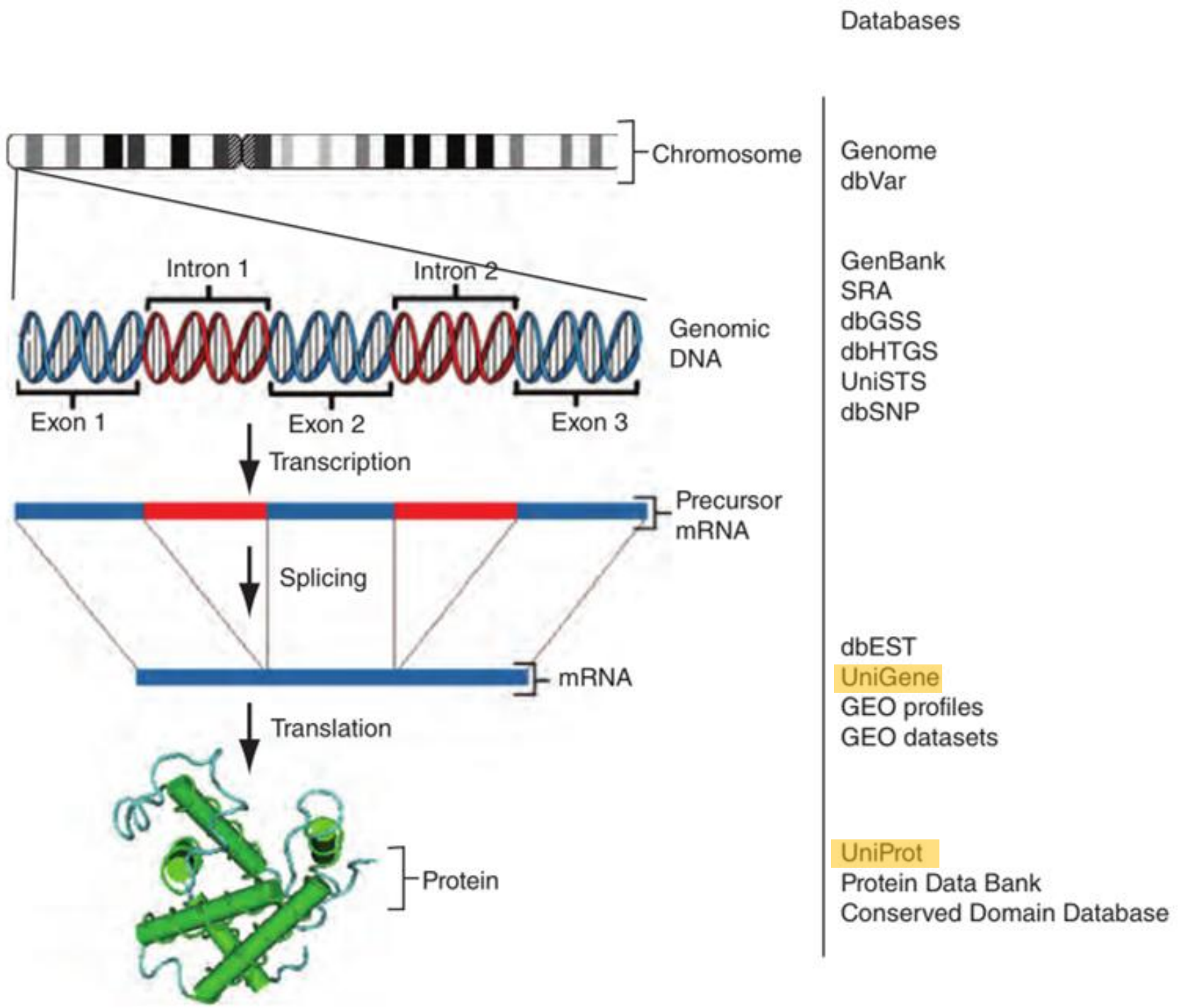
Типы данных

- Последовательности могут быть представлены на уровне **ДНК**, **РНК** или **белка**.

Основные базы данных



Основные базы данных



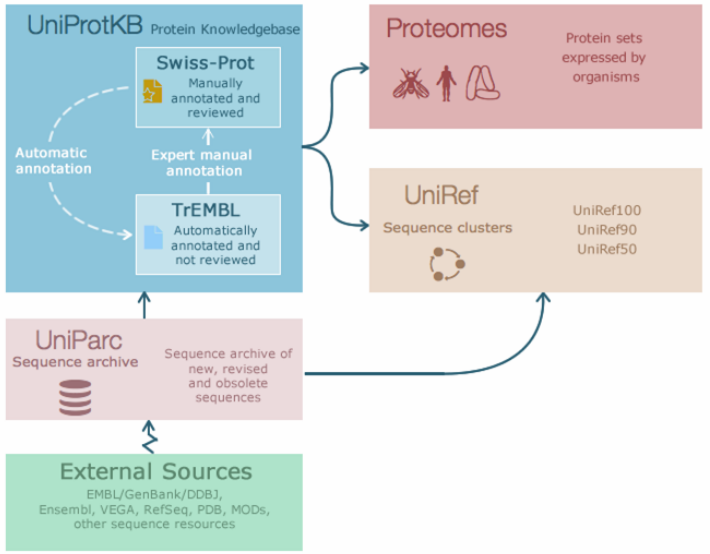
RefSeq (The Reference Sequence Project)

- Данный проект направлен на сопоставление каждому гену, РНК, белку одного уникального номера (**accession number**):
  - **NC_00123456** – геном;
  - **NM_00123456** – mRNA;
  - **NP_00123456** – белок;
  - **X*_00123456** – нет экспериментальных данных.
- Информация собирается со всех баз данных INSDC.

RefSeq (The Reference Sequence Project)

- В RefSeq имеется понятие уровня статуса:
  - PREDICTED – аннотация **автоматически** сгенерирована при помощи Eukaryotic Genome Annotation Pipeline;
  - PROVISIONAL – предварительный уровень аннотации;
  - REVIEWED/VALIDATED – последовательность аннотирована **вручную**.
- *PREDICTED: LOW QUALITY PROTEIN в заголовке означает, что последовательность была изменена относительно исходного геномного материала (т.е. скорее всего в геноме - ошибка).*

UniProtKB (Knowledgebase)



- Центр обработки информации – состоит из двух секций:
  - UniProtKB/**Swiss-Prot** – последовательности просмотрены и аннотированы вручную (**553 941 всего**);
  - UniProtKB/**TrEMBL** – не просмотрены, аннотированы автоматически (**80 204 459 всего**).

Примитивное сравнение основных баз данных белков

Swiss-Prot	RefSeq	GenBank
Большое количество точной, высококачественной информации по каждому конкретному белку.	Малоинформативный, но надёжный источник, учитывая отсутствие избыточной информации.	«Сырые» (raw) данные, возможна избыточная информация, требует самостоятельной обработки.
Мало данных.	Много данных.	Очень много данных.

Задание

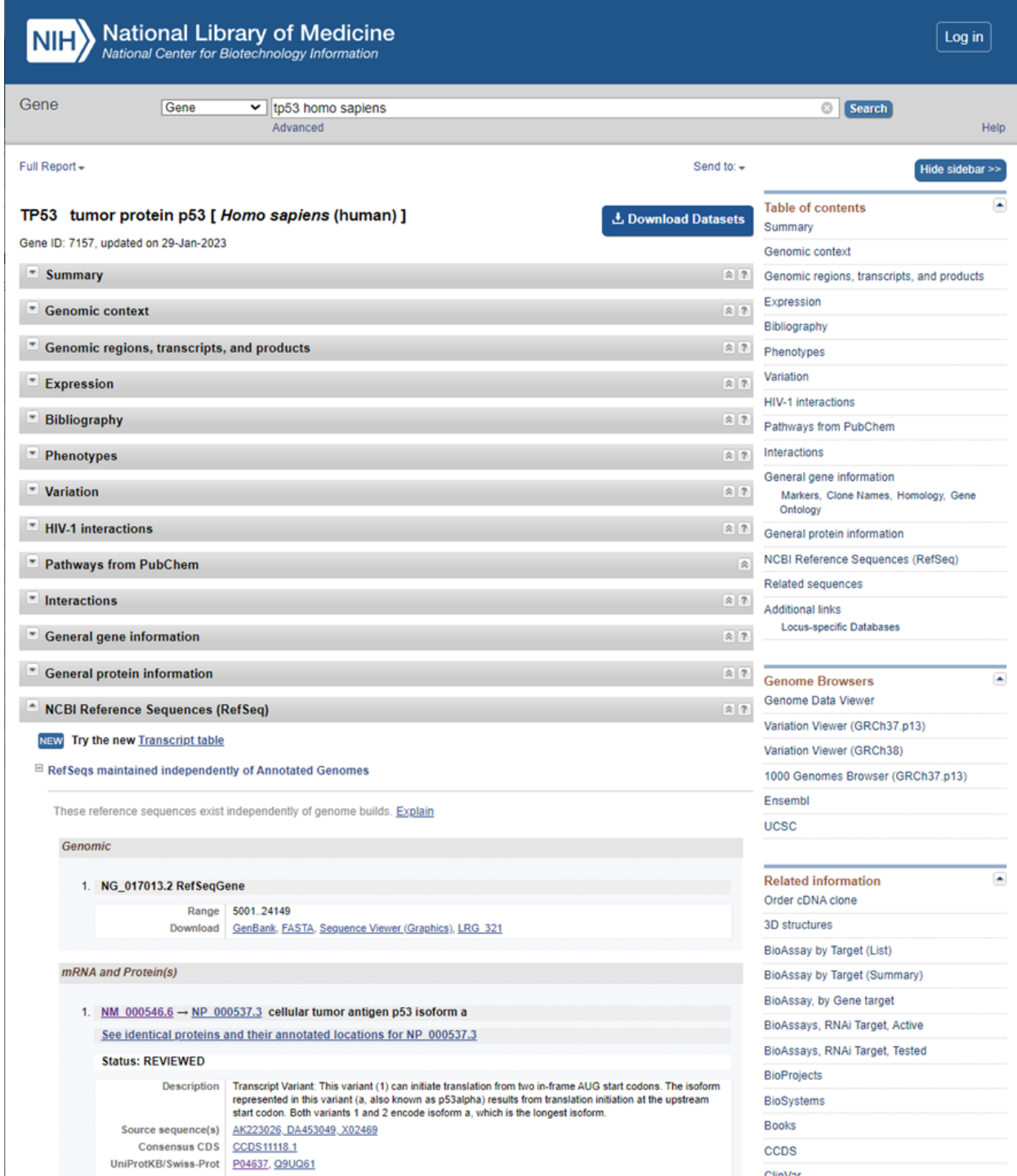
- Установить biopython через pip;
- Зайти на NCBI, найти “tp53 homo sapiens”;
- В разделе NCBI Reference Sequences (RefSeq) скопировать идентификаторы белков (NP_…);
- Внести их в список proteins в Python;
- Написать скрипт:

```
from Bio import Entrez

Entrez.email = "example@mail.com" #УКАЖИТЕ ВАШУ ПОЧТУ!!!

handle = Entrez.efetch(db = "protein", id = proteins, rettype = 'fasta')

result = handle.read()
print(result)
```



Способ записи нуклеотидных последовательностей

- Один из наиболее распространённых способов записи нуклеотидных и аминокислотных последовательностей – формат FASTA.
- Этот формат подразумевает, что у каждой последовательности есть своё название, которое отражается строкой выше, начинающейся со знака “>”.



Задание

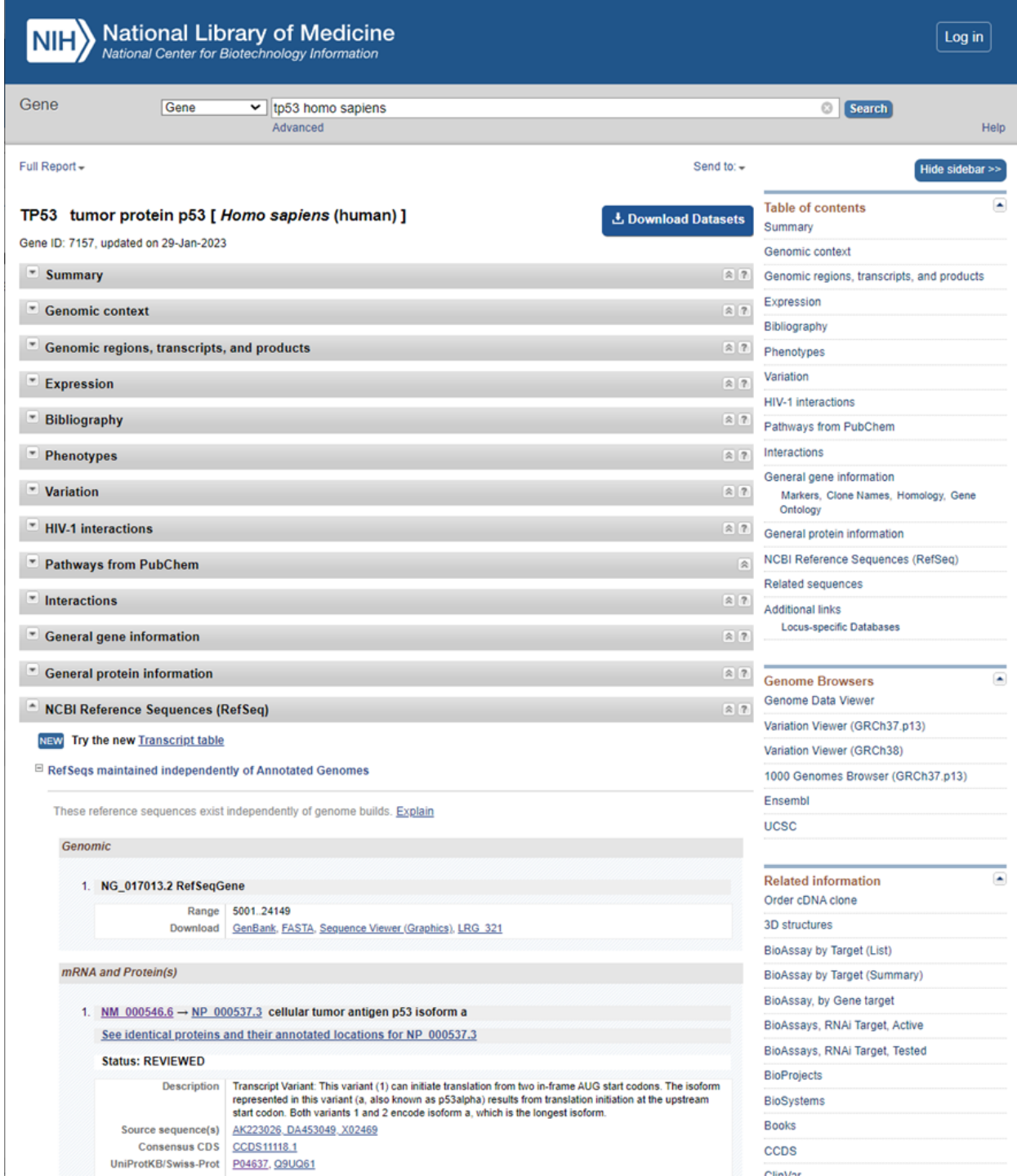
- Сделайте то же самое, но поменяйте rettype на ‘xml’;
- Что получилось?

```
from Bio import Entrez

Entrez.email = "example@mail.com" #УКАЖИТЕ ВАШУ ПОЧТУ!!!

handle = Entrez.efetch(db = "protein", id = proteins, rettype = 'xml')

result = handle.read()
print(result)
```



XML

В формат XML можно вместить гораздо больше информации, чем в, например, fasta: связанные с белком публикации, его домены, таксономическая классификация вида, которому белок принадлежит и прочее.

Однако, чтобы корректно прочитать XML, его необходимо парсить. В Biopython есть подходящий инструмент для этого – Entrez.read(), который переводит XML в словарь:

```
from Bio import Entrez

Entrez.email = "example@mail.com" #УКАЖИТЕ ВАШУ ПОЧТУ!!!

handle = Entrez.efetch(db = "protein", id = proteins, rettype = 'xml')

result = Entrez.read(handle)
print(result)
```

XML

XML переводится как расширяемый язык разметки (extensible markup language).

Кликните правой кнопкой мыши в браузере и нажмите “просмотреть код”. Это – язык HTML, который описывает содержание веб-страницы и тоже является языком разметки, в котором, однако, все теги предопределены.



XML – пример

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to>Студент</to>
    <from>Домашка</from>
    <heading>Напоминание</heading>
    <body>Не забудь обо мне в эти выходные!</body>
</note>
```

XML – теги

XML существует для хранения данных и состоит из тегов, которые не предопределены (т.е. соответствуют авторской задумке). Есть открывающие и закрывающие теги, ограничивающие элементы XML.

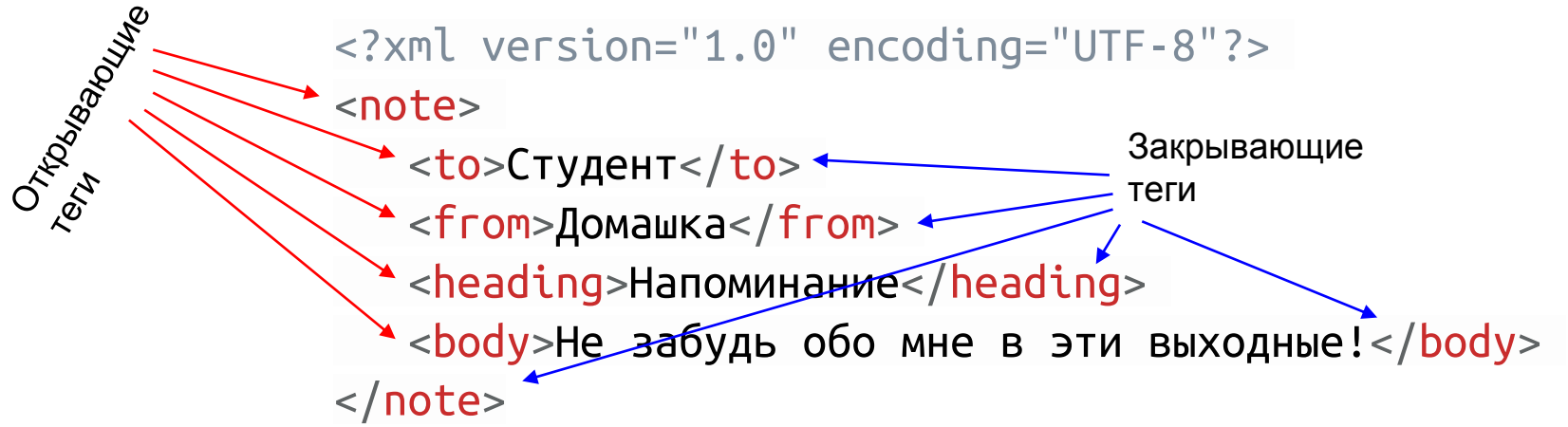
Открывающий тег

Закрывающий тег

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to>Студент</to>
    <from>Домашка</from>
    <heading>Напоминание</heading>
    <body>Не забудь обо мне в эти выходные!</body>
</note>
```

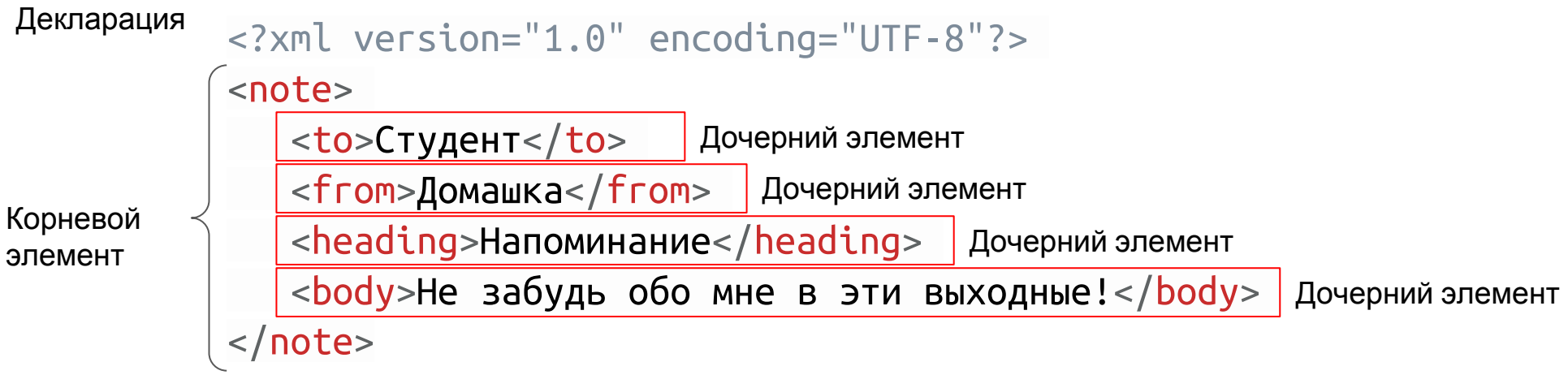
XML – теги открывающие и закрывающие

Все открывающие теги пишутся в “<>” без “/” на месте первого символа. Закрывающие – в “<>” с “/” на месте первого символа. Их названия должны соответствовать друг другу.



Дерево XML

Для описания взаимоотношений между элементами обычно используют термины "родитель", "потомок" и "брат"/"сестра". У "родительского" элемента есть "потомки" или "дочерние" элементы.



XML – атрибуты

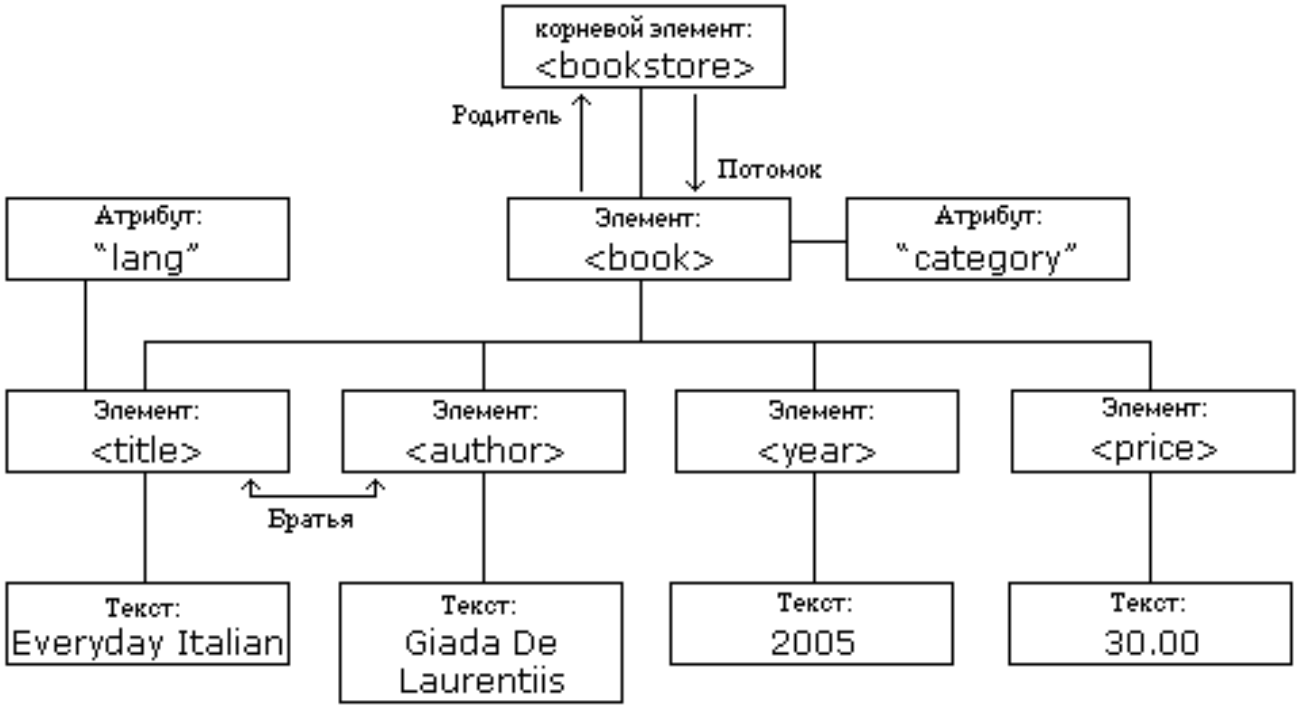
Для передачи информации кроме текстового контента могут быть атрибуты.

В данном примере описаны: родительский элемент с атрибутами и дочерними элементами (note), элемент с атрибутами и текстовым вложенным контентом (heading), также элемент с атрибутом без текстового контента (body) и пустой элемент (void).

```
<?xml version="1.0" encoding="UTF-8"?>
<note to=”Студент” from=”Домашка”>
   <heading void=”None”>Напоминание</heading>
   <body text=”Не забудь обо мне в эти выходные!”></body>
   <void></void>
</note>
```

XML – пример дерева

```xml
<bookstore>
   <book category="COOKING">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
   </book>
   <book category="CHILDREN">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
   </book>
   <book category="WEB">
      <title lang="en">Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price>
   </book>
</bookstore>
```



XML – правила

Регистр!!!

```
<Message>Это неправильно</message>
<message>Это правильно</message>
```

Корректная вложенность

```
<tag1><tag2>Это неправильно</tag1></tag2>
<tag1><tag2>Это правильно</tag2></tag1>
```

Обязательное наличие
корневого элемента
и пролога

```
<?xml version="1.0" encoding="UTF-8"?>
<корневой>
   <потомок>
      <подпотомок>.....</подпотомок>
   </потомок>
</корневой>
```

Значения атрибутов
в кавычках

```
<tag category=Это неправильно></tag>
<tag category=”Это неправильно”></tag>
```

```
<!-- Это комментарий -->
```

XML – запрещённые символы

Вы уже могли заметить, что символы “<” и “>” используются XML для обозначения тегов. Что делать, если данные символы используются в текстовом контенте?

```
<message>если жалование < 1000</message>
```

Сущность	Символ	Значение
<	<	меньше, чем
>	>	больше, чем
&	&	амперсанд
'	'	апостроф
"	"	кавычки

```
<message>если жалование &lt; 1000</message>
```

XML – пространства имён

По умолчанию элементы с одинаковыми именами тегов должны содержать аналогичный контент. Если это не так, следует использовать префиксы или определять их пространства имён атрибутом xmlns:

```
<table>
   <tr>
      <td>Яблоки</td>
      <td>Бананы</td>
   </tr>
</table>

<table>
   <name>Кофейный столик</name>
   <width>80</width>
   <length>120</length>
</table>
```

```
<h:table>
   <h:tr>
      <h:td>Яблоки</h:td>
      <h:td>Бананы</h:td>
   </h:tr>
</h:table>

<f:table>
   <f:name>Кофейный столик</f:name>
   <f:width>80</f:width>
   <f:length>120</f:length>
</f:table>
```

```
<table xmlns="http://www.w3.org/TR/html4/">
   <tr>
      <td>Яблоки</td>
      <td>Бананы</td>
   </tr>
</table>

<table xmlns="http://www.w3schools.com/furniture">
   <name>Кофейный столик</name>
   <width>80</width>
   <length>120</length>
</table>
```

XML

Вы уже смогли убедиться, что в Biopython есть свой парсер XML, который способен переводить этот формат в словарь Python.

Однако, для парсинга XML произвольного вида (а не только XML из системы Entrez) в Python существуют свои парсеры.

```
from Bio import Entrez

Entrez.email = "example@mail.com" #УКАЖИТЕ ВАШУ ПОЧТУ!!!

handle = Entrez.efetch(db = "protein", id = proteins, rettype = 'xml')

result = Entrez.read(handle)
print(result)
```

xml.etree.ElementTree

В Python есть библиотека xml.

Для создания или парсинга xml нам нужен субмодуль ElementTree:

```
import xml.etree.ElementTree as ET
```

Создание xml

Посмотрите, что делает каждая строка следующего блока:

```
import xml.etree.ElementTree as ET
p = ET.Element('parent')
c = ET.SubElement(p, 'child1')
ET.dump(p)
```

Сохраните полученное дерево:

```
tree = ET.ElementTree(p)
tree.write("sample.xml")
```

Парсинг xml

Посмотрите, что делает каждая строка следующего блока:

```
import xml.etree.ElementTree as ET

tree = ET.parse('sample.xml')

root = tree.getroot()
for child in root:
        print(child.tag, child.attrib,

child.text)

element = root[0]
```

Установите атрибут и текстовый контент элемента:

```
element.set('attribute_name', 'attribute_value')

element.set('attribute_name2', 'attribute_value')

element.text="string_text"
```

Как вывести атрибут и текстовый контент на экран?

JSON

Ещё один частый формат для хранения данных – JSON.

Он естественно переводится в словарь Python и обратно.

```python
import json
#sample python dictionary
python_dict = {
  "key1": "Value1",
  "key2": "Value2",
  "key3": "Value3",
}
JSON_obj=json.dumps(python_dict)
print(JSON_obj)
```

```
{"key1": "Value1", "key2": "Value2", "key3": "Value3"}
```

```python
import json

json_str = """
   {
      "name":"Bryan",
      "age":29,
      "city":"New York"
   }
"""

json_obj = json.loads(json_str)

print(json_obj["name"])
print(json_obj["age"])
print(json_obj["city"])
```

```
Bryan
29
New York
```