

RNA-Seqによる 大量発現データ解析の基礎・応用

国立研究開発法人 農業・食品産業技術総合研究機構

次世代作物開発研究センター

（併任：高度解析センター）

川原 善浩



y.kawahara@affrc.go.jp



@YoshiKawahara

9時00分-12時00分

「RNA-Seqによる大量発現データ解析の基礎」



～お昼休み～



13時00分-15時00分

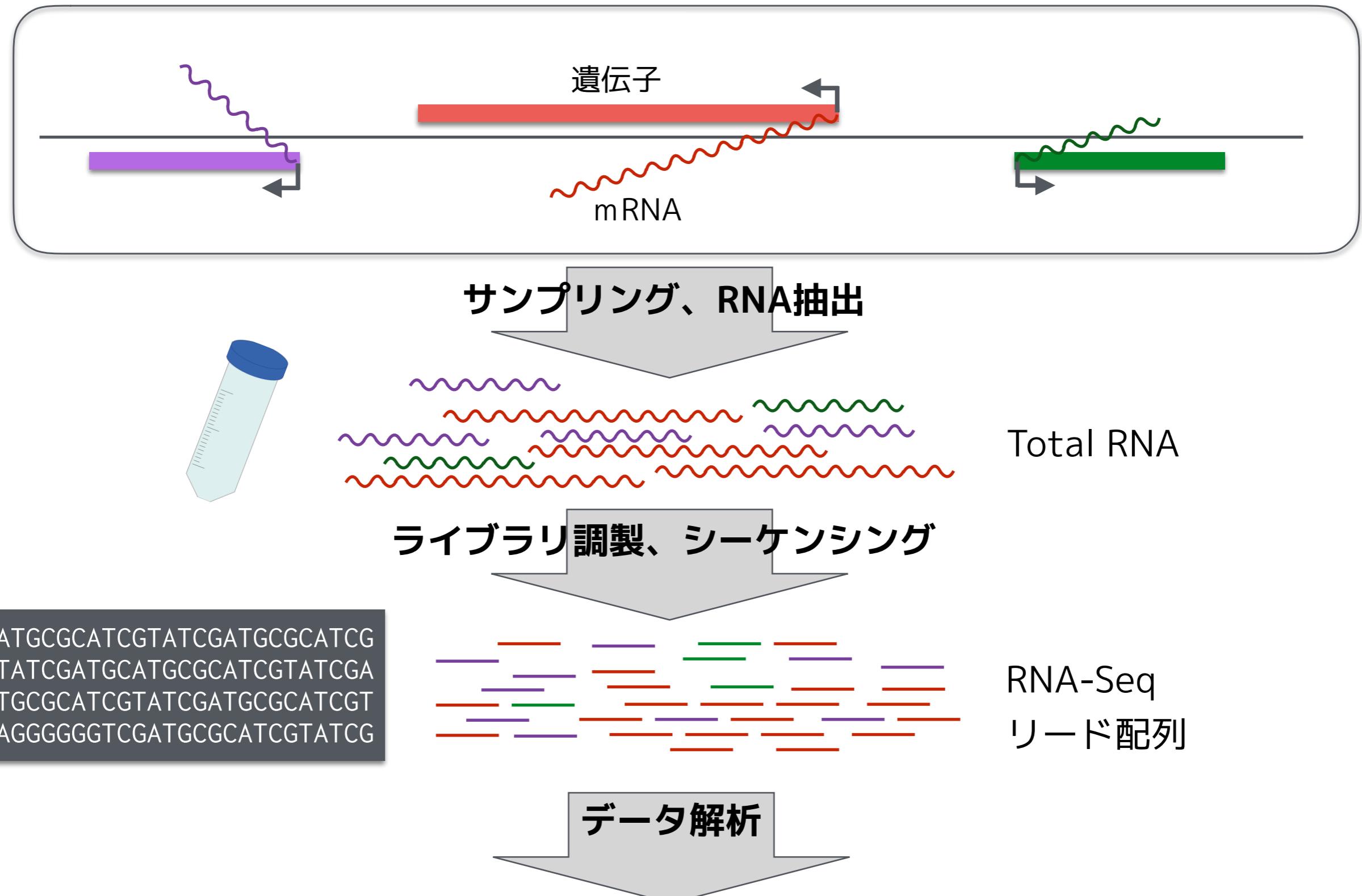
「RNA-Seqによる大量発現データ解析の応用」

適当に質問タイム、休憩を挟みながら・・・

- リファレンス情報を用いたRNA-Seq解析
- De novoトランスクリプトーム解析
- RNA-Seqデータを用いた多型検出

の演習をおこないます。

RNA-Seqとは

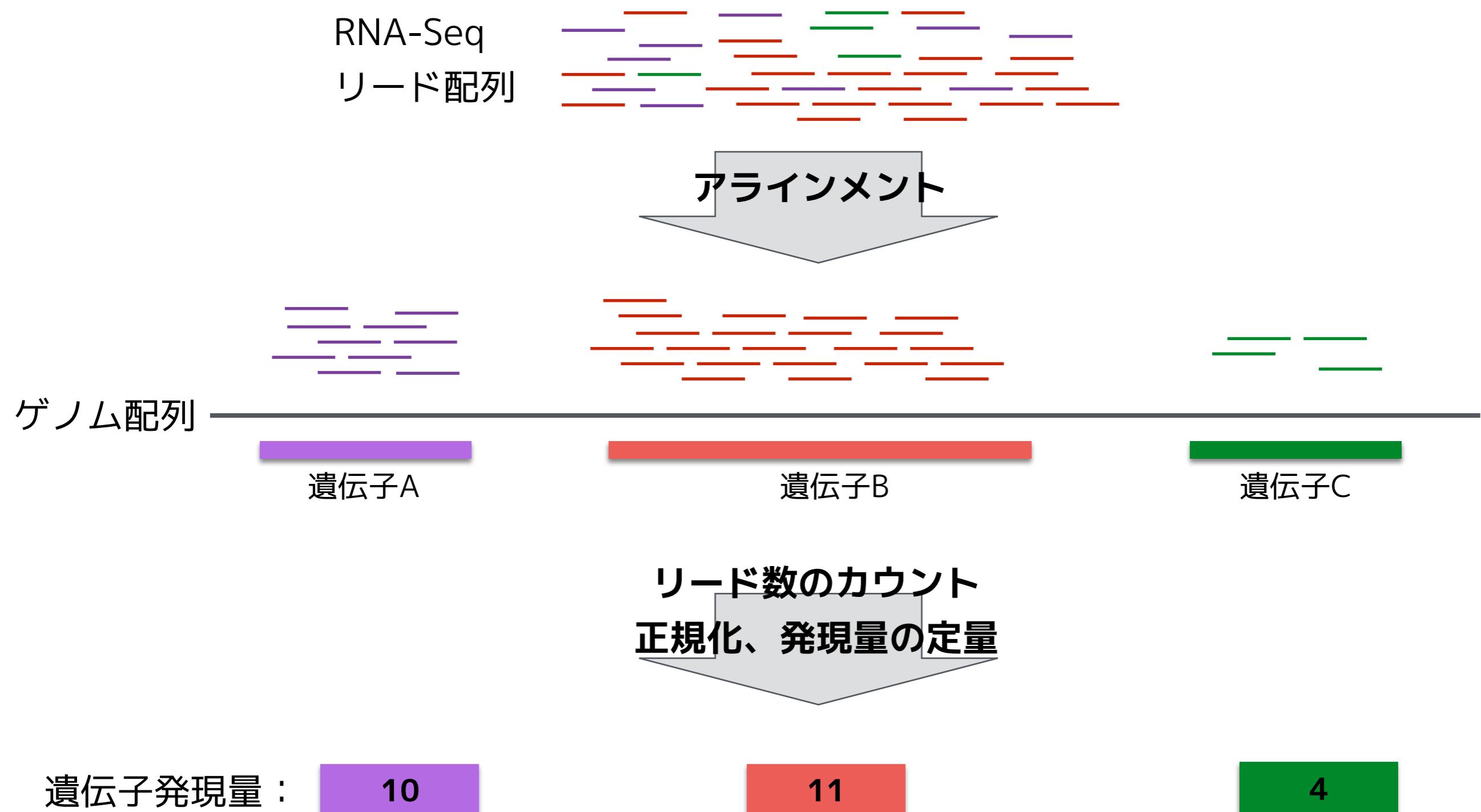


発現している遺伝子カタログの作成、
遺伝子発現解析など、様々な目的に利用可能

手法1：ゲノム配列や遺伝子アノテーションを用いた遺伝子発現解析



RNA-Seqリードをゲノム配列にアラインメント（マッピング）し、
遺伝子ごとにマップされたリードをカウントし、遺伝子発現を定量する。

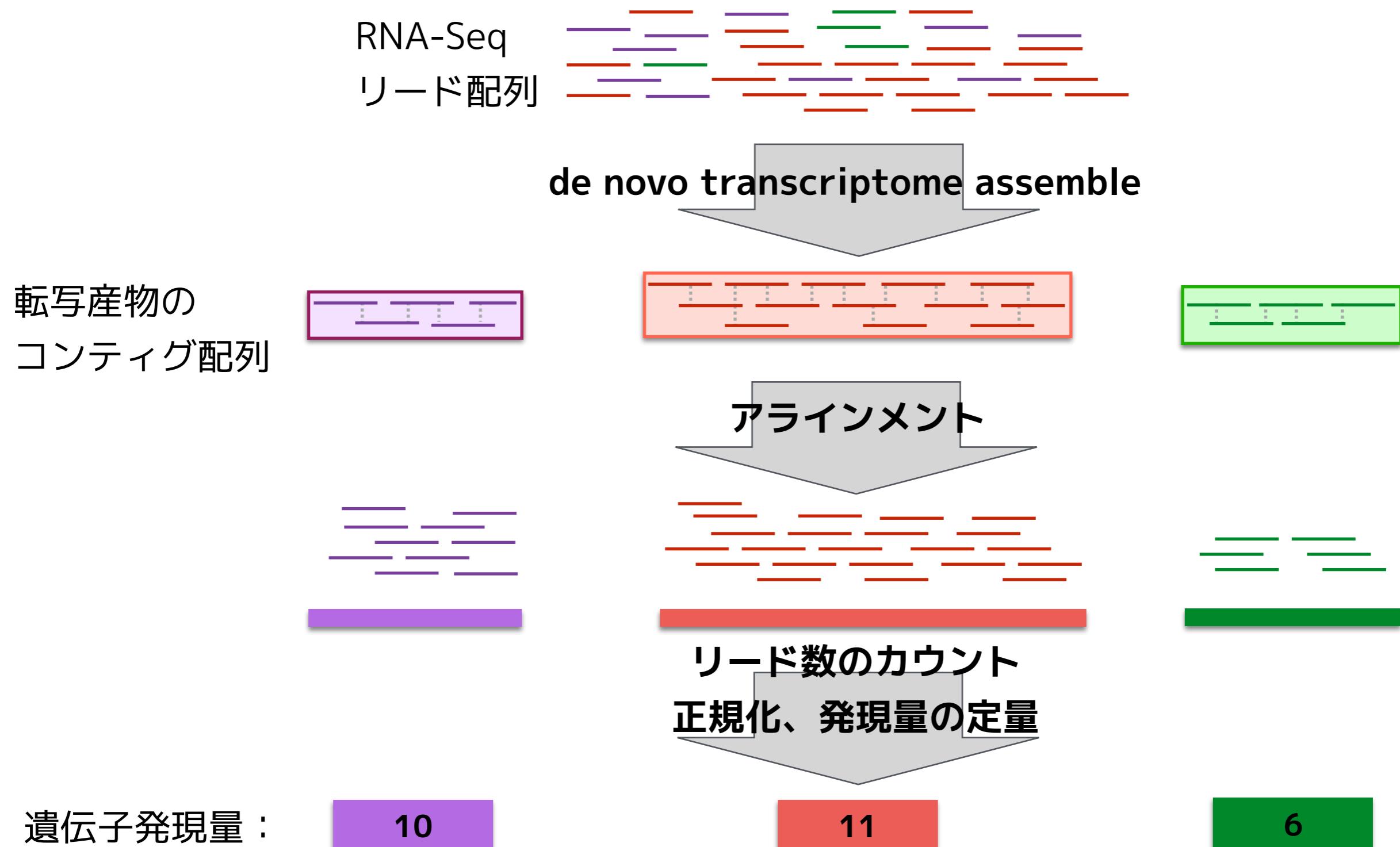


手法2：de novo transcriptome assemble法による遺伝子発現解析



まず始めに、RNA-Seqリード配列から転写産物配列を再構築する。

アセンブルされた転写産物配列上にRNA-Seqリードをアラインメントし、転写産物ごとにリード数をカウントして遺伝子発現を定量する。



遺伝子発現量 :

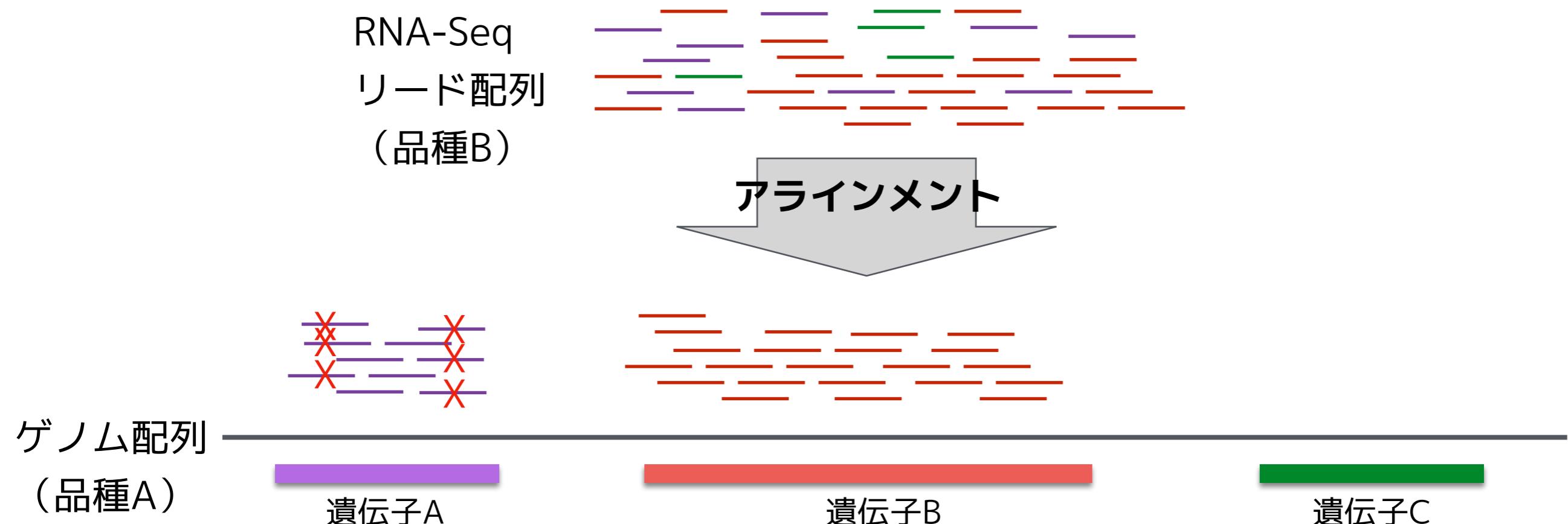
10

11

6

RNA-Seqデータからは発現量以外にも得られる情報はある

リファレンスとは異なる種や品種由来のRNA-Seqリードをリファレンスゲノム配列にアラインメントし、ゲノムとリード配列の違い（種間や品種間の転写領域上の多型情報）を得る。

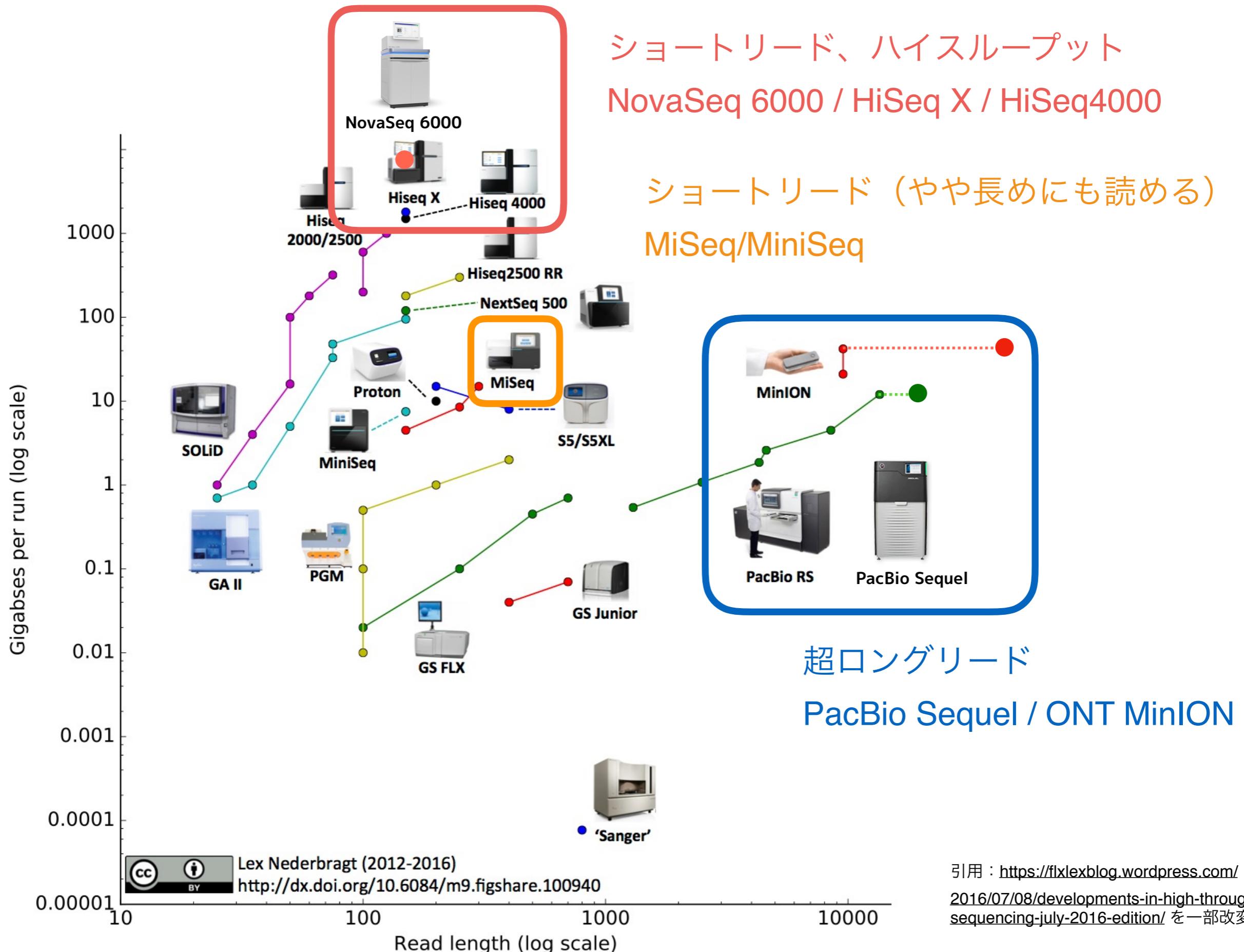


X:品種AとBでゲノム配列が不一致する箇所

ライブラリ調製、シーケンシングの前に
どのようにデータを解析するかを考えよう

「データが出てから考える」
では手遅れ！（ということもある）

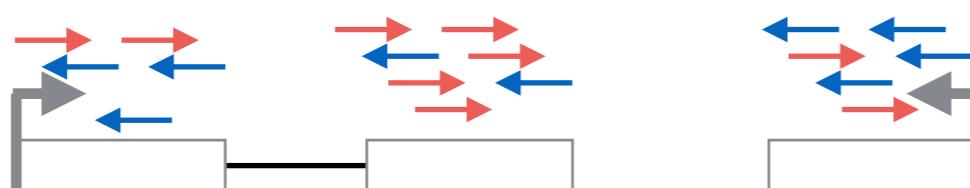
シークンサーの種類はいろいろある



ライブラリ調製やシークエンシング方法にもいろいろある

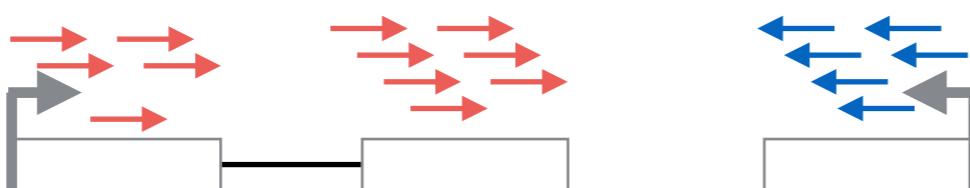


普通の (standard) RNA-Seq



転写の向きとリードの向きは無関係

strand-specific (stranded) RNA-Seq



転写の向きとリードの向きに対応関係があり、アセンブルや発現解析の精度が向上する。

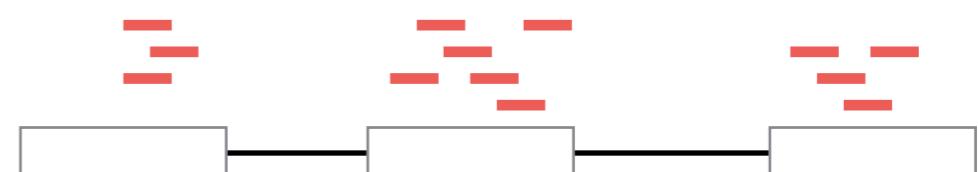
断片化したcDNAの片側だけを読むか両端を読むかの違い。ある一定の距離をもつペアリード情報を用いることで、遺伝子構造予測やアセンブル精度が向上する。

シークエンシング

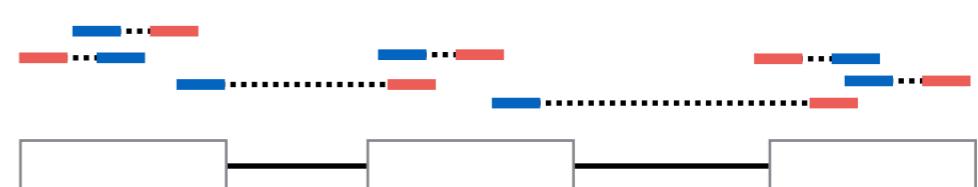
single read
(SR/SE)



アラインメント



paired-end read
(PE)



目的にあったシークエンシング手法やライブラリ調製法を選ぶ



リファレンス情報を用いた
遺伝子発現解析

- リファレンス情報なし
の遺伝子発現解析
- 多型検出

遺伝子カタログ作成

ゲノムや遺伝子情報は充分。
とにかく発現量が知りたい。

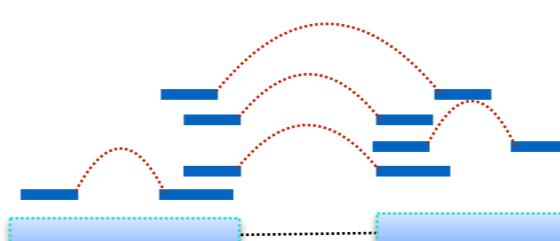
とにかく安価に多くの
リードを読む。

HiSeq, single-read, 50bp



リード数と転写構造の情
報をバランスよく得る。

HiSeq, paired-end, 100/150bp



発現量よりもまずはどん
な遺伝子が発現している
か知りたい。

リード数は少ないが、
とにかく長いリードで
転写産物の構造を得る。

PacBio, ONT



- ▶ リファレンス情報を用いたRNA-Seq解析
- ▶ De novoトランスクリプトーム解析
- ▶ RNA-Seqデータを用いた多型検出

解析ツールの取得、インストール



リファレンス情報を用いたRNA-Seq解析

1. FastQC
2. Trimmomatic
3. HISAT2
4. StringTie
5. Samtools
6. Kallisto

De novoトランскriプトーム解析

1. FastQC
2. Trimmomatic
3. Trinity
4. Bowtie2
5. Salmon
6. Jellyfish
7. BLAST+
8. edgeR (R/Bioconductor)

RNA-Seqデータを用いた多型検出

1. FastQC
2. Trimmomatic
3. Samtools
4. STAR2
5. Picard
6. GATK

データの可視化

1. IGV (デスクトップアプリ)
2. R/RStudio (デスクトップアプリ)
3. Ballgown (R/Bioconductor)
4. genefilter (R/Bioconductor)
5. tidyverse (R)

解析ツールの取得、インストール



ウェブサイトから最新版のツールのバイナリ、またはソースファイルをダウンロード、インストールする（資料末尾に各ツールのウェブサイト等の情報あり）。

biocondaやhomebrew（Mac）などのパッケージマネージャ、コンテナ型仮想環境「Docker」を用いてインストールできるものもある。

Samtoolsの場合 (<http://www.htslib.org>)

The screenshot shows the Samtools website's "Downloads" section. A red arrow points from the Japanese text "ダウンロード" (Download) in the heading to the "samtools-1.9" download button. Another red arrow points from the Japanese text "インストール方法の説明" (Installation method explanation) to the "Building and installing" section.

Current releases

SAMtools and BCFtools are distributed as individual packages. The code uses HTSlib internally, but these source packages contain their own copies of htslib so they can be built independently.

HTSlib is also distributed as a separate package which can be installed if you are writing your own programs against the HTSlib API. HTSlib also provides the `bgzip`, `htsfile`, and `tabix` utilities, so you may also want to build and install HTSlib to get these utilities, or see the additional instructions in [INSTALL](#) to install them from a samtools or bcftools source package.

Download current source releases: [samtools-1.9](#) [bcftools-1.9](#) [htslib-1.9](#)

See also release notes for [samtools](#), [bcftools](#), and [htslib](#).

New releases are announced on the [samtools mailing lists](#) and by [@htslib](#) on Twitter. Previous releases available from the [samtools GitHub organisation](#) (see [samtools](#), [bcftools](#), or [htslib](#) releases) or from the [Sourceforge project](#).

Building and installing

Building each desired package from source is very simple:

```
cd samtools-1.x      # and similarly for bcftools and htslib
./configure --prefix=/where/to/install
make
make install
```

See [INSTALL](#) in each of the source directories for further details.

The executable programs will be installed to a `bin` subdirectory under your specified prefix, so you may wish to add this directory to your `$PATH`:

```
export PATH=/where/to/install/bin:$PATH      # for sh or bash users
```

```
setenv PATH /where/to/install/bin:$PATH      # for csh users
```

演習用データの準備



以下の要領で演習用データ（/work/NGSworkshop/RNA-Seq.tar.gz）を各自のホームディレクトリにコピー、解凍、展開して中身を確認する。

```
$ cd ..... ホームディレクトリへ移動
$ pwd ..... カレントディレクトリを表示
/home/guest01
$ cp /work/NGSworkshop/RNA-Seq.tar.gz ./ ..... 解析用データをカレントディレクトリにコピー
$ ls -lh
合計 169M
-rw-rw-r-- 1 guest01 guest01 169M 9月 17 11:11 RNA-Seq.tar.gz
$ tar xfz RNA-Seq.tar.gz ..... 解析用データを解凍、展開
$ ls -lh
合計 169M
drwxrwxr-x 6 guest01 guest01 109 9月 17 11:02 RNA-Seq
-rw-rw-r-- 1 guest01 guest01 169M 9月 17 11:11 RNA-Seq.tar.gz
$ cd RNA-Seq
$ ls ..... 解析用ディレクトリの中身を確認
data denovo install_tools.sh noaln useref varcall
```

RNA-Seqディレクトリ中の各ファイル/ディレクトリの説明

- data : 解析に用いるデータ（ゲノム配列、遺伝子アノテーション、RNA-Seqリードなど）
- denovo : De novoトランск립トーム解析用のディレクトリ
- noaln : アラインメントフリーのRNA-Seq解析用のディレクトリ
- useref : リファレンス情報を用いたRNA-Seq解析用のディレクトリ
- varcall : RNA-Seqデータを用いた多型検出解析用のディレクトリ
- install_tools.sh : 解析ツール群をインストールするためのシェルスクリプト

解析ツールのダウンロードとインストール



全解析ツールをまとめてインストールするためのシェルスクリプト

```
$ less install_tools.sh
```

- install_tools.shの先頭部分 -

```
mkdir tool
cd tool
ToolDir=`pwd`  
..... ツール類を置くためのディレクトリの作成と移動  
..... ツールディレクトリのパスの取得  
  
echo "START:" `date` ..... ツールインストールの開始時間を出力  
### Tools for "useref" analysis ..... #で始まるのはコメント行  
echo "installing FastQC..." ..... FastQCのダウンロード  
wget http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.8.zip  
unzip fastqc_v0.11.8.zip ..... 解凍・展開  
chmod +x FastQC/fastqc ..... fastqcプログラムに実行権限を与える  
rm fastqc_v0.11.8.zip ..... 不要なダウンロードファイルは削除  
  
echo "installing Trimmomatic..." ..... Trimmomaticのダウンロード  
wget http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.39.zip  
unzip Trimmomatic-0.39.zip ..... 解凍・展開  
rm Trimmomatic-0.39.zip ..... 不要なダウンロードファイルは削除
```

実行ファイル（バイナリファイル）やJavaプログラムのインストールは、
基本的にダウンロード（wget や curl）して、解凍・展開（unzip や tar）するだけ。

解析ツールのダウンロードとインストール



- install_tools.shのつづき1 -

```
echo "installing HISAT2..."                                HISAT2のダウンロード
wget ftp://ftp.ccb.jhu.edu/pub/infphilo/hisat2/downloads/hisat2-2.1.0-Linux_x86_64.zip
unzip hisat2-2.1.0-Linux_x86_64.zip ..... 解凍・展開
rm hisat2-2.1.0-Linux_x86_64.zip ..... 不要なダウンロードファイルは削除

echo "installing StringTie..."                            StringTieのダウンロード
wget http://ccb.jhu.edu/software/stringtie/dl/stringtie-2.0.3.Linux_x86_64.tar.gz
tar xfz stringtie-2.0.3.Linux_x86_64.tar.gz ..... 解凍・展開
rm stringtie-2.0.3.Linux_x86_64.tar.gz ..... 不要なダウンロードファイルは削除

echo "installing Samtools..."                           Samtoolsのダウンロード
wget https://github.com/samtools/samtools/releases/download/1.9/samtools-1.9.tar.bz2
tar xfj samtools-1.9.tar.bz2 ..... 解凍・展開
rm samtools-1.9.tar.bz2 ..... 不要なダウンロードファイルは削除
cd samtools-1.9 ..... Samtoolsディレクトリに移動
make ..... コンパイル
make prefix=`pwd` install ..... prefix (インストール先) を指定してインストール
cd .. ..... toolディレクトリに戻る
```

- *.tar.gz や *.tar.bz2 の解凍・展開は、 tar コマンドにオプションをつけて実行する
(*.tar.gz の場合は「tar xfz」、 *.tar.bz2 の場合は「tar xfj」)。
- Samtoolsについてはソースファイルをダウンロードし、コンパイル (make) 、インストール (make install) を行なう。

解析ツールのダウンロードとインストール

- install_tools.shのつづき2 -

```
echo "installing Kallisto..."                                Kallistoのダウンロード
wget https://github.com/pachterlab/kallisto/releases/download/v0.46.0/kallisto_linux-
v0.46.0.tar.gz
tar xfz kallisto_linux-v0.46.0.tar.gz ..... 解凍・展開
rm kallisto_linux-v0.46.0.tar.gz ..... 不要なダウンロードファイルは削除
```

解析ツールのダウンロードとインストール

- install_tools.shのつづき3 -

```
### Tools for "denovo" analysis
echo "installing CMake..."
wget https://github.com/Kitware/CMake/releases/download/v3.15.3/cmake-3.15.3-Linux-x86_64.tar.gz
tar xfz cmake-3.15.3-Linux-x86_64.tar.gz ..... 解凍・展開
rm cmake-3.15.3-Linux-x86_64.tar.gz ..... 不要なダウンロードファイルは削除
PATH=$ToolDir/cmake-3.15.3-Linux-x86_64/bin:$PATH ..... CMakeの実行ファイルのあるディレクトリをパスに追加、設定
export PATH

echo "installing Trinity..."
wget https://github.com/trinityrnaseq/trinityrnaseq/archive/Trinity-v2.8.5.tar.gz
tar xfz Trinity-v2.8.5.tar.gz ..... 解凍・展開
rm Trinity-v2.8.5.tar.gz ..... 不要なダウンロードファイルは削除
cd trinityrnaseq-Trinity-v2.8.5 ..... Trinityのディレクトリに移動
make ..... コンパイル
cd ..
```

- 最新版のTrinityのコンパイルにはCMakeが必要なので、先にCMakeをインストールし、パスの設定をする。CMakeはソフトウェアのビルドの自動化ツールである。

解析ツールのダウンロードとインストール

- install_tools.shのつづき4 -

```
echo "installing Jellyfish..."  
mkdir Jellyfish-2.3.0  
cd Jellyfish-2.3.0  
wget https://github.com/gmarcais/Jellyfish/releases/download/v2.3.0/jellyfish-linux  
mv jellyfish-linux jellyfish  
chmod 744 jellyfish ..... jellyfishプログラムに実行権限を与える  
cd .. (FastQCの時と権限の指定方法が違うが  
やっていることはほぼ同じ)  
  
echo "installing Salmon..."  
wget https://github.com/COMBINE-lab/salmon/releases/download/v0.14.1/  
salmon-0.14.1_linux_x86_64.tar.gz  
tar xfz salmon-0.14.1_linux_x86_64.tar.gz ..... バイナリファイルをダウンロード、展開、  
rm salmon-0.14.1_linux_x86_64.tar.gz 不要なファイルの削除  
  
echo "installing Bowtie2..."  
wget https://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.3.5.1/bowtie2-2.3.5.1-linux-  
x86_64.zip  
unzip bowtie2-2.3.5.1-linux-x86_64.zip ..... バイナリファイルをダウンロード、展開、  
rm bowtie2-2.3.5.1-linux-x86_64.zip 不要なファイルの削除  
  
echo "installing BLAST+"  
wget ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST//ncbi-blast-2.9.0+-x64-  
linux.tar.gz  
tar xfz ncbi-blast-2.9.0+-x64-linux.tar.gz ..... バイナリファイルをダウンロード、展開、  
rm ncbi-blast-2.9.0+-x64-linux.tar.gz 不要なファイルの削除
```

解析ツールのダウンロードとインストール

- install_tools.shのつづき5 -

```
### Tools for "varcall" analysis
echo "installing STAR..."
wget https://github.com/alexdobin/STAR/archive/2.7.2b.tar.gz
tar xfv 2.7.2b.tar.gz                                バイナリファイルをダウンロード、展開、
rm 2.7.2b.tar.gz                                     不要なファイルの削除

echo "installing Picard..."
mkdir picard-2.20.7                                  バイナリファイルをダウンロード
cd picard-2.20.7
wget https://github.com/broadinstitute/picard/releases/download/2.20.7/picard.jar
cd ..

echo "installing GATK..."
# wget https://github.com/broadinstitute/gatk/releases/download/4.1.3.0/gatk-4.1.3.0.zip
cp /work/NGSworkshop/tool/gatk-4.1.3.0.zip ./      GATKのダウンロードは時間がかかるため、あらかじめ
unzip gatk-4.1.3.0.zip                             ダウンロードしておいたファイルをコピーする。
rm gatk-4.1.3.0.zip                               他のサーバにインストールする際には、この行をコメ
echo "END:" `date`                                 ントアウトし、1つ上の行のコメントを外して、wget
                                                    でダウンロードする。
```

解析ツールのダウンロードとインストール



ツールのインストールをするためのシェルスクリプトの実行 (実行時間：約5分)

- 普通にbashコマンドで実行すると、スクリプトの標準出力と標準エラー出力をターミナル画面に表示する。

```
$ bash ./install_tools.sh
```

- 以下のようなコマンドを実行すると、標準出力と標準エラー出力を合わせてターミナル画面に表示しながら、ファイルにも書き出してくれる。こうしておくとあとでログを見直すことができて便利。

```
$ bash ./install_tools.sh 2>&1 | tee install_tools.log
```

- 一連のコマンドが書かれたシェルスクリプトを作成すると、コマンドを連続して実行することができる。スクリプトとして残しておくと、あとで実行コマンドを確認する際にも役立つ。
- ツールによっては、解析結果の統計情報などを標準出力や標準エラー出力として出力するので、それらもファイルに保存しておくとよい。

インストールされた解析ツールを確認



全てのツールはtoolディレクトリにインストールされる。

```
$ ls tool/  
FastQC  
Jellyfish-2.3.0  
STAR-2.7.2b  
Trimmomatic-0.39  
bowtie2-2.3.5.1-linux-x86_64  
cmake-3.15.3-Linux-x86_64  
gatk-4.1.3.0  
hisat2-2.1.0  
kallisto  
ncbi-blast-2.9.0+  
picard-2.20.7  
salmon-latest_linux_x86_64  
samtools-1.9  
stringtie-2.0.3.Linux_x86_64  
trinityrnaseq-Trinity-v2.8.5
```

例えば、HISAT2は「tool/hisat2-2.1.0」以下にインストールされている。

```
$ ls tool/hisat2-2.1.0

AUTHORS          hisat2-align-s           hisat2_extract_snps_haplotypes_VCF.py
LICENSE          hisat2-align-s-debug    hisat2_extract_splice_sites.py
MANUAL           hisat2-build           hisat2_simulate_reads.py
MANUAL.markdown  hisat2-build-l         hisatgenotype.py
NEWS             hisat2-build-l-debug   hisatgenotype_build_genome.py
TUTORIAL         hisat2-build-s         hisatgenotype_extract_reads.py
VERSION          hisat2-build-s-debug   hisatgenotype_extract_vars.py
doc              hisat2-inspect         hisatgenotype_hla_cyp.py
example          hisat2-inspect-l       hisatgenotype_locus.py
extract_exons.py hisat2-inspect-l-debug hisatgenotype_modules
extract_splice_sites.py hisat2-inspect-s     hisatgenotype_scripts
hisat2            hisat2-inspect-s-debug scripts
hisat2-align-l    hisat2_extract_exons.py
hisat2-align-l-debug hisat2_extract_snps_haplotypes_UCSC.py
```

インストールされた解析ツールを確認



多くのツールは「`--help`」や「`-h`」オプションをつけて実行することで簡単な使い方や指定できるパラメータなどを確認できる。

```
$ ./tool/hisat2-2.1.0/hisat2 -h ..... -hオプションをつけてHISAT2を実行
```

HISAT2 version 2.1.0 by Daehwan Kim (infphilo@gmail.com, wwwccb.jhu.edu/people/infphilo)

Usage:

```
hisat2 [options]* -x <ht2-idx> {-1 <m1> -2 <m2> | -U <r> | --sra-acc <SRA accession number>} [-S <sam>]
```

<ht2-idx> Index filename prefix (minus trailing .X.ht2).

<m1> Files with #1 mates, paired with files in <m2>.

Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).

<m2> Files with #2 mates, paired with files in <m1>.

Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).

<r> Files with unpaired reads.

Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).

⋮ (省略)

`--version` print version information and quit

`-h/--help` print this usage message

ここでエラーが出なければ、正しくインストールされている（たぶん）

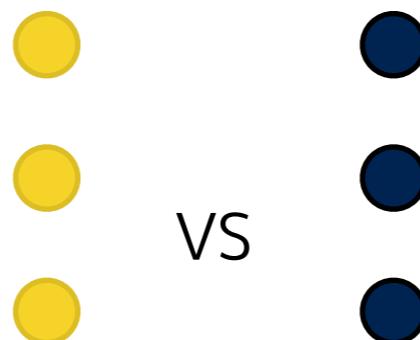
- ▶ リファレンス情報を使ったRNA-Seq解析
- ▶ De novoトランскriプトーム解析
- ▶ RNA-Seqデータを用いた多型検出

目的と使用するデータ

目的：リファレンス情報を用いて、
昼と夜でのイネの葉の遺伝子発現の違いを調べる



12:00 PM



00:00 AM

- 田んぼで栽培する、イネ（コシヒカリ）の葉身のサンプル。
- 3つの異なる生育ステージ（**3反復**）において、昼（12時）と夜（0時）にサンプリング（**2条件**）。
- Illumina社の**Stranded mRNA-Seq法**でライブラリ調製。
- Illumina社のHiSeq2000による、**101bpのPaired-endシーケンシング**。

*全データでは解析に時間がかかるため、2番染色体の特定の領域（2Mbp）にアラインメントされるリードだけを事前に抽出しており、演習ではそれを利用する。

目的と使用するデータ

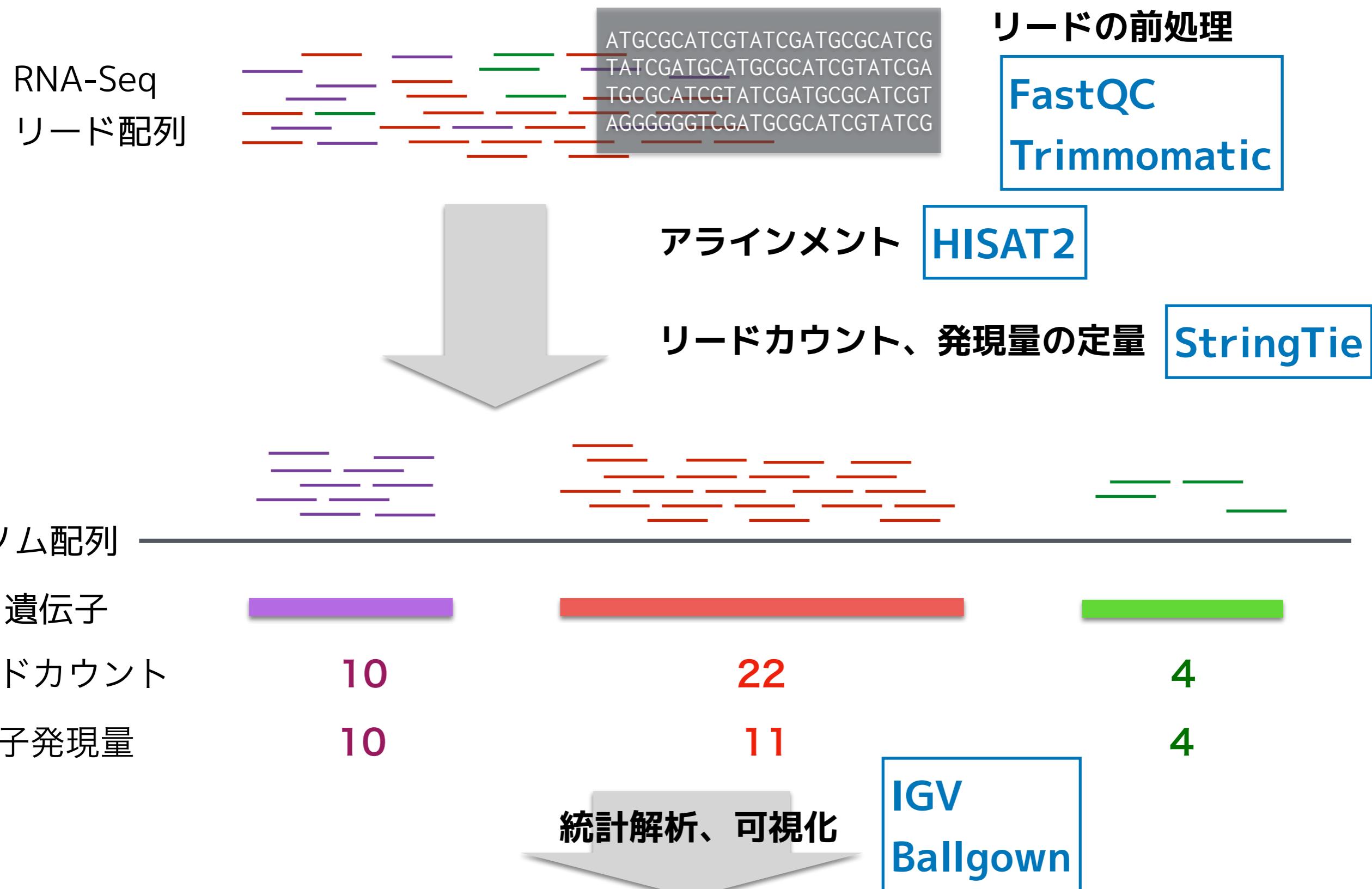
```
$ cd ..... ホームディレクトリに移動  
$ cd RNA-Seq ..... RNA-Seqディレクトリに移動  
$ ls data ..... dataディレクトリの中身を確認  
annotation.gtf ..... rice_D_rep2_r1.org.fastq.gz rice_N_rep2_r1.org.fastq.gz  
genome.dict ..... rice_D_rep2_r2.org.fastq.gz rice_N_rep2_r2.org.fastq.gz  
genome.fa ..... rice_D_rep3_r1.org.fastq.gz rice_N_rep3_r1.org.fastq.gz  
genome.fa.fai ..... rice_D_rep3_r2.org.fastq.gz rice_N_rep3_r2.org.fastq.gz  
rice_D_rep1_r1.org.fastq.gz ..... rice_N_rep1_r1.org.fastq.gz rice_protein.fa  
rice_D_rep1_r2.org.fastq.gz ..... rice_N_rep1_r2.org.fastq.gz transcript.fasta
```

dataディレクトリ中の各ファイルの説明

- ・ リファレンスゲノム配列 (genome.fa)
- ・ 遺伝子アノテーションファイル (annotation.gtf)
- ・ イネ遺伝子の塩基配列 (transcript.fasta)
- ・ mRNA-Seqリード配列 (*.fastq.gz)
- ・ イネ遺伝子のアミノ酸配列 (rice_protein.fa)



リファレンス情報を用いたRNA-Seq解析の流れ



2サンプル間比較によるDifferentially expressed gene (DEG)の検出、結果の可視化など

リファレンス情報を用いたRNA-Seq解析のためのツール



HISAT2
graph-based alignment of next generation sequencing reads to a population of genomes

JOHNS HOPKINS UNIVERSITY
CENTER FOR COMPUTATIONAL BIOLOGY
CCB

HISAT2 is a fast and sensitive alignment program for mapping next-generation sequencing reads (both DNA and RNA) to a population of human genomes (as well as against a single reference genome). Based on an extension of BWT for graphs [Sirén et al. 2014], we designed and implemented a graph FM index (GFM), an original approach and its first implementation to the best of our knowledge. In addition to using one global GFM index that represents a population of human genomes, HISAT2 also supports multiple local GFM indexes (LGFM) for each genome in the population.

HISAT2
splice-awareなリードアラインメント

HISAT2 2.0.4 Windows binary available [here](#), thanks to Andre Osorio Falcao
5/24/2016

HISAT2 2.0.4 release 5/18/2016

Version 2.0.4 is a minor release with the following changes.
◦ Improved template length estimation (the 9th column of the SAM format) of RNA-seq

<https://ccb.jhu.edu/software/hisat2/index.shtml>

StringTie
Transcript assembly and quantification for RNA-Seq

JOHNS HOPKINS UNIVERSITY
CENTER FOR COMPUTATIONAL BIOLOGY
CCB

Home Manual Examples CCB » Software » StringTie

▪ Overview
▪ News
▪ Obtaining and installing
▪ Licensing and contact
▪ Publications

StringTie
発現量の定量
遺伝子構造のアセンブル

StringTie is a fast and highly accurate transcript assembler. It uses a novel network flow algorithm as well as an efficient search space reduction strategy to find the best transcript assembly. It can align short reads to a genome, but also alignments longer sequences that have been assembled from those reads. In order to identify differentially expressed genes between experiments, StringTie's output can be processed by specialized software like Ballgown, Cuffdiff or other programs (DESeq2, edgeR, etc.).

a novel network flow algorithm for transcript assemblers, representing multiple transcript variants for each gene.

<https://ccb.jhu.edu/software/stringtie/>

Tuxedo suite tools

(TopHat, Cufflinksの後継プログラム)

[Home](#) » [Bioconductor 3.9](#) » [Software Packages](#) » [ballgown](#)

ballgown

platforms all rank 131 / 1741 post 0 / 0 / in Bioc 5 years
build ok updated before release dependencies 6 8

DOI: [10.18129/B9.bioc.ballgown](https://doi.org/10.18129/B9.bioc.ballgown) [f](#) [t](#)

Flexible, isoform-level differential expression analysis

Bioconductor version: Rel

Tools for statistical analysis and visualization of transcript

Author: Jack Fu [aut], Aly

Jeffrey T. Leek [aut, ths]

Maintainer: Jack Fu <jmfp at jhsph.edu>

Citation (from within R, enter `citation("ballgown")`):

Fu J, Frazee AC, Collado-Torres L, Jaffe AE, Leek JT (2019). *ballgown: Flexible, isoform-level differential expression analysis*. R package version 2.16.0.

Ballgown

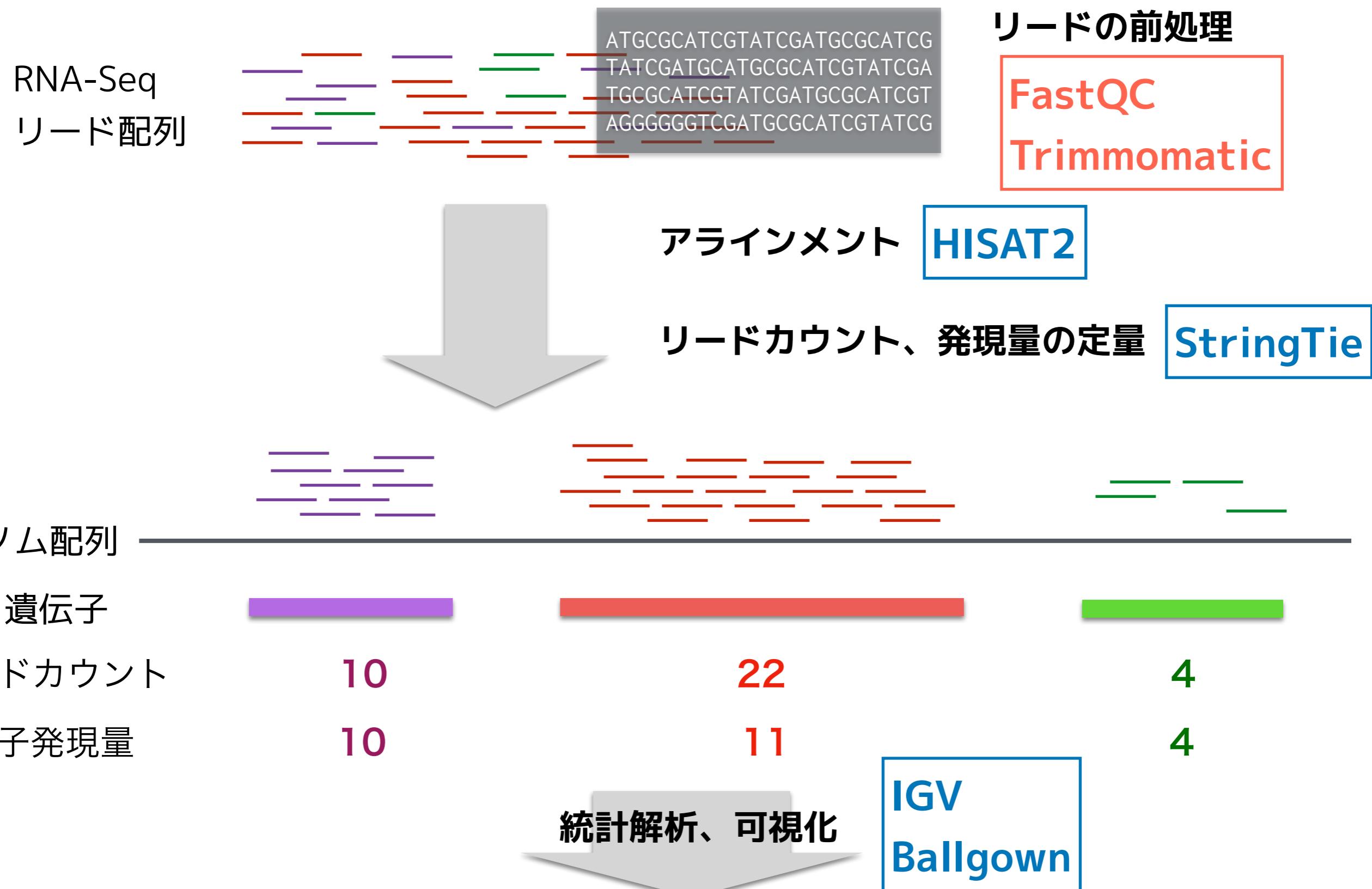
発現解析と可視化

differential expression analysis, with annotation.

Author: Jack Fu [aut], Andrew E. Jaffe [aut],

<http://bioconductor.org/packages/release/bioc/html/ballgown.html>

リファレンス情報を用いたRNA-Seq解析の流れ



2サンプル間比較によるDifferentially expressed gene (DEG)の検出、結果の可視化など

Step 1. 前処理 - 配列データのQC

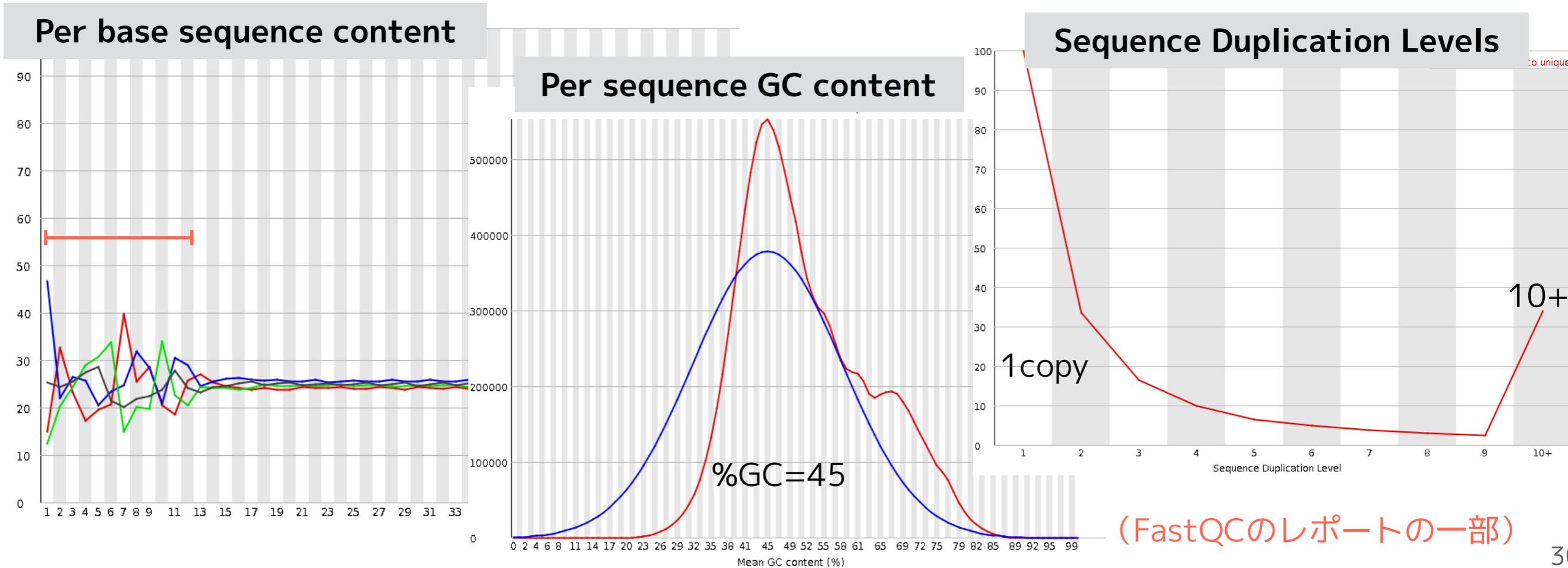
悪いデータからは良い結果は得られない。

FastQCなどのQCツールを使い、リード数や解読塩基量、クオリティ、塩基出現頻度の偏り、配列長などを事前によくチェックする！

RNA-Seqデータの特徴

- ・先頭部分の塩基頻度のバイアス
- ・GCバイアス
- ・高発現遺伝子によると思われる
Duplication levelの高さ

これらの特徴はゲノムシーケンスデータとはやや異なるので注意。



Step 1. 前処理 - 配列データのクリーニング



1. 低クオリティ塩基やアダプター配列の除去

TrimmomaticやFASTX-Toolkitなどのツールを用いる。

(例) Trimmomaticで「ILLUMINACLIP:[adapter file]:2:30:10 LEADING:15 TRAILING:15 SLIDINGWINDOW:10:15 MINLEN:50」などと指定して実行する。

2. 実験で除ききれなかったrRNA由来のリードの除去 (本演習ではスキップ)

既知のrRNA配列ライブラリに対してBowtie2等でアラインメントし、そのアンマップリード（つまり、rRNA由来ではないリード）を以降の解析で用いる。

(例) bowtie2 -x [rRNA bowti2 index file] --un-conc-gz [unmap_read.fastq]
-1 [read1.fastq] -2 [read2.fastq] -S [mapped_read.sam]

これらの処理は必須ではないが、塩基やリードの除去率やrRNAへのマップ率は、ライブラリ調製やシーケンシングに何か問題がないかを確認するための指標になる。

Step 1. 前処理 - 配列データのQCとクリーニング

リファレンス情報を用いたRNA-Seq解析用ディレクトリ「useref」に移動

```
$ cd ~/RNA-Seq/useref
```

解析用ディレクトリ内のファイルを確認

```
$ ls
step1_preprocessing.sh  step3_HISAT2.sh
step2_HISAT2-build.sh   step4_StringTie-abundance_estimation.sh
```

シェルスクリプトが4つ用意されており、順番にシェルスクリプトを実行することで解析が進められるようになっている。

```
$ bash ./step1_preprocessing.sh
$ bash ./step2_HISAT2-build.sh
$ bash ./step3_HISAT2.sh
$ bash ./step4_StringTie-abundance_estimation.sh
```

演習では各ステップを説明しながら進めていきます。

Step 1. 前処理 - 配列データのQCとクリーニング

リードの前処理をするためのシェルスクリプト

```
$ less ./step1_preprocessing.sh
```

- step1_preprocessing.shの前半 -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
FastQC_bin=$ToolDir/FastQC  
Trimmomatic_jar=$ToolDir/Trimmomatic-0.39/trimmomatic-0.39.jar  
Trimmomatic_adapters=$ToolDir/Trimmomatic-0.39/adapters
```

データや各ツールの
コマンドが置かれている
ディレクトリを変数に格納

```
export PATH=$FastQC_bin:$PATH
```

.....
export コマンドで各ツールのディレクトリをPATHに追加。
パスを指定しなくともコマンド実行が可能になる。

```
## Step1. Preprocessing by FastQC and Trimmomatic  
FASTQC_OUTDIR_BEFORE=FastQC_before_preprocess  
FASTQC_OUTDIR_AFTER=FastQC_after_preprocess  
mkdir $FASTQC_OUTDIR_BEFORE $FASTQC_OUTDIR_AFTER
```

FastQCの出力ディレクトリを作成

何度も出てくるパス等はスペルミスを避けたり、スクリプトの可読性を上げるために、変数に格納したり、exportコマンドでPATHに設定するとよい。

Step 1. 前処理 - 配列データのQCとクリーニング

- step1_preprocessing.shのつづき -

```

for DATASET in rice_D_rep1 rice_D_rep2 rice_D_rep3 rice_N_rep1 rice_N_rep2 rice_N_rep3
do ..... 繰り返し開始
    # FastQC before Trimmomatic ..... 前処理前のFastQCの実行 (r*でリード1と2を合せて指定)
    fastqc --threads 1 --nogroup --outdir $FASTQC_OUTDIR_BEFORE --format fastq
    $DataDir/${DATASET}_r*.org.fastq.gz
    # Trimmomatic ..... Trimmomaticの実行
    java -Xmx4G -Xms2G -jar $Trimmomatic_jar PE \
        -threads 1 -phred33 -trimlog Trimmomatic_${DATASET}.log \
        $DataDir/${DATASET}_r1.org.fastq.gz $DataDir/${DATASET}_r2.org.fastq.gz \
        ${DATASET}_r1.pe.fastq.gz ${DATASET}_r1.unpe.fastq.gz \
        ${DATASET}_r2.pe.fastq.gz ${DATASET}_r2.unpe.fastq.gz \
        ILLUMINACLIP:${Trimmomatic_adapters}/TruSeq3-PE-2.fa:2:30:10 \
        LEADING:15 TRAILING:15 SLIDINGWINDOW:10:15 MINLEN:50
    # FastQC after Trimmomatic ..... 前処理後のFastQCの実行
    fastqc --threads 1 --nogroup --outdir $FASTQC_OUTDIR_AFTER --format fastq ${DATASET}
    _r*.pe.fastq.gz
done ..... 繰り返し終了

```

for関数による繰り返し。変数DATASETにin以降のサンプル名が順番に格納されて実行される。

Trimmomatic Trimmomaticの実行

java -Xmx4G -Xms2G -jar \$Trimmomatic_jar PE \ \\ (バックスラッシュ) は実行時には無視され、ひと続きのコマンドとして実行される。可読性を上げるために使用。

サンプル数が6つあるため、「前処理前後のFastQCとTrimmomaticによる前処理」を各サンプルごとにfor文を使って繰り返し実行している。

Step 1. 前処理 - 配列データのQCとクリーニング



シェルスクリプトの実行 (実行時間：約3分)

```
$ bash ./step1_preprocessing.sh
```

出力ファイルの確認

```
$ ls
FastQC_after_preprocess          rice_D_rep2_r1.pe.fastq.gz    rice_N_rep2_r1.pe.fastq.gz
FastQC_before_preprocess         rice_D_rep2_r1.unpe.fastq.gz   rice_N_rep2_r1.unpe.fastq.gz
Trimmomatic_rice_D_rep1.log      rice_D_rep2_r2.pe.fastq.gz    rice_N_rep2_r2.pe.fastq.gz
Trimmomatic_rice_D_rep2.log      rice_D_rep2_r2.unpe.fastq.gz   rice_N_rep2_r2.unpe.fastq.gz
Trimmomatic_rice_D_rep3.log      rice_D_rep3_r1.pe.fastq.gz    rice_N_rep3_r1.pe.fastq.gz
Trimmomatic_rice_N_rep1.log      rice_D_rep3_r1.unpe.fastq.gz   rice_N_rep3_r1.unpe.fastq.gz
Trimmomatic_rice_N_rep2.log      rice_D_rep3_r2.pe.fastq.gz    rice_N_rep3_r2.pe.fastq.gz
Trimmomatic_rice_N_rep3.log      rice_D_rep3_r2.unpe.fastq.gz   rice_N_rep3_r2.unpe.fastq.gz
rice_D_rep1_r1.pe.fastq.gz       rice_N_rep1_r1.pe.fastq.gz    step1_preprocessing.sh
rice_D_rep1_r1.unpe.fastq.gz     rice_N_rep1_r1.unpe.fastq.gz   step2_HISAT2-build.sh
rice_D_rep1_r2.pe.fastq.gz       rice_N_rep1_r2.pe.fastq.gz    step3_HISAT2.sh
rice_D_rep1_r2.unpe.fastq.gz     rice_N_rep1_r2.unpe.fastq.gz   step4_StringTie-abundance_estimation.sh
```

- FastQC_after_preprocess: 前処理後のリードのFastQCの結果
- FastQC_before_preprocess: 前処理前のリードのFastQCの結果
- Trimmomatic_rice_*.log: Trimmomaticのログ (リードごとのトリミング情報)
- rice_*.pe.fastq.gz: 前処理後のリード配列ファイル (ペアを維持)
- rice_*.unpe.fastq.gz: 前処理後のリード配列ファイル (ペアの相方が捨てられたもの)

Trimmomaticによる前処理結果の統計情報は標準エラーとして出力される。標準エラー出力をファイルに書き出していくれば、以下のように確認できる。

```
$ less step1_preprocessing.log
...
Input Read Pairs: 18407642 Both Surviving: 16992068 (92.31%) Forward Only
Surviving: 1173984 (6.38%) Reverse Only Surviving: 159615 (0.87%) Dropped:
81975 (0.45%)
...
```

- ・ 前処理の結果、16,992,068 ペア (92.31%) が残り、このあとの解析に利用される。
- ・ ペアの片方しか残っていない (* Only Surviving) 、もしくはペアの両方が失われてしまった (Dropped) 割合があまりに多い場合は、トリミングのパラメータを緩めるなど再検討する。あまりに酷い場合は再シーケンシングも検討。

*この例は演習用データではなく、その元となった全ゲノムデータによる結果。
演習用データは前処理後にゲノムにアラインメントできたリードのみ抽出しているため、Both Survivingが100%になっているはず。

前処理で見つかる問題と対応策

1. 低クオリティ塩基が多い

シークエンシング時の問題であることが多いので読み直す。

2. アダプター配列の混入率が高い

RNAの濃度が薄い、解読リード長に対してインサート長が短いなど。ライブラリ調製からやり直す。

3. rRNAの混入率が高い

ポリA精製やrRNAの除去がうまくいっていない。ライブラリ調製からやり直す。

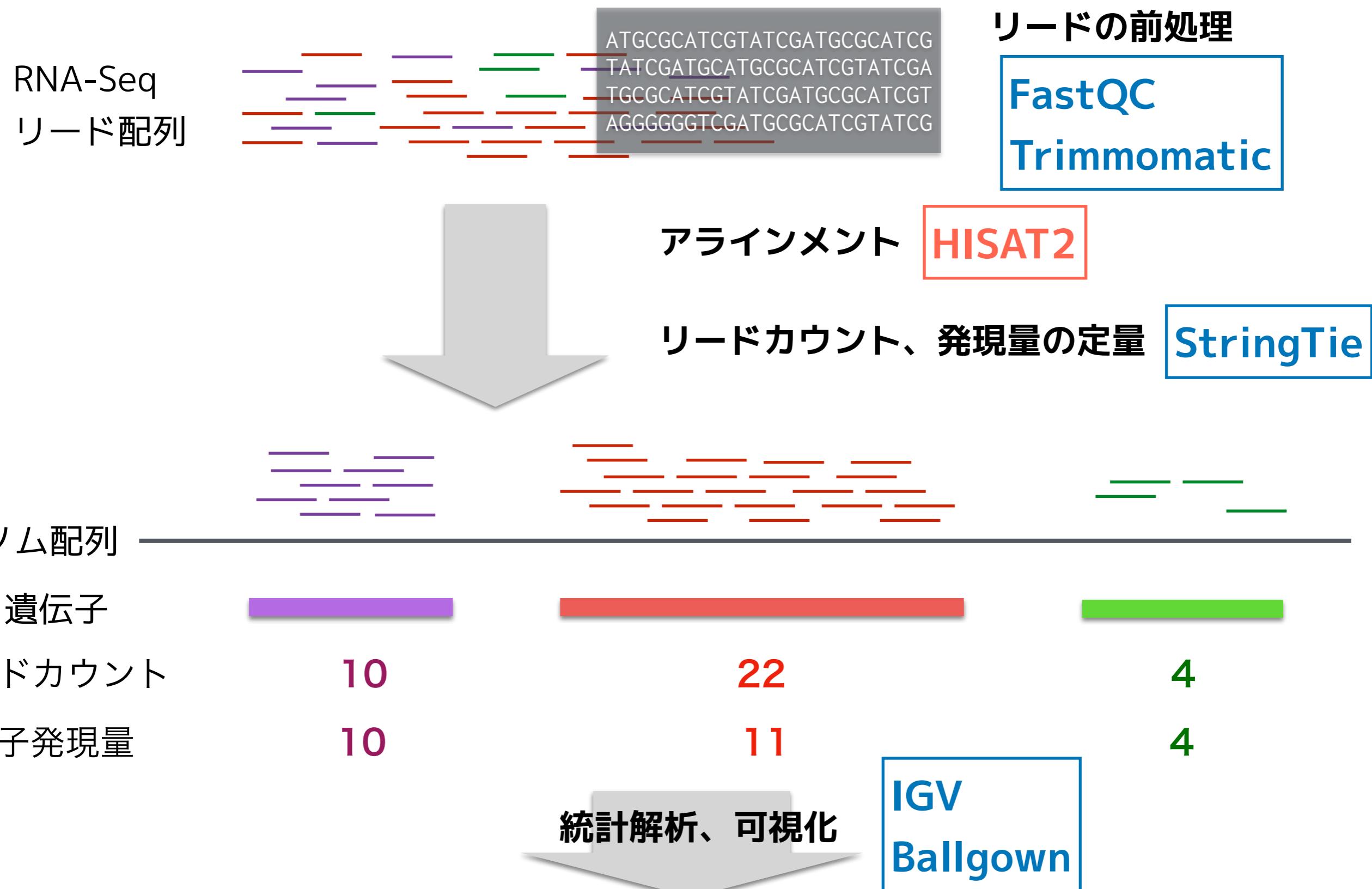
4. ゲノム、転写産物配列へのマップ率が低い

コンタミ等が疑われる。サンプリングからやり直した方が良い場合もある。

5. 有効なリード数が少ない

シークエンシングを追加する。生物種や対象組織、目的等によっても必要なリード数は異なるが、個人的にはイネの葉であれば1反復当たり1-2千万（10-20 million）リードがあれば十分だと思う。

リファレンス情報を用いたRNA-Seq解析の流れ

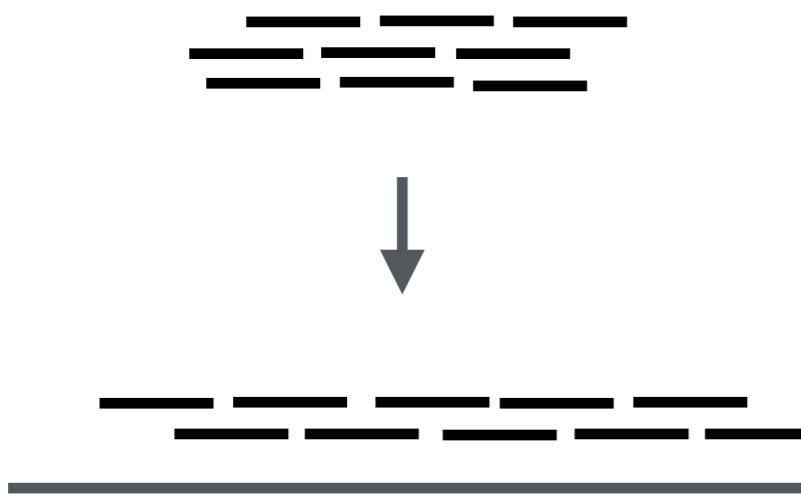


2サンプル間比較によるDifferentially expressed gene (DEG)の検出、結果の可視化など

RNA-Seqリードとゲノム配列のアラインメントの難しさ

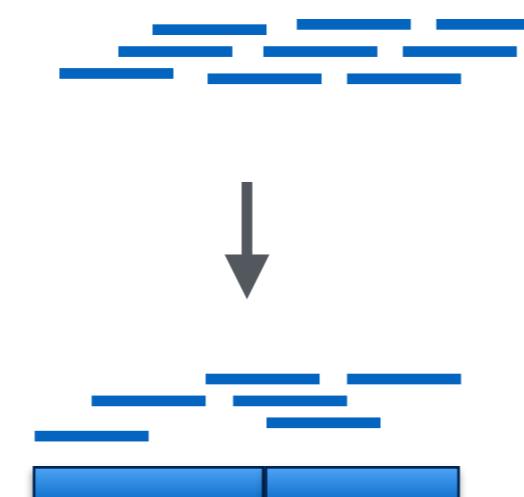


ゲノムリシーケンスリード



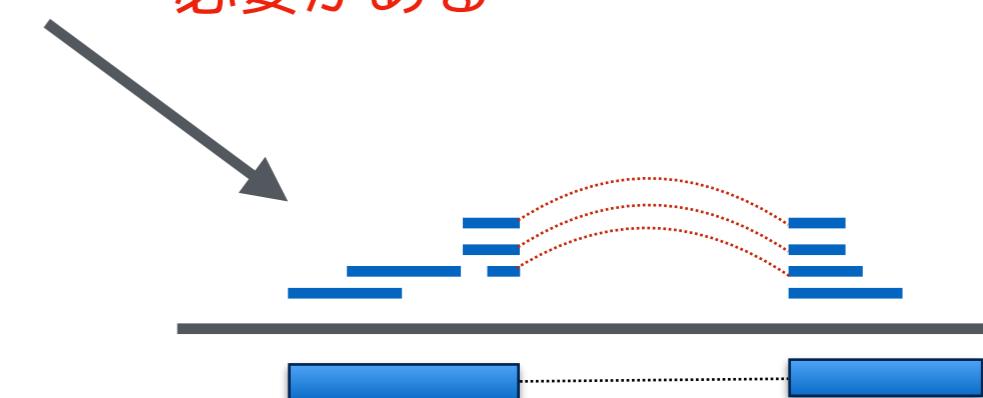
ゲノム配列

RNA-Seqリード



転写産物配列

長いギャップ（イントロン）を
含むアラインメントを考慮する
必要がある



ゲノム配列

- ・ ゲノムリシーケンス解析でよく使われるBWAやBowtie2などはRNA-Seqデータ解析には向かない。
- ・ RNA-Seqリードアラインメントに特化した様々なツール(HISAT2、STAR2、GSNAPなど)が開発されている。

Step 2. HISAT2のためのインデックスの作成

HISAT2のためのインデックスを作成するシェルスクリプト

```
$ less ./step2_HISAT2-build.sh
```

- step2_HISAT2-build.sh -

```
DataDir=$HOME/RNA-Seq/data
ToolDir=$HOME/RNA-Seq/tool
HISAT2_bin=$ToolDir/hisat2-2.1.0
Samtools_bin=$ToolDir/samtools-1.9

export PATH=$HISAT2_bin:$Samtools_bin:$PATH

### Step2. Build index of the reference genome sequence by hisat2-build and samtools
# make splice site and exon position info
python $HISAT2_bin/hisat2_extract_splice_sites.py $DataDir/annotation.gtf > ss.tab
python $HISAT2_bin/hisat2_extract_exons.py $DataDir/annotation.gtf > exon.tab
# build index for HISAT2
hisat2-build --ss ss.tab --exon exon.tab $DataDir/genome.fa genome ..... HISAT2のための
# build index for IGV etc. ..... インデックスの作成
samtools faidx $DataDir/genome.fa ..... IGVによる可視化時に必要なゲノム配列のインデックス作成
```

HISAT2用のhierarchical index作成時に、既知のスプライスサイト(ss.tab)やExon(exon.tab)の位置情報を指定している。これによりアラインメントの精度と効率が向上する。

Step 3. HISAT2のためのインデックスの作成



シェルスクリプトの実行 (実行時間：約1分)

```
$ bash ./step2_HISAT2-build.sh
```

作成されたインデックスファイルの確認

```
$ ls genome.*  
genome.1.ht2  genome.3.ht2  genome.5.ht2  genome.7.ht2  
genome.2.ht2  genome.4.ht2  genome.6.ht2  genome.8.ht2  
$ ls ../data/genome.fa*  
../data/genome.fa  ../data/genome.fa.fai
```

- genome.*.ht2ファイルはHISAT2のためのインデックス
- genome.fa.faiは、IGVなどでの可視化に必要なインデックス

遺伝子アノテーションファイル (GTF2/GFF3)



ゲノム上のどこにどんな構造の遺伝子が存在するのかを記載した情報。イネであればRAP-DBから代表転写産物のアノテーション情報 (GFF/GTF) がダウンロードできる。

<http://rapdb.dna.affrc.go.jp/download/irgsp1.html>

The screenshot shows the RAP-DB website interface. At the top, there is a navigation bar with icons for Home, News, About, Browser, Tools, Download (which is highlighted in red), Documents, Publications, and Links. Below the navigation bar, there is a search bar with 'Keywords' and 'Search' buttons, and an 'Advanced' button. The main content area is titled 'Annotation data on Os-Nipponbare-Reference-IRGSP-1.0'. Under this title, there are two sections: 'Genome sequences' and 'Gene set (genes supported by FL-cDNAs, ESTs or proteins)'. The 'Genome sequences' section contains links for genome assemblies (12 chromosomes) and unanchored sequences, both with download links. The 'Gene set' section contains links for gene structure and function information in GFF format, gene structure (only exon) information in GTF format, gene annotation information in tab-delimited text format, and gene sequences (CDS + UTRs + introns) in FASTA format.

Annotation data on Os-Nipponbare-Reference-IRGSP-1.0

Genome sequences

- Genome sequence (Os-Nipponbare-Reference-IRGSP-1.0)
Genome assemblies (12 chromosomes)* [\[DOWNLOAD\]](#) (gz file, 116MB, [MD5 checksum](#))
Unanchored sequences* [\[DOWNLOAD\]](#) (gz file, 356KB, [MD5 checksum](#))
(*) Sequences are masked by [Censor](#) with [MIPS](#) and [MSU](#) repeat data. The masked regions are replaced by lowercase letters.
- Chromosome sequences of the aus rice cultivar 'Kasalath'.
[\[DOWNLOAD\]](#) (gz file, 199MB)

Gene set (genes supported by FL-cDNAs, ESTs or proteins)

- Gene structure and function information in GFF format.
[\[DOWNLOAD\]](#) (gz file, 15MB)
- Gene structure (only exon) information in GTF format.
[\[DOWNLOAD\]](#) (gz file, 2.1MB)
- Gene annotation information in tab-delimited text format.
[\[DOWNLOAD\]](#) (gz file, 2.6MB)
- Gene sequences (CDS + UTRs + introns) in FASTA format.
[\[DOWNLOAD\]](#) (gz file, 39MB)

- 様々なデータベースから遺伝子の位置や機能情報が提供されているが、書式や記載方法は必ずしも統一されていない。
- HISAT2やStringTieで正しく扱えるような形式に整形しておくと解析に便利。RAP-DBのGTFファイルはHISAT2に対応。

色々な生物のアノテーションファイルを提供しているサイト

Illumina社 iGenomes

https://support.illumina.com/sequencing/sequencing_software/igenome.html

EnsemblPlants

<http://plants.ensembl.org/index.html>

Phytozome

<http://phytozome.jgi.doe.gov/pz/portal.html>

HISAT2やStringTieで使用可能なGTFファイルの書式



遺伝子の位置やID、アノテーション情報などが記載されたタブ区切りのテキストファイル

```
$ less ./data/annotation.gtf
...
chr02    irgsp1_rep    transcript    30094300      30099072      .          +          .
gene_id "Os02g0724000"; transcript_id "Os02t0724000-01"; gene_name "DTH2"; note "CONSTANS-like
protein, Heading promotion under long-day condition";
chr02    irgsp1_rep    exon        30094300      30094498      .          +          .          gene_id
"Os02g0724000"; transcript_id "Os02t0724000-01";
chr02    irgsp1_rep    exon        30096198      30096893      .          +          .          gene_id
"Os02g0724000"; transcript_id "Os02t0724000-01";
chr02    irgsp1_rep    exon        30097390      30097590      .          +          .          gene_id
"Os02g0724000"; transcript_id "Os02t0724000-01";
chr02    irgsp1_rep    exon        30097861      30098165      .          +          .          gene_id
"Os02g0724000"; transcript_id "Os02t0724000-01";
chr02    irgsp1_rep    exon        30098451      30099072      .          +          .          gene_id
"Os02g0724000"; transcript_id "Os02t0724000-01";
...
...
```

- 上の例では6行で1つの転写産物（5 exons）の構造とアノテーション情報を示す。
- 「gene_id」（遺伝子座ID）や「transcript_id」（転写産物ID）は、遺伝子座や転写産物を対象にした発現解析を行うために必須。
- 「gene_name」（遺伝子名やシンボル）を付けておくと、解析結果に含まれるので便利。

Step 3. HISAT2によるRNA-Seqリードのアラインメント



6サンプル（2条件、3反復）を1つずつ順番にHISAT2でアラインメントするためのシェルスクリプト。

```
$ less step3_HISAT2.sh
```

- step3_HISAT2.sh -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
HISAT2_bin=$ToolDir/hisat2-2.1.0  
Samtools_bin=$ToolDir/samtools-1.9  
  
export PATH=$HISAT2_bin:$Samtools_bin:$PATH  
  
### Step3. Align reads to the reference genome by HISAT2  
HISAT2_COMMON_PARAM="--threads 1 --min-intronlen 20 --max-intronlen 10000 --dta --rna-strandness RF -x genome"  
for DATASET in rice_D_rep1 rice_D_rep2 rice_D_rep3 rice_N_rep1 rice_N_rep2 rice_N_rep3  
do  
    hisat2 $HISAT2_COMMON_PARAM -1 ${DATASET}_r1.pe.fastq.gz -2 ${DATASET}_r2.pe.fastq.gz  
    -S ${DATASET}.sam  
    samtools sort -o ${DATASET}.bam ${DATASET}.sam  
    samtools index ${DATASET}.bam  
    rm ${DATASET}.sam  
done
```

共通パラメータ

共通パラメータ、リードファイル、
出力ファイルを指定してHISAT2を実行

SAM形式で出力されたアラインメントをソート、BAM形式へ変換したのち、BAMインデックスを作成し、SAMを削除

Step 3. HISAT2によるRNA-Seqリードのアラインメント



シェルスクリプトの実行 (実行時間：約1分)

```
$ bash ./step3_HISAT2.sh
```

作成されたアラインメントファイルとそのインデックスの確認

```
$ ls *.bam*
rice_D_rep1.bam      rice_D_rep3.bam      rice_N_rep2.bam
rice_D_rep1.bam.bai  rice_D_rep3.bam.bai  rice_N_rep2.bam.bai
rice_D_rep2.bam      rice_N_rep1.bam      rice_N_rep3.bam
rice_D_rep2.bam.bai  rice_N_rep1.bam.bai  rice_N_rep3.bam.bai
```

- *.bamファイルはHISAT2によるアラインメントファイル。IGVなどで読み込み可能。
- *.baiファイルはアラインメント情報のインデックス

HISAT2によるRNA-Seqリードのアラインメント結果の確認



HISAT2によるアラインメントの統計情報は標準エラー出力に書き出される。

```
$ less step3_HISAT2.stderr
```

```
16992068 reads; of these:
```

```
 16992068 (100.00%) were paired; of these:
```

```
    491486 (2.89%) aligned concordantly 0 times
```

```
    16005810 (94.20%) aligned concordantly exactly 1 time
```

```
    494772 (2.91%) aligned concordantly >1 times
```

```
----
```

```
491486 pairs aligned concordantly 0 times; of these:
```

```
    240097 (48.85%) aligned discordantly 1 time
```

```
----
```

```
251389 pairs aligned 0 times concordantly or discordantly; of these:
```

```
    502778 mates make up the pairs; of these:
```

```
        344266 (68.47%) aligned 0 times
```

```
        140870 (28.02%) aligned exactly 1 time
```

```
        17642 (3.51%) aligned >1 times
```

```
98.99% overall alignment rate
```

*この例は演習用データではなく、
全ゲノムデータによる結果。

- ・ 全16,992,068リードのうち、98.99%がアラインメントされている。
- ・ 温帯ジャポニカのリードを日本晴リファレンスゲノムにアラインメントした場合、特に問題がなければ90%台後半のアラインメント率を示すのが一般的。

HISAT2によるRNA-Seqリードのアライメント結果の確認



samtoolsを使い、個々のリードのアライメント情報を見ることができる。

```
$ ./tool/samtools-1.9/bin/samtools view ./rice_D_rep1.bam | less
```

MG00HS14:530:C6M7YACXX:8:2208:4438:63887	161	chr02	14245740	60
67M = 31527665	17282026			
TGAACGCTGGCGGCATGCTAACACATGCAAGTCGAACGGGAAGTGGTGTTCAGTGGCGAACGGG				
@@@A7DDDD811@FAFGFB9?BAD38B>FDFDFFFTEA@BFB>F==@a:?=>AADCABABBBB'05B			AS:i:0	XN:i:0
XM:i:0 X0:i:0		DP XS:A:+ NH:i:1		
MG00HS14:530:101M =	75M 112N 21M	chr02	23112840	60
CATCCCTGTCC	リード	← 1S		
CCACTGGCCGT	ゲノム配列		ATGTGTTCTCCATCACCGGTCGTGGTACCGTTG	
DCBB;@(;@CCG			=CC>@7==@@E=;?=?	
XG:i:0 NM:i:0 MD:Z:101	YS:i:0 YT:Z:DP XS:A:+ NH:i:1	AS:i:0 XN:i:0 XM:i:0 X0:i:0		
MG00HS14:530:C6M7YACXX:8:1103:1603:42247	113	chr02	23845634	60
75M112N21M1S = 31055654	7210007			
GTCAGCTTCGCTGCCGCCGTGCTGGGGCTCCTCGCAGCCATCCTCGGGTTCGTCGCGGAAGGCGCCAAGTCCAATTGTTCTGCG				
GTTCGACGGGG C@>CB>><BDB7DDDDBDDC?8(B?BA?B@>;DDDCC@A?<DDDDDDDB@B??				
<FFFHHGEJJIIIIJIIIEFJIHFIIIHGGJIGAFHHFFFFFCCC AS:i:-14 XN:i:0 XM:i:0				
X0:i:0 XG:i:0 NM:i:0 MD:Z:96 YS:i:-2 YT:Z:DP XS:A:+ NH:i:1				

CIGAR 75M 112N 21M

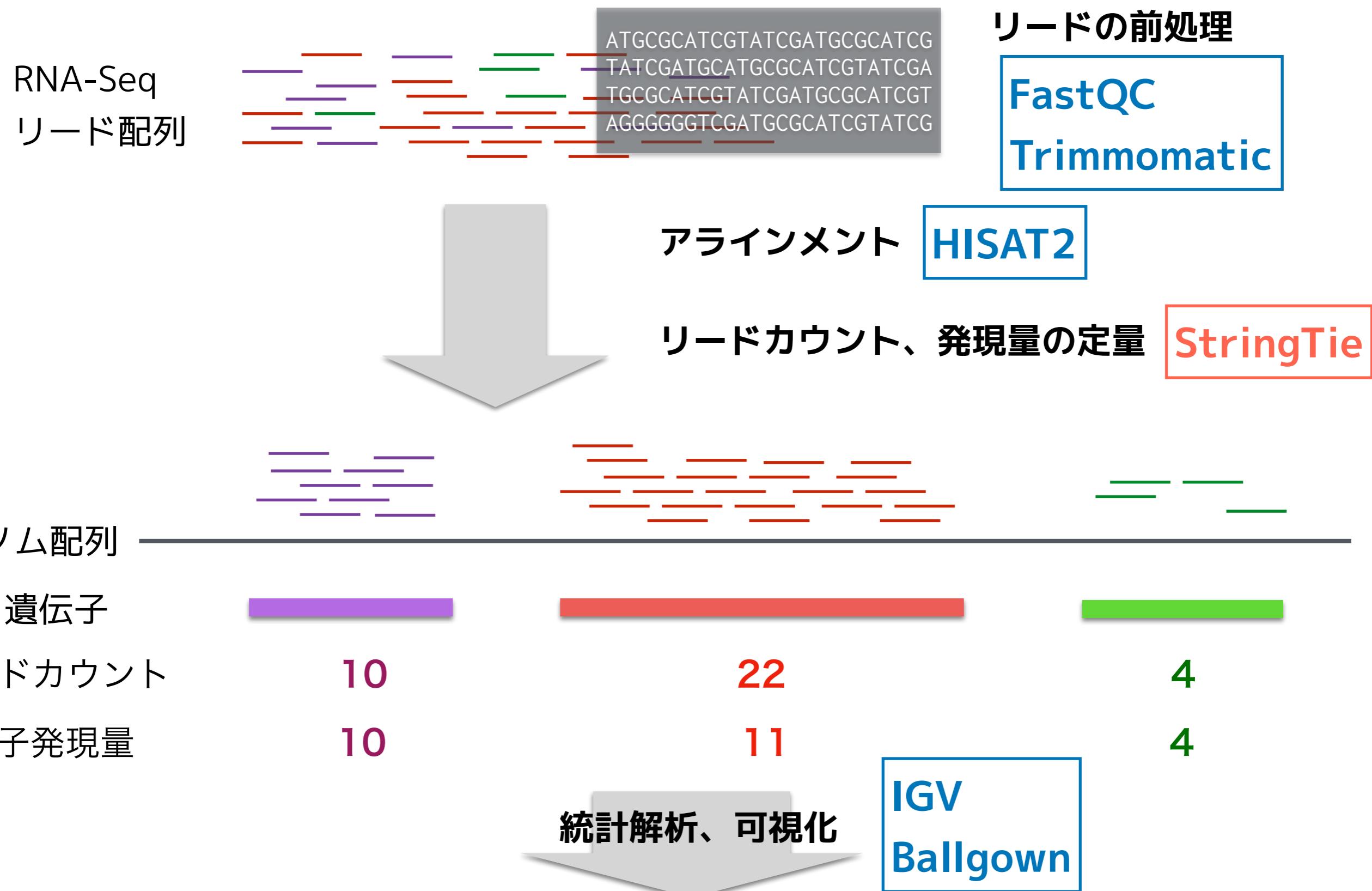
リード —————— ← 1S

ゲノム配列 ——————

75M112N21M1S

- ・ SAMフォーマットについては<http://samtools.github.io/hts-specs/SAMv1.pdf> を参照。
- ・ イントロンをまたぐリードアライメントは「75M112N21M1S」のように、イントロン部分が"N"で表されている。112bpのイントロンを挟むアライメントであるという意味。

リファレンス情報を用いたRNA-Seq解析の流れ



Step 4. StringTieによる遺伝子発現量の定量

StringTieによって、遺伝子アノテーション（GTF）と各サンプルのアラインメント情報（BAM）を元に、各遺伝子ごとにリードカウントや発現量を計算するシェルスクリプト。

```
$ less step4_StringTie-abundance_estimation.sh
```

- step4_StringTie-abundance_estimation.sh -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
StringTie_bin=$ToolDir/stringtie-2.0.3.Linux_x86_64  
  
export PATH=$StringTie_bin:$PATH  
  
### Step4. Estimate transcript abundances  
STRINGTIE_COMMON_PARAM="-e -B"  
for DATASET in rice_D_rep1 rice_D_rep2 rice_D_rep3 rice_N_rep1 rice_N_rep2 rice_N_rep3  
do  
    stringtie $STRINGTIE_COMMON_PARAM -G $DataDir/annotation.gtf -o ballgown/${DATASET}/${DATASET}.gtf ${DATASET}.bam  
done
```

→ 遺伝子アノテーションと出力ファイル、
アラインメントデータを指定して、
StringTieを実行し、発現量を定量。

Step 4. StringTieによる遺伝子発現量の定量



シェルスクリプトの実行 (実行時間：約10秒)

```
$ bash ./step4_StringTie-abundance_estimation.sh
```

サンプルごとの発現量やCoverageデータの確認

```
$ ls ballgown/rice_D_rep1  
e2t.ctab    e_data.ctab   i2t.ctab   i_data.ctab   rice_D_rep1.gtf   t_data.ctab
```

各サンプルごとに指定したディレクトリに結果が出力されている。

- rice_*.gtf : 遺伝子構造と遺伝子発現量
- e2t.ctab : exon id と transcript id の対応表
- e_data.ctab : exonごとの支持するリード数
- i2t.ctab : intron id と transcript id の対応表
- i_data.ctab : intronごとの支持するリード数
- t_data.ctab : transcriptごとの支持するリード数や発現量情報

StringTieの出力結果の確認

transcriptごとの支持するリード数や発現量情報 (t_data.ctab) の確認

```
$ less ballgown/rice_D_rep1/t_data.ctab
```

t_id	chr	strand	start	end	t_name	num_exons	length	gene_id
gene_name		cov			FPKM			
1	chr02	-	29998326		30002783			0s02t0722500-01 9 1436
0s02g0722500		-	5.784122		228.790451			
2	chr02	+	30004088		30004574			0s02t0722600-01 1 487
0s02g0722600		-	0.000000		0.000000			
3	chr02	+	30008280		30008810			0s02t0722650-00 1 531
0s02g0722650		-	2.084746		82.461945			
4	chr02	+	30011066		30015609			0s02t0722700-01 7 1436
0s02g0722700		-	121.268120		4796.750000			
5	chr02	-	30015998		30025935			0s02t0722800-01 13 4485
0s02g0722800	0sWD40-53		4.016716		158.880859			
6	chr02	-	30016149		30018289			0s02t0722800-02 6 1316
0s02g0722800		-	38.850327		1536.721436			
7	chr02	+	30051750		30053275			0s02t0723200-01 1 1526
0s02g0723200	0sGT7		0.394495		15.604217			

- 1転写産物／1行で位置や構造情報、transcript id やgene id、遺伝子名、支持するリード数、発現量 (FPKM) が記載されている。
- 点線 ----- は遺伝子座の区切り。

遺伝子発現量の指標（RPKM/FPKM）

シーケンス量や遺伝子の長さで正規化した発現量指標

RPKMやFPKM=Read (Fragment) 数／総リード数(M)／遺伝子の長さ(kb)

RPKM: reads per kilobase of exon model per million mapped reads

FPKM: fragments per kilobase of transcript per million fragments mapped

条件1

リードカウント（≠発現量）

13リード



6リード



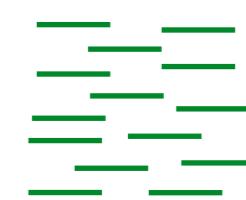
$$\text{RPKM: } 13/2/19 = \mathbf{0.34} \quad 6/1/19 = \mathbf{0.32}$$

条件2

18リード



14リード



$$18/2/32 = \mathbf{0.28} \quad 14/1/32 = \mathbf{0.44}$$

- 全リード中に含まれるある特定の転写産物由来のリードの割合。つまり、サンプル中の全RNAのうちのどれくらいがある特定の転写産物由来かを表す相対的な値。
- 発現する遺伝子の顔ぶれやたくさんの遺伝子の発現量が大きく異なるサンプル間の比較ではサンプル間で発現量を正規化する（発現量の分布を揃える）必要がある。

遺伝子発現量の指標（TPMとRPKM/FPKM）



	RPKM/FPKM	TPM
Step.1	総リード数を補正 (100万リード換算)	遺伝子の長さを補正 (1kb換算)
Step.2	遺伝子の長さを補正 (1kb換算)	総リード数を補正 (100万リード換算)

- 全遺伝子のTPM(transcripts per million)の総和はサンプル間で等しいので、個々の遺伝子のTPM値はそのままサンプル間で比較が可能。
- 最近はTPMを使った比較が勧められている。

- Wagner et al. (2012) Theory in Biosciences
- <https://www.rna-seqblog.com/rpkf-fpkf-and-tpm-clearly-explained/>

おまけ：新規転写産物構造の予測

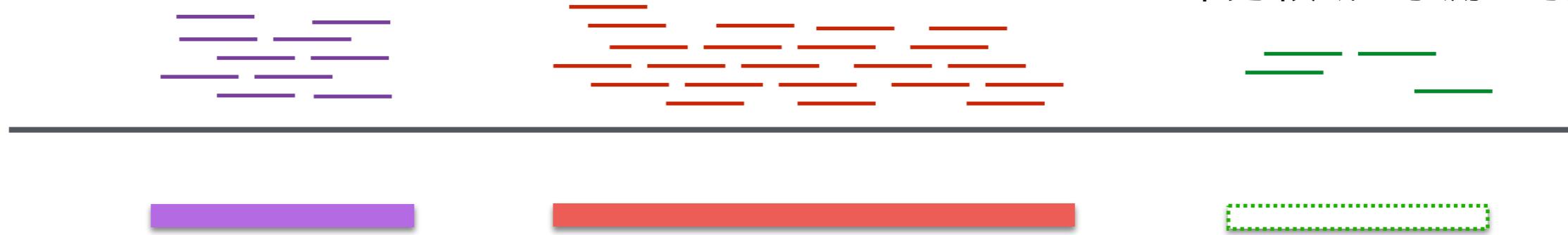


RNA-Seq

リード配列

ゲノム配列

遺伝子



RNA-Seqのアラインメント
から転写領域の予測が可能

1. サンプルごとにStringtieを実行し、遺伝子構造を予測 (sample*.gtfとして出力)

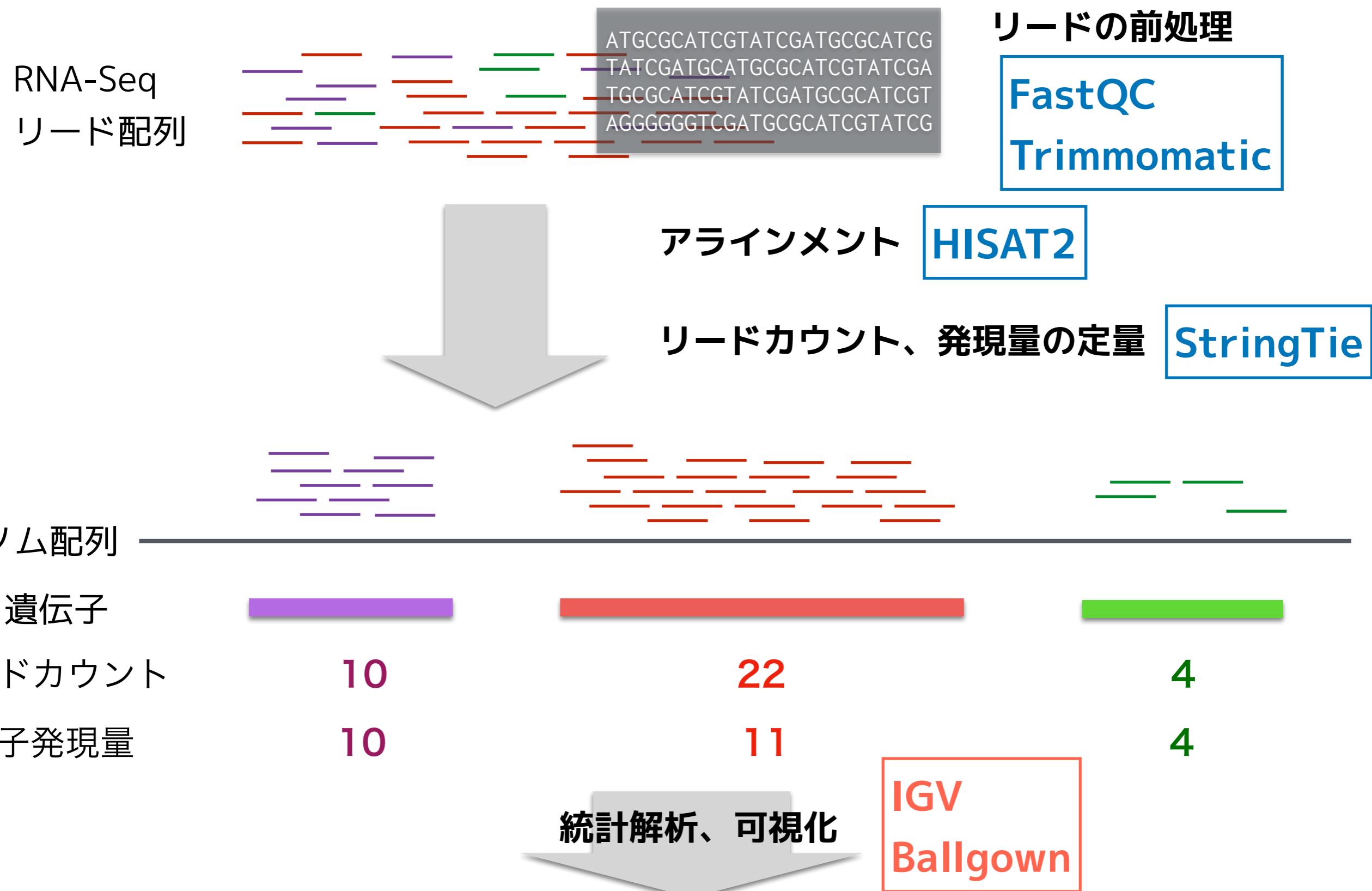
```
$ stringtie -o sample1.gtf -l sample1 sample1.bam  
$ stringtie -o sample2.gtf -l sample2 sample2.bam  
$ ...
```

2. 得られたサンプルごとの遺伝子構造 (*.gtf) をマージする。

```
$ stringtie --merge -G annotation.gtf -o stringtie_merged.gtf assemblies.txt
```

- assemblies.txtは、1.の出力のGTFファイル名が1行ごとに並んだテキストファイル
- -Gオプションで指定することで、既存の遺伝子構造 (annotation.gtf) も含めて、遺伝子構造を統合してくれる。
- step4のStringtie実行時に、この統合された遺伝子構造 (stringtie_merged.gtf) を指定することで予測遺伝子も含めた発現量が得られる。

リファレンス情報を用いたRNA-Seq解析の流れ

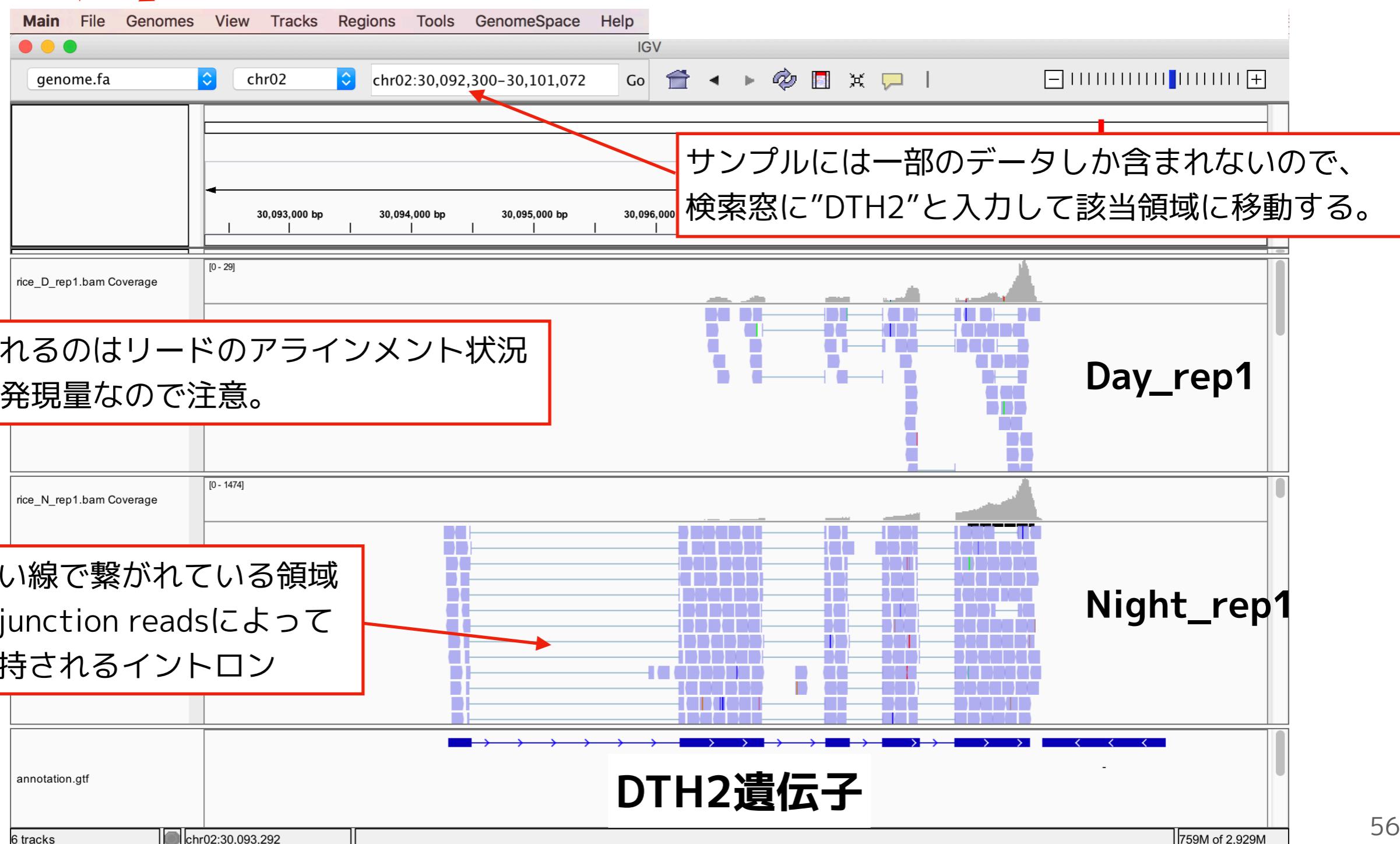


2サンプル間比較によるDifferentially expressed gene (DEG)の検出、結果の可視化など

IGVを用いたRNA-Seqデータの可視化

デスクトップ上の「workshop/RNA-Seq/useref/alignment」中のリファレンスゲノムと遺伝子アノテーション、HISAT2が出力するBAMファイルを読み込む。

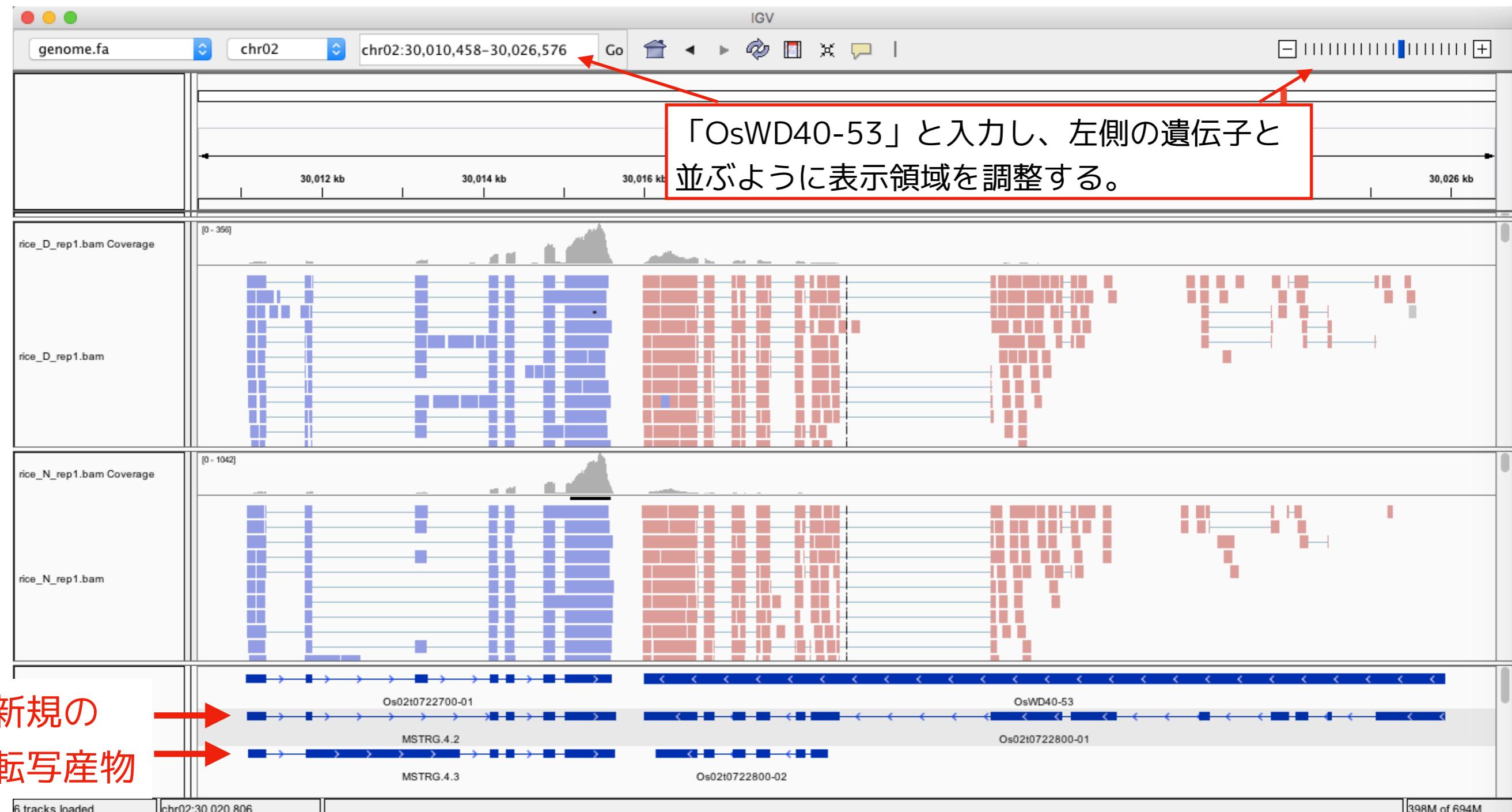
2. 「Load from File」からGTFファイルやBAMファイルを読み込む
1. 「Load Genome from File」からFASTAファイル（ゲノム配列）を読み込む



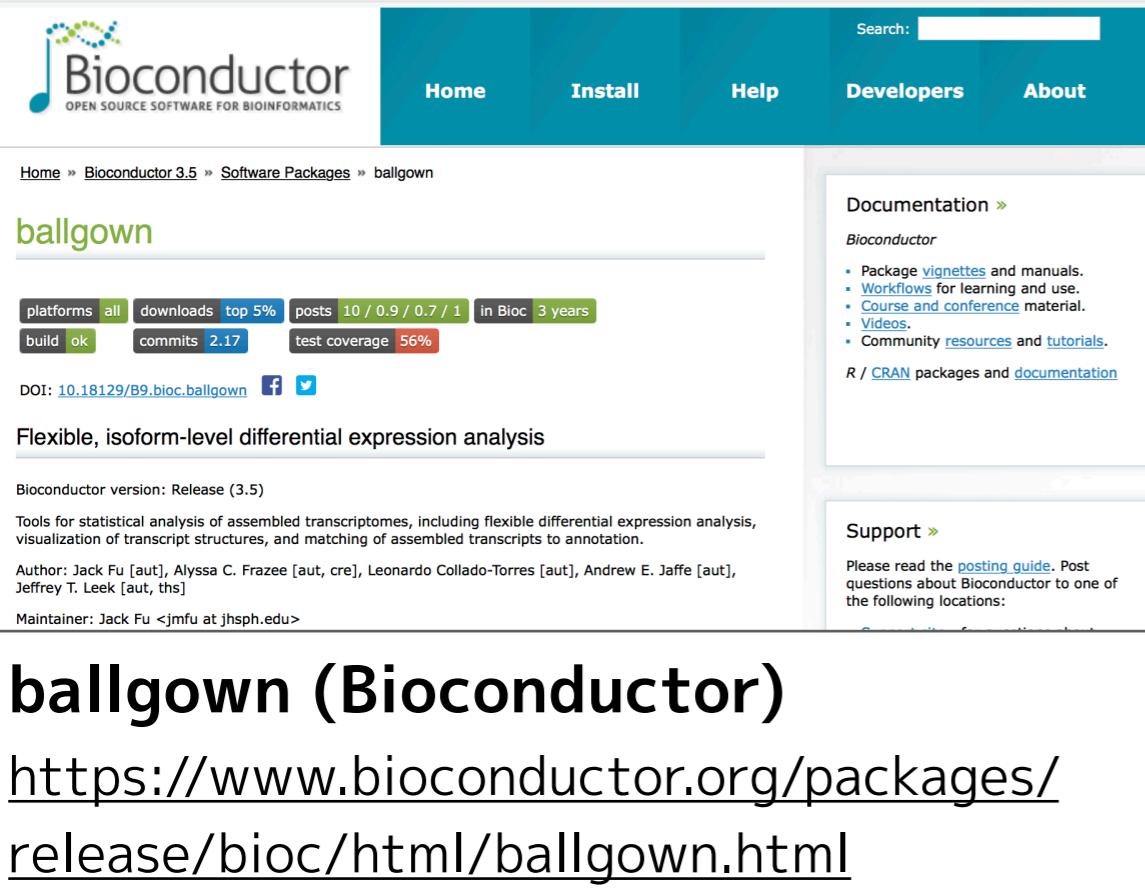
IGVによるRNA-Seqデータのアライメント、遺伝子構造の可視化



- 本演習データはIllumina stranded mRNA-Seq法によるものなので、ペアのリード2の向きと転写方向が一致する（アライメントのトラック上で右クリックし、[Color alignments by] -> [first-of-pair strand] を選択すると色で区別できるようになる）。
- 新規の転写産物構造にはMSTRG.XX.XXといったIDが振られている。



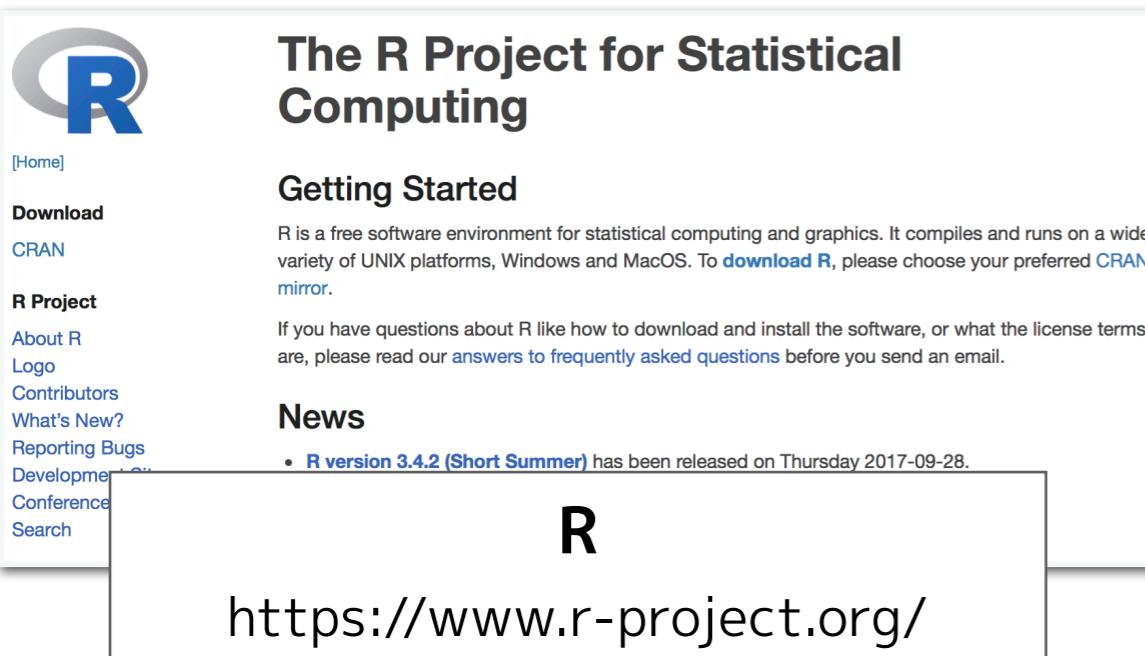
R/Bioconductorによる統計解析と可視化



The screenshot shows the Bioconductor website for the ballgown package. The top navigation bar includes links for Home, Install, Help, Developers, and About. A search bar is at the top right. Below the navigation, the URL is [Home » Bioconductor 3.5 » Software Packages » ballgown](#). The main content area is titled "ballgown" and describes it as "Flexible, isoform-level differential expression analysis". It includes sections for Documentation (with links to Bioconductor vignettes, workflows, course material, videos, and community resources), Support (with a posting guide), and package statistics (platforms: all, downloads: top 5%, posts: 10 / 0.9 / 0.7 / 1, build: ok, commits: 2.17, test coverage: 56%). DOI: [10.18129/B9.bioc.ballgown](#) with social media links for Facebook and Twitter.

ballgown (Bioconductor)

<https://www.bioconductor.org/packages/release/bioc/html/ballgown.html>



The screenshot shows the R Project for Statistical Computing homepage. The top navigation bar includes links for Home, Download, CRAN, R Project, About R, Logo, Contributors, What's New?, Reporting Bugs, Development, Conference, and Search. The main content area features the R logo and the text "The R Project for Statistical Computing". Below this is a "Getting Started" section explaining that R is a free software environment for statistical computing and graphics. It includes a link to download R from CRAN mirrors and a note about license terms. There is also a "News" section with a link to the latest R version 3.4.2 (Short Summer) release date (2017-09-28). At the bottom left is a large R logo, and at the bottom right is the URL <https://www.r-project.org/>.

ballgownは、R/Bioconductorのパッケージの一つであり、StringTieの結果を読み込んで、「発現比較解析（統計解析）」や「発現情報の可視化」の手法を提供している。

利用するためには統計計算とグラフィックスのための言語・環境である「R」をインストールする必要がある。また、RStudioはRの統合開発環境であり、使いやすいGUIを備えておりとても便利。



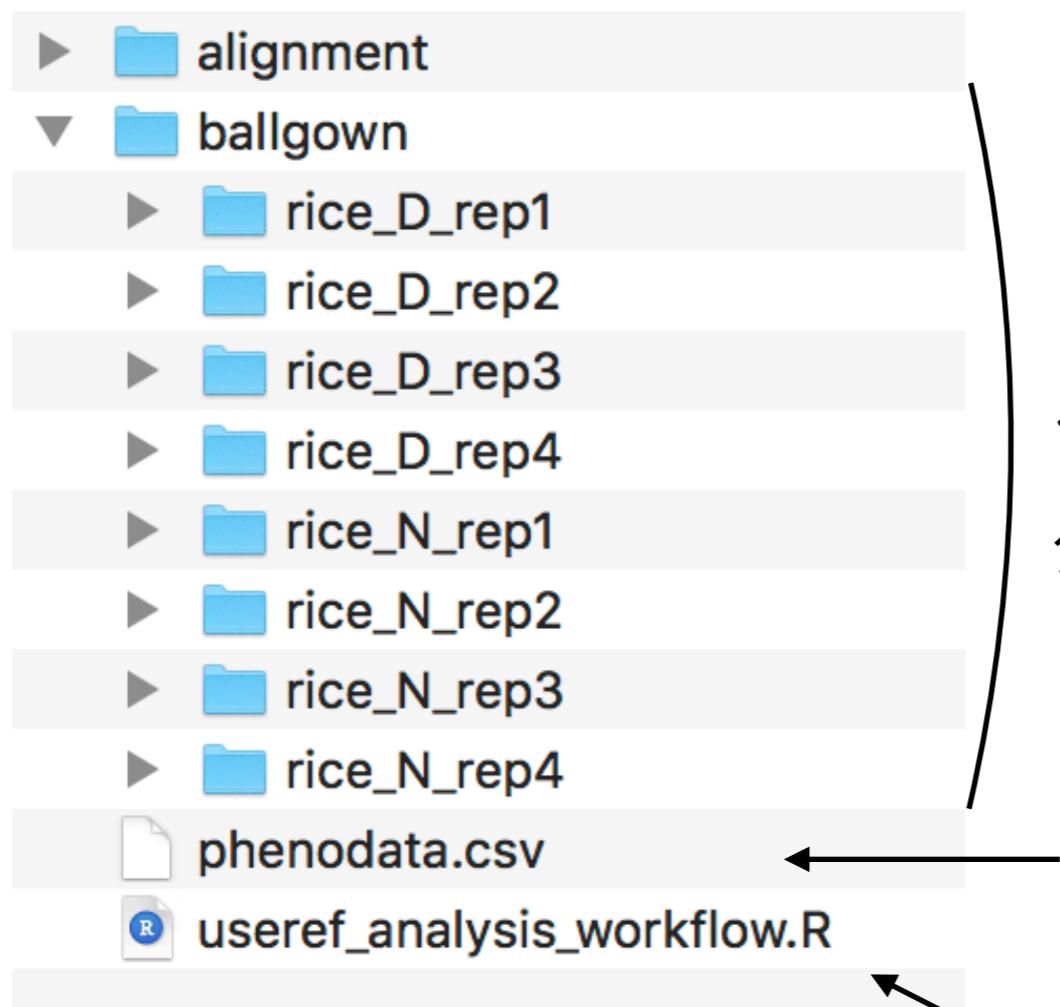
The screenshot shows the RStudio homepage. The top navigation bar includes links for rstudio::conf, Products, Resources, Pricing, About Us, Blogs, and a search icon. The main content area features the RStudio logo and the text "Open source and enterprise-ready professional software for R". It includes links for "Download RStudio", "Discover Shiny", "shinyapps.io Login", and "Discover RStudio Connect". Below this are images of the RStudio interface (Code, Workspace, Plots, Console) and various R packages: markdown, Shiny, tidyverse, knitr, and ggplot2.

RStudio
<https://www.rstudio.com/>

データやスクリプトの置かれているディレクトリの確認



デスクトップ上の「NGSworkshop/Data/RNA-Seq/useref」ディレクトリ
中にある、以下の3つのデータを用いて解析を行なう。



1. 「ballgown」ディレクトリにある8サンプル分（2条件、4反復）のStringTieの結果
2. サンプル名や条件を記載した、カンマ区切りのテキストファイル（phenodata.csv）
3. 実行するコマンドを記載したRのスクリプト（useref_analysis_workflow.R）

デフォルトでは4つの区画（ペイン）に分かれている

The screenshot shows the RStudio interface with several panels:

- Environment** pane: Shows the Global Environment with objects `bg`, `bg_filtered`, and `pheno_data`. `bg` is a large ballgown object (126.2 Mb), `bg_filtered` is a large ballgown object (74.7 Mb), and `pheno_data` is an 8 obs. of 3 variables data frame.
- Files** pane: Shows the file structure under `~/Desktop/workshop/RNA-Seq/useref`. The contents are:

Name	Size	Modified
..		Sep 16, 2018, 9:49 PM
.Rhistory	18 KB	Sep 16, 2018, 9:49 PM
alignment		
ballgown		
phenodata.csv	165 B	Sep 10, 2018, 8:00 AM
useref_analysis_workflow.R	3.7 KB	Sep 16, 2018, 9:48 PM
- Console** pane: Displays R session output. It includes a command to read `phenodata.csv`, set up a `ballgown` instance, and filter the data to remove transcripts with low variance across samples. It also shows the results of a statistical test comparing Day/Night samples at the transcript level.
- R Script** pane: Shows the R script code for the analysis.

Three boxes highlight specific features:

- A box around the **Environment** pane title bar contains the text: **• 変数一覧** (Variable List) and **• 作業履歴** (Job History).
- A box around the **Files** pane title bar contains the text: **• ディレクトリ構造** (Directory Structure), **• プロット** (Plots), and **• ヘルプ** (Help).
- A box around the **Console** pane title bar contains the text: **Rスクリプトの編集** (Edit R Script) and **Rコンソール** (R Console).

作業ディレクトリの指定

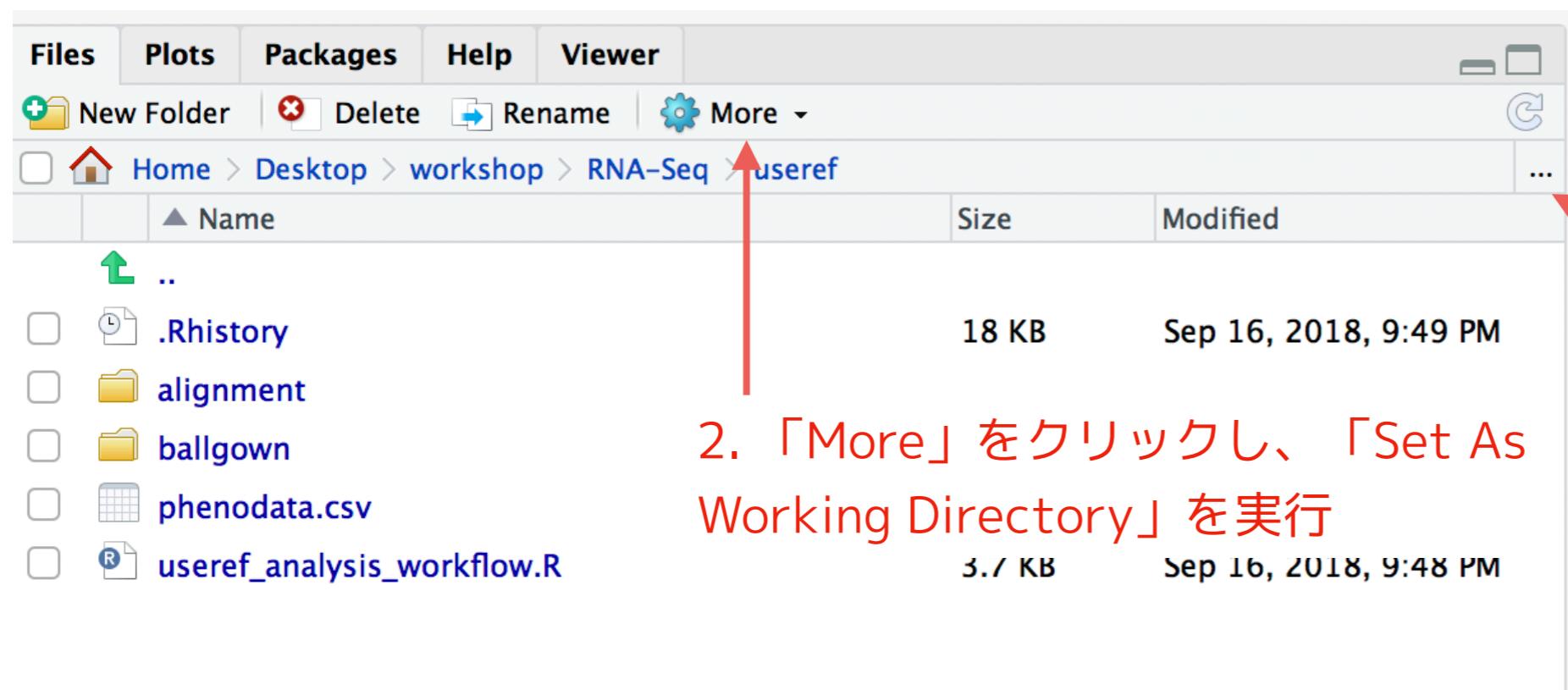
1. Rコンソール（左下のペイン）でコマンドを実行

```
> setwd("c:/Users/user/Desktop/NGSworkshop/Data/RNA-Seq/")
```

ballgownディレクトリがある場所を指定する。

もしくは、

2. ディレクトリ構造が表示されている右下のペインを操作



The screenshot shows the RStudio file browser interface. The top navigation bar includes Files, Plots, Packages, Help, and Viewer. Below the toolbar are buttons for New Folder, Delete, Rename, and More. The main area displays a file tree with the path Home > Desktop > workshop > RNA-Seq > useref. The 'More' button has a red arrow pointing to it. The 'useref' folder is selected. The bottom part of the browser shows a list of files and folders:

Name	Size	Modified
..		
.Rhistory	18 KB	Sep 16, 2018, 9:49 PM
alignment		
ballgown		
phenodata.csv		
useref_analysis_workflow.R	3.7 KB	Sep 16, 2018, 9:48 PM

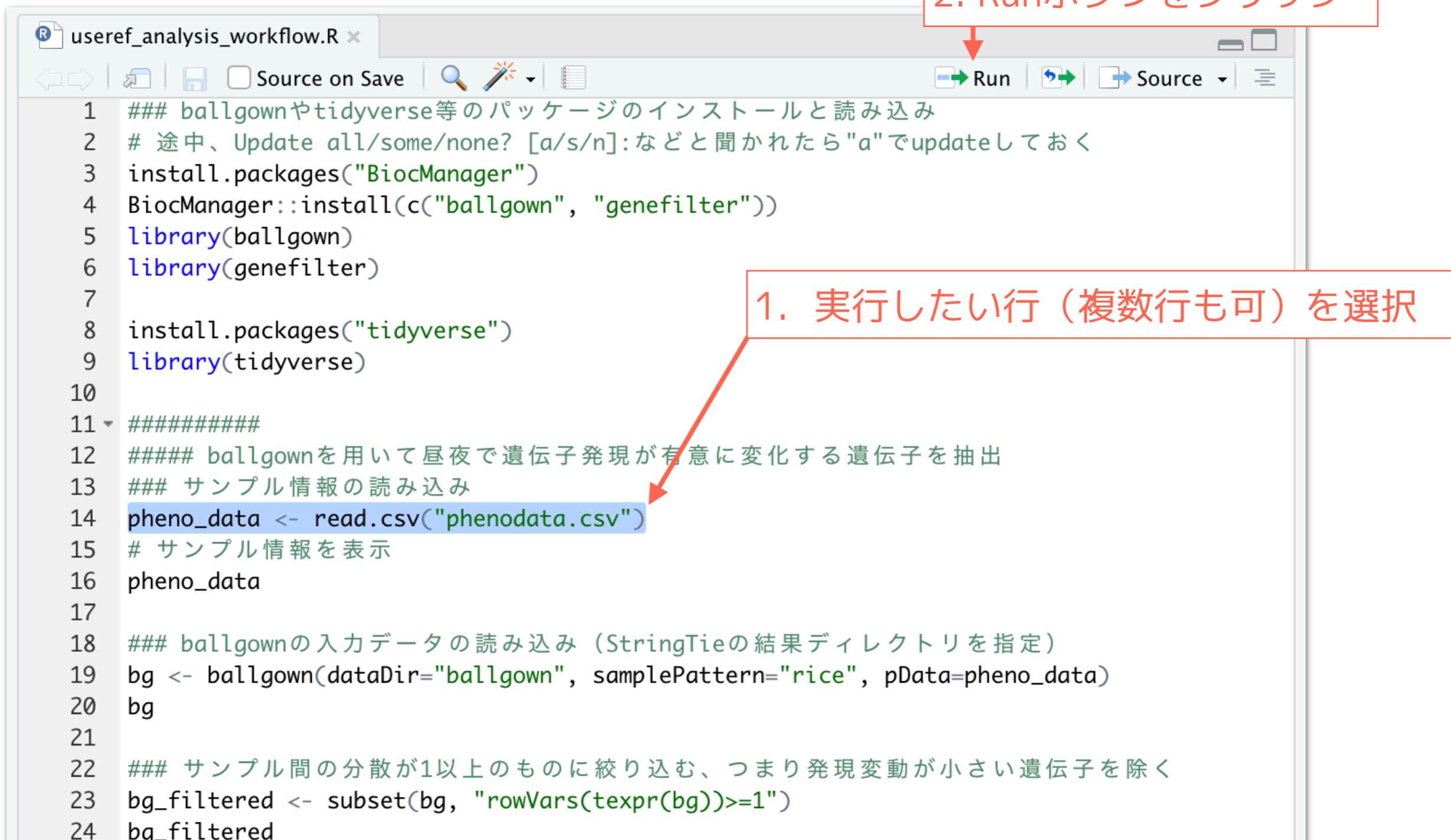
Red annotations provide instructions:

2. 「More」をクリックし、「Set As Working Directory」を実行
1. 「…」をクリックし、「RNA-Seq」ディレクトリを選択

Rスクリプト中のコマンドを順次実行していく

1. 実行したい行（複数行も可）を選択

2. Runボタンをクリック



```

R useref_analysis_workflow.R
Source on Save | Run | Source
1 ### ballgownやtidyverse等のパッケージのインストールと読み込み
2 # 途中、Update all/some/none? [a/s/n]:などと聞かれたら "a" でupdateしておく
3 install.packages("BiocManager")
4 BiocManager::install(c("ballgown", "genefilter"))
5 library(ballgown)
6 library(genefilter)
7
8 install.packages("tidyverse")
9 library(tidyverse)
10
11 #####
12 ##### ballgownを用いて昼夜で遺伝子発現が有意に変化する遺伝子を抽出
13 ### サンプル情報の読み込み
14 pheno_data <- read.csv("phenodata.csv") ← This line is selected
15 # サンプル情報を表示
16 pheno_data
17
18 ### ballgownの入力データの読み込み (StringTieの結果ディレクトリを指定)
19 bg <- ballgown(dataDir="ballgown", samplePattern="rice", pData=pheno_data)
20 bg
21
22 ### サンプル間の分散が1以上のものに絞り込む、つまり発現変動が小さい遺伝子を除く
23 bg_filtered <- subset(bg, "rowVars(expr(bg))>=1")
24 bg_filtered

```

シェルスクリプトと同様、どのような解析をおこなったかをあとで見直す際や、データやパラメータを変えて同じ解析をする際にRスクリプトとして保存しておくと便利。

パッケージのインストールと読み込み

ballgownを用いた統計解析に必要なパッケージのインストールと読み込み

```
install.packages("BiocManager")
BiocManager::install(c("ballgown", "genefilter"))
library(ballgown) ★
library(genefilter) ★
```

- ・ ballgownとgenefilterはR/Bioconductorのパッケージとして提供されており、BiocManagerのinstall関数でパッケージのインストールする。
- ・ 途中、「Update all/some/none? [a/s/n]: 」と聞かれたら、「a」と入力しリターンを押し、全てアップデートする。多少エラーが出ても問題ないこともある。

```
> install.packages(tidyverse)
> library(tidyverse) ★
```

Rのパッケージであるtidyverseは、install.packages関数を用いてインストールする。

*演習ではパッケージのインストールは完了しているため、★印のついたlibrary関数によるパッケージの読み込みだけを行えばOKです。

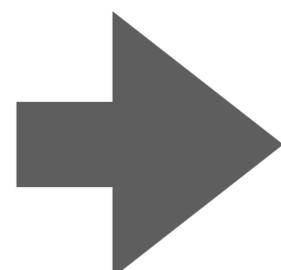
サンプル情報 (phenodata.csv)

サンプル情報の読み込み

```
> pheno_data <- read.csv("phenodata.csv")
```

カンマ区切りの書式で、サンプル名、
実験条件などを記載したファイル
(phenodata.csv) を準備する。

ids,cond,rep	← 1行目は項目名
rice_D_rep1,Day,1	
rice_D_rep2,Day,2	
rice_D_rep3,Day,3	
rice_D_rep4,Day,4	
rice_N_rep1,Night,1	
rice_N_rep2,Night,2	
rice_N_rep3,Night,3	
rice_N_rep4,Night,4	



上のコマンドを実行すると
データフレームとして読み込まれる。

> pheno_data	ids	cond	rep
1 rice_D_rep1	Day	1	
2 rice_D_rep2	Day	2	
3 rice_D_rep3	Day	3	
4 rice_D_rep4	Day	4	
5 rice_N_rep1	Night	1	
6 rice_N_rep2	Night	2	
7 rice_N_rep3	Night	3	
8 rice_N_rep4	Night	4	

StringTieデータをballgownオブジェクトに格納する



StringTieデータを読み込み、ballgownオブジェクトに格納することで、様々な解析が可能になる。

```
> bg <- ballgown(dataDir="ballgown", samplePattern="rice", pData=pheno_data)
Mon Sep 10 08:00:18 2018
Mon Sep 10 08:00:18 2018: Reading linking tables
Mon Sep 10 08:00:18 2018: Reading intron data files
Mon Sep 10 08:00:20 2018: Merging intron data
Mon Sep 10 08:00:20 2018: Reading exon data files
Mon Sep 10 08:00:24 2018: Merging exon data
Mon Sep 10 08:00:25 2018: Reading transcript data files
Mon Sep 10 08:00:26 2018: Merging transcript data
Wrapping up the results
Mon Sep 10 08:00:26 2018
```

ballgownオブジェクトを指定してみると

```
> bg
ballgown instance with 44,586 transcripts and 8 samples
```

44,586 transcriptsについて、8サンプル分の遺伝子発現情報が読み込まれていることが分かる。

昼夜で遺伝子発現量が変化した遺伝子（DEG）の検出



ballgownのsubset関数を使い、サンプル間のFPKMの分散が1以上の転写産物に絞り込む、つまり発現変動が小さいものを除く

```
> bg_filtered <- subset(bg, "rowVars(expr(bg))>=1")
> bg_filtered
ballgown instance with 19991 transcripts and 8 samples
```

転写産物数が19,991 transcriptsとなり、絞り込まれている。

転写産物レベルでサンプル（Day/Night）間での遺伝子発現変動を検定し、p-valueやq-valueを計算する

```
> results_transcripts <- stattest(bg_filtered, feature="transcript",
covariate="cond", getFC=TRUE, meas="FPKM")
```

- covariateで比較対象の列名「cond」、measで利用する発現量の指標「FPKM」、getFCで結果にFold-changeを出力するよう指定している。
- 他にもタイムコースサンプルの場合に指定するものなど複数のオプションがある。

昼夜で遺伝子発現量が変化した遺伝子（DEG）の検出



検定の結果の確認。各転写産物ごとにDay/Night間のFold-changeやp-value、q-valueが計算されている。

```
> head(results_transcripts)
  feature id      fc      pval      qval
1 transcript 1 0.9180662 0.5169128022 0.68235630
2 transcript 6 0.8906488 0.2519004300 0.44433240
3 transcript 7 1.0106792 0.9216822885 0.95930393
4 transcript 9 0.4228804 0.1215415562 0.29074712
5 transcript 10 0.9335729 0.4569417800 0.63598991
6 transcript 11 1.4025491 0.0003284612 0.02168817
```

- ・このままでは遺伝子IDや遺伝子名などもなく分かりにくい。
- ・IDや遺伝子名の追加、q-valueによるソートなどを行うとよい。

昼夜で遺伝子発現量が変化した遺伝子（DEG）の検出



遺伝子名や遺伝子ID、転写産物IDの列を追加する。

```
> results_transcripts = data.frame(geneNames=ballgown::geneNames(bg_filtered),  
geneIDs=ballgown::geneIDs(bg_filtered),  
transcriptIDs=ballgown::transcriptNames(bg_filtered), results_transcripts)
```

検定結果をp-valueの小さい順にソートする

```
> results_transcripts = arrange(results_transcripts,pval)
```

適当なq-valueの閾値(qval < 0.01)で切った遺伝子を抽出する（75転写産物がヒット）。

```
> subset(results_transcripts, results_transcripts$qval<0.01)
```

	geneNames	geneIDs	transcriptIDs	feature	id	fc	pval	qval
1	-	Os02g0623932	Os02t0623932-00	transcript	9347	1.334693e-01	3.016259e-07	0.003330334
2	LHY	Os04g0583900	Os04t0583900-01	transcript	19641	2.584726e+02	3.331834e-07	0.003330334
3	-	Os06g0660800	Os06t0660800-01	transcript	27355	6.321374e+00	6.792785e-07	0.003964490
4	OsBBX19, OsM, DTH2, qDTH-2	Os06g0298200	Os06t0298200-01	transcript	25939	6.919100e+01	9.295324e-07	0.003964490
5	PsbP	Os03g0279950	Os03t0279950-01	transcript	13109	2.380650e+01	9.915688e-07	0.003964490
6	-	Os08g0360100	Os08t0360100-01	transcript	32883	1.538616e-01	1.254840e-06	0.004073630
7	-	Os06g0281400	Os06t0281400-01	transcript	25839	3.943439e-01	1.426413e-06	0.004073630
...								
38	SIGA	Os08g0163400	Os08t0163400-01	transcript	31942	1.358213e-01	1.574039e-05	0.008169480
...								

遺伝子発現量の分布の可視化



全サンプルの平均発現量が1以上の転写産物を選び、遺伝子発現量（FPKM値に0.01を足してlog2をとったもの）を取得する（22,292個の転写産物）。

```
> bg.expressed <- subset(bg, "rowMeans(expr(bg))>=1")
> bg.expressed
ballgown instance with 22292 transcripts and 8 samples
> log2fpkm <- log2(expr(bg.expressed, meas="FPKM") + 0.01)
```

変数log2fpkmには、転写産物・サンプルごとに全FPKM値が格納されている。

```
> head(log2fpkm)
      FPKM.rice_D_rep1 FPKM.rice_D_rep2 FPKM.rice_D_rep3 FPKM.rice_D_rep4 FPKM.rice_N_rep1
1       3.539679        3.831836        4.0835609       4.1796822       3.897224
4       1.147461        1.055207        0.5679202      -0.4774179       1.388669
6       3.677559        3.757414        4.0814173       3.9224231       3.593344
7       3.333910        4.173667        4.0908662       4.1635102       3.890806
9       5.925956        6.953405        6.8972279       4.9053619       5.080402
10      3.458069        4.015651        4.1832976      4.0796425       3.893308
      FPKM.rice_N_rep2 FPKM.rice_N_rep3 FPKM.rice_N_rep4
1       4.2026522       4.325626       3.7204283
4       0.1995184       1.501587       0.6323222
6       3.8188390       4.363668       4.1252938
7       4.4518494       4.566686       4.6076217
9       5.5129445       5.091533       4.7157670
10      4.1539759       4.123816       4.2123396
```

遺伝子発現量の分布の可視化

データを整形する。

```
> log2fpkm.long <- as.data.frame(log2fpkm) %>%  
  tidyr::gather(Dataset, FPKM)
```

ワイドからロングフォーマットのデータフレームに変換されている。

```
> head(log2fpkm.long)  
  Dataset      FPKM  
1 FPKM.rice_D_rep1 3.539679  
2 FPKM.rice_D_rep1 1.147461  
3 FPKM.rice_D_rep1 3.677559  
4 FPKM.rice_D_rep1 3.333910  
5 FPKM.rice_D_rep1 5.925956  
6 FPKM.rice_D_rep1 3.458069
```

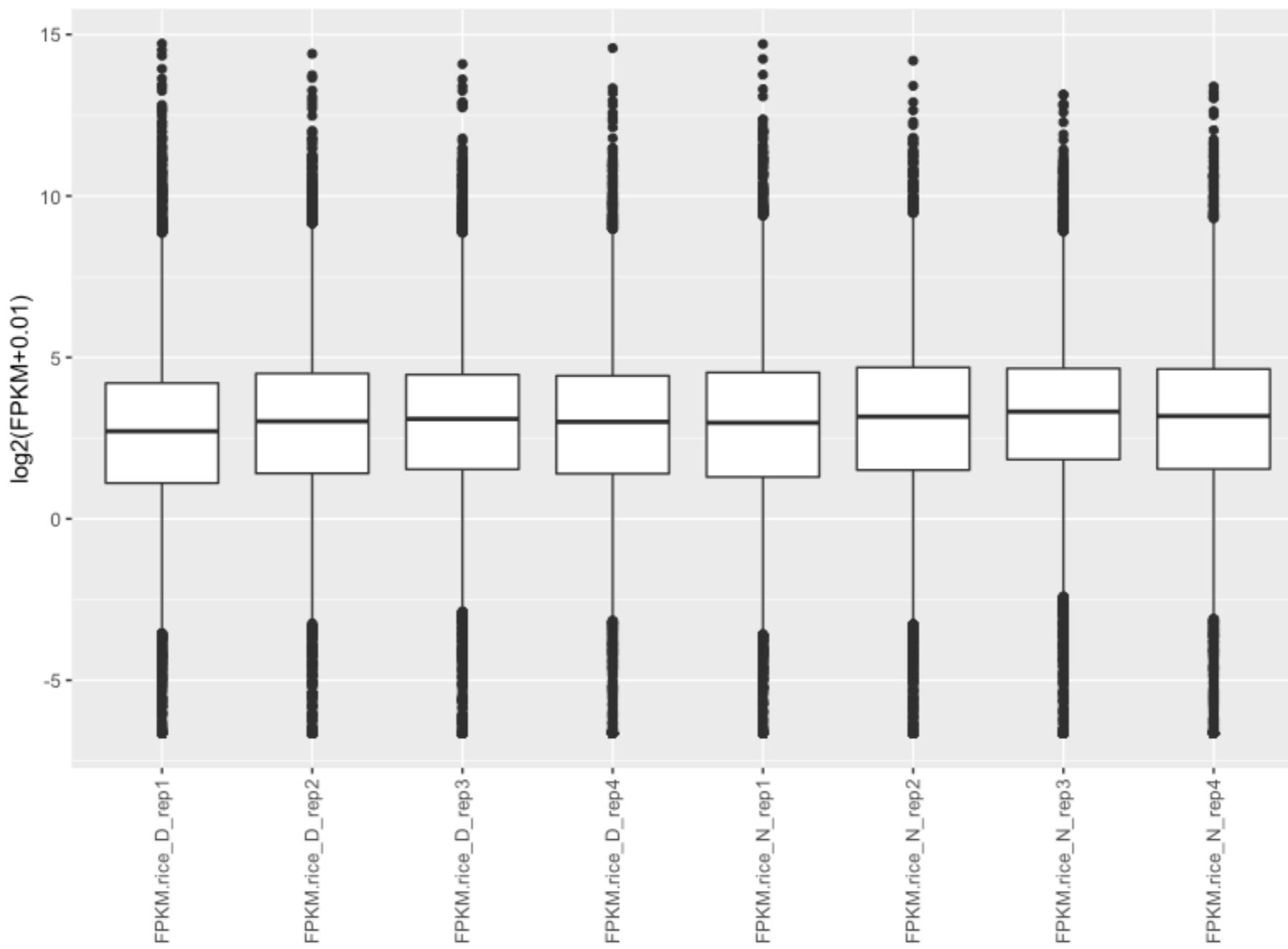
このような書式にしておくと、グラフ描画関数 ggplot でのグラフ化が容易になる。

データ解析の大部分はこのようなデータのフィルタリングや整形などの地味な仕事。

遺伝子発現量の分布の可視化

箱ひげ図を描く。

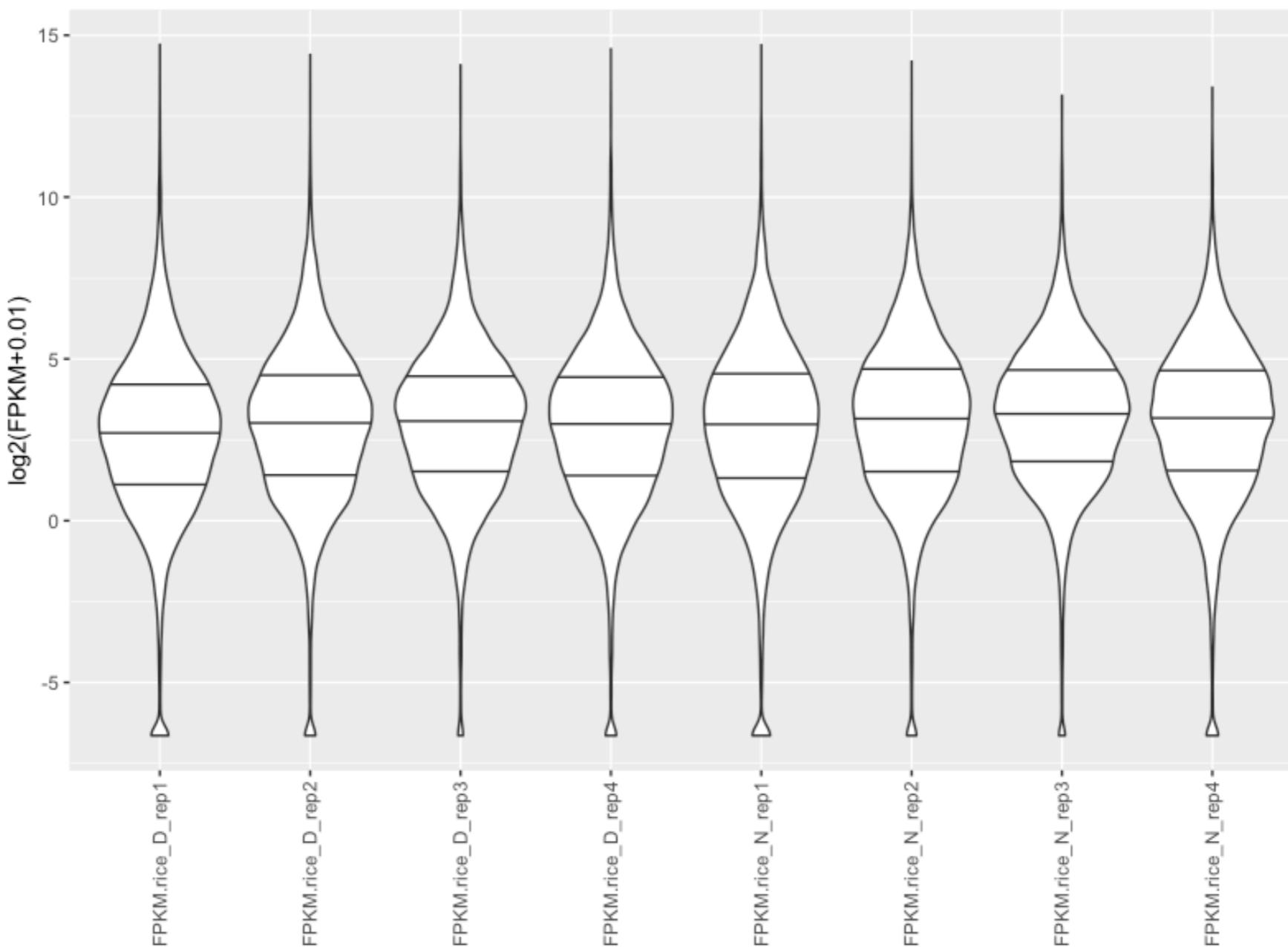
```
> ggplot(log2fpkm.long, aes(y=FPKM, x=Dataset)) + geom_boxplot() +  
  theme(axis.text.x = element_text(angle=90, hjust=0, vjust=.5)) +  
  xlab("") + ylab("log2(FPKM+0.01)")
```



遺伝子発現量の分布の可視化

同じデータを使って、ヴァイオリンプロットを描いてみる。

```
> ggplot(log2fpkm.long, aes(y=FPKM, x=Dataset)) +  
  geom_violin(draw_quantiles = c(0.25, 0.5, 0.75)) +  
  theme(axis.text.x = element_text(angle=90, hjust=0, vjust=.5)) +  
  xlab("") + ylab("log2(FPKM+0.01)")
```



特定の遺伝子の構造や発現プロファイルを可視化する



ballgownオブジェクトに入れる際に振られた"id"で遺伝子を指定する必要があるため、遺伝子のIDや遺伝子名と「ballgownが割り振るID」との対応を調べる。

```
> subset(results_transcripts, results_transcripts$qval<0.01)
```

	geneNames	geneIDs	transcriptIDs	feature	id	fc	pval	qval
1	-	Os02g0623932	Os02t0623932-00	transcript	9347	1.334693e-01	3.016259e-07	0.003330334
2	LHY	Os04g0583900	Os04t0583900-01	transcript	19641	2.584726e+02	3.331834e-07	0.003330334
3	-	Os06g0660800	Os06t0660800-01	transcript	27355	6.321374e+00	6.792785e-07	0.003964490
4	OsBBX19, OsM, DTH2, qDTH-2	Os06g0298200	Os06t0298200-01	transcript	25939	6.919100e+01	9.295324e-07	0.003964490
5	PsbP	Os03g0279950	Os03t0279950-01	transcript	13109	2.380650e+01	9.915688e-07	0.003964490
6	-	Os08g0360100	Os08t0360100-01	transcript	32883	1.538616e-01	1.254840e-06	0.004073630
7	-	Os06g0281400	Os06t0281400-01	transcript	25839	3.943439e-01	1.426413e-06	0.004073630
...								
38	SIGA	Os08g0163400	Os08t0163400-01	transcript	31942	1.358213e-01	1.574039e-05	0.008169480
...								

- ballgown id: 19641 = LHY = Os04t0583900-01
- Ballgown id: 31942 = SIGA = Os08t0163400-01

であることが分かる。

特定の遺伝子の構造や発現プロファイルを可視化する



LHY遺伝子（ballgown id: 19641）の発現量の箱ひげ図を作成し、個々のサンプルの発現量を個別に重ねてプロットする。

まずはballgown idを指定し、特定の転写産物の発現量データを取り出す。

```
> log2fpkm["19641", ]  
FPKM.rice_D_rep1 FPKM.rice_D_rep2 FPKM.rice_D_rep3 FPKM.rice_D_rep4  
-2.0672131      -0.7016209      -3.1568201      -4.5881052  
FPKM.rice_N_rep1 FPKM.rice_N_rep2 FPKM.rice_N_rep3 FPKM.rice_N_rep4  
8.2926856       8.4396024       7.9646139       8.1115949
```

特定の遺伝子の構造や発現プロファイルを可視化する

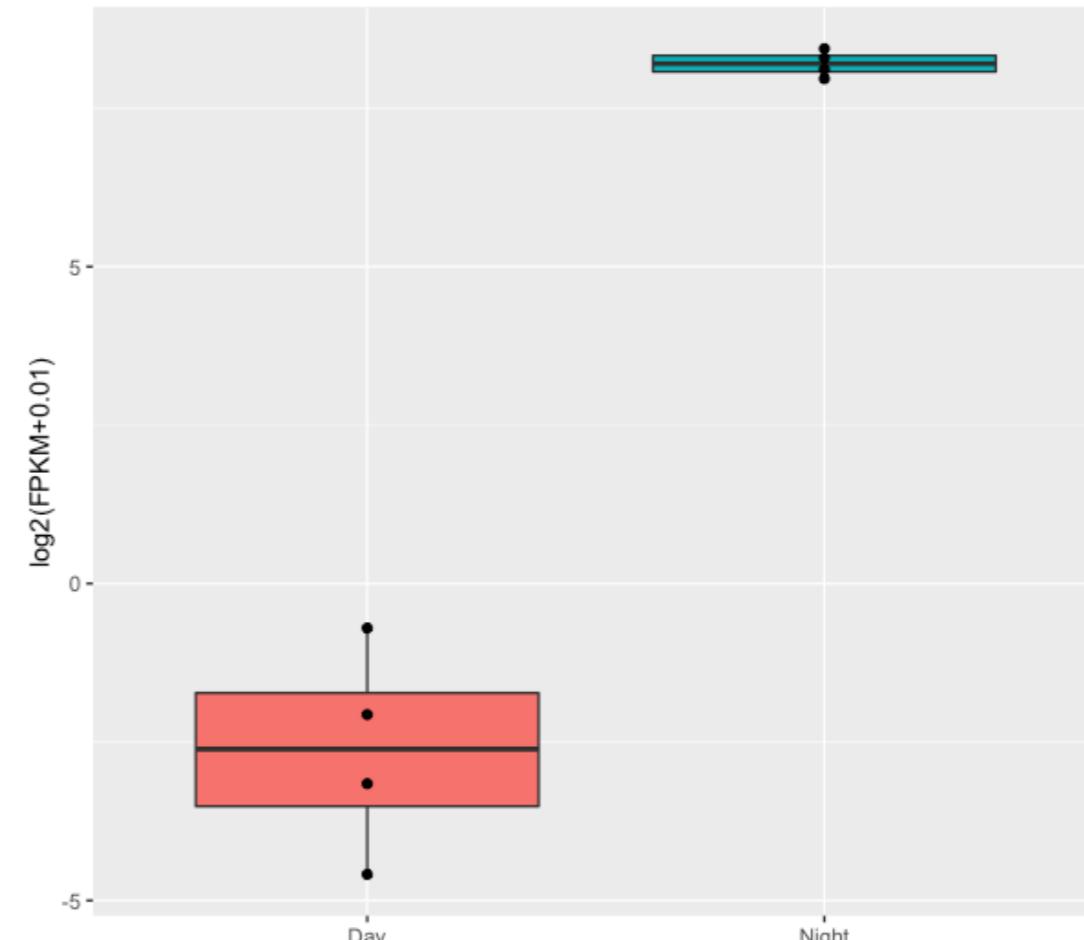
取り出した発現量データを整形し、ggplotで箱ひげ図と散布図を描画。

```
> log2fpkm.LHY.long <- as.data.frame(log2fpkm["19641", ]) %>%  
  tidyr::gather(Dataset, FPKM)  
> log2fpkm.LHY.long <- data.frame(log2fpkm.LHY.long, Condition=c(rep("Day", 4),  
rep("Night", 4)))  
> ggplot(log2fpkm.LHY.long, aes(x=Condition, y=FPKM, fill=Condition)) +  
  geom_boxplot() + geom_point() +  
  xlab("") + ylab("log2(FPKM+0.01)") + theme(legend.position = "bottom")
```

描画用のデータの確認

```
> log2fpkm.LHY.long
```

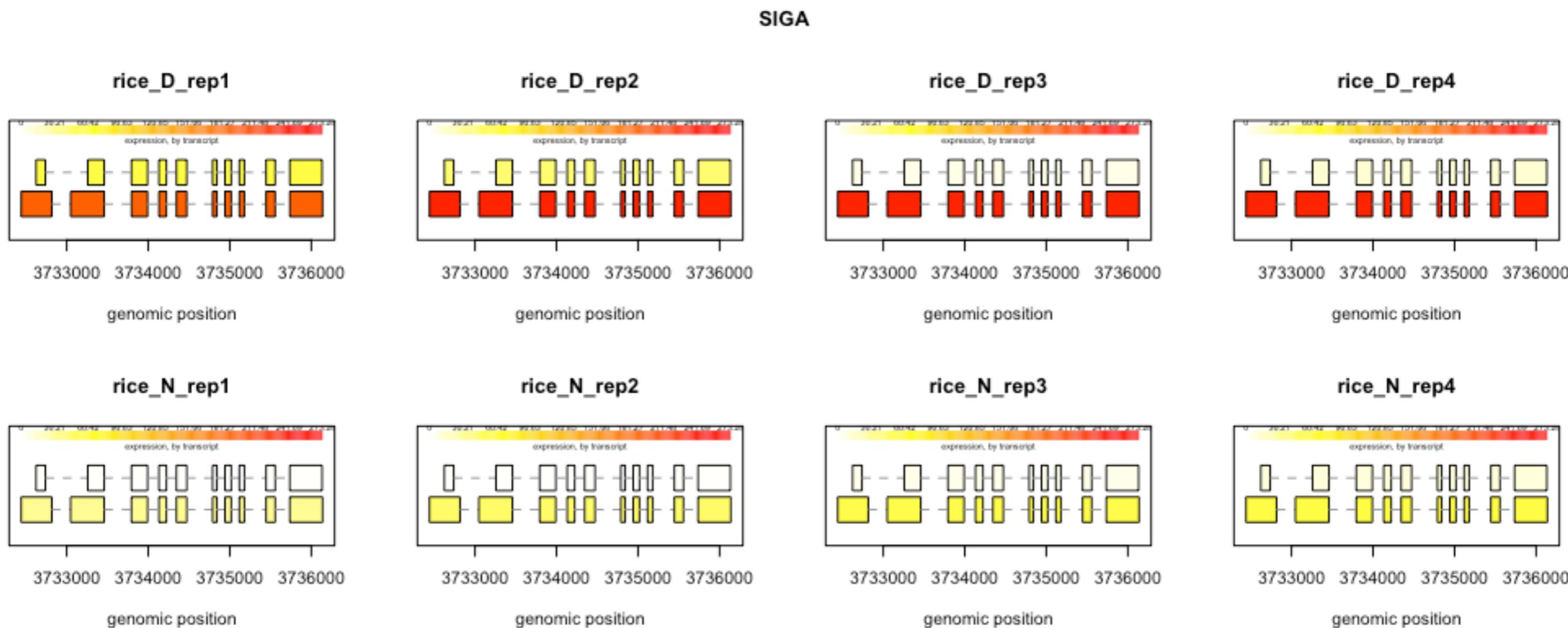
	Dataset	FPKM	Condition
1	log2fpkm["19641",]	-2.0672131	Day
2	log2fpkm["19641",]	-0.7016209	Day
3	log2fpkm["19641",]	-3.1568201	Day
4	log2fpkm["19641",]	-4.5881052	Day
5	log2fpkm["19641",]	8.2926856	Night
6	log2fpkm["19641",]	8.4396024	Night
7	log2fpkm["19641",]	7.9646139	Night
8	log2fpkm["19641",]	8.1115949	Night



特定の遺伝子の構造や発現プロファイルを可視化する

sigA遺伝子 (ballgown id: 31942) の転写産物構造と発現量を合わせて可視化する。

```
> plotTranscripts(ballgown::geneIDs(bg)[31942], bg, main=c('SIGA'),  
sample=c('rice_D_rep1','rice_D_rep2','rice_D_rep3','rice_D_rep4','rice_N_rep1',  
'rice_N_rep2','rice_N_rep3','rice_N_rep4'))
```



転写調節に関するイネのsigA遺伝子の発現量は昼に高く、夜に低いパターンを示し、2つあるsplicing isoformうちの一方が主に発現していることが分かる。

おまけ：Kallistoによるアライメントフリーの発現解析



Kallistoを使って発現量を定量するシェルスクリプト

```
$ less ./run_kallisto.sh
```

- run_kallisto.sh -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
Kallisto_bin=$ToolDir/kallisto  
  
### Step1. Make transcriptome index  
$Kallisto_bin/kallisto index -i transcript_index $DataDir/transcript.fasta  
  
### Step2. Quantification  
$Kallisto_bin/kallisto quant -i transcript_index -o output ../useref/  
rice_D_rep1_r1.pe.fastq.gz ../useref/rice_D_rep1_r2.pe.fastq.gz
```

T-DBGの構築

RNA-Seqリードの対応付けと発現量推定

ここでは1サンプル (rice_D_rep1) のみ発現解析を行っている。

シェルスクリプトを実行

```
$ bash ./run_kallisto.sh
```

(実行時間：約1秒)

おまけ：Kallistoによるアラインメントフリーの発現解析



解析が終わるとoutputディレクトリが作成され、
その中に発現量データ等の結果が出力される

```
$ less output/abundance.tsv
```

- abundance.tsv -

target_id	length	eff_length	est_counts	tpm
Os02t0722500-01	1436	1275.73	43	200.913
Os02t0722600-01	487	327.326	0	0
Os02t0722650-00	531	371.12	13	208.797
Os02t0722700-01	1436	1275.73	900	4205.15
Os02t0722800-01	4485	4324.73	161.034	221.951
Os02t0722800-02	1316	1155.73	187.966	969.438
Os02t0723200-01	1526	1365.73	3	13.0935
Os02t0723300-01	890	729.8	211	1723.36
Os02t0723400-01	716	555.868	1	10.7232
Os02t0723400-02	1003	842.727	7	49.5117
Os02t0723600-00	903	742.727	0	0

- 各転写産物のTPM値が出力されている。
- サンプル間で発現量が異なる遺伝子（DEG）の検出用に、専用のRパッケージ「sleuth」が開発されている。

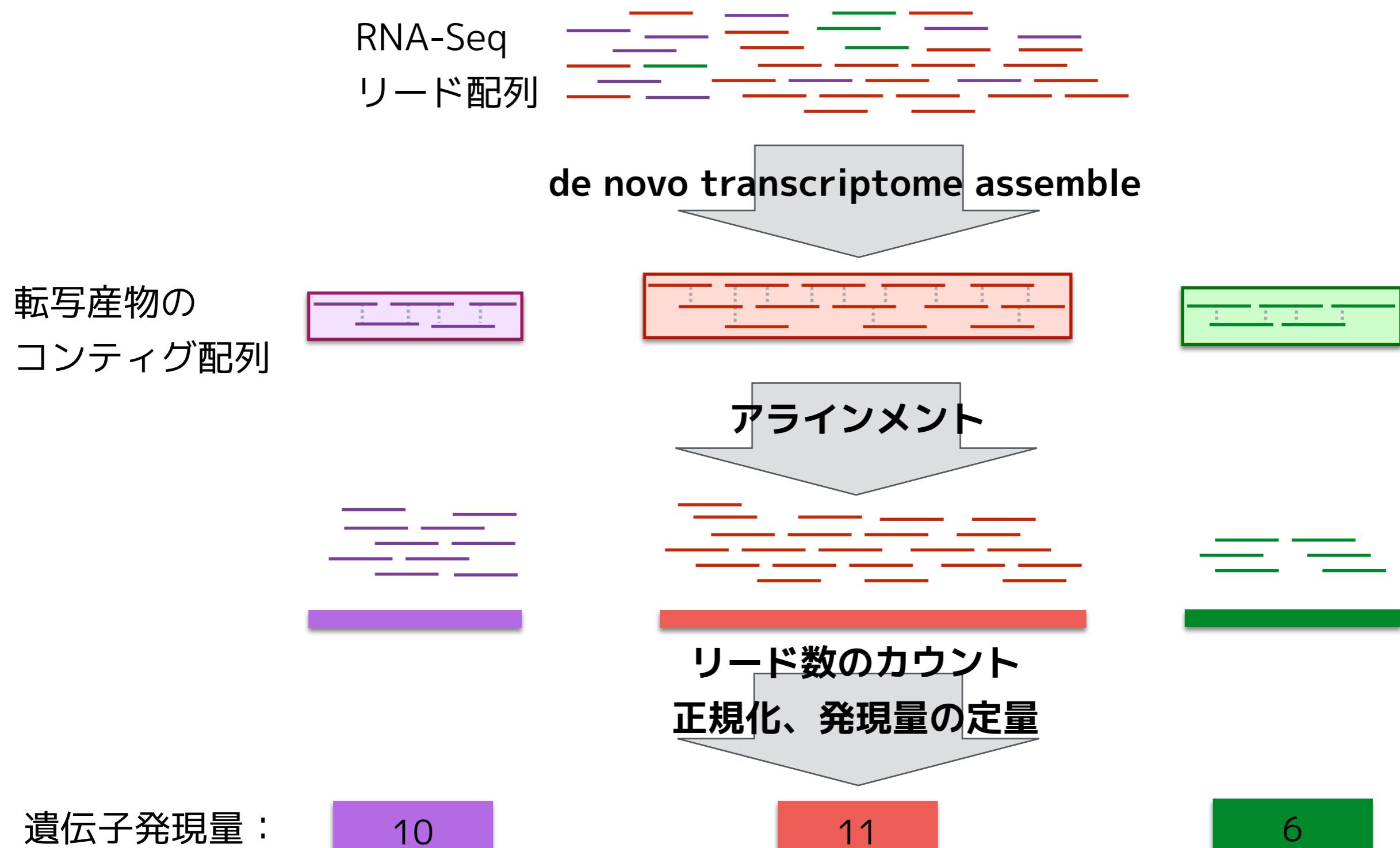
- ▶ リファレンス情報を用いたRNA-Seq解析
- ▶ **De novo**トランスクリプトーム解析
- ▶ RNA-Seqデータを用いた多型検出

de novo transcriptome assemble法を用いた遺伝子発現解析の流れ



まず始めに、RNA-Seqリード配列から転写産物の全長配列を再構築する。

アセンブルされた転写産物配列上にRNA-Seqリードをアラインメントし、転写産物ごとにリード数をカウントすることにより遺伝子発現を定量する。



De novo transcriptome assemble

良い点

- ・ゲノム配列を解読することなく、効率よく遺伝子配列を得ることができる上に発現情報も一緒に得られる。

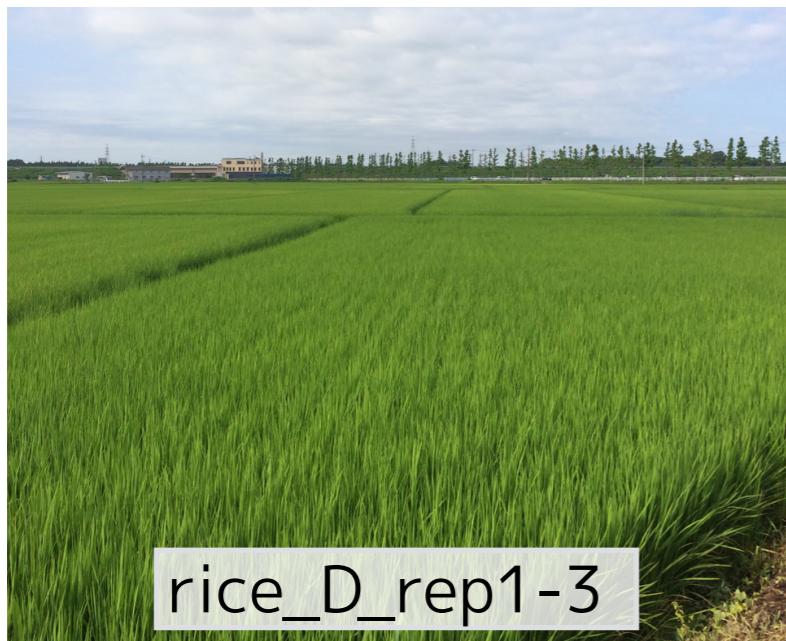
悪い点

- ・リファレンスゲノムベースでの発現解析に比べ、信頼度が劣る（キメラや断片化した転写産物が多い）。
- ・重複遺伝子だけではなく、選択的スプライシングによるアイソフォームも区別する必要があるうえに、発現量によって遺伝子間のリードの厚みにバラつきがあり、アセンブルが困難。

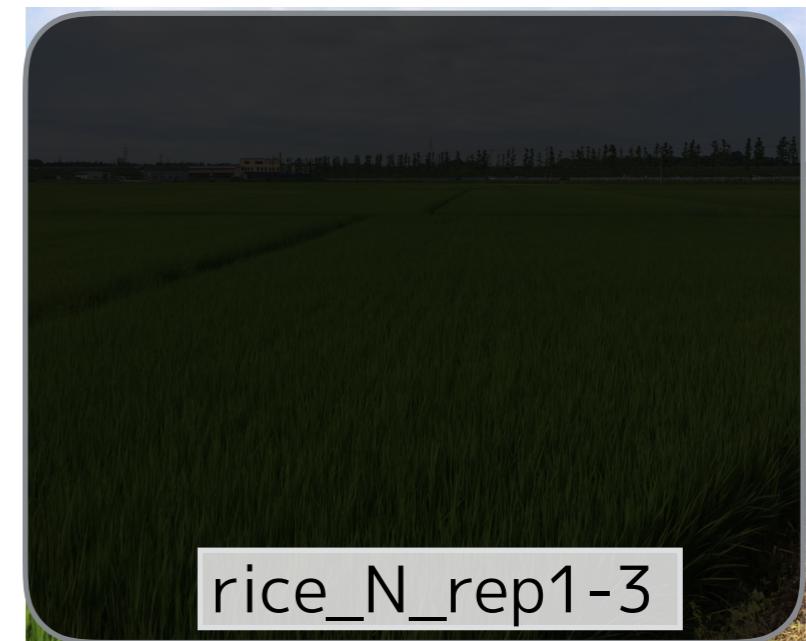
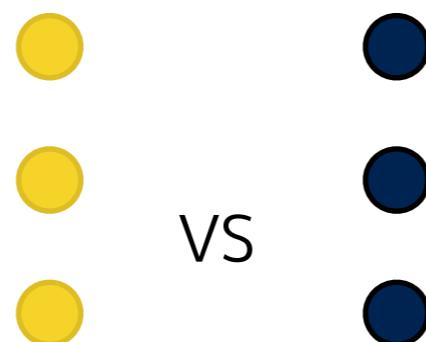
それでもゲノム配列などのない非モデル生物においては、トランскルiptオーム解析の主流になっている。

目的と使用するデータ

目的：de novo transcriptome assemble法を用いて、
昼と夜でのイネの葉の遺伝子発現の違いを調べる



12:00 PM



00:00 AM

- 田んぼで栽培する、イネ（コシヒカリ）の葉身のサンプル。
- 3つの異なる生育ステージ（**3反復**）において、昼（12時）と夜（0時）にサンプリング（**2条件**）。
- Illumina社の**Stranded mRNA-Seq法**でライブラリ調製。
- Illumina社のHiSeq2000による、**101bpのPaired-endシーケンシング**。

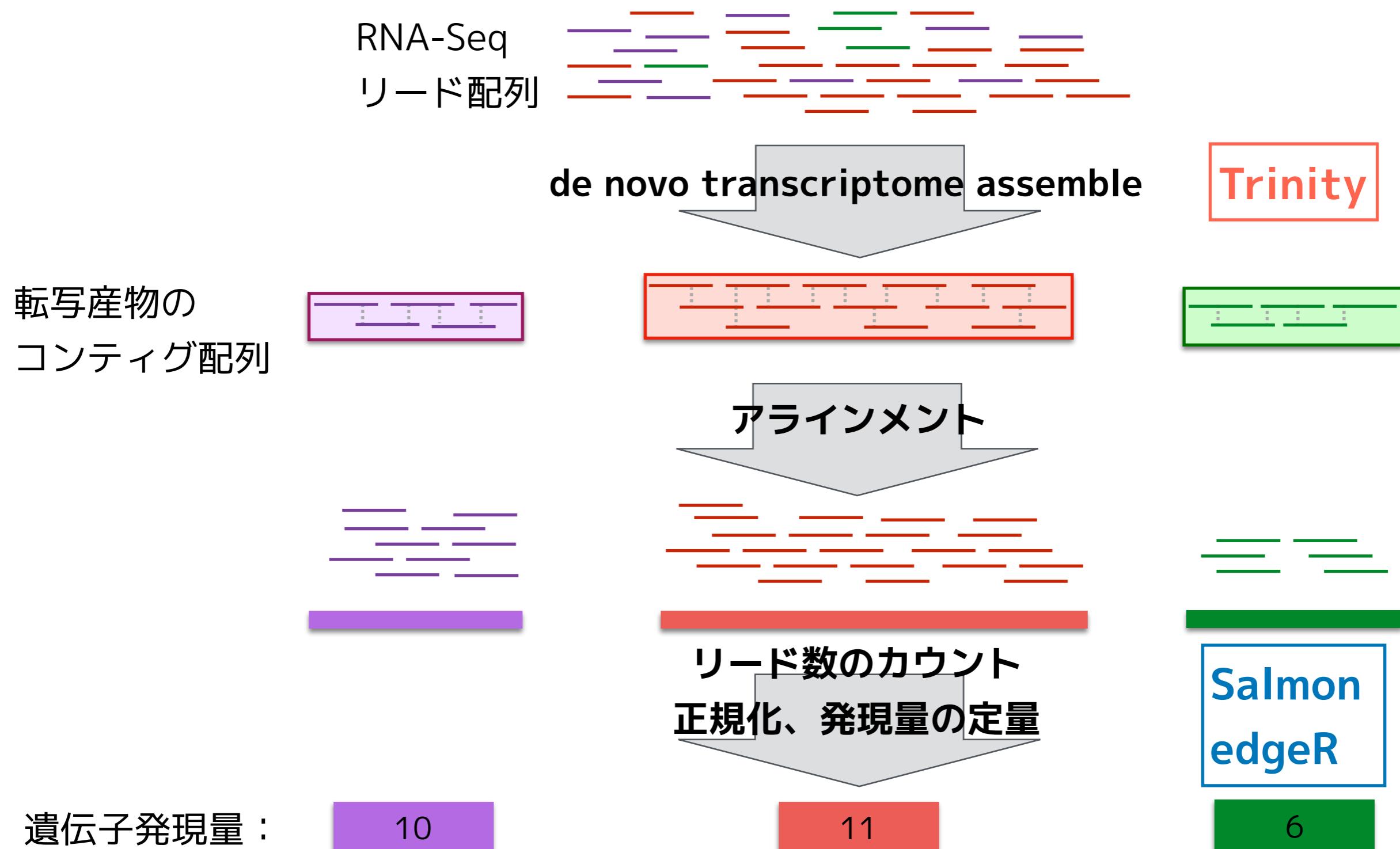
*全データでは解析に時間がかかるため、2番染色体の特定の領域（2Mbp）にアラインメントされるリードだけを事前に抽出しており、演習ではそれを利用する。

de novo transcriptome assemble法を用いた遺伝子発現解析の流れ



まず始めに、RNA-Seqリード配列から転写産物の全長配列を再構築する。

アセンブルされた転写産物配列上にRNA-Seqリードをアラインメントし、転写産物ごとにリード数をカウントすることにより遺伝子発現を定量する。



遺伝子発現量：

10

11

6

Step1. リード配列の準備

De novoトランск립トーム解析用ディレクトリ「denovo」に移動

```
$ cd ~/RNA-Seq/denovo
```

解析用ディレクトリ内のファイルを確認

```
$ ls
sample_info.txt      step2_Trinity.sh  step4_edgeR.sh
step1_prep_fastq.sh  step3_Salmon.sh   step5_eval_assembly.sh
```

シェルスクリプトが5つ用意されており、順番にシェルスクリプトを実行することで解析が進められるようになっている。

```
$ bash ./step1_prep_fastq.sh
$ bash ./step2_Trinity.sh
$ bash ./step3_Salmon.sh
$ bash ./step4_edgeR.sh
$ bash ./step5_eval_assembly.sh
```

Step1. リード配列の準備

Trinityに入力する配列データ（FASTQ）のヘッダーは、リード1の末尾が"/1"、リード2の末尾が"/2"にする必要がある。

ヘッダー末尾に"/1"や"/2"を加えるためのシェルスクリプト

```
$ less step1_prep_fastq.sh
```

```
- step1_prep_fastq.sh -
```

3つの処理から成り、真ん中のAWKスクリプトはオリジナルのFASTQを読み込み、ヘッダー行（行数を4で割って1余る行）の後に"/1"や"/2"を付け足している。

```
DataDir=$HOME/RNA-Seq_whole_genome/data
```

```
# Step1. Add suffix (/1 or /2) to original FASTQ header for Trinity
for DATASET in rice_D_rep1 rice_D_rep2 rice_D_rep3 rice_N_rep1 rice_N_rep2 rice_N_rep3
do
    echo "converting ${DATASET}_r1 ..."
    zcat $DataDir/${DATASET}_r1.org.fastq.gz | awk '{ if (NR%4==1) { print $1"/1" } else
{ print } }' | gzip -c > ${DATASET}_r1.trinity.fastq.gz
    echo "converting ${DATASET}_r2 ..."
    zcat $DataDir/${DATASET}_r2.org.fastq.gz | awk '{ if (NR%4==1) { print $1"/2" } else
{ print } }' | gzip -c > ${DATASET}_r2.trinity.fastq.gz
done
```

Step1. リード配列の準備

シェルスクリプトの実行 (実行時間: 約1分)

```
$ bash ./step1_prep_fastq.sh
```

作成されたリード配列の確認

```
$ ls -lh *.fastq.gz
-rw-rw-r-- 1 guest01 guest01 11M 9月 17 12:27 rice_D_rep1_r1.trinity.fastq.gz
-rw-rw-r-- 1 guest01 guest01 11M 9月 17 12:27 rice_D_rep1_r2.trinity.fastq.gz
-rw-rw-r-- 1 guest01 guest01 11M 9月 17 12:27 rice_D_rep2_r1.trinity.fastq.gz
-rw-rw-r-- 1 guest01 guest01 11M 9月 17 12:27 rice_D_rep2_r2.trinity.fastq.gz
...
-rw-rw-r-- 1 guest01 guest01 16M 9月 17 12:28 rice_N_rep2_r1.trinity.fastq.gz
-rw-rw-r-- 1 guest01 guest01 16M 9月 17 12:28 rice_N_rep2_r2.trinity.fastq.gz
-rw-rw-r-- 1 guest01 guest01 13M 9月 17 12:28 rice_N_rep3_r1.trinity.fastq.gz
-rw-rw-r-- 1 guest01 guest01 13M 9月 17 12:28 rice_N_rep3_r2.trinity.fastq.gz
```

```
$ less rice_D_rep1_r1.trinity.fastq.gz
@MG00HS14:530:C6M7YACXX:8:1101:1210:2214/1 ..... "/1"が付け足されている
GGCCGGAGCATCATCGTCCATTCCACCCATGTCGGCACAGCACCCTGGTACATCTTGGCAATAATTGGGTTGCACAGG
CCCTCCCGCTCCTTCATCT
+
@@@FDDADFFD?FGHHIGIIIEEE4CGIIIIIIIFIIGGID;BCA9CGEHHFGCAC@DDDFFDACEEEEE=A@;?8<>C3?
BDBBDBB(5>B>>CCDCEDA
...
```

Step 2. Trinityによるde novo transcriptome assemble



TrinityによってRNA-Seqリードをアセンブルするシェルスクリプト

```
$ less step2_Trinity.sh
```

- step2_Trinity.sh -

```
ToolDir=$HOME/RNA-Seq_whole_genome/tool  
Trinity_bin=$ToolDir/trinityrnaseq-Trinity-v2.8.5  
Samtools_bin=$ToolDir/samtools-1.9/bin  
Jellyfish_bin=$ToolDir/jellyfish-2.3.0/bin  
Salmon_bin=$ToolDir/salmon-0.14.1-linux_x86_64/bin  
Bowtie2_bin=$ToolDir/bowtie2-2.3.5.1-linux-x86_64  
  
export PATH=$Trinity_bin:$Samtools_bin:$Jellyfish_bin:$Salmon_bin:$Bowtie2_bin:$PATH
```

```
# Step2. De novo transcriptome assemble by Trinity
```

```
Trinity --seqType fq --max_memory 4G --CPU 1 --SS_lib_type RF \  
--samples_file sample_info.txt --output Trinity_out
```

実行するのコマンドはTrinityひとつだが、内部で様々なツールを使用しており、パスの設定が必要。

入力ファイル等は sample_info.txt内に記載することで指定 Stranded RNA-Seqライブラリ のオプションを指定

Step 2. Trinityによるde novo transcriptome assemble



- sample_info.txt -

```
$ cat sample_info.txt
Day      Day_rep1          rice_D_rep1_r1.trinity.fastq.gz rice_D_rep1_r2.trinity.fastq.gz
Day      Day_rep2          rice_D_rep2_r1.trinity.fastq.gz rice_D_rep2_r2.trinity.fastq.gz
Day      Day_rep3          rice_D_rep3_r1.trinity.fastq.gz rice_D_rep3_r2.trinity.fastq.gz
Night    Night_rep1        rice_N_rep1_r1.trinity.fastq.gz rice_N_rep1_r2.trinity.fastq.gz
Night    Night_rep2        rice_N_rep2_r1.trinity.fastq.gz rice_N_rep2_r2.trinity.fastq.gz
Night    Night_rep3        rice_N_rep3_r1.trinity.fastq.gz rice_N_rep3_r2.trinity.fastq.gz
```

- サンプル条件、ラベル、配列ファイルをタブ区切りで記載。
- edgeRによる発現変動遺伝子の検出時にも同じファイルを利用する。
- 解析全体で1つのsample infoファイルを使い回すことで、サンプルの指定間違いなどの防止にもなる。

Step 2. Trinityによるde novo transcriptome assemble



- シェルスクリプトの実行（実行時間：約20分）

```
$ bash ./step2_Trinity.sh
```

アセンブルされた転写産物（コンティグ）配列の確認

```
$ ls Trinity_out/
```

Trinity.fasta アセンブルされた転写産物（コンティグ）配列
Trinity.fasta.gene_trans_map inchworm.K25.L25.fa.finished pipeliner.134196.cmds
Trinity.timing inchworm.kmer_count read_partitions
both.fa insilico_read_normalization recursive_trinity.cmds
recursive_trinity.cmds.completed jellyfish.kmers.fa
...
「TRINITY_DN8_c0_g1_i1」は転写産物配列ID。
「TRINITY_DN8_c0_g1遺伝子座のアイソフォーム1」

```
$ less Trinity_out/Trinity.fasta
```

>TRINITY_DN11_c0_g1_i4 len=2370 path=[1:0-231 3:232-233 4:234-317 5:318-318 6:319-1595
7:1596-1745 8:1746-2147 10:2148-2369]
CCCCCTCTCCTCCTCCCTCTCCGCGCCGCCATCGCGTCGAGCTGCCCGCGAGATCCGCCGTTGACGGAGGAGGGAGTG
AGTGAGTAGCAAGAGGAAGAGGAAGAGGATGGCGTCGGGACGGTACATGGCGTACTCGCCTCGCCCTCCACCACCCGCACTCCCCGCGCATC
CCGGCCTCCGCGCCGCCTCCGCCGACCAAGAAGTACCTCGGGAGCTGC...
...

```
$ grep ">" Trinity_out/Trinity.fasta | wc -l
```

1045 アセンブルされた転写産物数

```
$ grep ">" Trinity_out/Trinity.fasta | cut -d "_" -f 1,2,3,4 | sort | uniq| wc -l
```

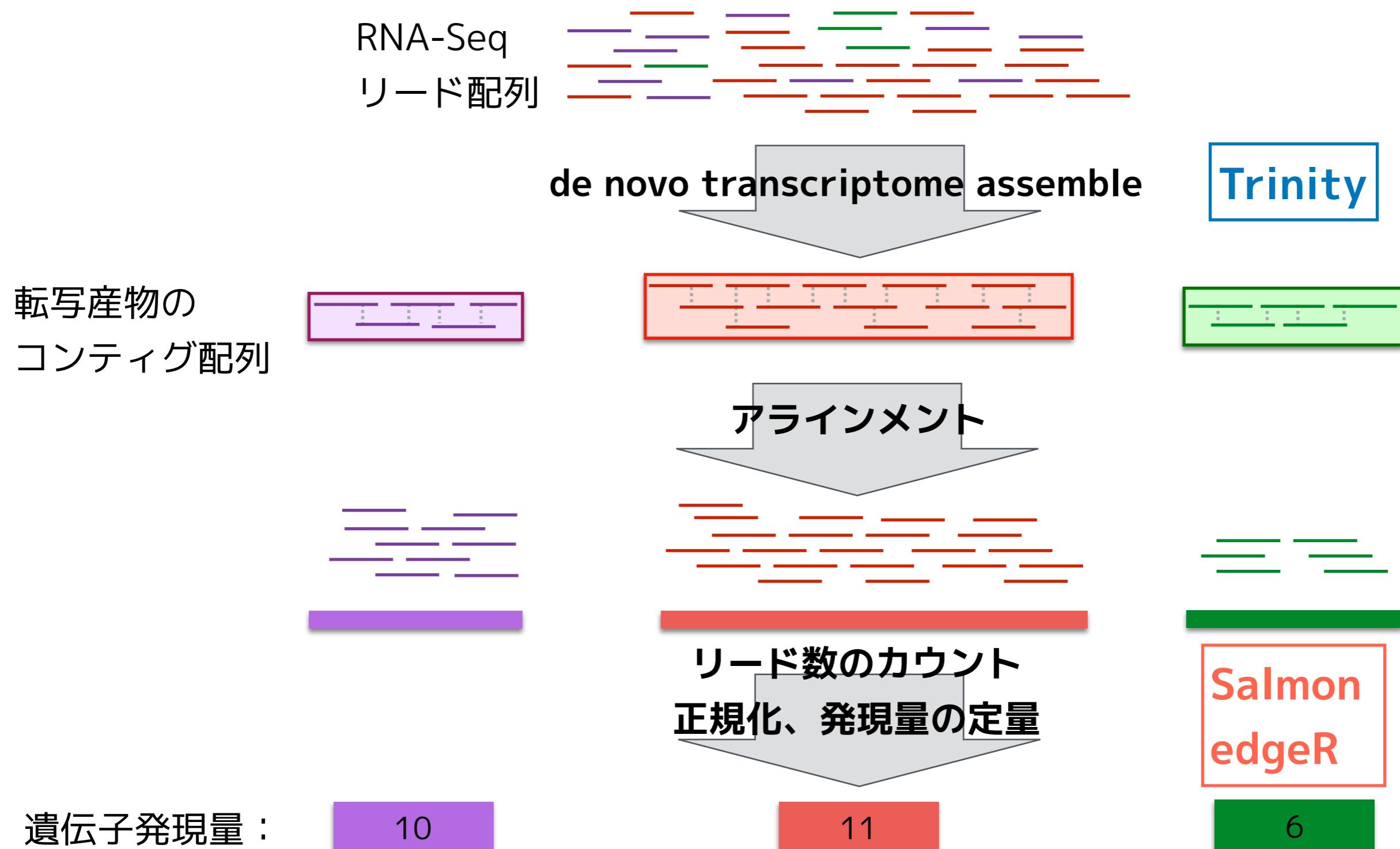
597 アセンブルされた遺伝子座数

de novo transcriptome assemble法を用いた遺伝子発現解析の流れ



まず始めに、RNA-Seqリード配列から転写産物の全長配列を再構築する。

アセンブルされた転写産物配列上にRNA-Seqリードをアラインメントし、転写産物ごとにリード数をカウントすることにより遺伝子発現を定量する。



遺伝子発現量：

10

11

6

Salmon : quasi-mappingによる遺伝子発現量の定量



- リファレンス転写産物配列を元に Suffix arrayを作成し、リード配列を高速に検索。
- 塩基同士のアラインメントは作らず、とにかくリードが由来する転写産物を高速に探索する方法。

Step 3. Salmonによる遺伝子発現量の定量



Salmonによって各Trinityコンティグの遺伝子発現量を定量するシェルスクリプト

```
$ less step3_Salmon.sh
```

- step3_Salmon.shの前半 -

```
ToolDir=$HOME/RNA-Seq_whole_genome/tool  
Trinity_bin=$ToolDir/trinityrnaseq-Trinity-v2.8.5  
Samtools_bin=$ToolDir/samtools-1.9/bin  
Salmon_bin=$ToolDir/salmon-0.14.1-linux_x86_64/bin  
R_bin=/work/NGSworkshop/tool/R-3.6.1/bin
```

```
export PATH=$Trinity_bin:$Samtools_bin:$Salmon_bin:$R_bin:$PATH
```

```
# Step3. Estimating Transcript Abundance by Salmon
```

```
$Trinity_bin/util/align_and_estimate_abundance.pl --transcripts Trinity_out/Trinity.fasta  
\  
--seqType fq --SS_lib_type RF --samples_file sample_info.txt \  
--est_method salmon \  
--thread_count 1 --trinity_mode --prep_reference
```

複数の発現量推定の手法が提供されて
いて、**--est_method**で指定可能

alignment_based: RSEM / eXpress
alignment_free: kallisto / **salmon**

Step 3. Salmonによる遺伝子発現量の定量



- step3_Salmon.shのつづき -

Salmonによってサンプルごとに得られた
遺伝子発現量をまとめ、正規化する。

```
# Build Transcript and Gene Expression Matrices
$Trinity_bin/util/abundance_estimates_to_matrix.pl \
--est_method salmon \
--gene_trans_map Trinity_out/Trinity.fasta.gene_trans_map \
--name_sample_by_basedir --out_prefix salmon \
Day_rep1/quant.sf Day_rep2/quant.sf Day_rep3/quant.sf \
Night_rep1/quant.sf Night_rep2/quant.sf Night_rep3/quant.sf
```

- --out_prefix trans_counts : 出力ファイルのprefix
- --est_method : 発現量の定量方法
- --name_sample_by_basedir : 出力結果のサンプル名をディレクトリ名にする
- 最後にSalmonの結果の4つのファイルをスペース区切りで指定

Step 3. Salmonによる遺伝子発現量の定量



シェルスクリプトの実行 (実行時間：約30秒)

```
$ bash ./step3_Salmon.sh
```

Salmonが出力する、サンプルごとの遺伝子発現量の確認

```
$ less Day_rep1/quant.sf
```

Name	Length	EffectiveLength	TPM	NumReads
...				
TRINITY_DN11_c0_g1_i4	2370	2205.908	17.918052	3.400
TRINITY_DN11_c0_g1_i5	1453	1288.908	3271.279807	362.674
TRINITY_DN11_c0_g1_i6	2222	2057.908	0.000000	0.000
TRINITY_DN11_c0_g1_i7	1831	1666.908	688.705191	98.747
TRINITY_DN11_c0_g1_i1	1568	1403.908	389.791814	47.070
TRINITY_DN11_c0_g1_i2	2731	2566.908	0.000000	0.000
...				

TPM: transcripts per kilobase million

Step 3. Salmonによる遺伝子発現量の定量

全サンプルの発現量データがまとめられたファイル
(リードカウントや正規化された発現量)

```
$ ls -lh salmon.*  
-rw-rw-r-- 1 guest01 guest01 38K 9月 17 12:56 salmon.gene.TMM.EXPR.matrix  
-rw-rw-r-- 1 guest01 guest01 42K 9月 17 12:56 salmon.gene.TPM.not_cross_norm  
-rw-rw-r-- 1 guest01 guest01 350 9月 17 12:56 salmon.gene.TPM.not_cross_norm.TMM_info.txt  
-rw-rw-r-- 1 guest01 guest01 522 9月 17 12:56 salmon.gene.TPM.not_cross_norm.runTMM.R  
-rw-rw-r-- 1 guest01 guest01 33K 9月 17 12:56 salmon.gene.counts.matrix  
-rw-rw-r-- 1 guest01 guest01 69K 9月 17 12:56 salmon.isoform.TMM.EXPR.matrix  
-rw-rw-r-- 1 guest01 guest01 74K 9月 17 12:56 salmon.isoform.TPM.not_cross_norm  
-rw-rw-r-- 1 guest01 guest01 350 9月 17 12:56 salmon.isoform.TPM.not_cross_norm.TMM_info.txt  
-rw-rw-r-- 1 guest01 guest01 528 9月 17 12:56 salmon.isoform.TPM.not_cross_norm.runTMM.R  
-rw-rw-r-- 1 guest01 guest01 64K 9月 17 12:56 salmon.isoform.counts.matrix
```

- *.TMM.EXPR.matrix: TMM正規化後のTPM（サンプル間の発現量の分布を補正済）
- *.TPM.not_cross_norm: TMM正規化前のTPM
- *.counts.matrix: リードカウント情報

Step 3. Salmonによる遺伝子発現量の定量

出力された遺伝子発現量データの確認

```
$ less salmon.gene.counts.matrix ..... 各コンティグごとのリードカウント数
```

	Day_rep1	Day_rep2	Day_rep3		Night_rep1	Night_rep2	Night_rep3
TRINITY_DN0_c0_g1		1074.74	1109.72	1177.57	453.39	765.04	465.04
TRINITY_DN0_c0_g2		0.00	11.59	12.91	0.00	11.31	13.18
TRINITY_DN0_c1_g1		3474.15	2265.54	2495.93	4309.03	4032.49	2973.35
TRINITY_DN0_c2_g1		941.11	1136.04	1562.60	196.39	395.61	397.52
TRINITY_DN100_c0_g1		41.01	48.15	22.86	52.64	47.26	62.23
TRINITY_DN100_c0_g2		166.14	153.42	173.19	231.18	251.42	199.13
TRINITY_DN101_c0_g1		14.54	11.67	3.19	25.39	31.85	39.26
TRINITY_DN101_c0_g2		258.58	62.45	95.17	278.95	102.73	125.59
...							

```
$ less salmon.gene.TMM.EXPR.matrix ..... 各コンティグごとのTMM正規化後のTPM
```

	Day_rep1	Day_rep2	Day_rep3		Night_rep1	Night_rep2	Night_rep3
TRINITY_DN0_c0_g1	8650.120	8005.950	5938.349		3294.978	4060.982	2425.264
TRINITY_DN0_c0_g2	0.000	83.633	65.123		0.000	60.019	68.728
TRINITY_DN0_c1_g1	27961.943	16344.399	12586.696		31315.699	21405.140	15506.694
TRINITY_DN0_c2_g1	7574.590	8195.782	7880.017		1427.220	2099.986	2073.140
TRINITY_DN100_c0_g1	330.076	347.400	115.302		382.563	250.852	324.556
TRINITY_DN100_c0_g2	1337.210	1106.825	873.365		1680.070	1334.571	1038.488
TRINITY_DN101_c0_g1	117.038	84.203	16.100		184.548	169.041	204.728
TRINITY_DN101_c0_g2	2081.190	450.558	479.919		2027.285	545.291	654.959
...							

Step 4. edgeRによる遺伝子発現量の違いの統計検定



edgeRによる発現変動遺伝子を検出するシェルスクリプト

```
$ less step4_edgeR.sh
```

- step4_edgeR.sh -

```
ToolDir=$HOME/RNA-Seq_whole_genome/tool  
Trinity_bin=$ToolDir/trinityrnaseq-Trinity-v2.8.5  
R_bin=/work/NGSworkshop/tool/R-3.6.1/bin ..... RにはedgeRパッケージ  
export PATH=$Trinity_bin:$R_bin:$PATH ..... をインストールしておく  
  
# Step 4. Differential Expression Analysis by edgeR  
$Trinity_bin/Analysis/DifferentialExpression/run_DE_analysis.pl \  
--matrix salmon.gene.counts.matrix ..... マージしたカウントデータ  
--method edgeR ..... 手法を選択 (edgeR|DESeq2|voom|ROTS)  
--samples_file sample_info.txt ..... Trinityでも使ったサンプル情報ファイル  
--output edgeR_out ..... 出力先
```

Step 4. edgeRによる遺伝子発現量の違いの統計検定

シェルスクリプトの実行 (実行時間：約5秒)

```
$ bash ./step4_edgeR.sh
```

出力されたDEG解析結果の確認

```
$ less edgeR_out/salmon.gene.counts.matrix.Day_vs_Night.edgeR.DE_results
```

sampleA	sampleB	logFC	logCPM	PValue	FDR
TRINITY_DN20_c0_g1	Day	Night	-3.60910954106426	15.7385010545598	9.76517320797912e-25
TRINITY_DN17_c0_g1	Day	Night	-5.07277658132845	12.9935453393215	1.79369745794944e-18
TRINITY_DN74_c0_g1	Day	Night	2.90252919563421	15.7808172939919	4.74431833680181e-12
TRINITY_DN47_c0_g1	Day	Night	2.70995304895816	12.1237688054613	4.6989024282568e-11
TRINITY_DN123_c0_g3	Day	Night	-2.97773297559472	13.5583202256039	6.13559962140135e-10
TRINITY_DN0_c2_g1	Day	Night	2.06470113525029	12.2558805934621	4.12609113274095e-08
TRINITY_DN91_c0_g1	Day	Night	3.31030585045054	10.7238806061214	4.78398009276254e-08
...					

logFC:log2(fold change), logCPM:log2(counts per million), PValue:P-value, FDR:False Discovery Rate

昼と夜の遺伝子発現量の差の有意差が小さい順に、
転写産物や遺伝子座が出力されている。

De novo assembleの確かしさの検証



1. 元のリードをコンティグに再マッピングすることで、どれぐらいのリードがアセンブルに貢献しているかを見る。
2. コンティグ配列をクエリーとして、既知のタンパク質配列データベースに相同性検索をし、マップ率やカバー率によって、アセンブルの良し悪しを評価する。

上記、1、2を実行するためのシェルスクリプト (step5_eval_assembly.sh) を用意しているが、2番染色体の一部のみを使った演習データで行なっても結果の評価が難しいため、**スライド上では全ranscriptomeを対象に解析した結果を示す。**

興味があれば演習用のデータで実行してみても構いません。

シェルスクリプトの実行 (実行時間：約10分)

```
$ bash ./step5_eval_assembly.sh
```

De novo assembleの確からしさの検証1

アセンブルに用いたRNA-Seqリードをコンティグに再マッピングし、どれくらいのリードがアセンブリに貢献しているかを調べる。

- step5_eval_assembly.sh の前半 -

```
ToolDir=$HOME/RNA-Seq/tool
Bowtie2_bin=$ToolDir/bowtie2-2.3.5.1-linux-x86_64
Samtools_bin=$ToolDir/samtools-1.9/bin
BLAST_bin=$ToolDir/ncbi-blast-2.9.0+/bin
Trinity_bin=$ToolDir/trinityrnaseq-Trinity-v2.8.5

export PATH=$Bowtie2_bin:$Samtools_bin:$BLAST_bin:$PATH

### Step5. Evaluation of de novo transcriptome assemblies by Trinity
# check read representation
bowtie2-build Trinity_out/Trinity.fasta Trinity.fasta ..... bowtie2のインデックス作成
bowtie2 -q --no-unal -k 20 -x Trinity.fasta \ ..... 以降はbowtie2によるアラインメント
-1
rice_D_rep1_r1.trinity.fastq.gz, rice_D_rep2_r1.trinity.fastq.gz, rice_D_rep3_r1.trinity.fastq.gz, rice_N_rep1_r1.trinity.fastq.gz, rice_N_rep2_r1.trinity.fastq.gz, rice_N_rep3_r1.trinity.fastq.gz \
-2
rice_D_rep1_r2.trinity.fastq.gz, rice_D_rep2_r2.trinity.fastq.gz, rice_D_rep3_r2.trinity.fastq.gz, rice_N_rep1_r2.trinity.fastq.gz, rice_N_rep2_r2.trinity.fastq.gz, rice_N_rep3_r2.trinity.fastq.gz \
| samtools view -Sb -o bowtie2_read_to_contig.bam
```

詳しくは「Assessing the Read Content of the Transcriptome Assembly」を参照

<https://github.com/trinityrnaseq/trinityrnaseq/wiki/RNA-Seq-Read-Representation-by-Trinity-Assembly>

De novo assembleの確かしさの検証1

Bowtie2によるアラインメントの統計情報は標準エラー出力で確認できる。

```
120885050 reads; of these:  
 120885050 (100.00%) were paired; of these:  
   12576489 (10.40%) aligned concordantly 0 times  
   21693242 (17.95%) aligned concordantly exactly 1 time  
   86615319 (71.65%) aligned concordantly >1 times → 89.6%  
   ----  
   12576489 pairs aligned concordantly 0 times; of these:  
     1562524 (12.42%) aligned discordantly 1 time  
   ----  
   11013965 pairs aligned 0 times concordantly or discordantly; of these:  
     22027930 mates make up the pairs; of these:  
       5554496 (25.22%) aligned 0 times  
       739220 (3.36%) aligned exactly 1 time  
       15734214 (71.43%) aligned >1 times  
97.70% overall alignment rate
```

- aligned concordantly exactly 1 time
- aligned concordantly >1 times

を合わせたものが、70~80%になるのが一般的のようです。

De novo assembleの確からしさの検証2



コンティグ配列をクエリーとして、既知のタンパク質配列データベースに相同意検索をし、マップ率やカバー率によって、アセンブルの良し悪しを評価する。

- step5_eval_assembly.sh の後半 -

```
# homology search against rice protein sequences  
ln -s ../data/rice_protein.fa  
$BLAST_bin/makeblastdb -dbtype prot -in rice_protein.fa  
$BLAST_bin/blastx -query Trinity_out/Trinity.fasta -db rice_protein.fa \  
-task blastx-fast -evalue 1e-20 -max_target_seqs 1 -outfmt 6 \  
-num_threads $CPU \  
-out blastx_trinity_to_rice_protein.outfmt6
```

イネの全タンパク質配列 (FASTA) の
ファイルのシンボリックリンクを配置
し、BLAST用DBを作成

```
perl $Trinity_bin/util/analyze_blastPlus_topHit_coverage.pl  
blastx_trinity_to_rice_protein.outfmt6 Trinity_out/Trinity.fasta rice_protein.fa >  
blastx_TopHitCoverage.txt
```

Trinityのコンティグを
クエリにして、イネの全
タンパク質配列に対して
BLASTX検索

Trinityに付属の
スクリプトで集計

詳しくは「Counting Full Length Trinity Transcripts」を参照

<https://github.com/trinityrnaseq/trinityrnaseq/wiki/Counting-Full-Length-Trinity-Transcripts>

De novo assembleの確からしさの検証2



Trinityのコンティグ配列（148,683本）をクエリーとして、イネの全タンパク質配列（42,229本）に対してBLASTX検索し、イネの各タンパク質にベストヒットするコンティグ配列のCoverage率（タンパク質全長のうちTrinityコンティグとアラインメントがとれた領域の割合）を集計。

#hit_pct_cov_bin	count_in_bin	>bin_below
100	13692	13692
90	2417	16109
80	1536	17645
70	1219	18864
60	1132	19996
50	1120	21116
40	1058	22174
30	1068	23242
20	796	24038
10	134	24172

イネの全タンパク質（42,229本）のうちの約半数（21,116本）が、Coverage>50%という閾値でTrinityのコンティグに対応するものが存在している。

個別の遺伝子について調べてみる（LHY遺伝子の場合）

De novo assembleの確からしさの検証2で使用したスクリプトの結果

(blastx_trinity_to_rice_protein.outfmt6.w_pct_hit_length) を見ると、LHY (Os04t0583900-01)にヒットするコンティグが複数ある。

#qseqid	sseqid	pident	length	mismatch	gapopen	qstart	qend	sstart	send	evalue	bitscore
db_hit_len		pct_hit_len_aligned		hit_descr							
TRINITY_DN376_c0_g1_i5	Os04t0583900-01	99.578	237	1	0	1559	2269	228	228	464	
4.31e-154	461	237	51.08	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i3	Os04t0583900-01	99.578	237	1	0	1479	2189	228	228	464	
2.05e-154	461	237	51.08	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i6	Os04t0583900-01	99.578	237	1	0	1757	2467	228	228	464	
2.49e-153	461	237	51.08	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i1	Os04t0583900-01	99.721	359	1	0	1154	2230	106	106	464	
0.0	668	359	77.37	Similar to LHY protein.							
TRINITY_DN376_c0_g4_i1	Os04t0583900-01	95.385	65	3	0	204	10	28	28	92	
1.64e-40	137	65	14.01	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i2	Os04t0583900-01	99.578	237	1	0	1660	2370	228	228	464	
1.08e-153	461	237	51.08	Similar to LHY protein.							
TRINITY_DN376_c1_g1_i1	Os04t0583900-01	99.099	111	1	0	359	27	354	354	464	
8.75e-61	192	111	23.92	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i7	Os04t0583900-01	100.000	358	0	0	959	2032	107	107	464	
0.0	668	358	77.16	Similar to LHY protein.							
TRINITY_DN376_c0_g4_i2	Os04t0583900-01	93.333	45	3	0	144	10	48	48	92	
2.55e-23	90.9	45	9.70	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i8	Os04t0583900-01	100.000	464	0	0	561	1952	1	1	464	
0.0	894	464	100.00	Similar to LHY protein.							
TRINITY_DN376_c0_g1_i9	Os04t0583900-01	99.721	359	1	0	1057	2133	106	106	464	
0.0	668	359	77.37	Similar to LHY protein.							

個別の遺伝子について調べてみる（LHY遺伝子の場合）

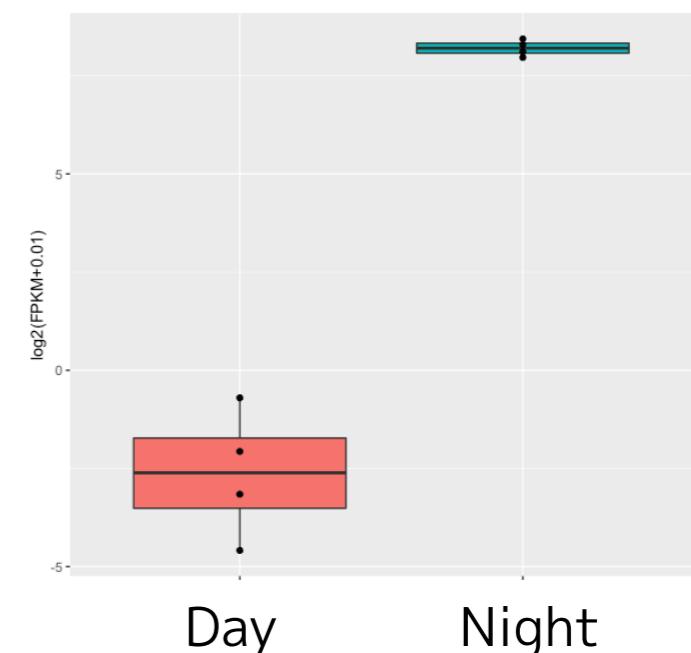
BLASTXの結果から、LHY転写産物（Os04t0583900-01）に最も相同意が高いTrinity contigは「TRINITY_DN376_c0_g1_i8」であることが分かる。

#qseqid	sseqid	pident	length	mismatch	gapopen	qstart	qend	sstart	send
eval	bitscore		db_hit_len		pct_hit_len_aligned		hit_descr		
TRINITY_DN376_c0_g1_i8	Os04t0583900-01	100.000	464		0		0	561	1952
1	464	0.0	894	464	100.00	Similar to LHY protein.			

「TRINITY_DN376_c0_g1」遺伝子座について、edgeRの結果（salmon.gene.counts.matrix.Day_vs_Night.edgeR.DE_results）を調べてみると昼夜の遺伝子発現量に有意差がみられる（昼<夜）。

	sampleA	sampleB	logFC	logCPM	PValue	FDR
TRINITY_DN376_c0_g1	Day	Night	-9.488	7.318	2.119e-41	4.962e-37

昼にくらべて、夜の遺伝子発現量が有意に高いという結果になっており、リファレンス情報を用いたRNA-Seq解析の結果（右図）と一致する。

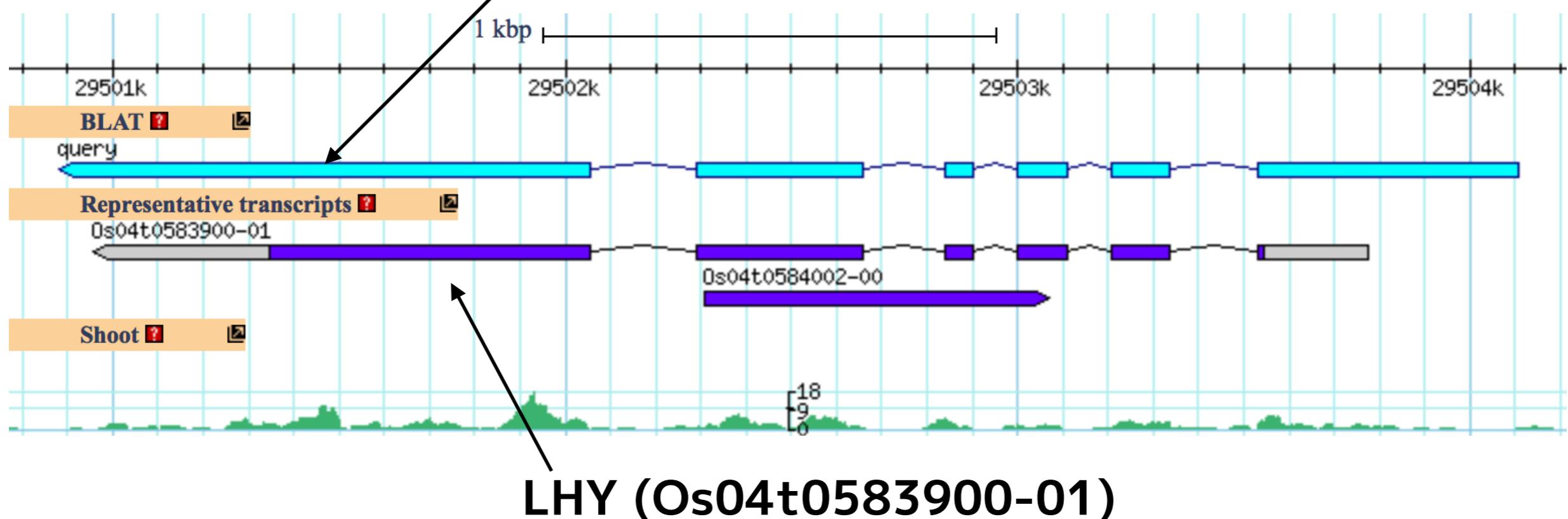


遺伝子全長がきちんとアセンブルされているか？

近縁種のゲノム配列や遺伝子アノテーションが整備されている場合、BLAT等でアラインメントし、マップ率や遺伝子構造を比較することでも評価が可能です。

RAP-DBのBLAT検索によって
イネゲノムにマッピングされたコンティグ配列

TRINITY_DN376_c0_g1_i8 (2,417 bp)



コンティグ配列のアノテーション



得られた転写産物が何者なのかが知りたい！
そのためにはコンティグ配列のアノテーションが必要

[TransDecoder / TransDecoder](https://github.com/TransDecoder/TransDecoder)

Watch 14 Star 83 Fork 34

Code Issues 54 Pull requests 0 Projects 0 Wiki Security Insights

Home

Patrick Douglas edited this page on 31 Jul 2018 · 8 revisions

TransDecoder (Find Coding Regions Within Transcripts)

TransDecoder identifies candidate coding regions within transcript sequences, such as those generated by de novo RNA-Seq transcript assembly using Trinity, or constructed based on RNA-Seq alignments to the genome using TopHat and Cufflinks.

TransDecoder identifies likely coding sequences based on the following criteria:

- a minimum length open reading frame (ORF) is found in a transcript sequence
- a log-likelihood score similar to what is computed by the [GeneID](#) software is > 0.
- the above coding score is greatest when the ORF is scored in the 1st reading frame as compared to scores in the other 2 forward reading frames.
- if a candidate ORF is found fully encapsulated by the coordinates of another candidate ORF, the longer one is reported. However, a single transcript can report multiple ORFs (allowing for operons, chimeras, etc).
- a PSSM is built/trained/used to refine the start codon prediction.
- optional the putative peptide has a match to a Pfam domain above the noise cutoff score.

Obtaining TransDecoder

The latest release of TransDecoder can be found [here](#).

<http://transdecoder.github.io/>

転写産物の塩基配列からタンパク質
コード領域（CDS）を予測

[Trinotate / Trinotate.github.io](https://github.com/Trinotate/Trinotate.github.io)

Watch 4 Star 18 Fork 7

Code Issues 16 Pull requests 0 Projects 0 Wiki Security Insights

Home

Brian Haas edited this page on 13 Jul 2018 · 8 revisions

Trinotate: Transcriptome Functional Annotation and Analysis

Trinotate

Trinity

Inchworm Chrysanthemum Butterfly

NCBI BLAST

Pfam

UniProt

eggNOG

SQLite

the Gene Ontology

RNA-Seq → Trinity → Transcripts/Proteins → Functional Data → Discovery

Automated Higher Order Biological Analysis

Pages 7

- Home
- Software installation and data required
- Loading Results into a Trinotate SQLite Database
- Automated Execution of Trinotate: Running computes and loading results
- TrinotateWeb for navigating annotation and expression data
- Contact info and References
- Frequent asked questions (FAQ) and known bugs

Clone this wiki locally

<https://github.com/Trinot>

<https://trinotate.github.io/>

- 機能が分かっているホモログ
 - 機能ドメイン
 - 機能分類（Gene Ontology）
-) 等の注釈付
を行う

- ▶リファレンス情報を用いたRNA-Seq解析
- ▶De novoトランスクriプトーム解析
- ▶**RNA-Seqデータを用いた多型検出**

RNA-Seqデータを用いた多型検出の特徴

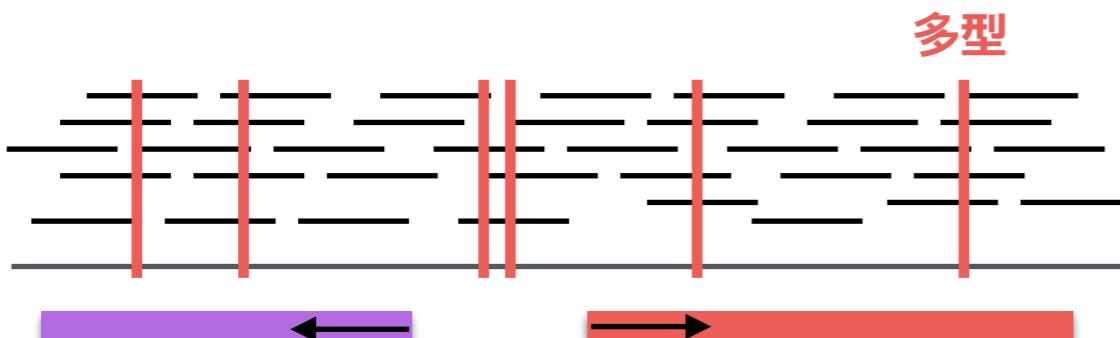
良い点

- ・ 巨大なゲノムでも効率よく多型情報が得られる。
- ・ 遺伝子発現量と多型情報が同時に得られる。
- ・ 転写領域は進化的に保存されており、進化的距離の離れた種の比較でも効率よく多型情報が得られる。

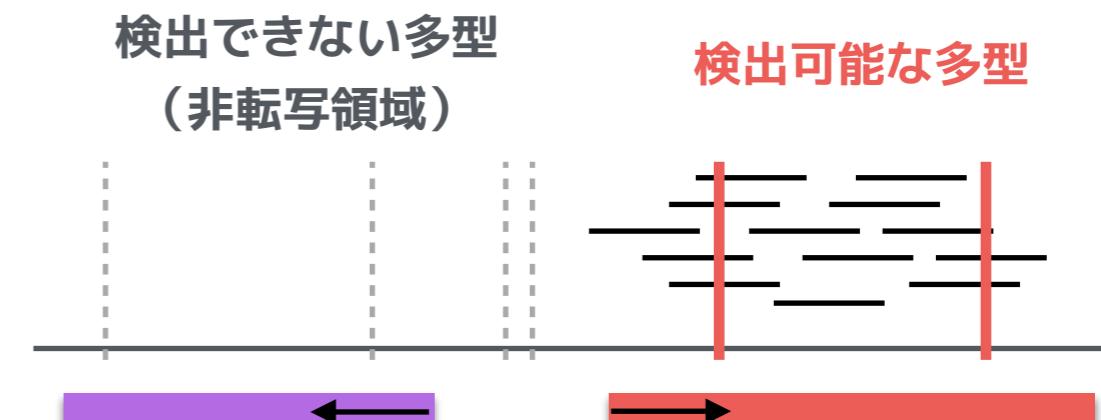
悪い点

- ・ 低発現、または発現していない遺伝子の多型情報は得られない。
- ・ 遺伝子発現解析と同時にできるが、転写開始点上流の発現調節領域の多型情報は得られない。

ゲノムリシーケンスデータ



RNA-Seqデータ



目的：RNA-Seqデータを用いて日本晴とコシヒカリの間のゲノム配列の違い（多型）を調べる

日本晴
(リファレンスゲノム)

VS



コシヒカリ (RNA-Seq)



12:00 PM

rice_N_rep1-3

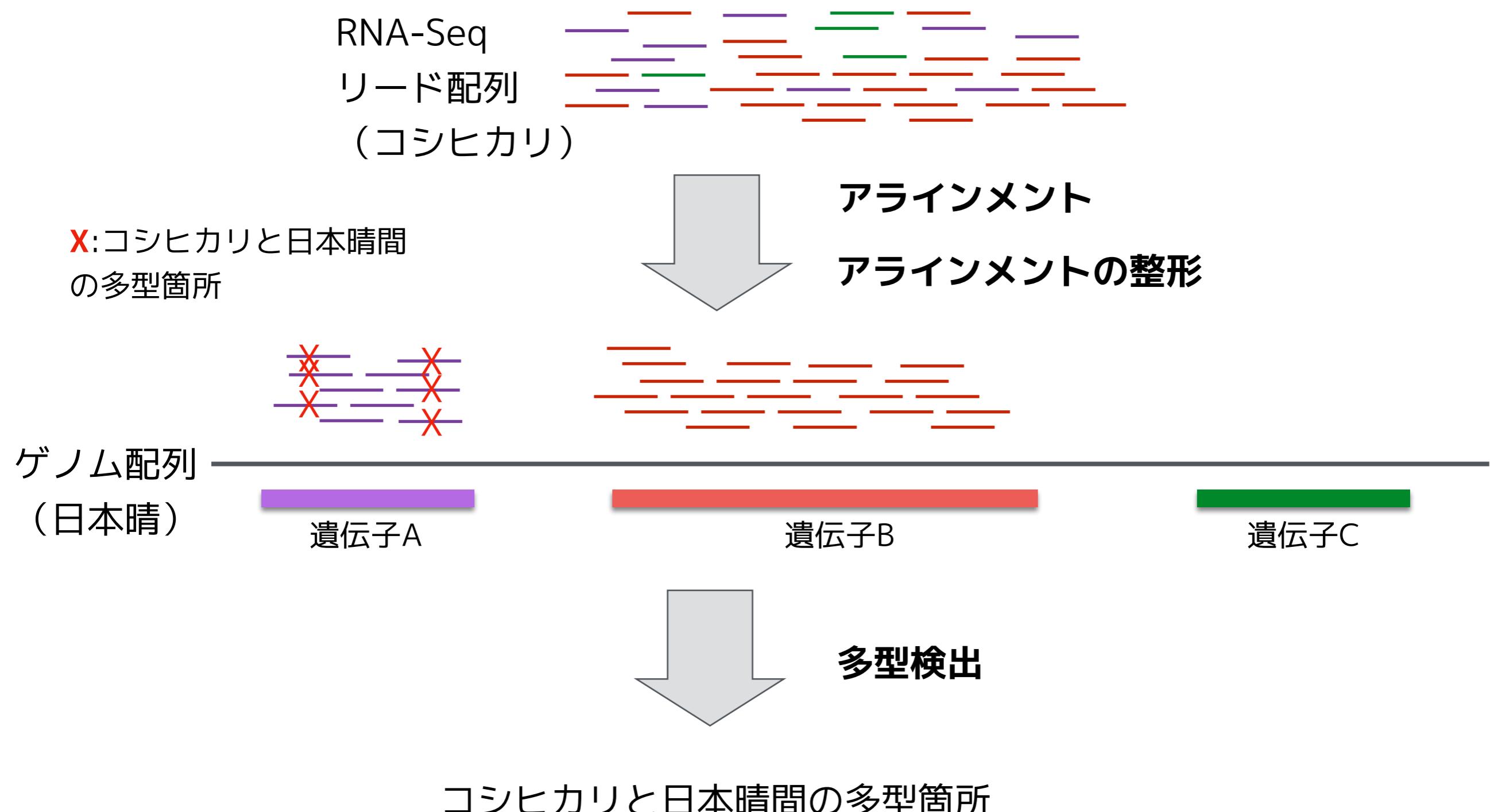
00:00 AM

- 田んぼで栽培する、イネ（コシヒカリ）の葉身のサンプル。
- 3つの異なる生育ステージ（3反復）において、**昼（12時）**と**夜（0時）**にサンプリング（2条件）。
- Illumina社の**Stranded mRNA-Seq**法でライブラリ調製。
- Illumina社のHiSeq2000による、**101bpのPaired-endシーケンシング**。

*全データでは解析に時間がかかるため、2番染色体の特定の領域（2Mbp）にアラインメントされるリードだけを事前に抽出しており、演習ではそれを利用する。

RNA-Seqデータを用いた多型検出の流れ

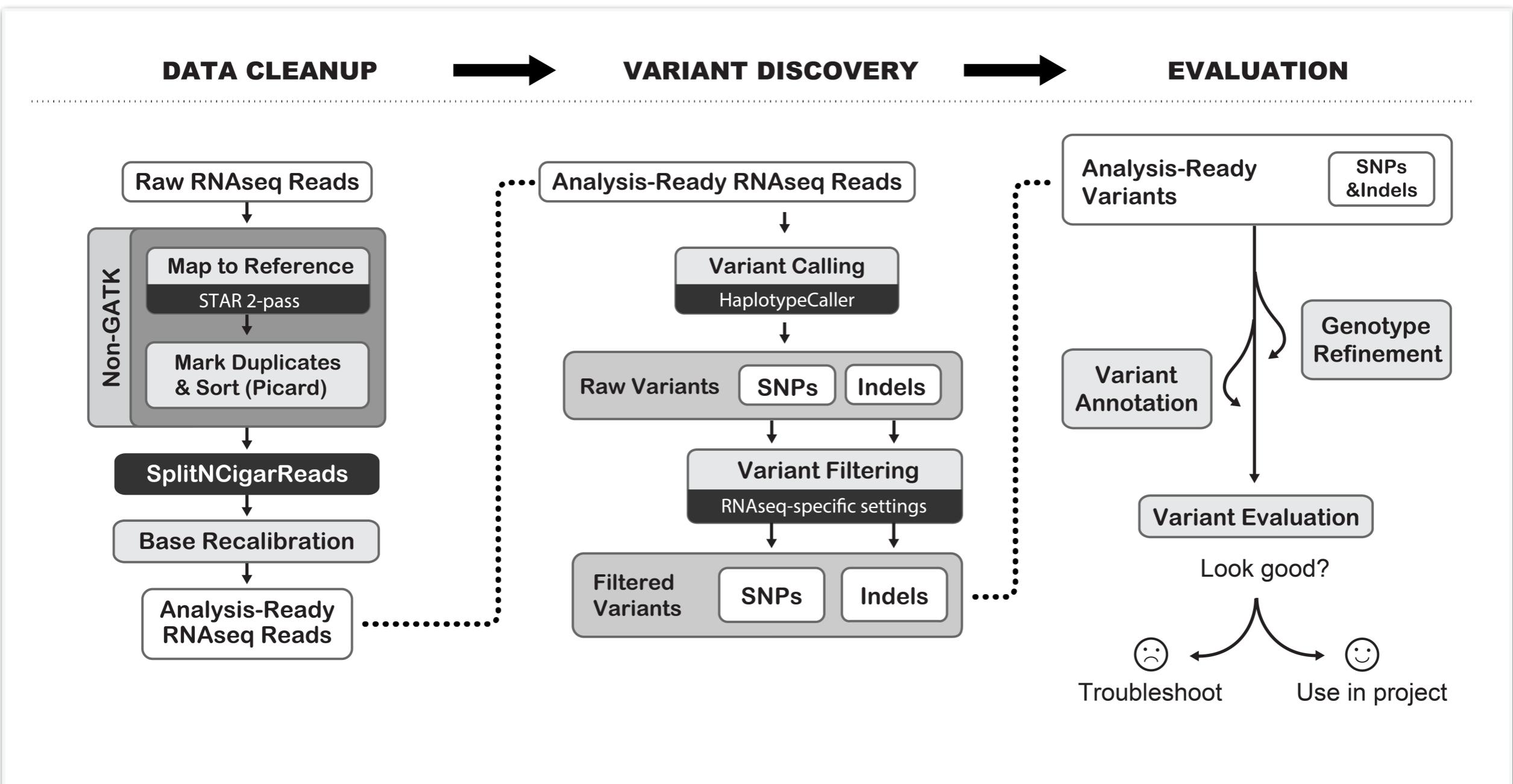
リファレンスとは異なる種や品種由来のRNA-Seqリードをゲノム配列にアラインメント（マッピング）し、種間や品種間の遺伝子配列の違い（多型情報）を得る。



GATKのウェブサイトのBest Practices

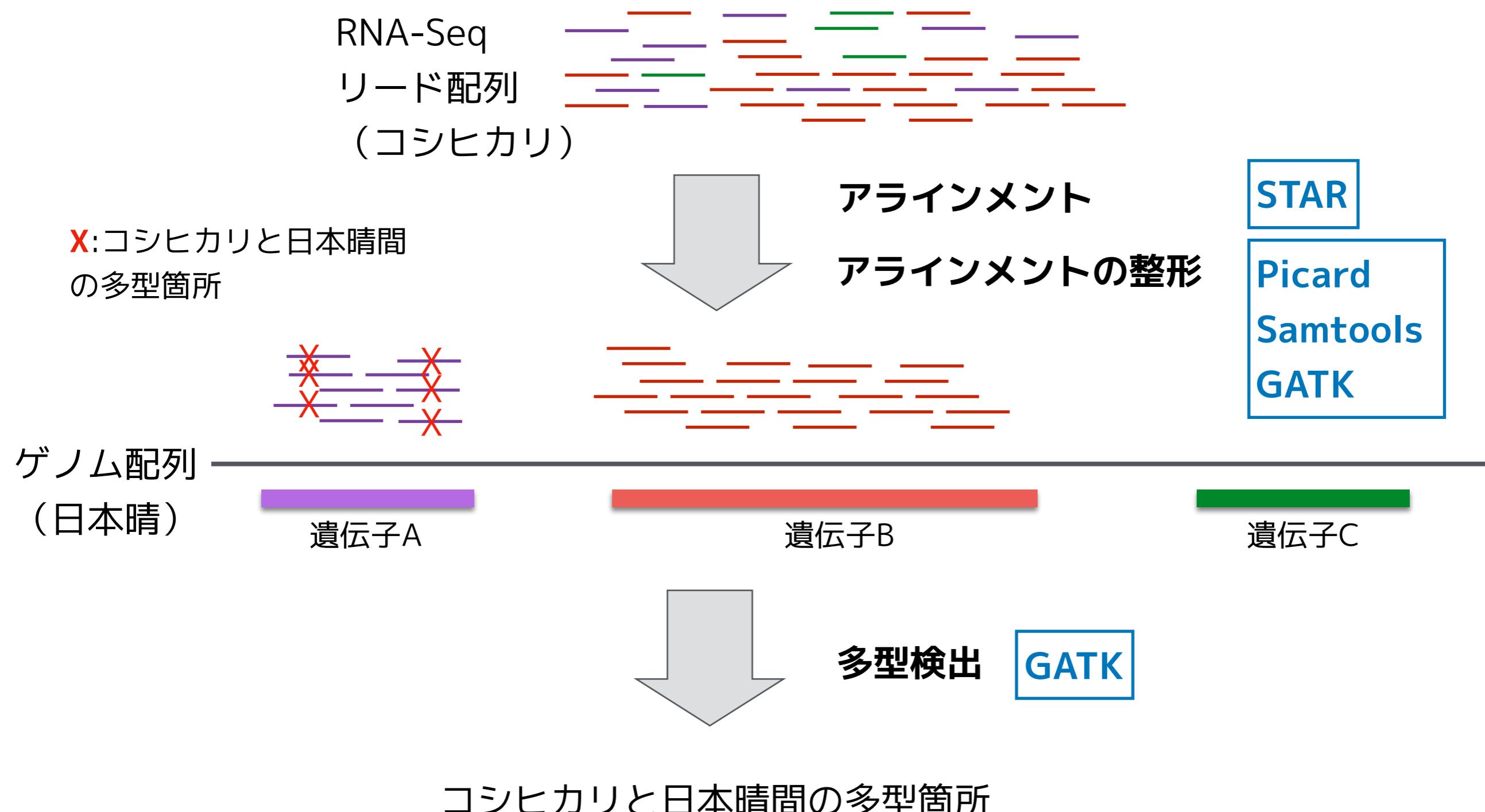


<https://software.broadinstitute.org/gatk/best-practices/workflow?id=11164>



RNA-Seqデータを用いた多型検出の流れ

リファレンスとは異なる種や品種由来のRNA-Seqリードをゲノム配列にアラインメント（マッピング）し、種間や品種間の遺伝子配列の違い（多型情報）を得る。



Step1. STARのためのインデックス作成

RNA-Seqデータを用いた多型検出用ディレクトリ「varcall」に移動

```
$ cd ~/RNA-Seq/varcall
```

解析用ディレクトリ内のファイルを確認

```
$ ls
step1_make_STAR_index.sh      step3_BAM_Processing.sh
step2_STAR.sh                  step4_GATK.sh
```

シェルスクリプトが5つ用意されており、順番にシェルスクリプトを実行することで解析が進められるようになっている。

```
$ bash ./step1_make_STAR_index.sh
$ bash ./step2_STAR.sh
$ bash ./step3_BAM_Processing.sh
$ bash ./step4_GATK.sh
```

Step1. STARのためのインデックス作成

STAR用のゲノム配列のインデックスを作成するシェルスクリプト

```
$ less step1_make_STAR_index.sh
```

- step1_make_STAR_index.sh -

```
DataDir=$HOME/RNA-Seq_whole_genome/data  
ToolDir=$HOME/RNA-Seq_whole_genome/tool  
STAR_bin=$ToolDir/STAR-2.7.2b/bin/Linux_x86_64
```

```
export PATH=$STAR_bin:$PATH
```

```
### Step1. Make genome index for STAR
```

```
GenomeDir=STAR_index  
mkdir $GenomeDir
```

```
STAR --runMode genomeGenerate \  
      --genomeFastaFiles $DataDir/genome.fa \  
      --genomeDir $GenomeDir \  
      --sjdbGTFfile $DataDir/annotation.gtf \  
      --sjdbOverhang 100 --runThreadN 1
```

..... インデックス作成モードを指定してSTARを実行
..... ゲノム配列 (FASTA) の指定
..... インデックス出力ディレクトリの指定
..... 遺伝子アノテーション (GTF) の指定
..... sjdbOverhangは (リード長-1) を指定

パス等の設定

インデックスの出力
ディレクトリの作成

Step1. STARのためのインデックス作成

シェルスクリプトの実行 (実行時間：約1分)

```
$ bash ./step1_make_STAR_index.sh
```

STAR_indexディレクトリ中に作成されたインデックスファイルの確認

```
$ ls -lh STAR_index/
合計 1.8G
-rw-rw-r-- 1 guest01 guest01 35M 9月 17 12:14 Genome
-rw-rw-r-- 1 guest01 guest01 285M 9月 17 12:14 SA
-rw-rw-r-- 1 guest01 guest01 1.5G 9月 17 12:14 SAindex
-rw-rw-r-- 1 guest01 guest01 9 9月 17 12:13 chrLength.txt
-rw-rw-r-- 1 guest01 guest01 6 9月 17 12:13 chrName.txt
-rw-rw-r-- 1 guest01 guest01 15 9月 17 12:13 chrNameLength.txt
-rw-rw-r-- 1 guest01 guest01 11 9月 17 12:13 chrStart.txt
-rw-rw-r-- 1 guest01 guest01 41K 9月 17 12:14 exonGeTrInfo.tab
-rw-rw-r-- 1 guest01 guest01 20K 9月 17 12:14 exonInfo.tab
-rw-rw-r-- 1 guest01 guest01 3.8K 9月 17 12:14 geneInfo.tab
-rw-rw-r-- 1 guest01 guest01 644 9月 17 12:14 genomeParameters.txt
-rw-rw-r-- 1 guest01 guest01 26K 9月 17 12:14 sjdbInfo.txt
-rw-rw-r-- 1 guest01 guest01 26K 9月 17 12:14 sjdbList.fromGTF.out.tab
-rw-rw-r-- 1 guest01 guest01 26K 9月 17 12:14 sjdbList.out.tab
-rw-rw-r-- 1 guest01 guest01 18K 9月 17 12:14 transcriptInfo.tab
```

Step2. STARによるRNA-Seqリードのアラインメント



STARによってRNA-Seqリードをゲノム配列にアラインメントするシェルスクリプト

```
$ less step2_STAR.sh
```

- step2_STAR.sh -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
STAR_bin=$ToolDir/STAR-2.7.2b/bin/Linux_x86_64
```

```
export PATH=$STAR_bin:$PATH
```

```
### Step2. Alignment of RNA-Seq reads by STAR
```

```
GenomeDir=STAR_index
```

```
Read1_list=$DataDir/rice_D_rep1_r1.org.fastq.gz,$DataDir/  
rice_D_rep2_r1.org.fastq.gz,$DataDir/rice_D_rep3_r1.org.fastq.gz,$DataDir/  
rice_N_rep1_r1.org.fastq.gz,$DataDir/rice_N_rep2_r1.org.fastq.gz,$DataDir/  
rice_D_rep3_r1.org.fastq.gz
```

```
Read2_list=$DataDir/rice_D_rep1_r2.org.fastq.gz,$DataDir/  
rice_D_rep2_r2.org.fastq.gz,$DataDir/rice_D_rep3_r2.org.fastq.gz,$DataDir/  
rice_N_rep1_r2.org.fastq.gz,$DataDir/rice_N_rep2_r2.org.fastq.gz,$DataDir/  
rice_D_rep3_r2.org.fastq.gz
```

```
STAR --genomeDir $GenomeDir --readFilesCommand "gunzip -c" \  
--readFilesIn $Read1_list $Read2_list \  
--alignIntronMin 20 --alignIntronMax 10000 \  
--outSAMtype BAM SortedByCoordinate \  
--runThreadN 1
```

パス等の設定

全サンプルをまとめて
変数に格納

リードファイルの圧縮形式に応じて指定

STARによる
アラインメントを実行

Step2. STARによるRNA-Seqリードのアラインメント



シェルスクリプトの実行 (実行時間：約1分)

```
$ bash ./step2_STAR.sh
```

作成されたアラインメントファイルの確認

```
$ ls -lhtr
合計 133M
...
-rw-rw-r-- 1 guest01 guest01 133M 9月 17 12:17 Aligned.sortedByCoord.out.bam
-rw-rw-r-- 1 guest01 guest01 59K 9月 17 12:17 SJ.out.tab
-rw-rw-r-- 1 guest01 guest01 246 9月 17 12:17 Log.progress.out
-rw-rw-r-- 1 guest01 guest01 25K 9月 17 12:17 Log.out
-rw-rw-r-- 1 guest01 guest01 1.9K 9月 17 12:17 Log.final.out
```

すべてのサンプルをまとめて1サンプルとしてアラインメントしたので、1つの BAMファイル（Aligned.sortedByCoord.out.bam）が出力されている。

Step3. アライメントデータの加工

PicardやGATKでアライメントデータを加工し、下流の解析に適したものにするシェルスクリプト。

```
$ less step3_BAM_Processing.sh
```

- step3_BAM_Processing.sh の前半 -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
Picard_bin=$ToolDir/picard-2.20.7  
Samtools_bin=$ToolDir/samtools-1.9  
GATK_bin=$ToolDir/gatk-4.1.3.1  
  
Export PATH=$Samtools_bin:$GATK_bin:$PATH  
  
### Step3. Processing of alignment (BAM) file  
java -jar $Picard_bin/picard.jar AddOrReplaceReadGroups \  
I=Aligned.sortedByCoord.out.bam \  
O=Aligned.sortedByCoord.RG.out.bam \  
SO=coordinate \  
RGID=Koshihikari RGLB=TruSeq_RNA_stranded RGPL=illumina RGPU=HiSeq2000  
RGSM=Koshihikari
```

パス等の設定

アライメントデータにRG (read group) 情報を追加した上で、位置でソートする。

Step3. アライメントデータの加工

- step3_BAM_Processing.shのつづき -

```
java -jar $Picard_bin/picard.jar MarkDuplicates \
I=Aligned.sortedByCoord.RG.out.bam \
O=Aligned.sortedByCoord.RG.MD.out.bam \
CREATE_INDEX=true \
VALIDATION_STRINGENCY=SILENT \
M=Aligned.sortedByCoord.RG.MD.metrics
```

PCRによって重複した（冗長になった）リードを取り除く

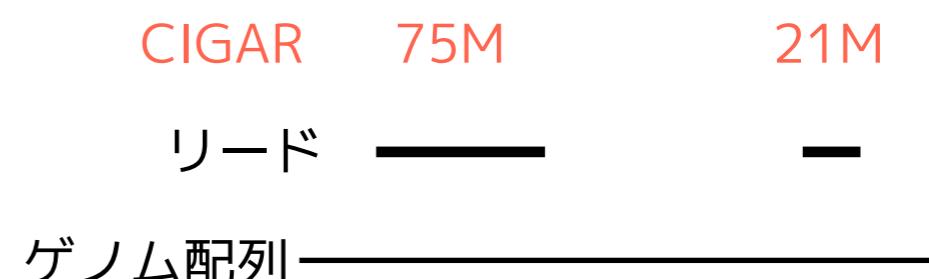
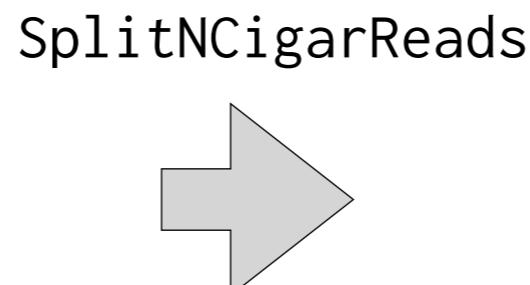
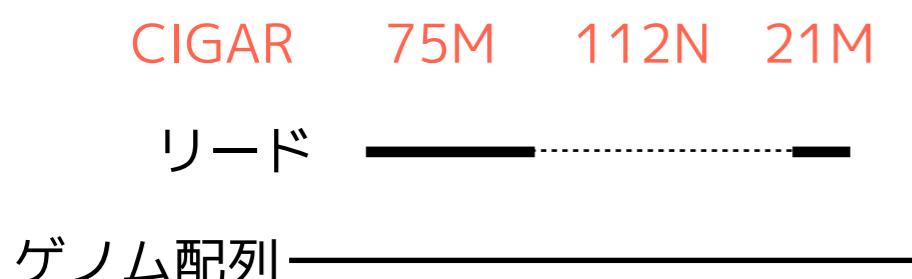
```
samtools faidx $DataDir/genome.fa
```

```
java -jar $Picard_bin/picard.jar CreateSequenceDictionary \
R=$DataDir/genome.fa \
O=$DataDir/genome.dict
```

ゲノム配列のindexやdictionaryの作成

```
gatk SplitNCigarReads \
-R $DataDir/genome.fa \
-I Aligned.sortedByCoord.RG.MD.out.bam \
-O Aligned.sortedByCoord.RG.MD.SplitN.out.bam
```

アライメント中のイントロン領域情報を除く



Step3. アラインメントデータの加工

シェルスクリプトの実行 (実行時間：約2分)

```
$ bash ./step3_BAM_Processing.sh
```

作成されたアラインメントファイルの確認

```
$ ls -lhtr
合計 562M
...
-rw-rw-r-- 1 guest01 guest01 118M 9月 17 12:18 Aligned.sortedByCoord.RG.out.bam
-rw-rw-r-- 1 guest01 guest01 120M 9月 17 12:19 Aligned.sortedByCoord.RG.MD.out.bam
-rw-rw-r-- 1 guest01 guest01 23K 9月 17 12:19 Aligned.sortedByCoord.RG.MD.out.bai
-rw-rw-r-- 1 guest01 guest01 2.8K 9月 17 12:19 Aligned.sortedByCoord.RG.MD.metrics
-rw-rw-r-- 1 guest01 guest01 193M 9月 17 12:20 Aligned.sortedByCoord.RG.MD.SplitN.out.bam
-rw-rw-r-- 1 guest01 guest01 22K 9月 17 12:20 Aligned.sortedByCoord.RG.MD.SplitN.out.bai
```

各ステップの出力ファイルが複数存在するが、最終アラインメントは「Aligned.sortedByCoord.RG.MD.SplitN.out.bam」である。

Step4. GATK HaplotypeCallerによる多型検出

GATK HaplotypeCallerによってRNA-Seqリードのアラインメントを元に多型(SNP、InDel)を検出するシェルスクリプト

```
$ less step4_GATK.sh
```

- step4_GATK.sh -

```
DataDir=$HOME/RNA-Seq/data  
ToolDir=$HOME/RNA-Seq/tool  
GATK_bin=$ToolDir/gatk-4.1.3.0
```

```
export PATH=$GATK_bin:$PATH
```

```
### Step4. Detection of variations by GATK HaplotypeCaller
```

```
gatk HaplotypeCaller -R $DataDir/genome.fa -I  
Aligned.sortedByCoord.RG.MD.SplitN.out.bam \  
-stand-call-conf 20 --dont-use-soft-clipped-bases \  
-O variation.vcf.gz
```

HaplotypeCallerによる多型検出

```
gatk VariantFiltration -R $DataDir/genome.fa -V variation.vcf.gz \  
-window 20 -cluster 3 \  
--filter-name "QD" --filter "QD < 2.0" \  
--filter-name "FS" --filter "FS > 60.0" \  
-O variation.filter.vcf.gz
```

複数の条件での
多型のフィルタ
リング

フィルタリングで指定しているQDやFSについては以下のページが参考になる

<https://software.broadinstitute.org/gatk/documentation/article?id=11069>

Step4. GATK HaplotypeCallerによる多型検出



シェルスクリプトの実行 (実行時間：約2分)

```
$ bash ./step4_GATK.sh
```

作成された多型情報ファイルの確認

```
$ ls -lhtr
合計 562M
...
-rw-rw-r-- 1 guest01 guest01 11K 9月 17 12:23 variation.vcf.gz
-rw-rw-r-- 1 guest01 guest01 839 9月 17 12:23 variation.vcf.gz.tbi
-rw-rw-r-- 1 guest01 guest01 11K 9月 17 12:24 variation.filter.vcf.gz
-rw-rw-r-- 1 guest01 guest01 840 9月 17 12:24 variation.filter.vcf.gz.tbi
```

基本的にはゲノムリシーケンスデータによる変異検出と同じで、多型情報はVCFファイル（gz圧縮）として出力されている。

検出された多型情報 (variation.filter.vcf.gz) の確認



VCF形式で記載された多型情報

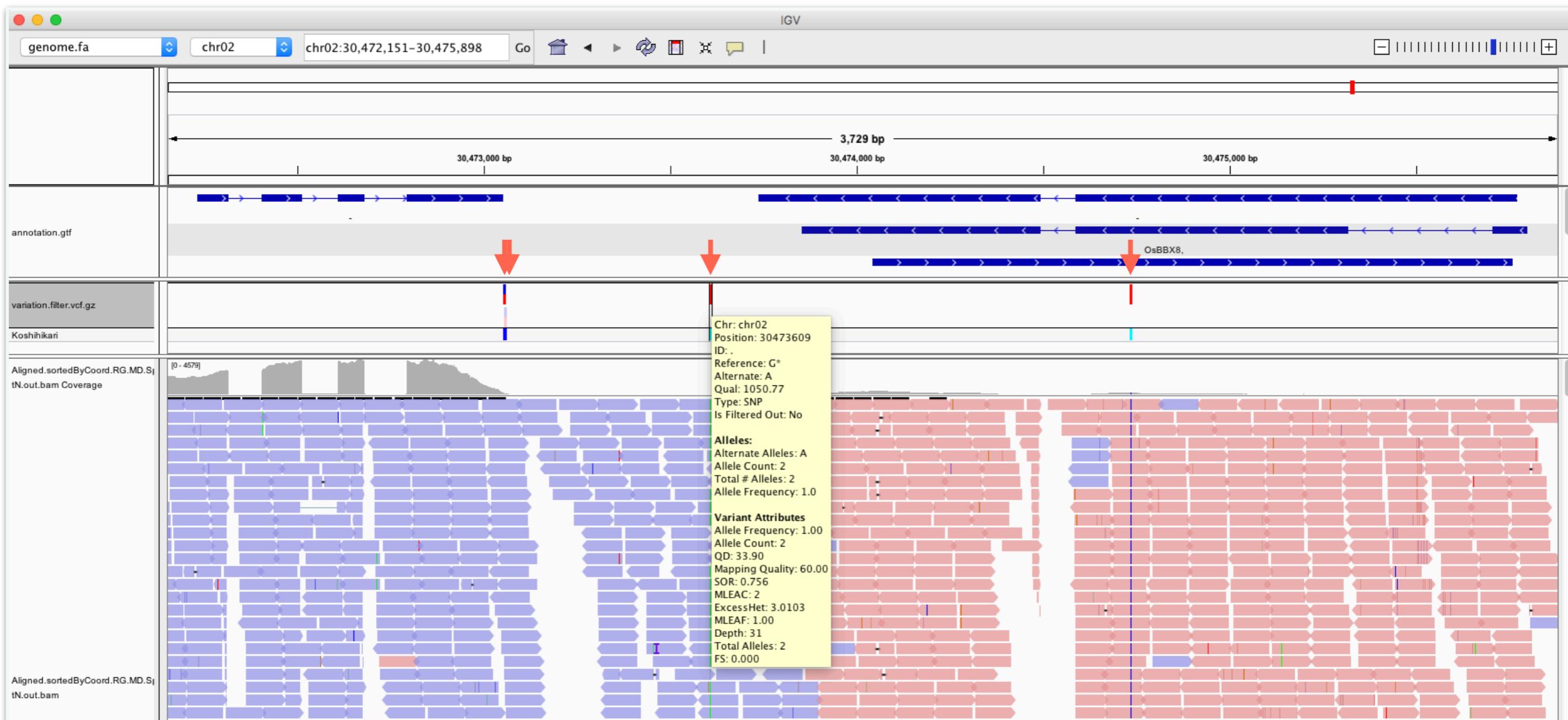
```
> less variation.filter.vcf.gz
```

```
...
1 chr02 30473056 . T A 419.77 PASS
AC=1;AF=0.500;AN=2;BaseQRankSum=-0.312;DP=210;ExcessHet=3.0103;FS=16.873;MLEAC=1;MLEAF=0.
500;MQ=60.00;MQRankSum=0.000;QD=2.00;ReadPosRankSum=-6.091;SOR=2.491 GT:AD:DP:GQ:PL
0/1:173,37:210:99:448,0,9241
2 chr02 30473059 . T A 53.77 QD
AC=1;AF=0.500;AN=2;BaseQRankSum=1.180;DP=164;ExcessHet=3.0103;FS=13.389;MLEAC=1;MLEAF=0.5
00;MQ=60.00;MQRankSum=0.000;QD=0.33;ReadPosRankSum=-5.911;SOR=3.304 GT:AD:DP:GQ:PL
0/1:147,17:164:82:82,0,9231
3 chr02 30473609 . G A 1050.77 PASS
AC=2;AF=1.00;AN=2;DP=31;ExcessHet=3.0103;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=60.00;QD=33.90;S0
R=0.756 GT:AD:DP:GQ:PL 1/1:0,31:31:93:1079,93,0
4 chr02 30474736 . T C 7195.77 PASS
AC=2;AF=1.00;AN=2;DP=242;ExcessHet=3.0103;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=60.00;QD=29.73;S
OR=0.815 GT:AD:DP:GQ:PL 1/1:0,242:242:99:7224,726,0
...
```

PASS:フィルタリングをパスした多型

IGVでアラインメントと多型情報の可視化

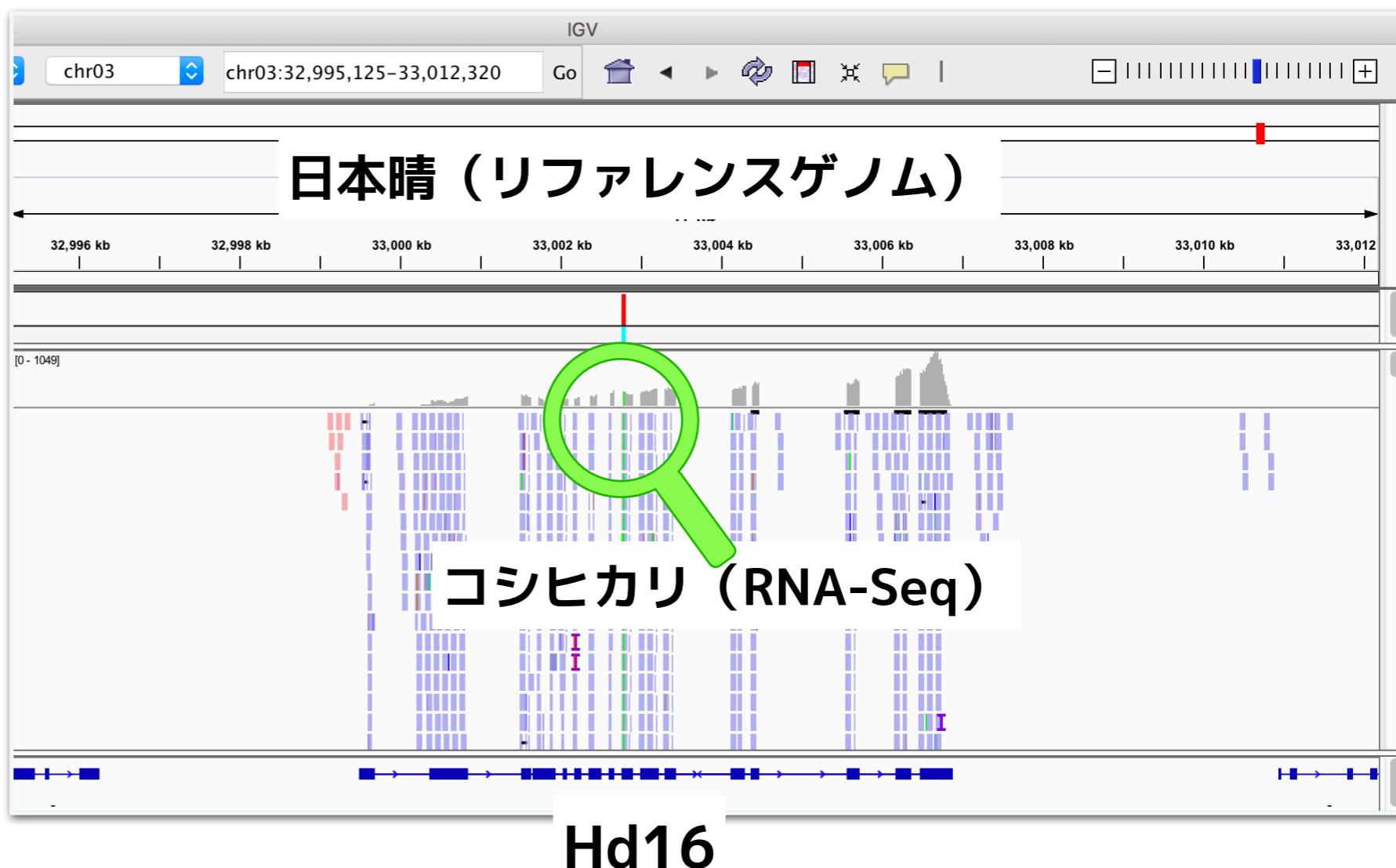
アラインメント情報 (Aligned.sortedByCoord.RG.MD.SplitN.out.bam) と 多型情報 (variation.filter.vcf.gz) は IGV で閲覧することができる。



↓ : 前のスライドで示したVCFファイル中の4つの多型

既知の品種間多型は検出されているか？

Hd16遺伝子上にある、日本晴とコシヒカリの開花期の違いに関わる既知のFNP*



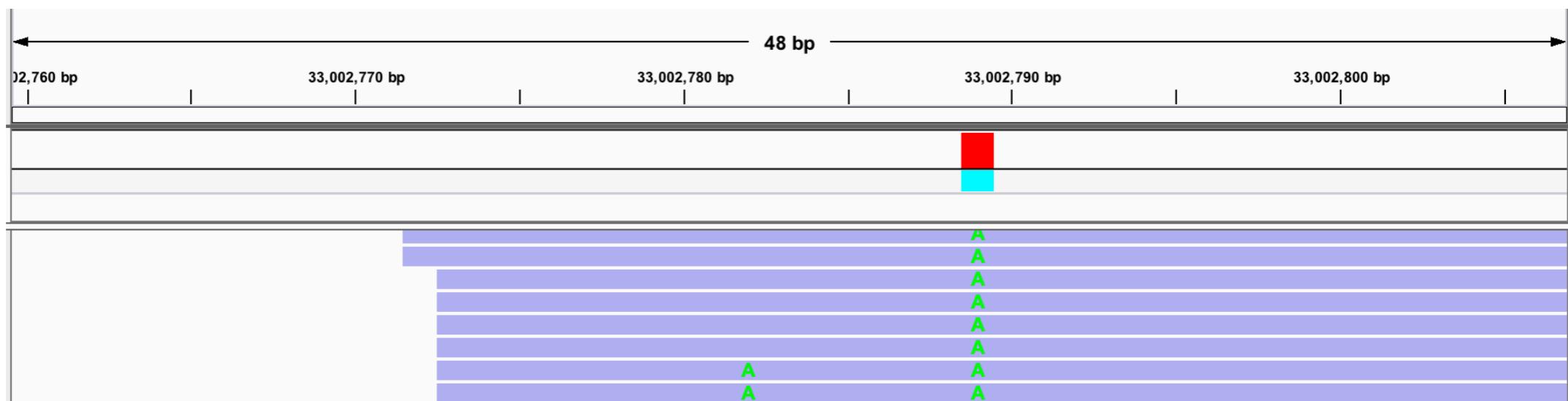
*FNP=Functional Nucleotide Polymorphism

ローカルPC上(workshop/RNA-Seq/varcall)にゲノムワイドな解析結果から3番染色体の情報を抜き出したアラインメントとVCFファイルがあるのでIGVで開いて見てみよう。

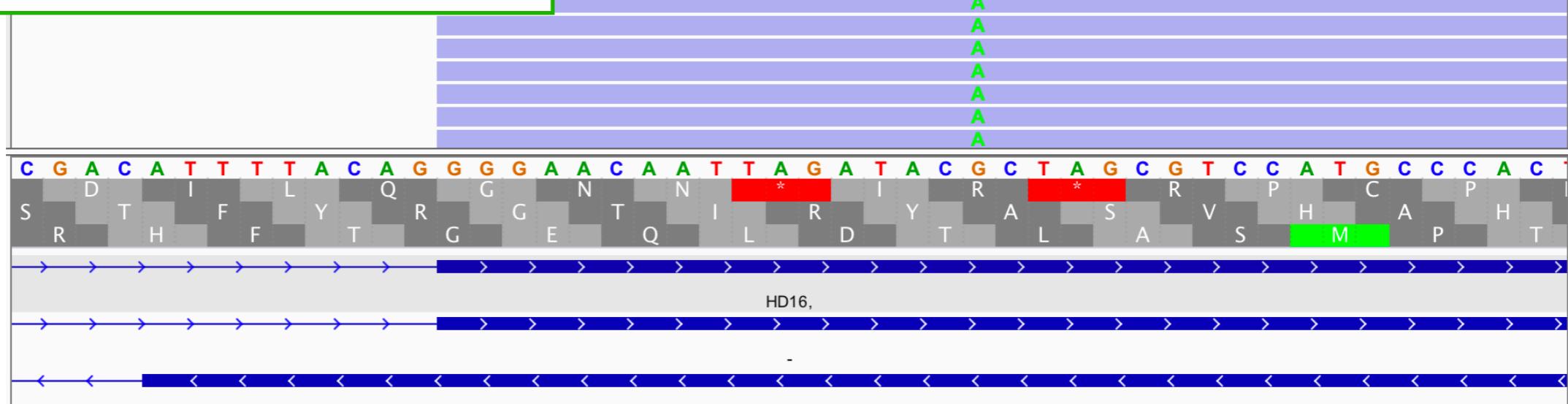
アラインメントとVCFファイルの双方で確認

VCFファイル中の多型情報

```
chr03    33002789      .      G      A      3117.77 PASS
AC=2;AF=1.00;AN=2;DP=92;ExcessHet=3.0103;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=60.00;
QD=33.89;SOR=0.782          GT:AD:DP:GQ:PL  1/1:0,92:92:99:3146,276,0
```



BAMファイル中のアラインメント 情報をIGVで可視化



日本晴 (G) とコシヒカリ (A) の多型が確認された