TDS Facility

# Analysis of flow cytometry data with R
# Training for life scientists

**João Lourenço, Tania Wyss & Nadine Fournier**

Translational Data Science – Facility

SIB Swiss Institute of Bioinformatics

# Outline & Schedule

## Day 3

01 **Importing data from Excel and arranging plots in a grid**

02 **Differential state analysis using a paired design**

03 **Normalization / batch correction**

04 **Working with gated data in R**

# Outline & Schedule

## Day 4


**Generate HTML or PDF reports**

Examples and exercises are integrated in the chapters

01

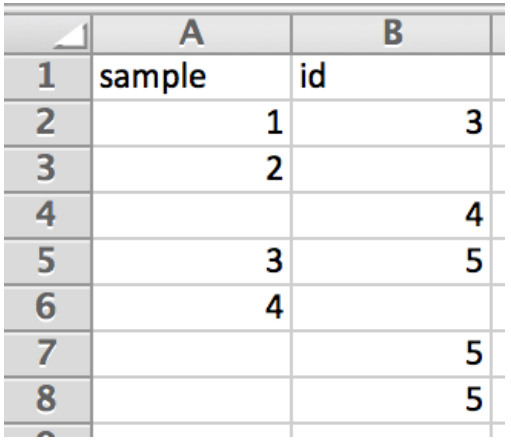
# A. Importing data from Excel
# B. Arranging plots in grids

# Importing data from an Excel file

Package xlsx

https://cran.r-project.org/web/packages/xlsx/index.html

```
> data <- read.xlsx2(file = "excelfile.xlsx",
                     sheetName = "Sheet1")
```



| | A | B |
|---|---|---|
| 1 | sample | id |
| 2 | 1 | 3 |
| 3 | 2 | |
| 4 | | 4 |
| 5 | 3 | 5 |
| 6 | 4 | |
| 7 | | 5 |
| 8 | | 5 |

The function r*ead.xlsx2* considers the empty cells as "" character.

```
> str(data)
'data.frame':   7 obs. of  2 variables:
 $ sample: chr  "1" "2" "" "3" ...
 $ id    : chr  "3" "" "4" "5" ...
```

The function *read.xlsx* considers the empty cells as NA

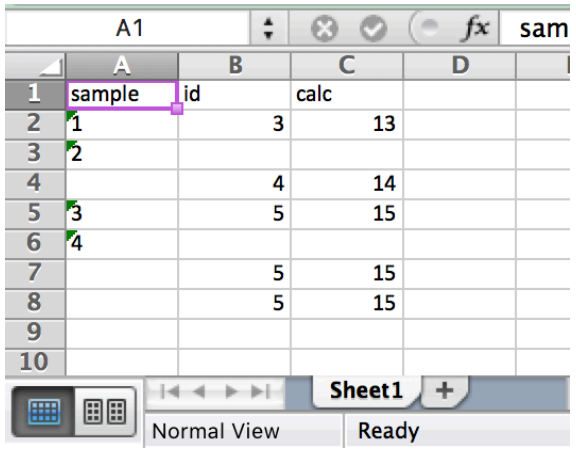
```
> str(data)
'data.frame':   7 obs. of  2 variables:
 $ sample: num  1 2 NA 3 4 NA NA
 $ id    : num  3 NA 4 5 NA 5 5
```

# Exporting data to an Excel file

Advantage: no risk of gene name conversion to date like with csv files

Package *xlsx* can be used to export a data.frame after manipulating it within R:

```
> data$id<-as.numeric(data$id)
> data$calc<-data$id+10
> write.xlsx2(x=data,
                  file = "excelfile2.xlsx",
                  row.names = FALSE,
                 SheetName = "Sheet1")
```



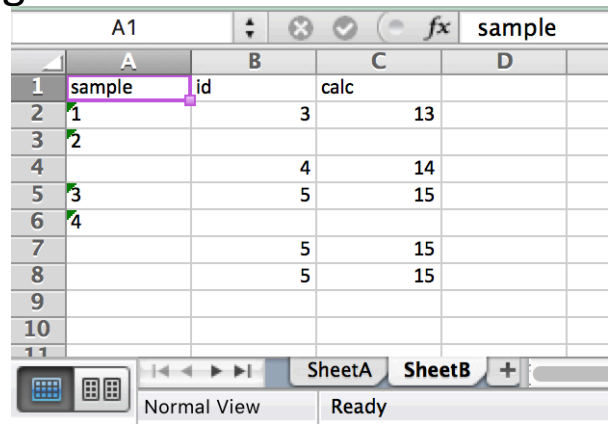

Another package: *openxlsx*

https://cran.r-project.org/web/packages/openxlsx/index.html

Allows to generate several separate sheets providing a named list of data.frame objects:

```
> openxlsx::write.xlsx(x = list(SheetA=data[ , c(1,2)],
                               SheetB=data),
                     file = "excelfile3.xlsx")
```

Doesn't export rownames by default.



# Arranging ggplot2 plots in a grid

Can be useful to compare distribution of values before and after transformation, or to compare a UMAP with cells colored according to different markers.

Package *cowplot*

https://cran.r-project.org/web/packages/cowplot/index.html

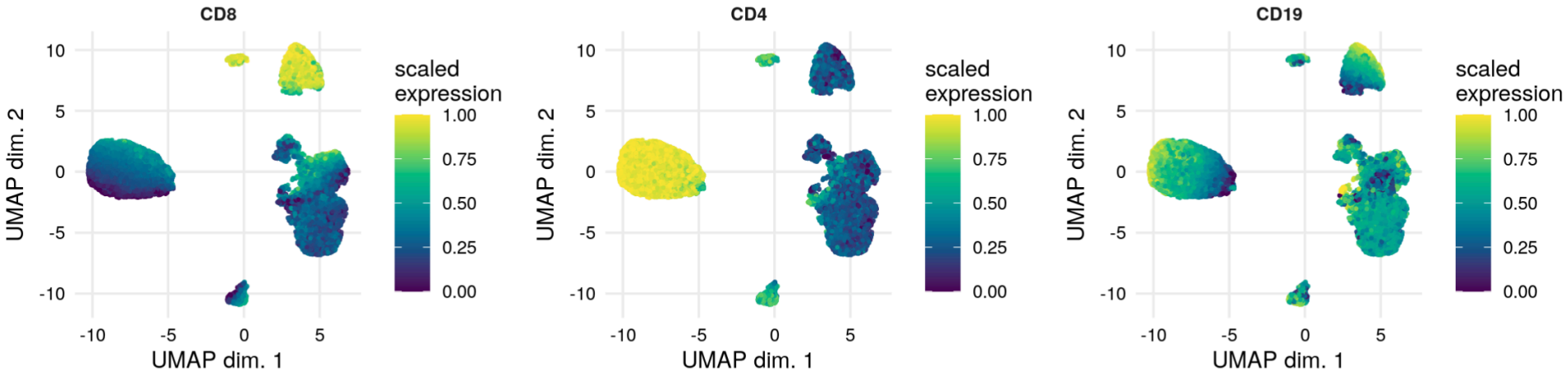
First save each ggplot2 plot to an object:

```
> p1 <- plotDR(sce,  dr =  "UMAP",
        assay = "exprs",  color_by="CD8" )
> p2 <- plotDR(sce,   dr =  "UMAP",
        assay = "exprs",   color_by="CD4")
> p3<-plotDR(sce,   dr =  "UMAP",
        assay = "exprs",   color_by="CD19")
```

# Arranging ggplot2 plots in a grid

Use the *plot_grid* function to arrange them in a grid, e.g. all on the same row:

```
> plot_grid(p1, p2, p3, nrow = 1)
```



02

# Differential state analysis with a paired design

# Differential state (DS) analysis with paired design

## Differential expression of cell state markers within clusters

```
> res_DS <- diffcyt(sce_PBMC,
                    clustering_to_use = ”final_annotation",
                    analysis_type = "DS",
                    method_DS = "diffcyt-DS-limma",
                    design = design,
                    contrast = contrast,
                    block_id = patient_id)
```

Methods for DS: uses the limma package

Vector of patient IDs

# How to create a vector of patient IDs from the fcs file names?

```
> ei(sce)$sample_id
 # [1] 0BF51C_0.fcs  0BF51C_14.fcs 0BF51C_7.fcs  0E1F8E_0.fcs  0E1F8E_14.fcs
 # [6] 0E1F8E_7.fcs  180E1A_0.fcs  180E1A_14.fcs 180E1A_7.fcs  1A9B20_0.fcs
# [11] 1A9B20_14.fcs 1A9B20_7.fcs  61BBAD_0.fcs  61BBAD_14.fcs 61BBAD_7.fcs
```

**Using gsub()** :
This function replaces all characters having a same pattern with other characters.
We can replace the *_0.fcs*, *_14.fcs* and *_7.fcs* extensions by an empty character:

Create a vector of patient IDs for block design:

```
> patient_id <- ei(sce)$sample_id
```

Use gsub to replace the extensions for the matching elements within the vector:

```
> patient_id <- gsub("_0.fcs", "", patient_id)
> patient_id <- gsub("_14.fcs", "", patient_id)
> patient_id <- gsub("_7.fcs", "", patient_id)
```

```
> patient_id
[1] "0BF51C" "0BF51C" "0BF51C" "0E1F8E" "0E1F8E" "0E1F8E" "180E1A" "180E1A"
 [9] "180E1A" "1A9B20" "1A9B20" "1A9B20" "61BBAD" "61BBAD" "61BBAD"
```

# Differential state analysis

Extract results table - same method as for unpaired design:

```
> tbl_DS <- rowData(res_DS$res)
> tbl_DS
```

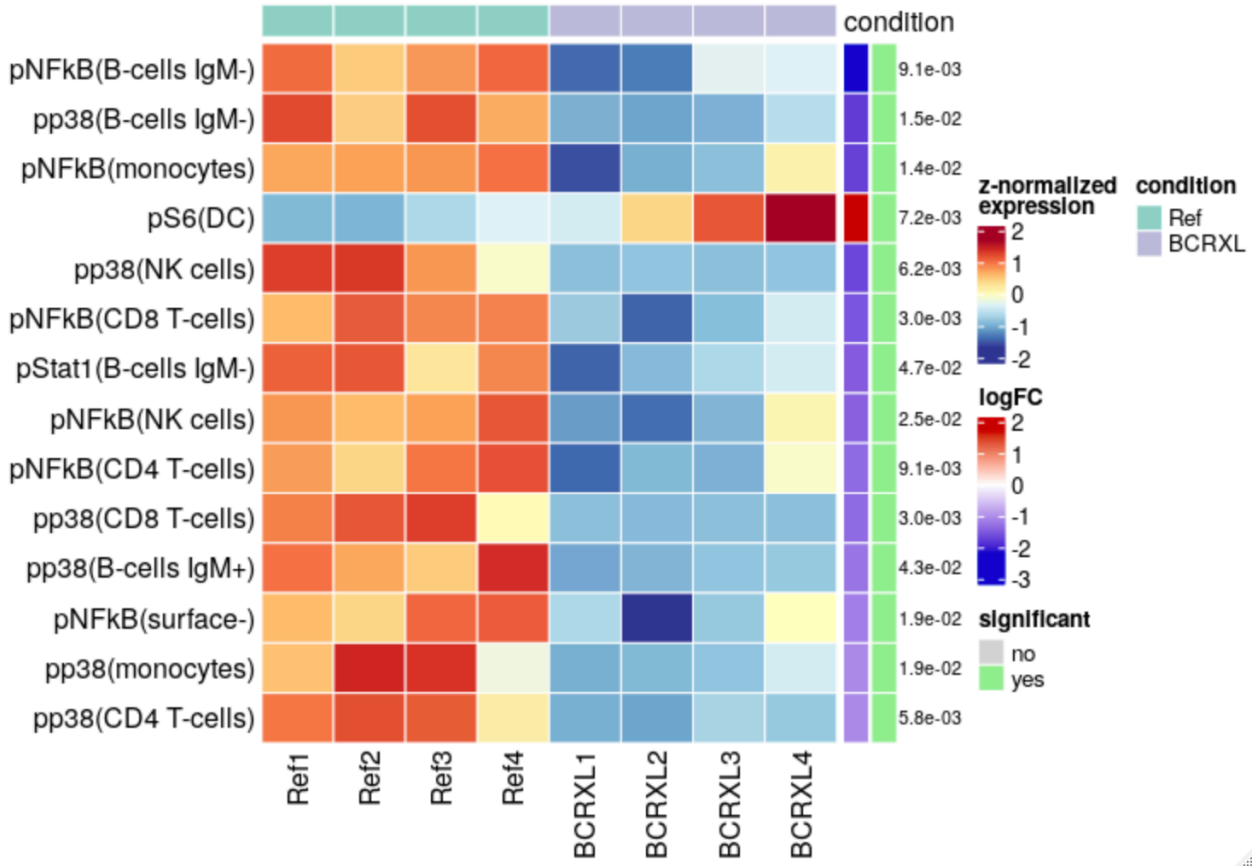
```
DataFrame with 20 rows and 4 columns
             cluster_id marker_id       p_val     p_adj
               <factor>  <factor>   <numeric> <numeric>
Other       Other          CD38   0.000883343  0.128085
Other       Other          TCR gd 0.006609759  0.479208
CD4 T cells CD4 T cells    CD141  0.027918113  0.481051
CD8 T cells CD8 T cells    CCR5   0.026749683  0.481051
Monocytes   Monocytes      CCR5   0.027850828  0.481051
...                  ...       ...         ...       ...
CD4 T cells CD4 T cells     CD16    0.0703599  0.534035
CD8 T cells CD8 T cells     CD16    0.0629339  0.534035
Other       Other           CD16    0.0688253  0.534035
CD8 T cells CD8 T cells     IgD     0.0704875  0.534035
Other       Other           CD161   0.0684019  0.534035
```

# Differential state analysis

Plot results for all markers

Sorts results by absolute value of logFoldChange

```
> plotDiffHeatmap(sce_PBMC, tbl_DS, all=T , sort_by = "lfc", col_anno ="condition")
```



# Let’s practice – 7 bis

In this exercise we will test if markers were differentially expressed between two time points (D14 compared to D0), **using a paired design**

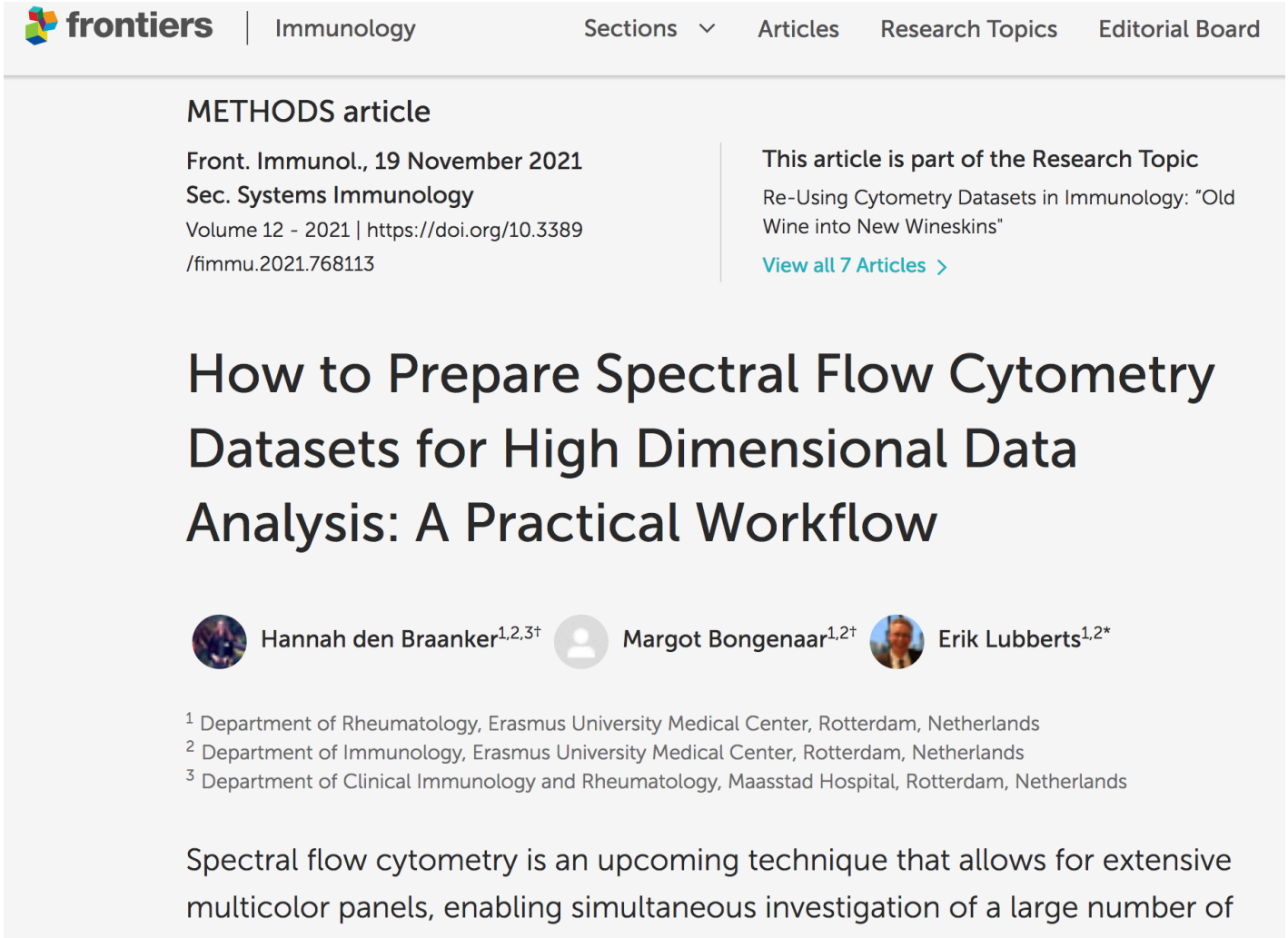
Create a new script in which you will

1) Load the sce object from exercise number 6 ("sce_annotated.RData").
2) Create a vector of patient IDs from the fcs file names using gsub()
3) Set up the design and contrast matrices.
4) Test for differences in marker expression between D14 and D0, including argument block_id
5) View table of results

# 03 Normalization / batch correction

# Spectral flow cytometry analysis workflow

- Workflow based on



frontiers | Immunology Sections Articles Research Topics Editorial Board

METHODS article

Front. Immunol., 19 November 2021
Sec. Systems Immunology
Volume 12 - 2021 | https://doi.org/10.3389/fimmu.2021.768113

This article is part of the Research Topic
Re-Using Cytometry Datasets in Immunology: "Old Wine into New Wineskins"
View all 7 Articles

How to Prepare Spectral Flow Cytometry Datasets for High Dimensional Data Analysis: A Practical Workflow

Hannah den Braanker[1,2,3†] Margot Bongenaar[1,2†] Erik Lubberts[1,2*]

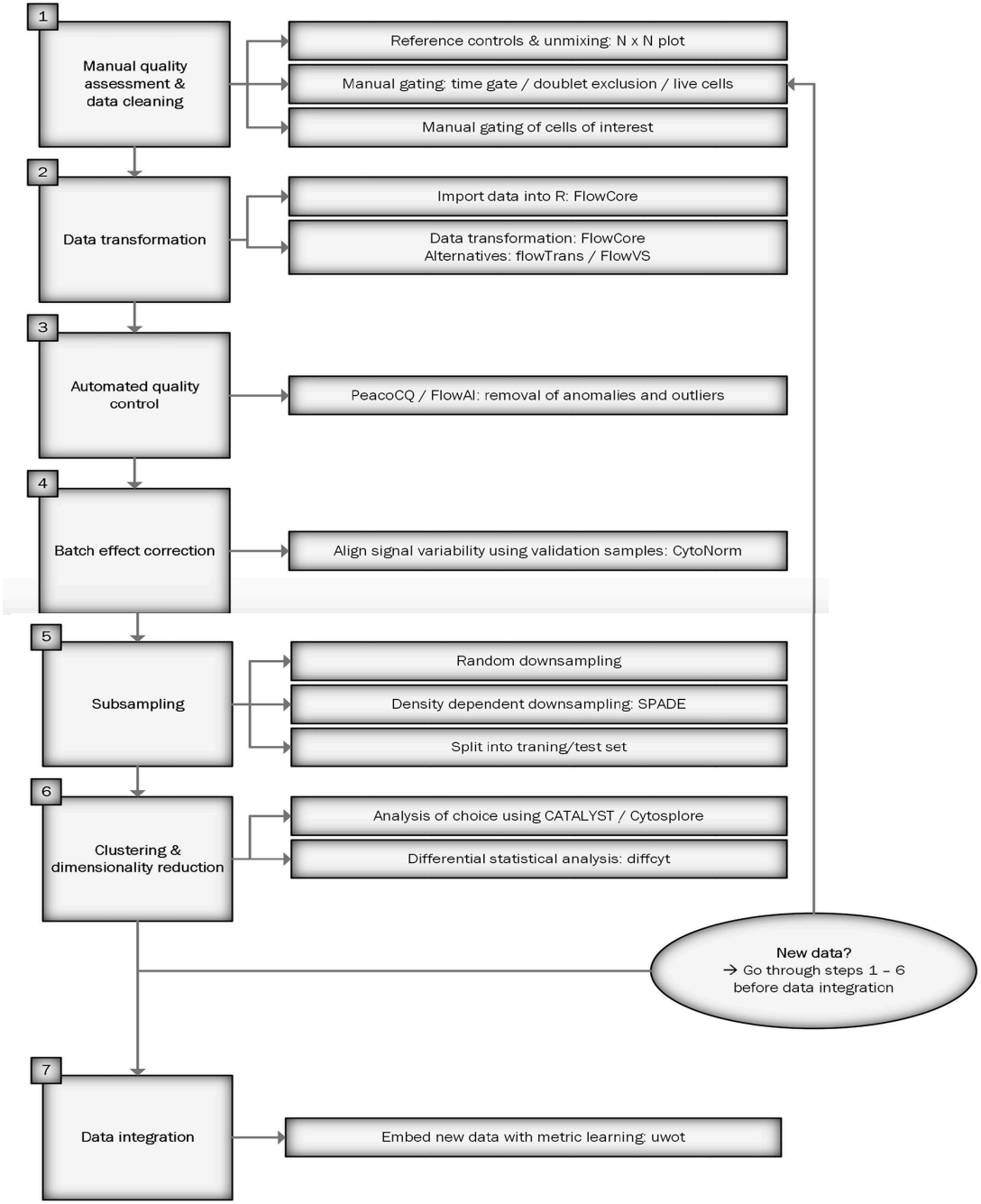
[1] Department of Rheumatology, Erasmus University Medical Center, Rotterdam, Netherlands
[2] Department of Immunology, Erasmus University Medical Center, Rotterdam, Netherlands
[3] Department of Clinical Immunology and Rheumatology, Maasstad Hospital, Rotterdam, Netherlands

Spectral flow cytometry is an upcoming technique that allows for extensive multicolor panels, enabling simultaneous investigation of a large number of

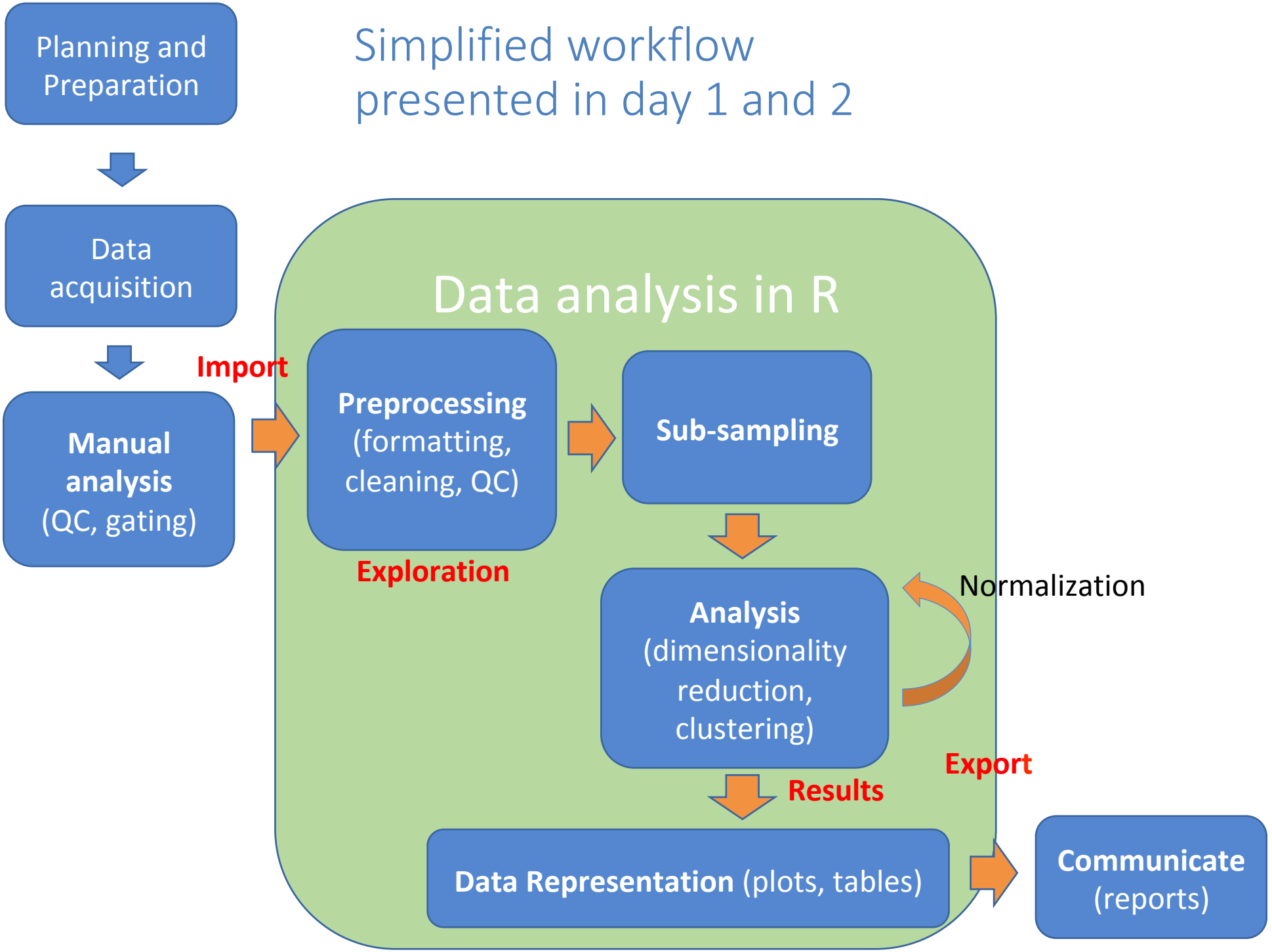
Provide R code to perform the proposed workflow

https://doi.org/10.3389/fimmu.2021.768113

# Suggested workflow: Figure 1, den Braanker et al

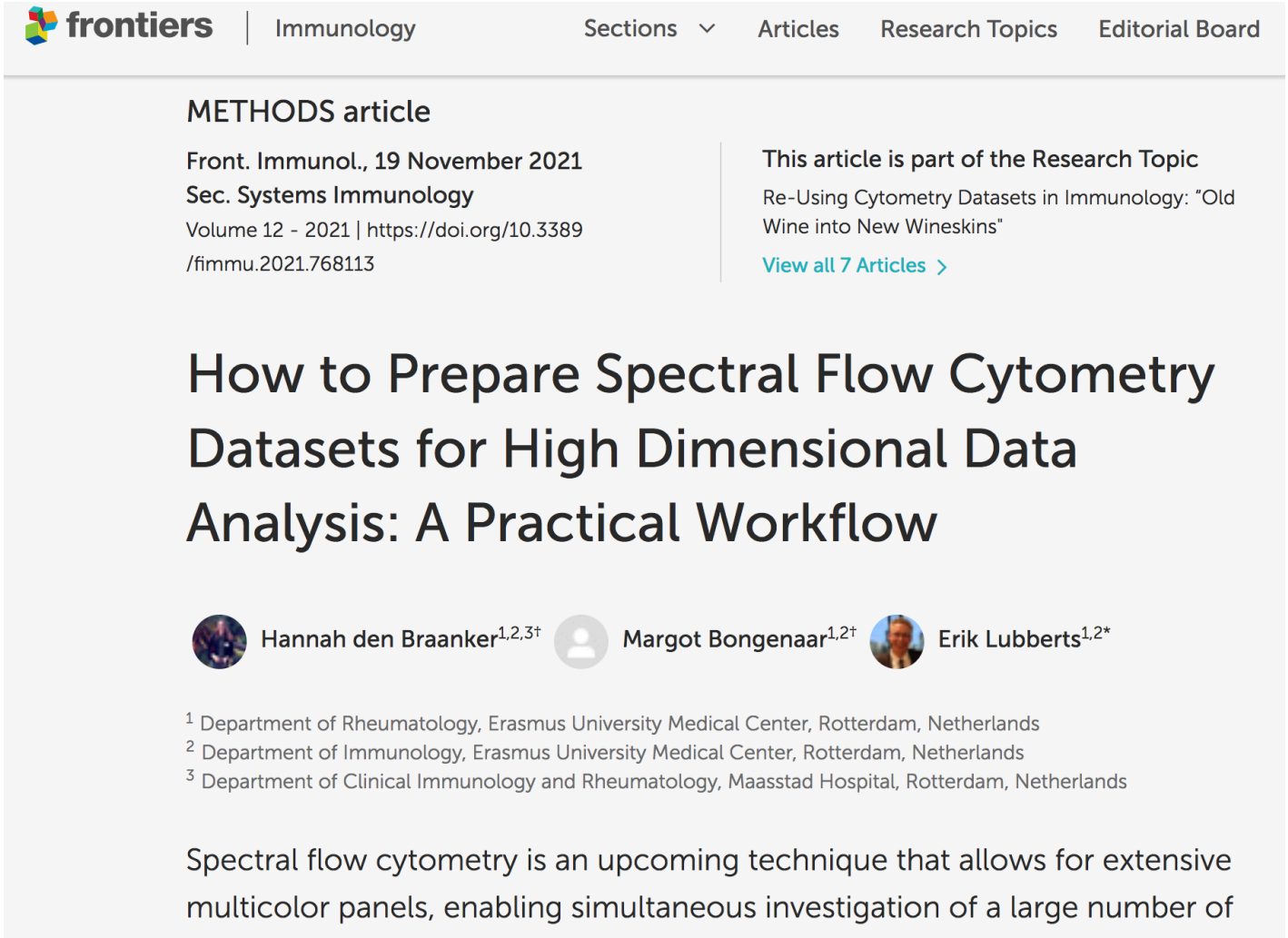


# Simplified workflow presented in day 1 and 2



# Spectral flow cytometry analysis workflow

- Workflow based on



frontiers | Immunology Sections Articles Research Topics Editorial Board

METHODS article

Front. Immunol., 19 November 2021
Sec. Systems Immunology
Volume 12 - 2021 | https://doi.org/10.3389/fimmu.2021.768113

This article is part of the Research Topic
Re-Using Cytometry Datasets in Immunology: "Old Wine into New Wineskins"
View all 7 Articles >

## How to Prepare Spectral Flow Cytometry Datasets for High Dimensional Data Analysis: A Practical Workflow

Hannah den Braanker[1,2,3†] Margot Bongenaar[1,2†] Erik Lubberts[1,2*]

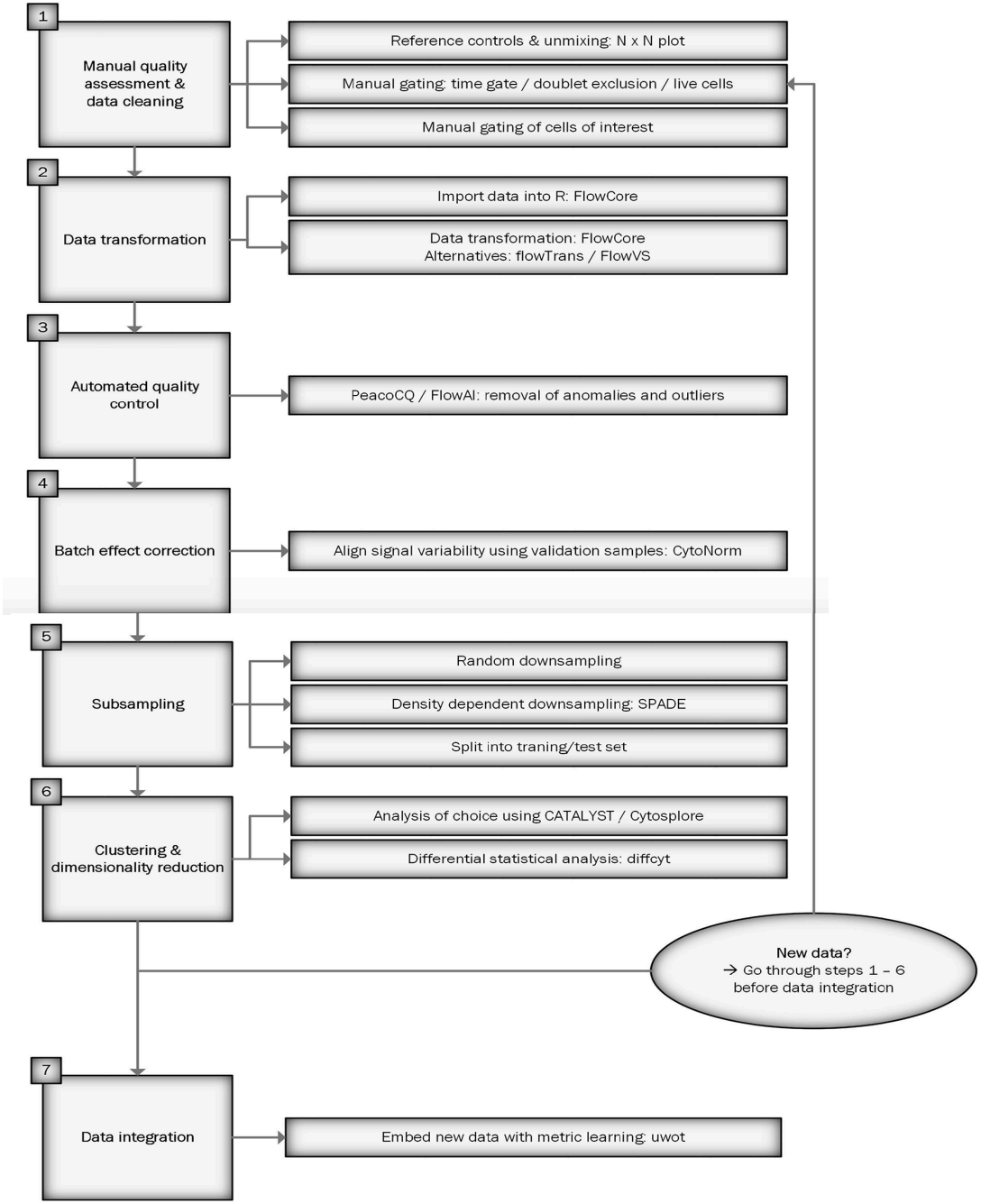
[1] Department of Rheumatology, Erasmus University Medical Center, Rotterdam, Netherlands
[2] Department of Immunology, Erasmus University Medical Center, Rotterdam, Netherlands
[3] Department of Clinical Immunology and Rheumatology, Maasstad Hospital, Rotterdam, Netherlands

Spectral flow cytometry is an upcoming technique that allows for extensive multicolor panels, enabling simultaneous investigation of a large number of

Provide R code to perform the proposed workflow

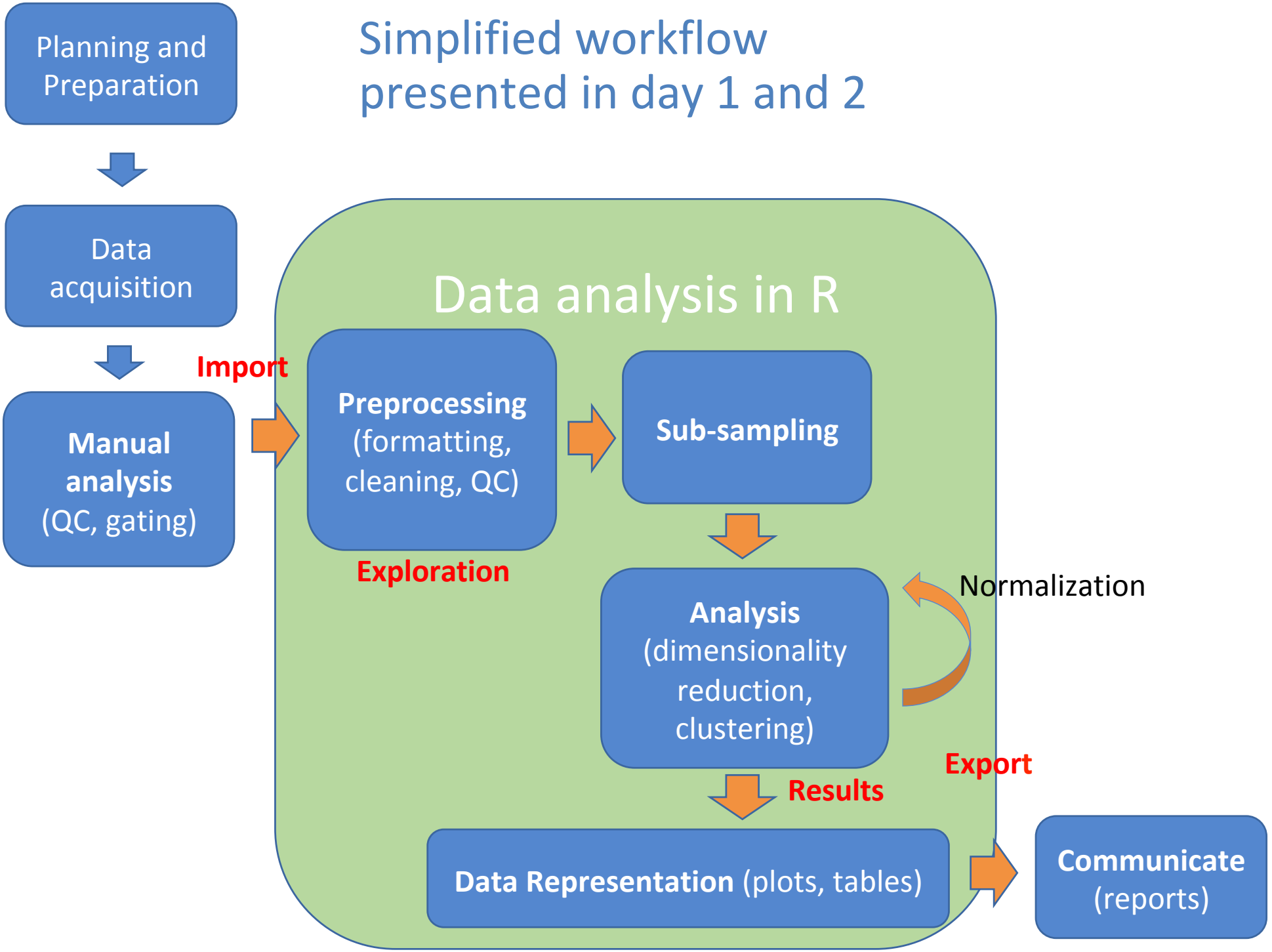
https://doi.org/10.3389/fimmu.2021.768113

# Suggested workflow: Figure 1, den Braanker et al




https://doi.org/10.3389/fimmu.2021.768113

# Simplified workflow presented in day 1 and 2



# CytoNorm

Available on github: https://github.com/saeyslab/CytoNorm
van Gassen et al, 2020 https://onlinelibrary.wiley.com/doi/epdf/10.1002/cyto.a.23904

Install using devtools package:

```
> library(devtools)
> install_github('saeyslab/CytoNorm')
```

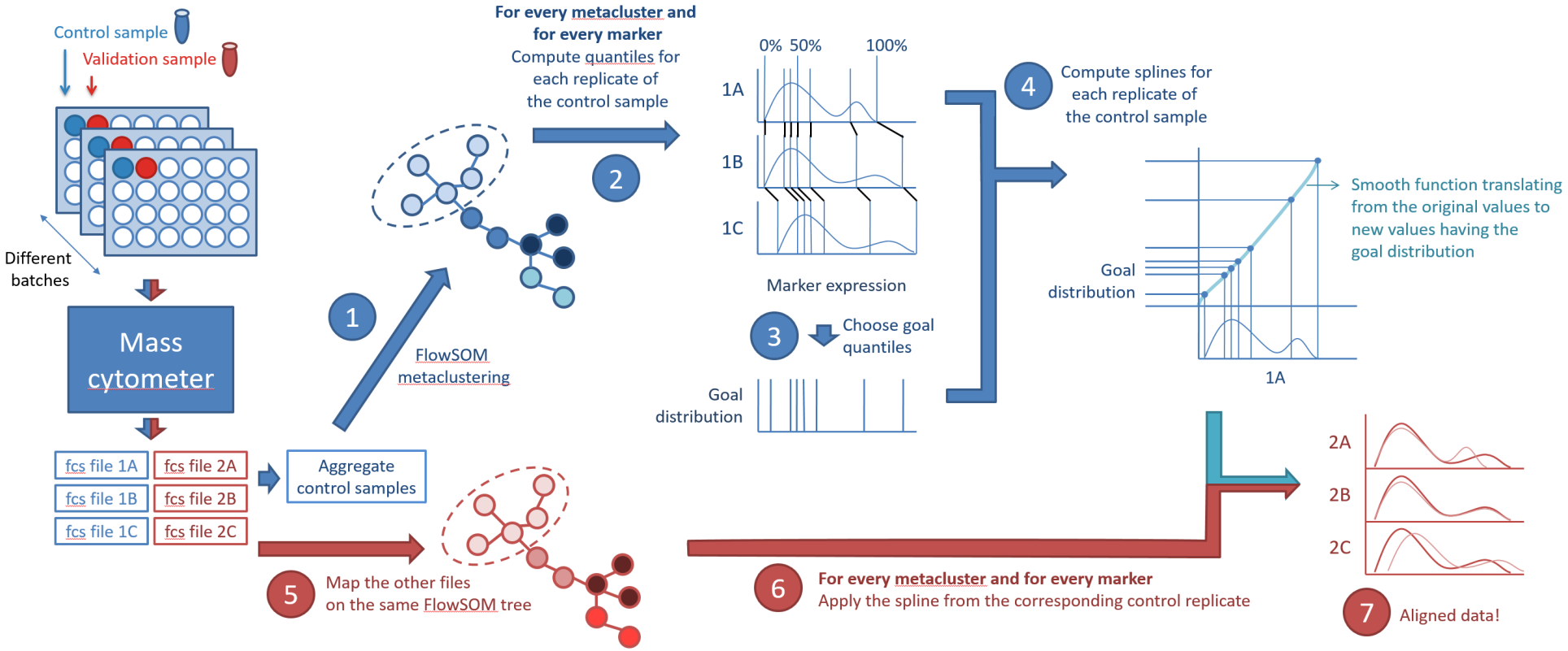
**Why do we need to normalize?**

Shifts that occur because of batch can have effects on DR for example.

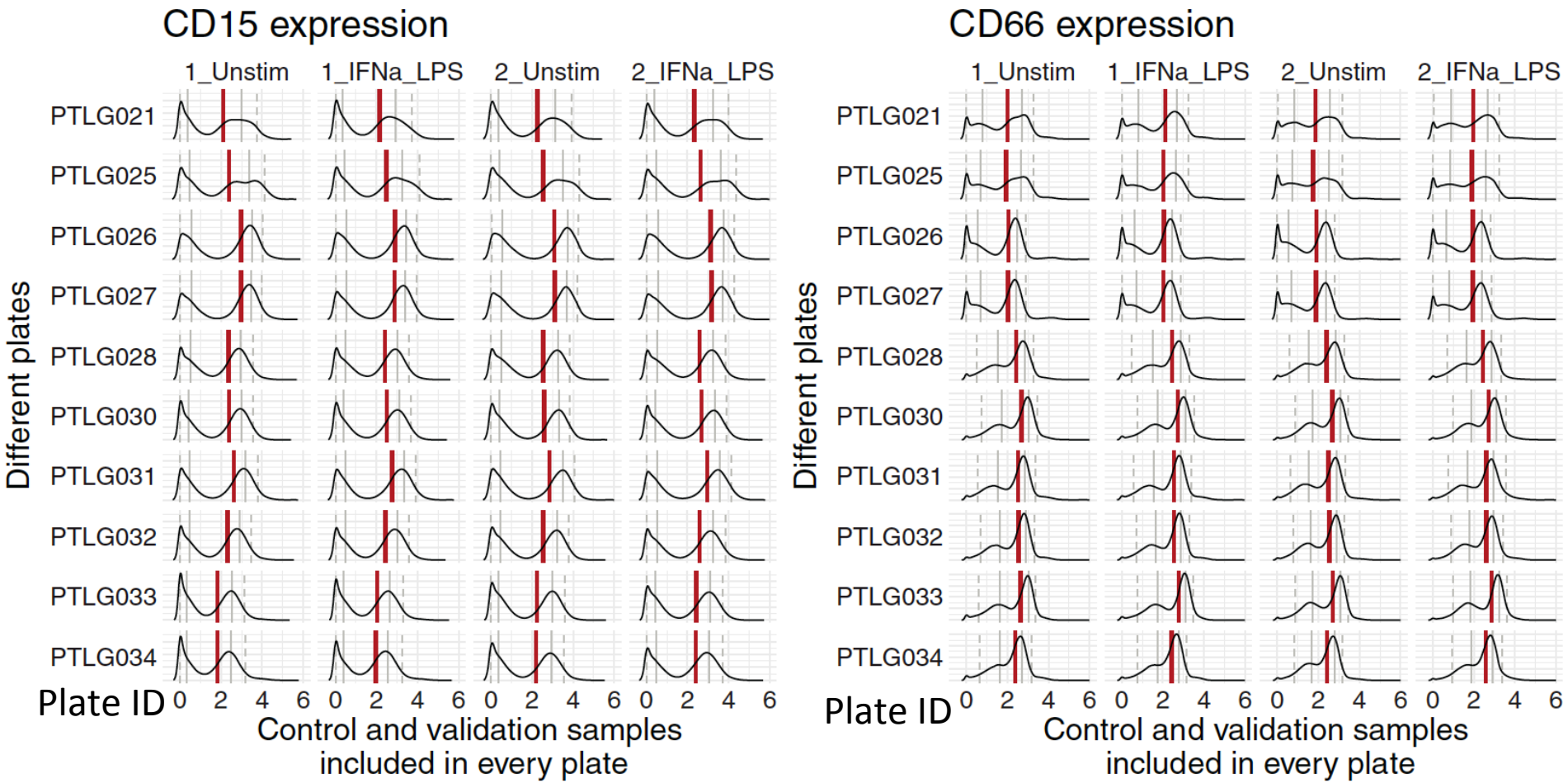
Computational methods can align the distribution of markers across samples.

Normalization methods without a reference sample can hide biologically relevant differences in cell subsets. Different cell types are impacted differently by batch effects (Finak et al, 2014, Cytom. Part A)

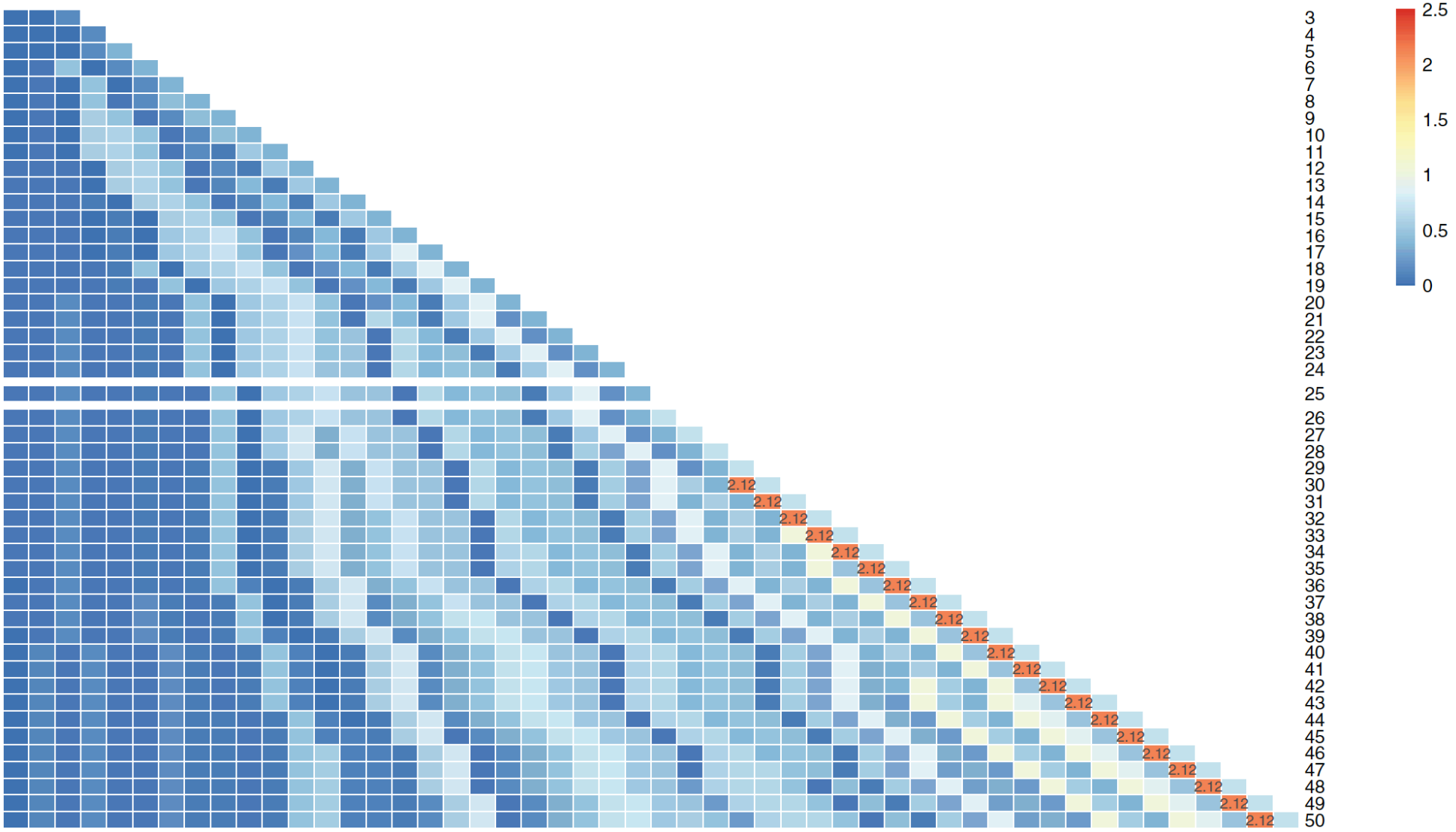
# CytoNorm – algorithm overview



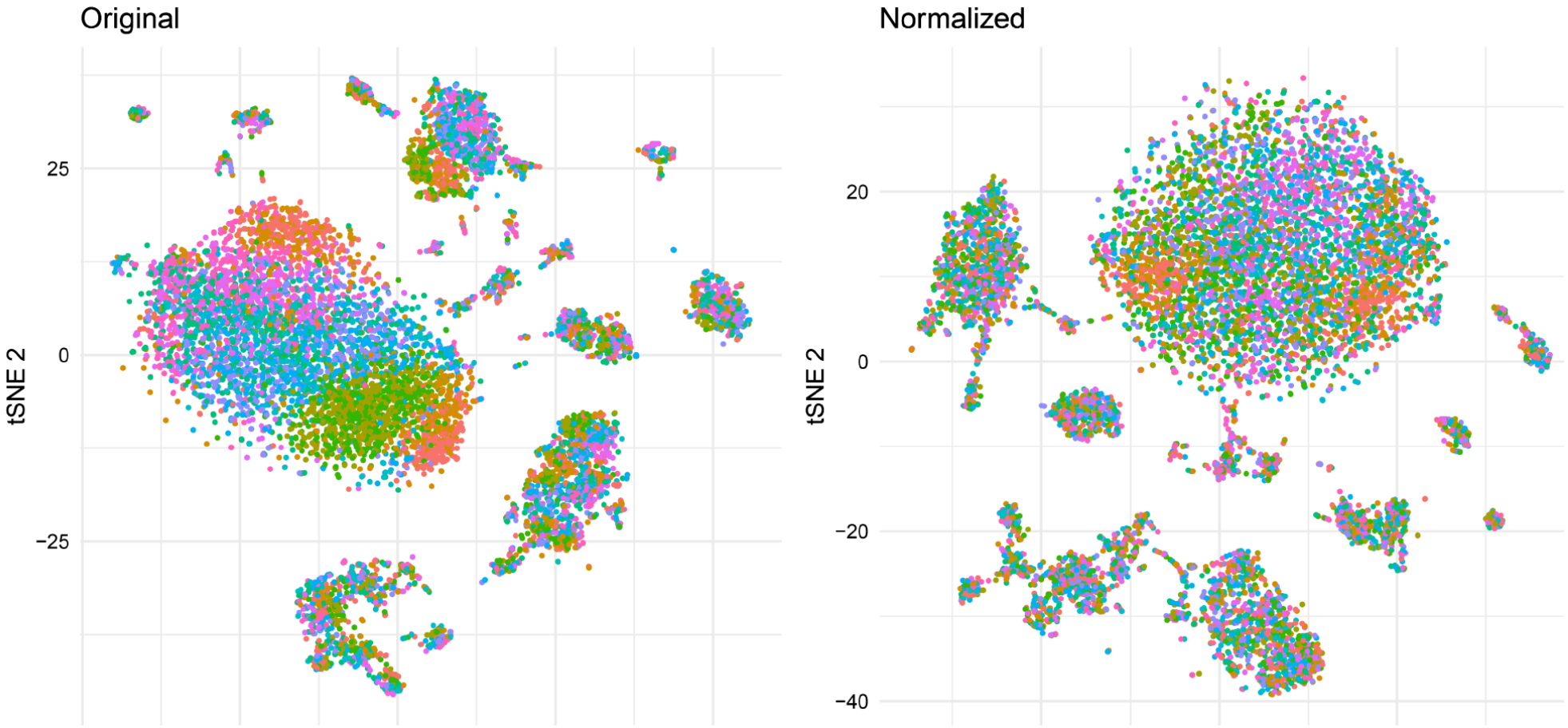
# An example of shifts across plates



# Assumption: clusters are not affected by batch



# Normalization enhances DR plot




1,000 randomly sampled cells for each of 10 samples

# Functions of CytoNorm

Functions use flowSet objects containing transformed data (eg arcsinh with fixed cofactor).

Generate pre-clustering with flowSOM:

```
> fsom <- prepareFlowSOM(train_files,
                         colsToUse = markerstotransf,
                         transformList = NULL,
                         FlowSOM.params = list(xdim=10,
                                               ydim=10,
                                               nClus=20, scale=FALSE))
```

flowSet with technical replicates of a single sample

Markers to use for clustering (eg. "type" marker_class)

FlowSOM parameters: number of grids and of metaclusters

Test for coefficient of variation within clusters:

```
> cvs <- CytoNorm::testCV(fsom, cluster_values = c(5,10,15,20), plot=TRUE)
> range(cvs$cvs$`20`) # 0.05758965 1.43114512
```

If the clusters are impacted by batch effects, CV values of >1.5 - 2 will occur, then you can choose to put FlowSOM.params to NULL and skip clustering.

# Functions of CytoNorm

Train the model, i.e. evaluate quantiles from technical replicates:

```
model <- CytoNorm.train(files = train_files,
                        labels = labels_train,
                        channels = markerstotransf,
                        transformList = NULL,
                        FlowSOM.params = list(nCells = 6000,
                                              xdim = 10,
                                              ydim = 10,
                                              nClus = 5,
                                              scale = FALSE),
                        normMethod.train = QuantileNorm.train,
                        normParams = list(nQ = 101,
                                          goal = "mean"),
                        seed = 1,
                        verbose = TRUE)
```

flowSet with technical replicates of a single sample

Vector of batch ID labels for each technical replicate within the train flowSet

Markers to use for clustering (eg. "type" marker_class)

FlowSOM parameters: number of grids and of metaclusters

Compute quantiles to describe the distribution of the data, and infer spline functions to equalize these distributions over the files.

# Functions of CytoNorm

Normalize the rest of the samples:

```
> CytoNorm.normalize(model = model,
                     files = validation_files,
                     labels = label_norm,
                     transformList = NULL,
                     transformList.reverse = NULL,
                     normMethod.normalize = QuantileNorm.normalize,
                     outputDir = "course_datasets/FR_FCM_Z4KT/Normalized",
                     prefix = "Norm_",
                     clean = TRUE,
                     verbose = TRUE)
```

Model with quantiles obtained using CytoNorm.train()

flowSet with the rest of the samples

Vector of batch ID labels for each sample of the samples to be normalized

Compute quantiles and infer spline functions to equalize these distributions over the files.

Output folder where new fcs files will be created with prefix "Norm_

# Before the exercise: the grep() function

It allows to search for a pattern of characters within a vector:

```
> grep("pattern", myvector)
```

Will return numbers of elements within the vector that correspond to that pattern:

```
> myvector <- c("abc", "xyz", "abcd", "abxyz", "cdy")
> grep("abc", myvector)
# [1] 1 3
```

Use it on flowSet sampleNames to split a flowSet based on sample ID:

```
> train_files <- fcs_transform[grep("REU271", sampleNames(fcs_transform))]

> validation_files <- fcs_transform[-c(grep("REU271", sampleNames(fcs_transform)))]
```

# Let's practice – 8

In this exercise we will perform normalization (i.e. batch correction) for fcs files provided in accession **FR_FCM_Z4KT** of the FlowRepository.

Create a new script in which you will:

1) Create a flowSet called fcs_data of all samples within the */course_dataset/FR_FCM_Z4KT* folder
2) Generate a panel data.frame using colnames(fcs_data) antigen names extracted with pData(parameters(fcs_data[[1]]))$desc. Create a new column called marker_class that will contain the type of markers: all that are not NA should be labeled as "type", except PD-1 which should be labeled as "state". Make sure that the antigen "Zombie UV" is labeled as "none" and not as "type". Save the panel to an Excel file using write.xlsx2().
3) Transform the data: extract a vector from the panel data.frame which are the channels to be transformed, which are not labeled with "none". Perform asinh transformation with a cofactor of 3000 for all channels to be transformed, using transFlowVS() from the flowVS package.
4) Split the flowSet resulting from transformation into a *training* flowSet containing all flowFrames from the sample "REU271", and a flowSet with the rest of the flowFrames not corresponding to sample "REU271". Use the grep() function on the sampleNames of the flowSet.

# Let’s practice – 8 - continued

5) Perform pre-clustering with flowSOM with function prepareFlowSOM(), providing the flowSet with the training flowFrames, the vector of channels to transform, and FlowSOM.params = list(xdim=10, ydim=10, nClus=20, scale=FALSE)

6) Test the coefficient of variation within clusters with the testCV() function.

7) Import the metadata with the batch label of each sample contained in the excel file md.xlsx, using read.xlsx2(). Create 2 vectors using the column "batch" in the md.xlsx file. One vector contains the batch labels of the samples that correspond to sample "REU271", and another vector contains the batch labels of the other samples (i.e. not "REU271").

8) Estimate quantiles from the training flowSet using CytoNorm.train(). Use FlowSOM.params = list(nCells = 6000, xdim = 10, ydim = 10, nClus = 5, scale = FALSE)

9) Normalize the rest of the samples using CytoNorm.normalize(), and using outputDir = "course_datasets/FR_FCM_Z4KT/Normalized" ; Make sure this is a new folder.

10) Choosing one channel, create a ridge plot of its distribution within samples before normalization (without the training samples), and one for the normalized samples. For this, you need to create a new flowSet with the created Norm_ fcs files. Use the densityplot() function for each flowSet, storing the output in 2 objects, then use the cowplot plot_grid() function to plot one above the other.

# 04 Working with gated data in R

# Gating data in R

- The *GatingSet* class of objects (*flowWorkspace*) for working with gating data in R
- Import a *FlowJo* or *Cytobank* workspace (xml file) with gating data into R
- Manual gating from scratch
  - Using functions from the *flowWorkspace* package
  - Using a graphic-based, interactive tool (*flowGate* package)
- Automated gating methods
  - *FlowClust* package
  - *OpenCyto* package

# *flowWorkspace*

https://bioconductor.org/packages/release/bioc/html/flowWorkspace.html

- Provides the *GatingSet* class of objects as an efficient data structure to store, query and visualize gated flow data
- A *GatingSet* (**gs**) stores multiple *GatingHierarchy* (**gh**) objects associated with individual samples

*GatingSet* ~ *flowSet*

*GatingHierarchy* ~ *flowFrame*

- Unlike *flowSet*s, functions that operate on a *GatingSet* have the potential side-effect of modifying the object (all the modifications are made to the *external pointer*, rather than the R object itself)

# *flowWorkspaceData*

https://bioconductor.org/packages/release/bioc/html/flowWorkspace.html

- Contains FCS data files, XML workspaces and GatingSets for testing the flowWorkspace and openCyto packages
- Data from whole blood

**Installation**

To install this package, start R (version "4.3") and enter:

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("flowWorkspaceData")
```

# Create a *gatingSet*

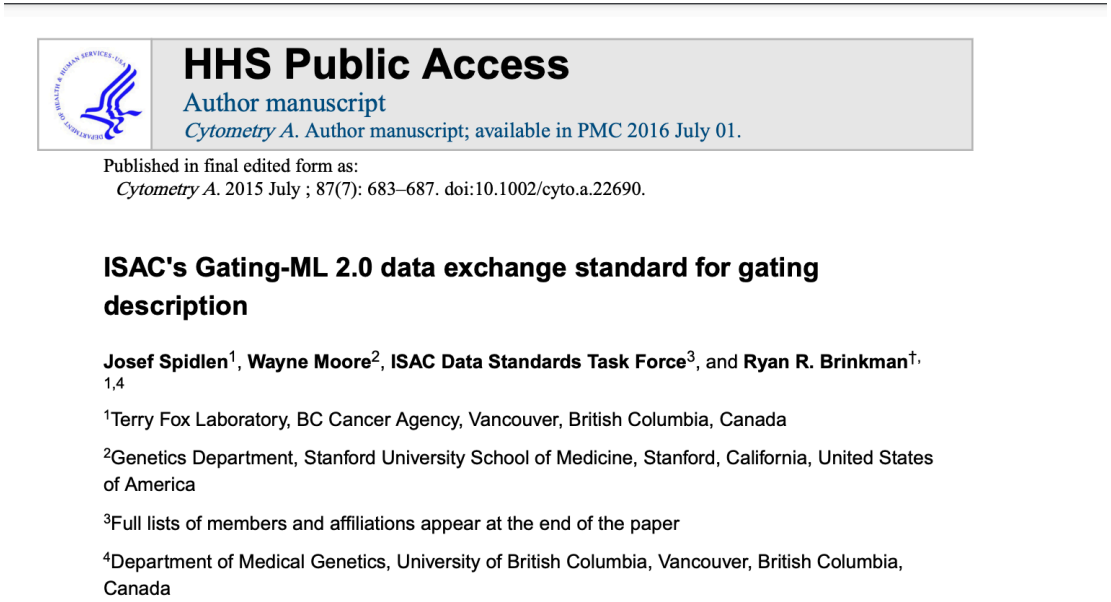
Ways to generate a *gatingSet:*

- Imported from workspace XML files from FlowJo, CytoBank or other software software using *CytoML* package
- Built from scratch within R (**manual gating**)
- Generated by **automated gating** methods (e.g. *openCyto* package)

# Import a workspace using *CytoML*

https://bioconductor.org/packages/release/bioc/html/CytoML.html

- Uses platform-specific implementations of the GatingML2.0 standard to exchange gated cytometry data



**HHS Public Access**

Author manuscript

*Cytometry A*. Author manuscript; available in PMC 2016 July 01.

Published in final edited form as:

*Cytometry A*. 2015 July ; 87(7): 683–687. doi:10.1002/cyto.a.22690.

**ISAC's Gating-ML 2.0 data exchange standard for gating description**

**Josef Spidlen**[1], **Wayne Moore**[2], **ISAC Data Standards Task Force**[3], and **Ryan R. Brinkman**[†,1,4]

[1]Terry Fox Laboratory, BC Cancer Agency, Vancouver, British Columbia, Canada

[2]Genetics Department, Stanford University School of Medicine, Stanford, California, United States of America

[3]Full lists of members and affiliations appear at the end of the paper

[4]Department of Medical Genetics, University of British Columbia, Vancouver, British Columbia, Canada

```
# import workspace from FlowJo
> ws <- open_flowjo_xml("course_datasets/flowWorspaceData/manual.xml")
> gs <- flowjo_to_gatingset(ws, name = "T-cell")
```

# *flowWorkspace*: basics on *GatingSet* objects

**# List the samples stored in the GatingSet**

```
> sampleNames(gs)
[1] "CytoTrol_CytoTrol_1.fcs_119531" "CytoTrol_CytoTrol_2.fcs_115728"
```

**# Access metadata**

```
> pData(gs)
                                                                name condition
CytoTrol_CytoTrol_1.fcs_119531 CytoTrol_CytoTrol_1.fcs treatment
CytoTrol_CytoTrol_2.fcs_115728 CytoTrol_CytoTrol_2.fcs   control
```

**# Add metadata**

```
pData(gs)$condition <- c("treatment","control")
```

**# Subset a GatingSet by metadata column**

```
> subset(gs, subset = treatment == "control")
A GatingSet with 1 samples
```

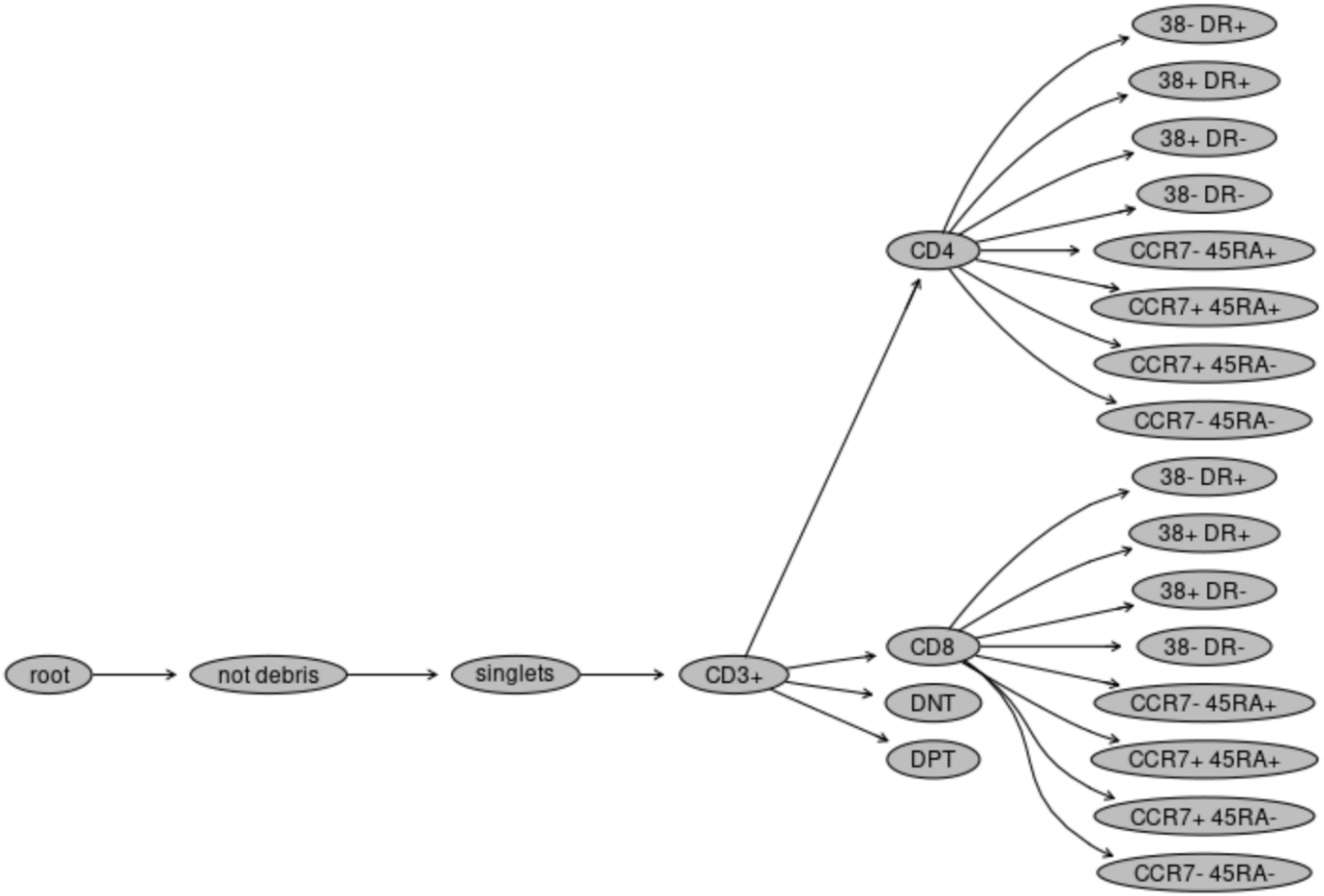
**# Retrieve a GatingHierarchical (one sample)**

```
> gh <- gs[[1]]
> gh
Sample:  CytoTrol_CytoTrol_1.fcs_119531
GatingHierarchy with  24  gates
```

# *flowWorkspace*: basics on *GatingSet* objects

**# plot the gating hierarchy (tree)**

> plot(gs)

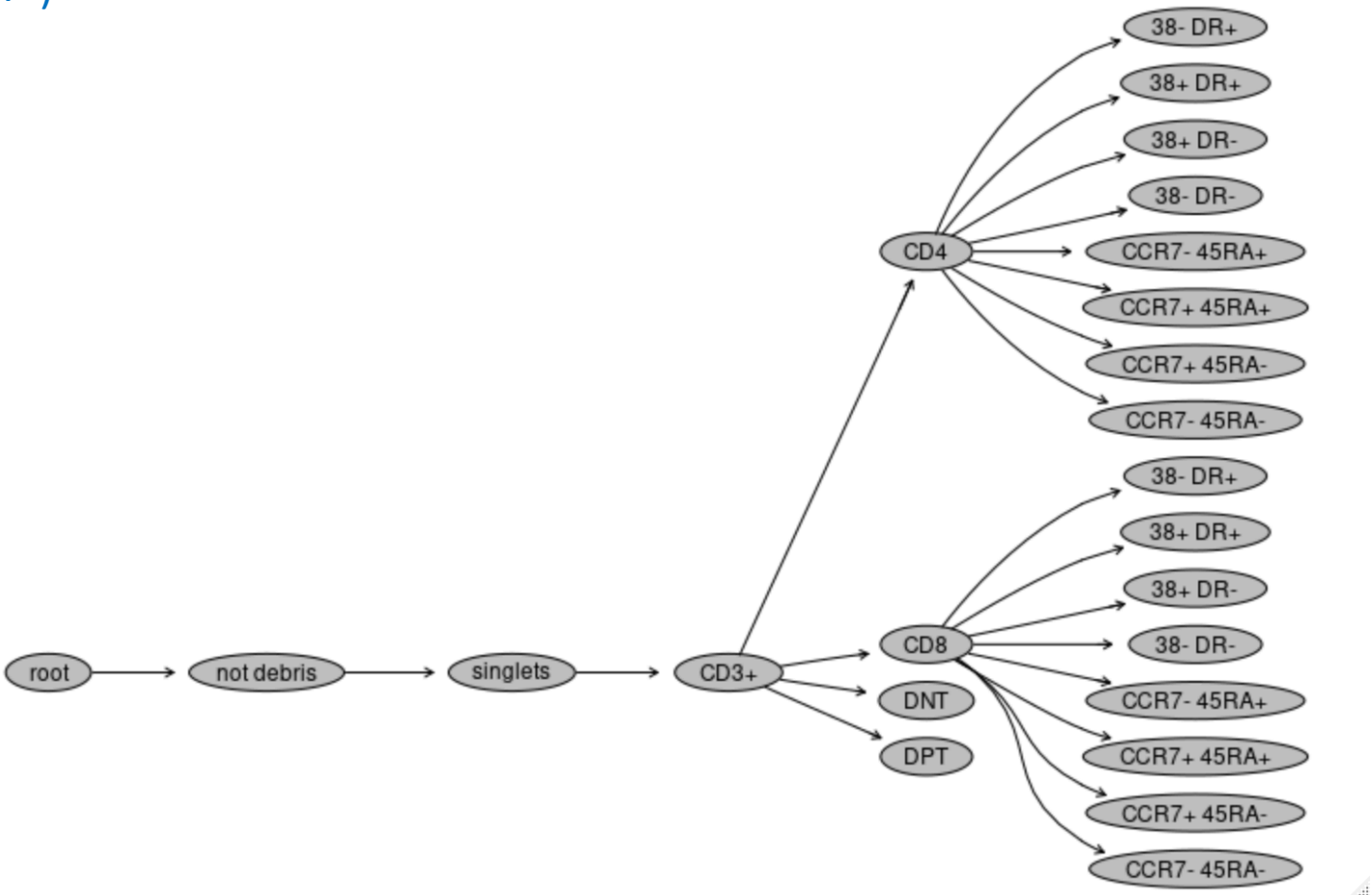


# *flowWorkspace*: basics on *GatingSet* objects

**# Delete a gate**

> Rm(gs, "DPT")

> plot(gs)



# *flowWorkspace*: basics on *GatingSet* objects

**# list nodes (cell populations)**

```
> gs_get_pop_paths(gs, path = 2)
 [1] "root"                "not debris"          "not debris/singlets" "singlets/CD3+"       "CD3+/CD4"
 [6] "CD4/38- DR+"         "CD4/38+ DR+"         "CD4/38+ DR-"         "CD4/38- DR-"         "CD4/CCR7- 45RA+"
[11] "CD4/CCR7+ 45RA+"     "CD4/CCR7+ 45RA-"     "CD4/CCR7- 45RA-"     "CD3+/CD8"            "CD8/38- DR+"
[16] "CD8/38+ DR+"         "CD8/38+ DR-"         "CD8/38- DR-"         "CD8/CCR7- 45RA+"     "CD8/CCR7+ 45RA+"
[21] "CD8/CCR7+ 45RA-"     "CD8/CCR7- 45RA-"     "CD3+/DNT"            "CD3+/DPT"

> gs_get_pop_paths(gs, path = "full")
 [1] "root"                                    "/not debris"
 [3] "/not debris/singlets"                    "/not debris/singlets/CD3+"
 [5] "/not debris/singlets/CD3+/CD4"           "/not debris/singlets/CD3+/CD4/38- DR+"
 [7] "/not debris/singlets/CD3+/CD4/38+ DR+"   "/not debris/singlets/CD3+/CD4/38+ DR-"
 [9] "/not debris/singlets/CD3+/CD4/38- DR-"   "/not debris/singlets/CD3+/CD4/CCR7- 45RA+"
[11] "/not debris/singlets/CD3+/CD4/CCR7+ 45RA+" "/not debris/singlets/CD3+/CD4/CCR7+ 45RA-"
[13] "/not debris/singlets/CD3+/CD4/CCR7- 45RA-" "/not debris/singlets/CD3+/CD8"
[15] "/not debris/singlets/CD3+/CD8/38- DR+"   "/not debris/singlets/CD3+/CD8/38+ DR+"
[17] "/not debris/singlets/CD3+/CD8/38+ DR-"   "/not debris/singlets/CD3+/CD8/38- DR-"
[19] "/not debris/singlets/CD3+/CD8/CCR7- 45RA+" "/not debris/singlets/CD3+/CD8/CCR7+ 45RA+"
[21] "/not debris/singlets/CD3+/CD8/CCR7+ 45RA-" "/not debris/singlets/CD3+/CD8/CCR7- 45RA-"
[23] "/not debris/singlets/CD3+/DNT"           "/not debris/singlets/CD3+/DPT"

> gs_get_pop_paths(gs, path = "auto")
 [1] "root"            "not debris"      "singlets"        "CD3+"            "CD4"             "CD4/38- DR+"     "CD4/38+ DR+"
 [8] "CD4/38+ DR-"     "CD4/38- DR-"     "CD4/CCR7- 45RA+" "CD4/CCR7+ 45RA+" "CD4/CCR7+ 45RA-" "CD4/CCR7- 45RA-" "CD8"
[15] "CD8/38- DR+"     "CD8/38+ DR+"     "CD8/38+ DR-"     "CD8/38- DR-"     "CD8/CCR7- 45RA+" "CD8/CCR7+ 45RA+" "CD8/CCR7+ 45RA-"
[22] "CD8/CCR7- 45RA-" "DNT"             "DPT"
```

# FlowWorkspace: basics on *GatingSet*

```
# retrieve data from all nodes as a cytoset
 > cs <- gs_pop_get_data(gs)
 > class(cs)
[1] "cytoset"
attr(,"package")
[1] "flowWorkspace"
```

```
# convert the cytoset to a flowSet
 > fs <- cytoset_to_flowSet(cs)
```

```
# check the number of cells in the flowSet
 > fsApply(fs, nrow)
```

```
                                                  [,1]
CytoTrol_CytoTrol_1.fcs_119531 119531
CytoTrol_CytoTrol_2.fcs_115728 115728
```

```
# retrieve data associated to one node (gate)
 > cs <- gs_pop_get_data(gs, "CD4")
 > fs <- cytoset_to_flowSet(cs)
 > fsApply(fs, nrow)
```

```
                                                  [,1]
CytoTrol_CytoTrol_1.fcs_119531 34032
CytoTrol_CytoTrol_2.fcs_115728 33751
```

# FlowWorkspace: basics on *GatingSet* object

**# Get membership indices with respect to a gate**

```
> gh_pop_get_indices(gs[[1]], "CD4")
```

```
  [1]  TRUE FALSE FALSE FALSE FALSE  TRUE FA
 [16]  TRUE FALSE  TRUE  TRUE  TRUE FALSE FA
 [31] FALSE FALSE FALSE FALSE  TRUE FALSE FA
```

```
> table(gh_pop_get_indices(gs[[1]], "CD4"))
```

```
FALSE  TRUE
85499 34032
```

**# save / load a GatingSet**

```
> save_gs(gs, path = "course_datasets/flowWorspaceData/gs")
> gs2 <- load_gs("course_datasets/flowWorspaceData/gs")
```

The regular R assignement (<-) or save() routine doesn't work for GatingSet objects

# Build a *GatingSet* from scratch

## Start from a *flowSet* (*flowCore*)

```
# Read the FCS files
> fs <- read.flowSet(path="course_datasets/flowWorspaceData/", pattern = "*.fcs")

# Arcsinh transform with flowVS
> panel <- pData(parameters(fs[[1]]))
> markerstotransf <- as.character(panel$name[!is.na(panel$desc)])
> fs <- transFlowVS(fs, channels = markerstotransf,
                    cofactors = rep(3000,length(markerstotransf)))
# Convert to a GatingSet
> gs <- GatingSet(fcs_transform) # convert the flowSet to GatingSet
```

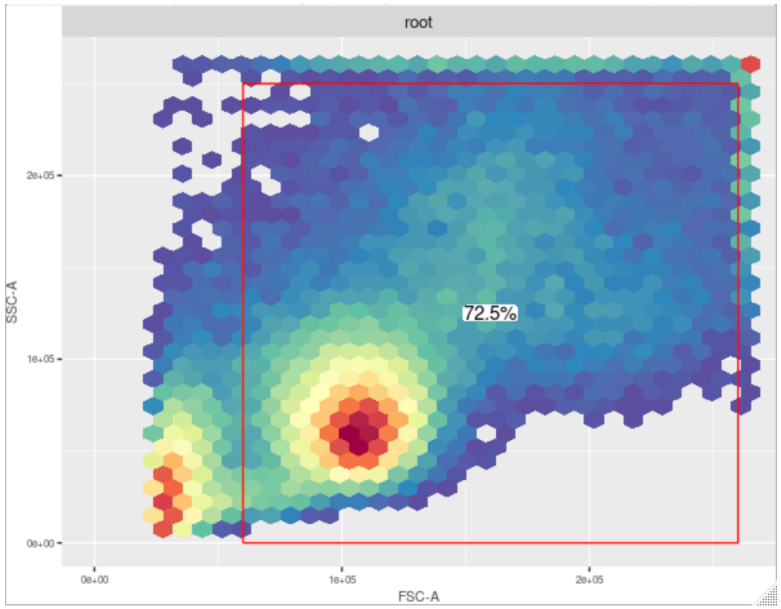
## Start from a single cell experiment object (*CATALYST*)

```
> load("course_datasets/FR_FCM_Z4KT/DA_example_sce_PBMC.RData") # load the sce
> fs <- sce2fcs(sce_PBMC, assay = "exprs") # convert sce to flowSet (CATALYST)
> gs <- GatingSet(fs) # convert flowSet to GatingSet
```

# Manual gating : rectangle gate

**# Create a rectangle gate**

```
> rg1  <- rectangleGate("FSC-A"=c(60000,260000),
                        "SSC-A"=c(1, 250000),
                        filterId="NotDebris")
```




**# Add the gate to the GatingSet object**

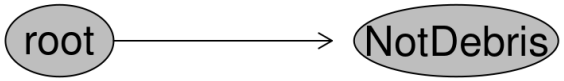
```
> gs_pop_add(gs, rg1, parent = "root")
```

**# Apply the gate to the data**

```
> recompute(gs)
```

**# Plot the gate**

```
> autoplot(gs[[1]], gate = "NotDebris")
```




**# Check gating hierarchy**

```
> plot(gs)
```

**# Get statistics**

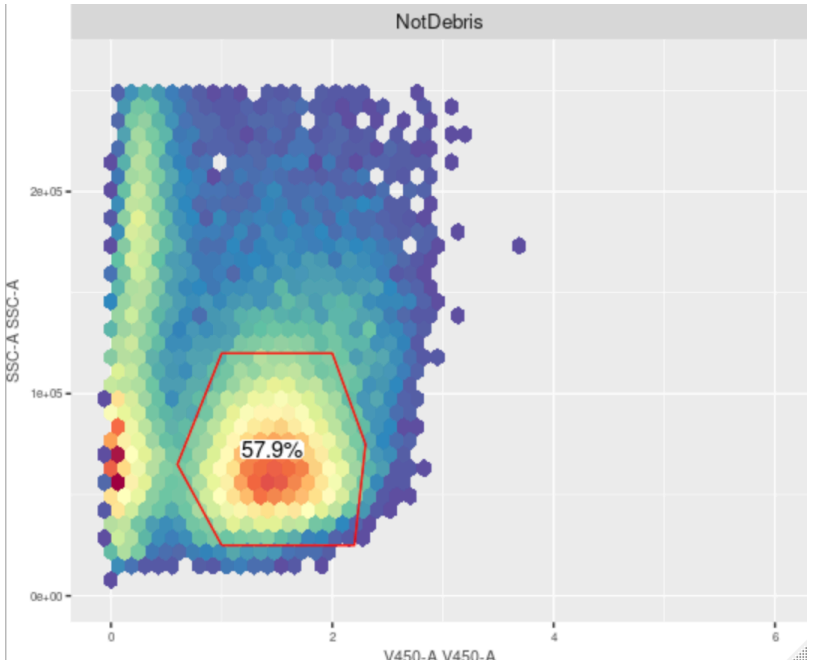
```
> gs_pop_get_stats(gs[[]1], "NonDebris") # counts
> gs_pop_get_stats(gs[[]1], "NonDebris", type = "percent") # proportions
```

```
                      sample       pop count
1: CytoTrol_CytoTrol_1.fcs NotDebris 86699
2: CytoTrol_CytoTrol_2.fcs NotDebris 86689
```

```
                      sample       pop   percent
1: CytoTrol_CytoTrol_1.fcs NotDebris 0.7253265
2: CytoTrol_CytoTrol_2.fcs NotDebris 0.7490754
```

# Manual gating: polygon gate

```
# Define the vertices of the polygon
my_vertices <-matrix(c(1,0.6,1,2,2.3,2.2,
                       25000,65000,120000,120000,75000,25000),
                       ncol=2,nrow=6)
colnames(my_vertices) <- c("V450-A","SSC-A")

# Create polygon gate "singlets"
rg2  <- polygonGate( boundaries= my_vertices,
                     filterId="CD3")

# Add the gate to the GatingSet
gs_pop_add(gs, rg2, parent = "NotDebris")

# Recompute the GatingSet
recompute(gs)

# Check
autoplot(gs[[1]], gate = "CD3")
plot(gs)
```




# Manual gating: quadrant gate

**# Create quadrant gate "CD4 CD8"**

```
> rg3  <- quadGate("B710-A"= 1.5,
                   "R780-A"= 3,
                   filterId = "CD4 CD8")
```

**# Add the gate to the GatingSet**

```
> gs_pop_add(gs, rg3, parent = "CD3")
```

**# Recompute the GatingSet**

```
> recompute(gs)
```

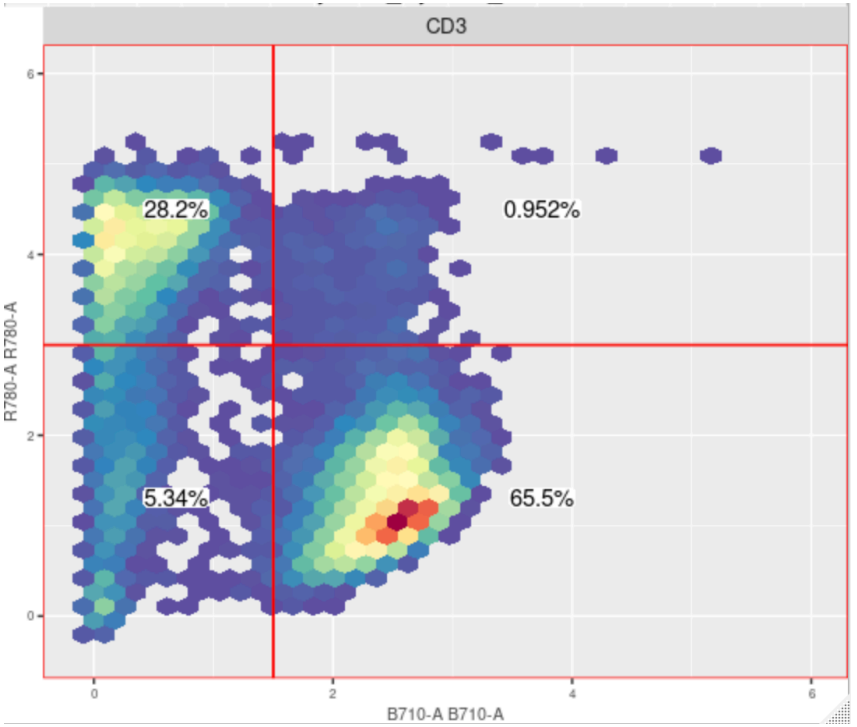
**# Check**

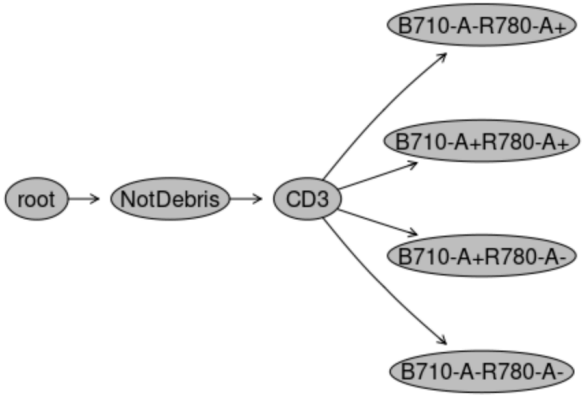
```
> gs_get_pop_paths(gs)
```

```
[1] "root"                          "/NotDebris"
[3] "/NotDebris/CD3"                "/NotDebris/CD3/B710-A-R780-A+"
[5] "/NotDebris/CD3/B710-A+R780-A+" "/NotDebris/CD3/B710-A+R780-A-"
[7] "/NotDebris/CD3/B710-A-R780-A-"
```

# Plot

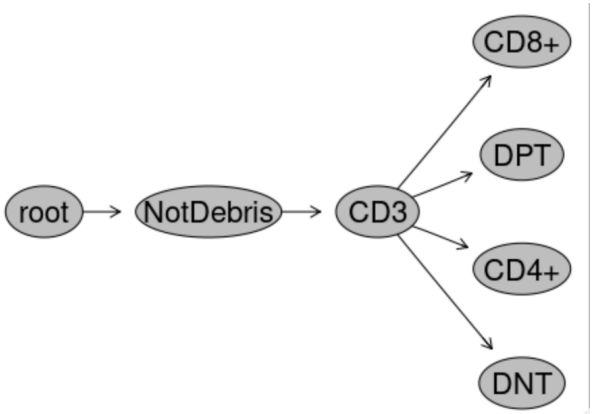
```
> autoplot(gs[[1]], gate = gs_get_pop_paths(gs)[4:7])
> plot(gs)
```



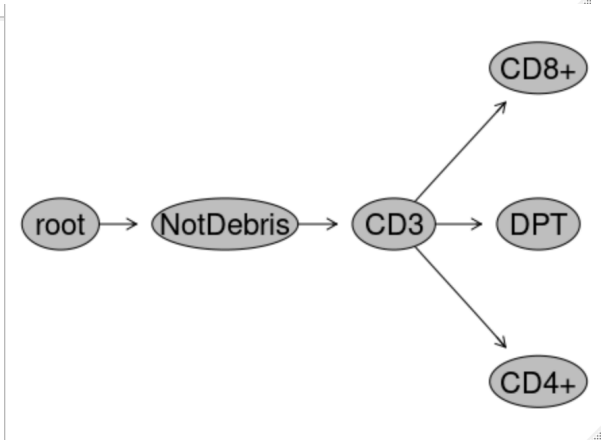



# FlowWorkspace: other utilities

```
# Rename nodes (gates)
> gs_pop_set_name(gs,"B710-A-R780-A+","CD8+")
> gs_pop_set_name(gs,"B710-A+R780-A-","CD4+")
> gs_pop_set_name(gs,"B710-A-R780-A-","DNT")
> gs_pop_set_name(gs,"B710-A+R780-A+","DPT")
> plot(gs)
```




```
# Remove a node
> gs_pop_remove(gs, "DNT")
> plot(gs)
```




```
# retrieve the flow data for a node
> fs_CD8 <- gs_pop_get_data( gs, "CD8+")
```

# Interactive gating with *flowGate*

https://bioconductor.org/packages/release/bioc/html/flowGate.html

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("flowGate")
```

- Interactive cytometry gating in R
- Based on a shiny app (web application using R)
- Especially geared toward wet-lab cytometerists looking to take advantage of R without having a lot of experience
- Uses GatingSet objects
- You can use transformed data, but flowGate was designed to apply desired transformations at the plotting level only

# Interactive gating with *FlowGate*

**# Load FCS files and preprocess**

```
> fs <- read.flowSet(path="course_datasets/flowWorspaceData/", pattern = "*.fcs")
 > panel <- pData(parameters(fs[[1]]))
 > markerstotransf <- as.character(panel$name[!is.na(panel$desc)])
 > fs <- transFlowVS(fs, channels = markerstotransf,
                          cofactors = rep(3000,length(markerstotransf)))
```

**# Convert to GatingSet**

```
> gs <- GatingSet(fs)
```

1st: chose type of gate

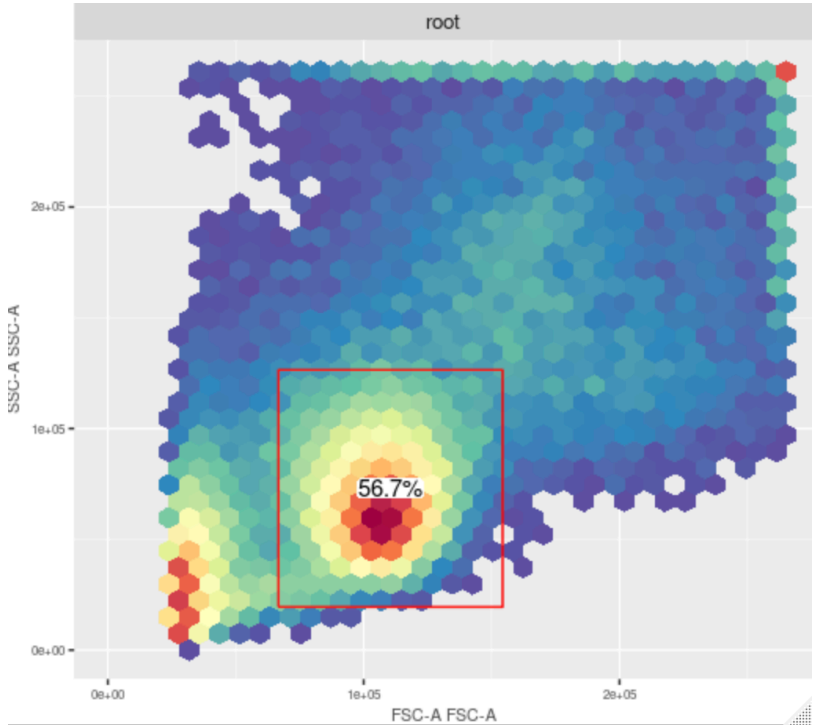
**# Start the interactive gating**

```
 > gs_gate_interactive(gs,
              filterId = "NotDebris",
              dims = list("FSC-A", "SSC-A"))
```

# Interactive gating with *FlowGate*

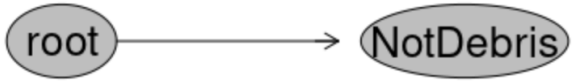
**# Plot the data with the new gate**

```
> autoplot(gs[[1]], gate = "NotDebris")
```




**# Plot hierarchy**

```
> plot(gs)
```

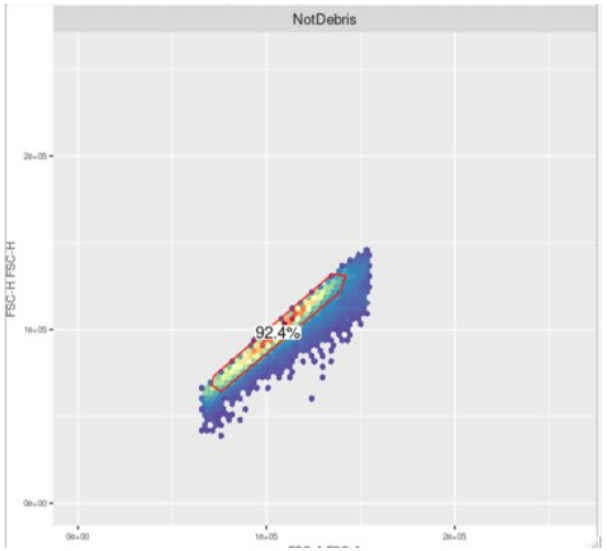


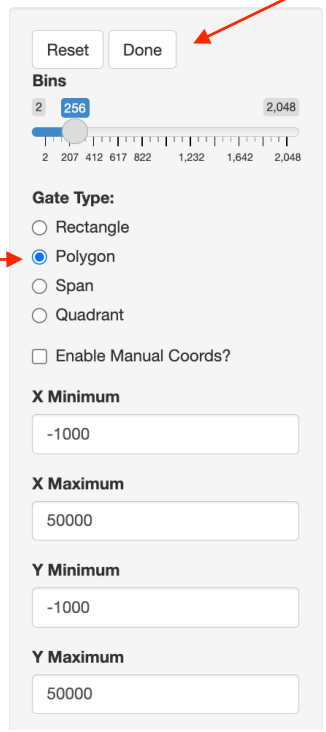
# Interactive gating with *FlowGate*

## Create a 2-D *polygon* gate

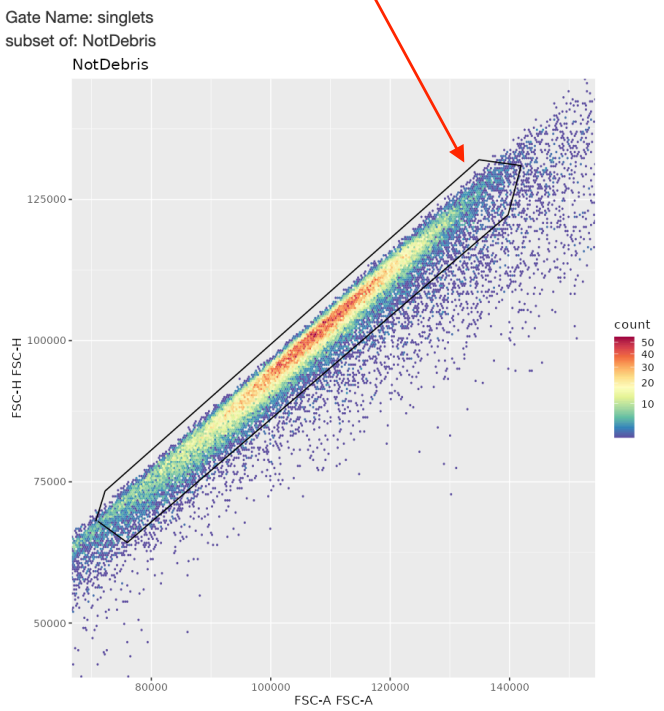
```
# Create a polygon gate
 > gs_gate_interactive(gs,
                filterId = "singlets",
                dims = list("FSC-A", "FSC-H"),
                subset = "NotDebris")
```

```
# check
 > autoplot(gs[[1]], gate = "singlets")
```




Click on the vertices



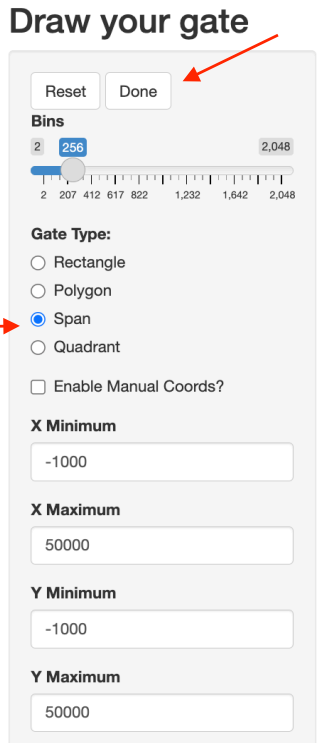

```
> plot(gs)
```

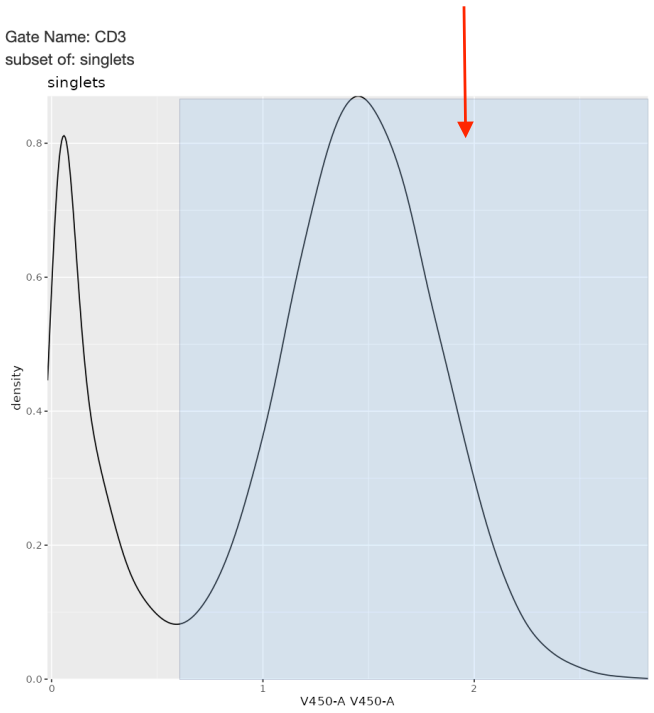
# Interactive gating with *FlowGate*

## Create a 1-D *span* gate

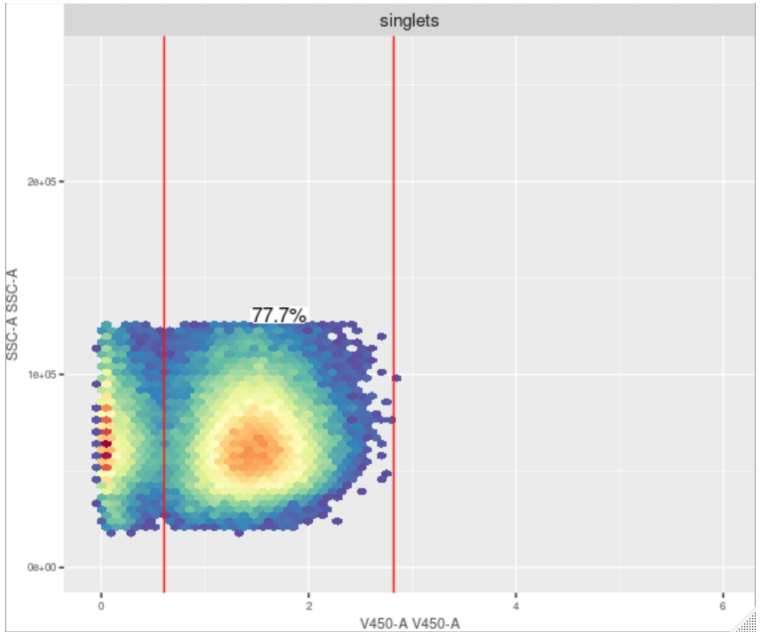
```
> gs_gate_interactive(gs,
                      filterId = "CD3",
                      dims = "V450-A",
                      subset = "singlets")
```

Drag a window

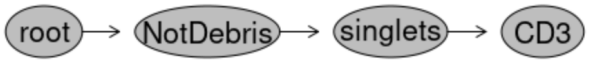



```
# Check the gate
 > autoplot(gs[[1]], gate = "CD3")
```




```
> plot(gs)
```

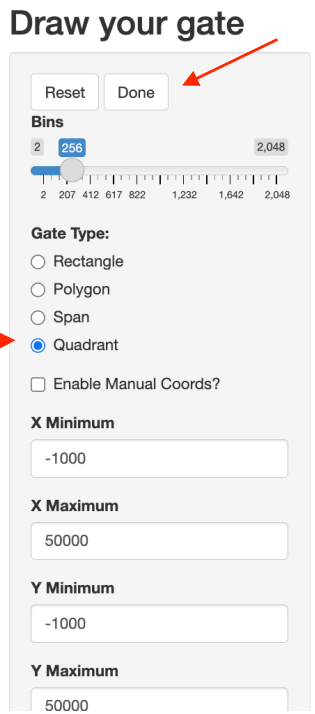


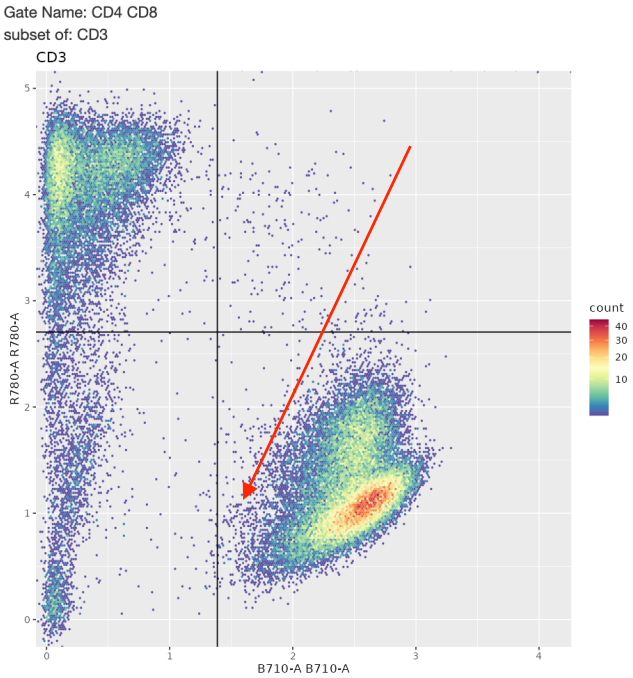
# Interactive gating with *FlowGate*

## Create a 2-D quadrant gate

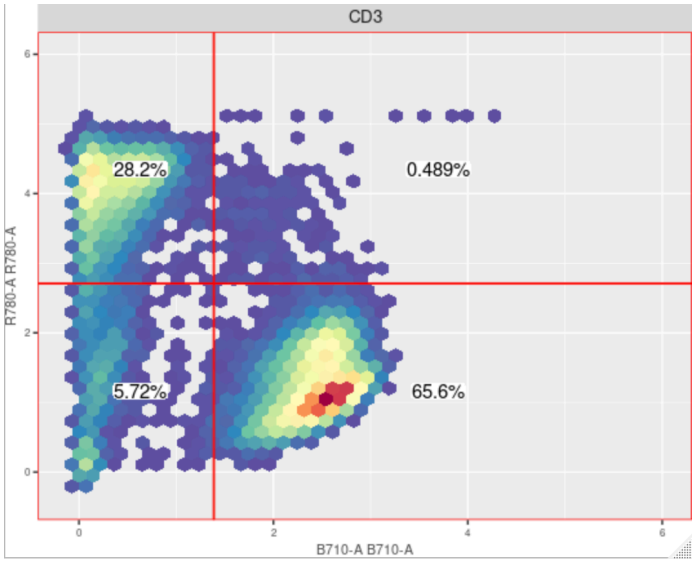
```
> gs_gate_interactive(gs,
                      filterId = "CD4 CD8",
                      dims = c("B710-A",
                               "R780-A"),
                      subset = "CD3")
```

Select the origin of the quadrant

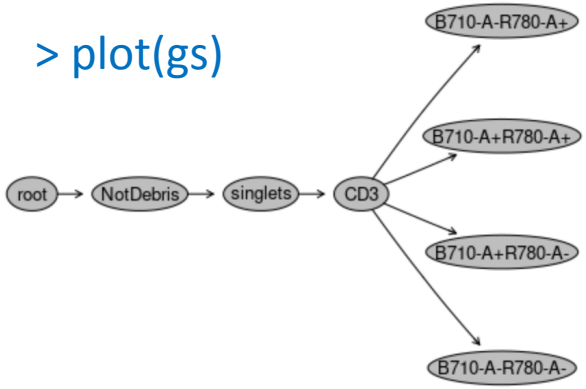



```
# check
my_nodes <- gs_pop_get_children(gs,"CD3")
autoplot(gs[[1]], my_nodes)
```




```
> plot(gs)
```



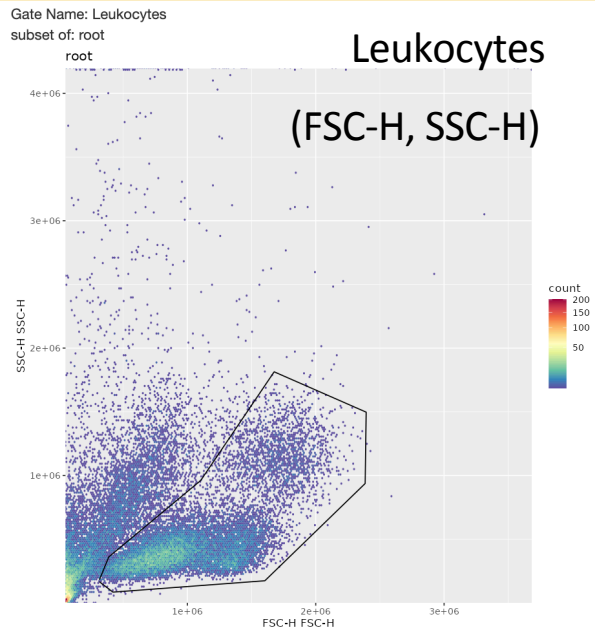
# Let’s practice – 9

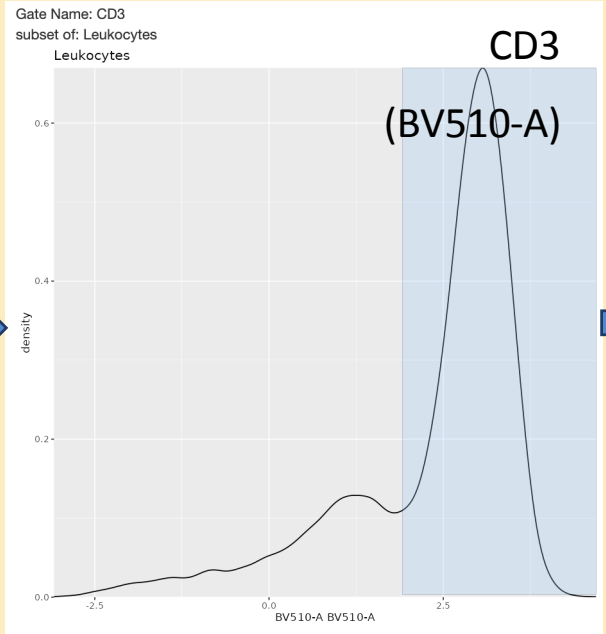
In this exercise we will  do some gating using flowGate and data from the **FR_FCM_Z3WR** of the FlowRepository.

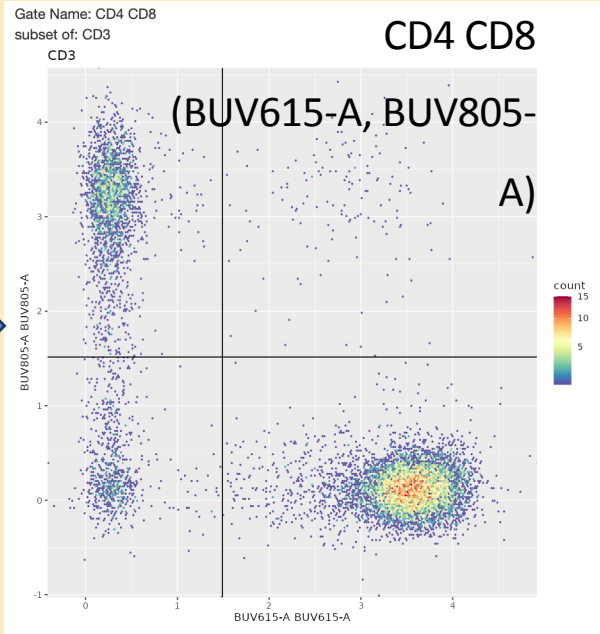
Create a new script in which you will:

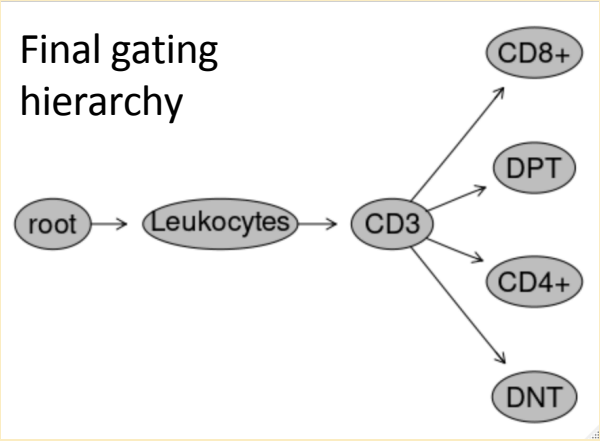
1) Create a flowSet of all samples within the */course_dataset/FR_FCM_Z3WR* folder
2) Perform asinh transformation with a cofactor of 3000 for all channels not labeled with "none", using transFlowVS() from the flowVS package. Use the csv file with the panel previously created (/course_datasets/FR_FCM_Z3WR/panel_with_marker_classes.csv)
3) Convert the *flowSet* to a *GatingSet*
4) Using *flowGate*, create a gating hierarchy according to the scheme depicted in the following slide. Don't forget to check your gating with scatter or density plots.
5) Do necessary adjustements so that your gating hierarchy looks like the one depicted in the next slide
6) What is the percentage of CD8+ T cells among T cells ("CD3")

# Let’s practice – 9 : Gates



# 04 Automated Gating

# Automated gating with *flowClust*

https://www.bioconductor.org/packages/release/bioc/html/flowClust.html

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("flowClust")
```

- Robust Model-based Clustering of Flow Cytometry Data (Lo et al. 2008)
- Identify cell populations in flow cytometry data
- Based on a multivariate t mixture model with Box-Cox transformation
- Two options for **estimating the number of clusters when it is unknown**
- **Input are *flowFrames***

# Automated gating with *openCyto*

https://bioconductor.org/packages/release/bioc/html/openCyto.html

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("openCyto")
```

- Hierarchical gating pipeline for flow cytometry data
- Based on *GatingSet* objects
- Wide variety of methods, including *flowClust*
- Automated gating using a *gatingTemplate* (hierarchical gating scheme)
- Possibility of step-by-step gating without a template

# Automated gating with *openCyto*

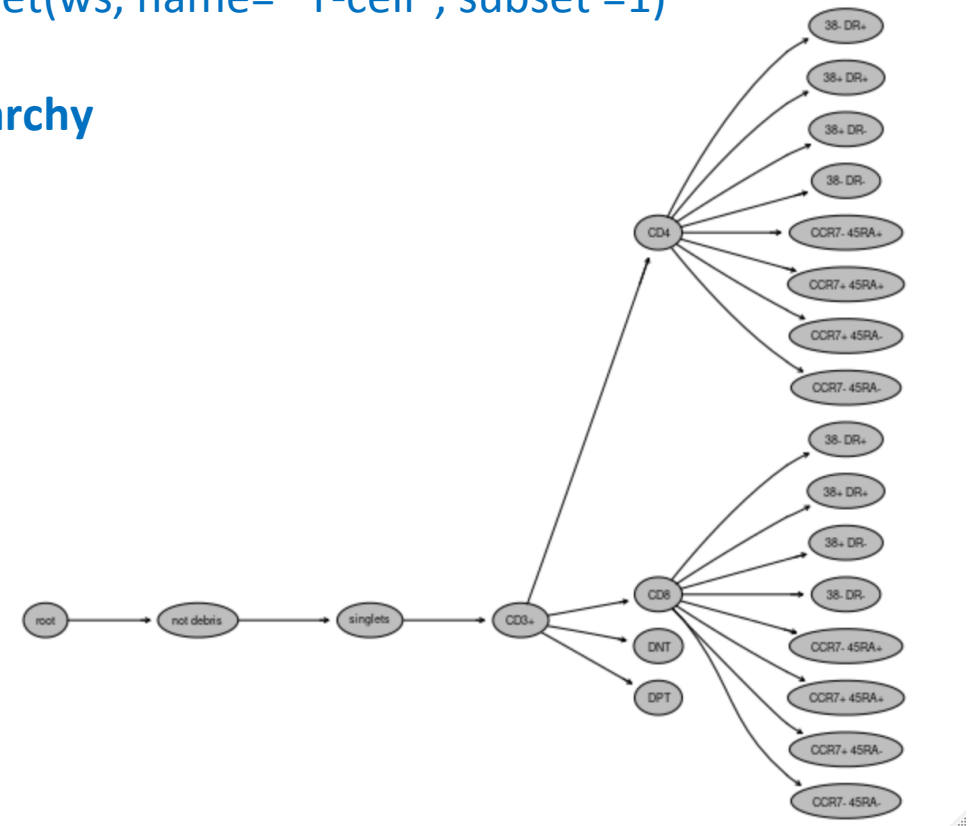
Example from flowWorkspaceData (gated data)

**# Load example gatingSet from FlowJo workspace (xml file)**

```
> library(CytoML)
> ws <- open_flowjo_xml("course_datasets/flowWorspaceData/manual.xml")
> gs_final <- flowjo_to_gatingset(ws, name= "T-cell", subset =1)
```

**# Check complete gating hierarchy**

```
> plot(gs_final[[1]])
```

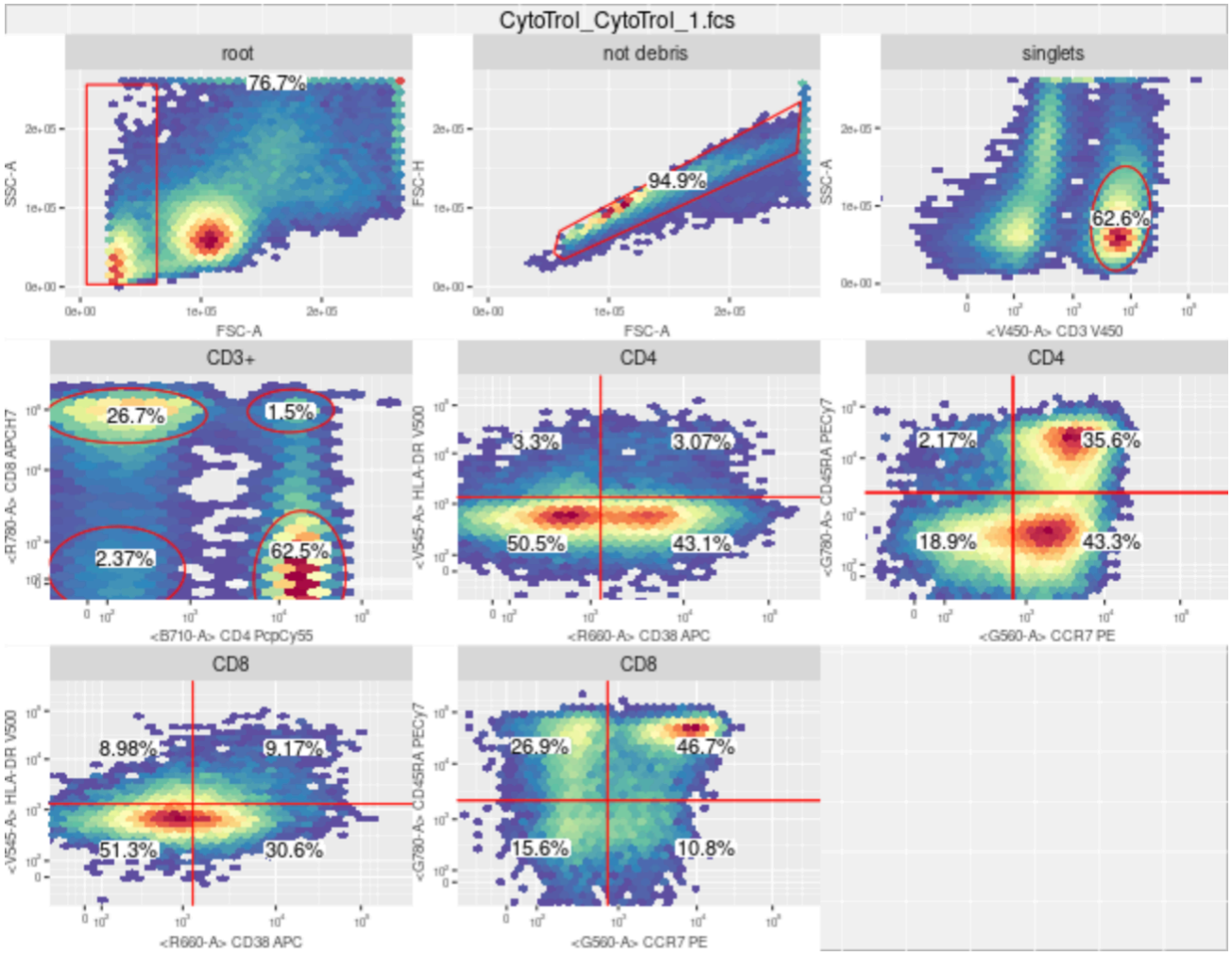


# Automated gating with *openCyto*

Example from flowWorkspaceData (gated data)

**# Plot gates**

> autoplot(gs_final[[1]])



# *gatingTemplate*

```
# Check the structure of the gatingTemplate
> my_gt <- read.csv("course_datasets/flowWorkspaceData/tcell.csv")
> view(my_gt)
```

| | alias | pop | parent | dims | gating_method | gating_args | collapseDataForGating | groupBy | preprocessing_method | preprocessing_args |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | nonDebris | + | root | FSC-A | gate_mindensity | | *NA* | *NA* | | *NA* |
| 2 | singlets | + | nonDebris | FSC-A,FSC-H | singletGate | | *NA* | *NA* | | *NA* |
| 3 | lymph | + | singlets | FSC-A,SSC-A | flowClust | K=2,target=c(1e5,5e4) | *NA* | *NA* | prior_flowClust | *NA* |
| 4 | cd3 | + | lymph | CD3 | gate_mindensity | | TRUE | 4 | | *NA* |
| 5 | * | -/++/- | cd3 | cd4,cd8 | gate_mindensity | gate_range=c(1,3) | *NA* | *NA* | | *NA* |
| 6 | activated cd4 | ++ | cd4+cd8- | CD38,HLA | gate_mindensity | | *NA* | *NA* | standardize_flowset | *NA* |
| 7 | activated cd8 | ++ | cd4-cd8+ | CD38,HLA | gate_mindensity | | *NA* | *NA* | standardize_flowset | *NA* |
| 8 | CD45_neg | - | cd4+cd8- | CD45RA | gate_mindensity | gate_range=c(2,3) | *NA* | *NA* | | *NA* |
| 9 | CCR7_gate | + | CD45_neg | CCR7 | flowClust | neg=1,pos=1 | *NA* | *NA* | | *NA* |
| 10 | * | +/-+/- | cd4+cd8- | CCR7,CD45RA | refGate | CD45_neg:CCR7_gate | *NA* | *NA* | | *NA* |
| 11 | * | +/-+/- | cd4-cd8+ | CCR7,CD45RA | gate_mindensity | | *NA* | *NA* | | *NA* |

label of the cell population

parent poulation

method used for gating

collapse all samples into one, gate and then replicate across samples

should any preprocessing be applied

arguments for preprocessing

which side (1-D) or quadrant (2-D) to keep

dimensions used for gating

Arguments for the method

group every n samples

https://bioconductor.org/packages/release/bioc/vignettes/openCyto/inst/doc/HowToWriteCSVTemplate.html

# *gatingTemplate*

Example of a 1-D scan gate

| | alias | pop | parent | dims | gating_method | gating_args | collapseDataForGating | groupBy | preprocessing_method | preprocessing_args |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | nonDebris | + | root | FSC-A | gate_mindensity | | *NA* | *NA* | | *NA* |

- The population name will be "nonDebris"
- The parent node is "root"
- The gating method is *mindensity*
- It will generate a scan gate on FSC-A and keep the only the positive cells

# *gatingTemplate*

Example of a *singlet* gate

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | singlets | + | nonDebris | FSC-A,FSC-H | singletGate | | NA | NA | NA |

- The population name will be "singlets"
- The parent node is "nonDebris"
- It will use FSC-A and FSC-H
- The gating method is *singletGate*
- It will generate a polygon gate on FSC-A and FSC-H to keep only the singlets

# *gatingTemplate*

Example of a *flowClust* gate

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | lymph | + | singlets | FSC-A,SSC-A | flowClust | K=2,target=c(1e5,5e4) | *NA* | *NA* | prior_flowClust | *NA* |

- The population name will be "lymph"
- The parent node is "singlets"
- Use the method from *flowClust*
- It will generate an ellipsoid gate on FSC-A and SSC-A and split the cells in "peaks" (high density areas)
- There are parameters to be passed to flowClust: apply some preprocessing before gating; it should identify to populations, centered

# *gatingTemplate*

Example of a quadrant gate

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | * | -/++/- | cd3 | cd4,cd8 | gate_mindensity | gate_range=c(1,3) | NA | NA | | NA |

- Quadrant gate
- Based on mindesity for determining the origin of the quadrant
- Specifies that population CD4+/-CD8+/- should be expanded into 6 cell populations
  - The first two gates are span gates on each channel (CD4 and CD8)
  - The other four gates are rectangle gates that correspond to the four quadrants in the 2-D projection

# *openCyto*

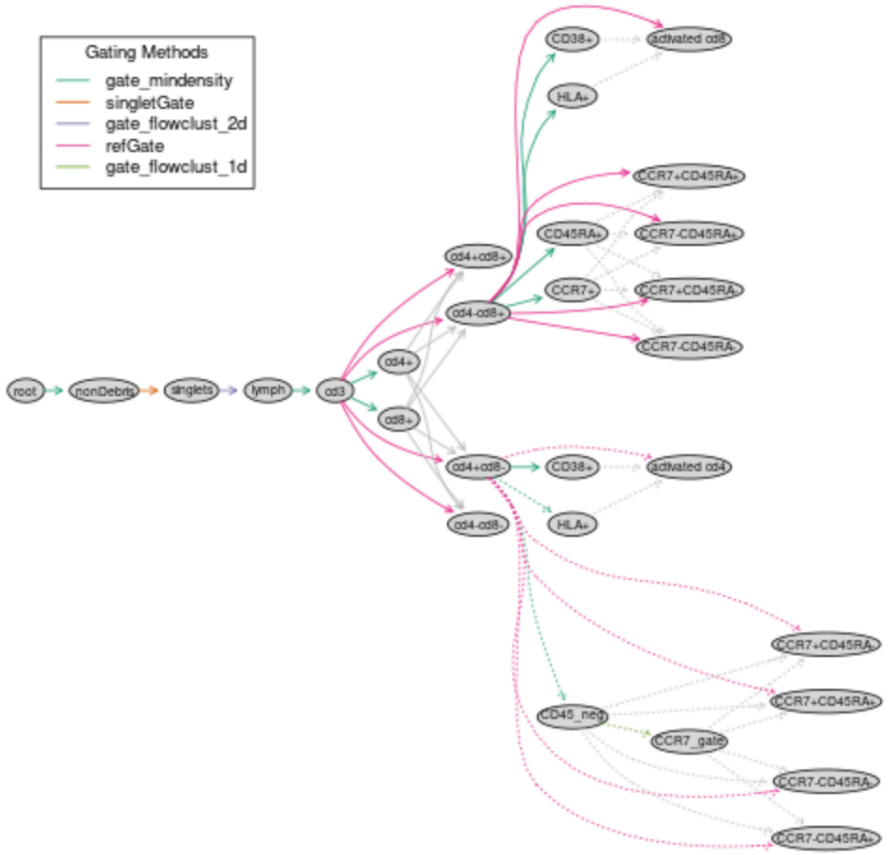
Create the gatingTemplate from the csv file

**# Create the gatingTemplate from a file**

```
> gt_tcell <- gatingTemplate("course_datasets/flowWorkspaceData/tcell.csv")
```

# Examine the gating scheme

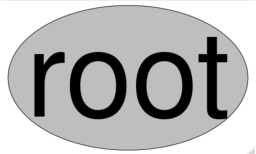
```
> plot(gt_tcell)
```



# *openCyto*: run the gating pipeline

Load the raw data and convert to gatingSet

```
# Load the preprocessed but ungated data
# The code used to preprocess this data is avalable in
# /Code_slides/Code_preprocessing_data_openCyto.R
> gs <- load_gs("course_datasets/flowWorspaceData/gs_preprocessed")

# Check
> plot(gs)
```



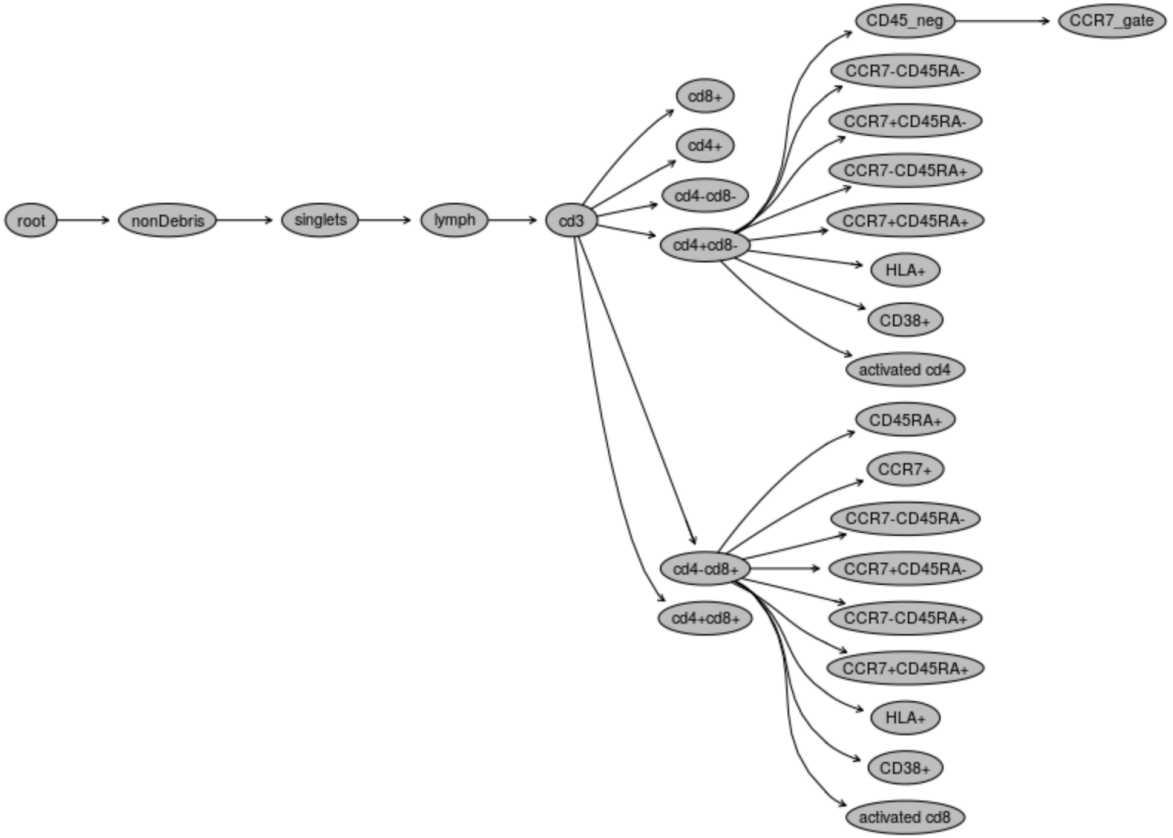

Apply the gating template

```
# Run the gating
 > gt_gating(gt_tcell, gs)
```

# *openCyto*: run the gating pipeline

## Check the gating

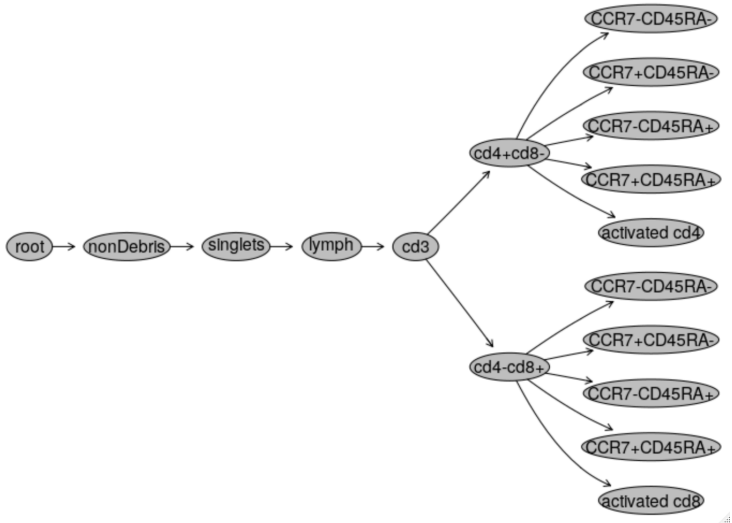
```
# Check the gating
 > plot(gs[[1]])
```



# *openCyto*

Hide populations we are not interested in

```
# Define nodes to hide
 > nodesToHide  <- c("cd8+", "cd4+" , "cd4-cd8", "cd4+cd8+" , "cd4+cd8-/HLA+",
                     "cd4+cd8-/CD38+" , "cd4-cd8+/HLA+", "cd4-cd8+/CD38+" ,
                     "CD45_neg/CCR7_gate", "cd4+cd8-/CD45_neg" ,
                     "cd4-cd8+/CCR7+", "cd4-cd8+/CD45RA+" )

# Apply the gs_pop_set_visibility() function to the nodes to hide
 > lapply(nodesToHide, function(thisNode) gs_pop_set_visibility(gs, thisNode, FALSE))

# Check after hiding nodes
> plot(gs[[1]])
```



# *openCyto*

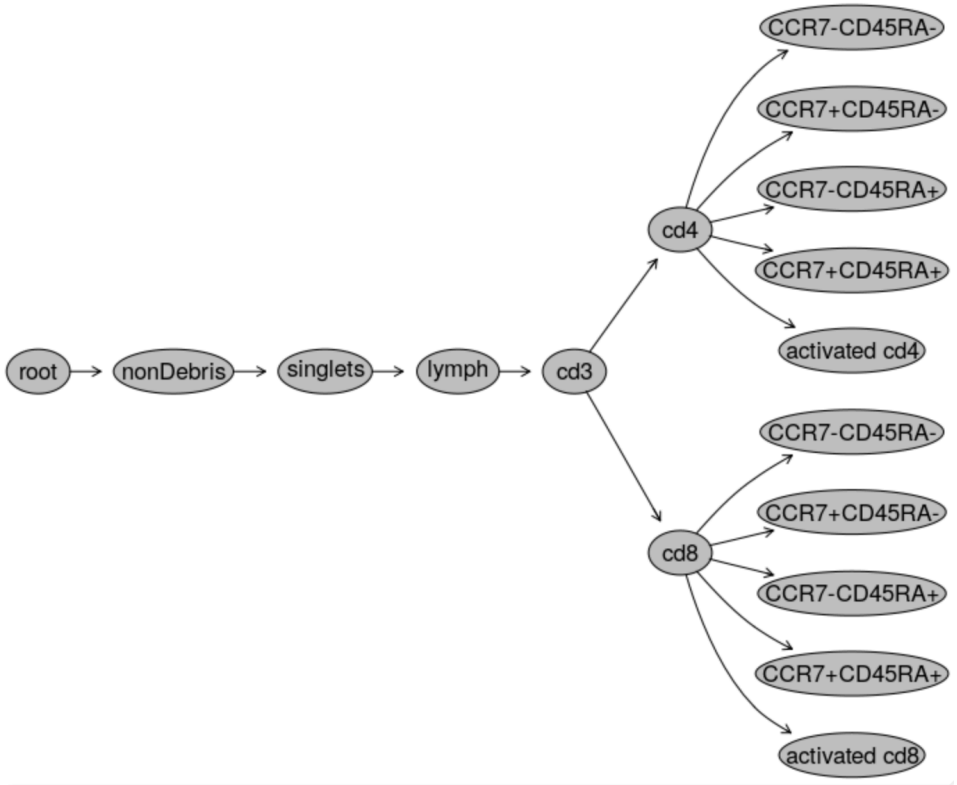
## Rename some populations

**# Rename**

```
> gs_pop_set_name(gs,"cd4+cd8-","cd4")
> gs_pop_set_name(gs,"cd4-cd8+","cd8")
```

**# Check**

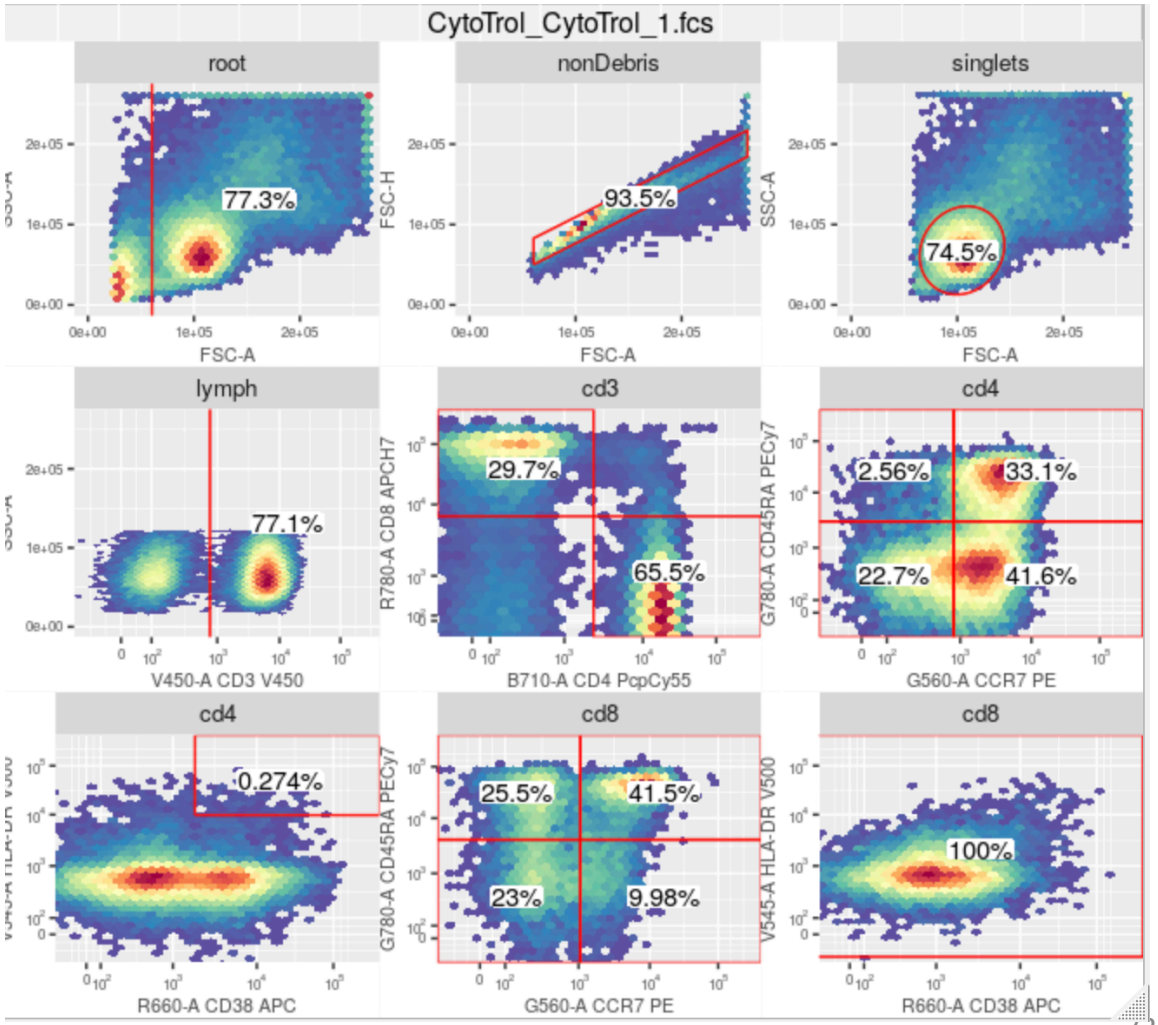
```
> plot(gs[[1]])
```



# *openCyto*

Visualize the gates with *ggcyto*

```
> autoplot(gs[[1]])
```



# *openCyto*: gating without a template

We can apply each automated gating step using the same fields as in the template

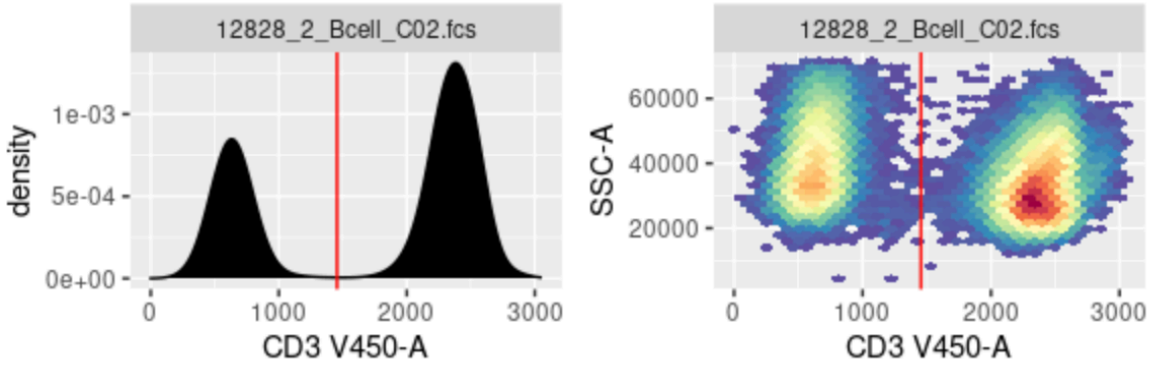
```
> gs_add_gating_method(gs,
                       alias = "non-activated cd4",
                       pop = "--",
                       parent = "cd4",
                       dims = "CD38,HLA",
                       gating_method = "tailgate")

# check
> plot(gs[[1]])
```

# *openCyto*: overview of gating methods

https://bioconductor.org/packages/release/bioc/vignettes/openCyto/inst/doc/HowToAutoGating.html

*mindensity*

Finds the minimum as the cutpoint between positive and negative peaks in a 1-D density plot




- Fast, robust and easy to use
- For markers with a good separation between + and – peaks
- Needs more guidance when there are more than 2 peaks

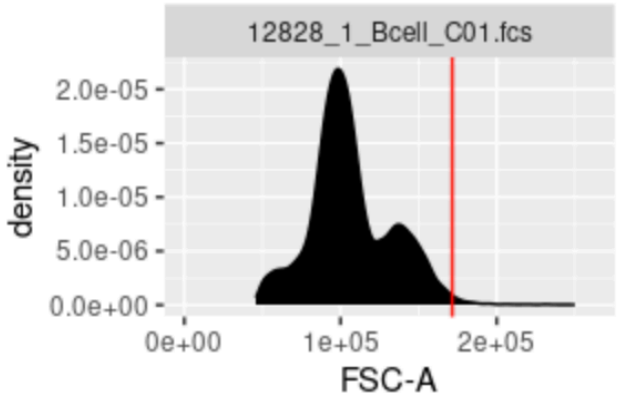
# openCyto: overview of gating methods

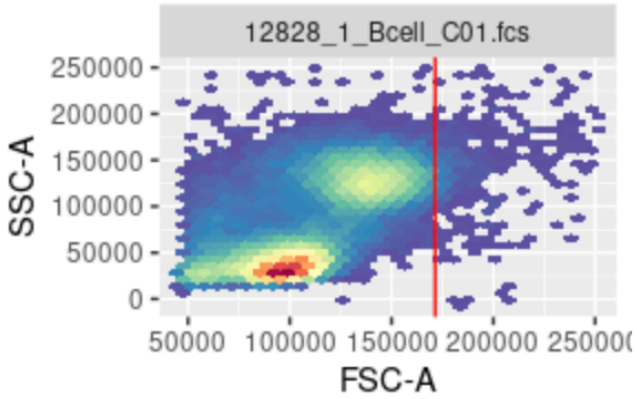
*tailgate*

Gates the right side or left side of the 1-D density based on a cutpoint (estimated) in the tail

*quantileGate*

Alternative to tailgate and it determines the cutpoint by the events quantile



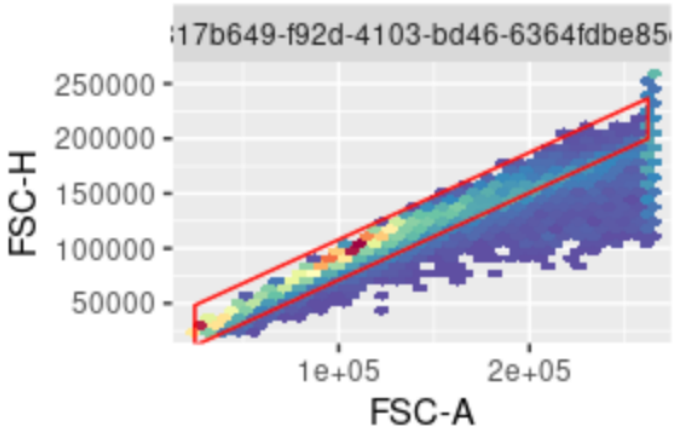

- More commonly used for rare populations (peak is not prominent enough)

# *openCyto*: overview of gating methods

*singletGate*

Use the area vs height to gate

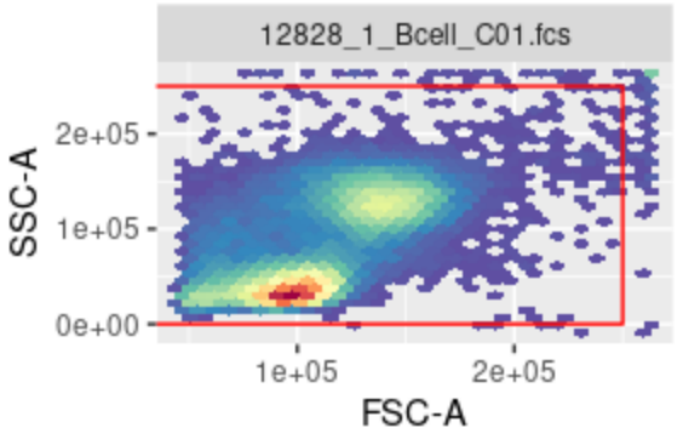
out the singlets




*boundary*

Constructs a rectangle gate

from input range (min and max)

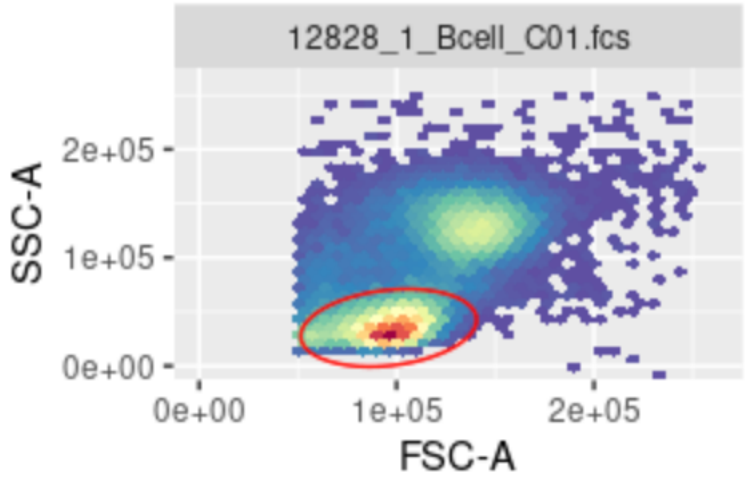



- Used for filtering out very extreme signals at the boundary

# openCyto: overview of gating methods

*flowClust*

1-D or 2-D automated gating methods from flowClust (more details in ?flowClust)




- *k*= how many cell populations are expected
- *target* = center of target population (by default the most prominent cluster)
- *quantile* = how large the ellipse should be

# Let’s practice – 10

In this exercise we will repeat the gating previously done with *flowGate* on the data from **FR_FCM_Z3WR** of the FlowRepository, but this time using automated gating with *openCyto*.

Create a new script in which you will:

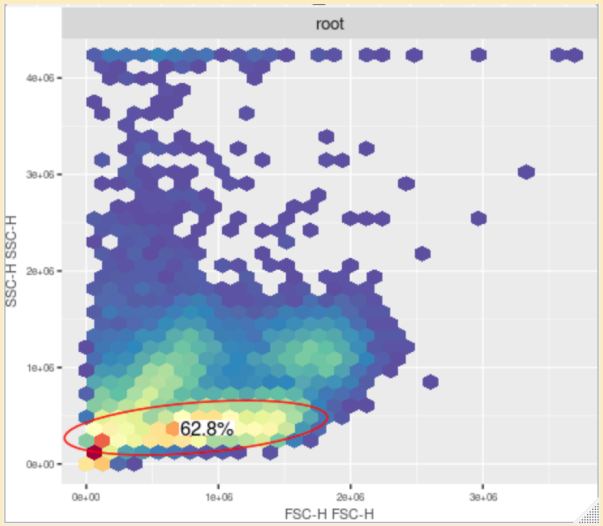
1) Repeat steps 1 to 4 from previous exercise (loading data from fcs files and preprocessing). You can also load the preprocessed GatingSet from /course_dataset/FR_FCM_Z3WR/gs_preprocessed/.
2) Using the *gs_add_gating_method()* function (i.e., without a template), create a gating hierarchy according to the scheme depicted in the following slide. Don't forget to check your gating with scatter or density plots.
3) Do necessary adjustements so that your gating hierarchy looks like the one depicted in the next slide (hide the "BUV805-A+" and "BUV615-A+" nodes from the tree, and rename the CD4, CD8, DNT and DPT nodes )
4) Create boxplots showing the percentage of CD8+ T cells among T cells ("CD3") as a function of time points

# Let’s practice – 10 : Gates

Leukocytes

(FSC-H, SSC-H)

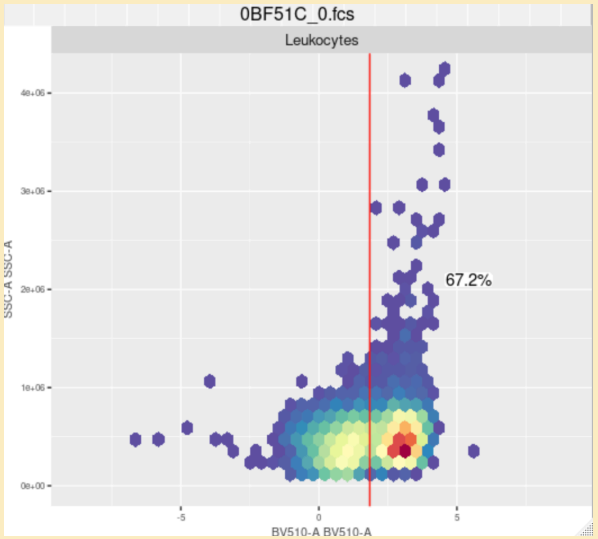
flowClust (K = 3)




CD3 +

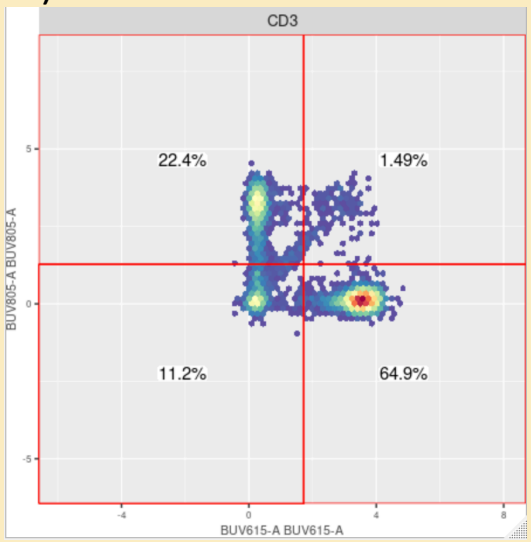
(BV510-A)

gate_mindensity




CD4 CD8

(BUV615-A, BUV805-A)




Final gating hierarchy