Integrating Multiple Samples in Single-cell RNA-seq

The Data Lab

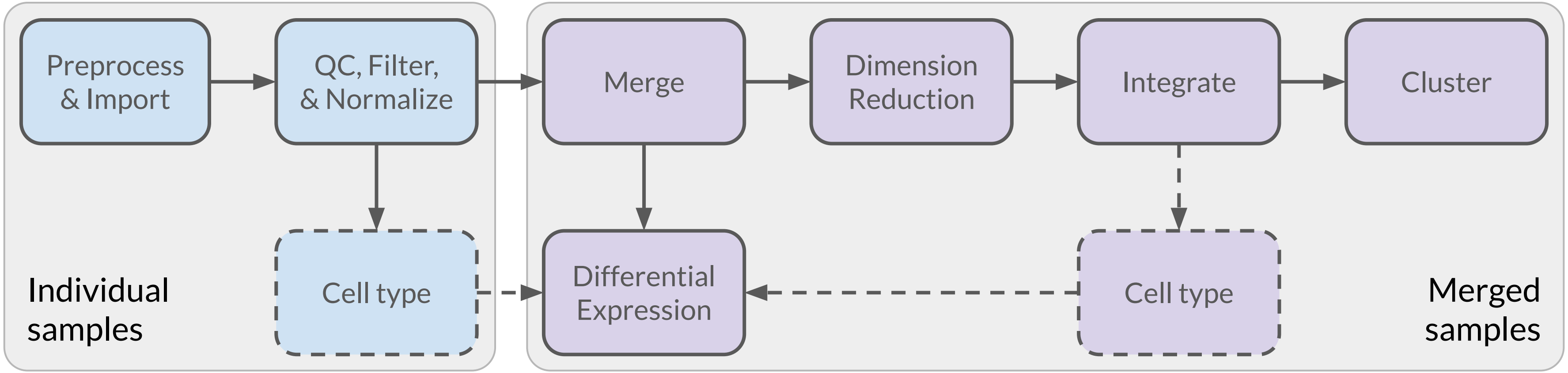
Working with multiple samples in scRNA-seq

Combine and harmonize samples for processing

Identify combined axes of variation across a set of samples

Correct for individual and batch effects among samples

Group cells across samples with similar (corrected) expression patterns



Identify genes with expression differences among samples (within a cell type)

Infer cell labels across samples

Why integrate samples?

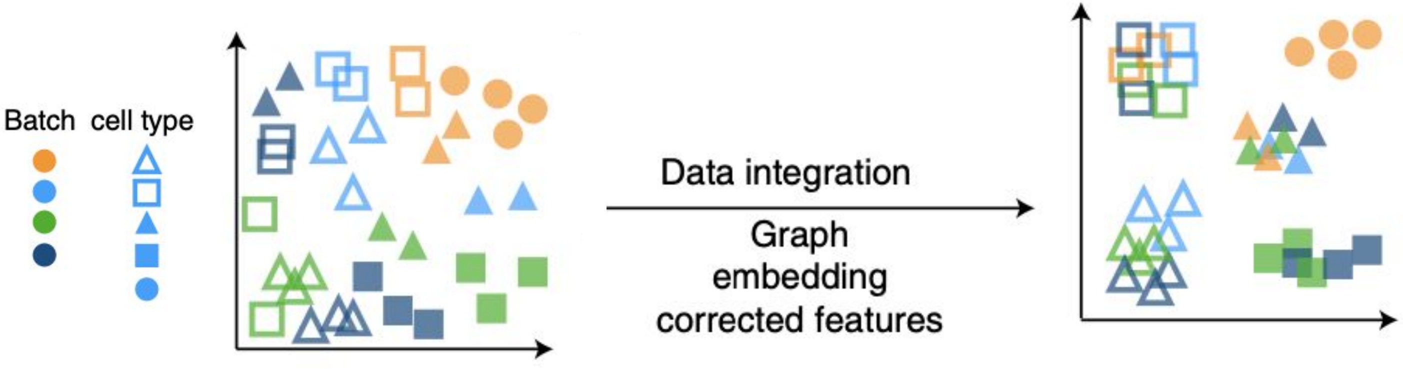
The goal of integration is to mitigate the *batch effects* caused by technical variation across samples, while still preserving biological information.

- Let's call each sample a "batch" of cells
- Cells in a given sample will share some technical variation
- This becomes a problem when we want to jointly consider several samples
  - Cells within a given sample appear more similar than they are, simply because they're from the same sample.
- To compare cells across samples, we need to account for this batch-level technical variation. Then, we can hopefully hone in on the more interesting biological variation 🕵️

What can('t) integration do for you?

- Integration is performed on reduced dimension representations (often principal components)
  - Integration also *returns* reduced dimension representations for downstream use
  - Some integration methods will "back-calculate" corrected gene expression values, but these aren't as important as you think!
  - For example, we do *not* use these for differential expression (stay tuned for more!)
  - Recommended reading on when to use, and not to use, corrected expression values: http://bioconductor.org/books/3.20/OSCA.multisample/using-corrected-values.html
- Integration allows us to...
  - Jointly visualize **cells** from multiple datasets
  - Jointly cluster **cells** from multiple datasets
  - Annotate or identify similar **cell types** across datasets

What does successful integration look like?



Before integration, the primary "clustering" is by batch

- Orange tends to group with orange, green with green, etc.

After successful integration:

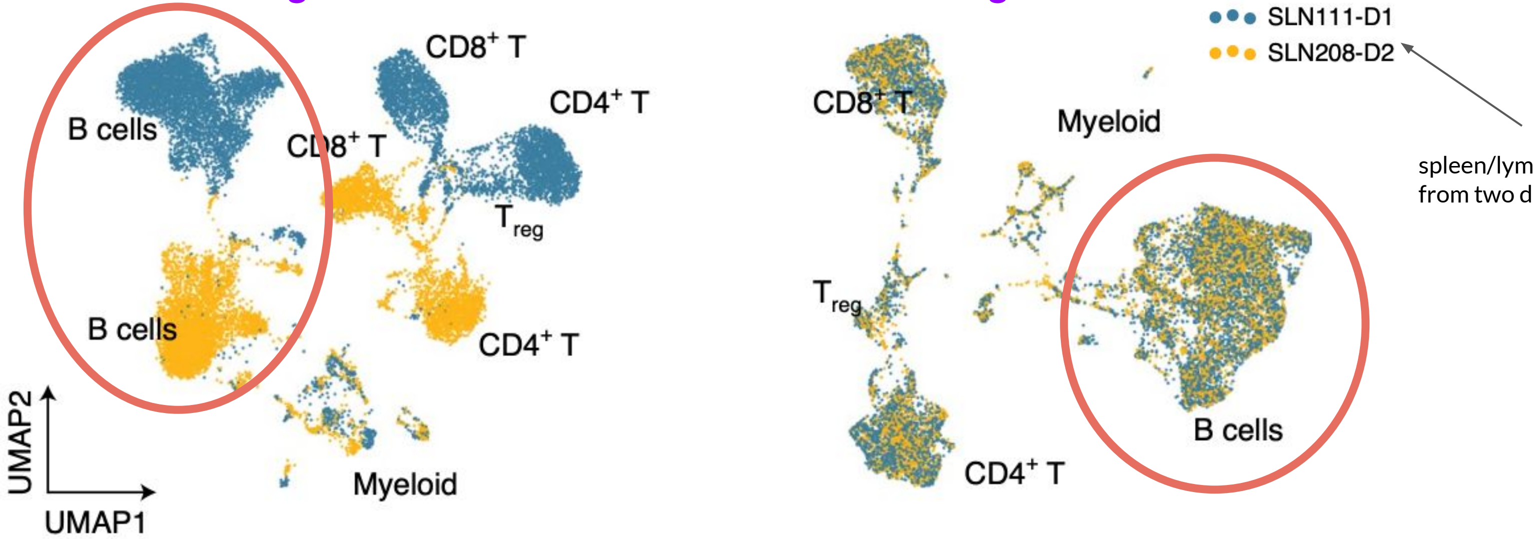
- Batches show lots of mixing
- Cell types ("biology") cluster together, and do *not* show lots of mixing

Successful integration depends on *shared information* across batches.

Figure adapted from Lueken *et al*. (2022) https://doi.org/10.1038/s41592-021-01336-8

Example of (what looks like!) successful integration

Before integration **After integration**



spleen/lymph node samples from two different mice

Gayoso *et al.* (2021) https://doi.org/10.1038/s41592-020-01050-x

How to evaluate integration

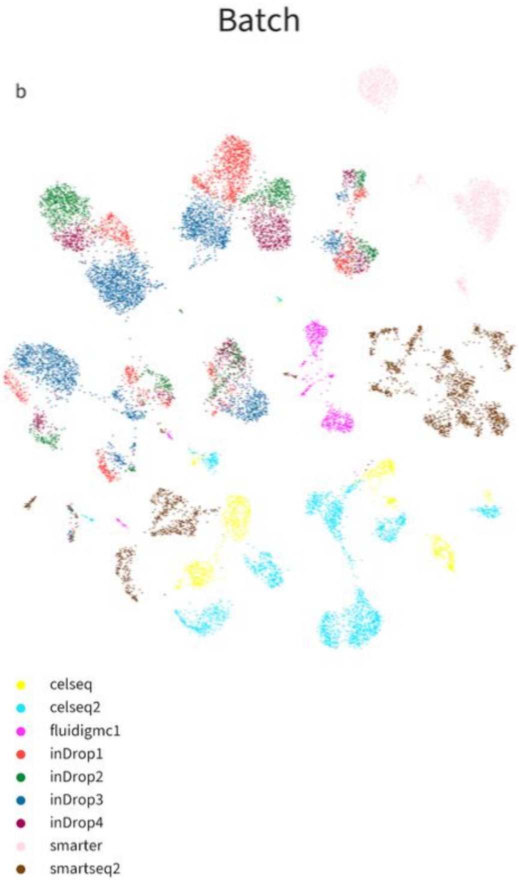
- Compare before and after UMAP vibes

  - Before integration, batches (datasets) will mostly cluster together
  - After integration...
    - Batches should not group together but should be highly mixed across the UMAP
    - Biologically similar cells (tissue, cell type, disease vs healthy) should group together
  - Usually, when it fails, *it fails.*

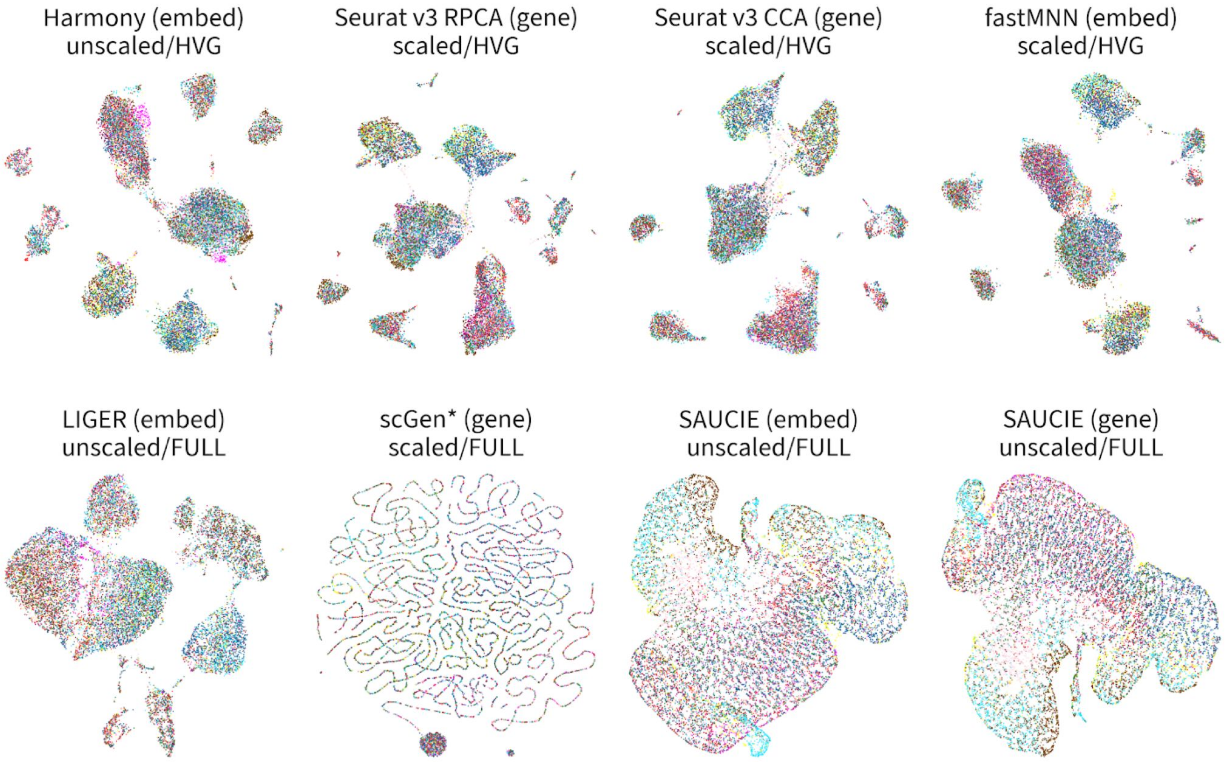
- There are several metrics for evaluating batch correction
  - Luecken *et al.* (2022) is an excellent reference https://doi.org/10.1038/s41592-019-0619-0
  - Caution: Metrics do not measure "was integration successful," but other proxies which sometimes can help us tell if integration was successful (or at least not unsuccessful)

How I stopped worrying and learned to love (the) UMAPs

- Some examples from Luecken et al. (2022)
    - Luecken, M.D., Büttner, M., Chaichoompu, K. et al. Benchmarking atlas-level data integration in single-cell genomics. (2022). https://doi.org/10.1038/s41592-021-01336-8
    - Panels from Figure S13 are shown on the next two slides
        - https://static-content.springer.com/esm/art%3A10.1038%2Fs41592-021-01336-8/MediaObjects/41592_2021_1336_MOESM1_ESM.pdf



Top 4 "best" integration methods


Harmony (embed) unscaled/HVG

Seurat v3 RPCA (gene) scaled/HVG

Seurat v3 CCA (gene) scaled/HVG

fastMNN (embed) scaled/HVG

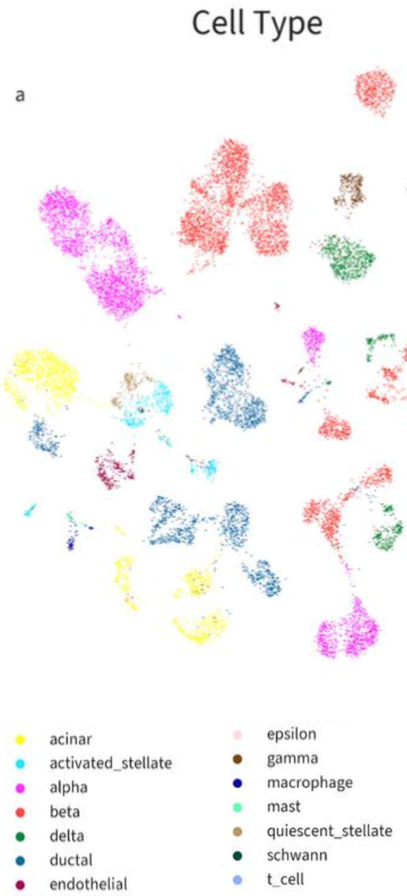
LIGER (embed) unscaled/FULL

scGen* (gene) scaled/FULL

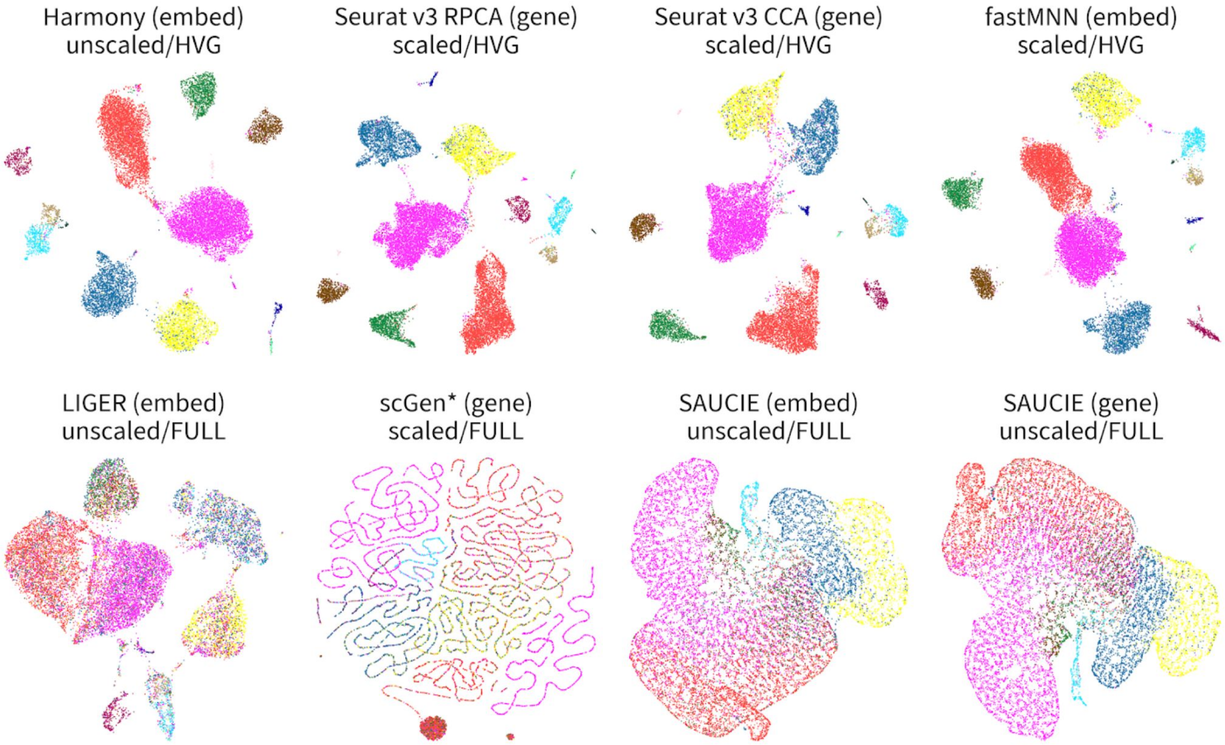
SAUCIE (embed) unscaled/FULL

SAUCIE (gene) unscaled/FULL

Bottom 4 "worst" integration methods



Top 4 "best" integration methods

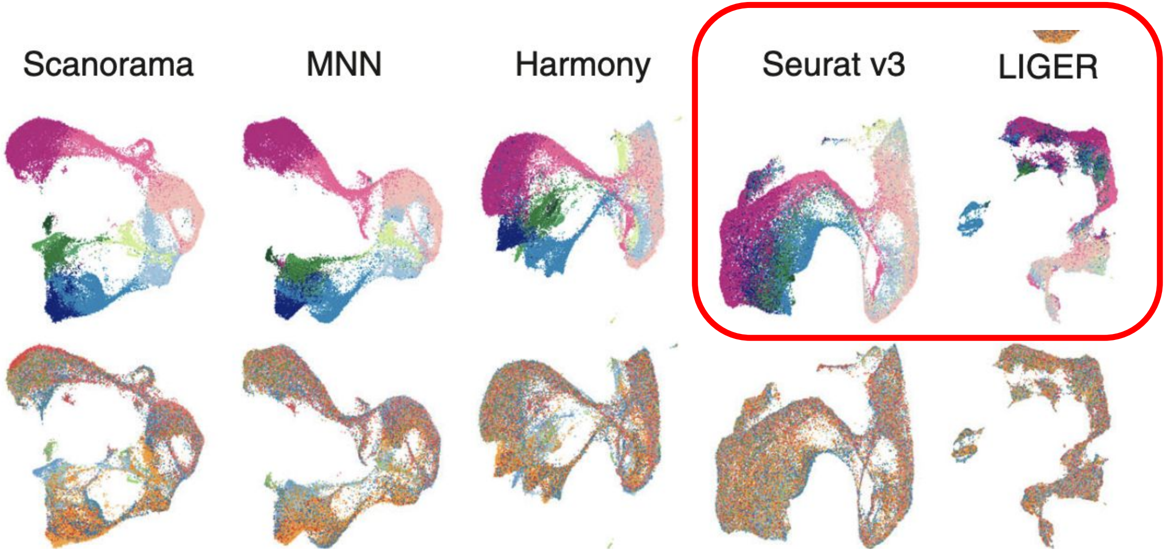


Bottom 4 "worst" integration methods

Over-correction leads to loss of distinct biology

Cell types

Sample



Integration of cerebral organoids generated from seven different human PSC lines

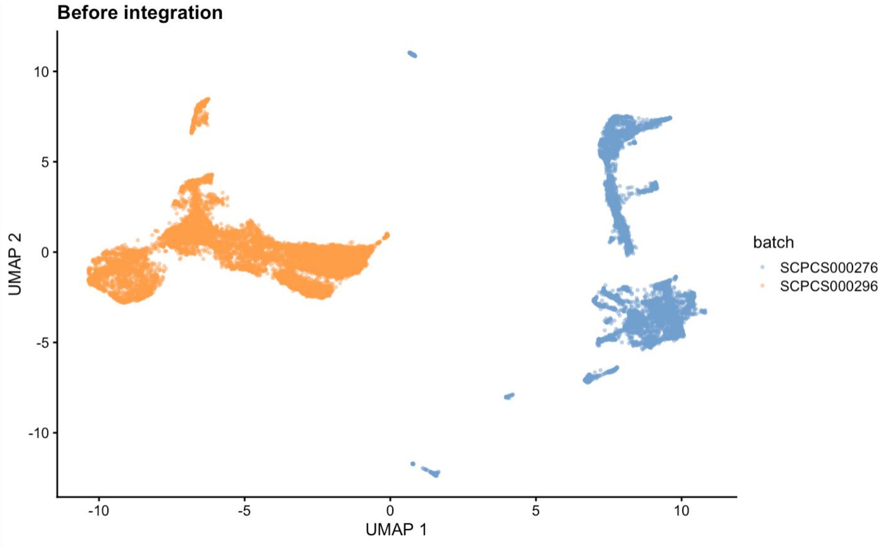
https://doi.org/10.1186/s13059-020-02147-4

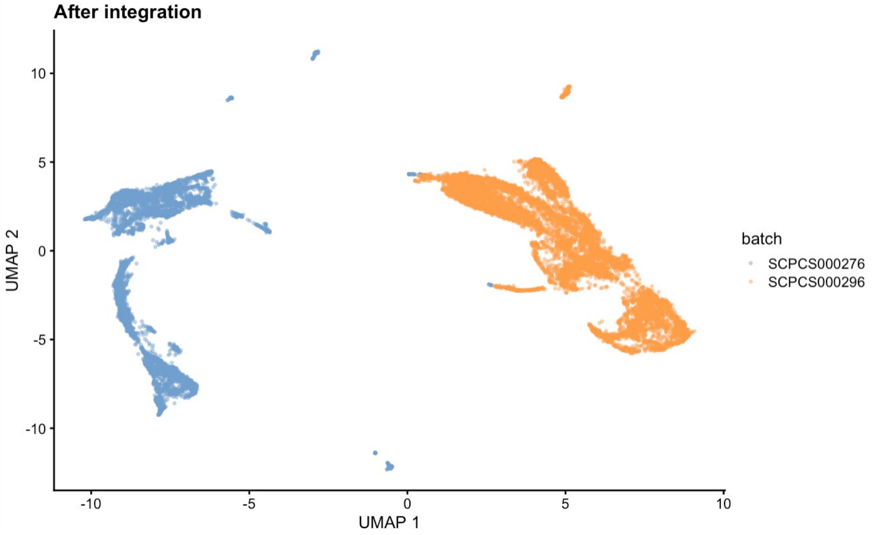
Will it integrate?

- Datasets that don't have shared cell types or states will be hard to integrate
  - Patient and xenograft
  - Tumor and normal
  - Data from different tissue types
  - Data from different organisms

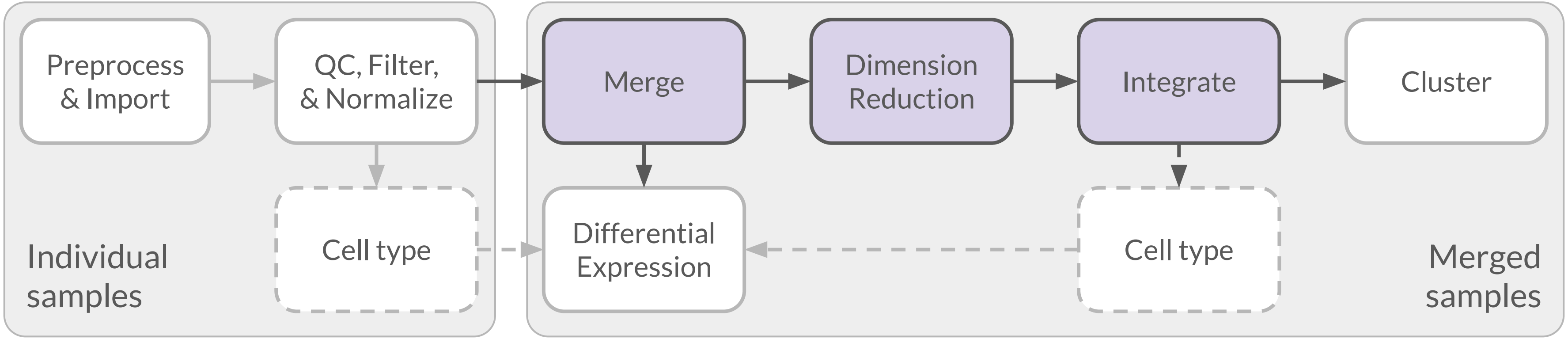
- The extent of "overlap" among datasets may also influence which integration method you should use, along with the results themselves

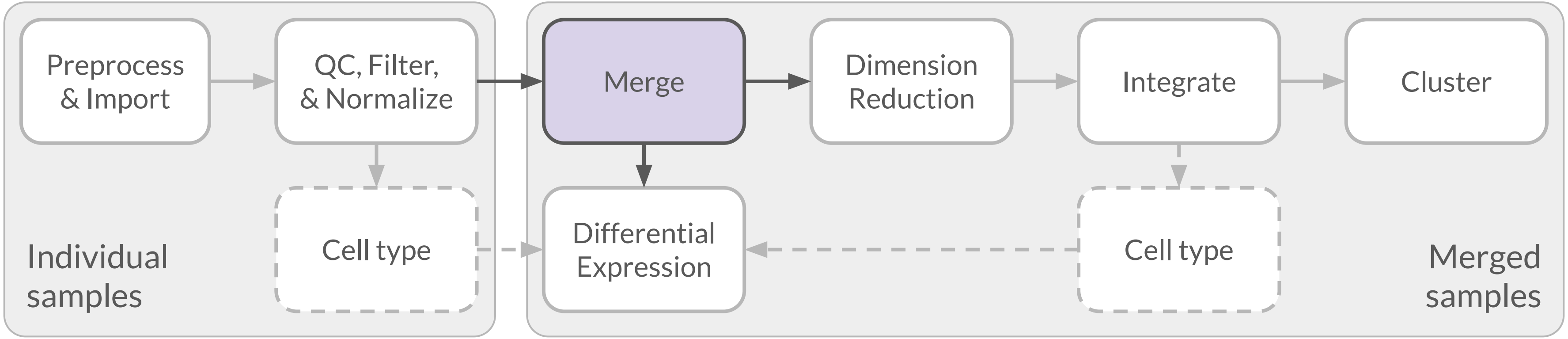
An example of failed integration





Integration in scRNA-seq overview





Preprocess & Import

QC, Filter, & Normalize

Cell type

Individual samples

Merge

Dimension Reduction

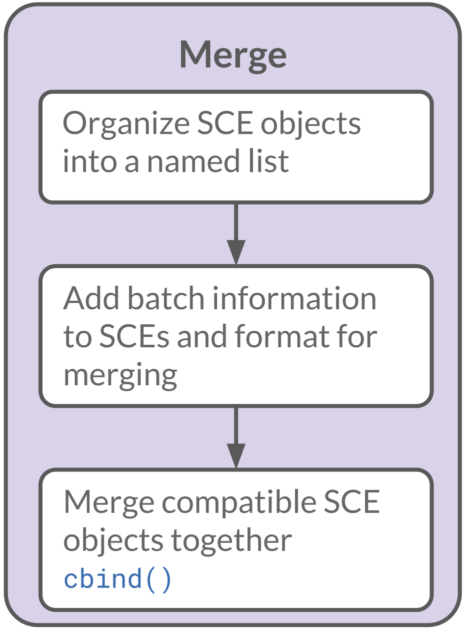
Integrate

Cluster

Differential Expression

Cell type

Merged samples



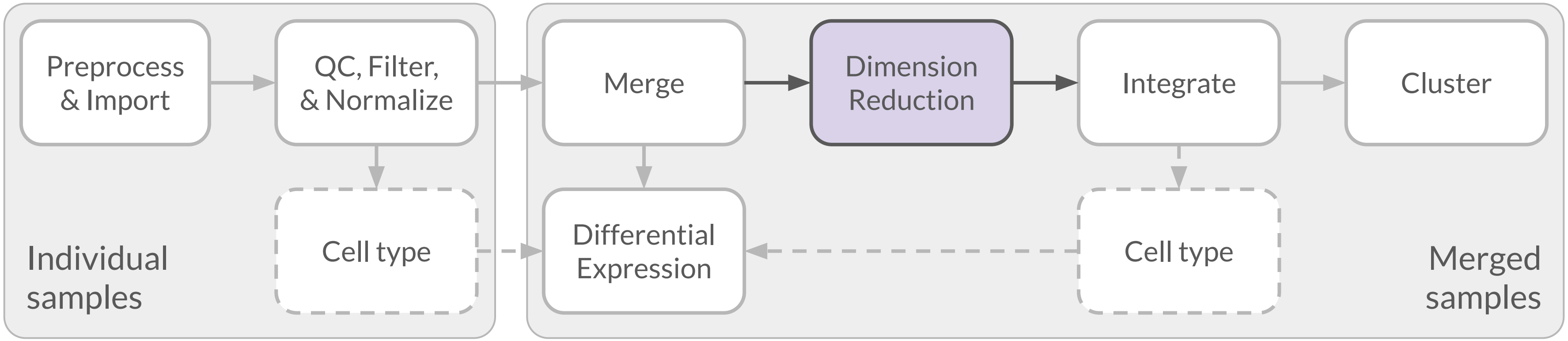
Merge

Organize SCE objects into a named list

Add batch information to SCEs and format for merging

Merge compatible SCE objects together `cbind()`

- It's useful to *merge SCEs together* for many downstream analyses, but this merging requires some bookkeeping:
  - After merging, how can we still tell which batch (sample) each cell came from?
    - We need to add this information into SCEs
  - Are SCEs formatted such that R will let us merge them?
    - They need to have compatible column and row names



Preprocess & Import

QC, Filter, & Normalize

Cell type

Individual samples

Merge

Dimension Reduction

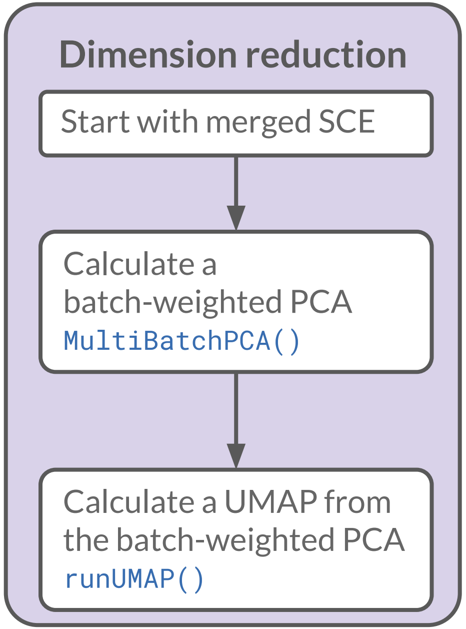
Integrate

Cluster

Differential Expression

Cell type

Merged samples



Dimension reduction

Start with merged SCE

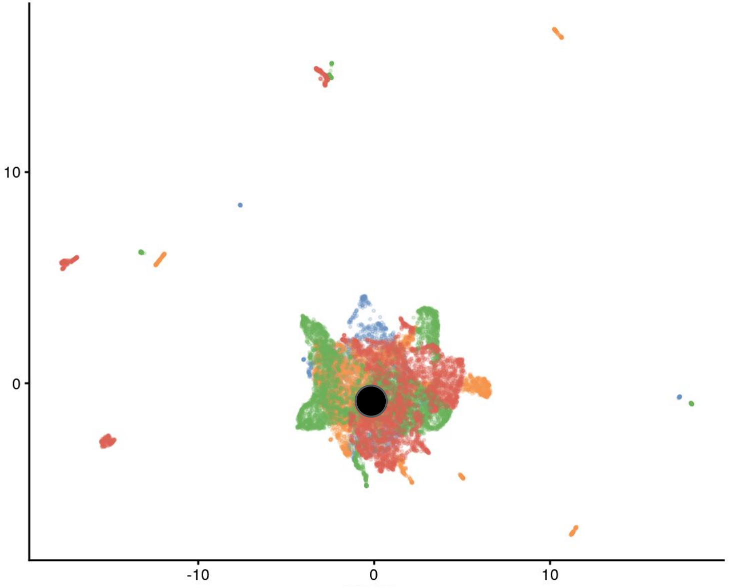
Calculate a batch-weighted PCA `MultiBatchPCA()`

Calculate a UMAP from the batch-weighted PCA `runUMAP()`

- Dimension reduction techniques like PCA and UMAP start by scaling data to be centered at 0.
- To use PCA/UMAP across samples, we need to calculate the variation jointly. Otherwise, we will be misled!

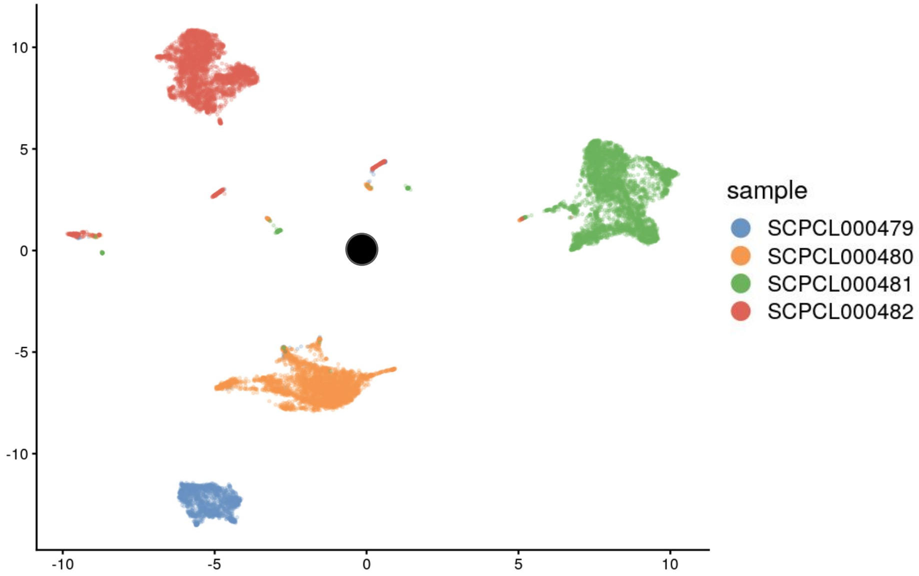
● the (0, 0) coordinate

PCA/UMAP calculated separately on each sample

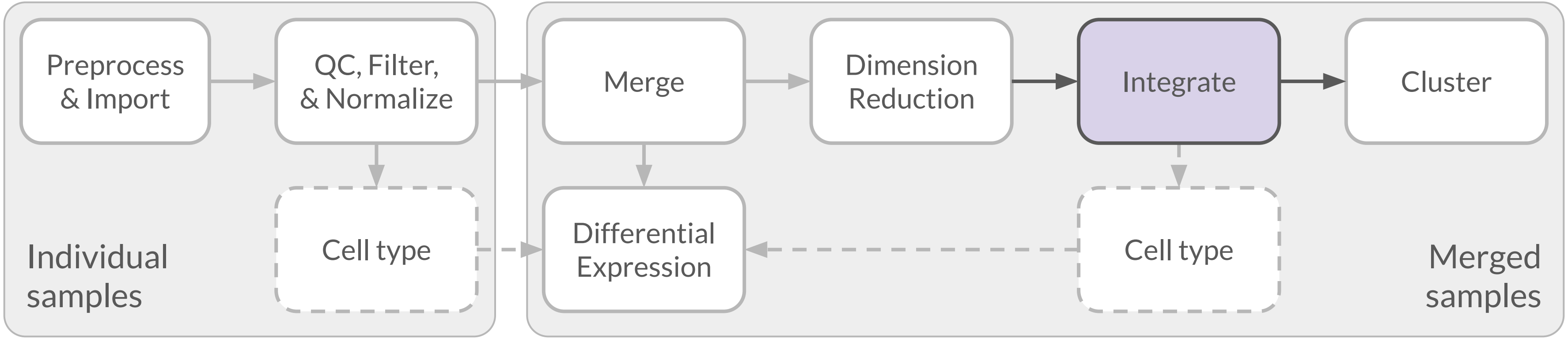


This looks integrated **but it's not!!**

PCA/UMAP calculated jointly on all samples together



This is your "before" UMAP



Preprocess & Import

QC, Filter, & Normalize

Cell type

Individual samples

Merge

Dimension Reduction

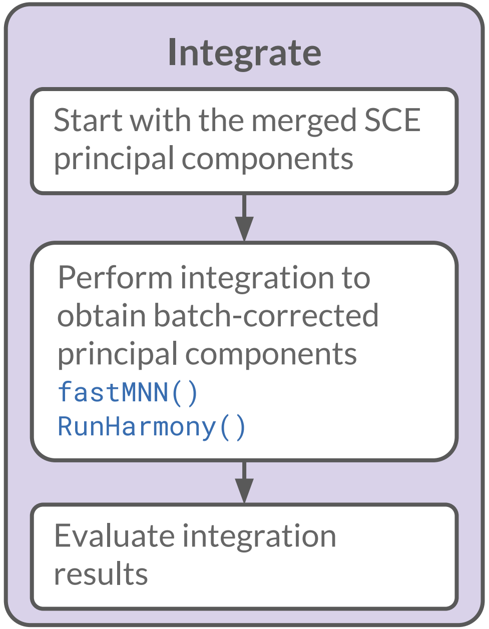
Integrate

Cluster

Differential Expression

Cell type

Merged samples



Integrate

Start with the merged SCE principal components

Perform integration to obtain batch-corrected principal components
`fastMNN()`
`RunHarmony()`

Evaluate integration results

- We can evaluate results by…
  - Comparing before/after UMAPs
  - Calculating metrics that tell us how well cells and batches mix
- If integration is successful, we should see…
  - Batches are well-mixed across the UMAP
  - Cell types (or similar biological grouping, if known) group together separately
  - Remember: Success depends on overlap among batches

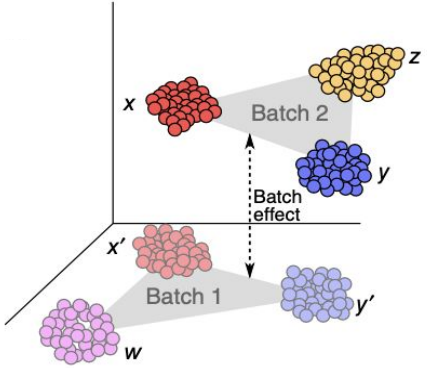
Let's have a closer look at methods we'll be using

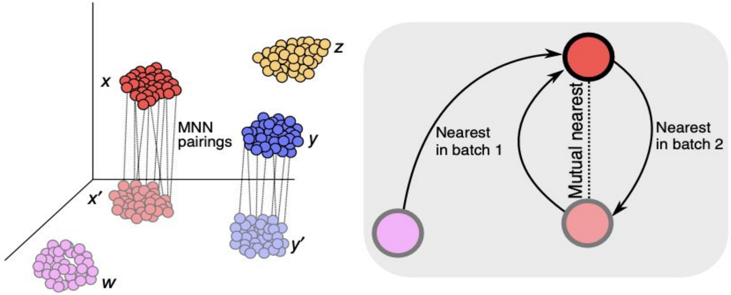
- **MNN**: Mutual nearest neighbors
  - Specifically, we'll use FastMNN 🚀
  - Haghverdi, L, Lun, A, Morgan, M, *et al. Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors.* (2018) https://doi.org/10.1038/nbt.4091

- **Harmony**
  - Korsunsky, I, Millard, N, Fan, J, *et al. Fast, sensitive and accurate integration of single-cell data with Harmony*. (2019) https://doi.org/10.1038/s41592-019-0619-0

Mutual nearest neighbors batch correction

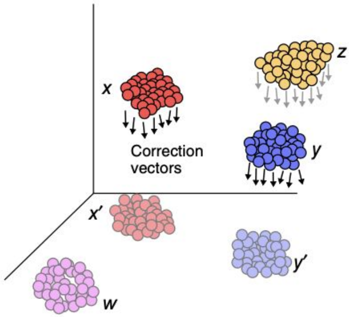
- Imagine we have 2 batches, each with 3 cell types
  - Red (x) and blue (y) are shared but pink (w) and yellow (z) are not!
  - Before beginning integration, cosine distances are first calculated among pairs of cells *within each sample*
  - This enables expression profile comparisons and sets up the data for integration
- First, we identify pairs of cells with mutually similar expression profiles
  - These are our "mutual nearest neighbors"



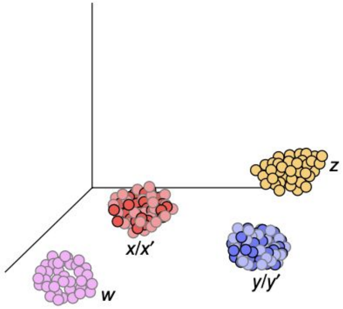


Mutual nearest neighbors batch correction

- Next, compute a batch correction vector for each MNN pair



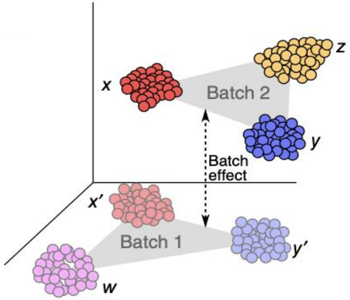
- Finally, calculate the weighted average of these vectors to get cell-specific batch corrections to perform the final integration
  - Note that **w** and **z** don't "look" as "integrated"! Why?



Some assumptions that MNN makes

- At least one cell population is present in both batches

- The batch effect is almost orthogonal to the biological effects
    - Roughly means, batches and biology are expected to have *separate variation*

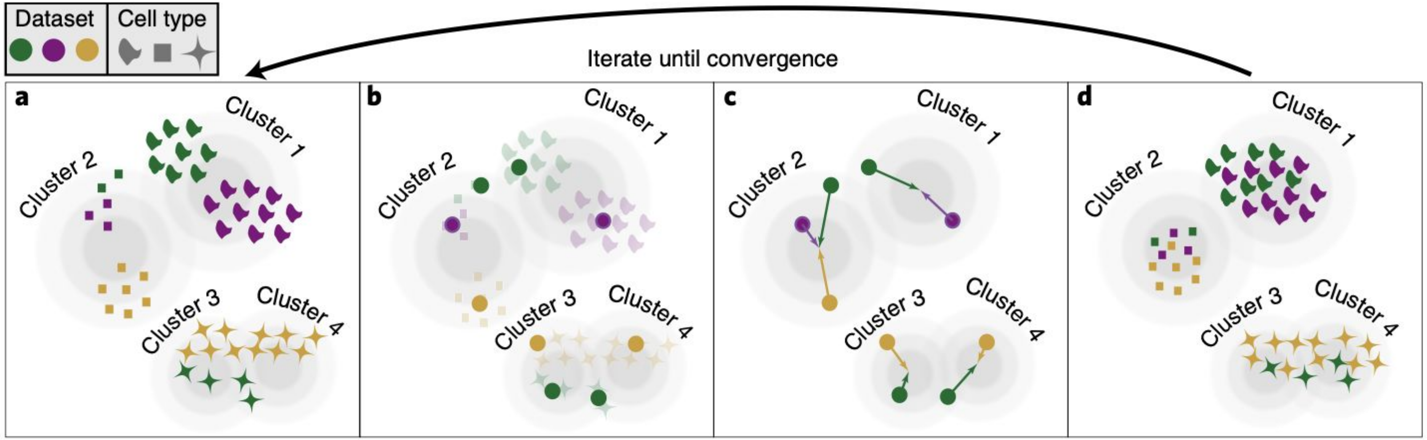


- The batch-effect variation is much smaller than the biological-effect variation across cell types

Harmony batch correction

- "Soft k-means clustering algorithm"

Input PCA



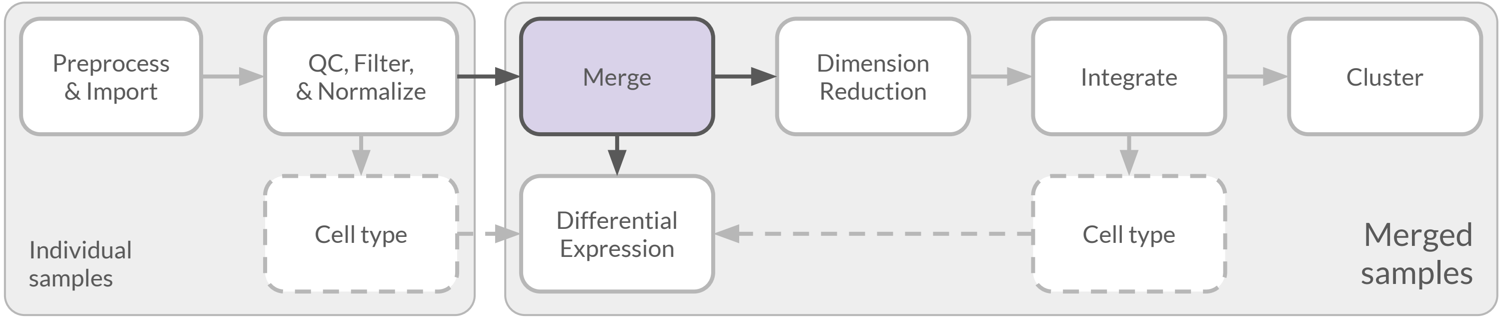
Soft assign cells to clusters, favoring mixed dataset representation

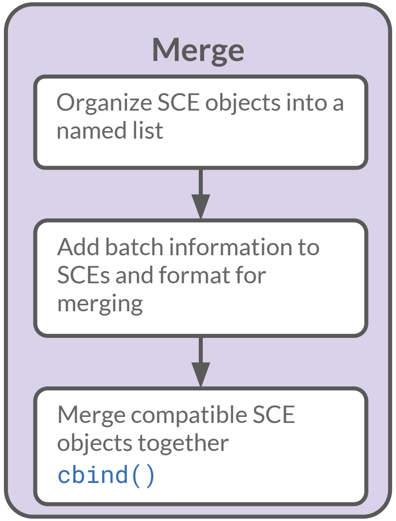
Get cluster centroids for each dataset

Get dataset correction factors for each cluster

Move cells based on soft cluster membership

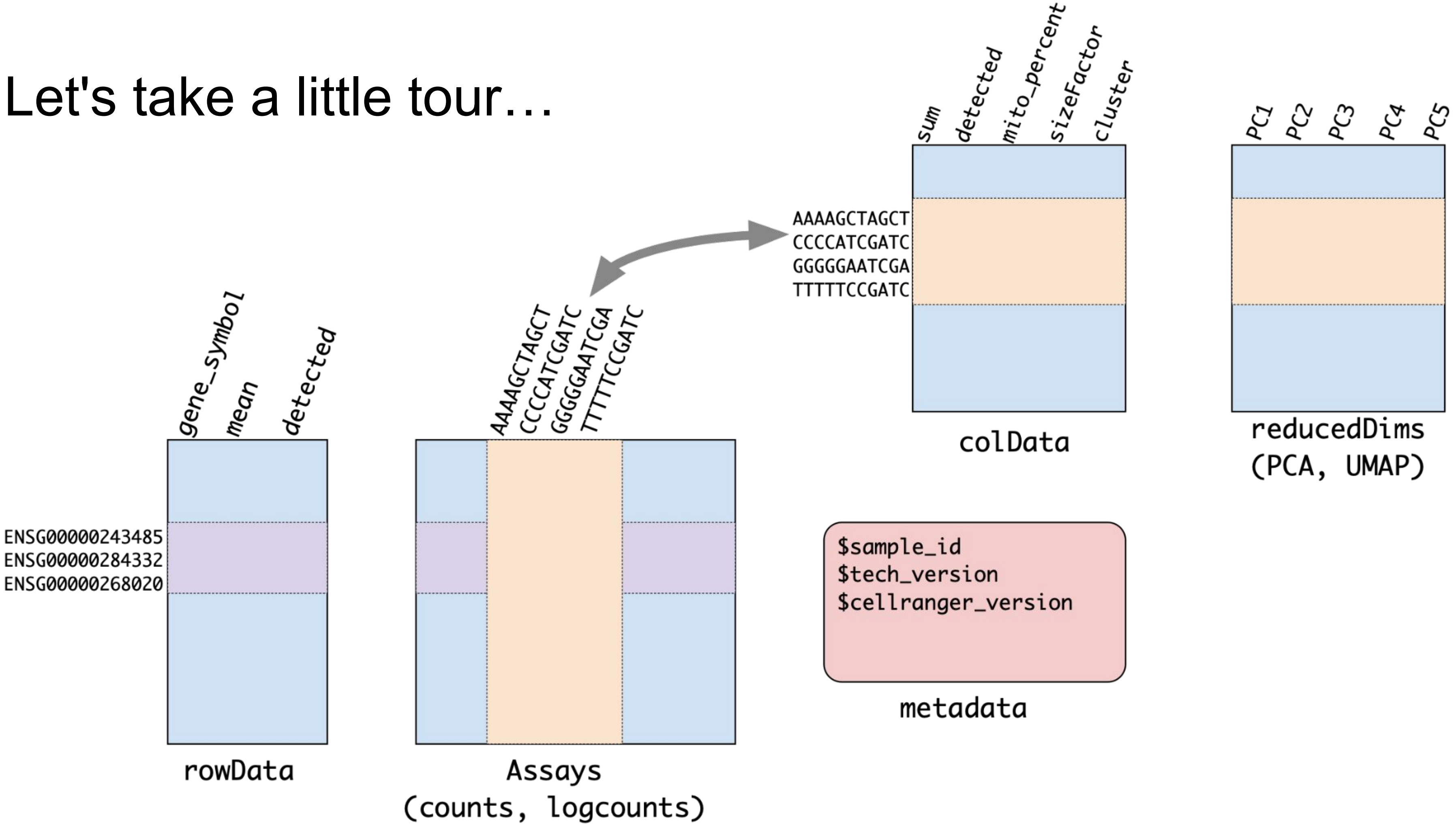
Before we dive in, let's see how some of this sausage gets made



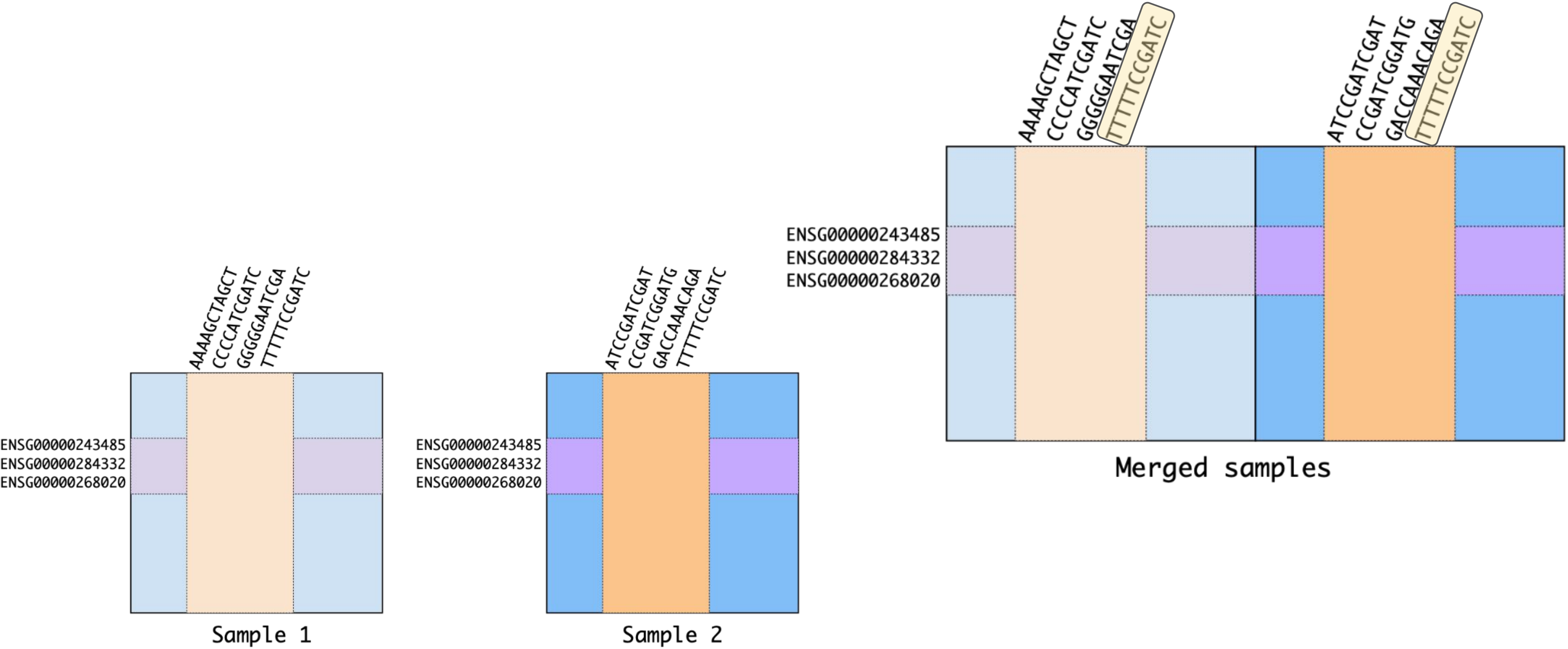

Merge
Organize SCE objects into a named list
Add batch information to SCEs and format for merging
Merge compatible SCE objects together
cbind()

- It's useful to *merge SCEs together* for many downstream analyses, but this merging requires some bookkeeping:
  - After merging, how can we still tell which batch (sample) each cell came from?
    - We need to add this information into SCEs
  - Are SCEs formatted such that R will let us merge them?
    - They need to have compatible column and row names

Let's take a little tour…

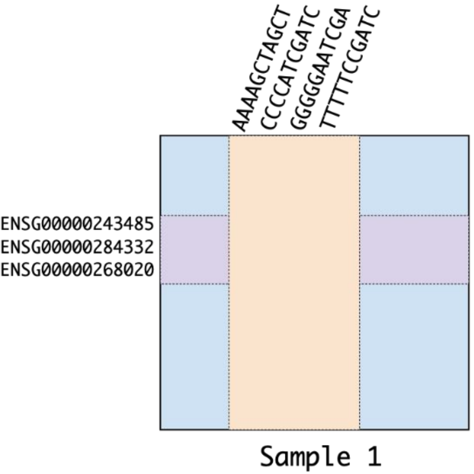


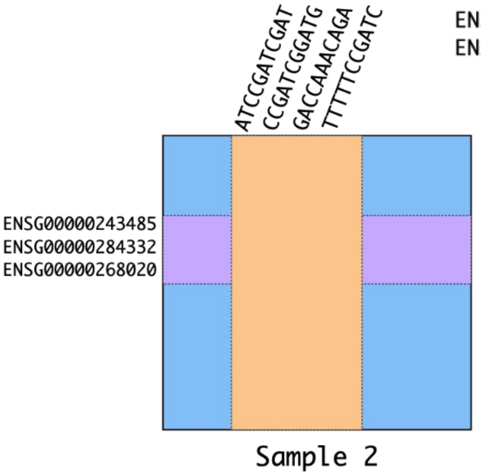
We'll use `cbind()` to merge objects to combine column-wise

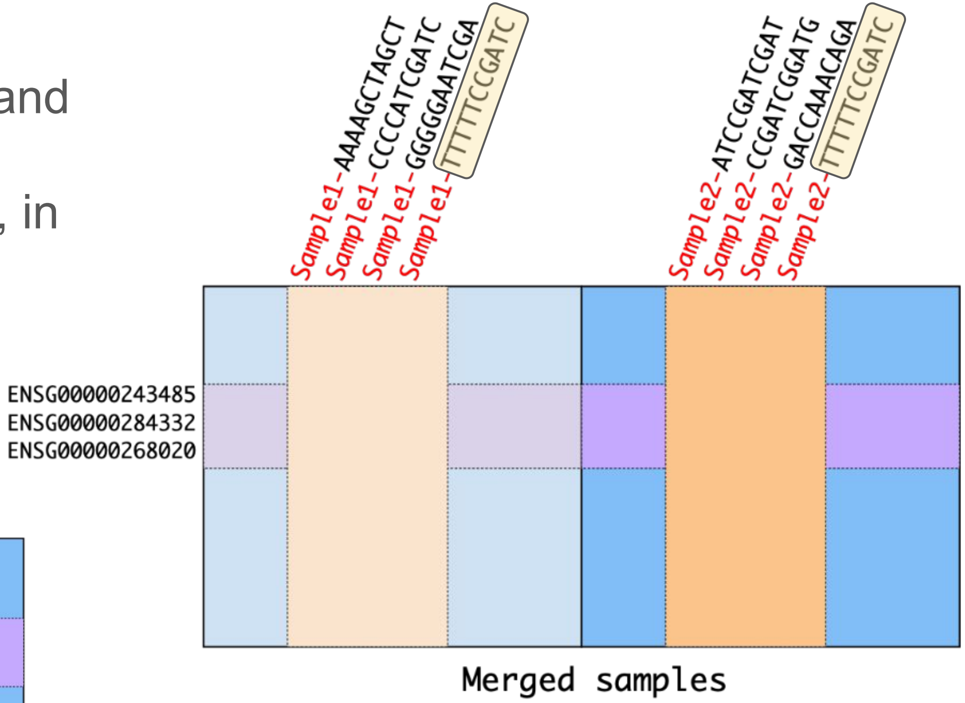


Keep cells unique and identifiable by attaching the sample id

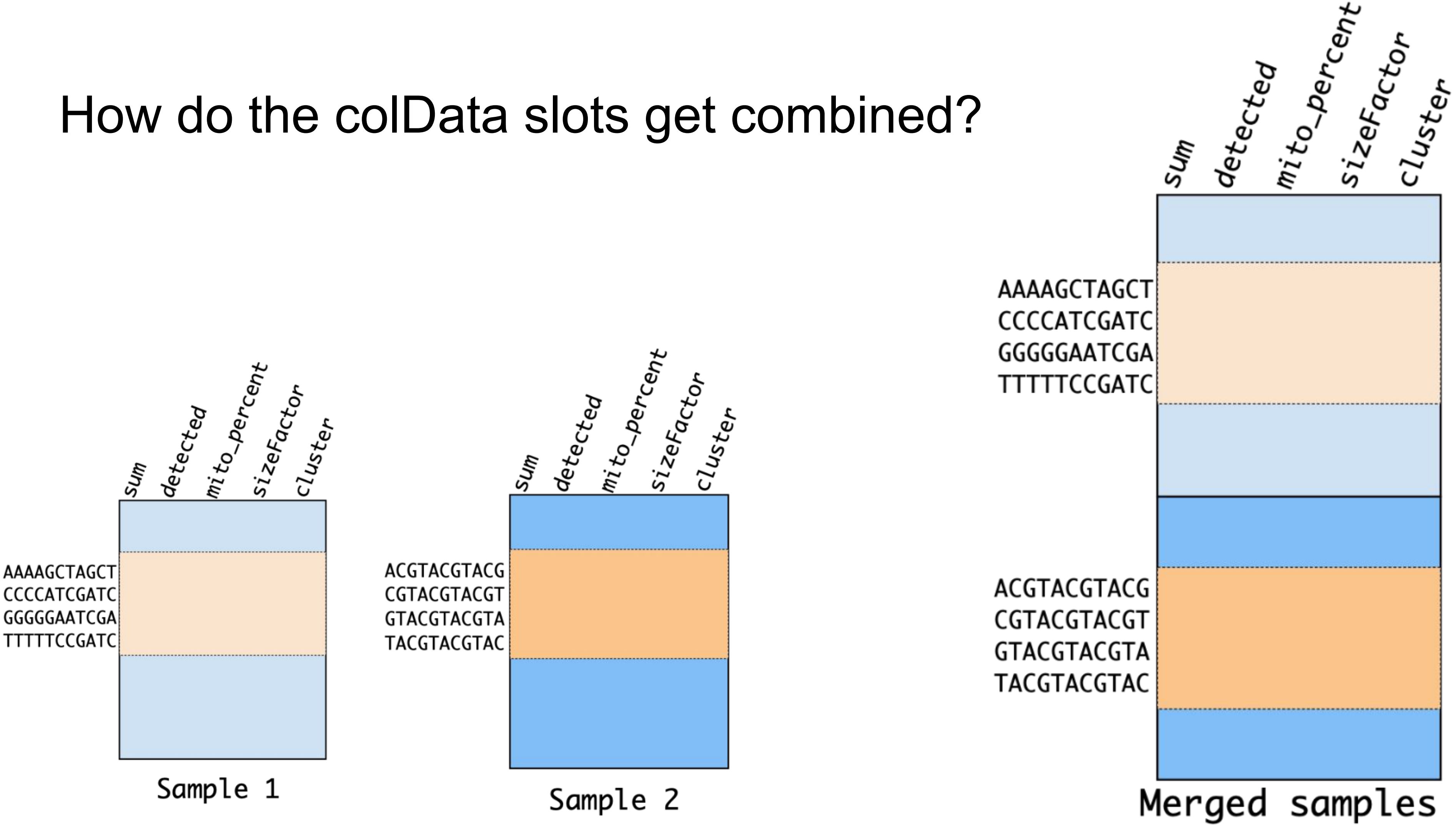
- ➔ Column names (cells) need to be unique and identifiable
- ➔ Row names (genes) need to be the same, in the same order



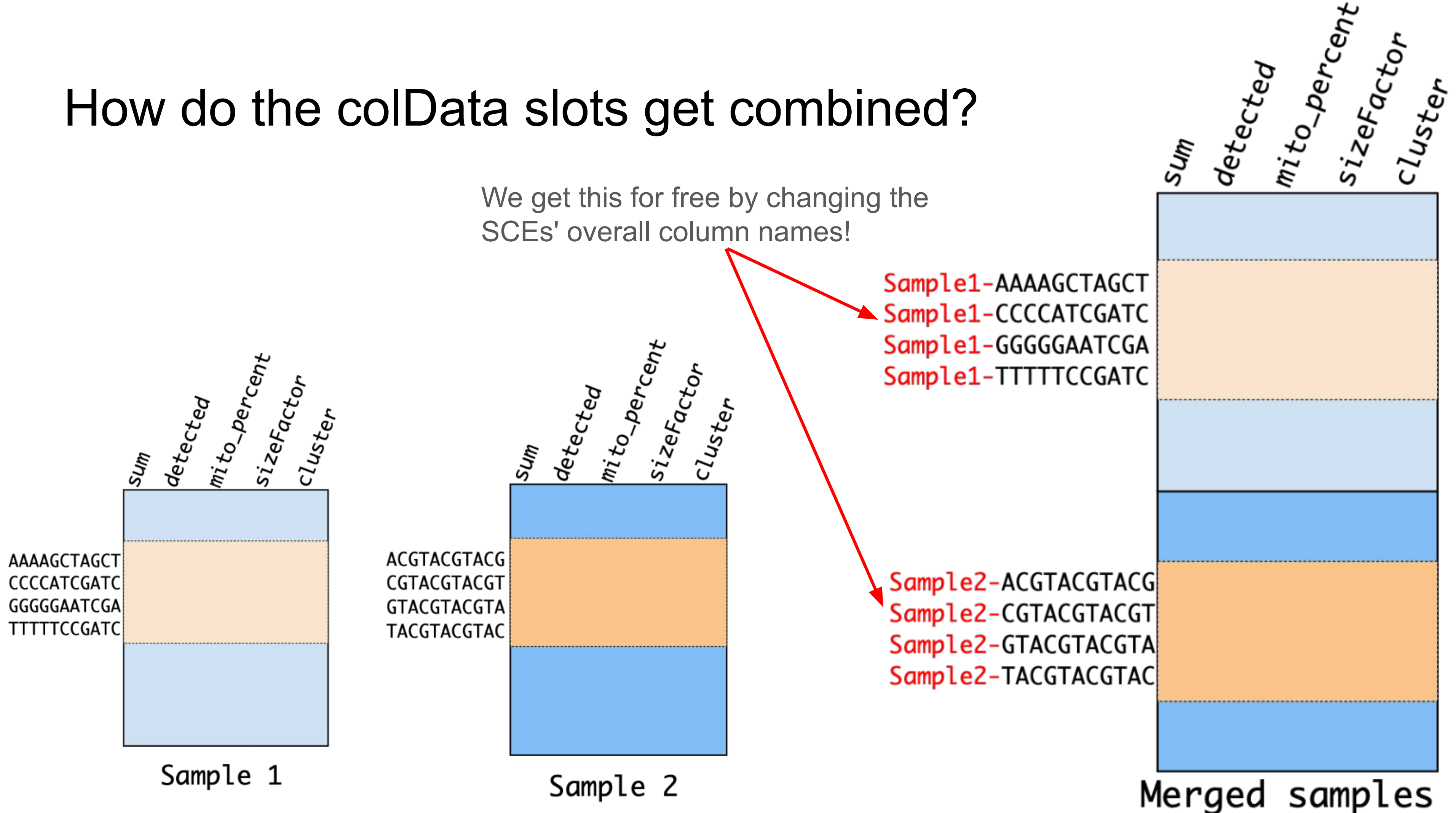




How do the colData slots get combined?



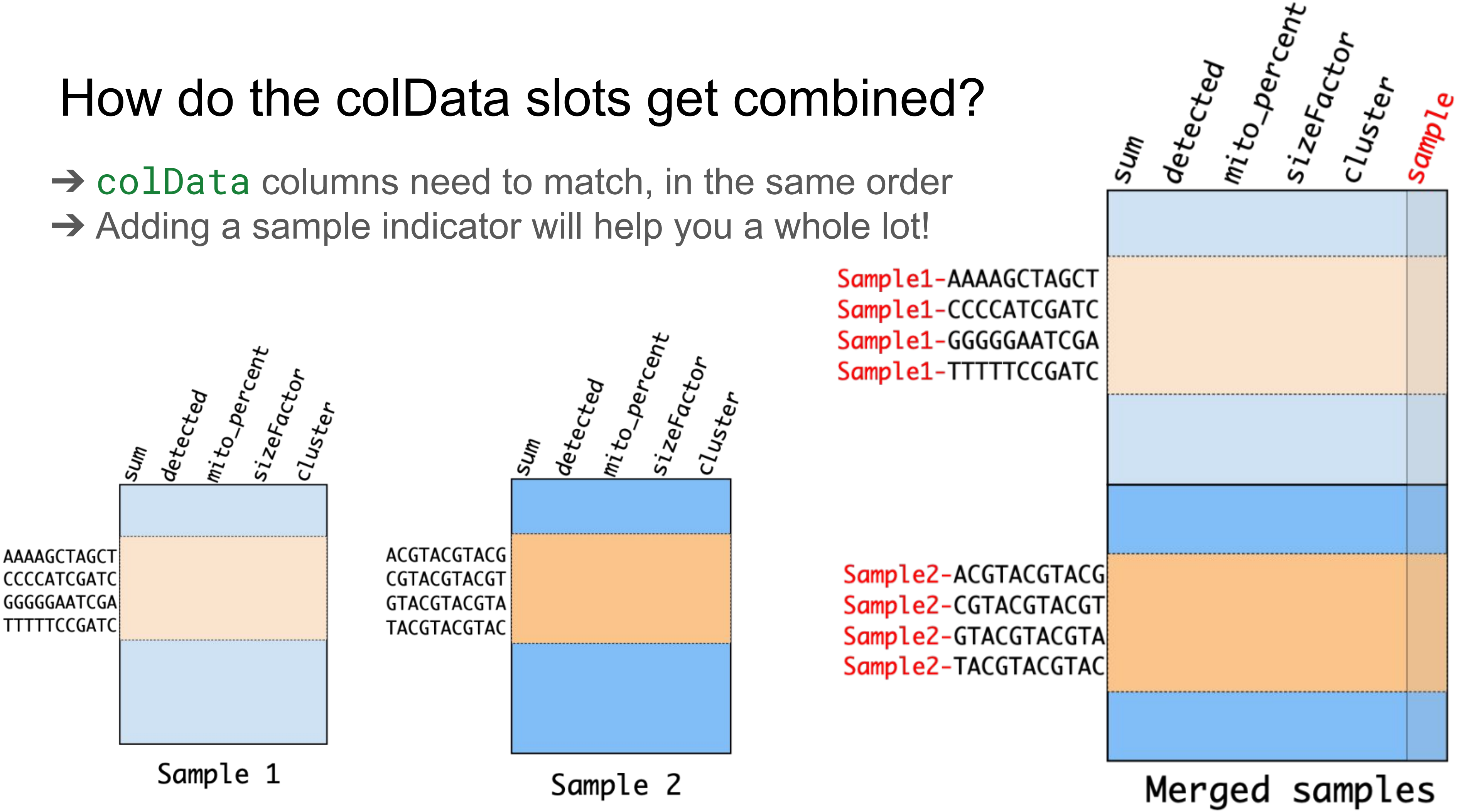
How do the colData slots get combined?



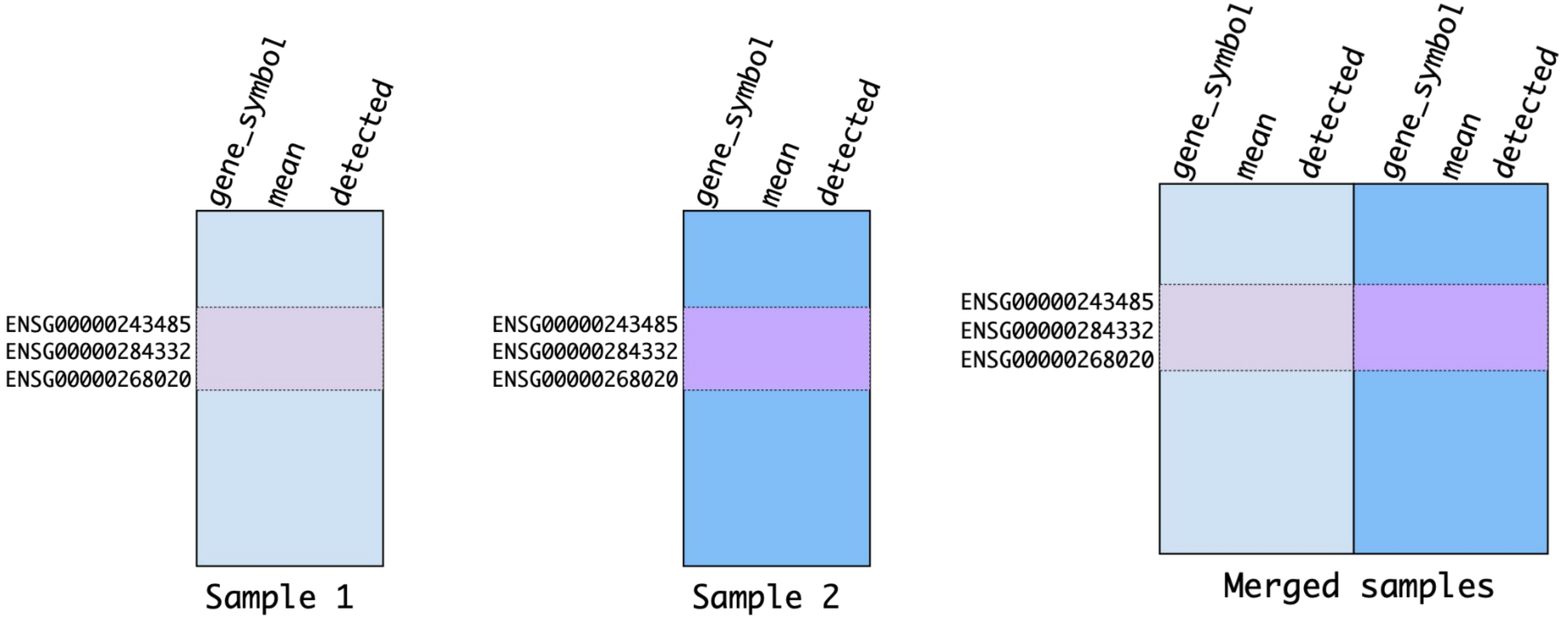
We get this for free by changing the SCEs' overall column names!

How do the colData slots get combined?

- ➔ colData columns need to match, in the same order
- ➔ Adding a sample indicator will help you a whole lot!

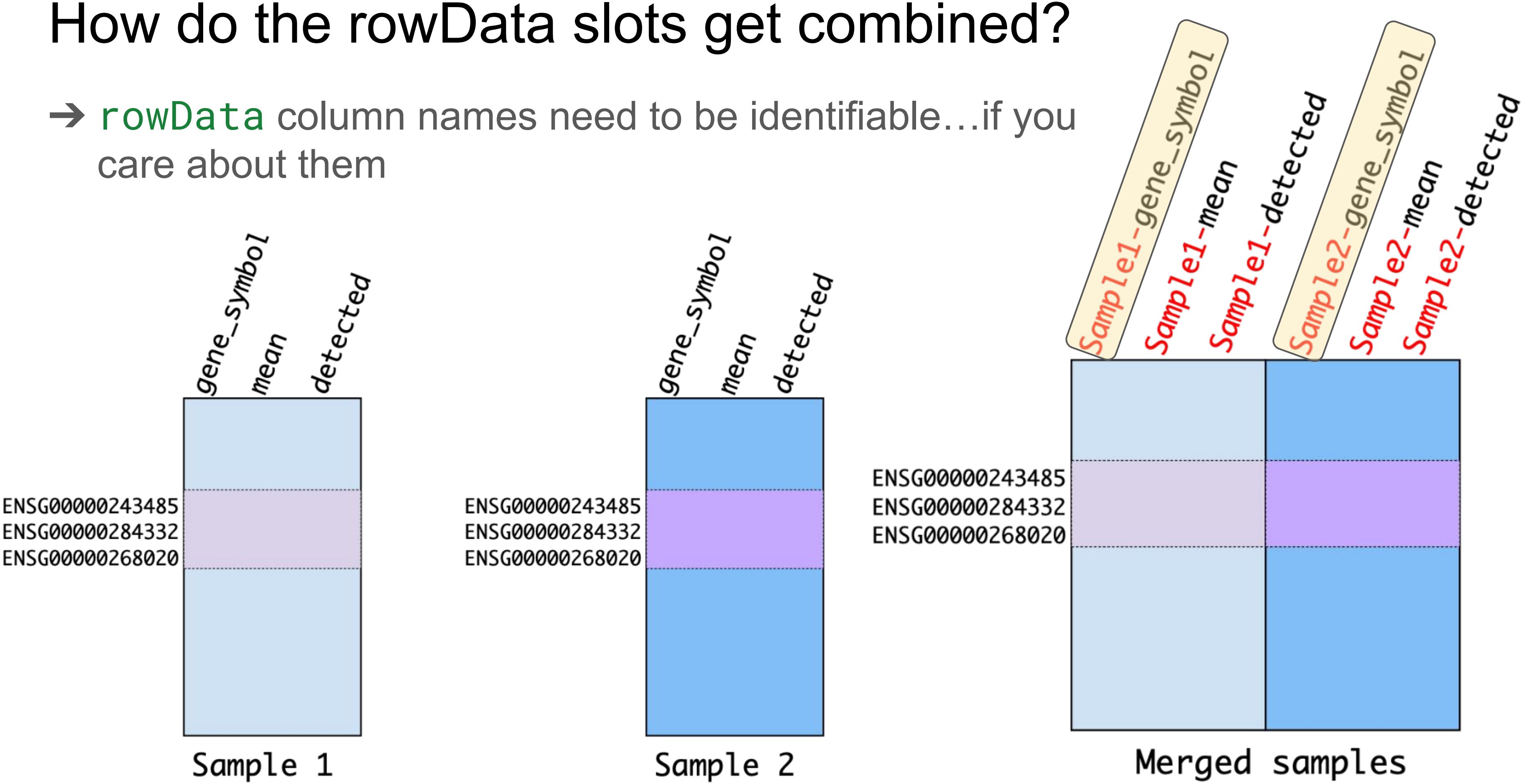


How do the rowData slots get combined?

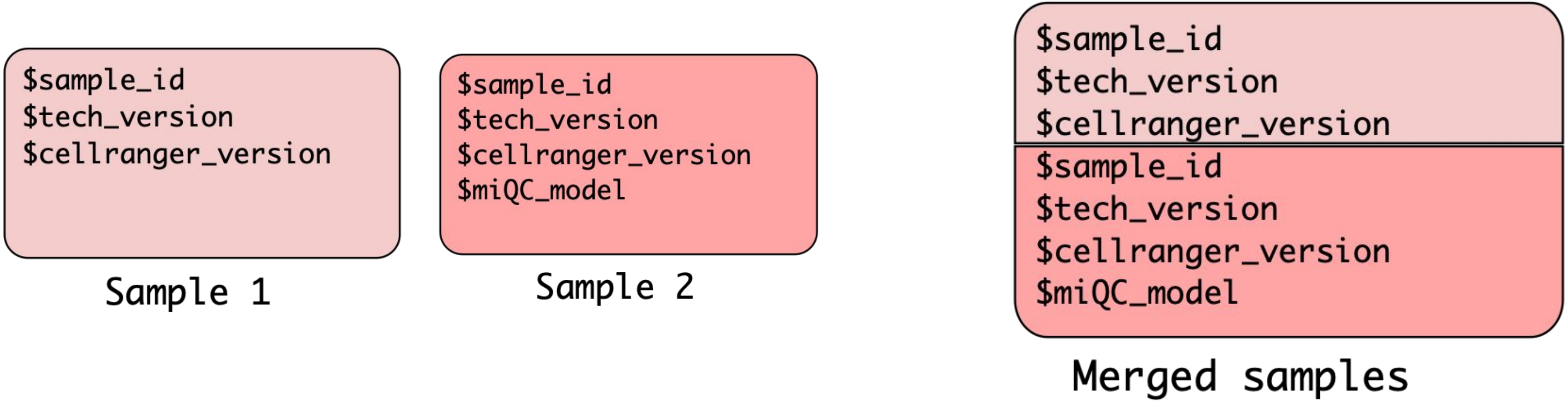


How do the rowData slots get combined?

➔ `rowData` column names need to be identifiable…if you care about them



How do the metadata slots combined?



$sample_id
$tech_version
$cellranger_version

Sample 1

$sample_id
$tech_version
$cellranger_version
$miQC_model

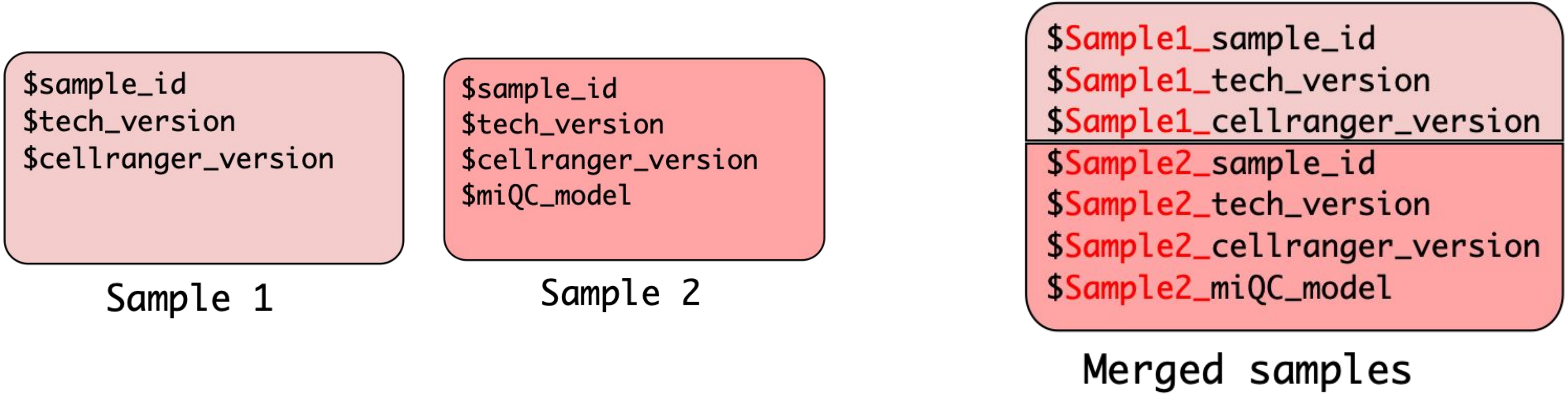
Sample 2

$sample_id
$tech_version
$cellranger_version
$sample_id
$tech_version
$cellranger_version
$miQC_model

Merged samples

How do the metadata slots combined?

➔ `metadata` field names need to be identifiable…if you care about them



```
$sample_id
$tech_version
$cellranger_version
```

Sample 1

```
$sample_id
$tech_version
$cellranger_version
$miQC_model
```

Sample 2

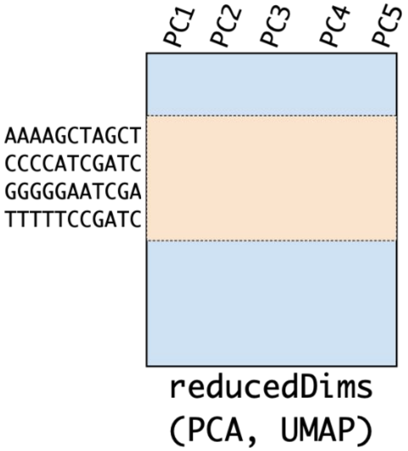
```
$Sample1_sample_id
$Sample1_tech_version
$Sample1_cellranger_version
$Sample2_sample_id
$Sample2_tech_version
$Sample2_cellranger_version
$Sample2_miQC_model
```

Merged samples

One slot left…what about the `reducedDims`?

- We'll need to recalculate PCA/UMAP on the merged object taking batches into consideration, so we're actually going to remove these entirely!

- But, if you wanted to keep them, they also have to be compatible!
  - Make sure to keep track of the fact that they were run on *individual objects before merging* so you don't get mixed up!



How to format your SCE objects *before merging*

- Keep your samples identifiable
  - Barcodes (column names) should indicate the sample id
  - `colData` should have a new sample id column
  - `rowData` column names should indicate the sample id (if you care)
  - `metadata` field names should indicate the sample id (if you care)

- Make sure your objects are compatible
  - Same genes, in the same order
  - Same `colData` column names, in the same order
  - Same assays (and reduced dimensions, if you're keeping them)

Some new functions we'll be seeing

- `purrr::map()` for faster and cleaner iteration ("looping")
  - (and friends! https://purrr.tidyverse.org/reference/map.html)
- `glue::glue()` for combining strings
  - Bonus! You also get to say `glue::glue()` all the time

Iterating with `purrr`

```r
histologies <- list(
  "low-grade gliomas"  = c("SEGA", "PA", "GNG", "PXA"),
  "high-grade gliomas" = c("DMG", "DIPG"),
  "embryonal tumors"   = c("MB", "ATRT", "ETMR")
 )
```

```r
for (diagnosis in histologies){
  print(diagnosis)
}
# [1] "SEGA" "PA"   "GNG"  "PXA"
# [1] "DMG"  "DIPG"
# [1] "MB"   "ATRT" "ETMR"
```

Iterating with `purrr`

```r
histologies <- list(
  "low-grade gliomas"  = c("SEGA", "PA", "GNG", "PXA"),
  "high-grade gliomas" = c("DMG", "DIPG"),
  "embryonal tumors"   = c("MB", "ATRT", "ETMR")
 )
```

```r
length(histologies)
[1] 3
```

```r
diag_lengths <- c()
for (diag in histologies){
  diag_lengths <- c(diag_lengths, length(diag))
}
diag_lengths
[1] 4  2  3
```

Iterating with `purrr` returns a list (with names, if you've got 'em!)

```
diag_lengths <- c()
for (diag in histologies){
  diag_lengths <- c(diag_lengths, length(diag))
}
diag_lengths
[1] 4  2  3
```

```
purrr::map(histologies, length)
$`low-grade gliomas`
[1] 4

$`high-grade gliomas`
[1] 2

$`embryonal tumors`
[1] 3
```

Syntax for more complicated operations

When you want to do more than just call a single function, you can either:

- Define your own function separately and use that
- Use this handy function shortcut **`\(x) { ... code ...}`**
  - `x` is your "looping" variable. Try to give it an informative name (usually not `x`)

```
histologies |>
  purrr::map(
    # function shortcut
    \(diagnoses) {
     length(diagnoses) + 5
    }
  )
$`low-grade gliomas`
[1] 9

$`high-grade gliomas`
[1] 7

$`embryonal tumors`
[1] 8
```

Introducing `glue::glue()`

```
org <- "Data Lab"

# paste combines multiple strings
paste("Welcome to the", org, "workshop!")
[1] "Welcome to the Data Lab workshop!"


# glue uses only 1 set of quotes!
glue::glue("Welcome to the {org} workshop!")
[1] "Welcome to the Data Lab workshop!"
```

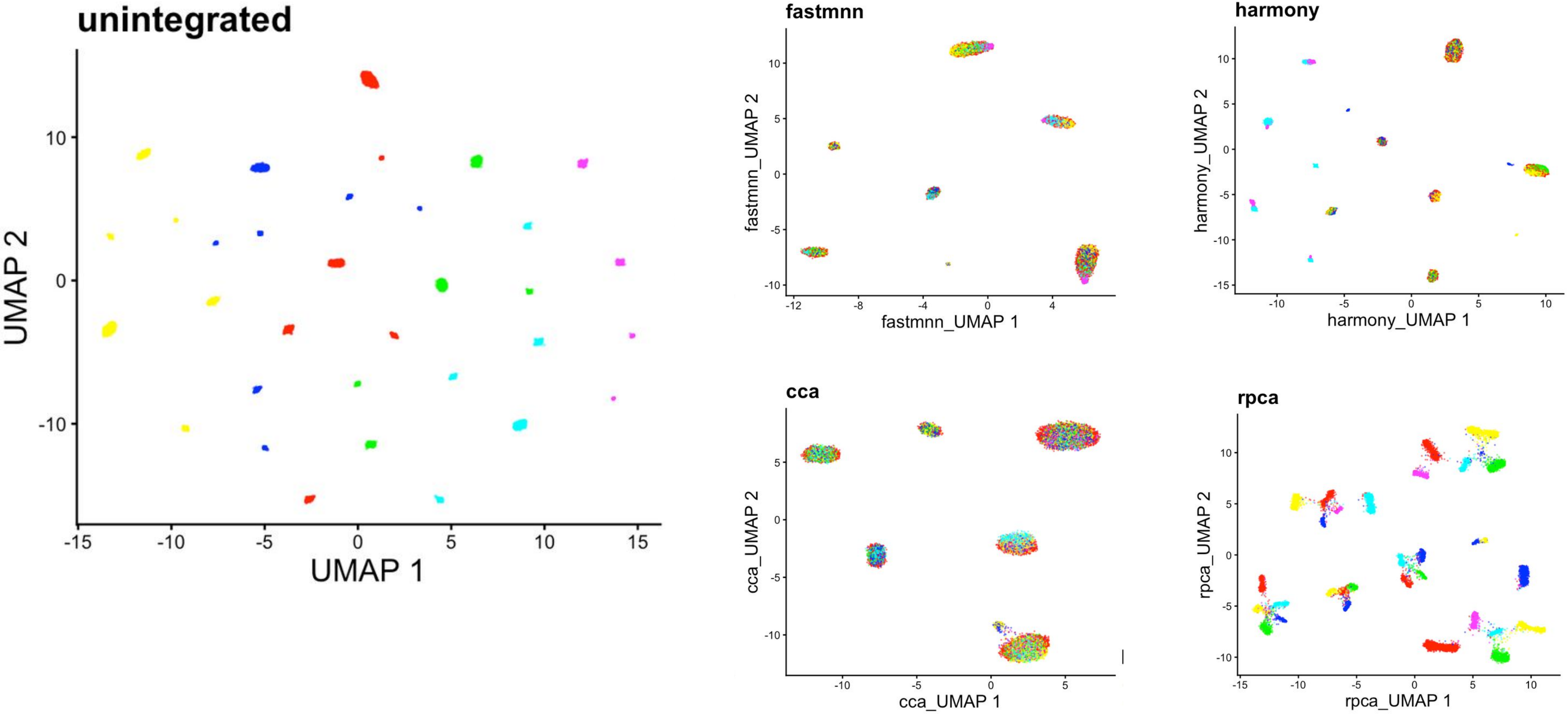
The following slides show some results we obtained benchmarking integration algorithms.

We performed some benchmarking on simulated data from Luecken *et al.*

- We evaluated several methods, four of which we'll show here:
  - **FastMNN**
  - **Harmony**
  - **Seurat using CCA** (canonical correlation analysis)
  - **Seurat using RPCA** (reciprocal PCA)
  - (We'll note that we also looked at scVI, which seemed to work well but is slow on CPU and it's in Python which is beyond the scope of our workshop!)
- We chose these methods based on performance in Luecken *et al.* and their usability
- See https://github.com/AlexsLemonade/sc-data-integration for our benchmarking code

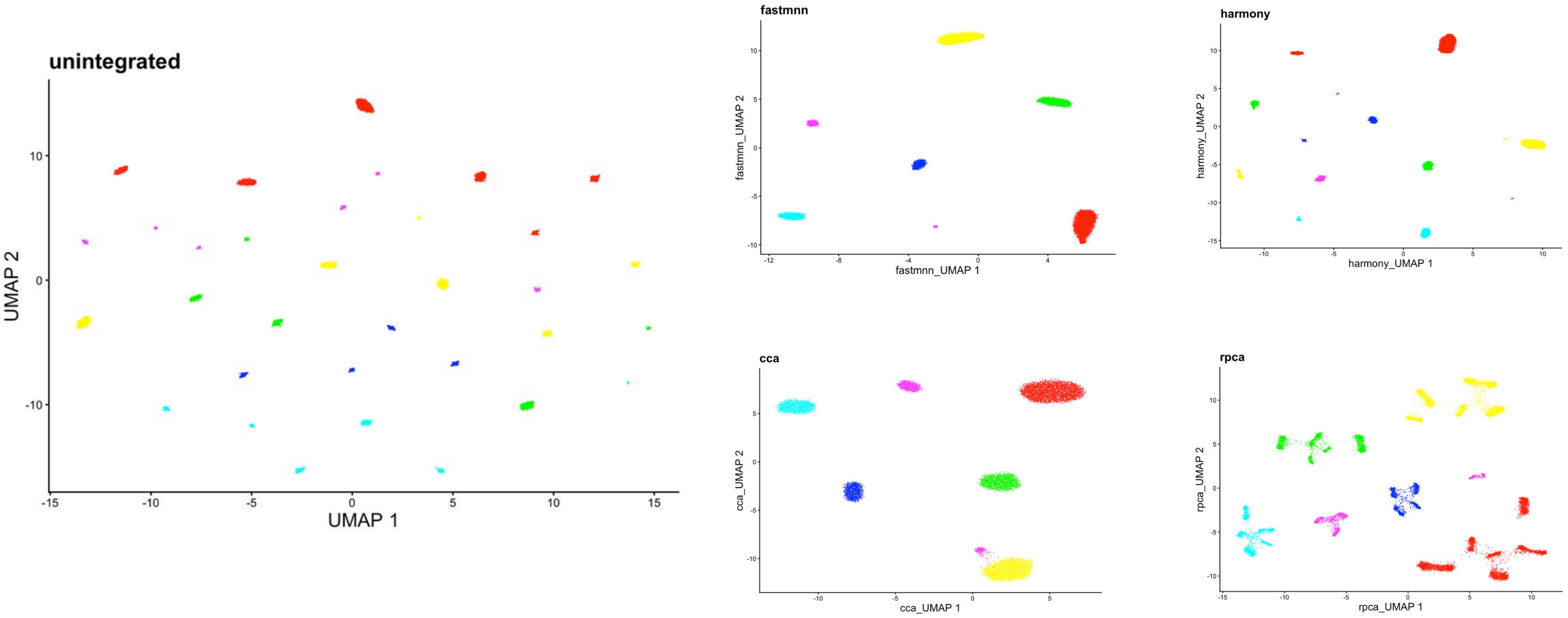
Scenario 1: All cell types are present in all batches

UMAPs colored by Batch



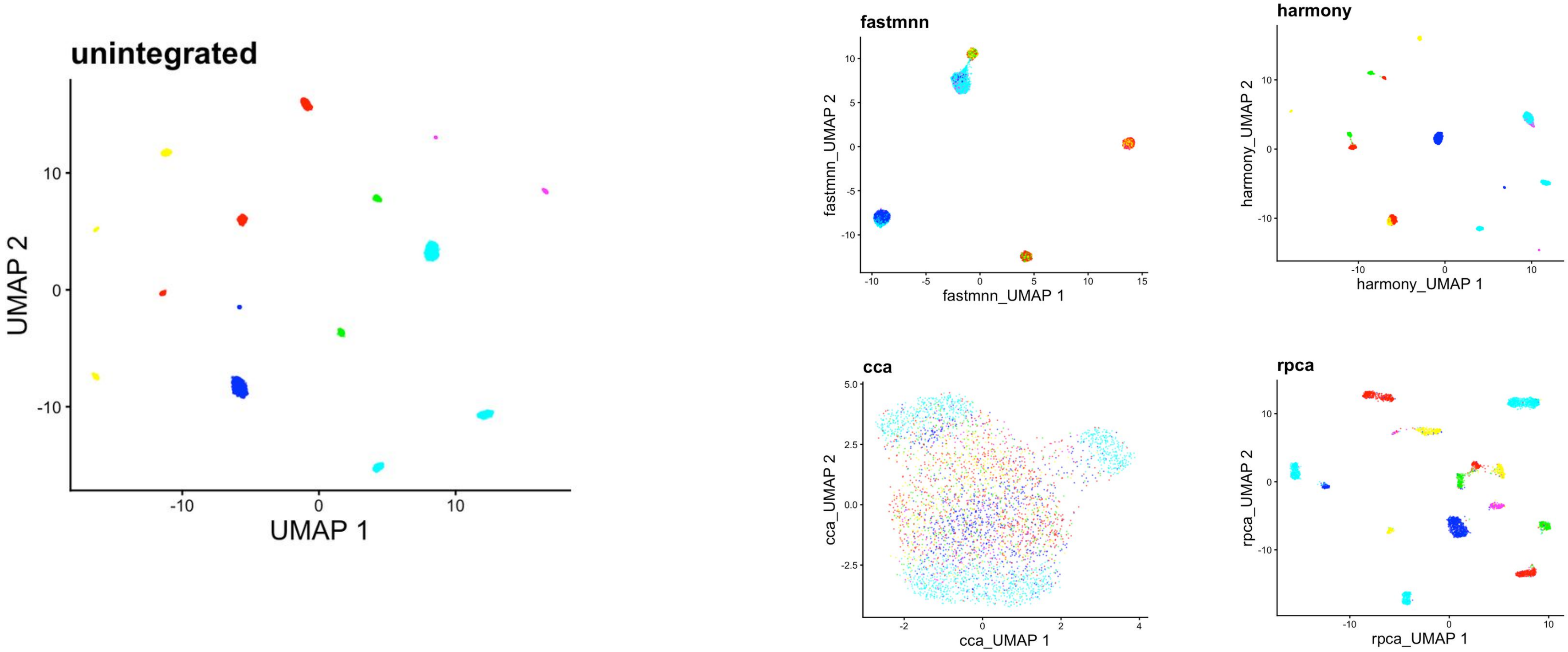
Scenario 1: All cell types are present in all batches

UMAPs colored by Cell Type



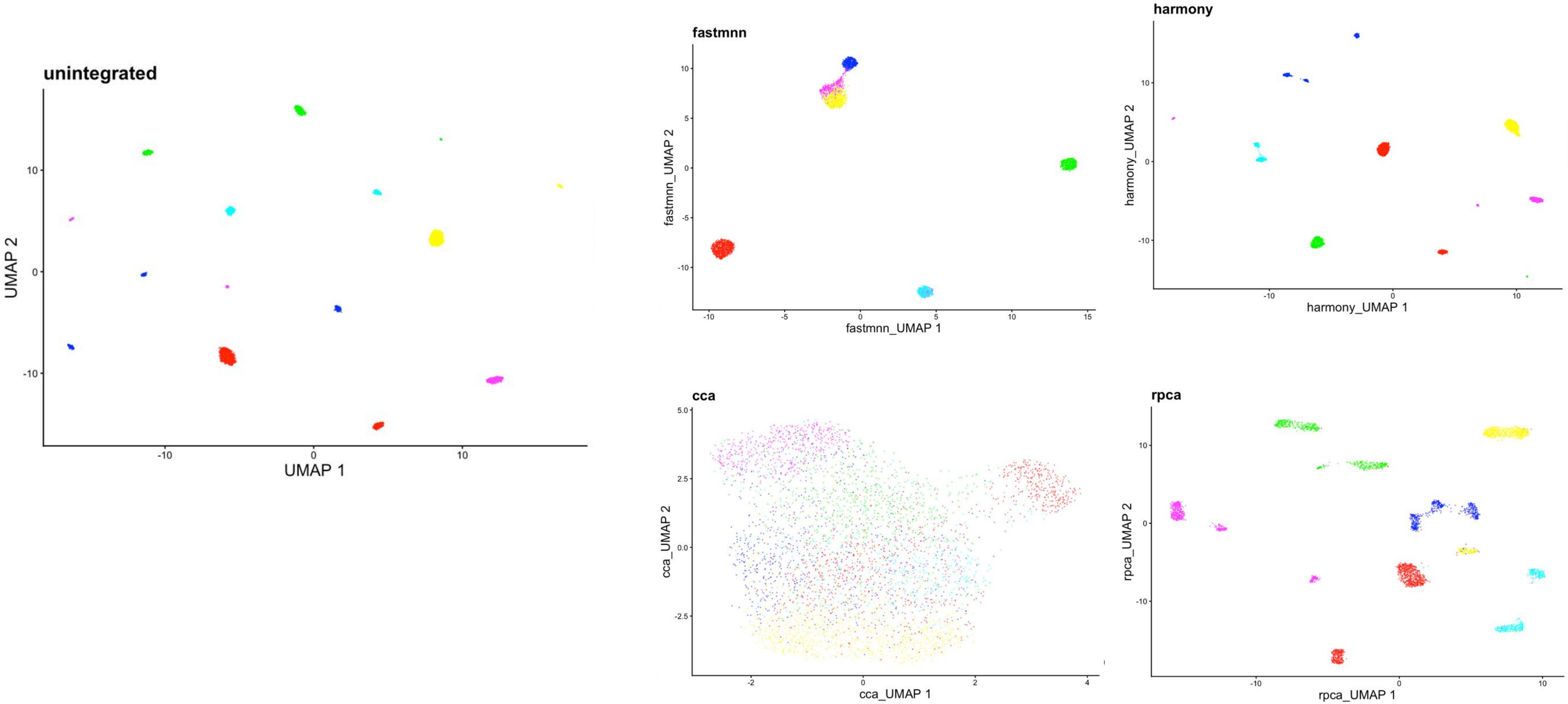
Scenario 2: Cell types are not present in all batches, and not all batches have cells in common

UMAPs colored by Batch



Scenario 2: Cell types are not present in all batches, and not all batches have cells in common

UMAPs colored by Cell Type


unintegrated
UMAP 2
UMAP 1
fastmnn
fastmnn_UMAP 2
fastmnn_UMAP 1
harmony
harmony_UMAP 2
harmony_UMAP 1
cca
cca_UMAP 2
cca_UMAP 1
rpca
rpca_UMAP 2
rpca_UMAP 1