Complex and tidy data structures in R

Workflows with transitions from Bioconductor to the tidyverse

Brooke Anderson

Overview

1. Data structures in R
2. The tidyverse approach
3. The Bioconductor approach
4. Transitioning across approaches in a workflow

Slides are available at: bit.ly/complex_tidy

Data structures

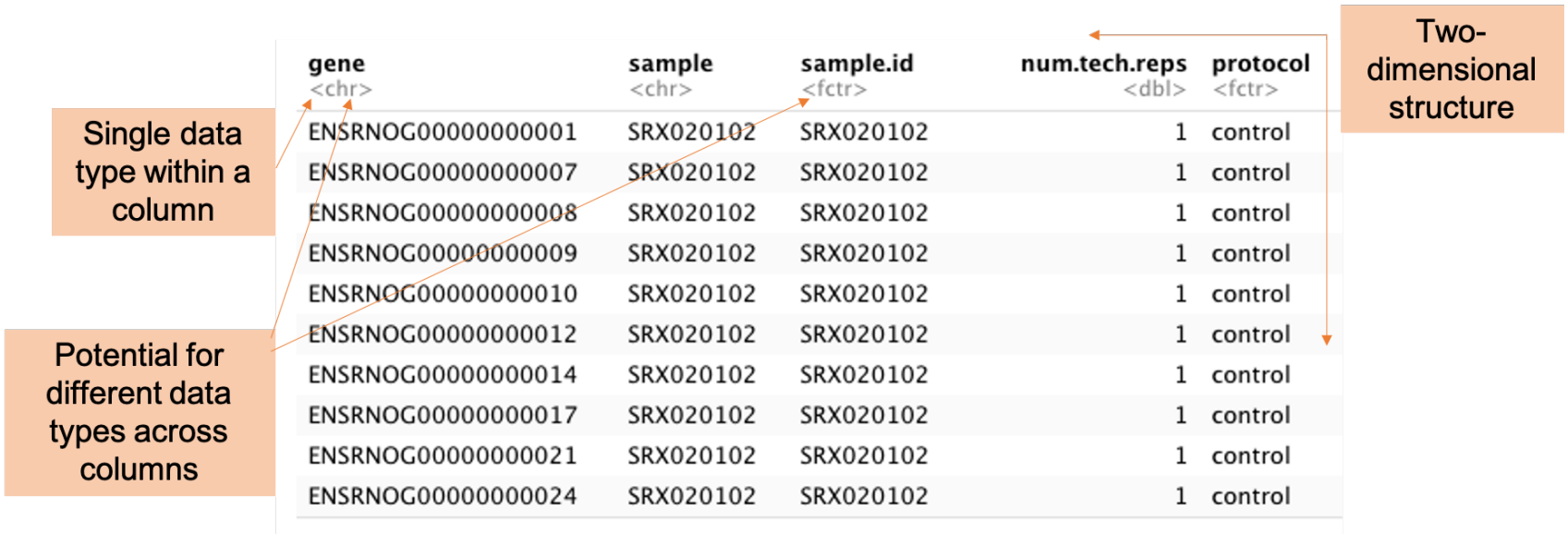
Where do you keep your data?

To work with data in R, you typically read it into memory.

Data structures help define the format in which you store your data.

Types of data structures in R: Simple

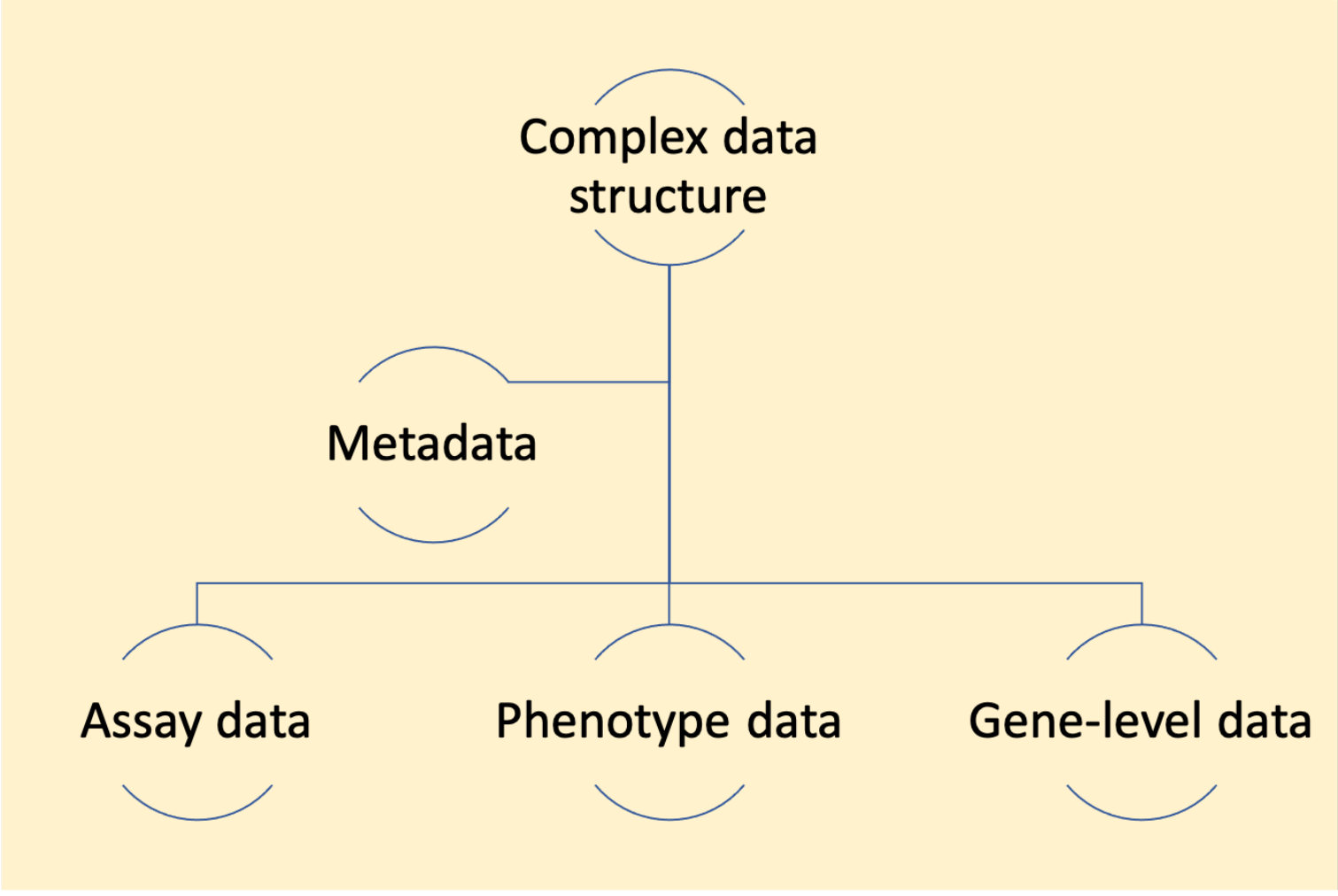
One simple data structure in R is the **dataframe**. This is the structure used extensively in the **tidyverse approach**.



gene <chr>	sample <chr>	sample.id <fctr>	num.tech.reps <dbl>	protocol <fctr>
ENSRNOG00000000001	SRX020102	SRX020102	1	control
ENSRNOG00000000007	SRX020102	SRX020102	1	control
ENSRNOG00000000008	SRX020102	SRX020102	1	control
ENSRNOG00000000009	SRX020102	SRX020102	1	control
ENSRNOG00000000010	SRX020102	SRX020102	1	control
ENSRNOG00000000012	SRX020102	SRX020102	1	control
ENSRNOG00000000014	SRX020102	SRX020102	1	control
ENSRNOG00000000017	SRX020102	SRX020102	1	control
ENSRNOG00000000021	SRX020102	SRX020102	1	control
ENSRNOG00000000024	SRX020102	SRX020102	1	control

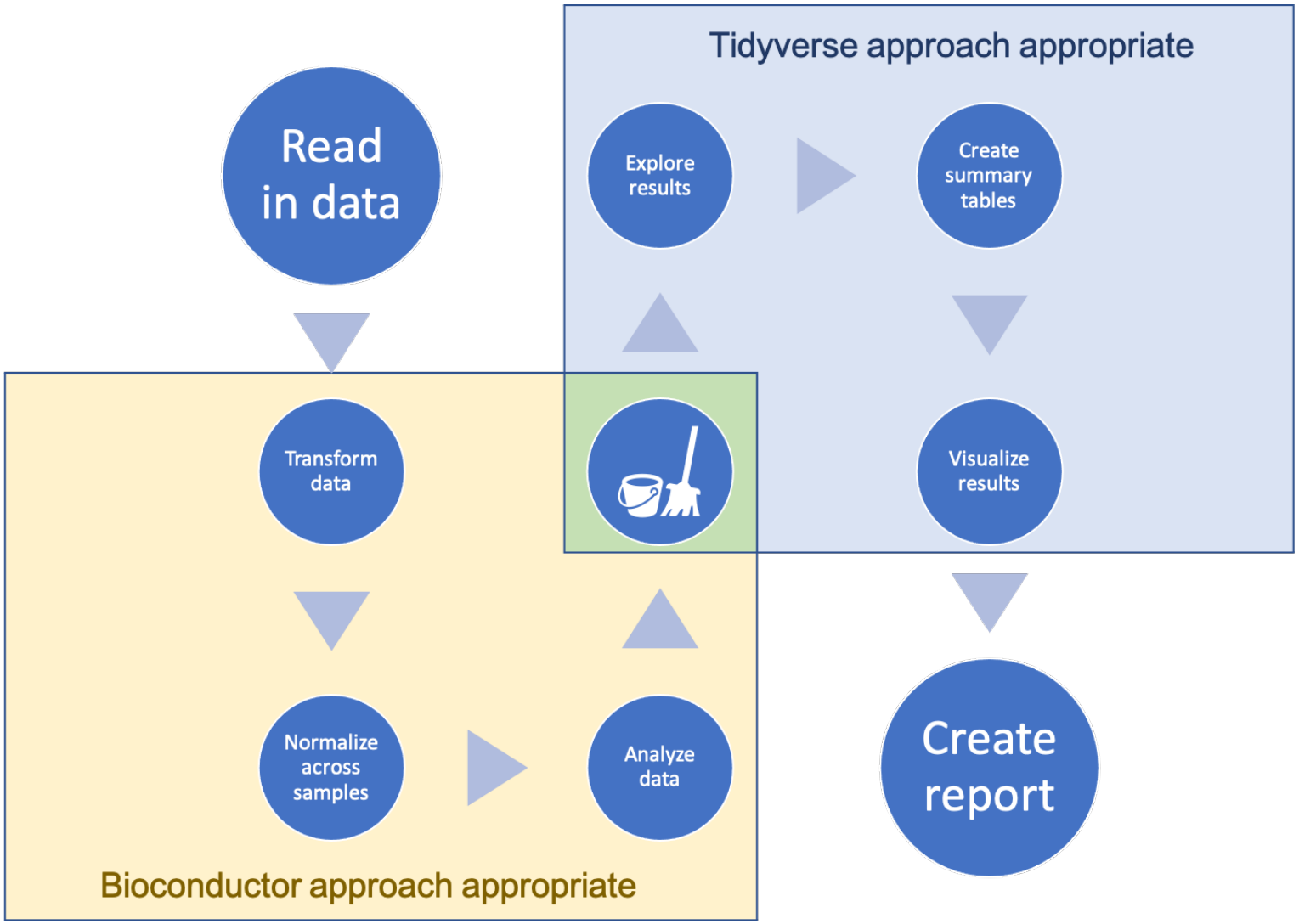
Types of data structures in R: Complex

More complex data structures in R are all, at heart based on lists. This format allows each object to collect different pieces of data, with different types and dimensions.



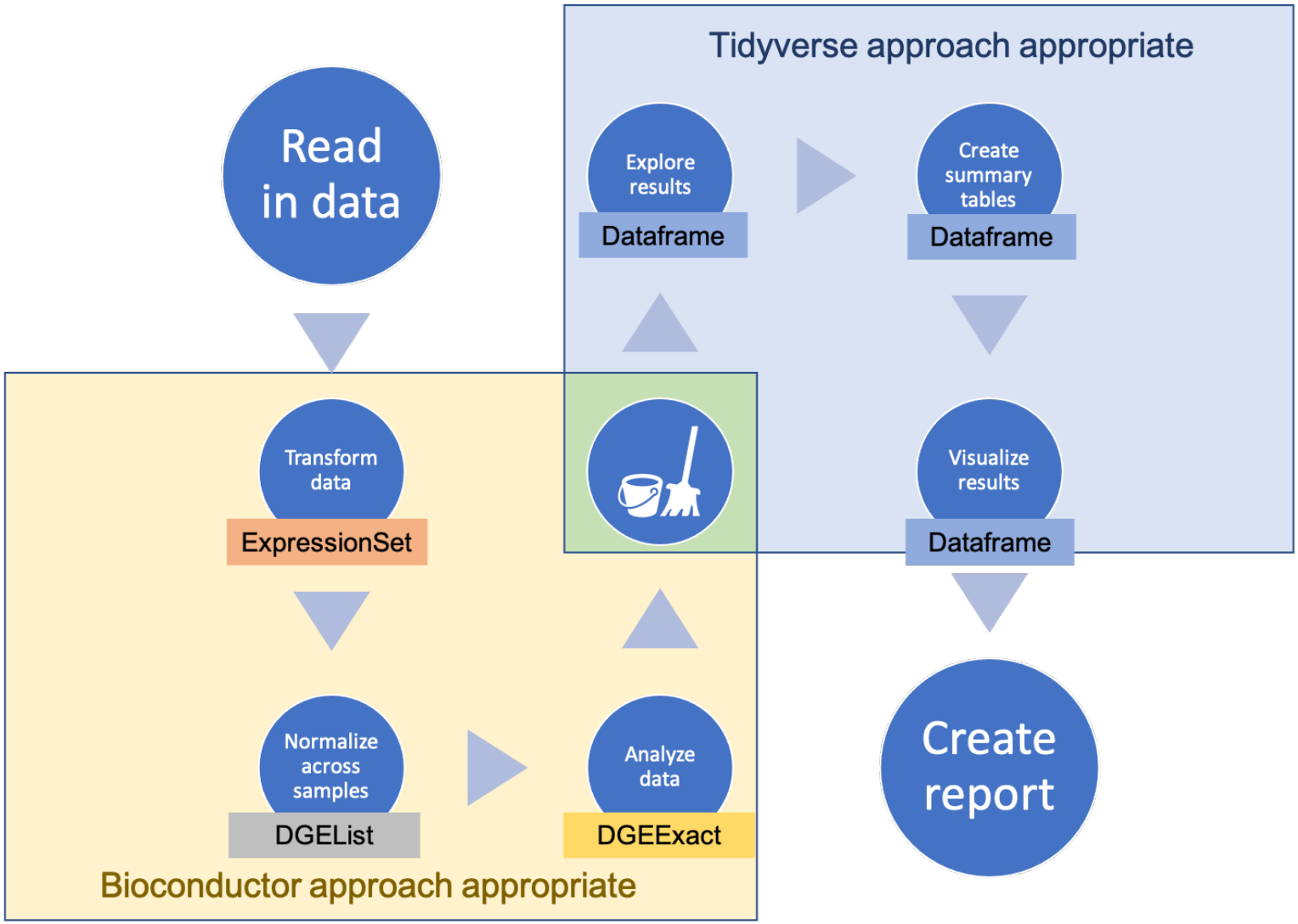
Data structures across a workflow

Work with research data will typically require a series of steps for pre-processing, analysis, exploration, and visualization. Collectively, these form a **workflow** for the data.



Data structures across a workflow

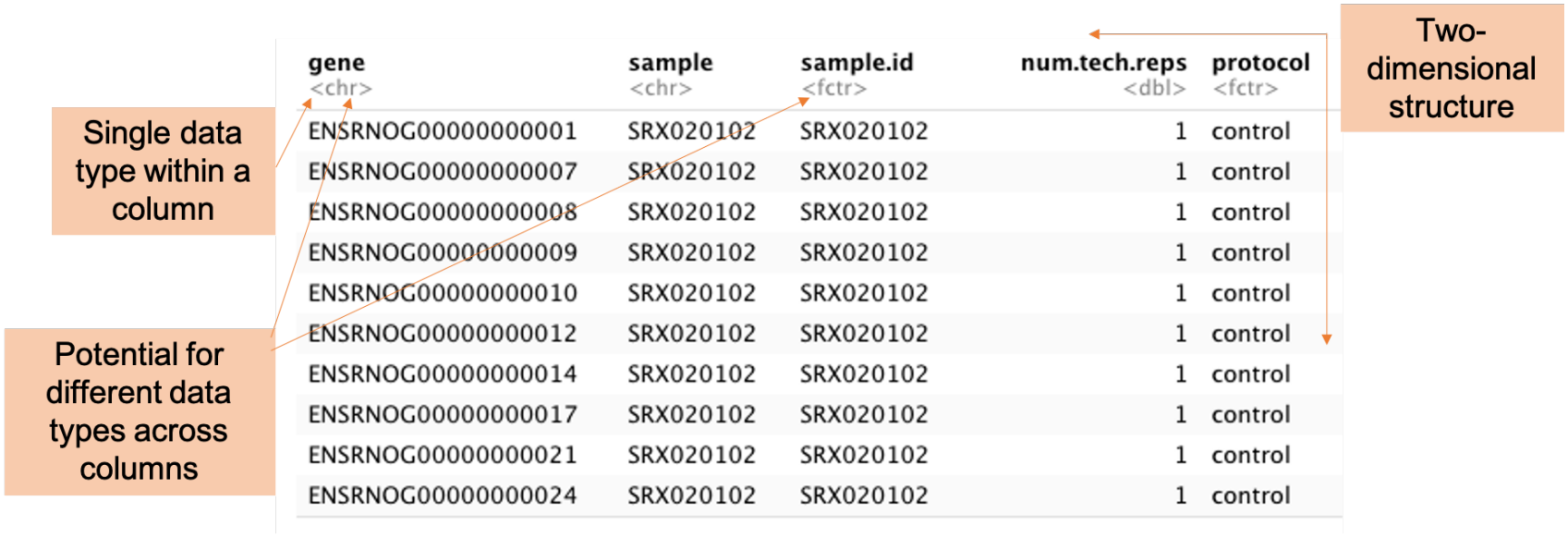
You can move your data among different structures across a workflow, including from more complex data to simpler data structures.



Tidyverse approach

Tidyverse data structure

The key data structure in the tidyverse approach is the **dataframe**:



gene <chr>	sample <chr>	sample.id <fctr>	num.tech.reps <dbl>	protocol <fctr>
ENSRNOG00000000001	SRX020102	SRX020102	1	control
ENSRNOG00000000007	SRX020102	SRX020102	1	control
ENSRNOG00000000008	SRX020102	SRX020102	1	control
ENSRNOG00000000009	SRX020102	SRX020102	1	control
ENSRNOG00000000010	SRX020102	SRX020102	1	control
ENSRNOG00000000012	SRX020102	SRX020102	1	control
ENSRNOG00000000014	SRX020102	SRX020102	1	control
ENSRNOG00000000017	SRX020102	SRX020102	1	control
ENSRNOG00000000021	SRX020102	SRX020102	1	control
ENSRNOG00000000024	SRX020102	SRX020102	1	control

Tidyverse approach

The tidyverse approach is built on the use of a common structure for storing data—almost all functions take data in this structure and almost all return data in this structure.

In other words, it is built on the idea of a **common interface** across all its functions.

Advantage of the tidyverse approach

This is similar to Legos: there are set dimensions for all blocks, so they can easily snap together in any order:

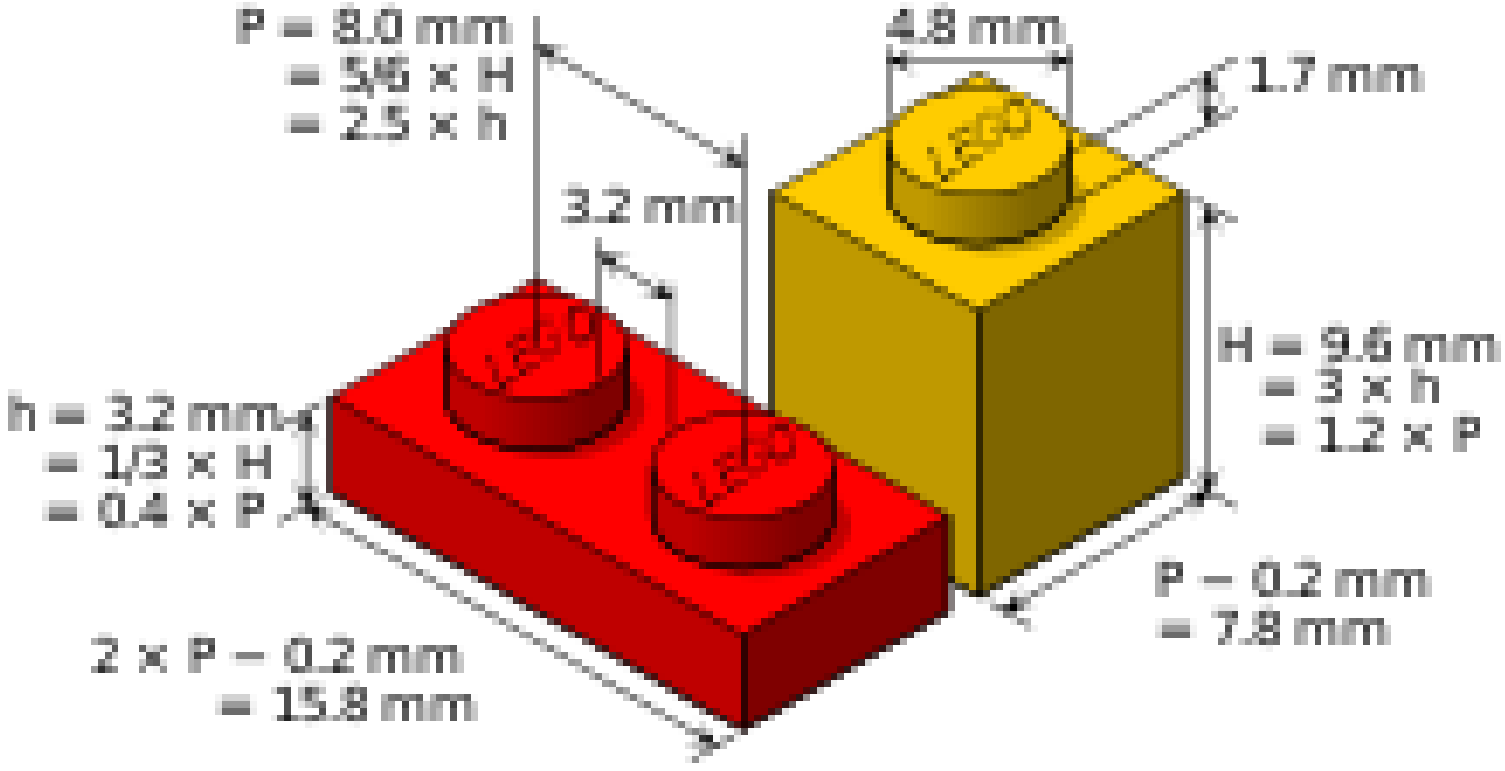


Image source:
https://en.wikipedia.org/wiki/Lego#/media/File:Lego_dimensions.svg

Advantage of the tidyverse approach

The common interface idea turns out to be very powerful. It allows you to not have to rely on large functions that do a lot all at once.

Instead, this idea allows for lots of **small functions** that each do **one small thing**, but that can be chained together in lots of different configurations to do very flexible and powerful things.

Advantage of the tidyverse approach

It is hard to overemphasize how powerful this approach is. Simple tools that can be connected together in different ways can be used to create very complex things:

Image source: *Architectural Digest*

Advantage of the tidyverse approach

This allows you to learn a single set of tools—most of which can be learned in a few months.

These work across all your data, as long as it's in a **tidy dataframe structure** while you're working on it.

By contrast, if you use a variety of data structures, you often have to learn different tools (functions) for each data structure, rather than being able to use a single set of tools for all your data.

More resources on the tidyverse approach

There are lots of excellent resources on learning the tidyverse approach in R, many of which are freely available online.

You may want to check:

- *R for Data Science*
- Course book for ERHS 535
- rstudio::conf talks

Bioconductor approach

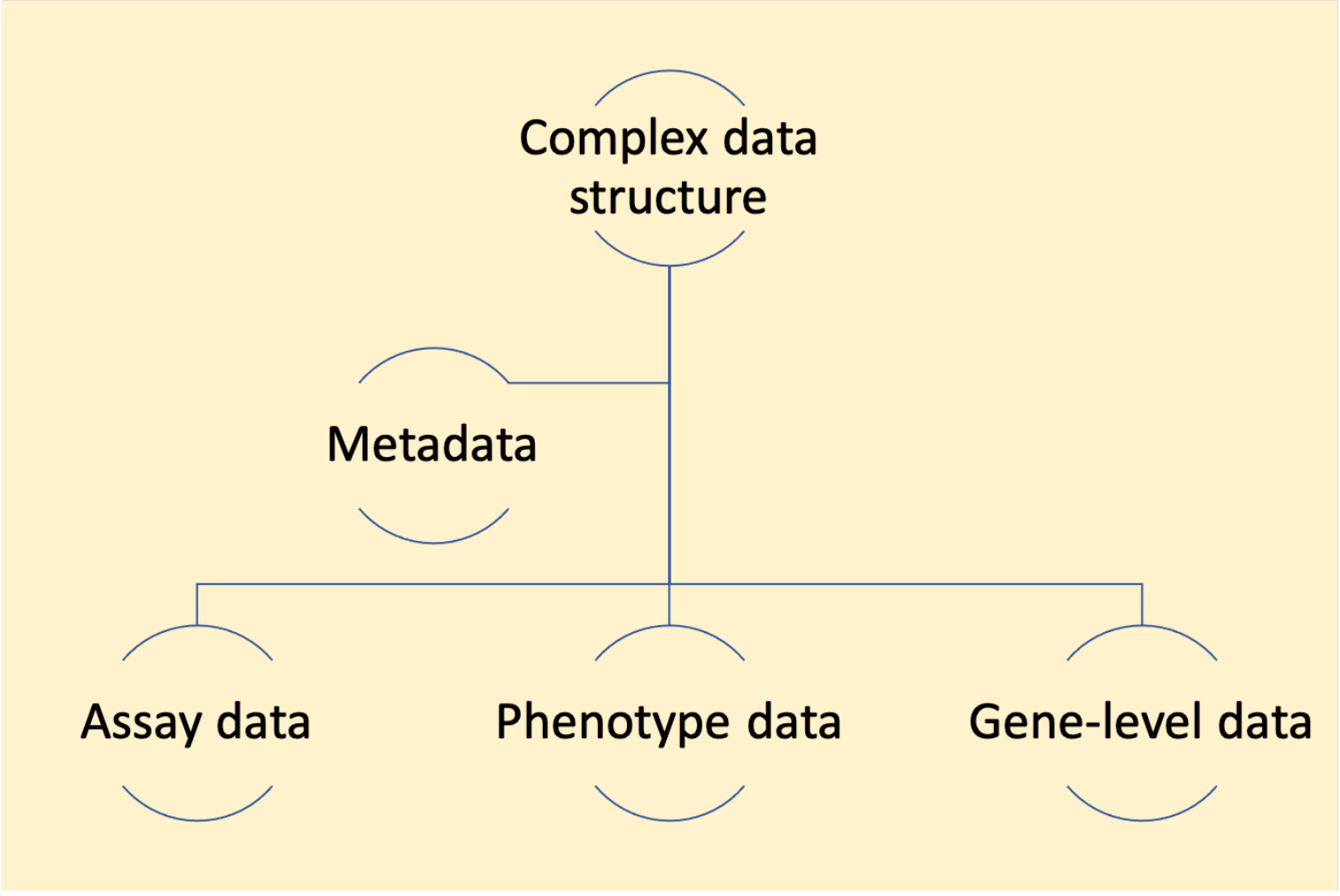
Bioconductor data structures

Bioconductor data structures tend to be more complex.

They are built on **S4 object classes**, which is one of R's **object-oriented programming** systems.

Bioconductor data structures

They often will include several elements. The elements might have different data types or different dimensions, but the data structure stores these disparate parts together in one object.



Types of data structures in R: Complex

You can start to see these levels in a complex data object by using the `str` function.

For example, here's a peak at the structure of data stored in an `ExpressionSet` data structure from Bioconductor:

```
library(Biobase); library(biobroom); data(hammer)
str(hammer, max.level = 3)
```

```
## Formal class 'ExpressionSet' [package "Biobase"] with 7 slots
##   ..@ experimentData   :Formal class 'MIAME' [package "Biobase"] wit
##   ..@ assayData        :<environment: 0x7f89fdee7768>
##   ..@ phenoData        :Formal class 'AnnotatedDataFrame' [package "
##   ..@ featureData      :Formal class 'AnnotatedDataFrame' [package "
##   ..@ annotation       : chr(0)
##   ..@ protocolData     :Formal class 'AnnotatedDataFrame' [package "
##   ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"]
```

Types of data structures in R: Complex

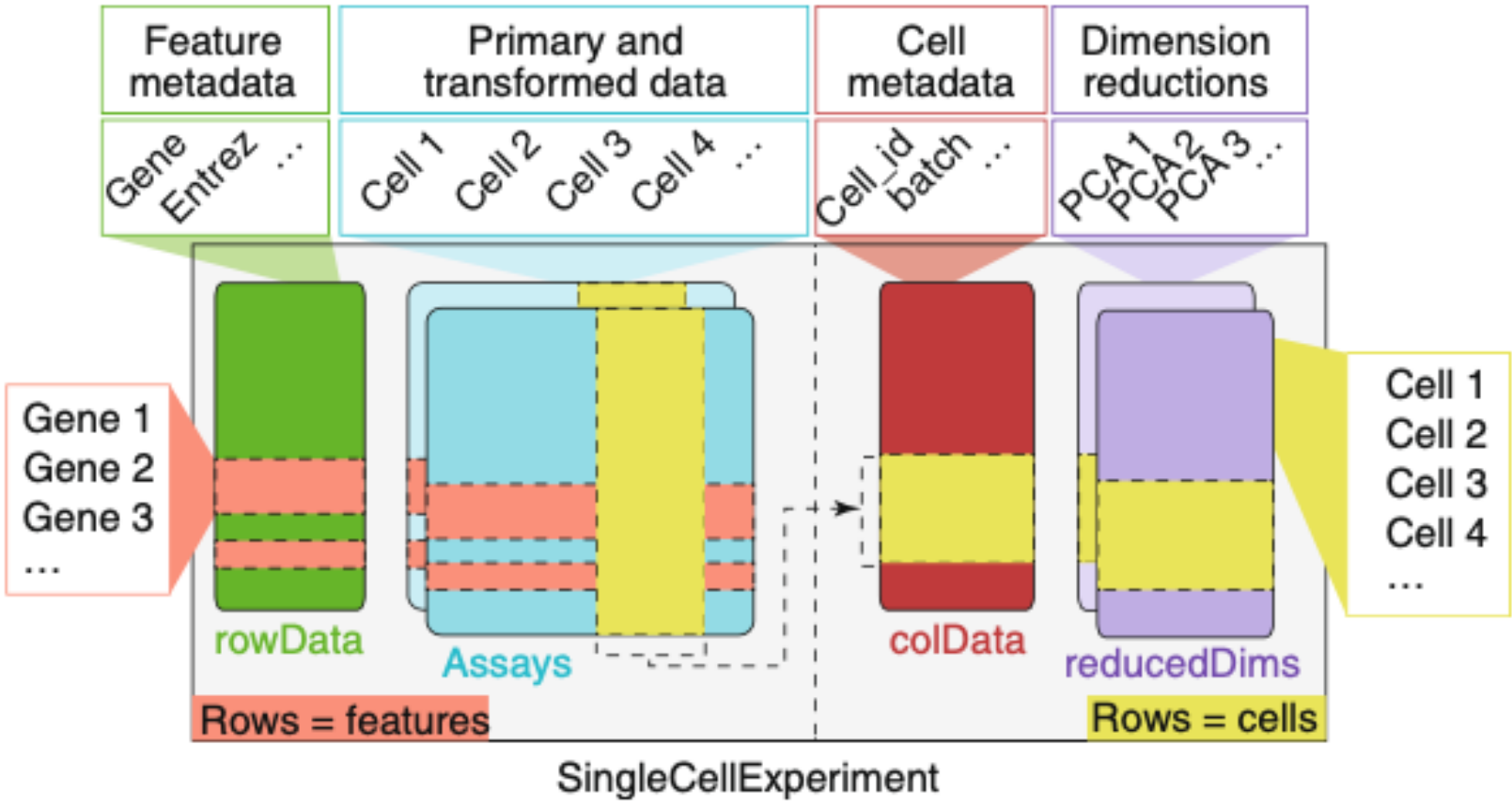


Image source: Amezquita et al., 2020

Data structures in Bioconductor

Examples of data structures in Bioconductor include:

- `SummarizedExperiment`
- `GRanges`
- `ExpressionSet`
- `MSnExp`
- `SingleCellExperiment`
- `MultiAssayExperiment`
- `DGEList`
- `DGEExact`

Once you have its package loaded, you can access help documentation on a data structure using, e.g., `` ?`DGEList-class` ``.

Why use more complex data structures?

1. To keep together lots of differently structured things

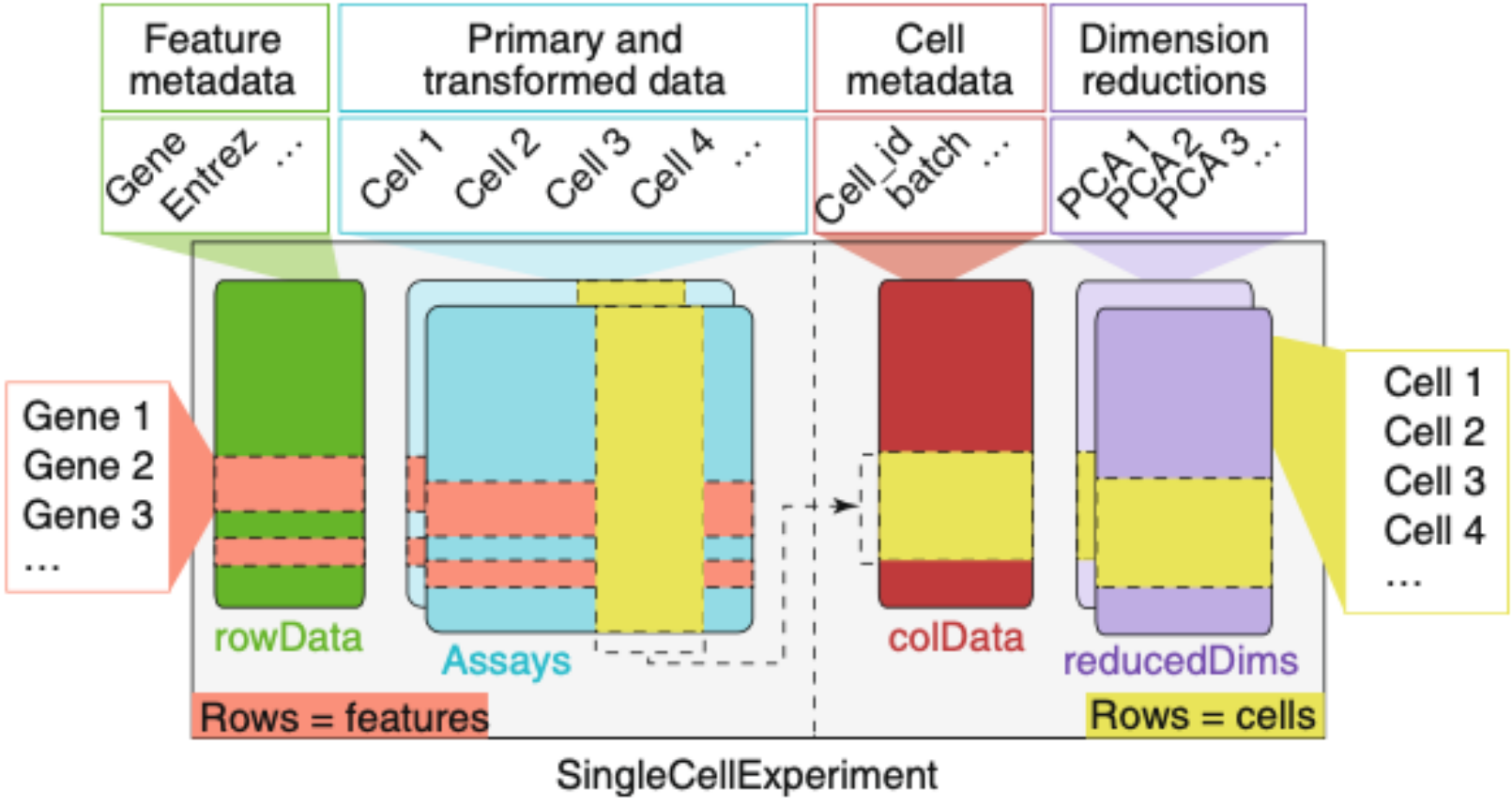


Image source: Amezquita et al., 2020

Why use more complex data structures?

2. To work with large data

Complex data structures can help ensure that data are efficiently stored. In some cases, they can also allow for data storage formats where part of the data stay on disk, rather than being read into memory.

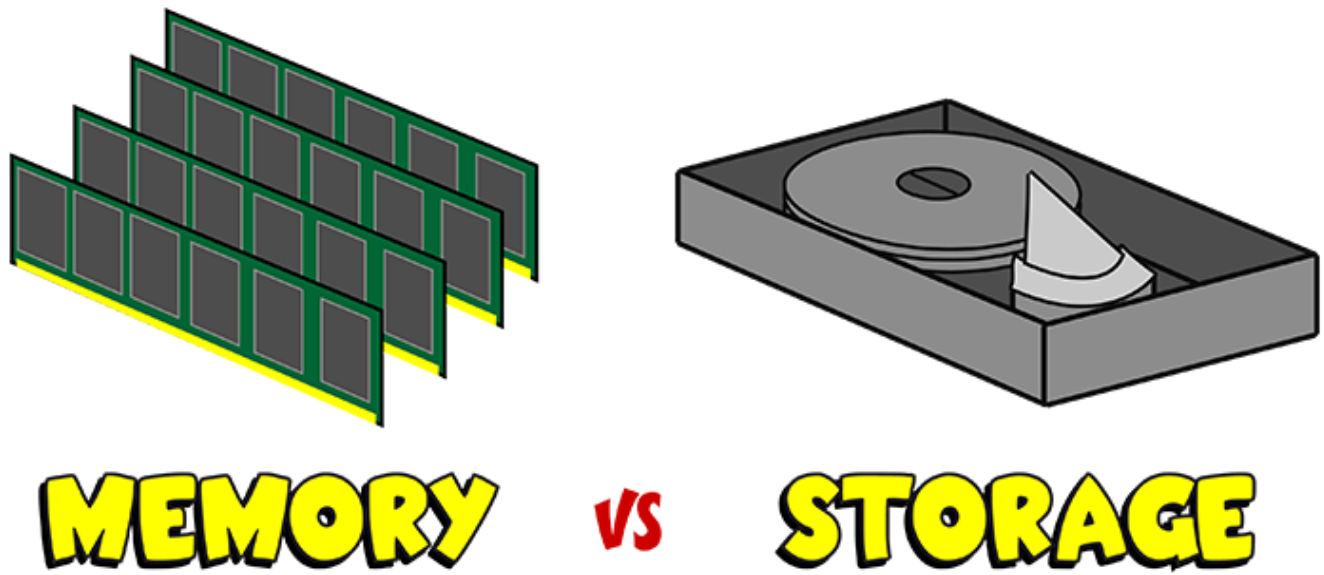


Image source: callnerds.com

Why use more complex data structures?

One example is the `OnDiskMSnExp` data structure from the `MSnbase` package:

> *"The distinction between MSnExp and OnDiskMSnExp is often not explicitly stated as* ***it should not matter, from a user's perspective, which data structure they are working with****, as both behave in equivalent ways. Often, they are referred to as in-memory and on-disk MSnExp implementations." (Gatto et al. 2013)*

Why use more complex data structures?

3. To validate data when an object's created

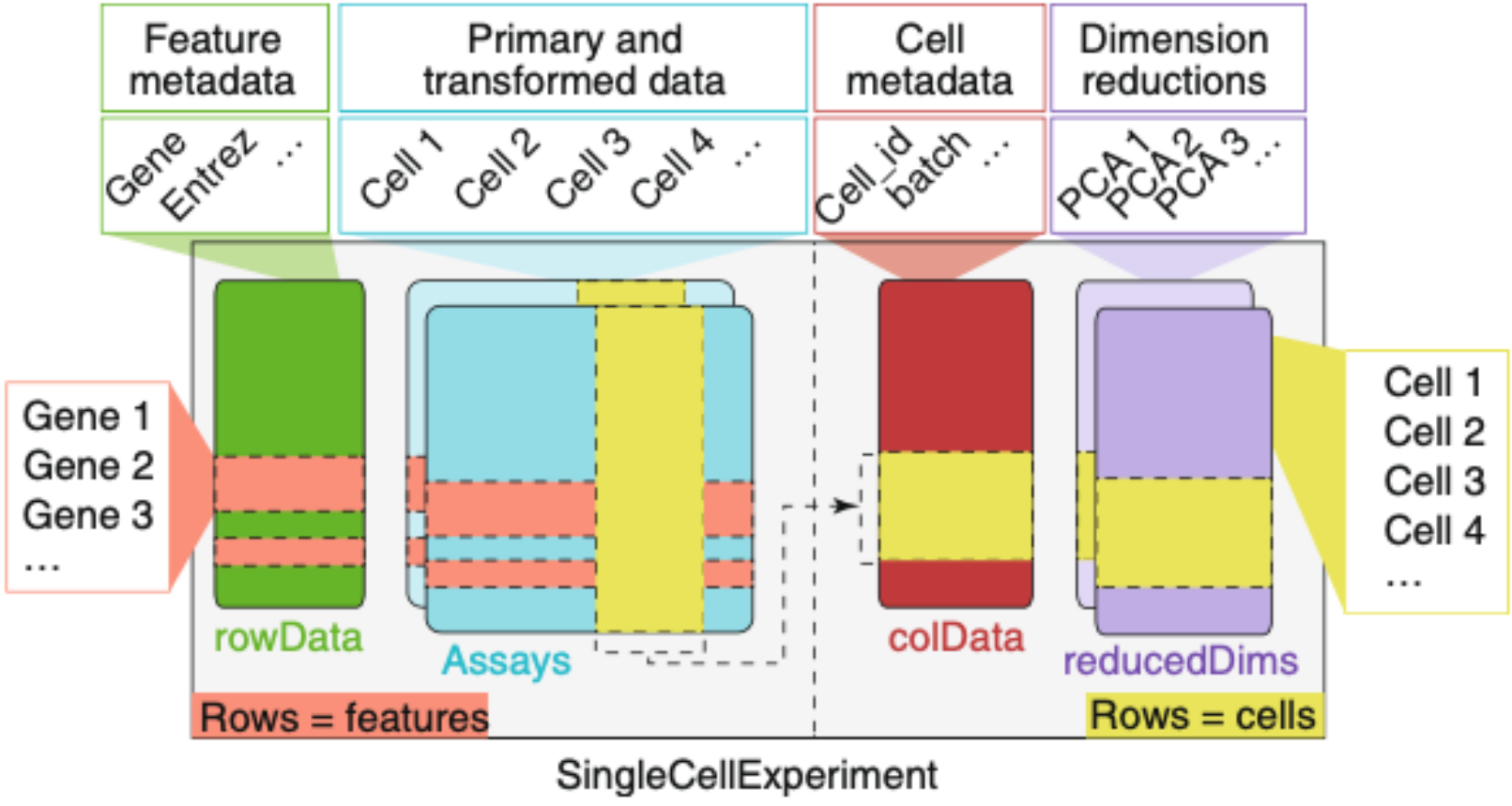


Image source: Amezquita et al., 2020

Why use more complex data structures?

4. To facilitate software development across large and diverse groups of contributors

   "S4 is a rigorous system that forces you to thing carefully about program design. ***It's particularly well-suited for building large systems that evolve over time and will receive contributions from many programmers.*** *This is why it's used by the Bioconductor project, so another reason to learn S4 is to equip you to contribute to that project." (Wickham, Advanced R)*

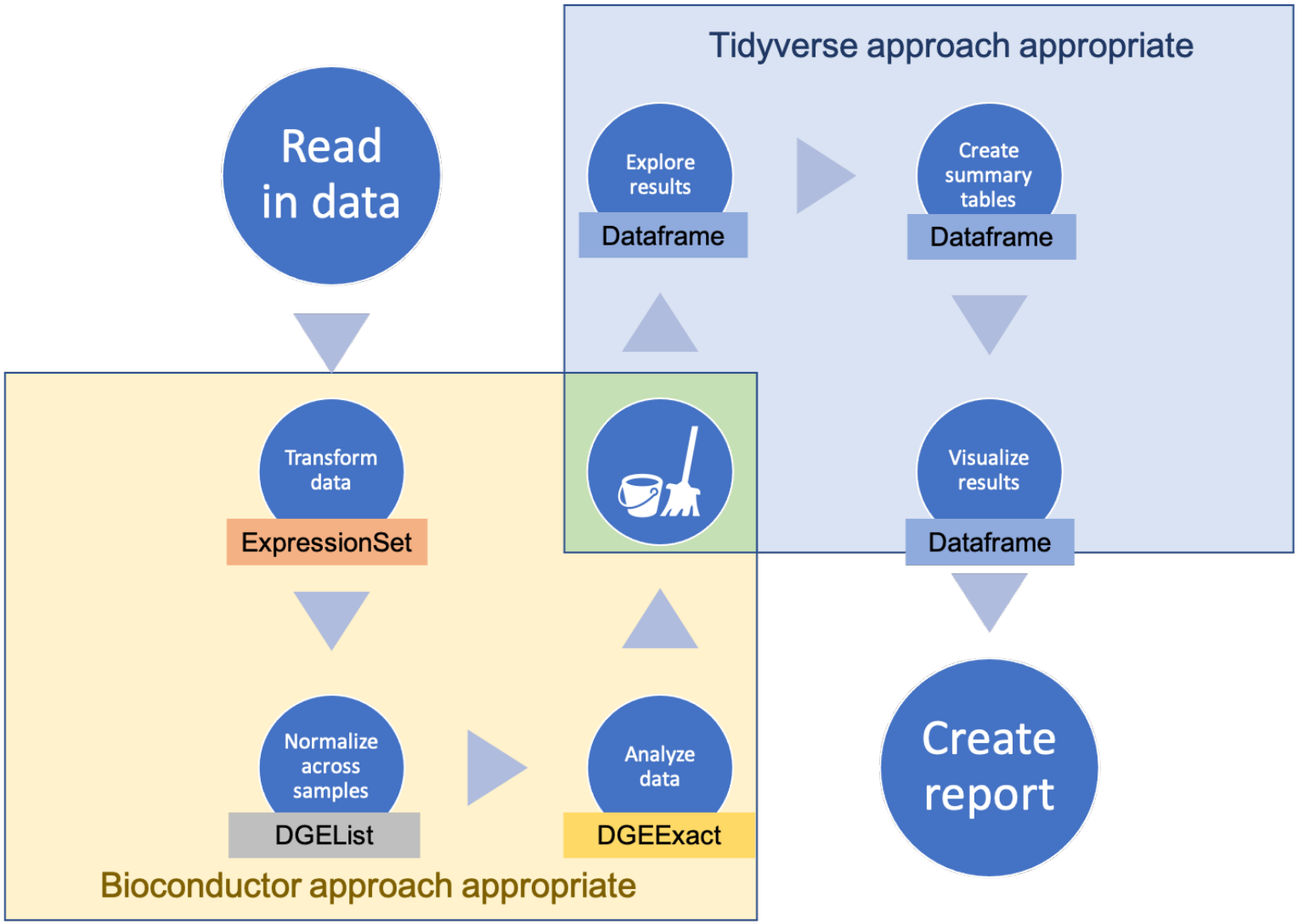
More resources on the Bioconductor approach

- Bioconductor site (especially the "Learn" section)
- BioC Conference

Connecting the two approaches

Data structures across a workflow

You can move your data among different structures across a workflow, including from more complex data to simpler data structures.



Example data

The `hammer` dataset is available through the `biobroom` package. It provides data from an RNA-Seq experiment for a study on nervous system transcriptomics (Hammer et al., 2010).

The data are stored in an `ExpressionSet` data structure, a common class used in Bioconductor work. Different elements of the data structure store data from the assay as well as phenotype data.

Example data

```
library(Biobase); library(biobroom)
data(hammer)
print(hammer)
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 29516 features, 8 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: SRX020102 SRX020103 ... SRX020098-101 (8 total)
##   varLabels: sample.id num.tech.reps ... Time (5 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: ENSRNOG00000000001 ENSRNOG00000000007 ...
##     ENSRNOG00000045521 (29516 total)
##   fvarLabels: gene
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

Connecting the two approaches

There are several ways you can connect the two approaches:

- Generic functions from the **`biobroom` package**
- **Accessor functions** written for specific Bioconductor data structures
- **Elemental tools** for extracting parts of data from R objects

`biobroom` package

If you're lucky, you can make the connection very easily using the **`biobroom` package**.

This package allows you to extract elements from several types of Bioconductor data structures.

It has generic functions that pull out elements and format them as tidy dataframes.

biobroom package

Bioconductor data structures that currently work with biobroom include:

- ExpressionSet
- GRanges
- GRangesList
- MSnSet
- RangedSummarizedExperiment
- qvalue
- DESeqDataSet
- DESeqResults
- MArrayLM

biobroom example

You can use the `tidy` function to extract a tidy dataframe with assay data from an `ExpressionSet` object in R:

```
tidy(hammer, addPheno = TRUE)
```

```
## # A tibble: 236,128 x 8
##    gene                sample sample.id num.tech.reps protocol strain
##    <chr>               <chr>  <fct>              <dbl> <fct>    <fct>
##  1 ENSRNOG00000000001 SRX02~ SRX020102              1 control  Sprag~
##  2 ENSRNOG00000000007 SRX02~ SRX020102              1 control  Sprag~
##  3 ENSRNOG00000000008 SRX02~ SRX020102              1 control  Sprag~
##  4 ENSRNOG00000000009 SRX02~ SRX020102              1 control  Sprag~
##  5 ENSRNOG00000000010 SRX02~ SRX020102              1 control  Sprag~
##  6 ENSRNOG00000000012 SRX02~ SRX020102              1 control  Sprag~
##  7 ENSRNOG00000000014 SRX02~ SRX020102              1 control  Sprag~
##  8 ENSRNOG00000000017 SRX02~ SRX020102              1 control  Sprag~
##  9 ENSRNOG00000000021 SRX02~ SRX020102              1 control  Sprag~
## 10 ENSRNOG00000000024 SRX02~ SRX020102              1 control  Sprag~
## # i 236,118 more rows
```

biobroom example

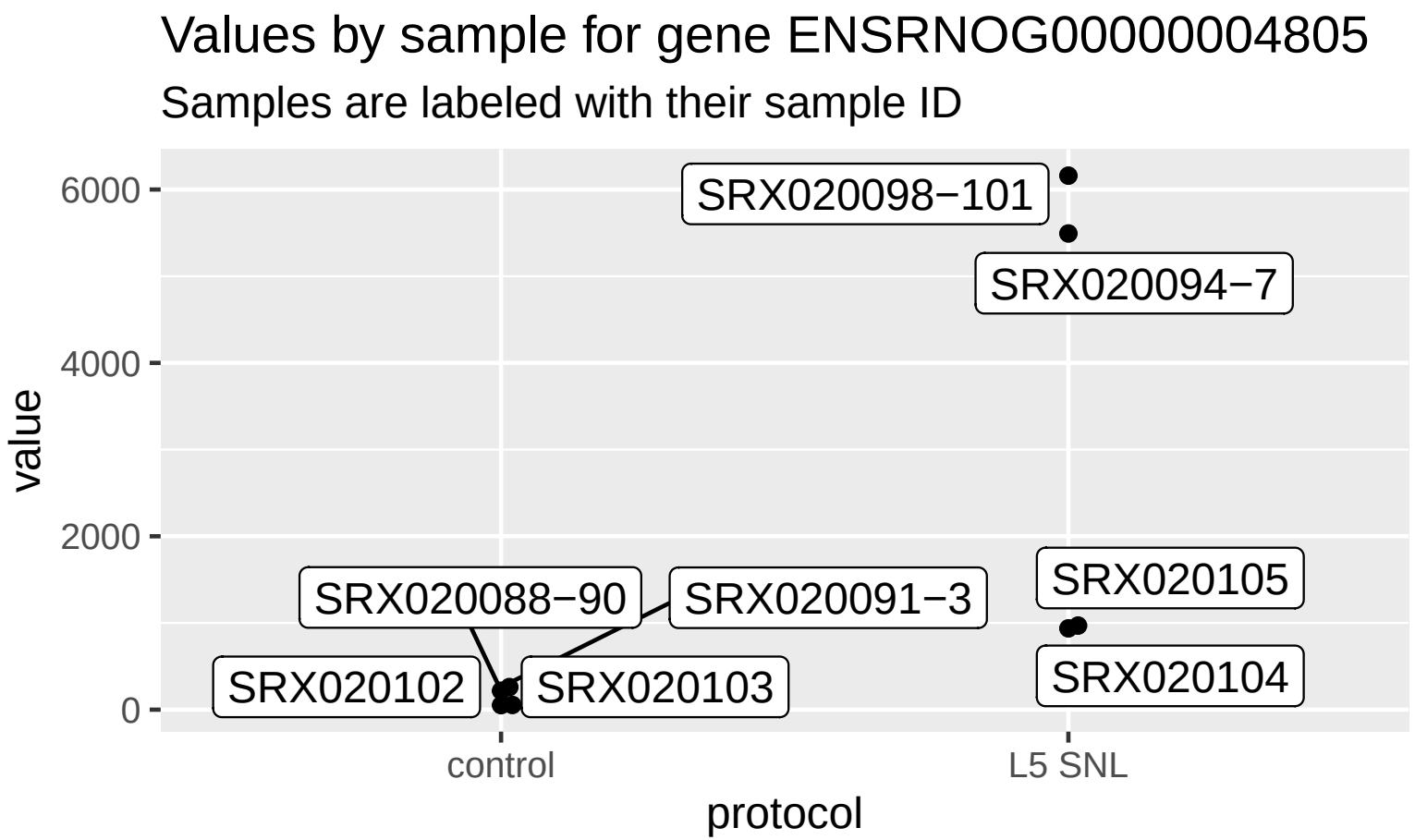
This allows you to use tidyverse functions to explore and visualize the data. For example, to explore the values for a single gene:

```
library(tidyverse); library(ggbeeswarm); library(ggrepel)
one_gene <- tidy(hammer, addPheno = TRUE) %>%
        filter(gene == "ENSRNOG00000004805")
one_gene
```

```
## # A tibble: 8 x 8
##   gene                sample  sample.id num.tech.reps protocol strain
##   <chr>               <chr>   <fct>              <dbl> <fct>    <fct>
## 1 ENSRNOG00000004805 SRX020~ SRX020102              1 control  Sprag~
## 2 ENSRNOG00000004805 SRX020~ SRX020103              2 control  Sprag~
## 3 ENSRNOG00000004805 SRX020~ SRX020104              1 L5 SNL   Sprag~
## 4 ENSRNOG00000004805 SRX020~ SRX020105              2 L5 SNL   Sprag~
## 5 ENSRNOG00000004805 SRX020~ SRX02009~              1 control  Sprag~
## 6 ENSRNOG00000004805 SRX020~ SRX02008~              2 control  Sprag~
## 7 ENSRNOG00000004805 SRX020~ SRX02009~              1 L5 SNL   Sprag~
## 8 ENSRNOG00000004805 SRX020~ SRX02009~              2 L5 SNL   Sprag~
```

biobroom example

```
one_gene %>%
        ggplot(aes(x = protocol, y = value)) +
        geom_beeswarm() +
        geom_label_repel(aes(label = sample)) +
        ggtitle("Values by sample for gene ENSRNOG00000004805",
                subtitle = "Samples are labeled with their sample ID")
```



Accessor functions

Accessor functions are written to allow you to extract specific elements from complex data structures.

For example, you can extract the assay data (in a matrix format) from an `ExpressionSet` object using the `exprs` accessor function:

```
exprs(hammer)[1:6, 1:4]
```

```
##                    SRX020102 SRX020103 SRX020104 SRX020105
## ENSRNOG00000000001         2         4        18        24
## ENSRNOG00000000007         4         1         3         1
## ENSRNOG00000000008         0         1         4         2
## ENSRNOG00000000009         0         0         0         0
## ENSRNOG00000000010        19        10        19        13
## ENSRNOG00000000012         7         5         1         0
```

Accessor function example

You can use the `pData` accessor function to extract the phenotype data (as a "messy" dataframe) from an `ExpressionSet` object:

```
pData(hammer)[1:6, 1:3]
```

```
##                 sample.id num.tech.reps protocol
## SRX020102       SRX020102             1  control
## SRX020103       SRX020103             2  control
## SRX020104       SRX020104             1   L5 SNL
## SRX020105       SRX020105             2   L5 SNL
## SRX020091-3   SRX020091-3             1  control
## SRX020088-90 SRX020088-90             2  control
```

Accessor functions

Accessor functions are meant to be more robust than more elemental tools.

Developers are meant to consider these a "contract" with users, so their changes "under the hood" shouldn't affect code that uses these functions.

Extracting from R objects

The most elemental way of extracting data from R objects is to use the `$` or `@` operators.

For S4 objects (most Bioconductor objects), `@` is used for this extraction:

```
hammer@phenoData@data[1:4, 1:3]
```

```
##             sample.id num.tech.reps protocol
## SRX020102 SRX020102              1  control
## SRX020103 SRX020103              2  control
## SRX020104 SRX020104              1   L5 SNL
## SRX020105 SRX020105              2   L5 SNL
```

Extracting from R objects

You can use the `str` function (short for "structure") to investigate what's stored in any type of R object, to figure out what you can extract:

```
str(hammer@phenoData)
```

```
## Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
##   ..@ varMetadata      :'data.frame':    5 obs. of  1 variable:
##   .. ..$ labelDescription: chr [1:5] NA NA NA NA ...
##   ..@ data             :'data.frame':    8 obs. of  5 variables:
##   .. ..$ sample.id    : Factor w/ 8 levels "SRX020088-90",..: 5 6 7
##   .. ..$ num.tech.reps: num [1:8] 1 2 1 2 1 2 1 2
##   .. ..$ protocol     : Factor w/ 2 levels "control","L5 SNL": 1 1 2
##   .. ..$ strain       : Factor w/ 1 level "Sprague Dawley": 1 1 1 1
##   .. ..$ Time         : Factor w/ 3 levels "2months","2 months",..:
##   ..@ dimLabels        : chr [1:2] "sampleNames" "sampleColumns"
##   ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"]
##   .. .. ..@ .Data:List of 1
##   .. .. .. ..$ : int [1:3] 1 1 0
##   .. .. ..$ names: chr "AnnotatedDataFrame"
```

Extracting from R objects

In some cases, especially for large data, a slot in an object might just point to an environment—it might be trickier in these cases to extract the data directly from the object.

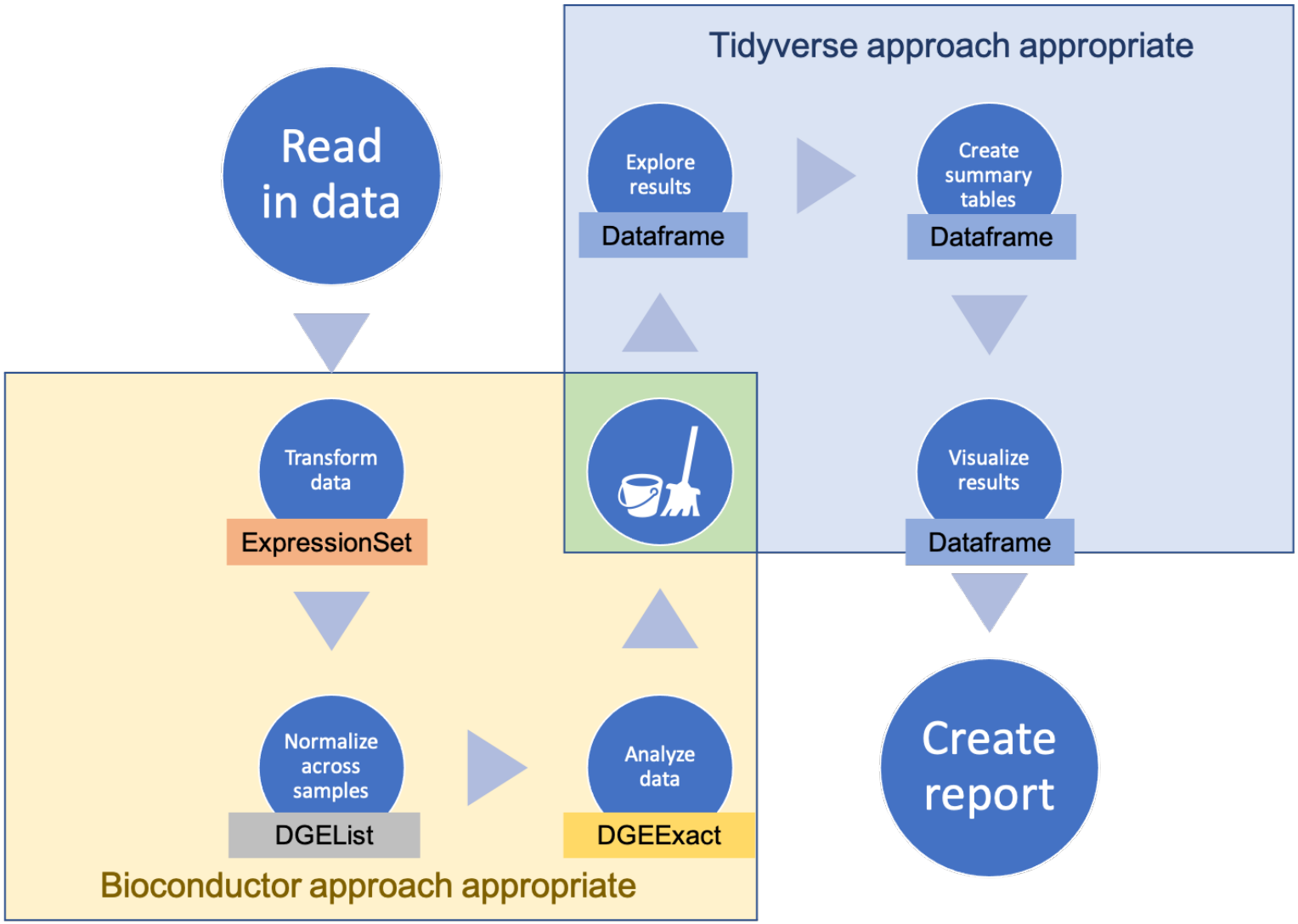
For example, the `assayData` slot in the example `ExpressionSet` object points to an environment, rather than directly storing the assay data:

```
hammer@assayData
```

```
## <environment: 0x7f8a039316d8>
```

Data structures across a workflow

You can move your data among different structures across a workflow, including from more complex data to simpler data structures.



Future directions in connecting workflows

Future directions: ggbio package

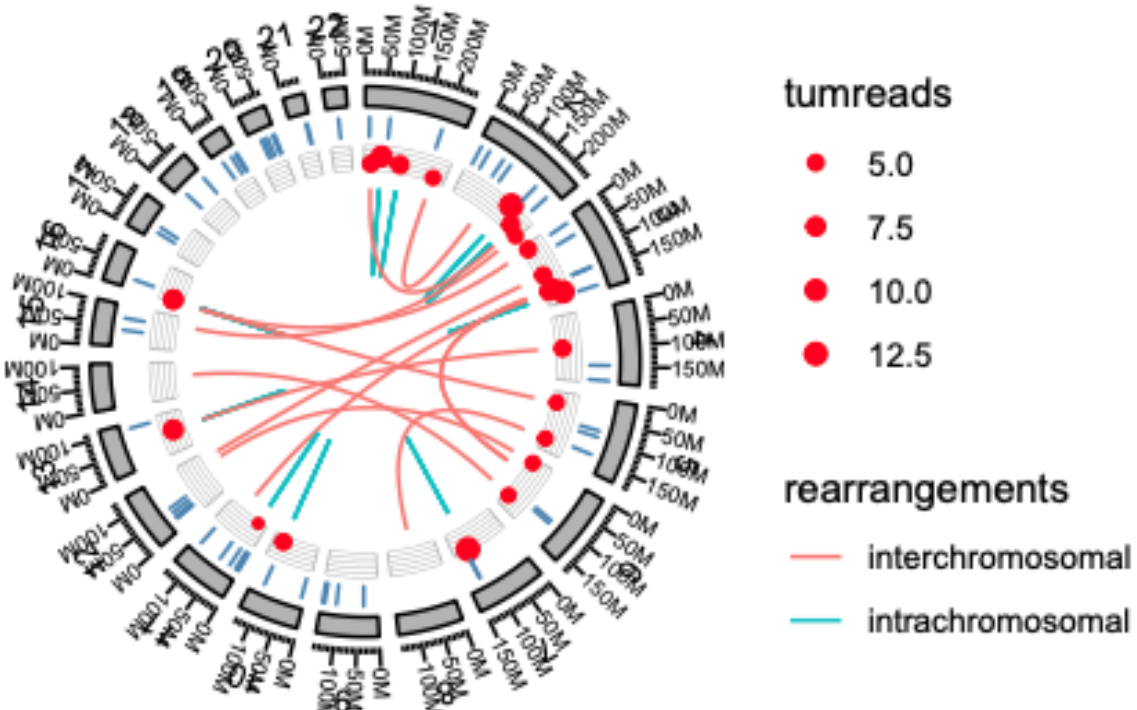
The ggbio package allows you to coordinate Bioconductor-style analysis with the tidyverse style of visualization, which is based on the ggplot2 package.

This package enables the use of "layers" of small simple functions to build up a plot, aligning with the general tidyverse approach of combining small, simple tools to do complex things.

Future directions: ggbio package

Here is an example from the ggbio vignette, using data stored in a GRanges data structure:

```
p <- ggbio() +
   circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangements)) +
  circle(gr.crc1, geom = "point", aes(y = score, size = tumreads),
                 color = "red", grid = TRUE) + scale_size(range = c(1, 2.5)) +
  circle(mut.gr, geom = "rect", color = "steelblue") +
    circle(hg19sub, geom = "ideo", fill = "gray70") +
    circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```



Future directions: List-columns

In some areas, there is a movement to allow a tidyverse approach even in the context of very complex data that doesn't fit naturally into a dataframe.

More complex "tidy" dataframes are being developed that allow some columns to be **list-columns** and store pretty complex data within a cell of the dataframe.

Future directions: List-columns

One example is with spatial data. List-columns are being powerfully used, for example, within the `sf` package for geospatial data in R.

```
## Simple feature collection with 100 features and 6 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID):    4267
## proj4string:    +proj=longlat +datum=NAD27 +no_defs
## precision:      double (default; no precision model)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79                              geom
## 1  1091     1      10  1364     0      19 MULTIPOLYGON(((-81.47275543...
## 2   487     0      10   542     3      12 MULTIPOLYGON(((-81.23989105...
## 3  3188     5     208  3616     6     260 MULTIPOLYGON(((-80.45634460...
```

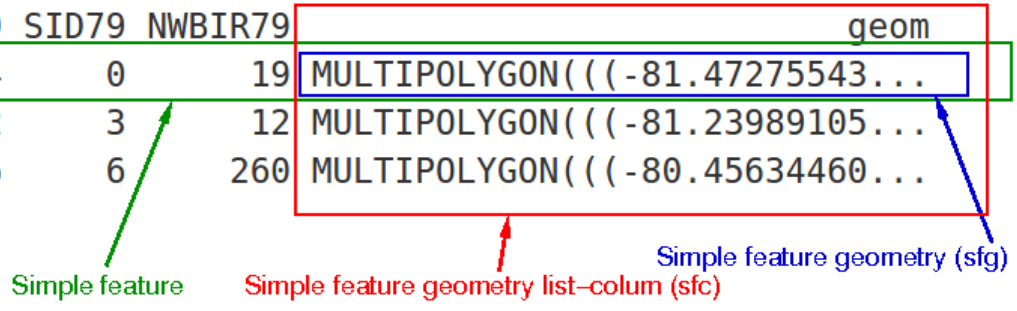


Image source: https://r-spatial.github.io/sf/articles/sf1.html