Scanpy for large-scale single-cell analysis

F. Alexander Wolf, Helmholtz Munich

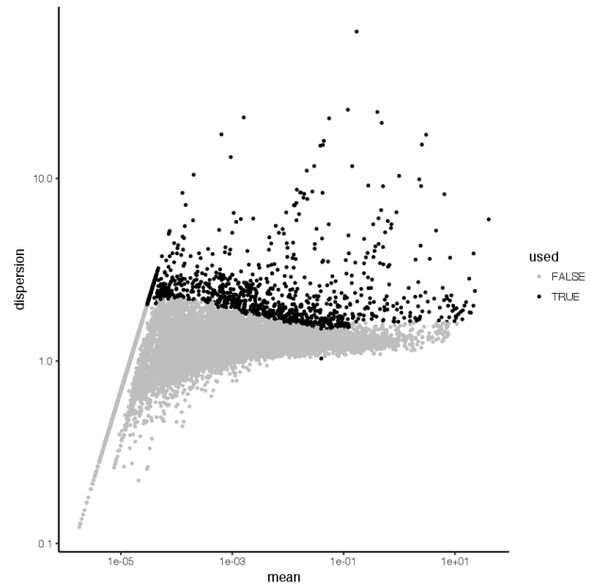
May 4, 2017 - *Single-Cell Mingling* Retreat - Charité Berlin

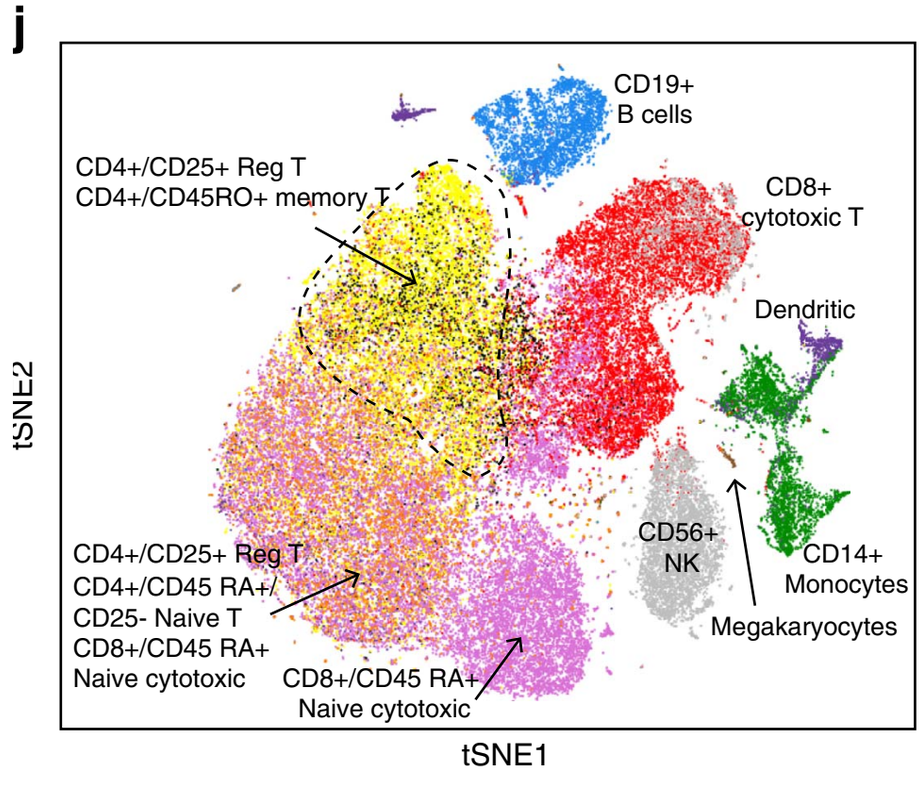
Zheng *et al.*, Nat. Comm. (2017)

Cell Ranger for 68k PBMC cells

The optimized official 10X Genomics pipeline might give you a wait and crash your laptop.

- Normalizing, filtering, selecting highly-variable genes: ~5 min vs. ~6 s for 3k
- Normalizing again, PCA: ~2 min vs. ~6 s for 3k
- a first tSNE visualization: ~26 min vs. ~27 s for 3k





Satja Lab, NY Genome Center (2017) / Macosko et al., Cell (2015)

Seurat for 3k PBMC cells

Powerful analysis toolkit, but not optimized for large data sets.

- Normalizing, filtering, selecting highly-variable genes: ~30 s
- Regressing out unwanted batch effects and other unwanted variation: ~2 min
- Selecting PCs, Clustering, Visualization etc.

```
#note that this overwrites pbmc@scale.data. Therefore, if you intend to use RegressOut, you can set do.scale=F an
d do.center=F in the original object to save some time.
pbmc <- RegressOut(pbmc, latent.vars = c("nUMI", "percent.mito"))
```

```
library(Seurat)
library(dplyr)
library(Matrix)

# Load the PBMC dataset
pbmc.data <- Read10X("~/Downloads/filtered_gene_bc_matrices/hg19/")

#Examine the memory savings between regular and sparse matrices
dense.size <- object.size(as.matrix(pbmc.data))
dense.size
```

```
## 709264728 bytes
```

```
sparse.size <- object.size(pbmc.data)
sparse.size
```

```
## 38715120 bytes
```

```
dense.size/sparse.size
```

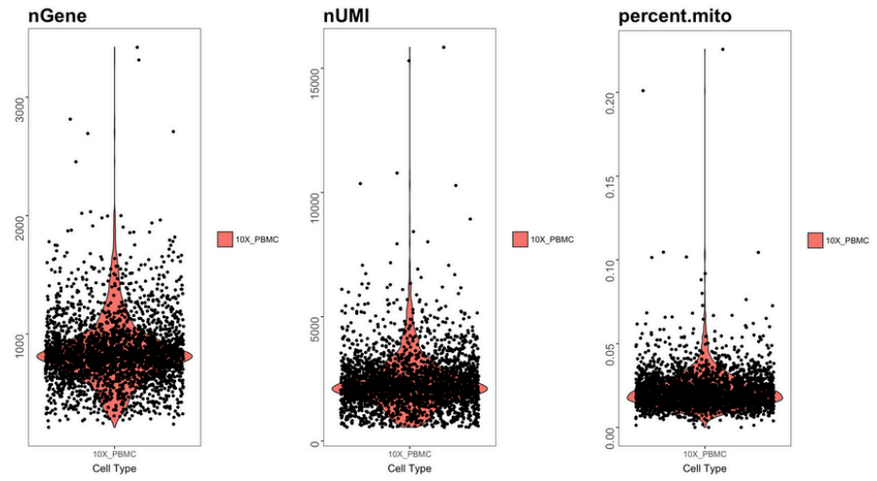
```
## 18.3200963344554 bytes
```

```
# Initialize the Seurat object with the raw (non-normalized data)
# Note that this is slightly different than the older Seurat workflow, where log-normalized values were passed in
directly.
# You can continue to pass in log-normalized values, just set do.logNormalize=F in the next step.
pbmc <- new("seurat", raw.data = pbmc.data)

# Keep all genes expressed in >= 3 cells, keep all cells with >= 200 genes
# Perform log-normalization, first scaling each cell to a total of 1e4 molecules (as in Macosko et al. Cell 2015)
pbmc <- Setup(pbmc, min.cells = 3, min.genes = 200, do.logNormalize = T, total.expr = 1e4, project = "10X_PBMC")
```

```
# The number of genes and UMIs (nGene and nUMI) are automatically calculated for every object by Seurat.
# For non-UMI data, nUMI represents the sum of the non-normalized values within a cell
# We calculate the percentage of mitochondrial genes here and store it in percent.mito using the AddMetaData.
# The % of UMI mapping to MT-genes is a common scRNA-seq QC metric.
# NOTE: You must have the Matrix package loaded to calculate the percent.mito values.
mito.genes <- grep("^MT-", rownames(pbmc@data), value = T)
percent.mito <- colSums(expm1(pbmc@data[mito.genes, ]))/colSums(expm1(pbmc@data))

#AddMetaData adds columns to object@data.info, and is a great place to stash QC stats
pbmc <- AddMetaData(pbmc, percent.mito, "percent.mito")
VlnPlot(pbmc, c("nGene", "nUMI", "percent.mito"), nCol = 3)
```



github.com/theislab/scanpy (2017)

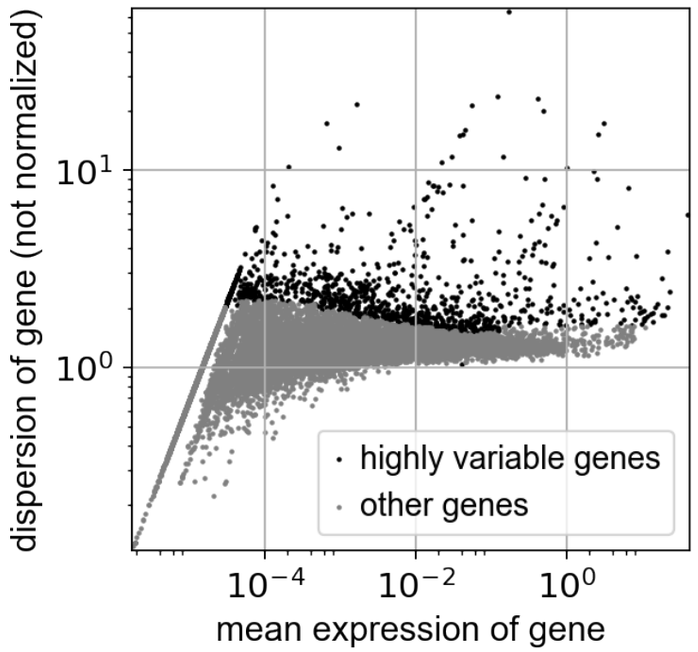
Scanpy for 68k PBMC cells

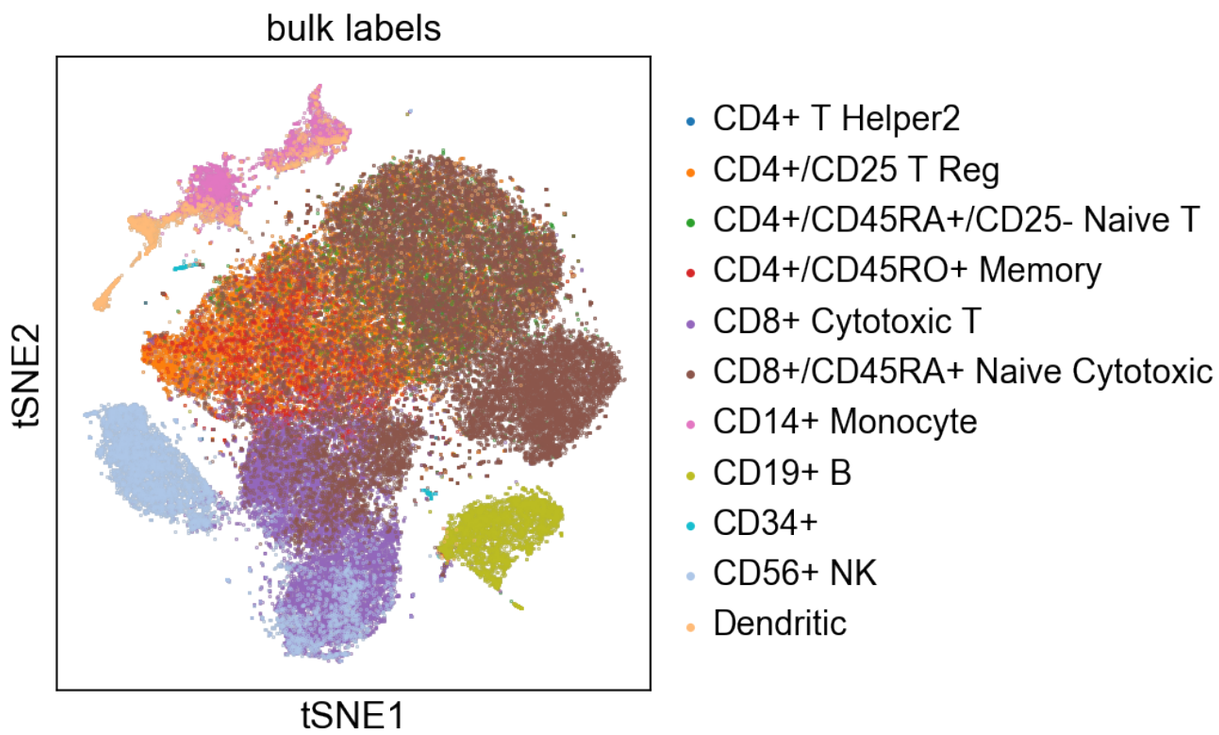
Scanpy's modular structure and *flavored* functions allow to produce *exactly* the same results as with Cell Ranger.

- Normalizing, filtering, selecting highly-variable genes: ~14 s
  vs. ~5 min for Cell Ranger

- Normalizing again, PCA: ~17 s
  vs. ~2 min for Cell Ranger

- a first tSNE visualization: ~5 min vs. ~26 min for Cell Ranger





github.com/theislab/scanpy (2017)

Scanpy for 3k PBMC cells

Scanpy's modular structure and *flavored* functions allow to produce *exactly* the same results as with Seurat.

- Normalizing etc. ~3 s vs. 30 s for Seurat
- Regressing out…: ~10 s vs. ~2 min for Seurat
- Selecting PCs, Clustering, Visualization etc.

In [3]:
```
filename_data = './data/pbmc3k_filtered_gene_bc_matrices/hg19/matrix.mtx'
filename_genes = './data/pbmc3k_filtered_gene_bc_matrices/hg19/genes.tsv'
filename_barcodes = './data/pbmc3k_filtered_gene_bc_matrices/hg19/barcodes.tsv'
adata = sc.read(filename_data).transpose()
adata.var_names = np.loadtxt(filename_genes, dtype='S')[:, 1]
adata.smp_names = np.loadtxt(filename_barcodes, dtype='S')
```

```
reading file ./write/data/pbmc3k_filtered_gene_bc_matrices/hg19/matrix.h5
```

Basic filtering.

In [4]:
```
adata.smp['n_counts'] = np.sum(adata.X, axis=1).A1
sc.pp.filter_cells(adata, min_genes=200)
sc.pp.filter_genes(adata, min_cells=3)
```

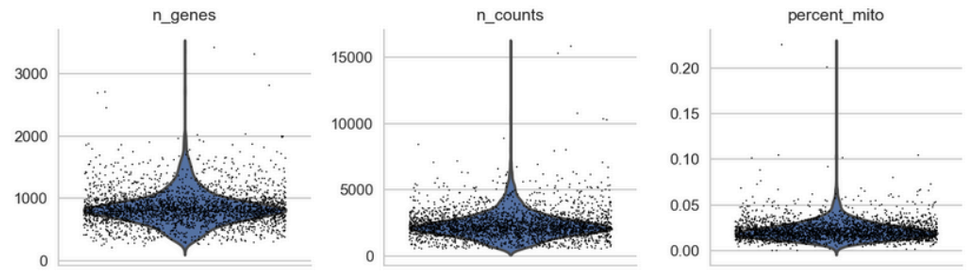
```
... filtered out 0 outlier cells
... filtered out 19024 genes that are detected in less than 3 cells
```

Plot some information about mitochondrial genes, important for quality control

In [5]:
```
mito_genes = np.array([name for name in adata.var_names
                       if bool(re.search("^MT-", name))])
# for each cell compute fraction of counts in mito genes vs. all genes
adata.smp['percent_mito'] = np.sum(adata[:, mito_genes].X, axis=1).A1 / np.sum(adata.X, axis=1).A1
# add the total counts per cell as sample annotation to adata
adata.smp['n_counts'] = np.sum(adata.X, axis=1).A1
```

A violin plot of the computed quality measures.

In [6]:
```
sc.pl.violin(adata, ['n_genes', 'n_counts', 'percent_mito'], jitter=0.4, show=True)
```

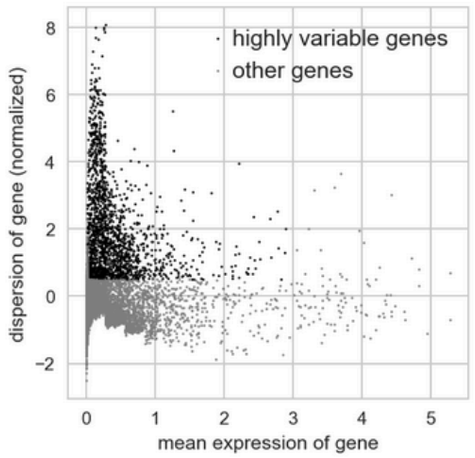


github.com/theislab/scanpy (2017)

Scanpy for 3k PBMC cells

In [9]:
```
sc.pp.normalize_per_cell(adata, scale_factor=1e4)
result = sc.pp.filter_genes_dispersion(adata.X, log=True, flavor='seurat', min_mean=0.0125, max_mean=3, min_disp=0.5)
sc.pl.filter_genes_dispersion(result)
```

```
... filter highly varying genes by dispersion and mean
    using `min_disp`, `max_disp`, `min_mean` and `max_mean`
--> set `n_top_genes` to simply select top-scoring genes instead
```



In [11]:
```
adata_corrected = sc.pp.regress_out(adata,
                                    smp_keys=['n_counts', 'percent_mito'],
                                    copy=True)
```

```
0:00:00.000 - regress out ['n_counts', 'percent_mito']
... sparse input is densified and may lead to huge memory consumption

0:00:09.418 - finished
```

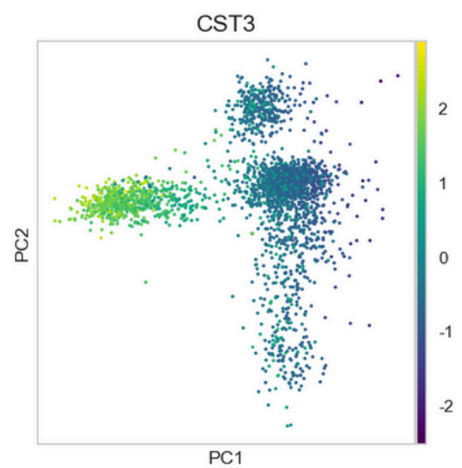
Compute PCA and make a scatter plot.

In [12]:
```
sc.pp.scale(adata_corrected, max_value=10)
```

```
clipping at max_value 10
```

In [13]:
```
sc.tl.pca(adata_corrected)
adata_corrected.smp['X_pca'] *= -1  # multiply by 1 for correspondence with R
sc.pl.pca_scatter(adata_corrected, color='CST3', right_margin=0.2)
```

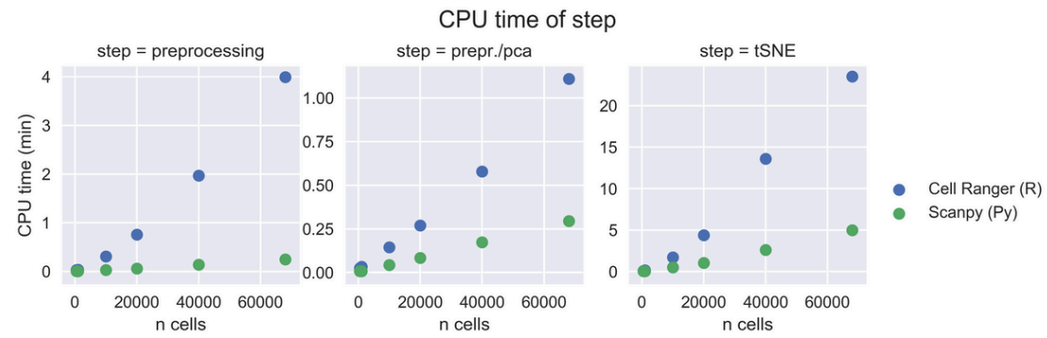
```
0:00:00.000 - compute PCA with n_comps = 10
0:00:00.668 - finished, added
    the data representation "X_pca" (adata.smp)
    the loadings "PC1", "PC2", ... (adata.var)
    and "pca_variance_ratio" (adata.add)
```

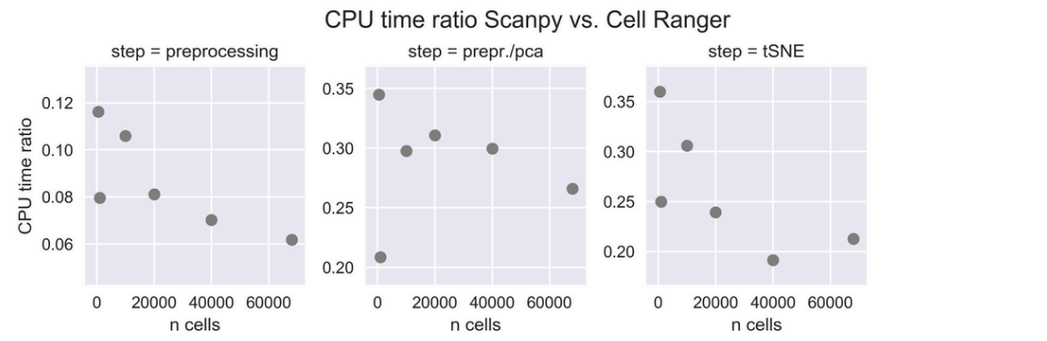


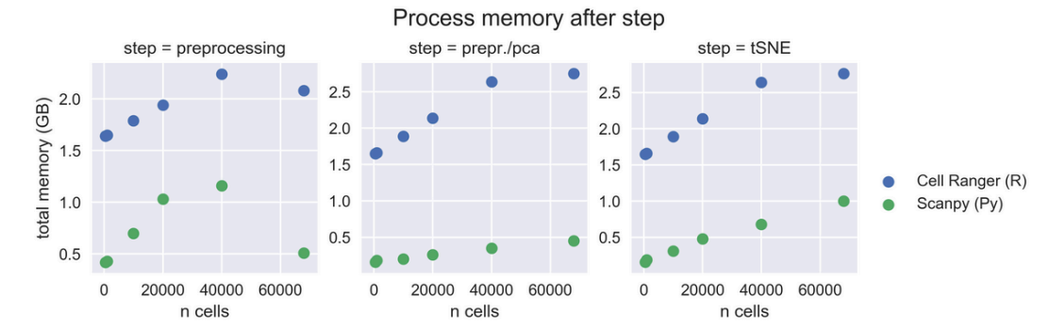
github.com/theislab/scanpy (2017)

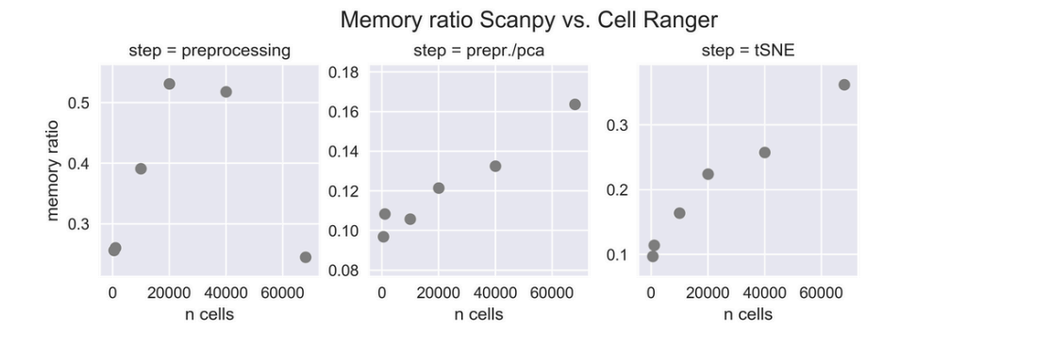
Scanpy vs *Cell Ranger*

- Scanpy is about a factor 10 faster in preprocessing and about a factor 3 - 5 in following steps.
- It's at least a factor 3 - often a factor 10 - more memory efficient.
- Provides a high level of modularity, e.g., fully using sparsity gives further improvements.





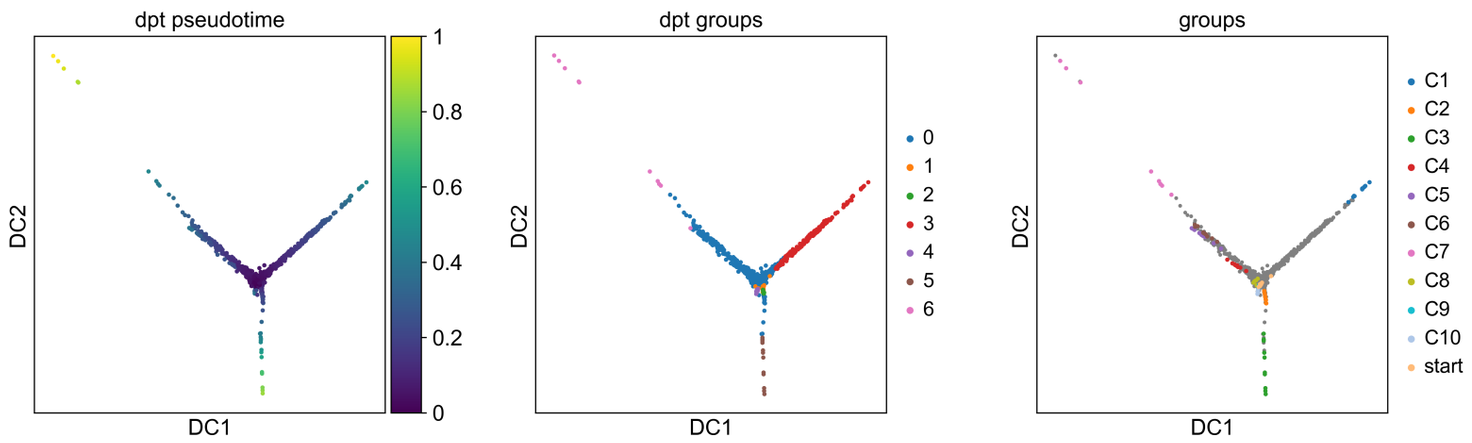




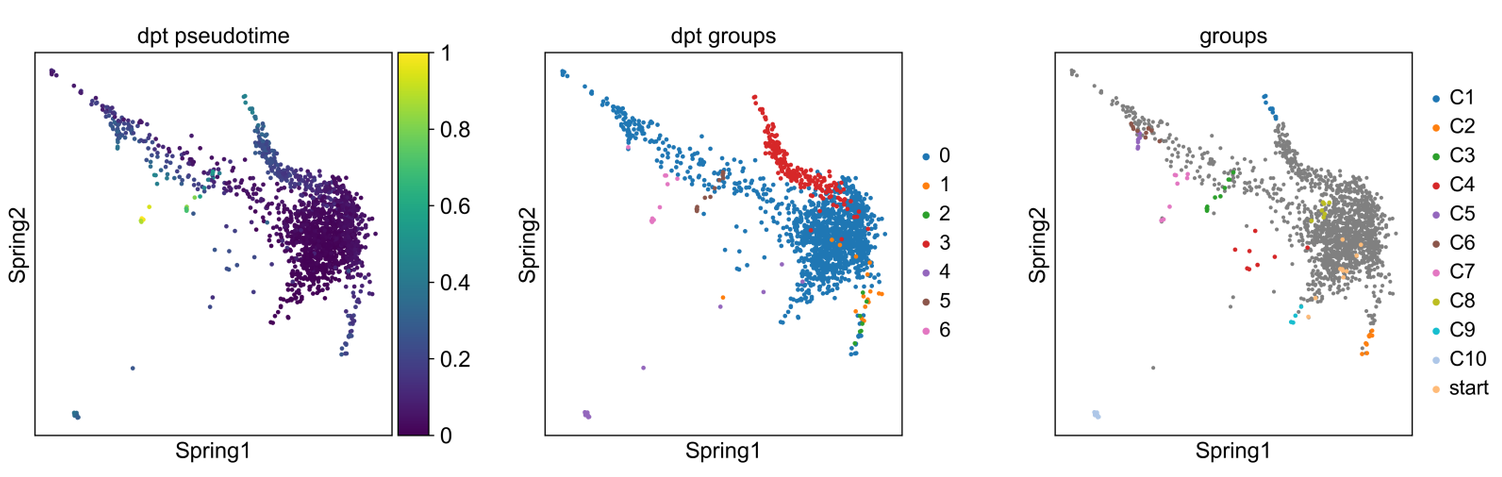
Collab. with Göttgens Lab, Cambridge / github.com/theislab/scanpy (2017)

Beyond Diffusion Pseudotime

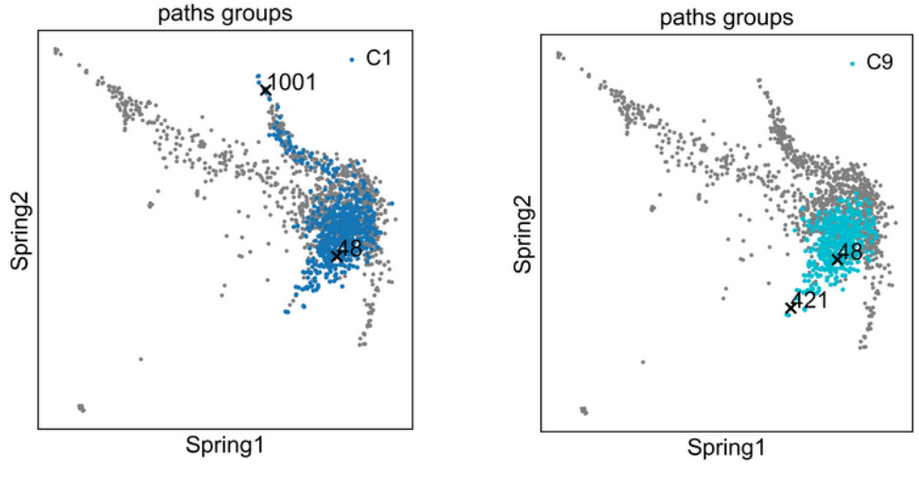
- hematopoiesis: differentiation from stem cells into 10 cell types



- DPT recovers the final cell types *fate* groups, but transitions/trajectories remain unclear



- Can be addressed by computing the *most probable path* and the *fluctuations* around it.



kaggle.com/c/data-science-bowl-2017 / github.com/theislab/scanpy (2017)

More Machine Learning

- Python is the de facto standard for Deep Learning. It is chosen over the *statistical programming* language R as it's a *general purpose* language, offering much more possibilities and control.

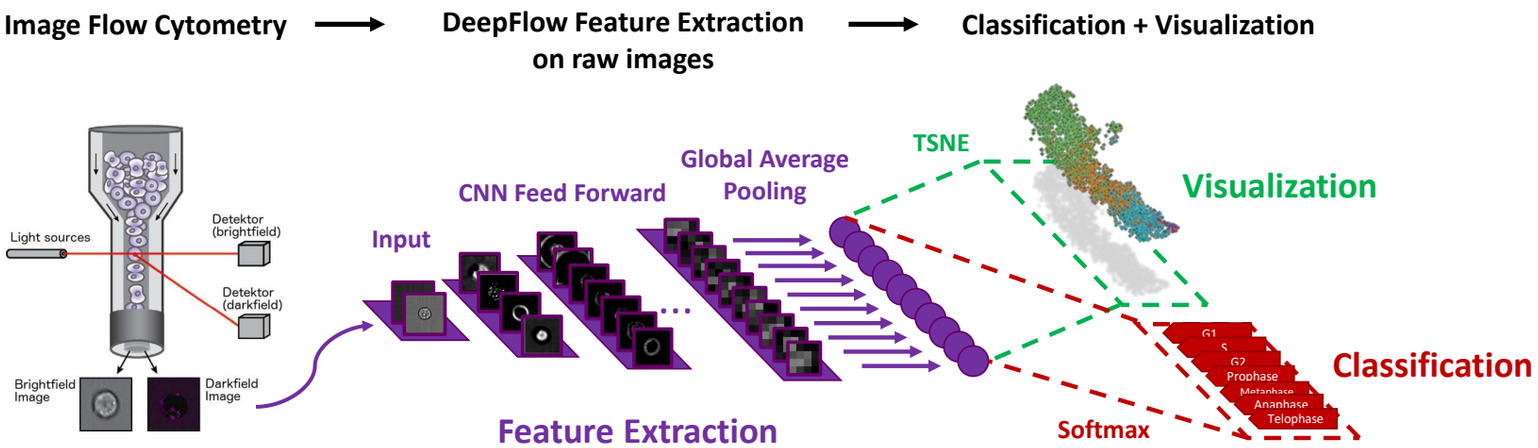
- Scanpy already now allows integrating advanced Machine Learning tools. Most prominent example: scLVM Buettner *et al.*, Nat. Biotechn. (2015) / bioRxiv (2017)

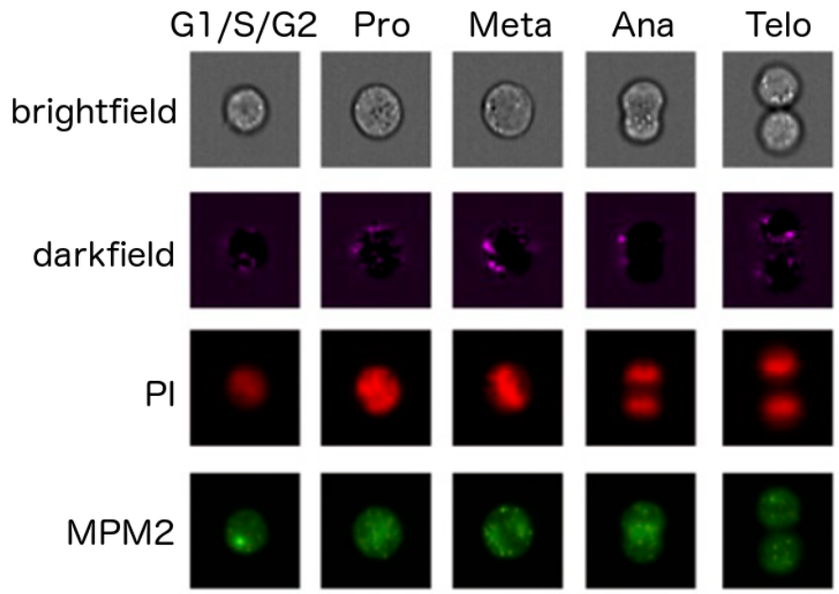
- Deep Learning will likely become prominent also in molecular biology. Scanpy already now shares the core features of one of the winning pipelines (ranked 7th out of 2000 internationally, ranked 1st across Germany) of the Data Science Bowl (2017).

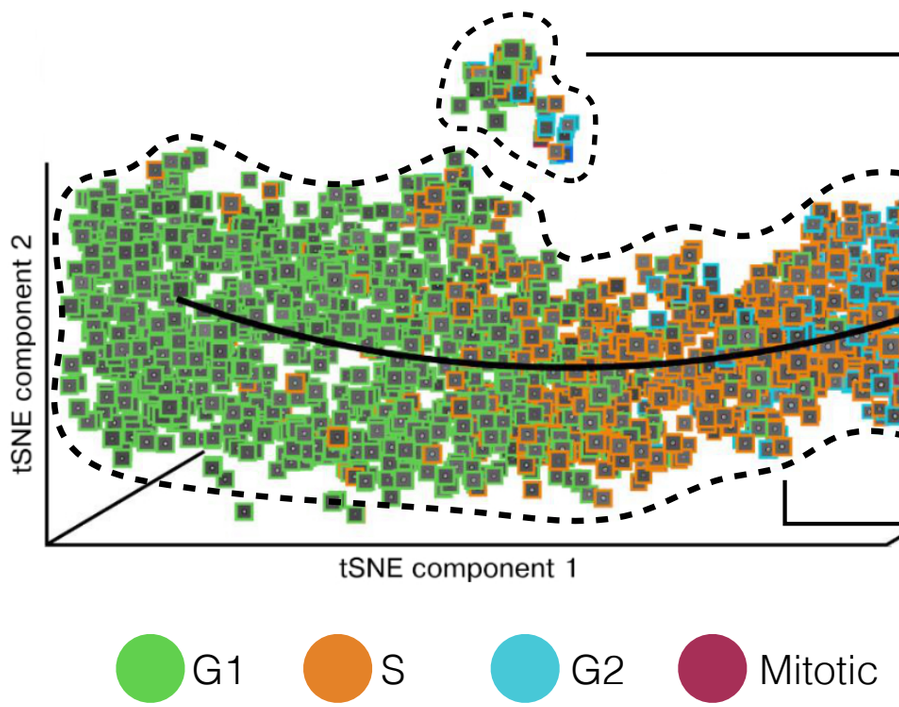
Eulenberg, Köhler et al. bioRxiv doi:10.1101/081364 (2016)

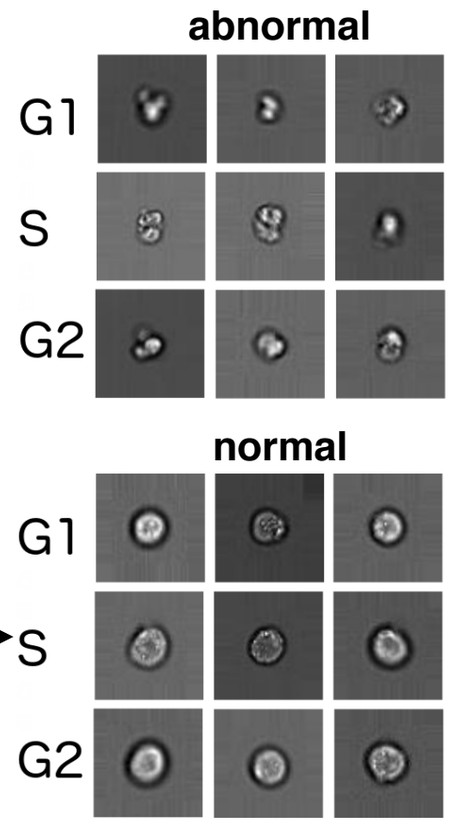
High-throughput Microscopy

- cell cycle classification
- detection of abnormal cells









Thanks to

Helmholtz Munich: Fabian Theis, Philipp Angerer, Lukas Simon, David Fischer, Sophie Tritschler, Maren Büttner, Niklas Köhler

U Cambridge: Fiona Hamey, Bertie Göttgens

Trapnell Lab, Seattle: Xiaoje Qiu

Thanks for your attention!