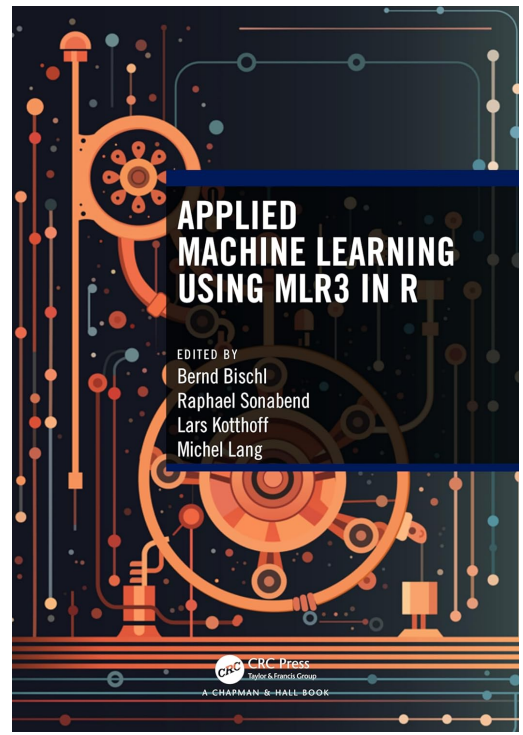
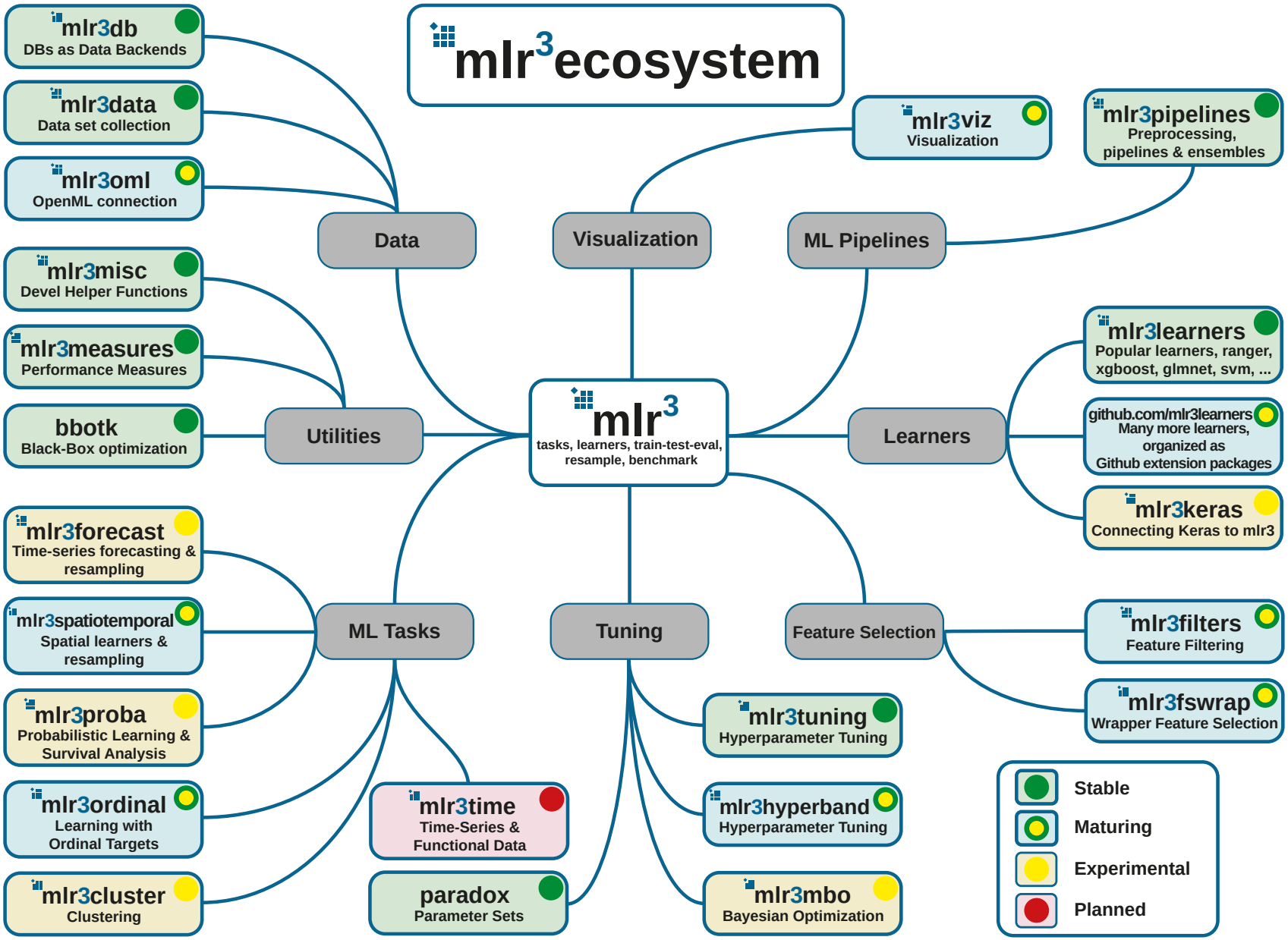
Machine Learning Pipelines in R Using mlr3pipelines

- **Website:** https://mlr-org.com/
- **Github:** https://github.com/mlr-org
- **Book:** https://mlr3book.mlr-org.com/



April 29, 2024

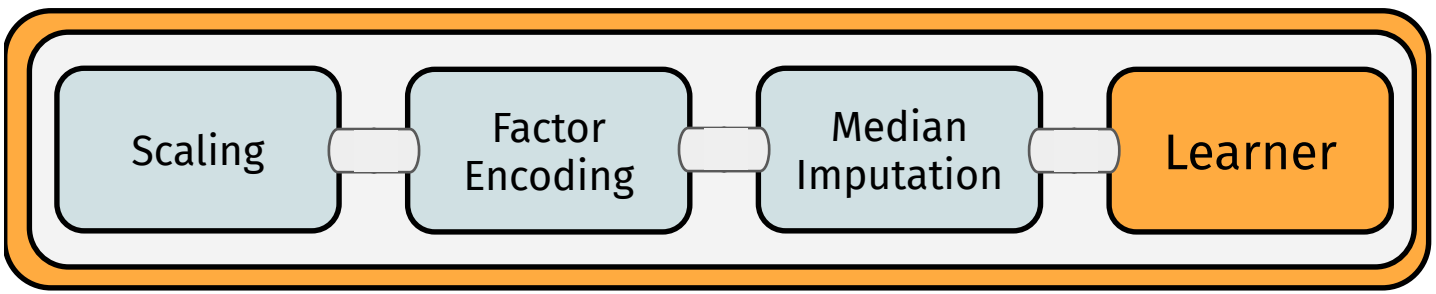

mlr3ecosystem
mlr3db
DBs as Data Backends
mlr3data
Data set collection
mlr3oml
OpenML connection
mlr3misc
Devel Helper Functions
mlr3measures
Performance Measures
bbotk
Black-Box optimization
mlr3forecast
Time-series forecasting & resampling
mlr3spatiotemporal
Spatial learners & resampling
mlr3proba
Probabilistic Learning & Survival Analysis
mlr3ordinal
Learning with Ordinal Targets
mlr3cluster
Clustering
Data
Utilities
ML Tasks
mlr3time
Time-Series & Functional Data
paradox
Parameter Sets
Visualization
mlr3viz
Visualization
ML Pipelines
mlr3pipelines
Preprocessing, pipelines & ensembles
mlr3
tasks, learners, train-test-eval, resample, benchmark
Tuning
mlr3tuning
Hyperparameter Tuning
mlr3hyperband
Hyperparameter Tuning
mlr3mbo
Bayesian Optimization
Learners
mlr3learners
Popular learners, ranger, xgboost, glmnet, svm, ...
github.com/mlr3learners
Many more learners, organized as Github extension packages
mlr3keras
Connecting Keras to mlr3
Feature Selection
mlr3filters
Feature Filtering
mlr3fswrap
Wrapper Feature Selection
Stable
Maturing
Experimental
Planned

MLR3PIPELINES

Machine Learning Workflows:

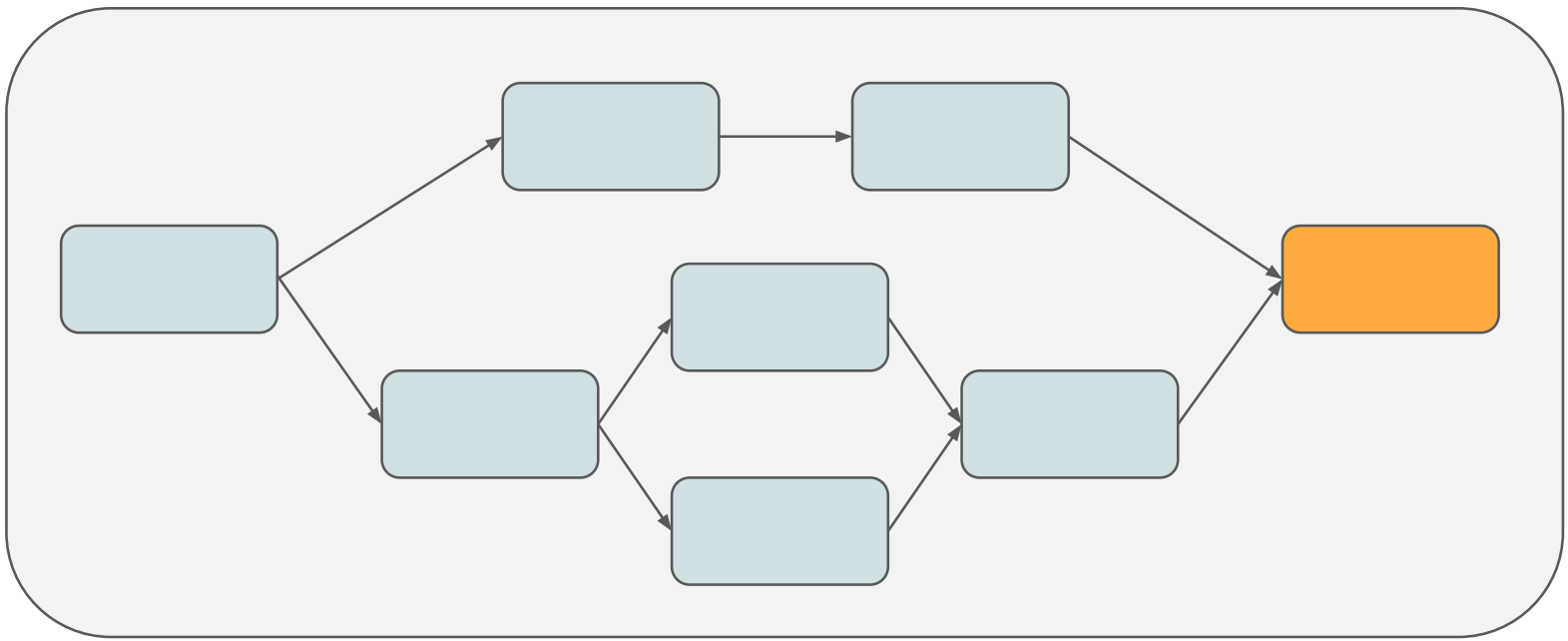
- **Preprocessing**: Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods**: Model averaging, model stacking
- `mlr3`: modular model fitting

$\Rightarrow$ `mlr3pipelines`: modular ML workflows



MACHINE LEARNING WORKFLOWS

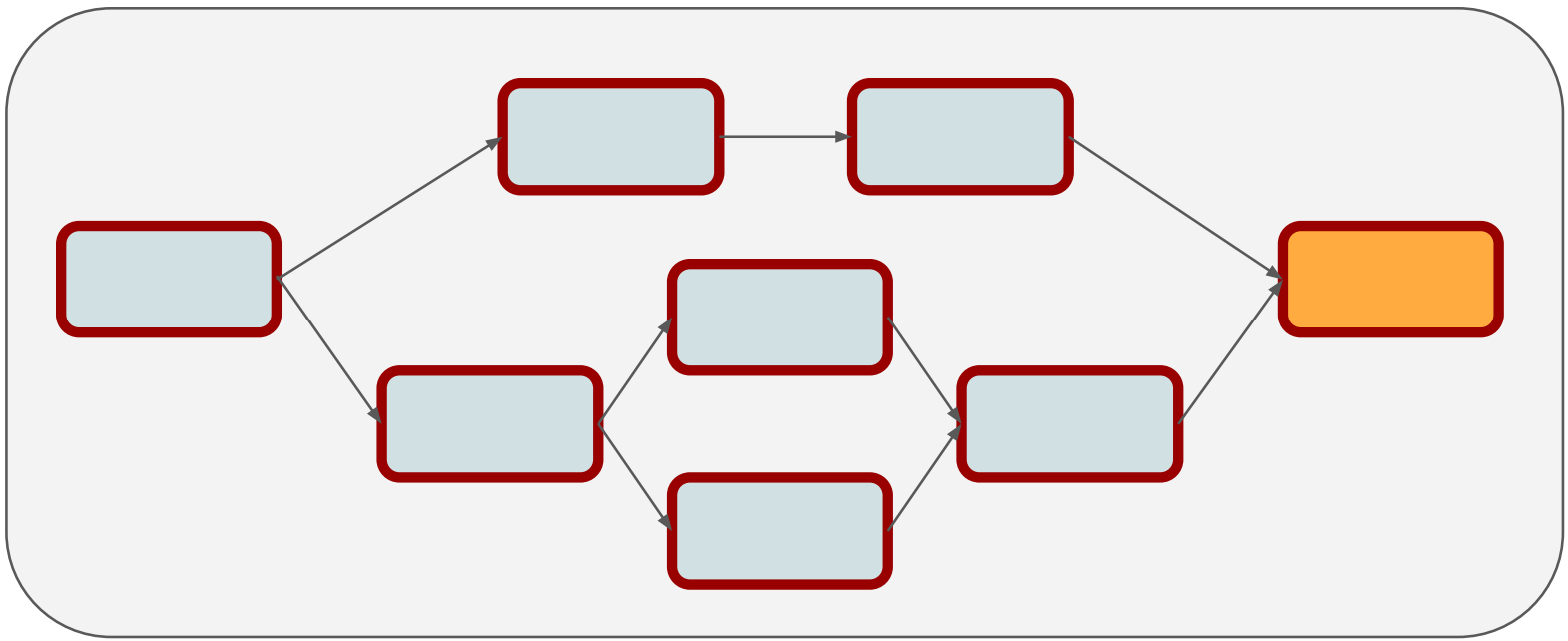
– what do they look like?



MACHINE LEARNING WORKFLOWS

– what do they look like?

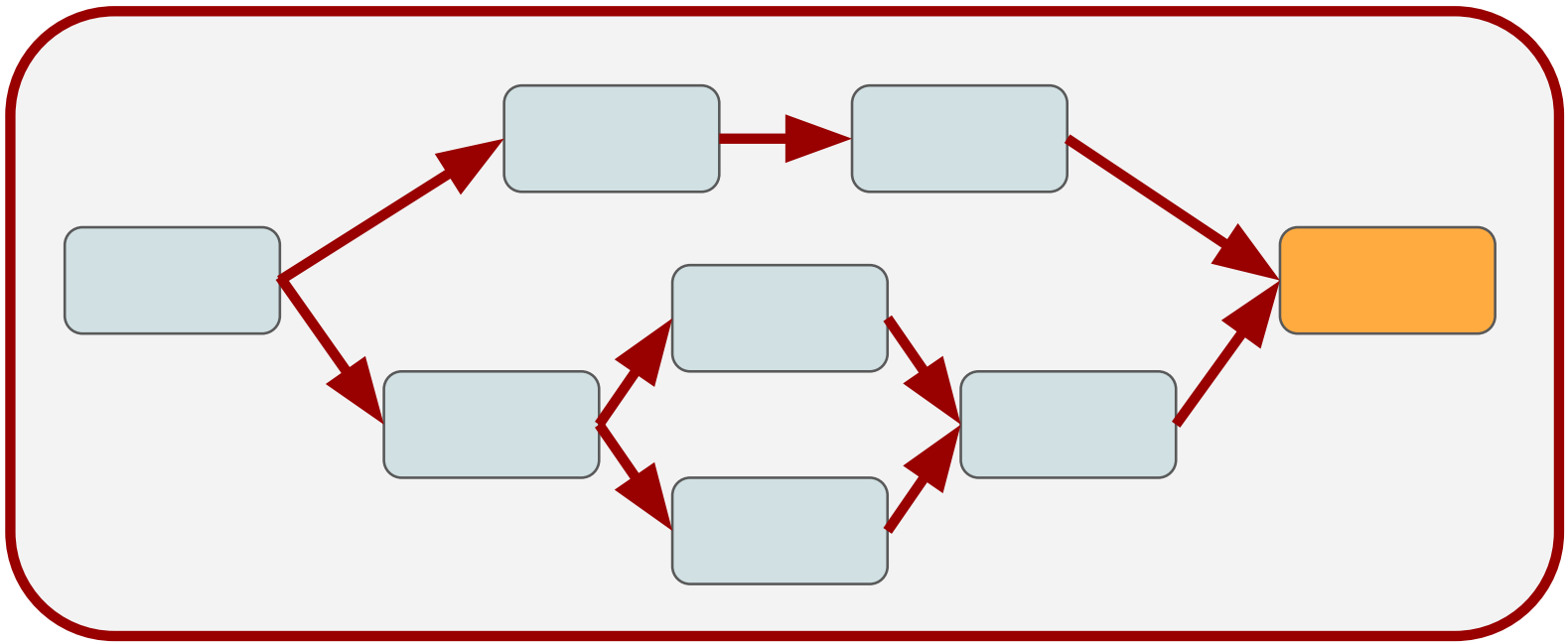
- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`



MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`
- **Structure**: in what *sequence* is it happening? $\rightarrow$ `Graph`

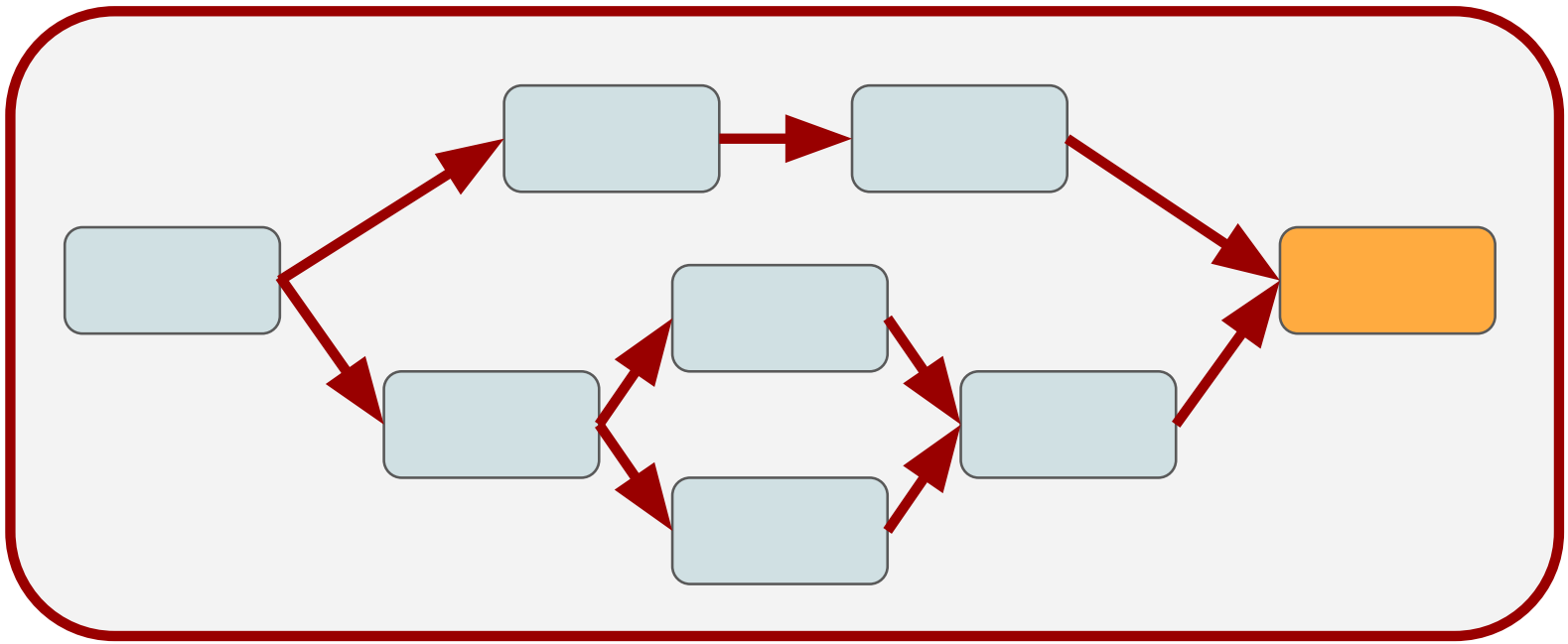


MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`
- **Structure**: in what *sequence* is it happening? $\rightarrow$ `Graph`

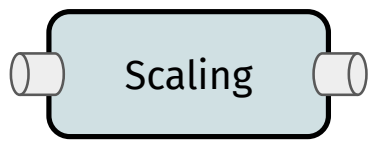
$\Rightarrow$ `Graph`: `PipeOps` as **nodes** with **edges** (data flow) between them



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

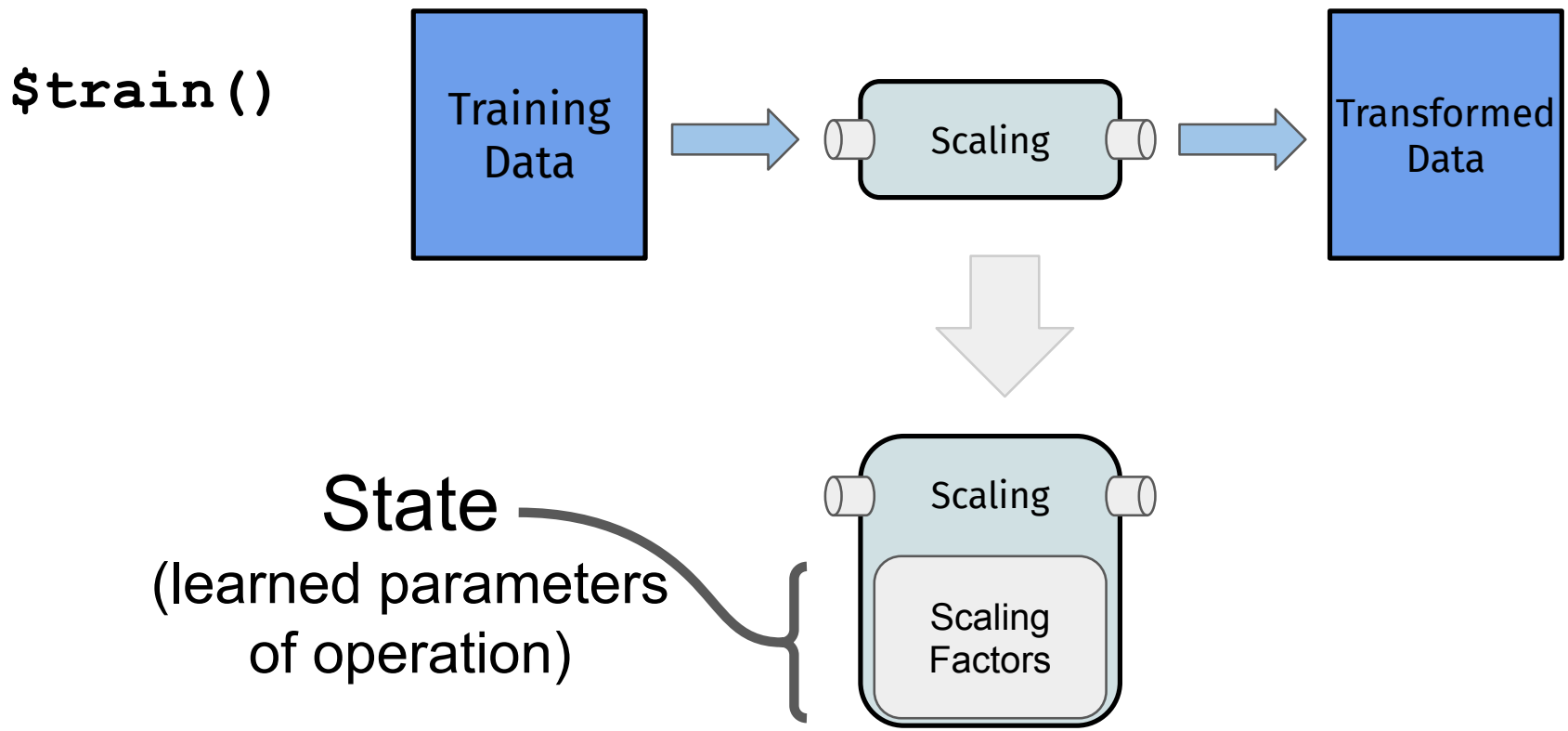
- `pip = po("scale")` to construct



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

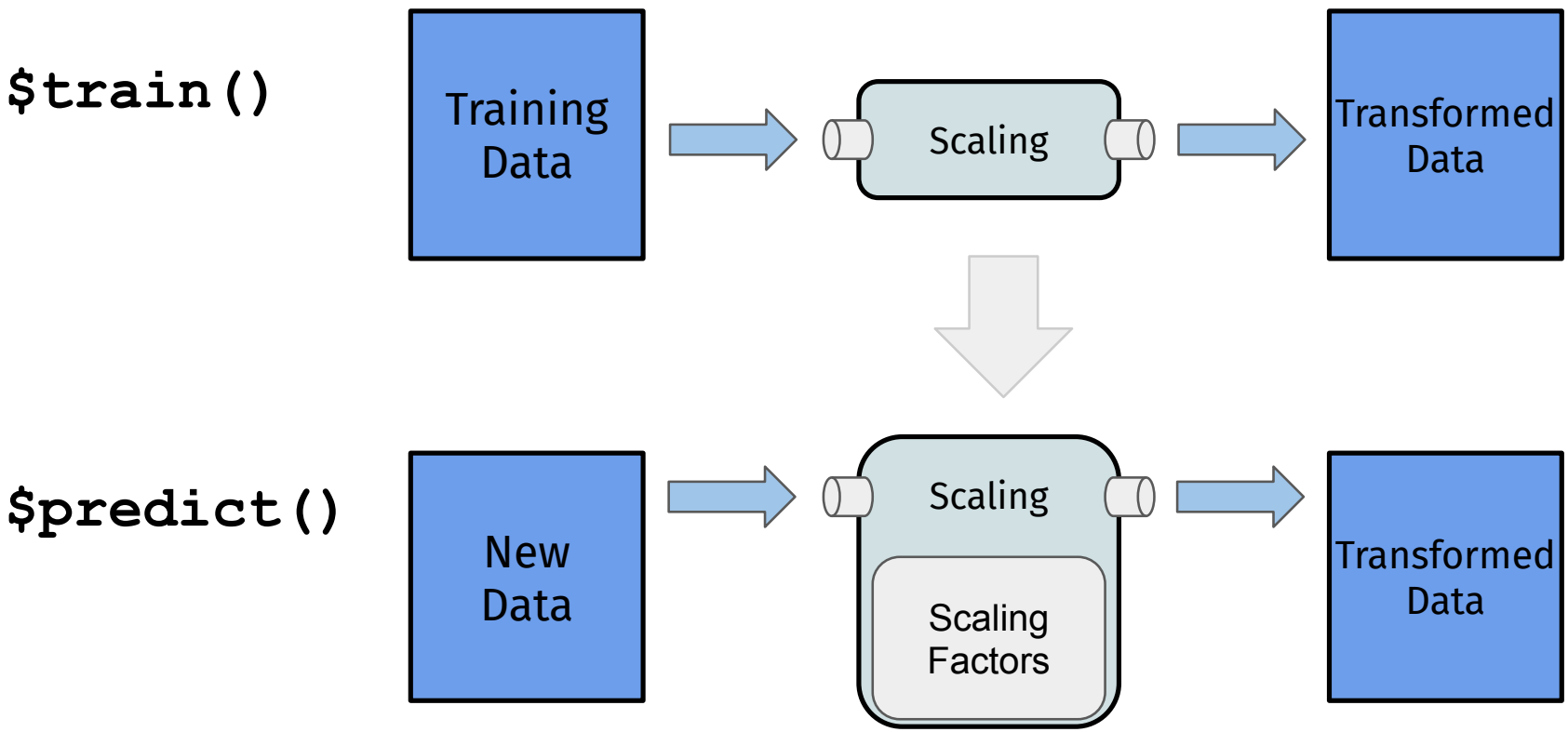
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

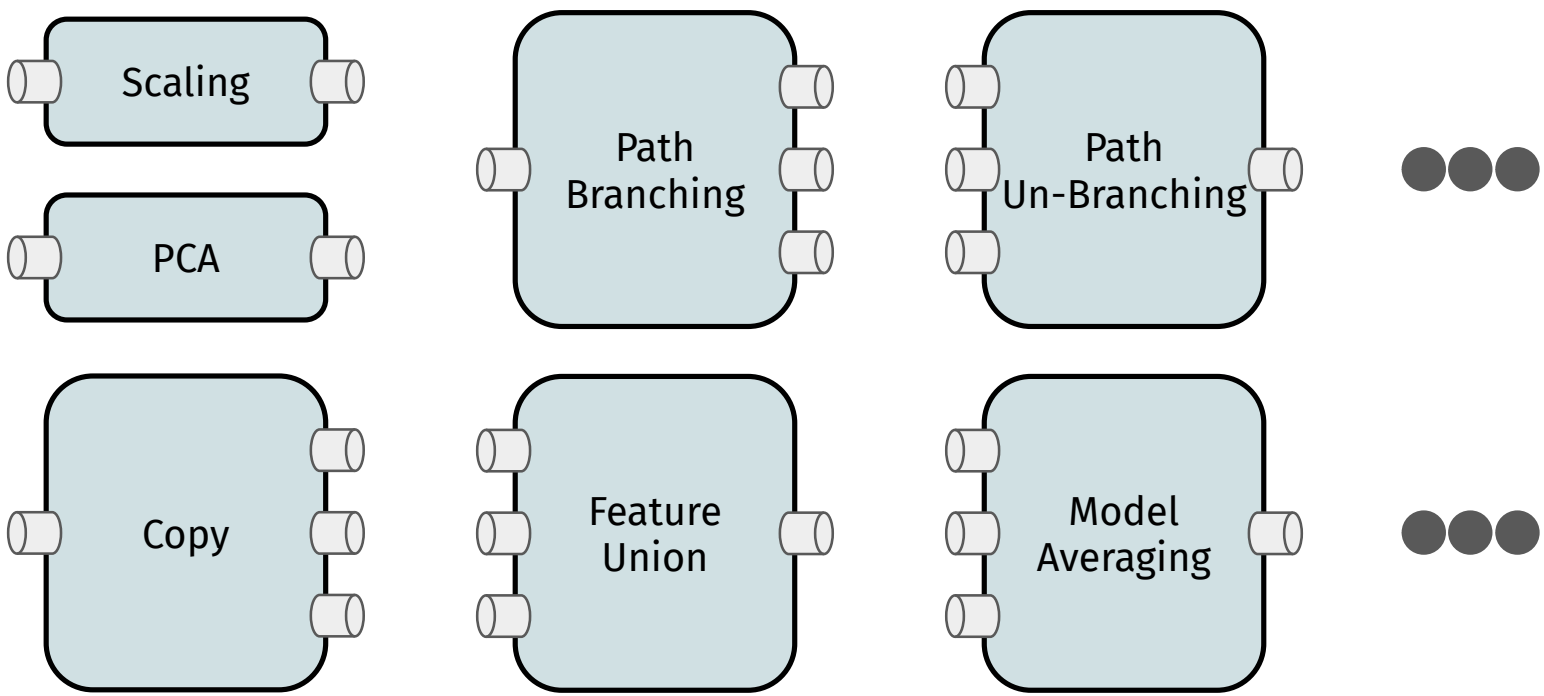
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`
- Multiple inputs or multiple outputs



THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

```
head(po$state, 2)

#> $center
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          3.8          1.2          5.8          3.1
#>
#> $scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>         1.77         0.76         0.83         0.44
```

THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)
```

```
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

```
smalltask = task$clone()
smalltask = smalltask$filter(1:3)
pred = po$predict(list(smalltask))
pred$output$data()
```

```
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

PIPEOPS SO FAR

```
mlr_pipeops$keys()
```

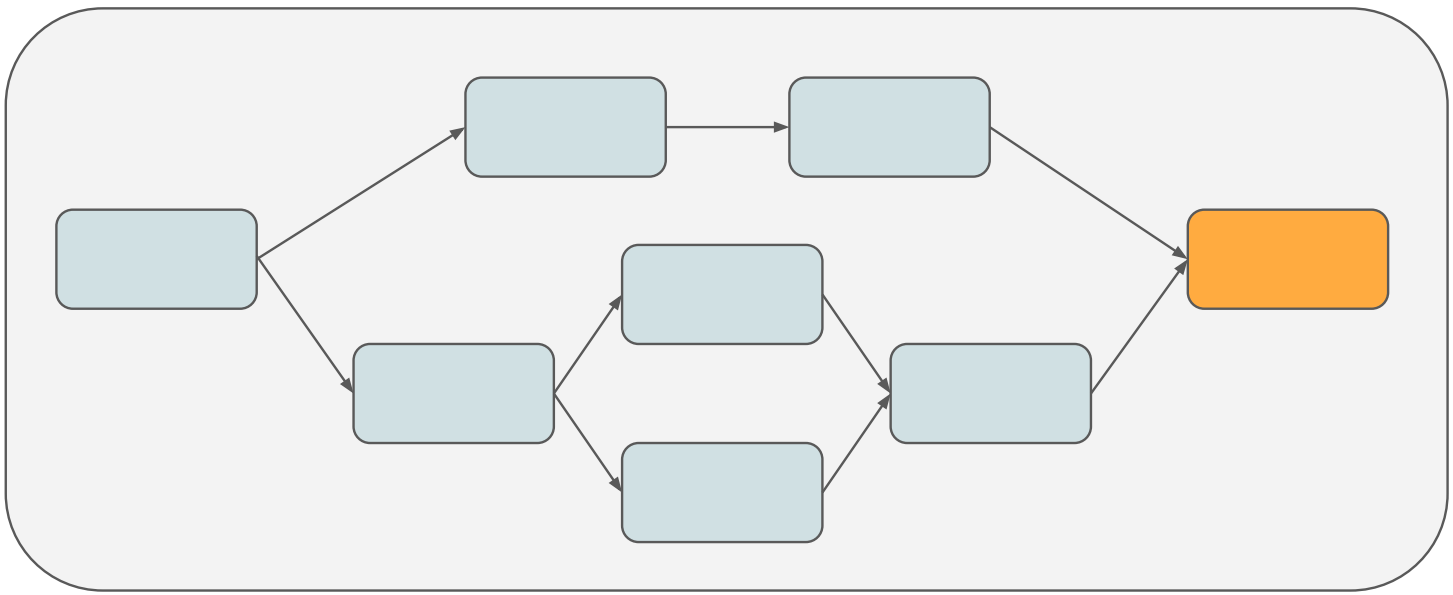
```
#>  [1] "boxcox"                "branch"                "chunk"
#>  [4] "classbalancing"        "classifavg"            "classweights"
#>  [7] "colapply"              "collapsefactors"       "colroles"
#> [10] "copy"                  "datefeatures"          "encode"
#> [13] "encodeimpact"          "encodelmer"            "featureunion"
#> [16] "filter"                "fixfactors"            "histbin"
#> [19] "ica"                   "imputeconstant"        "imputehist"
#> [22] "imputelearner"         "imputemean"            "imputemedian"
#> [25] "imputemode"            "imputeoor"             "imputesample"
#> [28] "kernelpca"             "learner"               "learner_cv"
#> [31] "missind"               "modelmatrix"           "multiplicityexply"
#> [34] "multiplicityimply"     "mutate"                "nmf"
#> [37] "nop"                   "ovrsplit"              "ovrunite"
#> [40] "pca"                   "proxy"                 "quantilebin"
#> [43] "randomprojection"      "randomresponse"        "regravg"
#> [46] "removeconstants"       "renamecolumns"         "replicate"
#> [49] "scale"                 "scalemaxabs"           "scalerange"
#> [52] "select"                "smote"                 "spatialsign"
#> [55] "subsample"             "targetinvert"          "targetmutate"
#> [58] "targettrafoscalerange" "textvectorizer"        "threshold"
#> [...]
```

PIPEOPS SO FAR AND PLANNED

- Simple data preprocessing operations (scaling, Box Cox, Yeo Johnson, PCA, ICA)
- Missing value imputation (sampling, mean, median, mode, new level, ...)
- Feature selection (by name, by type, using filter methods)
- Categorical data encoding (one-hot, treatment, impact)
- Sampling (subsampling for speed, sampling for class balance)
- Ensemble methods on Predictions (weighted average, possibly learned weights)
- Branching (simultaneous branching, alternative branching)
- Combination of data: `featureunion`
- Text processing
- Date processing
- Time series and spatio-temporal data *(planned)*
- Multi-output and ordinal targets *(planned)*
- Outlier detection *(planned)*

THE STRUCTURE

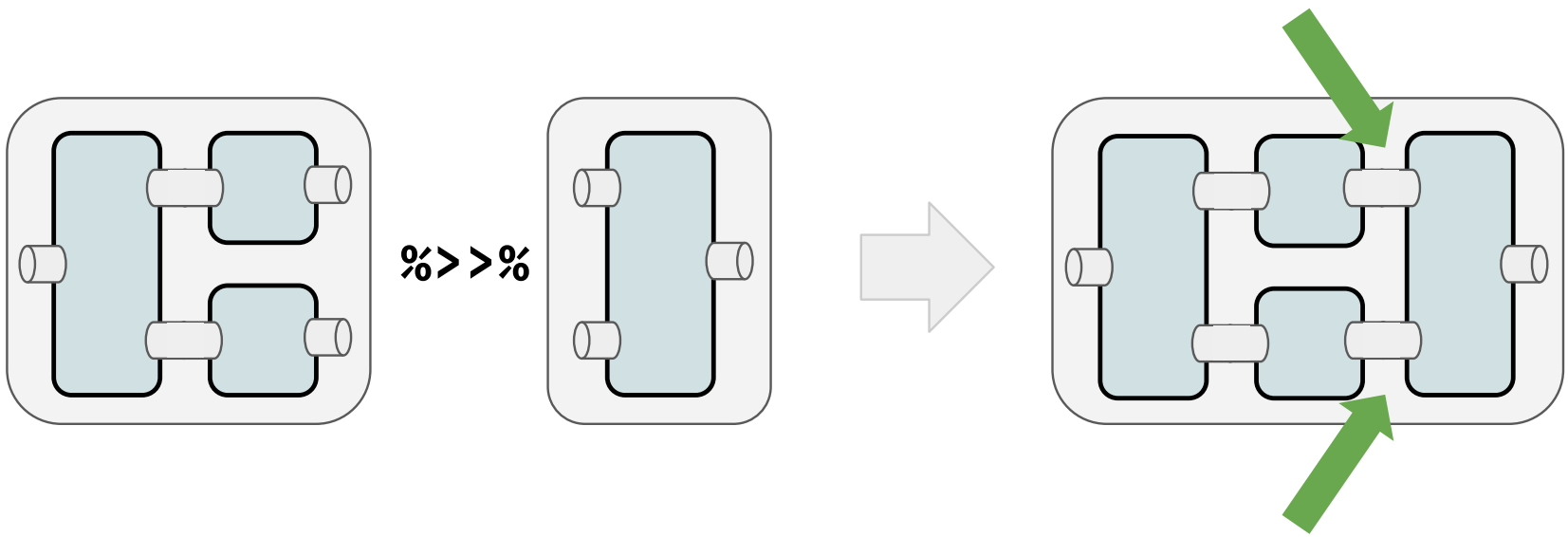
Graph Operations



THE STRUCTURE

Graph Operations

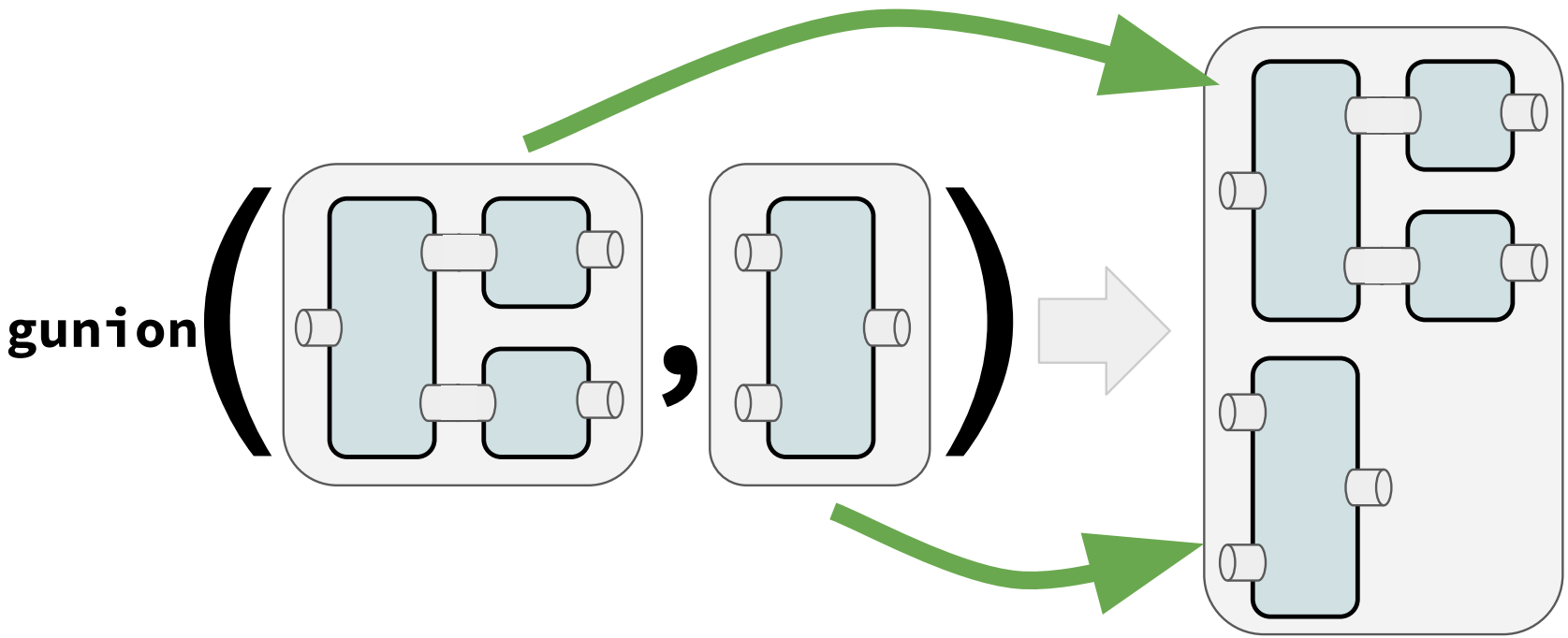
- The %>>%-operator concatenates `Graphs` and `PipeOps`



THE STRUCTURE

Graph Operations

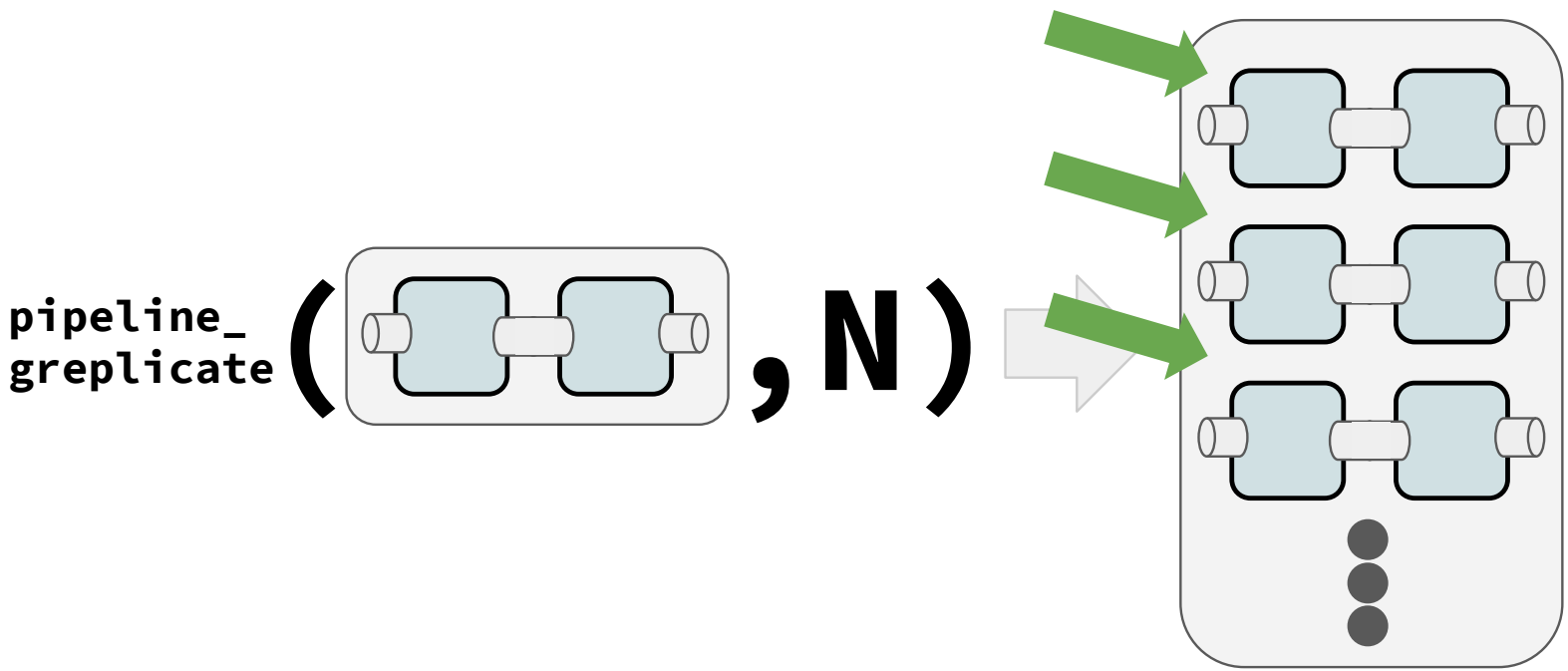
- The %>>%-operator concatenates `Graphs` and `PipeOps`
- The `gunion()`-function unites `Graphs` and `PipeOps`



THE STRUCTURE

Graph Operations

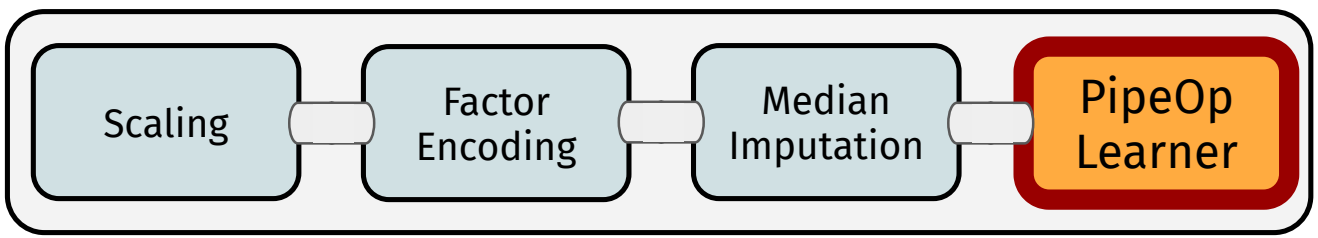
- The %>>%-operator concatenates `Graphs` and `PipeOps`
- The `gunion()`-function unites `Graphs` and `PipeOps`
- The `pipeline_greplicate()`-function unites copies of `Graphs` and `PipeOps`



LEARNERS AND GRAPHS

PipeOpLearner

- `Learner` as a `PipeOp`
- Fits a model, output is `Prediction`



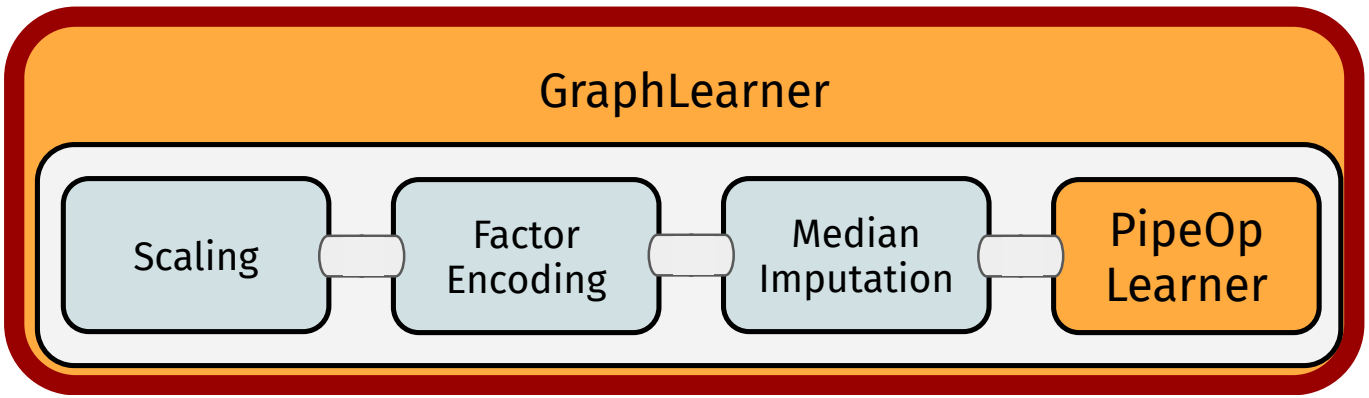
LEARNERS AND GRAPHS

PipeOpLearner

- `Learner` as a `PipeOp`
- Fits a model, output is `Prediction`

GraphLearner

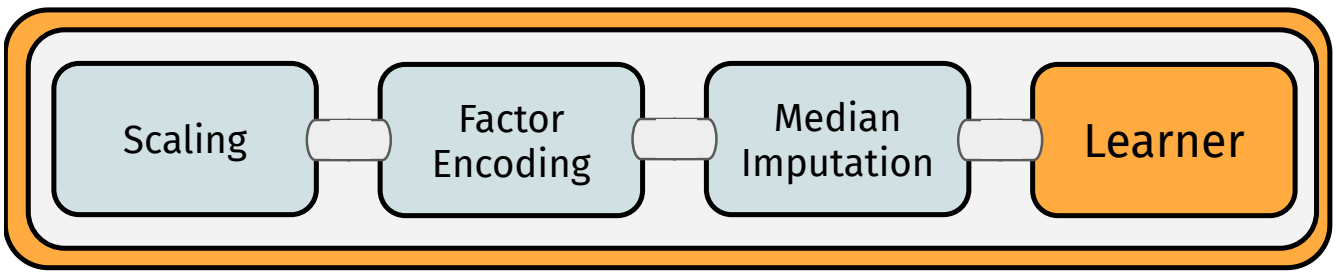
- `Graph` as a `Learner`
- All benefits of `mlr3`: **resampling**, **tuning**, **nested resampling**, . . .



MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

```
graph = po("scale") %>>%
  po("encode") %>>%
  po("imputemedian") %>>%
  lrn("classif.rpart")
```

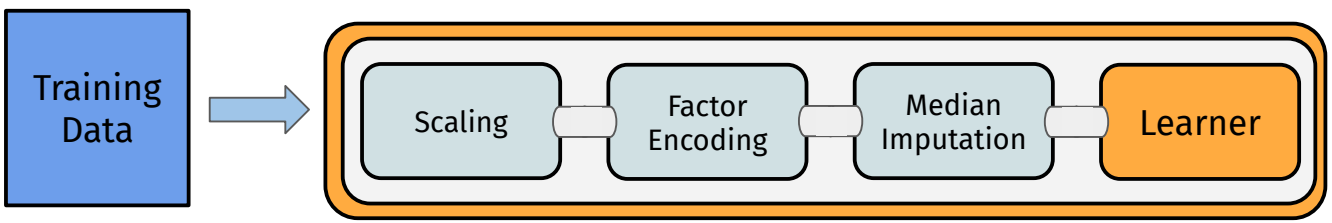


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = as_learner(graph) # or: GraphLearner$new(graph)
glrn$train(task)
```

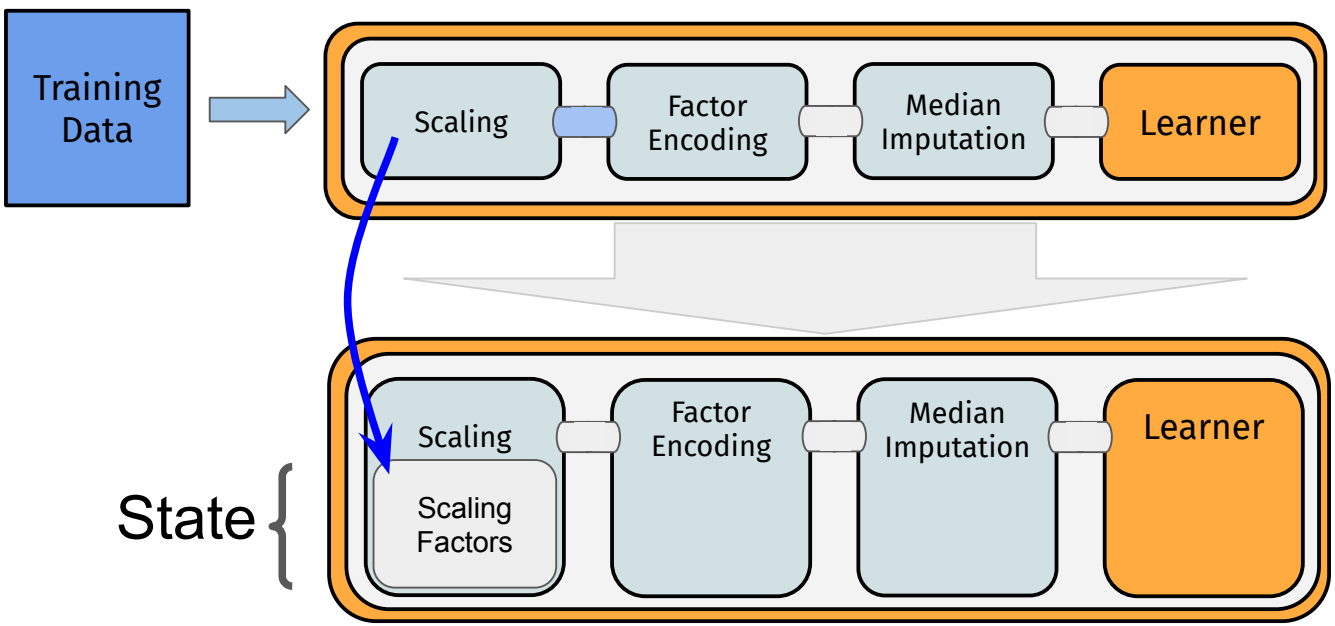


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = as_learner(graph) # or: GraphLearner$new(graph)
glrn$train(task)
```

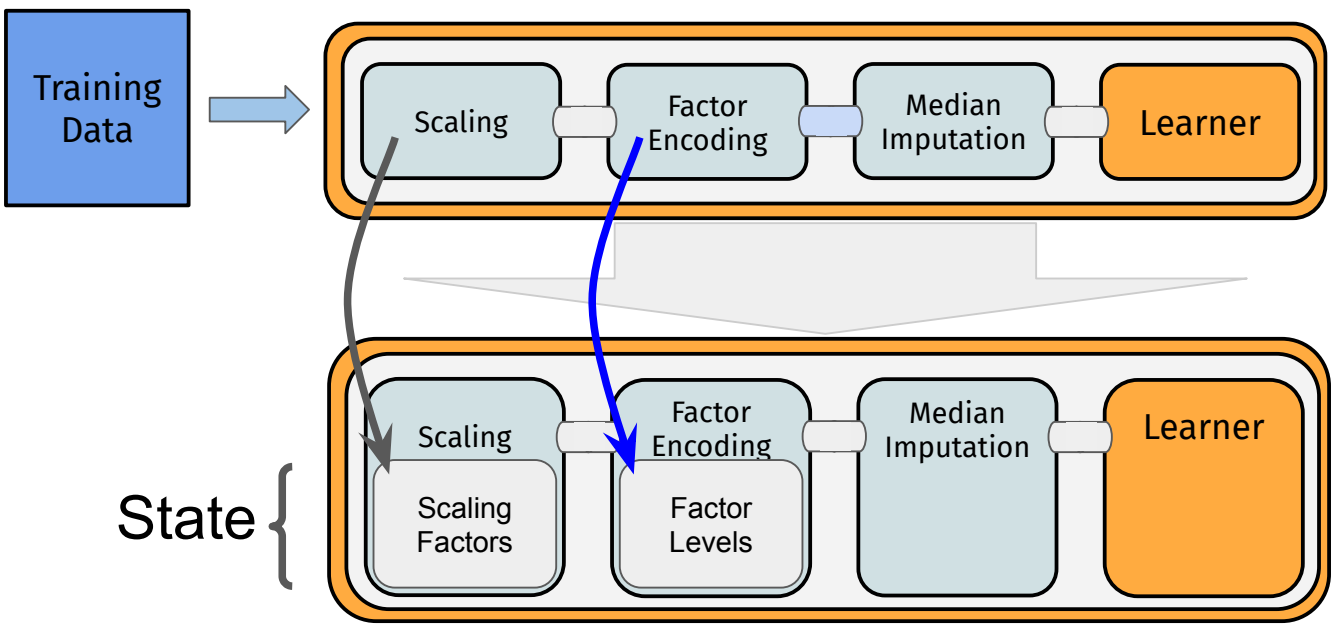


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = as_learner(graph) # or: GraphLearner$new(graph)
glrn$train(task)
```

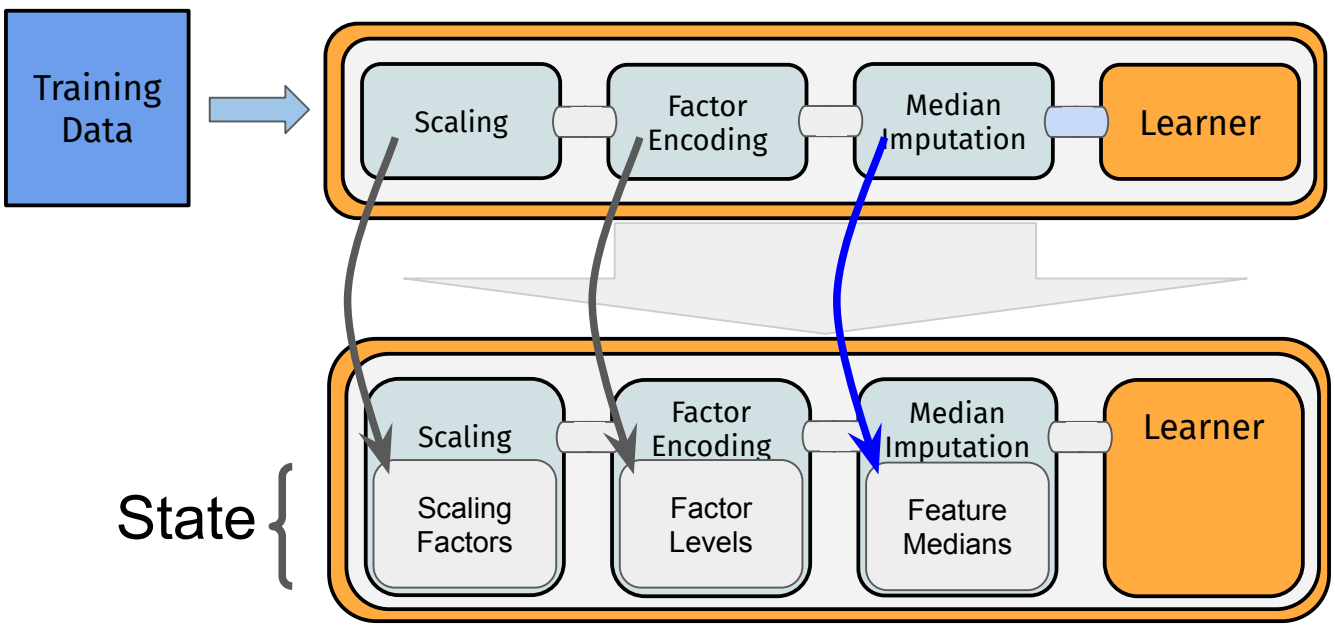


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = as_learner(graph) # or: GraphLearner$new(graph)
glrn$train(task)
```

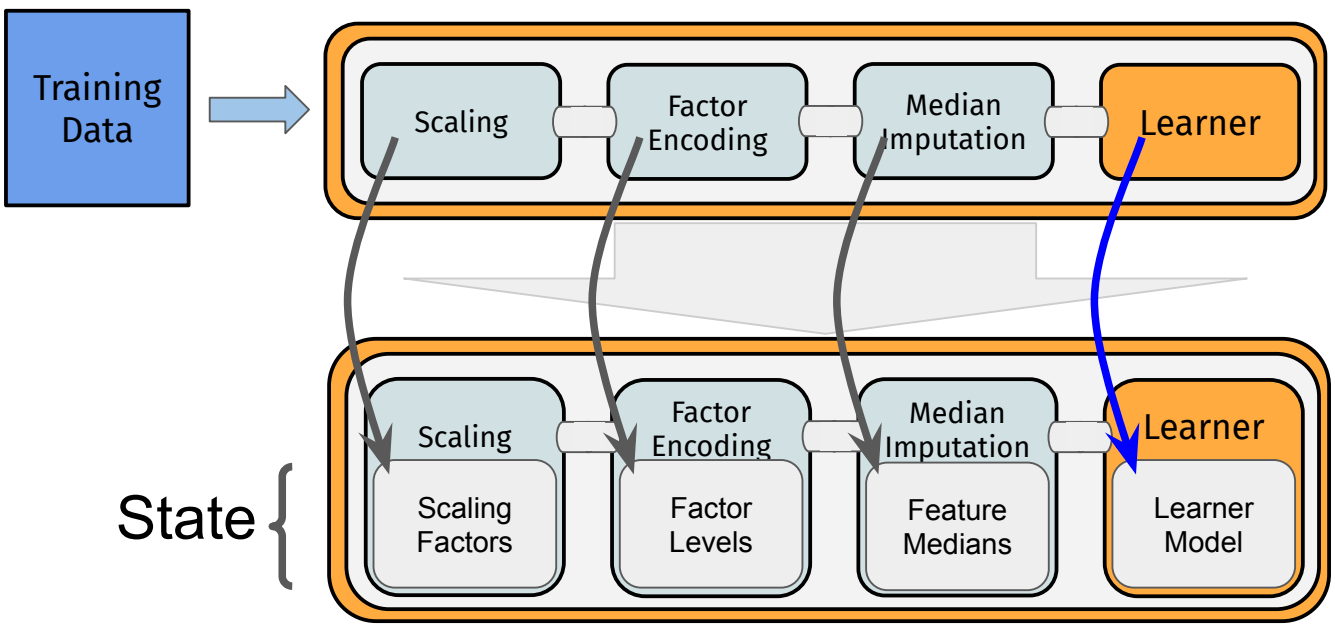


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = as_learner(graph) # or: GraphLearner$new(graph)
glrn$train(task)
```

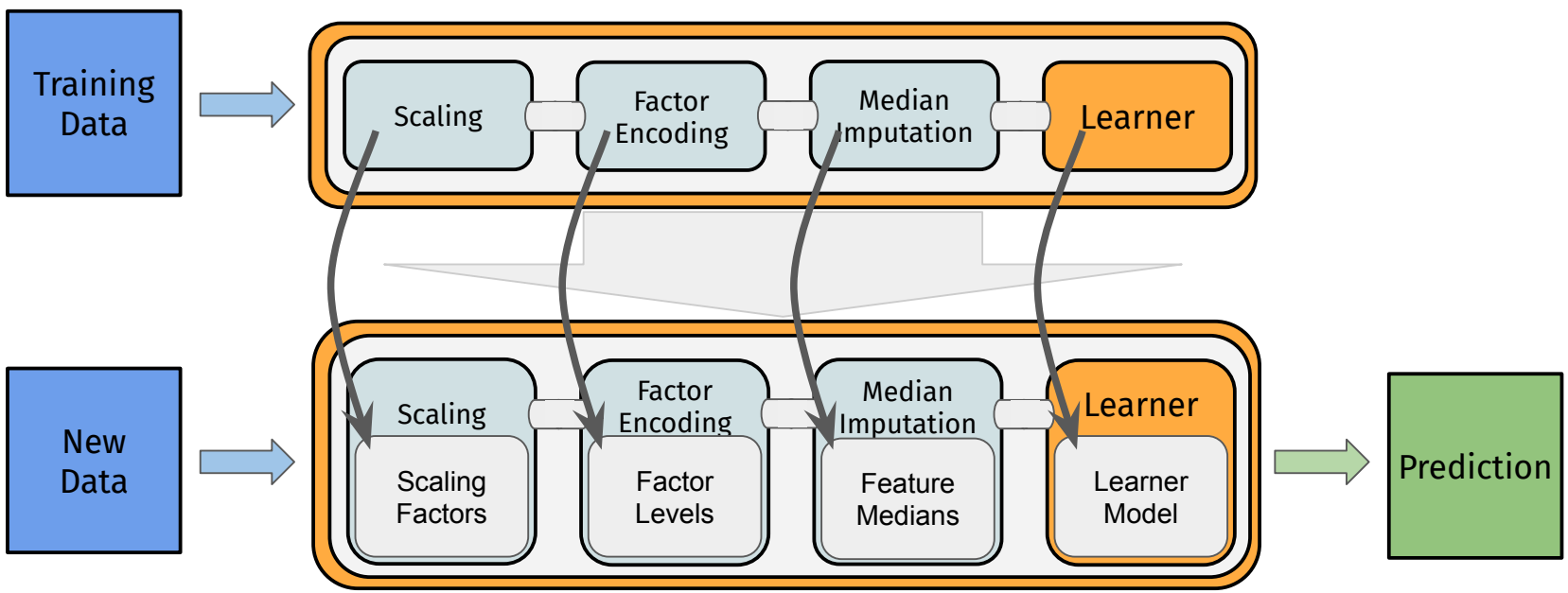


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`
- `predict()`tion: Data propagates, uses `$states`

```
glrn$predict(task)
```



MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph$pipeops$scale$param_set$values$center = FALSE
```

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual `PipeOps` (*after* `$train()`)

```
graph$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          4.2          1.4          5.9          3.1
```

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph$pipeops$scale$param_set$values$center = FALSE
```

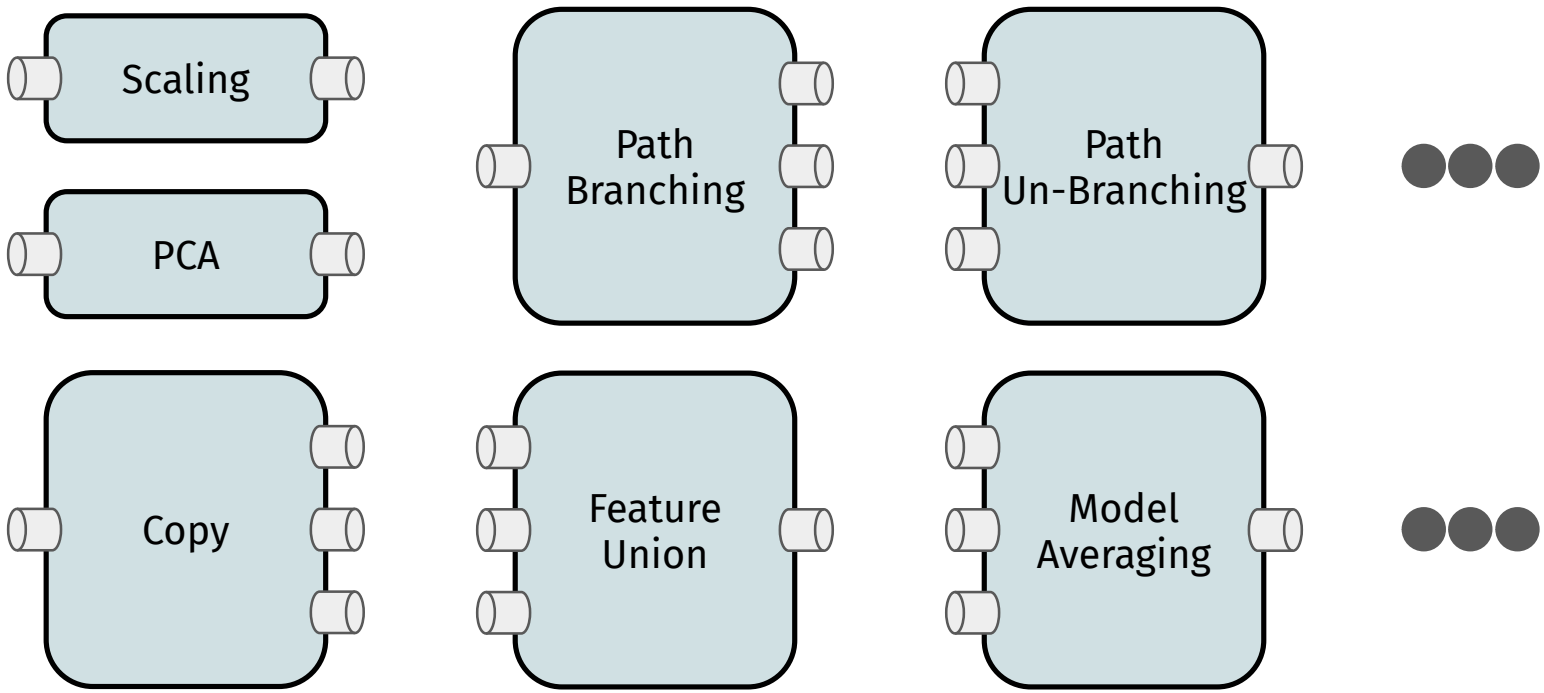
- Retrieving state: `$state` of individual `PipeOps` (*after* `$train()`)

```
graph$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          4.2          1.4          5.9          3.1
```

- Retrieving intermediate results: `$.result` (set debug option before)

```
graph$pipeops$scale$.result[[1]]$head(3)
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         0.34        0.14         0.86        1.13
#> 2:  setosa         0.34        0.14         0.83        0.97
#> 3:  setosa         0.31        0.14         0.79        1.03
```

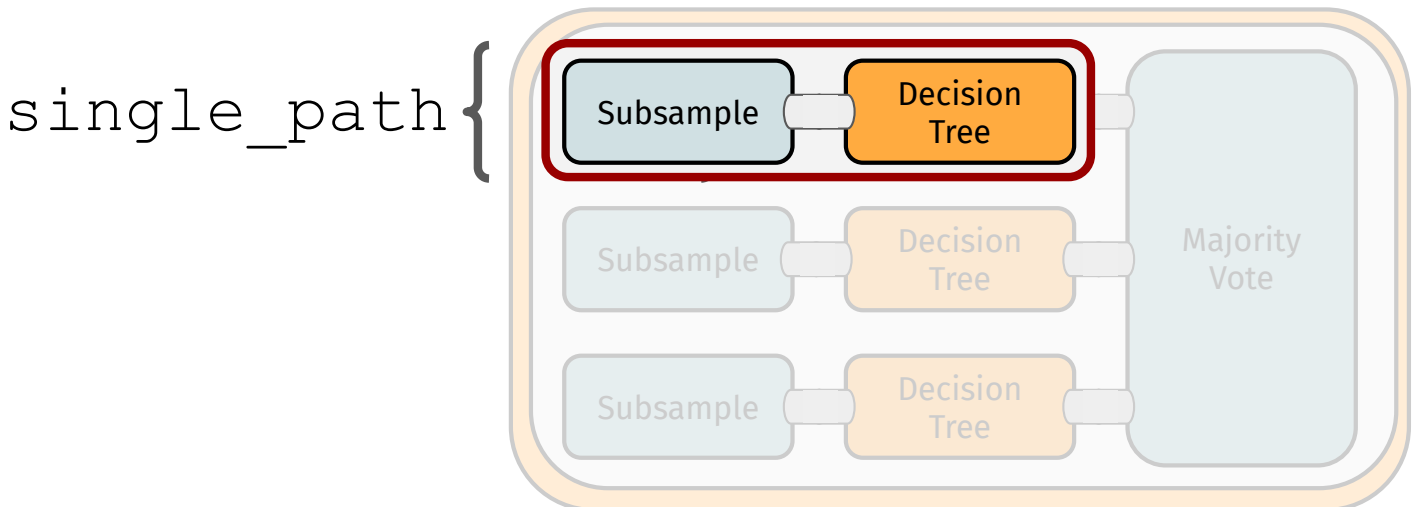
PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

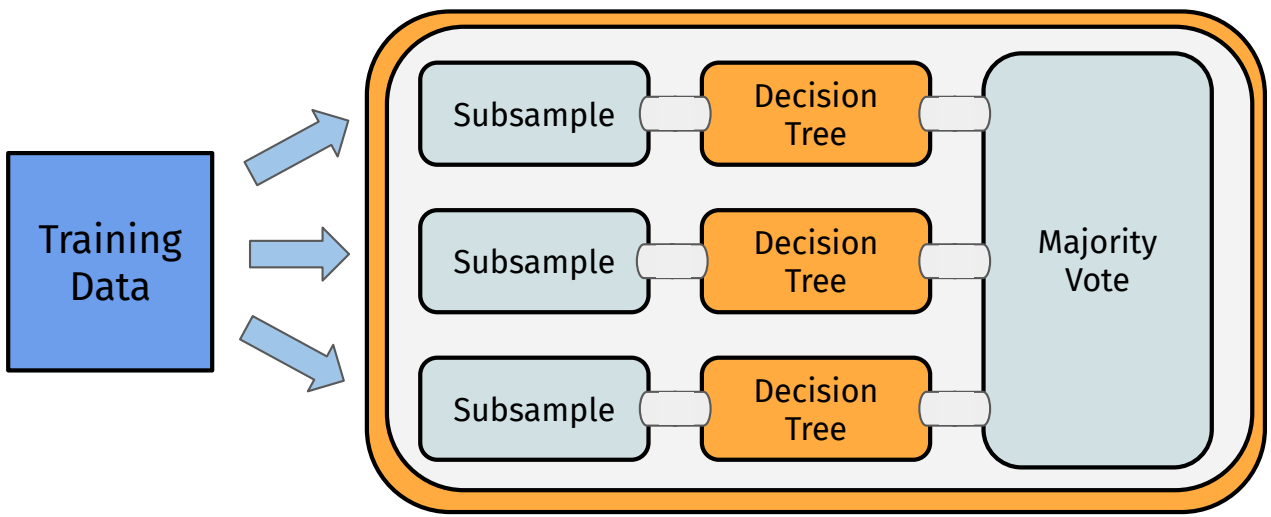
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

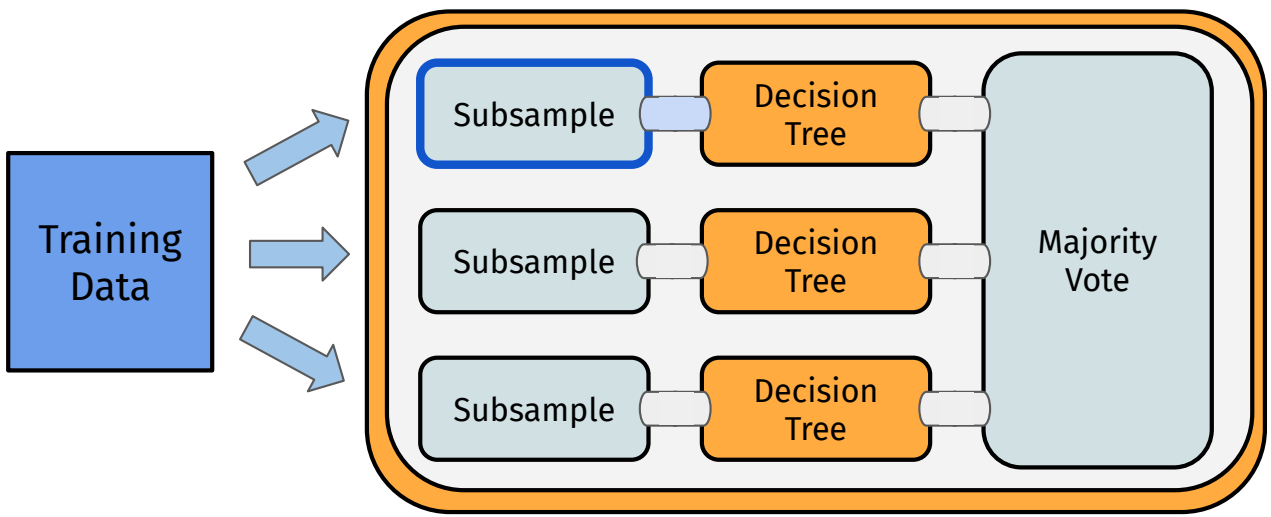
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

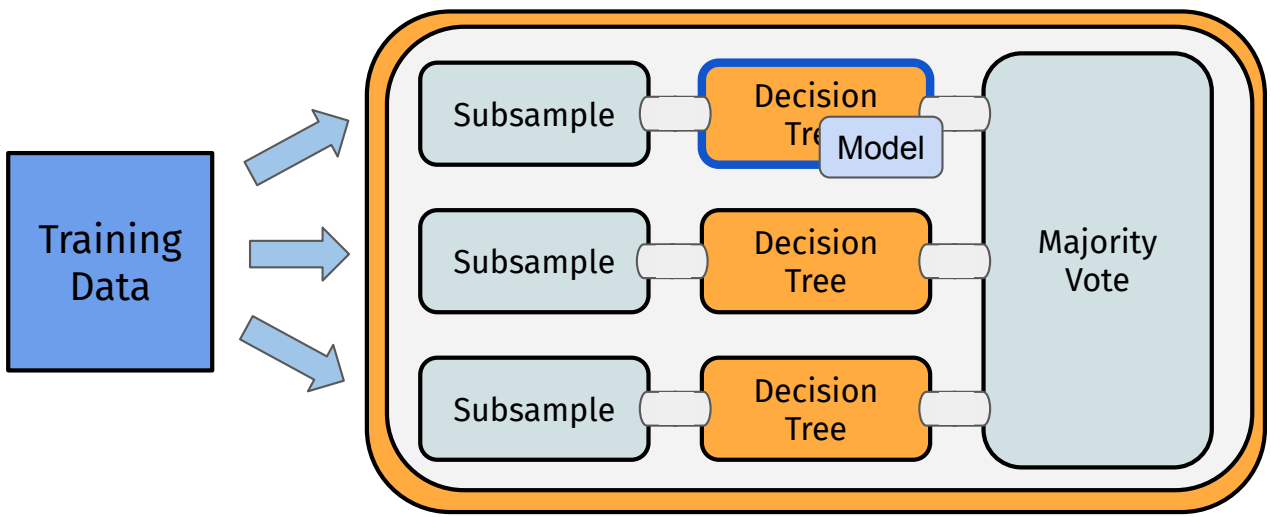
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

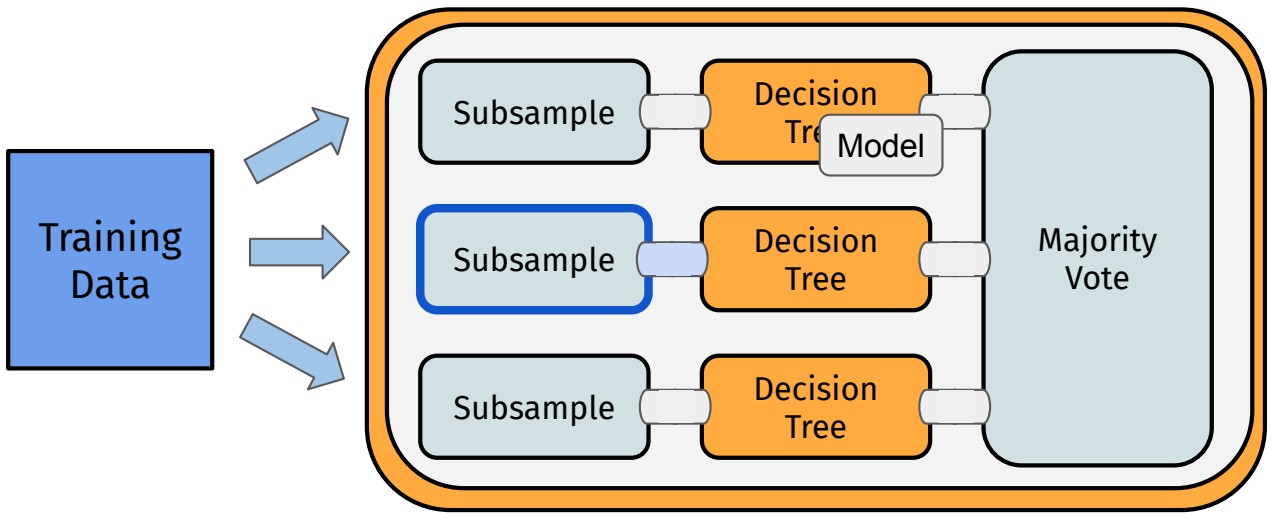
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

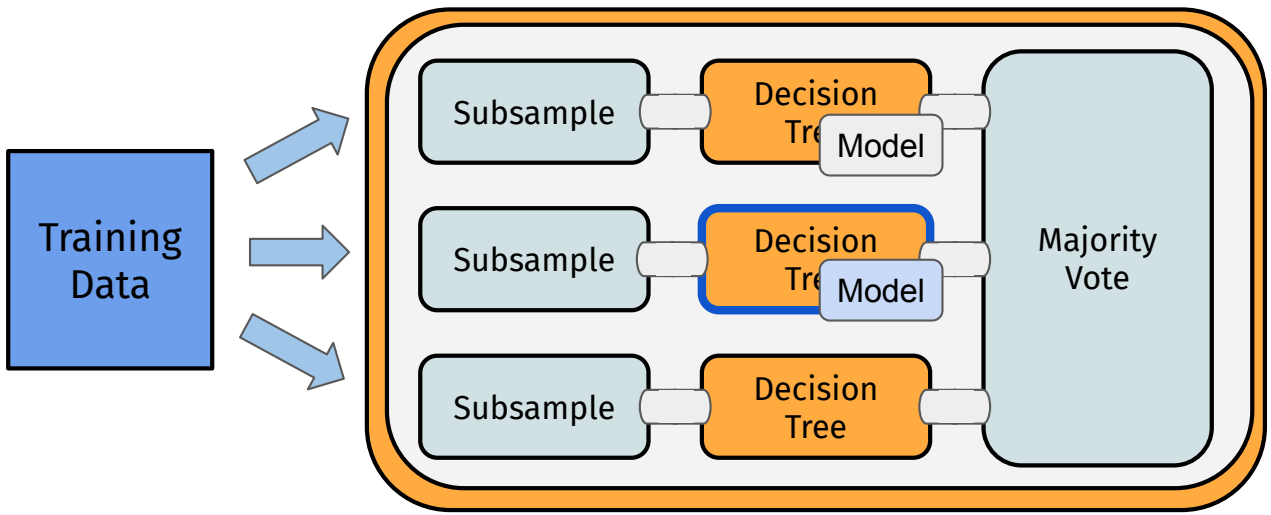
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

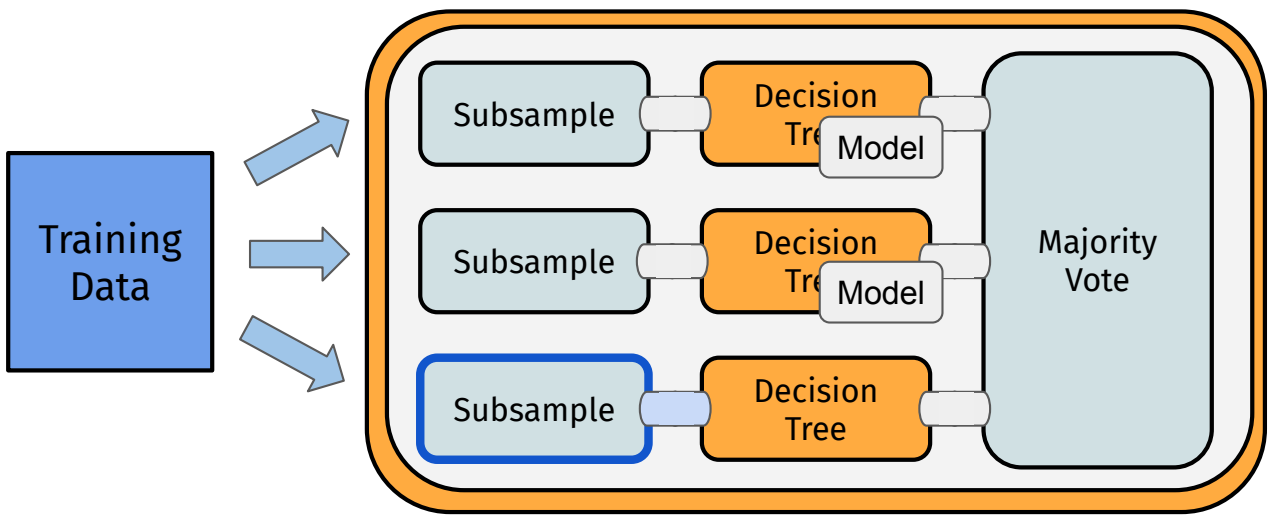
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

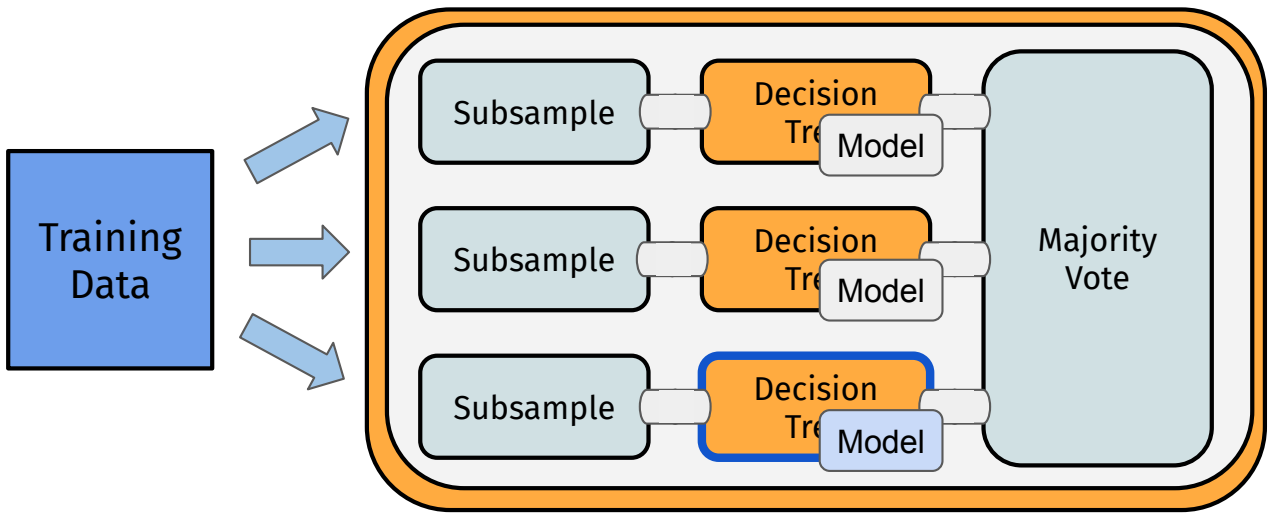
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

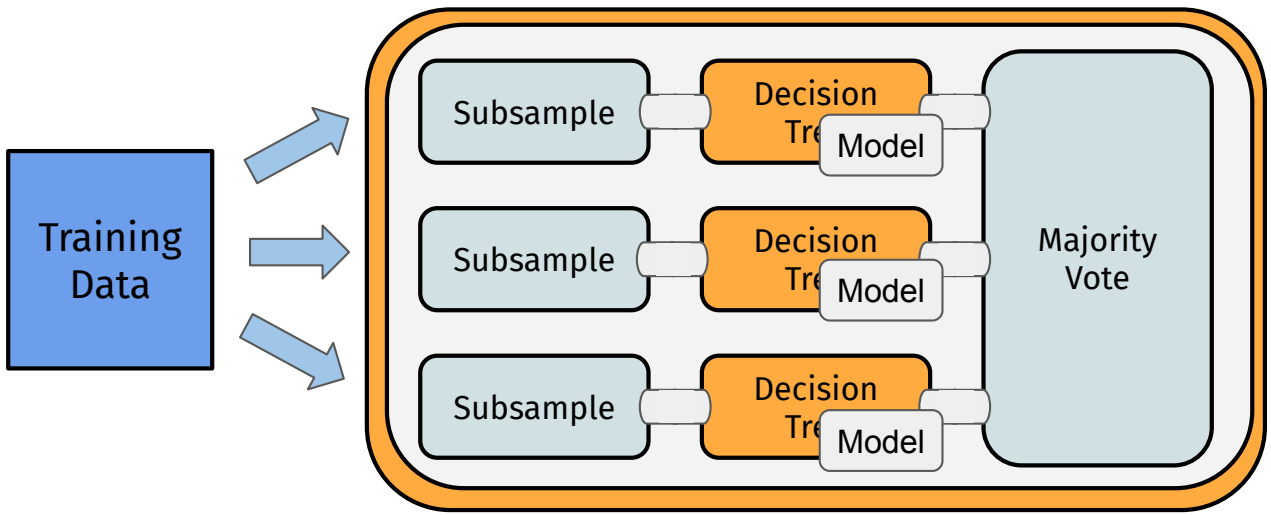
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

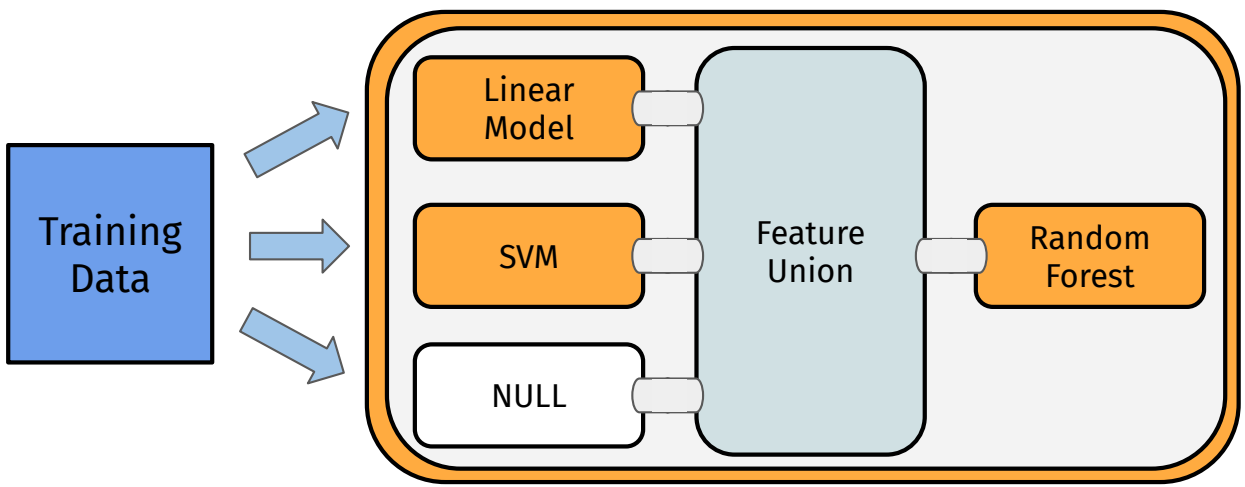
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Stacking

```
graph_stack = gunion(list(
      po("learner_cv", learner = lrn("regr.lm")),
      po("learner_cv", learner = lrn("regr.svm")),
      po("nop"))) %>>%
  po("featureunion") %>>%
  lrn("regr.ranger")
```

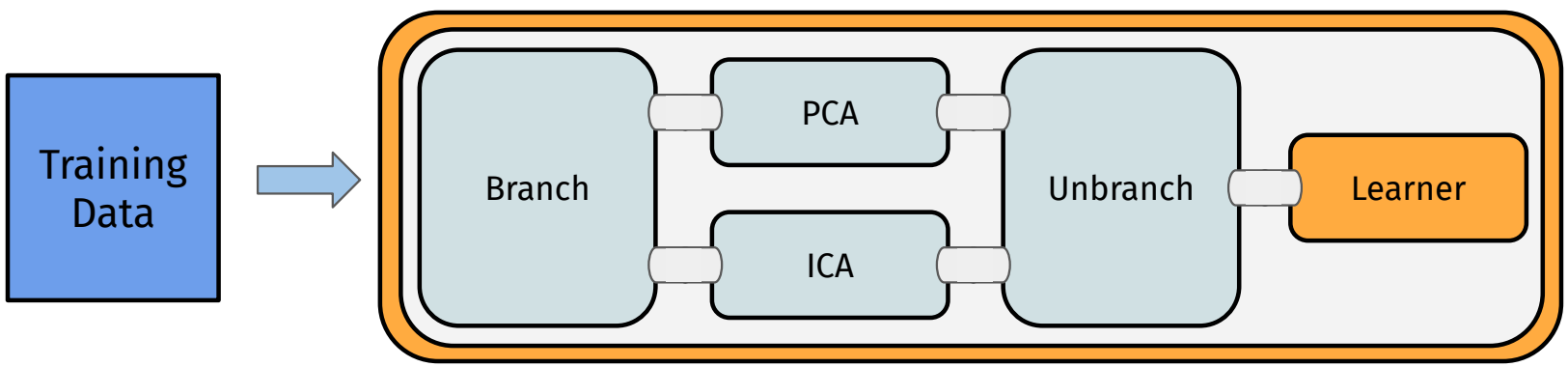


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(
  pca = po("pca"),
  ica = po("ica"))) %>>%
  lrn("classif.kknn")
```

Execute only one of several alternative paths

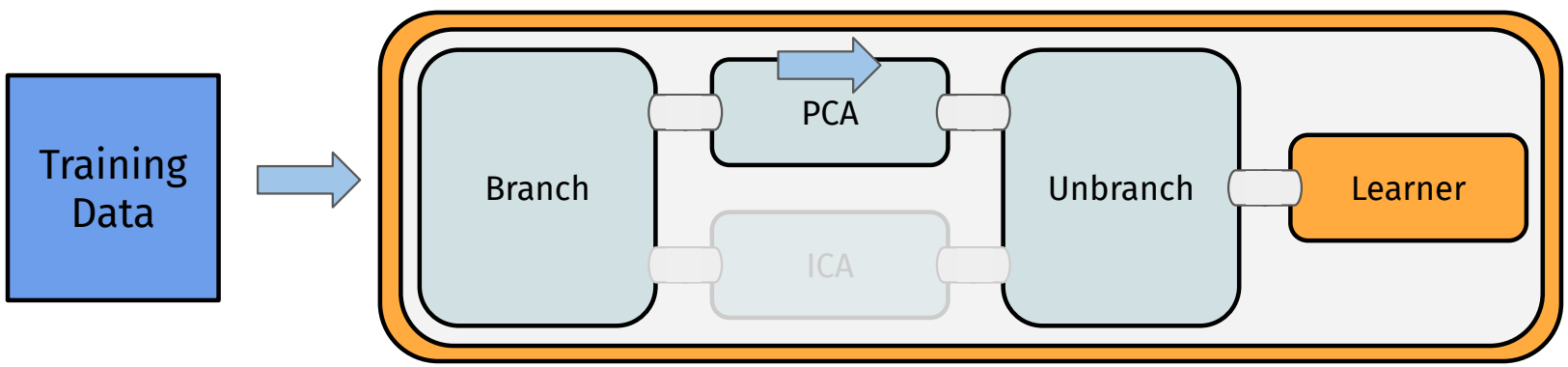


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(
  pca = po("pca"),
  ica = po("ica"))) %>>%
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$
    param_set$values$selection = "pca"
```

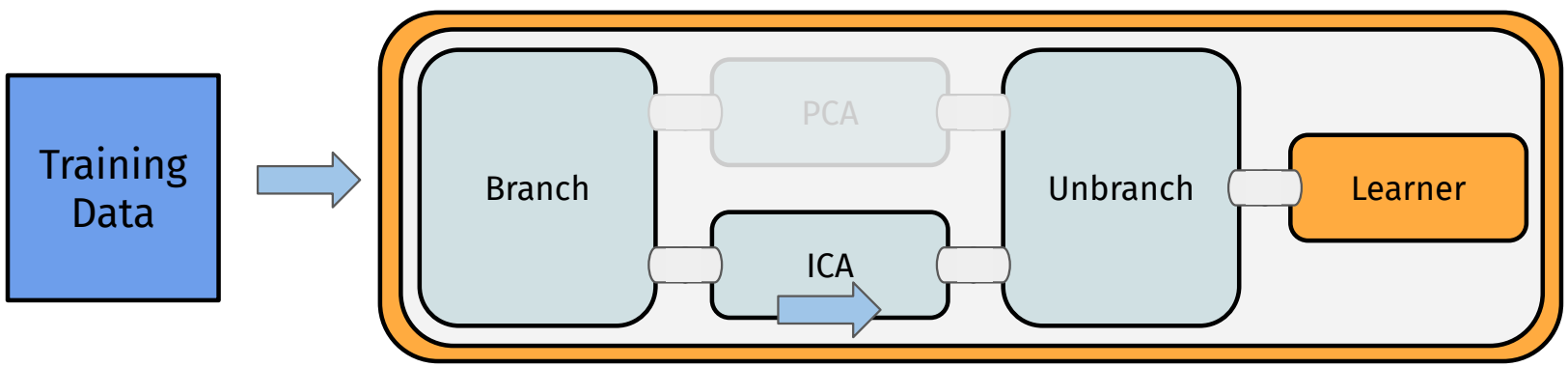


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(
  pca = po("pca"),
  ica = po("ica"))) %>>%
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$
    param_set$values$selection = "ica"
```



TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Suppose we only want PCA on some columns of our data:

```
task$data(1:9)
```

```
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa          1.4         0.2          5.1         3.5
#> 2:  setosa          1.4         0.2          4.9         3.0
#> 3:  setosa          1.3         0.2          4.7         3.2
#> 4:  setosa          1.5         0.2          4.6         3.1
#> 5:  setosa          1.4         0.2          5.0         3.6
#> 6:  setosa          1.7         0.4          5.4         3.9
#> 7:  setosa          1.4         0.3          4.6         3.4
#> 8:  setosa          1.5         0.2          5.0         3.4
#> 9:  setosa          1.4         0.2          4.4         2.9
```

TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Using `affect_columns`:

```
sel = selector_grep("^Sepal")

partial_pca = po("pca", affect_columns = sel)

result = partial_pca$train(list(task))

result[[1]]$data(1:3)

#>    Species   PC1    PC2 Petal.Length Petal.Width
#>     <fctr> <num>  <num>        <num>       <num>
#> 1:  setosa -0.78  0.378          1.4         0.2
#> 2:  setosa -0.94 -0.137          1.4         0.2
#> 3:  setosa -1.15  0.045          1.3         0.2
```

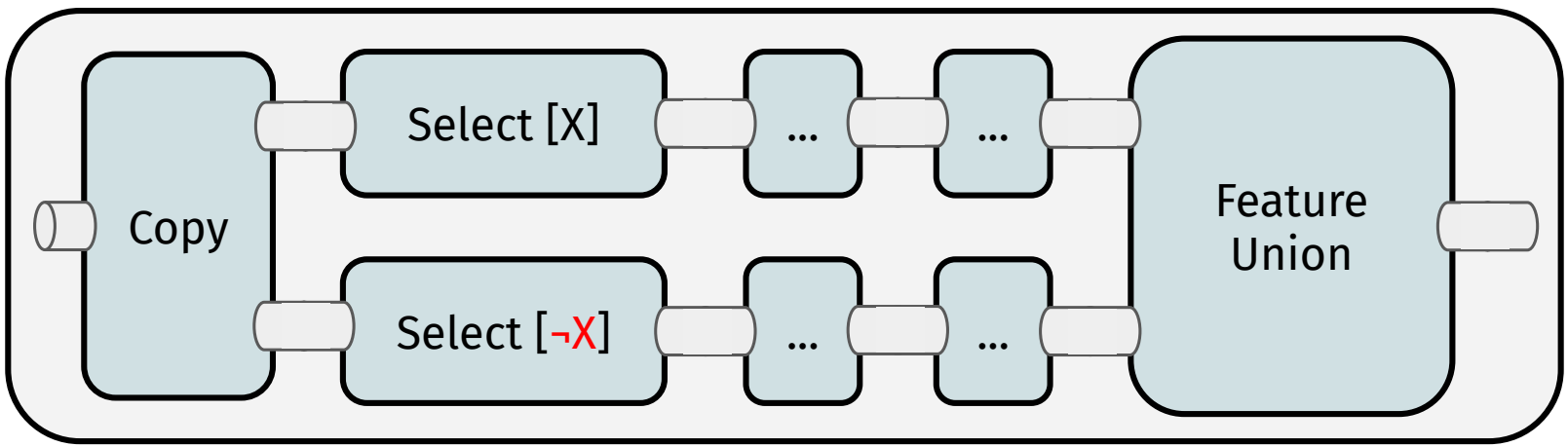
TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Using `po("select")`:



TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Using `po("select")`:

```
sel = selector_grep("^Sepal")
selcomp = selector_invert(sel)

partial_pca = gunion(list(
  po("select", selector = sel) %>>% po("pca"),
  po("select", selector = selcomp, id = "select2"))) %>>%
  po("featureunion")

partial_pca$train(task)[[1]]$data(1:3)
```

```
#>    Species   PC1    PC2 Petal.Length Petal.Width
#>     <fctr> <num>  <num>        <num>       <num>
#> 1:  setosa -0.78  0.378          1.4         0.2
#> 2:  setosa -0.94 -0.137          1.4         0.2
#> 3:  setosa -1.15  0.045          1.3         0.2
```

“PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + . . .)
- Bagging
- Branching

"PIPELINES" DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

Collection of these is in mlr3pipelines

```
head(as.data.table(mlr_pipeops), 5)[, list(key, input.num, output.num)]

#> Key: <key>
#>                key input.num output.num
#>             <char>     <int>      <int>
#> 1:         boxcox         1          1
#> 2:         branch         1         NA
#> 3:          chunk         1         NA
#> 4: classbalancing         1          1
#> 5:     classifavg        NA          1
```

"PIPELINES" DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

Collection of these is in mlr3pipelines

`po()` accesses the `mlr_pipeops` "Dictionary".

```
pca = po("pca")
pca

#> PipeOp: <pca> (not trained)
#> values: <list()>
#> Input channels <name [train type, predict type]>:
#>   input [Task,Task]
#> Output channels <name [train type, predict type]>:
#>   output [Task,Task]
```

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  1. *what learner* to use,
  2. *what preprocessing* to use, and
  3. *what hyperparameters* to use.

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  1. *what learner* to use,
  2. *what preprocessing* to use, and
  3. *what hyperparameters* to use.
- (1) and (2) are decisions about *graph structure* in `mlr3pipelines`

AUTOML <3 PIPELINES

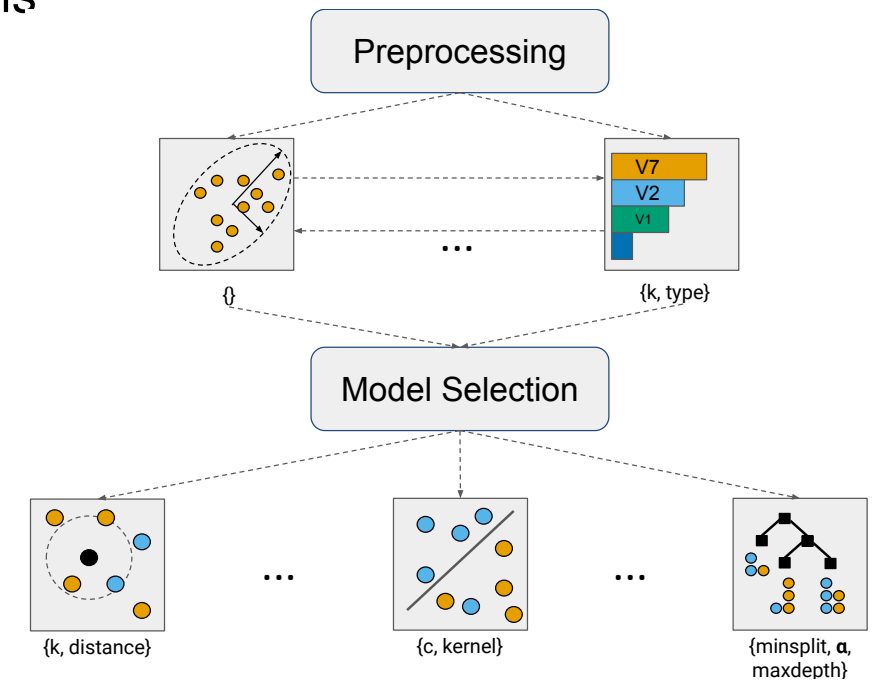
- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  1. *what learner* to use,
  2. *what preprocessing* to use, and
  3. *what hyperparameters* to use.
- (1) and (2) are decisions about *graph structure* in `mlr3pipelines`

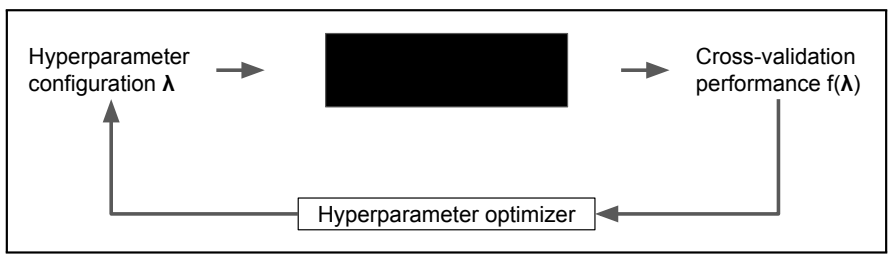
$\Rightarrow$ The problem reduces to **pipelines + parameter tuning**

AUTOML WITH MLR3PIPELINES

AutoML in a Nutshell

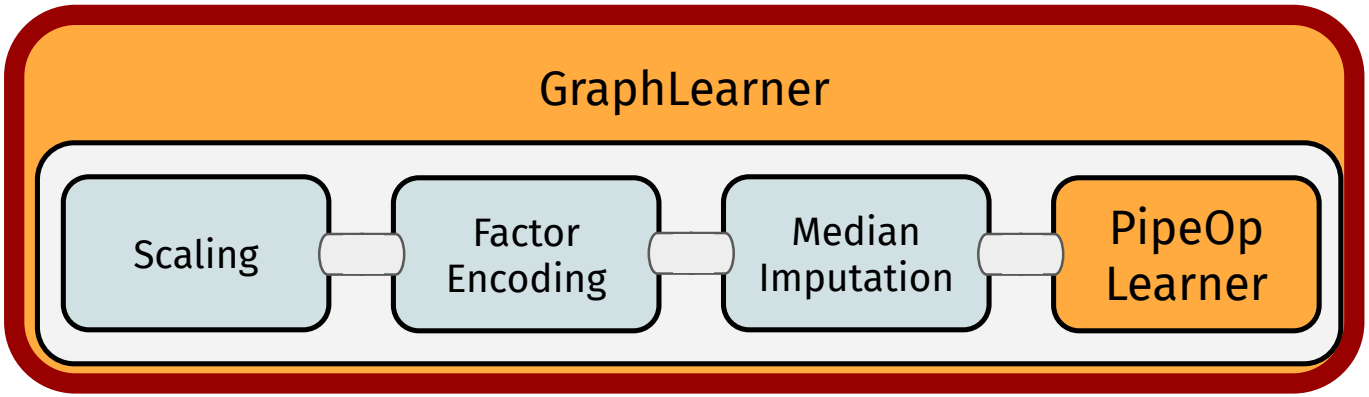
- Preprocessing steps
- ML Algorithms
- Tuner





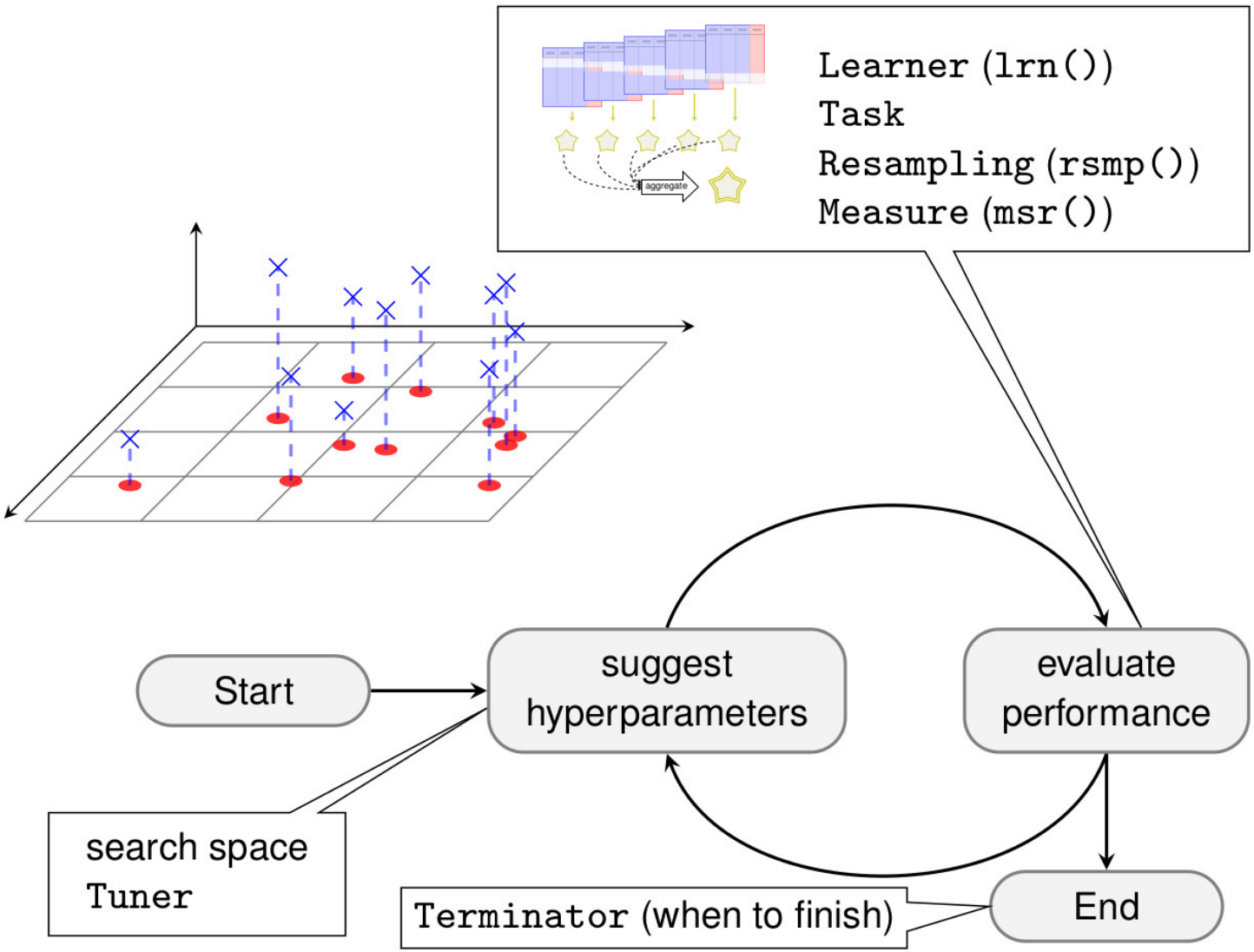
GRAPHLEARNER

- `Graph` as a `Learner`
- All benefits of `mlr3`: **resampling**, **tuning**, **nested resampling**, ...




```
graph = po("scale") %>>% po("encode") %>>%
  po("imputemedian") %>>% lrn("classif.rpart")
glrn = as_learner(graph)
glrn$train(task)
glrn$predict(task)
resample(task, glrn, rsmp("cv", folds = 3))
```

TUNING

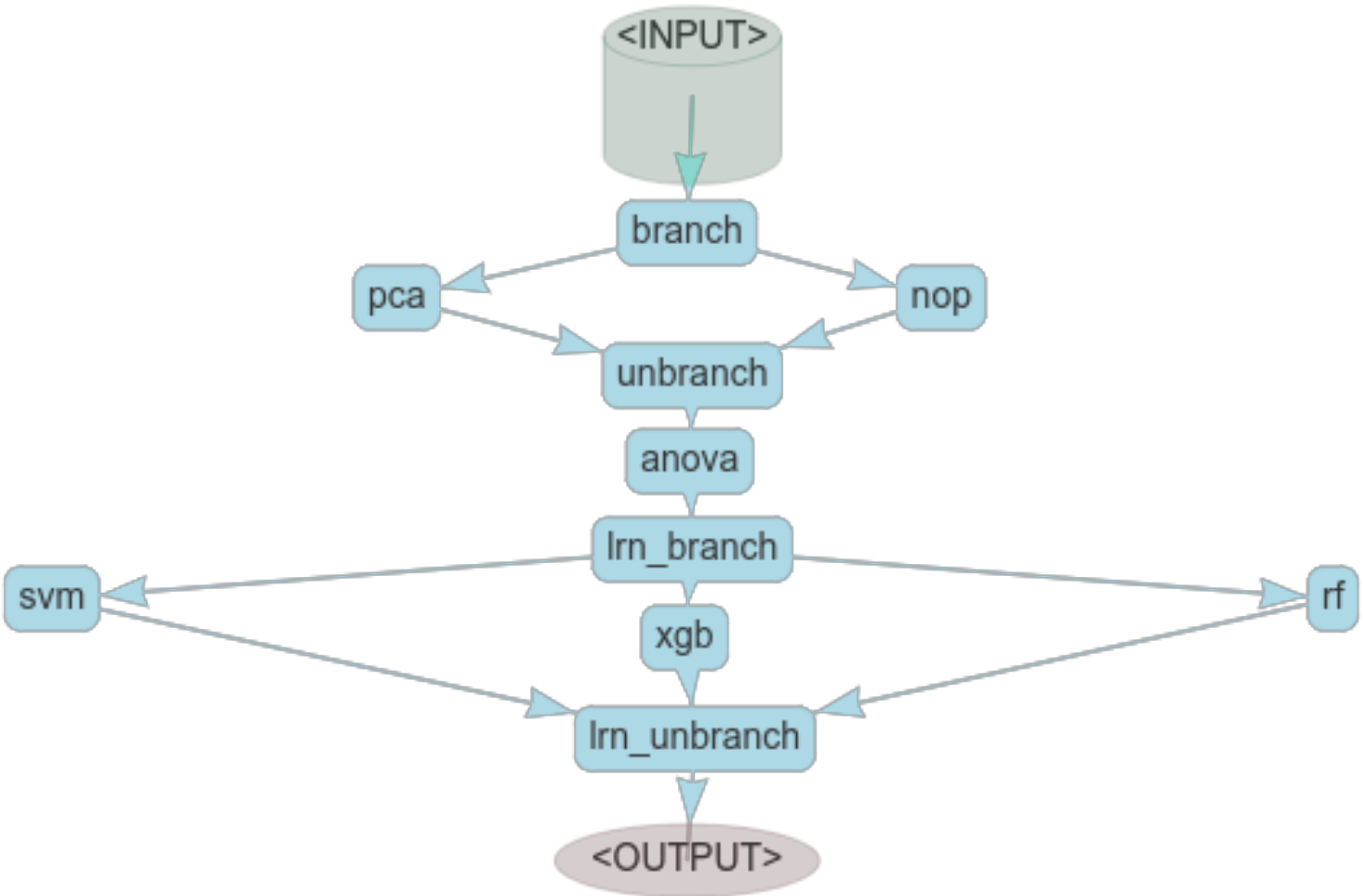

Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
aggregate
Start
suggest hyperparameters
evaluate performance
search space
Tuner
Terminator (when to finish)
End

PIPELINES TUNING

- Works **exactly** as in basic `mlr3` / `mlr3tuning`
- `PipeOps` have *hyperparameters* (using `paradox` pkg)
- `Graphs` have hyperparameters of all components *combined*
- $\Rightarrow$ Joint **tuning** and nested CV of complete graph

```
p1 = ppl("branch", list(
  "pca" = po("pca"),
  "nothing" = po("nop")
))
p2 = flt("anova")
p3 = ppl("branch", list(
  "svm" = lrn("classif.svm", id = "svm", kernel = "radial",
    type = "C-classification"),
  "xgb" = lrn("classif.xgboost", id = "xgb"),
  "rf" = lrn("classif.ranger",  id = "rf")
), prefix_branchops = "lrn_")
gr = p1 %>>% p2 %>>% p3
glrn = as_learner(gr)
```

PIPELINES TUNING

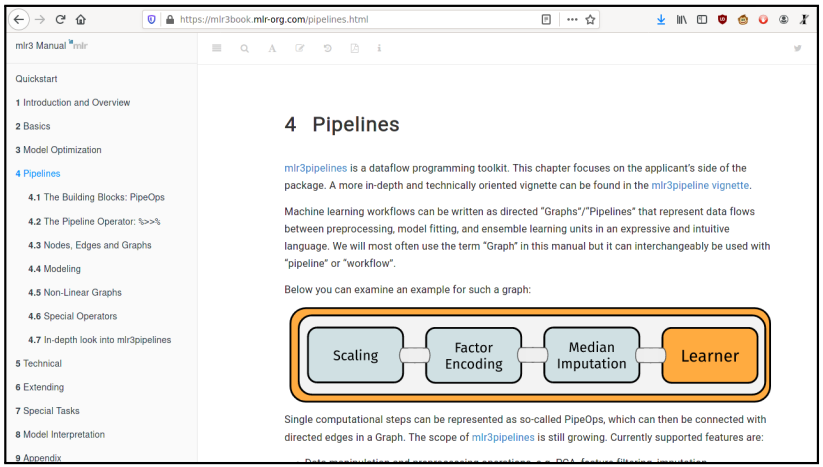


PIPELINES TUNING

```
ps = ps(
  branch.selection = p_fct(levels = c("pca", "nothing")),
  anova.filter.frac = p_dbl(lower = 0.1, upper = 1),
  lrn_branch.selection = p_fct(levels = c("svm", "xgb", "rf")),
  rf.mtry.ratio = p_int(lower = 1L, upper = 20L, trafo = function(x) 1/x,
    depends = lrn_branch.selection == "rf" ),
  xgb.nrounds = p_int(lower = 1, upper = 500,
    depends = lrn_branch.selection == "xgb"),
  svm.cost = p_dbl(lower = -12, upper = 4, trafo = function(x) 2^x,
    depends = lrn_branch.selection == "svm"),
  svm.gamma = p_dbl(lower = -12, upper = -1, trafo = function(x) 2^x,
    depends = lrn_branch.selection == "svm"))

inst = ti(task = tsk("sonar"), learner = glrn,
  resampling = rsmp("cv", folds = 3), measures =  msr("classif.ce"),
  terminator = trm("evals", n_evals = 10), search_space = ps)

gsearch = tnr("random_search")
gsearch$optimize(inst)
```

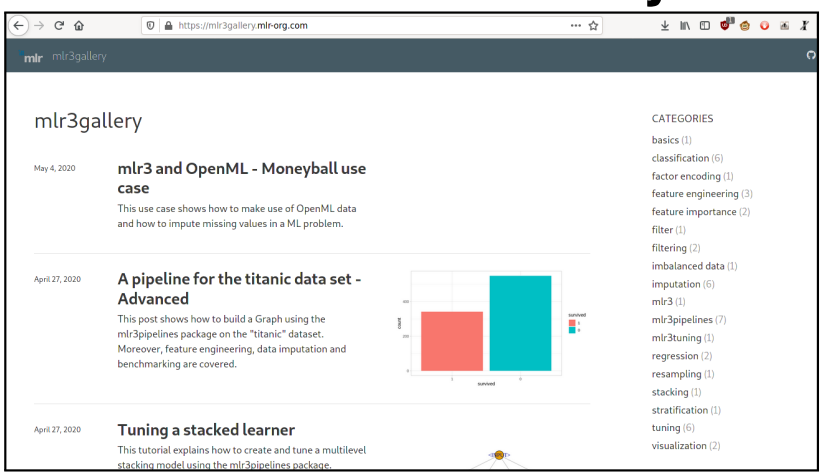
MLR3(PIPELINES) RESOURCES

mlr3 book



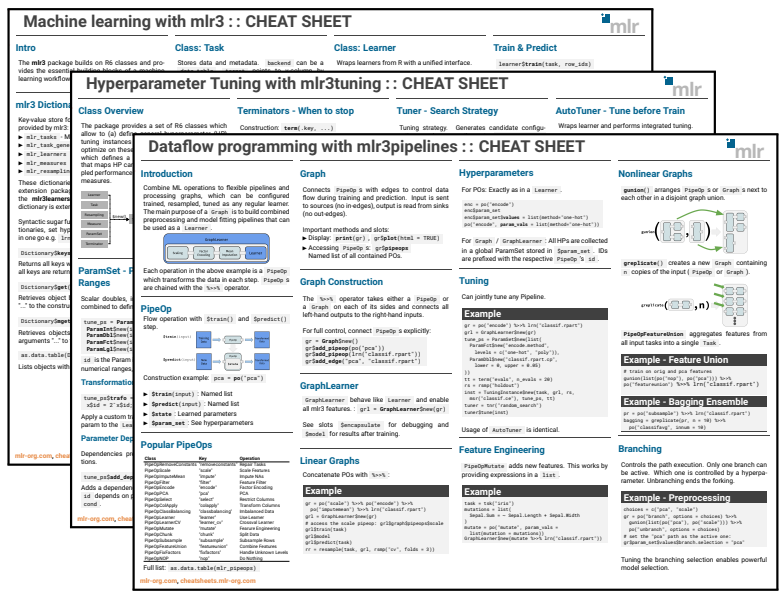
https://mlr3book.mlr-org.com/

mlr3 Use Case "Gallery"



https://mlr3gallery.mlr-org.com/

"cheat sheets"



https://cheatsheets.mlr-org.com/

OUTLOOK

What is to come?

- `mlr3pipelines`: caching, parallelization
- Better **tuners**: Bayesian Optimization, Hyperband
- Survival and Forecasting (via `mlr3proba`, `mlr3forecast`)
- Deep Learning (via `mlr3keras`)

Thanks! Please ask questions!

MLR3PIPELINES

`mlr3pipelines` overview:

- Construct a `PipeOp` using `po()`
- Use `Graph` operators to connect them
  - %>>%—chain operations
  - `gunion()`—put operations in parallel
  - `pipeline_greplicate()`—put many copies of an operation in parallel
- Train/predict with the `PipeOp` or `Graph` using `$train()`/`$predict()`
- Inspect the trained state through `$state`
- Encapsulate the `Graph` in a `GraphLearner` for resampling, benchmarking, and tuning