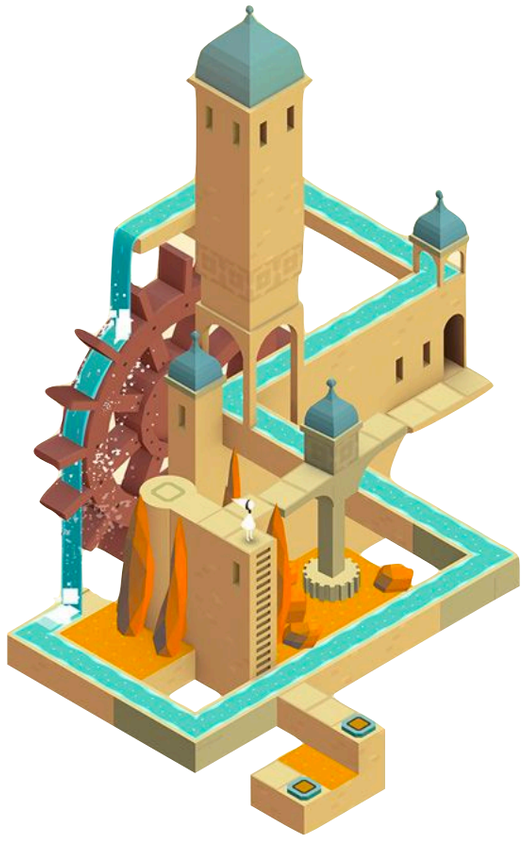
**Advanced Course on Data Science & Machine Learning**
**Siena, 2019**



# Automatic Machine Learning & Meta-Learning

part 2


Joaquin Vanschoren

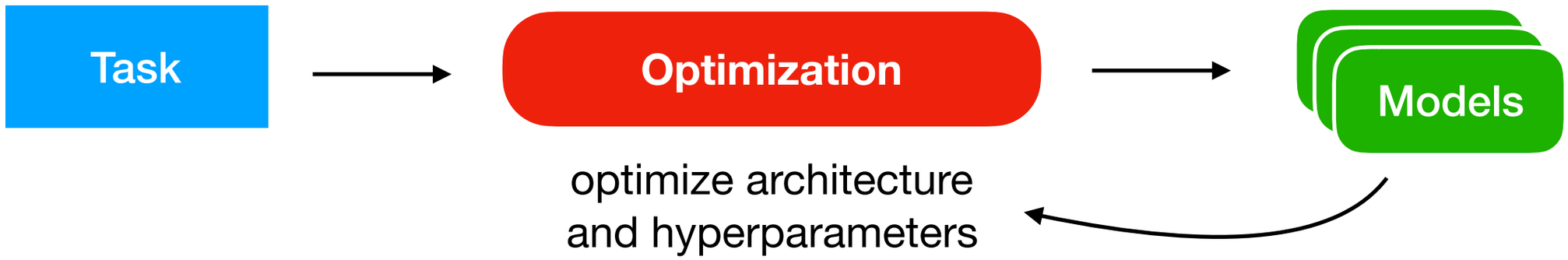
Eindhoven University of Technology

j.vanschoren@tue.nl @joavanschoren http://bit.ly/openml


slides!

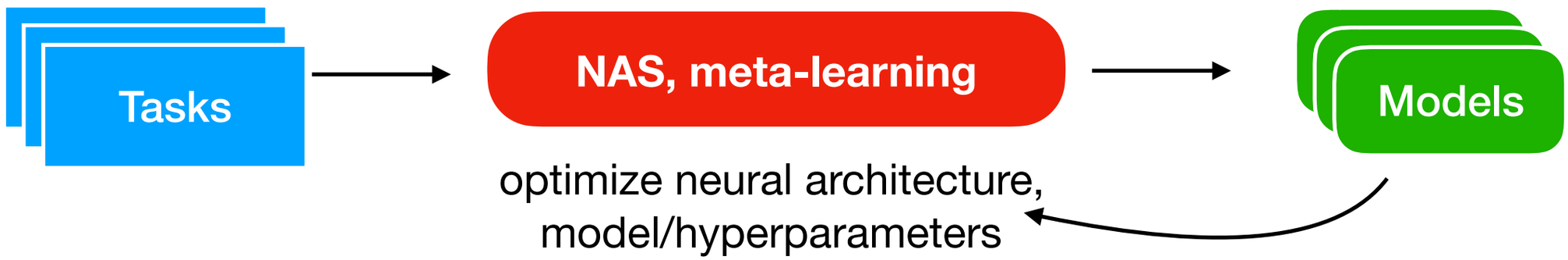
# Overview

part 1 AutoML introduction, promising optimization techniques




part 2 Meta-learning


part 3 Neural architecture search, meta-learning on neural nets



# Learning is a never-ending process

Humans don’t learn from scratch

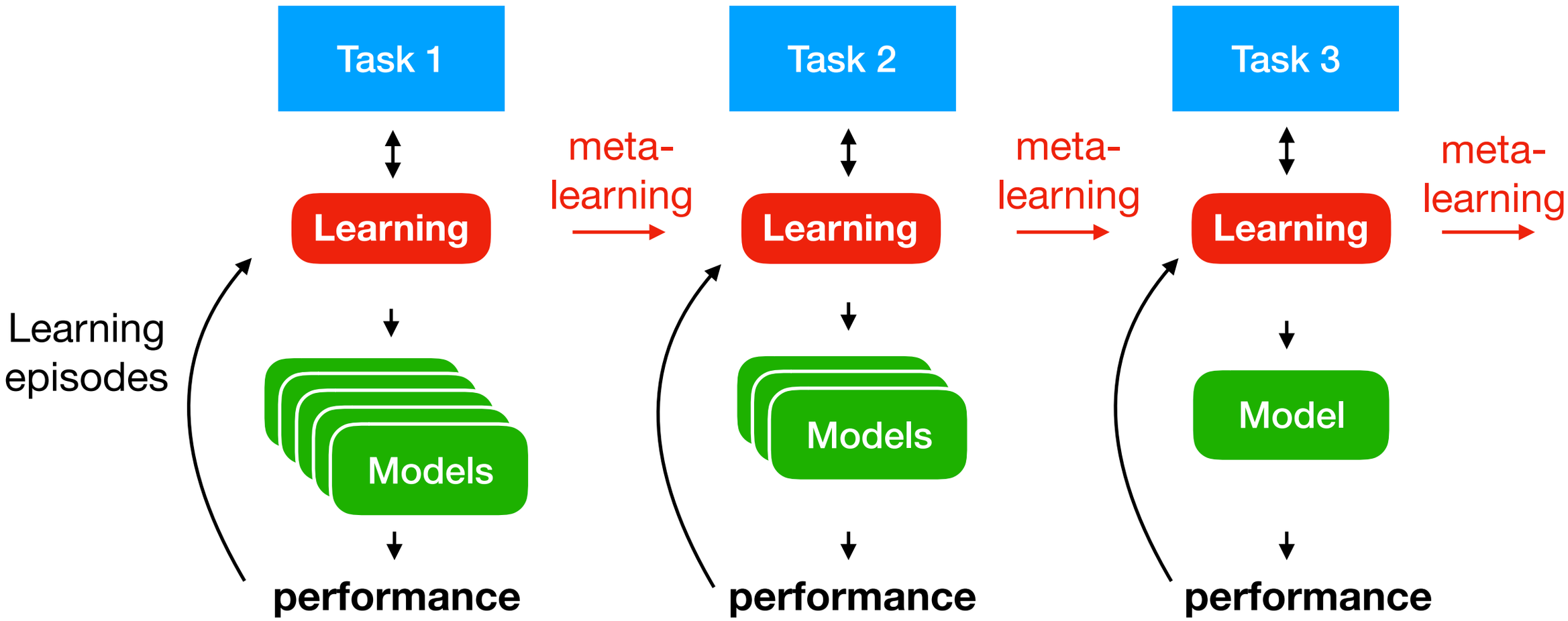
# Learning is a never-ending process

Learning humans also seek/create related tasks


E.g. find many similar puzzles, solve them in different ways,…

# Learning is a never-ending process

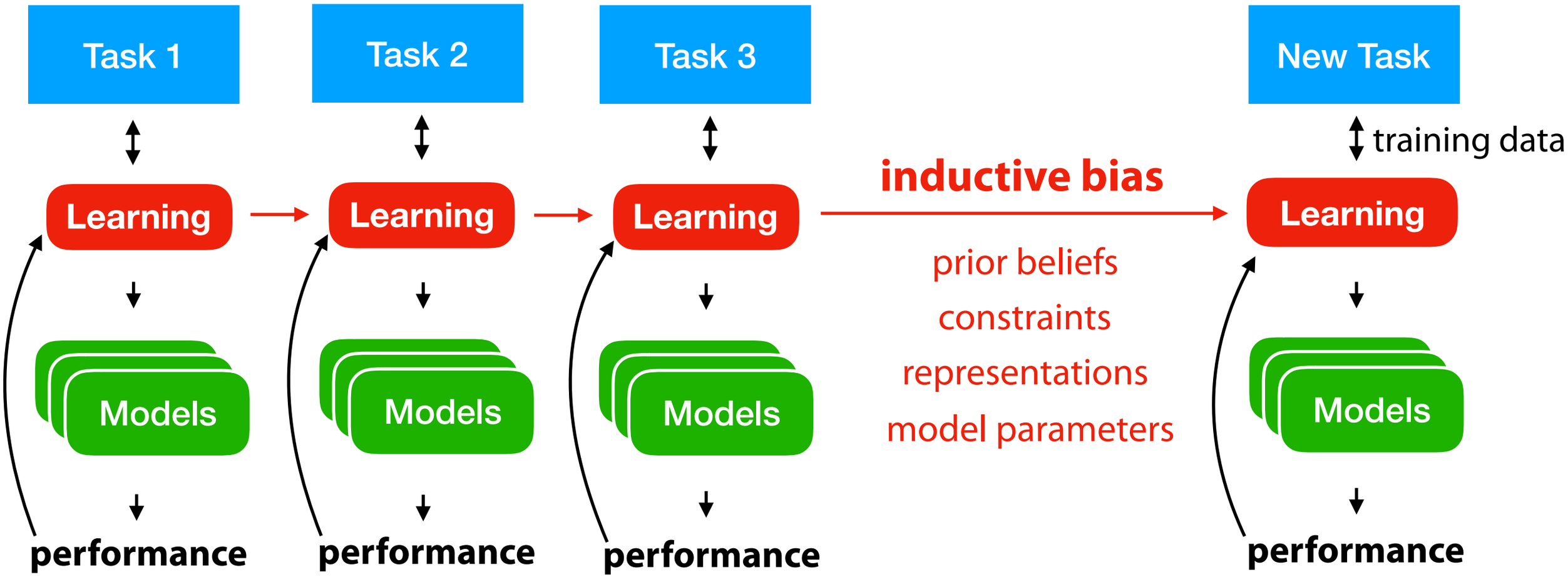
Humans learn *across* tasks
Why? Requires less trial-and-error, less data



# Learning to learn

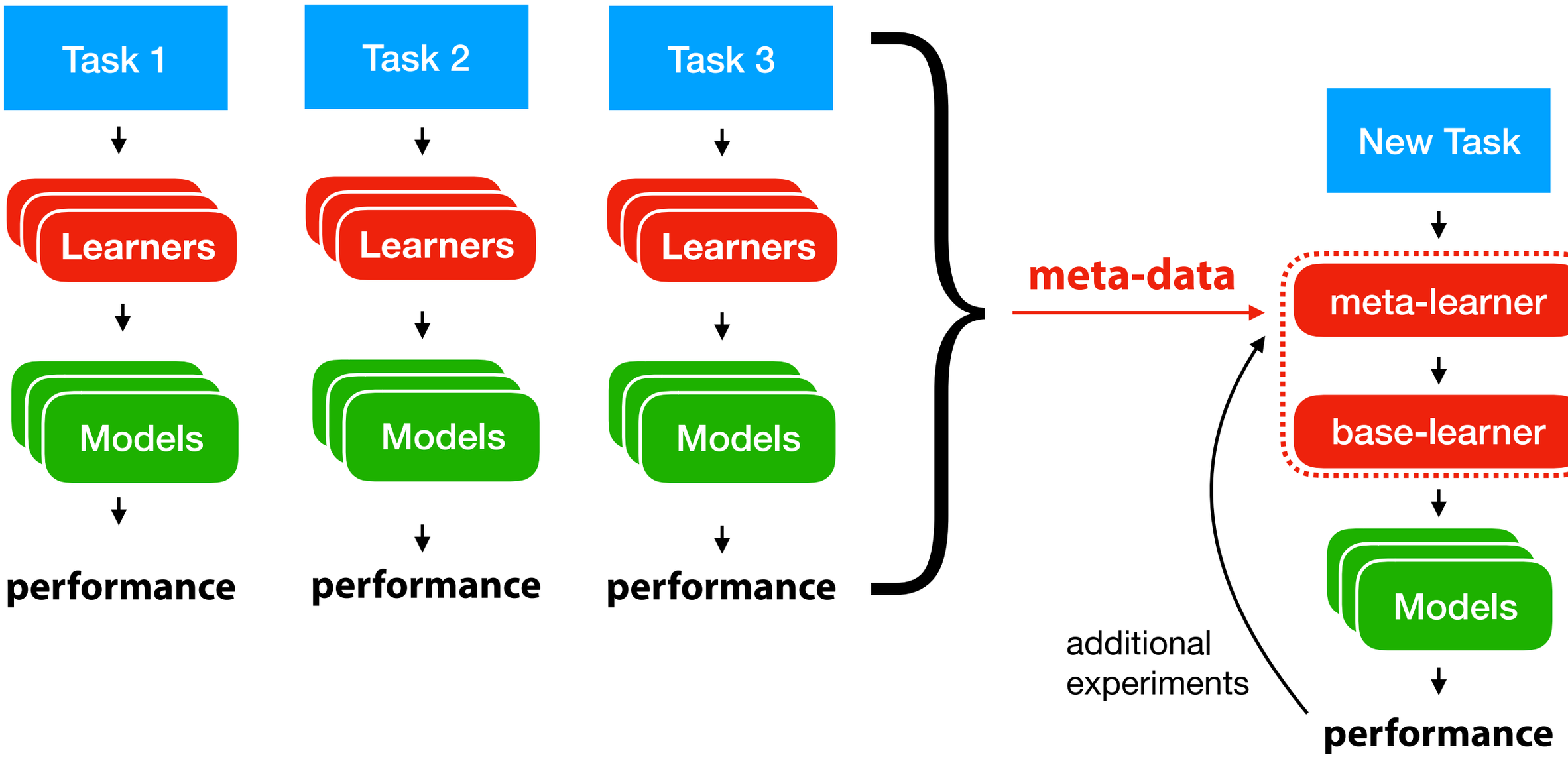
If prior tasks are *similar*, we can ***transfer*** prior knowledge to new tasks

**Inductive bias**: assumptions added to the training data to learn effectively



# Meta-learning

Meta-learner *learns* a (base-)learning algorithm, based on *meta-data*



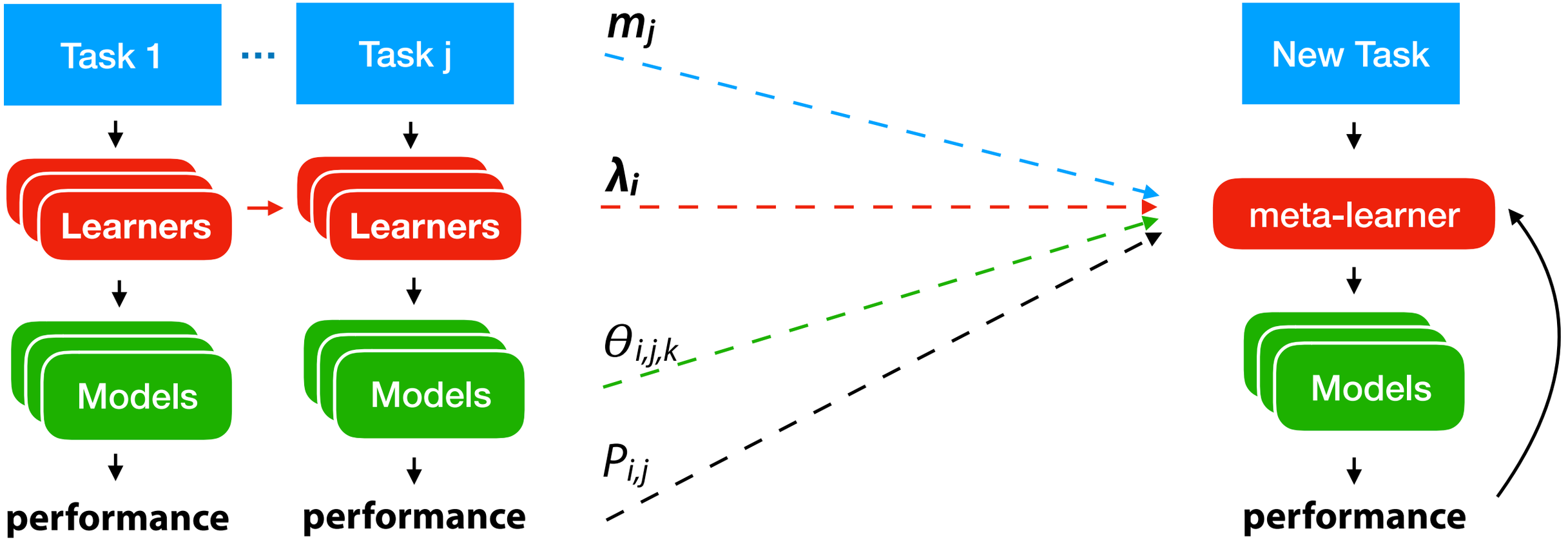
# Meta-data

$\boldsymbol{m_j}$ Description of task j (meta-features)

$\boldsymbol{\lambda_i}$ Configuration i (architecture + hyperparameters)

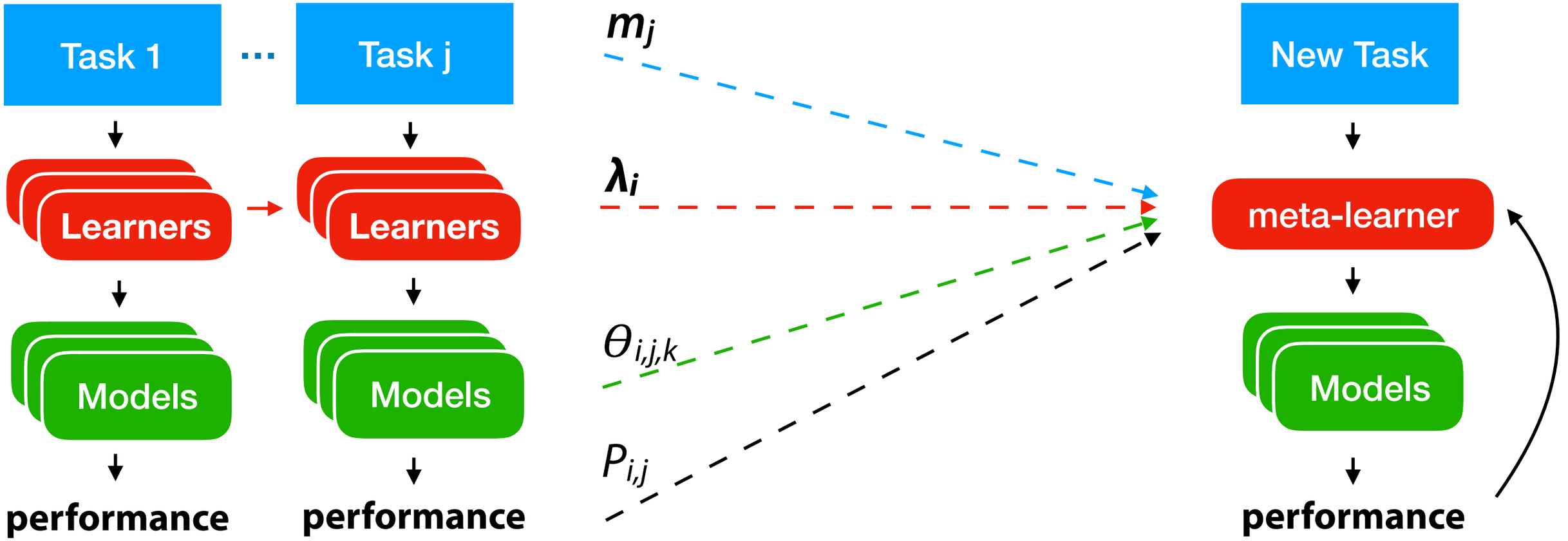
$\theta_{i,j,k}$ Model parameters (e.g. weights)

$P_{i,j}$ Performance estimate of model built with $\lambda_i$ *on task j*



# How?

1. Learn from observing learning algorithms (for *any* task) $\boldsymbol{\lambda_i}$ $P_{i,j}$
2. Learn what may likely work (for *somewhat similar* tasks) $\boldsymbol{m_j}$ $\boldsymbol{\lambda_i}$ $P_{i,j}$
3. Learn from previous models (for *very similar* tasks) $\theta_{i,j,k}$ $\boldsymbol{(m_j)}$ $\boldsymbol{\lambda_i}$ $P_{i,j}$

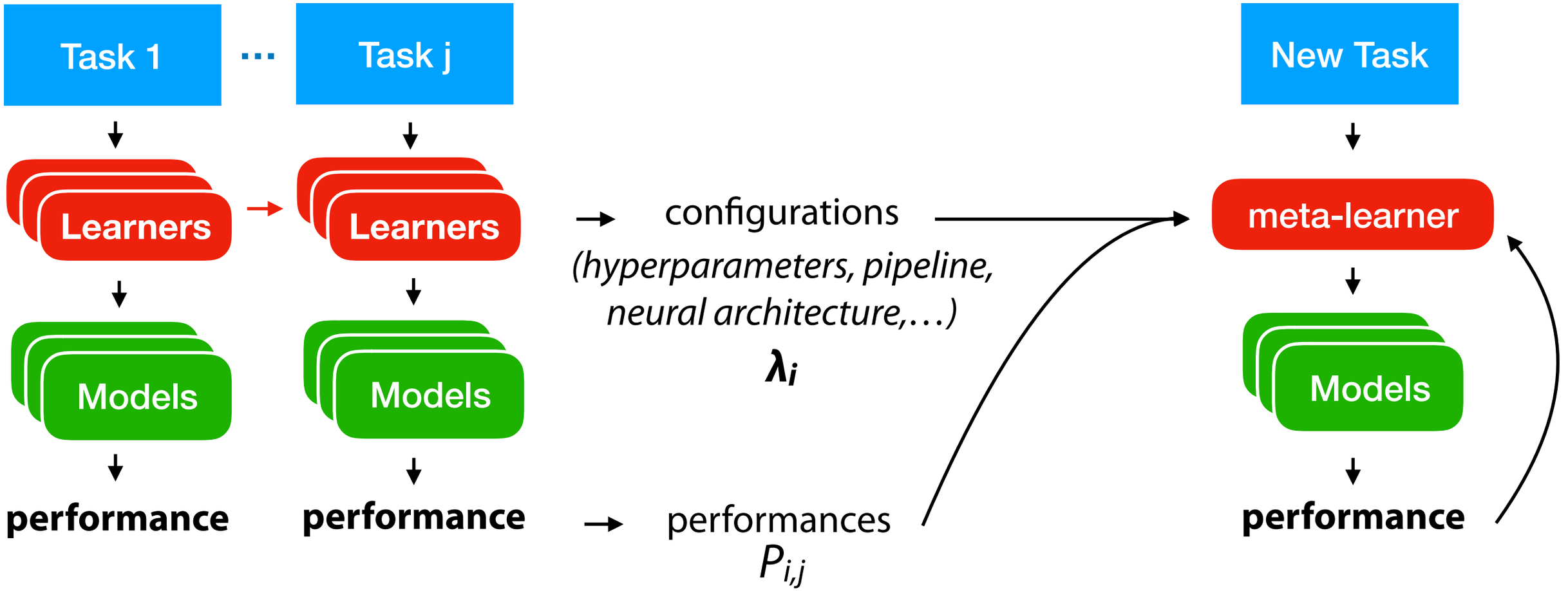


# 1. Learn from observing learning algorithms

Define, store and use meta-data:

- ***configurations***: settings that uniquely define the model

- ***performance*** (e.g. accuracy) on specific tasks

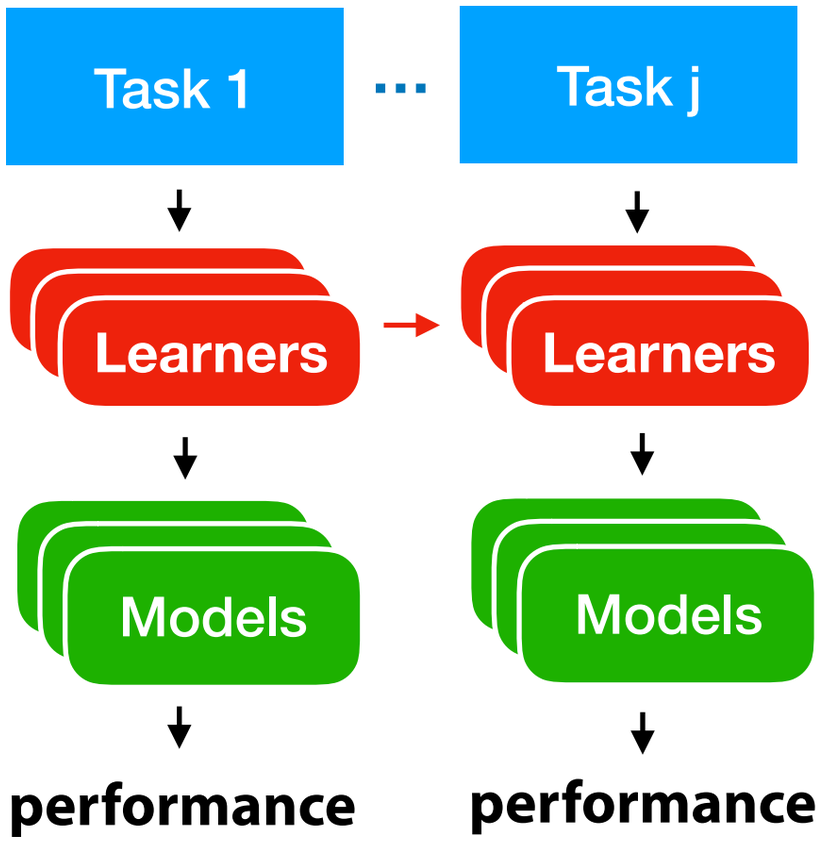


# 1. Learn from observing learning algorithms

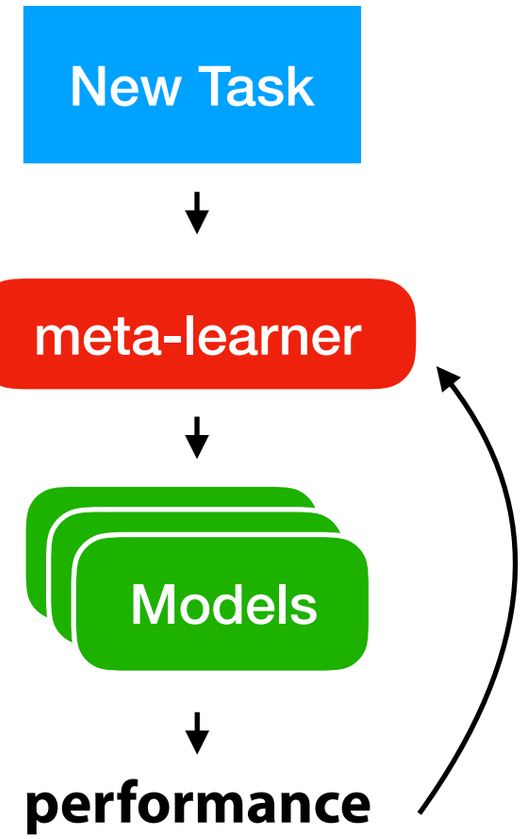
Define, store and use meta-data:

- ***configurations***: settings that uniquely define the model

- ***performance*** (e.g. accuracy) on specific tasks

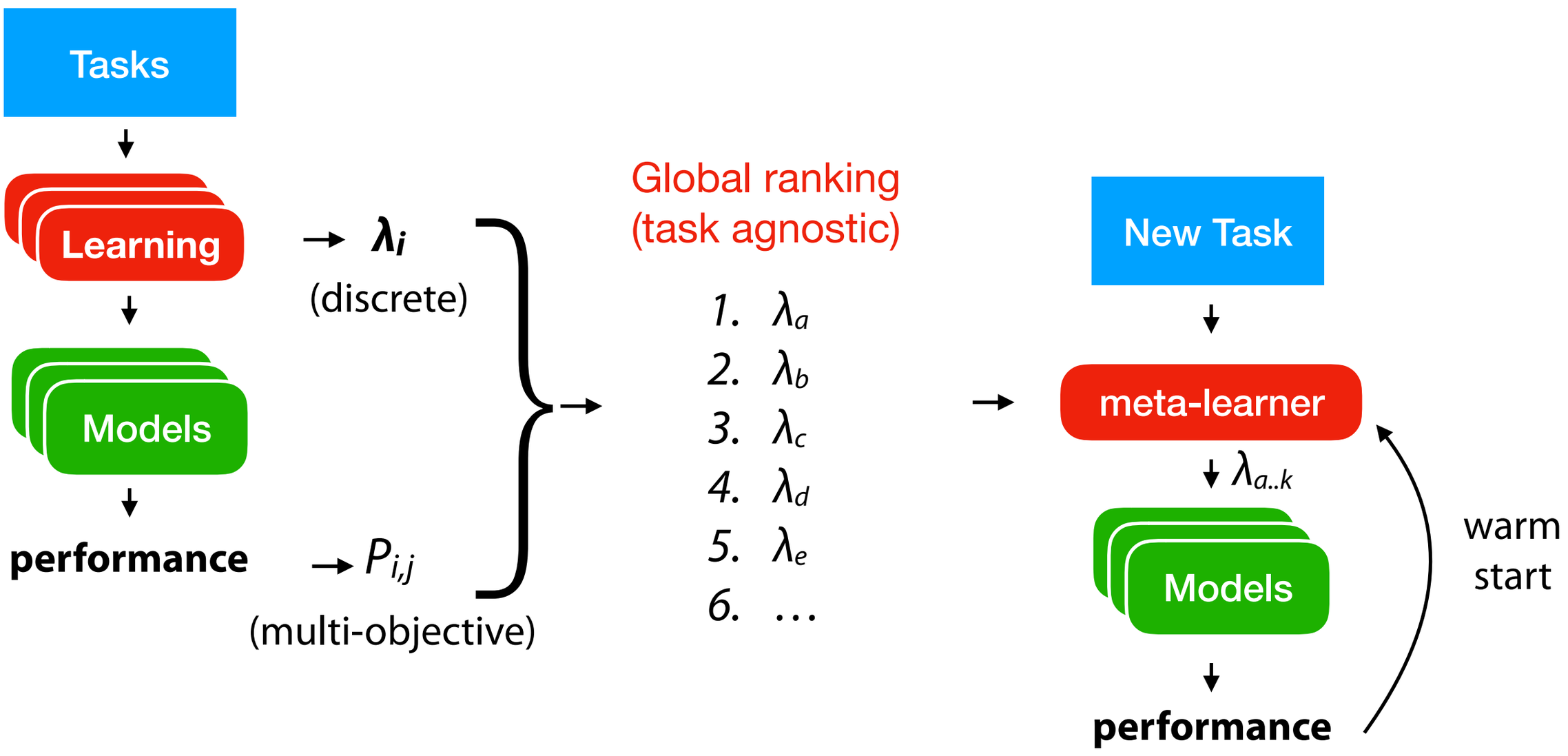



| task | model config $\lambda$ | score P |
|---|---|---|
| 0 | alg=SVM, C=0.1 | 0,876 |
| 0 | alg=NN, lr=0.9 | 0,921 |
| 0 | alg=RF, mtry=0.1 | 0,936 |
| 1 | alg=SVM, C=0.2 | 0,674 |
| 1 | alg=NN, lr=0.5 | 0,777 |
| 1 | alg=RF, mtry=0.4 | 0,791 |



# Rankings

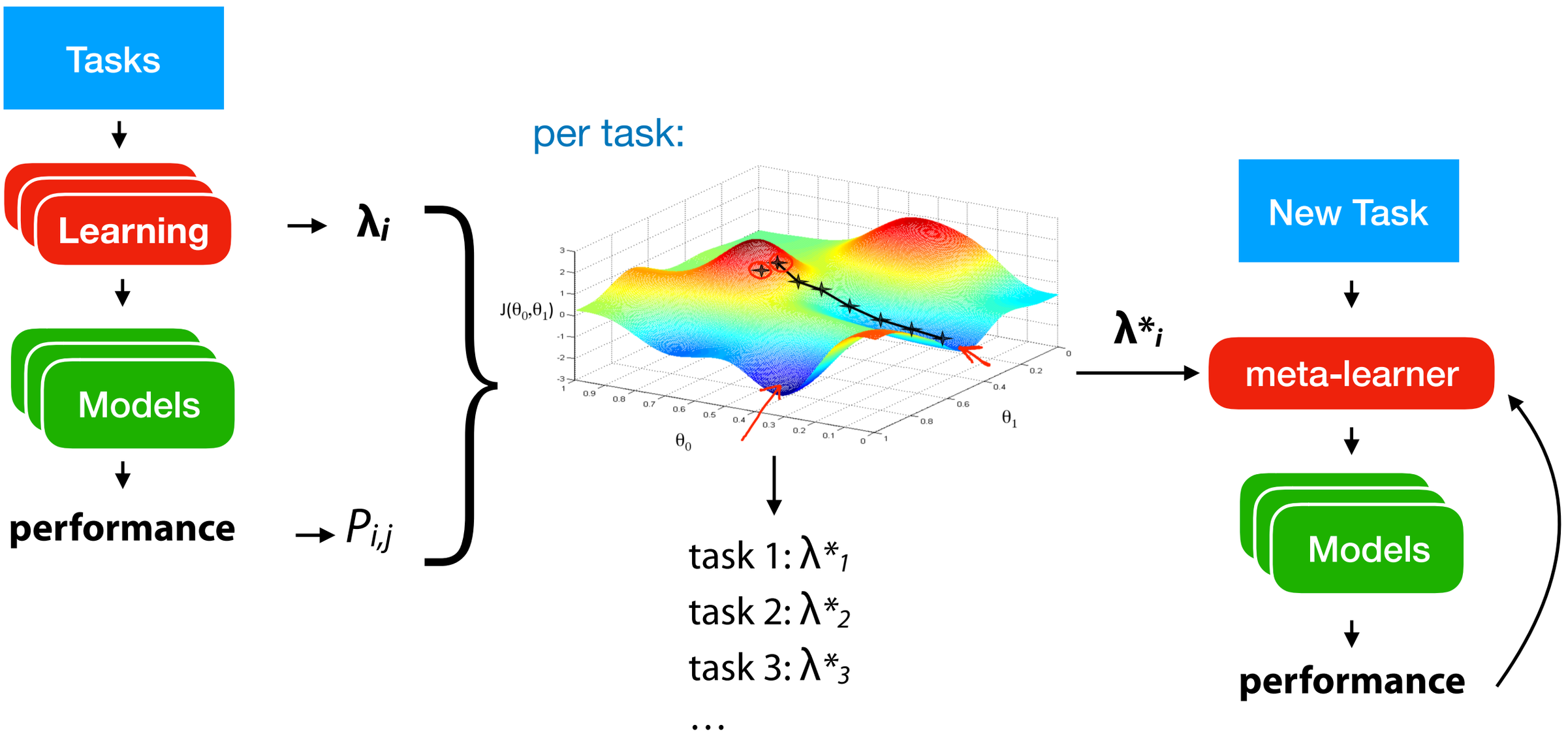
- Build a *global (multi-objective) ranking,* recommend the top-K
- Can be used as a *warm start* for optimization techniques
  - E.g. Bayesian optimization, evolutionary techniques,…



*Wistuba et al. 2015*

# Warm-starting with plugin estimators

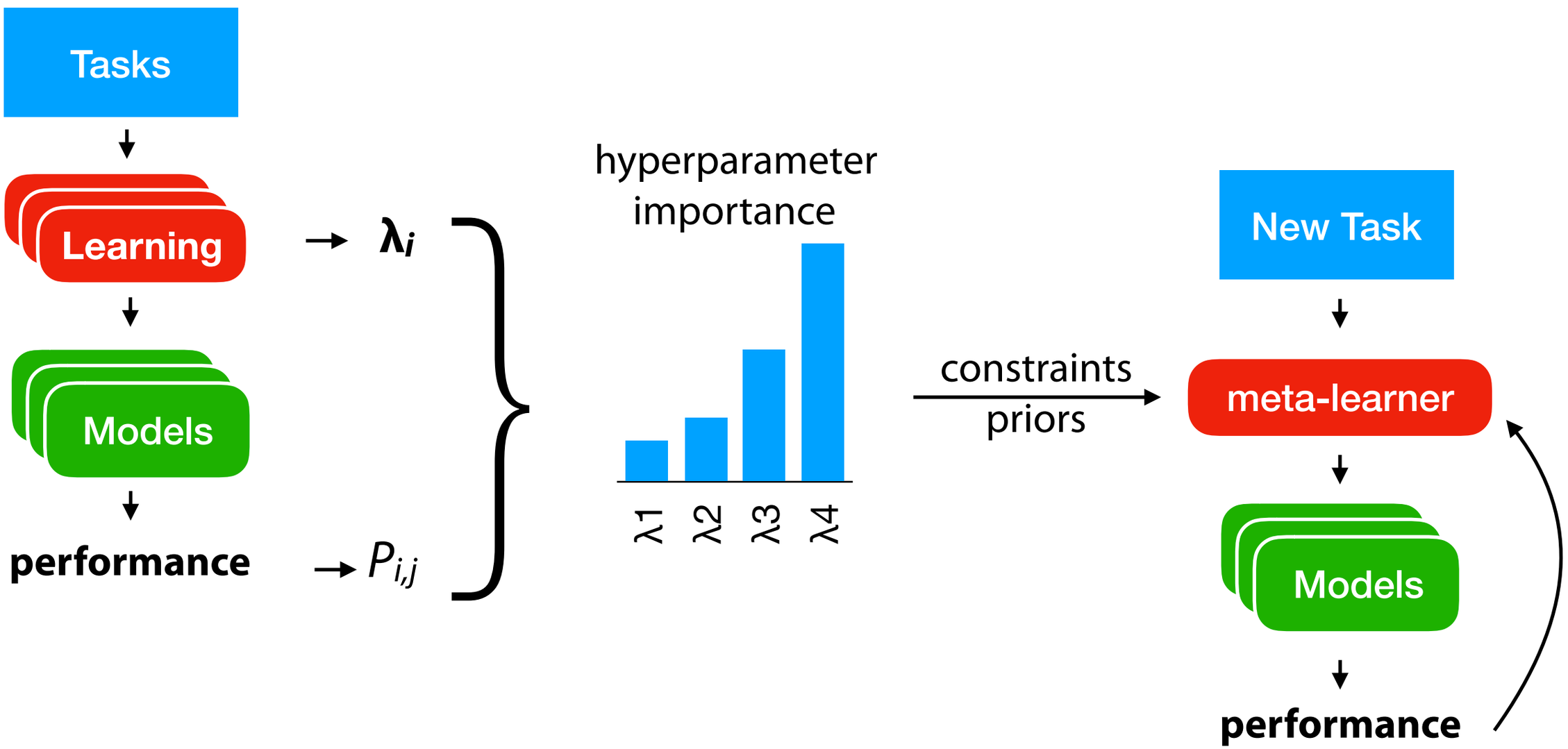
- What if prior configurations are not optimal?
- Per task, fit a differentiable plugin estimator on all evaluated configurations
- Do gradient descent to find optimized configurations, recommend those



[1] van Rijn & Hutter 2018

# Hyperparameter behavior

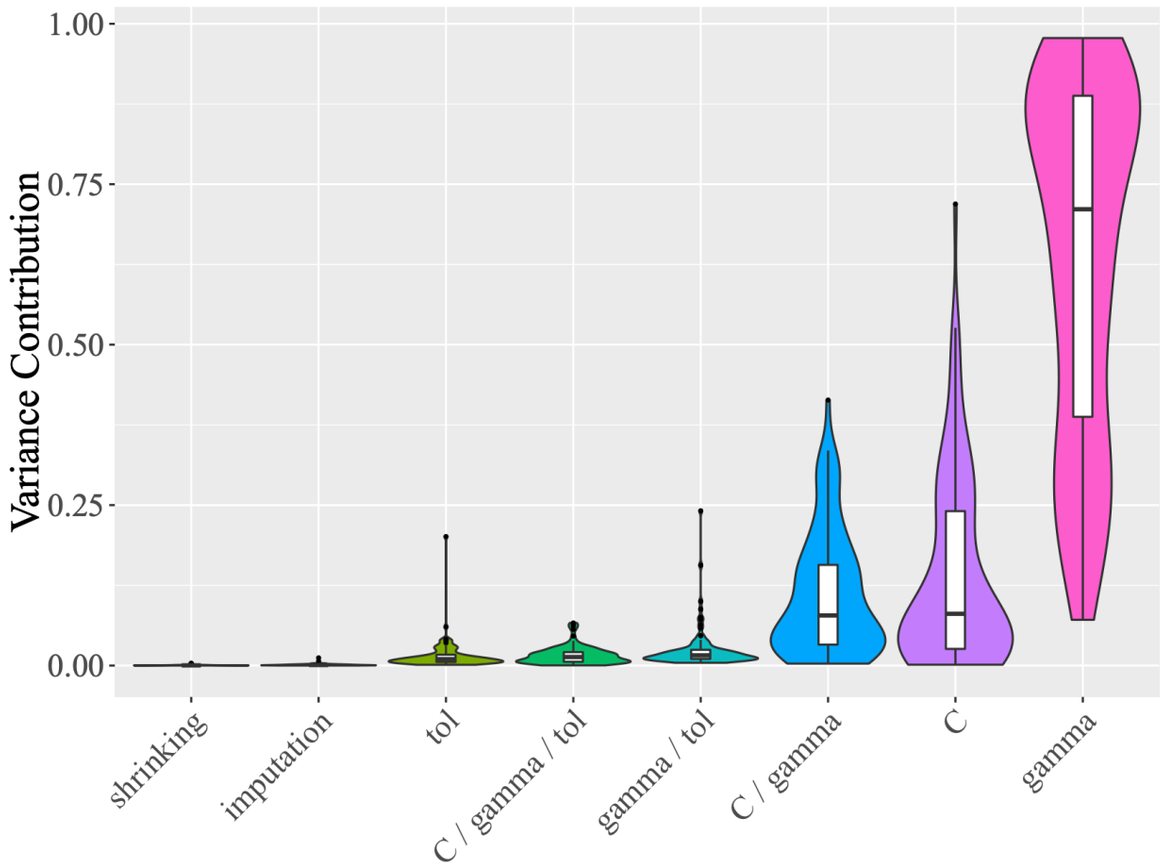
- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.



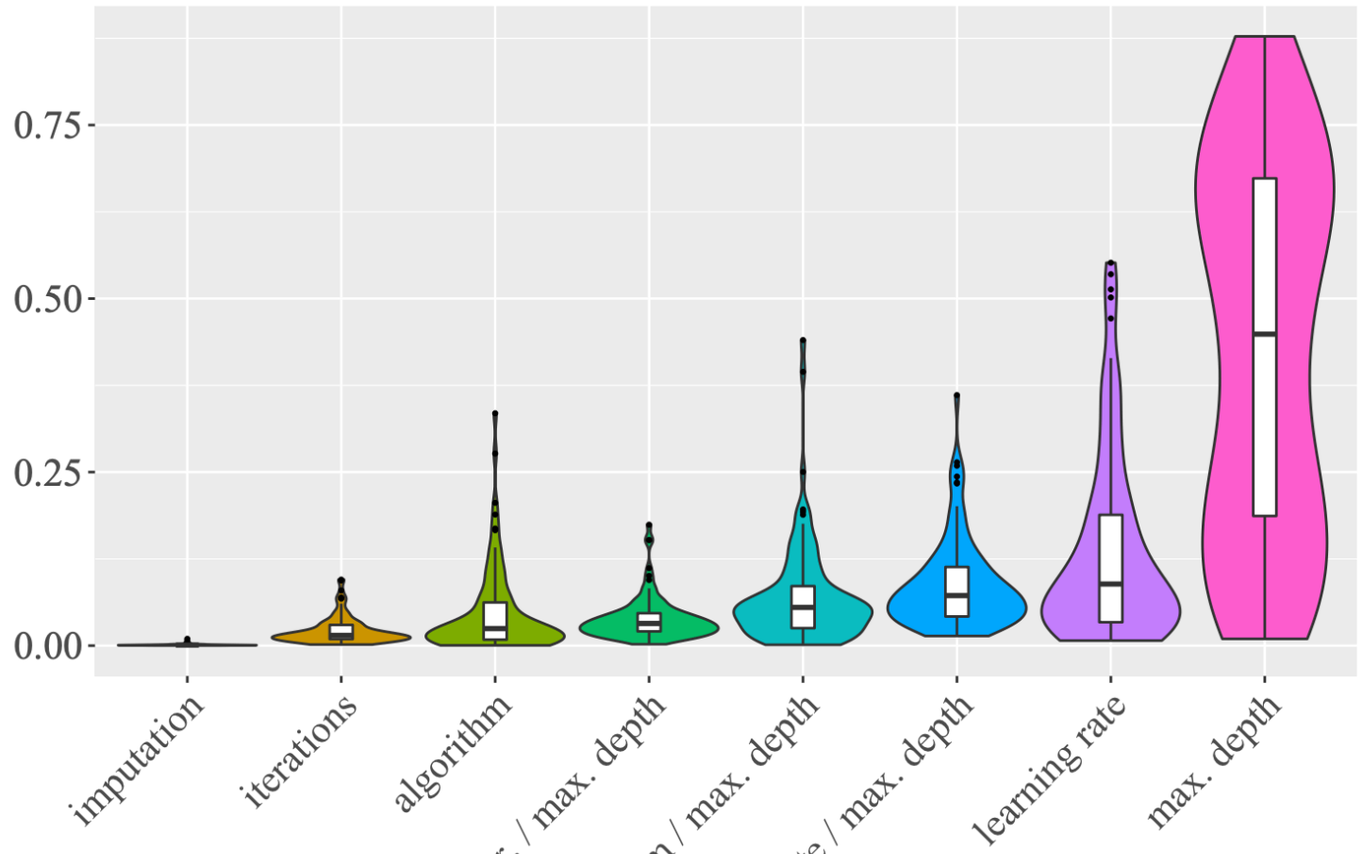
[1] van Rijn & Hutter 2018

# Hyperparameter behavior

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.




**SVM (RBF)**

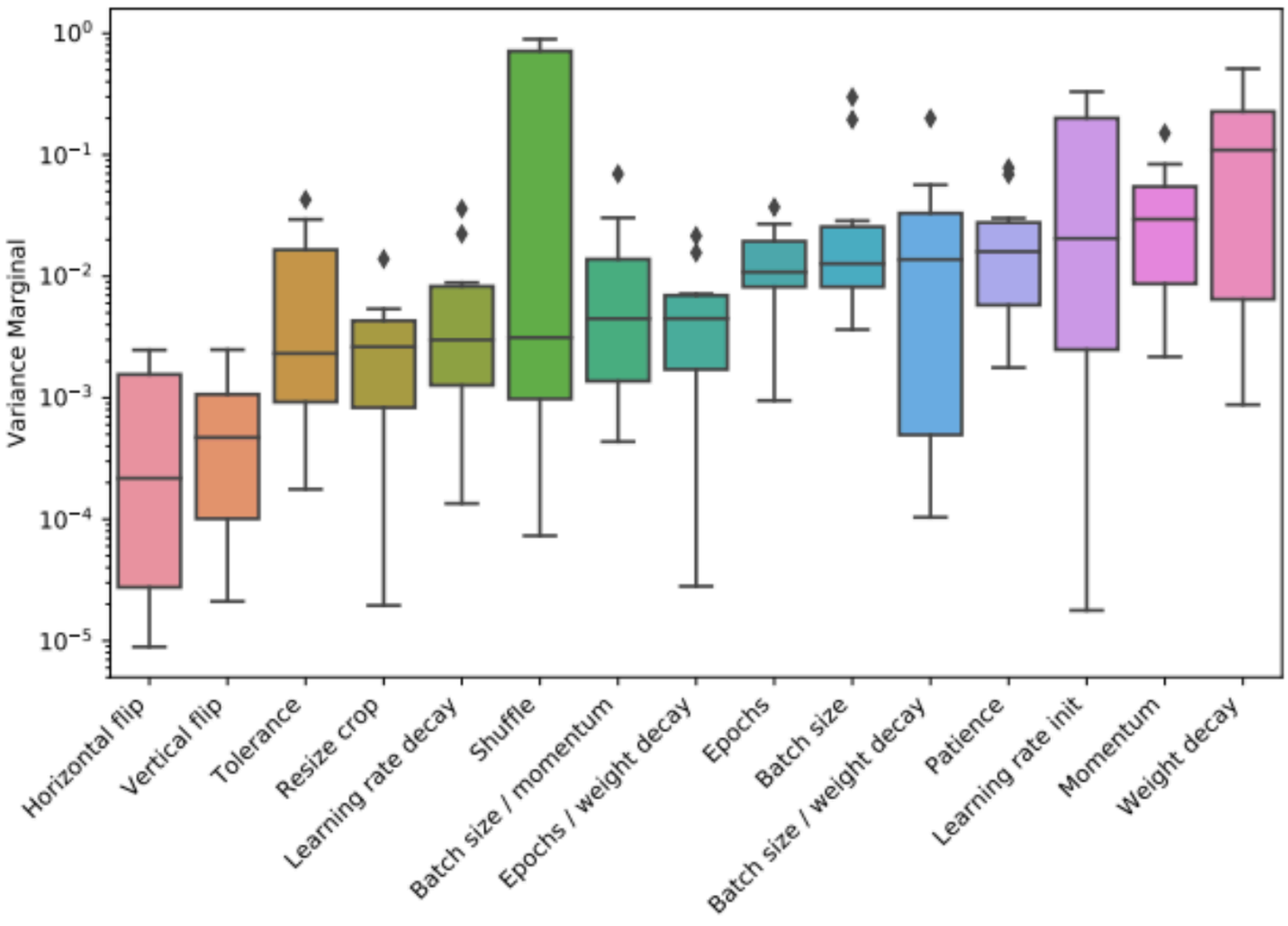



**AdaBoost**

[1] van Rijn & Hutter 2018

# Hyperparameter behavior

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.

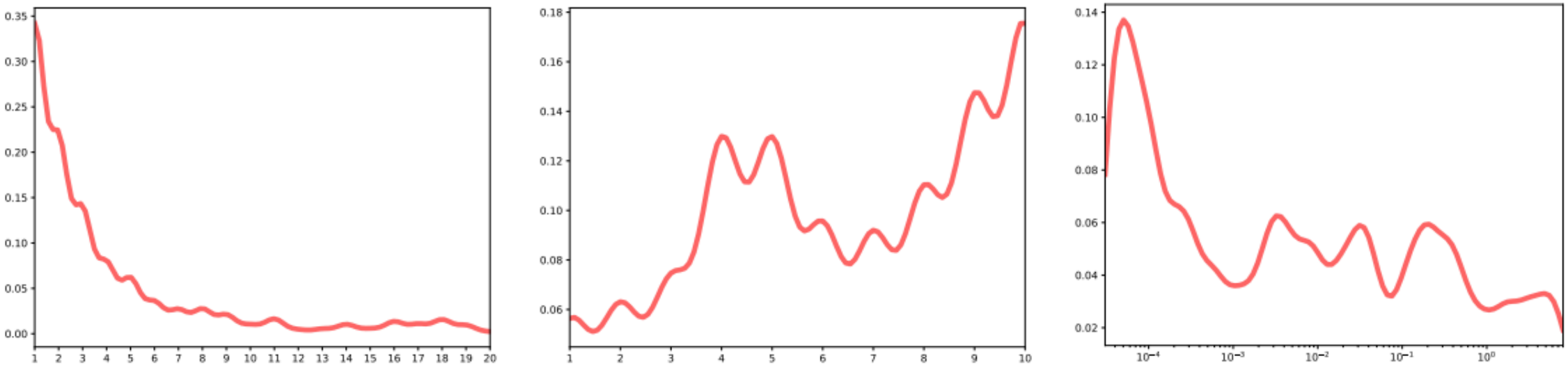



**ResNets for image classification** *(under review)*

[1] van Rijn & Hutter 2018

# Hyperparameter behavior

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.
  Learn priors for hyperparameter tuning (e.g. Bayesian optimization)



(a) RF: min. samples per leaf

(b) Adaboost: max. depth of tree

(c) SVM (RBF kernel): gamma

**Priors (KDE) for the most important hyperparameters**

[1] van Rijn & Hutter 2018

[2] Probst et al. 2018

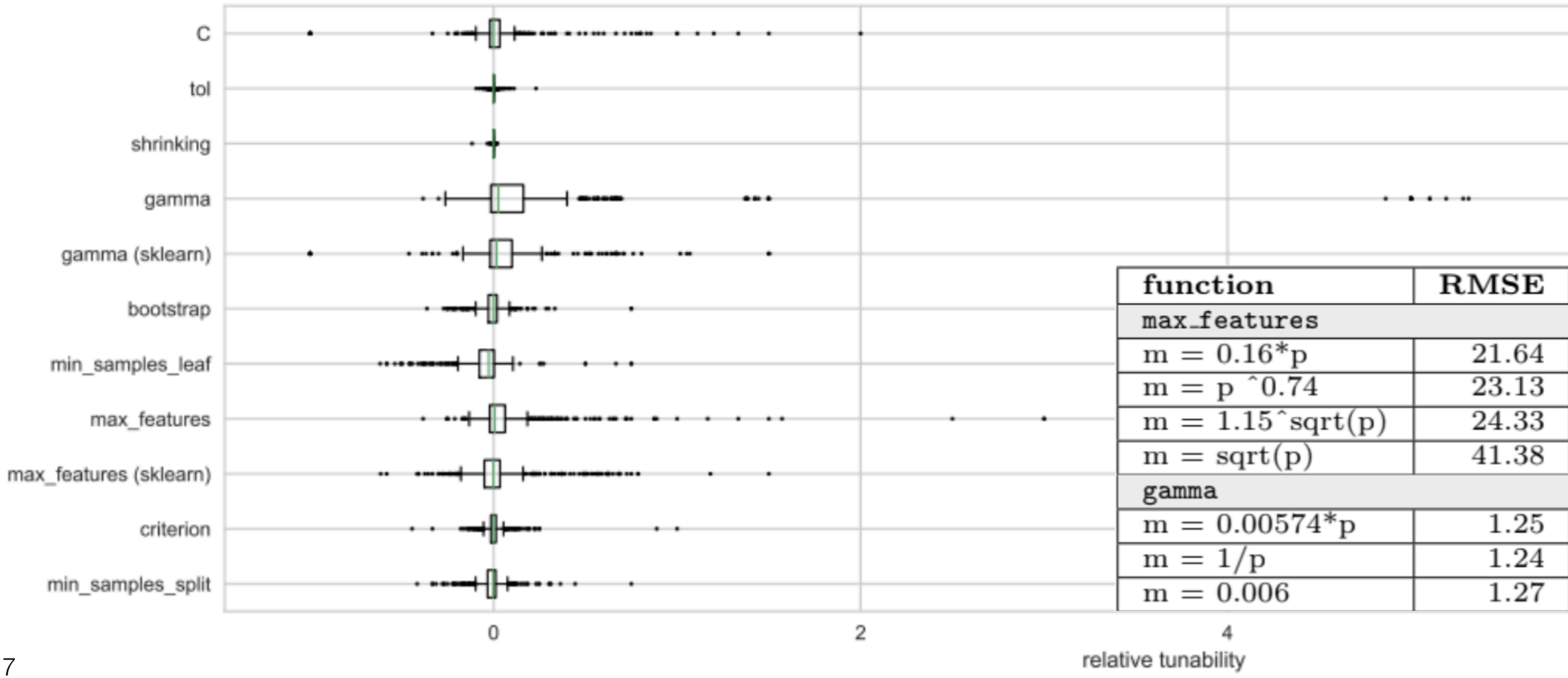
[3] Weerts et al. 2018

[4] van Rijn et al. 2018

# Hyperparameter behavior

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.

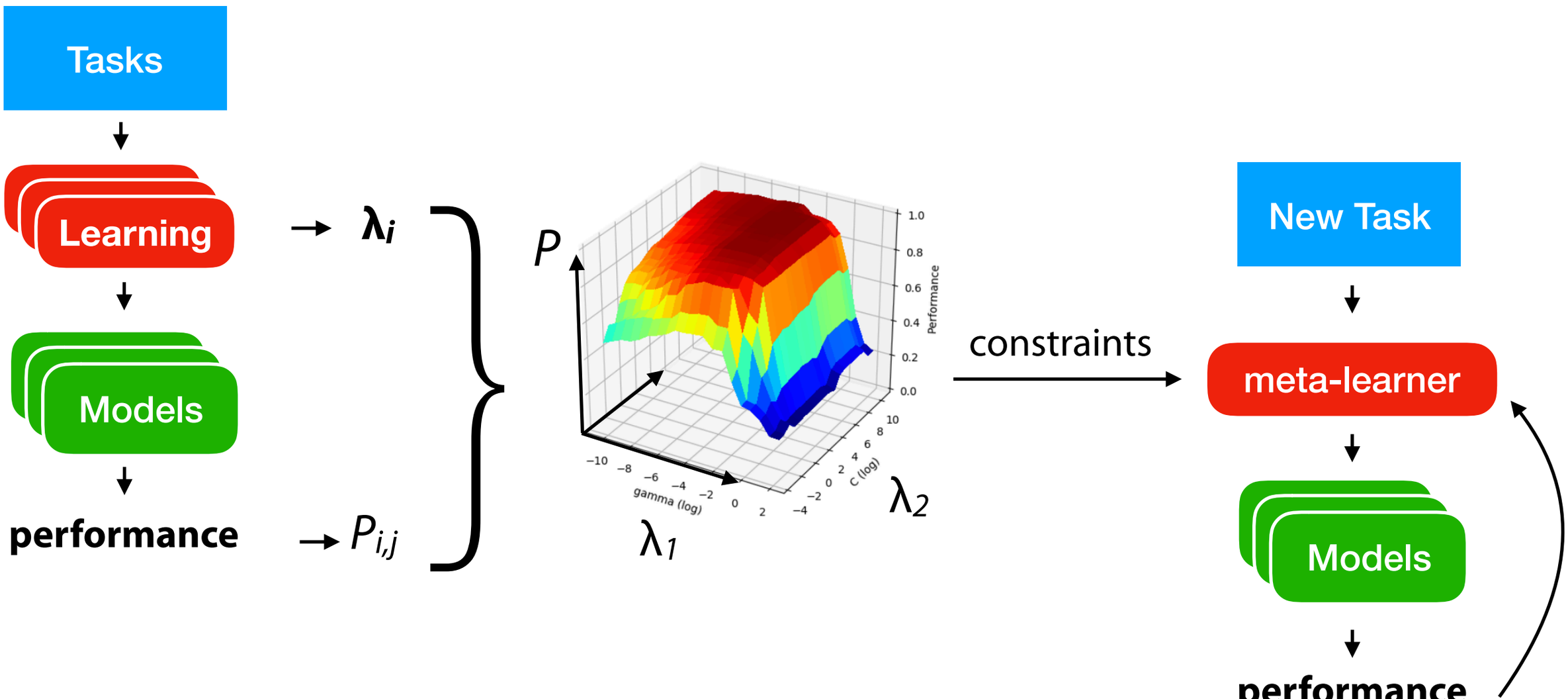
- **Tunability** [2,3,4]
  *Learn* good defaults, measure improvement from tuning over defaults




| function | RMSE |
|---|---|
| max_features | |
| m = 0.16*p | 21.64 |
| m = p ^0.74 | 23.13 |
| m = 1.15^sqrt(p) | 24.33 |
| m = sqrt(p) | 41.38 |
| gamma | |
| m = 0.00574*p | 1.25 |
| m = 1/p | 1.24 |
| m = 0.006 | 1.27 |

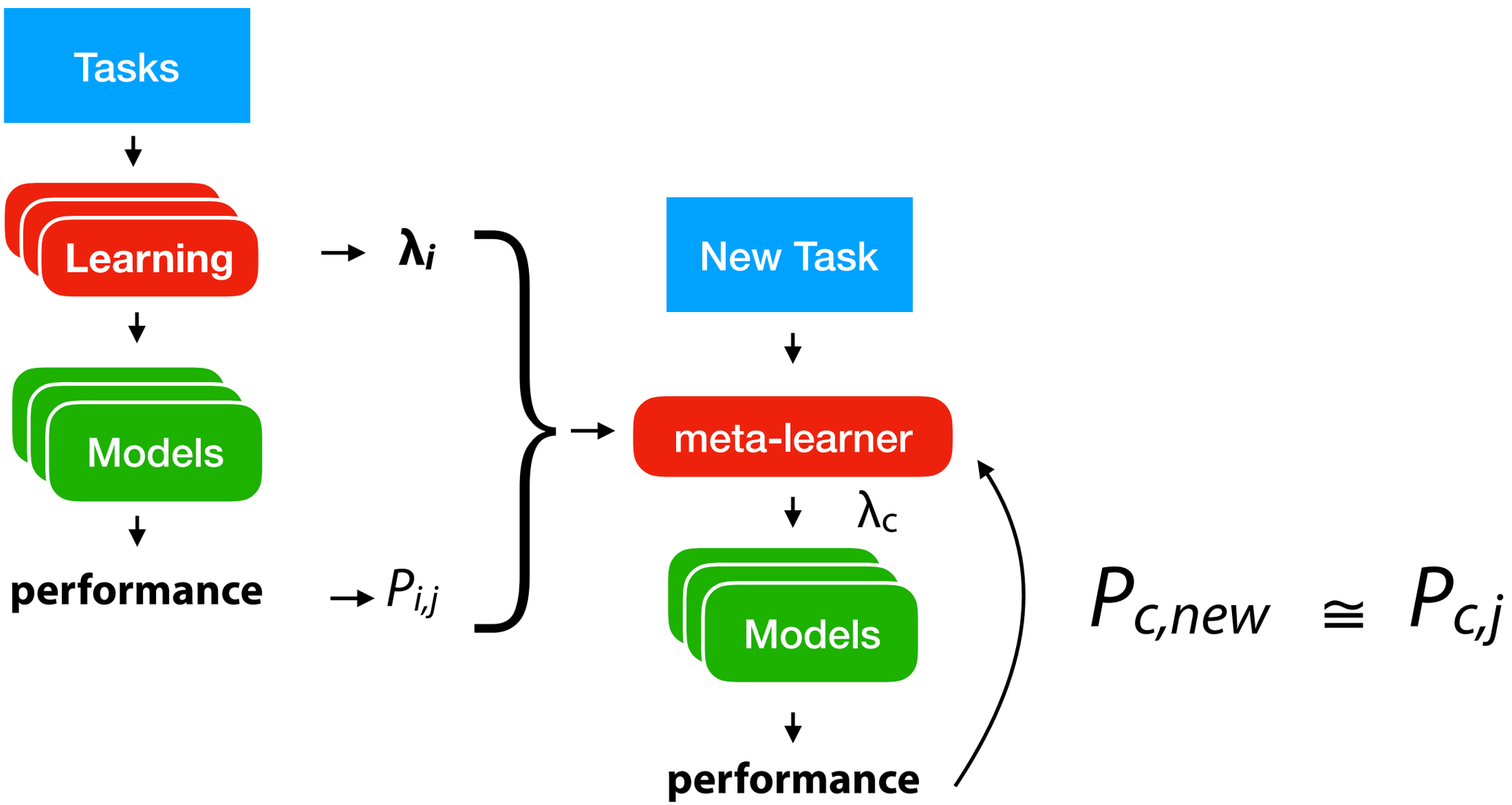
# Hyperparameter behavior

- **Search space pruning**
  Exclude regions yielding bad performance on (similar) tasks



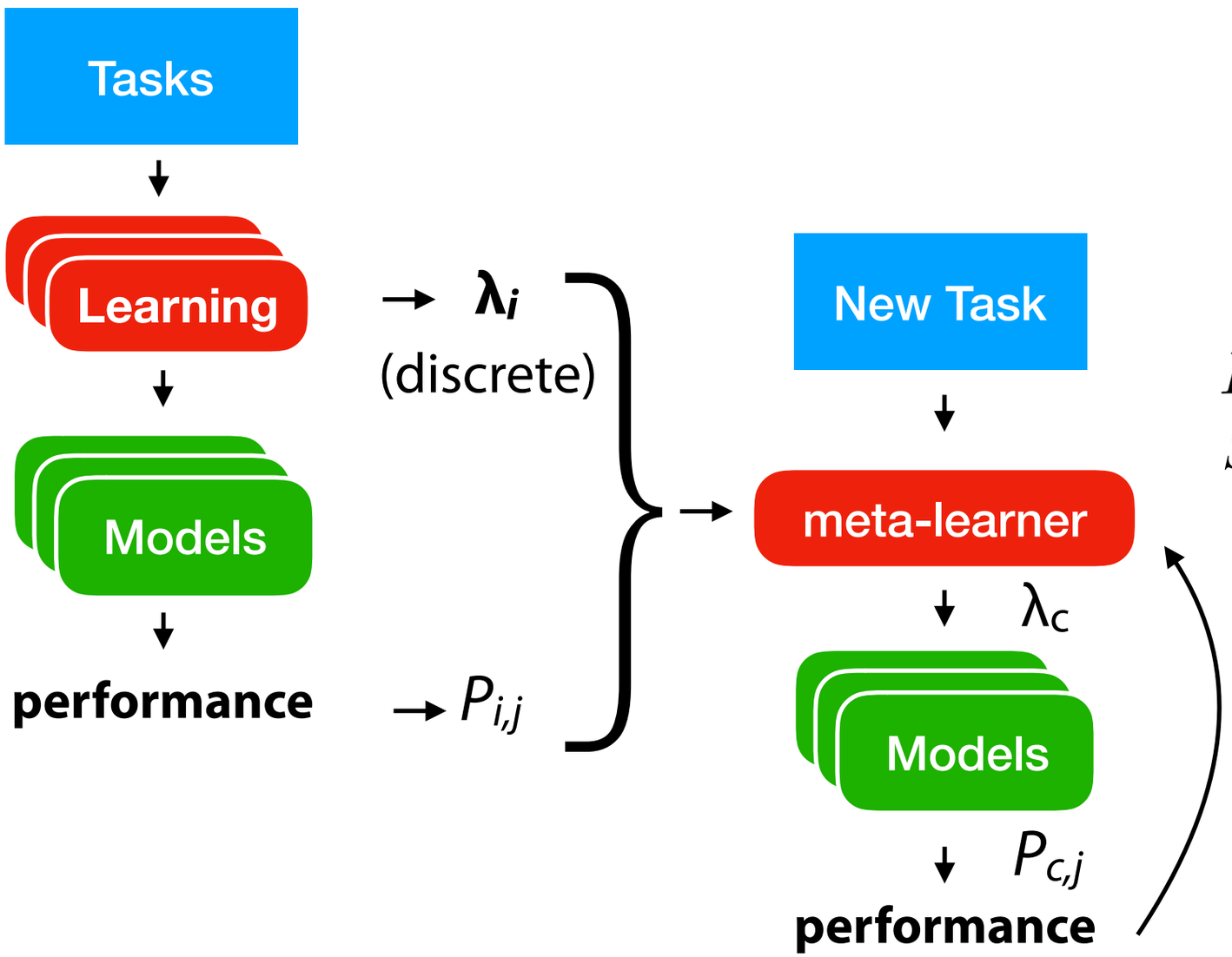
# Learning task similarity

- Experiments on the *new* task can tell us how it is similar to *previous* tasks
- **Task are *similar*** if observed *performance* of configurations is similar
- Use this to recommend new configurations, run, repeat



# Active testing

- Learn task similarity while tuning configurations
- Tournament-style selection, warm-start with overall best config $\lambda_{best}$
- Next candidate $\lambda_c$ : the one that beats current $\lambda_{best}$ on similar tasks




$$RL_{a,b,j} = P_{a,j} - P_{b,j}$$
$$Sim(t_j, t_{new}) = Corr([RL_{a,b,j}], [RL_{a,b,new}])$$

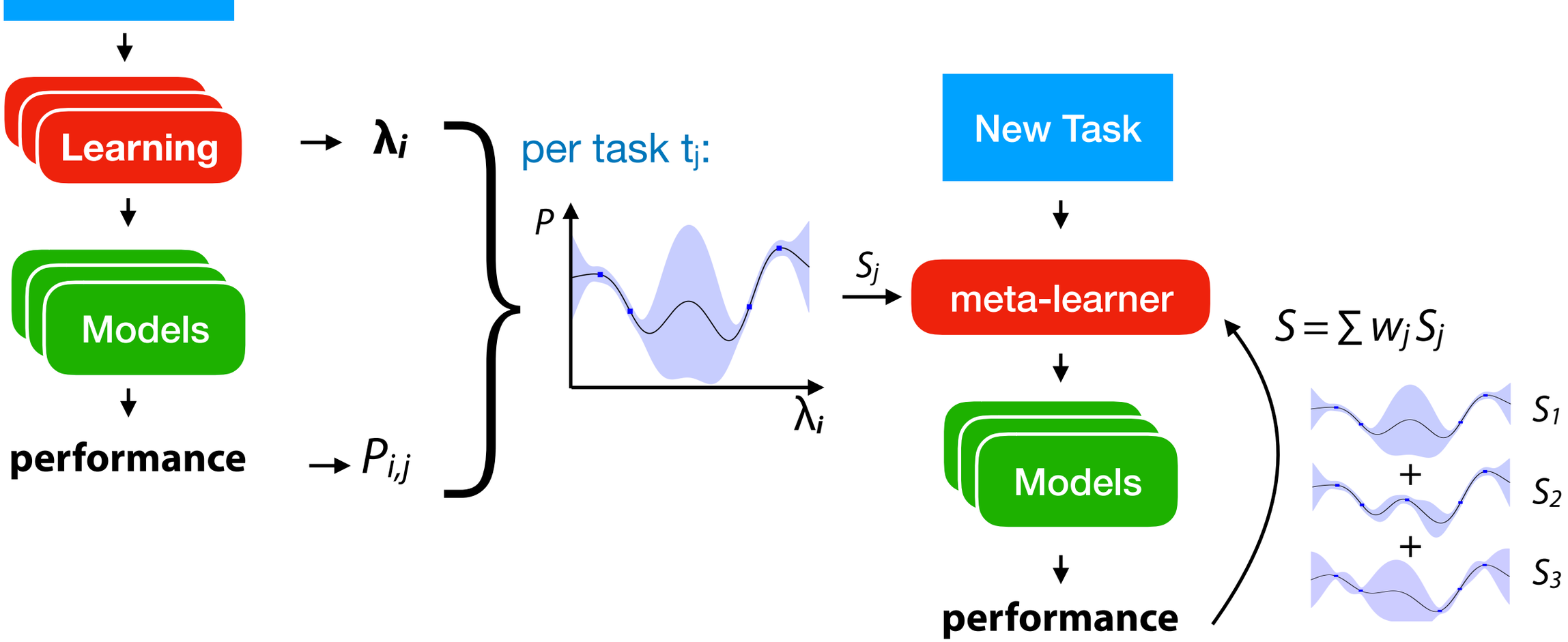
Select $\lambda_c > \lambda_{best}$ on similar tasks

[1] Wistuba et al. 2018

[2] Feurer et al. 2018

# Surrogate model transfer

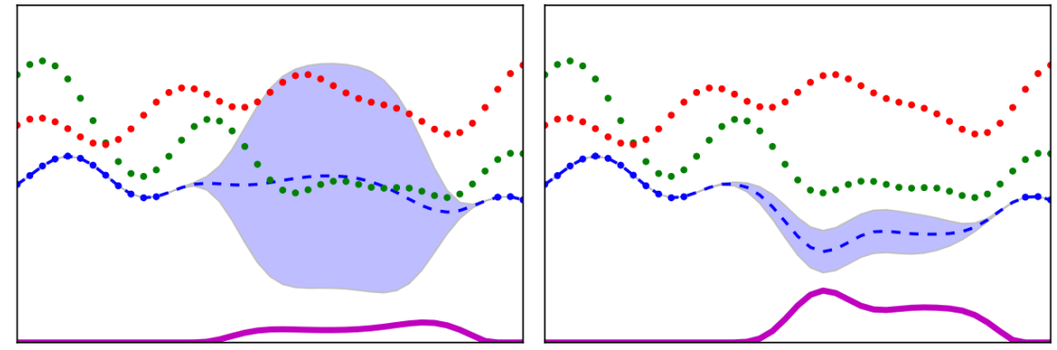
- If task j is *similar* to the new task, its surrogate model $S_j$ will likely transfer well
- Sum up all $S_j$ predictions, weighted by task similarity (as in active testing)[1]
- Build combined Gaussian process, *weighted by current performance* on new task[2]



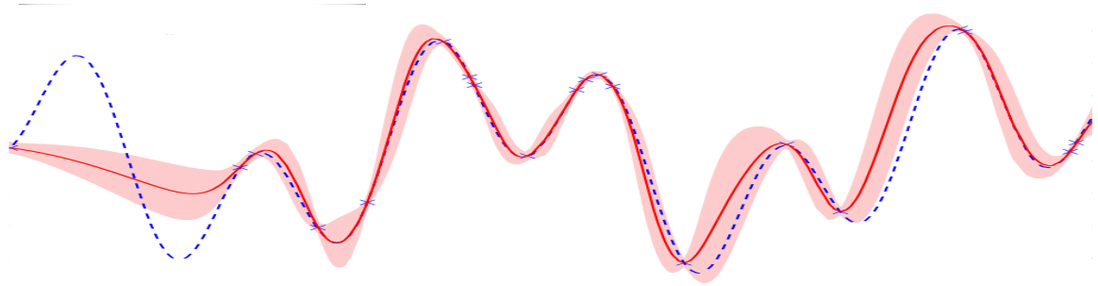
[1] Swersky et al. 2013
[2] Springenberg et al. 2016
[3] Golovin et al. 2017

# Multi-task Bayesian optimization

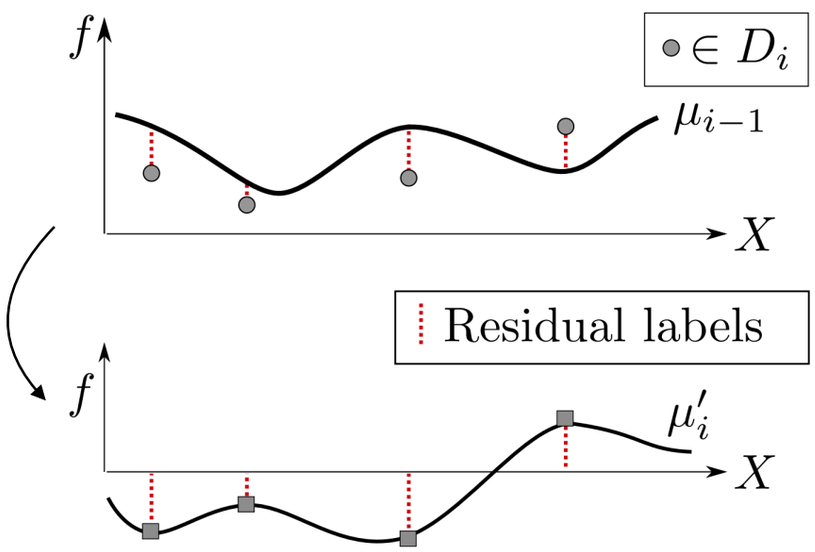
- **Multi-task Gaussian processes:** train surrogate model on t tasks simultaneously[1]
  - If tasks are similar: transfers useful info
  - Not very scalable




- **Bayesian Neural Networks** as surrogate model[2]
  - Multi-task, more scalable

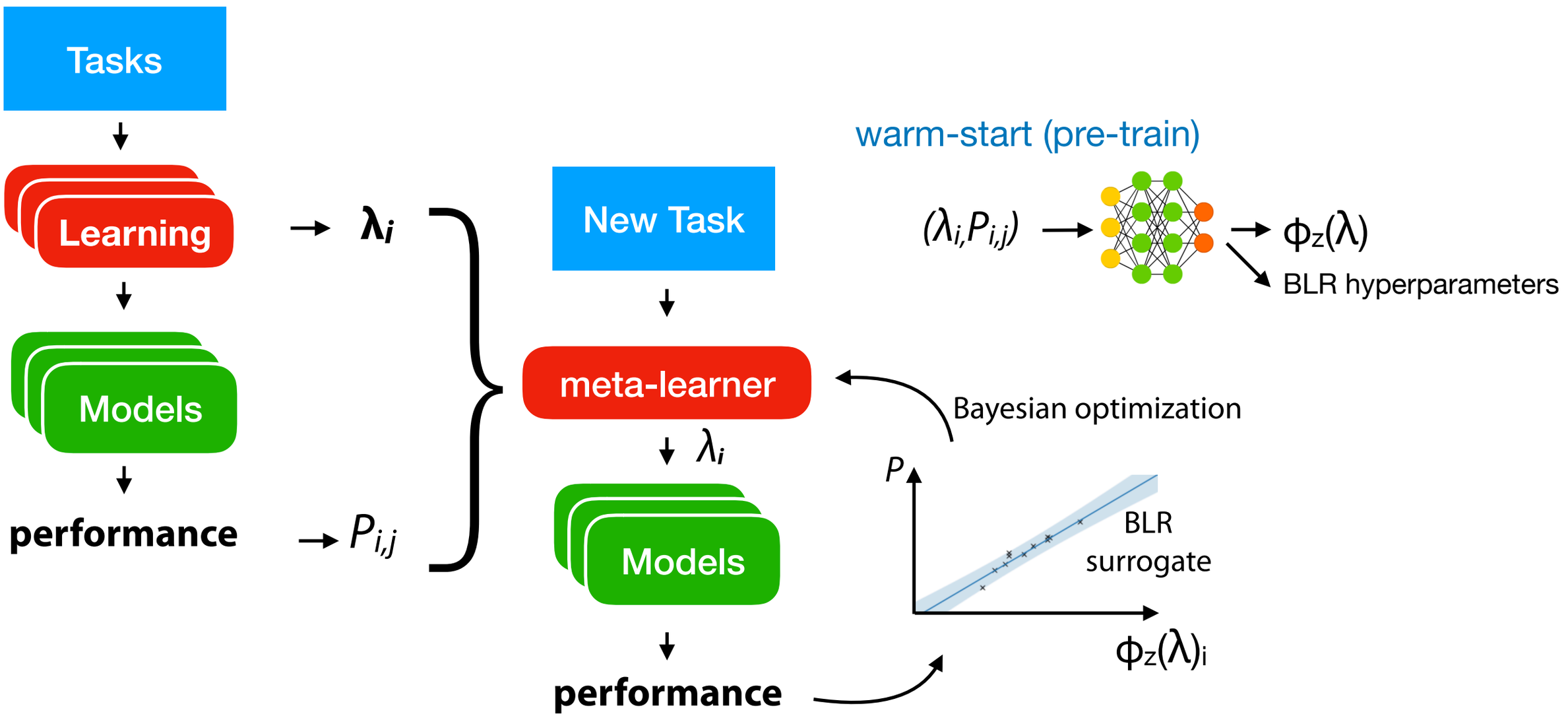


- **Stacking** Gaussian Process regressors (Google Vizier)[3]
  - Continual learning (sequential similar tasks)
  - Transfers a prior based on residuals of previous GP



# More scalable variants

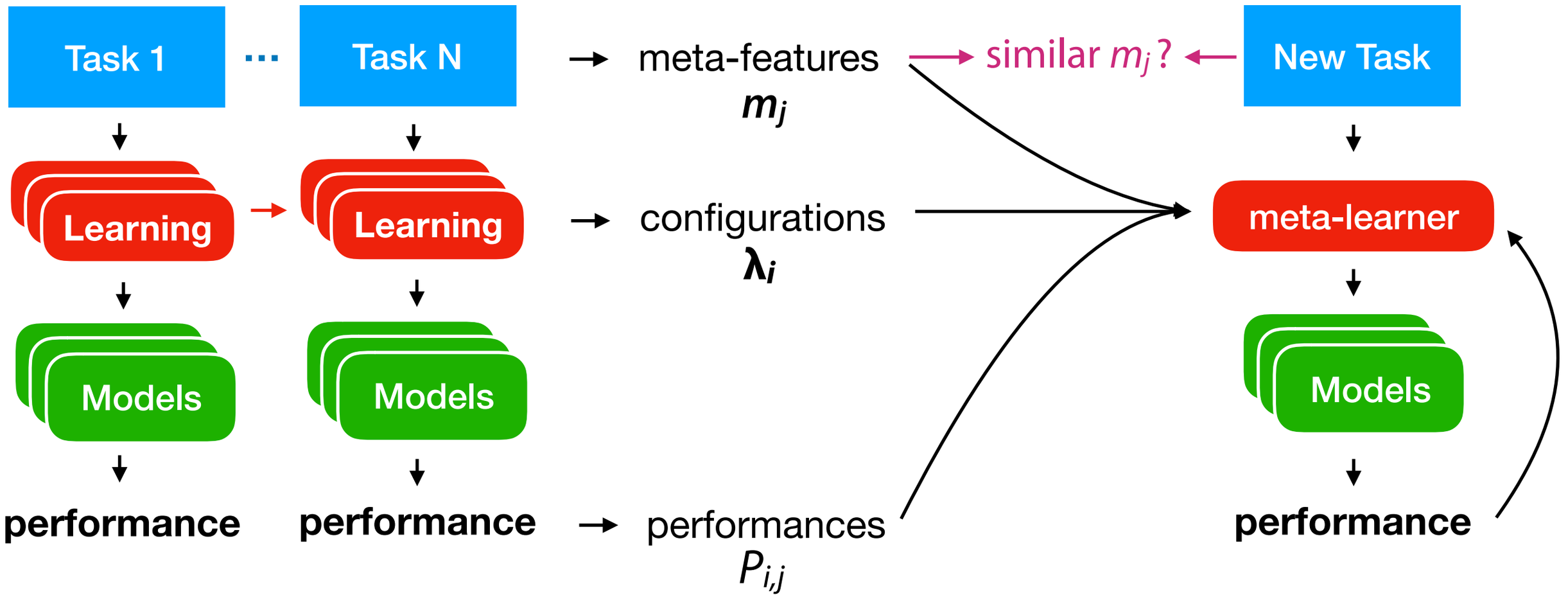
- Bayesian linear regression (BLR) surrogate model on every task
- Use neural net to learn a suitable basis expansion $\phi_z(\lambda)$ for all tasks
- Scales linearly in # observations, transfers info on configuration space



# 2. Learn what may likely work (for *partially similar* tasks)

***Meta-features: measurable properties of the tasks***

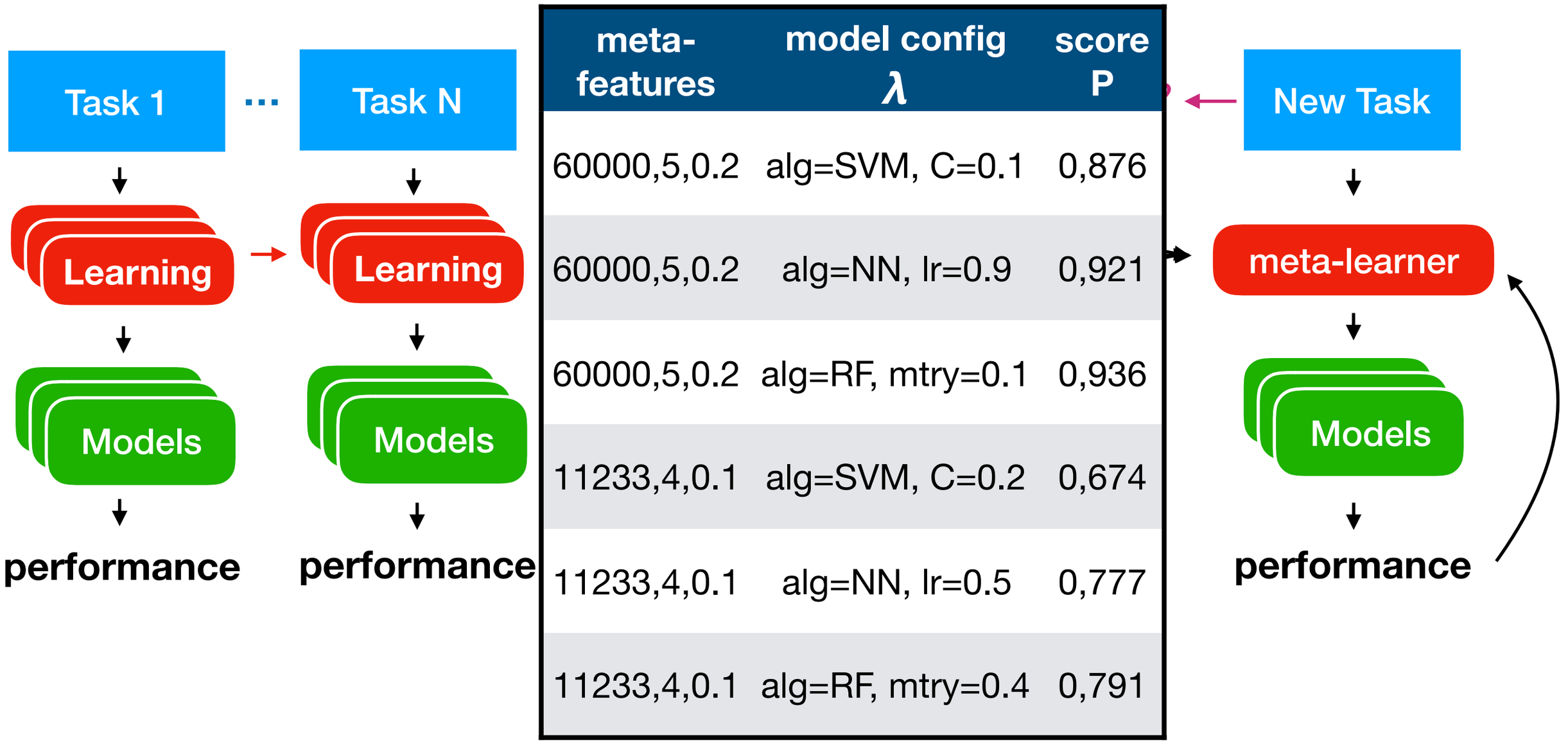
(number of instances and features, class imbalance, feature skewness,…)



# 2. Learn what may likely work (for *partially similar* tasks)

***Meta-features: measurable properties of the tasks***

(number of instances and features, class imbalance, feature skewness,…)



[1] Vanschoren 2018

[2] Rivolli et al 2019

# Meta-features

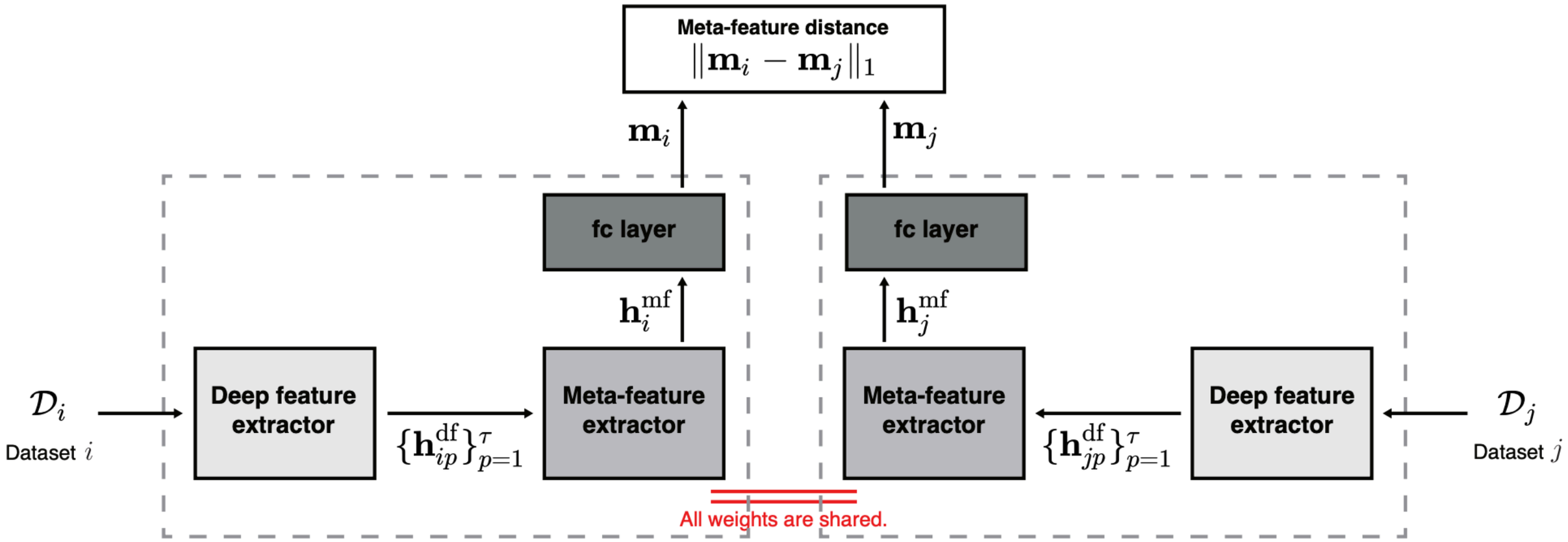
- **Hand-crafted (interpretable) meta-features[1]**
  - **Number of** instances, features, classes, missing values, outliers,…
  - **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,…
  - **Information-theoretic**: class entropy, mutual information, noise-signal ratio,…
  - **Model-based**: properties of simple models trained on the task
  - **Landmarkers**: performance of fast algorithms trained on the task
  - Domain specific task properties
- Optimized (recommended) representations exist [2]

[1] Kim et al. 2017

[2] Achille et al. 2019

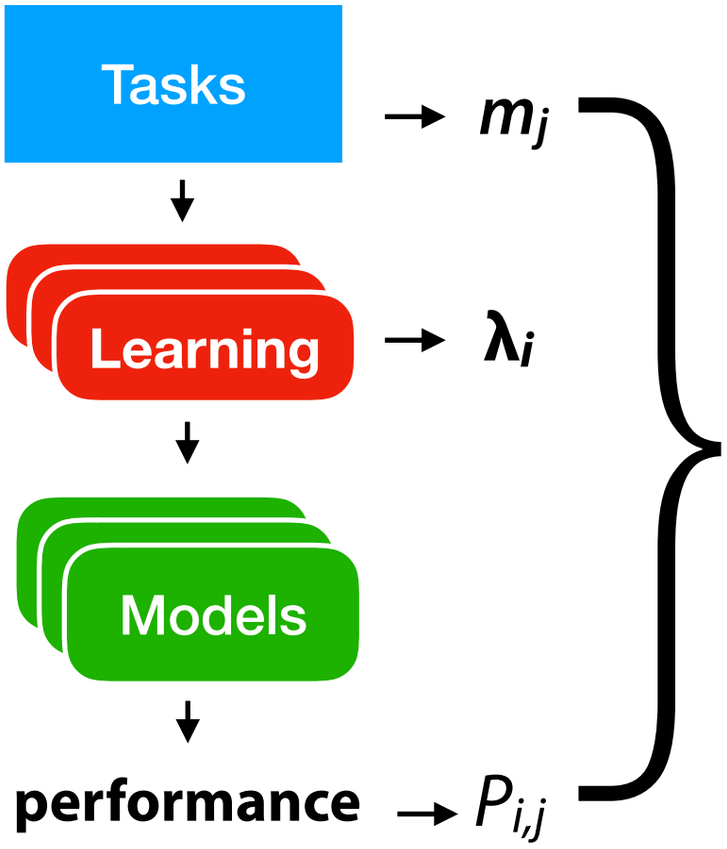
# Meta-features

- ***Learning* a joint task representation**
  - Deep metric learning: learn a representation $h^{mf}$ using ground truth distance
    - Siamese Network: similar task, similar representation [1]
    - Feed through pretrained CNN, extract vector from weights [2]
  - Recommend neural architectures, or warm-start neural architecture search



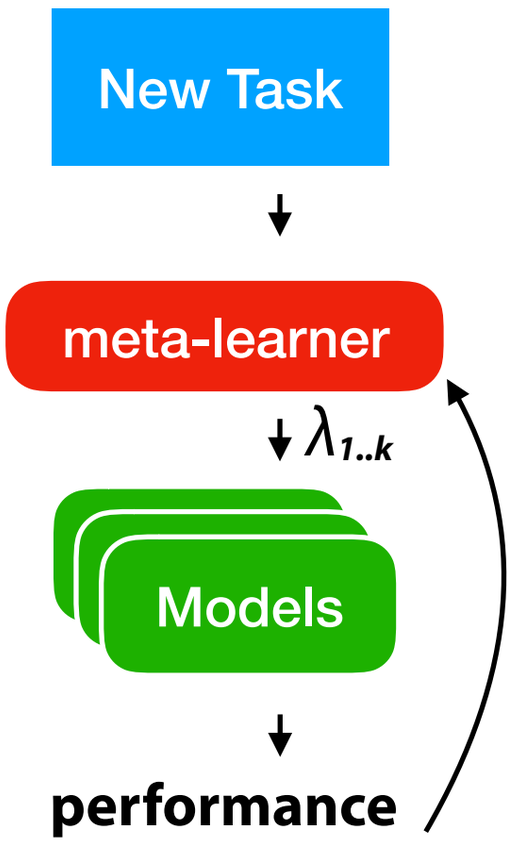
# Warm-starting from similar tasks

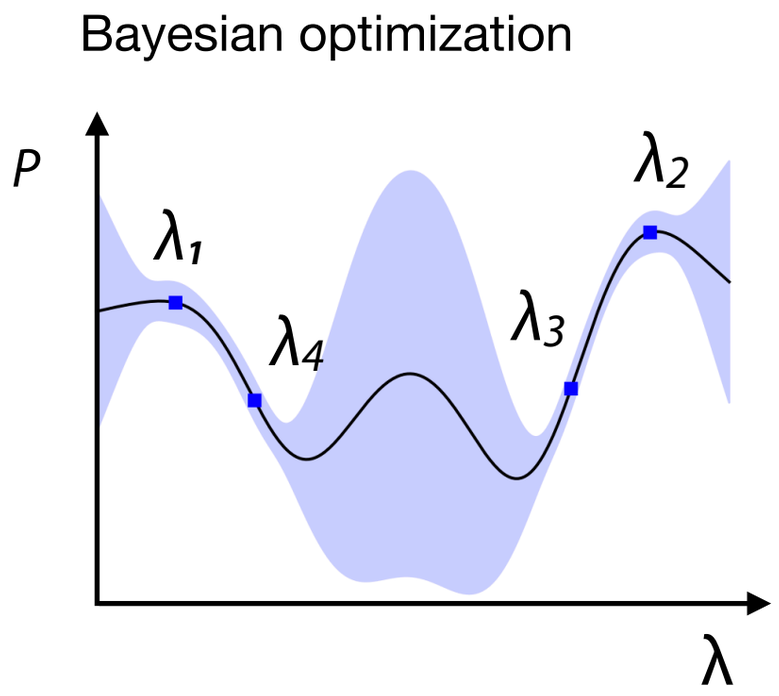
- Find k most similar tasks, warm-start search with best $\lambda_i$
  - Auto-sklearn: Bayesian optimization (SMAC)
    - Meta-learning yield better models, faster
    - Winner of AutoML Challenges




best $\lambda_i$ on similar tasks

$\|\mathbf{m}_i - \mathbf{m}_j\|_1$

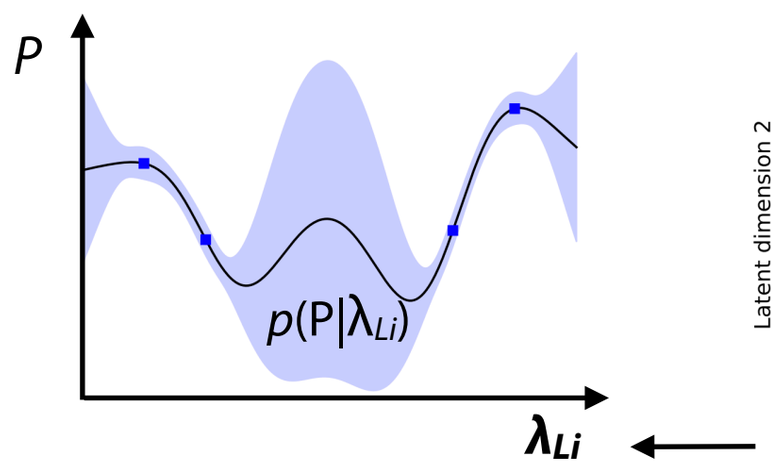


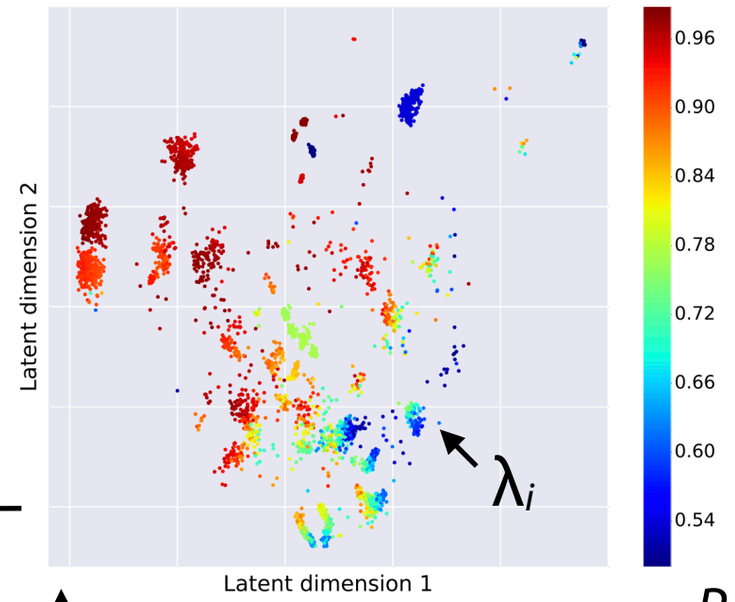



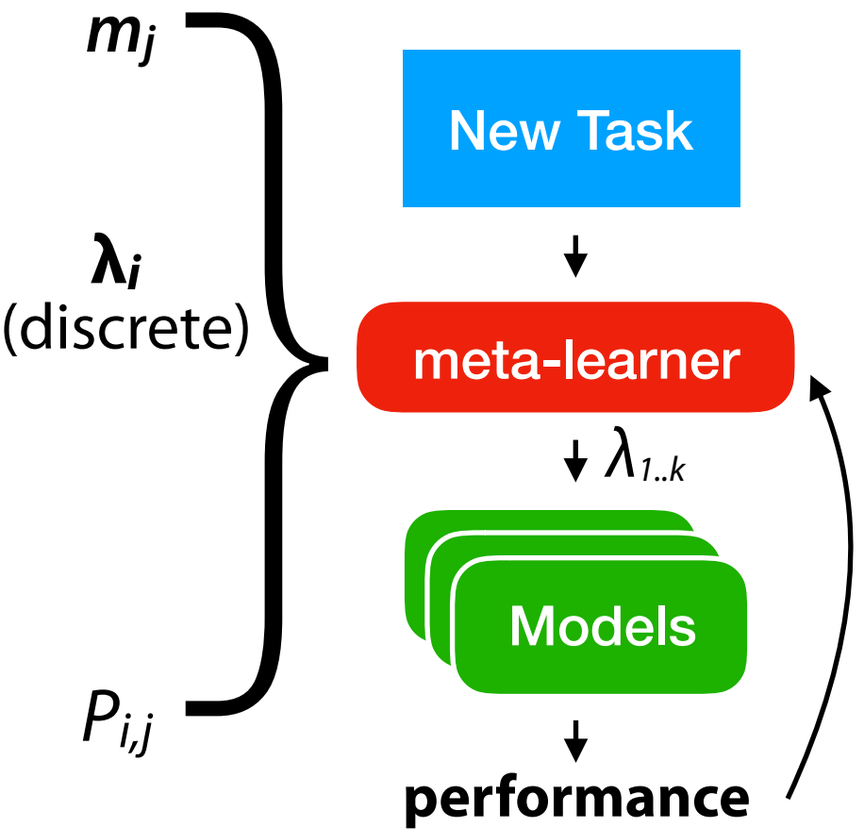
# Warm-starting from similar tasks

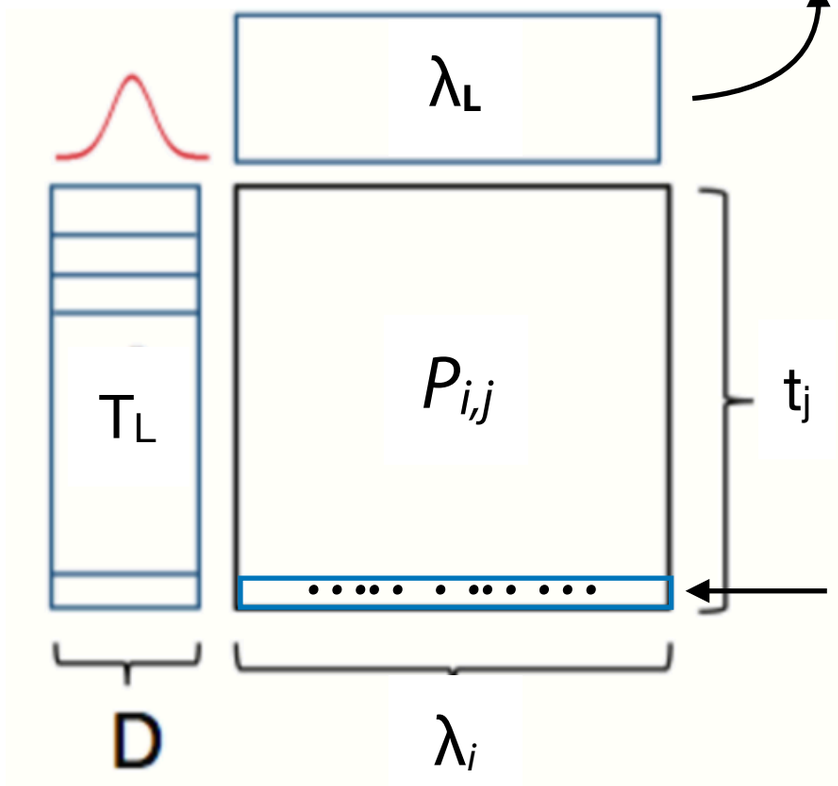
- Collaborative filtering: configurations $\lambda_i$ are `rated' by tasks $t_j$

- Learn latent representation for tasks and configurations
- Use meta-features to warm-start on new task
- Returns probabilistic predictions for BayesOpt



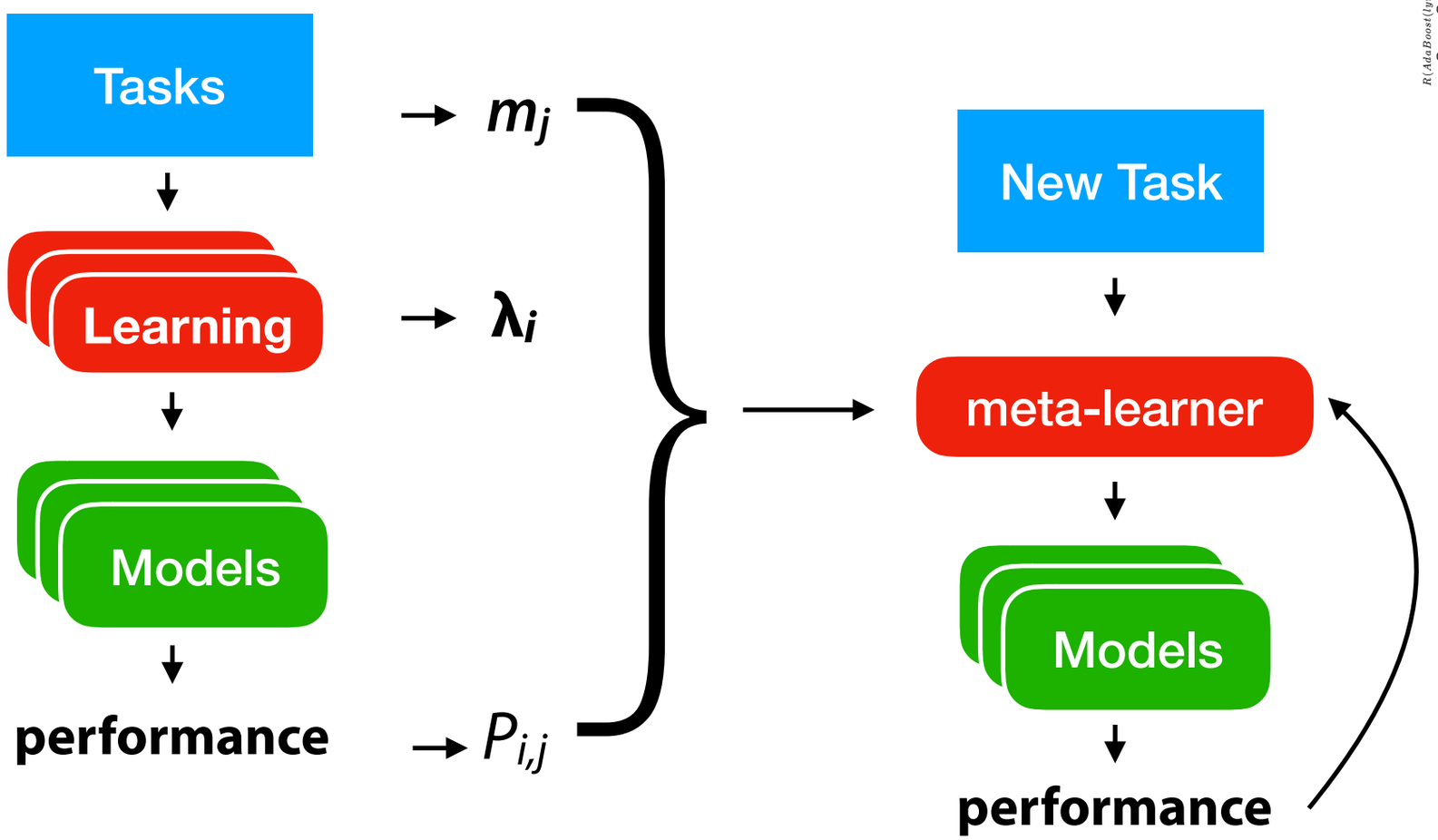

latent representation

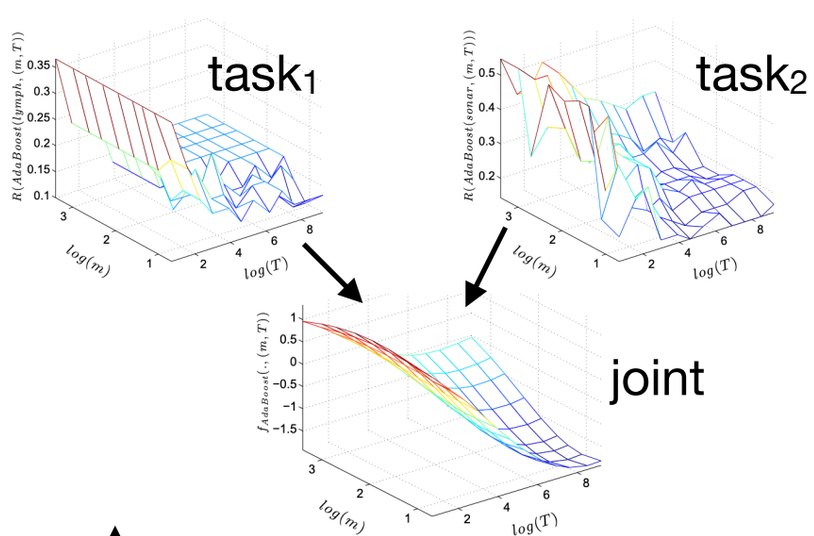
$t_{new}$ warm-started with $\lambda_{1..k}$

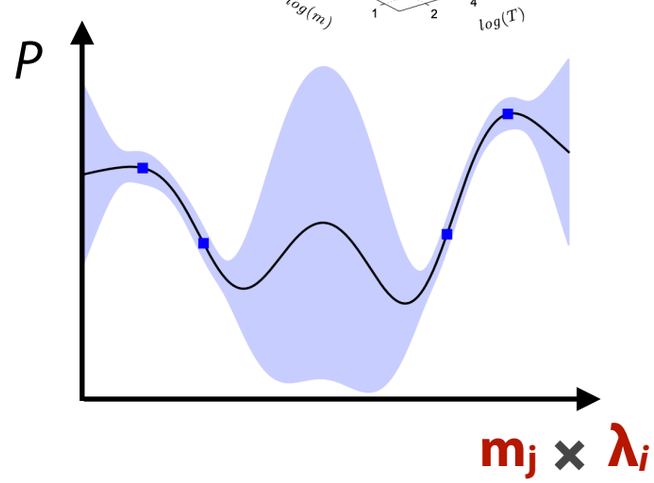
[1] Bardenet et al. 2013
[2] Schilling et al. 2015
[3] Yogatama et al. 2014

# Global surrogate models

- Train a task-independent surrogate model with meta-features in inputs
  - SCOT: Predict *ranking* of $\lambda_i$ with surrogate ranking model + $m_j$. [1]
  - Predict $P_{i,j}$ with multilayer Perceptron surrogates + $m_j$. [2]
  - Build joint GP surrogate model on most similar ($\|\mathbf{m}_i - \mathbf{m}_j\|_2$) tasks. [3]
  - Scalability is often an issue



[1] Brazdil et al. 2009, Lemke et al. 2015

[2] Sun and Pfahringer 2013, Pinto et al. 2017

[3] Sanders and C. Giraud-Carrier 2017

[4] Yang et al. 2018

# Meta-models

- *Learn* direct mapping between meta-features and $P_{i,j}$
  - Zero-shot meta-models: predict best $\lambda_i$ given meta-features [1]


  - Ranking models: return ranking $\lambda_{1..k}$ [2]

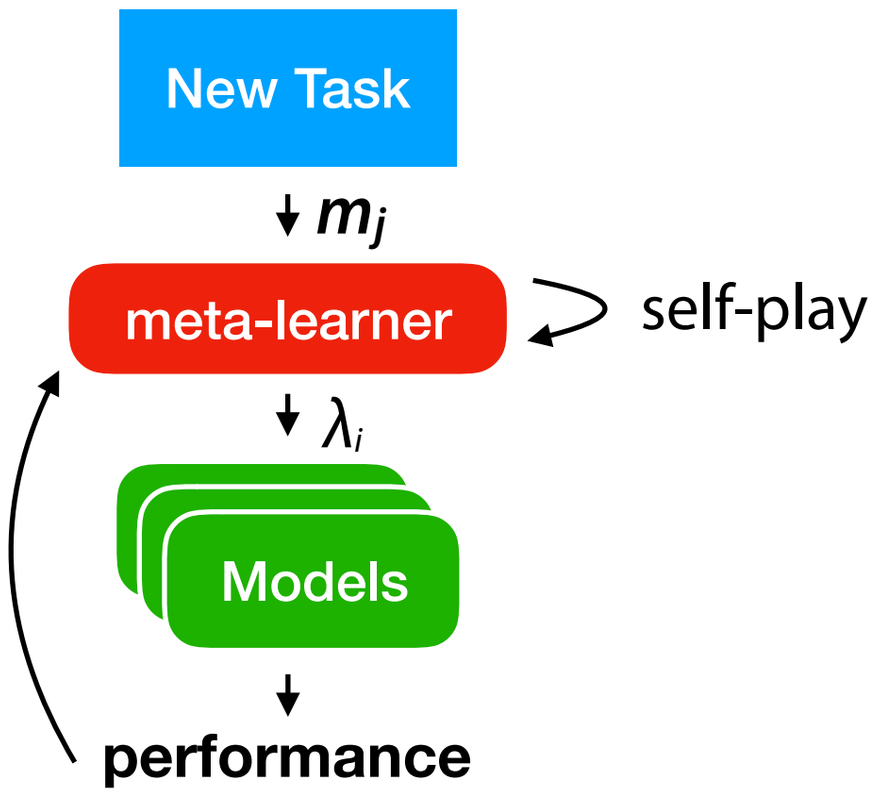

  - Predict which algorithms / configurations to consider / tune [3]

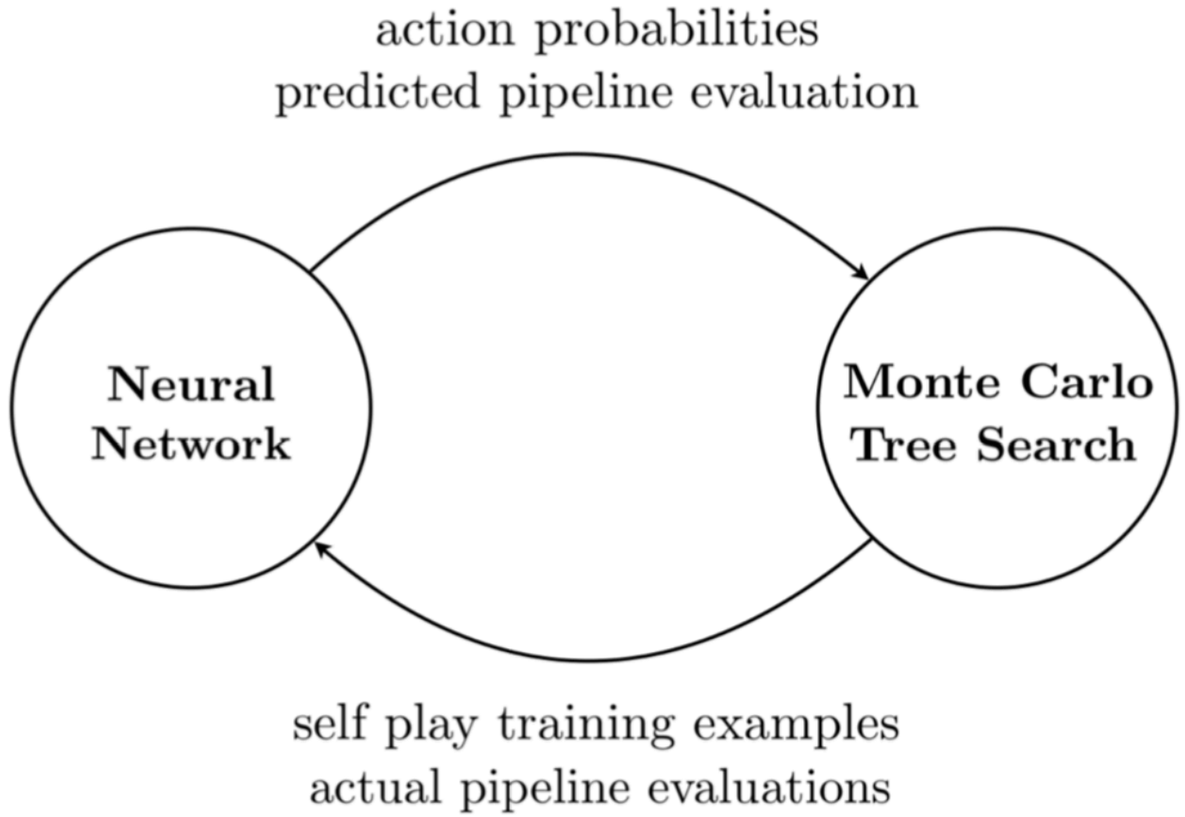

  - Predict performance / runtime for given $\theta_i$ and task [4]


- Can be integrated in larger AutoML systems: warm start, guide search,…

# Learning to learn through self-play

- Build pipelines by inserting, deleting, replacing components (actions)
- Neural network (LSTM) receives task meta-features, pipelines and evaluations
  - Predicts pipeline performance and action probabilities
- Monte Carlo Tree Search builds pipelines, runs simulations against LSTM

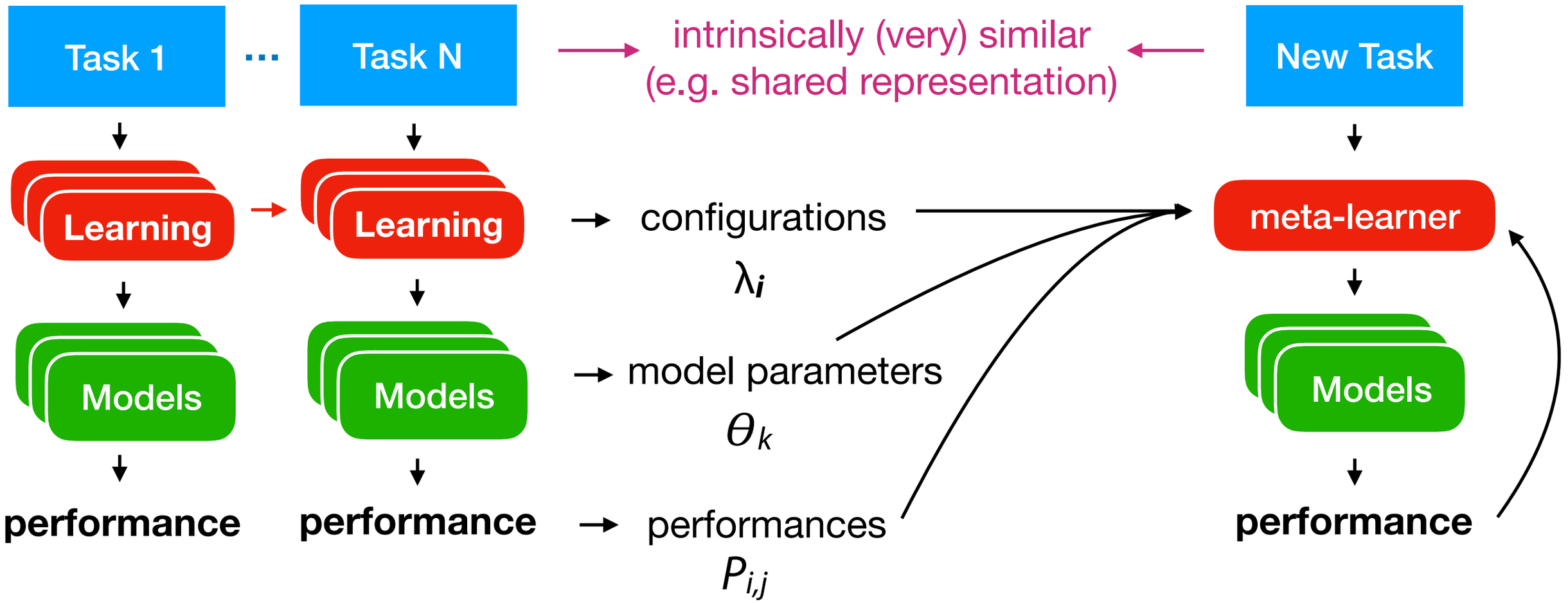


# 3. Learn from previous *models* (for *very similar* tasks)

***Models trained on intrinsically similar tasks***

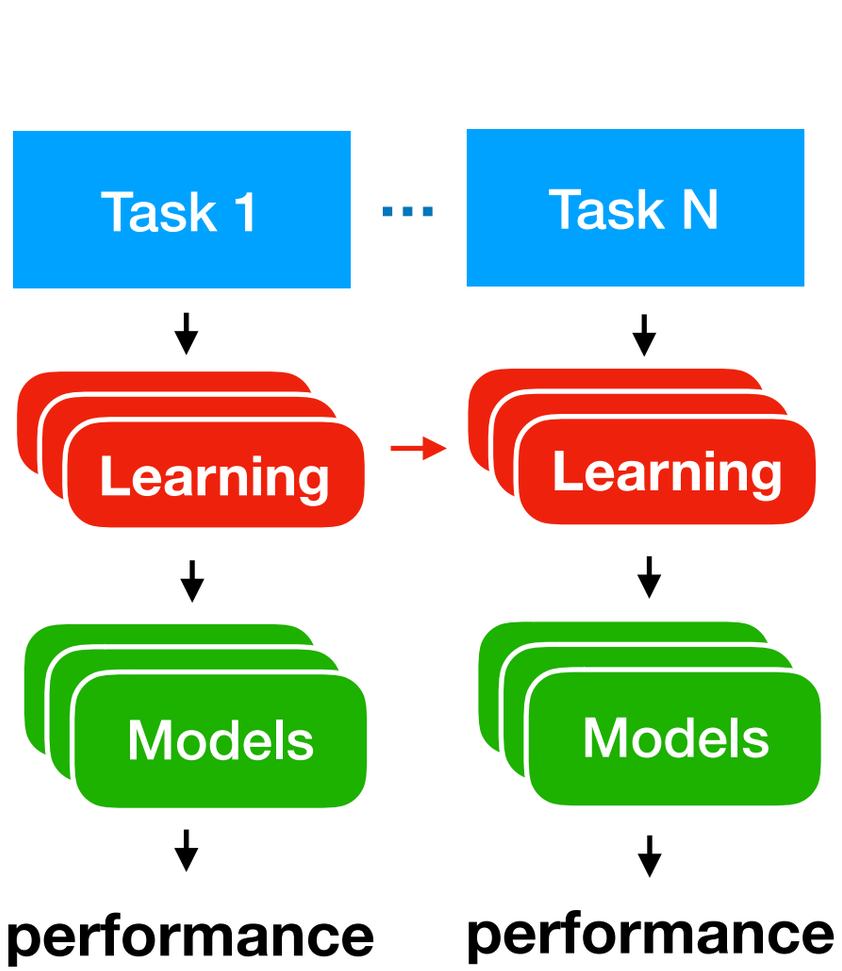
(model parameters, features,…)



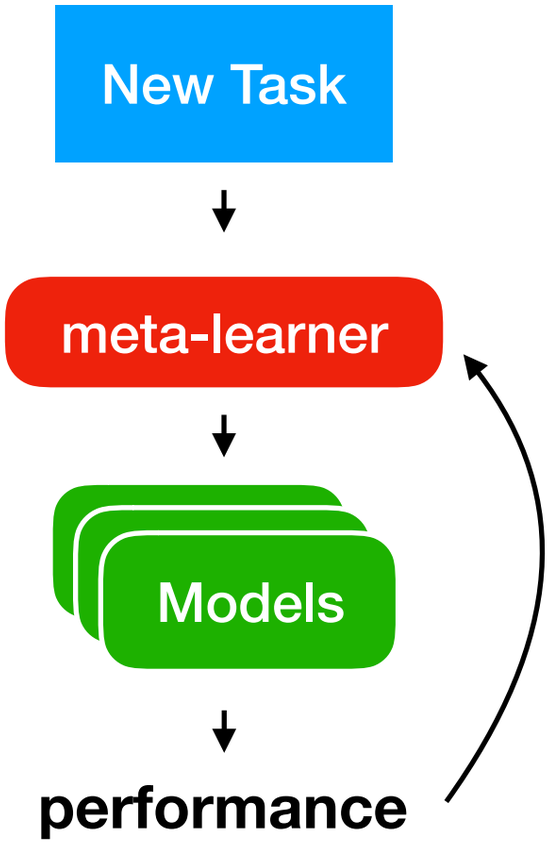
# 3. Learn from previous *models* (for *very similar* tasks)

***Models trained on intrinsically similar tasks***

(model parameters, features,…)




| model param $\theta$ | model config $\lambda$ | score P |
|---|---|---|
| 0.34,0.43,… | alg=NN, lr=0.9 | 0,876 |
| 0.04,0.03,… | alg=NN, lr=0.9 | 0,921 |
| 0.94,0.03,… | alg=NN, lr=0.9 | 0,936 |
| 0.34,0.23,… | alg=NN, lr=0.5 | 0,674 |
| 0.04,0.23,… | alg=NN, lr=0.5 | 0,777 |
| 0.94,0.23,… | alg=NN, lr=0.5 | 0,791 |



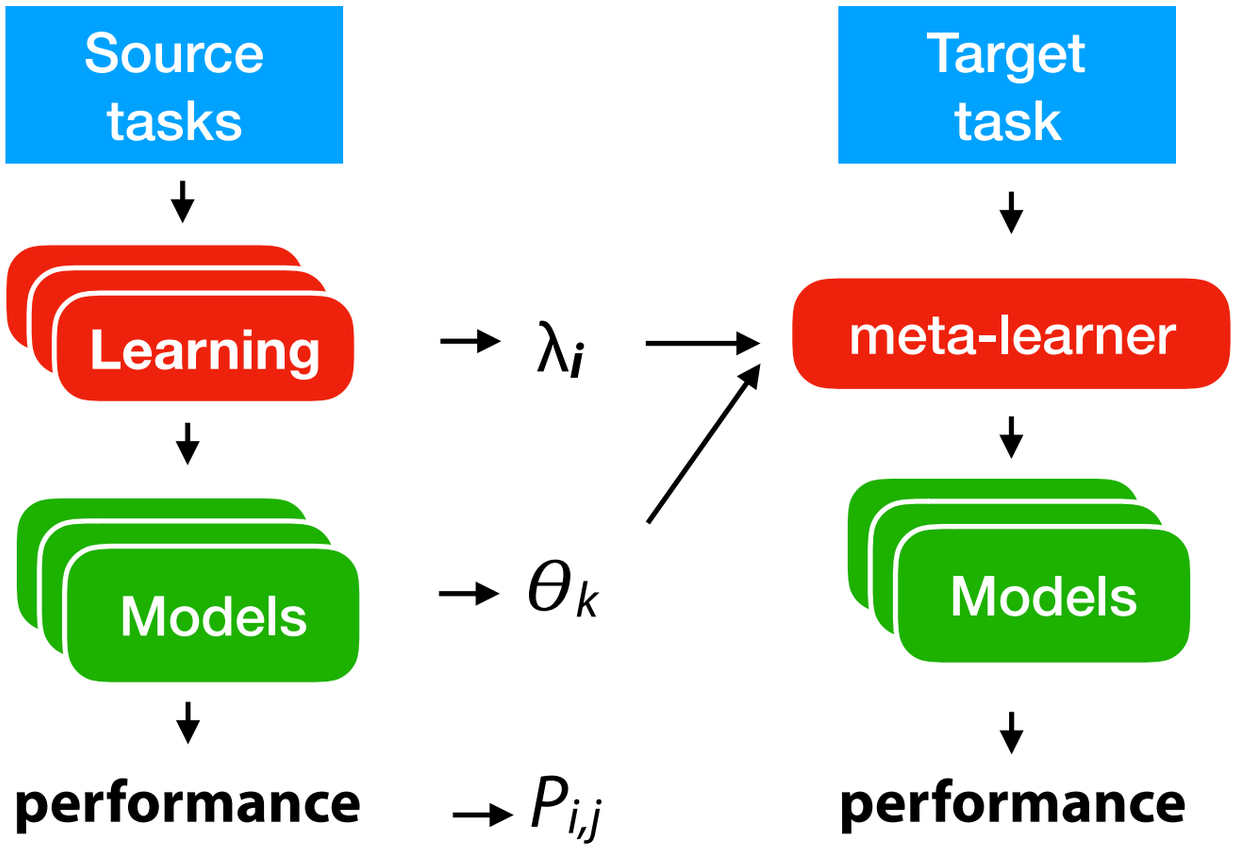
[1] Thrun and Pratt 1998

[2] Niculescu-Mizil and Caruana 2005

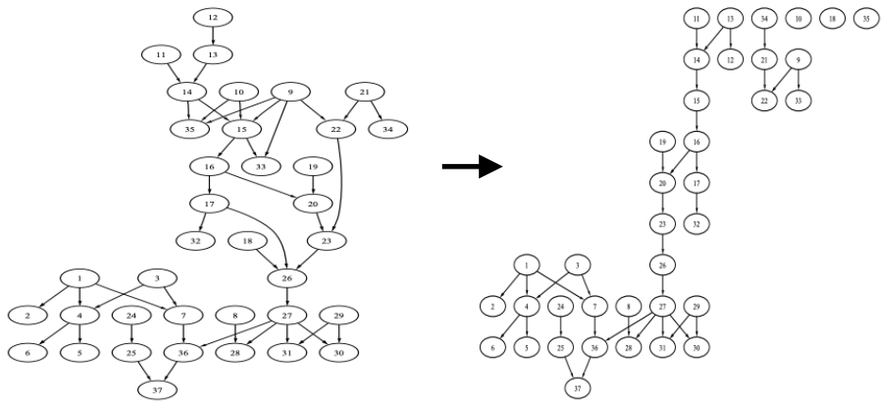
[3] Taylor and Stone 2009

# Transfer Learning

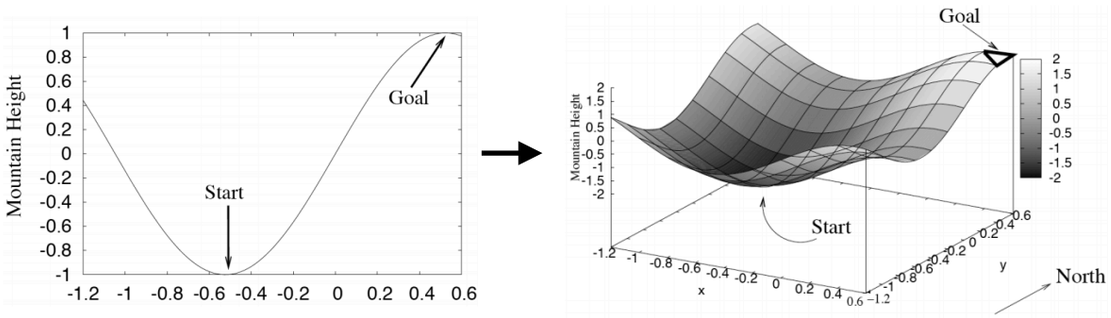
- Select source tasks, transfer trained models to similar target task [1]
- Use as starting point for tuning, or *freeze* certain aspects (e.g. structure)
  - Bayesian networks: start structure search from prior model [2]
  - Reinforcement learning: start policy search from prior policy [3]




Bayesian Network transfer



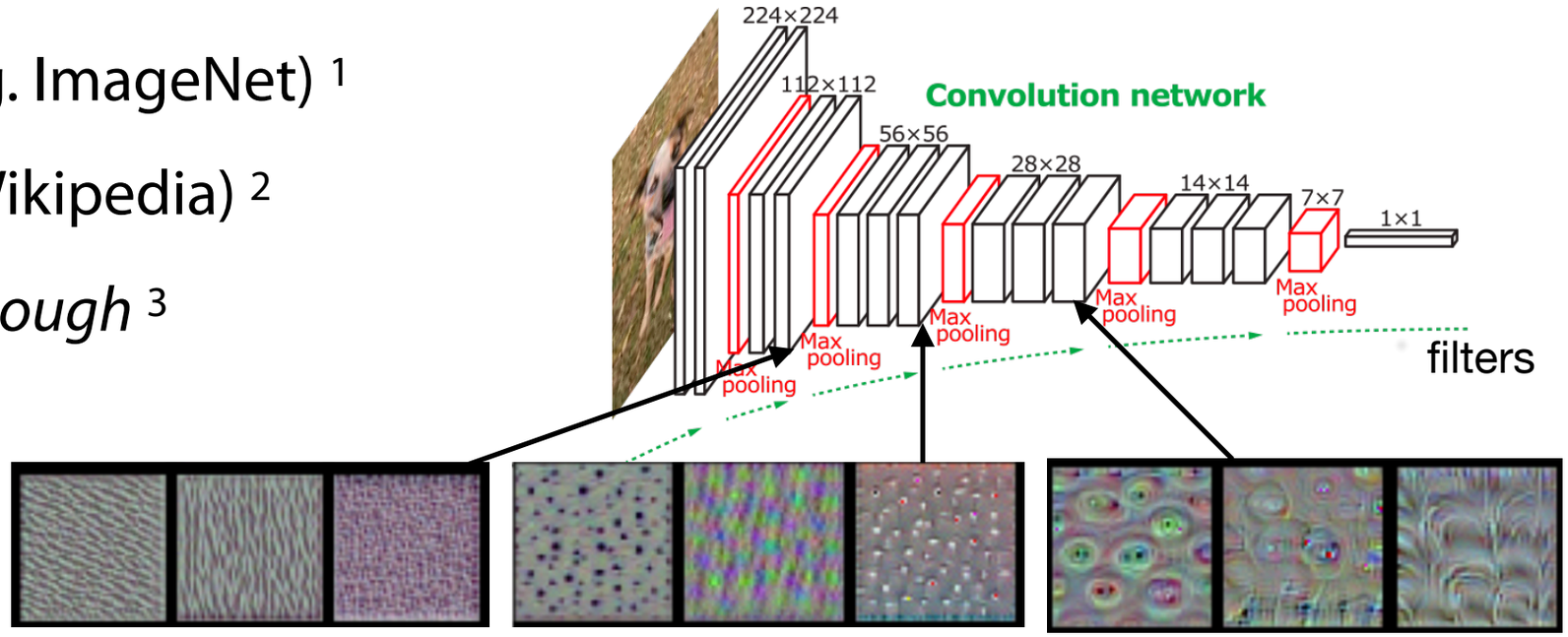
Reinforcement learning: 2D to 3D mountain car

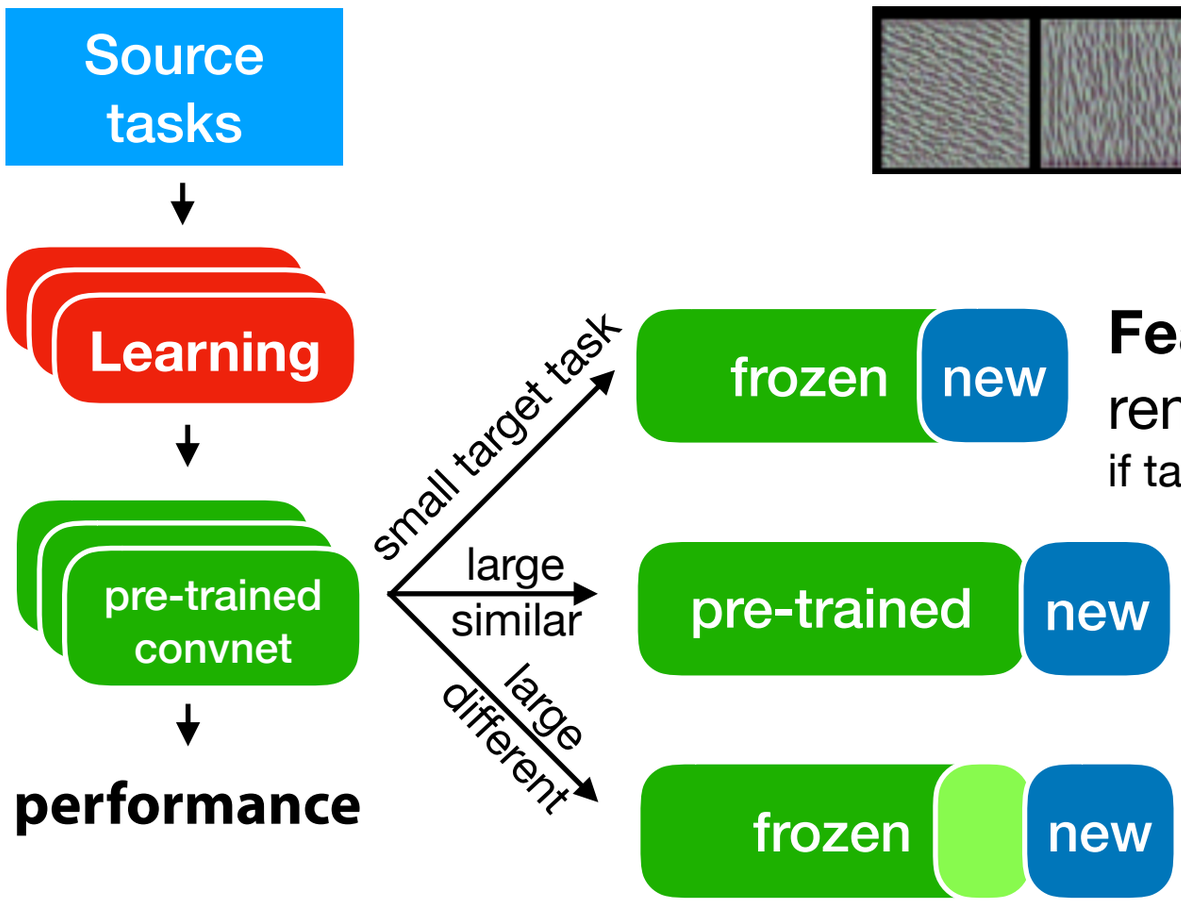


[1] Razavian et al. 2014
[2] Mikolov et al. 2013
[3] Yosinski et al. 2014

# Transfer learning

- For neural networks, both structure and weights can be transferred
- Features and initializations learned from:
  - Large image datasets (e.g. ImageNet) [1]
  - Large text corpora (e.g. Wikipedia) [2]
- Fails if tasks are *not similar enough* [3]




**Feature extraction:**
remove last layers, use output as features
if task is quite different, remove more layers

**End-to-end tuning:**
train from initialized weights

**Fine-tuning:**
unfreeze last layers, tune on new task

# Learning to learn Neural networks

- End-to-end differentiable models afford many meta-learning opportunities
  - Transfer learning
  - Learning gradient descent procedures (weight update rules)
  - Few shot learning
  - Model-agnostic meta-learning (learning weight initializations)
  - Learning to reinforcement learn
  - …

See part 3

# *Thank you!*

## *Never stop learning*

Part 3: *Deeper* into AutoML and meta-learning