NeurIPS 2018 Tutorial on Automatic Machine Learning

Learning to *Learn*

automl.org/events -> AutoML Tutorial -> Slides

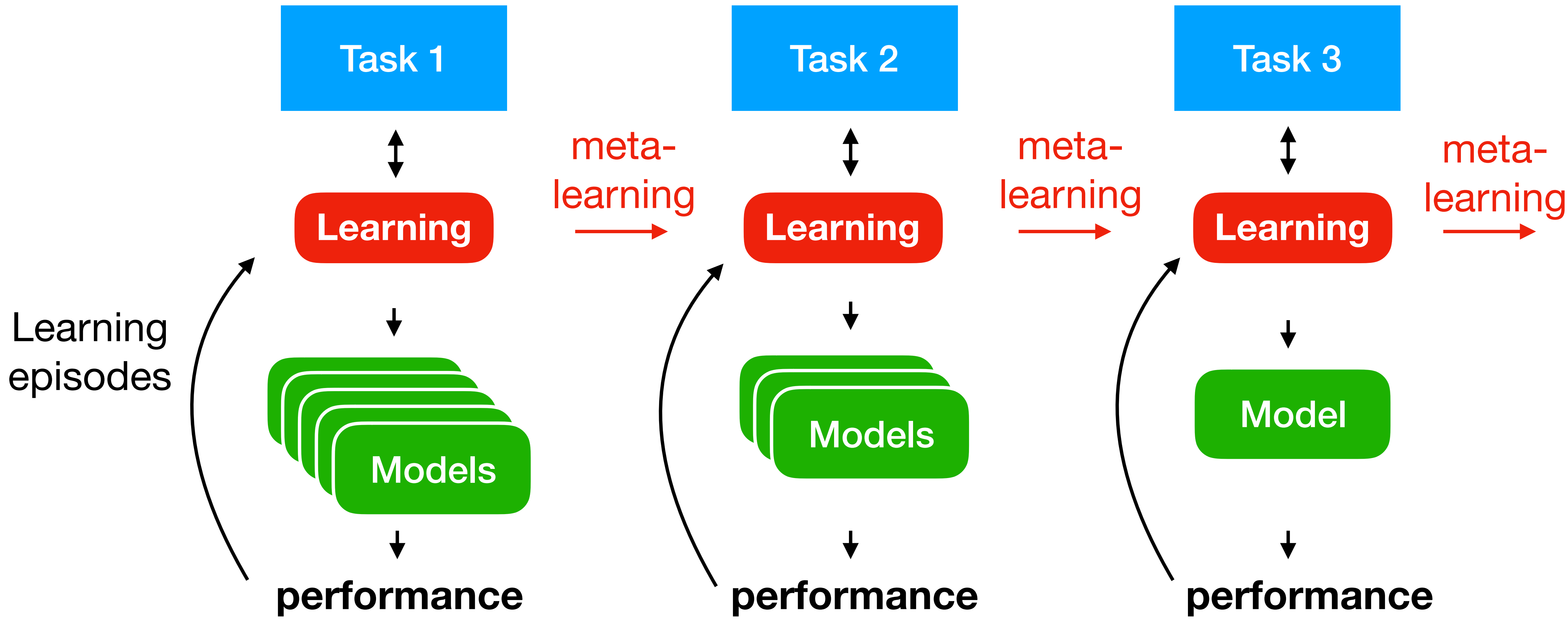
Frank Hutter
University of Freiburg
fh@cs.uni-freiburg.de

Joaquin Vanschoren
Eindhoven University of Technology
j.vanschoren@tue.nl
@joavanschoren

Learning is a never-ending process

Tasks come and go, but learning is forever

Learn more effectively: less trial-and-error, less data

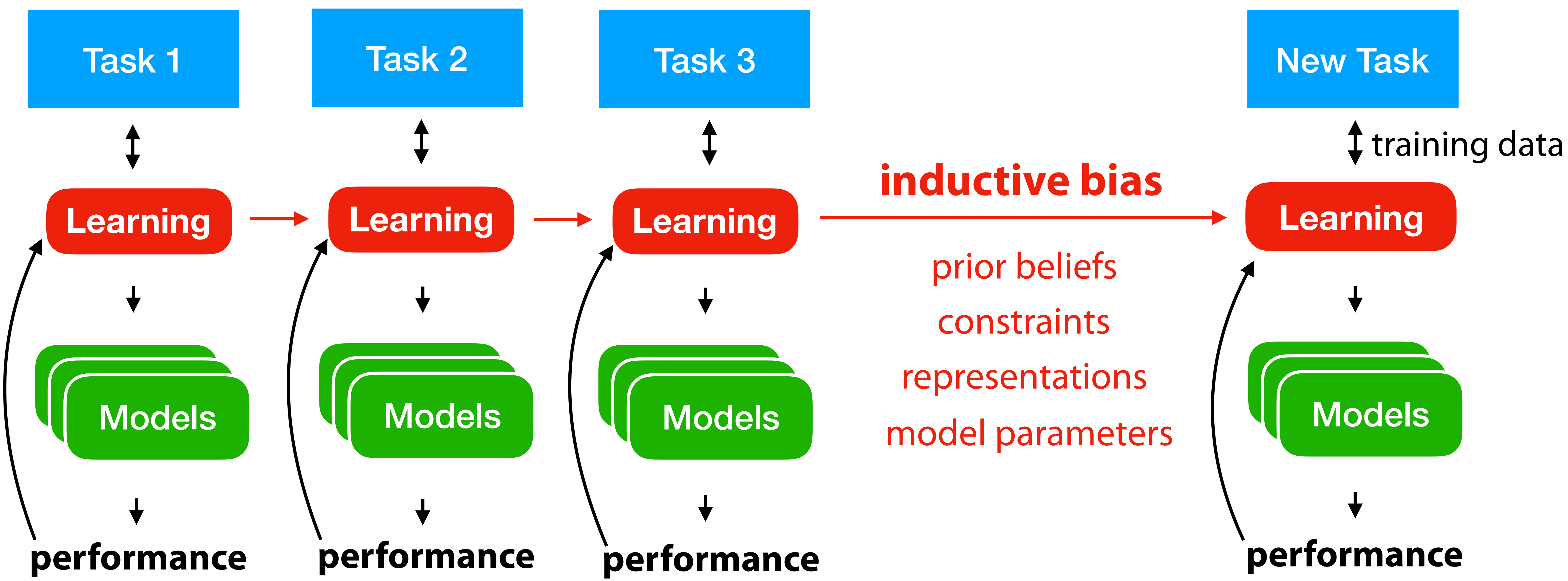


Learning to learn

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

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

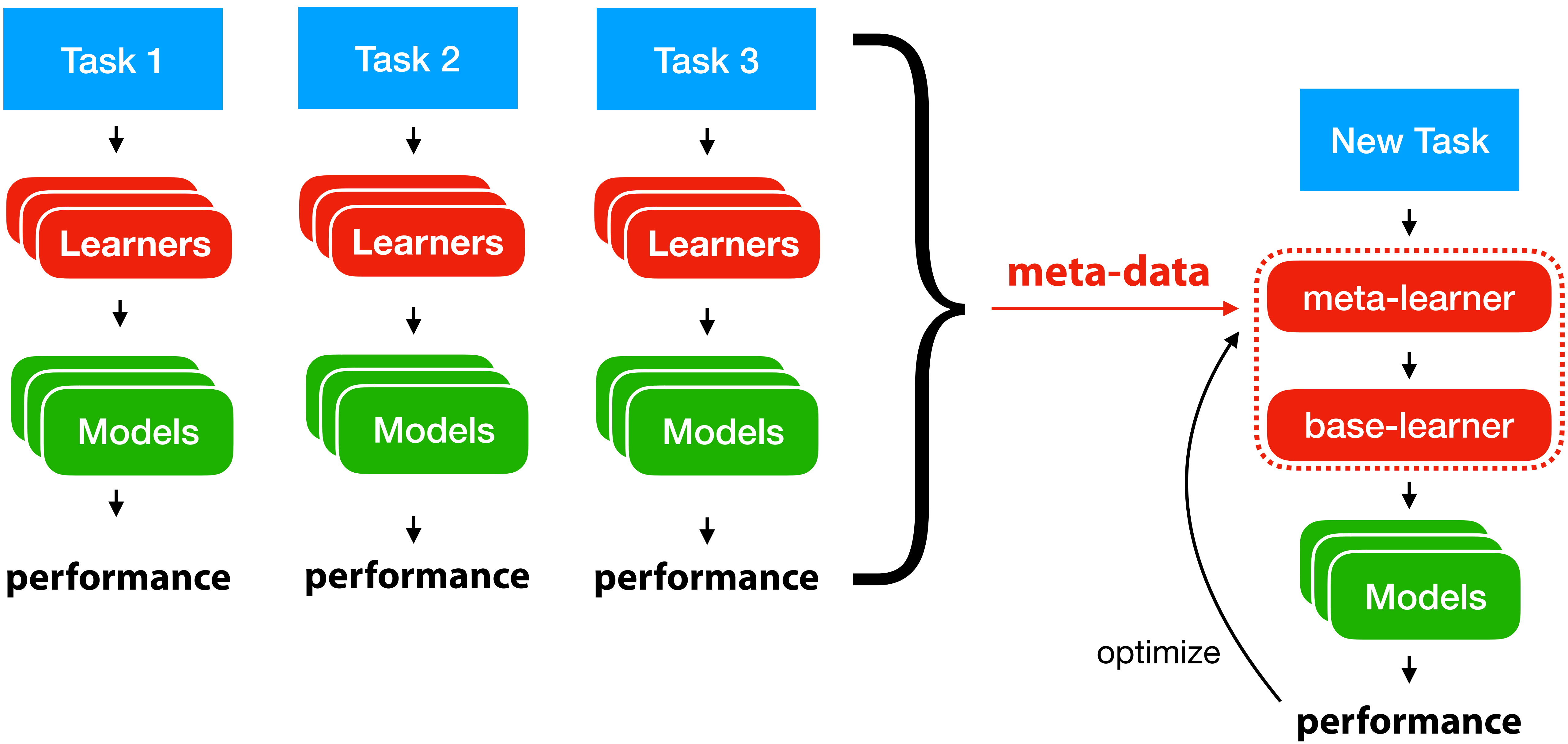
(if not it may actually harm learning)



Meta-learning

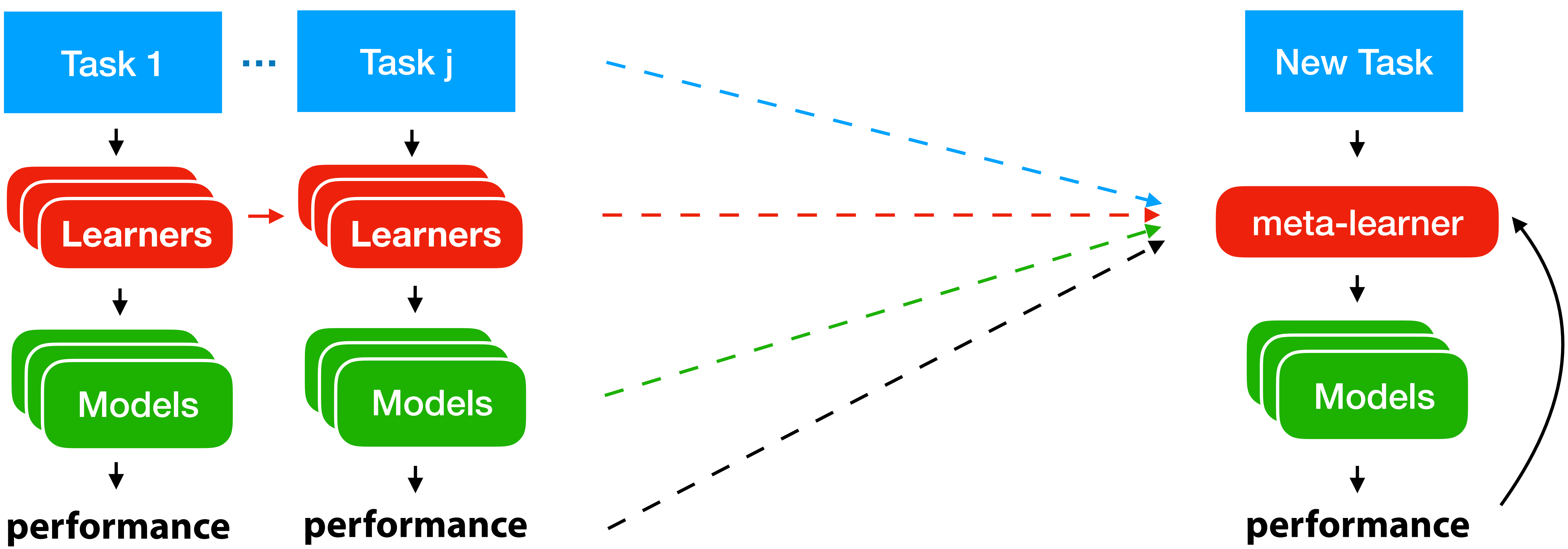
Collect meta-data about learning episodes and learn from them

Meta-learner *learns* a (base-)learning algorithm, *end-to-end*



Three approaches for increasingly similar tasks

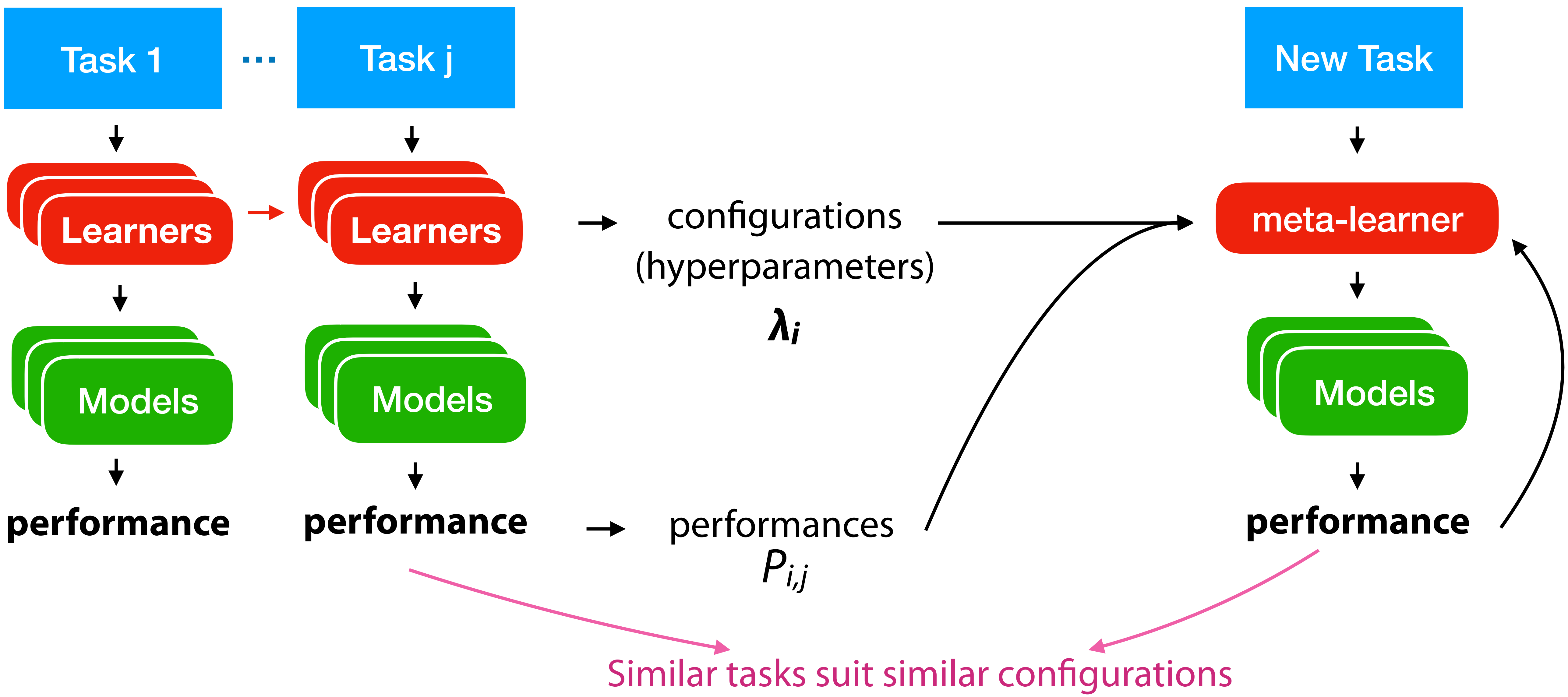
1. Transfer prior knowledge about what generally works well
2. Reason about model performance across tasks
3. Start from models trained earlier on similar tasks



1. Learning from prior evaluations

Configurations: settings that uniquely define the model

(algorithm, pipeline, neural architecture, hyper-parameters, …)

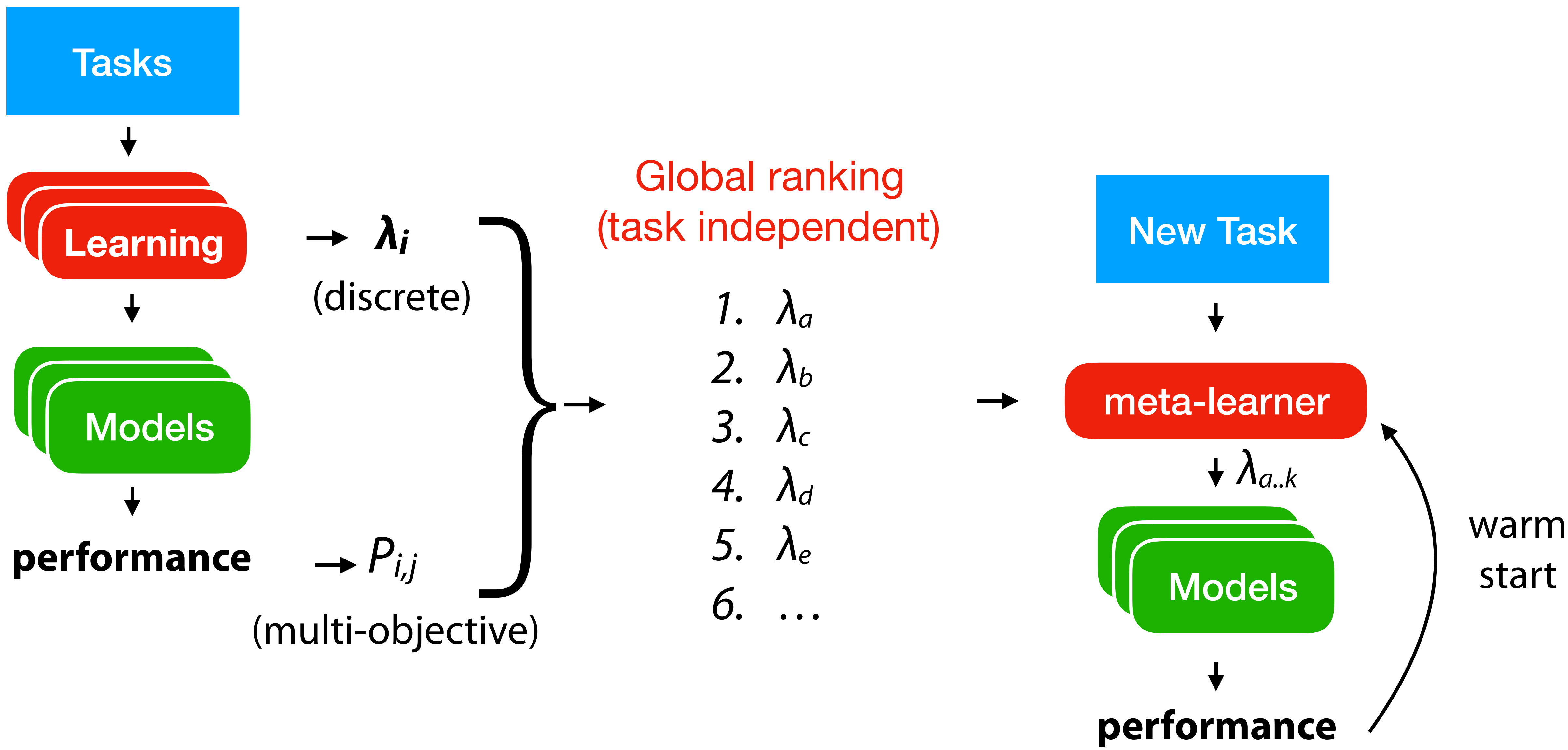


Leite et al. 2012

Abdulrahman et al. 2018

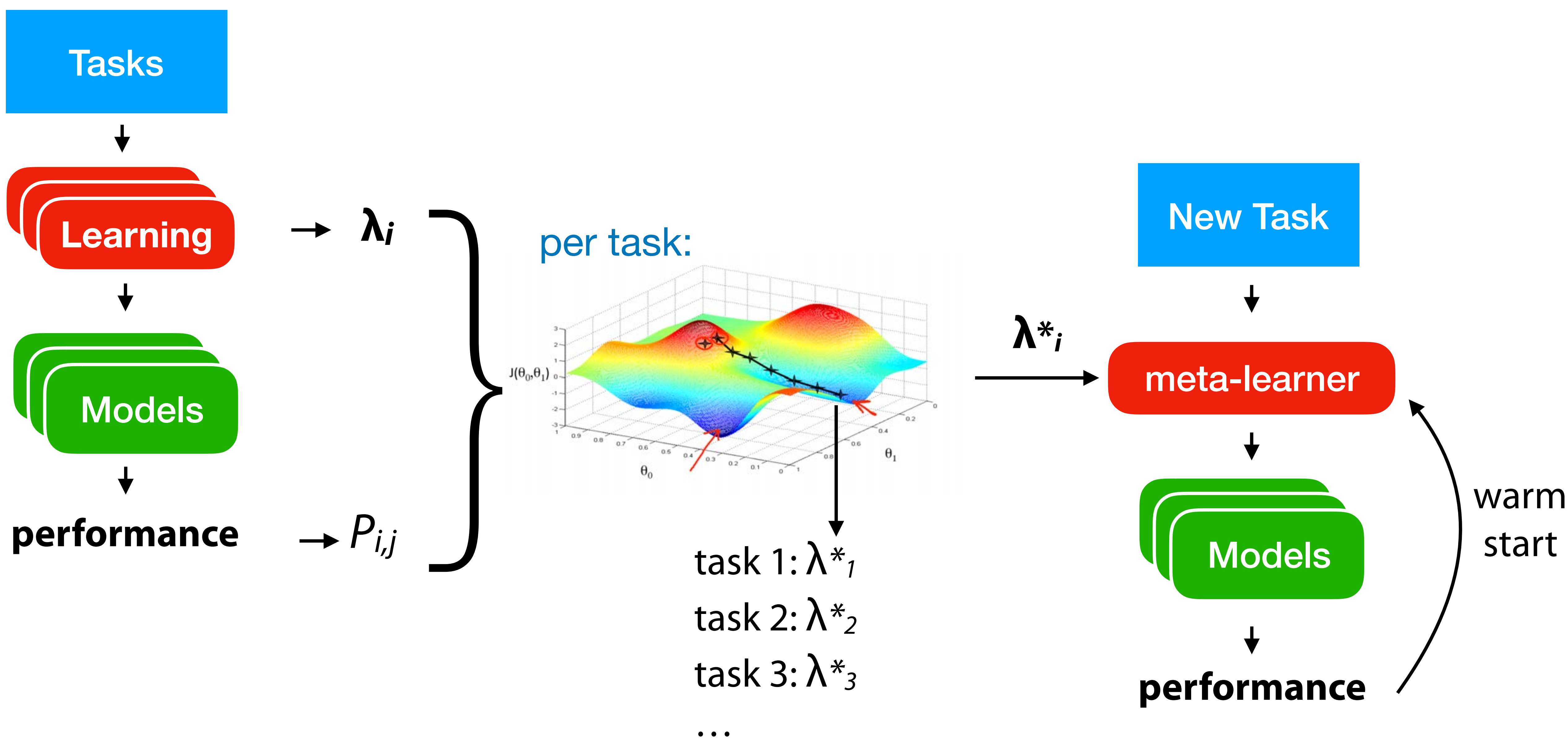
Top-K recommendation

- Build a *global (multi-objective) ranking,* recommend the top-K
- Requires fixed selection of candidate configurations *(portfolio)*
- Can be used as a warm start for optimization techniques



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



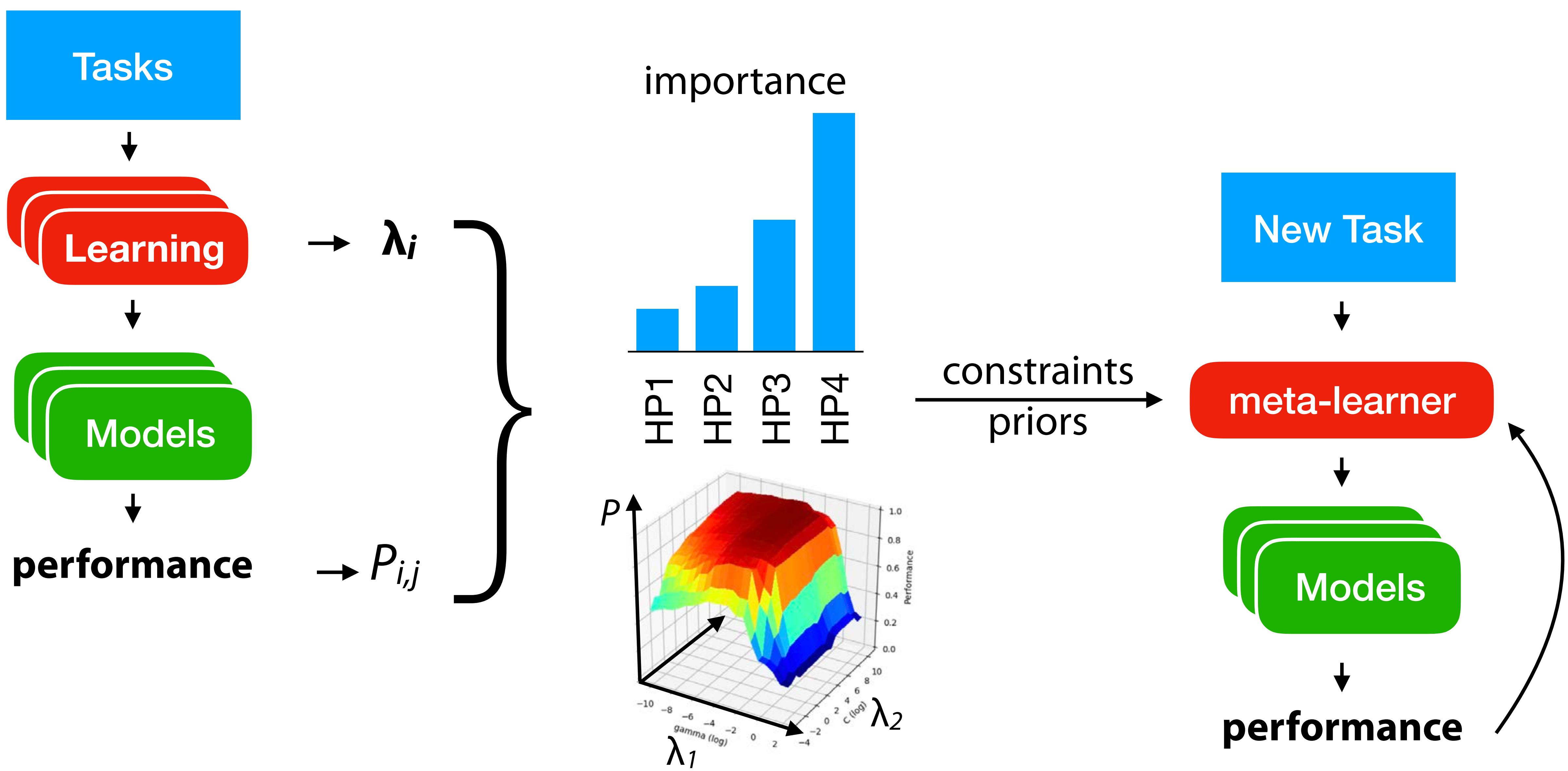
[1] van Rijn & Hutter 2018

[2] Probst et al. 2018

[3] Wistuba et al. 2015

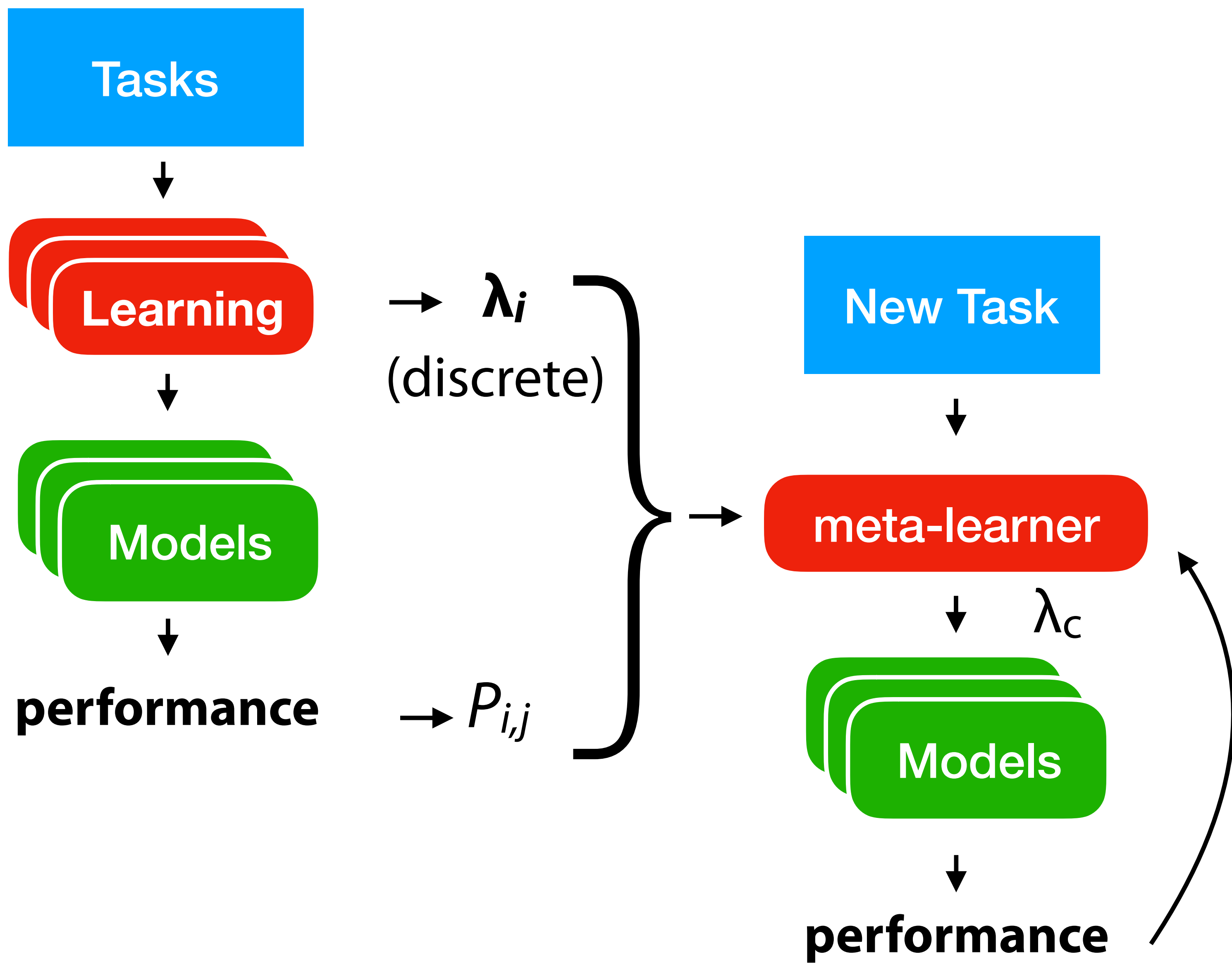
Configuration space design

- **Functional ANOVA**: select hyperparameters that cause variance in the evaluations[1]
- **Tunability**: improvement from tuning a hyperparameter vs. using a good default[2]
- **Search space pruning**: exclude regions yielding bad performance on similar tasks[3]



Active testing

- **Task are *similar*** if observed *relative performance* of configurations is similar
- Tournament-style selection, warm-start with overall best configurations λ_{best}
- Next candidate λ_c : the one that beats current λ_{best} on similar tasks (from portfolio)



Relative landmark on λ_a, λ_b, task t_j:

$$RL_{a,b,j} = P_{a,j} - P_{b,j}$$

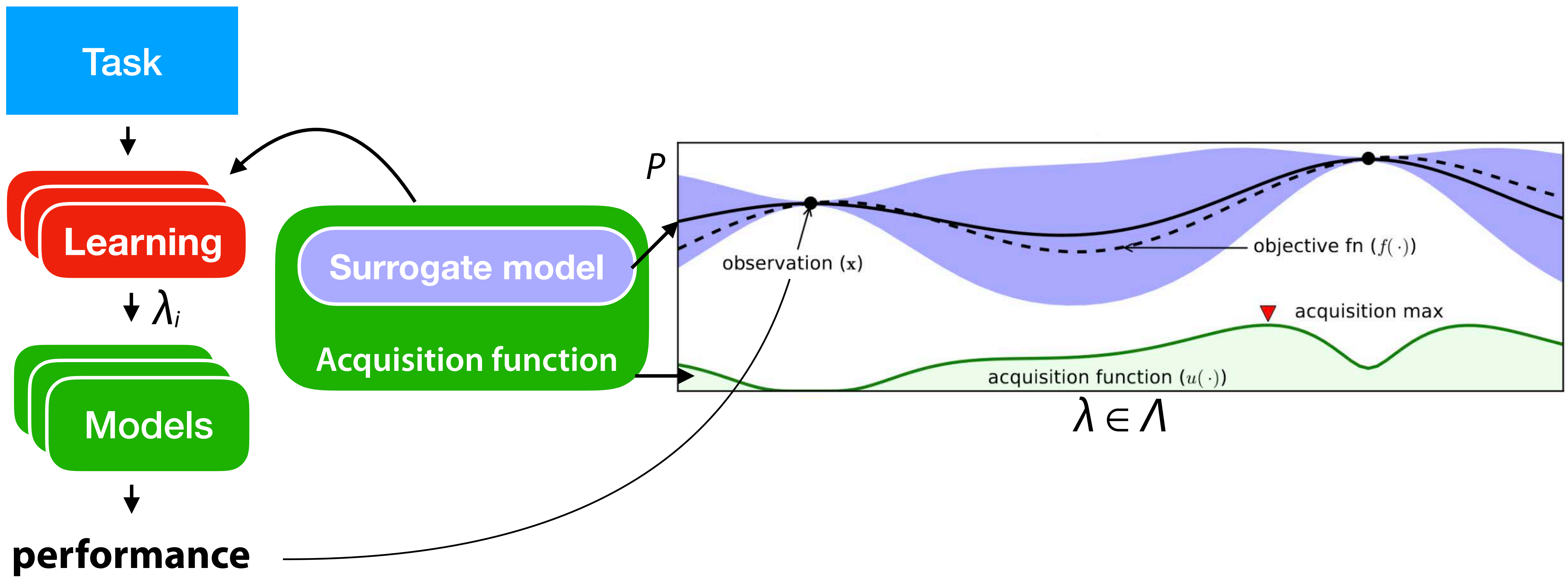
Update:

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

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

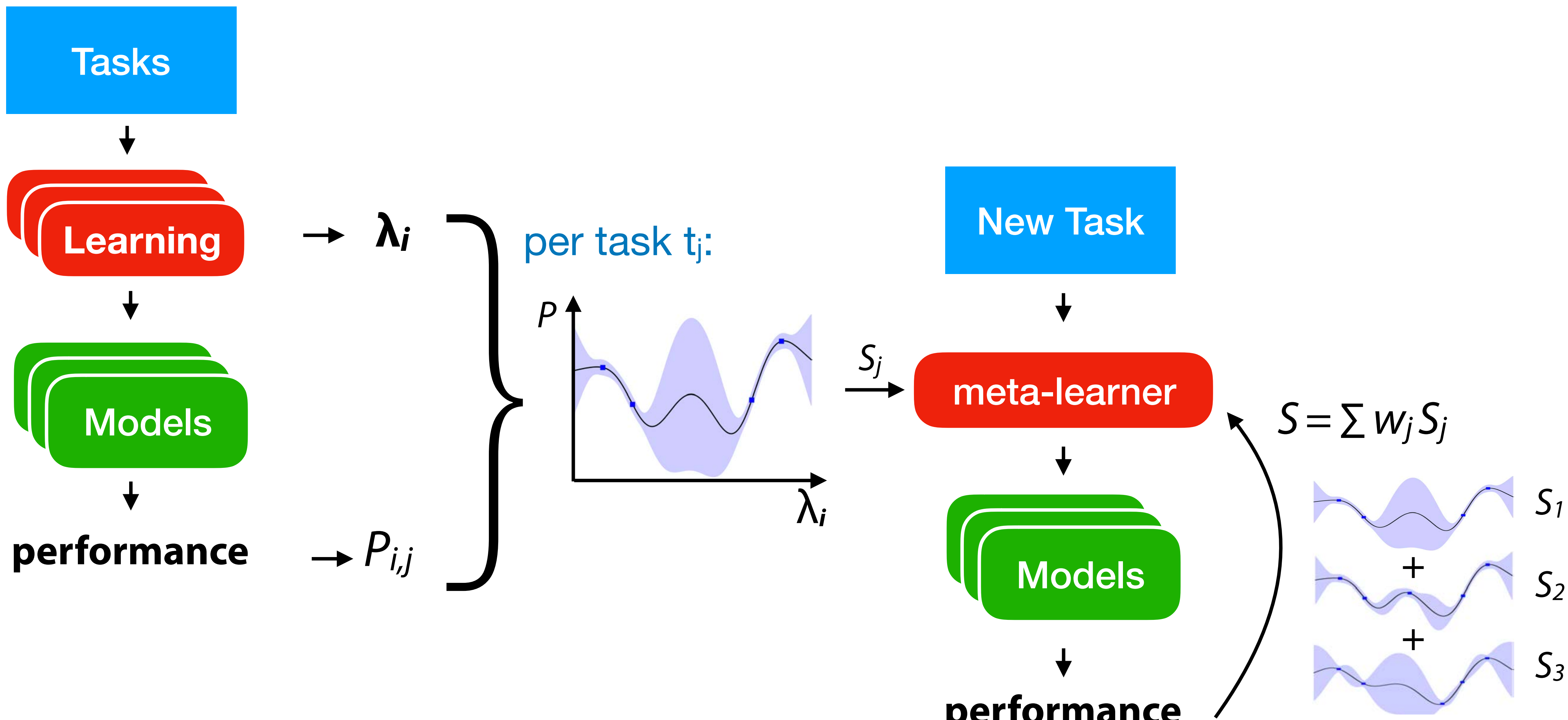
Bayesian optimization (refresh)

- Learns how to learn within a single task (short-term memory)
- Surrogate model: *probabilistic* regression model of configuration performance
- **Can we transfer what we learned to *new* tasks (long term memory)?**



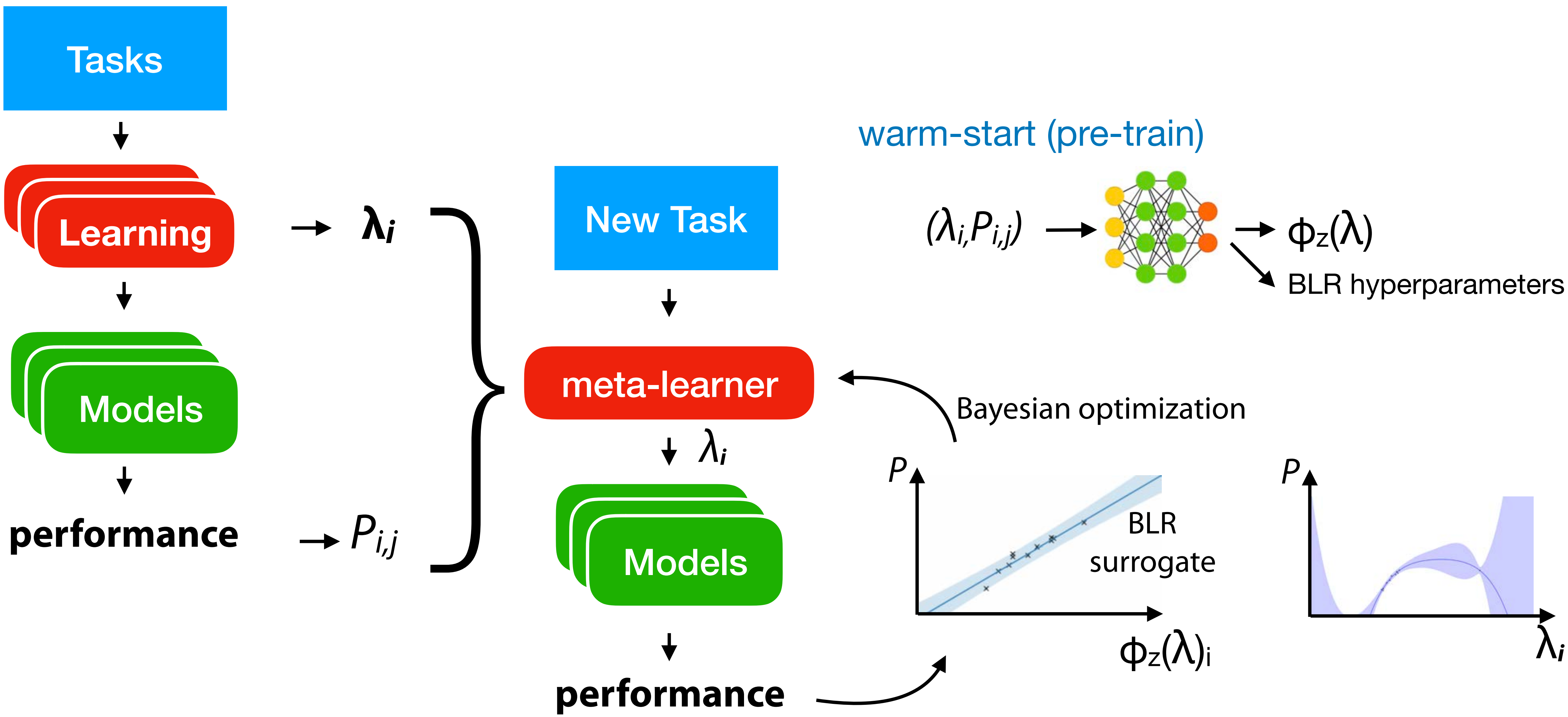
Surrogate model transfer

- If task j is *similar* to the new task, its surrogate model S_j will do well
- Sum up all S_j predictions, weighted by task similarity (relative landmarks)[1]
- Build combined Gaussian process, weighted by current performance on new task[2]



Warm-started multi-task learning

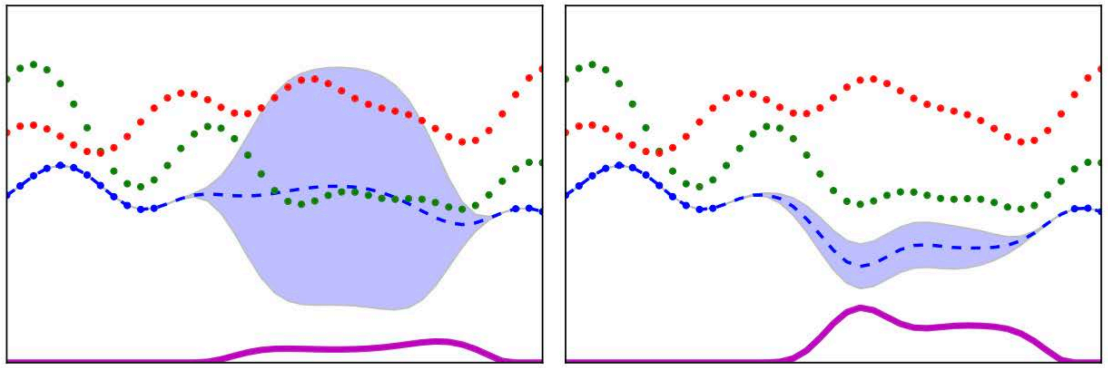
- Bayesian linear regression (BLR) surrogate model on every task
- Learn a suitable basis expansion $\phi_z(\lambda)$, joint representation for all tasks
- Scales linearly in # observations, transfers info on configuration space



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

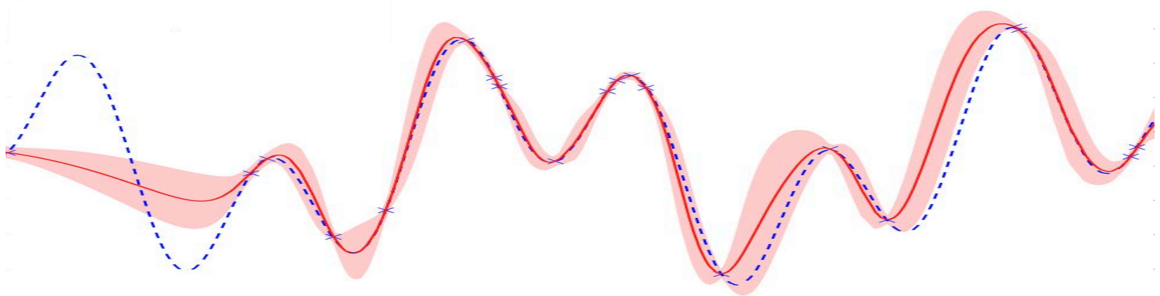
Multi-task Bayesian optimization

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

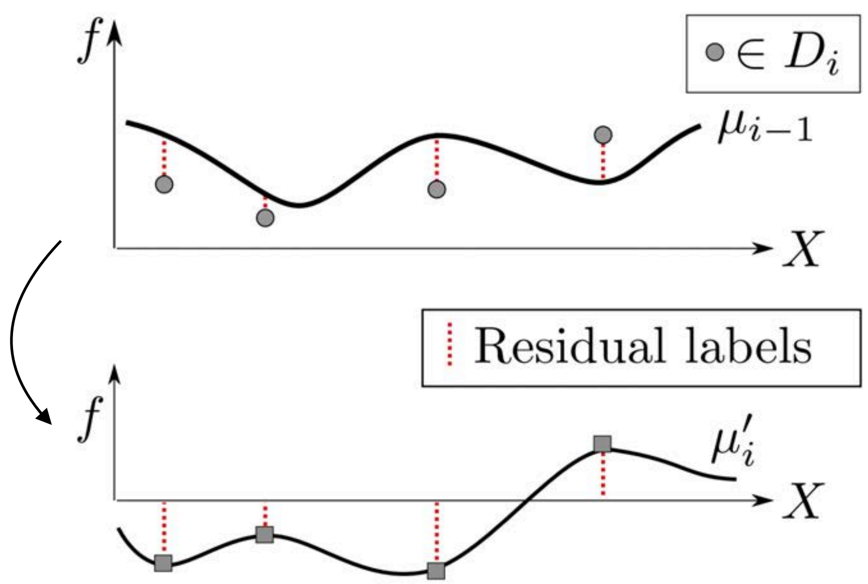


Independent GP predictions   Multi-task GP predictions

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

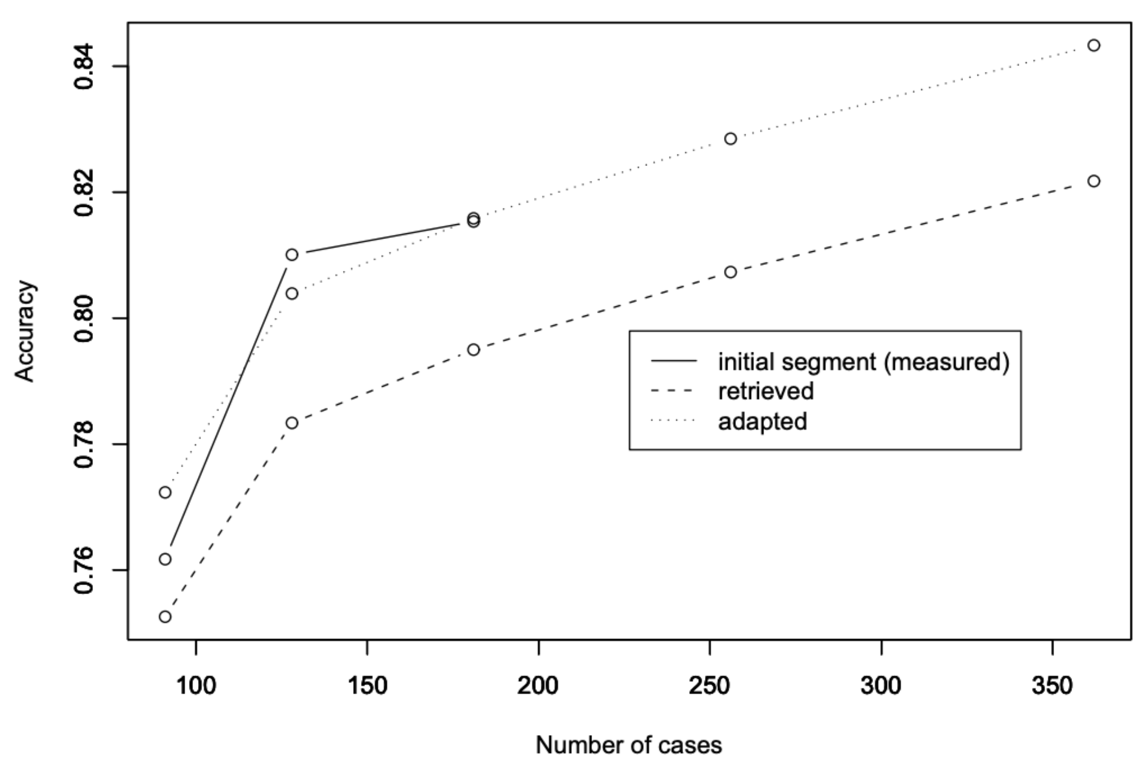


- **Stacking** Gaussian Process regressors (Google Vizier)[3]
  - Sequential tasks, each similar to the previous one
  - Transfers a prior based on residuals of previous GP



Other techniques

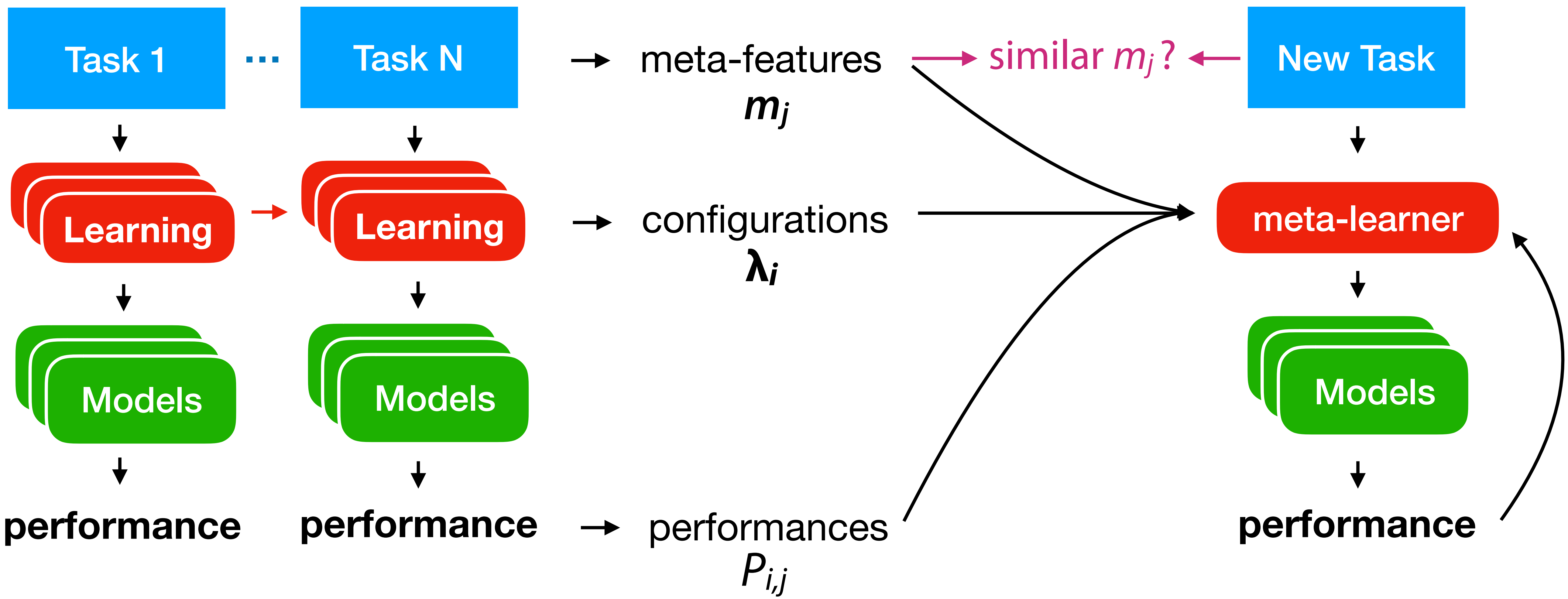
- Transfer learning with multi-armed bandits[1]
  - View every task as an arm, learn to `pull` observations from the most similar tasks
  - Reward: accuracy of configurations recommended based on these observations
- Transfer learning curves[2,3]
  - Learn a partial learning curve on a new task, find best matching earlier curves
  - Predict the most promising configurations based on earlier curves



2. Reason about model performance across tasks

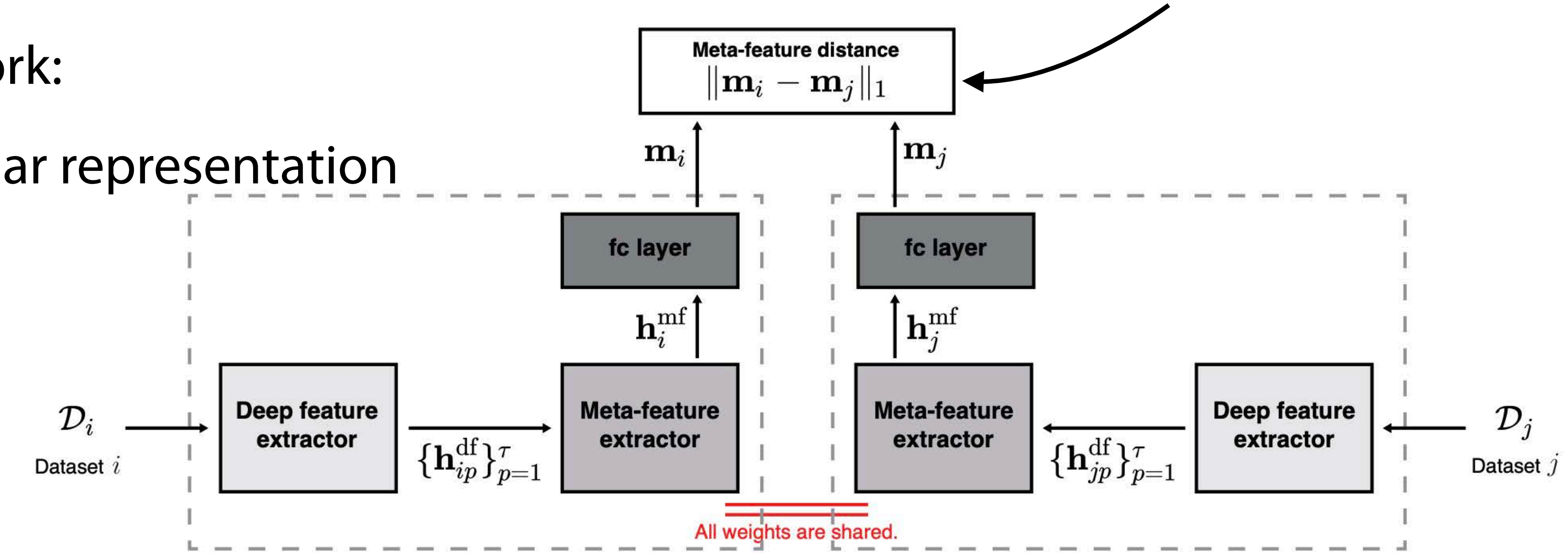
Meta-features: measurable properties of the tasks

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



Meta-features

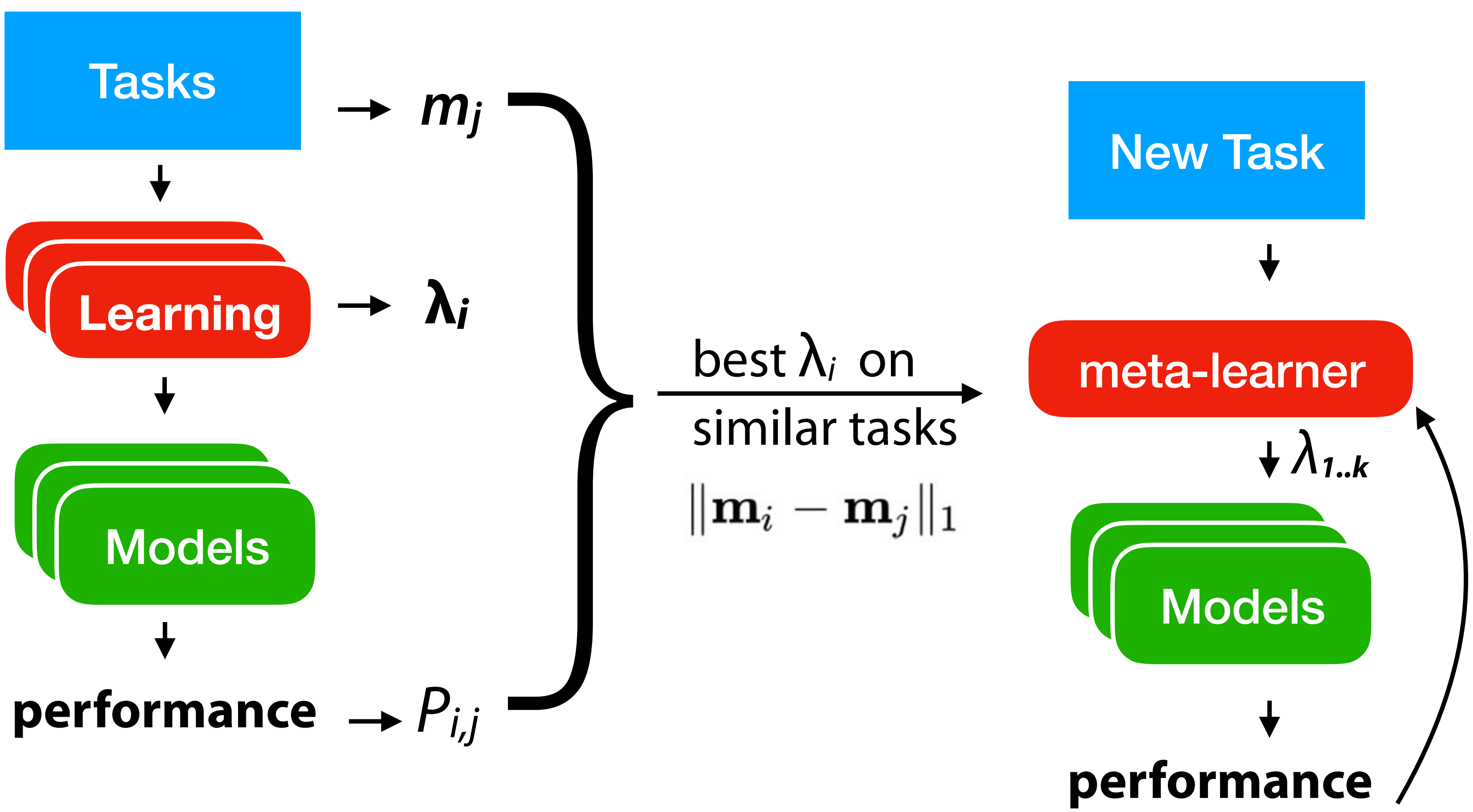
- **Hand-crafted (interpretable) meta-features**[1]
  - **Number of** instances, features, classes, missing values, outliers,…
  - **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,…
  - **Information-theoretic**: class entropy, mutual information, noise-signal ratio,…
  - **Model-based**: properties of simple models trained on the task
  - **Landmarkers**: performance of fast algorithms trained on the task
  - Domain specific task properties
- **Learning a joint task representation**
  - Deep metric learning: learn a representation h^{mf} using a ground truth distance[2]
  - With Siamese Network:
    - Similar task, similar representation



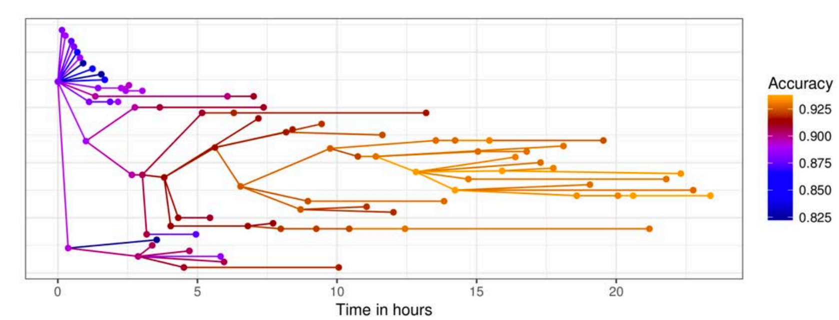
[1] Gomes et al. 2012, Reif et al. 2012
[2] Feurer et al. 2015

Warm-starting from similar tasks

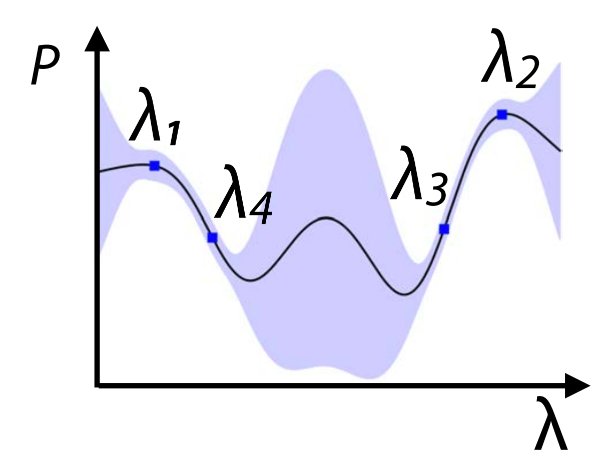
- Find k most similar tasks, warm-start search with best θ_i
  - Genetic hyperparameter search [1]
  - Auto-sklearn: Bayesian optimization (SMAC) [2]
    - Scales well to high-dimensional configuration spaces



Genetic optimization



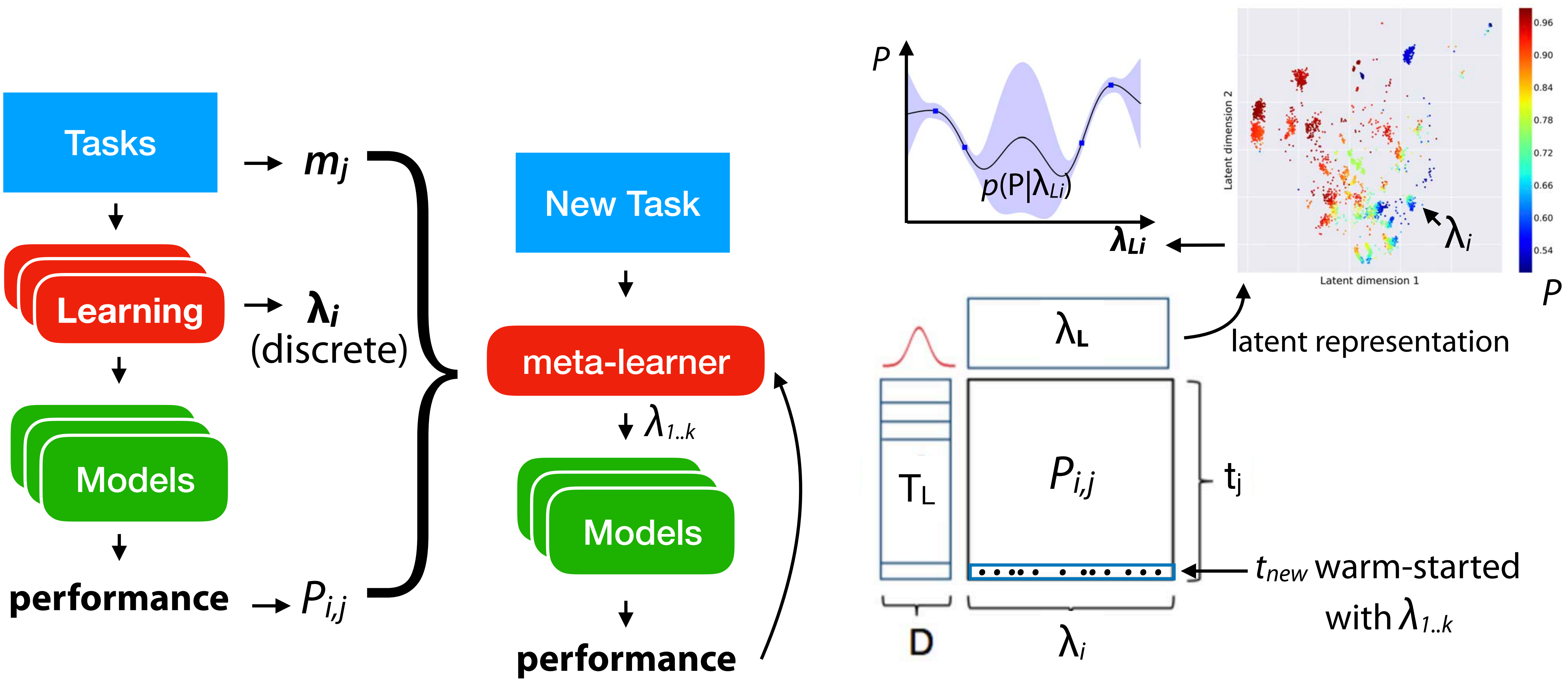
Bayesian optimization



Fusi et al. 2017

Warm-starting from similar tasks

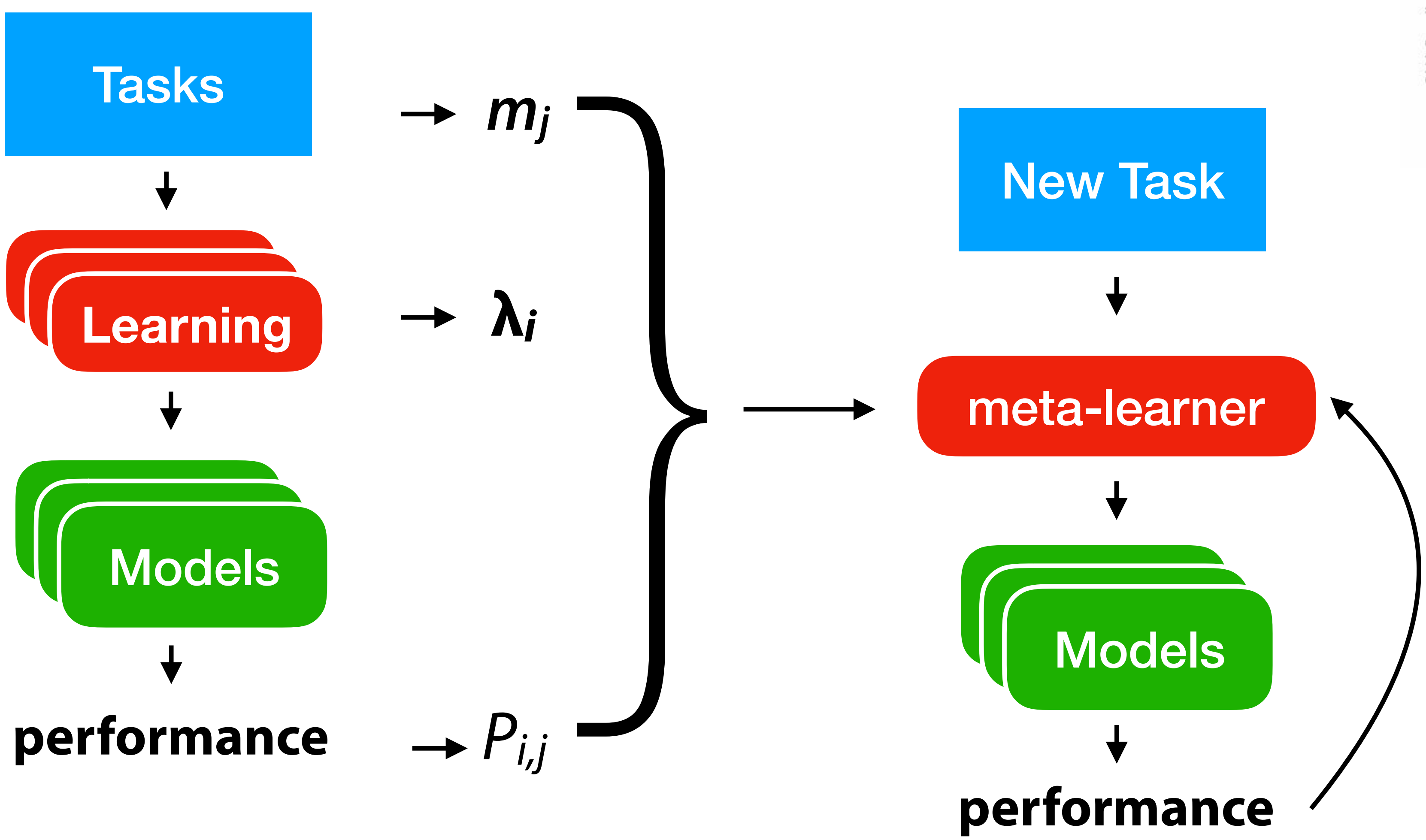
- Collaborative filtering: configurations λ_i are `rated' by tasks t_j
  - Probabilistic matrix factorization
    - Learns a latent representation for tasks and configurations
    - Returns probabilistic predictions for Bayesian optimization
    - Use meta-features to warm-start on new task

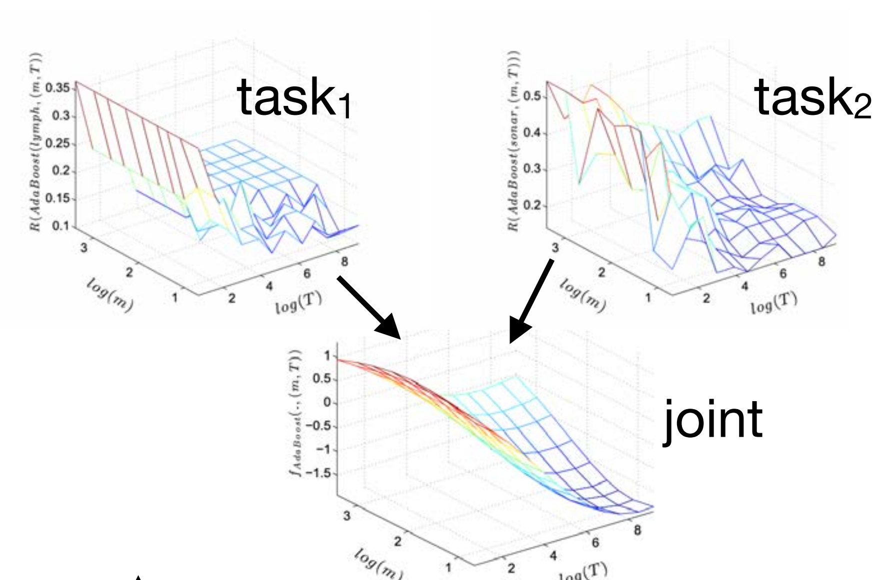


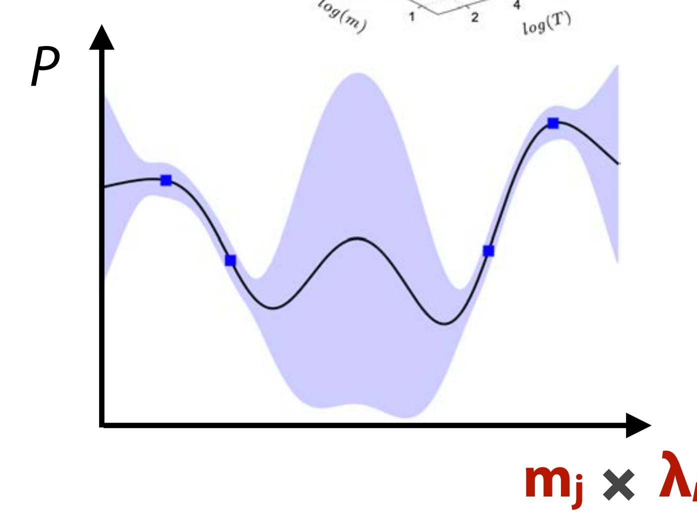
[1] Bardenet et al. 2013
[2] Schilling et al. 2015
[3] Yogatama et al. 2014

Global surrogate models

- Train a task-independent surrogate model with meta-features in inputs
  - SCOT: Predict *ranking* of λ_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 λ_i given meta-features [1]

    $\boldsymbol{m_j} \rightarrow$ meta-learner $\rightarrow \lambda_{best}$

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

    $\boldsymbol{m_j} \rightarrow$ meta-learner $\rightarrow \lambda_{1..k}$

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

    $\boldsymbol{m_j} \rightarrow$ meta-learner $\rightarrow \boldsymbol{\Lambda}$

  - Predict performance / runtime for given θ_i and task [4]

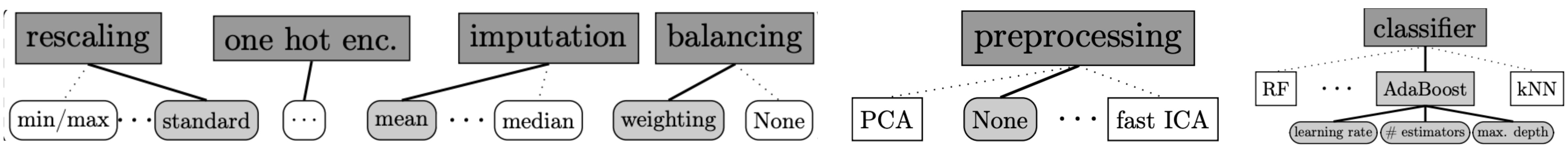
    $\boldsymbol{m_j}, \lambda_i \rightarrow$ meta-learner $\rightarrow P_{ij}$

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

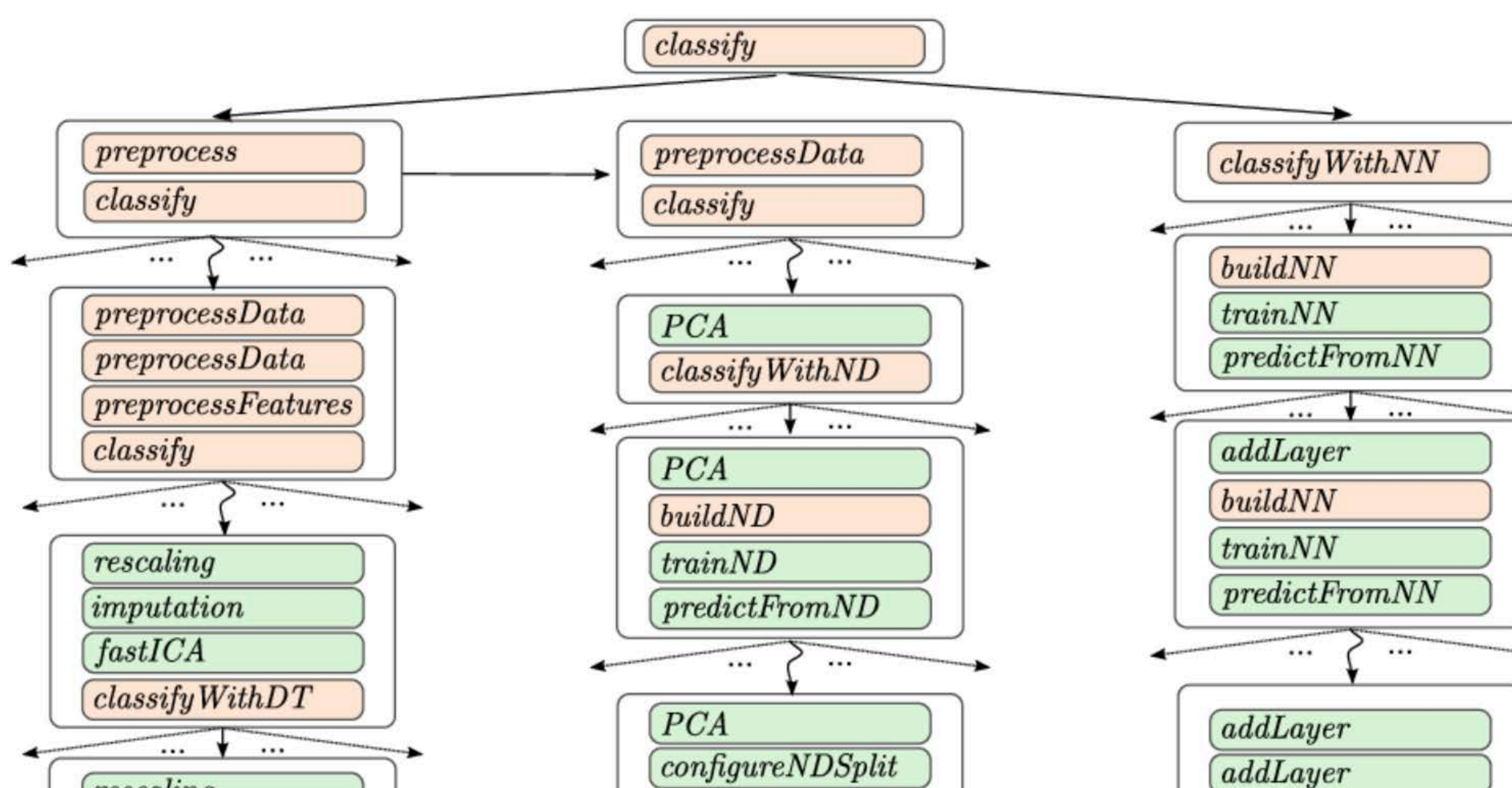
[1] Fusi et al. 2017
[2] Feurer et al. 2015
[3] Mohr et al. 2018

Learning Pipelines

- **Compositionality**: the learning process can be broken down into smaller tasks
    - Easier to learn, more transferable, more robust
- Pipelines are one way of doing this, but how to control the search space?
    - Select a fixed set of possible pipelines. Often works well (less overfitting) [1]
    - Impose a fixed structure on the pipeline [2]



- (Hierarchical) Task Planning [3]
    - Break down into smaller tasks

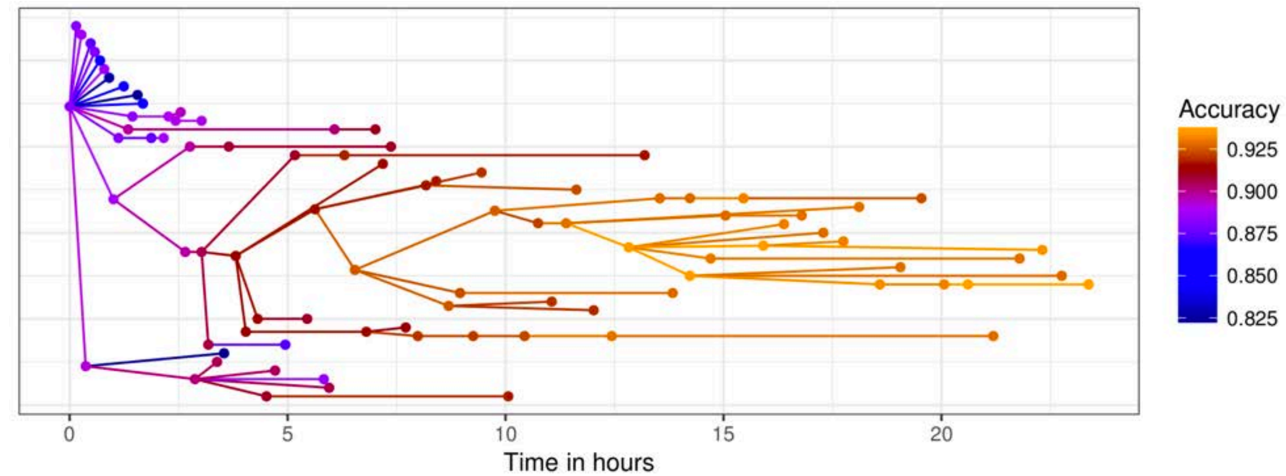


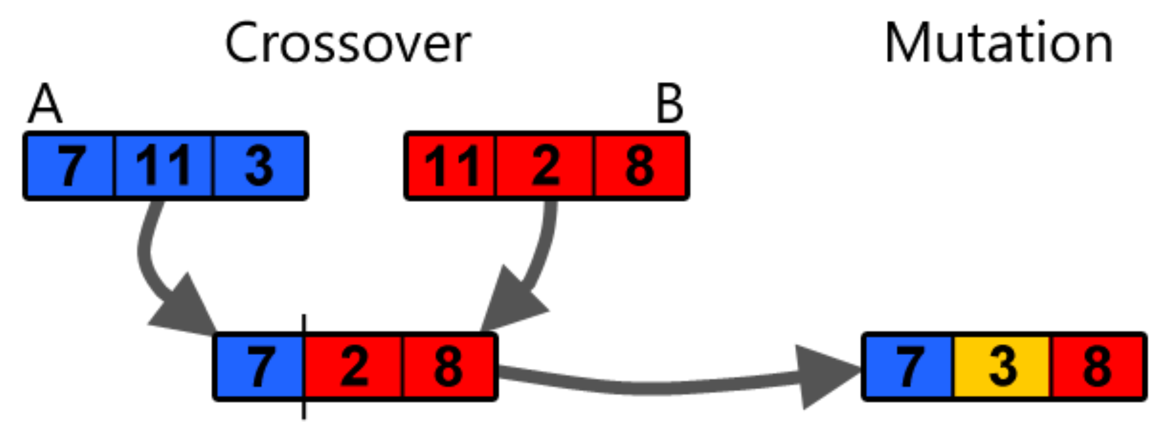
- Meta-learning:
    - Mostly warm-starting

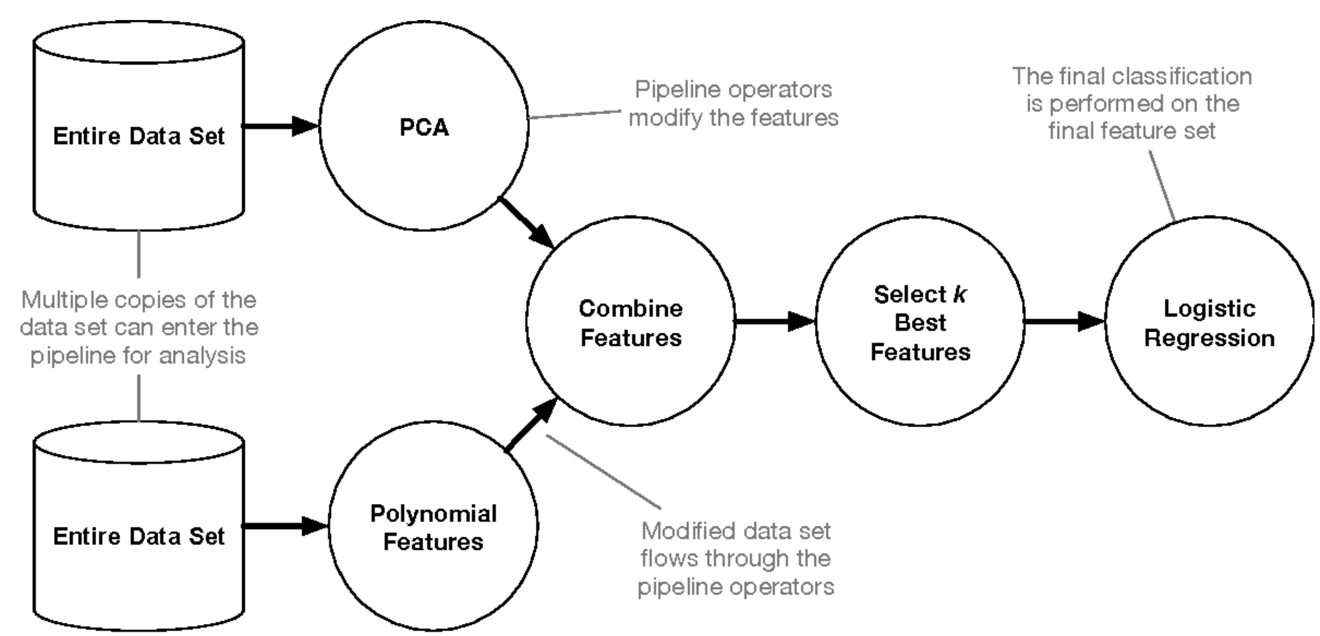
[1] Olson et al. 2017
[2] Gijsbers et al. 2018
[3] De Sa et al. 2017

Evolving pipelines

- Start from simple pipelines
- *Evolve* more complex ones if needed
- Reuse pipelines that do specific things
- Mechanisms:
    - Cross-over: reuse partial pipelines
    - Mutation: change structure, tuning
- Approaches:
    - TPOT: Tree-based pipelines[1]
    - GAMA: asynchronous evolution[2]
    - RECIPE: grammar-based[3]
- Meta-learning:
    - Largely unexplored
    - Warm-starting, meta-models

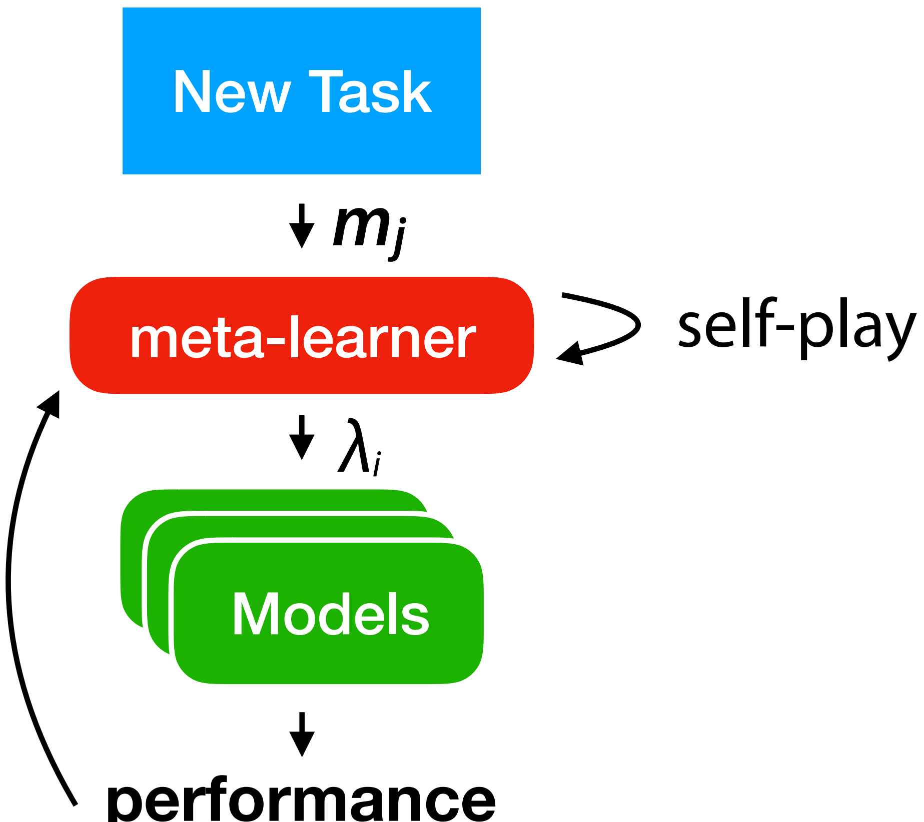


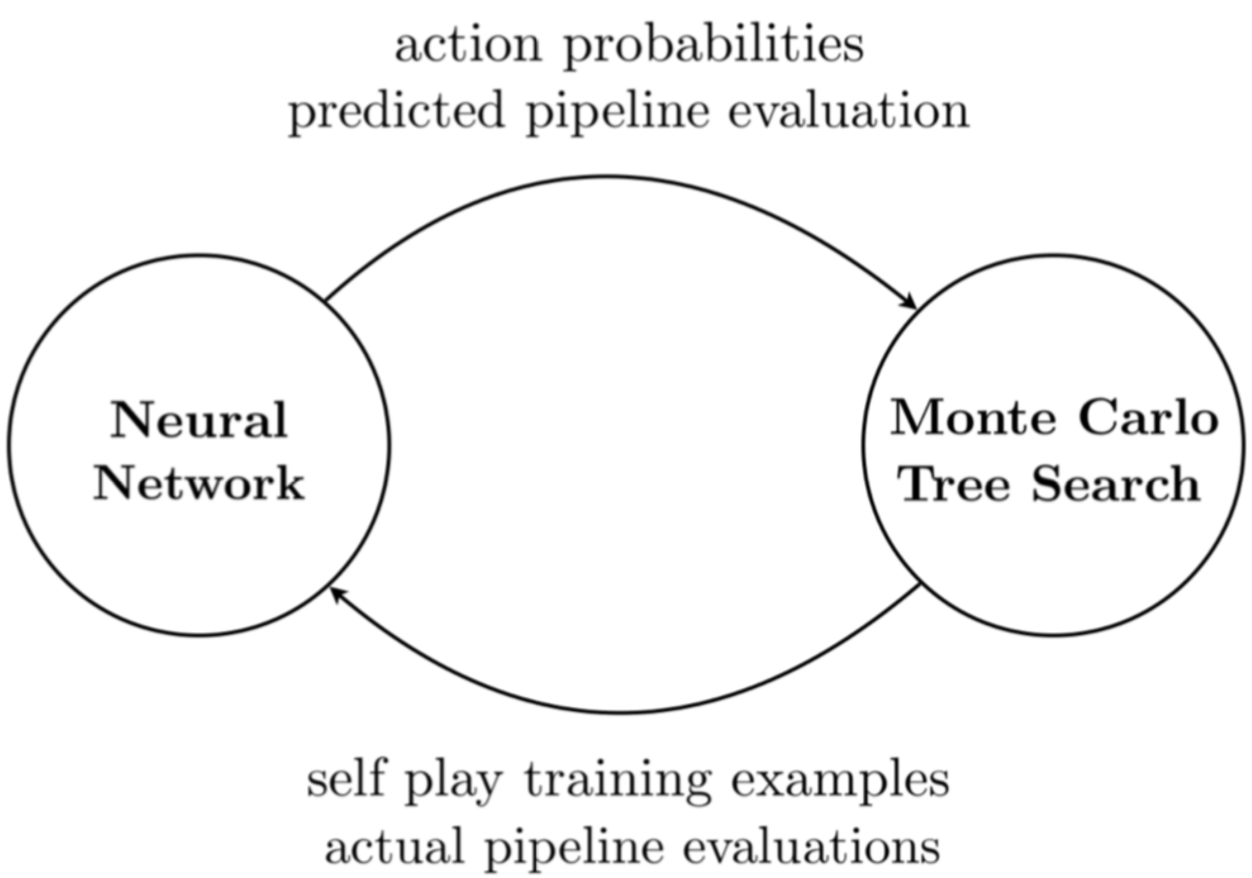




Learning to learn through self-play

- Build pipelines by selecting among actions
  - insert, delete, replace pipeline parts
- Neural network (LSTM) receives task meta-features, pipelines and evaluations
  - Predict pipeline performance and action probabilities
- Monte Carlo Tree Search builds pipelines based on probabilities
  - Runs multiple simulations to search for a better pipeline

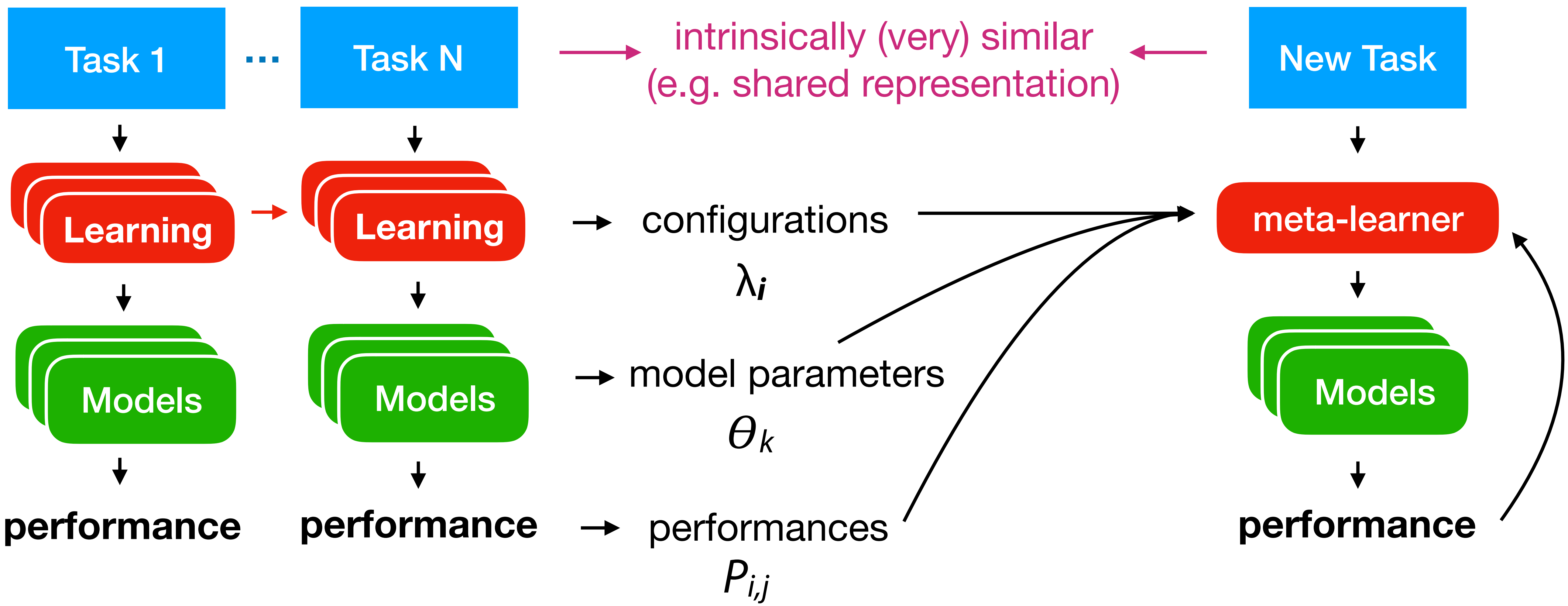




3. Learning from trained models

Models trained on similar tasks

(model parameters, features,…)



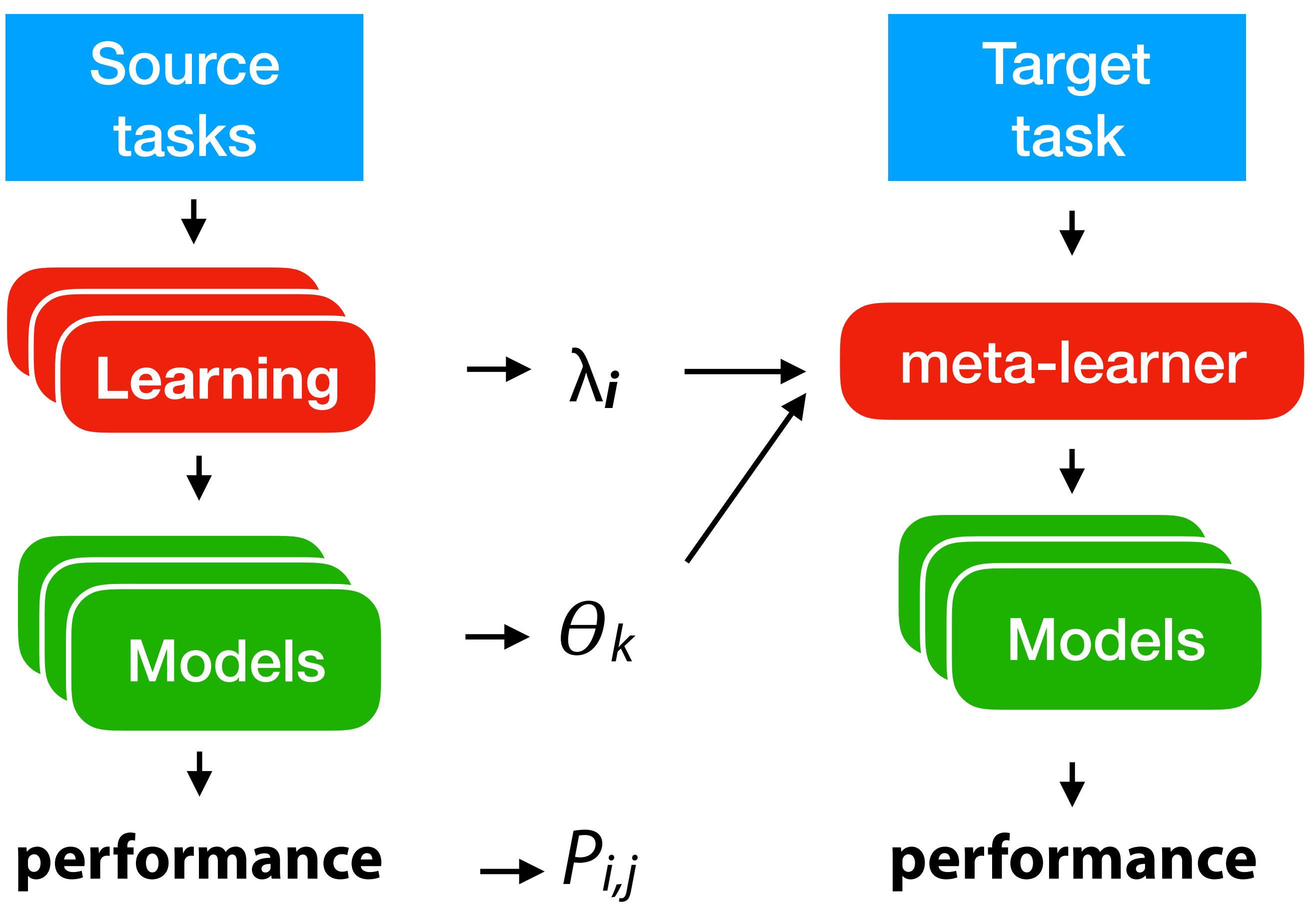
[1] Thrun and Pratt 1998

[2] Niculescu-Mizil and Caruana 2005

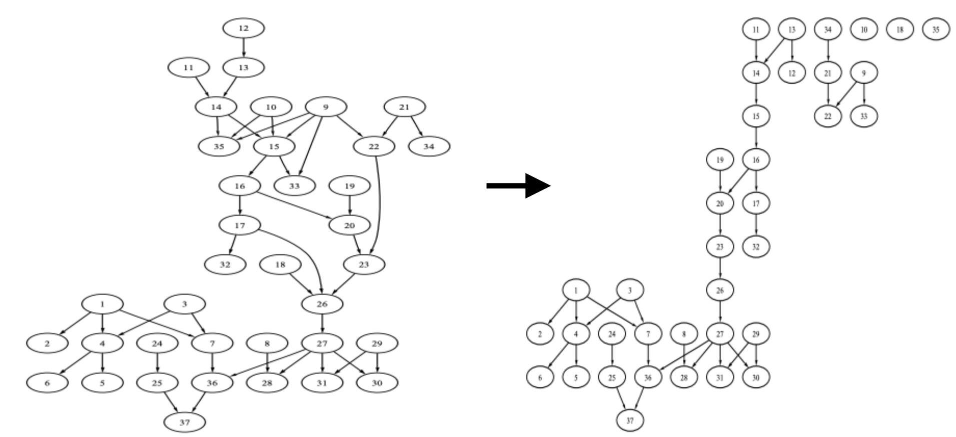
[3] Taylor and Stone 2009

Transfer Learning

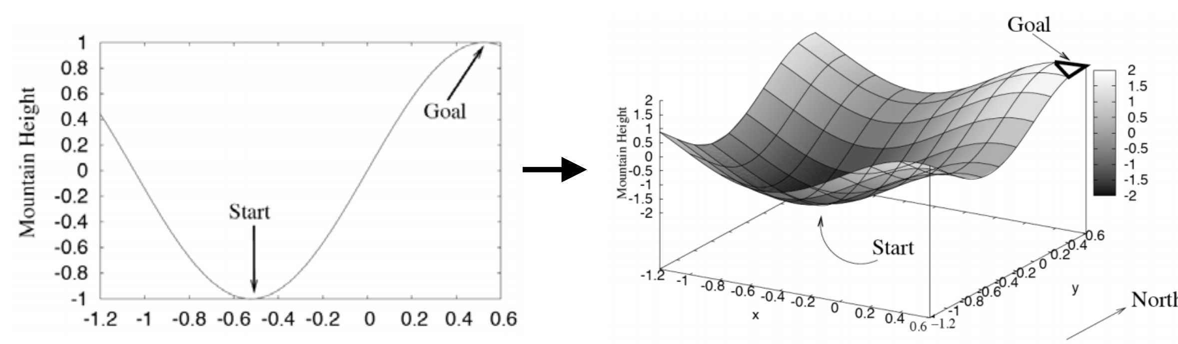
- Select source tasks, transfer trained models to similar target task [1]
- Use as starting point for tuning, or *freeze* certain aspects (e.g. structure)
  - Bayesian networks: start structure search from prior model [2]
  - Reinforcement learning: start policy search from prior policy [3]



Bayesian Network transfer



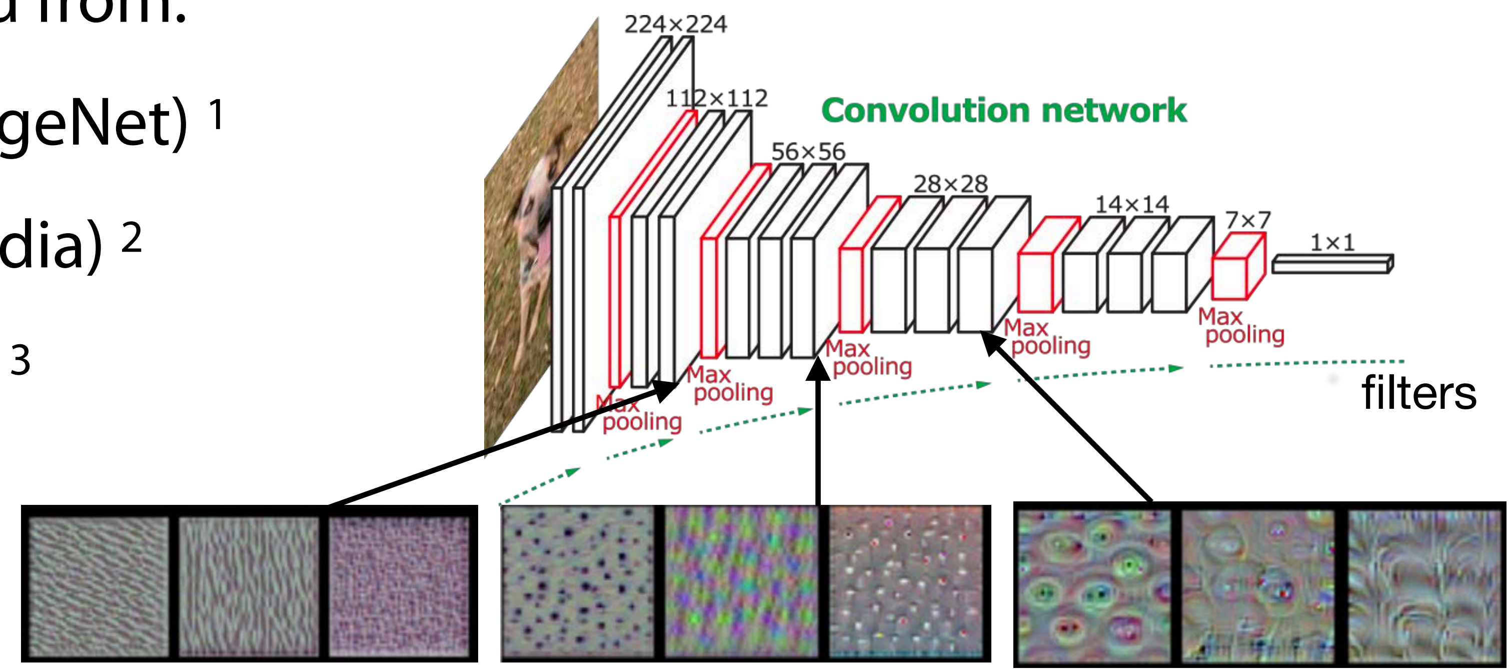
Reinforcement learning: 2D to 3D mountain car

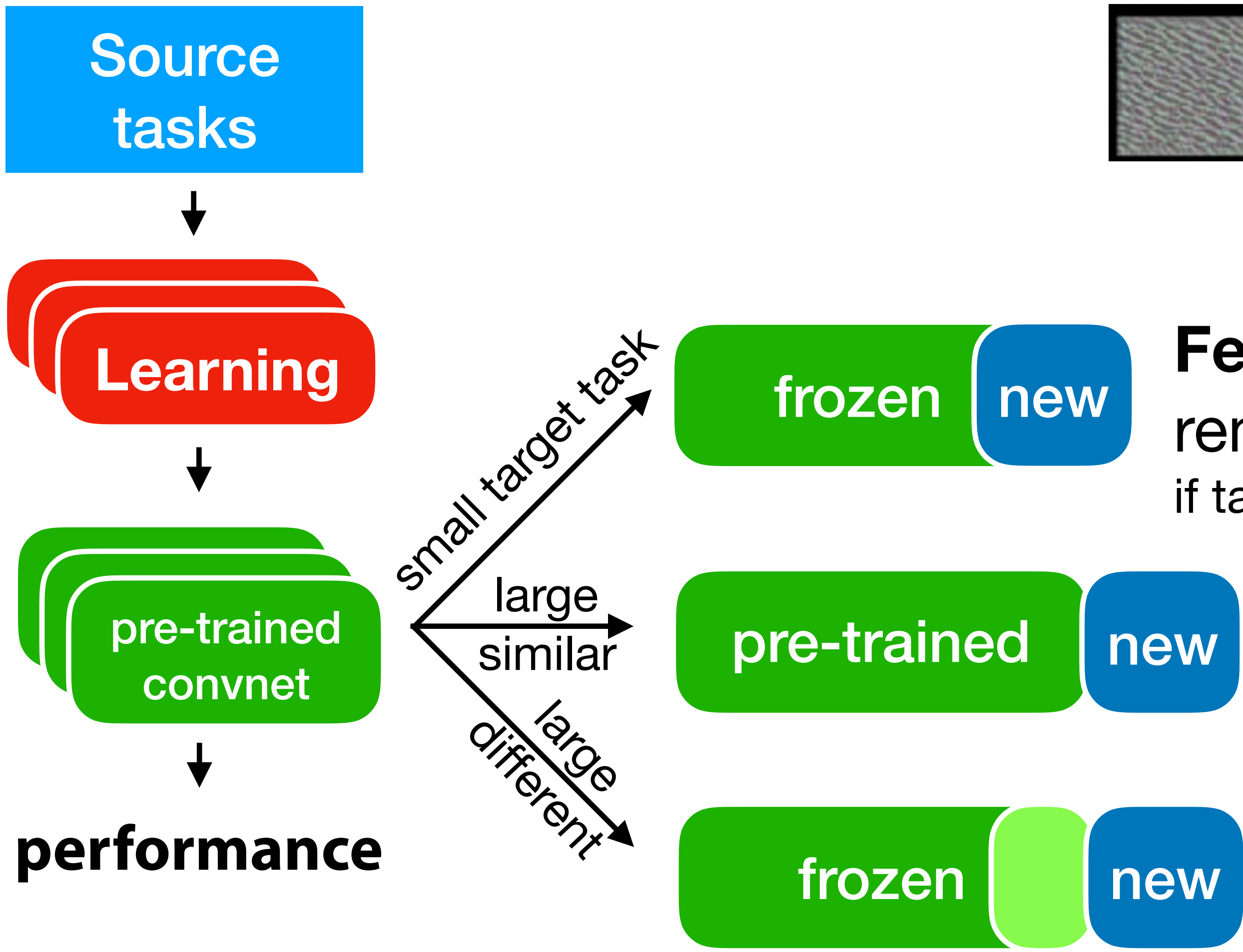


[1] Razavian et al. 2014
[2] Mikolov et al. 2013
[3] Yosinski et al. 2014

Transfer features, initializations

- 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

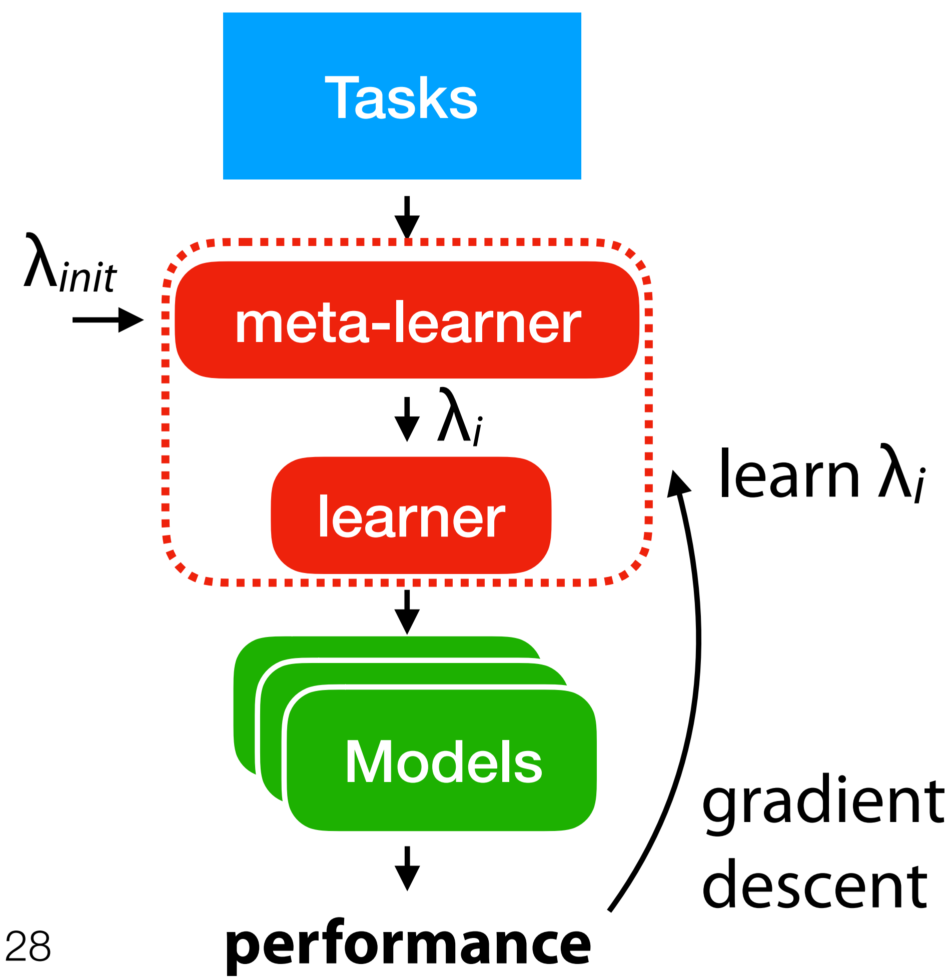
Fine-tuning:
unfreeze last layers, tune on new task

[1] Bengio et al. 1995

[2] Runarsson and Jonsson 2000

Learning to learn by gradient descent

- Our brains *probably* don't do backprop, replace it with: $\cancel{\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}}$
  - Simple *parametric* (bio-inspired) rule to update weights [1]
  - Single-layer neural network to learn weight updates [2]
- Learn parameters across tasks, by gradient descent (meta-gradient)

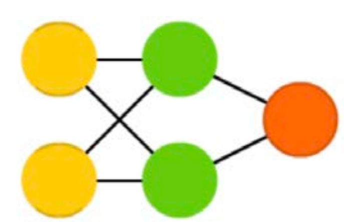


Bengio et al.

$$\Delta\,\theta_i = \underbrace{\epsilon}_{\text{learning rate}}\; \overbrace{y_{pre(i)}}^{\text{presynaptic activity}}\; \underbrace{k}_{\text{reinforcing signal}}$$

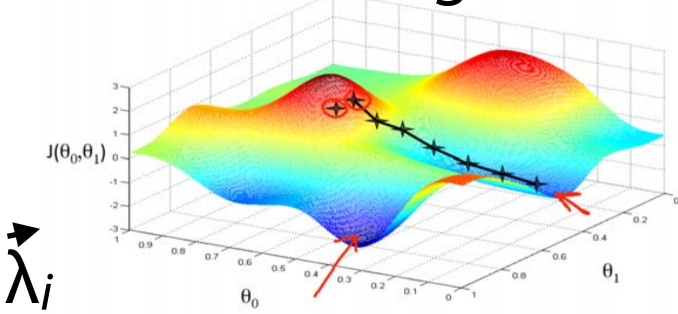
Runarsson and Jonsson

$$\Delta\,\theta_i = \underbrace{\eta}_{\text{learning rate}}\;(\quad)$$



weights λ_i

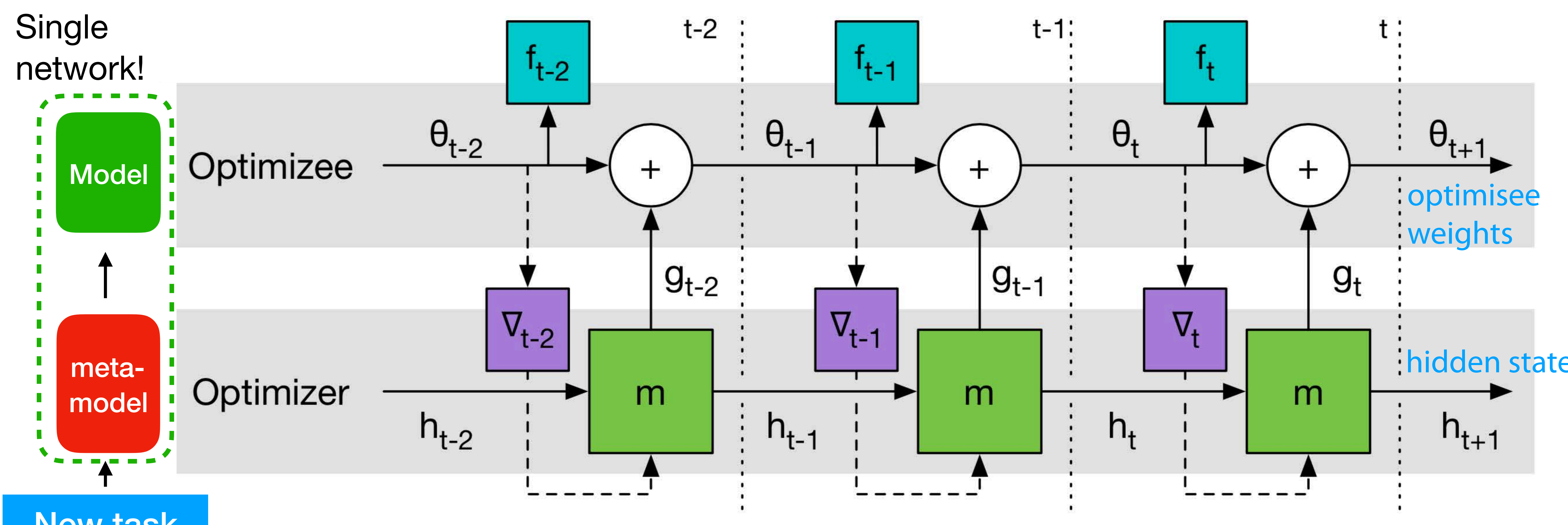
meta-gradient



[1] Hochreiter 2001
[2] Andrychowicz et al. 2016

Learning to learn gradient descent by gradient descent

- Replace backprop with a recurrent neural net (LSTM)[1], not so scalable
- Use a coordinatewise LSTM [m] for scalability/flexibility (cfr. ADAM, RMSprop) [2]
  - Optimizee: receives weight update g_t from optimizer
  - Optimizer: receives gradient estimate ∇_t from optimizee
    - Learns how to do gradient descent across tasks

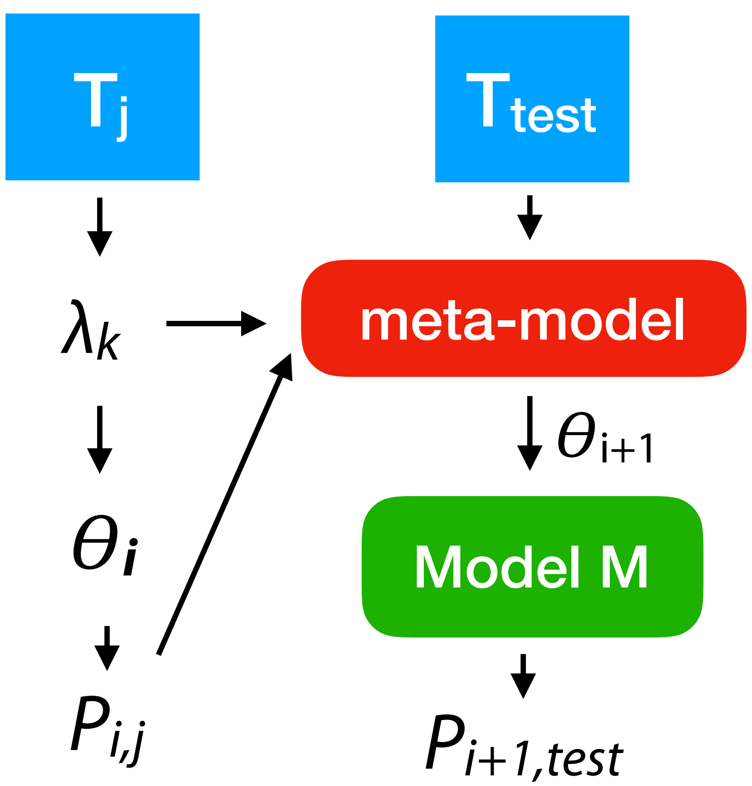


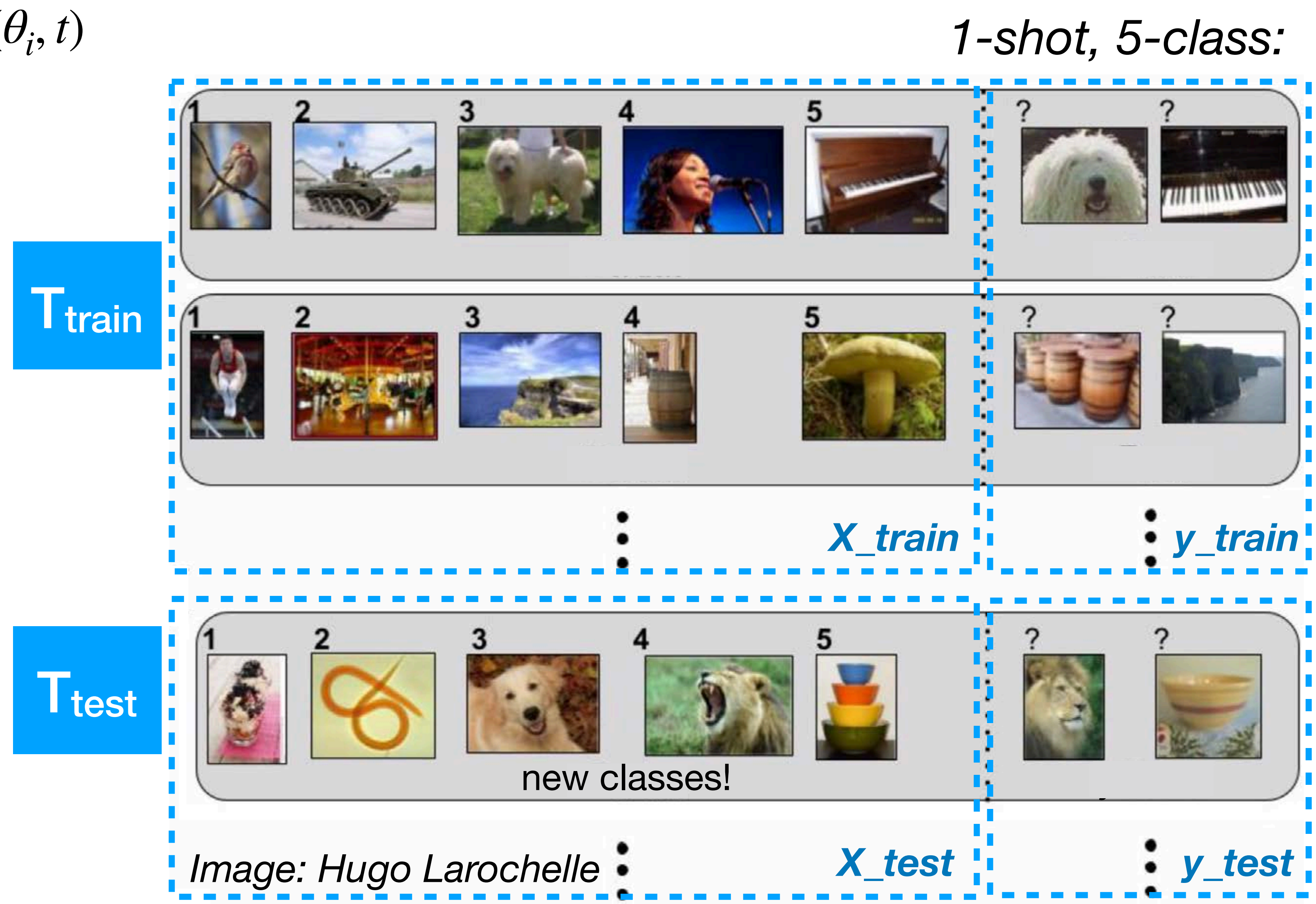
LSTM parameters shared for all θ

Few-shot learning

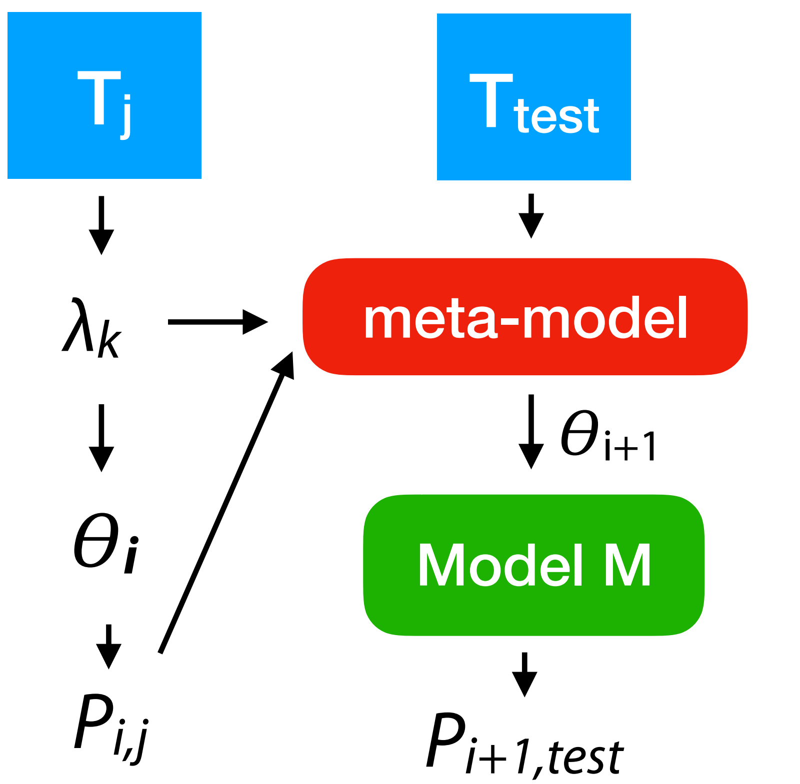
- Learn how to learn from few examples (given similar tasks)
  - Meta-learner must learn how to train a base-learner based on prior experience
    - Parameterize base-learner model and learn the parameters θ_i

$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$





Few-shot learning: approaches



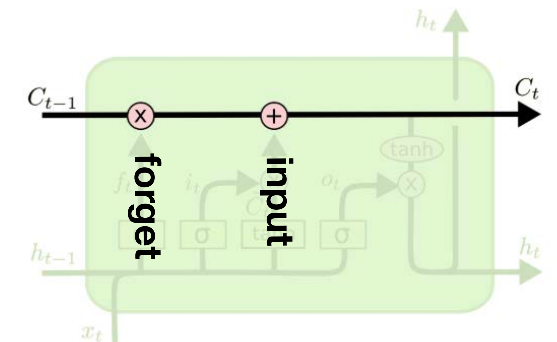
$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$

- Existing algorithm as meta-learner:
  - LSTM + gradient descent — Ravi and Larochelle 2017
  - Learn θ_{init} + gradient descent — Finn et al. 2017
  - kNN-like: Memory + similarity — Vinyals et al. 2016
  - Learn embedding + classifier — Snell et al. 2017
  - …
- Black-box meta-learner
  - Neural Turing machine (with memory) — Santoro et al. 2016
  - Neural attentive learner — Mishra et al. 2018
  - …

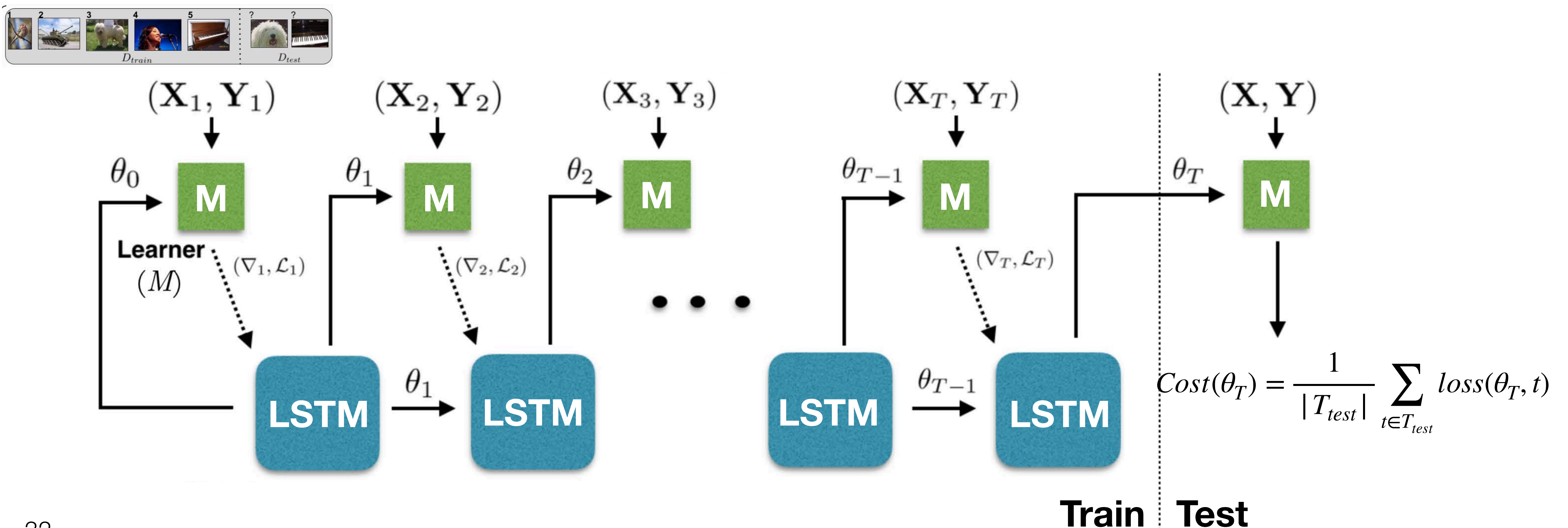
LSTM meta-learner + gradient descent

- Gradient descent update θ_t is similar to LSTM cell state update c_t

$$\boxed{\theta_t} = \boxed{\theta_{t-1}} - \alpha_t \boxed{\nabla_{\theta_{t-1}} \mathcal{L}_t} \qquad \boxed{c_t} = f_t \odot \boxed{c_{t-1}} + i_t \odot \boxed{\tilde{c}_t}$$



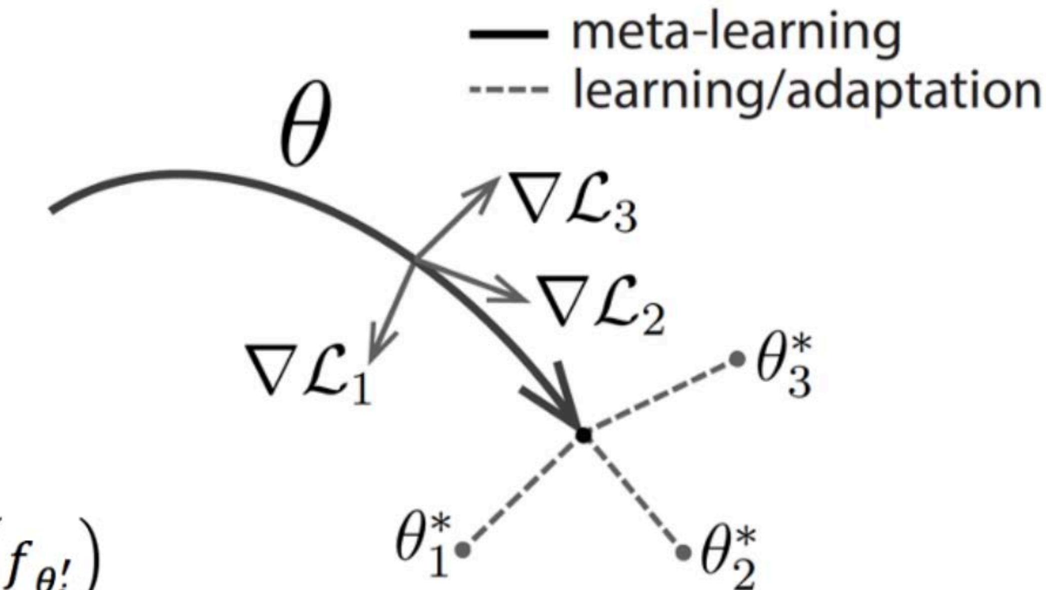
- Hence, training a meta-learner LSTM yields an update rule for training M
  - Start from initial θ_0, train model on first batch, get gradient and loss update
  - Predict θ_{t+1}, continue to t=T, get cost, backpropagate to learn LSTM weights, optimal θ_0



Model-agnostic meta-learning

- Quickly learn new skills by learning a model *initialization* that generalizes better to similar tasks
  - Current initialization θ
  - On K examples/task, evaluate $\nabla_\theta L_{T_i}(f_\theta)$
  - Update weights for $\theta_1, \theta_2, \theta_3$
  - Update θ to minimize sum of per-task losses
  - Repeat

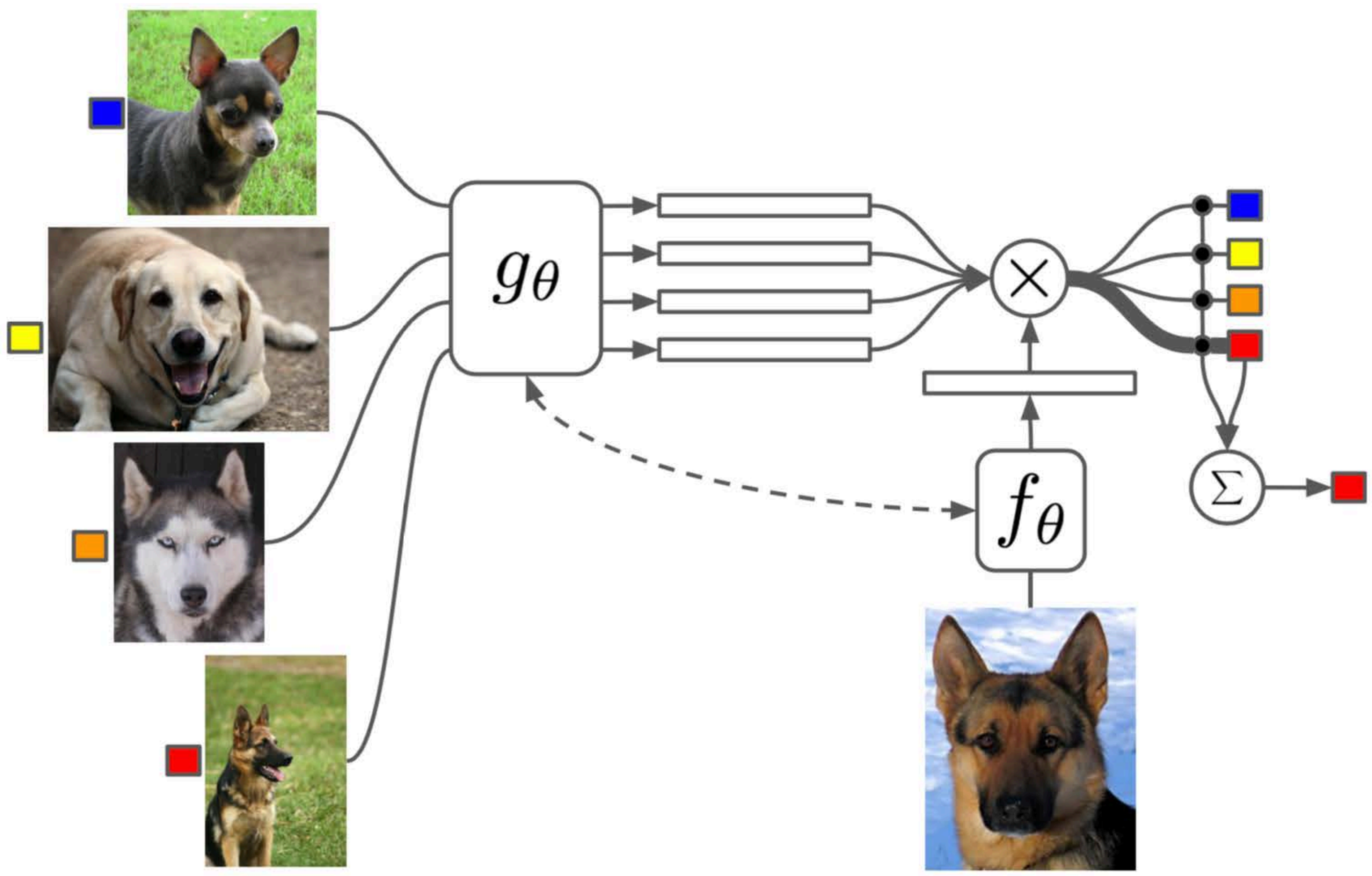
$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}\left(f_{\theta_i'}\right)$$



- More resilient to overfitting
- Generalizes better than LSTM approaches
- *Universality*: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- REPTILE: do SGD for k steps in one task, only then update initialization weights[3]

1-shot learning with Matching networks

- Don't learn model parameters, use non-parameters model (like kNN)
- Choose an embedding network f and g (possibly equal)
- Choose an attention kernel $a(\hat{x}, x_i)$, e.g. softmax over cosine distance
- Train complete network in minibatches with few examples per task



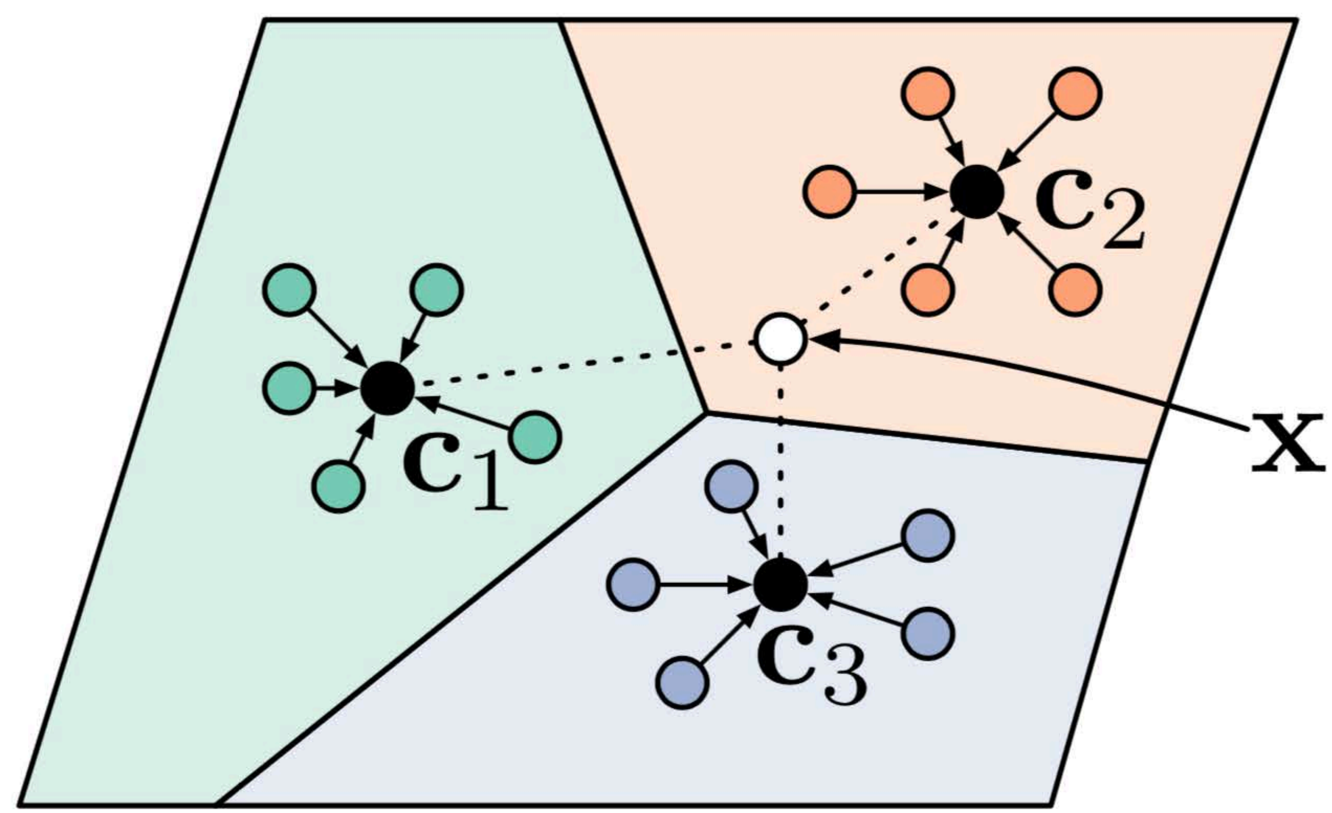
$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

$\theta = \{\text{VGG, Inception,}\ldots\}$

Snell et al. 2017
Ren et al. 2018

Prototypical networks

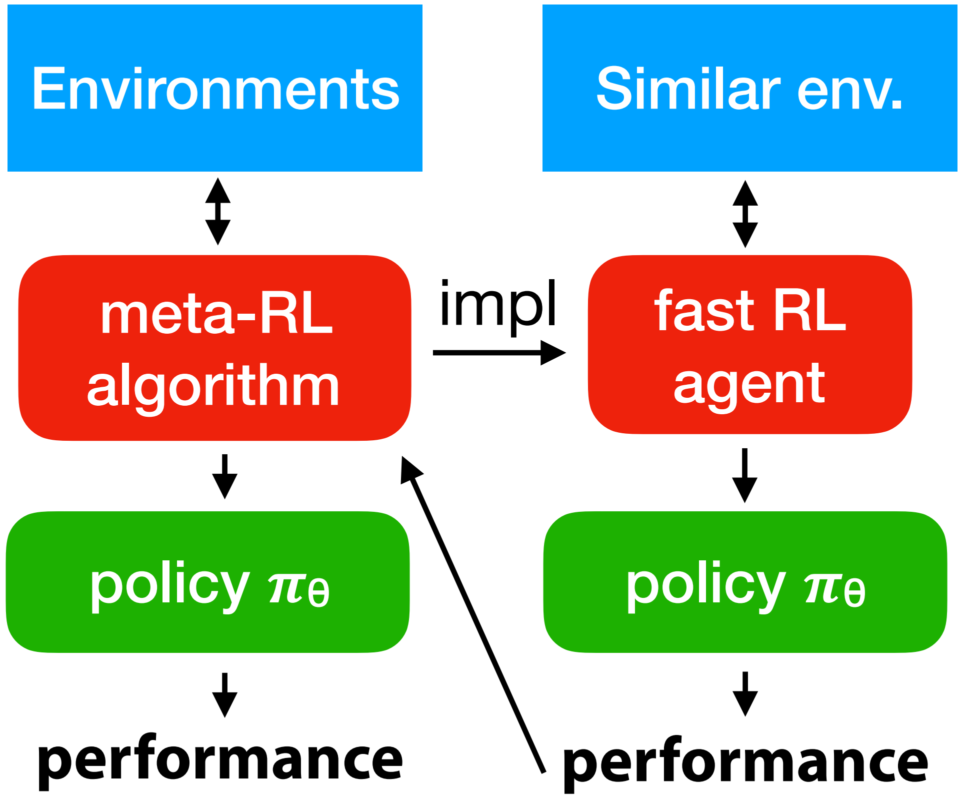
- Train a “prototype extractor” network
- Map examples to p-dimensional embedding so examples of a given class are close together
- Calculate a prototype (mean vector) for every class
- Map test instances to the same embedding, use softmax over distance to prototype
- Using more classes during meta-training works better!

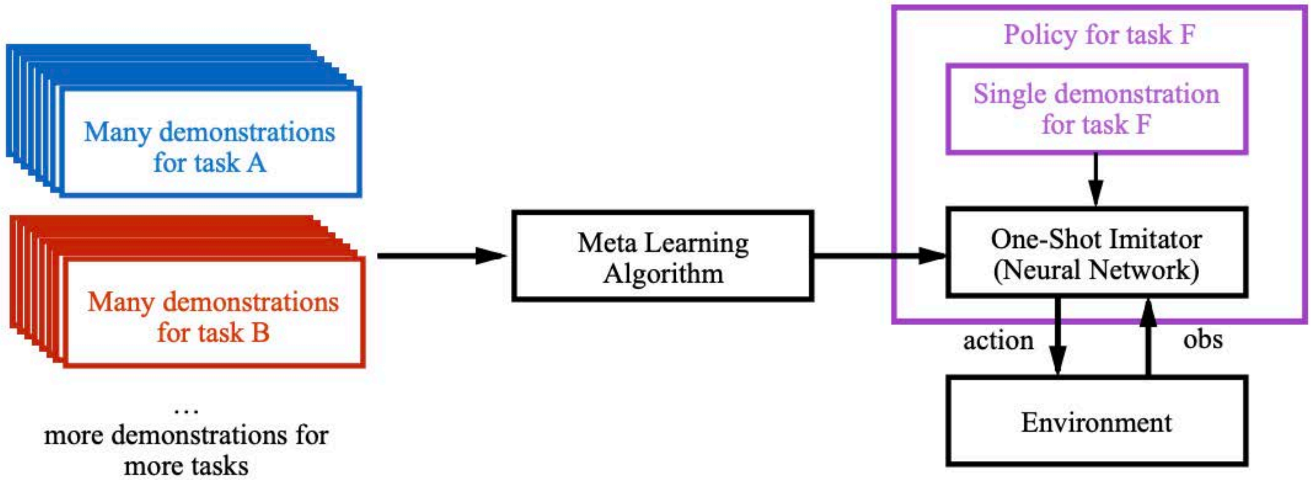


[1] Duan et al. 2017
[2] Wang et al. 2017
[3] Duan et al. 2017

Learning to reinforcement learn

- Humans often learn to play new games much faster than RL techniques do
- Reinforcement learning is very suited for learning-to-learn:
  - Build a learner, then use performance as that learner as a reward
- Learning to reinforcement learn [1,2]
  - Use RNN-based deep RL to train a recurrent network on many tasks
  - Learns to implement a 'fast' RL agent, encoded in its weights





- Also works for few-shot learning [3]
  - Condition on observation + upcoming demonstration
- You don't know what someone is trying to teach you, but you prepare for the lesson

Learning to learn more tasks

- Active learning *Pang et al. 2018*
  - Deep network (learns representation) + policy network
  - Receives state and reward, says which points to query next
- Density estimation *Reed et al. 2017*
  - Learn distribution over small set of images, can generate new ones
  - Uses a MAML-based few-shot learner
- Matrix factorization *Vartak et al. 2017*
  - Deep learning architecture that makes recommendations
  - Meta-learner learns how to adjust biases for each user (task)
- Replace hand-crafted algorithms by learned ones.
- Look at problems through a meta-learning lens!

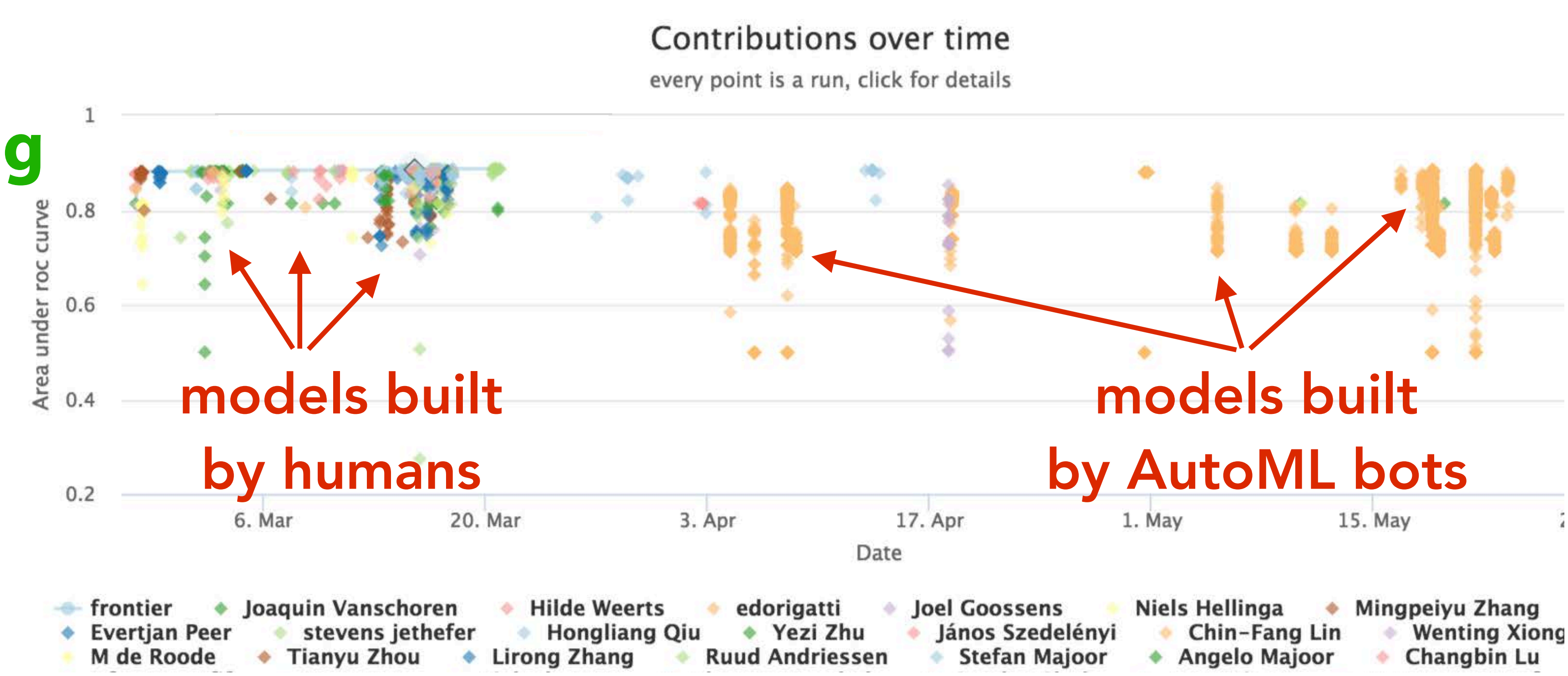
Meta-data sharing building a shared memory

- OK, but how do I get large amounts of meta-data for meta-learning?
- OpenML.org
  - Thousands of uniform datasets
  - 100+ meta-features
  - Millions of evaluated runs
    - Same splits, 30+ metrics
    - Traces, models *(opt)*
  - **APIs in Python, R, Java,…**
  - Publish your own runs
  - **Never ending learning**
  - Benchmarks

```
import openml as oml
from sklearn import tree

task = oml.tasks.get_task(14951)
clf = tree.ExtraTreeClassifier()
flow = oml.flows.sklearn_to_flow(clf)
run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()
```

run *locally*, share *globally*



Open positions!
Scientific programmer
Teaching PhD

Towards human-like learning to learn

- Learning-to-learn gives humans a significant advantage
  - **Learning how to learn any task empowers us far beyond knowing how to learn specific tasks.**
  - It is a **universal** aspect of life, and how it evolves
- Very exciting field with many unexplored possibilities
  - Many aspects not understood (e.g. task similarity), need more experiments.
- **Challenge**:
  - Build learners that **never stop learning**, that **learn from each other**
  - Build a ***global memory*** for learning systems to learn from
  - **Let them explore by themselves**, active learning

Thank you!

Merci!

→ Questions → Questions → Questions → Questions → Questions →

more to learn

http://www.automl.org/book/

Chapter 2: Meta-Learning

special thanks to

Pavel Brazdil, Matthias Feurer, Frank Hutter, Erin Grant, Hugo Larochelle, Raghu Rajan, Jan van Rijn, Jane Wang