

# Automatic Machine Learning

## A Tutorial

Joaquin Vanschoren  
Eindhoven University of Technology

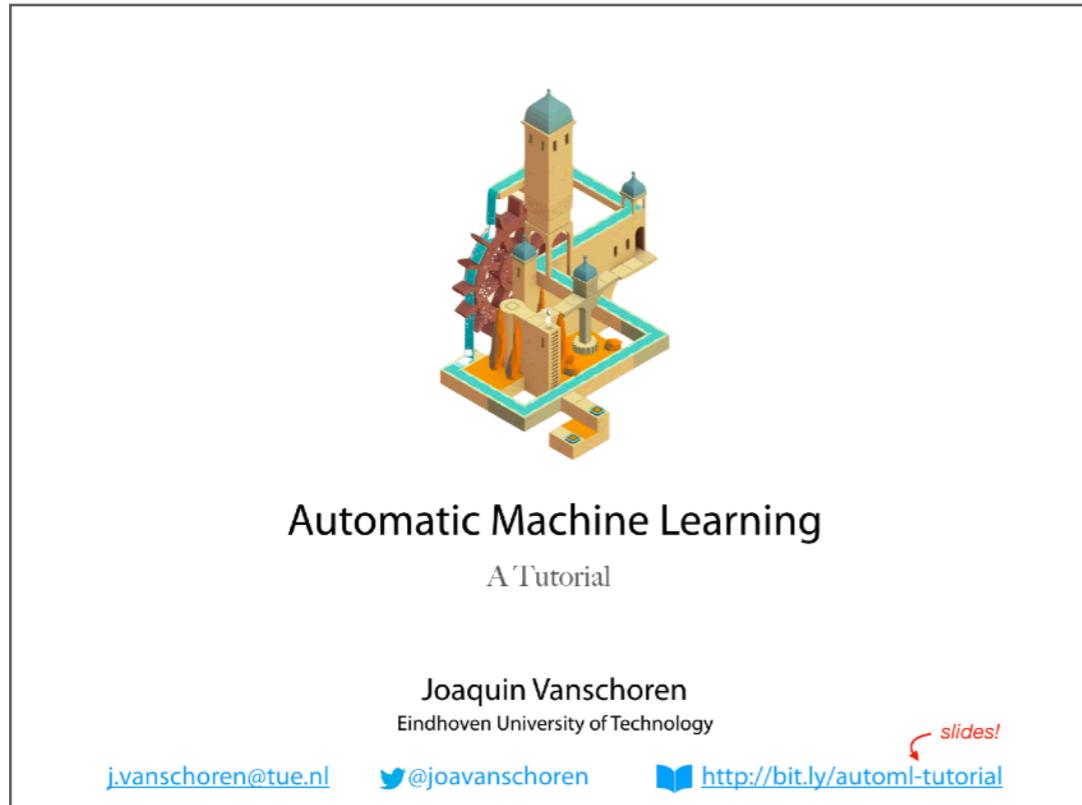
[j.vanschoren@tue.nl](mailto:j.vanschoren@tue.nl)

 [@joavanschoren](https://twitter.com/joavanschoren)

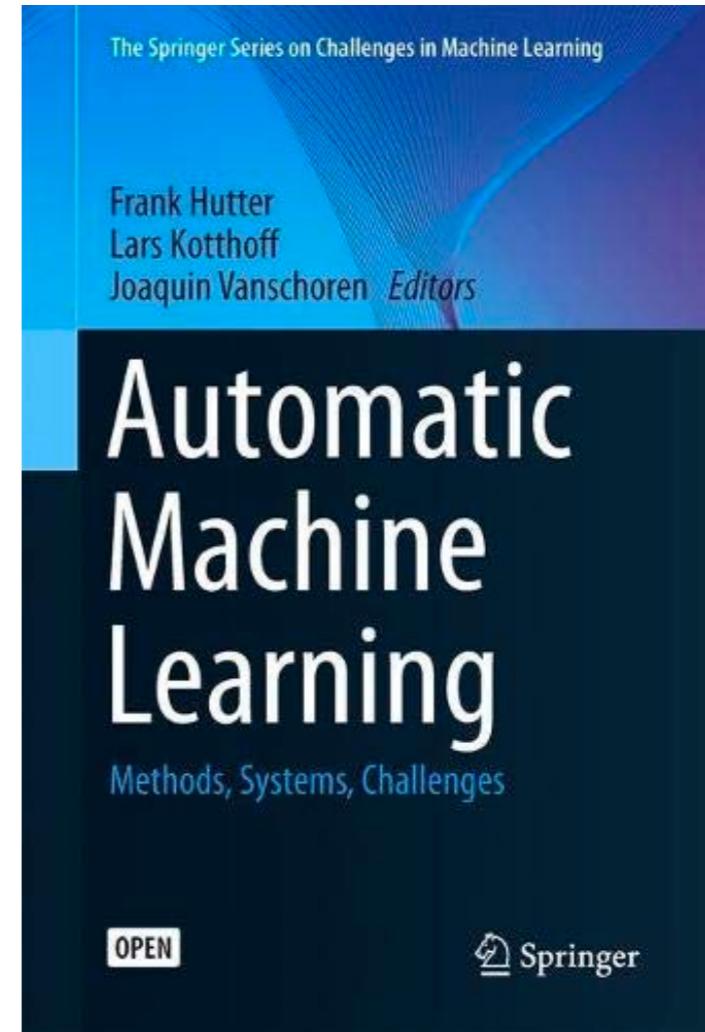
 <http://bit.ly/automl-tutorial>

slides!

# Slides / Book



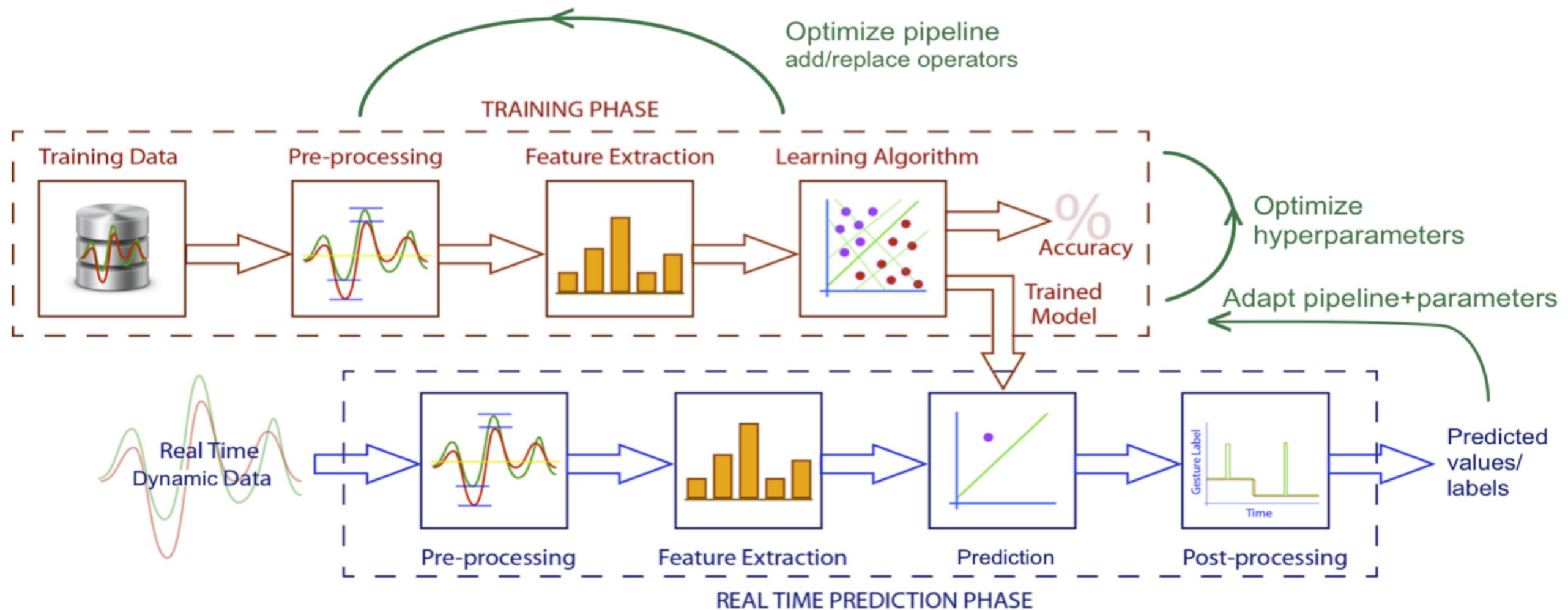
Slides  
<http://bit.ly/automl-tutorial>



Open access book  
(free PDF)  
[www.automl.org/book](http://www.automl.org/book)  
[www.amazon.de/dp/3030053172](http://www.amazon.de/dp/3030053172)

# *Doing machine learning requires a lot of expertise*

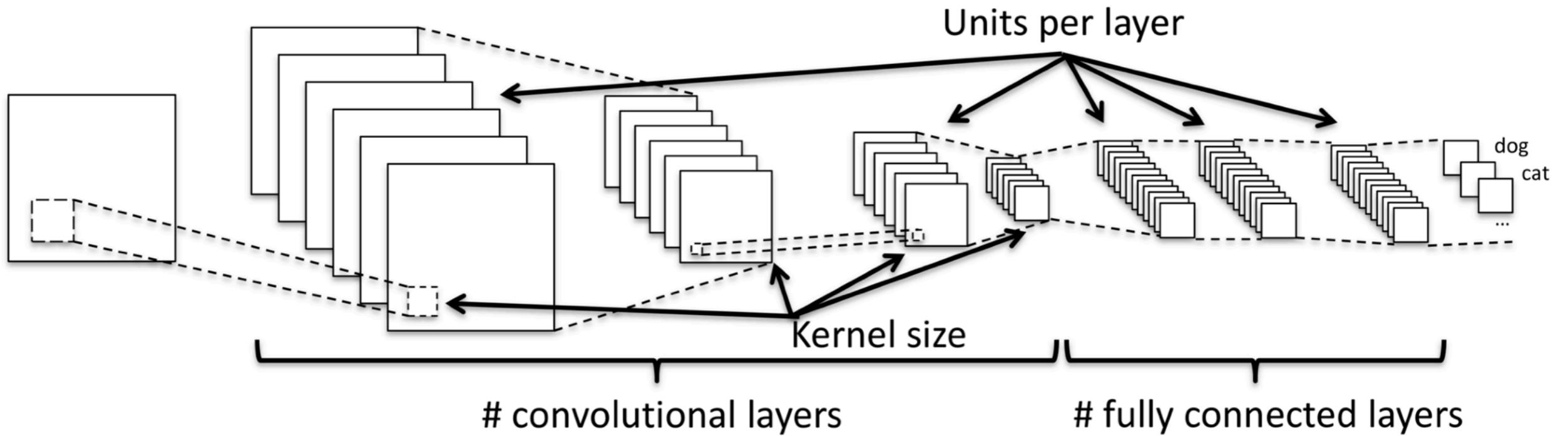
## Designing pipelines



- Excellent tools (e.g. scikit-learn), but little guidance
- Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,...

# *Doing machine learning requires a lot of expertise*

## Designing neural architectures

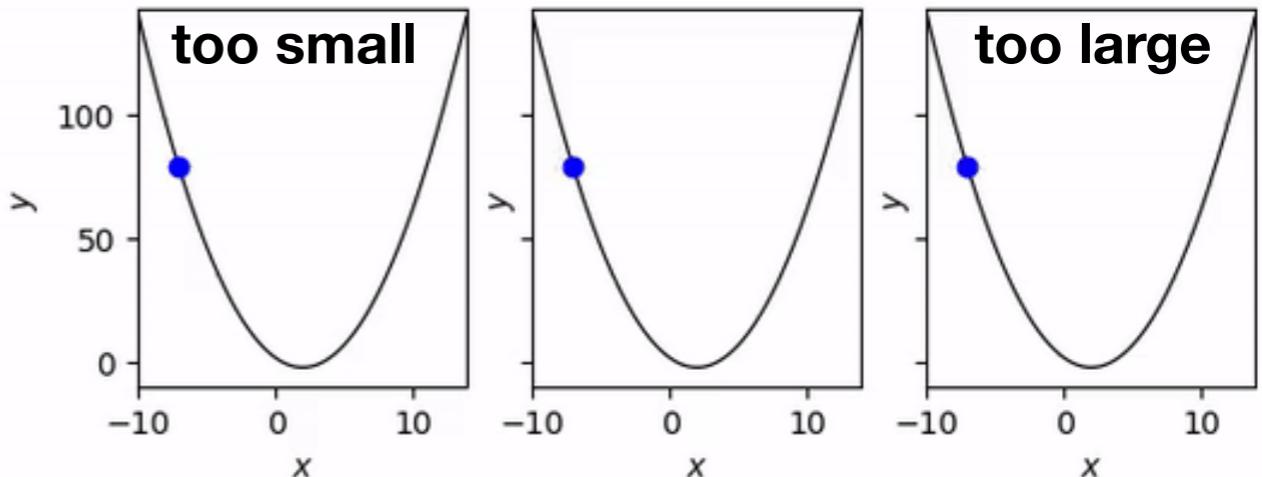


- Choose operators (layers), data preparation, data augmentation, hyperparameter tuning, regularization, early stopping,...

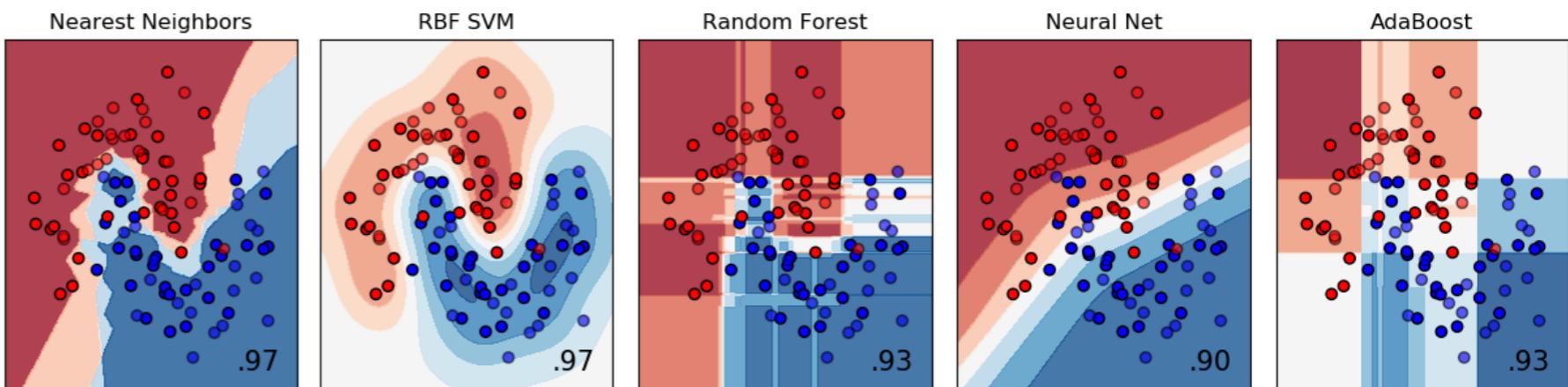
# Hyperparameters

Every design decision made by the user (*architecture, operators, tuning,...*)

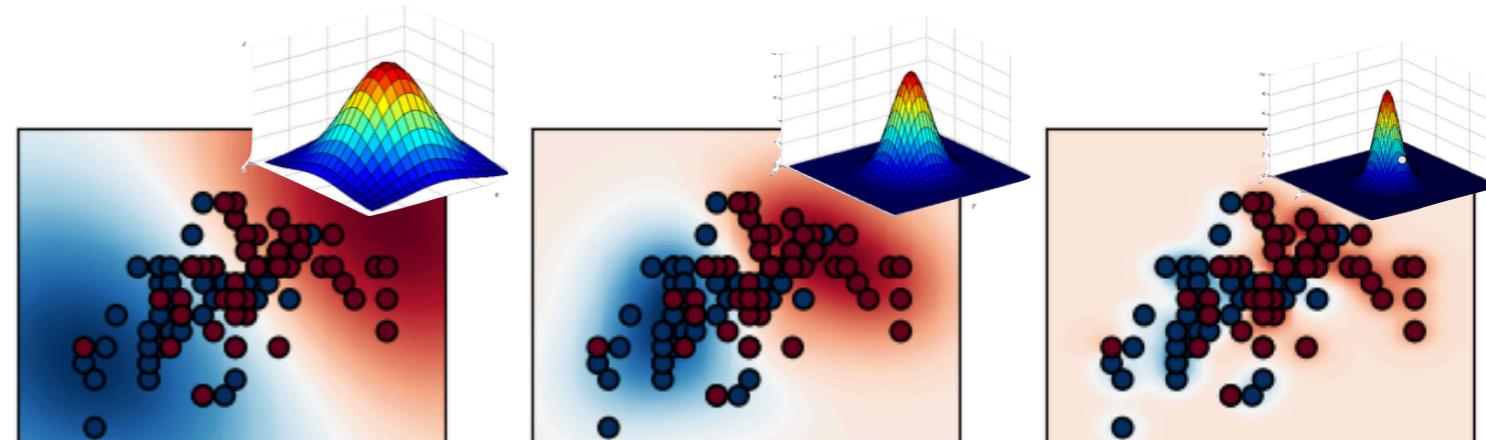
- Numeric
  - e.g. learning rate



- Categorical
  - e.g. classifier



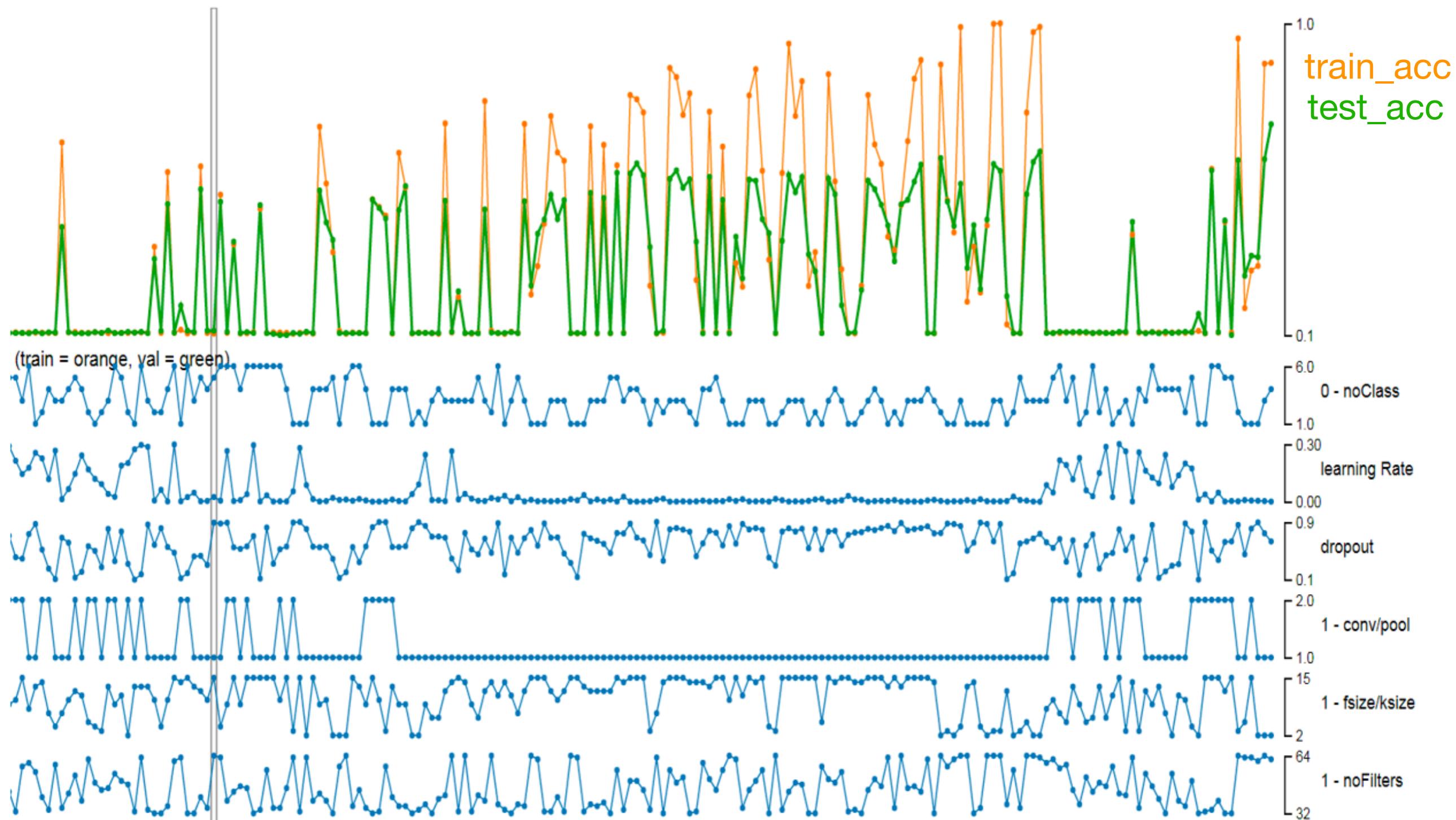
- Conditional
  - SVM -> kernel?
  - RBF kernel -> gamma?



# Hyperparameters

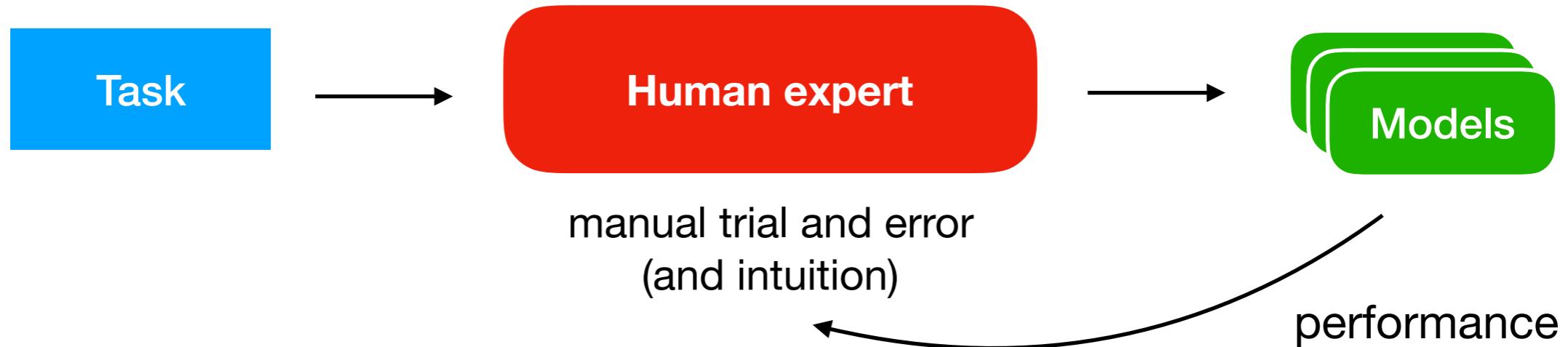
Some are very sensitive, others have very little impact

What is the right strategy to find the right configuration?

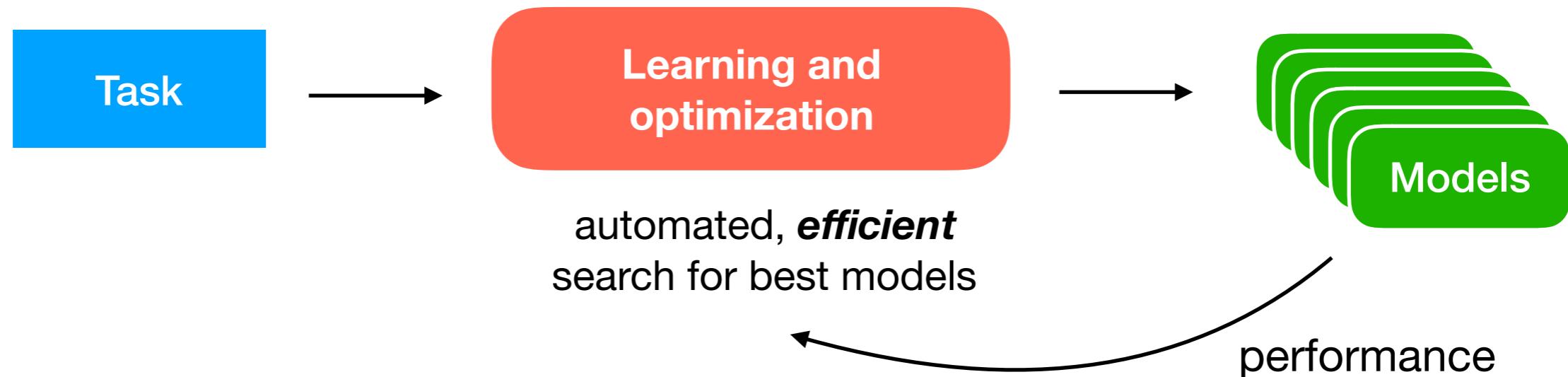


# *Automated machine learning*

## Manual machine learning

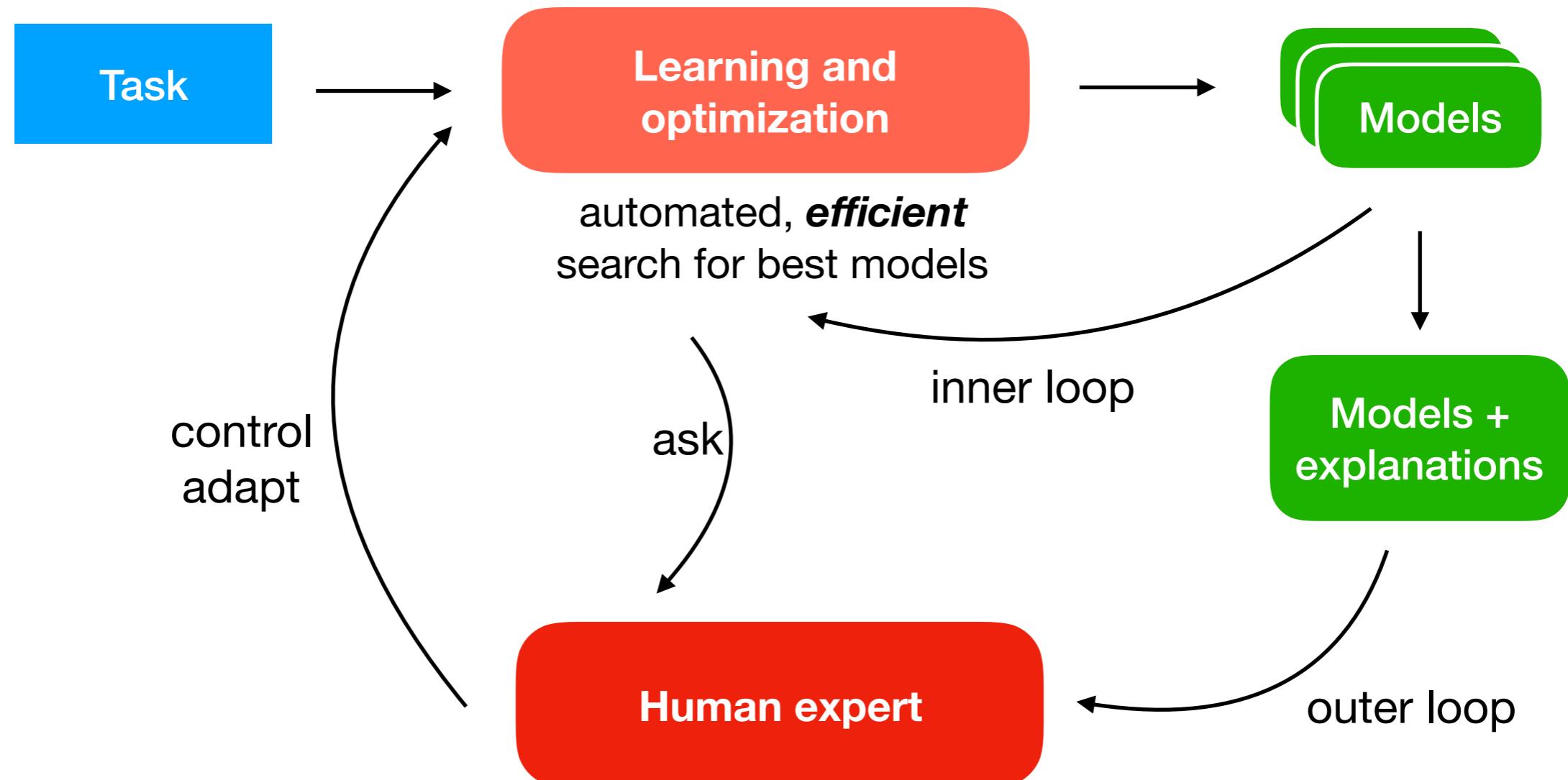


AutoML: build models in a data-driven, objective, and automated way



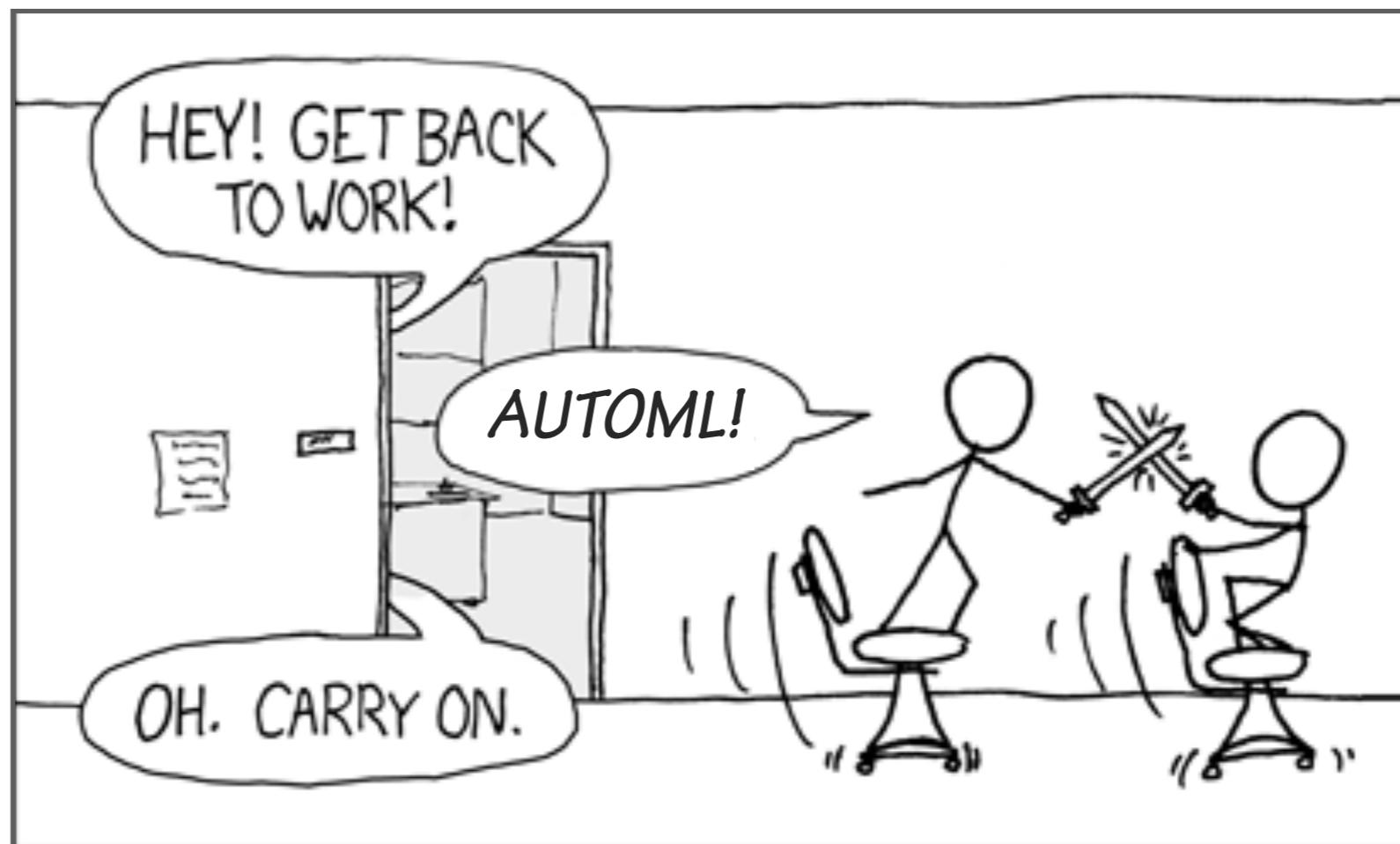
# Semi-Automated machine learning

Human-in-the-loop



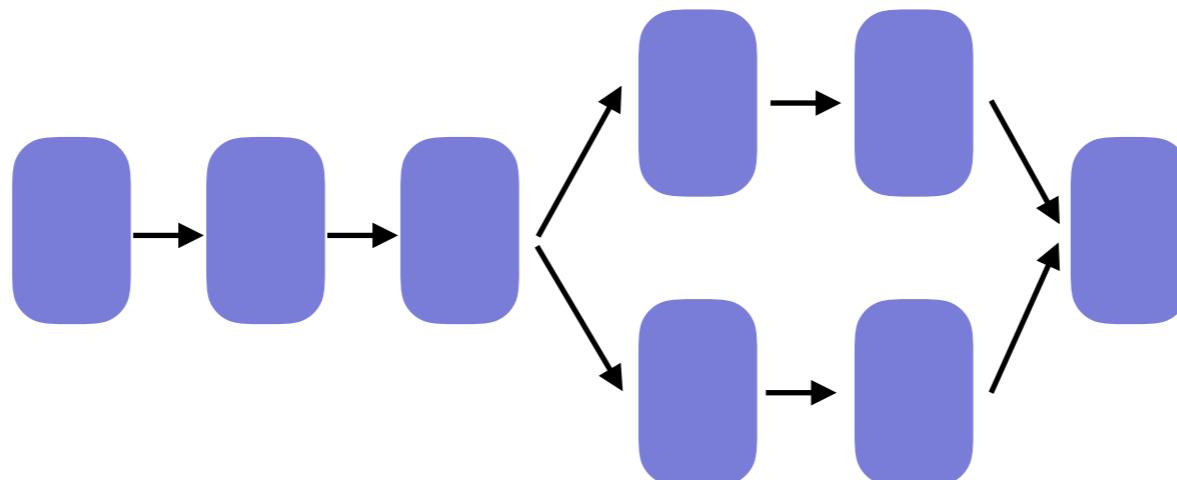
Domain knowledge and human expertise are very valuable  
e.g. unknown unknowns, preference learning,...

*THE DATA SCIENTIST'S #1 EXCUSE FOR  
LEGITIMATELY SLACKING OFF:  
“THE AUTOML TOOL IS OPTIMIZING MY MODELS!”*



# AutoML: subproblems

- **Architecture search:** *represent and search possible architectures*
  - Pipeline operators, neural layers, ...
  - Defines the search space



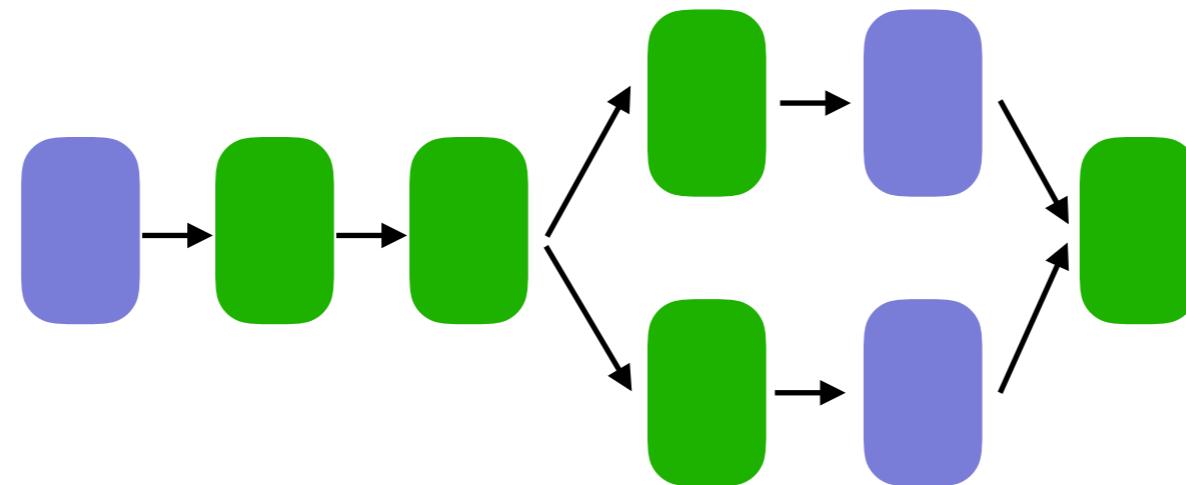
```
make_pipeline(  
    OneHotEncoder(),  
    Imputer(),  
    StandardScaler(),  
    SVC())
```



```
model.add(Conv2D(32, (3, 3))  
model.add(MaxPooling2D((2, 2)))  
model.add(Conv2D(64, (3, 3))  
model.add(MaxPooling2D((2, 2)))  
model.add(Conv2D(64, (3, 3)))
```

# AutoML: subproblems

- **Architecture search:** *represent and search all possible architectures*
- **Hyperparameter optimization:**
  - Which hyperparameters are important? Which values to try?
  - How to *optimize* remaining hyperparameters



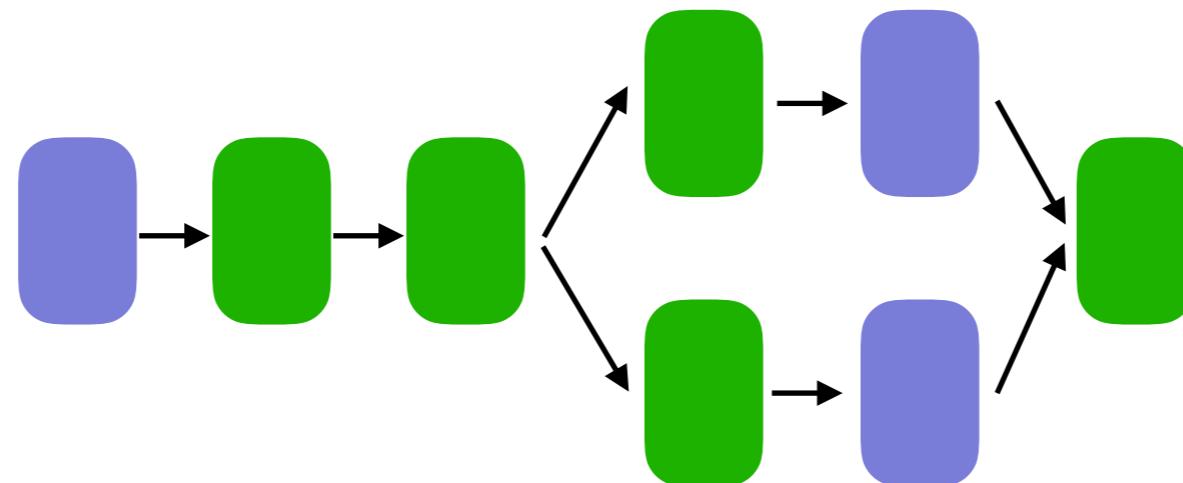
```
hyper_space = {'SVC__C': expon(scale=100),  
               'SVC__gamma': expon(scale=.1)}
```

```
RandomizedSearchCV(pipe, param_distributions=hyper_space, n_iter=200)
```



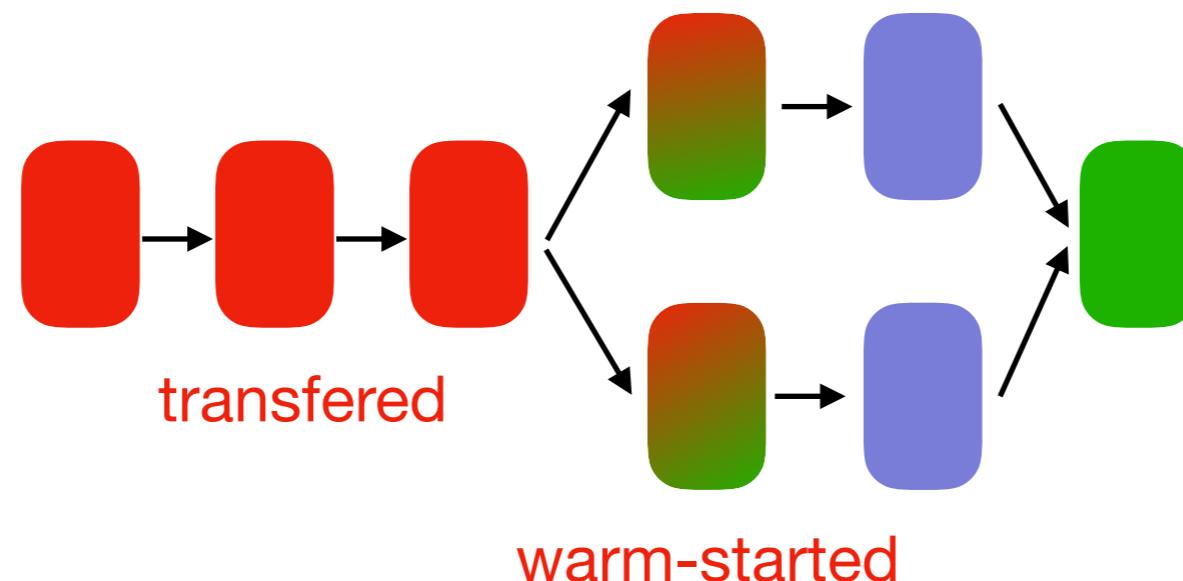
# AutoML: subproblems

- **Architecture search:** *represent and search all possible architectures*
- **Hyperparameter optimization:** *optimize remaining hyperparameters*
- **Objective function:** *what to optimize, can be multi-objective*
  - E.g. high performance but also simplicity, interpretability,...



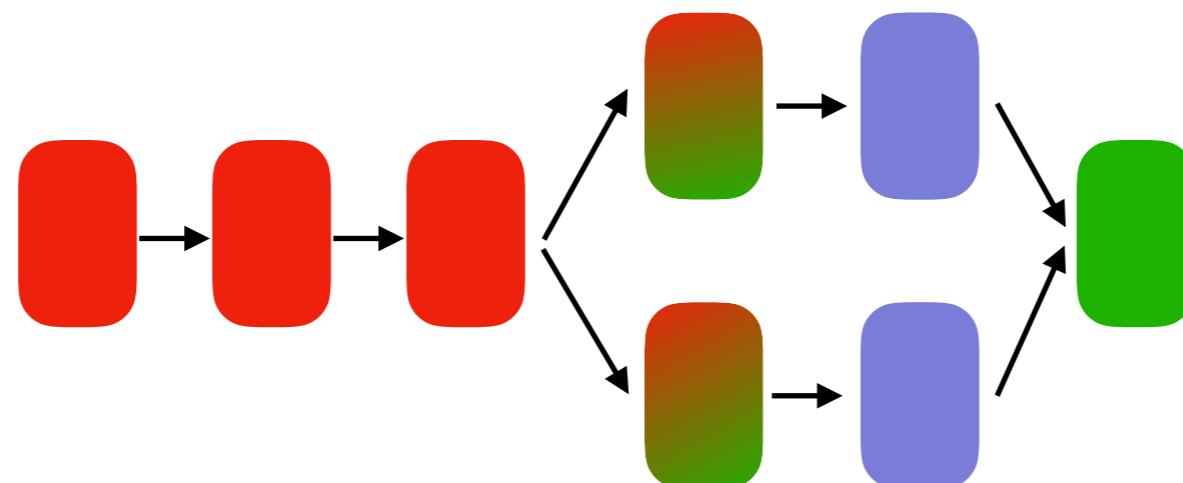
# AutoML: subproblems

- **Architecture search:** represent and search all possible architectures
- **Hyperparameter optimization:** optimize remaining hyperparameters
- **Objective function:** *what* to optimize, can be multi-objective
- **Meta-learning:** how can we transfer experience from previous tasks?
  - Transfer learning: reuse good architectures/configurations
  - Warm starting: start search from promising architectures/configurations
  - ...



# AutoML: subproblems

- **Architecture search**: represent and search all possible architectures
- **Hyperparameter optimization**: optimize remaining hyperparameters
- **Objective function**: what to optimize, can be multi-objective
- **Meta-learning**: how can we transfer experience from previous tasks?
  - Don't start from scratch every time

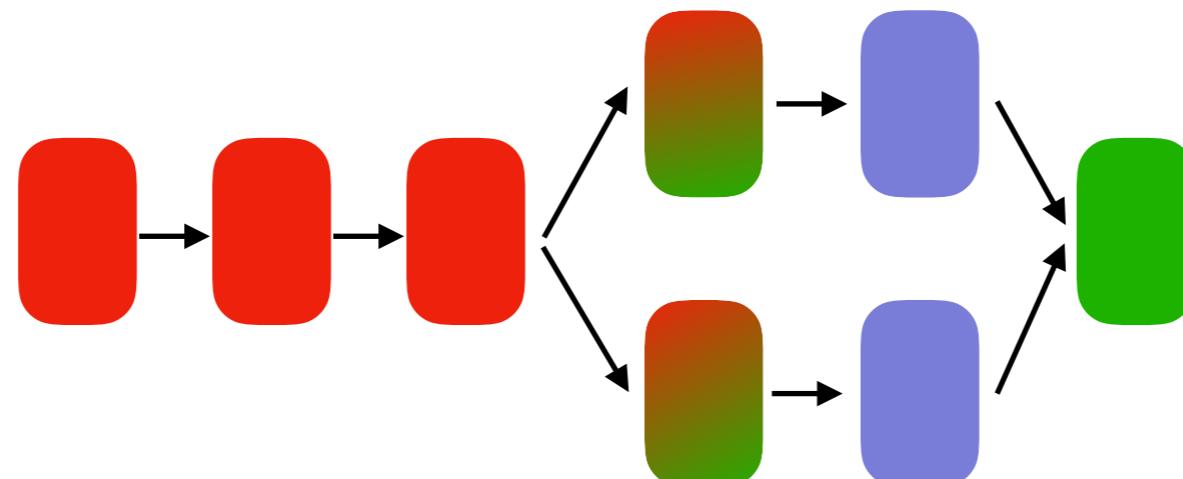


Goal:

```
model = AutoML_Learner().fit(X_train,y_train)
model2 = AutoML_Learner().fit(X2_train,y2_train) → meta-learn
```

# AutoML: subproblems

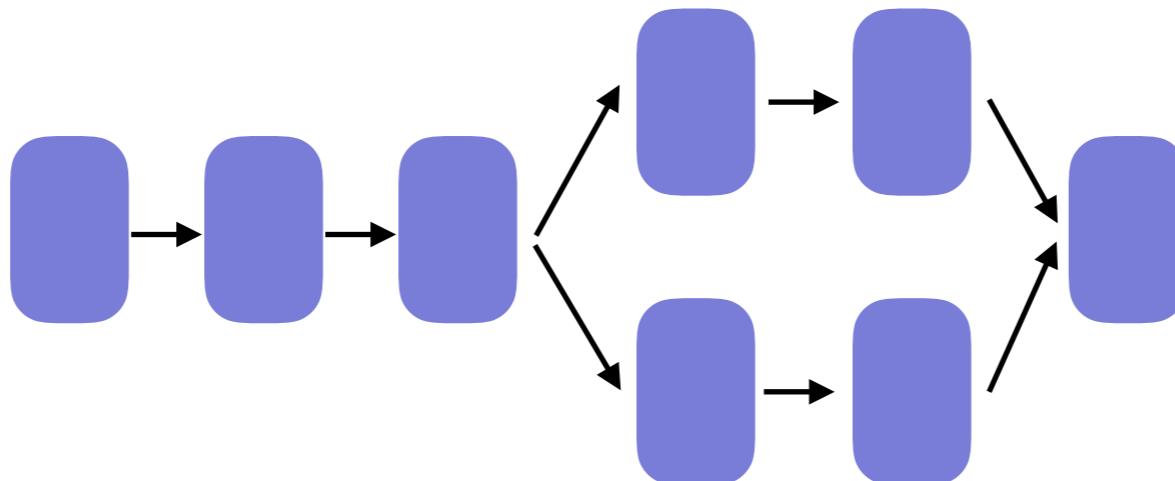
- **Architecture search:** *represent and search all possible architectures*
- **Hyperparameter optimization:** *optimize remaining hyperparameters*
- **Meta-learning:** how can we transfer experience from previous tasks?
  - Don't start from scratch every time



these 3 subproblems can be solved  
*consecutively, simultaneously or interleaved*

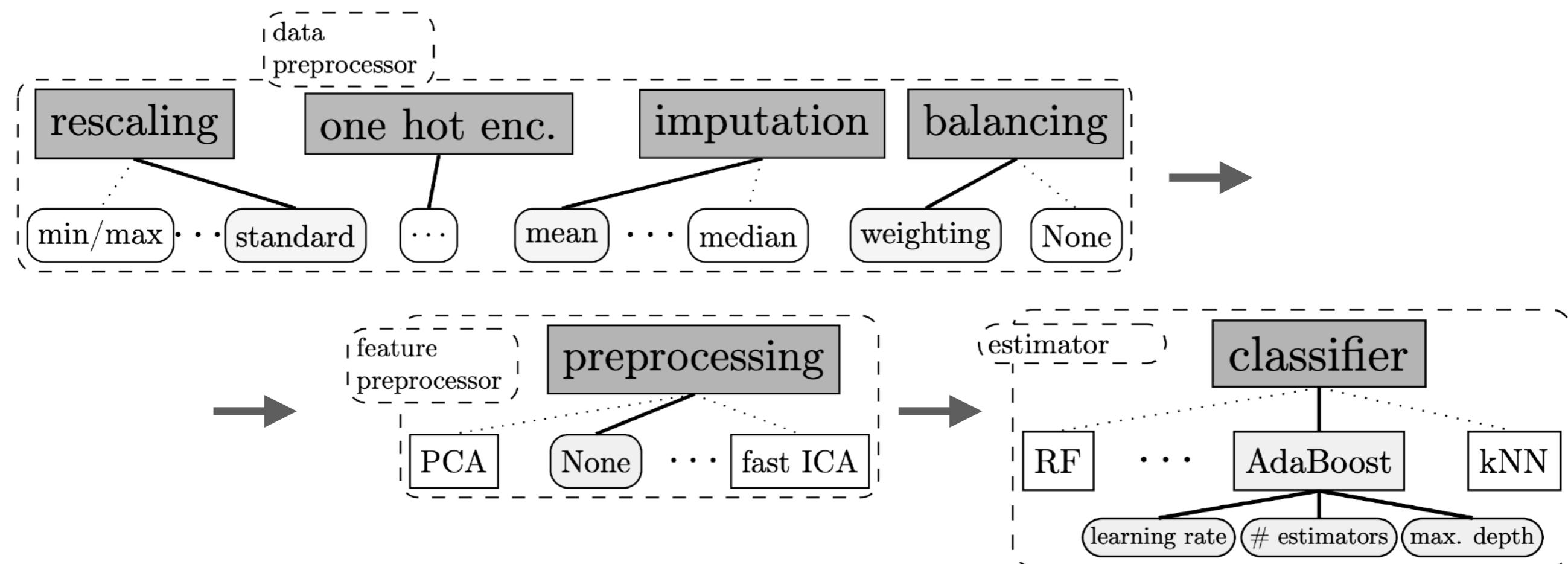
# AutoML: subproblems

- **Architecture search:** *many different approaches*
  - Parameterized (fixed) architectures
  - Reinforcement learning
  - Evolution
  - Monte-Carlo tree search
  - Planning
  - ...



# Parameterized architectures

- Observation: most successful pipelines have a similar structure -> transfer!
- Fixed (linear) architecture, all possible choices encoded as extra hyperparameters
- *Architecture search becomes hyperparameter optimization*



+ smaller search space

- you can't learn entirely new architectures

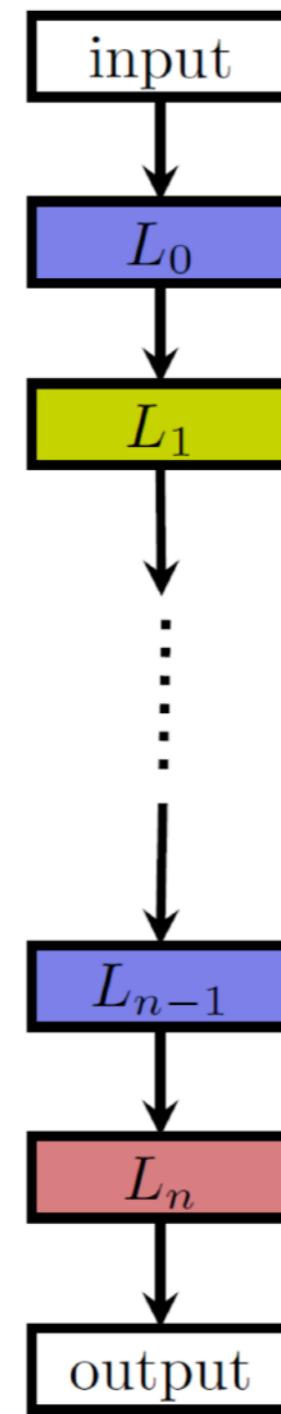
# Neural Architecture Search

## Parameterized Sequential

Choose:

- number of layers
- type of layers
  - dense
  - convolutional
  - max-pooling
  - ...
- hyperparameters of layers

+ easier to search  
- sometimes too simple

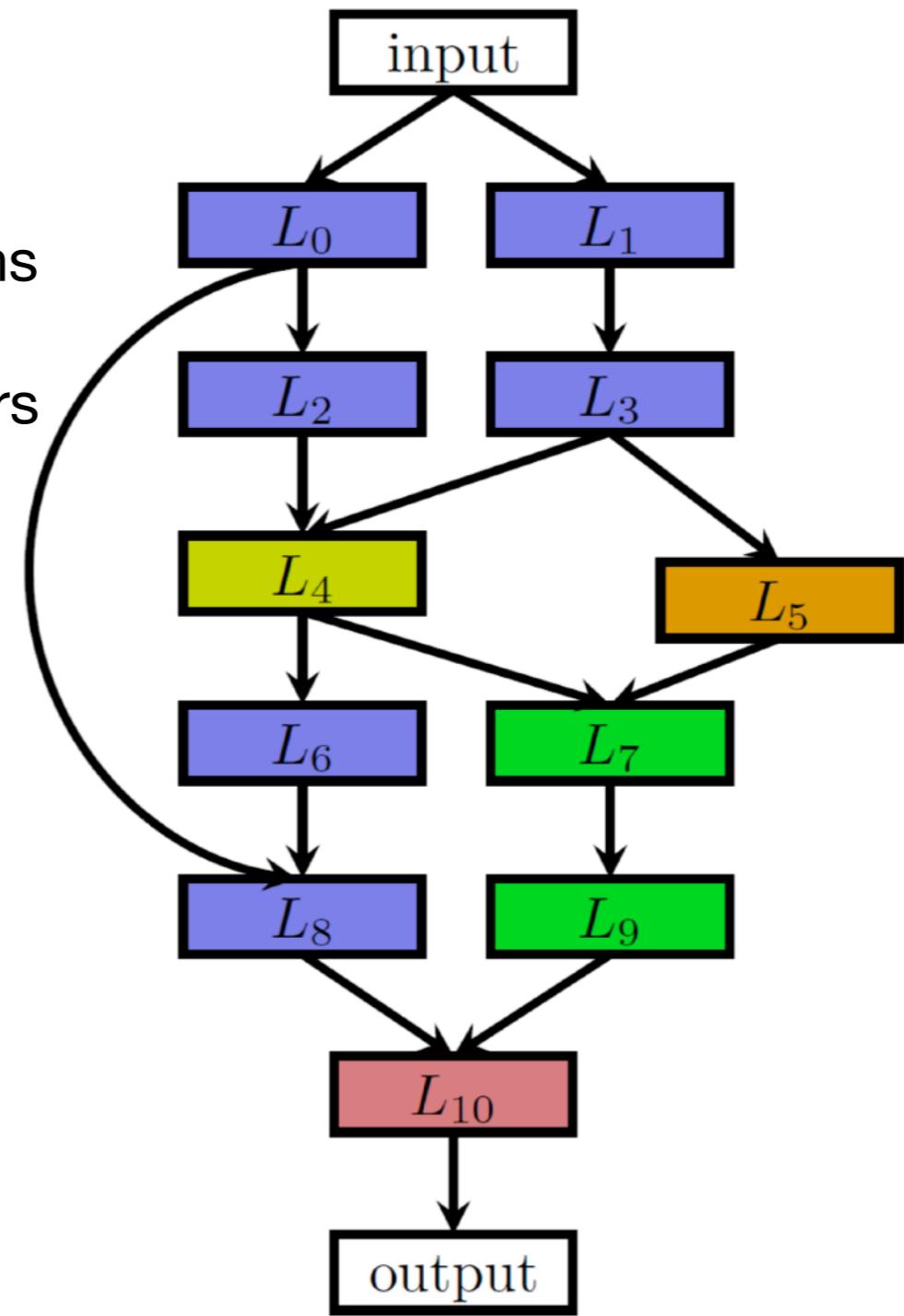


## Parameterized Graph

Choose:

- branching
- joins
- skip connections
- types of layers
- hyperparameters of layers

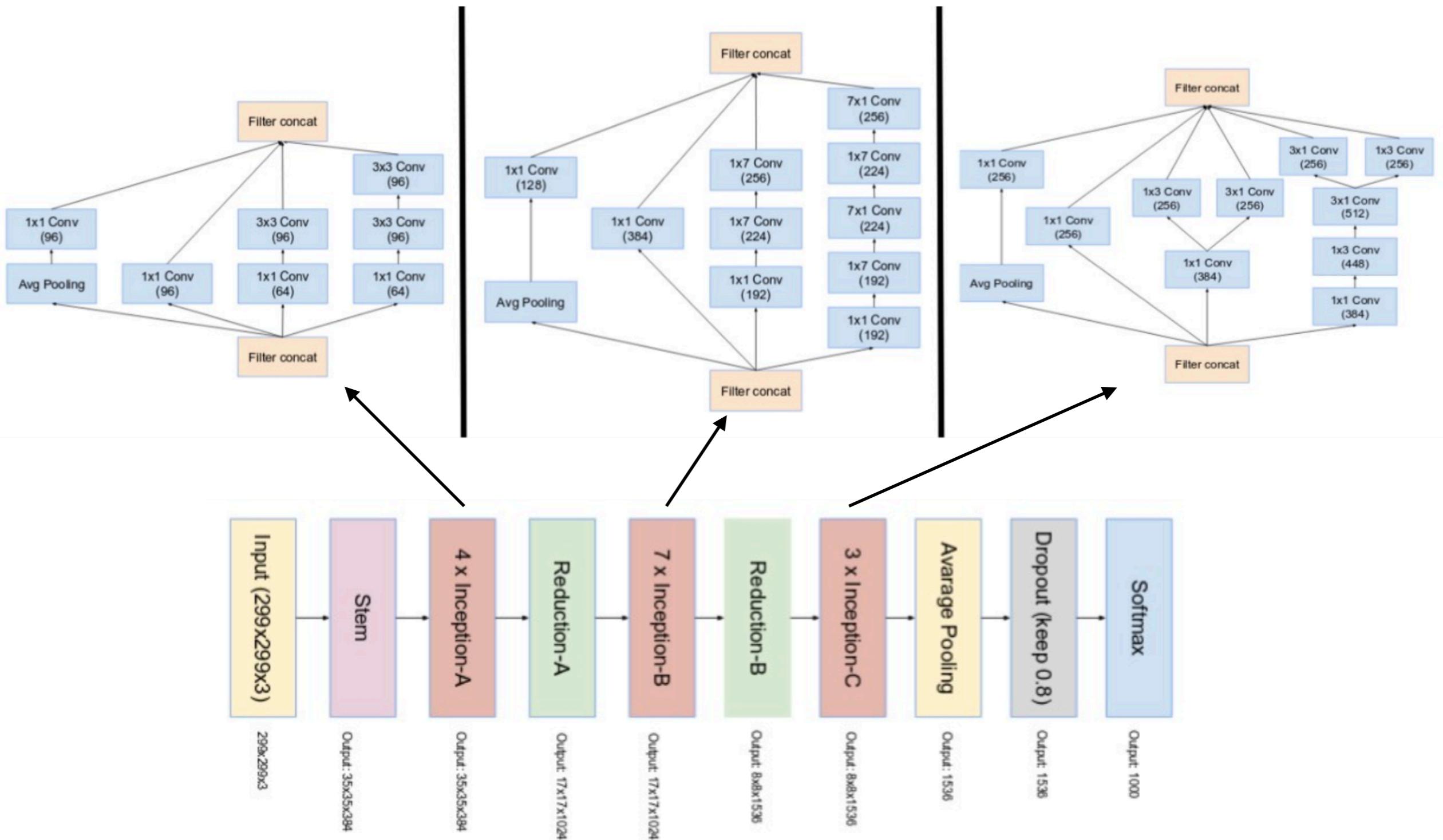
+ more flexible  
- much harder to search



# Neural Architecture Search

Observation: successful deep networks have repeated motifs (cells)

e.g. Inception v4:

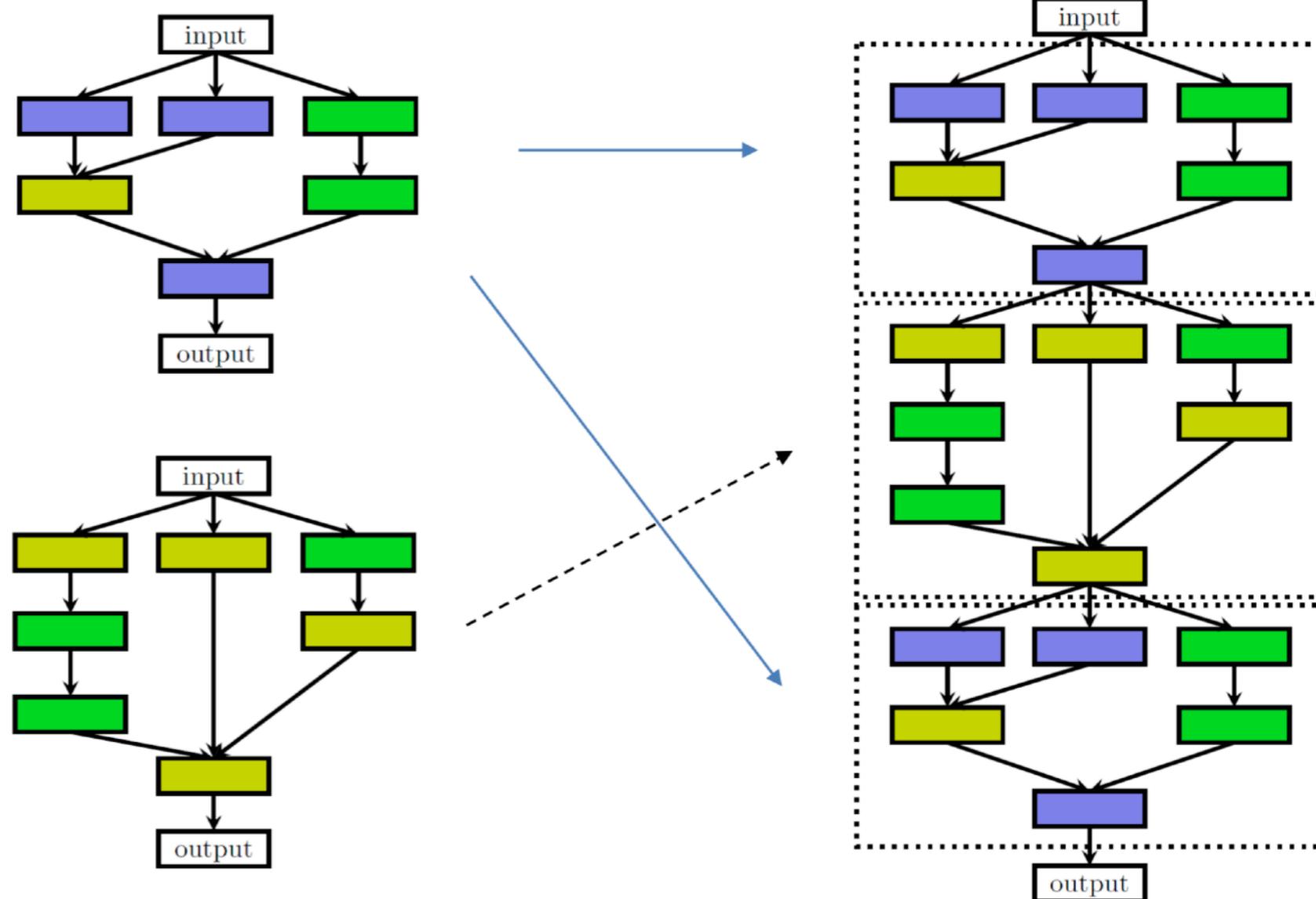


# Neural Architecture Search

Observation: successful deep networks have repeated motifs (cells) -> transfer!

## Cell search space

- parameterized building blocks (*cells*)
  - stack cells together in macro-architecture
    - usually a chain
- + smaller search space  
+ cells can be learned on a small dataset & transferred to a larger dataset  
- you can't learn entirely new architectures

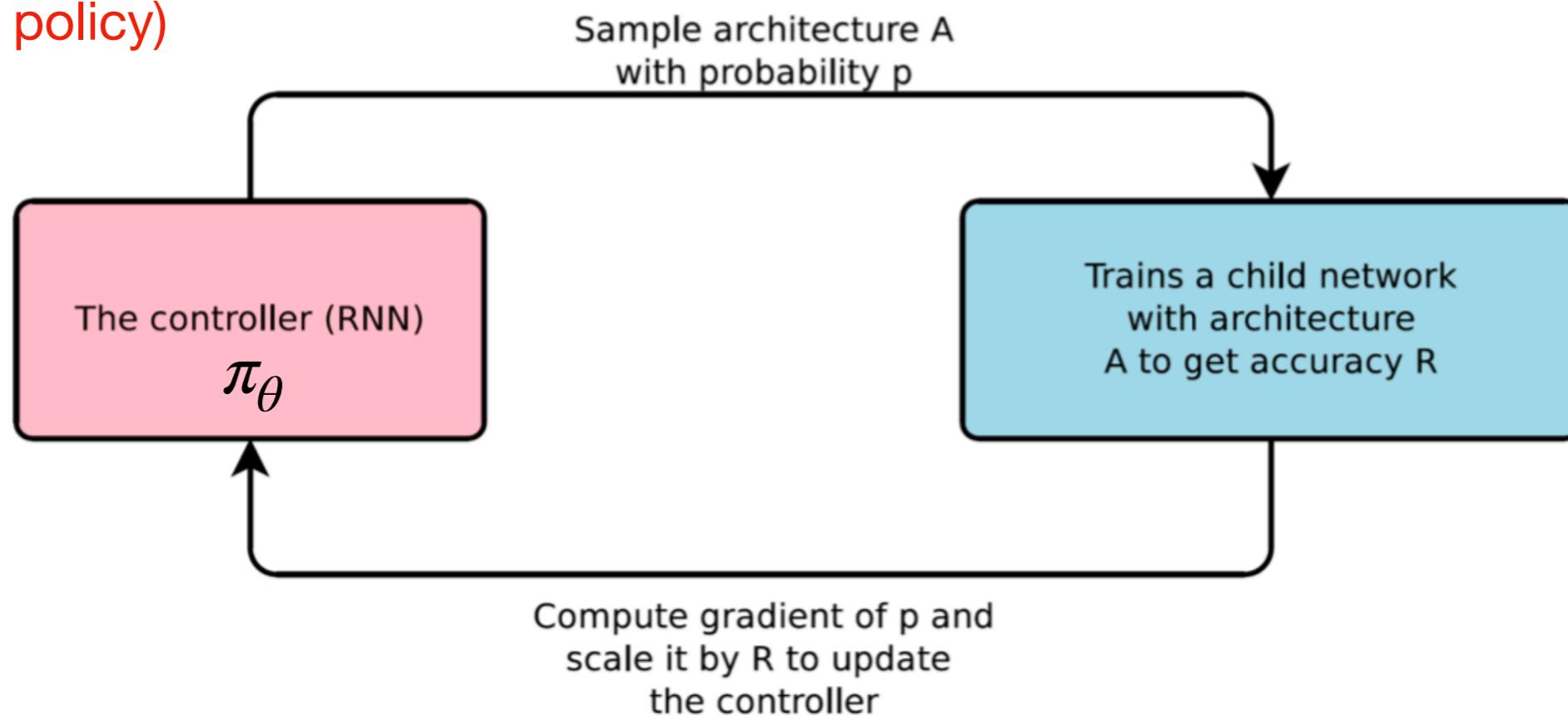


# NAS with Reinforcement learning

Build network step-by-step, learn general strategy (policy)

Actions:  
add layer, change filter size,...  
(according to policy)

States:  
Possible networks



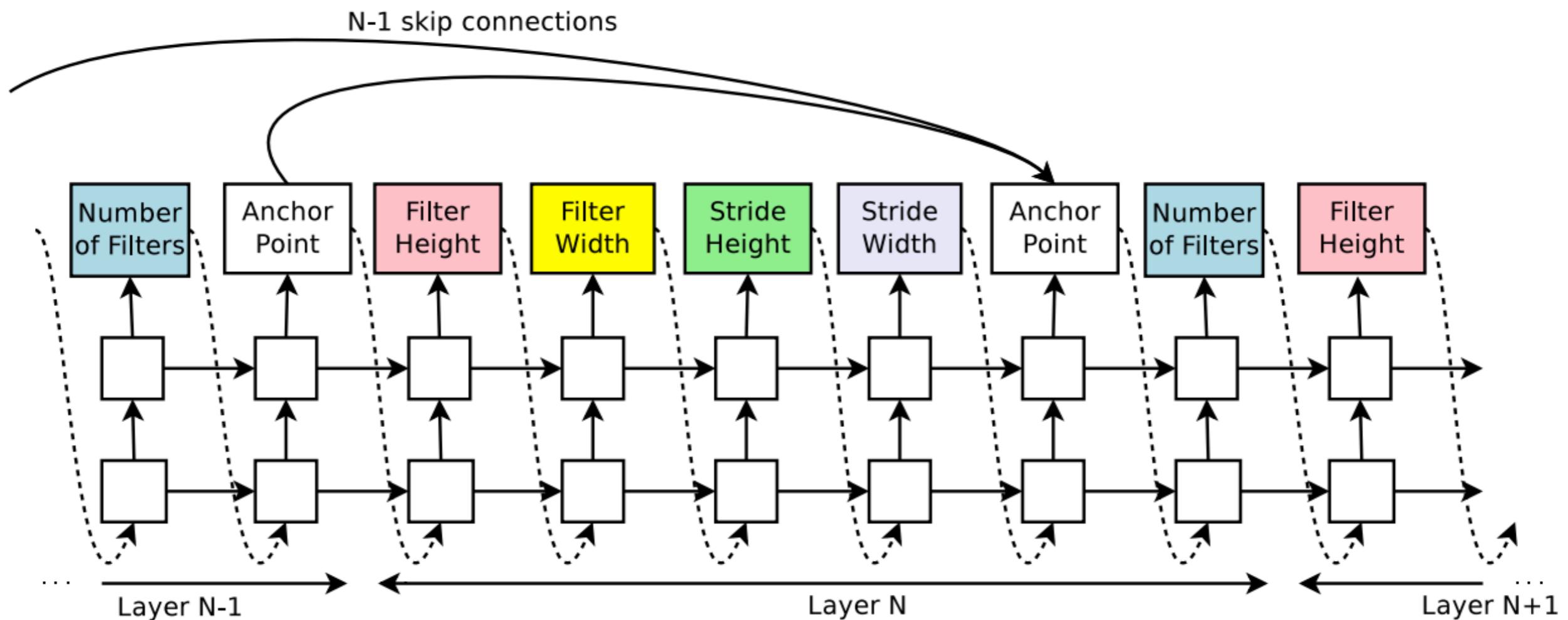
Update RNN with policy gradient

Reward:  
Network performance  
after training

# NAS with Reinforcement learning

2-layer LSTM (REINFORCE), chains of convolutional layers

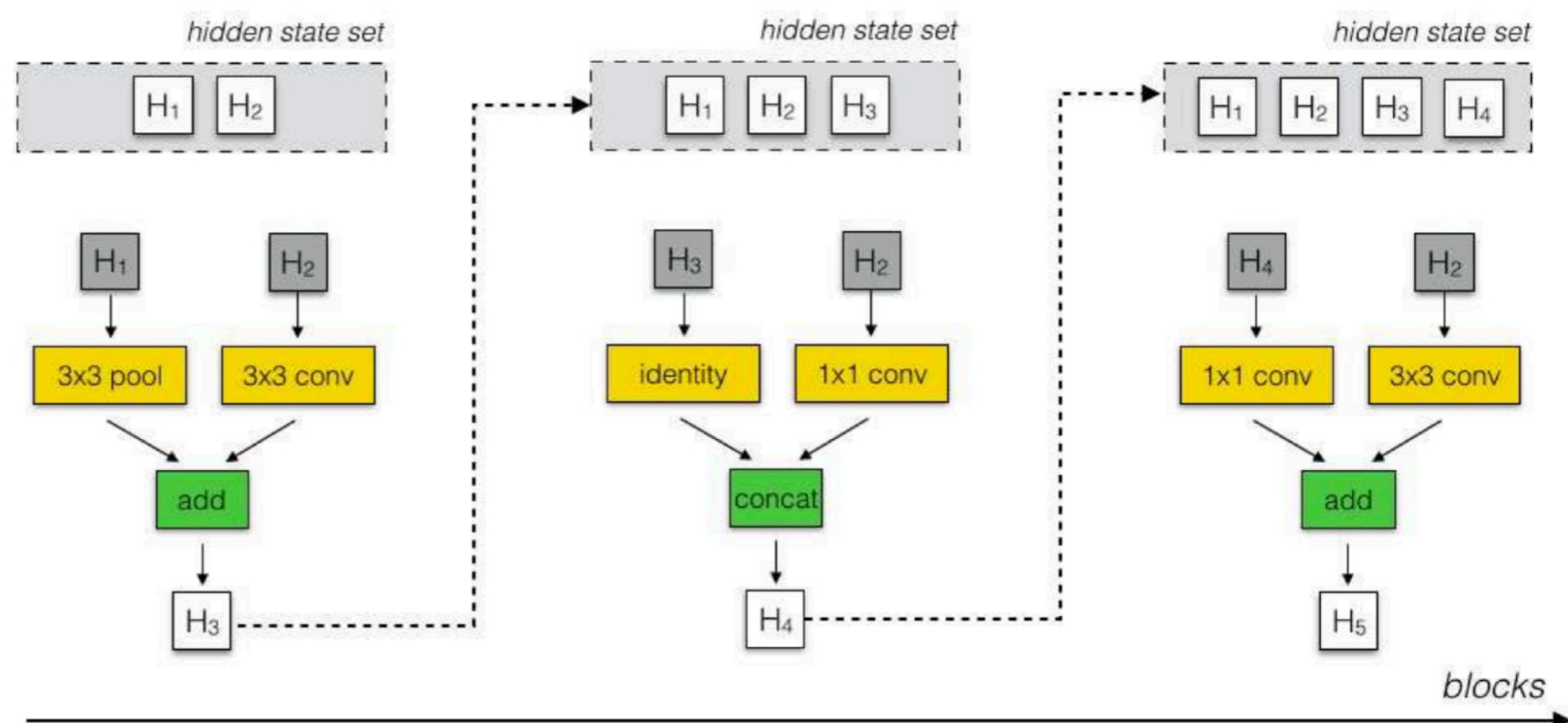
- State of the art on CIFAR-10, Penn Treebank
- **800 GPUs for 3-4 weeks**, 12800 architectures



# NAS with Reinforcement learning

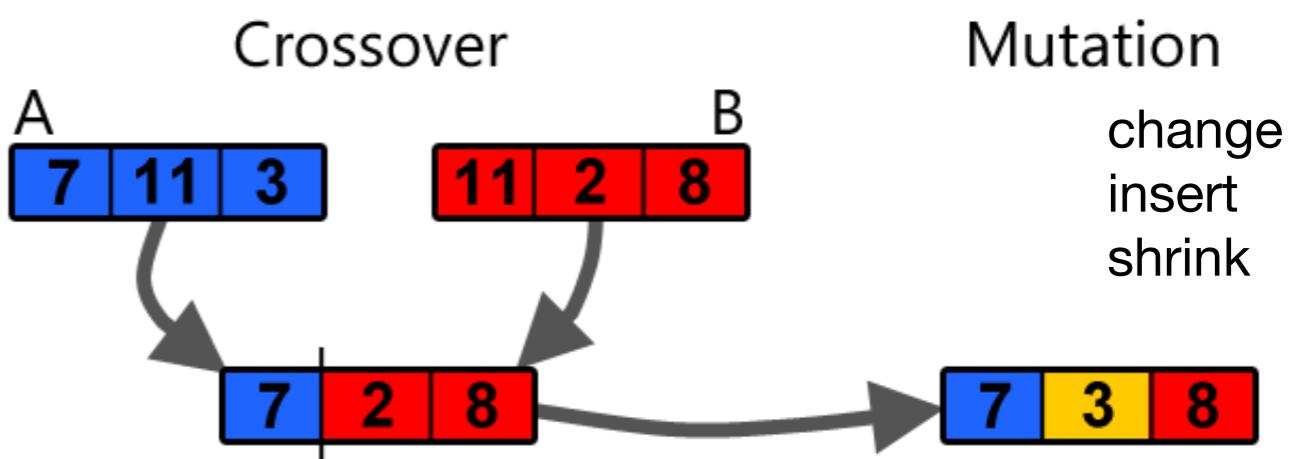
1-layer LSTM (PPO), cell space search

- State of the art on ImageNet
- **450 GPUs, 1 week**, 20000 architectures
- Cell construction:
  - Select existing layers (hidden states, e.g. cell input)  $H_i$  to build on
  - Add operation (e.g. 3x3conv) on  $H_i$
  - Combine into new hidden state (e.g. concat, add,...)
- Iterate over  $B$  blocks

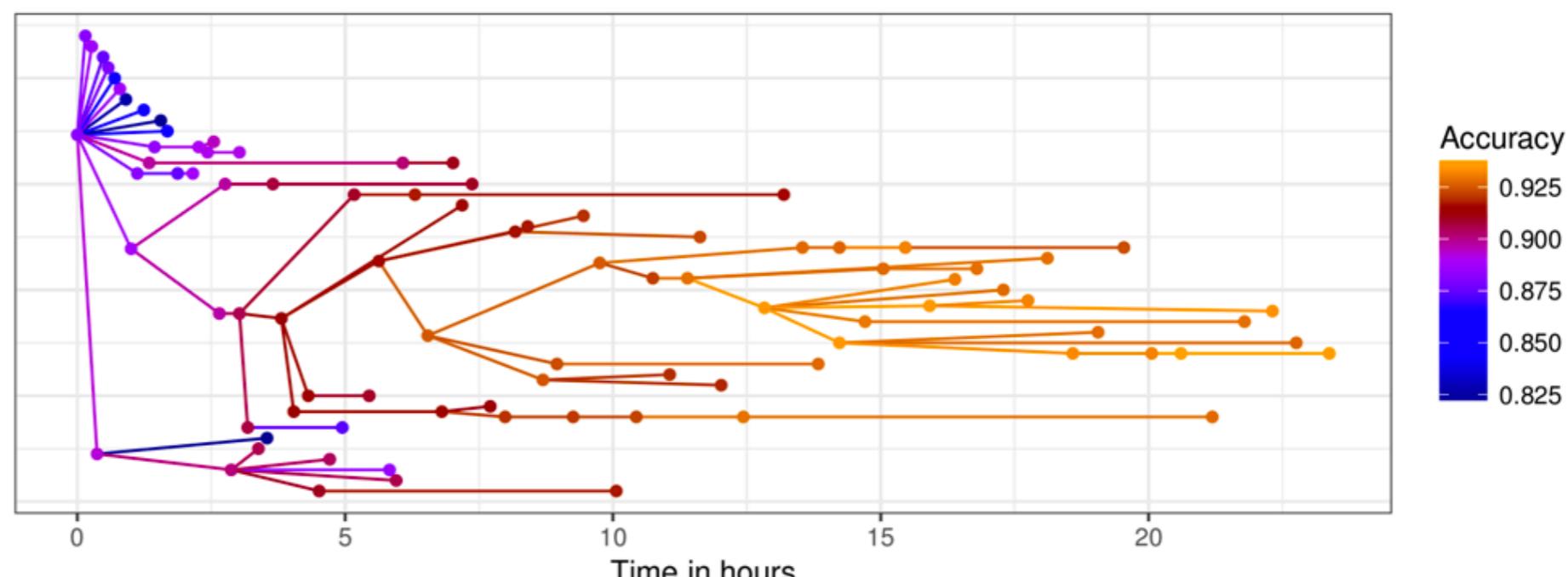


# Evolving pipelines

- Start with initial architecture
- Best architectures *evolve*: cross-over or mutation
  - Per generation (TPOT)
  - Asynchronous (GAMA)
- Tree-shaped pipelines
- No fixed maximal length

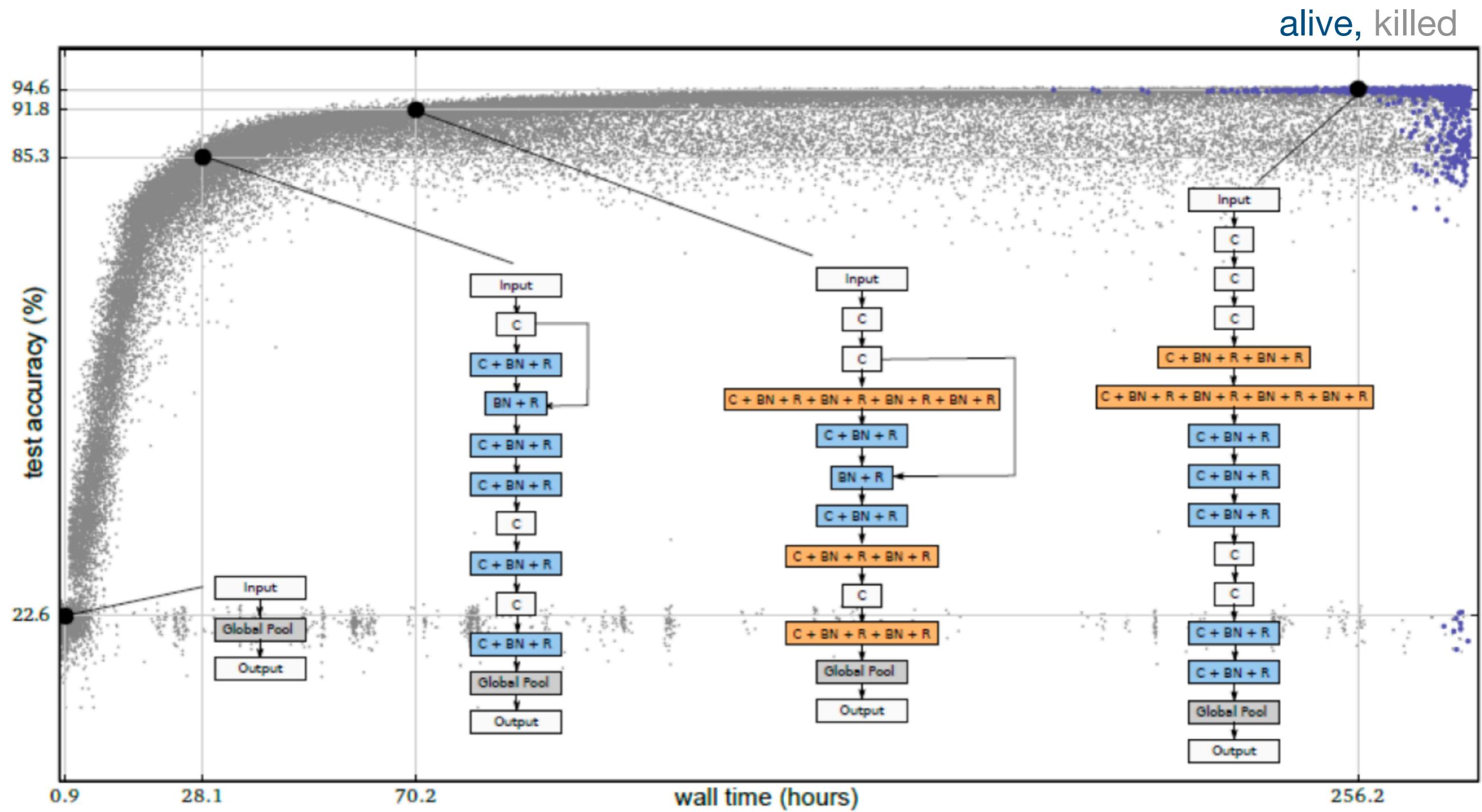


+ more flexible  
- larger search space,  
can be slower



# Neuro-evolution

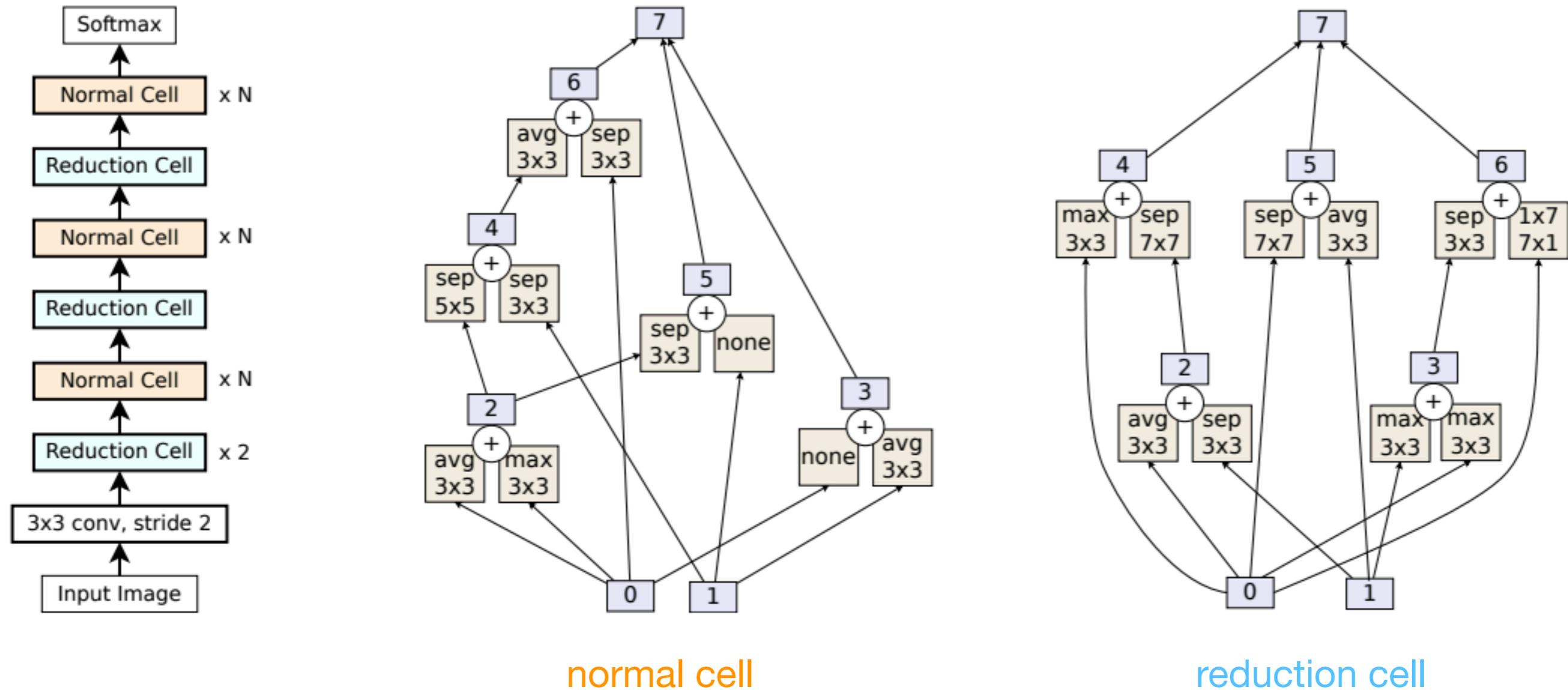
- same idea: learn the neural architecture through evolution
- mutations: add, change, remove layer



# Neuro-evolution

*AmoebaNet*: State of the art on ImageNet, CIFAR-10

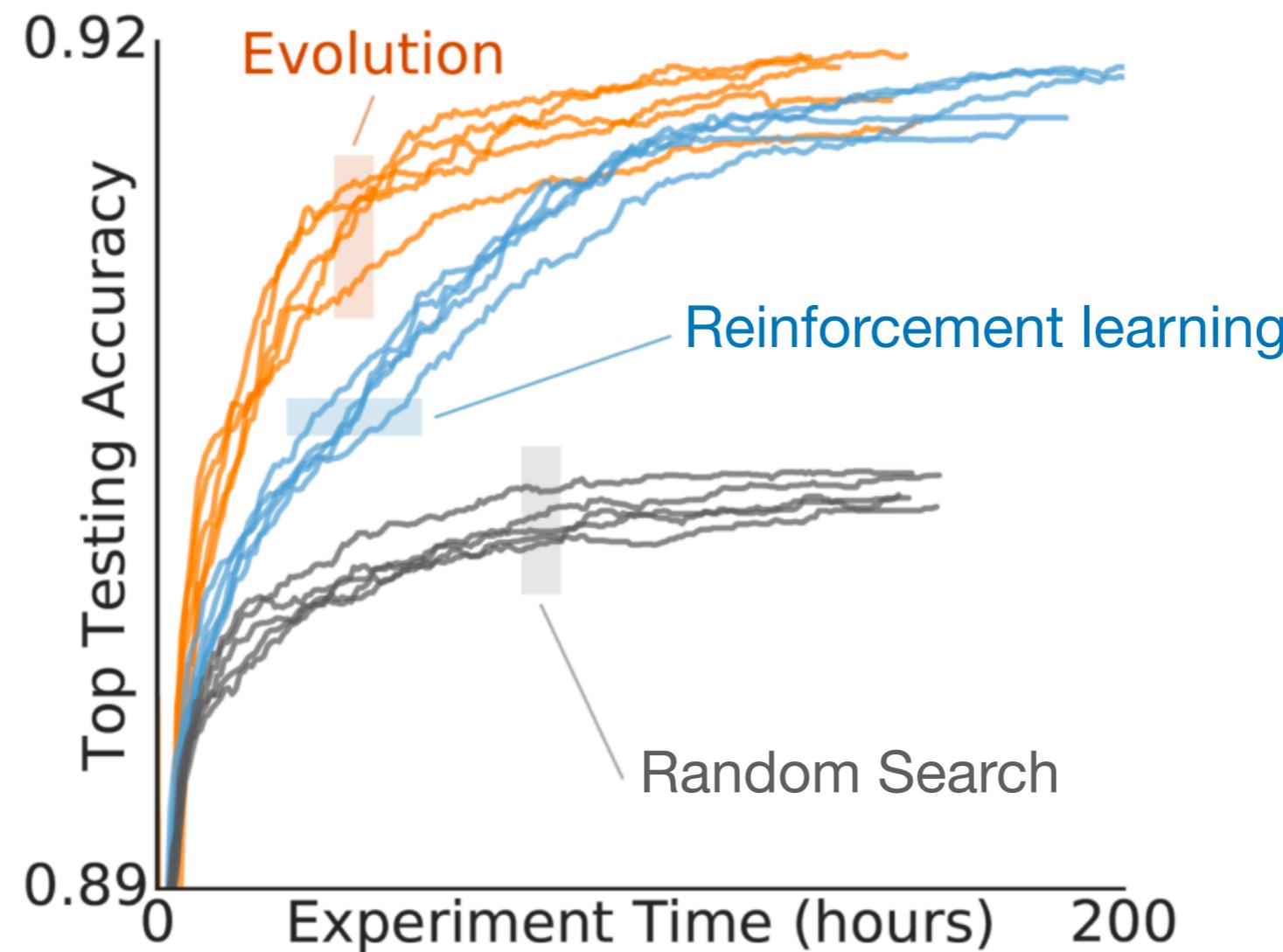
- Cell search space, aging evolution (kill oldest networks)



# Neuro-evolution

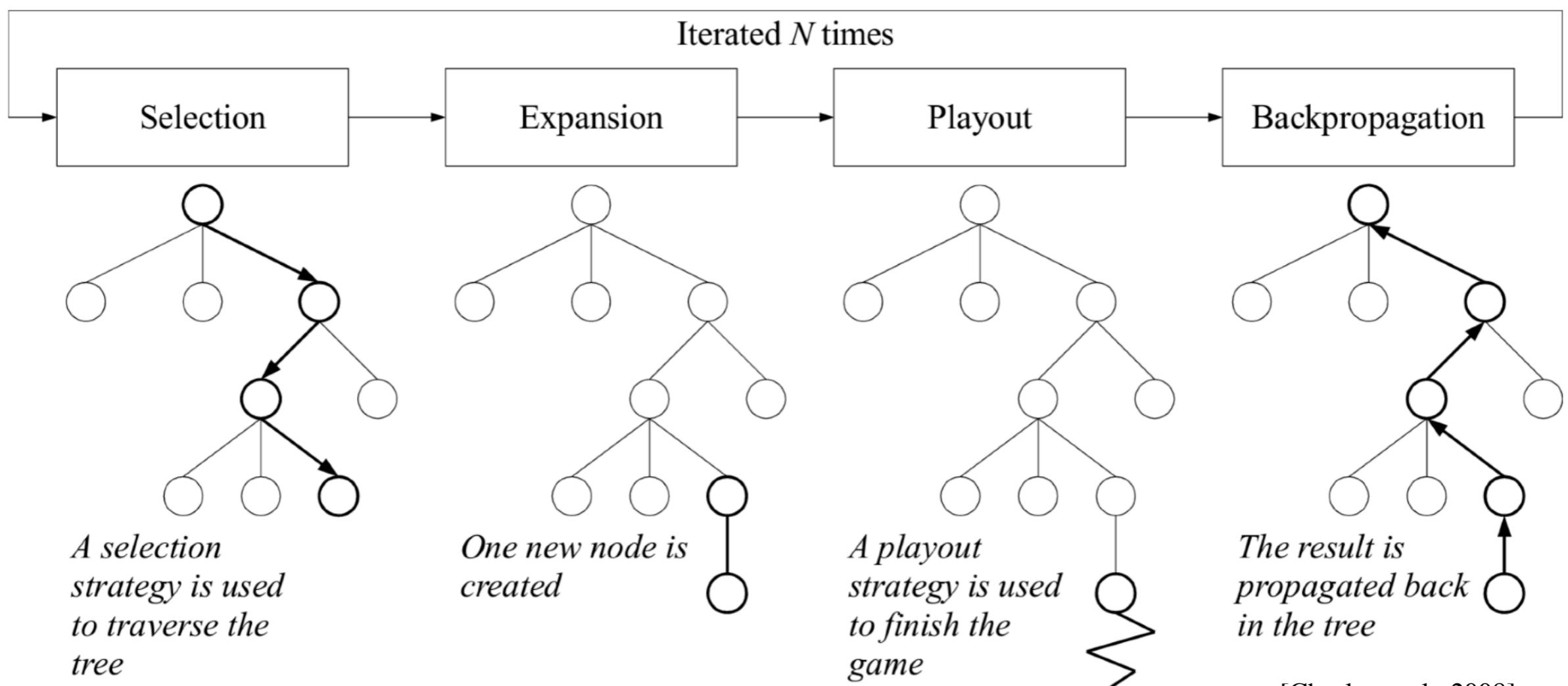
*AmoebaNet*: State of the art on ImageNet, CIFAR-10

- Cell search space + aging evolution (kill oldest networks)
- More efficient than reinforcement learning



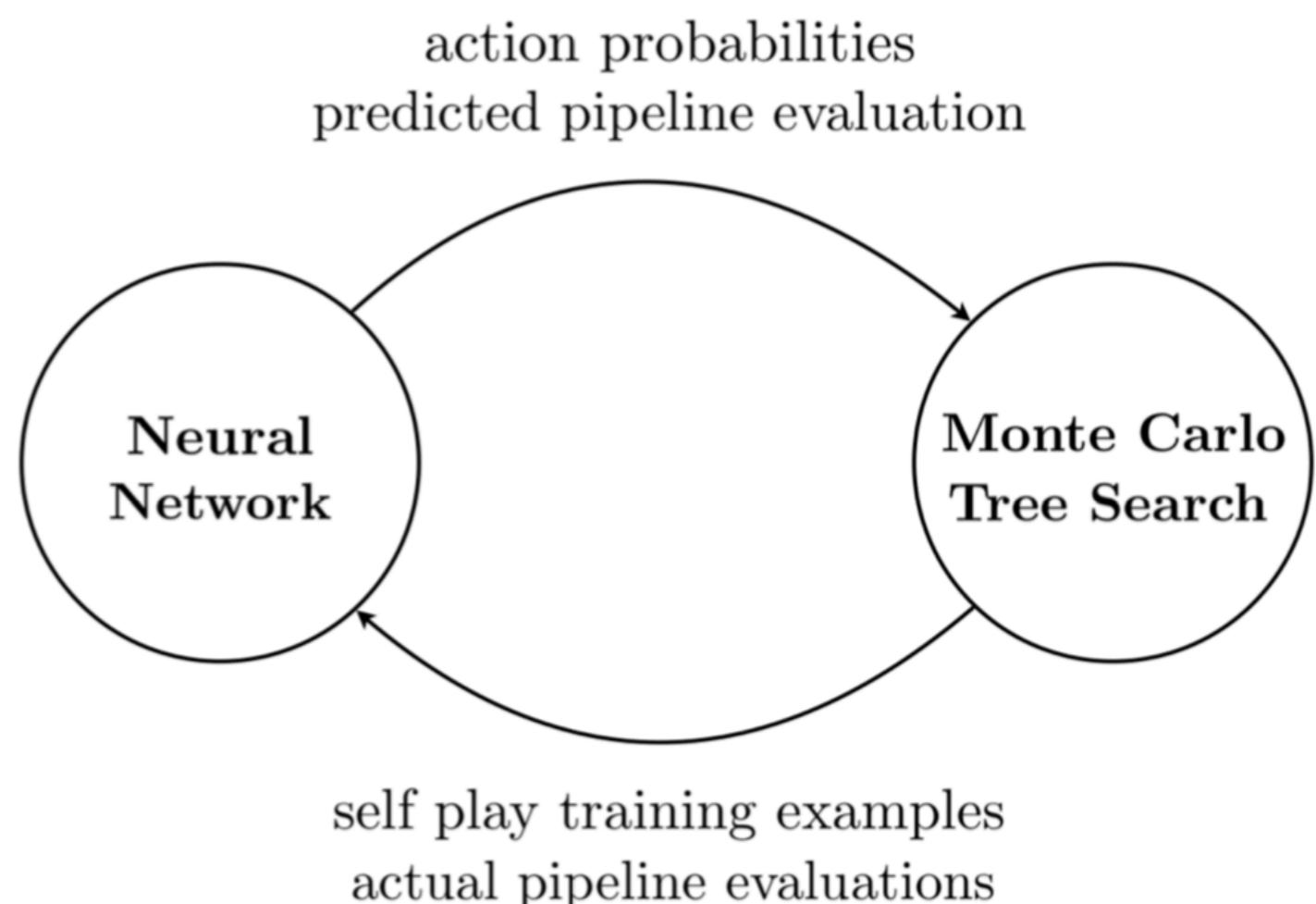
# Monte Carlo Tree Search (MCTS)

- Tree of all possible pipelines (1 path = 1 pipeline)
- Select partial pipeline from tree of all pipelines
- Add one new operator, do random completion to estimate accuracy
- Update nodes according to performance



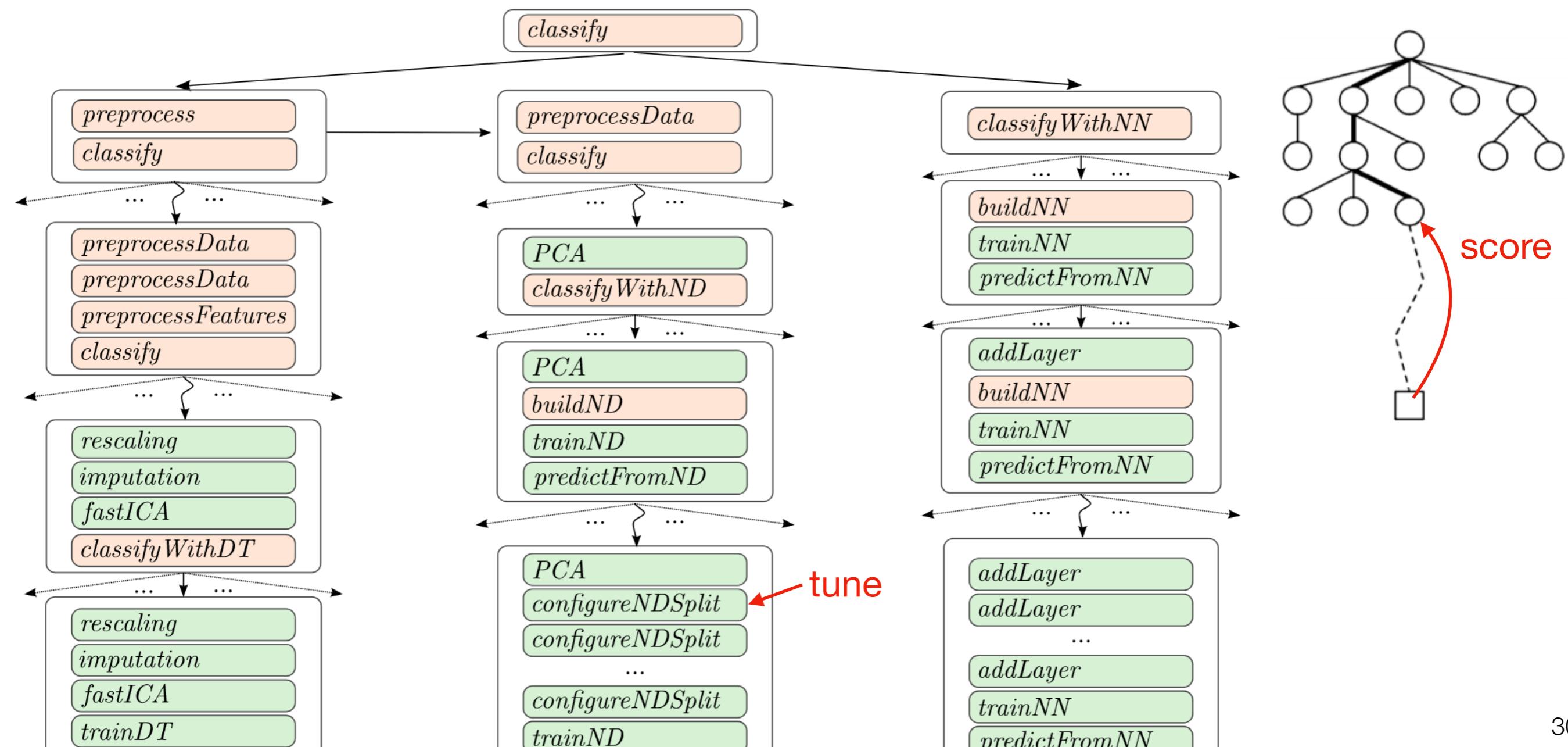
# MCTS + reinforcement learning

- Build pipelines by inserting, deleting, replacing components (actions)
- *Self-play:*
  - Monte Carlo Tree Search builds pipelines given action probabilities
  - Neural network (LSTM) Predicts pipeline performance



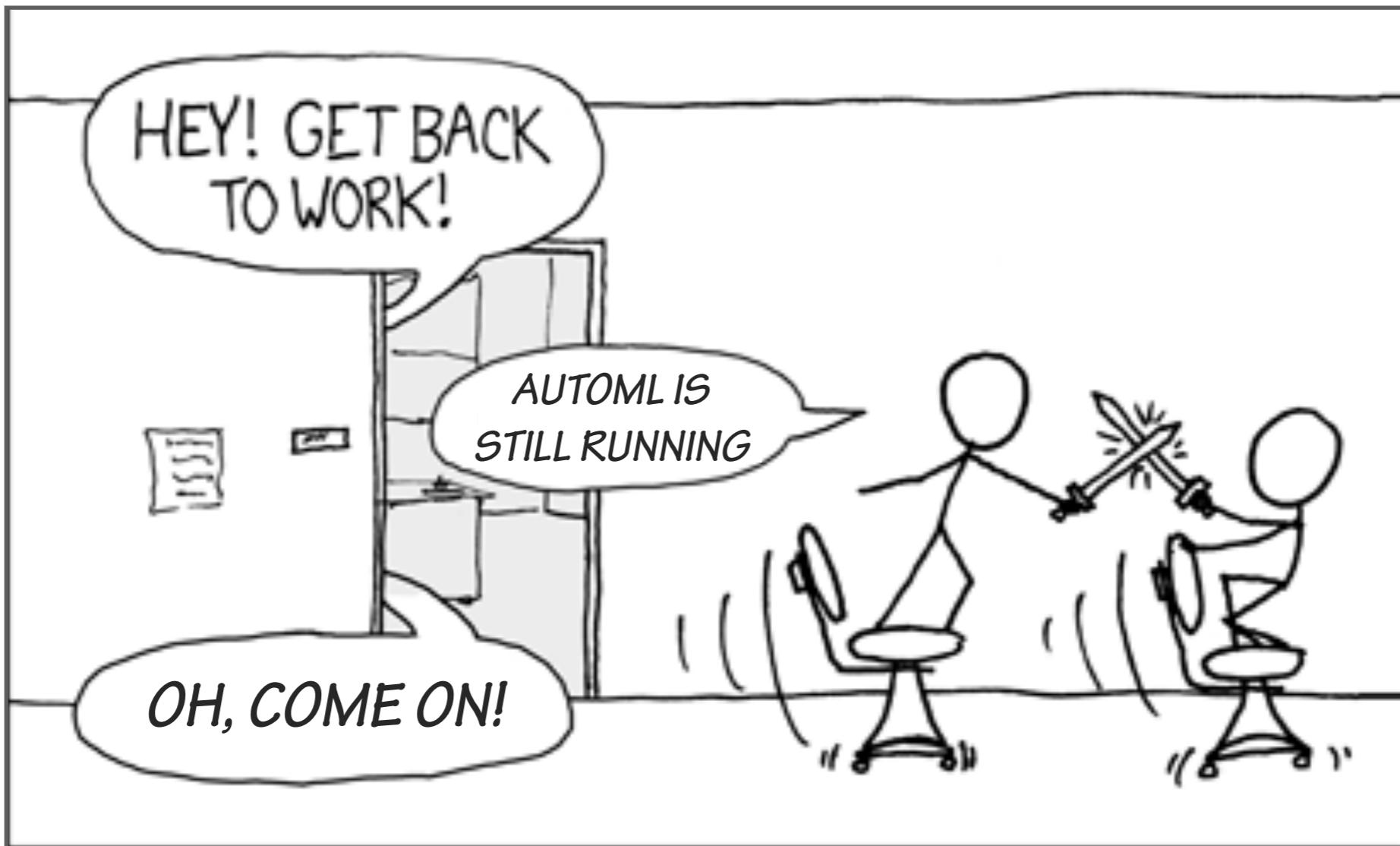
# Hierarchical planning

- Use planning to 'plan' pipeline, e.g. with best first search
- Use random path completion (as in MCTS) to evaluate each node



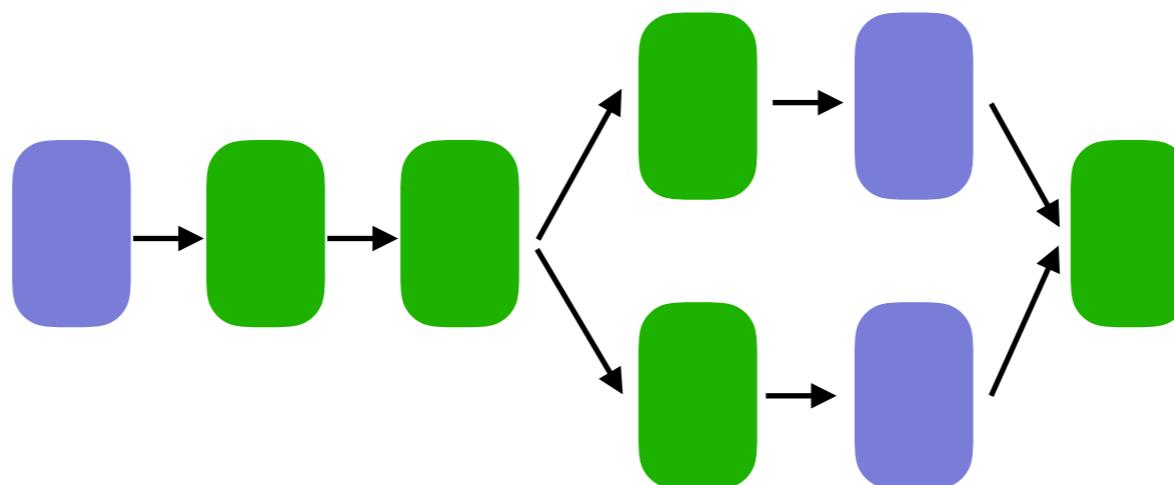
# Hyperparameter Optimization

Searching the hyperparameter space **efficiently**



# AutoML: subproblems

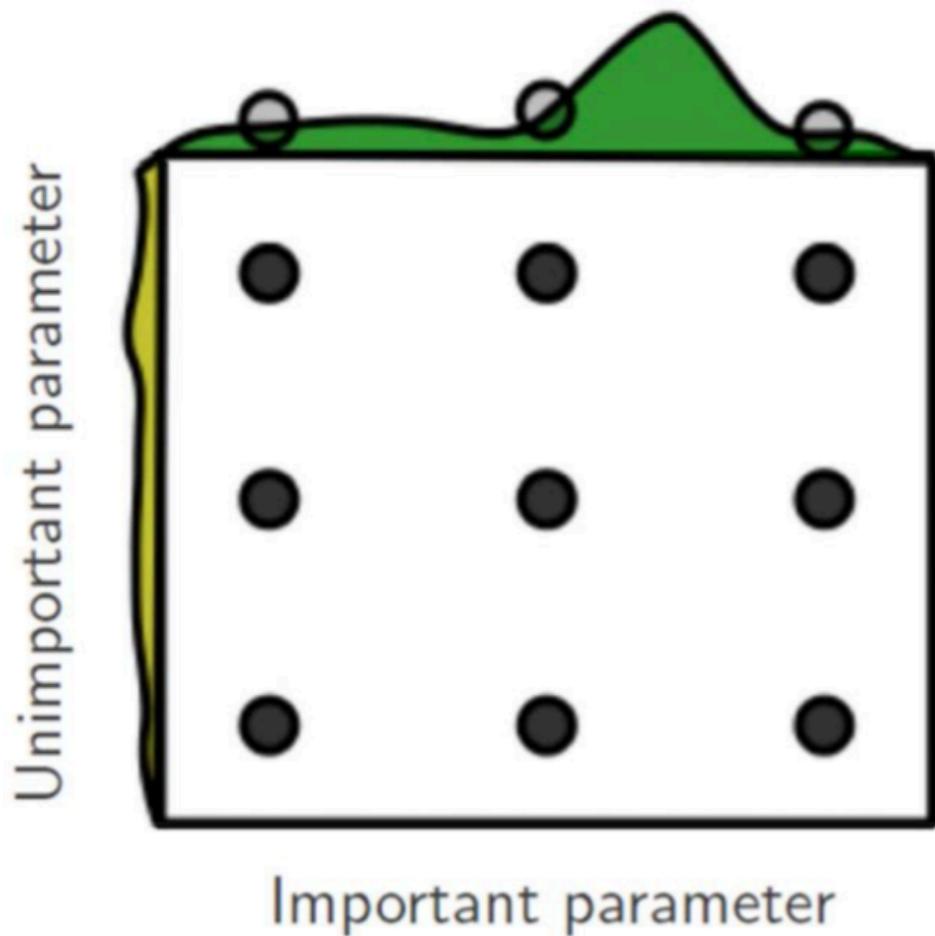
- **Hyperparameter optimization:** *many different approaches*
  - Random search
  - Bayesian optimization
  - Population-based methods (evolution)
  - Multi-fidelity optimization
  - Differentiable optimization
  - ...



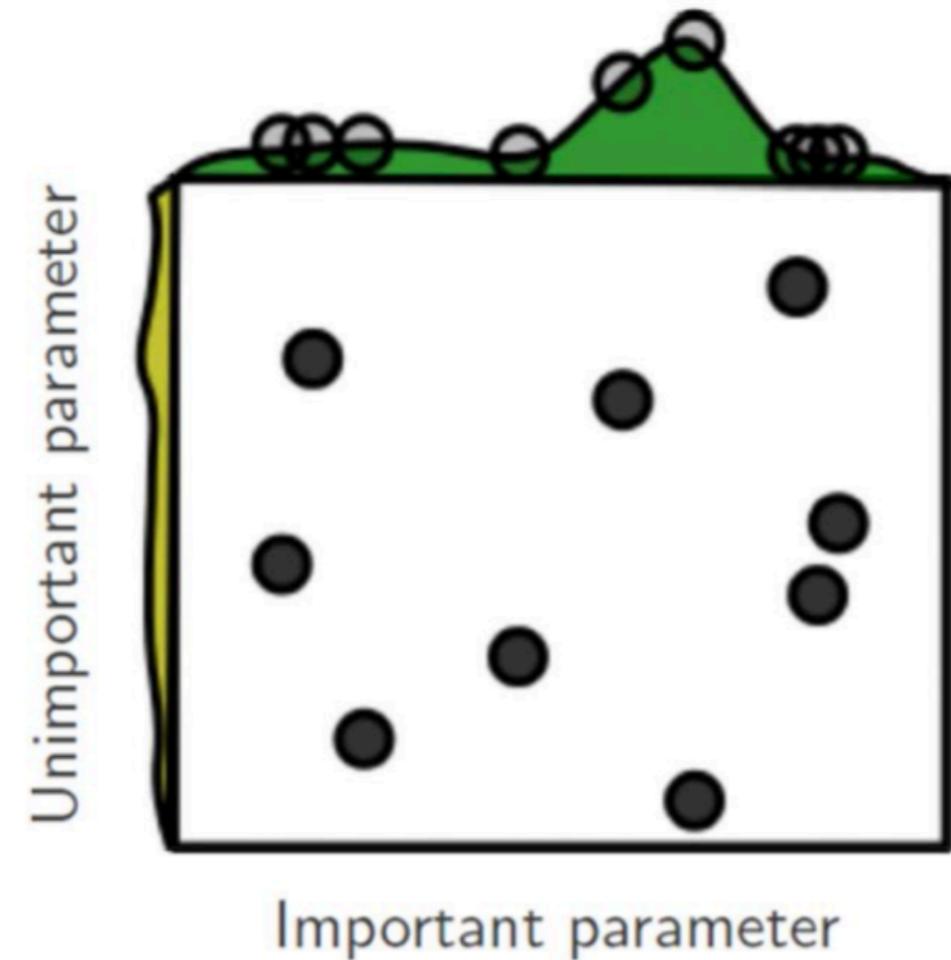
# Random search

- Handles unimportant dimensions better than grid search
- Easily parallelizable, but uninformed (no learning)

Grid Layout

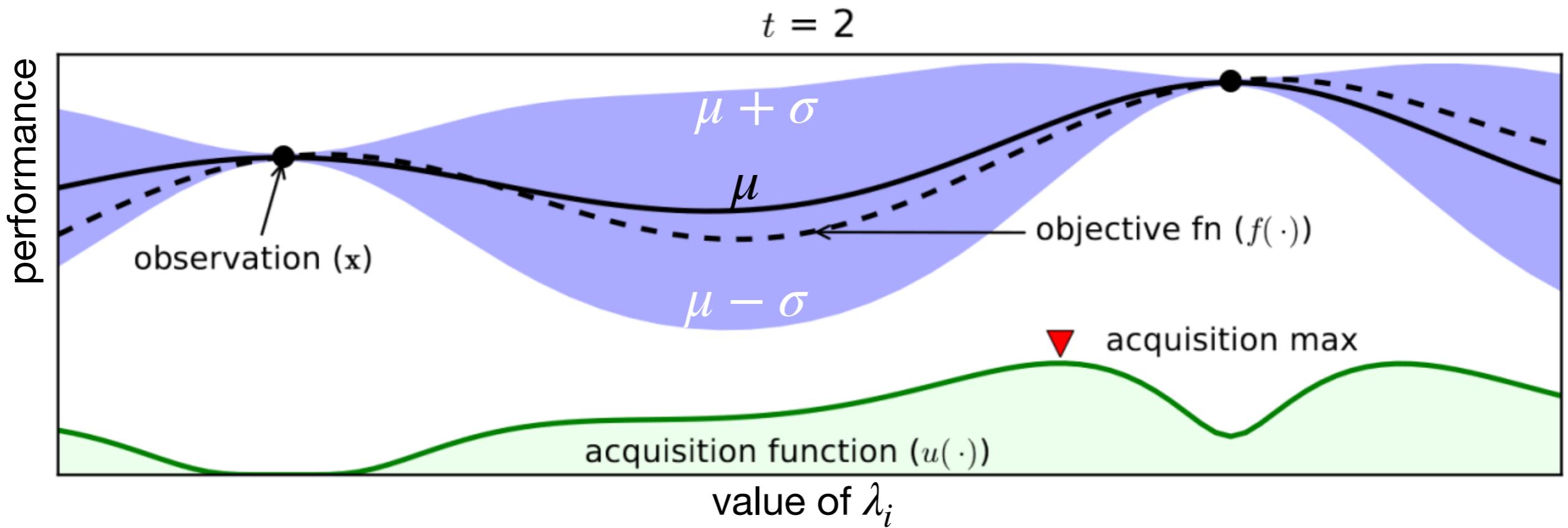


Random Layout



# Bayesian Optimization

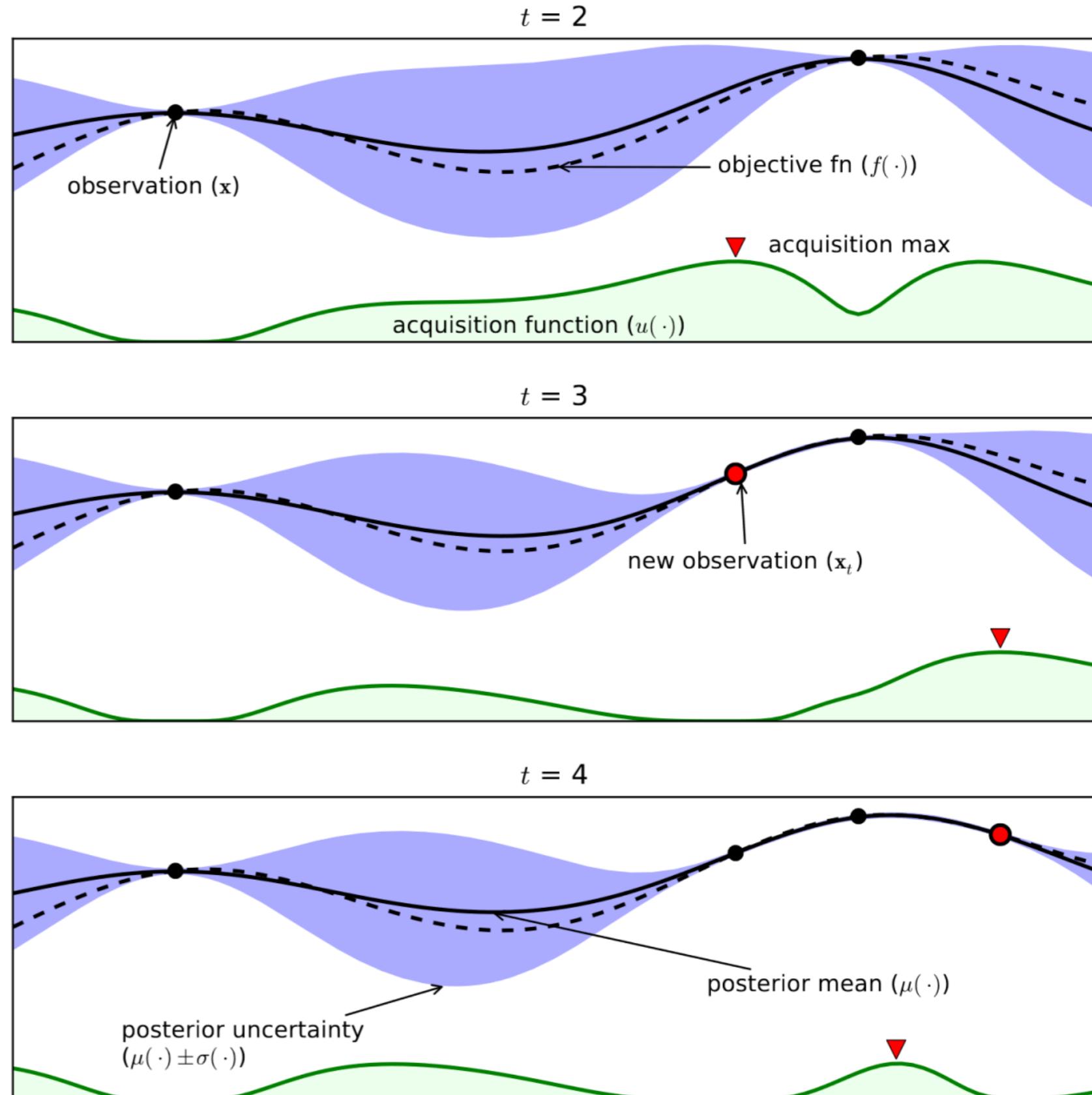
- Start with a few (random) hyperparameter configurations
- Fit a *surrogate model* to predict other configurations
- Probabilistic regression: mean  $\mu$  and standard deviation  $\sigma$  (blue band)
- Use an *acquisition function* to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample for the best configuration under that function

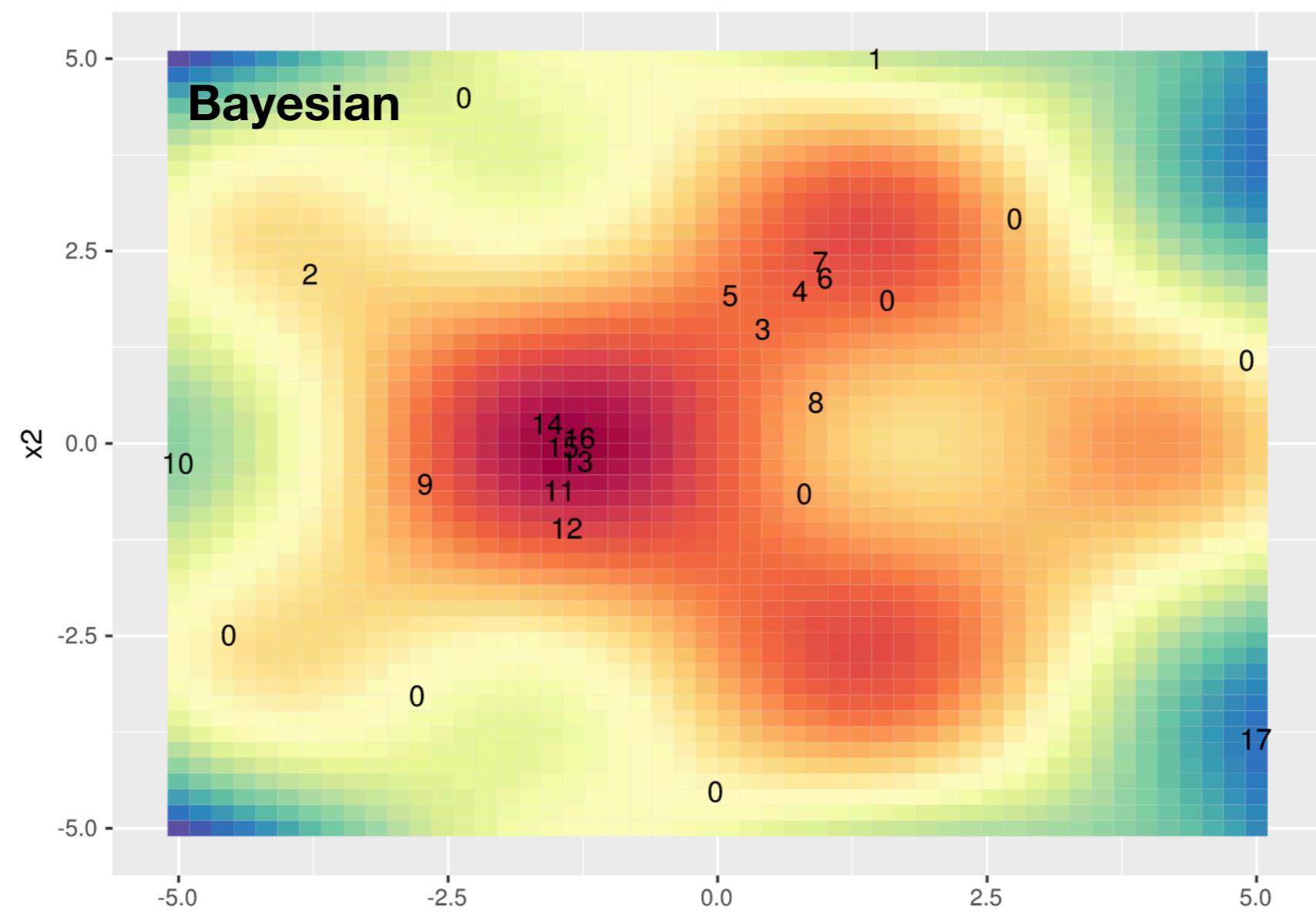
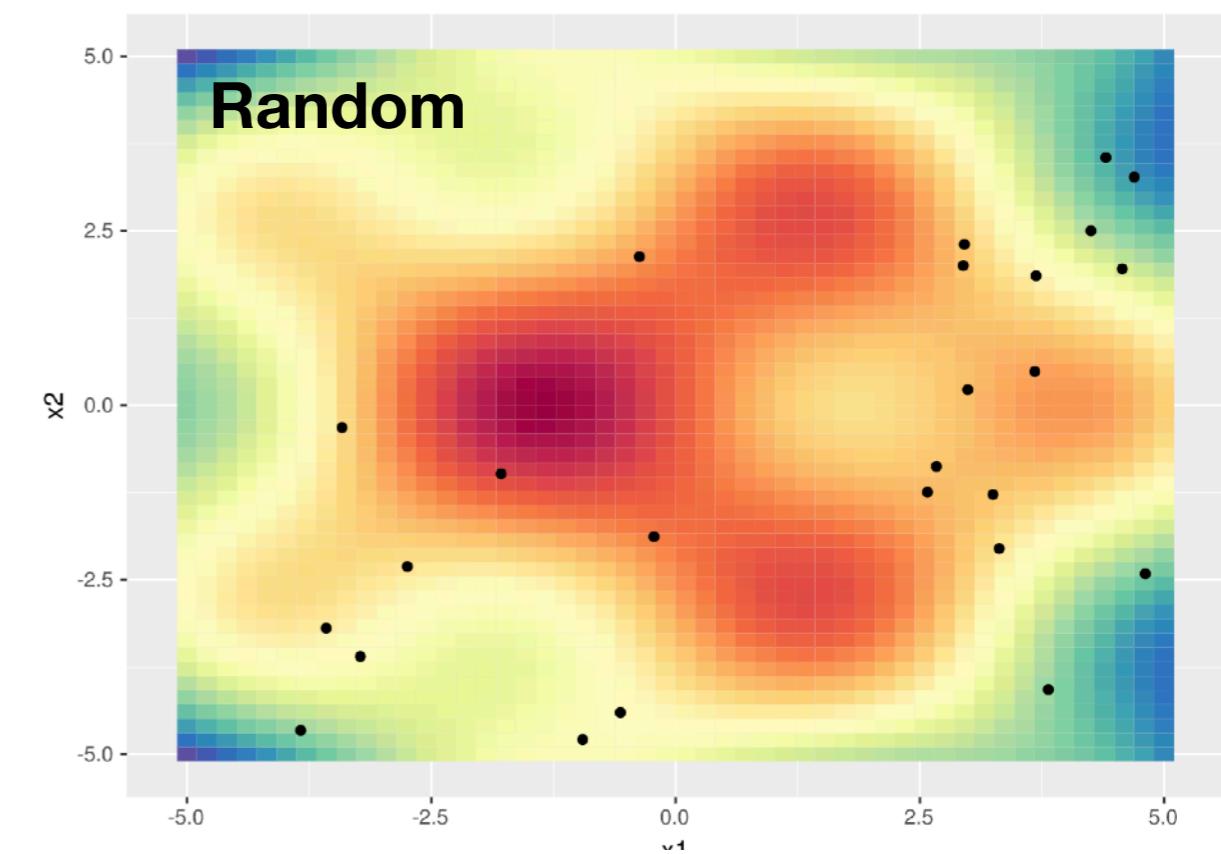
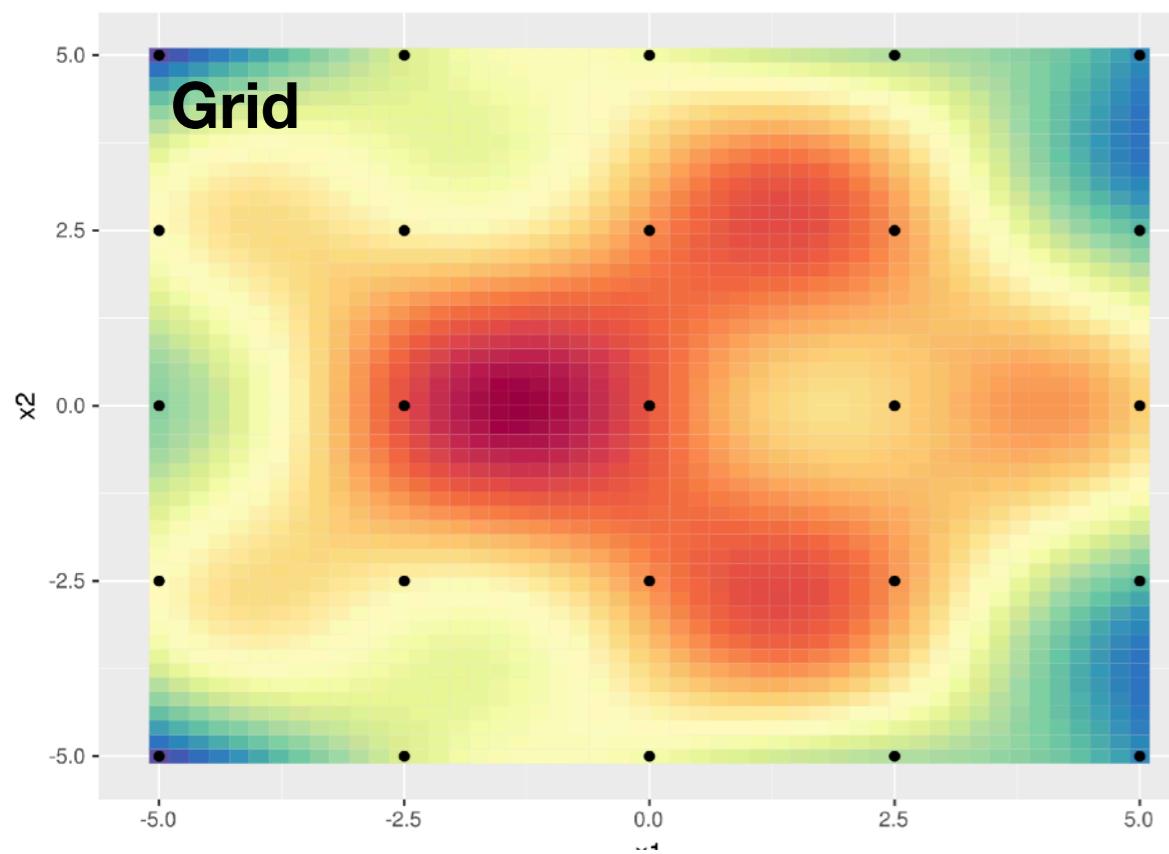


## Hyperparameter optimization

# Bayesian Optimization

- Repeat until some stopping criterion:
  - Fixed budget
  - Convergence
  - EI threshold
- Theoretical guarantees  
Srinivas et al. 2010, Freitas et al. 2012, Kawaguchi et al. 2016
- Also works for non-convex, noisy data
- Used in AlphaGo



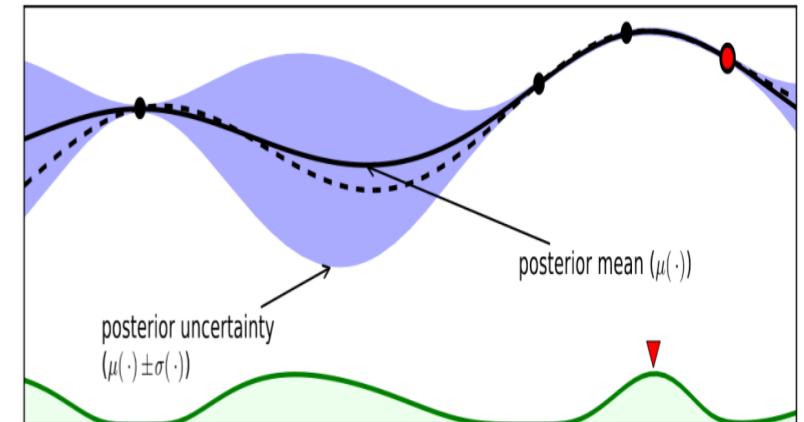


## Hyperparameter optimization

# Bayesian Optimization: which surrogate?

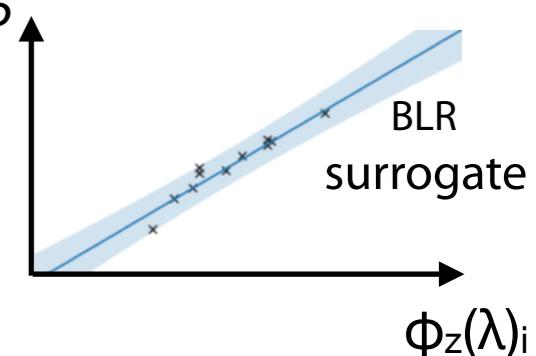
### Gaussian processes [\[skopt\]](#)

- + handles uncertainty, extrapolates well
- + ideal for few numeric hyperparameters
- scales badly (cubic), try random embeddings [\[Wang et al. 2013\]](#)



### Bayesian Linear Regression [\[Snoek et al. 2015\]](#) [\[Amazon AutoML\]](#)

- + Scalable, Bayesian, - requires good basis expansion



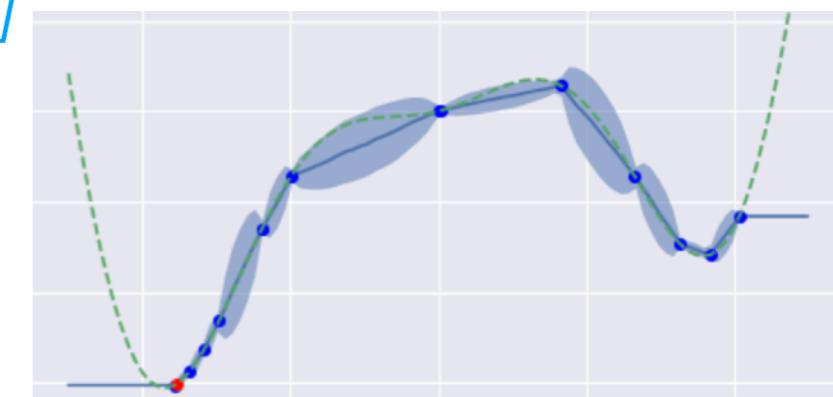
### Bayesian neural networks [\[Springenberg et al. 2016\]](#)

- + Bayesian - Scalability?

[\[autosklearn, autoWEKA\]](#)

### Random Forests [\[Hutter et al. 2011, Feurer et al. 2015\]](#)

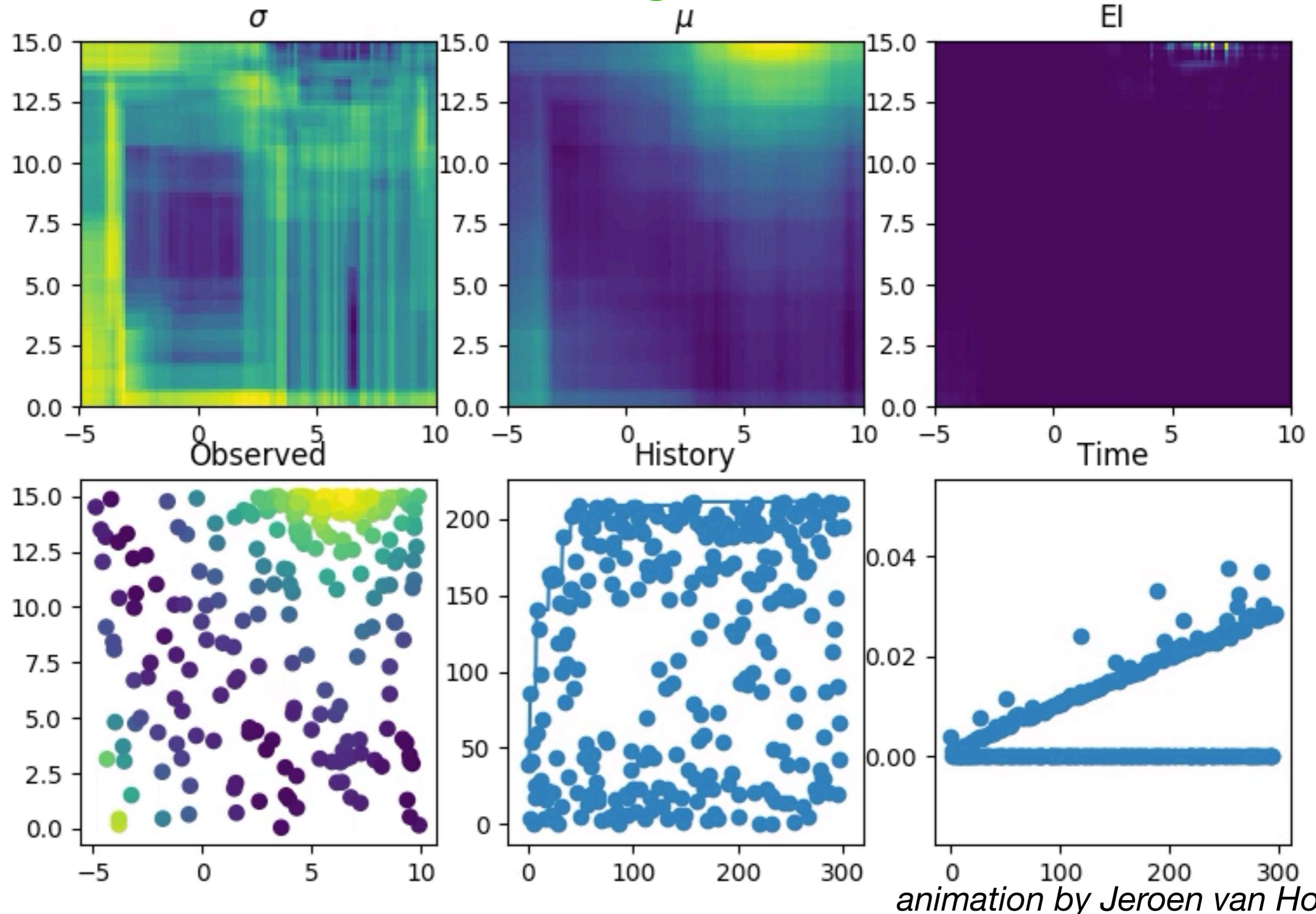
- + scalable, handles conditional hyperparams
- bad uncertainty estimates, extrapolation



### Boosting + quantile regression [\[van Hoof, Vanschoren 2019\]](#) [\[HyperBoost\]](#)

- + better fit than RF, handles conditionals, adapts to drift - new

# Random Forest Surrogate



## Hyperparameter optimization

# Tree of Parzen Estimators

1. Test some hyperparameters

2. Separate into **good** and **bad** hyperparameters (with some quantile)

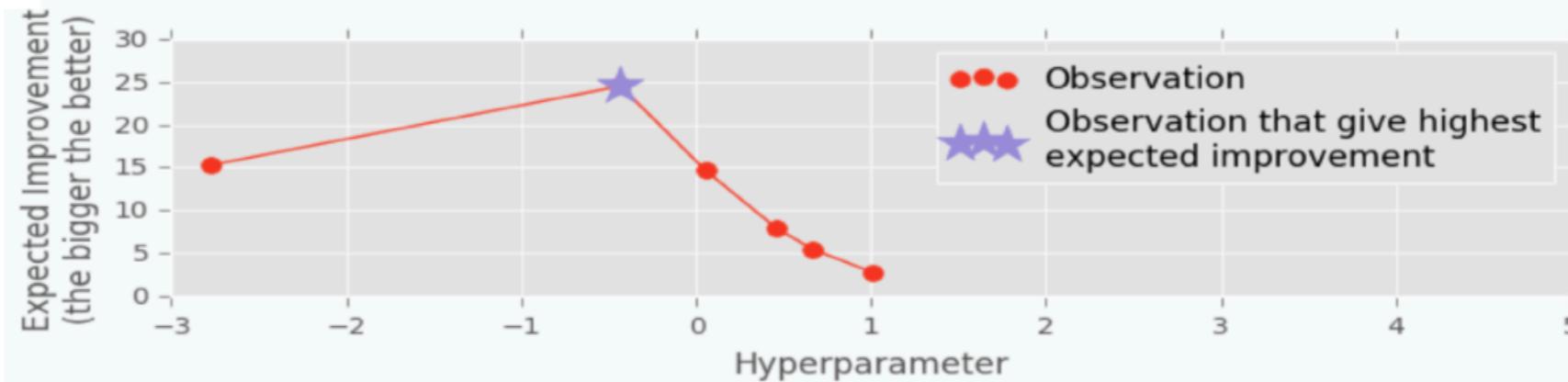
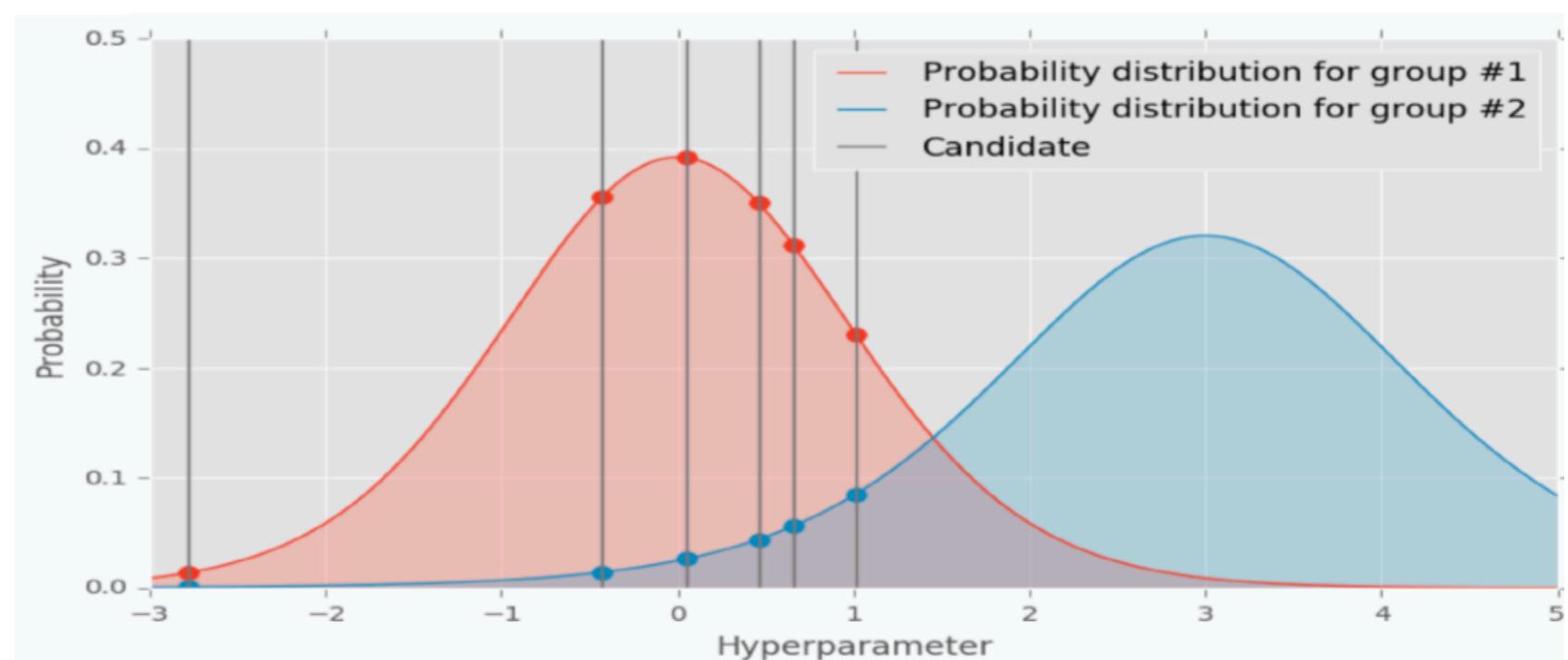
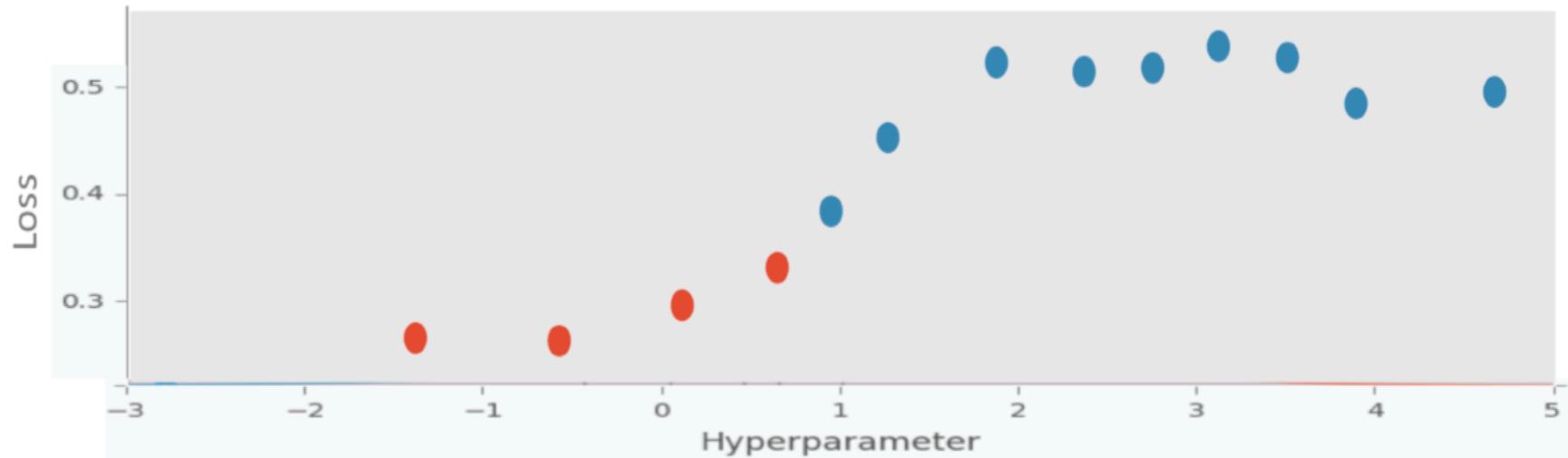
3. Fit non-parametric KDE for  $p(\lambda = \text{good})$  and  $p(\lambda = \text{bad})$

4. For a few samples, evaluate  $\frac{p(\lambda = \text{good})}{p(\lambda = \text{bad})}$

Shown to be equivalent to EI!

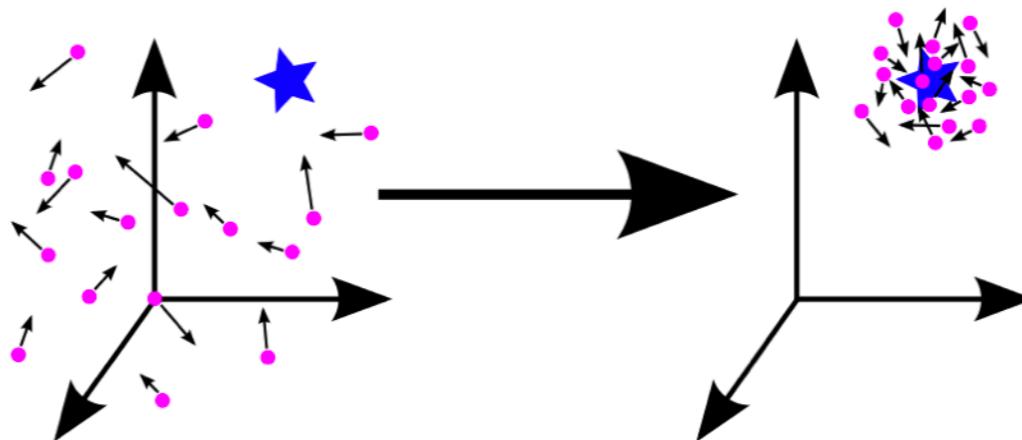
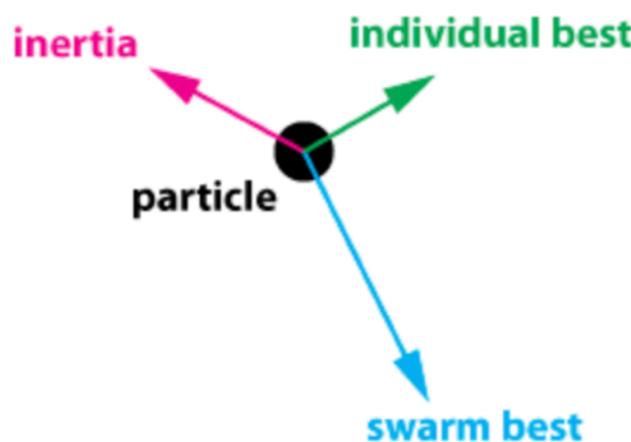
Efficient, **parallelizable**, robust, but less sample efficient than GPs

Used in HyperOpt-sklearn  
[Komera et al. 2019]



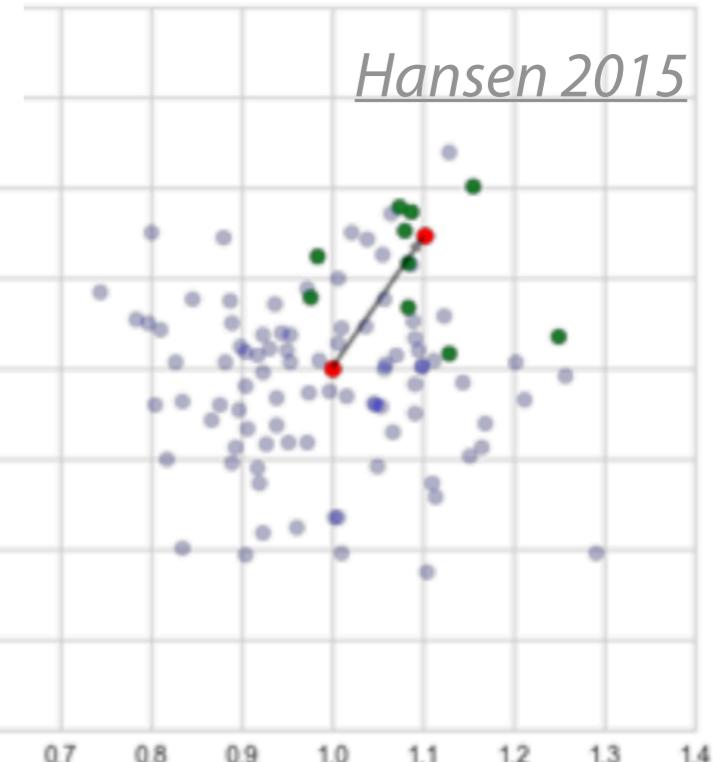
# Population-based methods

- Less sample efficient, but easy to parallelize, and adapts to changes
- Genetic programming *Olson, Moore 2016, 2019*
  - Mutations: add, mutate/tune, remove HP
- Particle swarm optimization *Mantovani et al 2015*



- Covariance matrix adaptation evolution (CMA-ES)
  - Purely continuous, expensive
  - Very competitive to optimize deep neural nets

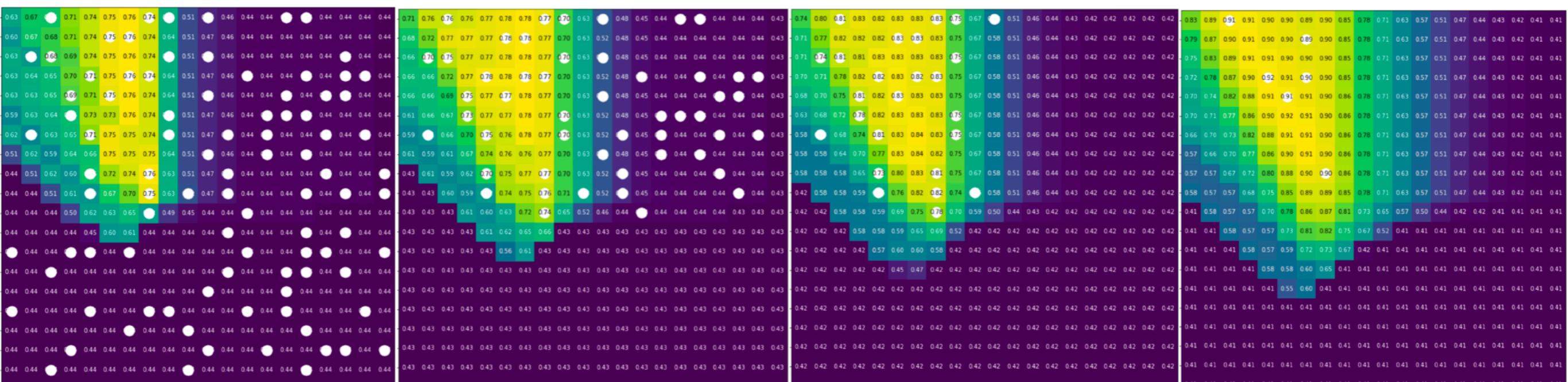
*[Loshilov, Hutter 2016]*



# Multi-fidelity optimization

Successive halving:

- train on small subsets, infer which regions may be interesting to evaluate in more depth
- Randomly sample candidates and evaluate on a small data sample
- retrain the 50% best candidates on twice the data



1/16

1/8

1/4

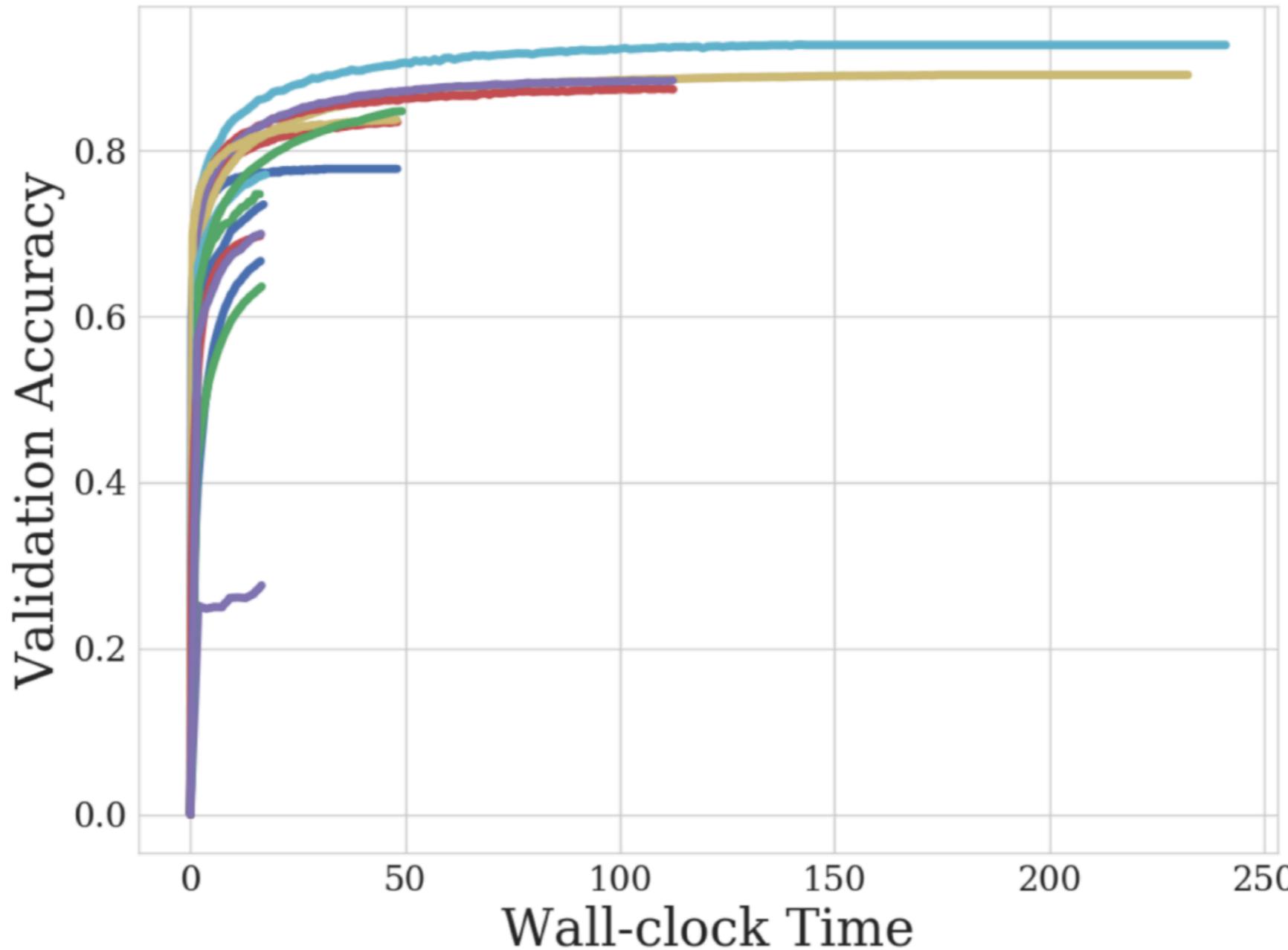
1/2

sample size

# Multi-fidelity optimization

Successive halving:

- Randomly sample candidates and evaluate on a small data sample
- retrain the best half candidates on twice the data

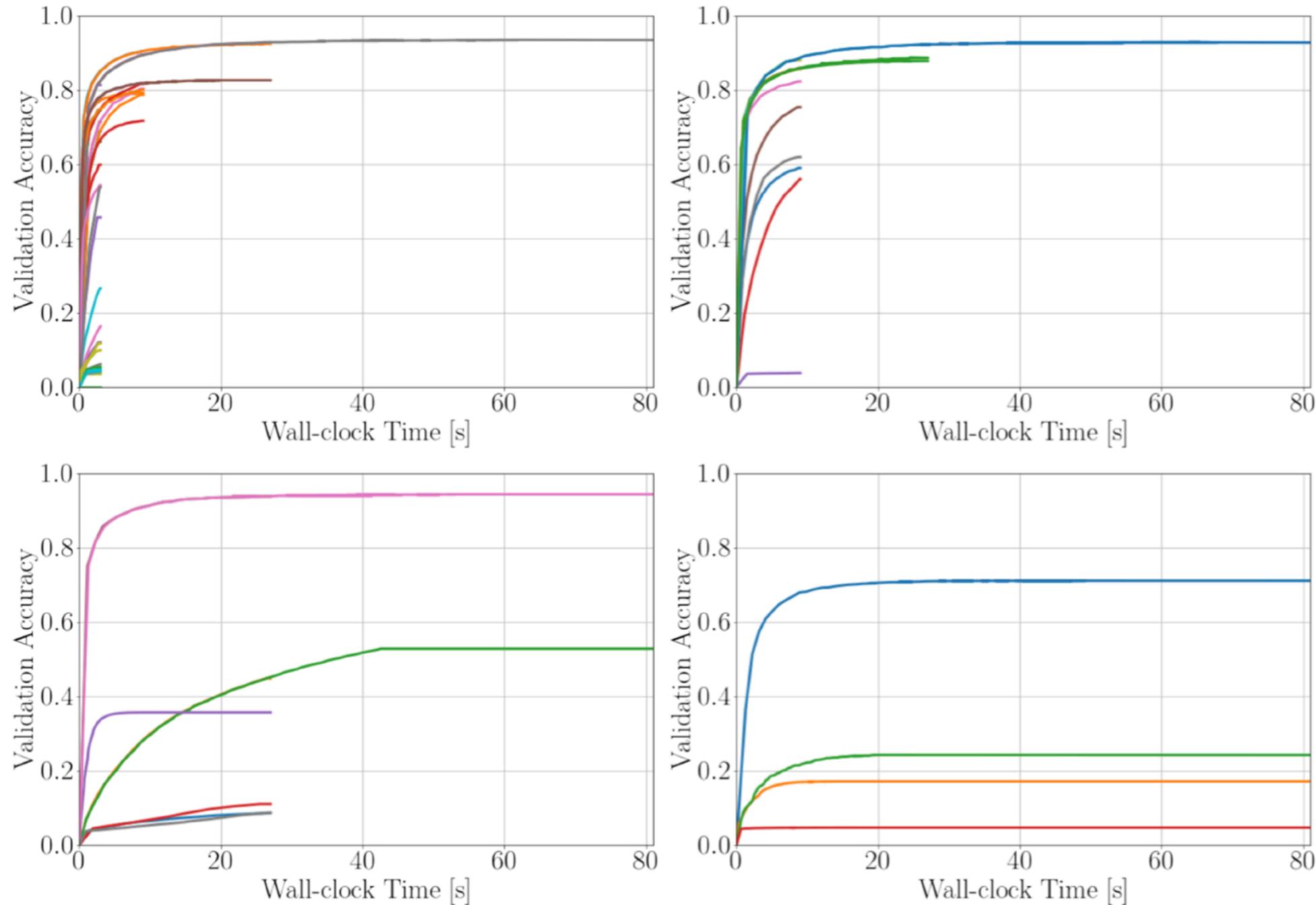


## Hyperparameter optimization

# Multi-fidelity optimization

**Hyperband (HB):** Repeated, decreasingly aggressive successive halving

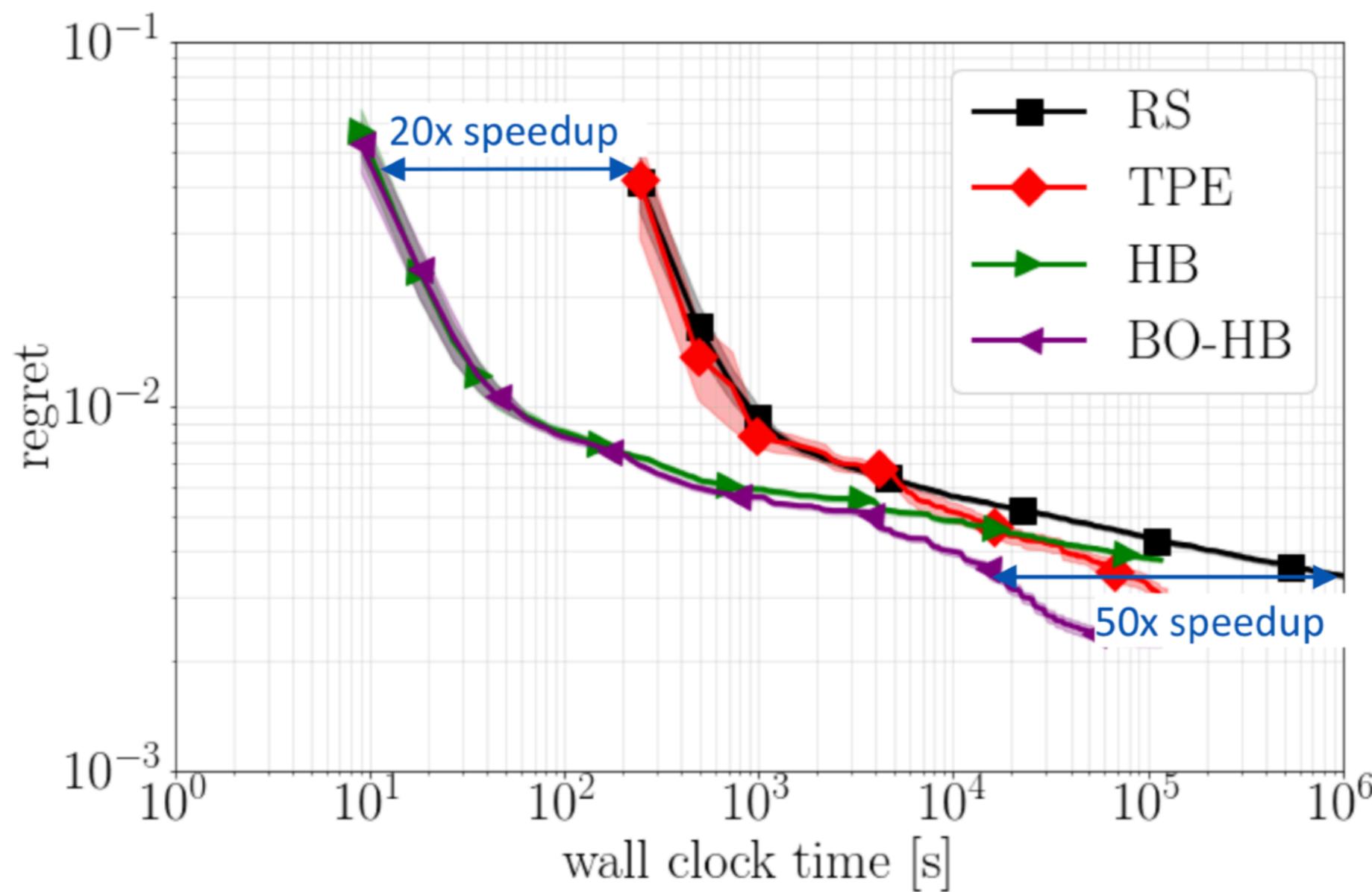
- Minimizes (doesn't eliminate) chance that candidate was pruned too early
- Strong anytime performance, easy to implement, scalable, parallelizable



# Multi-fidelity optimization

Combined Bayesian Optimization and Hyperband (BO-HB)

- Choose which configurations to evaluate (with TPE)
- Hyperband: allocate budgets more efficiently
- Strong anytime and final performance



# Hyperparameter gradient descent

Optimize neural network hyperparameters and weights simultaneously

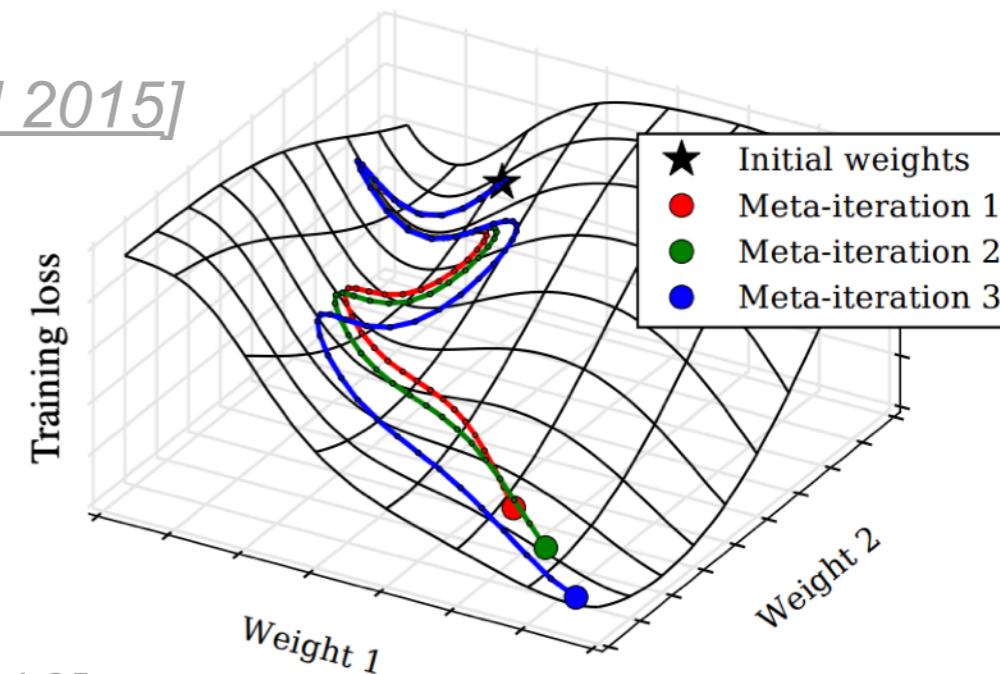
- Bilevel program *[Franceschi et al 2018]*

- Outer obj.: optimize  $\lambda$
- Inner obj.: optimize weights given  $\lambda$

$$\begin{aligned} & \min_{\lambda} \mathcal{L}_{val}(w^*(\lambda), \lambda) \\ \text{s.t. } & w^*(\lambda) = \operatorname{argmin}_w \mathcal{L}_{train}(w, \lambda) \end{aligned}$$

- Derive through the entire SGD *[MacLaurin et al 2015]*

- Get *hypergradients* wrt. validation loss
- Expensive! But useful if you have many hyper parameters and high parallelism

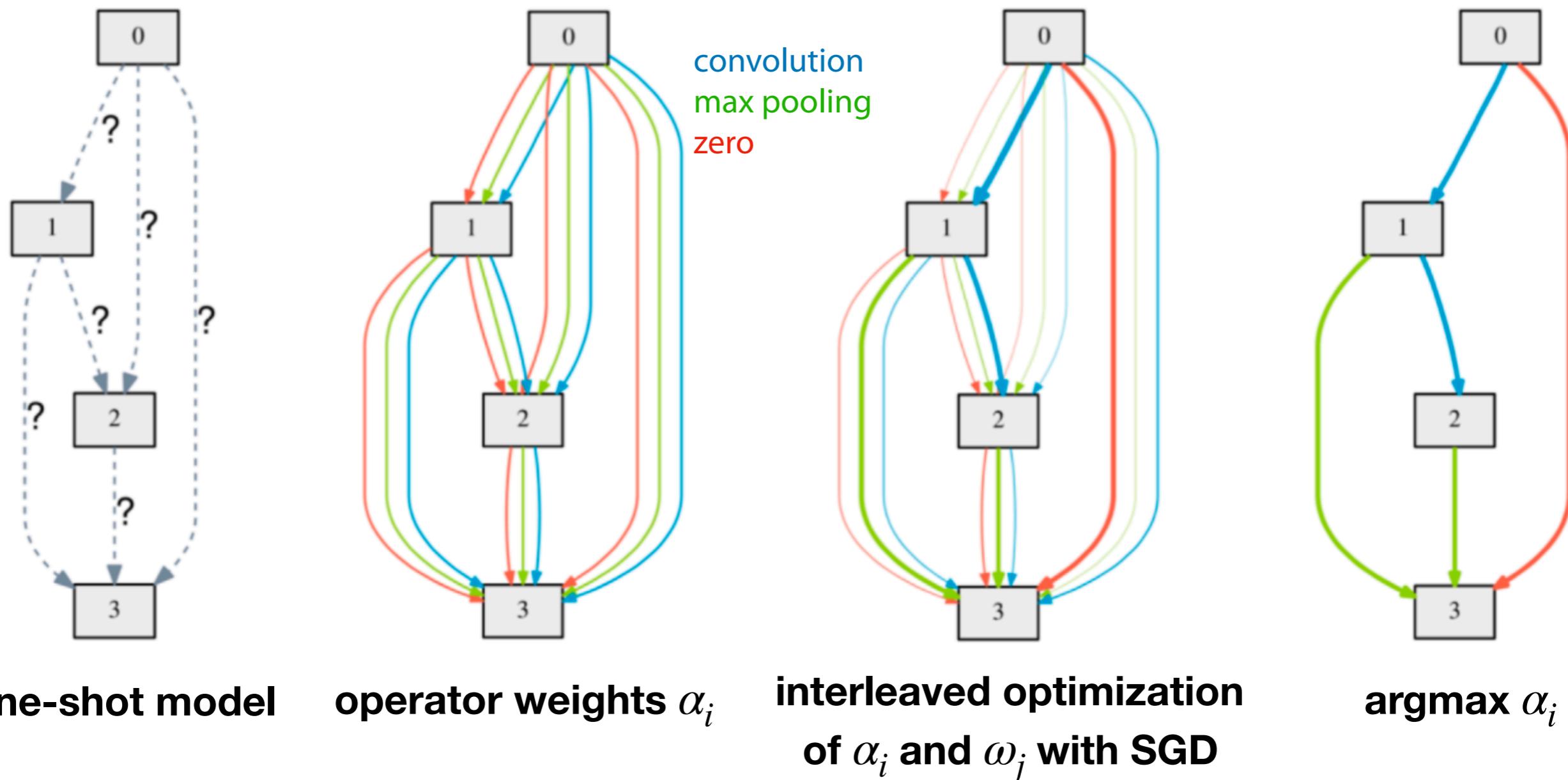


- Interleave optimization steps *[Luketina et al 2016]*
- Alternate SGD steps for  $\omega$  and  $\lambda$

Hyperparameter gradient step w.r.t.  $\nabla_{\lambda} \mathcal{L}_{val}$   
 Parameter gradient step w.r.t.  $\nabla_w \mathcal{L}_{train}$

# DARTS: Differentiable NAS

- Fixed (one-shot) structure, learn which operators to use
- Give all operators a weight  $\alpha_i$
- Optimize  $\alpha_i$  and model weights  $\omega_j$  using bilevel programming



# Learning is a never-ending process

Learning humans also seek/create related tasks

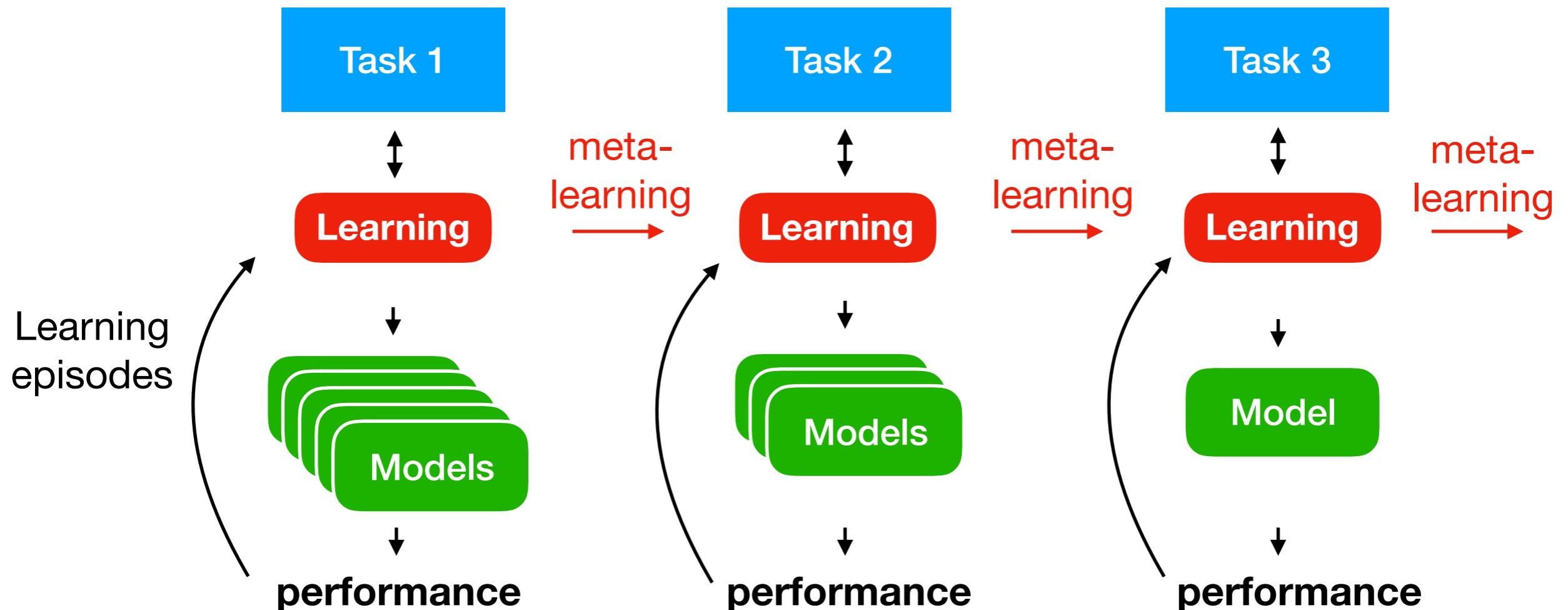


E.g. find many similar puzzles, solve them in different ways,...

# Learning is a never-ending process

Humans learn *across* tasks

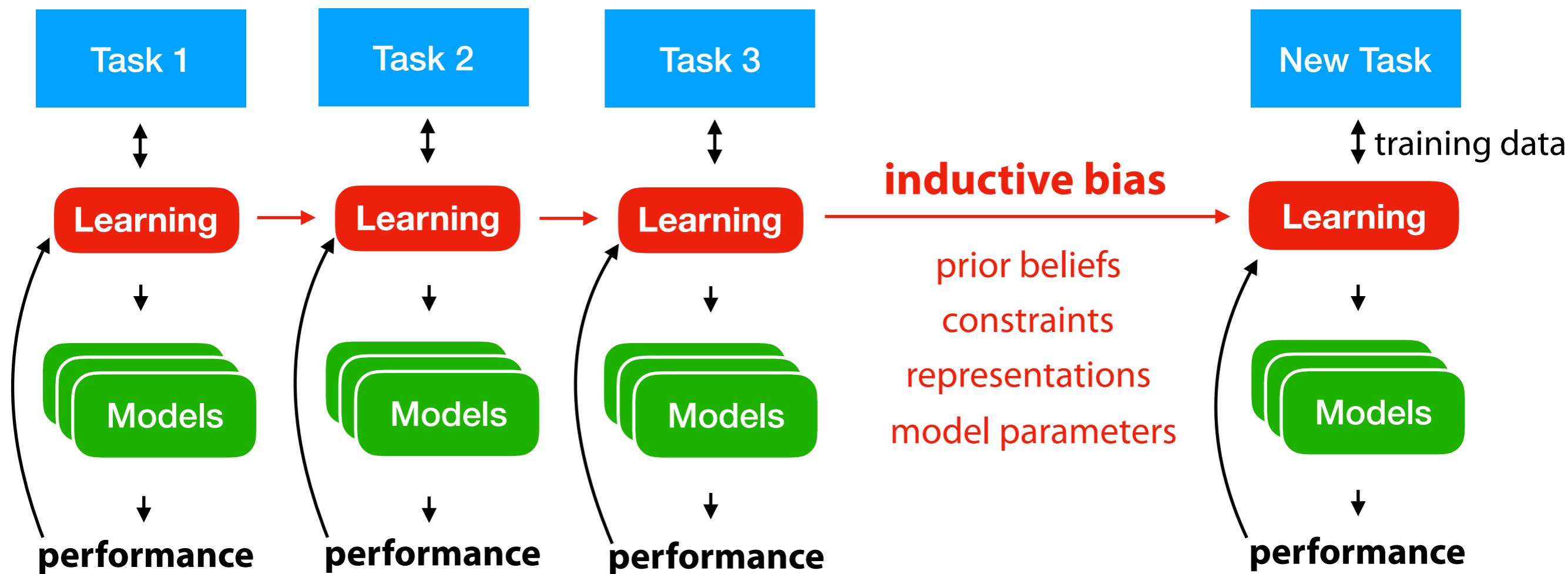
Why? Requires less trial-and-error, less data



# Learning to learn

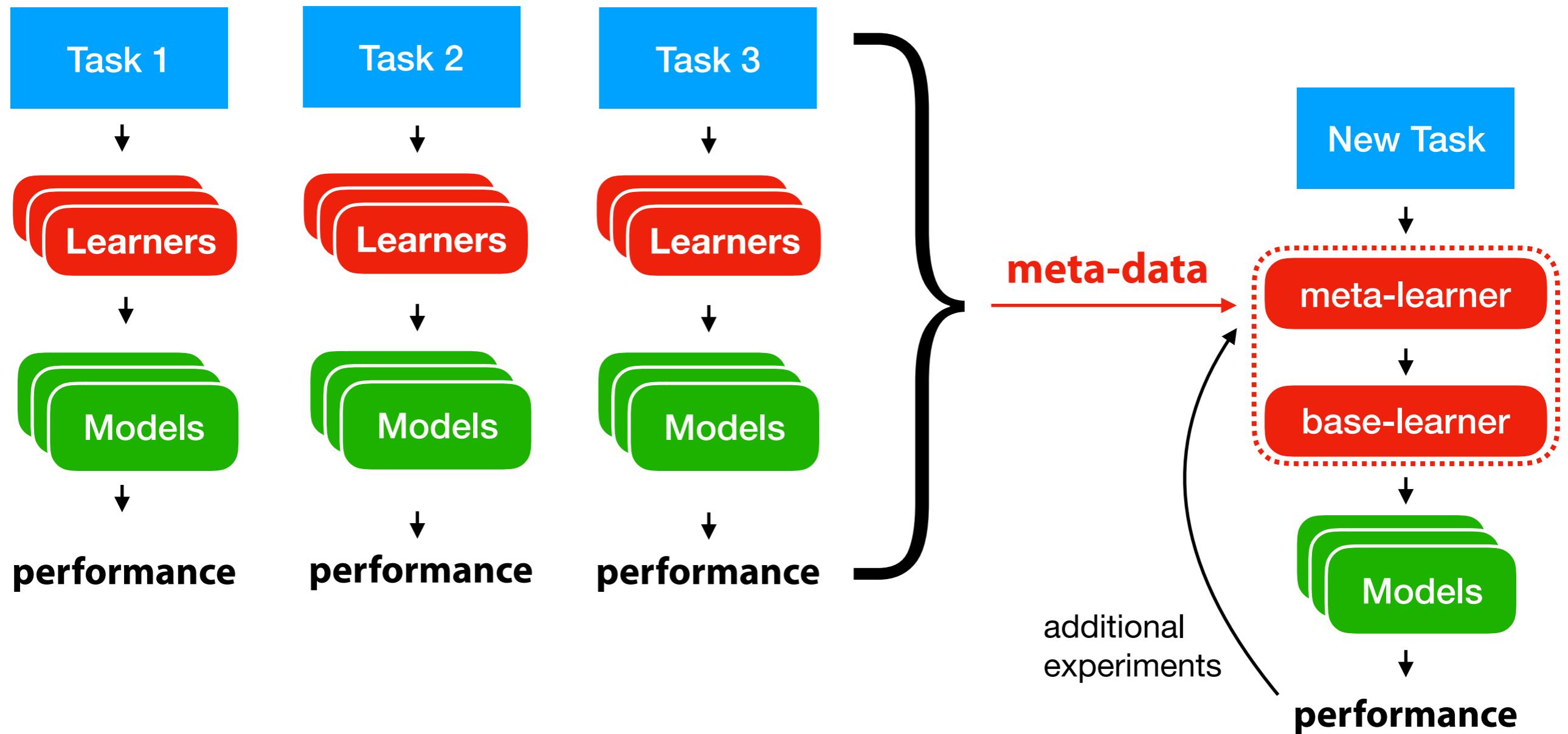
If prior tasks are *similar*, we can **transfer** prior knowledge to new tasks

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



# Meta-learning

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



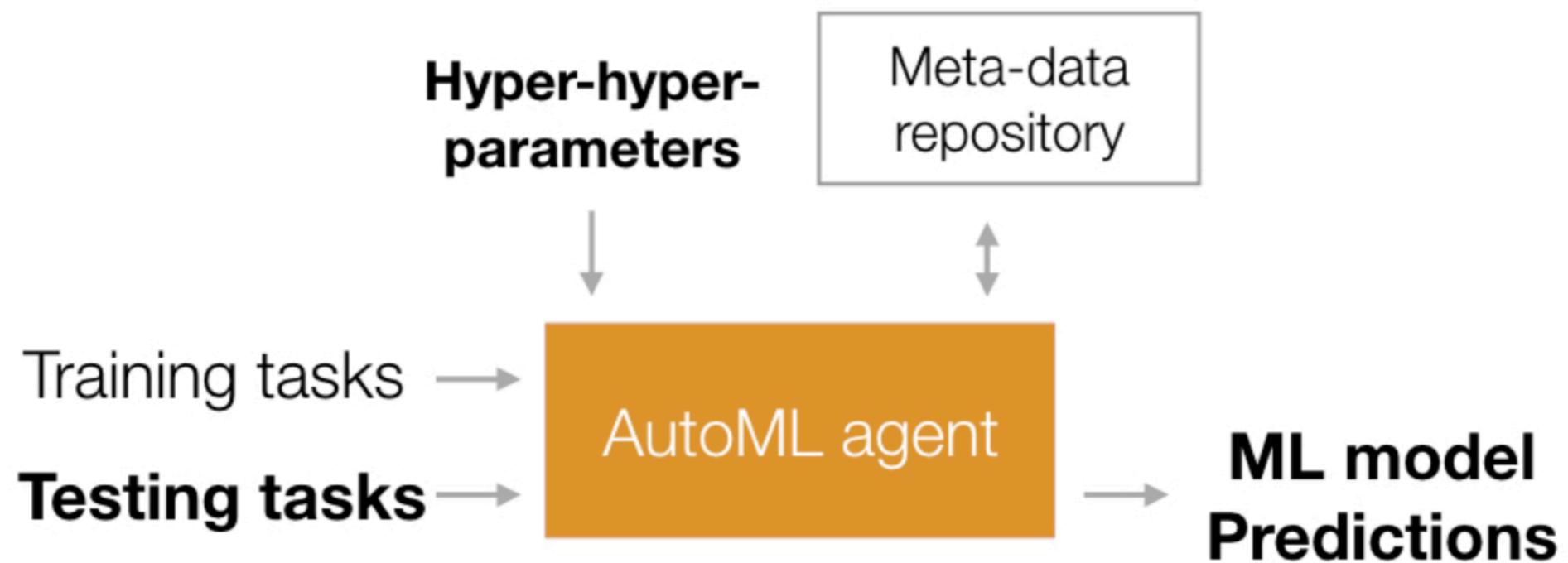
# Meta-learning

Meta-learning can drastically speed up the architecture and hyperparameter search, in combination with the optimization techniques we just discussed

- Learn which hyperparameters are really important
- Learn which hyperparameter values should be tried first
- Learn which architectures will most likely work
- Learn how to clean and annotate data
- Learn which feature representations to use
- Learn which embeddings, pre-trained nets to use
- ...

# Meta-learning in practice

- We need a meta-data repository of relevant prior machine learning experiments
- Ideally, a *shared* memory that all AutoML tools can access



# Meta-learning in practice

- OpenML.org
  - Thousands of uniform datasets, 100+ meta-features
  - Millions of evaluated and *reproducible* runs
    - Same splits, 30+ metrics
    - Traces, models (*opt*)
- APIs in Python, R, Java,... Sklearn, (Keras, PyTorch, MXNet,...)
- Publish your own runs
- Never ending learning
- Benchmarks

```
import openml as om1
from sklearn import ensembles

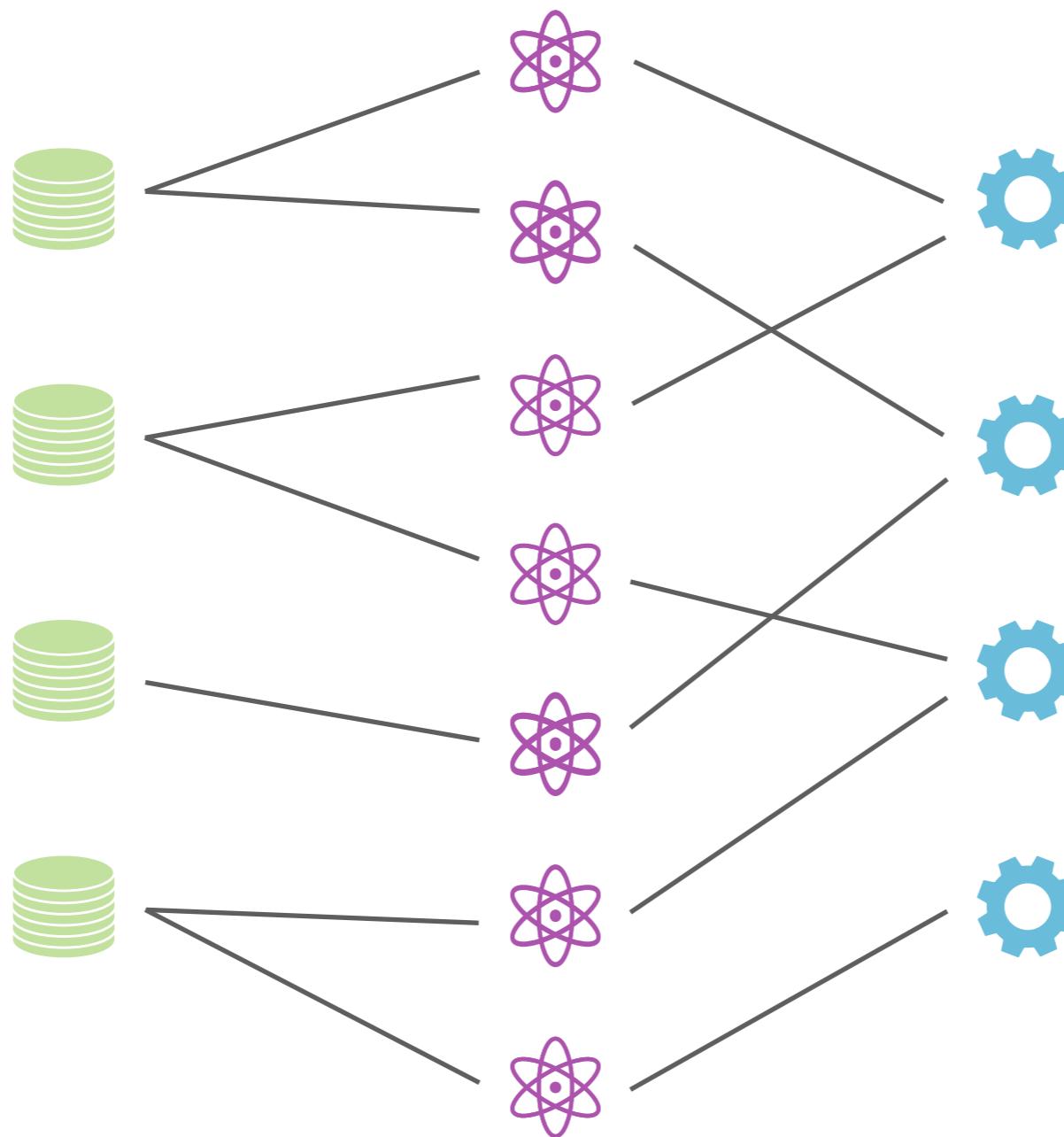
task = om1.tasks.get_task(14951)
clf = ensembles.RandomForestClassifier()
run = om1.runs.run_model_on_task(task, flow)
myrun = run.publish()
```

# OpenML: Organizing ML meta-data

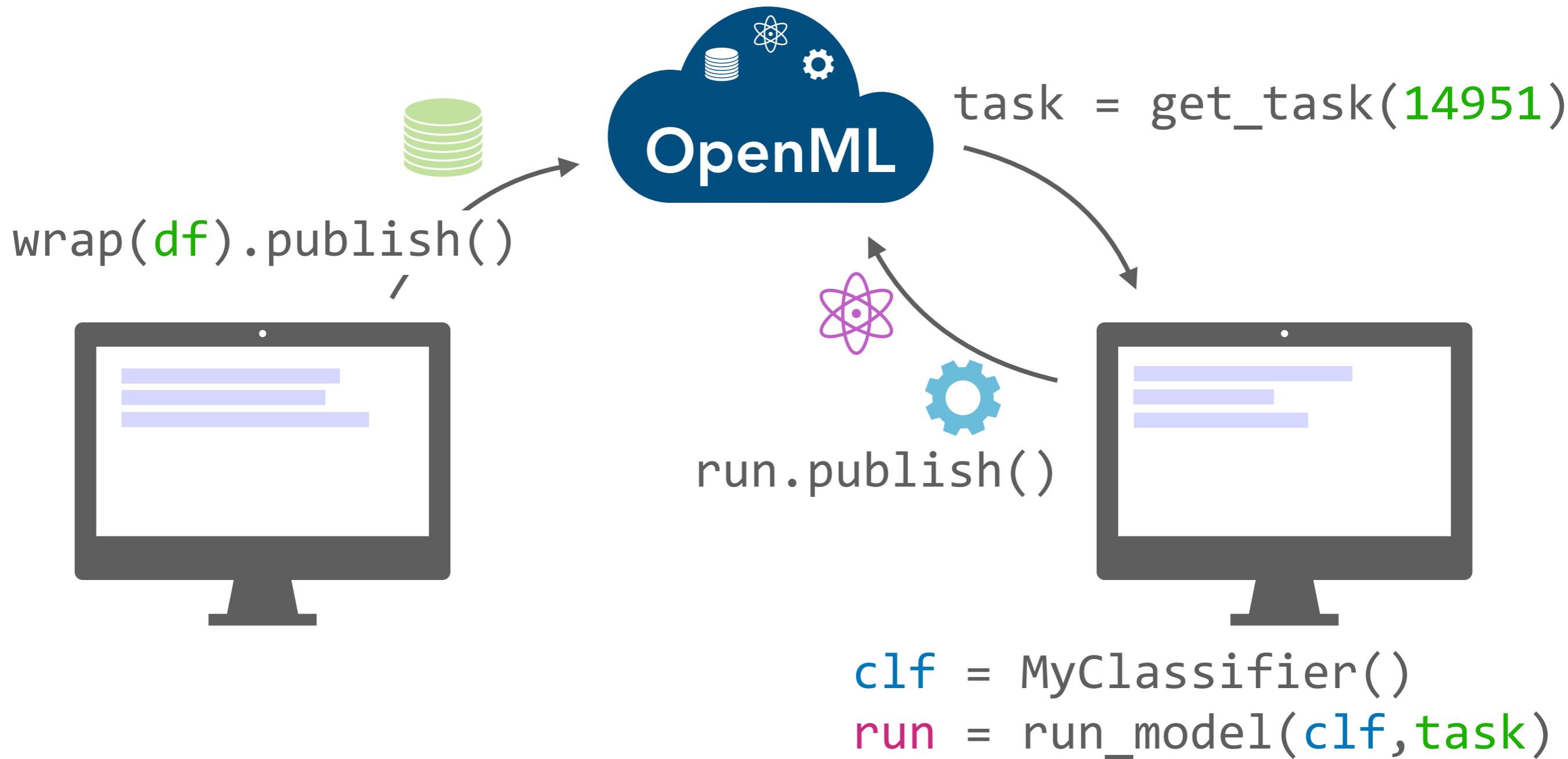
For every **dataset**, find all models built (and how good they are)

For every **model**, get the *exact* dataset and algorithm used

For every **algorithm**, find how useful it is for every dataset



# OpenML: Frictionless machine learning



APIs:

Integrations:

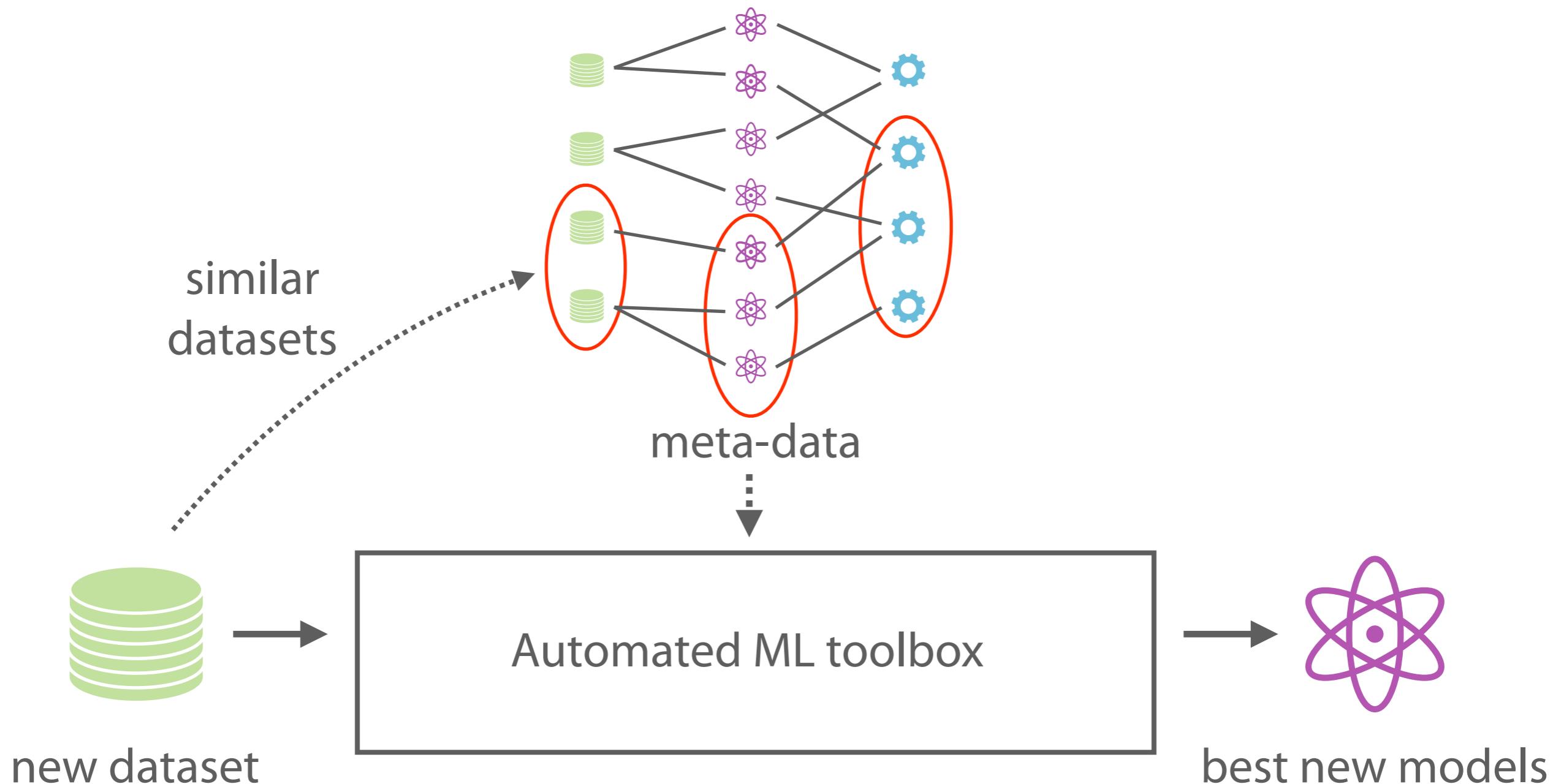
# OpenML.org

The screenshot shows the OpenML web interface. On the left, a dark sidebar lists various navigation options: Discover, Data sets, Tasks (90.2k), Flows, Runs, Collections, Task Types, Measures, People, Learn more, Documentation, Get involved, OpenML Foundation, Terms & Citation, and Our team. At the bottom of the sidebar are five colored stars. The main content area has a light background. At the top, there's a search bar with placeholder text "Search datasets...". Below it, a header bar shows "90172 tasks found" with filter icons (X, 🔍, ⚡) and tabs for "Detail" and "Dashboard". The "Detail" tab is active, displaying a list of tasks:

- eeg-eye-state: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 96.2k datasets, 0 likes, 0 clouds, posted 2 years ago.
- hill-valley: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 95.5k datasets, 0 likes, 0 clouds, posted 4 years ago.
- hill-valley: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 92.4k datasets, 0 likes, 0 clouds, posted 2 years ago.
- ozone-level-8hr: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 90.8k datasets, 0 likes, 0 clouds, posted 4 years ago.
- ozone-level-8hr: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 90.7k datasets, 0 likes, 1 cloud, posted 2 years ago.
- banknote-authentication: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 89.0k datasets, 0 likes, 0 clouds, posted 4 years ago.
- hill-valley: Supervised Classification: predict 'Class', evaluate with 10-fold Crossvalidation. 88.3k datasets, 0 likes, 0 clouds, posted 4 years ago.
- scene

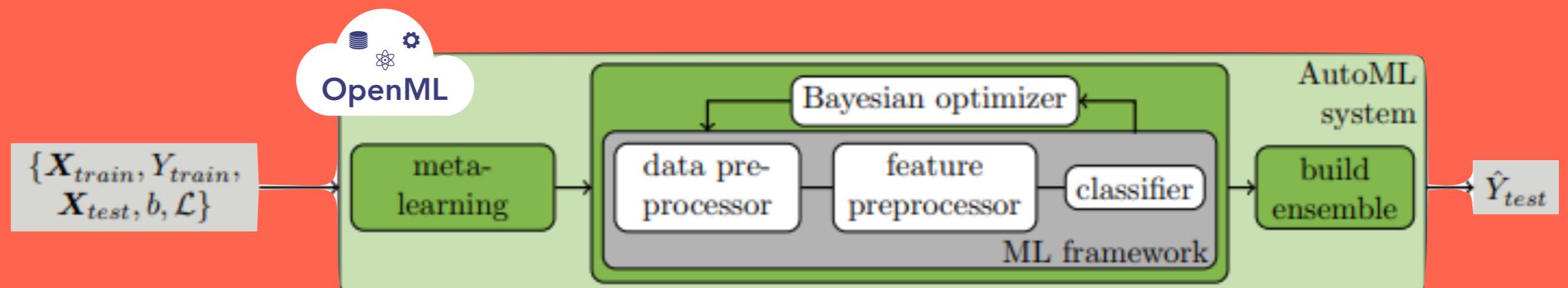
The "Dashboard" tab is active, showing a scatter plot titled "Top 1000 runs shown". The x-axis represents accuracy values ranging from 0.985 to 0.995. The y-axis lists various machine learning models and pipelines. Each model is represented by a series of colored diamonds (blue, red, green) indicating different runs or configurations. The models listed include:  
sklearn.pipeline.Pipeline(imputation=openmlstudy14..  
sklearn.svm.classes.SVC(5)  
weka.SMO\_RBFKernel(4)  
sklearn.model\_selection.\_search.RandomizedSearchCV..  
sklearn.pipeline.Pipeline(imputation=openmlstudy14..  
weka.KStar(4)  
sklearn.neighbors.classification.KNeighborsClassif..  
sklearn.neighbors.classification.KNeighborsClassif..  
mlr.classif.xgboost(6)  
mlr.classif.xgboost(4)  
sklearn.pipeline.Pipeline(pca=sklearn.decompositio..  
mlr.classif.ranger(15)  
sklearn.pipeline.Pipeline(imputer=sklearn.preproce..  
weka.kf.AttributeSelection-Ranker-ReliefF-KStar(1)  
mlr.classif.ranger(16)  
sklearn.pipeline.Pipeline(imputation=openmlstudy14..  
sklearn.pipeline.Pipeline(columntransformer=sklear..  
mlr.classif.ranger(9)  
sklearn.svm.classes.SVC(32)  
weka.FilteredClassifier\_MultiSearch\_RandomForest(1..  
mlr.classif.ranger(13)  
weka.RandomForest(9)  
weka.kf.RandomForest(1)  
mlr.classif.ranger(10)  
weka.Bagging\_RandomForest(9)  
mlr.classif.ranger(8)  
mlr.classif.ranger.imputed.dummied.preproc(1)  
weka.Bagging\_IBk(3)  
weka.kf.Bagging-IBk5(1)

# Meta-learning with OpenML



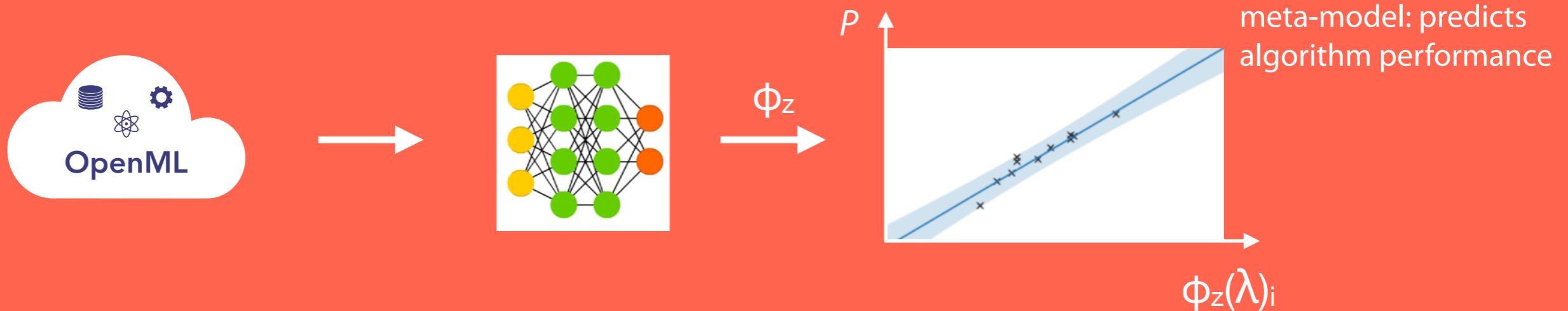
# AutoML + metalearning

auto-sklearn: uses OpenML to *warm-start* the search for the best pipelines



Feurer et al. 2016

ABLR (Amazon): uses OpenML to learn how to search hyperparameters



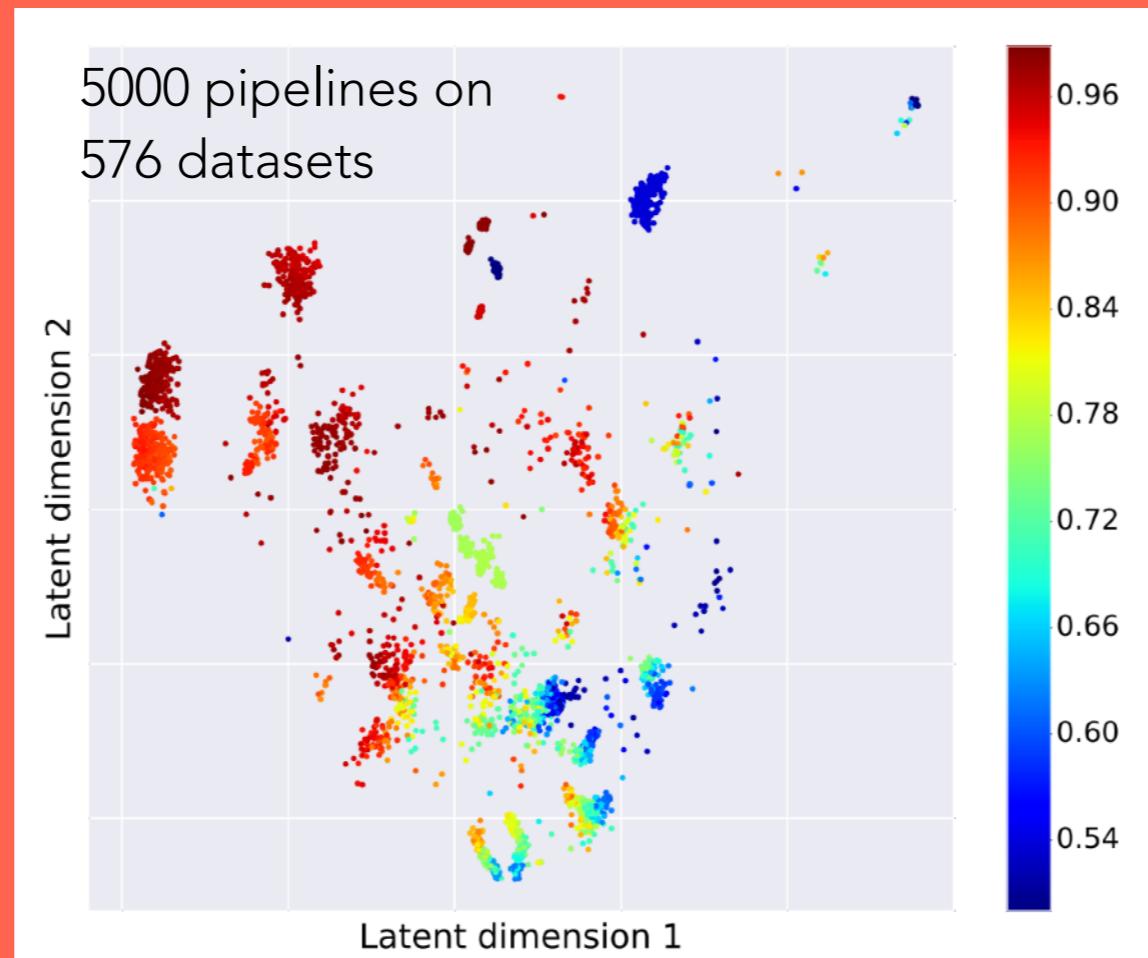
Perrone et al. 2017

# AutoML + metalearning

ProbMF (Microsoft): uses OpenML data to train pipeline recommender



datasets →



**Recommender  
system**

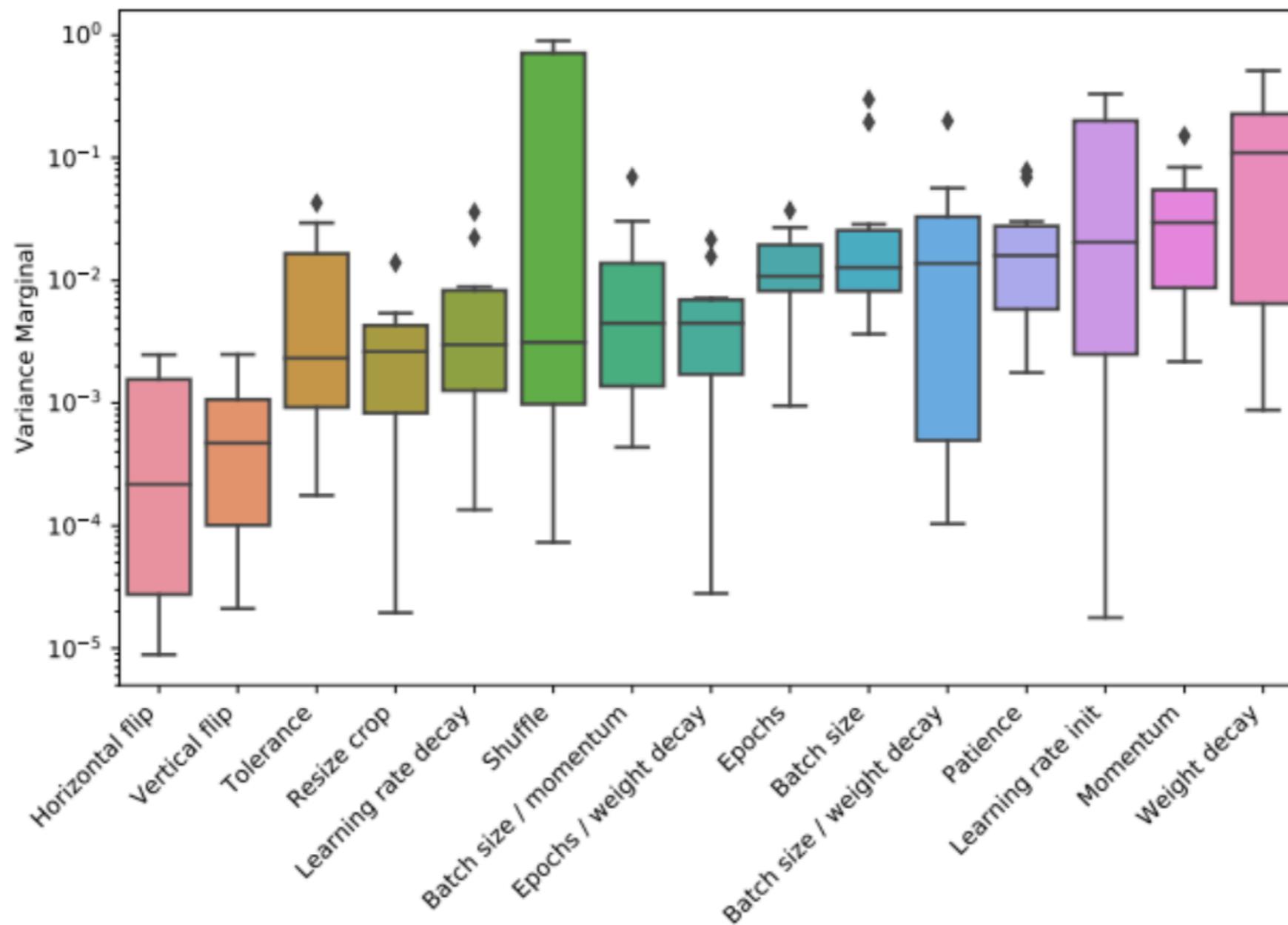
Fusi et al. 2018

## Meta-Learning

# Meta-learning for Hyperparameter optimization

- **Functional ANOVA** <sup>1</sup>

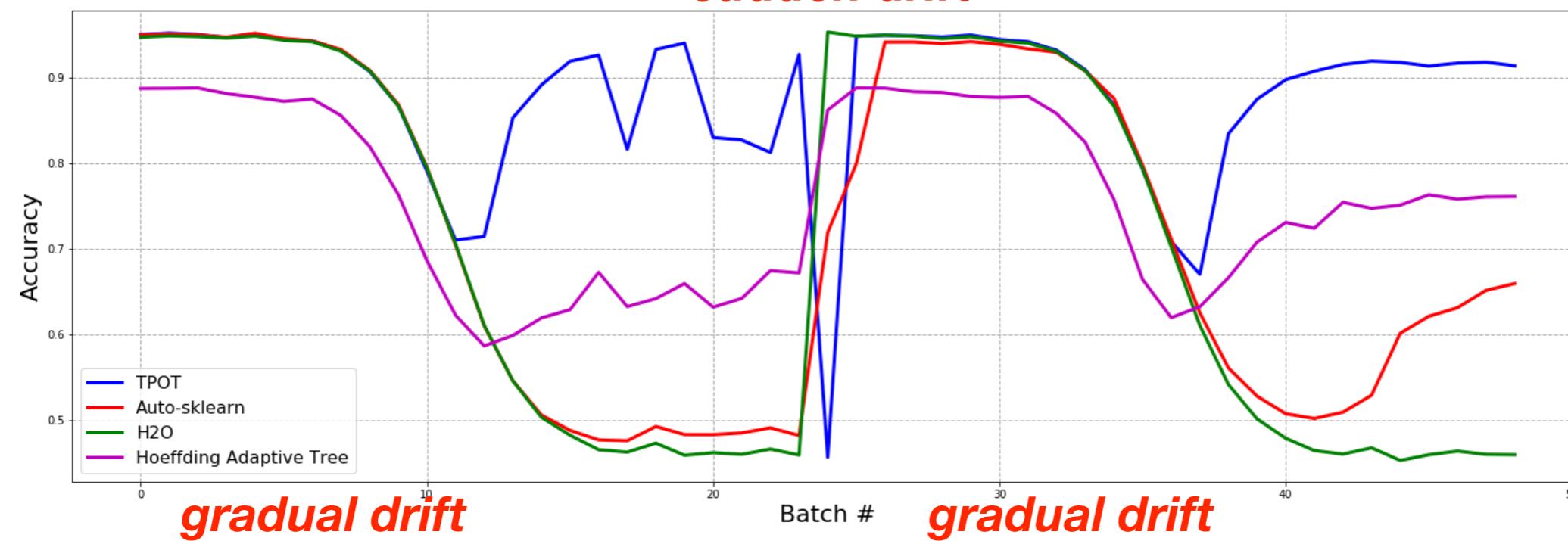
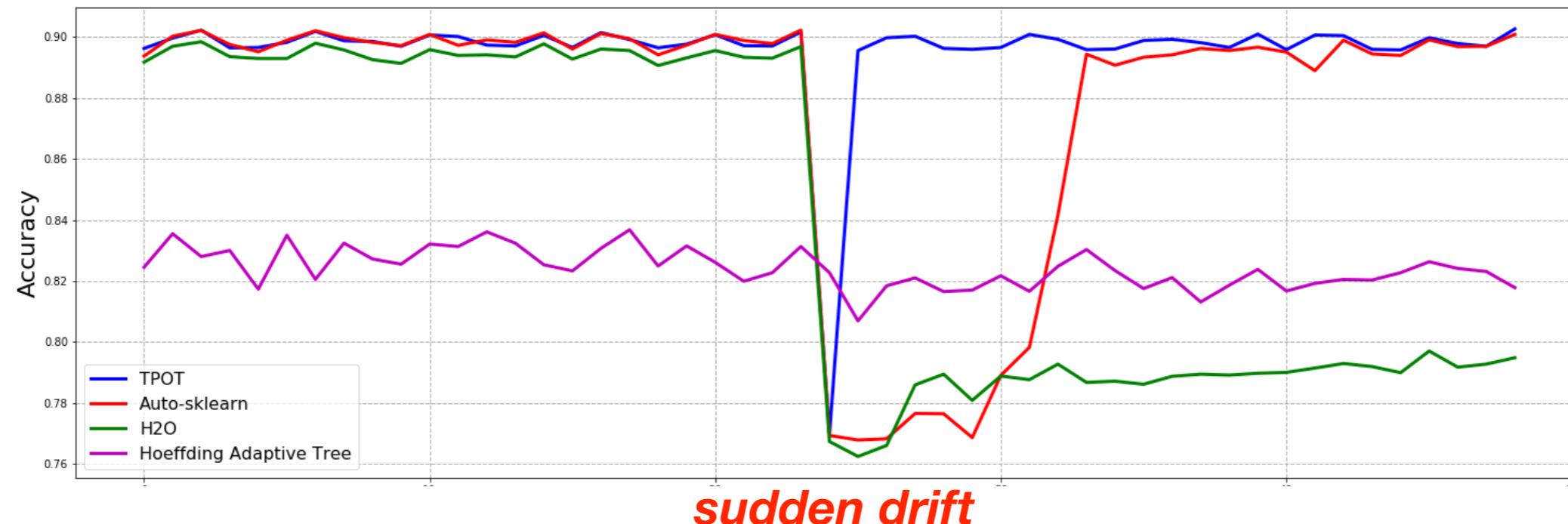
Select hyperparameters that cause variance in the evaluations.



## Meta-Learning

# Meta-learning for AutoML on streams

- Different AutoML methods adapt differently to (synthetic) concept drift
- All need specific adaptations to work well on streaming data

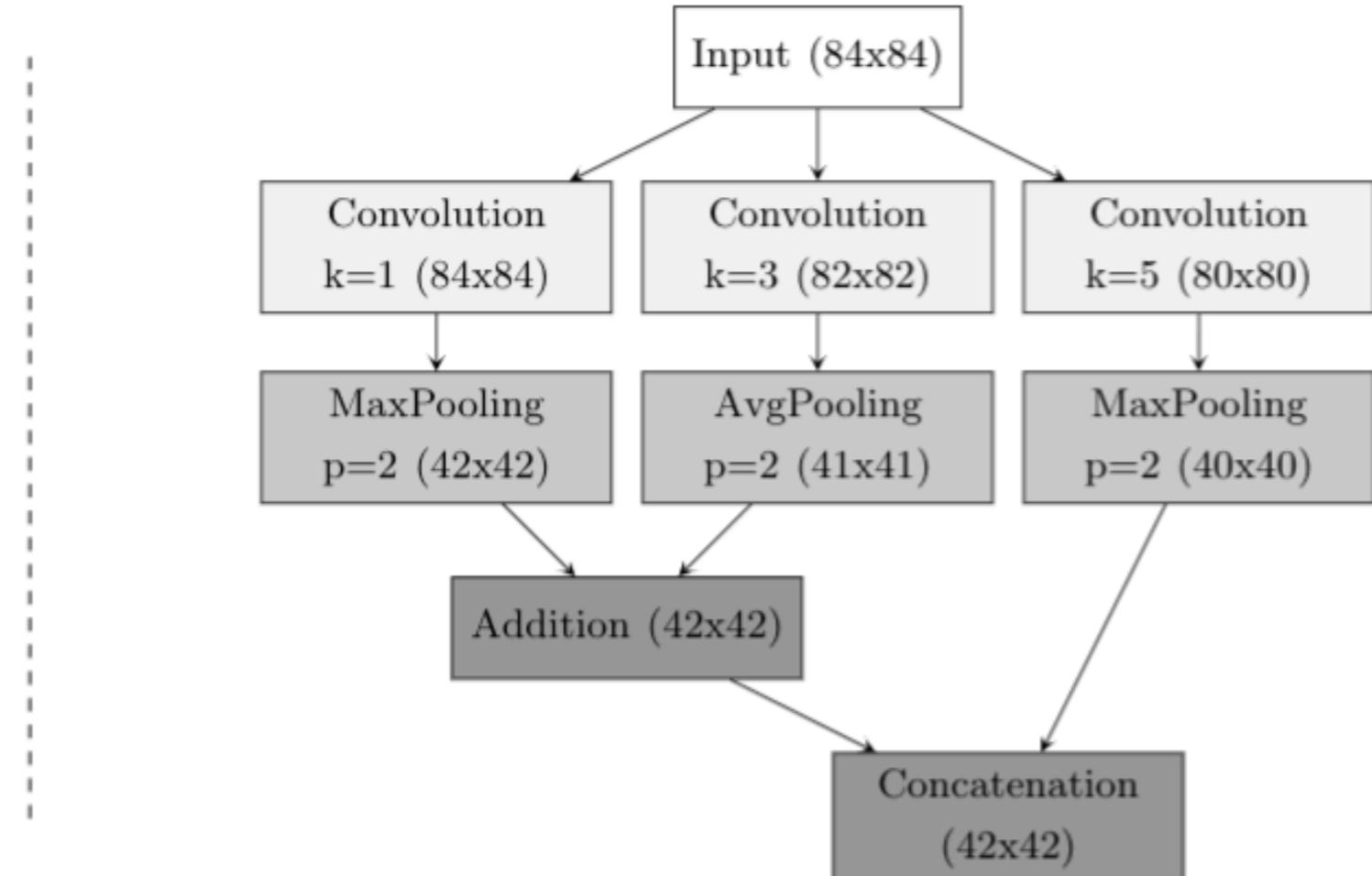


## Meta-Learning

# Meta-Reinforcement Learning for NAS

- Train an agent how to build a neural net, across tasks
- Should transfer but also adapt to new tasks

[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]  
[1, 1, 1, 0, 0]  
[2, 2, 2, 1, 0]  
[3, 1, 3, 0, 0]  
[4, 3, 2, 3, 0]  
[5, 1, 5, 0, 0]  
[6, 2, 2, 5, 0]  
[7, 5, 0, 2, 4]  
[8, 7, 0, 0, 0]



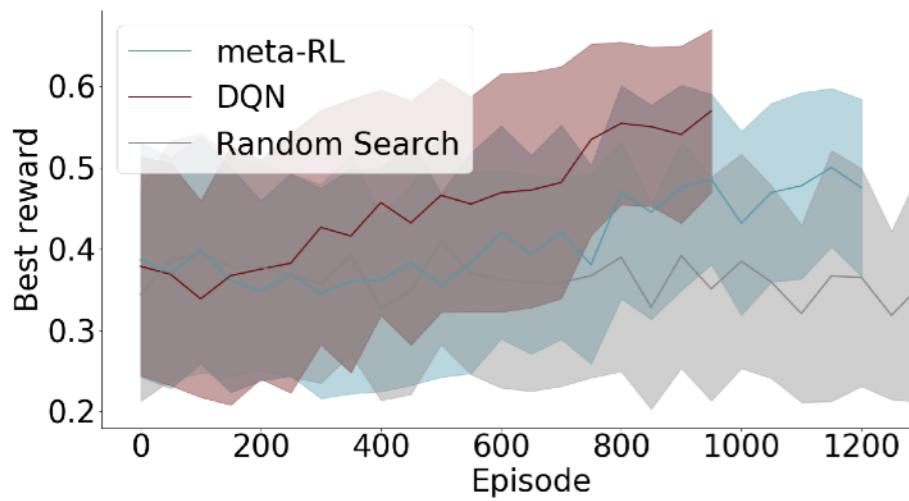
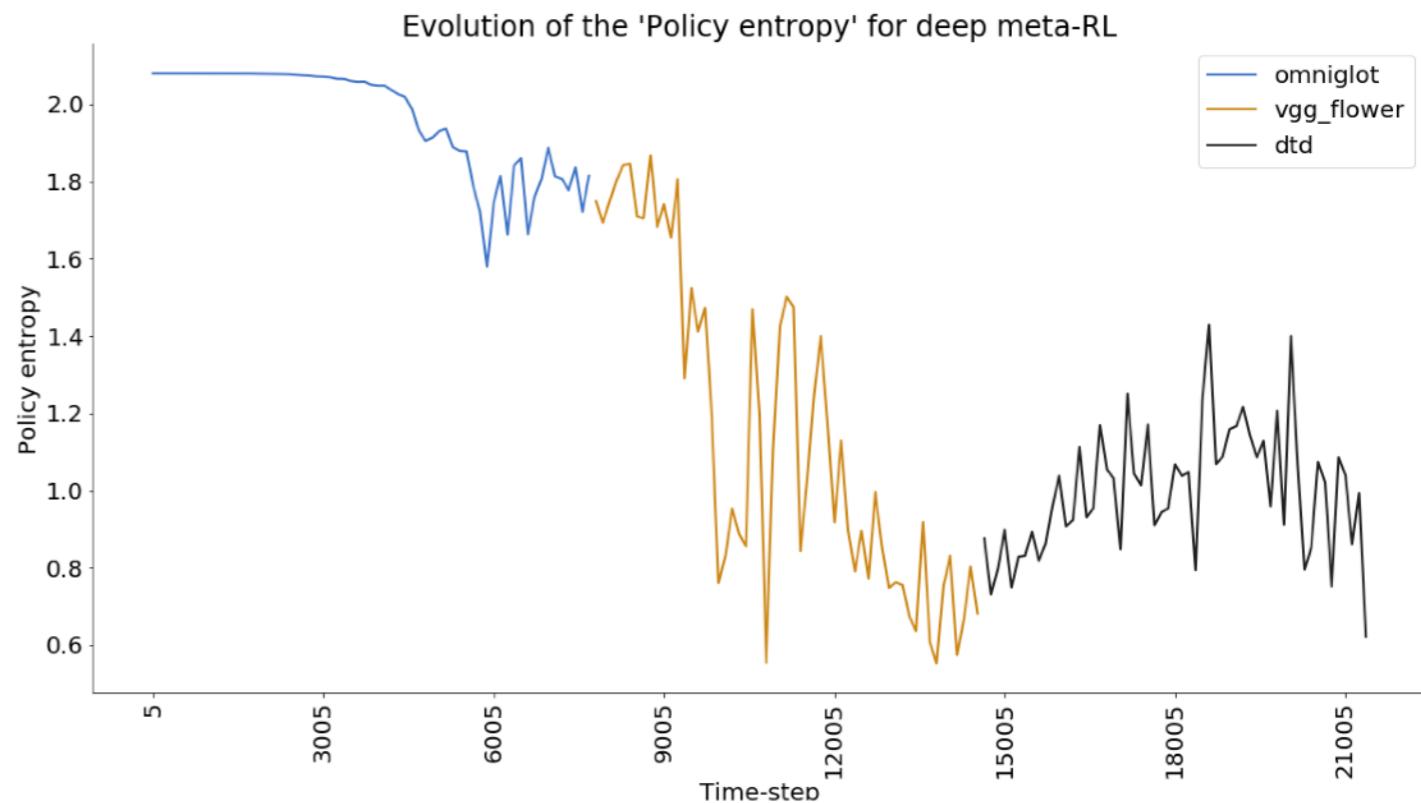
*Actions: add certain layers  
in certain locations*

## Meta-Learning

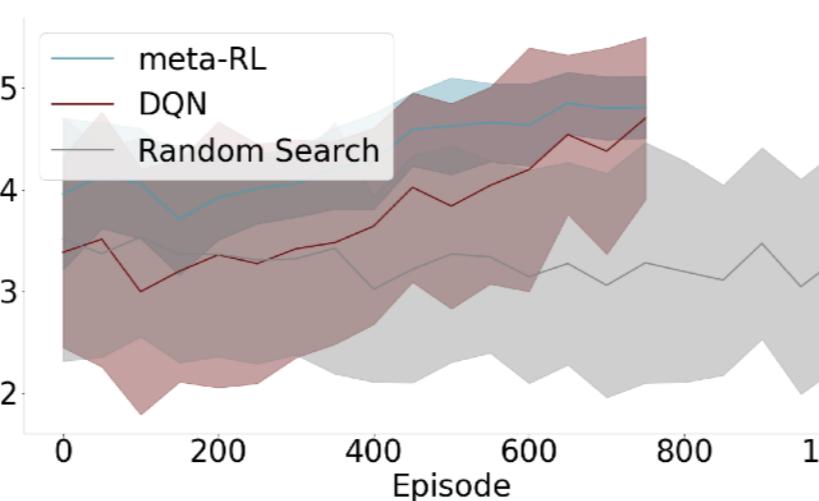
# Meta-Reinforcement Learning for NAS

Results on increasingly difficult tasks:

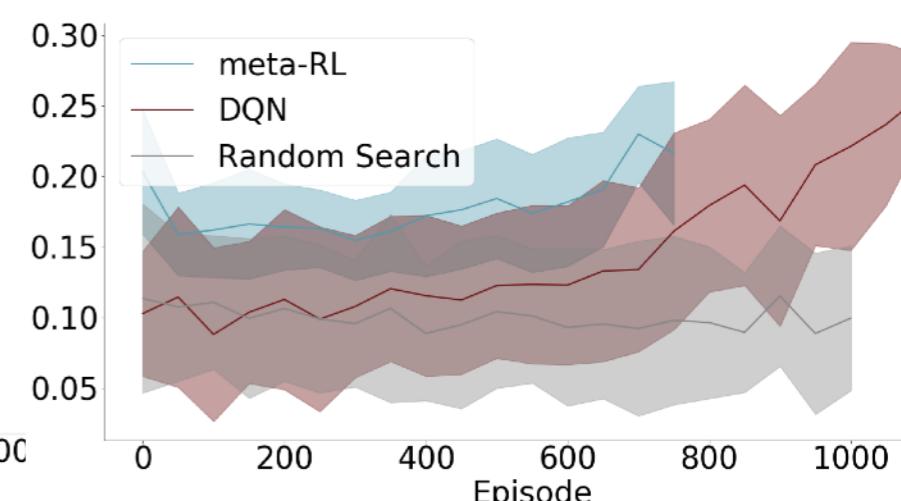
- omniglot
- vgg\_flower
- dtd



omniglot



vgg\_flower



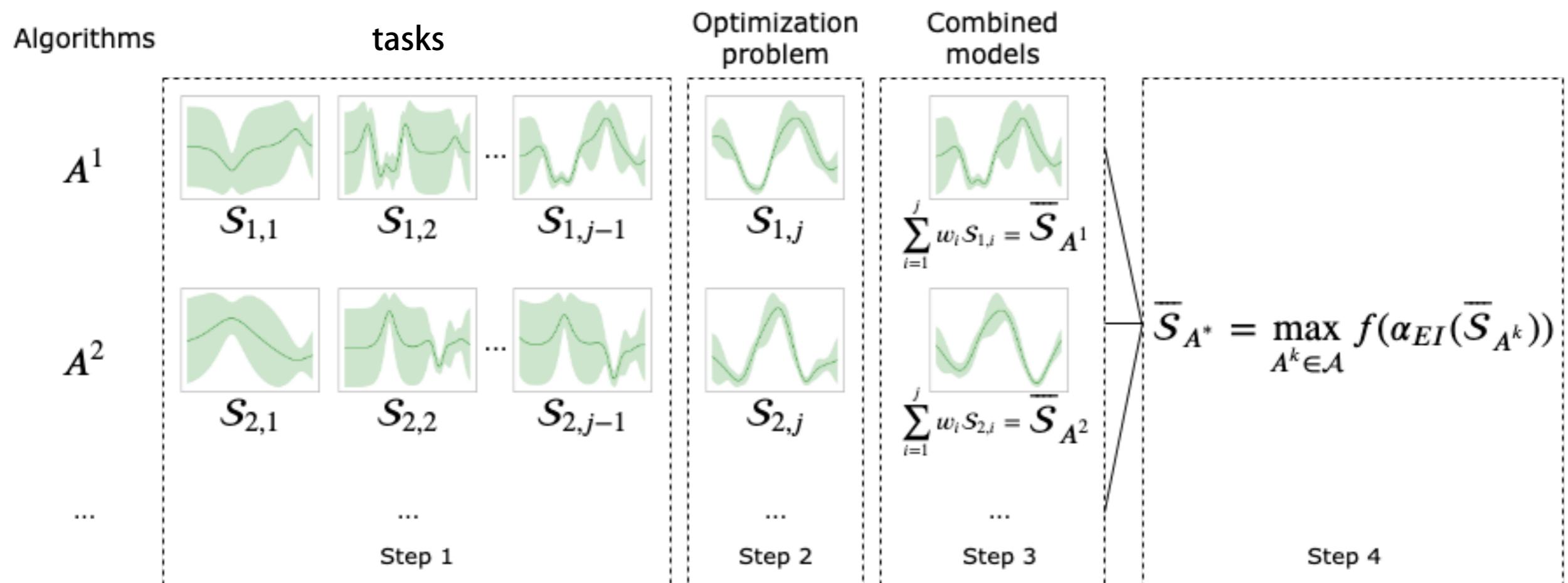
dtd

## Meta-Learning

# Meta-Learning for Bayesian Optimization

- **Surrogate model transfer**

- Store the surrogate models for previous tasks
- Learn a weighted function over them on a new task
- Much faster optimization



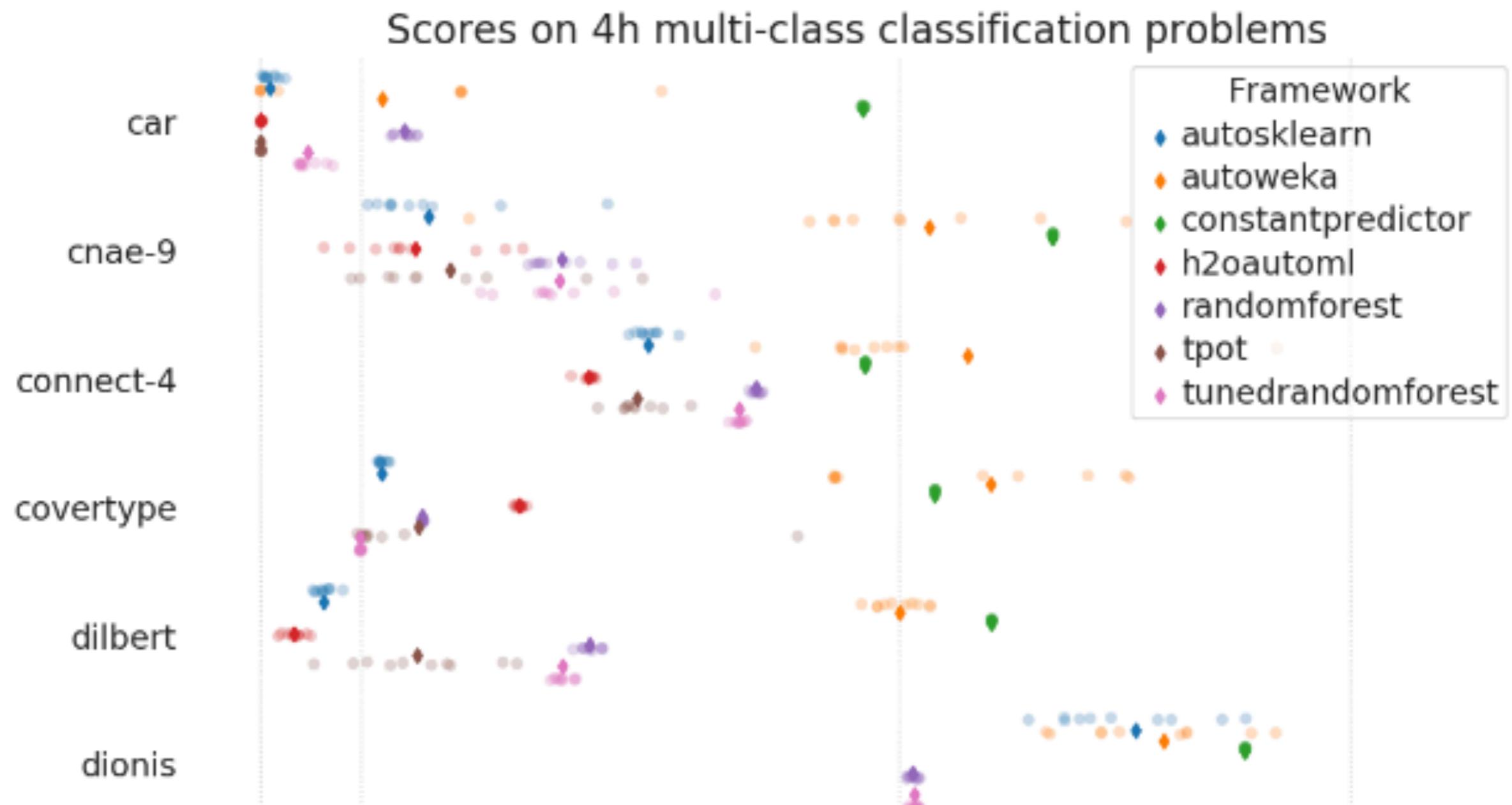
# AutoML open source tools

	Architect. search	Operators	Hyperpar. search	Improvements	Metalearnin
<u>Auto-WEKA</u>	Param. pipeline	WEKA	Bayesian Opt. (RF)		
<u>auto-sklearn</u>	Param. pipeline	sklearn	Bayesian Opt. (RF)	Ensemble	warm-start
mlr-mbo	Param. pipeline	mlr	Bayesian Opt.	multi-obj.	
BO-HB	Param. pipeline	sklearn	Tree of Parzen Estim.	Ensemble, HB	
<u>hyperopt-sklearn</u>	Param. pipeline	sklearn	Tree of Parzen Estim.		
<u>skopt</u>	Param. pipeline	sklearn	Bayesian Opt. (GP)		
<u>TPOT</u>	Evolving pipelines	sklearn	Population-based		
<u>GAMA</u>	Evolving pipelines	sklearn	Population-based	Ensemble, ASHA	
<u>ML-Plan</u>	Planning	sklearn	Interleaved		
<u>MOSAIC</u>	MCTS	config	Surrogate model		
<u>H2O AutoML</u>	Param. pipeline	H2O	Random search	Stacking	
<u>OBOE</u>	Single algorithms	sklearn	Low rank approx.	Ensembling	runtime pred
<u>Auto-Keras</u>	Param. NAS	keras	Bayesian Opt.	Net Morphisms	
<u>Auto-pyTorch</u>	Param. pipeline	pyTorch	BO-HB		

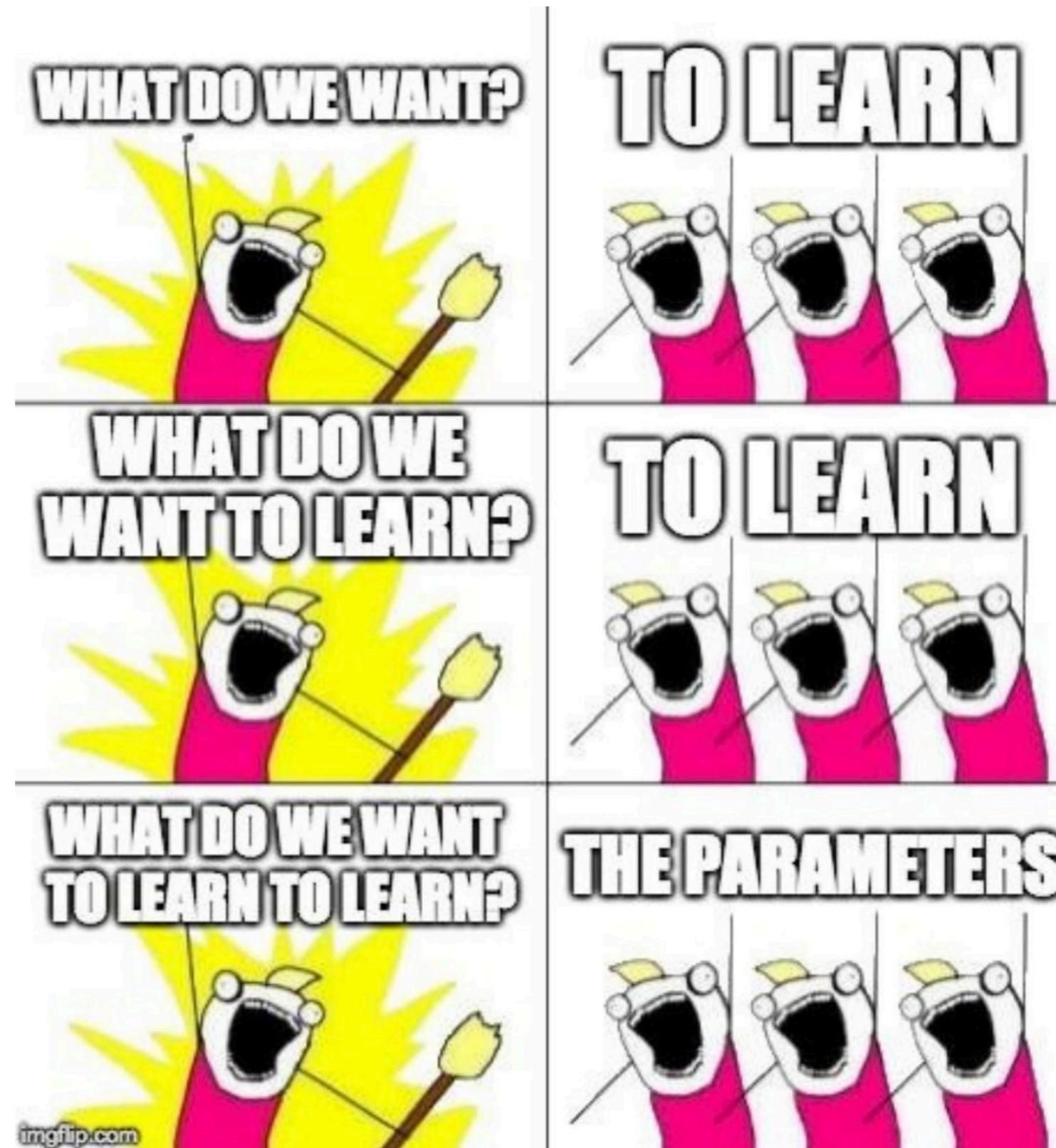
# AutoML benchmarks

Open source benchmark on <https://openml.github.io/automlbenchmark/>

- On OpenML datasets, will regularly be updated, anyone can add



*Thank you!*



[imgflip.com](https://imgflip.com)

Image: Pesah et al. 2018

**Andreas Mueller**  
**Bernd Bischl**

**Bilge Celik**

**Erin LeDell**

**Frank Hutter**

**Guiseppe Casalicchio**

**Heidi Seibold**

**Jakub Smid**

**Jan van Rijn**

**Janek Thomas**

**Joaquin Vanschoren**

**Matthias Feurer**

**Marcel Wever**

**Markus Weimer**

**Michel Lang**

**Mitar Milutinovic**

**Neeratjoy Mallik**

**Neil Lawrence**

**Pieter Gijsbers**

**Prabhant Singh**

**Sahithya Ravi**

**William Raynaut**

**Thanks to the OpenML team!**

**Join us :)**

