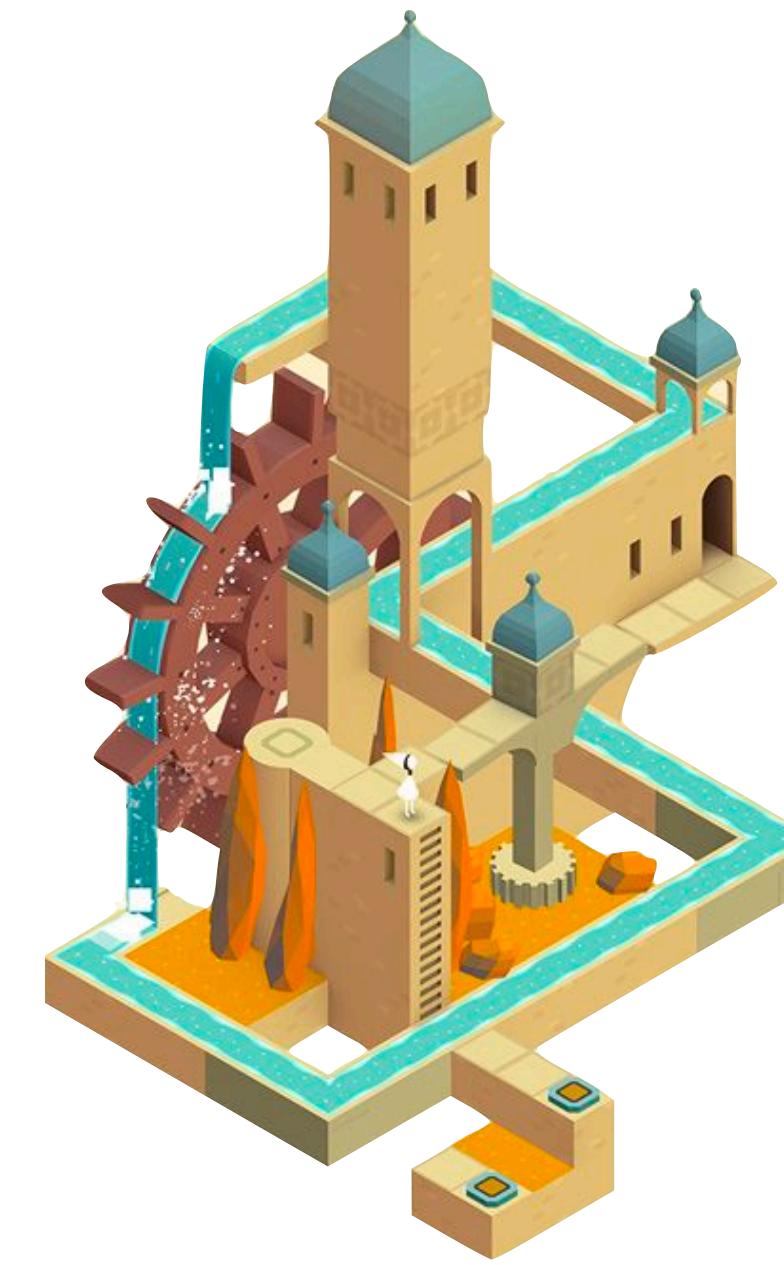


# Tutorial on Automated Machine Learning

*Open Data Science Conference  
June 10th, 2021*

Joaquin Vanschoren & Pieter Gijsbers  
Eindhoven University of Technology

*image credit: ustwo*



## Schedule

*Learning to Automate ML*

*Hands-on (OpenML, GAMA, AutoML benchmark)*

- 50min - Joaquin Vanschoren

- 40min - Pieter Gijsbers

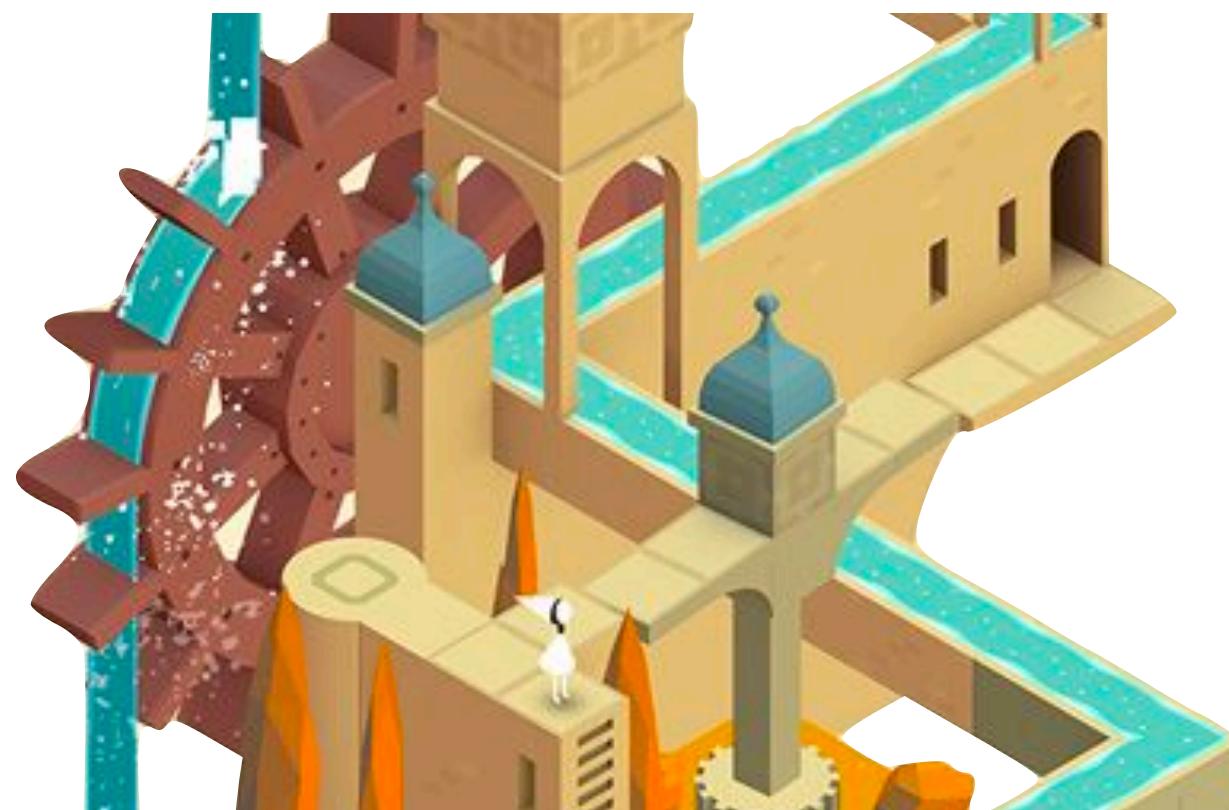
**Materials:** <https://git.io/JZIyl>

# Overview



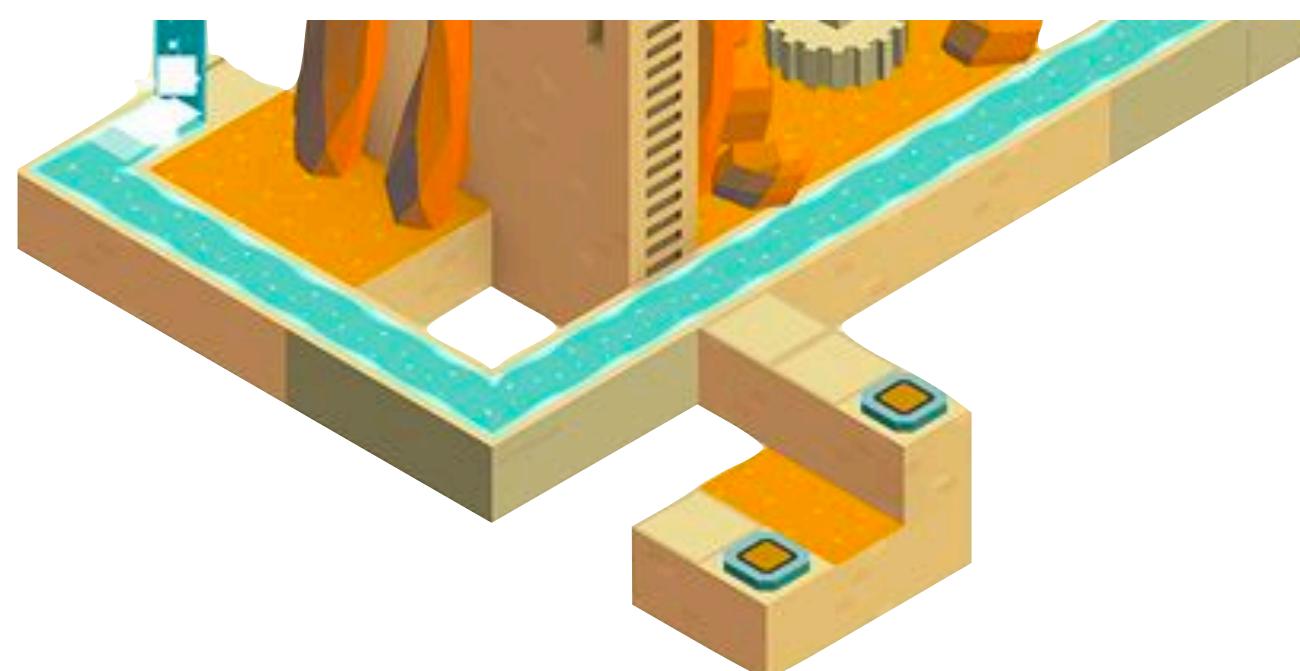
**Part 1: Why automate machine learning?**

High-level goals



**Part 2: How AutoML works**

The machinery



**Part 3: Learning how to do AutoML**

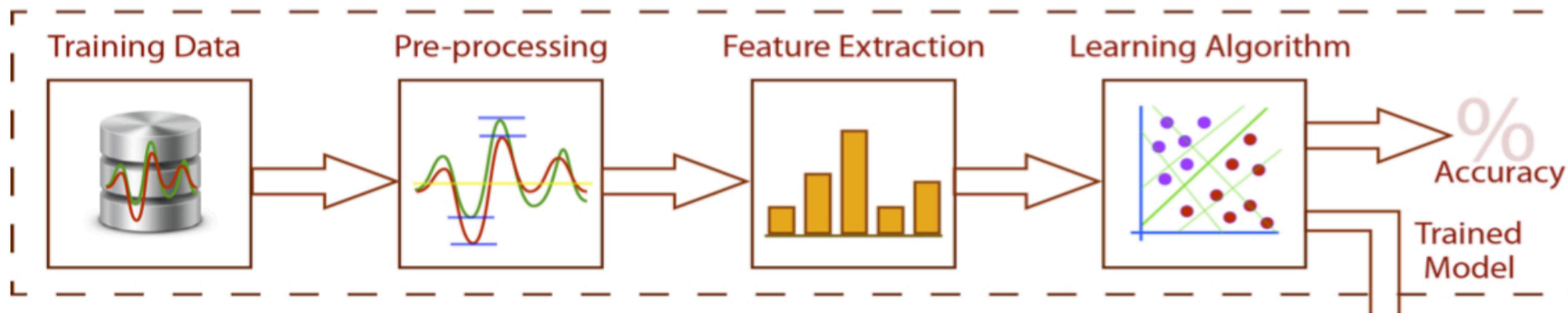
Closing the loop



## **Part 1: *Why* automate machine learning?**

### High-level goals

# *Doing machine learning requires lots of expertise and exploration*



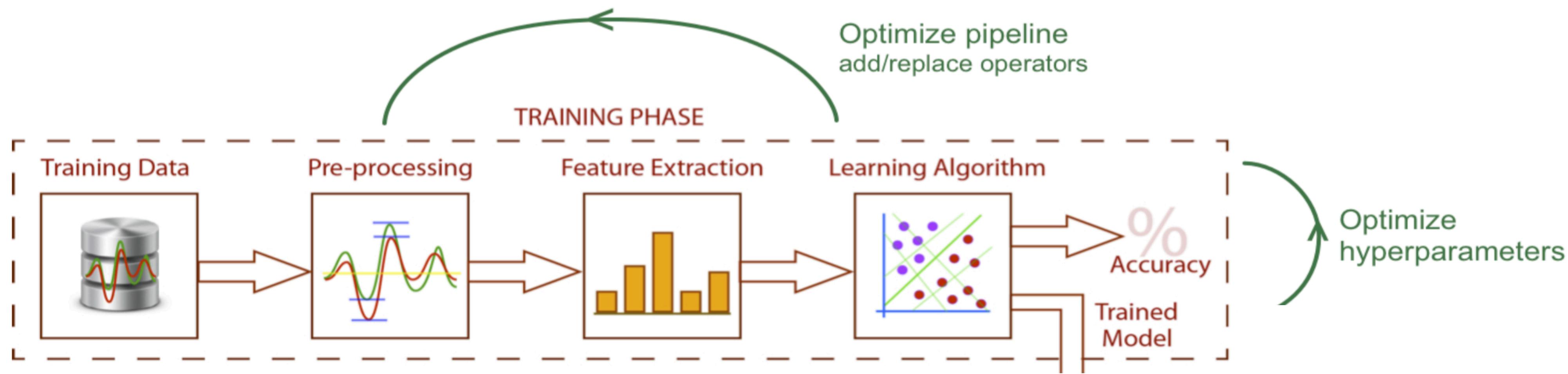
Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,...

# *Doing machine learning requires lots of expertise and exploration*



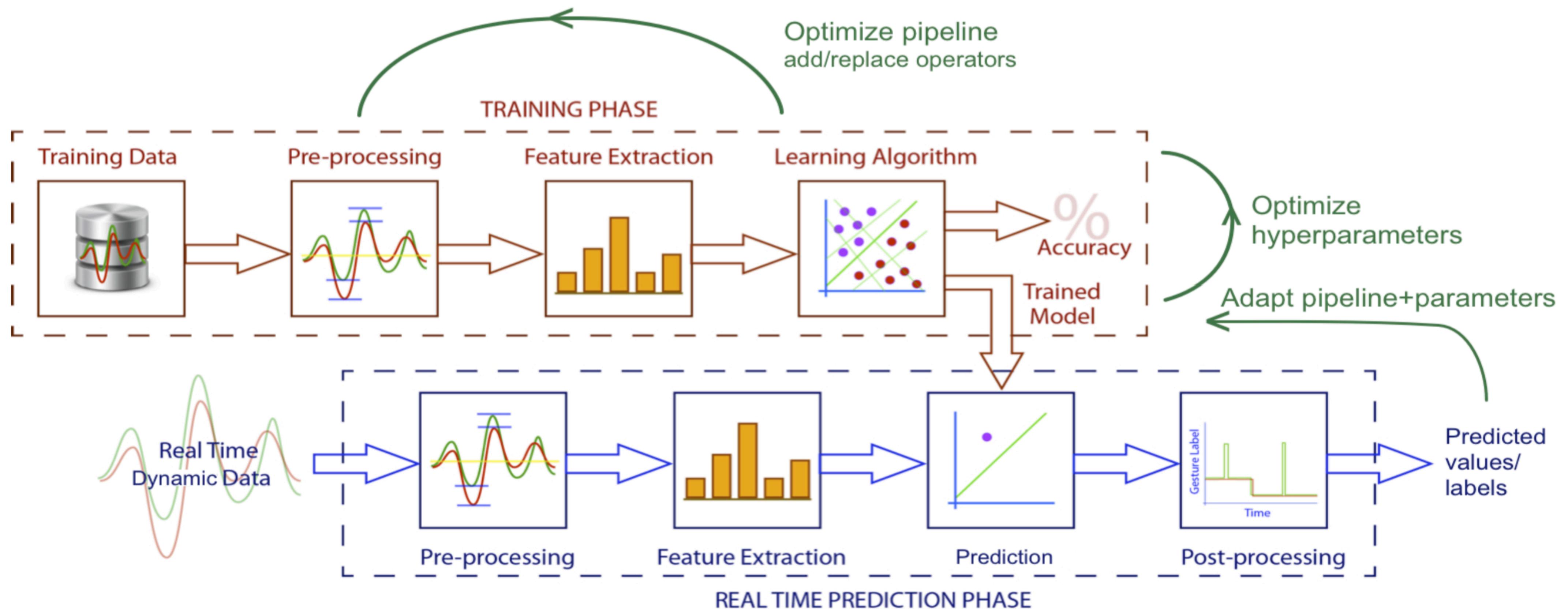
Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,...

# *Doing machine learning requires lots of expertise and exploration*



Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,...

# *Doing machine learning requires lots of expertise and exploration*

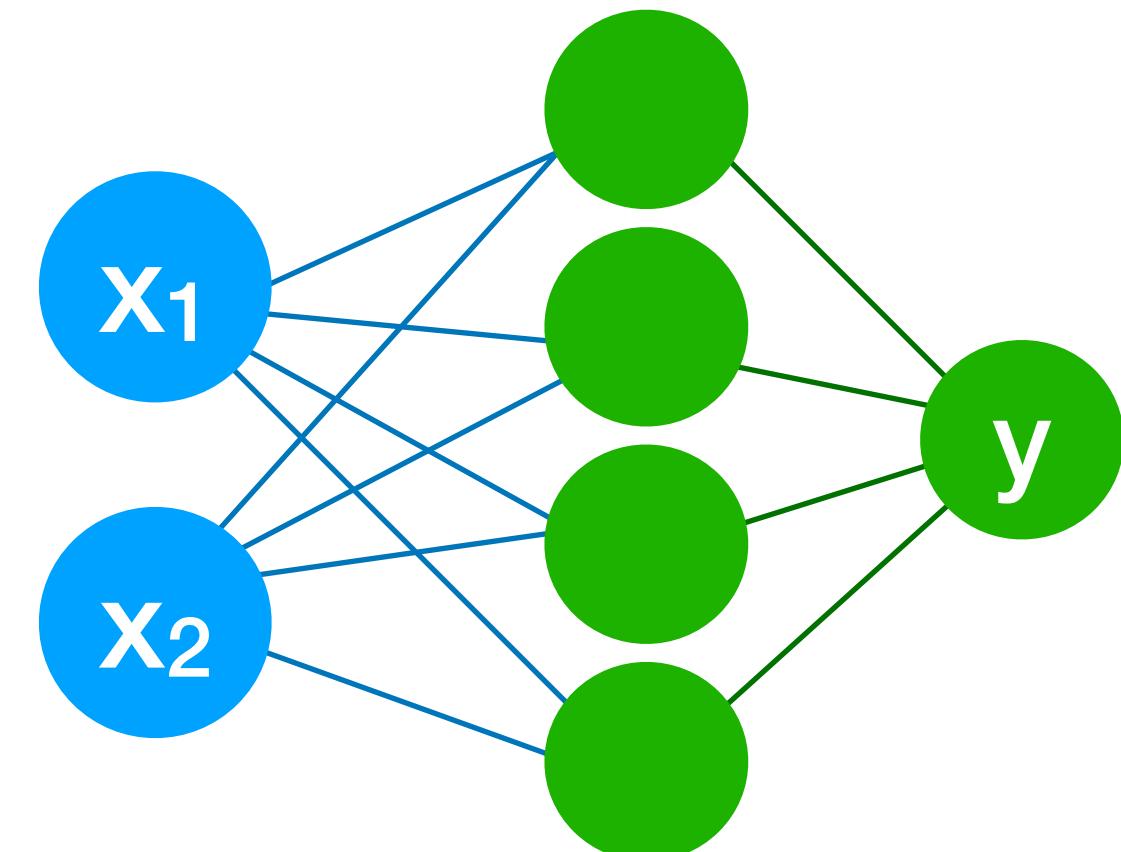


Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,...

# *Doing machine learning requires lots of expertise and exploration*

Architecture:

- Type of operators
- Size of layers
- Filter sizes
- Skip connections
- Pre-trained layers
- Transformers
- ...



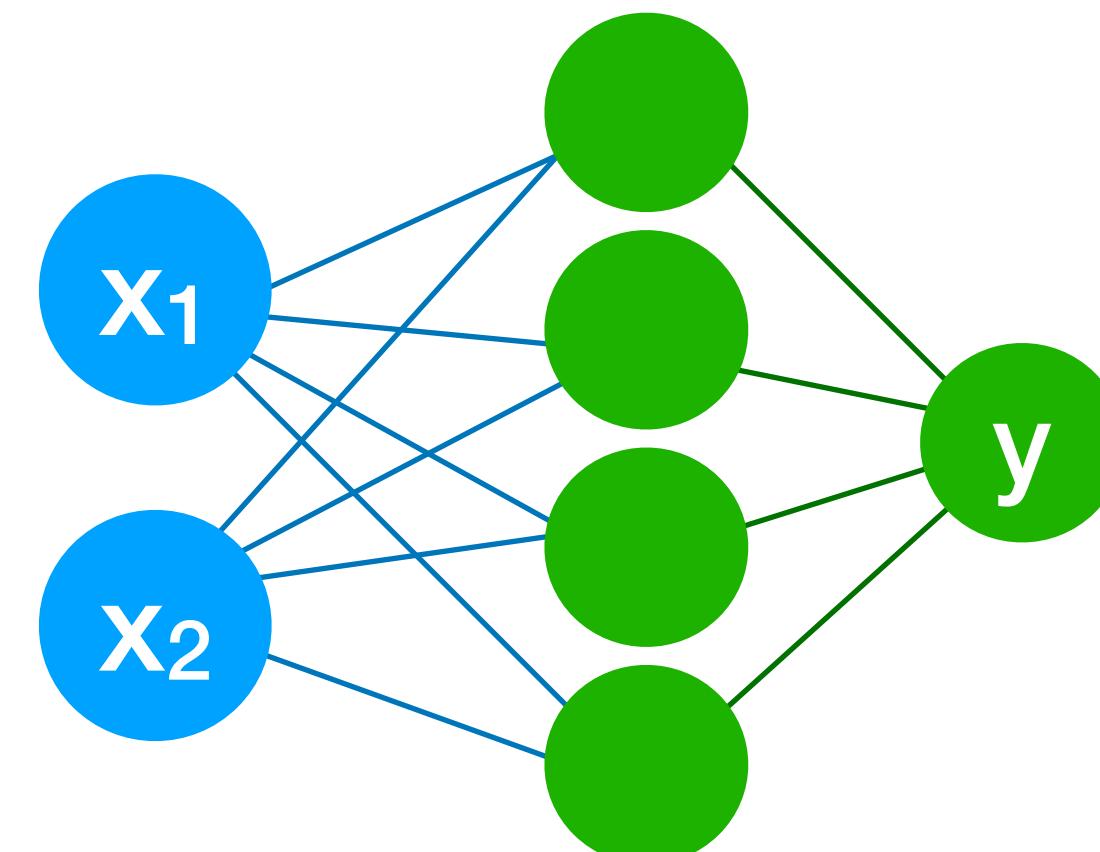
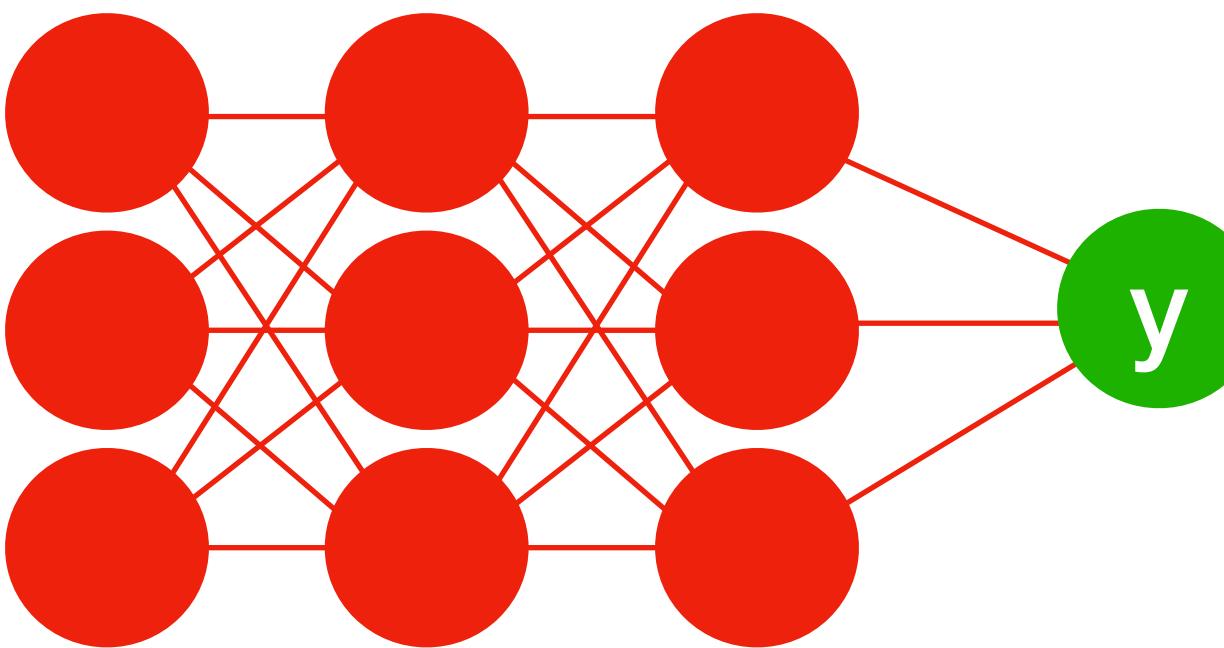
Optimization:

- Gradient descent
- hyperparameters
- Regularization
- ...

# *Doing machine learning requires lots of expertise and exploration*

Architecture:

- Type of operators
- Size of layers
- Filter sizes
- Skip connections
- Pre-trained layers
- Transformers
- ...



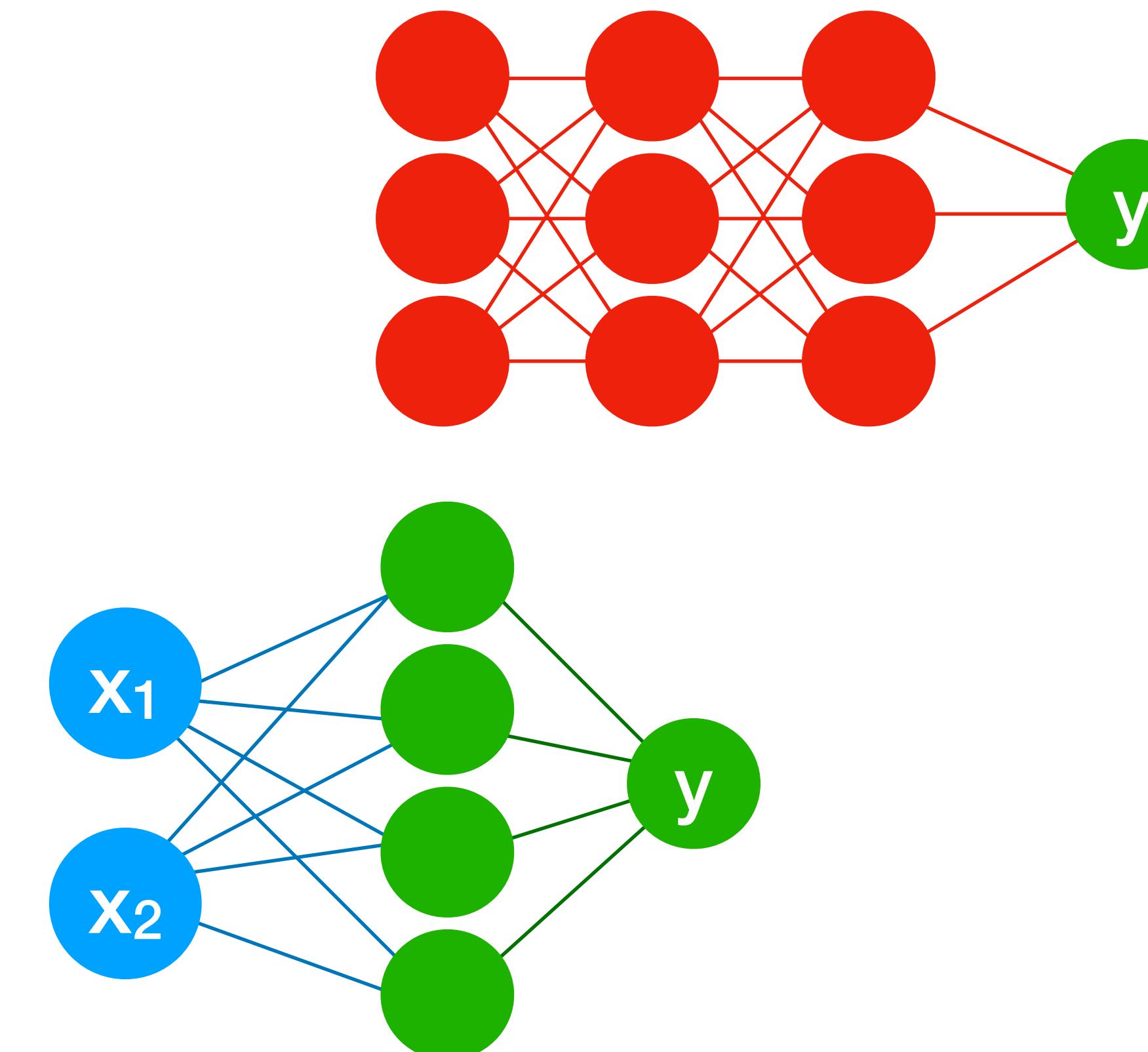
Optimization:

- Gradient descent
- hyperparameters
- Regularization
- ...

# *Doing machine learning requires lots of expertise and exploration*

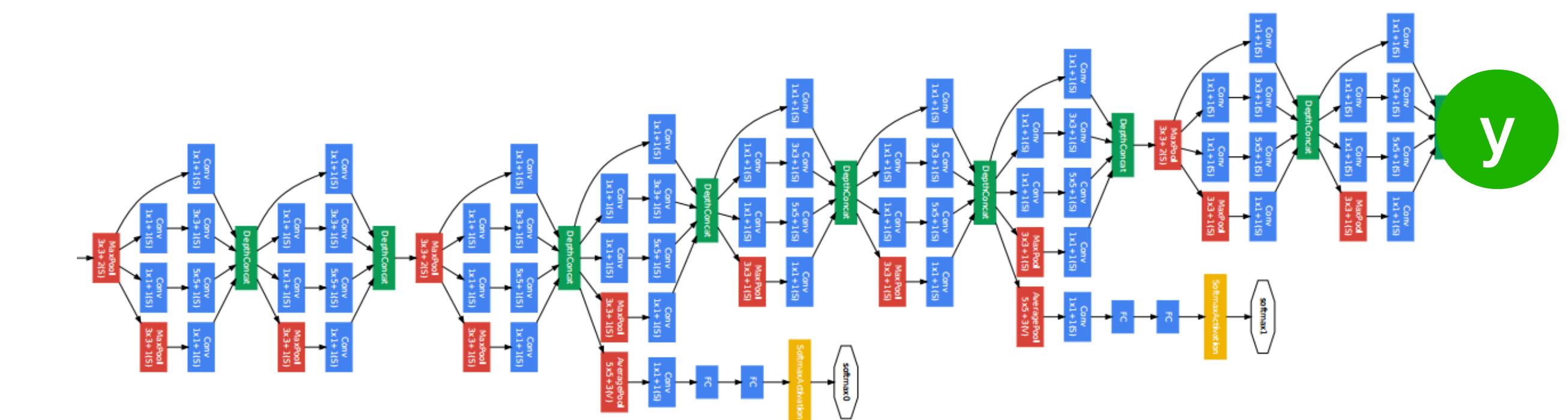
Architecture:

- Type of operators
- Size of layers
- Filter sizes
- Skip connections
- Pre-trained layers
- Transformers
- ...



Optimization:

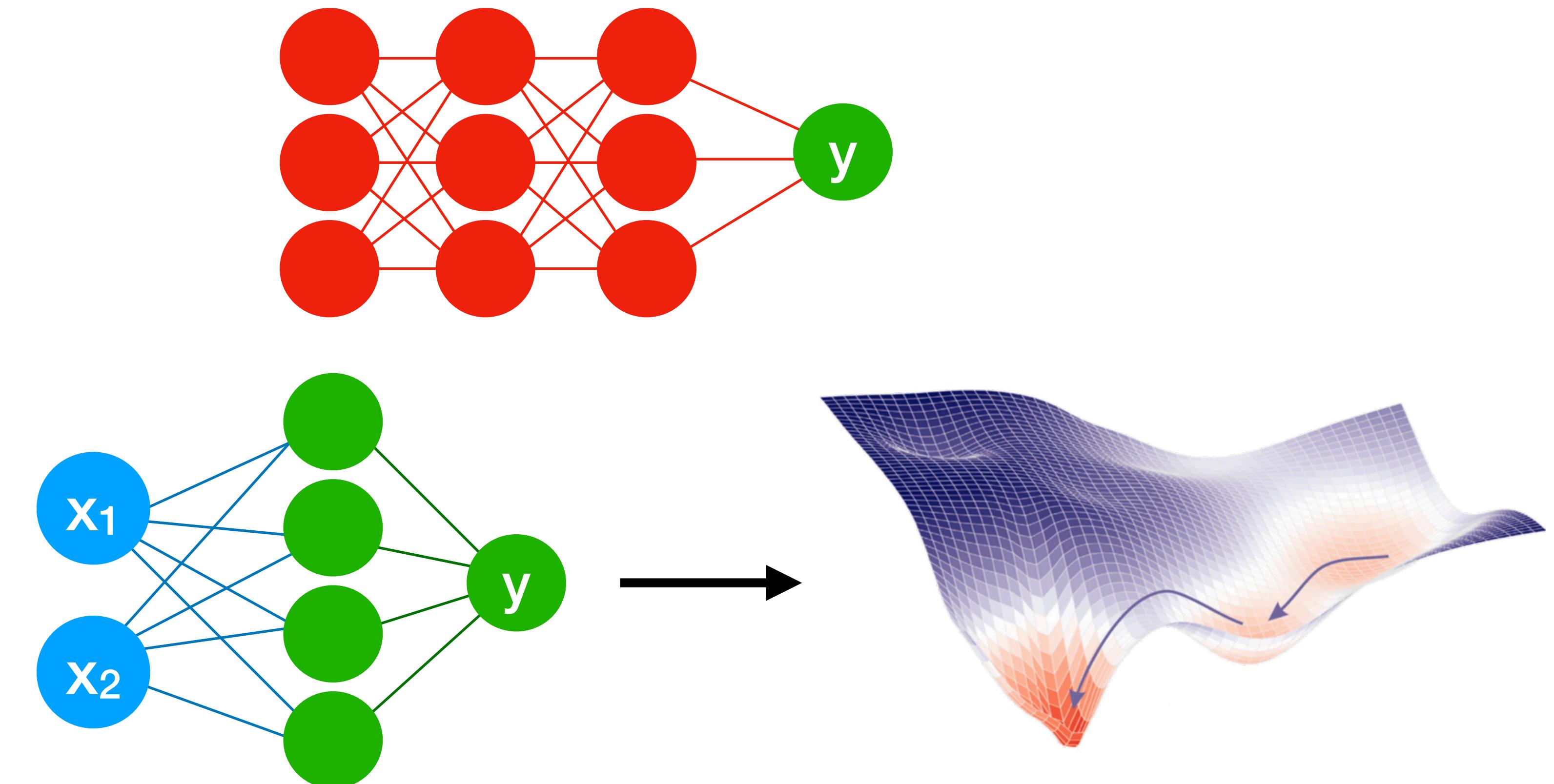
- Gradient descent
- hyperparameters
- Regularization
- ...



# *Doing machine learning requires lots of expertise and exploration*

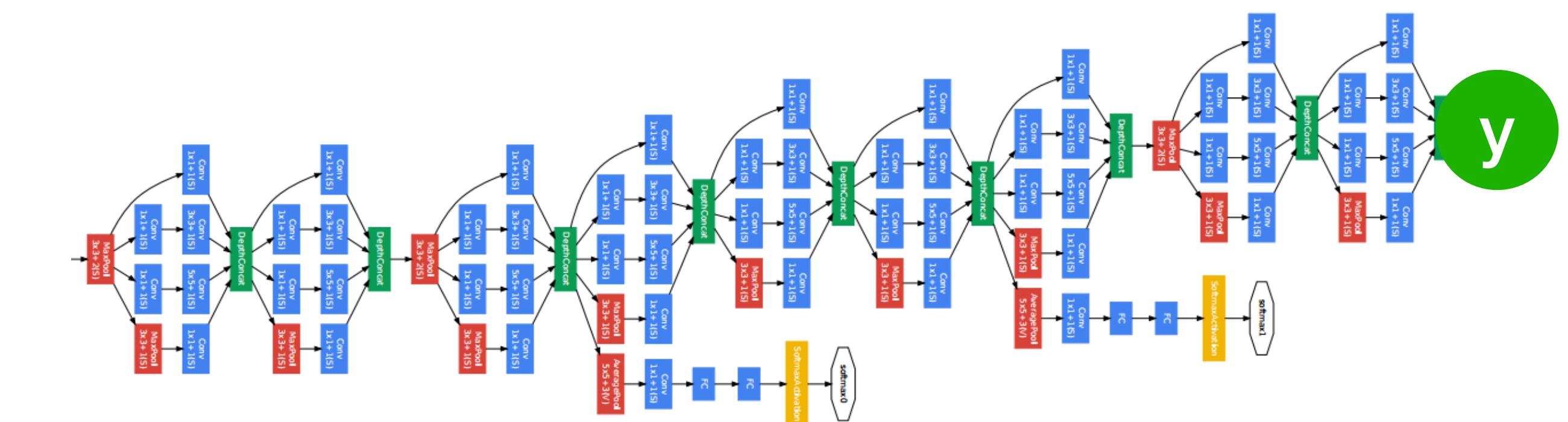
Architecture:

- Type of operators
- Size of layers
- Filter sizes
- Skip connections
- Pre-trained layers
- Transformers
- ...



Optimization:

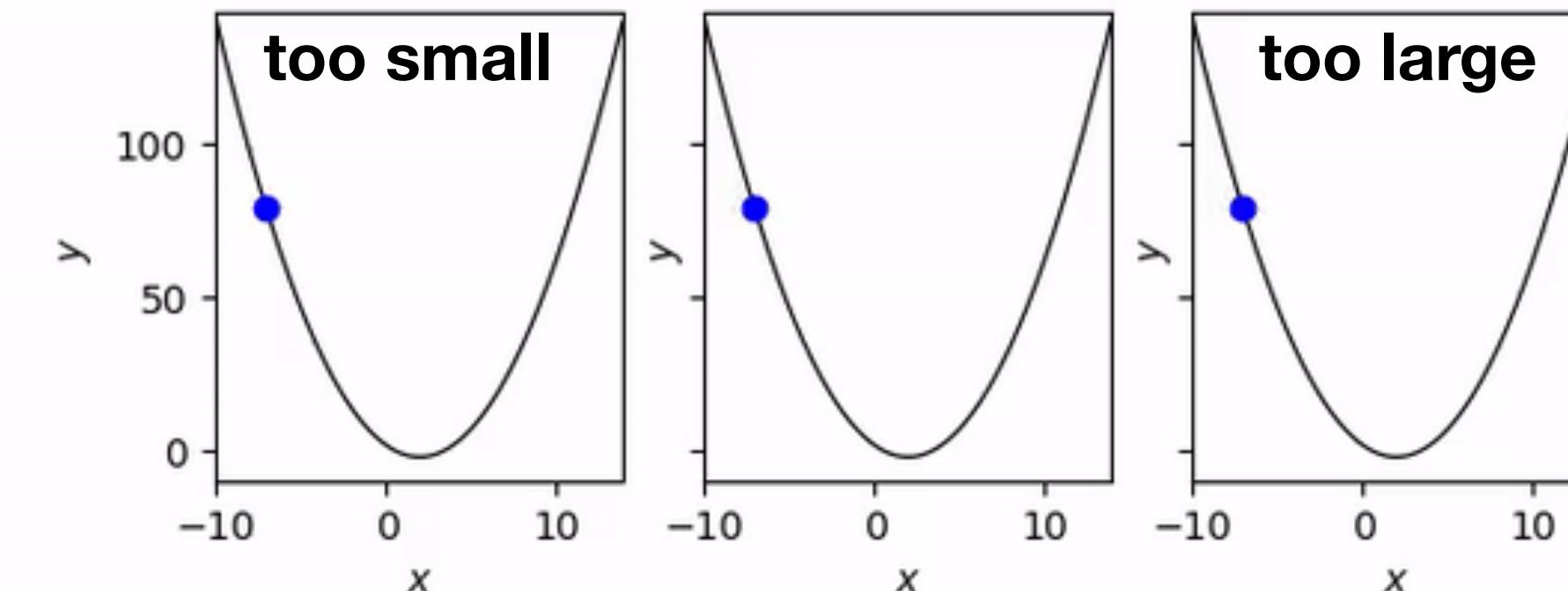
- Gradient descent
- hyperparameters
- Regularization
- ...



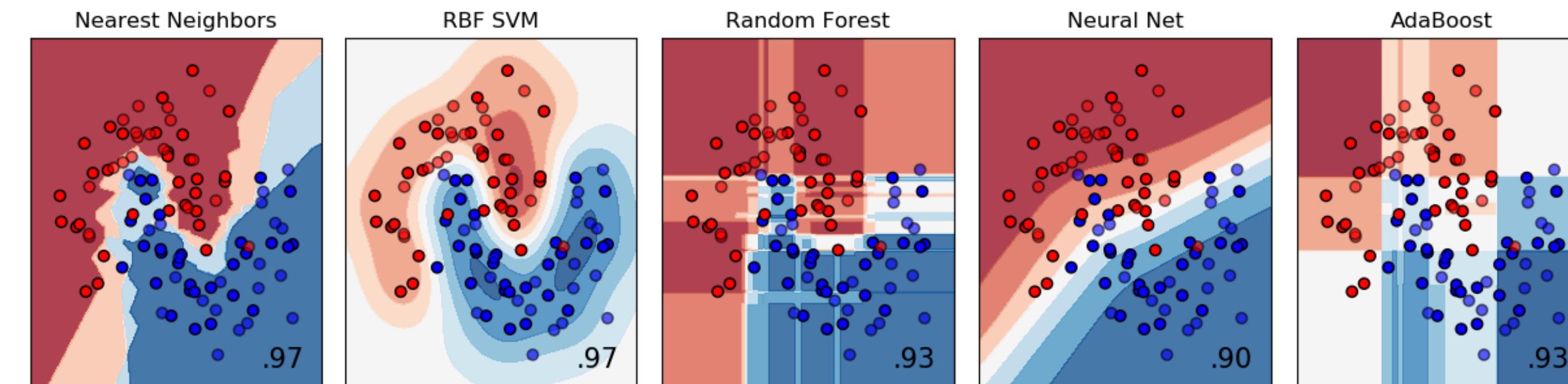
# Hyperparameters

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

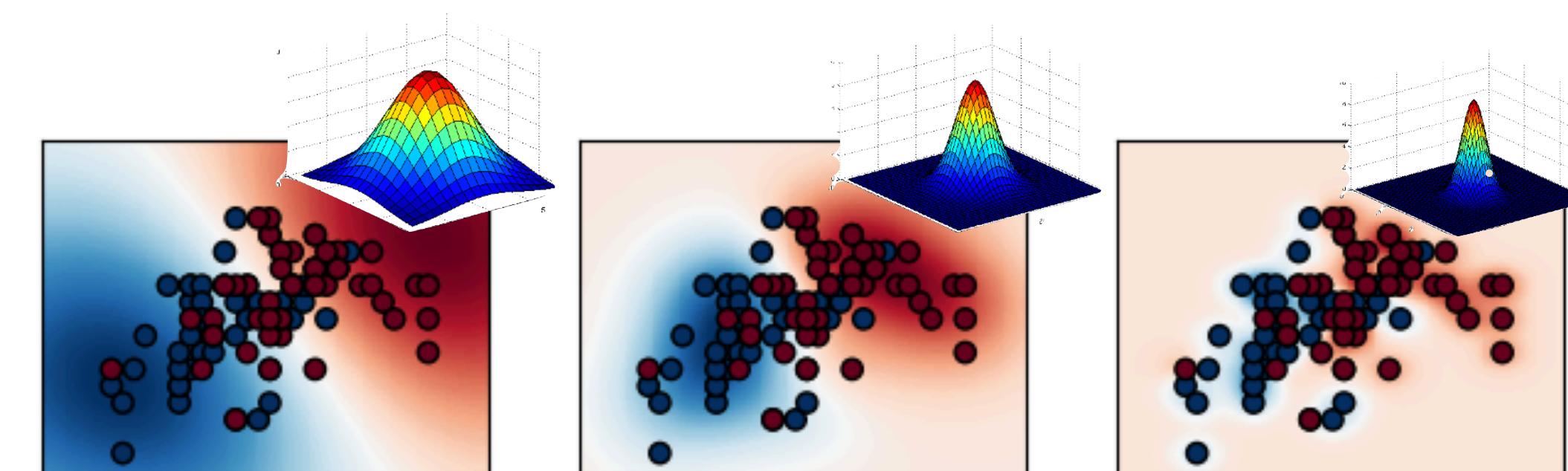
- Numeric
  - e.g. learning rate



- Categorical
  - e.g. classifier



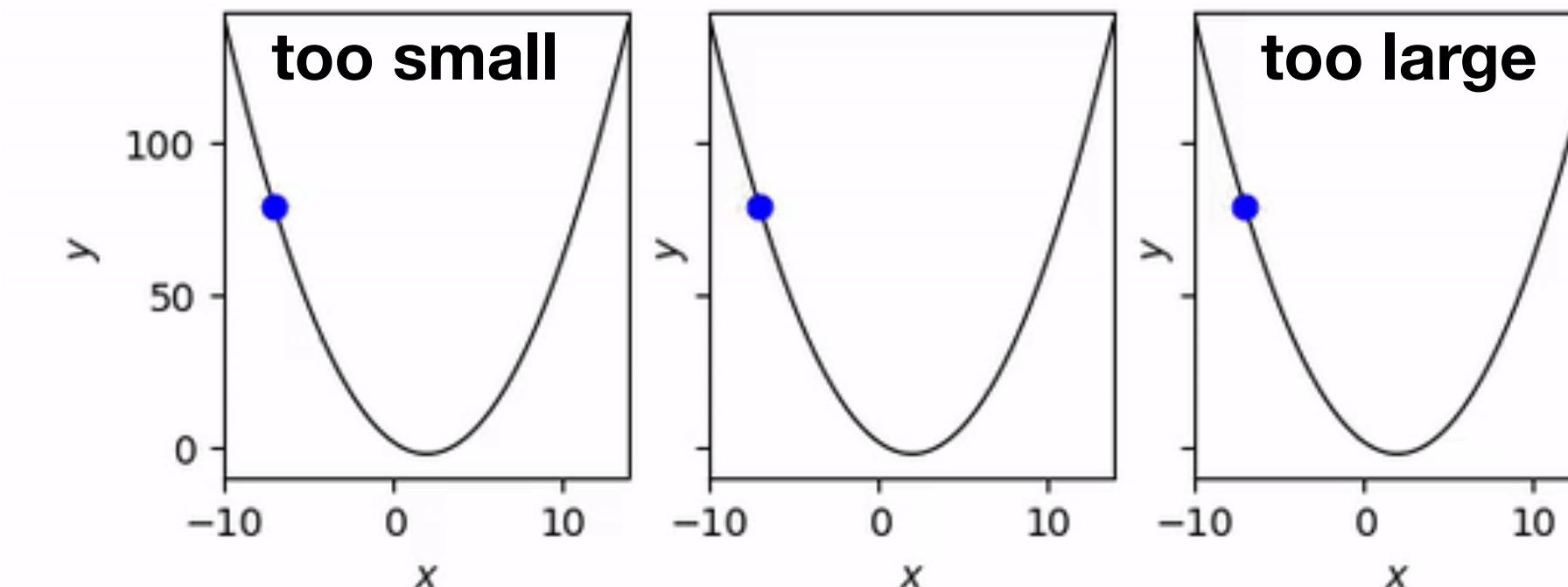
- Conditional
  - SVM -> kernel?  
RBF kernel -> gamma?



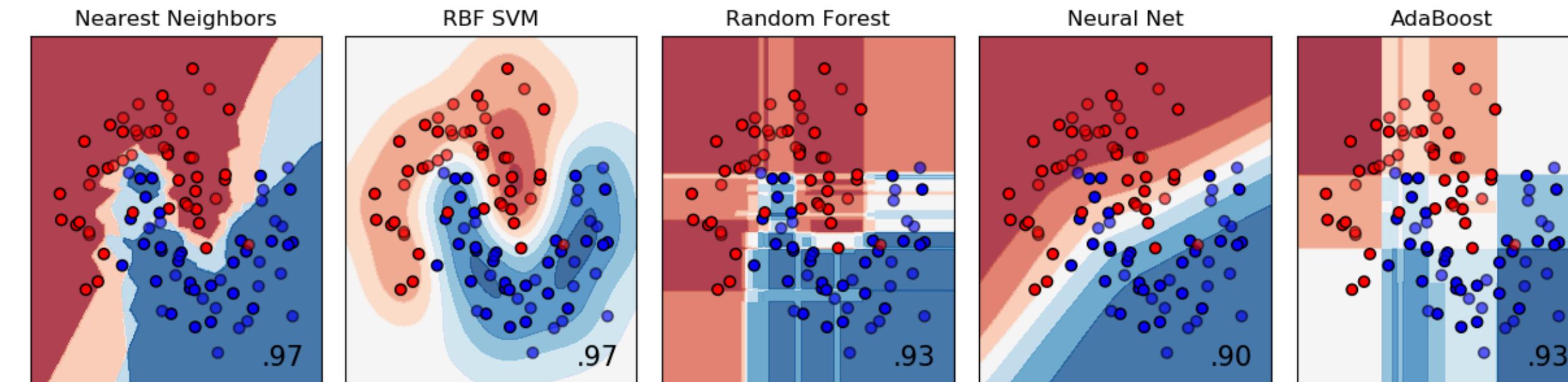
# Hyperparameters

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

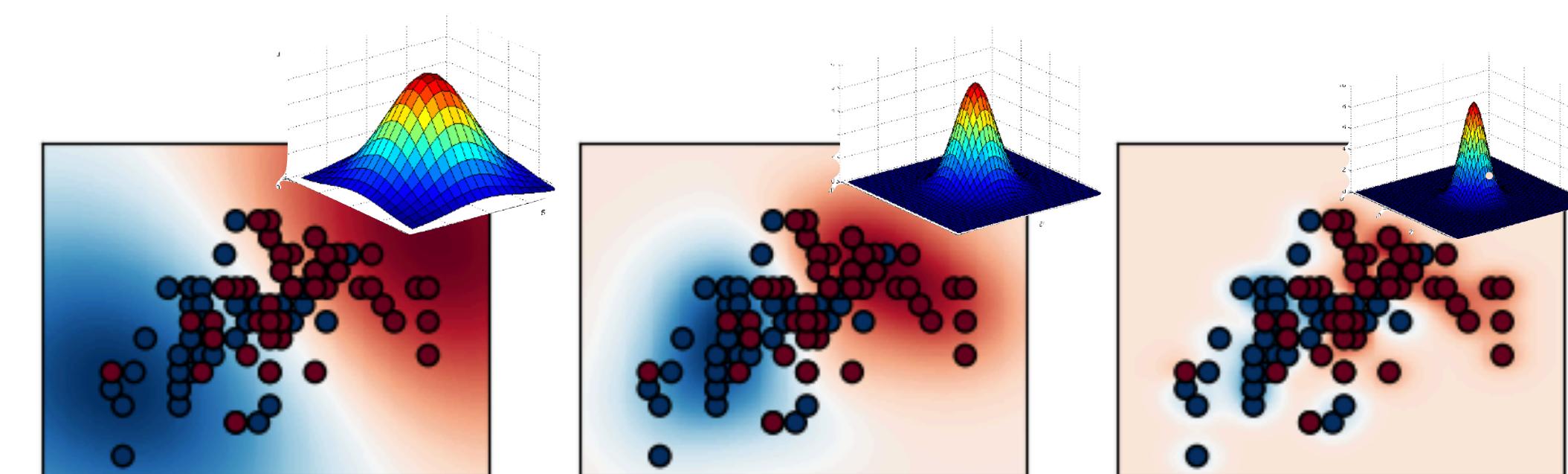
- Numeric
  - e.g. learning rate



- Categorical
  - e.g. classifier

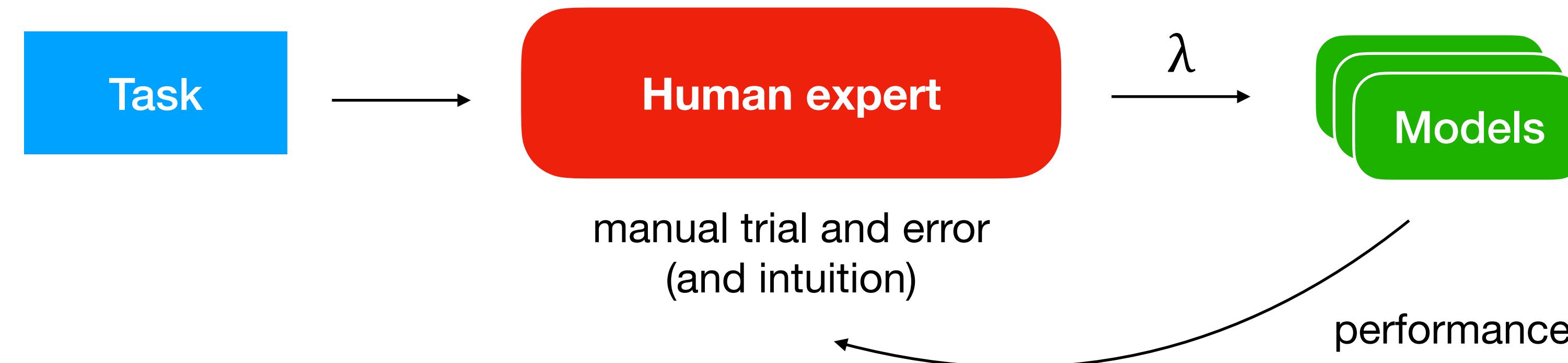


- Conditional
  - SVM -> kernel?  
RBF kernel -> gamma?

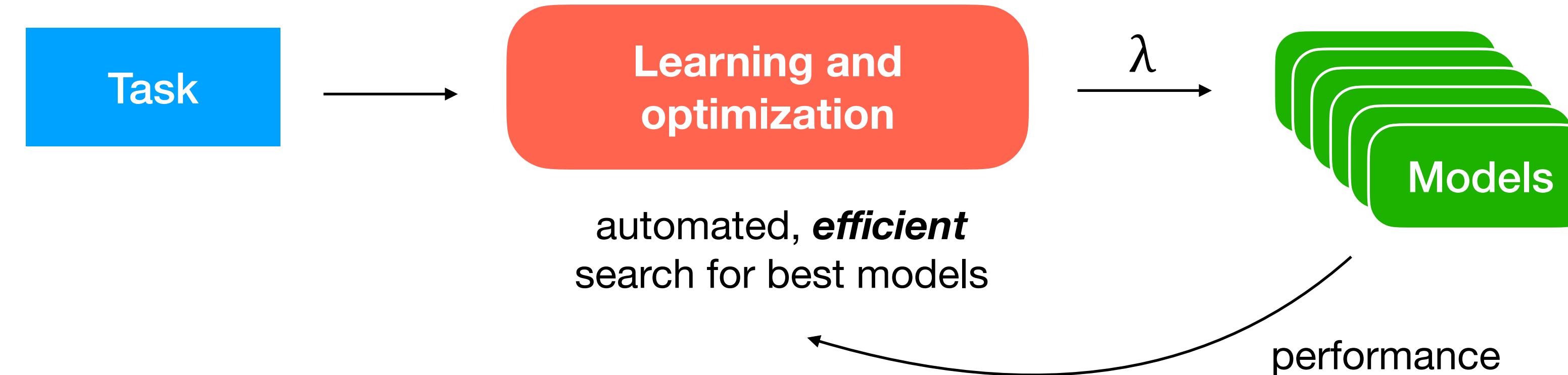


# Automatic Machine Learning (AutoML)

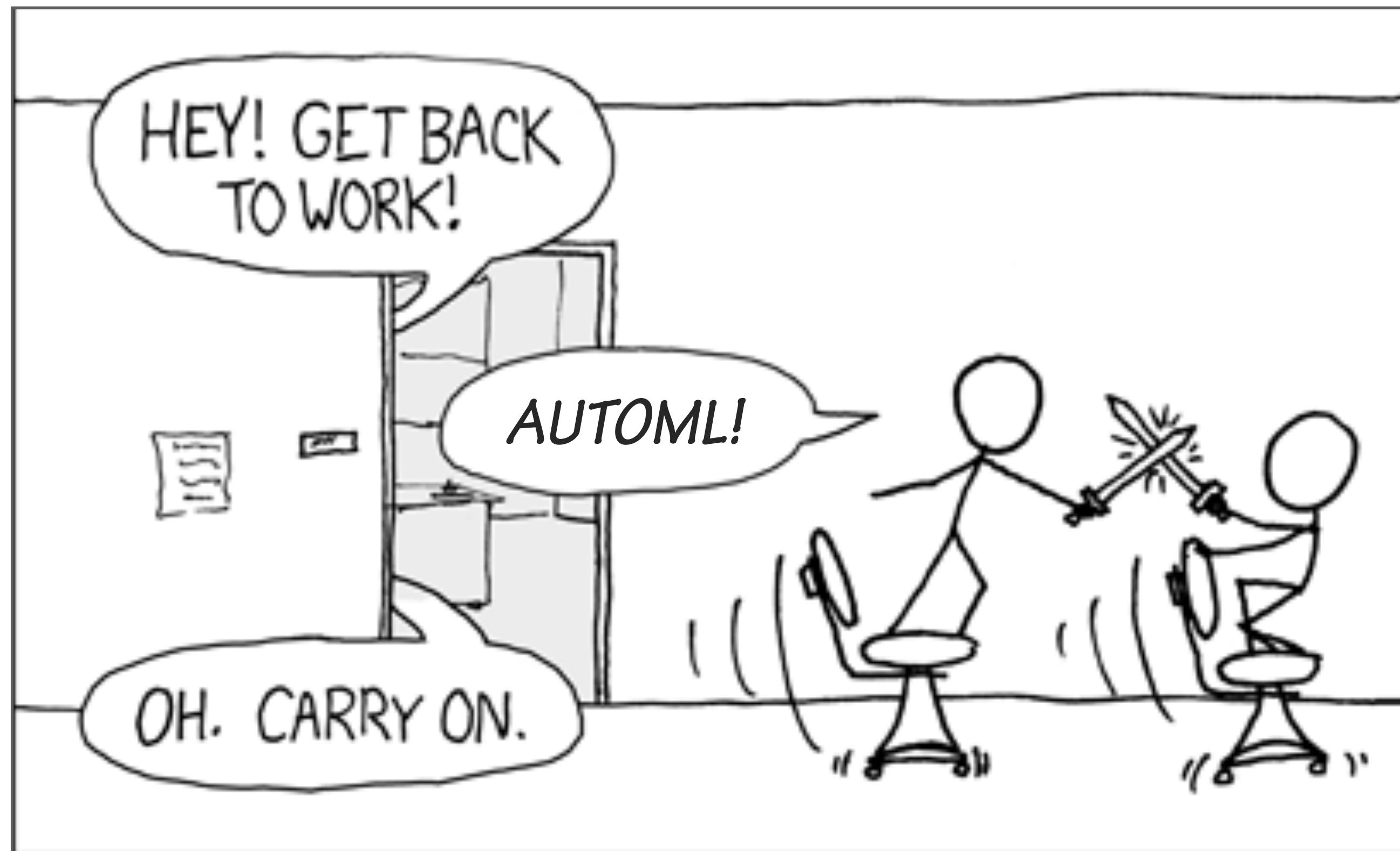
Manual machine learning



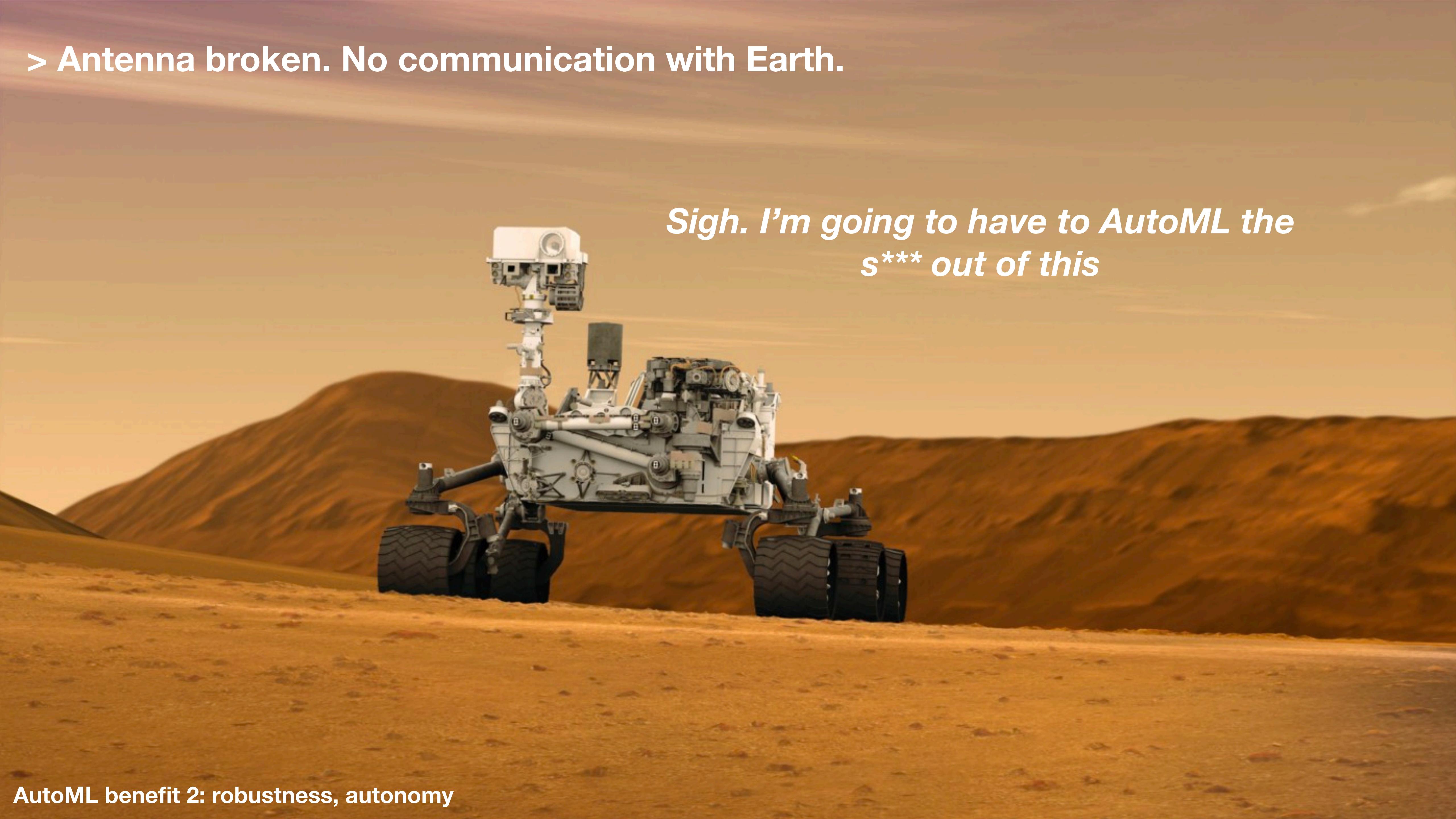
AutoML: build models in a *data-driven, intelligent, purposeful* way



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

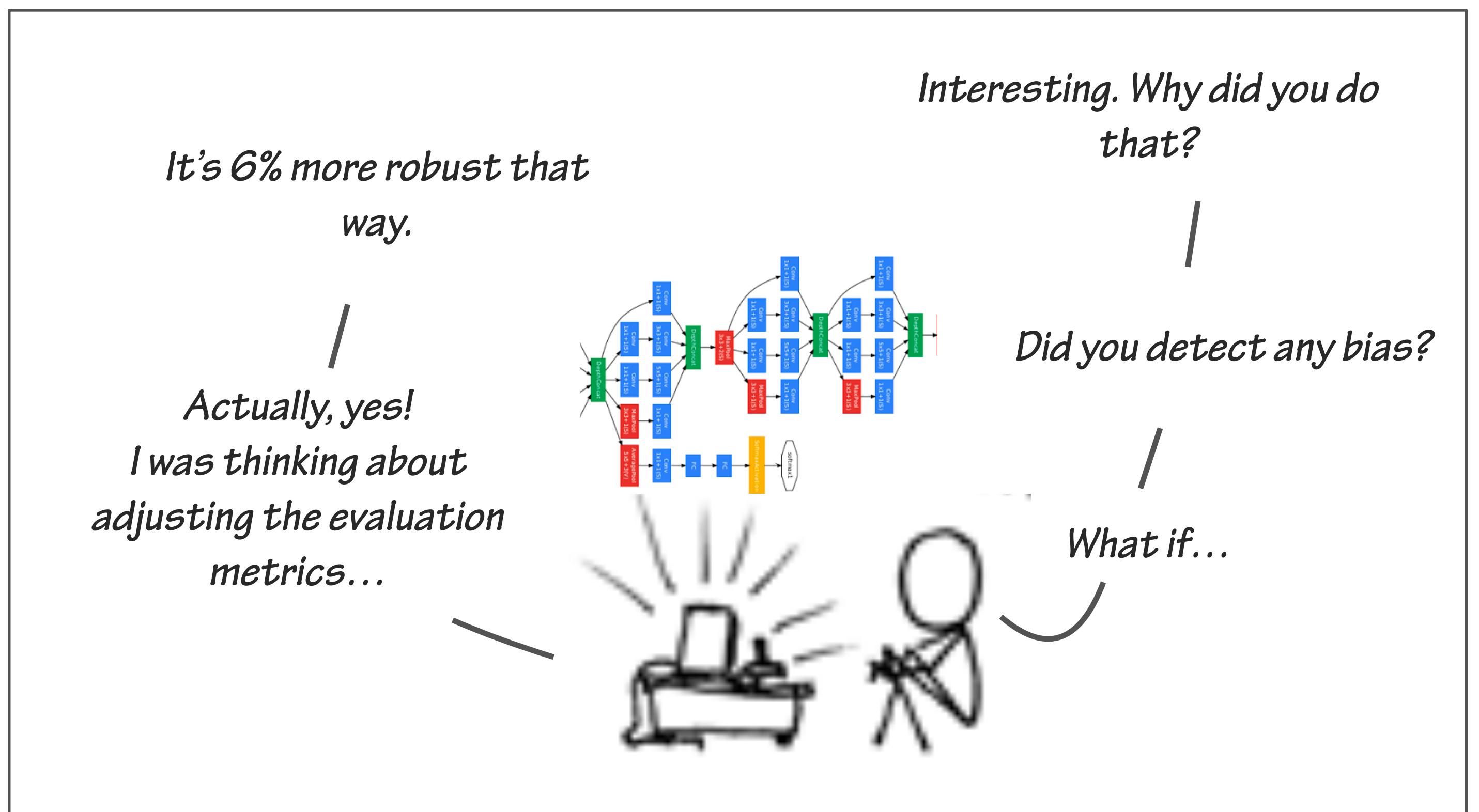


> Antenna broken. No communication with Earth.



*Sigh. I'm going to have to AutoML the  
s\*\*\* out of this*

# *Many open challenges in AutoML!*



Adapted from XKCD (Randall Munroe)

# *Many open challenges in AutoML!*

Human in the loop

Explainable process/models

Ask for help/data

Self-assess trustworthiness

Real-world constraints

Meta-learning

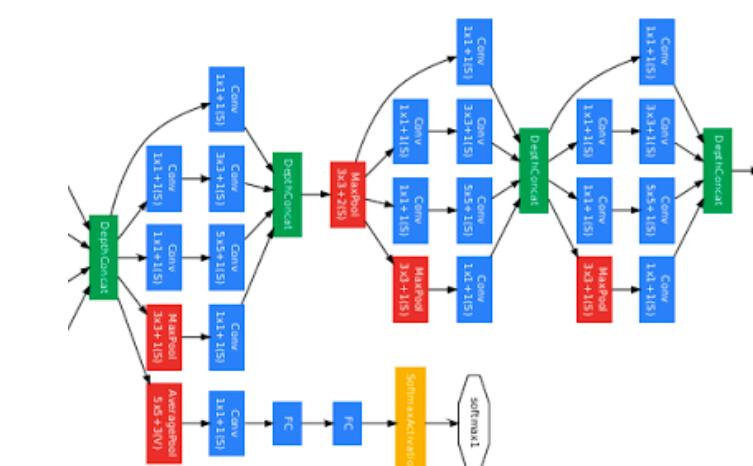
*It's 6% more robust that  
way.*

*Actually, yes!  
I was thinking about  
adjusting the evaluation  
metrics...*

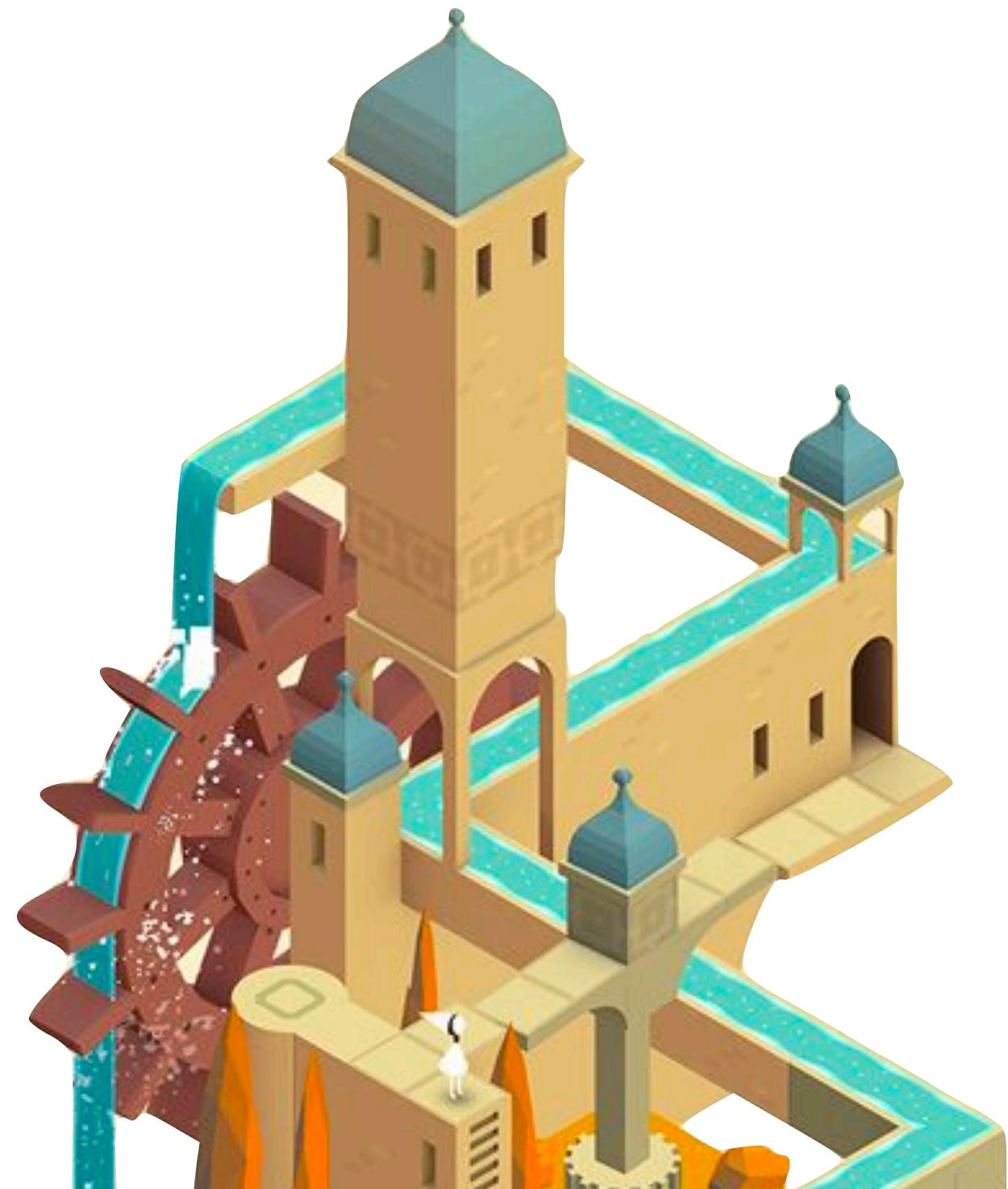
*Interesting. Why did you do  
that?*

*Did you detect any bias?*

*What if...*



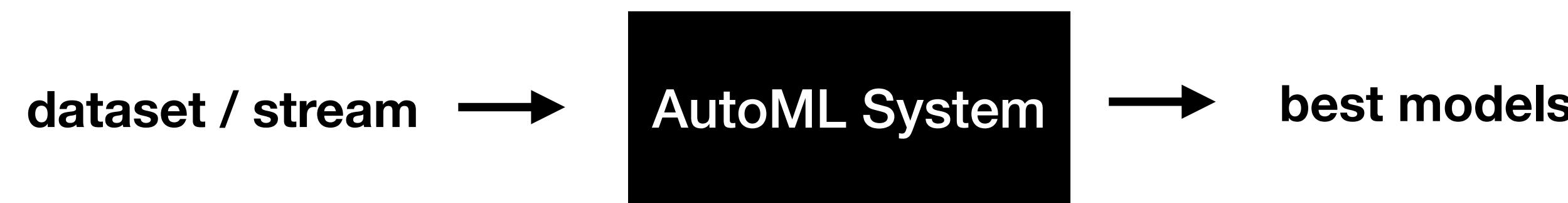
# Overview



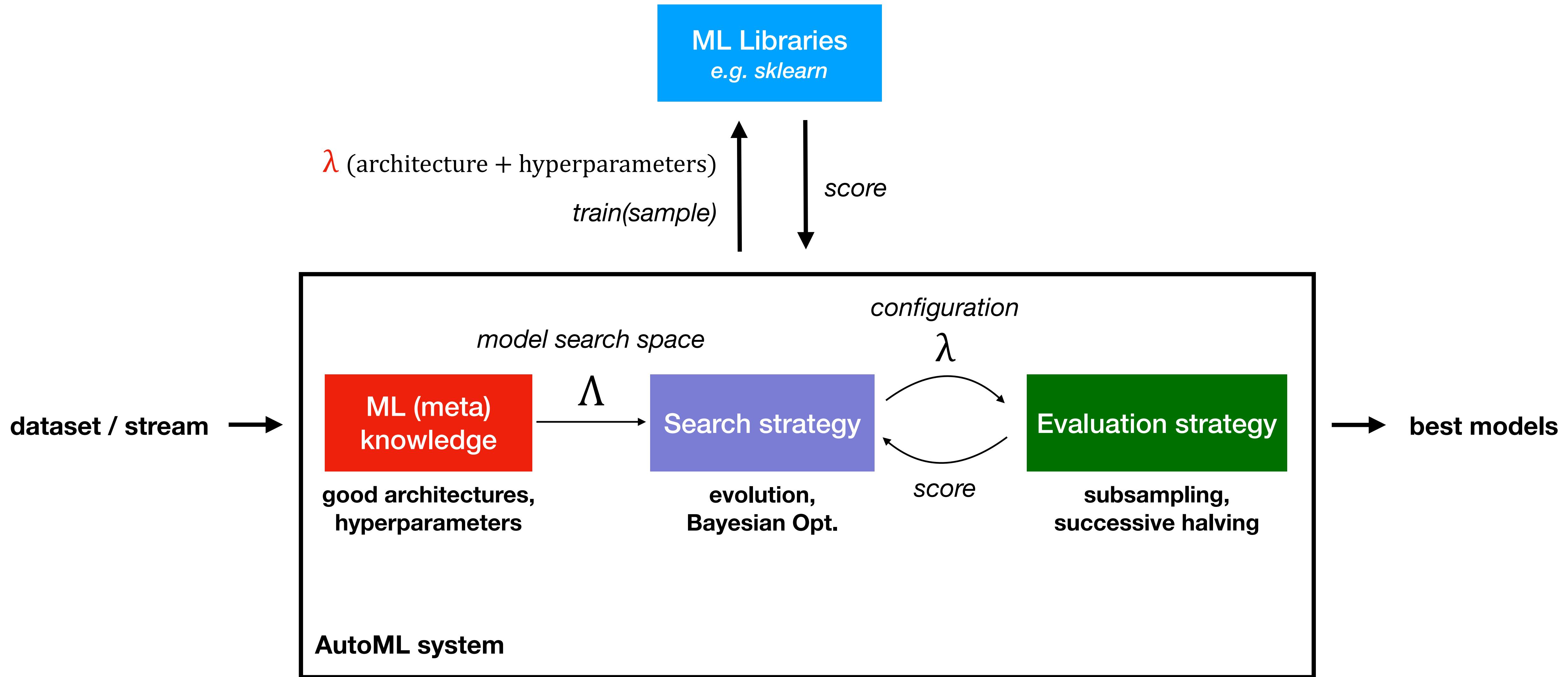
## Part 2: *How AutoML works*

The machinery

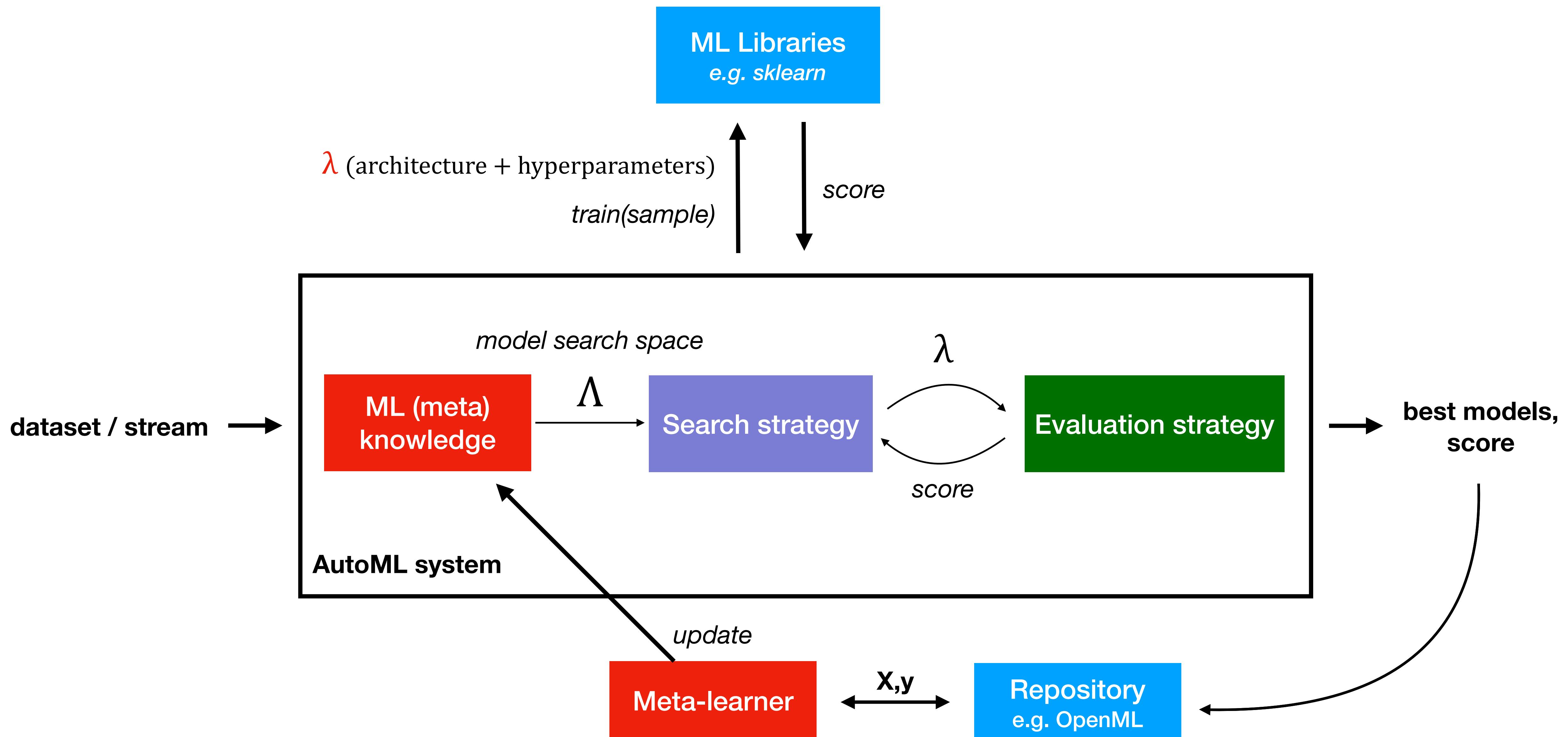
# Structure of AutoML systems



# Structure of AutoML systems

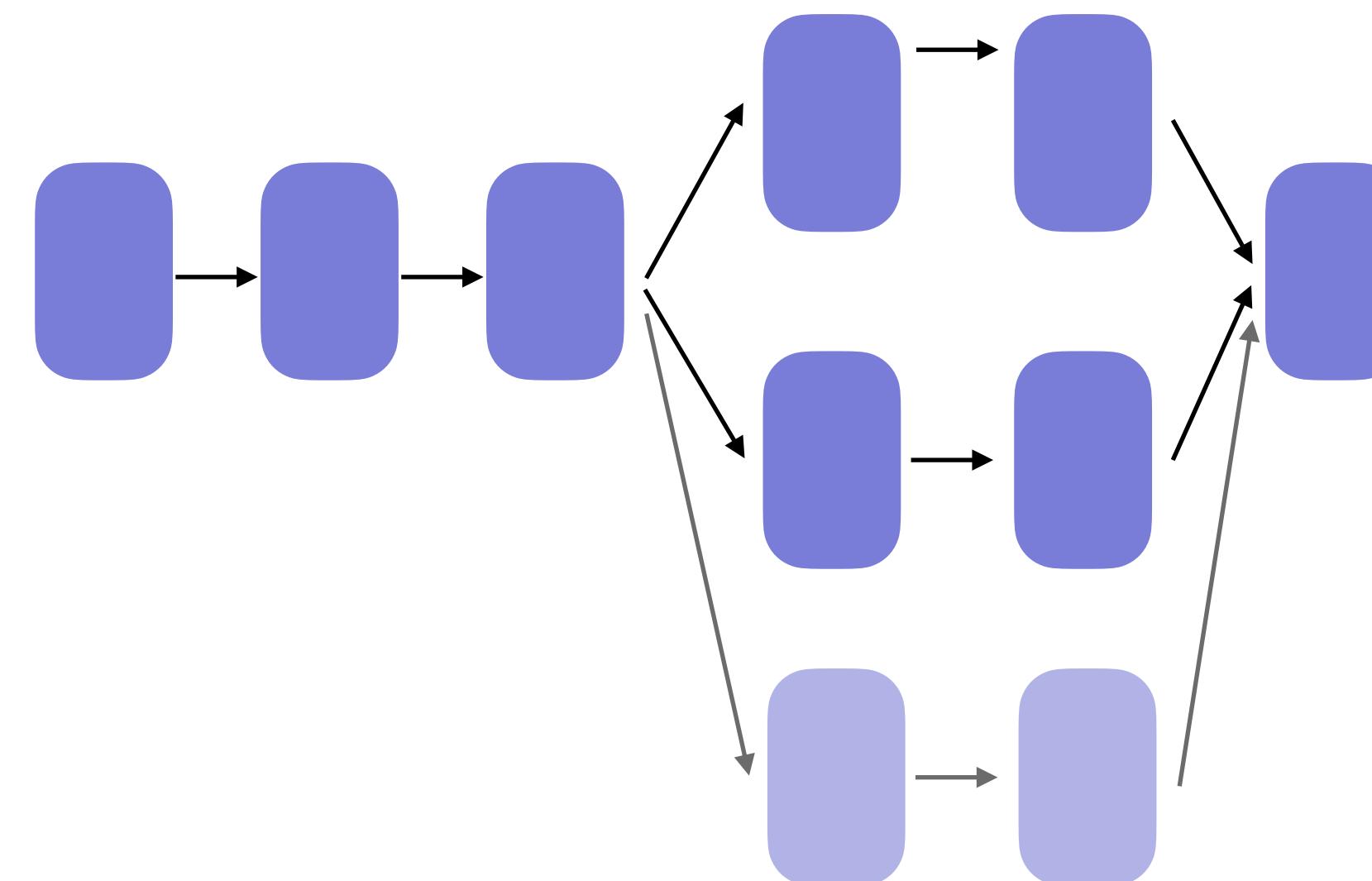


# Structure of learning AutoML systems



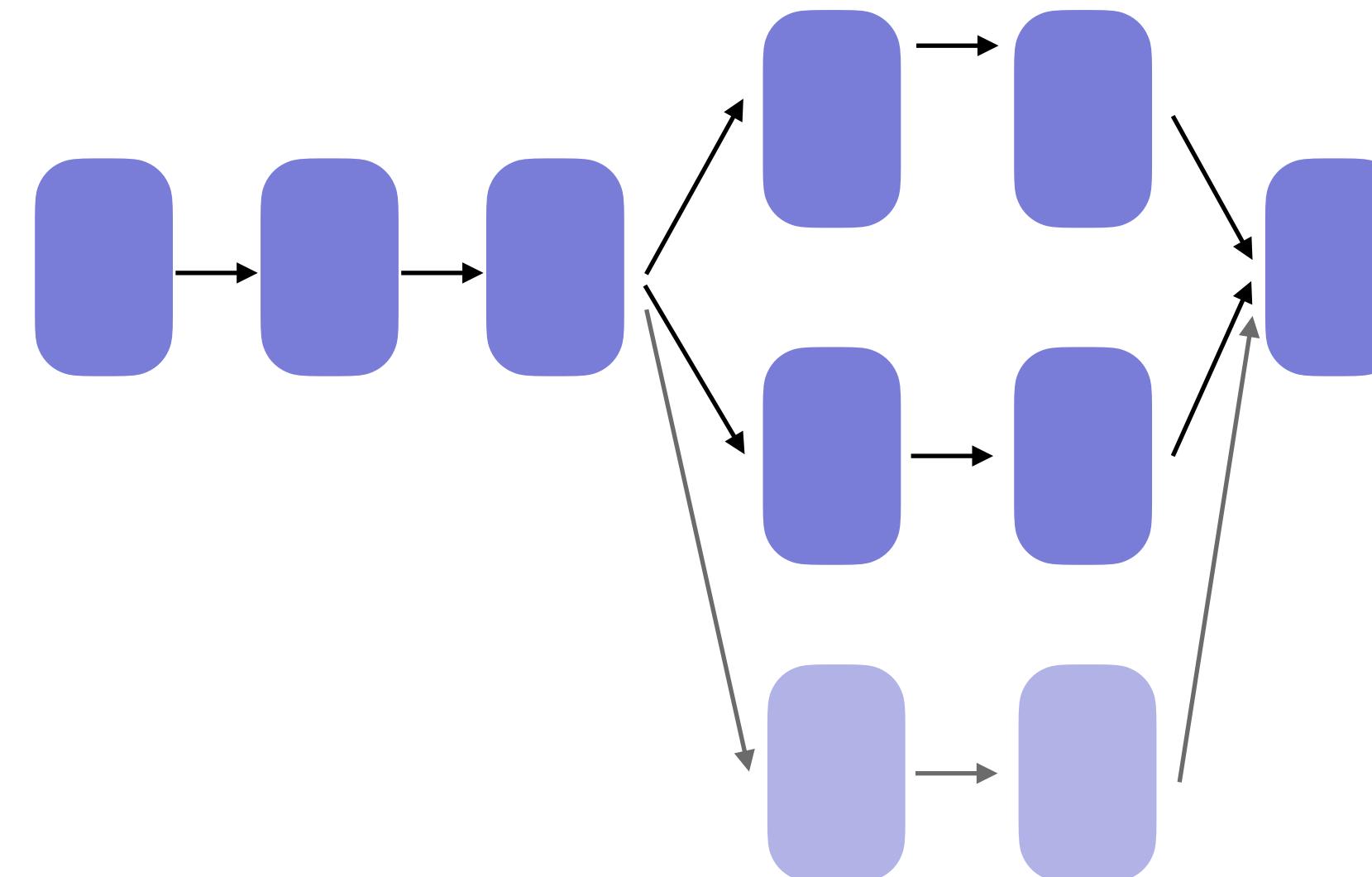
# AutoML: subproblems

- **Model search space  $\Lambda$ :** represent all pipelines or neural architectures
  - Pipeline operators, neural layers, interconnections,...
  - Defines a (complex) search space



# AutoML: subproblems

- **Model search space  $\Lambda$ :** represent all pipelines or neural architectures
  - Pipeline operators, neural layers, interconnections,...
  - Defines a (complex) search space



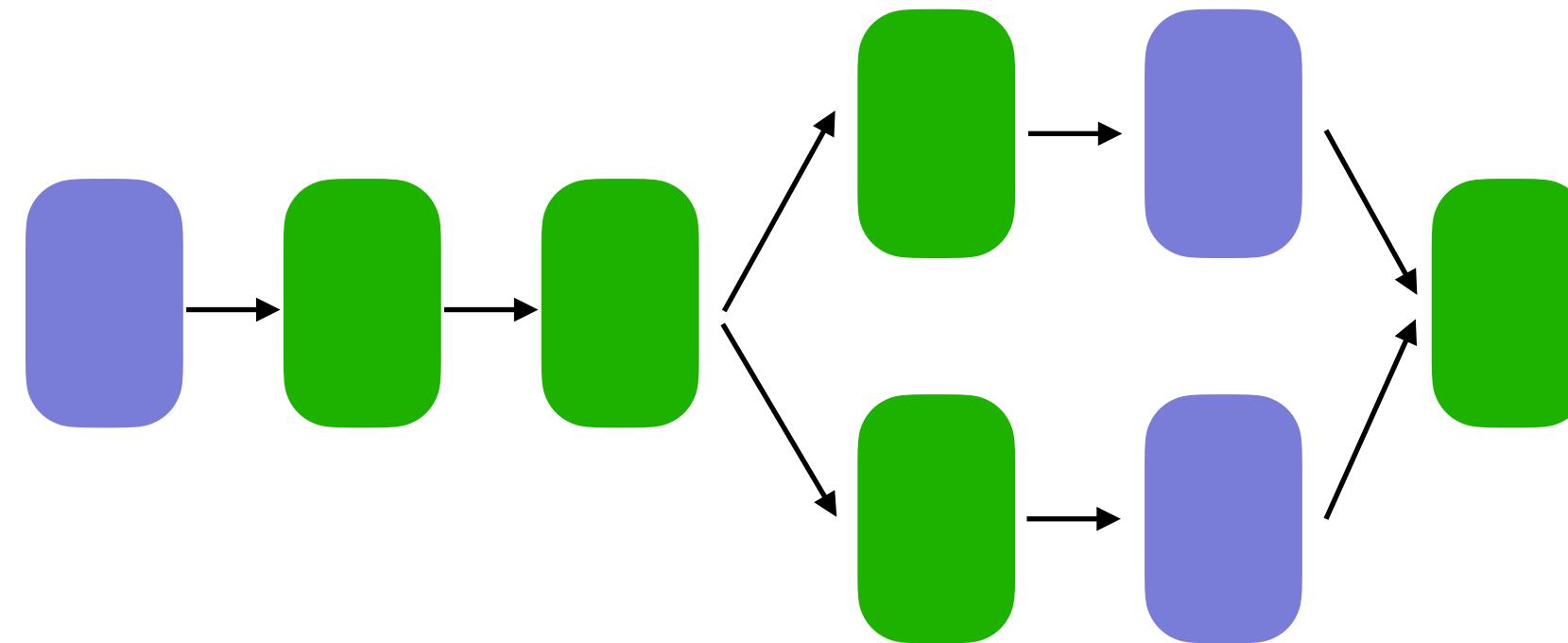
```
make_pipeline(  
    OneHotEncoder(),  
    Imputer(),  
    StandardScaler(),  
    SVC())
```



```
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

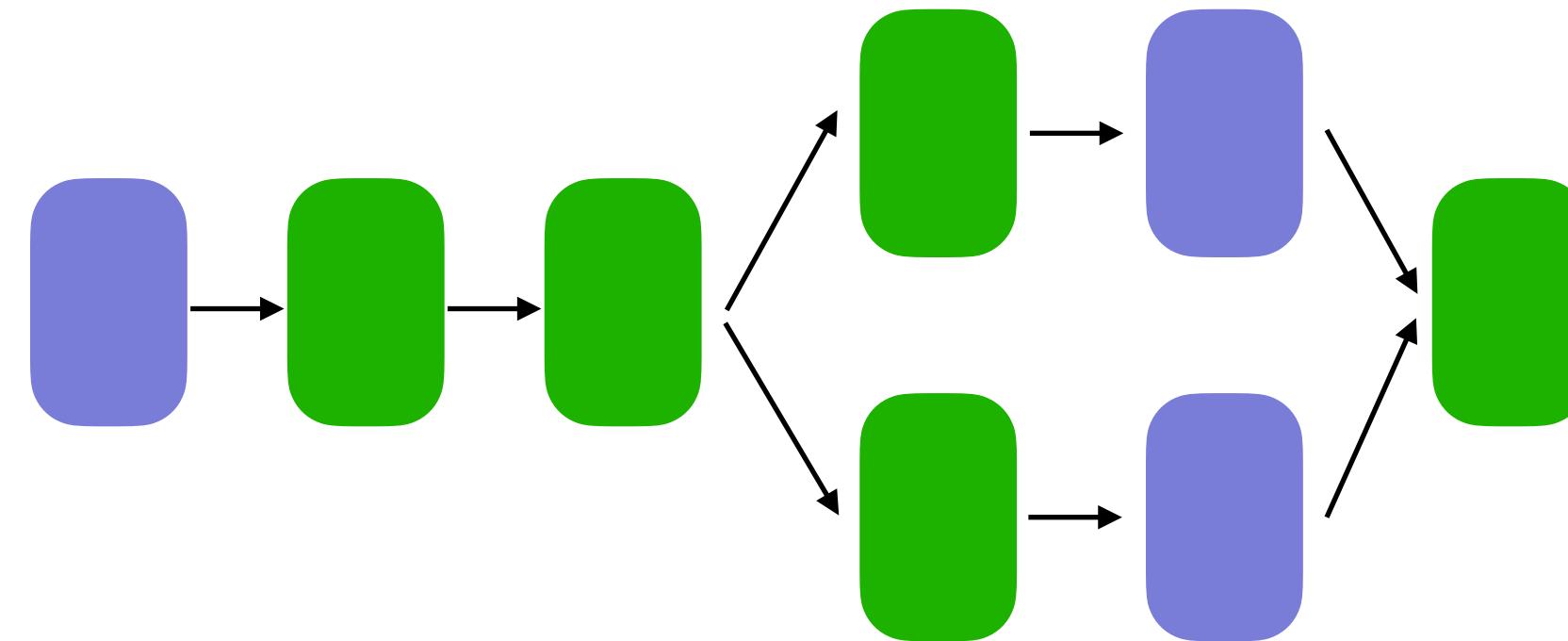
- **Model search space:** *represent all possible architectures*
- **Optimization:**
  - What is the best architecture? Which hyperparameters are important?
  - How to optimize them? What is the (multi-) objective function?



```
hyper_space = {'SVC__C': expon(scale=100),  
              'SVC__gamma': expon(scale=.1)}  
RandomizedSearchCV(pipe, param_distributions=hyper_space, n_iter=200)
```

# AutoML: subproblems

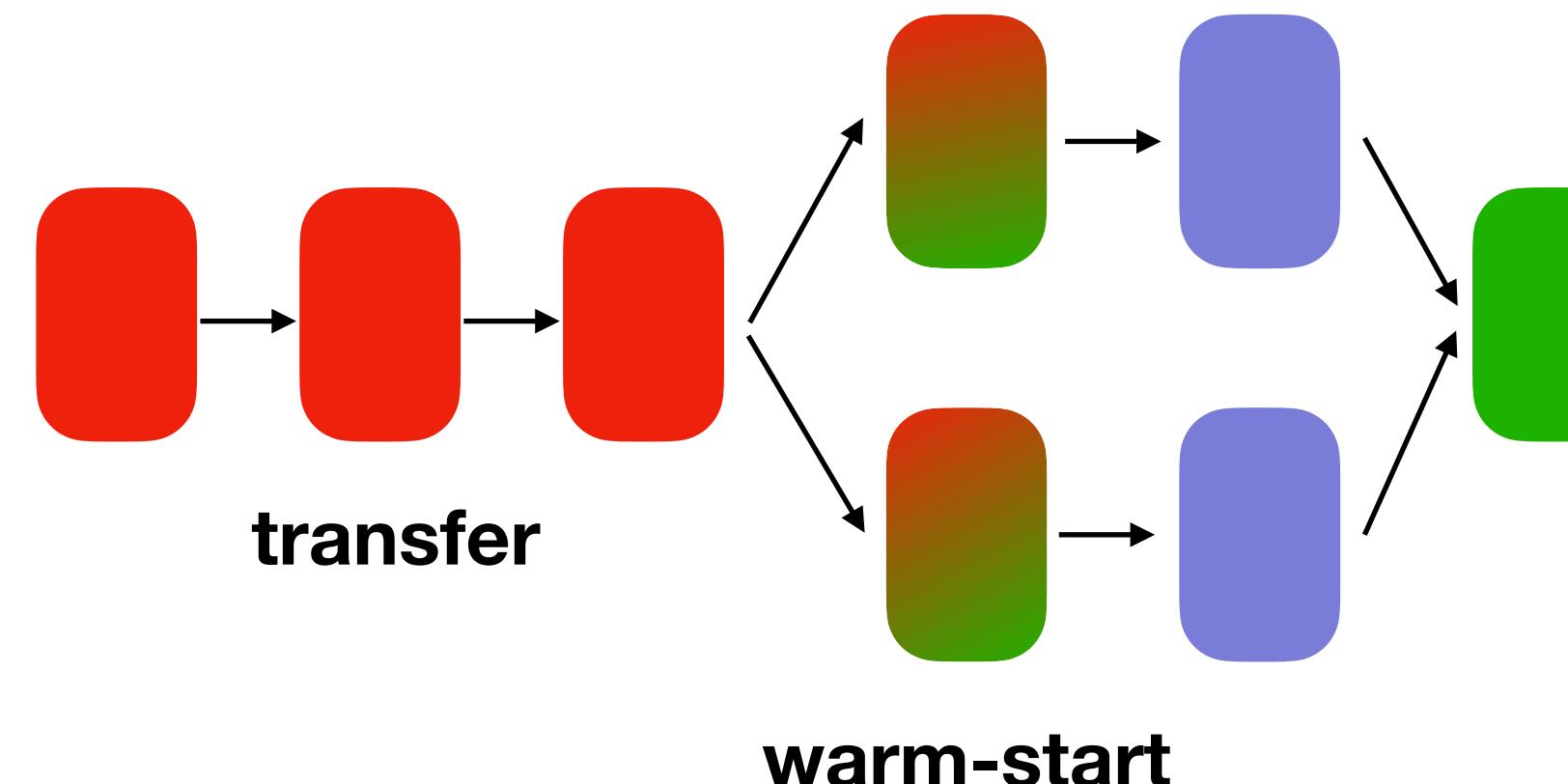
- **Model search space:** *represent all possible architectures*
- **Optimization:**
  - What is the best architecture? Which hyperparameters are important?
  - How to optimize them? What is the (multi-) objective function?



```
hyper_space = {'SVC__C': expon(scale=100),
               'SVC__gamma': expon(scale=.1)}
RandomizedSearchCV(pipe, param_distributions=hyper_space, n_iter=200)
```

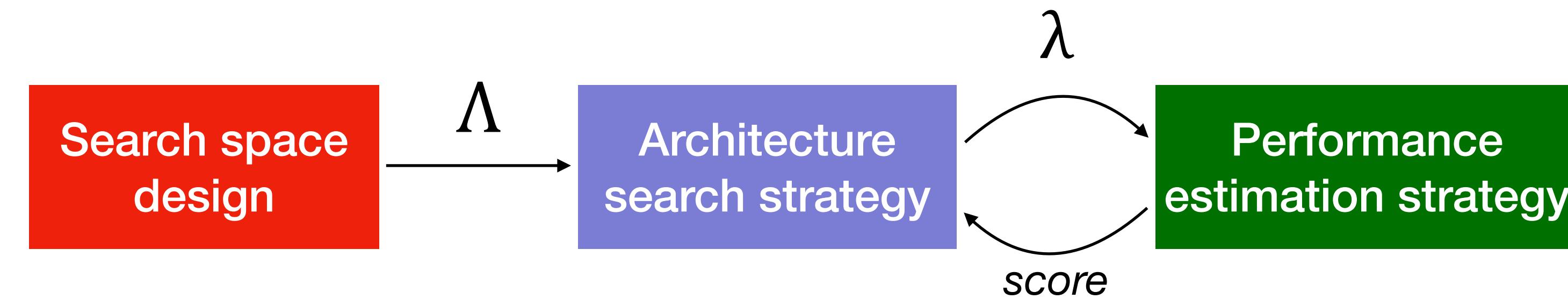
# AutoML: subproblems

- **Architecture search space**: *represent all possible architectures*
- **Optimization**: *optimize architecture and hyperparameters*
- **Meta-learning**: how can we transfer *experience* from previous tasks?
  - Don't start from scratch (search space is too large)
  - Transfer learning: reuse good architectures/configurations/weights
  - Warm starting: start from promising architectures/configurations/initializations



# AutoML: subproblems

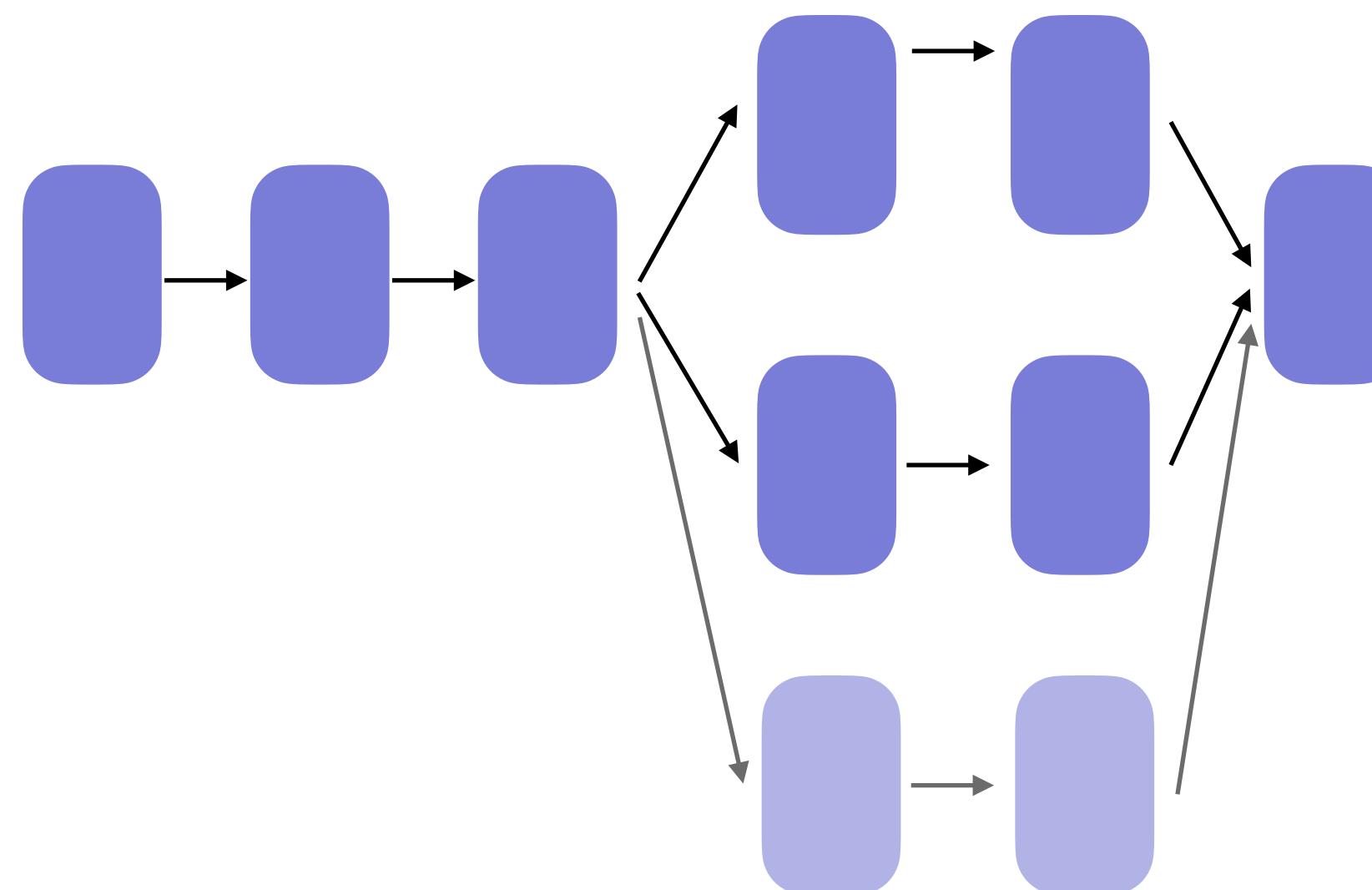
- **Architecture search**: *represent all possible architectures*
- **Optimization**: *optimize architecture and hyperparameters*
- **Meta-learning**: how can we transfer experience from previous tasks?



Many combinations are possible!

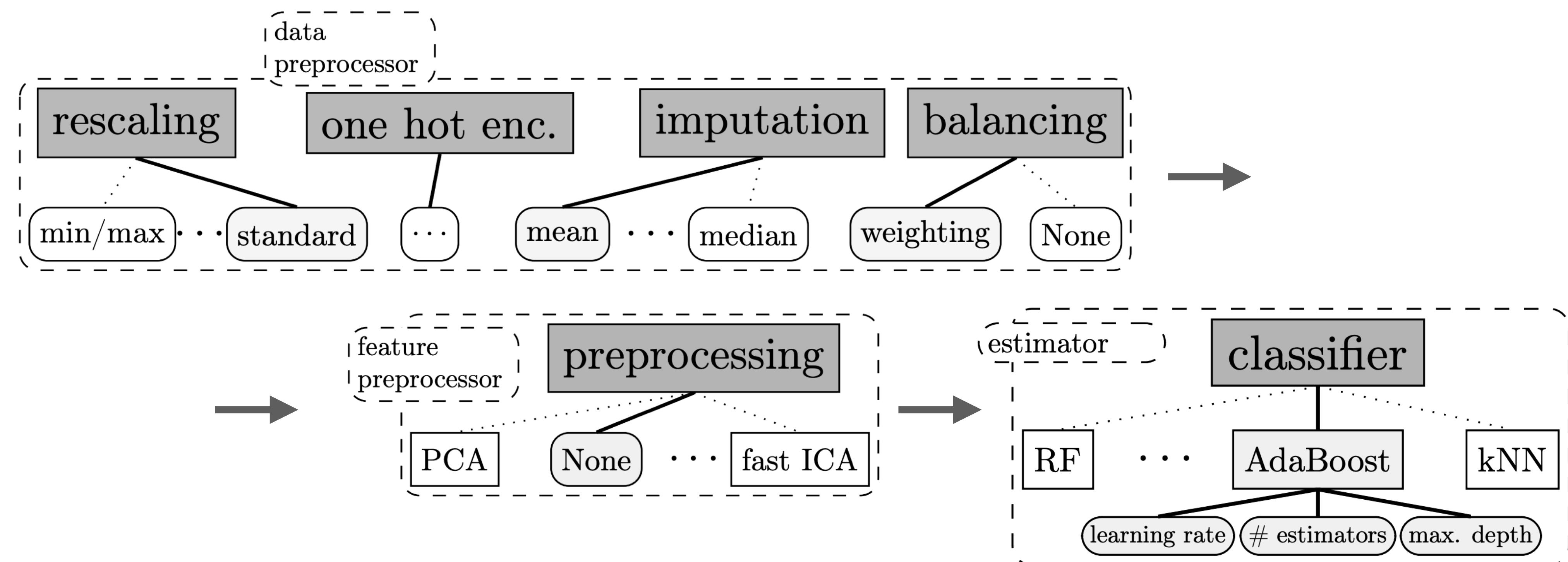
They can be done *consecutively*, *simultaneously* or *interleaved*

# Architecture search space



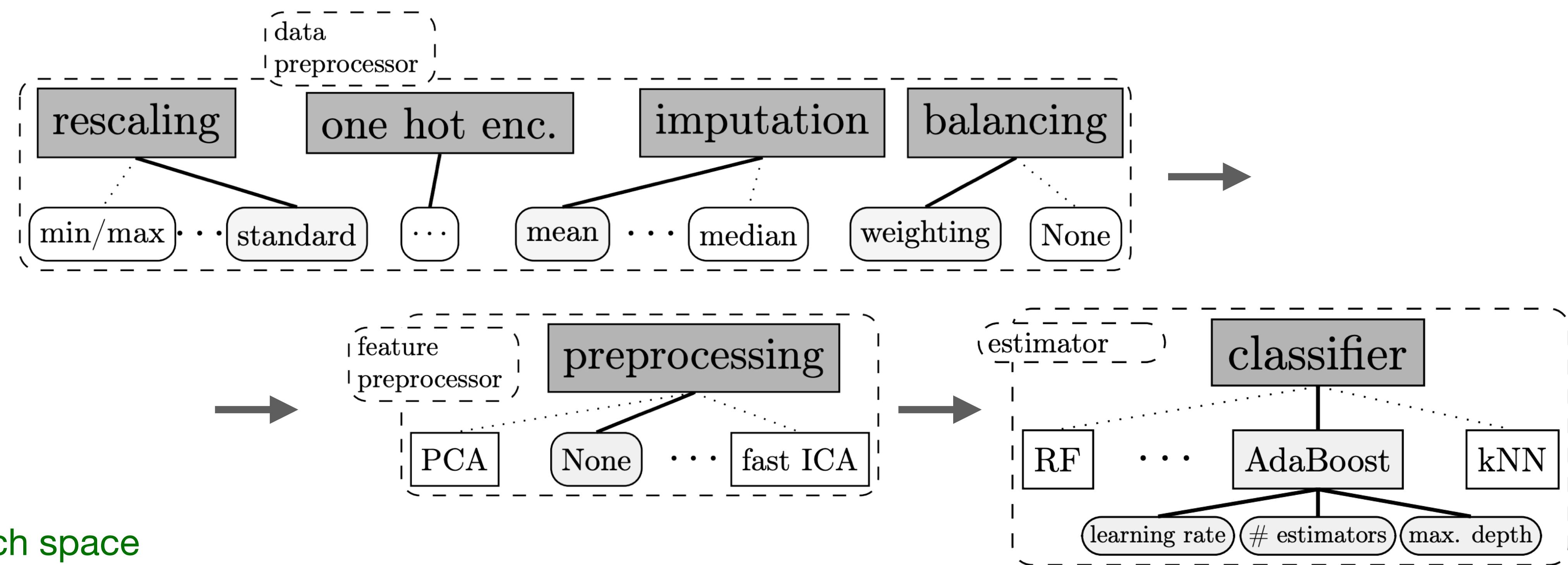
# Parameterized architectures

- Manual bias: most successful pipelines have a similar structure



# Parameterized architectures

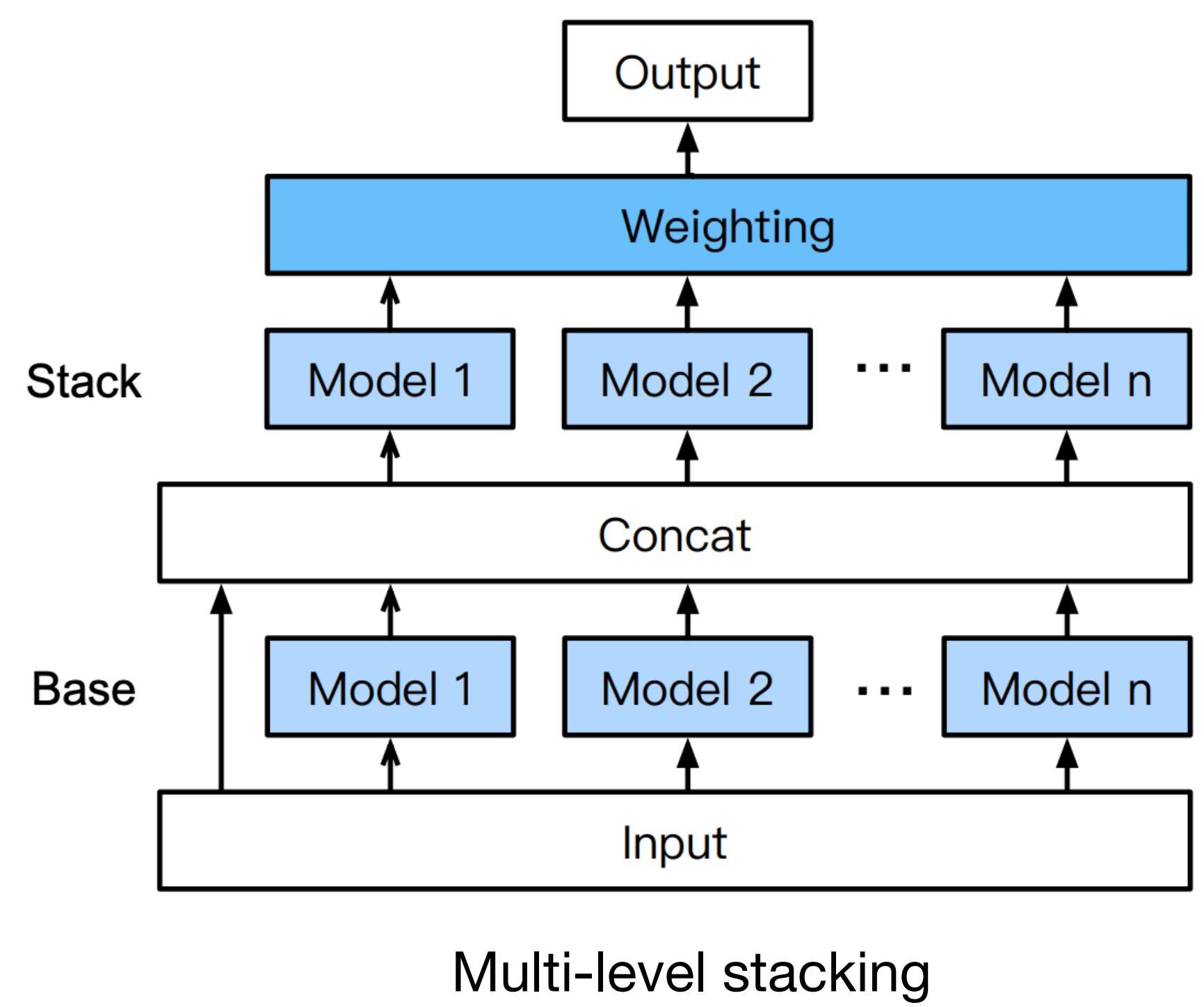
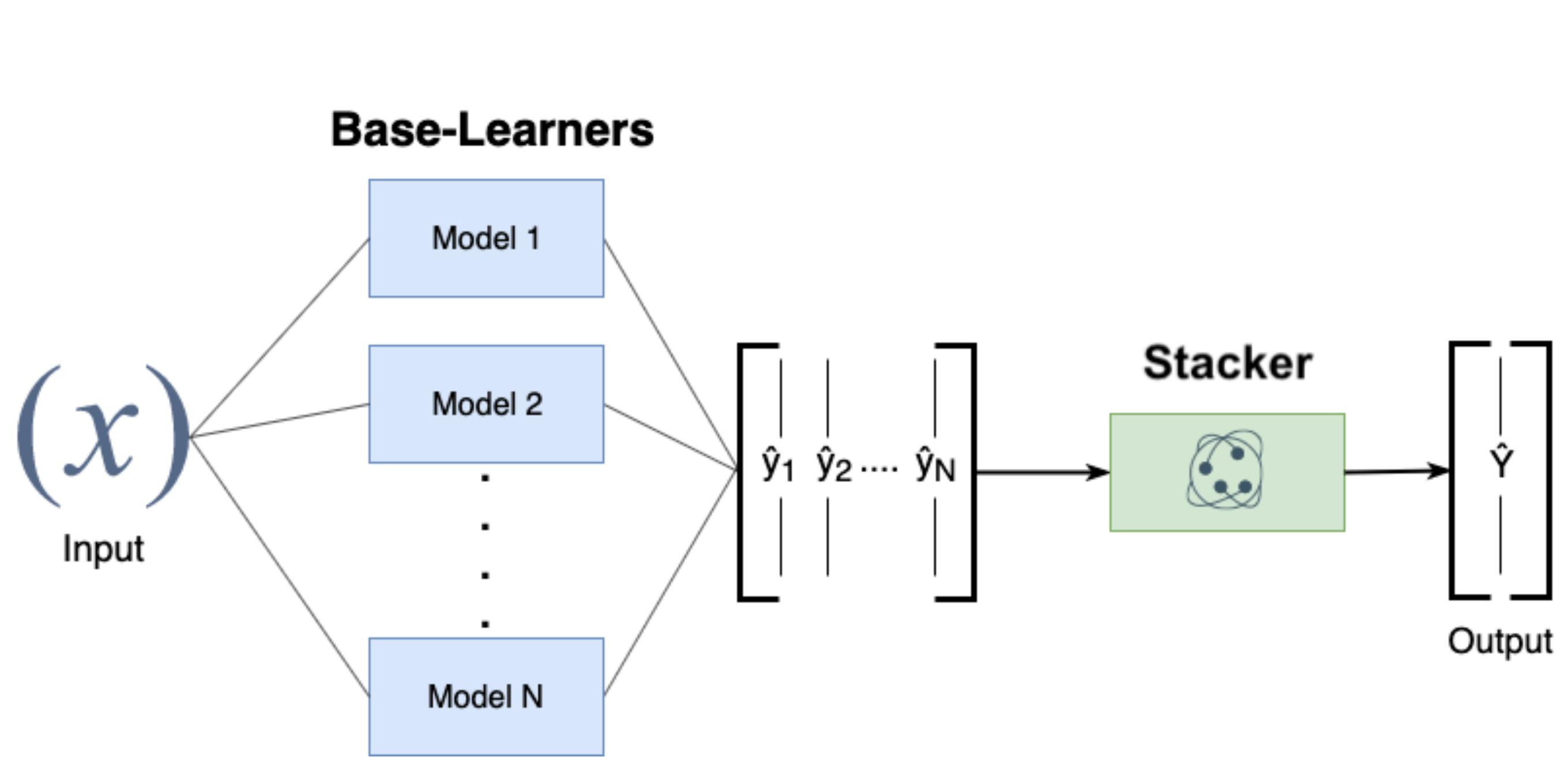
- Manual bias: most successful pipelines have a similar structure
- Fix architecture, encode ***all*** choices as extra hyperparameters
  - *Architecture search becomes hyperparameter optimization*



**- you can't learn entirely new architectures**

# Ensembling

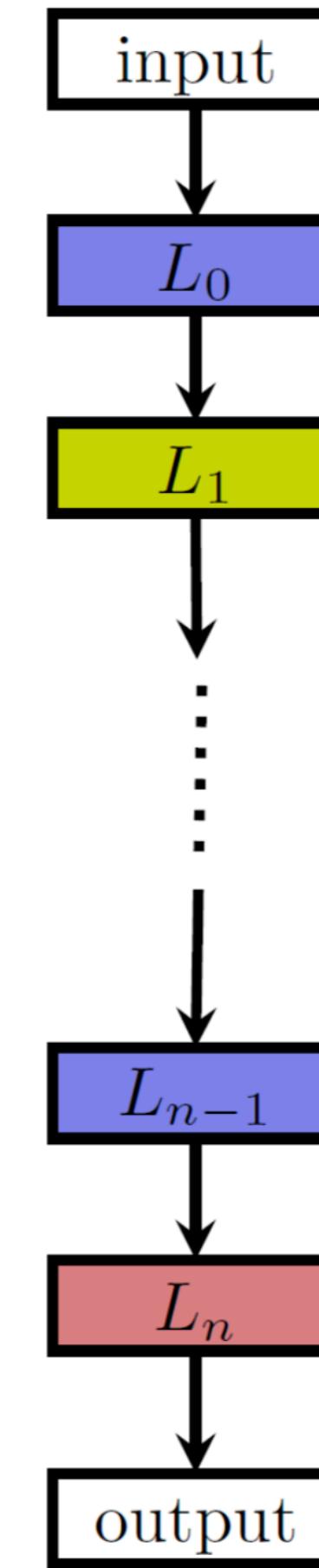
- Build ensembles of multiple pipelines to avoid overfitting
  - RandomForests (Auto-sklearn, GAMA,...)
  - Stacking (AutoGluon-Tabular, H2O AutoML,...)



# Parameterized neural architectures

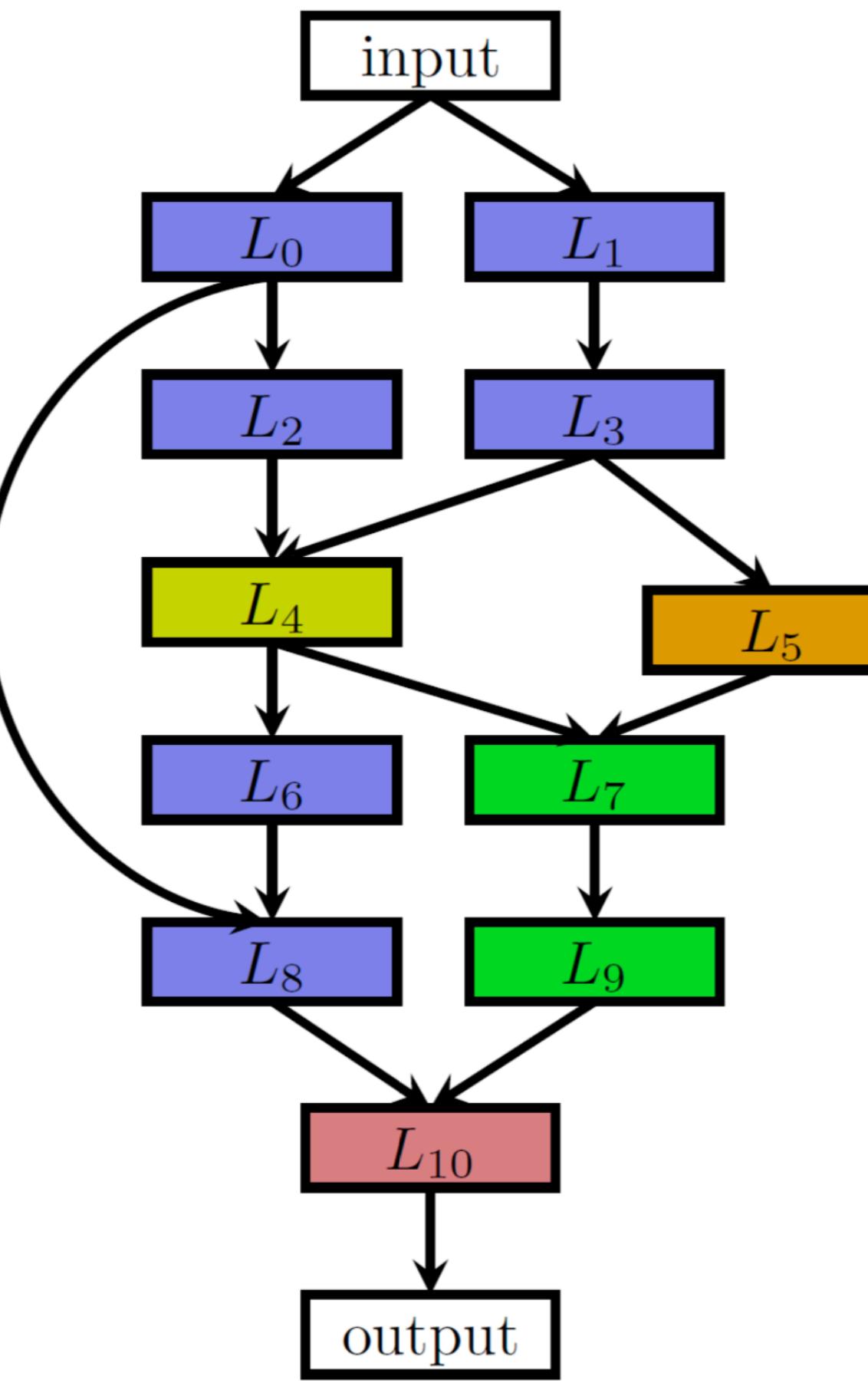
## Parameterized Sequential

- Choose:
- number of layers
  - type of layers
    - dense
    - convolutional
    - max-pooling
    - ...
  - hyperparameters of layers
- + easier to search  
- sometimes too simple



## Parameterized Graph

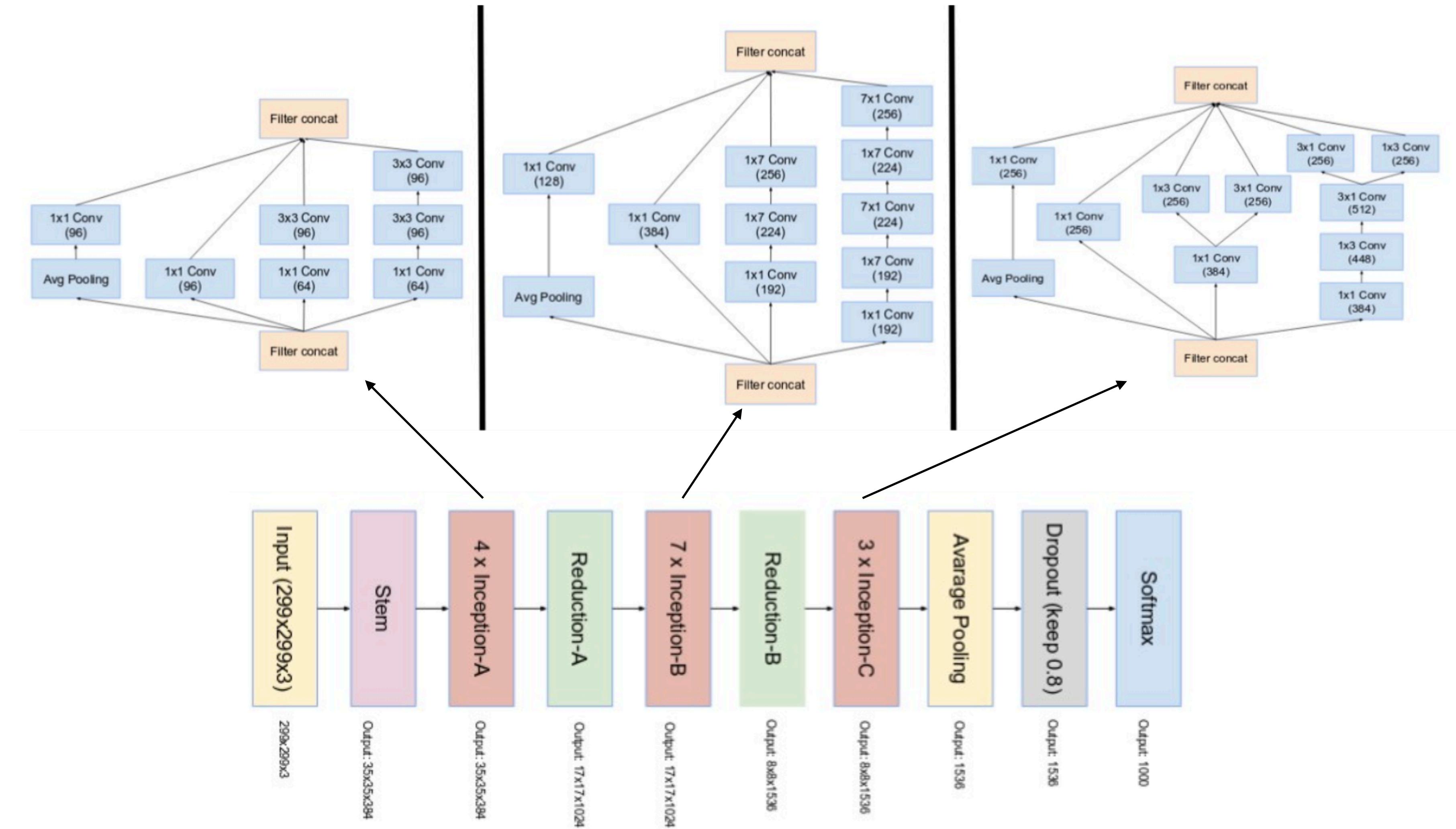
- Choose:
- branching
  - joins
  - skip connections
  - types of layers
  - hyperparameters of layers
- + more flexible  
- much harder to search



# Cell search spaces

Manual bias: successful deep networks have repeated motifs (cells)

e.g. Inception v4:

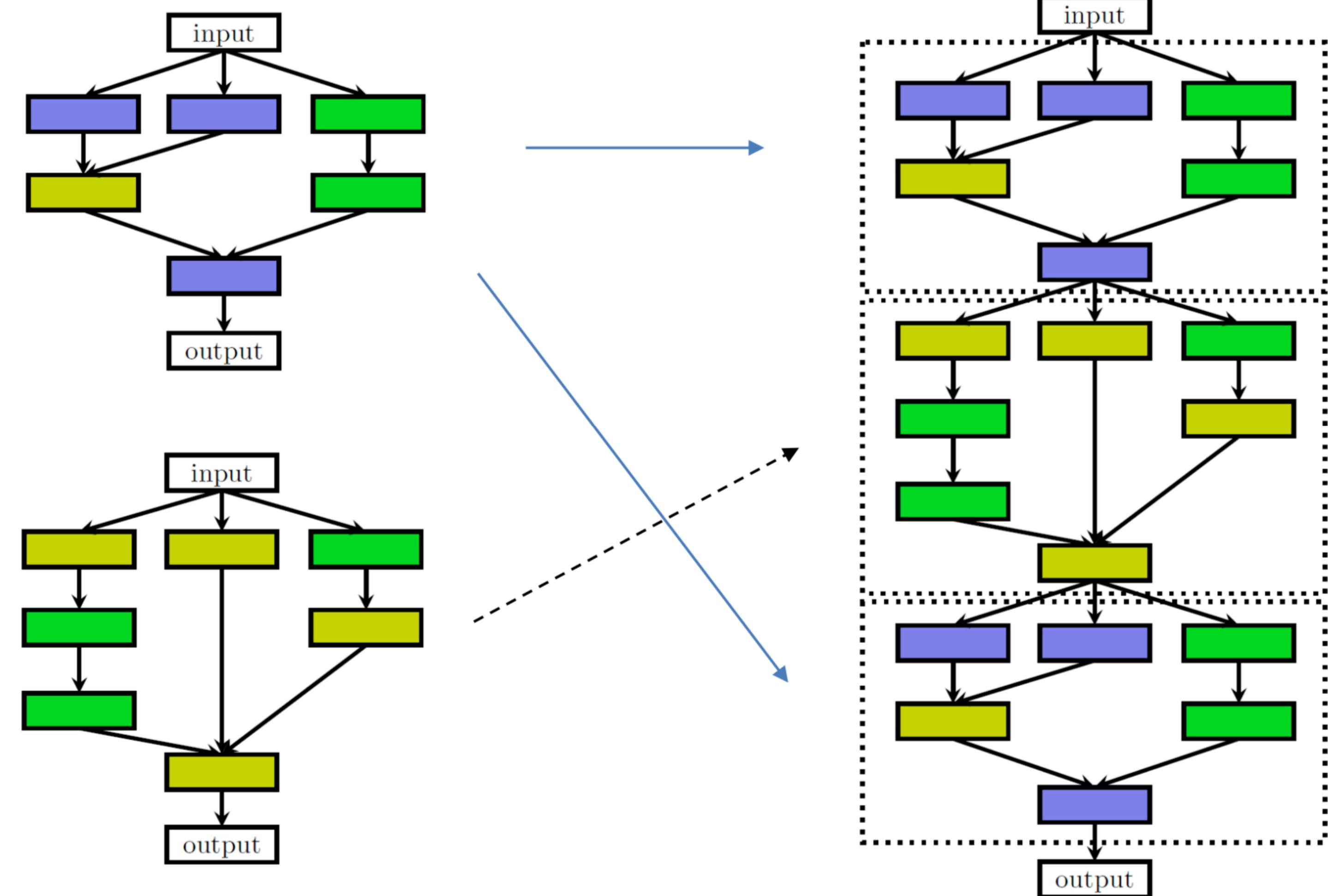


# Cell search spaces

**Compositionality:** learn hierarchical building blocks to simplify the task

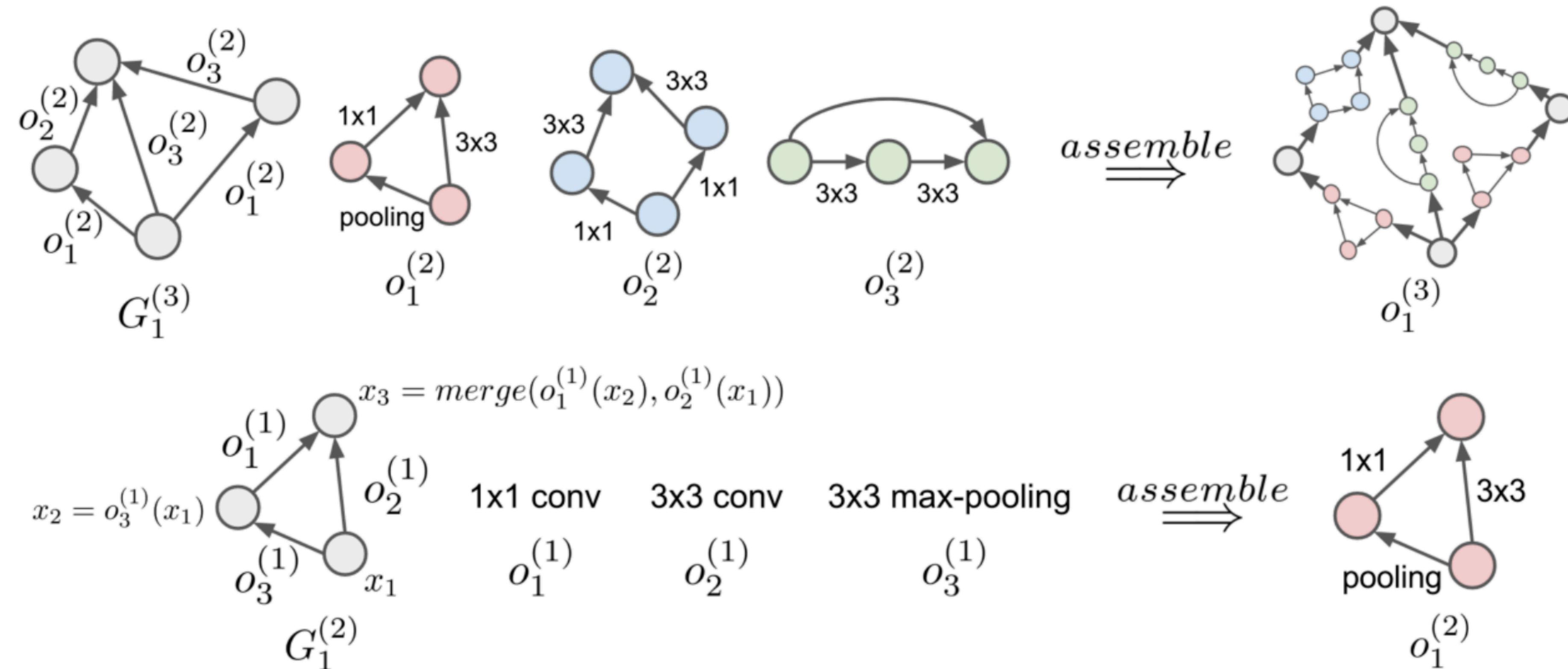
## Cell search space

- learn parameterized building blocks (*cells*)
  - stack cells together in macro-architecture
- + smaller search space  
+ cells can be learned on a small dataset & transferred  
- strong domain priors, doesn't generalize well



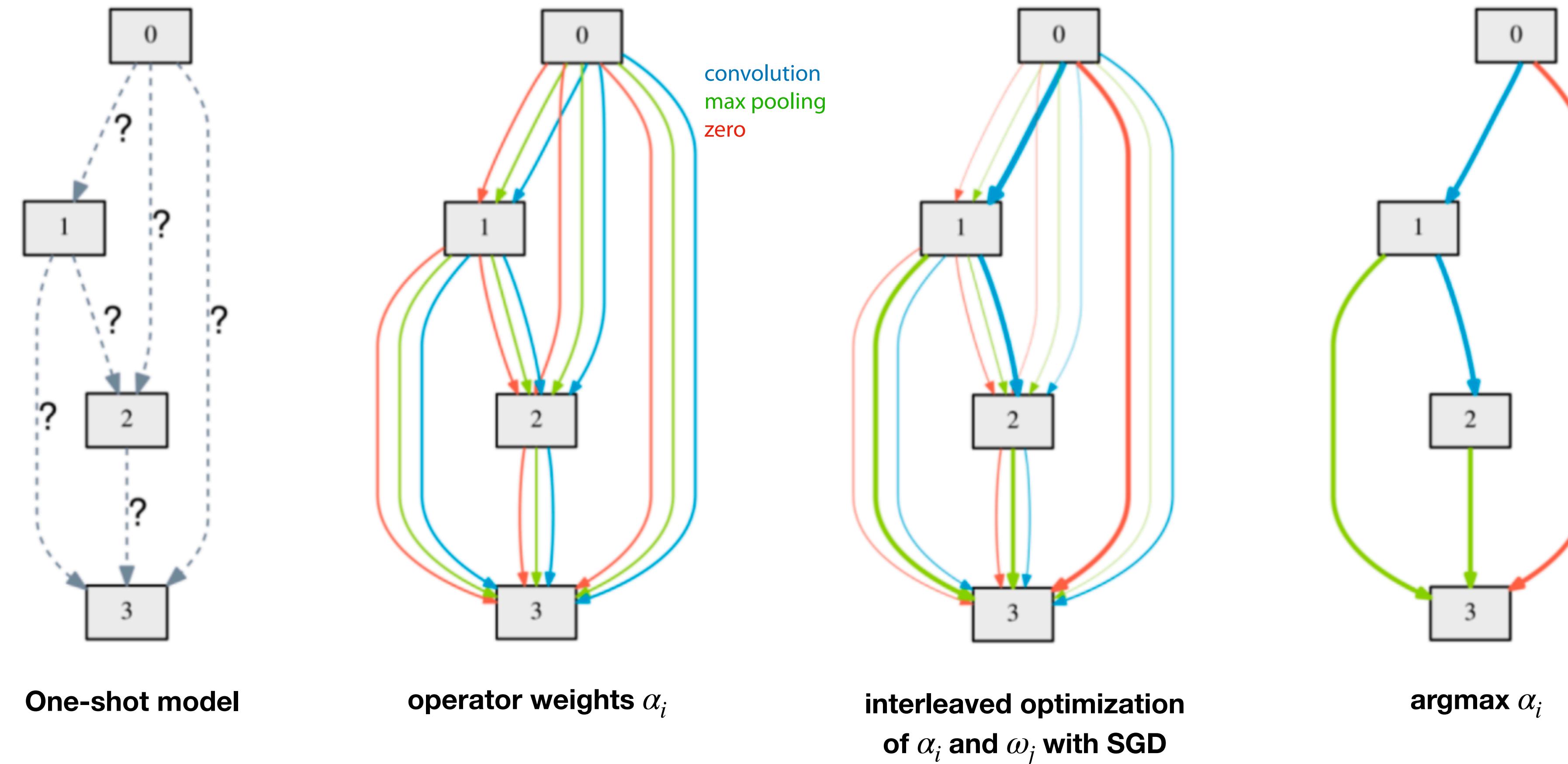
# Hierarchical Search Spaces

- Define a number of primitive operations (e.g. convolution, max-pooling,...)
- Build small motifs of primitives ( $o_1, o_2, o_3, \dots$ )
- Choose a graph structure  $G_i$ , replace edges with motifs, repeat

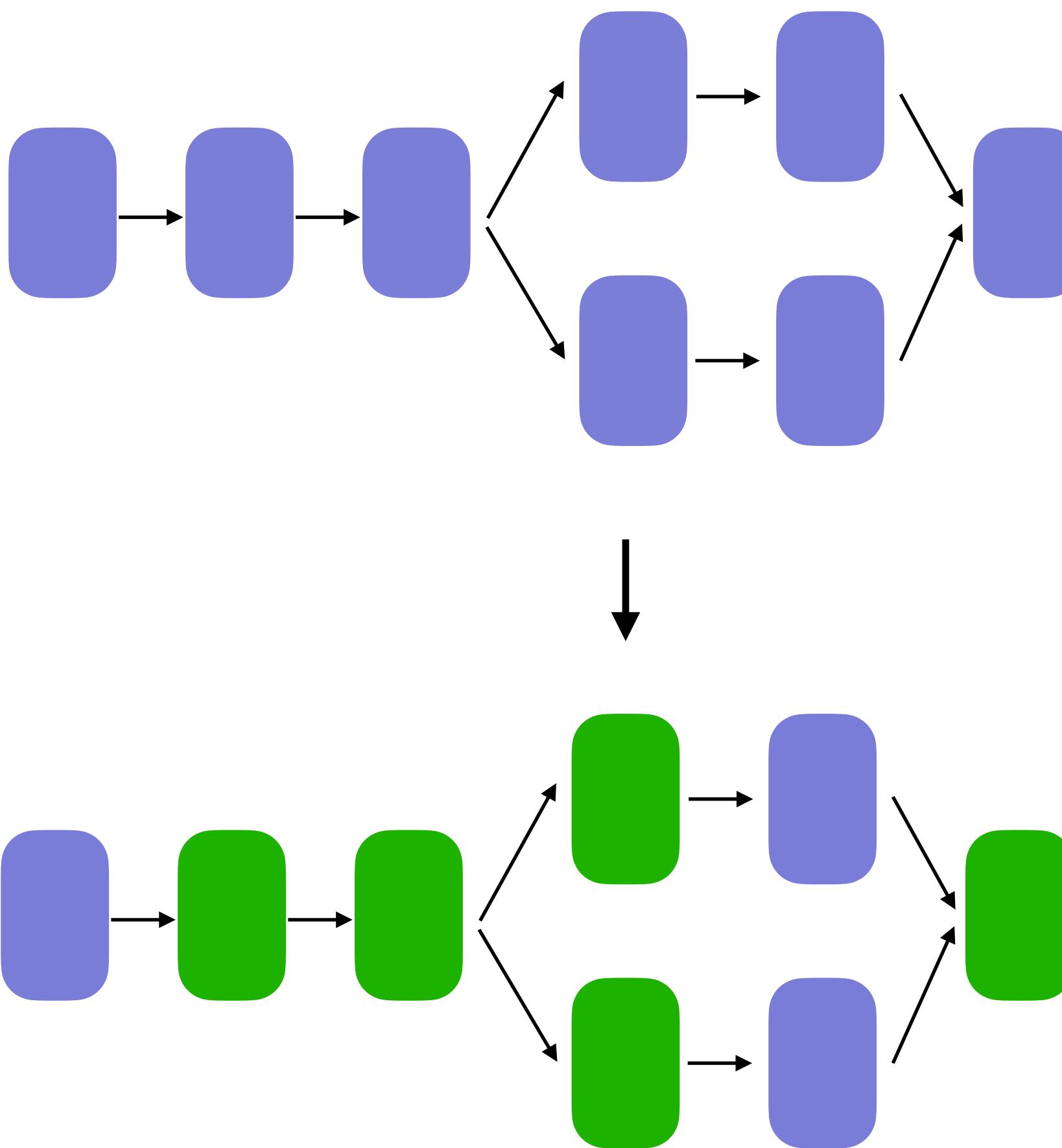


# Differentiable Search Spaces (DARTS)

- Fixed (one-shot) structure, each edge can be any primitive operation
- Give all operators a weight  $\alpha_i$  -> *pure numeric/differentiable search space!*
- Optimize  $\alpha_i$  and model weights  $\omega_j$  using bilevel programming

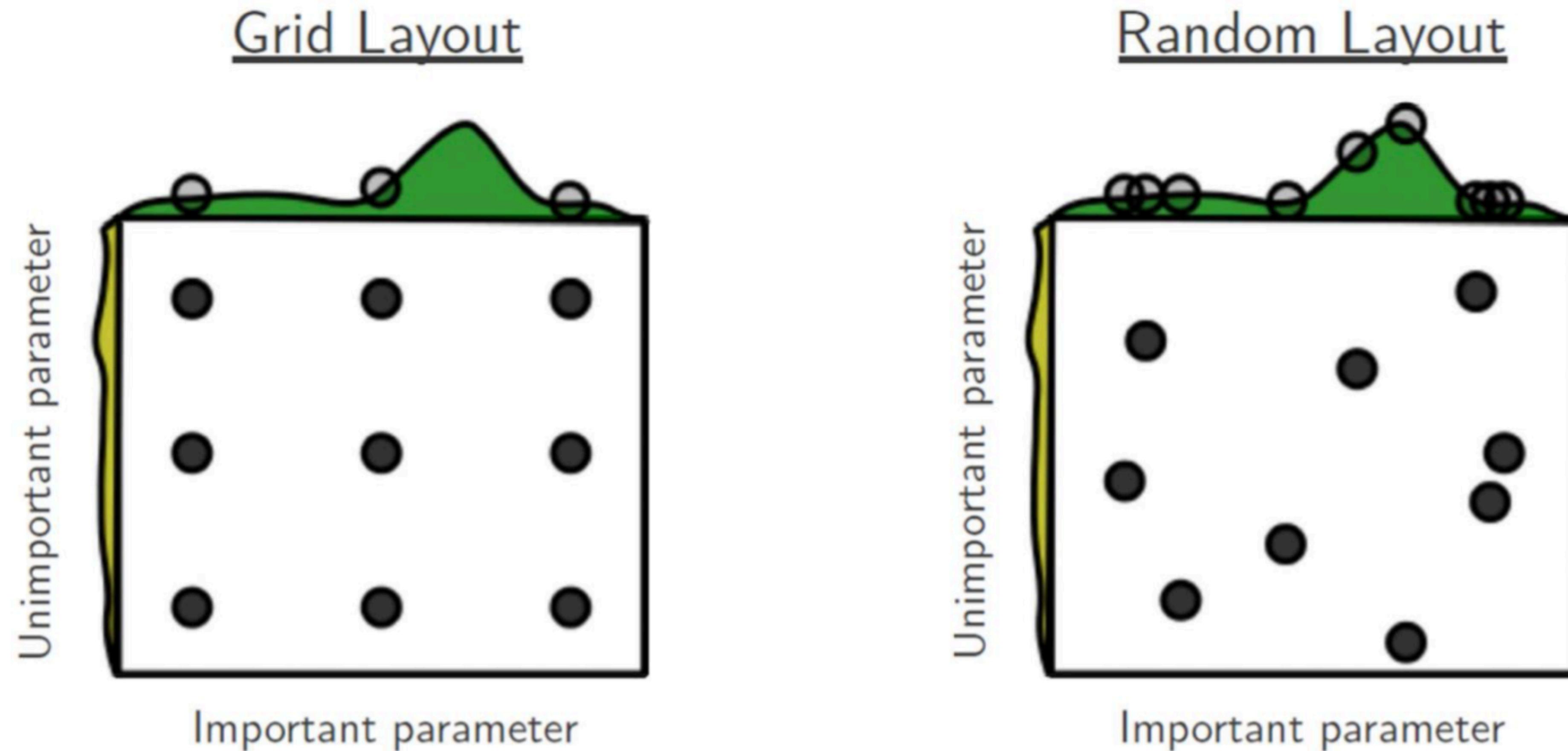


# Optimization



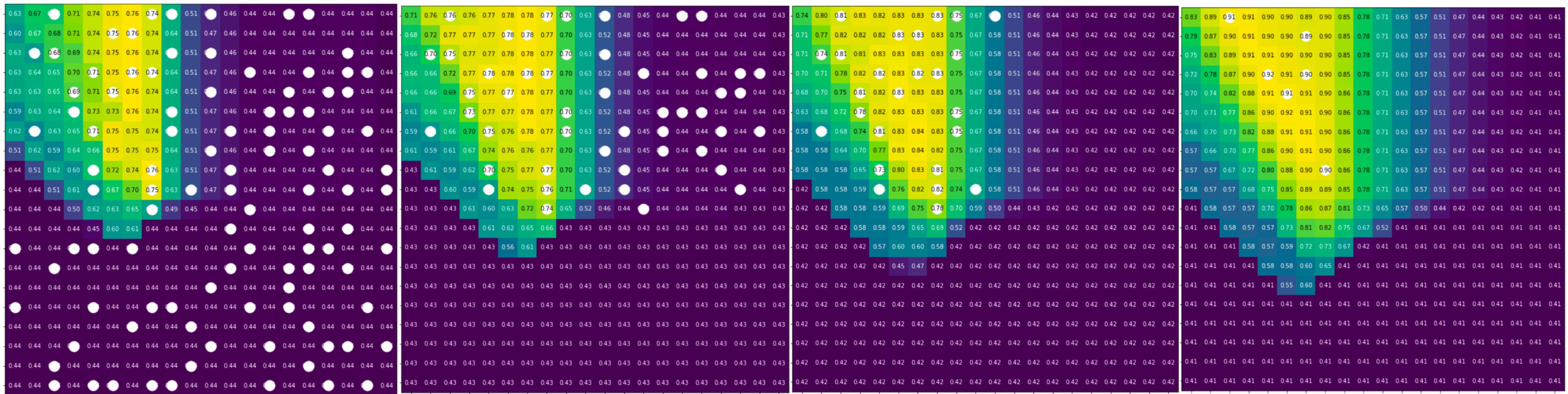
# Random search

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



# Successive Halving

- Train on small data 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, repeat



1/16

1/8

1/4

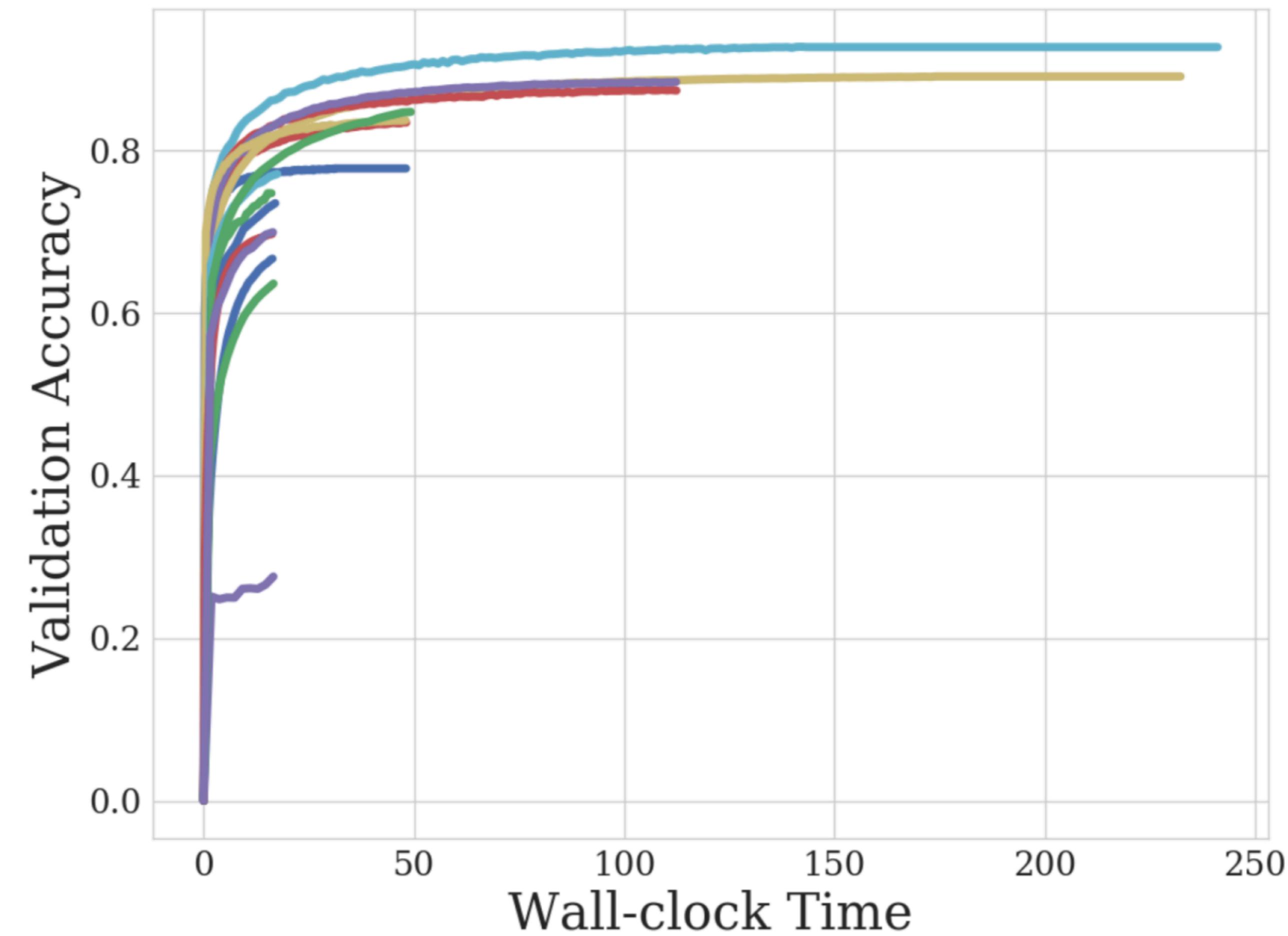
1/2

sample size

# Hyperband

Successive halving risks killing good models too early

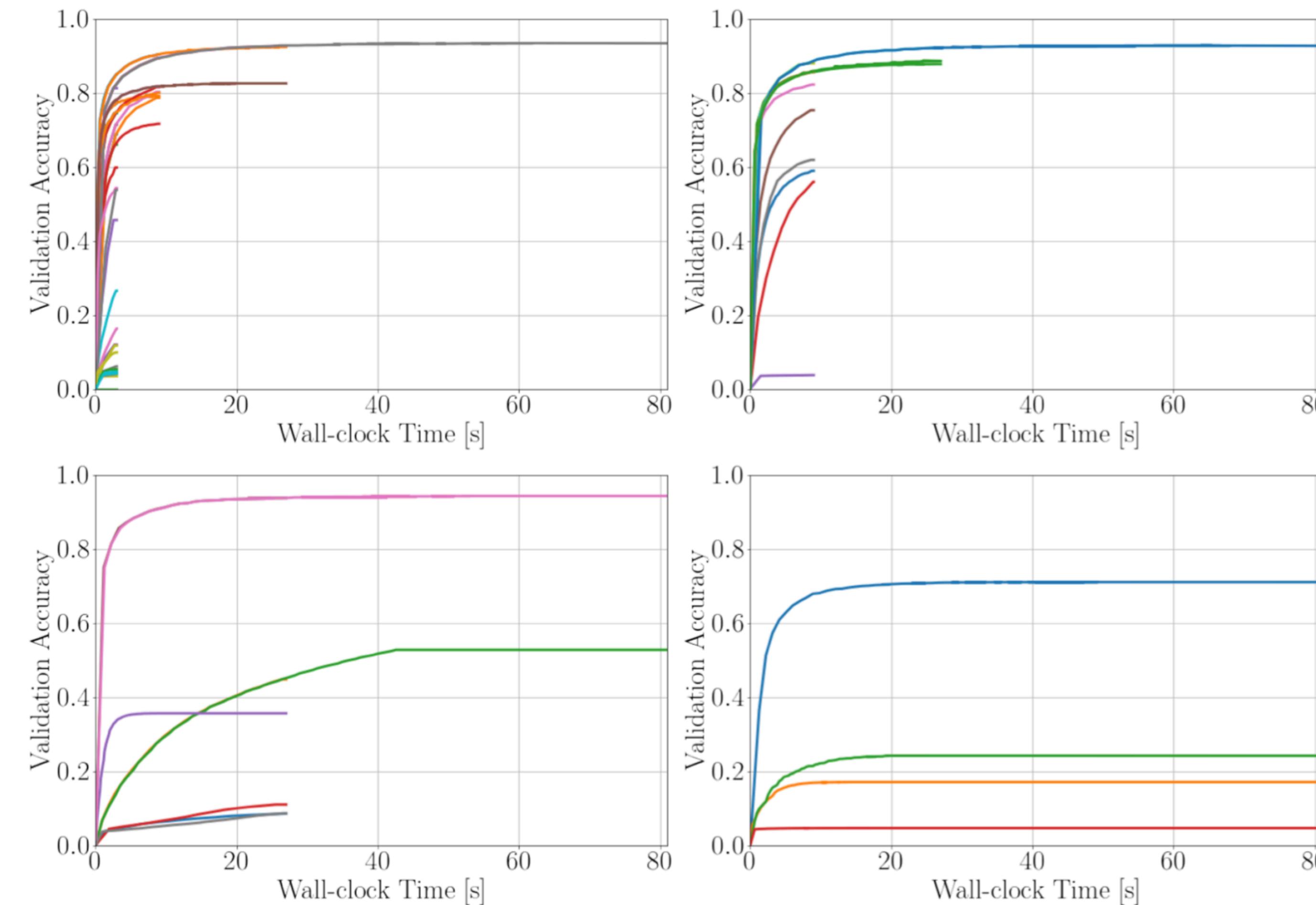
- Repeat in multiple, decreasingly aggressive iterations (brackets)
- Strong anytime performance, easy to implement, scalable, parallelizable



# Hyperband

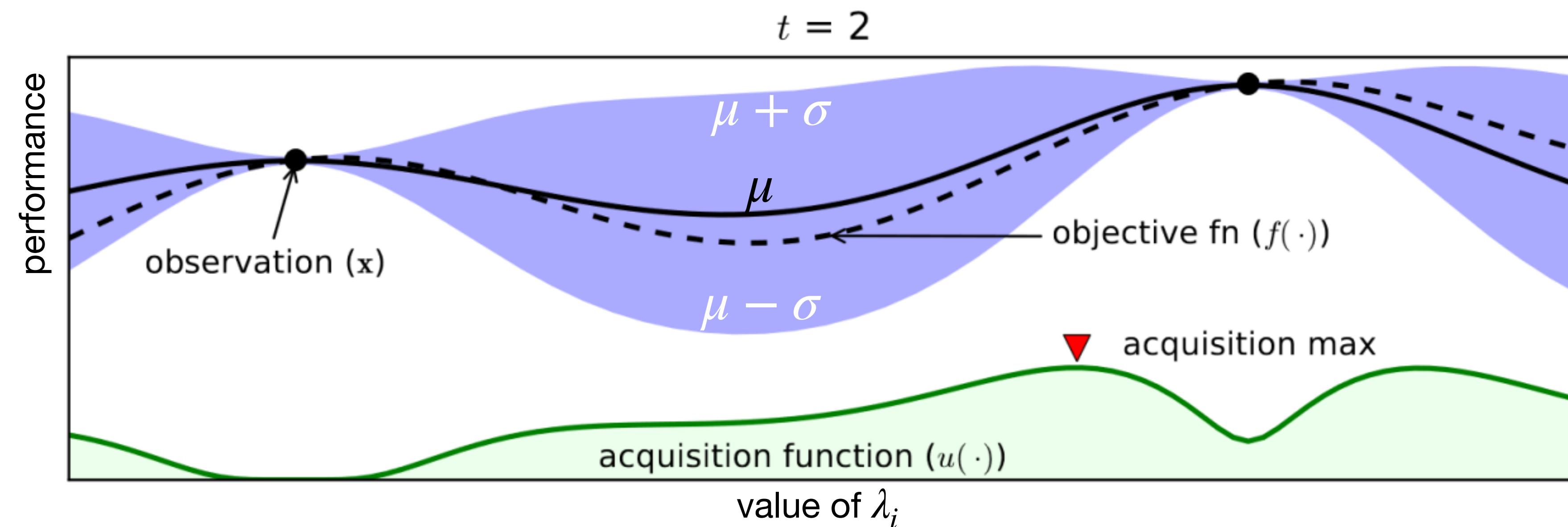
Successive halving risks killing good models too early

- Repeat in multiple, decreasingly aggressive iterations (brackets)
- Strong anytime performance, easy to implement, scalable, parallelizable



# Bayesian Optimization

- Start with a few (random) hyperparameter configurations
- Fit a **surrogate model** to predict other configurations
- Probabilistic regression (e.g. Gaussian Processes): 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

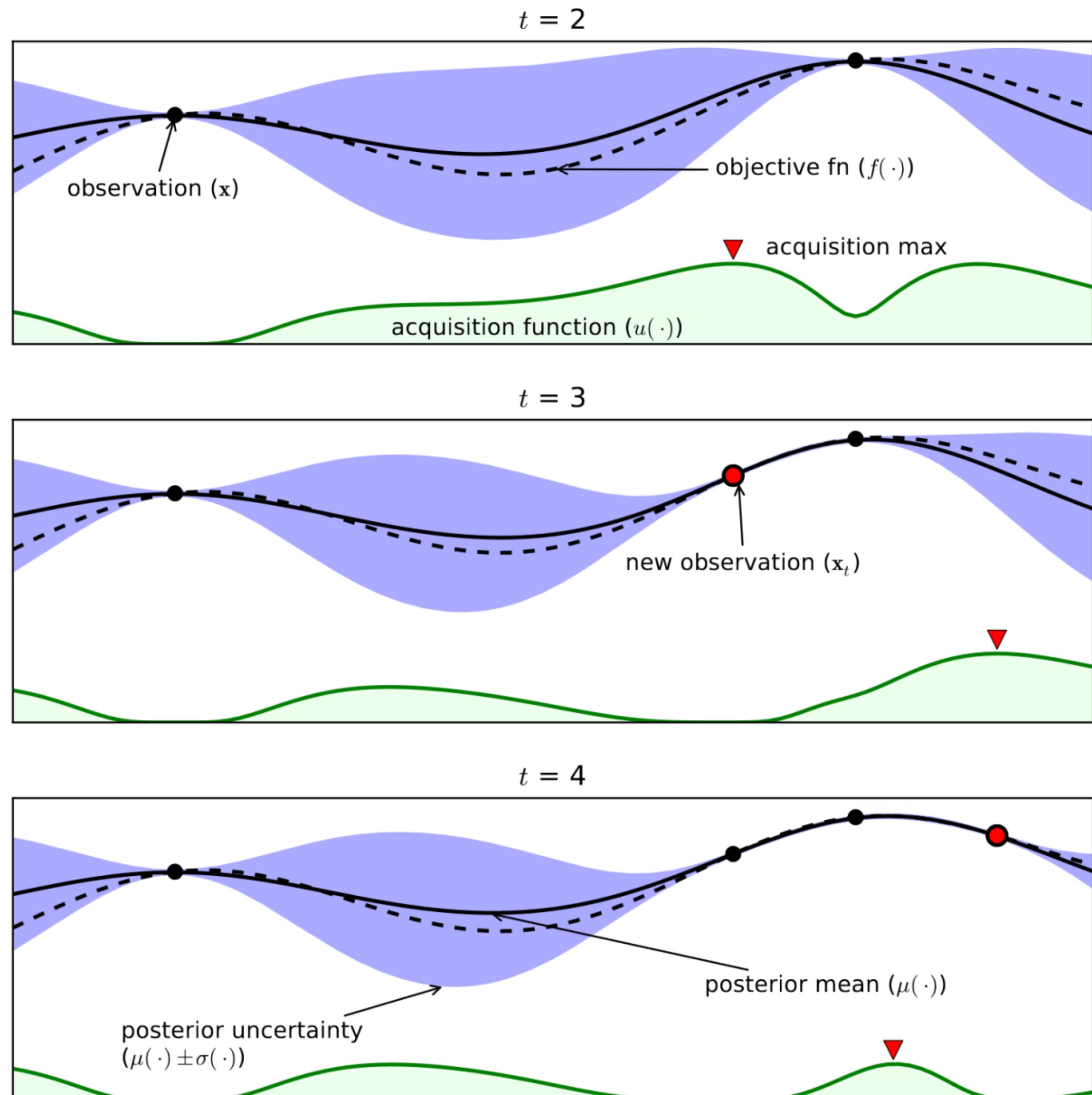


# Bayesian Optimization

- Repeat until some stopping criterion:
  - Fixed budget
  - Convergence
  - EI threshold
- Theoretical guarantees

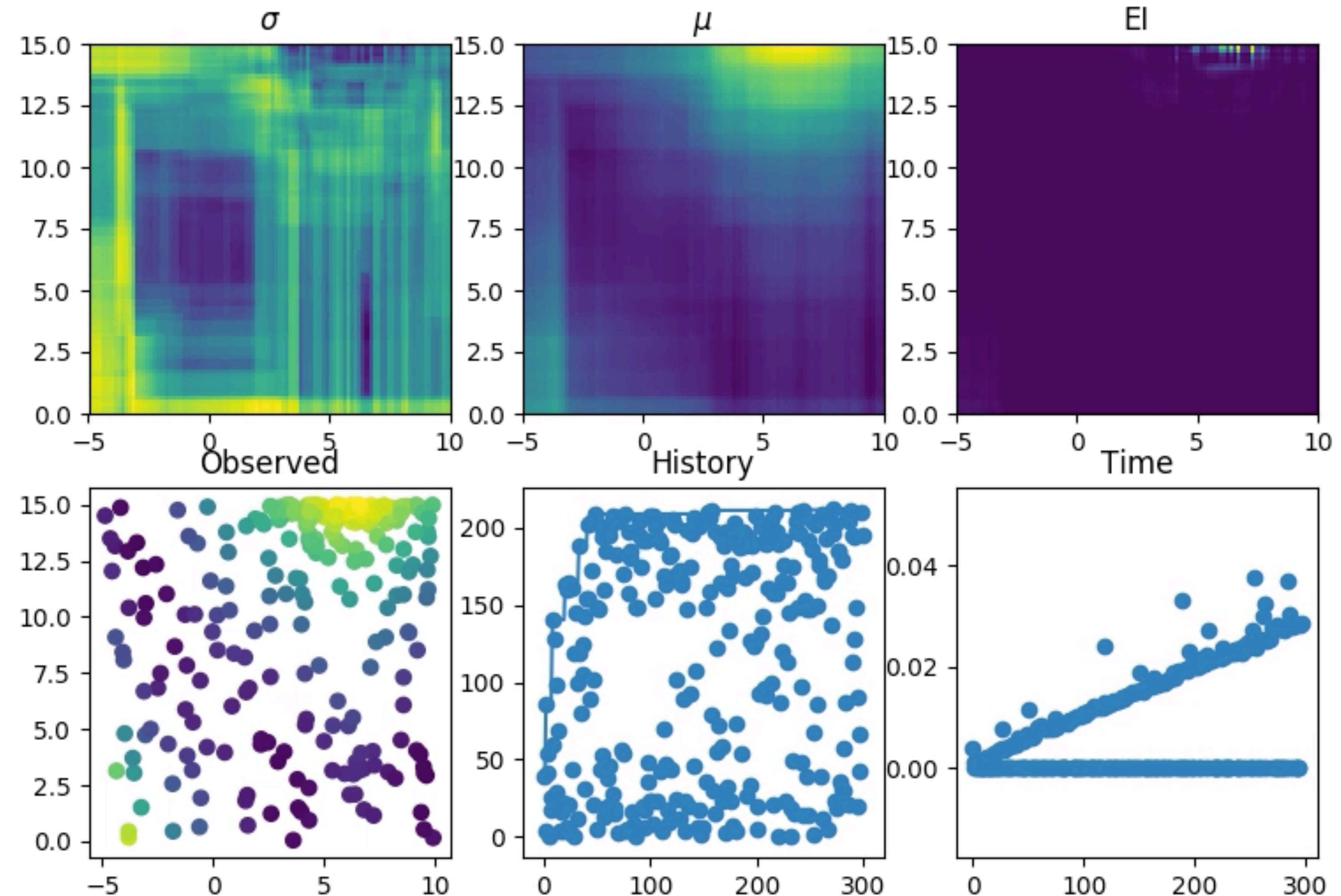
Srinivas et al. 2010, Freitas et al. 2012,  
Kawaguchi et al. 2016

- Also works for non-convex, noisy data
- Used in AlphaGo



# Bayesian Optimization (surrogate: random forests)

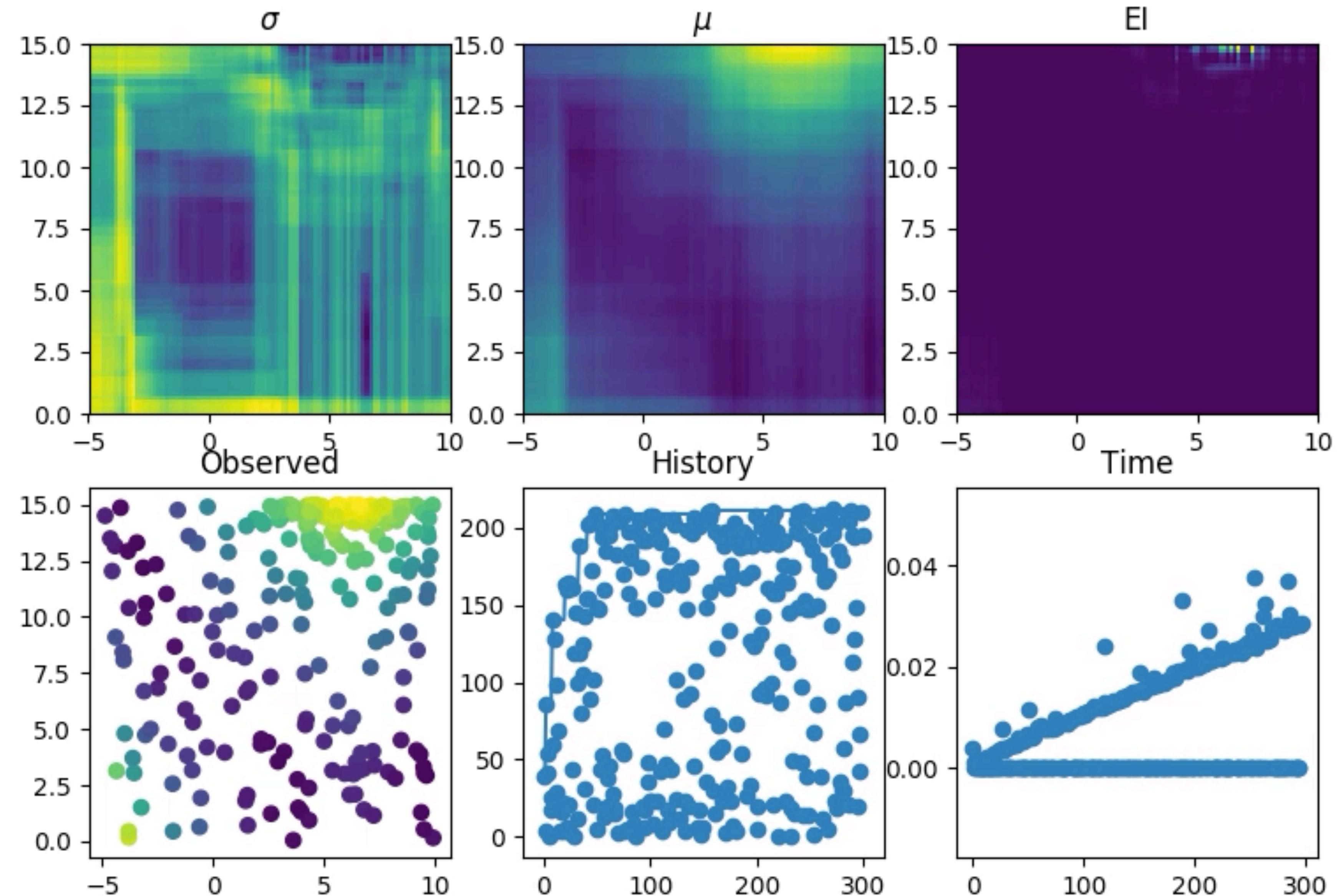
- Use random forest variance
- Scales well to many hyperparameters
- Used in Auto-sklearn



*animation by Jeroen van Hoof*

# Bayesian Optimization (surrogate: random forests)

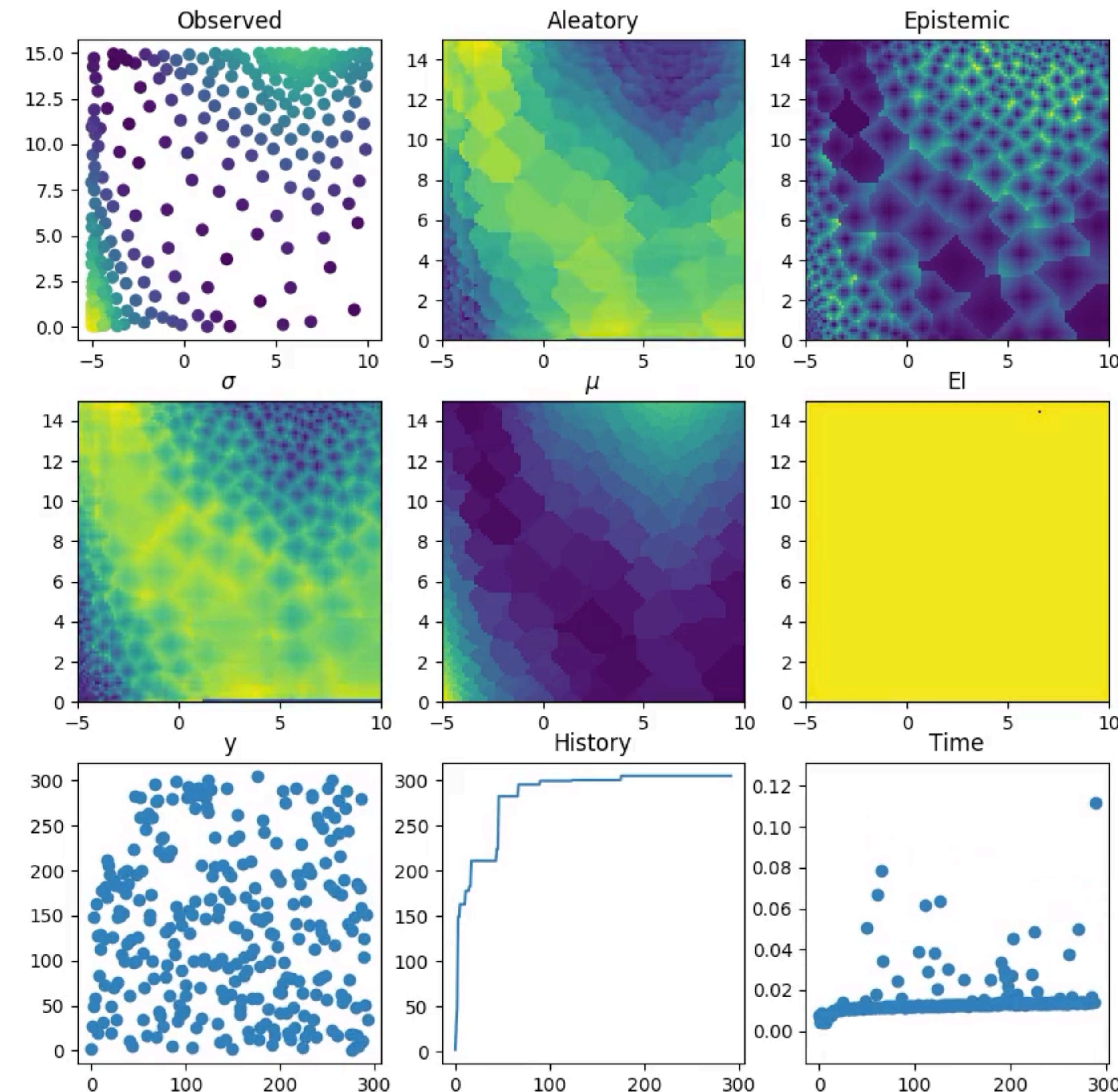
- Use random forest variance
- Scales well to many hyperparameters
- Used in Auto-sklearn



*animation by Jeroen van Hoof*

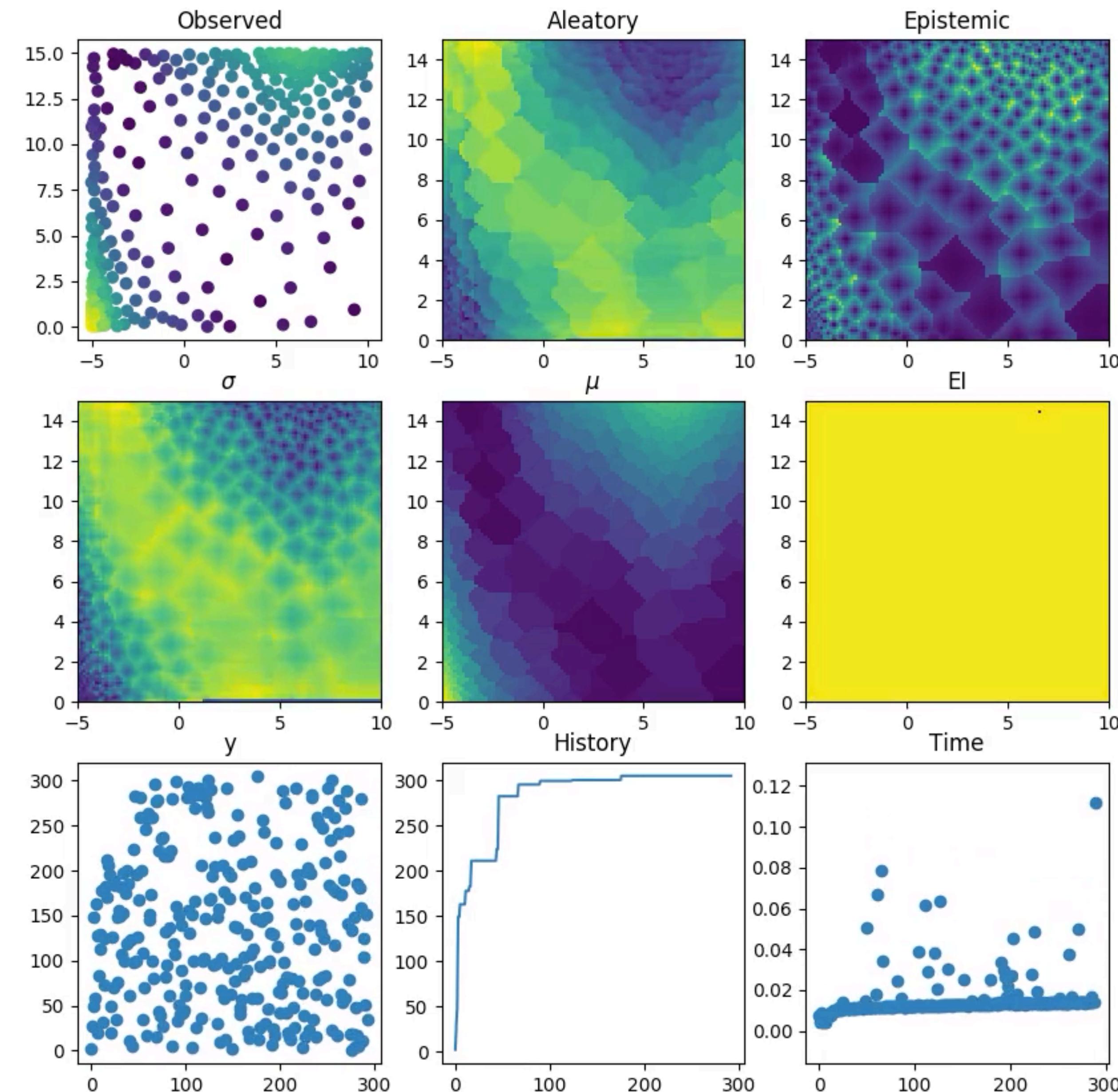
# Bayesian Optimization (surrogate: gradient boosting)

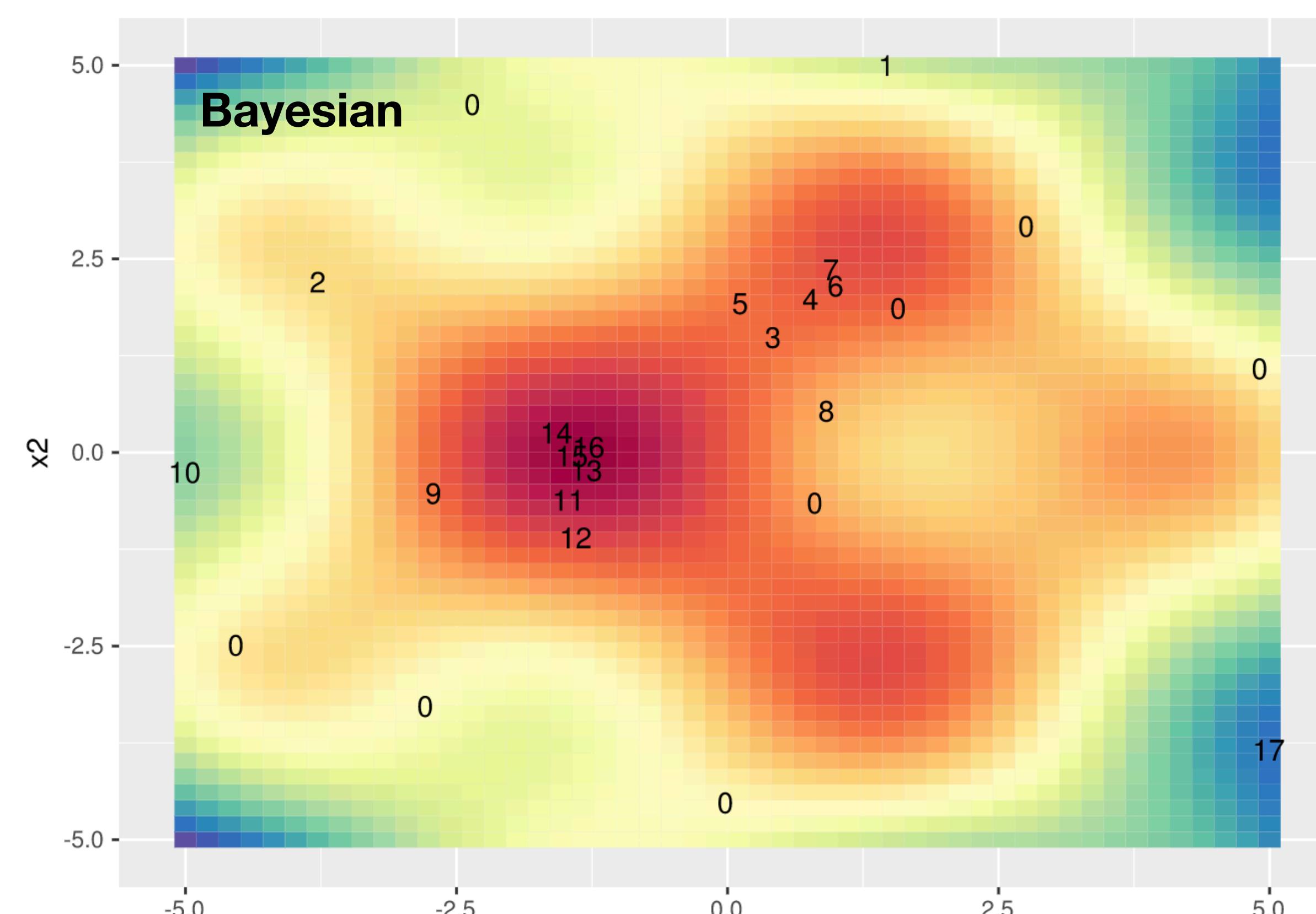
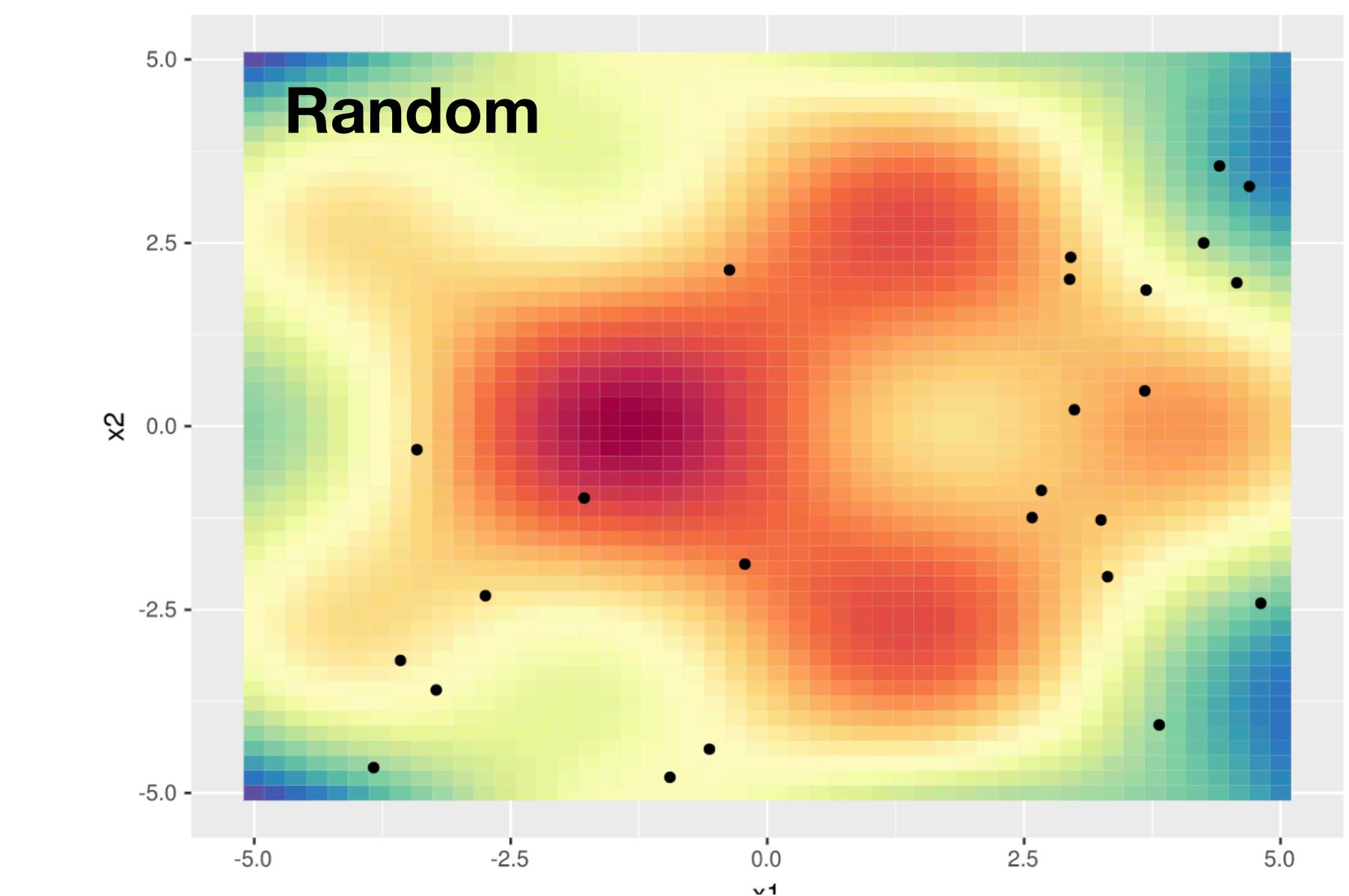
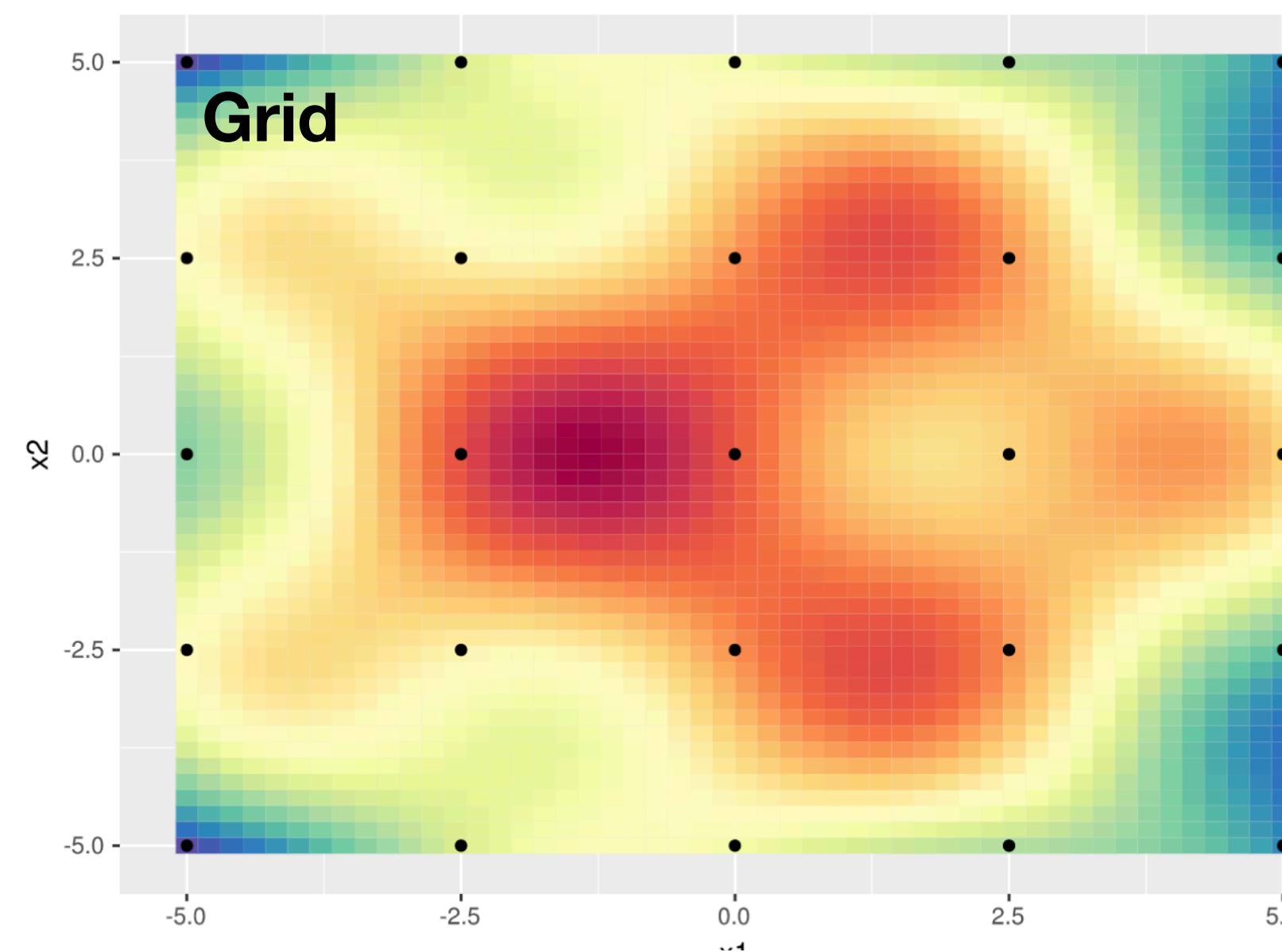
- Let gradient boosting predict the mean + upper and lower quantile
  - Faster
  - More robust to concept drift



# Bayesian Optimization (surrogate: gradient boosting)

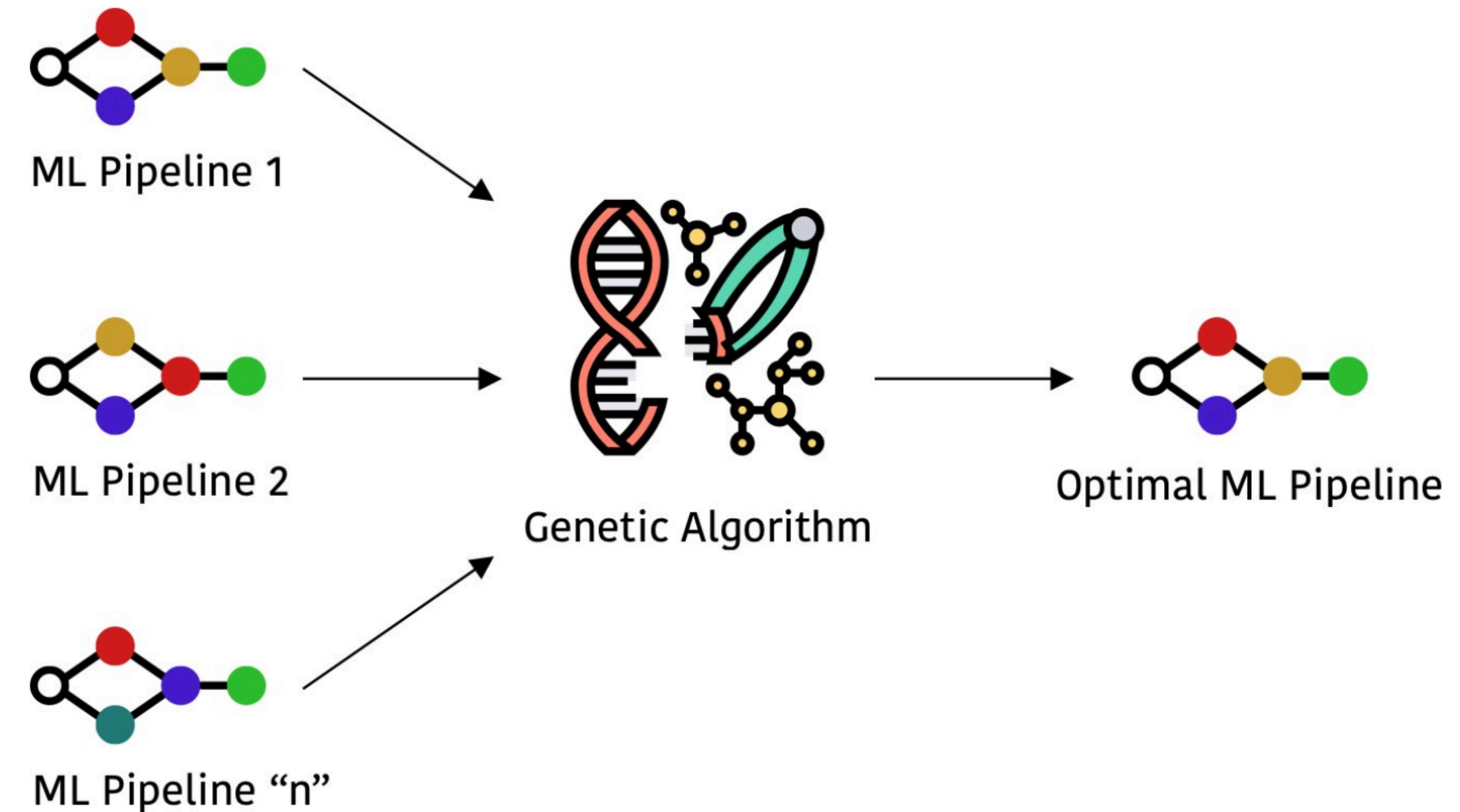
- Let gradient boosting predict the mean + upper and lower quantile
  - Faster
  - More robust to concept drift





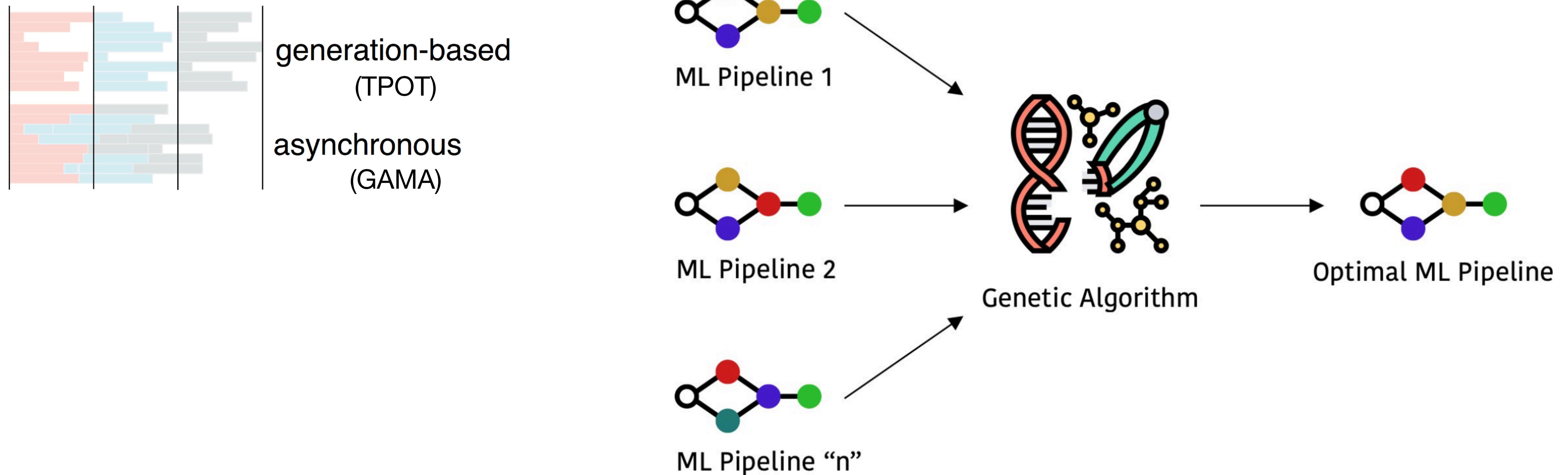
# Evolution

- Start with initial pipeline
- Best pipelines *evolve*: cross-over or mutation
- No fixed pipeline length: adapts to complexity of the problem
- Used in GAMA, TPOT,...



# Evolution

- Start with initial pipeline
- Best pipelines *evolve*: cross-over or mutation
- No fixed pipeline length: adapts to complexity of the problem
- Used in GAMA, TPOT,...

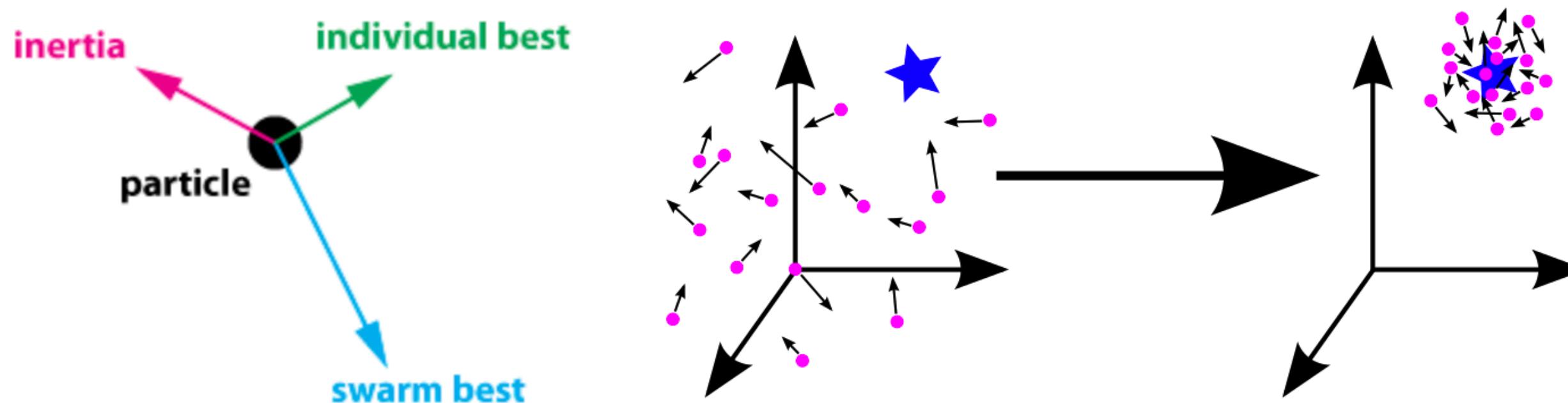
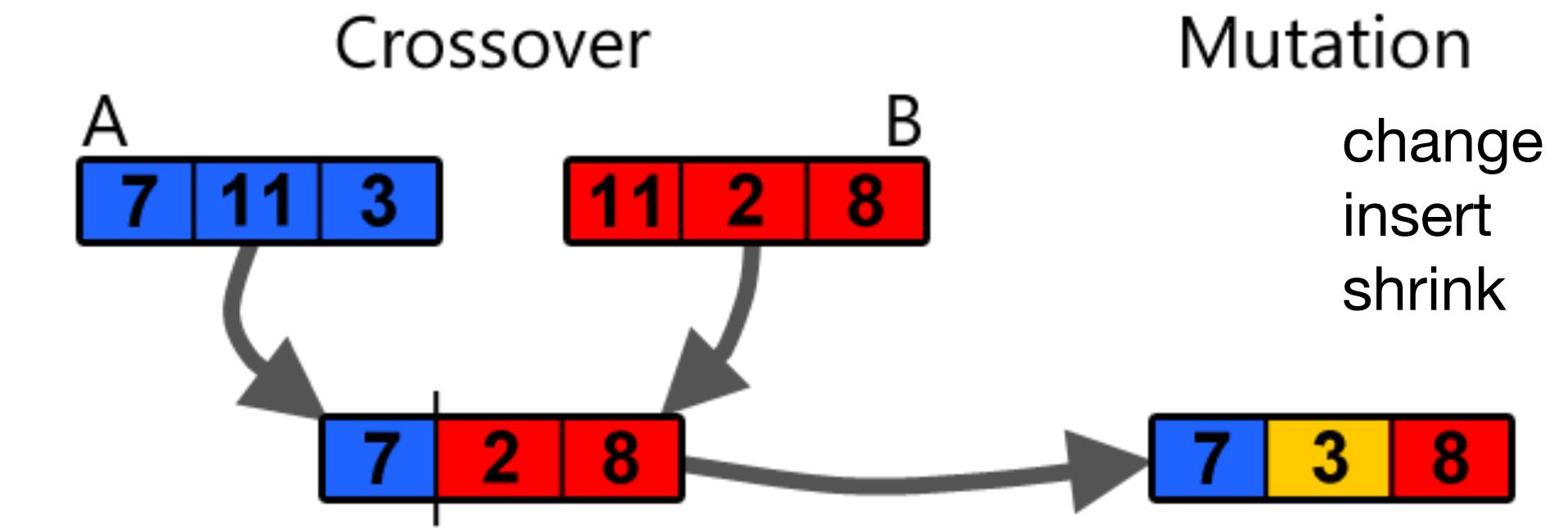


# Evolution

- Genetic programming
  - Mutations: add, mutate/tune, remove HP
- Particle swarm optimization *Mantovani et al 2015*

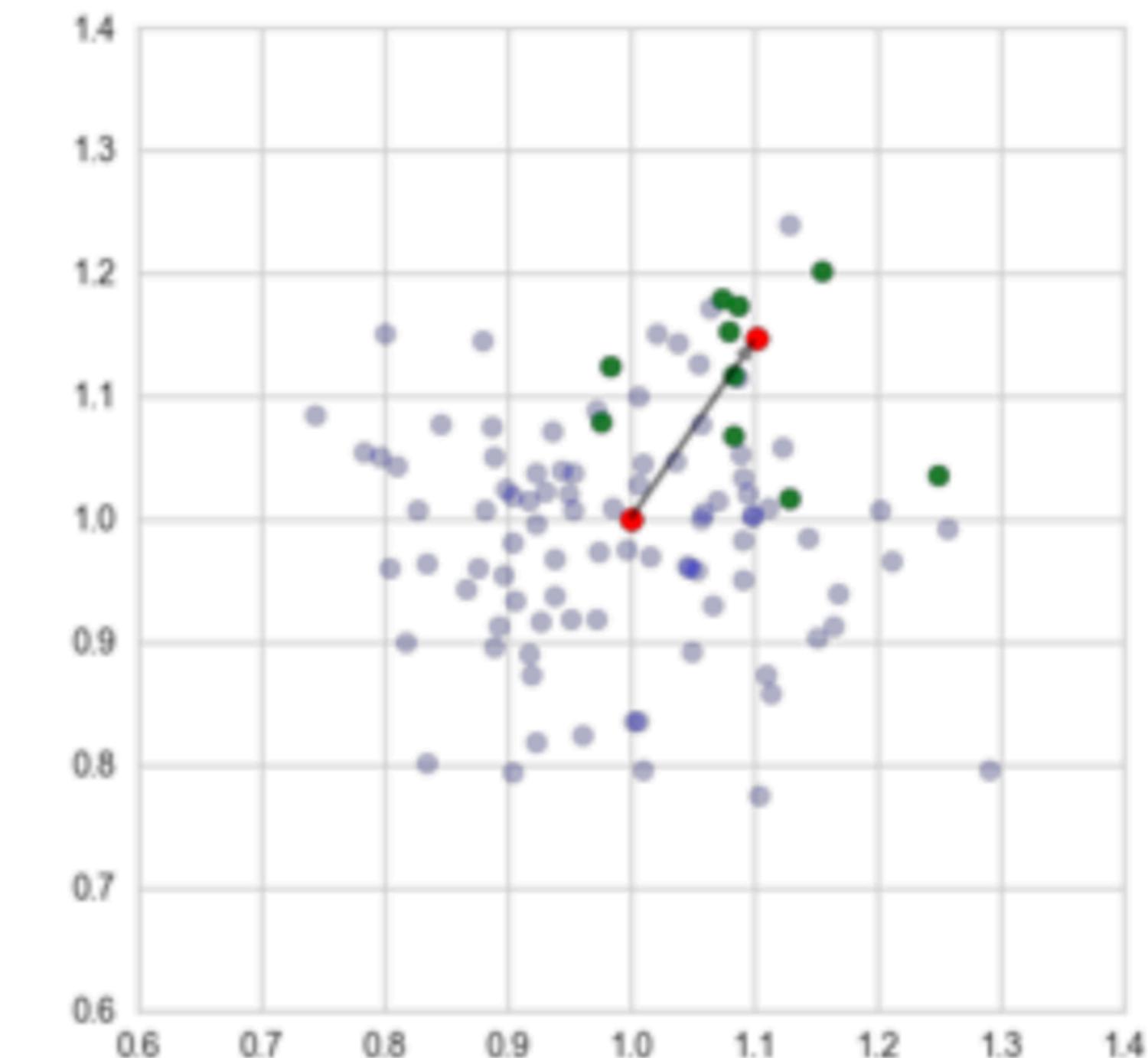
*TPOT Olson 2016*

*GAMA Gijsbers 2020*



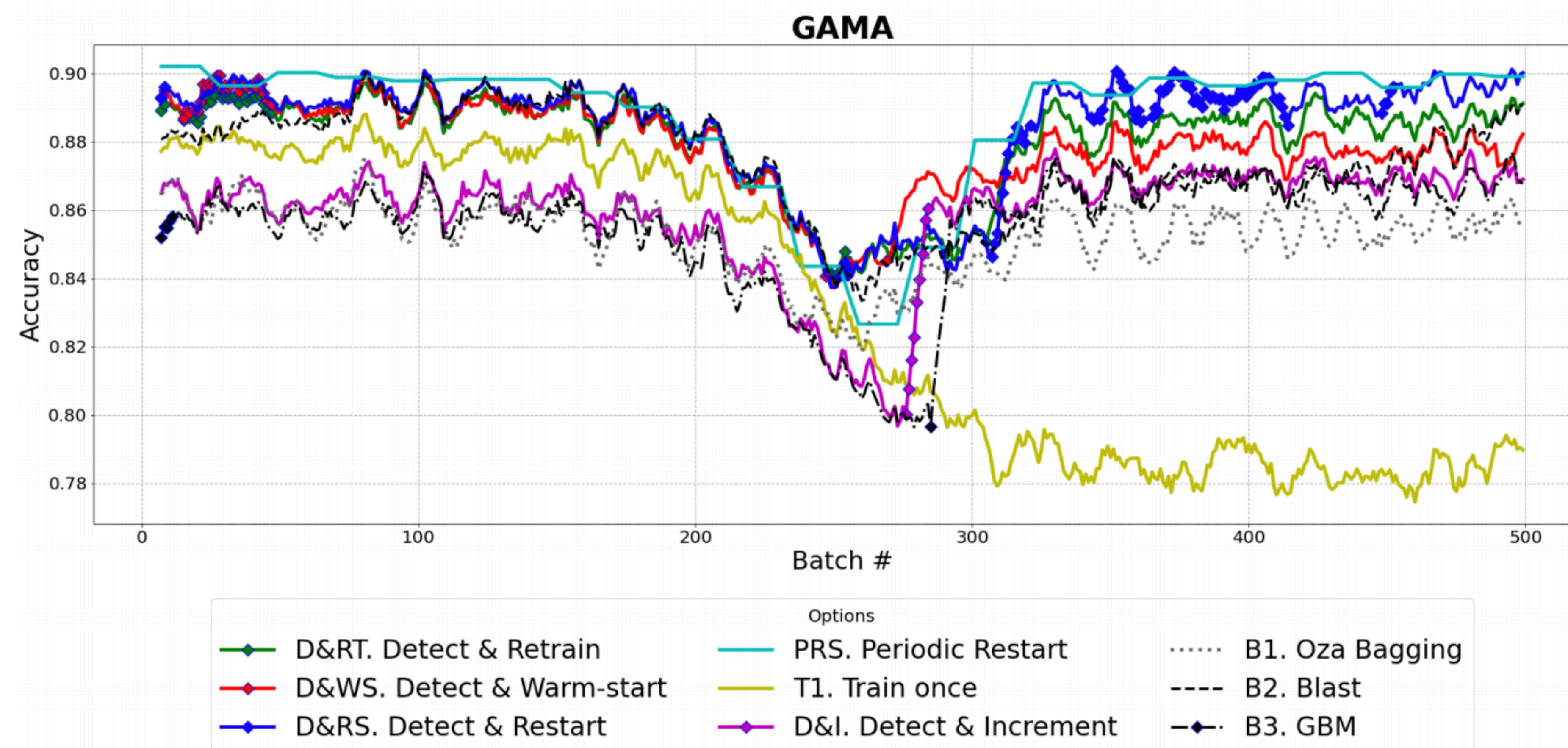
- Covariance matrix adaptation evolution (CMA-ES) *Hansen 2015*
  - Purely continuous, expensive
  - Competitive to optimize deep neural nets

*[Loshilov, Hutter 2016]*



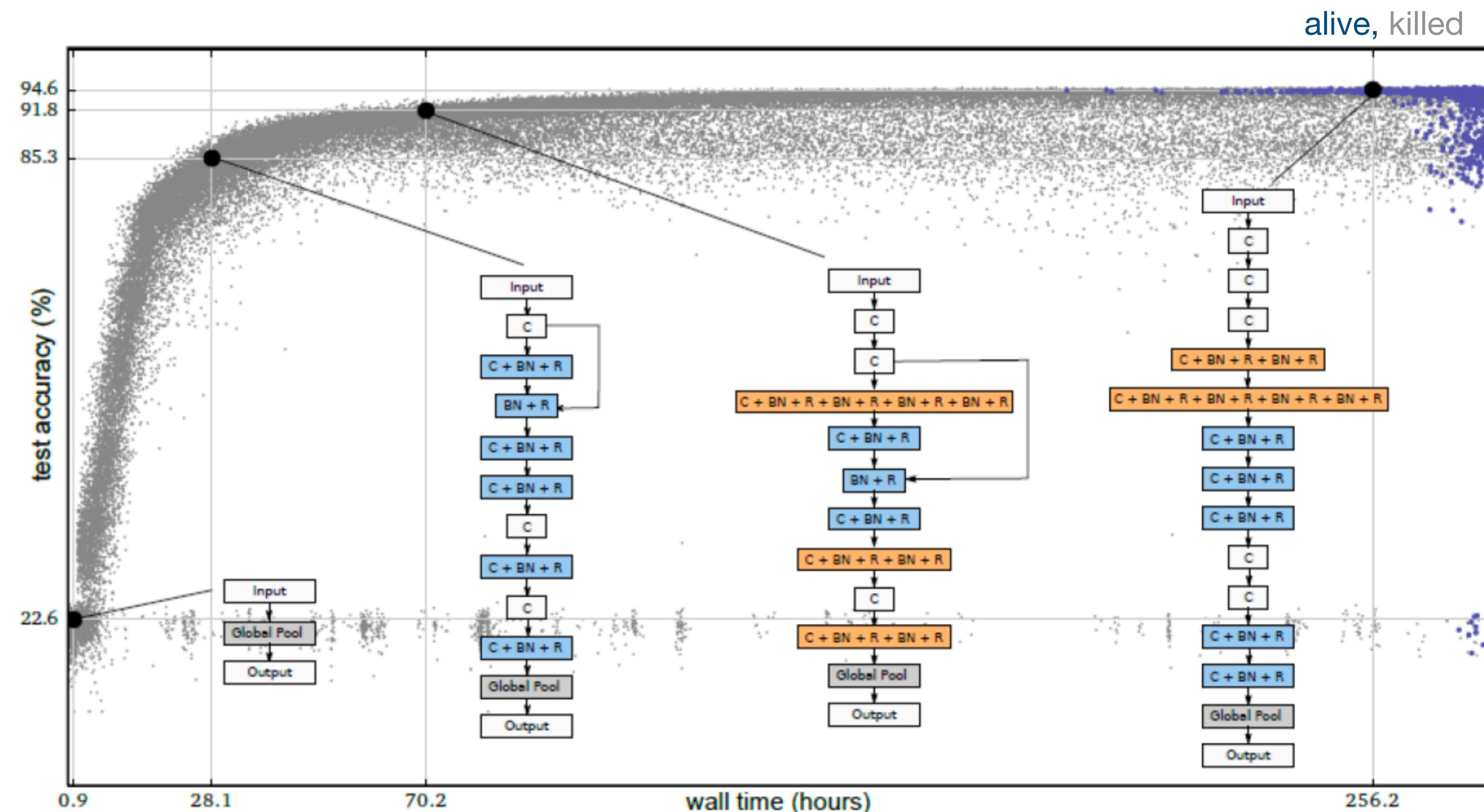
# Evolution: benefits

- Easy to parallelize, and quickly adapts to changes in the data
- Easy warm-starting: if the data changes, re-start AutoML but start with best prior pipelines



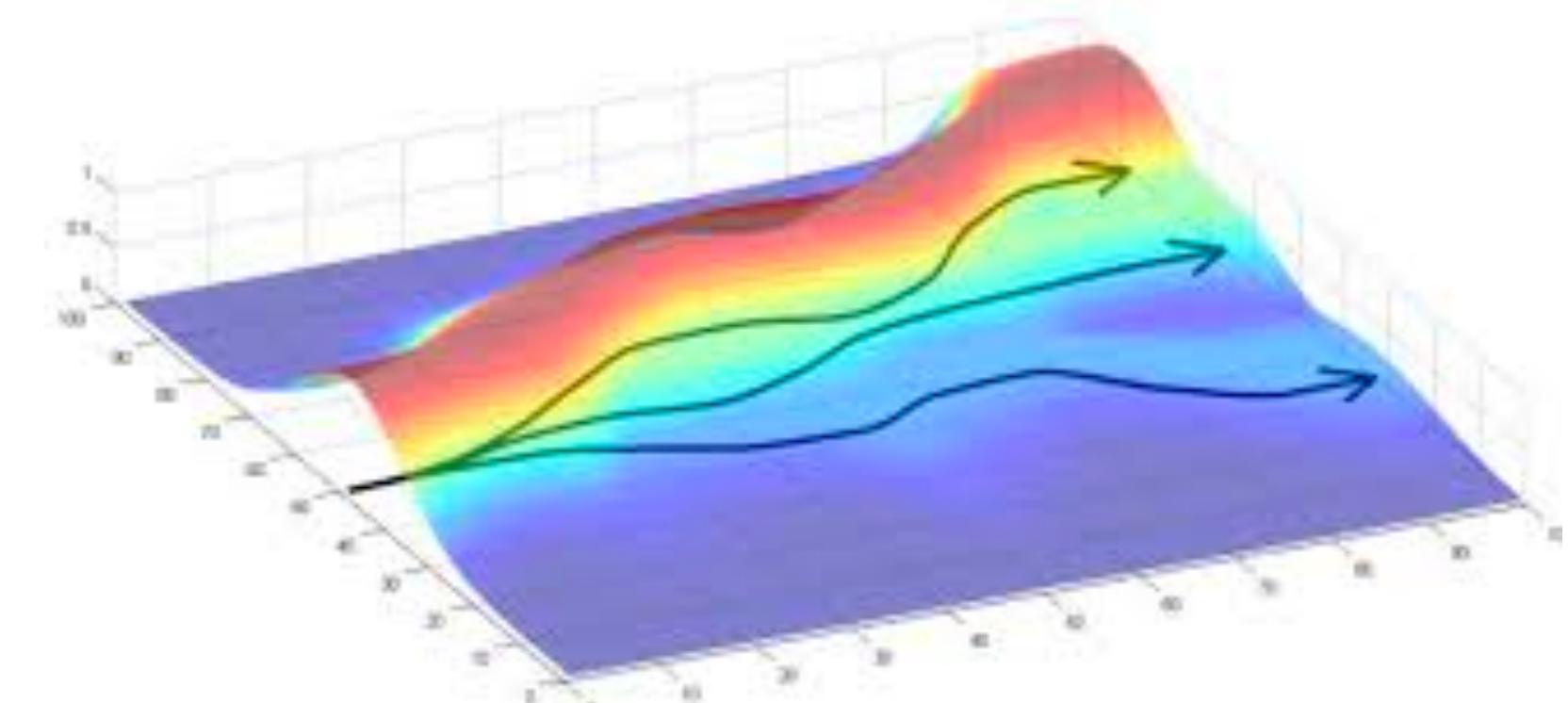
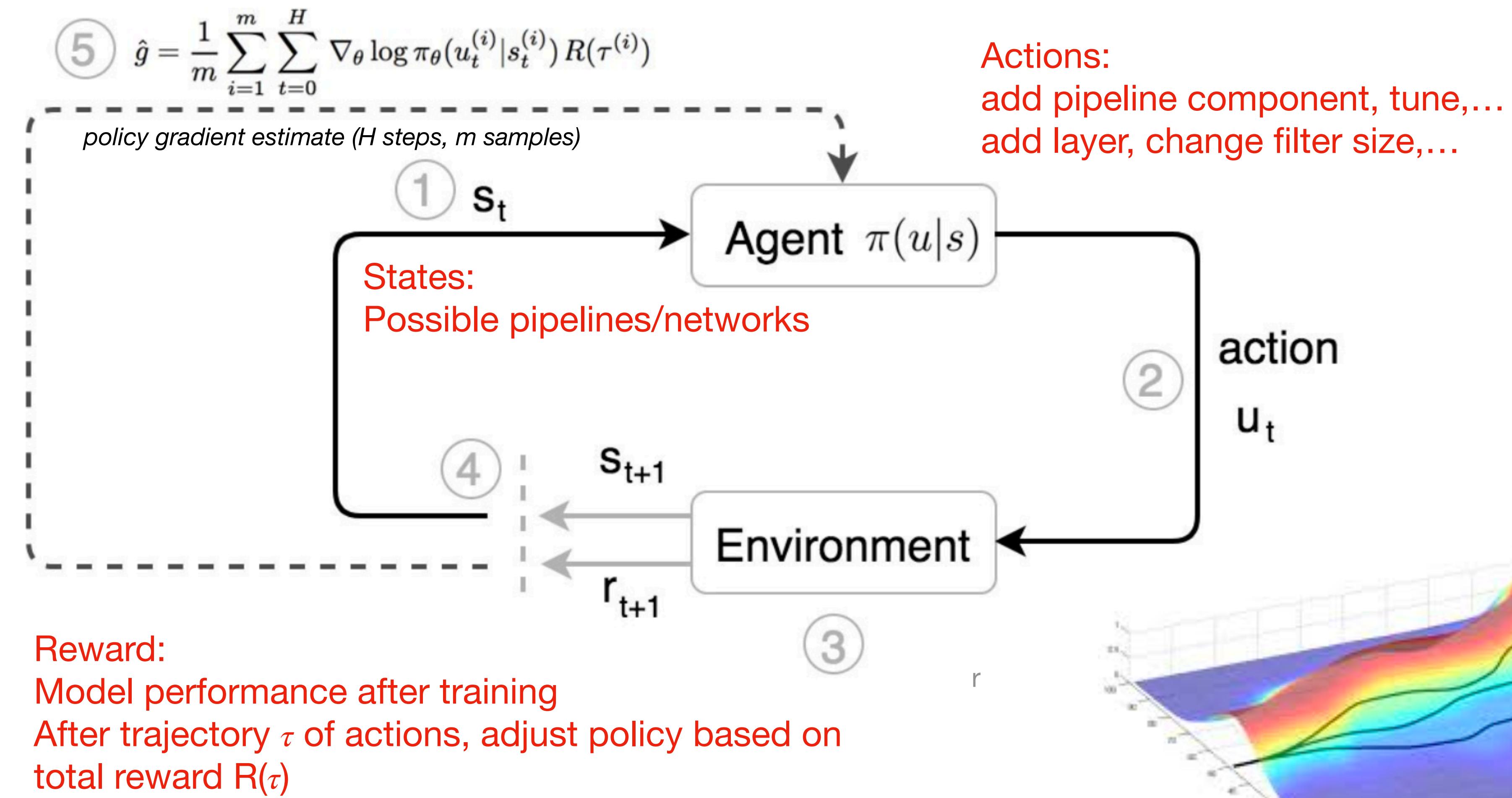
# Neuro-evolution

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



# Optimization: Reinforcement learning

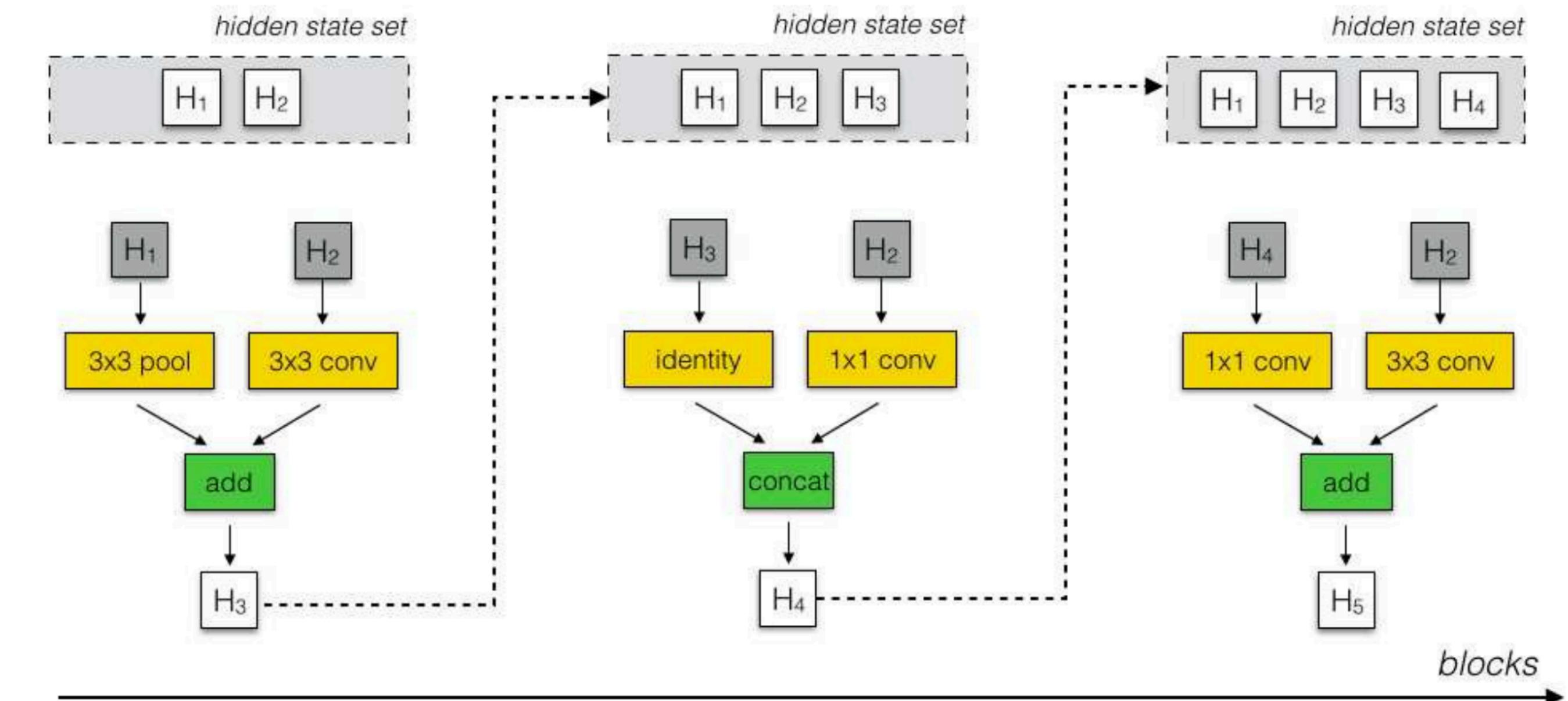
Build pipeline or network step-by-step, learn general strategy (policy)



# NAS with Reinforcement learning

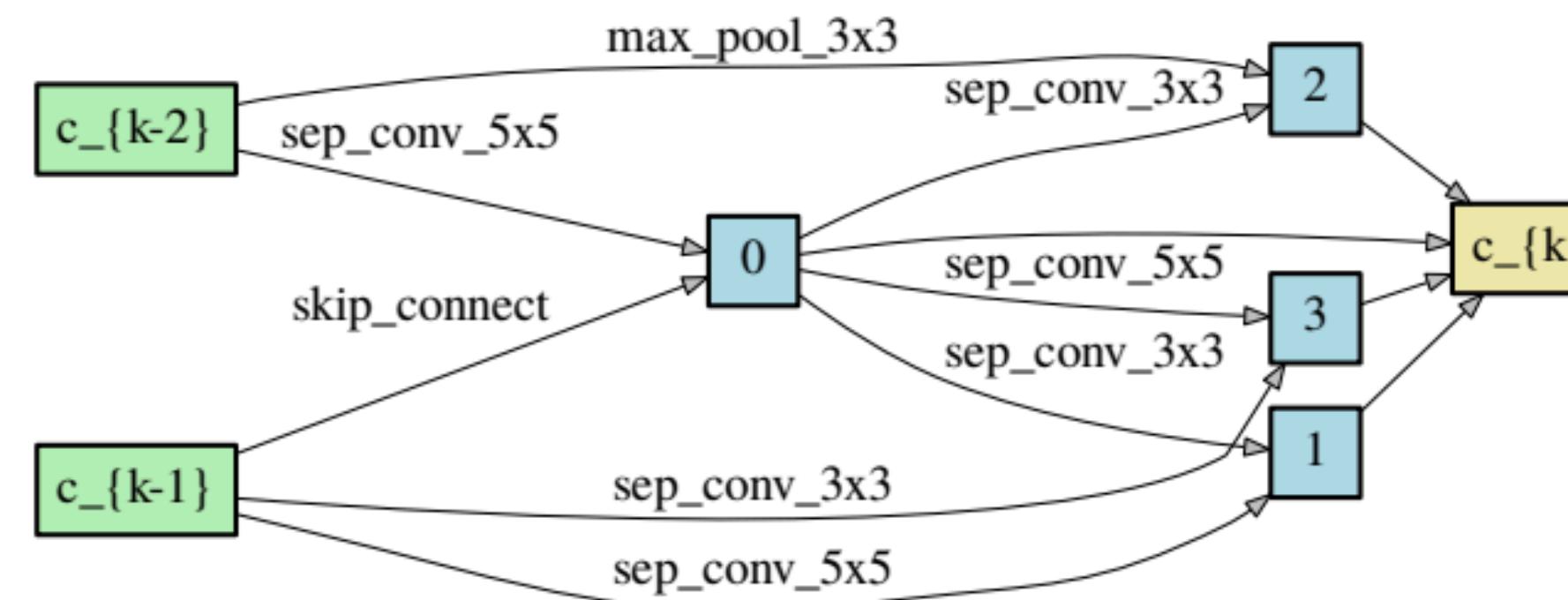
1-layer LSTM (PPO), cell space search

- State of the art on ImageNet
- **450 GPUs, 3-4 days**, 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

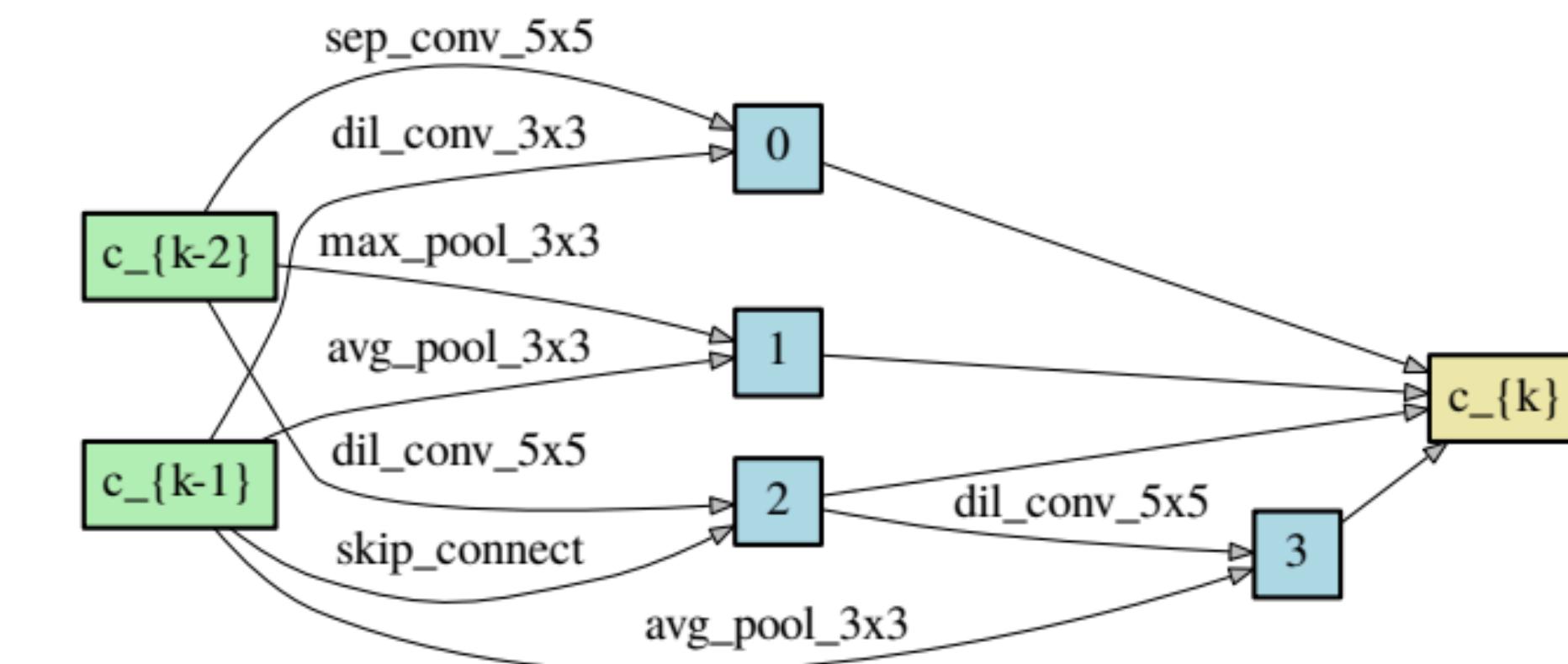


# Sidenote: NAS with random search?

*If you constrain the search space enough, you can get SOTA results with random search!*



(a) Normal Cell



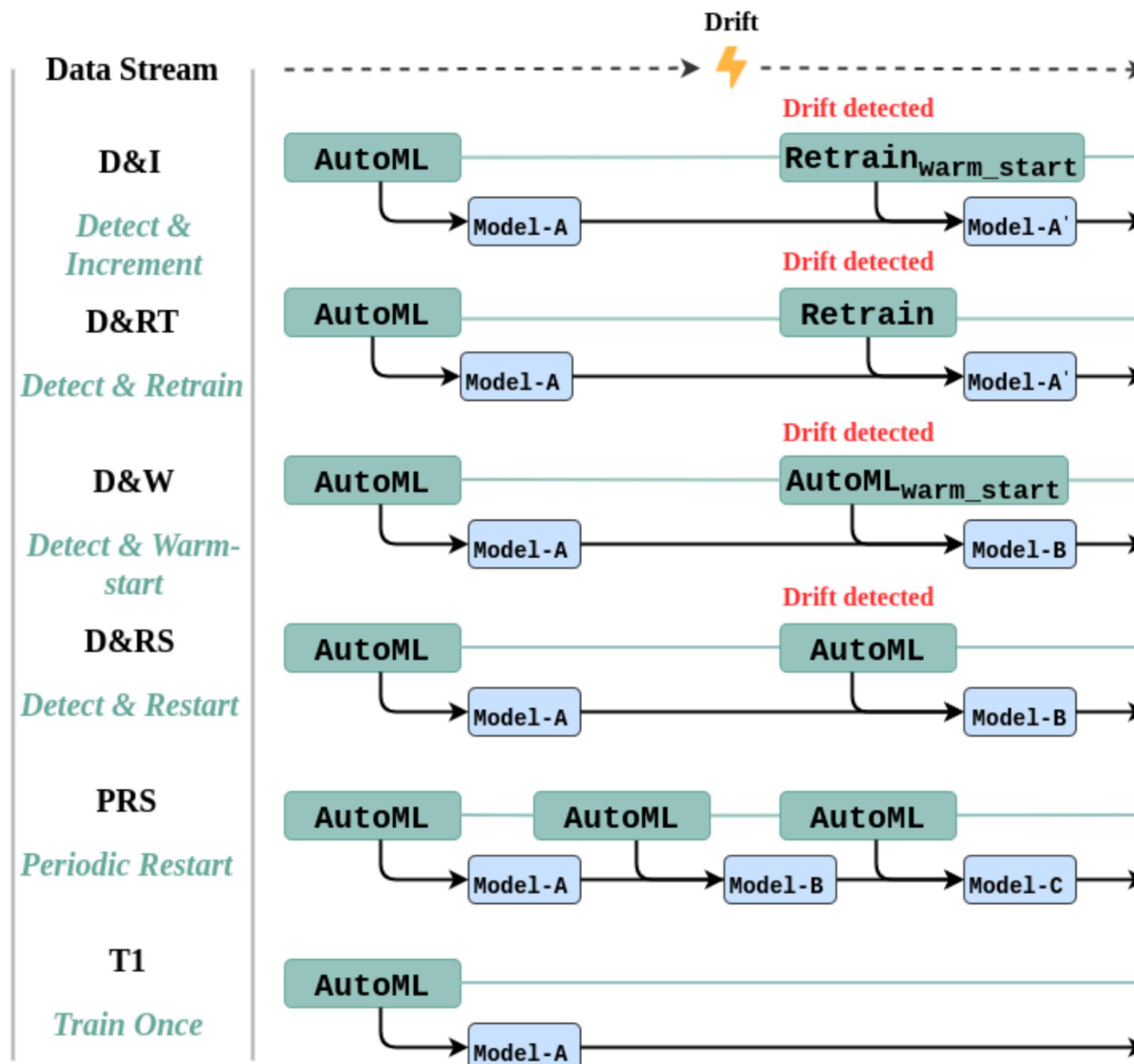
(b) Reduction Cell

**Convolutional Cells on CIFAR-10 Benchmark:** Best architecture found by random search with weight-sharing.

# Handling concept drift

In many real-word cases, the data evolves over time

Requires adaptation strategies



- *High Abrupt Drift*

Re-optimize pipelines from scratch after drift detection.  
Replace with better models when found.

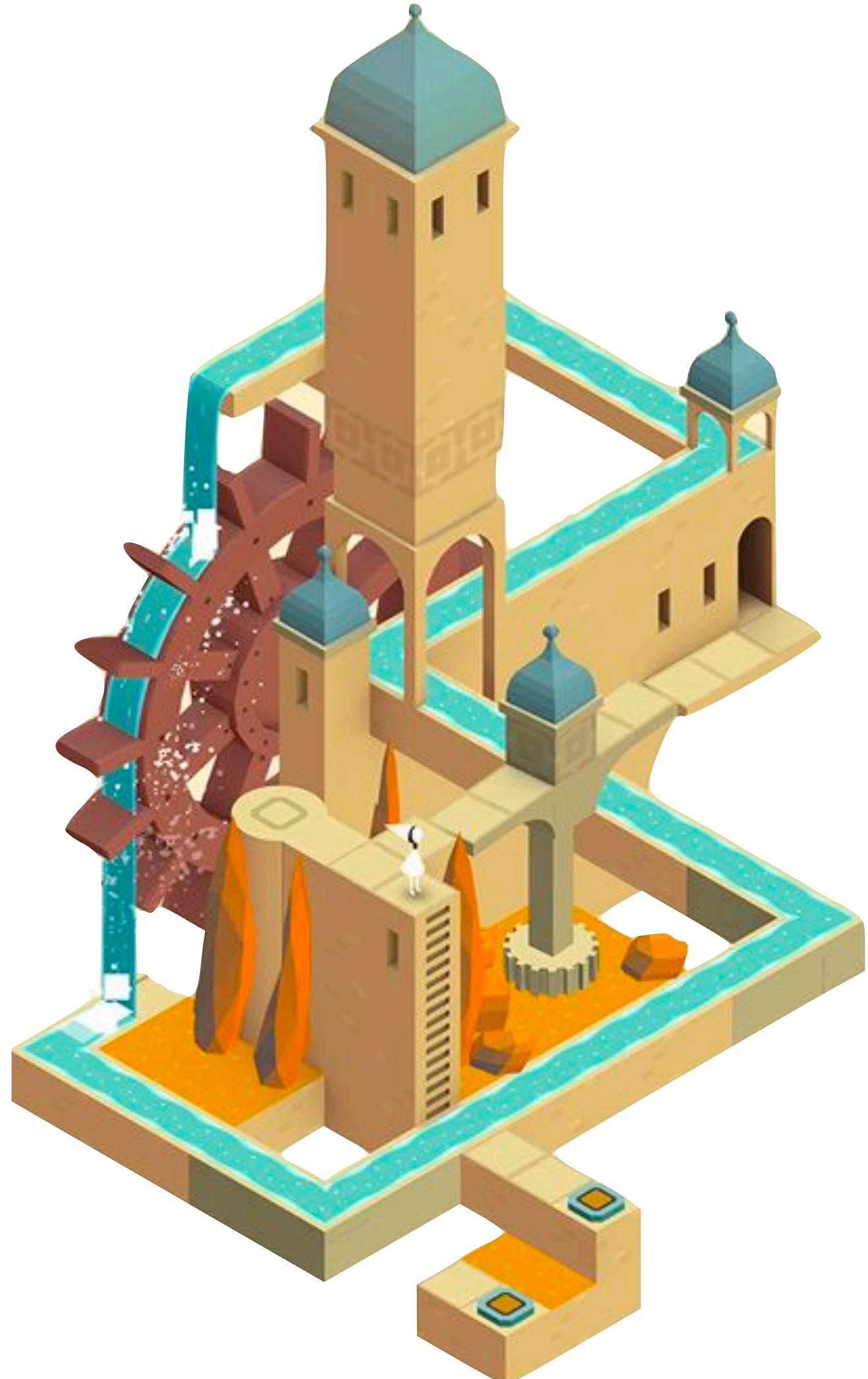
- *Effect of Drift Type: High Gradual Drift*

Retraining pipelines from scratch after a high gradual drift performs  
Incremental learning also works quite well for all AutoML methods under this  
drift.

- *Effect of Drift Type: High Mixed Drift*

Restarting the AutoML on new data (D&RS) performs best overall.

# Overview

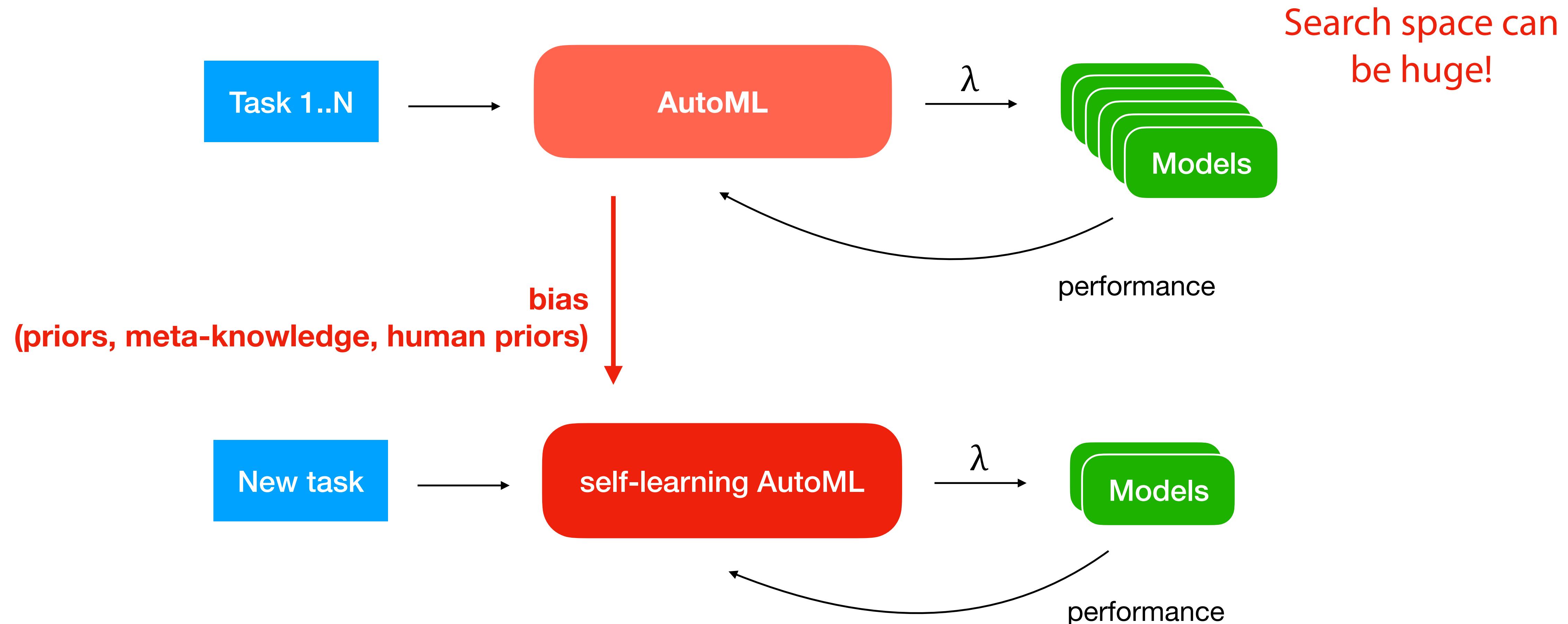


**Part 3: Learning how to do AutoML**  
**Closing the loop**

# AutoML + meta-learning

Meta-learn how to design architectures/pipelines and tune hyper parameters

*Human data scientists also learn from experience*



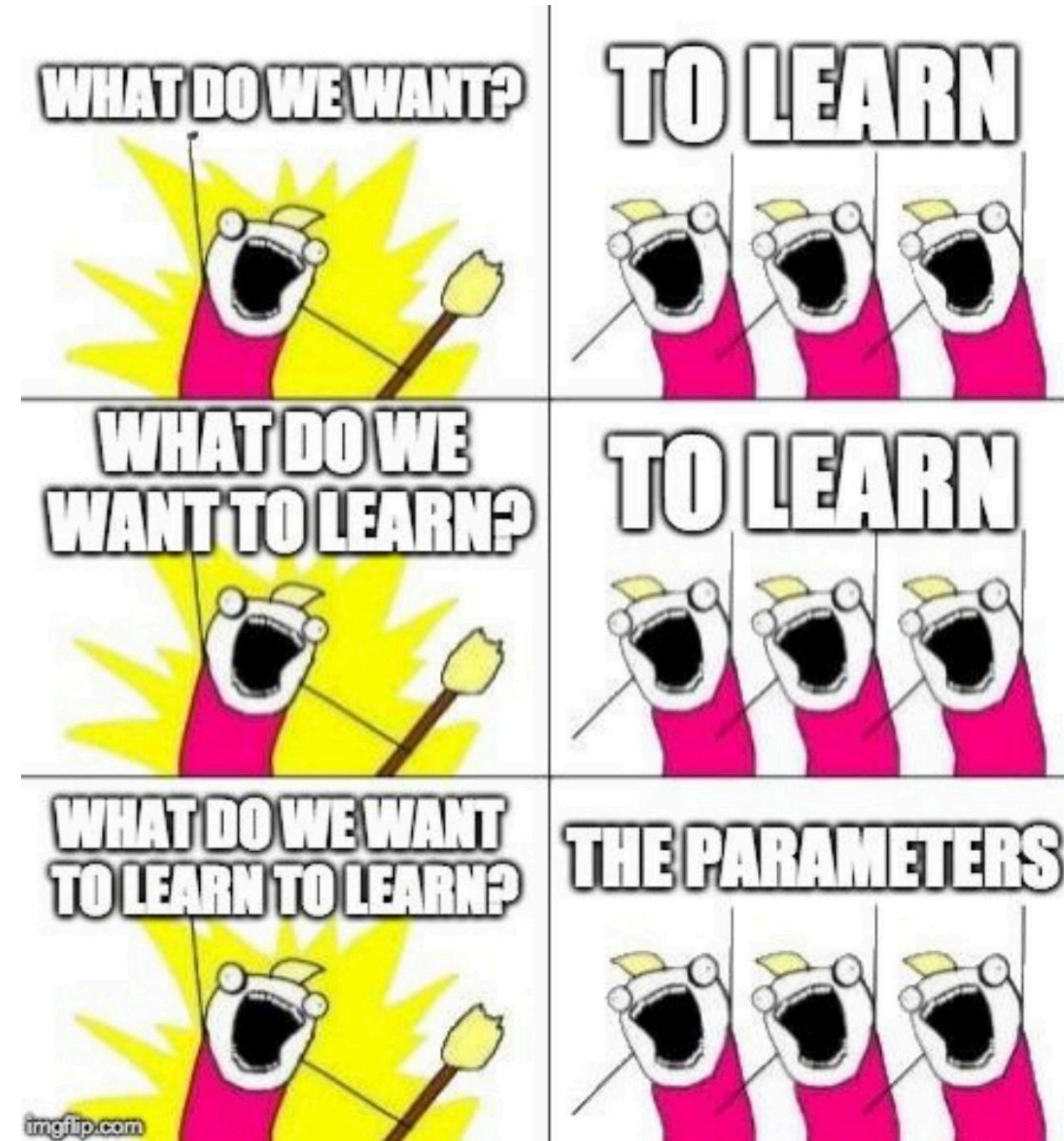
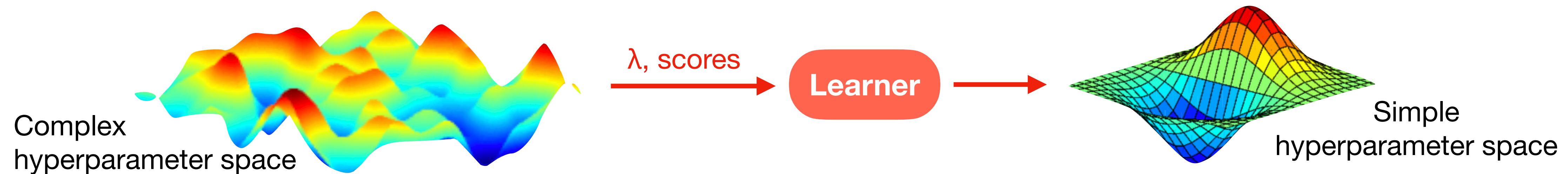


Image: Pesah et al. 2018

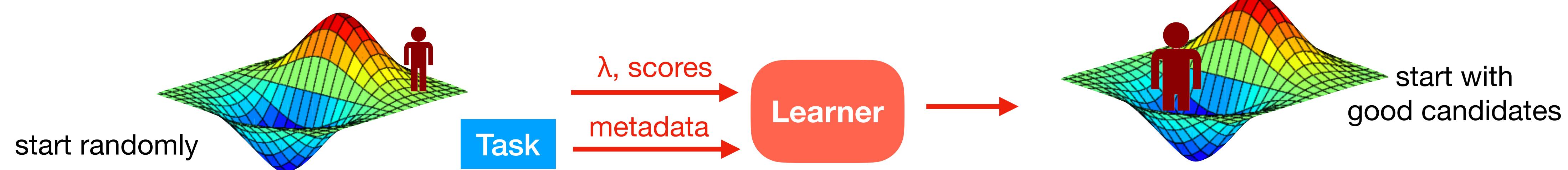
# Meta-learning for AutoML: how?

hyperparameters = architecture + hyperparameters

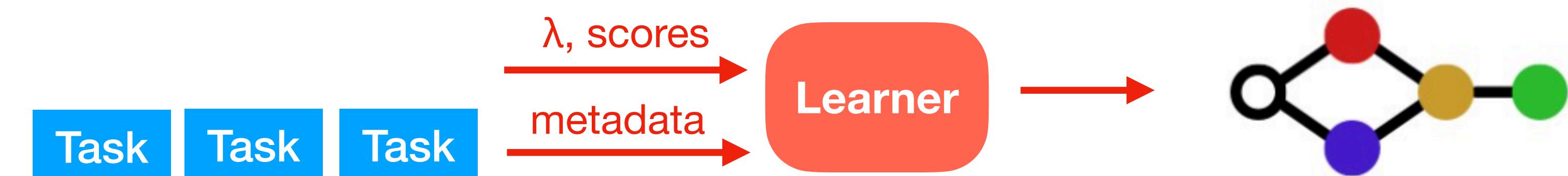
## Learning hyperparameter priors



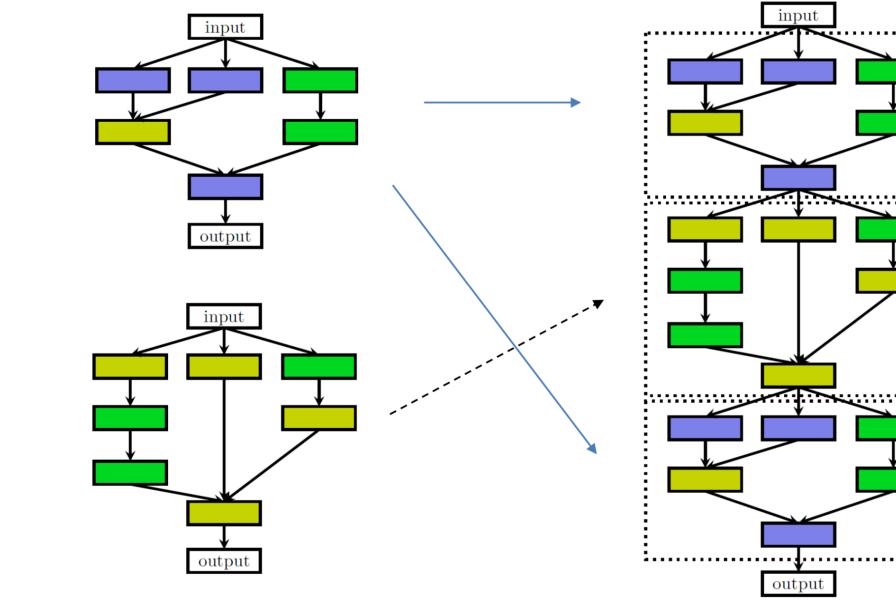
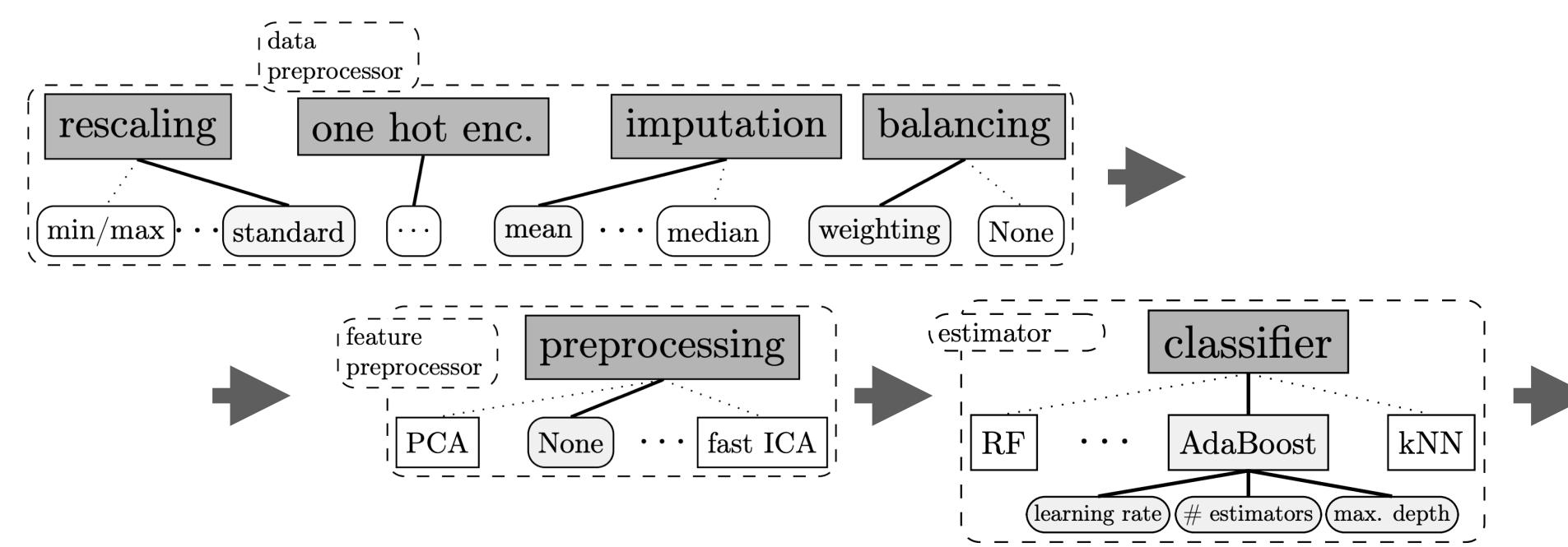
## Warm starting (what works on *similar* tasks?)



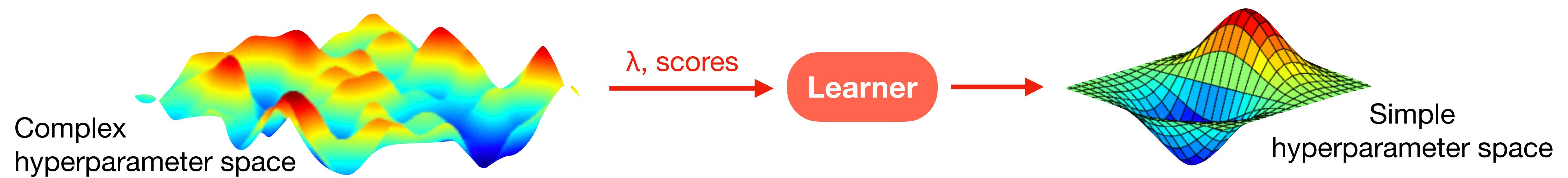
## Meta-models (learn how to build models/components)



# Observation: current AutoML strongly depends on learned priors



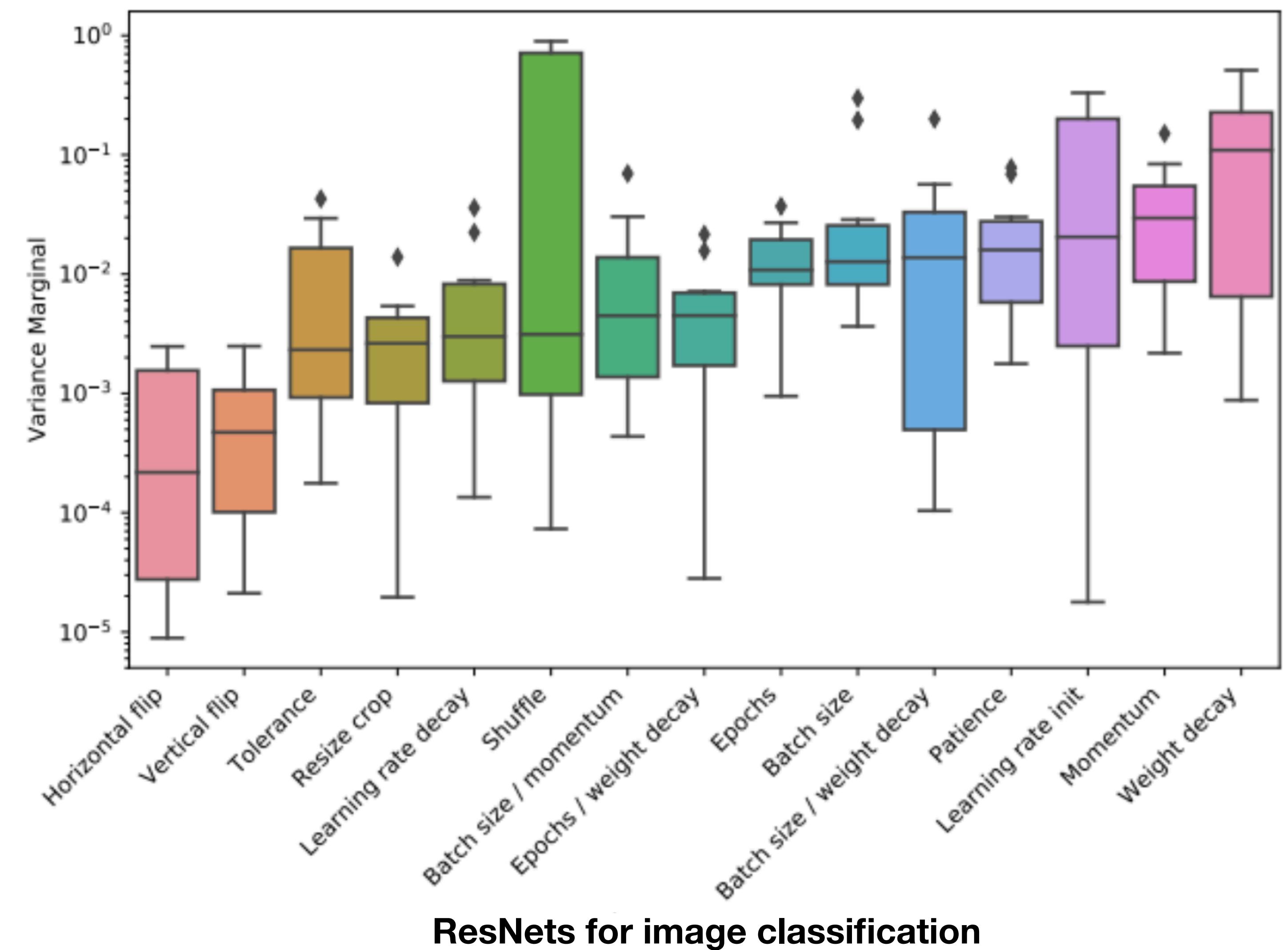
# Can we *learn* good hyperparameter priors?



# Learn hyperparameter importance

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

- Select hyperparameters that cause variance in the evaluations.
- Useful to speed up black-box optimization techniques



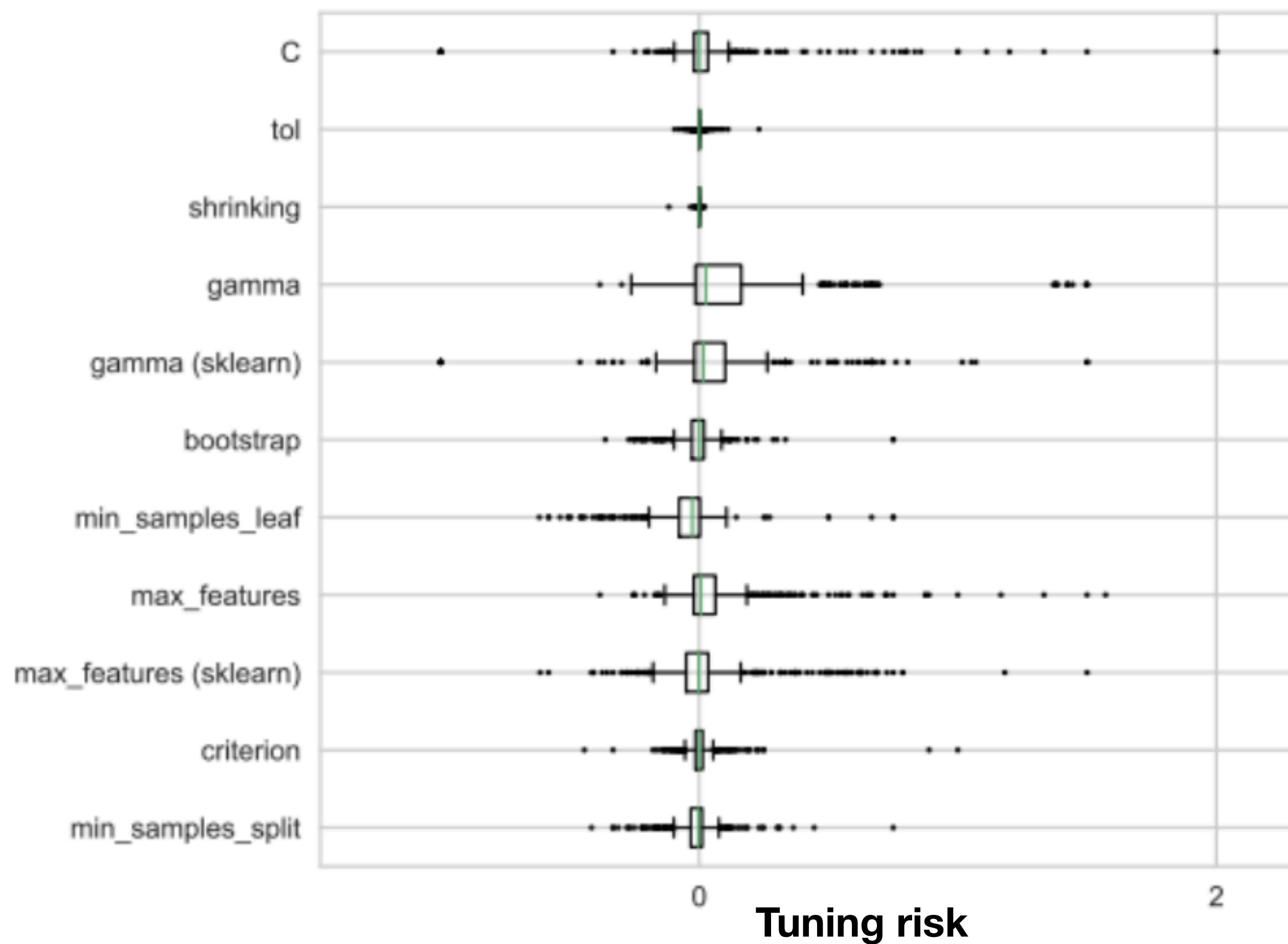
# Learn defaults + hyperparameter importance

- **Tunability** <sup>1,2,3</sup>

**Learn** good defaults, measure importance as **improvement** via tuning

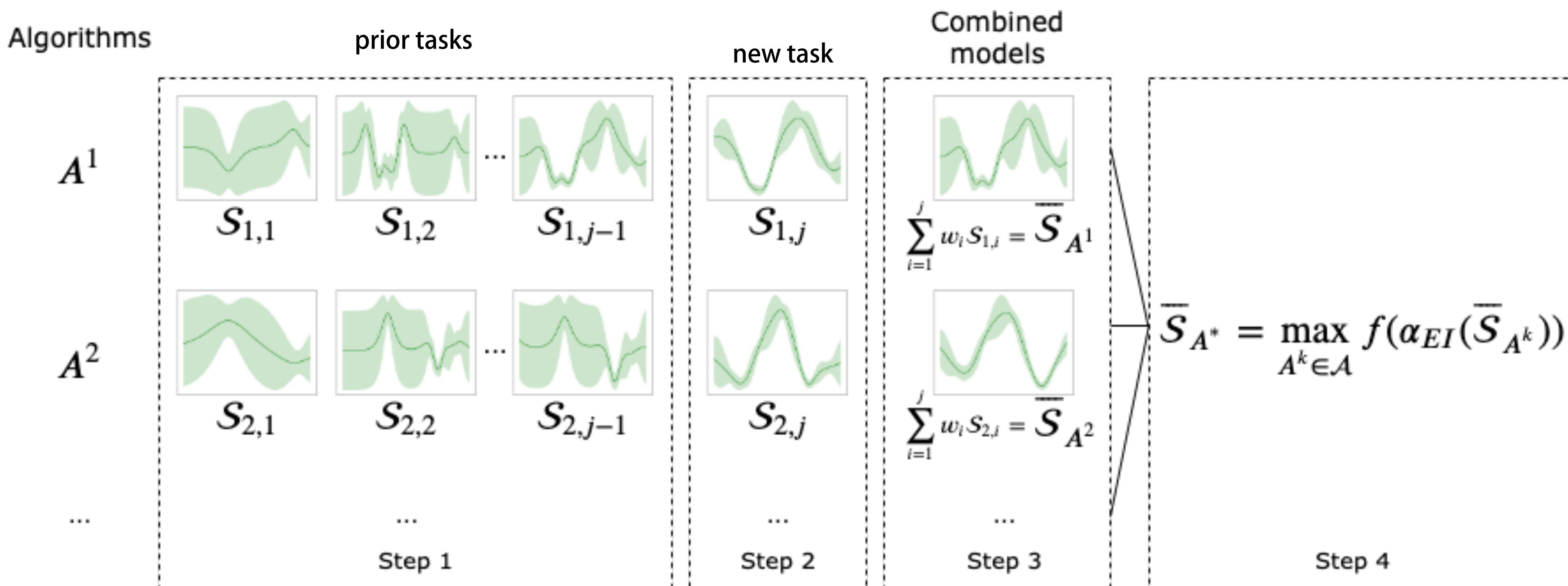
function
max_features
$m = 0.16^*p$
$m = p^{0.74}$
$m = 1.15^{\sqrt{p}}$
$m = \sqrt{p}$
gamma
$m = 0.00574^*p$
$m = 1/p$
$m = 0.006$

Learned defaults



# Surrogate model transfer

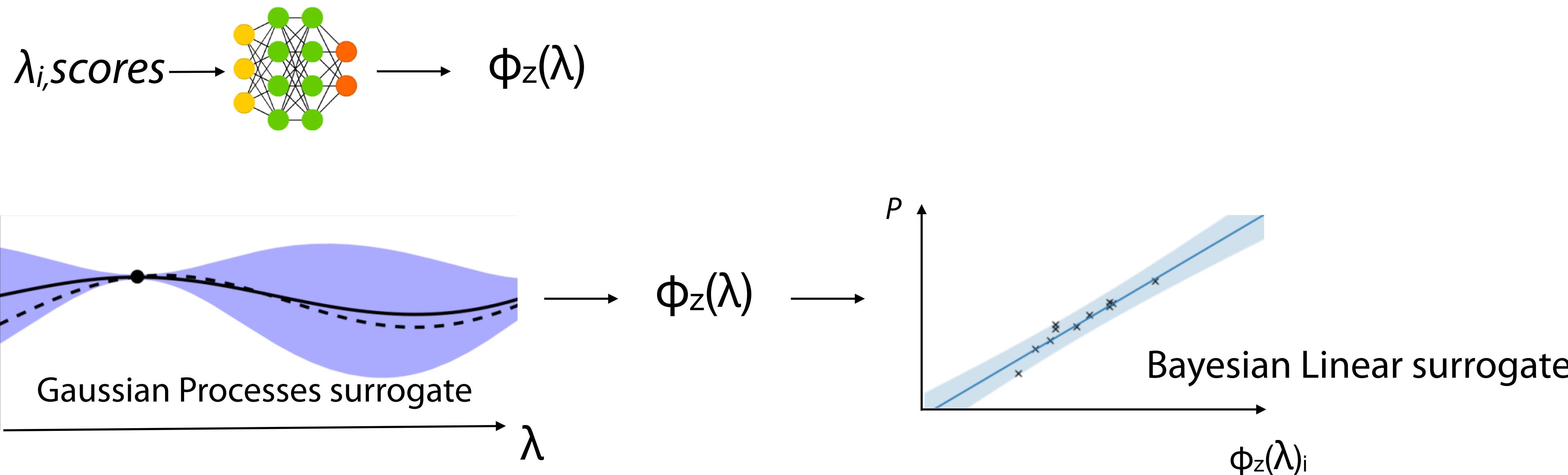
- If task  $j$  is *similar* to the new task, its surrogate model  $S_j$  will likely transfer well
- Sum up all  $S_j$  predictions, weighted by task similarity
- Build combined surrogate, *weighted by current performance* on new task<sup>2</sup>



# Learn basis expansions for hyperparameters

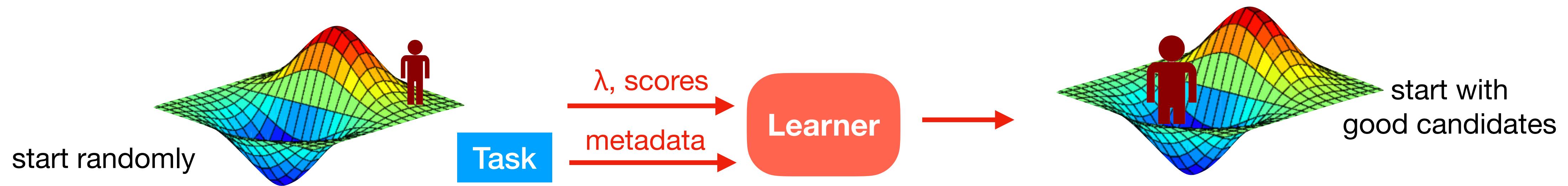
- Hyperparameters can interact in very non-linear ways
- Use a neural net to learn a suitable transform  $\phi_z(\lambda)$  so that they behave linearly
- Used in SageMaker AutoML

Learn basis expansion on lots of data (e.g. OpenML)



# Warm starting

(what works on *similar* tasks?)



<sup>1</sup> [Vanschoren 2018](#)

<sup>2</sup> [Achille et al. 2019](#)

<sup>3</sup> [Alvarez-Melis et al. 2020](#)

<sup>4</sup> [Drori et al. 2019](#)

<sup>5</sup> [Jooma et al. 2020](#)

<sup>6</sup> [de Bie et al. 2020](#)

# How to measure task similarity?

- Hand-designed (statistical) meta-features that describe (tabular) datasets <sup>1</sup>
- Task2Vec: task embedding for image data <sup>2</sup>
- Optimal transport: similarity measure based on comparing probability distributions <sup>3</sup>
- Metadata embedding based on textual dataset description <sup>4</sup>
- Dataset2Vec: compares batches of datasets <sup>5</sup>
- Distribution-based invariant deep networks <sup>6</sup>

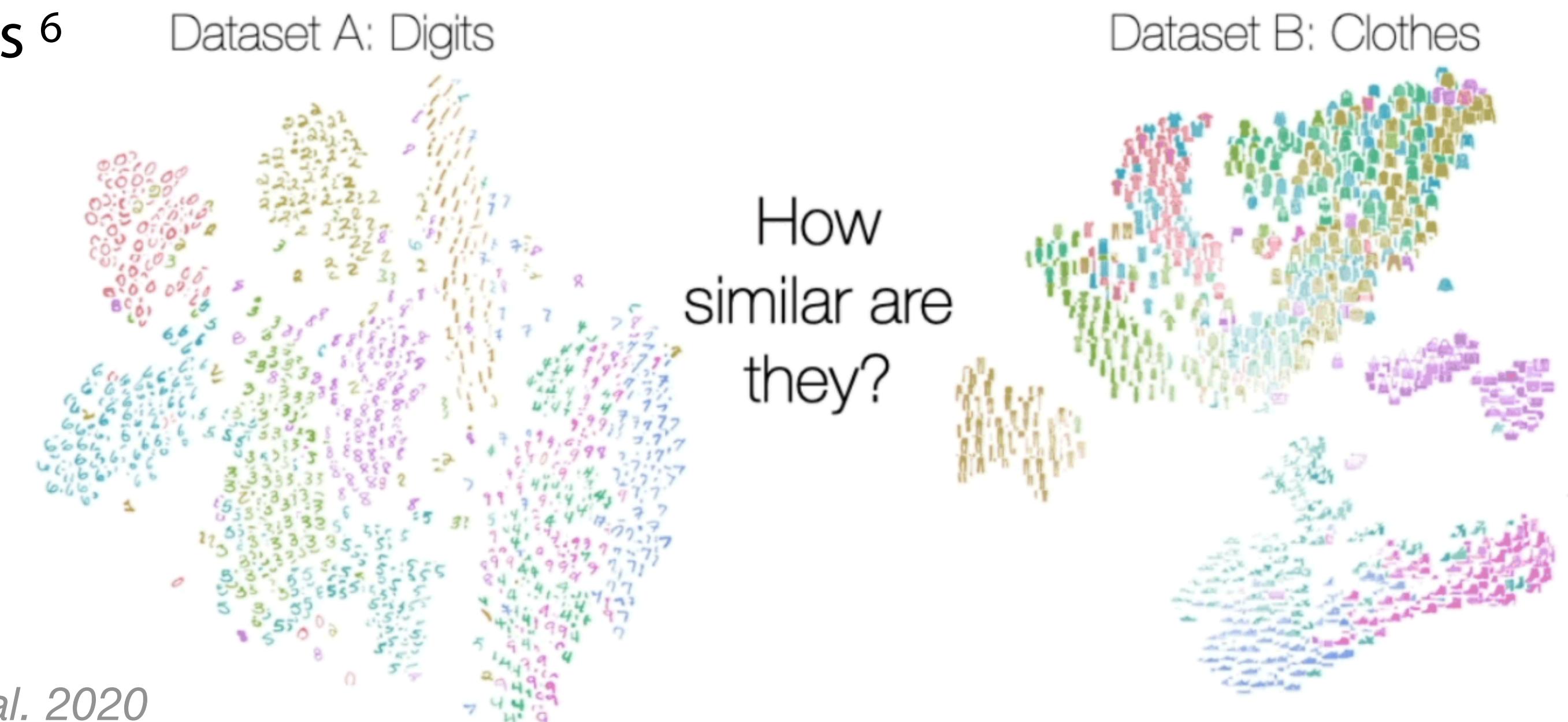
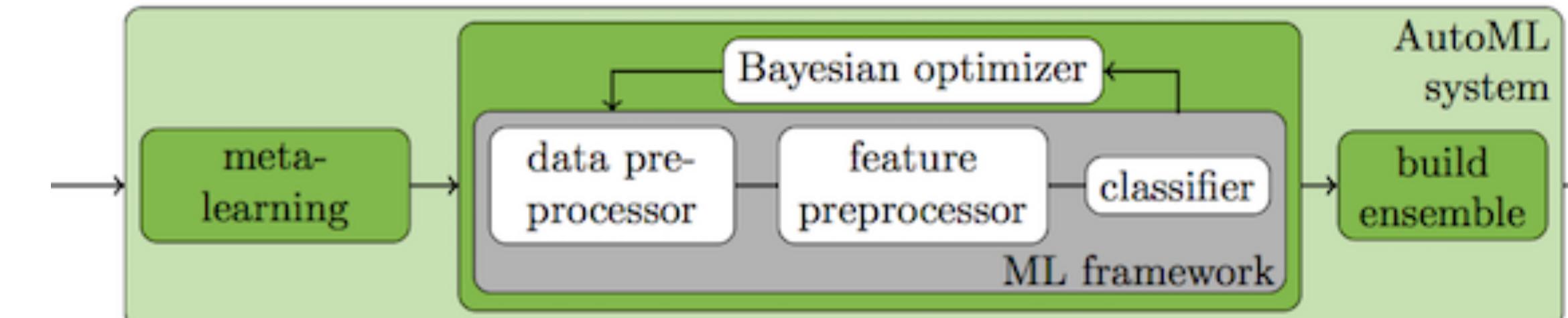


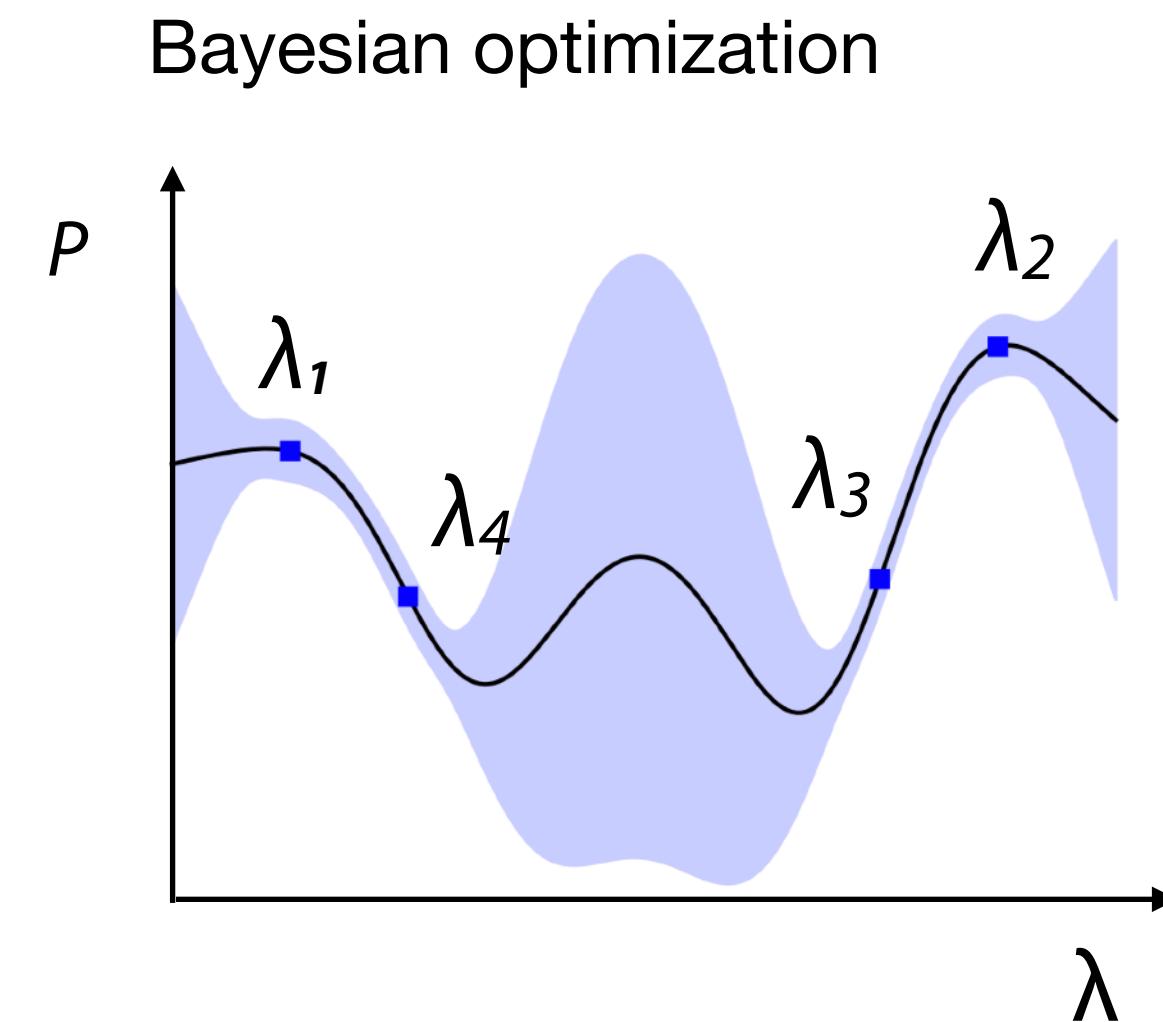
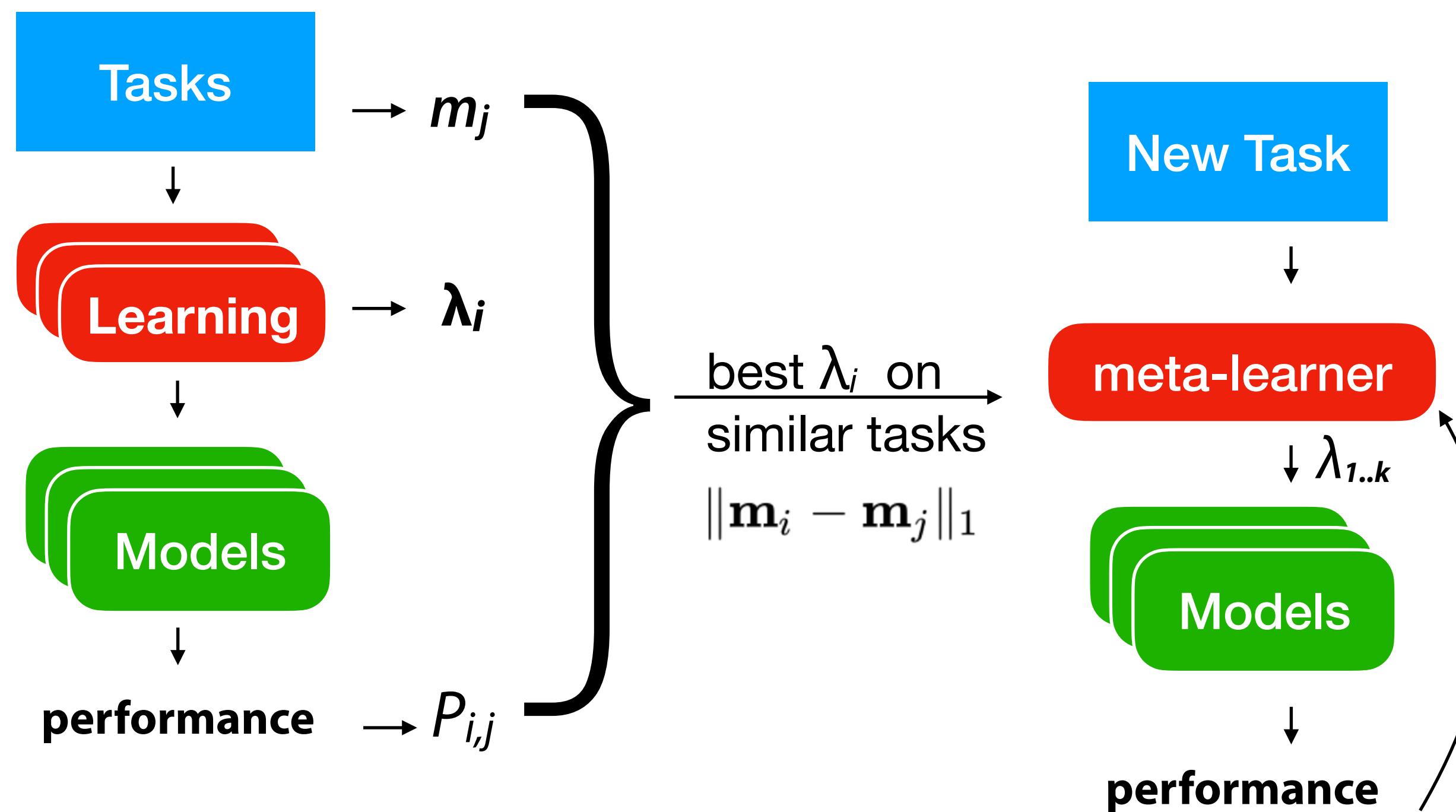
Figure source: Alvarez-Melis et al. 2020

# Warm-starting with kNN

- 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 several AutoML Challenges



*Figure source: Feurer et al., 2015*



# Probabilistic Matrix Factorization

- Collaborative filtering: configurations  $\lambda_i$  are ‘rated’ by tasks  $t_j$
- Learn latent representation for tasks  $T$  and configurations  $\lambda$
- Use meta-features to warm-start on new task
- Returns probabilistic predictions for Bayesian optimization
- Used in Azure AutoML

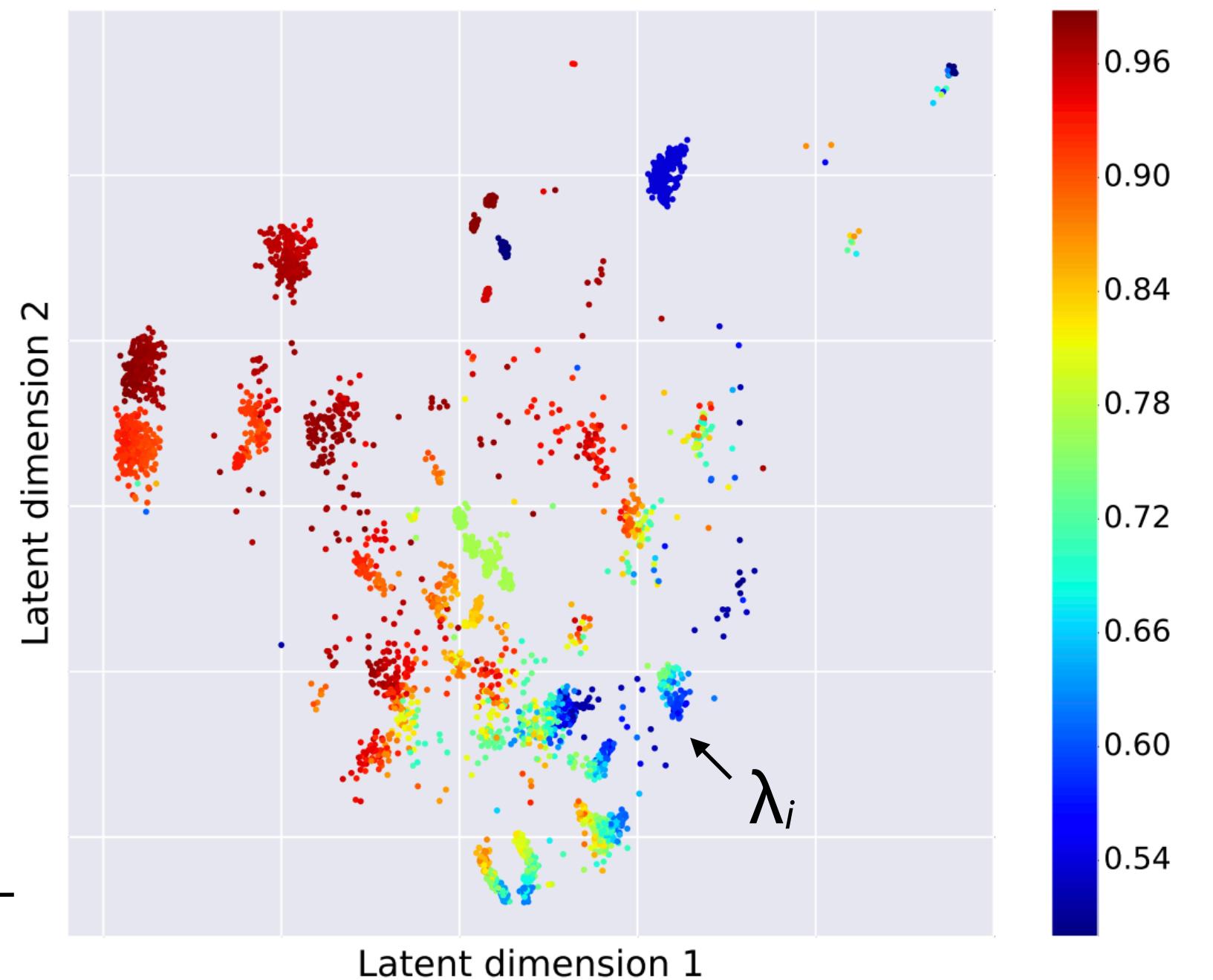
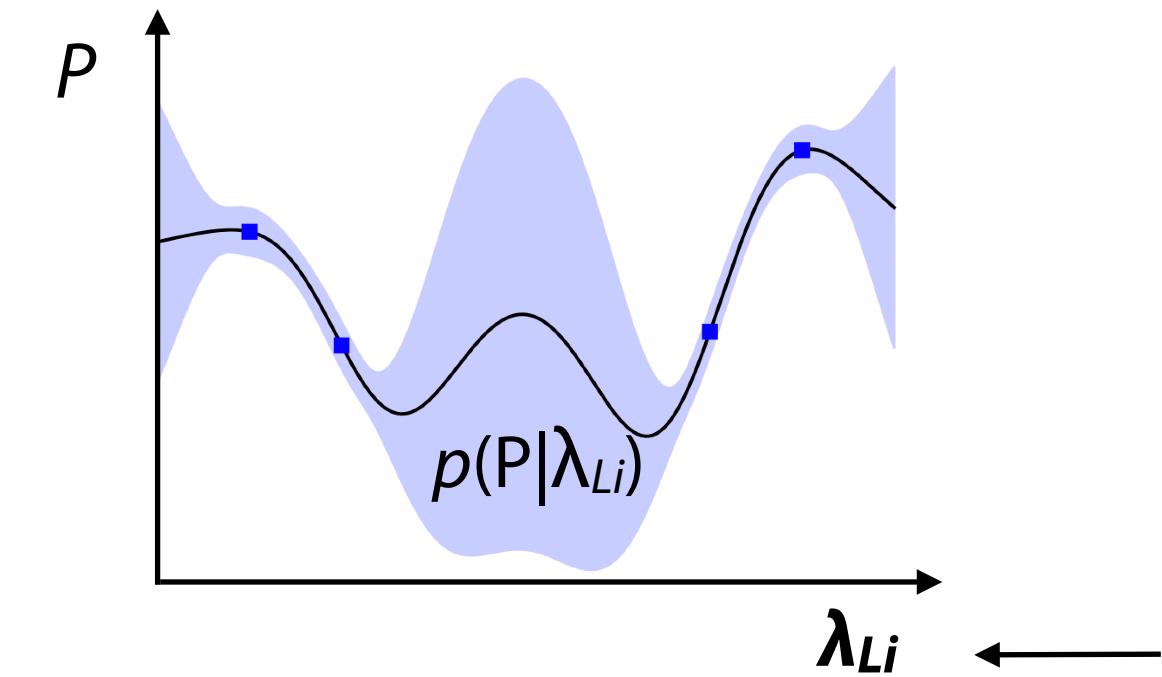
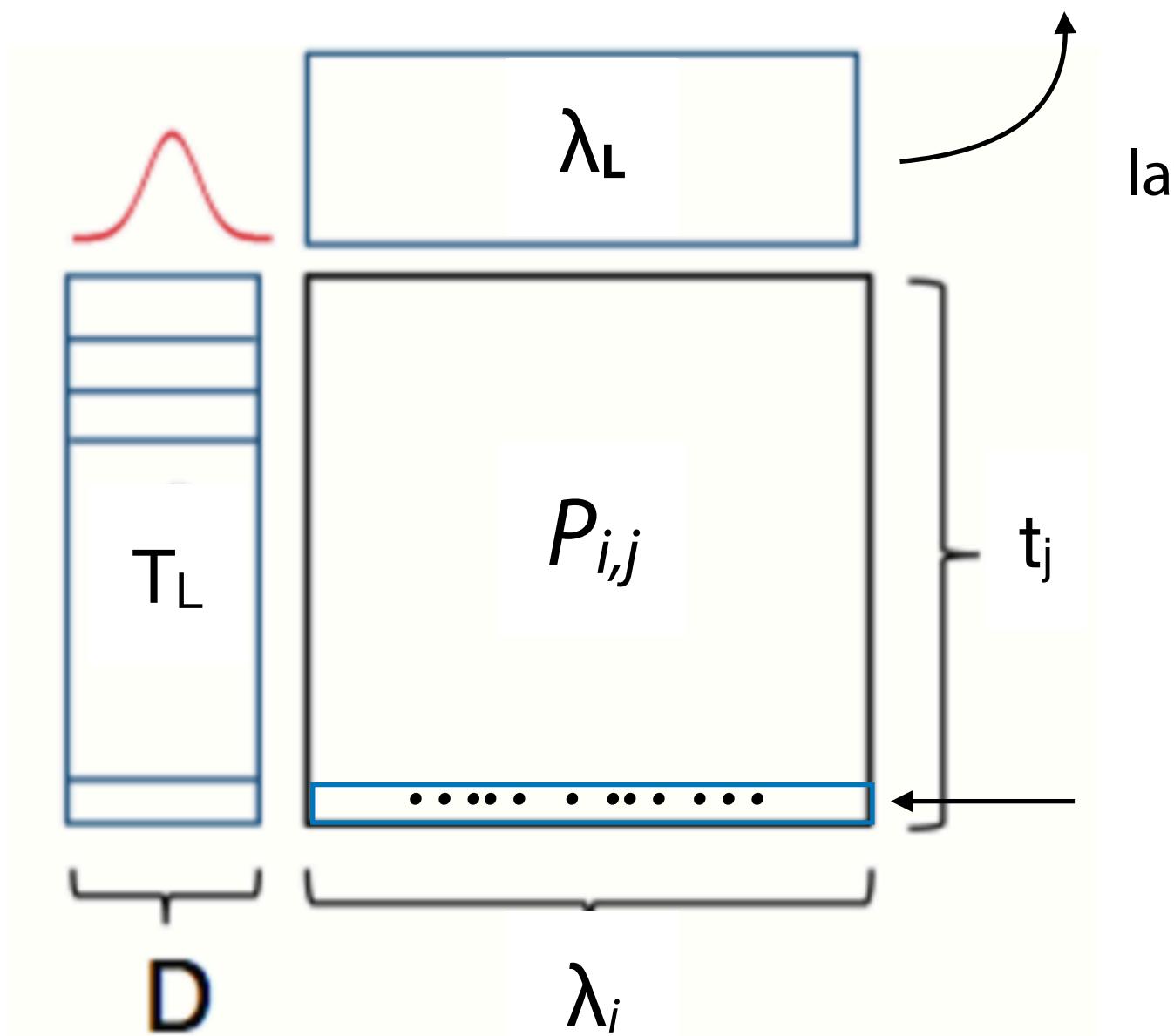
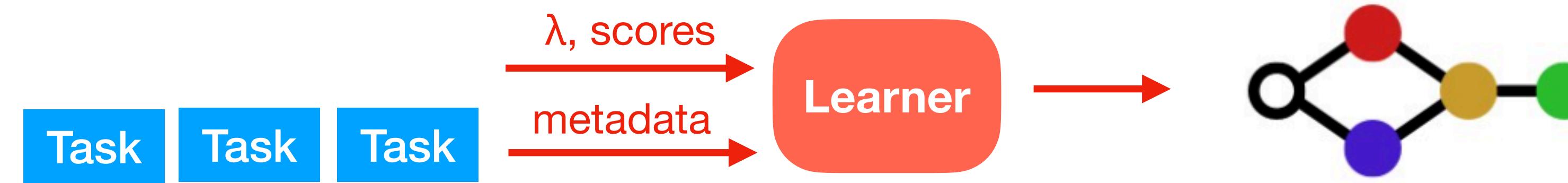


Figure source: Fusi et al., 2017



# Meta-models

(learn how to build models/components)



# Algorithm selection models

- Learn direct mapping between meta-features and  $P_{i,j}$ 
  - Zero-shot meta-models: predict best  $\lambda_i$  given meta-features <sup>1</sup>

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

- Ranking models: return ranking  $\lambda_{1..k}$  <sup>2</sup>

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

- Predict which algorithms / configurations to consider / tune <sup>3</sup>

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

- Predict performance / runtime for given  $\Theta_i$  and task <sup>4</sup>

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

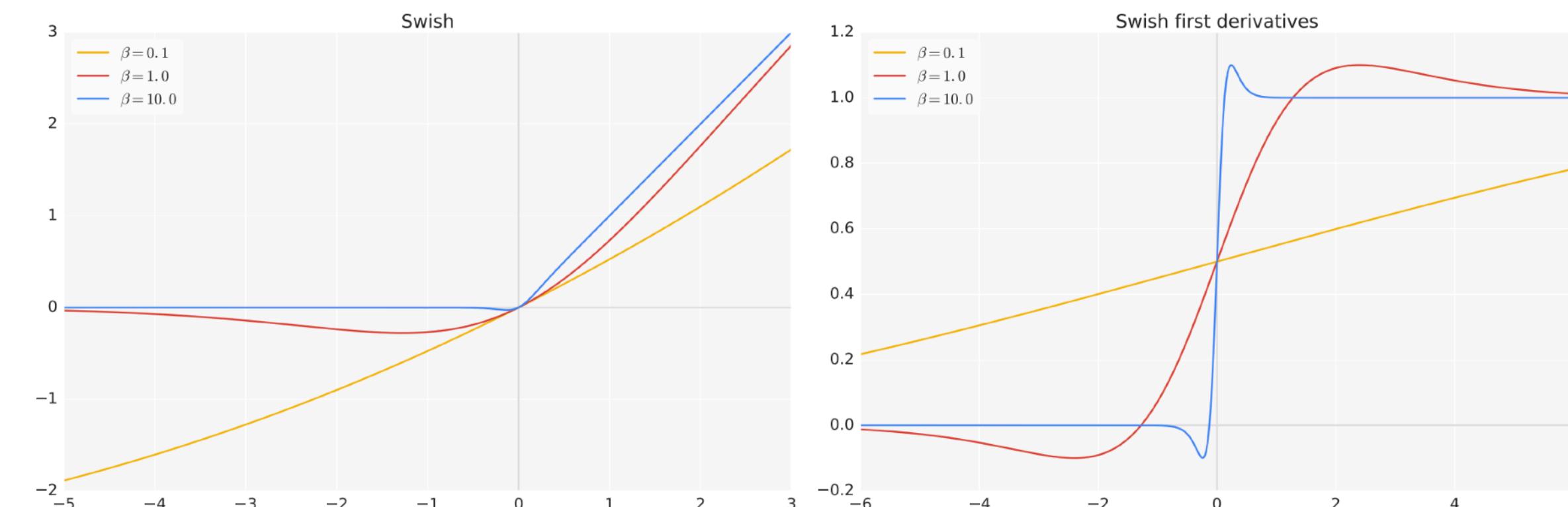
- Can be integrated in larger AutoML systems: warm start, guide search,...

# Learning model components

- Learn nonlinearities: RL-based search of space of likely useful activation functions <sup>1</sup>

- E.g. *Swish* can outperform ReLU

$$\text{Swish} : \frac{x}{1 + e^{-\beta x}}$$

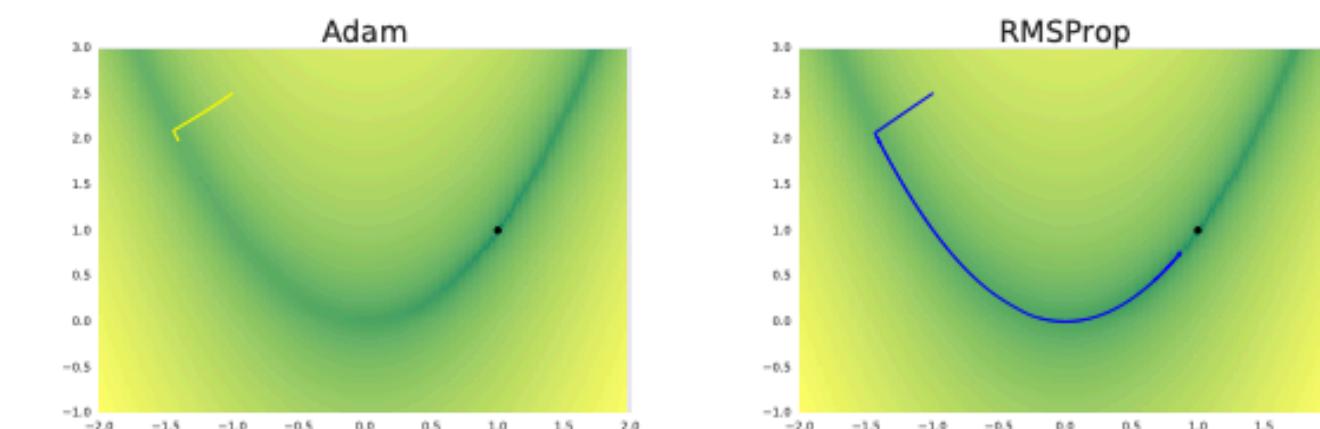


- Learn optimizers: RL-based search of space of likely useful update rules <sup>2</sup>

- E.g. PowerSign can outperform Adam, RMSprop

$$\text{PowerSign} : e^{\text{sign}(g)\text{sign}(m)} g$$

*g: gradient, m:moving average*

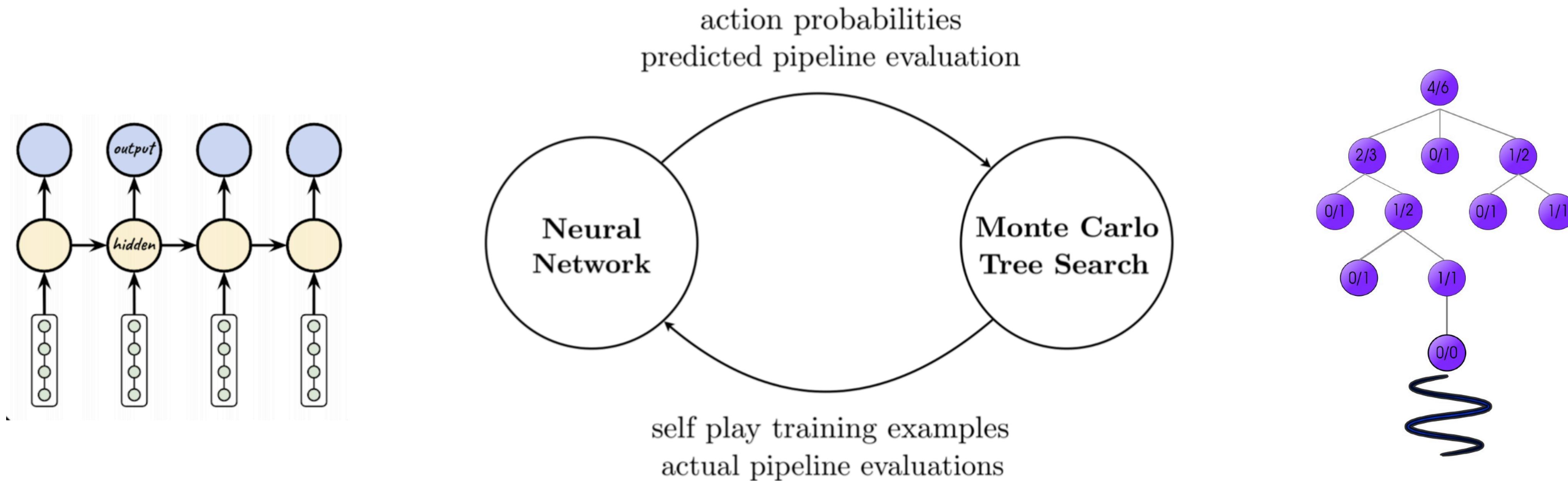


- Learn acquisition functions for Bayesian optimization <sup>3</sup>

# Monte Carlo Tree Search + reinforcement learning

- ***Self-play:***

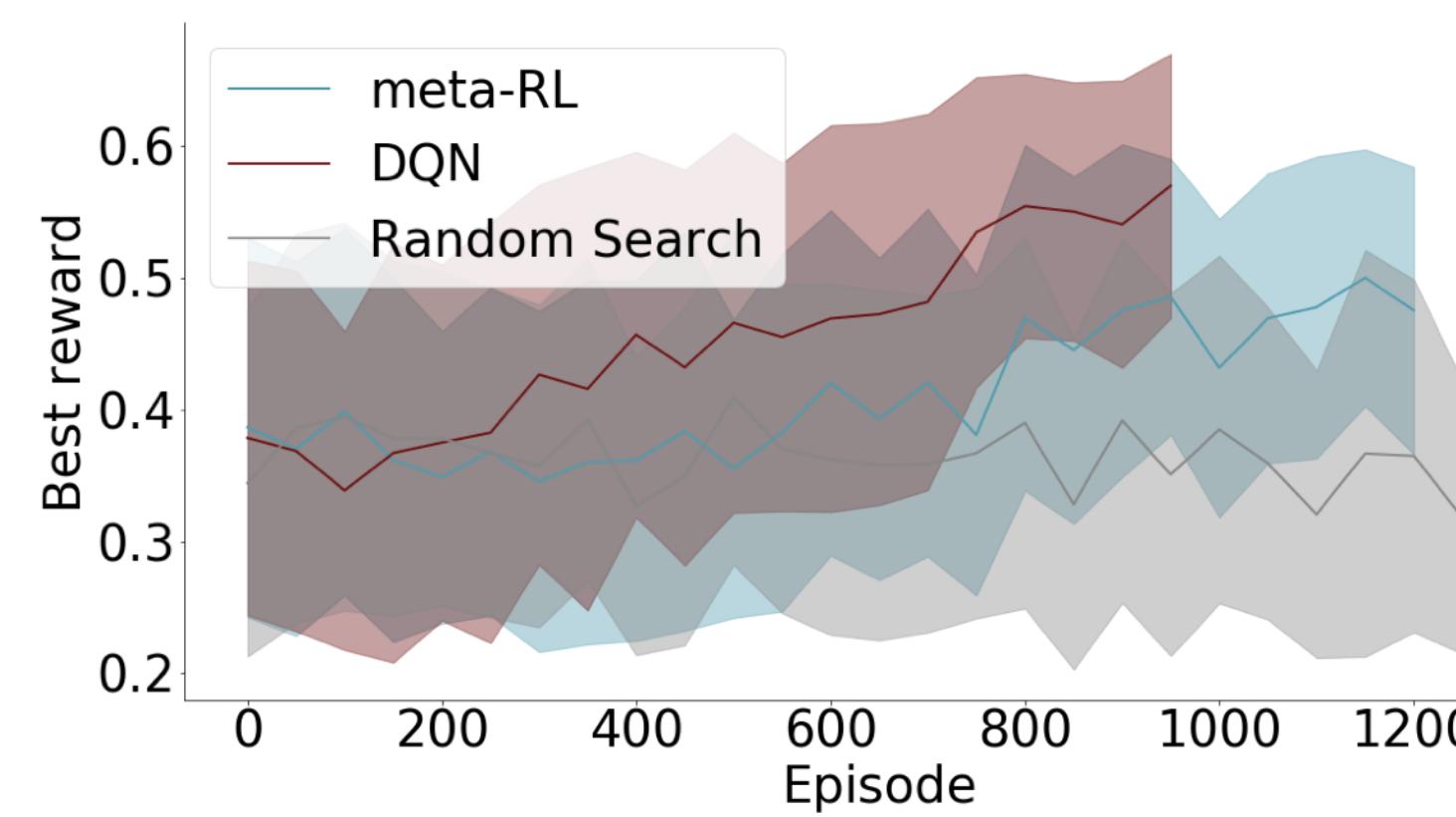
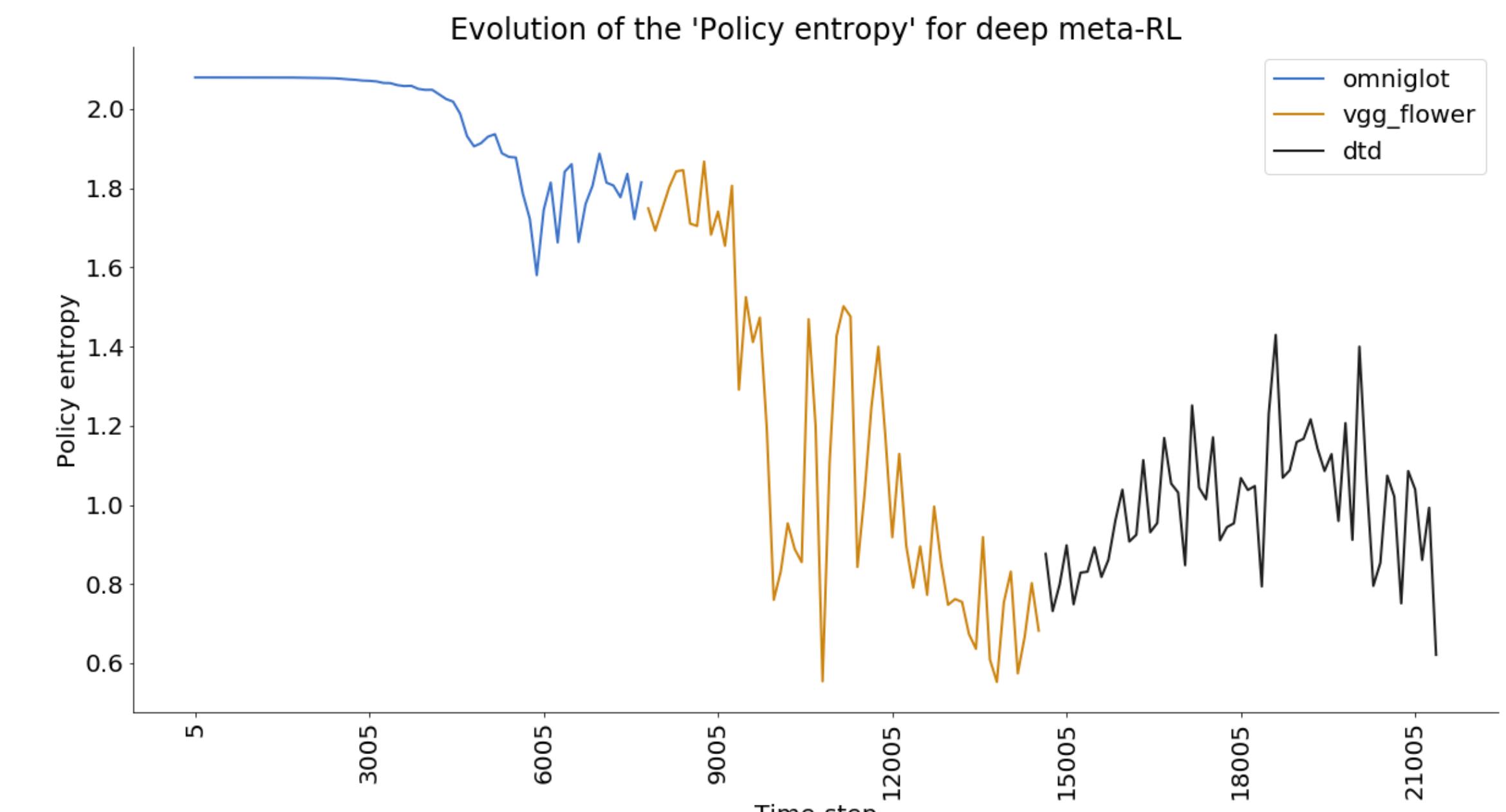
- Game actions: insert, delete, replace components in a pipeline
- Monte Carlo Tree Search builds pipelines given action probabilities
- Neural network (LSTM) Predicts pipeline performance



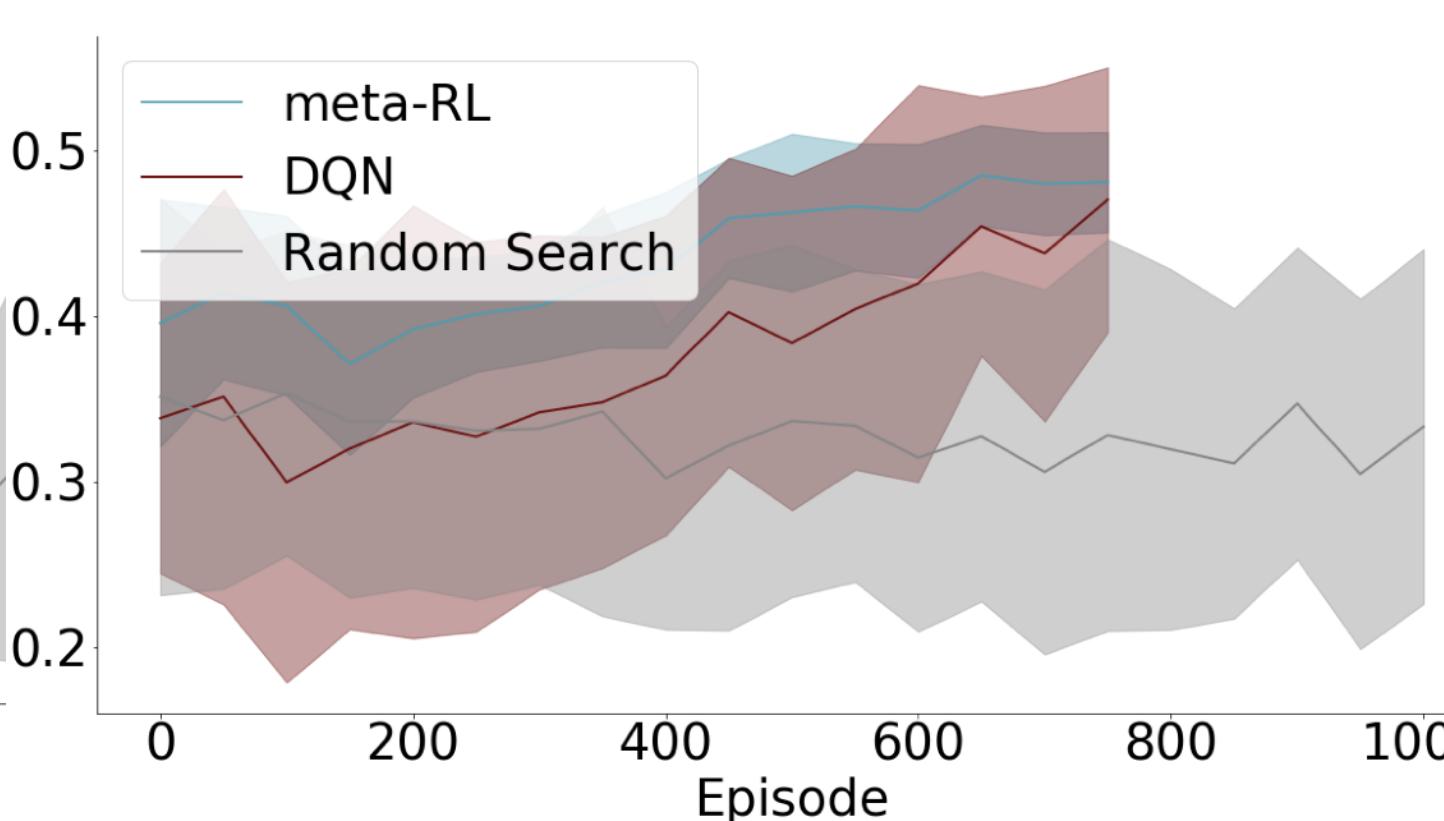
# Meta-Reinforcement Learning for NAS

Results on increasingly difficult tasks:

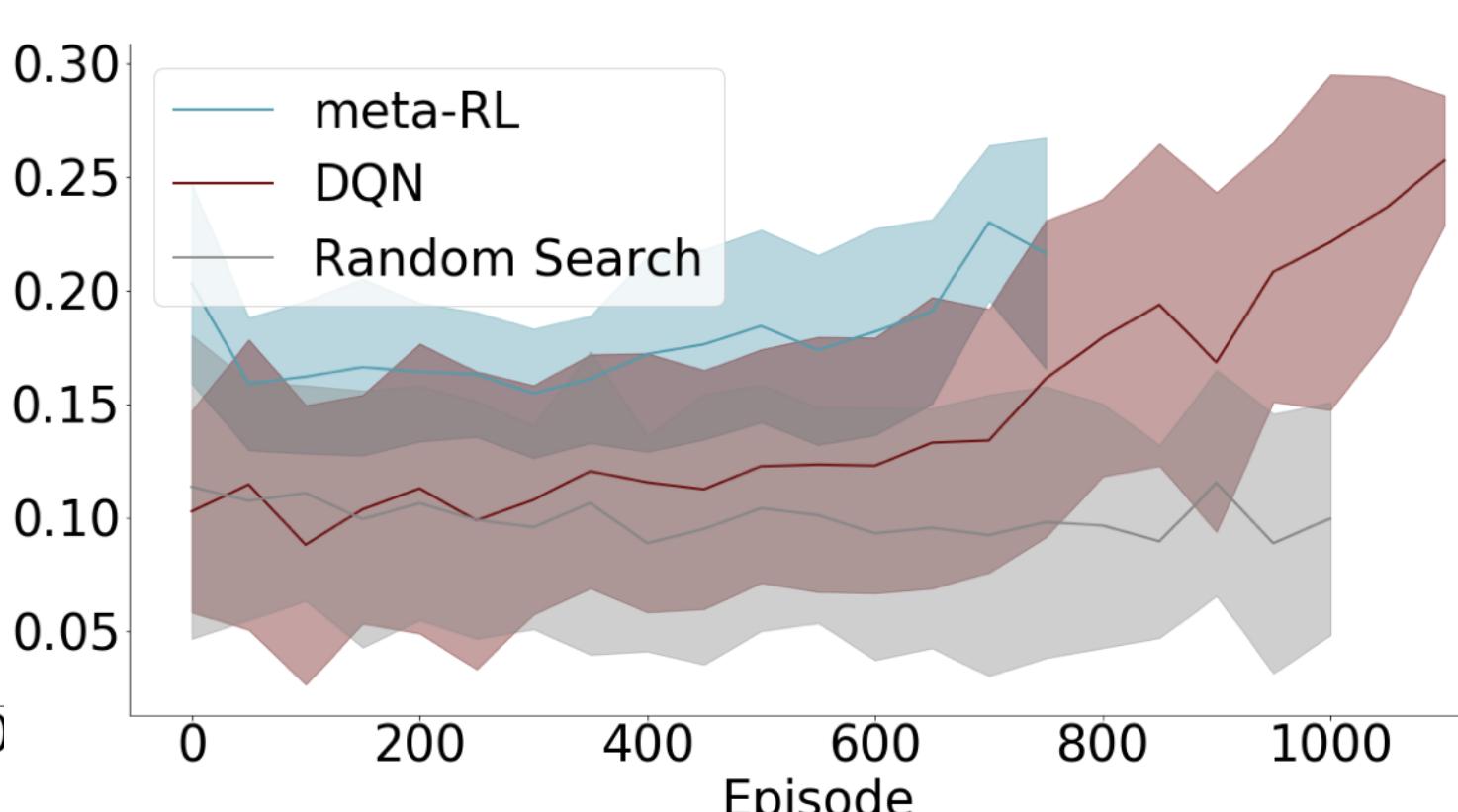
- Initially slower than DQN, but faster after a few tasks
- Policy entropy shows learning/re-learning



omniglot



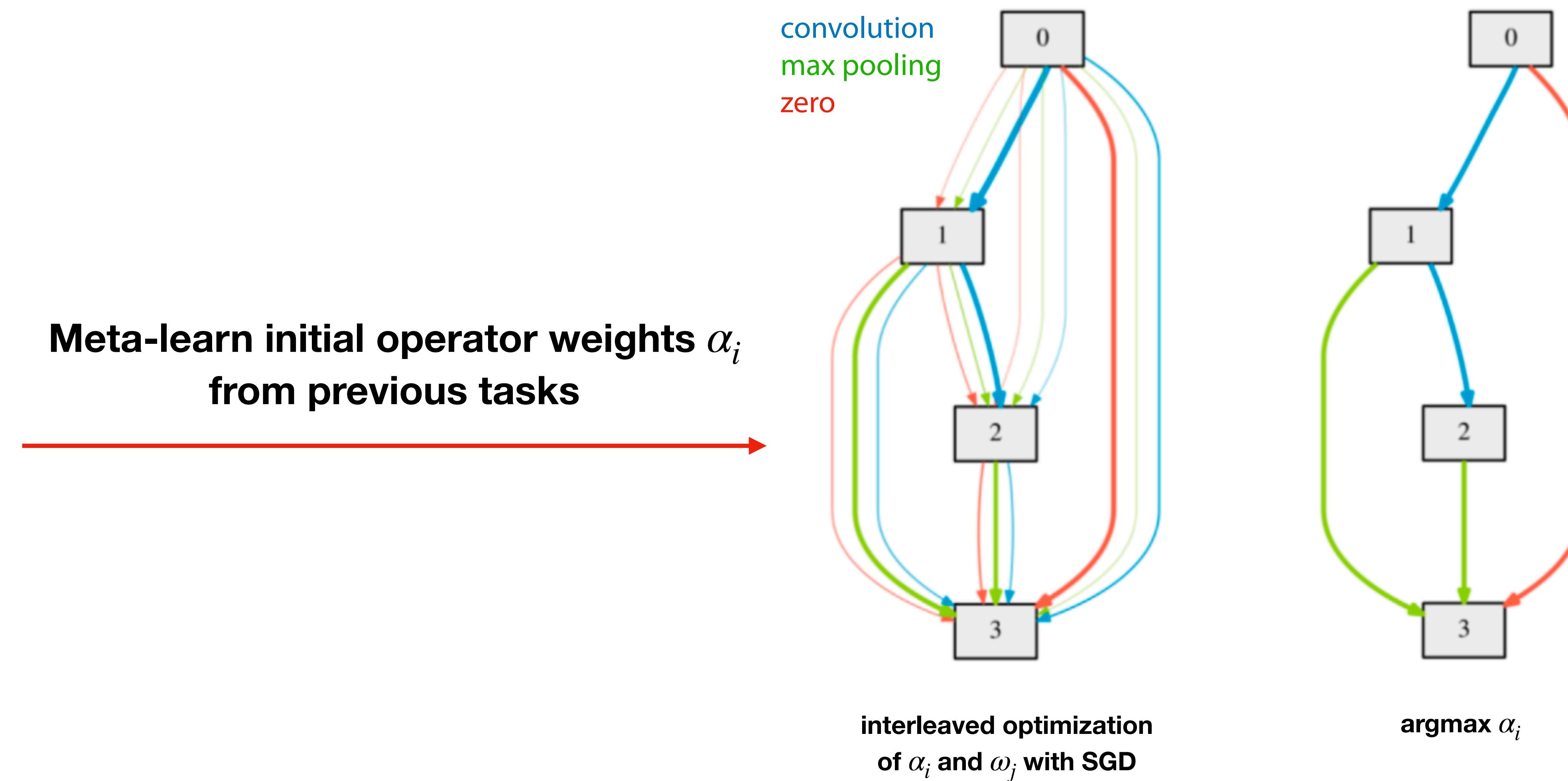
vgg\_flower



dtd

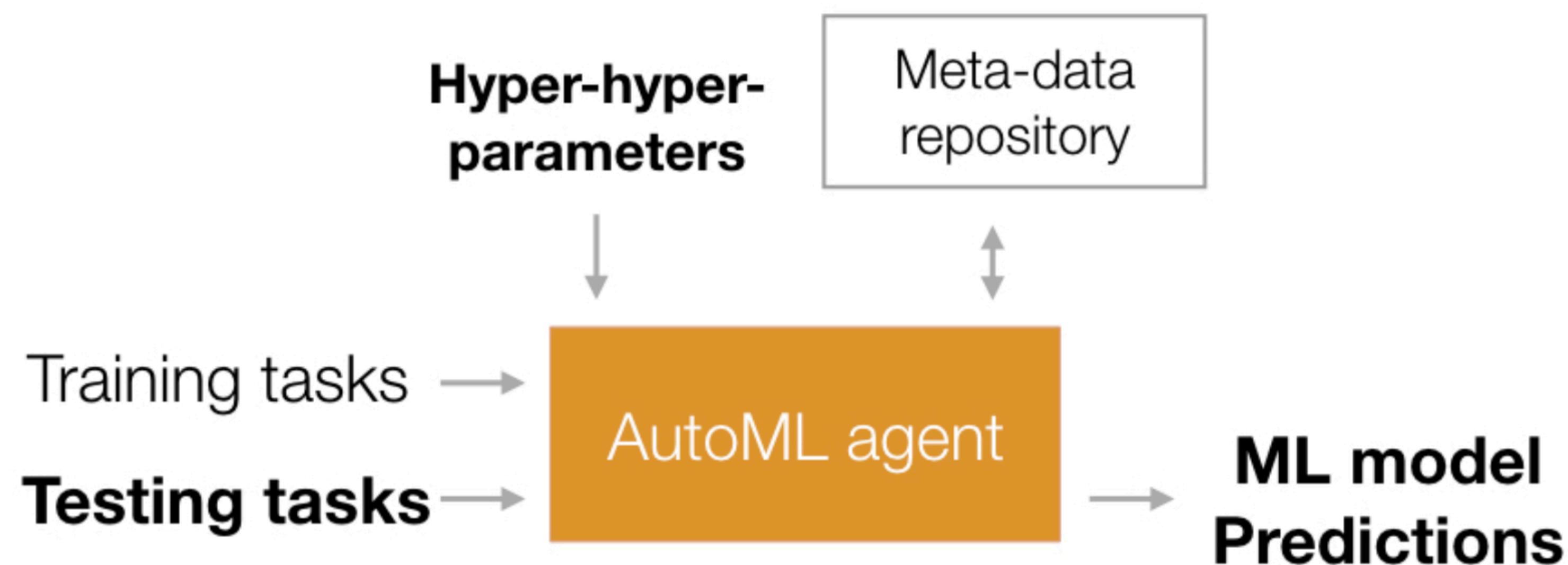
# MetaNAS: meta-learning + NAS

- Use meta-learning (MAML) to learn a good weight initialization for the network

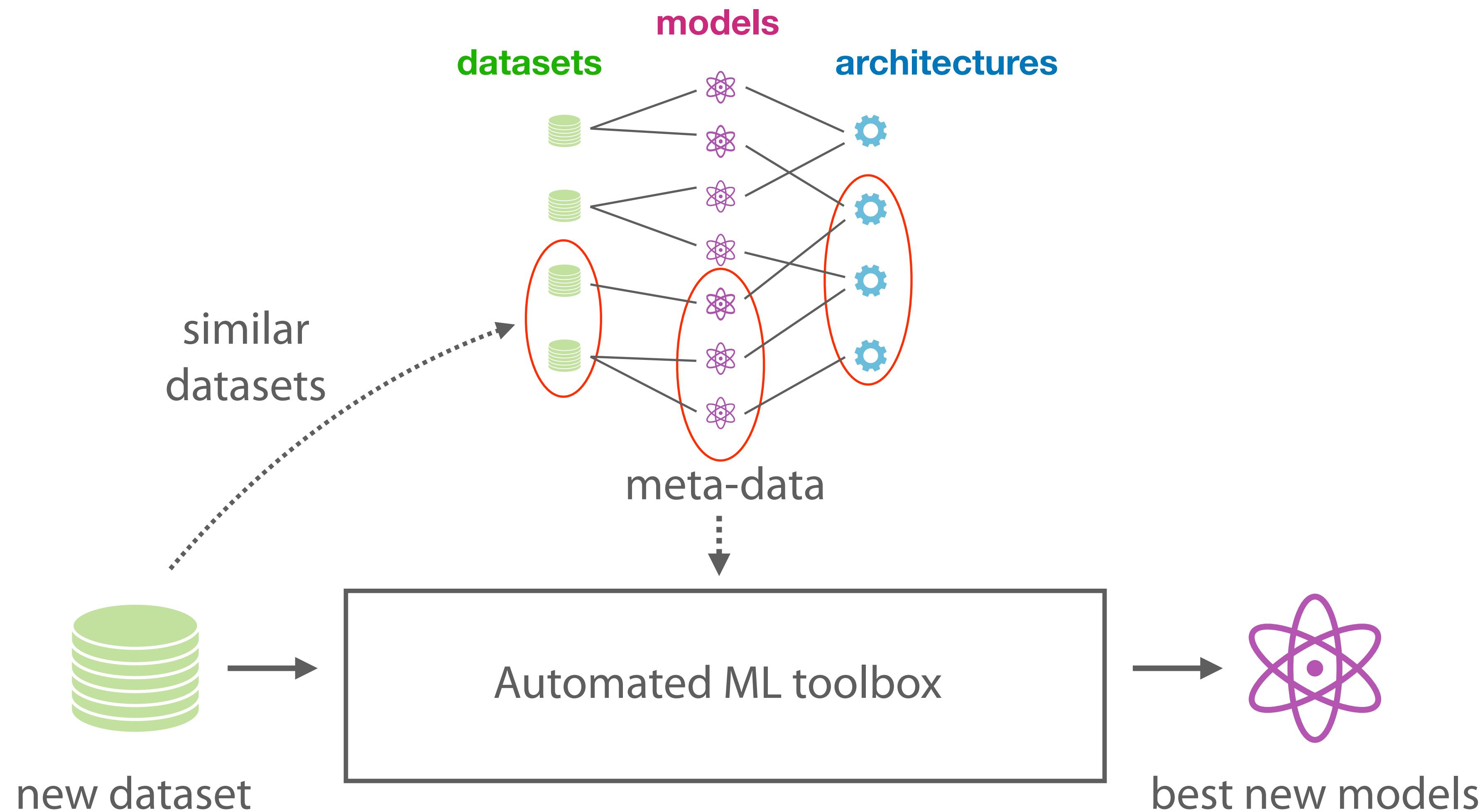


# Meta-learning AutoML in practice

- We need a meta-data repository of prior machine learning datasets (tasks) and experiments
  - e.g. [OpenML.org](#)
  - Ideally, a *shared* memory that all AutoML tools can access



# Meta-learning with OpenML



# AutoML open source tools

	Architect. search	Operators	Hyperpar. search	Improvements	Metalearning
<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>H2O AutoML</u>	Param. pipeline	H2O	Random search	Stacking	
AutoGluon-Tabular	Param. pipeline	Sagemaker	Random search	multi-level 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		
TensorFlow 2	/	keras	RS or HB		
Talos	/	keras	RS variants		

*Many other tools for hyperparameter optimization alone*

# *Further reading*

Open access book

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

