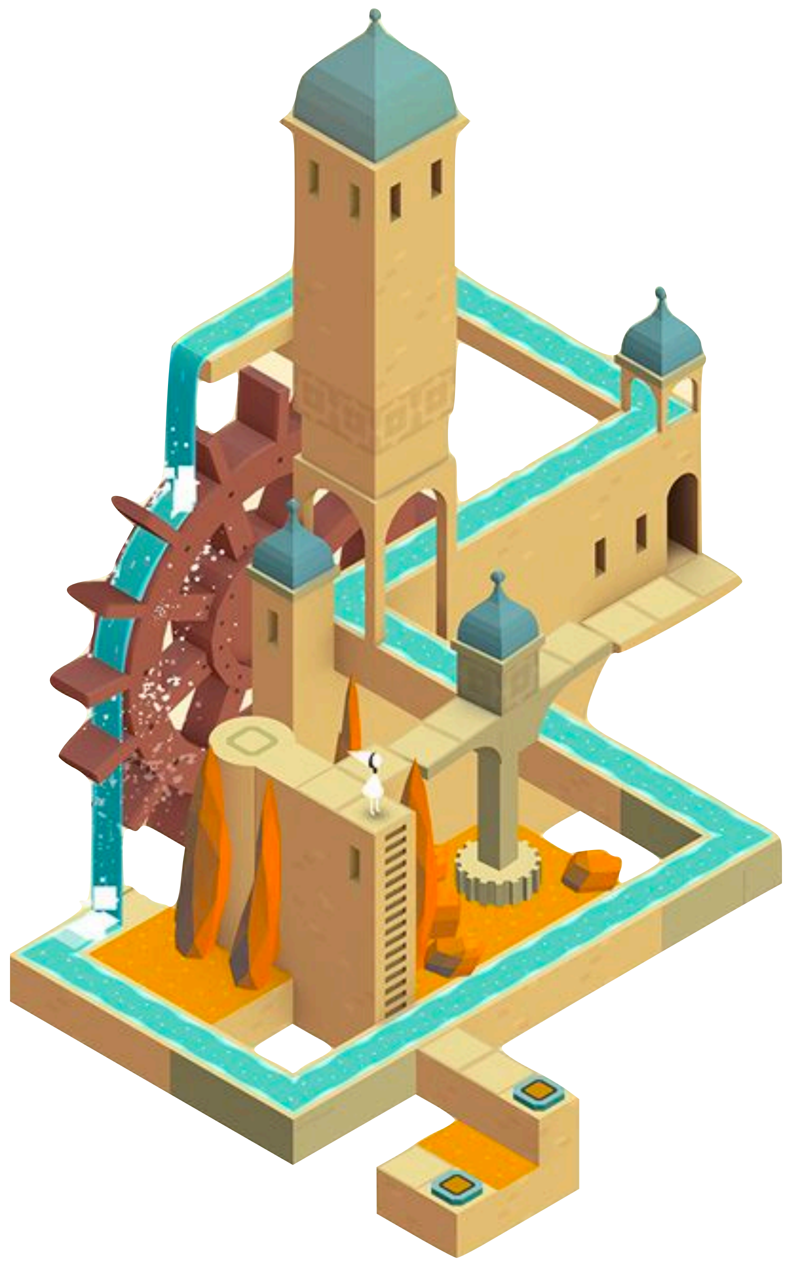


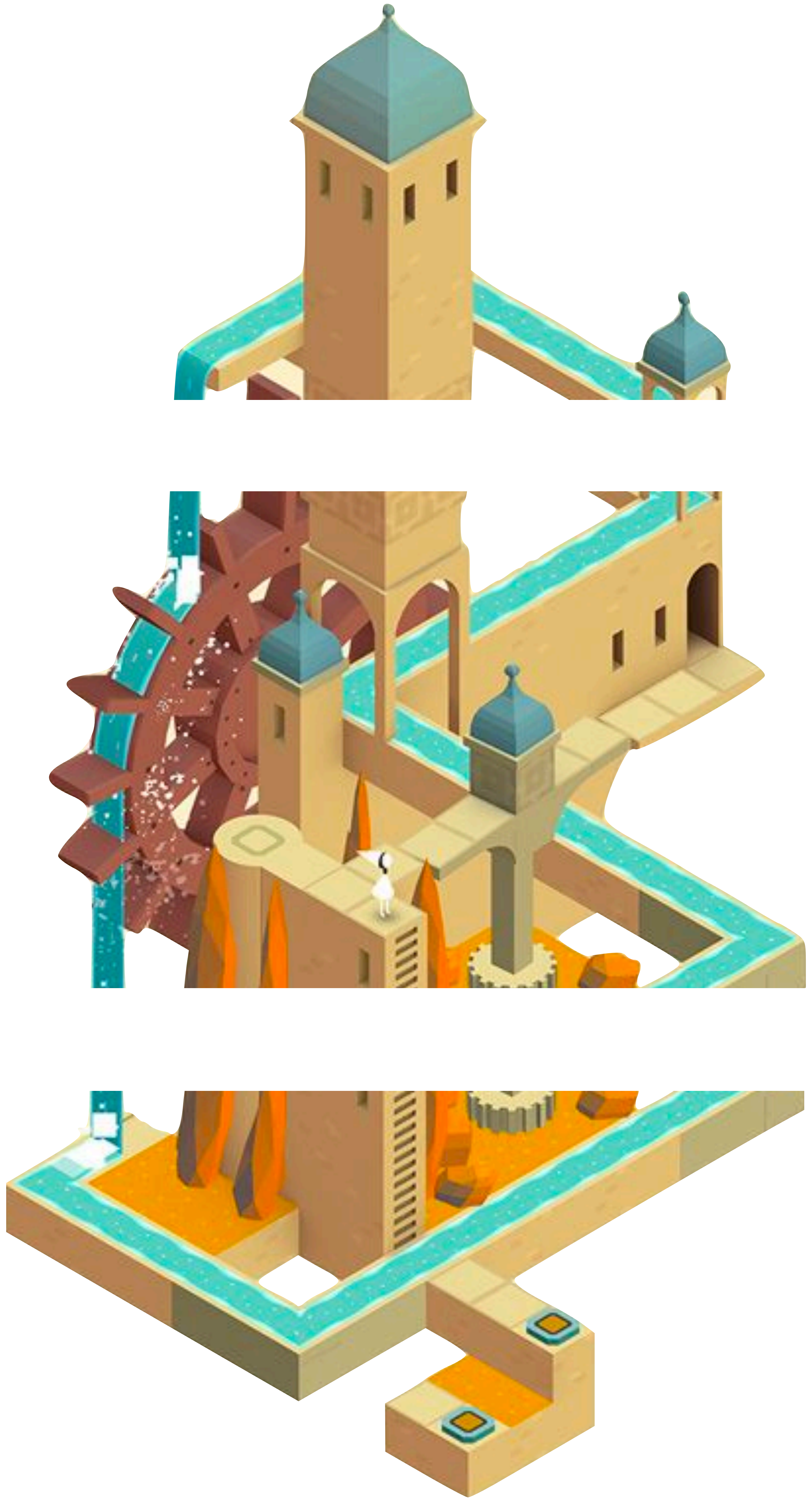
Automated Machine Learning

2 December, 2021

Joaquin Vanschoren
Eindhoven University of Technology

image credit: ustwo

Overview



Part 1: *Why* automate machine learning?

High-level goals

Part 2: *How* AutoML works

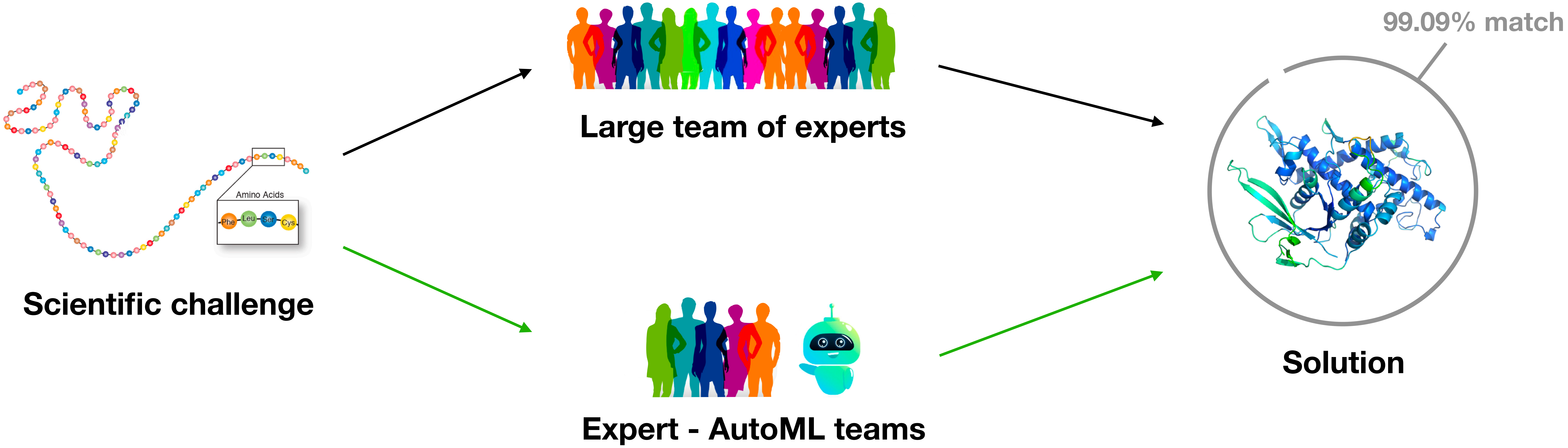
The machinery

Part 3: Learning how to do AutoML

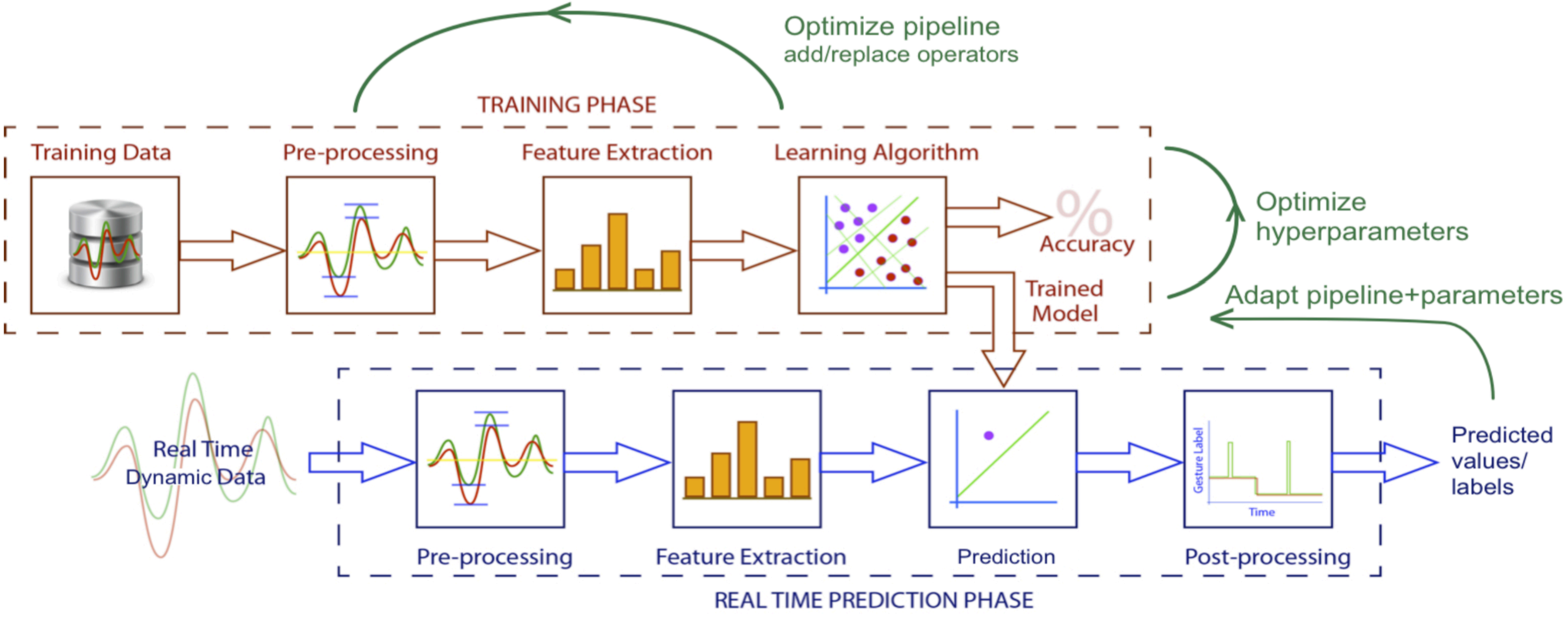
Closing the loop

#1 Democratization of Machine Learning

Allow many more scientists to apply machine learning much more easily



Doing machine learning requires lots of expertise and exploration



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

Why is Machine Learning labor-intensive?

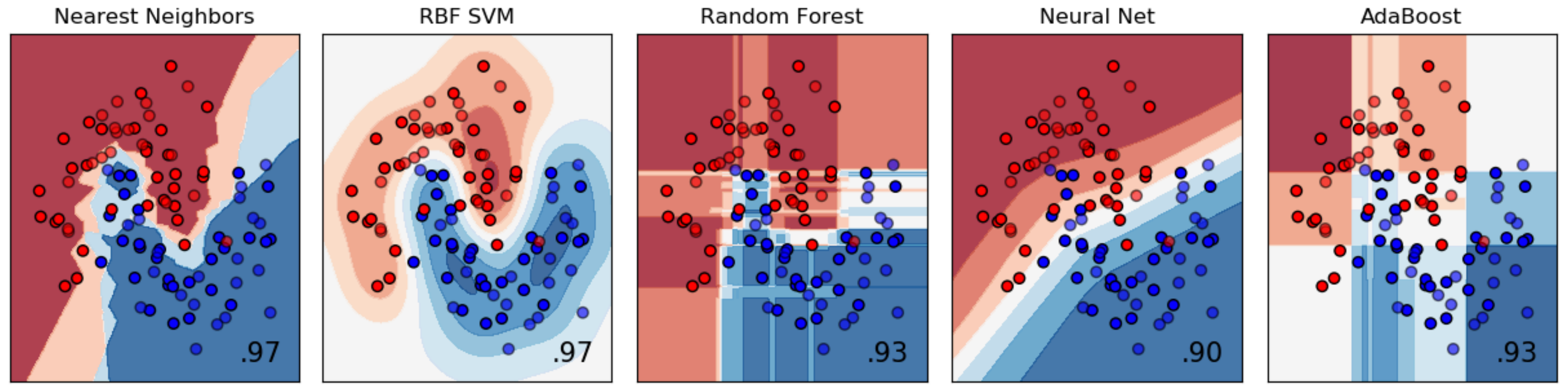
Machine learning pipelines / models have an **infinite** range of possibilities (many still unknown)

Requires implicit knowledge

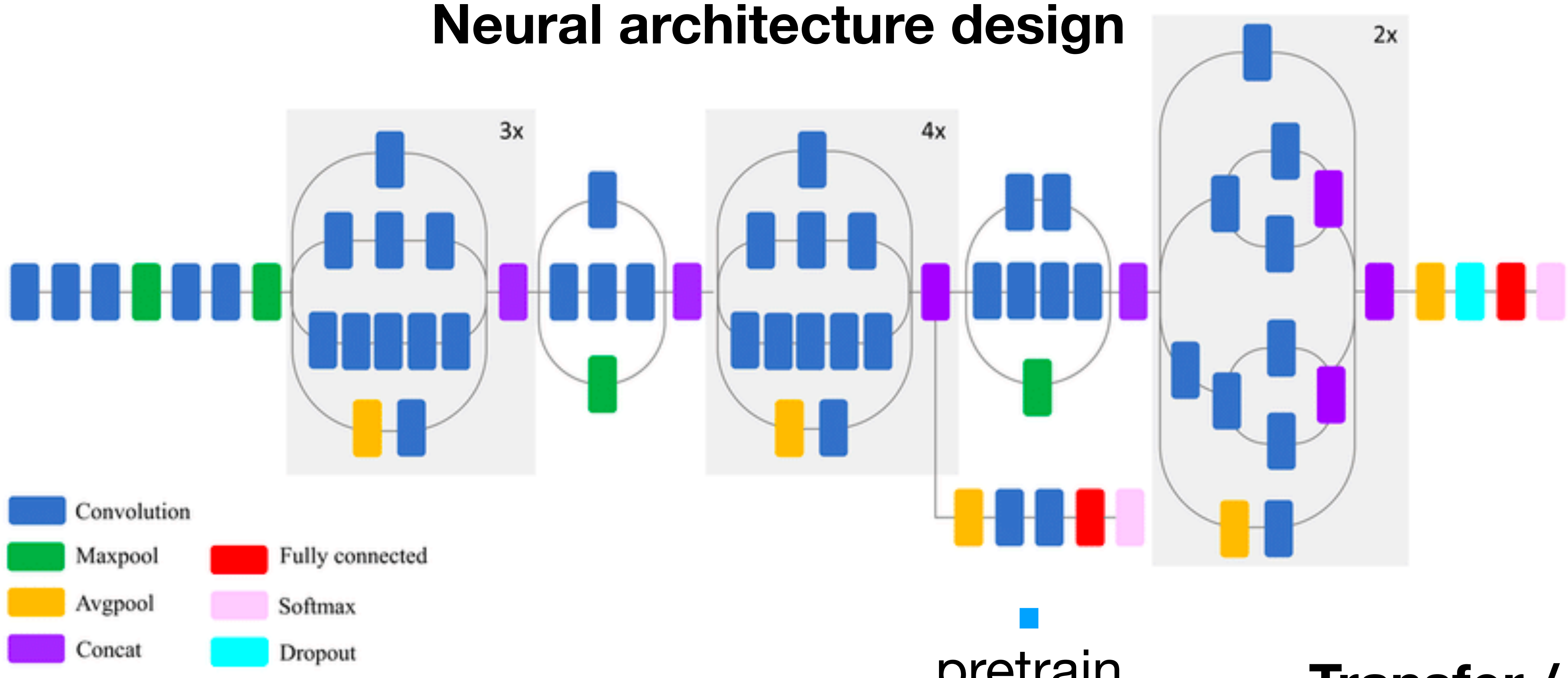
Data

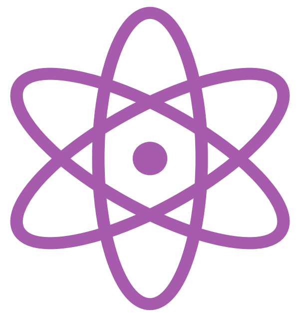
cleaning, preprocessing, featurization, selection,…

Model selection



Neural architecture design



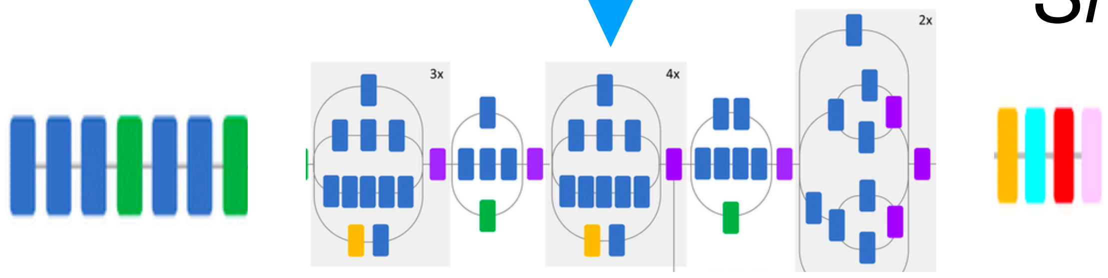


Solution

pretrain

Transfer / continual learning

Small data, few-shot learning

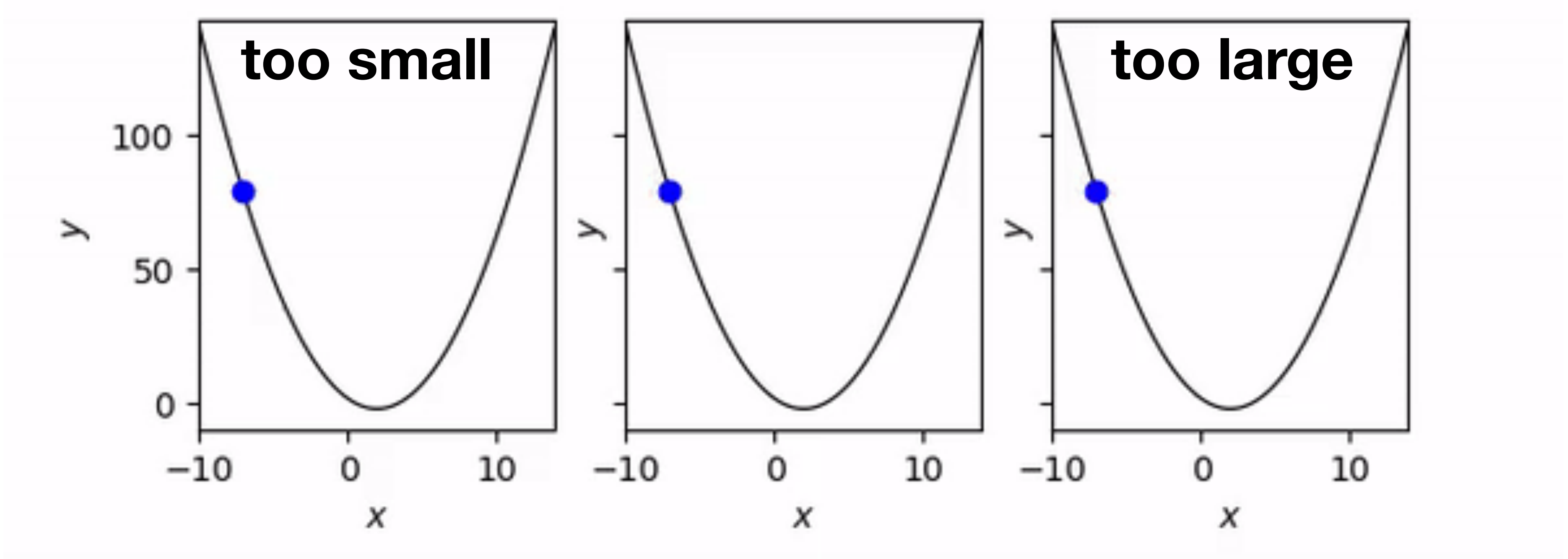


Can we automate this process and share implicit knowledge?

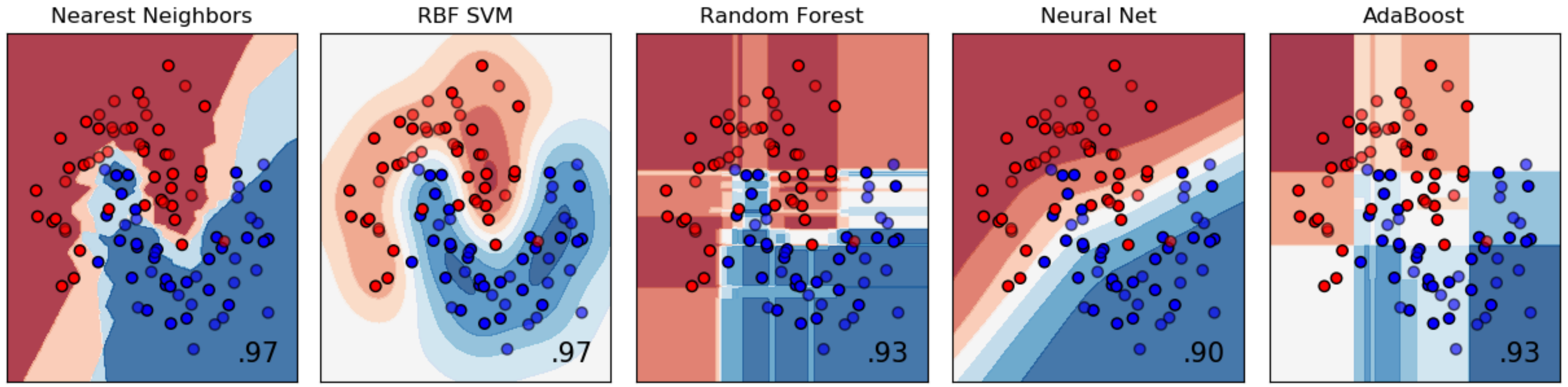
Hyperparameters

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

- Numeric
  - e.g. learning rate

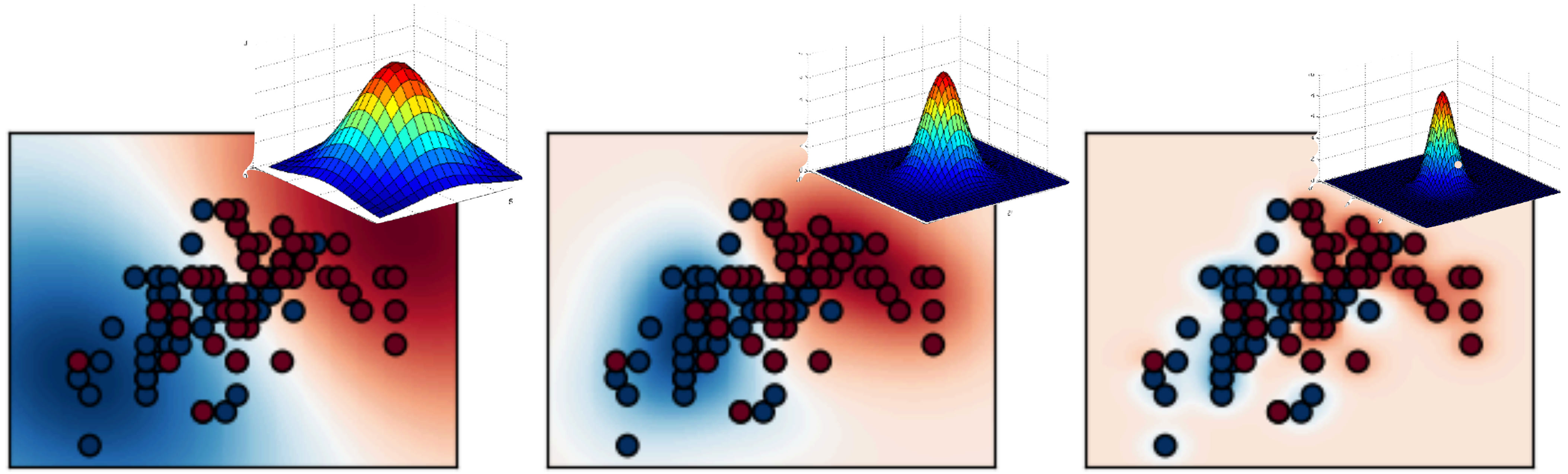


- Categorical
  - e.g. classifier



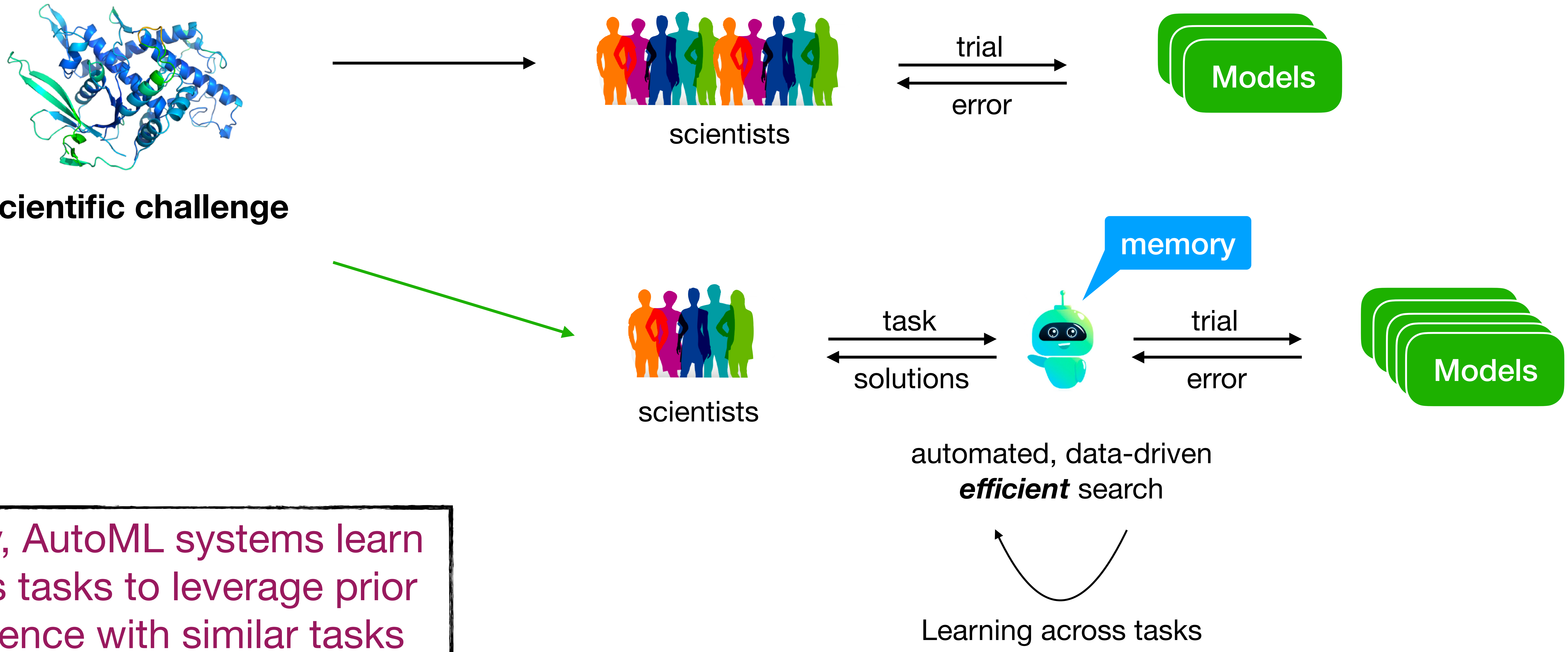
- Conditional
  - SVM -> kernel?

    RBF kernel -> gamma?



Automatic Machine Learning (AutoML)

Replace manual trial and error with automated search (based on prior experience)



Ideally, AutoML systems learn across tasks to leverage prior experience with similar tasks

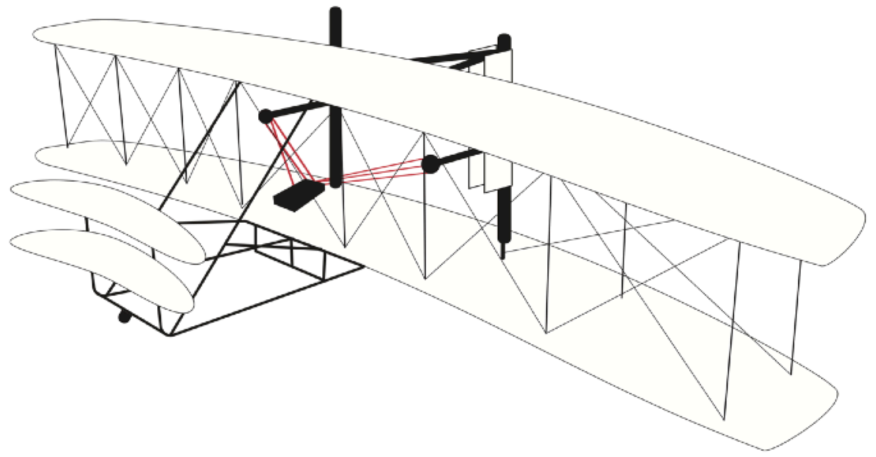
#2 Robust autonomous systems

> Antenna broken. No communication with Earth.

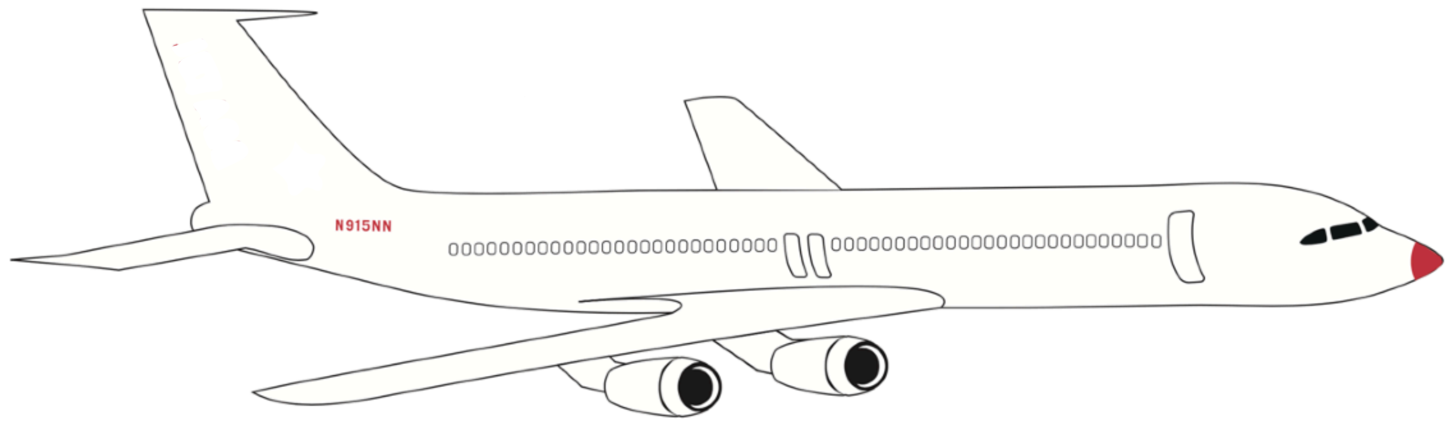
Sigh… I'll have to learn this by myself

A (key) part of the wider AI challenge

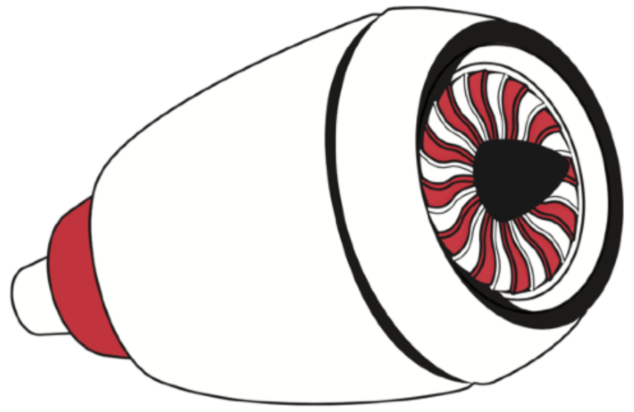
We’re only in the pioneering age



1903: first powered controlled flight



1910: cultural acceptance, only for elites



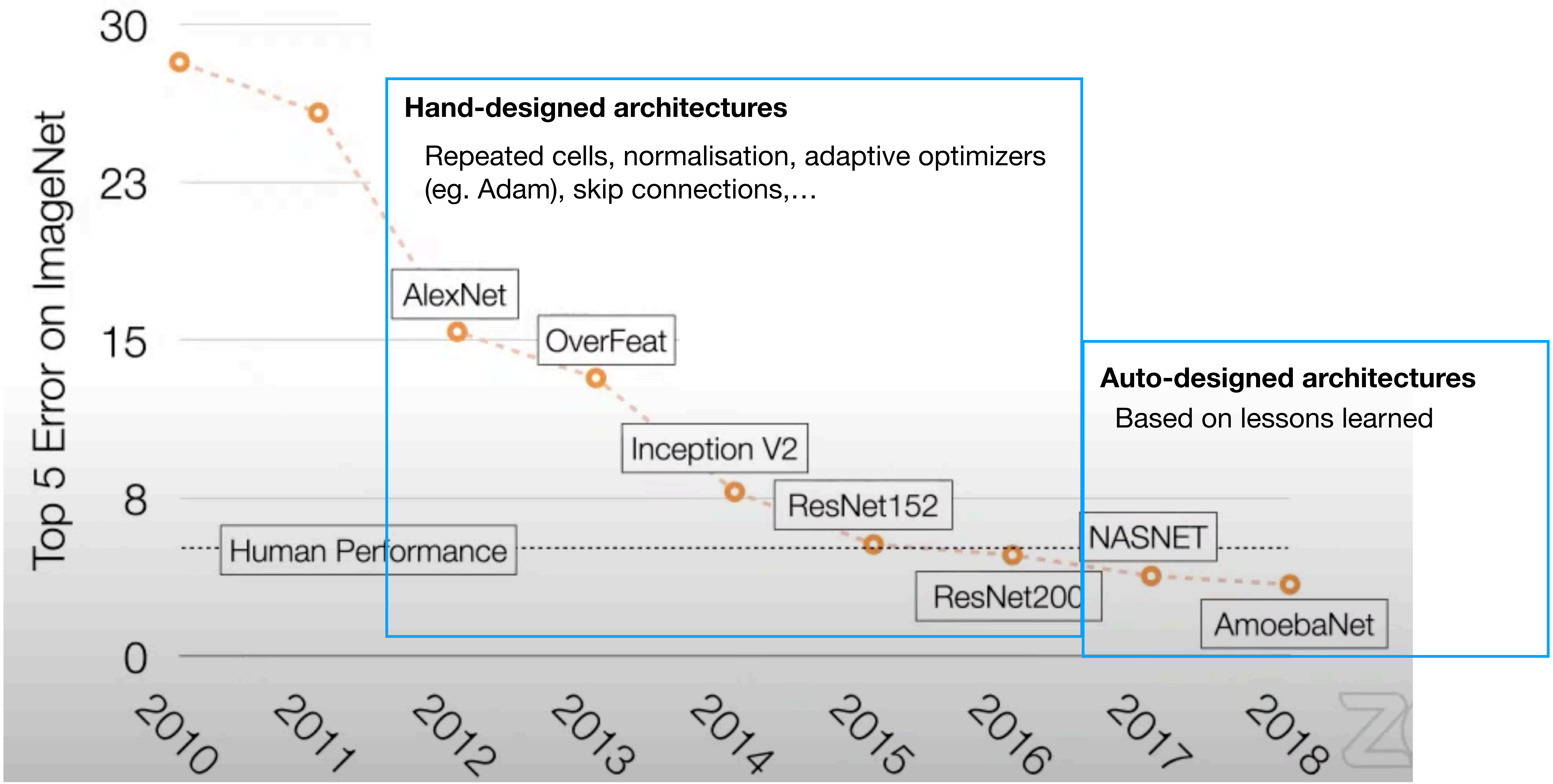
1952: democratized flight

Efficiency, automation, safety
(Science + engineering)

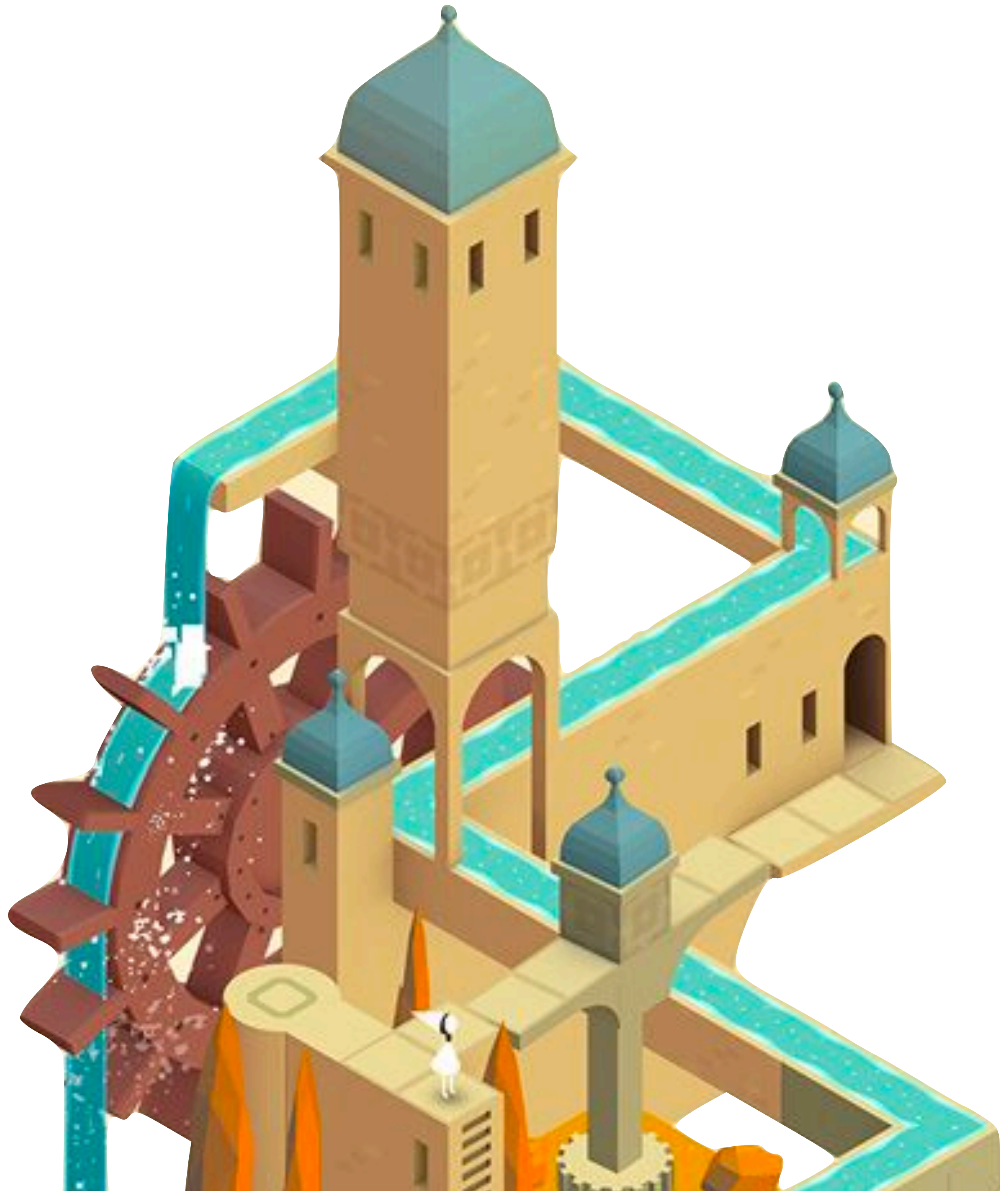
Many challenges remain to democratize AI

- Efficiency: efficient algorithms (transfer, continual), hardware
- Automation: efficient, adaptive AutoML
- Safety: explainability, fairness, causal analysis
- Human-in-the-loop, domain knowledge, real-world constraints

What drove progress?



Overview

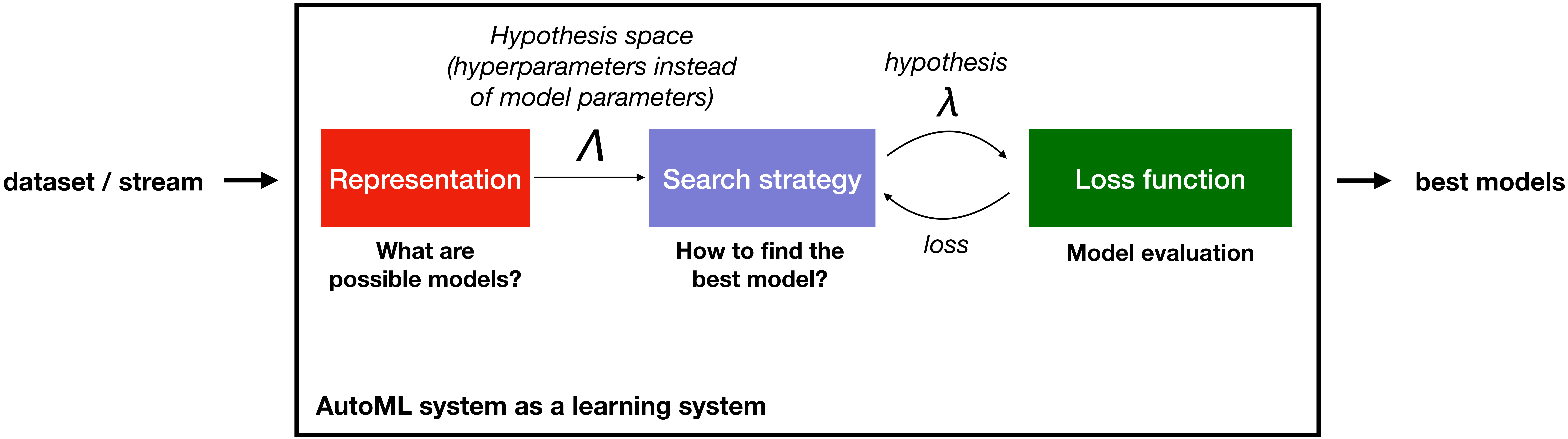


Part 2: *How* AutoML works

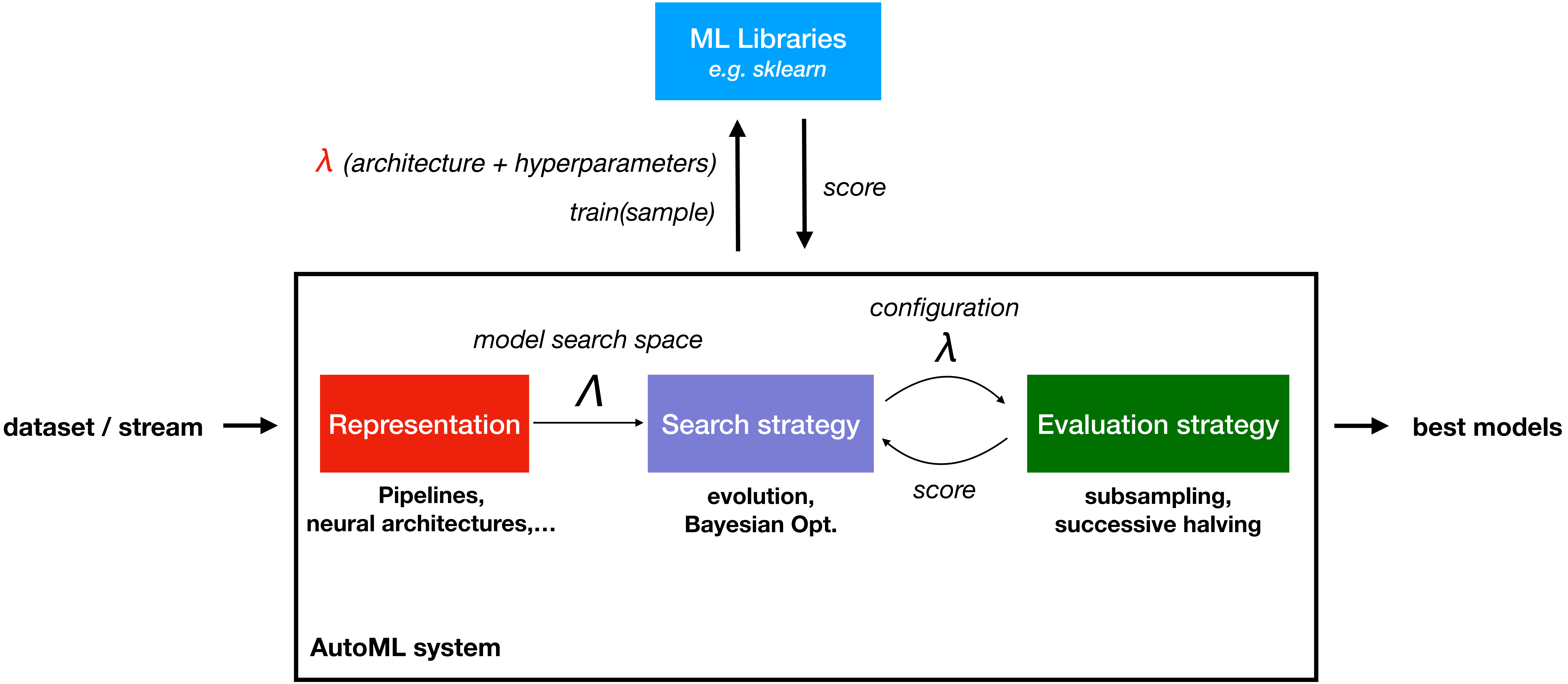
The machinery

Structure of AutoML systems

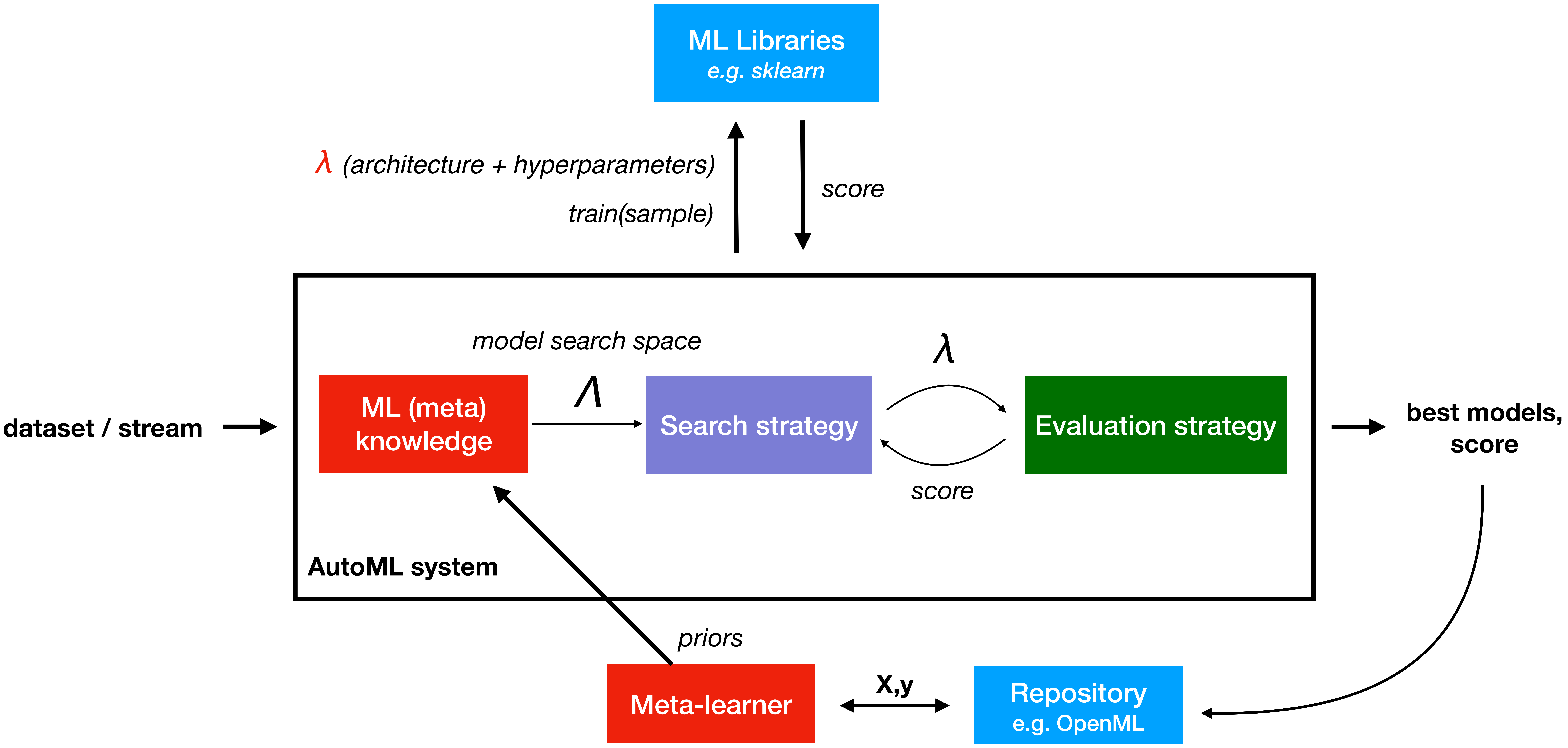
Structure of AutoML systems



Structure of AutoML systems

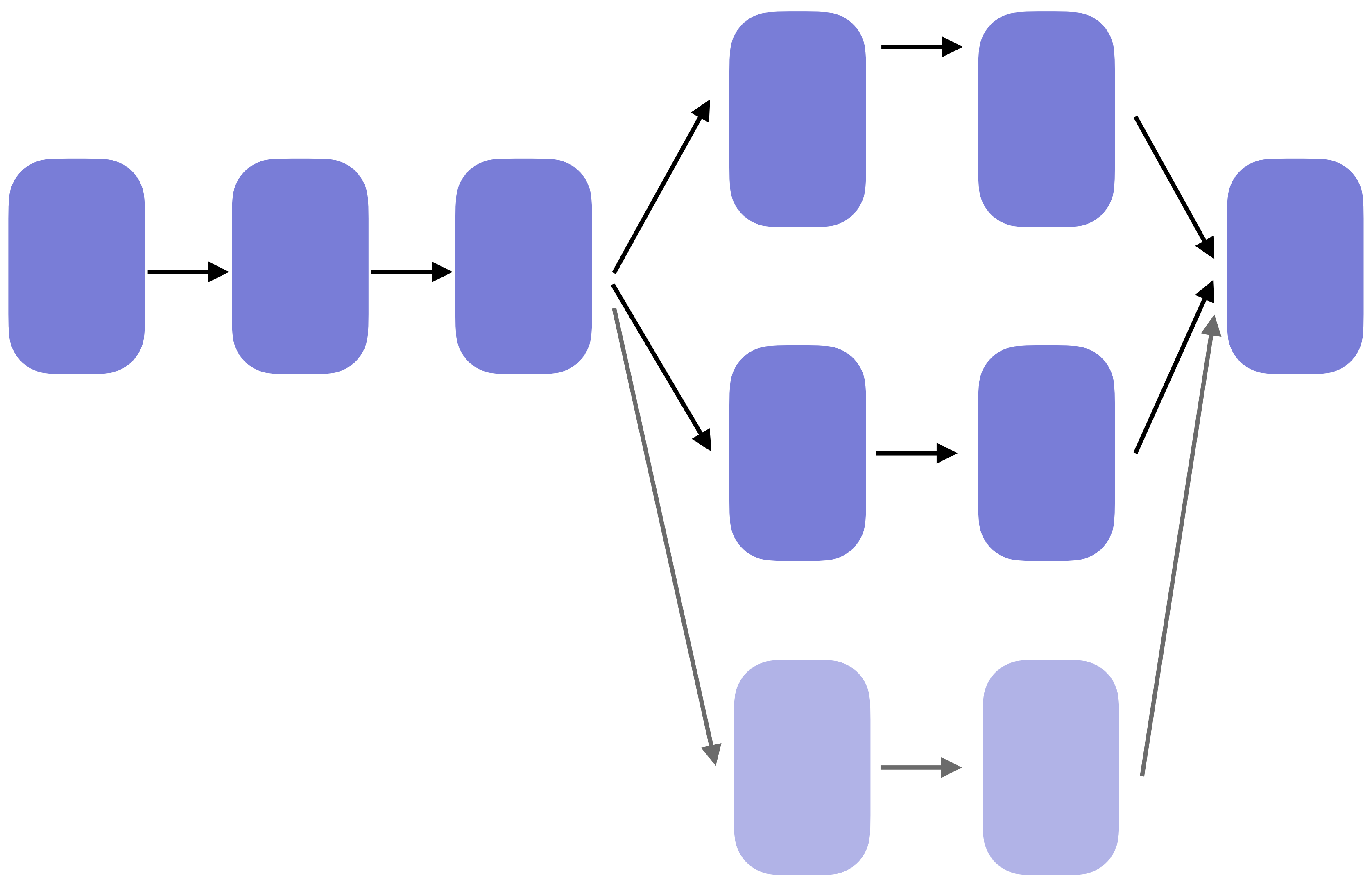


Structure of *learning* AutoML systems



AutoML: subproblems

- **Model search space Λ**: *represent* all pipelines or neural architectures
  - Pipeline operators, neural layers, interconnections,…
  - Defines a (complex) search space

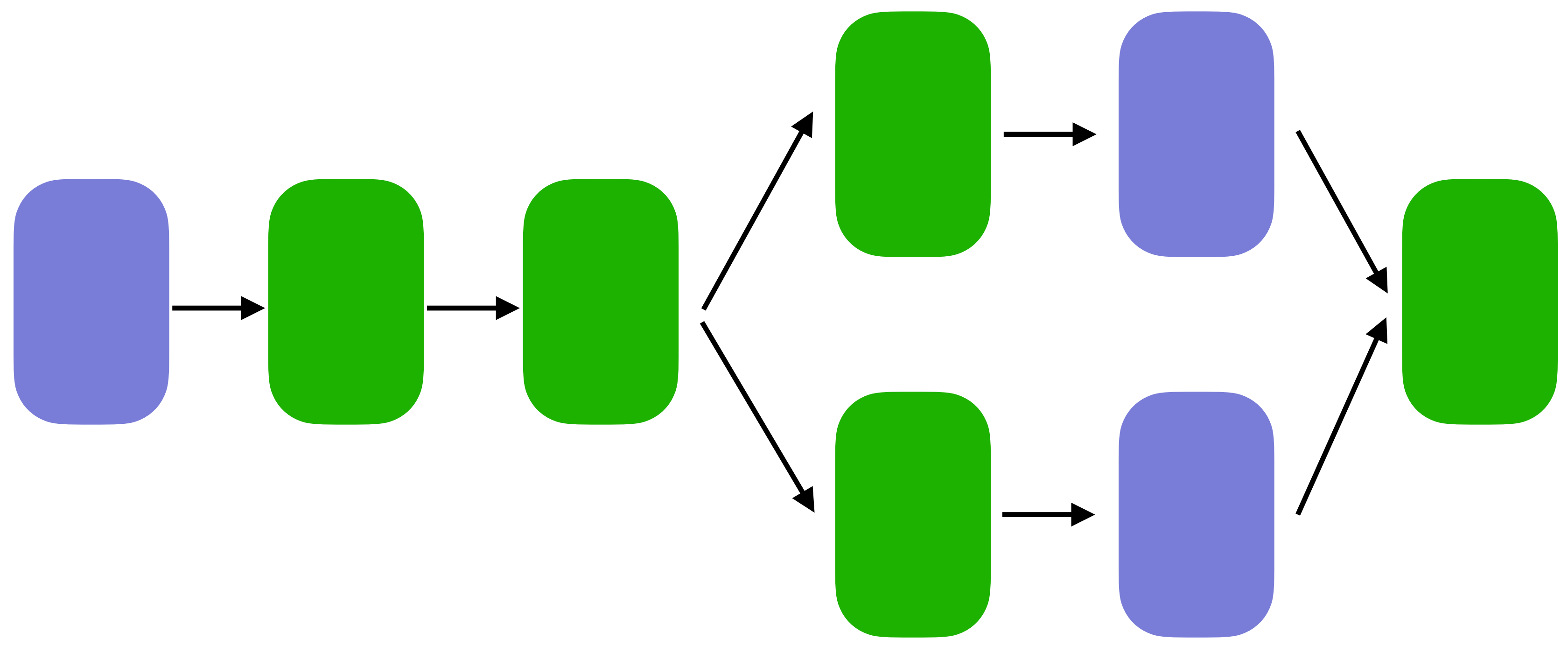


```
make_pipeline(
  OneHotEncoder(),
  Imputer(),
  StandardScaler(),
  SVC())
```

```
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

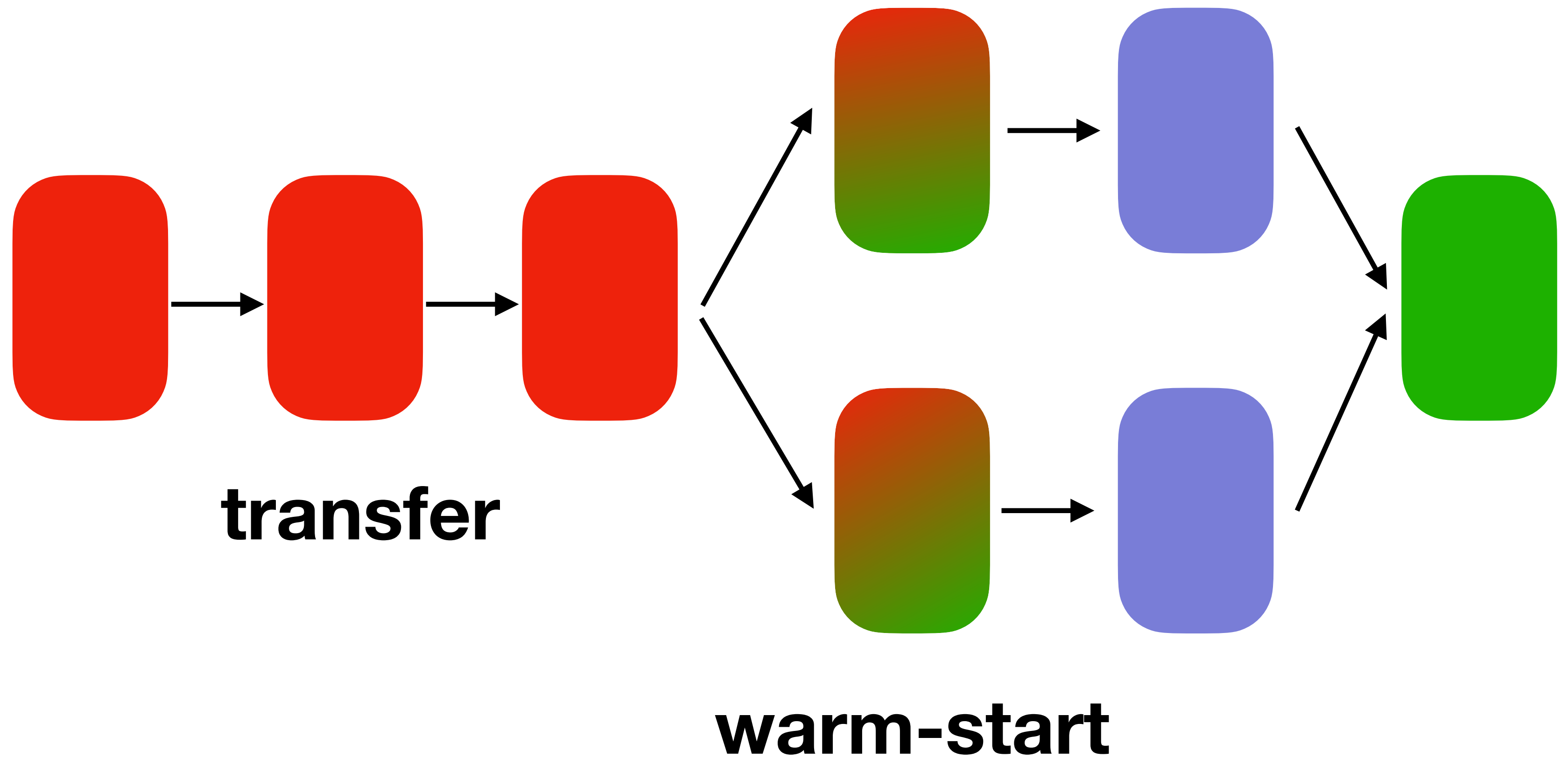
- **Model search space**: *represent* all possible architectures
- **Optimization**:
  - What is the best architecture? Which hyperparameters are important?
  - How to optimize them? What is the (multi-) objective function?



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

AutoML: subproblems

- **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

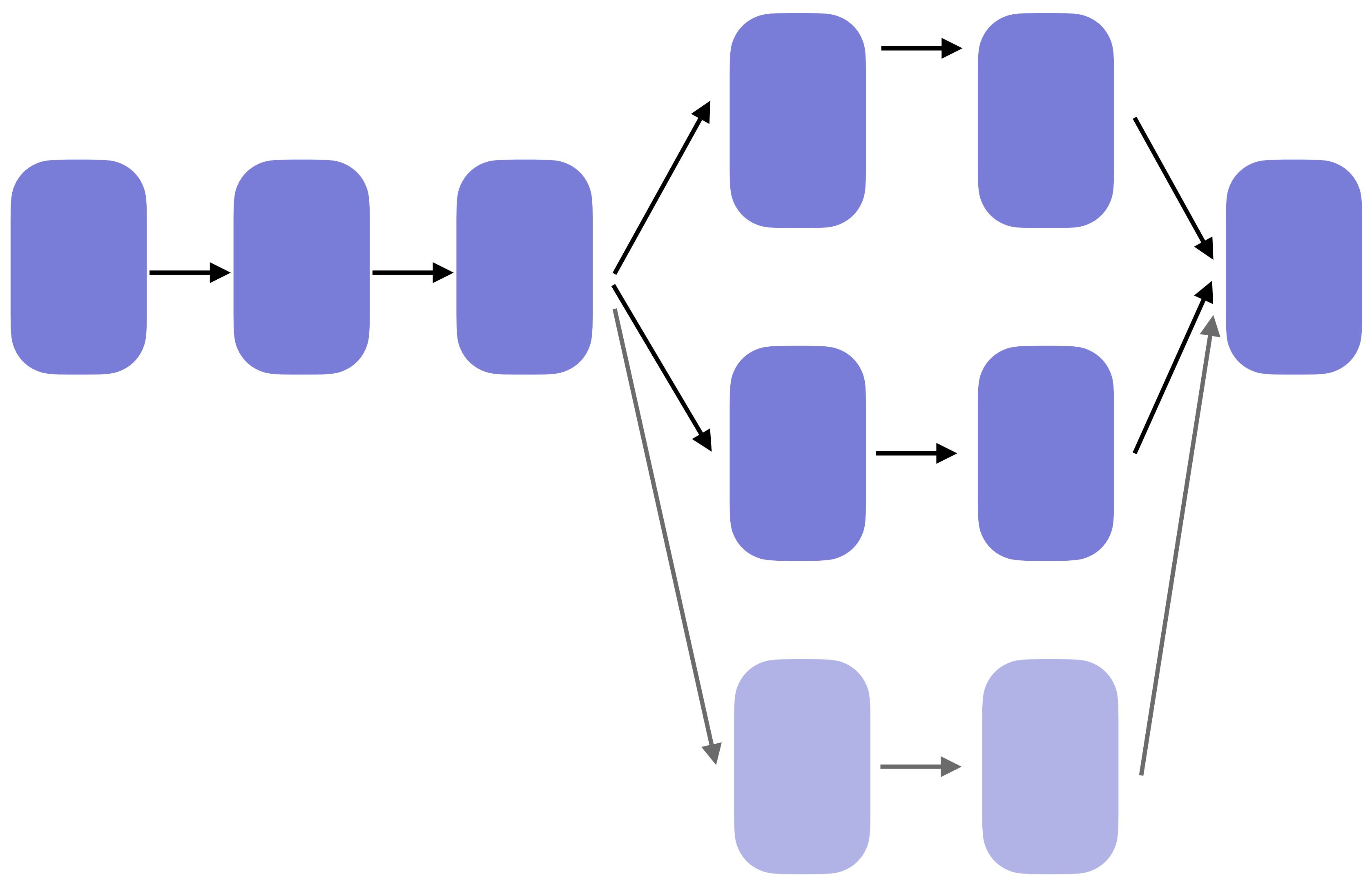


AutoML: subproblems

- **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*

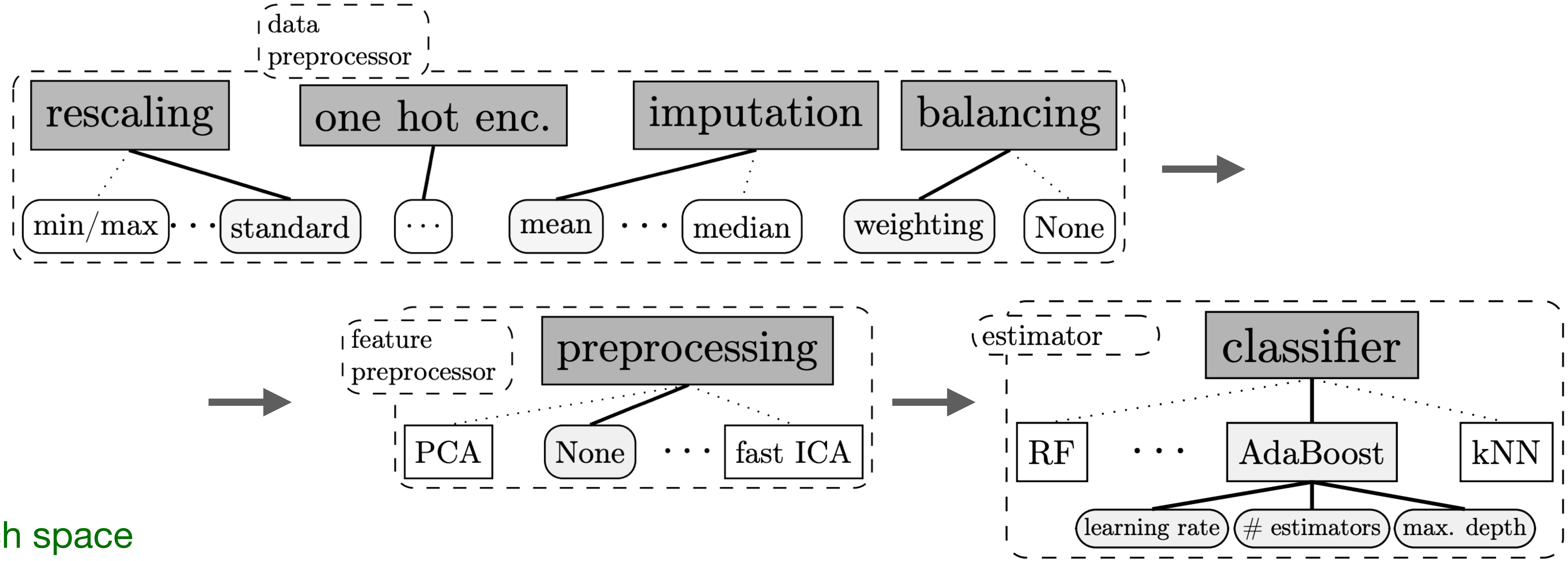
Architecture search space



Parameterized architectures

autosklearn Feurer et al. 2015
autoWEKA Thornton et al. 2013
hyperopt-sklearn Komer et al. 2014

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

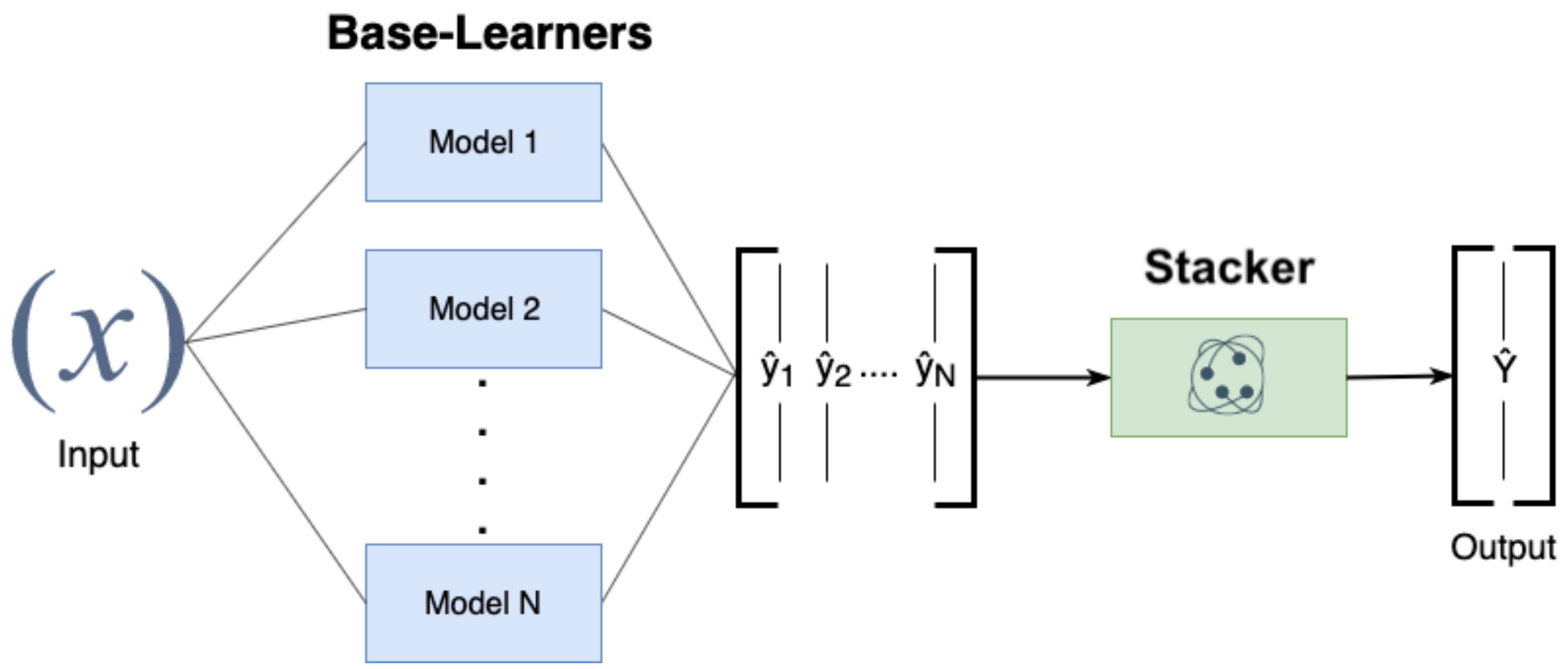


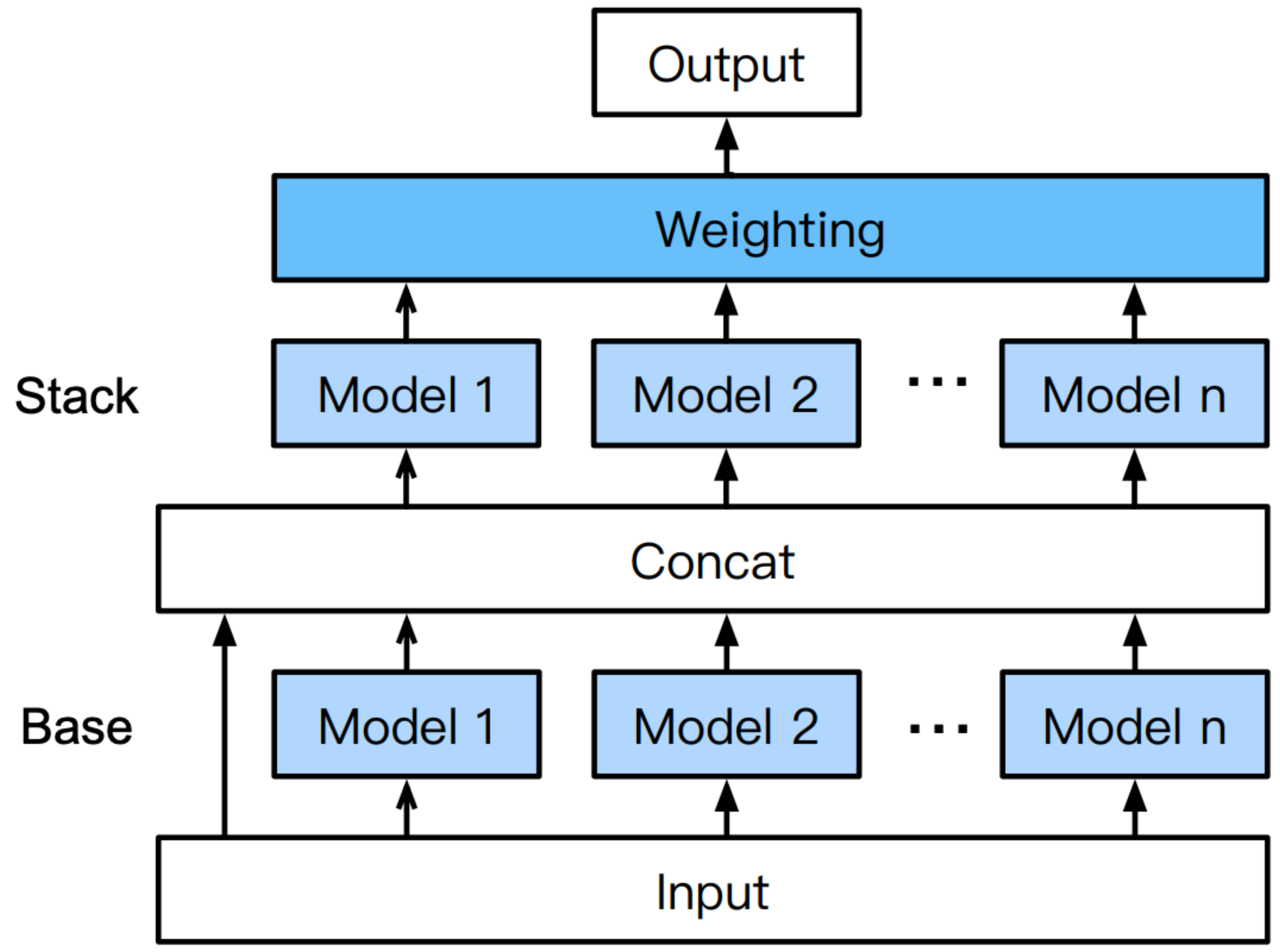
+ smaller search space
- you can't learn entirely new architectures

Ensembling

autosklearn Feurer et al. 2015
GAMA Gijsbers & Vanschoren, 2020
AutoGluon-Tabular, Erikson et al. 2020
H2O AutoML, LeDell et al.

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





Multi-level stacking

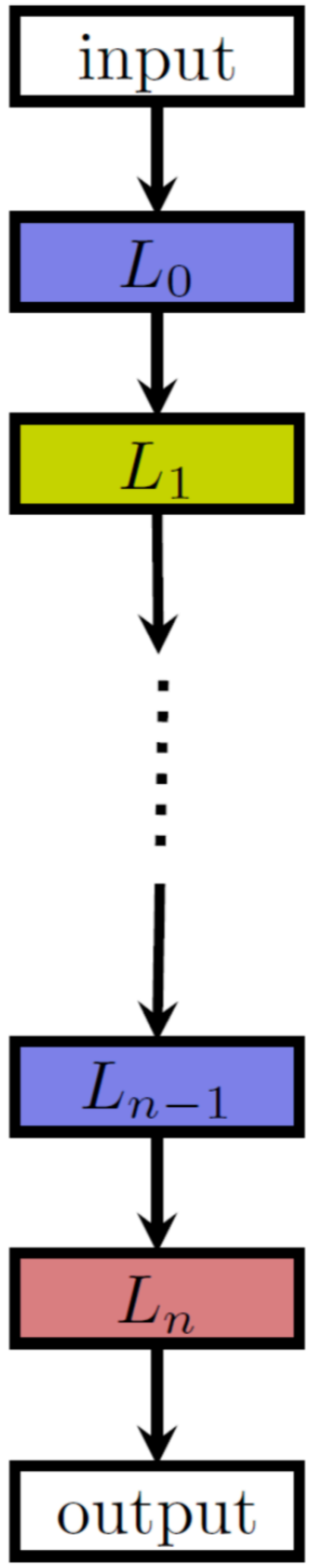
Elsken et al. 2019

Parameterized neural architectures

Parameterized Sequential

Choose:
- number of layers
- type of layers
  - dense
  - convolutional
  - max-pooling
  - …
- hyperparameters of layers

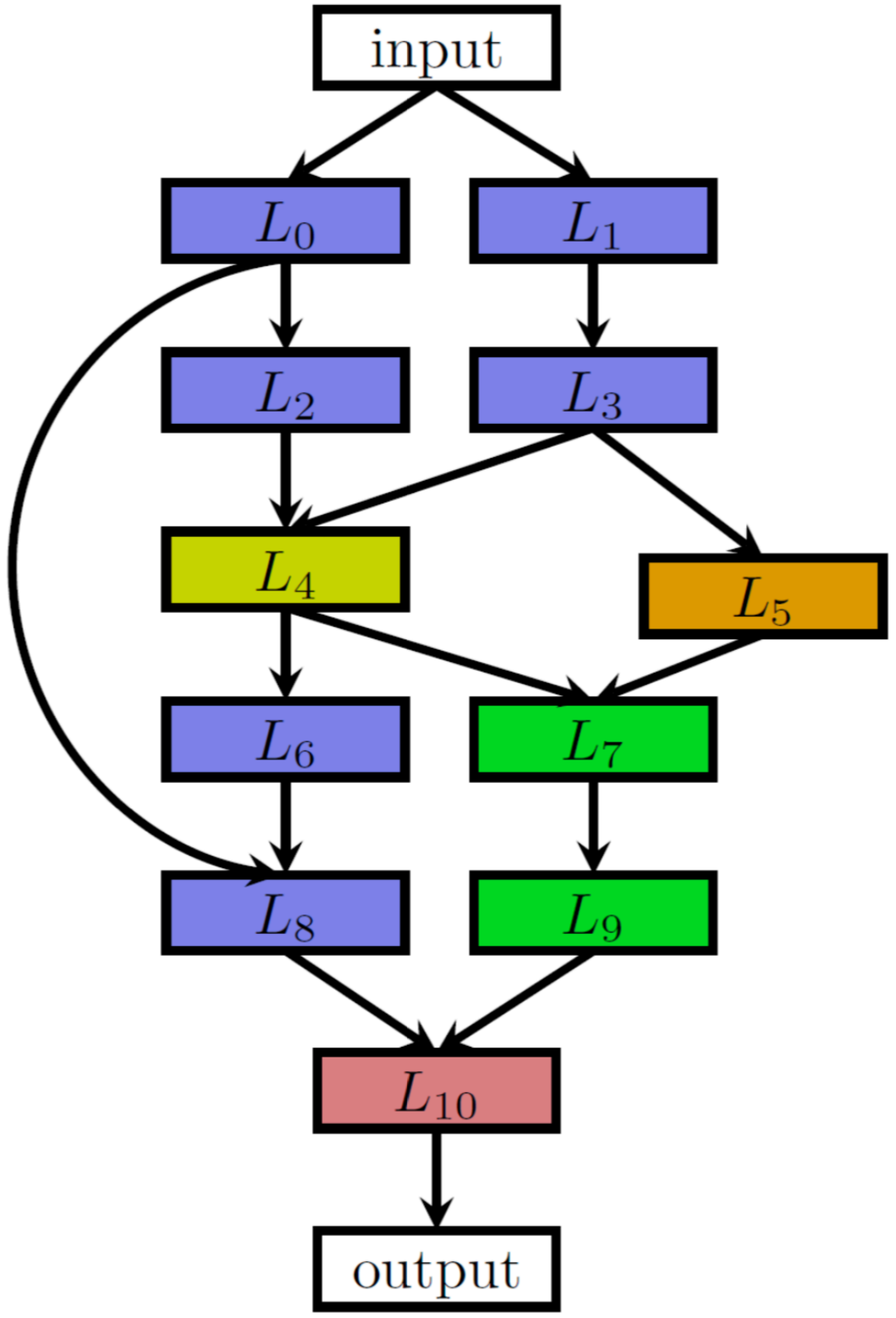
+ easier to search
- sometimes too simple



Parameterized Graph

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

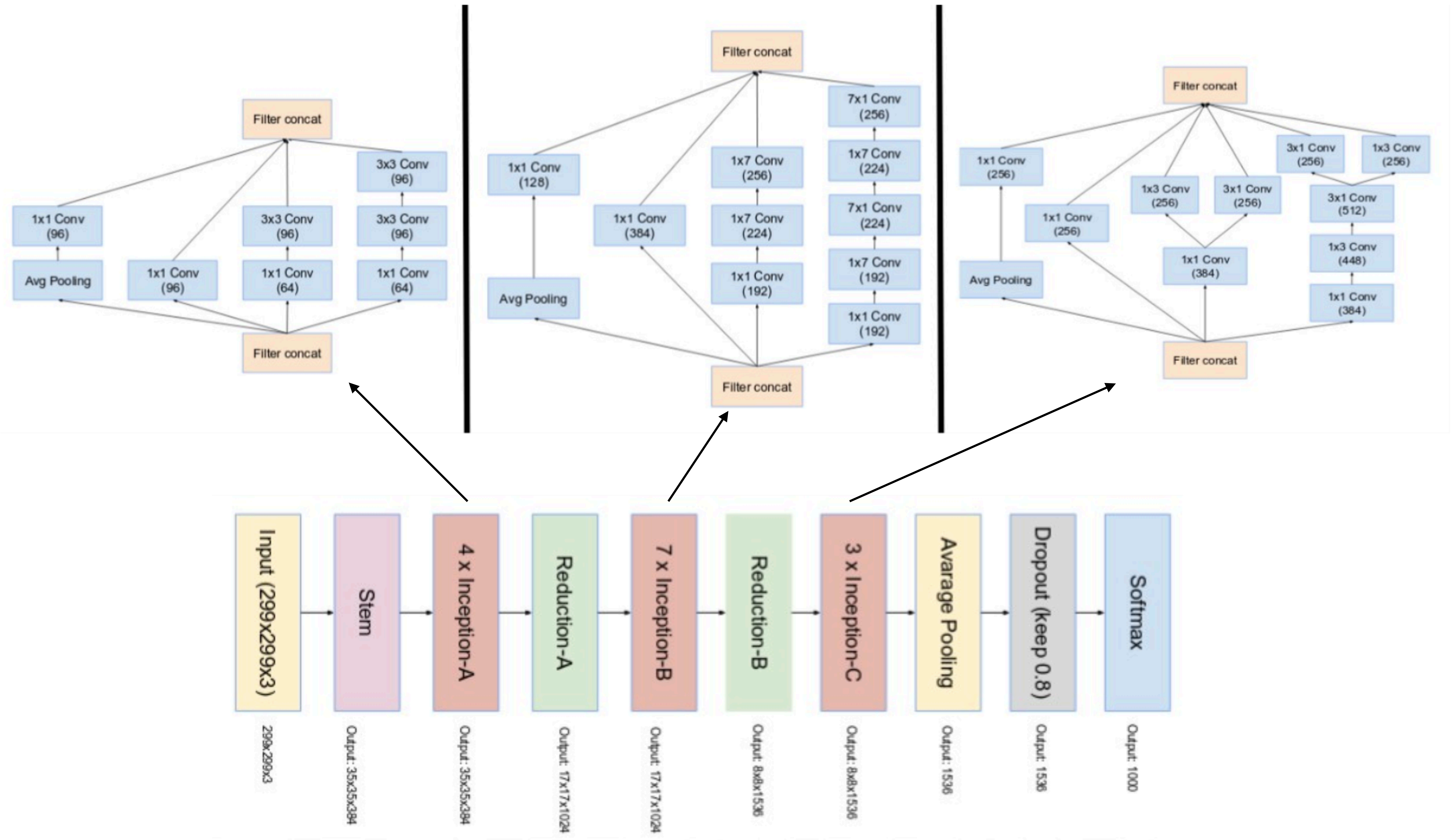
+ more flexible
- much harder to search



Cell search spaces

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

e.g. Inception v4:



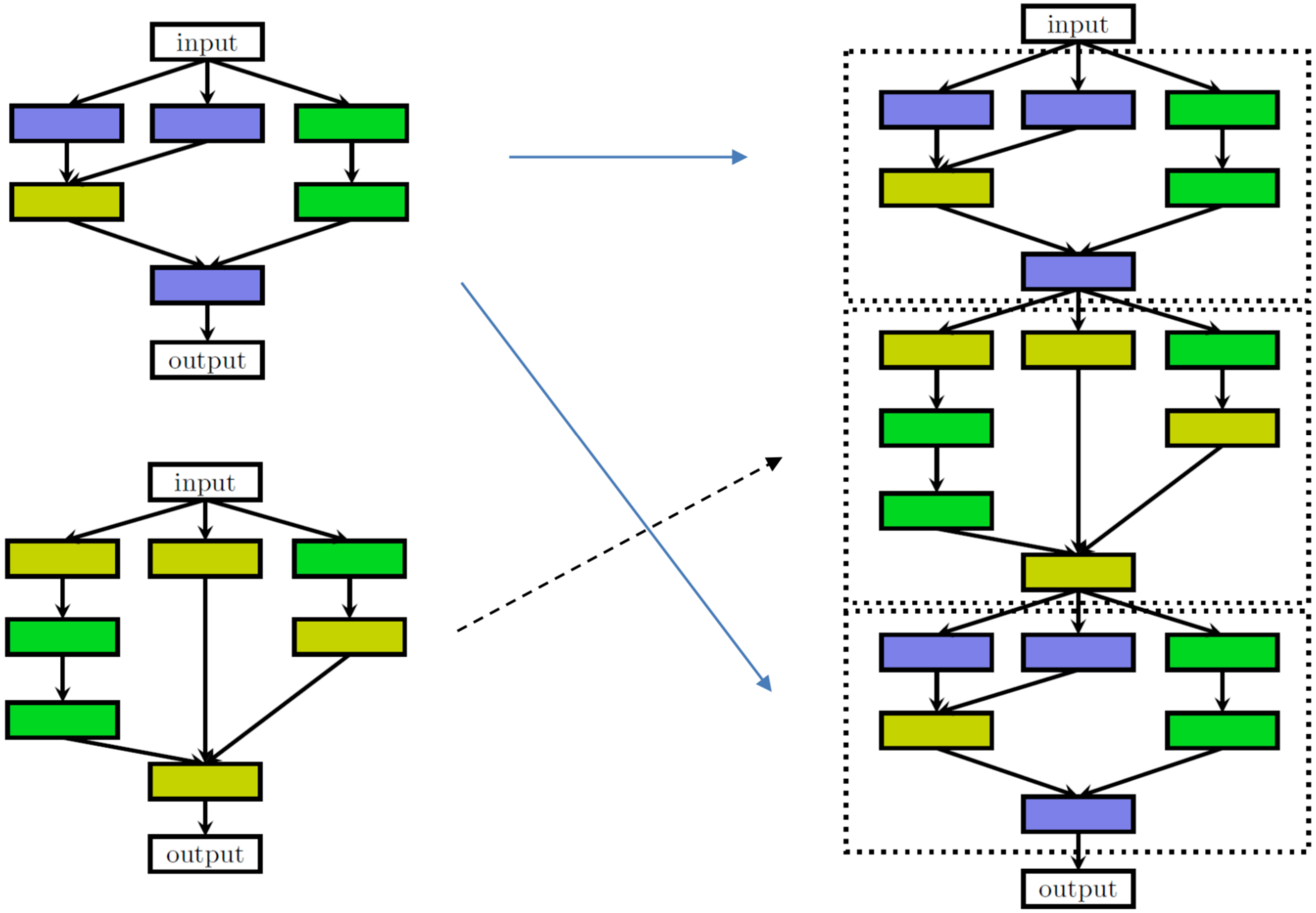
Cell search spaces

Google NASNet [Zoph et al 2018]

Compositionality: learn hierarchical building blocks to simplify the task

Cell search space

- learn parameterized building blocks *(cells)*
- stack cells together in macro-architecture

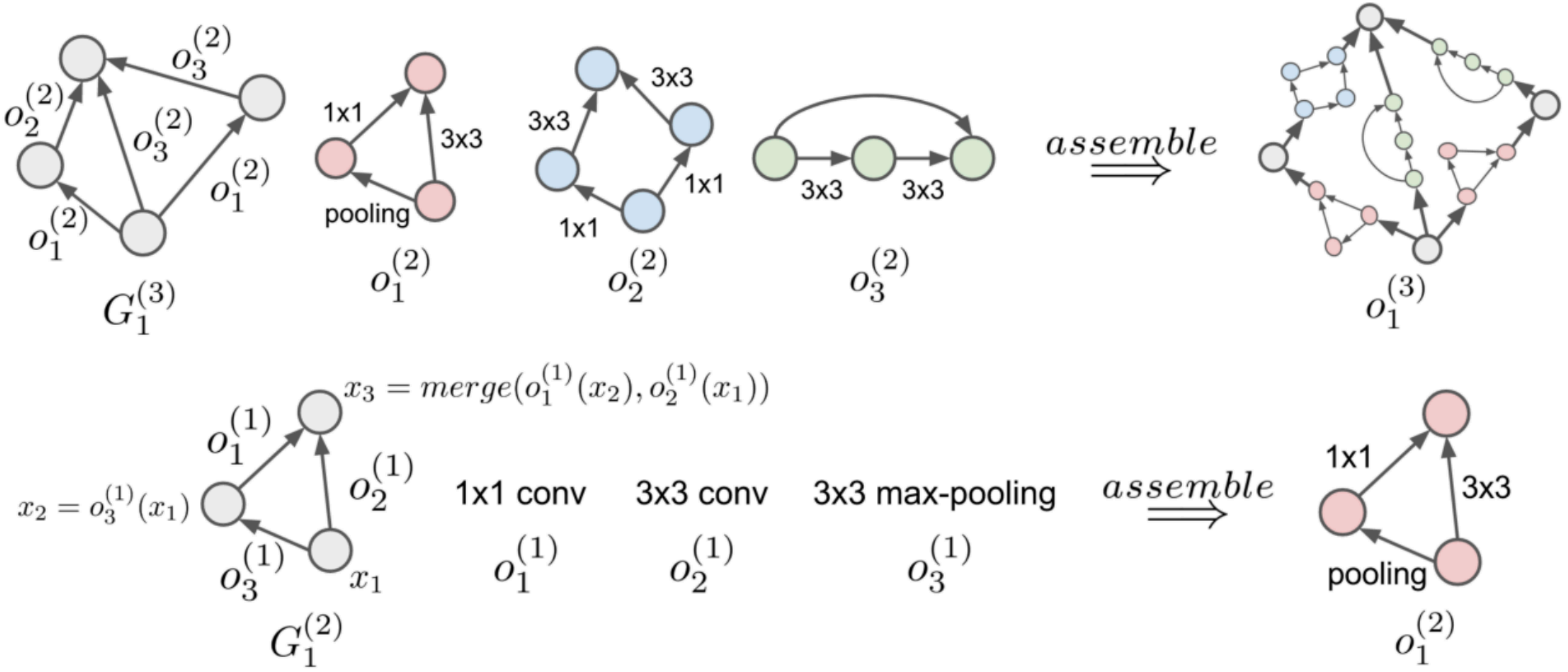


+ smaller search space
+ cells can be learned on a small dataset & transferred
- strong domain priors, doesn't generalize well

Hierarchical Search Spaces

Liu et al. 2018

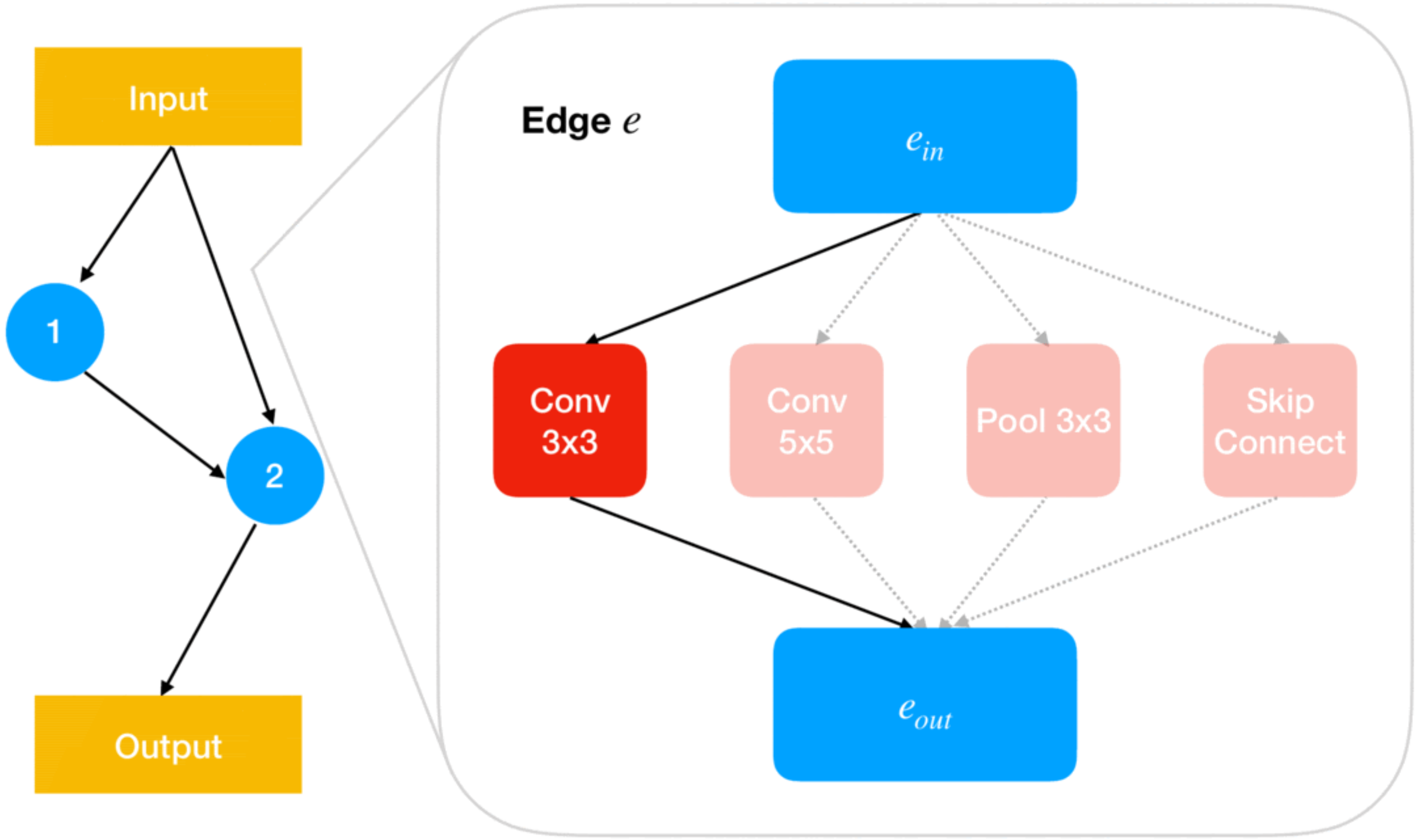
- Define a number of primitive operations (e.g. convolution, max-pooling,…)
- Build small motifs of primitives (o_1, o_2, o_3,…)
- Choose a graph structure G_i, replace edges with motifs, repeat



Differentiable Search Spaces

Liu et al. 2018

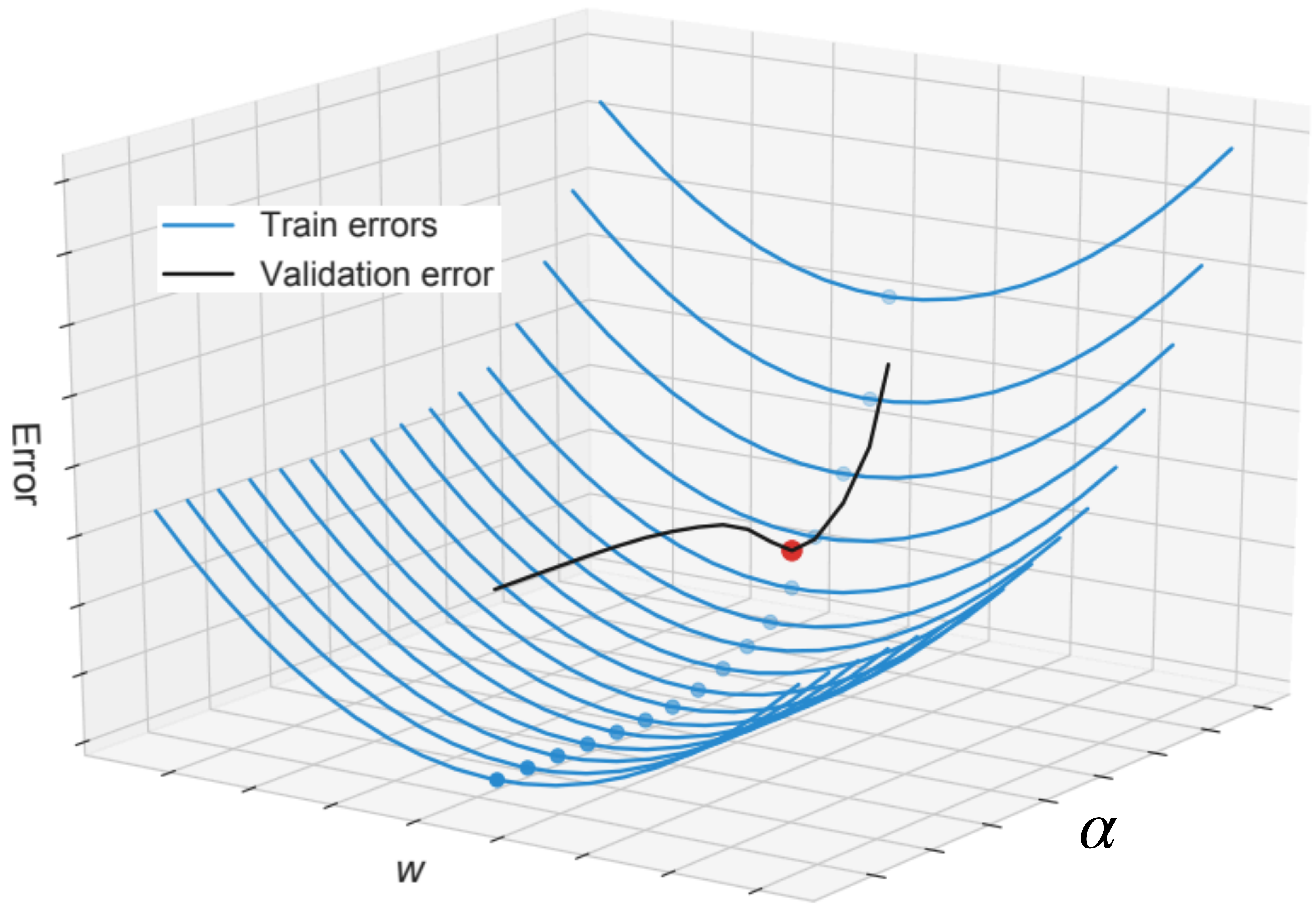
- Fixed (one-shot) structure, each edge can be any primitive operation
- Give all operators a weight α_i -> *pure numeric/differentiable search space!*
- Each edge output: weighted sum of outputs of all operators



Differentiable Search Spaces

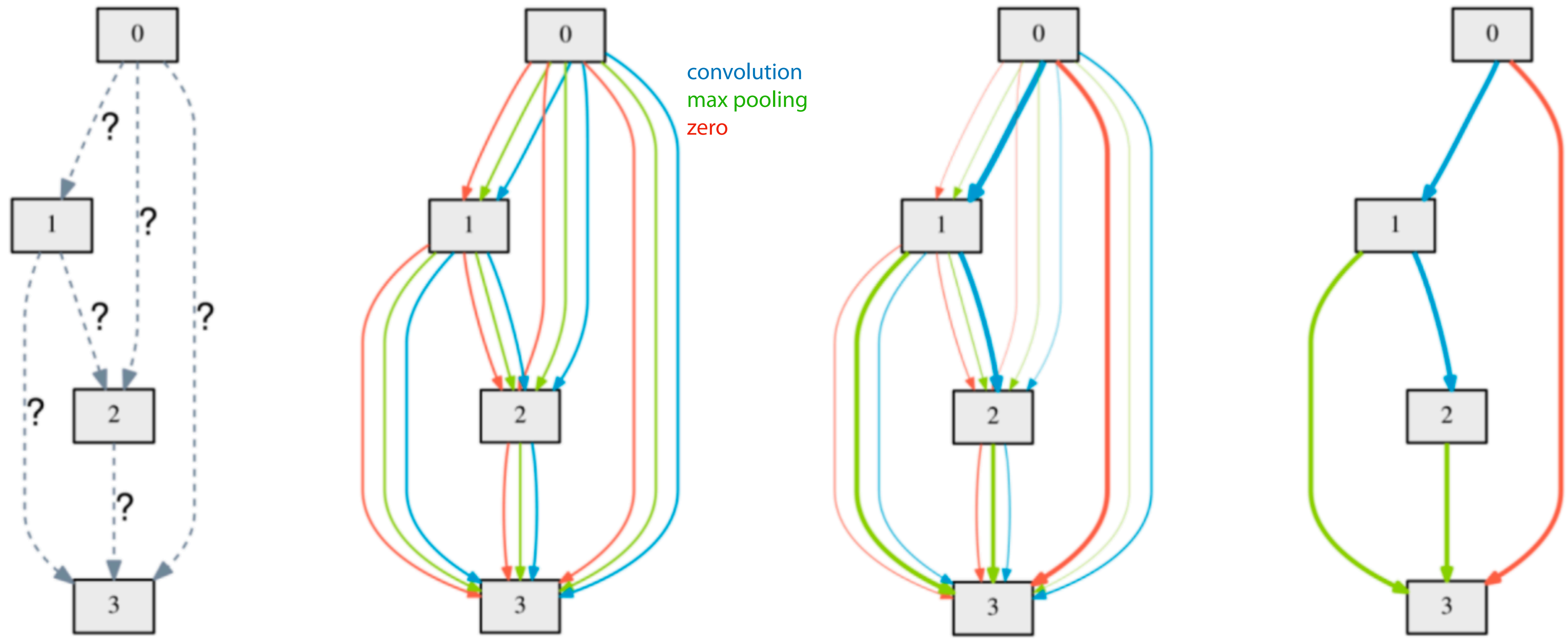
Liu et al. 2018

- Optimize operator weights α_i and model weights ω_j using bilevel optimization



Differentiable Search Spaces (DARTS)

- Optimize α_i and model weights ω_j using bilevel programming
- After convergence, make the model sparse by keeping only the largest weights
- This 'weight sharing' between possible models speeds up the search dramatically



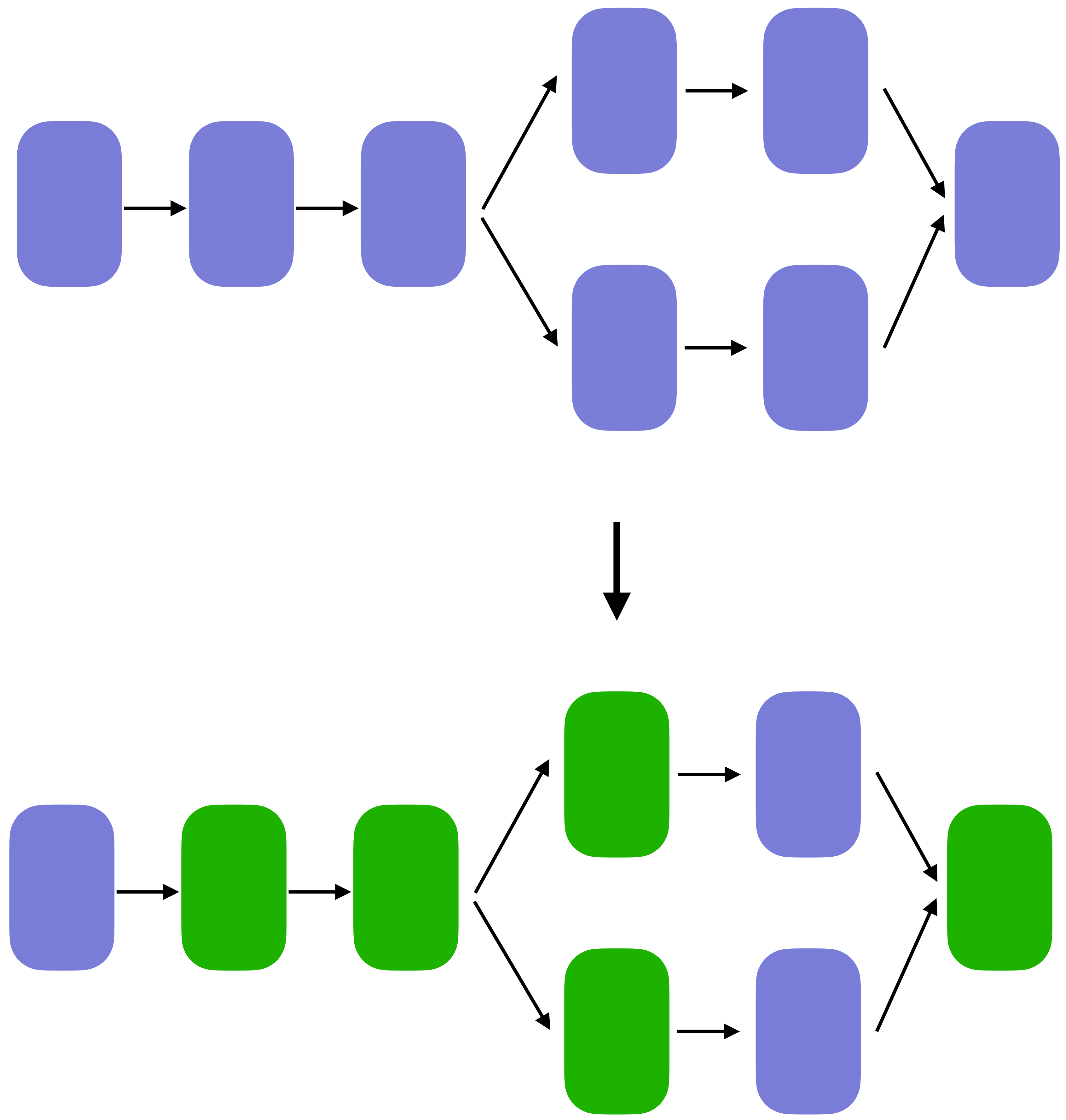
One-shot model

operator weights α_i

interleaved optimization of α_i and ω_j with SGD

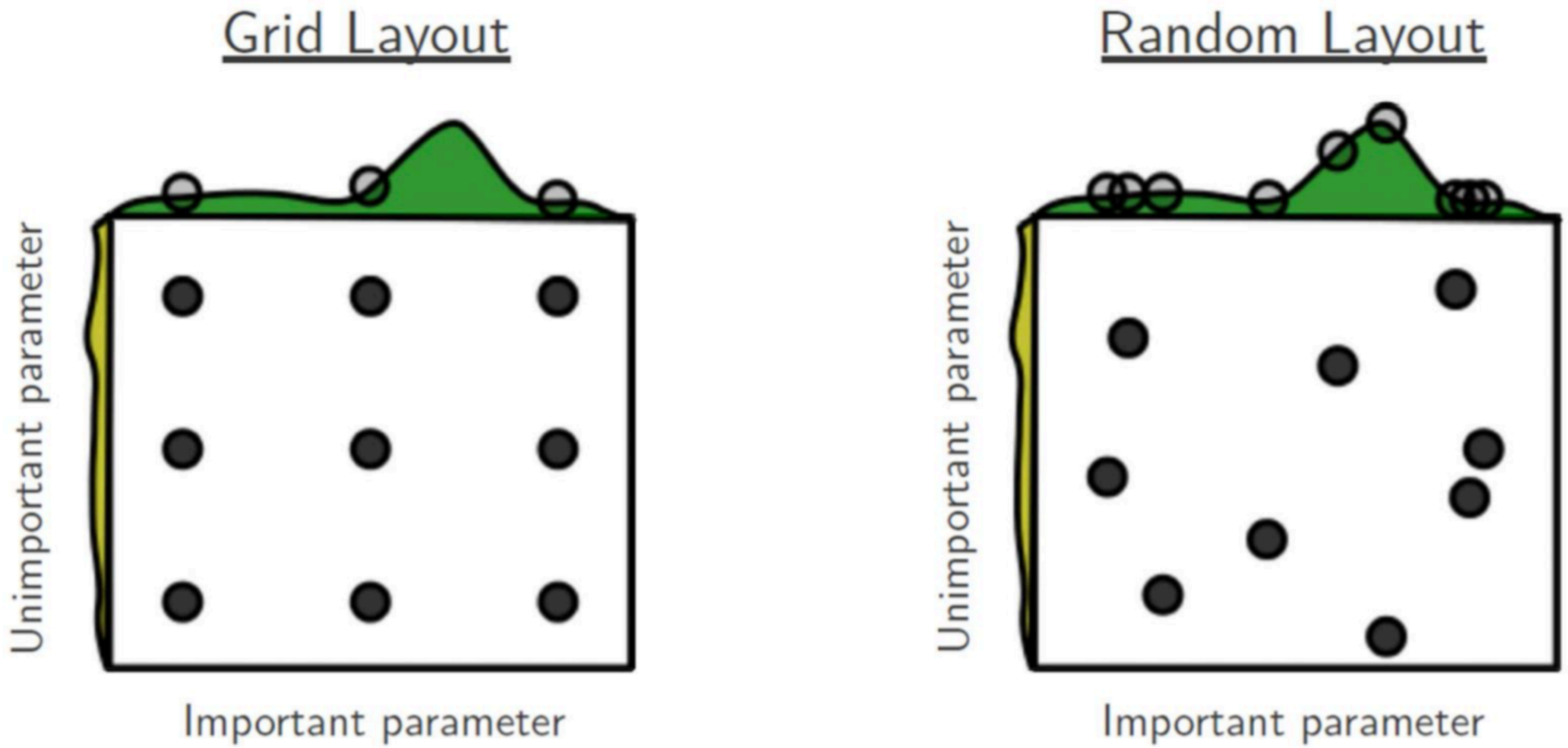
argmax α_i

Optimization



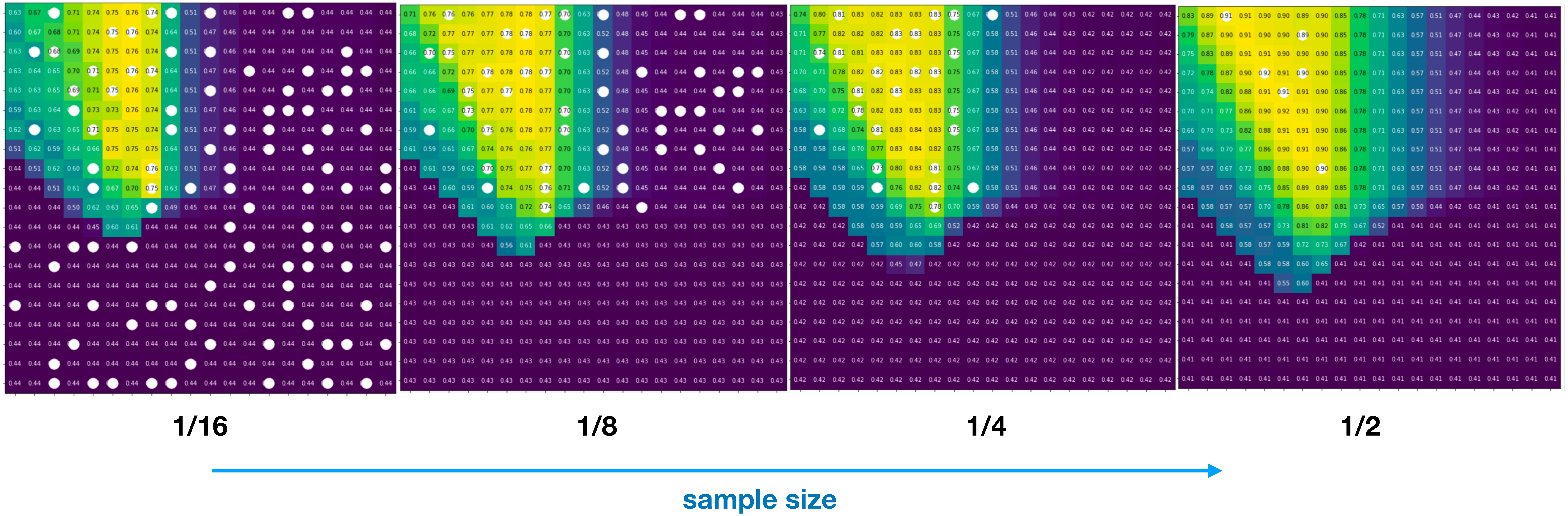
Random search

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



Successive Halving

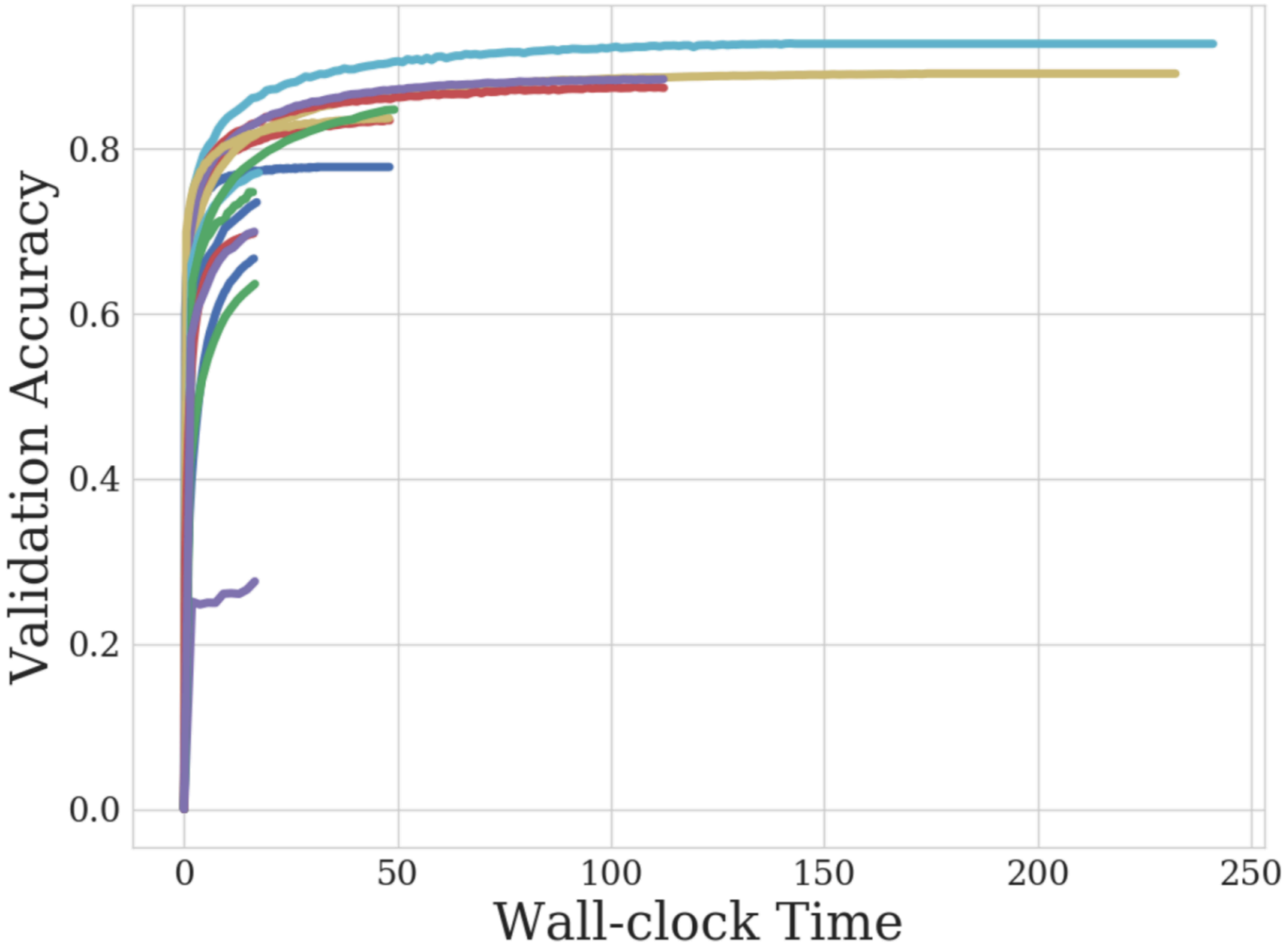
- Train on small data subsets, infer which regions may be interesting to evaluate in more depth
  - Randomly sample candidates and evaluate on a small data sample
  - Retrain the 50% best candidates on twice the data, repeat



Hyperband

Successive halving risks killing good models too early

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

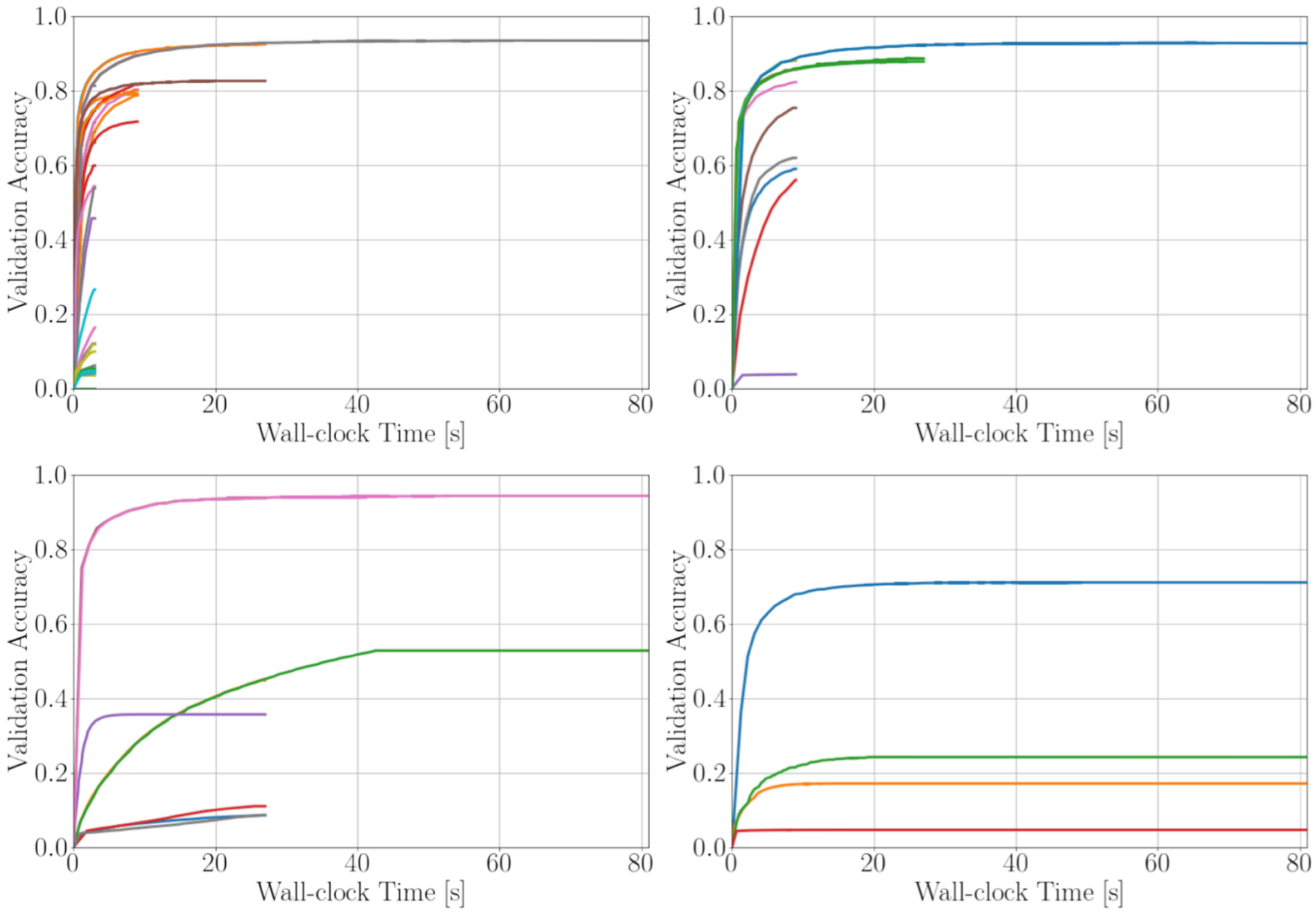


Li et al. 2017

Hyperband

Successive halving risks killing good models too early

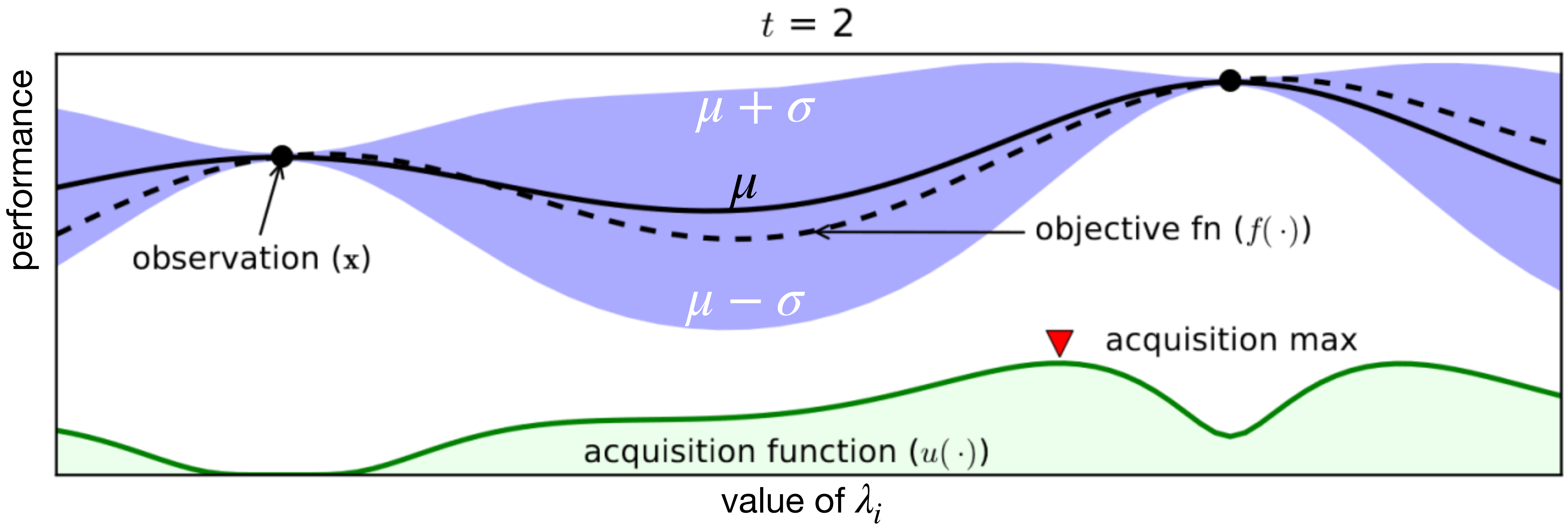
- Repeat in multiple, decreasingly aggressive iterations (brackets)
- Strong anytime performance, easy to implement, scalable, parallelizable



Bayesian Optimization

Mockus [1974]

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



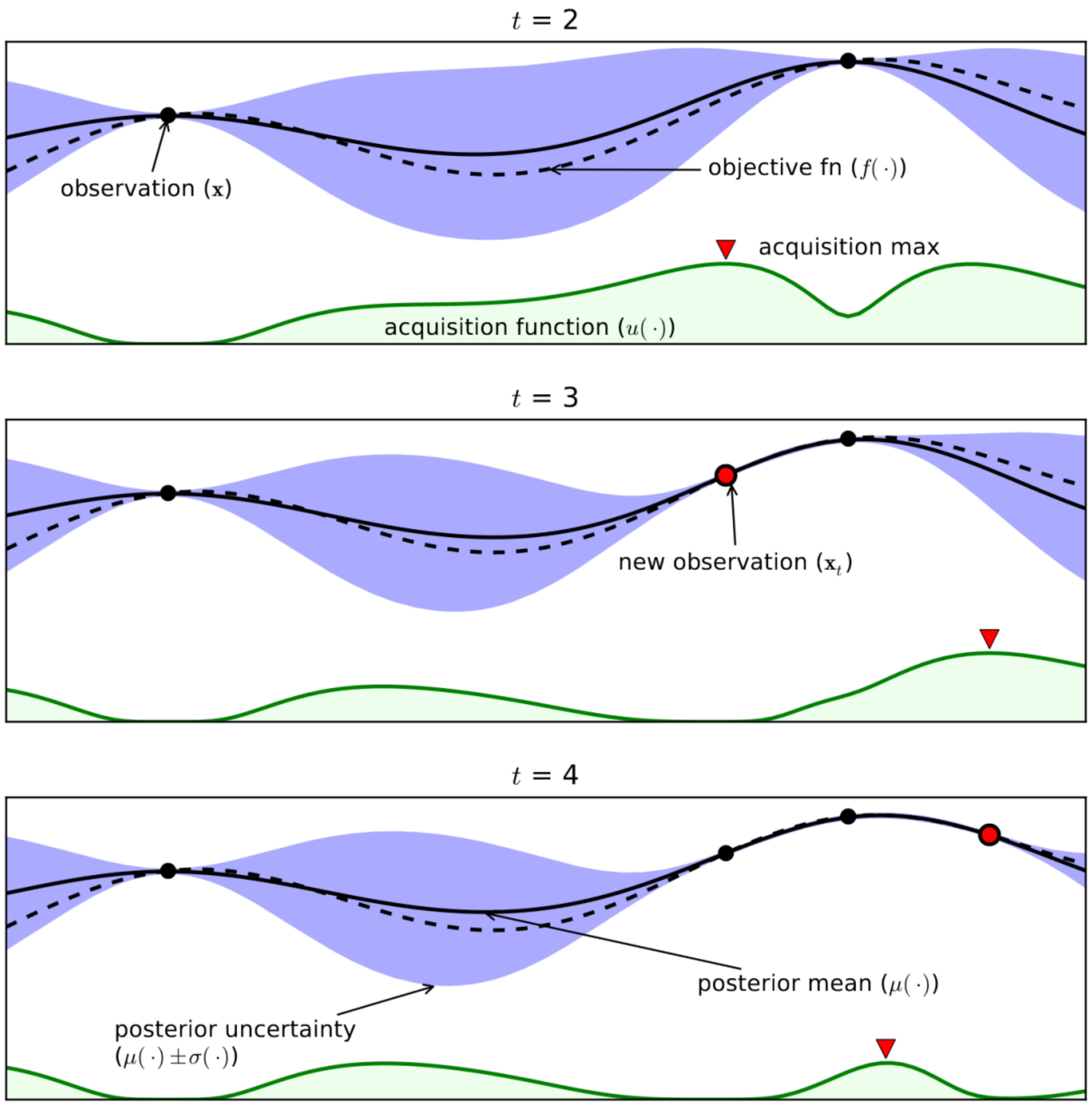
Bayesian Optimization

- 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

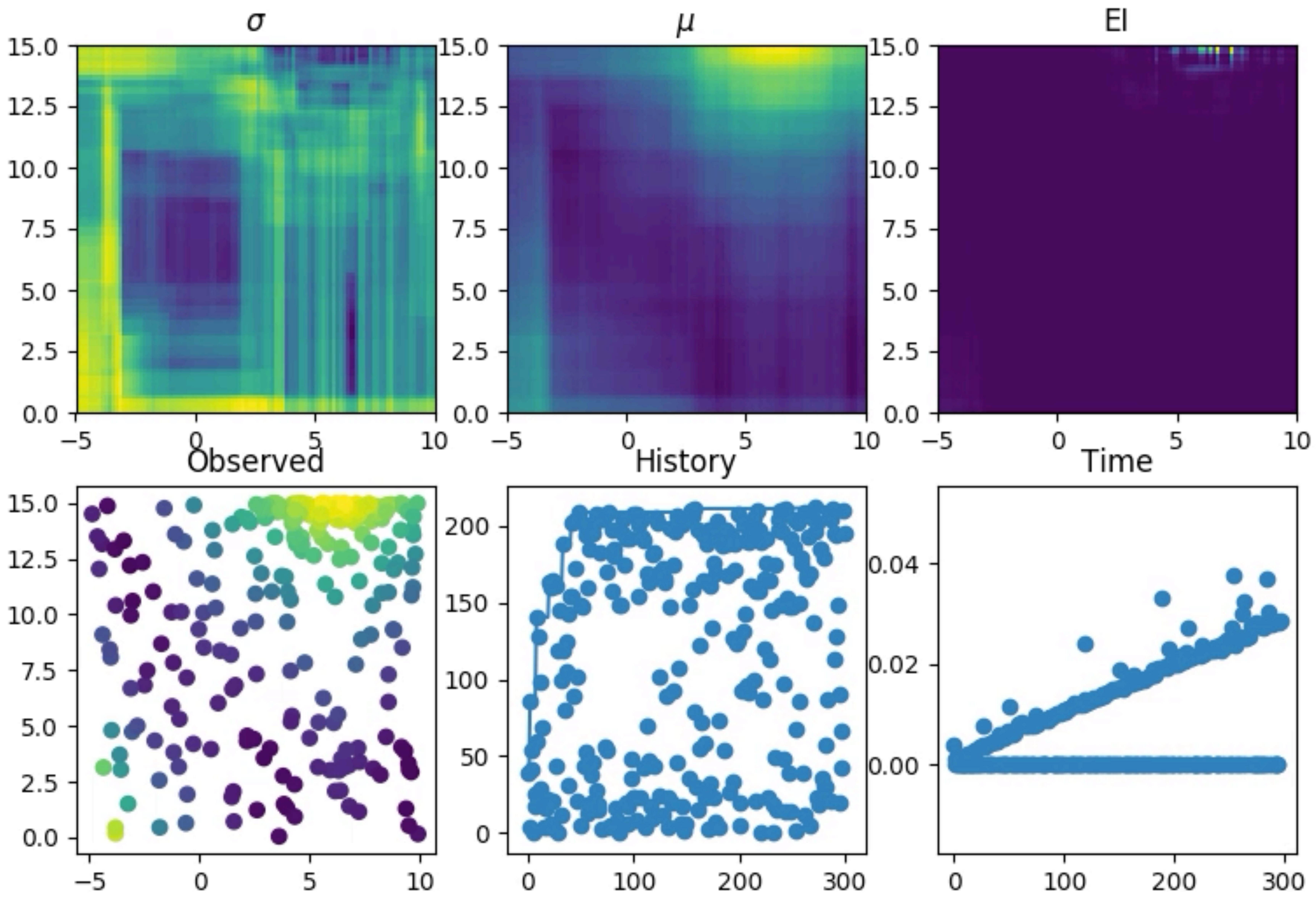


Bayesian Optimization (surrogate: random forests)

Hutter et al. 2011

Feuerer et al. 2015

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

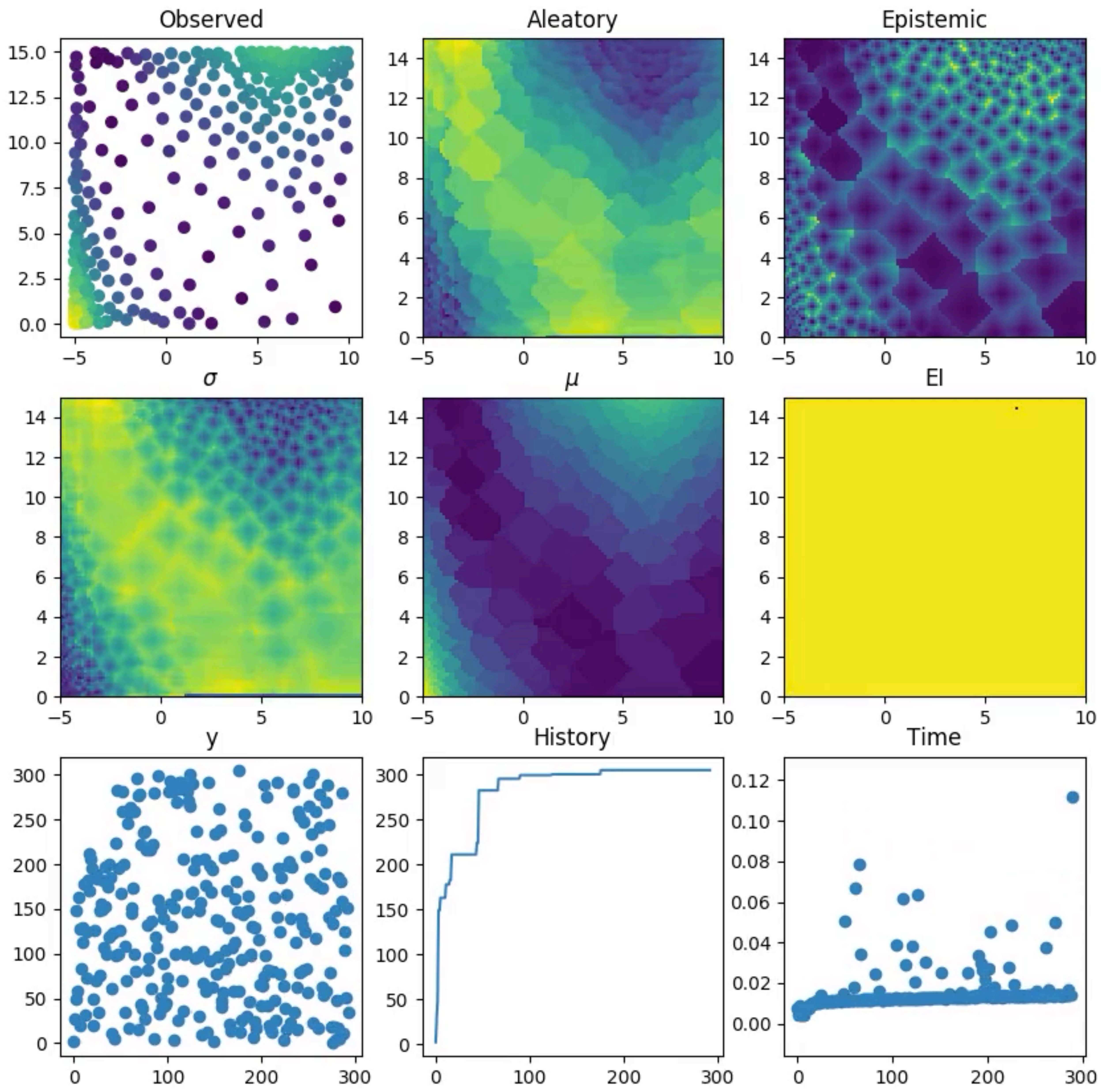


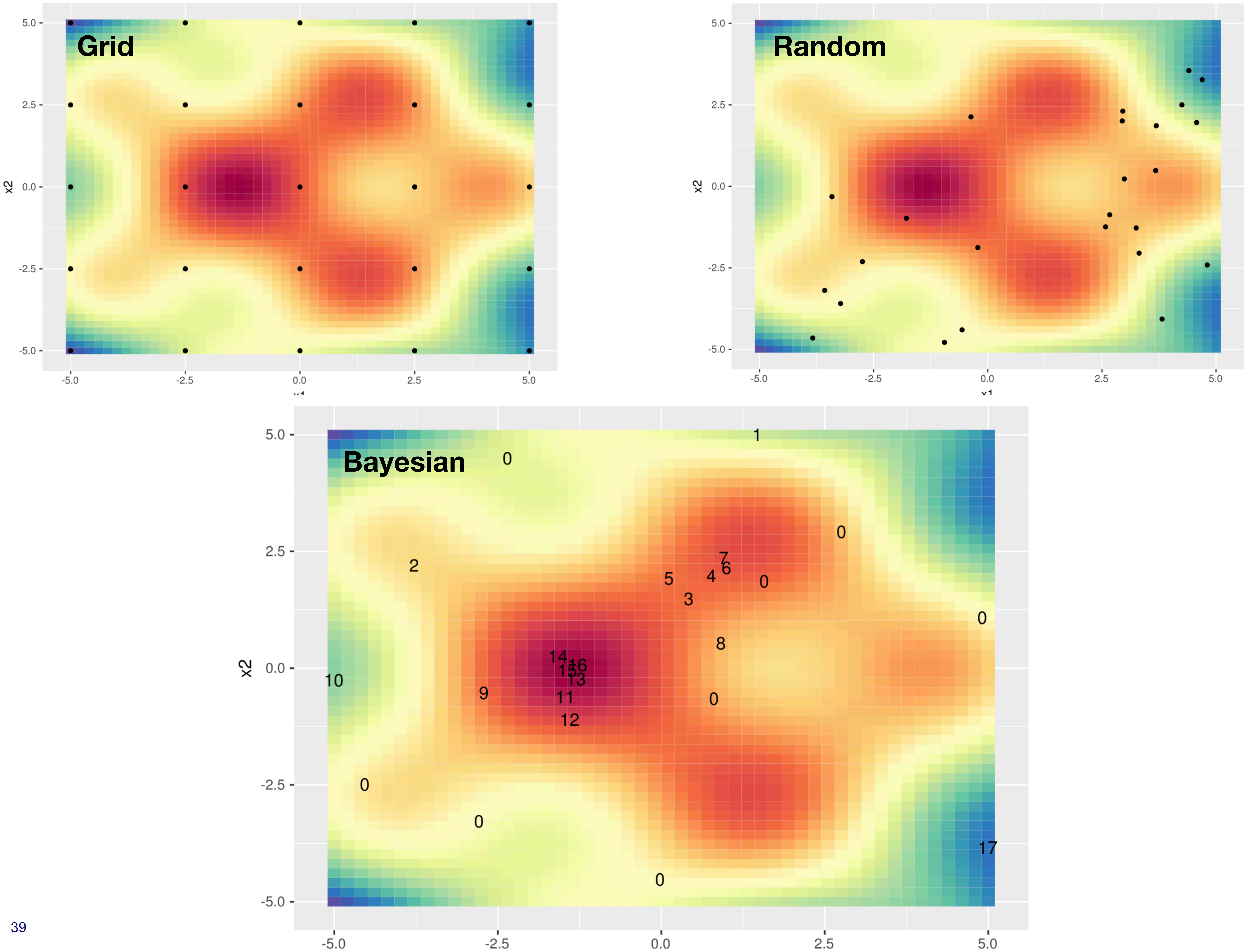
animation by Jeroen van Hoof

van Hoof & Vanschoren 2020

Bayesian Optimization (surrogate: gradient boosting)

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

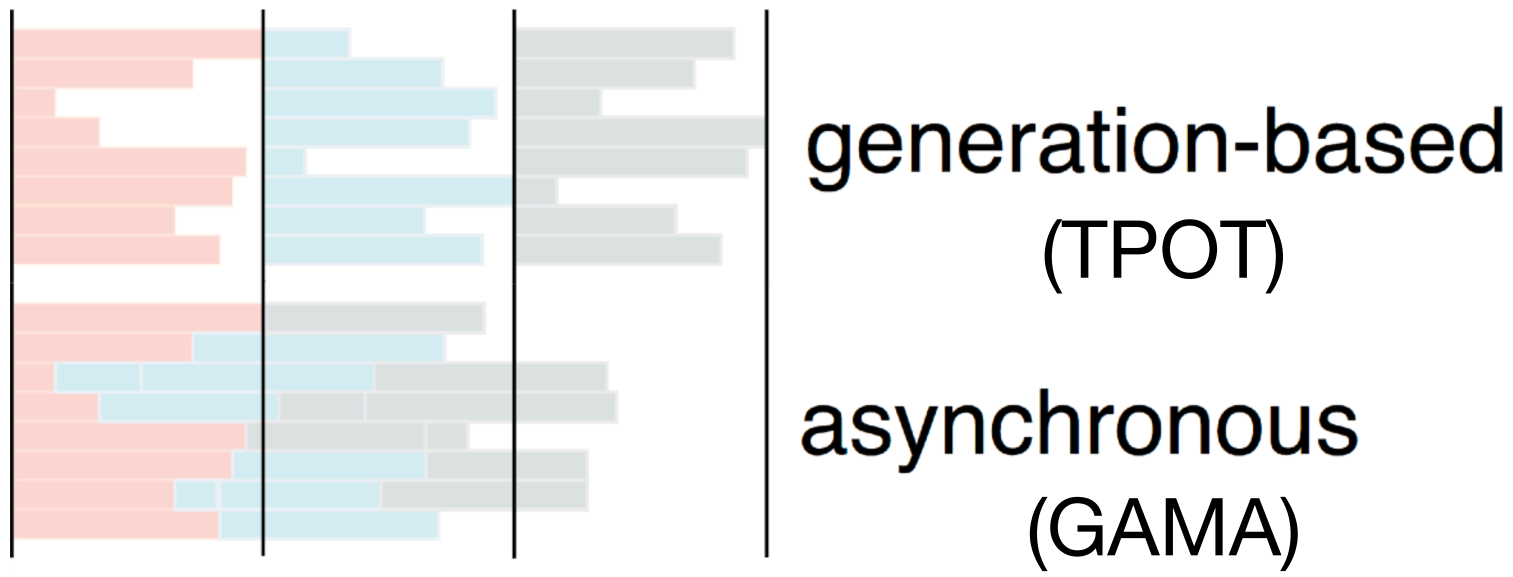


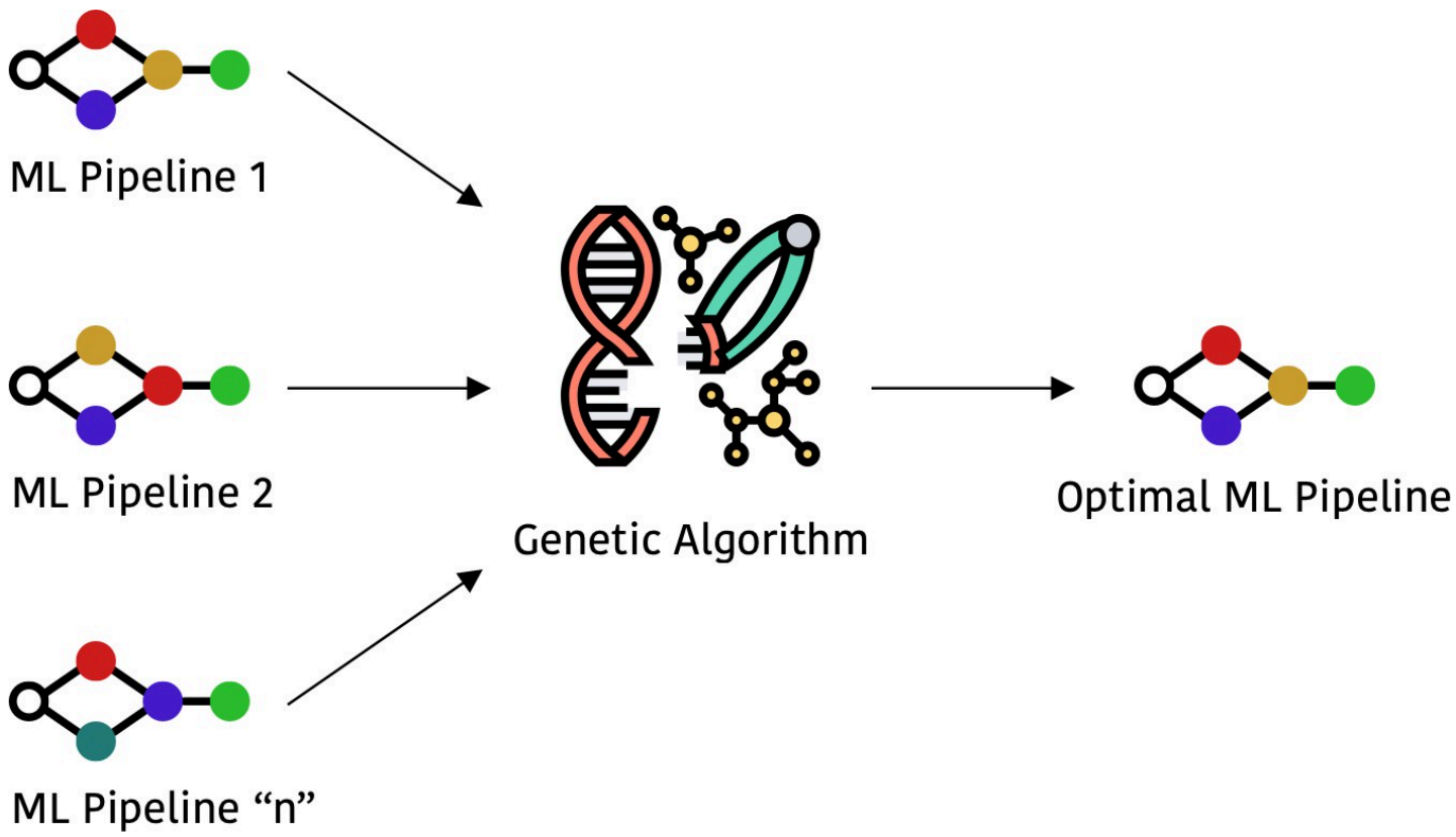

Grid
Random
Bayesian
x2
5.0
2.5
0.0
-2.5
-5.0
-5.0
-2.5
0.0
2.5
5.0
1
0
0
2
7
5
4
6
0
3
0
8
14
16
15
13
10
9
11
0
12
0
0
17
0

Evolution

TPOT - Olson & Moore 2016
GAMA - Gijsbers & Vanschoren 2018

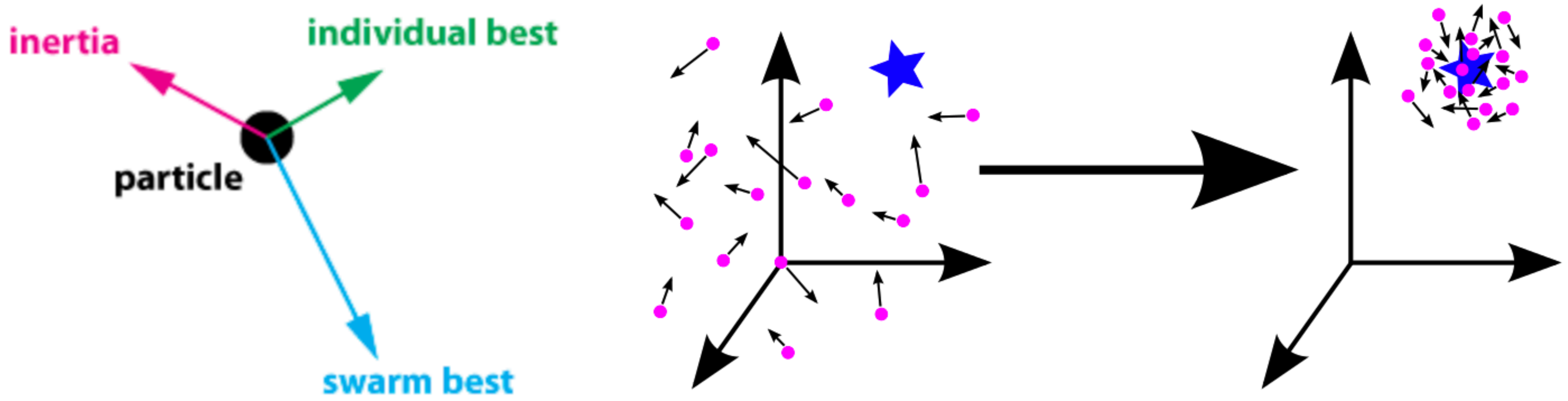
- Start with initial pipeline
- Best pipelines evolve: cross-over or mutation
- No fixed pipeline length: adapts to complexity of the problem
- Used in GAMA, TPOT,…





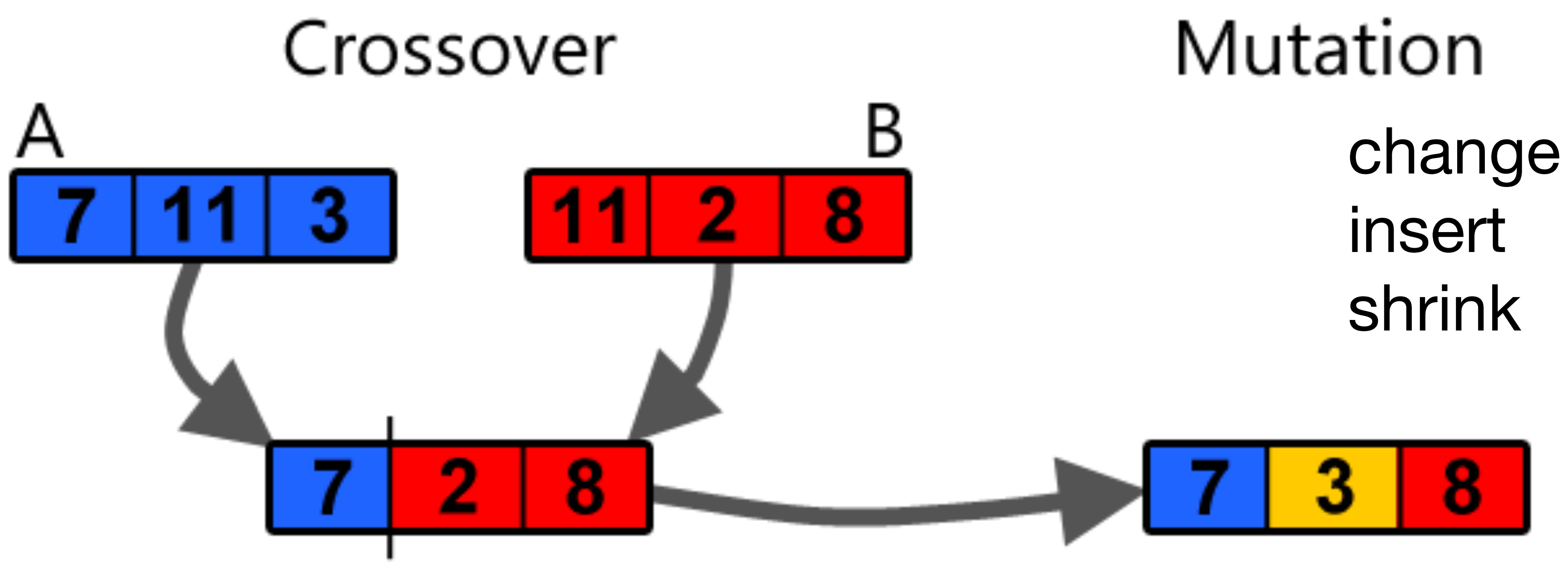
Evolution

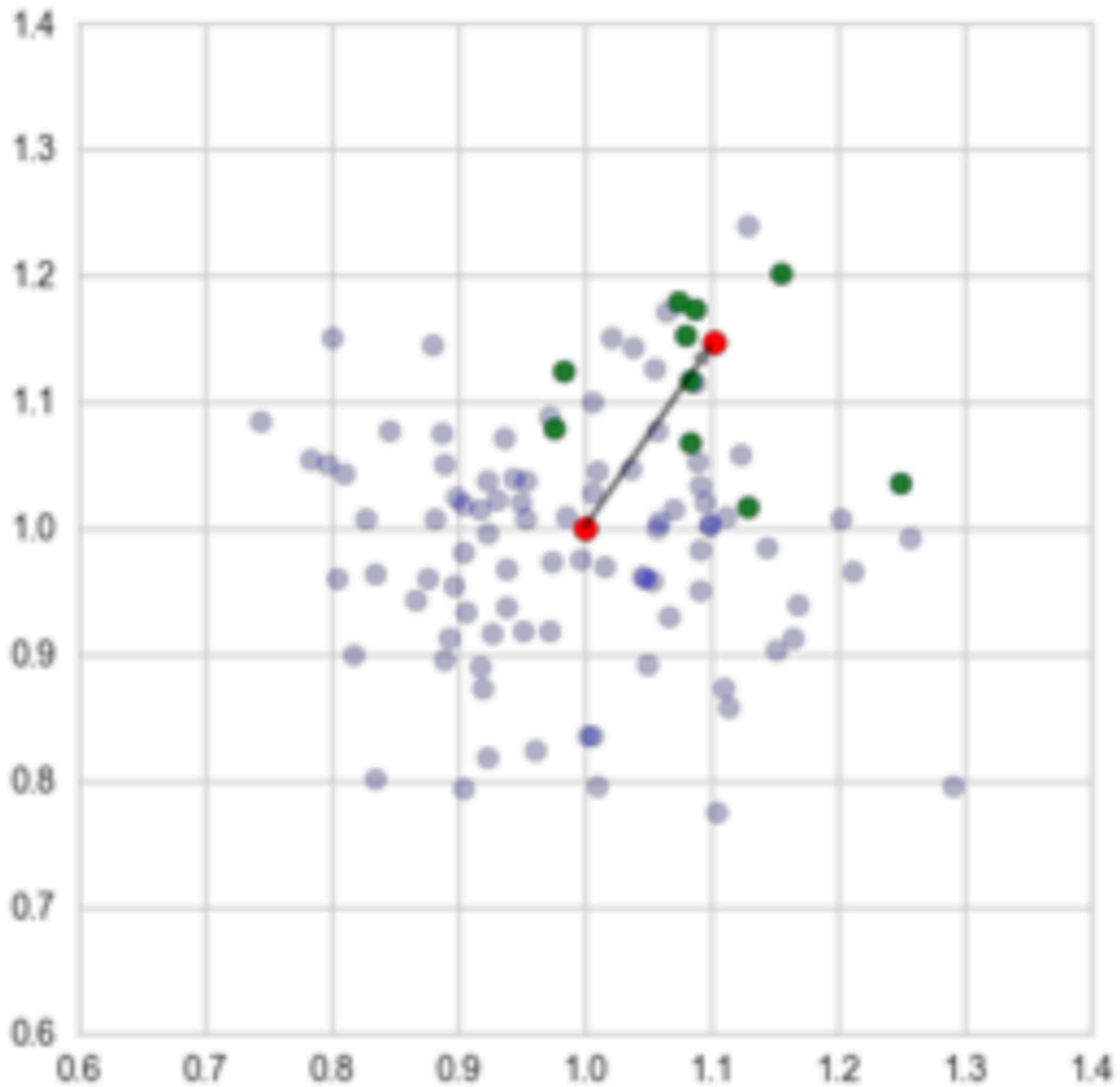
- Genetic programming TPOT Olson 2016 GAMA Gijsbers 2020
    - Mutations: add, mutate/tune, remove HP
- Particle swarm optimization Mantovani et al 2015



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

[Loshilov, Hutter 2016]



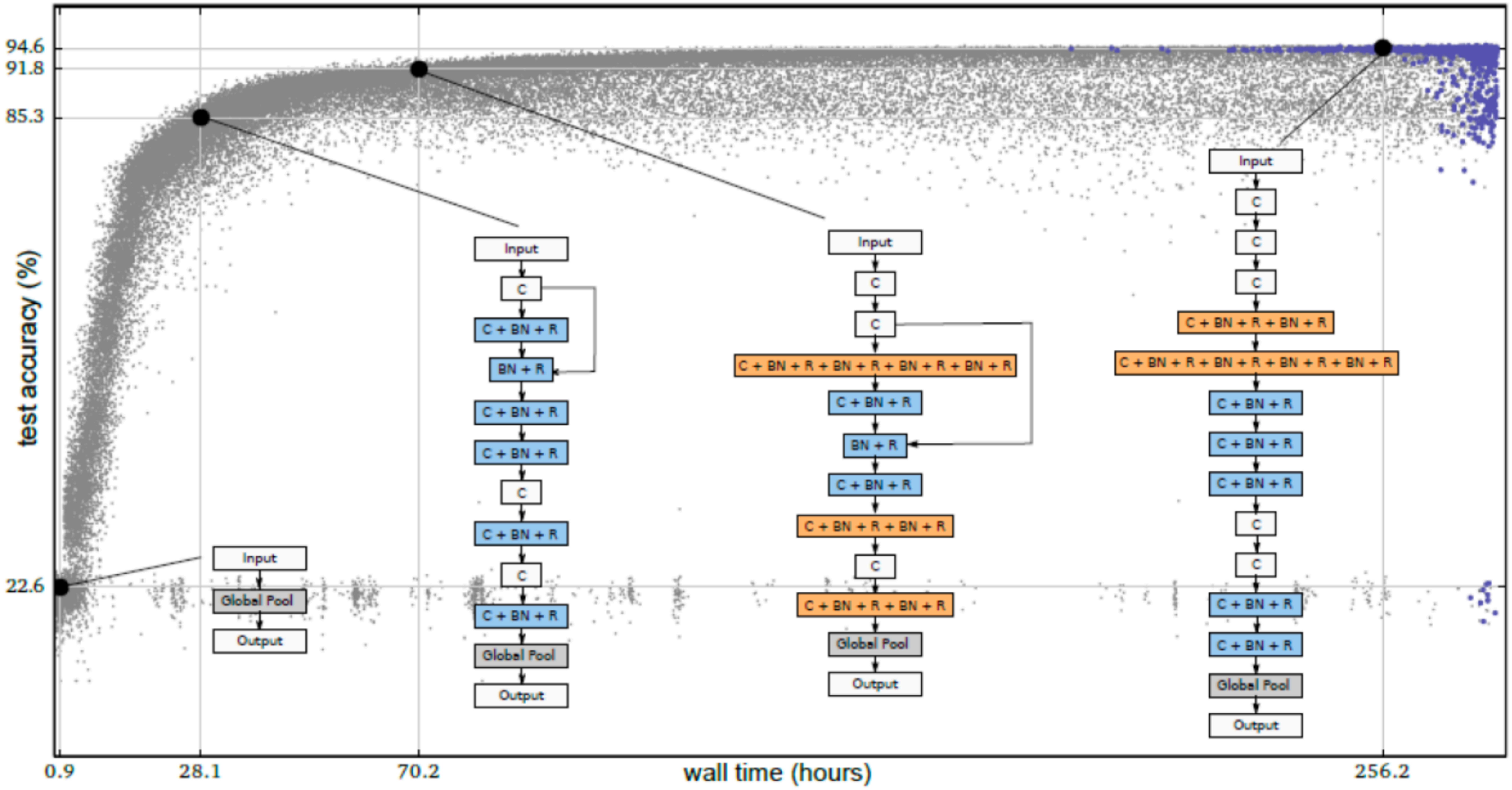


Neuro-evolution

Real et al 2017
Miikkulainen et al 2017
Wistuba et al 2018

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

alive, killed



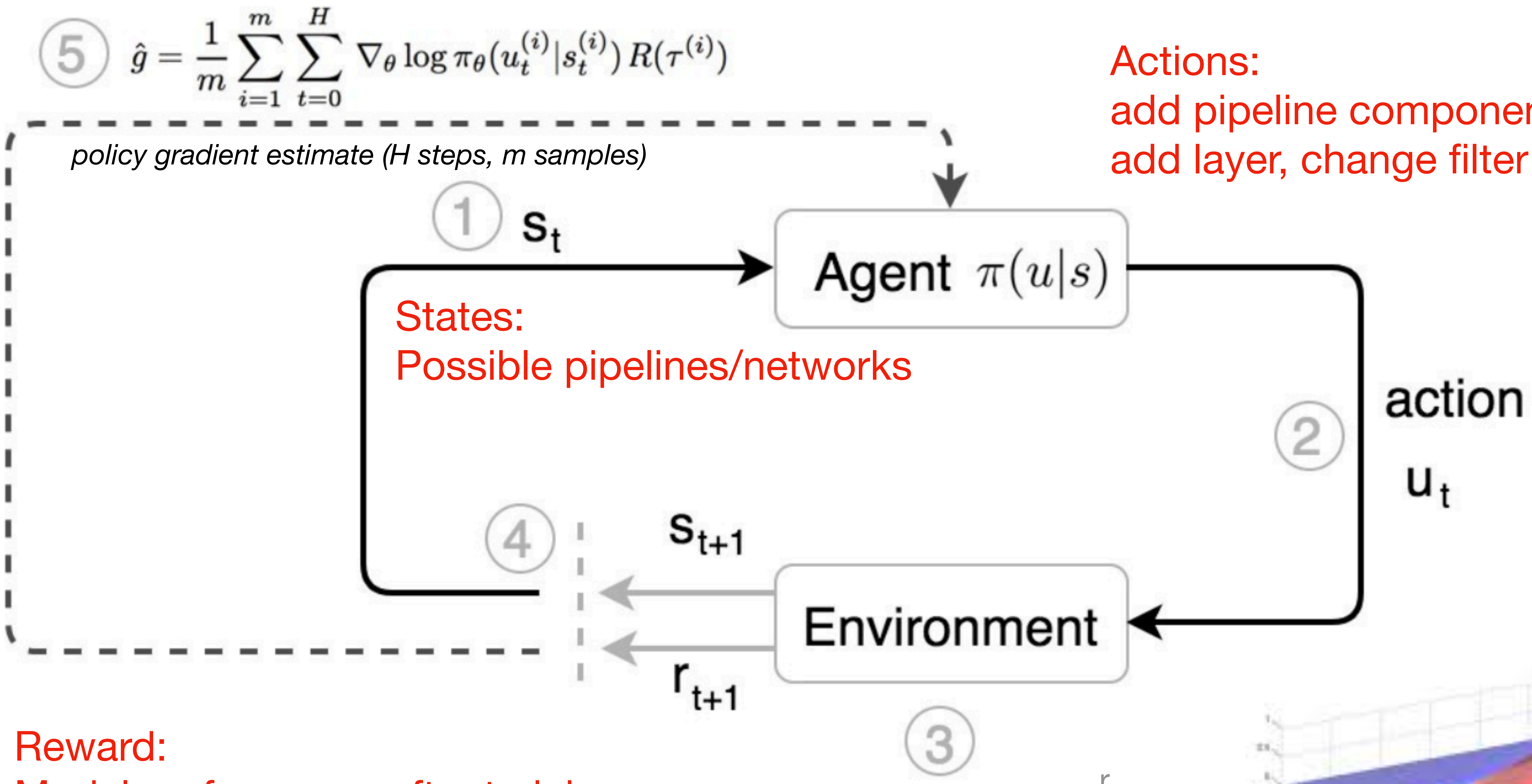
Optimization: Reinforcement learning

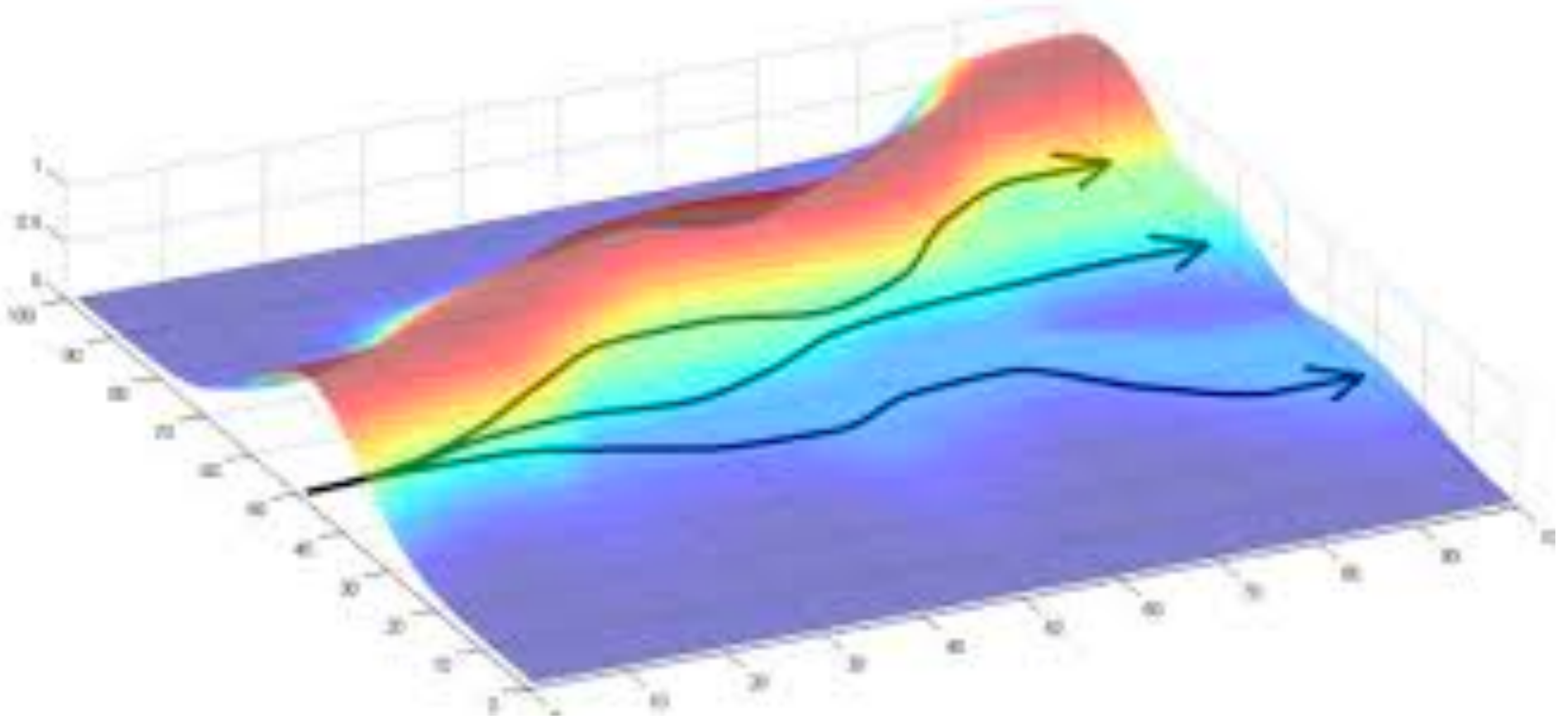
Zoph & Le 2017

Levine 2018

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

Actions:
add pipeline component, tune,…
add layer, change filter size,…





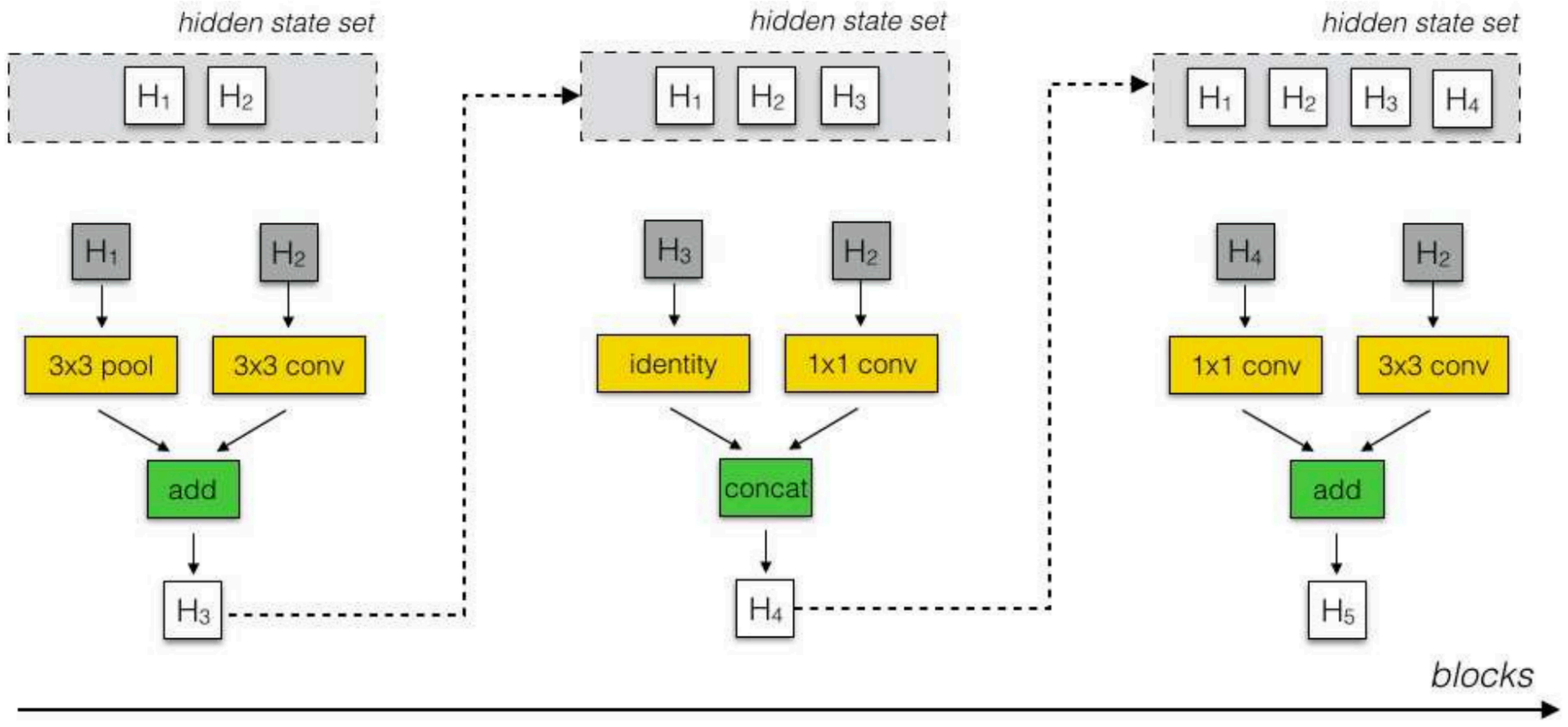
Reward:
Model performance after training
After trajectory τ of actions, adjust policy based on total reward $R(\tau)$

NAS with Reinforcement learning

Zoph et al 2018

1-layer LSTM (PPO), cell space search
- State of the art on ImageNet
- 450 GPUs, 3-4 days, 20000 architectures

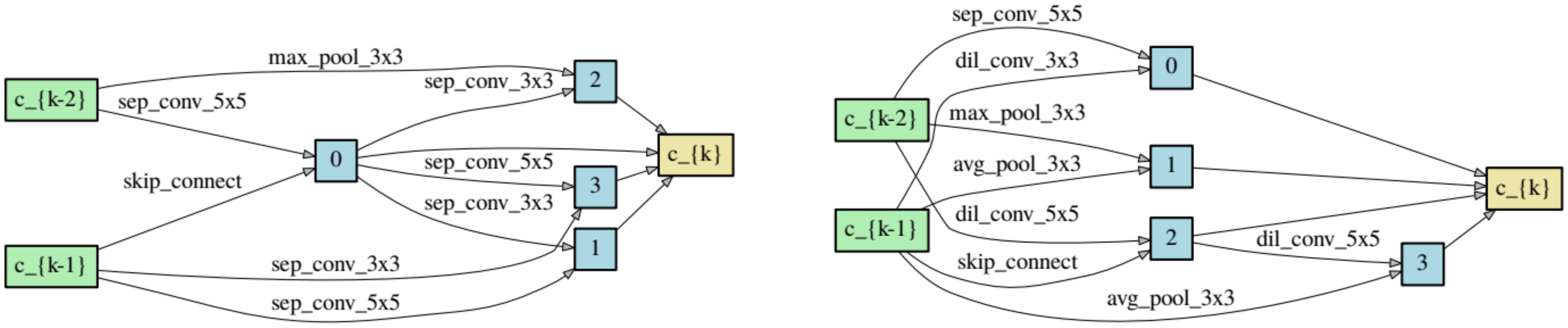
- Cell construction:
  - Select existing layers (hidden states, e.g. cell input) H_i to build on
  - Add operation (e.g. 3x3conv) on H_i
  - Combine into new hidden state (e.g. concat, add,…)
  - Iterate over *B* blocks



[Li & Talwalkar 2019](#)
[Yu et al. 2019](#)
[Real et al. 2019](#)

Sidenote: NAS with random search?

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



(a) Normal Cell

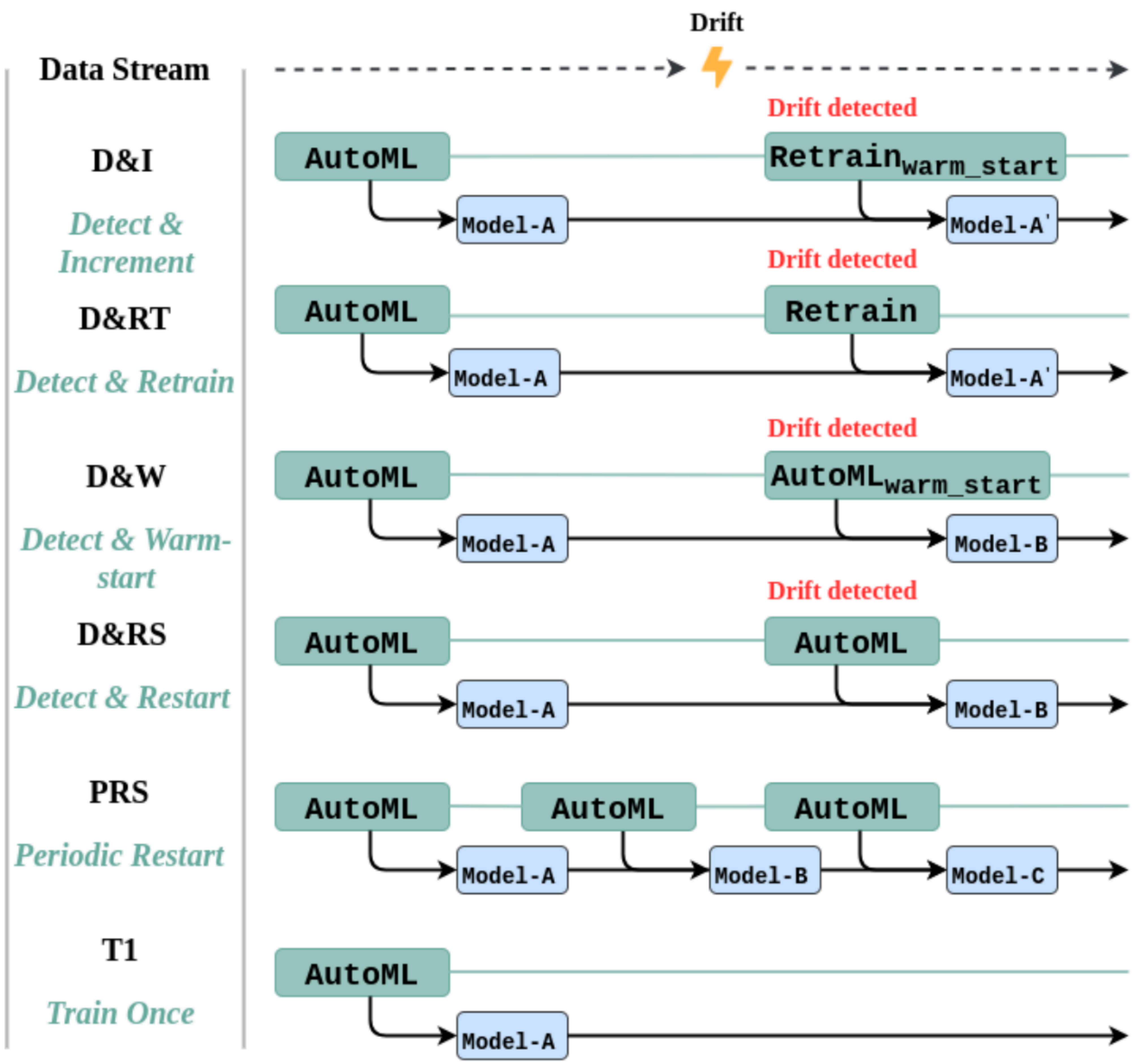
(b) Reduction Cell

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



Gradual Drift

Warm-start from (set of) last good solutions, where possible

High Abrupt Drift

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

Effect of Drift Type: High Gradual Drift

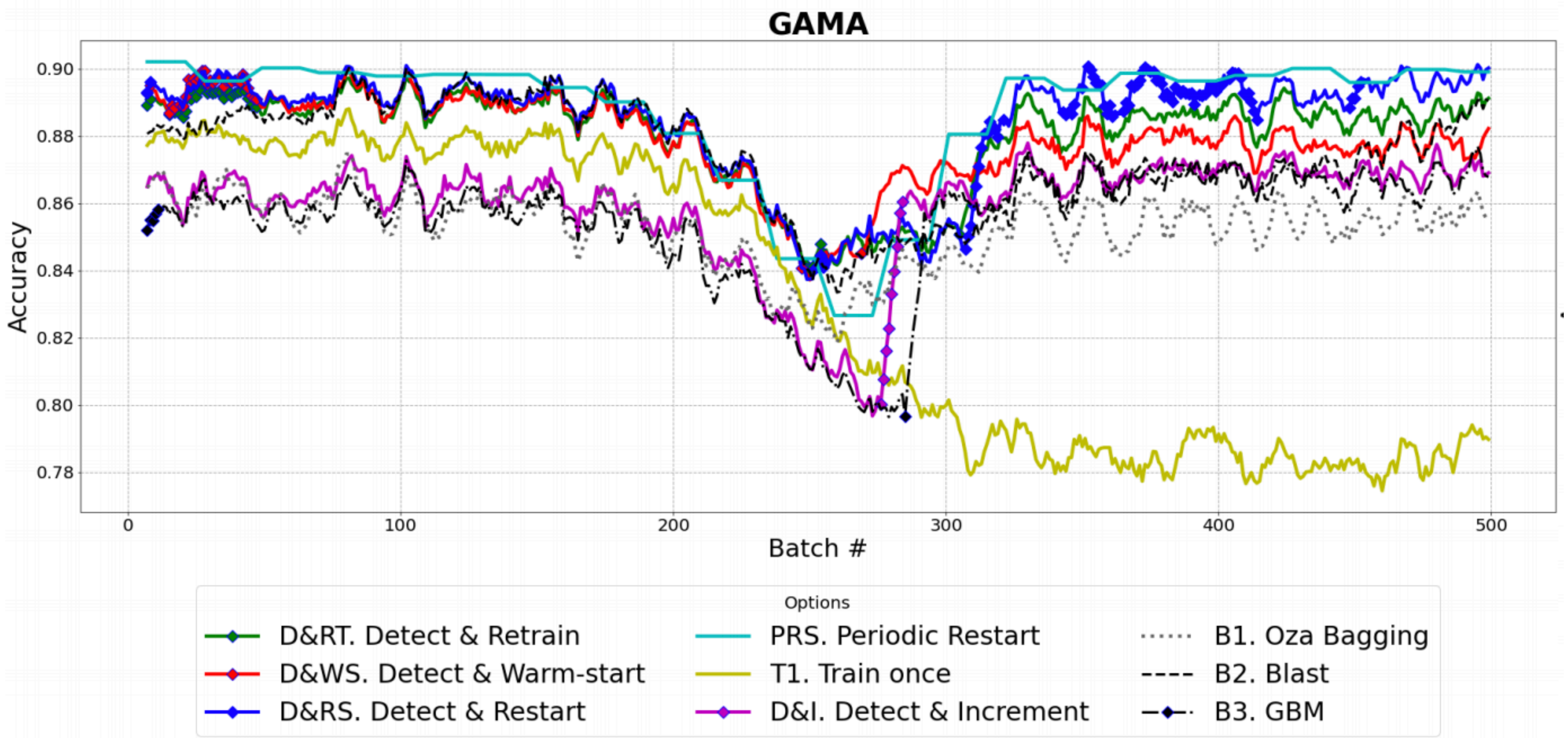
Retrain (don't reoptimize) pipelines from scratch
Incremental learning also works quite well

Effect of Drift Type: High Mixed Drift

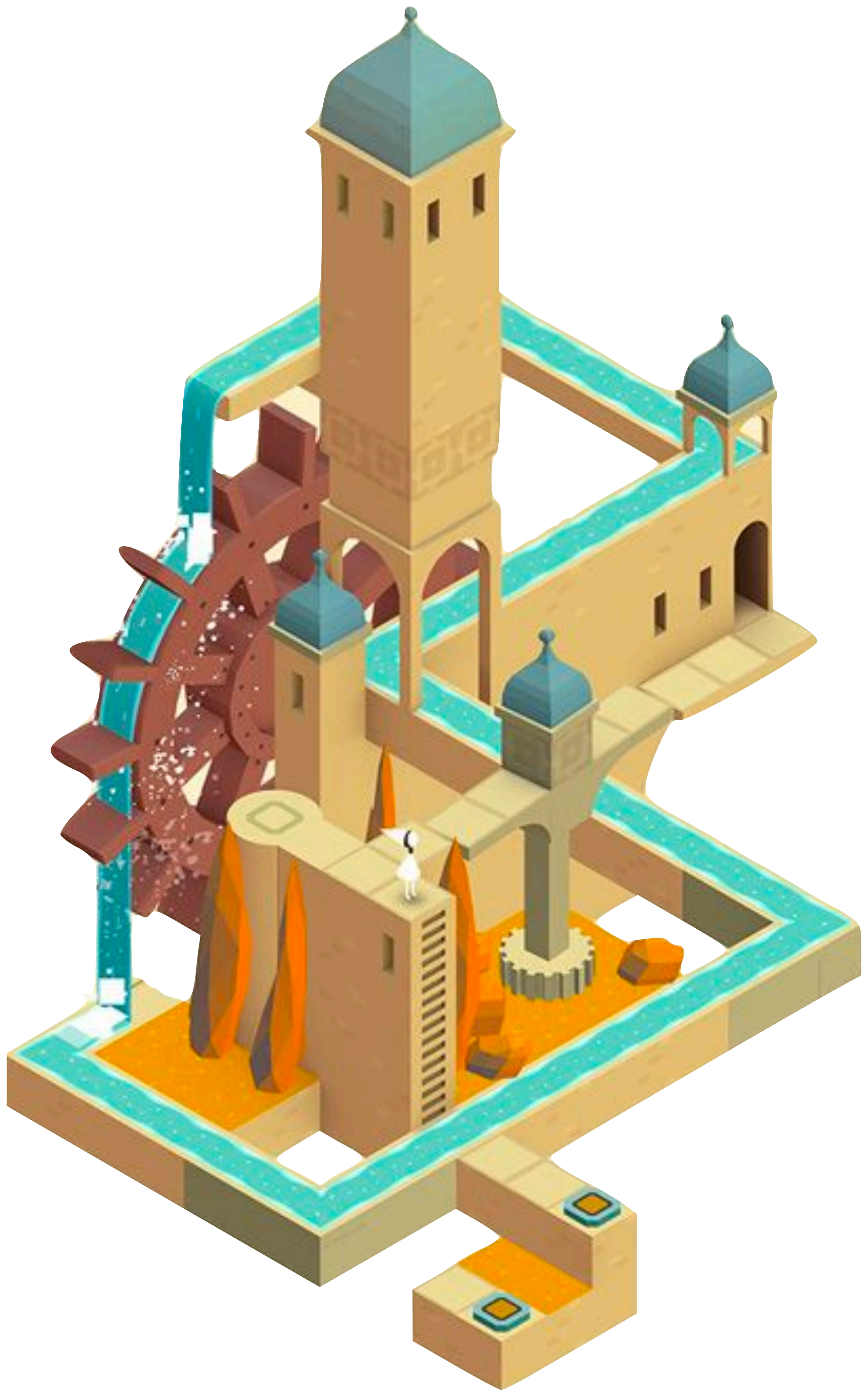
Re-optimize pipelines on new data (D&RS) performs best overall.

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



Overview



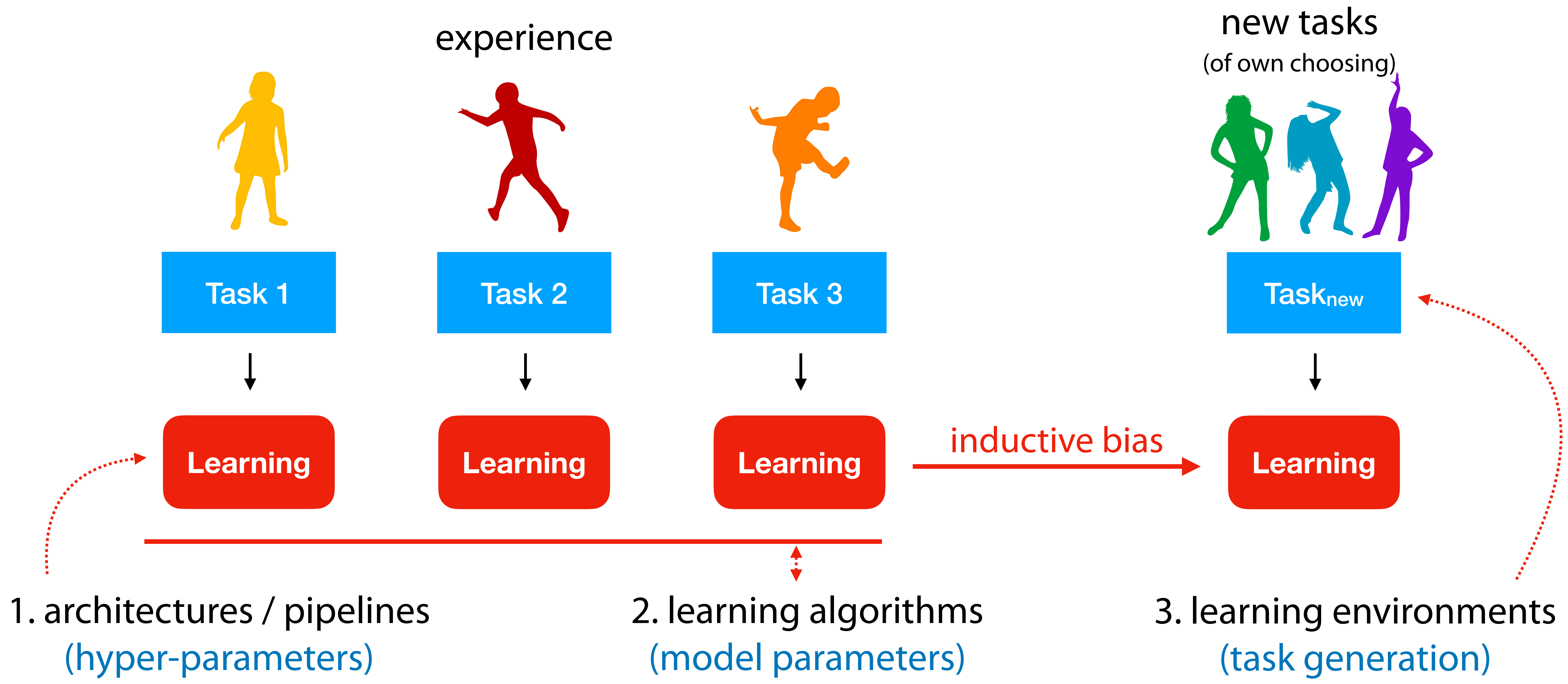
Part 3: Learning how to do AutoML

Closing the loop

Clune 2019

What can we learn to learn?

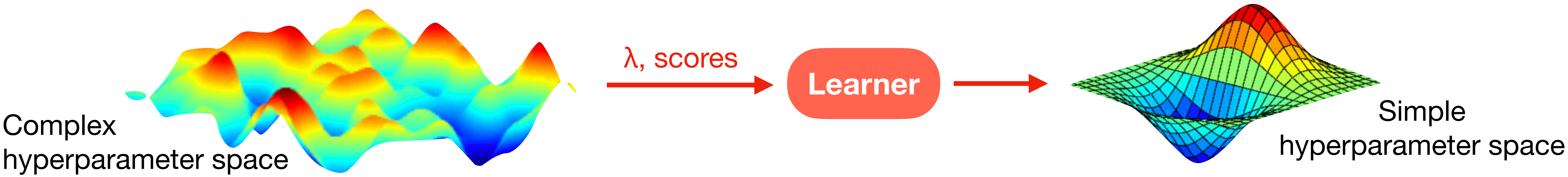
3 pillars



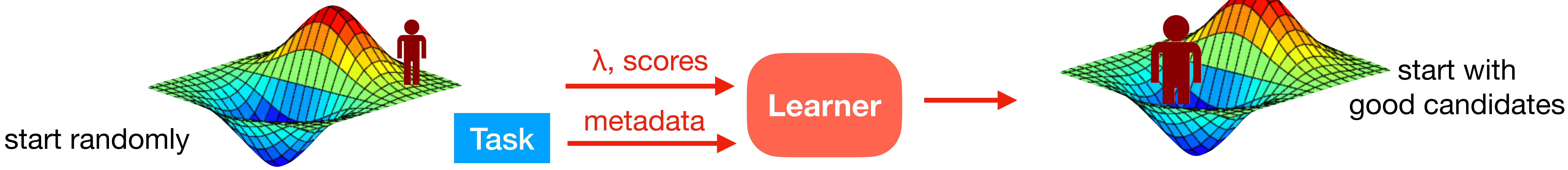
Meta-learning for AutoML: how?

hyperparameters = architecture + hyperparameters

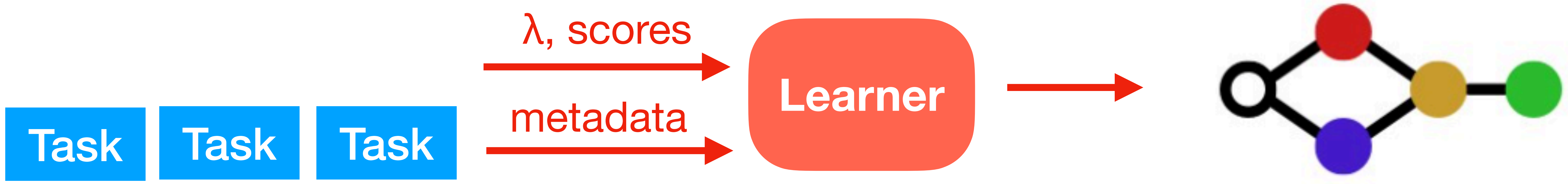
Learning hyperparameter priors (what works usually?)



Warm starting (what works on similar tasks?)

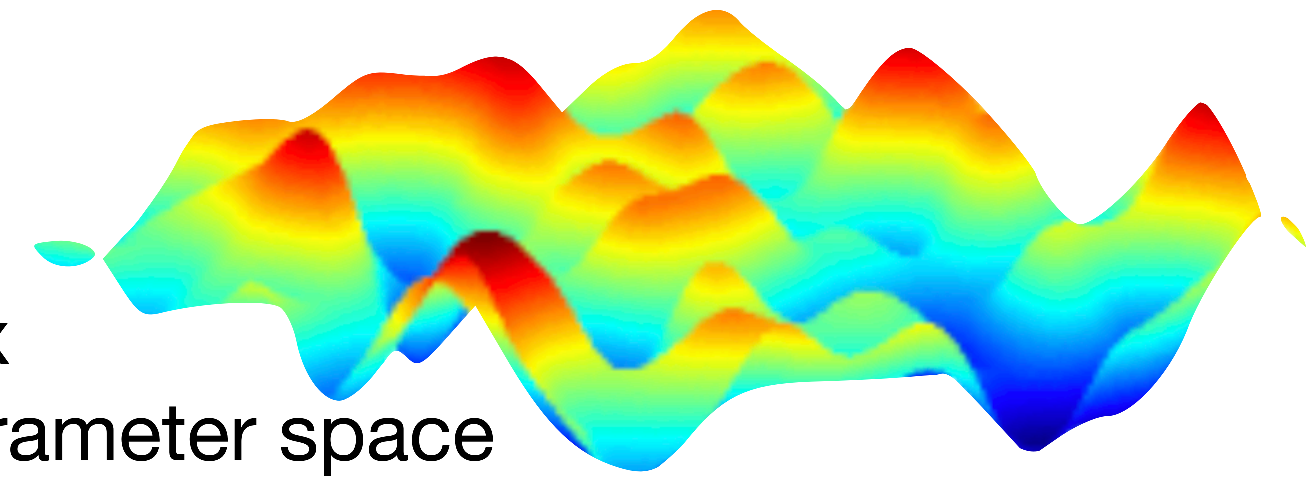


Meta-models (learn and predict useful models/components)

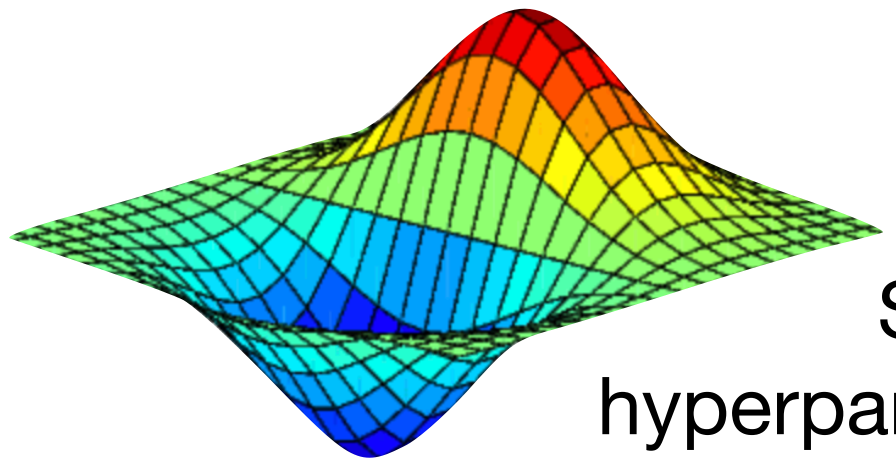


Observation:
current AutoML strongly depends on learned priors

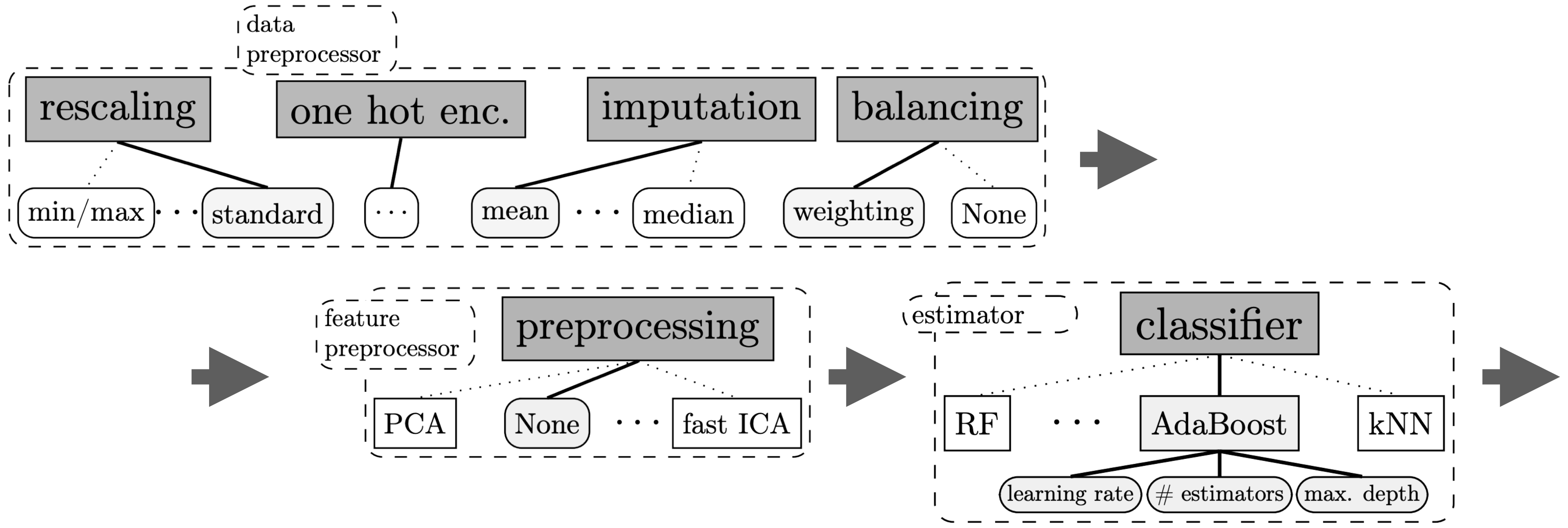
Complex
hyperparameter space

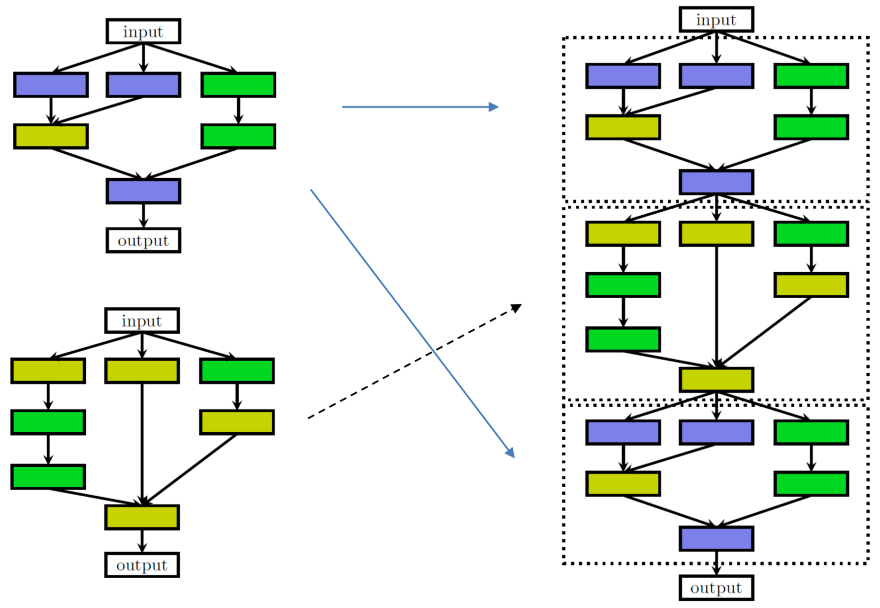


observation

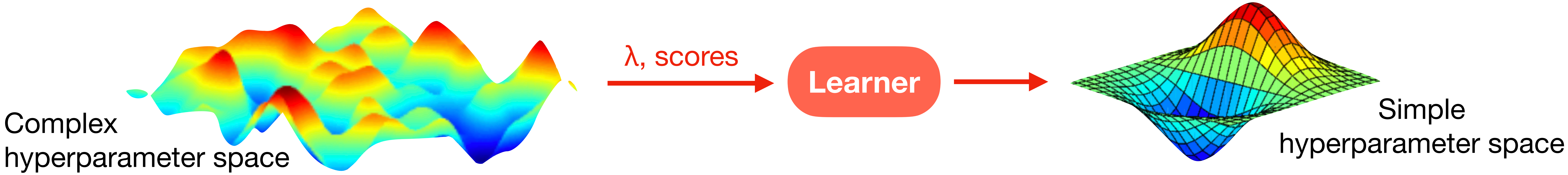


Simple
hyperparameter space



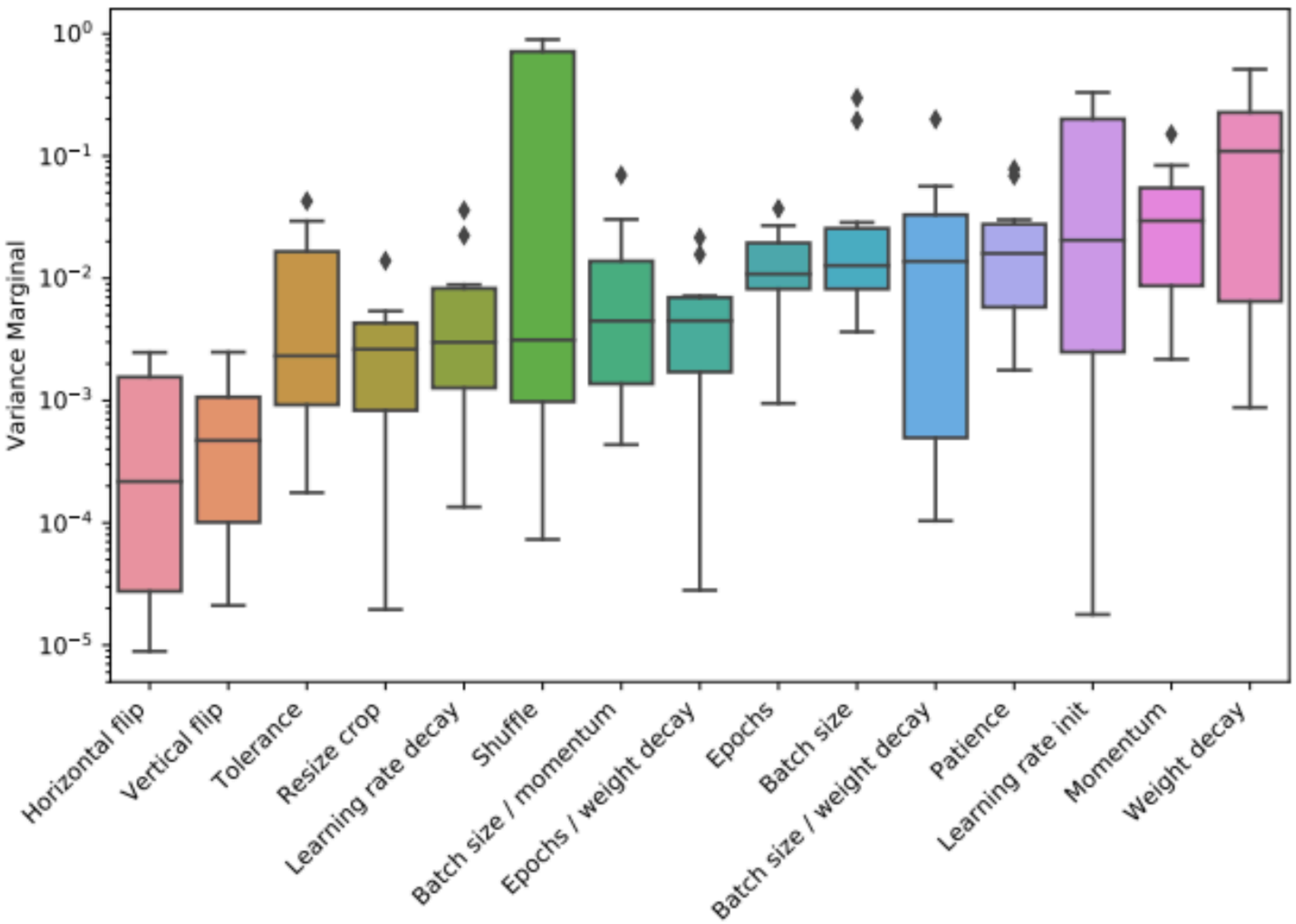


Can we *learn* good hyperparameter priors?



Learn hyperparameter importance

- **Functional ANOVA** [1]
  - Select hyperparameters that cause variance in the evaluations.
  - Useful to speed up black-box optimization techniques



ResNets for image classification

Figure source: van Rijn & Hutter, 2018

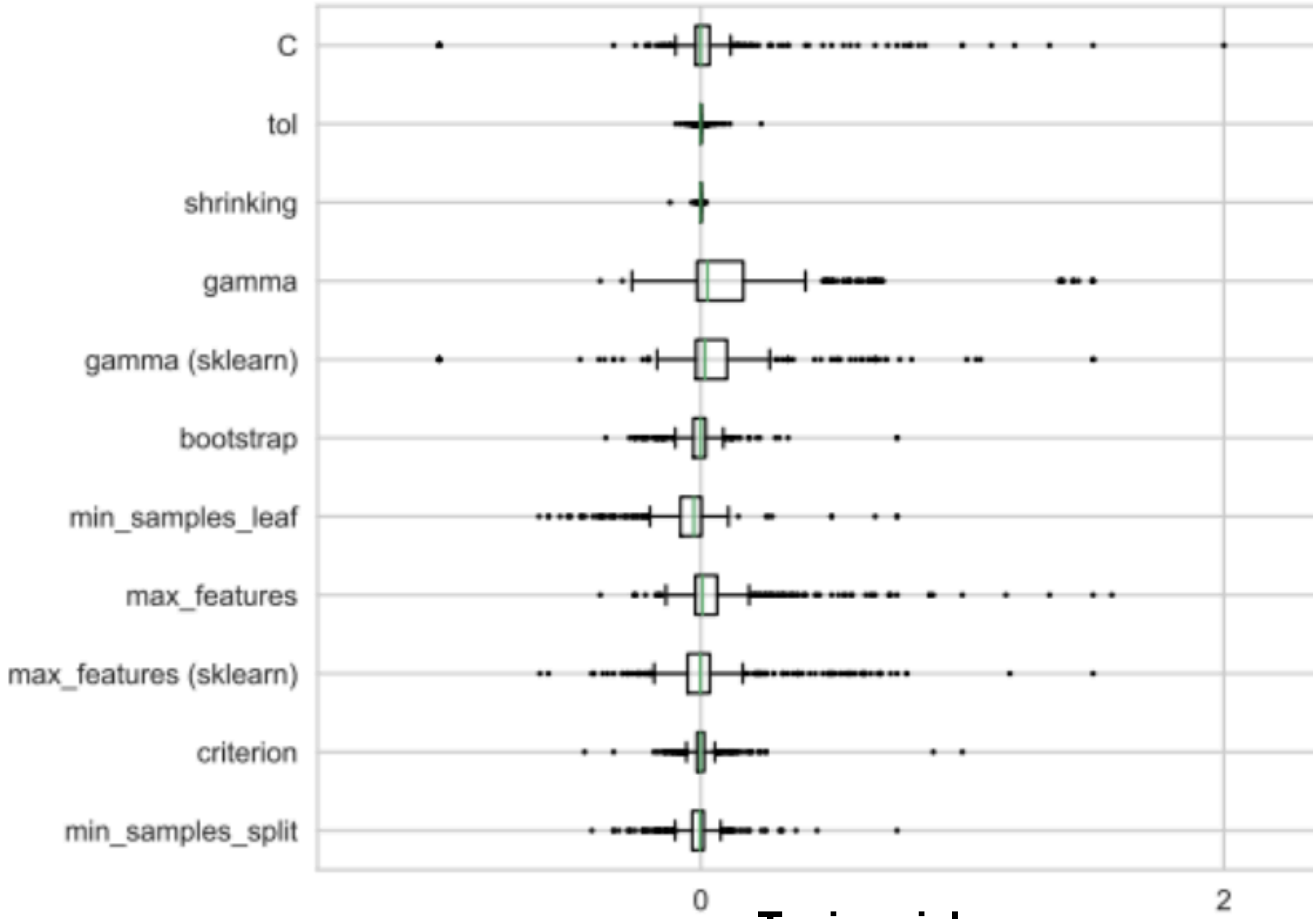
Learn defaults + hyperparameter importance

[1] Probst et al. 2018
[2] Weerts et al. 2018
[3] van Rijn et al. 2018
[4] Gijsbers et al. 2021

- **Tunability** [1,2,3]
  Learn good hyperparameter defaults. Is tuning to given task still needed?
  Learn ***symbolic defaults*** based on data properties [4]

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

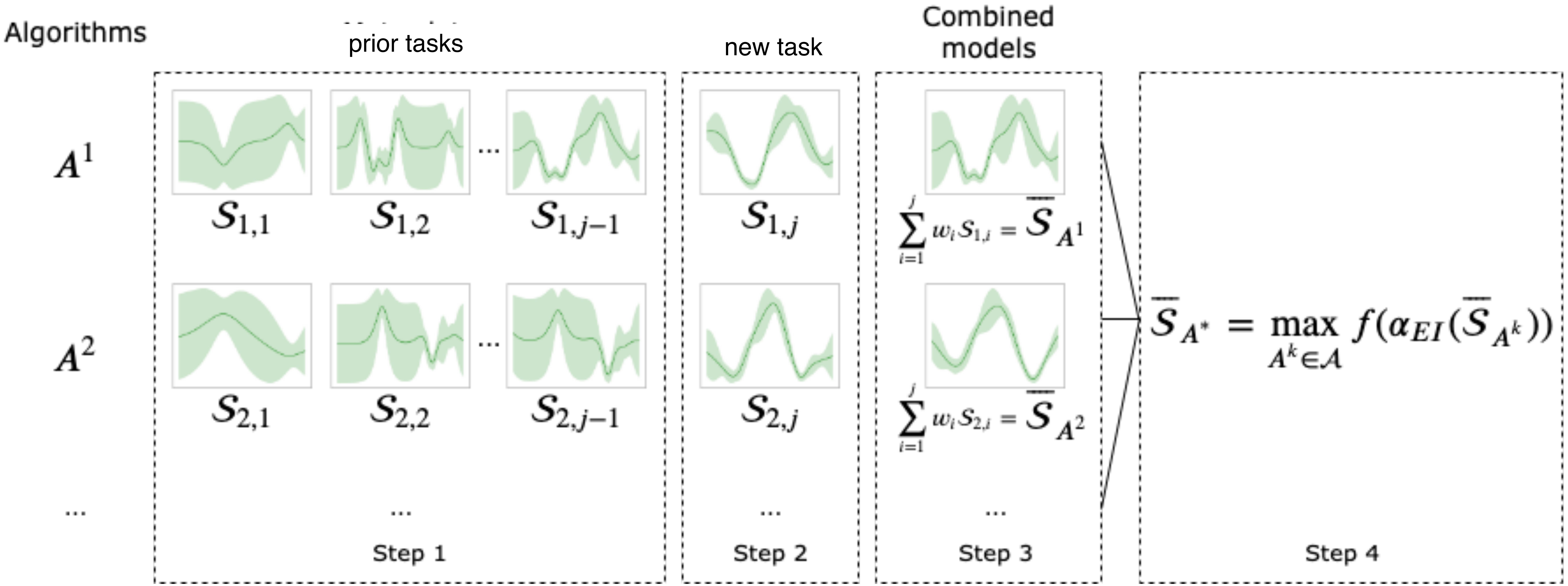


Tuning risk

Manolache & Vanschoren 2019

Surrogate model transfer

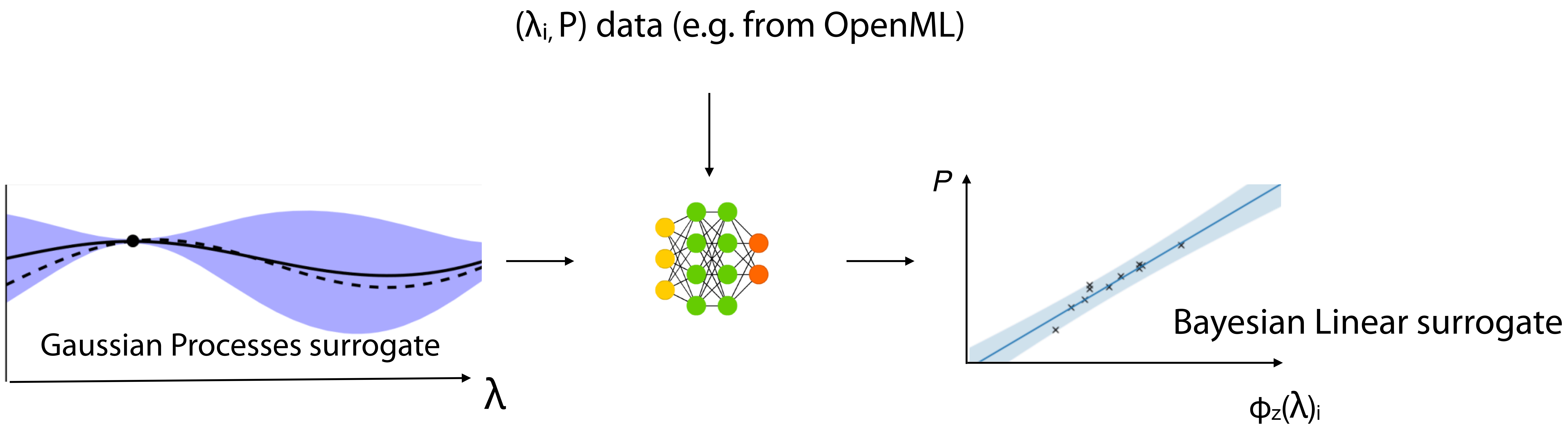
- If task j is similar to the new task, its surrogate model S_j will likely transfer well
- Sum up all S_j predictions, weighted by task similarity
- Build combined surrogate, weighted by current performance on new task[2]



Perrone et al. 2018

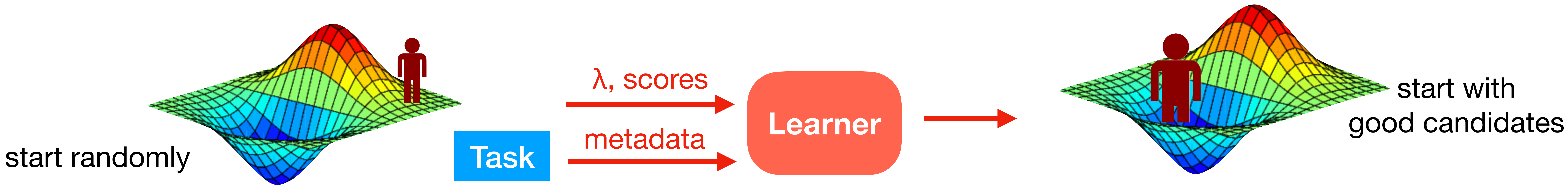
Deep learning models for hyperparameter spaces

- Use a neural net to learn a simpler hyperparameter search space
- Used in SageMaker AutoML



Warm starting

(what works on similar tasks?)



How to measure task similarity?

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

Figure source: Alvarez-Melis et al. 2020

[1] Vanschoren 2018
[2] Achille et al. 2019
[3] Alvarez-Melis et al. 2020
[4] Drori et al. 2019
[5] Jooma et al. 2020
[6] de Bie et al. 2020

Warm-starting with kNN

- Find k most similar tasks, warm-start search with best λ_i (instead of random points)
  - Auto-sklearn: warm-started Bayesian optimization
    - Meta-learning yield better models, faster
    - Winner of several AutoML Challenges

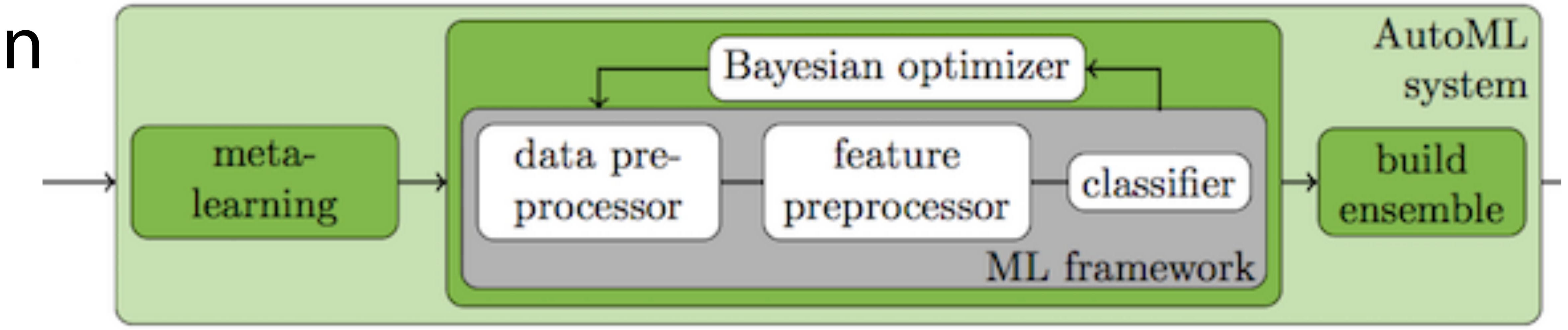
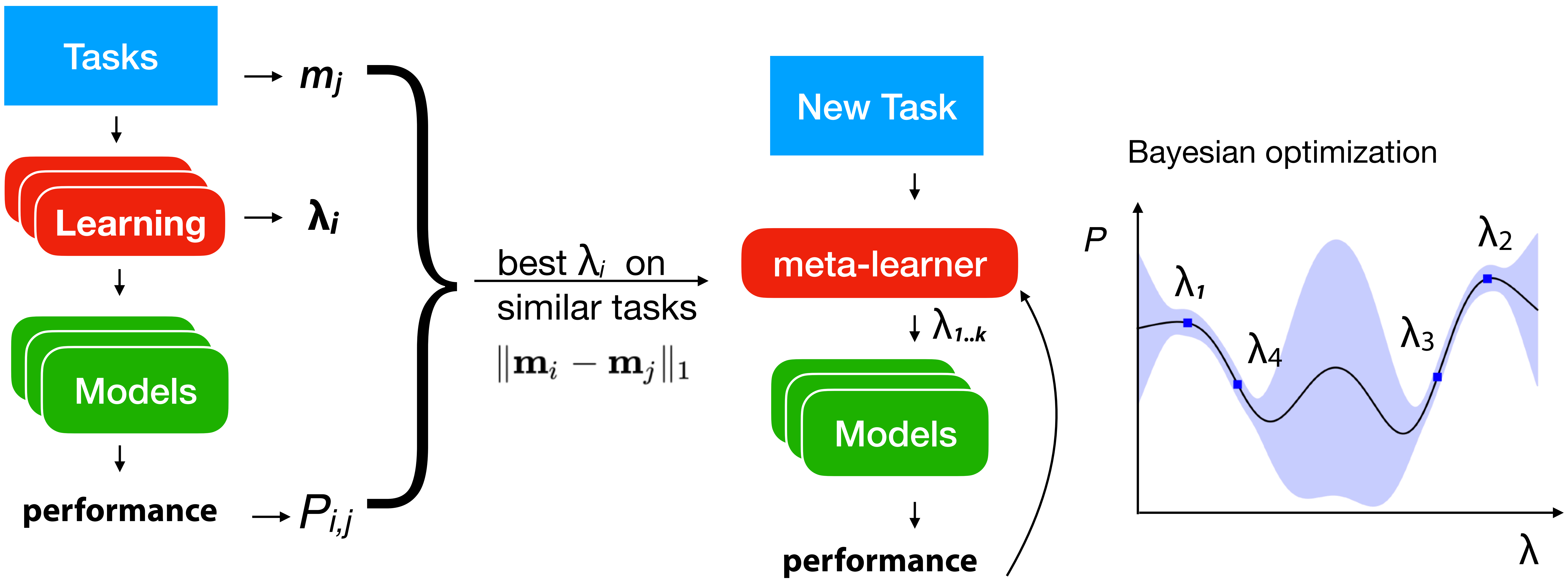


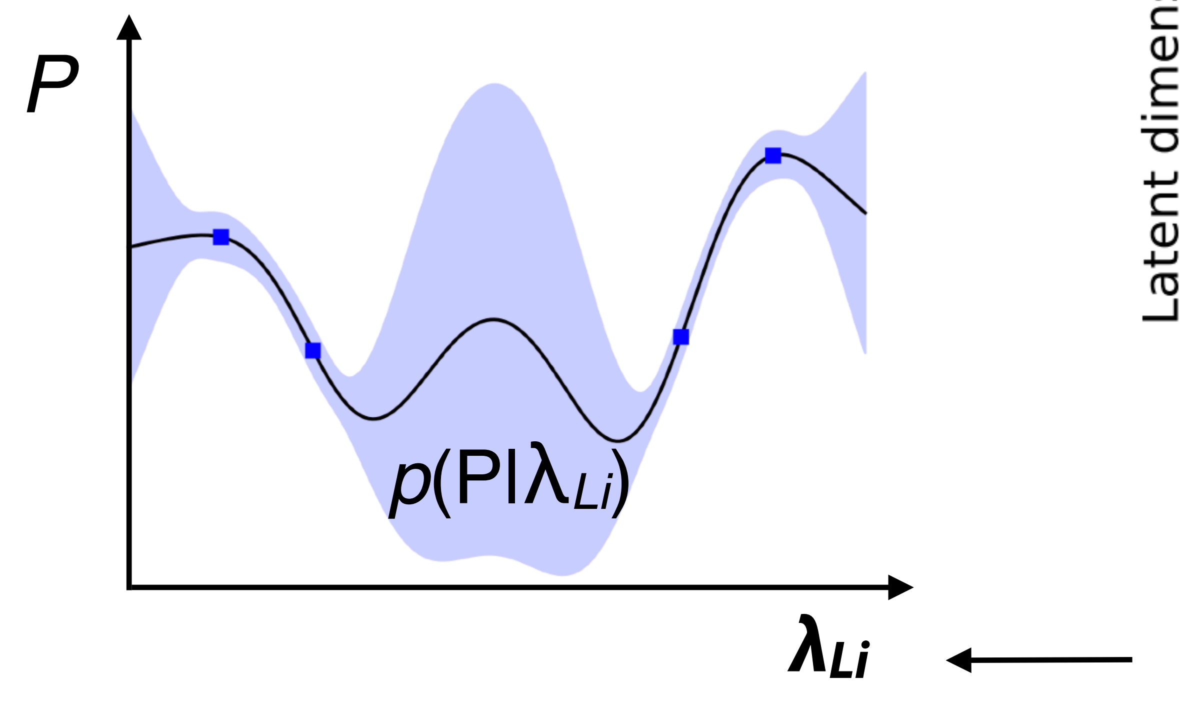
Figure source: Feurer et al., 2015



Probabilistic Matrix Factorization

- Collaborative filtering: configurations λ_i are `rated' by tasks t_j

- Learn latent representation for tasks T and configurations λ
- Use meta-features to warm-start on new task
- Returns probabilistic predictions for Bayesian optitmization
- Used in Azure AutoML



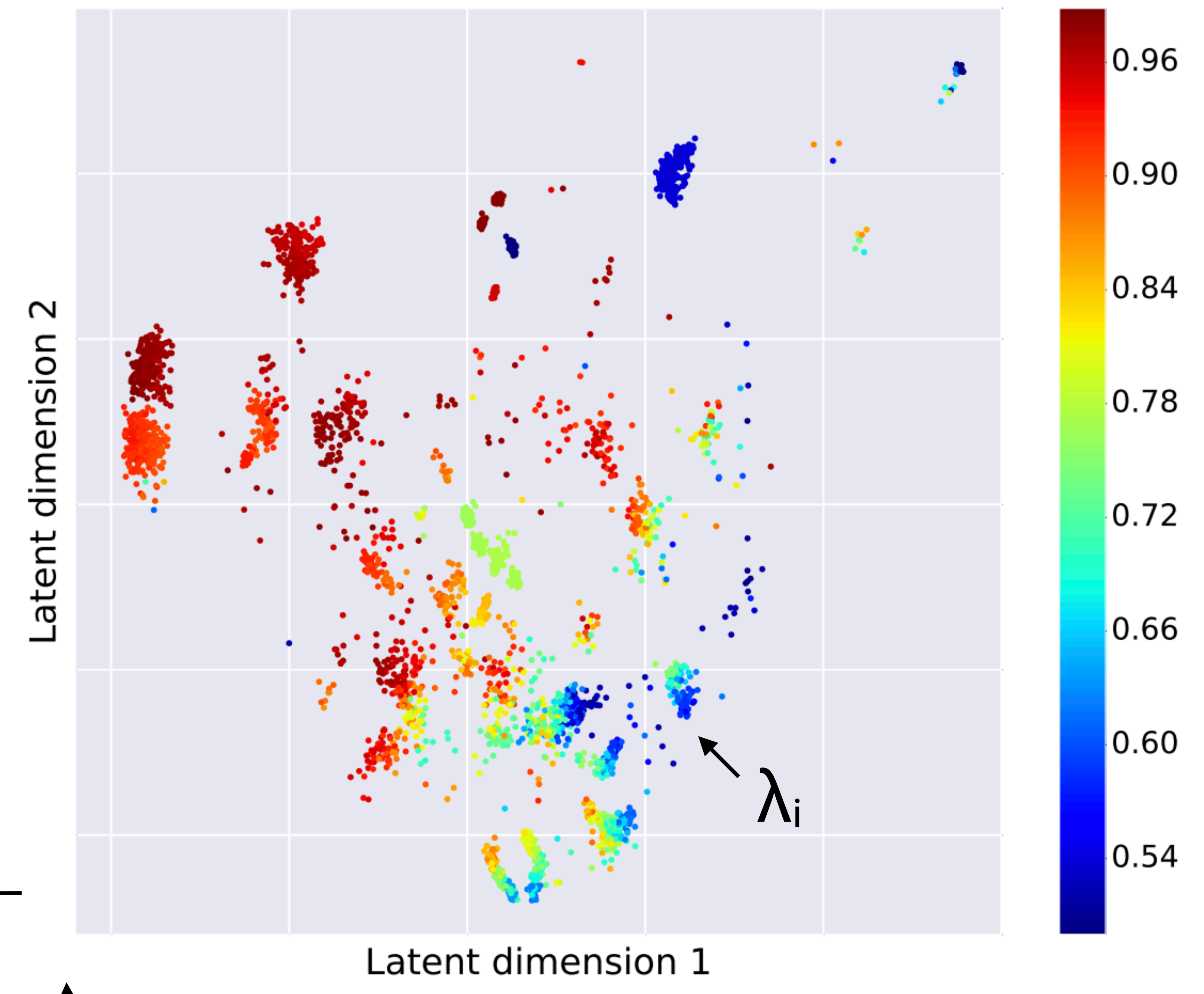
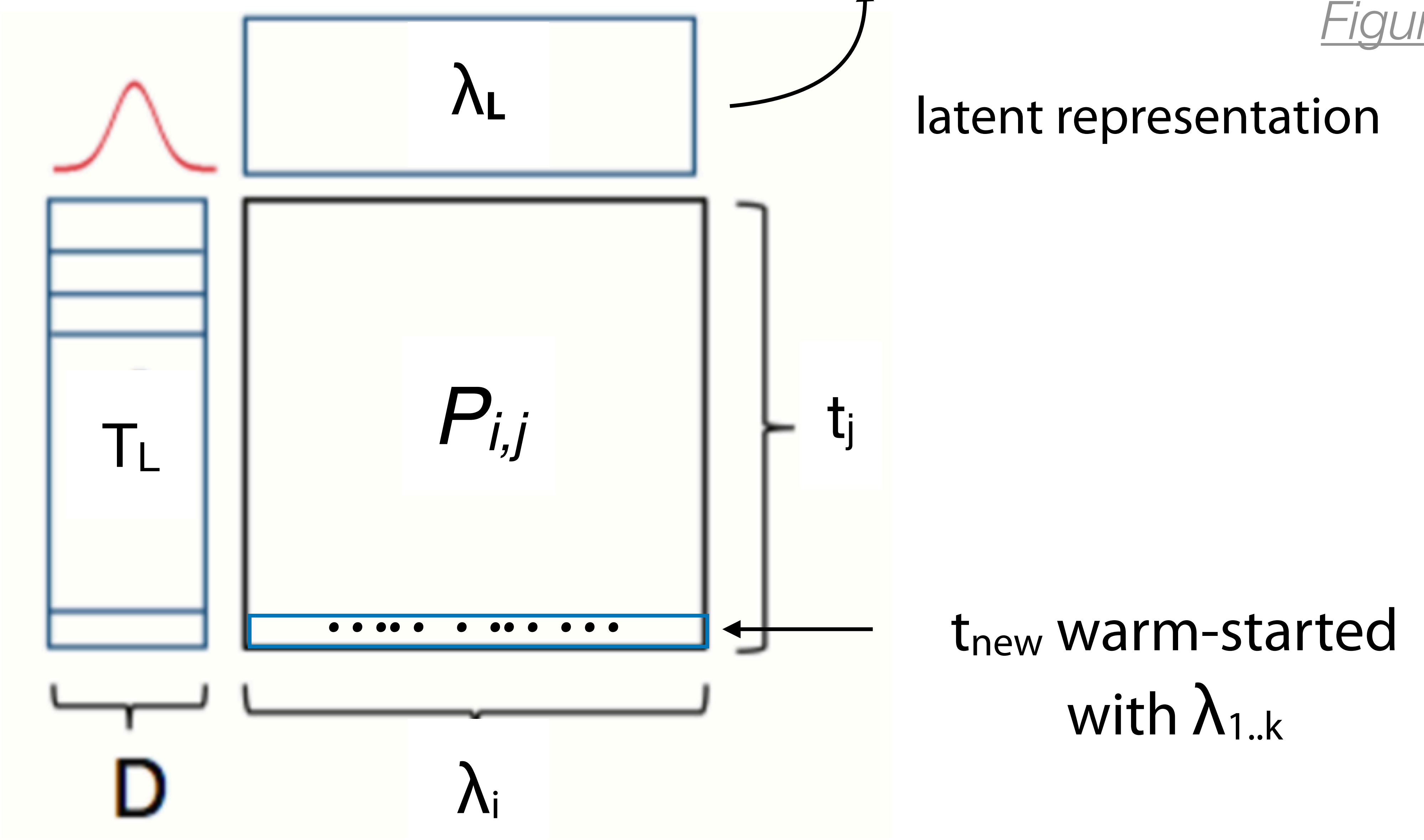
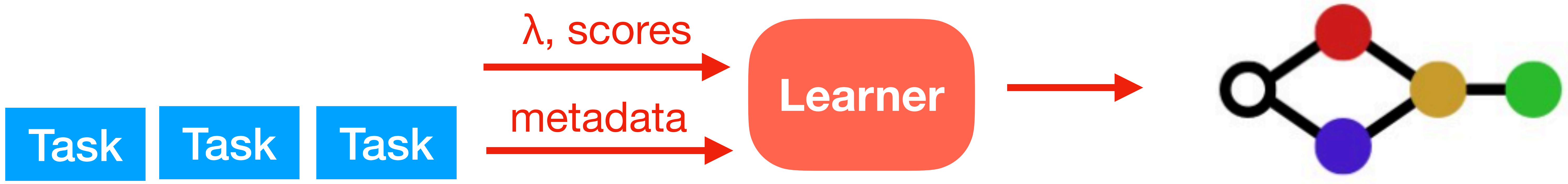


Figure source: Fusi et al., 2017



Meta-models

(learn how to build models/components)



Meta-learning = how to learn better for specific ***types*** of (real world) tasks

Algorithm selection models

[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

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

$$m_j \longrightarrow \boxed{\text{meta-learner}} \longrightarrow \lambda_{best}$$

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

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

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

$$m_j \longrightarrow \boxed{\text{meta-learner}} \longrightarrow \boldsymbol{\Lambda}$$

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

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

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

[1] Ramachandran et al. 2017

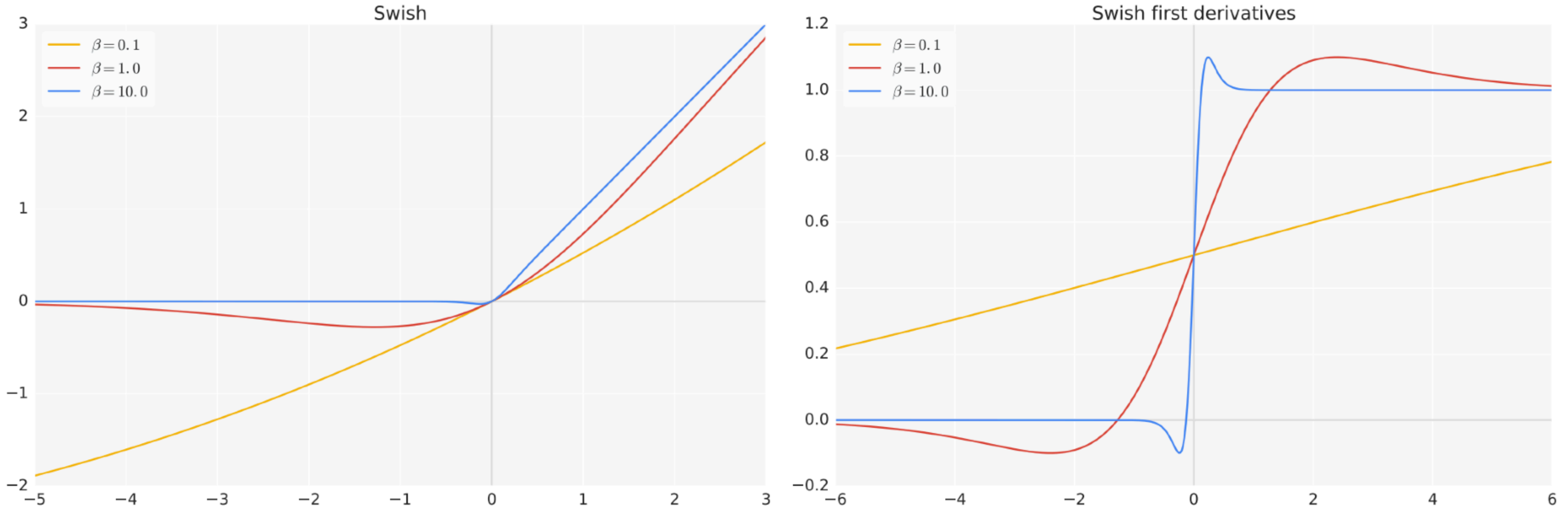
[2] Bello et al. 2017

[3] Volpp et al. 2020

Learning model components

- Learn nonlinearities: RL-based search of space of likely useful activation functions [1]
  - E.g. Swish can outperform ReLU

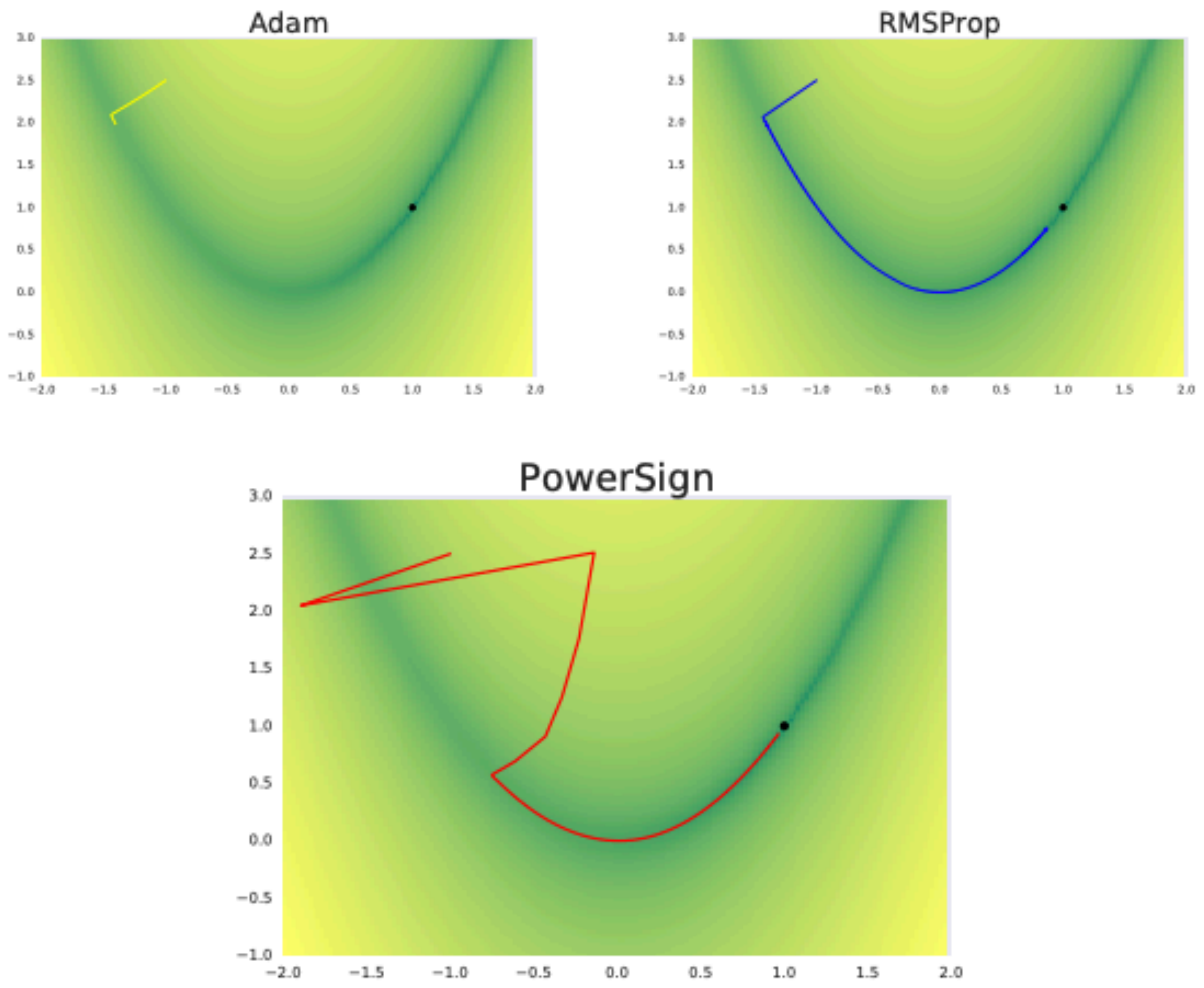
$$\textit{Swish}: \frac{x}{1 + e^{-\beta x}}$$



- Learn optimizers: RL-based search of space of likely useful update rules [2]
  - E.g. PowerSign can outperform Adam, RMPprop

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

g: gradient, m:moving average



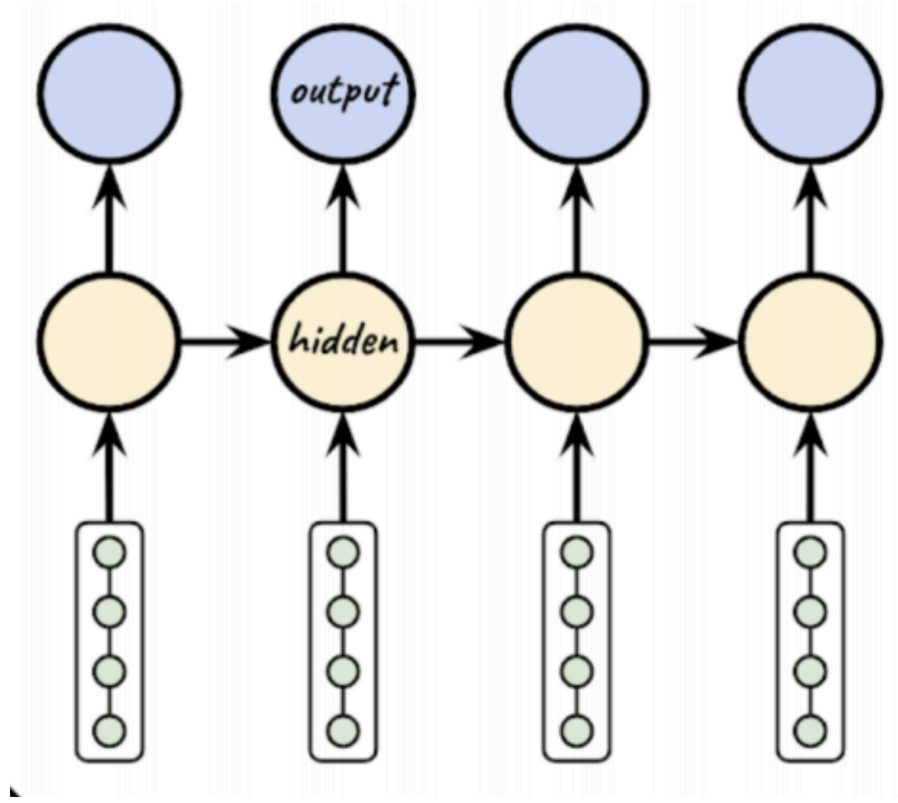
- Learn acquisition functions for Bayesian optimization [3]

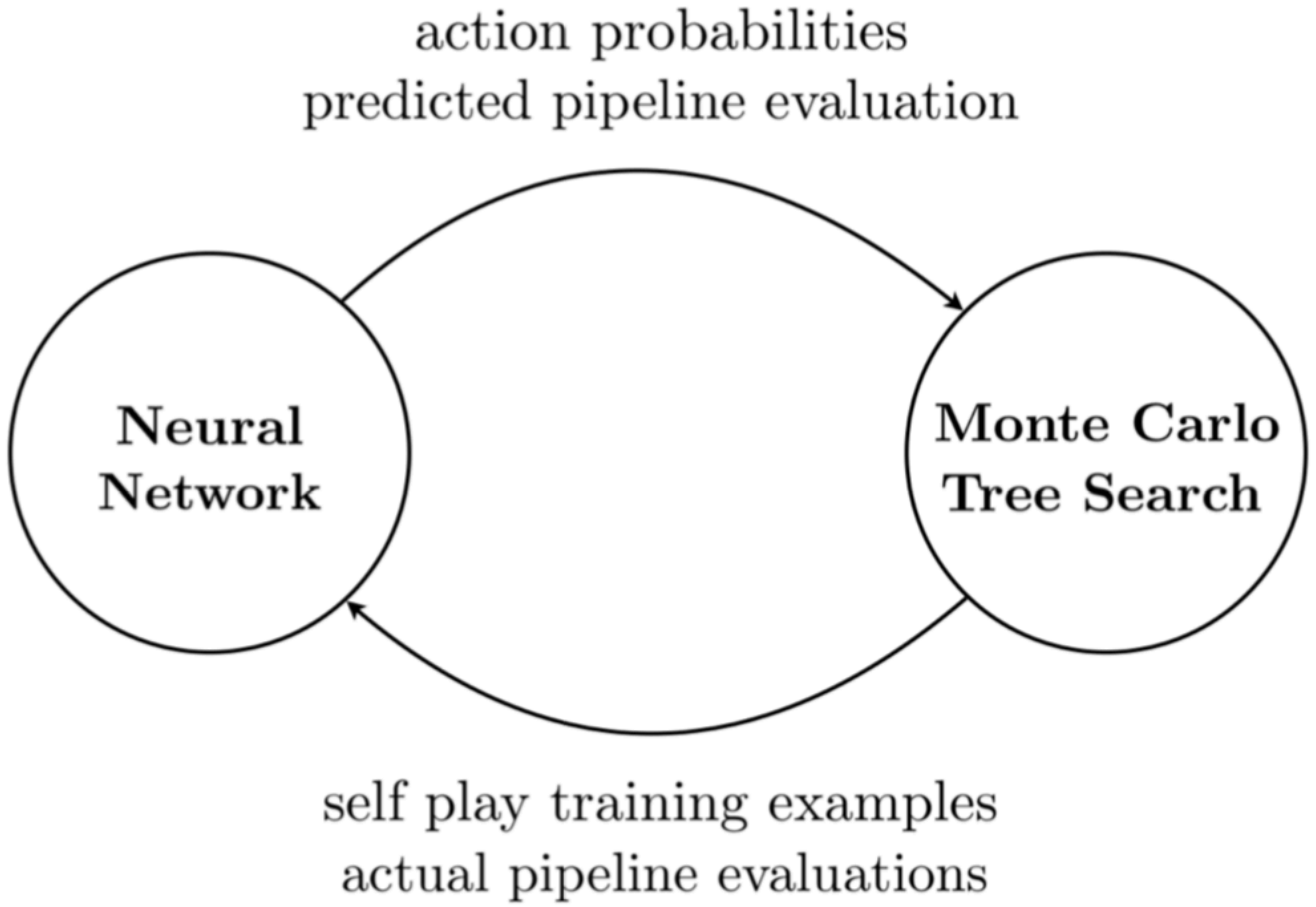
Figure source: Ramachandran et al., 2017 (top), Bello et al. 2017 (bottom)

MOSAIC [Rakotoarison et al. 2019]
AlphaD3M [Drori et al. 2019]

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





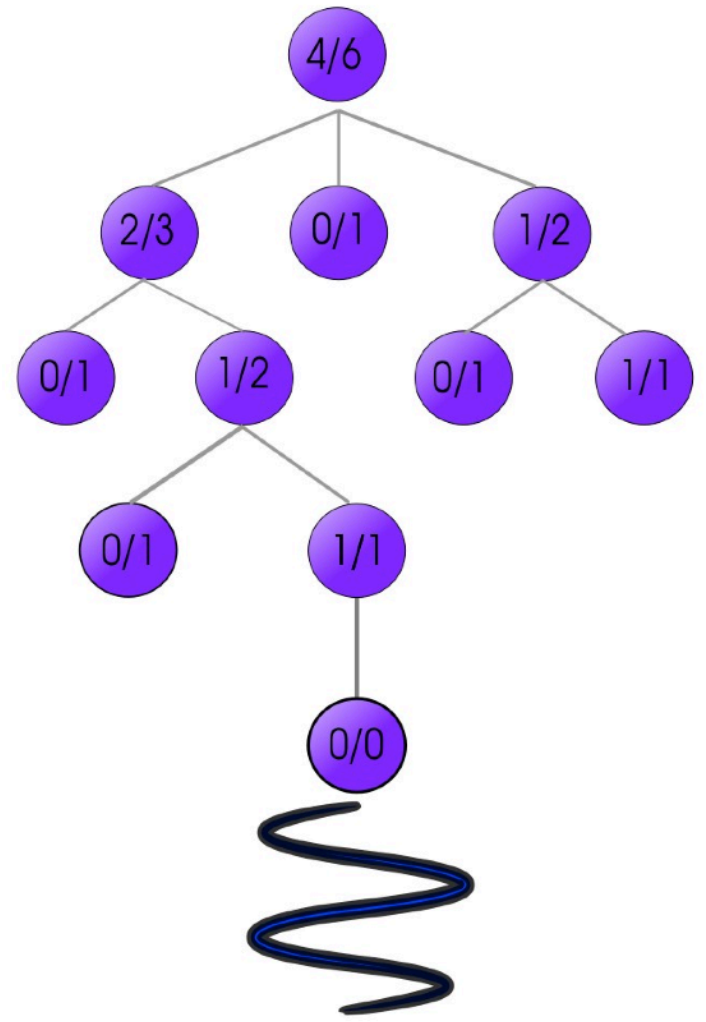


Figure source: Drori et al., 2019

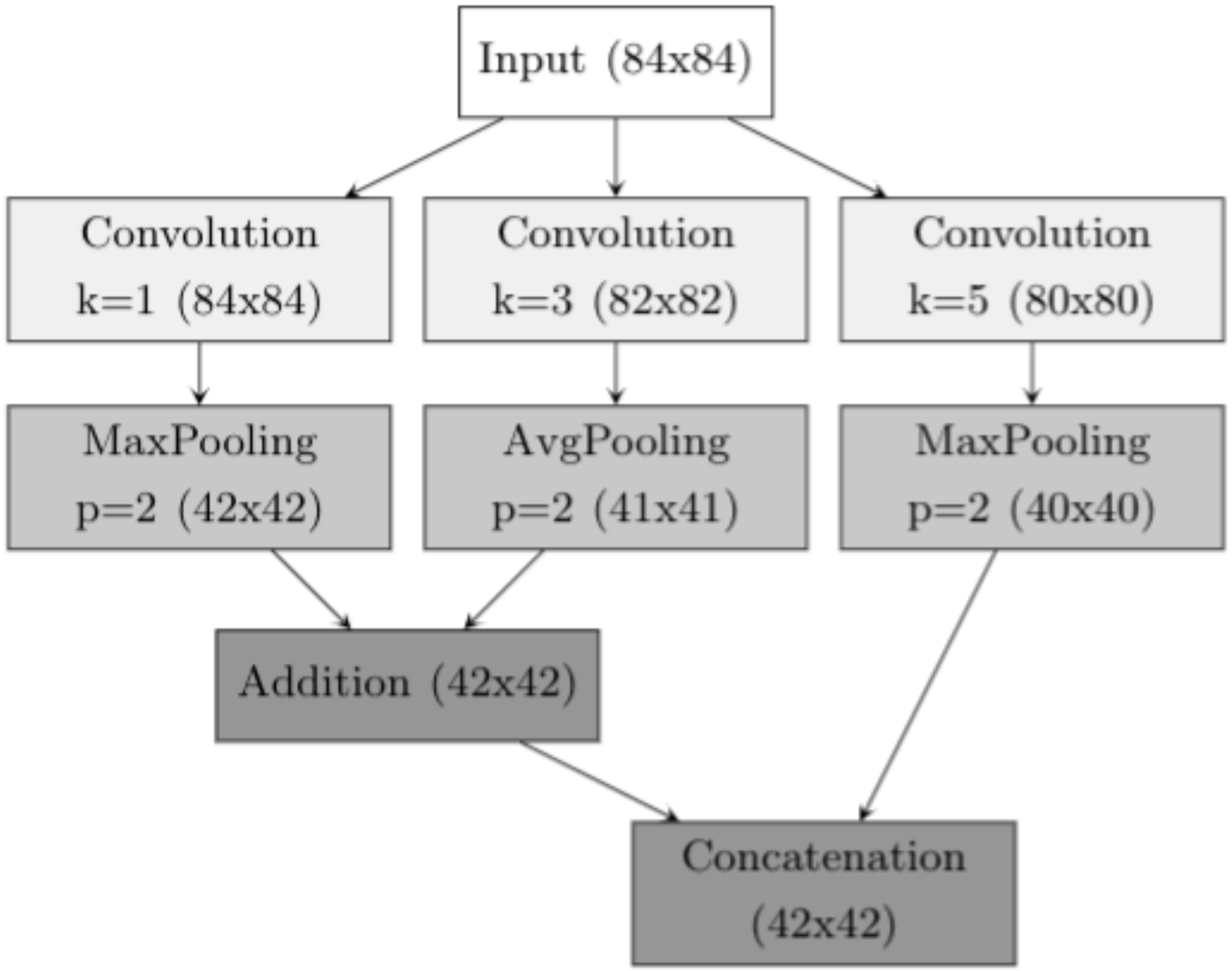
Gomez & Vanschoren, 2019

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/remove certain layers in certain locations

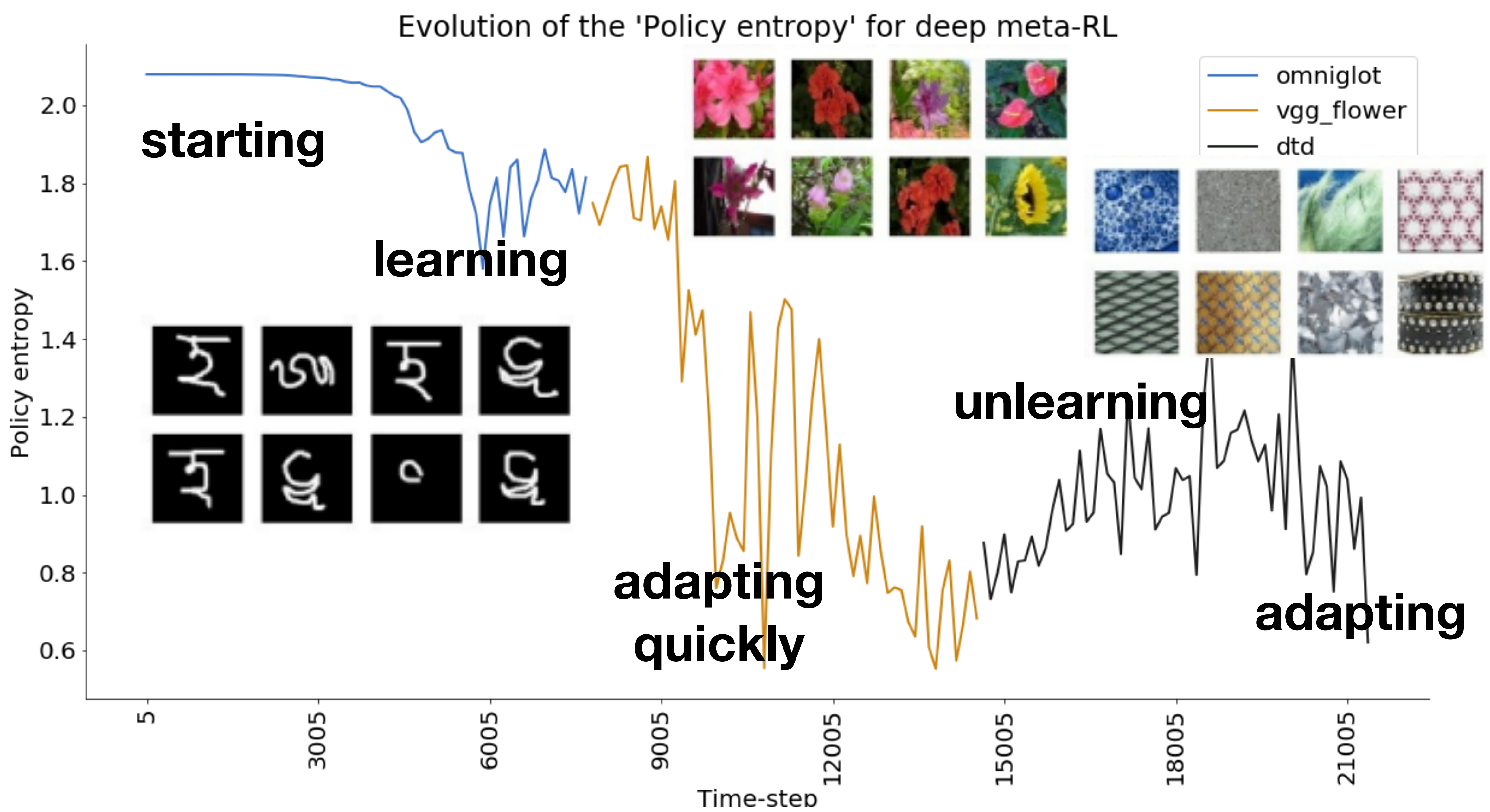


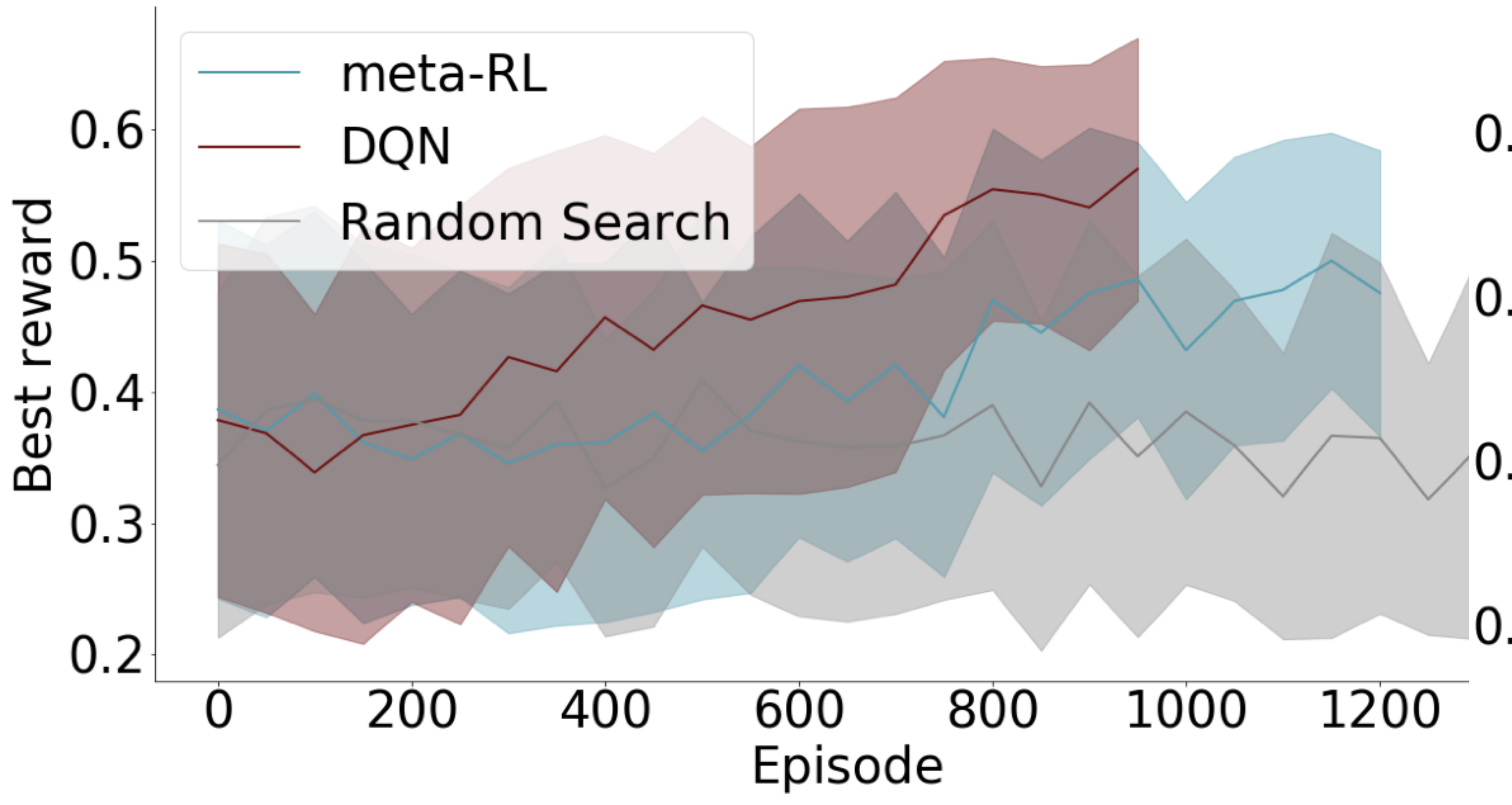
Meta-Reinforcement Learning for NAS

Gomez & Vanschoren, 2019

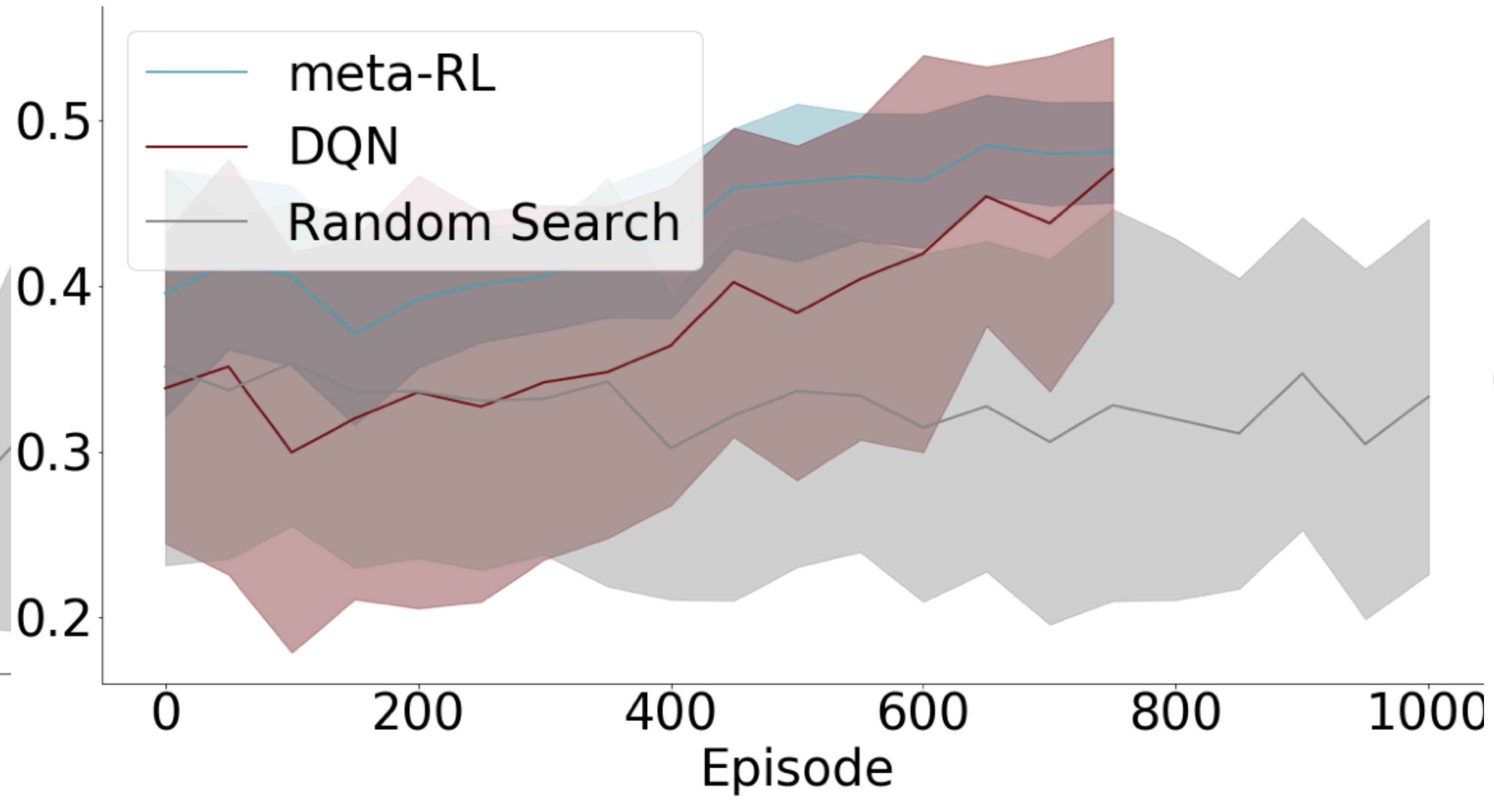
Image classifiers for increasingly difficult tasks:

- Initially slower than normal RL techniques, but faster after a few tasks
- Policy entropy (agent predictability) shows learning, forgetting, re-learning,….

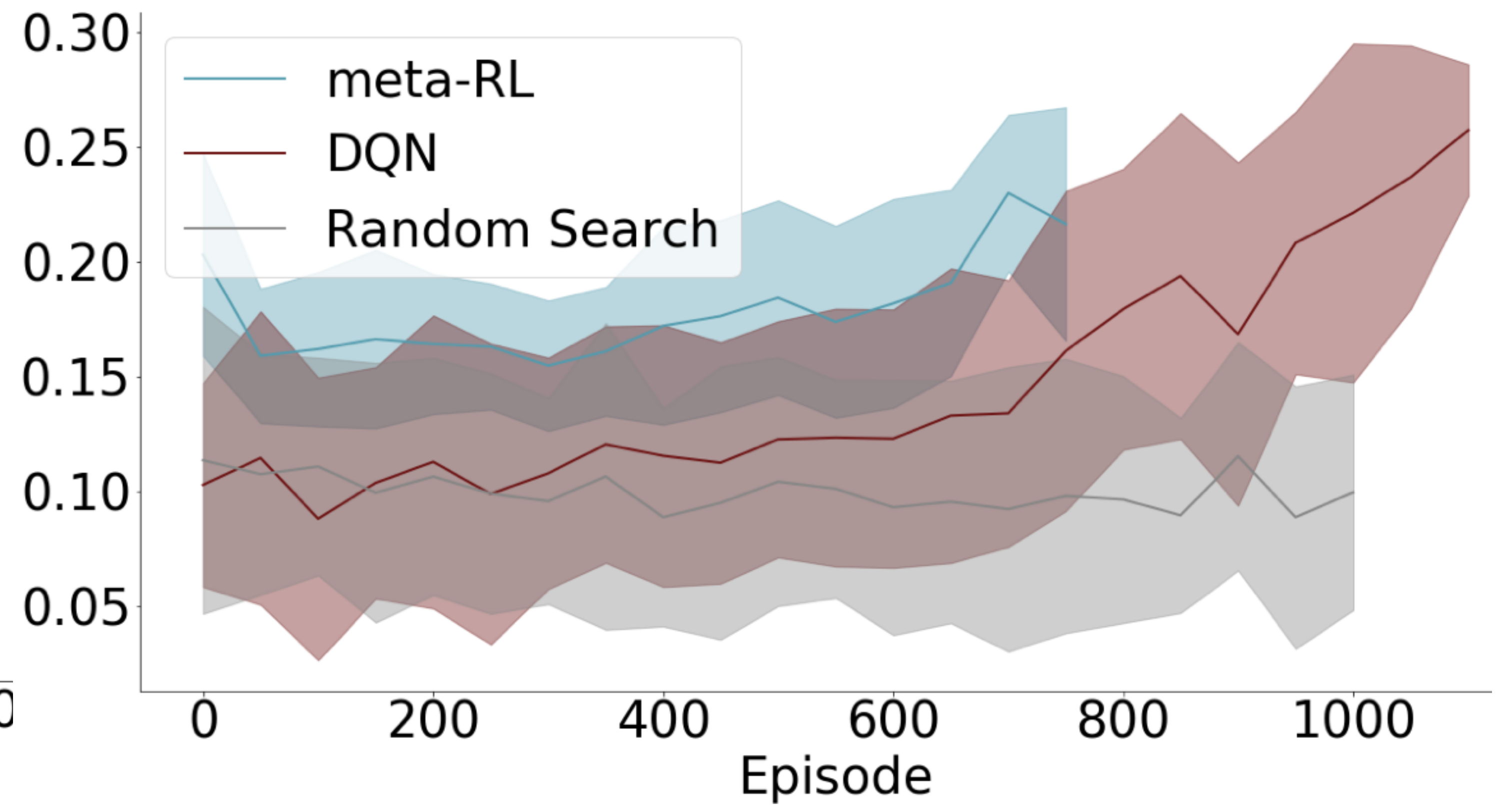




omniglot



vgg_flower



dtd

Model agnostic meta-learning (MAML)

Meta-training

- Current initialization θ, model f_θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$

derivative of test-set loss

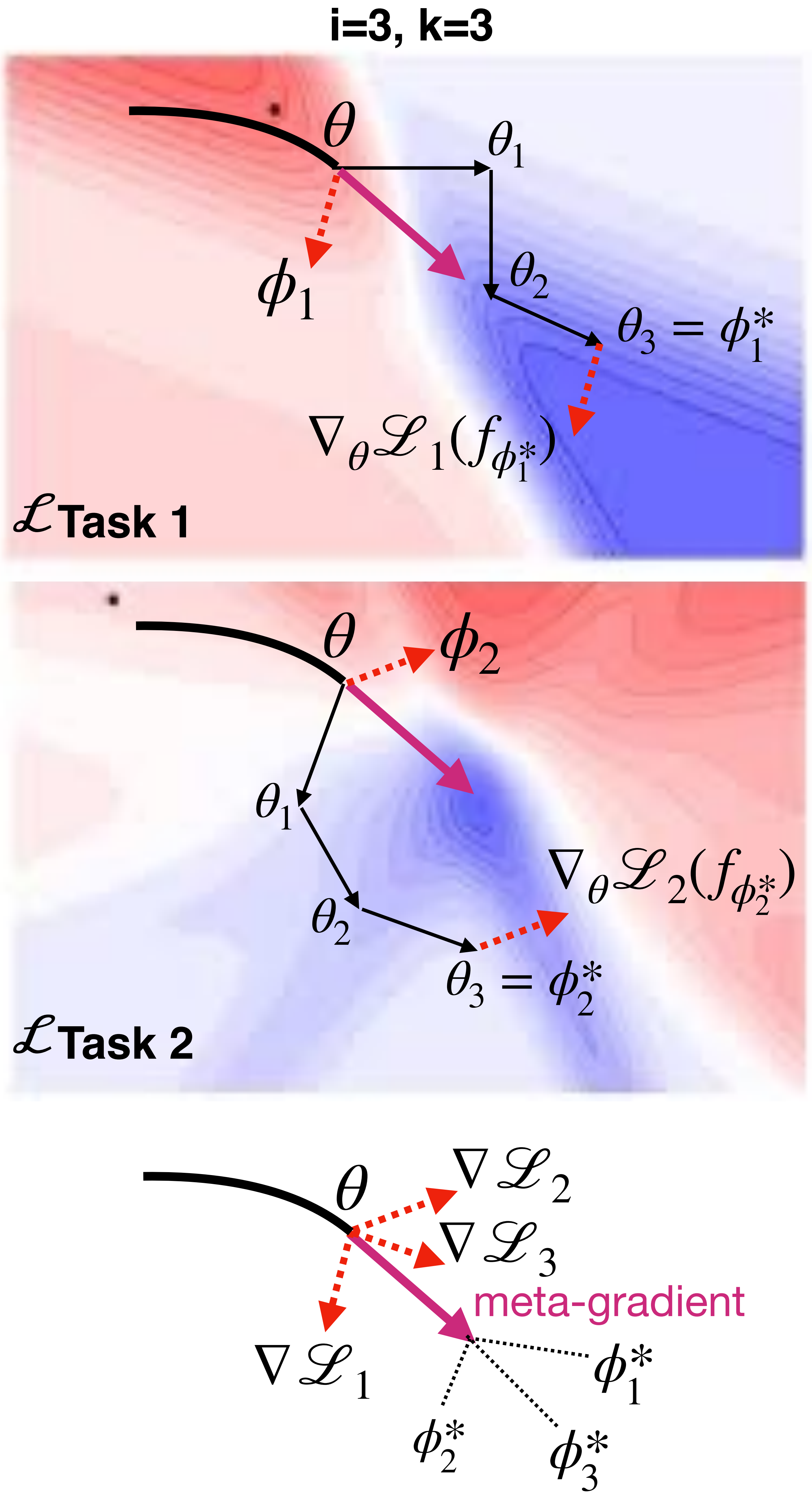
- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update θ to minimize sum of per-task losses, repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

α, β: learning rates

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

meta-gradient: <u>*second-order gradient*</u> + backpropagate
compute how changes in θ affect the gradient at new θ



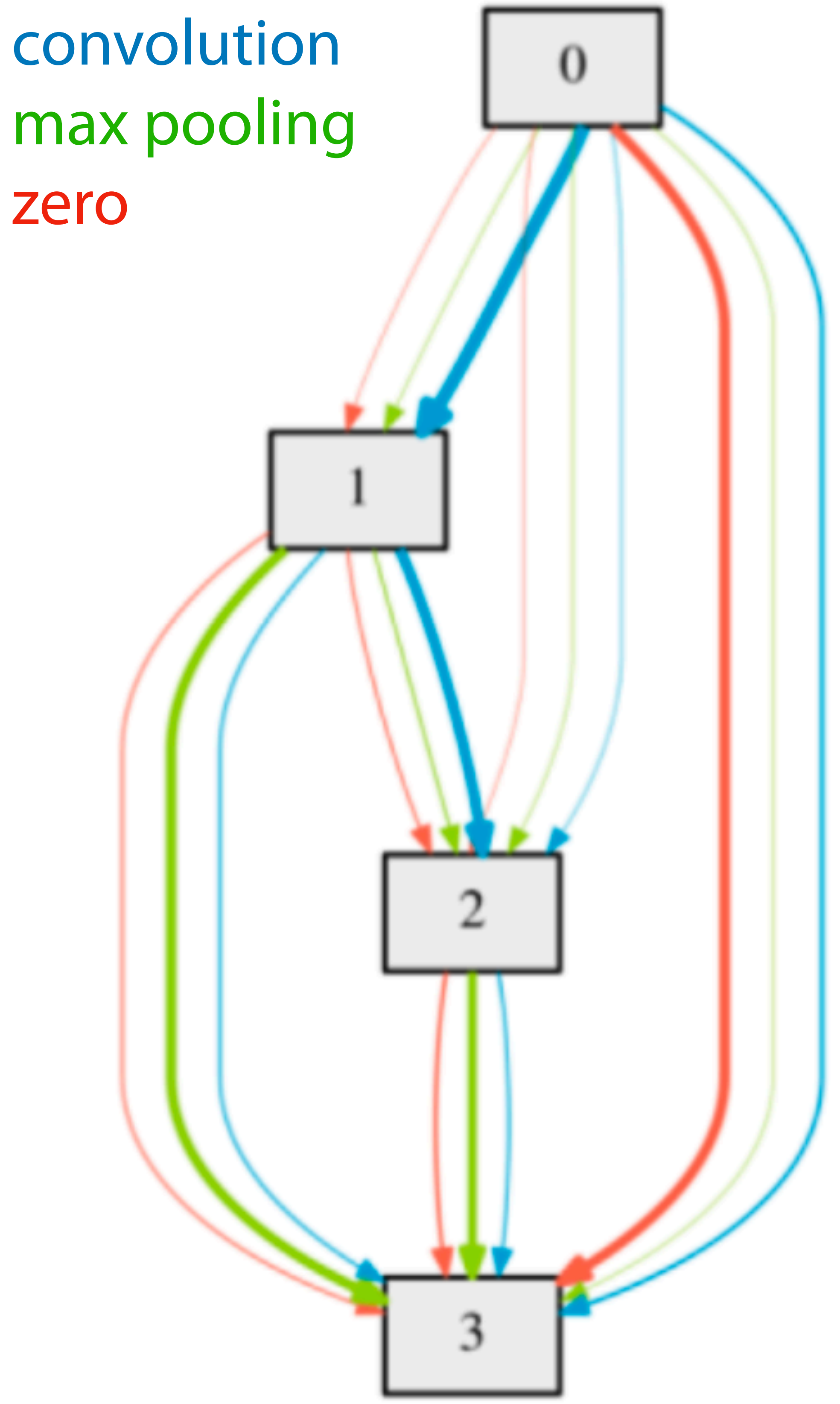
Meta-testing

- Training data of new task D_{train}
- θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$

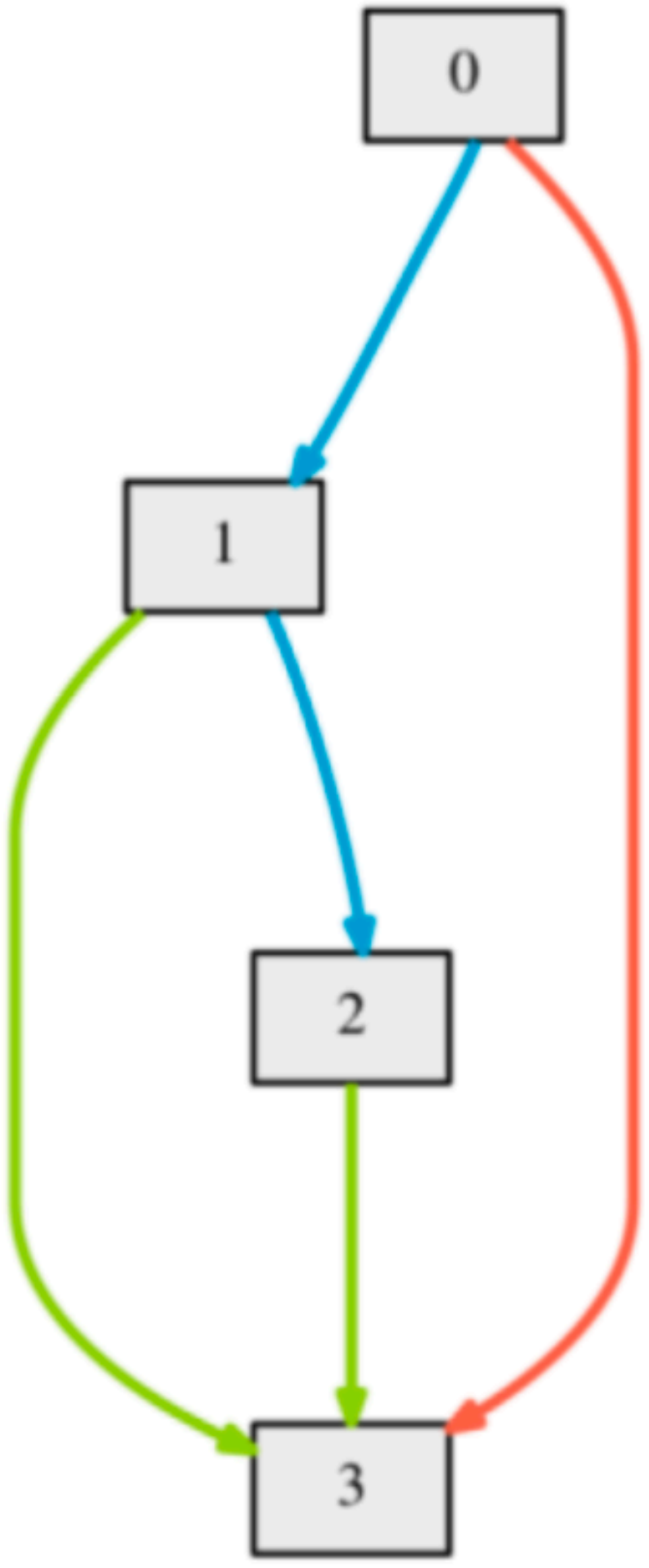
MetaNAS: meta-learning + NAS

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

Meta-learn initial operator weights α_i from previous tasks



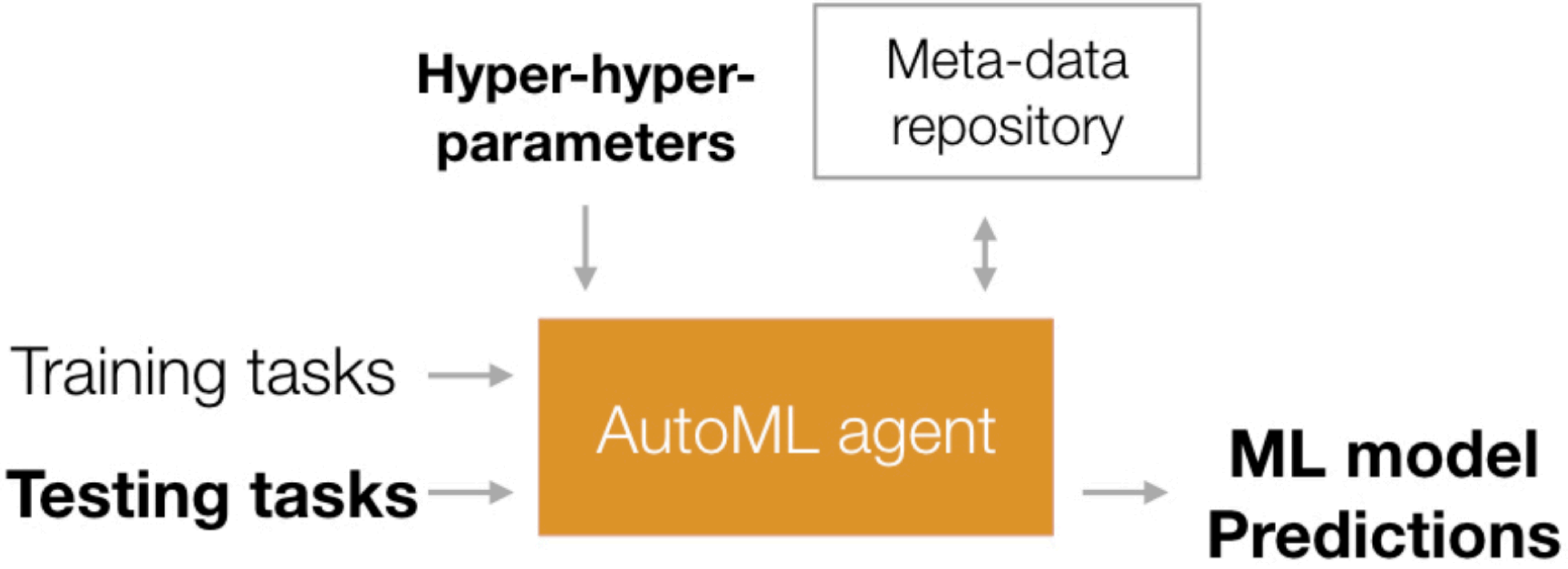
interleaved optimization of α_i and ω_j with SGD



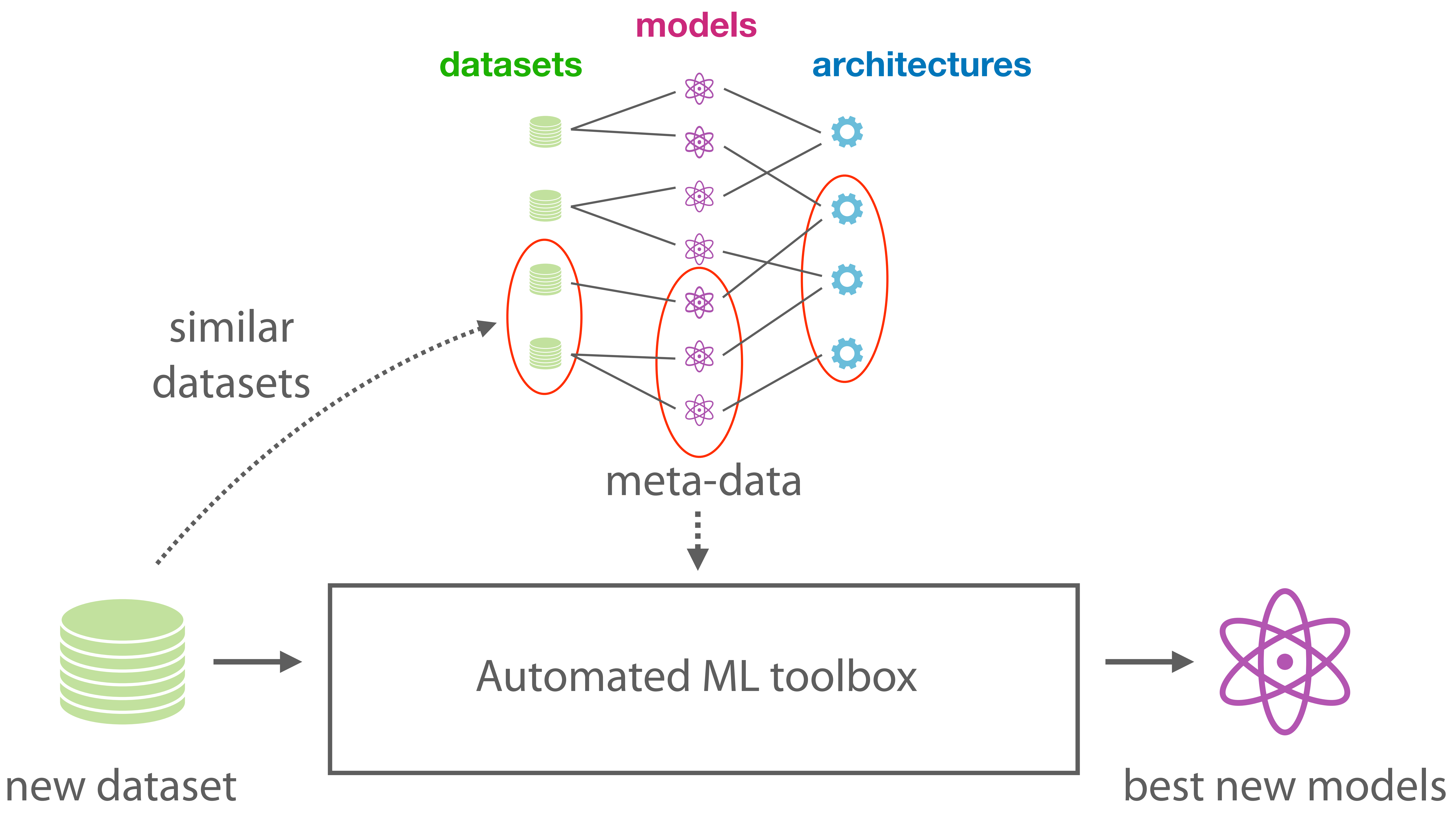
argmax α_i

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
Auto-WEKA	Param. pipeline	WEKA	Bayesian Opt. (RF)		
auto-sklearn	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	
hyperopt-sklearn	Param. pipeline	sklearn	Tree of Parzen Estim.		
skopt	Param. pipeline	sklearn	Bayesian Opt. (GP)		
TPOT	Evolving pipelines	sklearn	Population-based		
GAMA	Evolving pipelines	sklearn	Population-based	Ensemble, ASHA	
H2O AutoML	Param. pipeline	H2O	Random search	Stacking	
AutoGluon-Tabular	Param. pipeline	Sagemaker	Random search	multi-level Stacking	
OBOE	Single algorithms	sklearn	Low rank approx.	Ensembling	runtime pred
Auto-Keras	Param. NAS	keras	Bayesian Opt.	Net Morphisms	
Auto-pyTorch	Param. pipeline	pyTorch	BO-HB		
TensorFlow 2	/	keras	RS or HB		
Talos	/	keras	RS variants		

Many other tools for hyperparameter optimization alone

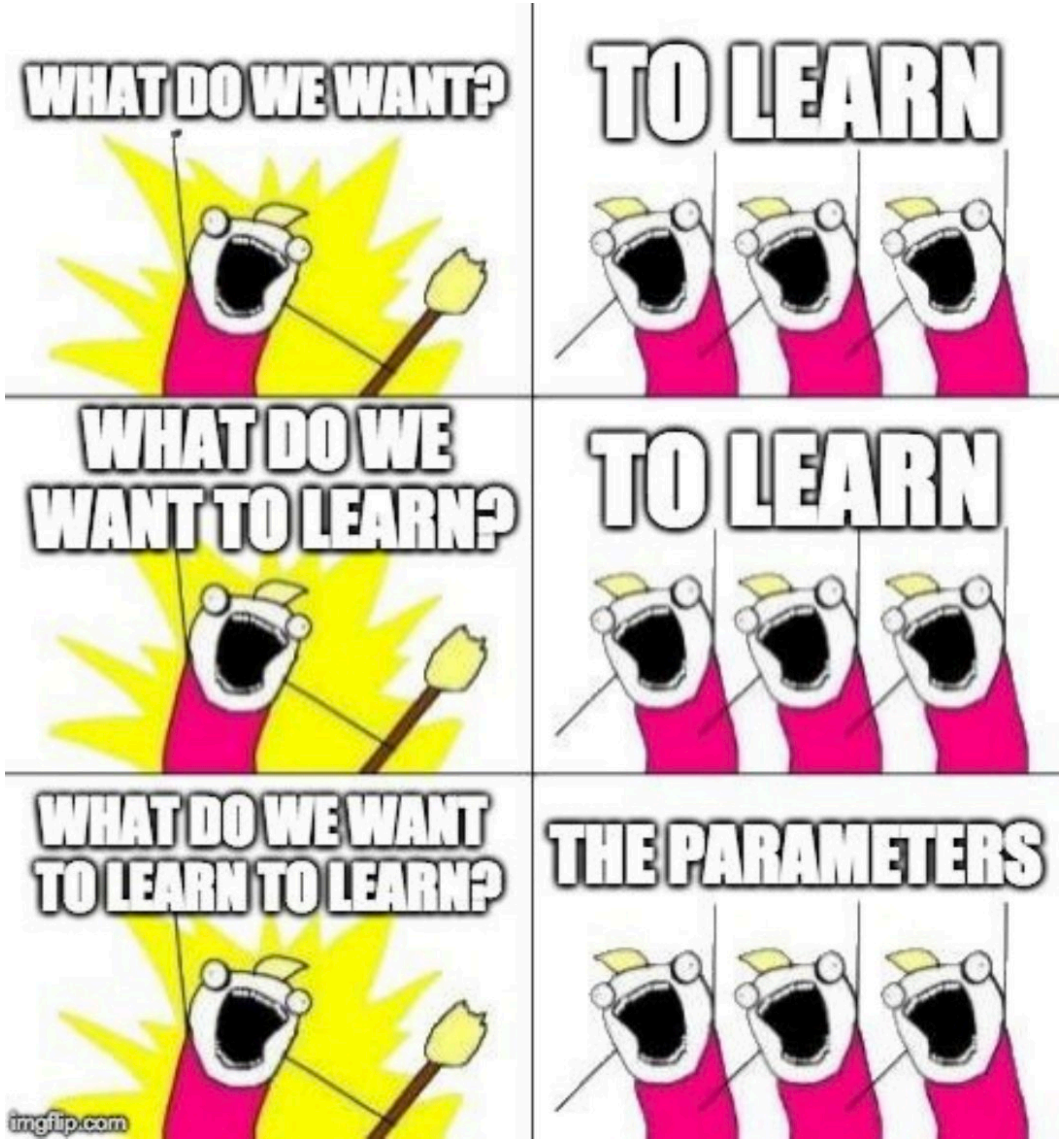


Image: Pesah et al. 2018

Further reading

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