

Architecture of ML Systems (AMLS)

06 Parameter Servers

Prof. Dr. Matthias Boehm

Technische Universität Berlin

Berlin Institute for the Foundations of Learning and Data

Big Data Engineering (DAMS Lab)

■ #1 Hybrid & Video Recording

- Hybrid lectures (in-person, zoom) with optional attendance
<https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09>
- Zoom [video recordings](#), links from website
https://mboehm7.github.io/teaching/ss25_aml/index.htm



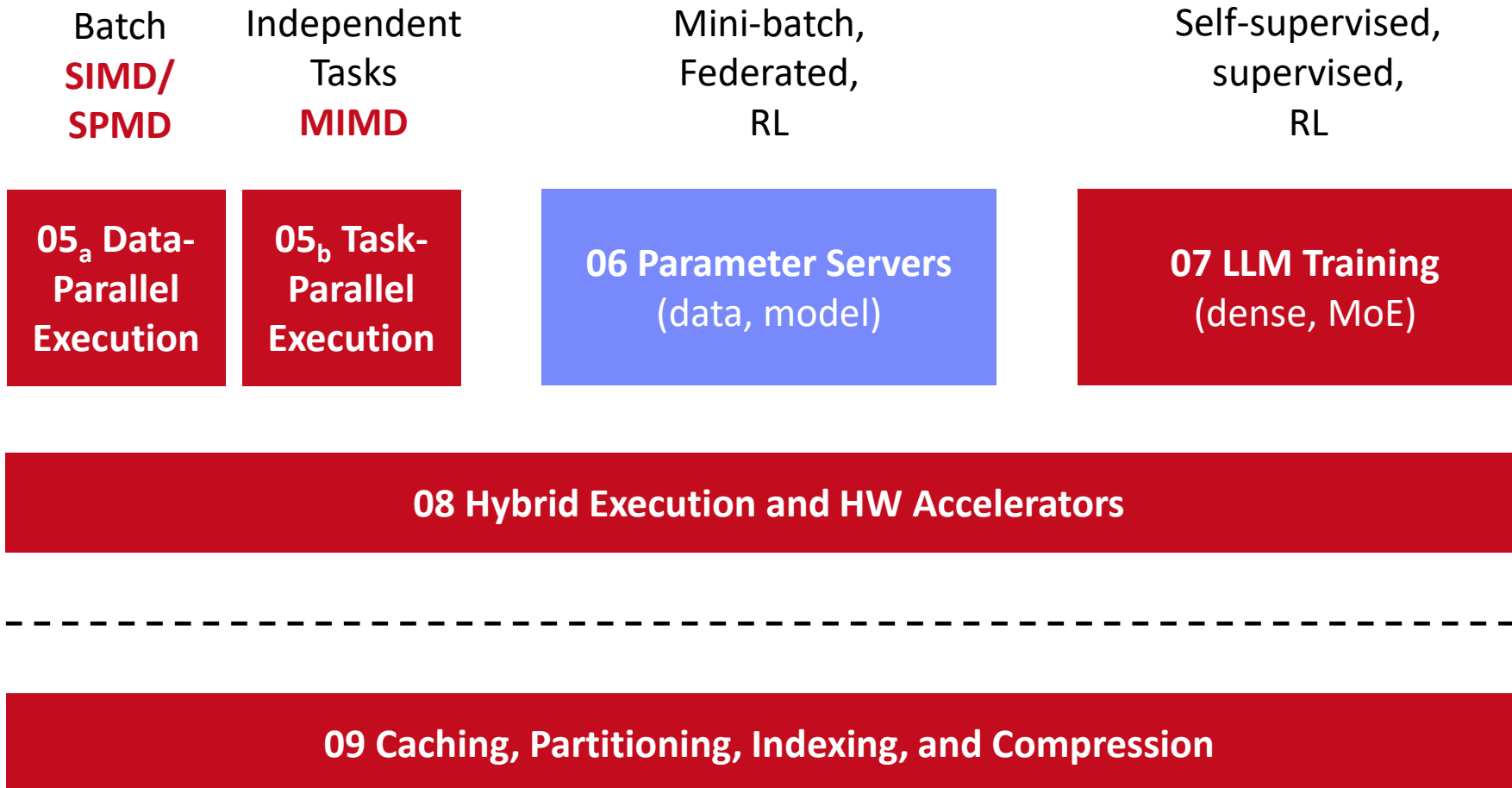
■ #2 Proposals Empowerment and Antidiscrimination

- TU-Berlin-internal call, **deadline Jul 15**, all university members, **up to 14,000 Euro**
- Ideas for impactful actions for diversity, tolerance, inclusion, underrepresented minorities

■ #3 FG DAMS Student Assistant Position

- New call for positions coming out in June
- Earliest starting date **Sep 01, 2025**, up to **two years** (BerlHG)

Categories of Execution Strategies



Agenda



- **Data-Parallel Parameter Servers**
- **Model-Parallel Parameter Servers**
- **Distributed Reinforcement Learning**
- **Federated Machine Learning**

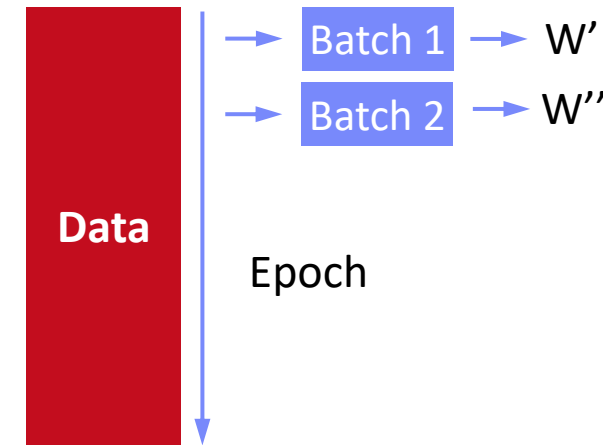
Data-Parallel Parameter Servers

Recap: Mini-batch ML Algorithms



■ Mini-batch ML Algorithms

- Iterative ML algorithms, where each iteration only uses a **batch of rows** to make the next model update (in **epochs** or w/ **sampling**)
- For large and **highly redundant training sets**
- **Applies to almost all iterative**, model-based ML algorithms (LDA, reg., class., factor., DNN)
- **Stochastic Gradient Descent** (SGD)



■ Statistical vs Hardware Efficiency (batch size)

- **Statistical efficiency**: # accessed data points to achieve certain accuracy
- **Hardware efficiency**: number of independent computations to achieve high hardware utilization (parallelization at different levels)
- **Beware higher variance / class skew for too small batches!**

➔ Training **Mini-batch** ML algorithms sequentially **is hard to scale**

Background: Mini-batch DNN Training (LeNet)



```
# Initialize W1-W4, b1-b4
# Initialize SGD w/ Nesterov momentum optimizer
iters = ceil(N / batch_size)

for( e in 1:epochs ) {
  for( i in 1:iters ) {
    X_batch = X[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]
    y_batch = Y[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]

    ## layer 1: conv1 -> relu1 -> pool1
    ## layer 2: conv2 -> relu2 -> pool2
    ## layer 3: affine3 -> relu3 -> dropout
    ## layer 4: affine4 -> softmax
    outa4 = affine::forward(outd3, W4, b4)
    probs = softmax::forward(outa4)

    ## layer 4: affine4 <- softmax
    douta4 = softmax::backward(dprobs, outa4)
    [doutd3, dW4, db4] = affine::backward(douta4, outr3, W4, b4)
    ## layer 3: affine3 <- relu3 <- dropout
    ## layer 2: conv2 <- relu2 <- pool2
    ## layer 1: conv1 <- relu1 <- pool1

    # Optimize with SGD w/ Nesterov momentum W1-W4, b1-b4
    [W4, vW4] = sgd_nesterov::update(W4, dW4, lr, mu, vW4)
    [b4, vb4] = sgd_nesterov::update(b4, db4, lr, mu, vb4)
  }
}
```

[Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner: Gradient-Based Learning Applied to Document Recognition, **Proc of the IEEE 1998**]



NN Forward
Pass

NN Backward
Pass
→ Gradients

Model
Updates

Overview Parameter Servers

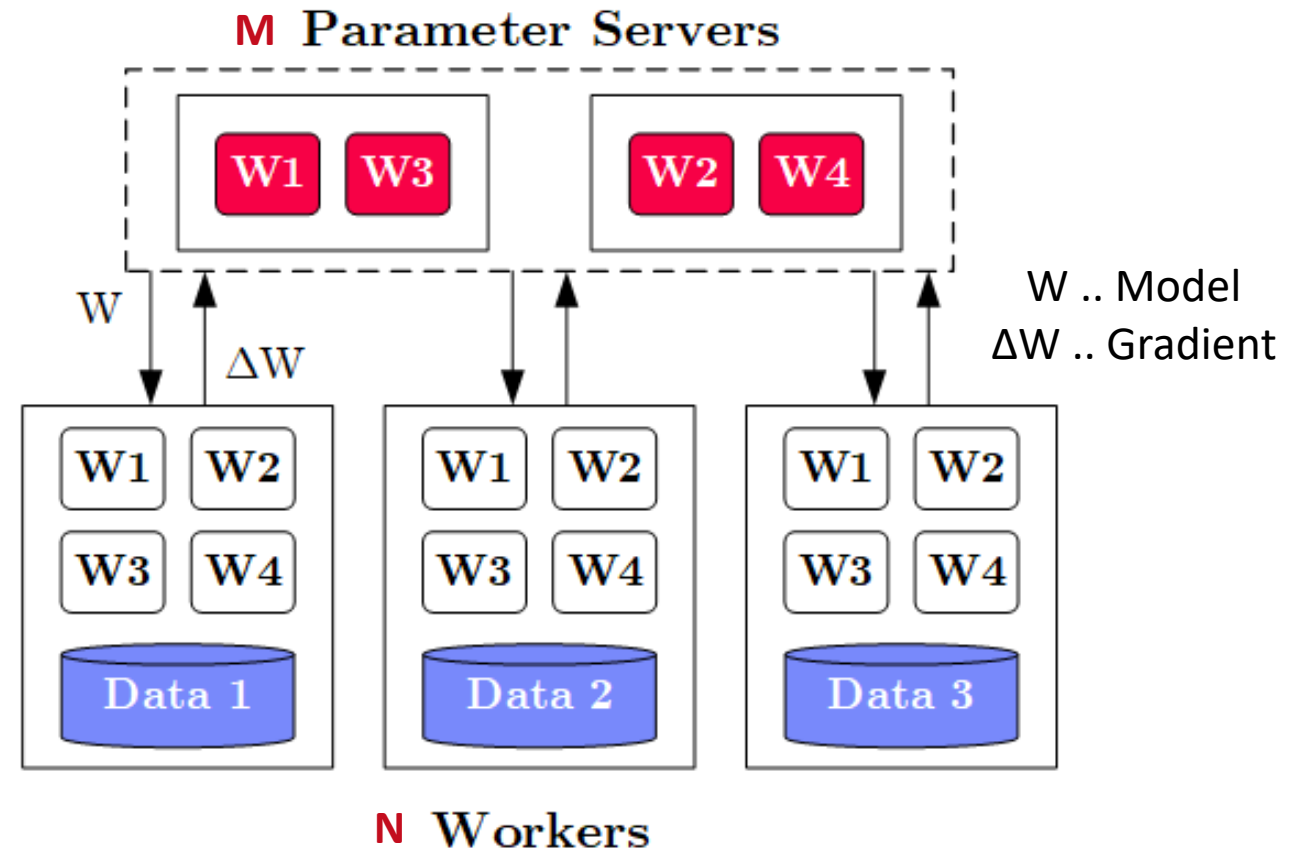


■ System Architecture

- **M** Parameter Servers
- **N** Workers
- Optional Coordinator

■ Key Techniques

- Data partitioning $D \rightarrow$ workers D_i (e.g., disjoint, reshuffling)
- Updated strategies (e.g., synchronous, asynchronous)
- Batch size strategies (small/large batches, hybrid methods)



History of Parameter Servers



■ 1st Gen: Key/Value

- **Distributed key-value store** for parameter exchange and synchronization
- Relatively high overhead

[Alexander J. Smola, Shravan M. Narayanamurthy: An Architecture for Parallel Topic Models. **PVLDB 2010**]



■ 2nd Gen: Classic Parameter Servers

- **Parameters as dense/sparse matrices**
- Different **update/consistency strategies**
- Flexible configuration and fault tolerance

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NeurIPS 2012**]



[Mu Li et al: Scaling Distributed Machine Learning with the Parameter Server. **OSDI 2014**]



■ 3rd Gen: Parameter Servers w/ improved **data communication**

- Prefetching and range-based pull/push
- Lossy or lossless compression w/ compensations

[Jiawei Jiang, Bin Cui, Ce Zhang, Lele Yu: Heterogeneity-aware Distributed Parameter Servers. **SIGMOD 2017**]



[Jiawei Jiang et al: SketchML: Accelerating Distributed Machine Learning with Data Sketches. **SIGMOD 2018**]



■ Examples

- TensorFlow, MXNet, PyTorch, CNTK, Petuum

Basic Worker Algorithm (batch)



```
for( i in 1:epochs ) {  
  for( j in 1:iterations ) {  
    params = pullModel(); # W1-W4, b1-b4 lr, mu  
    batch = getNextMiniBatch(data, j);  
    gradient = computeGradient(batch, params);  
    pushGradients(gradient);  
  }  
}
```

[Jeffrey Dean et al.: Large Scale Distributed
Deep Networks. **NeurIPS 2012**]



Extended Worker Algorithm (nfetch batches)



```
gradientAcc = matrix(0,...);
for( i in 1:epochs ) {
  for( j in 1:iterations ) {
    if( step mod nfetch = 0 )
      params = pullModel();
    batch = getNextMiniBatch(data, j);
    gradient = computeGradient(batch, params);
    gradientAcc += gradient; # parallel to updateModel
    params = updateModel(params, gradients);
    step++;
    if( step mod nfetch = 0 ) {
      pushGradients(gradientAcc); step = 0;
      gradientAcc = matrix(0, ...);
    }
  }
}
```

nfetch batches require
local gradient accrual and
local model update

[Jeffrey Dean et al.: Large Scale Distributed
Deep Networks. **NeurIPS 2012**]

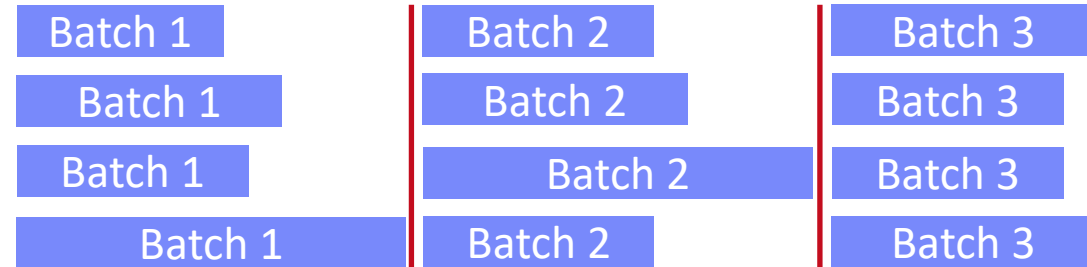


Update Strategies



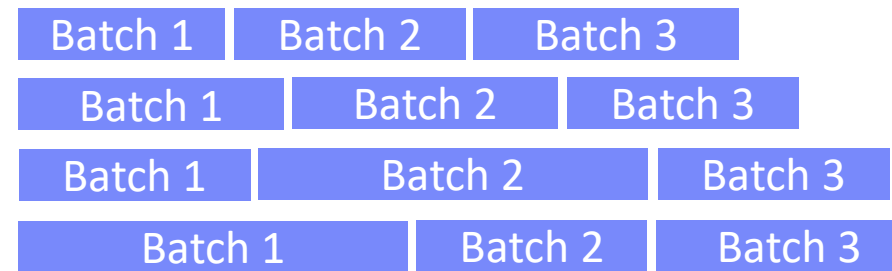
- **Bulk Synchronous Parallel (BSP)**

- Update model w/ accrued gradients
- Barrier for N workers



- **Asynchronous Parallel (ASP)**

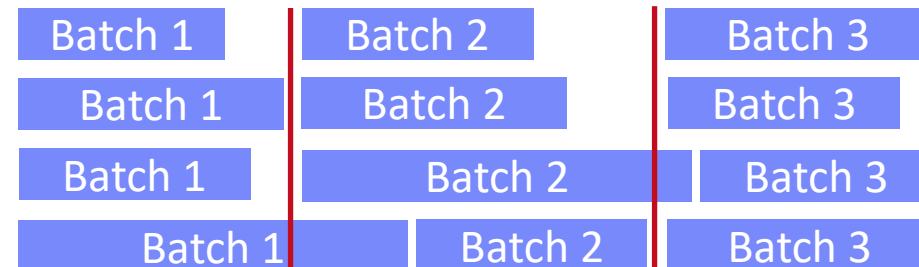
- Update model for each gradient
- No barrier



but, stale
model
updates

- **Synchronous w/ Backup Workers**

- Update model w/ accrued gradients
- Barrier for N of N+b workers



[Martín Abadi et al:
TensorFlow: A System for
Large-Scale Machine
Learning. **OSDI 2016**]



Update Strategies, cont.



■ Stale-Synchronous Parallel (SSP)

- Similar to backup workers,
weak synchronization barrier
- Maximum staleness of s clocks between fastest and slowest worker → **if violated, block fastest**

[Qirong Ho et al: More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. **NeurIPS 2013**]



■ Hogwild!

- Even the model update completely **unsynchronized**
- Shown to converge for **sparse model updates**

[Benjamin Recht, Christopher Ré, Stephen J. Wright, Feng Niu: Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. **NeurIPS 2011**]



■ Decentralized

- #1: Exchange partial gradient updates with local peers
- #2: Peer-to-peer re-assignment of work
- Other Examples: **Ako**, **FlexRR**

[Xiangru Lian et al: Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. **NeurIPS 2017**]



■ Goals Data Partitioning

- Even distribute data across workers
- Avoid skew regarding model updates → shuffling/randomization

■ #1 Disjoint Contiguous

- Contiguous row partition of features/labels

```
Xp = X[id*bsize+1:(id+1)*bsize,];
```

■ #2 Disjoint Round Robin

- Rows of features distributed round robin

```
Xp = X[seq(1,nrow(X))%%N==id,];
```

■ #3 Disjoint Random

- Random non-overlapping selection of rows

```
P = table(seq(1,nrow(X)),  
          sample(nrow(X),nrow(X),FALSE));  
Xp = P[id*bsize+1: (id+1)*bsize,] %*% X
```

■ #4 Overlap Reshuffle

- Each worker receives a reshuffled copy of the whole dataset

```
Xp = Pi %*% X
```

Example Distributed TensorFlow DP



```
# Create a cluster from the parameter server and worker hosts
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

# Create and start a server for the local task.
server = tf.train.Server(cluster, job_name=..., task_index=...)

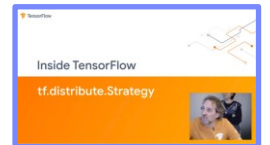
# On worker: initialize loss
train_op = tf.train.AdagradOptimizer(0.01).minimize(
    loss, global_step=tf.contrib.framework.get_or_create_global_step())

# Create training session and run steps asynchronously
hooks=[tf.train.StopAtStepHook(last_step=1000000)]
with tf.train.MonitoredTrainingSession(master=server.target,
    is_chief=(task_index == 0), checkpoint_dir=..., hooks=hooks) as sess:
    while not mon_sess.should_stop():
        sess.run(train_op)

# Program needs to be started on ps and worker
```

**But new experimental
APIs and Keras Frontend**

[Inside TensorFlow: tf.distribute.Strategy, 2019,
<https://www.youtube.com/watch?v=jKV53r9-H14>]



Example SystemDS Parameter Server



Initialize SGD w/ Adam optimizer

```
[W1, mW1, vW1] = adam::init(W1);  
[b1, mb1, vb1] = adam::init(b1); ...
```

Create the model object

```
modelList = list(W1, W2, W3, W4, b1, b2, b3, b4, vW1, vW2, vW3, vW4,  
    vb1, vb2, vb3, vb4, mW1, mW2, mW3, mW4, mb1, mb2, mb3, mb4);
```

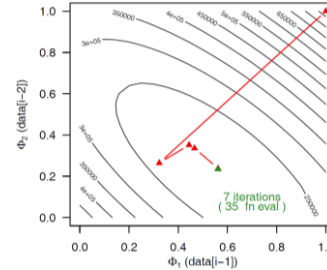
Create the hyper parameter list

```
params = list(lr=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8, t=0,  
    C=C, Hin=Hin, Win=Win, Hf=Hf, Wf=Wf, stride=1, pad=2, lambda=5e-04,  
    F1=F1, F2=F2, N3=N3)
```

Use paramserv function

```
modelList2 = paramserv(model=modelList, features=X, labels=Y,  
    upd=fGradients, aggregation=fUpdate, mode=REMOTE_SPARK, utype=ASP,  
    freq=BATCH, epochs=200, batchsize=64, k=144, scheme=DISJOINT_RANDOM,  
    hyperparams=params)
```

Selected Optimizers (updateModel)



- **Stochastic Gradient Descent (SGD)**
 - Vanilla SGD, basis for many other optimizers
 - See **05 Data/Task-Parallel**: $-\gamma \nabla f(\mathbf{D}, \boldsymbol{\theta})$
- **SGD w/ Momentum**
 - Incorporates parameter velocity w/ momentum
- **SGD w/ Nesterov Momentum**
 - Incorporates parameter velocity w/ momentum, but update from position **after** momentum
- **AdaGrad**
 - Adaptive learning rate w/ regret guarantees
- **RMSprop**
 - Adaptive learning rate, extended AdaGrad

$$\mathbf{X} = \mathbf{X} - \text{lr} * d\mathbf{X}$$

$$\mathbf{v} = \mu * \mathbf{v} - \text{lr} * d\mathbf{X}$$
$$\mathbf{X} = \mathbf{X} + \mathbf{v}$$

$$\mathbf{v}\theta = \mathbf{v}$$

$$\mathbf{v} = \mu * \mathbf{v} - \text{lr} * d\mathbf{X}$$
$$\mathbf{X} = \mathbf{X} - \mu * \mathbf{v}\theta + (1 + \mu) * \mathbf{v}$$

[John C. Duchi et al: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. **JMLR 2011**]



$$c = \text{dr} * c + (1 - \text{dr}) * d\mathbf{X}^2$$
$$\mathbf{X} = \mathbf{X} - (\text{lr} * d\mathbf{X} / (\text{sqrt}(c) + \text{eps}))$$

Selected Optimizers (updateModel), cont.



■ Adam

[Diederik P. Kingma, Jimmy Ba:
Adam: A Method for Stochastic
Optimization. **ICLR 2015**]



- Individual adaptive learning rates for different parameters + momentum

```
t = t + 1
```

```
m = beta1*m + (1-beta1)*dX # update biased 1st moment est
```

```
v = beta2*v + (1-beta2)*dX^2 # update biased 2nd raw moment est
```

```
mhat = m / (1-beta1^t) # bias-corrected 1st moment est
```

```
vhat = v / (1-beta2^t) # bias-corrected 2nd raw moment est
```

```
X = X - (lr * mhat/(sqrt(vhat)+epsilon)) # param update
```

■ Shampoo

- Preconditioned gradient method
(Newton's method, Quasi-Newton)
- Retains gradients tensor structure by
maintaining a preconditioner per dim
- Space: $O(m^2n^2) \rightarrow O(m^2 + n^2)$, time: $O(m^3 + n^3)$

```
L = L + dX %*% t(dX)
```

```
R = R + t(dX) %*% dX
```

```
X = X - lr * pow(L,1/4)  
      %*% dX %*% pow(R,1/4))
```

[Vineet Gupta, Tomer Koren,
Yoram Singer: Shampoo:
Preconditioned Stochastic Tensor
Optimization. **ICML 2018**]



Batch Size Configuration



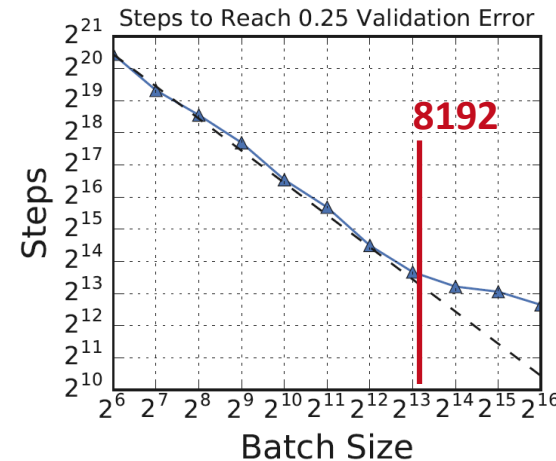
- **What is the right batch size for my data?**
 - Maximum useful batch size is dependent on data redundancy and model complexity



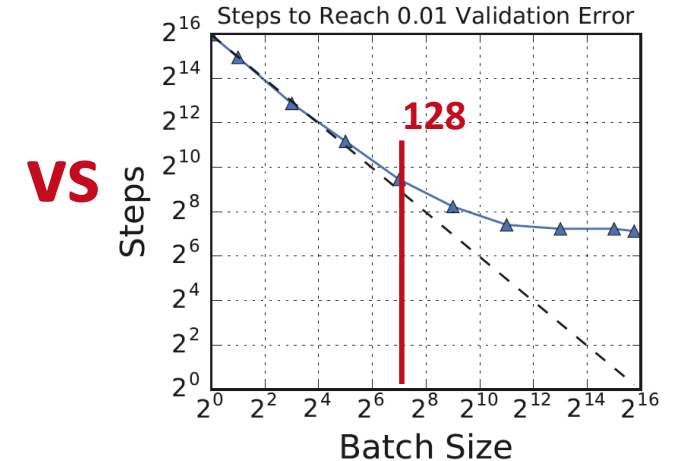
[Christopher J. Shallue et al.: Measuring the Effects of Data Parallelism on Neural Network Training. **JMLR 2019**]

- **Additional Heuristics/Hybrid Methods**
 - #1 Increase the batch size instead of decaying the learning rate
 - #2 Combine batch and mini-batch algorithms (full batch + n online updates)

ResNet-50 on ImageNet



Simple CNN on MNIST



VS

[Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le: Don't Decay the Learning Rate, Increase the Batch Size. **ICLR 2018**]



[Ashok Cutkosky, Róbert Busa-Fekete: Distributed Stochastic Optimization via Adaptive SGD. **NeurIPS 2018**]



Reducing Communication Overhead



■ Large Batch Sizes

- Larger batch sizes reduce the relative communication overhead

[Priya Goyal et al: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. **CoRR 2017** (kn=8K, 256 GPUs)]



■ Overlapping Computation/Communication

- For deep NN w/ many weight/bias matrices, compute and comm. can be overlapped
- **Collective operations:** all-Reduce / ring all-reduce / hierarchical all-reduce

tf.distribute:
MirroredStrategy
MultiWorkerMirroredStrategy

■ Compressed Communication

- Lossy (mantissa truncation, quantization), and lossless (delta, bitpacking) for W and dW
- Residual accumulation (accrue clipping errors)

[Frank Seide et al: **1-bit stochastic gradient descent** and its application to data-parallel distributed training of speech DNNs. **INTERSPEECH 2014**]



■ In-Network Aggregation (SwitchML)

- Aggregate worker updates in programmable switches
- 32b fix-point, coordinated updates

[Amedeo Sapio et al: Scaling Distributed Machine Learning with In-Network Aggregation, **NSDI 2021**]



Reducing Communication Overhead, cont.



■ Sparse Communication

- Mini-batches of sparse data → sparse dW
- Gradient sparsification/clipping (send gradients larger than a threshold)

■ Non-Uniform Parameter Access

- Exploit sparsity skew to select replication (sync/async) vs relocation

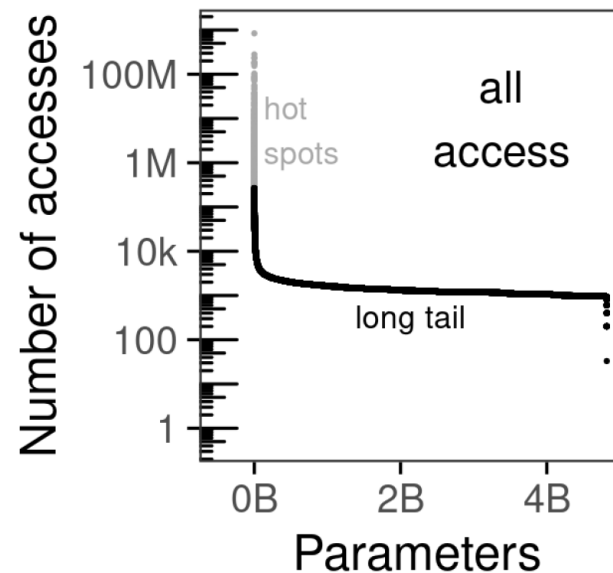


[Alexander Renz-Wieland, Rainer Gemulla, Steffen Zeuch, Volker Markl: Dynamic Parameter Allocation in Parameter Servers. **PVLDB 13(11) 2020**]

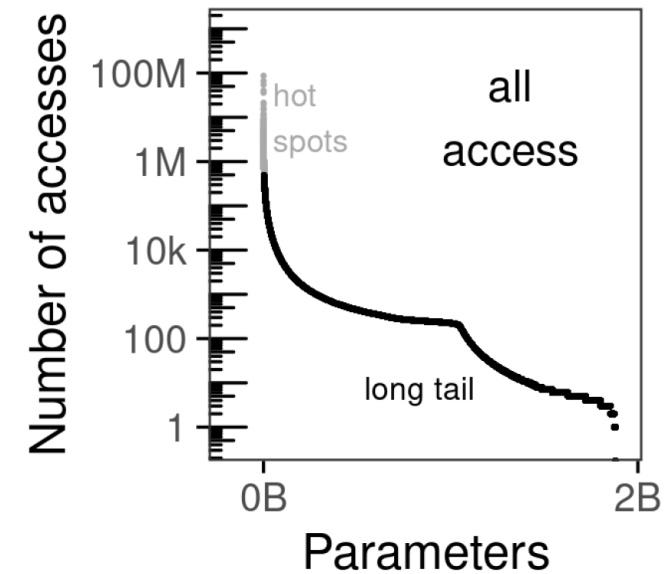


[Alexander Renz-Wieland, Rainer Gemulla, Zoi Kaoudi, Volker Markl: NuPS: A Parameter Server for Machine Learning with Non-Uniform Parameter Access. **SIGMOD 2022**]

Knowledge Graph Embeddings
(1 Epoch ComplEx on Wikidata5m)



Word Embeddings
(1 Epoch word2vec on Billion Word Benchmark)



Model-Parallel Parameter Servers

■ Limitations Data-Parallel Parameter Servers

- Need to fit entire model and activations into each worker node/device (or overhead for repeated eviction & restore)
- Very deep and wide networks (e.g., **ResNet-1001**)

[Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Identity Mappings in Deep Residual Networks. **ECCV 2016**]



■ Model-Parallel Parameter Servers

- Workers responsible for **disjoint partitions of the network/model**
- Exploit pipeline parallelism and independent subnetworks
- **Examples:** recurrent neural networks, **pre-processing tasks**

■ Hybrid Parameter Servers

- *“To be successful, however, we believe that model parallelism must be combined with clever distributed optimization techniques that leverage data parallelism.”*
- *“[...] it is possible to use **tens of thousands of CPU cores** for training a single model”*

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NeurIPS 2012**]



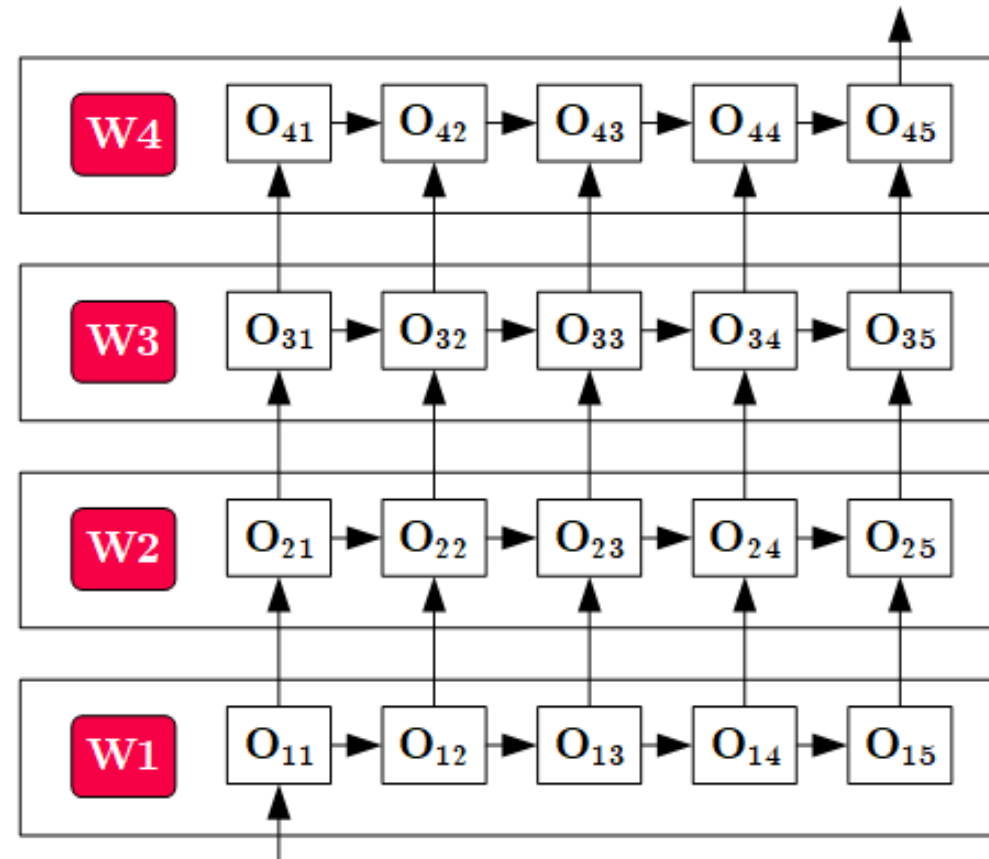
Overview Model-Parallel Execution



■ System Architecture

- Nodes act as workers and parameter servers
- Data Transfer for boundary-crossing data dependencies

■ Pipeline Parallelism



Workers
w/ disjoint
network/model
partitions

Example Distributed TensorFlow MP



Place variables and ops on devices

```
with tf.device("/gpu:0"):
    a = tf.Variable(tf.random.uniform(...))
    a = tf.square(a)
with tf.device("/gpu:1"):
    b = tf.Variable(tf.random.uniform(...))
    b = tf.square(b)
with tf.device("/cpu:0"):
    loss = a+b
```

**Explicit Placement of
Operations**

(shown via toy example)

Declare optimizer and parameters

```
opt = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = opt.minimize(loss)
```

Force distributed graph evaluation

```
ret = sess.run([loss, train_op])
```

Pathways: Asynchronous, Distributed Data Flow

[Paul Barham et al: Pathways:
Asynchronous Distributed
Dataflow for ML, **MLSys 2022**]

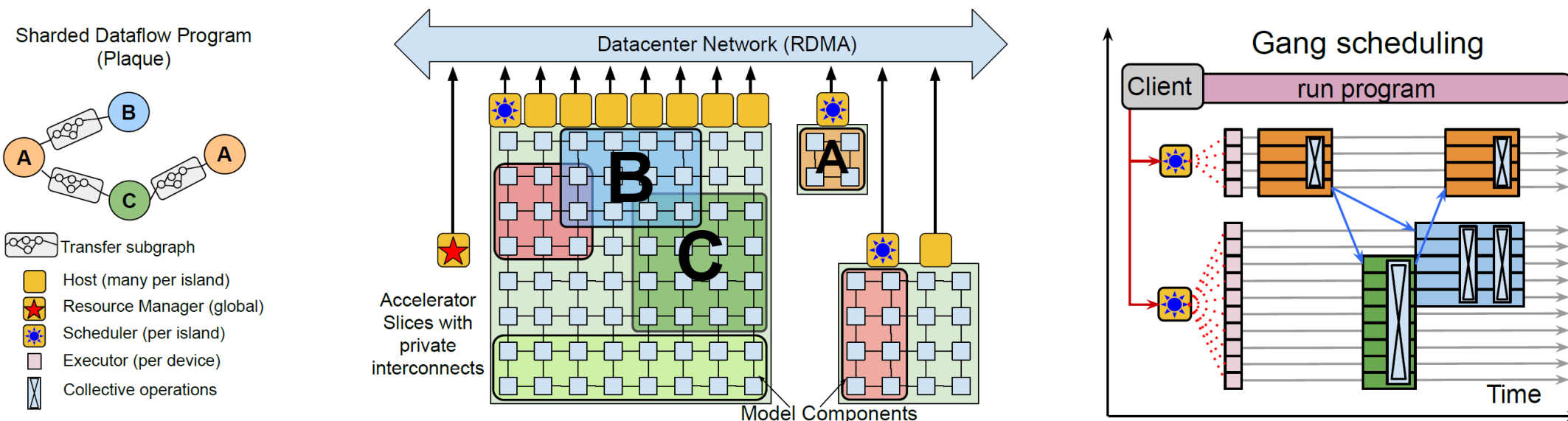


■ System Overview

- **TF** and **JAX** programs (e.g., JAX `pmap()`)
- Virtual device requests → **device islands**
- **MLIR** dialect, lowering to physical devices
- **PLAQUE** shared data-flow system w/ **sharded buffer**, **sparse comm.**, **gang scheduling**

```
def get_devices(n):  
    """Allocates `n` virtual TPU devices on an island."""  
    device_set = pw.make_virtual_device_set()  
    return device_set.add_slice(tpu_devices=n).tpus  
  
a = jax.pmap(lambda x: x * 2., devices=get_devices(2))  
b = jax.pmap(lambda x: x + 1., devices=get_devices(2))  
c = jax.pmap(lambda x: x / 2., devices=get_devices(2))  
  
@pw.program # Program tracing (optional)  
def f(v):  
    x = a(v)  
    y = b(x)  
    z = a(c(x))  
    return (y, z)  
  
print(f(numpy.array([1., 2.])))  
# output: (array([3., 5.]), array([2., 4.]))
```

■ Resource Management and Scheduling



Distributed Reinforcement Learning

Hybrid Data- and Task- Parallel Execution
Data-Parallel Parameter Servers
Nested Parallelism

■ RL Characteristics

- **Closed-loop:** goal-directed learning from interaction
- **Time-delayed reward:** map situations $\rightarrow$ actions, max reward
- **No instructions:**
exploitation (known actions)
vs exploration (find actions)

[Richard S. Sutton, Andrew G. Barto: Reinforcement Learning: An Introduction, MIT Press, 2015]



■ RL Elements

- **Policy:** stimulus-response rules (perceived environment state $\rightarrow$ actions)
- **Reward Signal:** scalar reward at each time step (direct vs indirect)
- **Value Function:** long-term desirability of states (expected reward)
- **Model of the environment:** expected behavior of environment $\rightarrow$ planning

Distributed RL in RLlib

[Eric Liang, Richard Liaw et al: **RLlib**:
Abstractions for Distributed
Reinforcement Learning. **ICML 2018**]

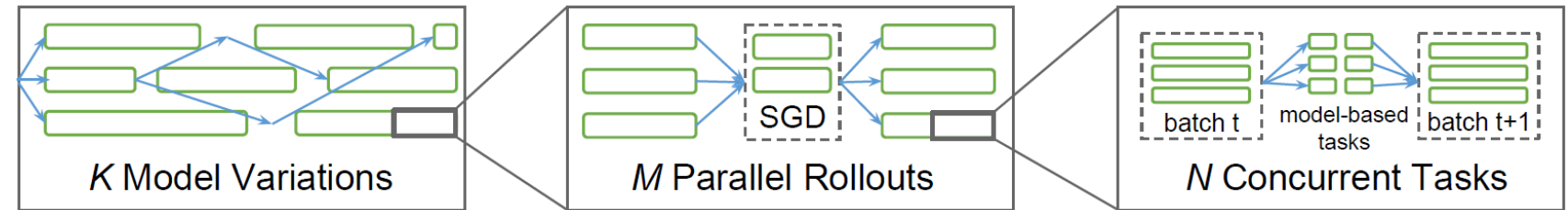


[Philipp Moritz, Robert Nishihara et al.:
Ray: A Distributed Framework for Emerging
AI Applications. **OSDI 2018**]



Framework Overview

- RLlib on tasks/actors in Ray
- Interleaved policy training, simulations, etc



Parallelization Strategies

- Hierarchical Parallel Task Model (locally, centralized control)
- Policy optimizer** step methods (All-reduce, local multi-GPU, async, **parameter server**)
- Policy graph** (algorithm-specific) on multiple remote evaluator replicas



```
grads = [ev.grad(ev.sample())  
          for ev in evaluators]  
for _ in range(NUM_ASYNC_GRADS):  
    grad, ev, grads = wait(grads)  
    for ps, g in split(grad, ps_shards):  
        ps.push(g)  
    ev.set_weights(concat(  
        [ps.pull() for ps in ps_shards])  
    )  
    grads.append(ev.grad(ev.sample()))
```

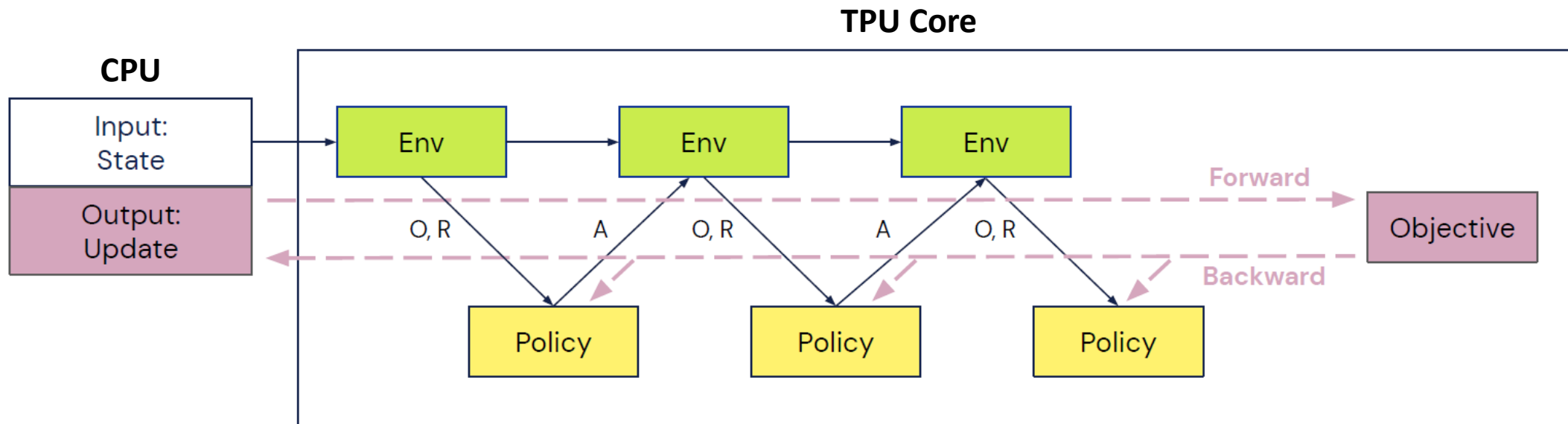
Example
Parameter Server
(task stream, wait
for #updates)

Podracer RL Architectures

[Matteo Hessel, Manuel Kroiss, et al:
Podracer architectures for scalable
Reinforcement Learning, **CoRR 2021**]



- Use of TPU Pods via JAX/TF XLA
- #1 Anakin
 - Agent-environment interaction can be compiled into a single XLA program
 - **Scalability:** replicate basic setup to larger TPU slices



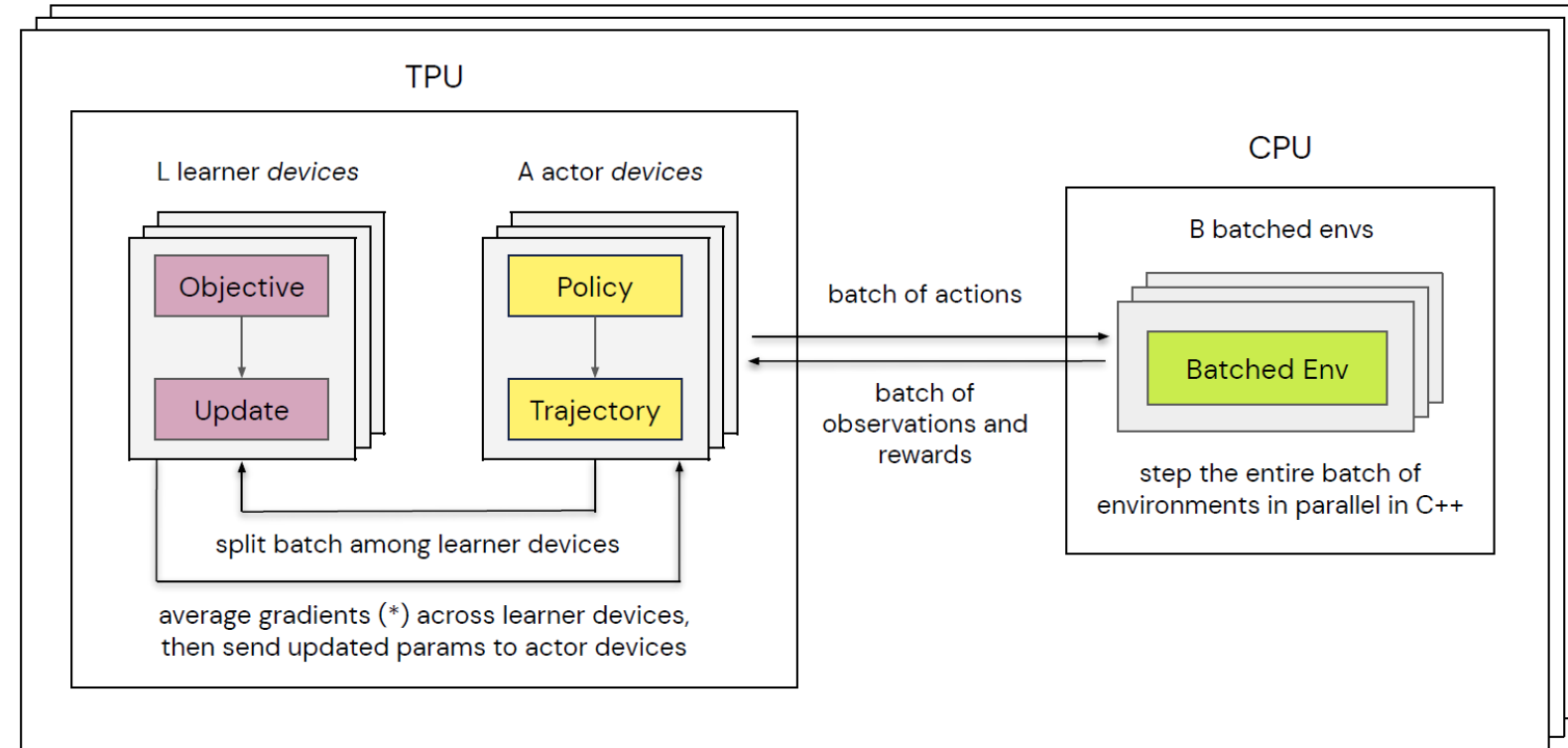
Podracer RL Architectures, cont.

[Matteo Hessel, Manuel Kroiss, et al:
Podracer architectures for scalable
Reinforcement Learning, **CoRR 2021**]



■ #2 Sebulba

- Decomposed actors and learners
- Support for arbitrary environments



This entire computation is replicated across S slices of a TPU Pod,
in which case gradients in (*) are averaged across all learner devices of all slices

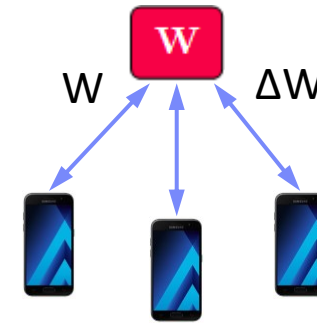
Federated Machine Learning

Problem Setting and Overview



■ Motivation Federated ML

- Learn model **w/o central data consolidation**
- **Privacy + data/power caps** vs **personalization and sharing**
- Applications Characteristics
 - #1 On-device data more relevant than server-side data
 - #2 On-device data is privacy-sensitive or large
 - #3 Labels can be inferred naturally from user interaction
- **Example:** Language modeling for mobile keyboards and voice recognition



■ Challenges

- Massively distributed (data stored across many devices)
- Limited and unreliable communication
- Unbalanced data (skew in data size, non-IID)
- Unreliable compute nodes / data availability



[Jakub Konečný: Federated Learning - Privacy-Preserving Collaborative Machine Learning without Centralized Training Data, **UW Seminar 2018**]

A Federated ML Training Algorithm



```
while( !converged ) {  
  1. Select random subset (e.g. 1000)  
    of the (online) clients  
  2. In parallel, send current parameters  $\theta_t$   
    to those clients  
    At each client  
    2a. Receive parameters  $\theta_t$  from server [pull]  
    2b. Run some number of minibatch SGD steps,  
        producing  $\theta'$   
    2c. Return  $\theta' - \theta_t$  (model averaging) [push]  
  3.  $\theta_{t+1} = \theta_t +$  data-weighted average of client updates  
}
```

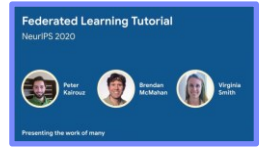
[Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Agüera y Arcas: Communication-Efficient Learning of Deep Networks from Decentralized Data. **AISTATS 2017**]



Algorithmic Parameter-Server Extensions



- #1 Client Sampling (**FedAvg** w/ model averaging)
- #2 Decentralized, Fault-tolerant Aggregation
- #3 Peer-to-peer Gradient and Model Exchange
- #4 Meta-learning for Private Models
- #5 Handling Statistical Heterogeneity (non-IID data)
 - Reducing variance
 - Selecting relevant subsets of data
 - Tolerating partial client work
 - Partitioning clients into congruent groups
 - Adaptive Optimization (**FedOpt**, **FedAvgM**)



[Peter Kairouz, Brendan McMahan, Virginia Smith:
Federated Learning Tutorial. **NeurIPS 2020**,
[https://slideslive.com/38935813/
federated-learningtutorial](https://slideslive.com/38935813/federated-learningtutorial)]

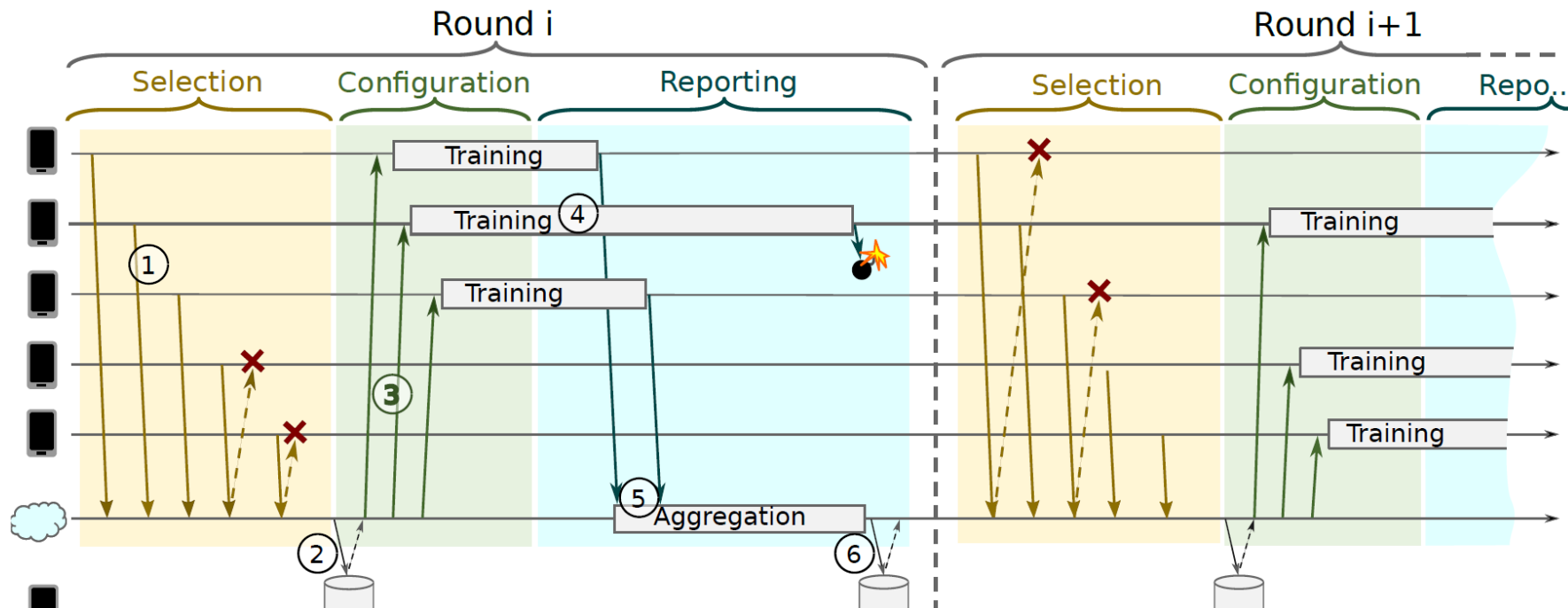
[Sashank J. Reddi et al:
Adaptive Federated
Optimization. **CoRR 2020**]



Federated Learning Protocol

■ Recommended Reading

- [Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, Jason Roselander: [Towards Federated Learning at Scale: System Design](#). MLSys 2019]



Federated Learning at the Device



■ Data Collection

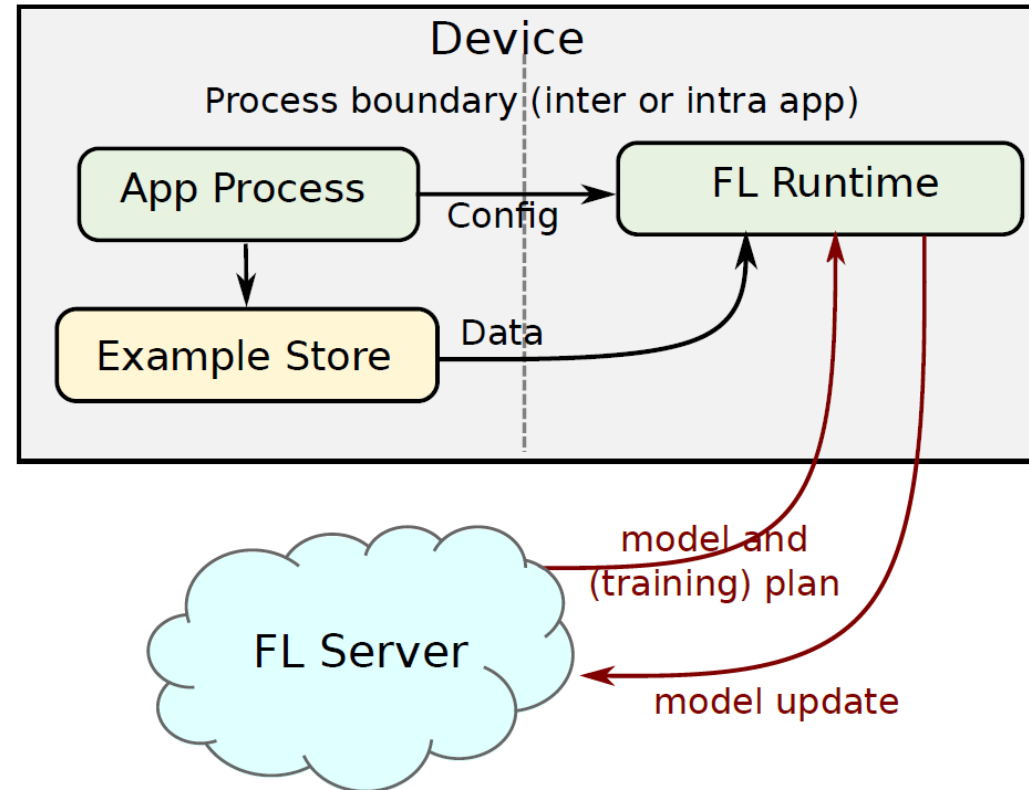
- Maintain repository of locally collected data
- Apps make data available via dedicated API

■ Configuration

- **Avoid negative impact** on data usage or battery life
- Training and evaluation tasks

■ Multi-Tenancy

- Coordination between **multiple learning tasks** (apps and services)



Federated Learning at the Server



■ Actor Programming Model

- Comm. via message passing
- Actors sequentially process stream of events/messages

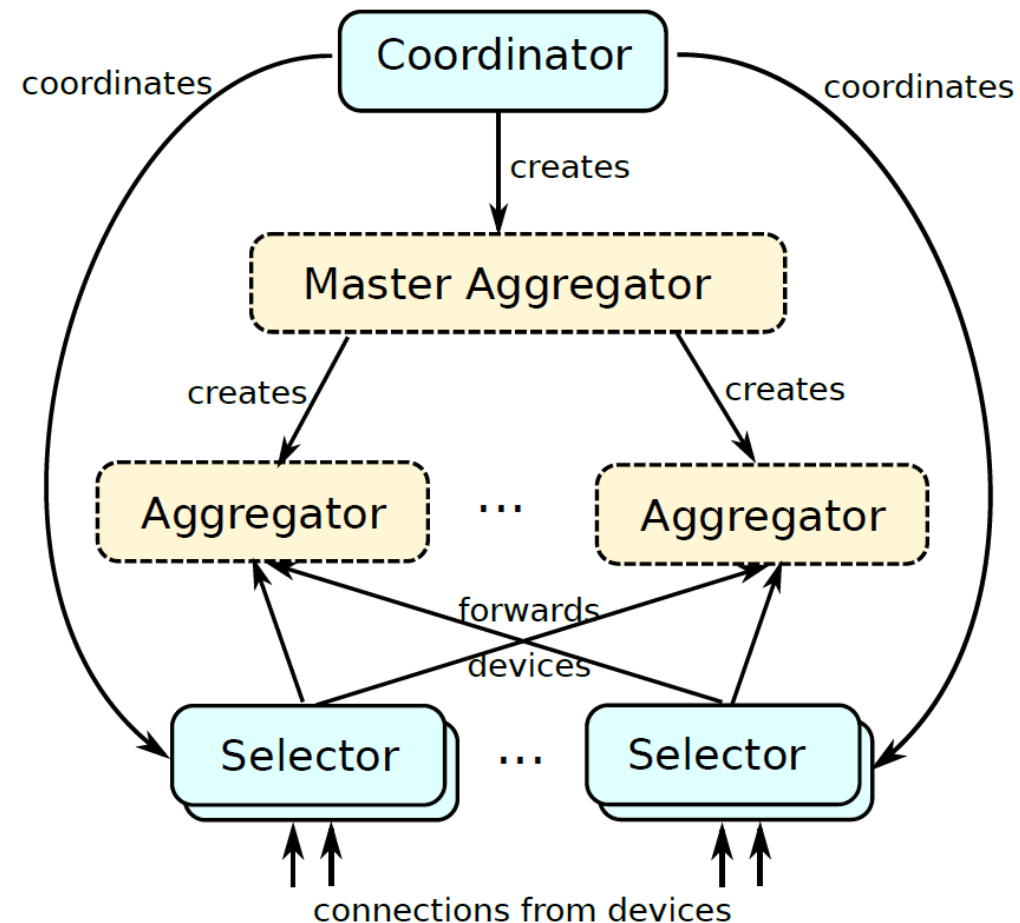
➔ **Scaling w/ # actors**

■ Coordinators

- Driver of overall learning algorithm
- **Orchestration of aggregators** and selectors (conn handlers)

■ Robustness

- Pipelined selection and aggregation rounds
- Fault Tolerance at aggregator/master aggregator levels



- Persistent (long-lived) actor
- Ephemeral (short-lived) actor

Excursus: Data Ownership



- **Limited Access to Data Sources**

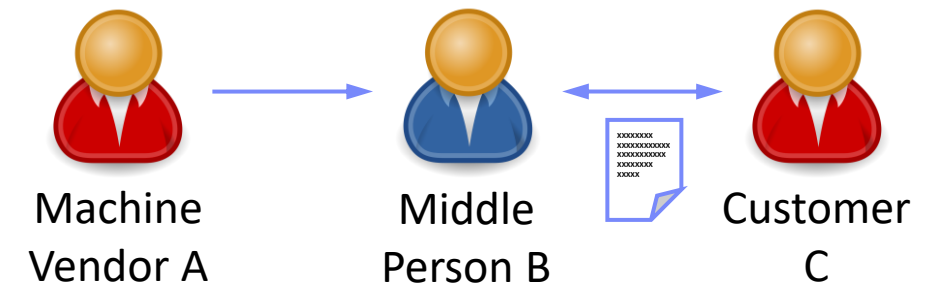
- #1 Infeasible data consolidation (privacy, economically/technically)
- #2 Data ownership (restricted data enrichment and consolidation)

- **Example Data Ownership**

- **Thought experiment:**

- middle person B uses machine from machine vendor A to test customer C's equipment.

- Who owns the data?
 - **Unclear. Usually negotiated in bilateral contracts!**



- **Note: Recent Work on Incentives**

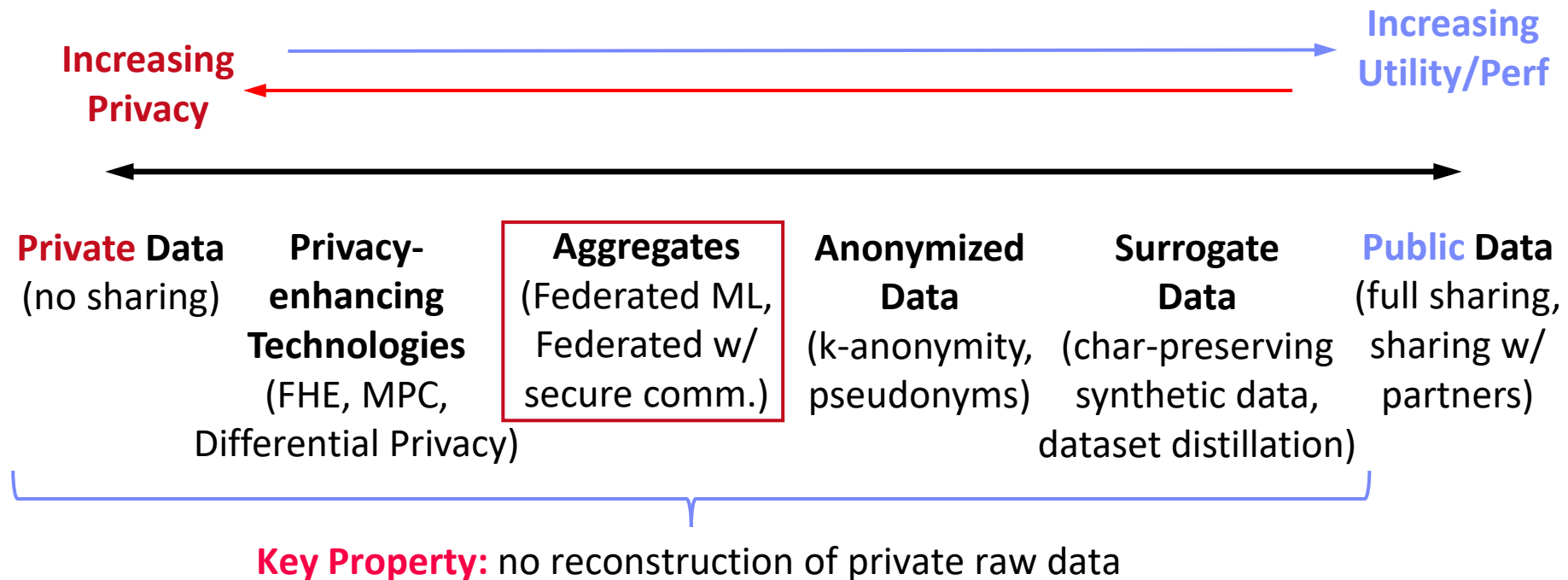
- Payments according to value of data (e.g., improved model accuracy)

Excursus: Spectrum of Data Sharing



■ Fine-grained Spectrum

- Spectrum of technologies with **performance/privacy/utility** tradeoffs
- Different applications with different requirements → **Potential for new markets**

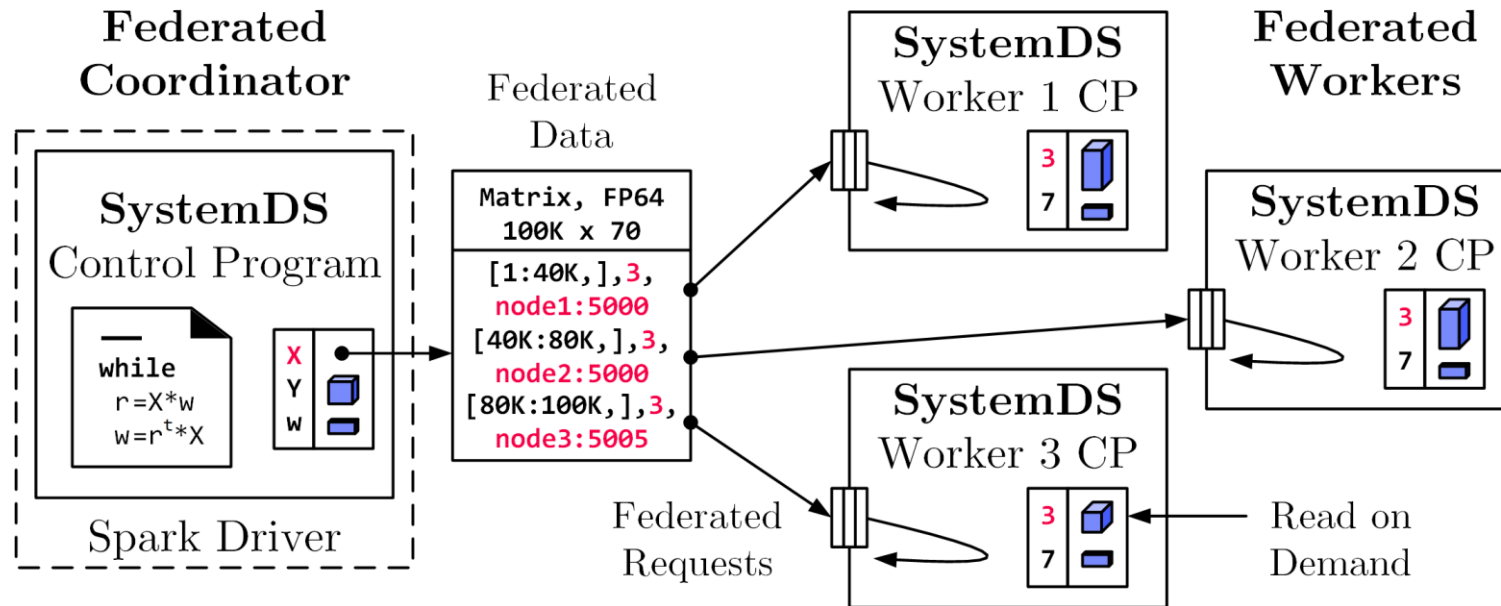


Federated Learning in SystemDS

■ Federated Backend

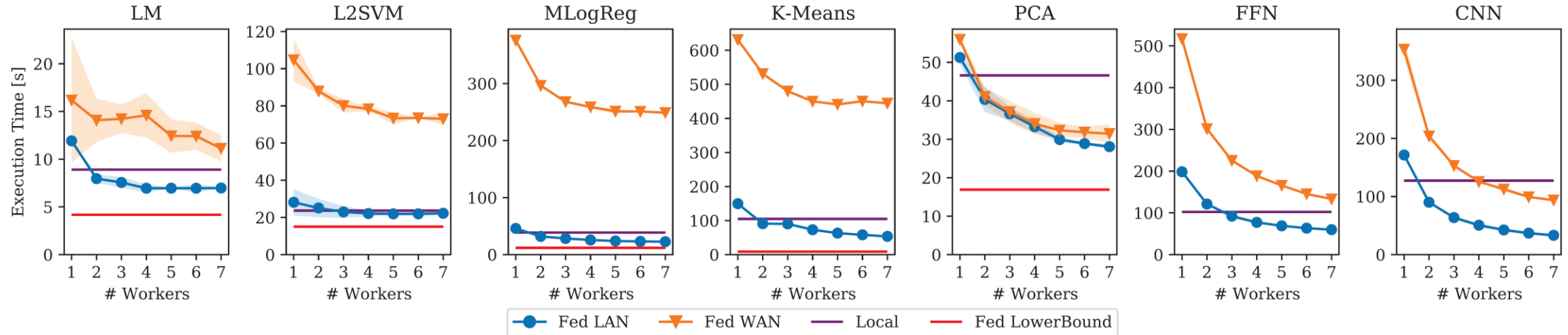
- **Federated data** (matrices/frames) as meta data objects
- **Federated linear algebra**, (and **federated parameter server**)

```
X = federated(addresses=list(node1, node2, node3),  
              ranges=list(list(0,0), list(40K,70), ..., list(80K,0), list(100K,70)));
```



Federated Requests:
READ, PUT, GET, EXEC_INST,
EXEC_UDF, CLEAR

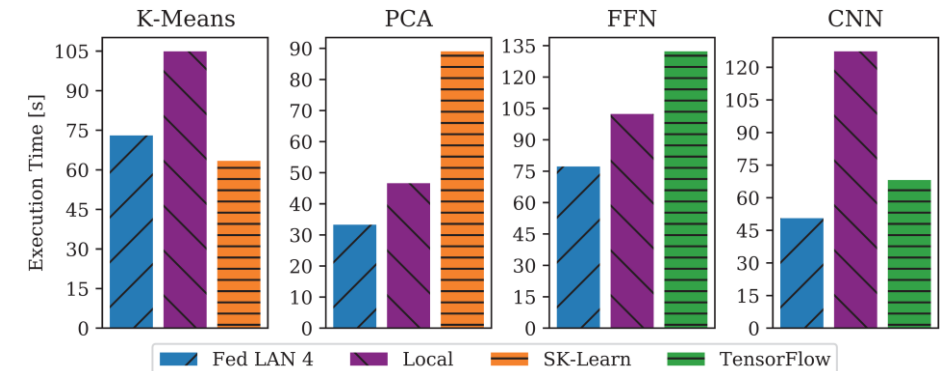
- ➔ **Design Simplicity:**
- (1) reuse instructions
 - (2) federation hierarchies



Workloads and Baselines

- **LM**: linear regression, lmCG
- **L2SVM**: l2-regularized SVM
- **MLogReg**: multinomial logreg
- **K-Means**: Lloyd's alg. w/ K-Means++ init
- **PCA**: principal component analysis
- **FFN**: fully-connected feed-forward NN
- **CNN**: convolutional NN

Comparisons w/
Scikit-learn and
TensorFlow

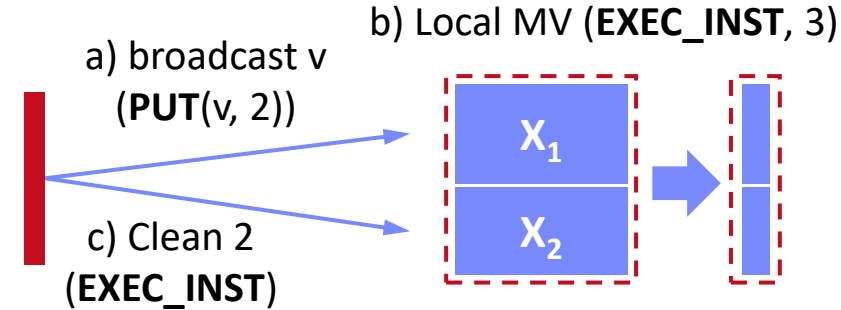


Federated Learning in SystemDS – Example Operations



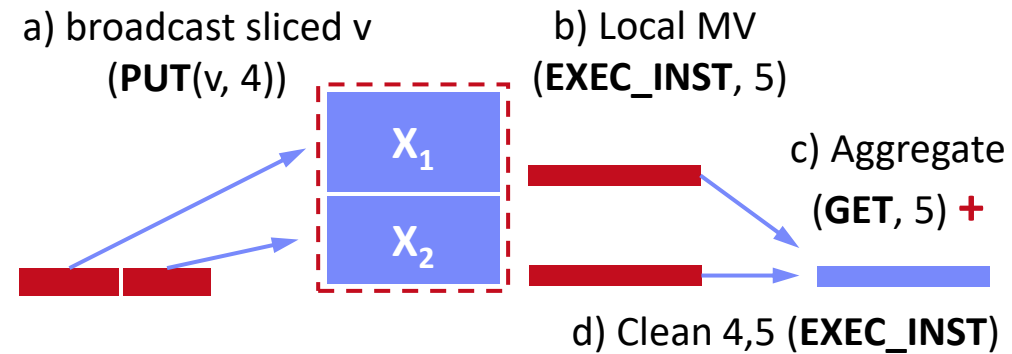
Matrix-Vector Multiplication

- $o = X \%* \% v$, local v
- Row-partitioned, federated X
- Row-partitioned, federated o



Vector-Matrix Multiplication

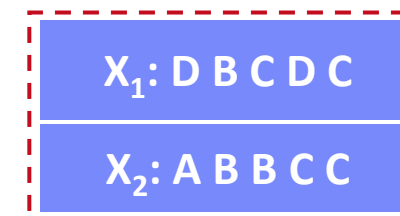
- $o = v \%* \% X$, local v
- Row-partitioned, federated X , local o
- New broadcast handling



Data Preparation

- $[X, M] = \text{transformencode}(F, \text{spec})$
- Recoding, feature hashing, binning, one-hot encoding

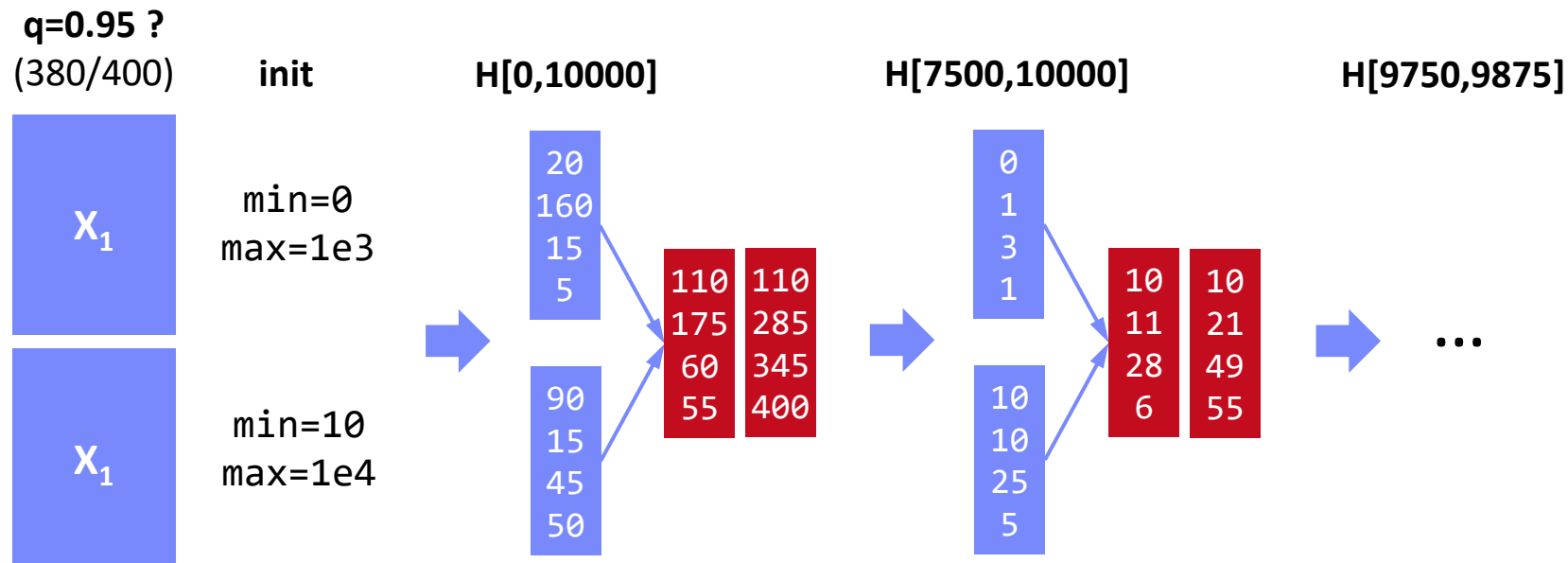
- 1) Build local record maps (EXEC_UDF)
- 2) Aggregate, broadcast, recode



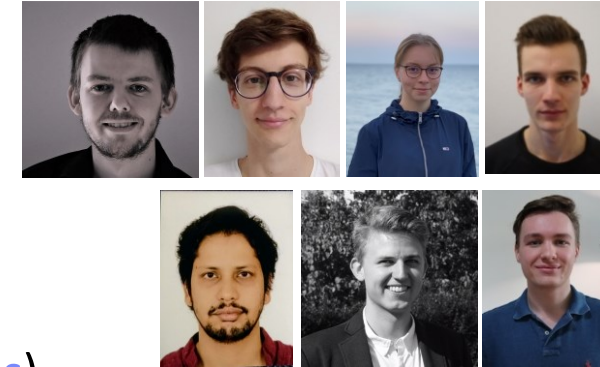


■ Federated Quantiles

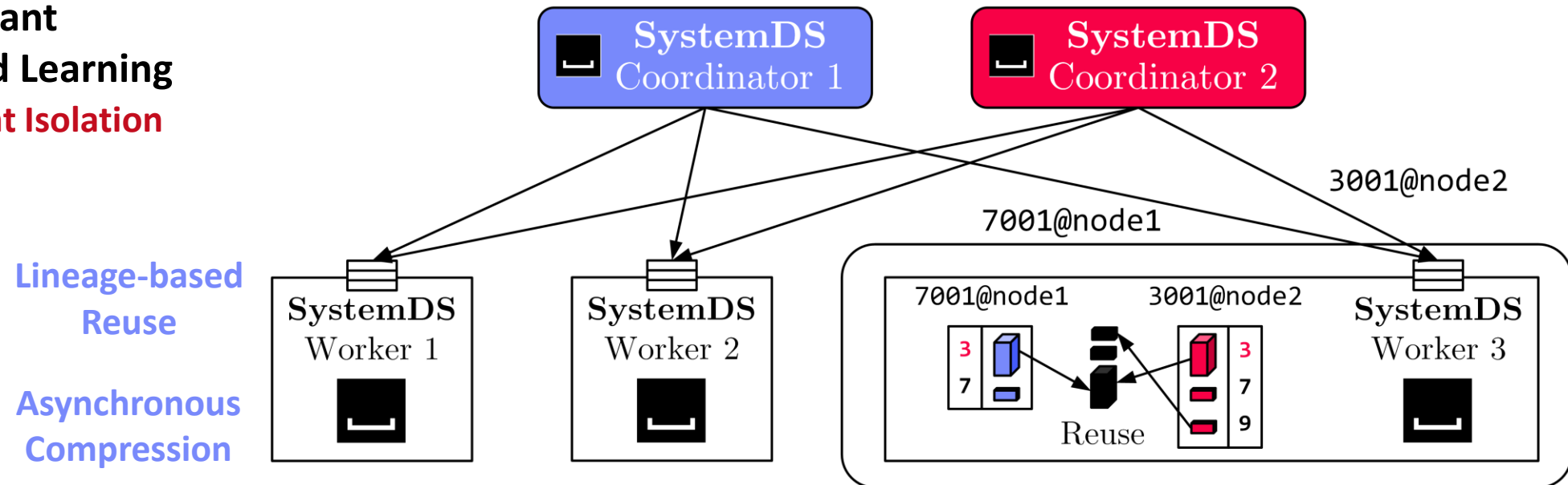
- Key operation for equi-height binning, outlier removal, etc
- **Problem:** requires sorting → infeasible in a federated environment
- Recursive equi-width histograms (w/ 256 bins)



Federated Data Preparation, Learning, and Debugging



- Federated Feature Transformations
- Federated Linear-algebra-based Data Cleaning, Data Preparation, and Model Debugging (e.g., [federated quantiles](#))
- Multi-tenant Federated Learning
 - Tenant Isolation



■ Overview TFF

- **Federated PS algorithms** and **federated second order functions**
- Primarily for simulating federated training, no OSS federated runtime

[<https://www.tensorflow.org/federated/>]



■ #1 Federated PS

- See algorithmic PS improvements

```
iterative_process = tff.learning.build_federated_averaging_process(  
    model_fn, # function for created federated models  
    client_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=0.02),  
    server_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=1.0))
```

■ #2 Federated Analytics

- $r = t(y) \% \% X$
- User-level composition of federated algorithms
- **PET primitives**
(privacy-enhancing technologies)

```
X = ... # tff.type_at_clients(tf.float32)  
by = tff.federated_broadcast(y)  
R = tff.federated_sum(  
    tff.federated_map(X, by, foo_mm), foo_s)  
# note: tff.federated_secure_sum
```

Summary & QA



- **Data-Parallel Parameter Servers**
- **Model-Parallel Parameter Servers**
- **Distributed Reinforcement Learning**
- **Federated Machine Learning**

- **Next Lectures (Part A)**
 - **07 LLM Training and Inference** [Jun 05]
 - **08 Hybrid Execution and HW Accelerators** [Jun 12]
 - **09 Caching, Partitioning, Indexing and Compression** [Jun 19]