11/18 CS240 - Salus

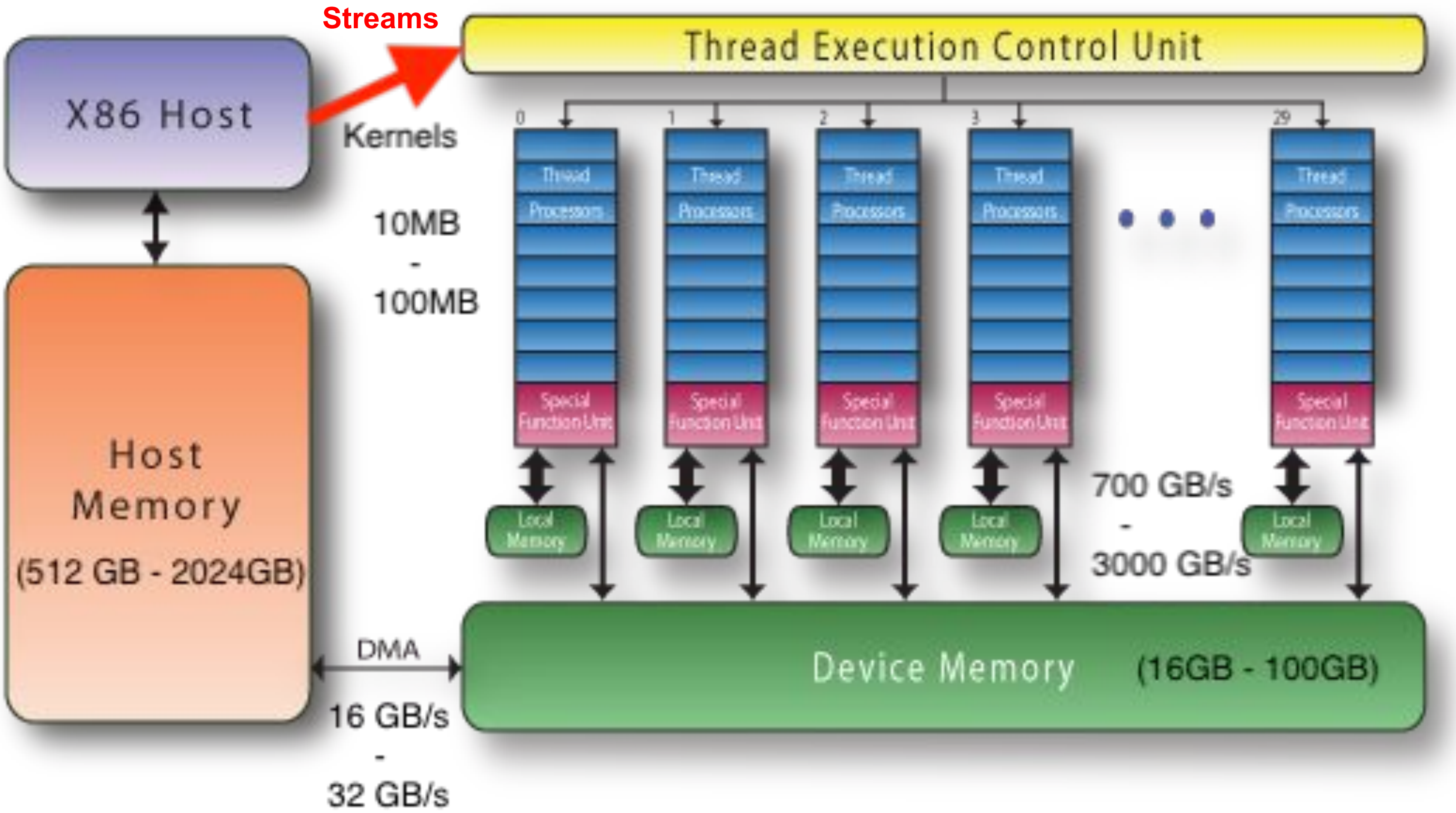
Announcements

For next class (Thursday 11/20)

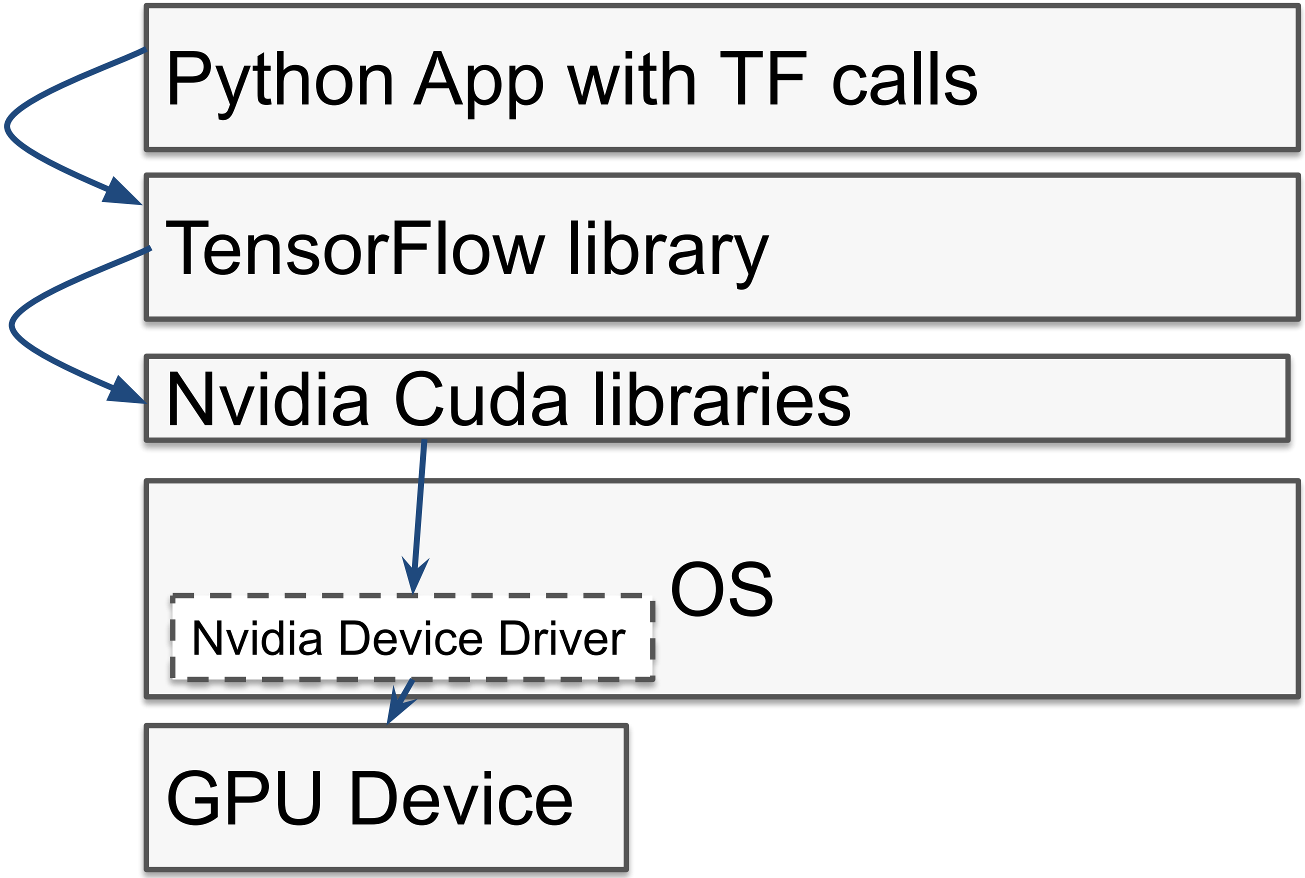
1. Read: Efficient Memory Management for Large Language Model Serving with Paged Attention
2. Submit answers to reading questions (see course schedule) before class

Paper

- Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications
    - 2020 MLSys - Third Conference on Machine Learning and Systems
    - Early systems paper GPUs as OS I/O devices doing machine learning workloads


Streams
Thread Execution Control Unit
X86 Host
Kernels
10MB
-
100MB
0
1
2
3
29
Thread
Processors
Special
Function Unit
Local
Memory
700 GB/s
-
3000 GB/s
Host
Memory
(512 GB - 2024GB)
DMA
16 GB/s
-
32 GB/s
Device Memory
(16GB - 100GB)

Context switch overheads: CPUs vs GPUs


Python App with TF calls
TensorFlow library
Nvidia Cuda libraries
OS
Nvidia Device Driver
GPU Device

Deep Learning Training Job

```
# PERSISTENT (lives across all iterations)
W = gpu_alloc(shape=..., kind="weights")        # model parameters
opt_state = gpu_alloc(shape=..., kind="opt")    # optimizer state

def forward(x, stream):
    # EPHEMERAL activations inside an iteration
    a1 = gpu_alloc(shape=..., kind="activation")
    launch_kernel("matmul",  inputs=[x, W], output=a1, stream=stream)

    a2 = gpu_alloc(shape=..., kind="activation")
    launch_kernel("relu",    inputs=[a1],    output=a2, stream=stream)

    return a2  # a1, a2 freed after iteration ends
```

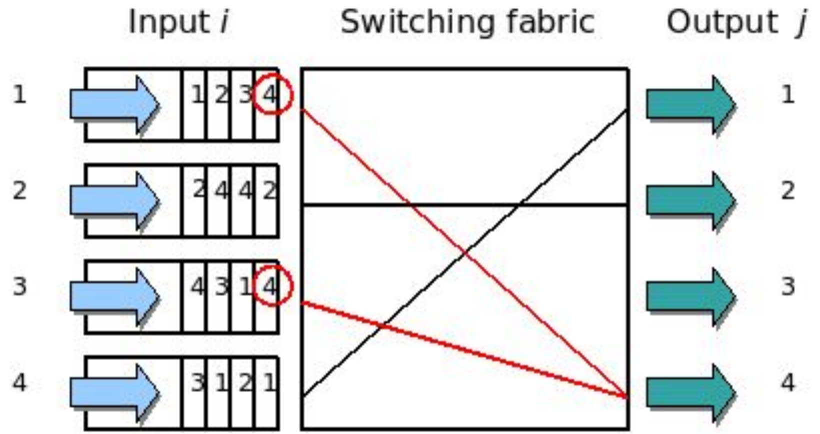
```
for epoch in range(num_epochs):
    for batch_x, batch_y in training_dataset:
        training_step(batch_x, batch_y, stream=job_stream)
        # --- iteration boundary ---
```

```
def training_step(batch_x, batch_y, stream):
    # ---- forward pass ----
    logits = forward(batch_x, stream)            # ephemeral activations
    loss   = gpu_alloc(shape=..., kind="activation")
    launch_kernel("softmax_xentropy",
                  inputs=[logits, batch_y],
                  output=loss,
                  stream=stream)

    # ---- backward pass ----
    grad_W = gpu_alloc(shape=..., kind="gradient")
    launch_kernel("backprop_wrt_W",
                  inputs=[loss, W],
                  output=grad_W,
                  stream=stream)

    # ---- optimizer update (uses persistent state) ----
    launch_kernel("optimizer_update",
                  inputs=[W, grad_W, opt_state],
                  output=W,          # in-place update
                  stream=stream)

    # At the *end* of this function:
    #   - loss, logits, activations, grad_W are EPHEMERAL → freed
    #   - W and opt_state remain PERSISTENT across iterations
```

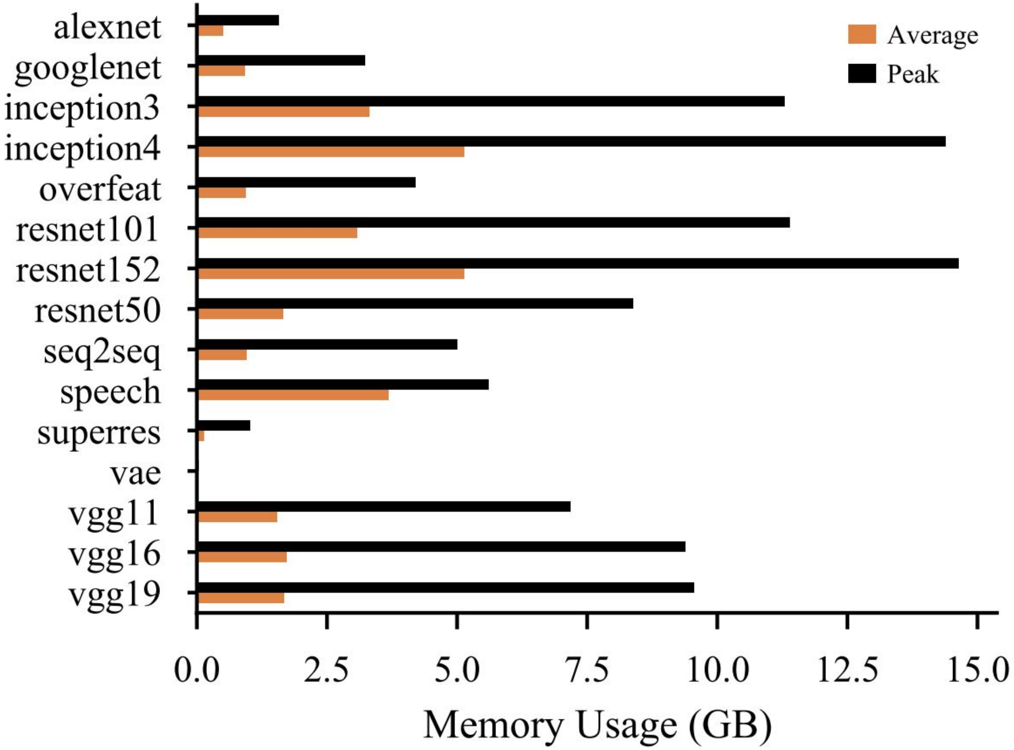
Deep Learning Interference

```
for batch_x in inference_request_stream:
    probs = inference_step(batch_x, stream=job_stream)
    send_results_to_client(probs)
    # --- iteration boundary ---

def inference_step(batch_x, stream):
    # EPHEMERAL activations only; no optimizer, no KV cache
    logits = forward(batch_x, stream)

    probs  = gpu_alloc(shape=..., kind="activation")
    launch_kernel("softmax", inputs=[logits], output=probs, stream=stream)

    # At the end of this call:
    #   - activations + logits + probs are freed (EPHEMERAL)
    #   - W is still resident in GPU memory (PERSISTENT)
    return probs
```

Head of line (HOL) blocking?



Can multiple jobs fit in device memory?



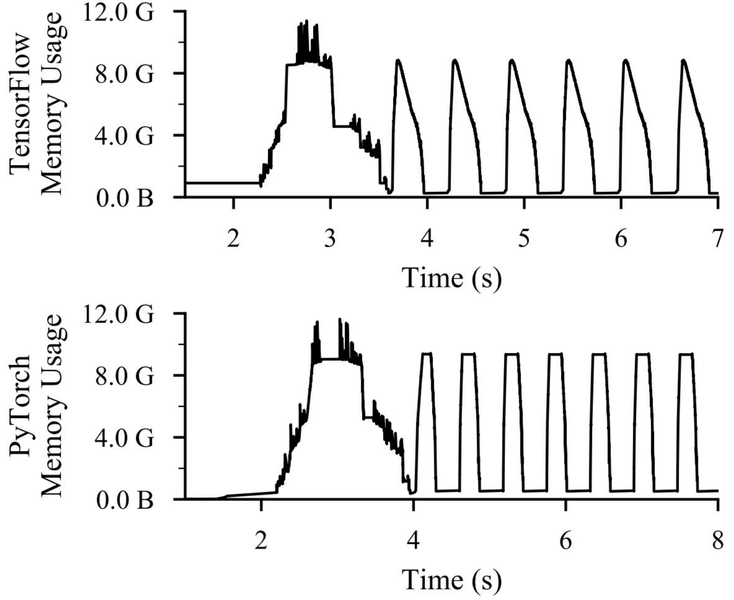
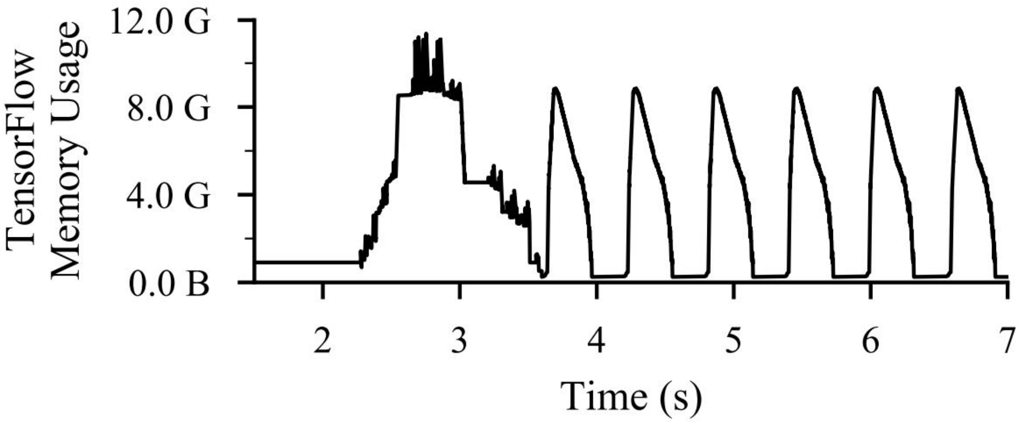


Figure 2. Part of the GPU memory usage trace showing the spatiotemporal pattern when training `resnet101_75` on NVIDIA P100, using TensorFlow and PyTorch.

Salus memory management classifications



- How does Salus tell which are temporary allocations?

Lanes

- What is the Lane relationship with CUDA streams?
- What type of allocations goes into Lanes?
- How is defragmentation handled?

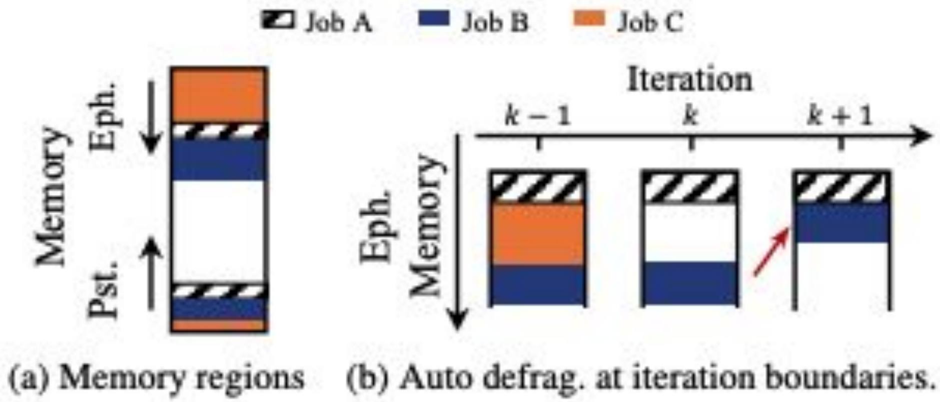
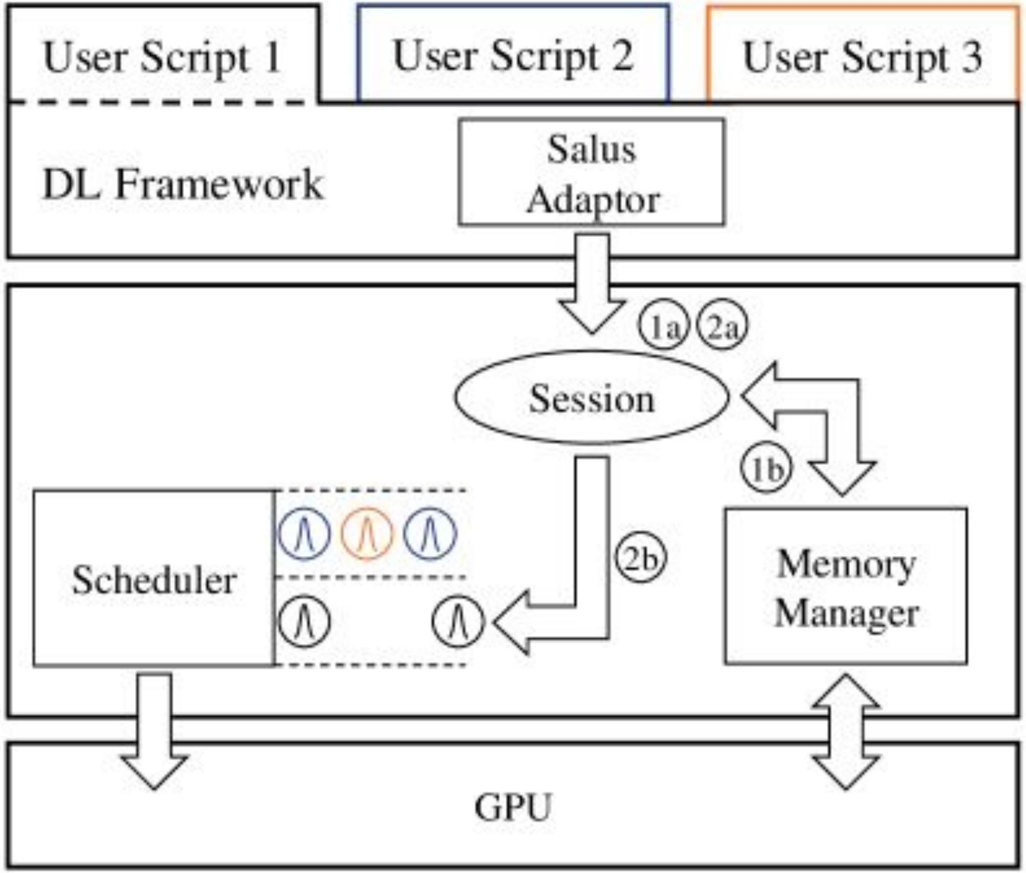


Figure 6. The memory layout of the GPU lane scheme.

Salus



User Script 1
User Script 2
User Script 3
DL Framework
Salus Adaptor
1a
2a
Session
1b
2b
Scheduler
Memory Manager
GPU

Algorithm 1 Find GPU Lane for Job

1: **Input:** P: the job's persistent memory requirement
E: the job's ephemeral memory requirement
C: total memory capacity of the GPU
P_i: persistent memory usage of existing job i
L_j: lane size of existing lane j
$\mathbb{L}$: set of existing lanes
2: **if** $\sum_i P_i + P + \sum_j L_j + E \leq C$ **then**
3:     $lane \leftarrow$ new GPU lane with capacity E
4:     **return** $lane$
5: **end if**
6: **for all** $j \in \mathbb{L}$ **do**
7:     **if** $L_j \geq E$ and is the best match **then**
8:         **return** j
9:     **end if**
10: **end for**
11: **for** $r \in \mathbb{L}$ in L_r ascending order **do**
12:     **if** $\sum_i P_i + P + \sum_j L_j - L_r + E \leq C$ **then**
13:         $L_r \leftarrow E$
14:         **return** r
15:     **end if**
16: **end for**
17: **return** not found

Explain code at lines 2, 6, 11.

Explain safety condition:

$$\sum_{\text{job } i} P_i + \sum_{\text{lane } l} \max_{\text{job } j \text{ in } l} (E_j) \leq C$$

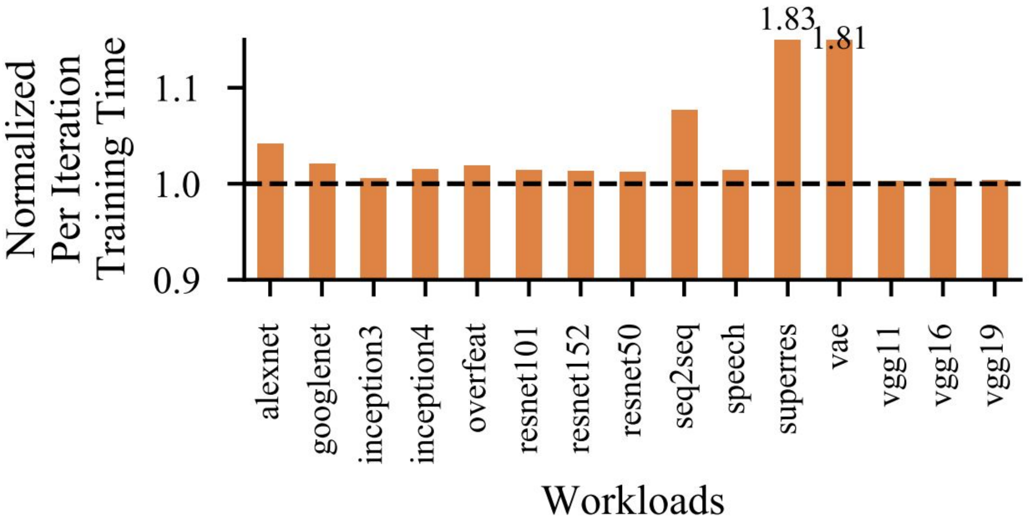


Figure 12. Per iteration time per workload in Salus, normalized by that of TensorFlow. Only the largest batch size for each model is reported, as other batch sizes have similar performance.

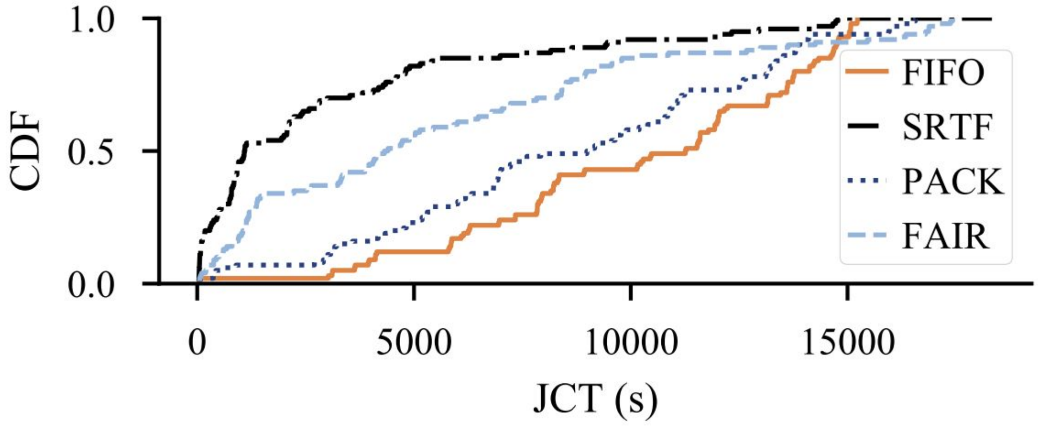
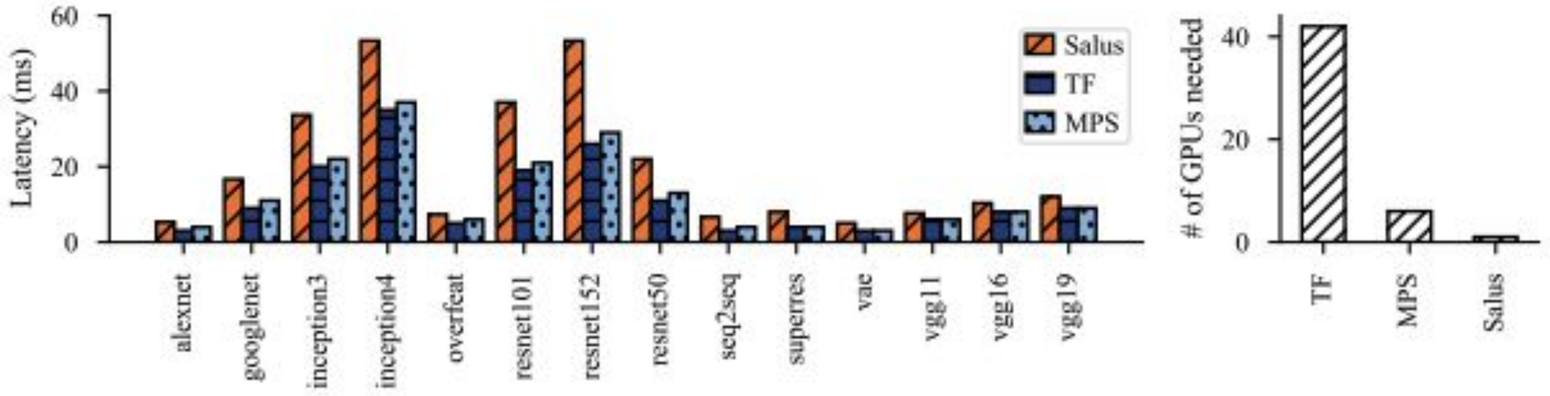


Figure 7. CDFs of JCTs for all four scheduling policies.

Sched.	Makespan	Avg. Queuing	Avg. JCT	95% JCT
FIFO	303.4 min	167.6 min	170.6 min	251.1 min
SRTF	306.0 min	28.6 min	53.4 min	217.0 min
PACK	287.4 min	129.9 min	145.5 min	266.1 min
FAIR	301.6 min	58.5 min	96.6 min	281.2 min

Table 1. Makespan and aggregate statistics for different schedulers.


Latency (ms)
60
40
20
0
Salus
TF
MPS
alexnet
googlenet
inception3
inception4
overfeat
resnet101
resnet152
resnet50
seq2seq
superres
vae
vgg11
vgg16
vgg19
# of GPUs needed
40
20
0
TF
MPS
Salus

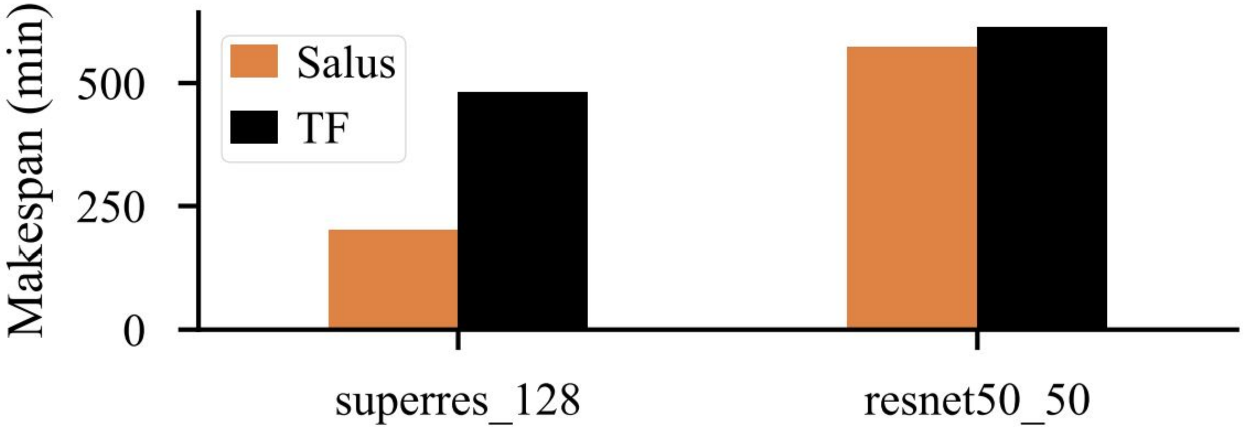


Figure 11. Makespan of two hyper-parameter tuning multi-jobs each of which consists of 300 individual jobs.