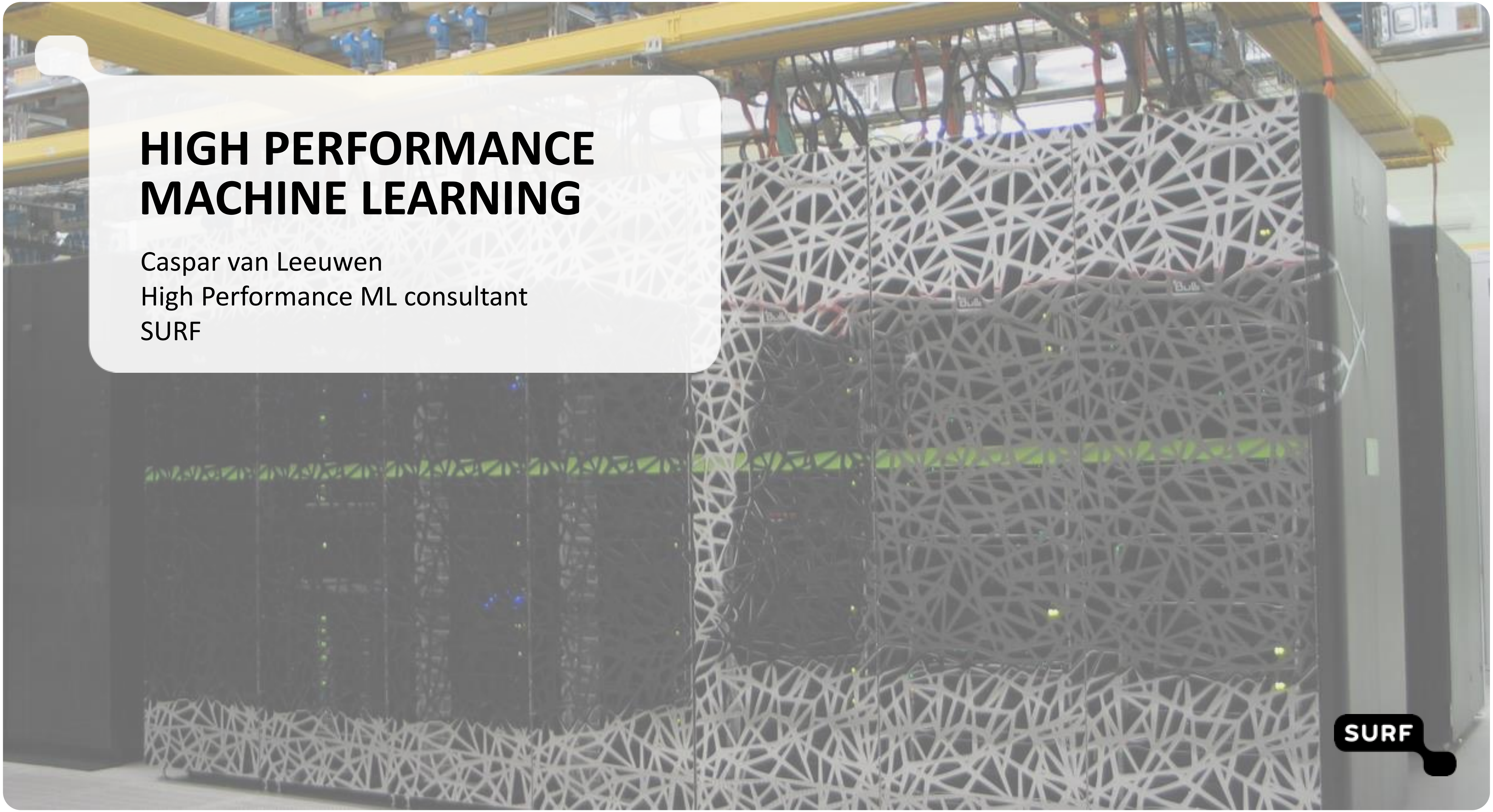
HIGH PERFORMANCE MACHINE LEARNING

Caspar van Leeuwen
High Performance ML consultant
SURF



Program, 2nd day

- 9:00 – 9:30 Introduction to Parallel Computing (Caspar van Leeuwen)
- 9:30 – 10:30 Parallel Computing for Deep Learning: ideas, frameworks, and hardware bottlenecks (Caspar van Leeuwen)
- 10:30 – 11:00 Coffee break
- 11:00 – 11:30 Structure of Deep Learning Frameworks: computational graph, autodiff, and optimizers (Joris Mollinga)
- 11:30 – 12:30 Hands-on: Profiling TensorFlow with TensorBoard (Caspar van Leeuwen)
- 12:30 – 14:00 Lunch Break
- 14:00 – 15:00 Hands-on: Data Parallelism with Horovod (CIFAR10) (Joris/Maxwell)
- 15:00 – 15:30 Coffee break
- 15:30 – 16:15 Introduction to hybrid parallelism (Caspar van Leeuwen)
- 16:15 – 17:00 Open Discussion

Frameworks

Goals:

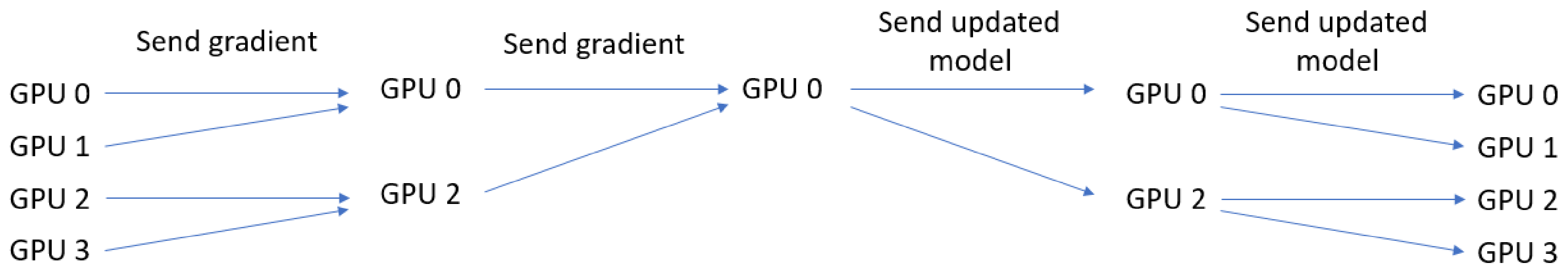
- Get an overview of various frameworks, as well as their optimization options and distribution strategies

Frameworks

- Caffe
- NVCaffe
- IntelCaffe
- PyTorch
- TensorFlow
- Horovod (DL distribution framework only)

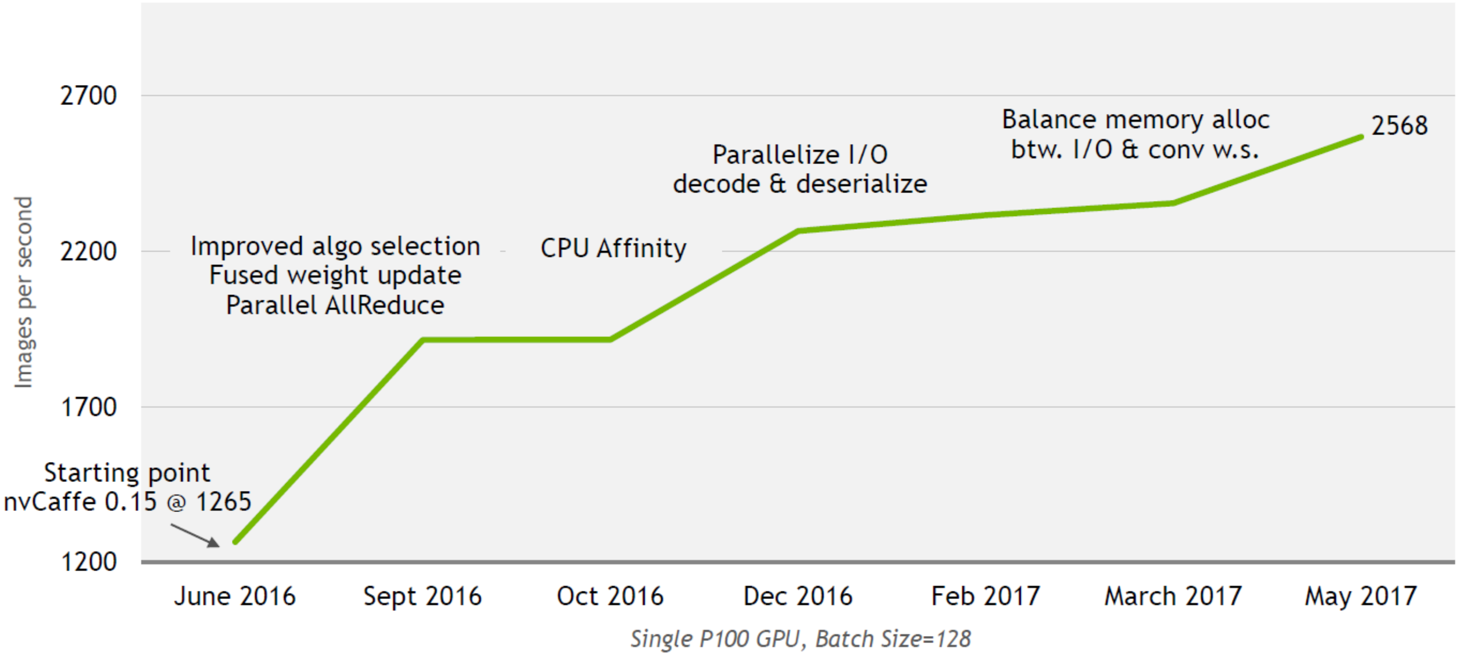
Caffe

- Berkely Vision and Learning Center (BVLC)
- One of the first DL frameworks
- No support for distributed training
- Support for multi-GPU
- Gradient aggregation accordig to tree reduction strategy



NVCaffe

- Nvidia fork of BVLC Caffe, tuned for Nvidia GPUs.
- Integration with cuDNN, NCCL.
- Support for mixed precision
- Support for multi-node as of v 0.17.1
- Nvidia improved performance on their hardware by factor of 2 within a year



IntelCaffe

- Intel fork of BVLC Caffe, tuned for Intel CPUs.
- Integration with MKL-DNN, Intel MLSL (Machine Learning Scaling Library).
- Intel MLSL supports data & model parallelism
- Intel MLSL uses Intel MPI, thus AllReduce strategy can be changed through I_MPI_ADJUST environment variable*

Example environment variables for dual socket system, with 12 cores/socket:

- MLSL_NUM_SERVERS=2 MLSL_SERVER_AFFINITY=“0,1,12,13" OMP_NUM_THREADS=6 KMP_AFFINITY="granularity=fine,compact,1,0"

Launches 2 processes per socket, binds those to the socket, sets 6 threads per process and binds those to cores**

*https://software.intel.com/en-us/mpi-developer-reference-linux-i-mpi-adjust-family-environment-variables

**http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Deep-Learning.pdf

PyTorch

- Probably one of todays most used packages (next to TensorFlow)
- Multi-GPU in one node through *torch.multiprocessing*
- Parallelism accross nodes with *torch.distributed*
- Supports cuDNN and MKL-DNN*
- Supports various communication backends: Gloo, MPI, NCCL
- Recommended backend: see https://pytorch.org/docs/stable/distributed.html

*https://software.intel.com/en-us/articles/getting-started-with-intel-optimization-of-pytorch

TensorFlow

- Probably one of todays most used packages (next to PyTorch)
- Multi-GPU in one node through device placement (*tf.device*)*
- Parallelism accross nodes with *tf.distribute*
- Support for TPUs
- Supports cuDNN and MKL-DNN
- Supports NCCL allreduce
- Pip install NOT well optimized for CPU (doesn't use AVX512 instructions). Ops that use MKL-DNN are ok (MKL-DNN uses AVX512), but non-accelerated ops will be slow. Solution: build from source or install intel-tensorflow (pip install intel-tensorflow)
- Further optimization tips for MKL https://www.tensorflow.org/guide/performance/overview#manual_tuning

*https://www.tensorflow.org/guide/using_gpu

TensorFlow performance tips

- User Tfrecords (or other large packed files) to avoid I/O bottlenecks
- Overlap computation and data preparation using *tf.data.Dataset.prefetch*
- Parallelize data transformation using *tf.data.Dataset.map* (and set *num_parallel_calls* > 1)
- If data fits in memory, use *tf.data.Dataset.cache* (if memory allows: cache after preprocessing the data. That way, preprocessing only needs to be done once). Note: if you use *tf.data.Dataset.cache*, there is no use staging your dataset in /dev/shm beforehand.
- Set *tf.config*'s *intra_op_parallelism_threads* to #physical cores. Determines #threads available to multithreaded ops.
- Set *tf.config*'s *inter_op_parallelism_threads* to #sockets usually works best, but you may experiment with higher values (not higher than #physical cores). Determines #threads available to non-multithreaded ops.
- https://www.tensorflow.org/guide/performance/datasets
- https://www.tensorflow.org/guide/performance/overview#manual_tuning

MKL performance tips

Tune environment variables:

- KMP_BLOCKTIME: Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping. Recommended setting: 0 (but may depend on network).
- KMP_AFFINITY: Enables the run-time library to bind threads to physical processing units. Recommended setting: granularity=fine,verbose,compact,1,0
- KMP_SETTINGS: Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- OMP_NUM_THREADS: Specifies the number of threads to use. Recommended setting: #cores (available to the process). Sometimes, leaving 1 or 2 cores for OS and other tasks is even faster (especially in many-core nodes).

https://www.tensorflow.org/guide/performance/overview#manual_tuning

https://software.intel.com/en-us/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference

Horovod

Is a distribution framework for deep learning (not a deep learning framework itself). Design goals:

- Minimal code changes to make serial program distributed
- High performance distribution

https://github.com/horovod/horovod

Horovod

- Support for Keras, MXNet, TensorFlow and PyTorch.
- Supports MPI and NCCL as communication backends
- Requires about 6 lines of code change
- Supports Tensor Fusion to batch small allreduce operations (remember: small allreduce operations hit latency bottleneck)
- Has it’s own profiling ability, making it easy to assess communication overhead

SURF