THE ROAD TO SOFTWARE ENGINEERING 2.0

Steven Dalton

WINDING ROAD

Oblivious-Assisted-

WINDING ROAD

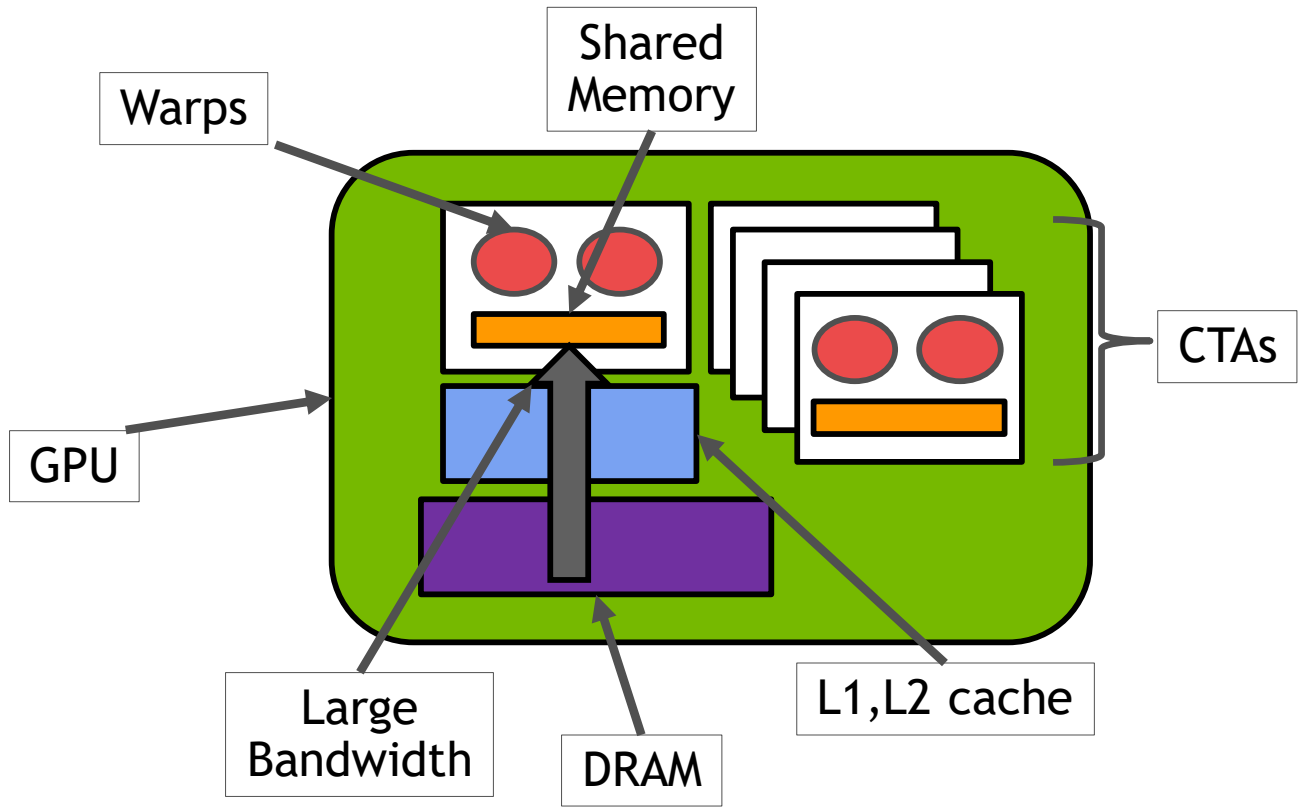
Asset Generation

Alan Amati [https://www.flickr.com/photos/12376670@N07/14329298961]
[http://demos.algorithmia.com/deep-style/public/images/thumb/crafty_painting.jpg]

MOTIVATION

In the beginning…

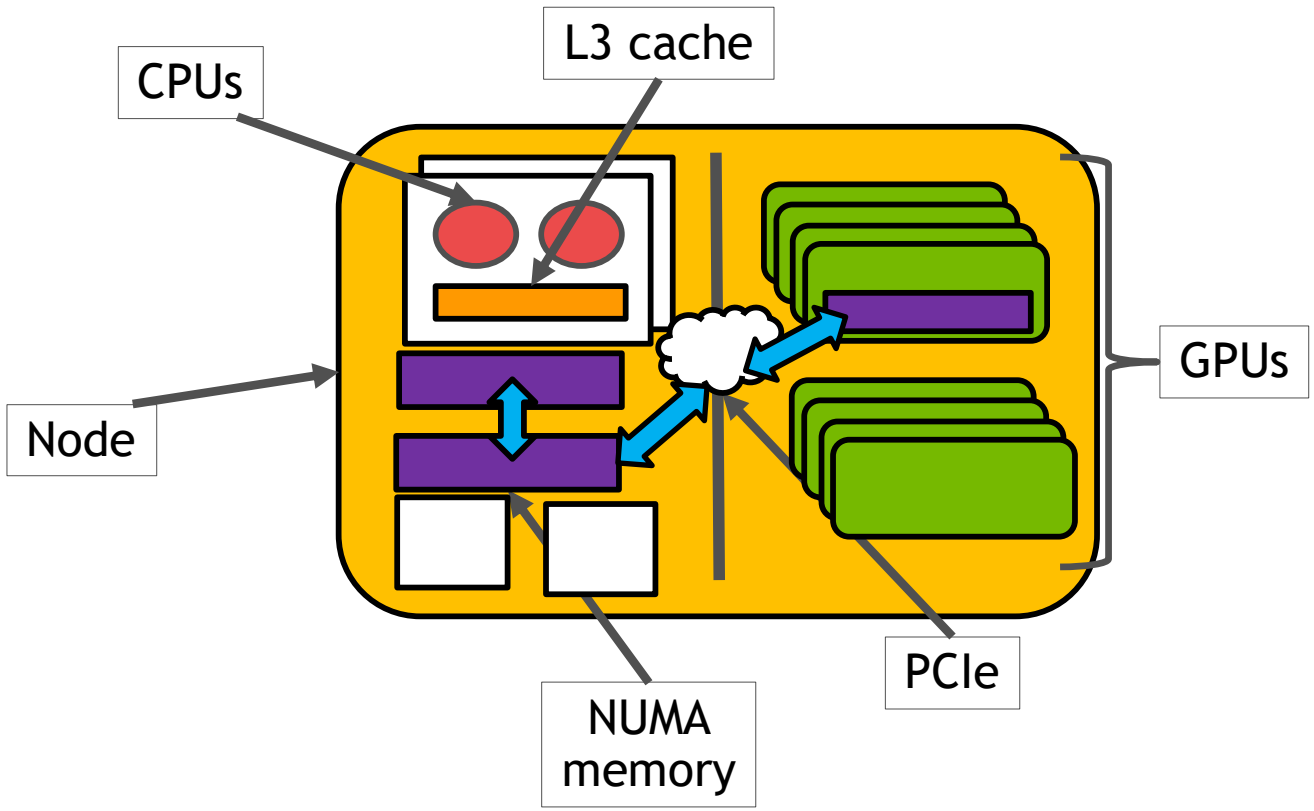
- Single GPU ports
- Avalanche of work concerning sparse methods
- Nuances of throughput oriented computing
- *Simple* communication patterns



MOTIVATION

Evolved into…

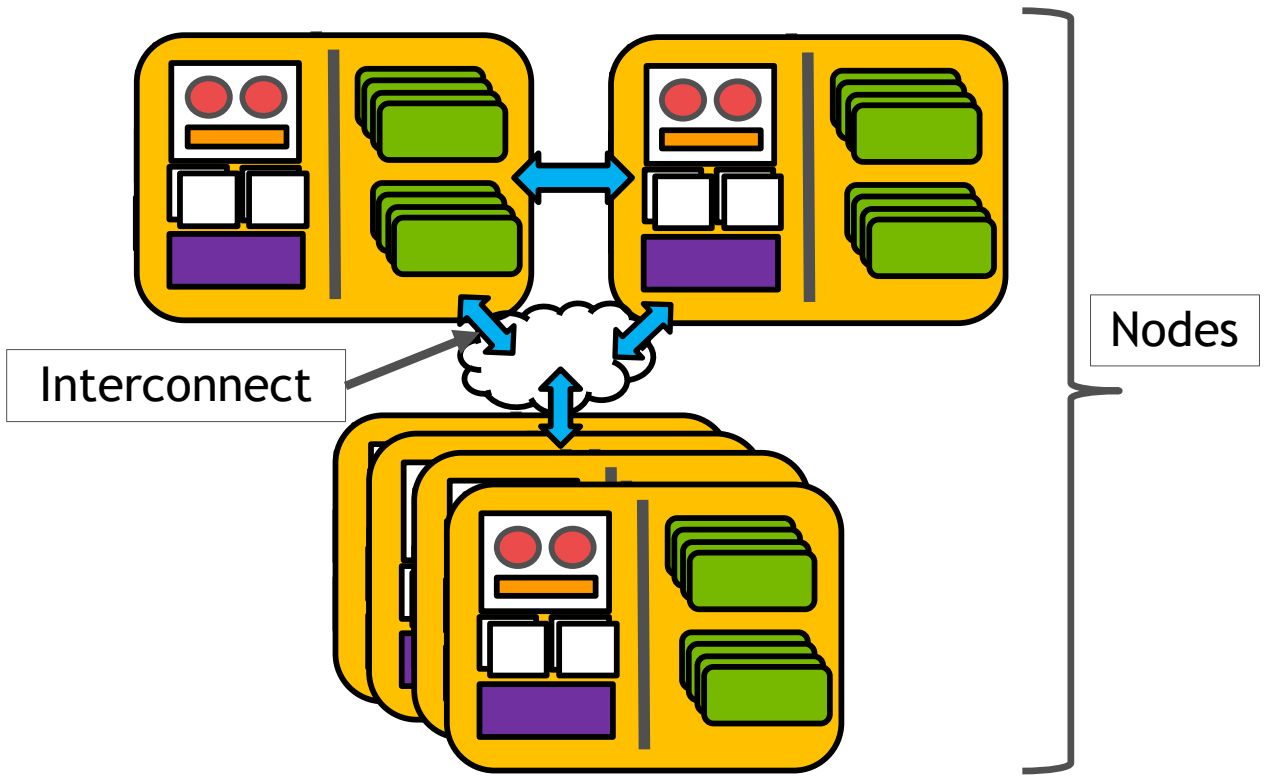
- Multi-GPU ports
- Inter-GPU communication & scheduling
- Still they came…
- More concerns about decompositions



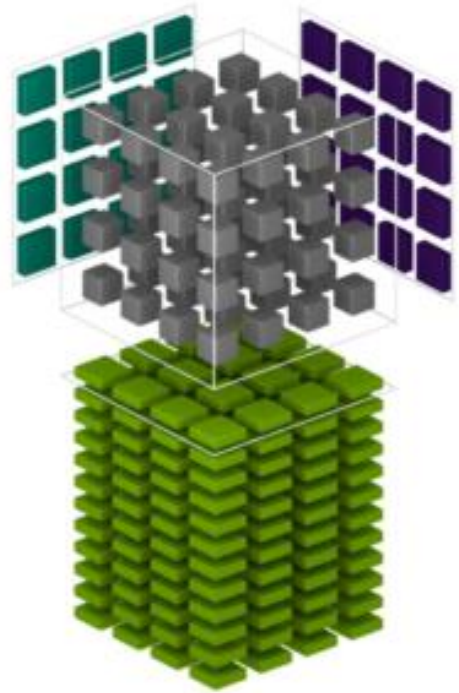
MOTIVATION

Finally

- Multi-node ports
- Inter- + Intra-node communication & scheduling
- Usual suspects still available
- Harder to reason about complex applications across entire system



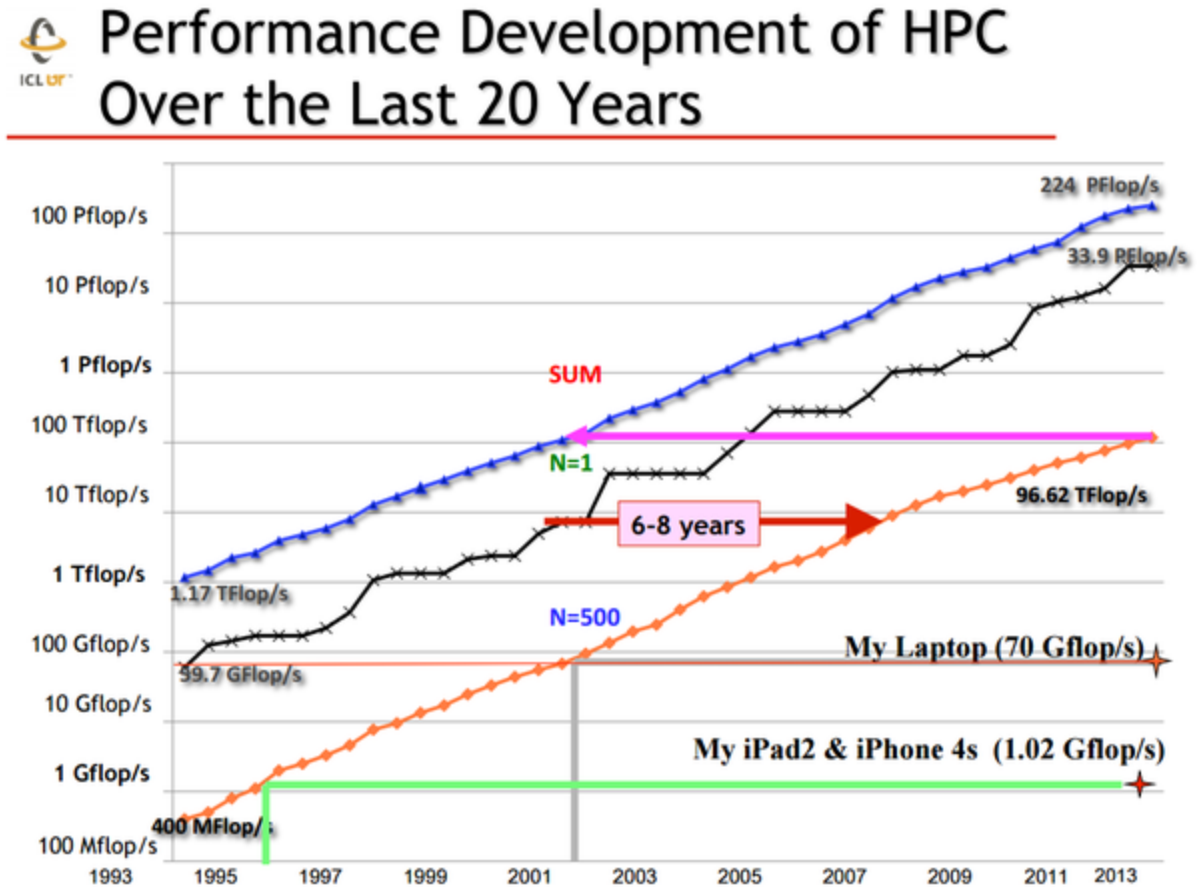
TENSOR CORES



$$D = \underbrace{\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}_{\text{FP16 or FP32}} \underbrace{\begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}}_{\text{FP16}} + \underbrace{\begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}}_{\text{FP16}}_{\text{FP16 or FP32}}$$

https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

TOP500



https://www.hpcwire.com/2013/06/21/the_top_500_visualized/

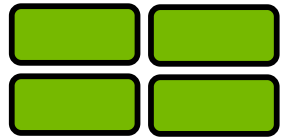
1.0

```cpp
#include <thrust/device_vector.h>
#include <thrust/sort.h>

#include <thrust/system/cuda/execution_policy.h>

void main(void)
{
  cudaStream_t s; cudaStreamCreate(&s);
  thrust::device_vector<int> vec(10, …);
  thrust::sort(thrust::cuda::par.on(s),
                        vec.begin(),
                        vec.end());
}
```

C++/CUDA

GPU

N GPUs



Thrust/Agency

CUB/CUTLASS

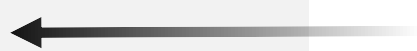
THRUST

```
#include <vector>
#include <algorithms>

void main(void)
{
  thrust::vector<int> vec(10, …);
  thrust::sort(vec.begin(), vec.end());
}
```

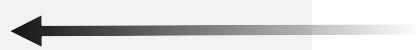
- Collection (extensions) of standard algorithms and containers
- Transparent support for a number of backends (omp, cuda, …)

SORT

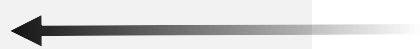
```
#include <vector>
#include <algorithms>



void main(void)
{

  std::vector<int> vec(10, …);
  std::sort(
              vec.begin(),
              vec.end());
}
```



Sort header



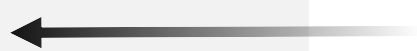
Data



Sort

THRUST SORT

```
#include <thrust/device_vector.h>
#include <thrust/sort.h>



void main(void)
{

  thrust::device_vector<int> vec(10, …);
  thrust::sort(
               vec.begin(),
               vec.end());
}
```



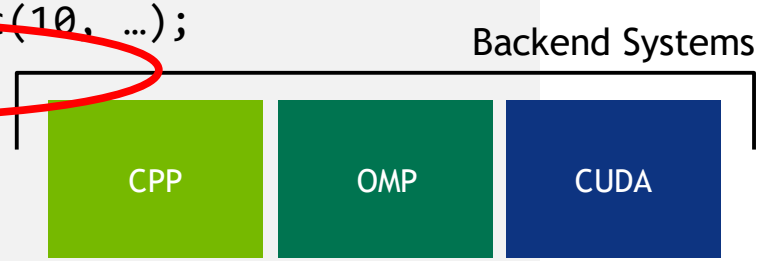
Sort header

Data

Sort

THRUST SORT

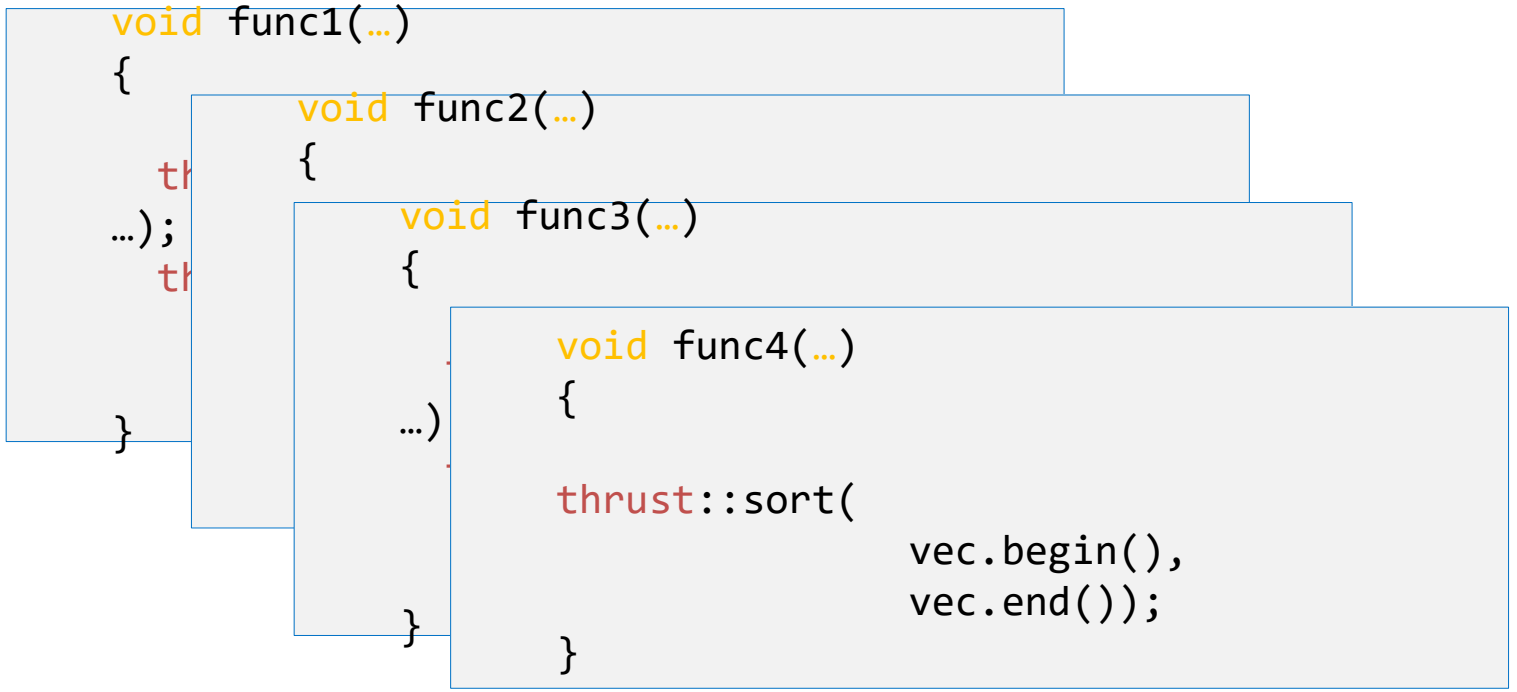
```
#include <thrust/device_vector.h>
#include <thrust/sort.h>



void main(void)
{

  thrust::device_vector<int> vec(10, …);
  thrust::sort(
               vec.begin(),
               vec.end());
}
```



MOTIVATION

Profiling Thrust-based library

Several sorting calls across multiple functions/files



PROFILING

Possible Thrust profiling solutions

REDESIGN INTERFACE

How? What would new thrust::sort require?

How would you profile STL routines?

DO-IT-YOURSELF

```
timer t;
thrust::sort(begin, end);
t.elapsed_milliseconds();
```

INTERCEPT CALLS

```
LD_PRELOAD=prof_thrust.so exec_file
```

EXECUTION POLICIES

```
thrust::sort(exec,
             vec.begin(),
             vec.end());
```

PROFILING

Possible Thrust profiling solutions

REDESIGN INTERFACE

How? What would new thrust::sort require?

How would you profile STL routines?

DO-IT-YOURSELF

```
timer t;
thrust::sort(begin, end);
t.elapsed_milliseconds();
```

INTERCEPT CALLS

LD_PRELOAD=prof_thrust.so exec_file

EXECUTION POLICIES

```
thrust::sort(exec,
             vec.begin(),
             vec.end());
```

PROFILING

Possible Thrust profiling solutions

REDESIGN INTERFACE

How? What would new thrust::sort require?

How would you profile STL routines?

DO-IT-YOURSELF

```
timer t;
thrust::sort(begin, end);
t.elapsed_milliseconds();
```

INTERCEPT CALLS

```
LD_PRELOAD=prof_thrust.so exec_file
```

EXECUTION POLICIES

```
thrust::sort(exec,
             vec.begin(),
             vec.end());
```

PROFILING

Possible Thrust profiling solutions

REDESIGN INTERFACE

How? What would new thrust::sort require?

How would you profile STL routines?

DO-IT-YOURSELF

```
timer t;
thrust::sort(begin, end);
t.elapsed_milliseconds();
```

INTERCEPT CALLS

```
LD_PRELOAD=prof_thrust.so exec_file
```

EXECUTION POLICIES

```
thrust::sort(exec,
             vec.begin(),
             vec.end());
```

THRUST SORT

```
#include <thrust/device_vector.h>
#include <thrust/sort.h>

#include <thrust/system/cuda/execution_policy.h>

void main(void)
{
  cudaStream_t s; cudaStreamCreate(&s);
  thrust::device_vector<int> vec(10, …);
  thrust::sort(thrust::cuda::par.on(s),
               vec.begin(),
               vec.end());
}
```

Policy header

Sort with policy

THRUST SORT

```
#include <thrust/device_vector.h>
#include <thrust/sort.h>

#include <thrust/system/cuda/execution_policy.h>

void main(void)
{
  cudaStream_t s; cudaStreamCreate(&s);
  thrust::device_vector<int> vec(10, …);
  thrust::sort(thrust::cuda::par.on(s),
               vec.begin(),
               vec.end());
}
```

Policy header

Sort with policy

WHAT?

HOW?

EXECUTION-POLICY DESIGN PATTERN

```
template<typename Policy, typename Iterator>
void sort(Policy& exec, Iterator begin, Iterator end)
{
    // add generic sort to local context
    using generic::sort;
    // use ADL lookup for dispatching sort
    sort(derived_cast(exec), begin, end);
}

template<typename Iterator>
void sort(Iterator begin, Iterator end)
{
    // no policy specified
    // use generic sort
    sort(exec, begin, end);
}
```

AGENCY

Overview

```
template<typename ExecutionPolicy>
void saxpy(Policy policy, float a, float *x, float *y, int n)
{
    bulk_invoke(policy(n), [=](agent& self)
    {
            int i = self.index();
            x[i] = a * x[i] + y[i];
    }
}
```

https://agency-library.github.io

AGENCY

```
void par_saxpy(float a, float *x, float *y, int n)
{
    bulk_invoke(par(n), [=](parallel_agent& self)
    {
            int i = self.index();
            x[i] = a * x[i] + y[i];
    }
}

void seq_saxpy(float a, float *x, float *y, int n)
{
    bulk_invoke(seq(n), [=](sequenced_agent& self)
    {
            int i = self.index();
            x[i] = a * x[i] + y[i];
    }
}
```

https://agency-library.github.io

AGENCY

Futures

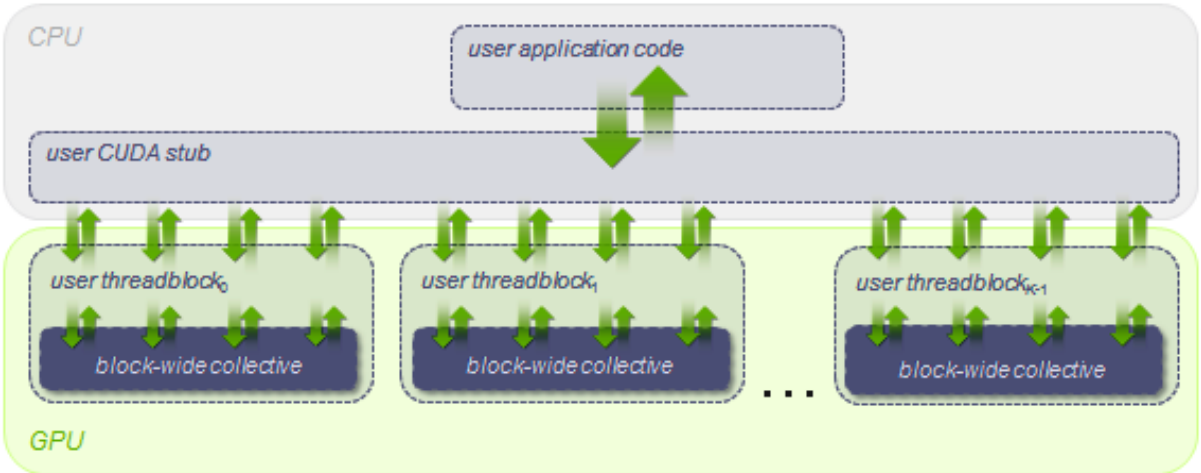
```
future<void> async_par_saxpy(float a, float *x, float *y, int n)
{
    return bulk_async(par(n), [=](parallel_agent& self)
    {
            int i = self.index();
            x[i] = a * x[i] + y[i];
    }
}
```

- Extended support for asynchronous execution using futures
- Composable execution of dependent tasks

http://nvlabs.github.io/cub/

CUB

Overview



- CUDA focused
- Collection of primitives similar to Thrust

http://nvlabs.github.io/cub/

CUTLASS

Domain Specific

```
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
       for(int k = 0; k < N; k++)
           C[i][j] += A[i][k]*B[k][j];
```

https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

Domain Specific

```
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
       for(int k = 0; k < N; k++)
           C[i][j] += A[i][k]*B[k][j];
```

https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

Domain Specific

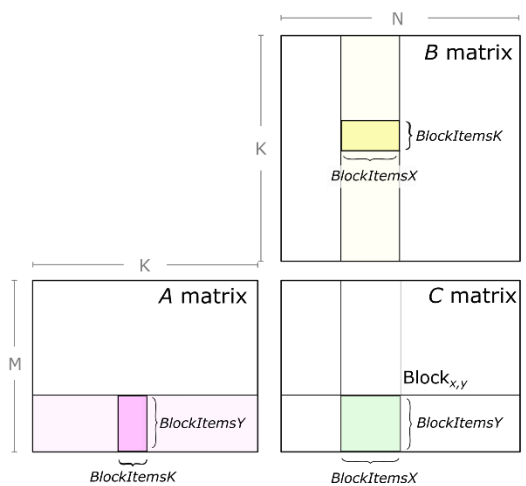
```
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
       for(int k = 0; k < N; k++)
           C[i][j] += A[i][k]*B[k][j];
```

https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

Domain Specific

```
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
       for(int k = 0; k < N; k++)
           C[i][j] += A[i][k]*B[k][j];
```

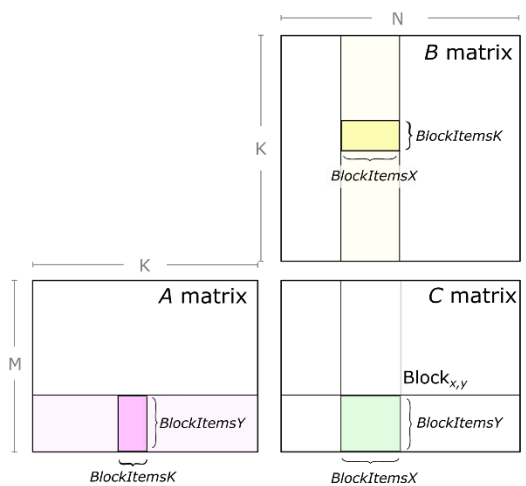


https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

Domain Specific

```
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
        for(int k = 0; k < N; k++)
            C[i][j] += A[i][k]*B[k][j];
```

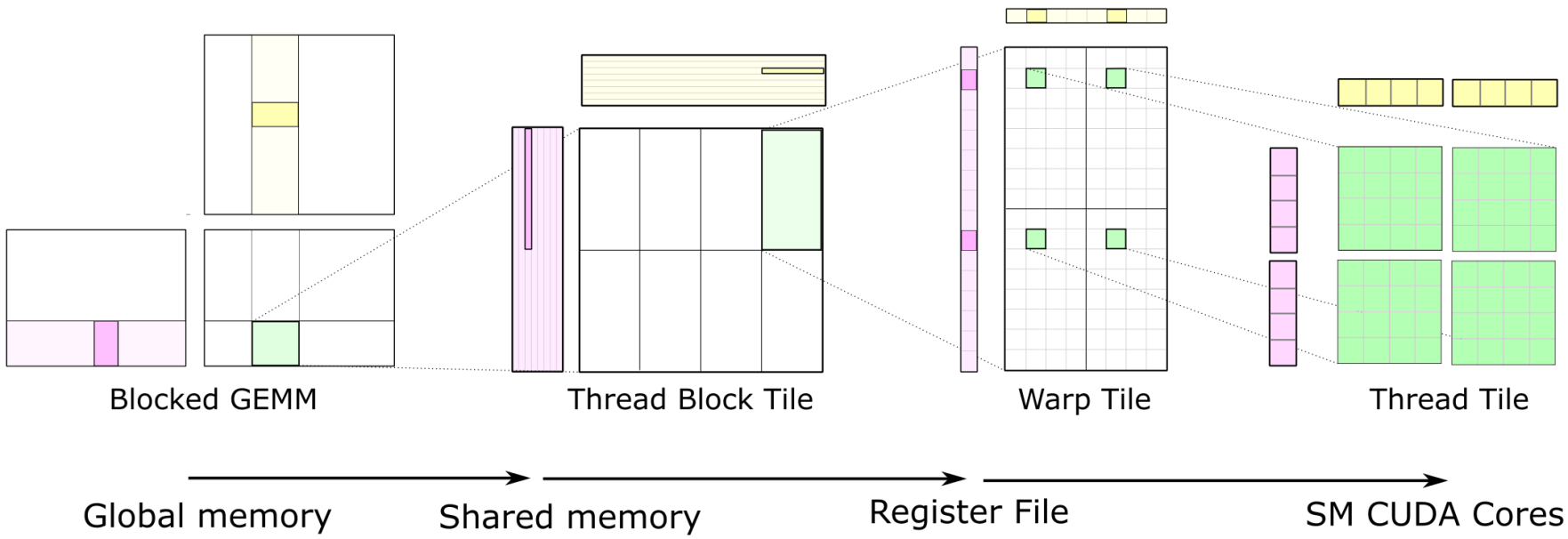



```
__global__ void cuda_dgemm_shmem(int n, double alpha, const double *A, const double *B, double beta, double *C)
{
    int thread_col = threadIdx.x;
    int thread_row = threadIdx.y;
    int row = blockDim.y * blockIdx.y + threadIdx.y;
    int col = blockDim.x * blockIdx.x + threadIdx.x;
    int aBegin = n * blockDim.x * block_row;
    int aEnd = aBegin + n-1;
    int bBegin = blockDim.x * block_col;
    int bStep = n * blockDim.x;
   double Csub = 0;
   for (int a=aBegin, b=bBegin, istep=0; a <= aEnd; a+= blockDim.x, b+=bStep, ++istep)
   {
      __shared__ double As[blockDim.x][blockDim.x];
      __shared__ double Bs[blockDim.x][blockDim.x];
      if ((istep*blockDim.x+thread_col < n) && (block_row*blockDim.x+ thread_row < n))
         As[thread_row][thread_col] = A[a + n * thread_row + thread_col];
      else
         As[thread_row][thread_col] = 0;

      if ((block_col*blockDim.x+thread_col < n) && (istep*blockDim.x + thread_row < n))
         Bs[thread_row][thread_col] = B[b + n * thread_row + thread_col];
      else
         Bs[thread_row][thread_col] = 0;

      __syncthreads();

      for (int k = 0; k < blockDim.x; ++k)
         Csub += As[thread_row][k] * Bs[k][thread_col];

      __syncthreads();

      int c = n * blovkDim.x * block_row + blockDim.x * block_col;
      if ((block_col*blockDim.x+thread_col < n) && (block_row*blockDim.x+ thread_row < n))
        C[c + n * thread_row + thread_col] = alpha * Csub + beta * C[c +n * thread_row + thread_col];
}
```

https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

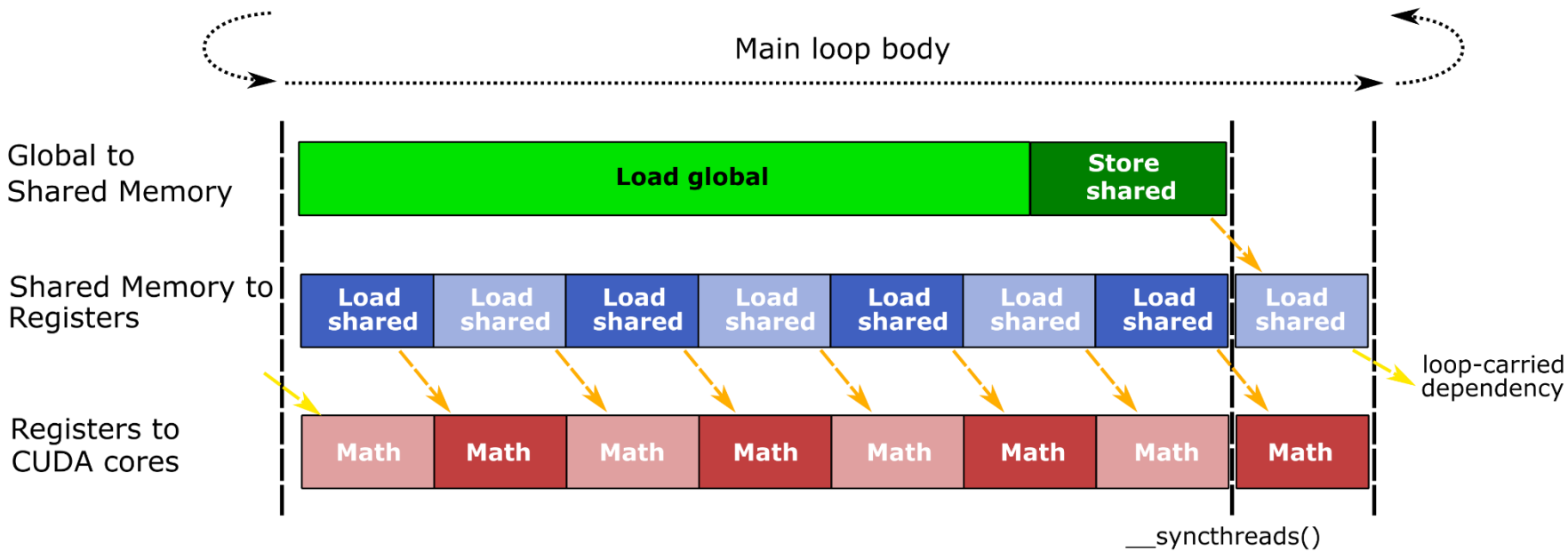
Hierarchical Execution for DLA



https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

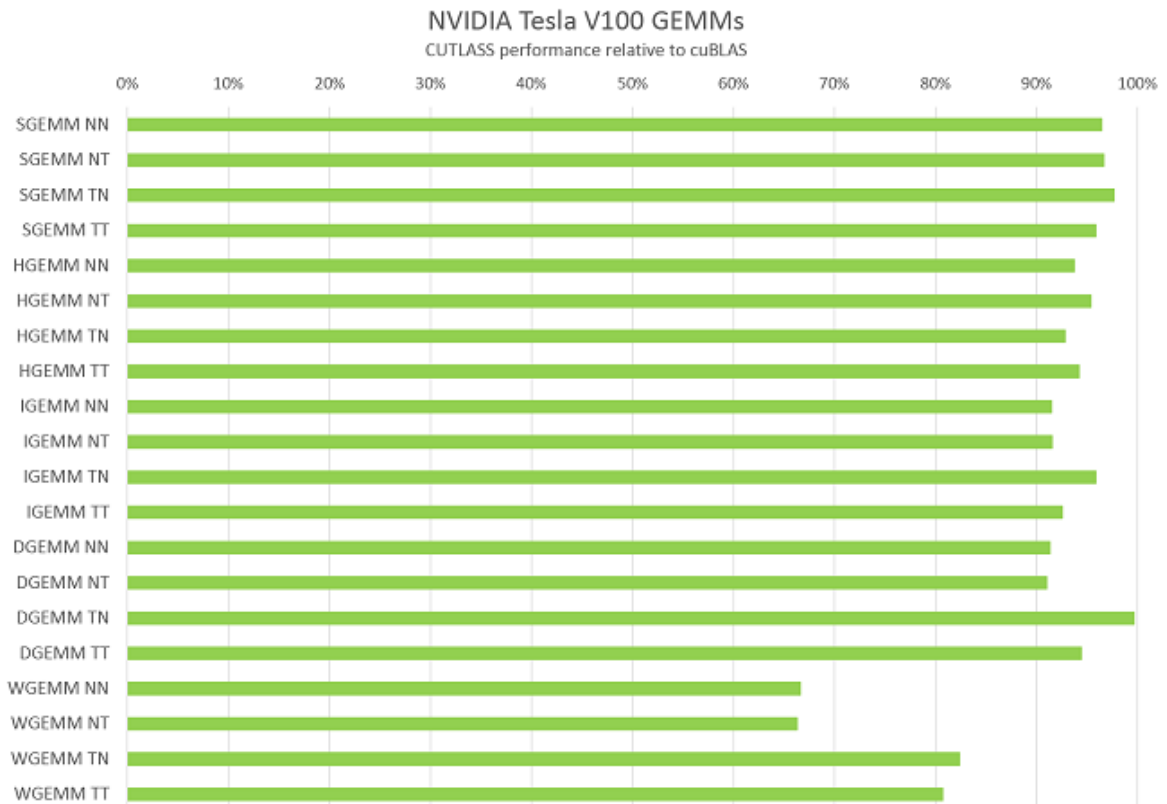
Hierarchical Execution for DLA



https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

CUTLASS

V100 Performance

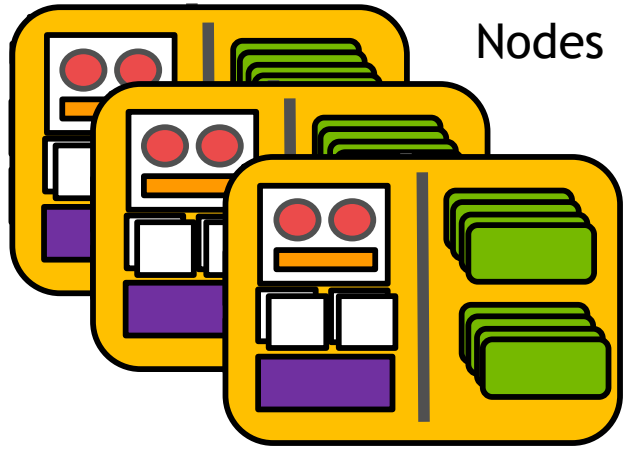


https://devblogs.nvidia.com/parallelforall/cutlass-linear-algebra-cuda/

1.5

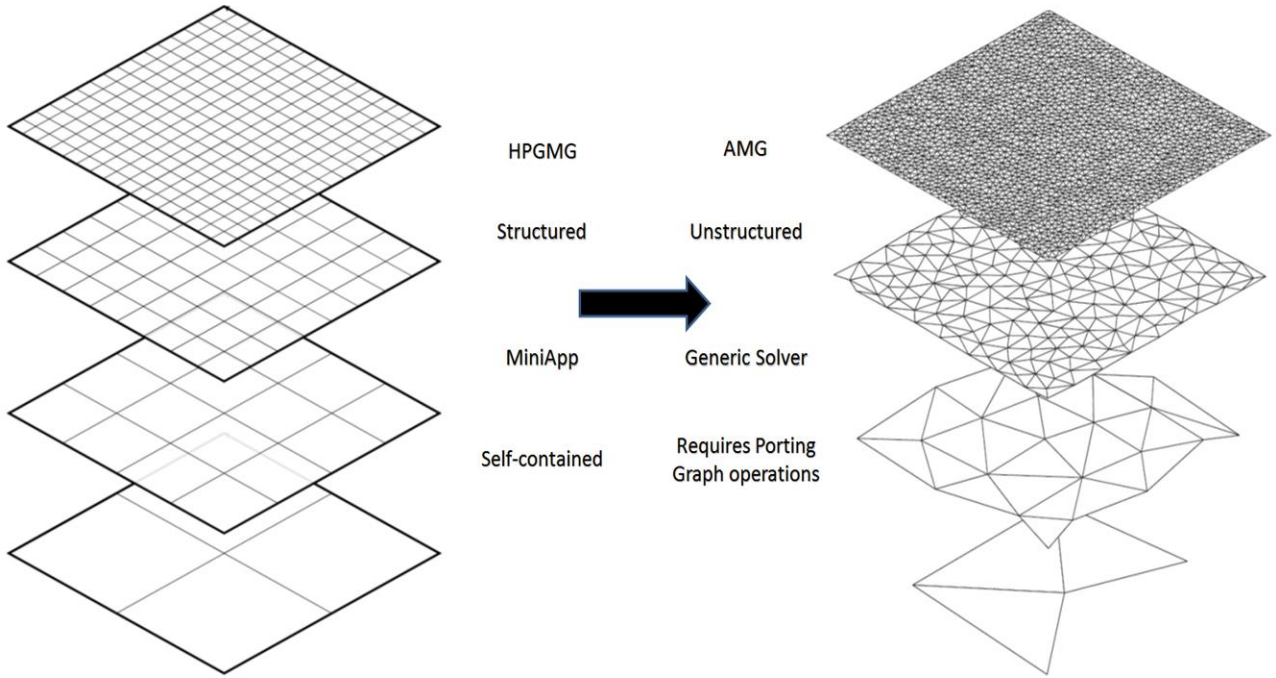
Runtimes

Nodes



APPLICATION

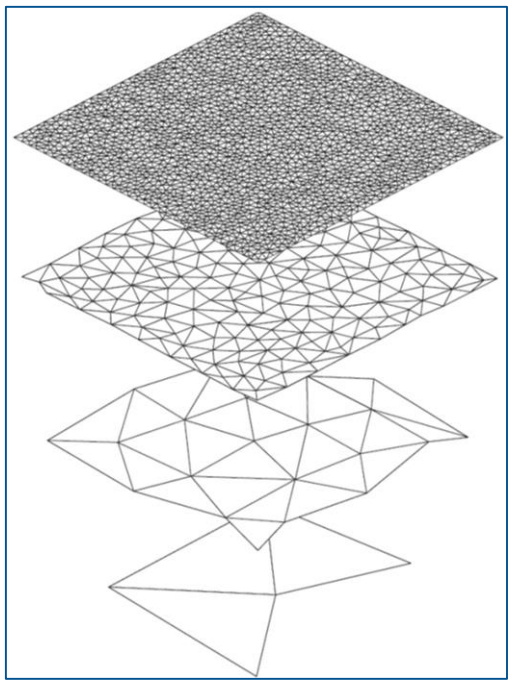
GMG to AMG



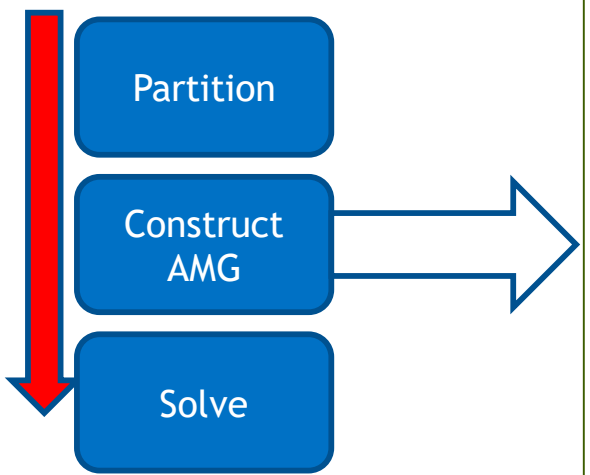
- GMG is a good building block to understand MG solvers
- Generalization to AMG utilizes very similar routines during the solve phase, boundary communication, coarse solver, etc.

CRITICAL PATH

Setup



PATH



Algorithm 1: AMG Setup: `setup`

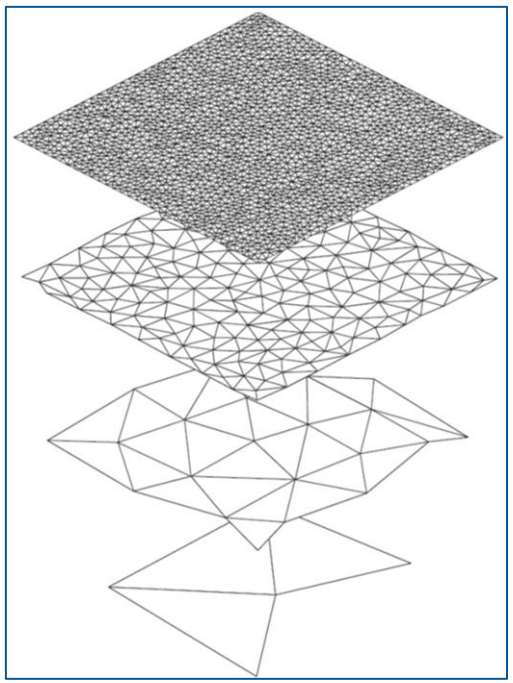
parameters: A, B
return: $A_0, \ldots, A_M$, $P_0, \ldots, P_{M-1}$

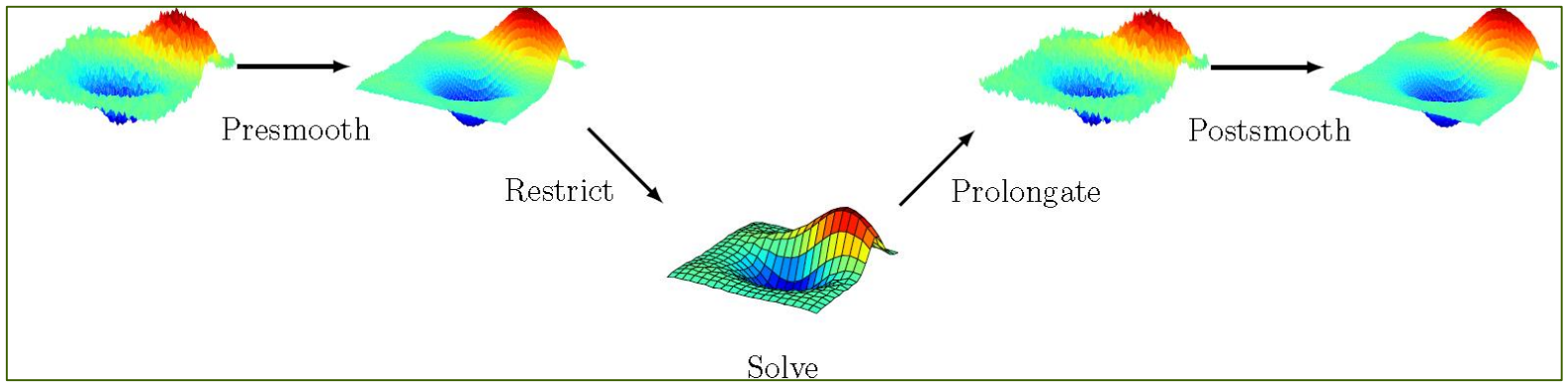
$A_0 \leftarrow A$, $B_0 \leftarrow B$
for $k = 0, \ldots, M$

1. $C_k \leftarrow \texttt{strength}(A_k)$ {strength-of-connection}
2. $Agg_k \leftarrow \texttt{aggregate}(C_k)$ {construct coarse aggregates}
3. $T_k,\ B_{k+1} \leftarrow \texttt{tentative}(Agg_k, B_k)$ {form tentative interpolation}
4. $P_k \leftarrow \texttt{prolongator}(A_k, T_k)$ {improve interpolation}
5. $R_k \leftarrow P_k^T$ {transpose}
6. $A_{k+1} \leftarrow R_k A_k P_k$ {coarse matrix, triple-matrix product}

CRITICAL PATH

Cycling/Solve





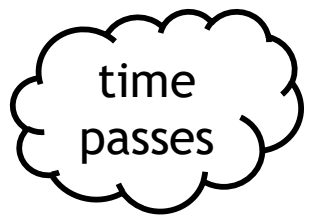
ADOPTING LEGION

Back & Forth

What are you working on?

Programming system for distributed methods, you?

Runtime system for same thing, you should check it out.



Runtime system???

Hey Mike, multi-GPU seems reasonable not sure about multi-node, what was that system again?

Let me tell you all about it!

HPC PROGRAMMING GOALS

What are the critical design criteria for productivity?

High Performance

We must be fast

Performance Portability

Across many kinds of machines and over many generations

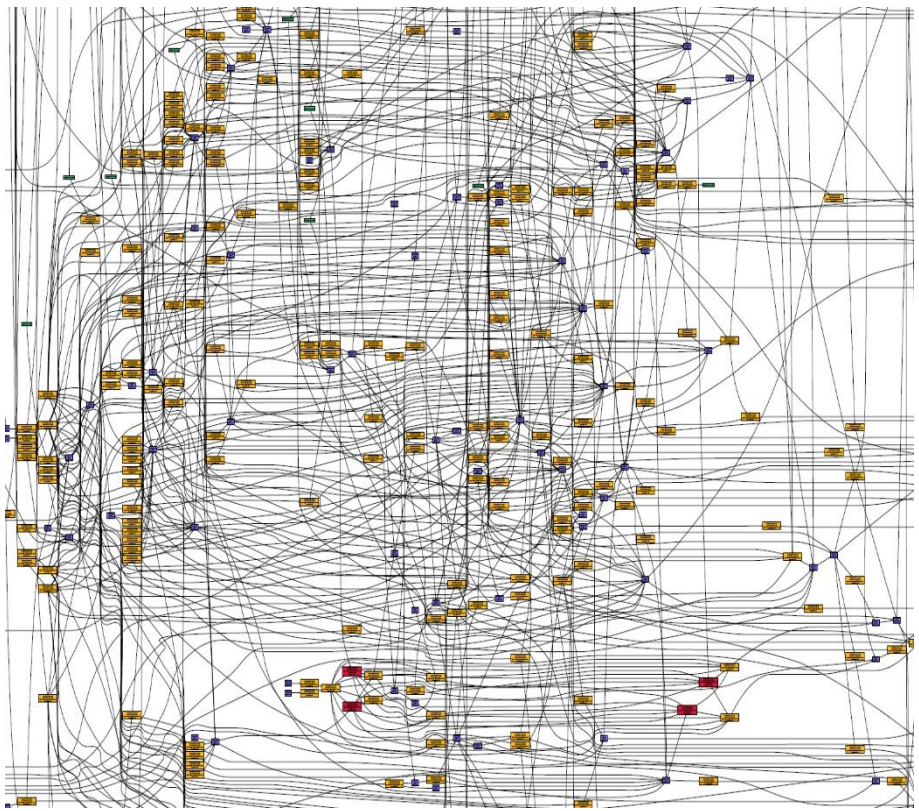
Programmability

Sequential semantics, parallel execution

CAN WE FULFILL THESE GOALS TODAY?

We can…

… at great cost: Programmer Pain



Task graph for one time step on one node…

… of a mini-app

Do you want to schedule that graph?
(High Performance)

Do you want to re-schedule that graph for every new machine?
(Performance Portability)

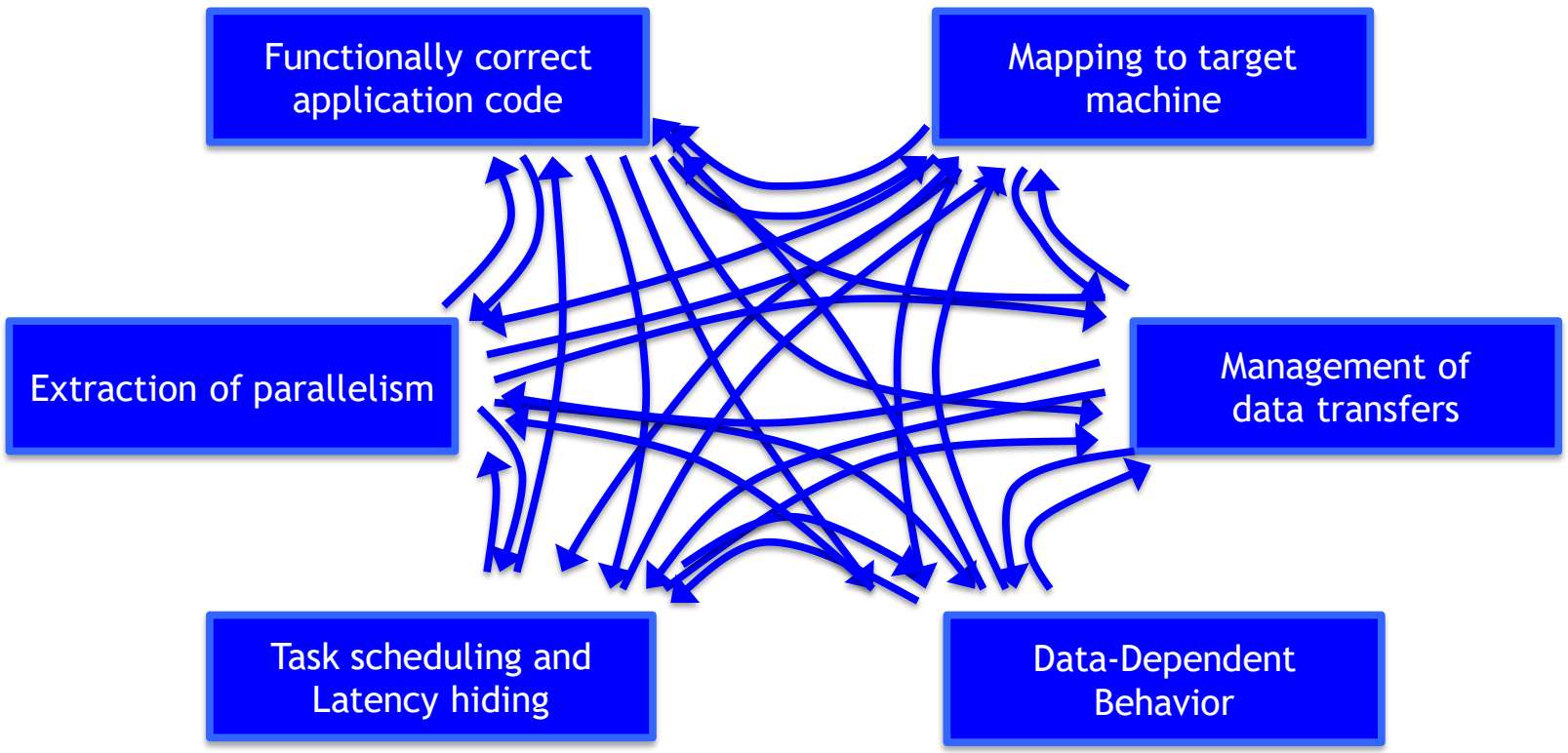
Do you want to be responsible for generating that graph?
(Programmability)

Today: programmer's responsibility

Tomorrow: programming system's responsibility

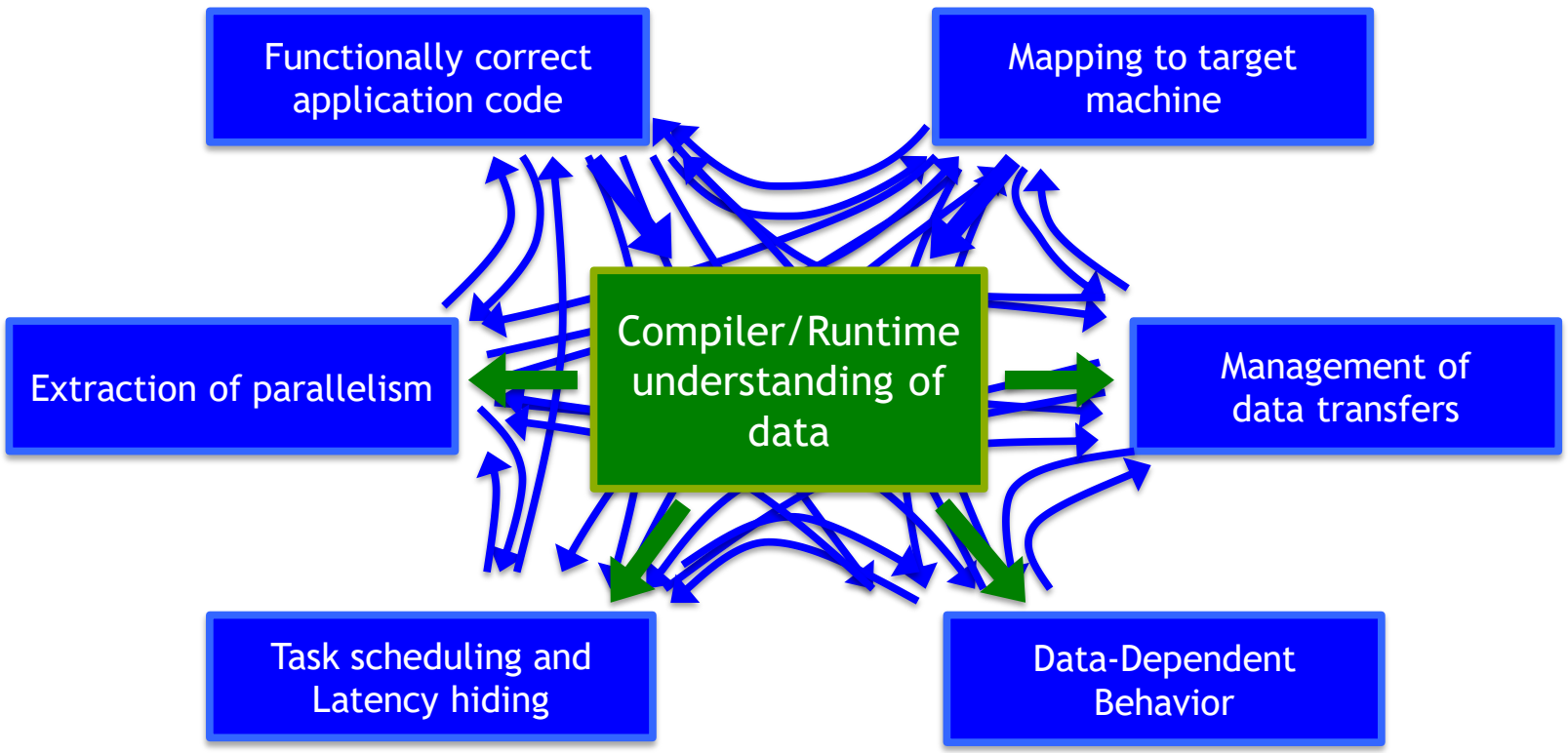
LEGION FROM FIRST PRINCIPLES

Parallel Programming is Hard!



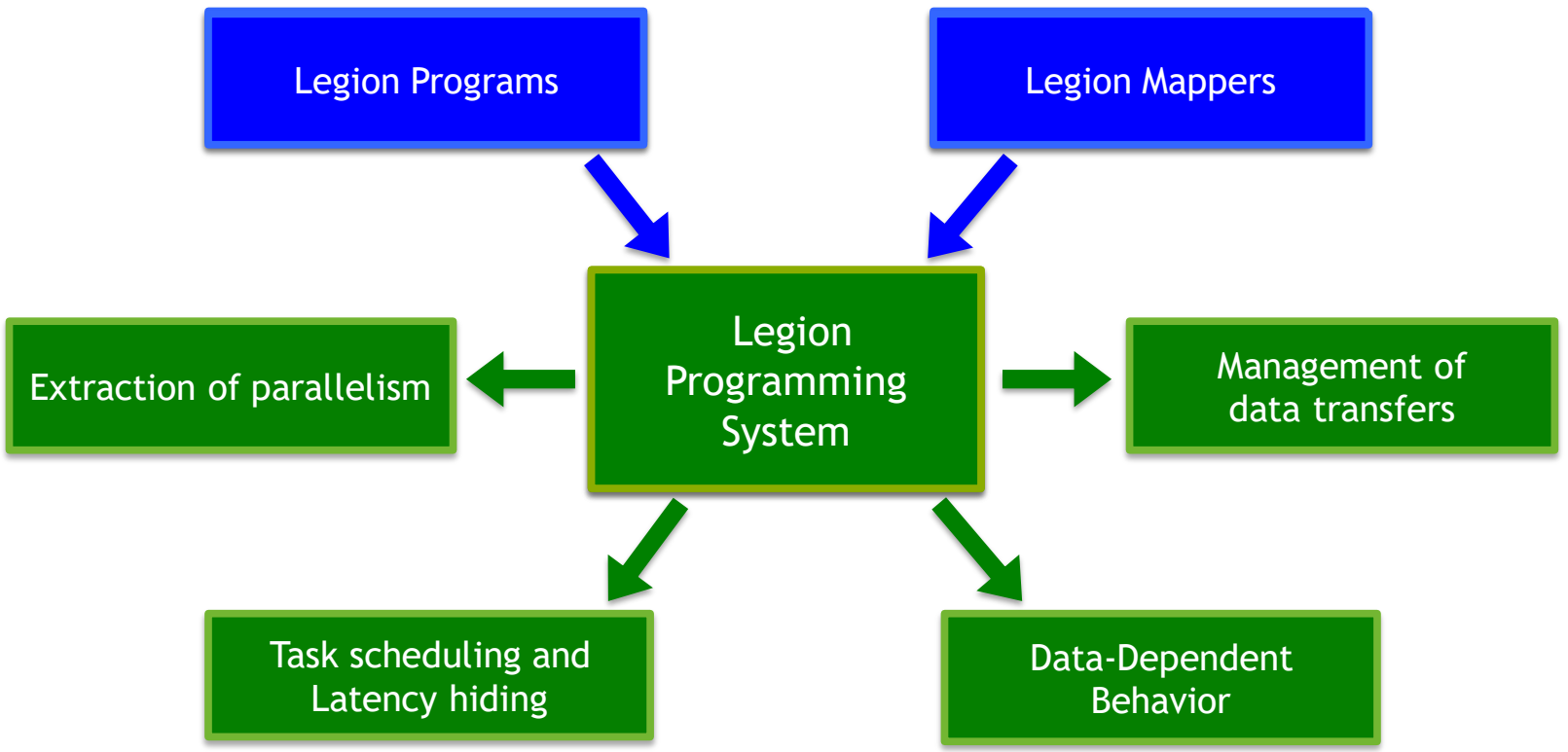
LEGION FROM FIRST PRINCIPLES

Understanding of program data makes all the difference



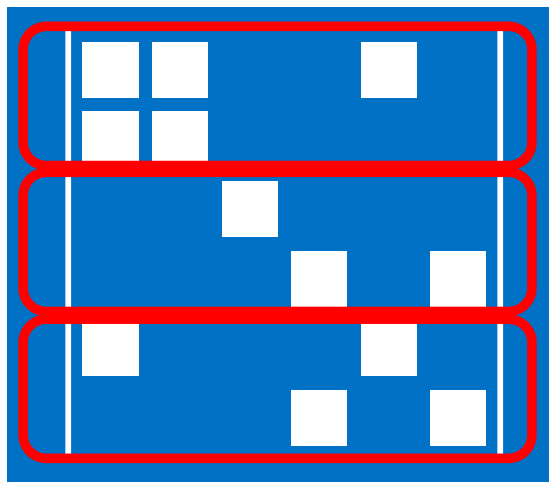
LEGION FROM FIRST PRINCIPLES

Separating correctness specification from performance



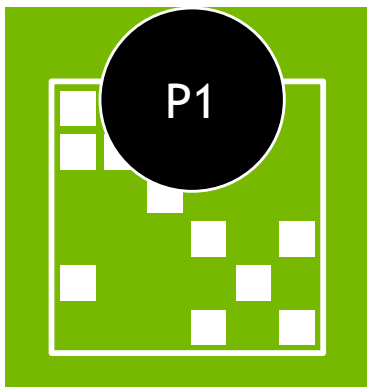
PARTITIONING

Legion based

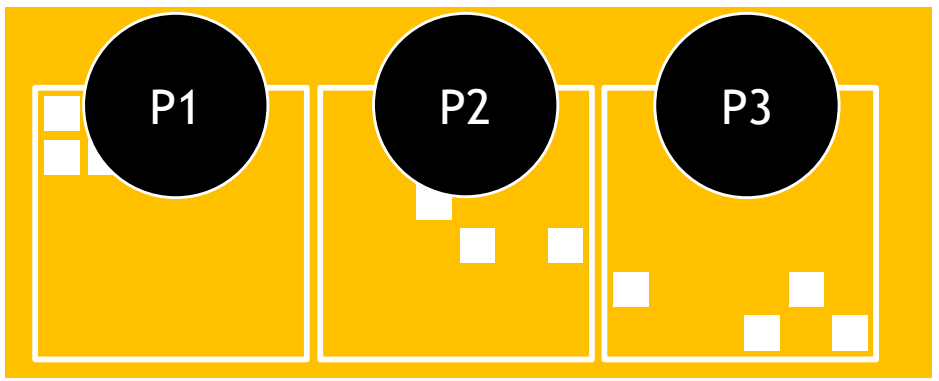


```
array1d<V> v(rt,ctx,10);
DomainColoring coloring;
…
v.partition(coloring);
```

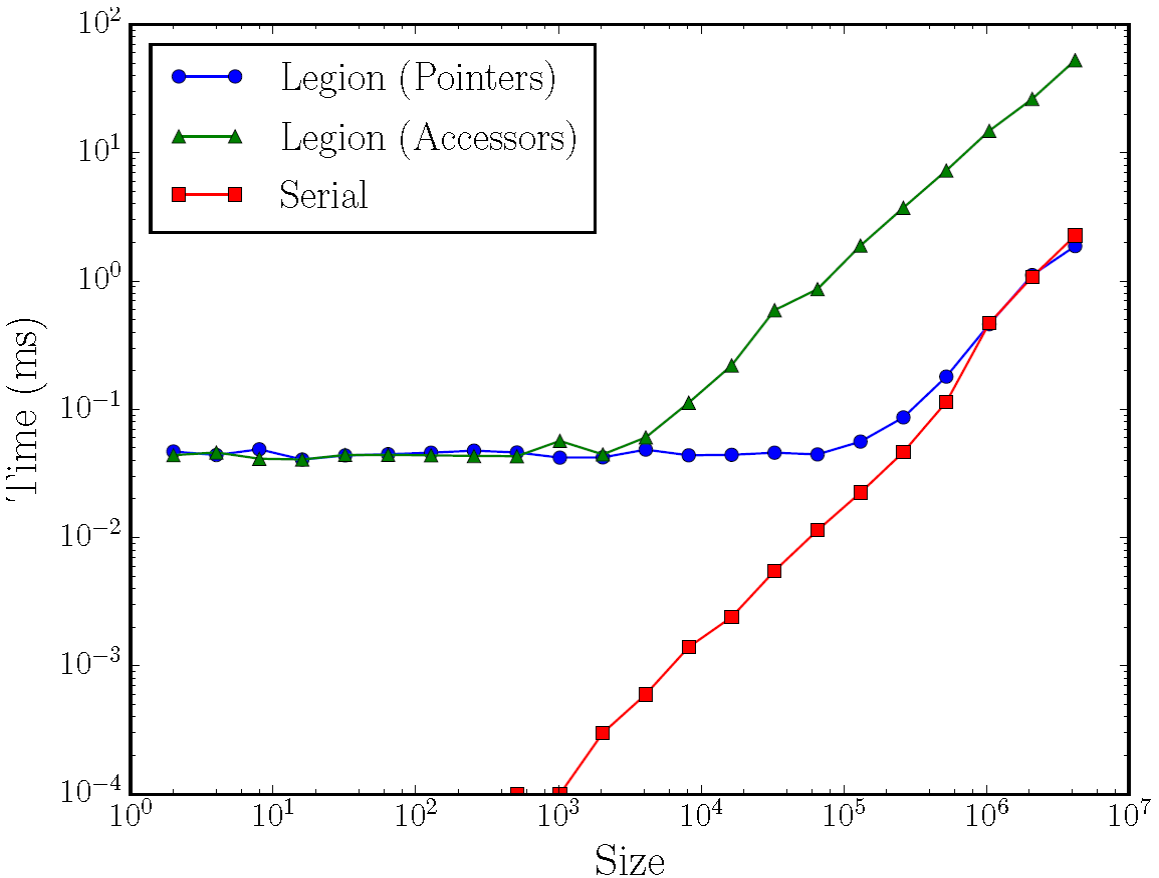
TaskLauncher



IndexLauncher



RUNTIME OVERHEAD



CONTAINERS

All array1d object must be initialized with a Legion context and runtime

```
template<typename T>
array1d(Context* ctx, Runtime* runtime, size_t num_entries,
FieldID field_id)
```

Important: Initial array1d container abstract a single logical region with a single field. Implications? Limits ability of custom mappers

FUNCTIONS

Sparse Matrix Datatypes

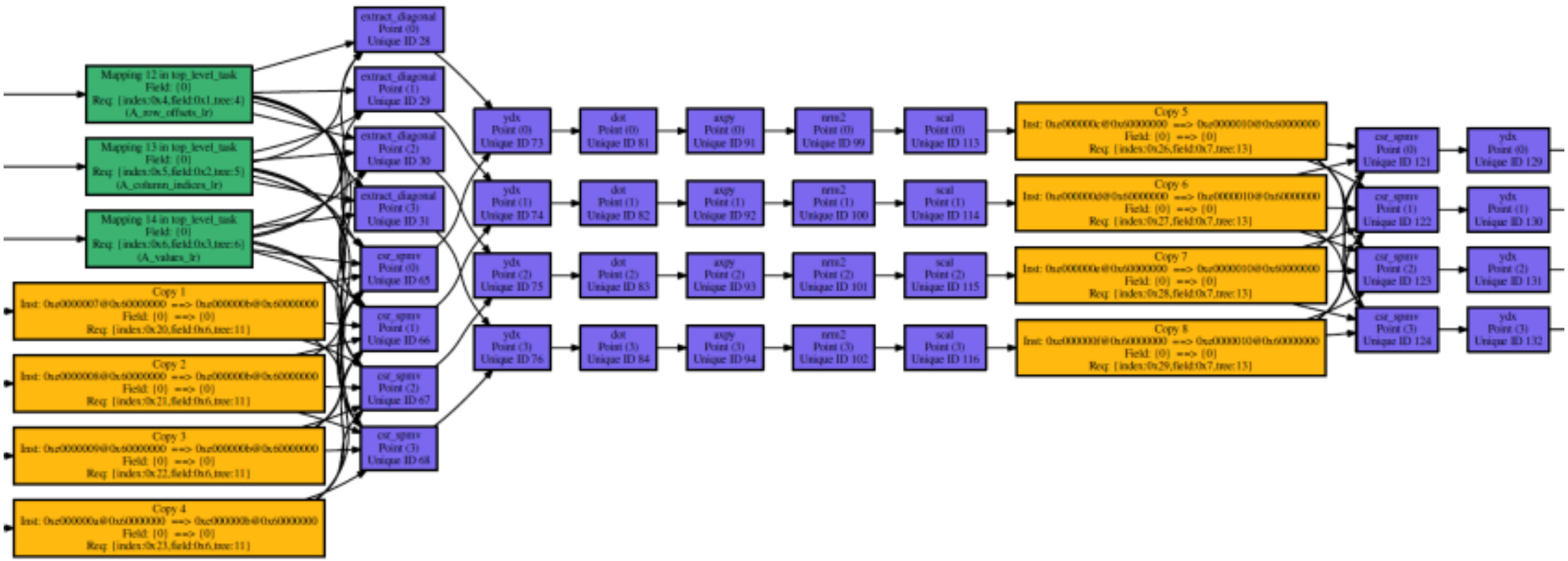
```
template<typename I, typename V>
FutureMap multiply(csr_matrix<I,V>& A, array1d<V>& x, array1d<V>&
y, const Predicate &pred = Predicate::TRUE_PRED)
```

Support speculative execution of routines inside of conditional loops

All functions return a Future or FutureMap (Avoid waiting like the _____.)

ANALYSIS

Task Graphs

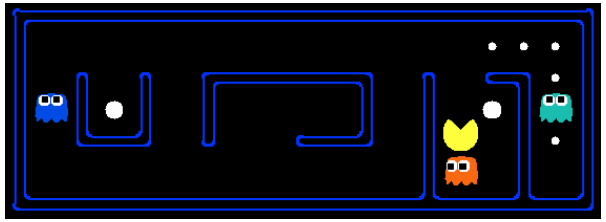


2.0

- What is this?

- Draw a face.

- Beat this game.



- AI?
- Deep Learning?

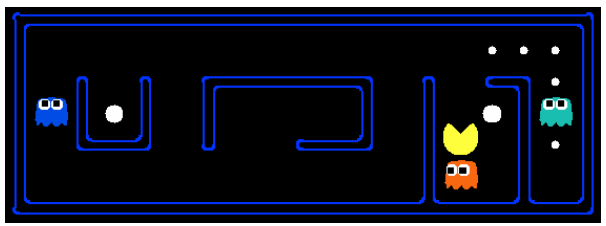
2.0

- Differentiable programming?

- What is this?

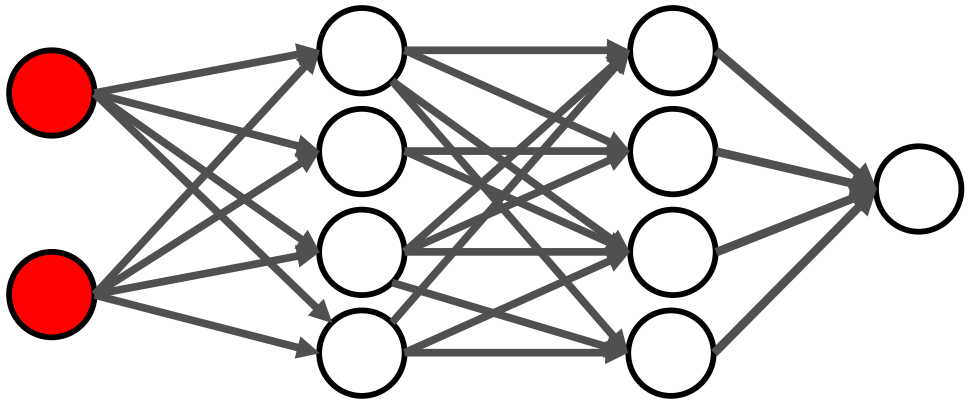
- Draw a face.

- Beat this game.



NETWORKS

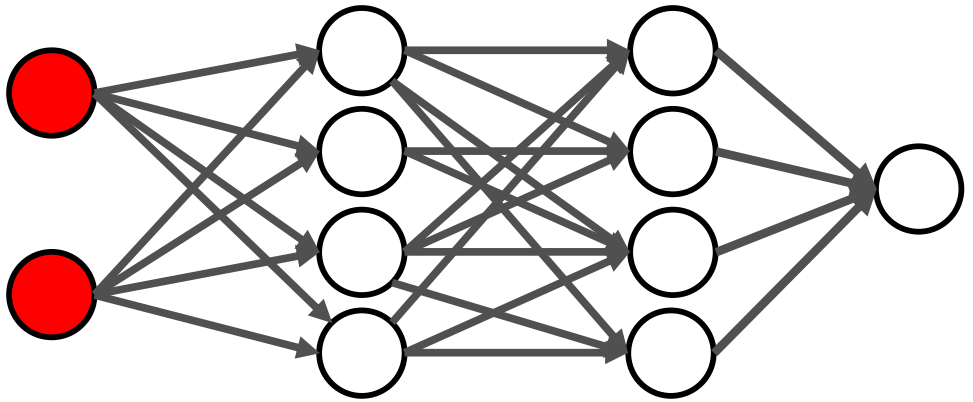
DNNs



CAT

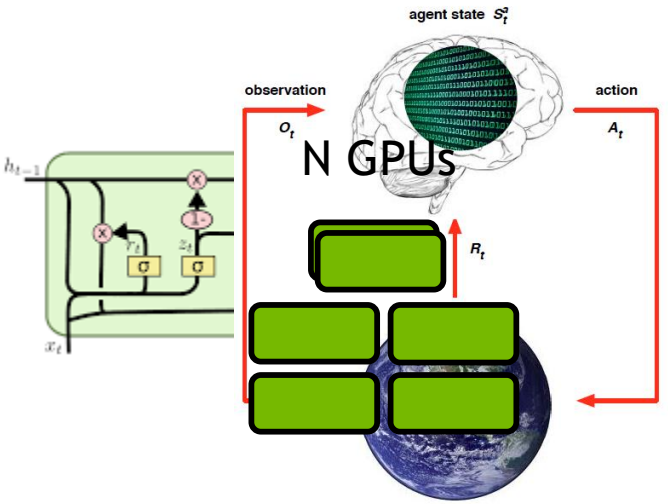
NETWORKS

Improvement

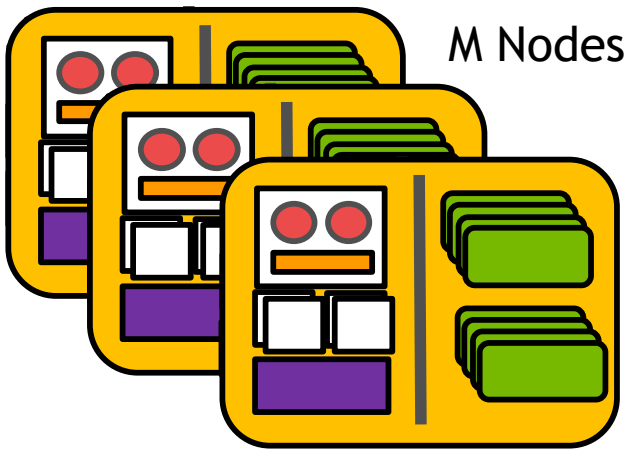


"KITTEN RAISING PAW"
CAT
UP ARROW

GPU



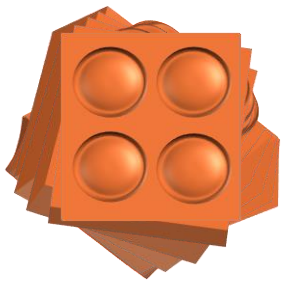
N GPUs

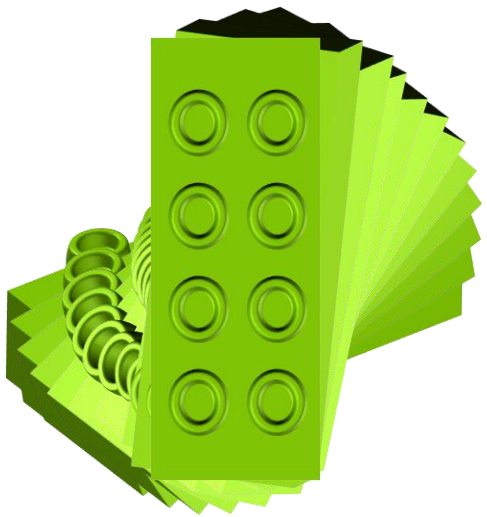


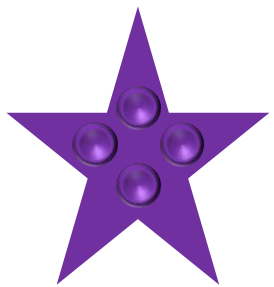
M Nodes

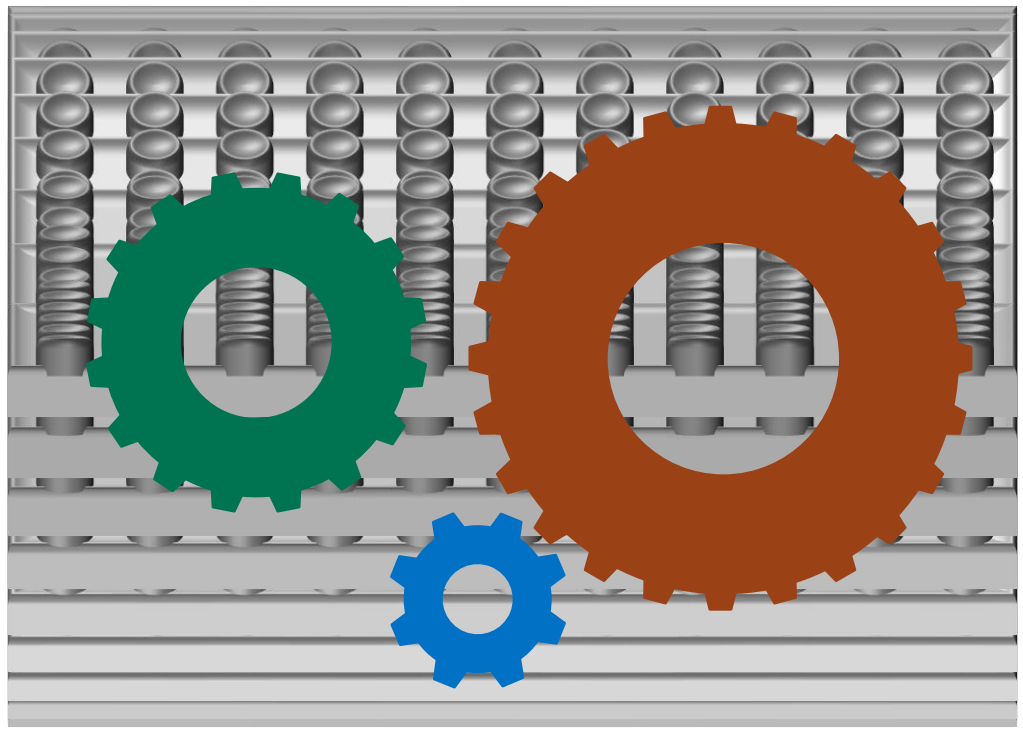
COMPONENTS

Frameworks



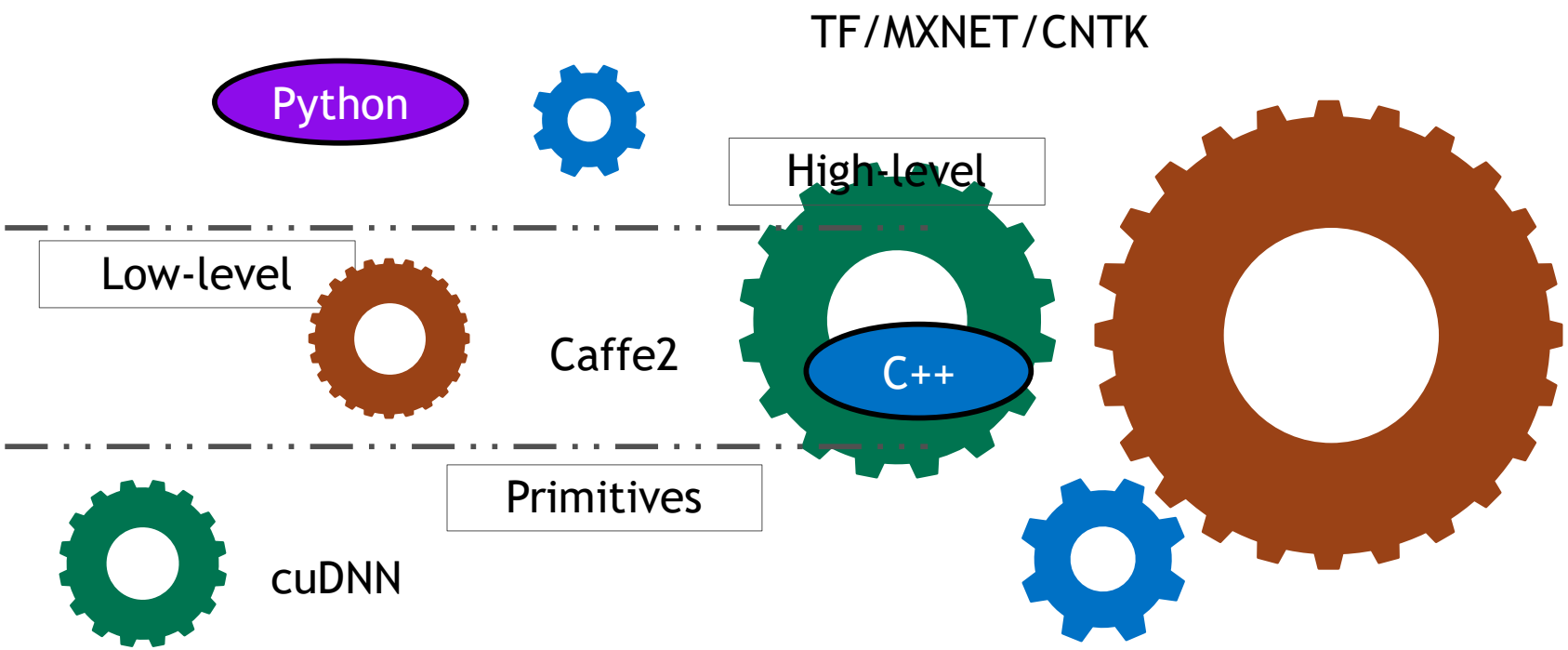






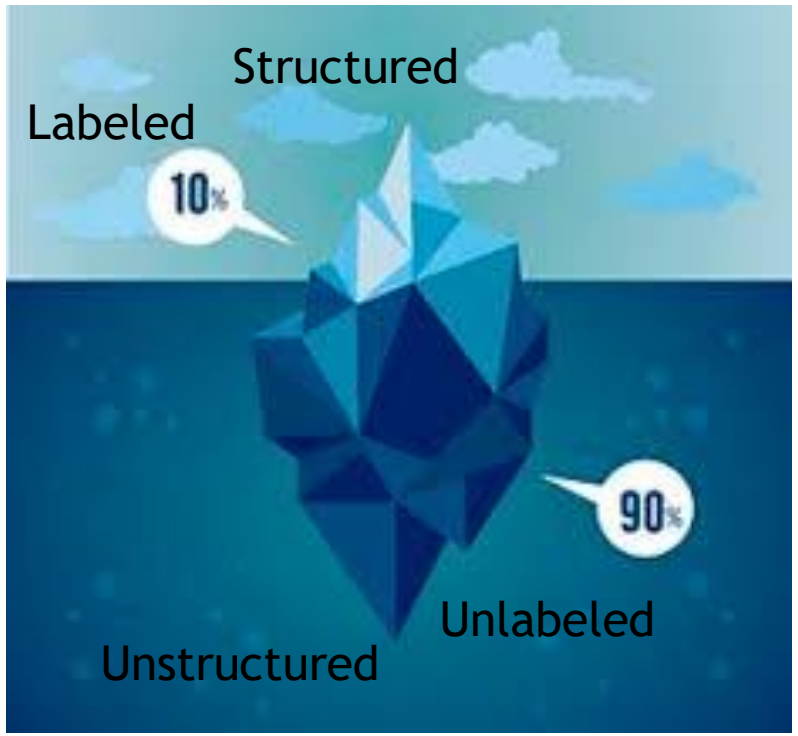
DEPENDENCIES

Frameworks

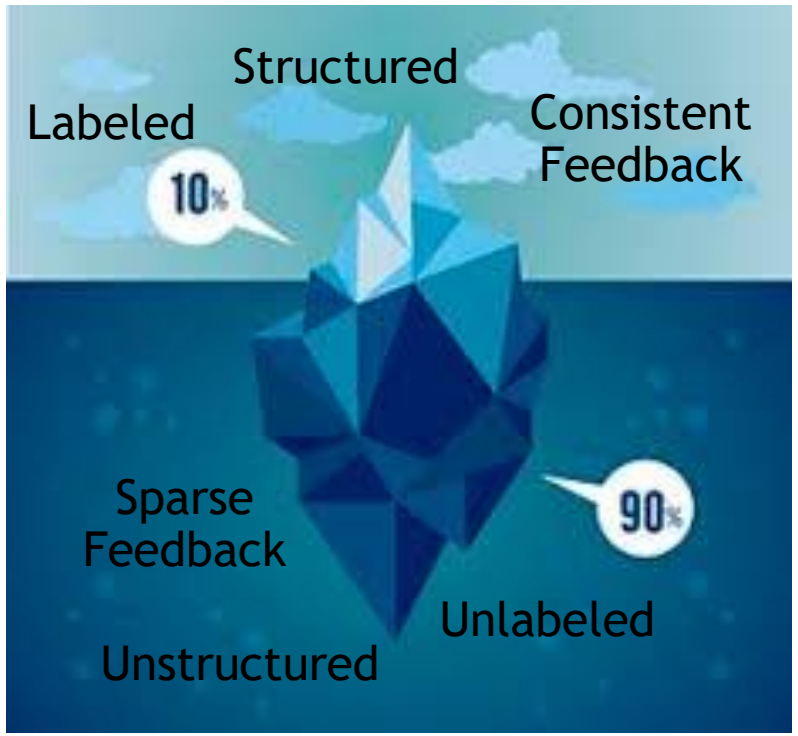


DATA

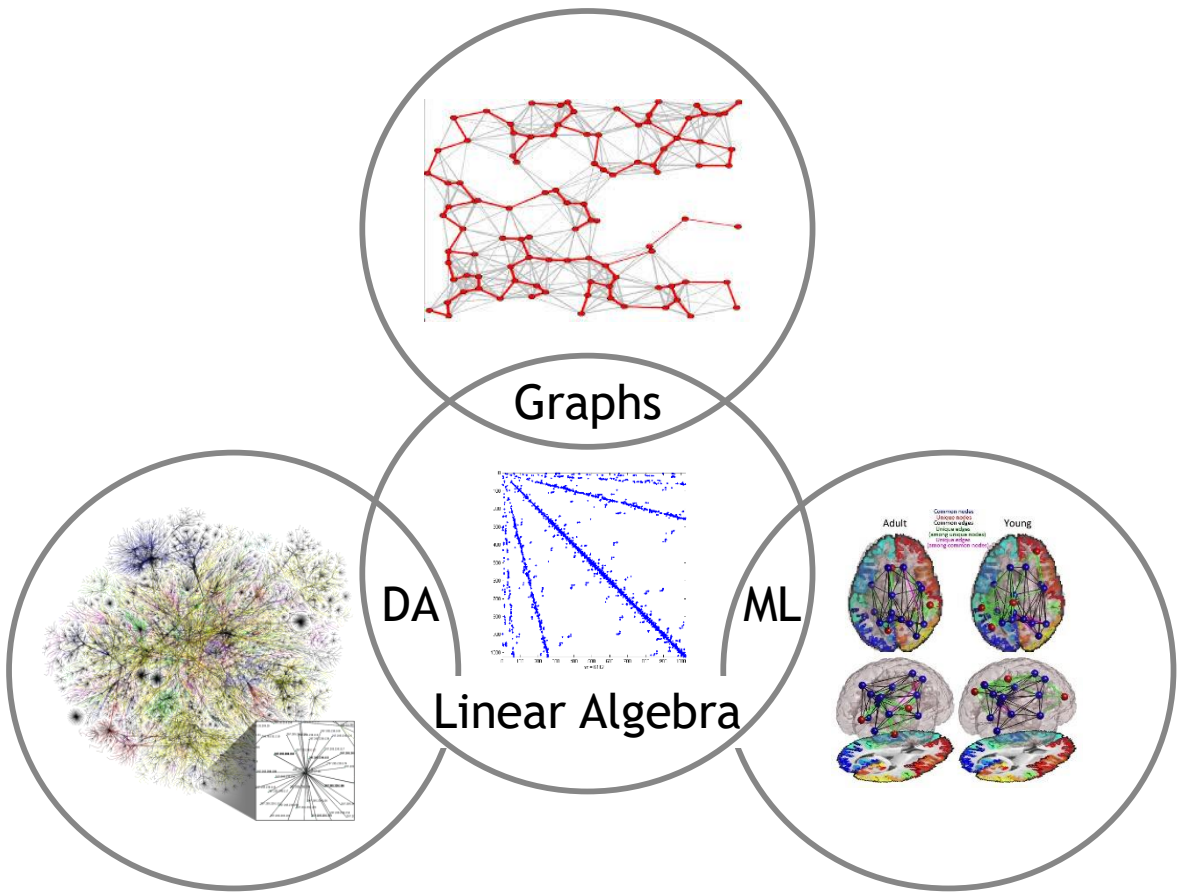
DATA



DATA


Structured
Labeled
Consistent
Feedback
10%
Sparse
Feedback
90%
Unlabeled
Unstructured

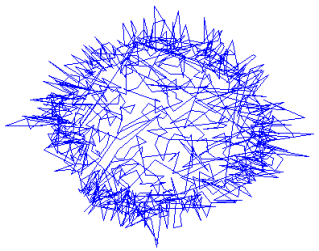
UNSTRUCTURED DATA

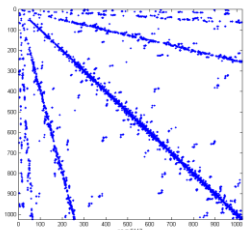


GRAPH ANALYSIS

- Global Partitioning/Clustering Scheme

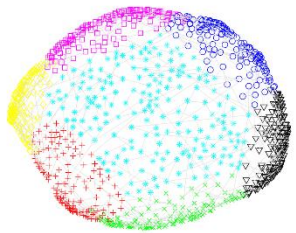
Graph → Laplacian → Eigensolver → Clustering → Partitions



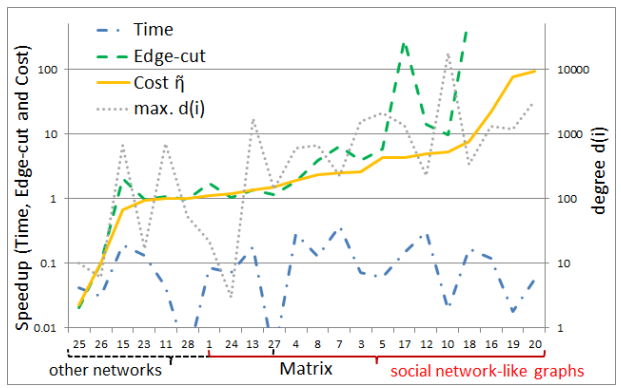


$Ax = \lambda x$
(real values)

k-means
(discrete values)



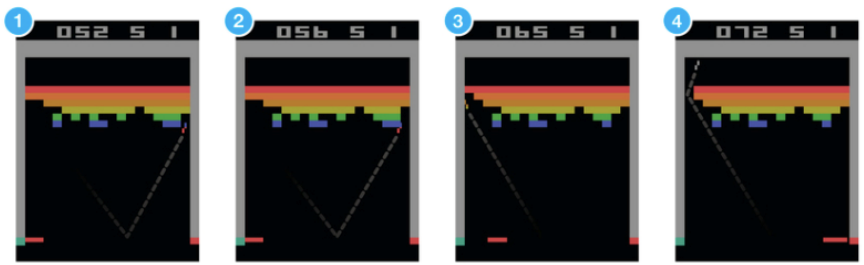
- There are tradeoffs between quality and speed
- Byproducts can be used to visualize graphs
- Important in multi-GPU settings

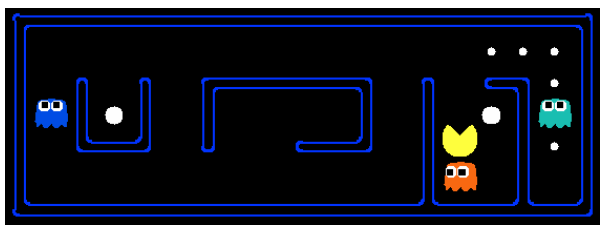


REINFORCEMENT LEARNING

Applications

- Space management
- ***Video games***
- Financial investments
- ML training – learning-to-learn





BATTLEFIELD

RL Performance Libraries

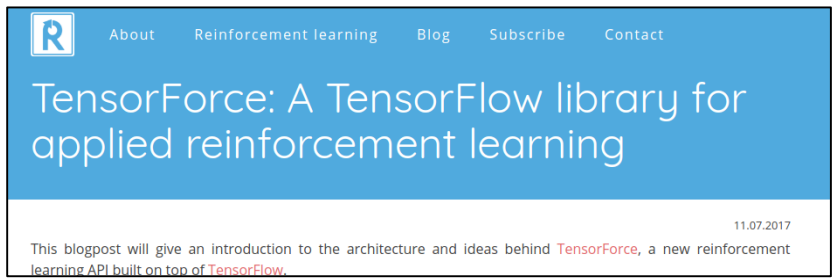
TensorFlow Agents

This project provides optimized infrastructure for reinforcement learning. It extends the OpenAI gym interface to multiple parallel environments and allows agents to be implemented in TensorFlow and perform batched computation. As a starting point, we provide BatchPPO, an optimized implementation of Proximal Policy Optimization.

Please cite the TensorFlow Agents paper if you use code from this project in your research:

```
@article{hafner2017agents,
  title={TensorFlow Agents: Efficient Batched Reinforcement Learning in TensorFlow},
  author={Hafner, Danijar and Davidson, James and Vanhoucke, Vincent},
  journal={arXiv preprint arXiv:1709.02878},
  year={2017}
}
```

Dependencies: Python 2/3, TensorFlow 1.3+, Gym, rumamel.yaml



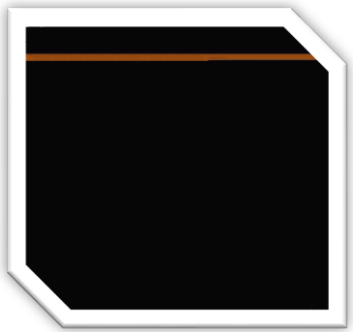
Caffe2 Reinforcement Learning Models

Reinforcement Learning (RL) is an area of machine learning focused on agents maximizing a total reward after a duration in an environment. The agent is often some robot or game avatar, but it can also be a recommender system, a notification bot, and a variety of other avatars that make decisions. The reward can be points in a game, or more engaging content on a website. Facebook uses reinforcement learning to power several efforts in the company. Sharing an open-source fork of our caffe2 RL framework allows us to give back to the open source community and also collaborate with other institutions as RL finds more applications in industry.

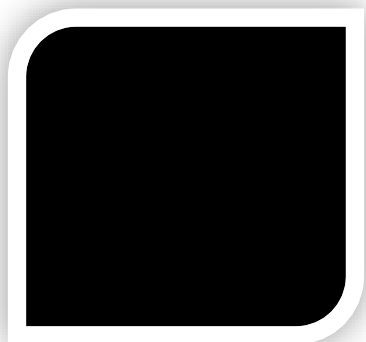
This project, called RL_Caffe2, contains several RL implementations built on caffe2 and running inside OpenAI Gym.

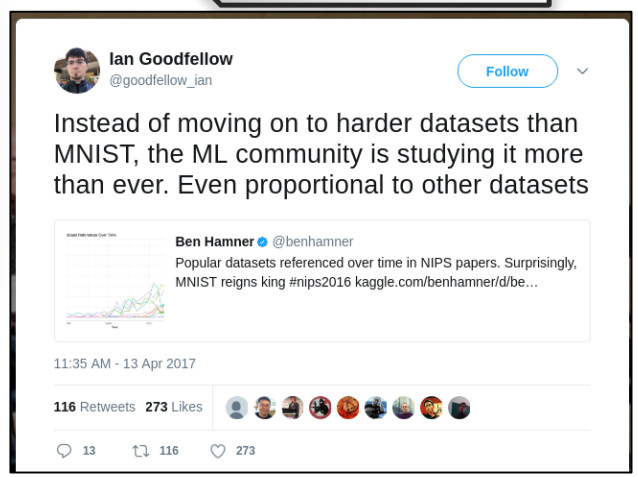
ATARI

ALE (Atari Learning Environment)

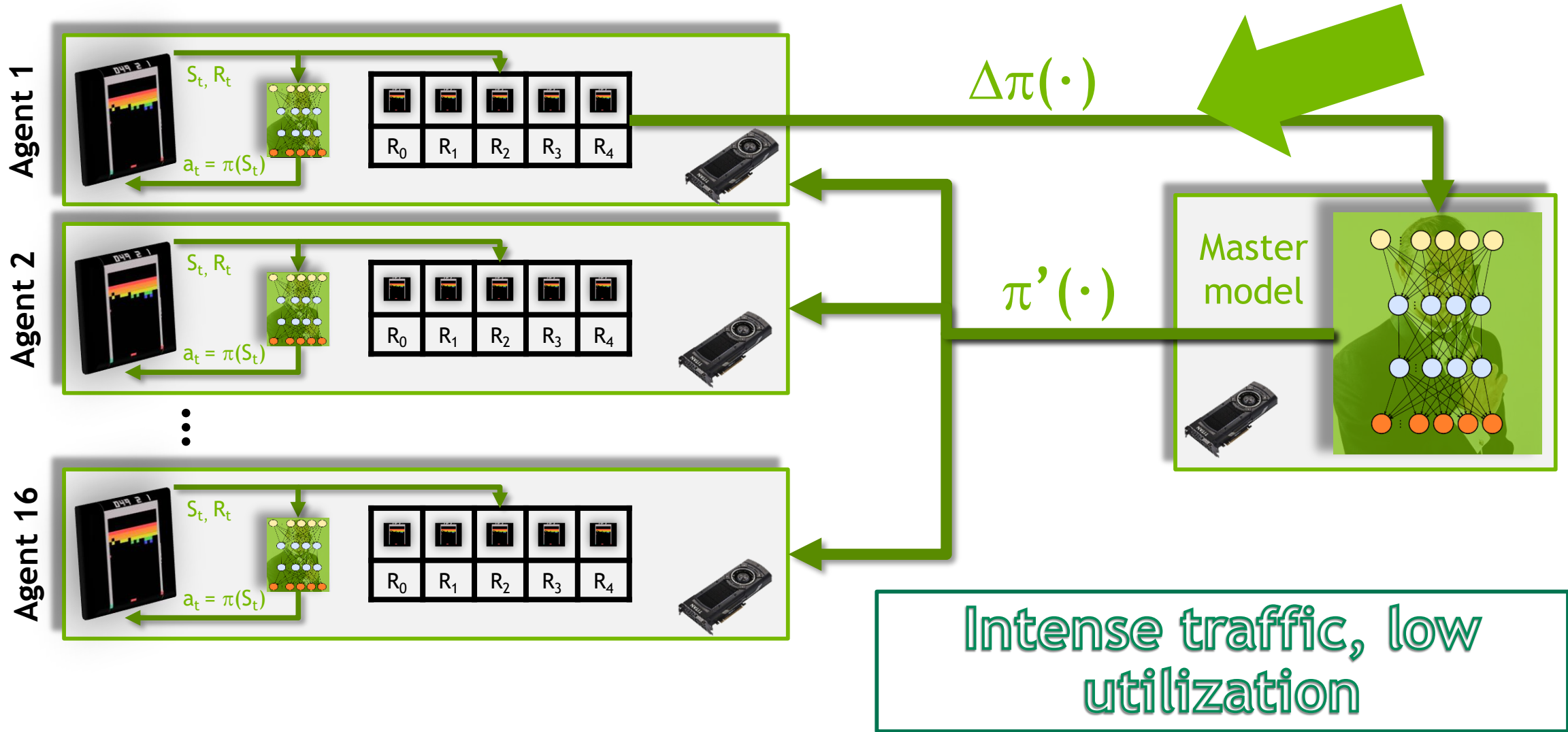


- Diverse set of tasks
- Established benchmark – MNIST of RL?



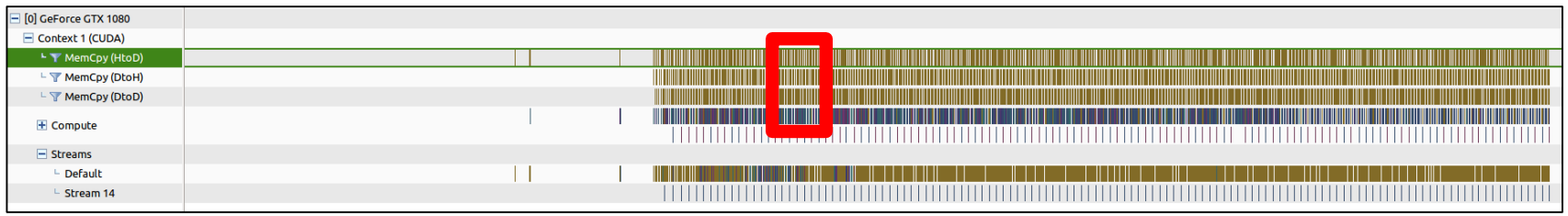


A3C


Agent 1
Agent 2
Agent 16
S_t, R_t
$a_t = \pi(S_t)$
R_0 R_1 R_2 R_3 R_4
$\Delta\pi(\cdot)$
$\pi'(\cdot)$
Master model
Intense traffic, low utilization

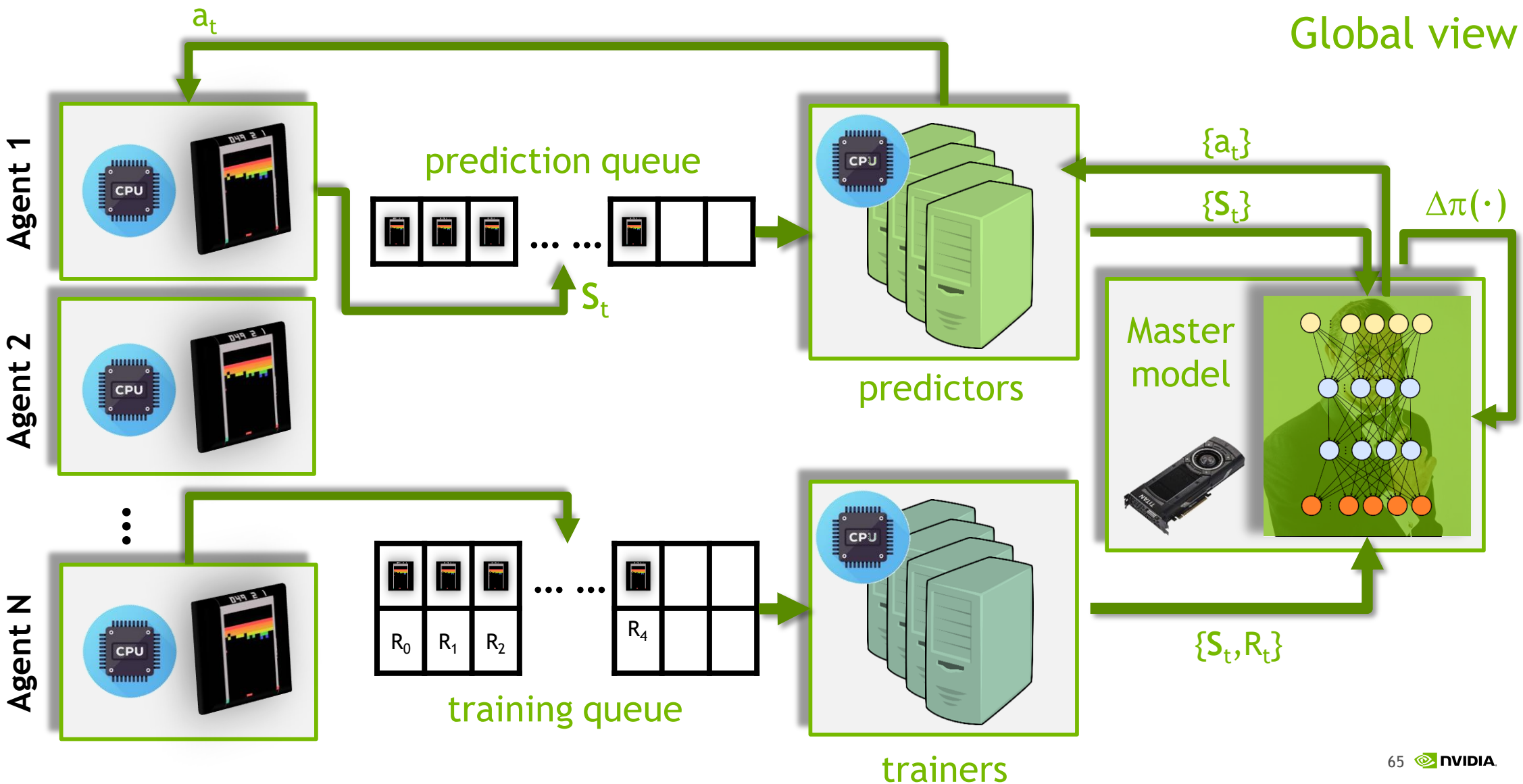
NVVP

Bubbles

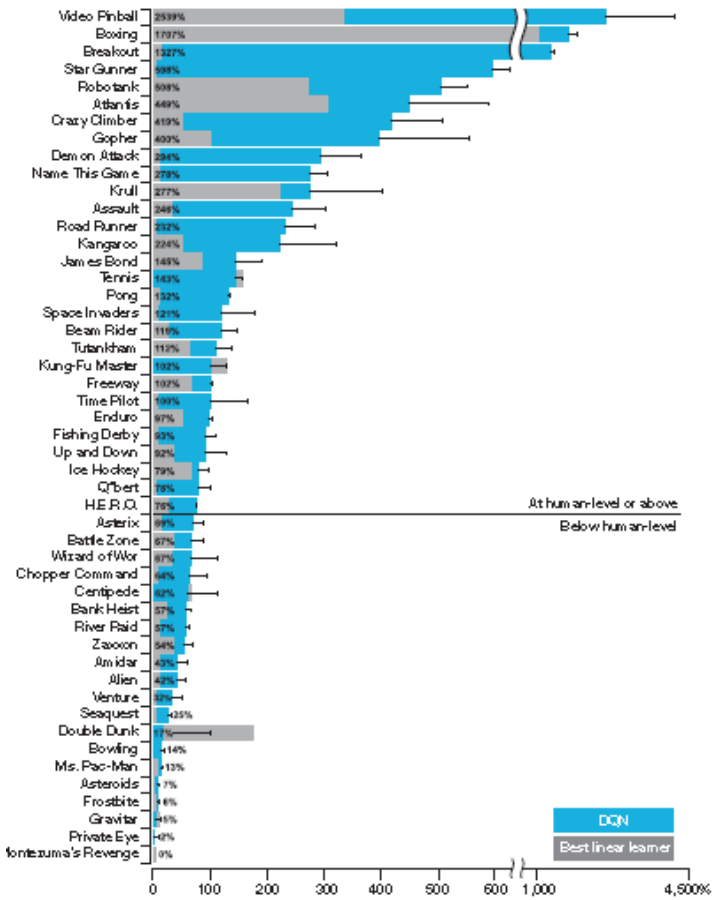


GA3C

Global view



REINFORCEMENT LEARNING



Minh et al. [Human Level Control Through Deep Reinforcement Learning, 2015]

BATTLEFIELD

RL Performance Libraries

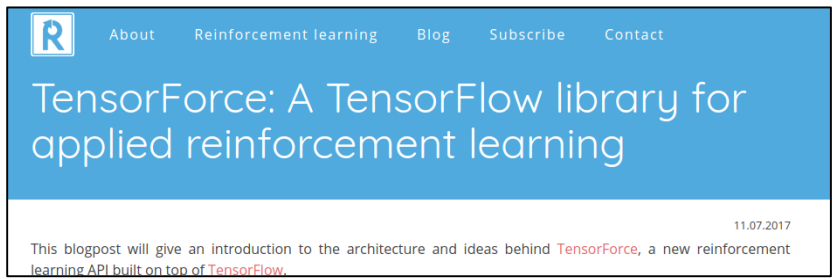
TensorFlow Agents

This project provides optimized infrastructure for reinforcement learning. It extends the OpenAI gym interface to multiple parallel environments and allows agents to be implemented in TensorFlow and perform batched computation. As a starting point, we provide BatchPPO, an optimized implementation of Proximal Policy Optimization.

Please cite the TensorFlow Agents paper if you use code from this project in your research:

```
@article{hafner2017agents,
  title={TensorFlow Agents: Efficient Batched Reinforcement Learning in TensorFlow},
  author={Hafner, Danijar and Davidson, James and Vanhoucke, Vincent},
  journal={arXiv preprint arXiv:1709.02878},
  year={2017}
}
```

Dependencies: Python 2/3, TensorFlow 1.3+, Gym, rumamel.yaml



Caffe2 Reinforcement Learning Models

Reinforcement Learning (RL) is an area of machine learning focused on agents maximizing a total reward after a duration in an environment. The agent is often some robot or game avatar, but it can also be a recommender system, a notification bot, and a variety of other avatars that make decisions. The reward can be points in a game, or more engaging content on a website. Facebook uses reinforcement learning to power several efforts in the company. Sharing an open-source fork of our caffe2 RL framework allows us to give back to the open source community and also collaborate with other institutions as RL finds more applications in industry.

This project, called RL_Caffe2, contains several RL implementations built on caffe2 and running inside OpenAI Gym.

CONCLUSION

Future Directions

- Short - Striving to enhance current programming models with specialized libraries
- Medium - Moving toward assisted execution using runtime systems
- Long – Integrate deep learning processes to automatically discover optimizations for program execution