

Modern AI in manufacturing

Kris Bhaskar
Sep 2021

IITM HPC workshop



Agenda



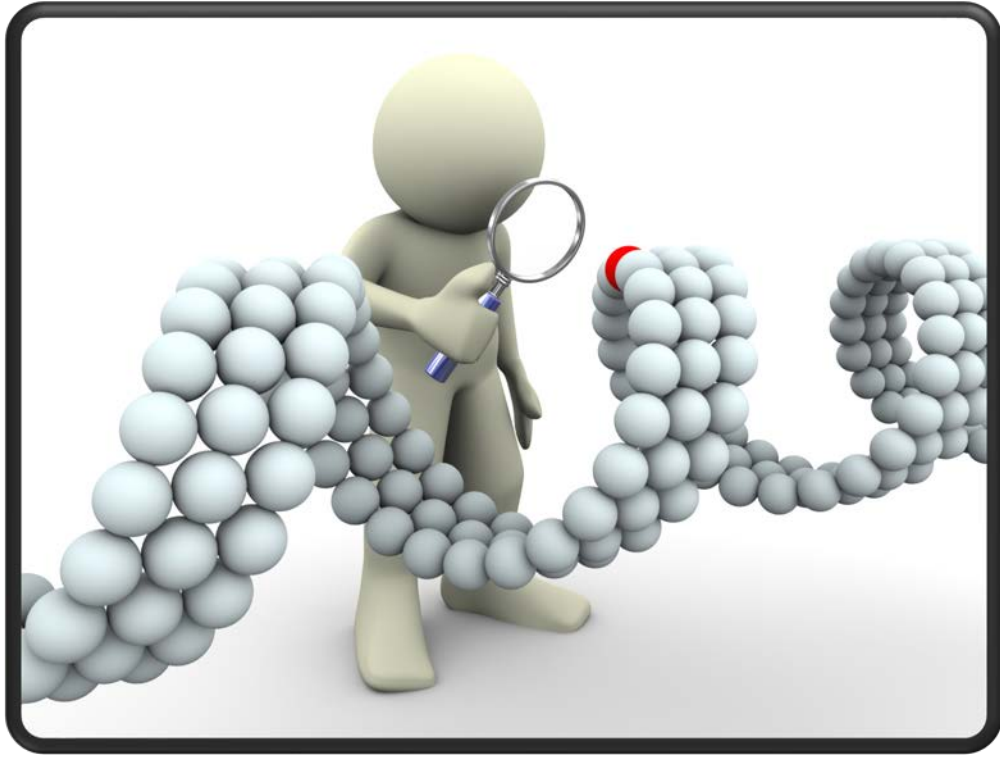
- KLA Process Control Background
- KLA AI systems : Recent success stories
- Lessons Learned & Challenges
- Looking Ahead

KLA process control background



Process Control

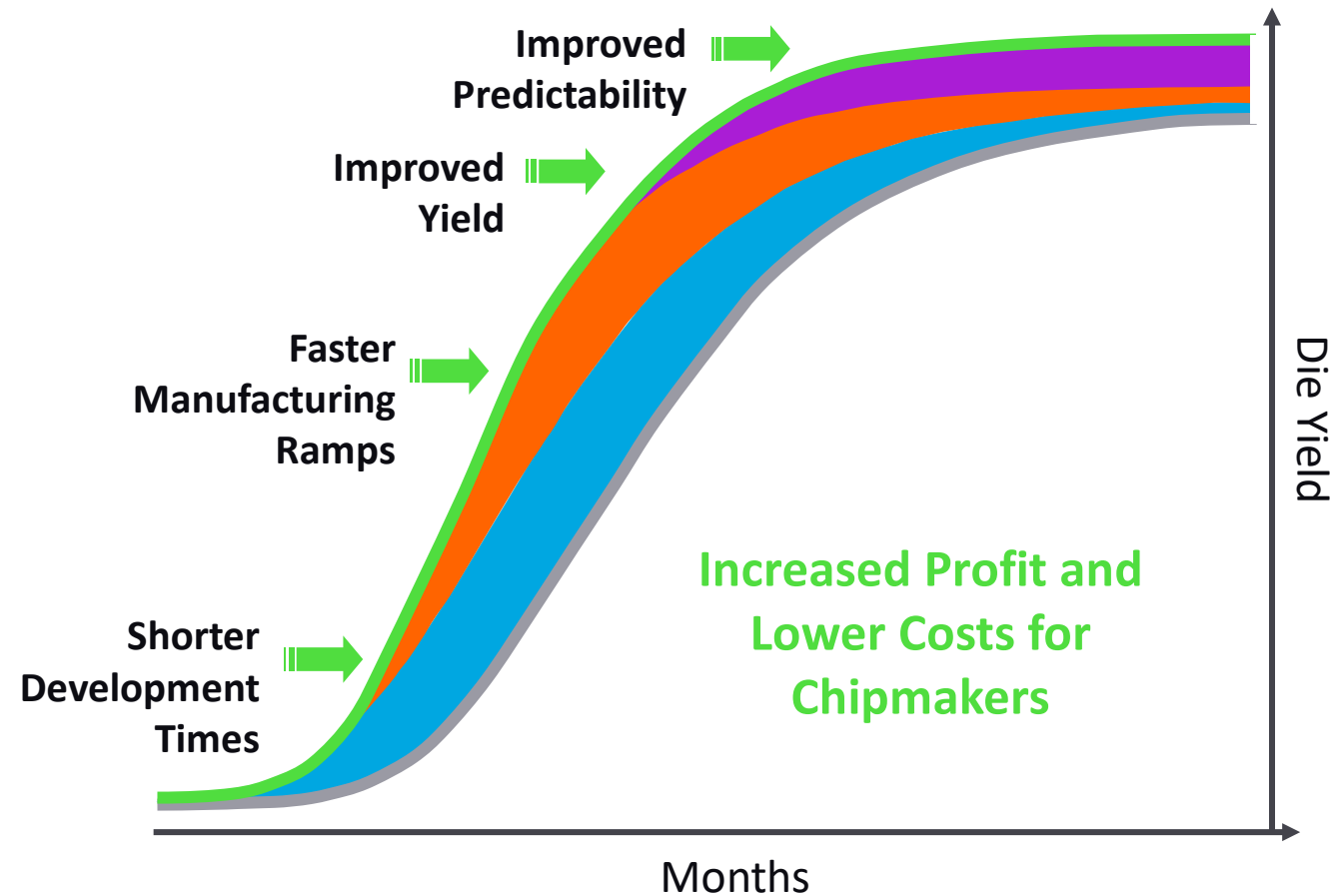
The act of maintaining products and equipment within established specifications during manufacturing operations



... Critical for IC Manufacturing success



- Ramp Yields Faster
- Provide More Predictable Delivery

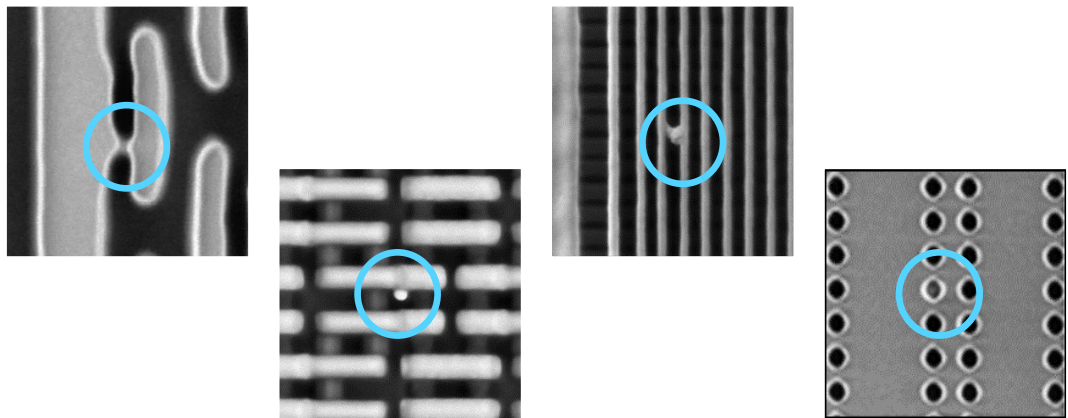


Leveraging ML since the '90s



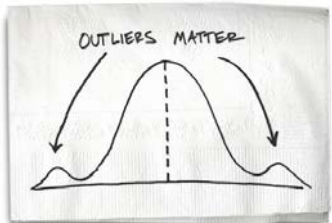
Inspection

Find Critical Defects



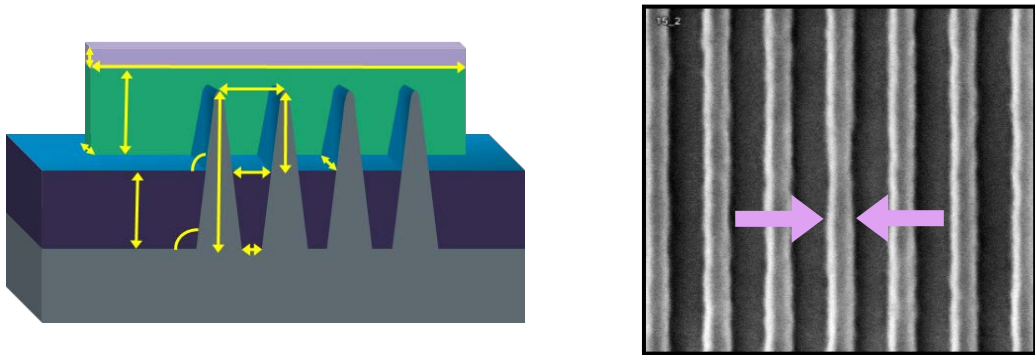
Statistically: Outlier detection

1995: First ever Classification system (KLA 2135)
2018: First ever Physics based DL system (KLA eSL10™)



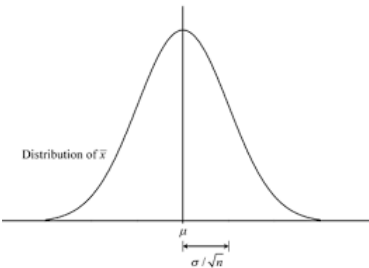
Metrology

Measure Critical Parameters

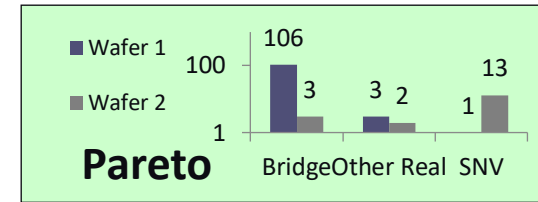


Statistically: Measure $\mathcal{N}(\mu, \sigma^2)$

1993: First ever NN based Metrology system (KLA Films)
2017+: Models enhanced by DL



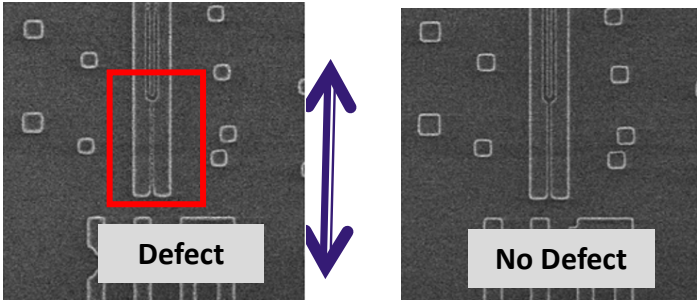
KLA inspector technologies pushing on many fronts



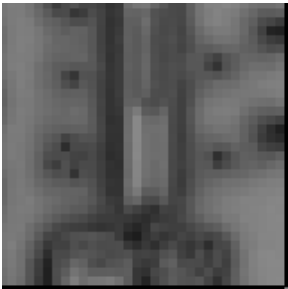
- 1. Optics , SEM
- 2. High Speed Sensors
- 3. Computing Stack



SEM Verification



Optical Detection



CPU
Structured Data
GPU
Streaming

GPU & CPU-SIMD
Streaming

FPGA
Near Real Time

Optics



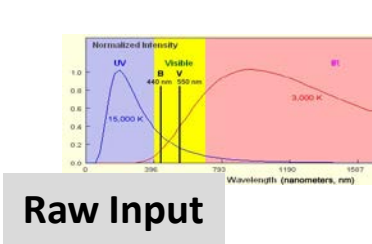
Sensors



Image / Data Processing

IMC Stack

ASIC/FPGA
Real Time



KIA-Tencor Confidential

KLA AI Systems : Recent success stories



Two recently released KLA DL products

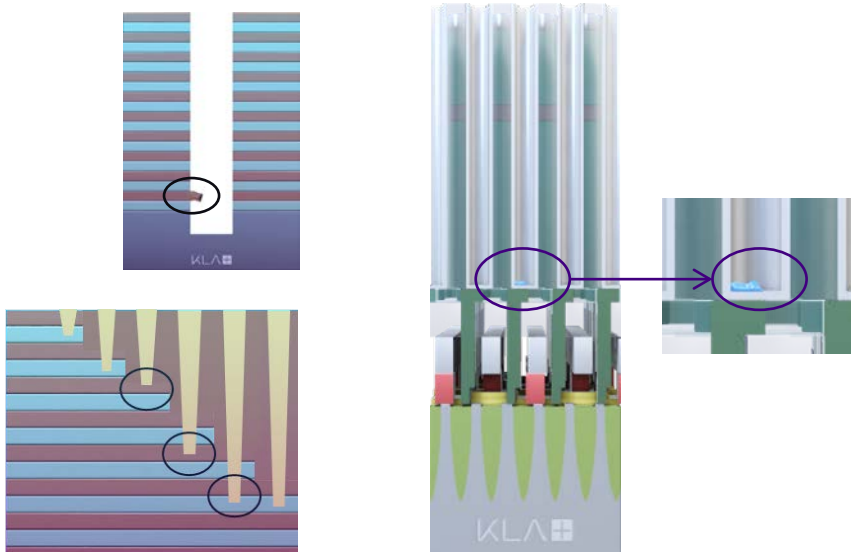


- eSL10™ : Revolutionary e-beam inspector < semi front end >
- Kronos™ 1190 w DefectWise® : Industry's first DL based classification system < semi back end packaging >

Advanced IC defect challenges



high aspect ratio structures



defect size at 7/5/3nm
design nodes

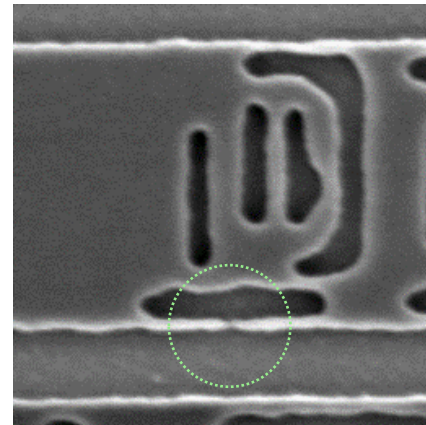


image: KLA

R&D: high defectivity and pattern noise
EUV pattern variation

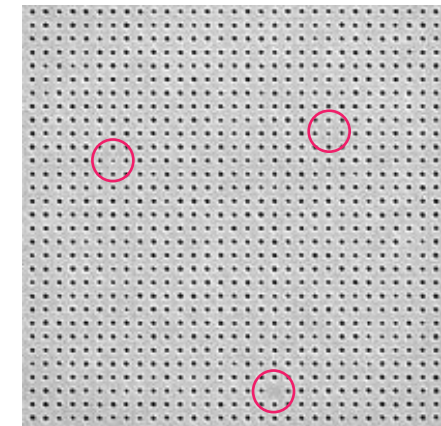
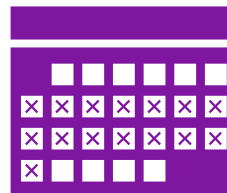


image: Litho Workshop 2019, imec and KLA

time to results

electrical test
FIB/TEM



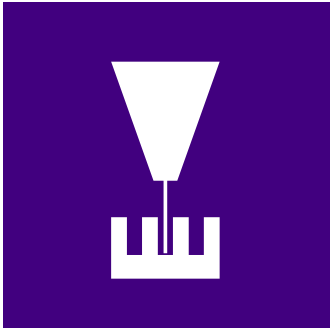
= \$\$

Need:
actionable inspection results...quickly!

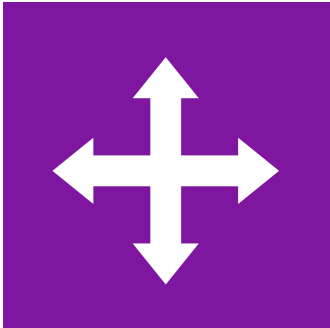
eSL10 technologies address advanced IC defect challenges



High Beam
Current Density



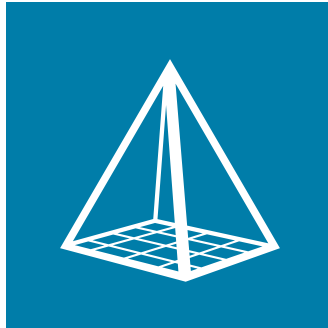
Wide Optics Range



Simul-6™



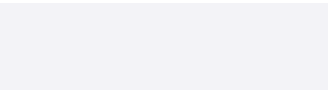
Yellowstone™



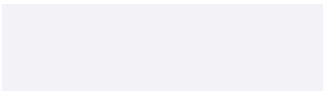
Artificial
Intelligence



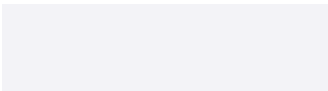
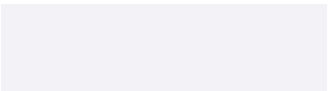
high aspect ratio structures



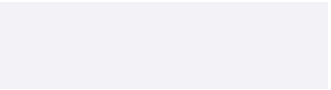
defect size at 7/5/3nm nodes



EUV pattern variation



R&D: high defectivity, noise



time to results



eSL10™ Artificial Intelligence



industry-first
deep learning system
isolates key DOI from other
defects and pattern noise

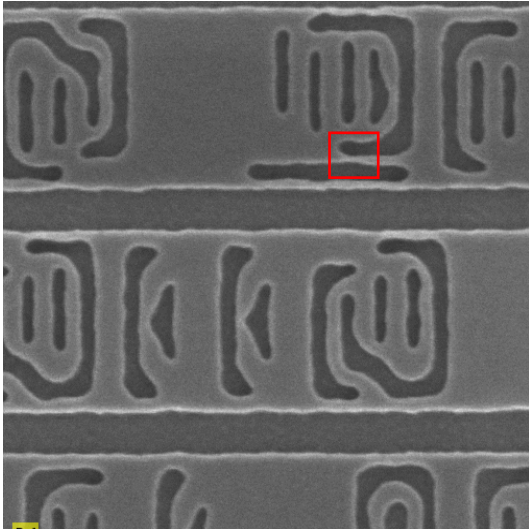
uses state of art NVIDIA GPUs



eSL10™: AI algorithms vs traditional

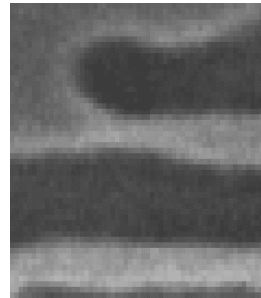


Defective Images



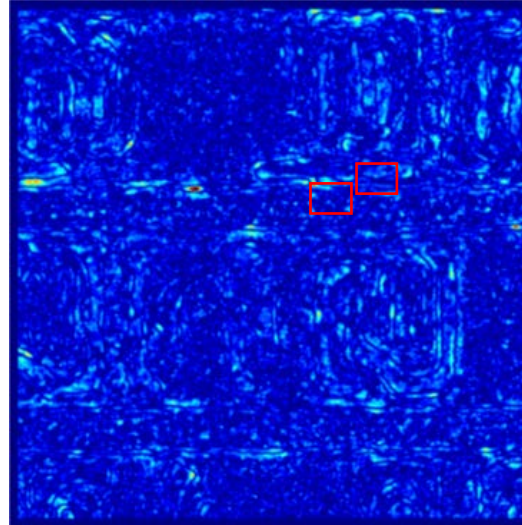
A – B blinking

Defective Images
zoomed

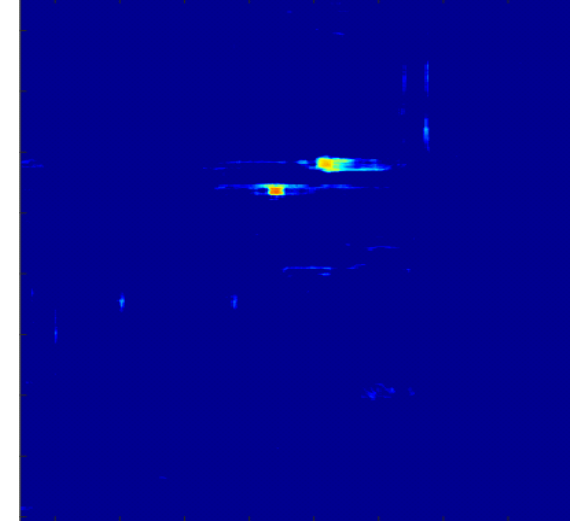


A – B blinking

Traditional Algorithms

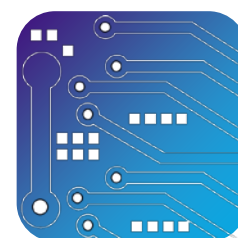


AI Algorithms



Kronos™ 1190

Wafer-Level Packaging Defect Inspection



High Resolution



Runtime AI



Accurate
Inspection Area

The Kronos™ 1190 provides sensitive defect inspection, helping chip manufacturers to quickly detect, resolve and monitor excursions to provide greater control of quality during wafer-level packaging

Kronos™ 1190

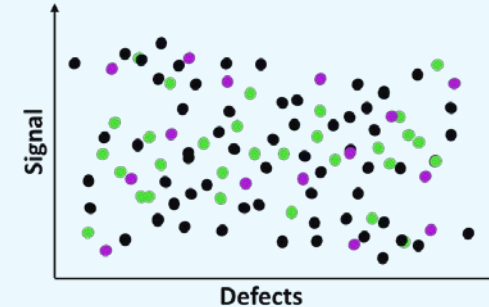
Wafer-Level Packaging Defect Inspection



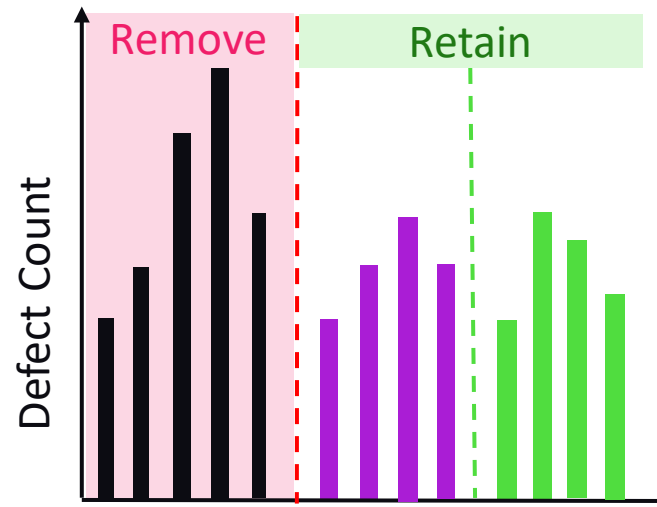
DefectWise® integrates AI as a system level solution to boost productivity

- Throughput
- Sensitivity
- Classification accuracy
- Defect discovery
- Ease of use

How can we separate key defects from nuisance?



- Key Defect 1
- Key Defect 2
- Nuisance Defect



Retain and sort key defects
Remove nuisance

Lessons Learned & Challenges



Key observations

Lessons Learned

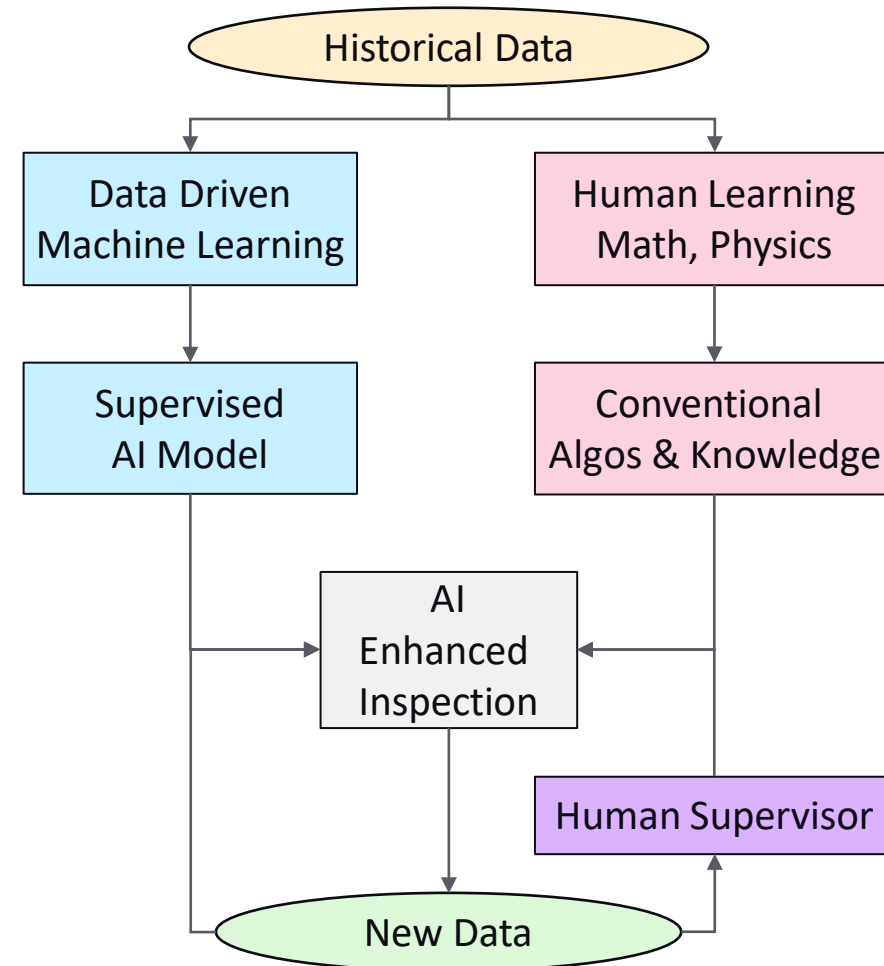
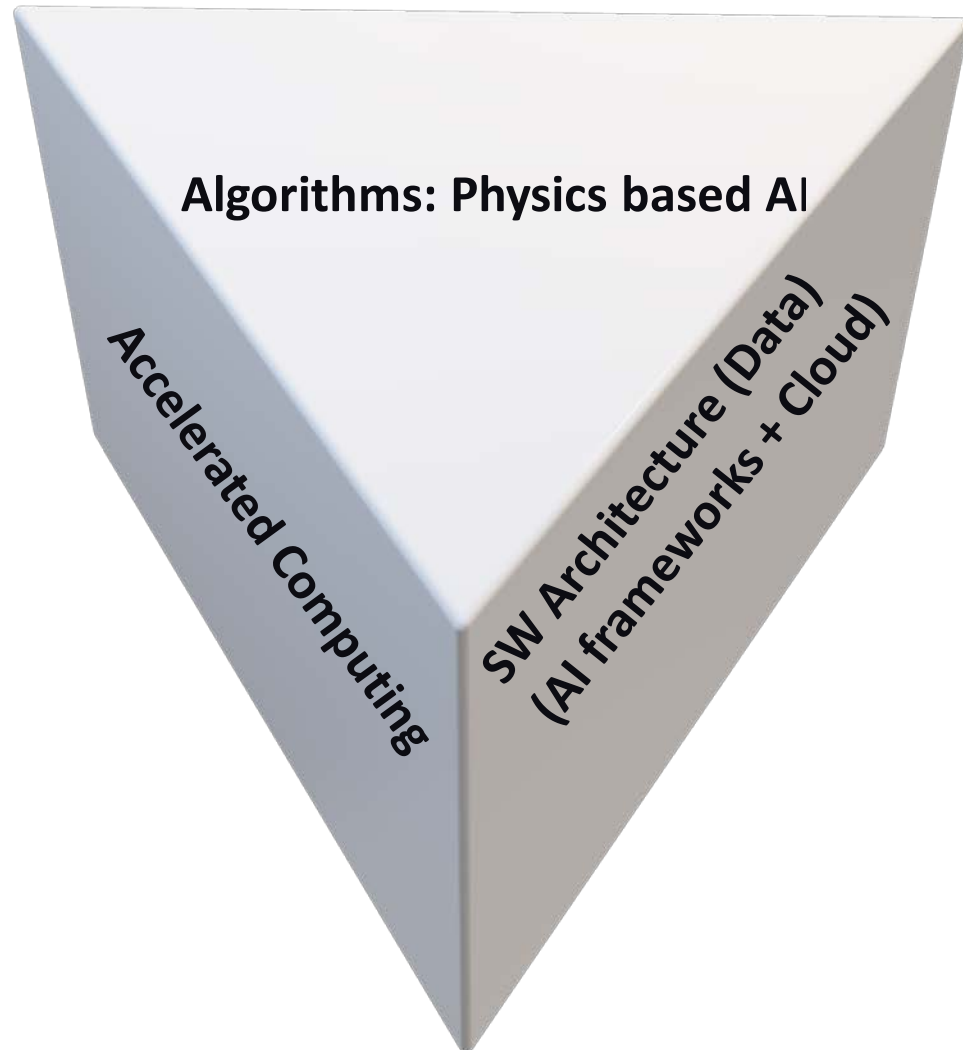
1. SW Stack : As usual it's the key to success
2. Embrace:
 - 80/20 rule of Data Science
 - Physics based constraints
 - Cloud based infrastructure for on-prem
 - Heterogenous computing
3. Internal Adoption: Hire internal evangelists
4. Invest in training your human capital

Challenges

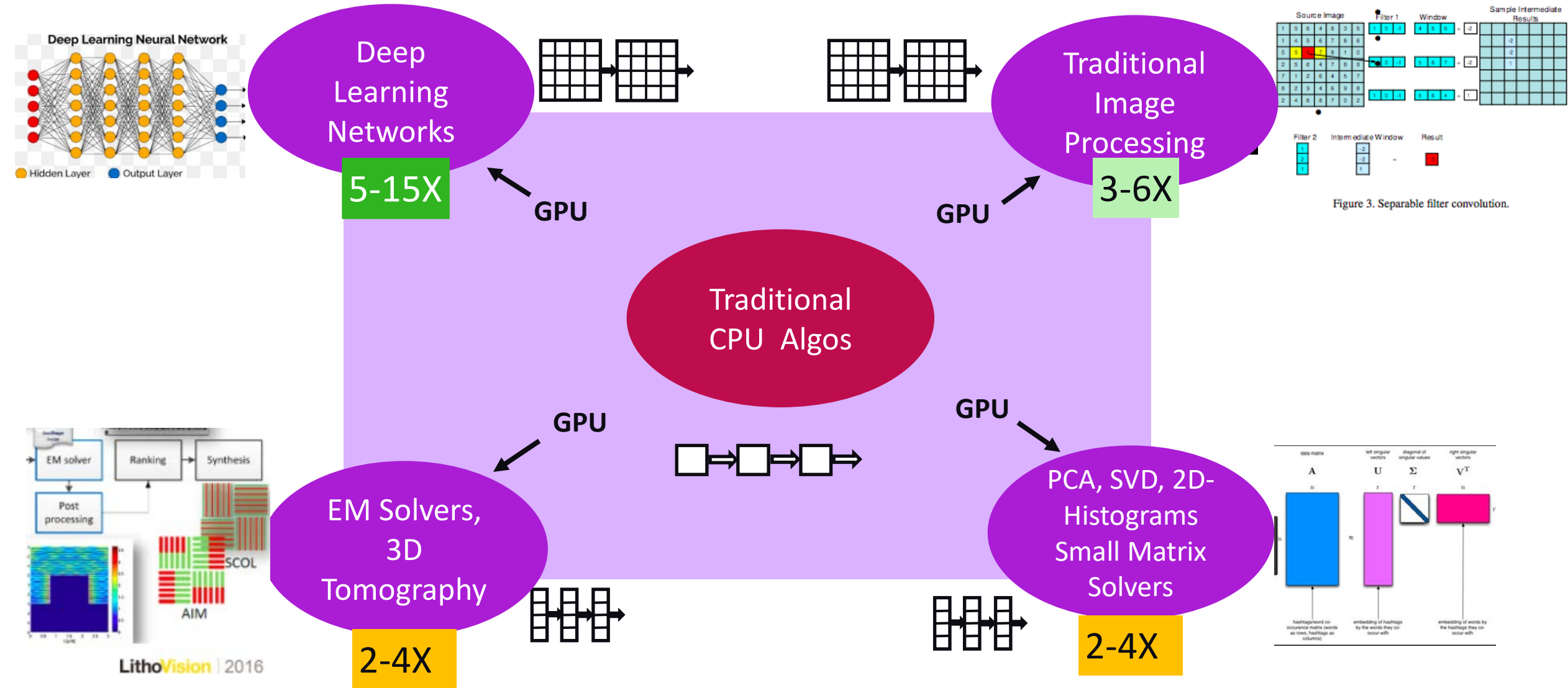
1. AI / HPC / Cloud convergence is hard
 - Micro-services may not be appropriate
 - Security issues: IP & Data leakage
2. Balancing new with proven approaches
 - Mfg floor doesn't like rapid changes
 - Mfg floors want LONG LIFE HW (GPU challenge)
3. Training data may be limited
4. Making models robust to Covariate Shift
5. Explainable Models



Keys for success < besides data >



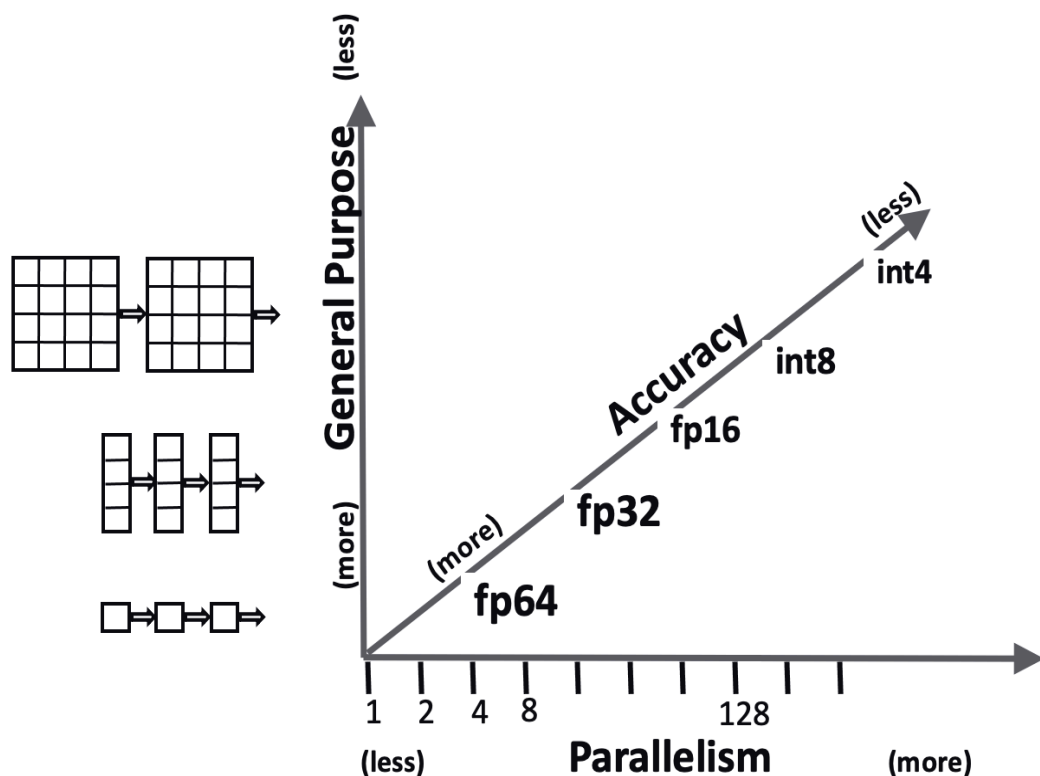
Why GPU based accelerated computing is winning



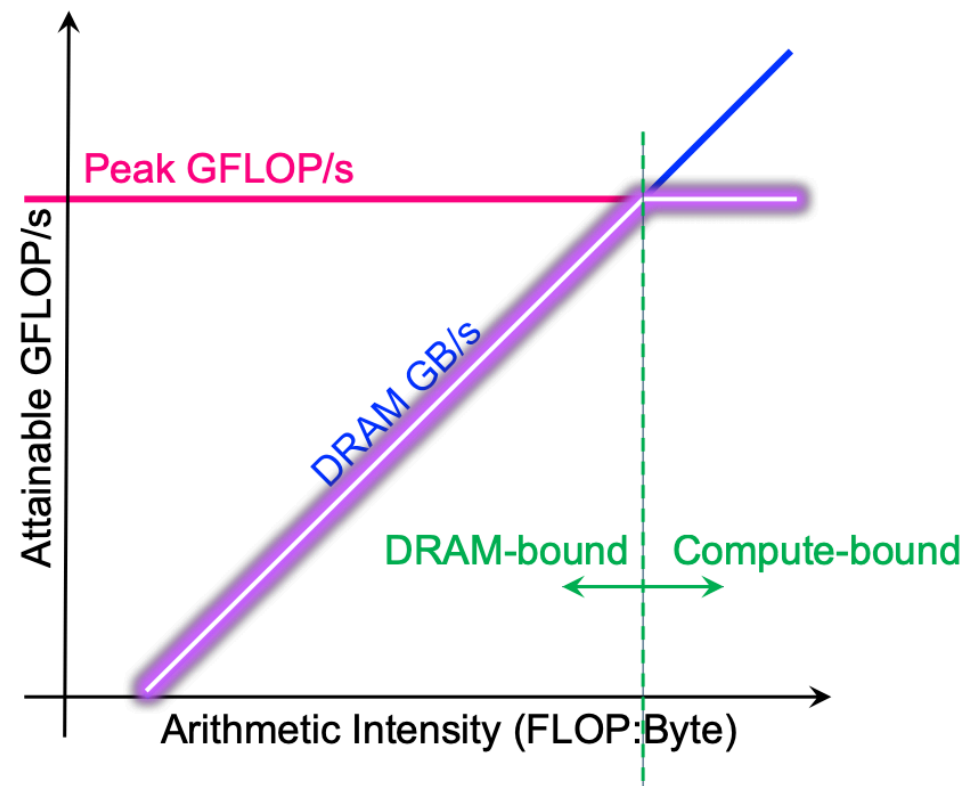
GPU “cultural advantage”



Early stages of parallel computing adoption



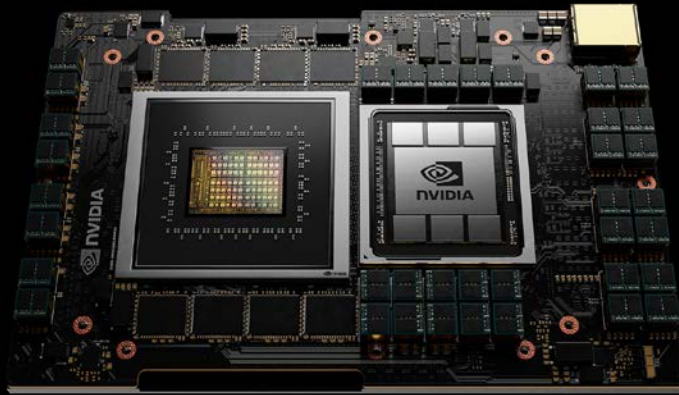
However subject to Arithmetic Intensity



Lots of choices on GP-GPUs in the next few years



NVIDIA Grace (ARM + Ampere Next)



Ponte Vecchio
Execution Progress

A0 Silicon Current Status

> 45 TFLOPS

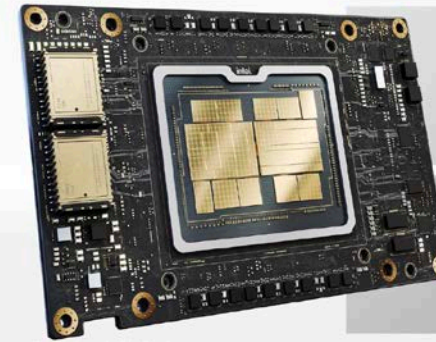
FP32 Throughput

> 5 TBps

Memory Fabric
Bandwidth

> 2 TBps

Connectivity
Bandwidth



COMPUTE GPU ARCHITECTURE ROADMAP

COMPUTE DNA FOR THE DATA CENTER



7nm



7nm



Advanced Node



2019 ————— 2022

- Does the best HW GP-GPU architecture win?
- Or is the one with the best SW eco-system?

Looking Ahead



How do we balance new HW vs Ninja programming

FAST FORWARD TO ML/ AI

Explosion of ML/AI programming models, languages, frameworks



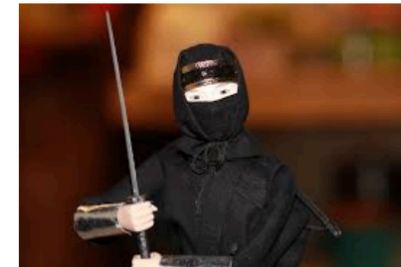
Explosion of AI chips and accelerators



Uday Bondhugula, IISc

Slides courtesy of Prof. Uday Bondhugula, IISc

STATE-OF-THE-ART DEEP LEARNING SYSTEMS: CURRENT LANDSCAPE



- ▶ Primarily driven by hand-optimized highly tuned libraries (manual or semi-automatic at most)
- ▶ Expert/Ninja programmers
- ▶ Not a scalable approach! — bleeds resources, not modular, too much repetition

Uday Bondhugula, IISc

51

Slippery slope – from high productivity zone to fast but illegible



KLA Algo / Data Scientists

Preferred Sandbox Language

MATLAB  PYTHON

KLA IMC SW Eng.

Need to scale for whole wafer

Re-writes in C (10-100X faster)

KLA/Partner Ninja Eng

Need to meet RPC targets

Re-writes in SIMD/CUDA/assembly
(100-1000X faster)

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

eg: Sobel Edge Detector Python
< Code is understandable >

```
#define the
vertical_f
```

```
#define the
horizontal
```

```
#read in the
img = plt.
```

```
#get the d
n,m,d = im
```

```
#initialize
edges_img =
```

```
#loop over
for row in
for col
```

```
#c
loc
```

```
#ap
ve
```

```
#re
ve
```

```
#ap
ho
```

```
#re
ho
```

```
#c
ed
```

```
#in
ed
```

```
#remap the
edges_img =
```

```
void FPUSobel()
{
```

```
BYTE* image_0 = g_image +
BYTE* image_1 = g_image +
```

```
BYTE* image_2 = g_image + g_image_width * 2;
DWORD* screen =
```

```
for(int y=1; y<g
{
```

```
for(int x=1;
{
```

```
float gx
```

```
float gy
```

```
int resu
```

```
screen[x
```

```
image_0 += g
image_1 += g
image_2 += g
screen += g
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

eg: Sobel Edge Detector C optimized
< Code is somewhat understandable >

```
1 void SseSobel()
2 {
3     BYTE* image_0 =
4     BYTE* image_1 =
5     BYTE* image_2 =
6     DWORD* screen =
```

```
7
8     __m128 const_p_one = _mm_set1_ps(1.0f);
9     __m128 const_p_two = _mm_set1_ps(2.0f);
10    __m128 const_n_one = _mm_set1_ps(-1.0f);
11    __m128 const_n_two = _mm_set1_ps(-2.0f);
```

```
12
13    for(int y=1; y<g_image_height-1; ++y)
14    {
```

```
15        for(int x=1; x<g_image_width-1; ++x)
16        {
```

```
17            // load 16 components. (0~6 will be used)
18            __m128i current_0 = _mm_unpacklo_epi8(_mm_loadu_si128((__m128i*)(image_0+x-1)), _mm_setzero_si128());
19            __m128i current_1 = _mm_unpacklo_epi8(_mm_loadu_si128((__m128i*)(image_1+x-1)), _mm_setzero_si128());
20            __m128i current_2 = _mm_unpacklo_epi8(_mm_loadu_si128((__m128i*)(image_2+x-1)), _mm_setzero_si128());
```

```
21
22            // image_00 = { image_0[x-1], image_0[x+0], image_0[x+1], image_0[x+2] }
23            __m128 image_00 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(current_0, _mm_setzero_si128()));
24            // image_01 = { image_0[x+0], image_0[x+1], image_0[x+2], image_0[x+3] }
25            __m128 image_01 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(_mm_srli_si128(current_0, 2), _mm_setzero_si128()));
26            // image_02 = { image_0[x+1], image_0[x+2], image_0[x+3], image_0[x+4] }
27            __m128 image_02 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(_mm_srli_si128(current_0, 4), _mm_setzero_si128()));
28            __m128 image_10 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(current_1, _mm_setzero_si128()));
29            __m128 image_12 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(_mm_srli_si128(current_1, 4), _mm_setzero_si128()));
30            __m128 image_20 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(current_2, _mm_setzero_si128()));
31            __m128 image_21 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(_mm_srli_si128(current_2, 2), _mm_setzero_si128()));
32            __m128 image_22 = _mm_cvtepi32_ps(_mm_unpacklo_epi16(_mm_srli_si128(current_2, 4), _mm_setzero_si128()));
```

```
33
34            __m128 gx = _mm_add_ps(_mm_mul_ps(image_00,const_p_one),
35                                   _mm_add_ps(_mm_mul_ps(image_02,const_n_one),
36                                                 _mm_add_ps(_mm_mul_ps(image_10,const_p_two),
37                                                             _mm_add_ps(_mm_mul_ps(image_12,const_n_two),
38                                                             _mm_add_ps(_mm_mul_ps(image_20,const_p_one),
39                                                             _mm_mul_ps(image_22,const_n_one))))));
```

```
40
41            __m128 gy = _mm_add_ps(_mm_mul_ps(image_00,const_p_one),
42                                   _mm_add_ps(_mm_mul_ps(image_01,const_p_two),
43                                                 _mm_add_ps(_mm_mul_ps(image_02,const_p_one),
44                                                             _mm_add_ps(_mm_mul_ps(image_20,const_n_one),
45                                                             _mm_add_ps(_mm_mul_ps(image_21,const_n_two),
46                                                             _mm_mul_ps(image_22,const_n_one))))));
```

```
47
48            screen[x] = gx;
49        }
50    }
```

```
51
52    }
```

```
53
54    }
```

```
55
56    }
```

```
57
58    }
```

```
59
60    }
```

```
61
62    }
```

```
63
64    }
```

But wait : Real World Algo. Optimization is worse



GPU-Histogram Code example

```
1 __global__ void kernelHistogram1D(TensorInfo a,
2   TensorInfo b, rhine, minvalue, maxvalue,
3   smem[];
4
5   output_t* smem;
6
7   // PART A: Initialize shared memory counters
8   smem = reinterpret_cast<output_t*>(my_smem);
9   for (int i = threadIdx.x; i < a.sizes[0];
10      i += blockDim.x) { smem[i] = 0; }
11   __syncthreads();
12
13   // PART B: Go over the input b to increment shared
14   ↪ counters
15   FOR_KERNEL_LOOP(1, blockDim.x) {
16       const int bOffset = IndexToOffset(b.sizes[0], i);
17       const input_t bVal = b.data[bOffset];
18       if (bVal >= minvalue && bVal <= maxvalue) {
19           const int bin = IndexToOffset(smem.sizes[0], bVal - minvalue);
20           atomicAdd(&smem[bin], 1);
21       }
22   }
23   __syncthreads();
24
25   // PART C: Increment counters
26   ↪ counters
27   for (int i = threadIdx.x; i < a.sizes[0]; i +=
28      blockDim.x) {
29       const IndexType aOffset =
30           IndexToOffset<output_t, IndexType>(a.sizes[0], i);
31       ↪ ADims>::get(i, a);
32       atomicAdd(&a.data[aOffset], smem[i]);
33   }
```

Fig. 3: Histogram kernel.

GPU-Fused Histogram example

```
1 void fused_kernel(...) {
2   // Prologue of the fused kernel
3   ...
4   int threadIdx_x, threadIdx_y, threadIdx_z;
5   int blockDim_x, blockDim_y, blockDim_z;
6   if (global_tid < 896) {
7       blockDim_x = 896 / 16;
8       blockDim_y = 16; blockDim_z = 1;
9       threadIdx_x = global_tid % blockDim_x;
10      threadIdx_y = global_tid / blockDim_x %
11          ↳ blockDim_y;
12      threadIdx_z = 1;
13   } else {
14       blockDim_x = 128;
15       blockDim_y = 1; blockDim_z = 1;
16       threadIdx_x = (global_tid - 896) % blockDim_x;
17       threadIdx_y = 1; threadIdx_z = 1;
18   }
```

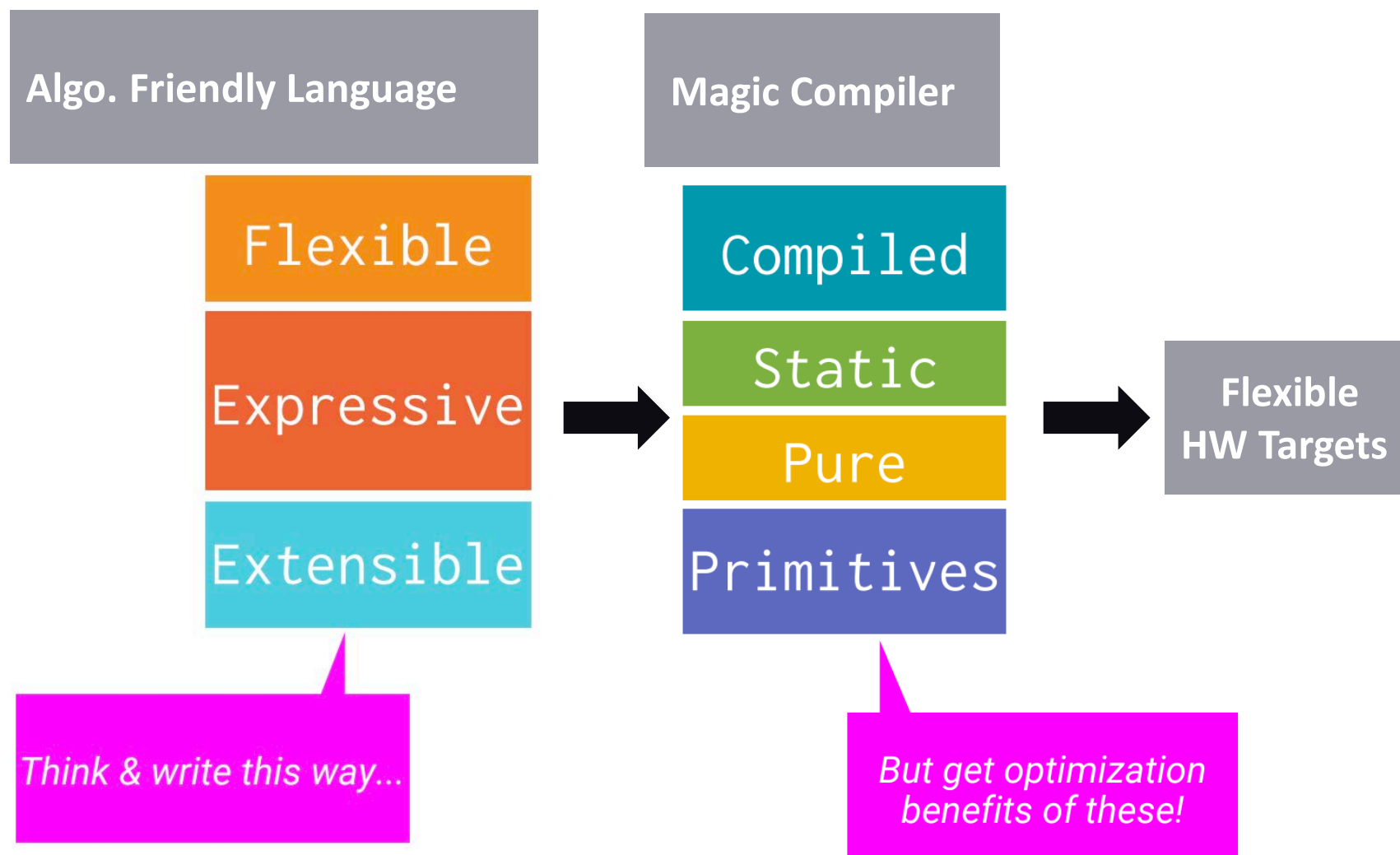
Most real-world algorithms are not compute bound.

- Multiple kernels have to be “fused” to reduce memory bottlenecks
- Original Code intent “vanishes”

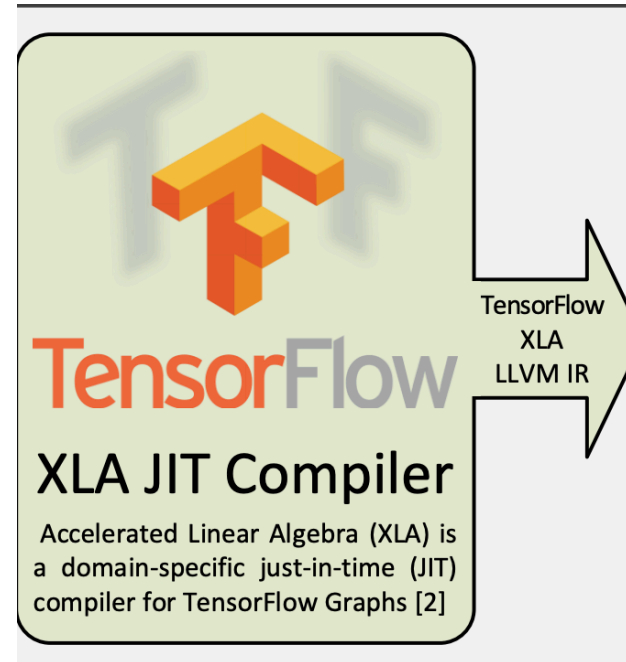
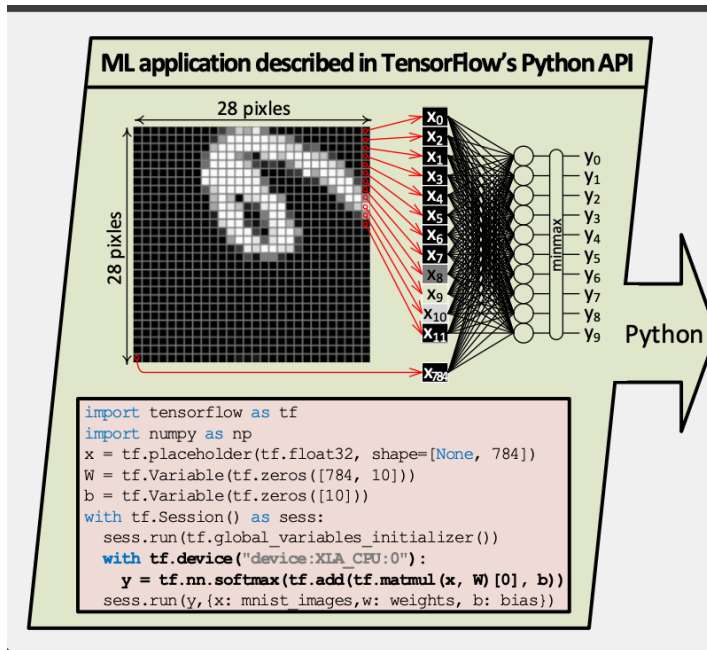
```
32 asm("bar.sync 1, 896;");
33 // batch_norm_collect_statistics() PART C
34 ...
35 K1_end:
36 if (global_tid < 896) goto K2_end;
37 // kernelHistogram1D() PART A
38 smem = reinterpret_cast<output_t*>(my_smem);
39 for (int i = threadIdx_x; i < a.sizes[0];
40      i += blockDim_x) { smem[i] = 0; }
41 // A PTX assembly to only sync 128 threads.
42 asm("bar.sync 2, 128;");
43 // kernelHistogram1D() PART B
44 ...
45 asm("bar.sync 2, 128;");
46 // kernelHistogram1D() PART C
47 ...
48 K2_end:
49 }
```

Fig. 4: HFUSE fused kernel.

What do we all want ?



Google's XLA compiler (codegen for CPU/GPU/TPU)



Shipping Today

CPU (X86 & ARM)

GPU (NVIDIA)

TPU (Google)

- XLA & TVM are very early success demonstrations
- Industry is now pivoting to MLIR open-source initiative
- Is this the possible future ?

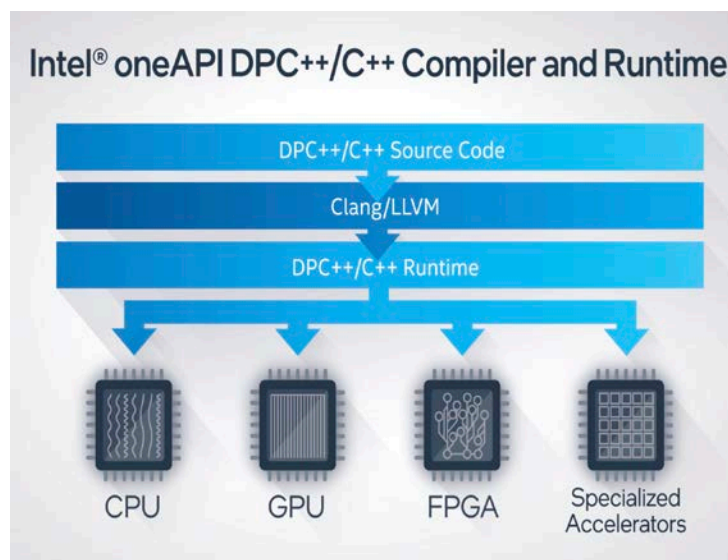

Modern C++ will also be a significant player

- Modern C++ (11/14/17/20) is having a significant revival
- Many language constructs specifically designed with parallelism in mind
- At the same time, many constructs to enable developer productivity
- It is the basis for most modern advanced HPC + AI (TensorFlow C++ / Intel Sycl DPC++ / Nvidia HPC compiler



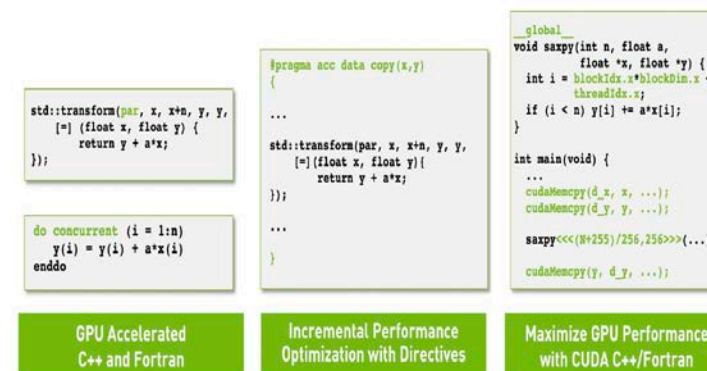
```
// Add the graph to the session
status = session->Create(graph_def);
if (!status.ok()) {
    std::cout << status.ToString() << "\n";
    return 1;
}

// Feed dict
std::vector<std::pair<std::string, tensorflow::Tensor>> inputs = {
    { "Input:0", Input0 },
};
status = session->Run(inputs, {"Layer2/Output"}, {}, &output);
if (!status.ok()) {
    std::cout << status.ToString() << "\n";
    return 1;
}
auto Result = output[0].matrix<float>();
std::cout << "Input: 0 | Output: " << Result(0,0) << std::endl;
```



THE FUTURE OF PARALLEL PROGRAMMING

Standard Languages | Directives | Specialized Languages



In Conclusion



- Very exciting time to be an engineer working in the intersection of AI + HPC + Cloud
- Semiconductors are becoming an even more critical part of the global economy
- Semi Inspection & Metrology requires cutting edge AI + HPC technologies to keep progressing
- KLA and IITM have had active research collaborations since 2003
- The KLA India AI ACL (AI Advanced Computing labs) located in the IIT M research park will enable more opportunities for research collaboration.
- Contact Dr. Pradeep Ramachandran for avenues of collaborations / internships / employment at KLA

Thank you

