Fast Parallel Similarity Calculations with FPGA Hardware

Dan McCreary, Distinguished Engineer, Optum

Kumar Deepak, Distinguished Engineer, Xilinx

Graph + AI World 2020

September 29, 2020

Talk Description

The foundation of recommendation is finding **similar** customers and their purchasing patterns. Yet, if you have 100 million customers it can take hours to do similarity calculations on just 200 features. However, since these calculations can be done in parallel, we show that using an FPGA can allow these calculations to be done in under 30 msec. This session will show how using TigerGraph User Defined Functions (UDF), similarity calculations, and therefore product recommendations can be done in real-time as customers visit your web site.

Overview

Dan:

- What is graph similarity?
- Why is it critical in recommendation systems?
- Serial vs. Parallel algorithms
- Cosine similarity and graph embeddings

Kumar:

- What is an FPGA?
- FPGA configuration used in our benchmarks
- Calling FPGA from TigerGraph using a User Defined Function
- Benchmark Results

Summary: Both

About Dan

- Distinguished Engineer at Optum Healthcare (330K employees and 32K technical staff)
- Focused on AI enterprise knowledge graphs
- Help create the worlds largest healthcare graph
- Coauthor of book "Making Sense of NoSQL”
- Worked at Bell Labs as a VLSI circuit designer
- Worked for Steve Jobs at NeXT

Why Are Similarity Calculations Critical?

Similarity is at the foundation of recommendation engines

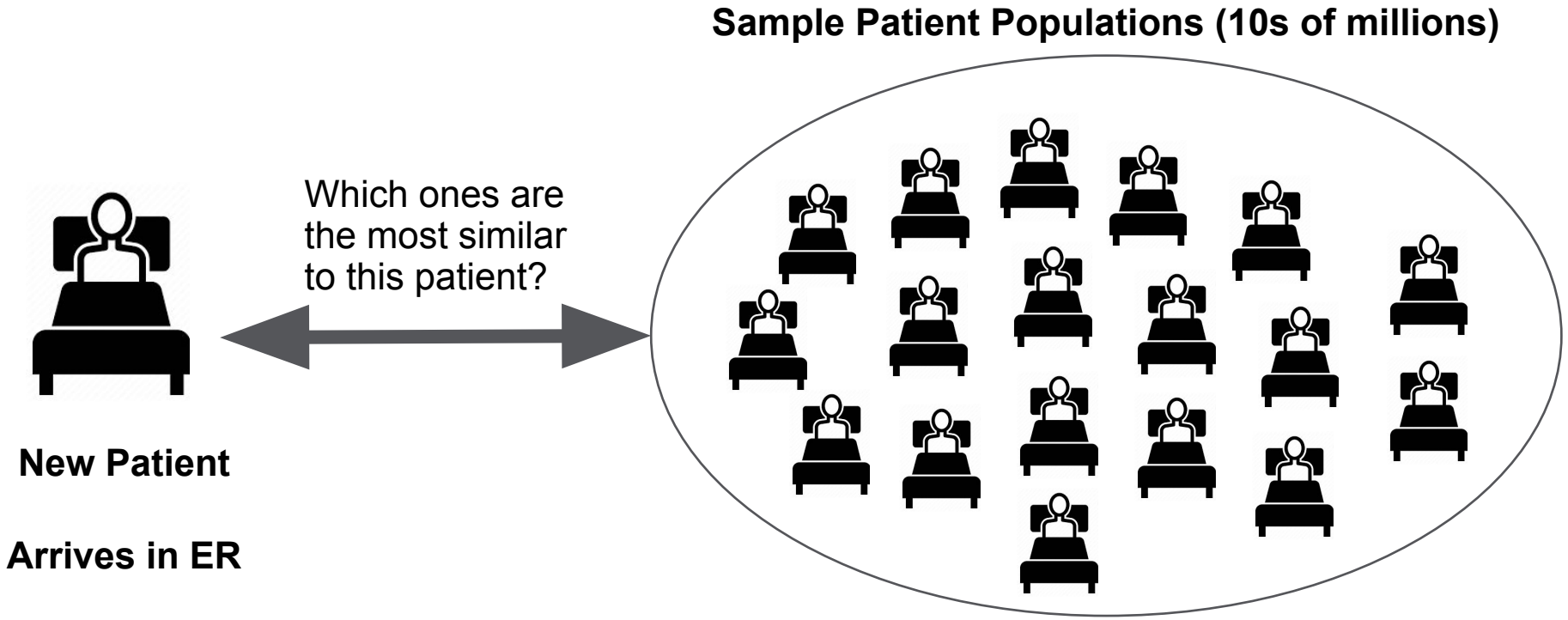
Recommendation engines power sites like:

- Google – recommend a document
- NetFlix™ – recommend a movie
- Amazon - recommend a product
- Pintrest™ – recommend an interest
- Healthcare – recommend a care path

Recommendations must take into account many factors including recent searches

To be useful in interactive web sites we set a goal of response times of under 200 milliseconds

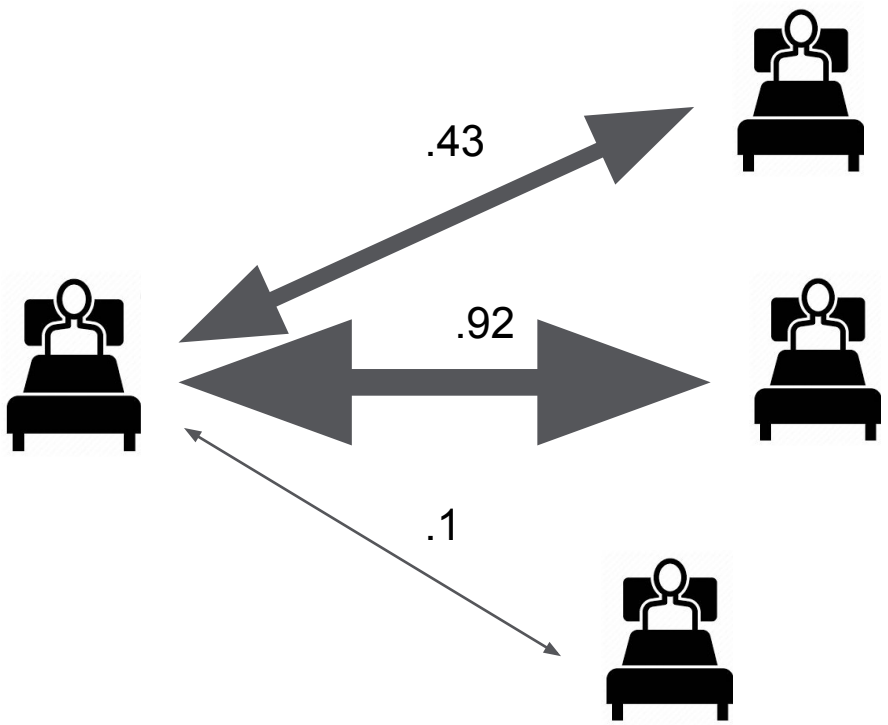
Real-Time Patient Similarity



- Given a new patient that arrives a clinical setting, how can we **quickly** find the most similar patients?
- Assumption: we have 10M clinical records of our population of 235 million members
- Can we find the 100 most similar patients in under 200 milliseconds?

OPTUM®

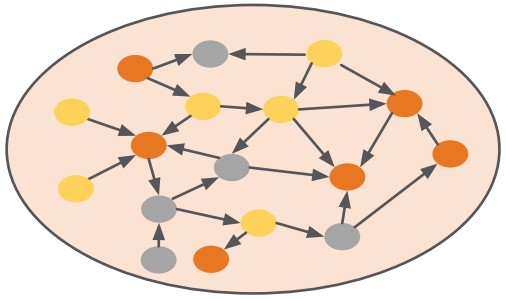
Similarity Score – A scaled measure of “alikeness” for a context



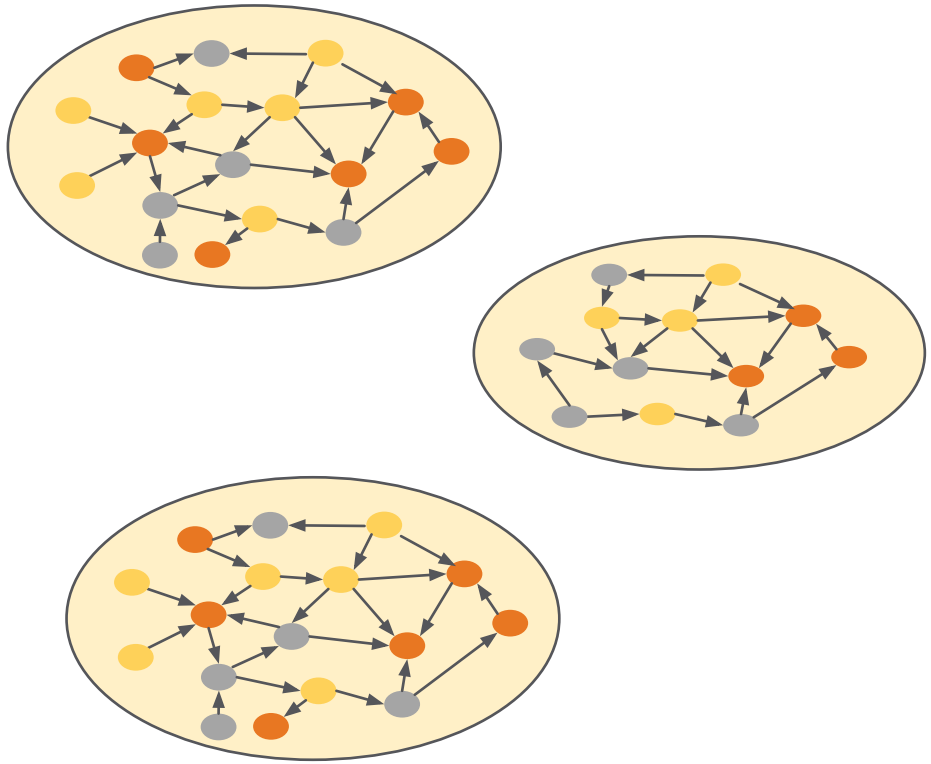
- A single scaled dimension of comparison for a given setting or context
- Comparing a patient to itself would have a similarity score of 1.0
- Patients that have few common characteristics would have a score of 0.1

Graph Representation of Patients – Includes Structure

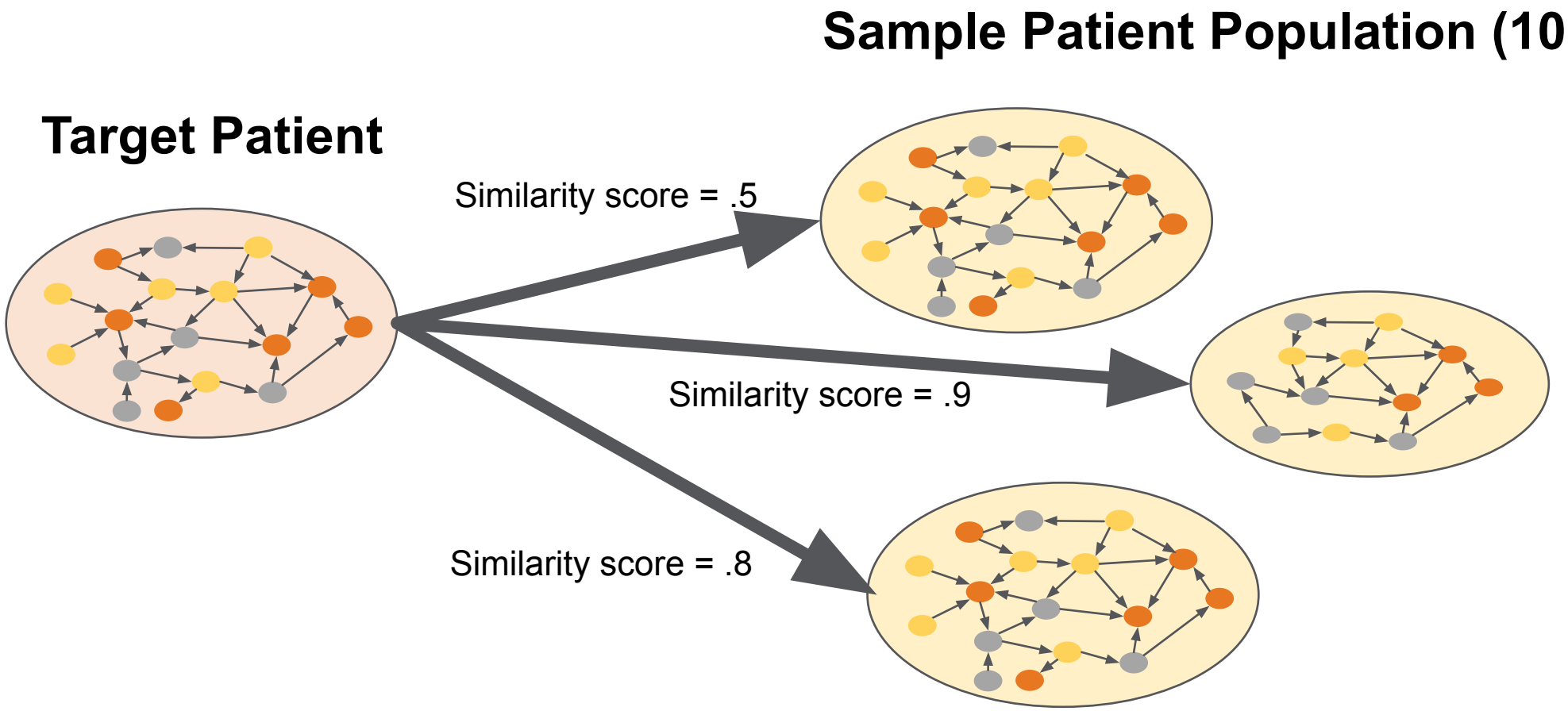
Target Patient



Sample Patient Population (10M)



Graph Representation of Patients – Includes Structure



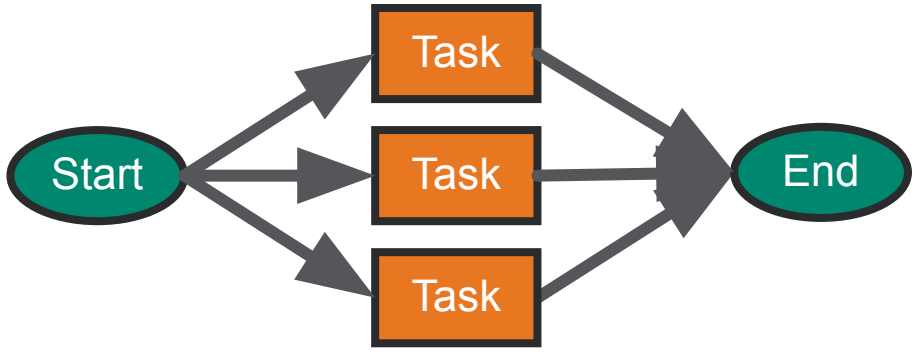
How can I **quickly** compare these graphs and find the most similar patients?

Serial vs. Parallel Graph Algorithms

Serial Graph Algorithms

- One task cannot begin before the prior task is complete
- Task order is important
- Serial algorithms work well on traditional CPUs

Parallel Graph Algorithms



- Many tasks can be done independently
- Task order in not relevent
- Tasks can usually be done faster on GPU or FPGAs

The Human Brain Does Both Serial and Parallel Computation

Let’s use our brains as a demo!

The following slide has photos of two people

- One is a famous actor
- The other is a synthetically generated image of a person (generated by a AI program)

How long will it take for you to recognize which one the famous actor?

Which photo is the famous actor?

What just happened?

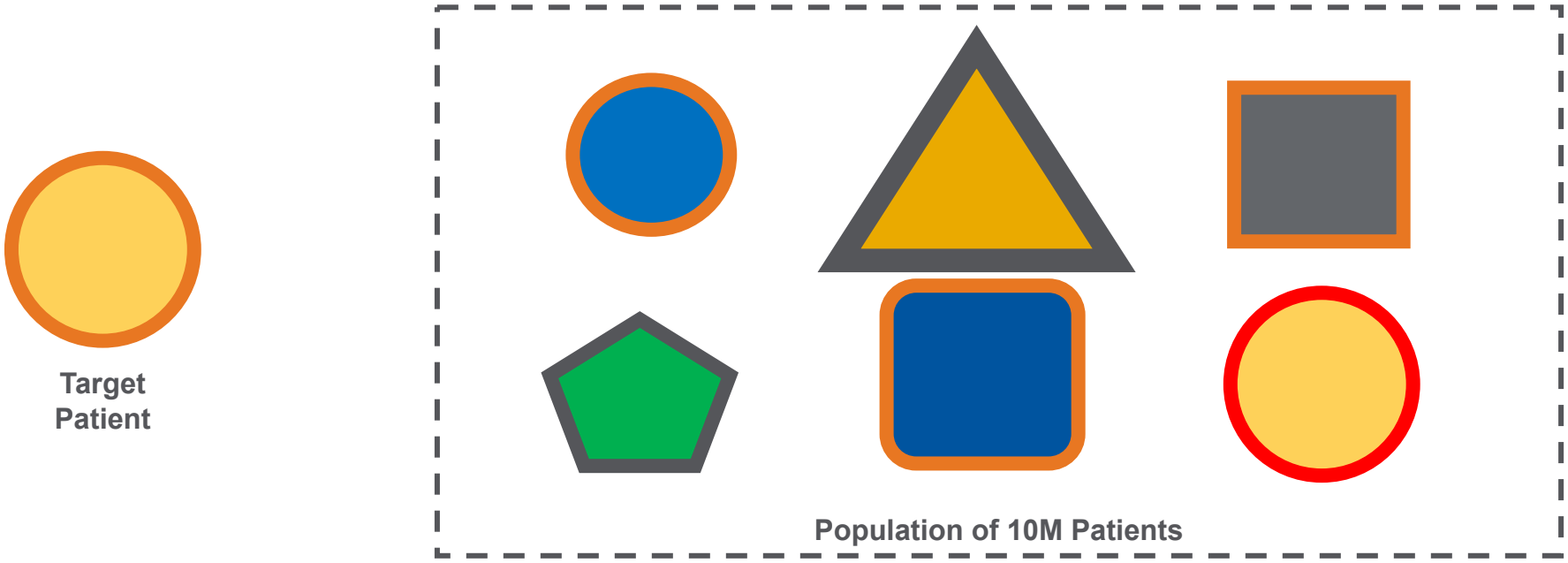
1. Your visual cortex received the images as electrical signals from your eyes
2. Your brain identified key **features** of each face from the images - in **parallel**
3. Your brain sent these features as electrical signals to your memories of people's faces
4. Your brain compared these features to **every memory you have ever had of a person's face** – in **parallel**
5. Your brain sent their recognition scores to a control center of your brain
6. Your brain's speech center vocalized the word "right" – in **series**

Key Questions:

1. How does the brain know to pay **attention** to specific features of a face?
2. What portions of real-time clinical decision support systems can be done cost effectively in parallel?

Answer: The human brain, comprised of around 84B neurons, does **both** parallel and series calculations

Property-based Similarity Example

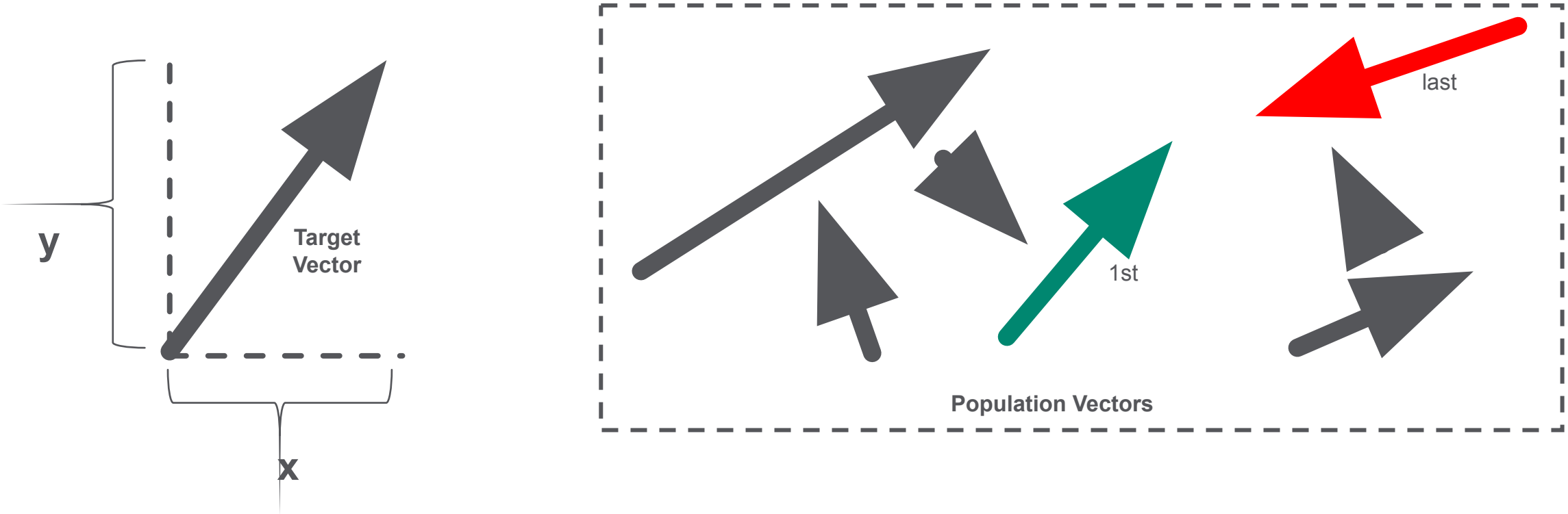


Used to find the most similar items in a graph by comparing **properties and structure**

Ideal when you a can compare individual features of an item **numerically**

Algorithms return a **ranking** of similarity between a target an a population based on the counts and weights of properties that are similar

Vector Similarity



Vectors are **similar** in two dimensional space if they have the same length and direction

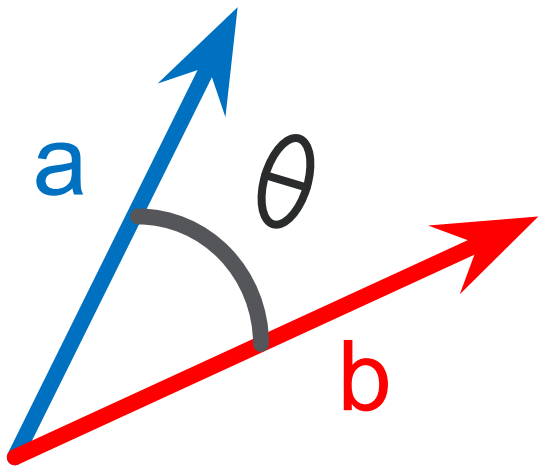
Compare all the “x” lengths and the “y” lengths and rank by the sum of the totals of the difference

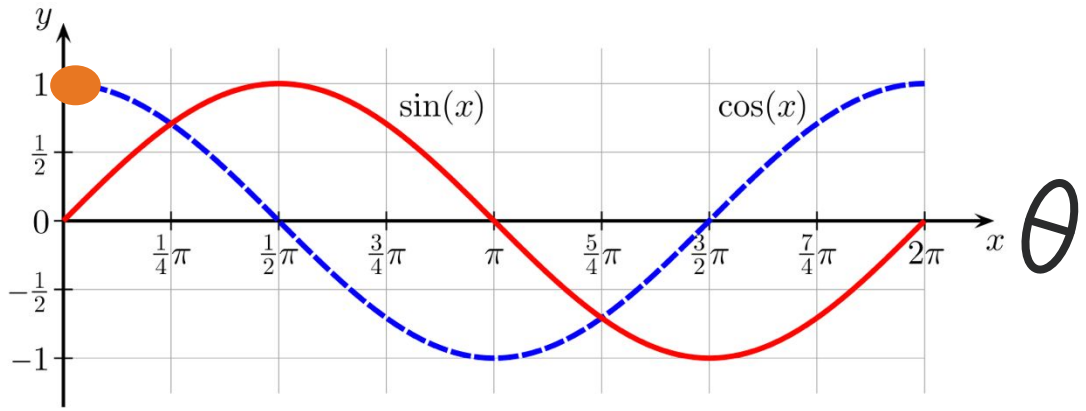
Vector Representation

Each item can be represented by a series of “feature” vectors

The numbers are scalers

Cosine Used in Comparing Direction





The dot product of two vectors a and b (sometimes called the inner product, or, since its result is a scalar, the scalar product) is denoted by a · b and is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

If the lines are **exactly** in the same direction, then theta is 0, **cos(0) = 1**

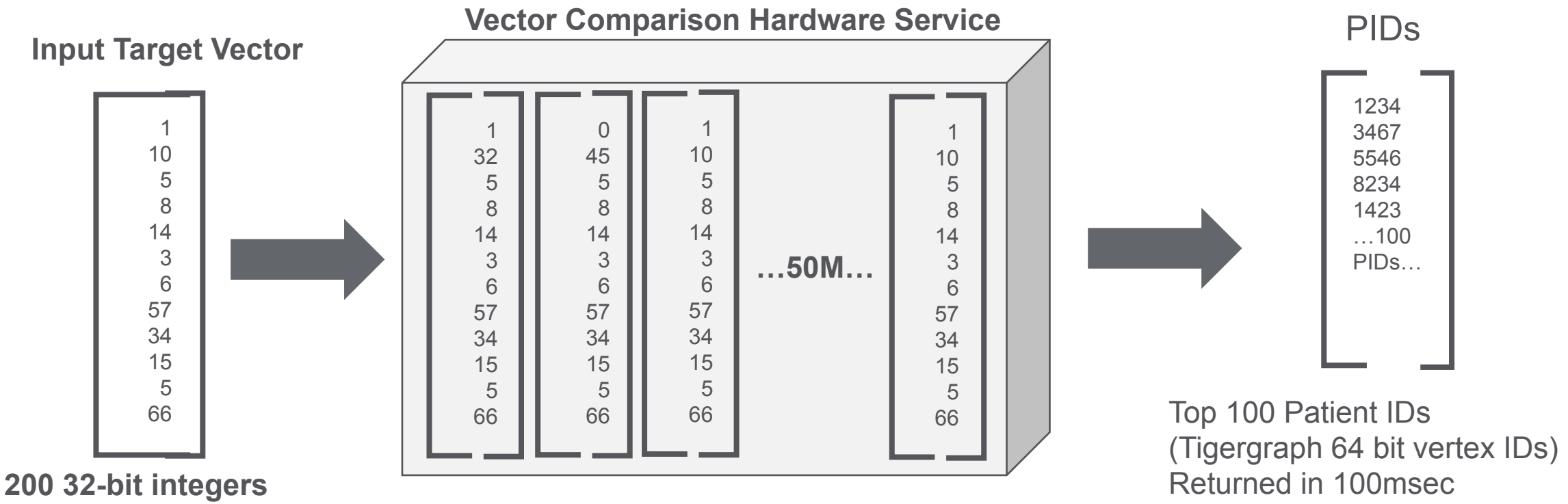
If the lines are 90 or -90 degrees apart they are in orthogonal directions cos(90) = 0

Why Vector Conversion

Computers are very good at comparing numeric values

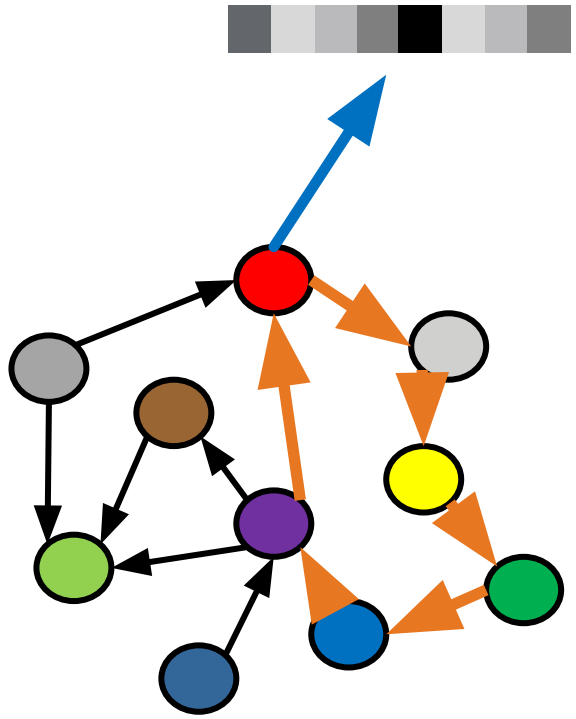
Comparison of vectors is a well studied problem (weighted cosine similarity)

Comparing a target vector to **many** other vectors (50M) is a class of “Embarrassingly Parallel” type problem that is perfect for hardware acceleration using FPGAs



Can Machine Learning Tell us What Features are Important?

Embedding: 200 32-bit integers



Old way: manually create a program that will extract 200 integers for each customer that classifies their behavior

- Age
- Gender
- Location
- Responsiveness to e-mail survey
- How proactive are they about their health?
- Likely to recommend your company
- Slow process requiring manual coding of feature extraction rules

New algorithms such as node2vec use a random walk algorithms to **automatically** create the 200 integers that will help use differentiate patients

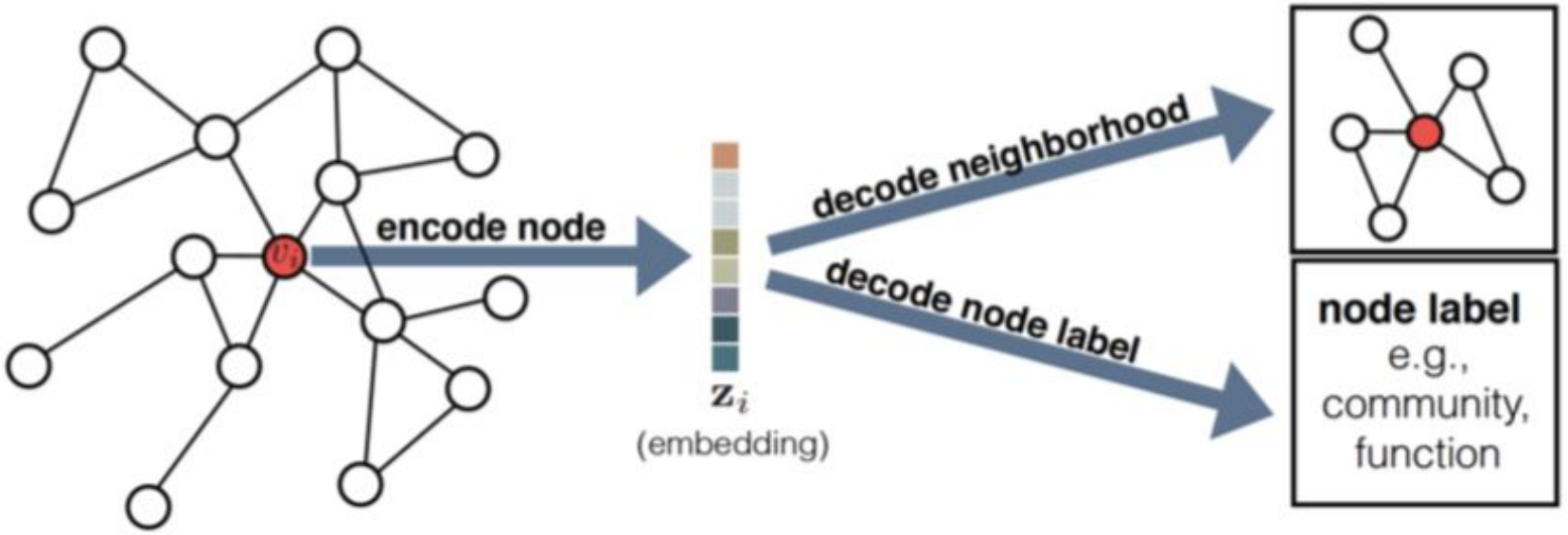
The Rise of Automatic Feature Engineering

*Recent years have seen a surge in approaches that **automatically** learn to encode graph structure into low-dimensional **embeddings**.*

*The central problem in machine learning on graphs is finding a way to incorporate information about the **structure** of the graph into the machine learning model.*

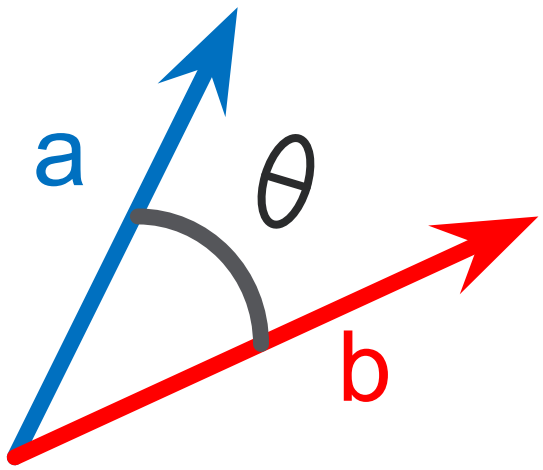
From Representation Learning on Graphs: Methods and Applications by Hamilton (et. al.)

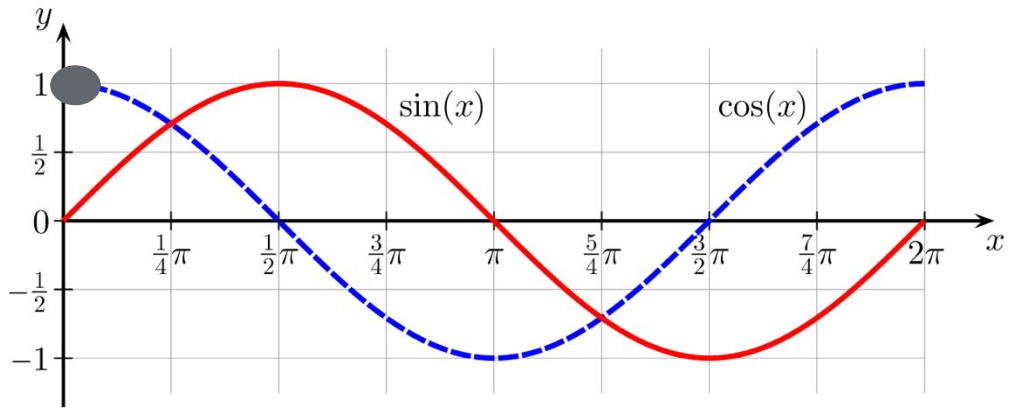
Example of Graph Embedding – Encode and Decode



From Representation Learning on Graphs: Methods and Applications

Cosine Used in Comparing Direction





The dot product of two vectors a and b (sometimes called the inner product, or, since its result is a scalar, the scalar product) is denoted by a · b and is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

If the lines are **exactly** in the same direction, then theta is 0, cos(0) = 1

If the lines are 90 or -90 degrees apart they are in orthogonal directions cos(90) = 0

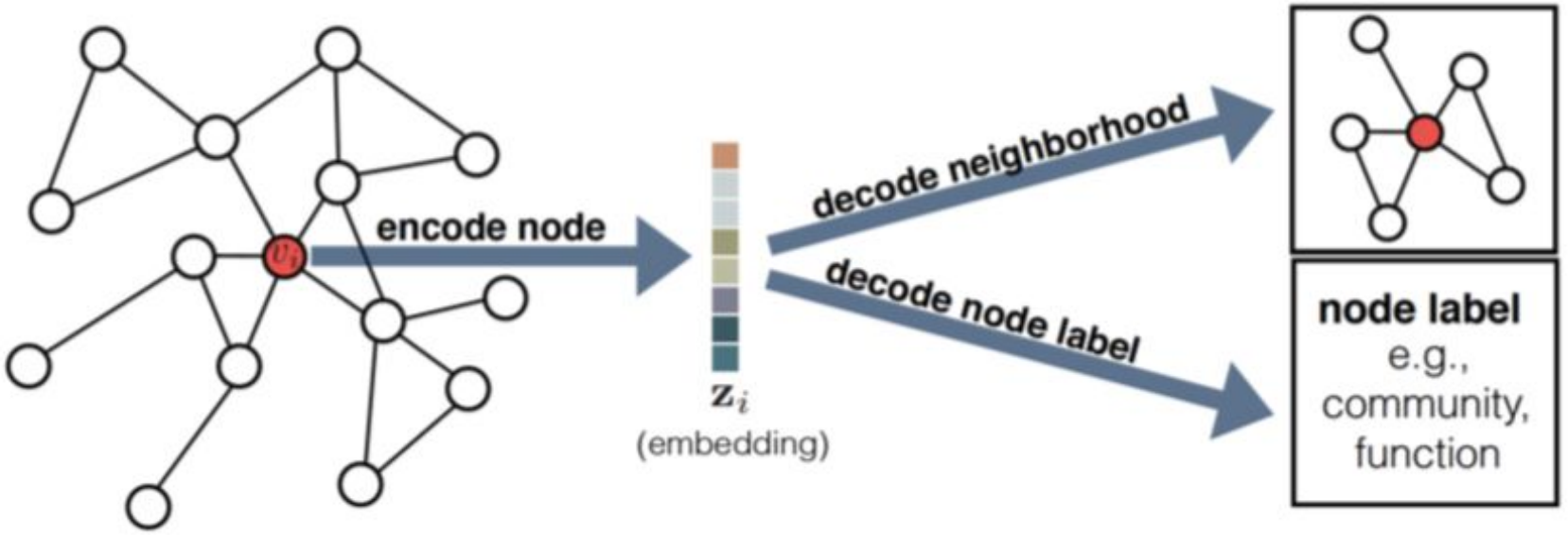
The Rise of Automatic Feature Engineering

*Recent years have seen a surge in approaches that **automatically** learn to encode graph structure into low-dimensional **embeddings**.*

*The central problem in machine learning on graphs is finding a way to incorporate information about the **structure** of the graph into the machine learning model.*

From Representation Learning on Graphs: Methods and Applications by Hamilton et. El.

Example of Graph Embedding – Encode and Decode



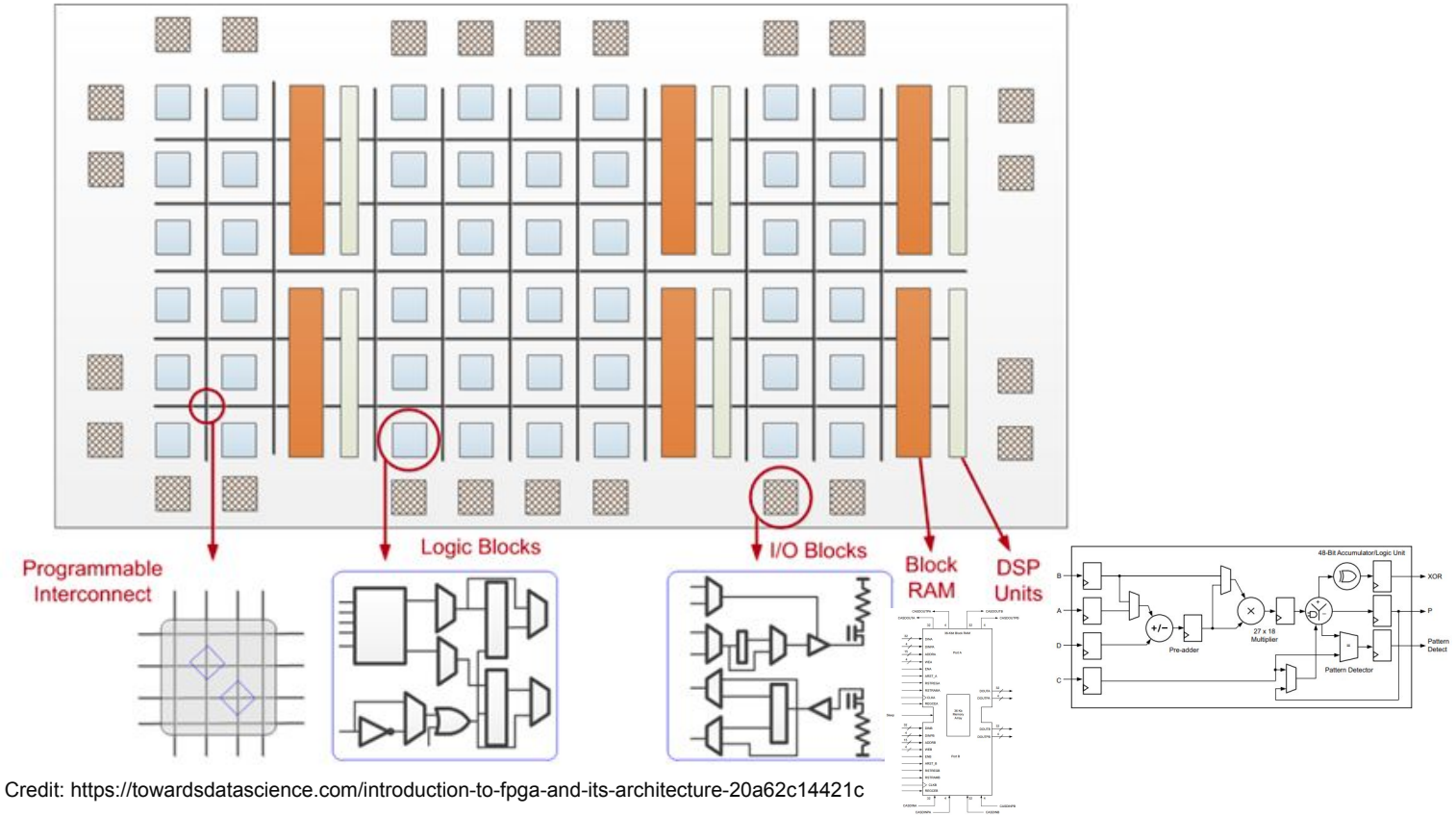
From Representation Learning on Graphs: Methods and Applications by Hamilton et. El.

About Kumar

- Distinguished Engineer at Xilinx Inc.
  - Focused on Data Analytics Acceleration
  - Architected and co-developed Xilinx Simulator, Vitis Profiler and Debugger from scratch in prior assignments
  - 20+ US patents
- Xilinx Inc:
  - Inventor of FPGAs
  - Leader in Adaptive Compute Acceleration
  - ~4.8K employees, ~3B revenue, ~25 B market cap

What is an FPGA (Field Programmable Gate Array)?

- Logic blocks
  - Look-up tables – combinatorial logic
  - Flip flops – sequential logic
- DSP (Digital Signal Processing)
  - Pre-adder, Multiplier, Accumulator
  - And, OR, NOT, NAND, NOR, XOR, XNOR
  - Pattern Detector
- Writable Memory
  - LUTRAM (Look-up table RAM)
  - BRAM (Block RAM)
  - URAM (Ultra RAM)
- Communication
  - I/O, Transceiver, PCIe, Ethernet
- Programmable Interconnect



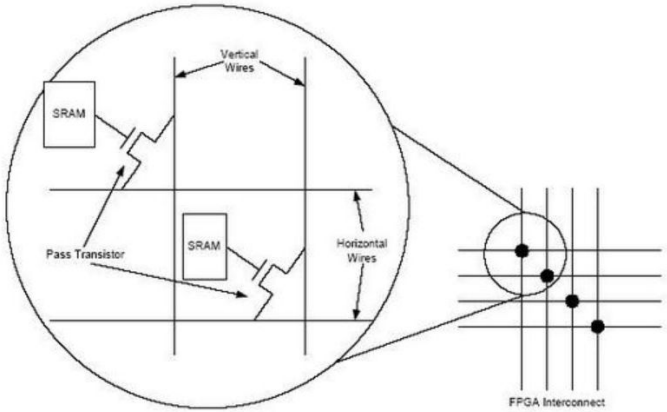
Credit: https://towardsdatascience.com/introduction-to-fpga-and-its-architecture-20a62c14421c

Xilinx VU9P FPGA has:

LUTs: 1.2 M
Flip-Flops: 2.4M
Writable Memory: 47 MB
DSP Units: 6800

XILINX.

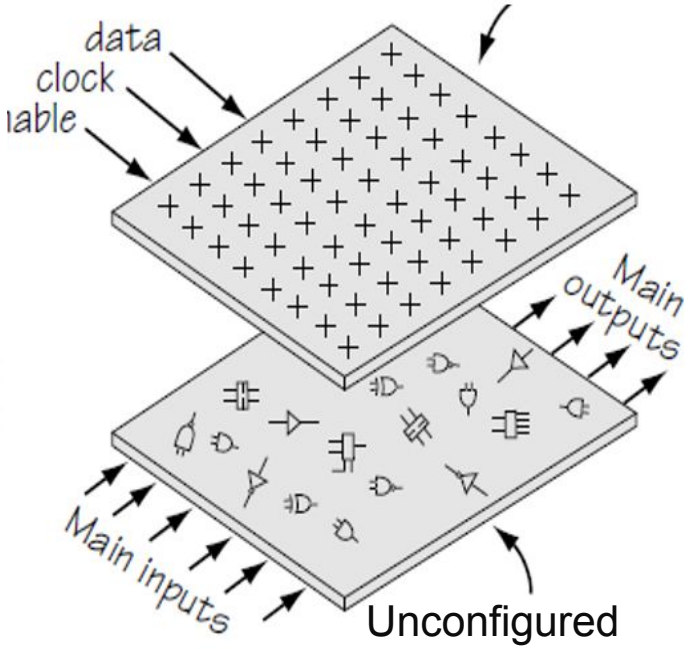
Configuring an FPGA



SRAM Driving a pass transistors to make connections

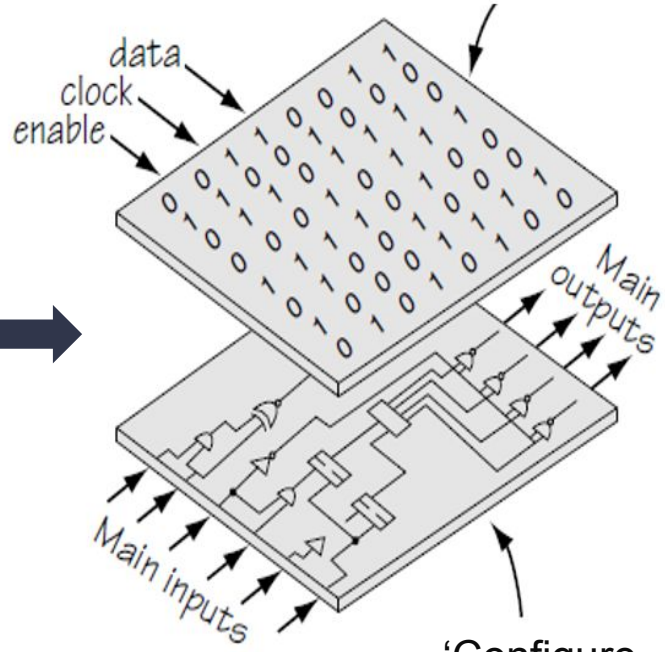
Credit: https://www.researchgate.net/publication/288835032_FPGA_Implementation_of_CORDIC_Processor

‘Unprogrammed’
configuration memory (SRAM cells)



Unconfigured logic circuit

‘Programmed’
configuration memory (SRAM cells)

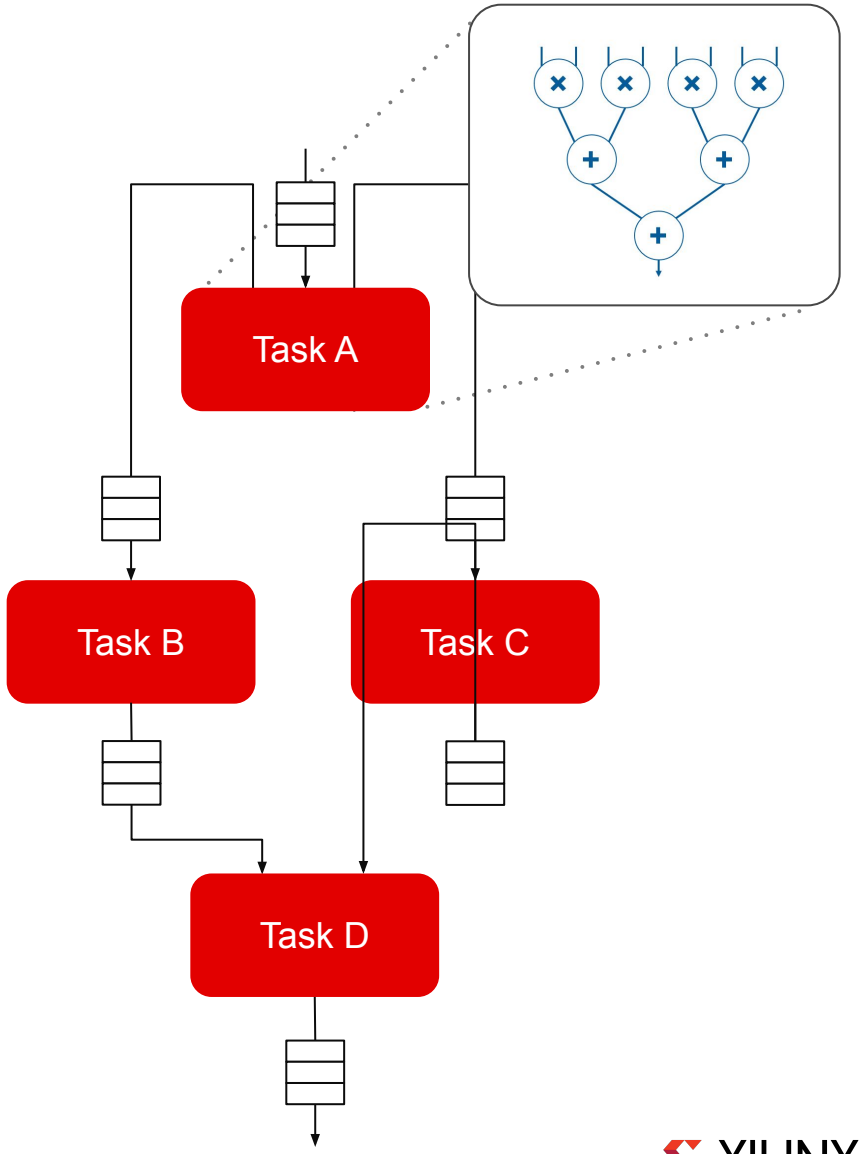


‘Configured’
logic circuit

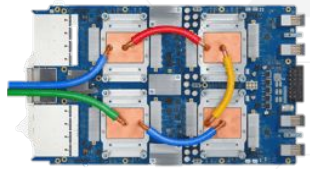
Credit: ‘Bebop to the Boolean Boogie: An Unconventional Guide to Electronics’

High-Performance FPGA Applications: Think “Parallel”

- Data-level parallelism
  - Processing different blocks of a data set in parallel
- Task-level parallelism
  - Executing different tasks in parallel
  - Executing different tasks in a pipelined fashion
- Instruction-level parallelism
  - Parallel instructions (superscalar)
  - Pipelined instructions
- Bit-level parallelism
  - Custom word width



Computing Devices

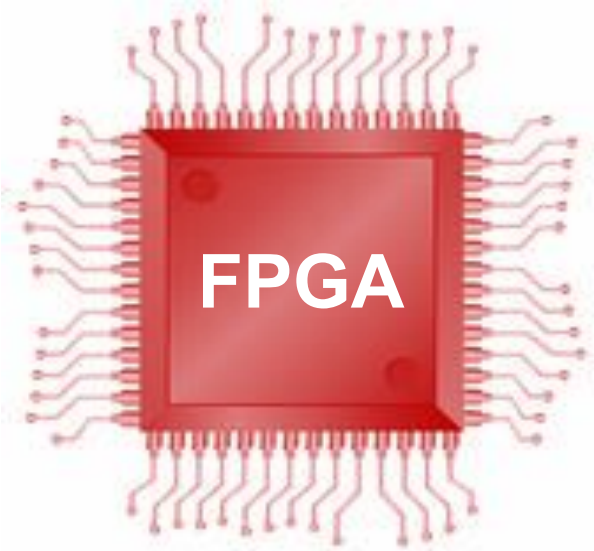


	CPU	GPU	FPGA	ASIC
Example	AMD EPYC 7702	NVIDIA A100	Xilinx Alveo U50	Google TPU
Architecture	Instruction Set	Instruction Set	Domain Specific	Domain Specific
Purpose	General Purpose	General Purpose	Domain Specific	Domain Specific
Workload Types	Serialized Workloads	Parallel Workloads	Any workload	Single Workload
Ease of Programming	Easy	Medium	Medium	No programmability
Energy Efficiency	Low	Medium	High	Very High

Using C, C++ or OpenCL to Program FPGAs

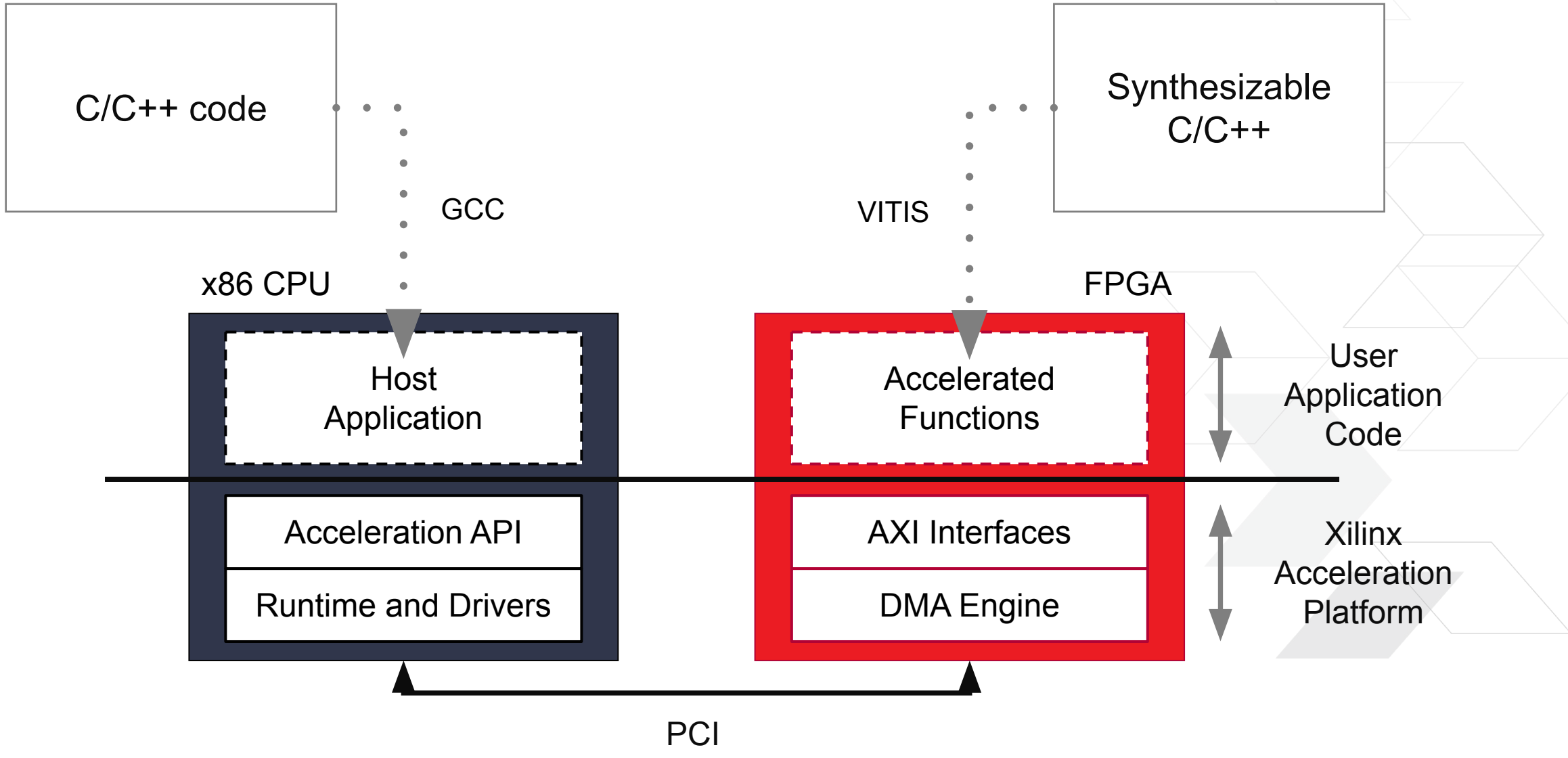
```
loop_main:for(int j=0;j<NUM_SIMGROUPS;j+=2) {
  loop_share:for(uint k=0;k<NUM_SIMS;k++) {
    loop_parallel:for(int i=0;i<NUM_RNGS;i++) {
      mt_rng[i].BOX_MULLER(&num1[i][k],&num2[i][k],ratio4,ratio3);
      float payoff1 = expf(num1[i][k])-1.0f;
      float payoff2 = expf(num2[i][k])-1.0f;
      if(num1[i][k]>0.0f)
        pCall1[i][k]+= payoff1;
      else
        pPut1[i][k]-=payoff1;
      if(num2[i][k]>0.0f)
        pCall2[i][k]+=payoff2;
      else
        pPut2[i][k]-=payoff2;
    }
  }
}
```

Vitis Compiler (v++)

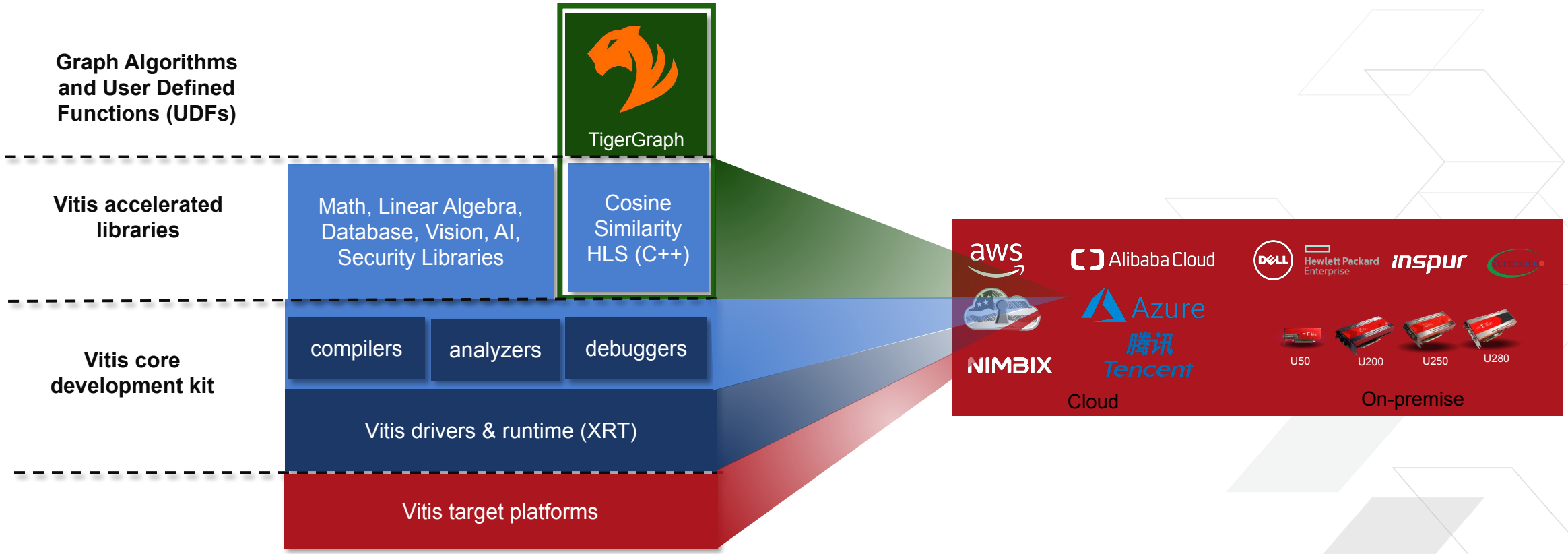


- Xilinx pioneered C to FPGA compilation technology (aka “HLS”) in 2011
- No need for low-level hardware description languages
- FPGAs are “Software Programmable”

Software Programmability: FPGA Development in C/C++



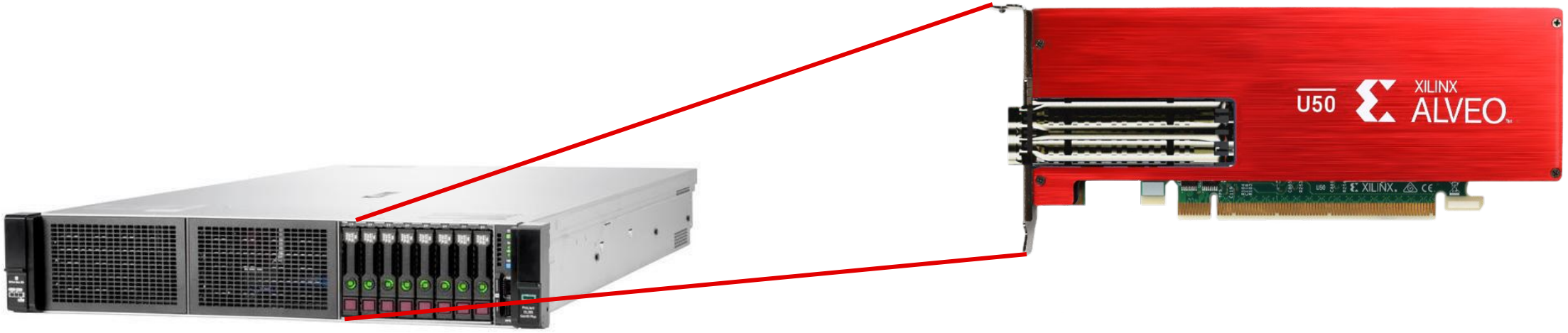
Xilinx Accelerated TigerGraph



XILINX

Benchmark: Similarity for 1.5 million patients

Dataset: Synthetic patient data generated by “Synthea” (
Algorithm: Cosine Similarity (cos theta between property vectors)
Property Vector: 197 int properties for each patient

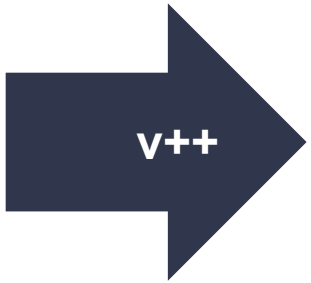


HPE DL385 Gen10+ server
2x AMD EPYC 7742 @ 2.2GHz
128 cores, 256 GB RAM

Xilinx Alveo U50 PCIE Accelerator Card
8GB HBM, 75W

Cosine Similarity: Accelerated Function

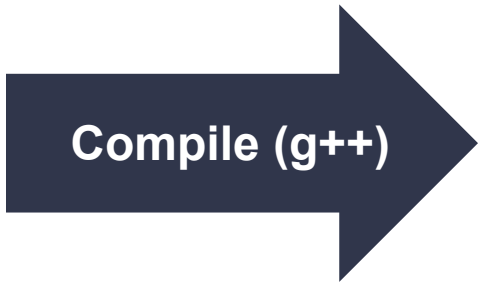
```
extern "C" void topKCosSim(uint32_t p_n, …, KVResType* p_res) {

…

 for (int i = 0; i < SPATIAL_numChannels; i++) {

#pragma HLS UNROLL

   patientInfoParse<>(p_n, p_m, l_strXs[i], l_dataX[i], l_normX[i]);

   patientInfoParse<>(p_n, p_m, l_strY[i], l_dataY[i], l_id[i], l_normY[i]);

   cos<SPATIAL_logParEntries, SPATIAL_macDataType,
SPATIAL_indexType>(p_n, p_m, l_dataX[i], l_dataY[i], l_normX[i], l_normY[i],
l_dis[i]);

   addKey<>(p_m, l_id[i], l_dis[i], l_pair[i]);

 }

 mergeStream<SPATIAL_numChannels>(p_m, l_pair, l_merge);

 maxK<SPATIAL_maxK>(p_m * SPATIAL_numChannels, p_k, l_merge, l_res);

 stream2mem<>(p_k, l_res, p_res);

}
```



similarity.xclbin

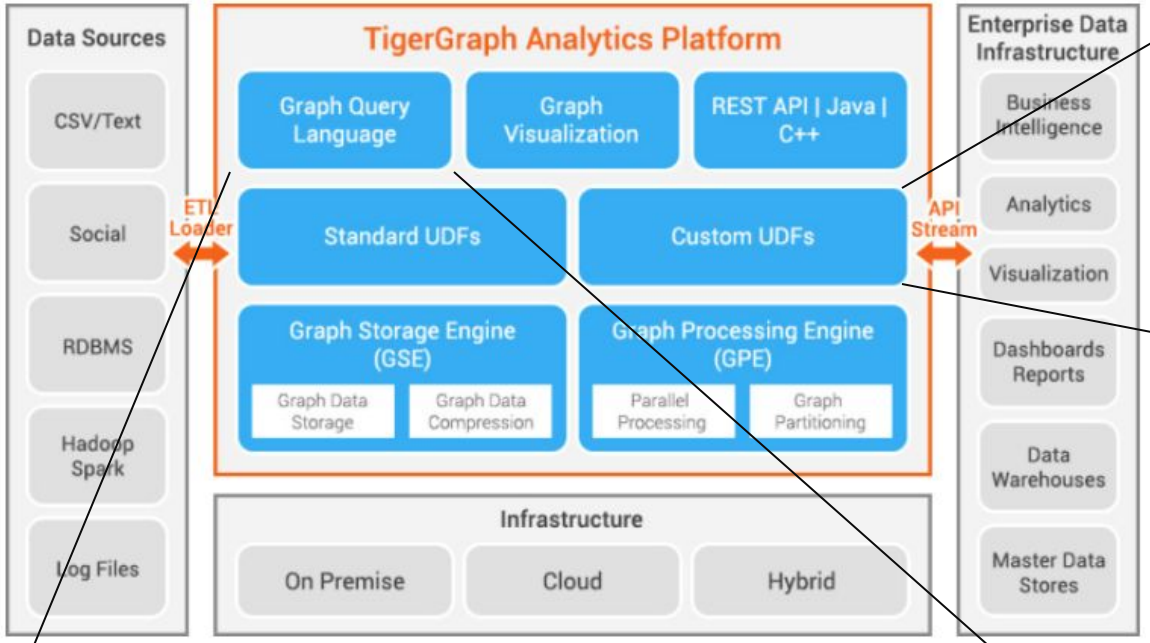
Cosine Similarity: Host Application

```
…
 for (unsigned int di = 0;di<deviceCount; di++){
          KVResType* l_res0_tmp;
          KVResType* l_res1_tmp;
          posix_memalign((void**)&l_res0_tmp, 4096, l_k * sizeof(KVResType));
          memset(l_res0_tmp, 0, l_k * sizeof(KVResType));
          posix_memalign((void**)&l_res1_tmp, 4096, l_k * sizeof(KVResType));
          memset(l_res1_tmp, 0, l_k * sizeof(KVResType));
          l_res0[di]=l_res0_tmp;
          l_res1[di]=l_res1_tmp;
          xfspatialSendMat(l_res0[di], l_k * sizeof(KVResType), 35, 1, di);
          xfspatialSendMat(l_res1[di], l_k * sizeof(KVResType), 32, 0, di);
          xfspatialTopKCosSim(appContext->m_vecX0, l_res0[di], l_n, l_m, l_k, 1, di);
          xfspatialTopKCosSim(appContext->m_vecX1, l_res1[di], l_n, l_m, l_k, 0, di);
      }
xfspatialExecuteKernelAsync(2, deviceCount);
…
```



libxilinxsimilarity.so

Integration with TigerGraph



<TG Install Dir>/dev/gdk/gsql/src/QueryUdf/ExprFunctions.hpp:

inline string udf_open_alveo(int mode) { … }
inline bool udf_close_alveo(int mode) { … }
inline bool udf_write_device(int mode) { … }
inline ListAccum<testResults>
udf_cosinesim_ss_alveo(ListAccum<int64_t>& patient_vector, uint64_t topK) { … }

libxilinxsimilarity.so (code to manage client requests)

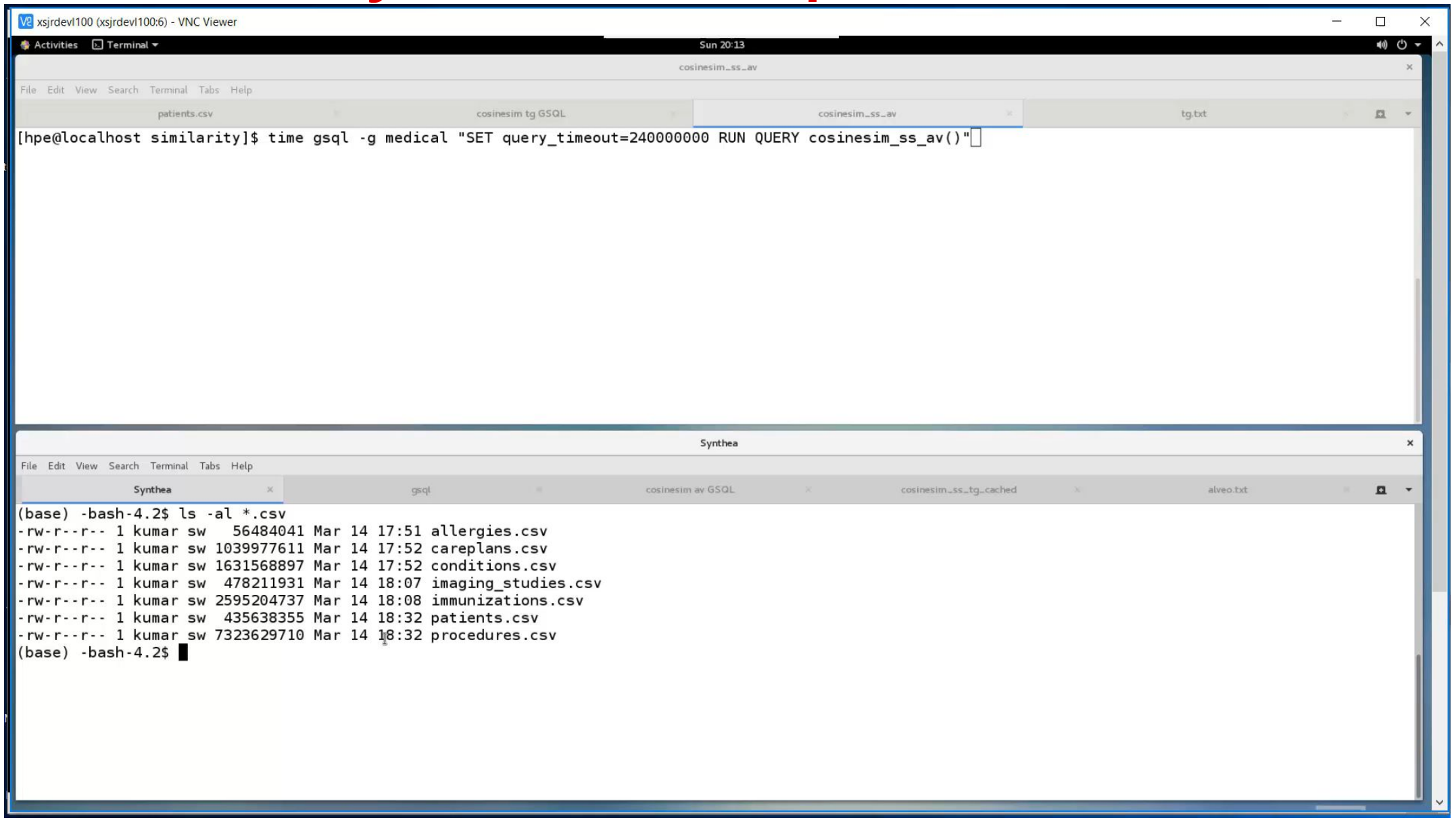
Xilinx Runtime (PCIE driver and card management)

similarity.gsql:

open_alveo() { …; udf_open_alveo(1); *…}*
load_alveo() {…; udf_write_device(1); *…}*
close_alveo() { …; udf_close_alveo(1); *…}*
cosinesim_ss_alveo(newPatient, topK) { …; udf_cosinesim_ss_alveo(*newPatient, topK); …}*

XILINX

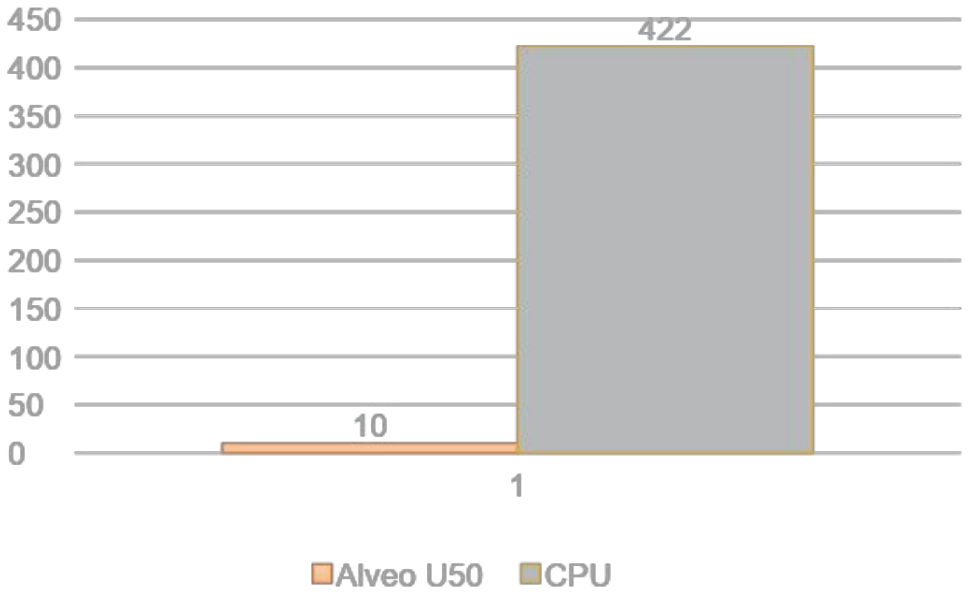
Demo: Similarity for 1.5 million patients



XILINX.

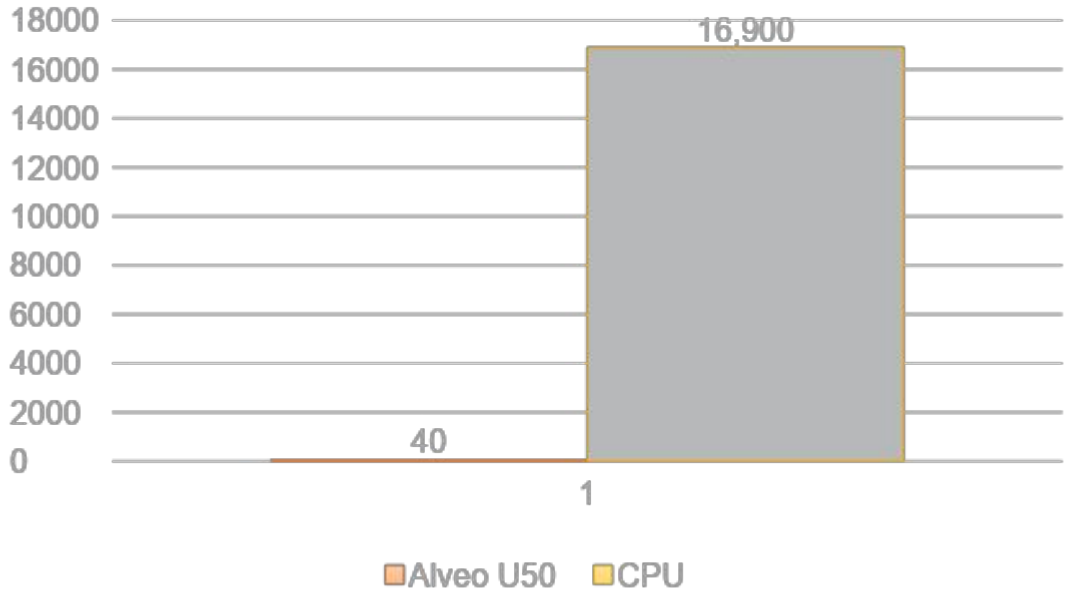
Time (milli-seconds) to get top 100 similar patients

40x faster than CPU



Using one Alveo U50

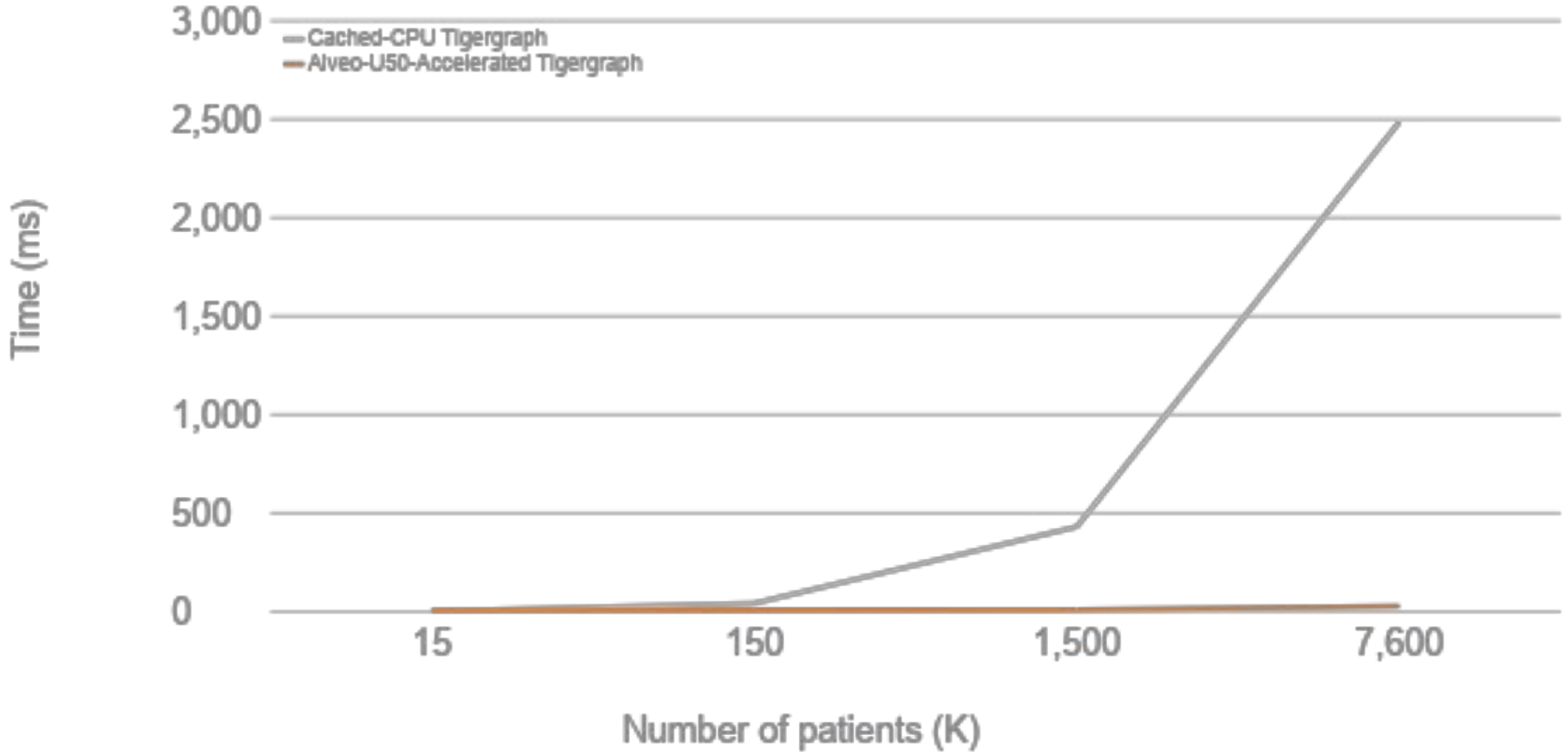
400x faster than CPU



Using 5 Alveo U50's

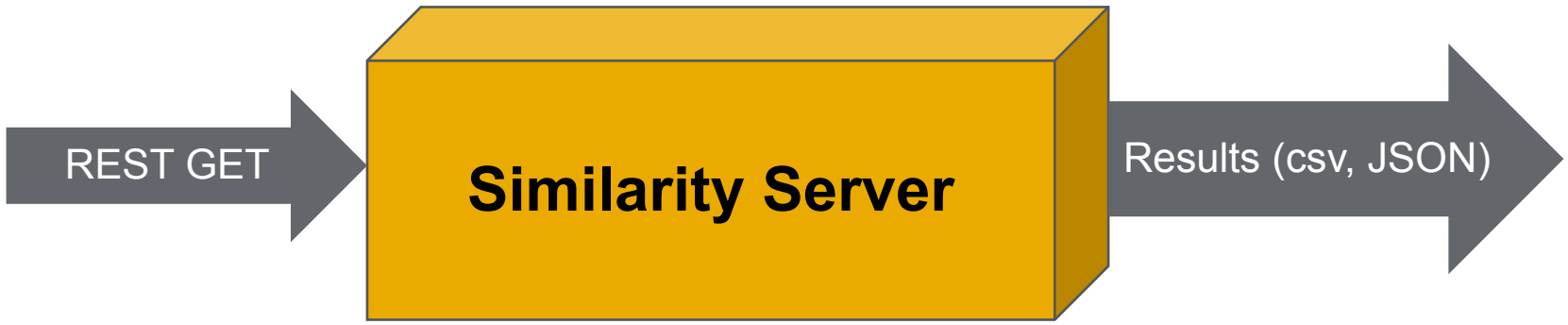
XILINX.

Scaling with number of patients



XILINX.

Three General REST Services to Support Similarity

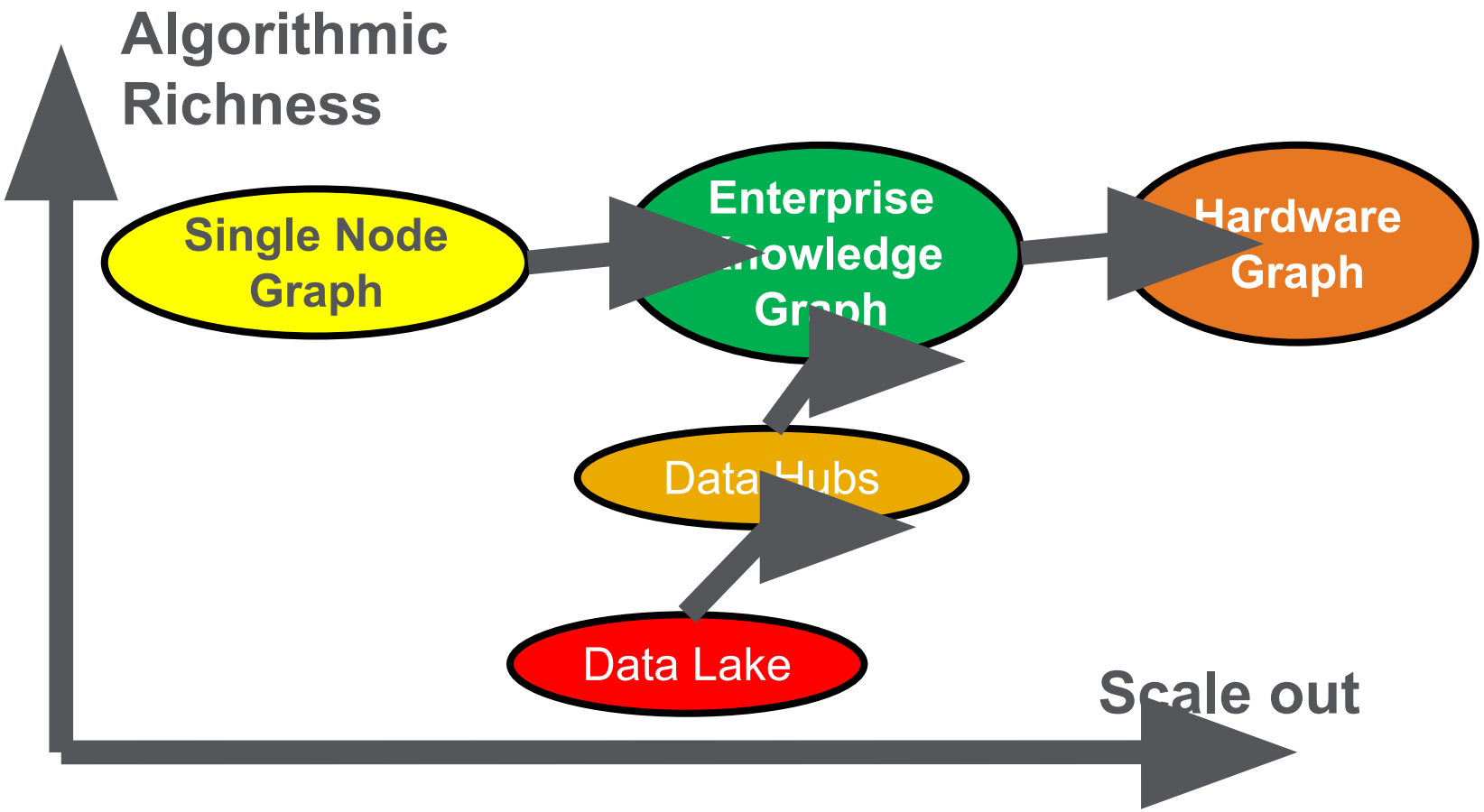


Bulk Upload Data: input - millions of vectors; output – success/failure code

Update Vector: input - vertex ID, 198 integers; output – success/failure code

Find Similar: input - vertex ID, 198 integers; output – 100 vertex IDs (64 bits)

Onward to the Hardware Graph!



Related Use Cases

Recommendation Engines for Healthcare

- For any person calling in for a recommended provider or senior living facility, can we find similar recommendations in the past?

Incident Reporting

- When trouble ticket is reported, what are the most similar problems and what were their solutions?

Errors in Log Files

- When there are are error messages in log files, how can we find similar errors and their solutions?

Learning Content

- Can we recommend learning content for employees that have similar goals?

Schema Mapping

- Automate the process of creating data transformation maps for new data to existing schemas