Universität Bremen

Future of Computer Architecture

GPUs, Quantum Computing, Neuromorphic Computing

Prof. Dr. Rolf Drechsler

Dr. Muhammad Hassan

M.Sc. Jan Zielasko

M.Sc. Milan Funck

Universität Bremen

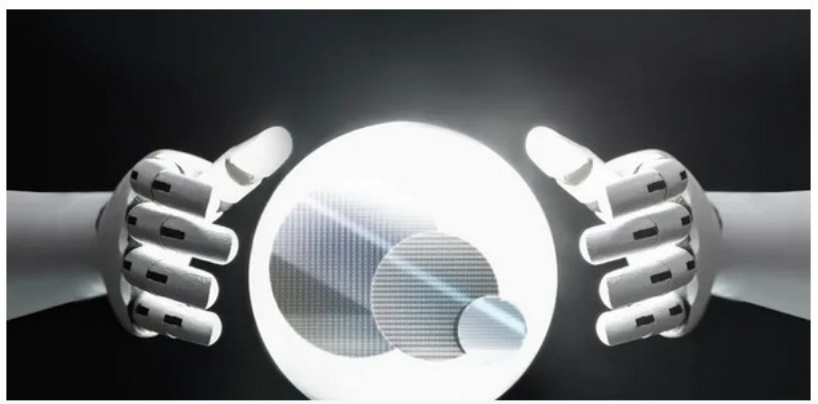
Sam Altman is talking to TSMC about AI chip venture

News By Anton Shilov published January 24, 2024

Altman is looking for alternatives to Nvidia GPUs.

Comments (2)

When you purchase through links on our site, we may earn an affiliate commission. Here's how it works.



World Business Markets Sustainability Legal Breakingviews Technology Investigations Mor

Exclusive: OpenAI builds first chip with Broadcom and TSMC, scales back foundry ambition

By **Krystal Hu**, **Fanny Potkin** and **Stephen Nellis**

October 30, 2024 4:30 PM GMT+1 · Updated 3 months ago

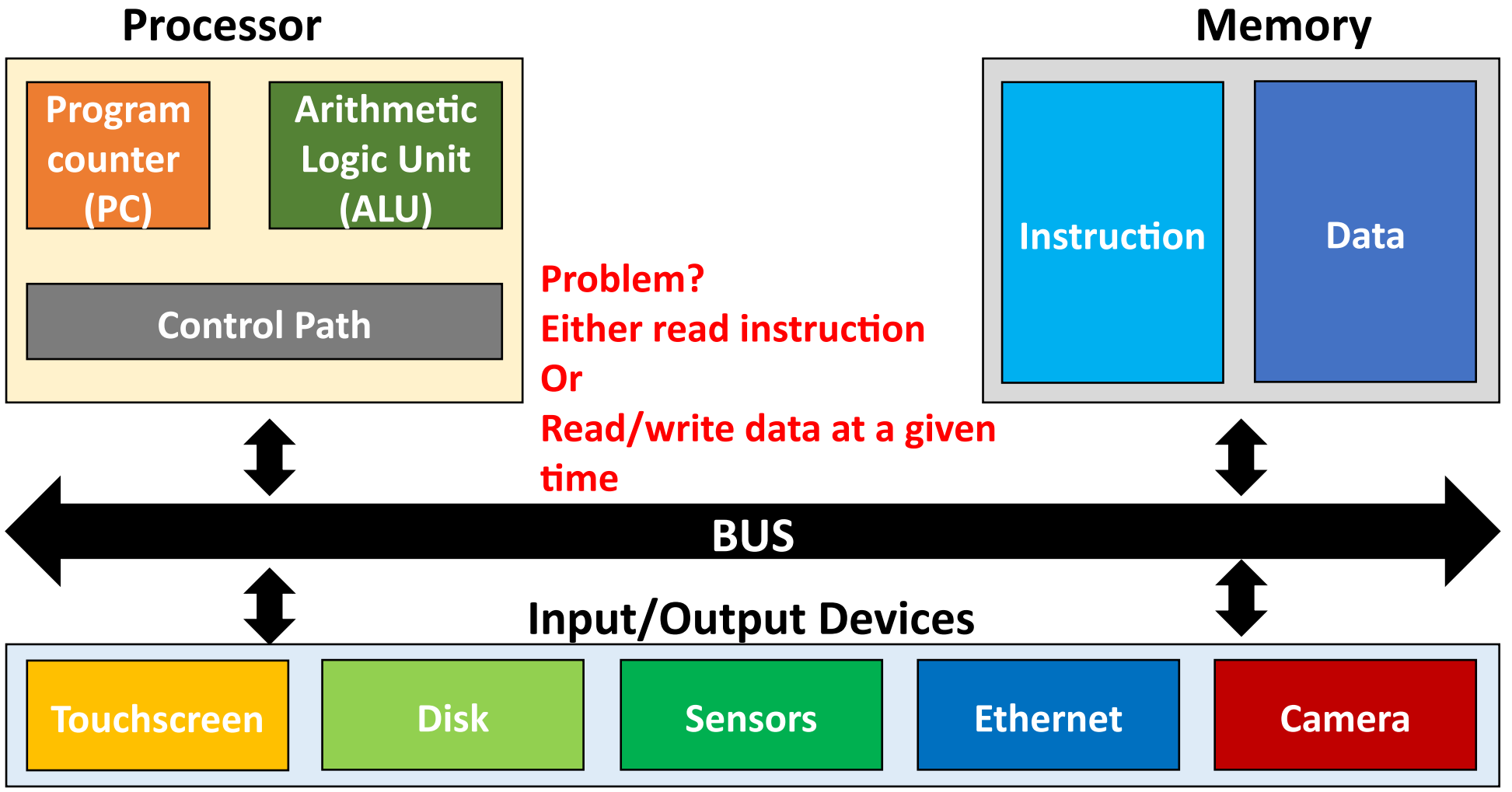
Aa

Universität Bremen

Today's Agenda

- Bottlenecks and Challenges
  - Parallelism
  - Speed
  - Energy
- GPUs
- Quantum Processors
- Neuromorphic Processor

Von Neumann Bottleneck



- Caching
- Prefetching
- Parllelism
- Accelerators

Vectorised by Vulphere from https://riscv.org/wp-content/uploads/2017/05/Tue1100-RISC-V-Foundation-Update.pdf

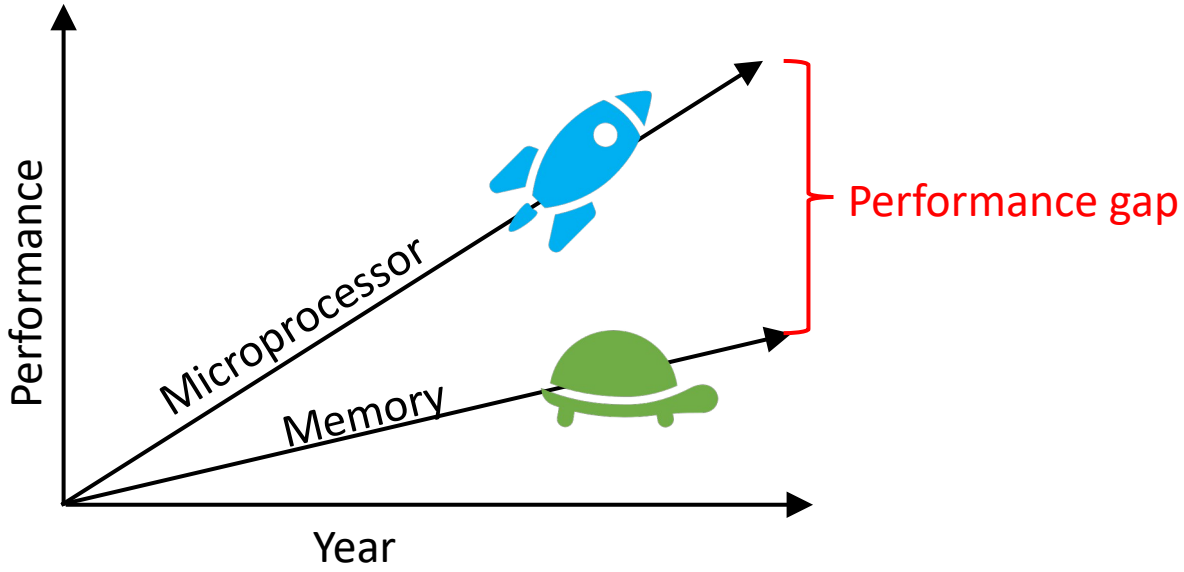
Universität Bremen

CPU: Three Walls to Serial Performance

- Memory Wall
- Power Wall
- Instruction Level Parallelism (ILP) Wall

CPU: Three Walls to Serial Performance

- Memory Wall
  - CPU clock is much faster than memory latency
  - Main memory is stored outside the CPU
  - Bottleneck is bandwidth between the two



Source: excellent article, "*The Many-Core Inflection Point for Mass Market Computer Systems*", by John L. Manferdelli, Microsoft Corporation
https://icl.utk.edu/ctwatch/quarterly/articles/2007/02/the-many-core-inflection-point-for-mass-market-computer-systems/2/index.html

CPU: Three Walls to Serial Performance

- Memory Wall
  - CPU clock is much faster than memory latency
  - Main memory is stored outside the CPU
  - Bottleneck is bandwidth between the two

AI and Memory Wall

Amir Gholami[1,2] Zhewei Yao[1] Sehoon Kim[1] Coleman Hooper[1] Michael W. Mahoney[1,2,3] Kurt Keutzer[1]

[1]University of California, Berkeley [2]ICSI [3]LBNL

***Abstract*—The availability of unprecedented unsupervised training data, along with neural scaling laws, has resulted in an unprecedented surge in model size and compute requirements for serving/training LLMs. However, the main performance bottleneck is increasingly shifting to memory bandwidth. Over the past 20 years, peak server hardware FLOPS has been scaling at 3.0×/2yrs, outpacing the growth of DRAM and interconnect bandwidth, which have only scaled at 1.6 and 1.4 times every 2 years, respectively. This disparity has made memory, rather than compute, the primary bottleneck in AI applications, particularly in serving. Here, we analyze encoder and decoder Transformer models and show how memory bandwidth can become the dominant bottleneck for decoder models. We argue for a redesign in model architecture, training, and deployment strategies to overcome this memory limitation.**

I. Introduction

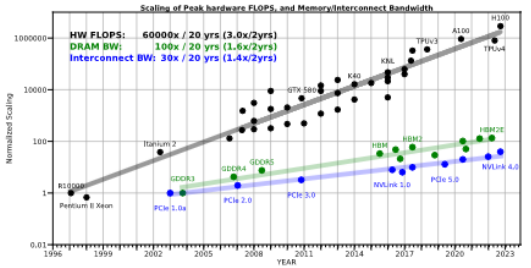


Fig. 1: *The scaling of the bandwidth of different generations of interconnections and memory, as well as the Peak FLOPS. As can be seen, the bandwidth is increasing very slowly. We are*

https://arxiv.org/pdf/2403.14123v1

CPU: Three Walls to Serial Performance

- Power Wall
  - Moore's law: transistor density increases over time
  - Dennard scaling: as transistors get smaller, so do power requirements
    - as transistors get smaller
      - they can operate with lower voltages
      - the power density remains constant
      - **voltage and current should be proportional to the linear dimensions of a transistor**
  - Current leakage: tradeoffs aren't equal – power per unit area is increasing
    - limited practical processor frequency to around 4 GHz since 2006

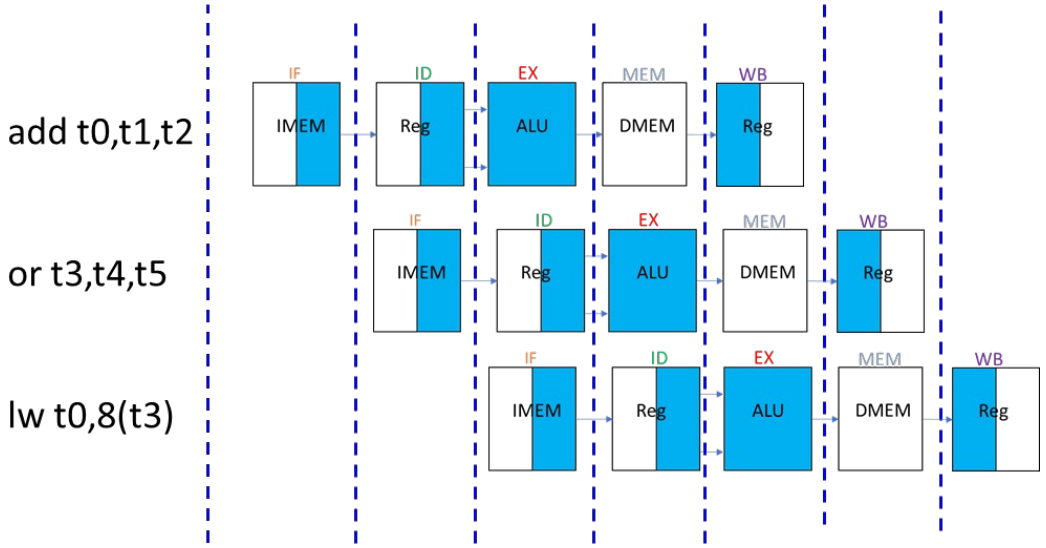
Why haven't clock speeds increased, even though transistors have continued to shrink?

Source: excellent article, "*The Many-Core Inflection Point for Mass Market Computer Systems*", by John L. Manferdelli, Microsoft Corporation
https://icl.utk.edu/ctwatch/quarterly/articles/2007/02/the-many-core-inflection-point-for-mass-market-computer-systems/2/index.html

Instruction Level Parallelism

- Instruction level parallelism (ILP) wall
  - CPU speed increases rely on finding parallelism within a single thread

pipelining



speculative execution

```
1)  int sign;
2)  if(x == NaN)
3)      sign = x;
4)  else if(x == 0)
5)      sign = 0;
6)  else
7)      sign = x/abs(x);
```

- (7) requires an arithmetic operation and is most likely
- Execute (7) while (2) and (4) are tested

out-of-order execution

```
1)  e = a + b;
2)  f = c + d;
3)  m = e * f;
```

- (3) is the only line with a dependence
- (1) and (2) can be executed in parallel

Universität Bremen

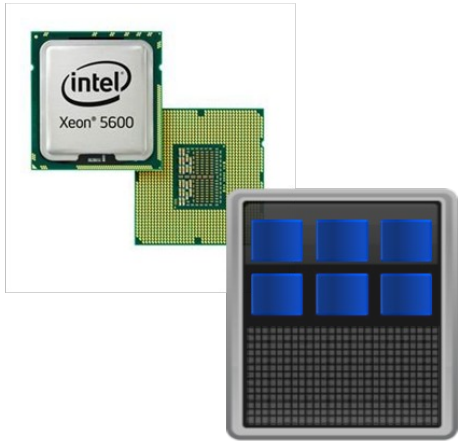
Today's Agenda

- Bottlenecks and Challenges
  - Parallelism
  - Speed
  - Energy
- GPUs
- Quantum Processors
- Neuromorphic Processor

CPUs vs GPUs

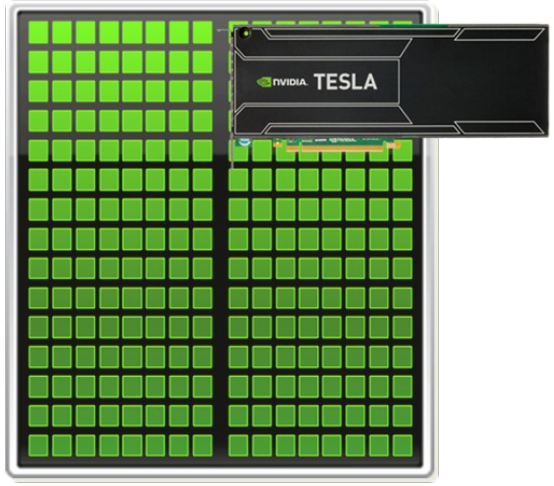
- Optimized to process a single sequence of instructions
- Extremely fast – clock speed reflects speed of operations
  - Not getting a lot faster
    - memory latency
    - power
    - sequential processing
- We're still very good at adding transistors
- Current trend: instead of doing something faster, do more things

CPU



CPUs consist of a few cores optimized for serial processing and general purpose calculations.

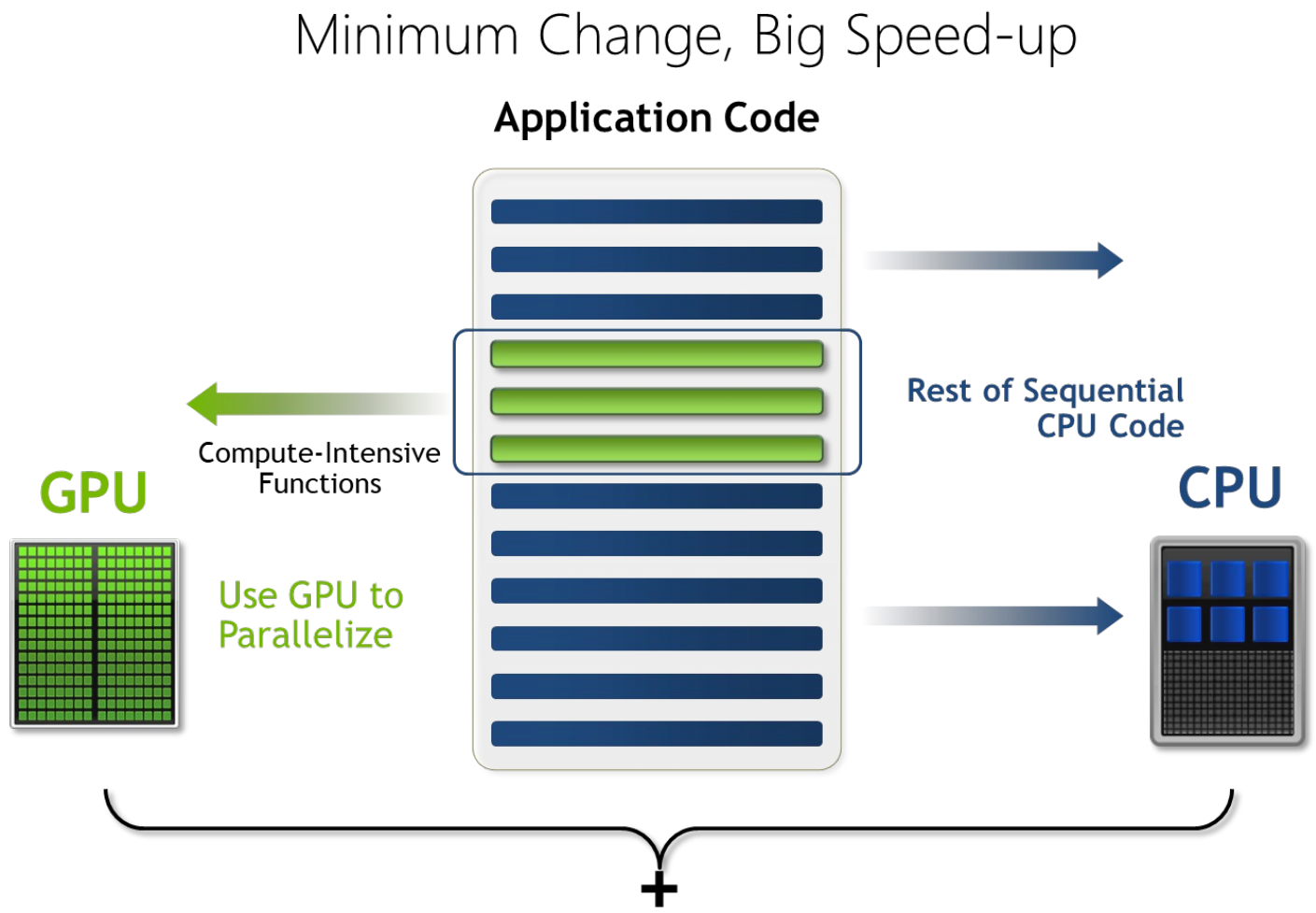
GPU



GPUs consist of hundreds or thousands of smaller, efficient cores designed for parallel performance. The hardware is designed for specific calculations.

https://www.researchgate.net/figure/CPU-vs-GPU-architecture-each-blue-square-represents-one-core_fig1_323281068

The Power of Two!



NVIDIA HGX Blackwell B200 vs NVIDIA HGX Hopper H200 GPUs

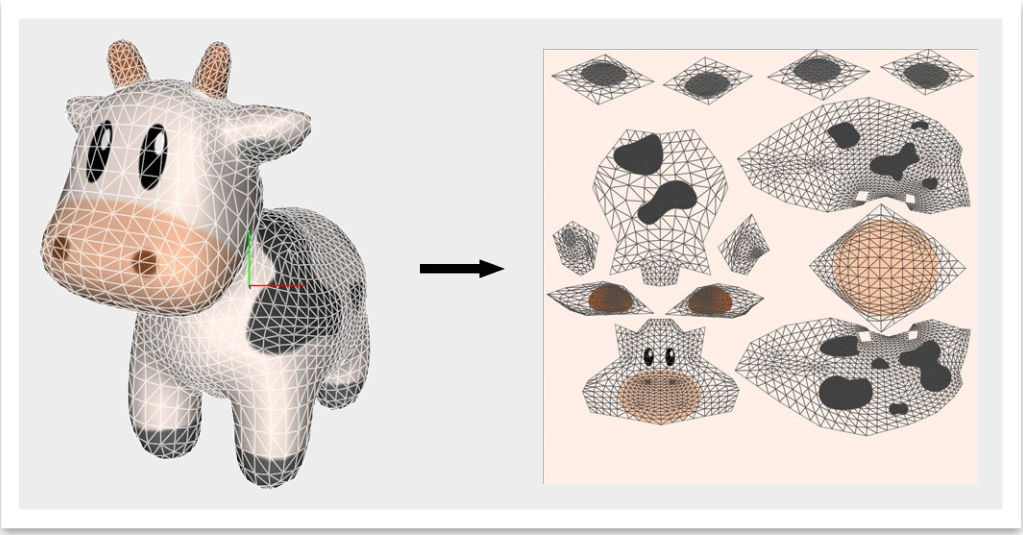
	HGX B200	HGX H200
Form Factor	8x NVIDIA B200 SXM	8x NVIDIA H200 SXM
FP8 Tensor Core	72 PFLOPS	32 PFLOPS
INT8 Tensor Core	72 PFOPS	32 PFOPS
FP16/BF16 Tensor Core	36 PFLOPS	16 PFLOPS
TF32 Tensor Core	18 PFLOPS	8 PFLOPS
FP32	640 TFLOPS	540 TFLOPS
FP64	320 TFLOPS	270 TFLOPS
FP64 Tensor Core	320 TFLOPS	540 TFLOPS
Memory	Up to 1.5TB	1.1TB HBM3
NVLink	Fifth generation	Fourth generation
NVIDIA NVSwitch™	Fourth generation	Third generation
NVSwitch GPU-to-GPU Bandwidth	1.8TB/s	900GB/s
Total Aggregate Bandwidth	14.4TB/s	7.2TB/s

https://aivres.com/blog/nvidia-hgx-blackwell-b200-vs-hgx-hopper-h200/

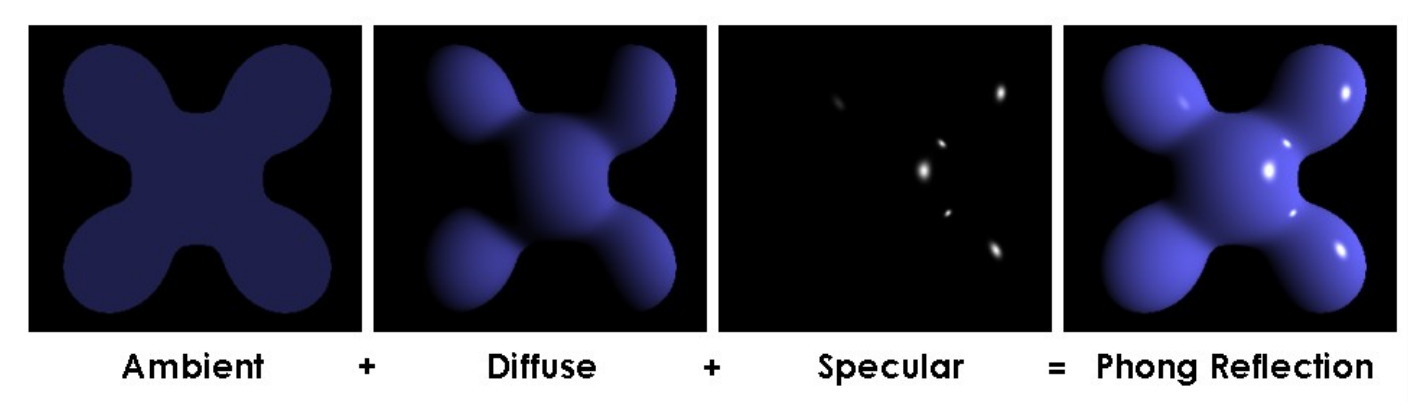
GPUs

Graphic Processing – Some History

- 1990s: Real-time 3D rendering for video games were becoming common
  - Doom, Quake, Descent, … (Nostalgia!)
- 3D graphics processing is immensely computation-intensive



Texture mapping



Shading

Warren Moore, “Textures and Samplers in Metal,” Metal by Example, 2014

Gray Olsen, “CSE 470 Assignment 3 Part 2 - Gourad/Phong Shading,” grayolsen.com, 2018

Graphic Processing – Some History

- Before 3D accelerators (GPUs) were common
  - CPUs had to do all graphics computation, while maintaining framerate!
  - Many tricks were played

Quake III arena (1999) : “Fast inverse square root” magic!

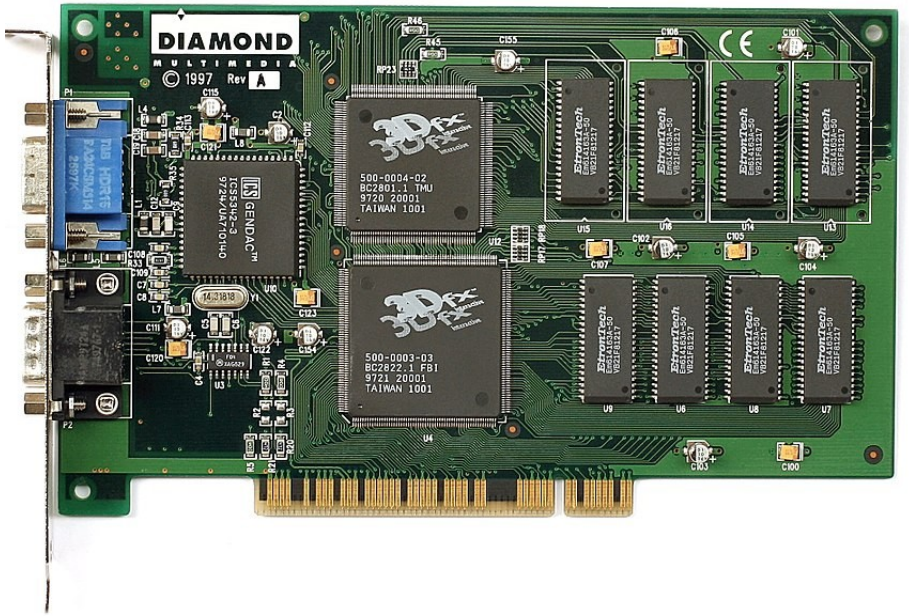
```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

Introduction of 3D Accelerator Cards

- Much of 3D processing is short algorithms repeated on a lot of data
  - pixels, polygons, textures, …
- Dedicated accelerators with simple, massively parallel computation

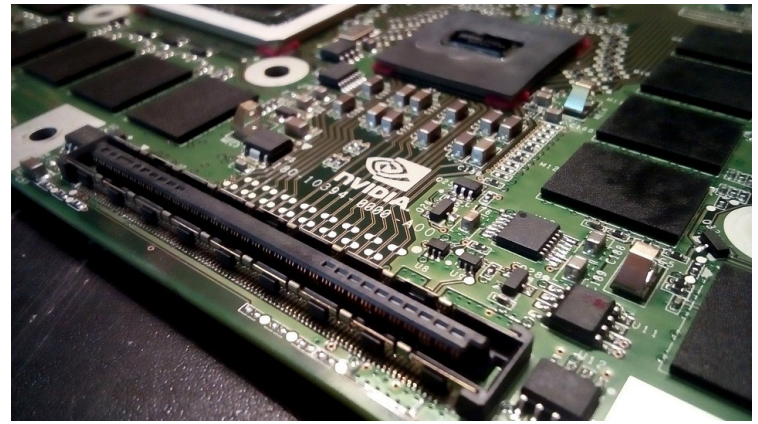


A Diamond Monster 3D, using the Voodoo chipset (1997)
(Konstantin Lanzet, Wikipedia)

17

GPU Evolution

- GPUs originally developed as specialized hardware for graphics computation
  - Now used as programmable accelerators for highly data-parallel workloads
- GPU hardware evolution closely tied to evolution in usage patterns
  - Desire for improved visual effects driven by games
  - More realism, more effects, more screen resolution, more frames per second
- Originally a fixed-function pipeline, programmability was added gradually to many stages.
  - Now the bulk of the GPU is a programmable data-parallel architecture.
  - Some fixed-function hardware remains for graphics.
  - New hardware being added to cope with modern workloads, e.g., machine learning



GPU Design Principles

- CPU design is about making a single thread run as fast as possible.
  - Pipeline stalls and memory accesses are expensive in terms of latency.
  - So increased logic was added to reduce the probability/cost of stalls.
  - Use of large cache memories to avoid memory misses
- GPU design is about maximizing computation throughput.
  - Individual thread latency not considered important
  - GPUs avoid much of the complex CPU pipeline logic for extracting ILP.
  - Instead, each thread executes on a relatively simple core with performance obtained through parallelism.
    - Single instruction, multiple threads
- Computation hides memory and pipeline latencies.
- Wide and fast bandwidth-optimized memory systems

Universität Bremen

GPU Computing – The Basic Idea

- The GPU is linked to the CPU by a reasonably fast connection
- The idea is to use the GPU as a co-processor
  - Farm out big parallel tasks to the GPU
  - Keep the CPU busy with the control of the execution and “corner” tasks

SIMT vs SIMD

```
for(i=0;i<n;++i) a[i]=b[i]+c[i];
```

SIMD

```
void add(uint32_t *a, uint32_t *b, uint32_t *c, int n) {
  for(int i=0; i<n; i+=4) {
    //compute c[i], c[i+1], c[i+2], c[i+3]
    uint32x4_t a4 = vld1q_u32(a+i);
    uint32x4_t b4 = vld1q_u32(b+i);
    uint32x4_t c4 = vaddq_u32(a4,b4);
    vst1q_u32(c+i,c4);
  }
}
```

SIMT

```
__global__ void add(float *a, float *b, float *c) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  a[i]=b[i]+c[i]; //no loop!
}
```

The idea is that the CPU spawns a thread per element, and the GPU then executes those threads. Not all of the thousands or millions of threads actually run in parallel, but many do

All threads running on the cores of an SM at a given cycle are executing the same instruction – hence Single Instruction, Multiple Threads. However, each thread has its own registers, so these instructions process different data

SIMT vs SIMD

SIMD + multithreading

SIMT

- Single instruction, multiple threads
- Takes advantage of data-level parallelism
- Can be considered a constrained form of multithreading
- Many threads each with their own state
  - Operating on scalar registers
  - With their own local memory

GPU

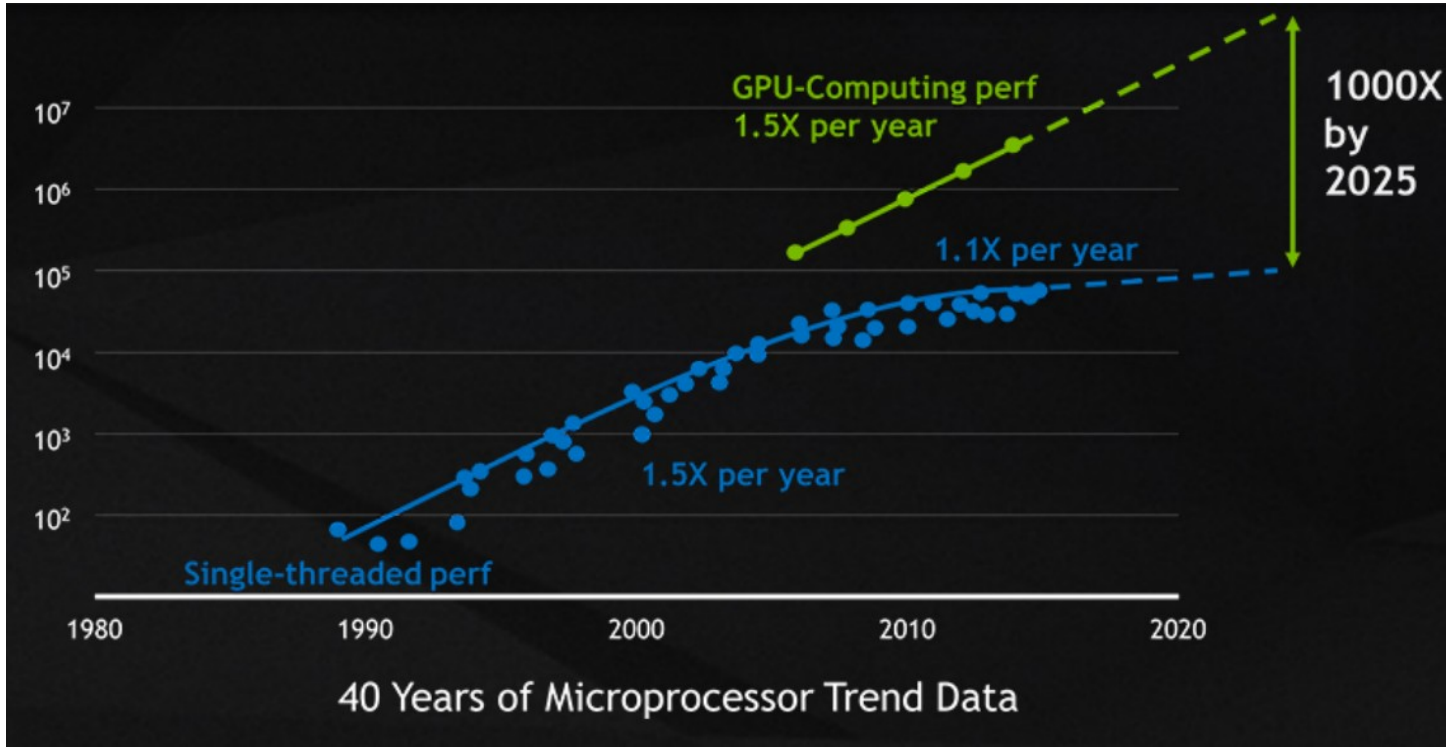
SIMD

- Single instruction, multiple data
- Takes advantage of data-level parallelism
- Can be considered a constrained form of vector processing
- One thread operating on vector registers

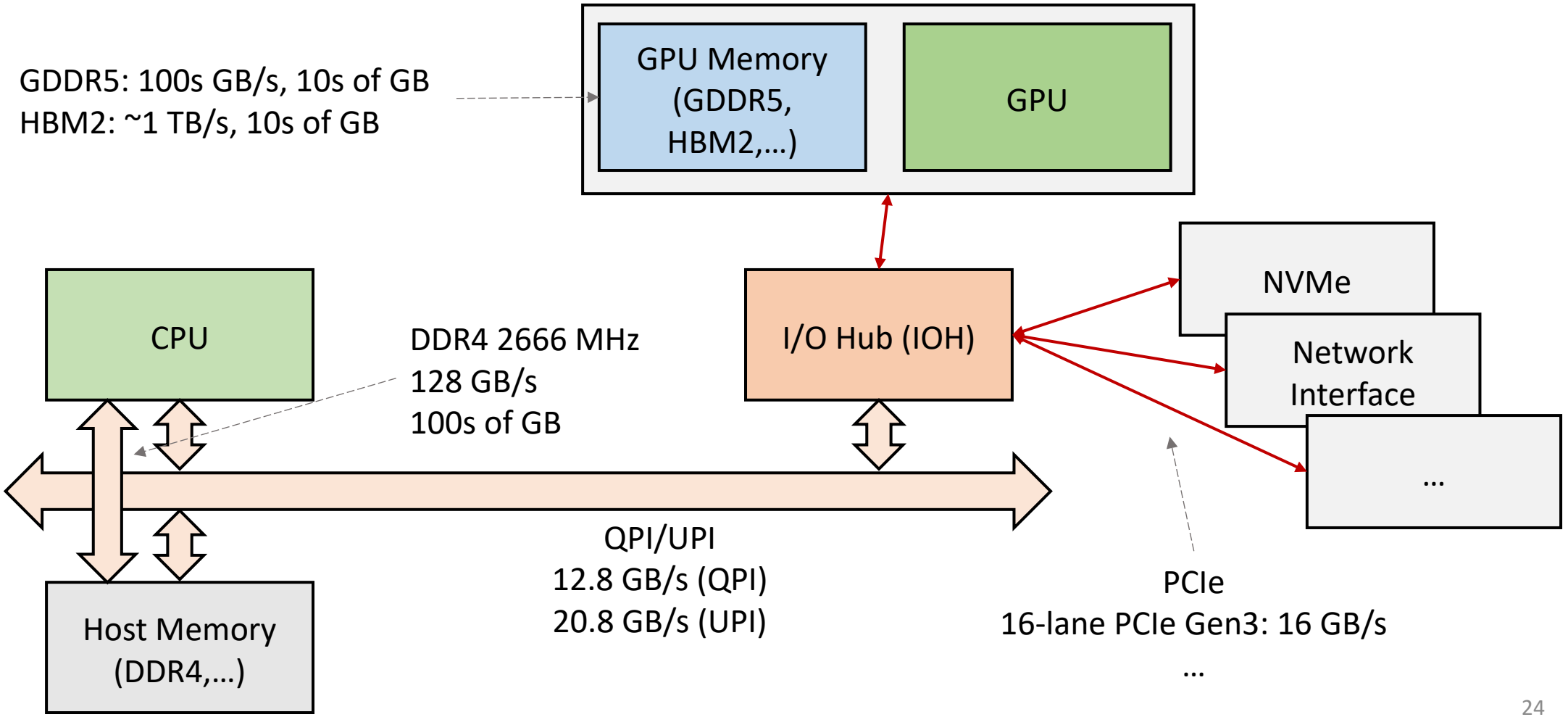
CPU

Peak Performance vs. CPU

	Throughput	Power	Throughput/Power
Intel Skylake	128 SP GFLOPS/4 Cores	100+ Watts	~1 GFLOPS/Watt
NVIDIA V100	15 TFLOPS	200+ Watts	~75 GFLOPS/Watt

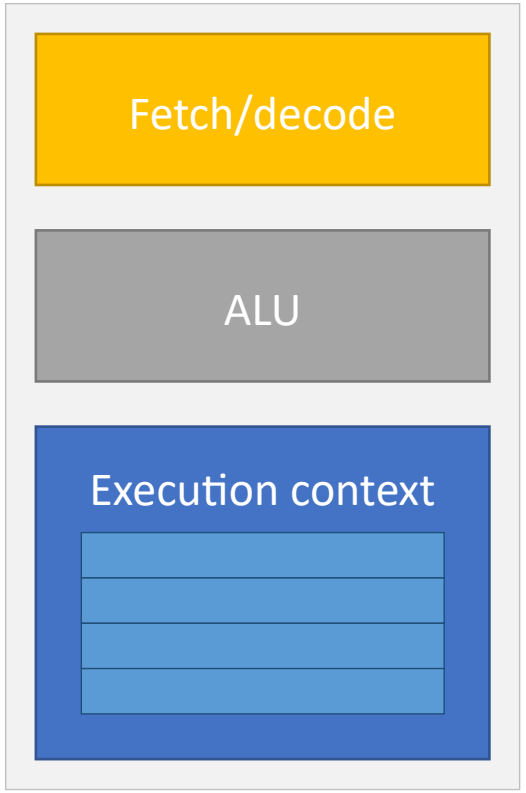


System Architecture Snapshot With a GPU (2020)



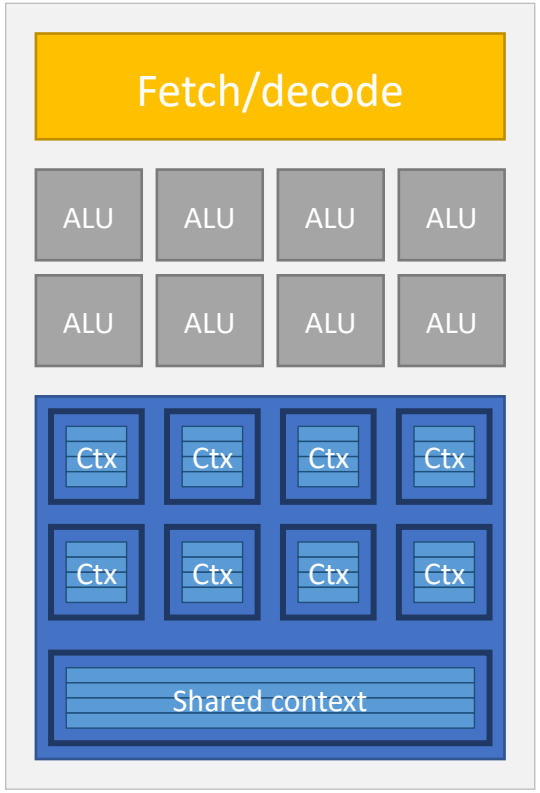
One Processing Element

- A single thread runs on a simple processing element.
- Short pipeline
- The execution context consists of the thread's state.
  - E.g., registers and local memory
- But fetch and decode are costly.



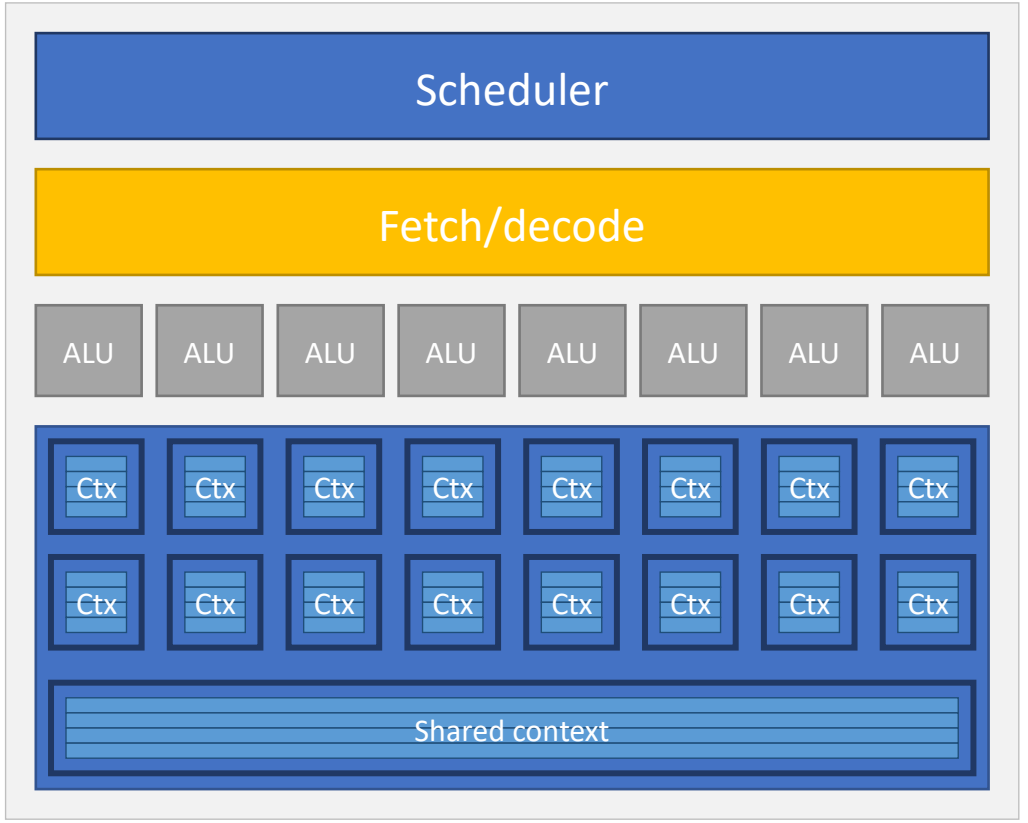
Multiple Processing Elements

- SIMT execution of threads
- Cost of fetch and decode spread across all threads in a work group (OpenCL terminology)
- Each thread has its own context.
  - And some shared context
- Multiple functional units for parallelism
  - One per thread for cheap units (e.g., simple ALU)
  - Fewer expensive units than threads (e.g., sqrt)
- However, stalls are costly.



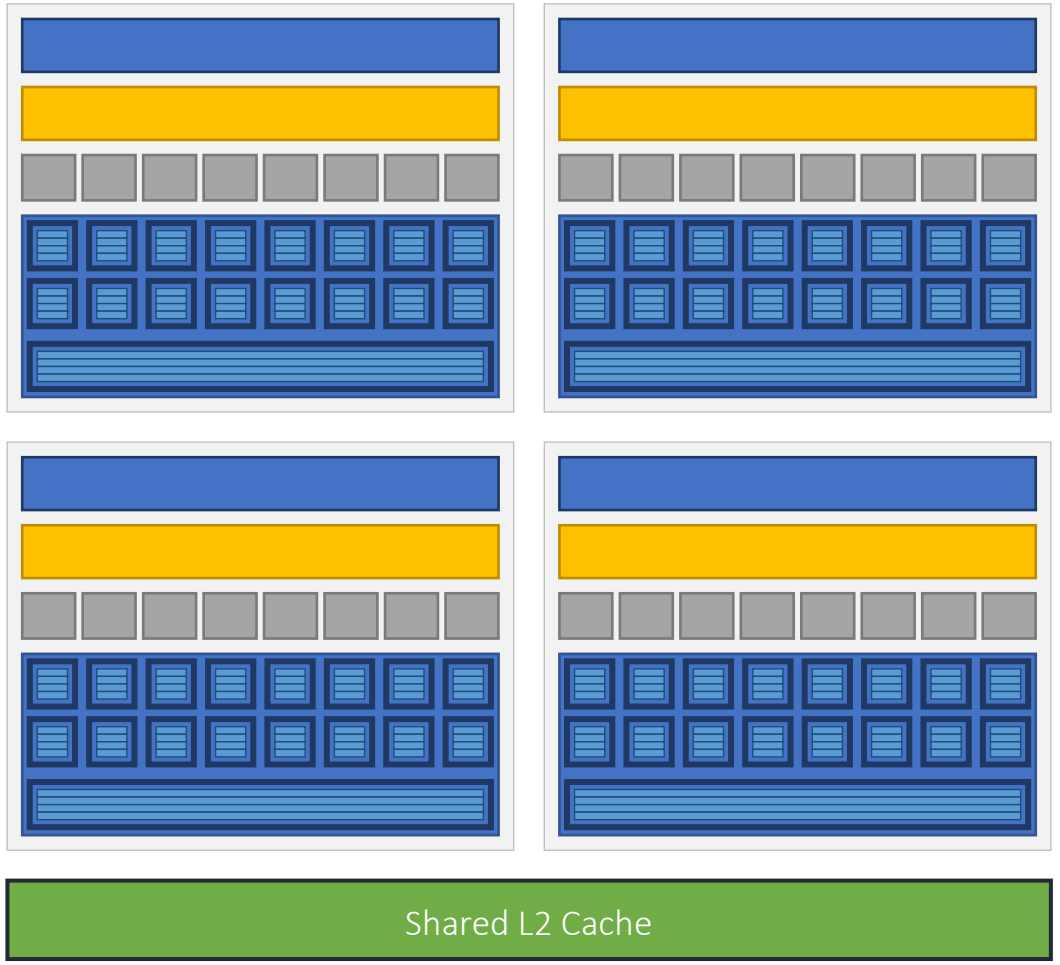
Minimizing Stalls

- Include support for multiple work groups
  - Each is independent of all others.
  - And this is guaranteed by the compiler.
  - Which means there is no fixed ordering of groups.
- A scheduler chooses work groups to run.
  - Maintains a list of ready work groups
  - Makes a choice each cycle
  - Some GPUs can schedule more than one work group per cycle.
  - Hides latency when a work group stalls
- This system is sometimes called a shader core.



Scaling Out

- Multiple instances of each shader core provided together
  - Each independent of the others
  - Each processes a subset of the work groups
- Massively increases parallelism
- Memory hierarchy provided, too
  - Shared L2 cache reduces memory bandwidth requirements.
  - Smaller caches local to each multithreaded core



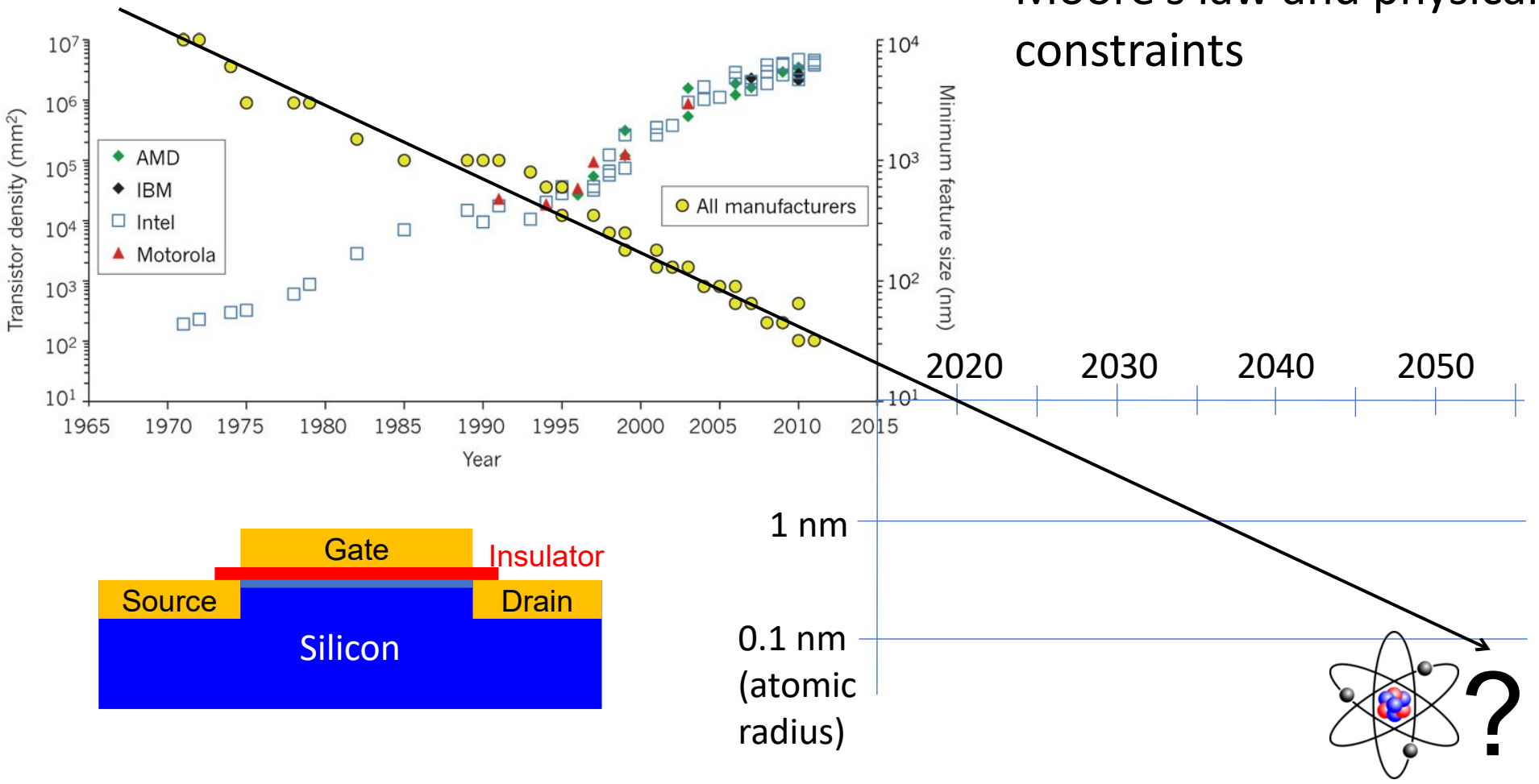
Universität Bremen

Today's Agenda

- Bottlenecks and Challenges
  - Parallelism
  - Speed
  - Energy
- GPUs
- Quantum Processors
- Neuromorphic Processor

Motivation: Future of Computers

Moore's law and physical constraints

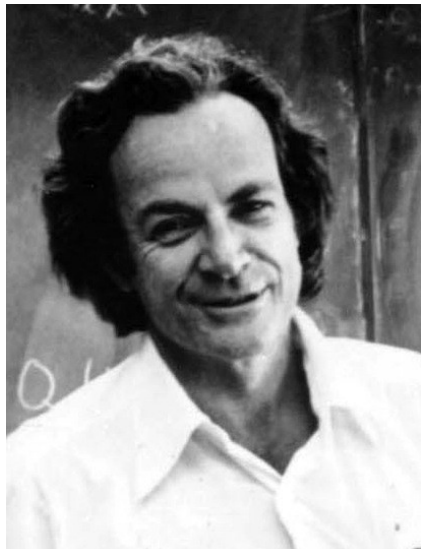


Moore, G. E. *Electronics* **8**, 114–117 (1965).
Image from: Ferain, I. *et al.*, *Nature* **479**, 310–316 (2011).

Limitations of Classical Computers

- RSA encryption (2048-bit)
  - 100,000 computers in parallel
  - 3 GHz processors
  - Factoring would take longer than the age of the universe

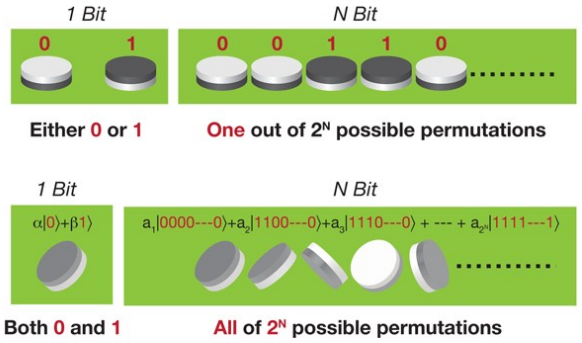
Such limitations in computational power underscore the need for advancements in computing methods and the exploration of alternative computing paradigms.

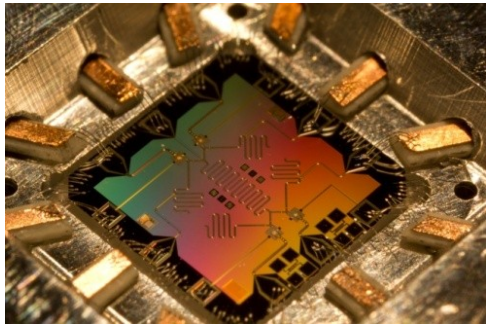


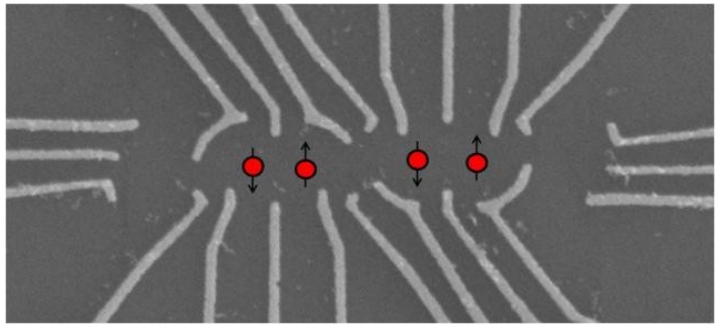
- Quantum Simulation: inefficient with classical computers
  - Feynman: why not use quantum mechanics for computation?

Outline

- What are quantum computers?
  - Classical vs. quantum bits
- Problems suited to quantum computation







Standard Computers

- Classical bits + Logic and Memory = Computer
  - 0
  - 1

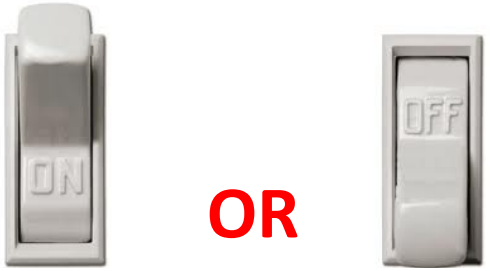
+

=

Quantum Bits (qubits)

Classical Bits

- Can be only 0 or 1



OR

Qubits

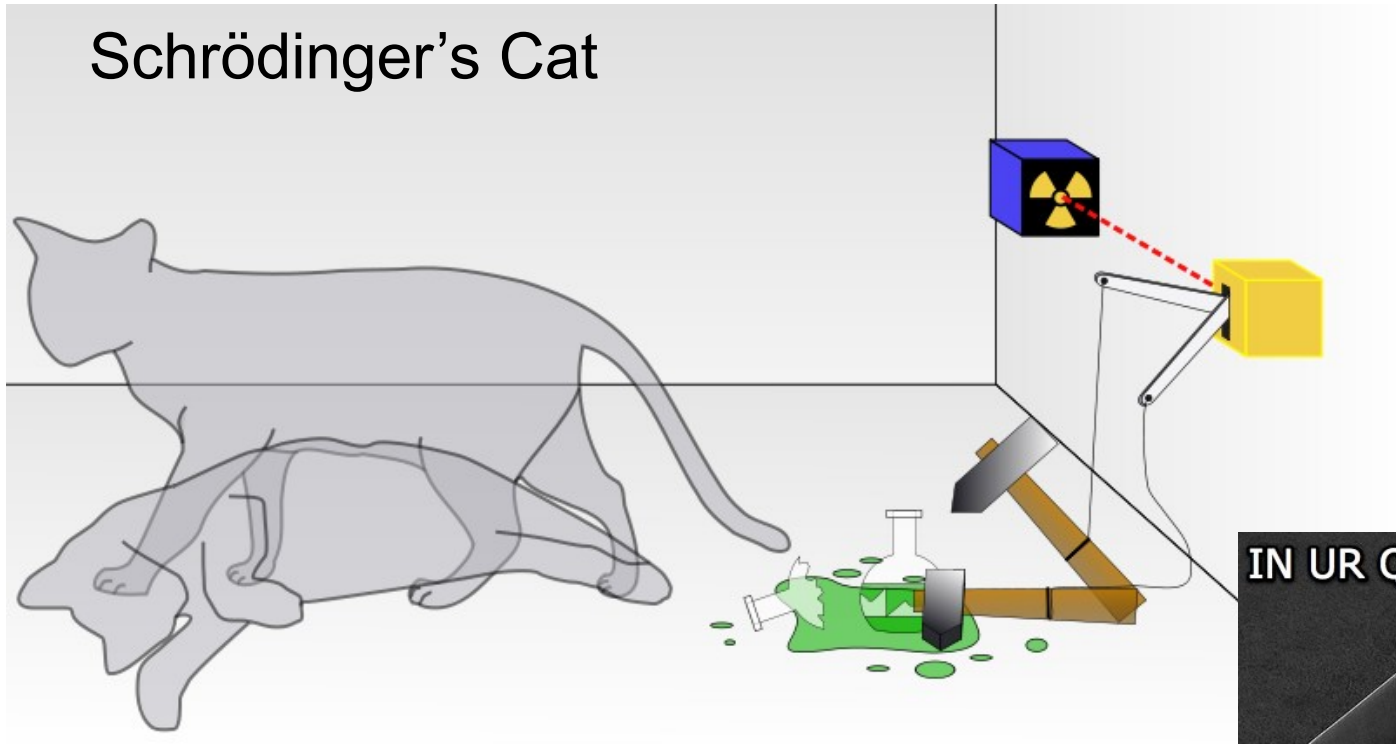
- Superposition of both 0 and 1

AND

- Any quantum two-level system can act as a qubit, e.g.
  - Atoms
  - Spins

http://www.quantummadesimple.com/

Paradox: Schrödinger’s Cat



Cat would be in a ‘superposition’ of alive and dead at the same time!

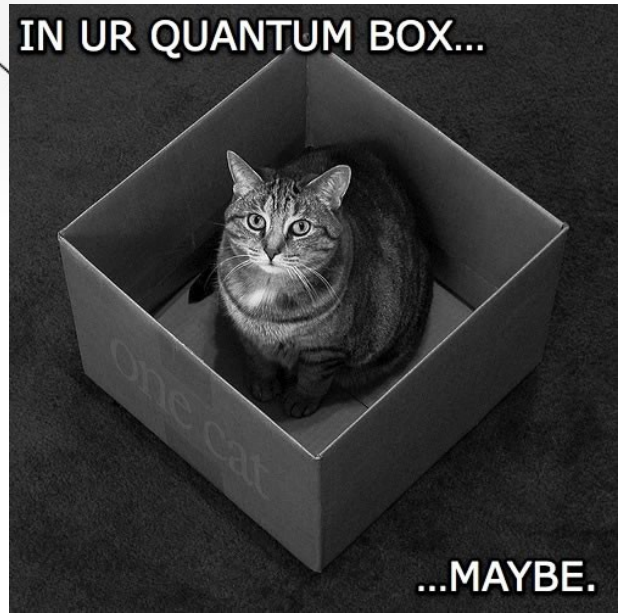


Image from: http://en.wikipedia.org/wiki/Schrödinger’s_cat

Quantum Superposition

https://en.wikipedia.org/wiki/File:Quantum_superposition_of_states_and_decoherence.ogv, Jubobroff J. Bobroff

How Can You Benefit?

- If measurement changes the answer, how can you fully utilize the 2^N states?
- The key: use ‘hidden’ information with a clever algorithm
- Still, in some cases, qubits still pay off!

Example: f(x)={0,1} for x = 0 or 1

- Consider f(x)
  - Input: either 0 or 1
  - Output: either 0 or 1

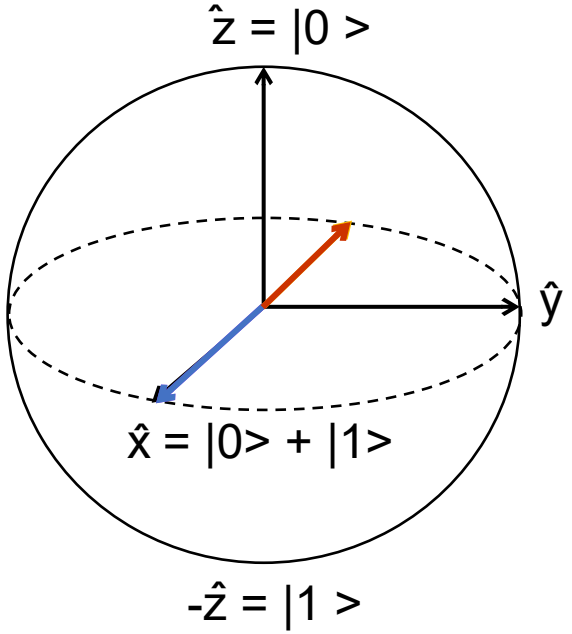
- Classically, you need at least two evaluations to determine if f(0) = f(1)

- Using quantum gates, you need only one!

Quantum gates and the inherent properties of qubits allow for certain computations to be performed more efficiently than with classical bits, which is a foundational concept in the field of quantum computing

Deutsch Algorithm

1. Start with states along +x and -x:
   - [|0> + |1>][|0> - |1>]
2. Perform y xor f(x)
   - [|0> + |1>] [(|0> xor |f(x)>) - (|1> xor |f(x)>)]
   - $(-1)^{f(x)}$[|0> + |1>][|0> - |1>]
     - For f(0) = f(1): ±[|0> + |1>][|0> - |1>]
     - For f(0) ≠ f(1): ±[|0> - |1>][|0> - |1>]

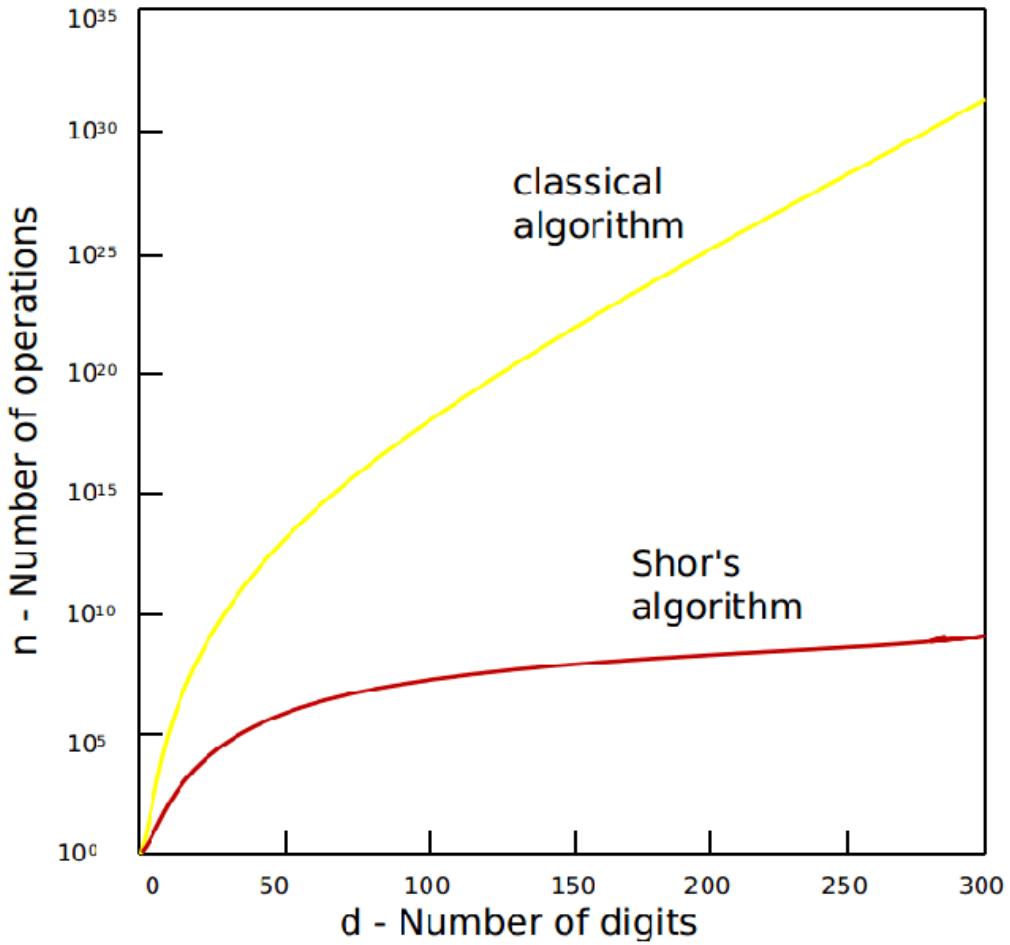


XOR	0	1
0	0	1
1	1	0

$$\frac{[|0\rangle + |1\rangle]}{}$$

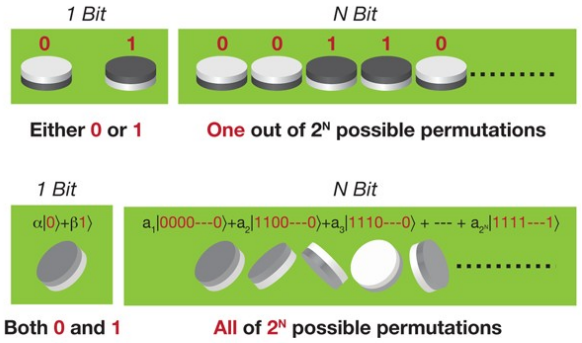
$$\frac{[|0\rangle - |1\rangle]}{}$$

Shor Algorithm

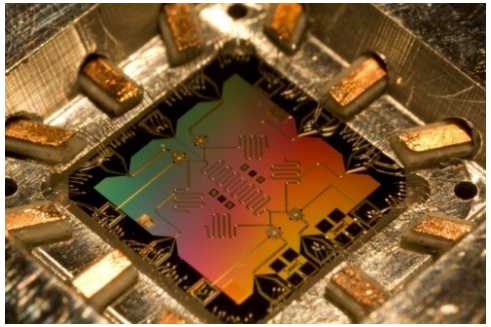


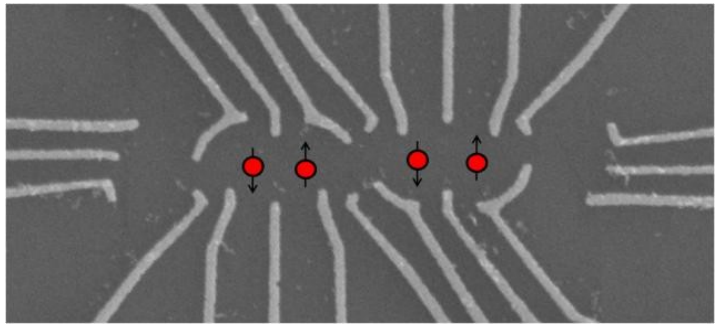
Outline

- What are quantum computers?
  - Classical vs. quantum bits



- Problems suited to quantum computation

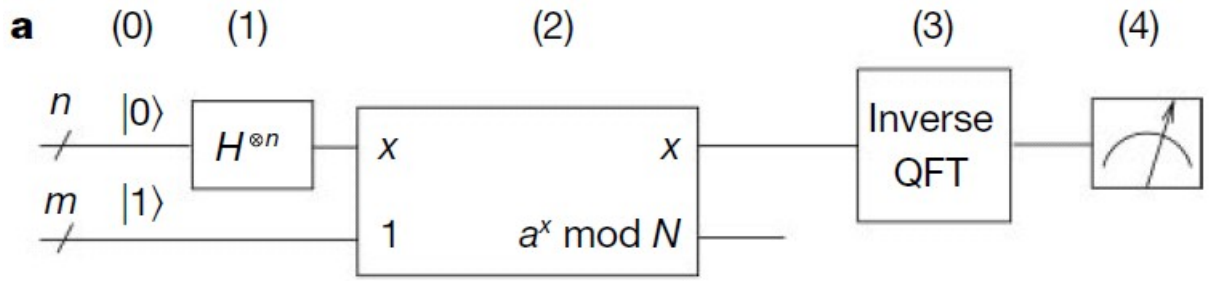




RSA encryption

- Recall: 100,000 computers in parallel operating at 3 GHz, RSA factoring would take longer than the age of the universe

- Quantum computation allows a speedup from exponential to polynomial time!
  - A single 1 MHz quantum computer could do it in about 1 minute

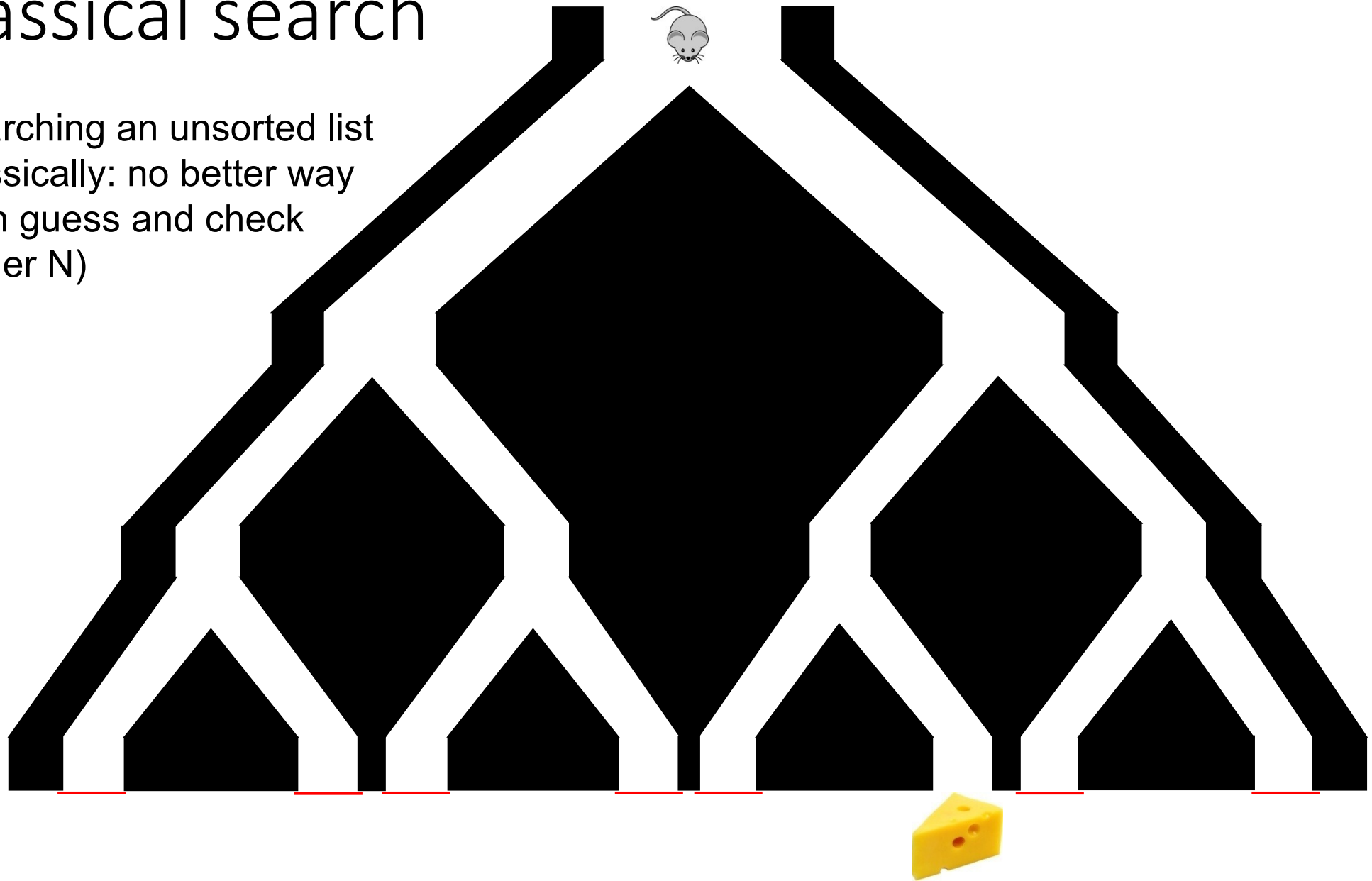


Shor's Algorithm

L. M. K. Vandersypen *et al.*, *Nature* **414**, 883 (2001).

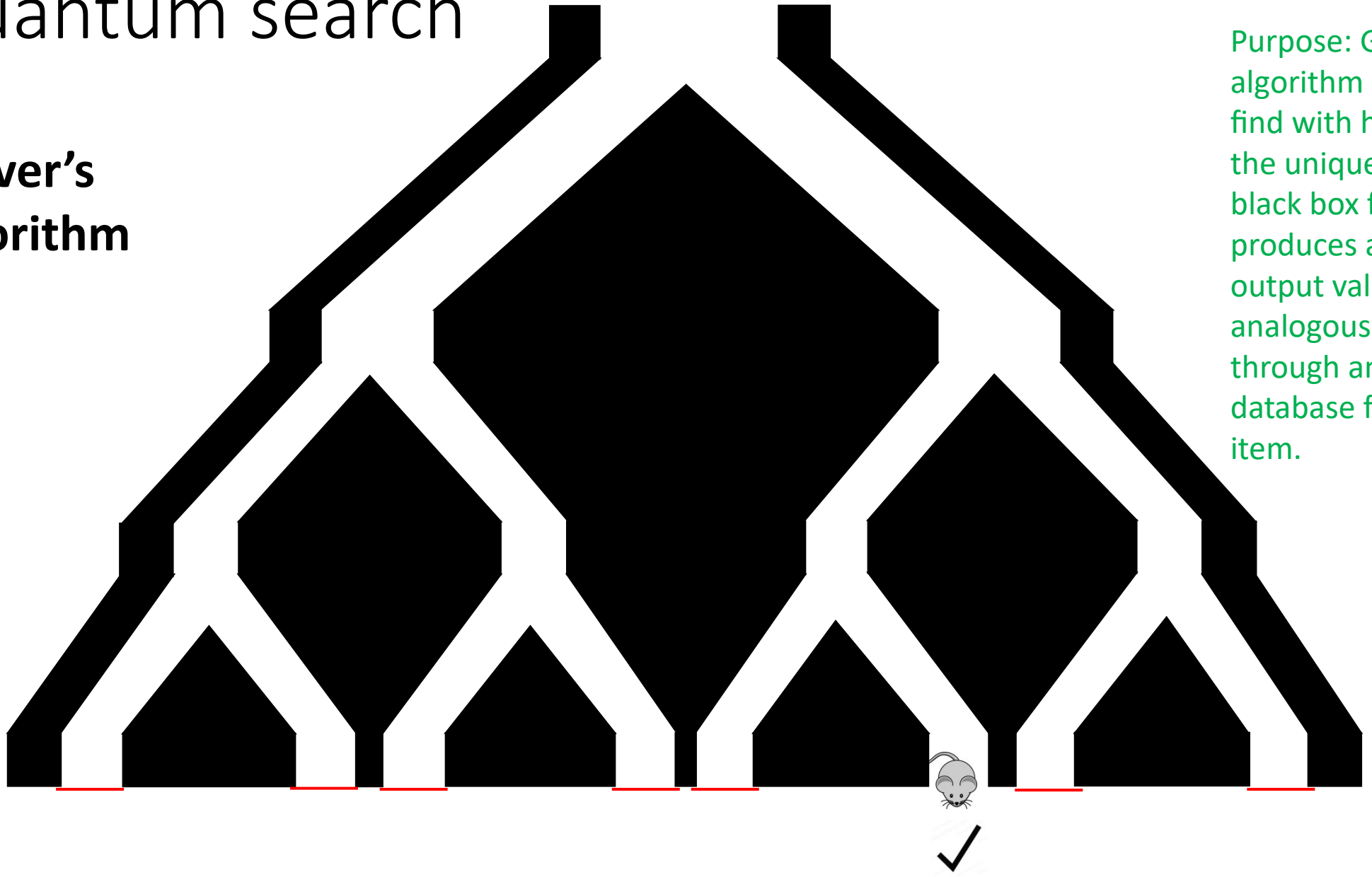
Classical search

Searching an unsorted list classically: no better way than guess and check (order N)



Quantum search

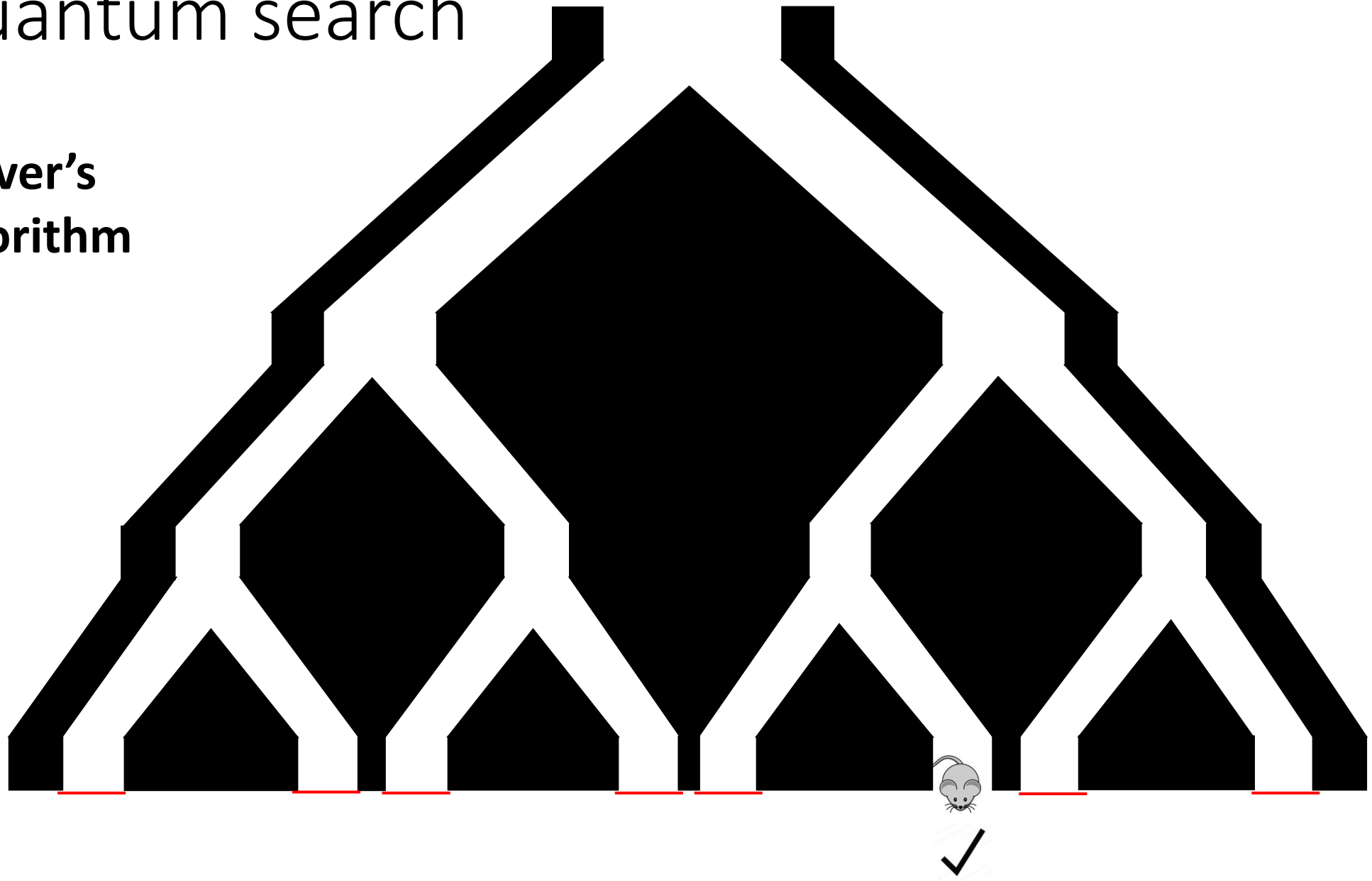
Grover’s Algorithm



Purpose: Grover's algorithm is designed to find with high probability the unique input to a black box function that produces a particular output value. This is analogous to searching through an unsorted database for a particular item.

Quantum search

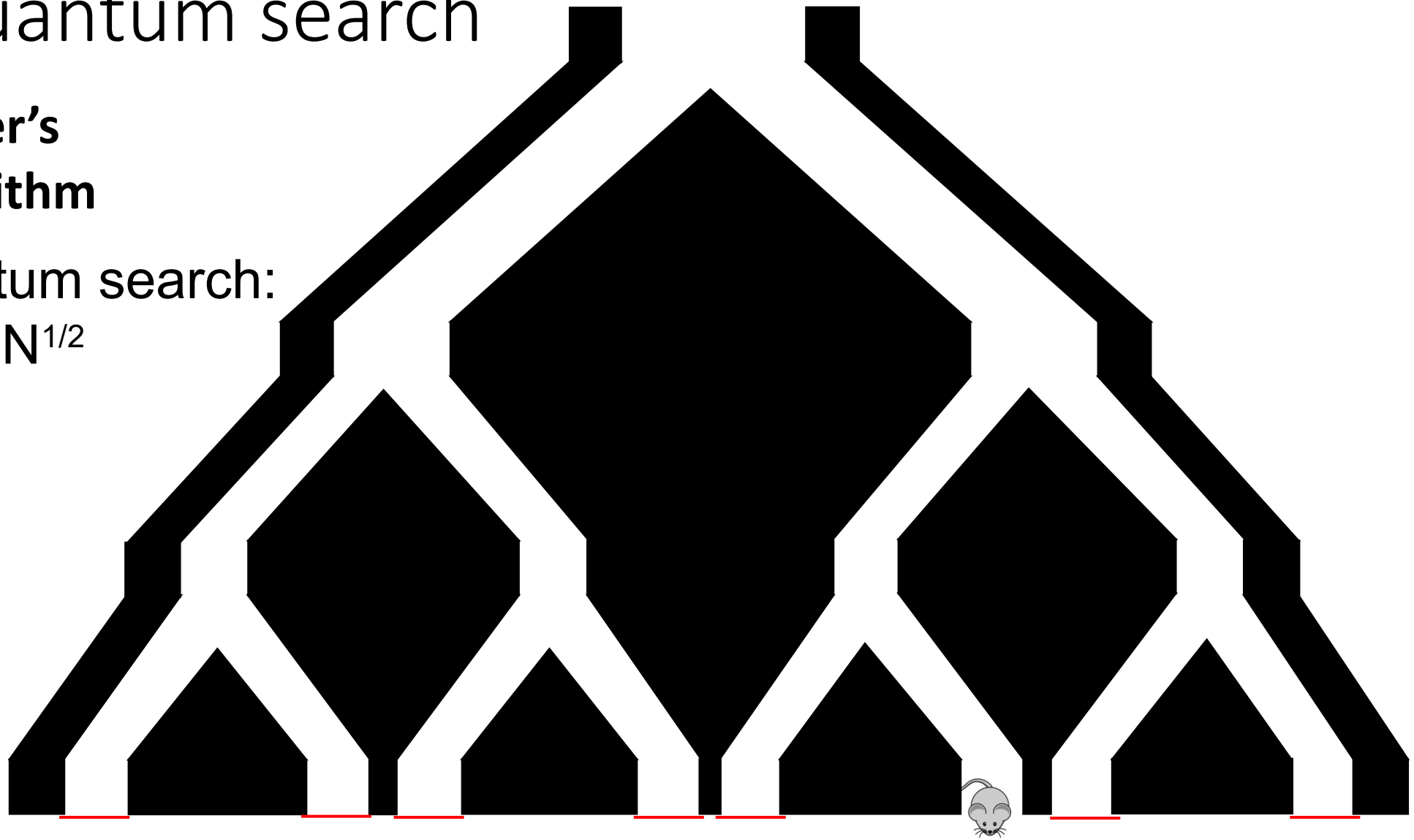
Grover's Algorithm



Quantum search

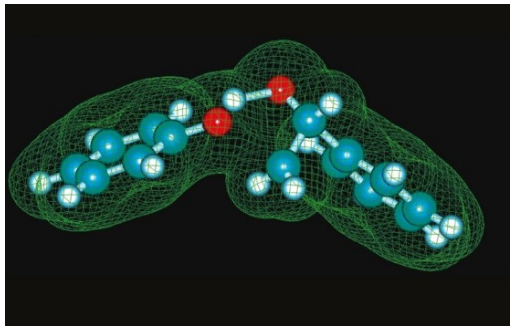
Grover's Algorithm

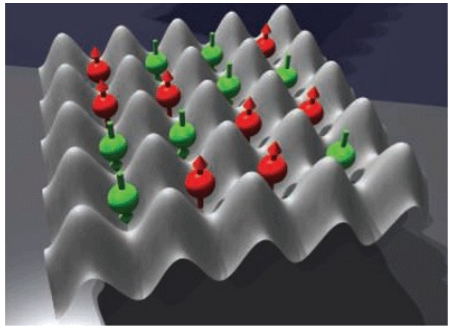
Quantum search: order $N^{1/2}$



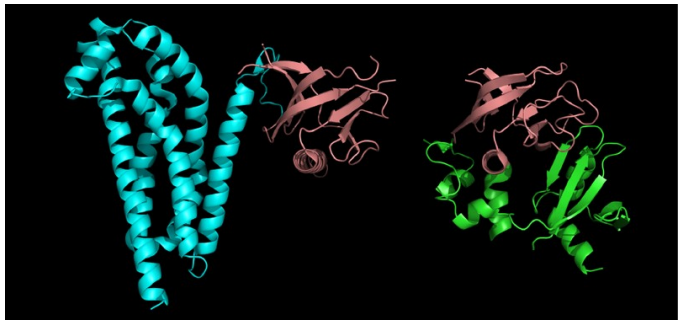
Other Potential Applications

- Traveling salesman problem
- Quantum Simulation





- Protein conformation, or other problems with many variables (weather, stock market)



Images from:
http://www.ornl.gov/info/ornlreview/v37_2_04/article08.shtml
L. Fallani, M. Inguscio, *Science*, **322**, 1480 (2008).
http://en.wikipedia.org/wiki/Protein_structure

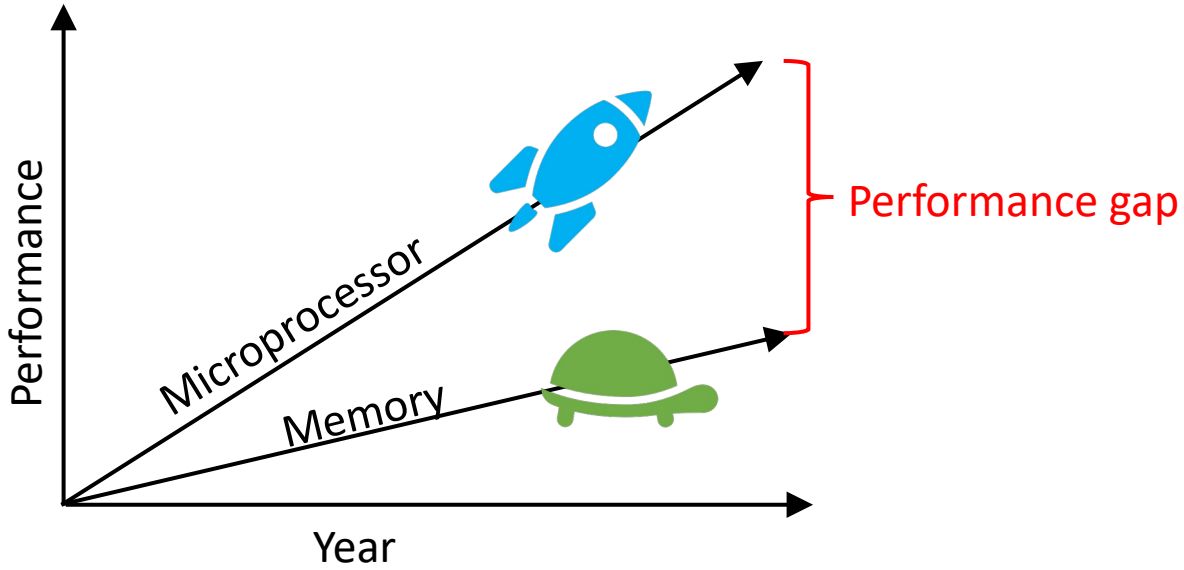
Universität Bremen

Today's Agenda

- Bottlenecks and Challenges
  - Parallelism
  - Speed
  - Energy
- GPUs
- Quantum Processors
- Neuromorphic Processor

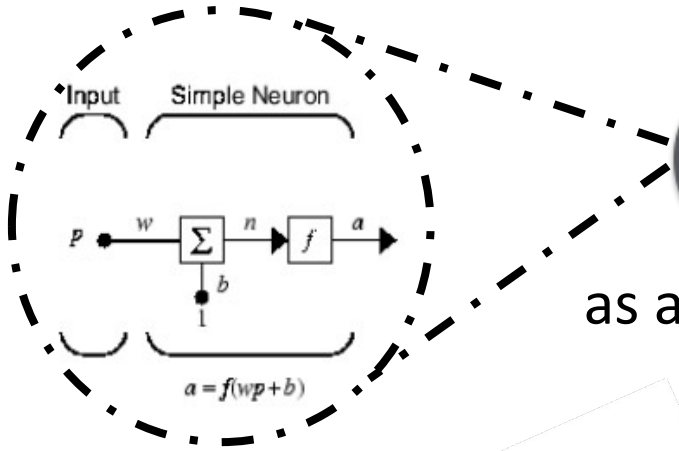
Memory Wall- Revisit

- Memory Wall
  - CPU clock is much faster than memory latency
  - Main memory is stored outside the CPU
  - Bottleneck is bandwidth between the two



Source: excellent article, "*The Many-Core Inflection Point for Mass Market Computer Systems*", by John L. Manferdelli, Microsoft Corporation
https://icl.utk.edu/ctwatch/quarterly/articles/2007/02/the-many-core-inflection-point-for-mass-market-computer-systems/2/index.html

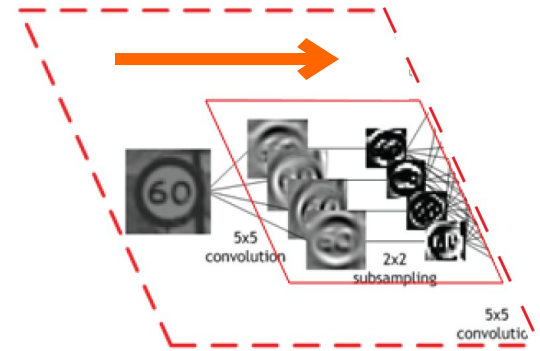
Too Many Memory Accesses



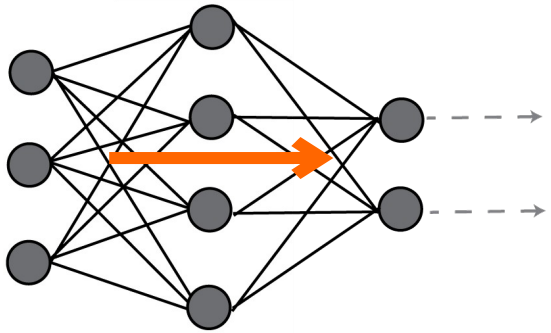
'neuron'

Simple computing elements…

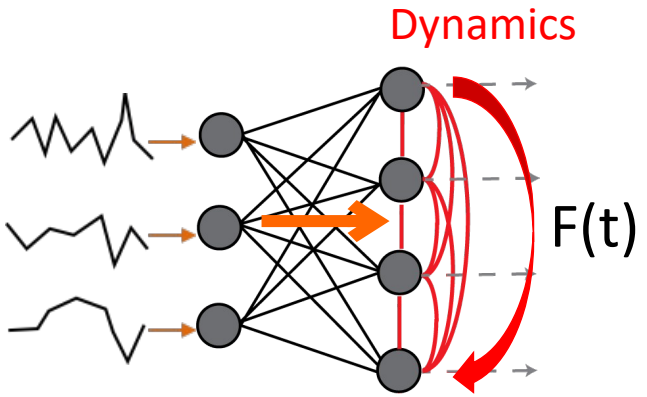
as a network, learn to perform diverse functions



Feature Extraction

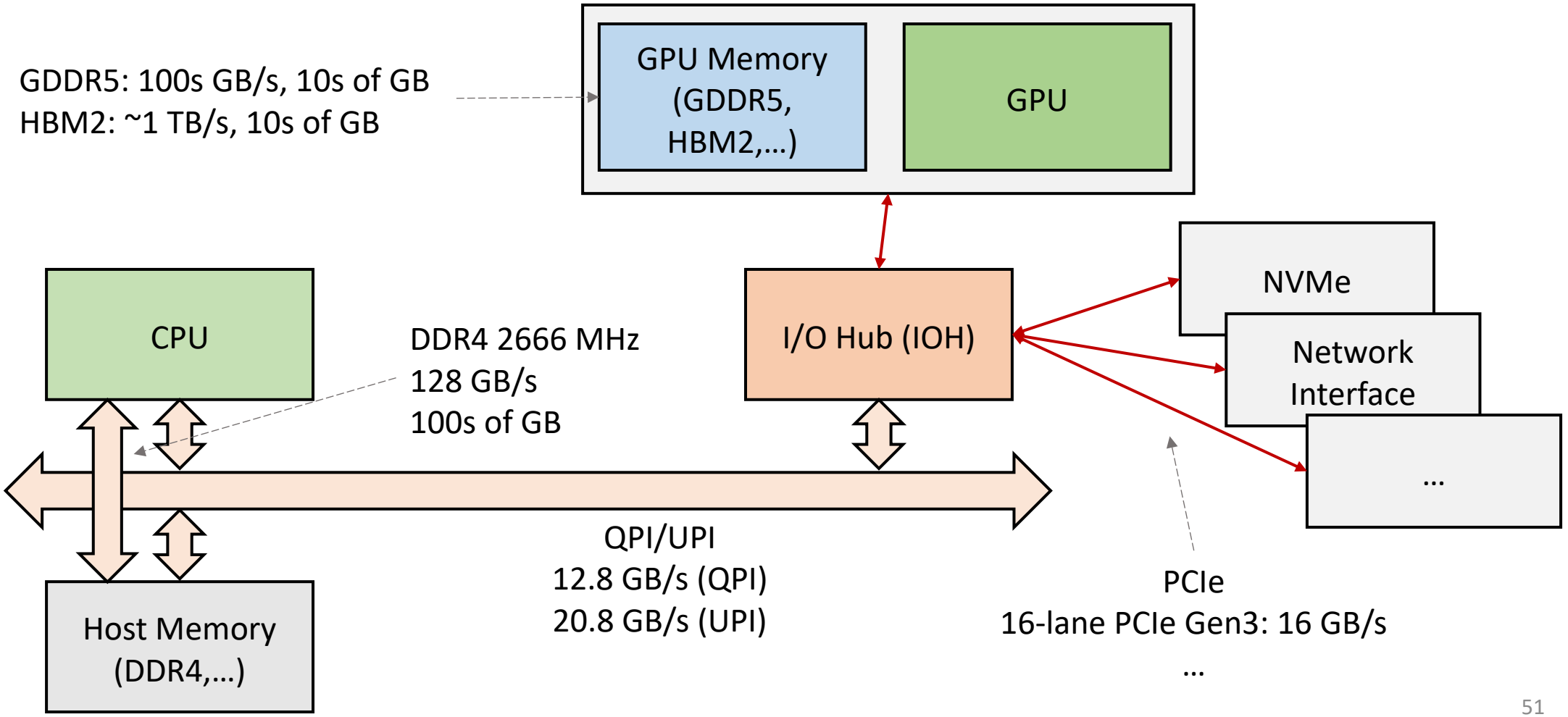


Classification



Time-varying Functions

System Architecture (2020)

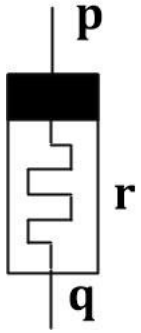


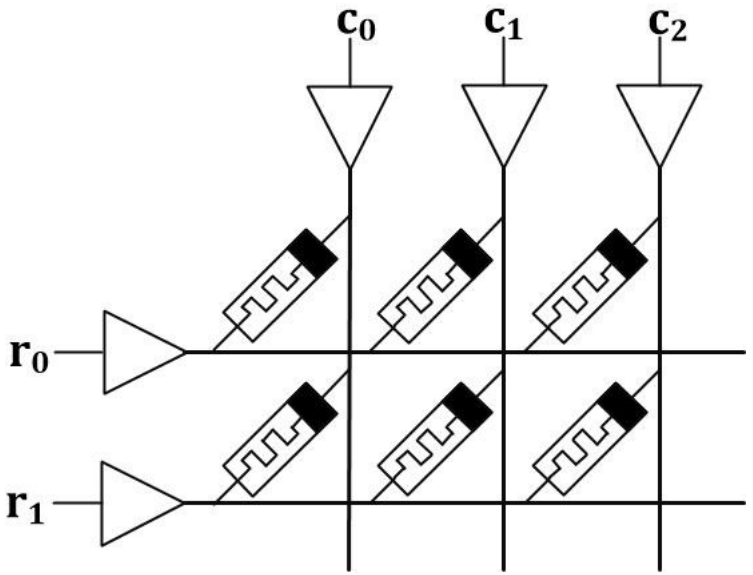
Universität Bremen

Neuromorphic Computing

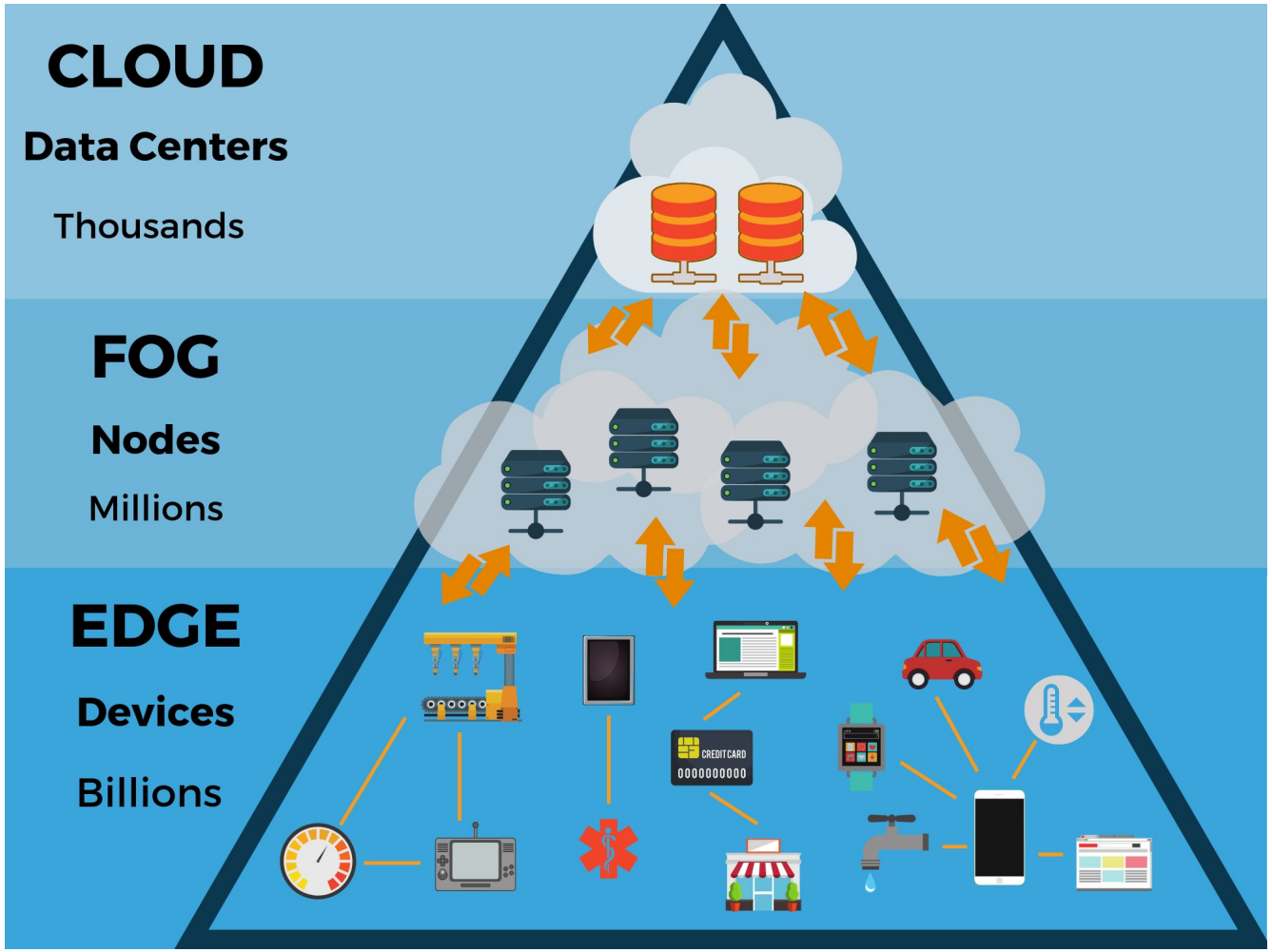
Memristors

- Resistive Random Access Memory
- Low resistance state – logic 1
- High resistance state – logic 0





Edge Computing



https://www.thinkebiz.net/what-edge-computing/

Universität Bremen

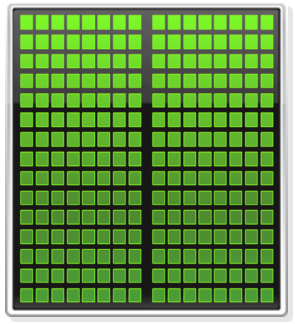
Future of Computer Architecture

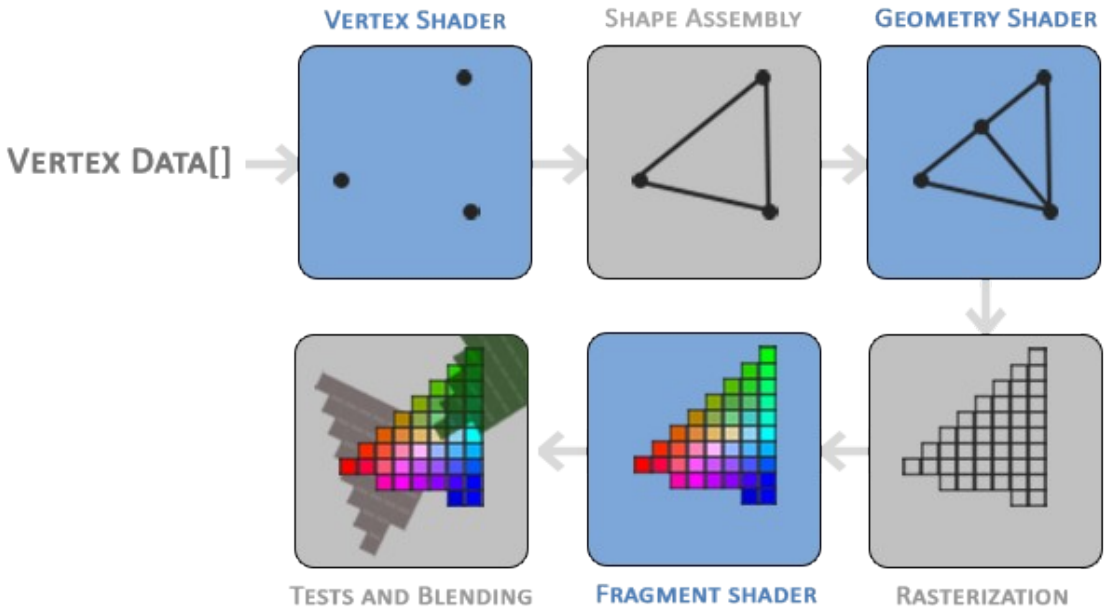
GPUs, Quantum Computing, Neuromorphic Computing

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck

- Backup

Traditional GPU workflow

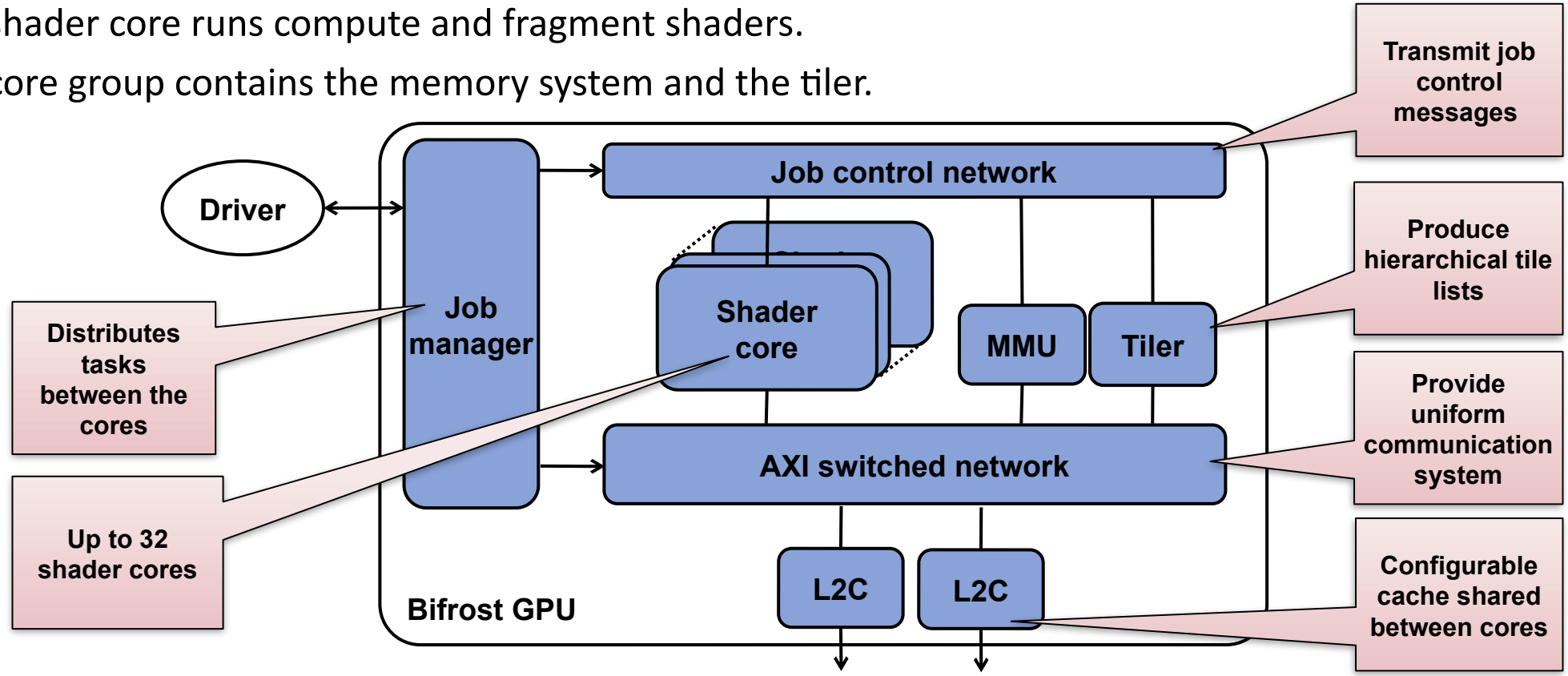




https://learnopengl.com/#!Getting-started/Hello-Triangle

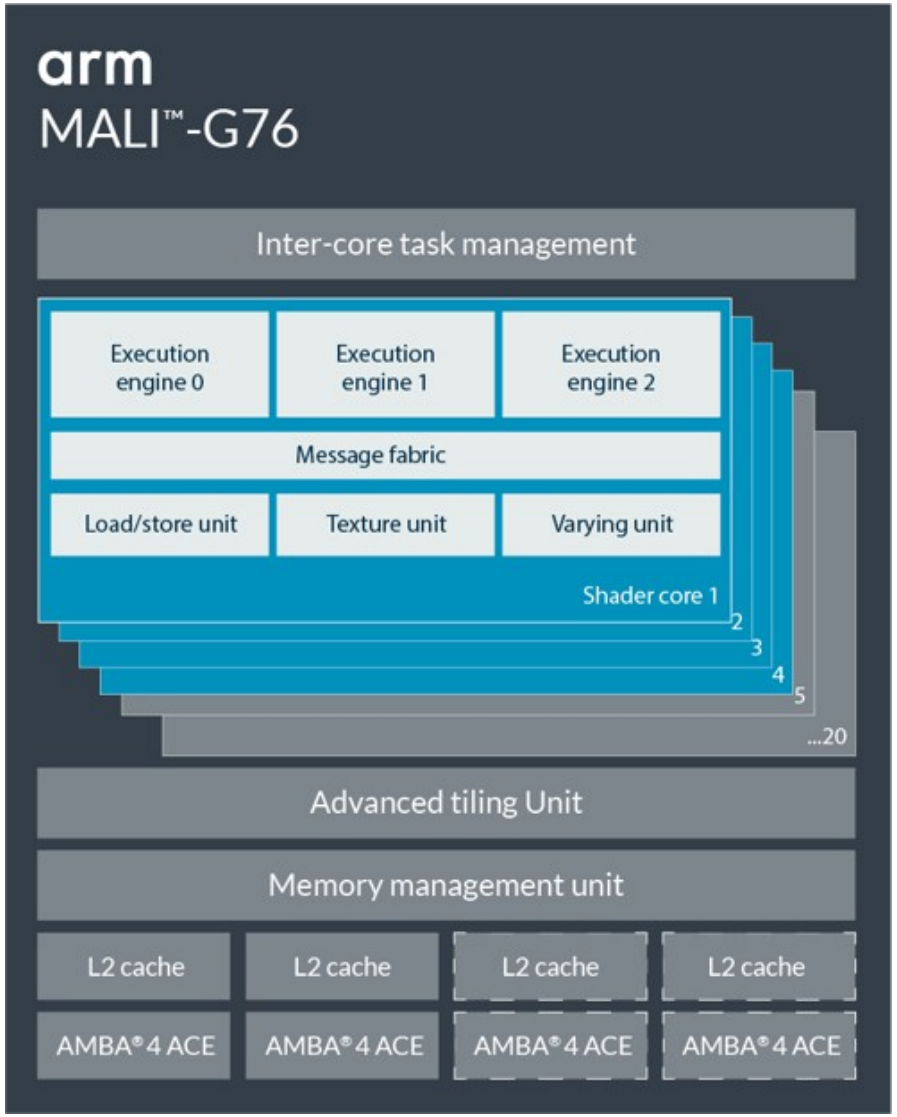
Case Study: Arm Mali Bifrost GPU Architecture

- Mali Bifrost GPU consists of 3 main blocks or groups.
  - The job manager interacts with the driver and controls the GPU HW.
  - The shader core runs compute and fragment shaders.
  - The core group contains the memory system and the tiler.



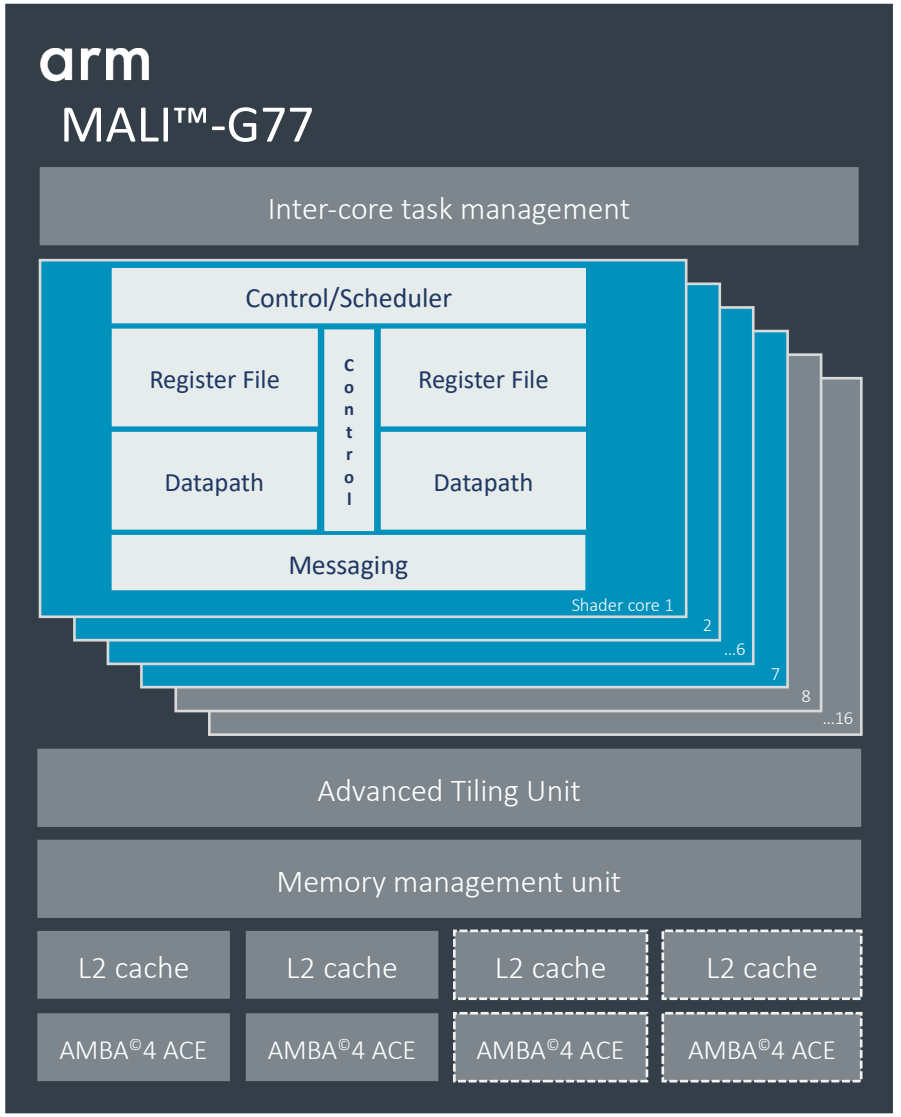
Case Study: Arm Mali-G76 GPU

- Third generation of the Bifrost architecture
- Maximum 20 shader cores (SC)
  - Wider execution engines with double the number of lanes
- Performance
  - Complex graphics and machine-learning workloads
- API support
  - OpenGL ES 1.1, 2.0, 3.1, 3.2
    OpenCL 1.1, 1.2, 2.0 Full profile
    Vulkan 1.1
- Shared L2 cache with 2 or 4 slices

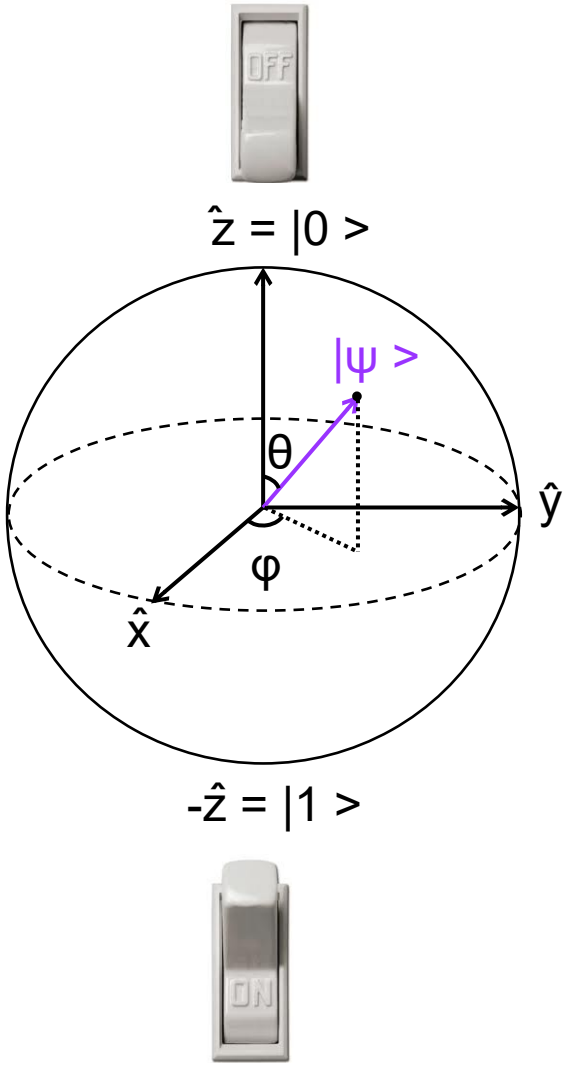


Case Study: Arm Mali-G77 GPU

- First generation of the Valhall architecture
  - Warp-based execution model
  - New instruction set with operational-equivalence to Bifrost
  - Dynamic instruction scheduling in hardware
- Configurable 7 to 16 shader cores
- Single execution engine per shader core
- Configurable 2 to 4 slices of L2 cache
  - 512 KB to 4 MB in total
- Texture mapper – 4 texture element (texel) per cycle
- Supports Vulkan and OpenCL



Quantum Bits (Qubits)



Quantum Bit:

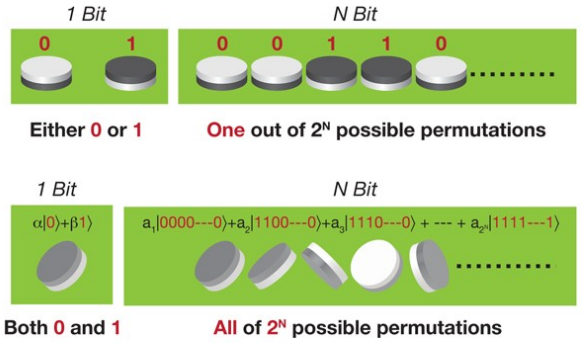
$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\varphi}\sin(\theta/2)|1\rangle$

$= a|0\rangle + b|1\rangle$

Any vector pointing to the surface of the sphere is a valid quantum state!

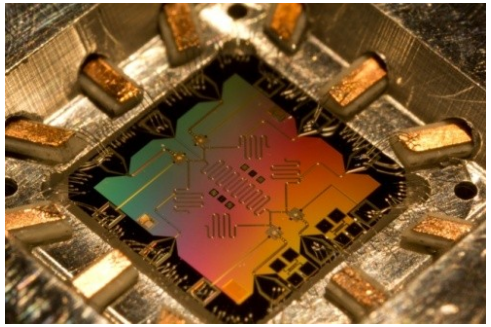
Outline

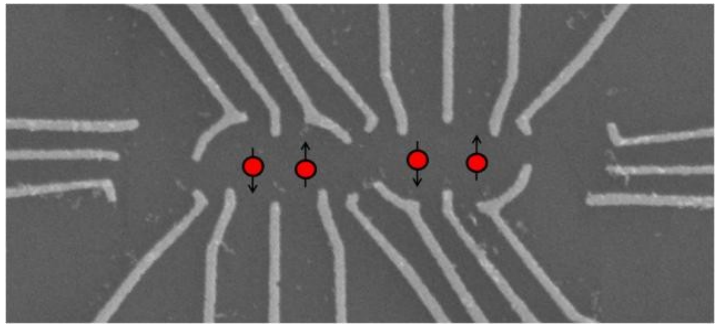
- What are quantum computers?
  - Classical vs. quantum bits



- Problems suited to quantum computation

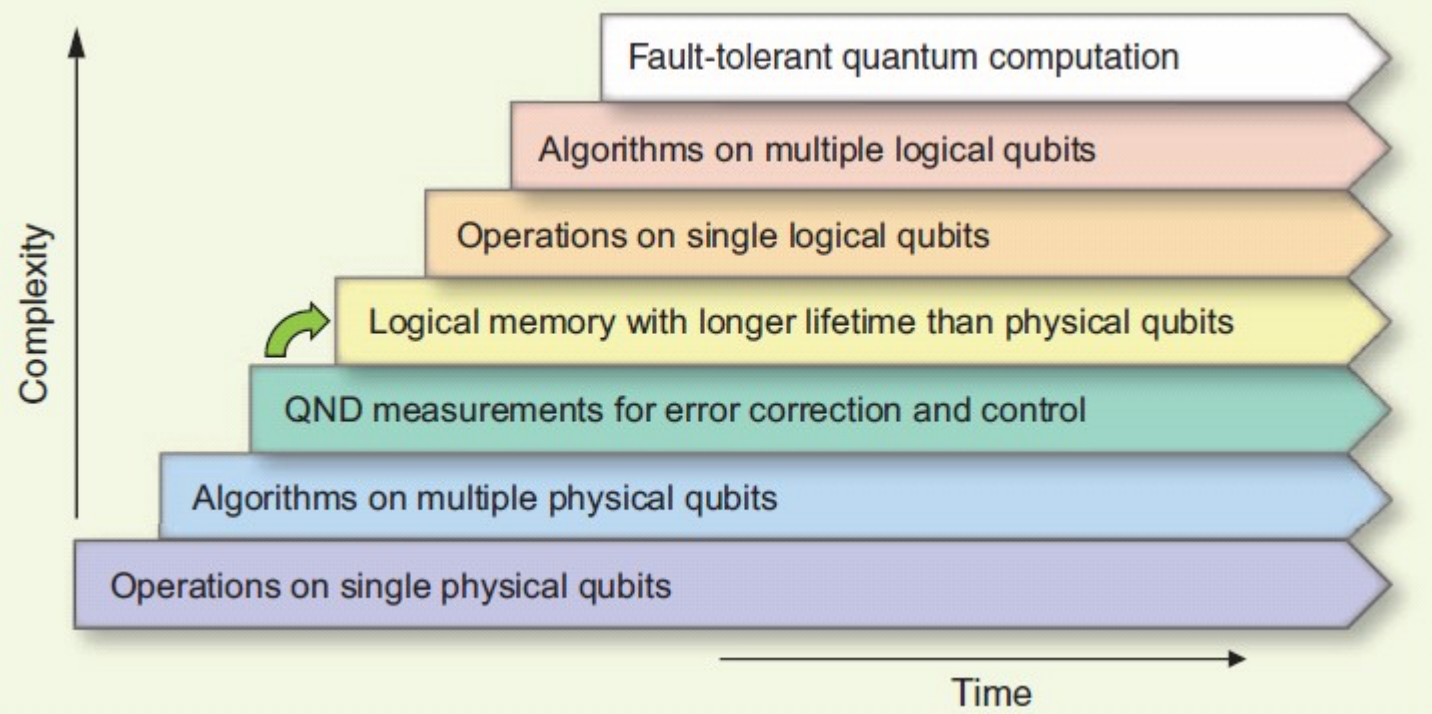
- How to realize a quantum computer





Steps to Build a Quantum Computer

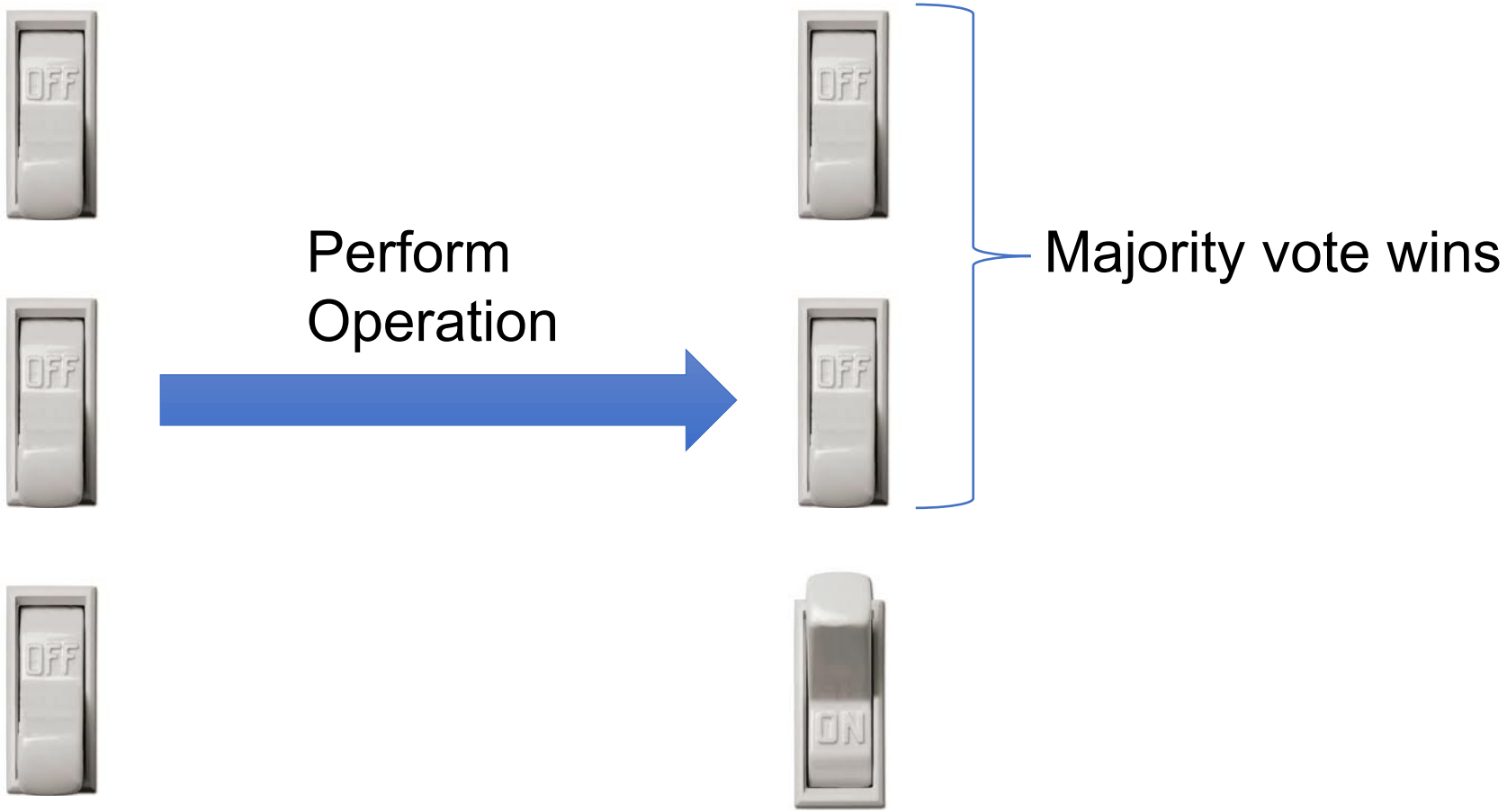
- Roadmap toward realizing a quantum computer



M. H. Devoret and R. J. Schoelkopf, *Science*, **339**, 1169 (2013).

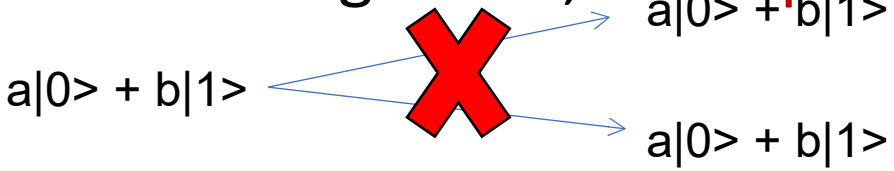
Classical Error Correction

- Simply make three identical bits, and use majority rules



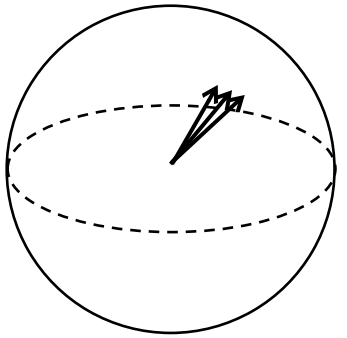
What about errors?

- No-cloning theorem: in general, it's impossible to copy a qubit



- Measurement collapses a quantum state
  - How do you even tell if an error has occurred?

- "Analog"-like system: continuous errors are possible



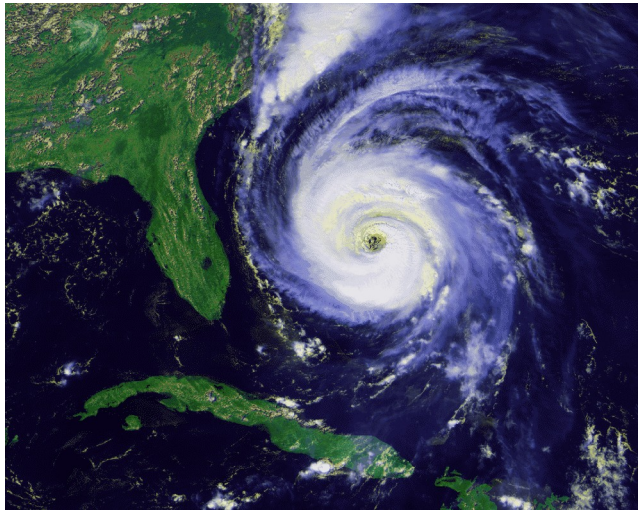
Universität Bremen

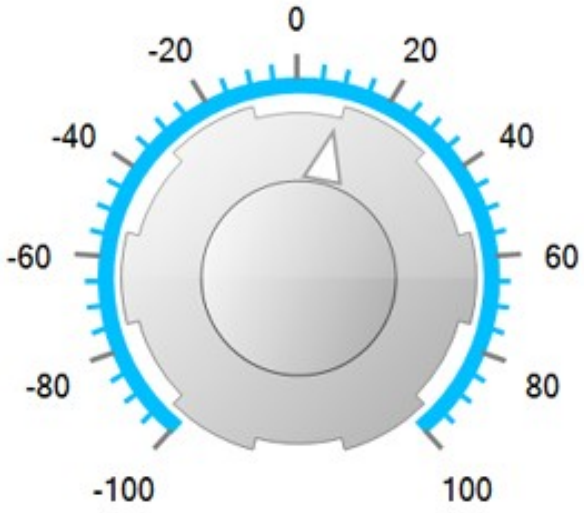
Quantum Error Correction is Possible

- Errors can be found without performing a quantum measurement
- Error rate threshold for fault-tolerant quantum computation: $<10^{-4}$ to 10^{-6}
- Many ancillary qubits likely necessary for each logical qubit, but it can be done!

Qubit Control vs. Isolation

- Fundamental dichotomy:
  - Any interaction with environment is bad: it ‘decoheres’ qubits (acts as a measurement)
  - Yet we want full control over the qubit





Quantum Annealing: D-wave

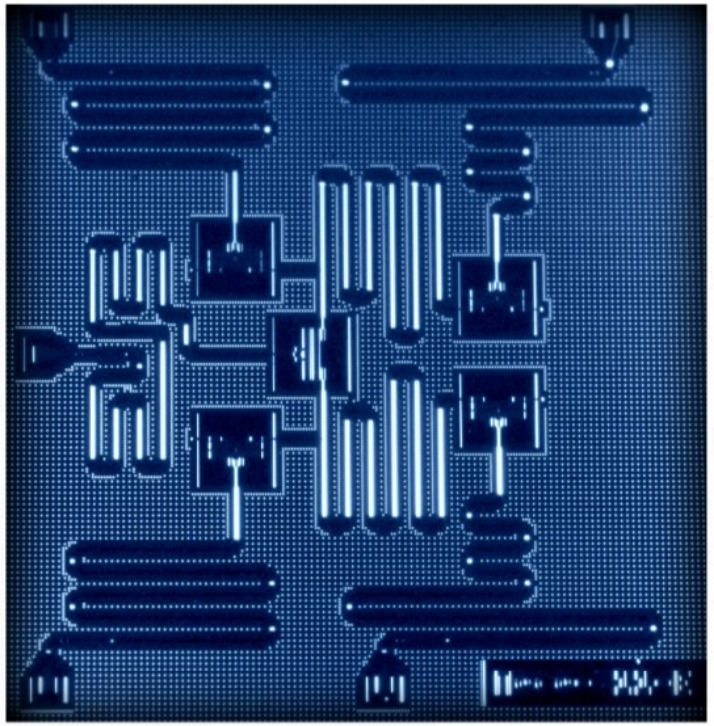
- D-wave makes and sells quantum annealers (superconducting qubit base)
- Lockheed Martin: bought a 128-qubit model
- Google and NASA: bought one with 512 qubits to work on artificial intelligence
- But: debate in community whether these are true quantum computers, and if they're better than classical algorithms

See: http://www.dwavesys.com/en/dw_homepage.html

http://scottaaronson.com/blog

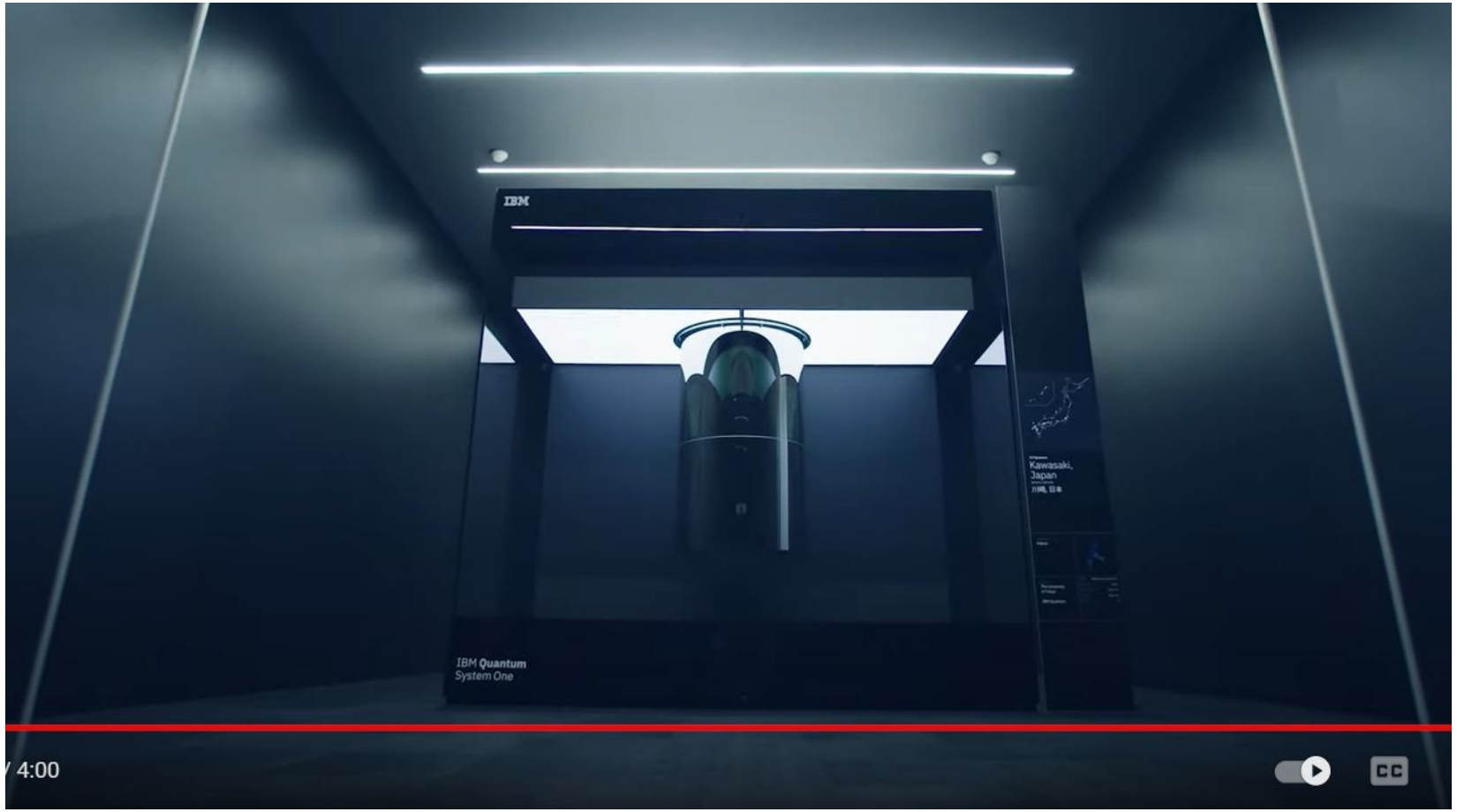
IBM 50 qubits quantum computer

IBM 5 qubits processor



Capture an atom at Chiengmai U, Physics

IBM quantum computer system one



https://www.youtube.com/watch?v=tPMY4oP3Qgk

Universität Bremen

CUDA: Making the GPU Tick…

- “Compute Unified Device Architecture” – freely distributed by NVIDIA
- It enables a general purpose programming model
  - User kicks off batches of threads on the GPU
- Targeted software stack
  - Scientific computing oriented drivers, language, and tools
- Driver for loading computation programs into GPU
  - Standalone Driver - Optimized for computation
  - Interface designed for compute - graphics free API
  - Explicit GPU memory management

Universität Bremen

CUDA Execution Abstraction

- Block: Multi-dimensional array of threads
  - 1D, 2D, or 3D
  - Threads in a block can synchronize among themselves
  - Threads in a block can access shared memory
  - CUDA (Thread, Block) ~= OpenCL (Work item, Work group)
- Grid: Multi-dimensional array of blocks
  - 1D or 2D
  - Blocks in a grid can run in parallel, or sequentially
- Kernel execution issued in grid units
- Limited recursion (depth limit of 24 as of now)

Simple CUDA Example

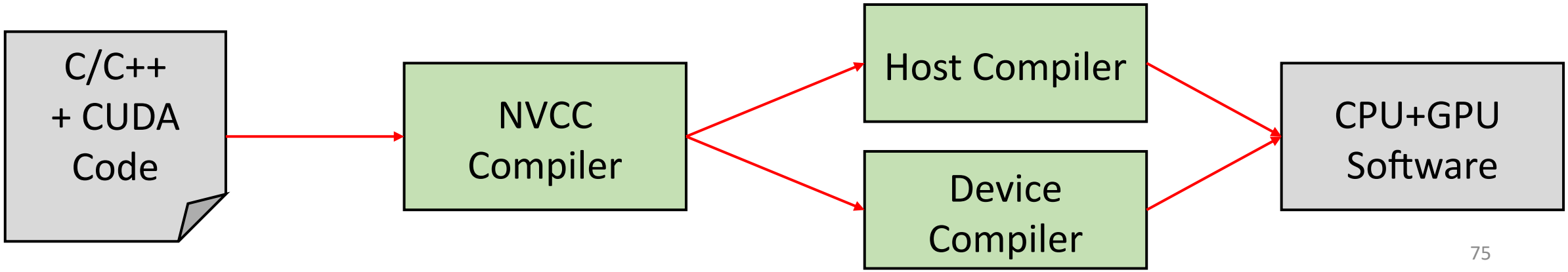
Asynchronous call

CPU side

```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

GPU side

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```



Simple CUDA Example

```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
}
```

1 block

N threads per block

Should wait for kernel to finish

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

Only void allowed

Which of N threads am I?
See also: blockIdx

__global__:
In GPU, called from host/GPU
__device__:
In GPU, called from GPU
__host__:
In host, called from host

N instances of VecAdd spawned in GPU

One function can be both

Universität Bremen

More Complex Example: Picture Blurring

- Slides from NVIDIA/UIUC Accelerated Computing Teaching Kit
- Another end-to-end example https://devblogs.nvidia.com/even-easier-introduction-cuda/

- Great! Now we know how to use GPUs

What is In-Memory Computing?