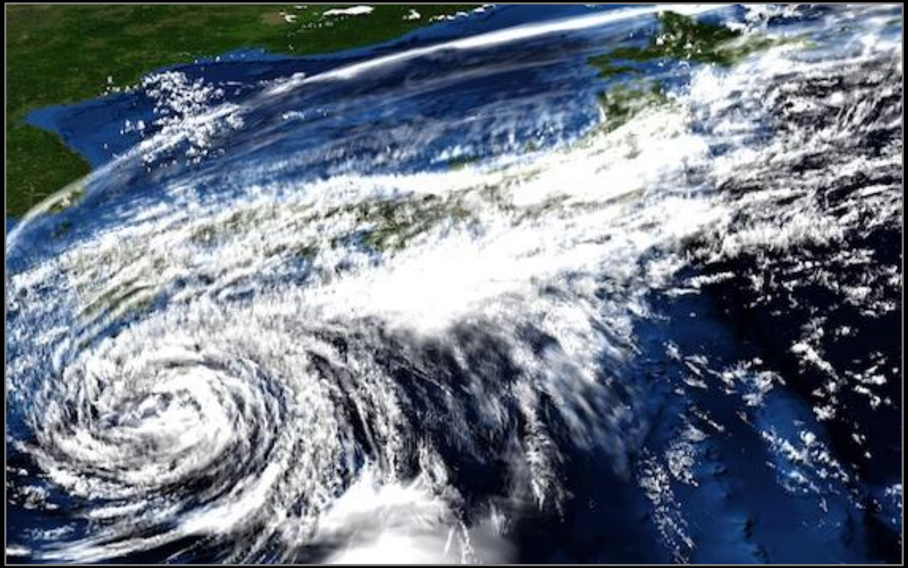
NVIDIA

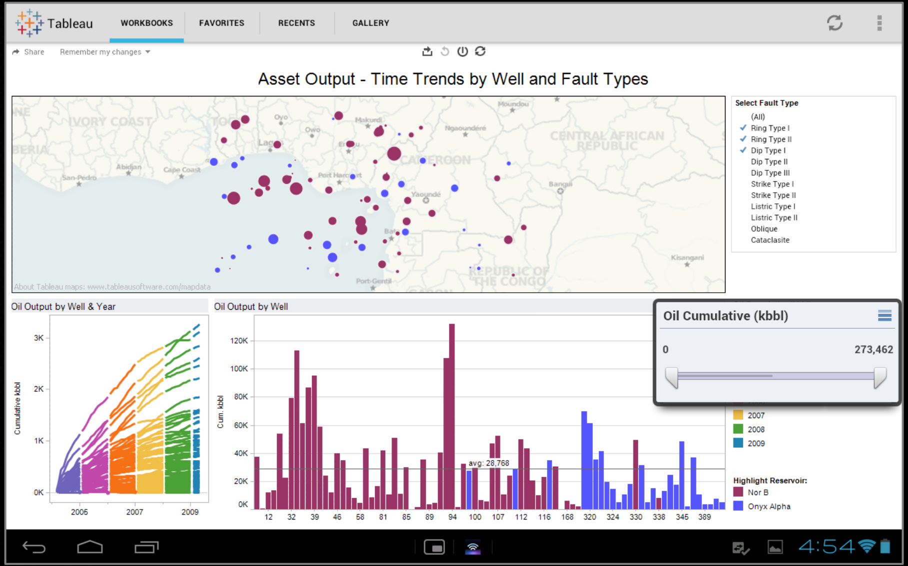
Efficiency and Programmability: Enablers for ExaScale

Bill Dally | Chief Scientist and SVP, Research NVIDIA | Professor (Research), EE&CS, Stanford

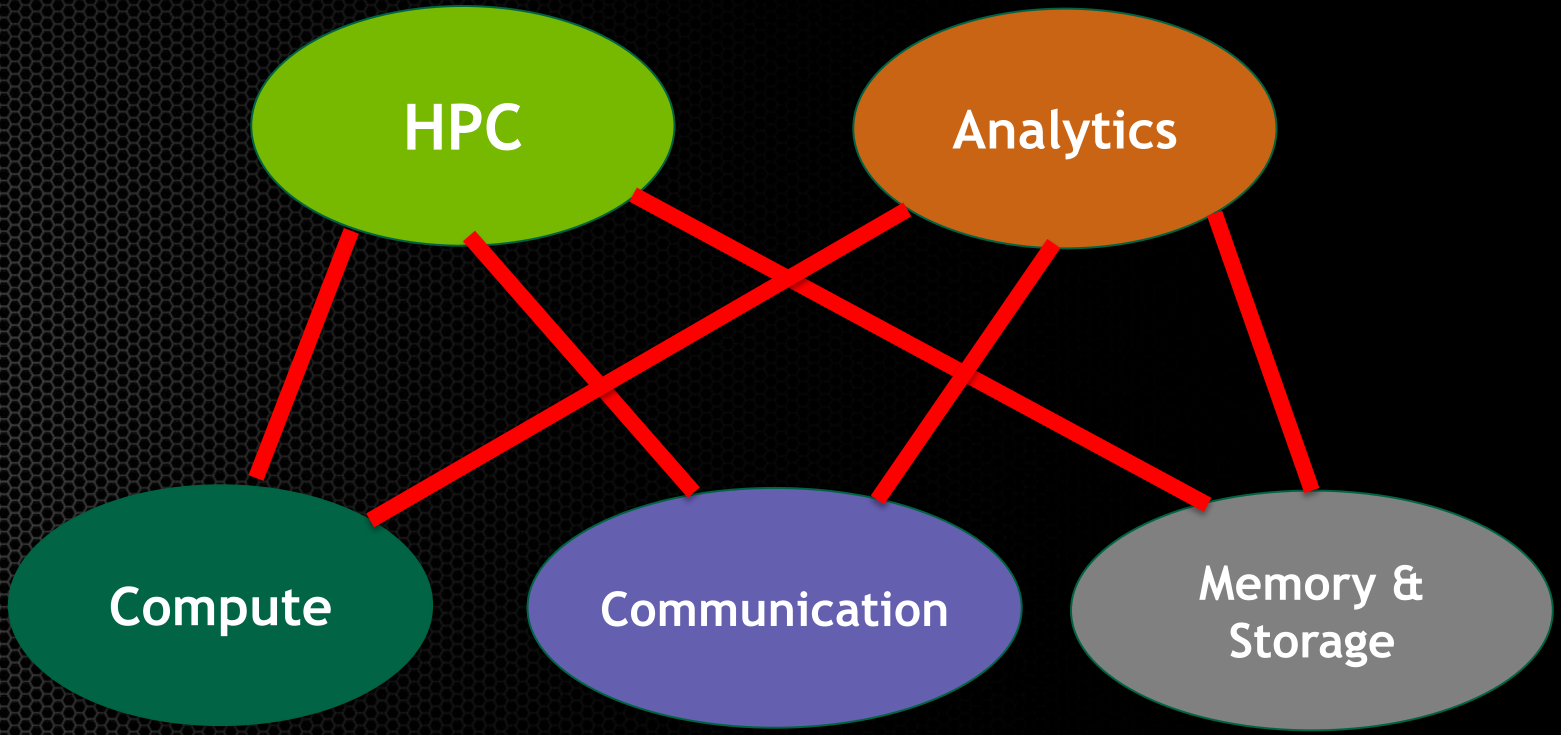
Scientific Discovery and Business Analytics

Driving an Insatiable Demand for More Computing Performance

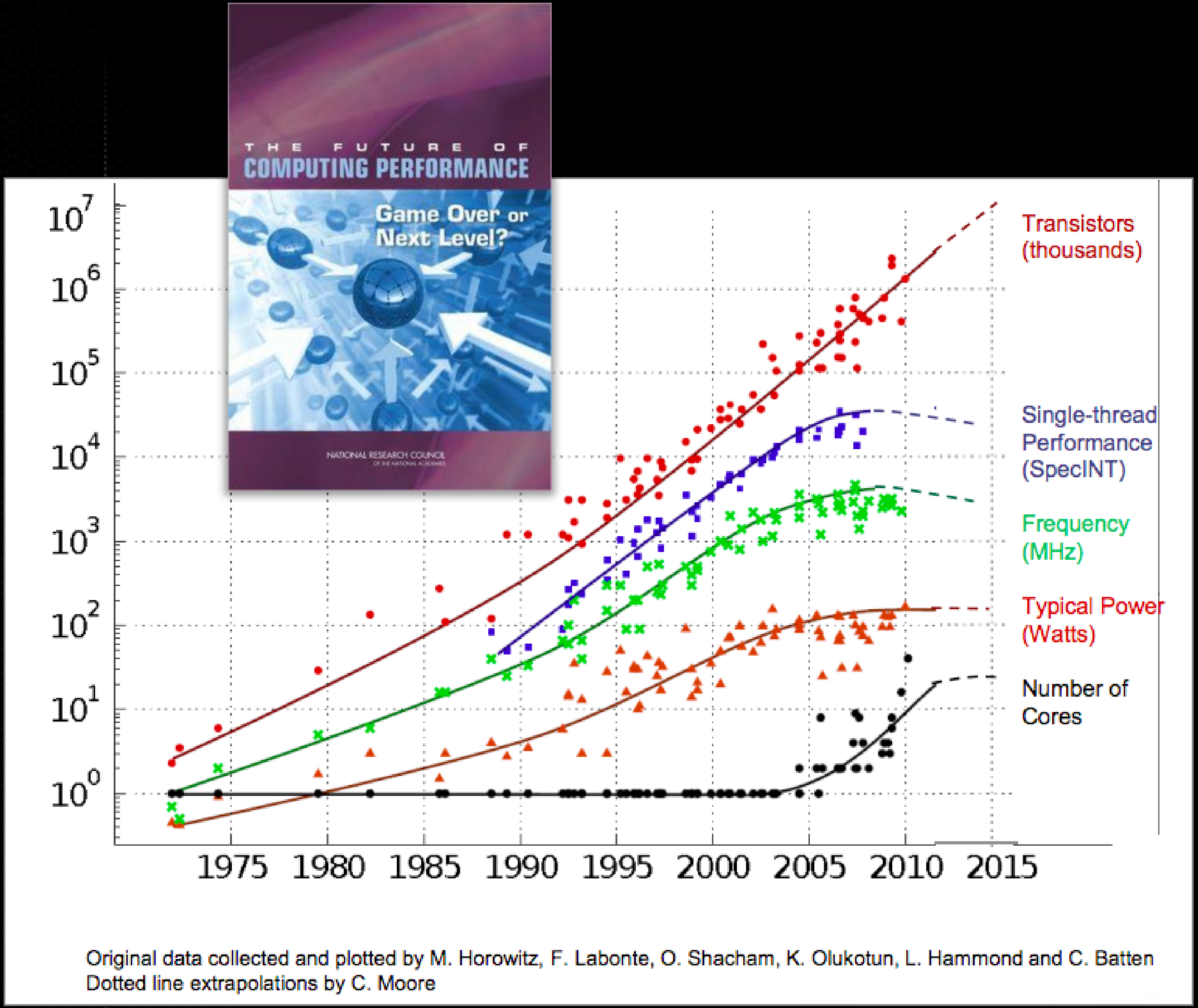




NVIDIA


HPC
Analytics
Compute
Communication
Memory &
Storage

The End of Historic Scaling



Source: C Moore, Data Processing in ExaScale-ClassComputer Systems, Salishan, April 2011

"Moore's Law gives us more transistors…
Dennard scaling made them useful."

Bob Colwell, DAC 2013, June 4, 2013

TITAN

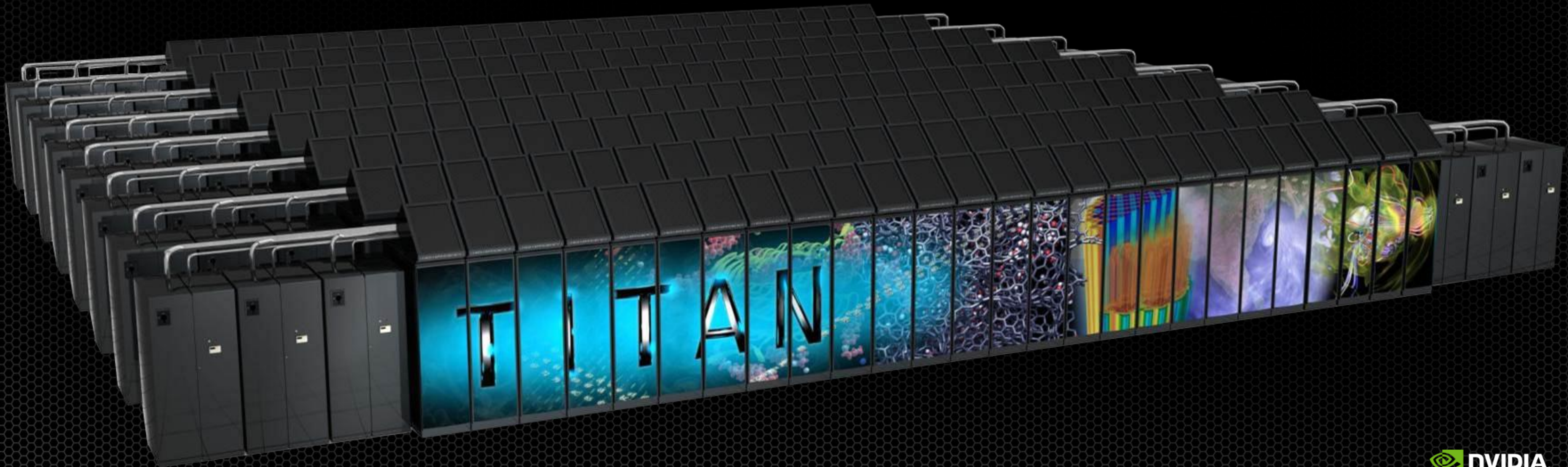
18,688 NVIDIA Tesla K20X GPUs

27 Petaflops Peak: 90% of Performance from GPUs

17.59 Petaflops Sustained Performance on Linpack

Numerous real science applications

2.14GF/W – Most efficient accelerator



TITAN

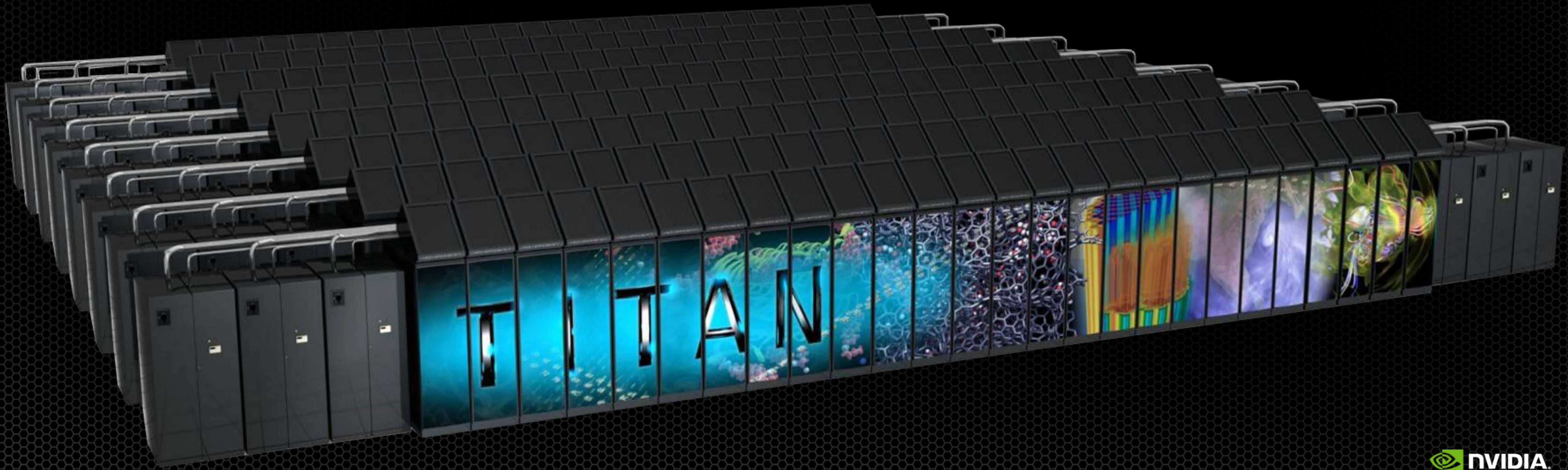
18,688 NVIDIA Tesla K20X GPUs

27 Petaflops Peak: 90% of Performance from GPUs

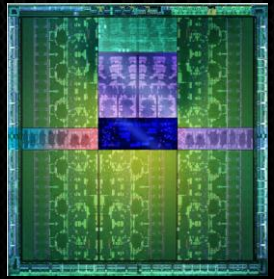
17.59 Petaflops Sustained Performance on Linpack

Numerous real science applications

2.14GF/W – Most efficient accelerator



You Are Here

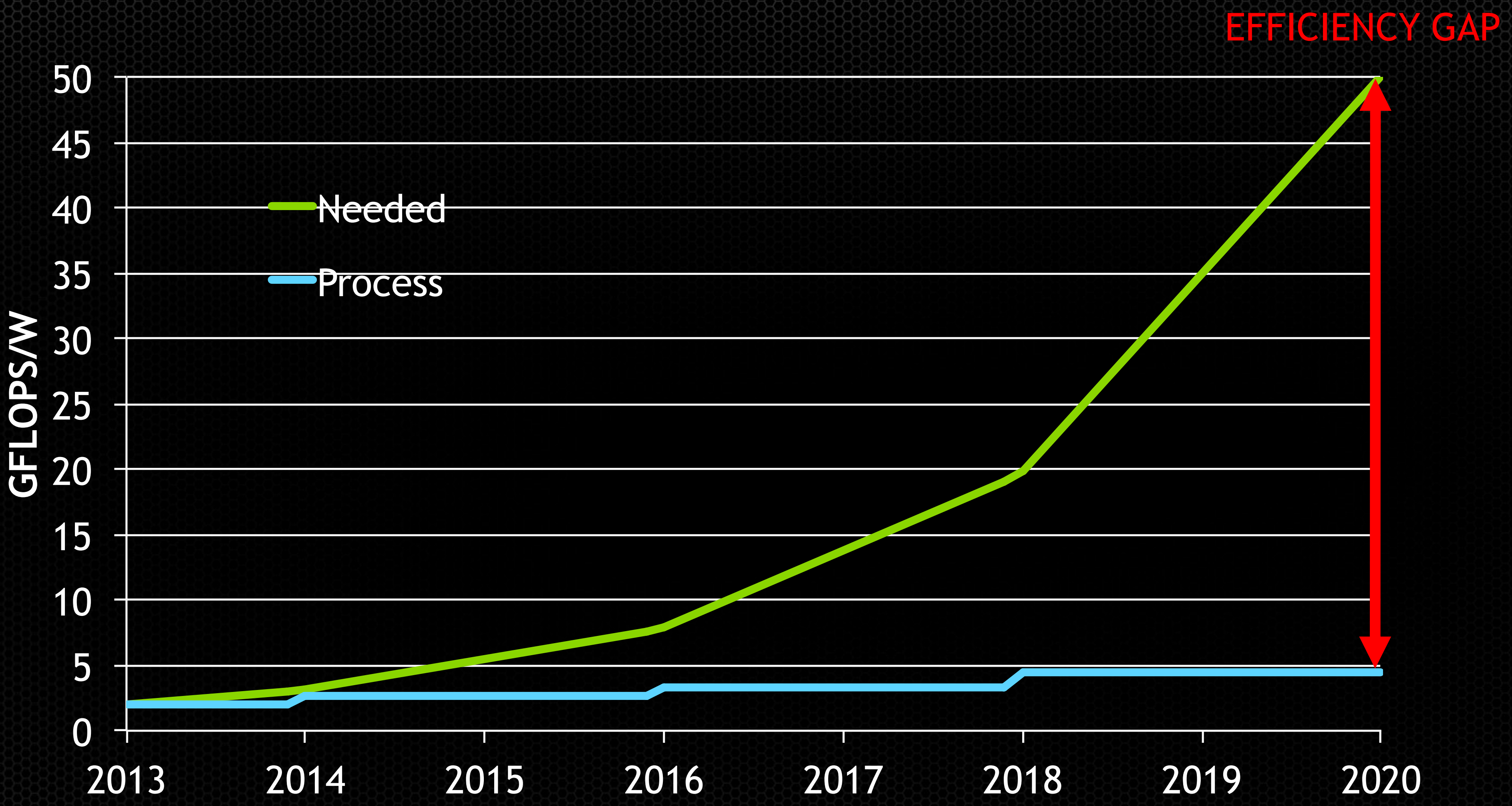


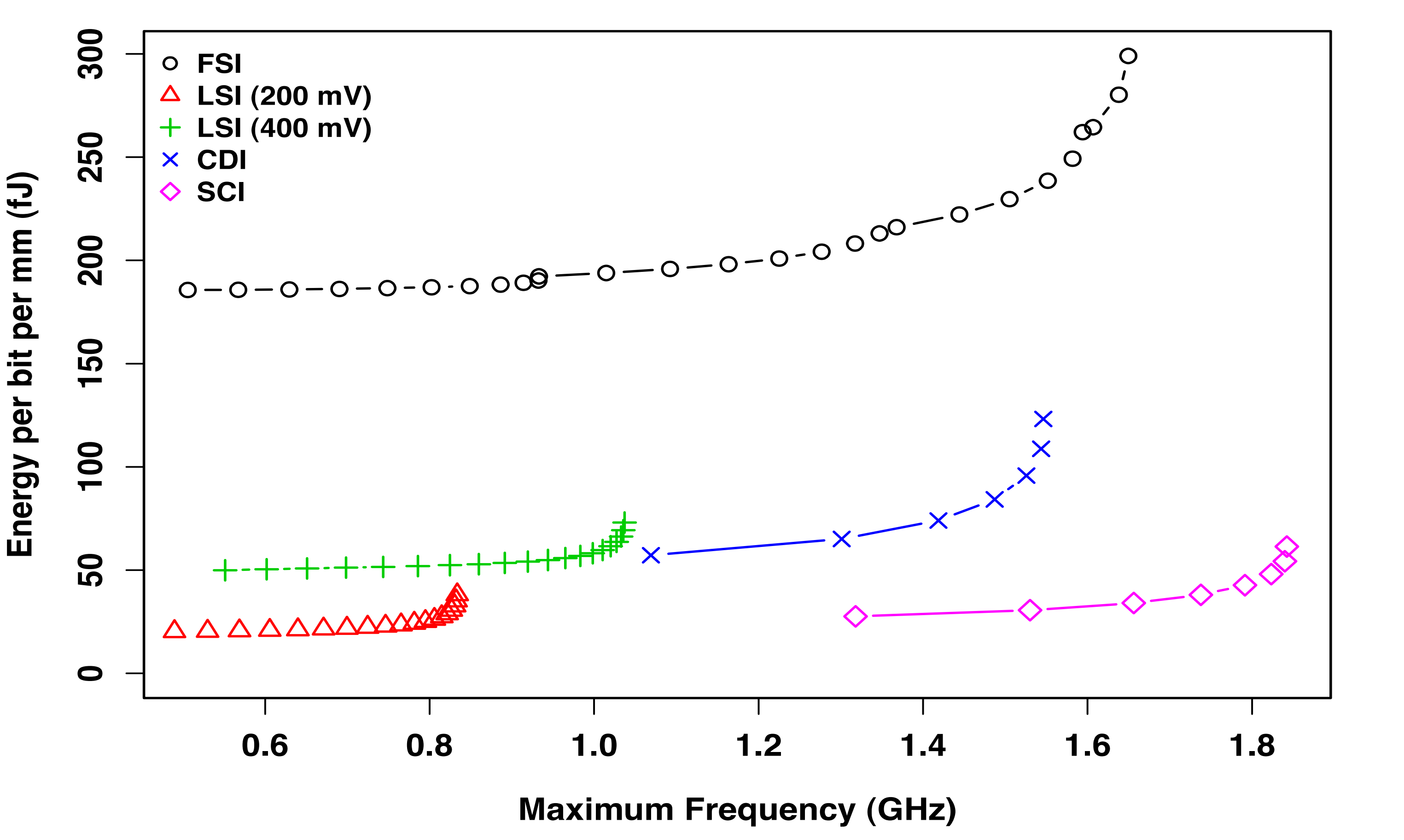
2013

20PF
18,000 GPUs
10MW
2 GFLOPs/W
~10^7 Threads

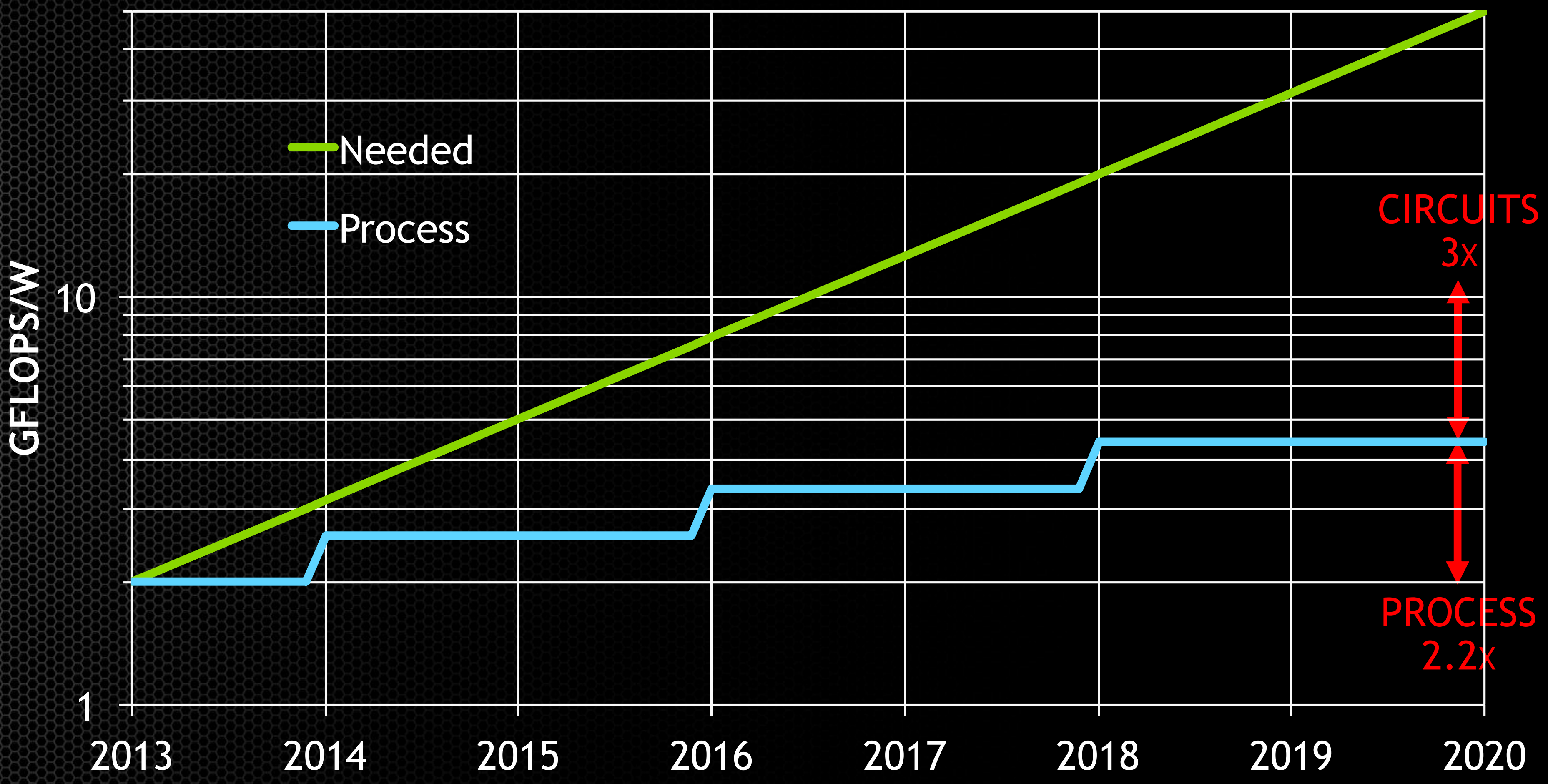
2020

1,000PF (50x)
72,000HCNs (4x)
20MW (2x)
50 GFLOPs/W (25x)
~10^{10} Threads (1000x)


EFFICIENCY GAP
GFLOPS/W
50
45
40
35
30
25
20
15
10
5
0
Needed
Process
2013
2014
2015
2016
2017
2018
2019
2020
NVIDIA



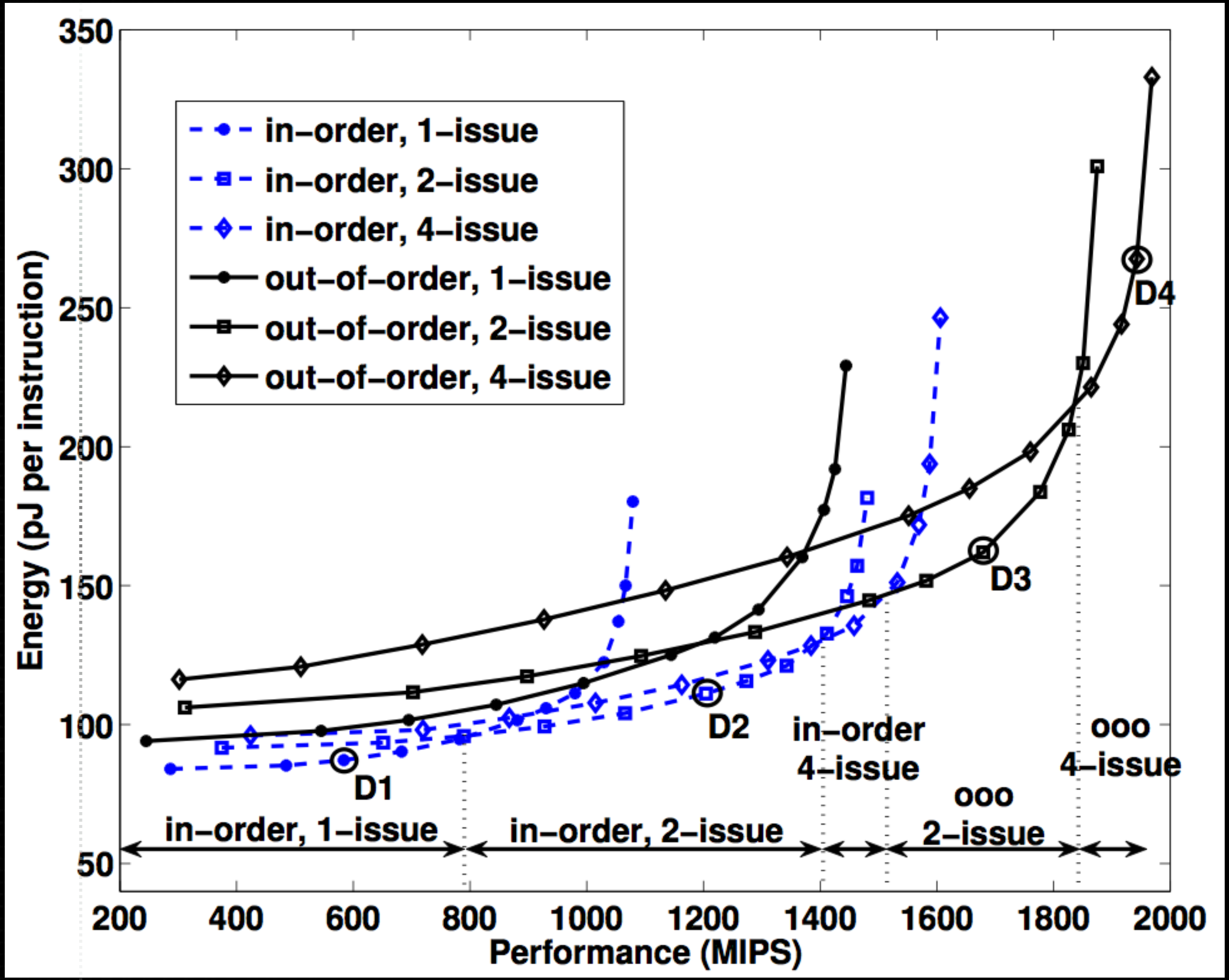
FSI
LSI (200 mV)
LSI (400 mV)
CDI
SCI
Energy per bit per mm (fJ)
0
50
100
150
200
250
300
0.6
0.8
1.0
1.2
1.4
1.6
1.8
Maximum Frequency (GHz)



Needed
Process
GFLOPS/W
10
1
2013
2014
2015
2016
2017
2018
2019
2020
CIRCUITS
3x
PROCESS
2.2x

NVIDIA

Simpler Cores
= Energy Efficiency



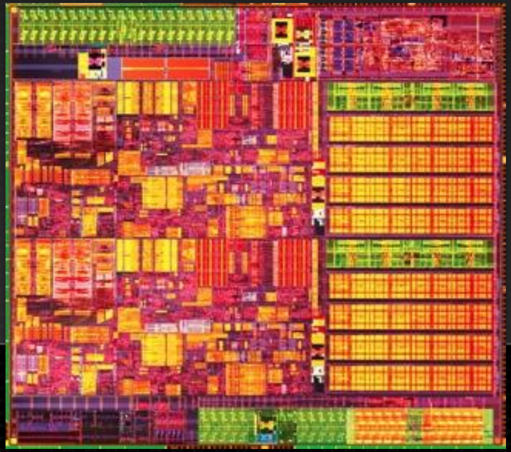
Source: Azizi [PhD 2010]

CPU

1690 pJ/flop

Optimized for Latency

Caches



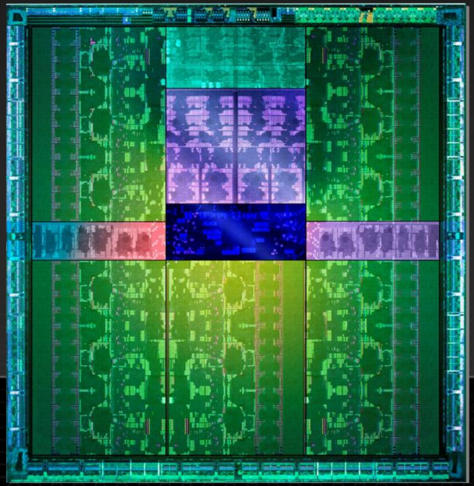
Westmere
32 nm

GPU

140 pJ/flop

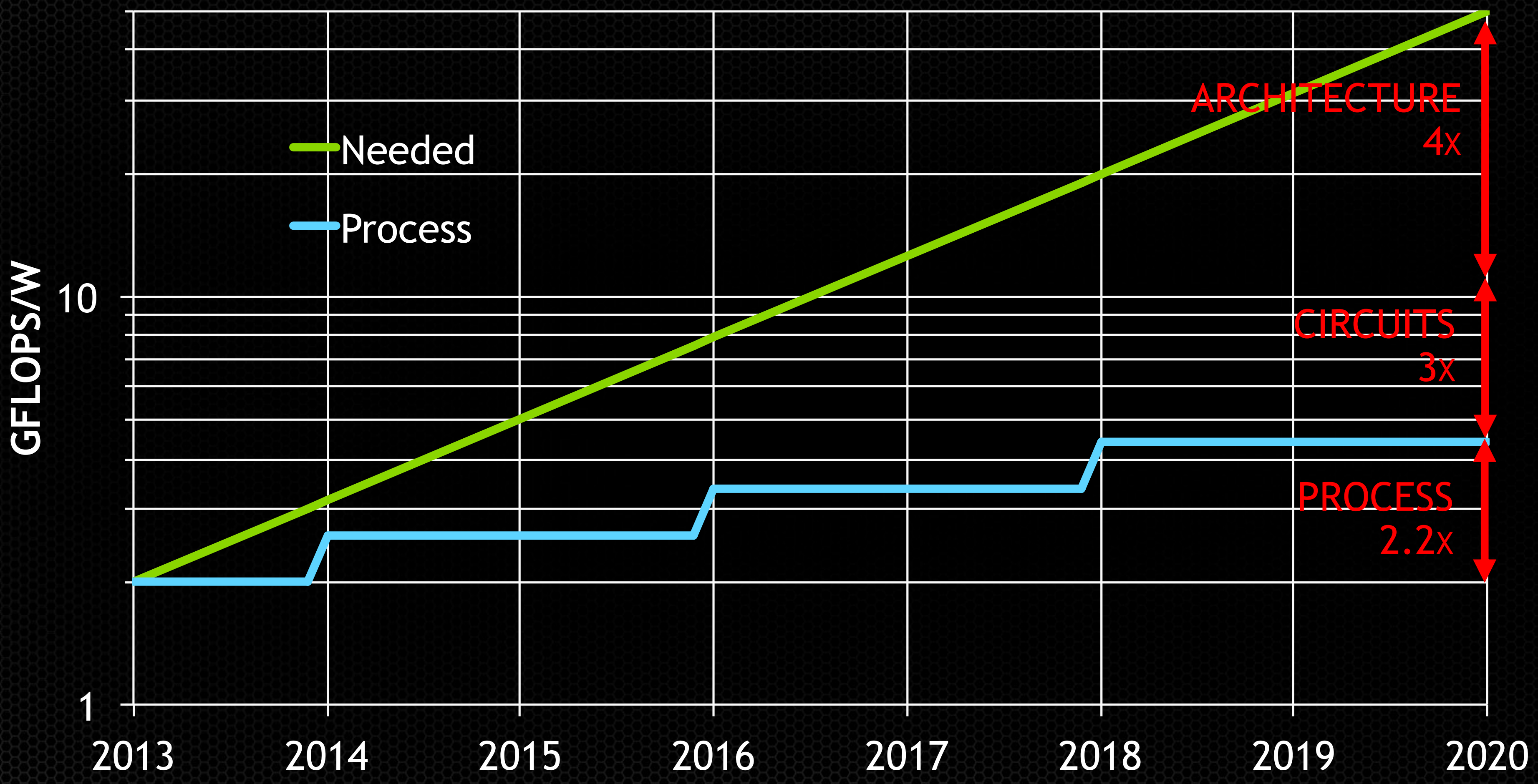
Optimized for Throughput

Explicit Management
of On-chip Memory



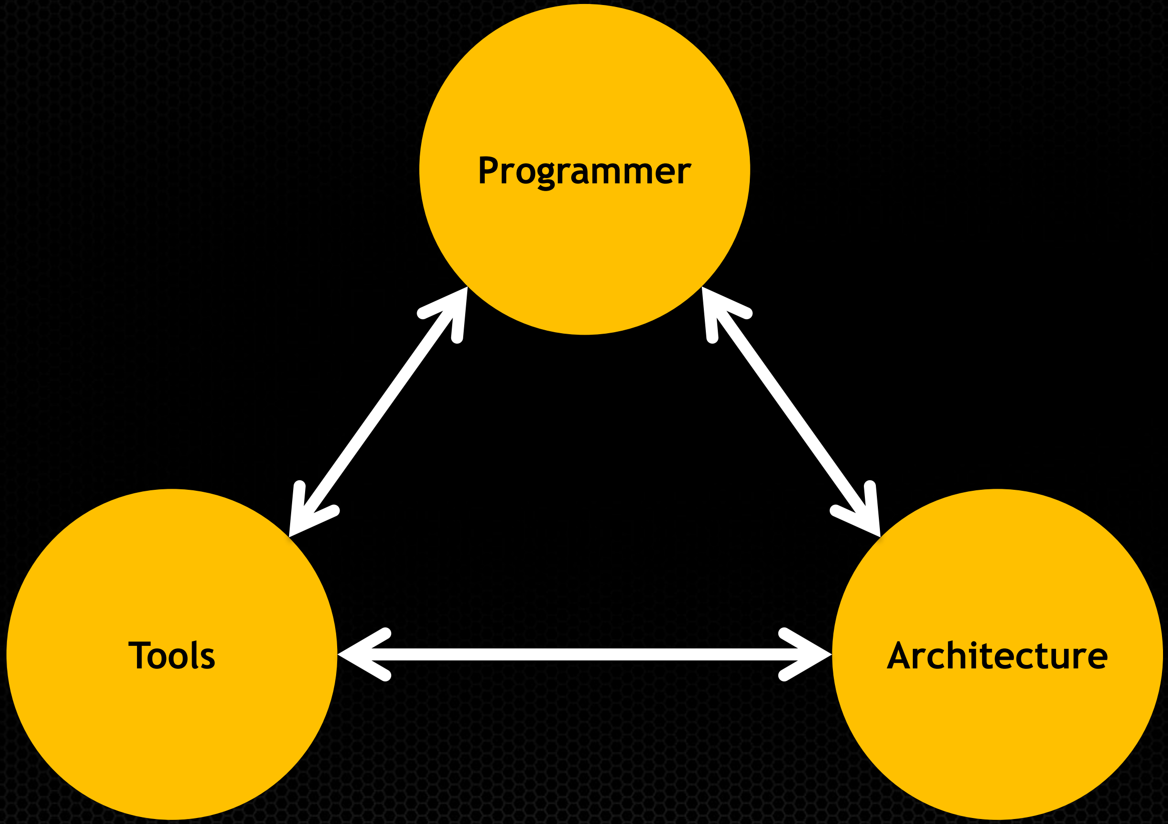
Kepler
28 nm

NVIDIA

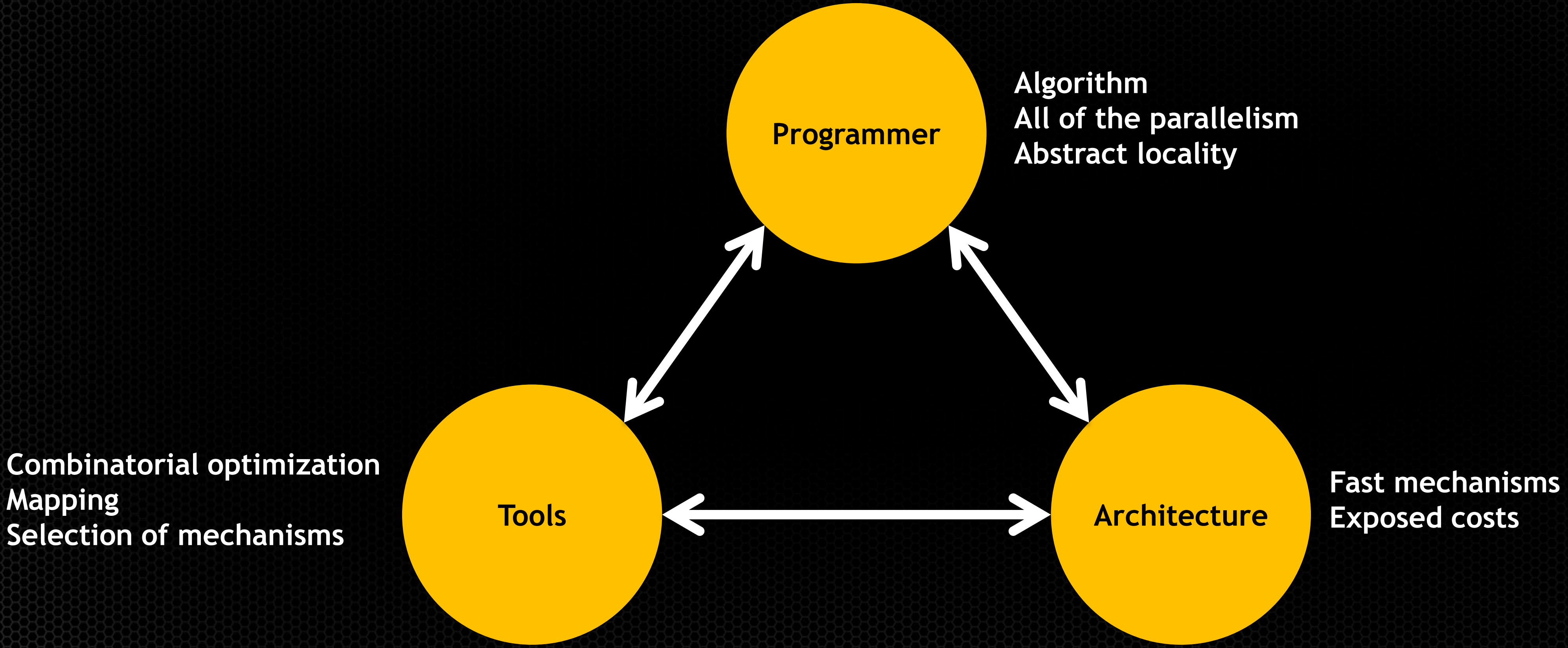


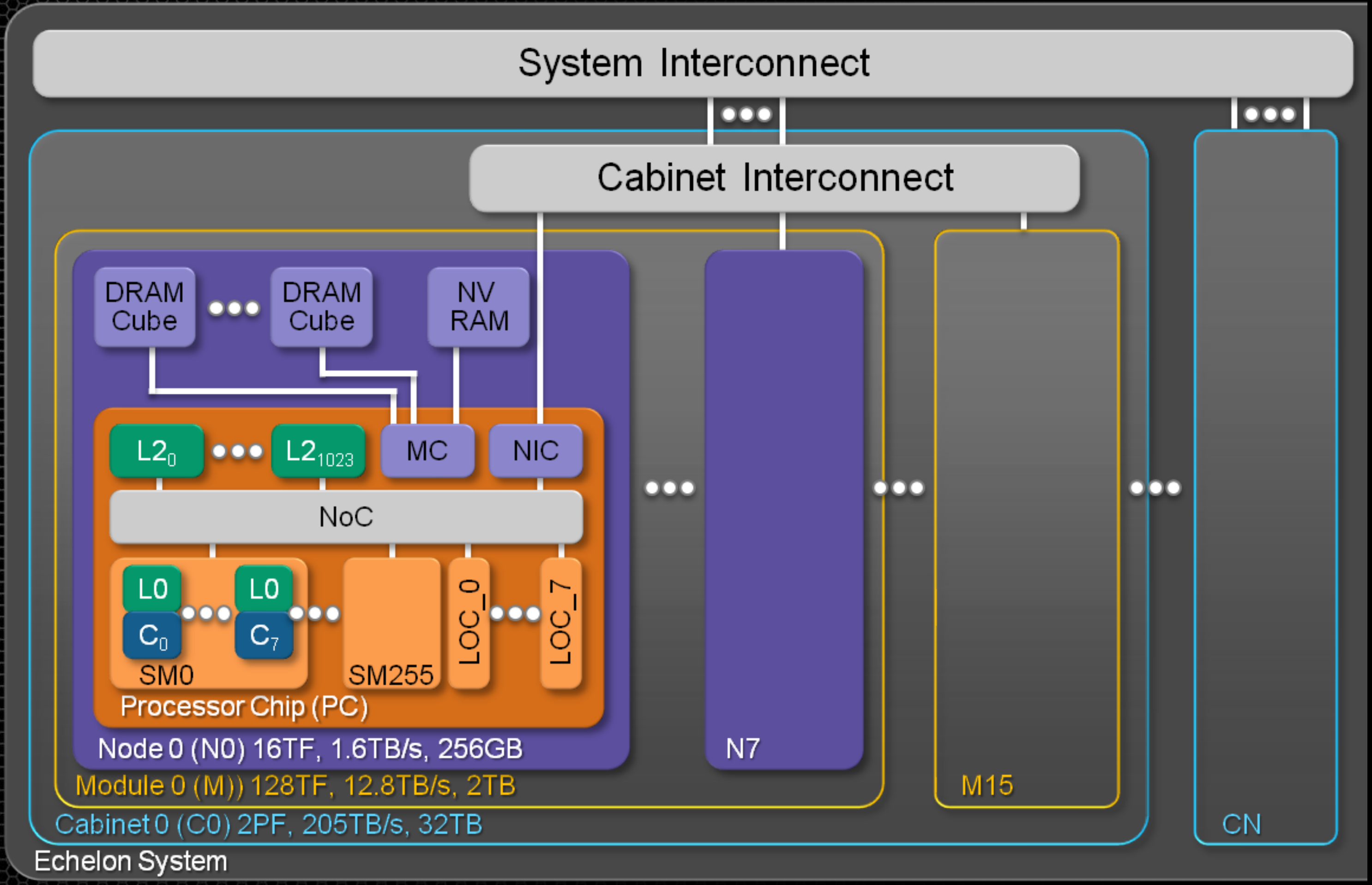
Needed
Process
GFLOPS/W
10
1
2013
2014
2015
2016
2017
2018
2019
2020
ARCHITECTURE
4x
CIRCUITS
3x
PROCESS
2.2x
NVIDIA

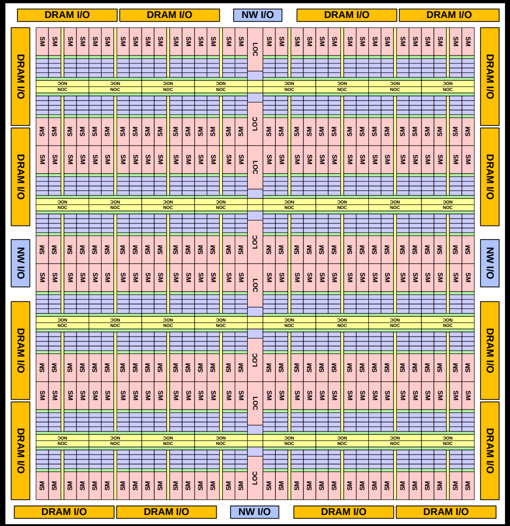
Programmers, Tools, and Architecture Need to Play Their Positions

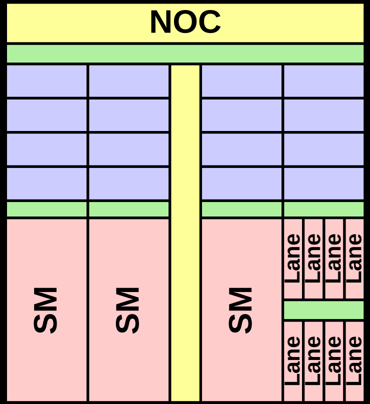


Programmers, Tools, and Architecture Need to Play Their Positions

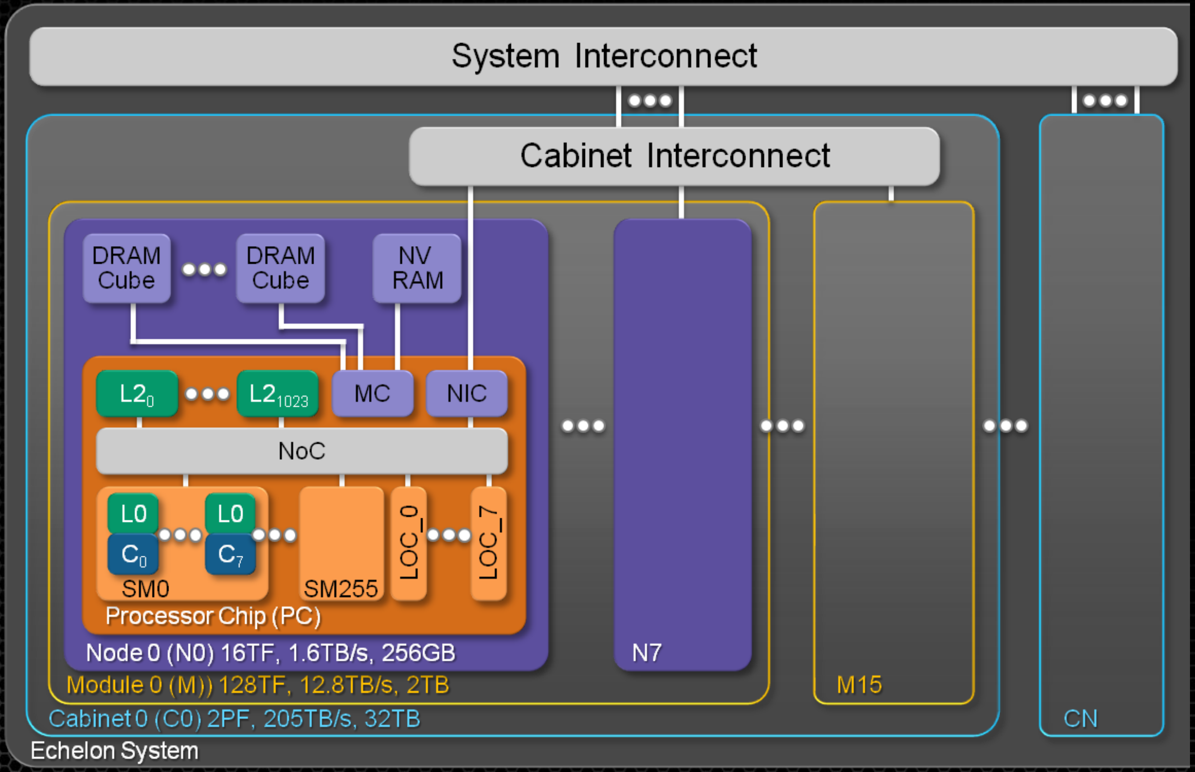



System Interconnect
Cabinet Interconnect
DRAM Cube
DRAM Cube
NV RAM
L2 0
L2 1023
MC
NIC
NoC
L0
C 0
L0
C 7
SM0
SM255
LOC_0
LOC_7
Processor Chip (PC)
Node 0 (N0) 16TF, 1.6TB/s, 256GB
N7
Module 0 (M)) 128TF, 12.8TB/s, 2TB
M15
Cabinet 0 (C0) 2PF, 205TB/s, 32TB
CN
Echelon System


DRAM I/O
NW I/O
DRAM I/O
NW I/O

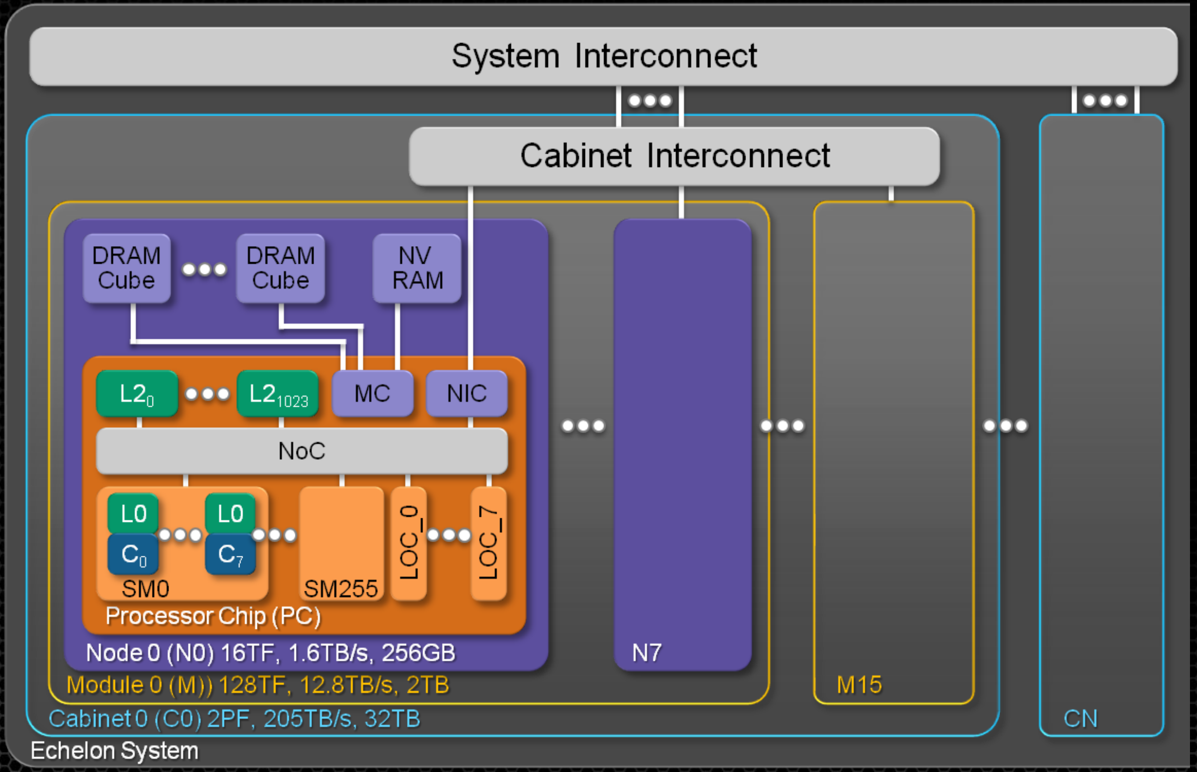

NOC
SM
SM
SM
Lane

An Enabling HPC Network



- <1μs Latency
- Scalable bandwidth
- Small messages 50% @ 32B
- Global adaptive routing
- PGAS
- Collectives & Atomics
- MPI Offload

An Open HPC Network Ecosystem



Common:

- Software-NIC API
- NIC-Router Channel

Processor/NIC Vendors
System Vendors
Networking Vendors

Power

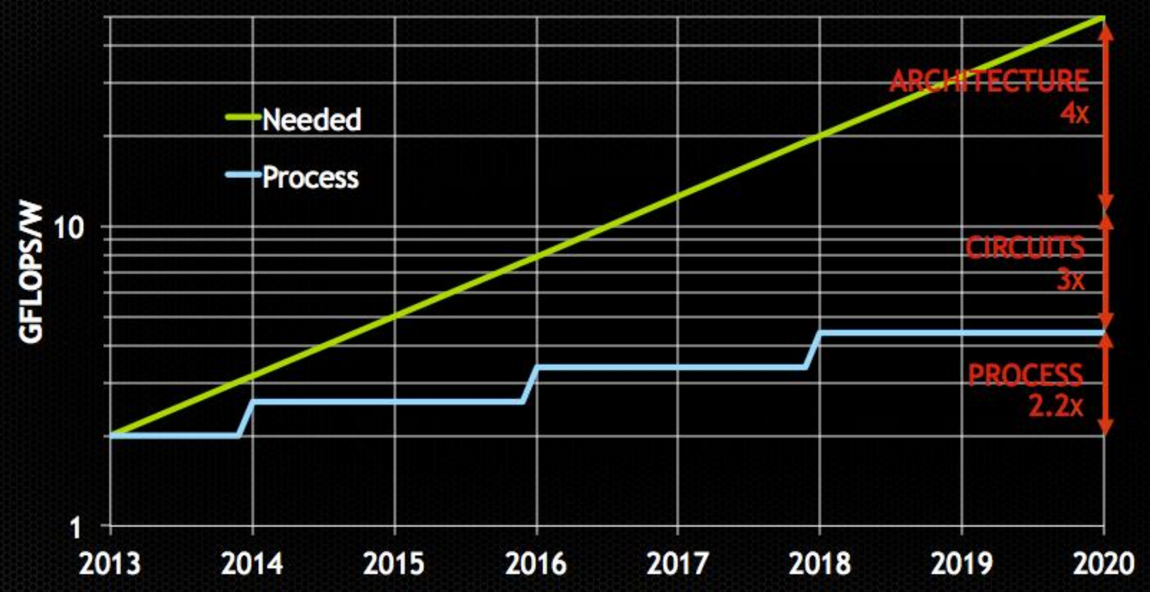
25x Efficiency
with 2.2x from process

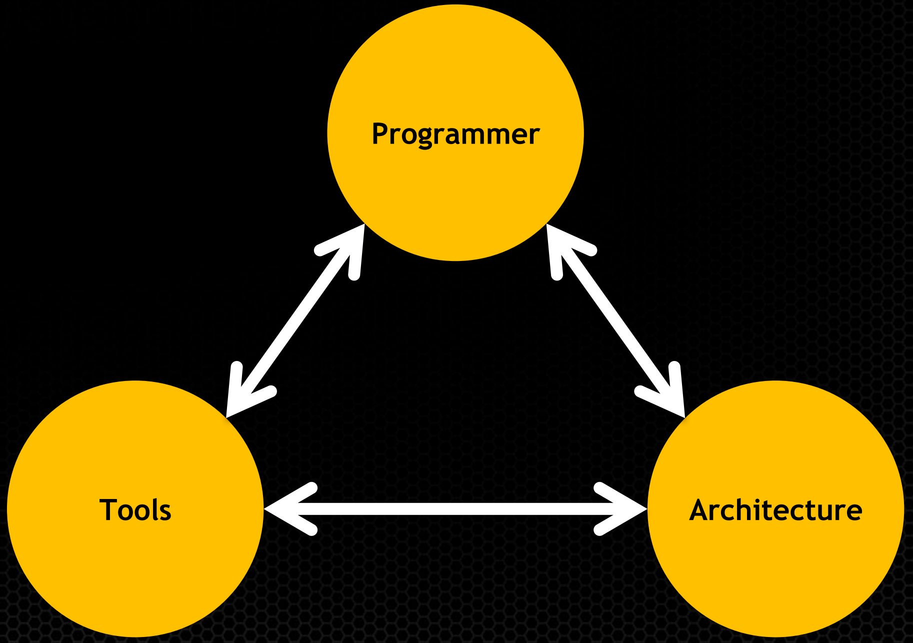
Programming

Parallelism

Heterogeneity

Hierarchy





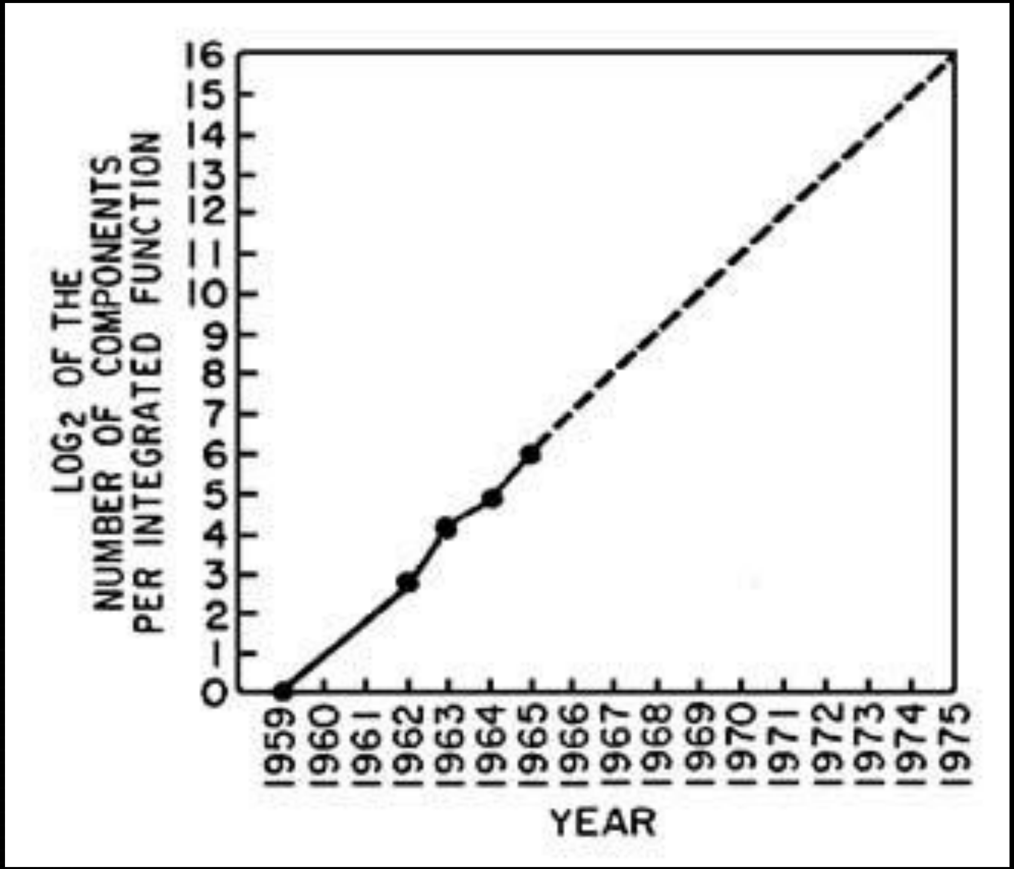
NVIDIA

“Super” Computing

From Super Computers to Super Phones

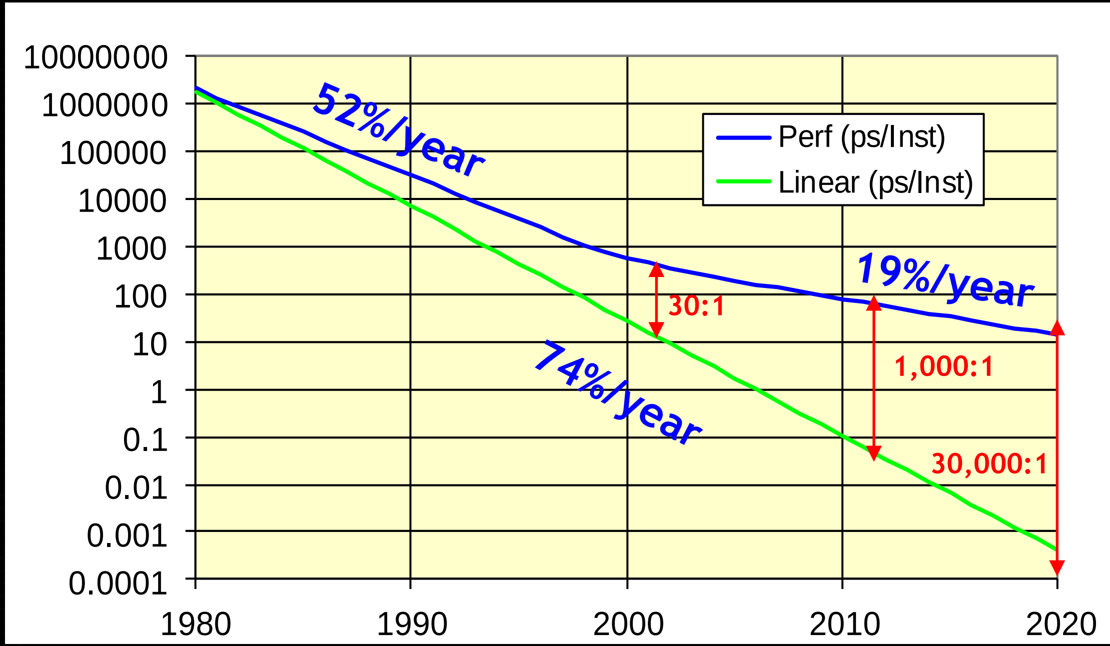
Backup

In The Past, Demand Was Fueled by Moore's Law



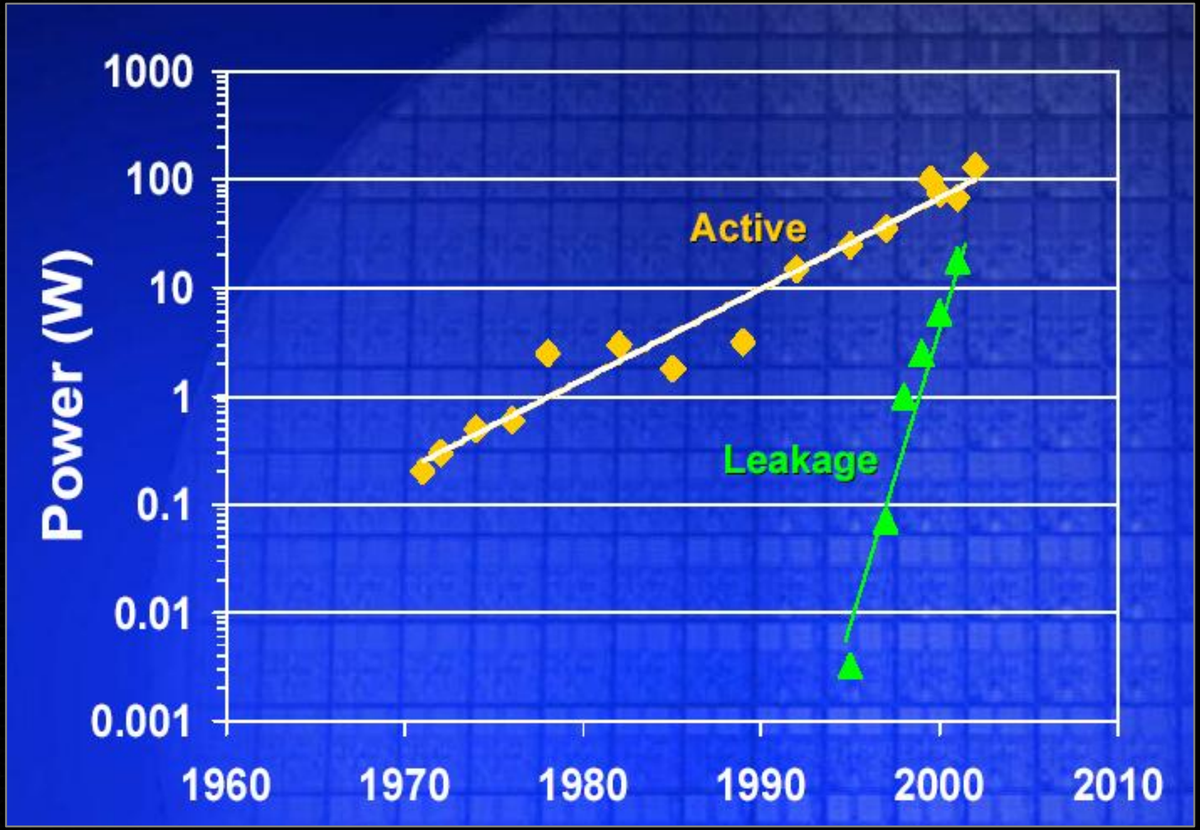
Source: Moore, Electronics 38(8) April 19, 1965

ILP Was Mined Out in 2001



Source: Dally et al. “The Last Classical Computer”, ISAT Study, 2001

Voltage Scaling Ended in 2005



Source: Moore, ISSCC Keynote, 2003

Summary

Moore's law is alive and well, but…

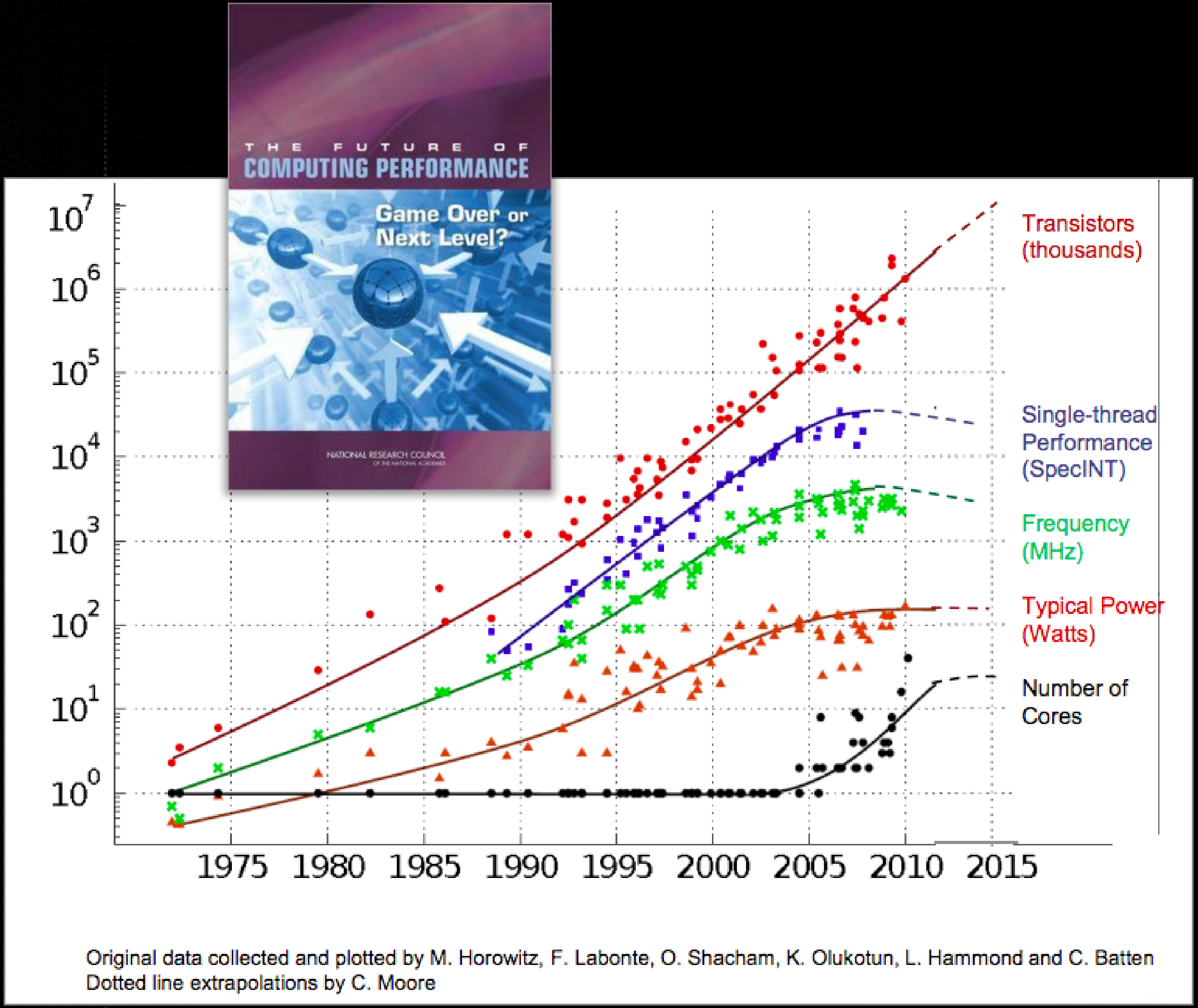
Instruction-level parallelism (ILP) was mined out in 2001

Voltage scaling (Dennard scaling) ended in 2005

Most power is spent on communication

What does this mean to you?

The End of Historic Scaling



Source: C Moore, Data Processing in ExaScale-ClassComputer Systems, Salishan, April 2011

In the Future

All performance is from parallelism

Machines are power limited
(efficiency IS performance)

Machines are communication limited
(locality IS performance)

Two Major Challenges

Energy Efficiency

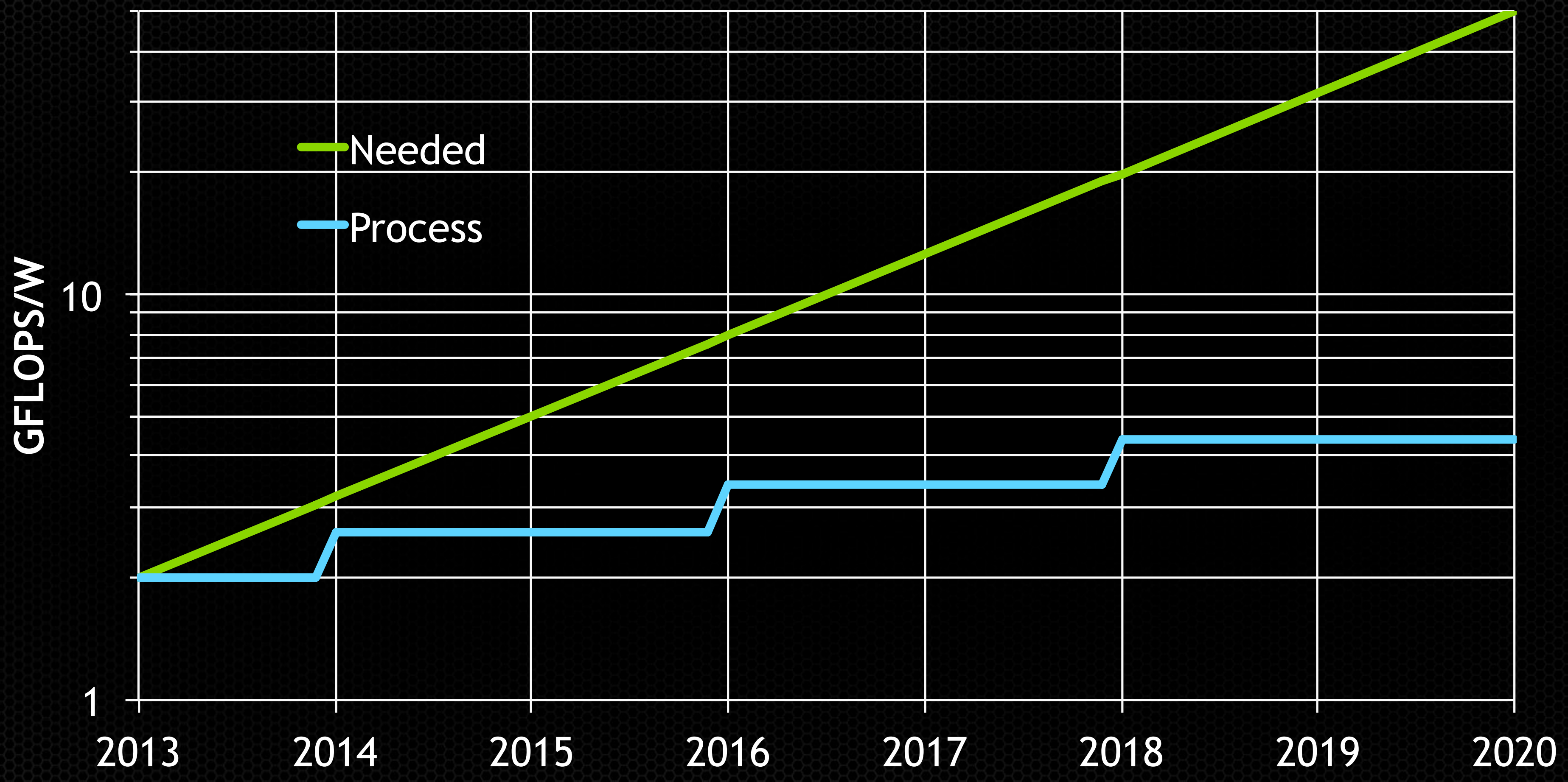
25x in 7 years
(~2.2x from process)

Programming

Parallel (10^{10} threads)

Hierarchical

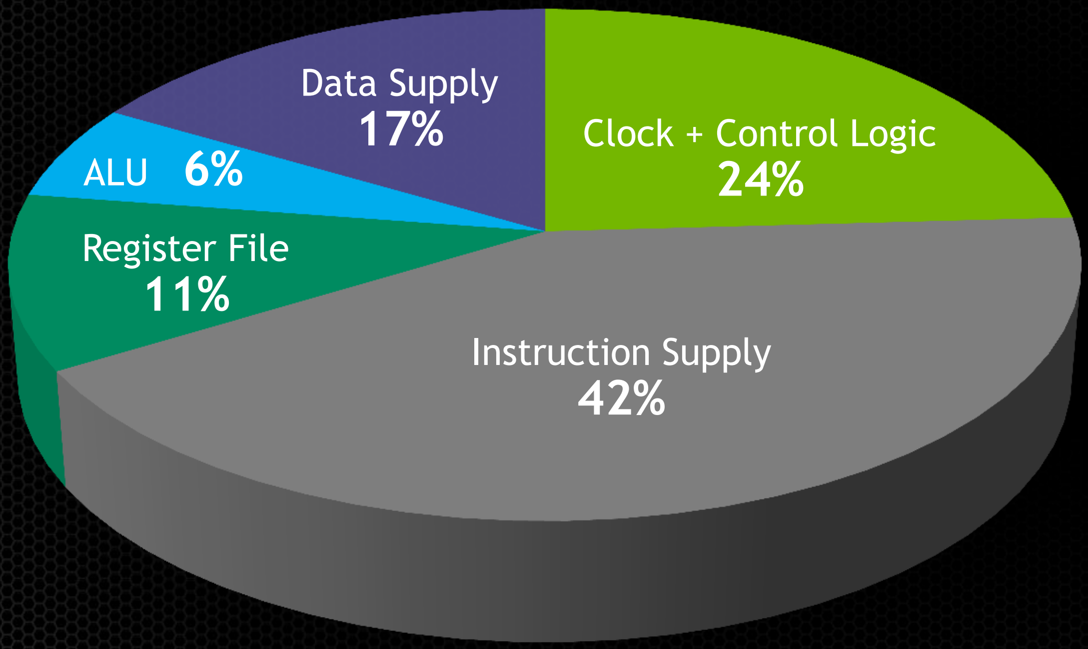
Heterogeneous



Needed
Process
GFLOPS/W
10
1
2013
2014
2015
2016
2017
2018
2019
2020

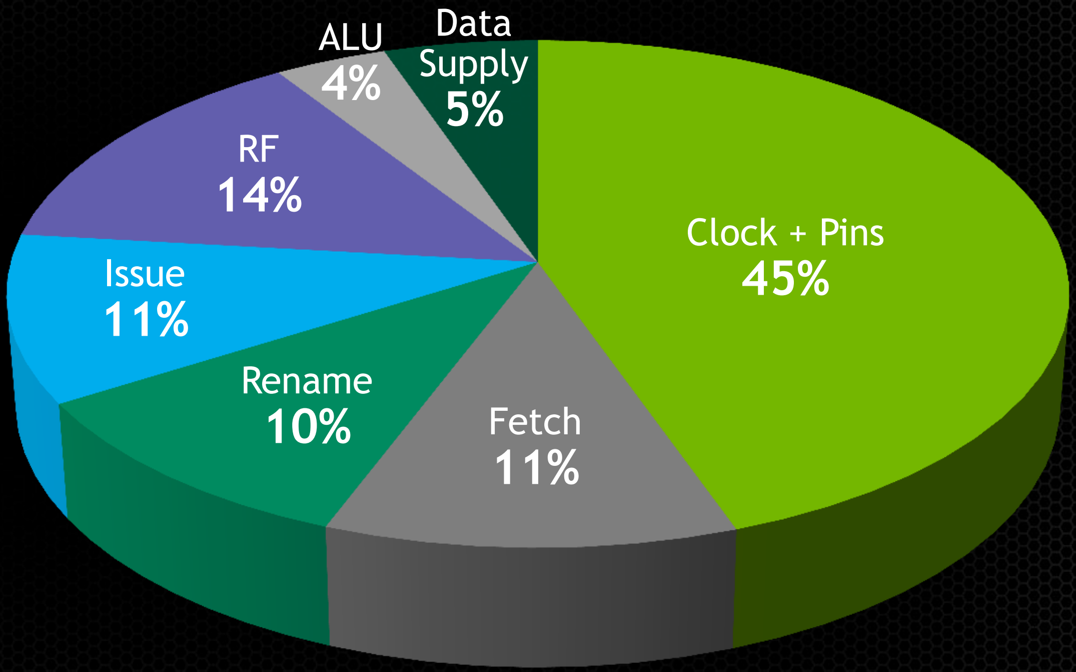
How is Power Spent in a CPU?

In-order Embedded



Dally [2008] (Embedded in-order CPU)

OOO Hi-perf



Natarajan [2003] (Alpha 21264)

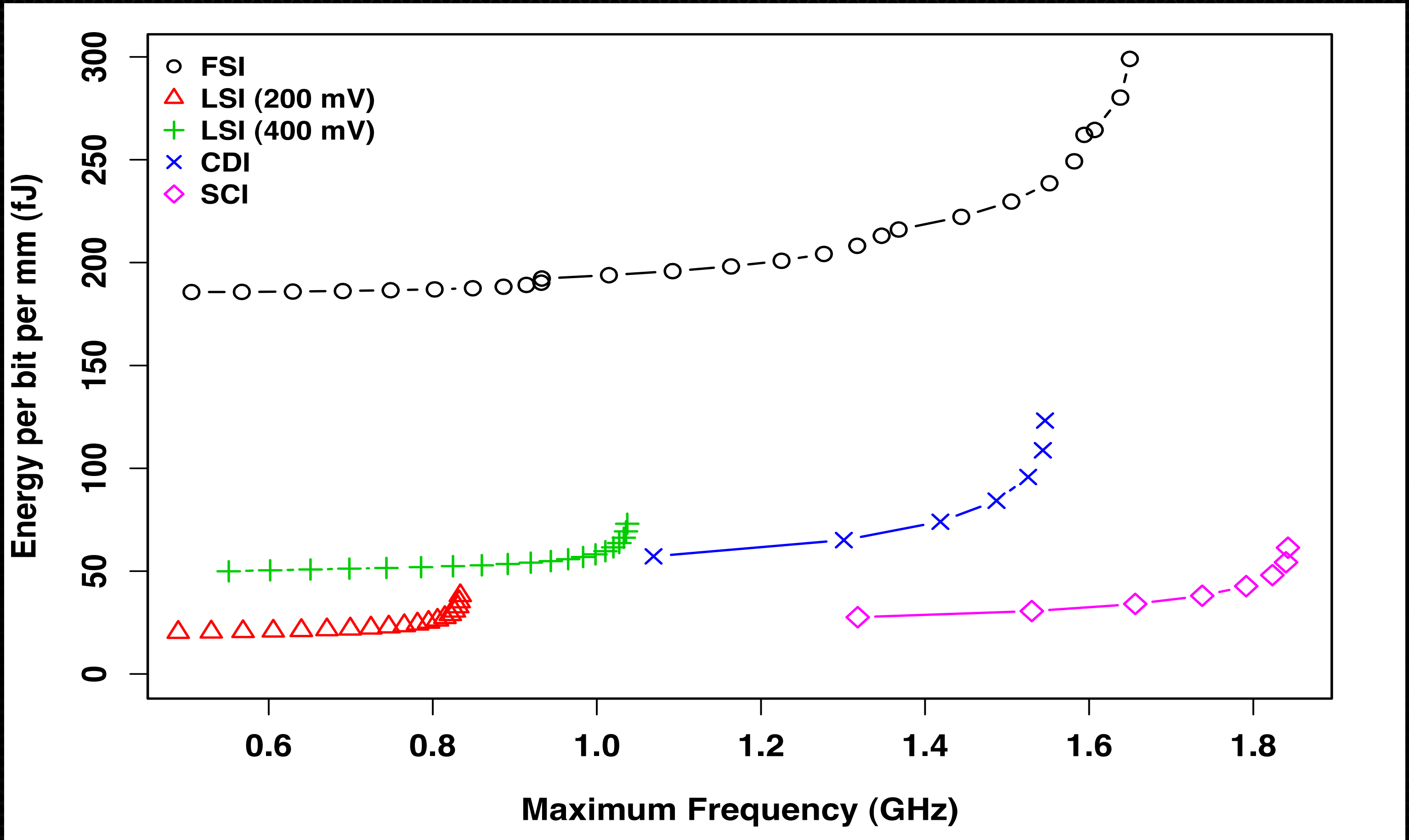
Energy Shopping List

Processor Technology	40 nm	10nm
Vdd (nominal)	0.9 V	0.7 V
DFMA energy	50 pJ	7.6 pJ
64b 8 KB SRAM Rd	14 pJ	2.1 pJ
Wire energy (256 bits, 10mm)	310 pJ	174 pJ

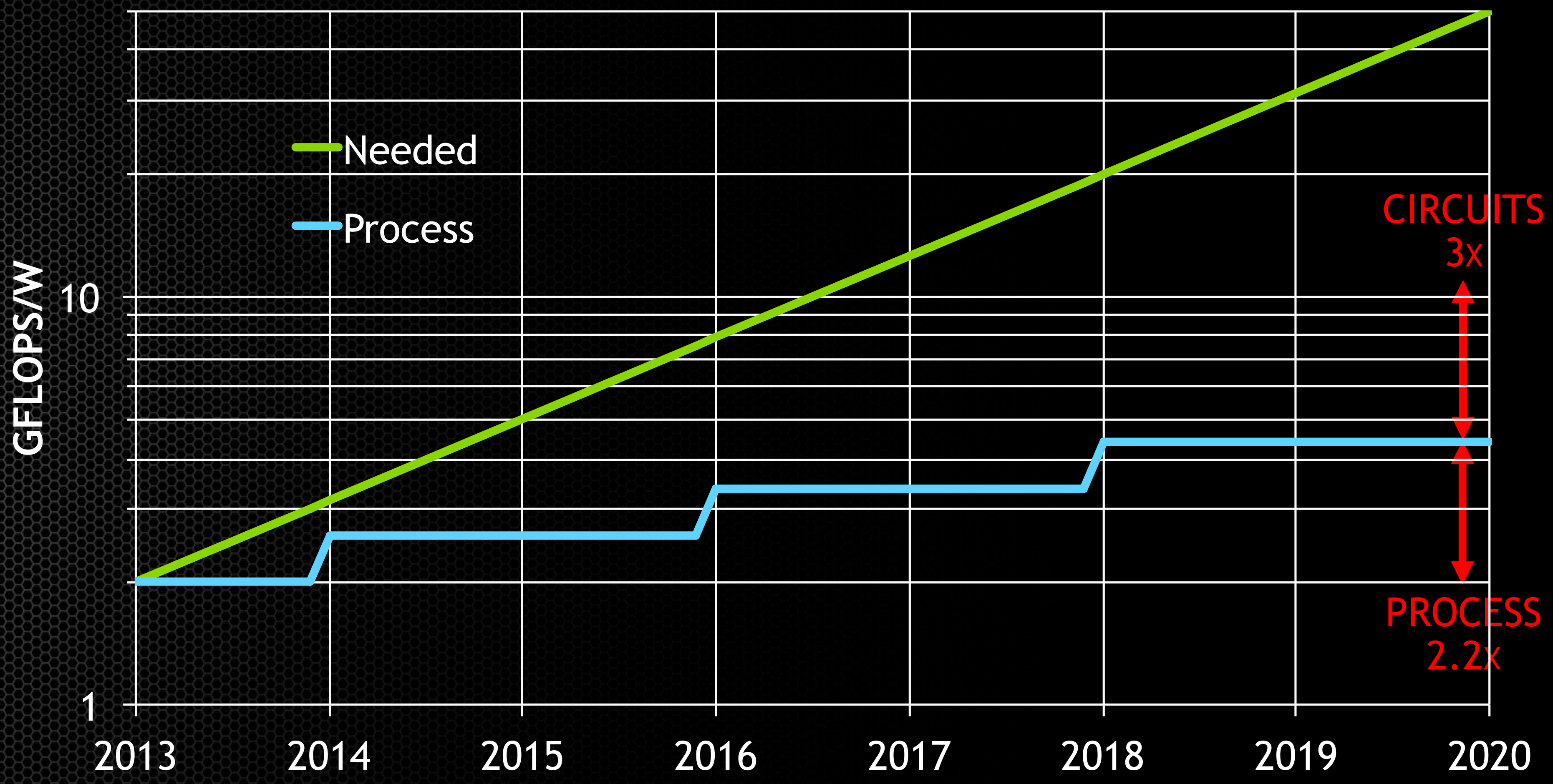
Memory Technology	45 nm	16nm
DRAM interface pin bandwidth	4 Gbps	50 Gbps
DRAM interface energy	20-30 pJ/bit	2 pJ/bit
DRAM access energy	8-15 pJ/bit	2.5 pJ/bit

FP Op lower bound
=
4 pJ

Keckler [Micro 2011], Vogelsang [Micro 2010]

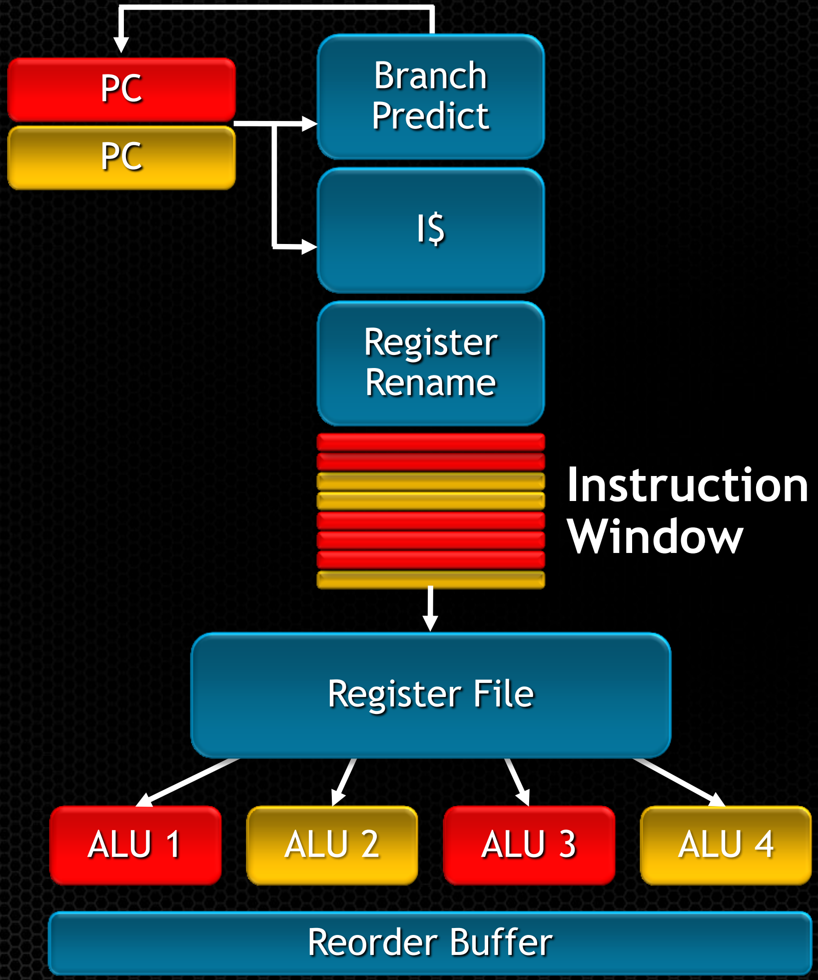


Energy per bit per mm (fJ)
Maximum Frequency (GHz)
FSI
LSI (200 mV)
LSI (400 mV)
CDI
SCI
0
50
100
150
200
250
300
0.6
0.8
1.0
1.2
1.4
1.6
1.8

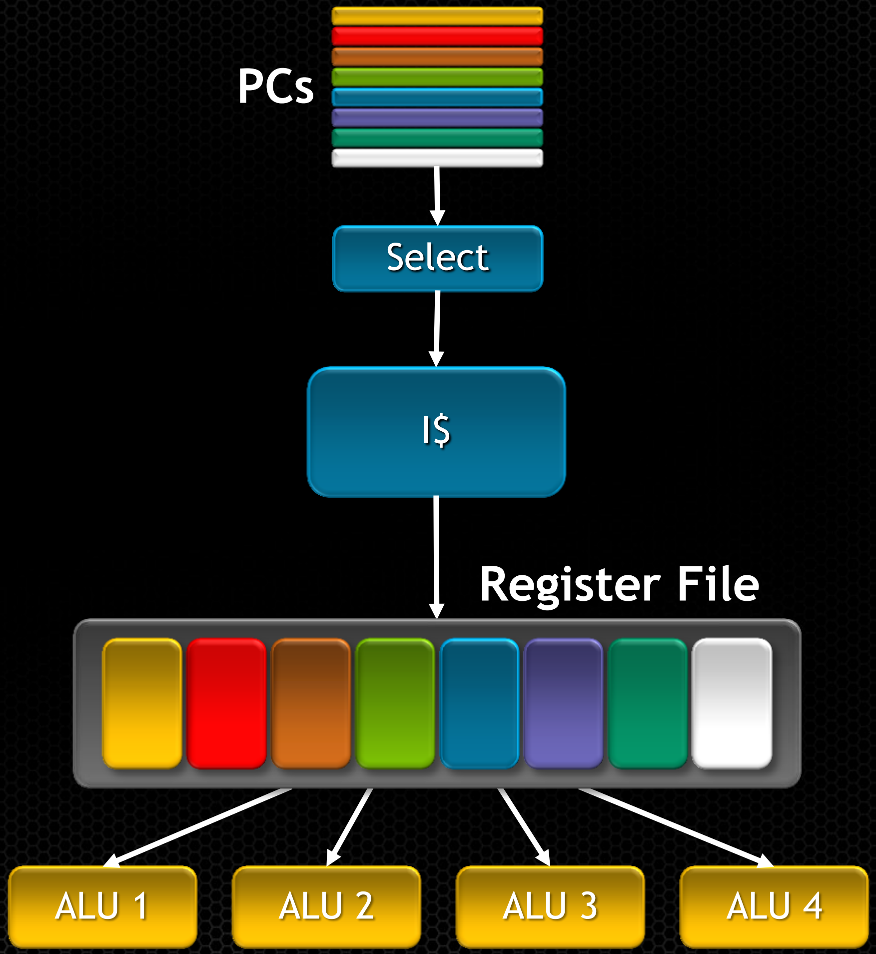


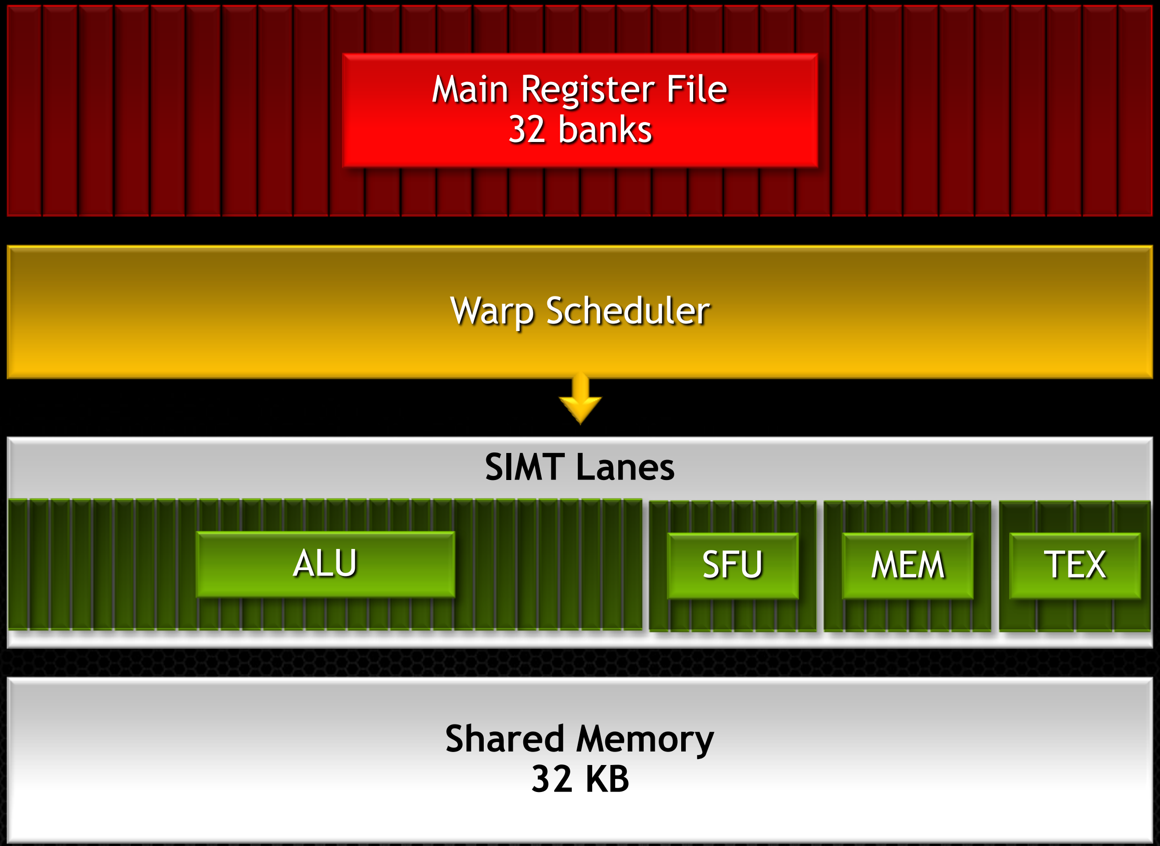
Needed
Process
GFLOPS/W
10
1
CIRCUITS
3x
PROCESS
2.2x
2013
2014
2015
2016
2017
2018
2019
2020

Latency-Optimized Core (LOC)



Throughput-Optimized Core (TOC)

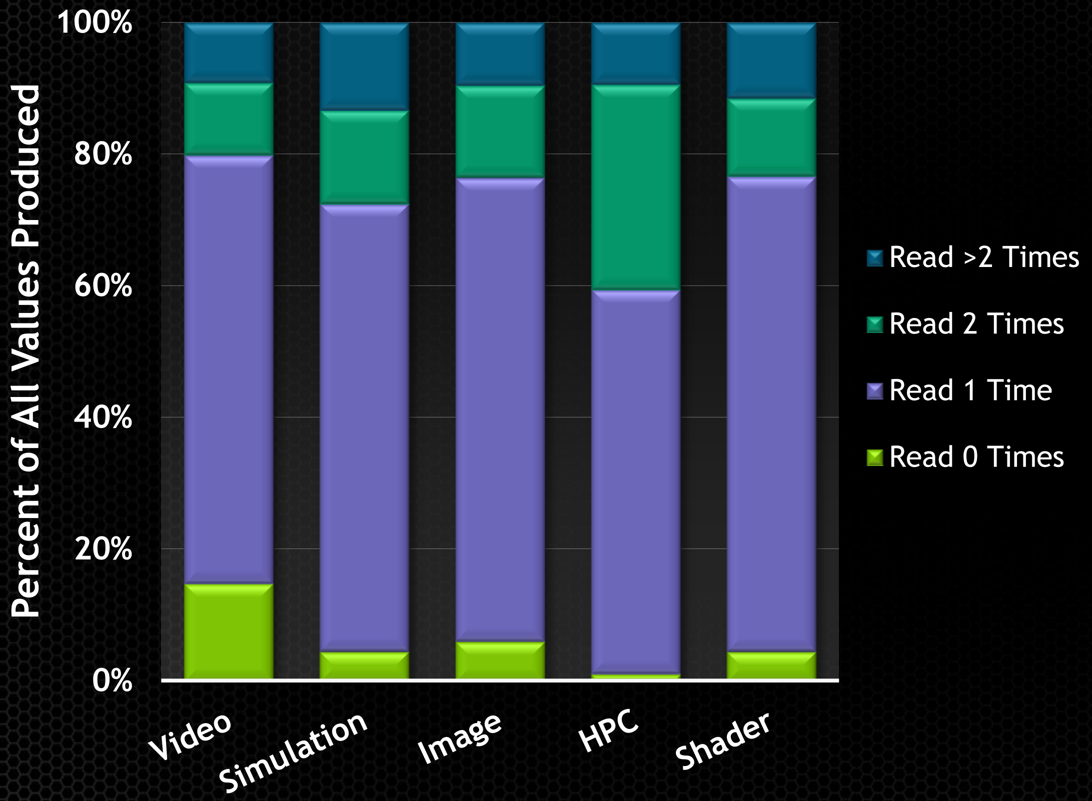


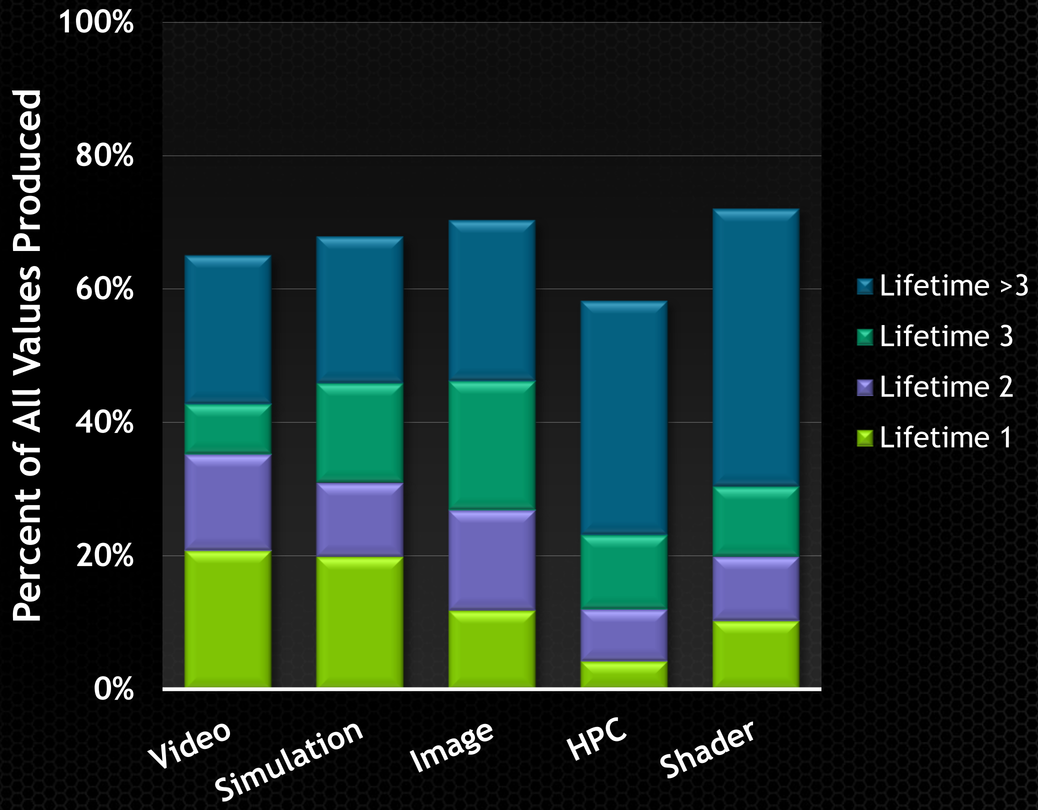


15% of SM Energy

Streaming Multiprocessor (SM)

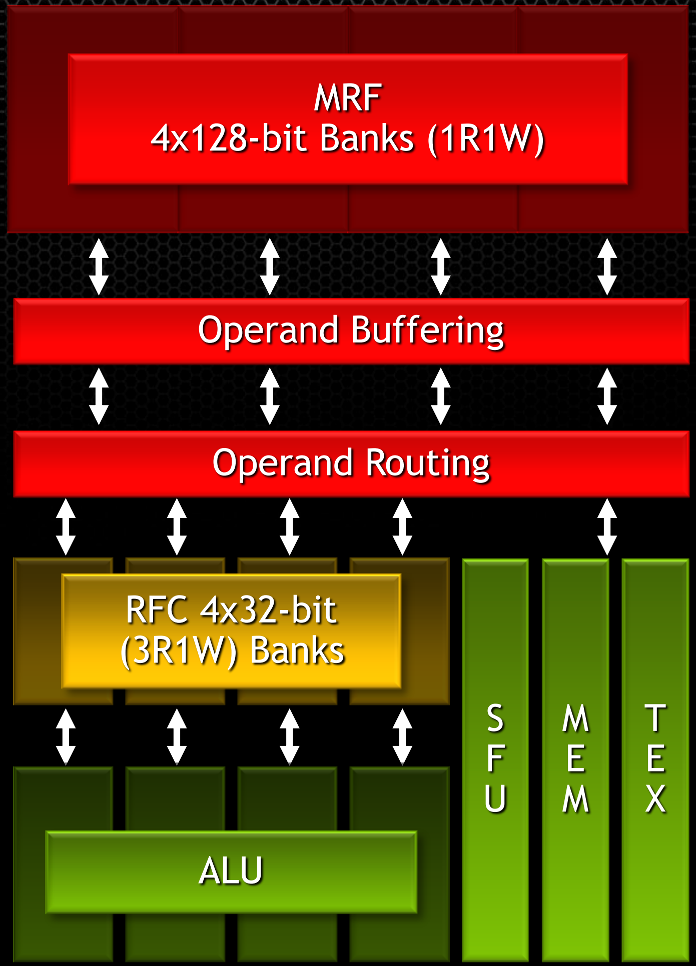
Hierarchical Register File





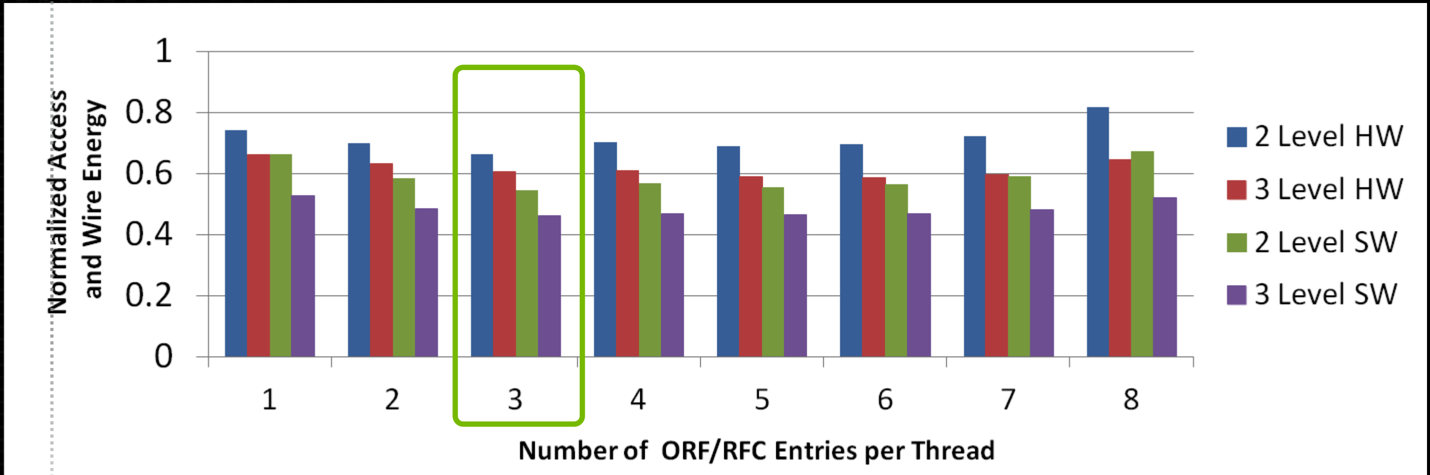
NVIDIA

Register File Caching (RFC)



Energy Savings from RF Hierarchy

54% Energy Reduction



Source: Gebhart, et. al (Micro 2011)

Two Major Challenges

Energy Efficiency

25x in 7 years
(~2.2x from process)

Programming

Parallel (10^{10} threads)

Hierarchical

Heterogeneous

Skills on LinkedIn	*Size (approx)*	*Growth (rel)*
C++	1,000,000	-8%
Javascript	1,000,000	-1%
Python	429,000	7%
Fortran	90,000	-11%
MPI	21,000	-3%
x86 Assembly	17,000	-8%
CUDA	14,000	9%
Parallel programming	13,000	3%
OpenMP	8,000	2%
TBB	389	10%
6502 Assembly	256	-13%

Mainstream Programming

Parallel and Assembly Programming

Source: linkedin.com/skills (as of Jun 11, 2013)

NVIDIA

Parallel Programming is Easy

```
forall molecule in set:                  # 1E6 molecules
    forall neighbor in molecule.neighbors: # 1E2 neighbors ea
        forall force in forces:          # several forces
            # reduction
            molecule.force += force(molecule, neighbor)
```

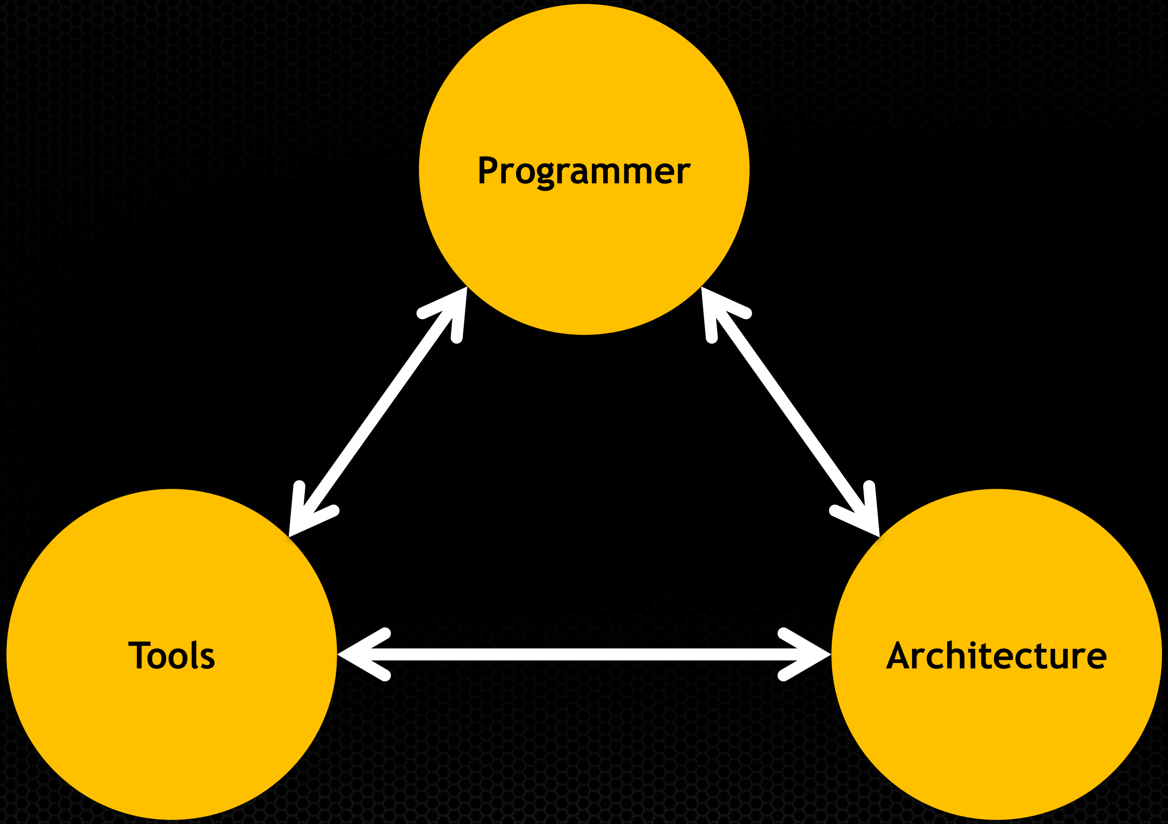
NVIDIA

We Can Make It Hard

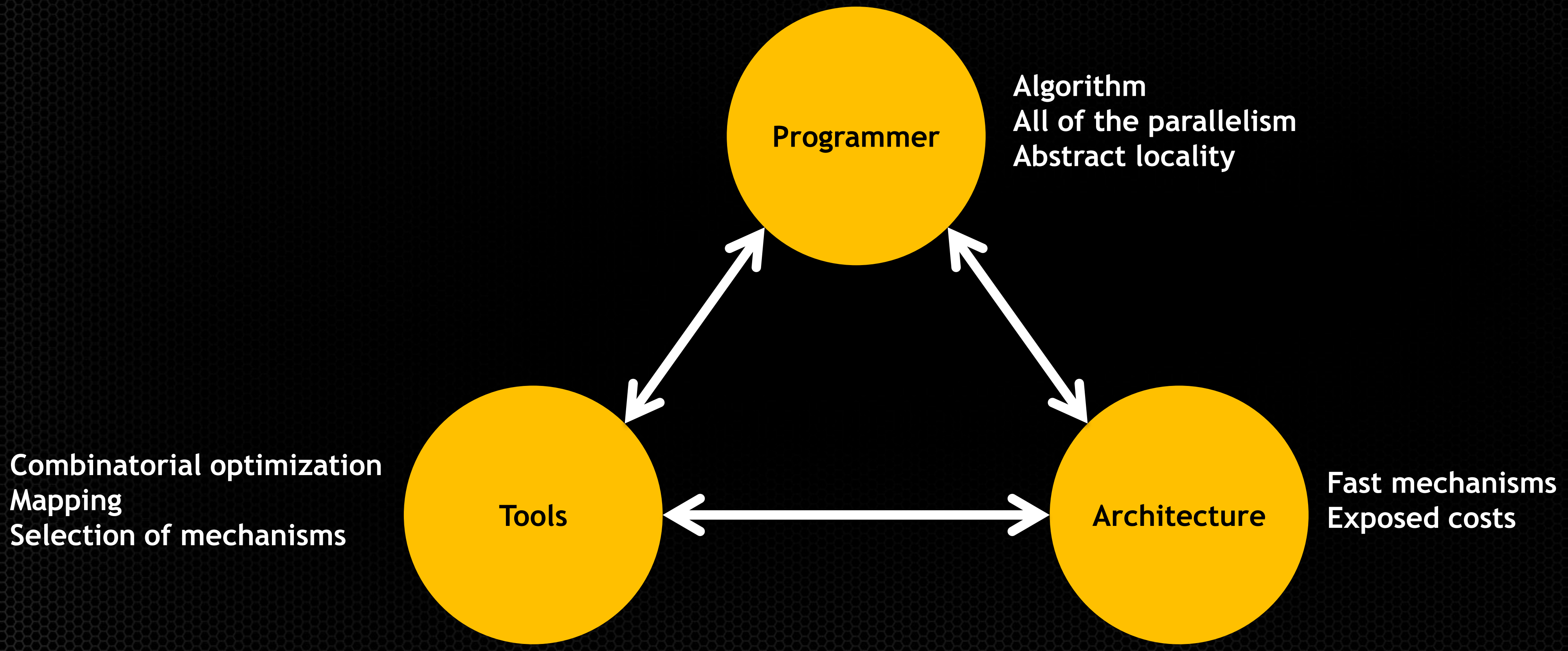
```
pid = fork() ; // explicitly managing threads

lock(struct.lock) ;  // complicated, error-prone synchronization
// manipulate struct
unlock(struct.lock) ;

code = send(pid, tag, &msg) ;  // partition across nodes
```

NVIDIA.

Programmers, Tools, and Architecture Need to Play Their Positions



Programmers, Tools, and Architecture Need to Play Their Positions



OpenACC: Easy and Portable

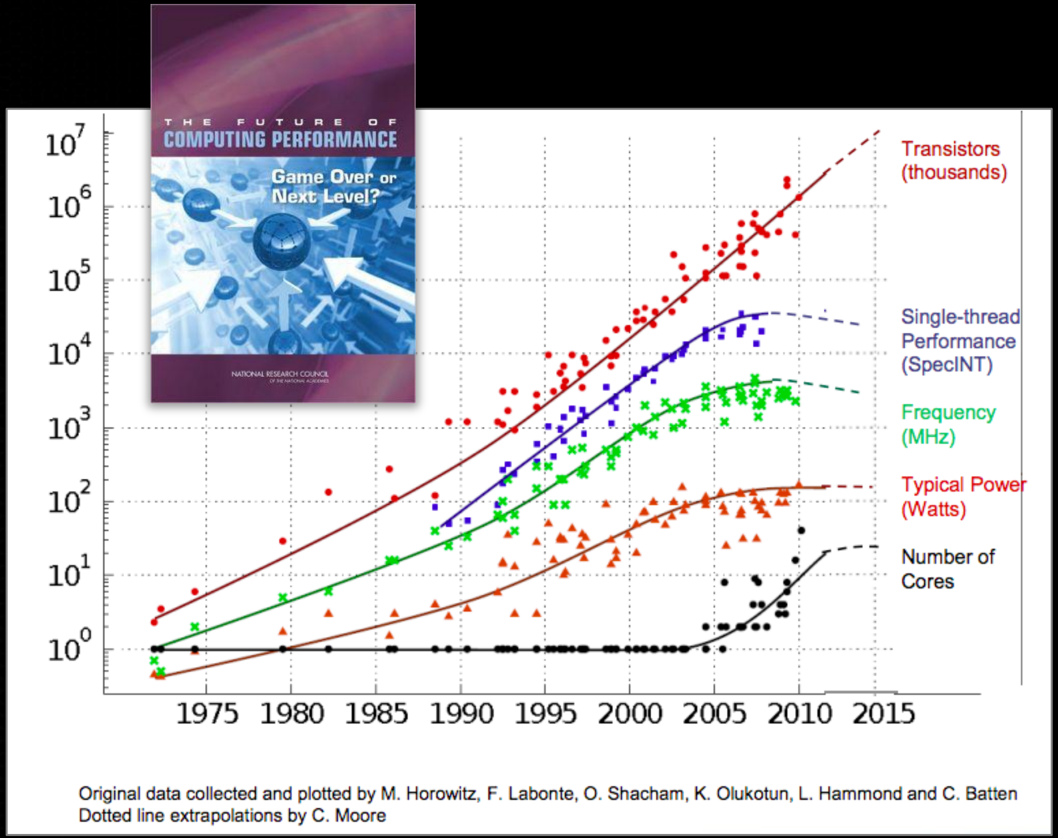
```
do i = 1, 20*128
    do j = 1, 5000000
        fa(i) = a * fa(i) + fb(i)
    end do
end do
```

Serial Code: SAXPY

```
!$acc parallel loop
do i = 1, 20*128
    !dir$ unroll 1000
    do j = 1, 5000000
        fa(i) = a * fa(i) + fb(i)
    end do
end do
```

Conclusion

The End of Historic Scaling



Source: C Moore, Data Processing in ExaScale-ClassComputer Systems, Salishan, April 2011

Parallelism is the source of all performance

Power limits all computing

Communication dominates power

Two Challenges

Power

25x Efficiency
with 2.2x from process

Programming

Parallelism

Heterogeneity

Hierarchy

Power

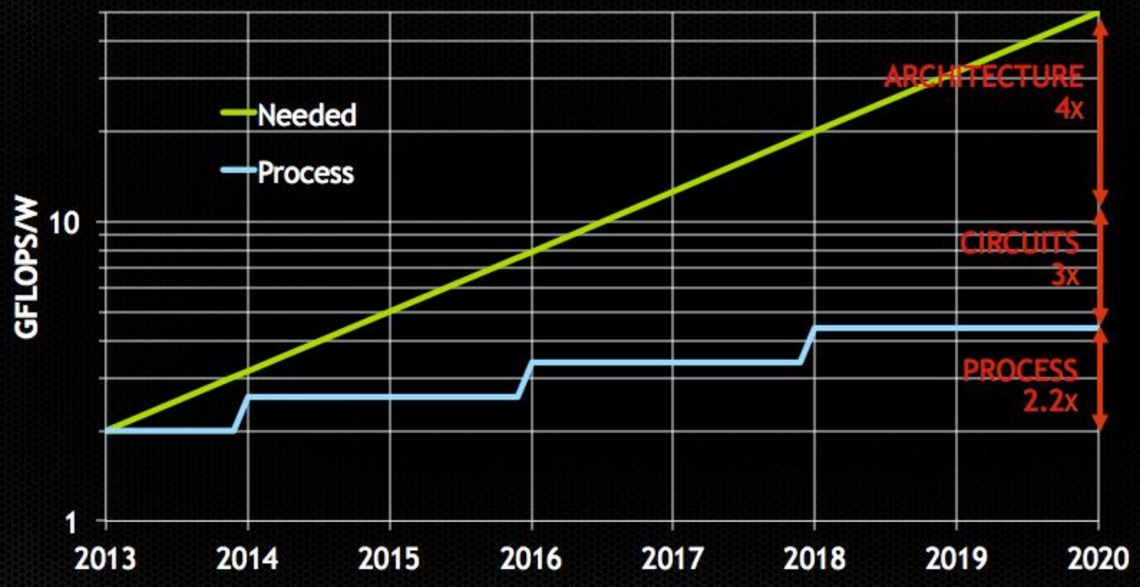
25x Efficiency
with 2.2x from process

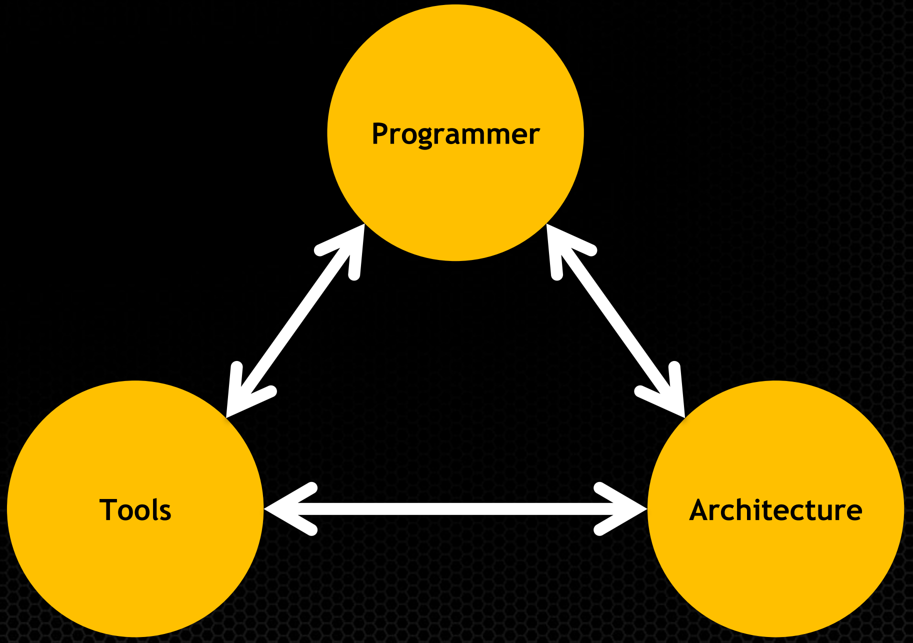
Programming

Parallelism

Heterogeneity

Hierarchy



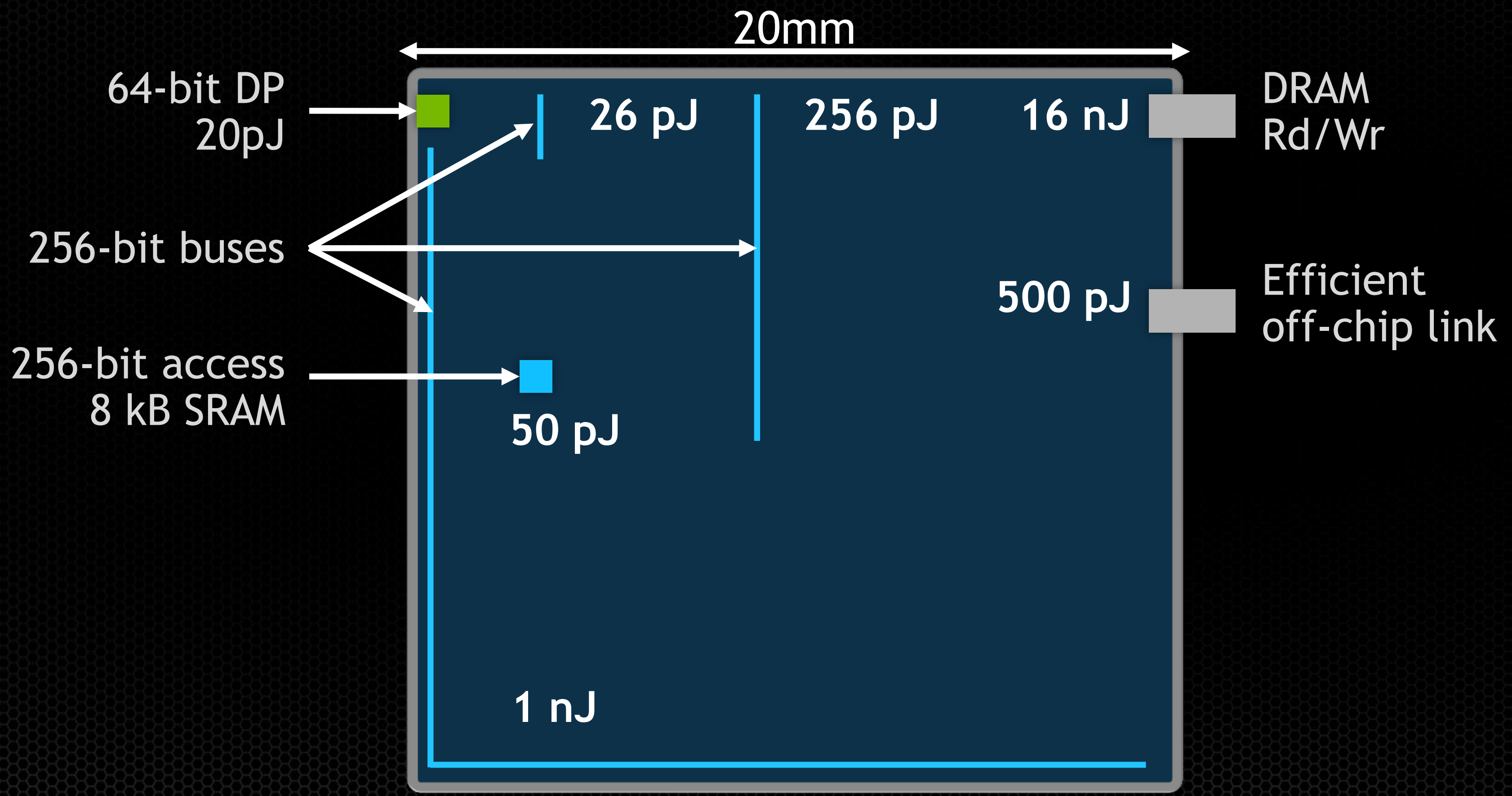


NVIDIA

“Super” Computing

From Super Computers to Super Phones

Communication Takes More Energy Than Arithmetic



Key to Parallelism: Independent operations on independent data

sum(map(multiply, x, x))

every pair-wise multiply is independent
parallelism is permitted

Key to Locality: Data decomposition should drive mapping

Flat computation

```
total = sum(x)
```

vs.

```
tiles = split(x)
partials = map(sum, tiles)
total = sum(partials)
```

Key to Locality:
Data decomposition should drive mapping

```
total = sum(x)
```

vs.

Explicit decomposition

```
tiles = split(x)
partials = map(sum, tiles)
total = sum(partials)
```

NVIDIA