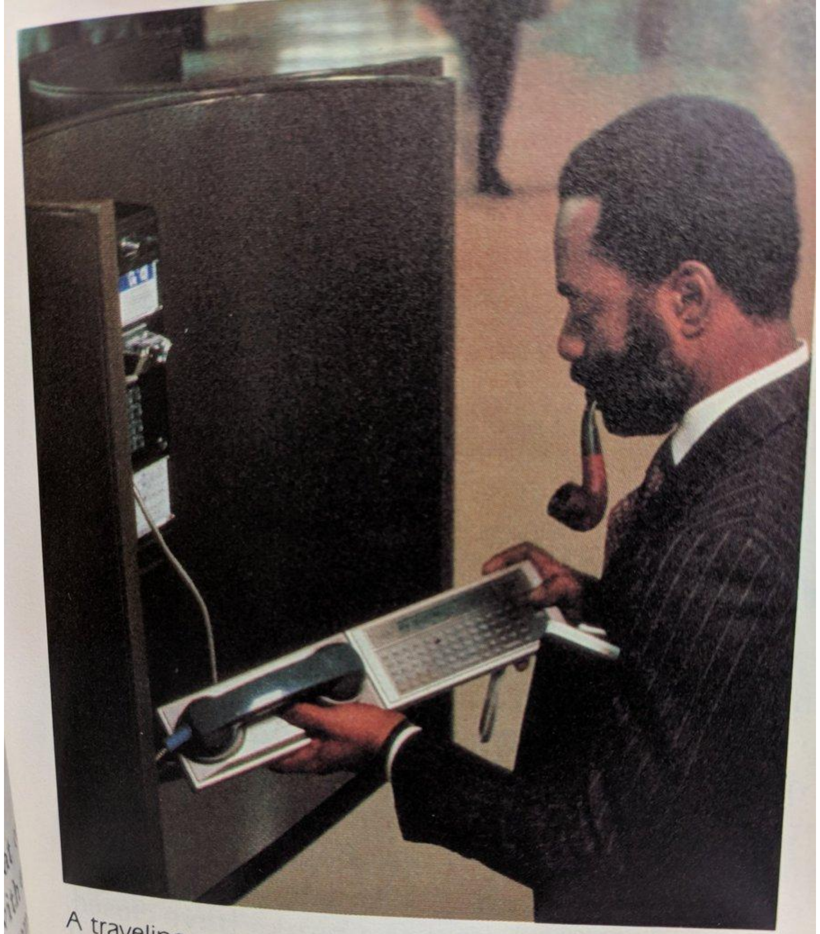
Databases and Technology Trends

David G. Andersen
Carnegie Mellon University
(& Google Brain, but not for this talk)

With key thanks to Michael Kaminsky, Anuj Kalia, Huanchen Zhang, Kim Keeton, Andy Pavlo, Erica Fuchs, and the Brain team.

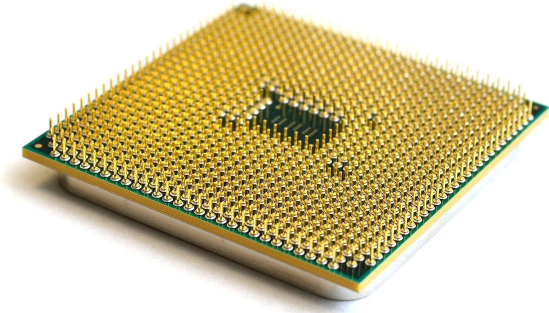


A traveling executive receives messages from his office electronic mail system by means of a hand-held computer and modem at a public telephone.
(Courtesy of RCA)

2018:

~~Moore’s Law~~

Moore’s Zombie



Intel’s Philosophy Prior to 2005

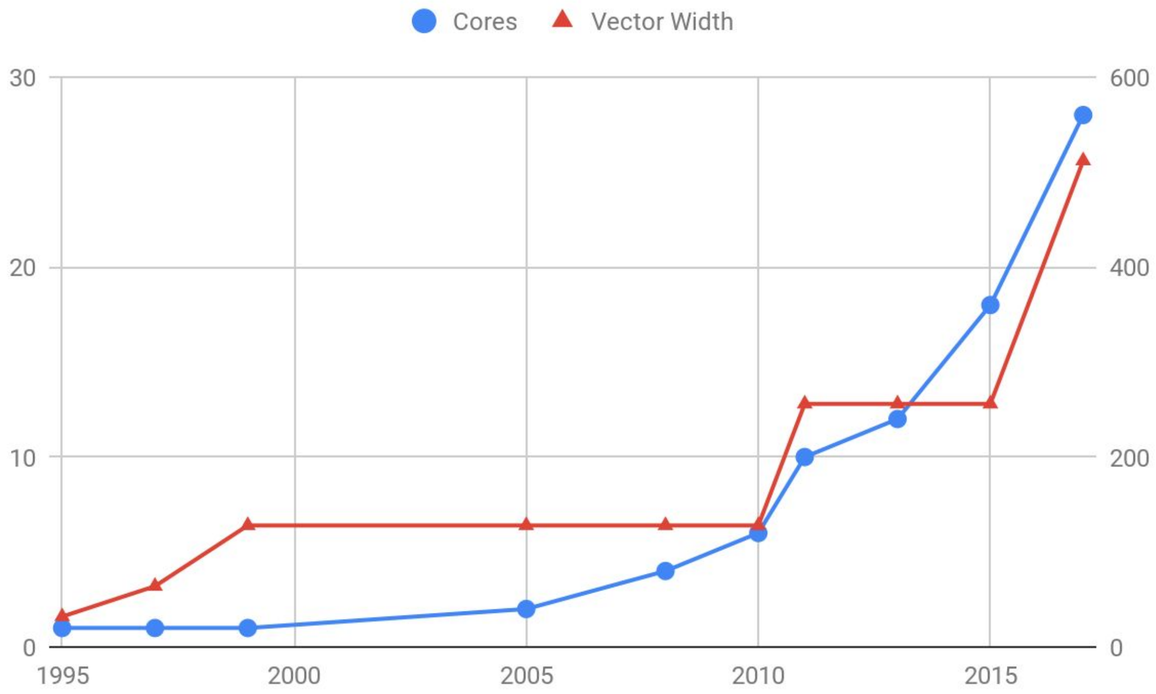
Transparently make existing code run faster

How?

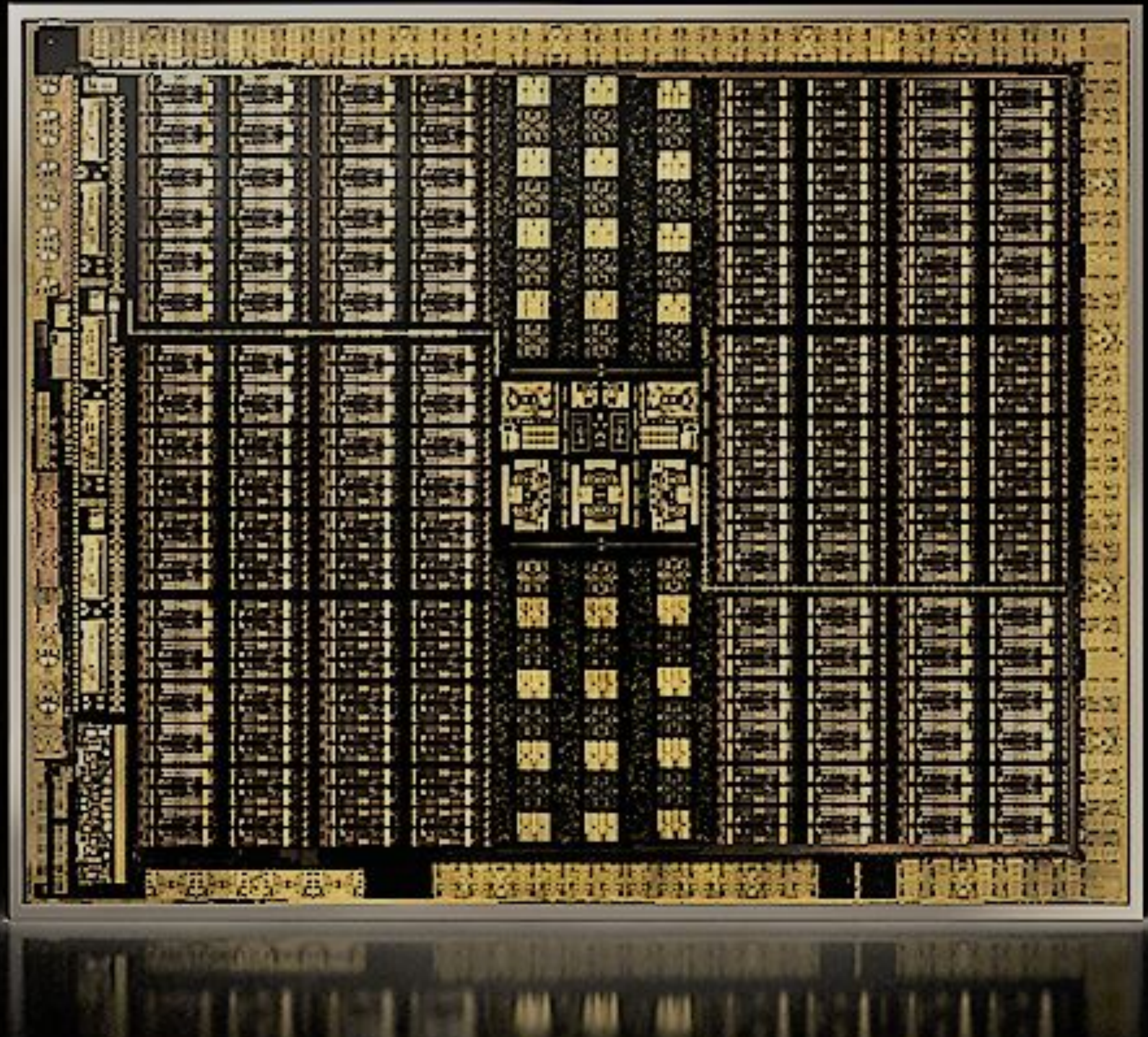
- Higher frequency
    - **Dennard scaling**: Smaller features let you reduce voltage & current
    - **Higher power**
- Better IPC
    - Faster multipliers, branch prediction, prefetching, … architectural magic.

2005: End of Dennard scaling.

2005-2018: Putting Parallelism on the Programmer

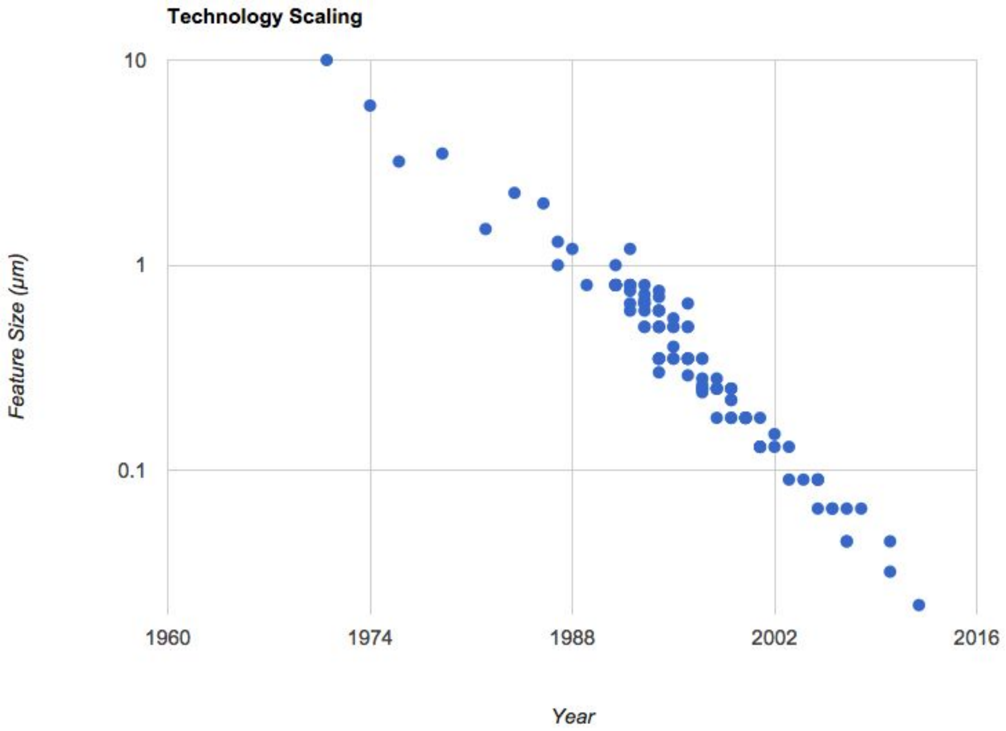


448x




EXPERTS
ONLY

Technology scaling over time



Technology Scaling in 2018

2010: Intel 32nm
2012: Intel 22nm **+2 years**
2015: Intel 14nm **+3 years**
2H 2019: Intel 10nm? **+4 years?**

Intel 7nm, TSMC 5nm use Extreme Ultraviolet Lithography:

2007: ”EUVL may be in pilot production [in 2010]
and in large-scale production [in 2012]”
2016: “EUV may be ready by 2018”
2018: “EUV is currently being developed for high volume use by 2020"

Moore's law as we know it is dead

The **cadence** is dead.

We're not done with all improvements, but they will be slower coming and increasingly irregular.

This is Moore's Zombie.

The “More than Moore” approach

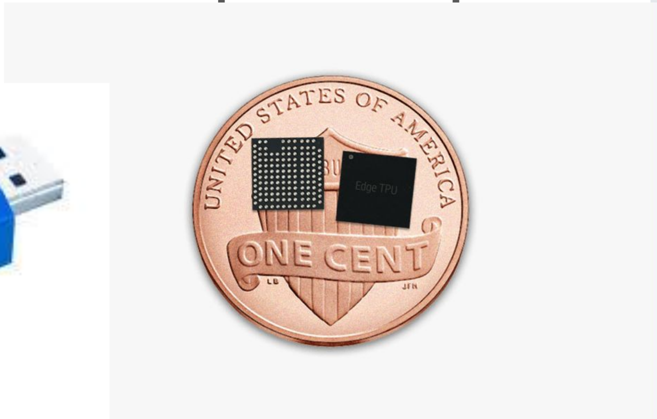
“Functional diversification of semiconductor-based devices”

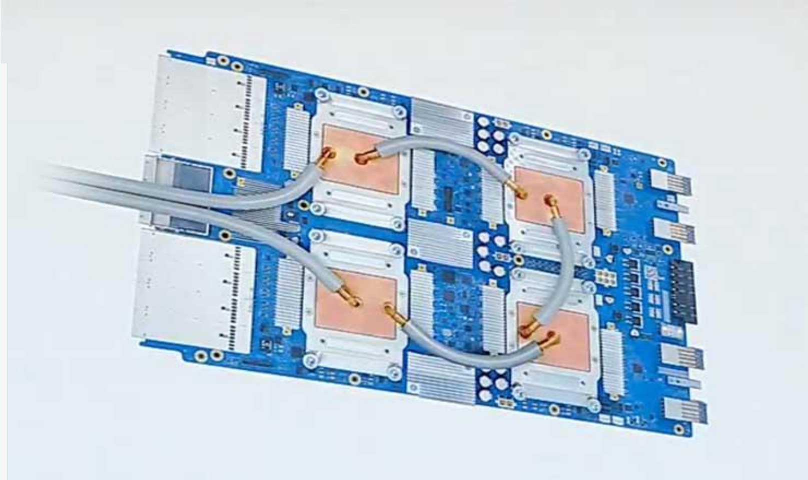
- Integration of sensors, RF, MEMS, quantum?, storage

GlobalFoundries Stops All 7nm Development: Opts To Focus on Specialized Processes

by Anton Shilov & Ian Cutress on August 27, 2018 4:01 PM EST

Flowering of application-specific chips





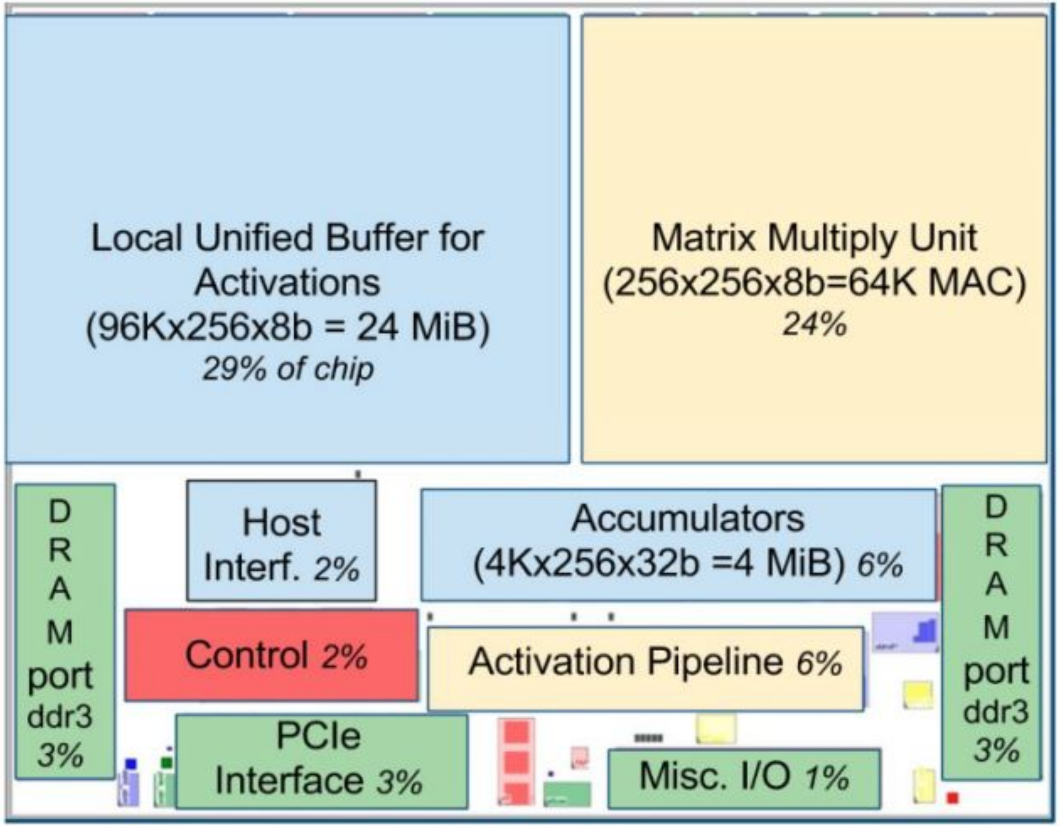
Google TPU v1

30-80x TOPS/Watt vs 2015 CPUs and GPUs

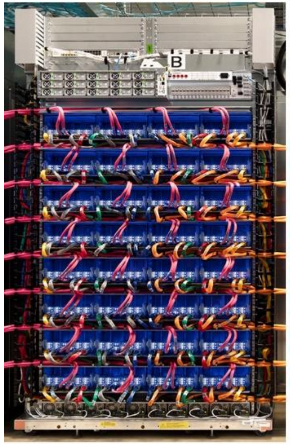
8GiB DRAM

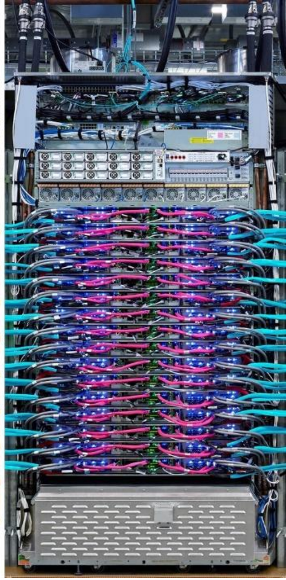
8-bit fixed point

256x256 MAC unit



ASICs in the wild at gigacorps

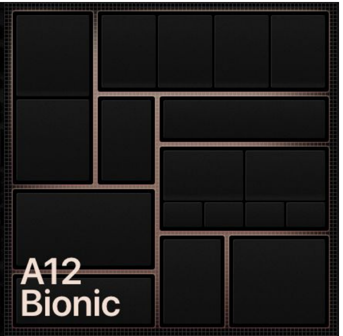




Google: TPUv1, TPUv2, TPUv3

Amazon: Hypothesized AI chip,
Custom VM controller,
Custom switching chip

Apple: Its own ARM series;
Custom AI chip



Vertically-integrated industry giants creating workload-optimized ASICs


Tirols Ski-Dimension
Serfaus·Fiss·Ladis
Direttissima
Kamikaze
SKIROUTE
70%
only experts
Schönjochbahn-Mitte

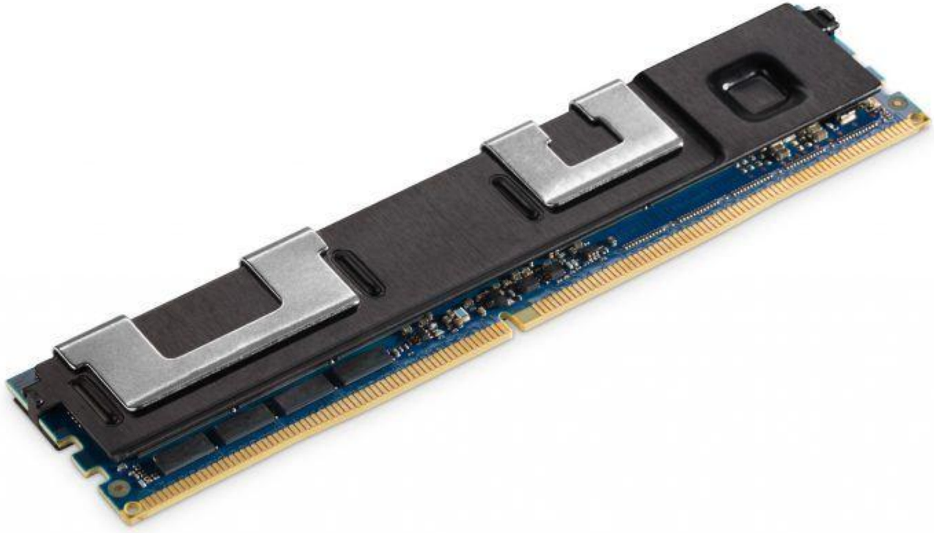
We will have more advances - but they're bumpy

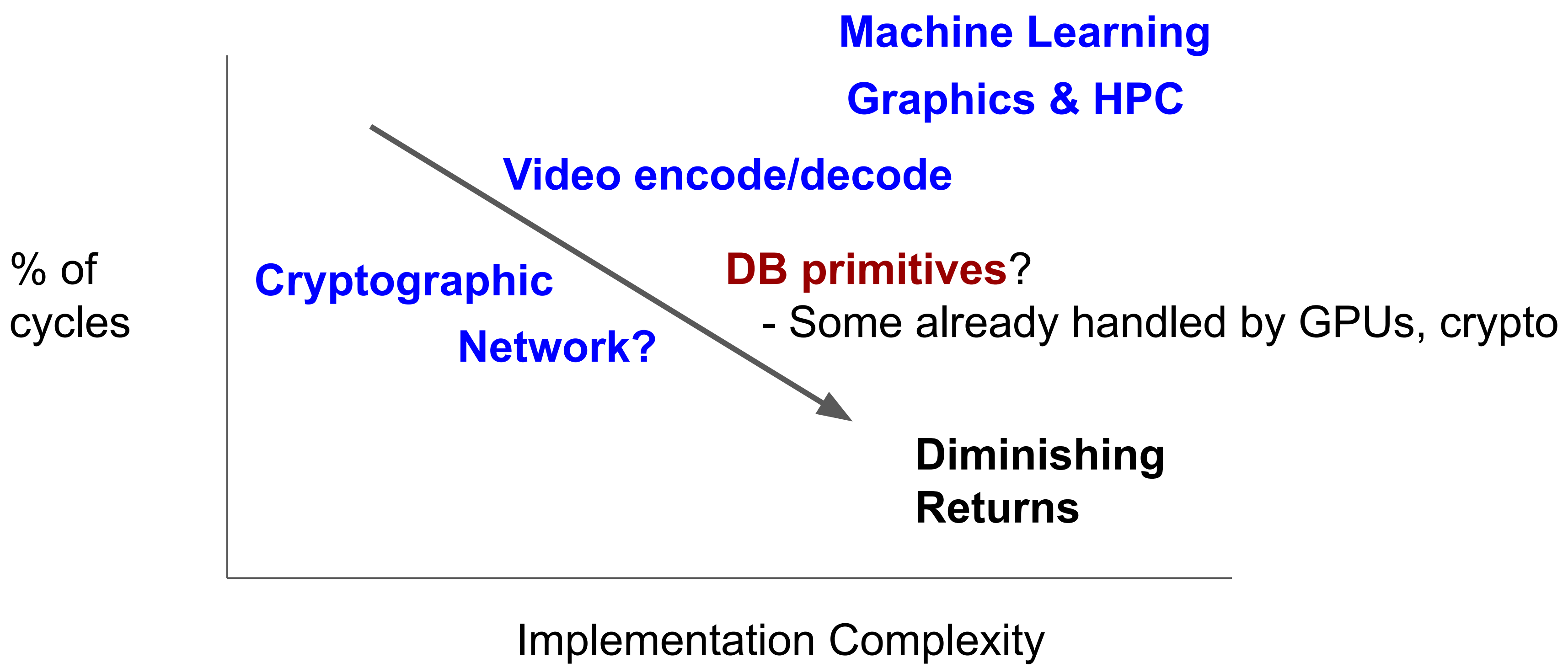
Intel "Apache Pass" persistent memory

(likely a phase-change memory)

… a one-time advance.

Optical interconnects would
Improve DRAM bandwidth/reach
… a one-time advance.




Machine Learning
Graphics & HPC
Video encode/decode
% of cycles
Cryptographic
Network?
DB primitives?
- Some already handled by GPUs, crypto
Diminishing Returns
Implementation Complexity

Can’t outrun Amdahl

Moore:

Most CPU functions got faster simultaneously;

Memory density scaled too!

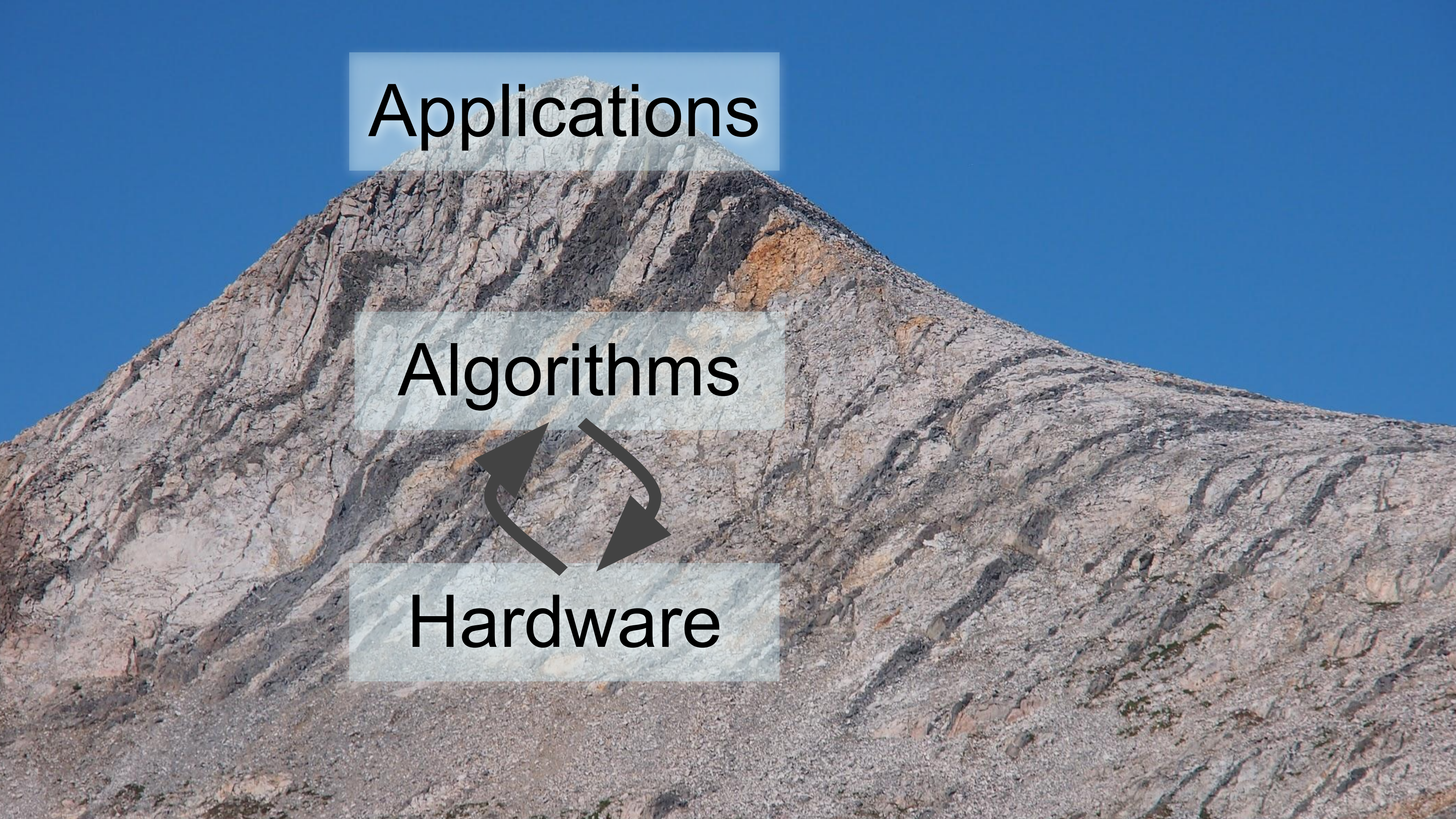
--> I/O primary bottleneck to work around.

Multicore:

+ Parallelization bottleneck

Post-Moore:

+ **Specialization bottleneck**

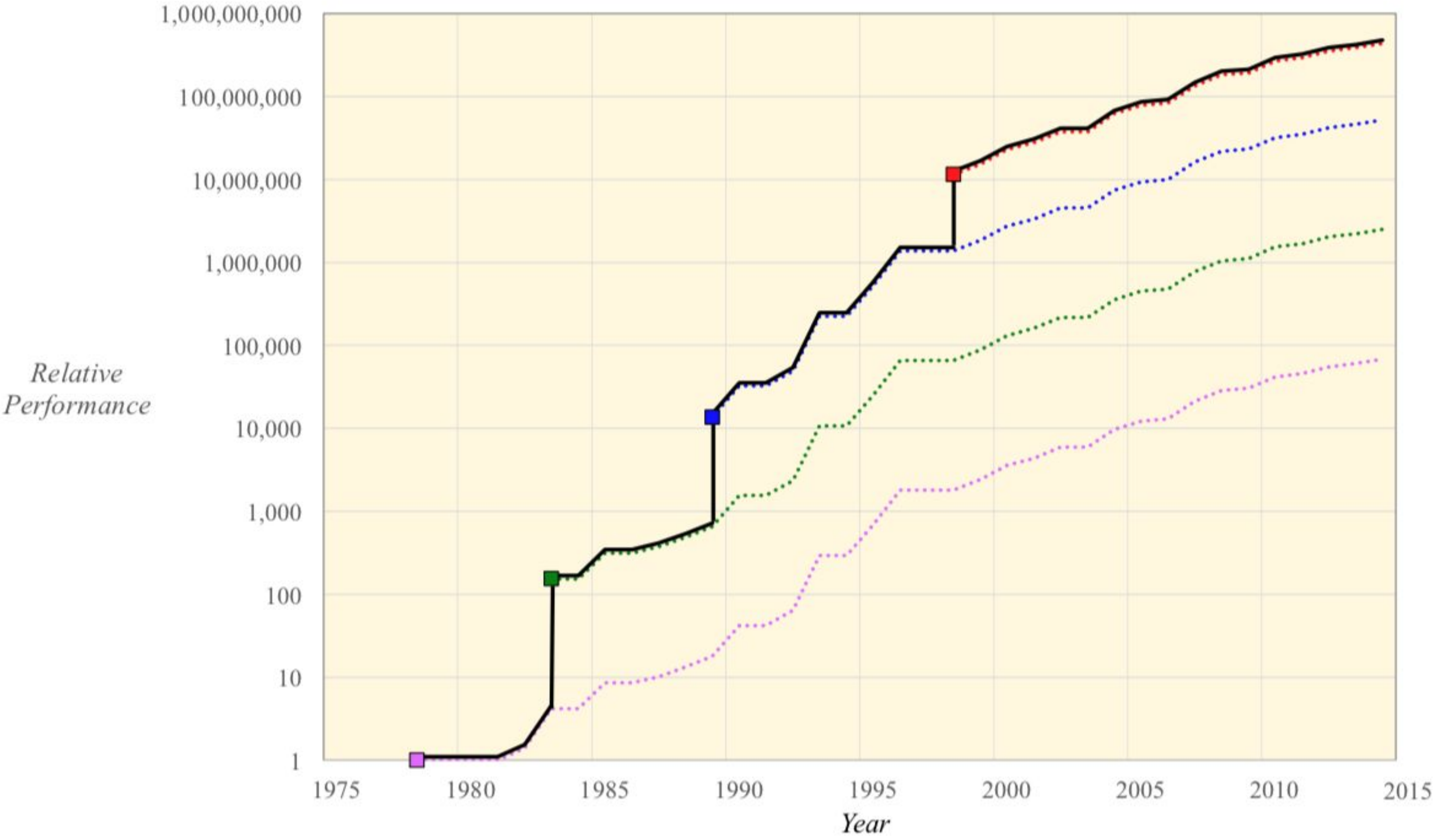

Applications
Algorithms
Hardware

4096 x 4096 matrix multiply

Implementation	*Running time (s)*	*GFLOPS*	*Absolute speedup*
Python	25,552.48	0.005	1
Java	2,372.68	0.058	11
C	542.67	0.253	47
Parallel loops	69.80	1.969	366
Parallel divide-and-conquer	3.80	36.180	6,727
+ vectorization	1.10	124.914	23,224
+ AVX intrinsics	0.41	337.812	62,806

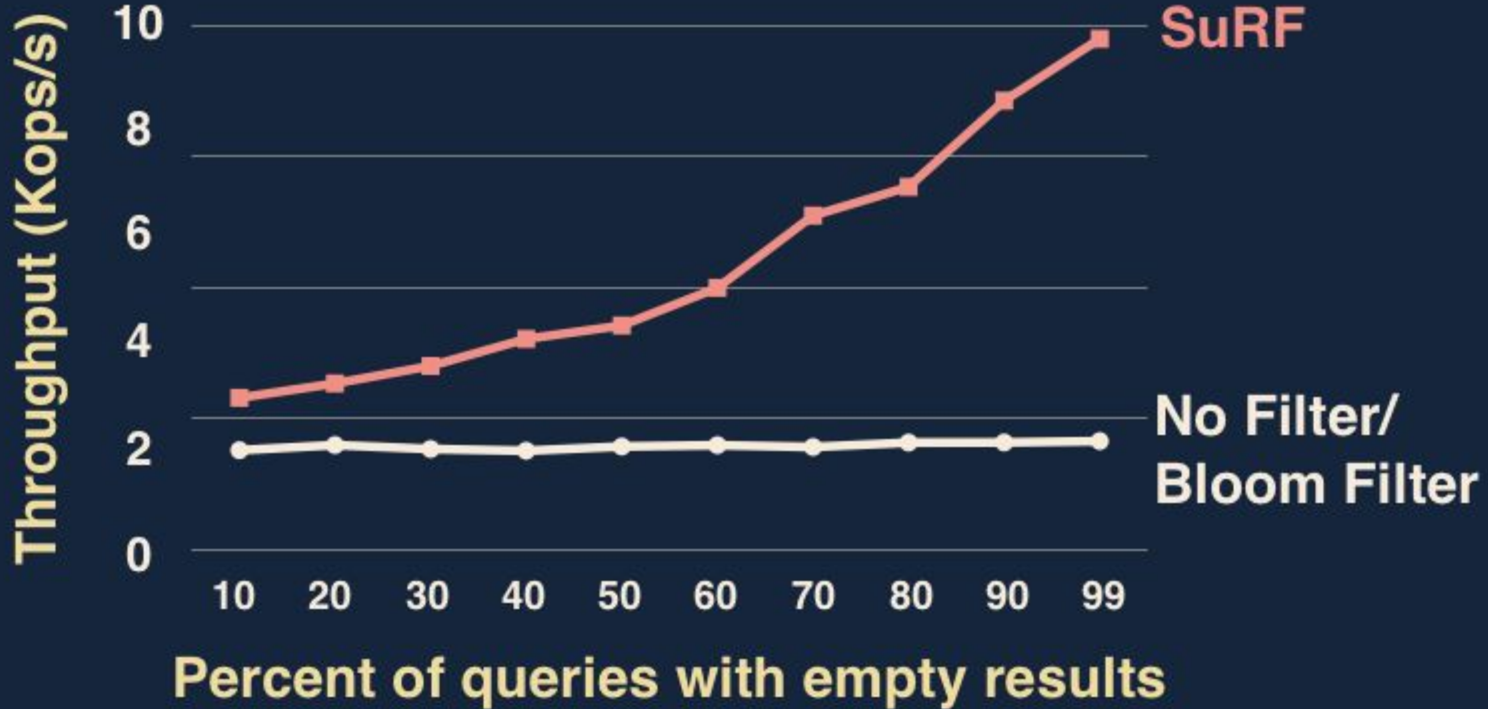
Leiserson et al. *There's Plenty Of Room At The Top*

MaxFlow over time



Leiserson et al.

SuRFs speed up range queries in RocksDB



2018--?: Putting Heterogeneity On The Programmer

The trend in architecture over the last decade:

increasingly shift pain to the programmer.

This will get worse. (No alternative yet)

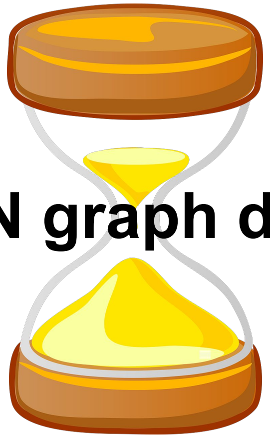
Applications

Narrow Waists

Heterogenous, experts-only Hardware

Waists are emerging: ML example

Applications



DNN graph definition

TensorFlow TensorRT Android NNAPI Apple CoreML

TPUs v1-3, EdgeTPU, Neural Compute Stick, A12 Bionic, Intel FPGA DLIA, GPUs, x86, ARM, ...

Today’s specialization

Machine Learning (‘nuff said)

Network cards

GPUs

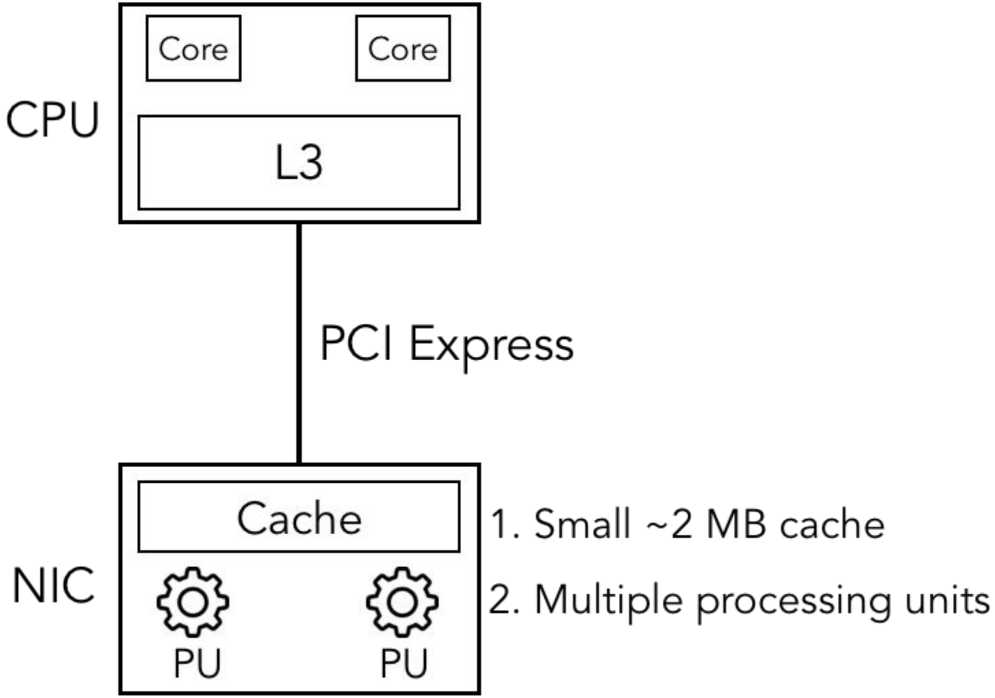
FPGAs

On-CPU functions (video codec, crypto, more)

NICs: Fabrics, Stack Bypass, and RDMA

Extensive - and nuanced - processing architecture on-NIC.

(Different from Smart NICs)

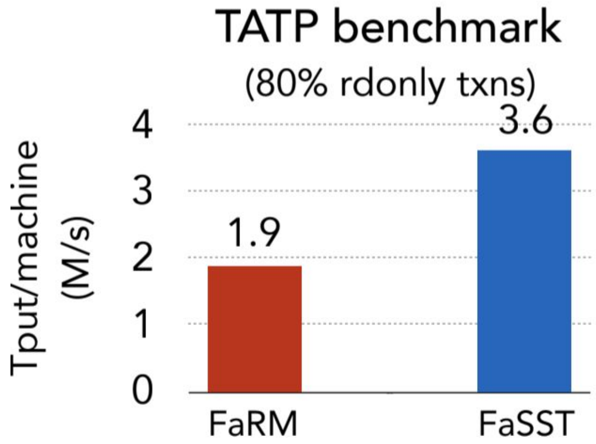


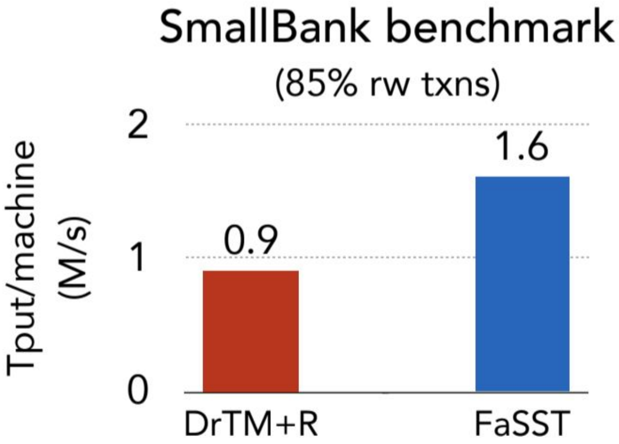
FaSST: Fabric-optimized Transactions with RPCs

	NICs	Cores
FaRM [SOSP 15]	2x ConnectX-3	16
DrTM+R [EuroSys 16]	1x ConnectX-3	10
FaSST	1x ConnectX-3	8

vs FaRM: FaSST uses 50% fewer h/w resources

vs DrTM+R: FaSST makes no data locality assumptions

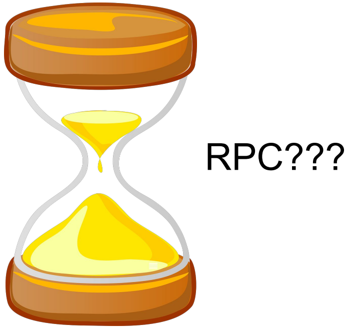




eRPC: Generalized RPC for fabrics

Replicated PUT latency:
Raft+eRPC vs others

Measurement	System	Median	99%
Measured at client	NetChain	9.7 µs	N/A
	eRPC	5.5 µs	6.3 µs
Measured at leader	ZabFPGA	3.0 µs	3.0 µs
	eRPC	3.1 µs	3.4 µs



What is the right, general abstraction for using datacenter networks?

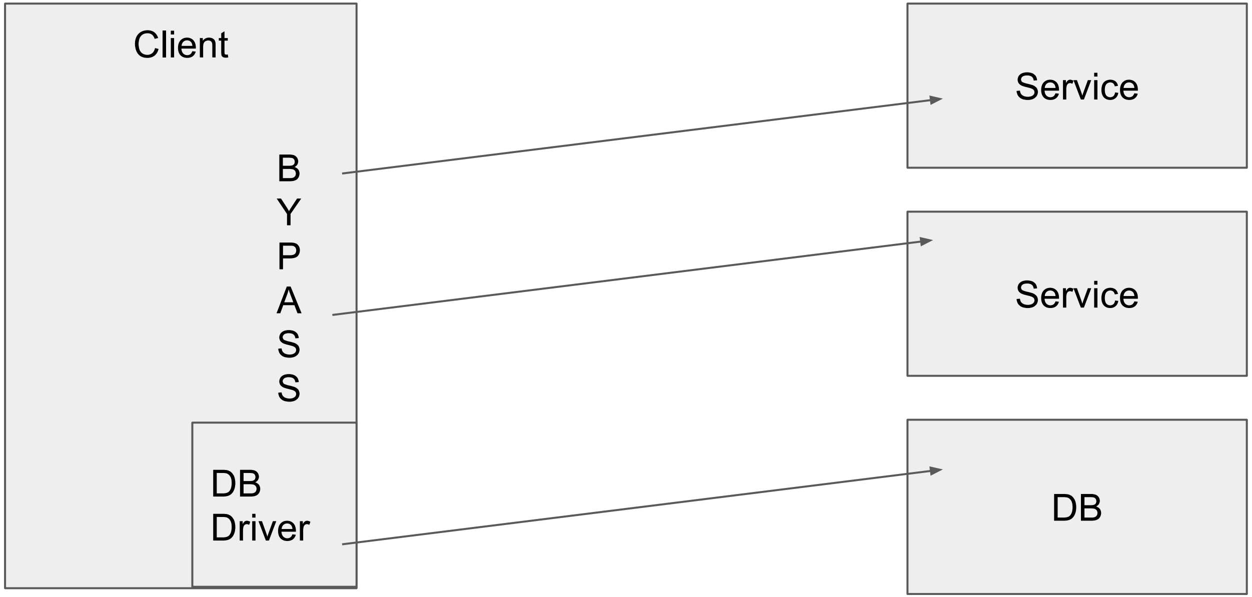
eRPC requires applications be designed with its needs in mind;
RAFT is low-level, easy to modify;
Real applications would likely take a lot of work.

Stack bypass matters more than RDMA

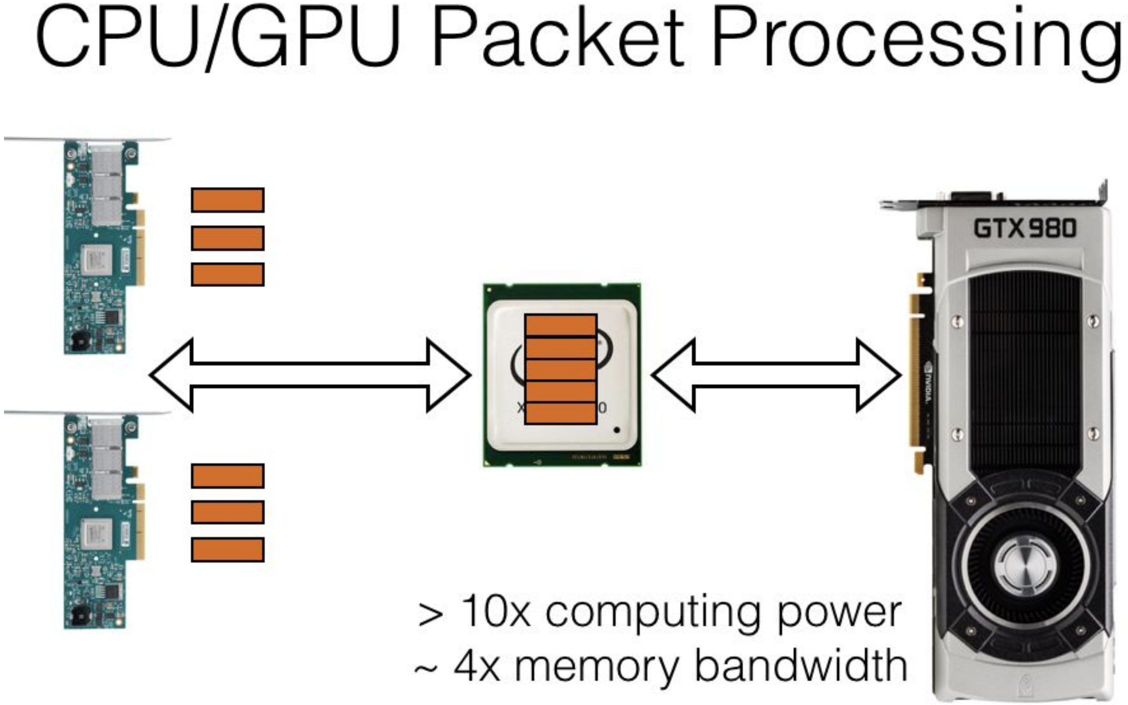
For network-intensive applications:

- OS network stack bypass: > 10x perf gains
- RDMA vs messaging: ~1x
- PCI bus transactions are key optimization goal

Stack bypass is great, but messes aren't



GPU Cautionary Tale (a little old)



Rethink GPU advantages

~~Higher computational power~~

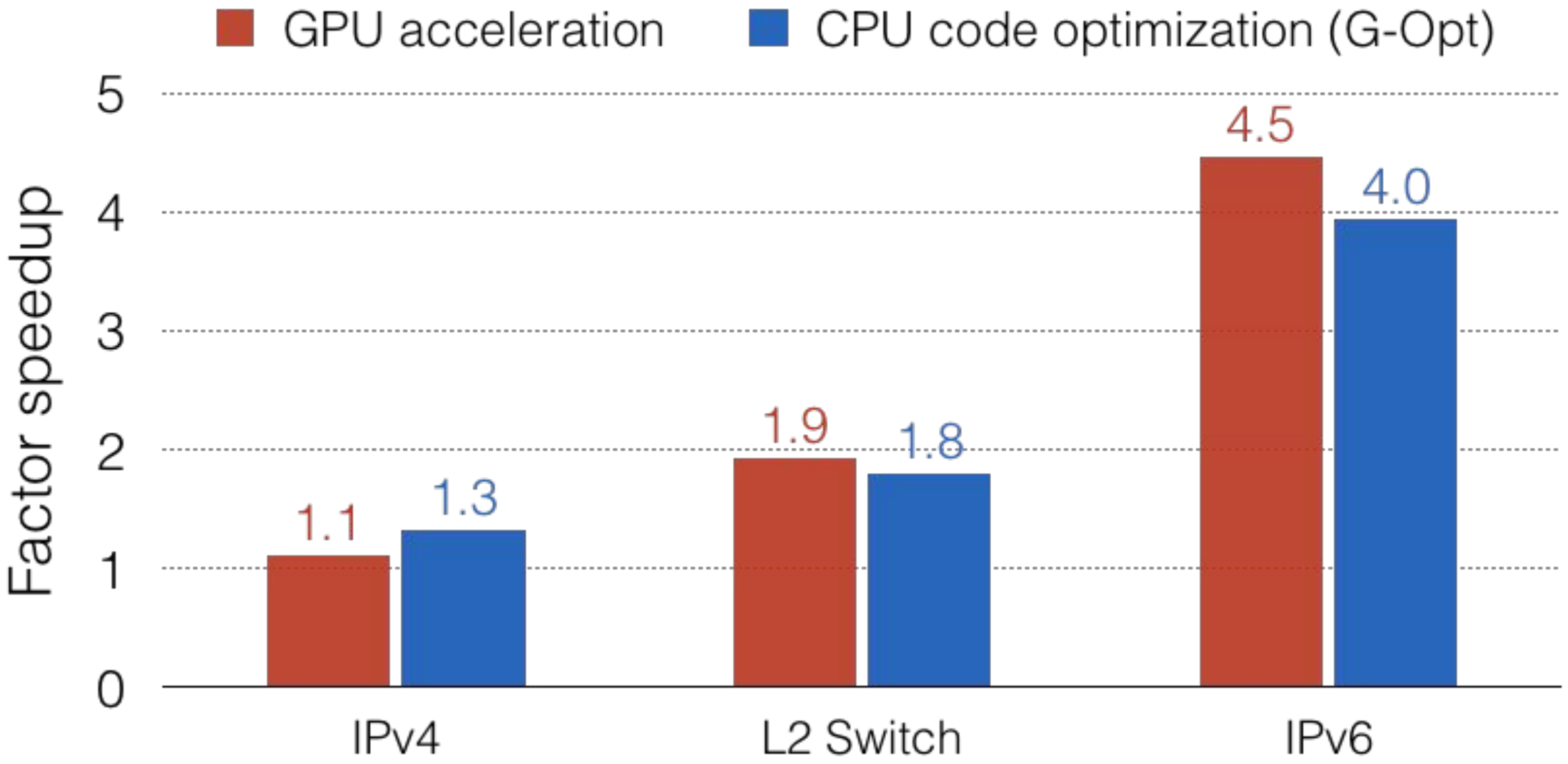
Packet forwarding usually not CPU intensive

~~Higher memory bandwidth~~

Most router applications not memory BW intensive

Memory latency hiding!

Doing it right on CPU...

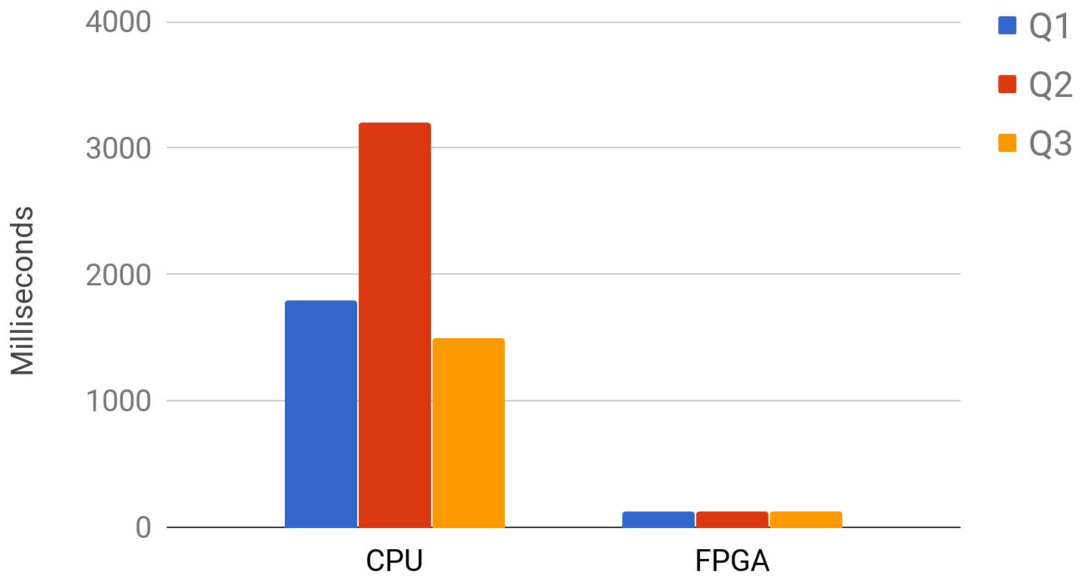


FPGA Cautionary Tale

A SIGMOD 2017 paper proposes using FPGAs for database pattern matching

```
select count(*) from test where regex(name, ‘Strasse|Str’);
```

End-to-end regex operator time for 10 Million records




```
= ‘(Strasse|Str\.) .* (8[0-9]{4})’
= ‘[0-9]+(USD|EUR|GPB)’
= ‘[A-Za-z]{3}\:[0-9]{4}’
```

So we optimized the CPU baseline...

1. Replace NFA with DFA
   a. Reduces matching complexity. DFA too big for FPGA, but fits in CPU cache.
   Used an off-the-shelf CPU accelerated library, Intel's HyperScan.
   (It's amazing and has some serious vector wizardry!)

2. Avoid dynamic memory allocation
   a. Reduces CPU cycles
   b. Reduces cache misses

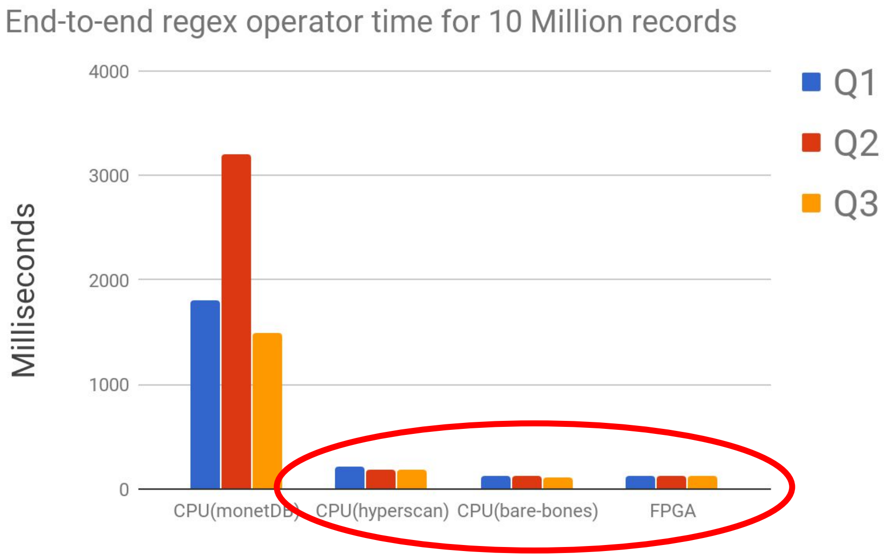
3. Process a batch of records instead of processing them one by one
   a. Reduce CPU pipeline stalls due to memory dependency

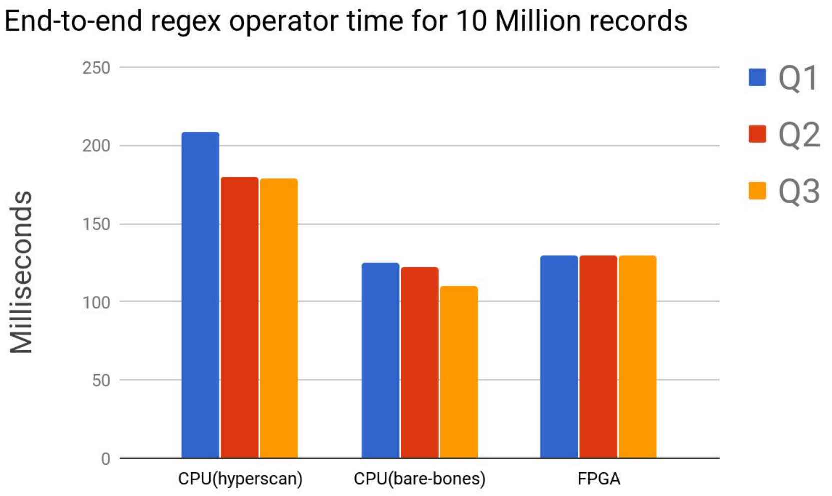
Xin Zhang(CMU)
Anuj Kalia (CMU)
Michael Kaminsky (Intel Labs),
David Andersen (CMU)

End-to-end performance

```
Q1:regex(name, ‘(Strasse|Str\.) .* (8[0-9]{4})’;
Q2:regex(name, ‘[0-9]+(USD|EUR|GPB)’);
Q3:regex(name ‘[A-Za-z]{3}\:[0-9]{4}’);
```

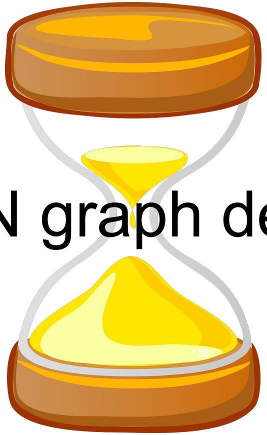
2. End-to-end regex operator time in ms (10 Million records)





Waists are **kind of** emerging: ML example

Applications



DNN graph definition

TensorFlow   TensorRT   Android NNAPI   Apple CoreML

How to create kernels?   How to specialize DNN to device?

TPUs v1-3, EdgeTPU, Neural Compute Stick, A12 Bionic, Intel FPGA DLIA, GPUs, x86, ARM, ...

Wither DBs?

What role do databases play in mediating the messy, heterogenous future?

- We're already seeing a lot of GPU-accelerated DBs

✓ **A locus for concentrated optimization, where many apps offload most work to the DB system**

- **Not the only locus, and we're in an "APIs flowering" phase**

Moving up a level, in a diverse and data-driven world, we must manage diverse programming abstractions against very large data sets. Rather than expecting to develop “the” data analysis language for Big Data, perhaps by extending SQL or another popular language, we must let users analyze their data in the medium they find most natural

Beckman Report, 2013

Standard TensorFlow format

Another approach is to convert whatever data you have into a supported format. This approach makes it easier to mix and match data sets and network architectures. The recommended format for TensorFlow is a TFRecords file containing

DataLoader and Postgres (or other SQL) an option?

[Pytorch]

J
JQVeenstra 13d

Hi,

I'm trying to keep things in a postgres database, because - well, it's complicated.

Is there anyone who's done this in an efficient manner with the DataLoader and Dataset classes? I'm relatively proficient at Google-Fu, and no dice so far.

SQL → MapReduce | BigTable → Flume → Spanner (SQL)

As in ML, so in DBs

- SQL isn't enough
  - UDFs provide a pathway, but how to specialize UDFs for device? Standardize on UDFs?
  - But will also be a mismatch at a high level
  - Same language + compiler research underway for ML will be needed here…
- Need to aim for reasonable # of APIs to support diverse applications
  "Strive to create a world where it is easy to write fast code" [Leiserson]
  - Balancing expressiveness and constraints is terribly hard

- How do databases play nicely with other emerging waists (ML, network, video, and the ones we haven't thought of yet)?

DB community / academia staying relevant

The default path: **Big industry will dominate**

Why?

- Vertically integrated, know needs well, can target cost reduction and perf improvements where they need. Large enough to fab.

But:

- FAAAAM [fb, aapl, amzn, goog, baba, msft] innovations will trickle,
  But their **priority order sometimes differs**
  [scale, vert. integrated, expert programmers]
  [Recent Abadi blog post about Spanner]

Heterogenous hardware

Need for across-the-board improvements

- In algos, languages, implementations

Incredible opportunity to create the next bridge APIs and systems.