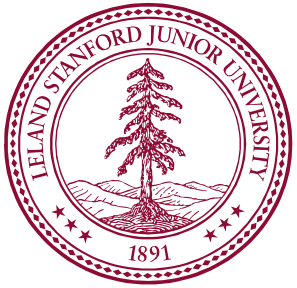
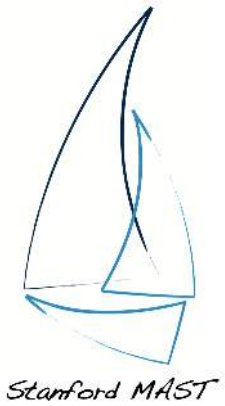
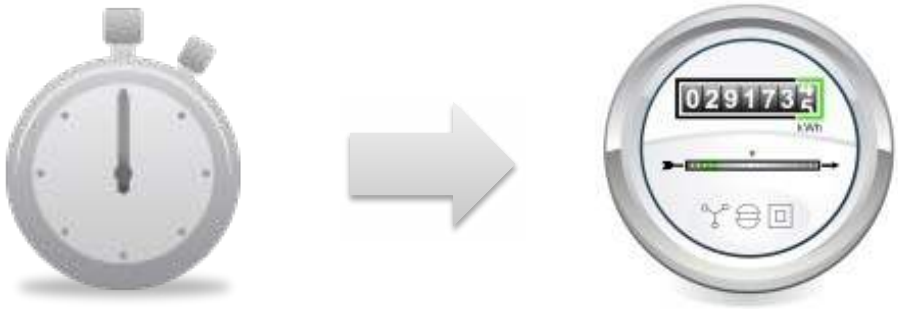
The PicoSecond is Dead
Long Live the PicoJoule

Christos Kozyrakis

Stanford University
http://mast.stanford.edu







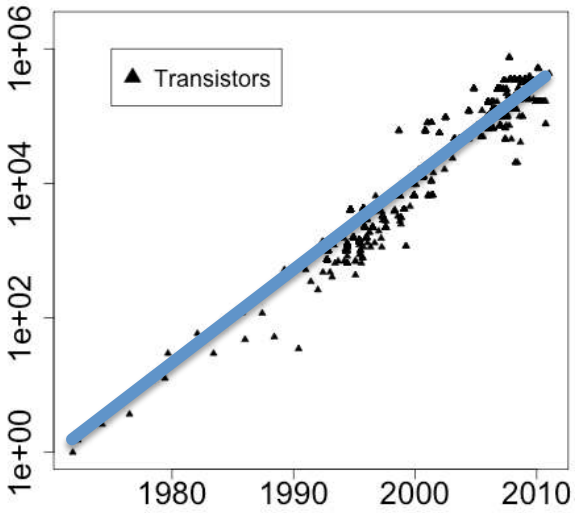
Scaling Systems without Technology Help

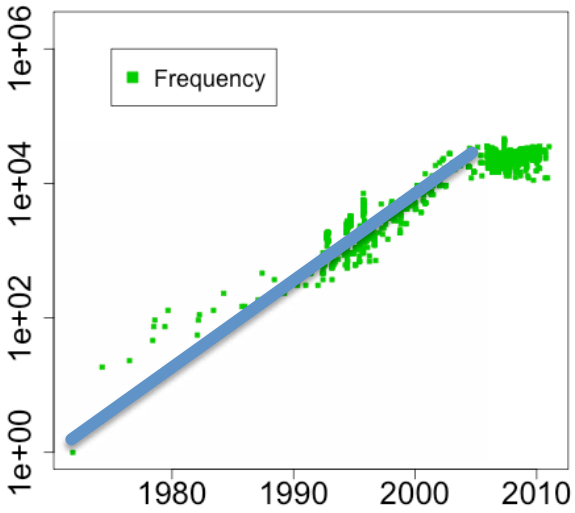
Systems Scaling

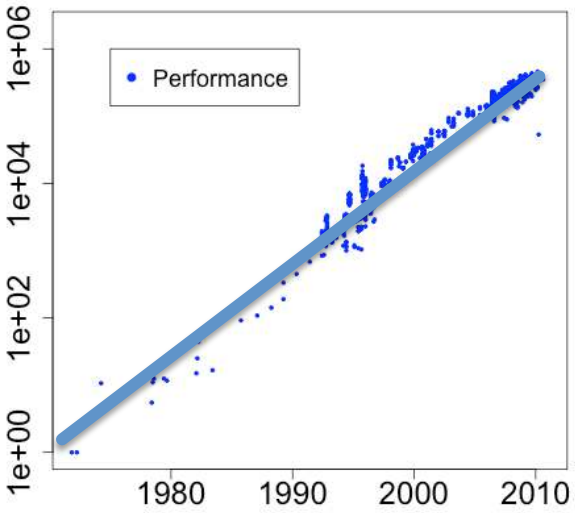
- Higher performance
- More data
- Constant cost (power, $)



Scaling Past







[Source: CPUDB]

Transistors ↑ + frequency ↑ = performance ↑

Scaling Past

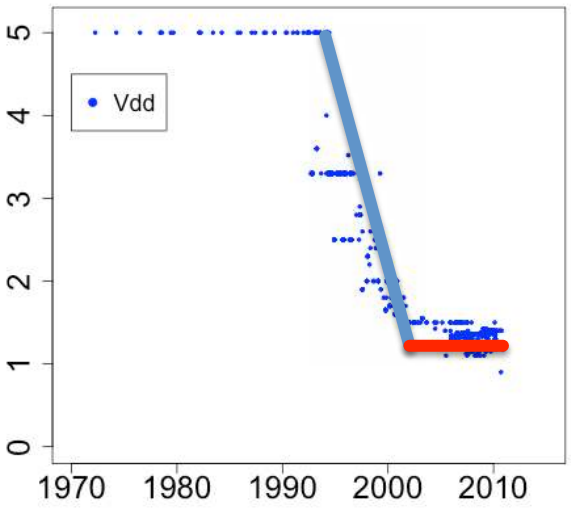
Power ≈ C•Vdd^2•Freq

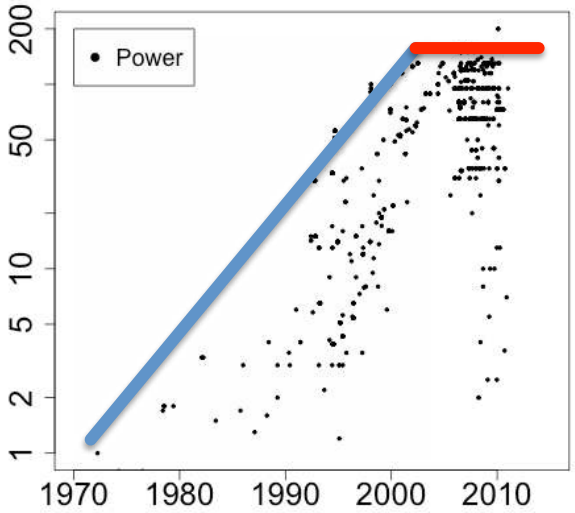
Transistors + Frequency ↑

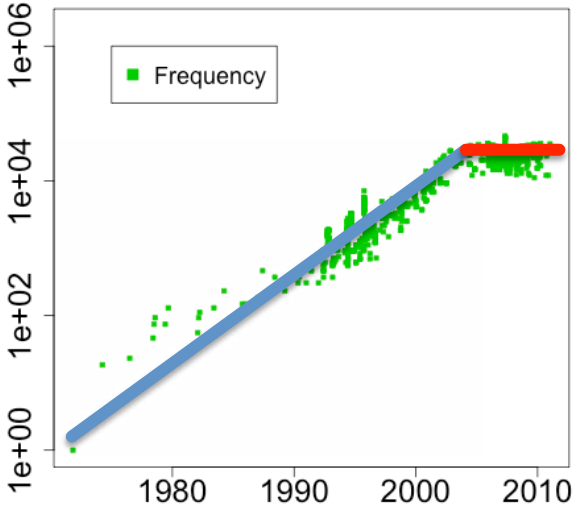
Capacitance + Voltage ↓

Power density →

Scaling Present







[Source: CPUDB]

Voltage scaling ✖

Transistors ↑ Power density ↑

Frequency ➔

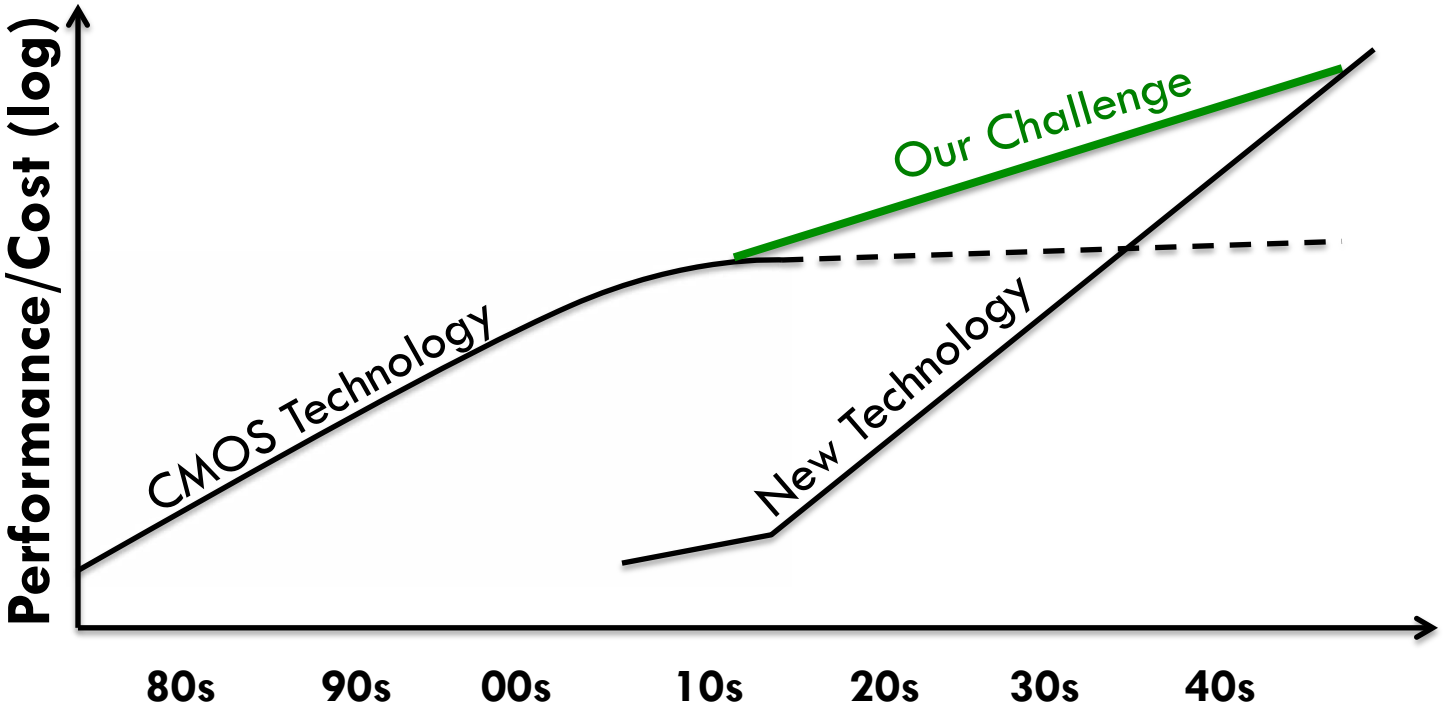
Scaling Present

$$\text{Power} = C \bullet Vdd^2 \bullet Freq$$

All chips are now power limited

Switch to multi-core does not change this

Scaling without Technology Help



[Source: Hill & Kozyrakis'12]

Scaling without Technology Help

Power = Ops/sec • Energy/op

Reduce energy/op through better HW

Reduce ops/sec needed through better SW

Tradeoff performance – power

Better HW with Specialization

Assume temporal locality for now

Caches work

Energy Overheads

Operation	Energy	Scale
8-bit add	0.03 pJ	1
32-bit add	0.10 pJ	3
8-bit mult	0.20 pJ	6
32-bit mult	3.00 pJ	100
16-bit FP mult	1.00 pJ	30
32-bit FP mult	4.00 pJ	133



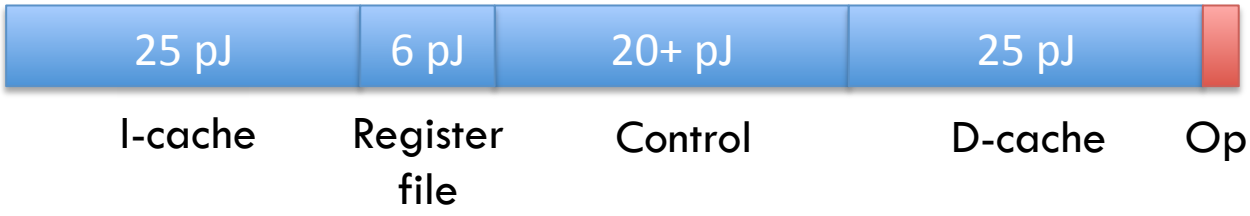
Obvious thought: specialize data-types

But Wait…

Operation	Energy	Scale
8-bit add	0.03 pJ	1
32-bit add	0.10 pJ	3
8-bit mult	0.20 pJ	6
32-bit mult	3.00 pJ	100
16-bit FP mult	1.00 pJ	30
32-bit FP mult	4.00 pJ	133
RISC instruction	**>70.00 pJ**	**>2,300**

Instruction overheads dominate!

Energy Overheads



First

- Amortize instruction and control overheads
- Avoid accesses to register files and data caches

Then

- Specialize data-types

SIMD & Vectors



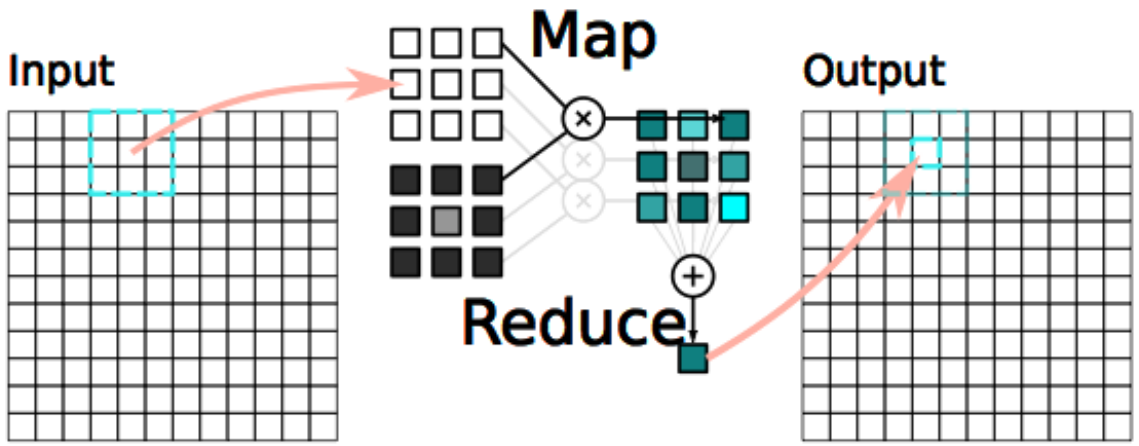
SIMD amortizes I-cache and control overheads

10x energy efficiency → must use

Not enough on its own

Must also amortize register and D-cache accesses

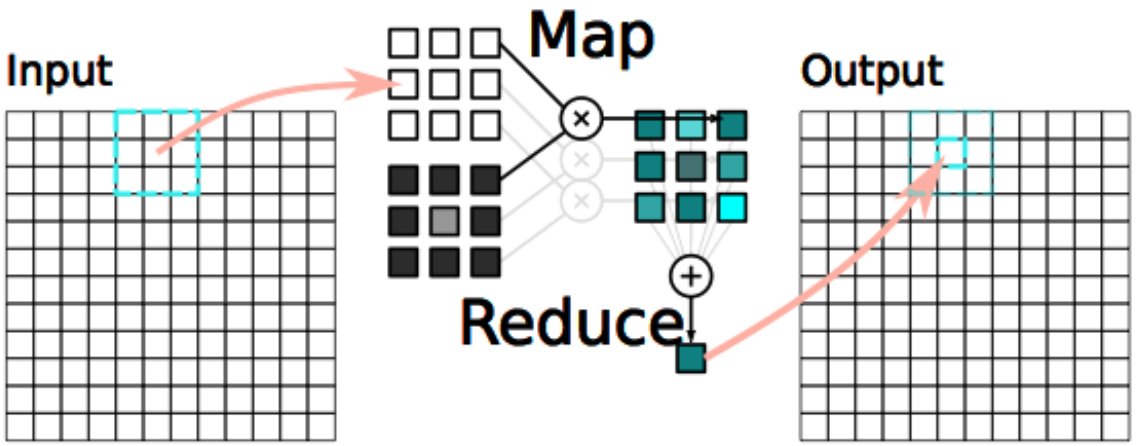
Specialization for Convolutions



Convolutional patterns

- Signal/photo/video processing, computer vision, …
- Map-reduce patterns over 1D/2D/3D stencils

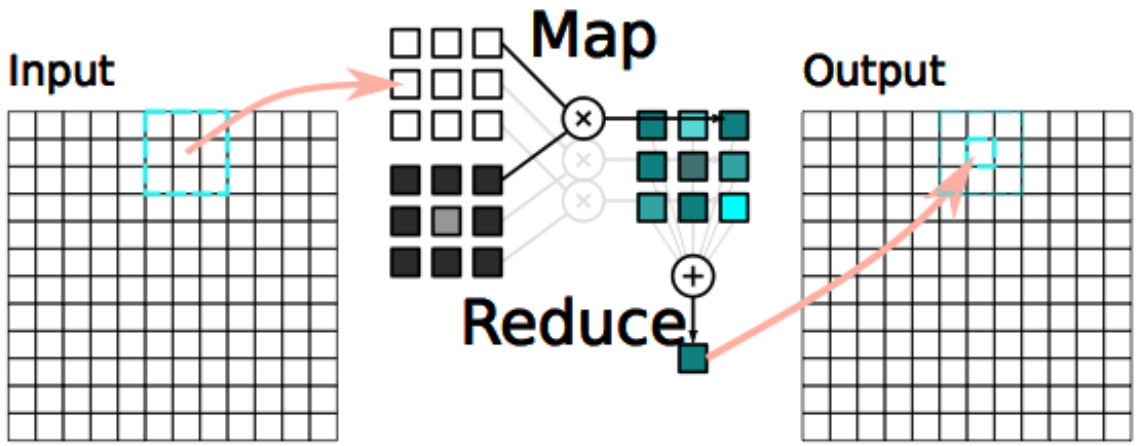
Specialization for Convolutions



Convolutional patterns

- Signal & photography processing, computer vision, …
- Map-reduce patterns over 1D/2D/3D stencils

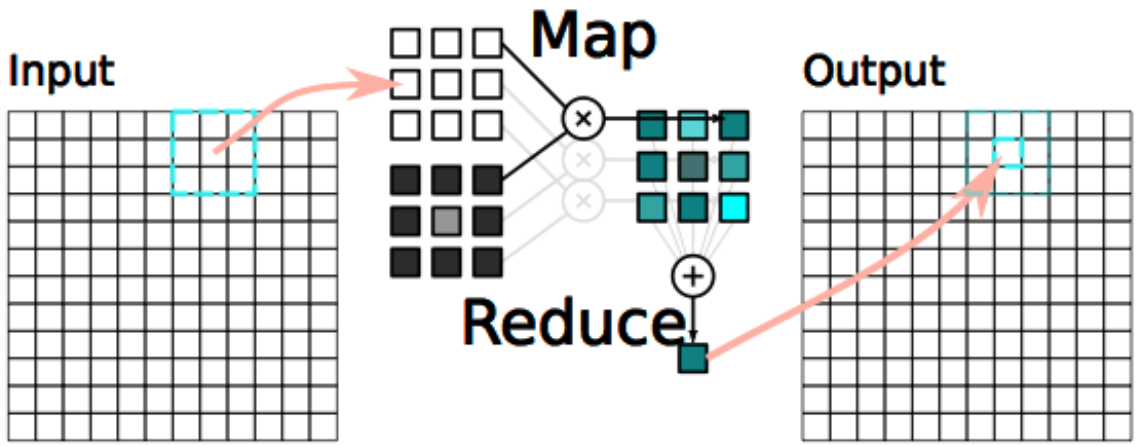
Specialization for Convolutions



Convolutional patterns

- Signal & photography processing, computer vision, …
- Map-reduce patterns over 1D/2D/3D stencils

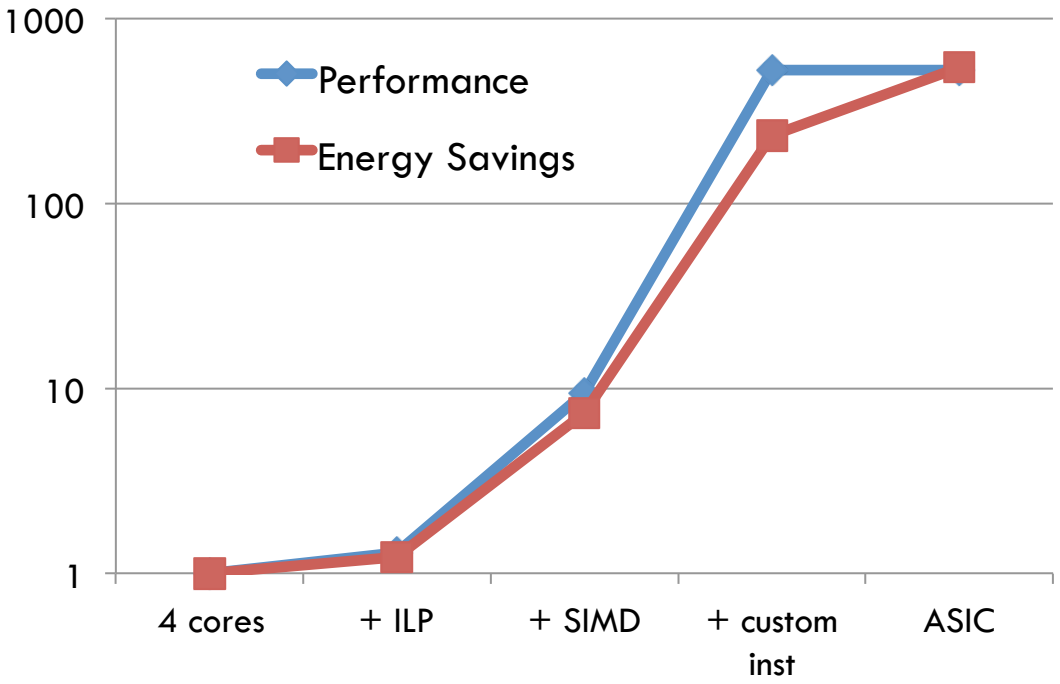
Specialization for Convolutions



Specialized unit

- Register files that shift support → no data reloading
- Map-reduce, SIMD instructions → amortize control, shared data
- Specialized data-types → avoid waste

Specialization Opportunity

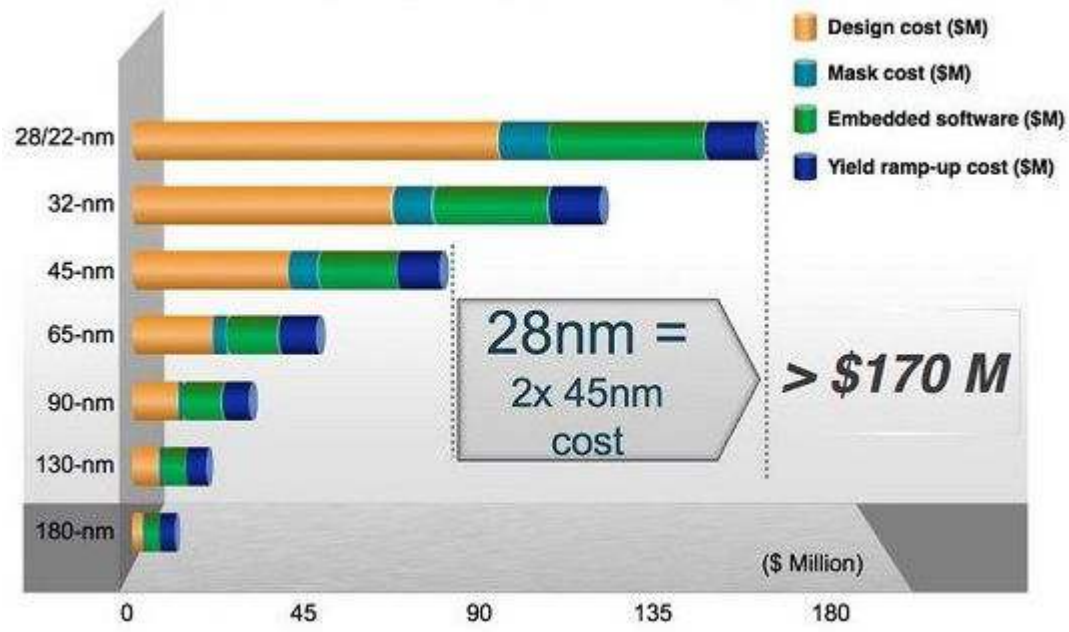


[Source: ISCA'10]

>100x opportunity in performance & energy

Close to ASIC efficiency

Specialization Challenge #1



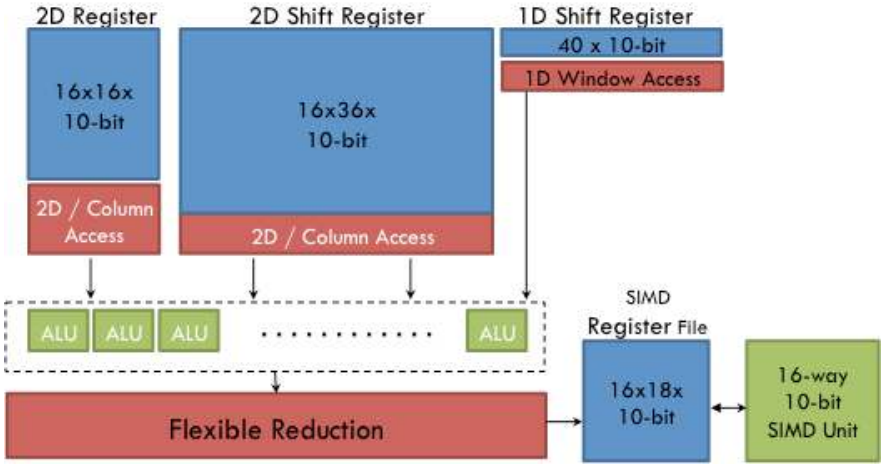
[Source: Xilinx]

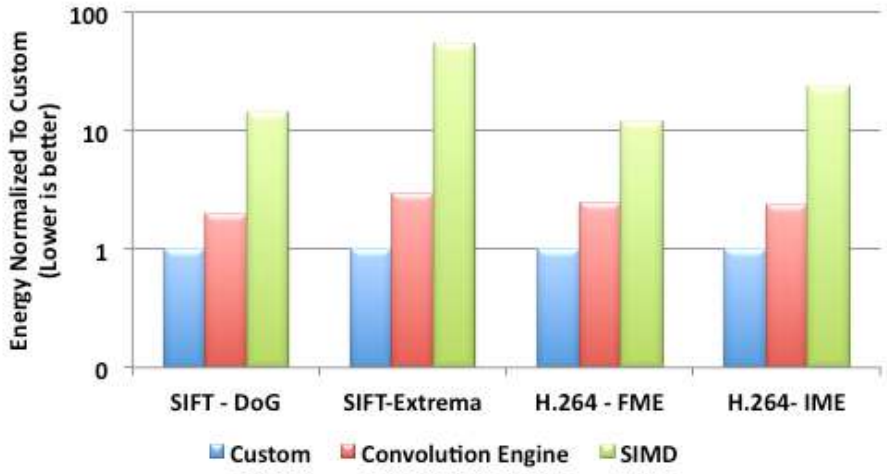
Efficiency Vs. generality

Solution #1: domain-specific accelerators

Solution #2: automatic generation of accelerators

Domain-specific Accelerators



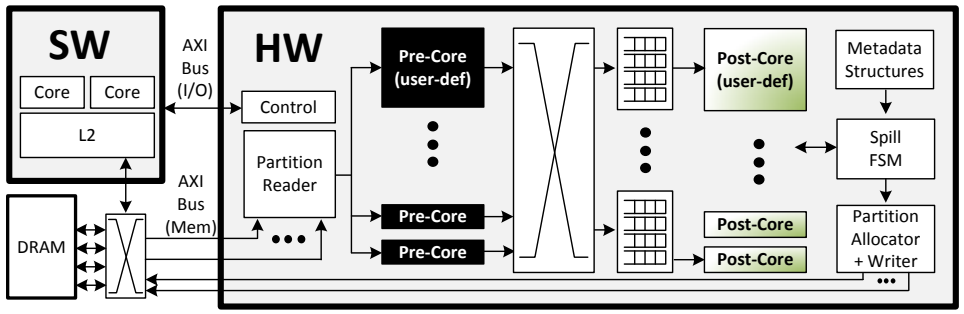


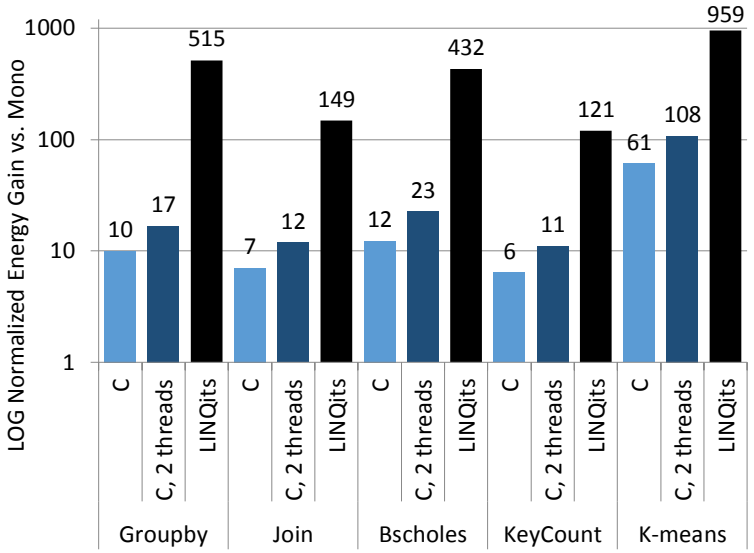
[Source: ISCA'13]

Convolution engine

- Generalized engine for convolution-line computations
- Wider storage, general ALUs, general reduction, …
- 10x better than SIMD, ~2x worse custom unit

Automatic Generation of Accelerators





[Source: Chung'13]

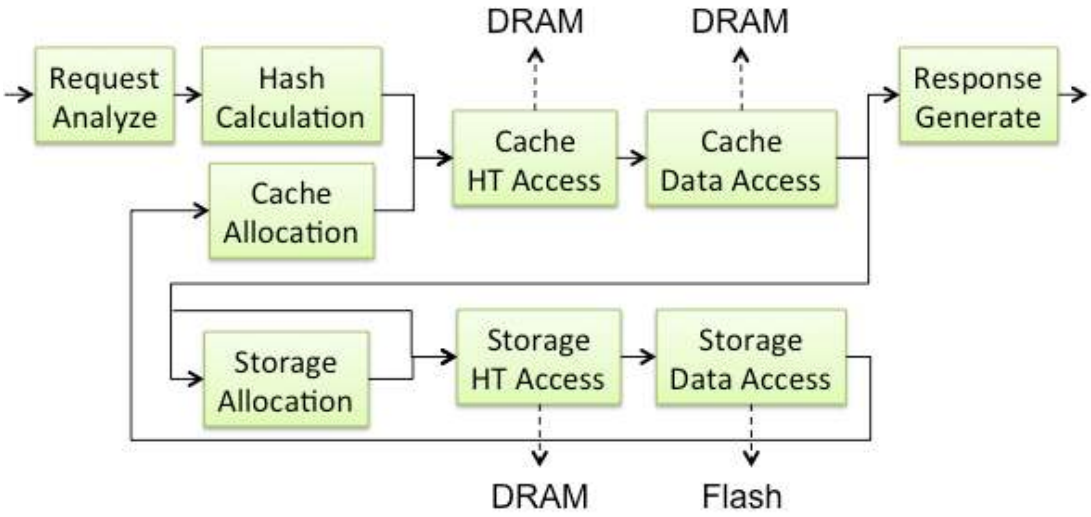
LINQits engine

- Configurable engine for LINQ map-reduce
- Ops: select, selectMany, where, join, groupBy, aggregate
- 10x performance & 10x energy efficiency

Specialization for I/O

40Gbps NIC





[Source:NVMW'14]

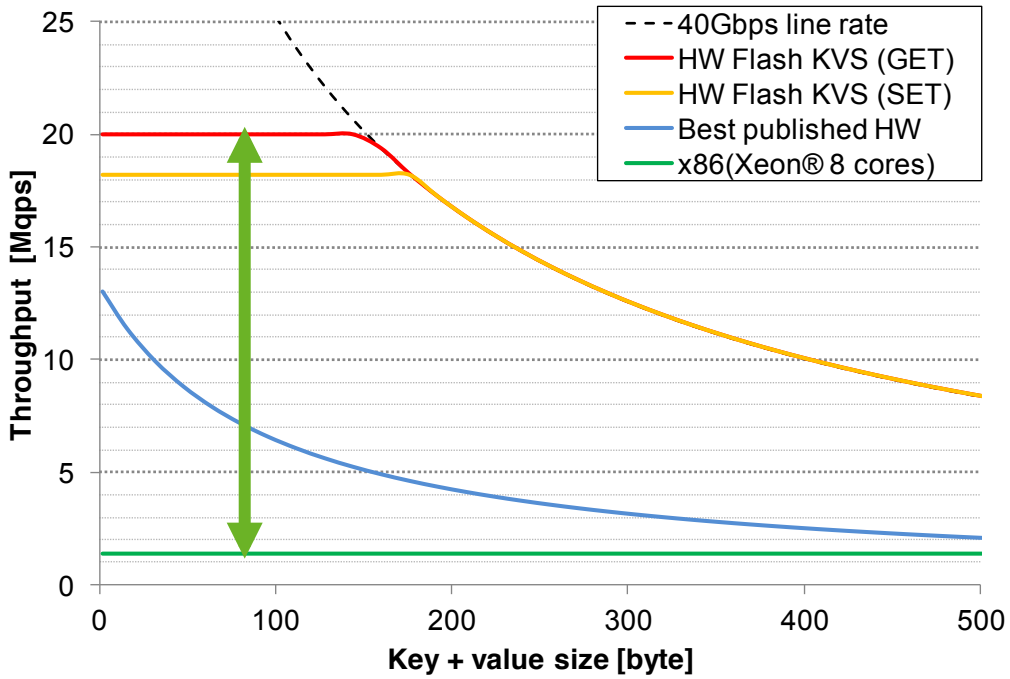
Data-serving accelerators (KVS, noSQL, …)

Common case in HW, complex cases in SW

Example: 20M IOPS @ 4–50usec (DRAM + SSD)

10x throughput & 10x energy efficiency of x86 servers

Domain-specific Specialization



[Source:NVMW'14]

Accelerator (DRAM): 20M IOPS, 3.8usec, 170 Kqps/W

Flash: 10M IOPS, 50usec, 85 KQPS/W

x86 Server: 2M IOPS, 300usec, 14 Kqps/W

Specialization Challenge #2

ASICs

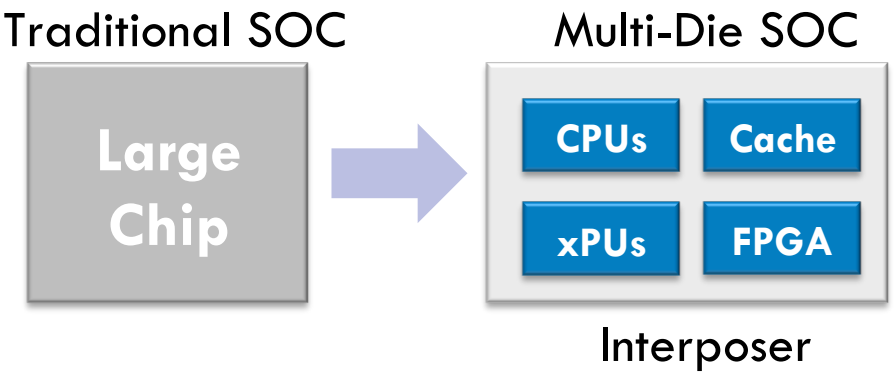
- Expensive and inflexible

FPGAs

- High overheads (bit-level config, I/O interface)
- Upto10x efficiency loss

Solutions

- Coarse-grain FPGAs
- 2.5D and 3D integration
- CPU + FPGA integration



Specialization Challenge #3

What if there is limited temporal locality?

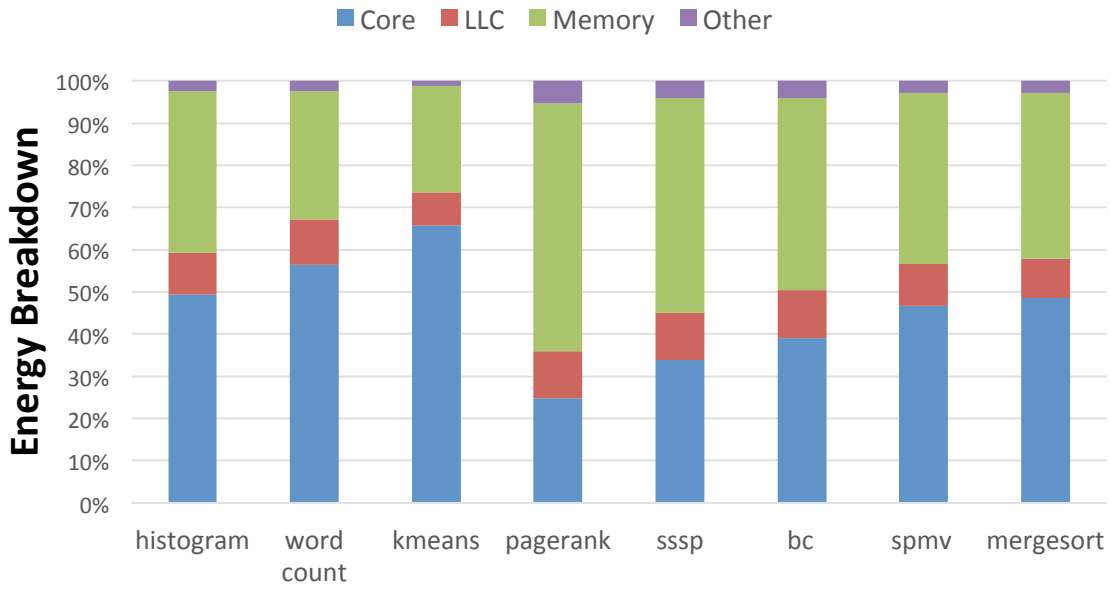
Graphs, in-memory analytics, …

Energy Overheads

Operation	Energy	Scale
8-bit add	0.03 pJ	1
32-bit add	0.10 pJ	3
32-bit FP mult	4.00 pJ	133
RISC instruction	**70.00 pJ**	**2,300**

Memory overheads dominate!

Full-system Energy Analysis

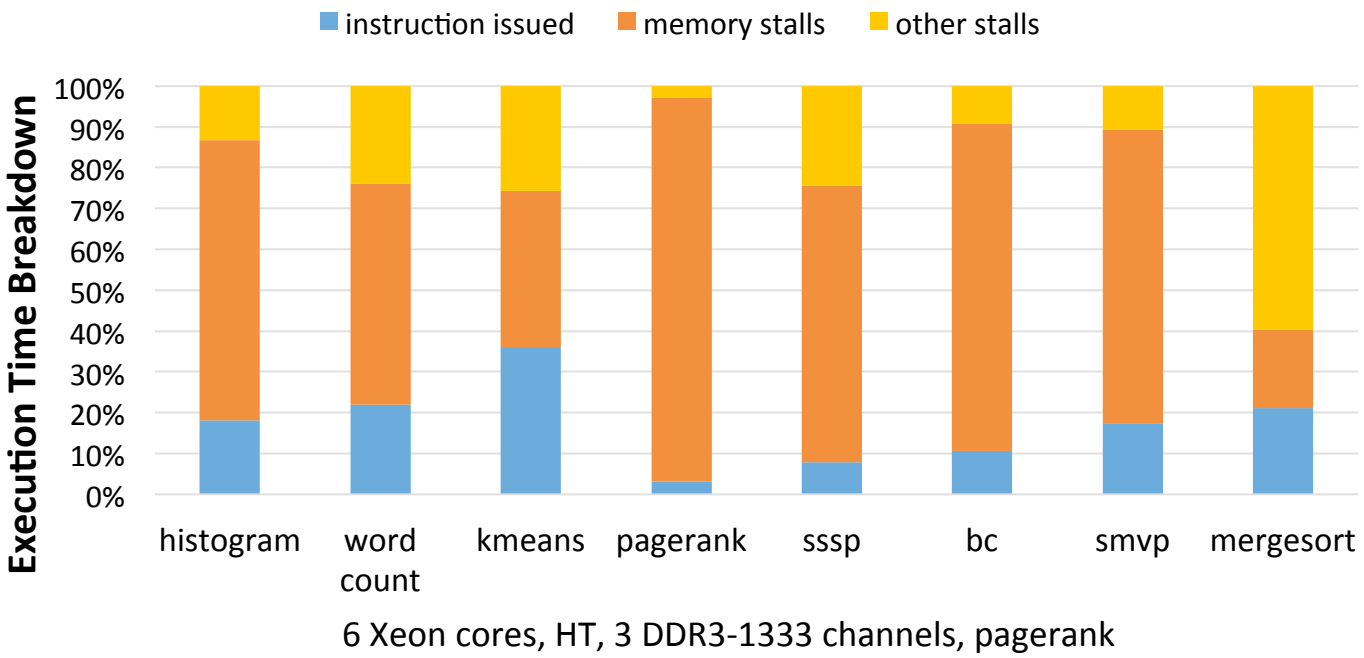


6 Xeon cores, HT, 3 DDR3-1333 channels, pagerank

~50% of energy due to memory

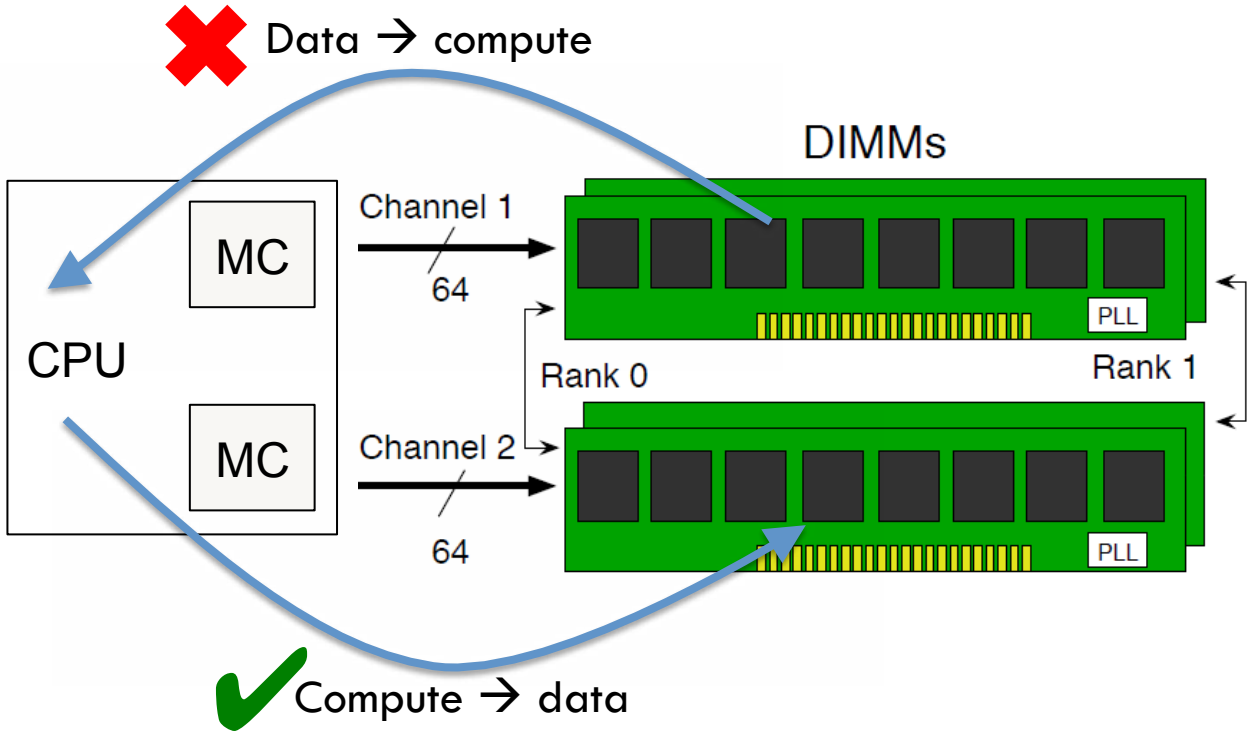
>50% of CPU energy due to idling

Full-system Runtime Analysis



>50% of the time waiting for memory

Memory-side Computing



Avoid interconnect overheads

Move computation function to the data

Streaming computations

Memory-side Computing



10x performance & 10x energy improvements

Compute capability scales with memory capacity

Didn't We Try this Before?

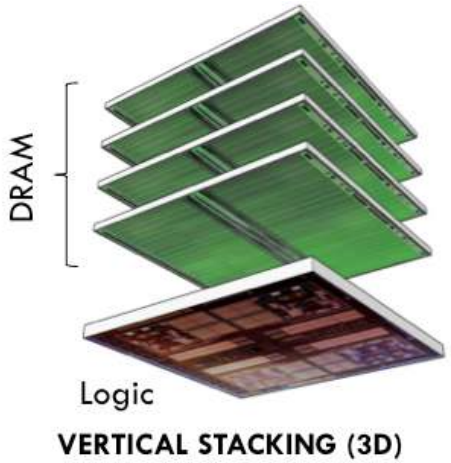
Exacube, PIM, IRAM, DIVA, FlexRAM, Active Pages, …

Reasons for past failures

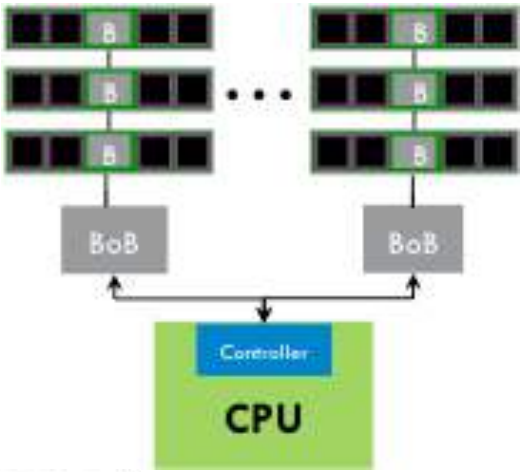
- ~~Conventional designs scale well~~
- ~~Apps have temporal locality~~
- ~~No good parallel programming models~~
- ~~All compute close to memory~~
- Poor technology options ?

Technology Options

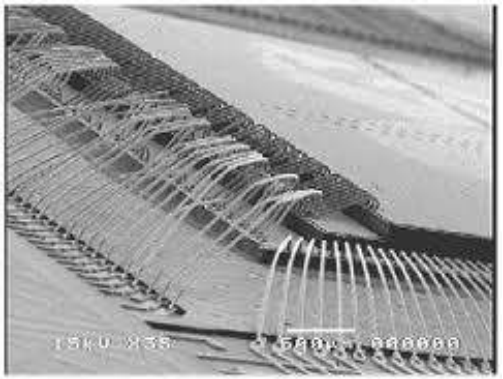
3D integration



Buffer-on-board



Stacking w/ edge-bonding



Better Software

Better Software

Better algorithms

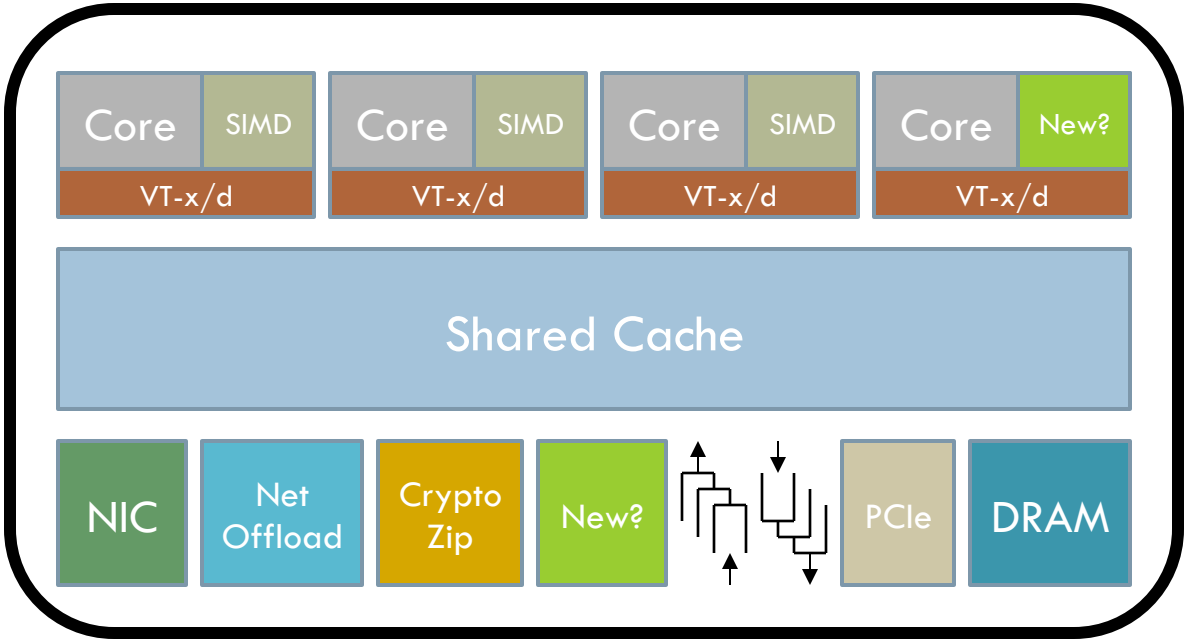
Match SW to modern HW

Reduce bloat

Specialization

Elasticity

Matching SW to Modern HW



Time for SW to exploit modern HW

Example: System Software

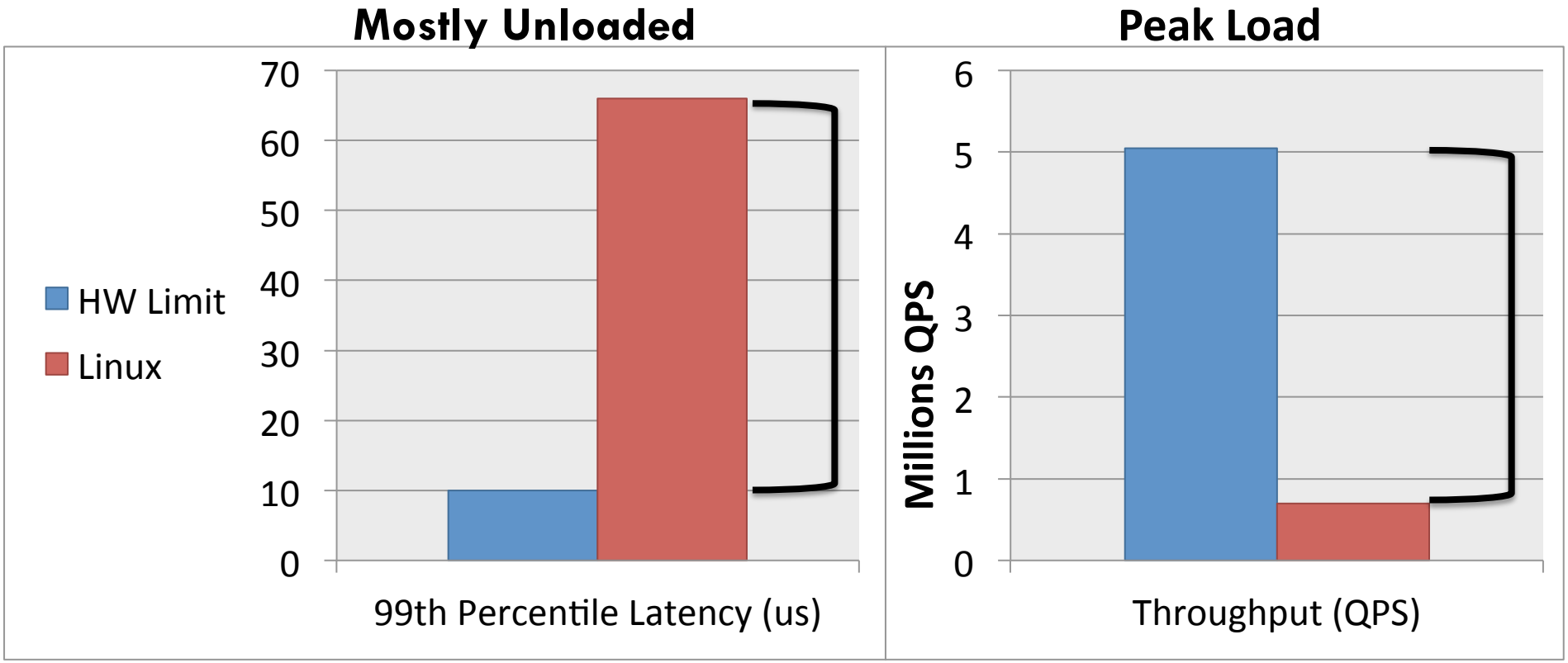
Hardware is fast

- 10 GbE widely deployed
- >10 cores per server

Large distributed apps should be fast

- Millions of QPS for small messages
- 10-20 us RTT

But Software is the Bottleneck



Example: Linux + memcached w/ 200 bytes values

Single-socket SandyBridge + 10GbE NIC

Conventional Wisdom

Bypass the kernel

- Move TCP to user-space
- Avoid protection domain crossings

Replace TCP

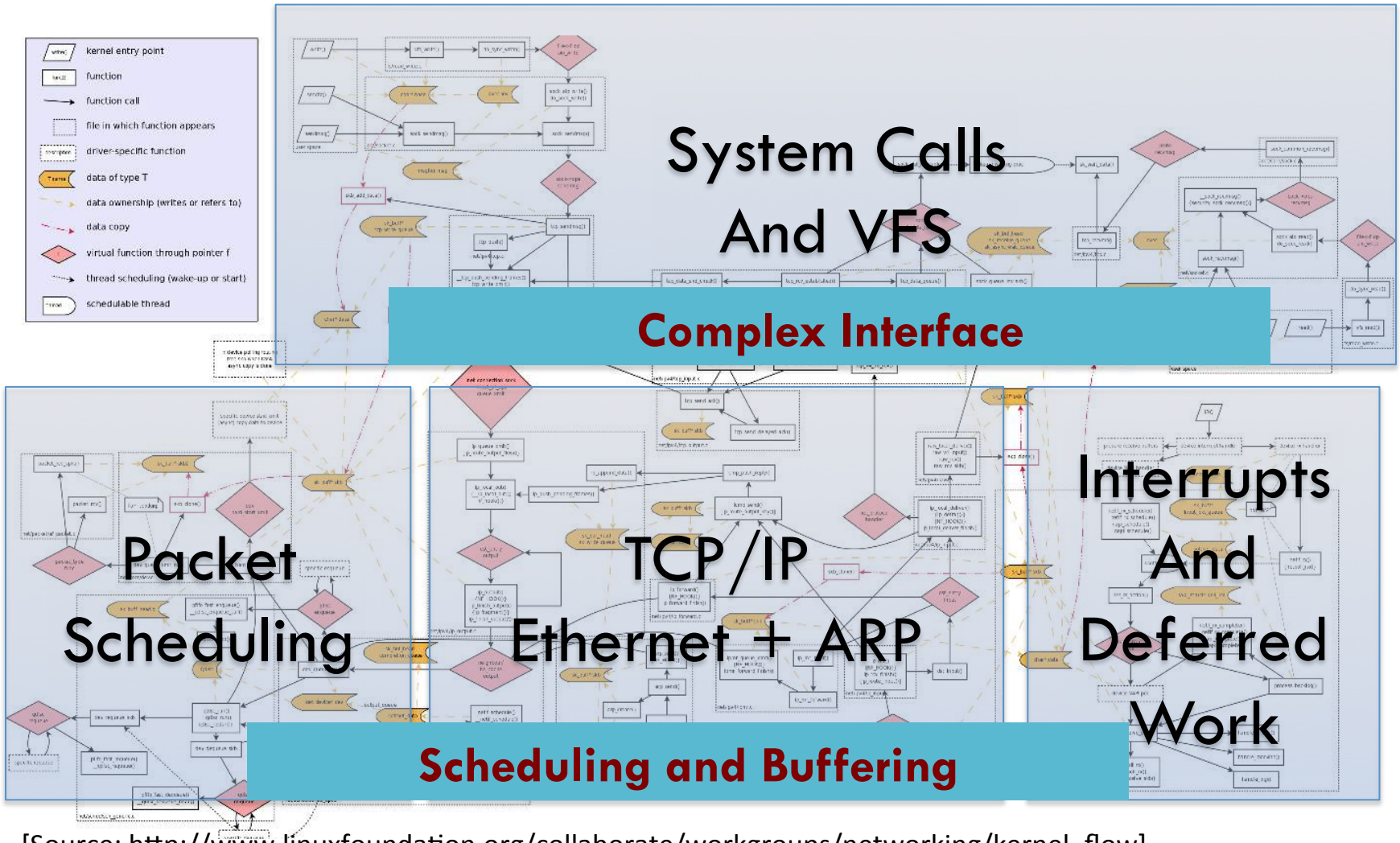
- Offload to hardware (TOE)
- Use a different transport protocol (UDP, new)

Replace Ethernet hardware

- Use a different fabric (Infiniband)
- Offload I/O processing to HW (rDMA)

How about looking into system SW?

Network I/O in Linux

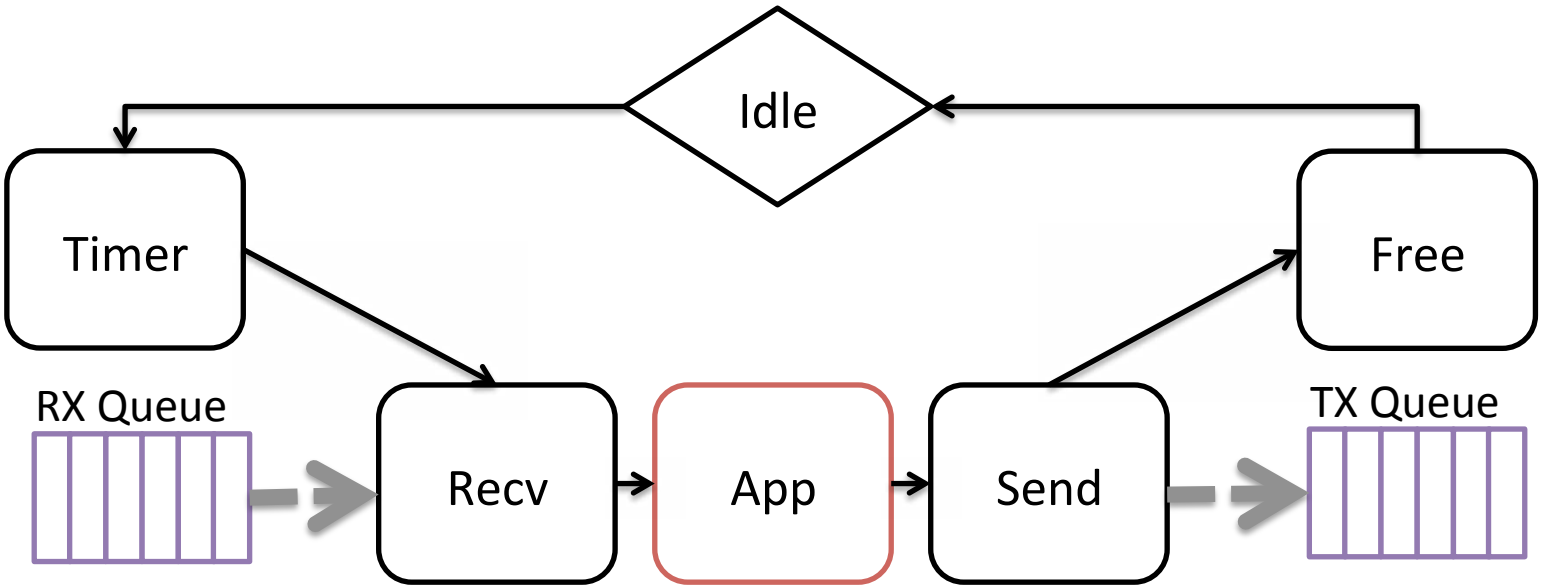


[Source: http://www.linuxfoundation.org/collaborate/workgroups/networking/kernel_flow]

Rethinking System Software

Efficiency mechanisms

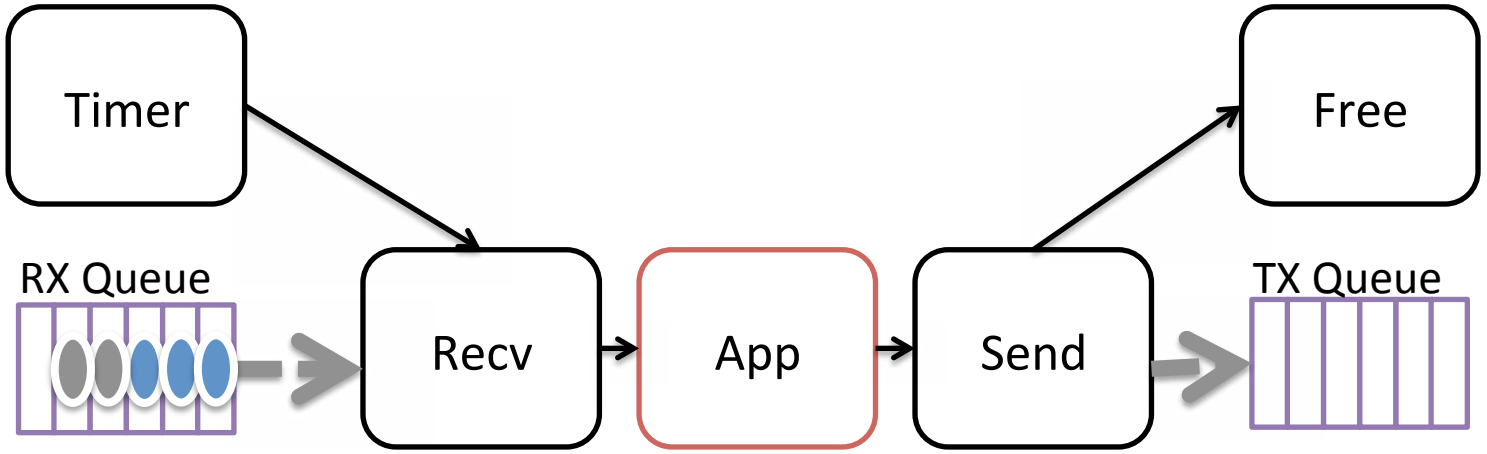
- Run to completion



Rethinking System Software

Efficiency mechanisms

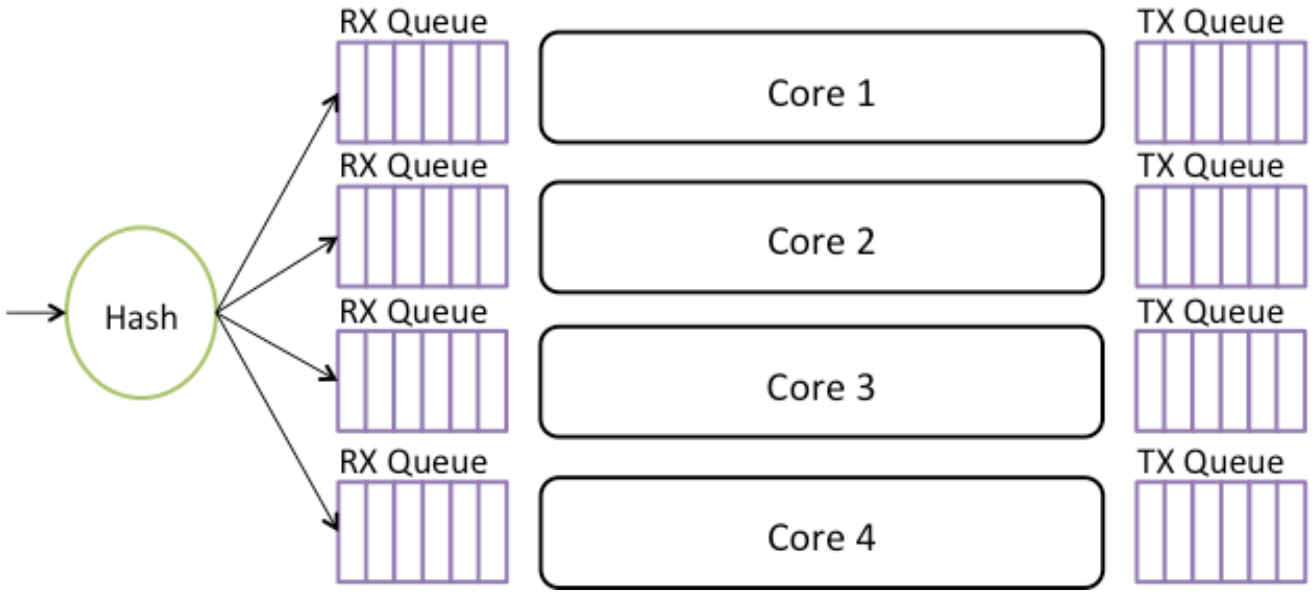
- Run to completion
- Adaptive batching



Rethinking System Software

Efficiency mechanisms

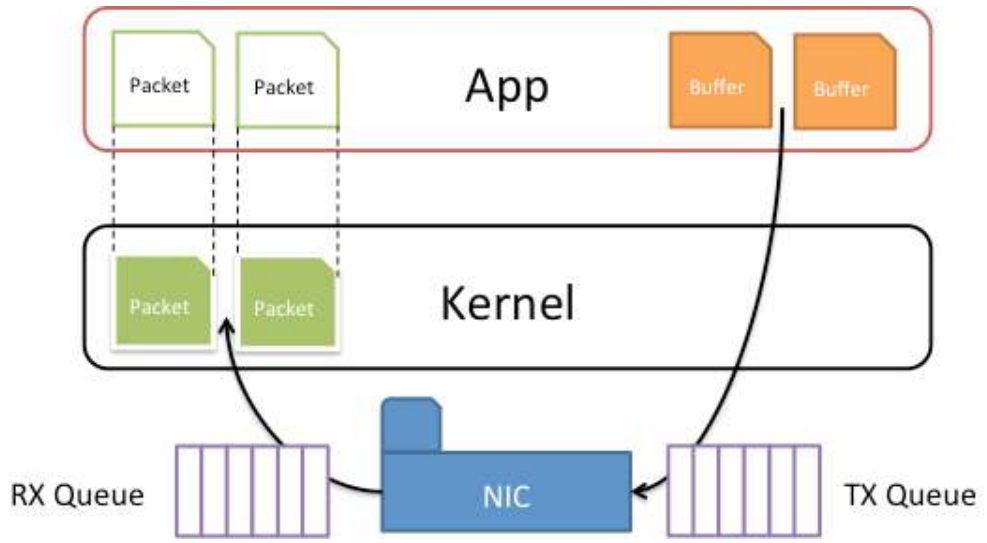
- Run to completion
- Adaptive batching
- Flow-consistent hashing



Rethinking System Software

Efficiency mechanisms

- Run to completion
- Adaptive batching
- Flow-consistent hashing
- Zero copy



Rethinking System Software

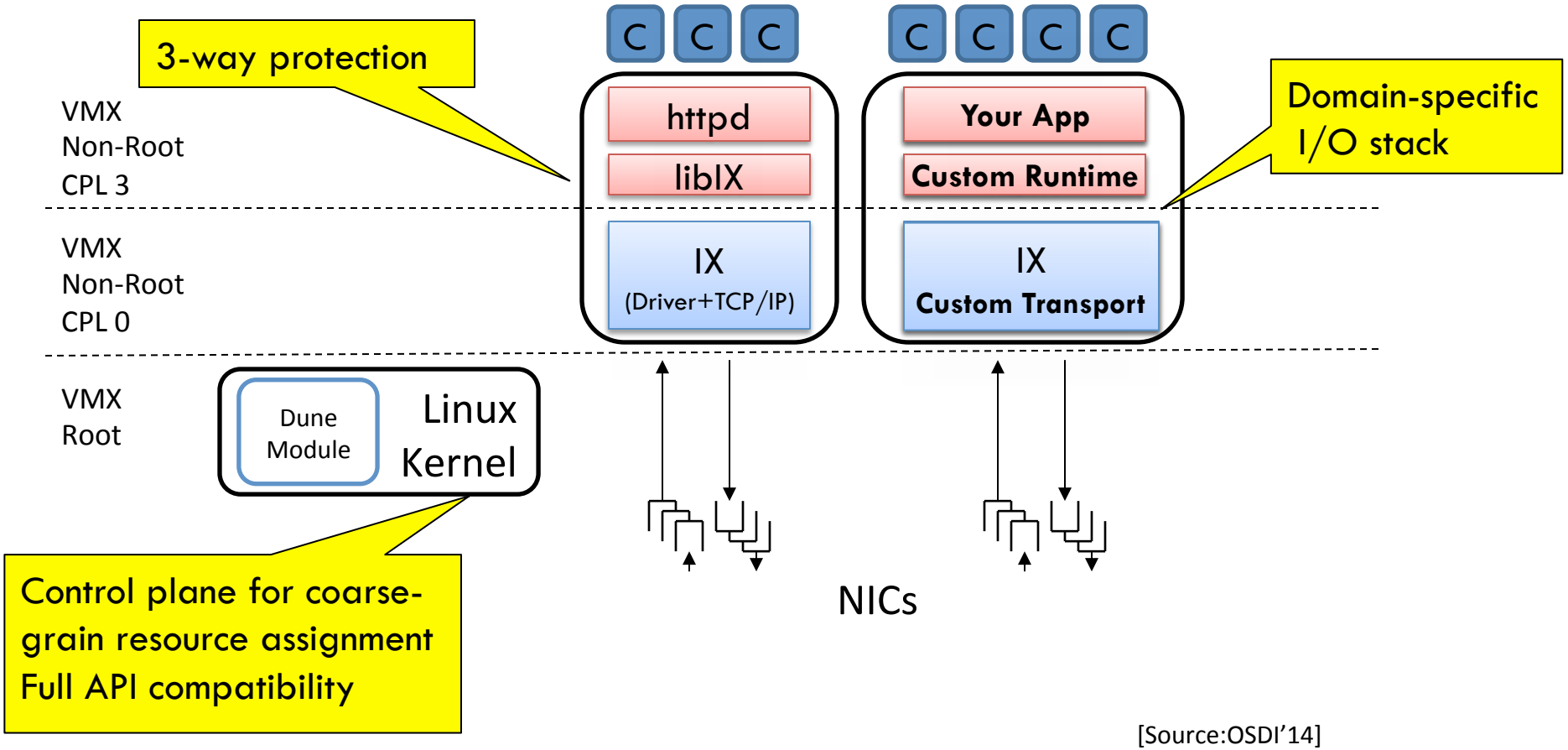
Efficiency mechanisms

- Run to completion
- Adaptive batching
- Flow-consistent hashing
- Zero copy

Scalable + practical APIs

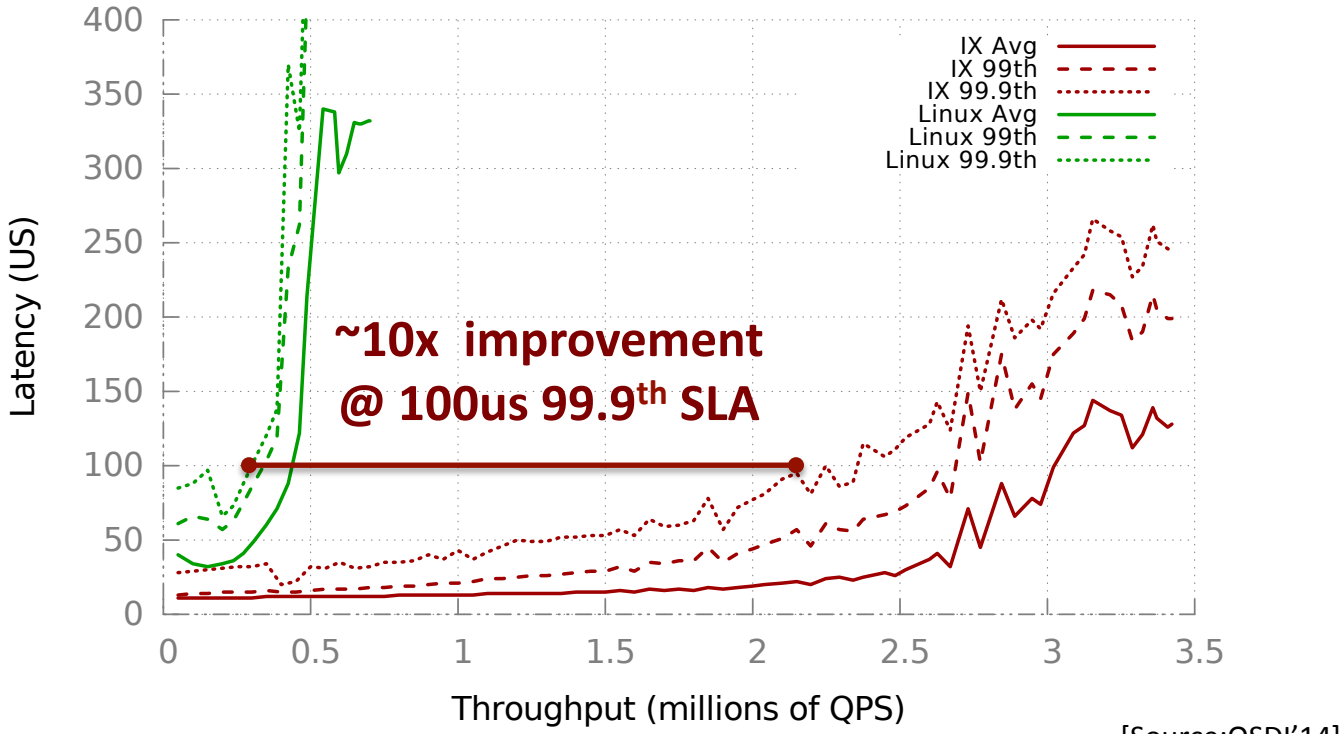
- POSIX sockets ✖, events/light-weight threads/futures ✔

System SW Implementation



[Source:OSDI'14]

Better Software Impact



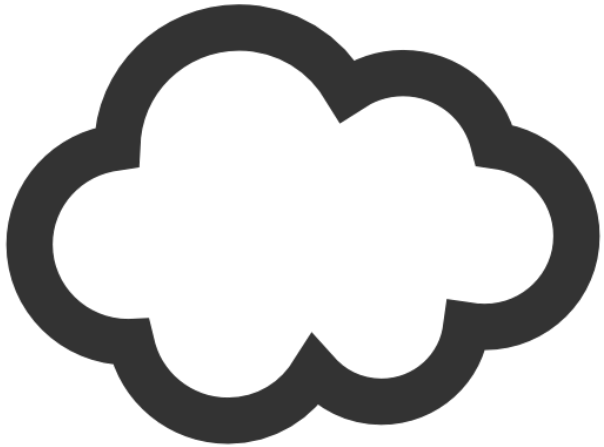
[Source:OSDI'14]

10x throughput &1/3 latency over Linux

Scalable, elastic, secure

Utilization

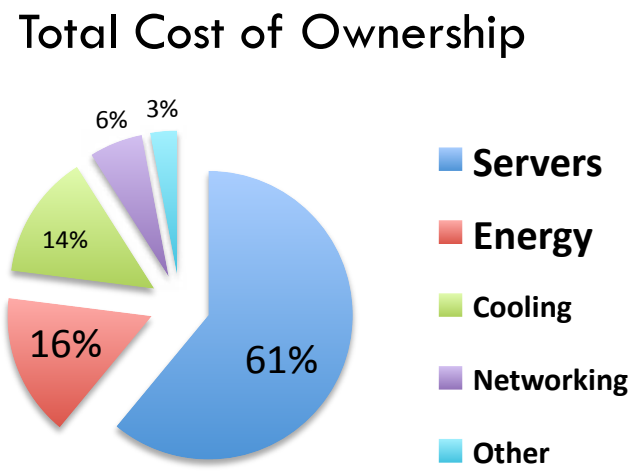
Why Does Utilization Matter?



The cloud makes specialized HW/SW practical

But it does not make them free

Cloud Economics



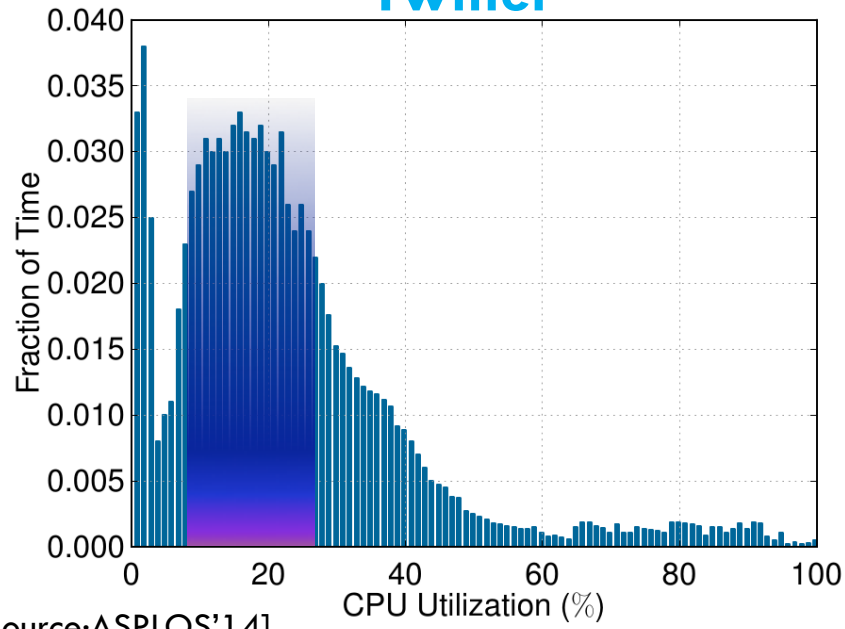
[Source: James Hamilton]

Hardware dominates TCO

Must use it as well as possible

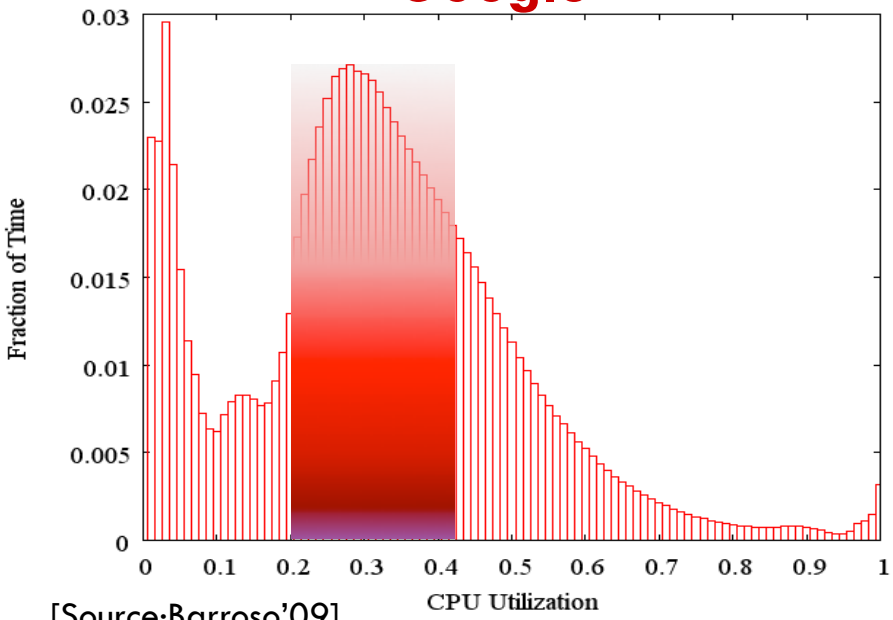
Cloud Utilization

Twitter



[Source:ASPLOS'14]

Google



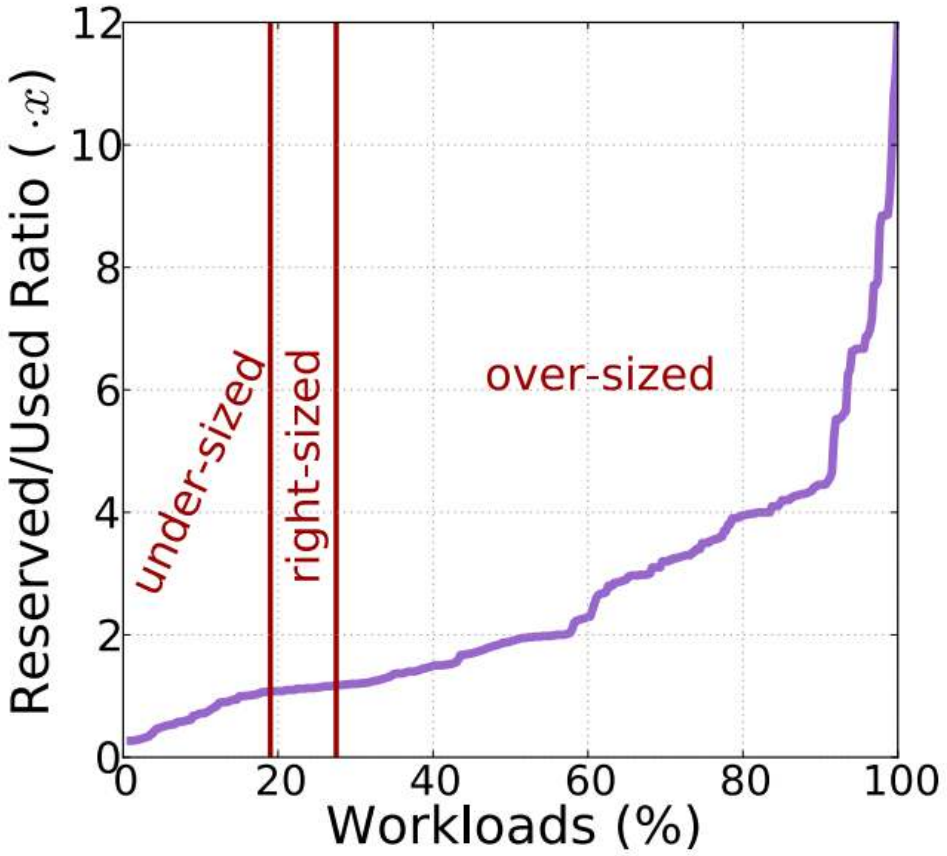
[Source:Barroso'09]

Low utilization in private & public clouds (<20%)

- Despite the aggressive use of multi-tenancy
- Specialized systems can make it worse

Why is Utilization Low?

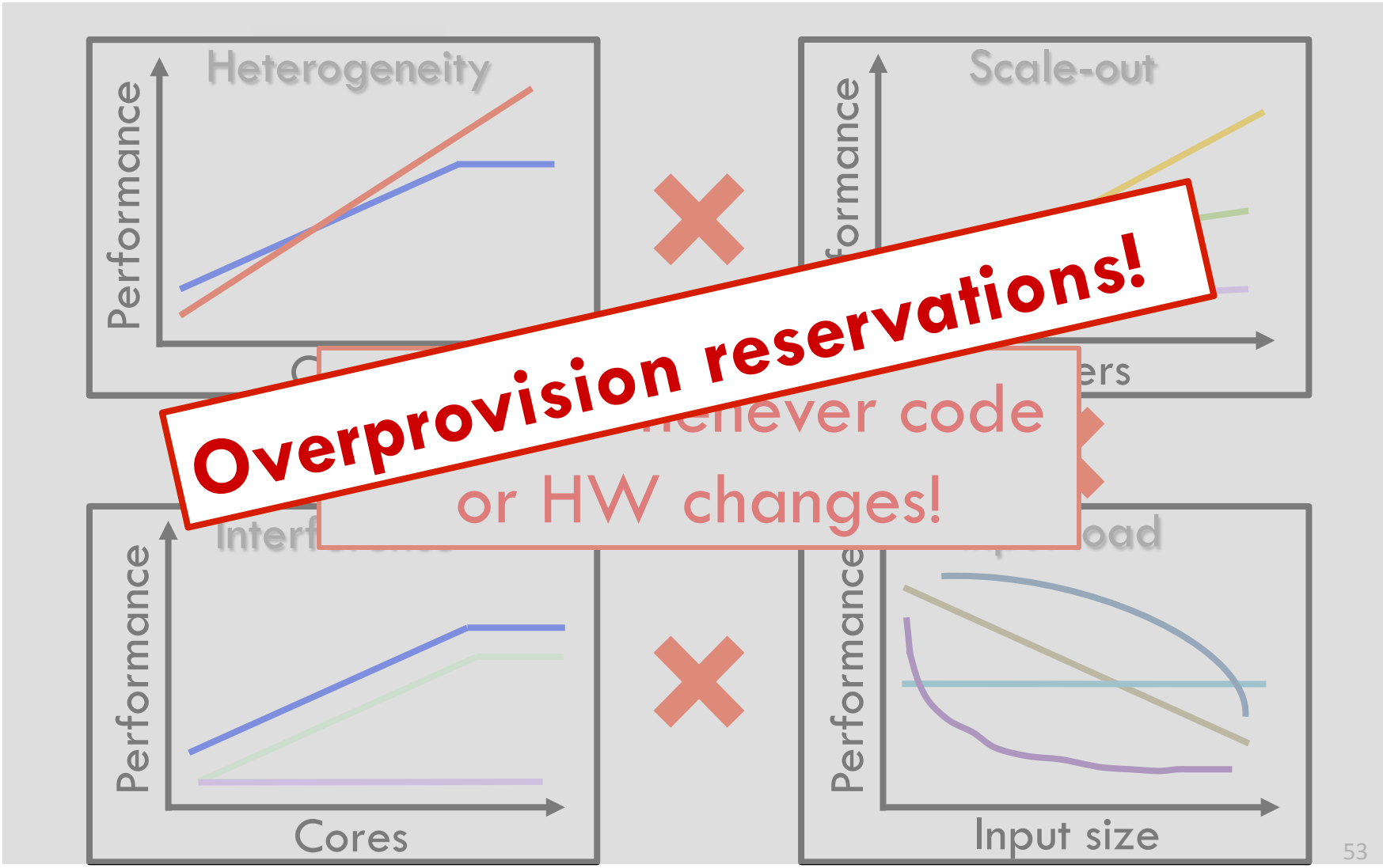
[Source:ASPLOS'14]



Overprovisioning @ Twitter

Similar challenges across the industry

Provisioning in the Cloud is Hard



The Path to High Utilization

Predictable systems

Reduce interference

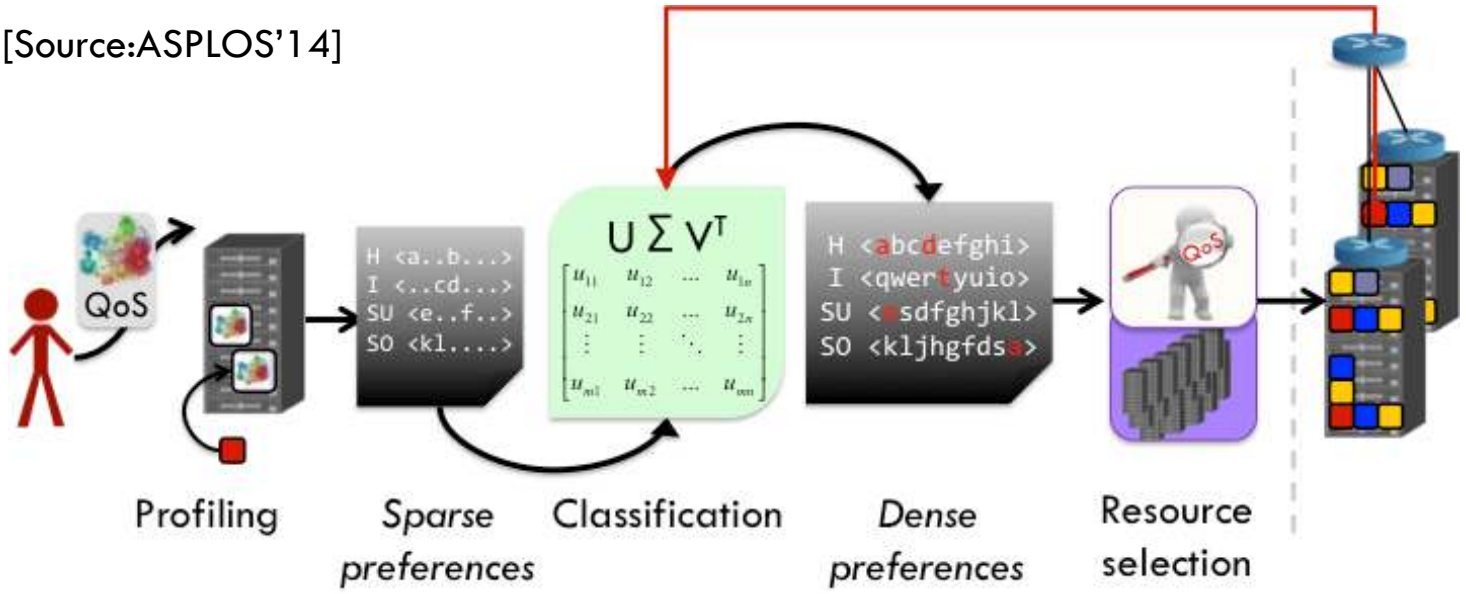
Elastic software

Reservations ✖, QoS goals ✔

Machine-learning to manage complex systems

High Utilization in the Cloud

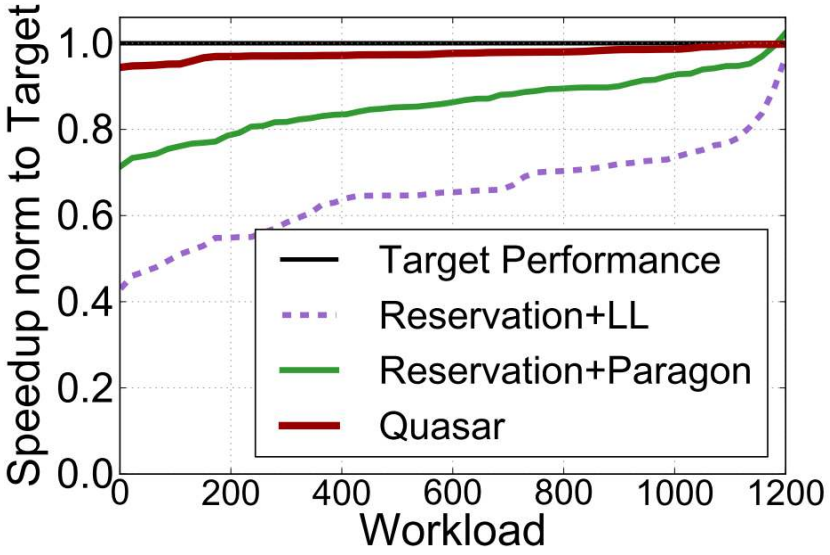
[Source:ASPLOS'14]

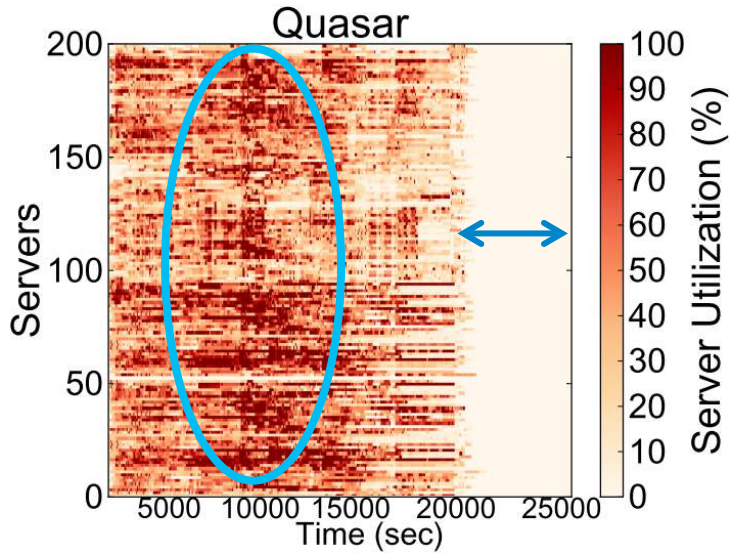


Quasar: apps QoS + ML to guide resource selection

- >70% cluster utilization
- Predictable app performance

High Utilization in the Cloud





Quasar: apps QoS + ML to guide resource selection

- >70% cluster utilization
- Predictable app performance

Summary

The bad news

- All systems are power limited



[Source:ThinkStock]

The good news

- We can built much better systems
- Lower energy/op + fewer ops/task

The challenges

- Design complexity, heterogeneity, utilization, …
- Think vertical: HW + system + app + management