
CERN

Performance Optimization

Why should we care?

Servesh Muralidharan

IT-DI-WLCG, UP Team

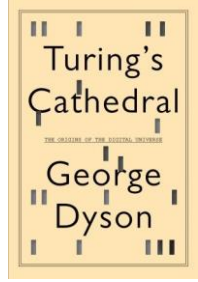
CERN

Nov 2018

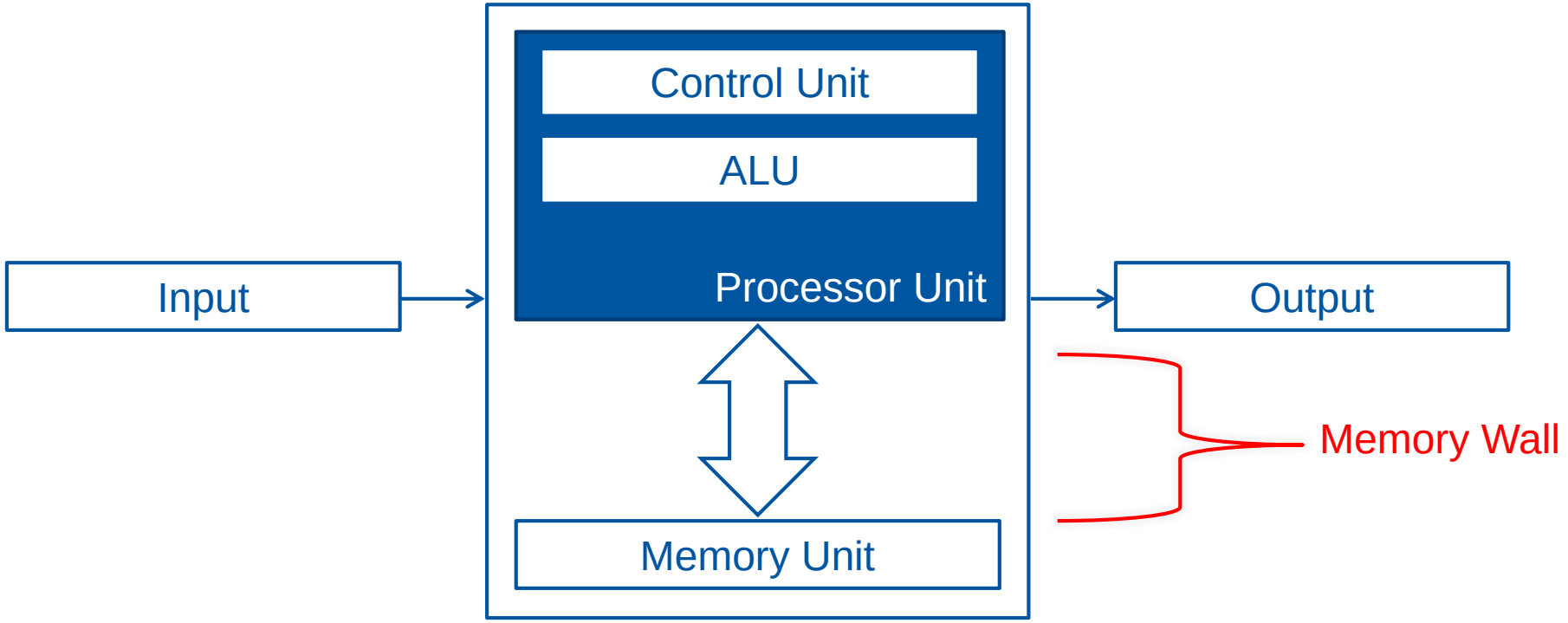
Optimization

The process of making something *as good* **or** *effective* **as possible.**

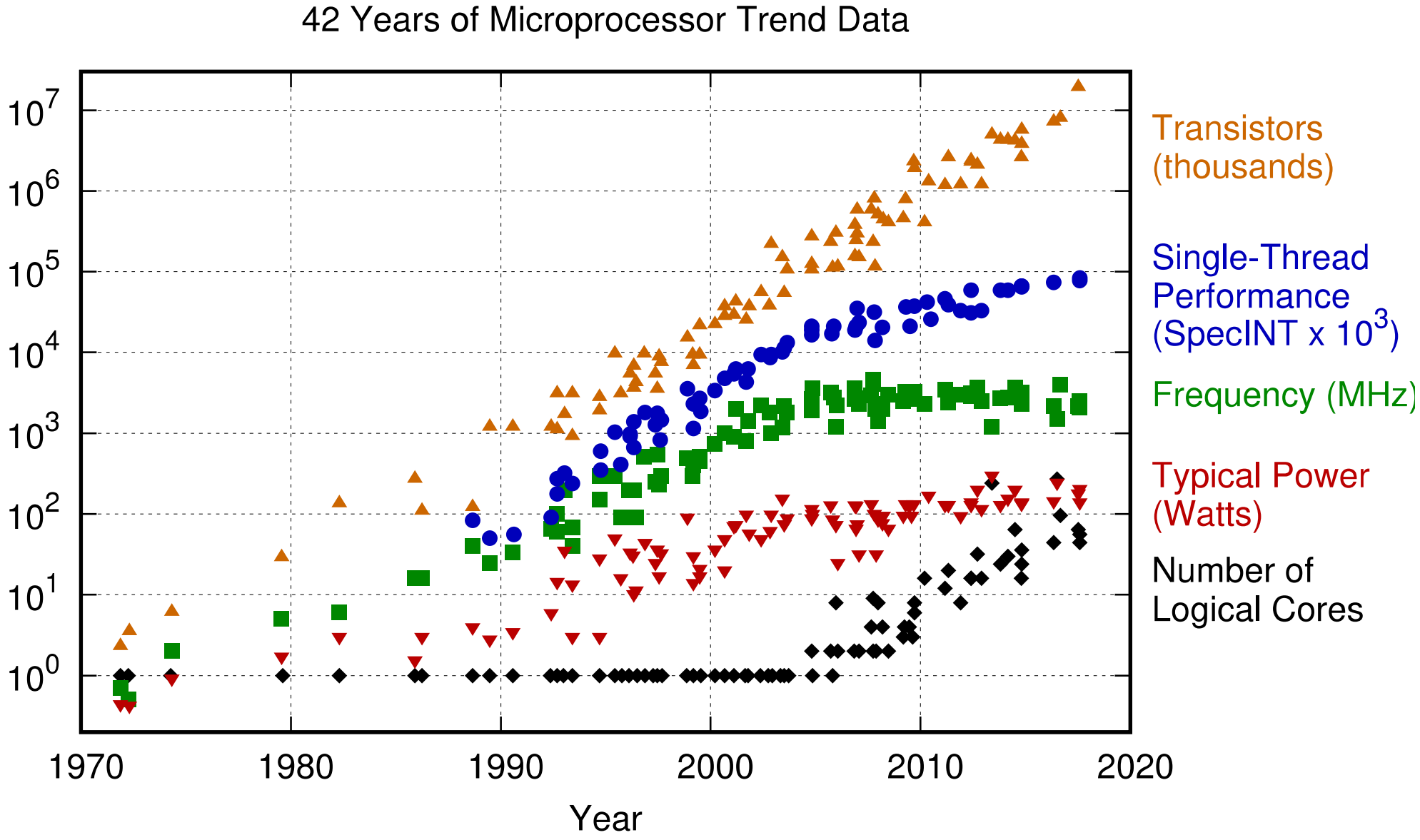
The Unchanged Computer Architecture



- Stored-Program computer
  - Von Neumann model
  - Program and data are stored in memory and then processed

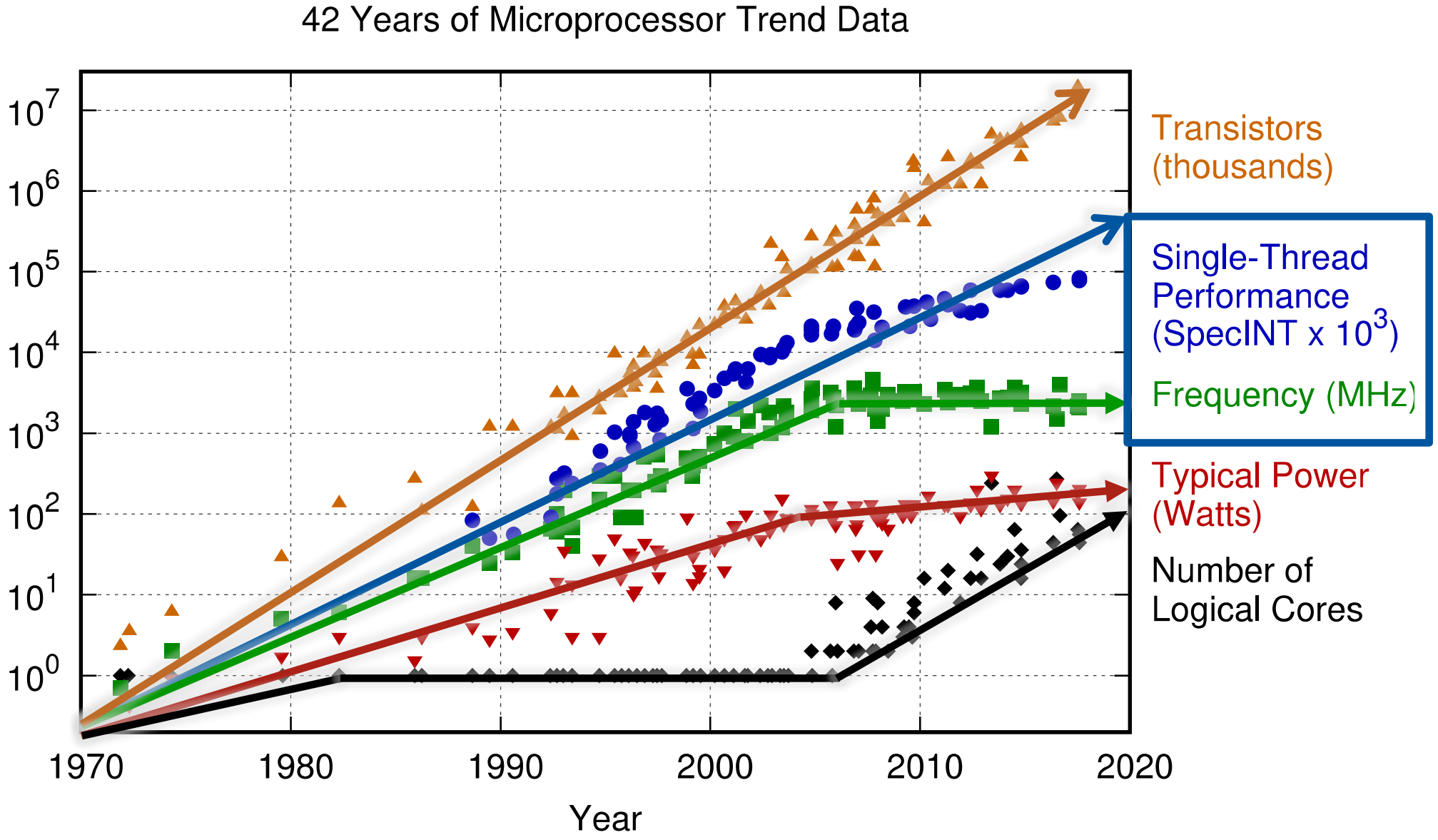


Moore's Law



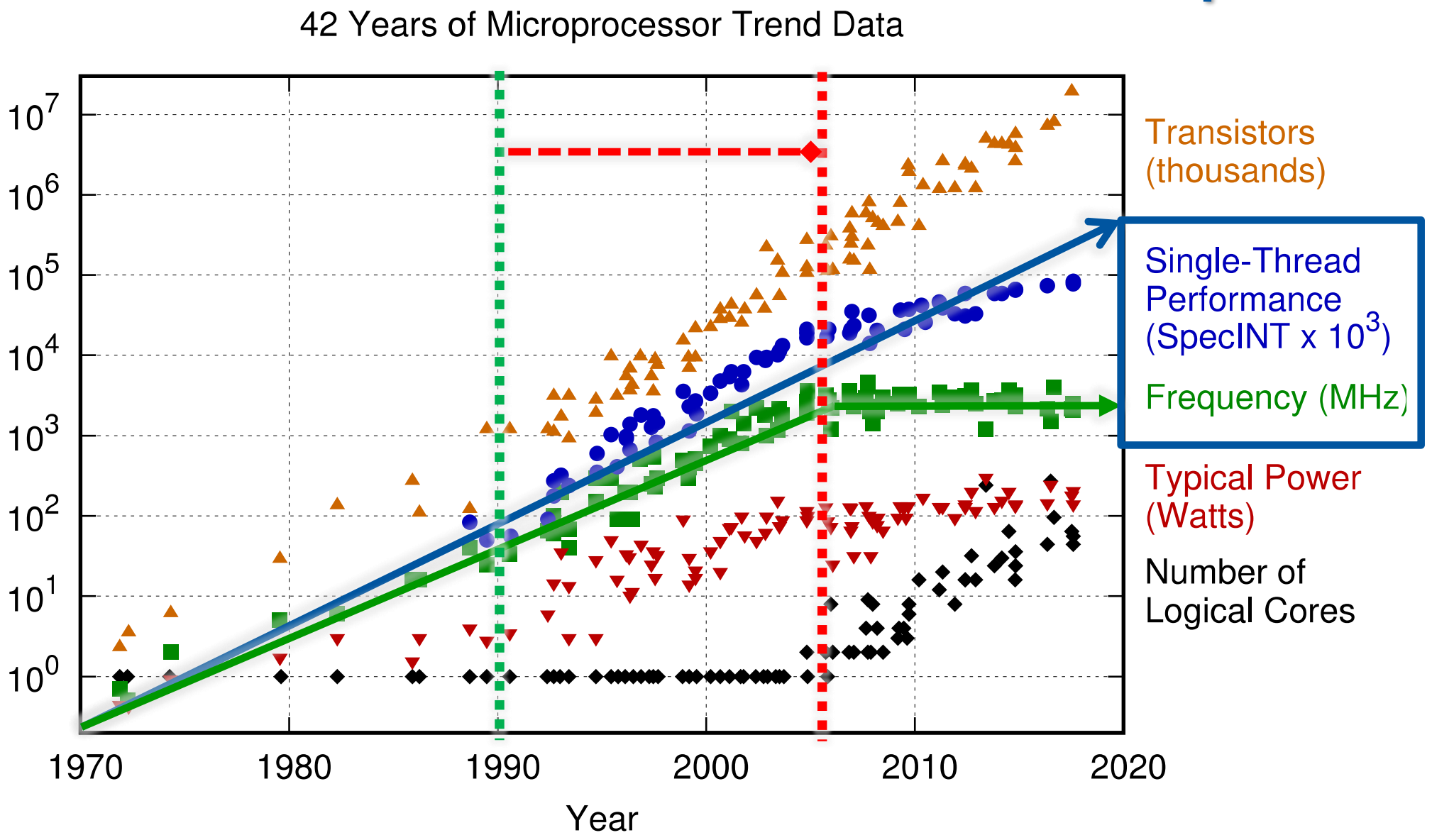
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
https://github.com/karlrupp/microprocessor-trend-data

Performance Trend



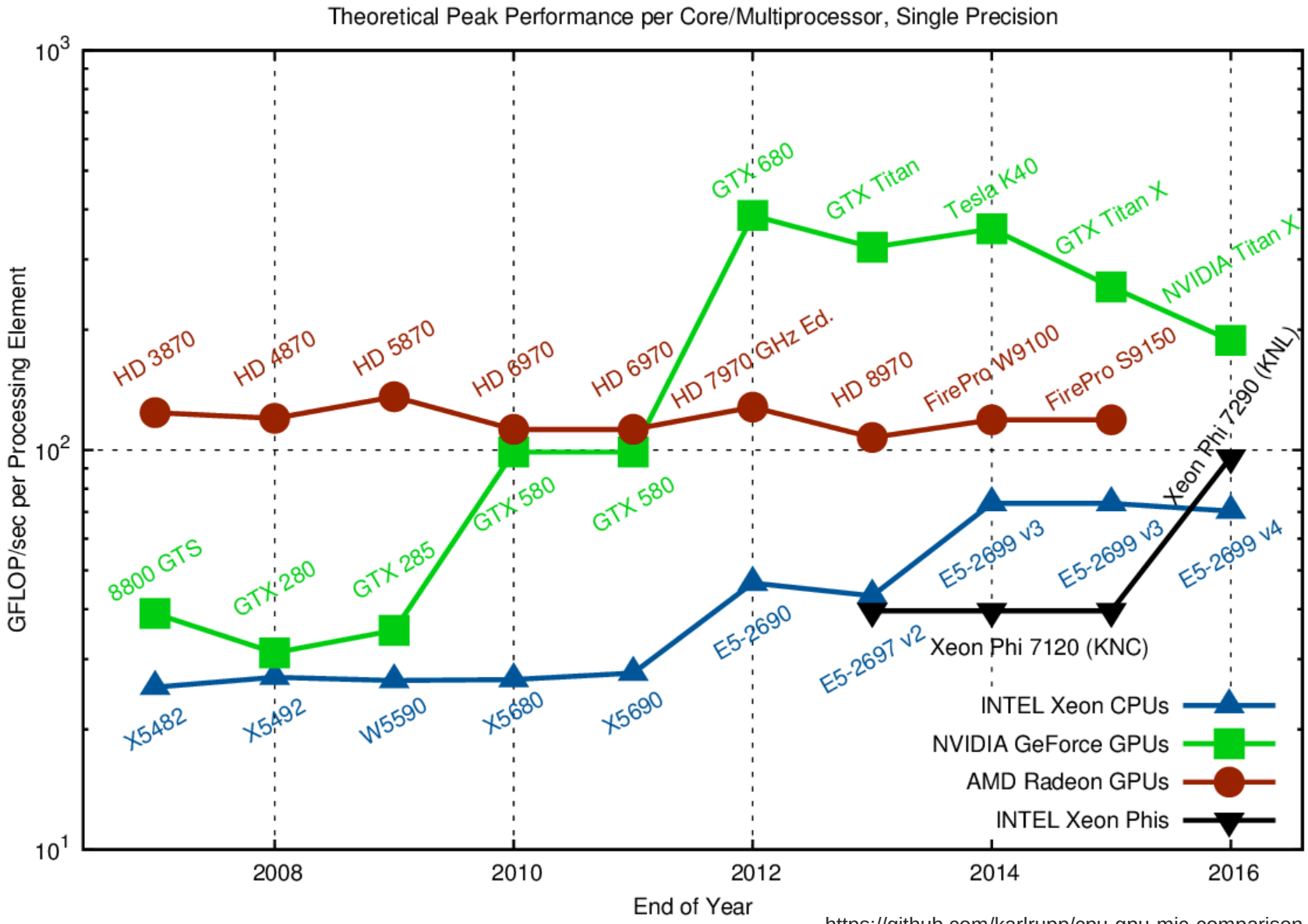
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
https://github.com/karlrupp/microprocessor-trend-data

The Free Lunch – When Did It Stop?



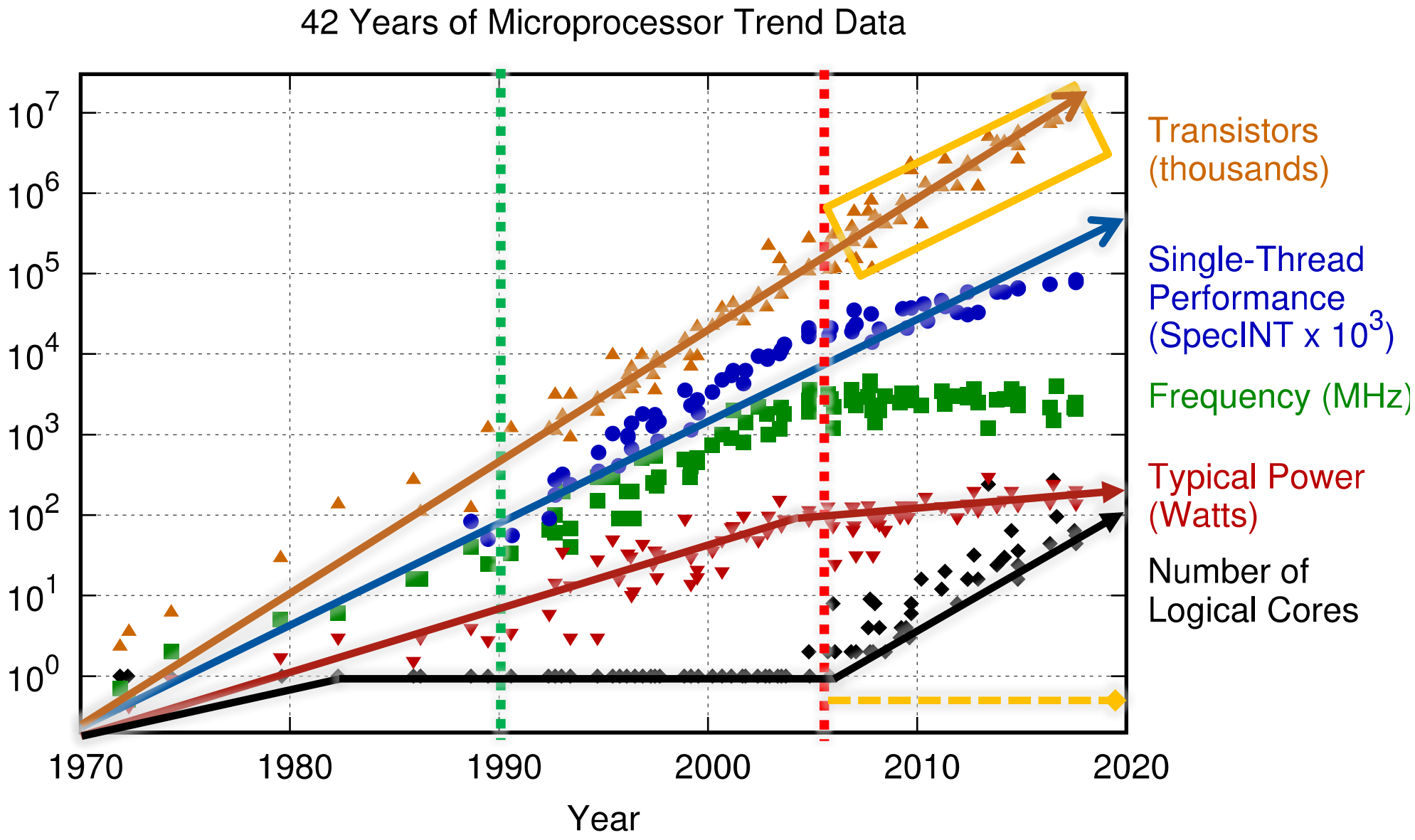
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
https://github.com/karlrupp/microprocessor-trend-data

Single Core Performance



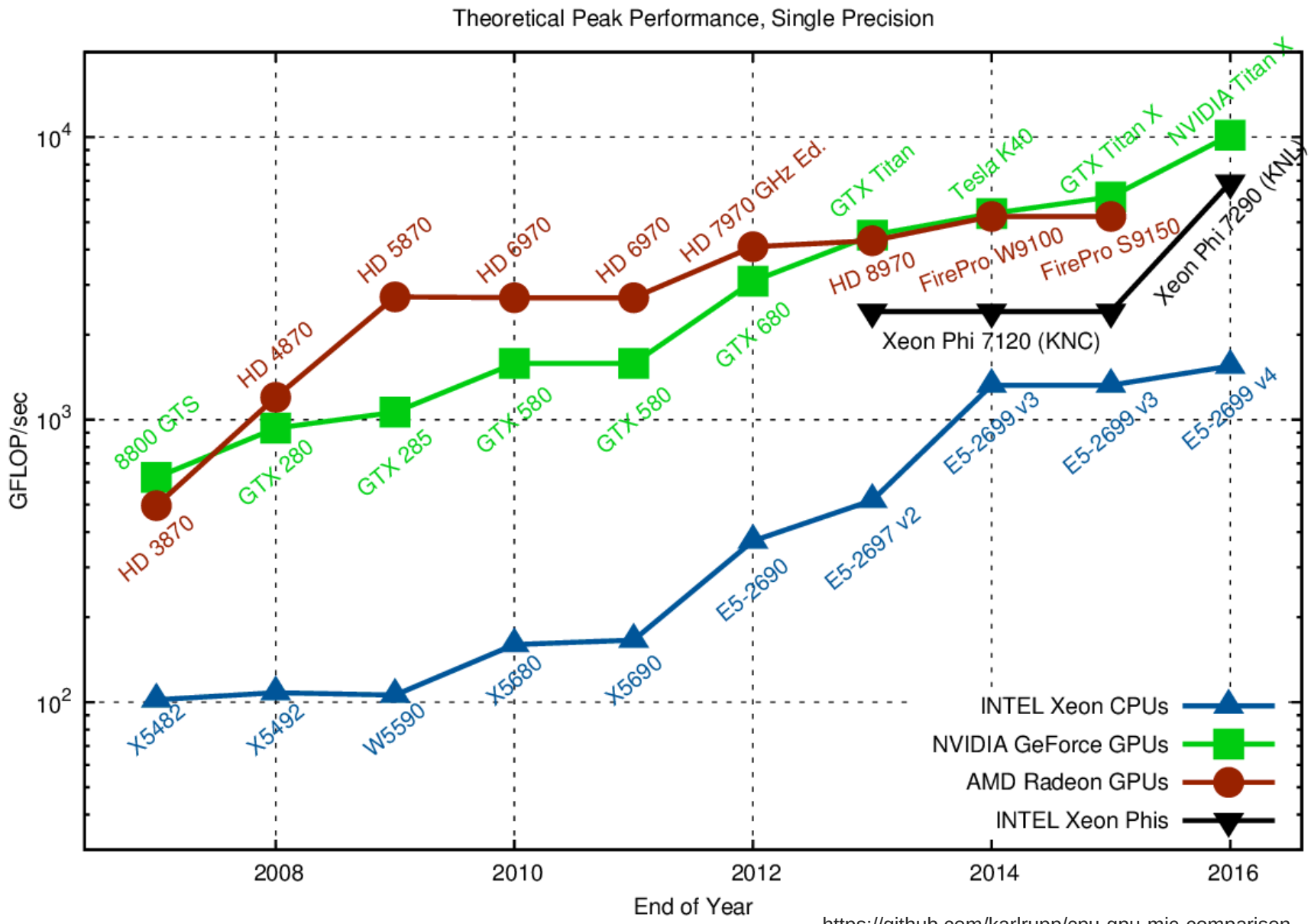
https://github.com/karlrupp/cpu-gpu-mic-comparison

Post Free Lunch



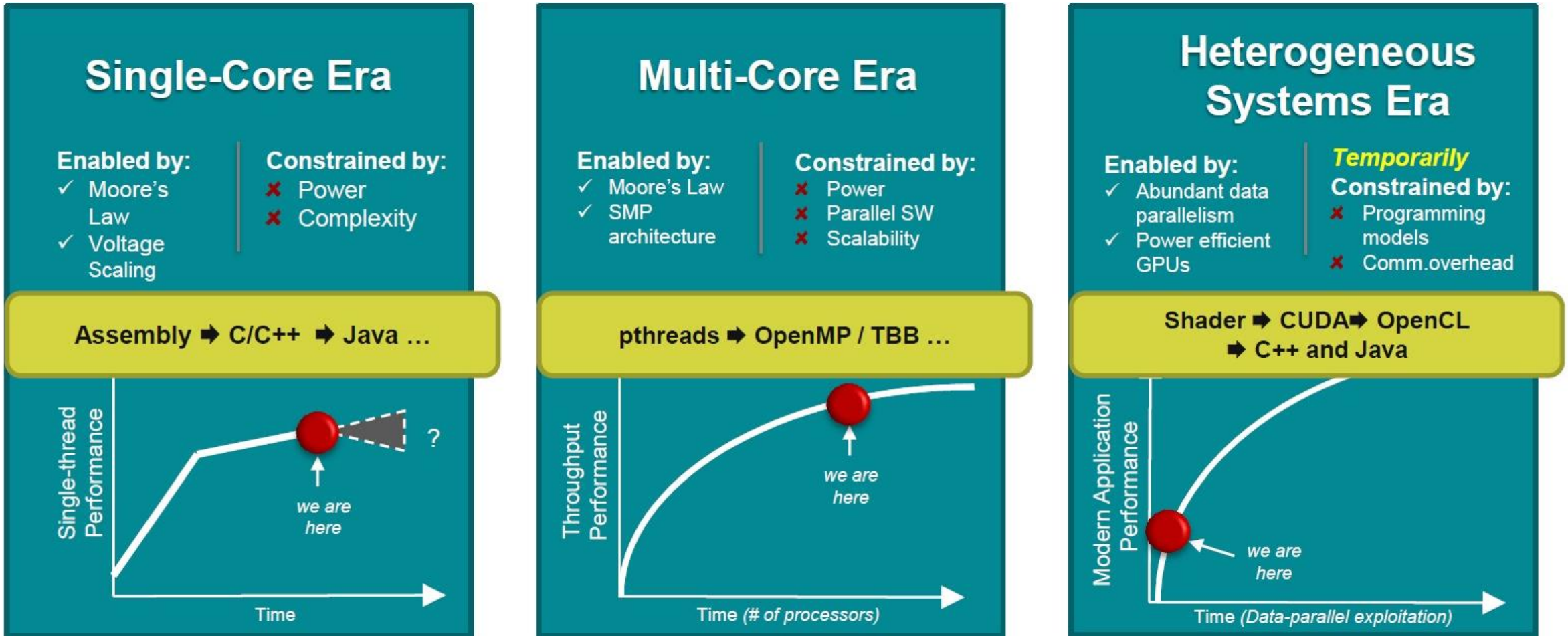
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
https://github.com/karlrupp/microprocessor-trend-data

Architectural Performance Trend



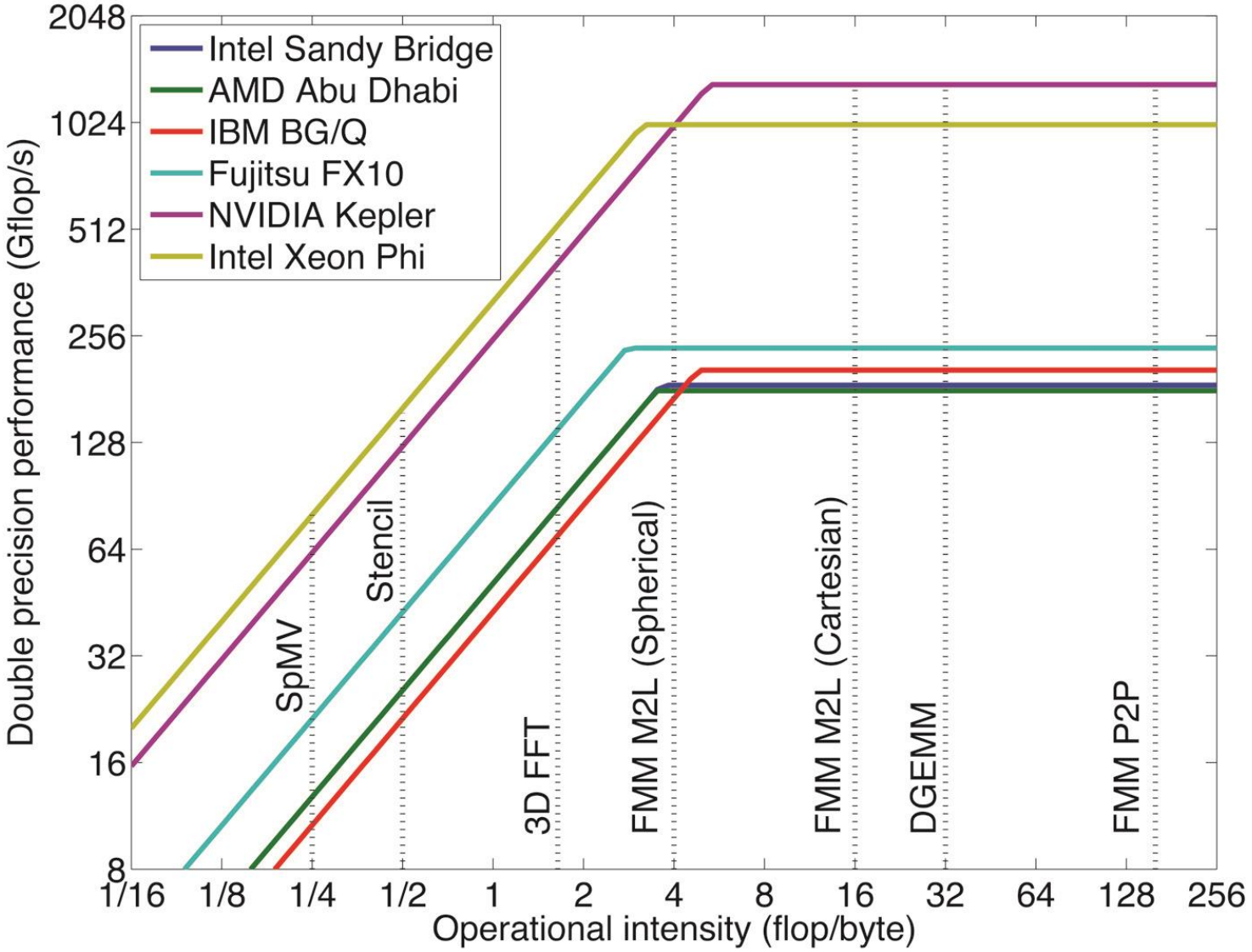
https://github.com/karlrupp/cpu-gpu-mic-comparison

The Future : Heterogeneous Architecture



https://opensourceforu.com/2016/12/how-heterogeneous-systems-evolved-and-the-challenges-going-forward/

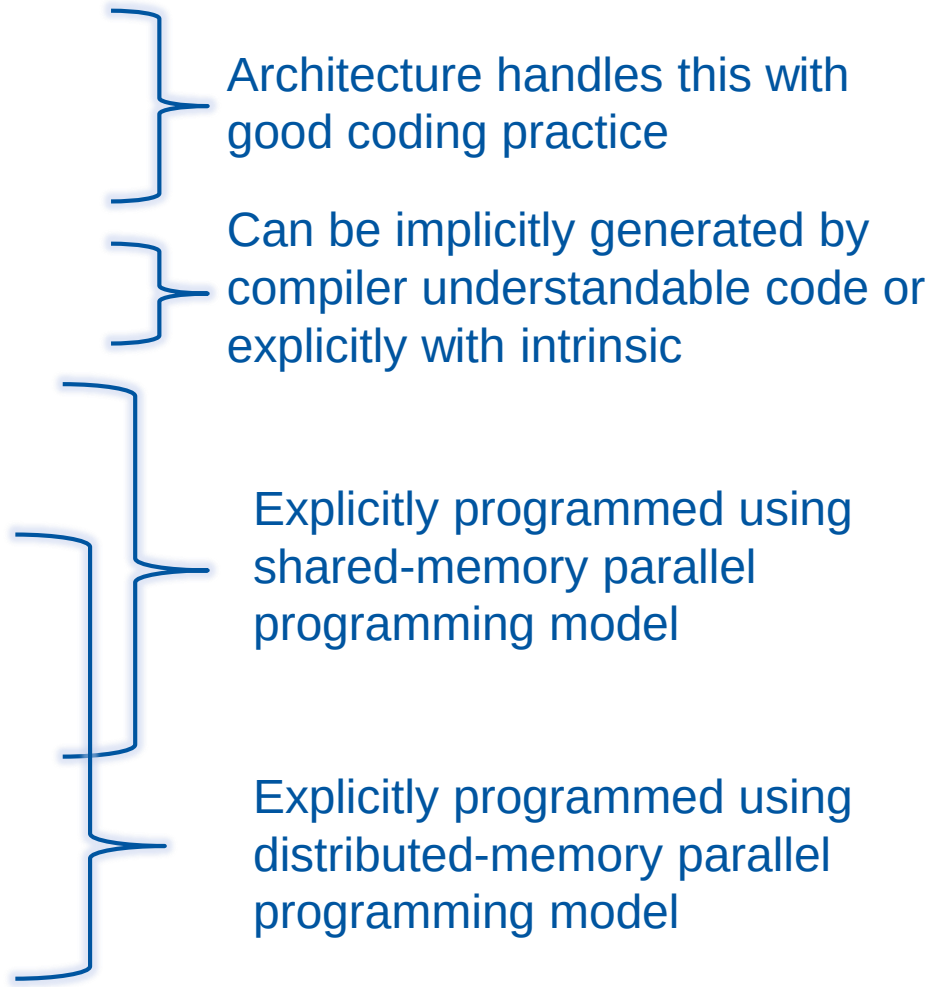
Roofline Performance Comparison



https://sinews.siam.org/About-the-Author/how-will-the-fast-multipole-method-fare-in-the-exascale-era

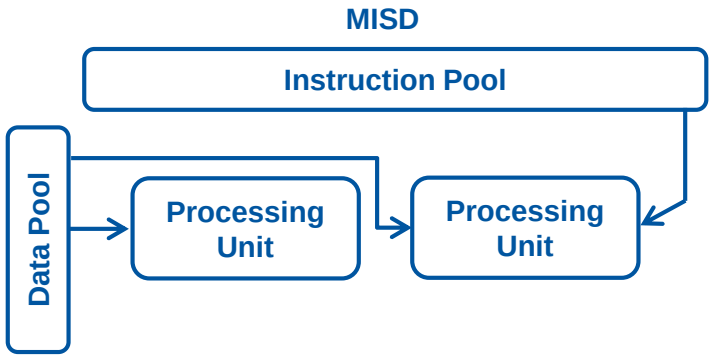
Sources of Parallelism in Modern Architectures

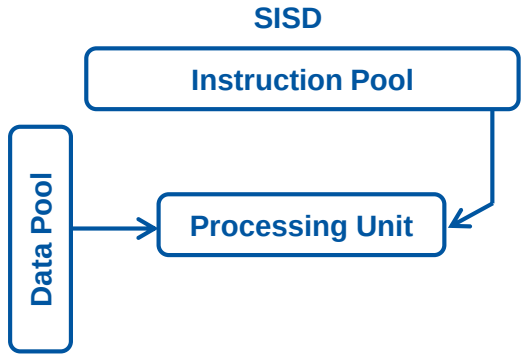
1. Instruction level parallelism (ILP)
2. Pipelining
3. Vector Operations
4. Hardware Threads
5. Multicore
6. Multi Socket
7. Cluster
8. Grid

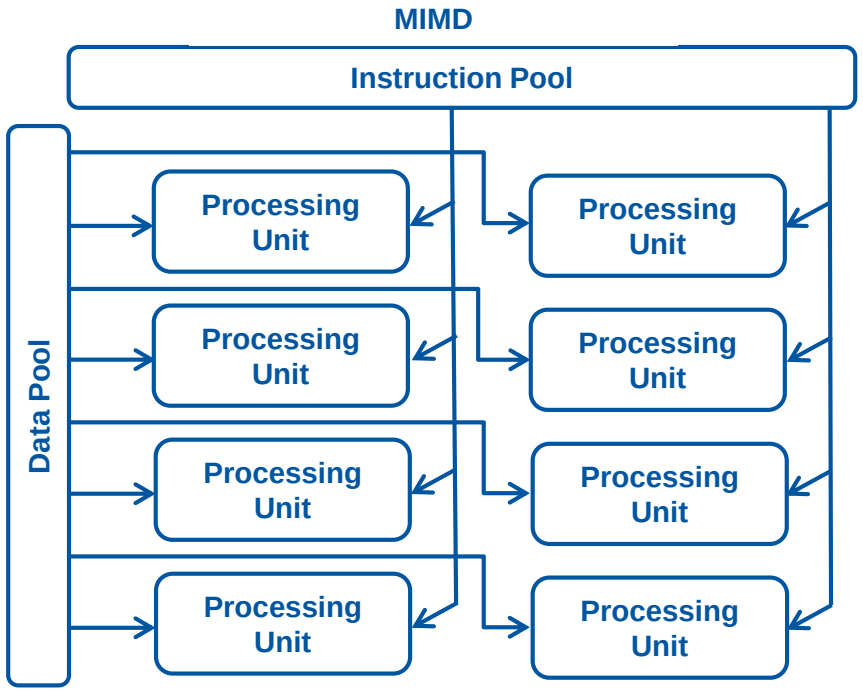


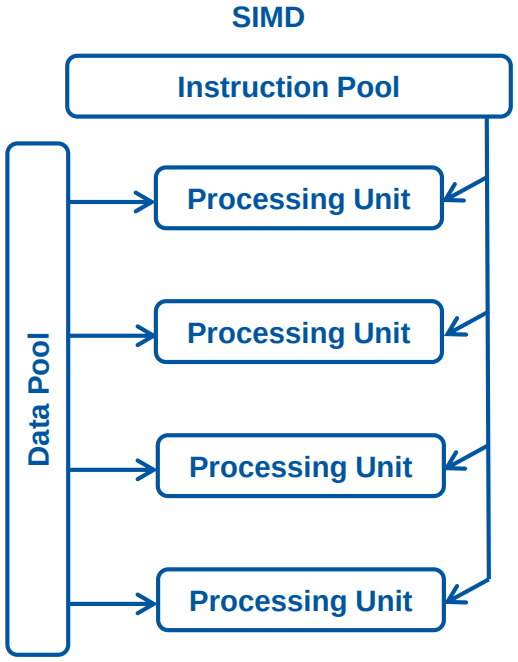
Flynn's taxonomy: Can be a programmer's guideline

- Proposed in 1966
- Instruction streams
  - single (SI) or multiple (MI)
- Data streams
  - single (SD) or multiple (MD)
- Types
  - SISD
  - SIMD
  - MISD
  - MIMD

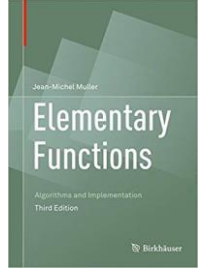








Concept of an Ideal Program



- Readability Vs Performance
- Compute Algorithm
  - **20% – 30% of code but has >90% of run time**
  - Most optimizations are applied here
  - *Big O notation*
    - Describes the worst case performance in terms of input size
    - Can represent time or space
    - Example: O(n) – Linear (Finding an item in an unsorted array)
  - Extremely readable code for **compilers**
    - Elegance and Obscurity
    - Compilers will love it and we only care about performance!!!
    - Be nice and use explicit comments for fellow human beings to understand
- Glue code
  - **70% – 80% of code but has <10% of run time**
  - Contains code used for structuring and connecting different blocks
  - Extremely readable code for **humans**
    - Compilers will hate you but that's okay, we don't care about performance here

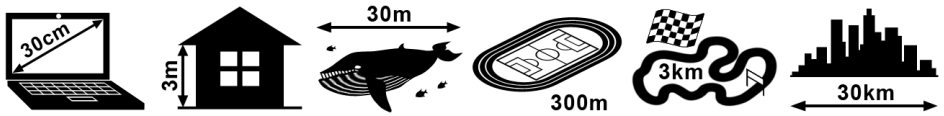
Not all CPU operations are created equal

ithare.com

Operation	Cost in CPU Cycles
“Simple” register-register op (ADD,OR,etc.)	<1
Memory write	~1
Bypass delay: switch between integer and floating-point units	0-3
“Right” branch of “if”	1-2
Floating-point/vector addition	1-3
Multiplication (integer/float/vector)	1-7
Return error and check	1-7
L1 read	3-4
TLB miss	7-21
L2 read	10-12
“Wrong” branch of “if” (branch misprediction)	10-20
Floating-point division	10-40
128-bit vector division	10-70
Atomics/CAS	15-30
C function direct call	15-30
Integer division	15-40
C function indirect call	20-50
C++ virtual function call	30-60
L3 read	30-70
Main RAM read	100-150
NUMA: different-socket atomics/CAS (guesstimate)	100-300
NUMA: different-socket L3 read	100-300
Allocation+deallocation pair (small objects)	200-500
NUMA: different-socket main RAM read	300-500
Kernel call	1000-1500
Thread context switch (direct costs)	2000
C++ Exception thrown+caught	5000-10000
Thread context switch (total costs, including cache invalidation)	10000 - 1 million

10^0 10^1 10^2 10^3 10^4 10^5 10^6

Distance which light travels while the operation is performed



http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles/

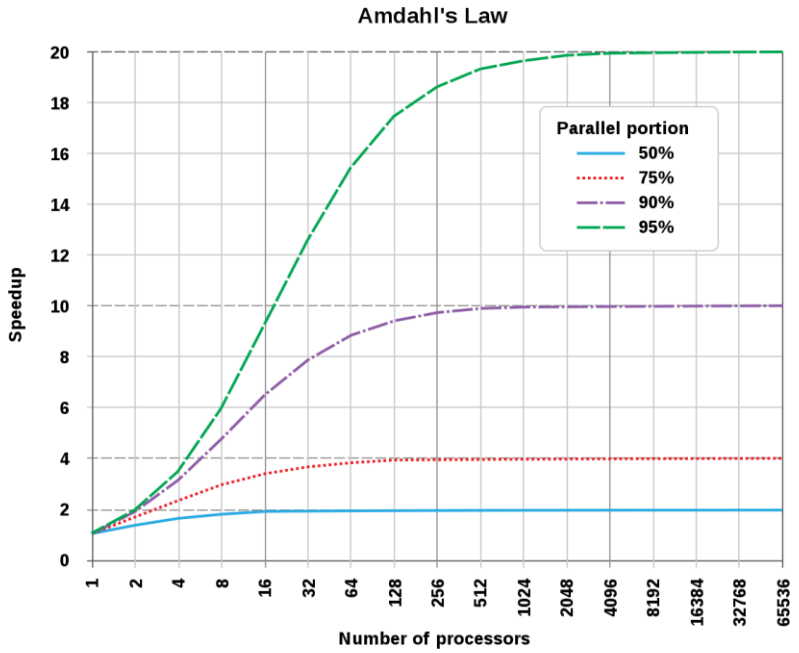
Parallel computing

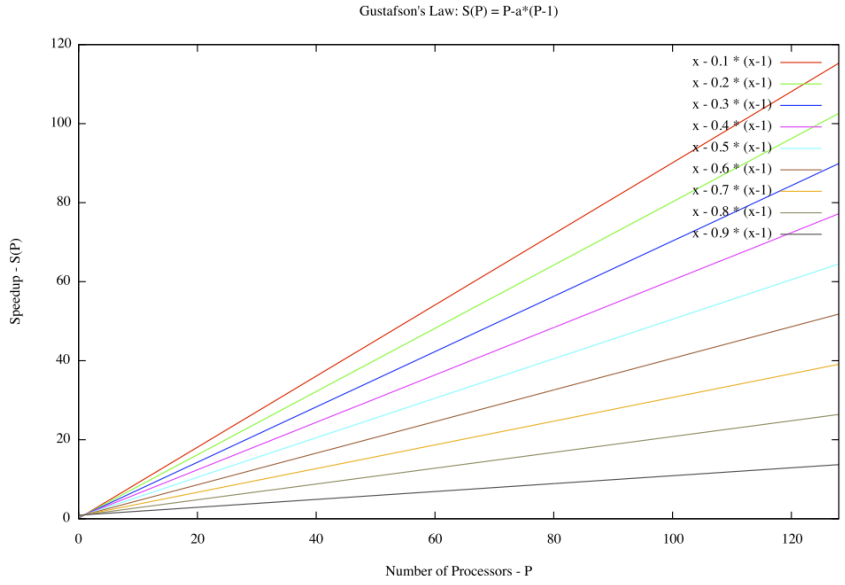
- Performing certain computations simultaneously using multiple resources
- **Amdahl's law** & **Gustafson's law**
  - Speedup only comes from the parallelizable part of the code
  - i.e. Serial part of the code will impact or limit the achievable performance

Amdahl's law $$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

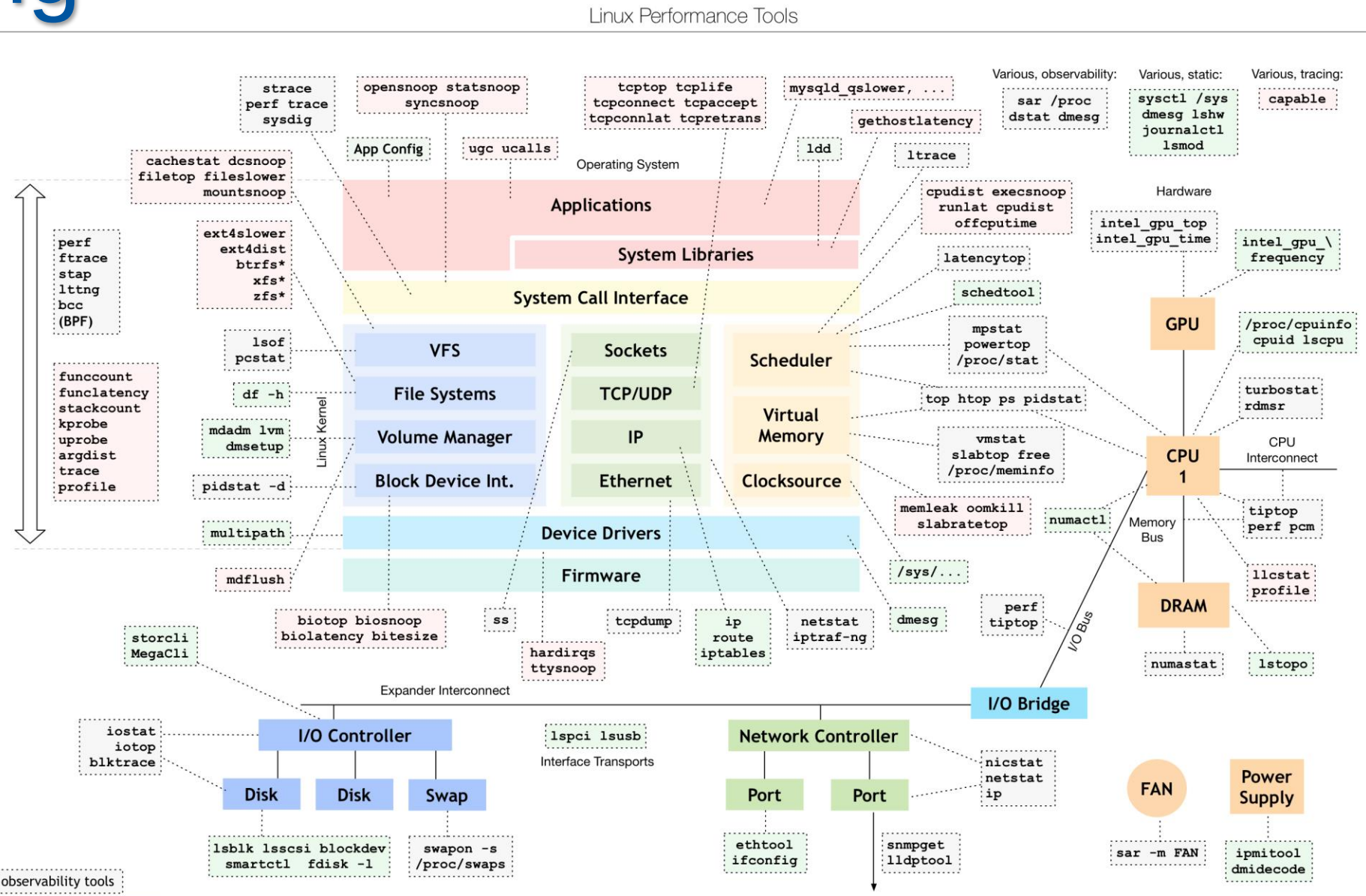
Gustafson's law $$S_{\text{latency}}(s) = 1 - p + sp,$$

Where,
S_{latency} is the theoretical overall speedup
s is the speedup in the parallel part
p is the percentage of the execution time of serial part



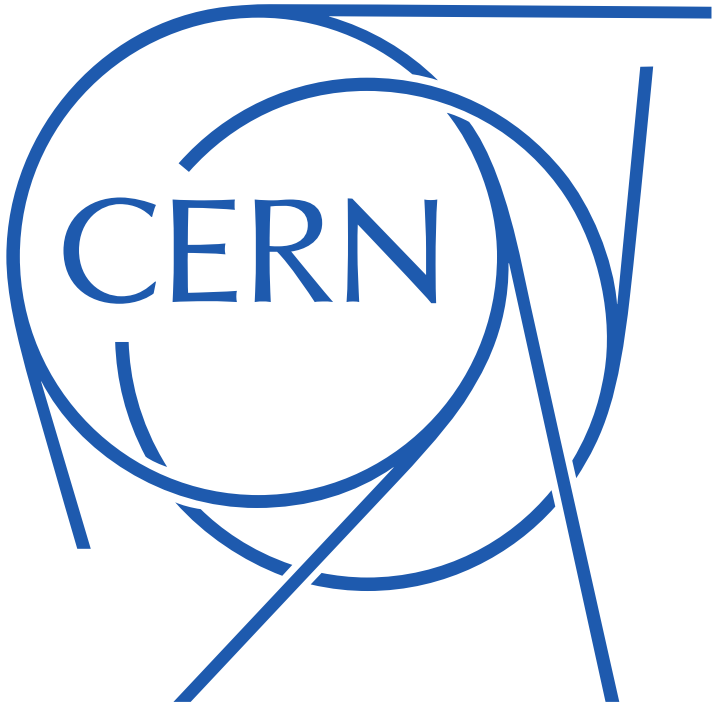


Profiling


Linux Performance Tools
Various, observability:
Various, static:
Various, tracing:
sar /proc dstat dmesg
sysctl /sys dmesg lshw journalctl lsmod
capable
strace perf trace sysdig
opensnoop statsnoop syncsnoop
tcptop tcplife tcpconnect tcpaccept tcpconnlat tcpretrans
mysqld_qslower, ...
gethostlatency
App Config
ugc ucalls
ldd
ltrace
Operating System
Applications
System Libraries
System Call Interface
cachestat dcsnoop filetop fileslower mountsnoop
perf ftrace stap lttng bcc (BPF)
ext4slower ext4dist btrfs* xfs* zfs*
cpudist execsnoop runlat cpudist offcputime
Hardware
intel_gpu_top intel_gpu_time
intel_gpu_\ frequency
latencytop
schedtool
GPU
/proc/cpuinfo cpuid lscpu
mpstat powertop /proc/stat
lsof pcstat
df -h
mdadm lvm dmsetup
pidstat -d
multipath
mdflush
funccount funclatency stackcount kprobe uprobe argdist trace profile
Linux Kernel
VFS
File Systems
Volume Manager
Block Device Int.
Sockets
TCP/UDP
IP
Ethernet
Scheduler
Virtual Memory
Clocksource
top htop ps pidstat
vmstat slabtop free /proc/meminfo
turbostat rdmsr
CPU 1
CPU Interconnect
memleak oomkill slabratetop
numactl
Memory Bus
tiptop perf pcm
llcstat profile
Device Drivers
Firmware
/sys/...
dmesg
DRAM
perf tiptop
I/O Bus
biotop biosnoop biolatency bitesize
ss
tcpdump
ip route iptables
netstat iptraf-ng
hardirqs ttysnoop
numastat
lstopo
storcli MegaCli
Expander Interconnect
I/O Bridge
iostat iotop blktrace
I/O Controller
lspci lsusb
Interface Transports
Network Controller
Disk
Disk
Swap
Port
Port
nicstat netstat ip
FAN
Power Supply
lsblk lsscsi blockdev smartctl fdisk -l
swapon -s /proc/swaps
ethtool ifconfig
snmpget lldptool
sar -m FAN
ipmitool dmidecode
observability tools
static performance tools
perf-tools/bcc tracing tools
these can observe the state of the system at rest, without load
https://github.com/brendangregg/perf-tools https://github.com/iovisor/bcc
style inspired by reddit.com/u/redct
http://www.brendangregg.com/linuxperf.html 2017

Conclusions

- Optimization is not an extra step
- Identification of a suitable form of parallelism for the algorithm
- Architecture agnostic implementation
- Reversible customizations
- Profile and Optimize


CERN