Memory Bandwidth and System Balance in HPC Systems

John D. McCalpin, PhD

mccalpin@tacc.utexas.edu

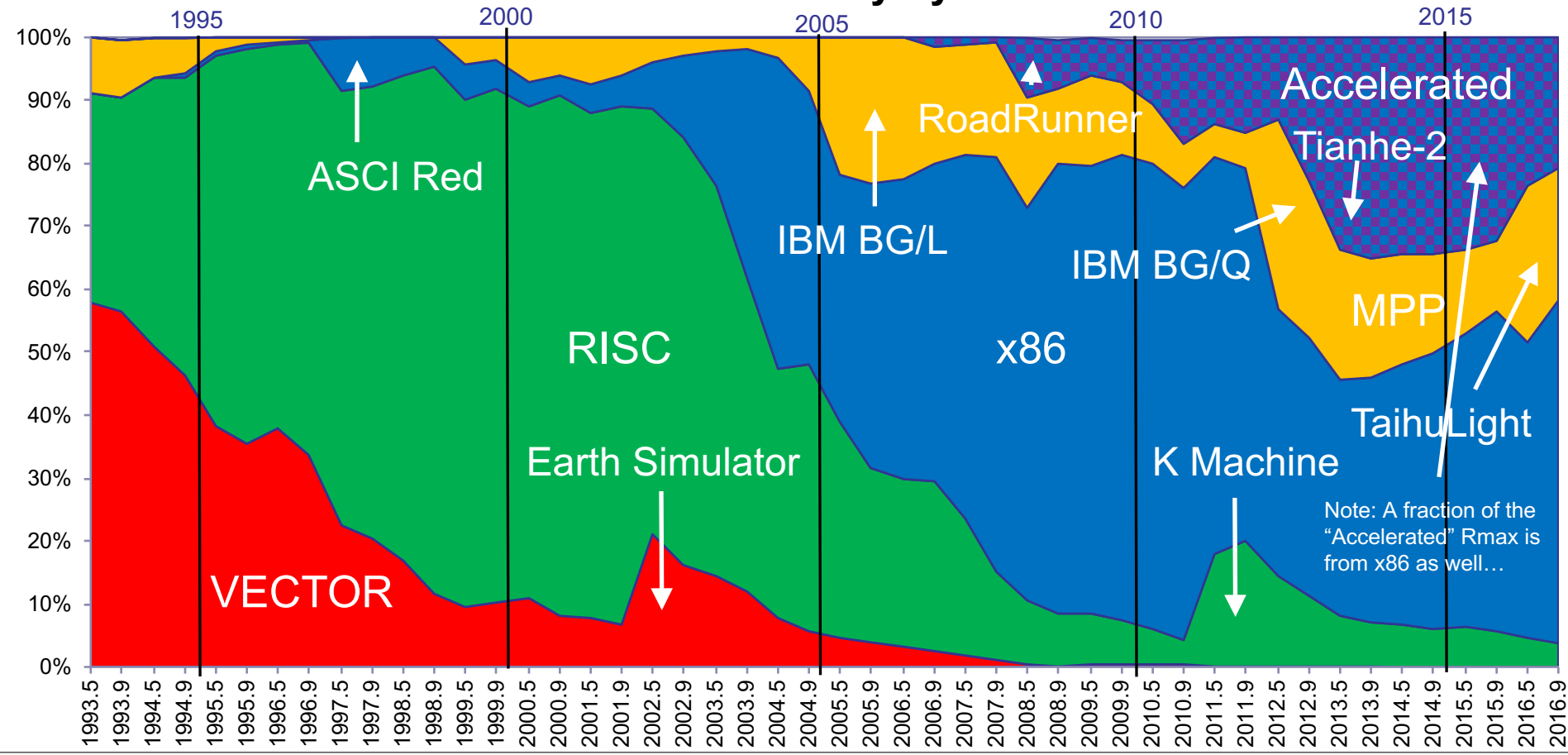
Outline

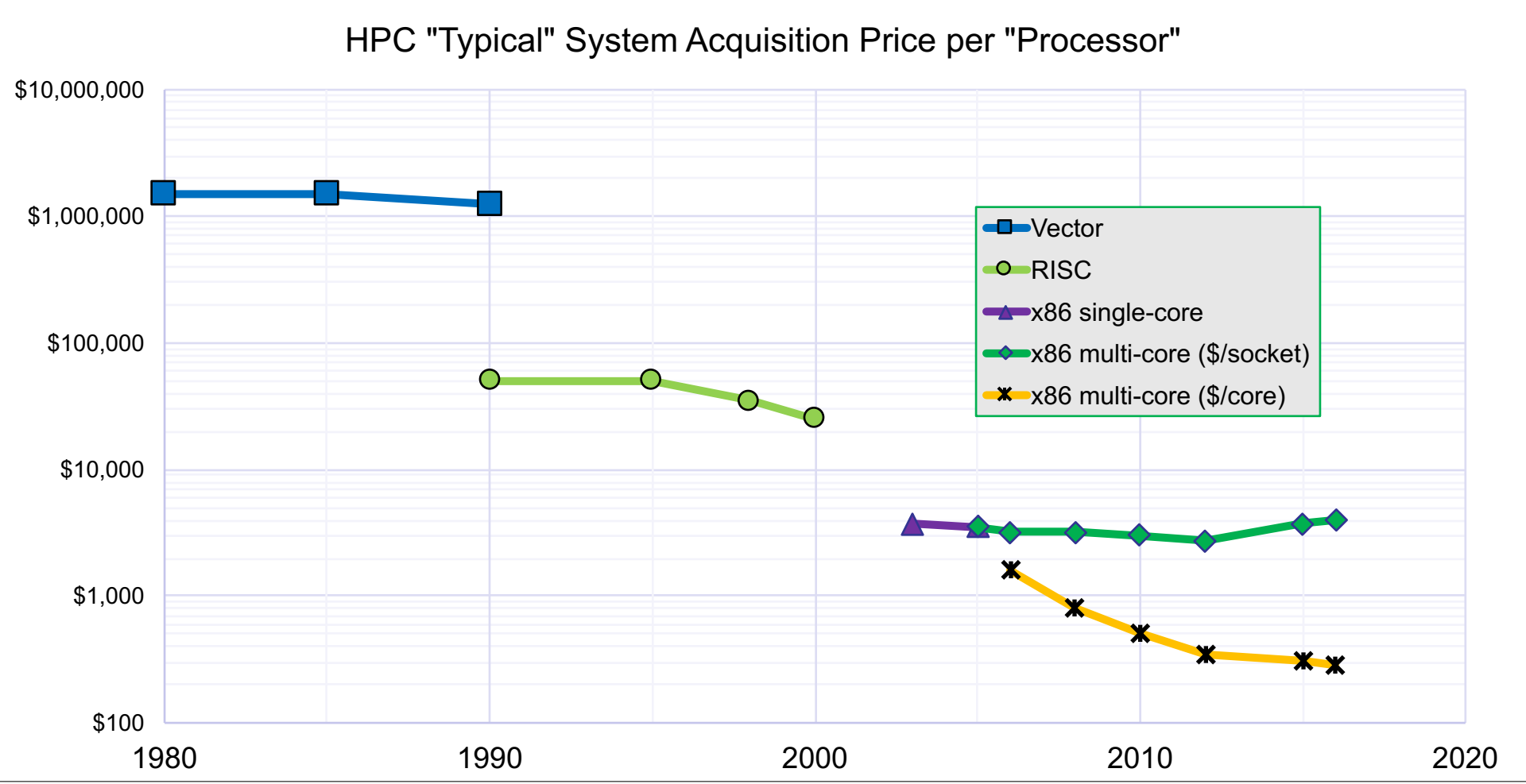
1. Changes in TOP500 Systems
   - System Architectures & System Sizes
2. Technology Trends & System Balances
   - The STREAM Benchmark
   - Computation Rates vs Data Motion Latency and Bandwidth
   - Required Concurrency to Exploit available Bandwidths
3. Comments & Speculations…
   - Impact on HPC systems and on HPC Applications
   - Potential disruptive technologies

Part 1

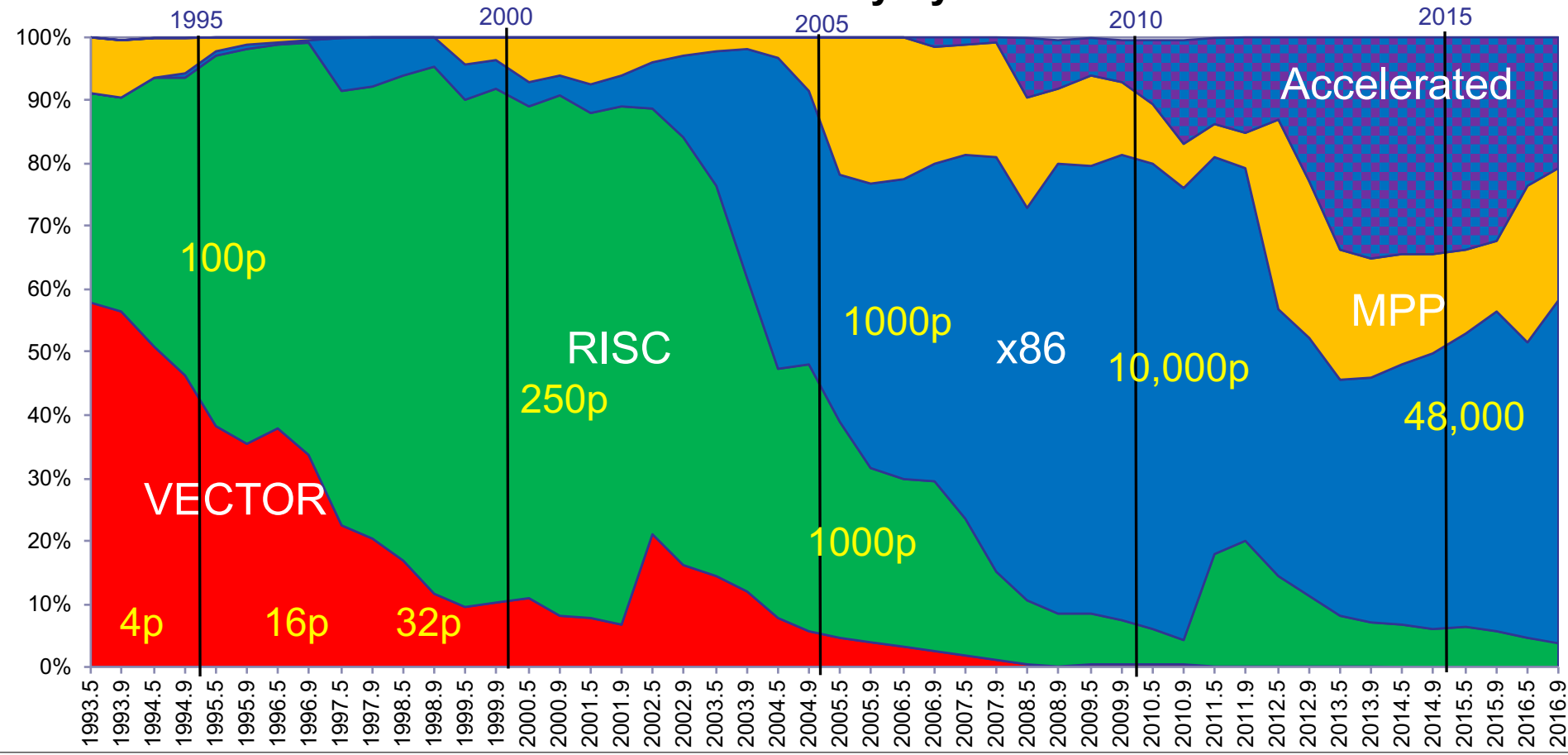
CHANGES IN TOP500 SYSTEMS

TOP500 Rmax Contributions by System Architecture




HPC "Typical" System Acquisition Price per "Processor"
$10,000,000
$1,000,000
$100,000
$10,000
$1,000
$100
1980
1990
2000
2010
2020
Vector
RISC
x86 single-core
x86 multi-core ($/socket)
x86 multi-core ($/core)

TOP500 Rmax Contributions by System Architecture



Heterogeneous Systems

- More sites are building “clusters of clusters”, e.g.:
  - Sub-cluster 1: 2-socket nodes with *small memory*
  - Sub-cluster 2: 2-socket nodes with *large memory*
  - Sub-cluster 3: 4-socket nodes with *very large memory*
  - Sub-cluster 4: nodes with *accelerators*, etc…
- Consistent with observed partial shift to accelerators
  - Peaking at ~30% of the aggregate Rmax of the list in 2013-2015
    - Under 20% for November 2016 list (after excluding x86 contributions)
  - Peaking at ~20% of the systems in 2015, now under 17%
    - No more than 16% of systems have had >2/3 of their Rpeak in accelerators
  - Accelerators have been split between many-core and GPU

Part 2

TECHNOLOGY TRENDS & SYSTEM BALANCES

The STREAM Benchmark

- Created in 1991 while on faculty at the University of Delaware College of Marine Studies
- Intended to be an extremely simplified representation of the low-compute-intensity, long-vector operations characteristic of ocean circulation models
- Widely used for research, testing, marketing
- Almost 1100 results in database at main site
- Hosted at *www.cs.virginia.edu/stream/*

The STREAM Benchmark (2)

- Four kernels, separately timed:

```
Copy:  C[i] = A[i];   i=1..N
Scale: B[i] = scalar * C[i]; i=1..N
Add:   C[i] = A[i] + B[i]; i=1..N
Triad  A[i] = B[i] + scalar*C[i]; i=1..N
```

- "N" chosen to make each array >> cache size
- Repeated (typically) 10 times, first iteration ignored
- Min/Avg/Max timings reported, best time used for BW

The STREAM Benchmark (3)

- Assumed Memory Traffic per index:

```
Copy:  C[i] = A[i];                  16 Bytes
Scale: B[i] = scalar * C[i];         16 Bytes
Add:   C[i] = A[i] + B[i];           24 Bytes
Triad  A[i] = B[i] + scalar*C[i];    24 Bytes
```

- Many systems will read data before updating it ("write allocate" policy)
  - STREAM ignores this extra traffic if present

What are "*System Balances*"?

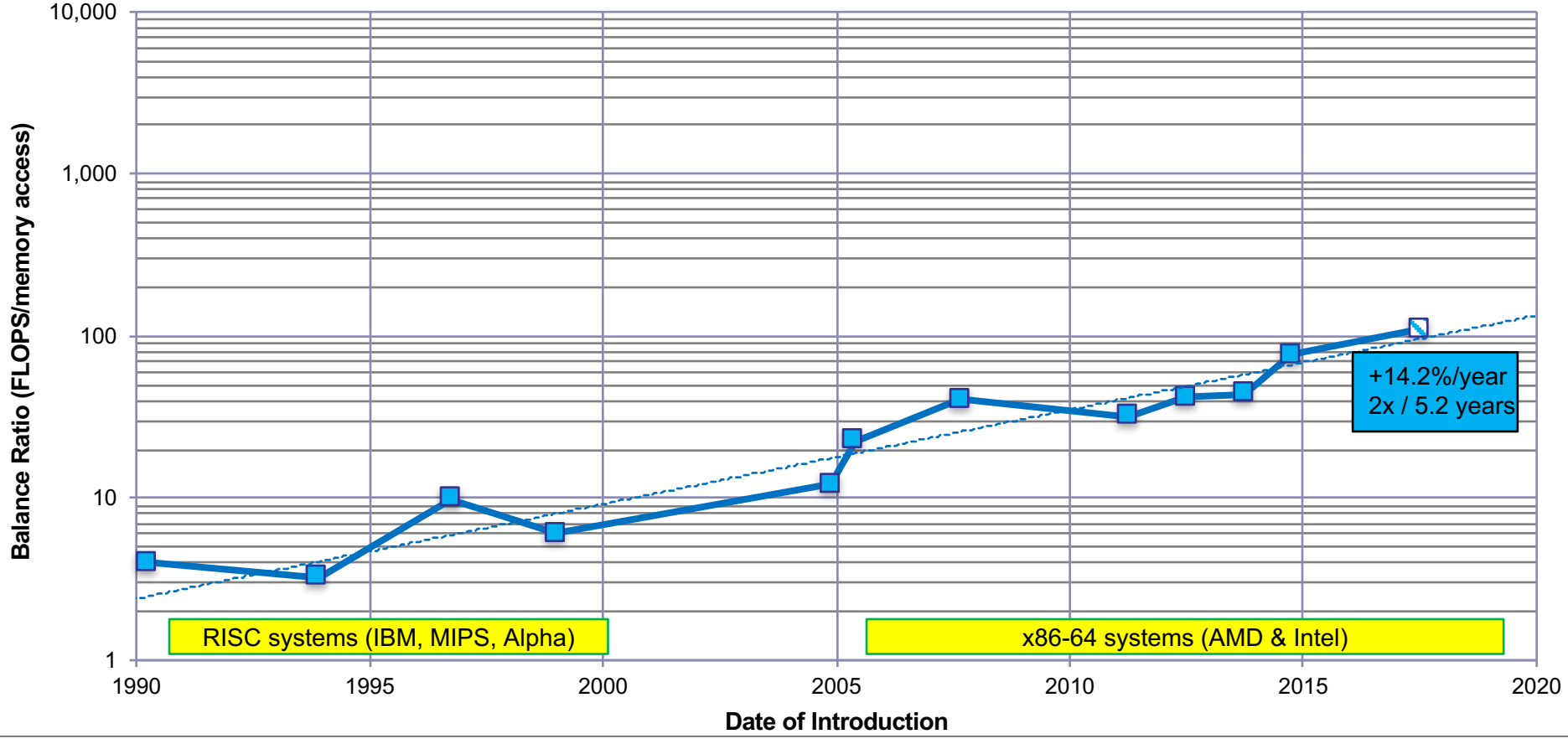
- "Performance" can be viewed as an N-dimensional vector of "mostly-orthogonal" components, e.g.:
  - Core performance (FLOPs) – *LINPACK*
  - Memory Bandwidth – *STREAM*
  - Memory Latency – *lmbench/lat_mem_rd*
  - Interconnect Bandwidth – *osu_bw, osu_bibw*
  - Interconnect Latency – *osu_latency*
- *System Balances* are the ratios of these components

Performance Component Trends

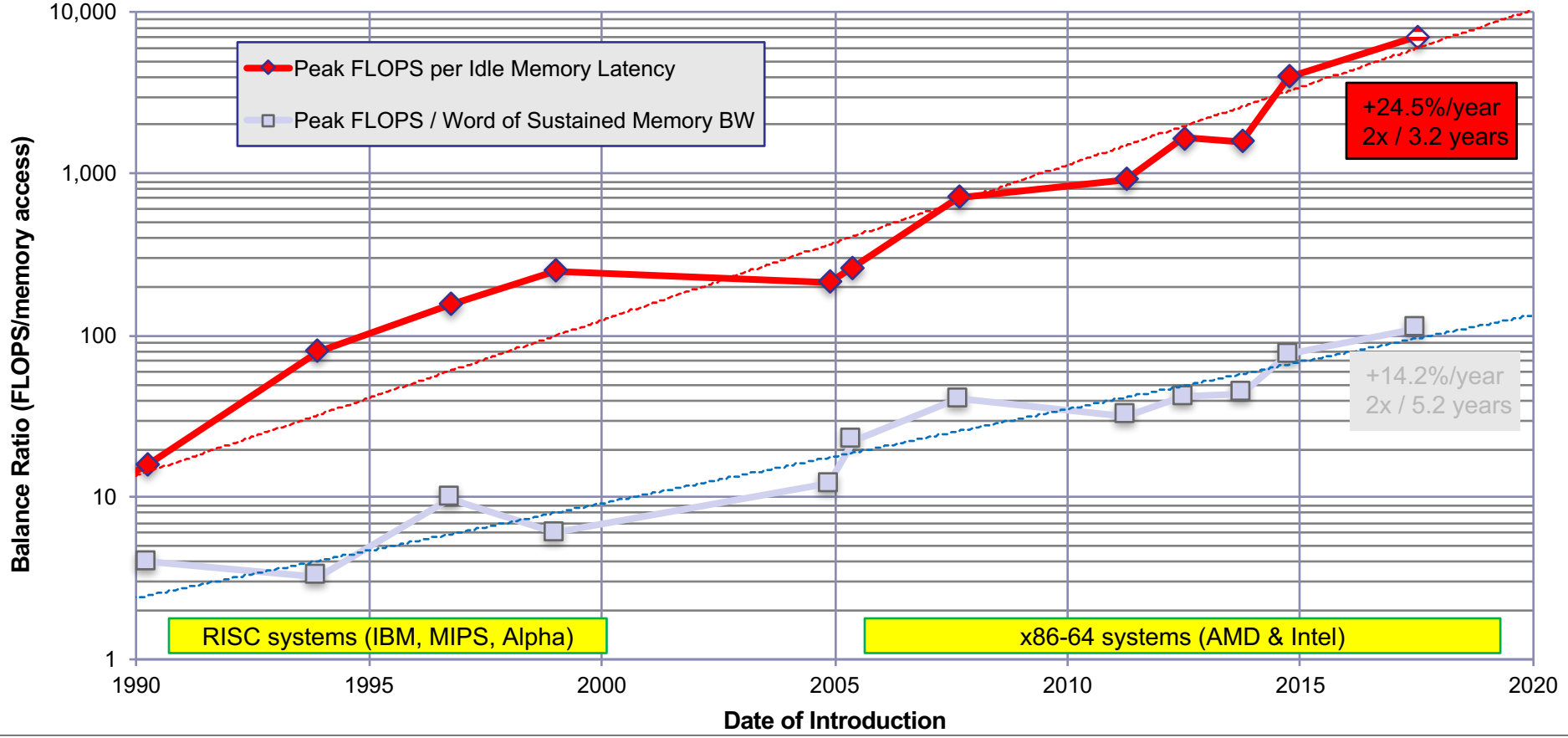
1. Peak FLOPS per socket increasing at **50%-60% per year**
2. Memory Bandwidth increasing at **~23% per year**
3. Memory Latency increasing at **~4% per year**
4. Interconnect Bandwidth increasing at **~20% per year**
5. Interconnect Latency decreasing at **~20% per year**

- *These ratios suggest that processors should be increasingly imbalanced with respect to data motion….*
  - *Today's talk focuses on (1), (2), and a bit of (3)*

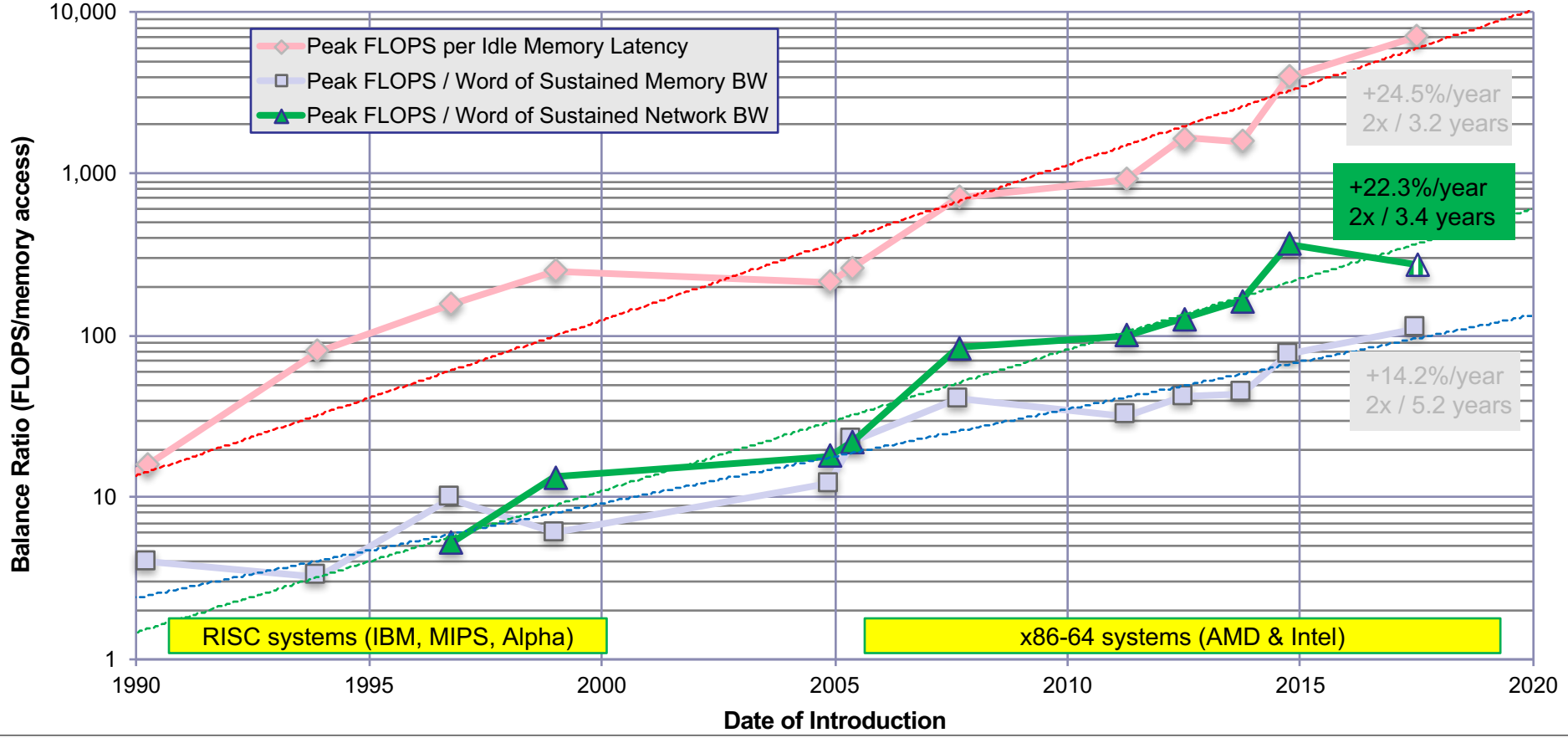
Memory Bandwidth is Falling Behind: (GFLOP/s) / (GWord/s)



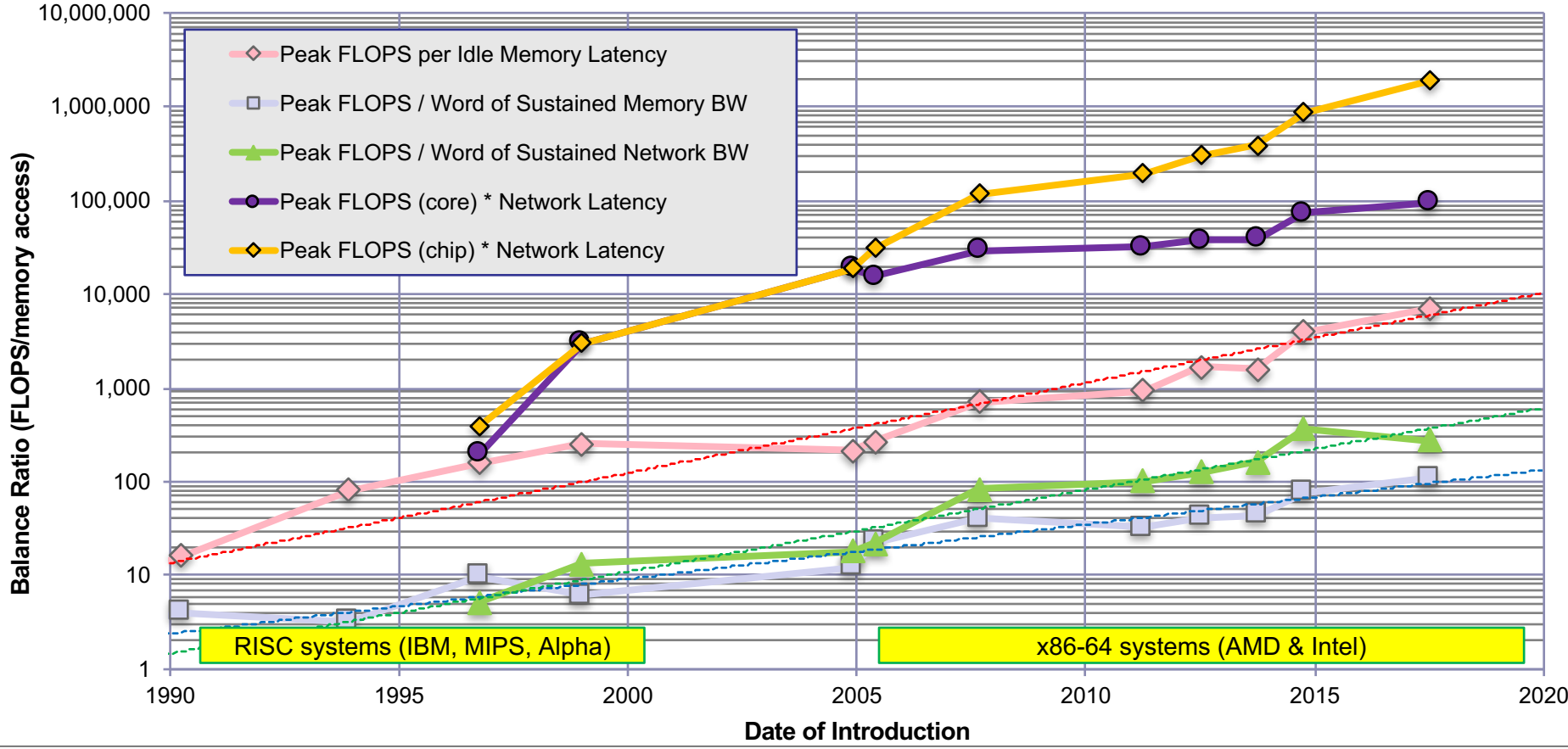
Memory Latency is much worse: (GFLOP/s) / (Memory Latency)



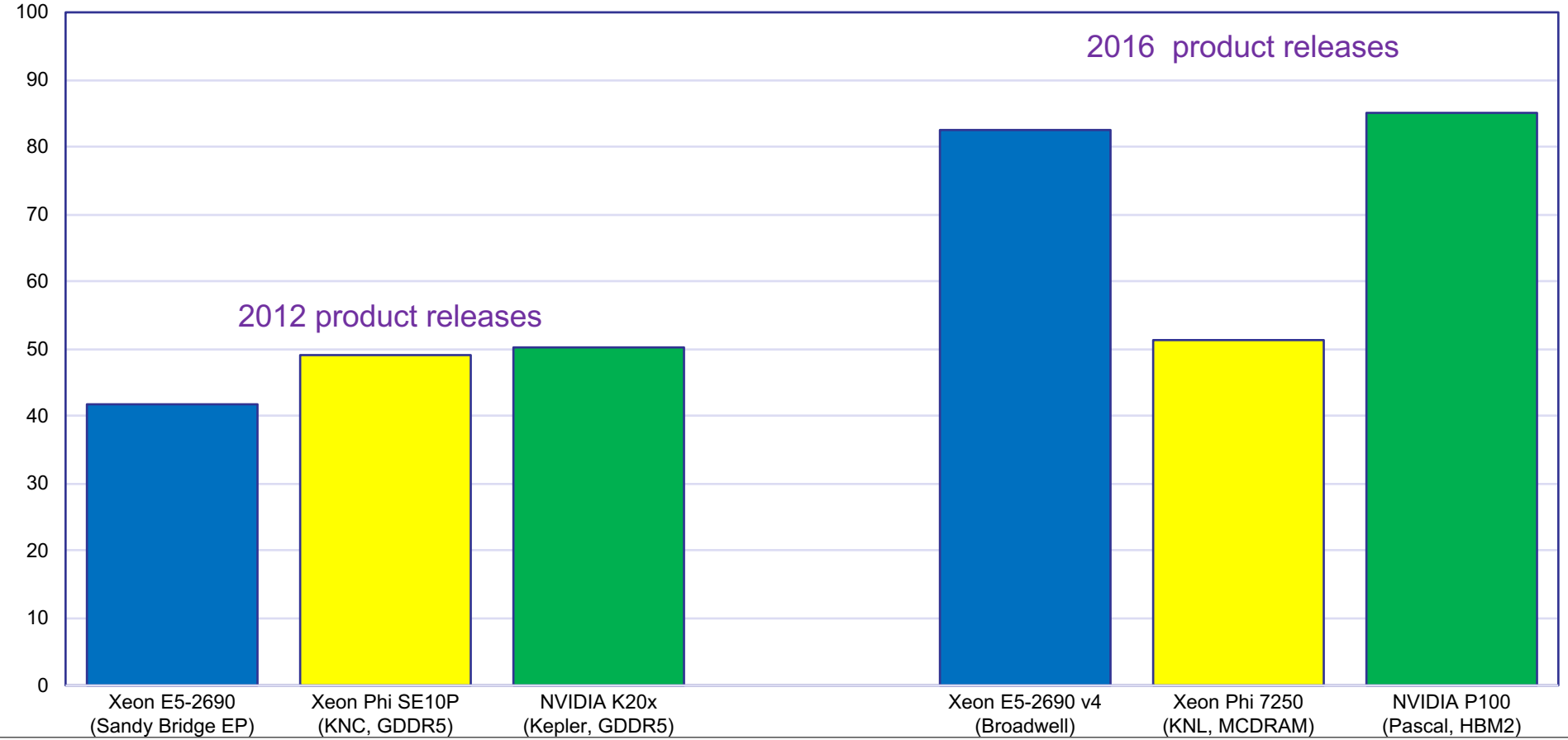
Interconnect Bandwidth is Falling Behind at a comparable rate



Interconnect latency follows a similar trend…

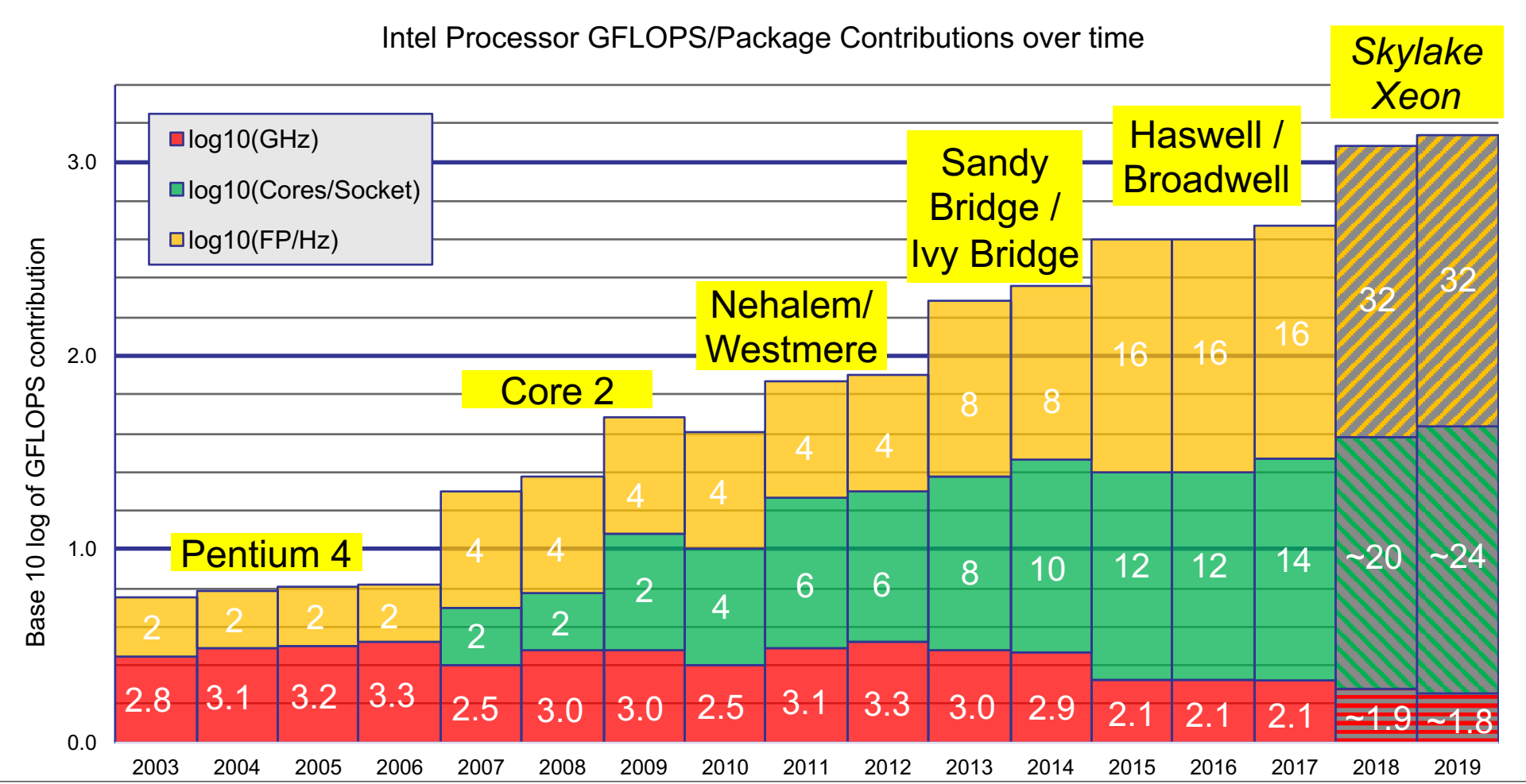


STREAM Balance (FLOPS/Word): Mainstream vs ManyCore vs GPGPU

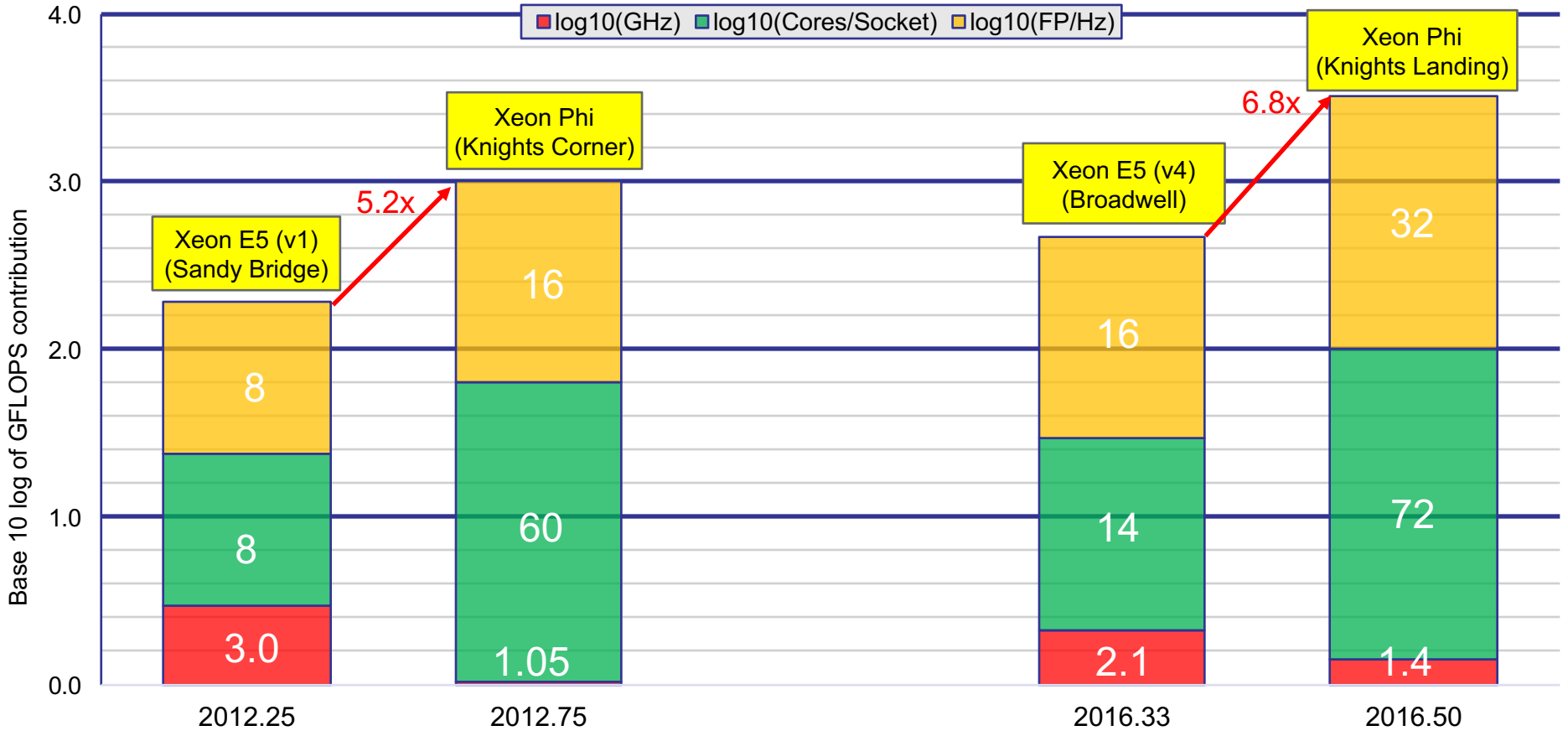


Why are FLOPS increasing so fast?

- Peak FLOPs per package is the product of several terms:
  - Frequency
  - FP operations per cycle per core
    - Product of #FP units, SIMD width of each unit, and complexity of FP instructions (e.g., separate ADD & MUL vs FMA)
  - Number of cores per package
- Low-level semiconductor technology tends to drive these terms at different rates…


Intel Processor GFLOPS/Package Contributions over time
log10(GHz)
log10(Cores/Socket)
log10(FP/Hz)
Base 10 log of GFLOPS contribution
0.0
1.0
2.0
3.0
Pentium 4
Core 2
Nehalem/ Westmere
Sandy Bridge / Ivy Bridge
Haswell / Broadwell
Skylake Xeon
2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
2.8 3.1 3.2 3.3 2.5 3.0 3.0 2.5 3.1 3.3 3.0 2.9 2.1 2.1 2.1 ~1.9 ~1.8
2 2 2 2 2 2 2 4 6 6 8 10 12 12 14 ~20 ~24
4 4 4 4 4 4 8 8 16 16 16 32 32

Intel Processor GFLOPS/Package Contributions: Mainstream vs ManyCore



Why is Memory Bandwidth increasing slowly?

- Slow rate of pin speed improvements
  - Emphasis has been on increasing capacity, not increasing bandwidth
  - Shared-bus architecture (multiple DIMMs per channel) is very hard at high frequencies
- DRAM cell cycle time almost unchanged in 20 years
  - Speed increases require increasing transfer sizes
  - DDR3/DDR4 have minimum 64 Byte transfers in DIMMs
- Slow rate of increase in interface width
  - Pins cost money!

Why is Memory Latency stagnant or growing?

- More levels in cache hierarchy
  - Many lookups serialized to save power
- More asynchronous clock domain crossings
  - Many different clock domains to save power
  - *Snoop (6):* Core -> Ring -> QPI -> Ring -> QPI -> Ring -> Core
  - *Local Memory (4):* Core -> Ring -> DDR -> Ring -> Core
  - *Remote Memory (8):*
    Core -> Ring -> QPI -> Ring -> DDR -> Ring -> QPI -> Ring -> Core
- More cores to keep coherent
  - Challenging even on a single mainstream server chip
  - Two-socket system latency typically dominated by coherence, not data
  - Manycore chips have much higher latency
- Decreasing frequencies!

Why is Interconnect Bandwidth growing slowly?

- Slow rate of pin speed improvements
  - About 20%/year
- Reluctance to increase interface width
  - Switch chips typically pin-limited – wider interfaces get fewer ports
  - Parallel links require more switches – too expensive and does not always provide improved real-world bandwidth

Why is Interconnect Latency improving slowly?

- Legacy IO architecture designed around disks, not communications
  - Control operations using un-cached loads/stores – hundreds of ns per operation and no concurrency
  - Interrupt-driven processing requires many thousands of cycles per transaction
- Mismatch between SW requirements and HW capabilities

A different implication of these technology trends

LATENCY, BANDWIDTH, AND CONCURRENCY

Latency, Bandwidth, and Concurrency

- "Little's Law" from queuing theory describes the relationship between latency (or occupancy), bandwidth, and concurrency.

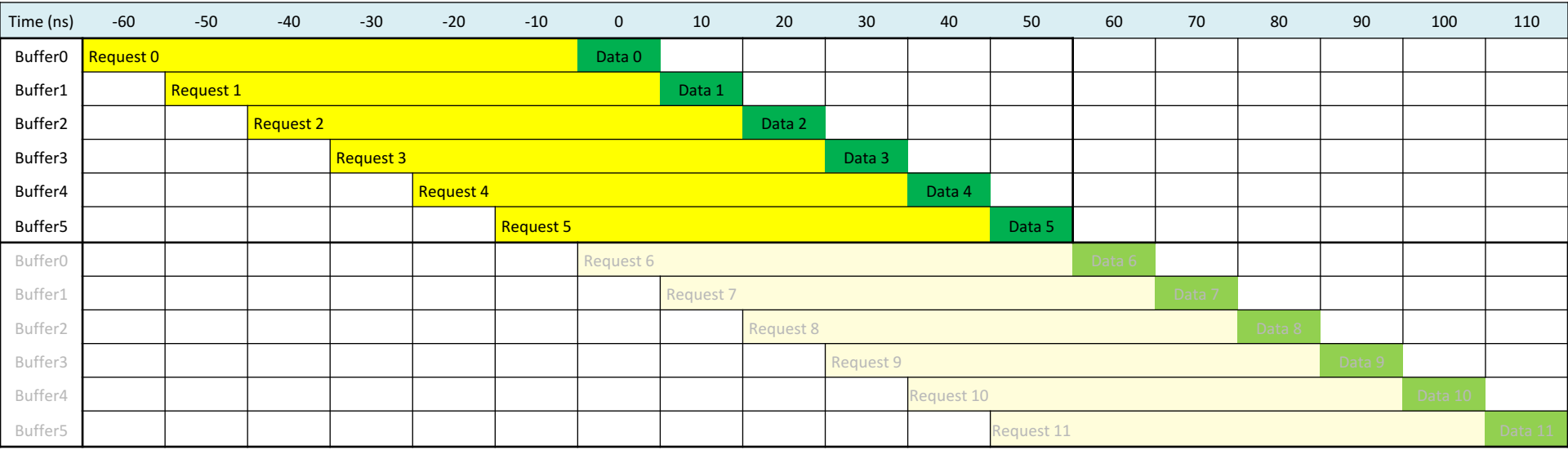
Latency * Bandwidth = Concurrency

- Flat Latency * Increasing Bandwidth → Increasing Concurrency

- Because these are exponential trends, these are not small changes…

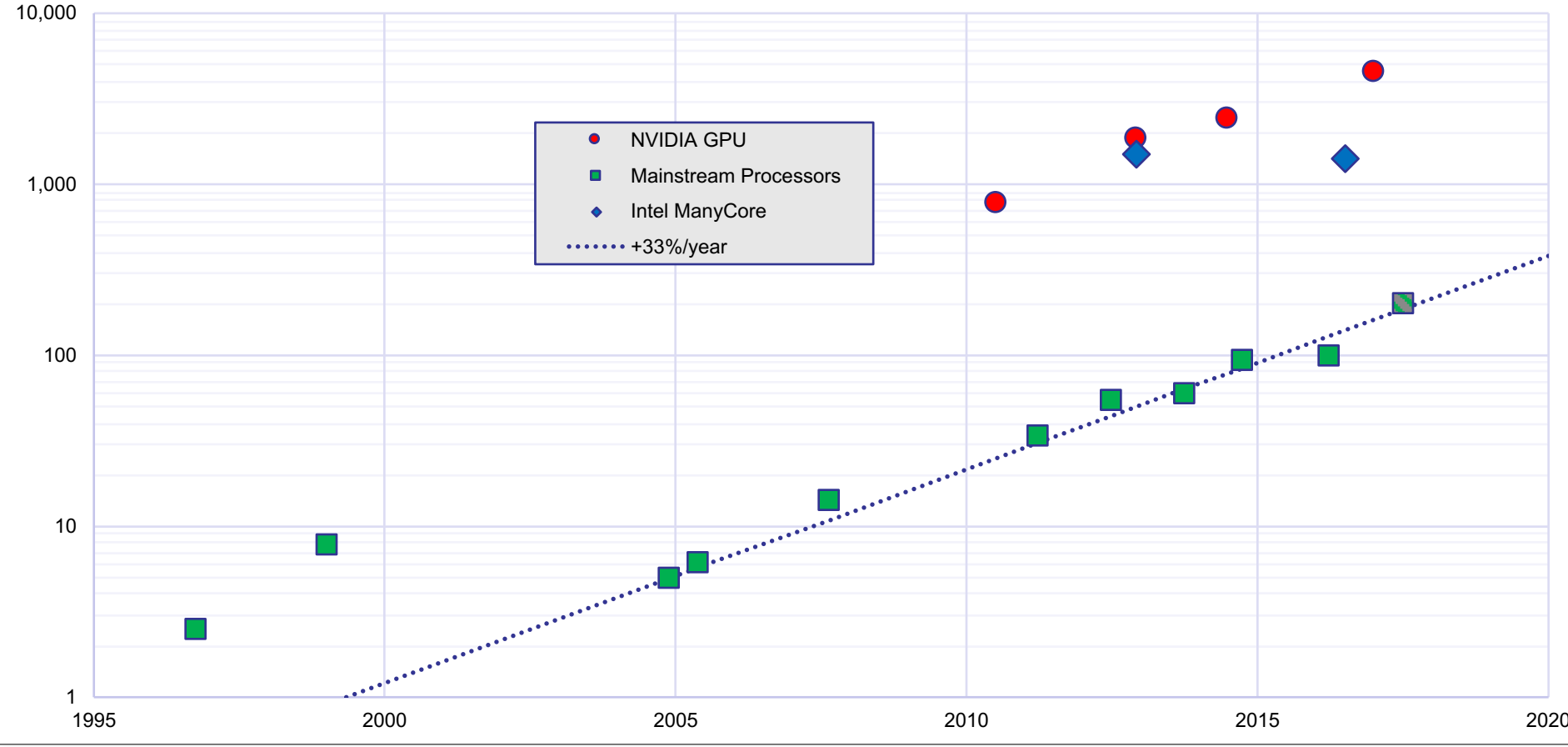
Little's Law: illustration for 2005-era Opteron processor

60 ns latency, 6.4 GB/s (=10ns per 64B cache line)



- 60 ns * 6.4 GB/s = 384 Bytes = 6 cache lines
- To keep the pipeline full, there must always be 6 cache lines "in flight"
- Each request must be launched at least 60 ns before the data is needed

Latency-Bandwidth Products per Package (64B units)



Why is Increasing Concurrency a Problem?

- Architectures are built assuming “flat” memory model
  - Location of data is invisible and uncontrollable
  - Caches and prefetchers are assumed to be “good enough” to cover latency and bandwidth differences
- Implementations support limited L1 Data Cache misses per core:
  - Xeon E5: 10 L1 misses (maximum)
  - L2 Hardware Prefetchers help, but are also “invisible” and not directly controllable

Increasing Concurrency (2)

- Many cores are needed just to generate concurrency, even if not needed to do computing
  - This costs a lot of energy in the cores!
- Large buffers and complex memory controllers are needed to handle the concurrent operations
  - DRAM page management requires memory schedule to be updated frequently as new transactions appear
  - DRAM open page hit rates still go down, so DRAM power increases too
  - Design cost up, power cost up, BW utilization down

Increasing Concurrency (3)

- More cores create more concurrent memory access streams, which requires more DRAM banks
- Examples:
  - 8-core Xeon E5 v1 with 2 streams per core needs >= 16 banks
    Requires 2 ranks of DDR3 DRAM (one dual-rank DIMM)
  - 12-core Xeon E5 v3 with 2 streams per core needs >= 24 banks
    Requires 2 ranks of DDR4 DRAM (one dual-rank DIMM)
- Problems:
  - Some codes generate many address streams per core – LBM >32
  - HyperThreading can double address streams per core
  - Adding more DIMMs can *decrease* performance due to rank-to-rank bus stalls

Another angle…

POWER AND ENERGY

What about Power/Energy?

- Power density is important in processor implementations
  - Frequencies can be limited by small-scale (core-sized) hot spots
  - Multi-core frequencies are now limited by package cooling
  - E.g., Xeon E5 v3 (Haswell) can only run DGEMM or LINPACK on ½ of the cores before running out of power & needing to throttle frequency
- Power is not a first-order concern in operating cost!!!
  - Purchase price is $2500-$4000/socket
  - Socket draws 100-150 Watts & needs 40-50 Watts for cooling
  - At $0.10/kWh, this is 5%-7% of purchase price per year
  - This ratio is very hard to change!!!

What about Power/Energy later?

- If much cheaper processors become available, power would become a first-order cost
- Example 1: “client” multicore processors
  - Use the same core architecture, but at much lower price
  - Typical configuration needs 25% of purchase cost per year for power
  - (High performance interconnect solution not available at reasonable price)
- Example 2: “embedded” processors
  - Hypothetical $5 processor using 5 Watts requires $7/year for power
  - Not a problem for mobile – not credible for HPC
  - Response will be sociological and bureaucratic, as well as technical

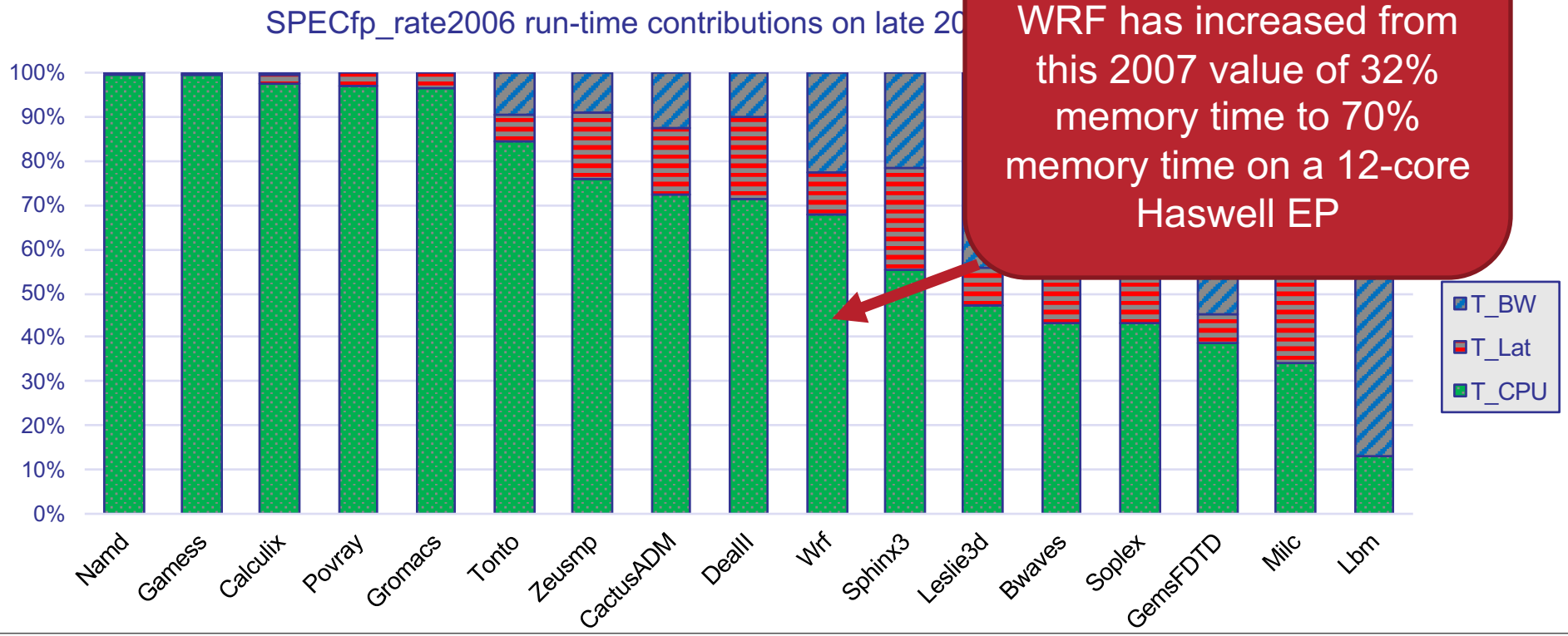
Part 3

COMMENTS & SPECULATIONS

Implications for HPC Systems?

- Applications have very different requirements for various performance components….
  - It is relatively easy to find 100:1 ratios in memory bandwidth requirements across applications
  - Ratios in other axes are mostly smaller due to self-selection, but 100:1 examples certainly exist
    - E.g., bisection bandwidth for 3D DNS Turbulence

- Example: CPU + Memory BW + Memory Latency model from 2007 – most of these applications are



A note on “optimization”

- Under fairly general assumptions, it can be easily proven that

A homogeneous system cannot be “optimal” for a heterogeneous workload!

  - “Optimal” here can refer to performance, power, overall cost.

- “Optimized for General-Purpose Workloads” is a contradiction
  - As systems get larger and the ratios of performance requirements increase, the advantage of specialization increases

Implications for HPC Systems?

- System Balance trends are due to a combination of technology factors and market factors
  - Technology trends suggest that balances will either get worse, or performance will stagnate
- Some applications will be able to exploit the increasing capabilities, while others will not
  - Changing algorithms may be required to obtain continuing performance increases, e.g.
    - Increasing computational intensity, increasing locality

Implications for HPC Applications?

- As systems scale, the number of performance axes that need to be considered will continue to increase
- Effective use of new & future hardware will require effective exploitation of
  - Node-level parallelism (e.g., MPI)
  - Multi-core parallelism (e.g., OpenMP)
  - SIMD Vector parallelism
    - Multiple independent SIMD vector ops to tolerate pipeline latency
  - Unit-stride memory access with increasing data re-use

John D. McCalpin, PhD

mccalpin@tacc.utexas.edu

512-232-3754

For more information:

www.tacc.utexas.edu

Disruptive Technology Ideas (1)

- Quit fighting the physics of data motion
- Move simple cores to the data
  - Distances are shorter – less energy for data transfer
  - Design is simpler – less development money to recover
  - Use more efficient point-to-point DRAM interfaces
    - GDDR5/6, LPDDR4/5
  - Core Performance requirements are lower
    - Lower frequency to exploit $P \sim f^3$
- Save high-power processors for complex processing

Disruptive Technology Ideas (2)

- Move out of the 1980's – memory is not flat!
- The architecture must include functionality to enable control over the most "expensive" operations
  - This must include data motion through a memory hierarchy as a first-order architectural concept
  - Provide semantic information to outer levels of the hierarchy to enable efficient and performant scheduling of operations on multiple data streams of differing priorities and consumption rates.

Disruptive Technology Ideas (3)

- Quit fighting physics – cache coherence
  - Limit coherence to small areas in physical space and small areas in address space that can be tightly constrained
  - Don’t use cache coherence for communication!
    - Data motion through the memory hierarchy is fundamentally different than communication or synchronization
    - Optimal behaviors are different, so these must be exposed as different operation types

Disruptive Technology Ideas (4)

- Admit that computers are parallel!
- Current architectures don't include communication or synchronization as concepts
  - Communication and Synchronization can be implemented as side effects of sequences of ordered memory reference
  - Indirect, inefficient, ugly
- Hardware is capable of extremely efficient communication and synchronization

Disruptive Technology Ideas (5)

- Design for Performance Predictability
- Programming languages that describe algorithms at higher levels of abstraction require significant transformations to map to hardware
  - Currently very limited because performance cannot be modeled!
- New HW architectures will require human cost for SW re-development but we a paying a cost now for incomprehensibly complex systems

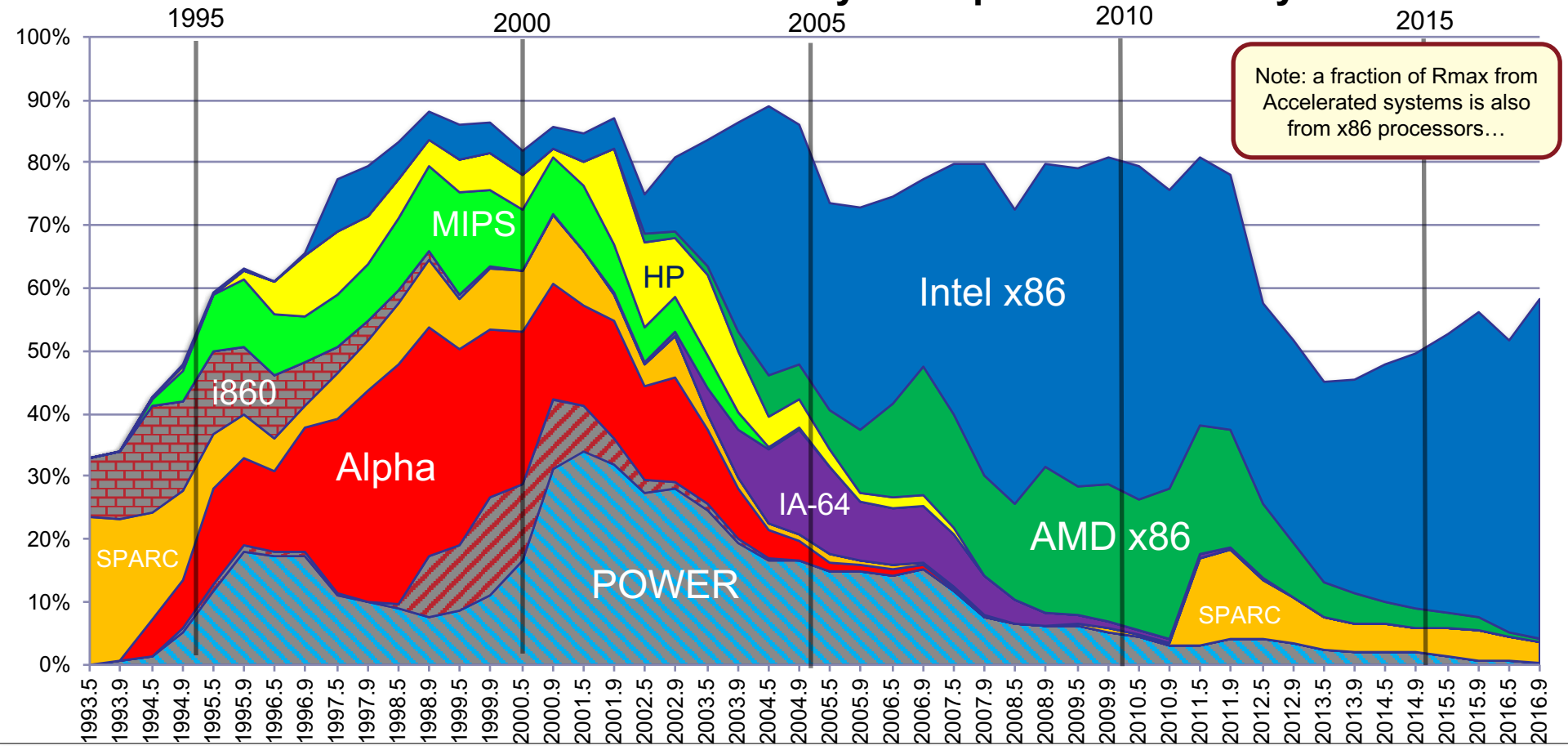
Summary

- Technology trends suggest that data access is increasingly expensive compared to computation
  - Benchmark data from 25 years confirms these trends
- Multi-level caching and aggressive prefetching have mitigated the impact of the imbalance, but
  - These are expensive to design
  - These are not energy-efficient
- Architectural changes are needed to address design cost and energy efficiency

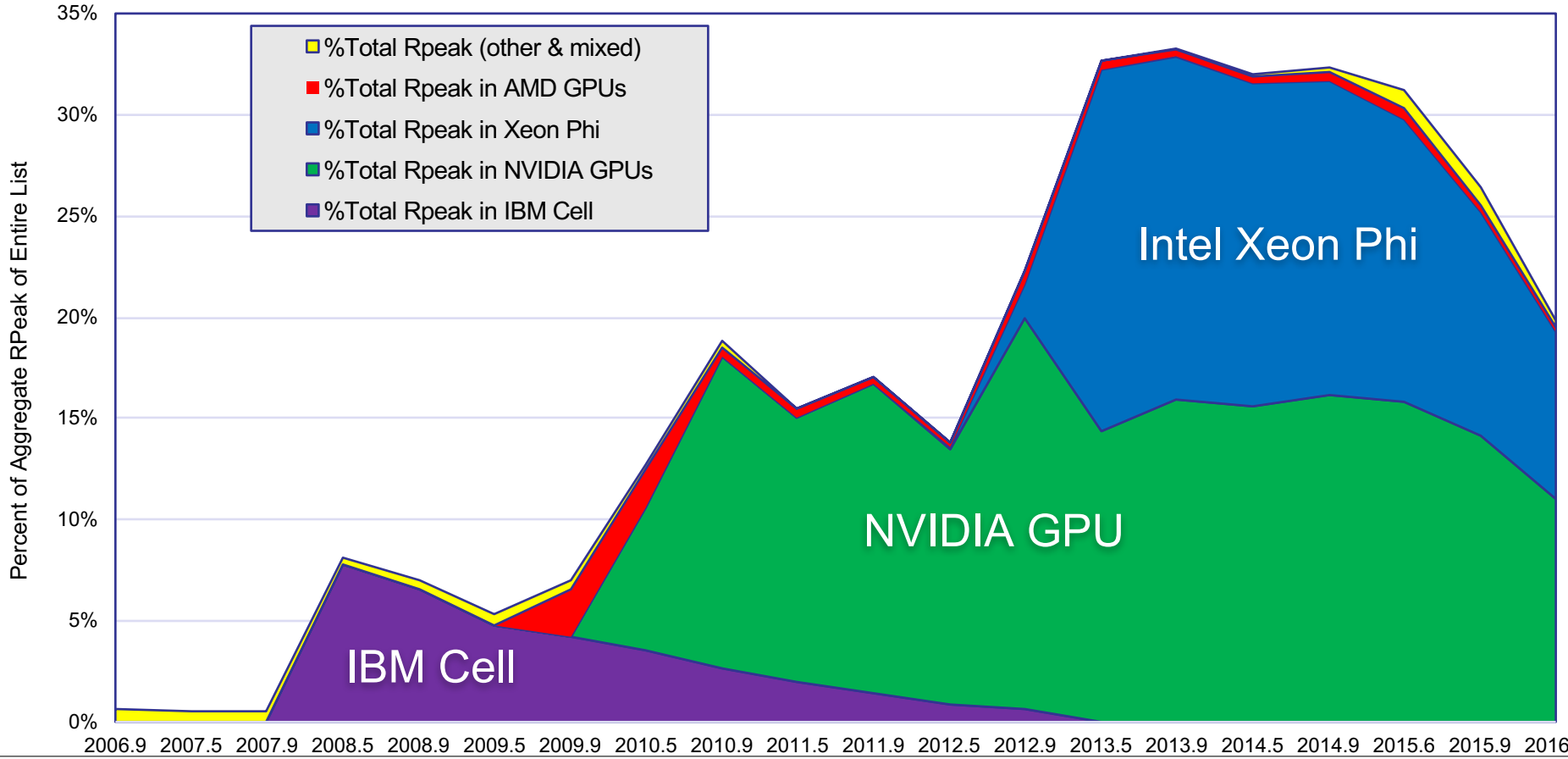
For the insomniacs among us…

BACKUP SLIDES

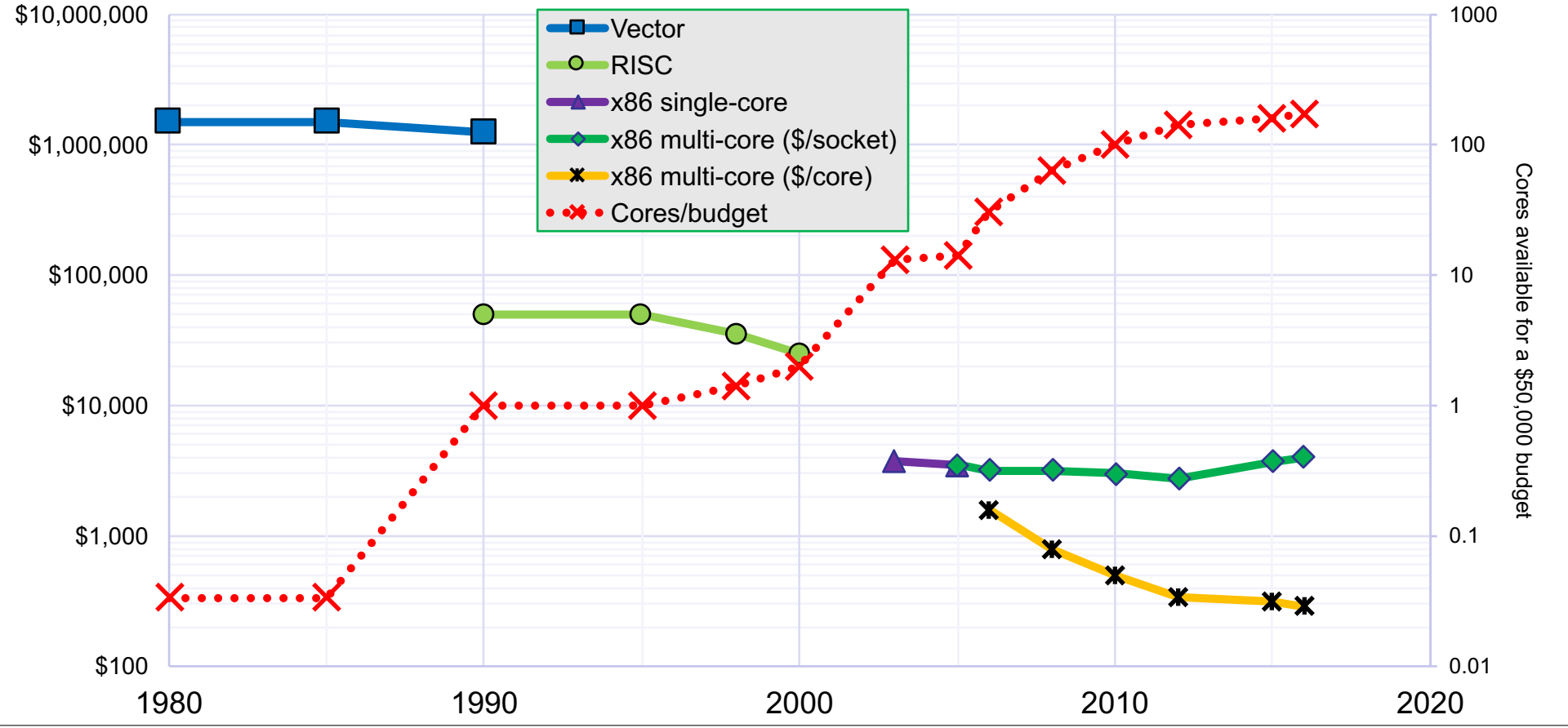
TOP500 Rmax Contributions by Microprocessor Family



Contributions of Various Accelerator Families to TOP500 RPeak



Decreasing Price/Core Leads to Increasing Available Core Counts



Why?

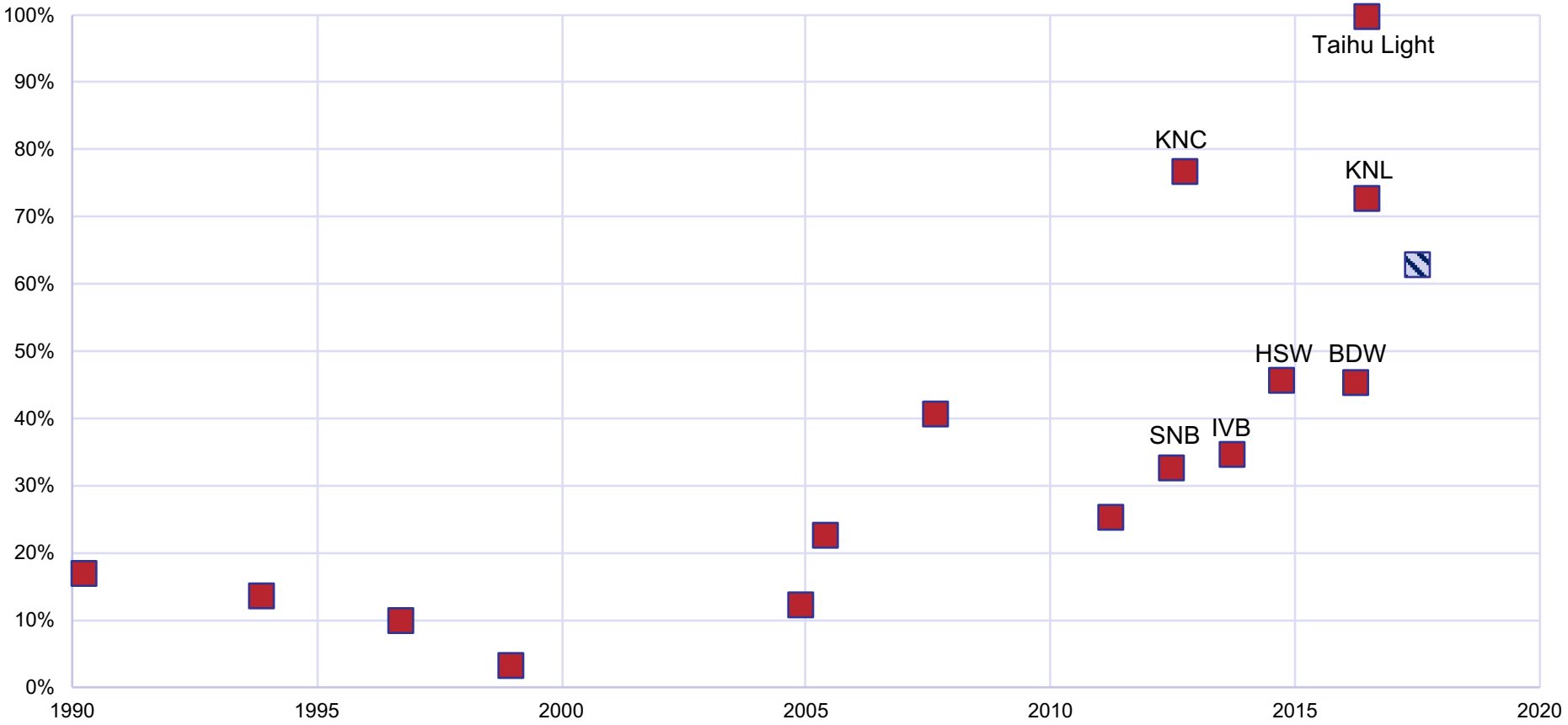
- FLOPS
  - More cores
  - More FLOPS/Cycle
- Memory Bandwidth
  - Slow rate of pin speed improvements
    - DRAM cell cycle time almost unchanged in 20 years
    - Speed increases require increasing transfer sizes
  - Slow rate of increase in interface width
    - Pins cost money!
- Memory Latency
  - More levels in of cache
  - More asynchronous clock domain crossings
  - More cores to keep coherent
- Interconnect Bandwidth
  - Slow rate of pin speed improvements
  - Slow rate of increase in interface width
- Interconnect Latency
  - Legacy IO architecture designed around disks, not communications
  - Mismatch between SW requirements and HW capabilities

SPEC CPU 2006 codes

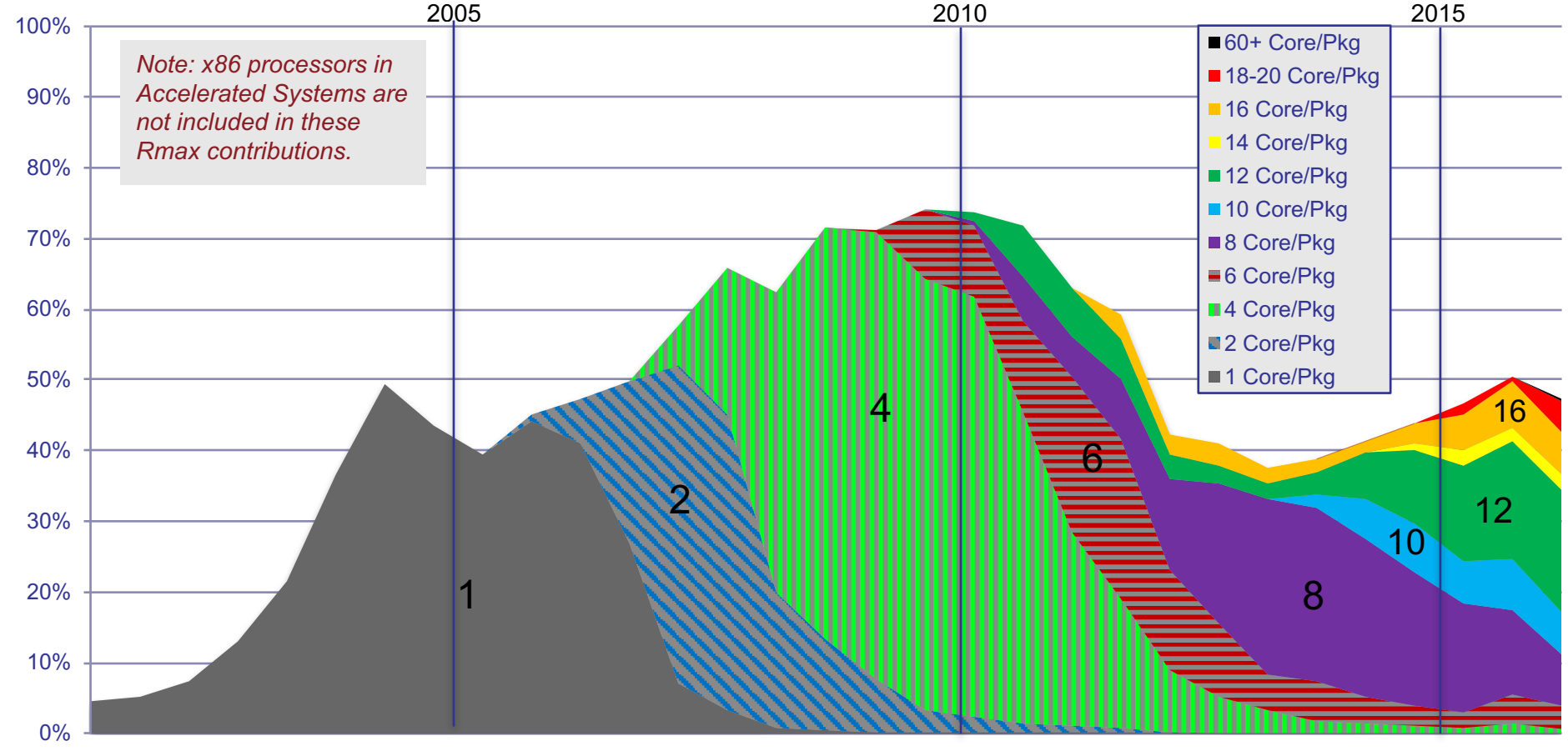
In decreasing order of computational intensity on reference system:

• NAMD	molecular dynamics
• GAMESS	quantum chemistry
• CalculiX	nonlinear structure finite element
• POV-Ray	ray-tracing
• GROMACS	molecular dynamics
• Tonto	quantum crystallography
• ZEUS-MP	CFD/MHD
• CactusADM	General Relativity
• Deal.II	adaptive finite element elliptic equation solver
• WRF	numerical weather prediction (limited area)
• Sphinx-3	speech recognition
• LESlie3d	CFD (finite volume explicit)
• Bwaves	CFD (transonic, implicit Bi-CGstab)
• SoPlex	Simplex Linear Programming solver
• GemsFDTD	computational electromagnetics, finite difference, explicit
• MILC	quantum chromodynamics
• LBM	CFD (Lattice-Boltzmann)

Required Memory BW for DGEMM vs Maximum Sustainable Memory BW



TOP500 x86 System Rmax Contributions by Cores/Package



Strawman: Processor At Memory (PAM)