BY THEIR FRUITS SHALL YE KNOW THEM

A DATA ANALYST'S PERSPECTIVE ON MASSIVELY PARALLEL SYSTEM DESIGN

Holger Pirk Sam Madden Mike Stonebraker

ISTC

BIG DATA

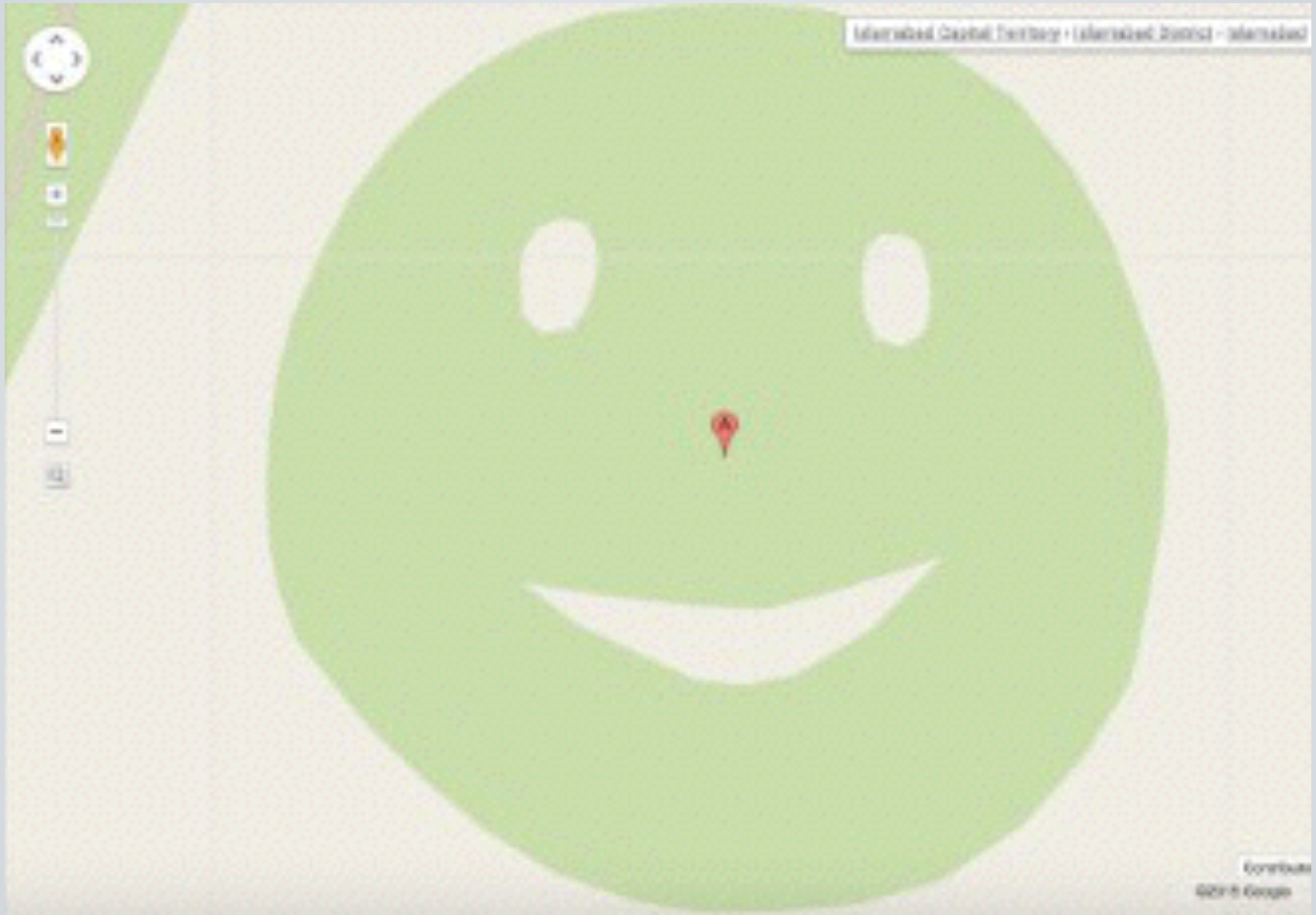
CSAIL

A CRUCIAL DISTINCTION

ISTC BIG DATA $\neq$ intel

CSAIL

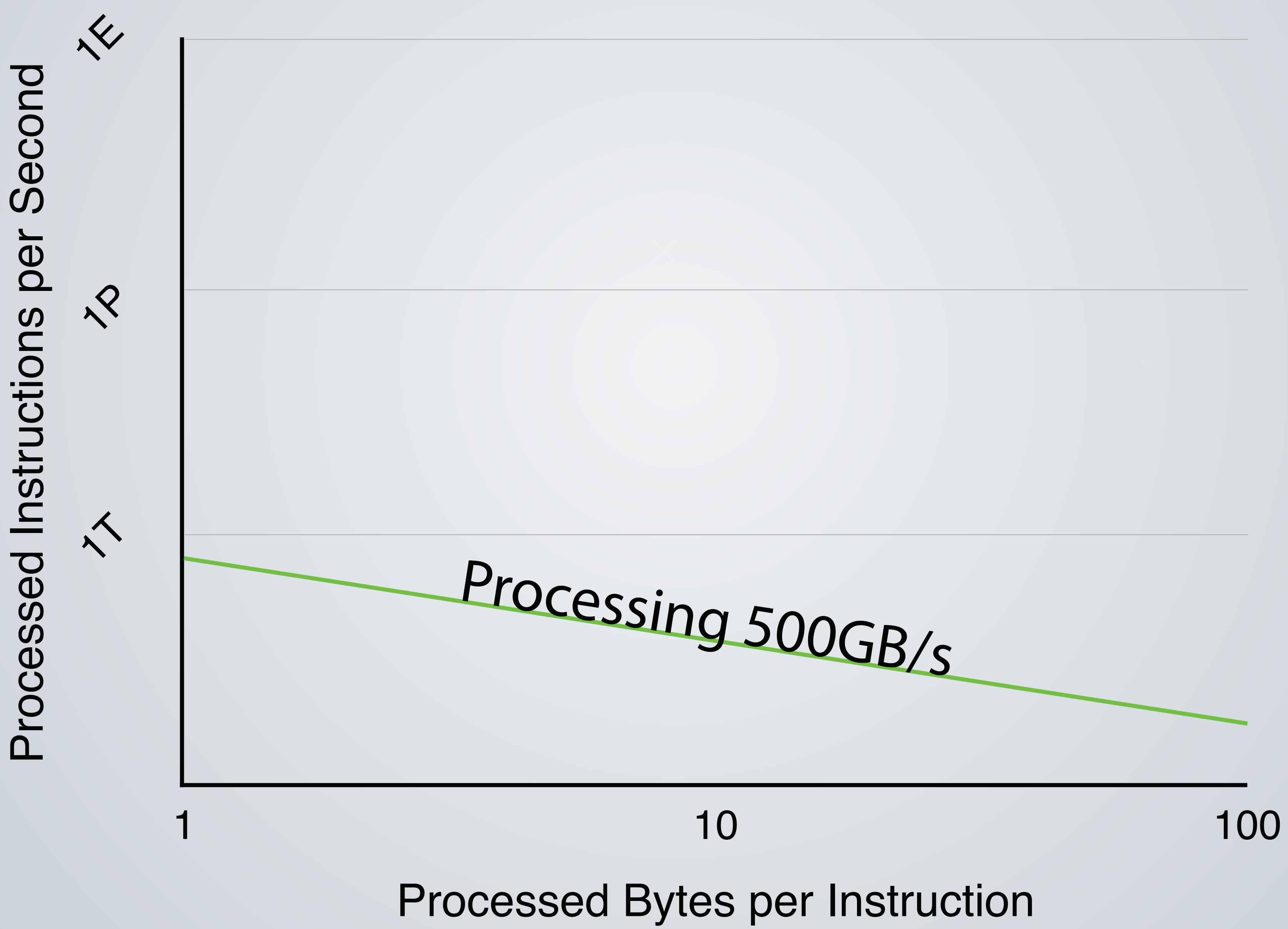
INSPIRATION



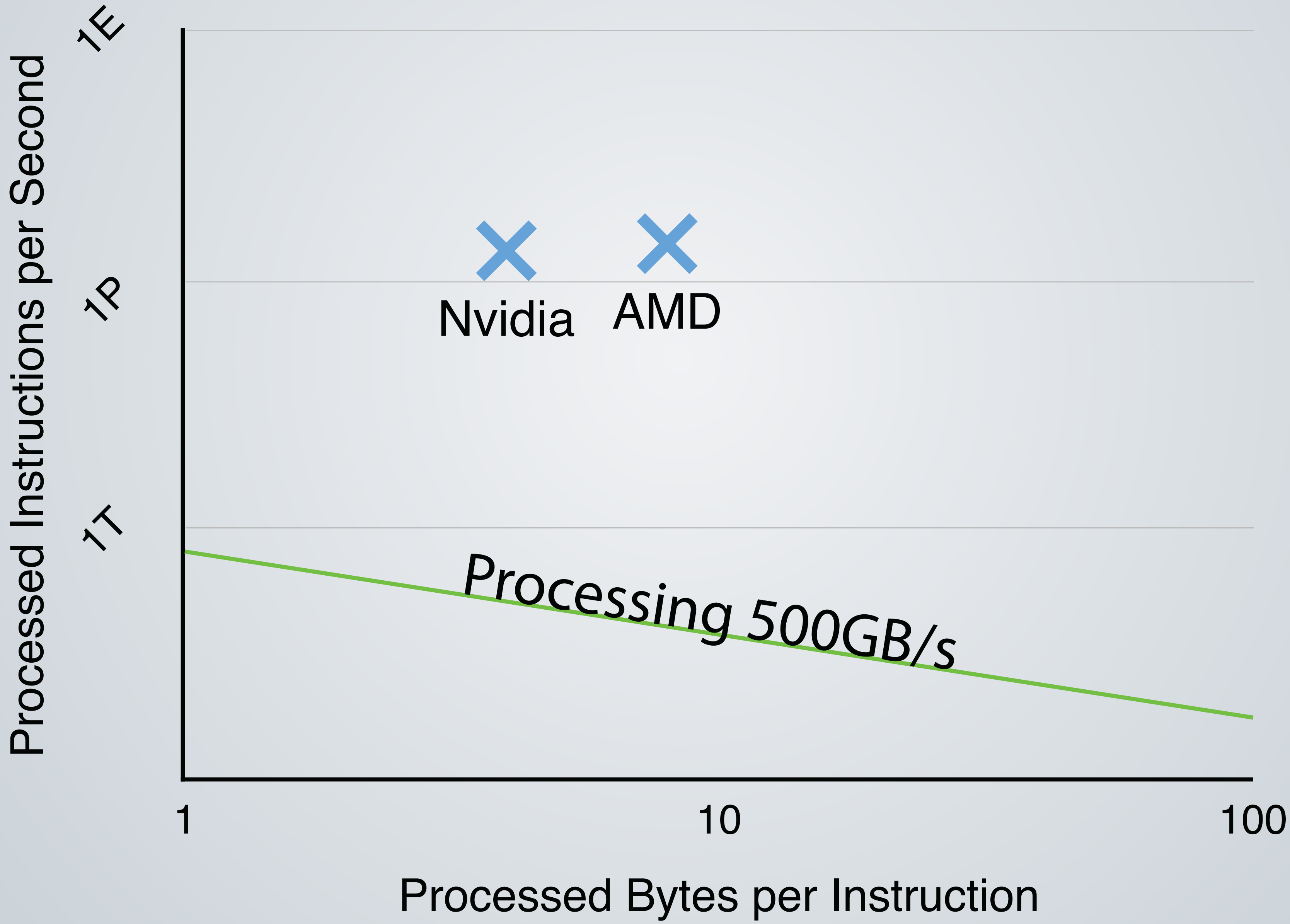
MY PLEDGE OF LOYALTY

SCIENTIFIC RATIONALE

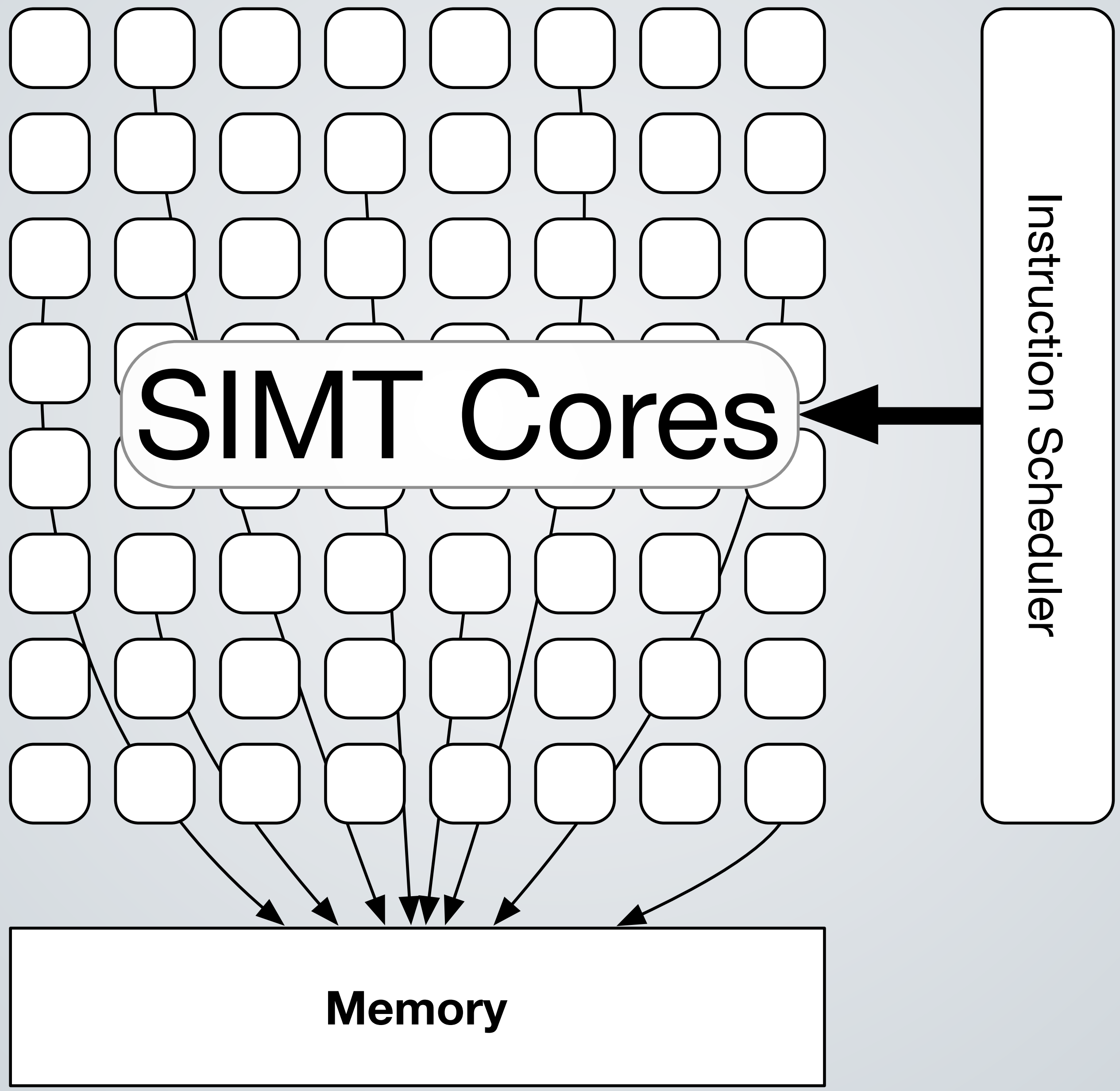
GENE AMDAHL TAUGHT US THAT SYSTEMS NEED TO BE BALANCED



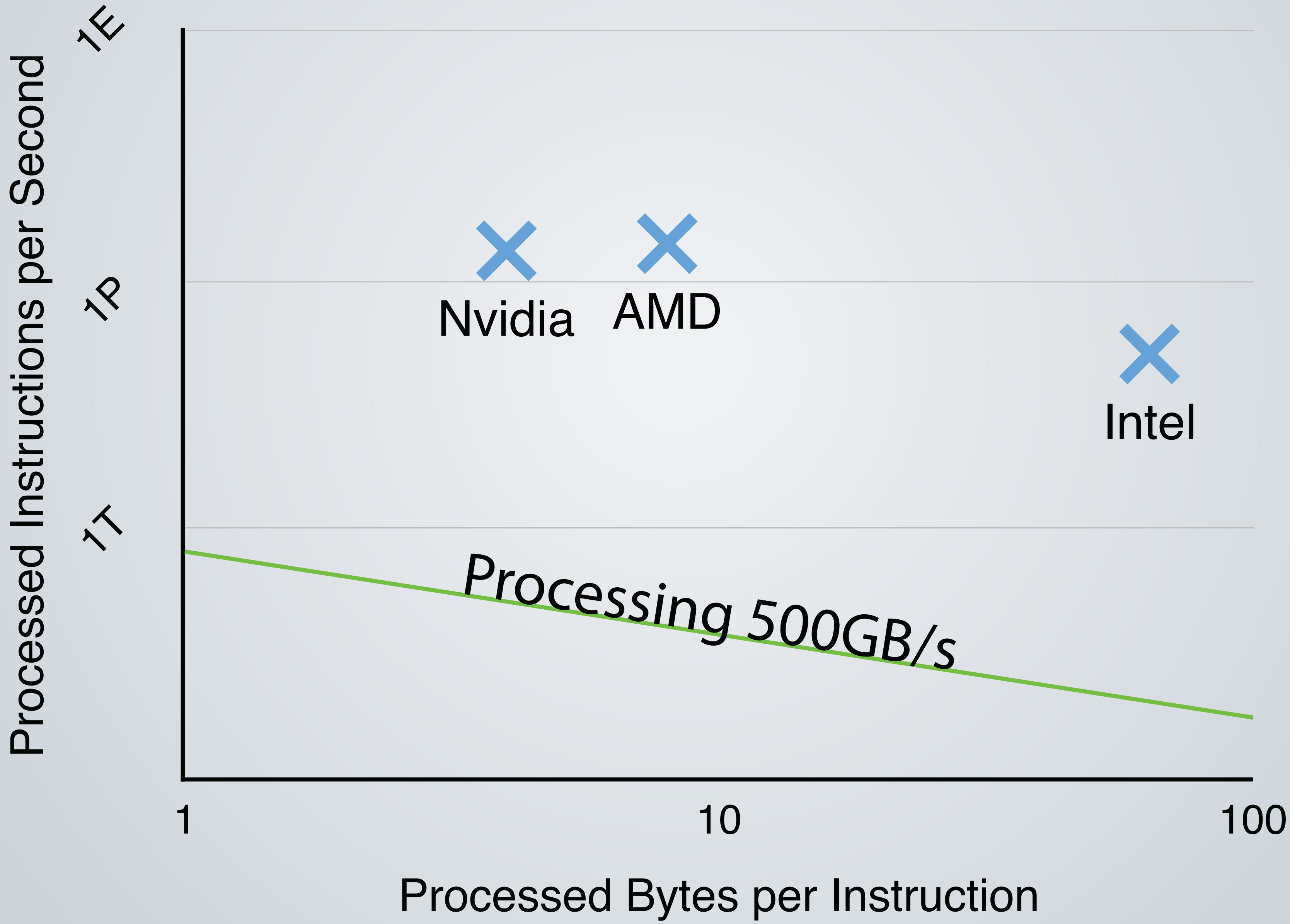
NVIDIA AND AMD PROCESS LOT OF SMALL DATA WORDS



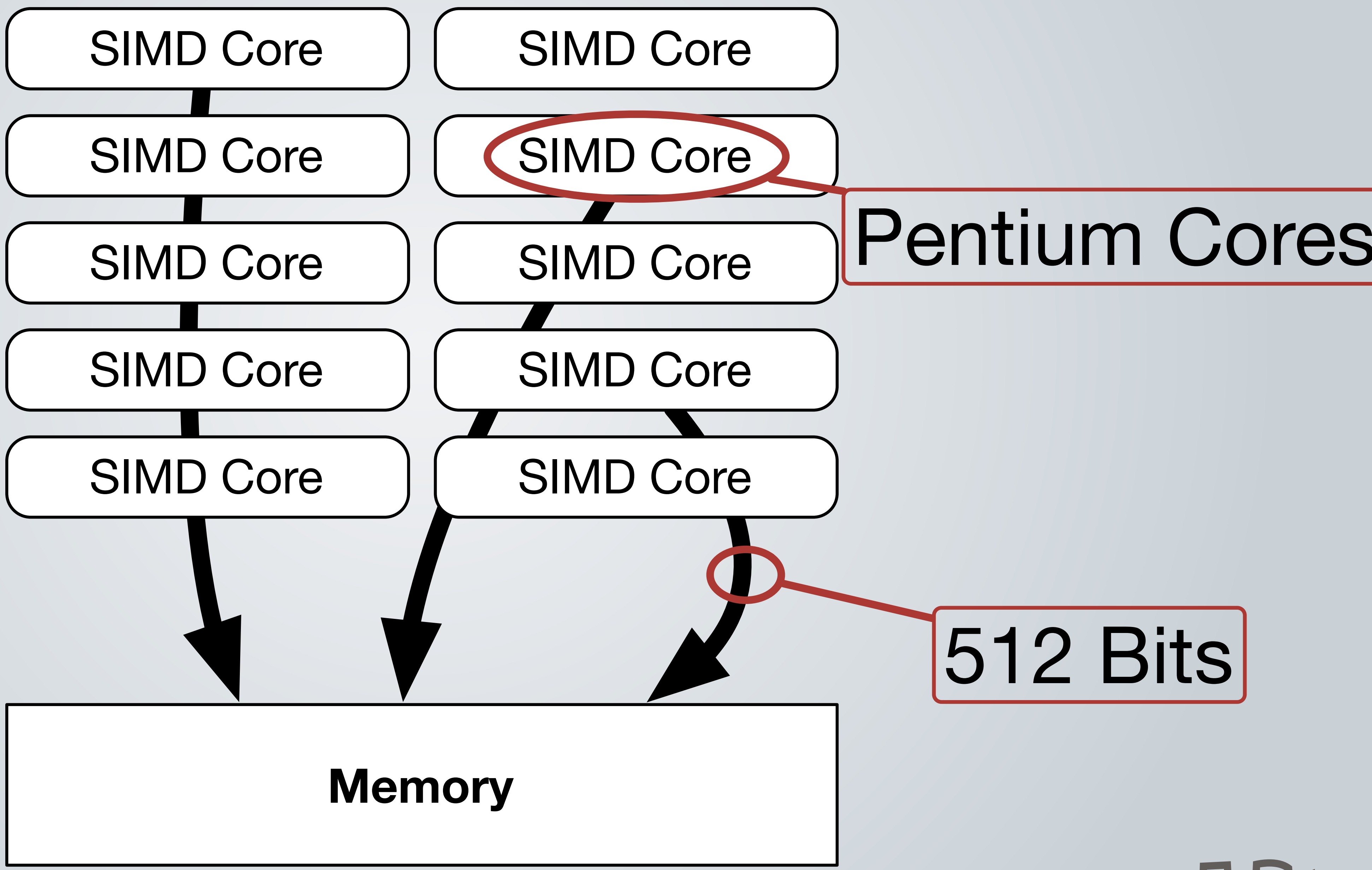
SIMT



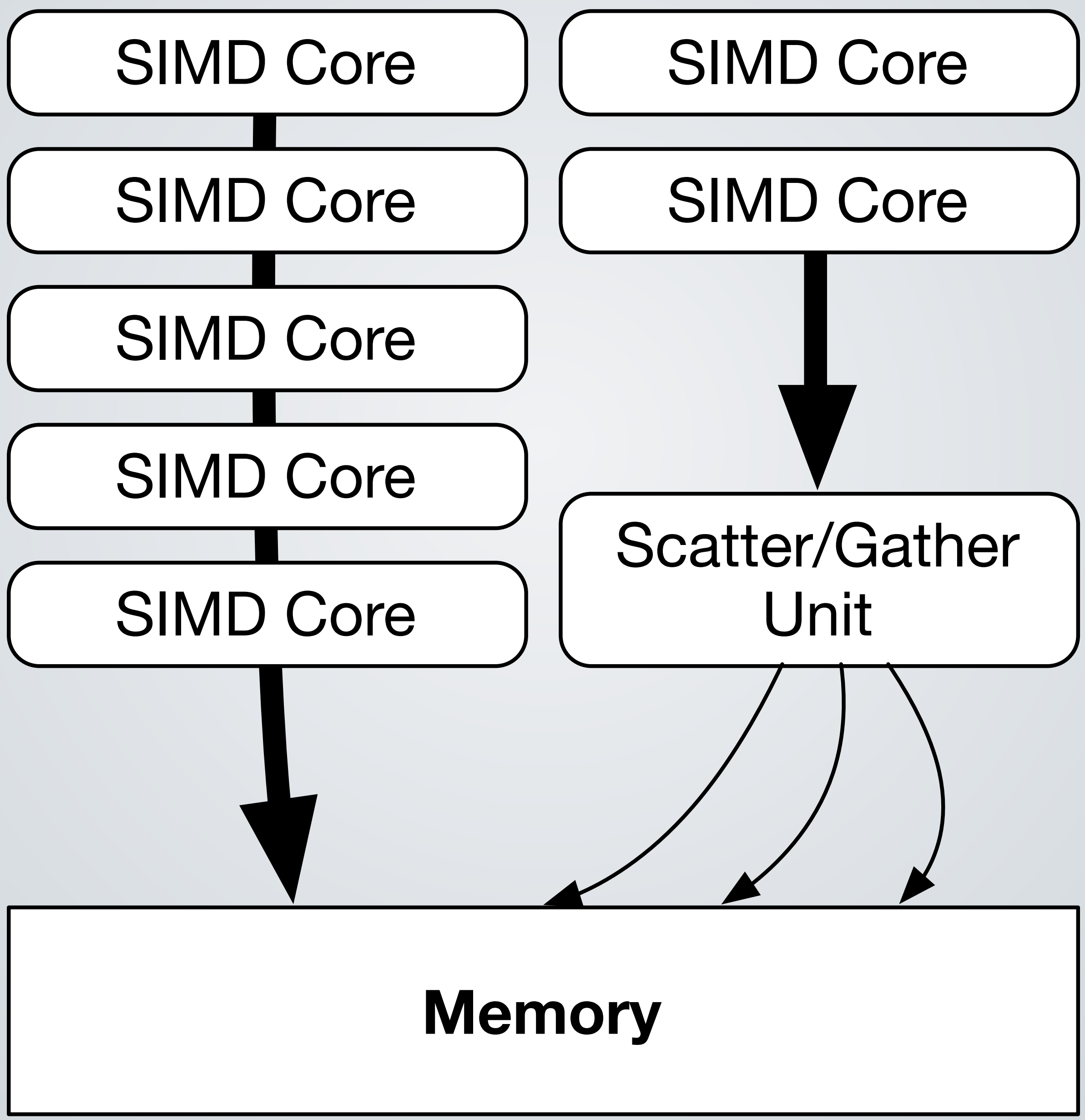
INTEL PROCESSES FEWER LARGE DATAWORDS



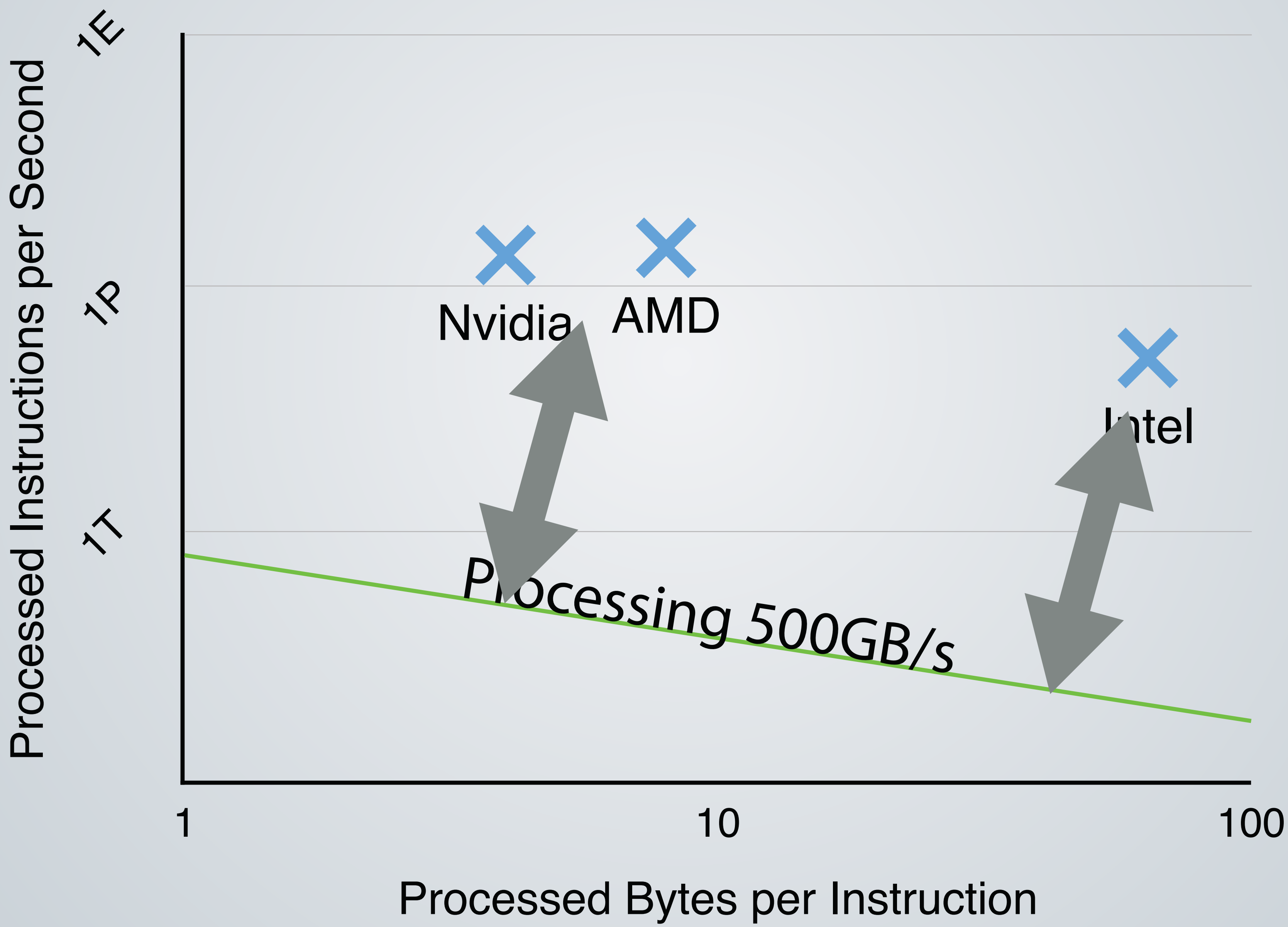
MANY-CORE SIMD



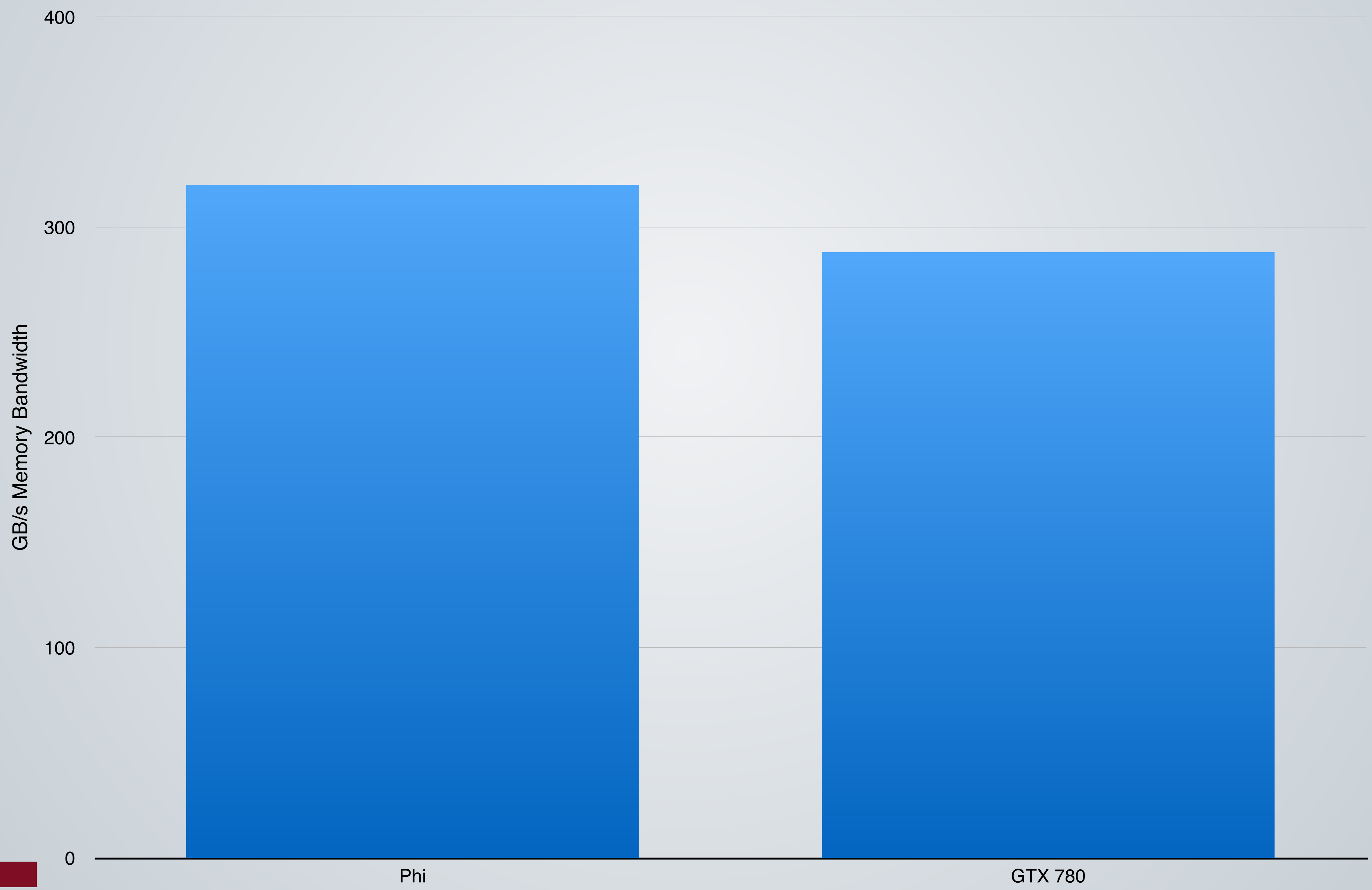
SIMD WITH SCATTER/GATHER



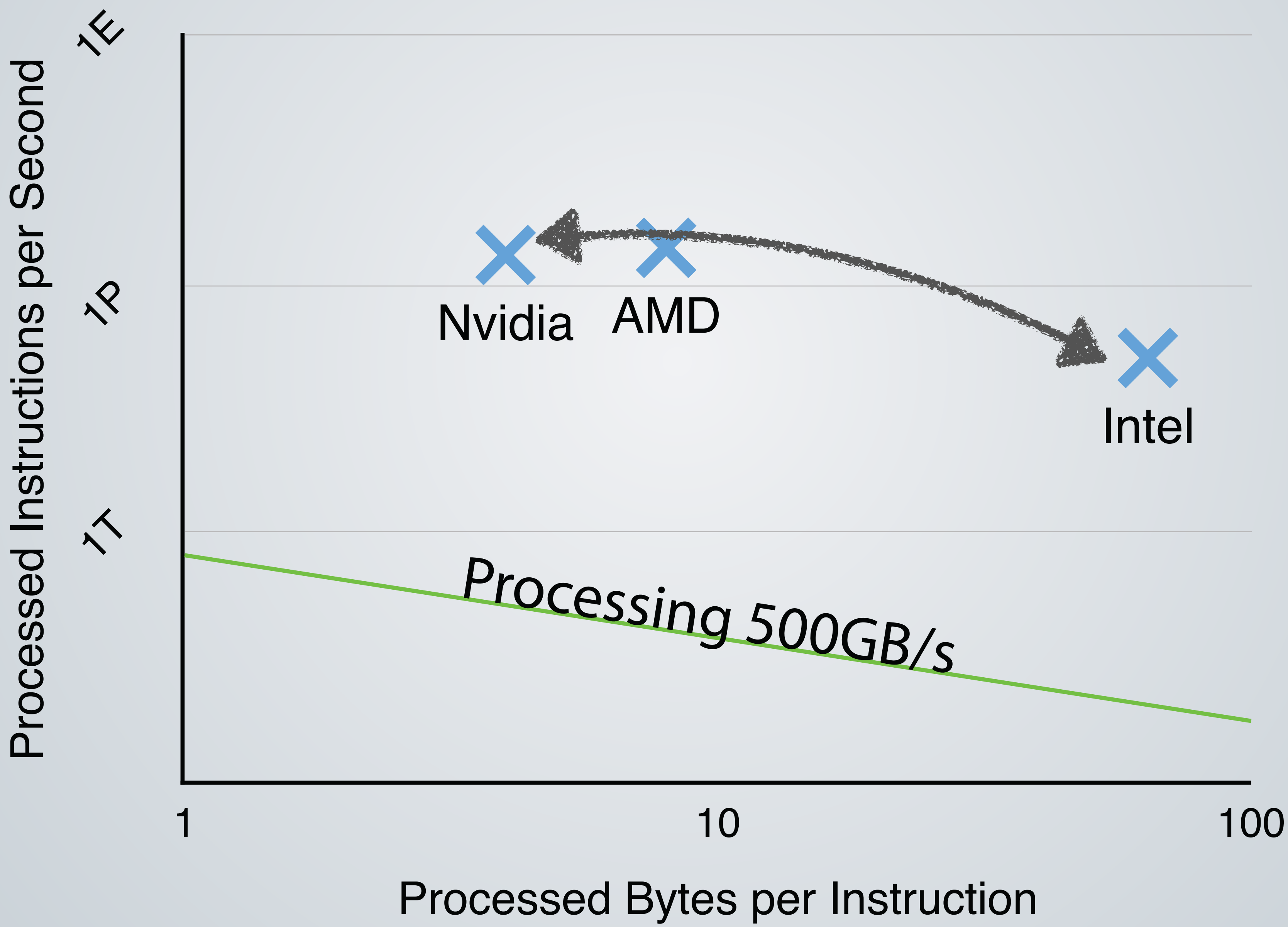
ALL OF THEM CAN PROCESS WAY MORE DATA THAN THEY CAN LOAD



SPEC BANDWIDTH-WISE, PHI OUTPERFORMS CURRENT GPUS

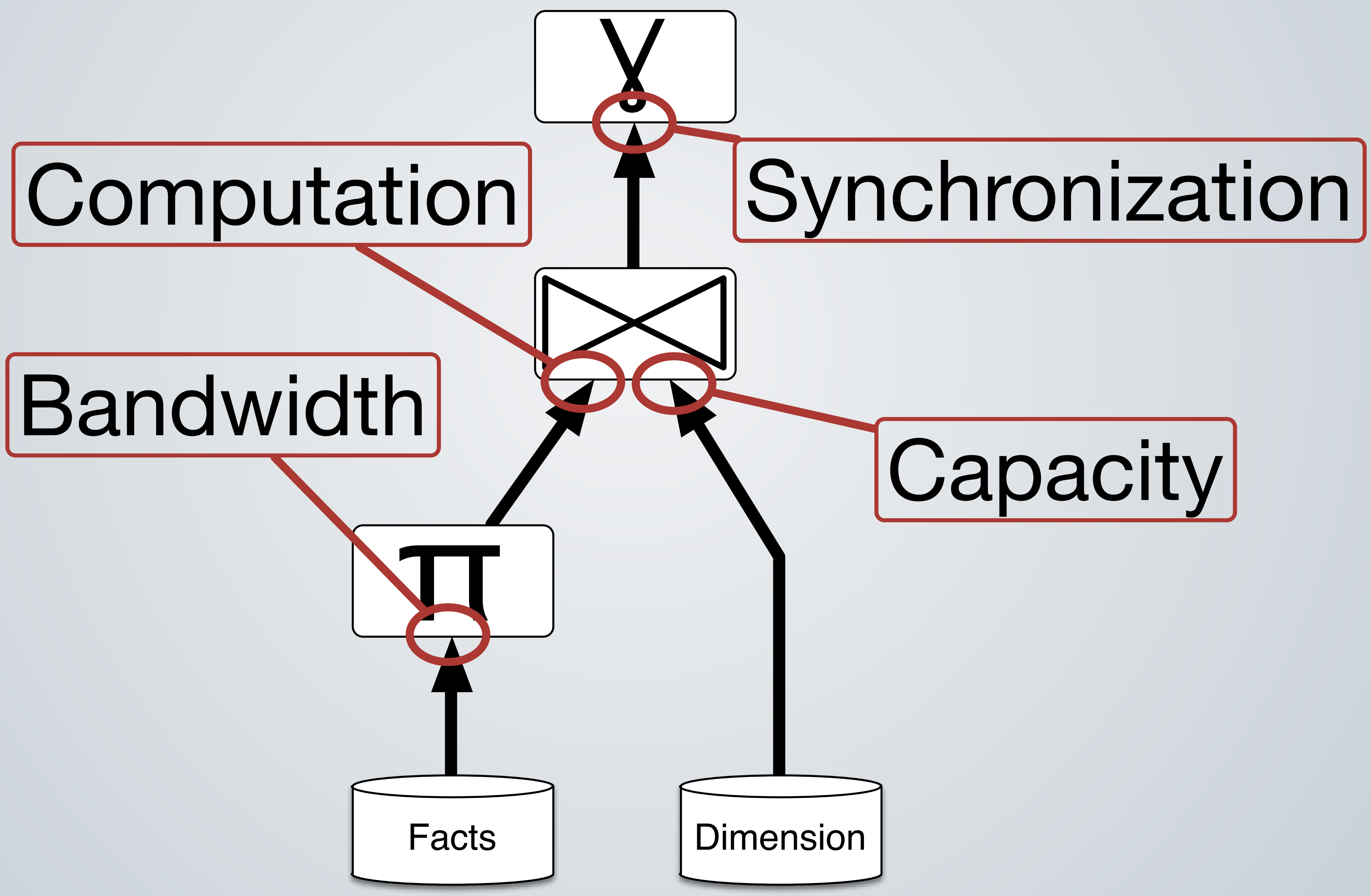


OUR QUESTION: DOES IT MATTER? DOES PHI CHANGE ANYTHING?

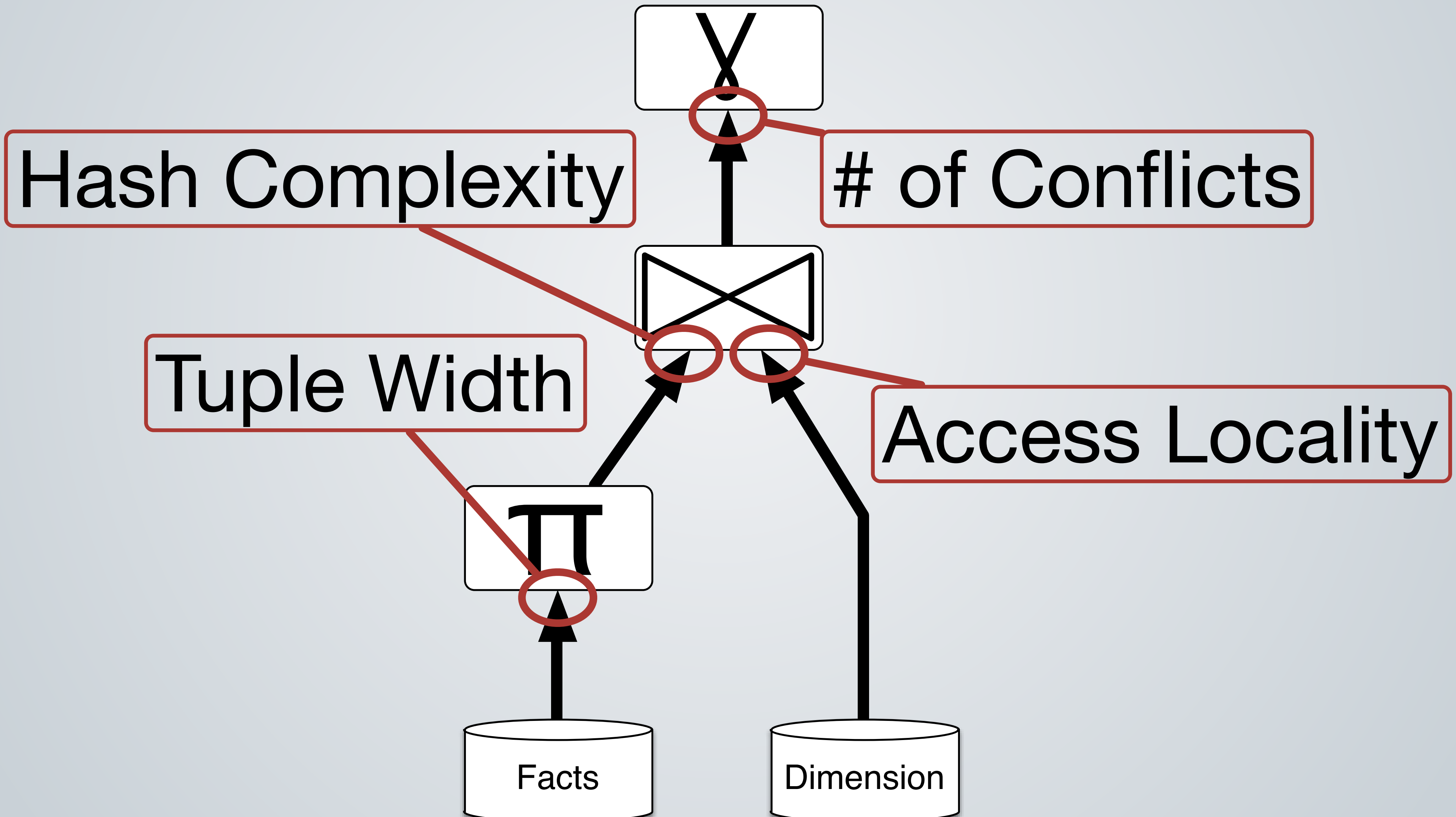


THE OBSTACLE COURSE

DATA-CENTRIC APPLICATIONS HAVE TYPICAL CHOKEPOINTS

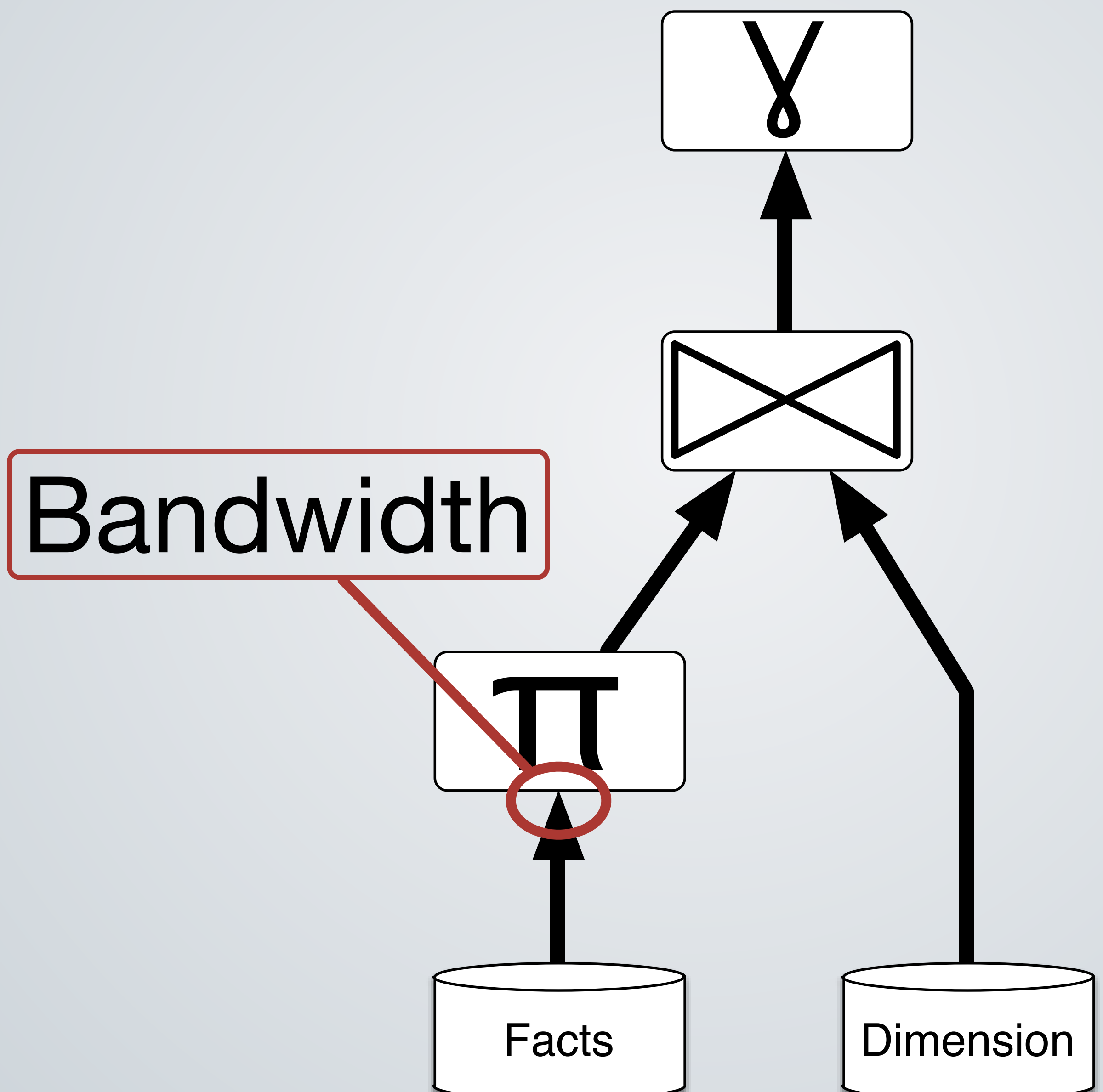


DATA-CENTRIC APPLICATIONS HAVE TYPICAL CHOKEPOINTS

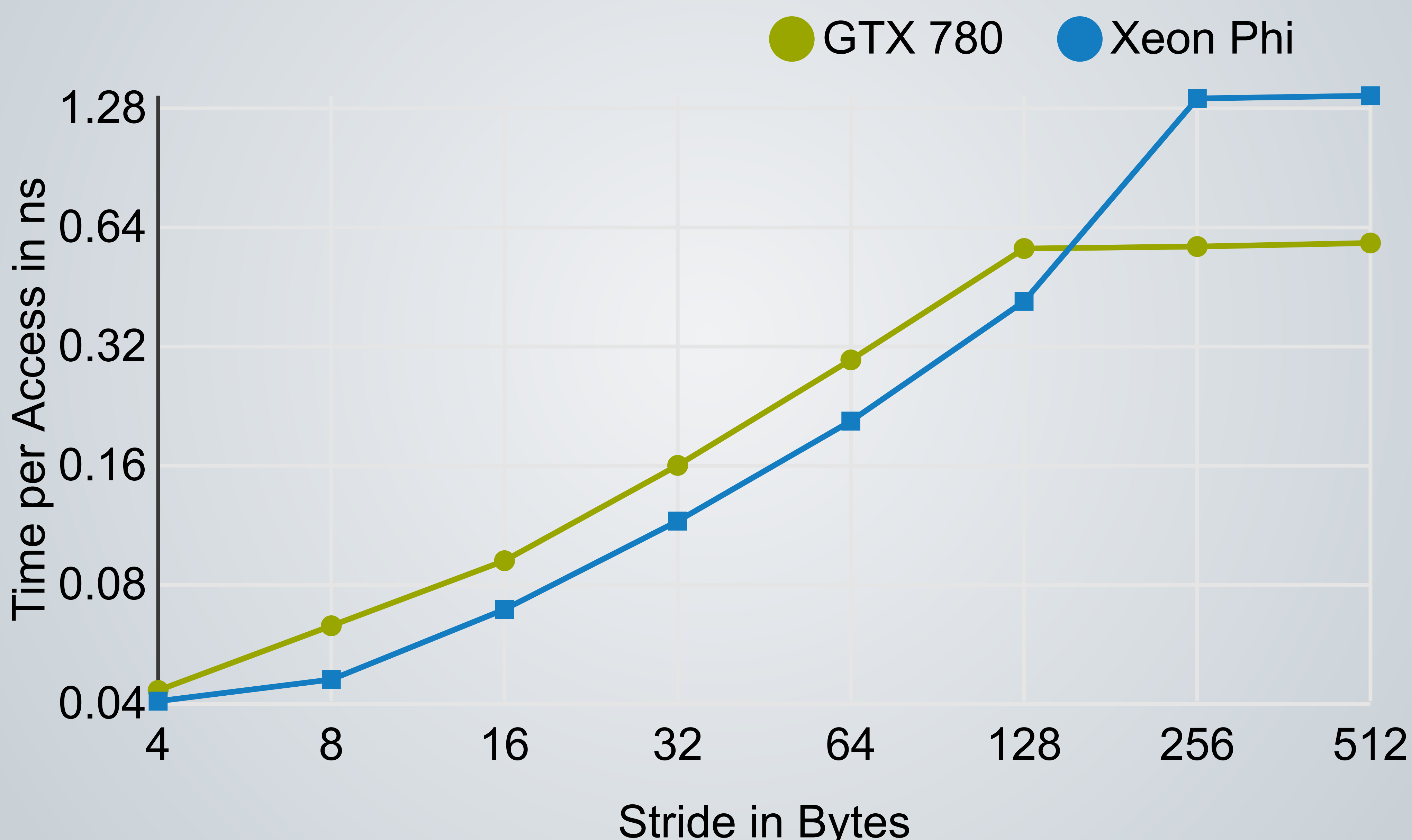


PHI VS. GTX 780

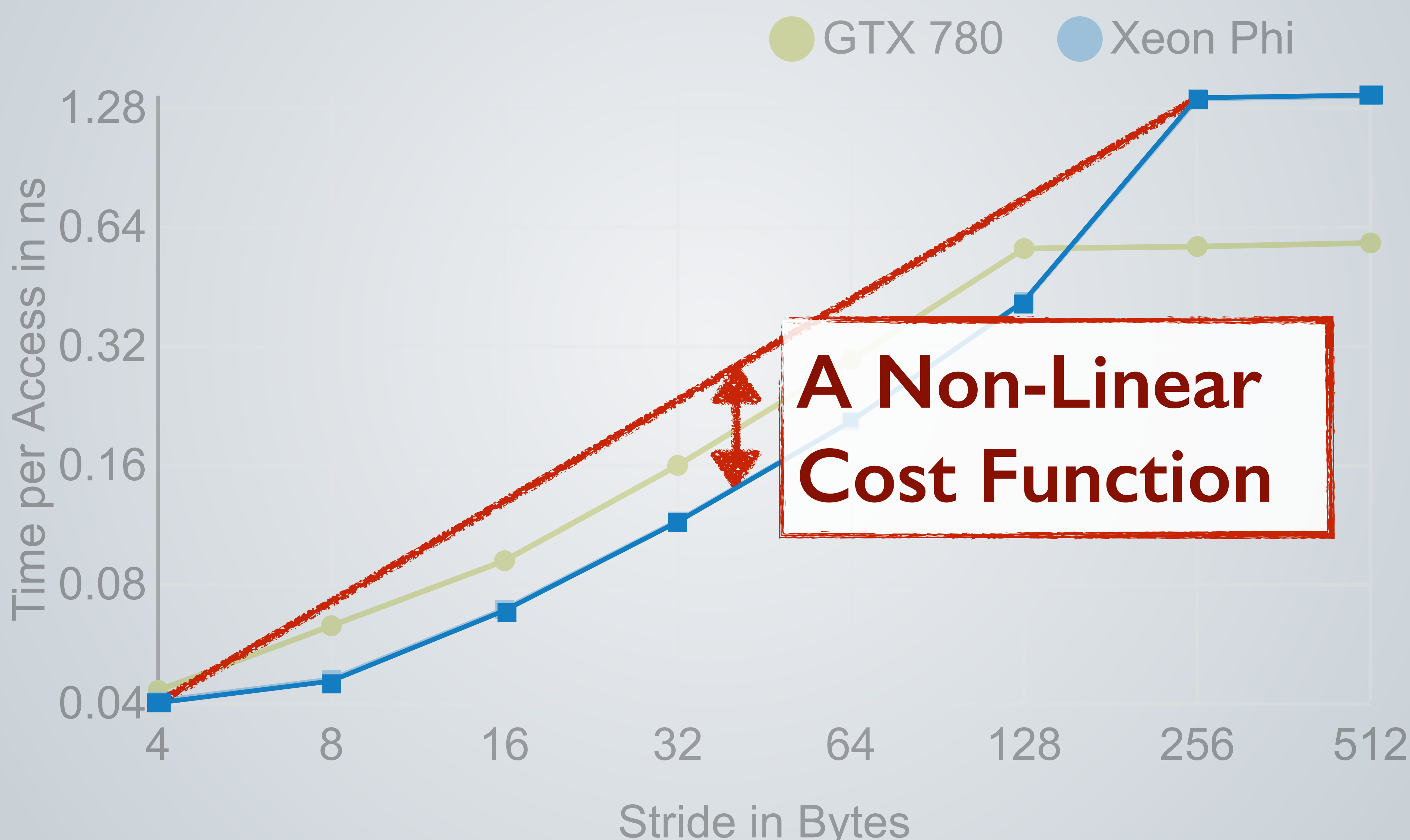
FIRST CHOKEPOINT



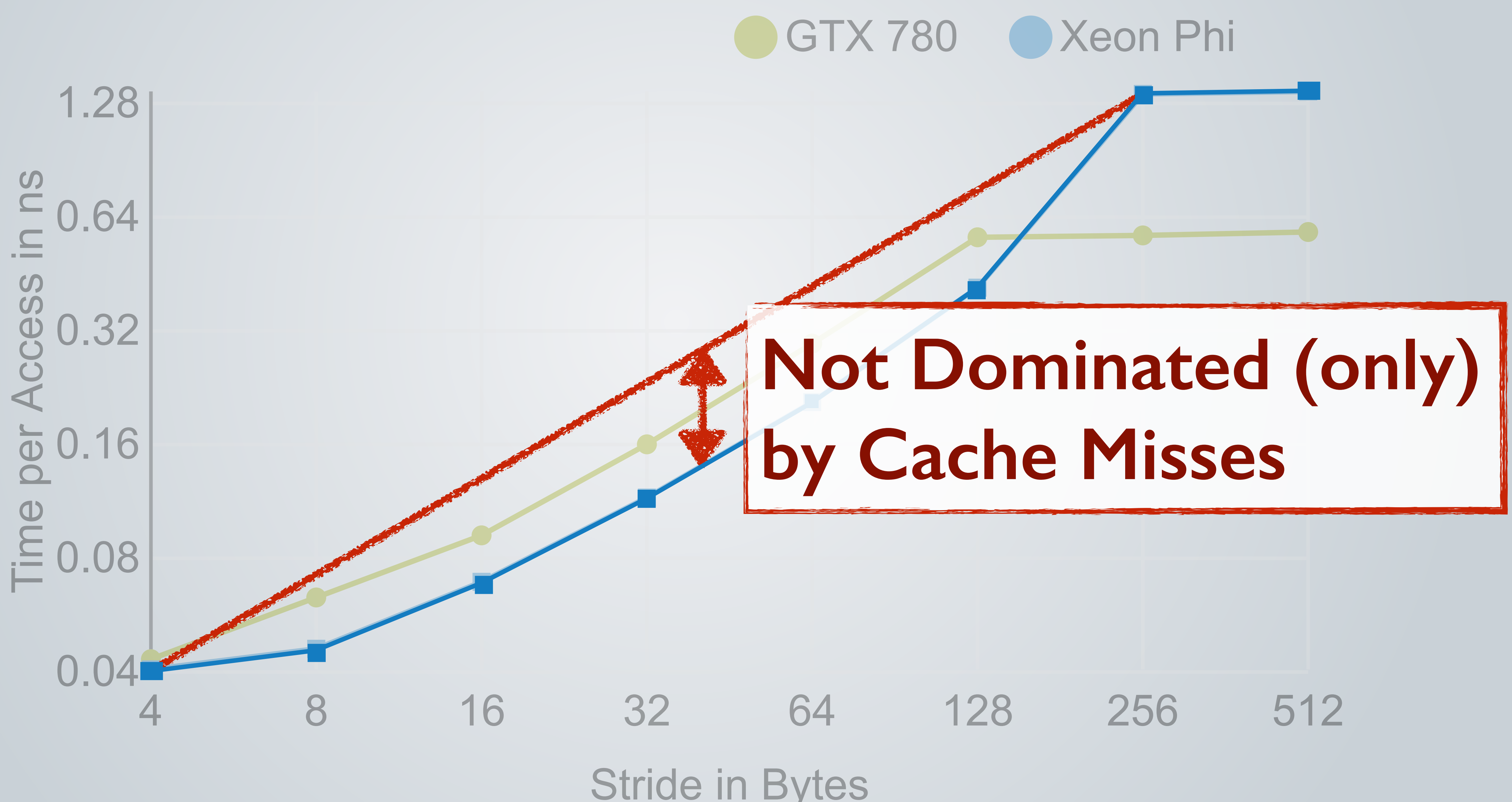
BANDWIDTH OF PHI LOOKS SIMILAR TO GPU AT FIRST GLANCE



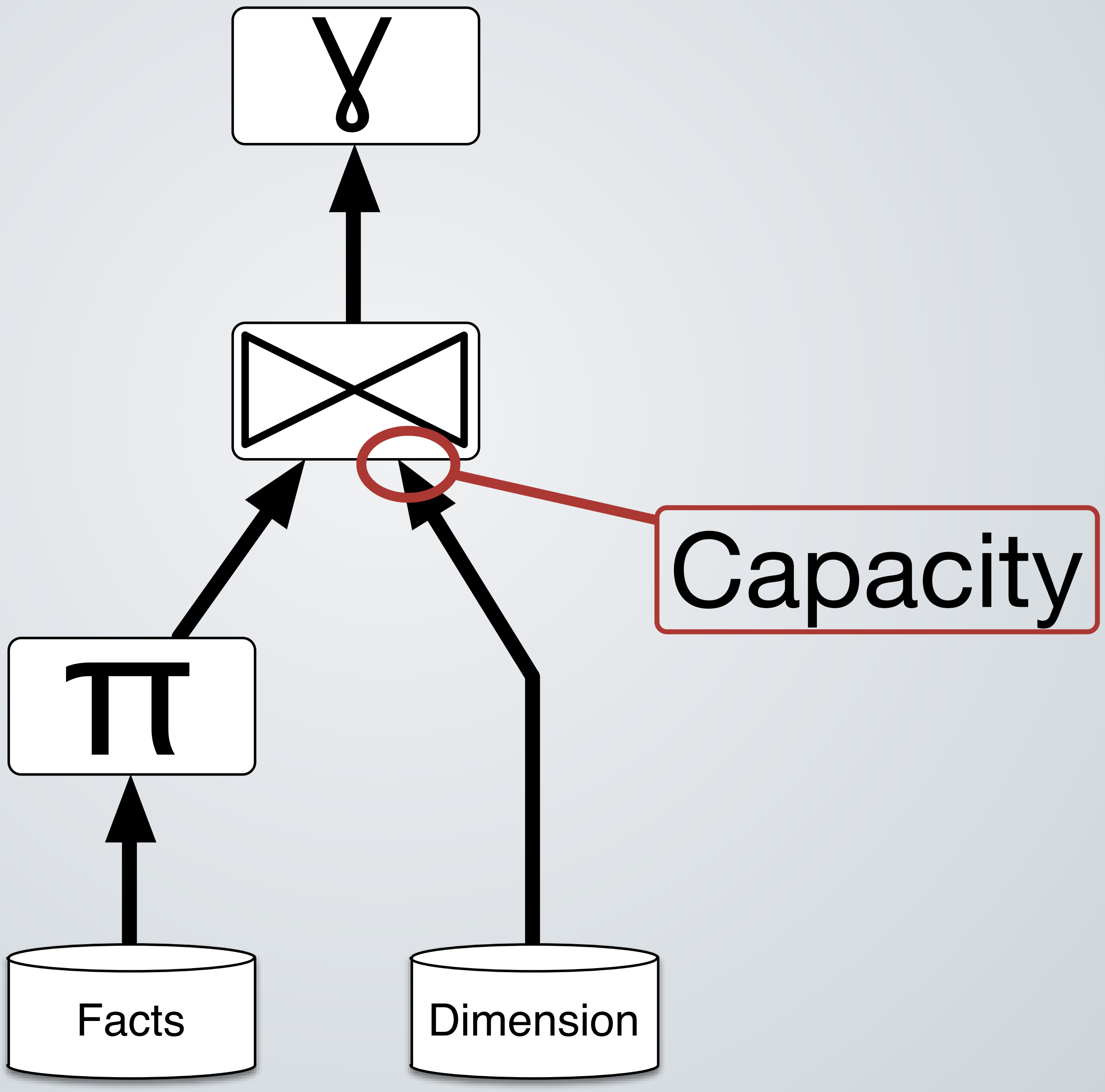
A SECOND GLANCE REVEALS SOMETHING ODD…



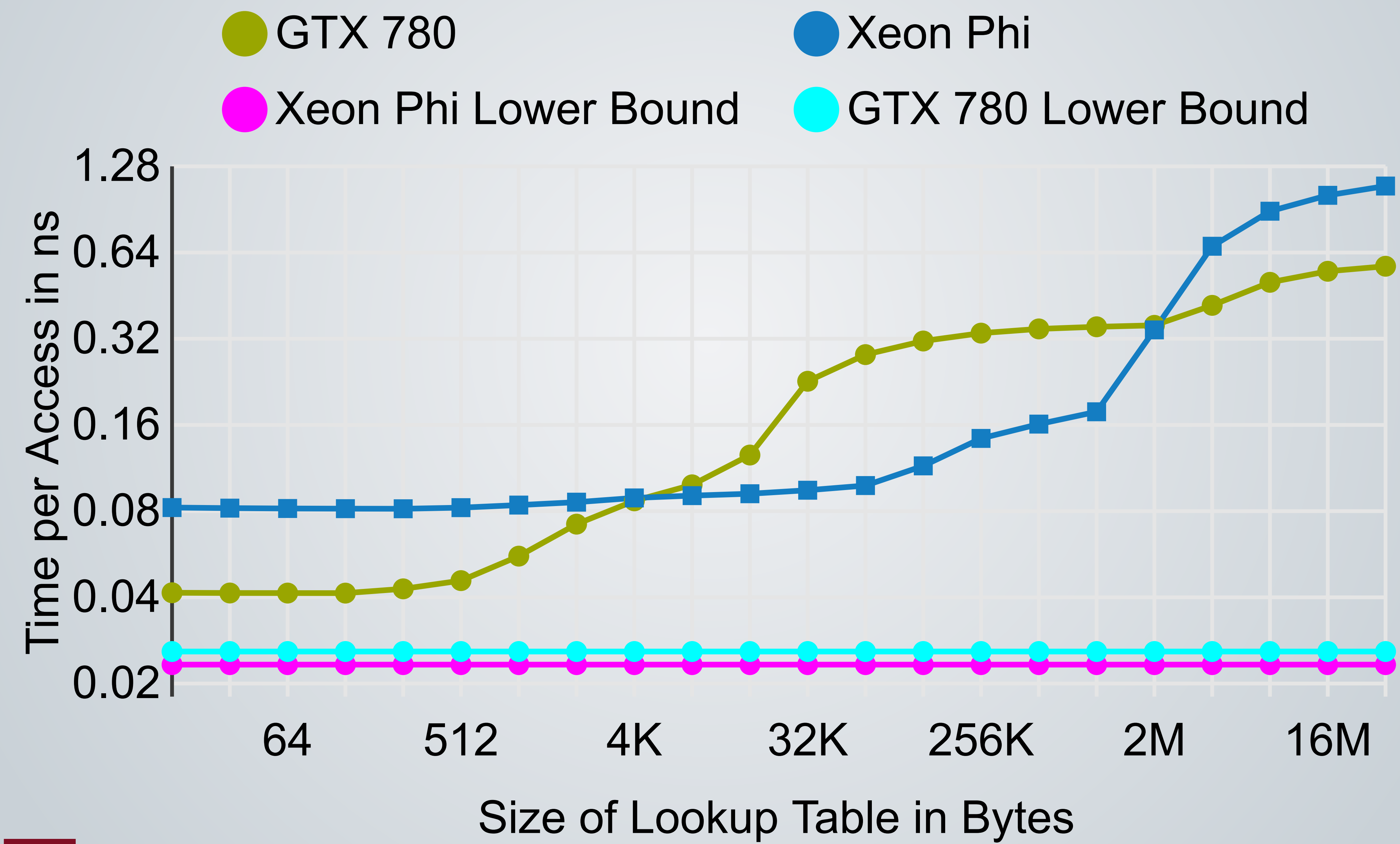
A SECOND GLANCE REVEALS SOMETHING ODD…



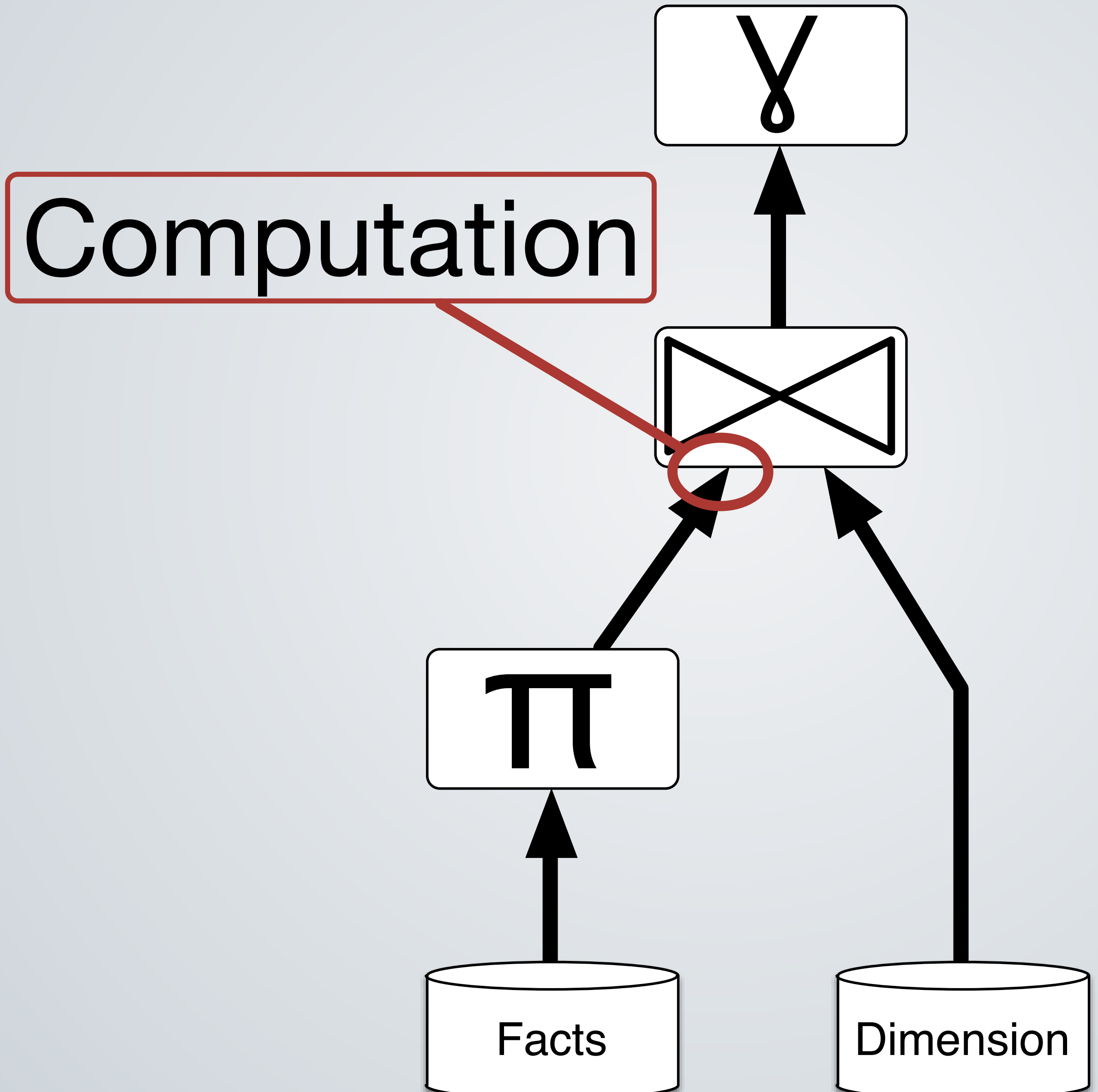
SECOND CHOKEPOINT



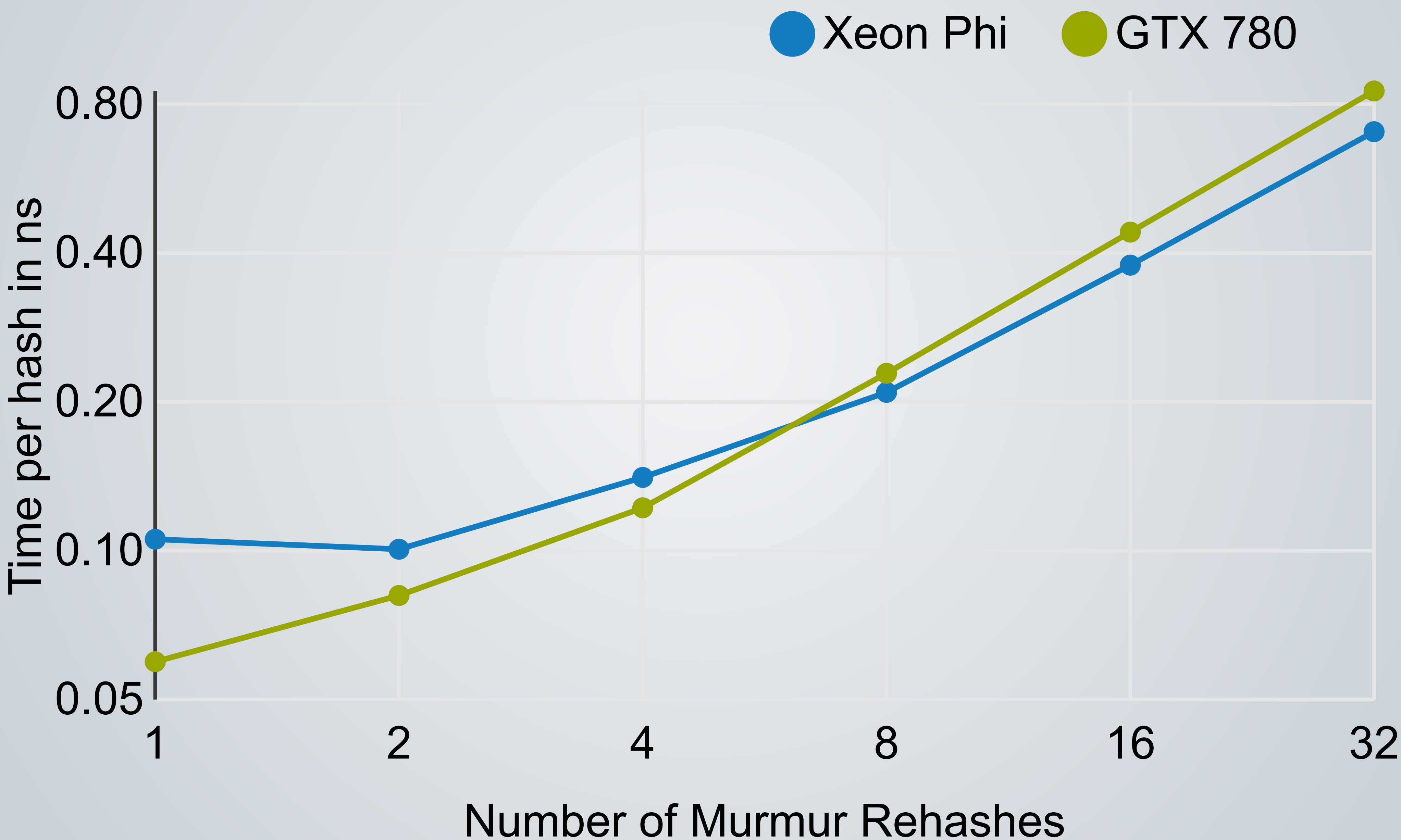
PHI BENEFITS FROM LARGER CACHES



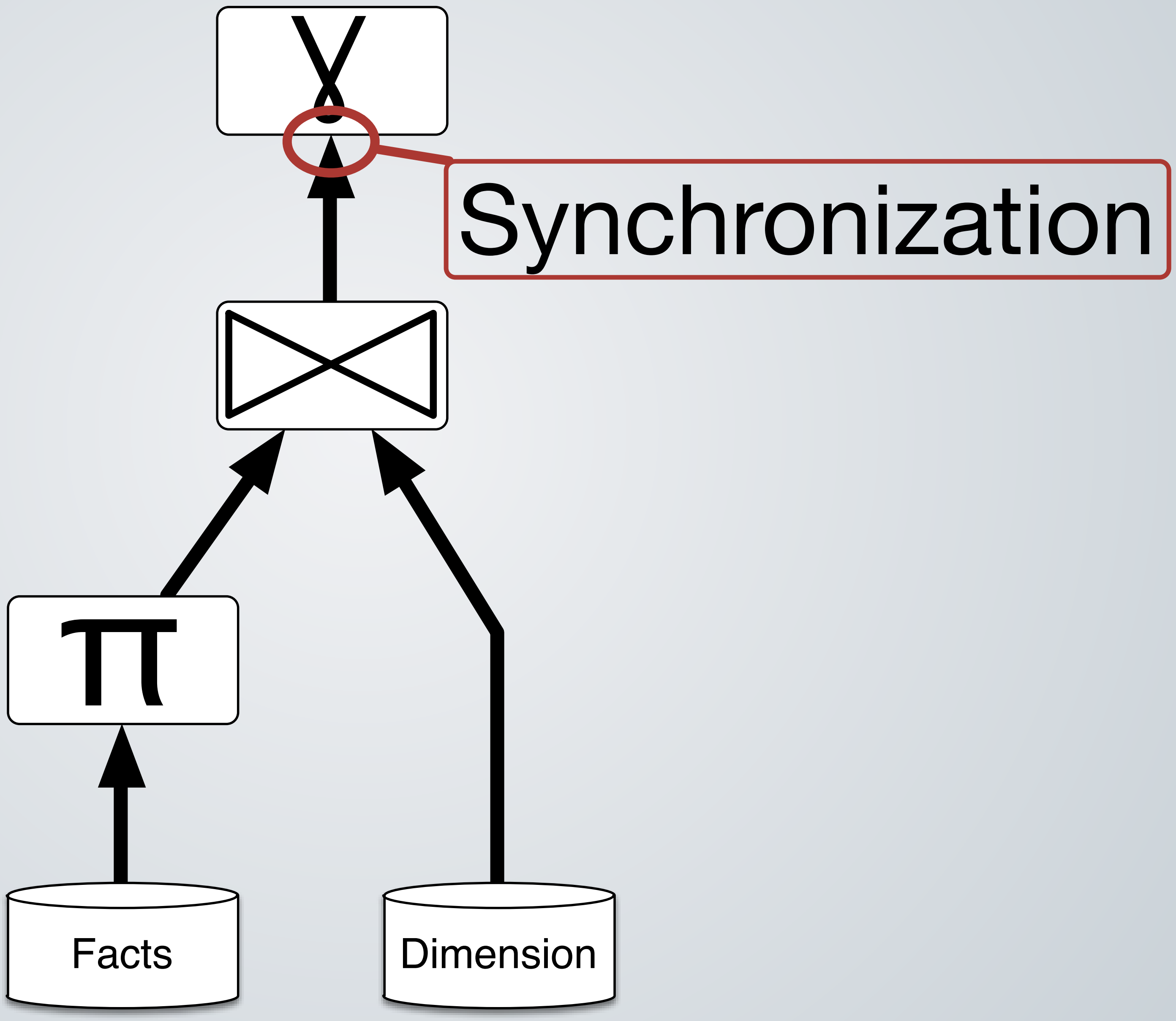
THIRD CHOKEPOINT



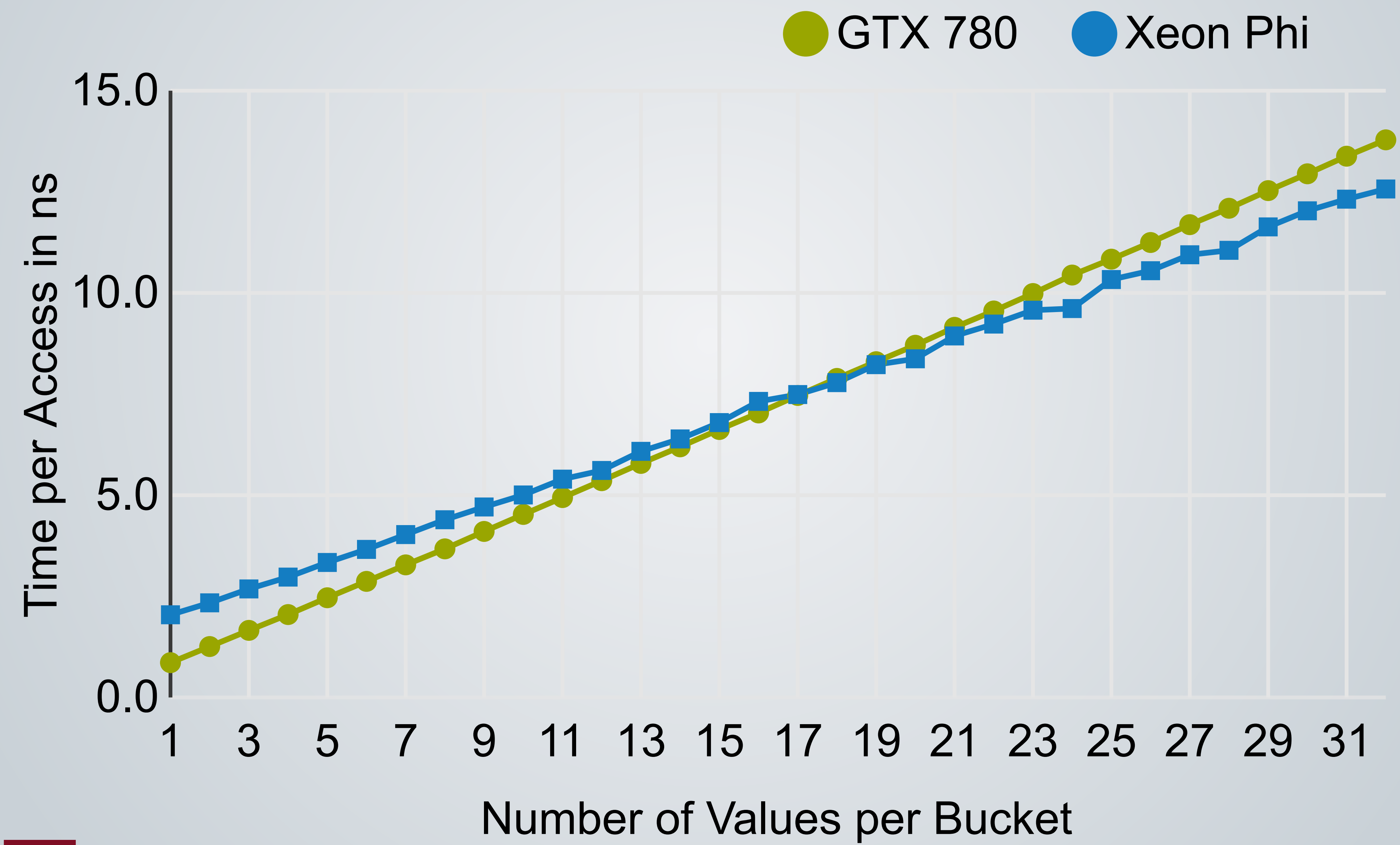
COMPUTATION PERFORMANCE IS VERY SIMILAR…



THIRD CHOKEPOINT



…AND SO IS HASH-BUILDING



RECAP

- Phi & GPU mostly en par in
  - Computation
  - Synchronization
  - Cache-Utilization
- But what is up with the memory access

PHI IN DEPTH

SCATTER/GATHER

LET'S LOOK AT THE DOCUMENTATION

VGATHERDPD - Gather Float64 Vector With Signed Dword Indices

Opcode	Instruction	Description
MVEX.512.66.0F38.W1 92 /r /vsib	vgatherdpd zmm1 {k1}, $U_{f64}(mv_t)$	Gather float64 vector $U_{f64}(mv_t)$ into float64 vector zmm1 using doubleword indices and k1 as completion mask.

Description

A set of 8 memory locations pointed by base address $BASE\_ADDR$ and doubleword index vector $VINDEX$ with scale $SCALE$ are converted to a float64 vector. The result is written into float64 vector zmm1.

Note the special mask behavior as only a subset of the active elements of write mask k1 are actually operated on (as denoted by function $SELECT\_SUBSET$). There are only two guarantees about the function: (a) the destination mask is a subset of the source mask (identity is included), and (b) on a given invocation of the instruction, **at least** one element (the least significant enabled mask bit) will be selected from the source mask.

Programmers should always enforce the execution of a gather/scatter instruction to be re-executed (via a loop) until the full completion of the sequence (i.e. all elements of the gather/scatter sequence have been loaded/stored and hence, the write-mask bits all are zero).

Note that accessed element by will always access 64 bytes of memory. The memory region accessed by each element will always be between *elemen_linear_address & (∼0x3F)* and *(element_linear_address & (∼0x3F)) + 63* boundaries.

This instruction has special disp8*N and alignment rules. N is considered to be the size of a single vector element before up-conversion.

Note also the special mask behavior as the corresponding bits in write mask k1 are reset with each destination element being updated according to the subset of write mask k1. This is useful to allow conditional re-trigger of the instruction until all the elements from a given write mask have been successfully loaded.

The instruction will #GP fault if the destination vector zmm1 is the same as index vector $VINDEX$.

Operation

```
// instruction works over a subset of the write mask
ktemp = SELECT_SUBSET(k1)

// Use mv_t as vector memory operand (VSIB)
for (n = 0; n < 8; n++) {
  if (ktemp[n] != 0) {
```

LET'S LOOK AT THE DOCUMENTATION

Instruction	Description
vgatherdpd zmm1 {k1}, $U_{f64}(mv_t)$	Gather float64 vector $U_{f64}(mv_t)$ into float64 vector zmm1 using doubleword indices and k1 as completion mask.

???

LET'S LOOK AT THE DOCUMENTATION

A set of 8 memory locations pointed by base address $BASE\_ADDR$ and doubleword index vector $VINDEX$ with scale $SCALE$ are converted to a float64 vector. The result is written into float64 vector zmm1.

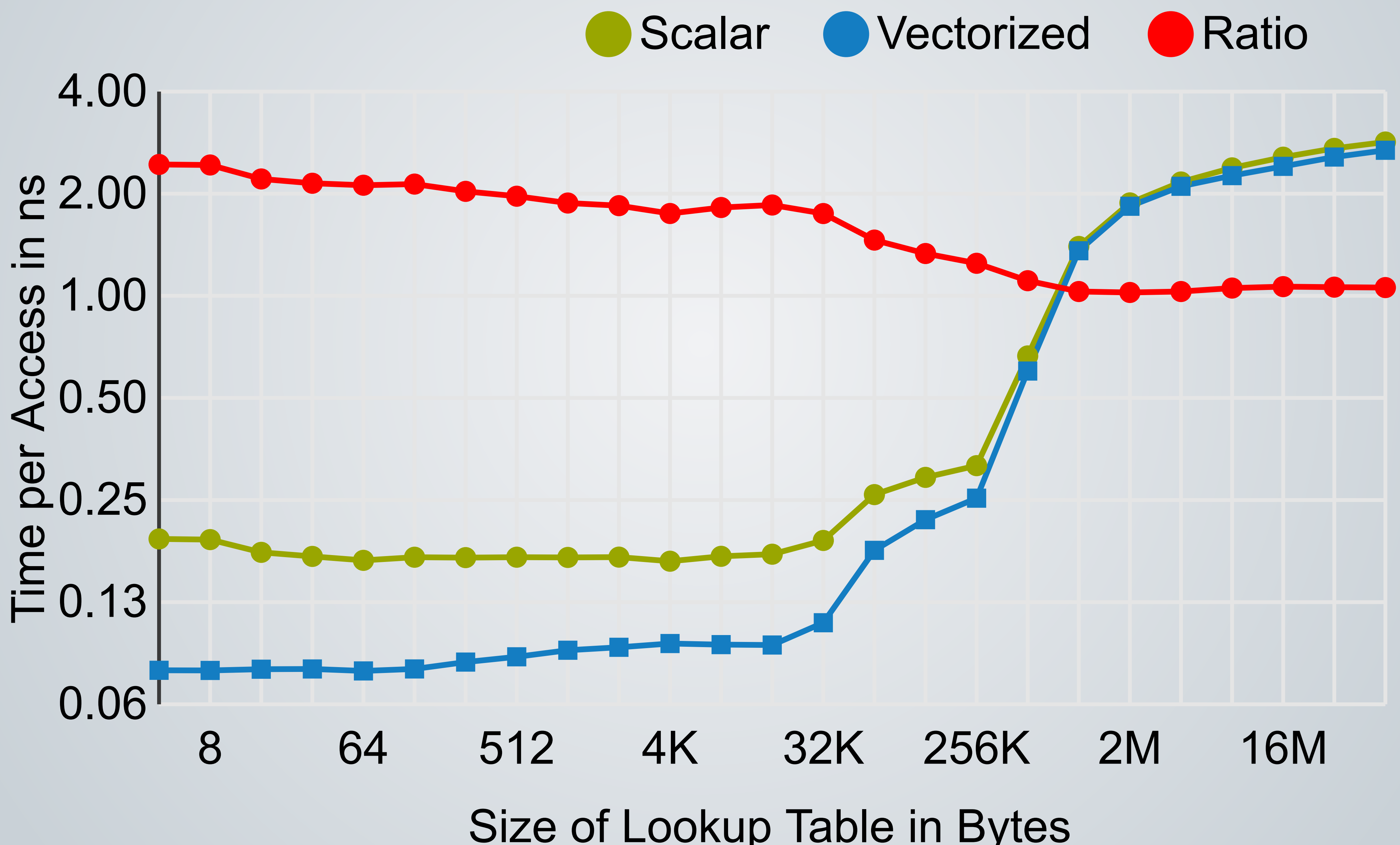
Note the special mask behavior as only a subset of the active elements of write mask k1 are actually operated on (as denoted by function $SELECT\_SUBSET$). There are only two guarantees about the function: (a) the destination mask is a subset of the source mask (identity is included), and (b) on a given invocation of the instruction, **at least** one element (the least significant enabled mask bit) will be selected from the source mask.

???

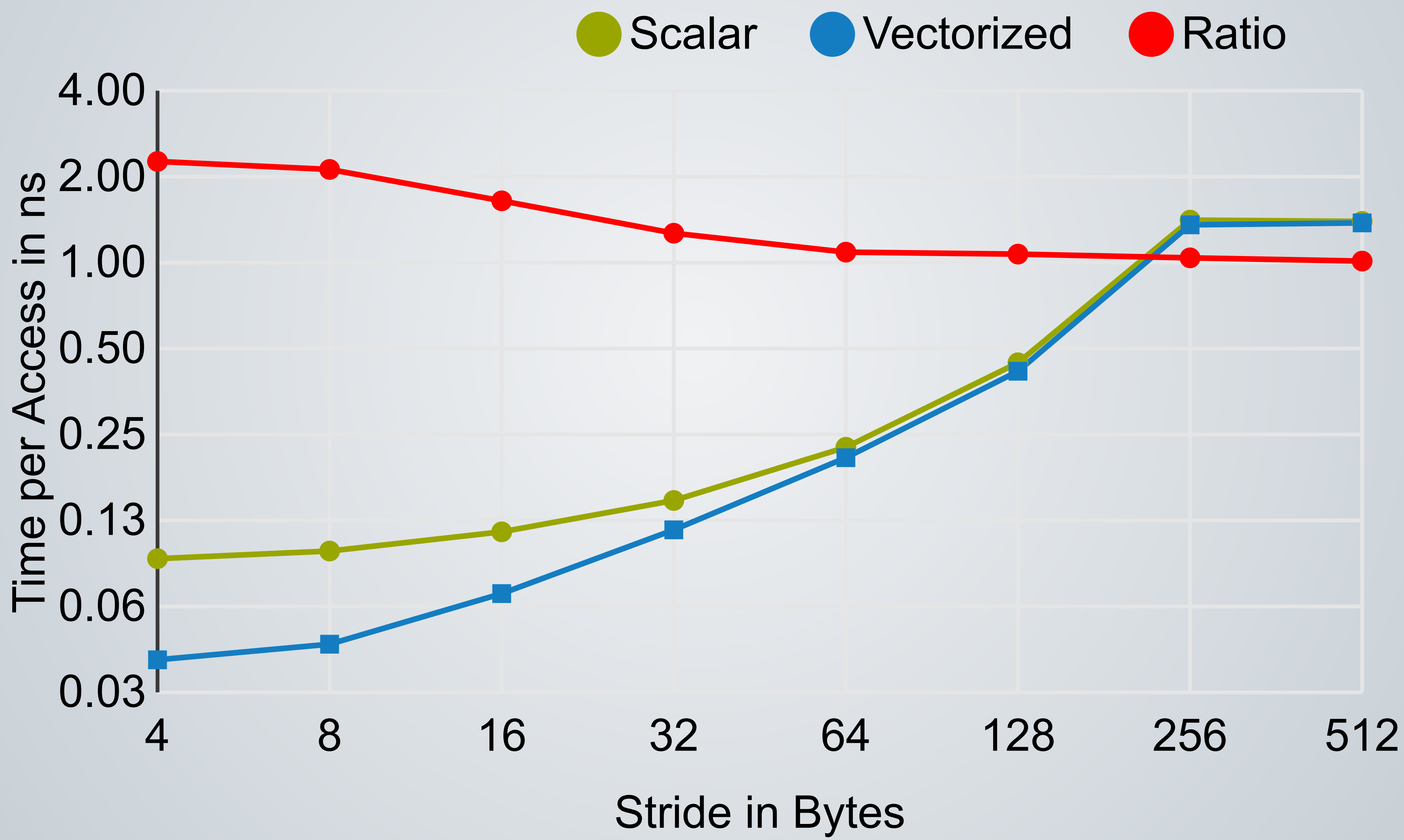
Programmers should always enforce the execution of a gather/scatter instruction to be re-executed (via a loop) until the full completion of the sequence (i.e. all elements of the gather/scatter sequence have been loaded/stored and hence, the write-mask bits all are zero).

💣☠️🔫

GATHER-LOADING ONLY YIELDS MODERATE LOOKUP IMPROVEMENT…

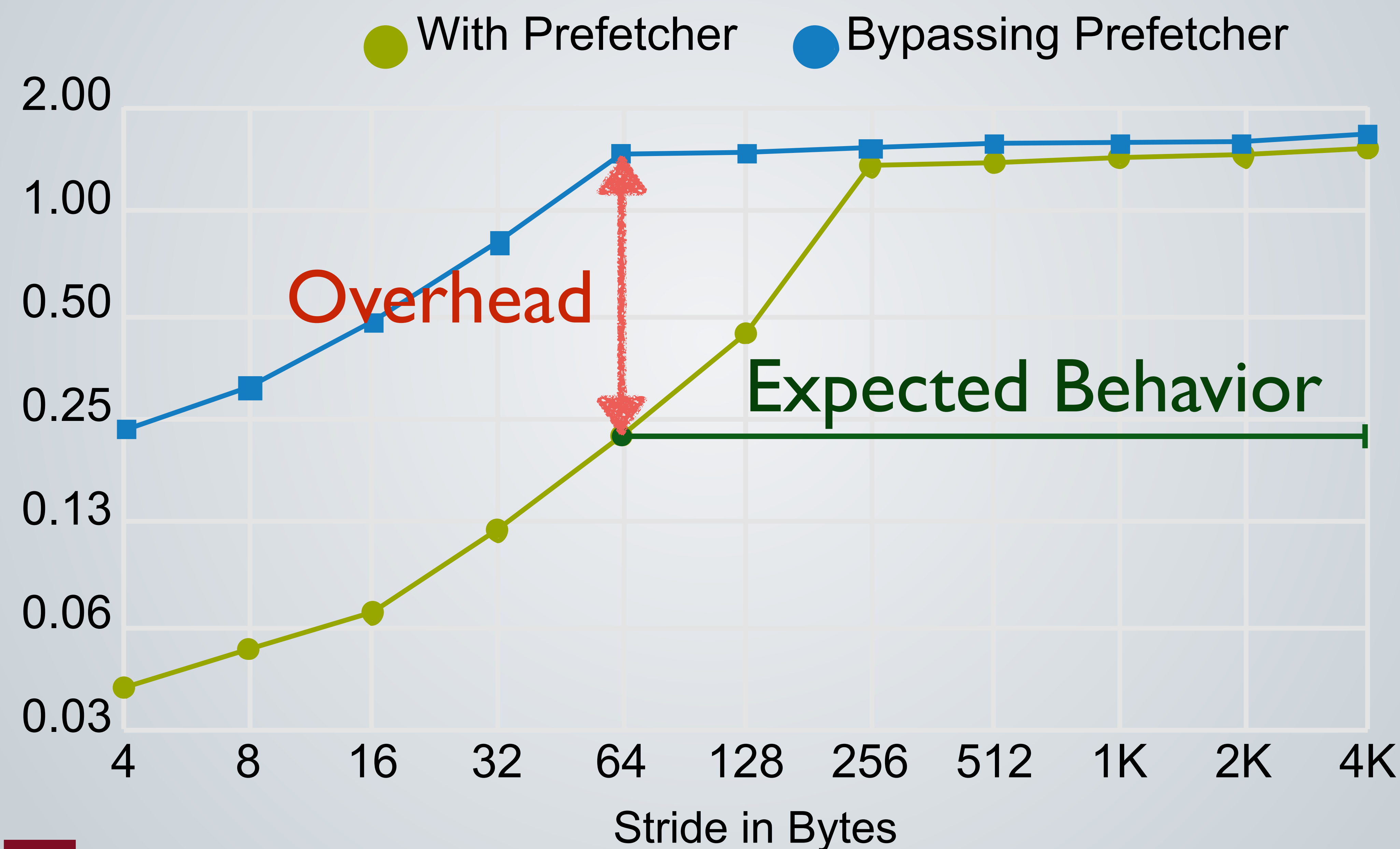


…SAME FOR PROJECTIONS

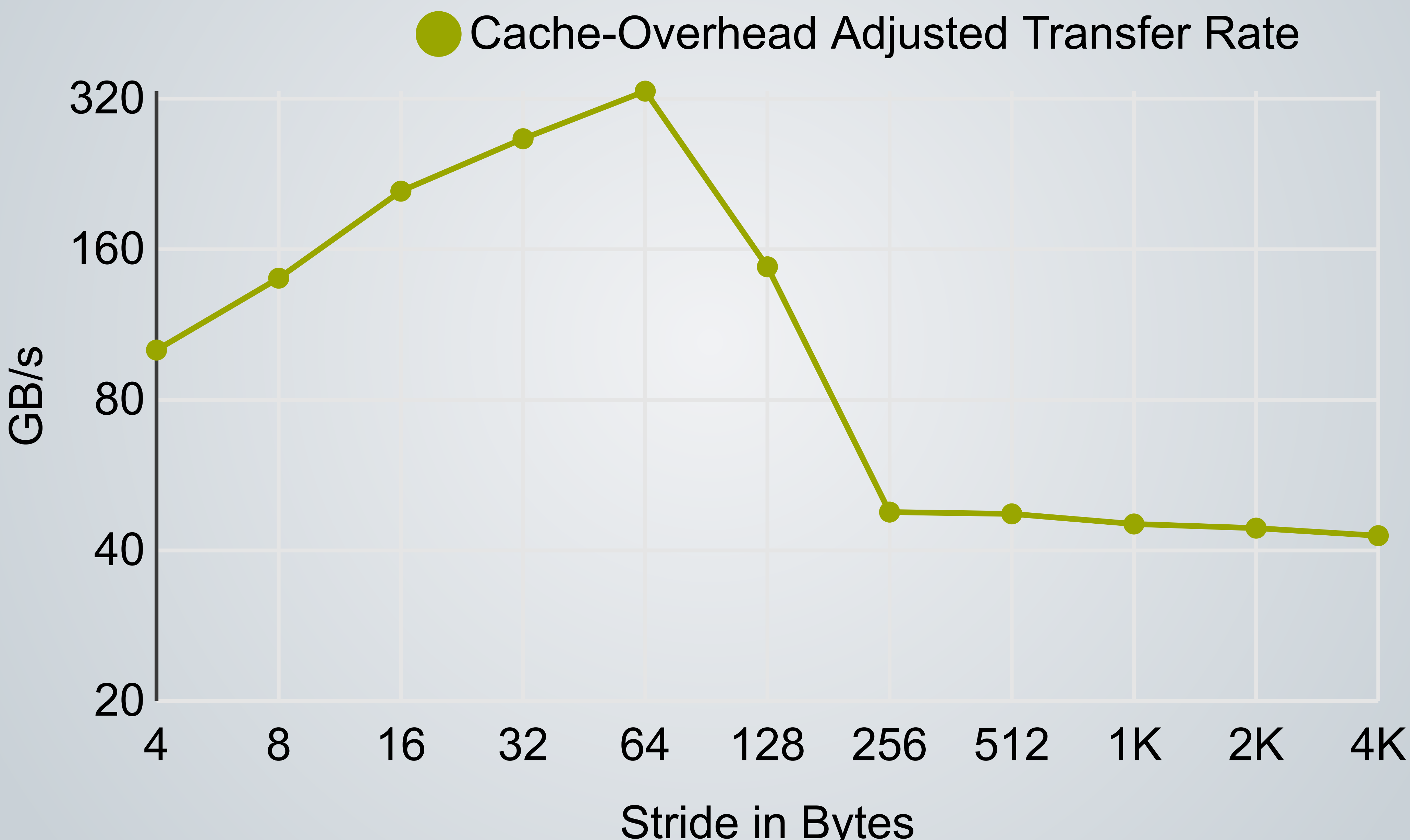


PREFETCHING

THE PHI PREFETCHER SEEMS OVERLY AGGRESSIVE



ONLY WHEN FACTORING IN TRANSFER OVERHEAD IS THE NOMINAL PHI BANDWIDTH ACHIEVED



TAKEAWAY

- Phi is en-par with mid-level GPUs compute-intensive applications
- Data-intensive performance is weird, though:
  - Prefetcher seems overly aggressive
  - Gather implementation seems half-baked: to few cache ports?

THANK YOU