# Parallel Computing Concepts

But ….
Why are we interested in Parallel Computing?
Because we do Supercomputing?
But …

# What is a Supercomputing?

John M Levesque
CTO Office – Applications
Director
Cray's Supercomputing Center of Excellence

# Well we know who the father is

Seymour Cray

**September 28, 1925**
**To**
**October 5, 1996**

# First – a supercomputer is:

- **A computer whose peak performance is capable of solving the largest complex science/engineering problems**
- **Peak performance is measured by**
  - Number of total processors in the system multiplied by the peak of each processor
  - Peak of each processor is measured in Floating Point Operations per second.
    - A scalar processor can usually generate 2-4 64 Bit results (Multiply/Add) operations each clock cycle
    - The Intel Skylake and Intel Knight's Landing has vector units that can generate 16-32 64 bit results each clock cycle.
      - Skylake's clock cycle varies and the fastest is 3-4 GHZ for 64-128 GFLOPS
- **Big Benefit from vectorizing applications**
- **Summit has 4,356 nodes, each one equipped with two 22-core Power9 CPUs, and six NVIDIA Tesla V100 GPUs for a whopping 187.6 Petaflops and it is the fastest computer in the world**

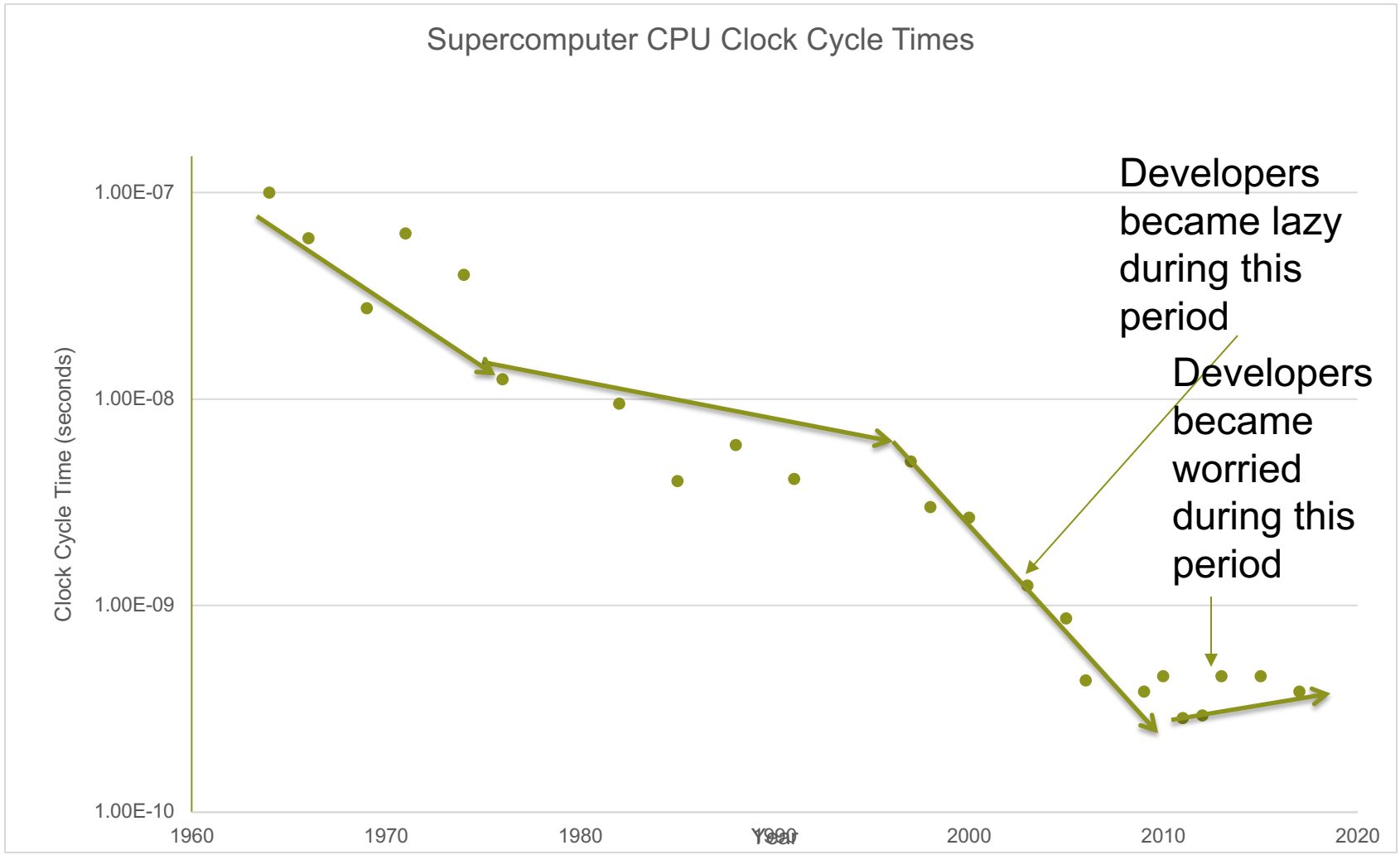
# However; The Supercomputer is just one piece of Supercomputing

- **We need three things to be successful at Supercomputing**
  - The Supercomputer – Summit is great example
  - Large Scientific problem to solve – we have many of those
  - Programming expertise to port/optimize existing applications and/or develop new applications to effectively utilize the target Supercomputing architecture
    - This is an area where we are lacking – too many application developers are not interested in how to write optimized applications for the target system
    - When you achieve less than one percent of the peak performance on a 200 Petaflop computer – you are only achieving less than 2 Petaflops
    - There are a few sites who are exceptions to this.

# Ways to make a computer faster

- **Lower the clock cycle – increase clock rate**
  - As the clock cycle becomes shorter the computer can perform more computations each second. Remember we are determining the peak performance by counting the number of floating point operations/second.

# For the last ten years the clock cycle is staying staying the same or increasing



# Ways to make a computer faster

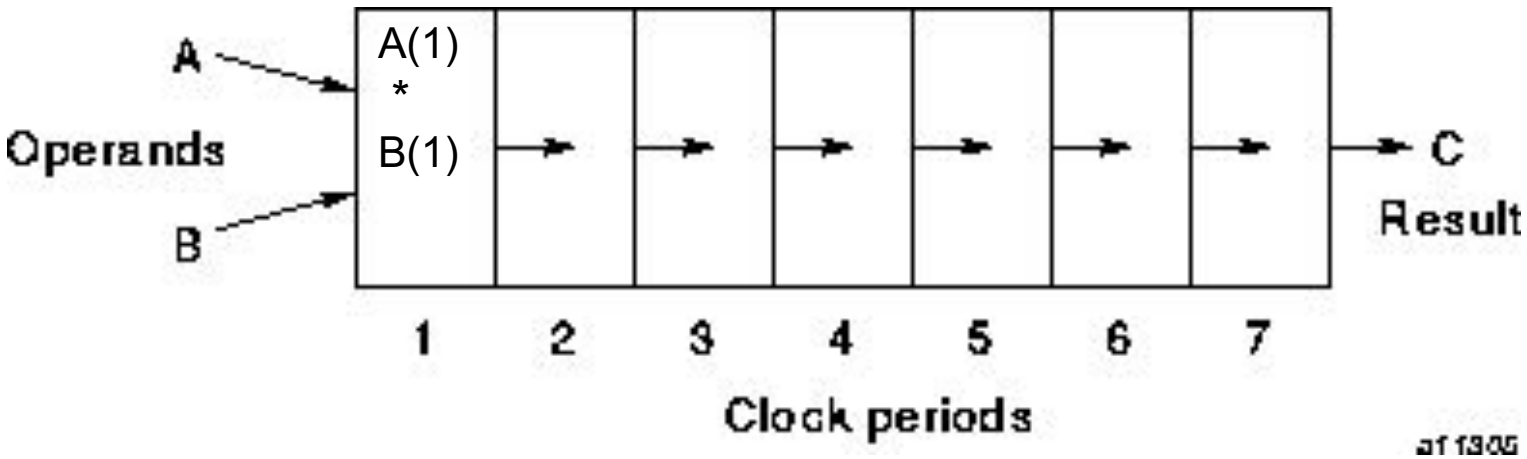
- **Lower the clock cycle – increase clock rate**
- **Generate more than one result each clock**
  - Single Instruction, Multiple Data - SIMD
    - Vector instruction utilizing multiple segmented functional units
    - Parallel instruction
  - Mulitple Instructions, Multiple Data - MIMD
    - Data Flow
- **Utilize more than one processor**
  - Shared memory parallelism
    - All processors share memory
  - Distributed memory parallelism
    - All nodes have separate memories, but:
      - A node can have several processors sharing memory within the node

# Vector Unit is an Assembly Line

# Cray 1 Vector Unit

Time to compute= Startup + Number of Operands

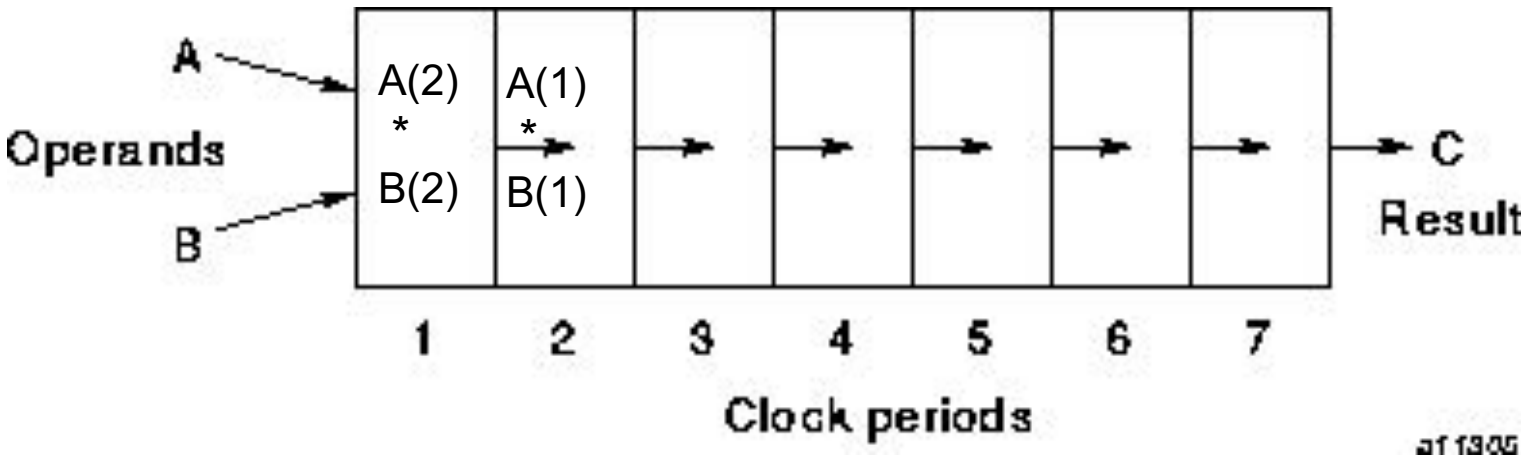
Clock Cycle = 1



# Cray 1 Vector Unit

Time to compute= Startup + Number of Operands

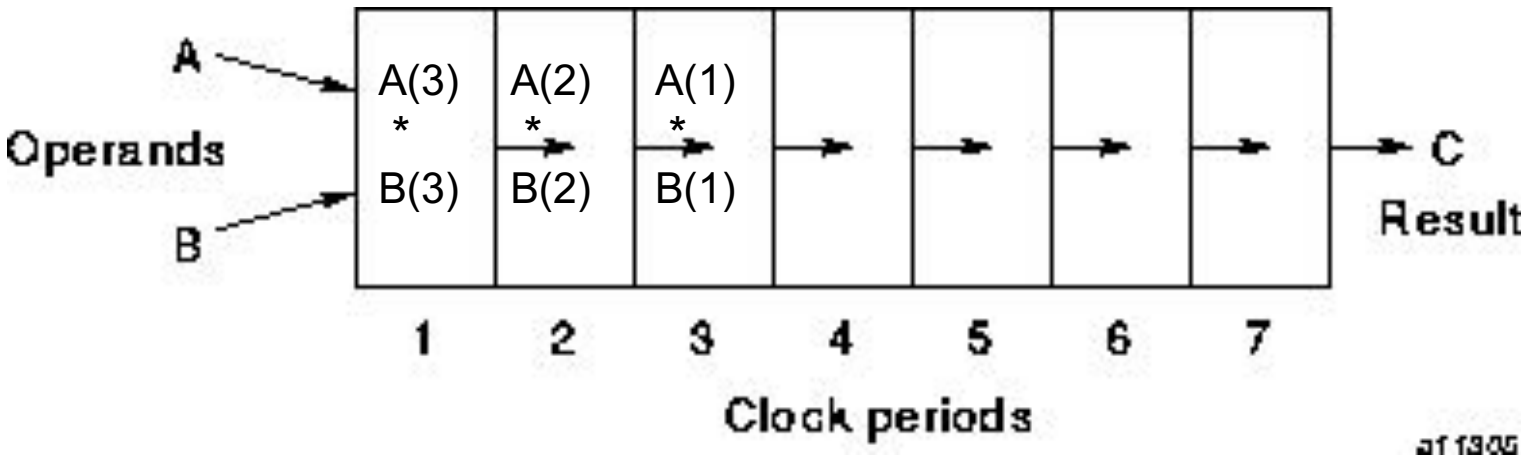
Clock Cycle = 2



# Cray 1 Vector Unit

Time to compute= Startup + Number of Operands

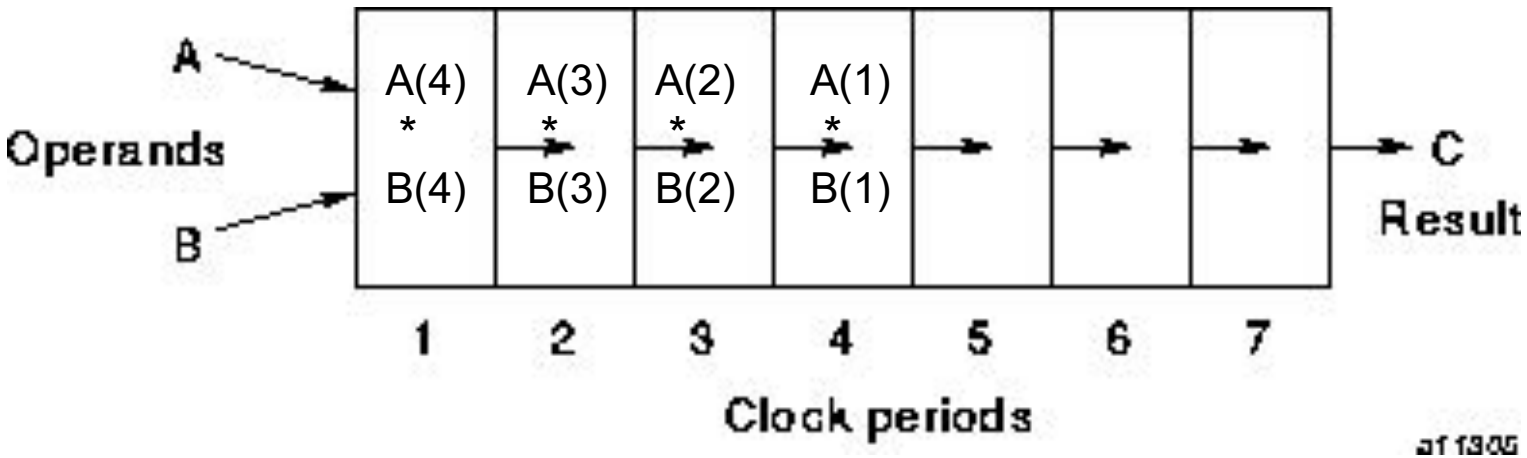
Clock Cycle = 3



# Cray 1 Vector Unit

Time to compute= Startup + Number of Operands

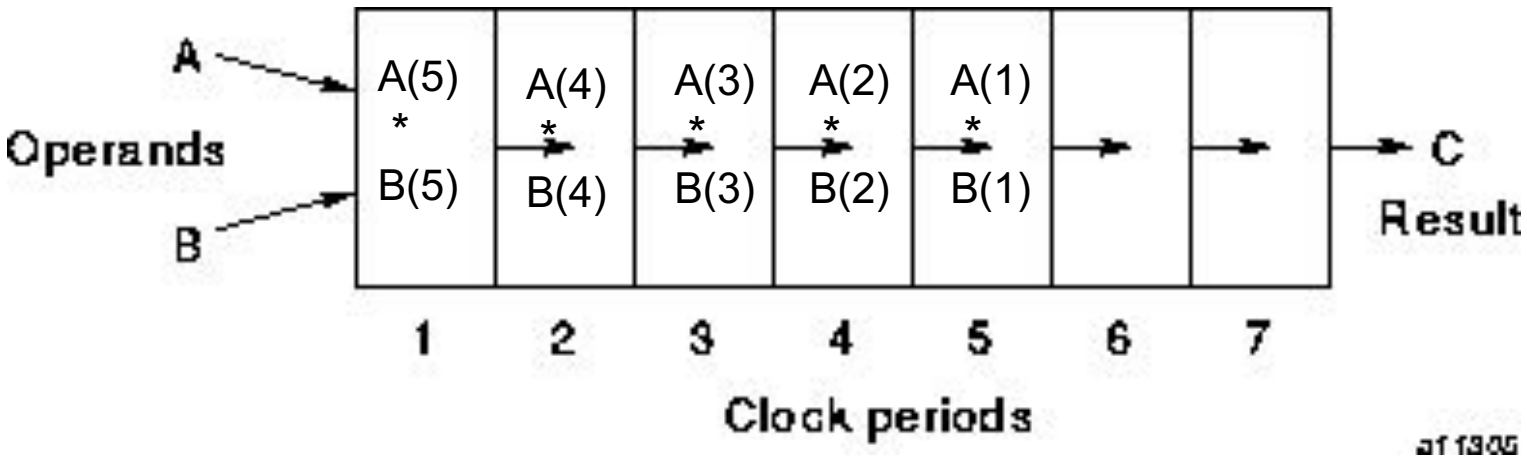
Clock Cycle = 4



# Cray 1 Vector Unit

Time to compute= Startup + Number of Operands

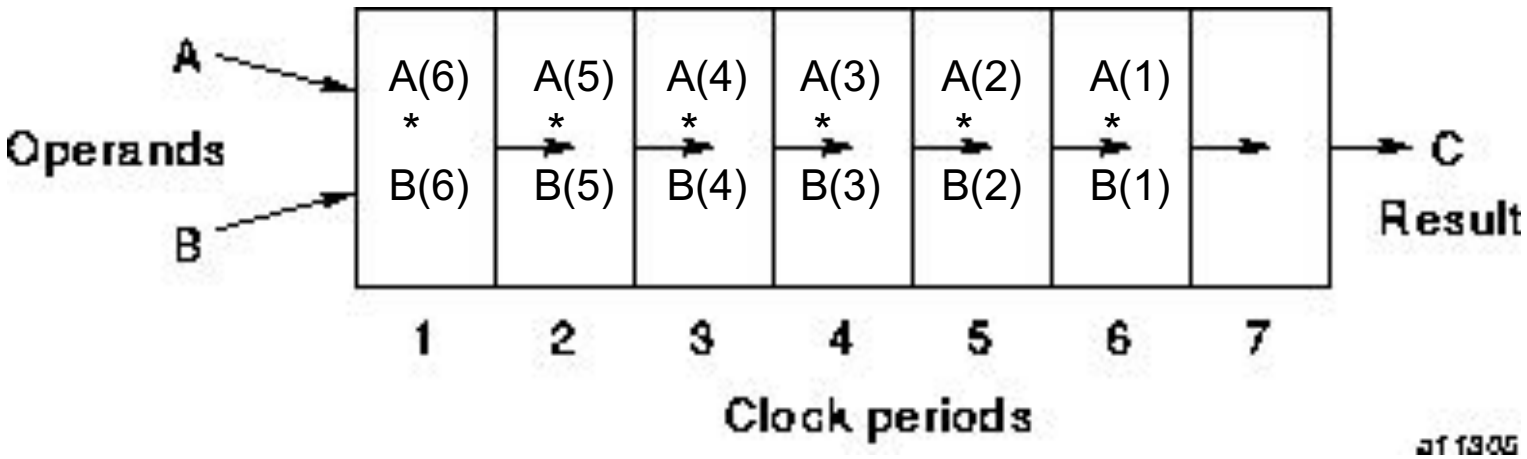
Clock Cycle = 5



# Cray 1 Vector Unit

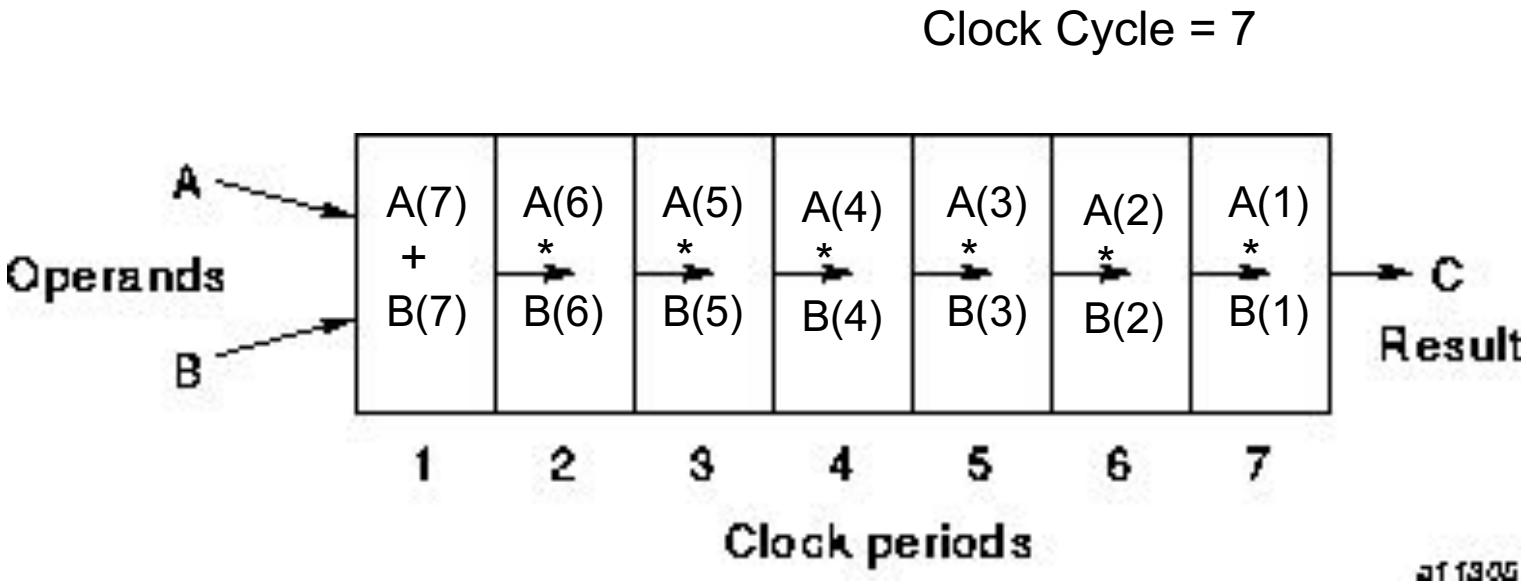
Time to compute= Startup + Number of Operands

Clock Cycle = 6



# Cray 1 Vector Unit

Time to compute= Startup + Number of Operands



# SIMD Parallel Unit

Result Rate = [Number of Operands/Number of Processing Elements]

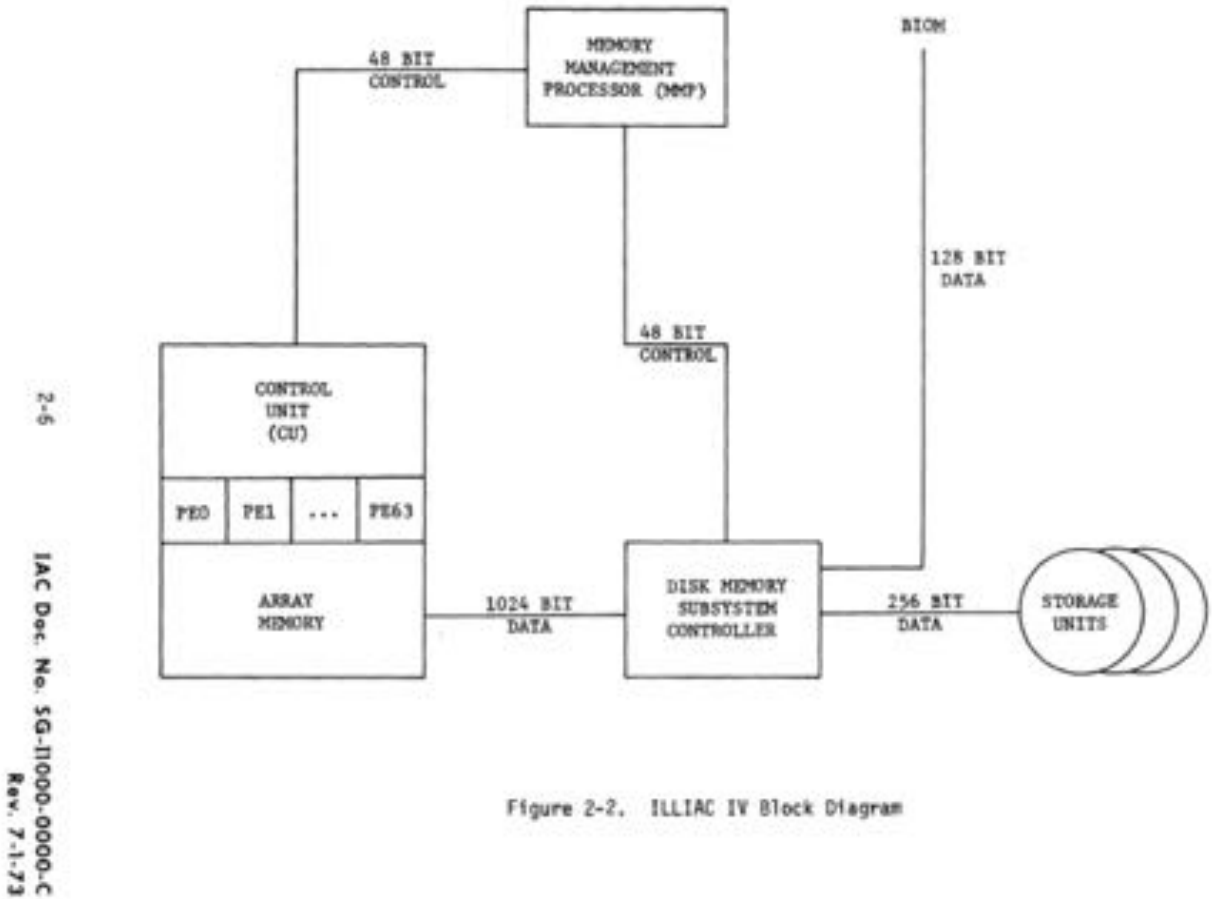
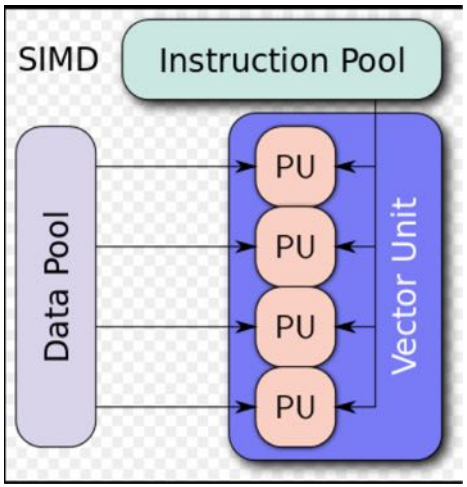



Figure 2-2. ILLIAC IV Block Diagram

# What are today's Vector processors?

- They are not like the traditional Cray vector processors
- The are actually like the Illiac.



# Outline

- **What was a Supercomputer in the 1960s,70s,80s,90s and the 2000s**
  - Single Instruction, Multiple Data (SIMD)
    - Vector
    - Parallel
  - Multiple Instructions, Multiple Data (MIMD)
    - Distributed Memory Systems
      - Beowulf, Paragon, nCUBE, Cray T3D/T3E, IBM SP
    - Shared Memory Systems
      - SGI Origin
- **What is a Supercomputer Today**
  - déjà vu
    - MIMD collection of distributed nodes with a MIMD collection of shared memory processors with SIMD instructions
- **So it is important to understand the history of Supercomputing**

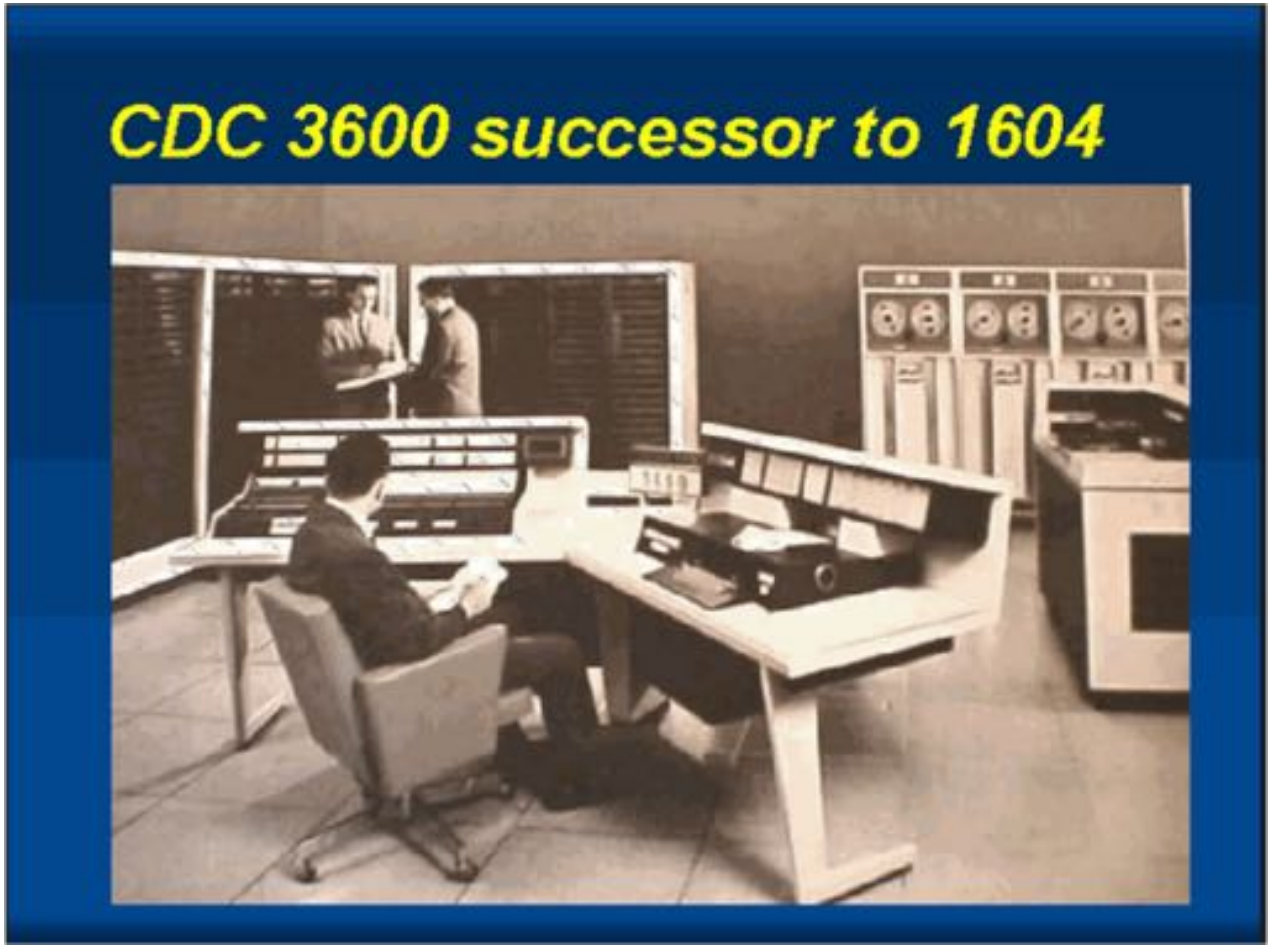
# Who is the historian - John Levesque?

- **1964 – 1965**
  - Delivered Meads Fine Bread, (Student at University of New Mexico 62-72)
- **3/1966 – 1969**
  - Sandia Laboratories – Underground Physics Department (CDC3600)
- **1969 – 1972**
  - Air Force Weapons Laboratory (CDC 6600)
- **1972 – 1978**
  - R & D Associates - (CDC 7600, Illiac IV, Cray 1, Cyber 203-205)
- **1978 -1979**
  - Massachusetts Computer Associates (Illiac IV, Cray 1) Parallizer
- **1979 – 1991**
  - Pacific Sierra Research – (Intel Paragon, Ncube, Univac APS) - VAST
- **1991 – 1998**
  - Applied Parallel Research – (MPPs of all sorts, CM5, NEX SX) – FORGE
- **1998 – 2001**
  - IBM Research – Director Advanced Computing Technology Center
- **1/2001 – 9/2001**
  - Times N Systems – Director of Software Development
- **9/2001 – Present ( 11/2018?) ← (9/2021)**
  - Cray Inc – Director – Cray's Supercomputing Center of Excellence

# 1960s

# First System I drove

# Then got job at Sandia Laboratories in Albuquerque



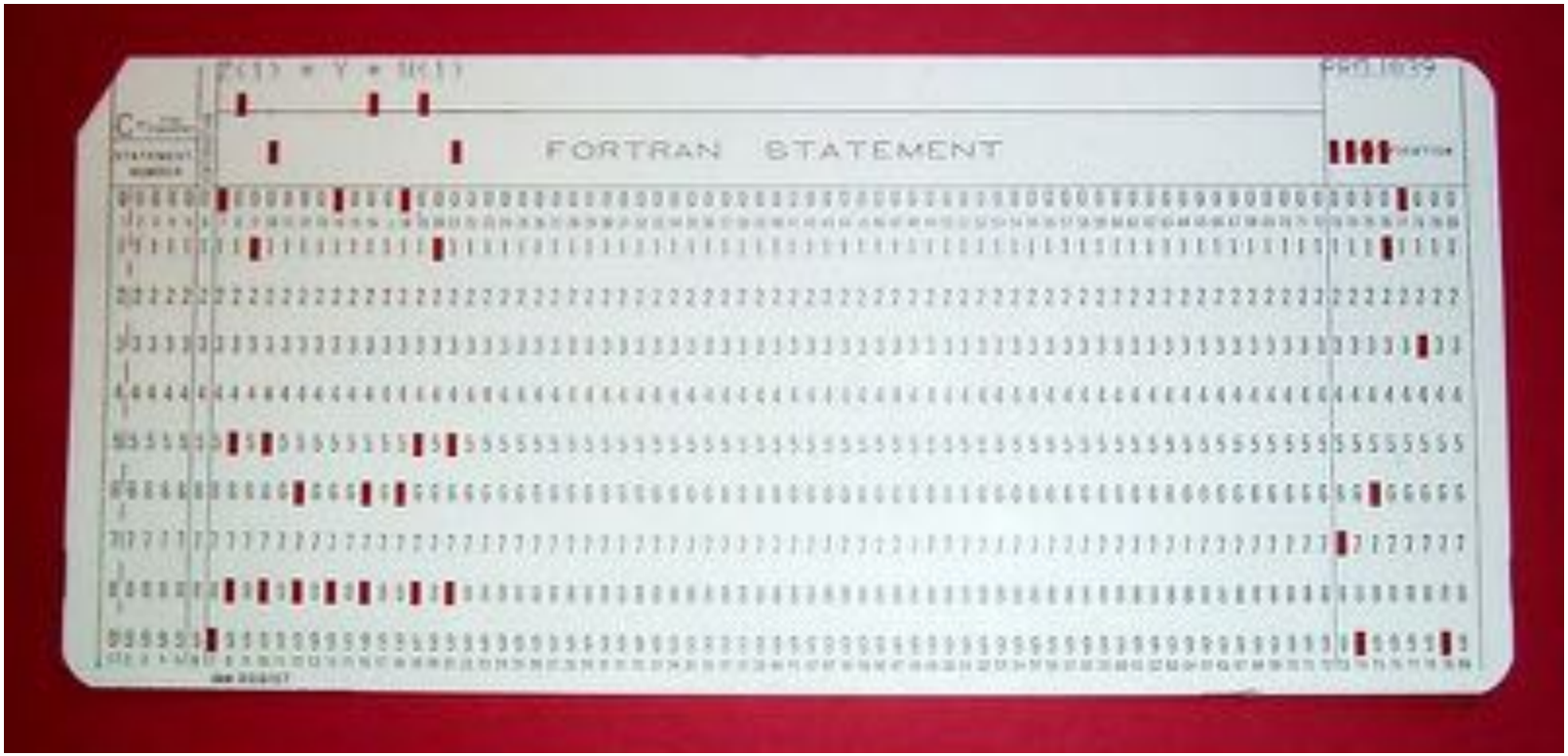
# 1970s

# CDC 6600 – Multiple Functional Units

Could generate an ADD and MULTIPLY at the same time

# Computer input devices

# Why was the Fortran line length 72?



# File organization

# Computer Listing

# Computational Steering in the 70s

Set a sense switch, dump the velocity Vector field to tape, take tape to Calcomp Plotter.

# Then in the early 70's I worked on the Illiac IV

Anyone know why the door is open?

Programming the 64 SIMD units is exactly the same as programming the 32 SIMT threads in a Nvidia Warp

# Actually accessed the Illiac IV remotely


Contrarily to popular belief, Al Core did not invent the Internet – Larry Roberts of ARPA did.

# Then started working with Seymour Cray's first Masterpiece – the CDC 7600

This System had a large slow memory (256K words) and a small fast memory (65K words)

Similar memory architecture is being used today Intel's Knights Landing and Summit's nodes

# Seymour's Innovations in the CDC 7600

- **Segmented Functional Units**
  - The first hint of vector hardware
- **Multiple functional Units capable of parallel execution**
- **Small main memory with larger extended core memory**

# Any Idea what this is?

# My first laptop

# Cyber 203/205

# From paper by Neil Lincoln -CDC

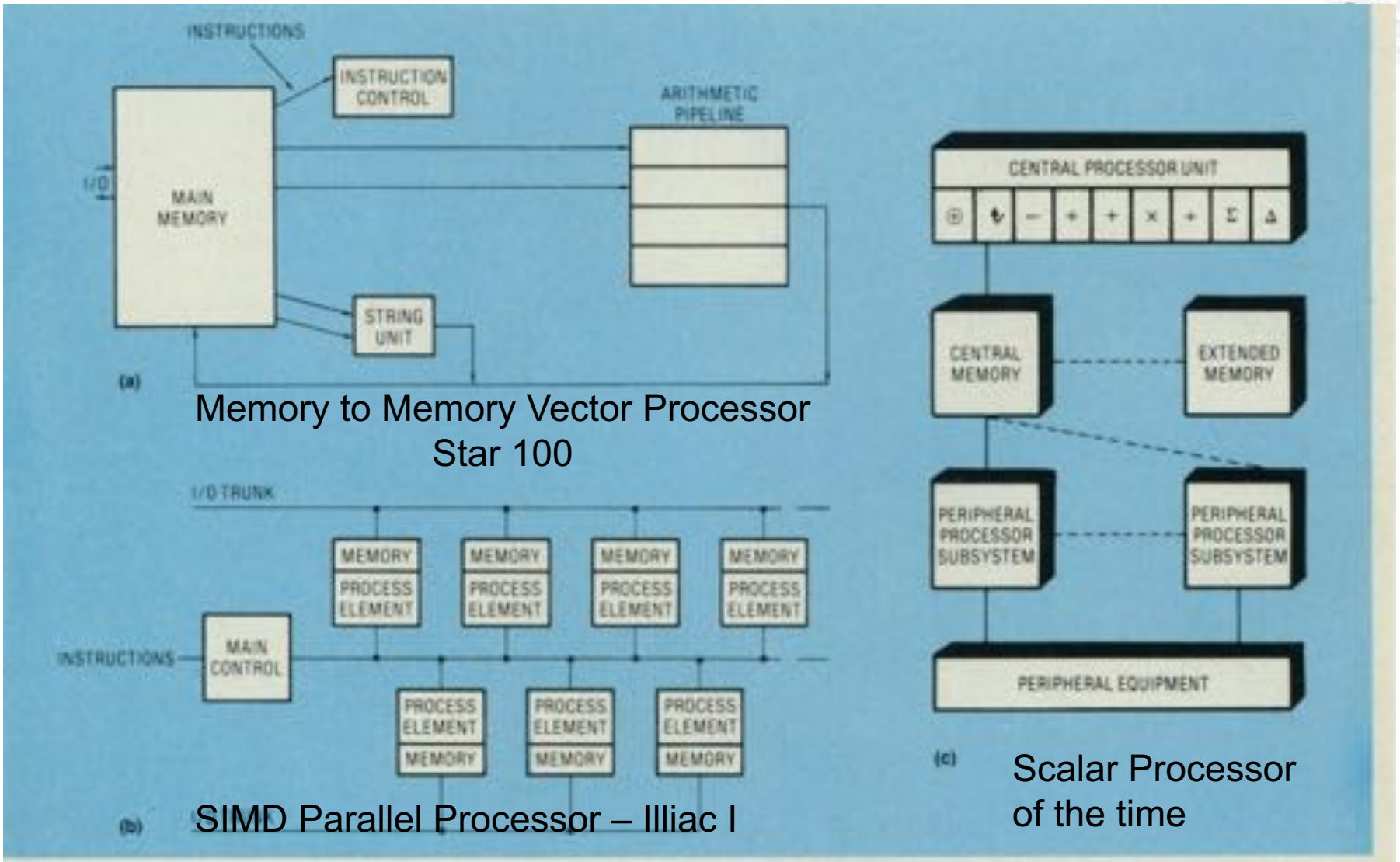
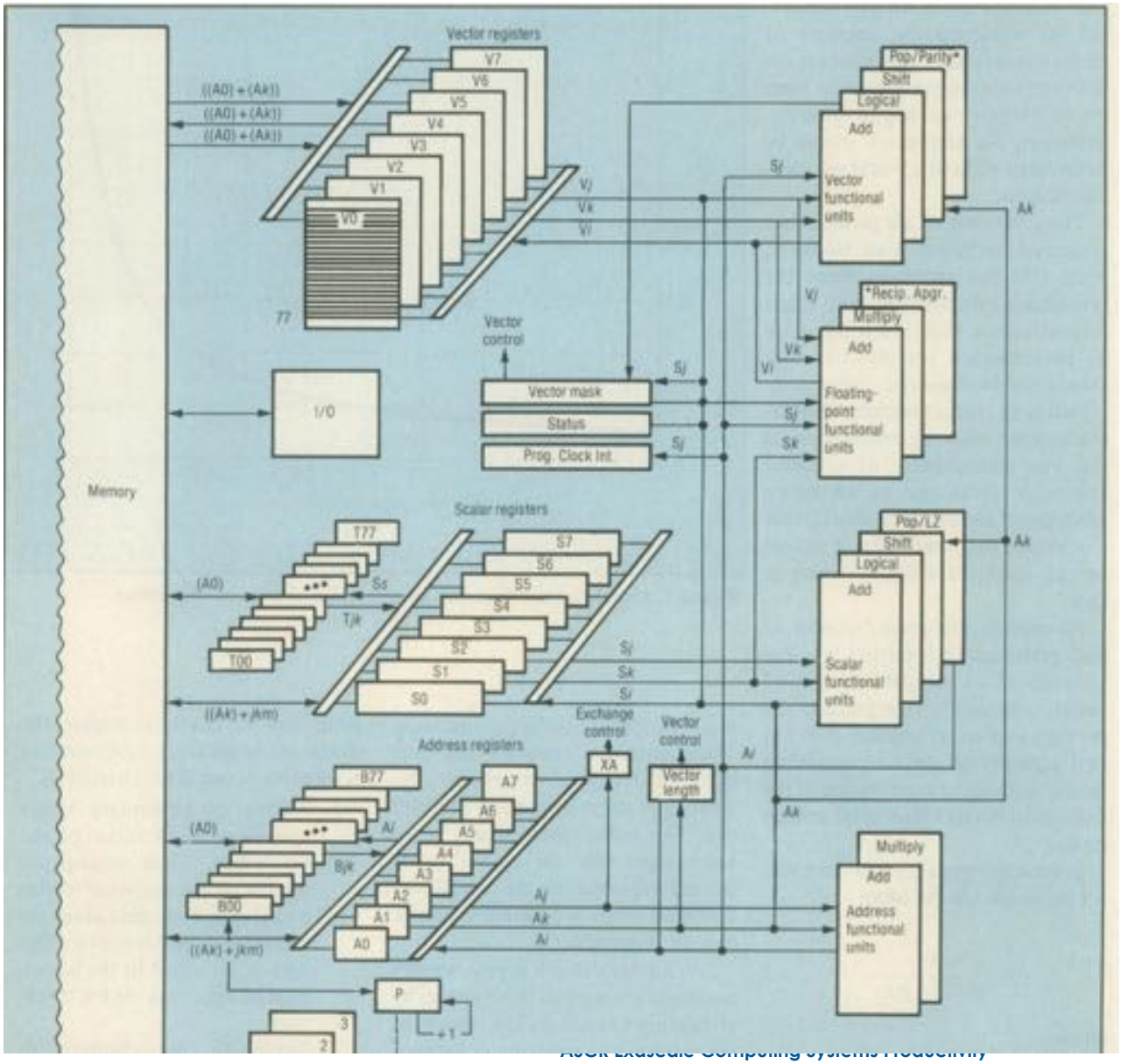



Figure 1. Summary schematics of supercomputer architectures for the Star-100, TI Advanced Scientific Computer, and the Cyber 200 (a); the Illiac/Pepe (b); and the 6000/7000/Cyber 170 (c).

# First Real Vector Machines

- **Memory to Memory Vector Processors**
  - Texas Instruments ASC
  - Star 100
  - Cyber 203/205
- **Register to Register Vector Processors**
  - Cray 1

# Seymour Cray's Innovative Register Architecture



# Really do not have vector instructions today

- **While Nvidia and Intel say they have vector instructions**
  - They are really like the Illiac IV
    - Intel's parallel width is 2,4,8 elements of 128,256,512 bit instructions
    - Nvidia's parallel width is 32 -64 bit elements, or 64 -32 bit elements
- **Having made this point, henceforth we will refer to them as vector instructions**

Result Rate = [Number of Operands/Number of Processing Elements]

# Cray 1A – First to Los Alamos in July 1976

# 1980s

## Golden Age of Cray

## Start of Shared Memory Systems

# Very active industry

- **Cray X-MP, Cray Y-MP, Cray 2**
  - Parallel shared memory vector systems
- **Japanese NEC, Fujitsu, Hitachi**
  - Parallel shared memory vector systems
- **Crayettes**
  - Convex, Supratek, Alliant
    - Parallel shared memory vector systems
- **ETA 10**
  - Last of the memory to memory vector processors

# NEC SX series

# 1990s

# Attack of the Killer Micros

- During this decade,
- More money was spent on Disposal Diapers than on Supercomputer

# Ncube

# Cray T3D/T3E

# Famous Jurassic Park Prop

SIMD Parallel with Scalar Host – Must transfer data to and from CM from host

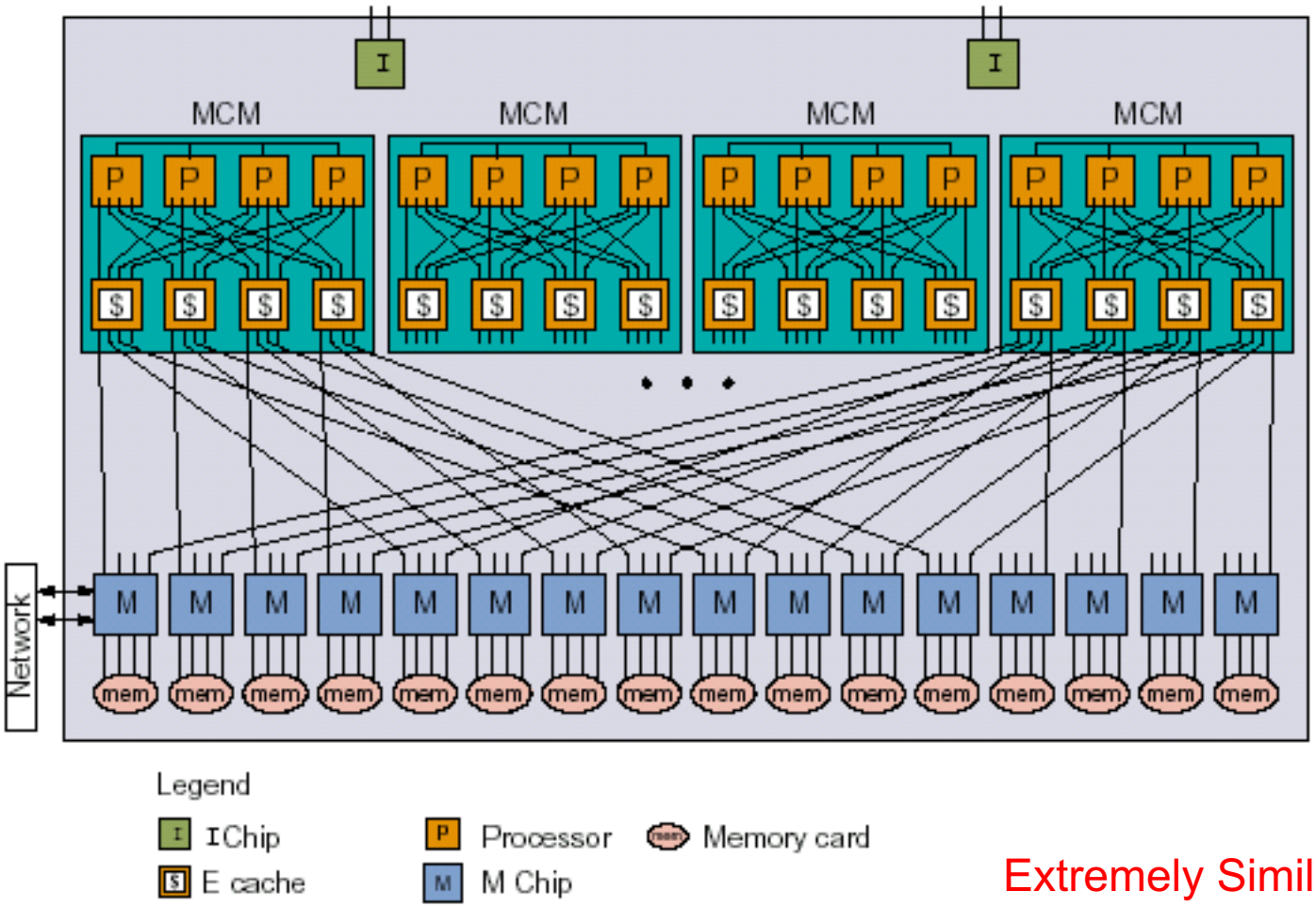
# All of these systems only had one core/node

- Performance was obtained by a constantly decreasing clock cycle (1995 – 2010)
- Application developers did not have to do anything to their programs to get additional performance
- Of course they did have to put in message passing to communicate between the distributed memory nodes

# 2000s

# Cray's last custom systems X1, X2

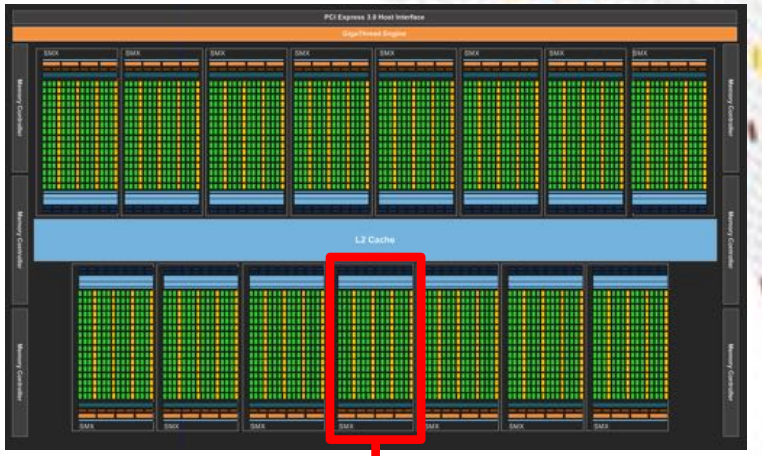
# Cray X1 – A preview of the future in 2003



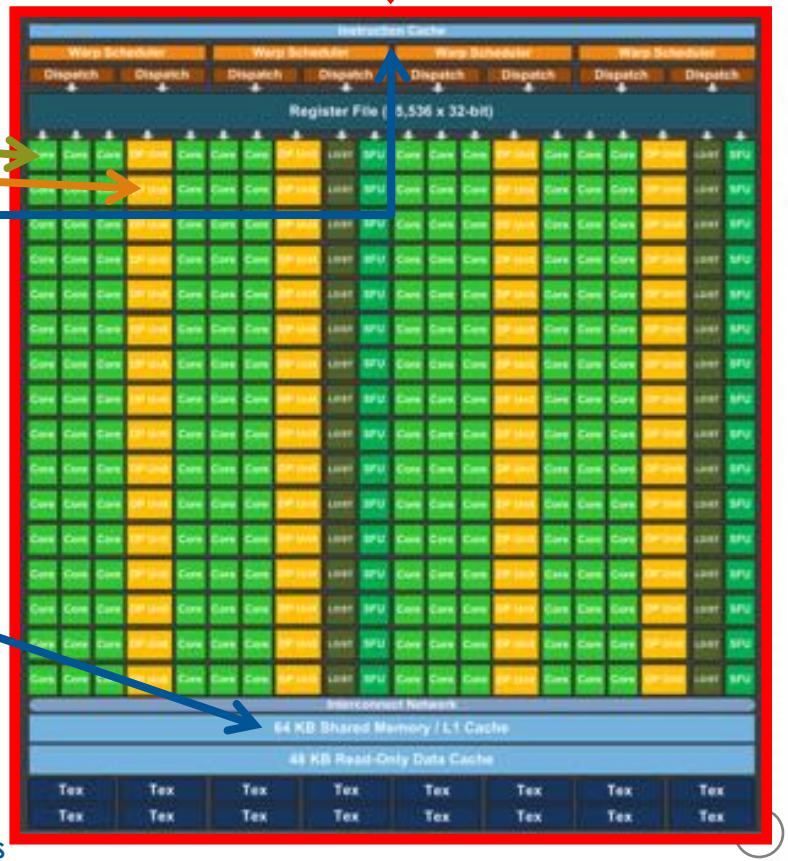

Extremely Similar to Nvidia GPU

# Nvidia K20X Kepler architecture

- **Global architecture**
  - a lot of compute cores
    - 2688 SP plus 896 DP; ratio 3:1
  - divided into 14 Streaming Multiprocessors
  - these operate independently




- **SMX architecture**
  - many cores
    - 64 SP
    - 32 DP
  - shared instruction stream; same ops
    - lockstep, SIMT execution of same ops
    - SMX acts like vector processor

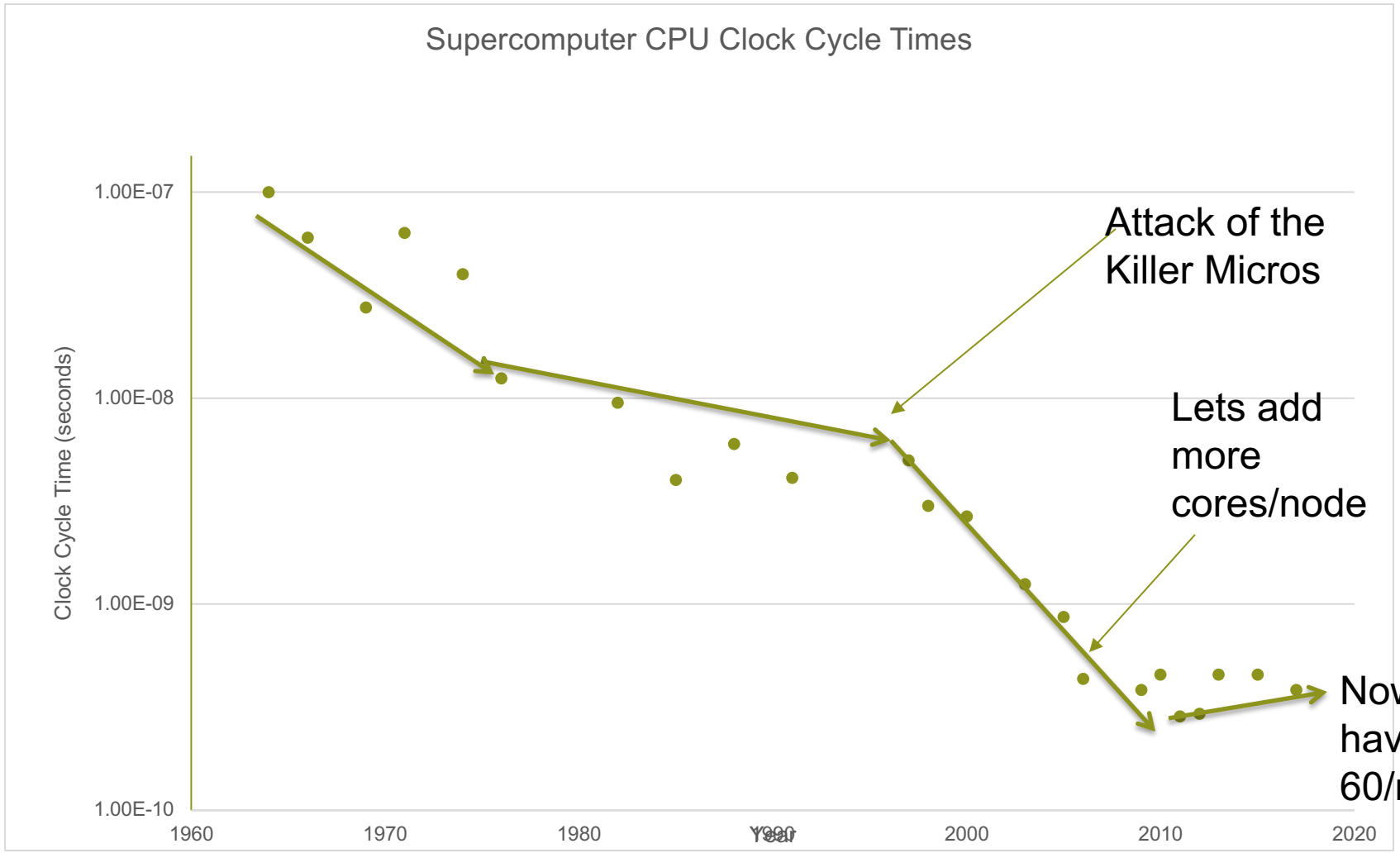



- **Memory hierarchy**
  - each core has private registers
    - fixed register file size
  - cores in an SM share a fast memory
    - 64KB, split between:
      - L1 cache and user-managed
  - all cores share large global memory
    - 6GB; also some specialist memory

# The System that shot down a satellite

# 2010s

# For the last ten years the clock cycle is staying staying the same or increasing



# Why can't the clock cycle keep decreasing

- **The shorter the clock cycle, the denser the circuity.**
  - Grace Hopper infamous nanosecond wire – 11.8 inches. The length an electronic signal can travel in one nanosecond.
  - Extremely difficult to cool the system so it will not melt.
- **Moore's Law**
  - postulates a reduction in the size of transistors leading to more and more transistors per chip at the cost-effective optimum
- **Dennard Scaling**
  - claims that the performance per watt of computing is growing exponentially at roughly the same rate

# Why can't the clock cycle keep decreasing

- **Dennard Scaling – performance increase with decrease in area**
  - Since around 2005–2007 Dennard scaling appears to have broken down. As of 2016, transistor counts in integrated circuits are still growing, but the resulting improvements in performance are more gradual than the speed-ups resulting from significant frequency increases. The primary reason cited for the breakdown is that at small sizes, current leakage poses greater challenges, and also causes the chip to heat up, which creates a threat of thermal runaway and therefore further increases energy costs.
  - The breakdown of Dennard scaling and resulting inability to increase clock frequencies significantly has caused most CPU manufacturers to focus on multicore processors as an alternative way to improve performance. An increased core count benefits many (though by no means all) workloads, but the increase in active switching elements from having multiple cores still results in increased overall power consumption and thus worsens CPU power dissipation issues. The end result is that only some fraction of an integrated circuit can actually be active at any given point in time without violating power constraints. The remaining (inactive) area is referred to as dark silicon.

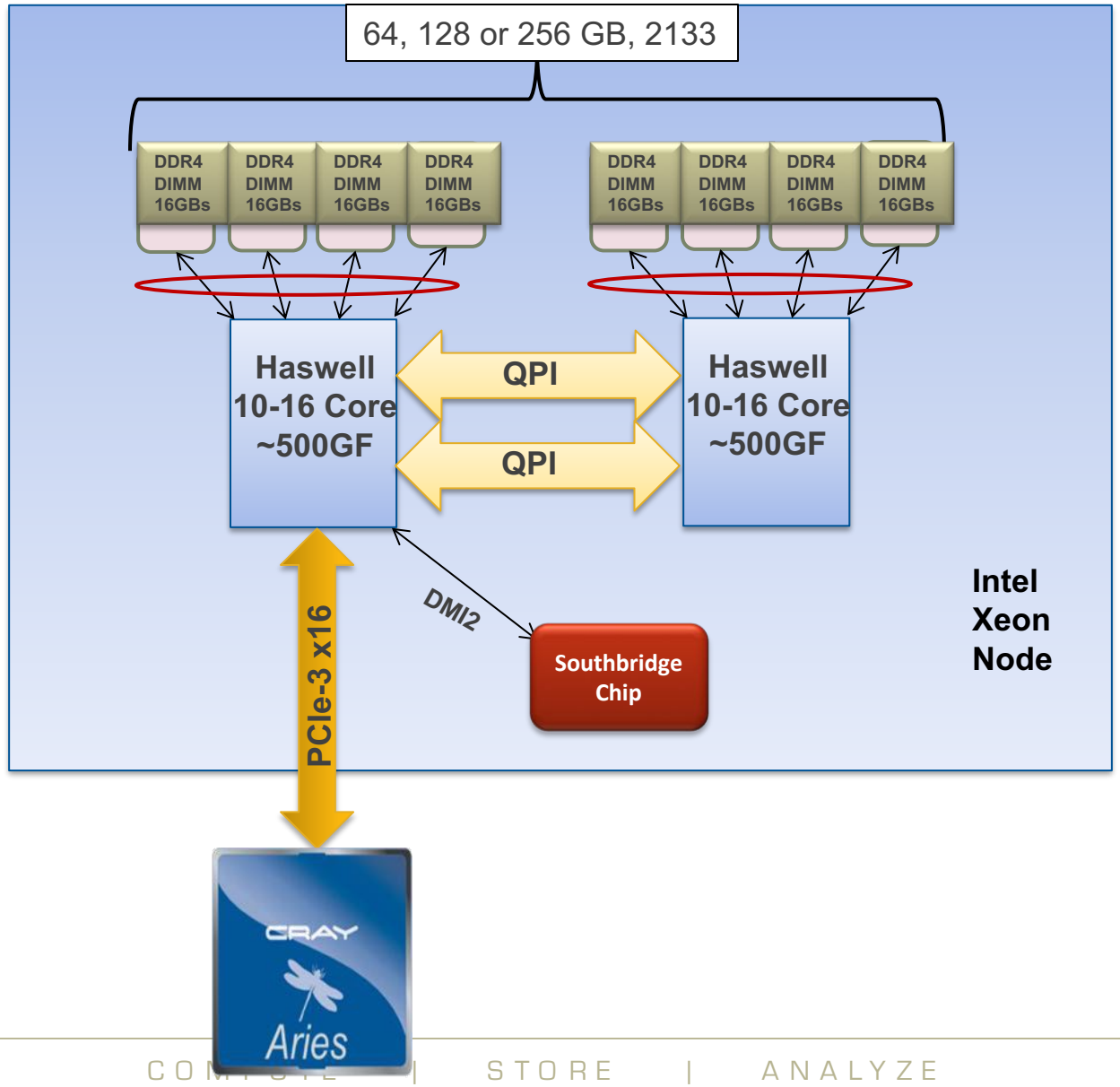
# Clock Cycle not giving us any improvement

- **So lets use chip real estate to add more cores and employ vector (SIMD parallel) instructions**
  - Cray Titan system and Summit using Nvidia processors
  - Intel KNC, KNL Trinity
  - Intel many-core chips – 8 – 28 cores/socket – Cray supplies two sockets on a node or one socket and multiple accelerators

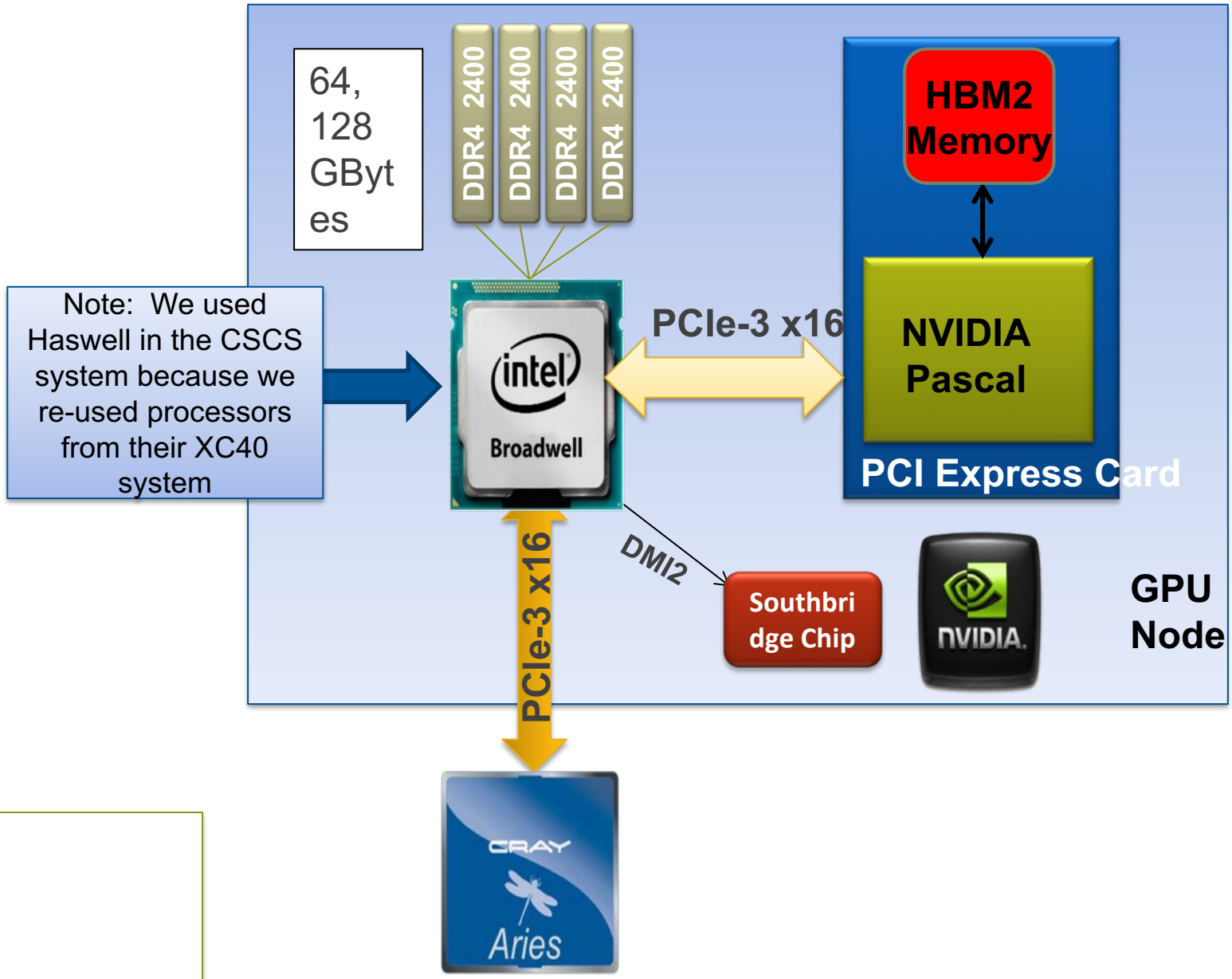
- **Software**
  - "Ask not what your compiler can do for you, Ask what you can do for your compiler"
    - No Silver Bullet
  - To get performance on todays and next generation systems the programmer must
    - Make sure applications vectorize
    - Make sure applications can utilize all the cores on the node
      - All MPI across all cores still working reasonably well
      - To use accelerator, must use either OpenACC or OpenMP 4.5

# Today

# Cray XC40 Compute Node

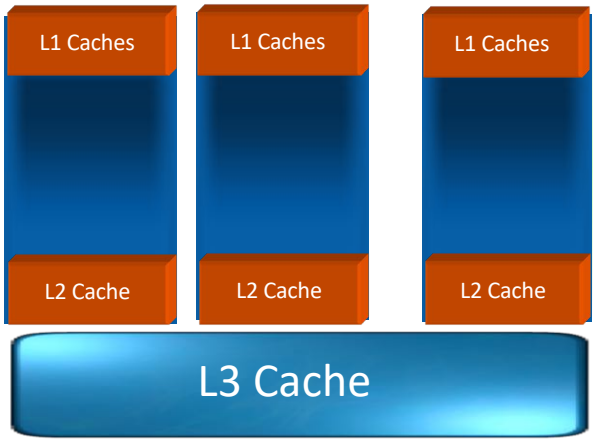


# Cray XC+ Pascal Node



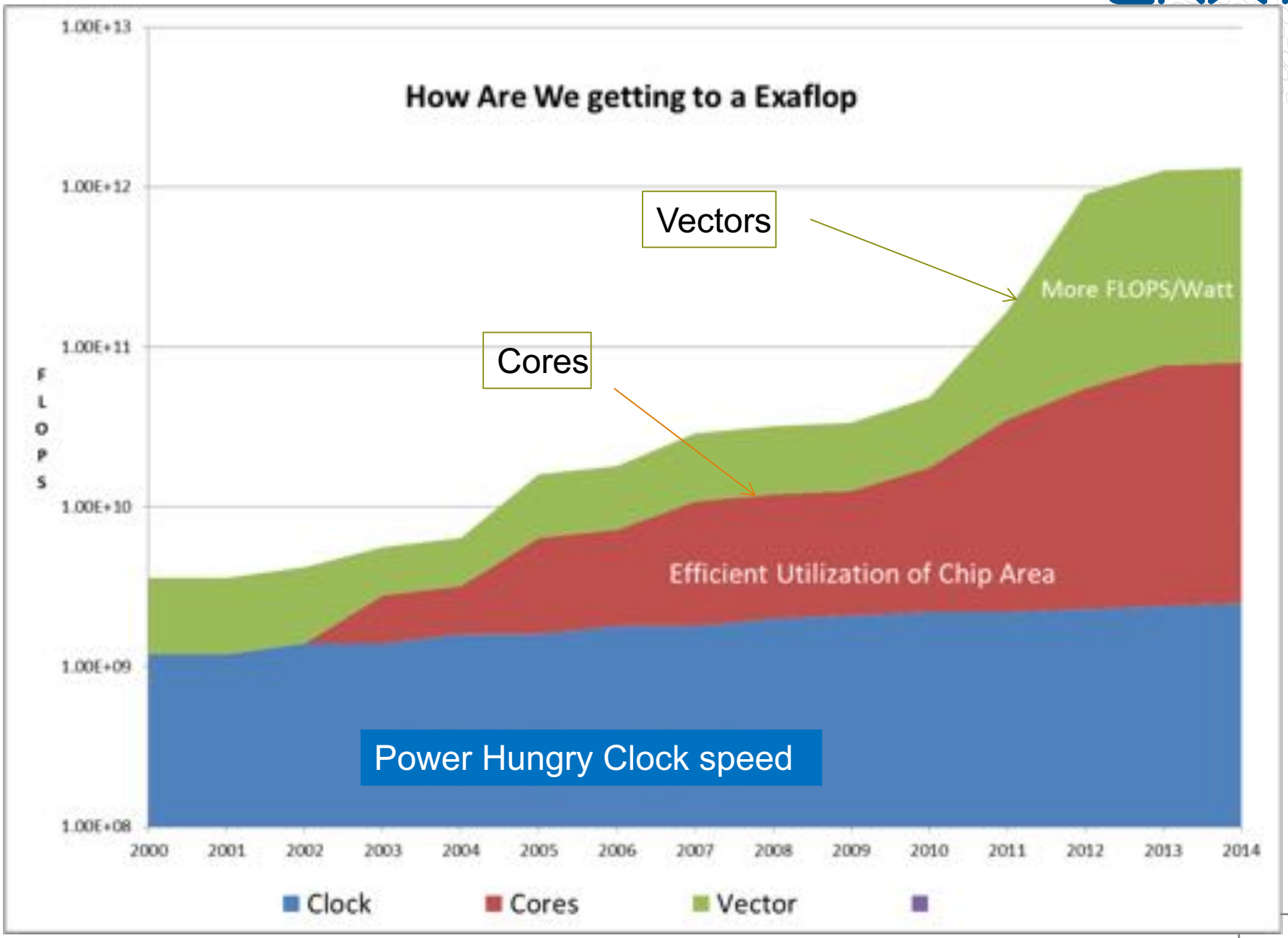
# Intel® Smart Cache Technology

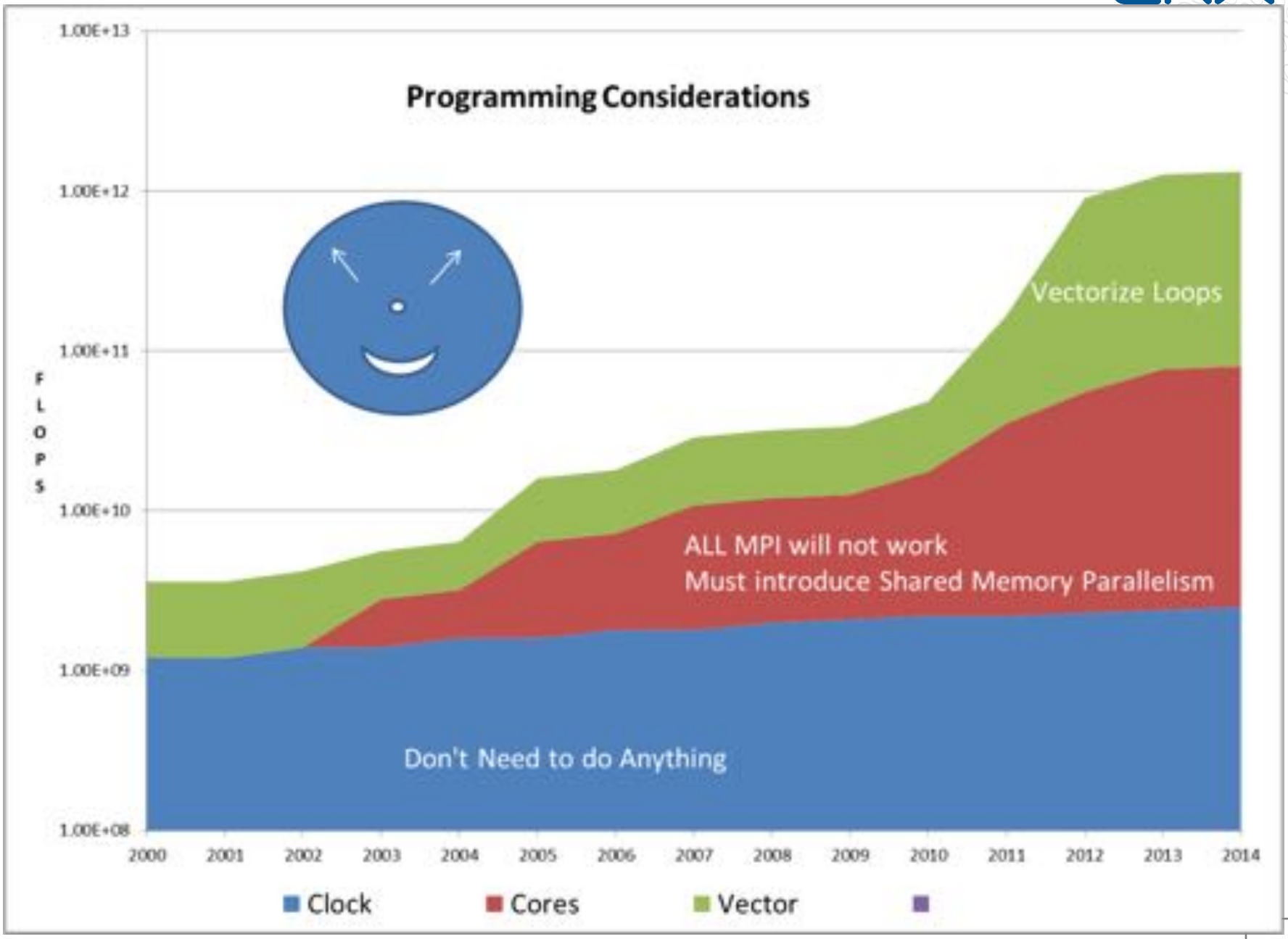
- **New 3-level Cache Hierarchy**
- **1st level caches**
  - 32kB Instruction cache
  - 32kB, 8-way Data Cache
- **2nd level cache**
  - New cache introduced in Intel® Core™ microarchitecture (Nehalem)
  - Unified (holds code and data)
  - 256KB per core (8-way)
  - Performance: Very low latency, 10 cycle load-to-use
- **3rd Level cache**
  - Shared across all cores
  - 2.5MB/Core (16-ways)
  - Latency depends on frequency ratio between core and UnCore
  - Inclusive cache policy for best performance
  - Address residing in L1/L2 must be present in 3rd level cache




* Other names and brands may be claimed as the property of others. Slide provided by Intel ®


How Are We getting to a Exaflop
1.00E+13
1.00E+12
1.00E+11
1.00E+10
1.00E+09
1.00E+08
FLOPS
Vectors
More FLOPS/Watt
Cores
Efficient Utilization of Chip Area
Power Hungry Clock speed
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Clock
Cores
Vector


Programming Considerations
1.00E+13
1.00E+12
1.00E+11
1.00E+10
1.00E+09
1.00E+08
FLOPS
Vectorize Loops
ALL MPI will not work
Must introduce Shared Memory Parallelism
Don't Need to do Anything
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Clock
Cores
Vector

# Now lets talk about Programming

- When the clock cycle gets faster the program runs faster without doing anything to your application
- When there is a vector unit the compiler has to vectorize the principal loops in the program to take advantage of the vector unit
- When there are more than one processor
  - If they are on the same node you have to use either OpenMP threads to orchestrate them to work on a single application of use MPI when they are on separate nodes
  - When they are on separate nodes, you have to use use MPI when they are on separate nodes
- So you have to do something to take advantage of the architectural improvement

# Shared Memory Parallelization

- **The most difficult task to parallelize a loop for shared memory parallelization is to scope the variables accessed within the loop**
  - What variable are **shared** – that is one address space to accessed by all processors (or hardware threads) on the node
  - What variables are **private** – that is an address space that is accessible by a single processor (or hardware thread)

# Vectorization

- **ABSOLUTELY** necessary for porting applications to accelerators
- Here the users work with the compiler to re-work important loops so the compiler generates vector instructions for the hardware

# If it doesn't Vectorize – fix it

Examine sampling exp with line # Table 3

```
A |||||=========================================
  ||  10.8% |   376.9 |    -- |    -- |riemann_
  3|        |         |       |       | riemann.f90
  ||||-------------------------------------------------------------------
  4|||   1.4% |    47.4 |  32.6 | 41.0% |line.77
  4|||   3.9% |   135.8 |  28.2 | 17.3% |line.78
```

```
63.  + 1----< do l = lmin, lmax
64.  + 1 2--<   do n = 1, 12
65.    1 2       pmold(l) = pmid(l)
66.    1 2       wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.    1 2       wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.    1 2       wlft (l) = clft(l) * sqrt(wlft(l))
69.    1 2       wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.    1 2       zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.    1 2       zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.    1 2       zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.    1 2       zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
74.    1 2       umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.    1 2       umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.    1 2       pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l) - zlft(l))
77.    1 2       pmid (l) = max(smallp,pmid(l))
78.    1 2       if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.    1 2-->   enddo
80.    1----> enddo
```

```
ftn-6254 ftn: VECTOR RIEMANN, File = riemann.f90, Line = 64
  A loop starting at line 64 was not vectorized because a recurrence was found
  on "pmid" at line 77.
```

# If it doesn't Vectorize – fix it

```
63.  + 1----< do l = lmin, lmax
64.  + 1 2--<   do n = 1, 12
65.    1 2         pmold(l) = pmid(l)
66.    1 2         wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.    1 2         wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.    1 2         wlft (l) = clft(l) * sqrt(wlft(l))
69.    1 2         wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.    1 2         zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.    1 2         zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.    1 2         zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.    1 2         zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) –
                                                                    prgh(l)))
74.    1 2         umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.    1 2         umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.    1 2         pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) /
                                                                    (zrgh(l) - zlft(l))
77.    1 2         pmid (l) = max(smallp,pmid(l))
78.    1 2         if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.    1 2-->    enddo
80.    1----> enddo
```

```
62.    A-----<>  converged =.F.
63.  + 1------<  do n = 1, 12
64.    1 Vr2--<    do l = lmin, lmax
65.    1 Vr2        if(.not.converged(l))then
66.    1 Vr2         pmold(l) = pmid(l)
67.    1 Vr2         wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
68.    1 Vr2         wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
69.    1 Vr2         wlft (l) = clft(l) * sqrt(wlft(l))
70.    1 Vr2         wrgh (l) = crgh(l) * sqrt(wrgh(l))
71.    1 Vr2         zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
72.    1 Vr2         zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
73.    1 Vr2         zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
74.    1 Vr2         zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
75.    1 Vr2         umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
76.    1 Vr2         umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
77.    1 Vr2         pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / &
78.    1 Vr2                  (zrgh(l)-zlft(l))
79.    1 Vr2         pmid (l) = max(smallp,pmid(l))
80.    1 Vr2         if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) then
81.    1 Vr2         converged(l) = .T.
82.    1 Vr2         endif
83.    1 Vr2        endif
84.    1 Vr2-->    enddo
85.  + 1          if(all(converged(lmin:lmax)))exit
86.    1------>  enddo
```

# Cache Utilization

- Latency to memory is significantly longer than required to keep functional units busy; therefore, caches must be effectively utilized.
- Accelerators have small caches and register sets. Spilling to memory significantly degrades performance

# Utilization of Multiple Memories

- Even though the system can automagically manage memory movement between host and accelerator, excessive data movement can kill performance

"Software is getting slower more rapidly than hardware becomes faster"

–Niklaus Wirth, 1995


**John Levesque**

levesque@cray.com