Parallel Computing

David Lawrence, JLab

Mar. 16, 2016

Types of “Parallel” Computing

- Nomenclature
  - Parallel vs. concurrent
  - Bit-level vs. data level
- Multi-threaded
- Multi-process
- Distributed
- Grid
- SIMD
- GPU/GPGPU
- MIC

SIMD = Single Instruction Multiple Data

- Special registers on CPU where multiple numbers can be packed and operated on simultaneously
- Also known as "vectorization"
  - *gcc: "…vectorization is enabled by the flag -ftree-vectorize and by default at -O3"*
- CPU vendors have their own implementations and evolutions
  (e.g. Intel has …)
  - MMX (1997, Pentium 5) 64bit
  - SSE (1999) – SSE4(2006) 128 bit
  - AVX (2008) 256 bit
  - MIC/VPU 512 bit

SIMD = Single Instruction Multiple Data

- Special registers on CPU where multiple numbers

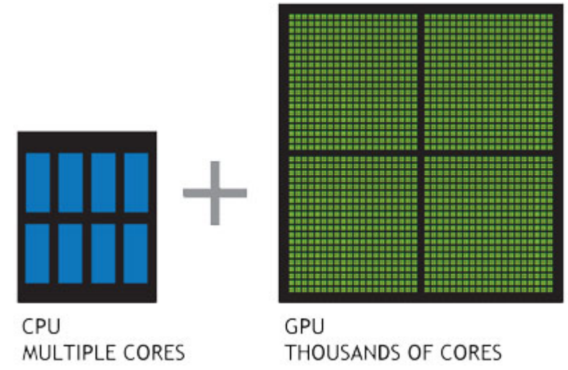
```
// Multiply a 5x1 matrix by its transpose
inline DMatrix5x5 MultiplyTranspose(const DMatrix5x1 &m1){
  ALIGNED_16_BLOCK_WITH_PTR(__m128d, 5, p)
  __m128d &b1=p[0];
  __m128d &b2=p[1];
  __m128d &b3=p[2];
  __m128d &b4=p[3];
  __m128d &b5=p[4];
  b1=_mm_set1_pd(m1(0));
  b2=_mm_set1_pd(m1(1));
  b3=_mm_set1_pd(m1(2));
  b4=_mm_set1_pd(m1(3));
  b5=_mm_set1_pd(m1(4));
  return DMatrix5x5(_mm_mul_pd(m1.GetV(0),b1),_mm_mul_pd(m1.GetV(0),b2),
            _mm_mul_pd(m1.GetV(0),b3),_mm_mul_pd(m1.GetV(0),b4),
            _mm_mul_pd(m1.GetV(0),b5),
            _mm_mul_pd(m1.GetV(1),b1),_mm_mul_pd(m1.GetV(1),b2),
            _mm_mul_pd(m1.GetV(1),b3),_mm_mul_pd(m1.GetV(1),b4),
            _mm_mul_pd(m1.GetV(1),b5),
            _mm_mul_pd(m1.GetV(2),b1),_mm_mul_pd(m1.GetV(2),b2),
            _mm_mul_pd(m1.GetV(2),b3),_mm_mul_pd(m1.GetV(2),b4),
            _mm_mul_pd(m1.GetV(2),b5));
}
```

  - MIC/VPU 512 bit

GPU = Graphics Processing Unit

- Driven by gaming industry where high frame rates of complex environments was required
- Many cores (few x$10^{1\text{-}3}$) used in lockstep to calculate same algorithm with different inputs
- Programmed via special API
  - CUDA, OpenCL, OpenGL

GPU – Example CUDA code



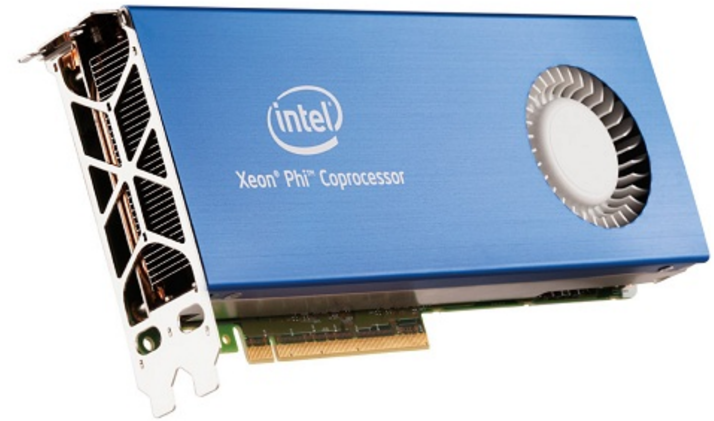
Standard C Code

```c
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;


// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

C with CUDA extensions

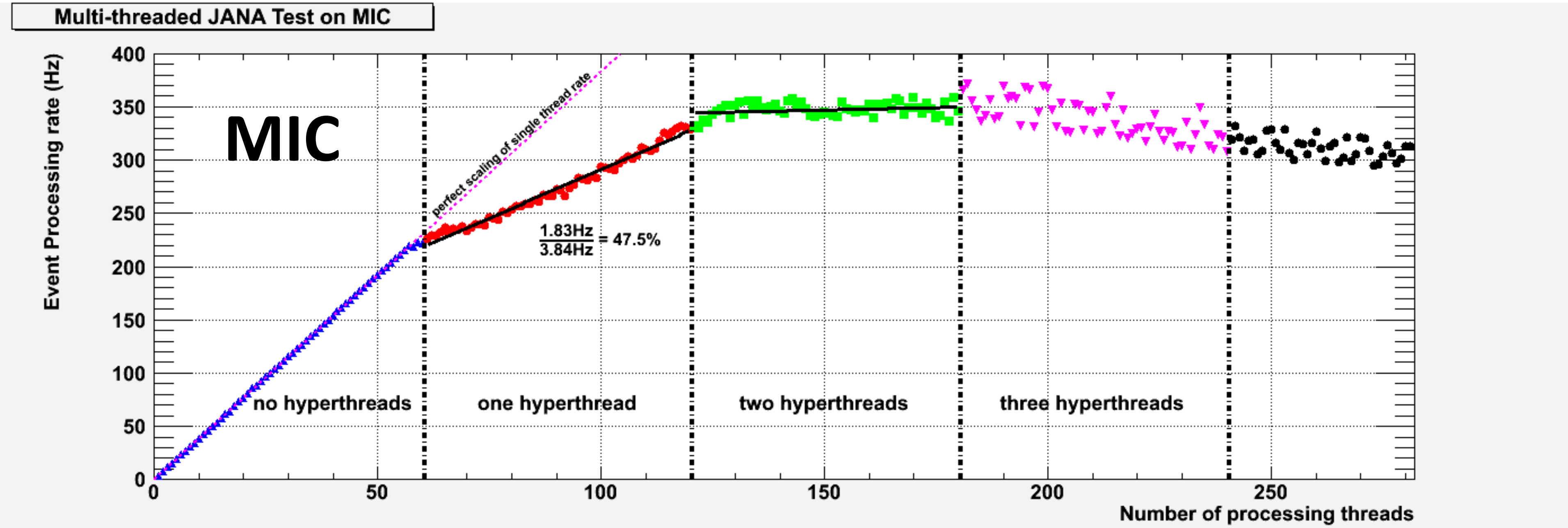
```c
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);
```

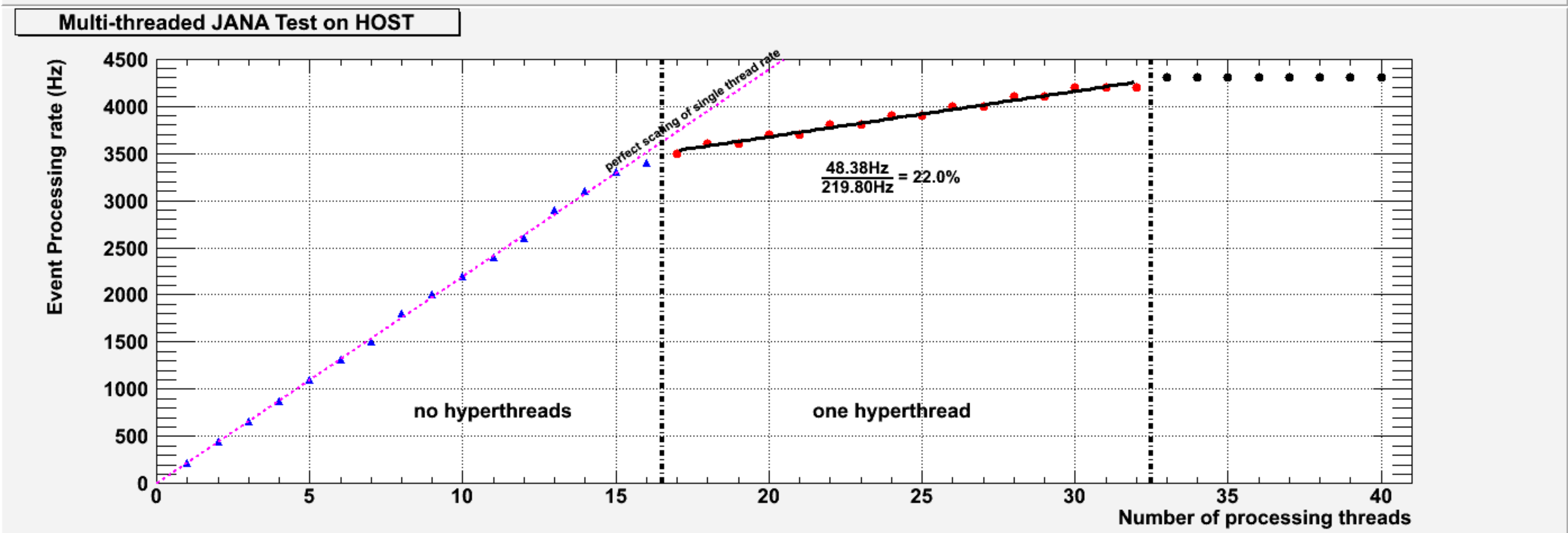
MIC = Many Integrated Cores



- Xeon Phi = Intel’s MIC system
  - 60 cores, 1GHz on a PCIe x16 card
  - 512 bit wide vectors
  - Original project: Larrabee
    - Attempt to make GPU from older x86 design
- Linux variant runs on MIC card independent of host OS
  - MIC system is based on 2.4 Linux kernel
  - File system not automatically shared
    - MIC cards can be configured to mount host’s filesystem via NFS
- Must use intel-provided cross-compiler to build executables
  - Could not build sim-recon because ROOT was needed
  - Could not build ROOT because libX11-devel was needed

Performance using JANA TestSpeed plugin





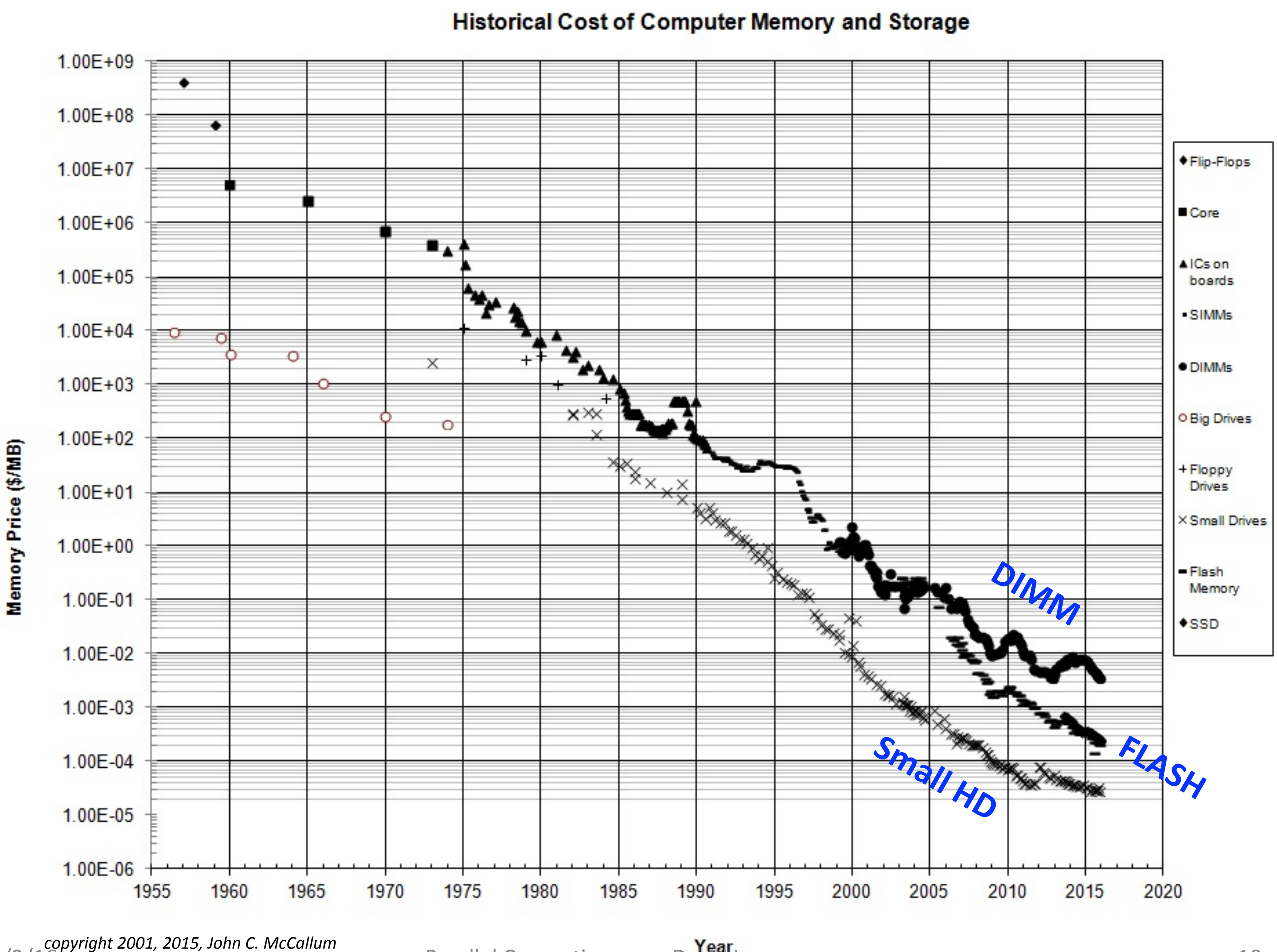
Multi-Threading vs. Multi-Process

Multi-threaded

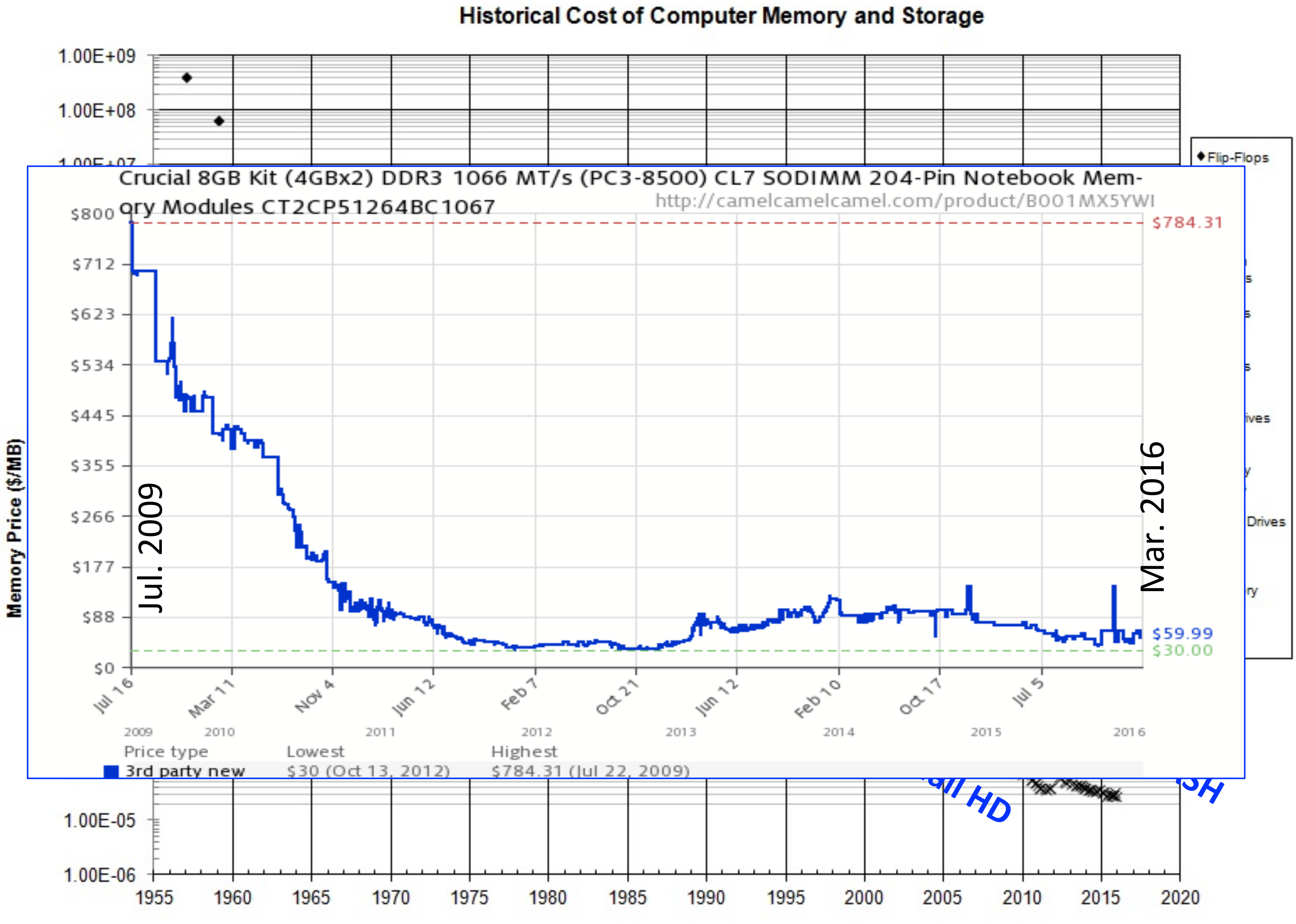
- Uses same memory space
- Uses less RAM
- Minimizes disk head thrashing
- Less overhead in process management

Multi-process

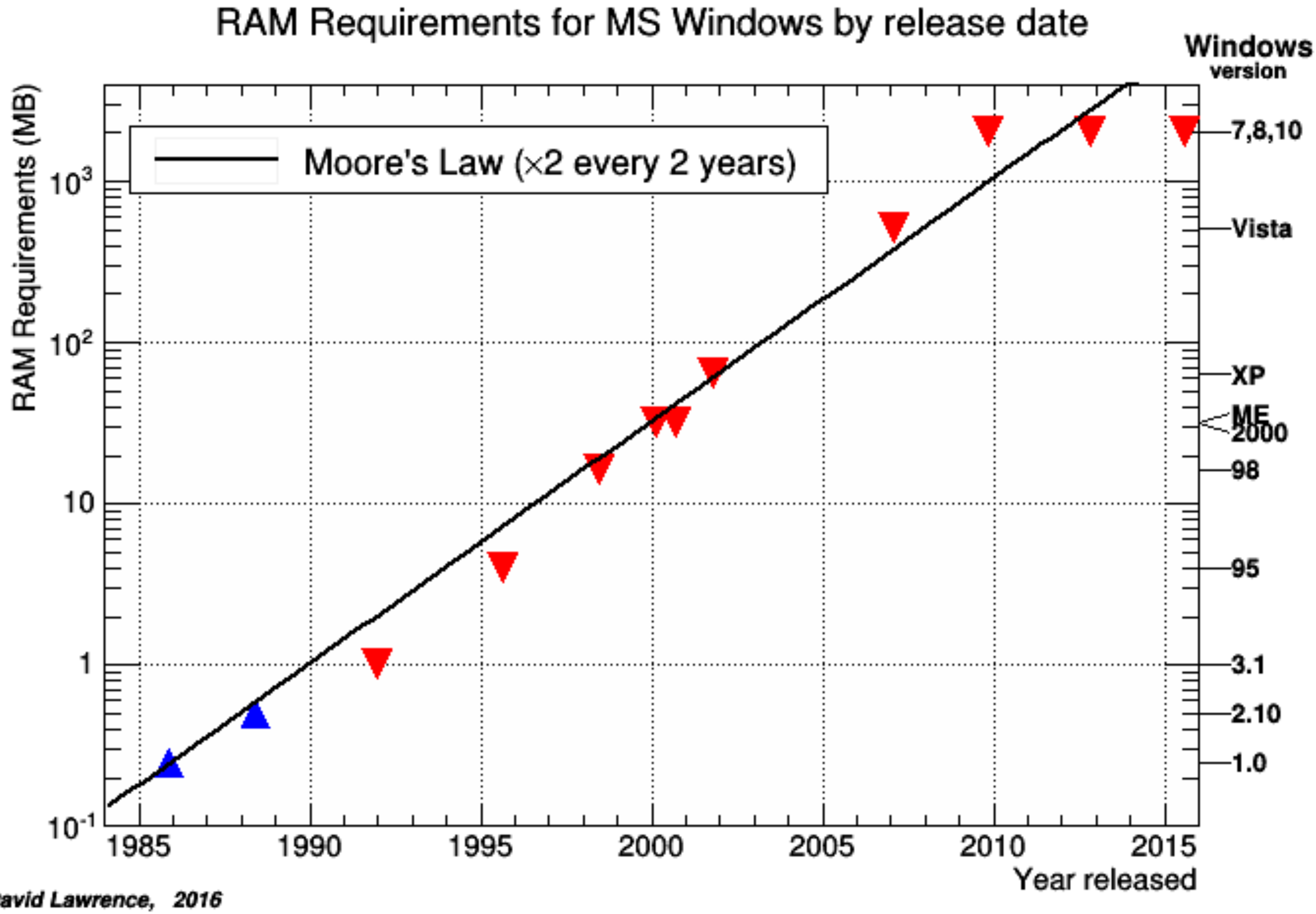
- Uses special shared memory segments or other message passing protocol
- Independent program contexts
- Simpler to debug single thread program crashes
- Less expertise required by novice programmers



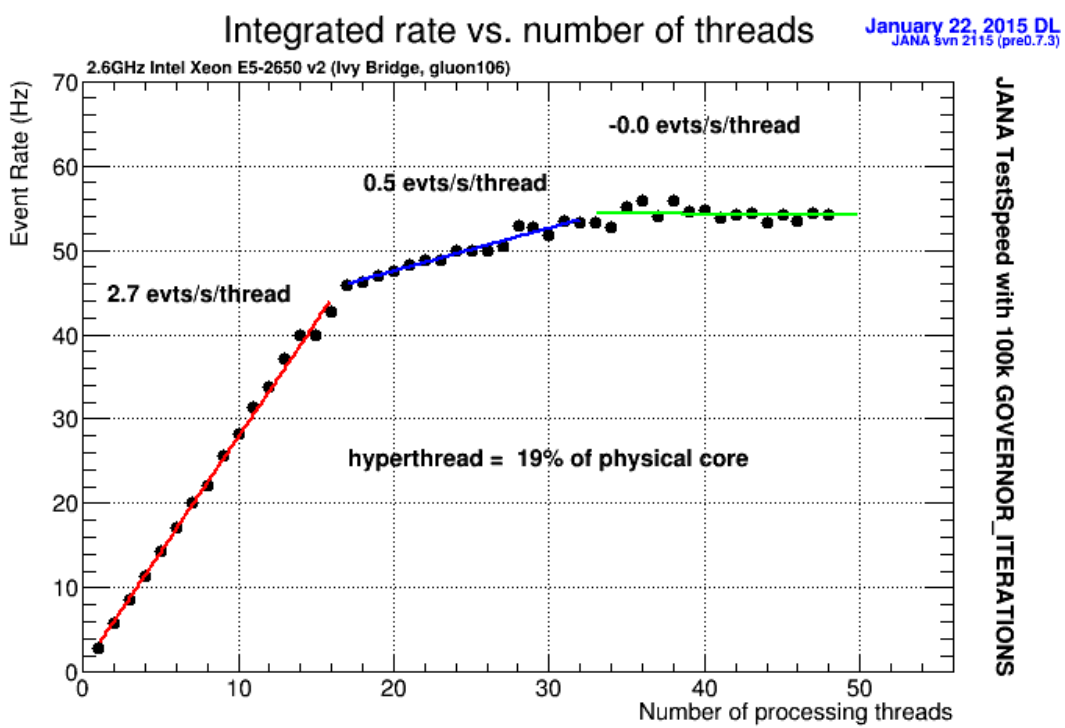
copyright 2001, 2015, John C. McCallum



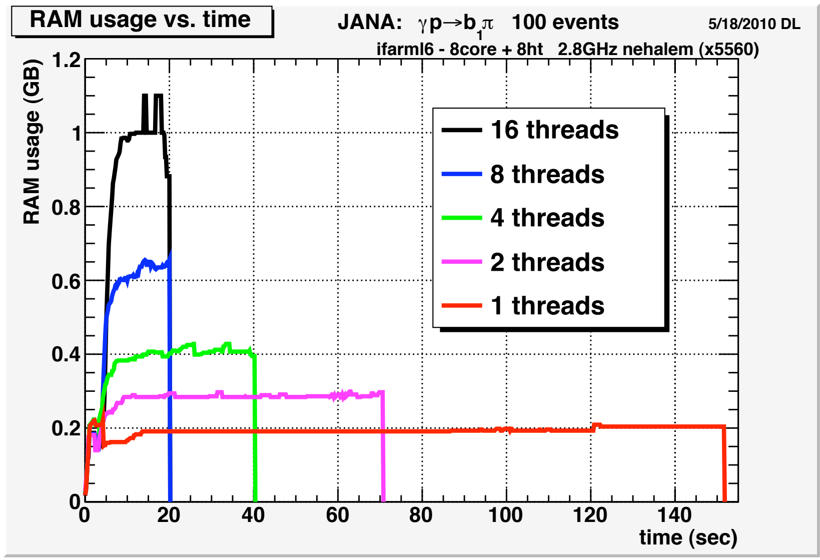
copyright 2001, 2015, John C. McCallum

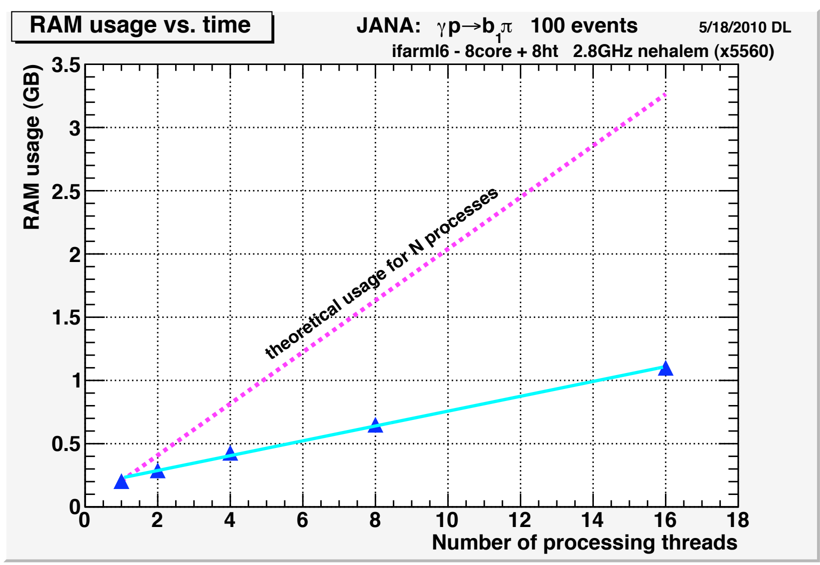

RAM Requirements for MS Windows by release date
Windows version
Moore's Law (×2 every 2 years)
RAM Requirements (MB)
10^3
10^2
10
1
10^-1
1985
1990
1995
2000
2005
2010
2015
Year released
7,8,10
Vista
XP
ME
2000
98
95
3.1
2.10
1.0
David Lawrence, 2016

JANA rate scaling for CPU intensive task



JANA RAM usage



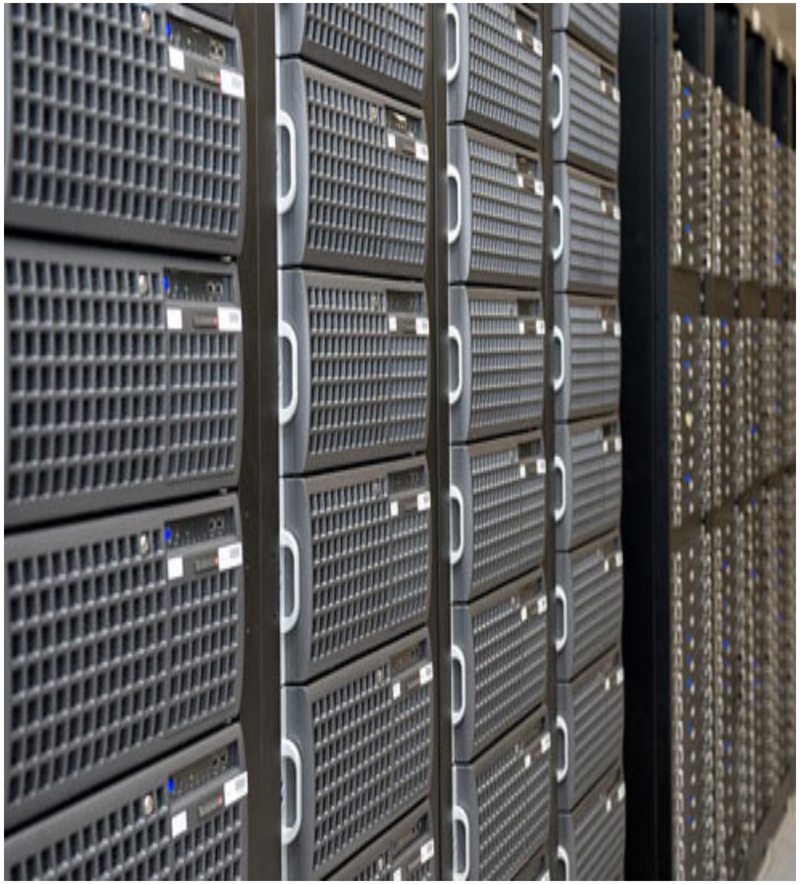


Multi-threaded GB/thread is about 1/3 that of multi-process

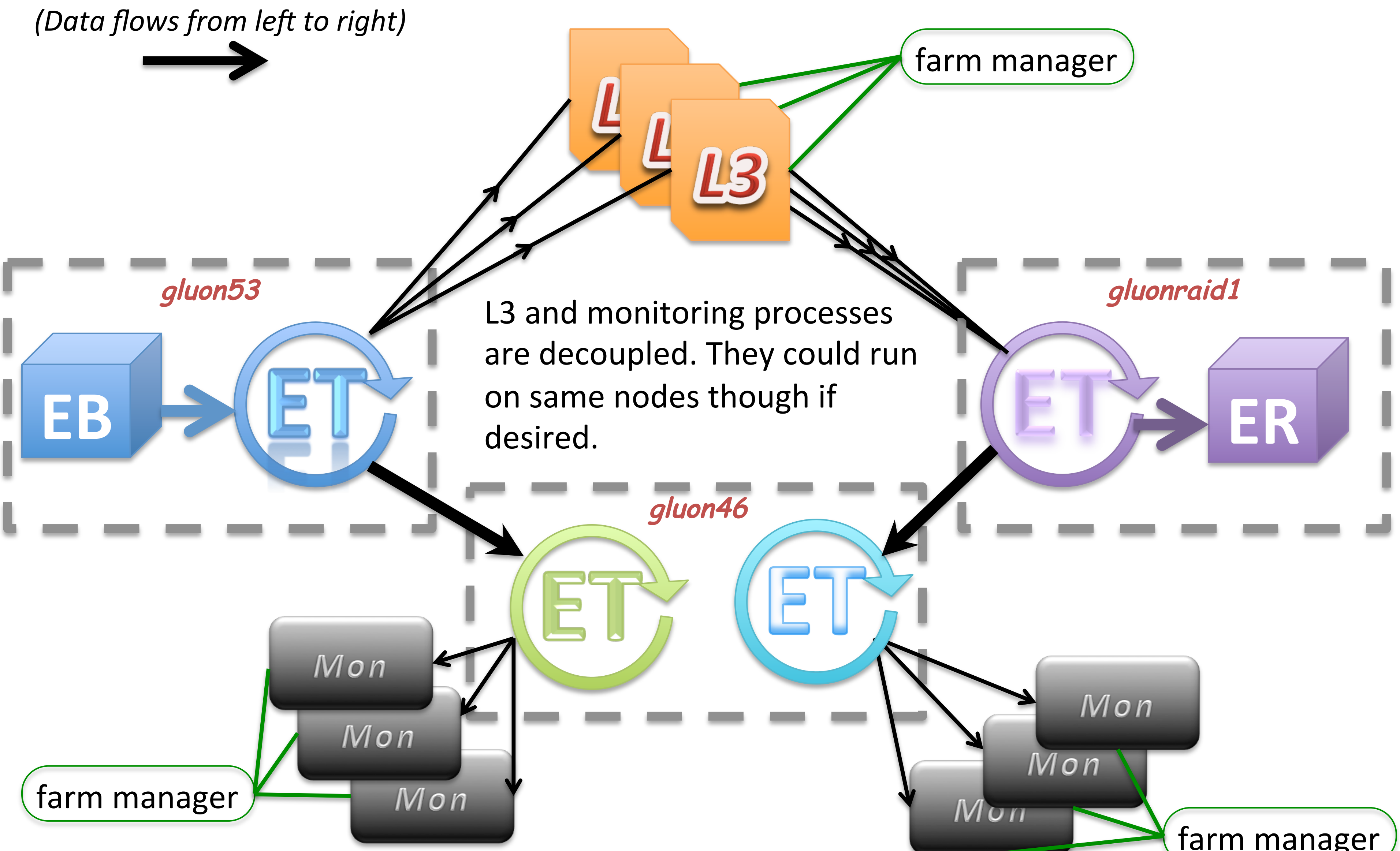
Distributed Computing

(Let's just call it "farms")

- large cluster of computers, housed in same location, and connected via fast LAN
- jobs run independently on single node *(… or maybe not …)*
- focuses significant compute power to dedicated job
- "clouds" tend to be made up of multiples of these connected via WAN



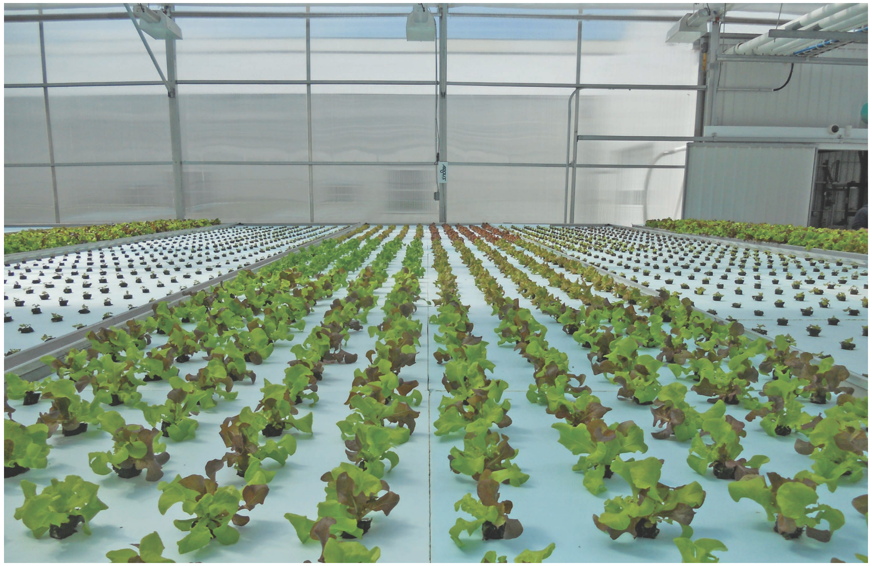
Hall-D L3 and monitoring architecture



Farms in the Future

Farms will play a role in the future due to power supply and dissipation

(i.e. You can't pack too many teraflops into a small volume without burning everything up!)



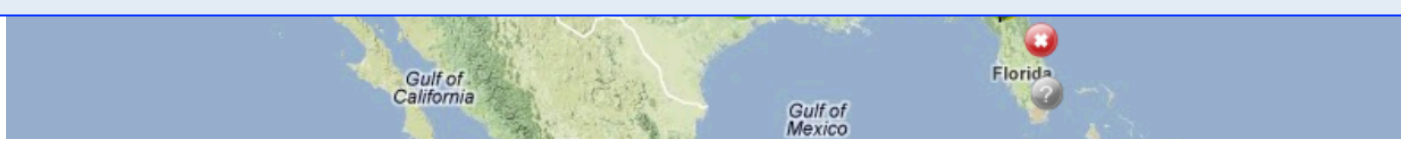
Open Science GRID

- US-based federation of compute infrastructures for research and academic communities

Open Science GRID

Characteristics of OSG Jobs

- **The application is a Linux application for the x86 or x86_64 architecture.**
- **The application is single- or multi-threaded but does not require message passing.**
- **The application has a small runtime between 1 and 24 hours.**
- **The application can handle being unexpectedly killed and restarted.**
- **The application is built from software that does not require contact to licensing servers.**
- **The scientific problem can be described as a workflow consisting of jobs of such kind.**
- **The scientific problem requires running a very large number of small jobs rather than a few large jobs.**



What do big LHC Experiments Do?

- CMS
  - Initially developed reconstruction algorithms as single-threaded
  - Developed framework that can identify which algorithms can be run in parallel giving sub-event level parallelism
- ATLAS
  - Single threaded using GAUDI
  - Developing GAUDIHive for parallelism at event, algorithm, and sub-algorithm levels

What do RHIC Experiments Do?

- Who knows?
  (Online documentation is quite old)

What to plan for? (IMHO)

- Parallelism will play a role. Effective systems will likely take advantage of multiple technologies
  - Tough to tell what emerging technologies will stick around. *(Be careful, but not overly cautious!)*
  - Spend time on the API. It will allow parts of the software to be replaced later without rewriting everything
  - Stay away from requiring commercial products (libraries). Stick with Open Source
  - Commit resources to software R&D. Not just at the beginning, but continuing…

Additional Thoughts (IMLTHO)

- The accessibility of a computing technology should be inversely proportional to the developer base *(i.e. if you commit everyone to an obscure or expensive technology, a lot less people we be able to contribute)*
- We have an obligation to next generation students/scientists to engage them in technologies that carry them at least a little way into the future as opposed to holding them in our past

Backups

Farms in the Future

- Far
  fut
  an
- Pa