

Heterogeneous Parallelism at Microsoft

Herb Sutter

Welcome to the jungle

The free lunch
is so over

1975-2005

Put a **computer**
on every desk, in
every home, in
every pocket.

2005-2011

Put a **parallel
supercomputer**
on every desk, in
every home, in
every pocket.

2011-201x

Put a **heterogeneous
supercomputer**
on every desk,
in every home,
in every pocket.



mainstream, *df.*:



**commercially available
to millions**

**commercially affordable
for millions**

**commercially programmable
by millions**

*note: everything in “the mainstream”
starts out in “the exotic”*

GUIs ✓ objects ✓ parallelism ✓



mainstream trends ~~X~~



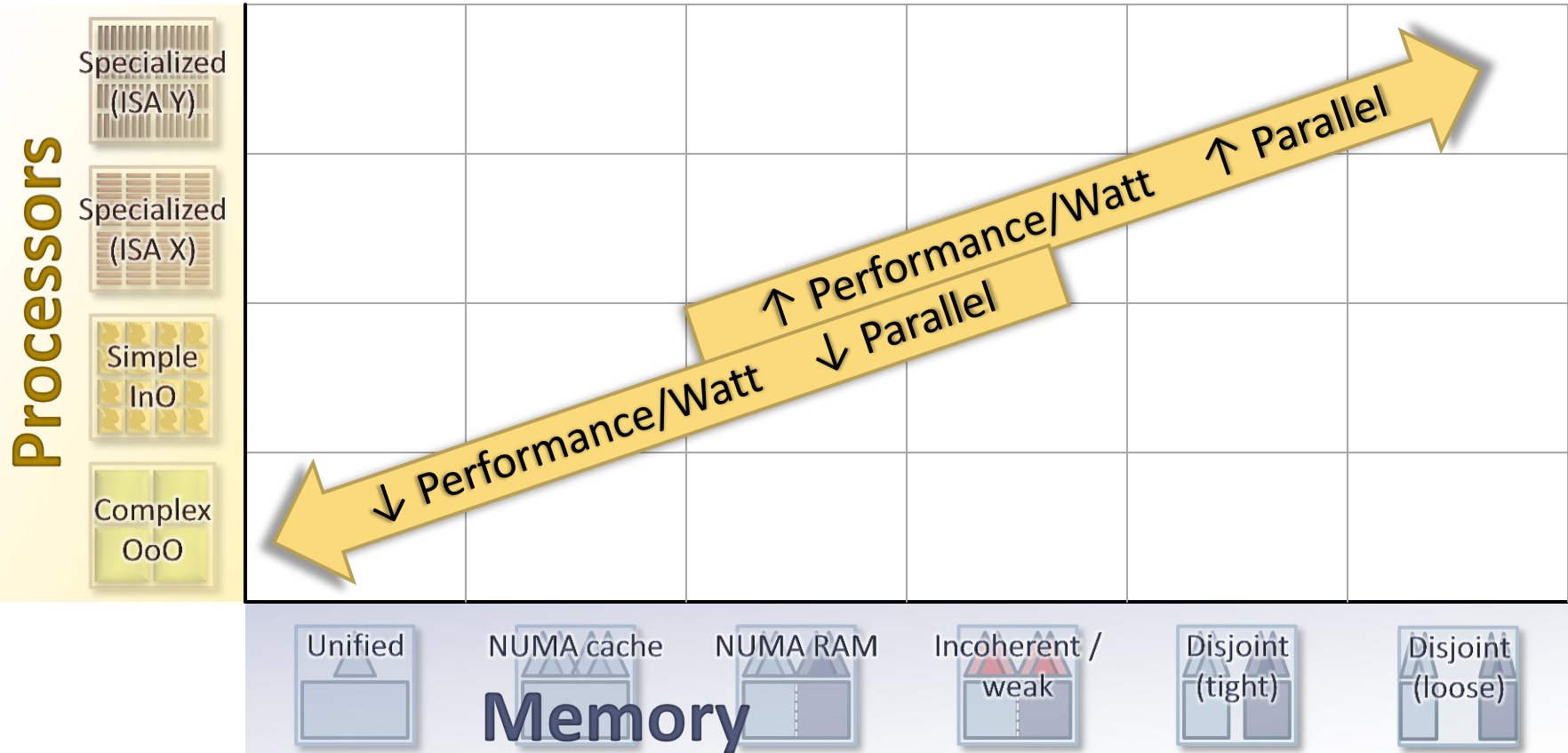
multicore CPUs

GPGPU

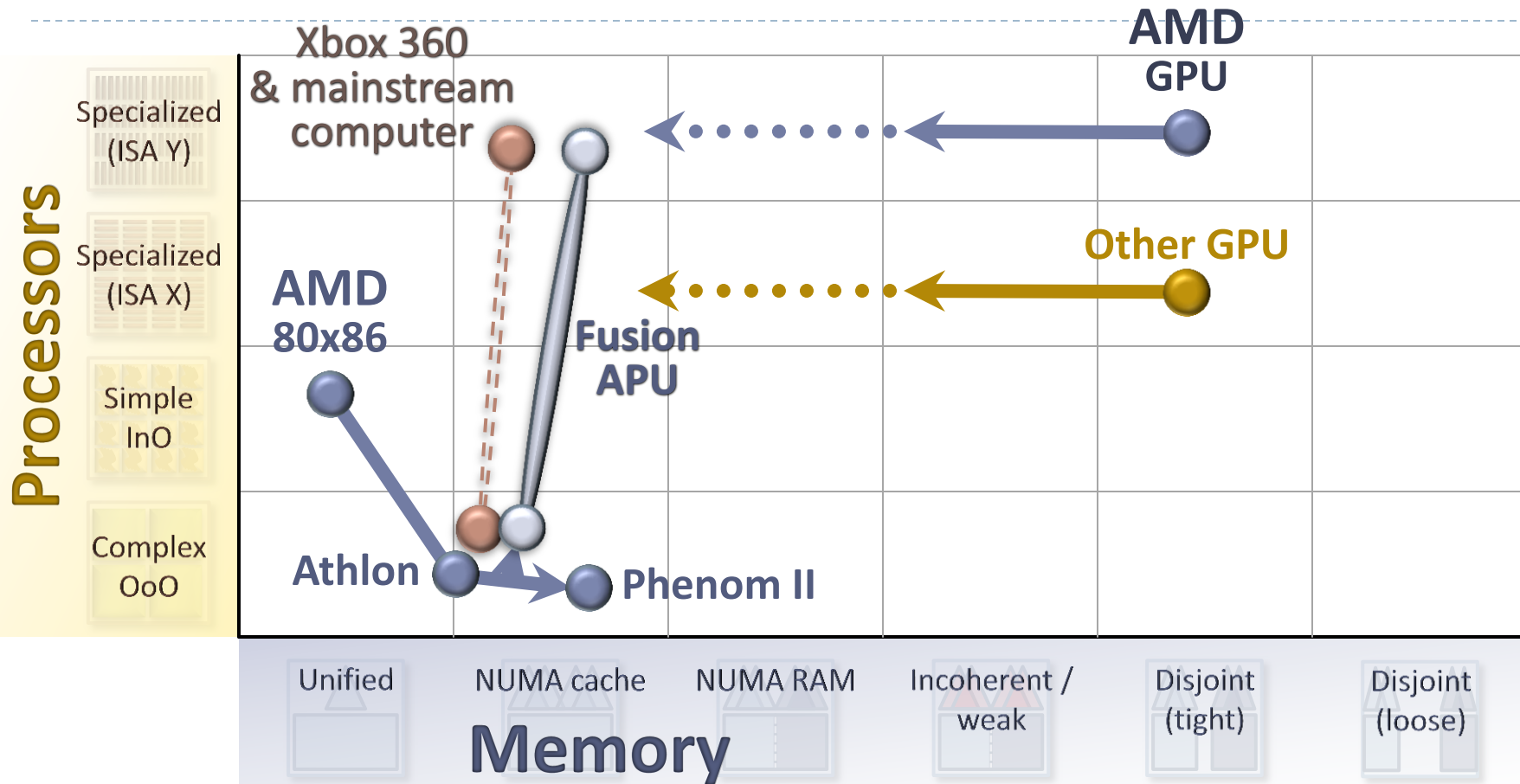
cloud IaaS/HaaS

**Heterogeneous
parallel
computing**

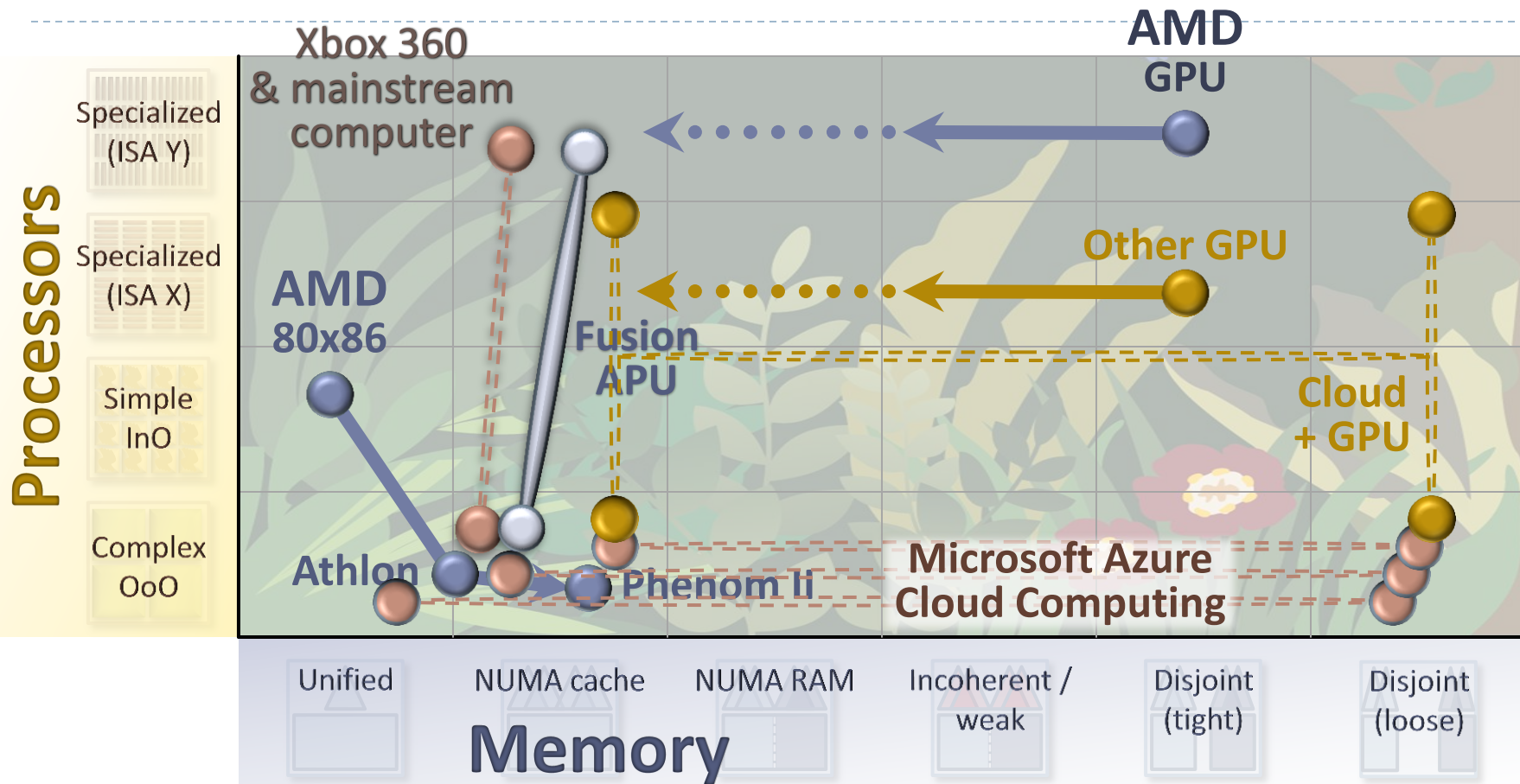
Charting the Landscape



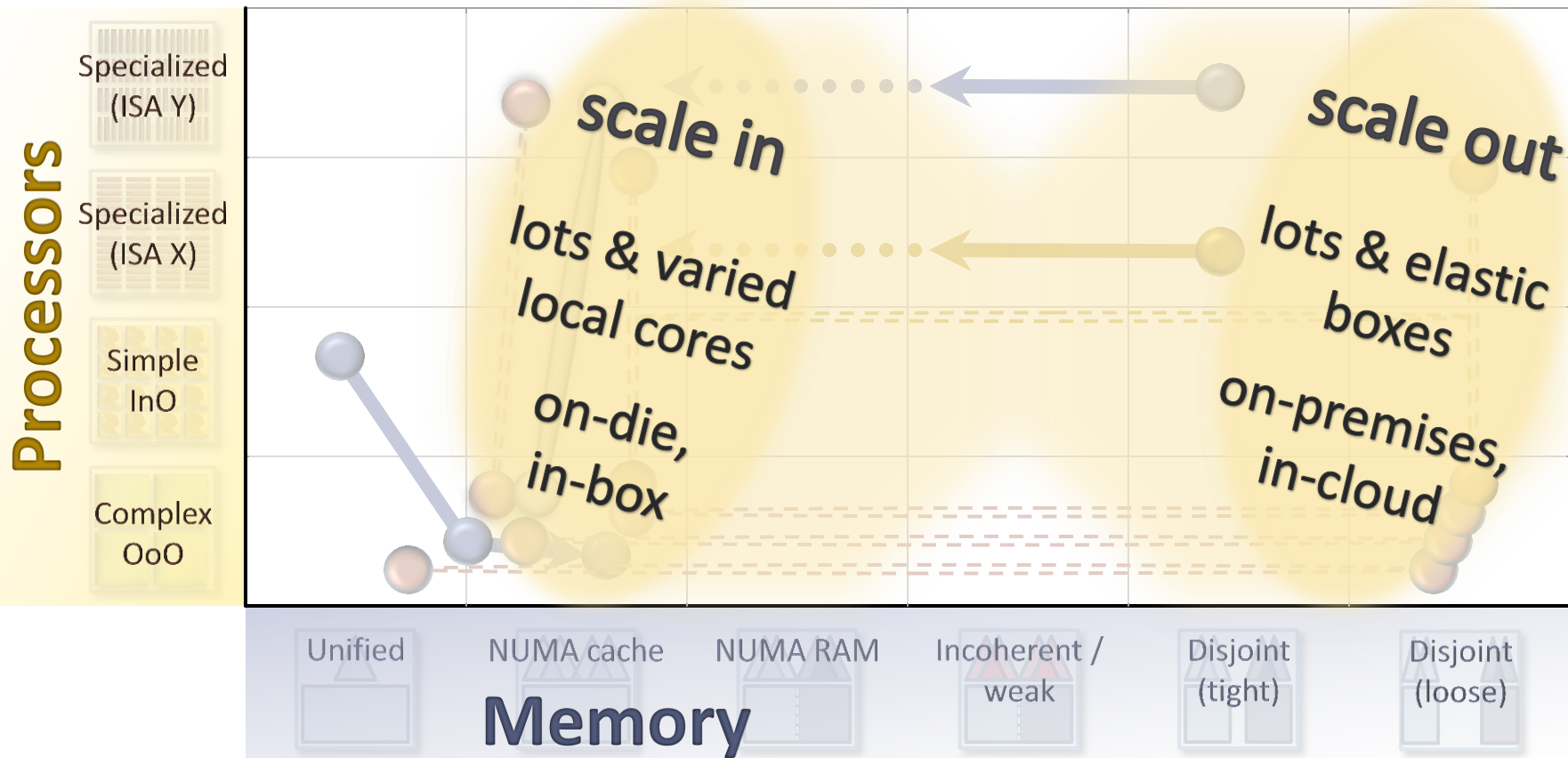
Hardware



Hardware



Hardware Evolution



News Flash

Slashdot



stories

recent

C++ the Clear Winner In Google's Language Performance Tests

Google has released a [research paper](#) that suggests C++ is the best-performing programming language in the market.

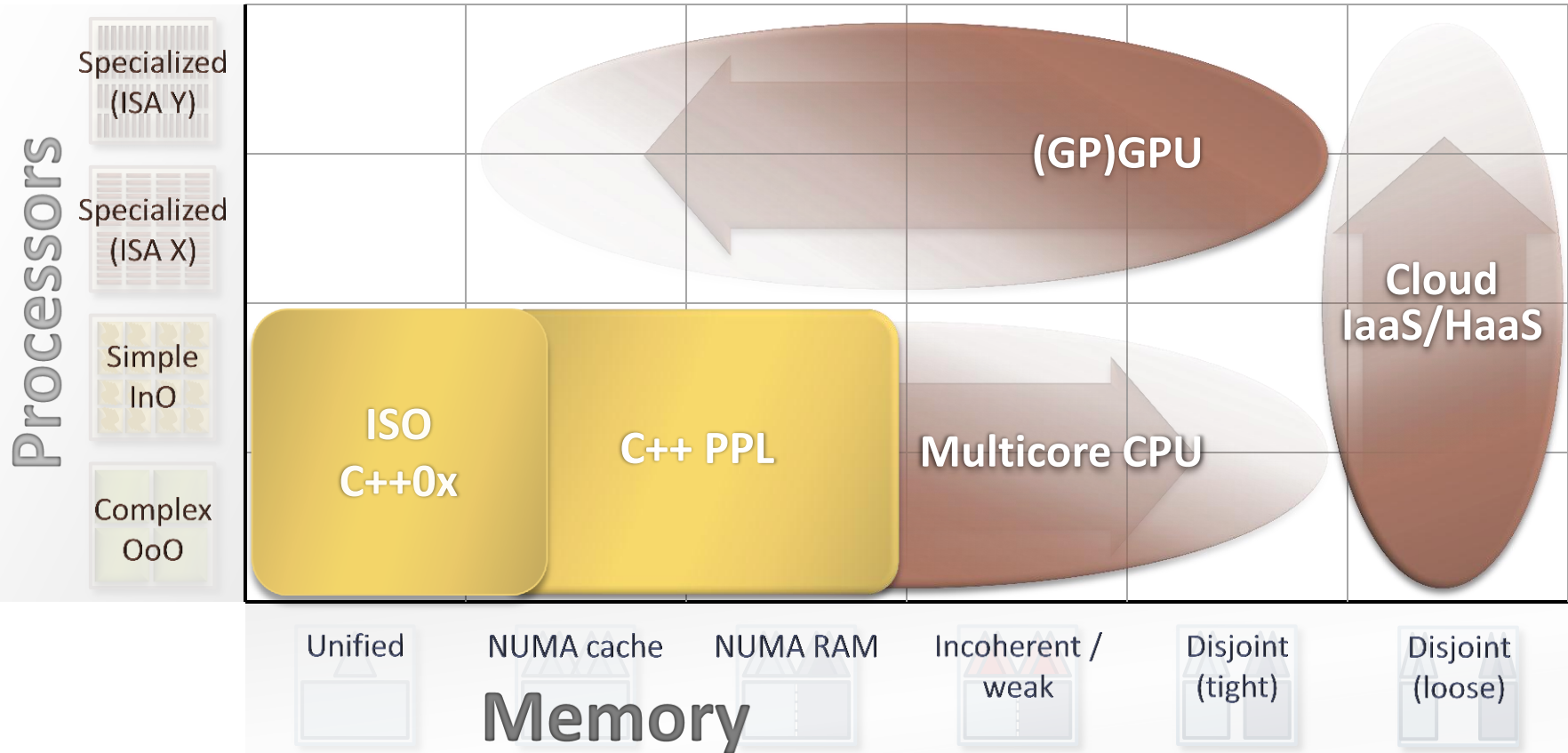
The internet giant implemented a compact algorithm in four languages – C++, Java, Scala and its own programming language Go – and then benchmarked results to find "factors of difference".

Further reading

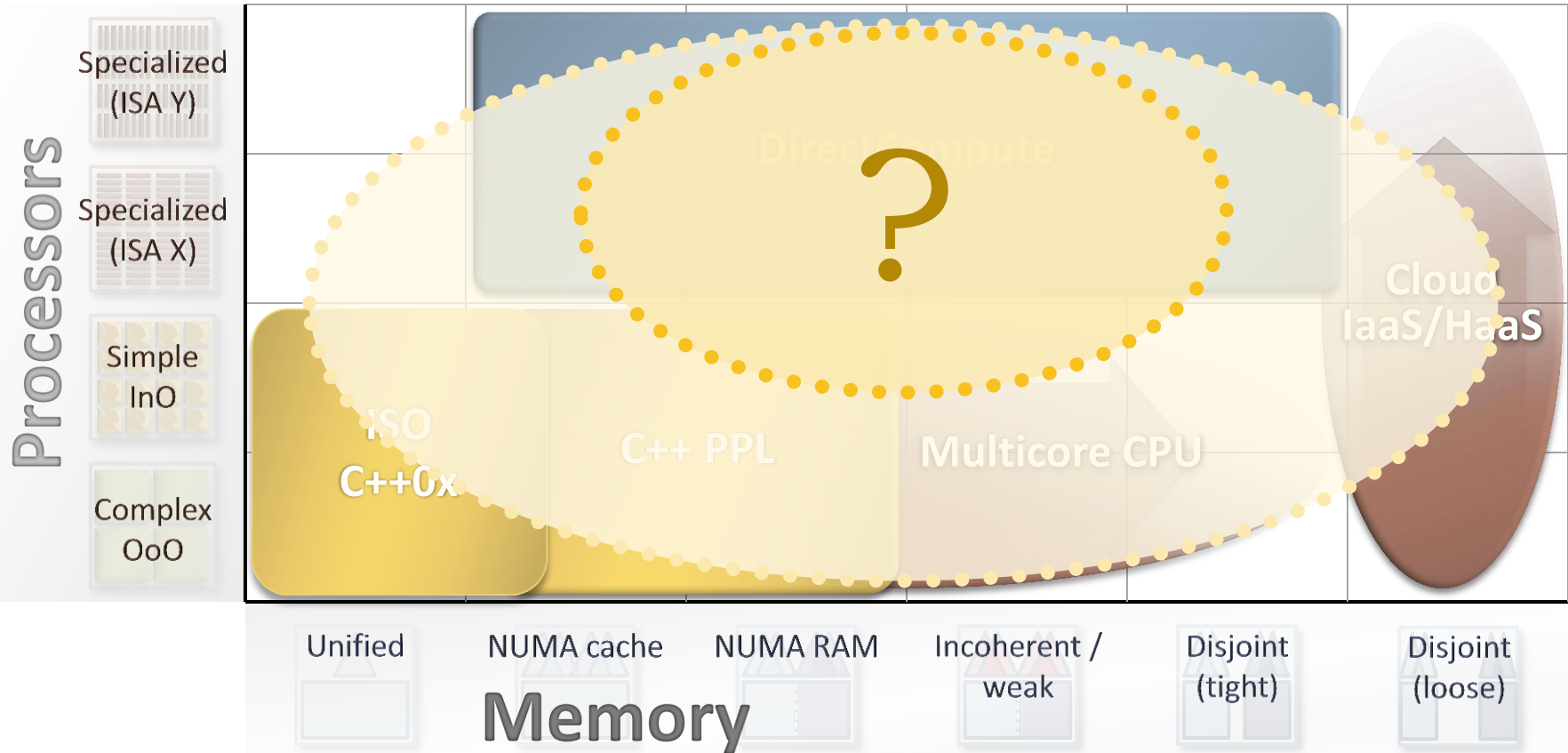
> [Surge in C++ demand](#)

"We find that in terms of performance, C++ wins out by a large margin," the paper says.

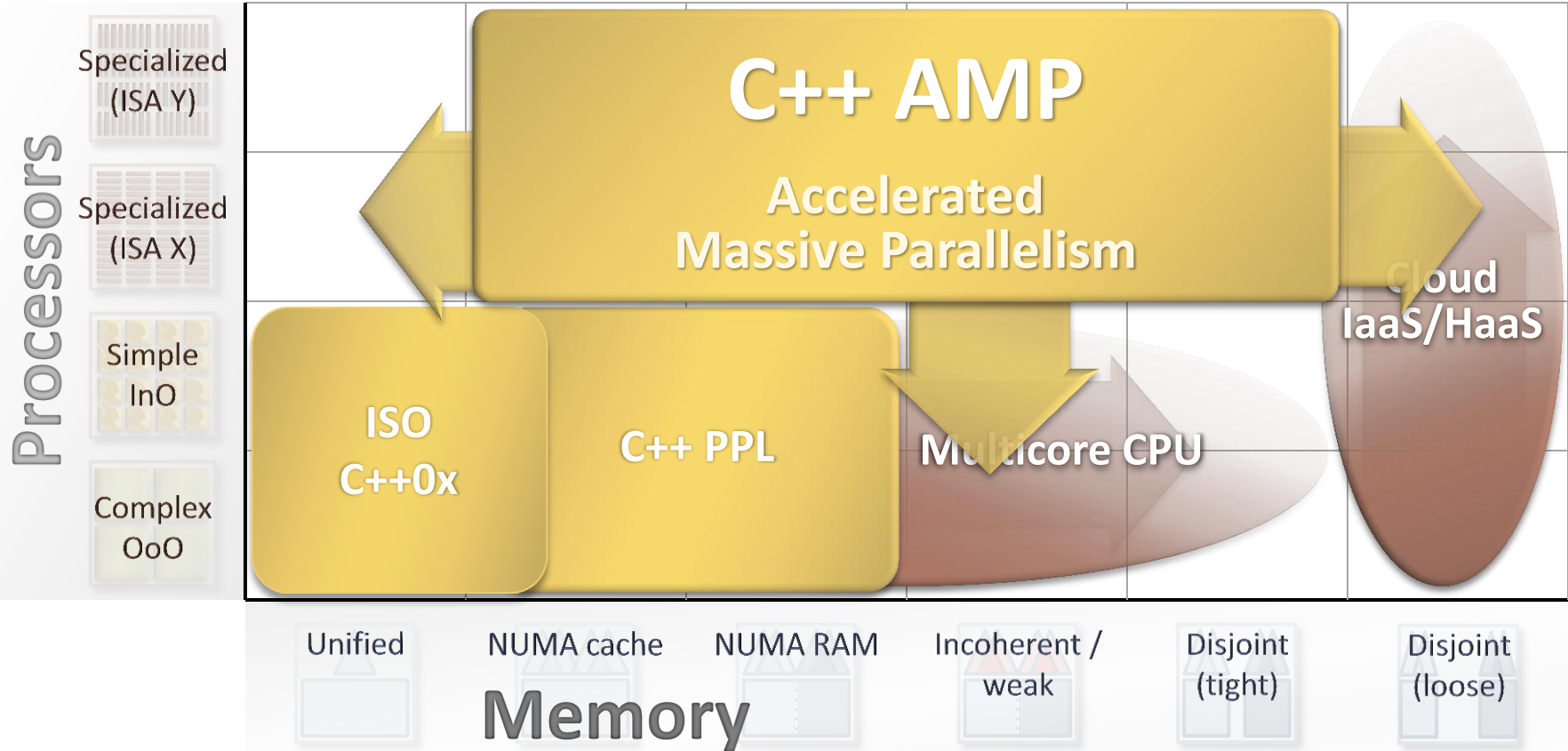
Programming Models & Languages



Programming Models & Languages



Programming Models & Languages



Matrix Multiply

Convert this (serial loop nest)

```
void MatrixMult( float* C, const vector<float>& A, const vector<float>& B,
                int M, int N, int W )
{
    for (int y = 0; y < M; y++)
        for (int x = 0; x < N; x++) {
            float sum = 0;
            for(int i = 0; i < W; i++)
                sum += A[y*W + i] * B[i*N + x];
            C[y*N + x] = sum;
        }
}
```

Matrix Multiply

Convert this (serial loop nest)

```
void Mat
```

```
{  
  for (int i = 0; i < M; i++)  
    for (int j = 0; j < N; j++)  
      for (int k = 0; k < W; k++)  
        sum += A[i][k] * B[k][j];  
  C[i][j] = sum;  
}
```

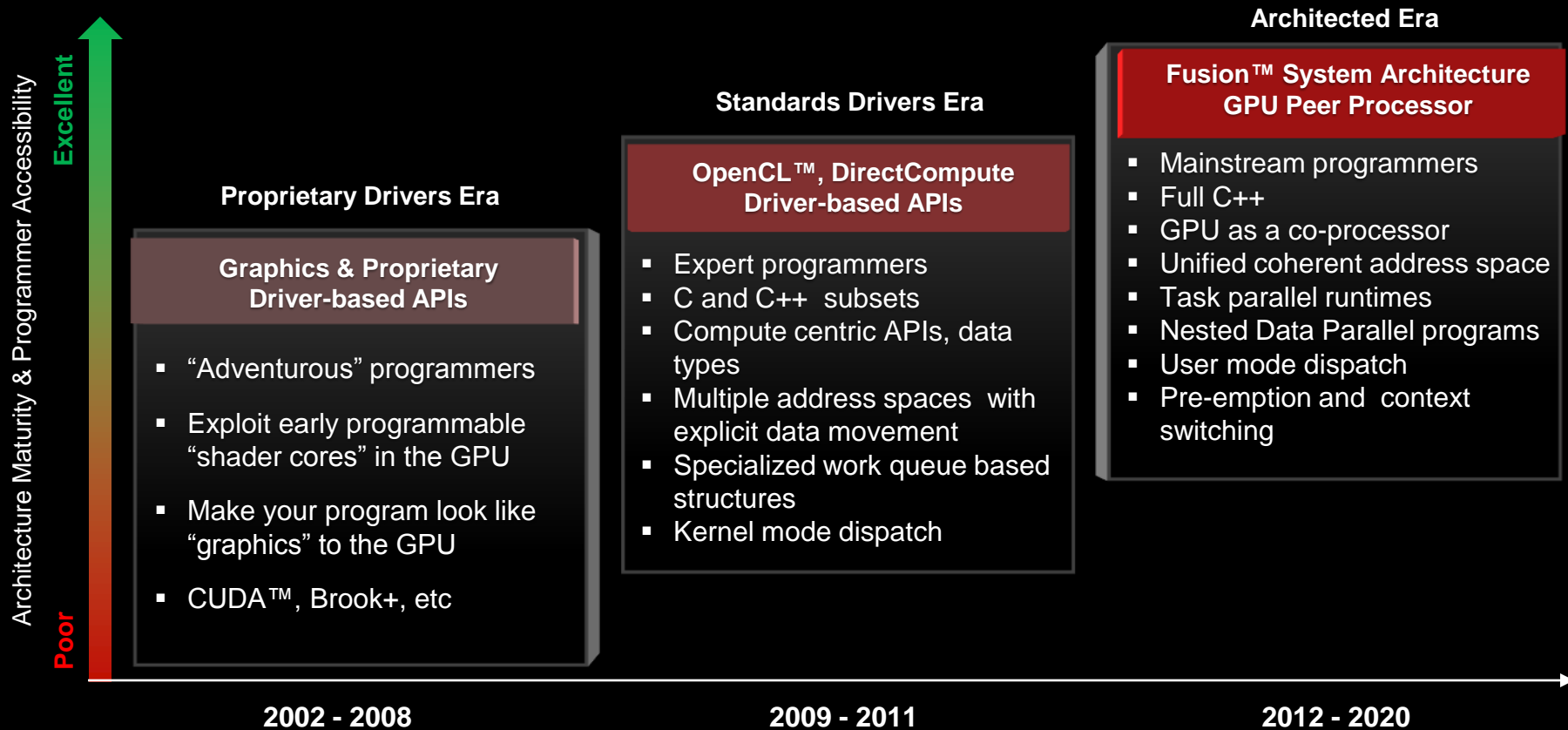
... to this (parallel loop, CPU or GPU)

```
void MatrixMult( float* C, const vector<float>& A, const vector<float>& B,  
                int M, int N, int W )  
{  
  array_view<const float,2> a(M,W,A), b(W,N,B);  
  array_view<writeonly<float>,2> c(M,N,C);  
  
  parallel_for_each( c.grid, [=](index<2> idx) restrict(direct3d) {  
    float sum = 0;  
    for(int i = 0; i < a.x; i++)  
      sum += a(idx.y, i) * b(i, idx.x);  
    c[idx] = sum;  
  } );  
}
```

Demo

Daniel Moth
Program Manager
Parallel Computing Platform

EVOLUTION OF HETEROGENEOUS COMPUTING

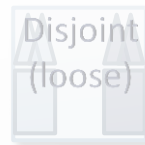
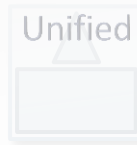


Why C++ AMP?

Processors



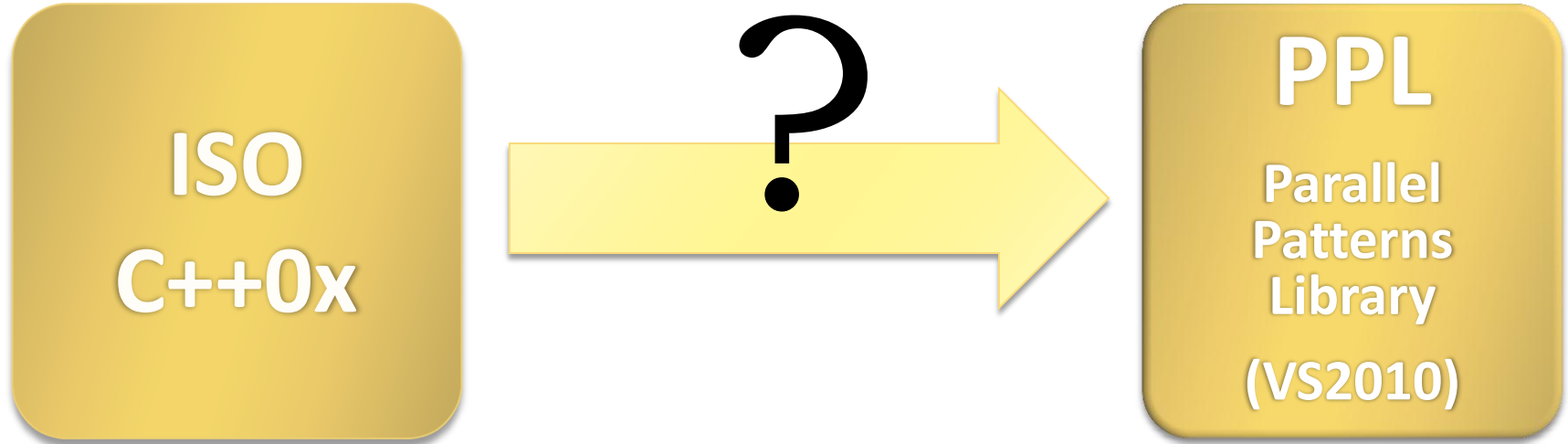
			C++ , not C		
			mainstream , programmable by millions		
			minimal , just one general language extension		
			portable , mix & match hardware from any vendor, one EXE		
			general and future-proof , designed to cover the full range of hardware heterogeneity – hardware is still in motion		



Memory

Language Design: Parallelism Phase 1

Single-core to multi-core



Language Design: Parallelism Phase 1

Single-core to multi-core

ISO
C++0x

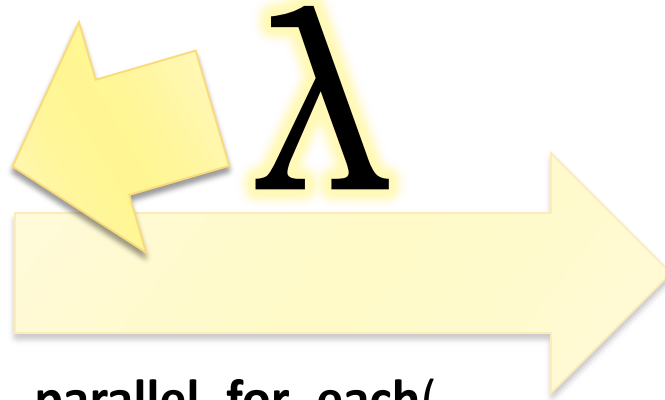
forall(x, y)
forall(z; w; v)
forall(k, l, m, n)
... ?

PPL
Parallel
Patterns
Library
(VS2010)

Language Design: Parallelism Phase 1

Single-core to multi-core

ISO
C++0x



```
parallel_for_each(  
    items.begin(), items.end(),  
    [=]( Item e )  
    {  
        ... your code here ...  
    }  
);
```

PPL
Parallel
Patterns
Library
(VS2010)

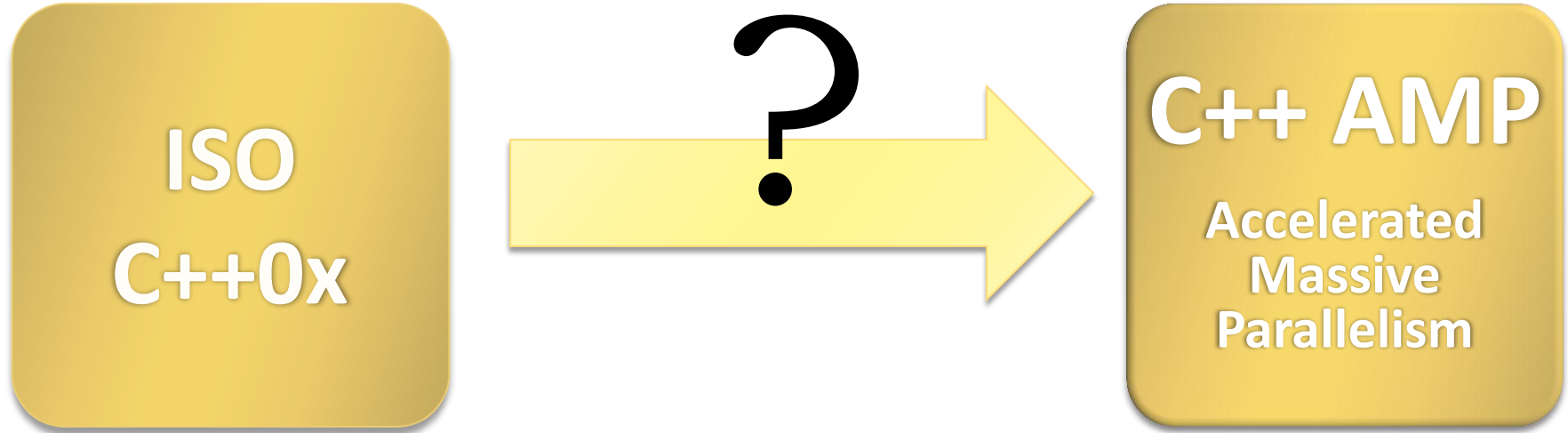
1

language feature for multicore

and STL, functors, callbacks, events, ...

Language Design: Parallelism Phase 2

Multi-core to hetero-core



Language Design: Parallelism Phase 2

Multi-core to hetero-core

ISO
C++0x

restrict

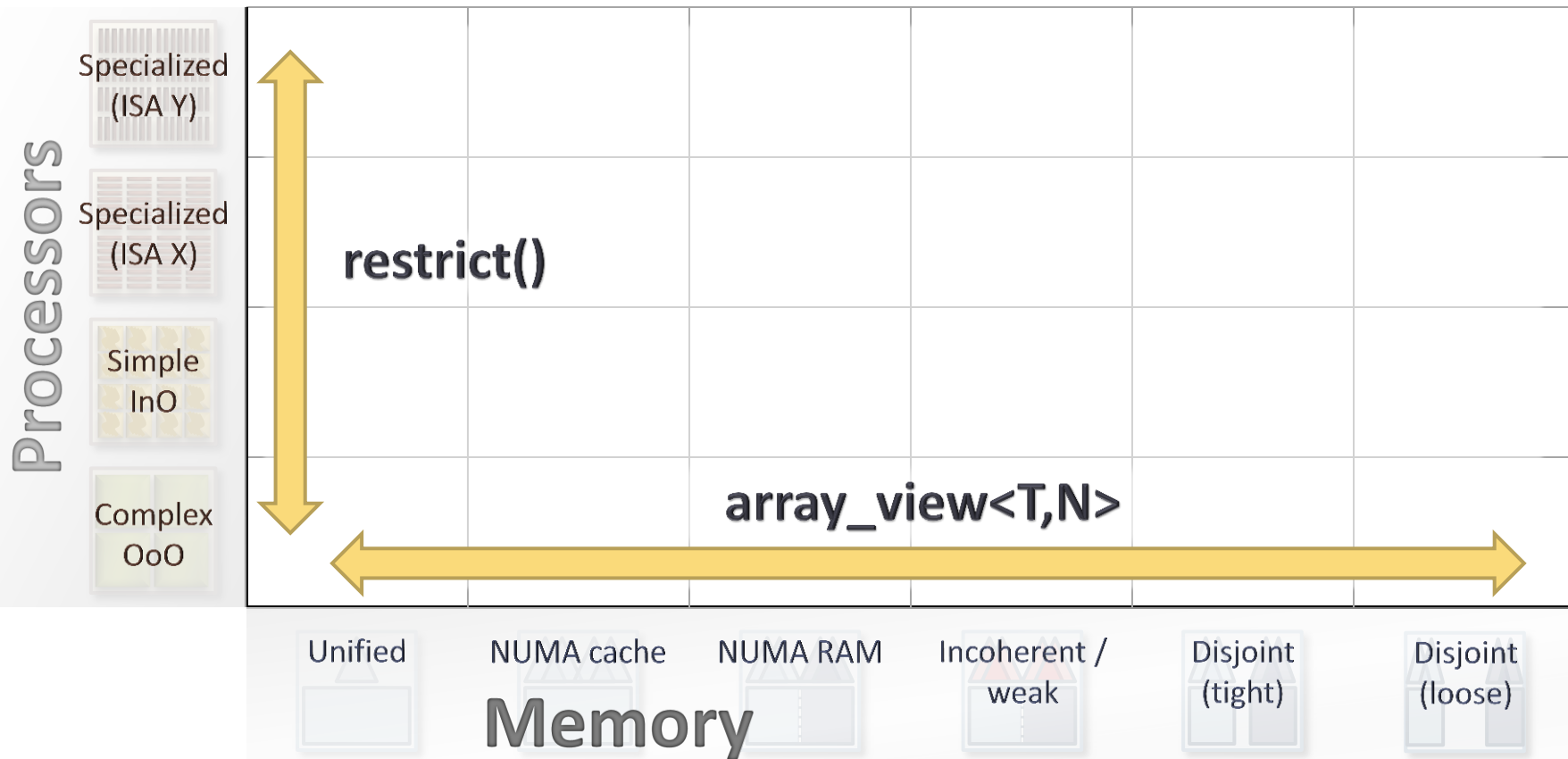
```
parallel_for_each(  
  items.grid,  
  [=](index<2> i) restrict(direct3d)  
  {  
    ... your code here ...  
  });
```

C++ AMP
Accelerated
Massive
Parallelism

1

language feature for
heterogeneous cores

C++ AMP at a Glance



restrict()

- ▶ **Problem:** Some cores don't support the entire C++ language.
- ▶ **Solution:** General restriction qualifiers enable expressing language subsets within the language. Direct3d math functions in the box.

Example

```
double sin( double ); // 1a: general code
double sin( double ) restrict(direct3d); // 1b: specific code

double cos( double ) restrict(direct3d); // 2: same code for either

parallel_for_each( c.grid, [=](index<2> idx) restrict(direct3d) {
    ...
    sin( data.angle ); // ok, chooses overload based on context
    cos( data.angle ); // ok
    ...
});
```

restrict()

- ▶ Initially supported restriction qualifiers:
 - ▶ **restrict(cpu)**: The implicit default.
 - ▶ **restrict(direct3d)**: Can execute on any DX11 device via DirectCompute.
 - ▶ Restrictions follow limitations of DX11 device model (e.g., no function pointers, virtual calls, goto).
- ▶ Potential future directions:
 - ▶ **restrict(pure)**: Declare and enforce a function has no side effects. Great to be able to state declaratively for parallelism.
 - ▶ General facility for language subsets, not just about compute targets.

array_view

- ▶ **Problem:** Memory may be flat, nonuniform, incoherent, *and/or* disjoint.
- ▶ **Solution:** Portable view that works like an N-dimensional “iterator range.”
 - ▶ Future-proof: No explicit `.copy()/sync()`. As needed by each actual device.

Example

```
void MatrixMult( float* C, const vector<float>& A,
                 const vector<float>& B, int M, int N, int W )
{
    array_view<const float,2> a(M,W,A), b(W,N,B); // 2D view over C array
    array_view<writeonly<float>,2> c(M,N,C);      // 2D view over C++ std::vector
    parallel_for_each( c.grid, [=](index<2> idx) restrict(direct3d) {
        ...
    } );
}
```



TM

GPU Debugging

MatrixMultiplication (Debugging) - Microsoft Visual Studio

```
49 random_init_vector(vB);
50
51 //Perform the matrix multiplication on the GPU
52 extent<2> eA(M, N), eB(N, W), eC(M, W);
53 array<int, 2> mA(eA, vA.begin());
54 array<int, 2> mB(eB, vB.begin());
55 array<int, 2> mC(eC);
56
57 parallel_for_each(grid<2>(eC), [=, &mA, &mB, &mC] (index<2> idx) restrict(direction)
58 int result = 0;
59 for(int i = 0; i < mA.grid.extent[1]; i++)
60 {
61     index<2> idxA(idx[0], i);
62     index<2> idxB(i, idx[1]);
63     result += mA[idxA] * mB[idxB];
64 }
65 mC[idx] = result;
66 });
67
68 vC = mC;
```

Parallel Stacks

- 4 GPU Threads
 - <lambda_31BC6D5D9B462AAF>::operator()
 - 256 GPU Threads
 - _kernel_stub

GPU Threads

Kernel: parallel_for_each<2, ??>::

Thread	Thread Count	Line	Address	Location	Status
0, 0	252 threads		0x000001BC	_kernel_stub()<	Active
0, 1	4 threads	Line 63	0x00008E70	wmain: __l5: <lambda_31BC6D5D9B462AAF>::	Active

Parallel Watch 1 - wmain: __l5: <lambda_31BC6D5D9B462AAF>::operator()

[Thread]	idx	idxA	idxB	<Add Watch>
[0, 0]	{_M_base= [0, 0]}	{_M_base= [0, 2]}	{_M_base= [2, 0]}	
[0, 1]	{_M_base= [0, 1]}	{_M_base= [0, 2]}	{_M_base= [2, 1]}	
[1, 0]	{_M_base= [1, 0]}	{_M_base= [1, 2]}	{_M_base= [2, 0]}	
[1, 1]	{_M_base= [1, 1]}	{_M_base= [1, 2]}	{_M_base= [2, 1]}	

Bring CPU debugging experience to the GPU

GPU Debugging

```
57 parallel_for_each(grid<2>(eC), [=, &mA, &mB, &mC] (index<2> idx) restrict(dire
58     int result = 0;
59     for(int i = 0; i < mA.grid.extent[1]; i++)
60     {
61         index<2> idxA(idx[0], i);
62         index<2> idxB(i, idx[1]);
63         result += mA[idxA] * mB[idxB];
64     }
65     mC[idx] = result;
66 });
67
```

The screenshot shows a debugger window with a table of GPU threads. The kernel being debugged is 'parallel_for_each<2, ??>'. The table shows 252 threads in the kernel stub and 4 threads at line 63.

Thread	Thread Count	Line	Address	Location	Status
0, 0 0..1, 0..1	252 threads		0x000001BC	__kernel_stub0	Active
	4 threads	Line 63	0x00008E70	wmain::_J5::<lambda_31BC6D5D9B462AAF>::	Active

Parallel Stacks

Threads

4 GPU Threads

<lambda_31BC6D5D9B462AAF>::operator()

256 GPU Threads

__kernel_stub

Bring CPU
debugging
experience
to the GPU

GPU Debugging

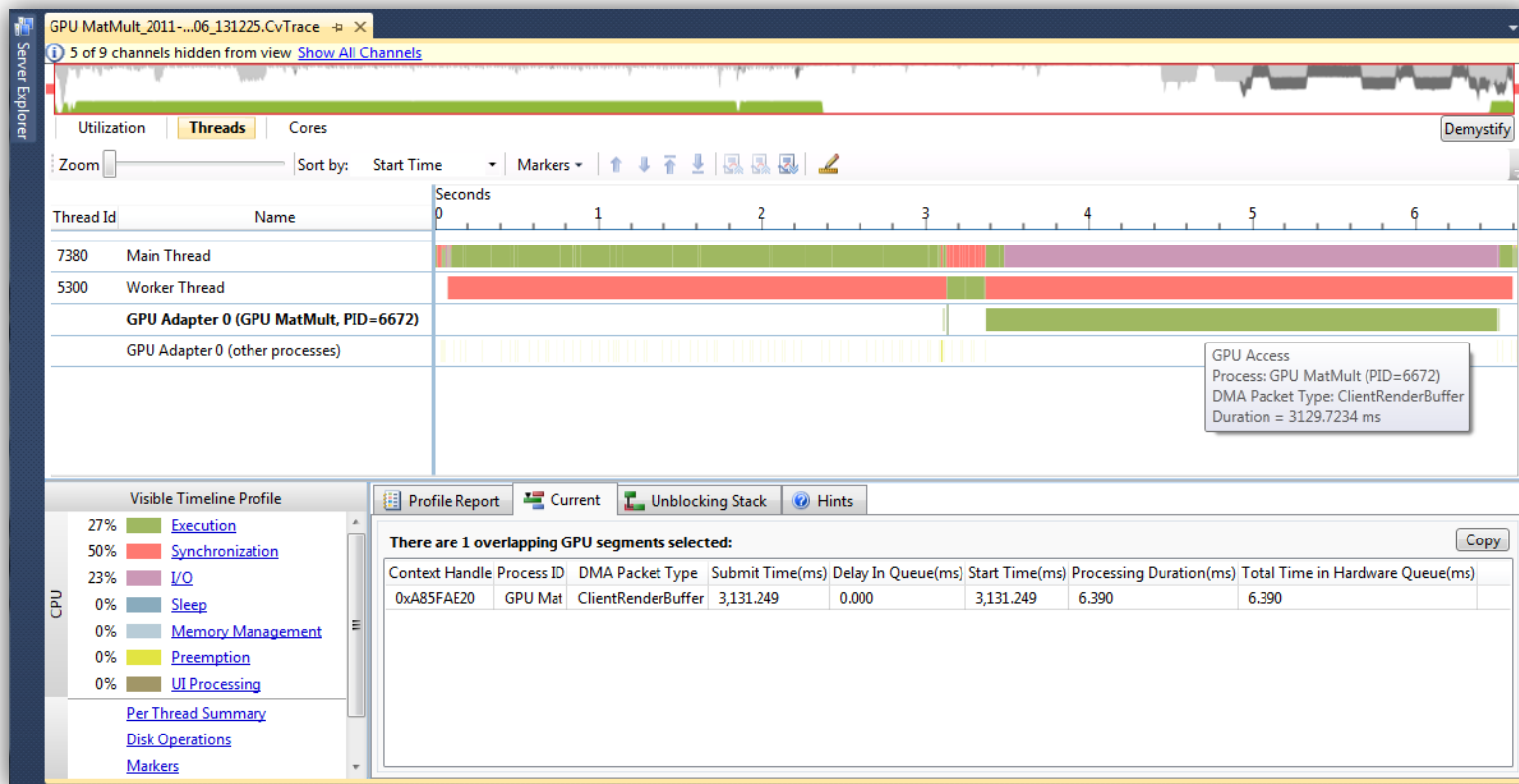
GPU Threads

Kernel: parallel_for_each<16,16, ...

Tile: 0, 0 0..7, 0..15 | Thread: 0, 0 0..15, 0..15

	Thread Count	Line	Address	Location	Status	Tile
Thread Group: [0, 0] (256 Threads)						
▼	252 threads		0x000127D8	_52C1BEEA_3B8D_4BFC_86DE_7D17D68D02A2_	Active	[0, 0]
▼	2 threads	Line 22	0x00012A88	ObtainAbsoluteIndex	Diverged	[0, 0]
▼ →	2 threads	Line 13	0x00012BAC	CalculateAbsoluteIndex	Active	[0, 0]
Thread Group: [0, 11] (256 Threads)						
▼	252 threads	1 warp; 1 diverged warp: warp 0		_52C1BEEA_3B8D_4BFC_86DE_7D17D68D02A2_	Active	[0, 1]
▼	2 threads		2 active threads at CalculateAbsoluteIndex line 13 (address 0x00012BAC)		Diverged	[0, 1]
▼	2 threads		2 diverged threads at ObtainAbsoluteIndex line 22 (address 0x00012A88)		Active	[0, 1]
Thread Group: [0, 10] (256 Threads)						
▼	252 threads		0x000127D8	_52C1BEEA_3B8D_4BFC_86DE_7D17D68D02A2_	Active	[0, 10]
▼	2 threads	Line 22	0x00012A88	ObtainAbsoluteIndex	Diverged	[0, 10]
▼	2 threads	Line 13	0x00012BAC	CalculateAbsoluteIndex	Active	[0, 10]
Thread Group: [0, 11] (256 Threads)						
▼	252 threads		0x000127D8	_52C1BEEA_3B8D_4BFC_86DE_7D17D68D02A2_	Active	[0, 11]
▼	2 threads	Line 22	0x00012A88	ObtainAbsoluteIndex	Diverged	[0, 11]
▼	2 threads	Line 13	0x00012BAC	CalculateAbsoluteIndex	Active	[0, 11]

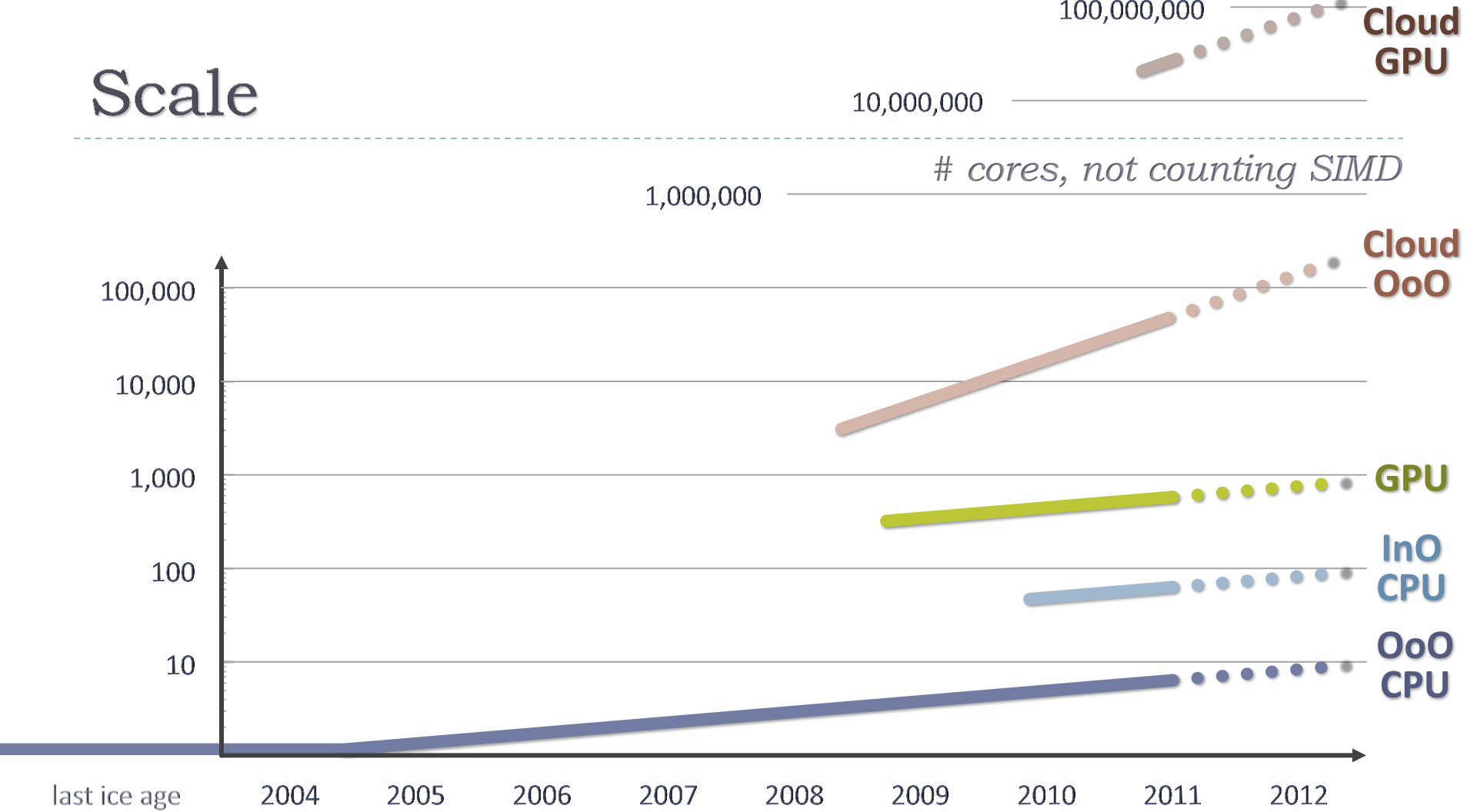
GPU Profiling



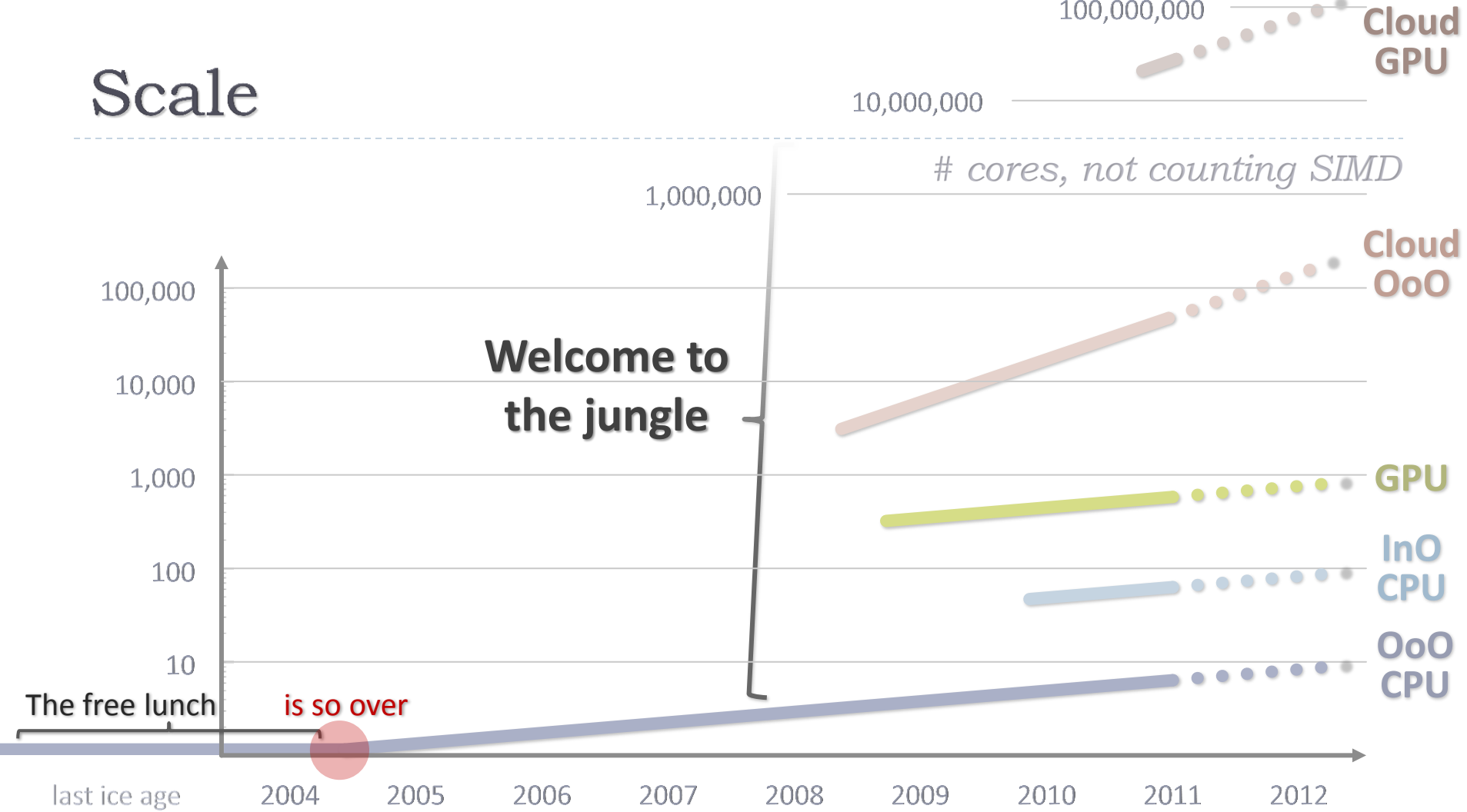


TM

Scale



Scale



C++ AMP

Processors



			C++ , not C		
			mainstream , programmable by millions		
			minimal , just one general language extension		
			portable , mix & match hardware from any vendor, one EXE		
			general and future-proof , designed to cover the full range of hardware heterogeneity – hardware is still in motion		
			open specification		



Memory

C++ PPL: 9:45am
C++ AMP: 2:00pm, Room 406

Heterogeneous Parallelism at Microsoft

Herb Sutter