Ct Technology: Steps Toward Starting a Productivity Revolution in Multi-core Computing

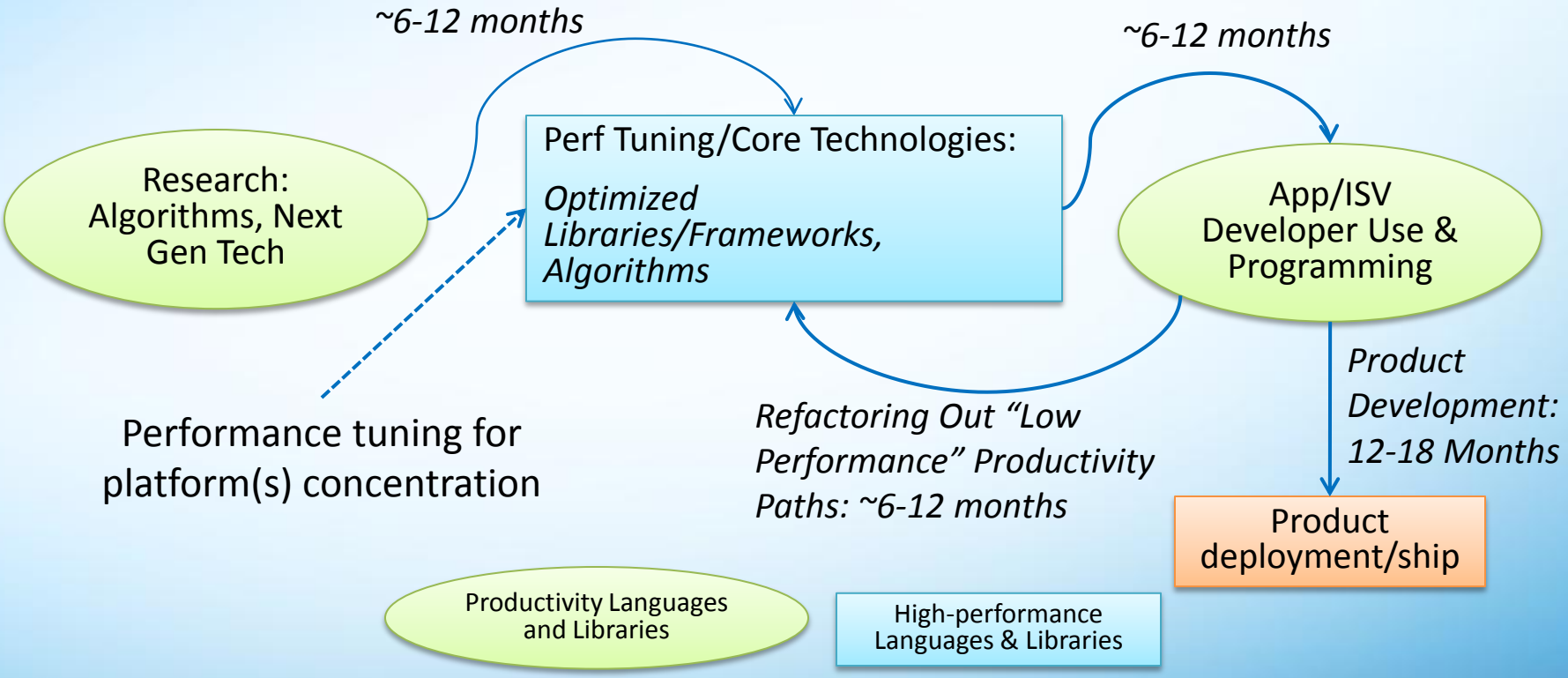
Anwar Ghuloum, Intel Corporation

http://www.intel.com/go/ct

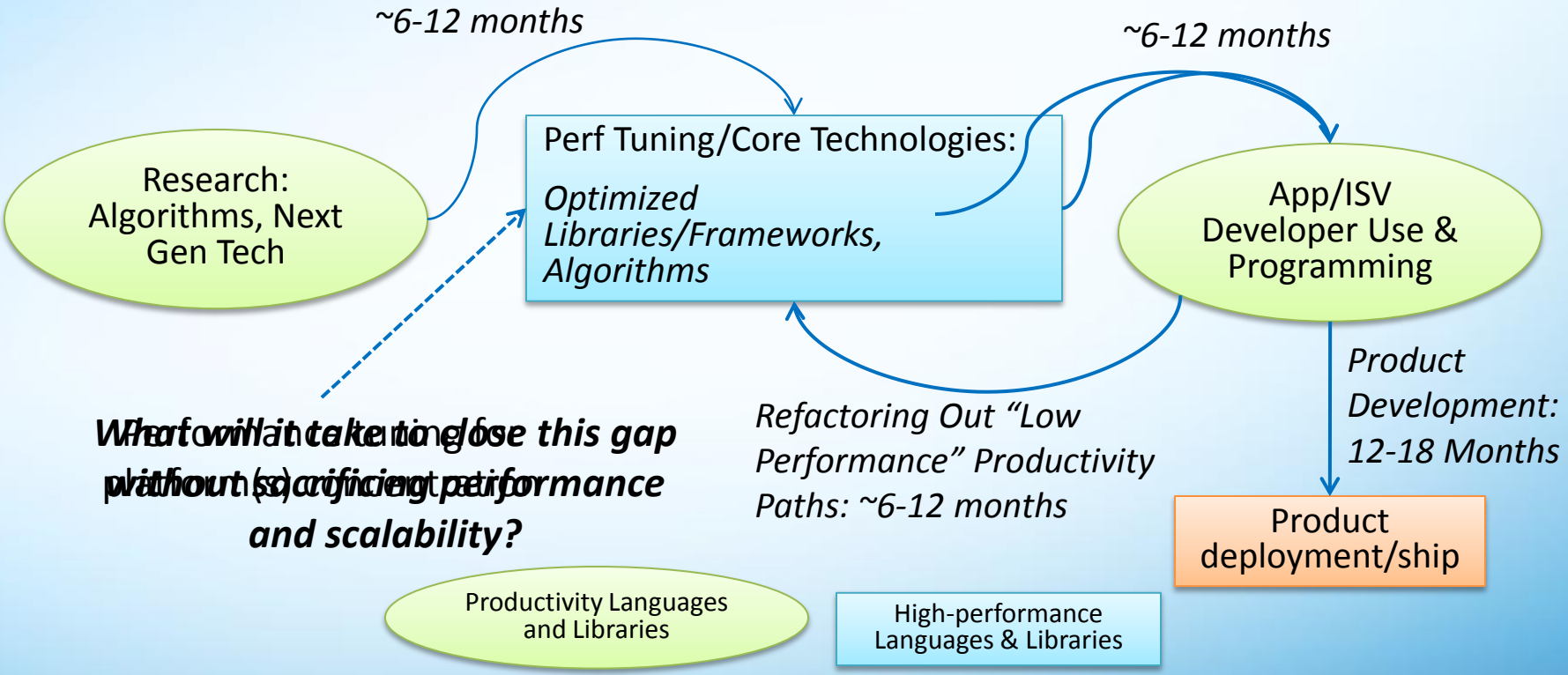
Agenda

- Why Dynamic Compilation is important multi-core
- What is Ct? (And how it solves this.)

The Product Lifecycle in Throughput (aka Multi-core) Computing



The Product Lifecycle in Throughput (aka Multi-core) Computing



Language Trends: Do We Really Only Care About C and Fortran?

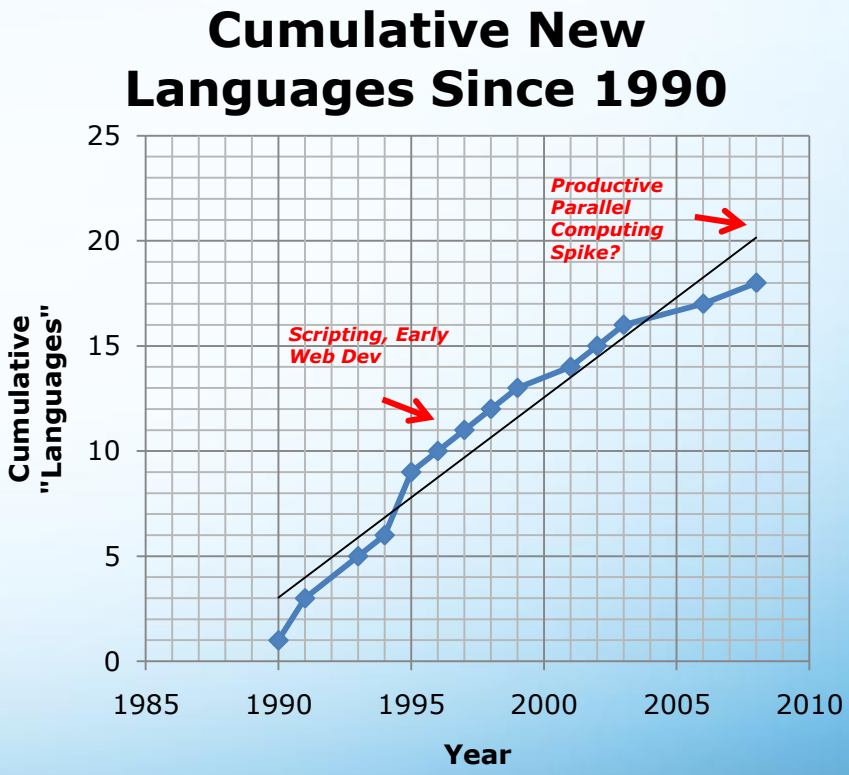
Languages with some commercial adoption:

- Java, C#, Ocaml, F#, Ruby, Python, Lua, PHP, Java/Ecmascript, Actionscript, OpenCL, Scala, OpenMP, C for Cuda, Cilk, R, D

→ New language every 12-18 months!

(*Not* including webapp frameworks, custom scripting engines in game platforms, etc.)

Mostly off the parallel computing radar…until ca. 2005



Domain Specific Languages and Libraries

- Domain specific languages: why narrow applicability?
  - ***Tradeoff between performance and productivity can only be relaxed by leveraging domain knowledge***
  - Multi-language development & new language adoption isn't the barrier we once thought it was
- Blurring the line between languages and "libraries"
  - Modern language mechanisms allow library development that significantly extends capabilities of languages
  - Lowers developer resistance to adoption
  - Examples:
    - Domain specific libs: ITK, CTL, QuantLib → High functionality/modularity, low performance
    - Template meta-programmed libs: uBlas
    - Dynamic meta-programmed APIs: Ct

C++ Developers: Is This An Issue?

It's not just a single kernel…

- Productivity craters when many kernels have to be tuned
  - Focusing energy on 1 algorithm makes sense, if it is the dominant algorithm

…in one place

- Widely used libraries often give up performance for well designed generic interfaces
  - Examples: ITK, Quantlib

→ Inherently spreads compute across many (virtual) functions

Reducing the Impact of Modularity

Providing user programmability at high performance

- Libraries with highly configurable interfaces often have reduced performance due to dynamic overhead of late binding and parameter generality
  - This is relatively more punishing for Multi-core!
- Example:
  - QuantLib is a financial modeling package designed to allow quantitative analysts to model and then price complex financial instruments
  - Provides a variety of ways to configure pricing and process models, often with user-provided functions and parameters
- Test case: binomial tree option pricing
  - Simple recurrence structure
  - User-configurable spot price and process functions

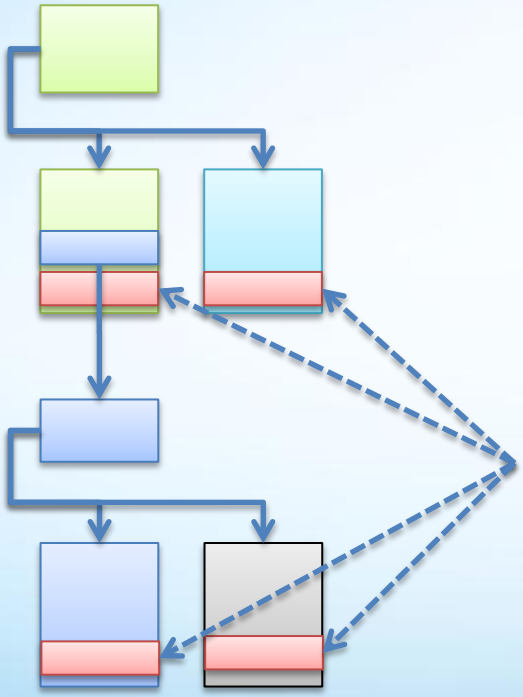
Financial example: high-level interface

High-Level Interface

- Financial analysts want high-level interface for modeling instruments, processes, pricing
- Concerned with mathematics, not details of parallelization
- Ct Technology can not only parallelize&vectorize, but can remove overhead of C++ modularity

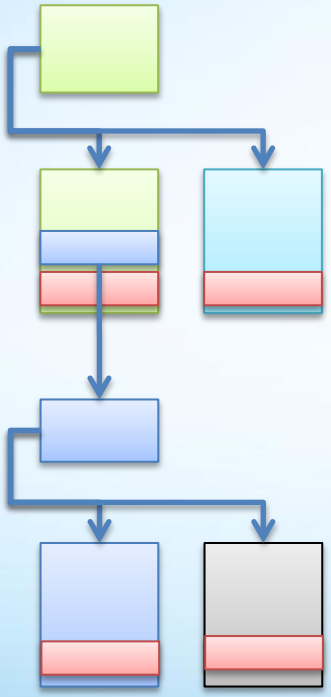
```
Real expiry = 1;
Real strike = 40.0, spot = 36.0;
Real vol = 0.2, r = 0.05;

shared_ptr<Payoff>
    callPay(new PayoffCall(strike));
shared_ptr<Exercise>
    euExercise(new EuropeanExercise(expiry));
shared_ptr<Option>
    euCallOpt(new VanillaOption(callPay, euExercise));
shared_ptr<StochasticProcess>
    bsm(new BlackScholesProcess(r, vol, S0));

float *npvArray = new float[binomial.get_numJobs()];
BOPMEngine<LocalArrayEvaluator> binomial_lattice(euCallOpt, bsm);
binomial_lattice.NPV(
    npvArray, npvArray + binomial.get_numJobs()
);
```

Performance Without De-architecting Software

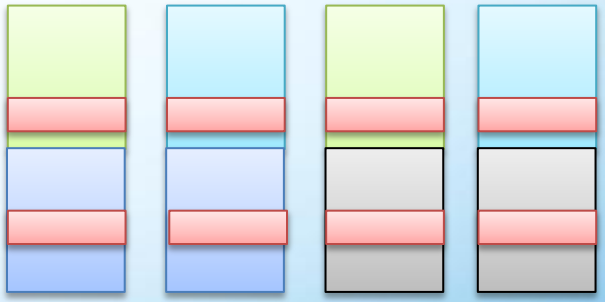


- Software is often architected for reuse, replacement, extension:
  - Use of generic algorithms, abstract classes, virtual function calls, C++ iterators, indirection is the norm…
- "Performance paths" are often spread across many objects and files

Performance Without De-architecting Software



- Performance tools typically want to see *everything!*
- You look at all possible/likely paths
  - Brittle
  - Difficult to maintain
  - Difficult to extend
  - Difficult to program

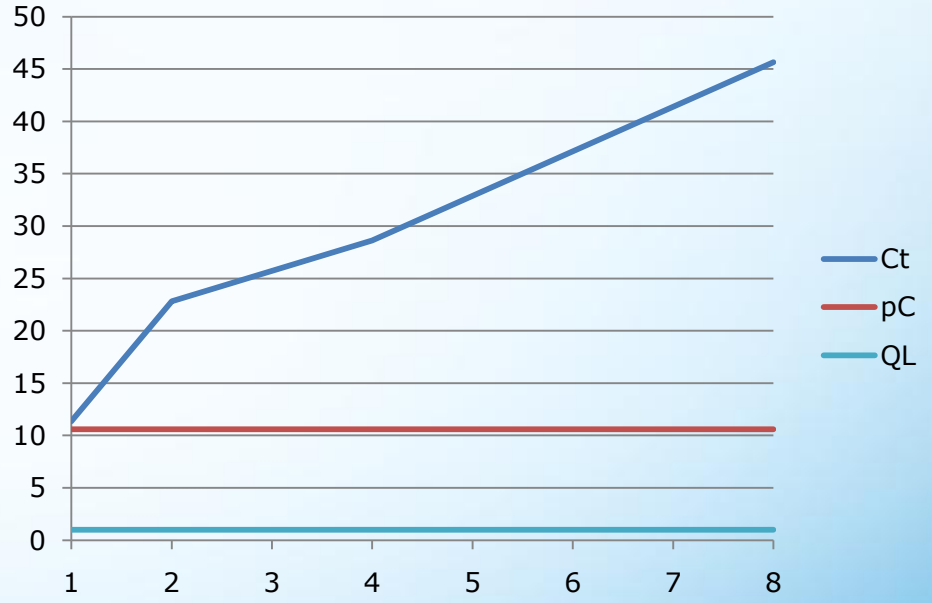


De-architecting/Flattening for performance

Binomial lattice: performance

Relative performance across implementations

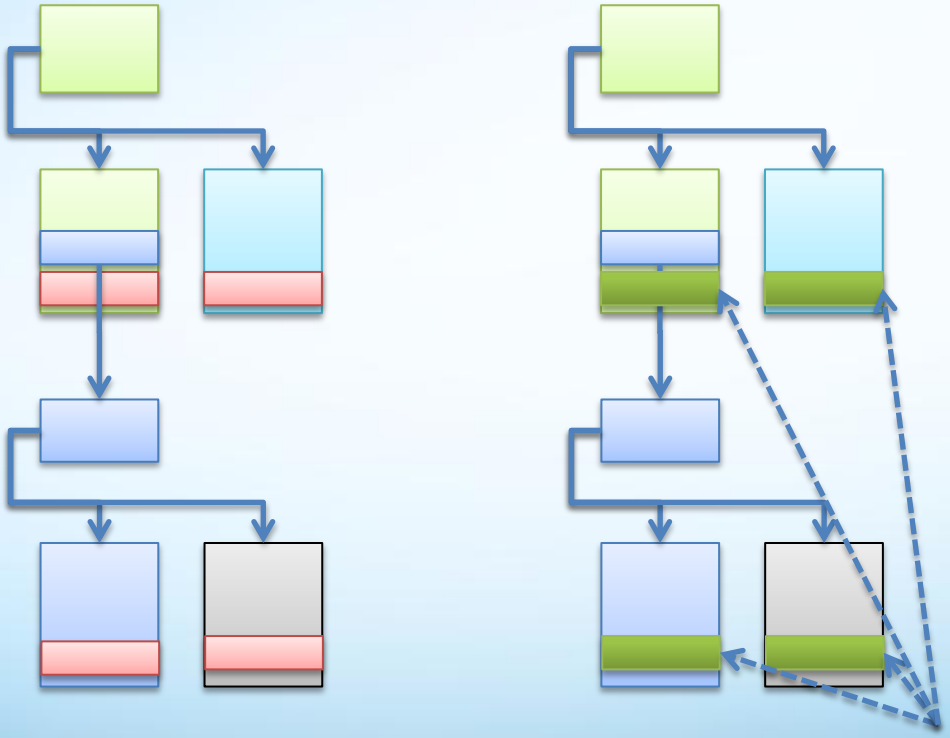
- QL: QuantLib baseline, not parallel
  - Modular library
  - Microsoft Visual Studio* at –O2
- pC: written with "plain C", not parallel
  - Modularity flattened by hand
  - Scalar code is 10.6x faster than QL
  - Microsoft Visual Studio* at –O2
- Ct: using Ct Technology
  - Scalar performance slightly better than pC
  - On 4 cores is 4.3x faster than pC



Number of threads (with 2 threads per core)
Intel® Core™ i7 microprocessor 920 quadcore @ 2.67GHz, double precision
Binomial lattice for 1024 options with 1500 timesteps each

*Other names and brands may be claimed as the property of others.

Performance Without De-architecting Software

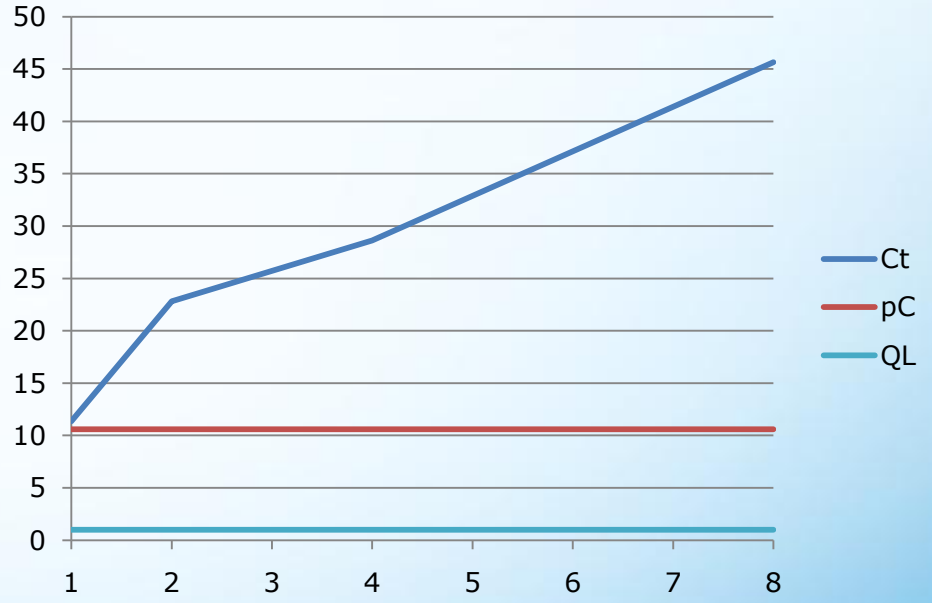


- Combine good software practices and performance with Ct:
  - Pepper your models/classes with Ct
  - Ct's VM takes care of *generatively* collecting the performance paths at run time (more later…)

Binomial lattice: performance

Relative performance across implementations

- QL: QuantLib baseline, not parallel
  - Modular library
  - Microsoft Visual Studio* at –O2
- pC: written with "plain C", not parallel
  - Modularity flattened by hand
  - Scalar code is 10.6x faster than QL
  - Microsoft Visual Studio* at –O2
- Ct: using Ct Technology
  - Scalar performance slightly better than pC
  - On 4 cores is 4.3x faster than pC



Number of threads (with 2 threads per core)
Intel® Core™ i7 microprocessor 920 quadcore @ 2.67GHz, double precision
Binomial lattice for 1024 options with 1500 timesteps each

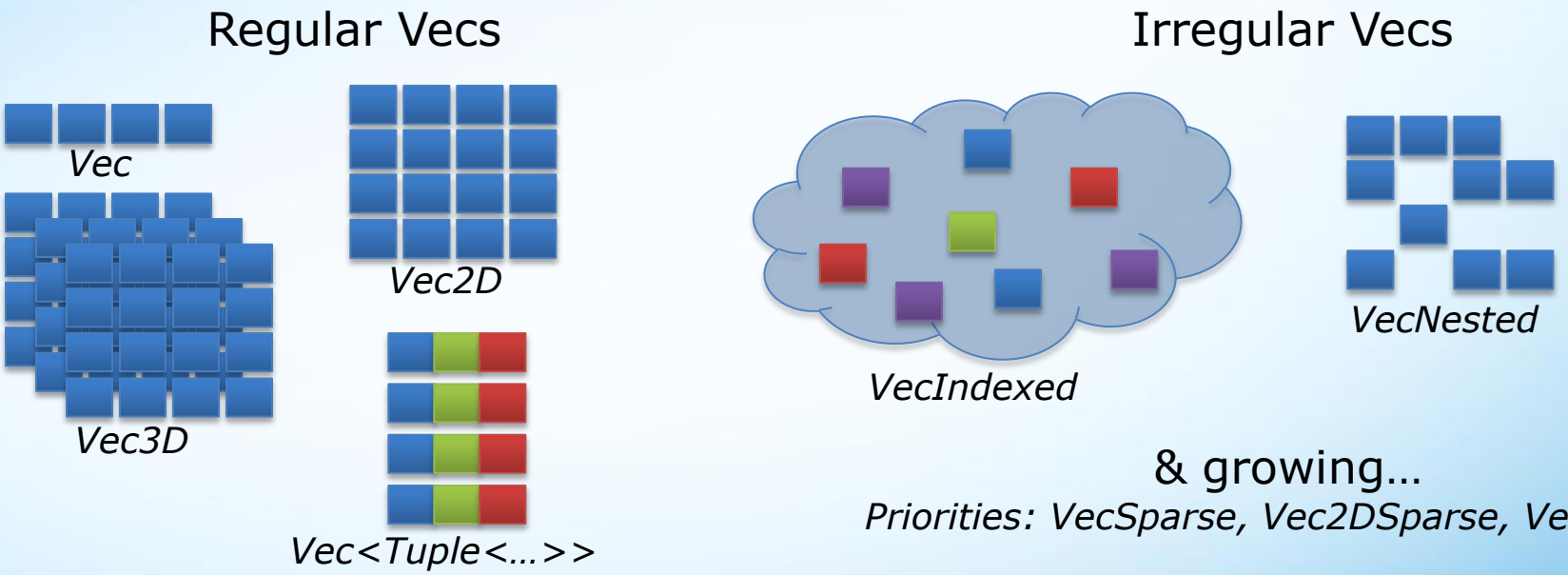
*Other names and brands may be claimed as the property of others.

So…What is Intel Ct Technology?

- Ct adds parallel collection objects & methods to C++
  - Library interface and is fully ANSI/ISO-compliant (works with ICC, VC++, GCC)
- Ct abstracts away architectural details
  - Vector ISA width / Core count / Memory model / Cache sizes
  - Focus on what to do, not how to do it
  - Sequential semantics
- Ct forward-scales software written today
  - Ct is designed to be dynamically retargetable to SSE, AVX, LRB, …
- Ct is safe, by default
  - …but with expert controls to override for performance

Programmers think sequential, not parallel

Operations over parallel collections



& growing…

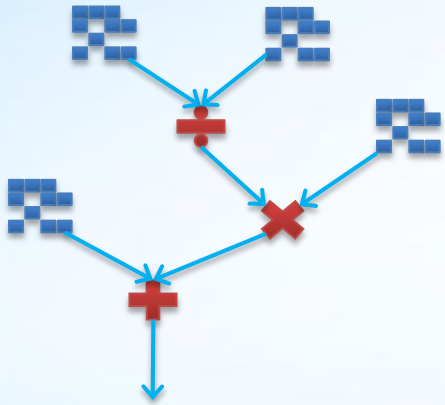
Priorities: VecSparse, Vec2DSparse, VecND

repeatCol, shuffle, transpose, swapRows, shift, rotate, scatter, …

Parallel Operations on Ct Collections

The Ct Runtime Automates This Transformation

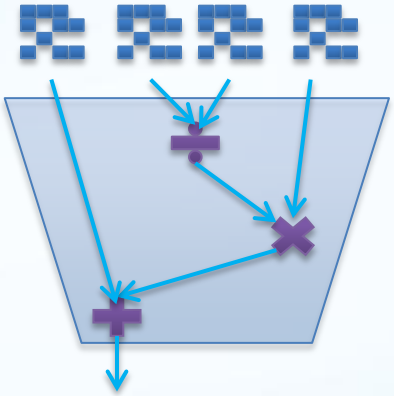
Vector Processing



```
Vec<F32> A, B, C, D;
A += B/C * D;
```

Linear algebra, global data movement/communication

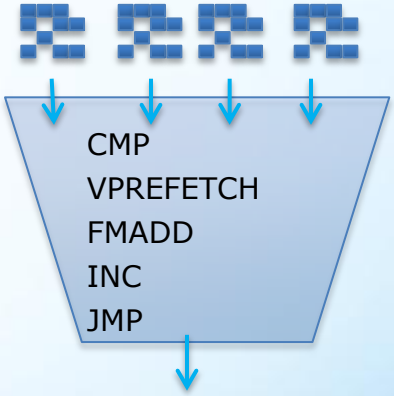
Kernel Processing



```
Elt<F32> kernel(Elt<F32> a, b, c, d) {
  return a + (b/c)*d;
}
…
Vec<F32> A, B, C, D;
A = map(kernel)(A, B, C, D);
```

Embarrassingly parallel, shaders, image processing

Native/Intrinsic Coding



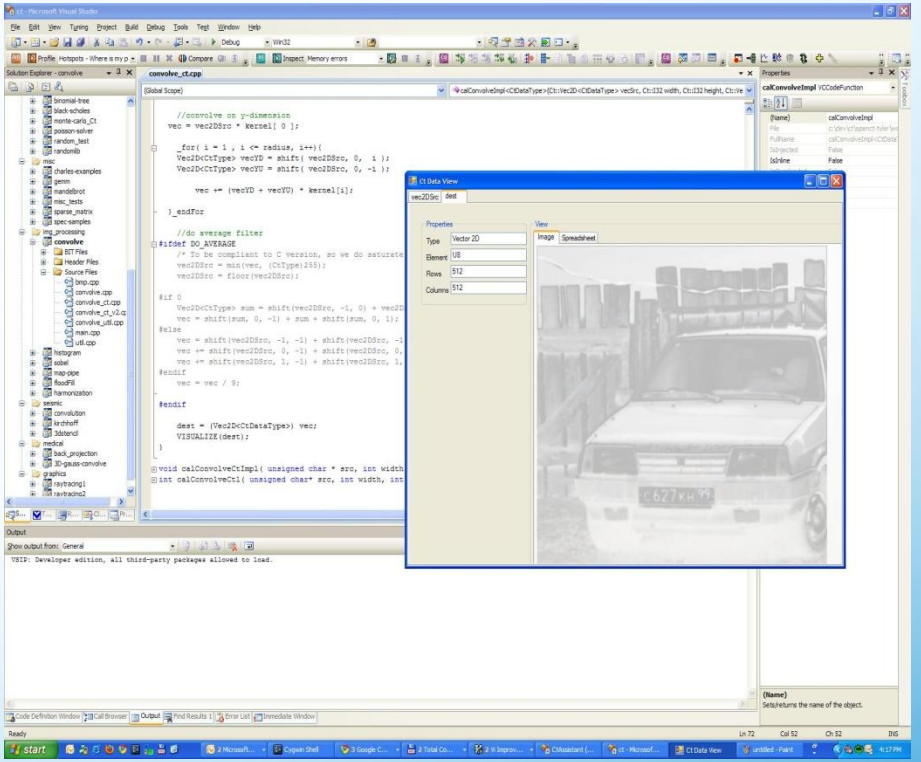

```
NVec<F32>native(NVec<F32> …) {
  __asm__ {
    …
  }
}
…
Vec<F32> A, B, C, D;
A = map(native)(A, B, C, D);
```

Or Programmers Can Choose Desired Level of Abstraction

IDE support

Use Ct Technology in your favorite development environment

- User-controlled display of Ct data structures
- Choose your display format, e.g. image, spreadsheet
- Invoke from either Ct operators or interactively from the IDE, e.g. Microsoft Visual Studio®
- There's no substitute for being able to visualize the results of transformation steps
- This is a key non-performance productivity feature



*Other names and brands may be claimed as the property of others.

3D order-6 stencil

```
template<typename T>
void fd3DStencilC(T *in, T *out, int nx, int ny, int nz)
{
    for (int i = 3; i < nx-3; i++){
        for (int j = 3; j < ny-3; j++){
            for (int k = 3; k < nz-3; k++){

                out[k+j*nz+i*nz*ny] = 2 * in[k+j*nz+i*nz*ny] - out[k+j*nz+i*nz*ny]

                      + coeff[0] * in[k+j*nz+i*nz*ny]
                      + coeff[1] *
                          (  in[k+j*nz+(i-1)*nz*ny] + in[k+j*nz+(i+1)*nz*ny]
                           + in[k+(j-1)*nz+i*nz*ny] + in[k+(j+1)*nz+i*nz*ny]
                           + in[(k-1)+j*nz+i*nz*ny] + in[(k+1)+j*nz+i*nz*ny] )
                      + coeff[2] *
                          (  in[k+j*nz+(i-2)*nz*ny] + in[k+j*nz+(i+2)*nz*ny]
                           + in[k+(j-2)*nz+i*nz*ny] + in[k+(j+2)*nz+i*nz*ny]
                           + in[(k-2)+j*nz+i*nz*ny] + in[(k+2)+j*nz+i*nz*ny] )
                      + coeff[3] *
                          (  in[k+j*nz+(i-3)*nz*ny] + in[k+j*nz+(i+3)*nz*ny]
                           + in[k+(j-3)*nz+i*nz*ny] + in[k+(j+3)*nz+i*nz*ny]
                           + in[(k-3)+j*nz+i*nz*ny] + in[(k+3)+j*nz+i*nz*ny] );

            }
        }
    }
}

void bench3DStencilC()
{
    fd3DStencilC(in, resC, NX, NY, NZ);
}
```

Original Code

```
template<typename T>
void stencil3DMap(Elt3D<T> in, Elt3D<T> &out)
{
    //! in  C(i, j, k) => Ct(k, i, j)
    T tmpOut = 2 * (T)in - (T)out

             + coeff[0] * (T)in
             + coeff[1] *
                 (  in(0, 0, -1) + in(0, 0, +1)
                  + in(-1, 0, 0) + in(+1, 0, 0)
                  + in(0, -1, 0) + in(0, +1, 0) )
             + coeff[2] *
                 (  in(0, 0, -2) + in(0, 0, +2)
                  + in(-2, 0, 0) + in(+2, 0, 0)
                  + in(0, -2, 0) + in(0, +2, 0) )
             + coeff[3] *
                 (  in(0, 0, -3) + in(0, 0, +3)
                  + in(-3, 0, 0) + in(+3, 0, 0)
                  + in(0, -3, 0) + in(0, +3, 0) );

    out = tmpOut;

}

template<typename priT>
void fd3DStencilCt(priT *out)
{
    typedef typename Pri2CtType<priT>::CtType T;

    //! in, (x, y, z) => (y(Row), z(Col), x(Page))
    Vec3D<T> vin(in, _NX, _NY, _NZ);

    //! out, (x, y, z) => (y(Row), z(Col), x(Page))
    Vec3D<T> vout(out, _NX, _NY, _NZ);

    rmap(stencil3DMap<T>)(vin, vout);
}

void bench3DStencilCt()
{
    fd3DStencilCt(resCt);
}
```

Ct Code

Back Projection

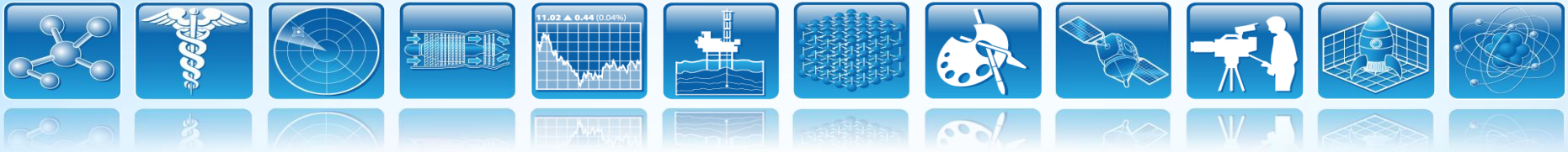
```
void backProjection(float *prArr, float *imgArr)
{
    for (int iy = 0; iy < numPixelsH; iy++) {
        float y = (float)iy + yMin;
        for (int ix = 0; ix < numPixelsW; ix++) {
            float x = (float)ix + xMin;
            float sum = 0.0f;
            //! For each pixel, sum of scans from all angles
            for (int thta = 0; thta < numAngles; thta ++) {

                float angle  = thta * aveAngle;
                float sinAng = sin(angle);
                float cosAng = cos(angle);
                float xN = (x - xCen)/xCen;                     //! New coordinate
                float yN = (y - yCen)/yCen;
                float t  = xN * cosAng + yN * sinAng;           //! Offset distance
                float mb = t * midPoint + midPoint;             //! Actual receiver
                int   lb = static_cast<int>(floorf(mb));        //! Lower  reciever
                int   hb = static_cast<int>(ceilf(mb));         //! Higher reciever

                float frac = mb - lb;                           //! Factor for line
                if(lb >= numRays) lb -= 1;
                if(hb >= numRays) hb -= 1;

                if ((lb >= 0) && (lb < numRays)) {
                    //! lb's weight is (1.0f-frac)
                    sum += (1.0f-frac) * prArr[thta*numRays+lb];    //! Accumulate proj
                }
                if ((hb >= 0) && (hb < numRays)) {
                    //! hb's weight is frac
                    sum += frac * prArr[thta*numRays+hb];           //! Accumulate proj
                }
                //! Output result
                imgArr[iy*numPixelsW+ix] = sum;

            }//! End of for(thta)
        }//! End of for(ix)
    }//! End of for(iy)
}
```

Original Code

```
void backProjectImp(Vec2D<F32> vProj, Vec2D<F32> &vImag)
{
    //! pre-compute sin, cos out of loop
    Vec<F32> idx = index<F32>(0.0f, (float)numAngles, 1.0f);
    Vec<F32> vAngle = idx * aveAngle;
    Vec<F32> vSinAng = sin( vAngle );
    Vec<F32> vCosAng = cos( vAngle );

    Vec2D<F32> idX = index2D<F32>(xMin, numPixelsW, 1.0f, numPixelsH, (Bool)true);
    Vec2D<F32> idY = index2D<F32>(yMin, numPixelsH, 1.0f, numPixelsW, (Bool)false)
    vImag = Vec2D<F32>::create(0.0f, numPixelsH, numPixelsW);

    Vec2D<Size> indxI = Vec2D<Size>::create(-1, numPixelsH, numPixelsW);

    Size i;
    //! For each pixel, sum of scans from all angles
    _for(i = (_Size)0, i < numAngles, i++ ){

        F32 cosAng(vCosAng[i]);
        F32 sinAng(vSinAng[i]);

        Vec2D<F32> pXn = (idX - xCen)/xCen;                         //! New coordi
        Vec2D<F32> pYn = (idY - yCen)/yCen;
        Vec2D<F32> pT  = pXn * cosAng + pYn * sinAng;               //! Offset dis
        Vec2D<F32> vMb = pT * midPoint + midPoint;                  //! Actual rec
        Vec2D<F32> vLb = floor(vMb);                                //! Lower  rec
        Vec2D<F32> vHb = ceiling(vMb);                              //! Higher rec

        Vec2D<F32> vFrac = vMb - vLb;                               //! Factor for

        Vec2D<Size> vLbi = (Vec2D<Size>)vLb;
        Vec2D<Size> vHbi = (Vec2D<Size>)vHb;

         indxI += 1;

            //! vLb's weight is (1.0f-vFrac)
        vImag += (-vFrac + 1.0f) * vProj[Vec2D<Tuple<2,Size> >(indxI, vLbi)];

        //! vHb's weight is vFrac
        vImag += vFrac * vProj[Vec2D<Tuple<2, Size> >(indxI, vHbi)];

    }_endFor
}
```

Ct Code

What can it be used for?



Bioinformatics

- Genomics and sequence analysis
- Molecular dynamics

Engineering design

- Finite element and finite difference simulation
- Monte Carlo simulation

Financial analytics

- Option and instrument pricing
- Risk analysis

Oil and gas

- Seismic reconstruction
- Reservoir simulation

Medical imaging

- Image and volume reconstruction
- Analysis and computer-aided detection (CAD)

Visual computing

- Digital content creation (DCC)
- Physics engines and advanced rendering
- Visualization
- Compression/decompression

Signal and image processing

- Computer vision
- Radar and sonar processing
- Microscopy and satellite image processing

Science and research

- Machine learning and artificial intelligence
- Climate and weather simulation
- Planetary exploration and astrophysics

Enterprise

- Database search
- Business information

Conclusions

- Dynamic code generation can significantly reduce the performance impact of high levels of abstraction and modularity
  - Elimination of cost for late binding of functions
  - Freezing of control flow once parameters known
  - Freezing size of dynamically size data structures
- Dynamic code generation can support high performance in productivity languages
- Dynamic code generation allows for radical program-driven hardware-adaptive restructuring of data flow at fine granularities
  - In order to improve data locality while respecting limits of microarchitecture
  - Support autotuning as mainstream programming technology

Fini

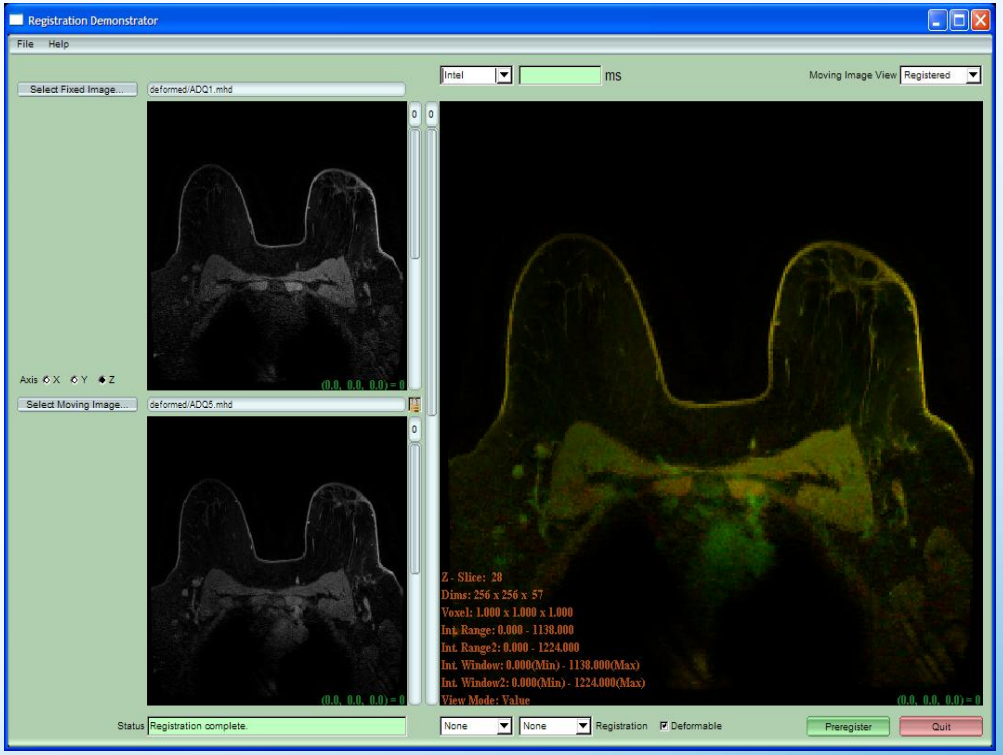
Questions?

http://www.intel.com/go/ct

Medical imaging: deformable registration

Contrast optical flow

- Portable to both multicore and manycore
- Ct Technology implementation of basic optical flow approx 2x faster than an also parallelized ITK baseline
- Algorithmic improvements (multigrid) give additional approx 10x speedup.



How Does it Really Work?

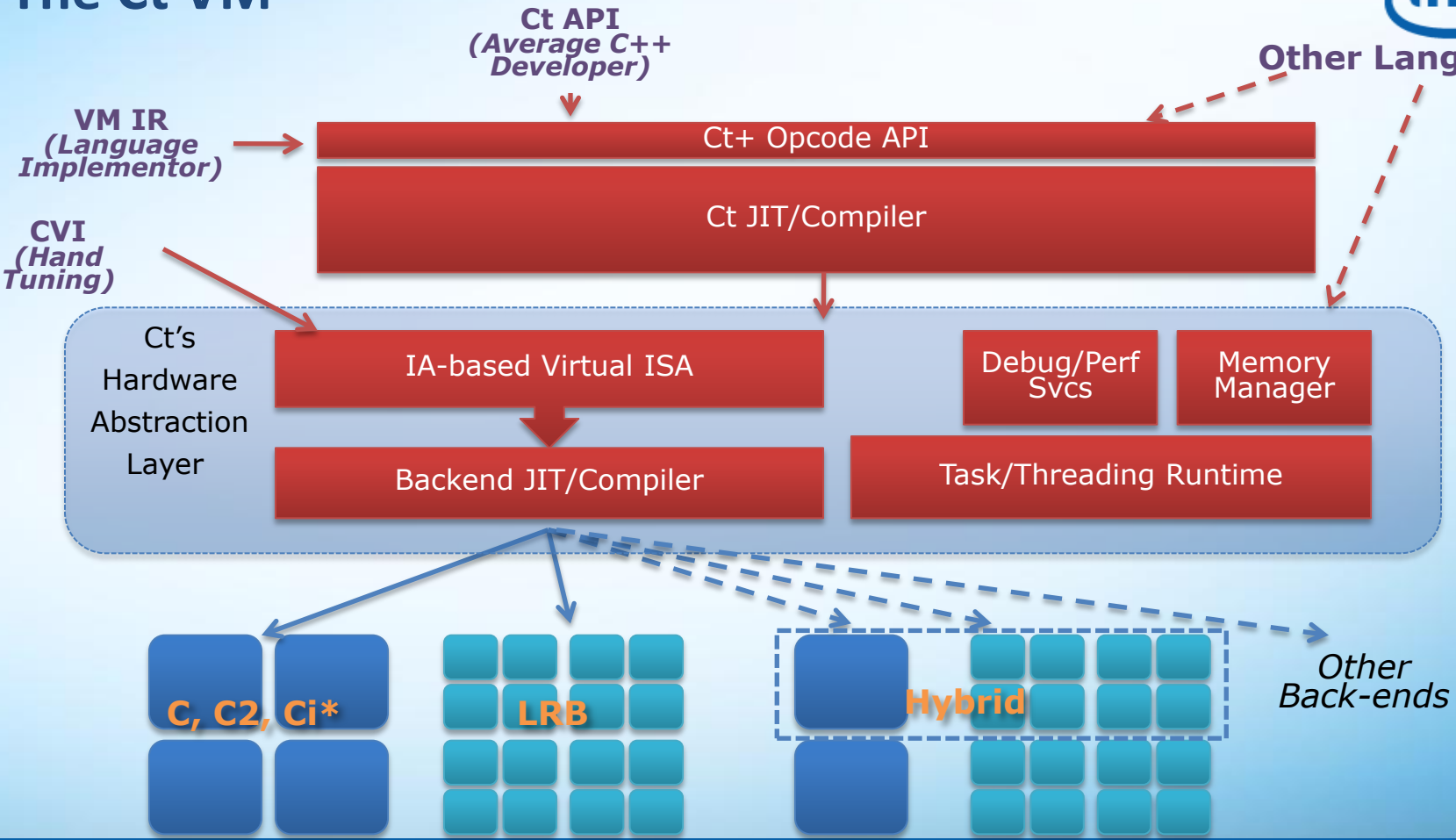
Ct is really a high-level APIs…

…that streams opcodes to an optimizing virtual machine

The source (front-end) can be anything:

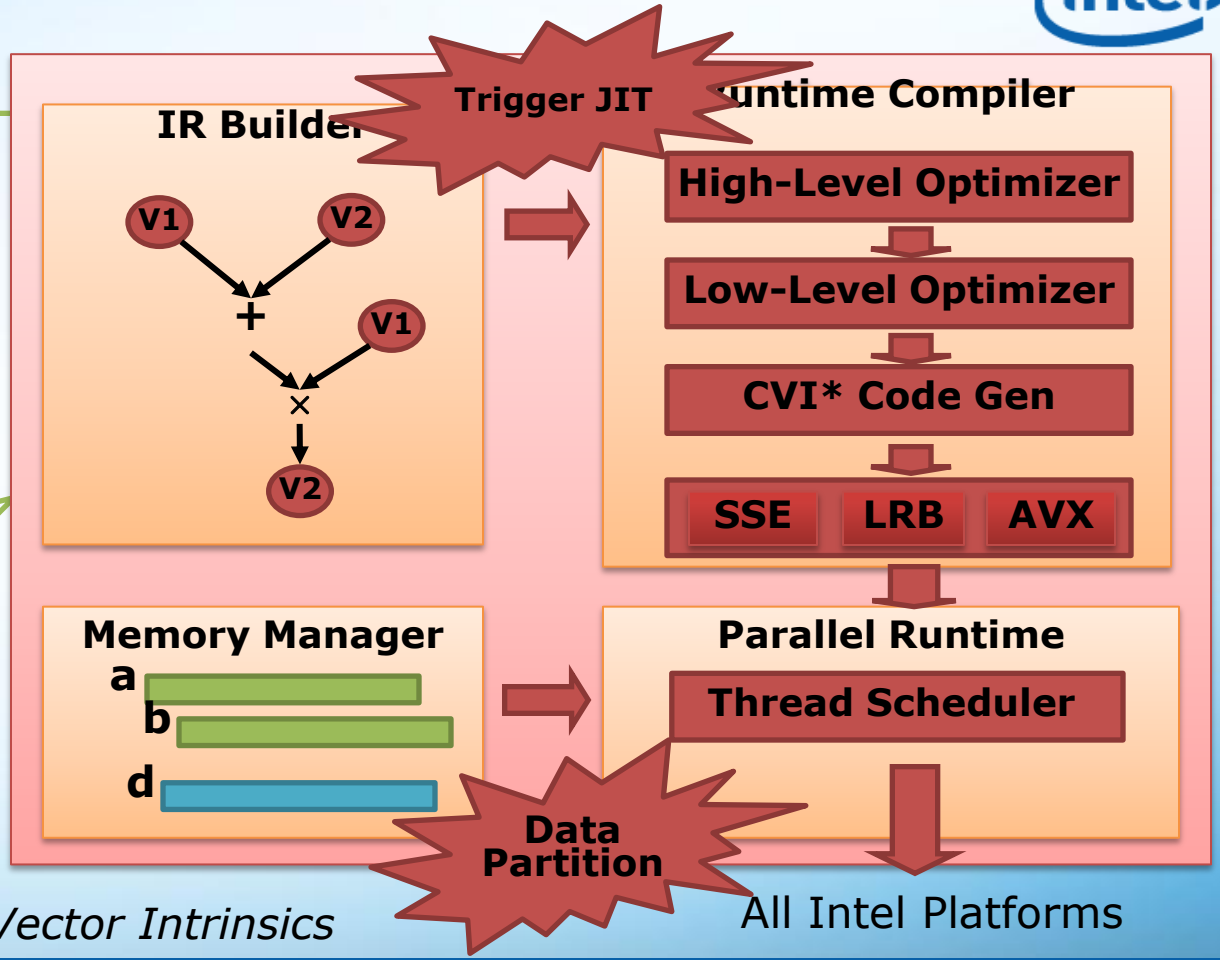
- A new language
- A bytecode parser
  - Experiments with Python, HLSL
- An application-specific library
- A compiler front-end

The Ct VM



Runtime Evaluation Model: Generative Programming

```
float src1[], src2[], dest[];

Vec<F32>a(src1,N), b(src2,N);
rcall(foo)(a, b)
…
foo(Vec<F32> a, Vec<F32> b) {
  Vec<F32> c = a + b;
  Vec<F32> d = c * a;
  return;
}
```



* *CVI = Converged Vector Intrinsics*

Ct Virtual Machine Interface

- The VM interface
  - Human readable, editable C++-like form
  - Extensible bytecode interface for compact storage
  - Extensible with compiler metadata for encoding "domain knowledge"
  - *Not* specific to C++ Ct API!
- Also, lower-level "unmanaged" interface: CVI
- A reusable infrastructure

  → Allow others focus on value add for vertical vs. infrastructure

```
defFunc vaddf32(
  in = vec<F32> a,
       vec<F32> b;
  out = vec<F32> c)
{
  c = add<vec<F32>>(a, b);
}
```

Infrastructure to close the productivity gap!

Productivity/Scripting Language Proof Points

- Excel front-end via VB
- Python byte-code translator
- HLSL compiler

…more to come!