Now *What*?

Sean Parent | Principal Scientist

Beauty

C++11 Standard
1338 Pages

Adobe

C++11 Standard
1338 Pages

C++98 Standard
757 Pages

Beauty

- Nearly every addition to the language is intended to make it *easier* for developers to write beautiful code
- Does it succeed?

STL was intended to be an example of beautiful code

Beauty

```
struct _LIBCPP_VISIBLE piecewise_construct_t { };
//constexpr
extern const piecewise_construct_t piecewise_construct;// = piecewise_construct_t();

template <class _T1, class _T2>
struct _LIBCPP_VISIBLE pair
{
    typedef _T1 first_type;
    typedef _T2 second_type;

    _T1 first;
    _T2 second;

    // pair(const pair&) = default;
    // pair(pair&&) = default;

    _LIBCPP_INLINE_VISIBILITY pair() : first(), second() {}

    _LIBCPP_INLINE_VISIBILITY pair(const _T1& __x, const _T2& __y)
        : first(__x), second(__y) {}

    template<class _U1, class _U2>
        _LIBCPP_INLINE_VISIBILITY
        pair(const pair<_U1, _U2>& __p
#ifndef _LIBCPP_HAS_NO_ADVANCED_SFINAE
                 ,typename enable_if<is_constructible<_T1, _U1>::value &&
                                    is_constructible<_T2, _U2>::value>::type* = 0
#endif
                                      )
            : first(__p.first), second(__p.second) {}

    _LIBCPP_INLINE_VISIBILITY
    pair(const pair& __p)
         NOEXCEPT (is_nothrow_copy_constructible<first_type>::value &&
    get(pair<_T1, _T2>&& __p) _NOEXCEPT {return _VSTD::forward<_T2>(__p.second);}
```

Beauty

```
#endif // _LIBCPP_HAS_NO_RVALUE_REFERENCES
};

template <size_t _Ip, class _T1, class _T2>
_LIBCPP_INLINE_VISIBILITY inline
typename tuple_element<_Ip, pair<_T1, _T2> >::type&
get(pair<_T1, _T2>& __p) _NOEXCEPT
{
    return __get_pair<_Ip>::get(__p);
}

template <size_t _Ip, class _T1, class _T2>
_LIBCPP_INLINE_VISIBILITY inline
const typename tuple_element<_Ip, pair<_T1, _T2> >::type&
get(const pair<_T1, _T2>& __p) _NOEXCEPT
{
    return __get_pair<_Ip>::get(__p);
}

#ifndef _LIBCPP_HAS_NO_RVALUE_REFERENCES

template <size_t _Ip, class _T1, class _T2>
_LIBCPP_INLINE_VISIBILITY inline
typename tuple_element<_Ip, pair<_T1, _T2> >::type&&
get(pair<_T1, _T2>&& __p) _NOEXCEPT
{
    return __get_pair<_Ip>::get(_VSTD::move(__p));
}

#endif  // _LIBCPP_HAS_NO_RVALUE_REFERENCES

#endif  // _LIBCPP_HAS_NO_VARIADICS
```

Complete std::pair
372 Lines

Complete std::pair
372 Lines

The compiler provided the copy and move constructors

Beauty

- The language is too large for *anyone* to master
  - So *everyone* lives within a subset

Beauty

- The language is too large for *anyone* to master
    - So *everyone* lives within a subset

Is there a beautiful subset?

Beauty

- Is the library now intended to be primitive constructs?
- How would I use pair or tuple to define a point or euclidean vector class?

- I still can't write:

```
pair<int> x;
```

- Unless I define it myself:

```
template <typename T> using pair = pair<T, T>;
```

- How much does language and library complexity matter?

“We’re getting an error that has something to do with rvalue references and std::pair.”

Beauty

```
1>c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include\utility(163): error
C2220: warning treated as error - no 'object' file generated
1>          c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include\utility(247) : see
reference to function template instantiation
'std::_Pair_base<_Ty1,_Ty2>::_Pair_base<_Ty,int>(_Other1 &&,_Other2 &&)' being
compiled
1>          with
1>          [
1>
_Ty1=std::_Tree_iterator<std::_Tree_val<std::_Tmap_traits<Mondo::num32,Mondo::CP
hotoshopFormat *,std::less<Mondo::num32>,std::allocator<std::pair<const
Mondo::num32,Mondo::CPhotoshopFormat *>>,false>>>,
1>              _Ty2=bool,
1>
_Ty=std::_Tree_iterator<std::_Tree_val<std::_Tmap_traits<Mondo::num32,Mondo::CPh
otoshopFormat *,std::less<Mondo::num32>,std::allocator<std::pair<const
Mondo::num32,Mondo::CPhotoshopFormat *>>,false>>>,
1>
_Other1=std::_Tree_iterator<std::_Tree_val<std::_Tmap_traits<Mondo::num32,Mondo::
CPhotoshopFormat *,std::less<Mondo::num32>,std::allocator<std::pair<const
Mondo::num32,Mondo::CPhotoshopFormat *>>,false>>>,
1>              _Other2=int
```

Beauty

```
1>              _Traits=std::_Tmap_traits<Mondo::num32,Mondo::CPhotoshopFormat
*,std::less<Mondo::num32>,std::allocator<std::pair<const
Mondo::num32,Mondo::CPhotoshopFormat *>>,false>
1>          ]
1>          c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include\map(81) : see
reference to class template instantiation 'std::_Tree<_Traits>' being compiled
1>          with
1>          [
1>              _Traits=std::_Tmap_traits<Mondo::num32,Mondo::CPhotoshopFormat
*,std::less<Mondo::num32>,std::allocator<std::pair<const
Mondo::num32,Mondo::CPhotoshopFormat *>>,false>
1>          ]
1>          c:\p4\m1710\khopps\dpxcode4\shared\mondo\source\photoshop
\CPhotoshopFormat.h(35) : see reference to class template instantiation
'std::map<_Kty,_Ty>' being compiled
1>          with
1>          [
1>              _Kty=Mondo::num32,
1>              _Ty=Mondo::CPhotoshopFormat *
1>          ]
1>c:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include\utility(163): warning
C4800: 'int' : forcing value to bool 'true' or 'false' (performance warning)
```

Beauty

```
template<class U, class V> pair(U&& x, V&& y);
```

- For a pair<T, bool> what happens if we pass an int to y?
- Why would we pass an int?

Beauty

```
ADMStandardTypes.h: #define false   0
AGFConvertUTF.cpp:  #define false   0
ASBasic.h:          #define false   0
ASBasicTypes.h:     #define false   0
ASNumTypes.h:       #define false   0
ASTypes.h:          #define false   0
basics.h:           #define false   ((Bool32) 0)
common.h:           #define false   0
config_assert.h:    #define false   0
ConvertUTF.cpp:     #define false   0
CoreExpT.h:         #define false   0
ICCUtils.h:         #define false   0
isparameter.cpp:    #define false   0
PITypes.h:          #define false   FALSE
piwinutl.h:         #define false   FALSE
PSSupportPITypes.h: #define false   FALSE
stdbool.h:          #define false   false
t_9_017.cpp:        #define false   0
WinUtilities.h:     #define false   FALSE
```

<Placeholder>

- Insert your own beautiful code here.

Beauty

What we lack in beauty,
we gain in efficiency

Beauty

What we lack in beauty,
we gain in efficiency?

Truth

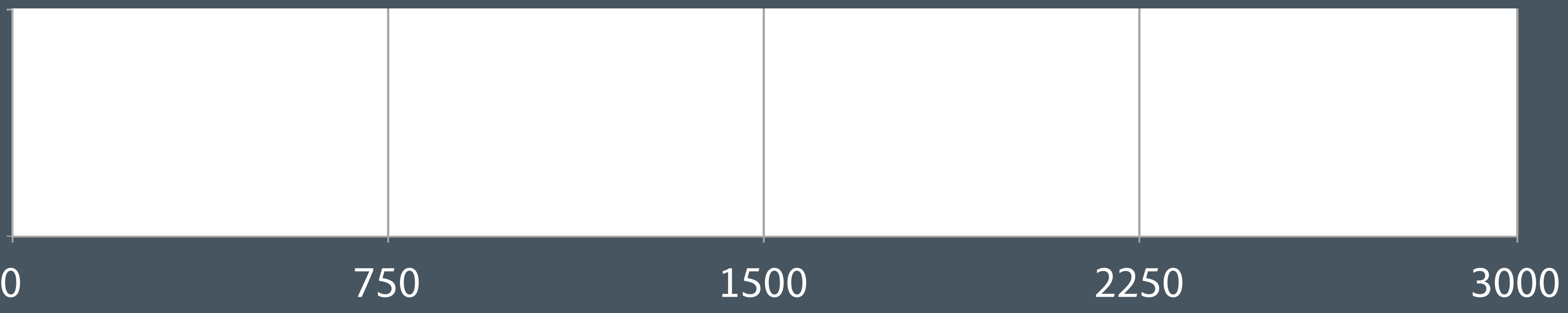
Demo

Adobe Revel

Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)

🟩 GPU 🟦 Vectorization 🟪 Multi-thread 🟥 Scalar

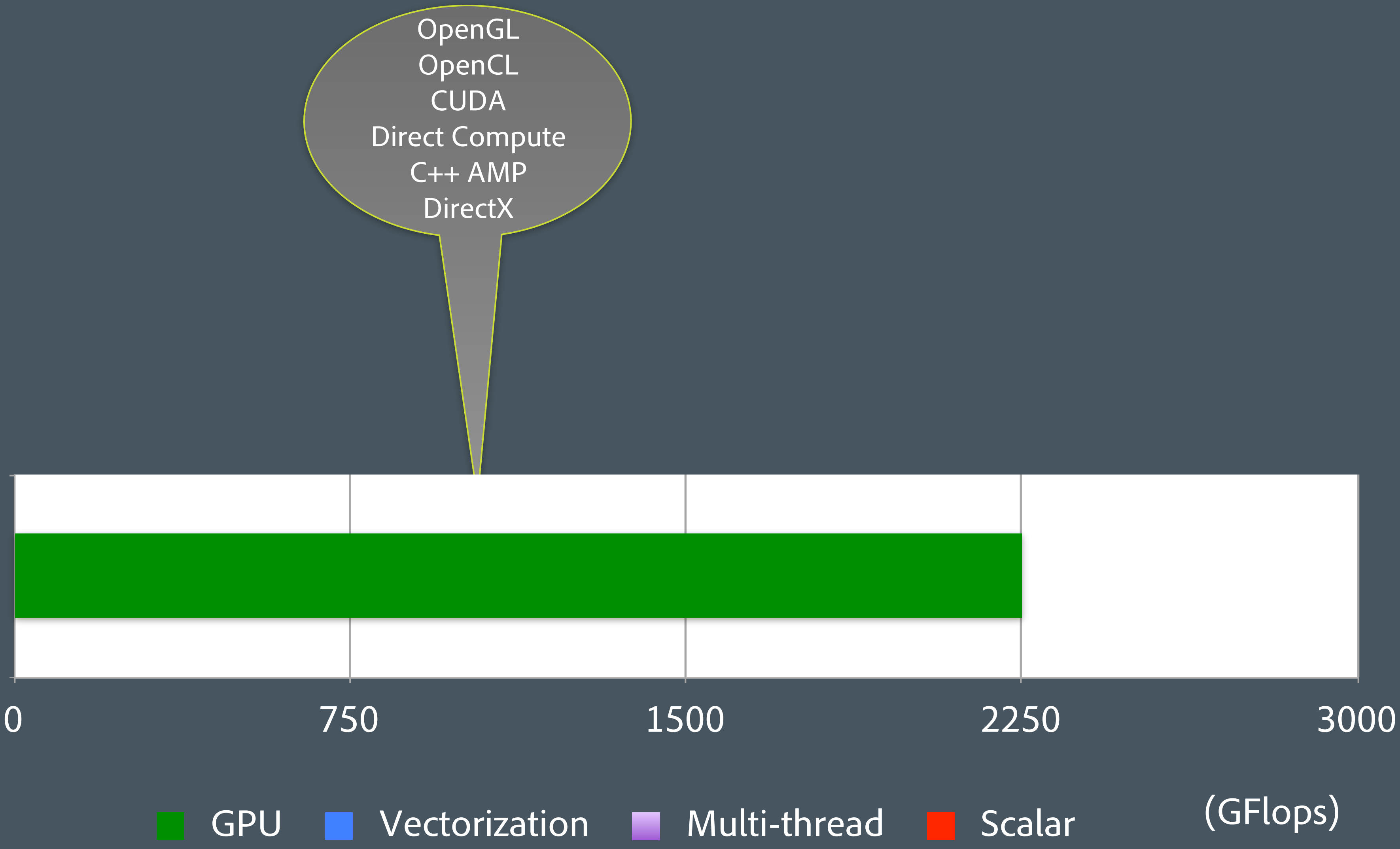
Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)



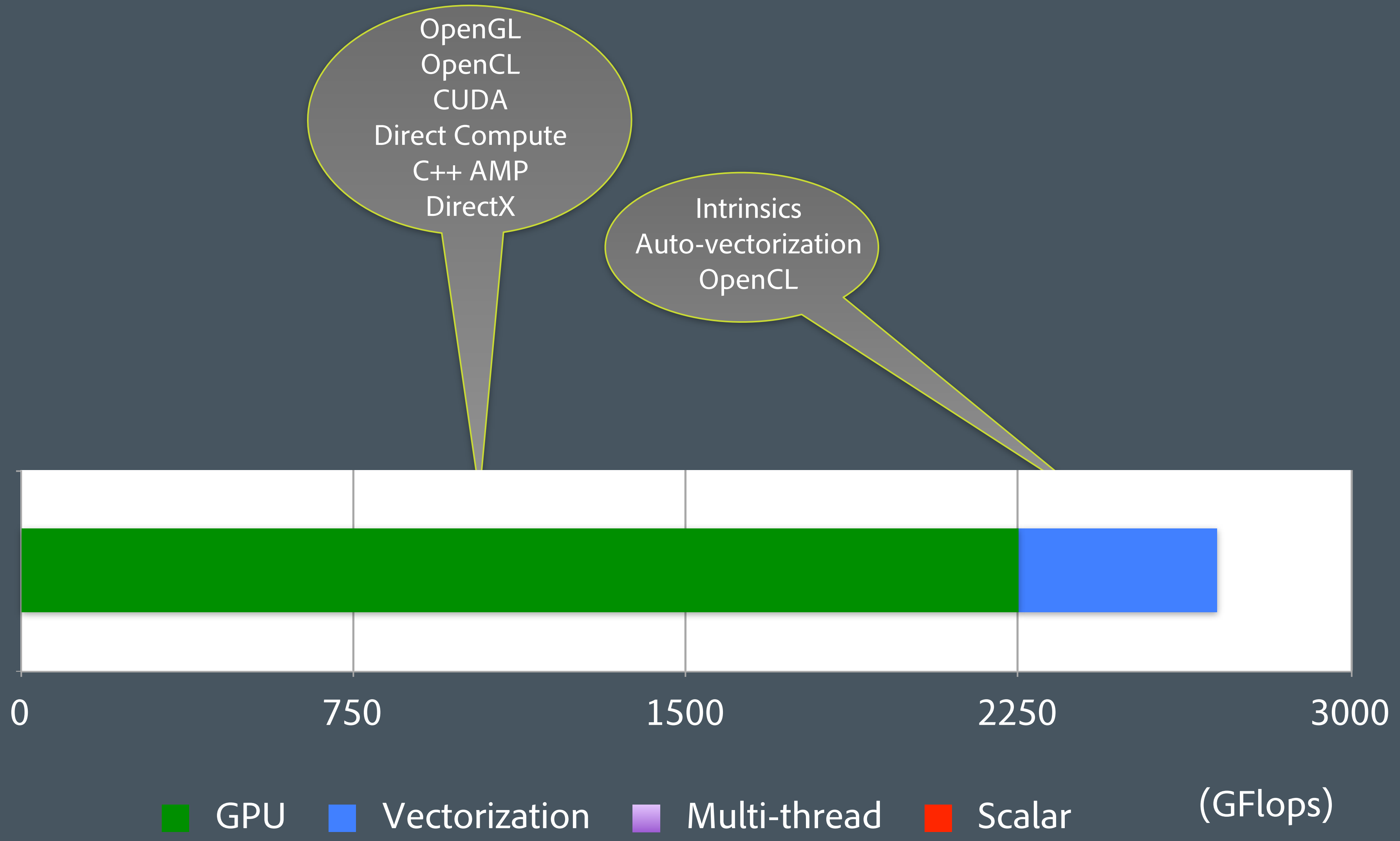
🟩 GPU 🟦 Vectorization 🟪 Multi-thread 🟥 Scalar

(GFlops)

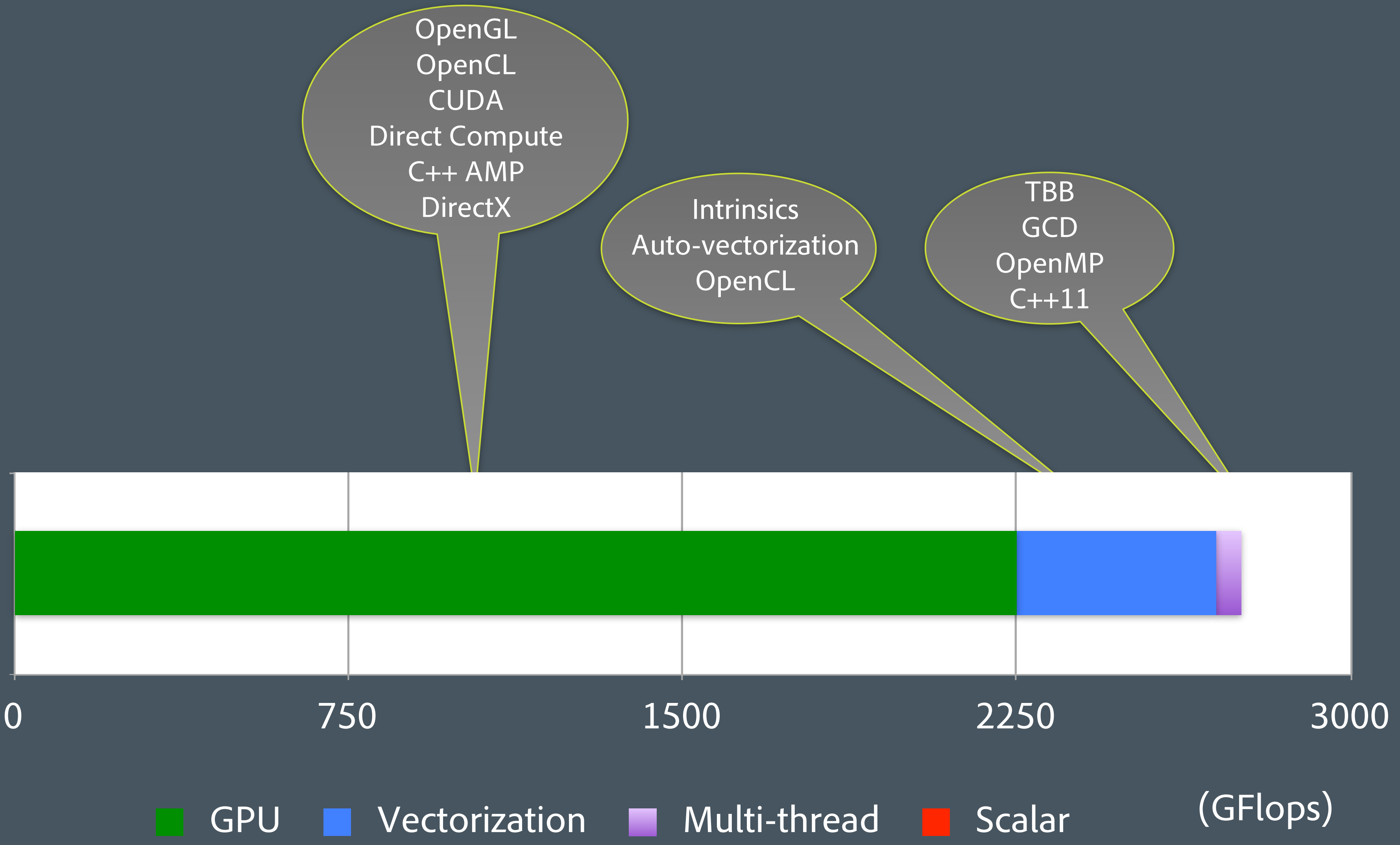
Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)



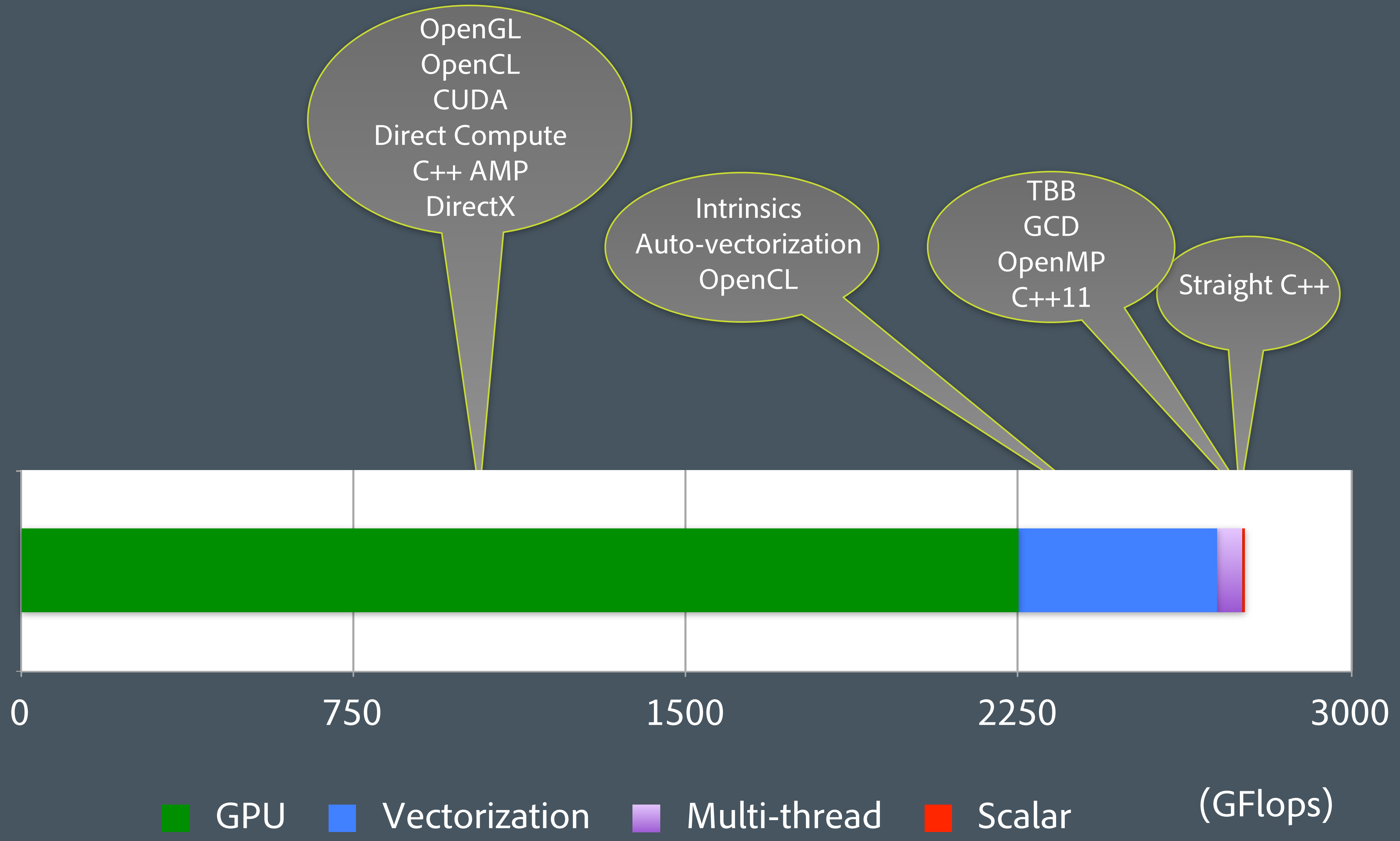
Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)



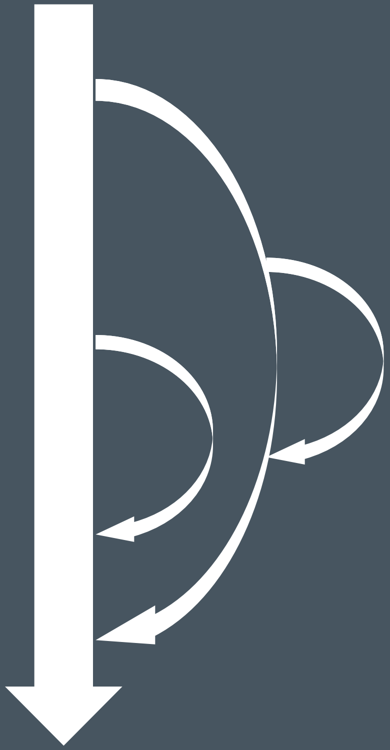
Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)

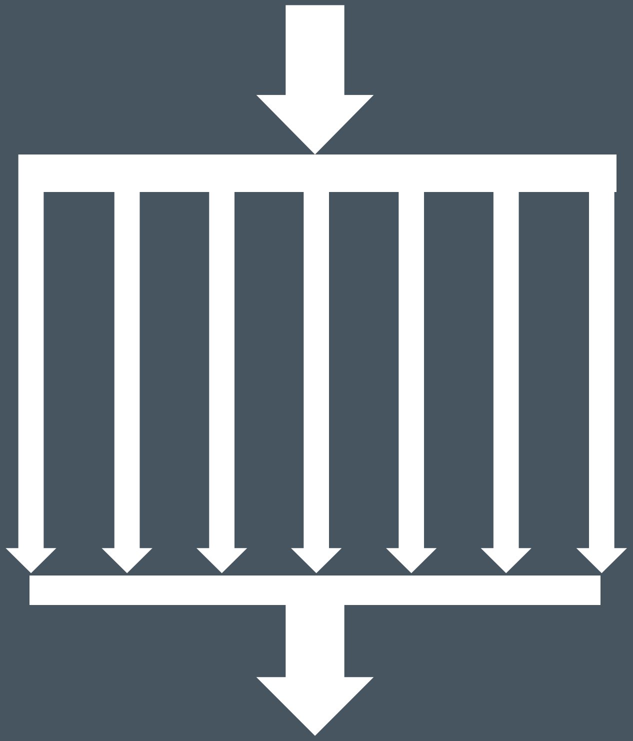


Desktop Compute Power (8-core 3.5GHz Sandy Bridge + AMD Radeon 6950)



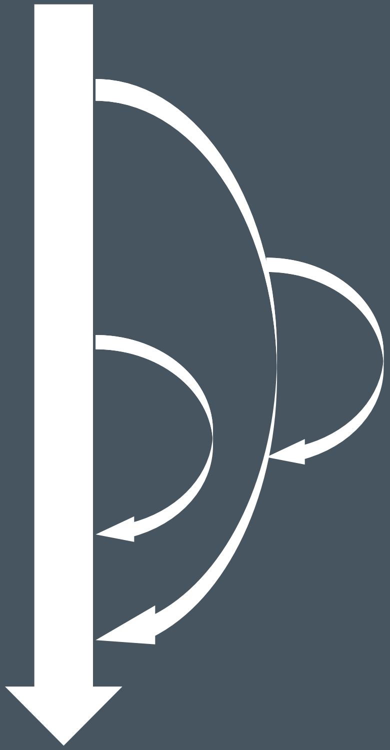
Two kinds of parallel



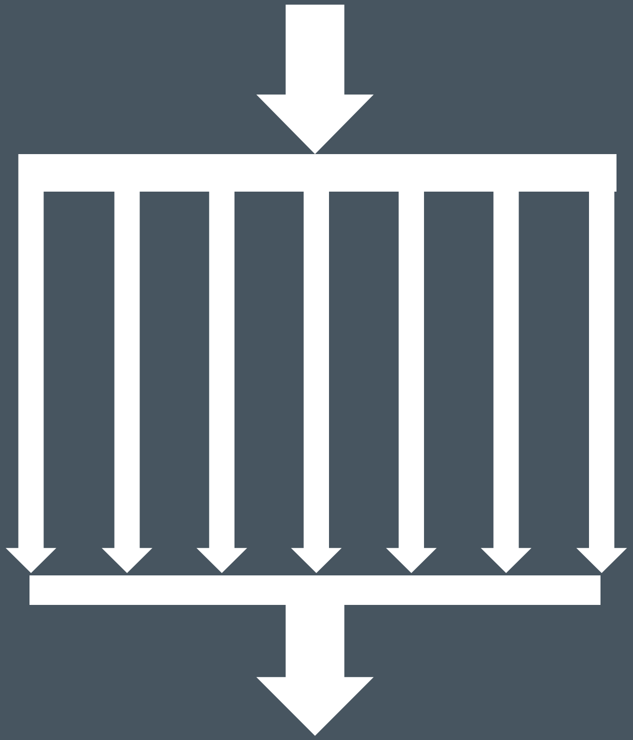


Two kinds of parallel

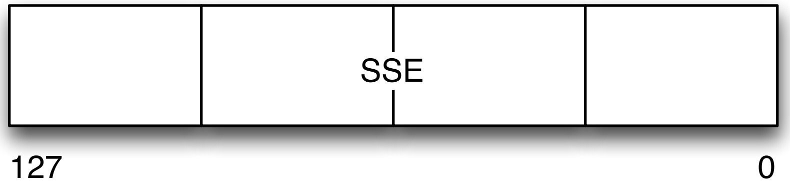
Functional



Data Parallel



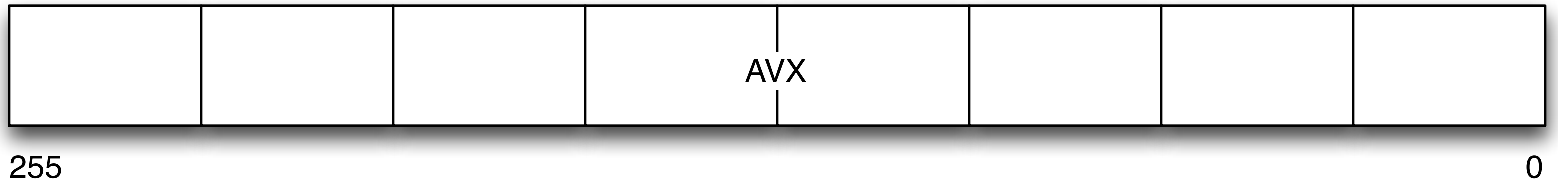
Vectorization



- Intrinsics: great speed potential, but…

```
__m128i vDst = _mm_cvttps_epi32(_mm_mul_ps(_mm_cvtepi32_ps(vSum0), vInvArea));
```

  - Moving target: MMX, SSE, SSE2, SSE3, SSE 4.1, SSE 4.2, AVX, AVX2, AVX3



- Solutions:
  - Auto-vectorization `#pragma SIMD`
  - CEAN `Dest[:] += src[start:length] + 2;`
  - OpenCL

Why Not Put Everything on the GPU?

Why Not Put Everything on the GPU?

Data Parallel	300	:	1

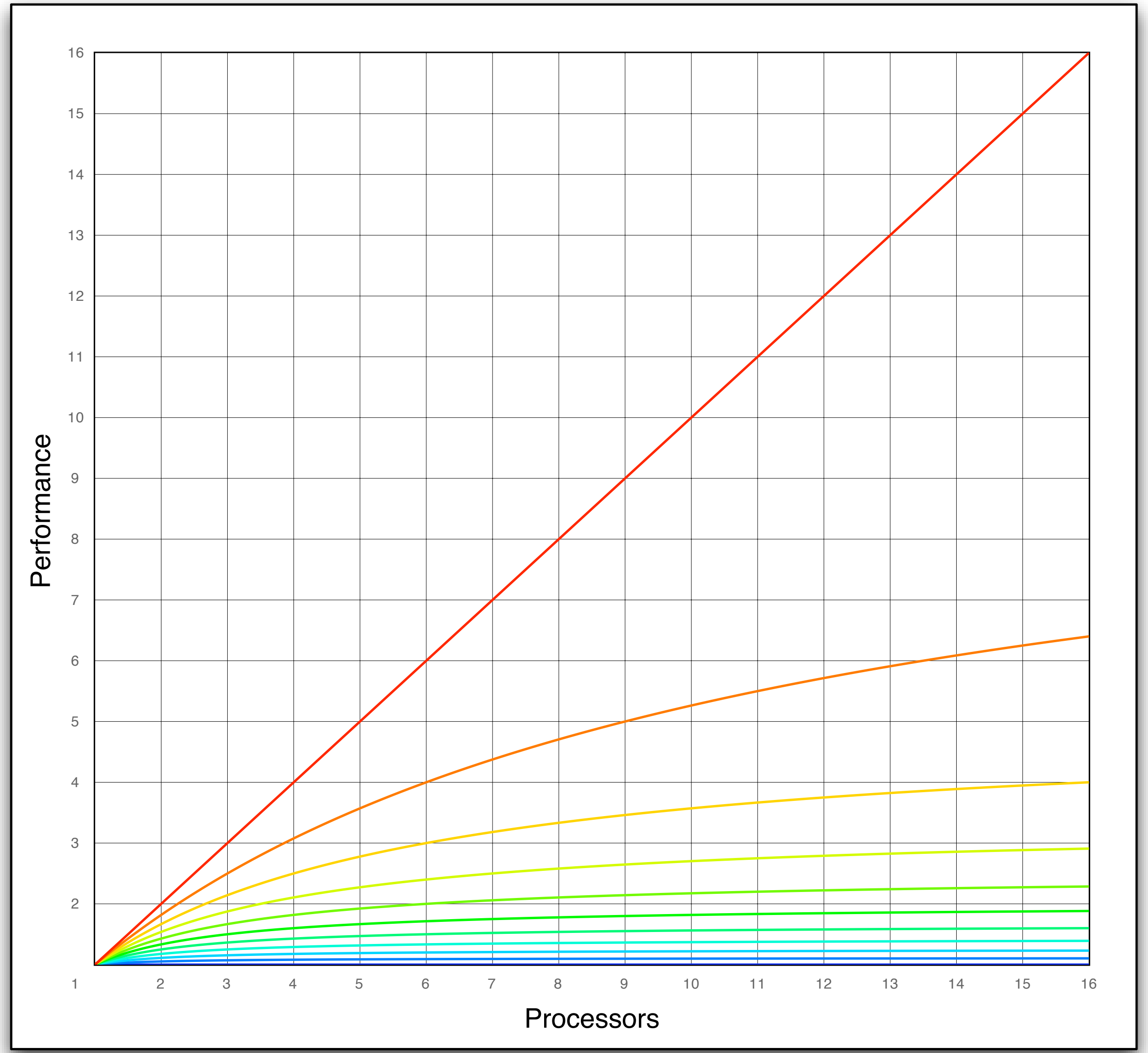
Why Not Put Everything on the GPU?

Data Parallel	300	:	1
Sequential	1	:	10

Truth

- That typical object oriented paradigms of using shared references to objects breaks down in a massively parallel environment
- Sharing implies either single threaded
  - Or synchronization

Amdahl's Law



Truth

- To utilize the hardware we need to move towards functional, declarative, reactive, and value semantic programming
- No raw loops

Truth

- Without addressing vectorization, GPGPU, and scalable parallelism, standard C++ is just a scripting system to get to the other 99% of the machine through other languages and libraries

Truth

- Without addressing vectorization, GPGPU, and scalable parallelism, standard C++ is just a scripting system to get to the other 99% of the machine through other languages and libraries

Do we need such a complex scripting system?

Goodness

Content Ubiquity

- Ubiquitous access to:
  - calendar
  - contacts
  - notes & tasks
  - e-mail (corporate and personal)
  - A full web experience
  - Music
    - iTunes Music Match
    - Spotify
    - Pandora
  - Movies
    - Netflix
    - Vudu
- Photos
  - Flickr
  - Facebook
  - Adobe Revel
- Documents
  - Google Docs
  - Microsoft Office
- Everything…

Content ubiquity is access to all your information, on all your devices, all of the time

The Problem

- Ubiquity has gone mainstream
  - A typical US household now has 3 TVs, 2 PCs, and 1 Smartphone
    - 1 in 3 households has an internet connected TV
  - A typical US worker has access to a PC at work or is provided an e-mail solution for communication
- The deluge of digital information has become a challenge to manage
  - How do I get this contract to my phone?
  - How do I get this video from my phone to my PC?
  - Which computer has the latest version of this photo?

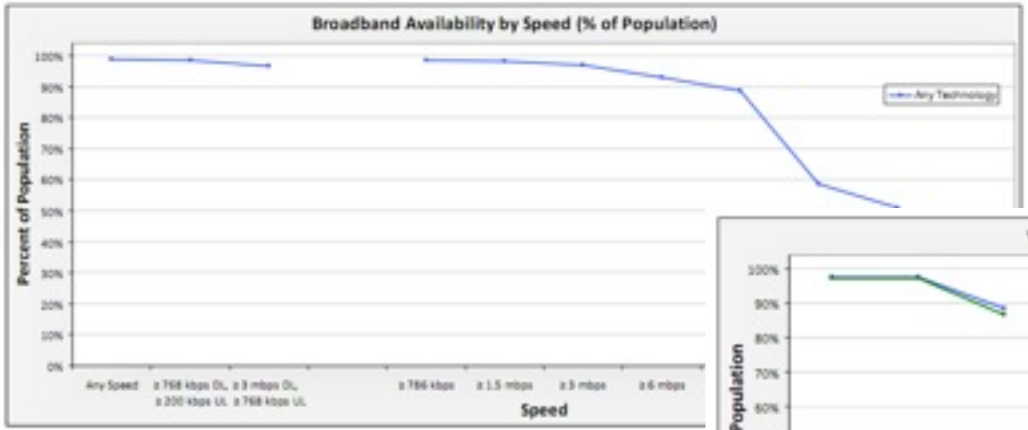
The Problem

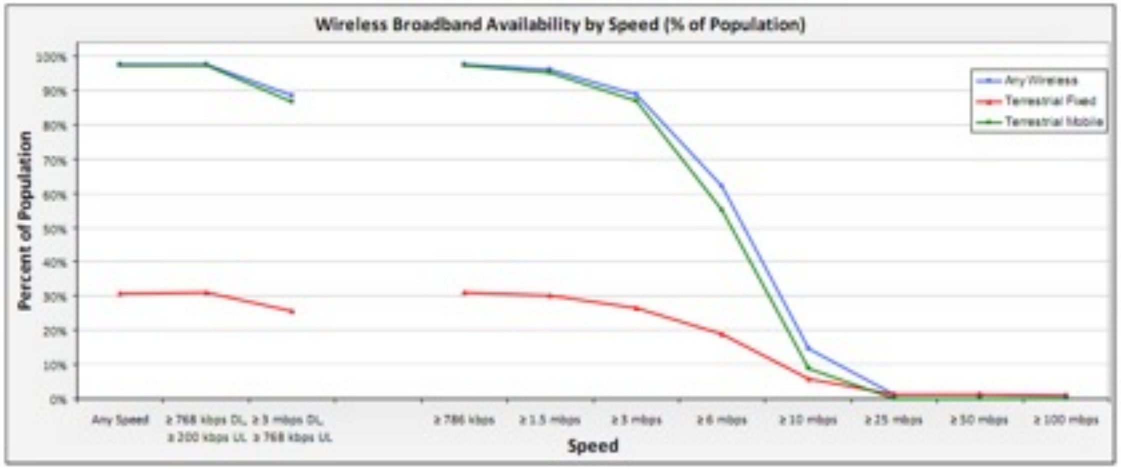
- Ubiquity has gone mainstream
  - A typical US household now has 3 TVs, 2 PCs, and 1 Smartphone
    - 1 in 3 households has an internet connected TV
  - A typical US worker has access to a PC at work or is provided an e-mail solution for communication
- The deluge of digital information has become a challenge to manage
  - How do I get this contract to my phone?
  - How do I get this video from my phone to my PC?
  - Which computer has the latest version of this photo?

Content ubiquity has become the expectation

The Technology is Here Now

- ≥ 3mbps broadband is available to 95% of the US population
- ≥ 3mbps mobile broadband is available to 85%
- US ranks 28th in broadband subscriptions per capita
  - Every other tier one market is ahead of US
  - France (12), Germany (19), UK (21), Japan (27)



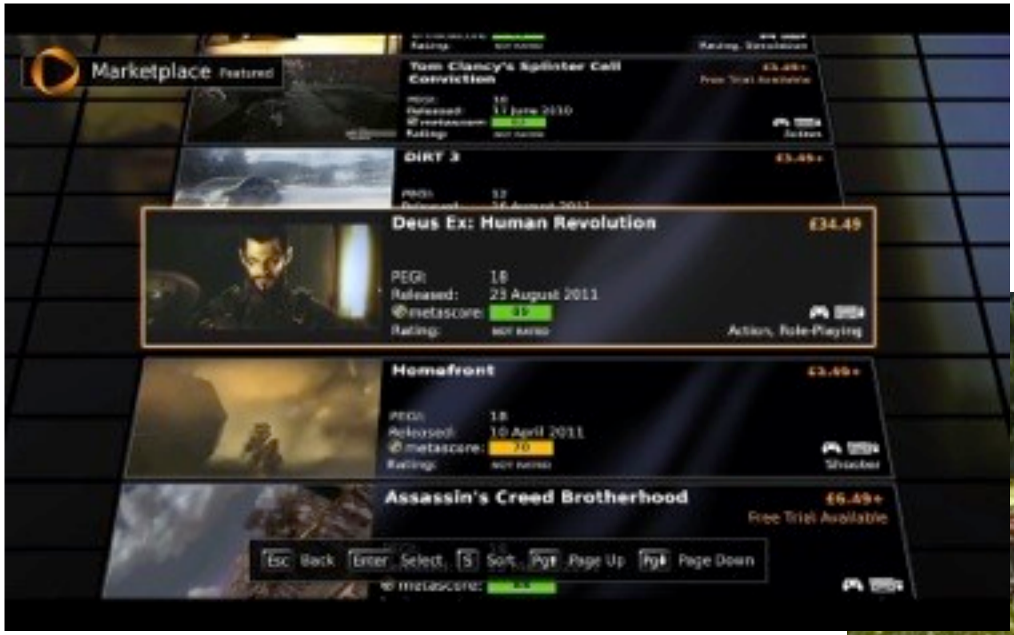


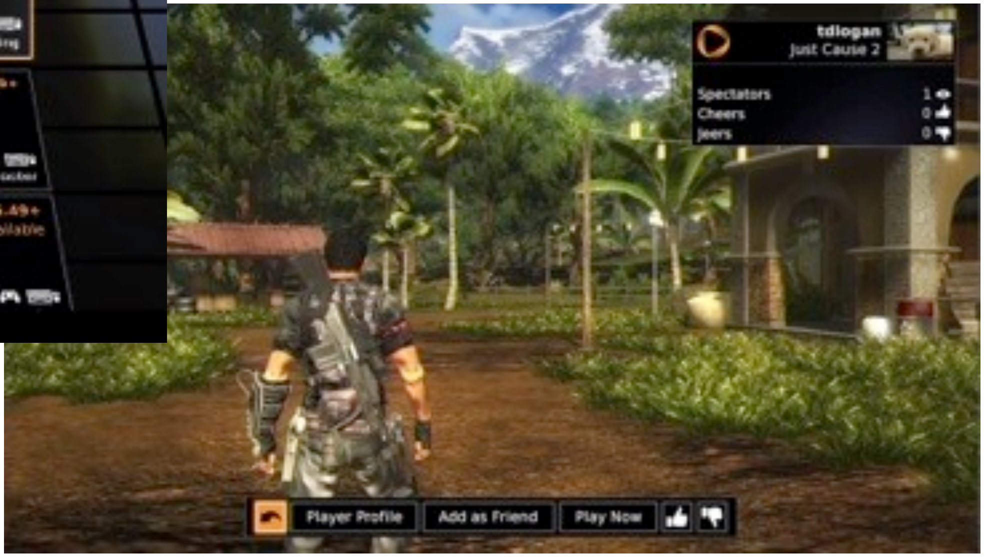
No Excuses

- Your data set is not too large

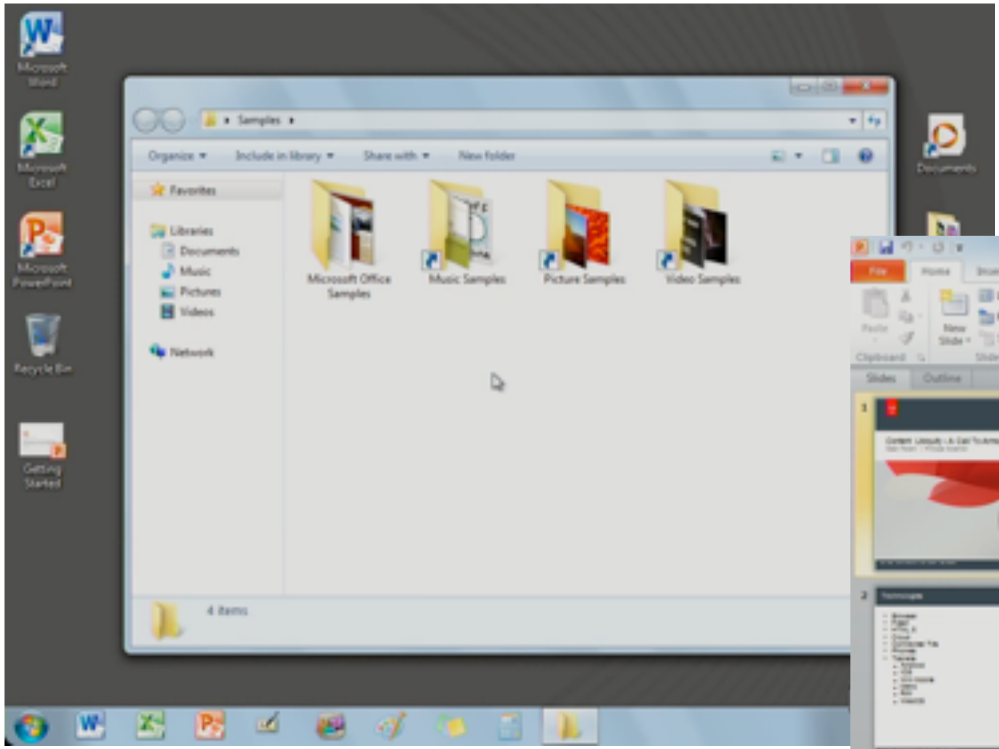
No Excuses

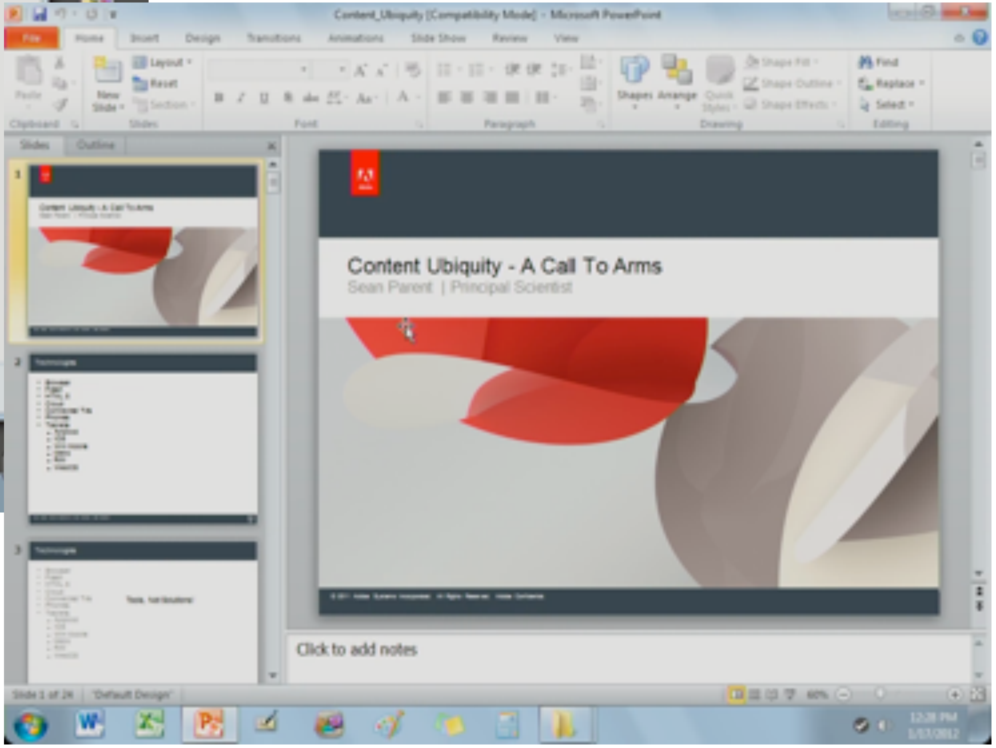
- Your application is not too interactive





No Excuses


Samples
Organize
Include in library
Share with
New folder
Favorites
Libraries
Documents
Music
Pictures
Videos
Network
Microsoft Office Samples
Music Samples
Picture Samples
Video Samples
4 items


Content_Ubiquity [Compatibility Mode] - Microsoft PowerPoint
File
Home
Insert
Design
Transitions
Animations
Slide Show
Review
View
Content Ubiquity - A Call To Arms
Sean Parent | Principal Scientist
Click to add notes

No Excuses

- Current hardware is capable enough
  - Typical: 2x1GHz cores, 512GB RAM, 32GB SSD, GPU, 802.11n
  - Revel runs the entire ACR image pipeline on an iPad 1 (half the above capabilities)

The Players

The Players

iCloud

SkyDrive

Dropbox

The Opportunity

- Focus on content ubiquity
  - all your content, instantly, on any available device
  - zero management overhead

- Users don't want to care about “The Cloud,” users want their content

The Challenge

- Content Ubiquity isn't a feature you can bolt-on
  - Dropbox, and similar technologies that require management and synchronization aren't the solution

- Achieving a seamless experience requires rethinking…
  - data model to support incremental changes
  - transactional models to support dynamic mobile environment
  - editor model to support partial editing (proxies, pyramid)
  - UI model to support touch, small devices, 10 foot interfaces

Content Ubiquity Opens the Door to Sharing and Collaboration

- If you can make changes available to other devices immediately then you can make changes available to other apps immediately (works with sandboxing technology)

- If you can make documents available to all your devices then you can make documents available to others - supporting both collaboration and sharing

New Products and New Technologies

- Start by putting yourself in todays customers shoes
  - Assume anything is possible
  - Build it

- Invest in technology
  - peer-to-peer
  - interactive streaming
  - proxy and pyramidal editing
  - transactional data-structures

Developer Pain

- The market is very fragmented
  - Windows, OS X, iOS, Android, Linux (for cloud service), Browsers, Roku,...

- And will become more so
  - Windows RT, ...

Developer Pain

- To provide a solution requires you write for multiple platforms

- And many vendors are focusing on proprietary technology to get to 99% of the machine

- C++ itself becomes a fragmented scripting system
  - Objective-C++, Managed C++

Developer Pain

- Vendor lock-in on commodity technologies only serves to slow development
  - including incorporating vendor specific technology that provides user benefit

Now What?

- C++Next
  - Simplicity
  - Standardize access to modern hardware

Adobe