The Future of GPU/Accelerator Programming Models

LLVM HPC 2015

Michael Wong (IBM)

michaelw@ca.ibm.com; http:://wongmichael.com

http://isocpp.org/wiki/faq/wg21:michael-wong

IBM and Canadian C++ Standard Committee HoD

OpenMP CEO

Chair of WG21 SG5 Transactional Memory , SG14 Games/Low Latency

Director, Vice President of ISOCPP.org

Vice Chair Standards Council of Canada Programming Languages

Acknowledgement and Disclaimer

- Numerous people internal and external to the original OpenMP group, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk.
- I even lifted this acknowledgement and disclaimer from some of them.
- But I claim all credit for errors, and stupid mistakes. **These are mine, all mine!**

Legal Disclaimer

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM, PowerPC and the IBM logo are trademarks or registered trademarks of IBM or its subsidiaries in the United States and other countries.
- Other company, product, and service names may be trademarks or service marks of others.

Agenda

- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model

OpenMP Mission Statement changed in 2013

- OpenMP's new mission statement
  - "Standardize directive-based multi-language high-level parallelism that is performant, productive and portable"
  - Updated from
    - "Standardize and unify shared memory, thread-level parallelism for HPC"

OpenMP in Clang update

- I Chair Weekly OpenMP Clang review WG (Intel, IBM, AMD, TI, Micron) to help speedup OpenMP upstream into clang: April 2015-on going
    - –Joint code reviews, code refactoring
    - –Delivered full OpenMP 3.1 into Clang 3.7 (default lib is still GCC OpenMP)
    - –Added U of Houston OpenMP tests into clang
    - –IBM team Delivered changes for OpenMP RT for PPC, other teams added their platform/architecture
    - –Released Joint design on Multi-device target interface for LLVM to llvm-dev for comment

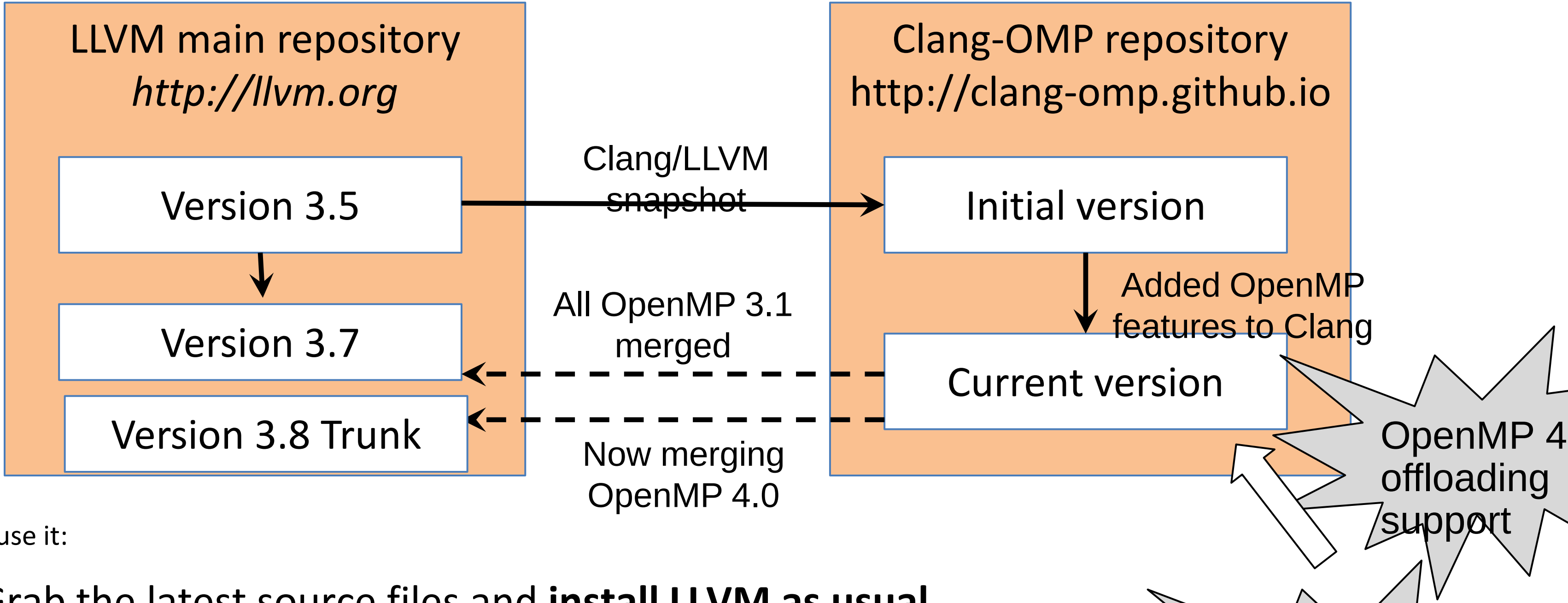
–LLVM developer Conf Oct 2015 talk:

- http://llvm.org/devmtg/2015-10/slides/WongBataev-OpenMPGPUAcceleratorsComingOfAgeInClang.pdf
- https://www.youtube.com/watch?v=dCdOaL3asx8&list=PL_R5A0lGi1AA4Lv2bBFSwhgDaHvvpVU21&index=18

Many Participants/companies

- Ajay Jayaraj, TI
- Alexander Musman, Intel
- Alex Eichenberger, IBM
- Alexey Bataev, Intel
- Andrey Bokhanko, Intel
- Carlo Bertolli, IBM
- Eric Stotzer, TI
- Guansong Zhang, AMD
- Hal Finkel, ANL
- Ilia Verbyn, Intel
- James Cownie, Intel
- Yaoqing Gao, IBM
- Kelvin Li, IBM
- Kevin O’Brien, IBM
- Samuel Antao, IBM
- Sergey Ostanevich, Intel
- Sunita Chandrasekaran, UH
- Michael Wong, IBM
- Wang Chan, IBM
- Robert Ho, IBM
- Wael Yehia, IBM
- Ettore Tiotto, IBM
- Melanie Ullmer, IBM
- Kevin Smith, Intel

The codebase



LLVM main repository

http://llvm.org

Version 3.5

Version 3.7

Version 3.8 Trunk

Clang/LLVM snapshot

All OpenMP 3.1 merged

Now merging OpenMP 4.0

Clang-OMP repository

http://clang-omp.github.io

Initial version

Added OpenMP features to Clang

Current version

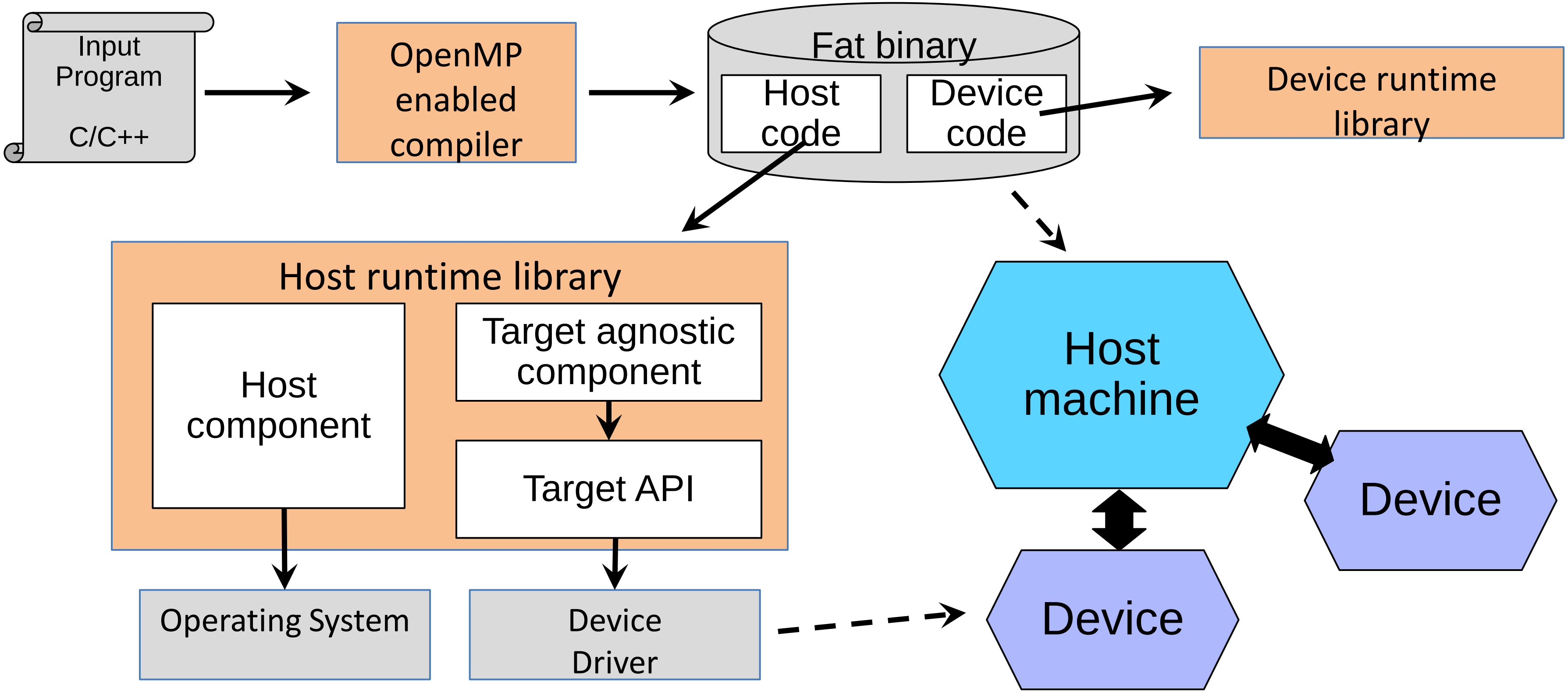
OpenMP 4 offloading support

- How to use it:
  - Grab the latest source files and **install LLVM as usual**
  - Use the right options to **specify host and targ**et machines, e.g.:

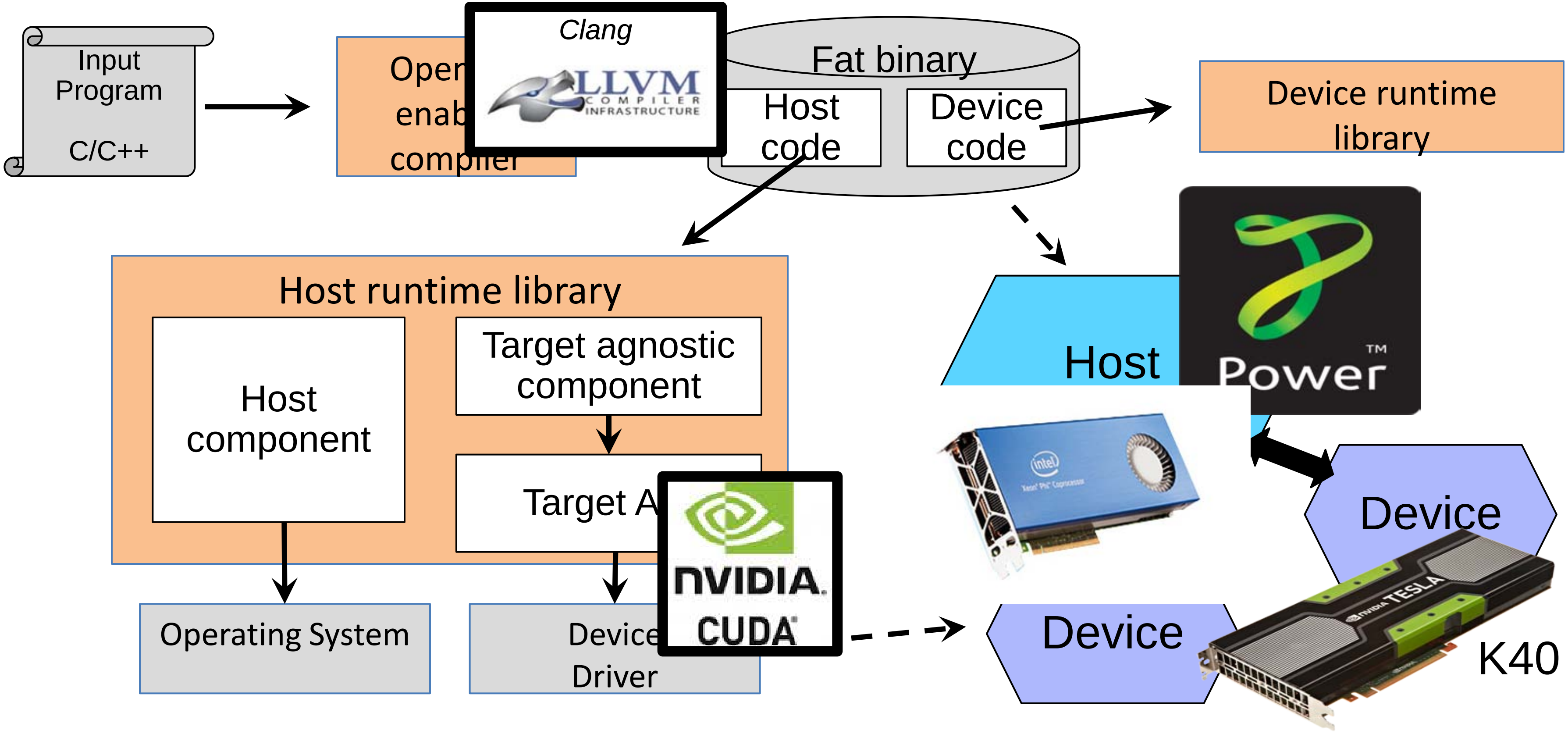
```
$ clang –fopenmp –target powerpc64le-ibm-linux-gnu –mcpu pwr8
        –omptargets=nvptx64sm_35-nvidia-cuda <source files>
```

OpenMP 4.5 support

Offloading in OpenMP – Impl. components



Offloading in OpenMP – Impl.



Input Program

C/C++

Open
enab
compiler

Clang

Fat binary

Host code

Device code

Device runtime library

Host runtime library

Host component

Target agnostic component

Target A

Operating System

Device Driver

Host

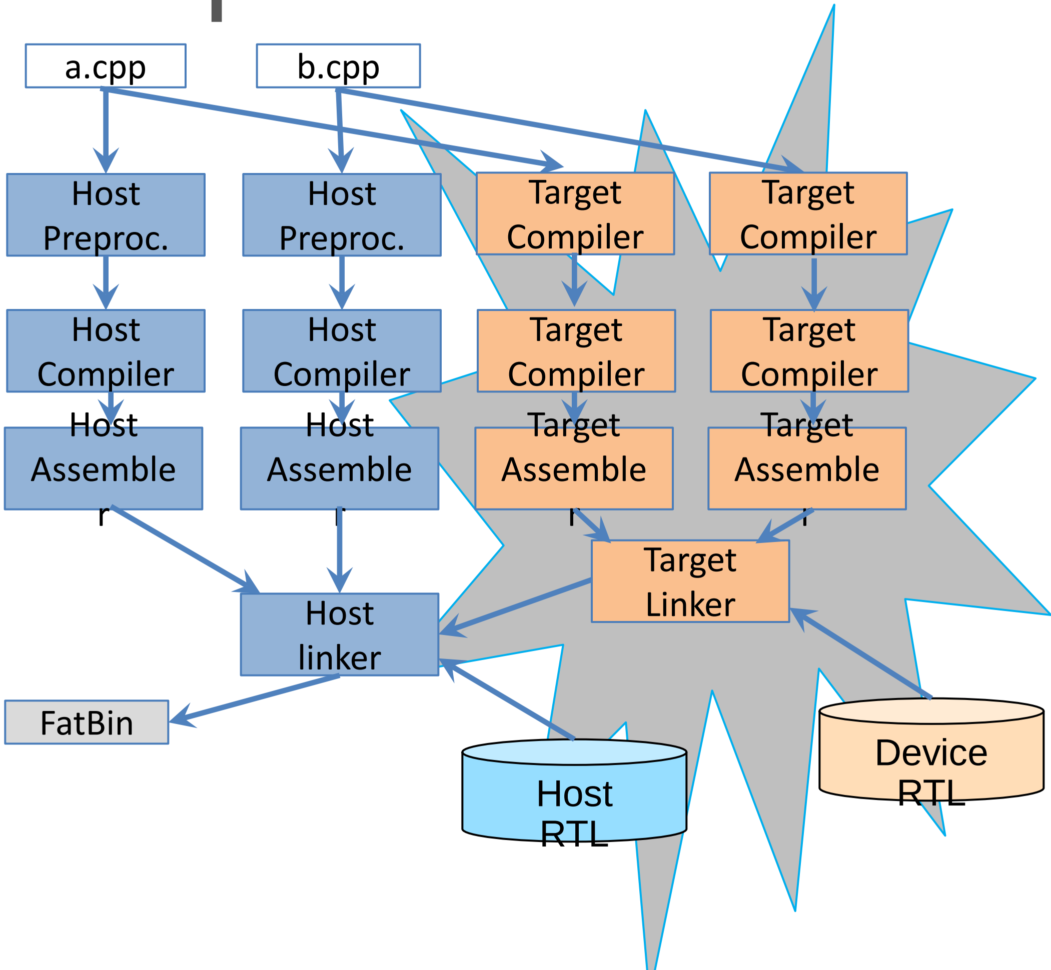
Device

Device

K40

Clang with OpenMP

- Compiler actions:
  - **Driver** preprocesses input source files **using host/target preprocessor**
    - Header files may be in different places
    - We may revisit this in the future
  - For each source file, the driver spawns **a job using the host** toolchain and an **additional job for each target** specified by the user
  - Flags informing the frontend that we are compiling code **for a target** so **only the relevant target regions are considered**
  - **Target linker creates a self-contained** (no undefined symbols) image file
  - **Target image file is embedded "as is"** by the host linker into the host fat binary
  - The **host linker** is provided with information to **generate the symbols required by the RTL**

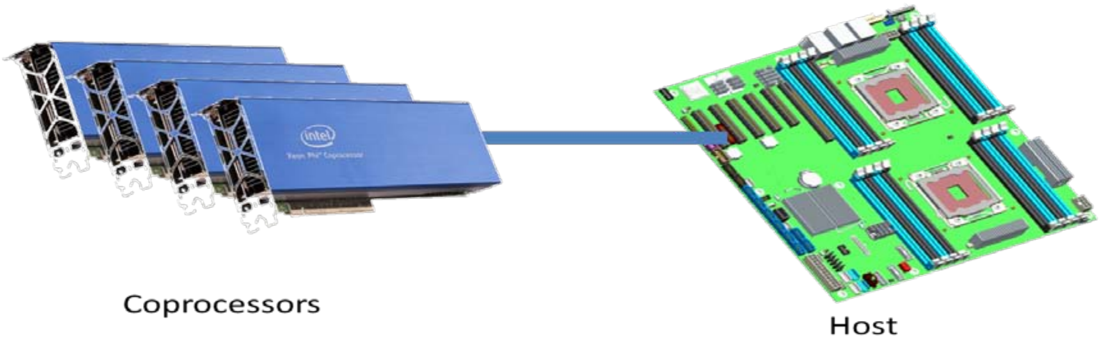


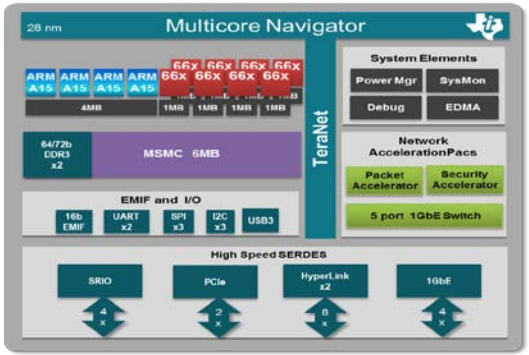
Offloading in Clang: Current Status

- Initial implementation available at https://github.com/clang-omp/clang_trunk
- First patches are committed to trunk
  - Support for target constructs parsing/sema/codegen for host
- Several patches are under review
  - Support for new driver option
  - Offloading descriptor registration and device codegen

heterogeneous device model

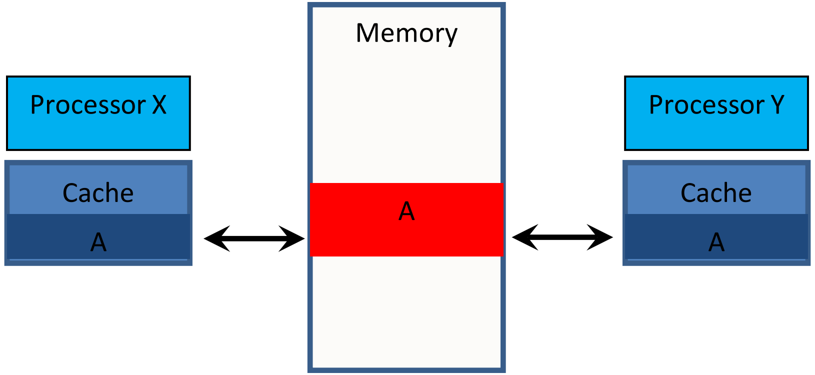
- OpenMP 4.0 supports accelerators/coprocessors
- Device model:
  - one host
  - multiple acclerators / coprocessors of the same kind





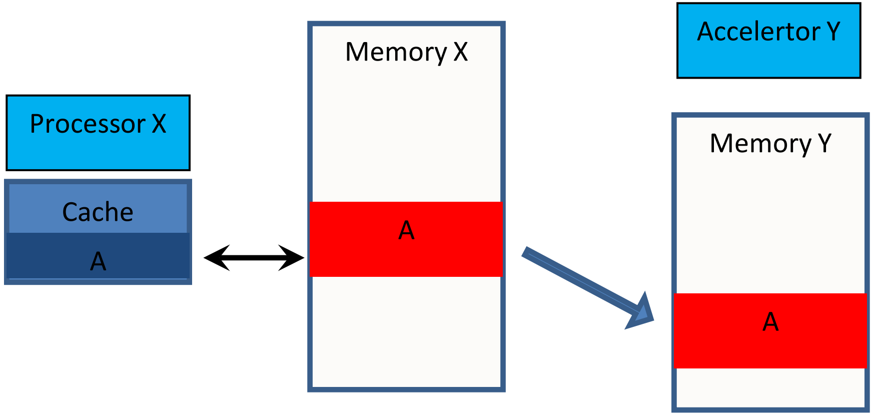
Data mapping: shared or distributed memory

Shared memory



- The corresponding variable in the device data environment *may* share storage with the original variable.
- Writes to the corresponding variable may alter the value of the original variable.

Distributed memory



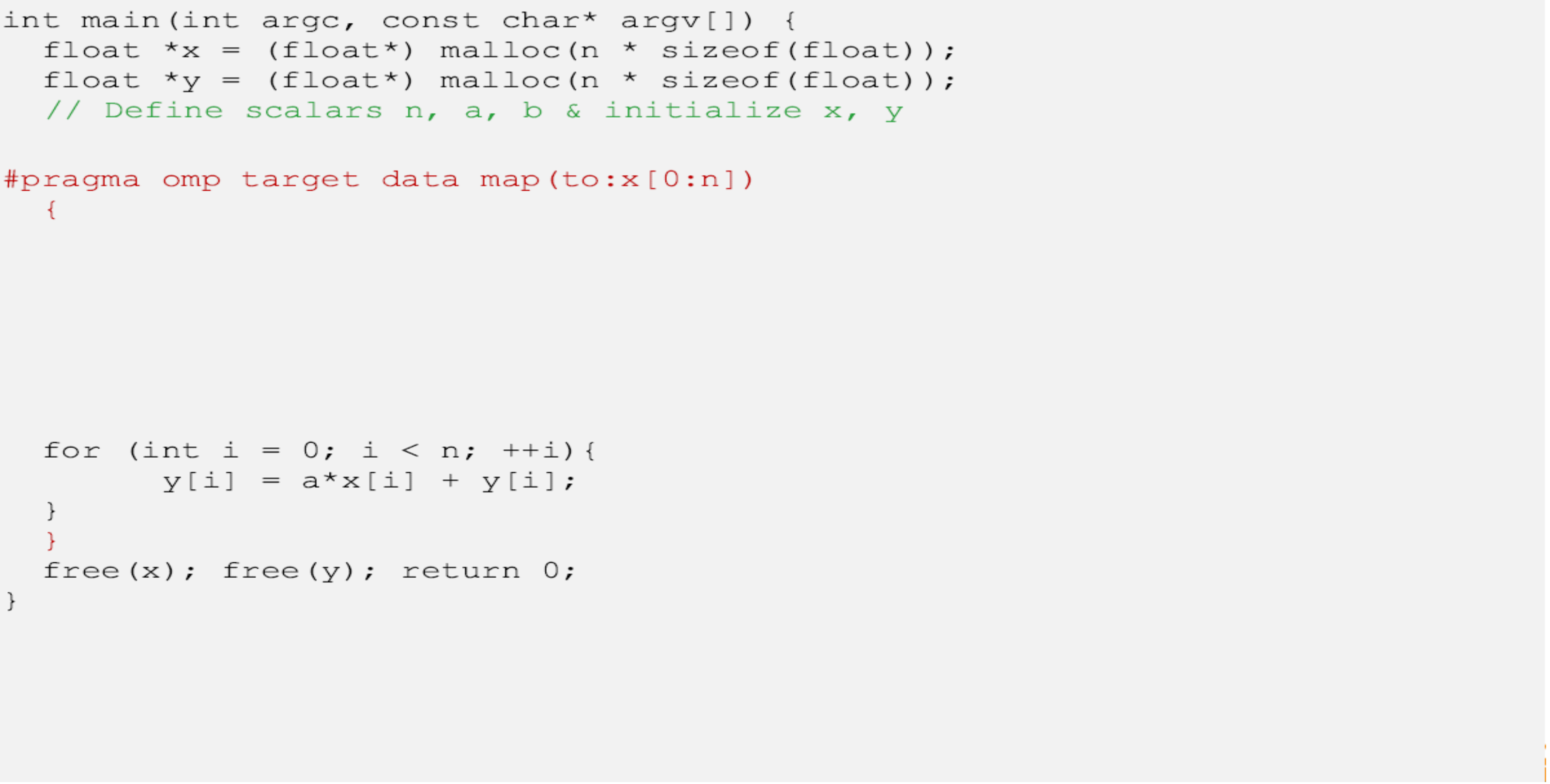
OpenMP 4.0 Device Constructs

- Execute code on a target device
  - **omp target** *[clause[[,] clause],…]*
    structured-block
  - **omp declare target**
    [function-definitions-or-declarations]
- Map variables to a target device
  - **map** *([map-type:] list) // map clause*
    *map-type := **alloc** | **tofrom** | **to** | **from***
  - **omp target data** *[clause[[,] clause],…]*
    structured-block
  - **omp target update** *[clause[[,] clause],…]*
  - **omp declare target**
    [variable-definitions-or-declarations]
- Workshare for acceleration
  - **omp teams** *[clause[[,] clause],…]*
    structured-block
  - **omp distribute** *[clause[[,] clause],…]*
    for-loops

SAXPY: Serial (host)

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y











  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

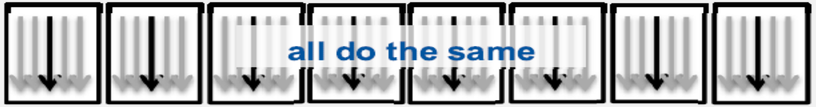
SAXPY: Serial (host)



```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {








  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
  }
  free(x); free(y); return 0;
}
```

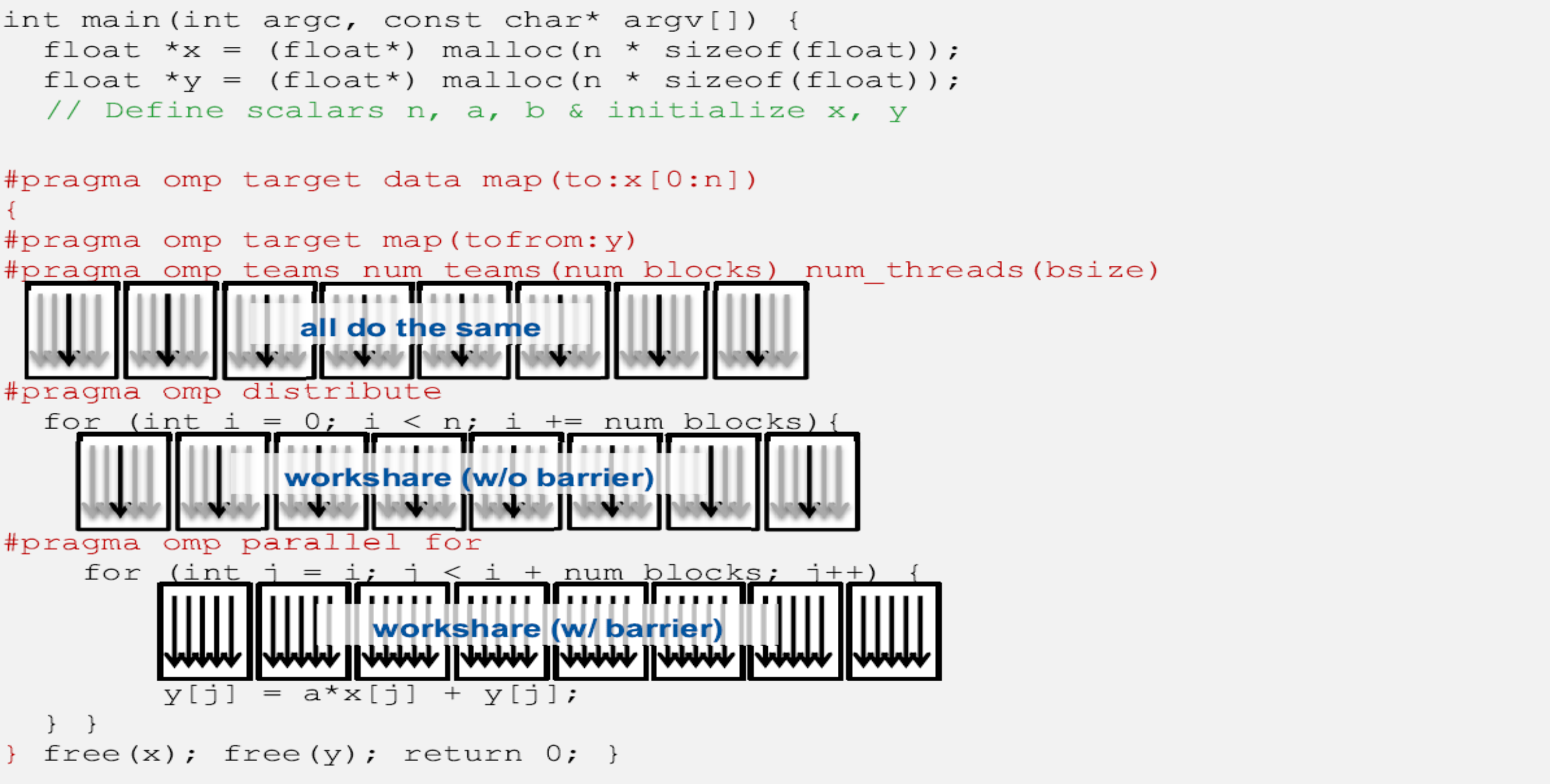
SAXPY: Coprocessor/Accelerator

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(nthreads)
```



```
  for (int i = 0; i < n; i += num_blocks){
    for (int j = i; j < i + num_blocks; j++) {
        y[j] = a*x[j] + y[j];
  } }
  }
  free(x); free(y); return 0;
}
```

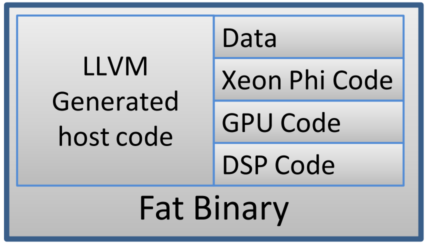
SAXPY: Coprocessor/Accelerator



```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)

#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){

#pragma omp parallel for
     for (int j = i; j < i + num_blocks; j++) {

        y[j] = a*x[j] + y[j];
  } }
} free(x); free(y); return 0; }
```

Building Fat Binary

- Clang generates objects for each target
- Target toolchains combine objects into target-dependent binaries
- Host linker combines host + target-dependent binaries into an executable (Fat Binary)
- New driver command-line option -omptargets=T1,…,Tn



LLVM Generated host code

Data

Xeon Phi Code

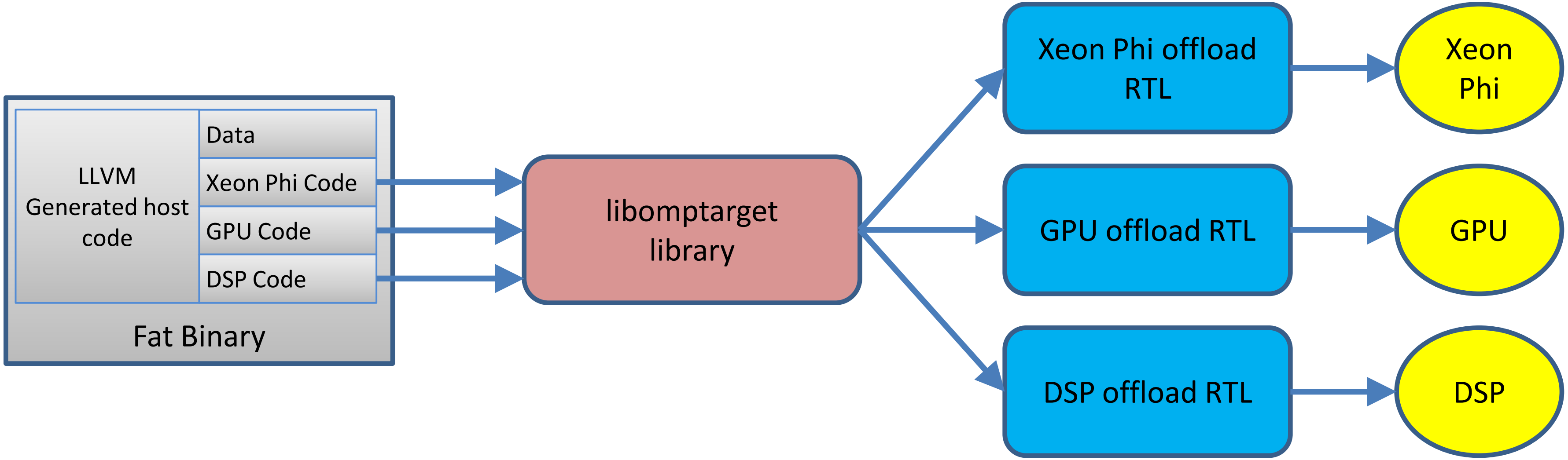
GPU Code

DSP Code

Fat Binary

```
clang -fopenmp -omptargets=nvptx64-nvidia-cuda,x86-pc-linux-gnu foo.c bar.c -o foobar.bin
```

Heterogeneous Execution of Fat Binary



Libomptarget and offload RTL

- Source code available at https://github.com/clang-omp/libomptarget
- Planned to be upstreamed
- Supported platforms
  - libomptarget
    - Platform neutral implementation (tested on Linux for x86-64, PowerPC*)
    - NVIDIA* (Tested with CUDA* compilation tools V7.0.27)
  - Offload target RTL
    - x86-64, PowerPC, NVIDIA

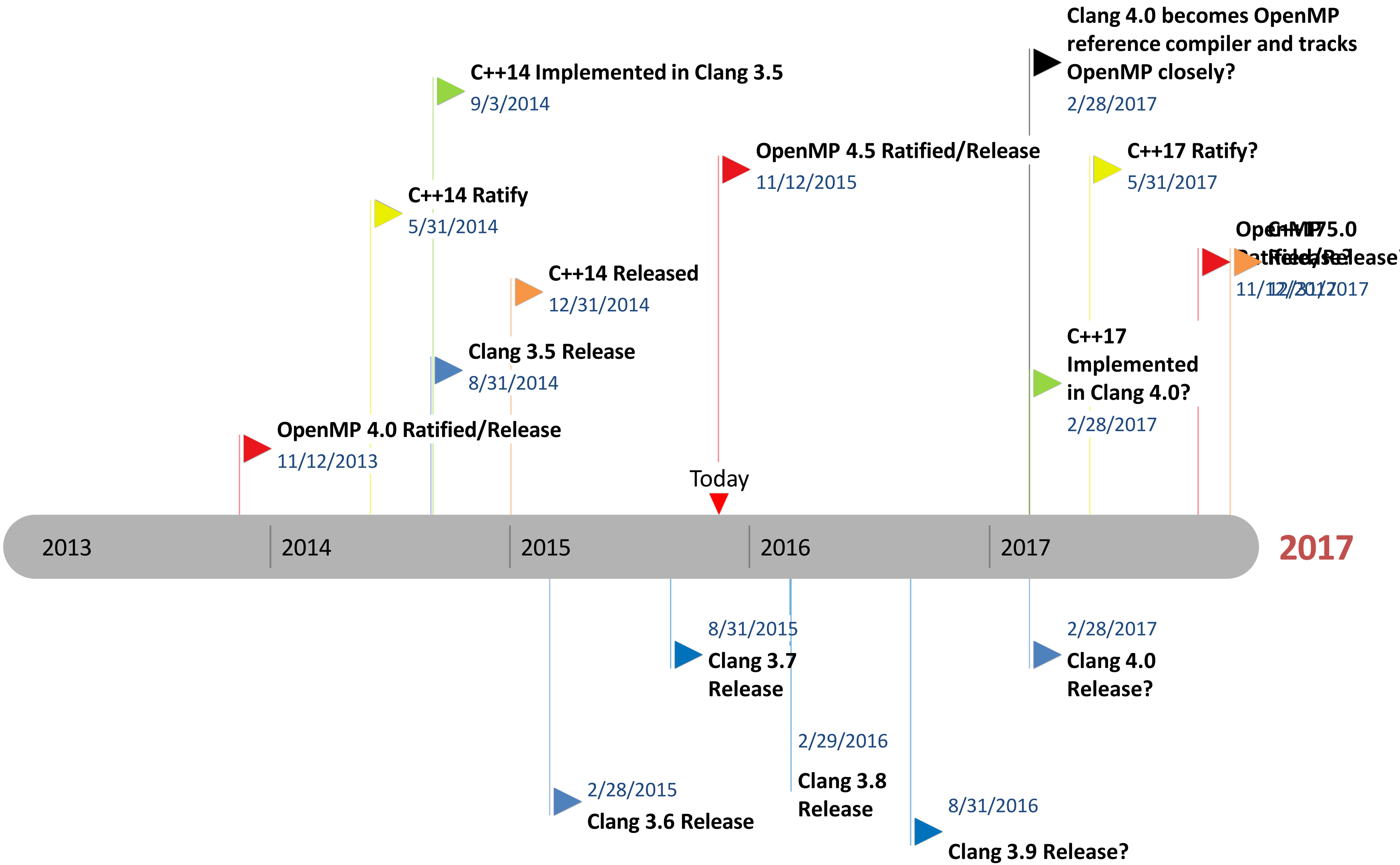
*Other names and brands may be claimed as the property of others.

What did we learn?

- Multi-Vendor/University collaboration works even outside of ISO
- Support separate vendor-dependent target RTL to enable other programming models
- Production compilers need support for L10N and I18N for multiple countries and languages

Future plans

- Clang 3.8 (~Feb, 2016): trunk switches to clang OpenMP lib, upstream OpenMP 4.0 with focus on Accelerator delivery; start code dropping for OpenMP 4.5
- Clang 3.9 (~Aug 2016): Complete OpenMP 4.0 and continue to Add OpenMP 4.5 functionality
- Clang 4.0 (~Feb 2017): clang/llvm becomes reference compiler; follow OpenMP ratification with collaborated contribution?



OpenMP 4.0 Ratified/Release
11/12/2013

C++14 Ratify
5/31/2014

Clang 3.5 Release
8/31/2014

C++14 Implemented in Clang 3.5
9/3/2014

C++14 Released
12/31/2014

2/28/2015
Clang 3.6 Release

8/31/2015
Clang 3.7 Release

Today

OpenMP 4.5 Ratified/Release
11/12/2015

2/29/2016
Clang 3.8 Release

8/31/2016
Clang 3.9 Release?

Clang 4.0 becomes OpenMP reference compiler and tracks OpenMP closely?
2/28/2017

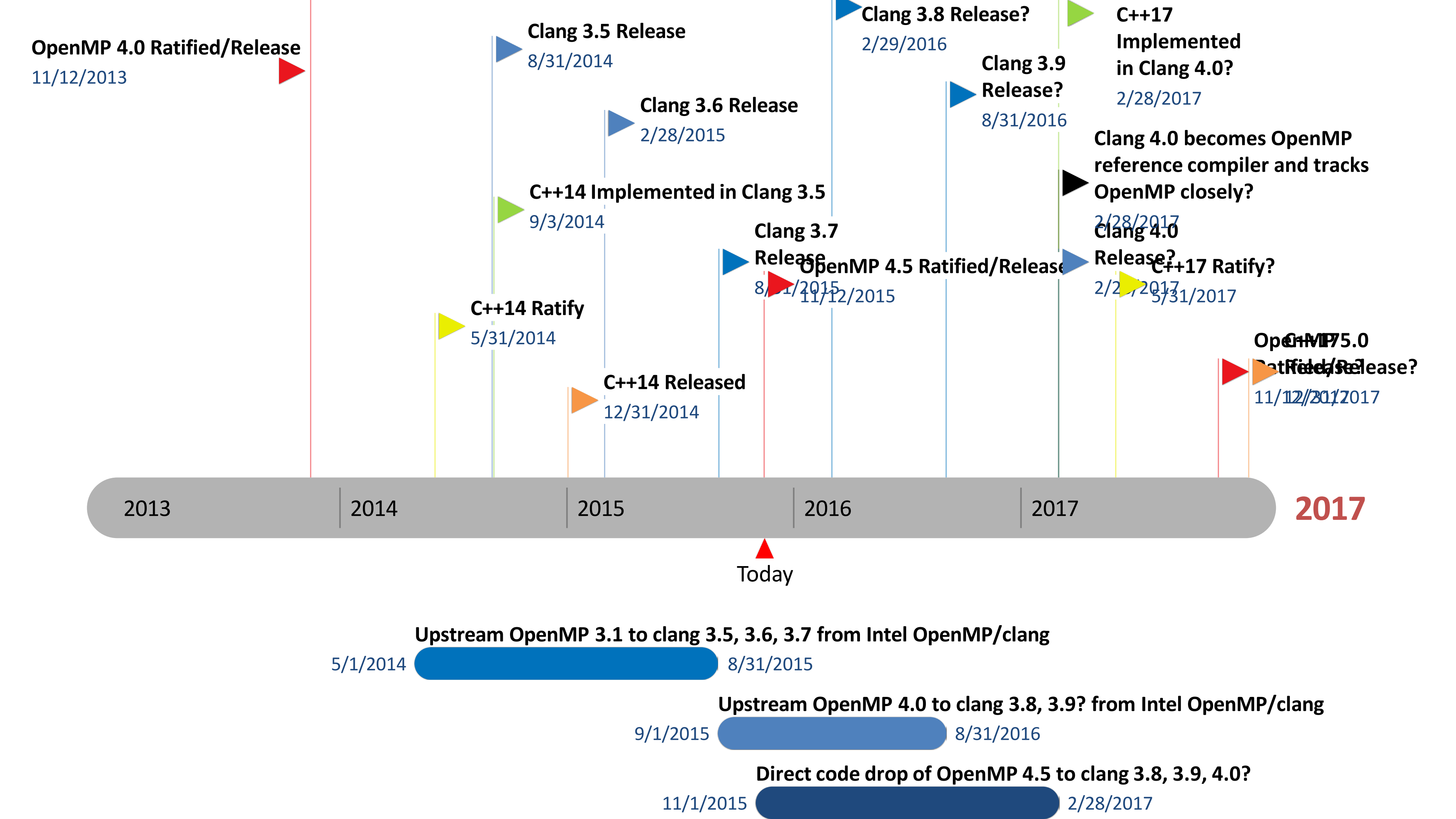
C++17 Implemented in Clang 4.0?
2/28/2017

2/28/2017
Clang 4.0 Release?

C++17 Ratify?
5/31/2017

OpenMP 5.0 Ratified/Release
11/12/2017

C++17 Release?
12/31/2017



OpenMP 4.0 Ratified/Release
11/12/2013

C++14 Ratify
5/31/2014

Clang 3.5 Release
8/31/2014

C++14 Implemented in Clang 3.5
9/3/2014

C++14 Released
12/31/2014

Clang 3.6 Release
2/28/2015

Clang 3.7 Release
8/31/2015

OpenMP 4.5 Ratified/Release
11/12/2015

Clang 3.8 Release?
2/29/2016

Clang 3.9 Release?
8/31/2016

C++17 Implemented in Clang 4.0?
2/28/2017

Clang 4.0 becomes OpenMP reference compiler and tracks OpenMP closely?
2/28/2017

Clang 4.0 Release?
2/28/2017

C++17 Ratify?
5/31/2017

OpenMP 5.0 Ratified/Release?
11/12/2017

C++17 Release?
12/31/2017

Upstream OpenMP 3.1 to clang 3.5, 3.6, 3.7 from Intel OpenMP/clang
5/1/2014 8/31/2015

Upstream OpenMP 4.0 to clang 3.8, 3.9? from Intel OpenMP/clang
9/1/2015 8/31/2016

Direct code drop of OpenMP 4.5 to clang 3.8, 3.9, 4.0?
11/1/2015 2/28/2017

Agenda

- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model

What now?

- The new C++11 Std is
  - –1353 pages compared to 817 pages in C++03
- The new C++14 Std is
  - –1373 pages (N3937), vs the free n3972
- The new C11 is
  - –701 pages compared to 550 pages in C99
- OpenMP 3.1 is
  - –160 pages and growing
- OpenMP 4.0 is
  - –320 pages
- OpenMP 4.5 is
  - –359 pages

A tale of two cities

PPL

stands for

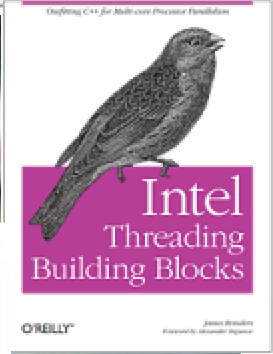
Parallel Patterns Library

ABBREVIATIONS

Abbreviations.com

OpenMP™

CILK ARTS



Outfitting C++ for Multi-core Processor Parallelism

Intel Threading Building Blocks

O'REILLY

James Reinders

Foreword by Alexander Stepanov

SYCL™

C++ AMP

Accelerated Massive Parallelism with Microsoft Visual C++

OpenCL

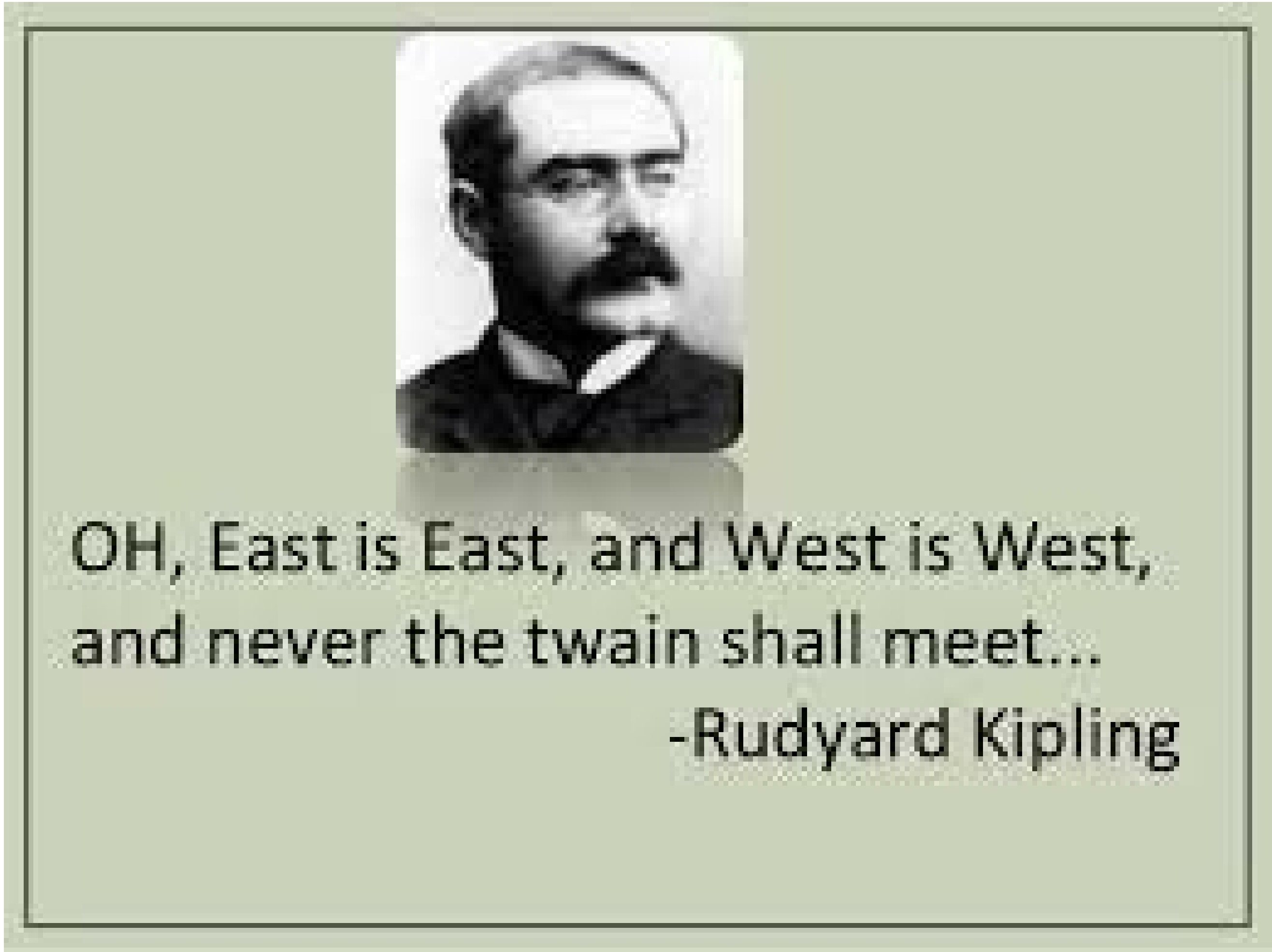
OpenACC®

DIRECTIVES FOR ACCELERATORS

NVIDIA.

CUDA.

Will the two galaxies ever join?



OH, East is East, and West is West,
and never the twain shall meet...

-Rudyard Kipling

What did we learn from the OpenMP Accelerator model?

- Consumer threads needed
- More concurrency controls needed
- Excellent HPC domain usage
- Some use in financials
- but almost none in consumers and commercial applications
- C++ support? Can it get better?

Its like the difference between:

An Aircraft Carrier Battle Group (ISO)

And a Cruiser (Consortium: OpenMP)

And a Destroyer (Company Specific language)

C++ support for Accelerators

- Memory allocation
- Templates
- Exceptions
- Polymorphism
- Current Technical Specifications
  - Concepts, Parallelism, Concurrency, TM

Programming GPU/Accelerators

- OpenGL
- DirectX
- CUDA
- OpenCL
- OpenMP
- OpenACC
- C++ AMP
- HPX
- HSA
- SYCL
- Vulkan
- A preview of C++ WG21 Accelerator model SG1/SG14 TS2 (SC15 LLVM HPC talk)

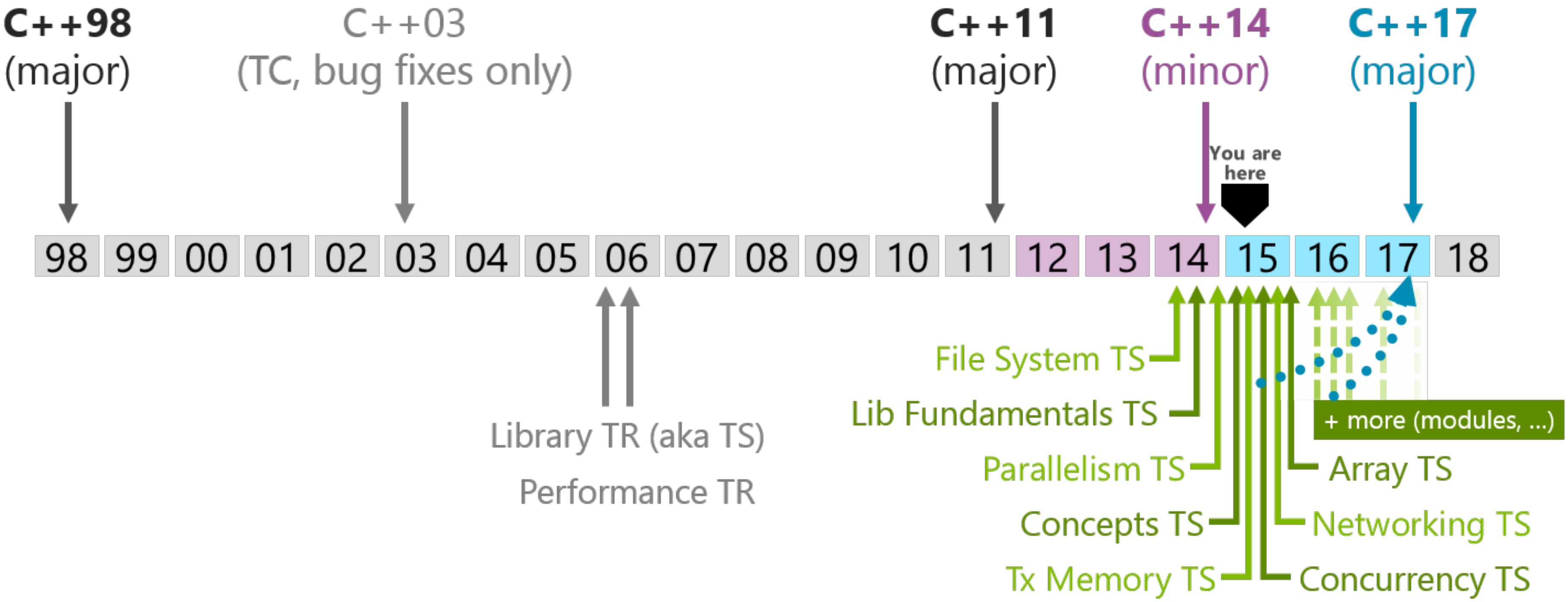
CUDA

```
texture<float, 2, cudaReadModeElementType> tex;
void foo() {
  cudaArray* cu_array;
  // Allocate array
  cudaChannelFormatDesc description = cudaCreateChannelDesc<float>();
  cudaMallocArray(&cu_array, &description, width, height);
  // Copy image data to array
  …
  // Set texture parameters (default)
  …
  // Bind the array to the texture
  …
  // Run kernel
  …
  // Unbind the array from the texture
}
```

C++AMP

```
void AddArrays(int n, int m, int * pA, int * pB, int * pSum) {
 concurrency::array_view<int,2> a(n, m, pA), b(n, m, pB),
 sum(n, m, pSum);
 concurrency::parallel_for_each(sum.extent,
 [=](concurrency::index<2> i) restrict(amp)
 {
   sum[i] = a[i] + b[i];
 });
}
```

C++11, 14, 17



C++98 (major)

C++03 (TC, bug fixes only)

C++11 (major)

C++14 (minor)

C++17 (major)

You are here

98	99	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18

Library TR (aka TS)

Performance TR

File System TS

Lib Fundamentals TS

Parallelism TS

Concepts TS

Tx Memory TS

Array TS

Networking TS

Concurrency TS

+ more (modules, ...)

C++1Y(1Y=17 or 22) Concurrency Plan

Parallelism

Parallel STL Algorithms:

Data-Based Parallelism. (Vector, SIMD, ...)

Task-based parallelism (cilk, OpenMP, fork-join)

MapReduce

Pipelines

Concurrency

Future Extensions (then, wait_any, wait_all):

Executors:

Resumable Functions, await (with futures)

Counters

Queues

Concurrent Vector

Unordered Associative Containers

Latches and Barriers

upgrade_lock

Atomic smart pointers

Status after Oct Kona C++ Meeting

Project	**What's in it?**	**Status**
Filesystem TS	Standard filesystem interface	Published!
Library Fundamentals TS I	optional, any, string_view and more	Published!
Library Fundamentals TS II	source code information capture and various utilities	Voted out for balloting by national standards bodies
Concepts ("Lite") TS	Constrained templates	Publication imminent
Parallelism TS I	Parallel versions of STL algorithms	Published!
Parallelism TS II	TBD. Exploring task blocks, progress guarantees, SIMD	Under active development
Transactional Memory TS	Transactional Memory TS	Published!

Project	**What’s in it?**	**Status**
Concurrency TS I	improvements to future, latches and barriers, atomic smart pointers	Voted out for publication!
Concurrency TS II	TBD. Exploring executors, synchronic types, atomic views, concurrent data structures	Under active development
Networking TS	Sockets library based on Boost.ASIO	Design review completed; wording review of the spec in progress
Ranges TS	Range-based algorithms and views	Design review completed; wording review of the spec in progress
Numerics TS	Various numerical facilities	Beginning to take shape
Array Extensions TS	Stack arrays whose size is not known at compile time	Direction given at last meeting; waiting for proposals
Reflection	Code introspection and (later) reification mechanisms	Still in the design stage, no ETA

Project	What's in it?	Status
Graphics	2D drawing API	Waiting on proposal author to produce updated standard wording
Modules	A component system to supersede the textual header file inclusion model	Microsoft and Clang continuing to iterate on their implementations and converge on a design. The feature will target a TS, not C++17.
Coroutines	Resumable functions	At least two competing designs. One of them may make C++17.
Contracts	Preconditions, postconditions, etc.	In early design stage

Agenda

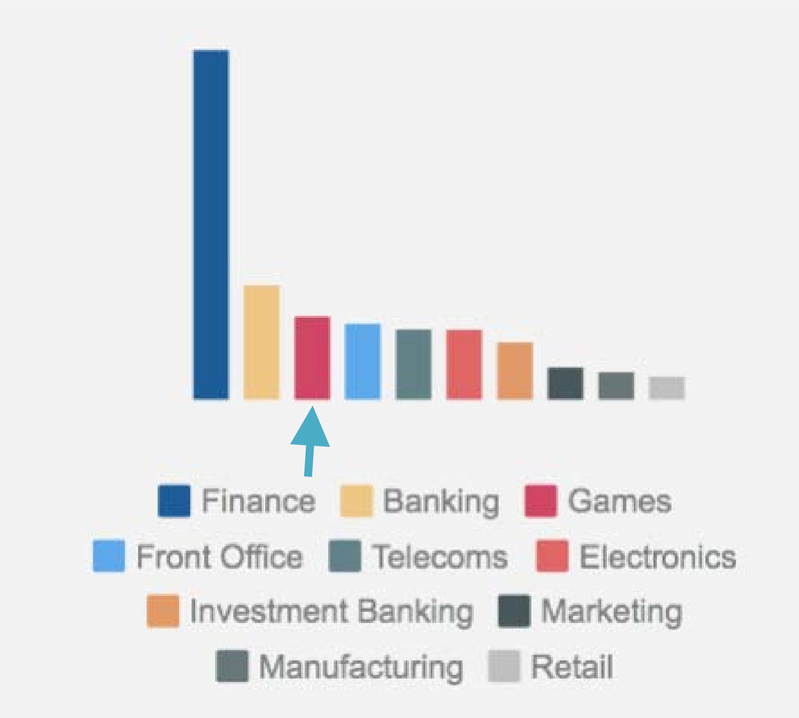
- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model

The Birth of Study Group 14

Towards Improving C++ for Games & Low Latency

WORLD OF WARSHIPS
WARGAMING.NET

Among the top users of C++!



http://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/

About SG14

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future

The Breaking Wave: N4456

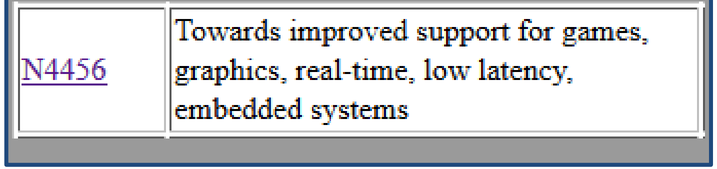
CppCon 2014

C++ committee panel leads to impromptu game developer meeting.

Google Group created.

Discussions have outstanding industry participation.

N4456 authored and published!



N4456	Towards improved support for games, graphics, real-time, low latency, embedded systems

Formation of SG14

N4456 presented at Spring 2015 Standards Committee Meeting in Lenexa.

Very well received!

SG14
Game Dev &
Low Latency

Formation of Study Group 14:
Game Dev & Low Latency

Chair: Michael Wong (IBM)

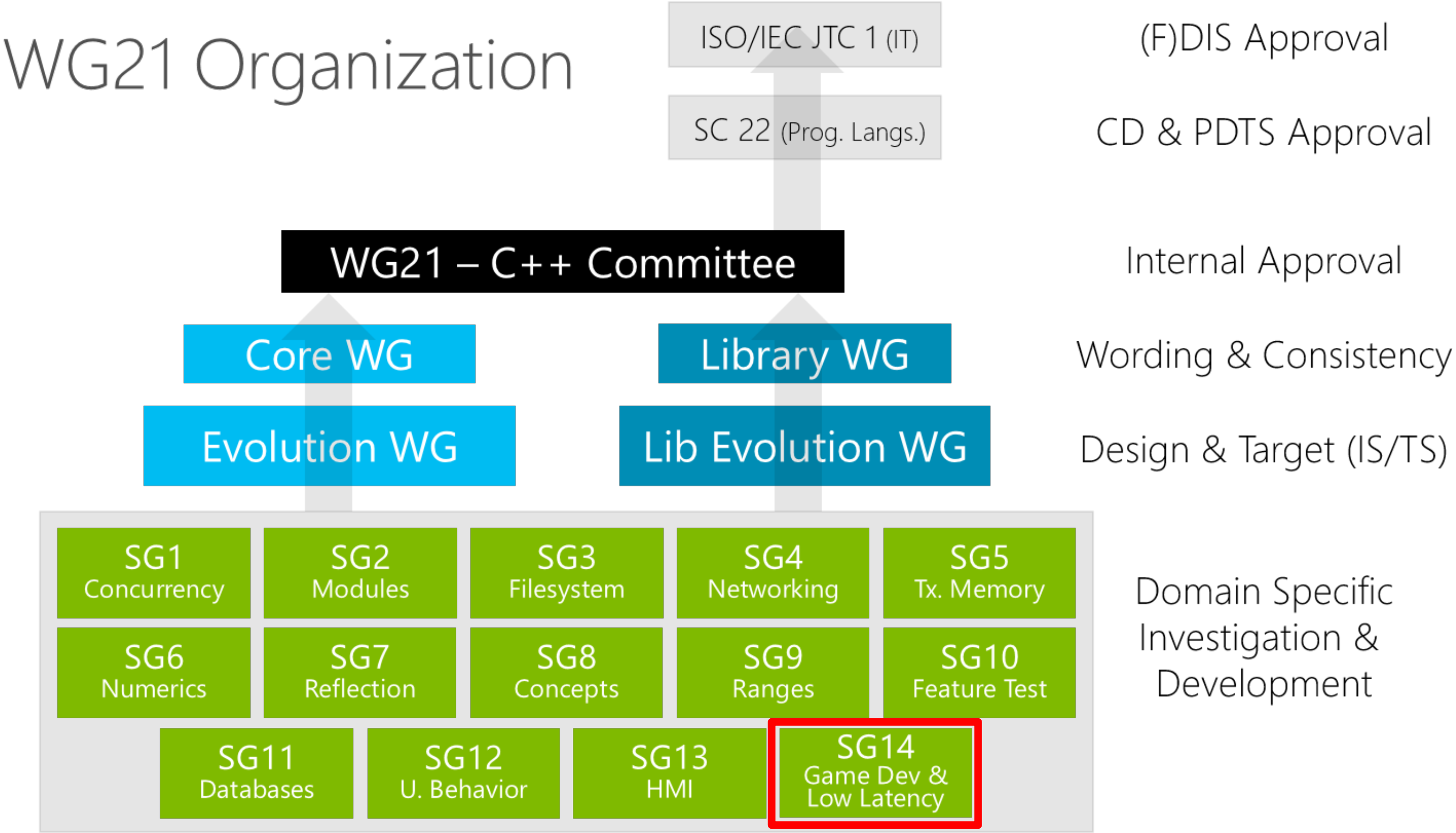
cppcon
the c++ conference

GDC

Two SG14 meetings planned:

- CppCon 2015 (this Wednesday)
- GDC 2016, hosted by SONY

WG21 Organization



ISO/IEC JTC 1 (IT)

(F)DIS Approval

SC 22 (Prog. Langs.)

CD & PDTS Approval

WG21 – C++ Committee

Internal Approval

Core WG

Library WG

Wording & Consistency

Evolution WG

Lib Evolution WG

Design & Target (IS/TS)

SG1 Concurrency

SG2 Modules

SG3 Filesystem

SG4 Networking

SG5 Tx. Memory

SG6 Numerics

SG7 Reflection

SG8 Concepts

SG9 Ranges

SG10 Feature Test

SG11 Databases

SG12 U. Behavior

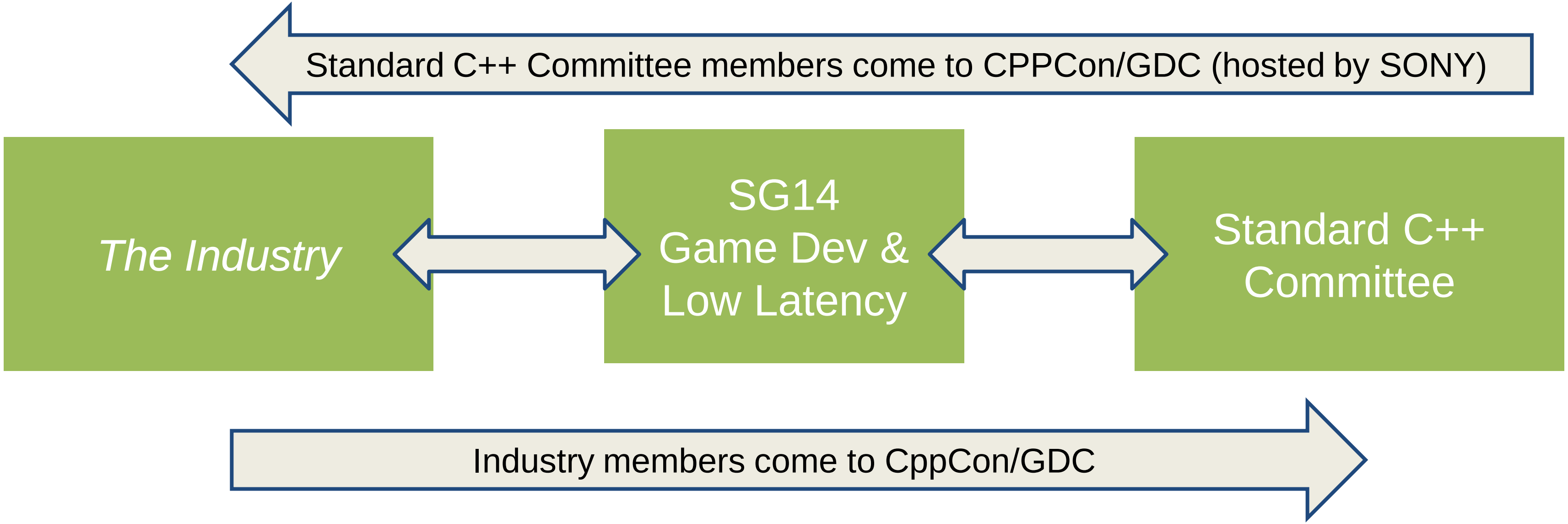
SG13 HMI

SG14 Game Dev & Low Latency

Domain Specific Investigation & Development

https://isocpp.org/std/the-committee

Improving Communication/Feedback/review cycle


Standard C++ Committee members come to CPPCon/GDC (hosted by SONY)

The Industry

SG14
Game Dev &
Low Latency

Standard C++
Committee

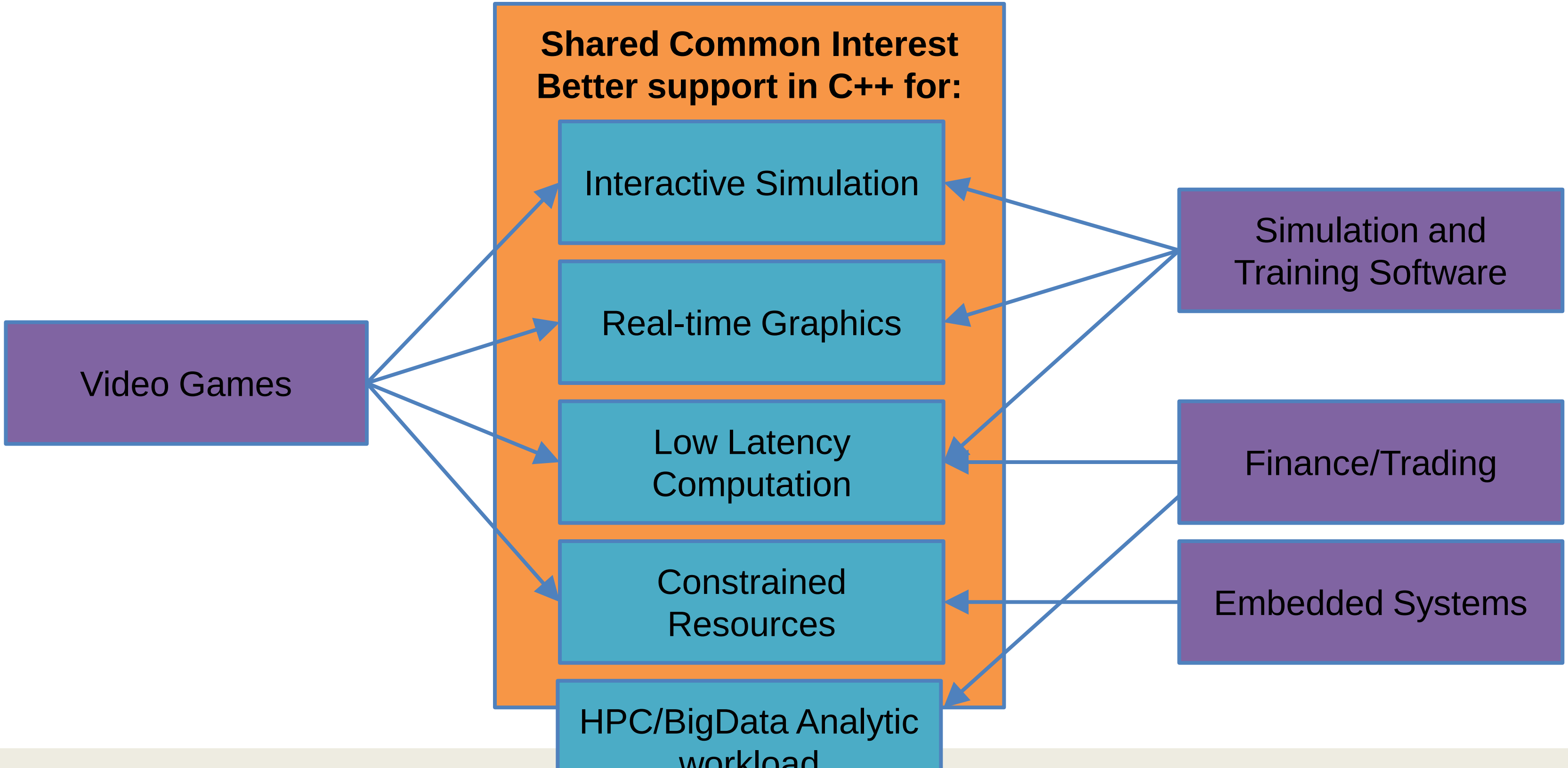
Industry members come to CppCon/GDC

Meetings are opportunities to present papers or discuss existing proposals
SG14 approved papers are presented by C++ Committee members at Standard meetings for voting
Feedback goes back through SG14 to industry for revision
Rinse and repeat

The Industry name linkage brings in lots of people

- The First Industry-named SG that gains connection with
  - Games
  - Financial/Trading
  - Banking
  - Simulation
  - +HPC/Big Data Analysis?

Audience of SG14 Goals and Scopes: Not just games!



Video Games

Shared Common Interest
Better support in C++ for:

Interactive Simulation

Real-time Graphics

Low Latency Computation

Constrained Resources

HPC/BigData Analytic workload

Simulation and Training Software

Finance/Trading

Embedded Systems

Where We Are

- Google Groups
  - https://groups.google.com/a/isocpp.org/forum/?fromgroups#!forum/sg14
- GitHub
  - https://github.com/WG21-SG14/SG14
  - Created by Guy Davidson

SG14 are interested in following these proposals

- GPU/Acccelerator support
- Executors
  - 3 ways: low-latency, parallel loops, server task dispatch
- Atomic views
- Coroutines
- `noexcept` library additions
  - Use `std::error_code` for signaling errors
- Early SIMD in C++ investigation
  - There are existing SIMD papers suggesting eg. “Vec<T,N>” and “for simd (;;)”

- Array View
- Node-based Allocators
- String conversions
- hot set
- vector and matrix
- Exception and RTTI costs
- Ring or circular buffers
- Flat_map
- Intrusive containers
- Allocator interface
- Radix sort
- Spatial and geometric algorithms
- Imprecise but faster alternatives for math algorithms
- Cache-friendly hash table
- Contiguous containers
- Stack containers
- Fixed-point numbers
- `plf::colony` and `plf::stack`

Agenda

- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model

C++ Standard GPU/Acelerators

- Attended by both National Labs and commercial/consumers
- Glimpse into the future
- No design as yet, but several competing design candidates
- Offers the best chance of a model that works across both domains for C++ (only)

Grand Unification?

“Hello World” with std::thread

```cpp
#include <thread>
#include <iostream>

void func()
{
  std::cout << "**Inside thread "
   << std::this_thread::get_id() << "!" << std::endl;
}

int main()
{
  std::thread  t;
  t = std::thread( func );

  t.join();
  return 0;
}
```

A simple function for thread to do…

Create and schedule thread…

Wait for thread to finish…

Avoiding errors / program termination…

```
#include <thread>
#include <iostream>

void func()
{
  std::cout << "**Hello world...\n";
}

int main()
{
  std::thread  t;
  t = std::thread( func );

  t.join();
  return 0;
}
```

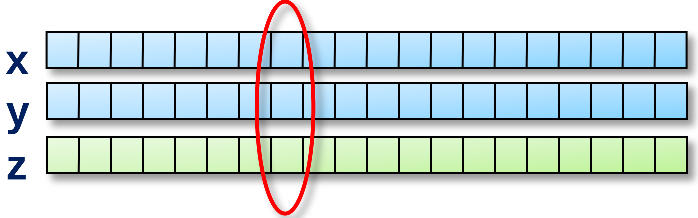
(1) Thread function must do ***exception handling****; unhandled exceptions ==> termination…*

```
void func()
{
  try
  {
     // computation:
  }
  catch(...)
  {
     // do something:
  }
}
```

(2) Must ***join****, otherwise termination…*

NOTE*: avoid use of detach( ) in C++11, difficult to use safely.*

Example: saxpy



- Saxpy == *Scalar Alpha X Plus Y*
  - *Scalar multiplication and vector addition*

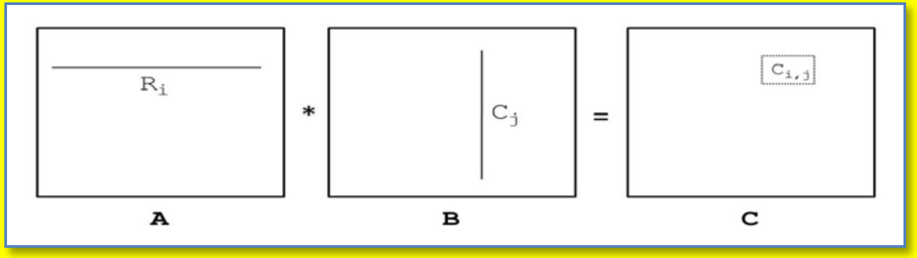
```
for (int i=0; i<n; i++)
   z[i] = a * x[i] + y[i];
```

Parallel

```
int start = …;
int end   = …;
for (int t=0; t<NumThreads; t++)
{
   thread(
     [&z,x,y,a,start,end]() -> void
     {
       for (int i = start; i < end; i++)
         z[i] = a * x[i] + y[i];
     }
   );

   start += …;
   end   += …;
}
```

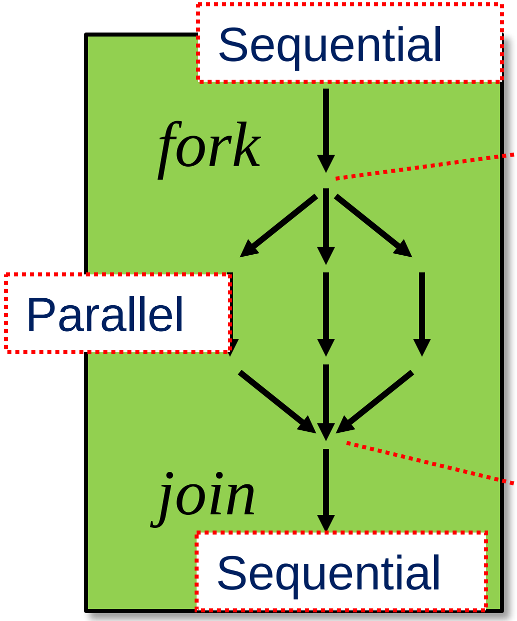
Sequential Matrix Multiplication

```
//
// Naïve, triply-nested sequential solution:
//
for (int i = 0; i < N; i++)
{
   for (int j = 0; j < N; j++)
   {
      C[i][j] = 0.0;

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



Structured ("fork-join") parallelism

- A common pattern when creating multiple threads



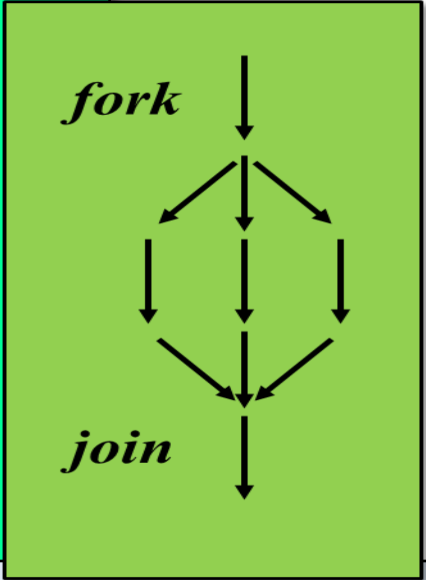
```
#include <vector>

std::vector<std::thread>  threads;

int cores = std::thread::hardware_concurrency();

for (int i=0; i<cores; ++i)  // 1 per core:
{
  auto code = []() { DoSomeWork(); };
  threads.push_back( thread(code) );
}
```

```
for (std::thread& t : threads) // new range-based for:
  t.join();
```

Parallel solution

```
// 1 thread per core:
numthreads = thread::hardware_concurrency();
```

```
int rows  = N / numthreads;
int extra = N % numthreads;
int start = 0;      // each thread does [start..end)
int end   = rows;
vector<thread>  workers;
for (int t = 1; t <= numthreads; t++)
{
  if (t == numthreads)  // last thread does extra rows:
    end += extra;
  workers.push_back( thread([start, end, N, &C, &A, &B]()
  {
    for (int i = start; i < end; i++)
      for (int j = 0; j < N; j++)
      {
        C[i][j] = 0.0;
        for (int k = 0; k < N; k++)
          C[i][j] += (A[i][k] * B[k][j]);
      }
  }));

  start = end;
  end   = start + rows;
}
```

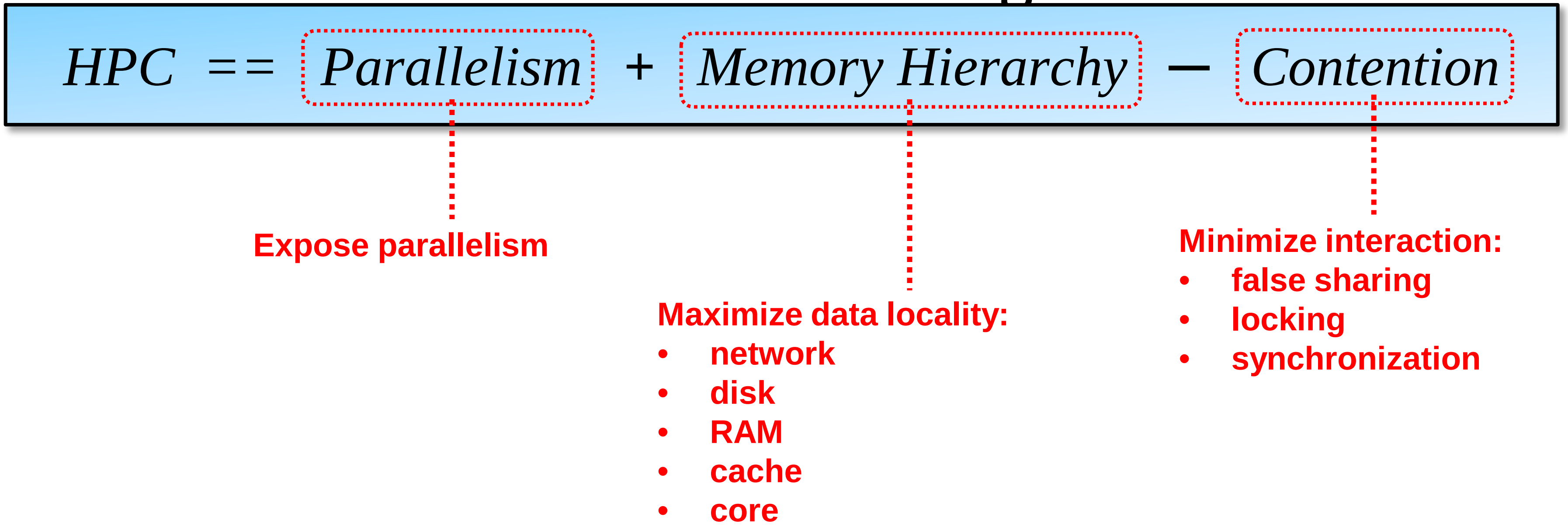


```
for (thread& t : workers)
   t.join();
```

Going Parallel with C++11 by Joe Hummel

What does C++ Standard parallelism still need?

- Parallelism alone is not enough for HPC…



HPC == Parallelism + Memory Hierarchy – Contention

Expose parallelism

Maximize data locality:

- **network**
- **disk**
- **RAM**
- **cache**
- **core**

Minimize interaction:

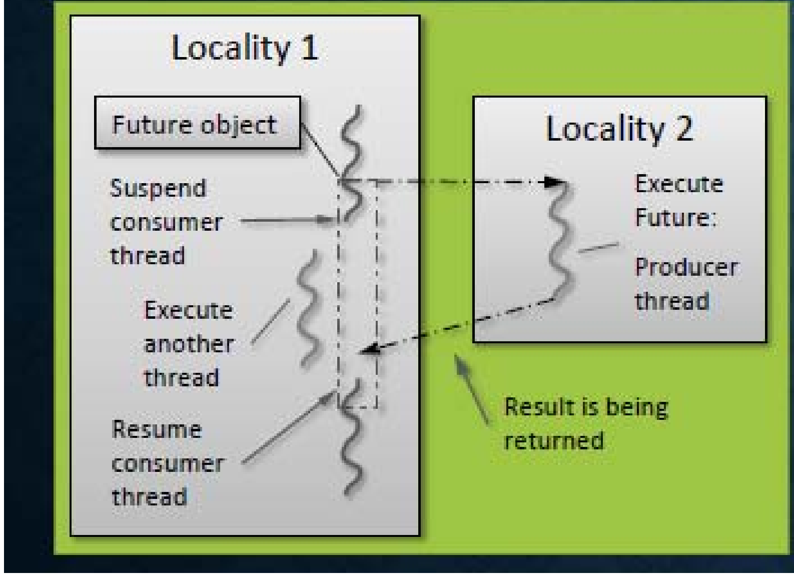
- **false sharing**
- **locking**
- **synchronization**

Asynchronous Calls

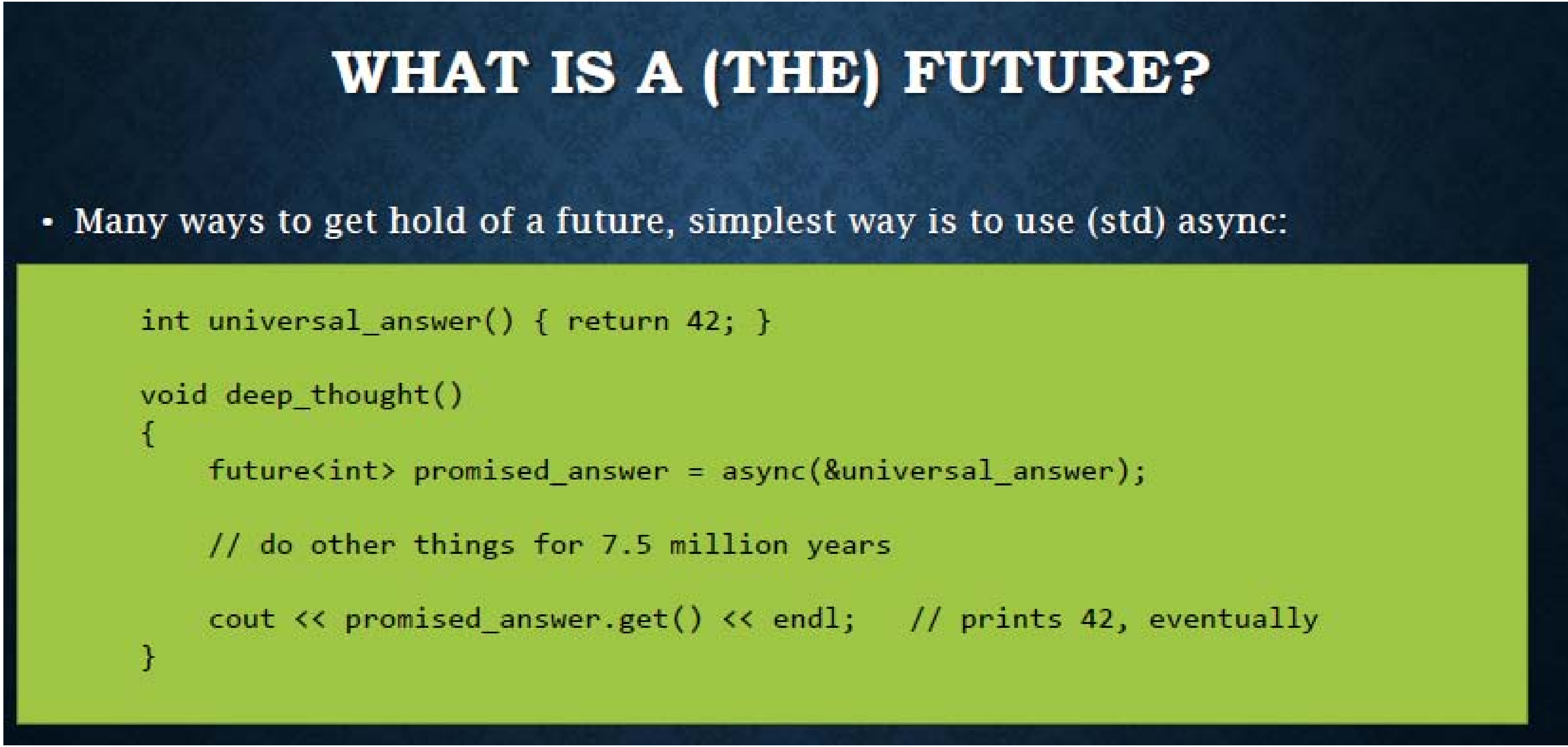
- Building blocks:
  - std::async: Request asynchronous execution of a function.
  - Future: token representing function’s result.
- Unlike raw use of std::thread objects:
  - Allows values or exceptions to be returned.
    - Just like “normal” function calls.

WHAT IS A (THE) FUTURE

- A future is an object representing a result which has not been calculated yet



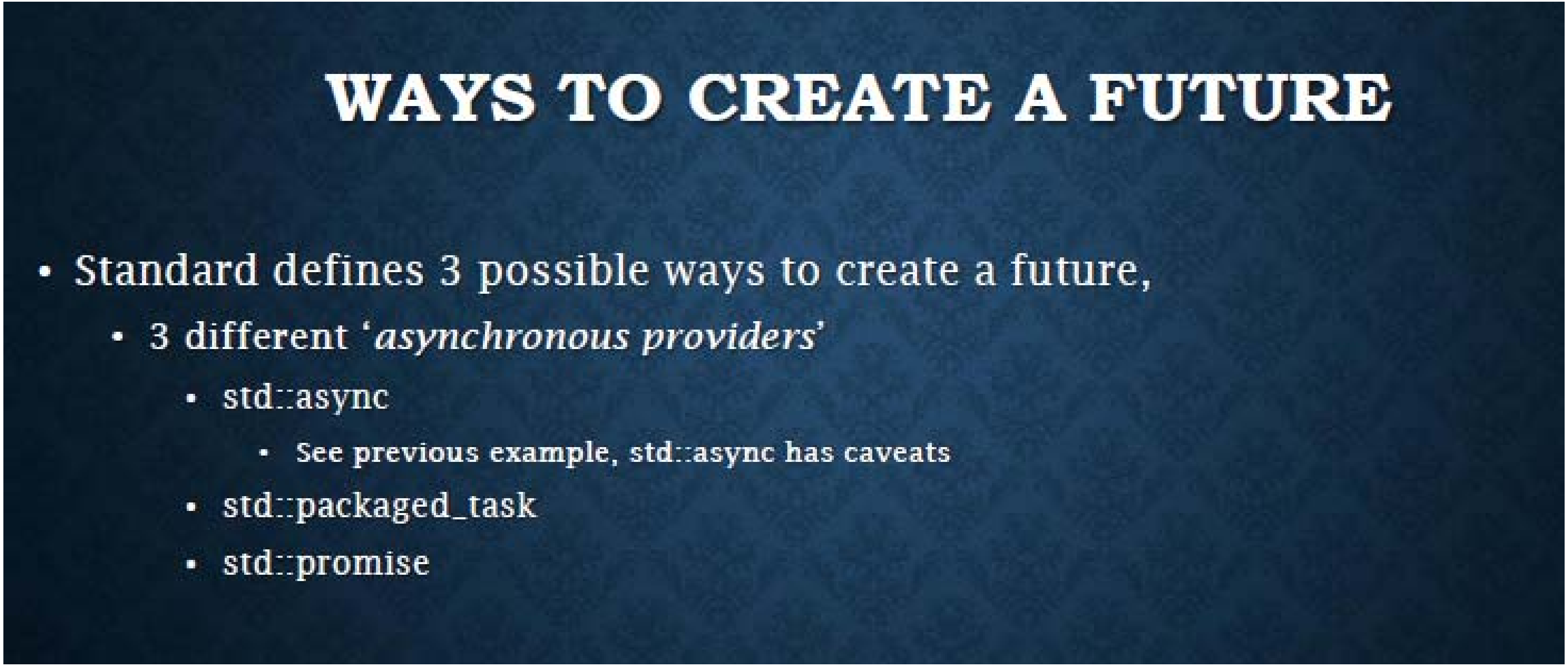
- Enables transparent synchronization with producer
- Hides notion of dealing with threads
- Makes asynchrony manageable
- Allows for composition of several asynchronous operations
- (Turns concurrency into parallelism)



WHAT IS A (THE) FUTURE?

- Many ways to get hold of a future, simplest way is to use (std) async:

```
int universal_answer() { return 42; }

void deep_thought()
{
    future<int> promised_answer = async(&universal_answer);

    // do other things for 7.5 million years

    cout << promised_answer.get() << endl;   // prints 42, eventually
}
```



WAYS TO CREATE A FUTURE

- Standard defines 3 possible ways to create a future,
  - 3 different '*asynchronous providers*'
    - std::async
      - See previous example, std::async has caveats
    - std::packaged_task
    - std::promise

Standard Concurrency Interfaces

- std::async<>and std::future<>: concurrency as with sequential processing
  - one location calls a concurrent task and dealing with the outcome is as simple as with local sub-functions
- std: :thread: lOW-level approach
  - one location calls a concurrent lask and has to provide low-level techniques to handle the outcome
- std::promise<> and std::future<>: Simplify processing the outcome
  - one location calls a concurrent task but dealing with the outcome is simplified
- packaged_task<> : helper to separate task definition from call
  - one location defines a task and provides a handle for the outcome
  - another location decides when to call the task and the arguments
  - the call must not necessarily happen in another thread

std::async + std::future

- Use async to start asynchronous operation
- Use returned future to wait upon result / exception

```
#include <future>

std::future<int> f = std::async( []() -> int
  {
    int result = PerformLongRunningOperation()
    return result;
  }
);
.
.
```

START

lambda return type…

```
try
{
  int x = f.get();   // wait if necessary, harvest result:
  cout << x << endl;
}
catch(exception &e)
{
  cout << "**Exception: " << e.what() << endl;
}
```

WAIT

Async operations

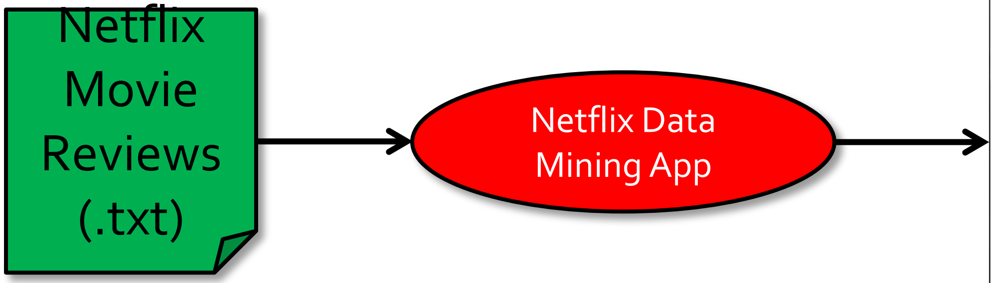
- Run on current thread *or* a new thread
- By default, system decides…

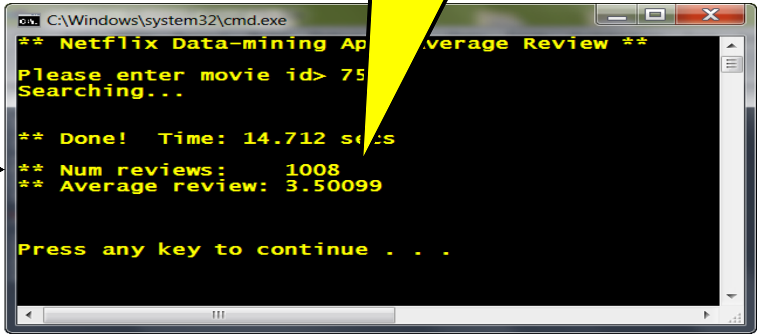
```
// runs on current thread when you “get” value (i.e. lazy execution):
future<T> f1 = std::async( std::launch::deferred, []() -> T {...} );


// runs now on a new, dedicated thread:
future<T> f2 = std::async( std::launch::async, []() -> T {...} );


// let system decide (e.g. maybe you created enough work to keep system busy?):
future<T> f3 = std::async(↑[]() -> T {...} );
```

optional argument missing

Commercial application

- Netflix data-mining…





```
** Netflix Data-mining Ap      verage Review **

Please enter movie id> 75
Searching...

** Done!   Time: 14.712 se  s

** Num reviews:     1008
** Average review: 3.50099


Press any key to continue . . .
```

Average rating…

Sequential solution

```
cin >> movieID;

vector<string> ratings = readFile("ratings.txt");

tuple<int,int> results = dataMine(ratings, movieID);

int numRatings = std::get<0>(results);
int sumRatings = std::get<1>(results);
double avgRating = double(numRatings) / double(sumRatings);

cout << numRatings << endl;
cout << avgRating << endl;
```

```
dataMine(vector<string> &ratings, int id)
{
  foreach rating
    if ids match num++, sum += rating;

  return tuple<int,int>(num, sum);
}
```

Parallel solution

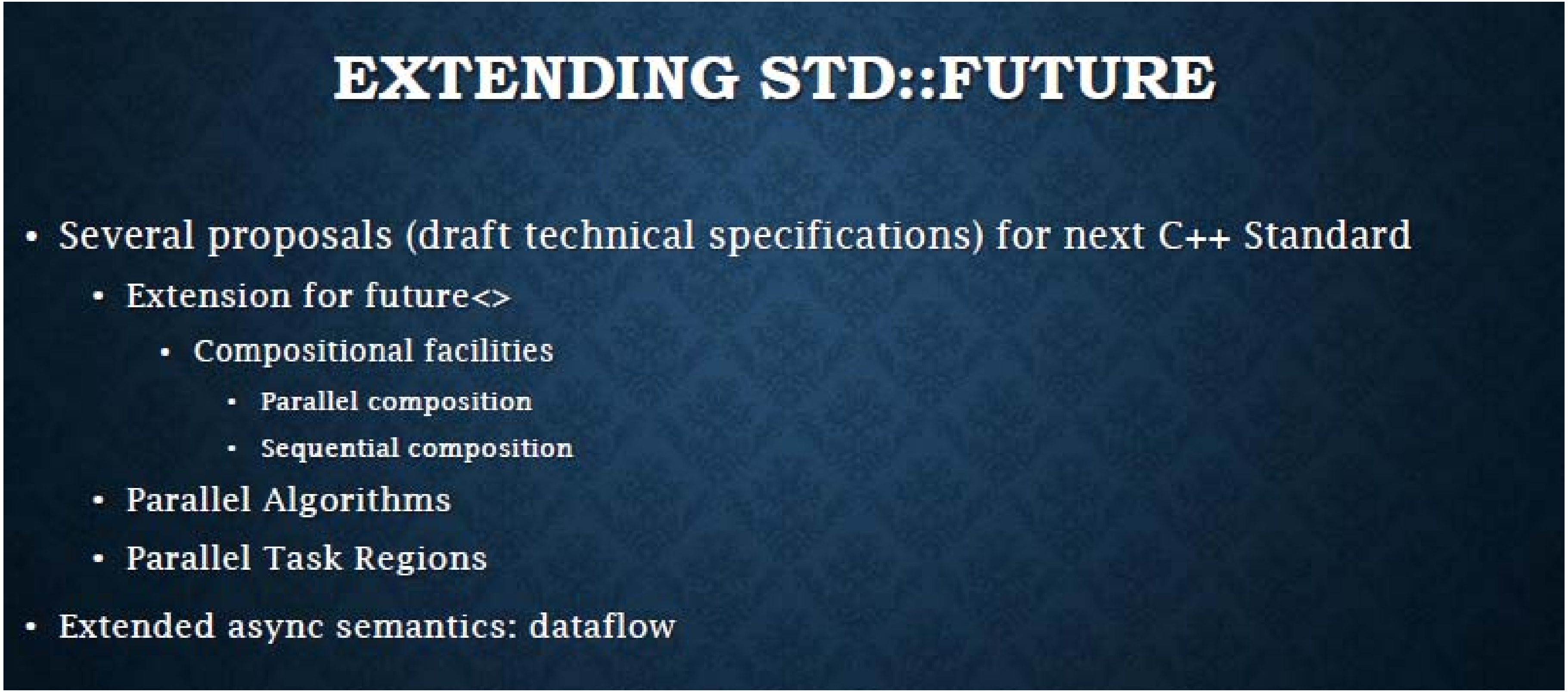
```
int chunksize = ratings.size() / numthreads;
int leftover  = ratings.size() % numthreads;
int begin     = 0;        // each thread does [start..end)
int end       = chunksize;

vector<future<tuple<int,int>>>  futures;

for (int t = 1; t <= numthreads; t++)
{
  if (t == numthreads)   // last thread does extra rows:
     end += leftover;

  futures.push_back(
    async([&ratings, movieID, begin, end]() -> tuple<int,int>
    {
        return dataMine(ratings, movieID, begin, end);
    })
  );

  begin = end;
  end   = begin + chunksize;
}
```

```
dataMine(..., int begin, int end)
{
  foreach rating in begin..end
    if ids match num++, sum += rating;

  return tuple<int,int>(num, sum);
}
```

```
for (future<tuple<int,int>> &f: futures)
{
   tuple<int, int> t = f.get();
   numRatings += std::get<0>(t);
   sumRatings += std::get<1>(t);
}
```

Other things C++ need: Types of parallelism

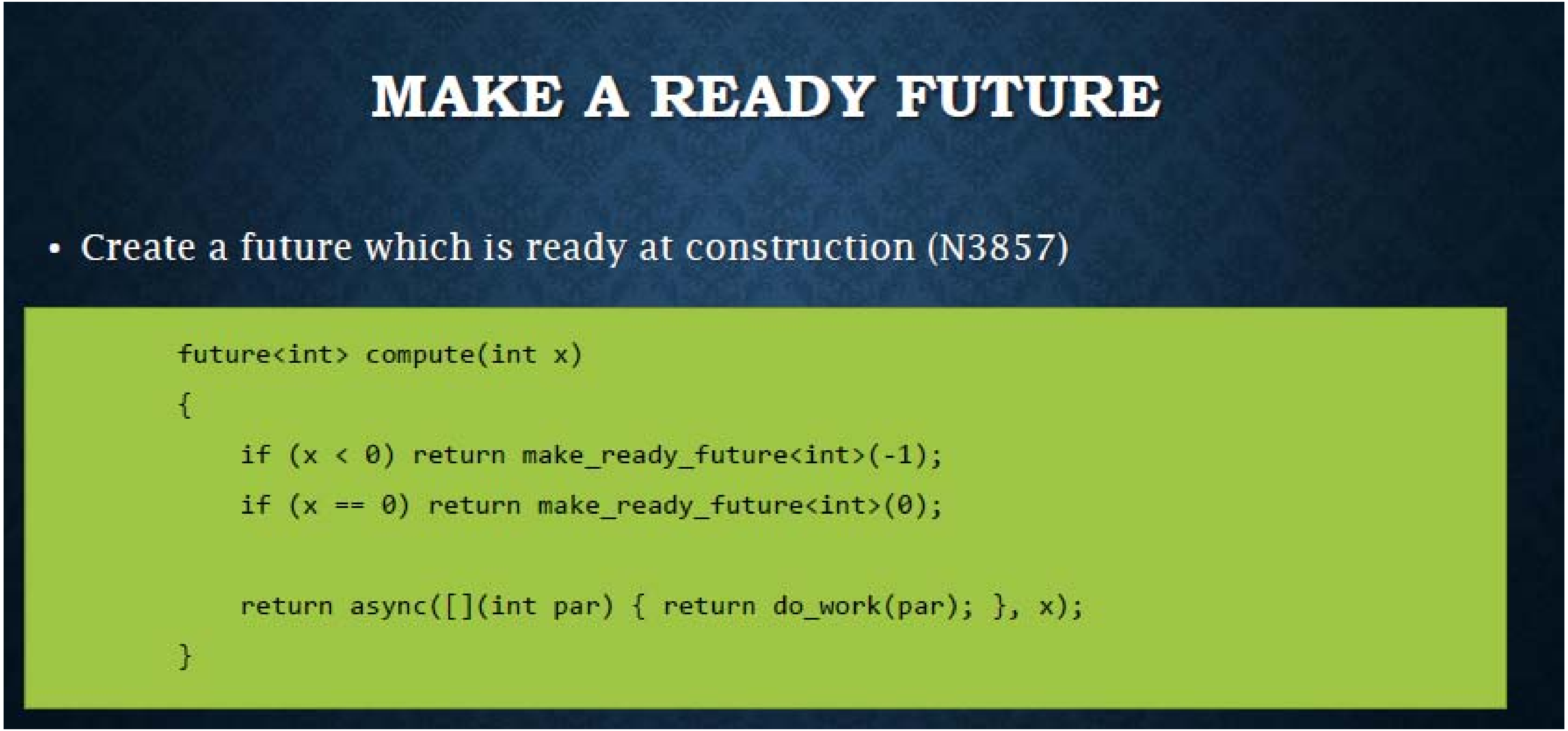
- Most common types:
  - Data: coming in SIMD proposal
  - Task: coming in executors and task blocks
  - Embarrassingly parallel: async and threads
  - Dataflow: Concurrency TS (.then)



EXTENDING STD::FUTURE

- Several proposals (draft technical specifications) for next C++ Standard
  - Extension for future<>
    - Compositional facilities
      - Parallel composition
      - Sequential composition
  - Parallel Algorithms
  - Parallel Task Regions
- Extended async semantics: dataflow

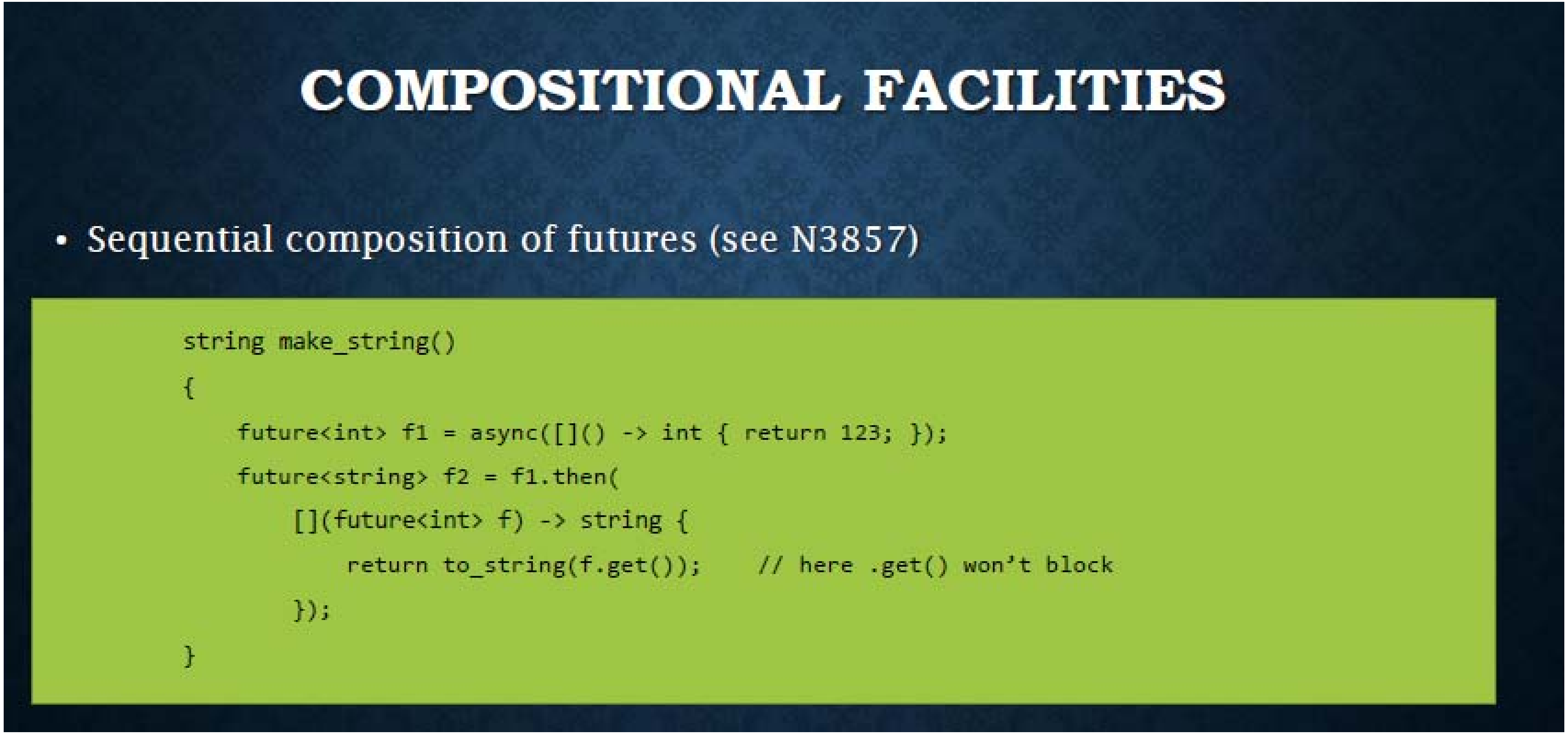
Asynchronous Computing in C++ by Hartmut Kaiser



MAKE A READY FUTURE

- Create a future which is ready at construction (N3857)

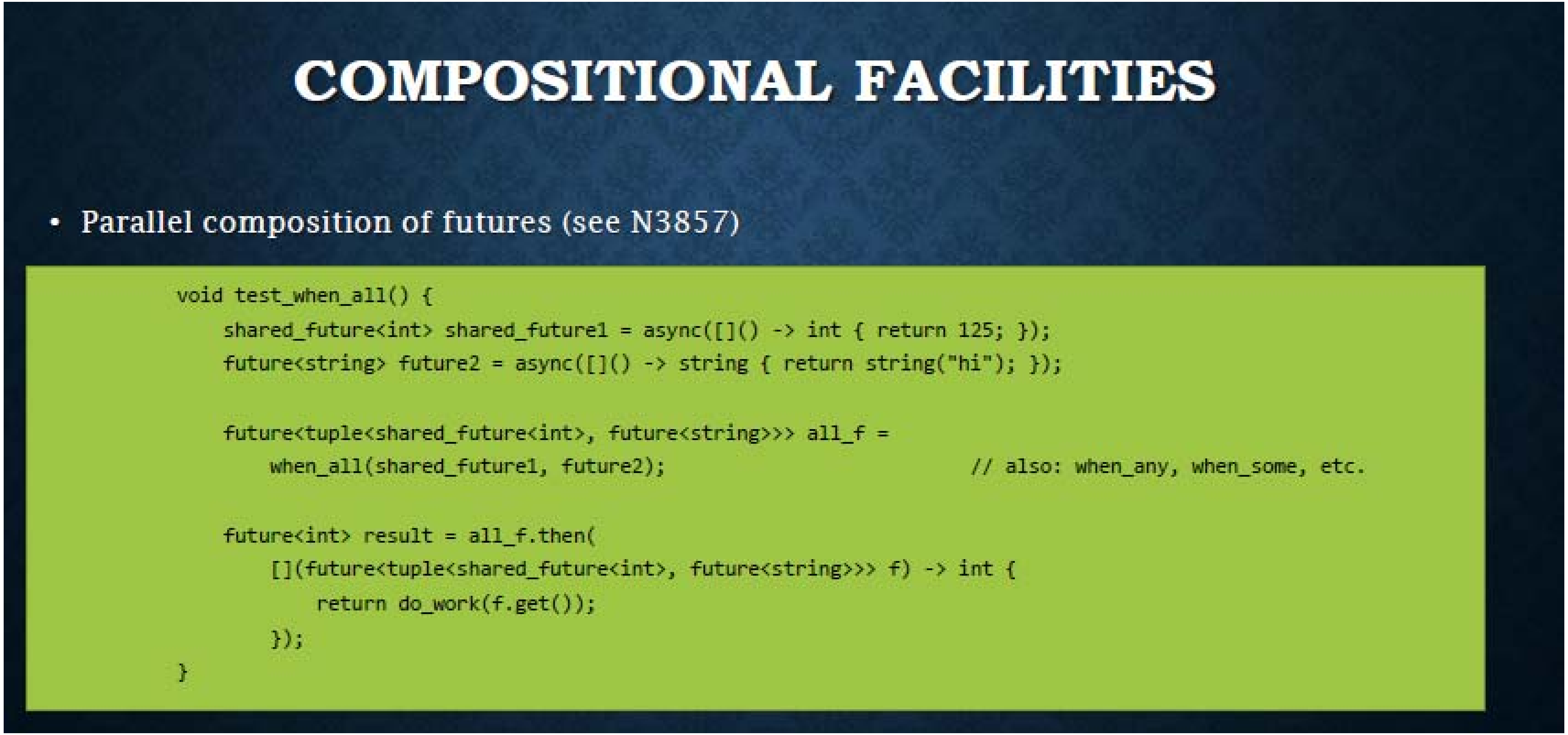
```
future<int> compute(int x)
{
    if (x < 0) return make_ready_future<int>(-1);
    if (x == 0) return make_ready_future<int>(0);

    return async([](int par) { return do_work(par); }, x);
}
```



COMPOSITIONAL FACILITIES

- Sequential composition of futures (see N3857)

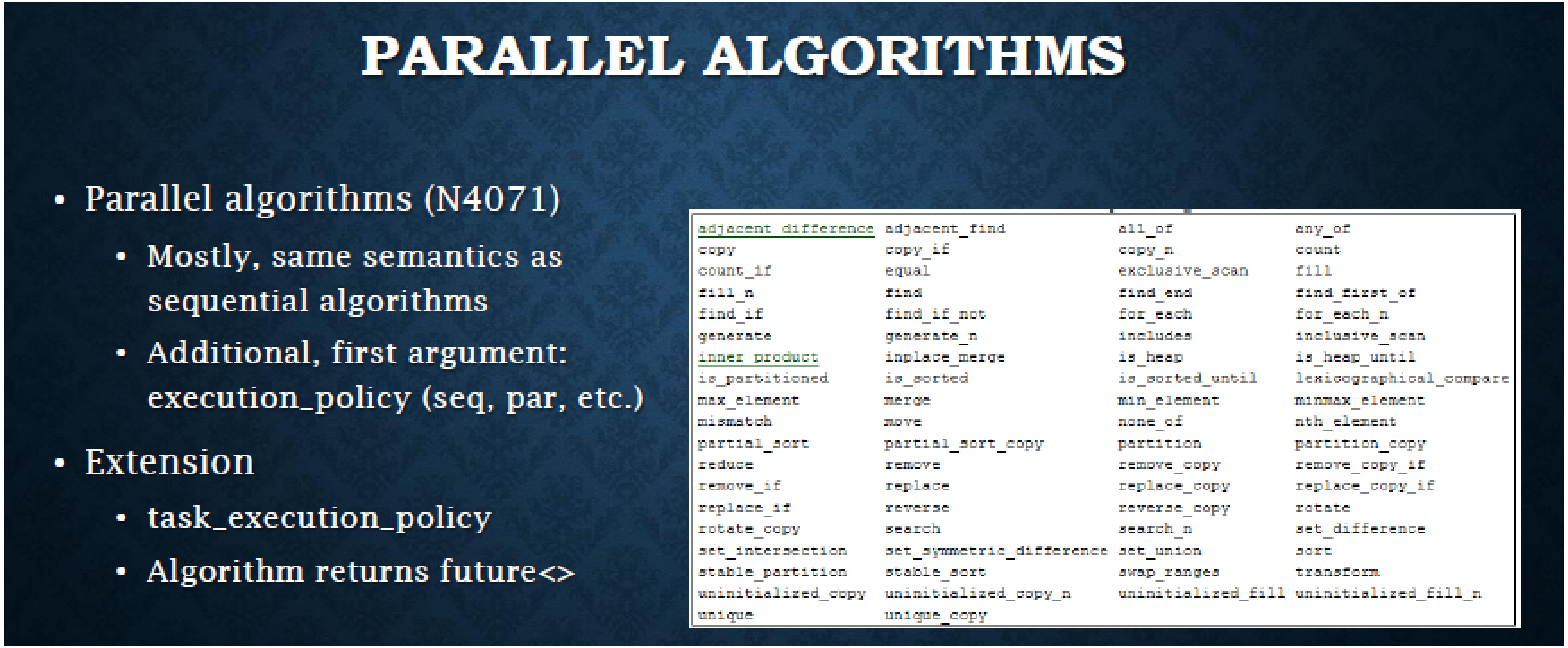
```
string make_string()
{
    future<int> f1 = async([]() -> int { return 123; });
    future<string> f2 = f1.then(
        [](future<int> f) -> string {
            return to_string(f.get());    // here .get() won’t block
        });
}
```



COMPOSITIONAL FACILITIES

- Parallel composition of futures (see N3857)

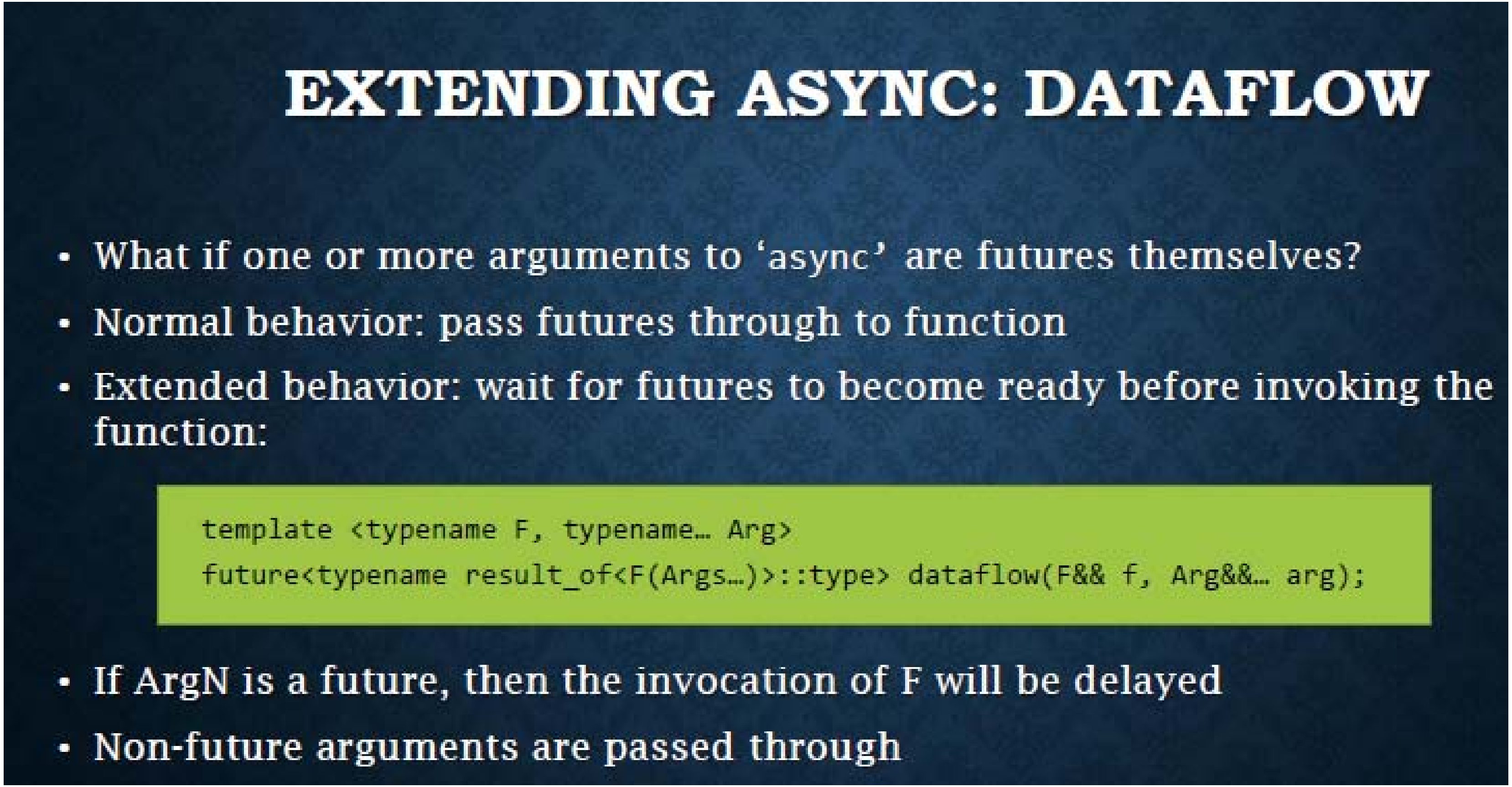
```
void test_when_all() {
    shared_future<int> shared_future1 = async([]() -> int { return 125; });
    future<string> future2 = async([]() -> string { return string("hi"); });

    future<tuple<shared_future<int>, future<string>>> all_f =
        when_all(shared_future1, future2);                          // also: when_any, when_some, etc.

    future<int> result = all_f.then(
        [](future<tuple<shared_future<int>, future<string>>> f) -> int {
            return do_work(f.get());
        });
}
```



PARALLEL ALGORITHMS

- Parallel algorithms (N4071)
  - Mostly, same semantics as sequential algorithms
  - Additional, first argument: execution_policy (seq, par, etc.)
- Extension
  - task_execution_policy
  - Algorithm returns future<>

adjacent_difference	adjacent_find	all_of	any_of
copy	copy_if	copy_n	count
count_if	equal	exclusive_scan	fill
fill_n	find	find_end	find_first_of
find_if	find_if_not	for_each	for_each_n
generate	generate_n	includes	inclusive_scan
inner_product	inplace_merge	is_heap	is_heap_until
is_partitioned	is_sorted	is_sorted_until	lexicographical_compare
max_element	merge	min_element	minmax_element
mismatch	move	none_of	nth_element
partial_sort	partial_sort_copy	partition	partition_copy
reduce	remove	remove_copy	remove_copy_if
remove_if	replace	replace_copy	replace_copy_if
replace_if	reverse	reverse_copy	rotate
rotate_copy	search	search_n	set_difference
set_intersection	set_symmetric_difference	set_union	sort
stable_partition	stable_sort	swap_ranges	transform
uninitialized_copy	uninitialized_copy_n	uninitialized_fill	uninitialized_fill_n
unique	unique_copy		



EXTENDING ASYNC: DATAFLOW

- What if one or more arguments to ‘async’ are futures themselves?
- Normal behavior: pass futures through to function
- Extended behavior: wait for futures to become ready before invoking the function:

```
template <typename F, typename… Arg>
future<typename result_of<F(Args…)>::type> dataflow(F&& f, Arg&&… arg);
```

- If ArgN is a future, then the invocation of F will be delayed
- Non-future arguments are passed through

Asynchronous Computing in C++ by Hartmut Kaiser

C++ Std+ proposals already have many features for accelerators

- Asynchronous tasks (C++11 futures plus C++17 *then, when*, is_ready,…)*
- Parallel Algorithms
- Executors
- Multi-dim arrays, Layouts

Candidates to C++ Std Accelerator Model

- C++AMP
  - Restrict keyword is a mistake
  - GPU Hardware removing traditional hurdles
  - Modern GPU instruction sets can handle nearly full C++
  - Memory systems evolving towards single heap

Better candidates

- Goal: Use standard C++ to express all intra-node parallelism
  - Agency extends Parallelism TS
  - HCC
  - SYCL extends Parallelism TS

Food for thought and Q/A

- C11/C++14 Standards
  - –C++ : http://www.open-std.org/jtc1/sc22/wg21/prot/14882fdis/n3937.pdf
  - –C++ (post C++14 free version): http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4296.pdf
  - –C: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf
- Participate and feedback to Compiler
  - –What features/libraries interest you or your customers?
  - –What problem/annoyance you would like the Std to resolve?
  - –Is Special Math important to you?
  - –Do you expect 0x features to be used quickly by your customers?
- Talk to me at my blog:
  - –http://www.ibm.com/software/rational/cafe/blogs/cpp-standard

My blogs and email address

- ISOCPP.org Director, VP http://isocpp.org/wiki/faq/wg21#michael-wong
  OpenMP CEO: http://openmp.org/wp/about-openmp/
  My Blogs: http://ibm.co/pCvPHR
  C++11 status: http://tinyurl.com/43y8xgf
  Boost test results
  http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911
  C/C++ Compilers Feature Request Page
  http://www.ibm.com/developerworks/rfe/?PROD_ID=700
  Chair of WG21 SG5 Transactional Memory:
  https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!forum/tm
  Chair of WG21 SG14 Games Dev/Low Latency:
  https://groups.google.com/a/isocpp.org/forum/?fromgroups#!forum/sg14