The OpenMP Common Core

A journey back to the roots of OpenMP

Tim Mattson

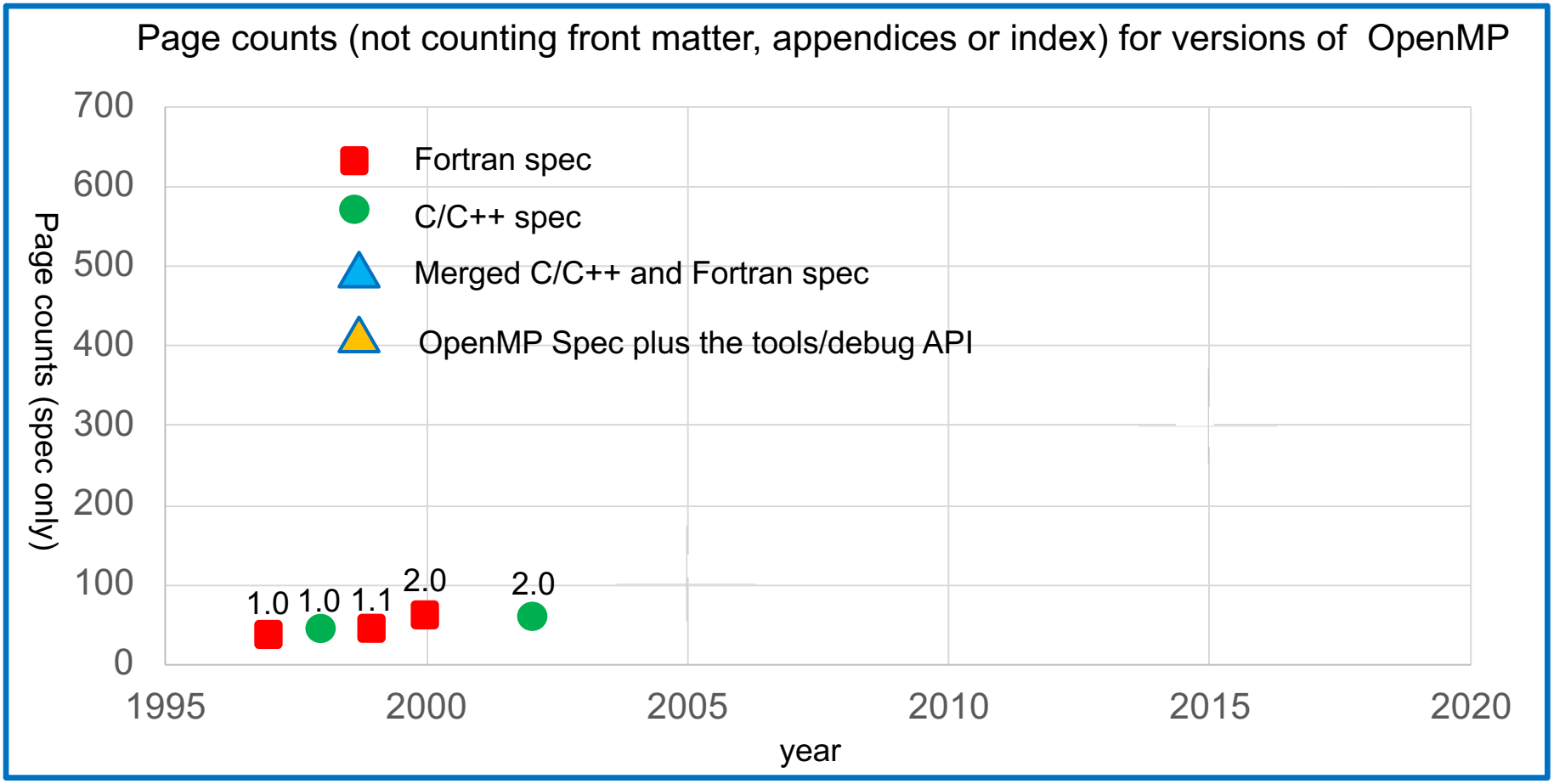
Intel Corp.

timothy.g.mattson@ intel.com

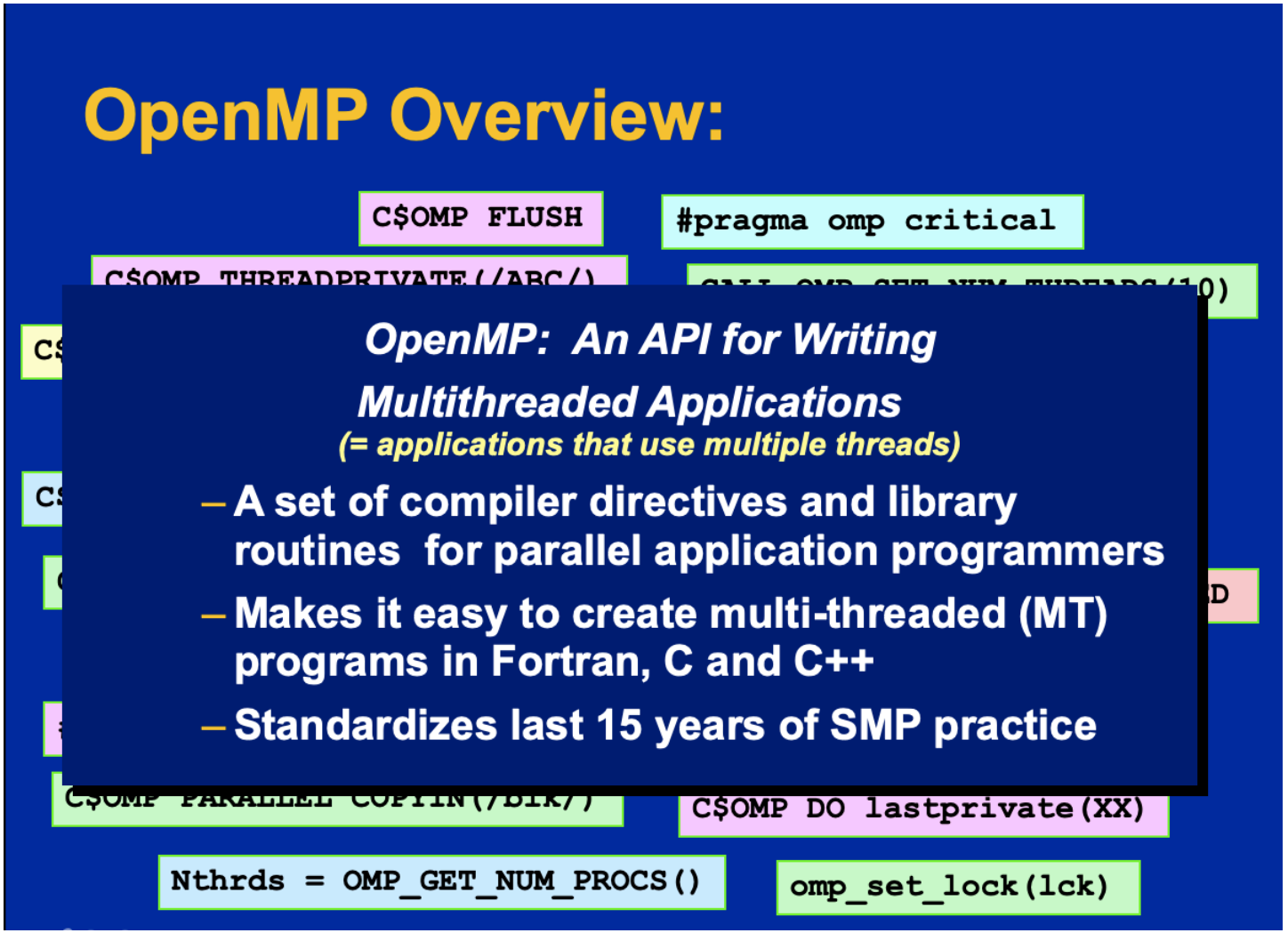
Tutorial materials are available here: https://github.com/uob-hpc/openmp-tutorial

The Growing OpenMP Specification

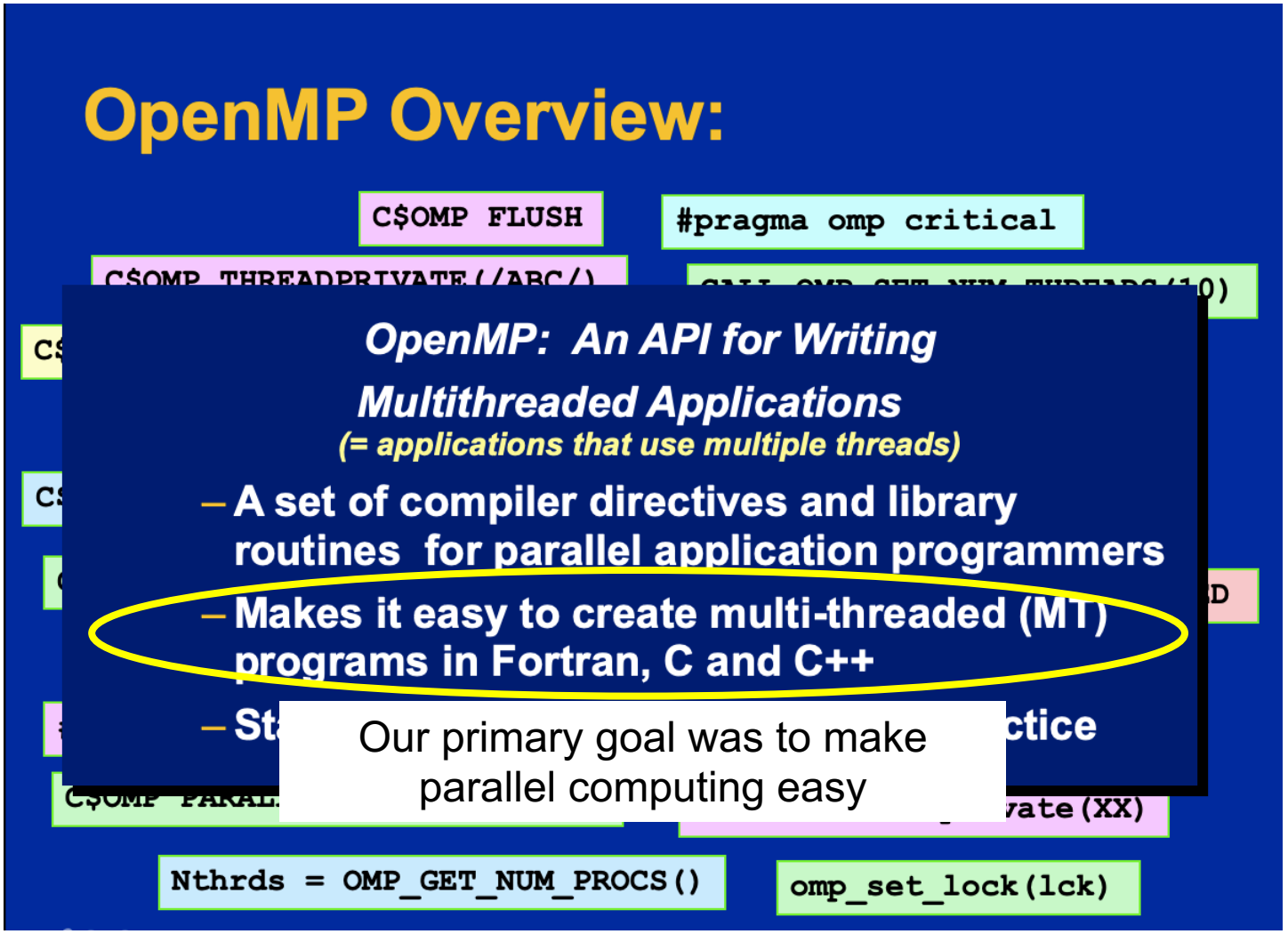
- OpenMP started out in 1997 as a simple interface for the application programmers more versed in their area of science than computer science



The roots of OpenMP:
A slide from my SC2003 OpenMP tutorial

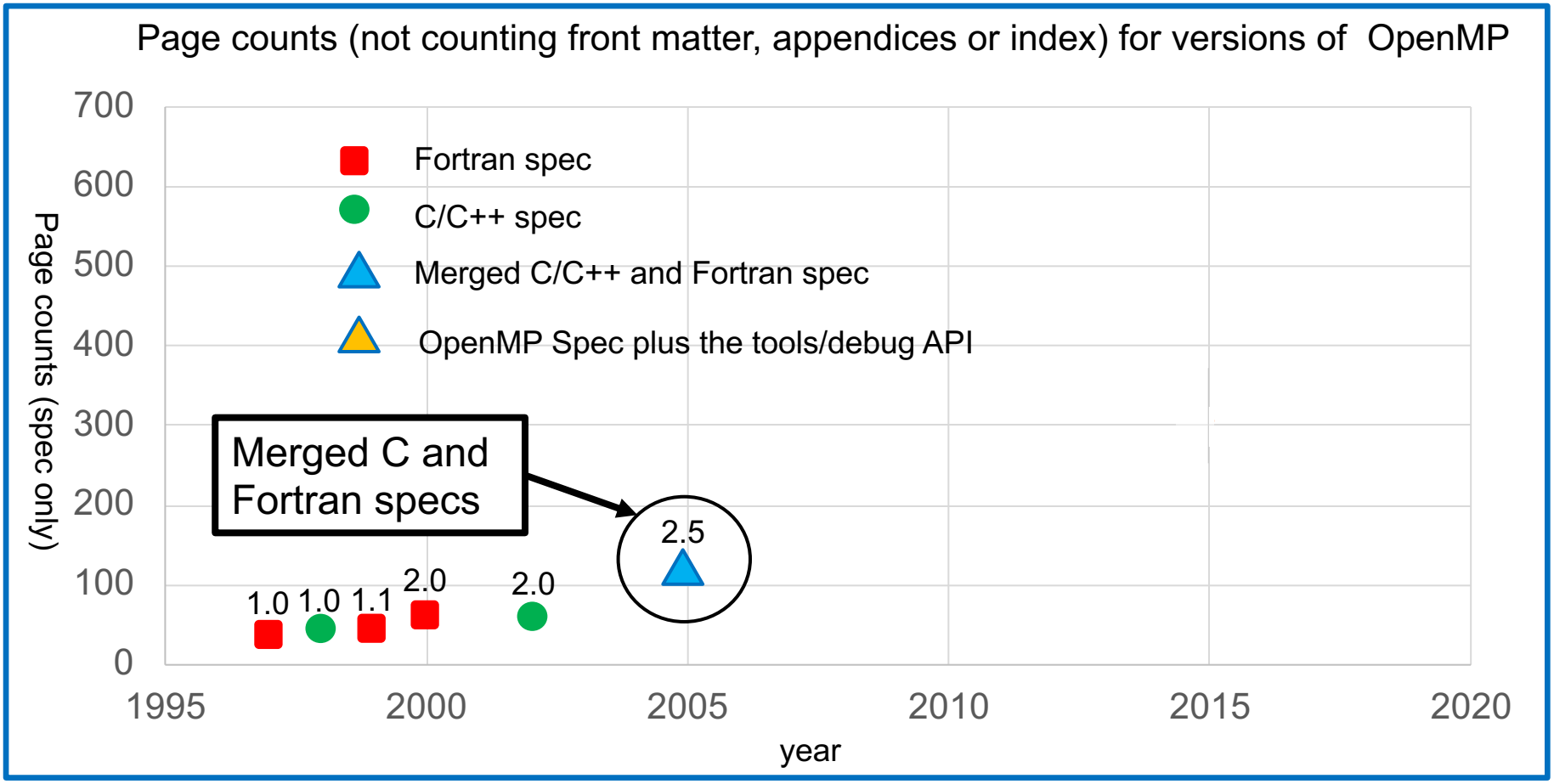


The roots of OpenMP:
A slide from my SC2003 OpenMP tutorial



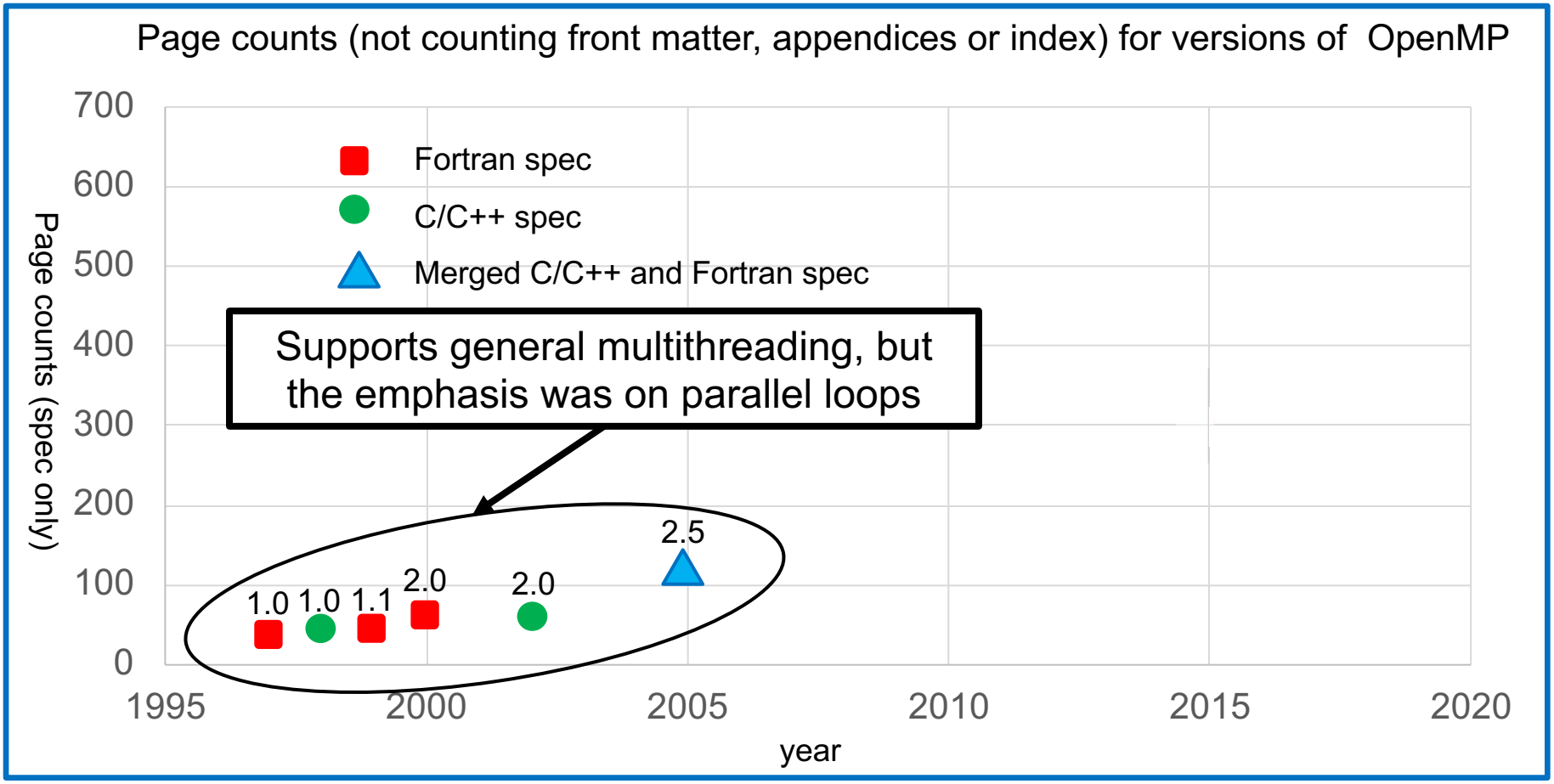
The Growing OpenMP Specification

- Rewrote the specification to merge languages … quality of the specification improved dramatically! It became a spec we could be really proud of.

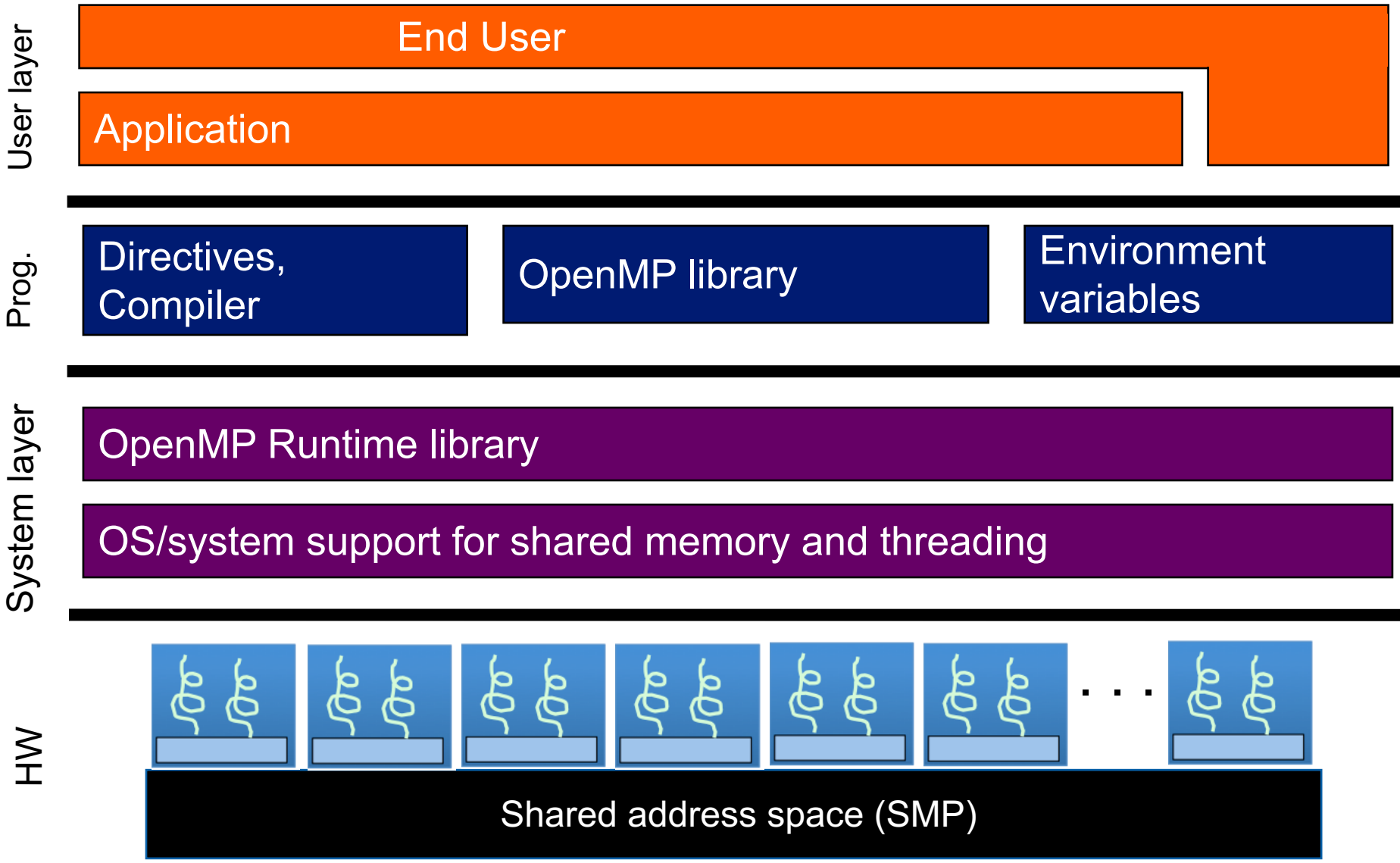


The Growing OpenMP Specification

- SPMD patterns worked well but for most programmers, it was all about parallel loops

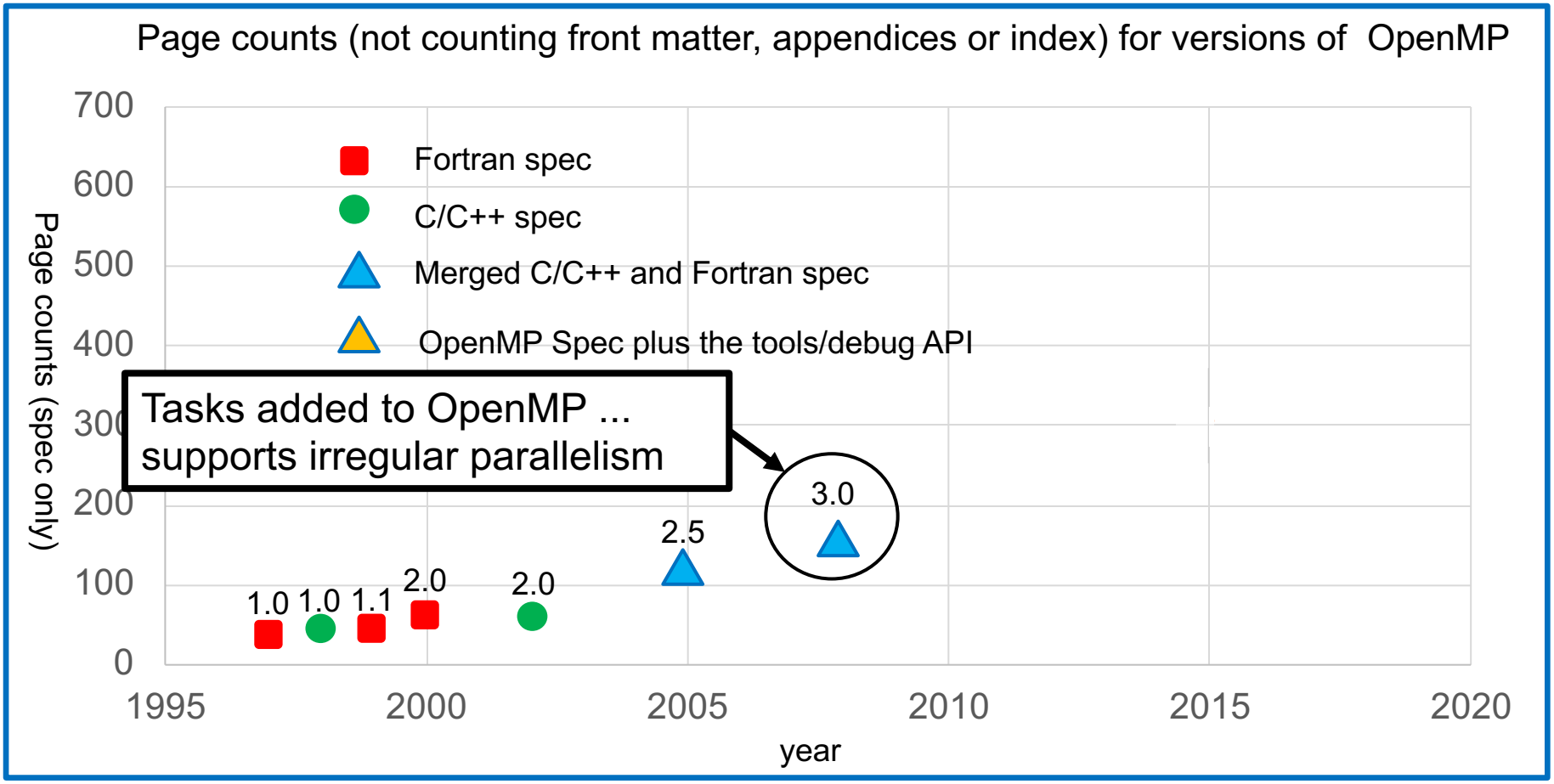


OpenMP Basic Definitions: Basic Solution Stack



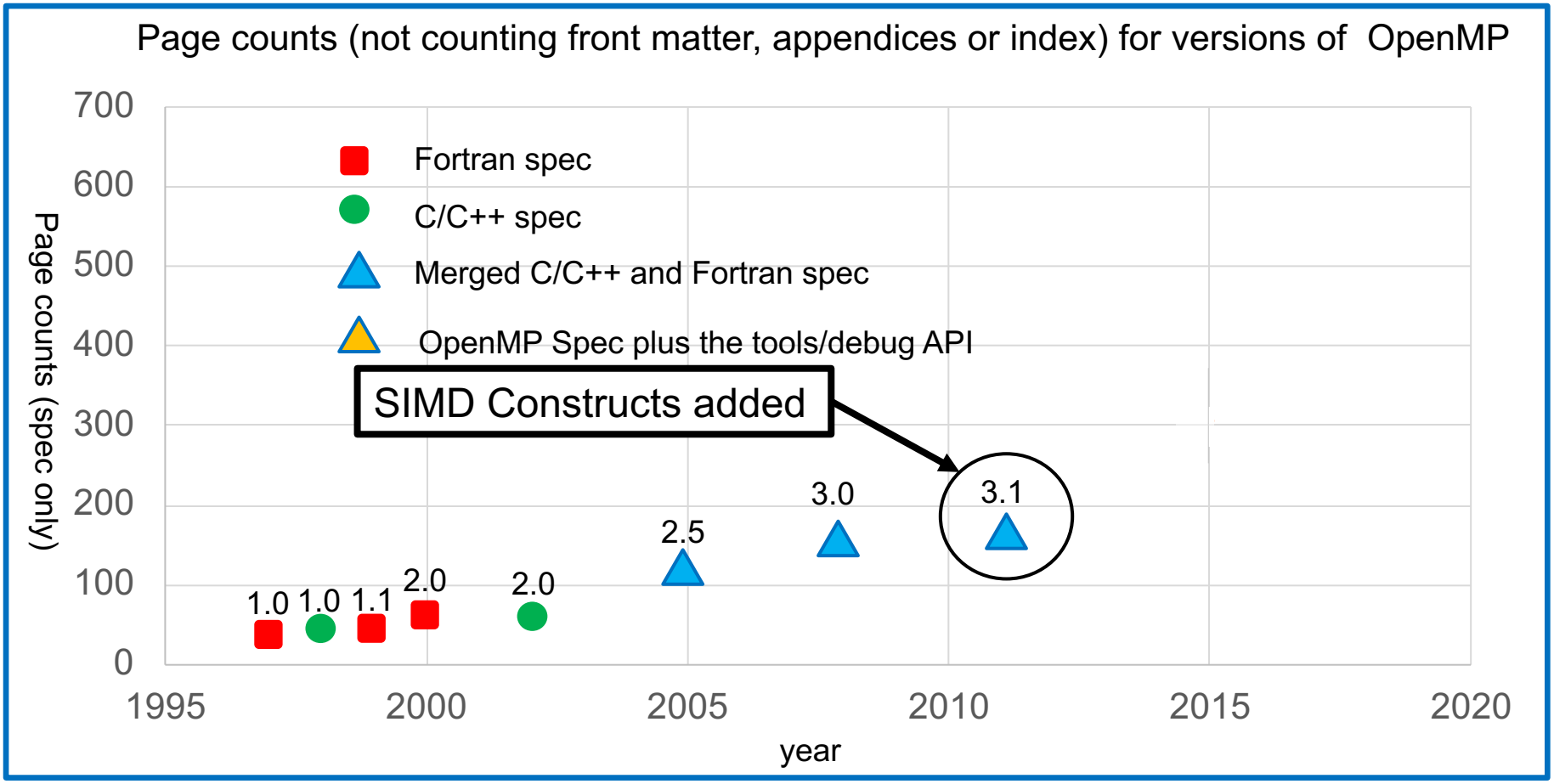
The Growing OpenMP Specification

- Tasks greatly expanded the scope of OpenMP.
- Required a major rewrite of the spec to express semantics in terms of tasks

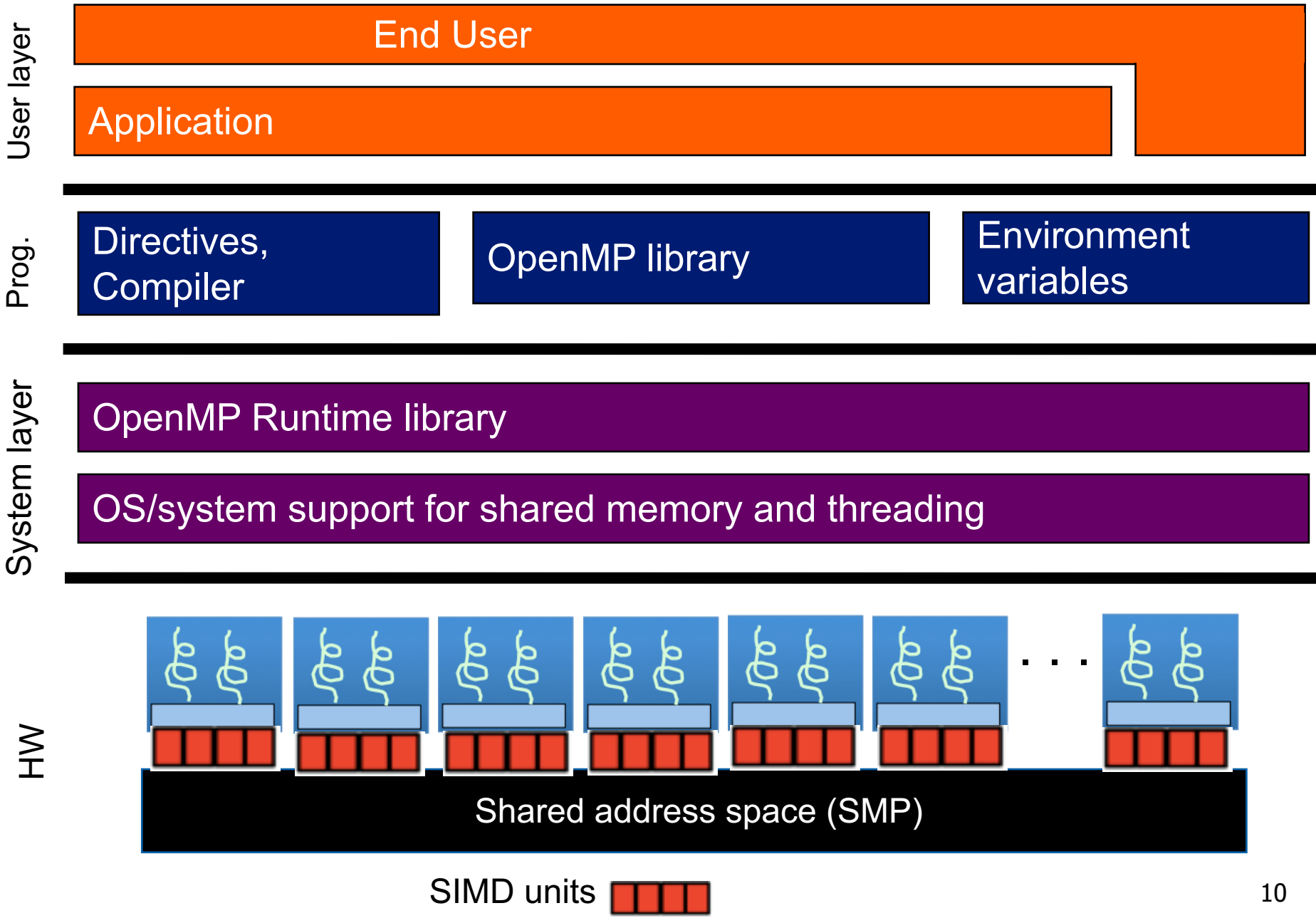


The Growing OpenMP Specification

- OpenMP started out in 1997 as a simple interface for the application programmers more versed in their area of science than computer science.
- The complexity has grown considerably over the years!

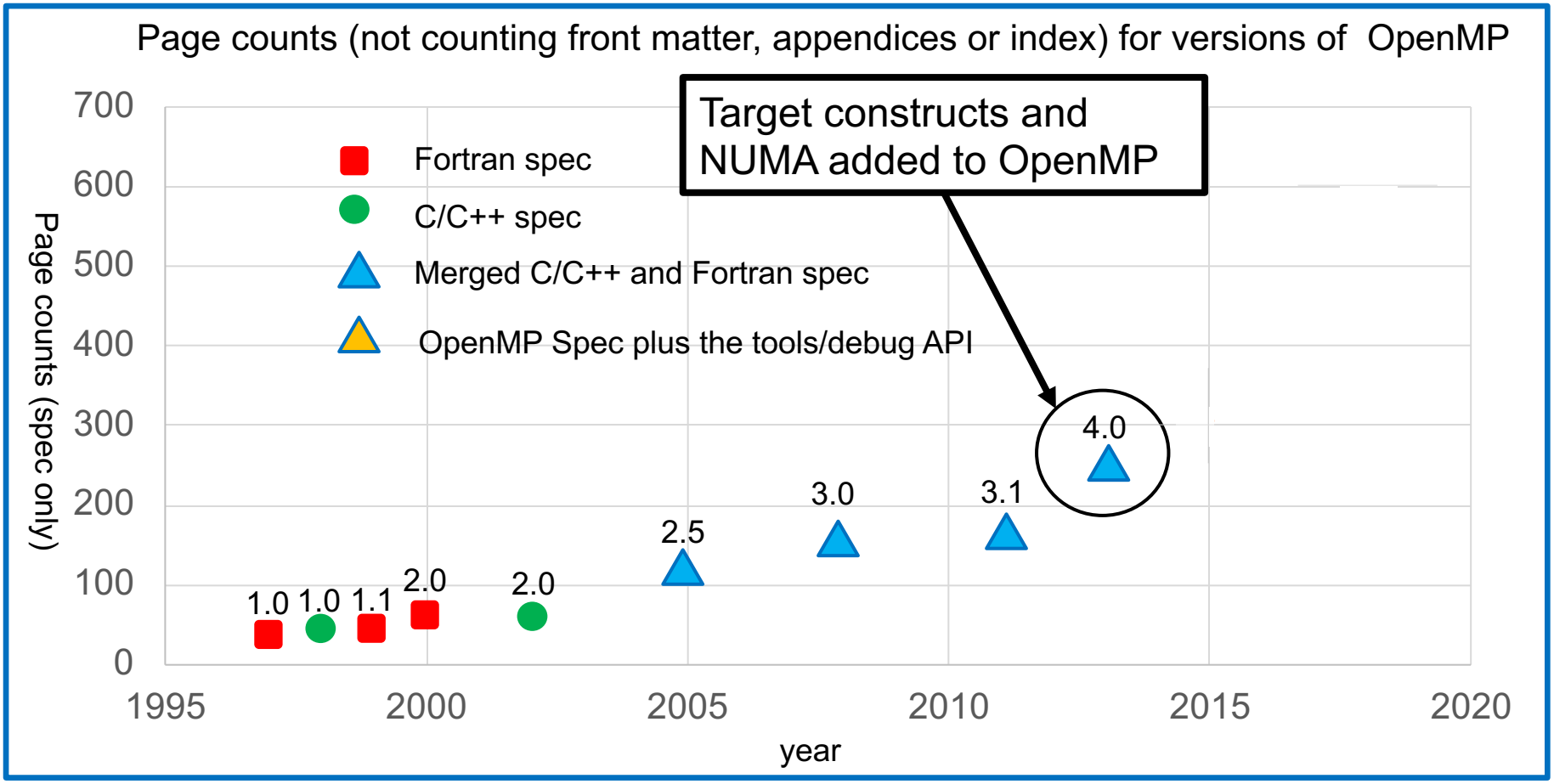


OpenMP Basic Definitions: Basic Solution Stack

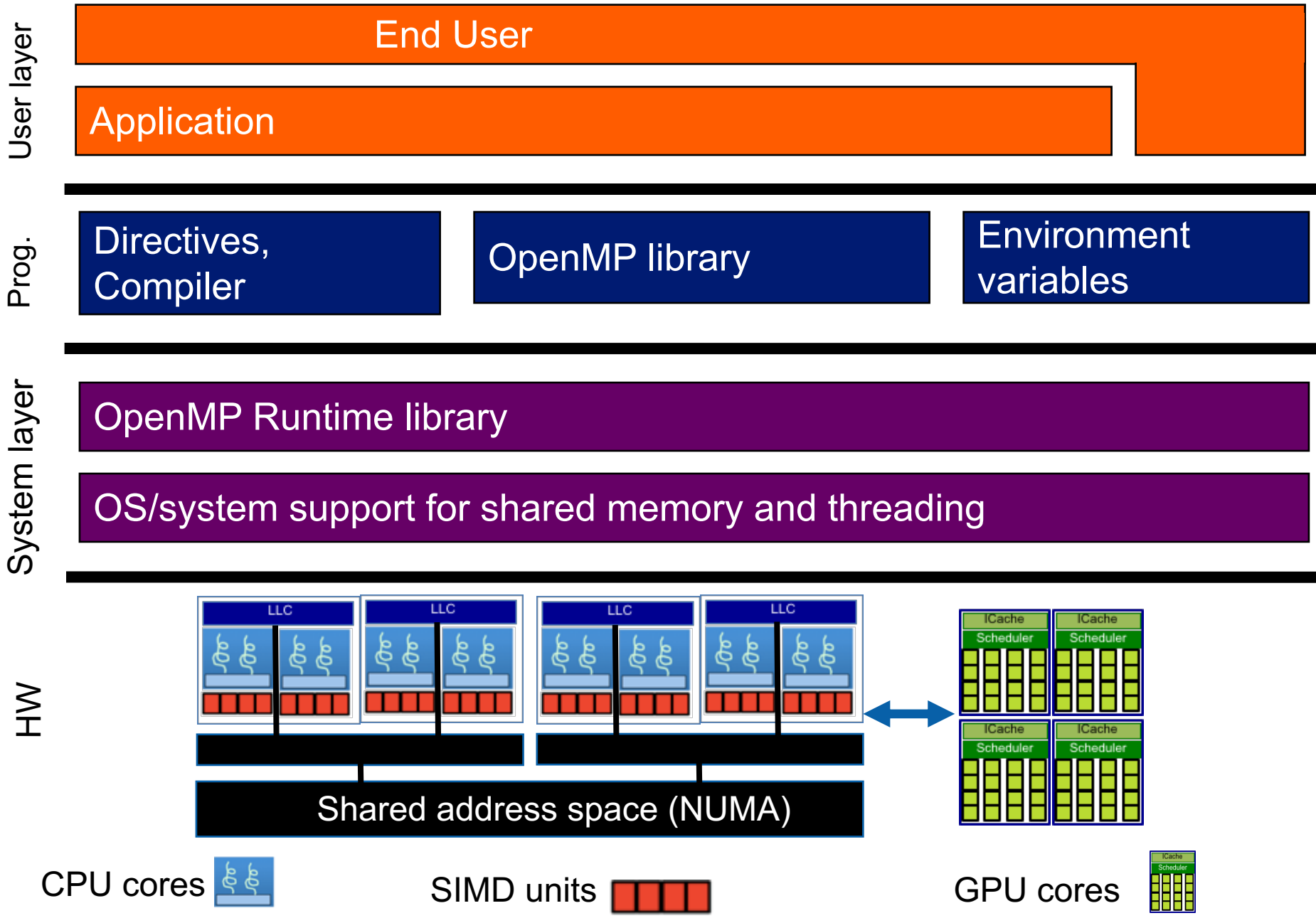


The Growing OpenMP Specification

- We knew the world was more complex than SMP when we started …. With 4.0 we finally embraced this complexity in OpenMP

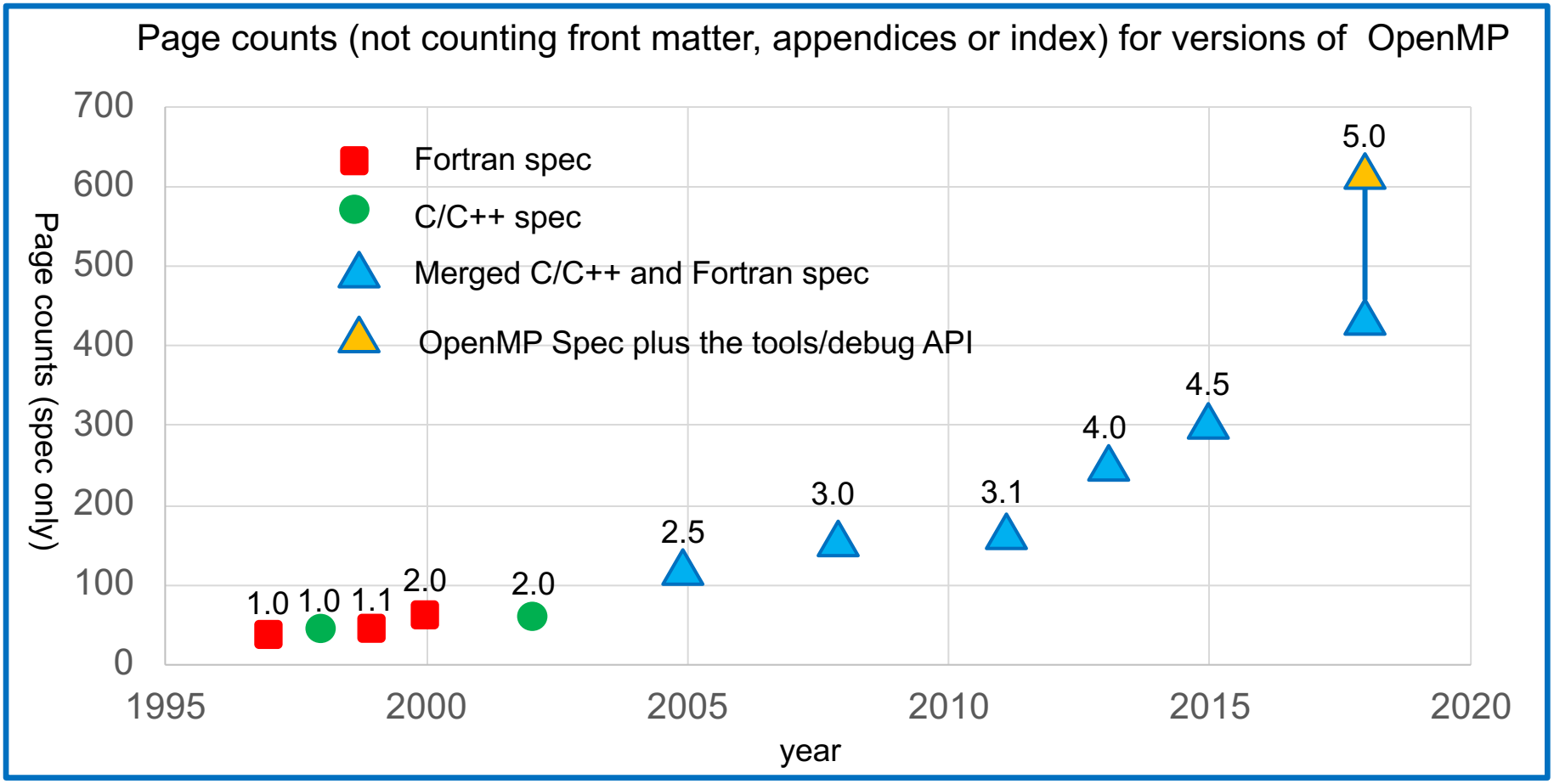


OpenMP basic definitions: Basic Solution stack



The Growing OpenMP Specification

- And with 5.0, the complexity accelerates!!!!

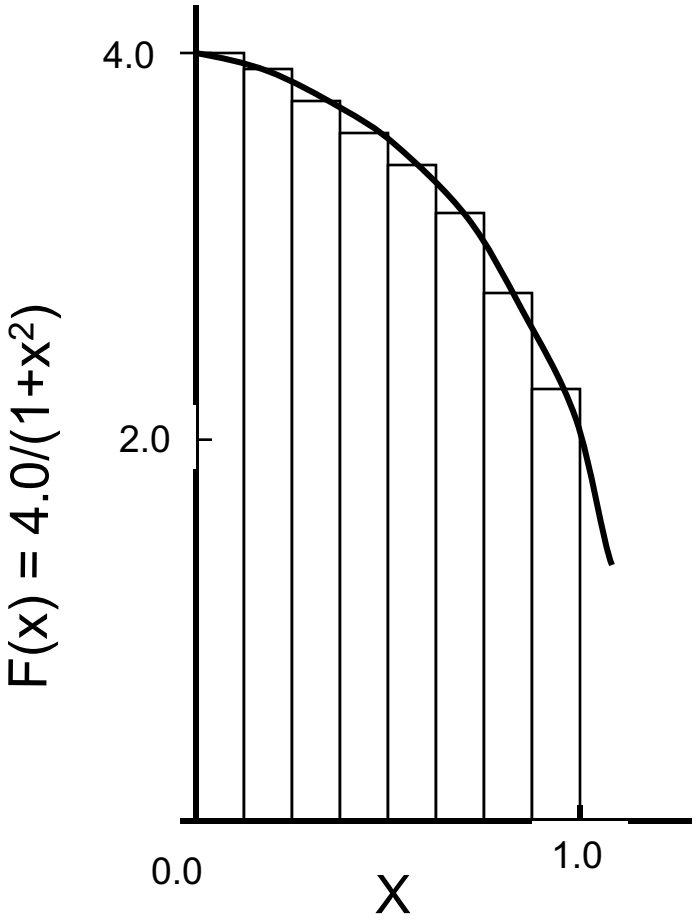


With a 600+ page spec, we risk scaring people away from OpenMP

- Can we make OpenMP less scary?
- What if we just focused on the subset of OpenMP that people actually use?
- … which begs the question … How do people most use OpenMP?

An Interesting Problem to Play With

Numerical Integration



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

Serial PI Program

```
static long num_steps = 100000;
double step;
int main ()
{           int i;      double x, pi, sum = 0.0;

            step = 1.0/(double) num_steps;

            for (i=0;i< num_steps; i++){
                        x = (i+0.5)*step;
                        sum = sum + 4.0/(1.0+x*x);
            }
            pi = step * sum;
}
```

See C/pi.c or Fortran/pi.f

The SPMD Pi program (C)

```
#include <omp.h>
#define MAX_THREADS 4
static long num_steps = 100000000;
int main ()
{
    int i,j;
    double step, pi, full_sum = 0.0, start_time, run_time, sum[MAX_THREADS];
     step = 1.0/(double) num_steps;
     omp_set_num_threads(MAX_THREADS);
     start_time = omp_get_wtime();
     #pragma omp parallel
     {
        int i;
        int id = omp_get_thread_num();
        int numthreads = omp_get_num_threads();

        for (i=id,sum[id]=0;i< num_steps; i+=numthreads){
              double x = (i+0.5)*step;
              sum[id] = sum[id] + 4.0/(1.0+x*x);
        }
     }
     for(full_sum = 0.0, i=0;i<j;i++)
          pi += sum[i]*step;
    run_time = omp_get_wtime() - start_time;
}
```

The OpenMP Common Core:

#pragma omp parallel

void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()

setenv OMP_NUM_THREADS N

The fundamental building blocks of multithreading

The SPMD Pi program (Fortran)

```
      PROGRAM MAIN
      USE OMP_LIB
      INTEGER i, j, id, numthreads, nthreads
      INTEGER, PARAMETER :: num_steps=100000000
      INTEGER, PARAMETER :: MAX_THREADS=4
      REAL*8 pi, real_sum, step, full_sum, x, start_time, run_time,sum(0:MAX_THREADS-1)
      step = 1.0/num_steps
      start_time = OMP_GET_WTIME()
     CALL OMP_SET_NUM_THREADS(j)
     start_time = omp_get_wtime()
!$OMP PARALLEL PRIVATE(id,x,numthreads)
        id = omp_get_thread_num()
        numthreads = OMP_GET_NUM_THREADS()
        sum(id) = 0.0
        DO i = id, num_steps-1, numthreads
          x = (i+0.5)*step
          sum(id) = sum(id) + 4.0/(1.0+x*x)
        ENDDO
!$OMP END PARALLEL
        pi = 0.0
        DO i = 0, nthreads-1
          full_sum = full_sum + sum(i)*step
        ENDDO
      run_time = OMP_GET_WTIME() - start_time
      STOP
      END
```

The OpenMP Common Core:

#pragma omp parallel

void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()

setenv OMP_NUM_THREADS N

shared(list), private(list), firstprivate(list)
default(none)

Controlling the data environment … absolutely essential for Fortran

The SPMD Pi program (eliminate false sharing)

```
#include <omp.h>
#define MAX_THREADS 4
static long num_steps = 100000000;
int main ()
{
     int i,j;
     double step, pi=0.0, full_sum = 0.0, start_time, run_time;
      step = 1.0/(double) num_steps;
      omp_set_num_threads(MAX_THREADS);
      start_time = omp_get_wtime();
      #pragma omp parallel
      {
          int i;
          int id = omp_get_thread_num();
          int numthreads = omp_get_num_threads();
          double sum=0.0;
          for (i=id;i< num_steps; i+=numthreads){
                double x = (i+0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
          }
          #pragma omp critical
                  pi += sum*step
      }
      run_time = omp_get_wtime() - start_time;
}
```

The OpenMP Common Core:

#pragma omp parallel

void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()

setenv OMP_NUM_THREADS N

shared(list), private(list), firstprivate(list)
default(none)

#pragma omp barrier
#pragma omp critical

Synchronization to define ordering constraints and prevent data races

Parallel loop pi … the simple case

```
#include <omp.h>
static long num_steps = 100000;        double step;
void main ()
{    int i;   double x, pi, sum = 0.0;
      step = 1.0/(double) num_steps;
      #pragma omp parallel for reduction(+:sum)
                for (i=0;i< num_steps; i++){
                          double x = (i+0.5)*step;
                          sum = sum + 4.0/(1.0+x*x);
                }
       }
              pi = step * sum;
}
```

The OpenMP Common Core:

#pragma omp parallel

void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()

setenv OMP_NUM_THREADS N

#pragma omp for
#pragma omp parallel for
reduction(op:list)

Parallel Loops … the essence or OpenMP Programming Practice

shared(list), private(list), firstprivate(list)
default(none)

#pragma omp barrier
#pragma omp critical

Program: OpenMP Tasks

```
 include <omp.h>
static long num_steps = 100000000;
#define MIN_BLK  10000000
double pi_comp(int Nstart,int Nfinish,double step)
{   int i,iblk;
   double x, sum = 0.0,sum1, sum2;
   if (Nfinish-Nstart < MIN_BLK){
     #pragma omp for private(x) recucution(+:sum)
     for (i=Nstart;i< Nfinish; i++){
       x = (i+0.5)*step;
       sum = sum + 4.0/(1.0+x*x);
     }
   }
   else{
     iblk = Nfinish-Nstart;
     #pragma omp task shared(sum1)
          sum1 = pi_comp(Nstart,        Nfinish-iblk/2,step);
     #pragma omp task shared(sum2)
           sum2 = pi_comp(Nfinish-iblk/2, Nfinish,       step);
     #pragma omp taskwait
       sum = sum1 + sum2;
   }return sum;
}
```

```
int main ()
{
  int i;
  double step, pi, sum;
   step = 1.0/(double) num_steps;
   #pragma omp parallel
   {
      #pragma omp single
         sum =
            pi_comp(0,num_steps,step);
    }
     pi = step * sum;
}
```

The OpenMP Common Core:

#pragma omp parallel

void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()

setenv OMP_NUM_THREADS N

shared(list), private(list), firstprivate(list)
default(none)

#pragma omp barrier
#pragma omp critical

#pragma omp for
#pragma omp parallel for
reduction(op:list)

#pragma omp single

#pragma omp task
#pragma omp taskwait

Generalizing OpenMP to irregular parallelism

Plus some Extra features we can't show with pi

```
double A[big], B[big], C[big];

#pragma omp parallel
{
        int id=omp_get_thread_num();
        A[id] = big_calc1(id);
#pragma omp barrier
#pragma omp for schedule(static, 10)
        for(i=0;i<N;i++){C[i]=big_calc3(i,A);}
#pragma omp for schedule(dynamic) nowait
        for(i=0;i<N;i++){ B[i]=big_calc2(C,  i); }
        A[id] = big_calc4(id);
}
```

The OpenMP Common Core:

The 21 constructs most OpenMP programmers use exclusively

#pragma omp parallel

void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()

setenv OMP_NUM_THREADS N

#pragma omp for
#pragma omp parallel for
reduction(op:list)

#pragma omp single

#pragma omp task
#pragma omp taskwait

shared(list), private(list), firstprivate(list)
default(none)

#pragma omp barrier
#pragma omp critical

nowait

schedule (static [,chunk])
schedule(dynamic [,chunk])

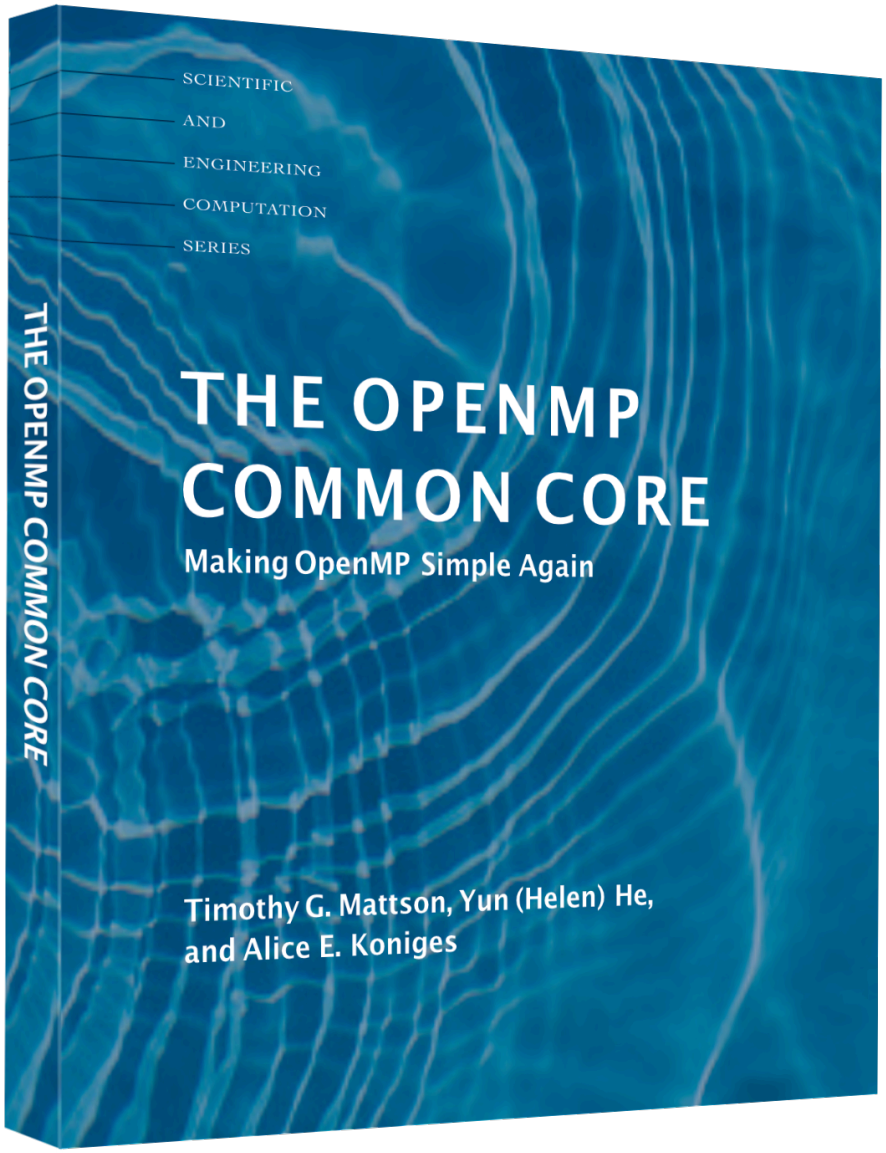
The OpenMP Common Core: An elegant API from a more civilized age

The OpenMP Common Core

- People should master the common core, and then pick up what the need from the rest of OpenMP “as needed”

- We have a new book that introduces the Common Core:
  - Part 1: Intro to parallel programming.
  - Part 2: the OpenMP common core
  - Part 3: Beyond the common core

- The book is geared at people new to OpenMP and OpenMP educators …. Though the memory model chapter in part 3 will be beneficial for more advanced programmers.



The OpenMP Common Core: Most OpenMP programs only use these 21 items

OpenMP pragma, function, or clause	Concepts
#pragma omp parallel	Parallel region, teams of threads, structured block, interleaved execution across threads.
void omp_set_thread_num() int omp_get_thread_num() int omp_get_num_threads()	Default number of threads and internal control variables. SPMD pattern: Create threads with a parallel region and split up the work using the number of threads and the thread ID.
double omp_get_wtime()	Speedup and Amdahl's law. False sharing and other performance issues.
setenv OMP_NUM_THREADS N	Setting the internal control variable for the default number of threads with an environment variable
#pragma omp barrier #pragma omp critical	Synchronization and race conditions. Revisit interleaved execution.
#pragma omp for #pragma omp parallel for	Worksharing, parallel loops, loop carried dependencies.
reduction(op:list)	Reductions of values across a team of threads.
schedule (static [,chunk]) schedule(dynamic [,chunk])	Loop schedules, loop overheads, and load balance.
shared(list), private(list), firstprivate(list)	Data environment.
default(none)	Force explicit definition of each variable's storage attribute
nowait	Disabling implied barriers on workshare constructs, the high cost of barriers, and the flush concept (but not the flush directive).
#pragma omp single	Workshare with a single thread.
#pragma omp task #pragma omp taskwait	Tasks including the data environment for tasks.