What Was Missing ?

Before OpenMP 3.0

- Constructs worked well for many cases
- But OpenMP's Big Brother had to see everything
  - → Loops with a known length at run time
  - → Finite number of parallel sections
  - → ....
- This didn't work well with certain common problems
  - → Linked lists and recursive algorithms being the cases in point
- Often, a solution was feasible, but ugly at best

OpenMP

Today's All New Episode

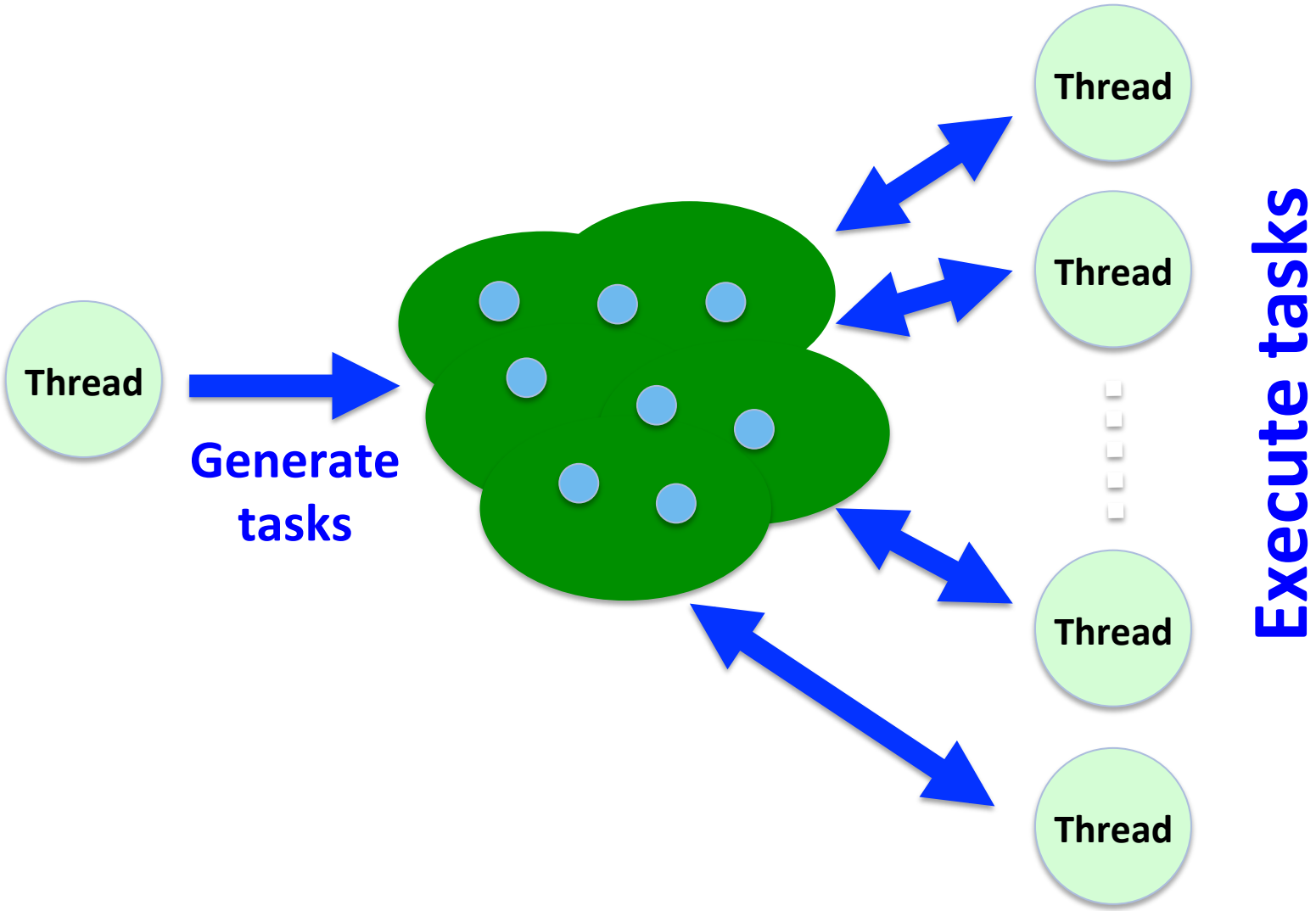
TASKING

SC13 Denver, CO 2013

Tasking in OpenMP

- Tasking was introduced in OpenMP 3.0
- Until then it was impossible to efficiently and easily implement certain types of parallelism
- The initial functionality was very simple by design
  - → The idea was (and still) is to augment tasking as we collectively gain more insight and experience
- Note that tasks can be nested
  - → But not for the faint of heart

The Tasking Concept In OpenMP



Who Does What And When ?

Developer

Use a pragma to specify where the tasks are

(The assumption is that all tasks can be executed independently)

OpenMP runtime system

- When a thread encounters a task construct, a new task is generated
- The moment of execution of the task is up to the runtime system
- Execution can either be immediate or delayed
- Completion of a task can be enforced through **task synchronization**

The Tasking Construct

```
#pragma omp task
```

```
!$omp task
```

Defines a task

Task Synchronization

OpenMP

There are two task synchronization constructs

#pragma omp barrier

```
#pragma omp barrier
```

```
!$omp barrier
```

#pragma omp taskwait

```
#pragma omp taskwait
```

```
!$omp taskwait
```

Task Completion

Explicitly wait on the completion of child tasks:

```
#pragma omp taskwait
```

```
!$omp flush taskwait
```

Tasking Explained By Ways Of One Example

A Simple Plan

OpenMP

Your Task for Today:

Write a program that prints either "A race car" or "A car race" and maximize the parallelism

Tasking Example/1

```
$ cc -fast hello.c
$ ./a.out
A race car
$
```

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

          printf("A ");
          printf("race ");
          printf("car ");

   printf("\n");
   return(0);
}
```

What will this program print ?

Tasking Example/2

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
         printf("A ");
         printf("race ");
         printf("car ");

   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Tasking Example/3

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car A race car
```

Note that this program could (for example) also print

"A A race race car car" or

"A race A car race car", or

"A race A race car car", or

.....

But I have not observed this (yet)

Tasking Example/4

```
#include <stdlib.h
#include <stdio.h

int main(int argc

   #pragma omp parallel
   {
     #pragma omp single
      {
          printf("A ");
          printf("race ");
          printf("car ");
     }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Tasking Example/5

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
```

But of course now only 1 thread executes .....

Tasking Example/6

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");
         #pragma omp task
          {printf("race ");}
         #pragma omp task
          {printf("car ");}
     }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Tasking Example/7

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
$ ./a.out
A race car
$ ./a.out
A car race
$
```

Tasks can be executed in arbitrary order

Another Simple Plan

OpenMP

You did well and quickly, so here is a final task to do

Have the sentence end with "is fun to watch"
(hint: use a print statement)

Tasking Example/8

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");
         #pragma omp task
          {printf("race ");}
         #pragma omp task
          {printf("car ");}
        printf("is fun to watch ");
     }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Tasking Example/9

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out

A is fun to watch race car
$ ./a.out

A is fun to watch race car
$ ./a.out

A is fun to watch car race
$
```

Tasks are executed at a task execution point

Tasking Example/10

```
int main(int argc, char

  #pragma omp parallel
  {
    #pragma omp single
     {
        printf("A ");
        #pragma omp task
          {printf("car ");}
        #pragma omp task
          {printf("race ");}
        #pragma omp taskwait
        printf("is fun to watch ");
     }
   } // End of parallel region

   printf("\n");return(0);
}
```

What will this program print using 2 threads ?

Tasking Example/11

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
$
A car race is fun to watch
$ ./a.out
A car race is fun to watch
$ ./a.out
A race car is fun to watch
$
```

Tasks are executed first now