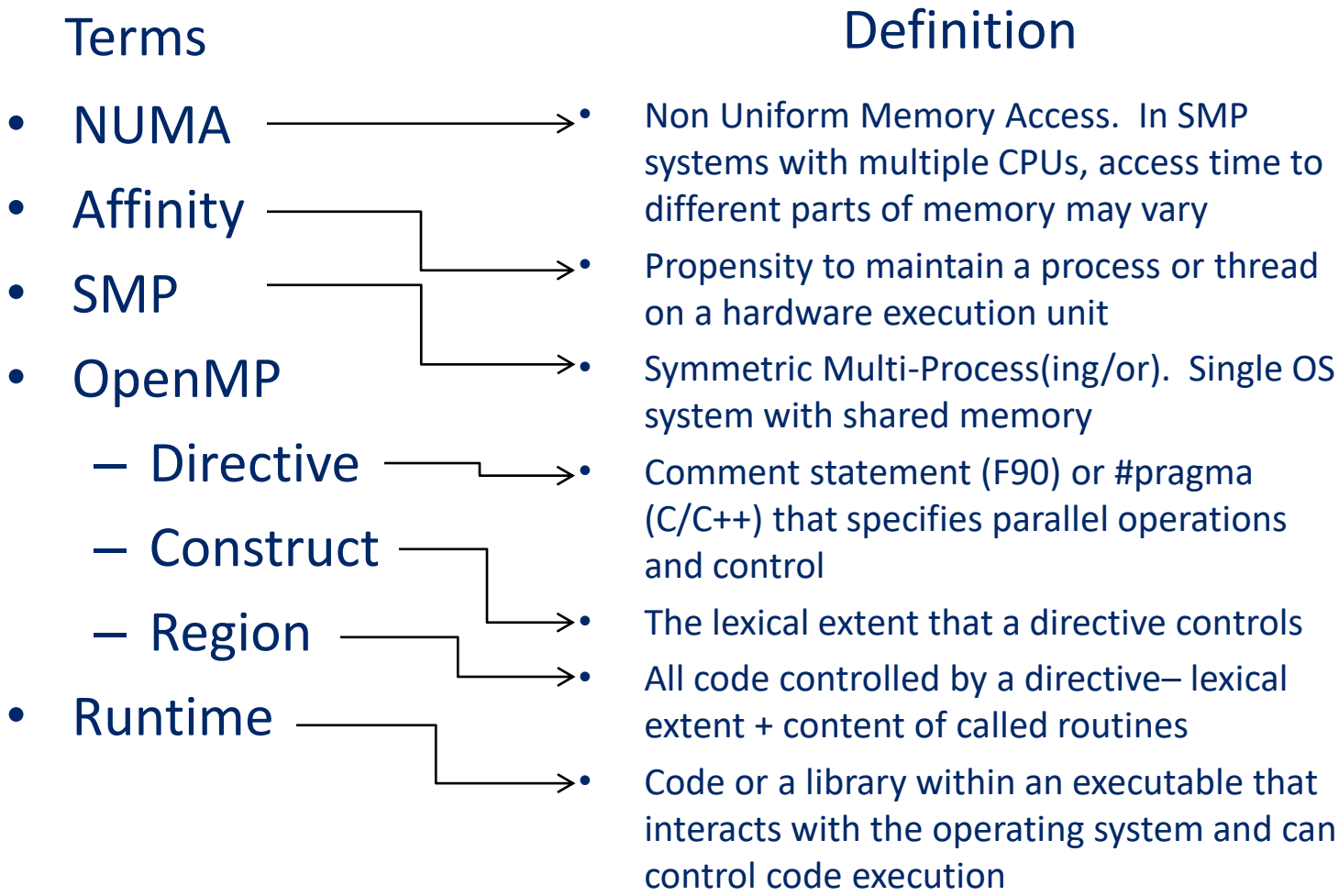
COMP 364 / 464
High Performance Computing

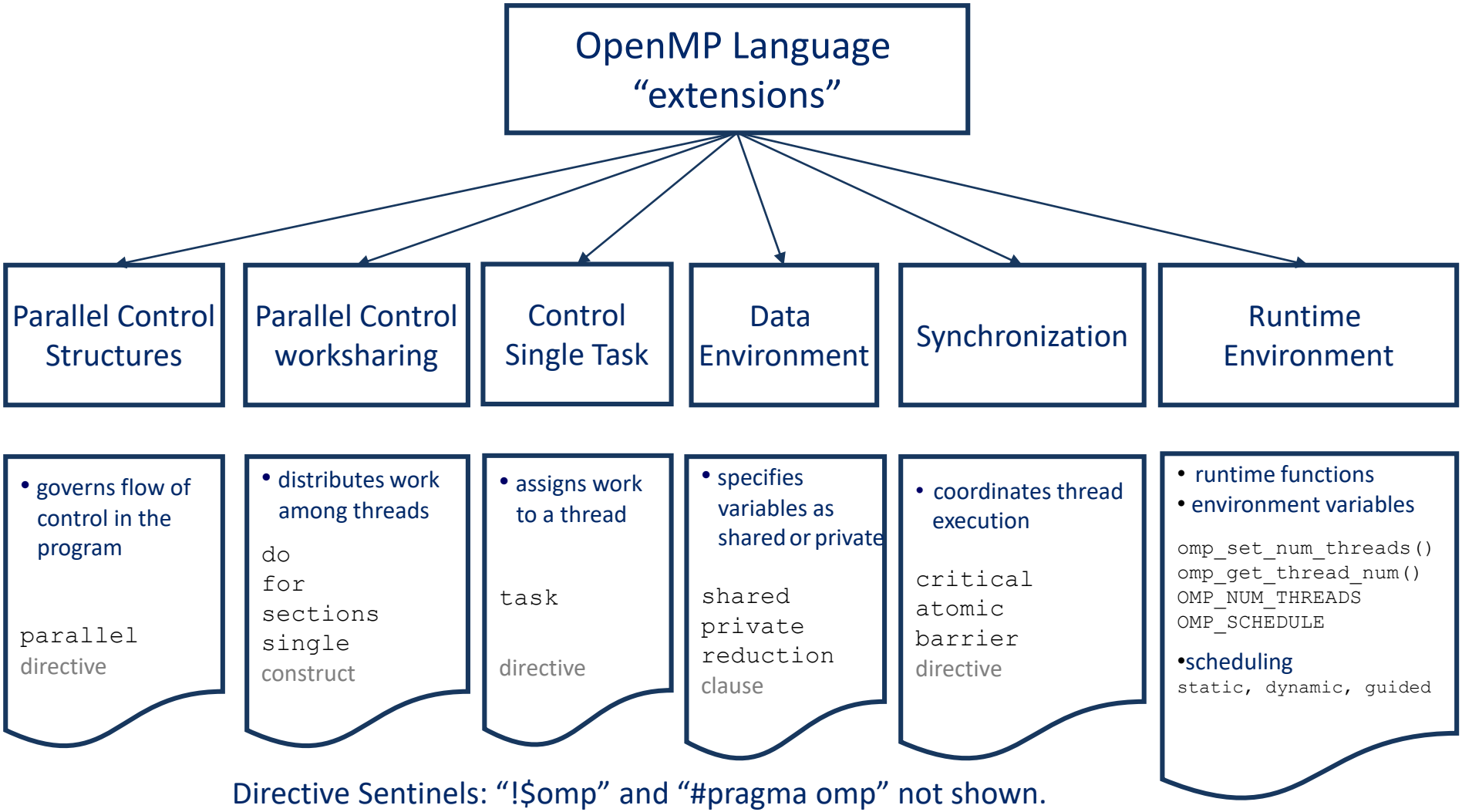
OpenMP:

task parallelism

Computing Terminology



OpenMP Constructs



Directive Sentinels: "!$omp" and "#pragma omp" not shown.

OpenMP Synchronization

```
InitializeQueue( allWorkItems );
#pragma omp parallel
{
   bool notEmpty = true;
   while (notEmpty)
   {
      workItemType workItem;

      // Pop a task off the queue one thread at a time.
      #pragma omp critical
      {
         notEmpty = PopQueue(&workItem);
      }

      // Do some work (if necessary)
      if (notEmpty)
         foo(workItem);
   }
}
// All threads block (or sync) here.
```

OpenMP Tasks

```
List *head = InitializeListItems( allItems ); // Serially
#pragma omp parallel shared(default)
{
   #pragma omp single nowait
   {
      List *p = head;
      for (List *p = head; p != NULL; p = p->next) {
         #pragma omp task
         doSomething (p); // p is private to each task.
      }

      #pragma omp taskwait

      List *p = head;
      for (List *p = head; p != NULL; p = p->next) {
         #pragma omp task
         doSomethingElse (p);
      }
   }
} // All threads block (or sync) here.
```

OpenMP Recursive Tasks

```
void doSomething (Node *p)
{
   if (p->size > thresholdSize)
      // Do some actual work.
   else
   {
      // Split up workitem *p further.
      Node *q = <Something to halve p>

      #pragma omp task
      doSomething (q);

      #pragma omp task
      doSomething (p);

      // Wait for child tasks to finish.
      #pragma omp taskwait
   }
   return;
}
```

OpenMP Recursive Tasks

```
void doSomething (Node *p, int depth)
{
   if (p->size > thresholdSize)
      // Do some actual work.
   else
   {
      // Split up workitem *p further.
      Node *q = <Something to halve p>

      #pragma omp task
      doSomething (q, depth+1);

      #pragma omp task if (depth < thresholdDepth)
      doSomething (p, depth+1);

      // Wait for child tasks to finish.
      #pragma omp taskwait
   }
   return;
}
```

OpenMP Dependent Tasks

```
ListNode *head = InitializeListItems( allItems ); // Serially
#pragma omp parallel shared(default)
#pragma omp single nowait
{
   ListNode *p = head;
   for (List *p = head; p != NULL; p = p->next)
   {
      ListItem *item = p->item;

      #pragma omp task shared(item) depend(out: item)
      doSomethingA (item);

      #pragma omp task shared(item) depend(in: item)
      doSomethingB (item);

      #pragma omp task shared(item) depend(in: item)
      doSomethingC (item);
   }
} // All threads block (or sync) here.
```