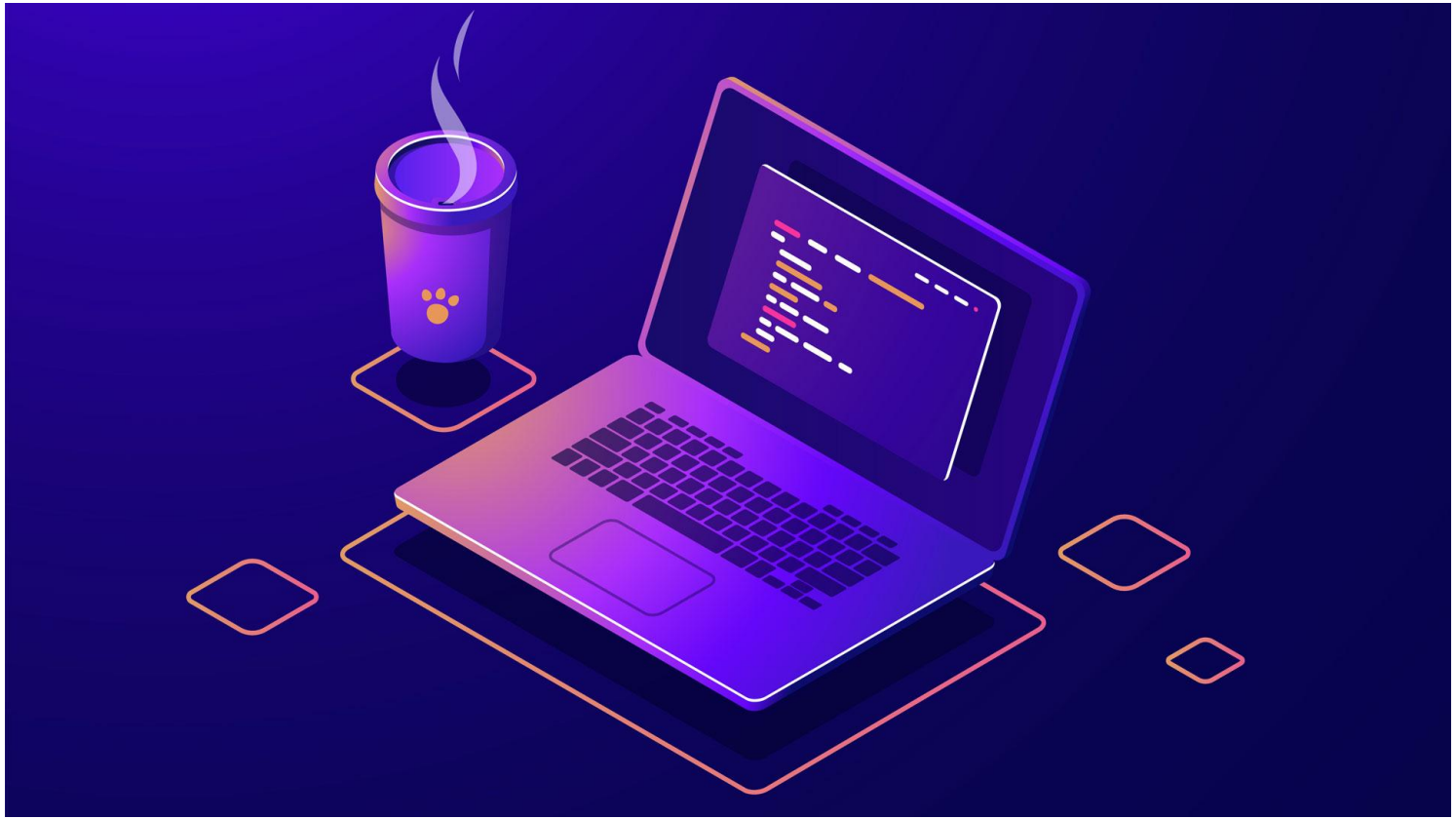
Recursive Task Decomposition



Task Decomposition Strategies

- **Linear** task decomposition
  - A task is a "code block" or a procedure invocation
- **Iterative** task decomposition
  - Tasks found in iterative constructs, such as loops (countable or uncountable)
- **Recursive** task decomposition
  - Tasks found in divide–and–conquer problems and other recursive problems

Linear Task Decomposition

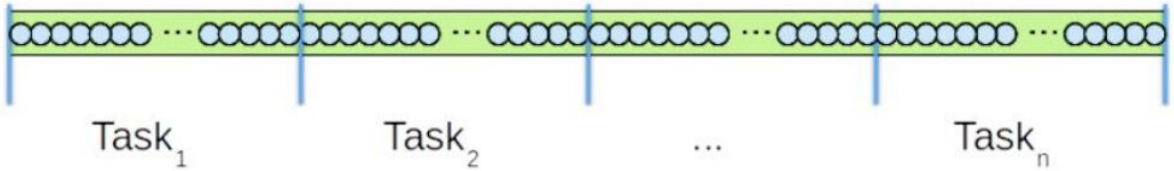
A task is a “ code block “ or a procedure invocation

```
int main() {
start_task(“init_A”);
initialize(A,N);
end_task(“init_A”);

start_task(“init_B”);
initialize(B,N);
end_task(“init_B”);

}
```

Iterative Task Decomposition

A task is a chunk of iterations of a loop, as for example, in the sum of two vectors

```
void vector_add(int *A, int *B, int *C, int n) {
   for (int i=0; i< n; i++) C[i] = A[i] + B[i];
}

void main() {
   ....
   vector_add(a, b, c, N);
   ...
}
```



Iterative Task Decomposition

Single loop iteration:

```
void vector_add(int *A, int *B, int *C, int n) {
   for (int i=0; i< n; ii++)
                .start_task("singleit");
        C[i] = A[i] + B[i];
                .end_task("singleit");
}
```

Chunk of loop iterations:

```
#define BS 16
void vector_add(int *A, int *B, int *C, int n) {
   for (int ii=0; ii< n; ii+=BS) {
                .start_task("chunkit");
        for (i = ii; i < min(ii+BS, n), i++)
            C[i] = A[i] + B[i];
                .end_task("chunkit");
        }
}
```

Iterative Task Decomposition

List of elements, traversed using an uncountable (while) loop

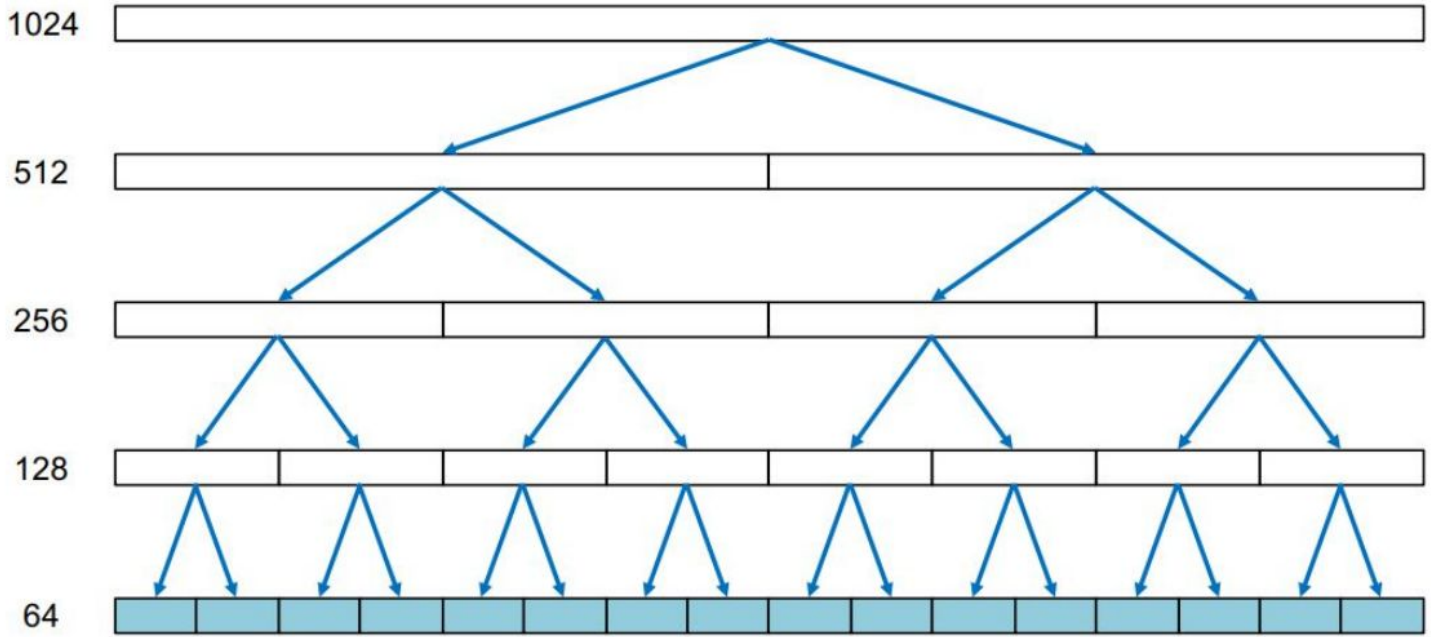
```
int main() {
struct node *p;

p = init_list(n);
...

while (p != NULL) {
            start_task("computeNode");
    process_work(p);
            end_task("computeNode");
    p = p->next;
    }
...
}
```

Recursive Task Decomposition

Sum of two vectors by recursively dividing the problem into smaller sub-problems

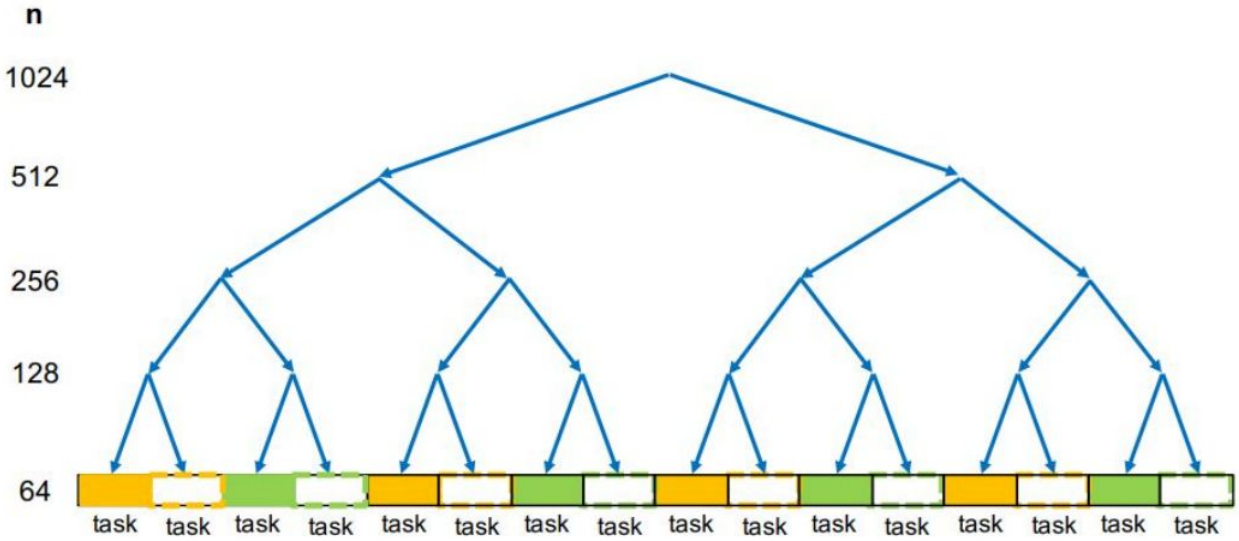
```
#define N 1024
#define MIN_SIZE 64

void vector_add(int *A, int *B, int *C, int n) {
   for (int i=0; i< n; i++) C[i] = A[i] + B[i];
}

void rec_vector_add(int *A, int *B, int *C, int n) {
   if (n>MIN_SIZE) {
       int n2 = n / 2;
       rec_vector_add(A, B, C, n2);
       rec_vector_add(A+n2, B+n2, C+n2, n-n2);
   }
   else vector_add(A, B, C, n);
}
void main() {
   ....
   rec_vector_add(a, b, c, N);
   ...
}
```

Recursive Task Decomposition



Two possible decomposition strategies

- **Leaf strategy**: a task corresponds with each invocation of `vector_add` once the recursive invocations stop

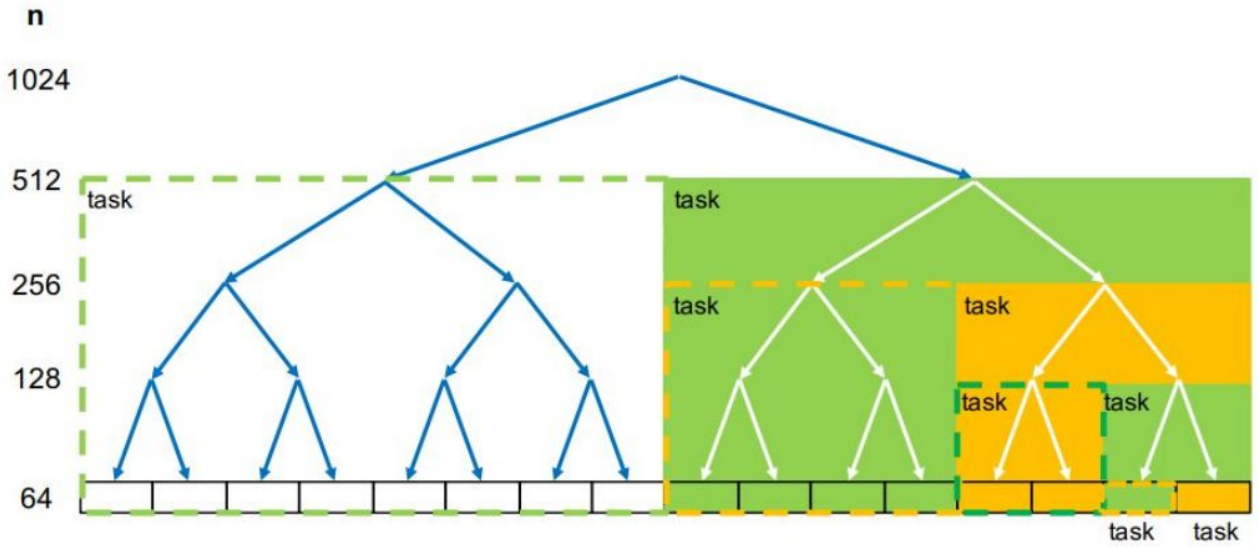


Leaf Strategy

```
#define N 1024
#define MIN_SIZE 64

void vector_add(int *A, int *B, int *C, int n) {
   for (int i=0; i< n; i++) C[i] = A[i] + B[i];
}

void rec_vector_add(int *A, int *B, int *C, int n) {
   if (n>MIN_SIZE) {
       int n2 = n / 2;
       rec_vector_add(A, B, C, n2);
       rec_vector_add(A+n2, B+n2, C+n2, n-n2);
   }
   else
       {
                start_task("leaftask");
       vector add(A, B, C, n);
                end_task("leaftask");
       }
}
void main() {
   ....
   rec_vector_add(a, b, c, N);
   ...
}
```

Two possible decomposition strategies

- **Tree strategy**: a task corresponds with each invocation of `rec_vector_add` during the *parallel* recursive execution

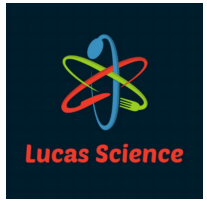


Tree Strategy

```
#define N 1024
#define MIN_SIZE 64

void vector_add(int *A, int *B, int *C, int n) {
   for (int i=0; i< n; i++) C[i] = A[i] + B[i];
}

void rec_vector_add(int *A, int *B, int *C, int n) {
   if (n>MIN_SIZE) {
       int n2 = n / 2;
                start_task("treetask1");
       rec_vector_add(A, B, C, n2);
               _end_task("treetask1");
               _start_task("treetask2");
       rec_vector_add(A+n2, B+n2, C+n2, n-n2);
                end_task("treetask2");
   }
   else vector_add(A, B, C, n);
}
void main() {
   ....
   rec_vector_add(a, b, c, N);
   ...
}
```

Instructor Social Media

Youtube: Lucas Science



Instagram: lucaasbazilio

Twitter: lucasebazilio