Punteros

Alberto de la Encina Vara

Punteros

- Declaración: tipoDato* nombreVar
- Acceso al valor: *nombreVar
- Inicialización: =, NULL, new
- Dirección: &nombreVar

Memoria dinámica:

- Pedir memoria: new, new ...[size]
- Liberar memoria: delete, delete[]

Punteros a estructuras:

- Acceso al valor: varPtrStruct->campo

```
int elInt;
int* ptrInt;        /* Puntero a entero */
int** ptrPtrInt;    /* Puntero a puntero entero */

ptrPtrInt = &ptrInt; /* Puntero contiene dirección de puntero entero */
ptrInt = &elInt;    /* Puntero contiene dirección de entero */

elInt = 2;          /* Asignación directa */
*ptrInt = 20;       /* Asignación indirecta */
**ptrPtrInt = 30;   /* Asignación con doble indirección */
```

Estructuras con memoria dinámica

Alberto de la Encina Vara

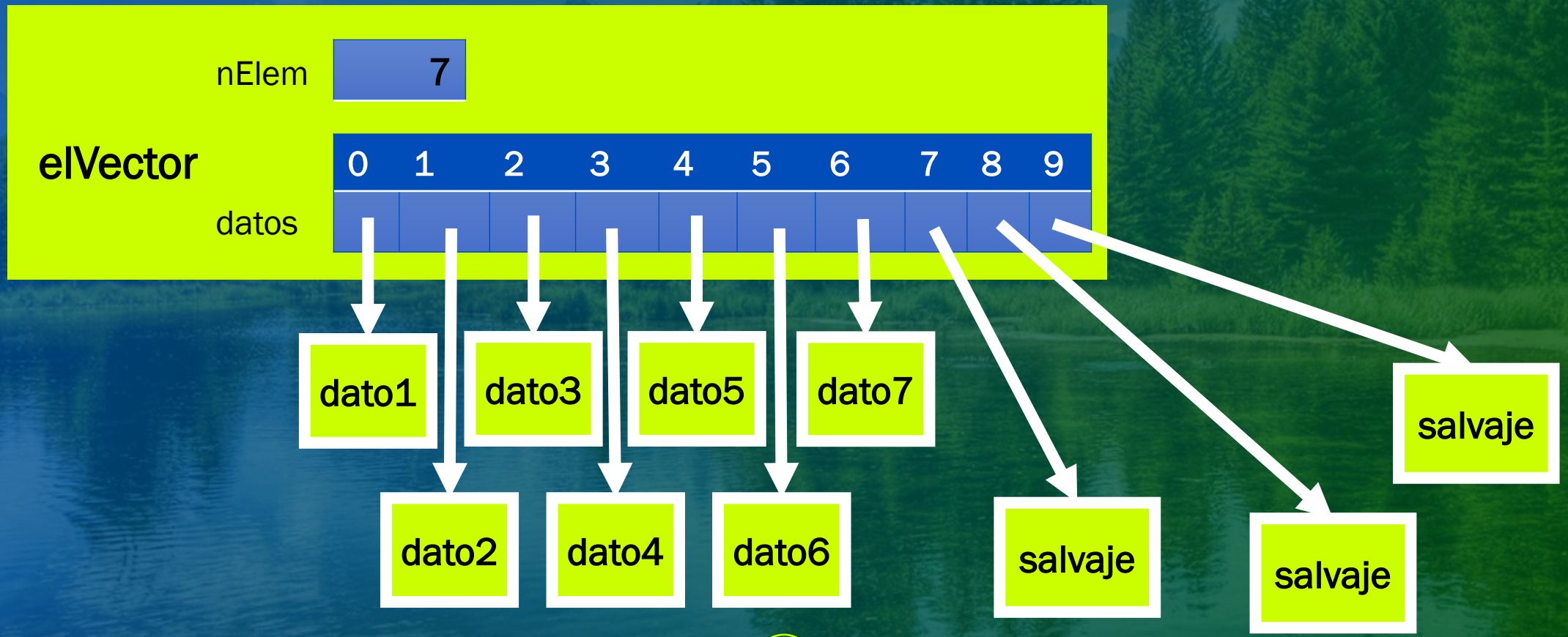
Dinámica: Array de Punteros

```
typedef tDato* tPtrDato;
typedef struct {
                    int nElem;
                    tPtrDato datos[MAX_DATOS];
          } tArrayPtrDato;


void inicializaArray( tArrayPtrDato & elArray){
          elArray.nElem = 0;
}
```

- Menos costosa la ordenación de estructuras “grandes”.

Dinámica: Array de Punteros



Dinámica: Array de Punteros

```
void deleteArrayPtr(tArrayPtrDato & elArray ){
        for (int i = 0; i < elArray.nElem; i++)
                {  delete elArray.datos[i];  }
        elArray.nElem = 0;
}
void insertarAlFinal (tArrayPtrDato & elArray , tDato &const elDato) {
                if  ( !estaLleno(elArray) ) {
                                elArray.datos[elArray.nElem] = new tDato;
                                *elArray.datos[elArray.nElem] = elDato;
                                elArray.nElem++;
                }
}
bool estaLleno (tArrayPtrDato & elArray)
                { return elArray.nElem == MAX_DATOS; }
```

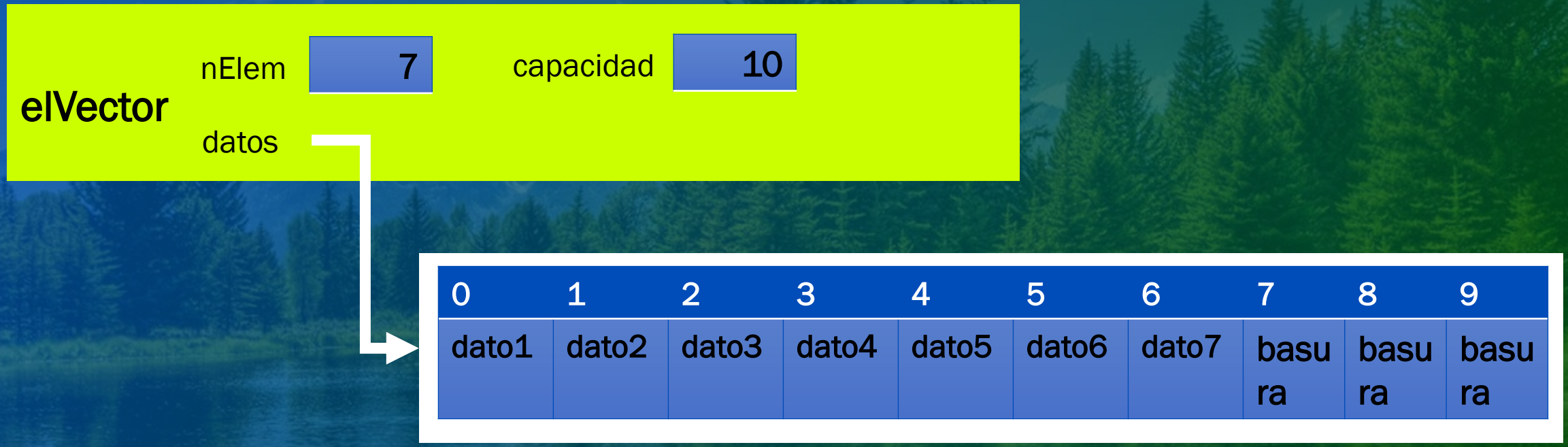
Dinámica: Vector (array dinámico)

```
typedef struct {
                int nElem;
                int capacidad;
                tDato *datos;
        } tVector;

void inicializaVector( tVector & elVector, int capacidad_INI){
        elVector.nElem = 0;
        elVector.capacidad = capacidad_INI;
        elVector.datos = new tDato [elVector.capacidad];
}
```

- El programa deja de estar limitado en tamaño en compilación, se crea en ejecución

Dinámica: Vector (array dinámico)



Dinámica: Vector (array dinámico)

```
void deleteVector( tVector & elVector ){
        delete [] elVector.datos;
        elVector.nElem = 0;
}
void insertarAlFinal ( tVector & elVector, tDato &const elDato) {
        if (estaLleno(elVector))
                redimensiona (elVector, 2* elVector.capacidad);
        elVector.datos[elVector.nElem] = elDato;
        elVector.nElem++;
}
bool estaLleno ( tVector & elVector )
        { return elVector.nElem == elVector.capacidad; }
```

- Además el vector puede crecer al añadir elementos.

Dinámica: Vector (array dinámico)

```
void redimensiona(tVector& elVector, int capacidad) {
        // Guardar la referencia a los tableros actuales y su cantidad de elementos
        tDato *auxArray = elVector.datos;
        int numElems = elVector.nElem;
        // Actualizar adecuadamente elVector
        inicializaVector(elVector, capacidad);
        // Copiar los elementos al nuevo array de elVector
        for (int i = 0; i < numElems; i++) { insertarAlFinal(elVector, auxArray[i]); }
        // Borrar la memoria apuntada por auxArray
        delete [] auxArray;
}
```

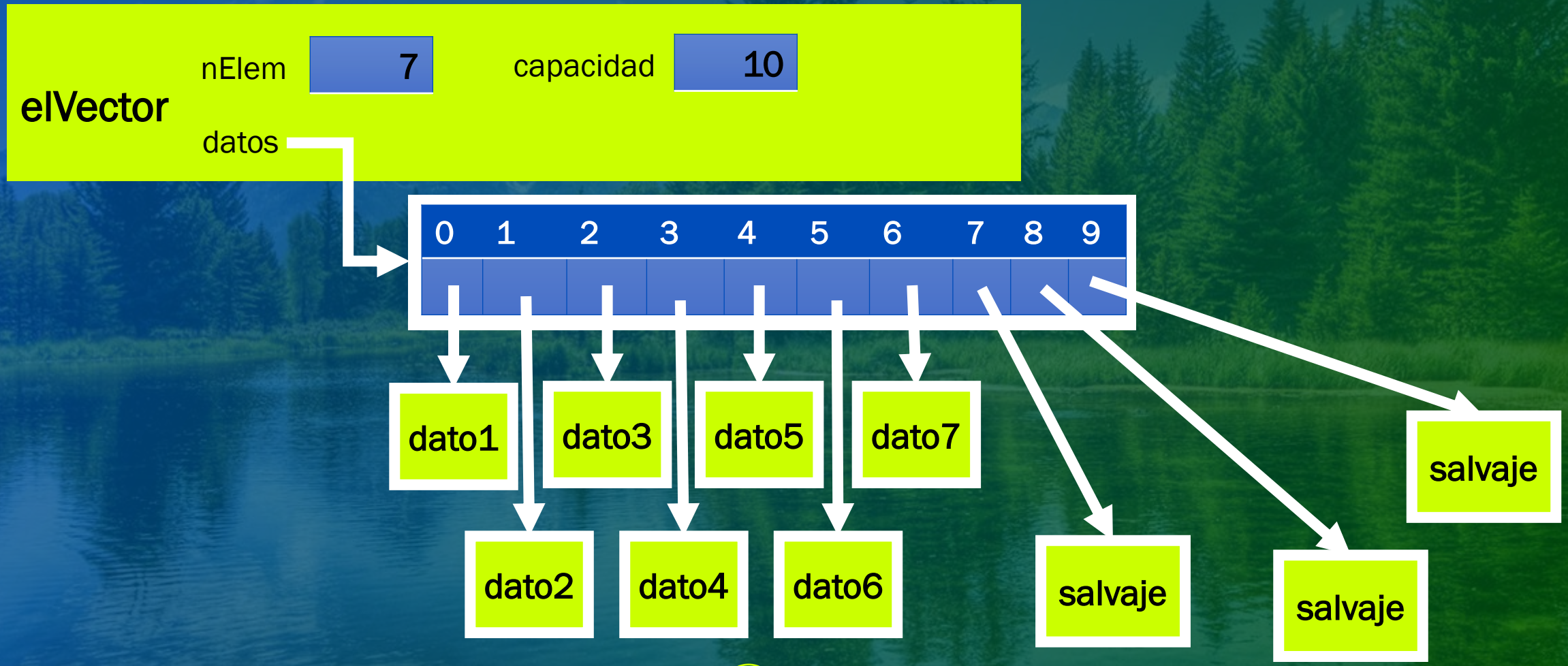
Dinámica: Vector (array dinámico+punteros)

```
typedef struct {
                int nElem;
                int capacidad;
                tPtrDato *datos;
        } tVector;

void inicializaVector( tVector & elVector, int capacidad_INI){
        elVector.nElem = 0;
        elVector.capacidad = capacidad_INI;
        datos = new tPtrDato [elVector.capacidad];
}
```

- Se crea en ejecución y se ordenan sin sobrecoste las estructuras “grandes”.

Dinámica: Vector (array dinámico+punteros)



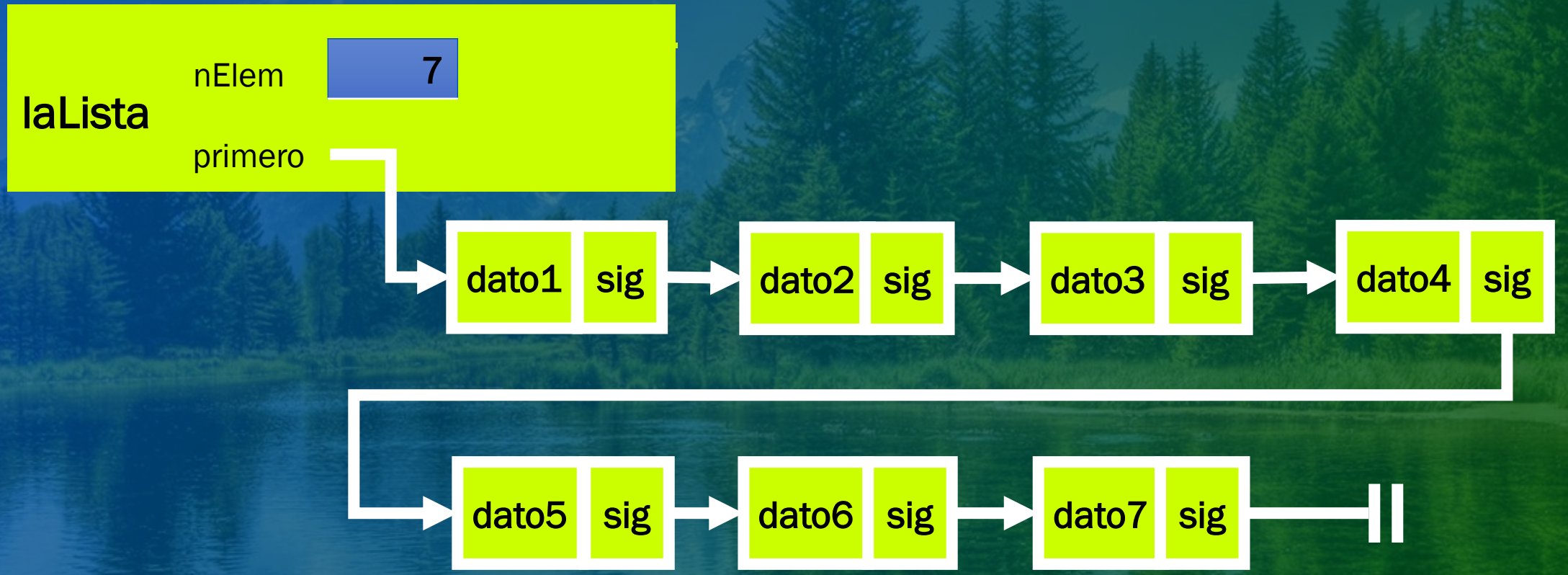
Dinámica: Lista dinámica

```
typedef struct {
                tDato dato;
                tNodoDato *sig;
        } tNodoDato;

typedef struct {
                int nElem;
                tNodoDato *primero;
        } tlListaDatos;
```

- Crece dinámicamente sin los sobrecostes de duplicar la estructura

Dinámica: Lista dinámica



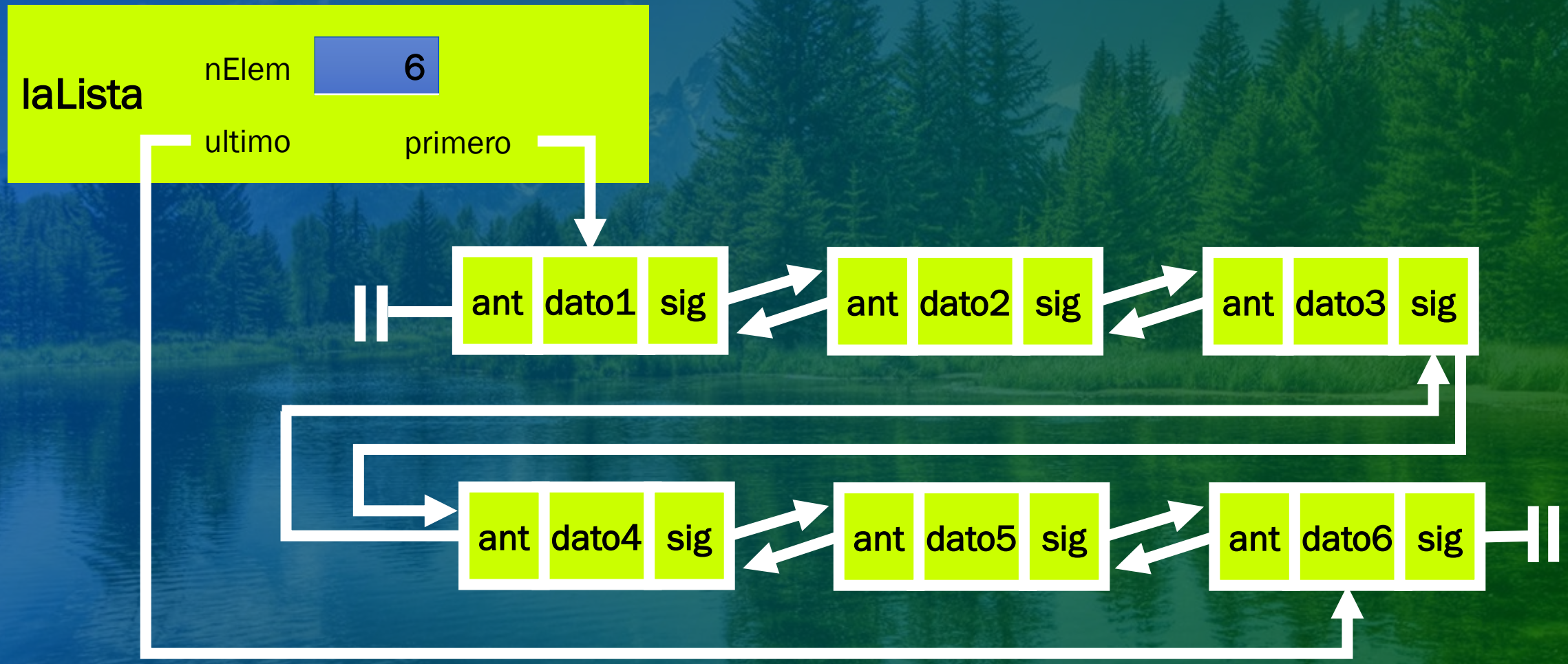
Dinámica: Lista dinámica doble

```
typedef struct {
                tDato dato;
                tNodoDato* sig, ant;
        } tNodoDato;

typedef struct {
                int nElem;
                tNodoDato* primero, ultimo;
        } tlListaDatos;
```

- Crece dinámicamente sin los sobrecostes de duplicar la estructura

Dinámica: Lista dinámica doble



CUIDADO!

- La memoria dinámica hay que borrarla.
- Las variables locales no son memoria dinámica y desaparecen al terminar el bloque.
- El nombre del array es el puntero a su primera posición, para acceder a las posiciones del array (tanto dinámico como estático) se utiliza elArray[3] y no (*elArray)[3]