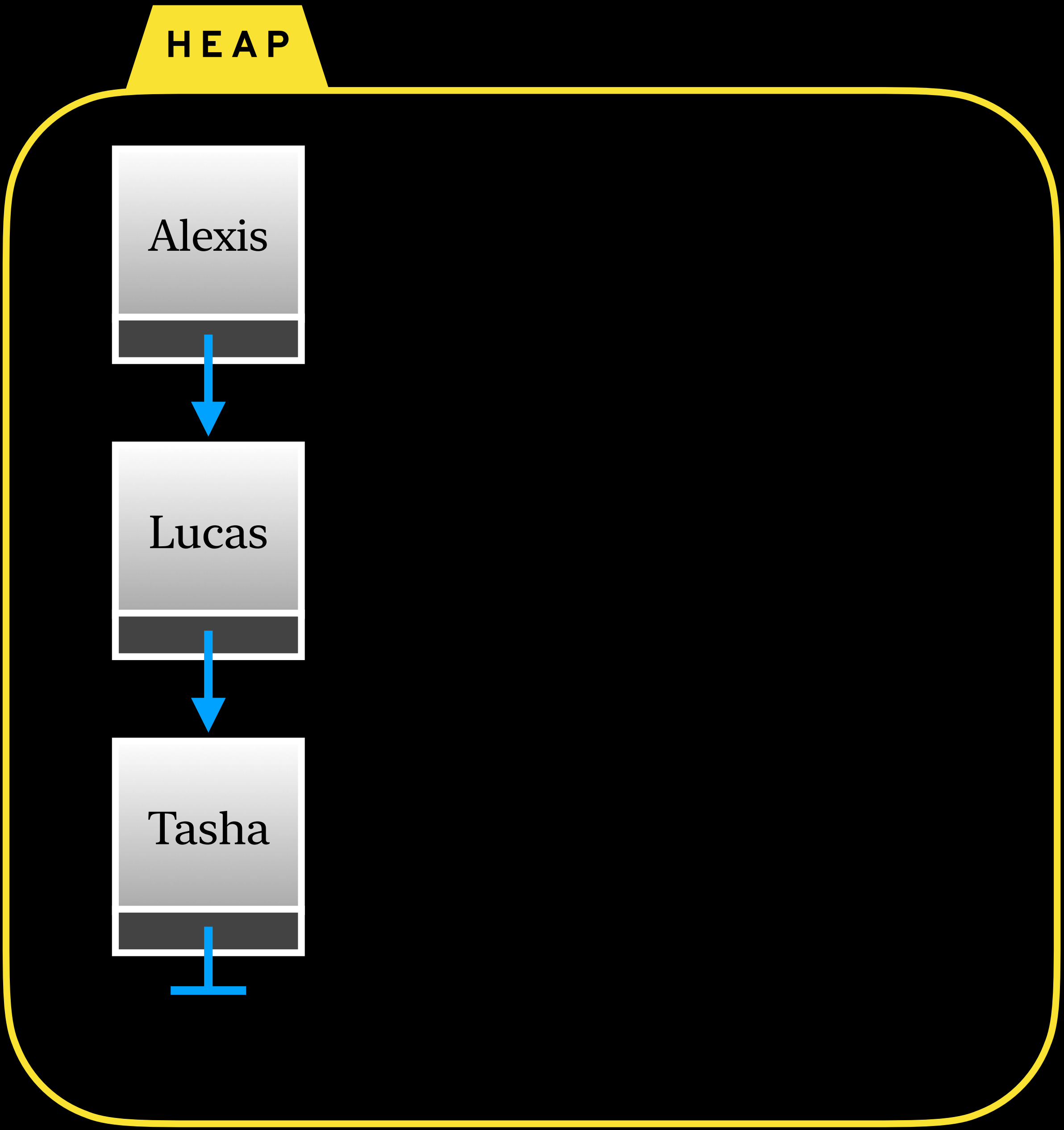
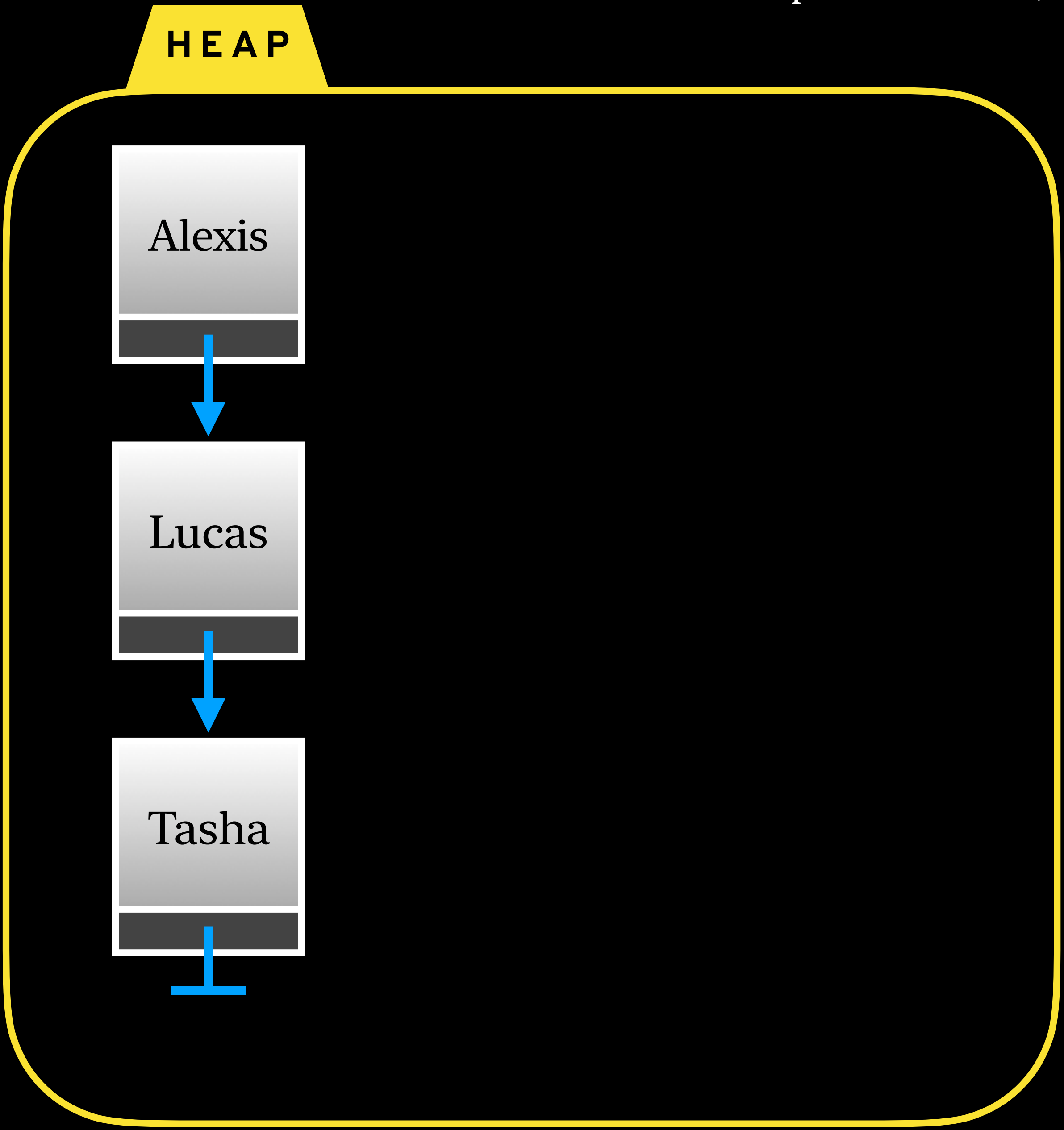
Singly-Linked List visualization



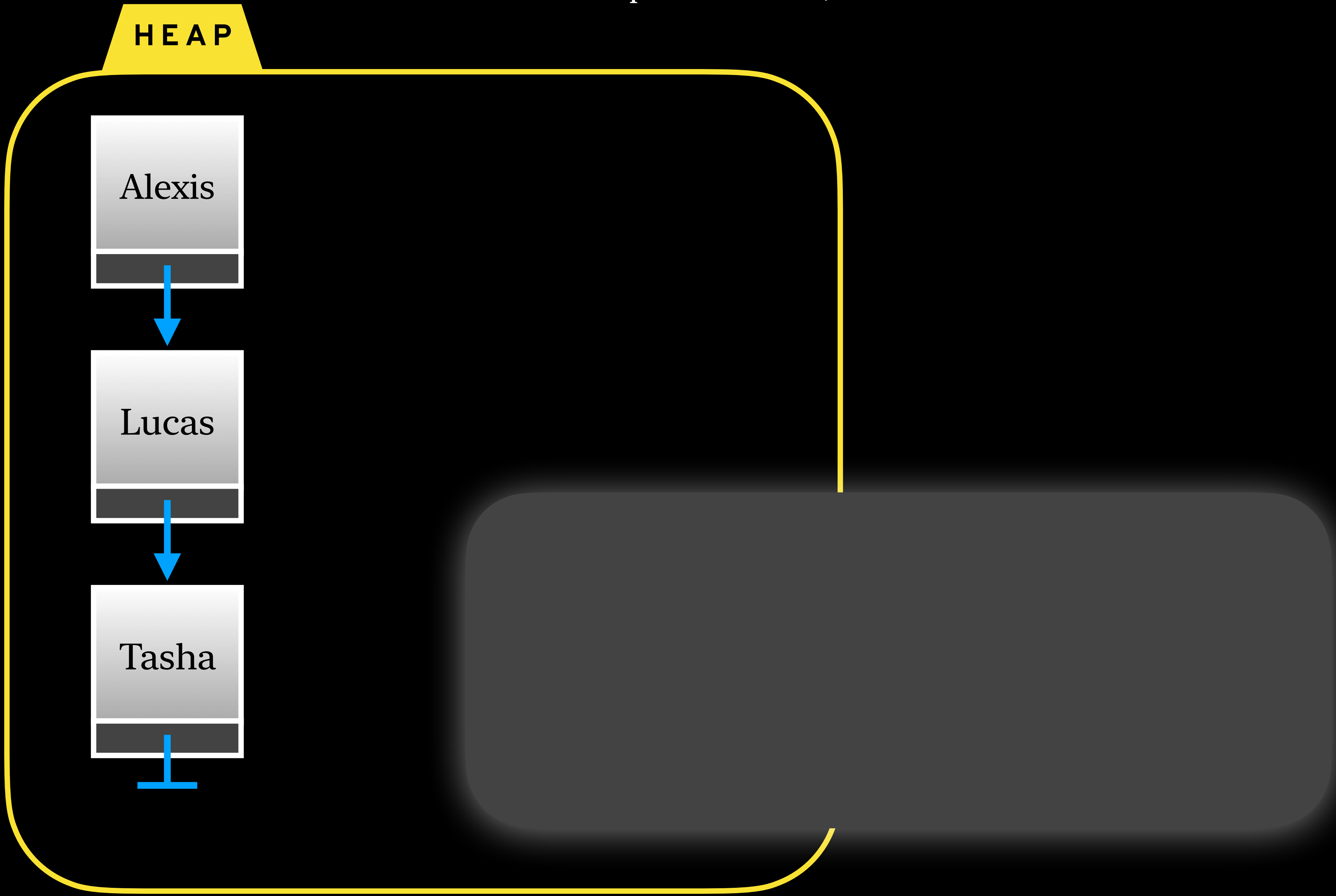
Singly-Linked List visualization

“Add a new person named, ‘Ethan’ and insert into index 1.”



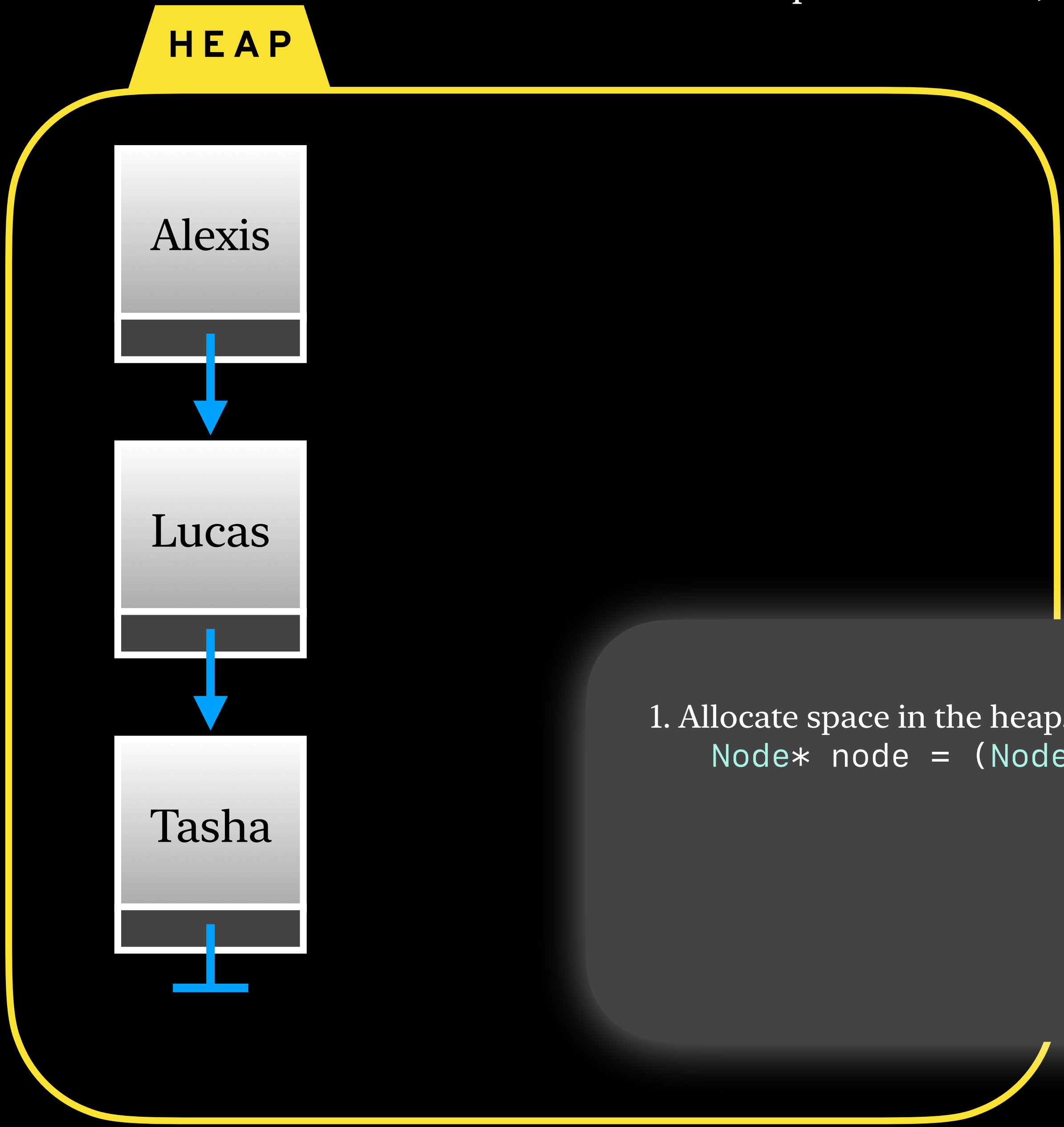
Singly-Linked List visualization

“Add a new person named, ‘Ethan’ and insert into index 1.”



Singly-Linked List visualization

“Add a new person named, ‘Ethan’ and insert into index 1.”

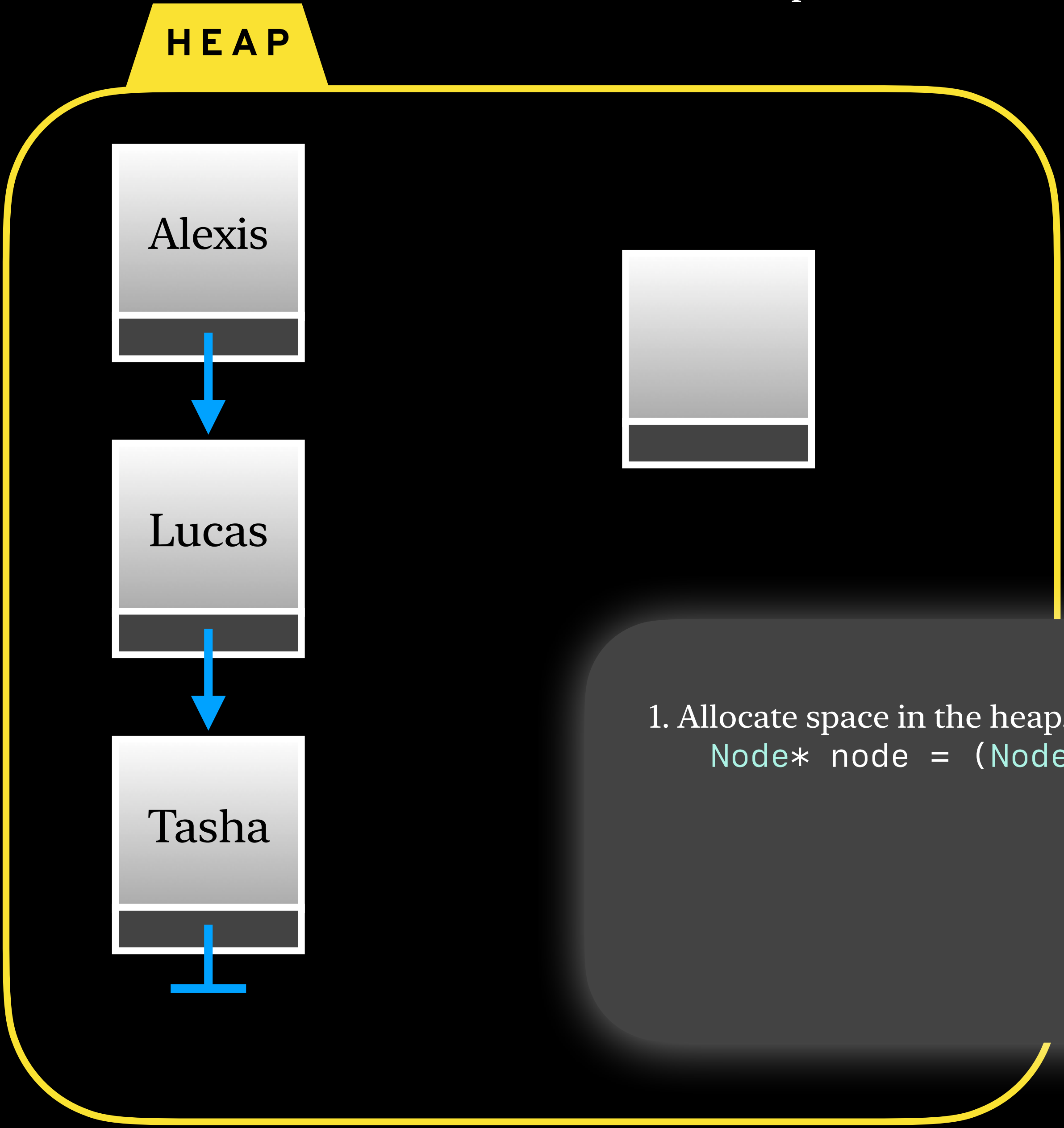


1. Allocate space in the heap.

```
    Node* node = (Node*)malloc(sizeof(Node));
```

Singly-Linked List visualization

“Add a new person named, ‘Ethan’ and insert into index 1.”

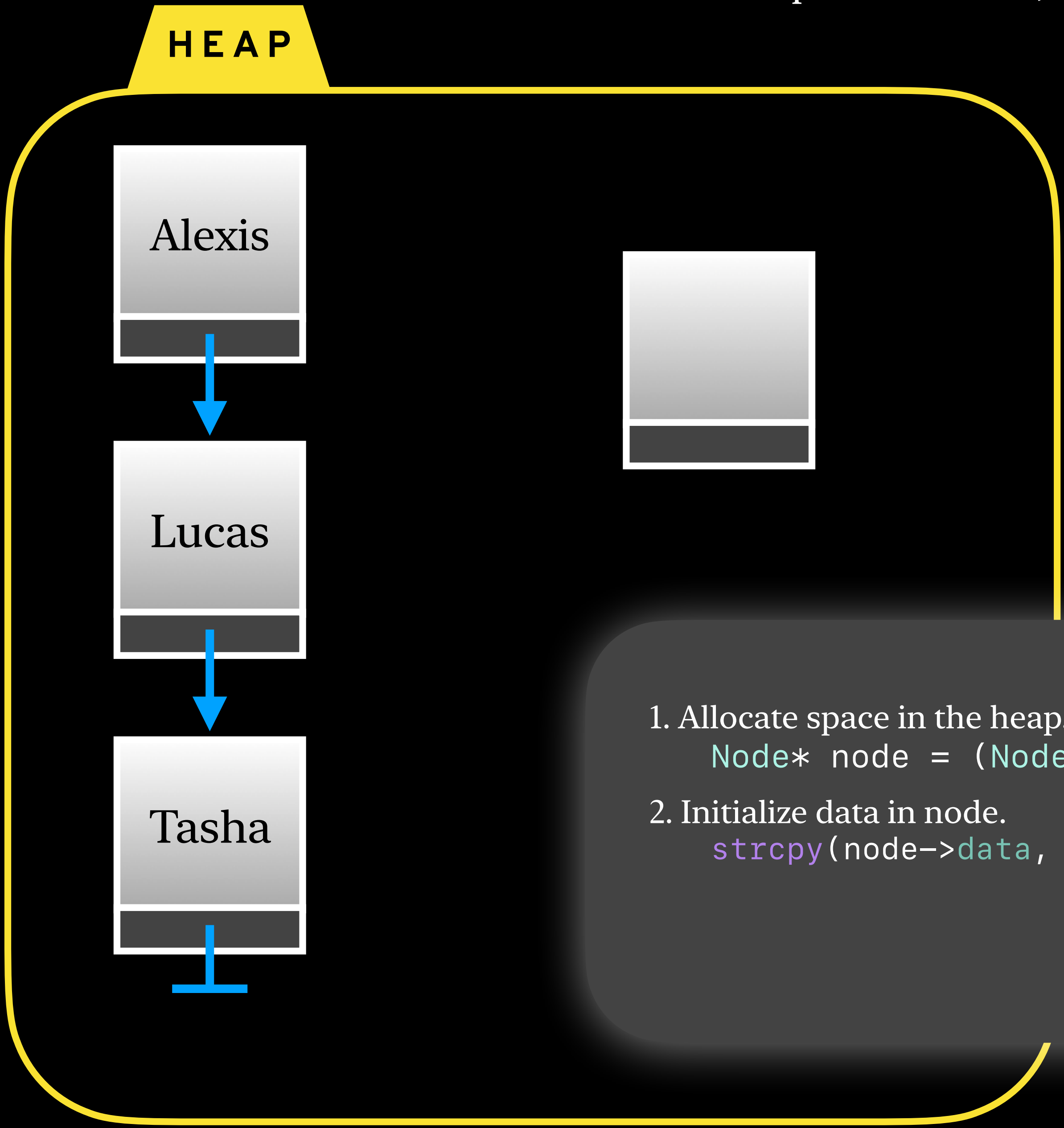


1. Allocate space in the heap.

```
Node* node = (Node*)malloc(sizeof(Node));
```

Singly-Linked List visualization

“Add a new person named, ‘Ethan’ and insert into index 1.”



1. Allocate space in the heap.

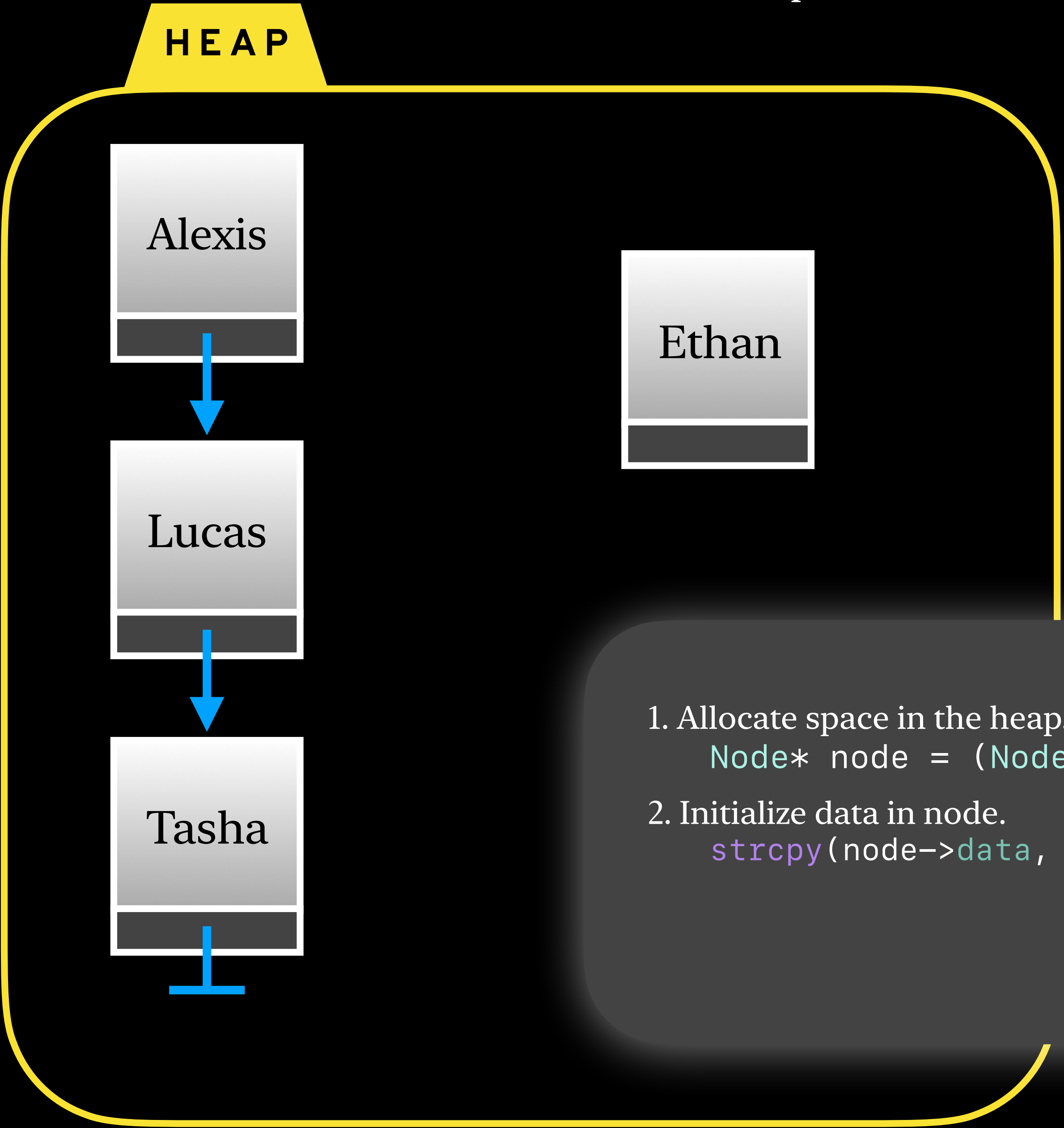
```
Node* node = (Node*)malloc(sizeof(Node));
```

2. Initialize data in node.

```
strcpy(node->data, "Ethan");
```

Singly-Linked List visualization

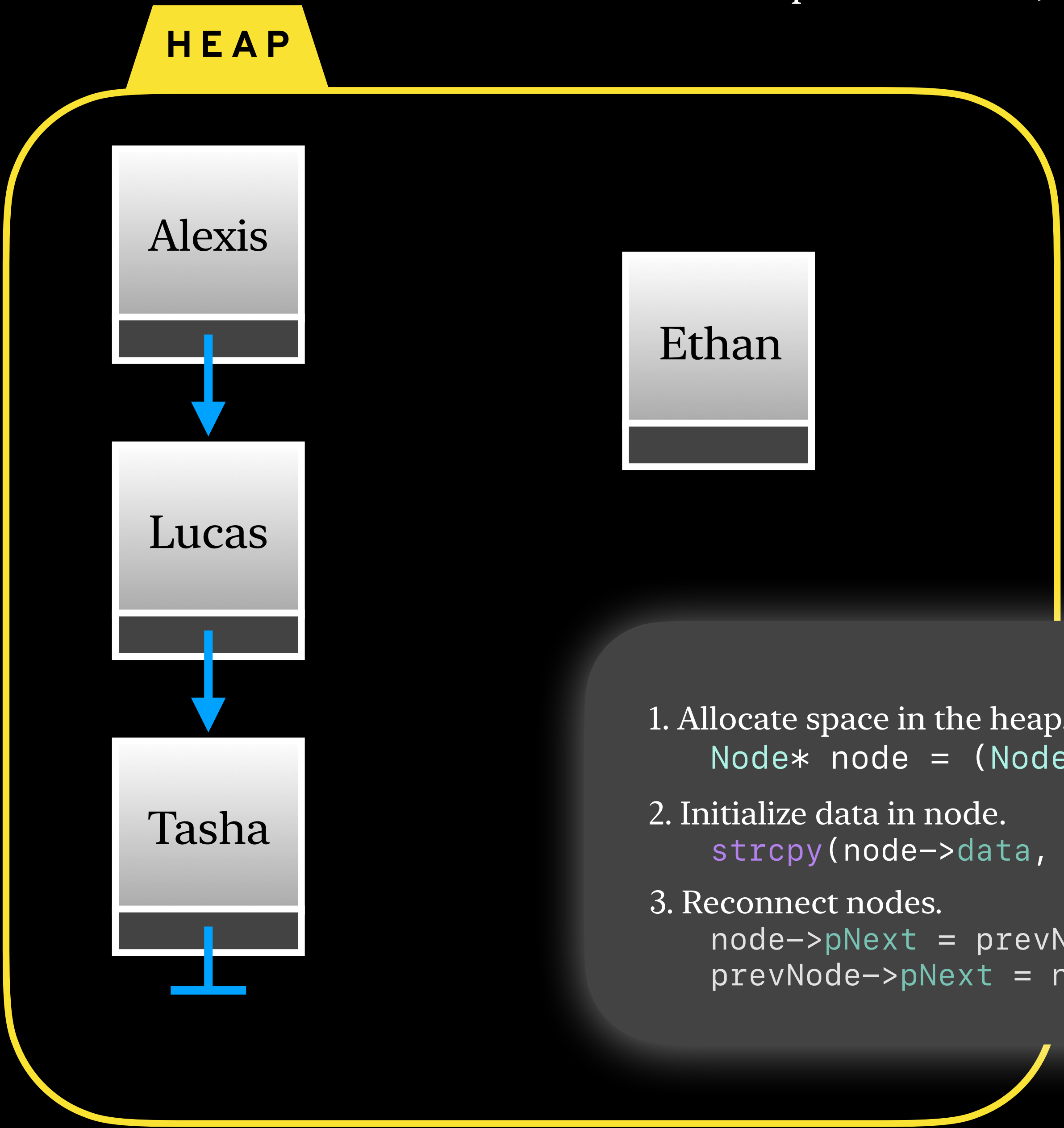
“Add a new person named, ‘Ethan’ and insert into index 1.”



1. Allocate space in the heap.
```
Node* node = (Node*)malloc(sizeof(Node));
```
2. Initialize data in node.
```
strcpy(node->data, "Ethan");
```

Singly-Linked List visualization

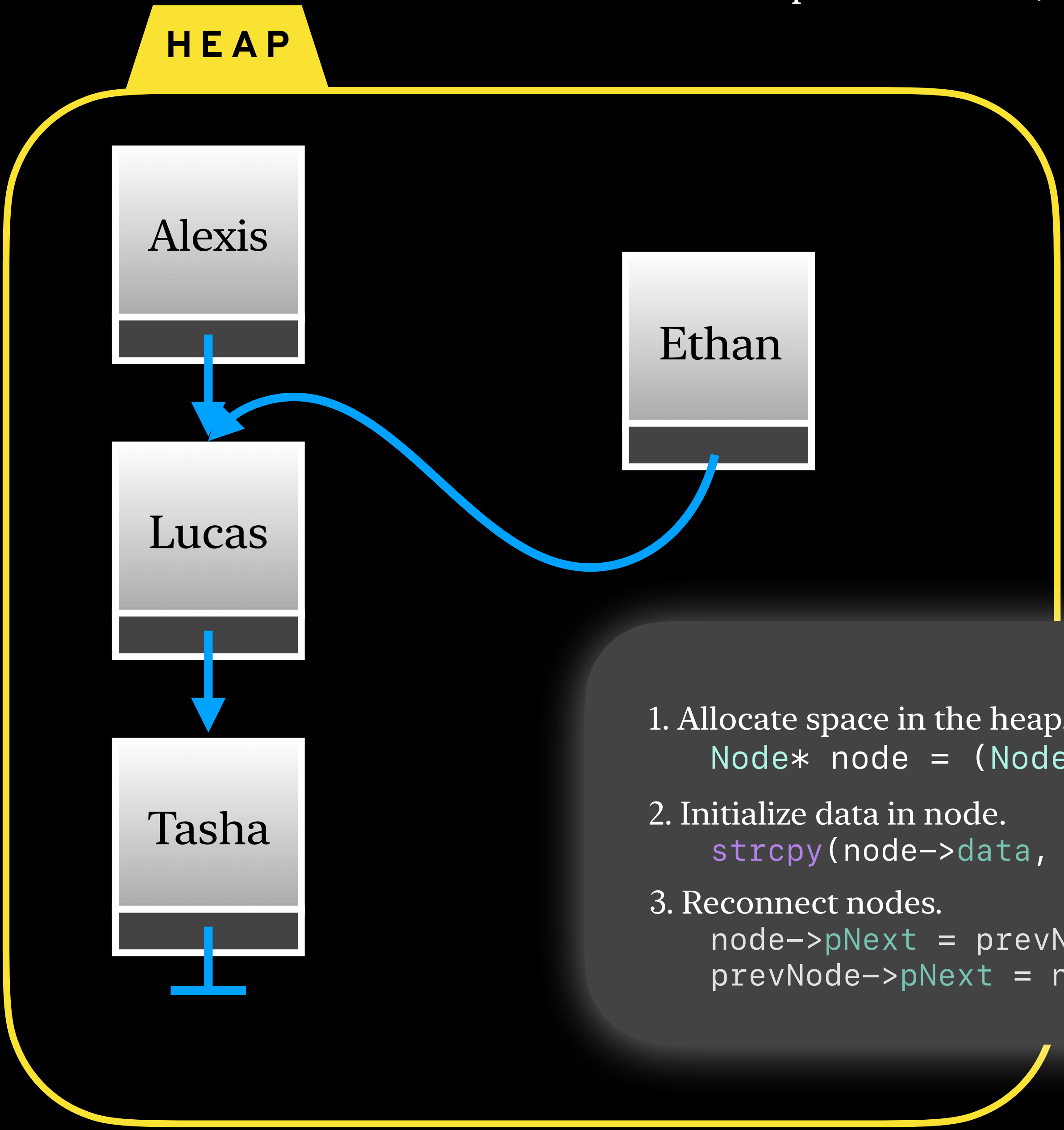
“Add a new person named, ‘Ethan’ and insert into index 1.”



1. Allocate space in the heap.
```
Node* node = (Node*)malloc(sizeof(Node));
```
2. Initialize data in node.
```
strcpy(node->data, "Ethan");
```
3. Reconnect nodes.
```
node->pNext = prevNode->pNext;
prevNode->pNext = node;
```

Singly-Linked List visualization

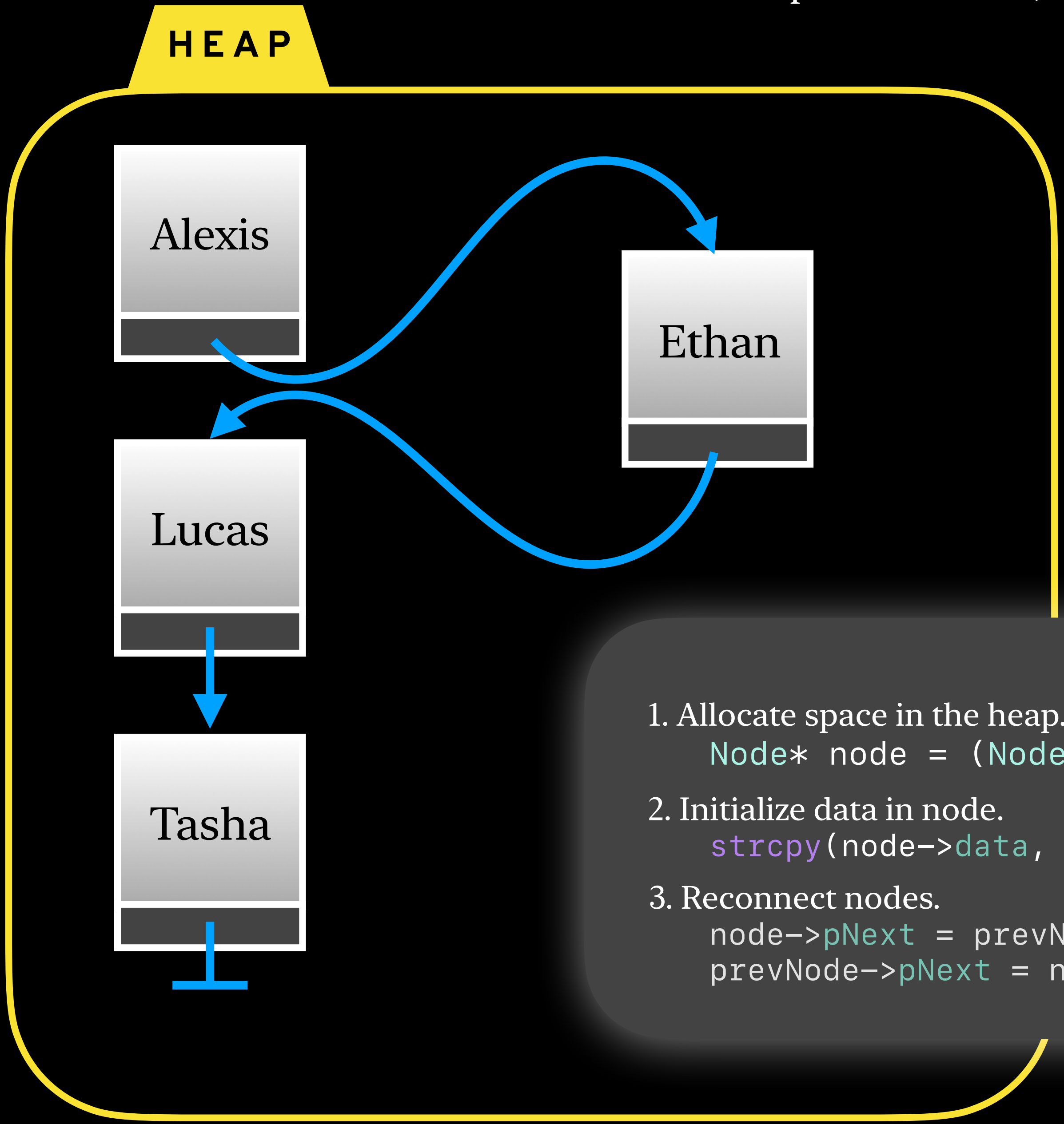
“Add a new person named, ‘Ethan’ and insert into index 1.”



1. Allocate space in the heap.
```
    Node* node = (Node*)malloc(sizeof(Node));
```
2. Initialize data in node.
```
    strcpy(node->data, "Ethan");
```
3. Reconnect nodes.
```
    node->pNext = prevNode->pNext;
    prevNode->pNext = node;
```

Singly-Linked List visualization

“Add a new person named, ‘Ethan’ and insert into index 1.”



1. Allocate space in the heap.

```
    Node* node = (Node*)malloc(sizeof(Node));
```

2. Initialize data in node.

```
    strcpy(node->data, "Ethan");
```

3. Reconnect nodes.

```
    node->pNext = prevNode->pNext;
    prevNode->pNext = node;
```

https://cs.gmu.edu/~kauffman/cs222/stack-demo.html

Stack table is a massive simplification.

When you call a function, the program creates a stack for that function call.

> Responsible for maintaining the local variables and parameters during that call.

```
https://cs.gmu.edu/~kauffman/cs222/stack-demo.html
Stack table is a massive simplification.
```

When you call a function, the program creates a stack for that function call.

> Responsible for maintaining the local variables and parameters during that call.

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}
```

https://cs.gmu.edu/~kauffman/cs222/stack-demo.html

Stack table is a massive simplification.

When you call a function, the program creates a stack for that function call.

> Responsible for maintaining the local variables and parameters during that call.

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}
```

Stack for add_sales_tax

index offset	origin	type	name
0	PARAMETER	double	cost
1	PARAMETER	double	percentage
2	LOCAL	double	salesTax
3	LOCAL	double	total
4	RETURN ADDRESS	pointer	▨

```
https://cs.gmu.edu/~kauffman/cs222/stack-demo.html
Stack table is a massive simplification.
```

Call Stack

Call Stack

```c
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

```
Thread 1 Queue : com.apple.main-thread (serial)
 #0 0x0000000100003f88 in add_sales_tax(double, double) at functions.cpp:14
 #1 0x0000000100003f38 in main at main.cpp:13
 #2 0x00000001000150f4 in start ()
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

```
Thread 1 Queue : com.apple.main-thread (serial)
#0 0x0000000100003f88 in add_sales_tax(double, double) at functions.cpp:14
#1 0x0000000100003f38 in main at main.cpp:13
#2 0x00000001000150f4 in start ()
```

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

```
Thread 1 Queue : com.apple.main-thread (serial)
#0 0x0000000100003f88 in add_sales_tax(double, double) at functions.cpp:14
#1 0x0000000100003f38 in main at main.cpp:13
#2 0x00000001000150f4 in start ()
```

Locals

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

```
Thread 1 Queue : com.apple.main-thread (serial)
#0 0x0000000100003f88 in add_sales_tax(double, double) at functions.cpp:14
#1 0x0000000100003f38 in main at main.cpp:13
#2 0x00
```

index offset	type	name	value
0	double	cost	10.0
1	double	percentage	0.1
2	double	salesTax	1.0
3	double	total	11.0
4	pointer	return	0x0000000100003f38

Locals

Call Stack

```
double add_sales_tax(double cost, double percentage) {
    double salesTax = cost * percentage;
    double total = cost + salesTax;

    return total;
}

int main() {
    printf("%.2f", add_sales_tax(10.00, 0.1));
    return 0;
}
```

```
Thread 1 Queue : com.apple.main-thread (serial)
#0 0x0000000100003f88 in add_sales_tax(double, double) at functions.cpp:14
#1 0x0000000100003f38 in main at main.cpp:13
#2 0x00
```

index offset	type	name	value
0	double	cost	10.0
1	double	percentage	0.1
2	double	salesTax	1.0
3	double	total	11.0
4	pointer	return	0x0000000100003f38

Locals