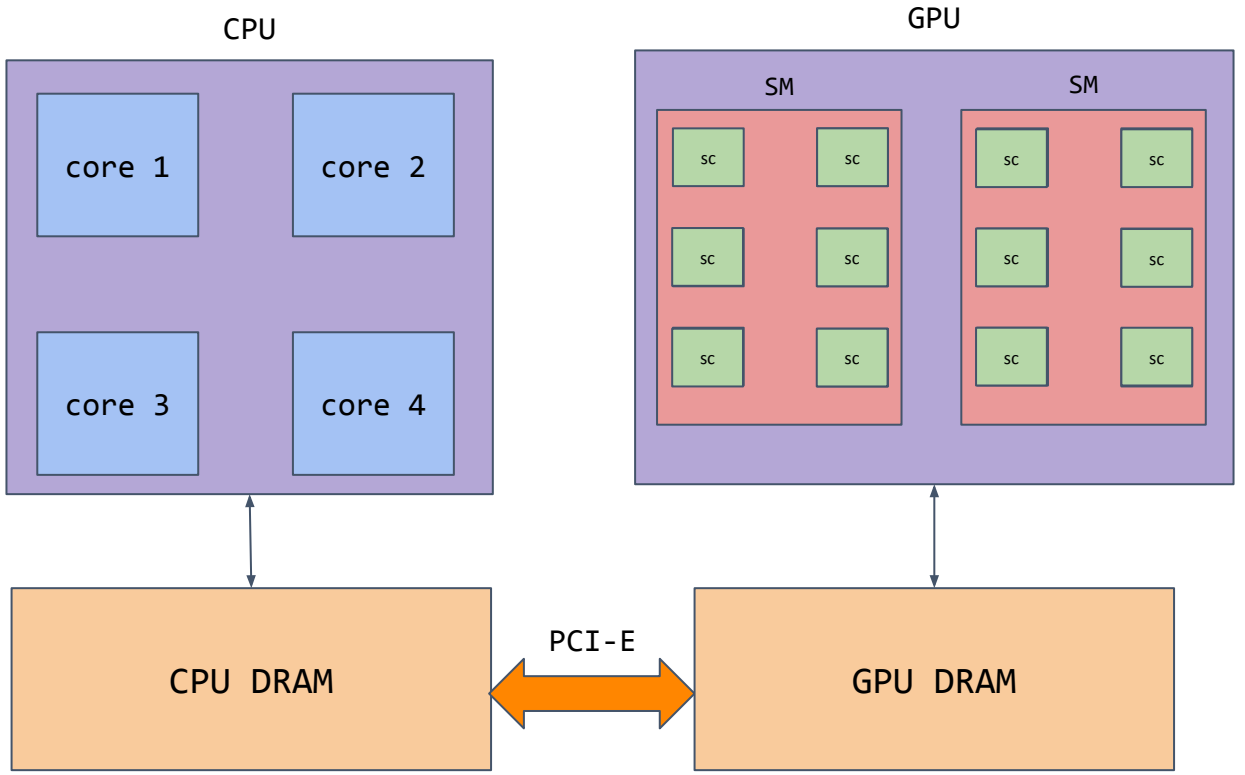
GSOHC:Global Synchronization Optimization for Heterogeneous Computing

Soumik Kumar Basu Jyothi Vedurada

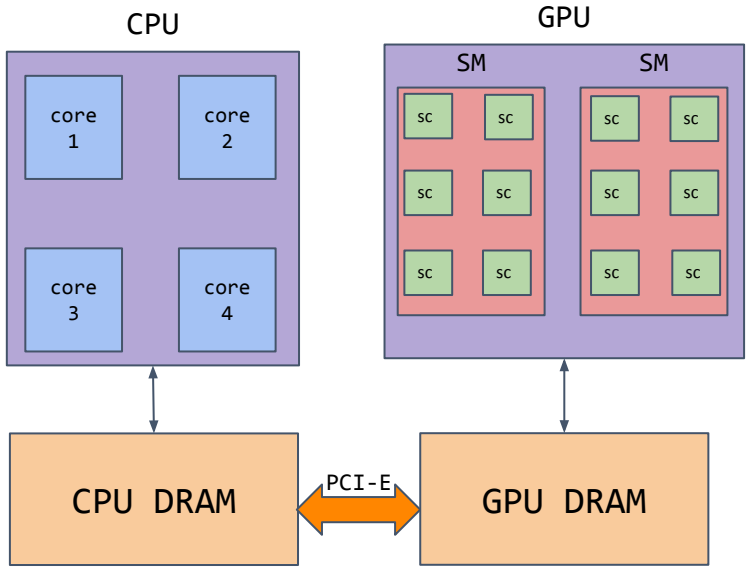
Department of Computer Science and Engineering

IIT Hyderabad

Execution of a heterogeneous program



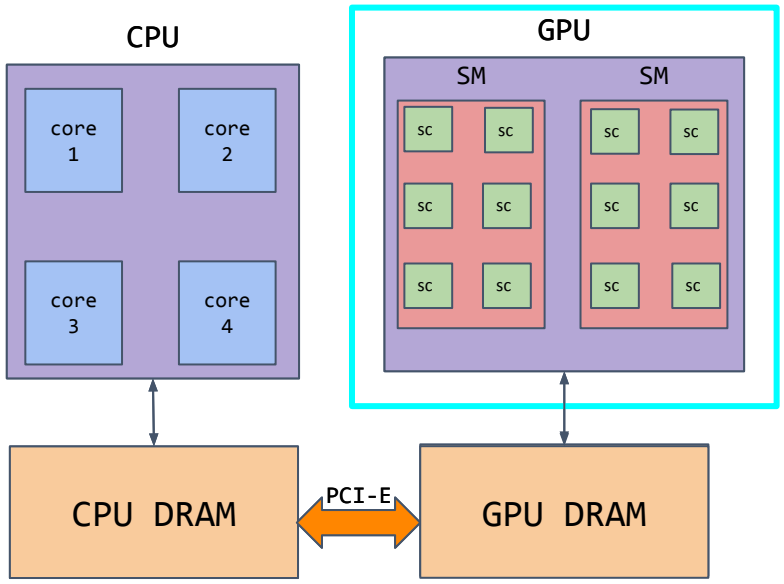
A typical execution of CUDA programs




```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

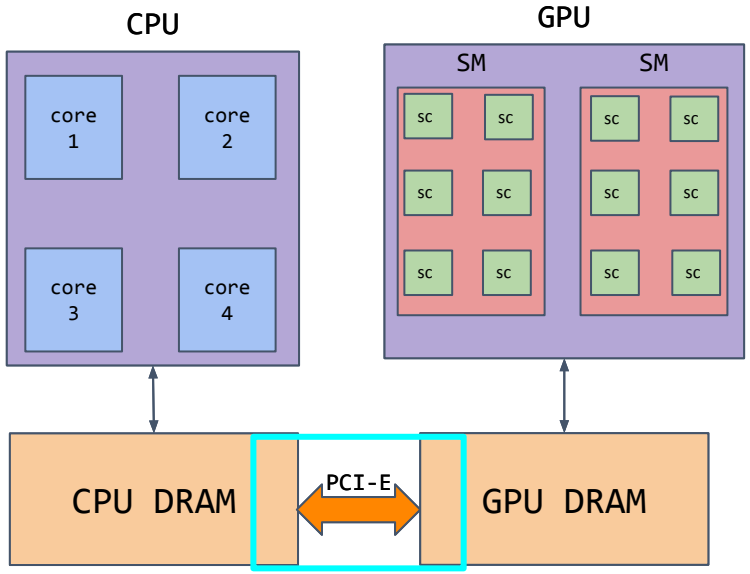
A typical execution of CUDA programs




```
1.  void main(){
2.    ...
3.    kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.    cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.    cudaDeviceSynchronize();/*Sync*/
8.    /*Host-side computations*/
9.    ...
10.   foo(v, size2);
11.   ...
12.   for(int i=0; i<size2; i++)
13.     v[i] = v[i] + x_points;
14.     ...
15. }
```

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

A typical execution of CUDA programs




```
1.  void main(){
2.    ...
3.    kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.    cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.    cudaDeviceSynchronize();/*Sync*/
8.    /*Host-side computations*/
9.    ...
10.   foo(v, size2);
11.   ...
12.   for(int i=0; i<size2; i++)
13.     v[i] = v[i] + x_points;
14.     ...
15. }
```

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

A brief introduction of CUDA programs

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

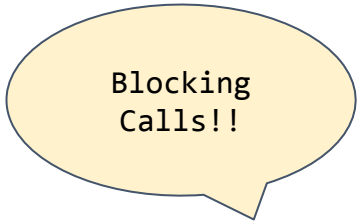
A brief introduction of CUDA programs

```
1.  void main(){
2.    ...
3.    kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.    cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.    cudaDeviceSynchronize();/*Sync*/
8.    /*Host-side computations*/
9.    ...
10.   foo(v, size2);
11.   ...
12.   for(int i=0; i<size2; i++)
13.     v[i] = v[i] + x_points;
14.     ...
15. }
```

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

A brief introduction of CUDA programs

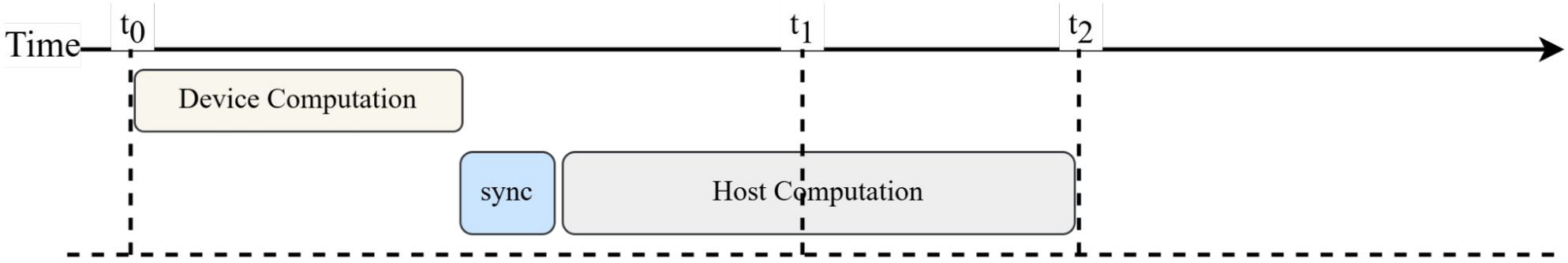



```
1.  void main(){
2.    ...
3.    kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.    cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.    cudaDeviceSynchronize();/*Sync*/
8.    /*Host-side computations*/
9.    ...
10.   foo(v, size2);
11.   ...
12.   for(int i=0; i<size2; i++)
13.     v[i] = v[i] + x_points;
14.     ...
15. }
```

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

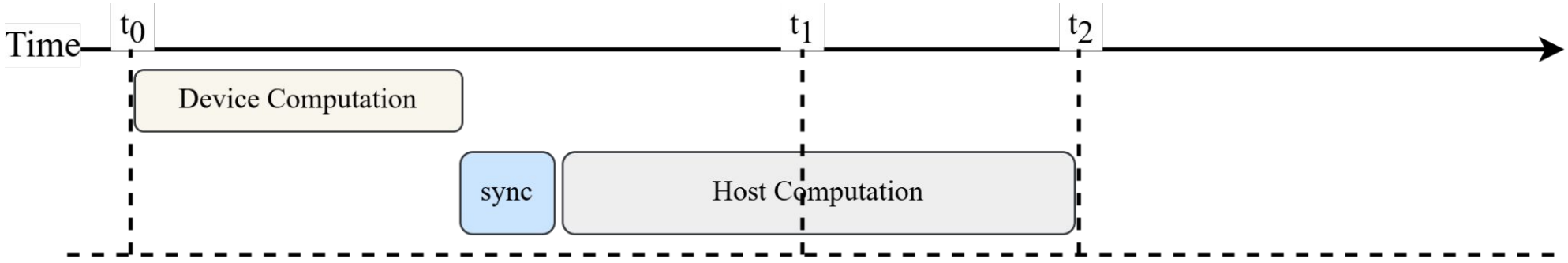
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Problem: Poor placement of synchronization statements



Total time taken = $t_2 - t_0$

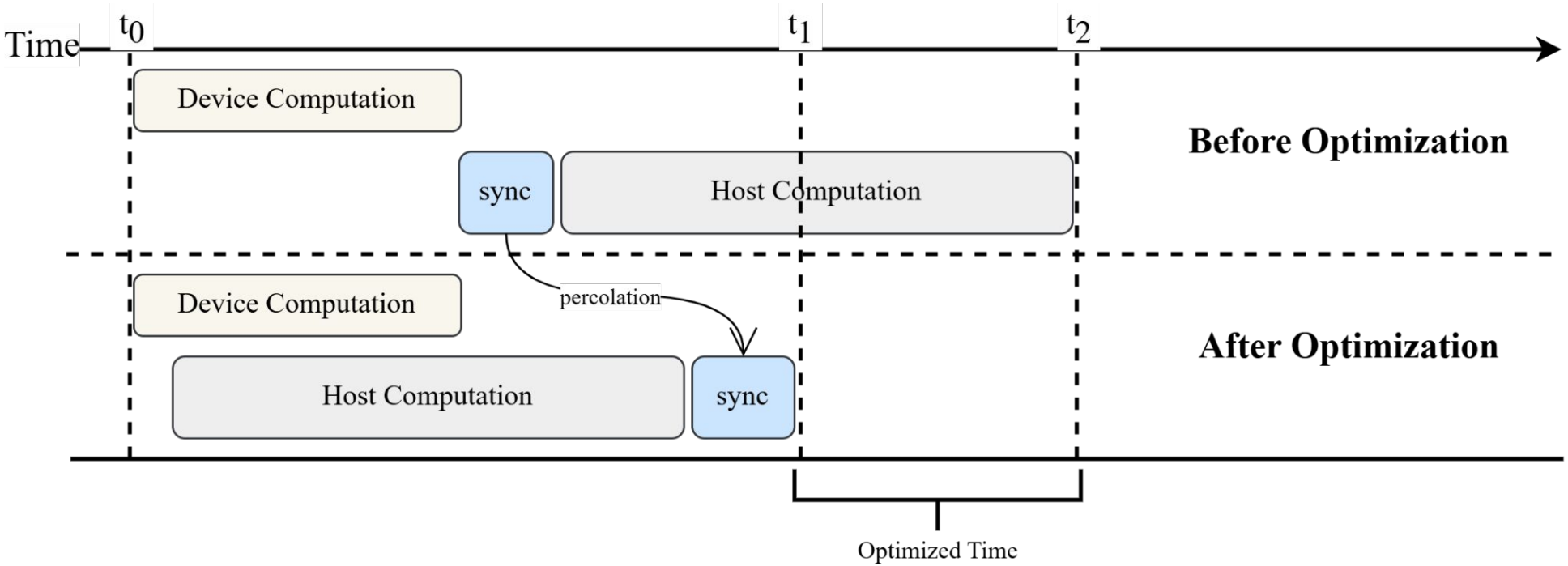
Problem: Poor placement of synchronization statements



Total time taken = $t_2 - t_0$

How can we improve such a program ?

Solution: Hetero-sync motion



Total time taken = $t_1 - t_0$

Hetero-sync motion opportunity: Graph Coloring (Case Study)

```
1.  void divideWork(...){
2.    if(value < size) {
3.      for(int i = 0; i < value; ++i) {...}
4.    }
5.    else {
6.      int portion = value / size;
7.      for(int i = 0; i < size; ++i) {
8.        h_head[i] = i * portion;
9.        h_tail[i] = (i + 1) * portion;
10.     }
11.       h_tail[size - 1] = value;
12.   }
13.   cudaMemcpy(d_head, h_head, ...,
      cudaMemcpyHostToDevice);
14.   cudaMemcpy(d_tail, h_tail, ...,
      cudaMemcpyHostToDevice);
15. }
```

```
1.  void main(){...
2.   while(!(*h_complete)) {
3.    divideWork(h_head, h_tail, d_head, d_tail, NBLOCKS, V);
4.
5.    assignColorsKernel<<<...>>> (d_head, ...);
6.
7.    cudaDeviceSynchronize();
8.    *h_complete = true;
9.    cudaMemcpy(d_complete, ..., cudaMemcpyHostToDevice);
10.
11.   divideWork(...);
12.   detectConflictsKernel<<<...>>> (..., d_complete, ...);
13.
14.   cudaDeviceSynchronize();
15.   cudaMemcpy(..., d_complete, ..., cudaMemcpyDeviceToHost);
16.
17.   divideWork(...);
18.   forbidColorsKernel<<<...>>> (d_head, ...);
19.
20.   cudaDeviceSynchronize();}
21.  ...
22. }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Hetero-sync motion opportunity: Graph Coloring (Case Study)

```
1.  void divideWork(...){
2.    if(value < size) {
3.      for(int i = 0; i < value; ++i) {...}
4.    }
5.    else {
6.      int portion = value / size;
7.      for(int i = 0; i < size; ++i) {
8.        h_head[i] = i * portion;
9.        h_tail[i] = (i + 1) * portion;
10.     }
11.       h_tail[size - 1] = value;
12.   }
13.     cudaMemcpy(d_head, h_head, ...,
      cudaMemcpyHostToDevice);
14.     cudaMemcpy(d_tail, h_tail, ...,
      cudaMemcpyHostToDevice);
15. }
```

```
1.  void main(){...
2.   while(!(*h_complete)) {
3.    divideWork(h_head, h_tail, d_head, d_tail, NBLOCKS, V);
4.
5.    assignColorsKernel<<<...>>> (d_head, ...);
6.
7.    cudaDeviceSynchronize();
8.    *h_complete = true;
9.    cudaMemcpy(d_complete, ..., cudaMemcpyHostToDevice);
10.
11.   divideWork(...);
12.   detectConflictsKernel<<<...>>> (..., d_complete, ...);
13.
14.   cudaDeviceSynchronize();
15.   cudaMemcpy(..., d_complete, ..., cudaMemcpyDeviceToHost);
16.
17.   divideWork(...);
18.   forbidColorsKernel<<<...>>> (d_head, ...);
19.
20.   cudaDeviceSynchronize();}
21.  ...
22. }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

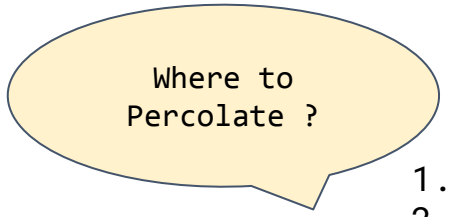
Hetero-sync motion opportunity: Graph Coloring (Case Study)

```
1.  void divideWork(...){
2.    if(value < size) {
3.      for(int i = 0; i < value; ++i) {...}
4.    }
5.    else {
6.      int portion = value / size;
7.      for(int i = 0; i < size; ++i) {
8.        h_head[i] = i * portion;
9.        h_tail[i] = (i + 1) * portion;
10.     }
11.       h_tail[size - 1] = value;
12.   }
13.   cudaMemcpy(d_head, h_head, ...,
      cudaMemcpyHostToDevice);
14.   cudaMemcpy(d_tail, h_tail, ...,
      cudaMemcpyHostToDevice);
15. }
```

```
1.  void main(){...
2.   while(!(*h_complete)) {
3.    divideWork(h_head, h_tail, d_head, d_tail, NBLOCKS, V);
4.
5.    assignColorsKernel<<<...>>> (d_head, ...);
6.
7.    cudaDeviceSynchronize();
8.    *h_complete = true;
9.    cudaMemcpy(d_complete, ..., cudaMemcpyHostToDevice);
10.
11.   divideWork(...);
12.   detectConflictsKernel<<<...>>> (..., d_complete, ...);
13.
14.   cudaDeviceSynchronize();
15.   cudaMemcpy(..., d_complete, ..., cudaMemcpyDeviceToHost);
16.
17.   divideWork(...);
18.   forbidColorsKernel<<<...>>> (d_head, ...);
19.
20.   cudaDeviceSynchronize();}
21.  ...
22. }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Challenges of manually applying hetero-sync motion




```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Challenges

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

Challenge 1: Need to check for the data dependence while percolation to maintain program correctness

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

Challenges

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

Challenge 1: Need to check for the data dependence while percolation to maintain program correctness

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Challenges

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

Challenge 2: Simply checking for data dependency may result in incorrectly percolation

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Challenges

```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

Challenge 3: While percolation, need to consider the farthest target location

```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Challenges

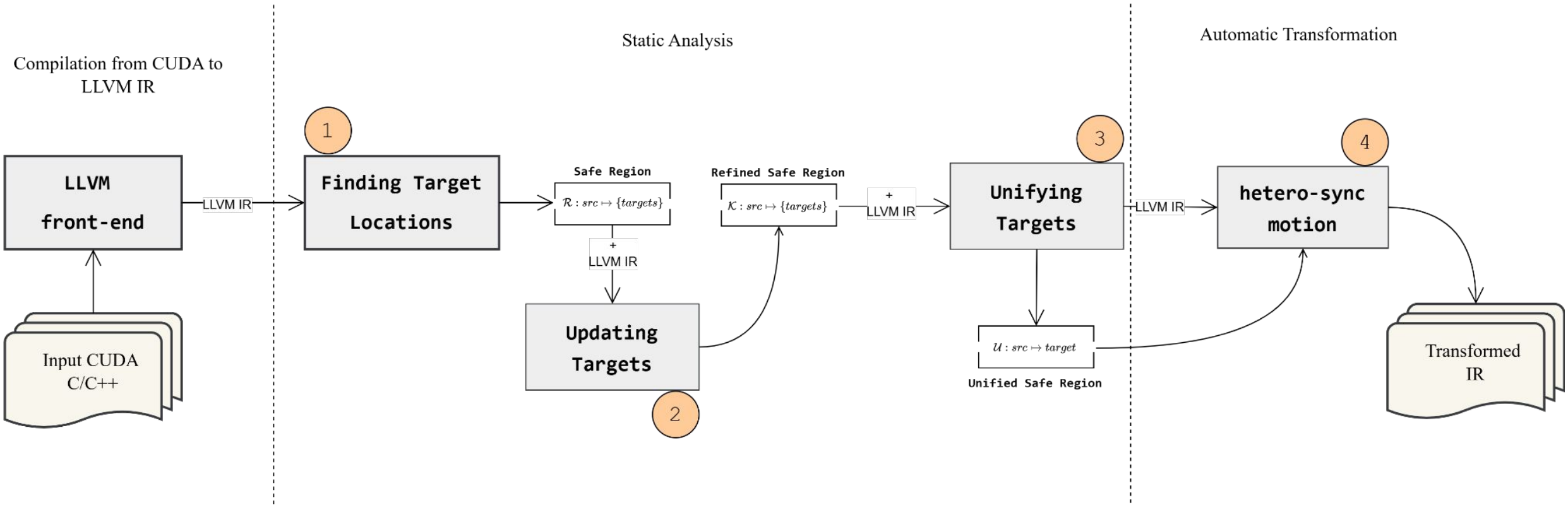
```
16.  void foo(int *x, int size2)
17.  {
18.    ...
19.    if(k < y_points)
20.      x[k] = x[0] + y_points;
21.    ...
22.  }
```

Challenge 4: A program may have multiple memcpy statements

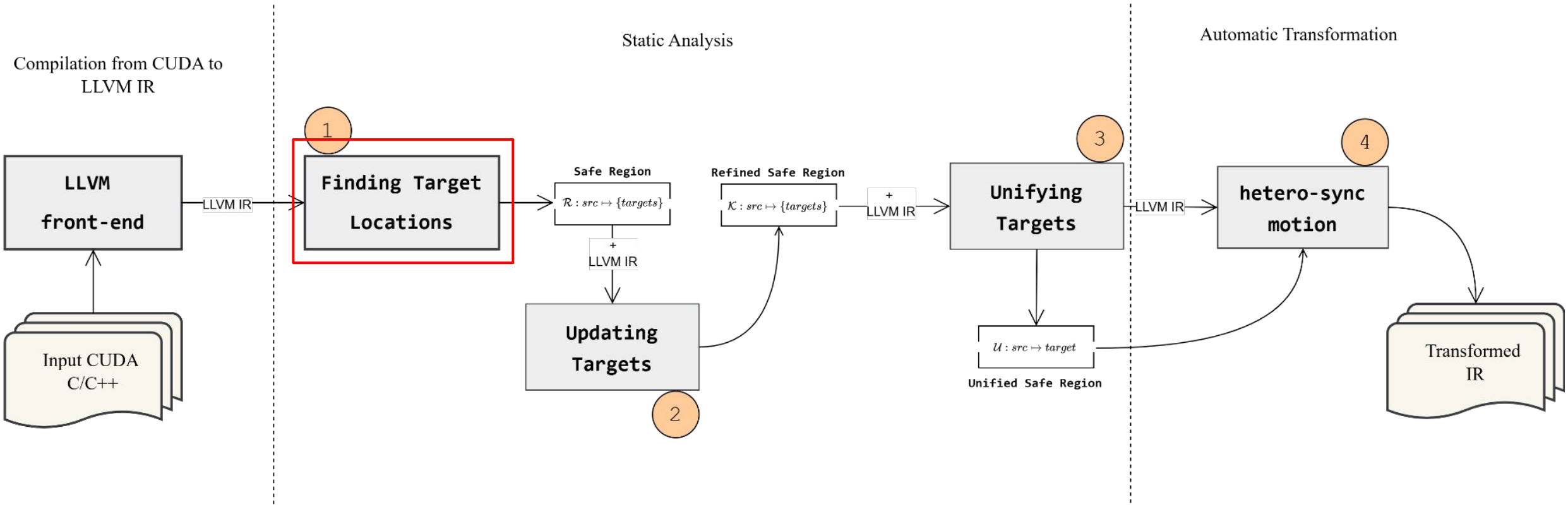
```
1.   void main(){
2.     ...
3.     kernel2<<<grid2, block2>>>(d_u,...);/*Device-side computations*/
4.
5.     cudaMemcpy(v, d_u, size2, cudaMemcpyDeviceToHost);
6.
7.     cudaDeviceSynchronize();/*Sync*/
8.     /*Host-side computations*/
9.     ...
10.    foo(v, size2);
11.    ...
12.    for(int i=0; i<size2; i++)
13.      v[i] = v[i] + x_points;
14.      ...
15.  }
```

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Solution: GSOHC

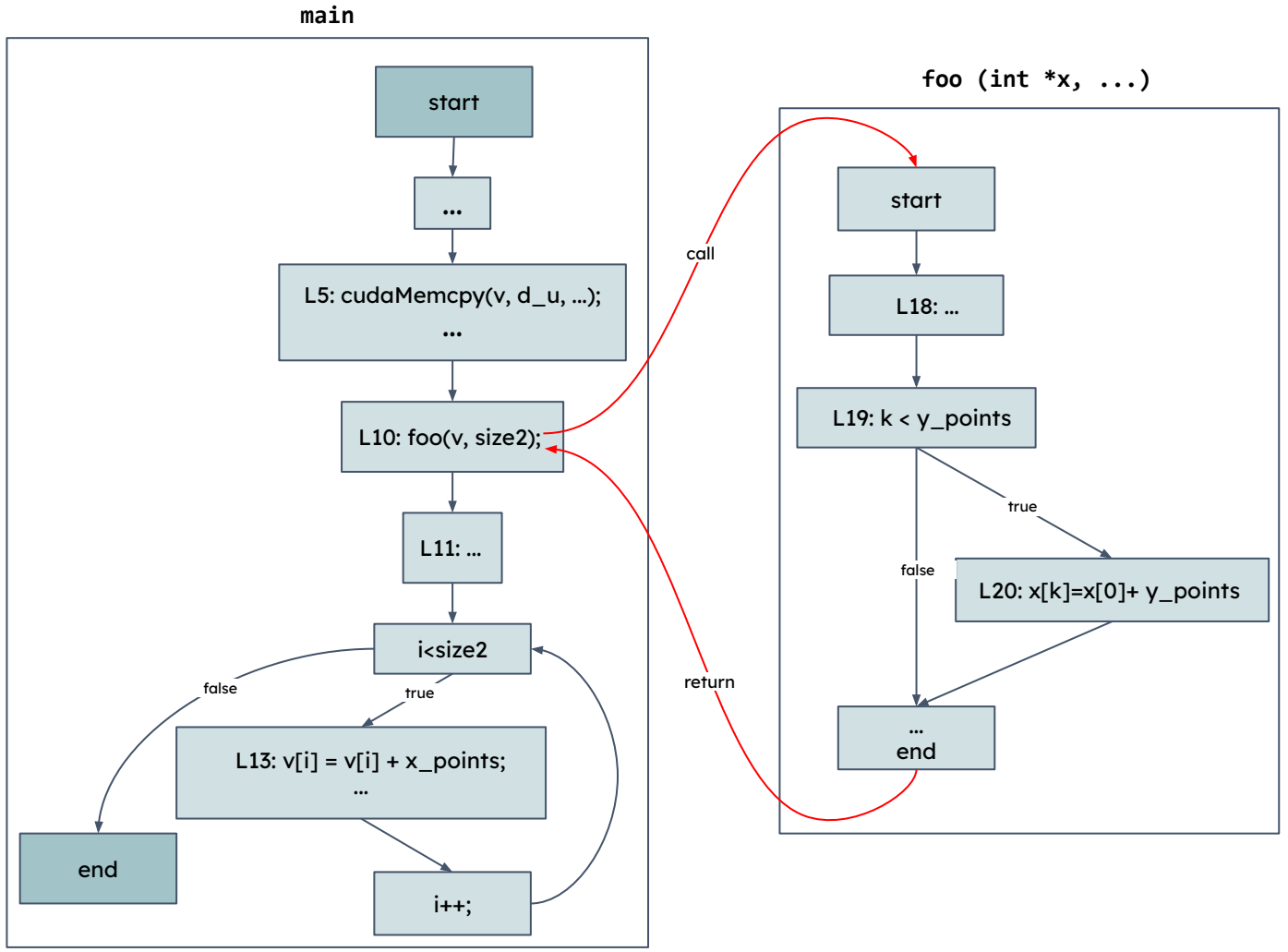


Step 1: Finding Target Locations

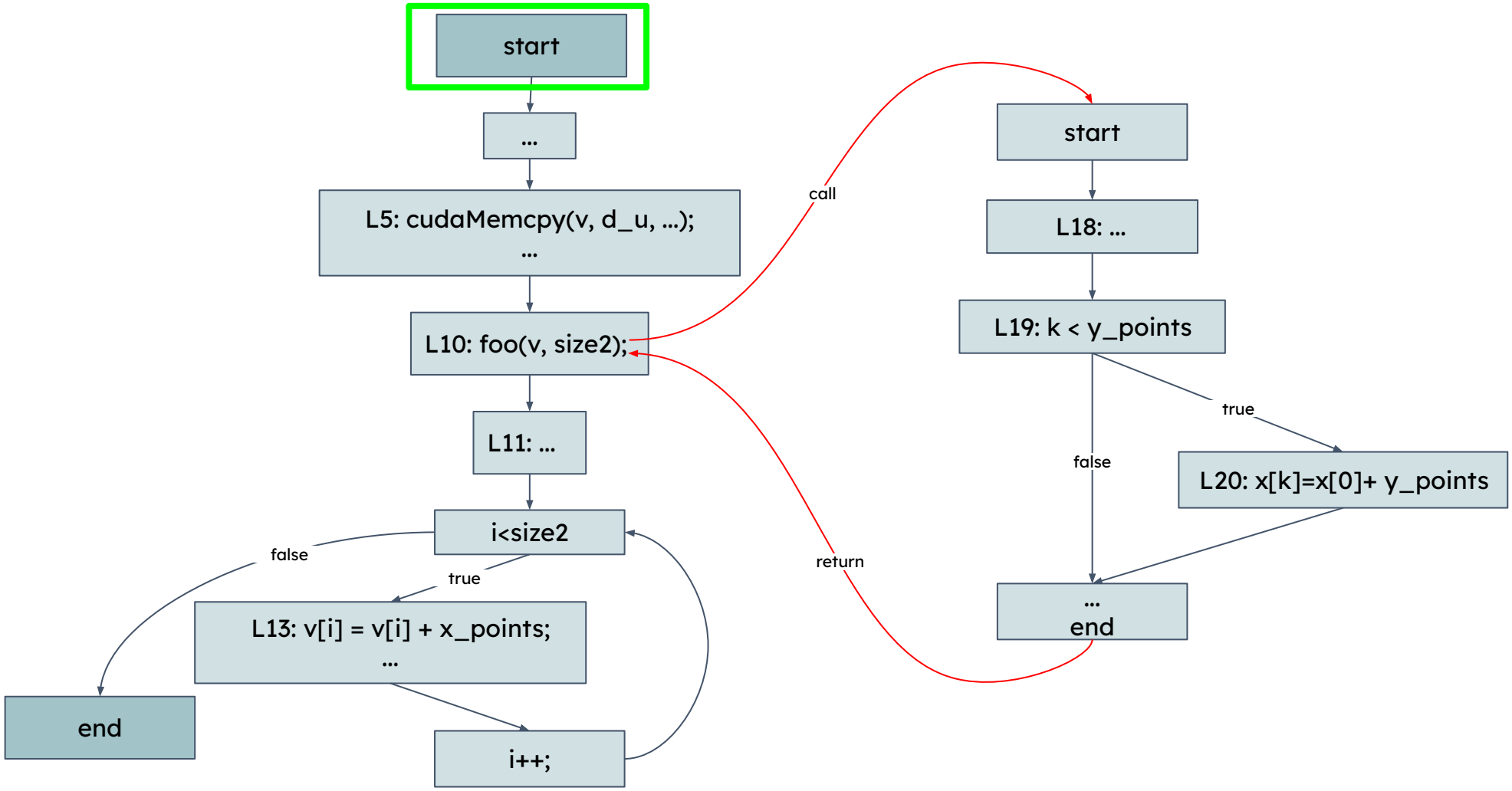


Step 1: Finding Target Locations

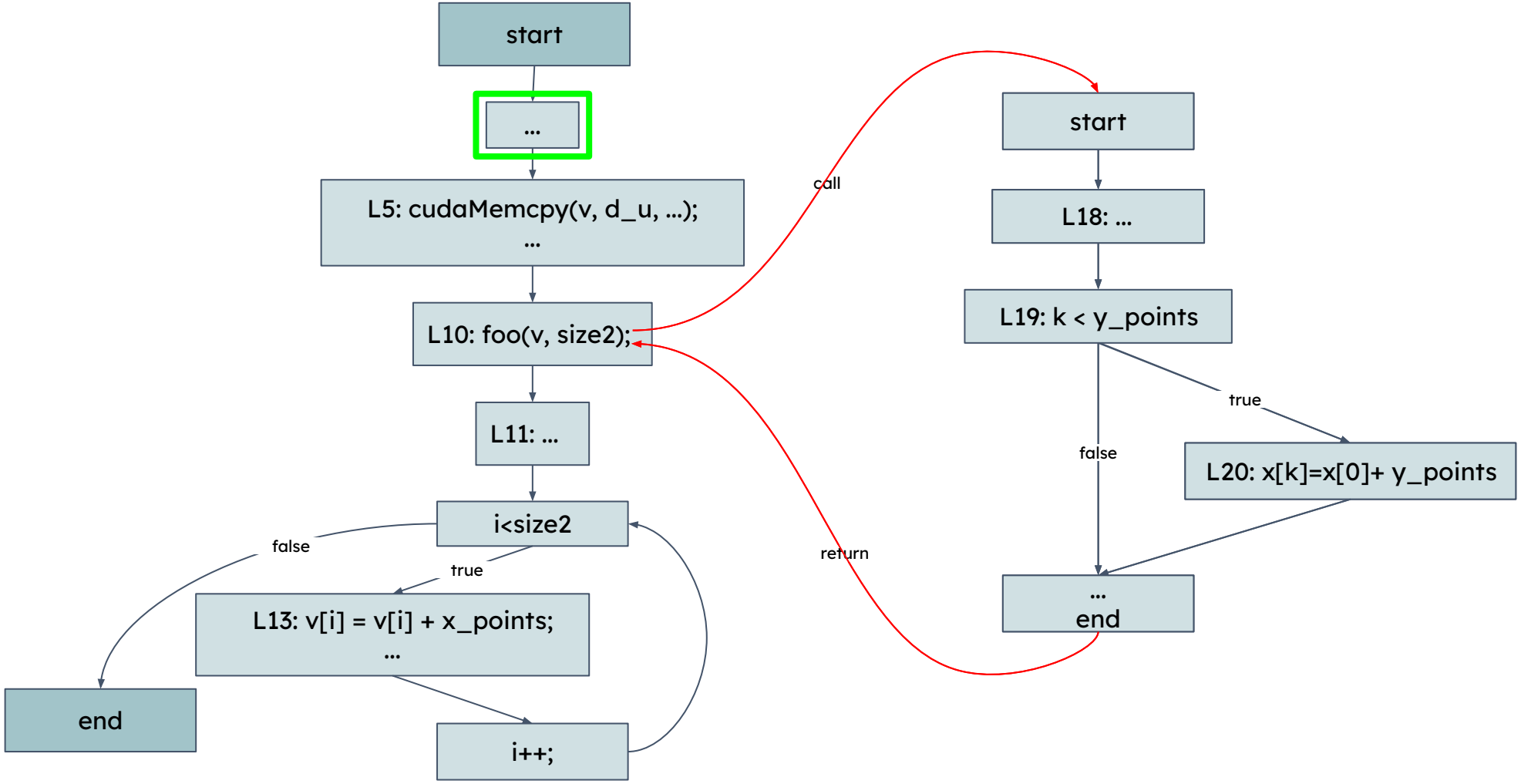
- Inter-procedural flow-sensitive, context-sensitive data-flow analysis
- Identifies target locations based on the data-dependence
- Handles multiple blocking calls at once



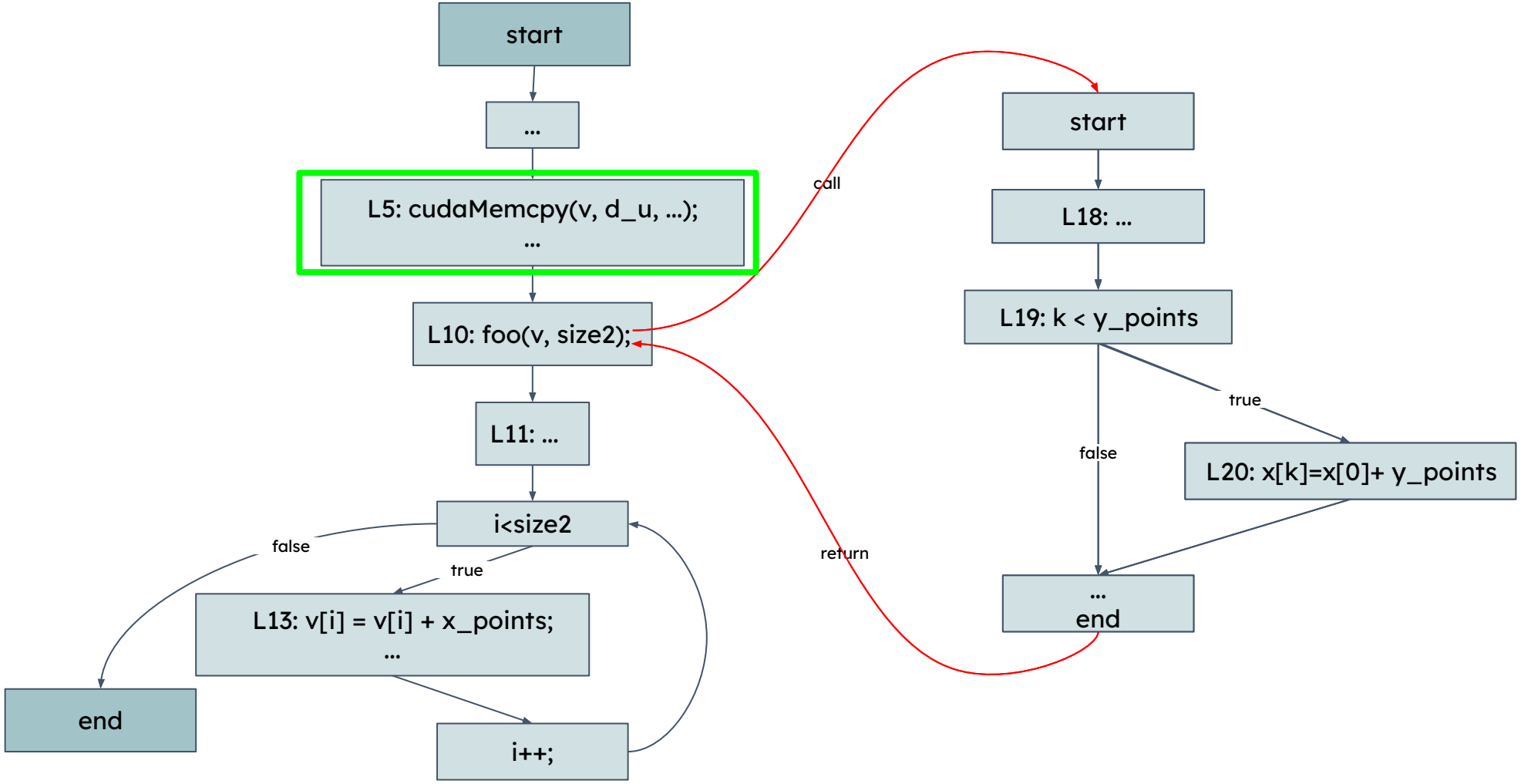
Step 1: Finding Target Locations (Contd.)



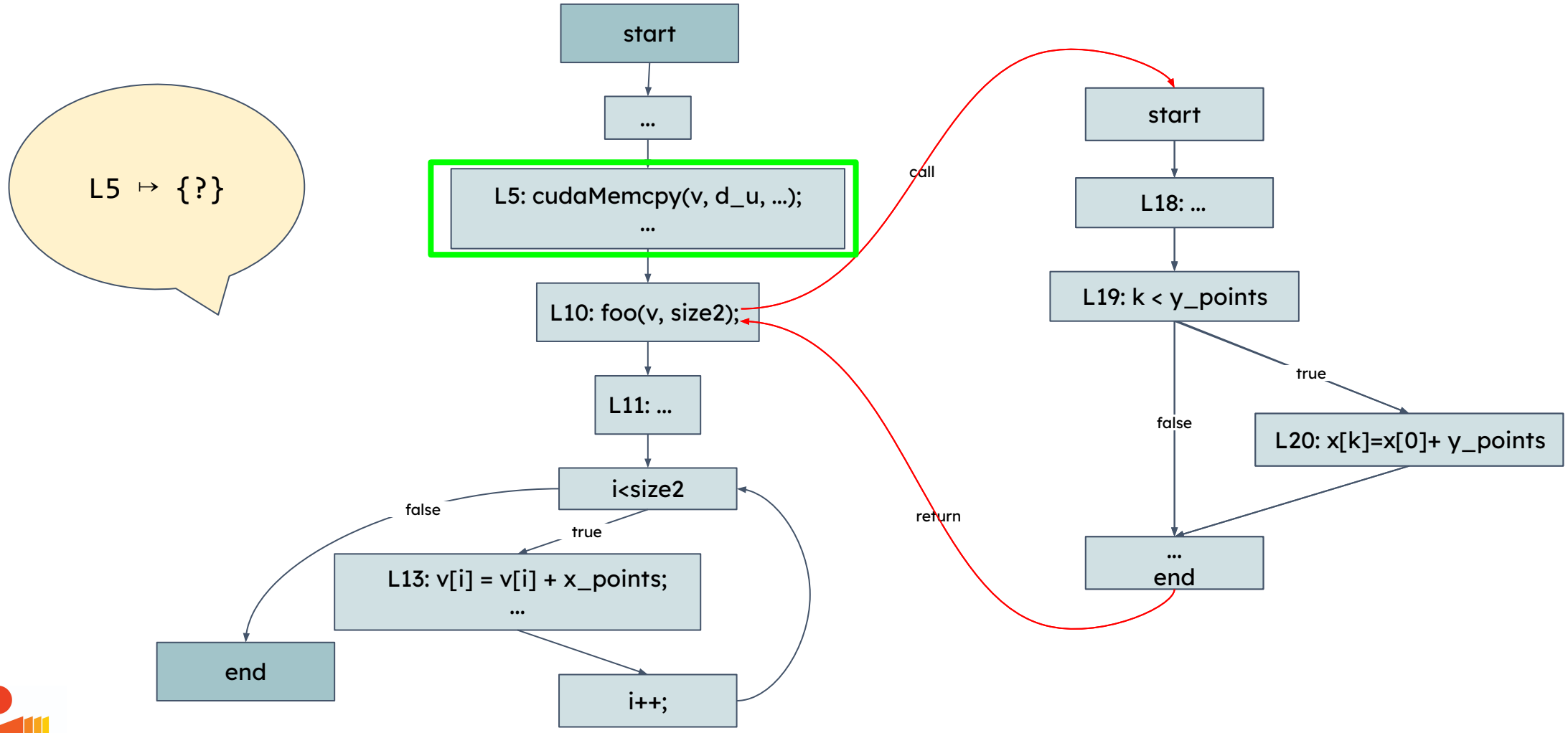
Step 1: Finding Target Locations (Contd.)



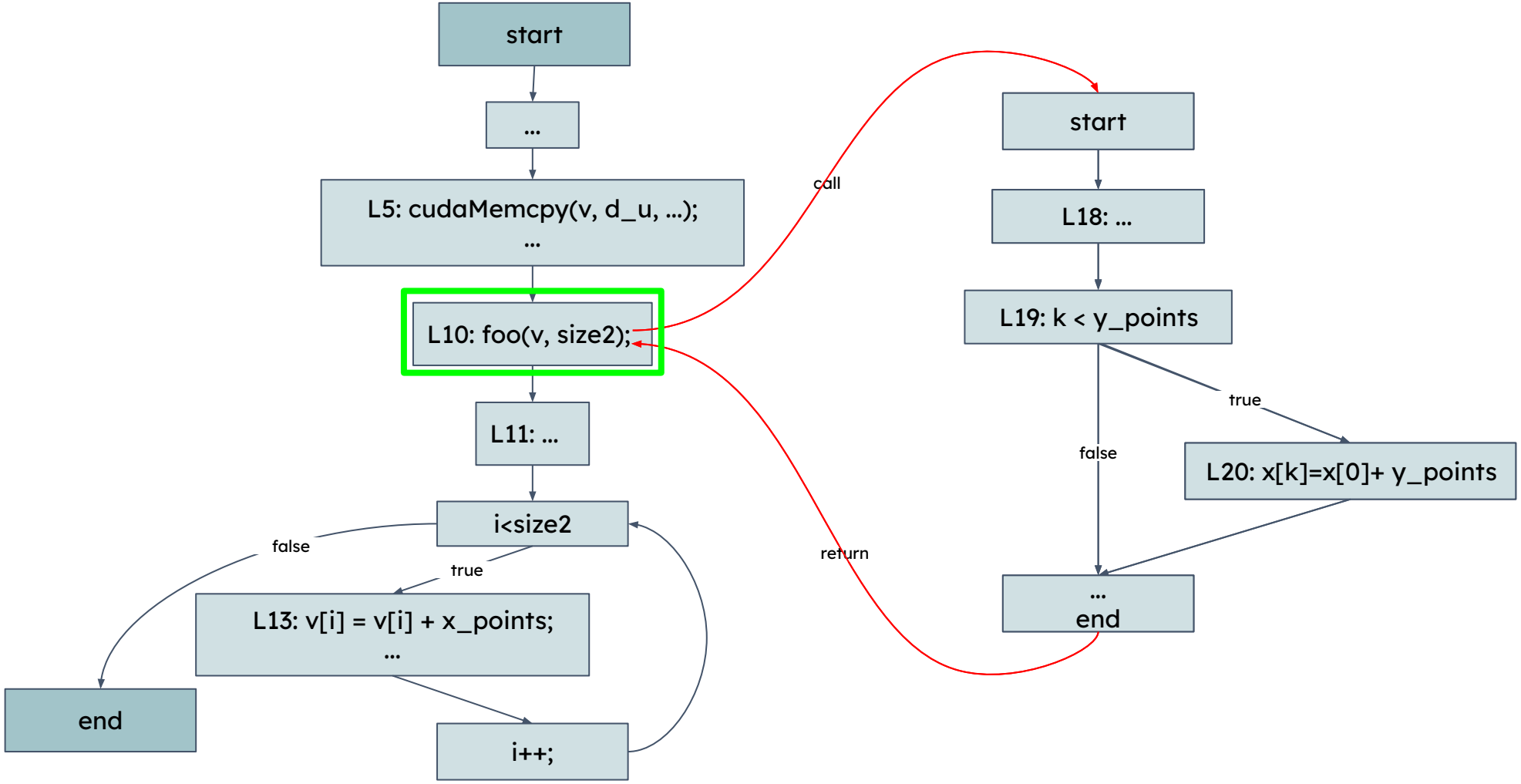
Step 1: Finding Target Locations (Contd.)



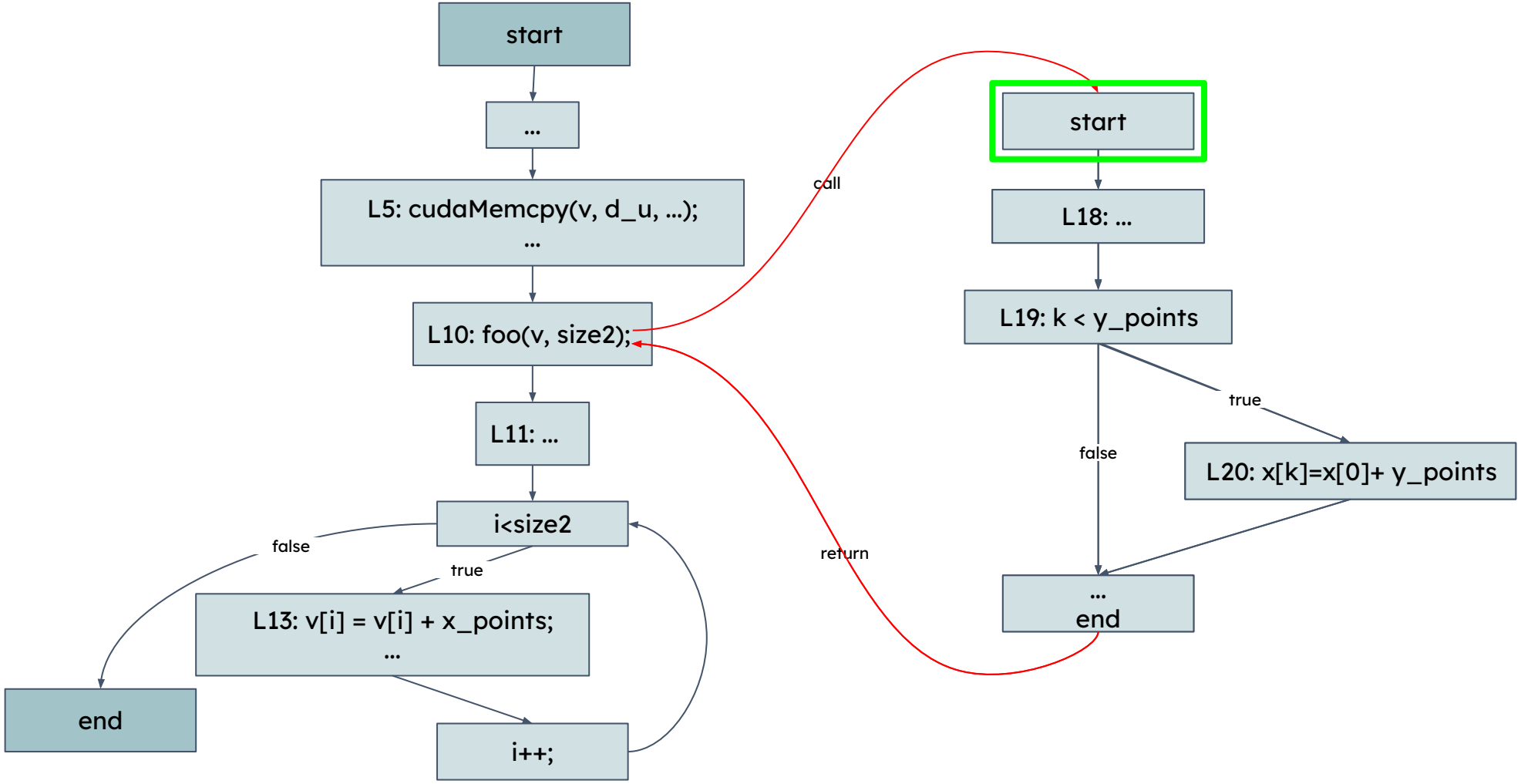
Step 1: Finding Target Locations (Contd.)



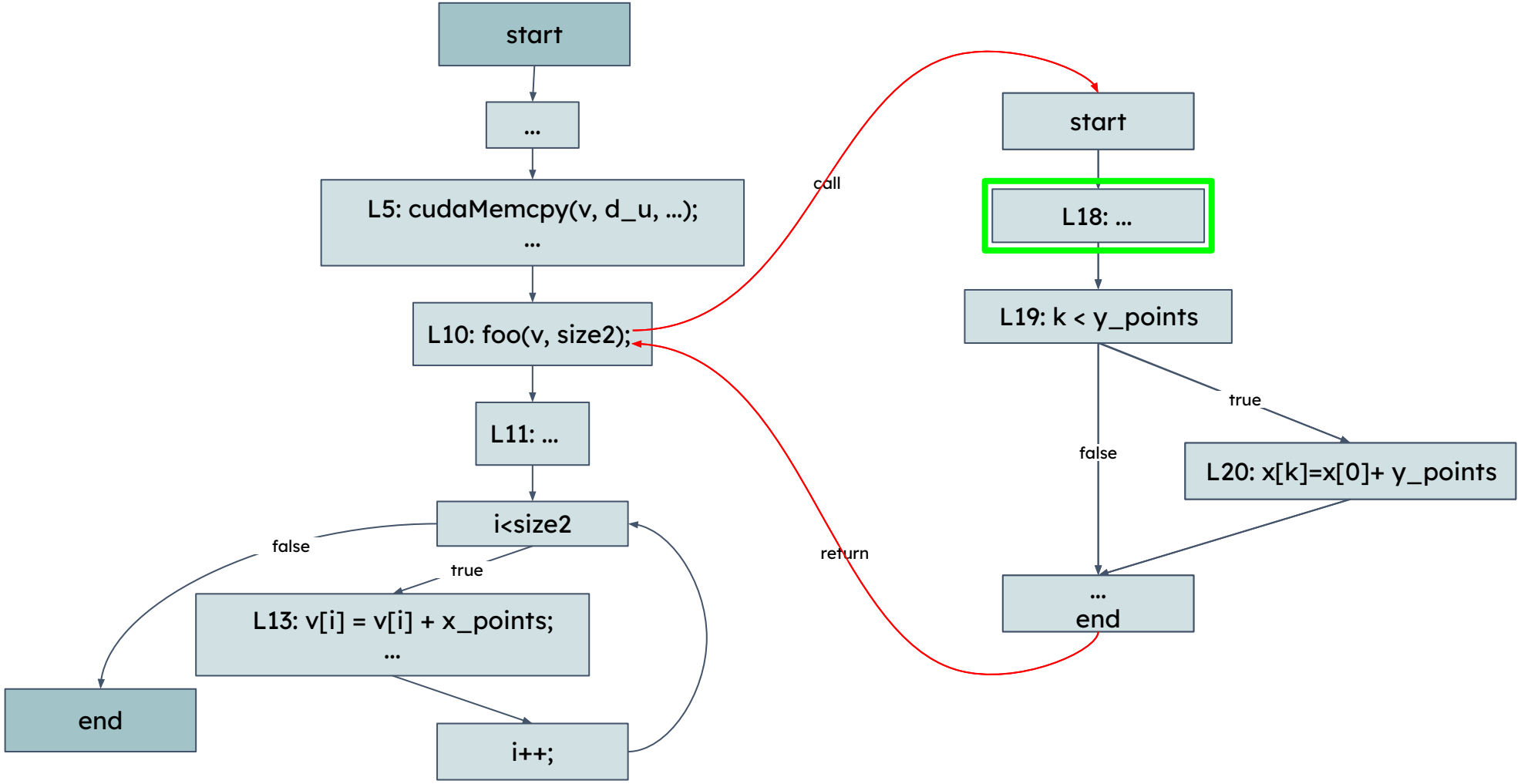
Step 1: Finding Target Locations (Contd.)



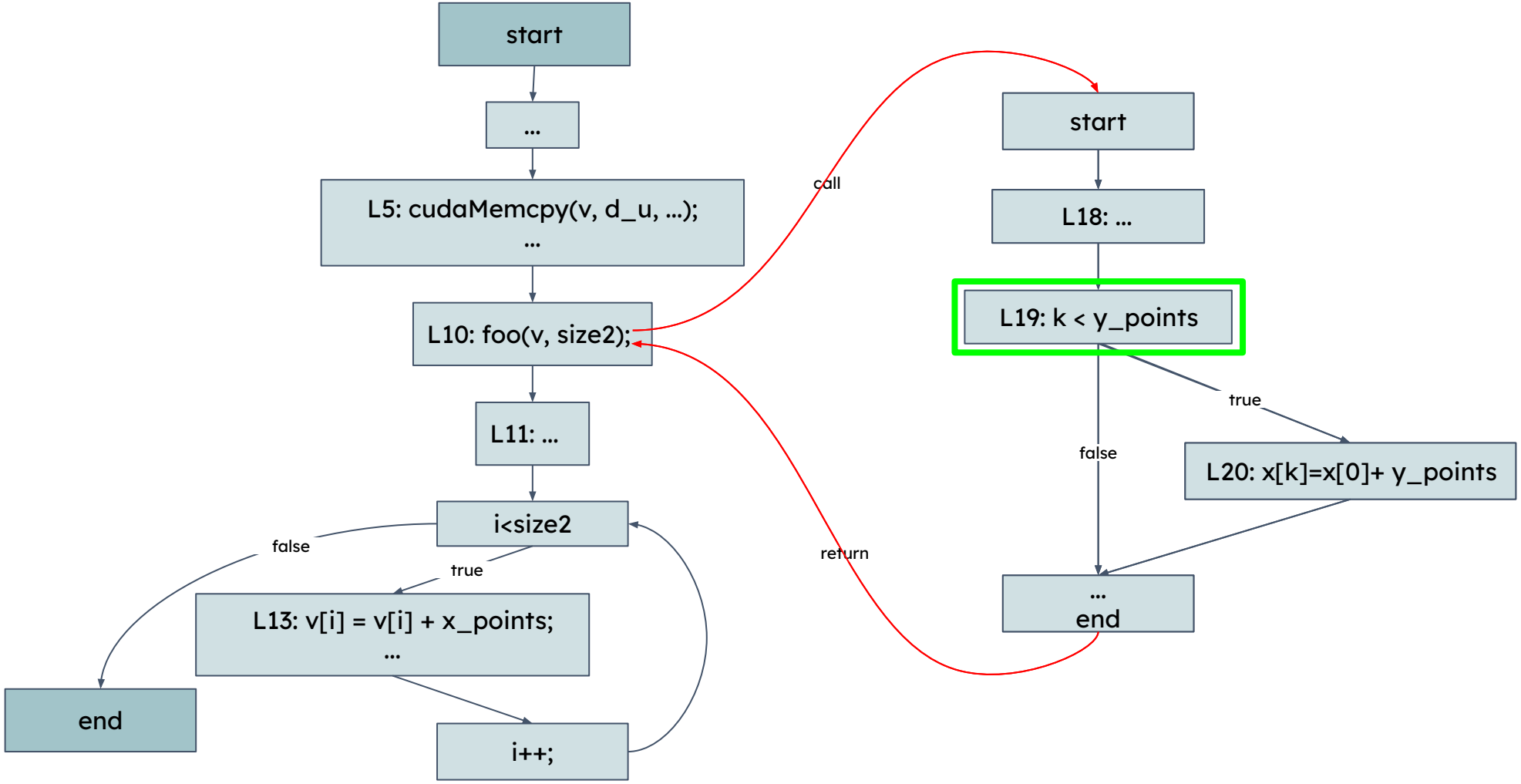
Step 1: Finding Target Locations (Contd.)



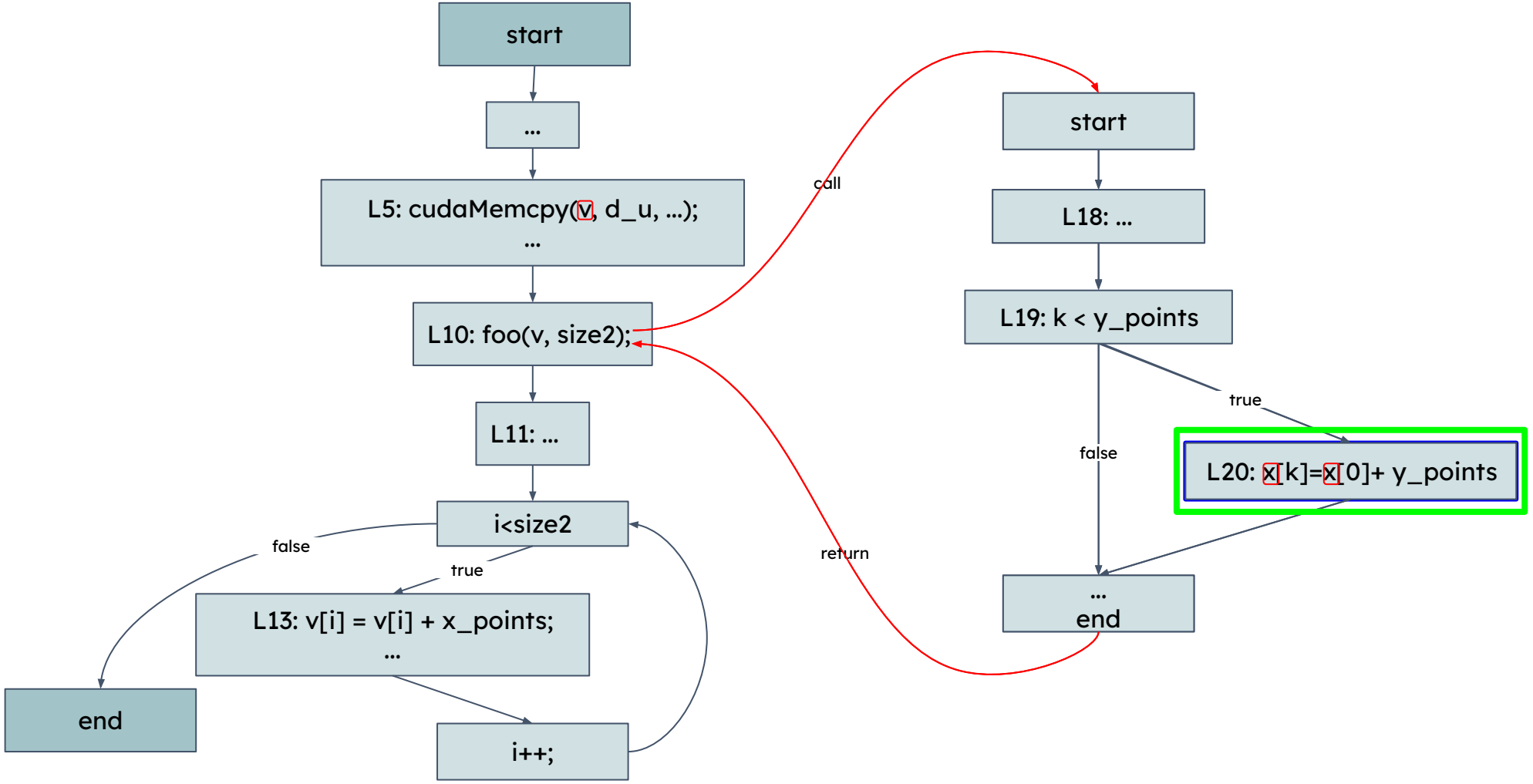
Step 1: Finding Target Locations (Contd.)



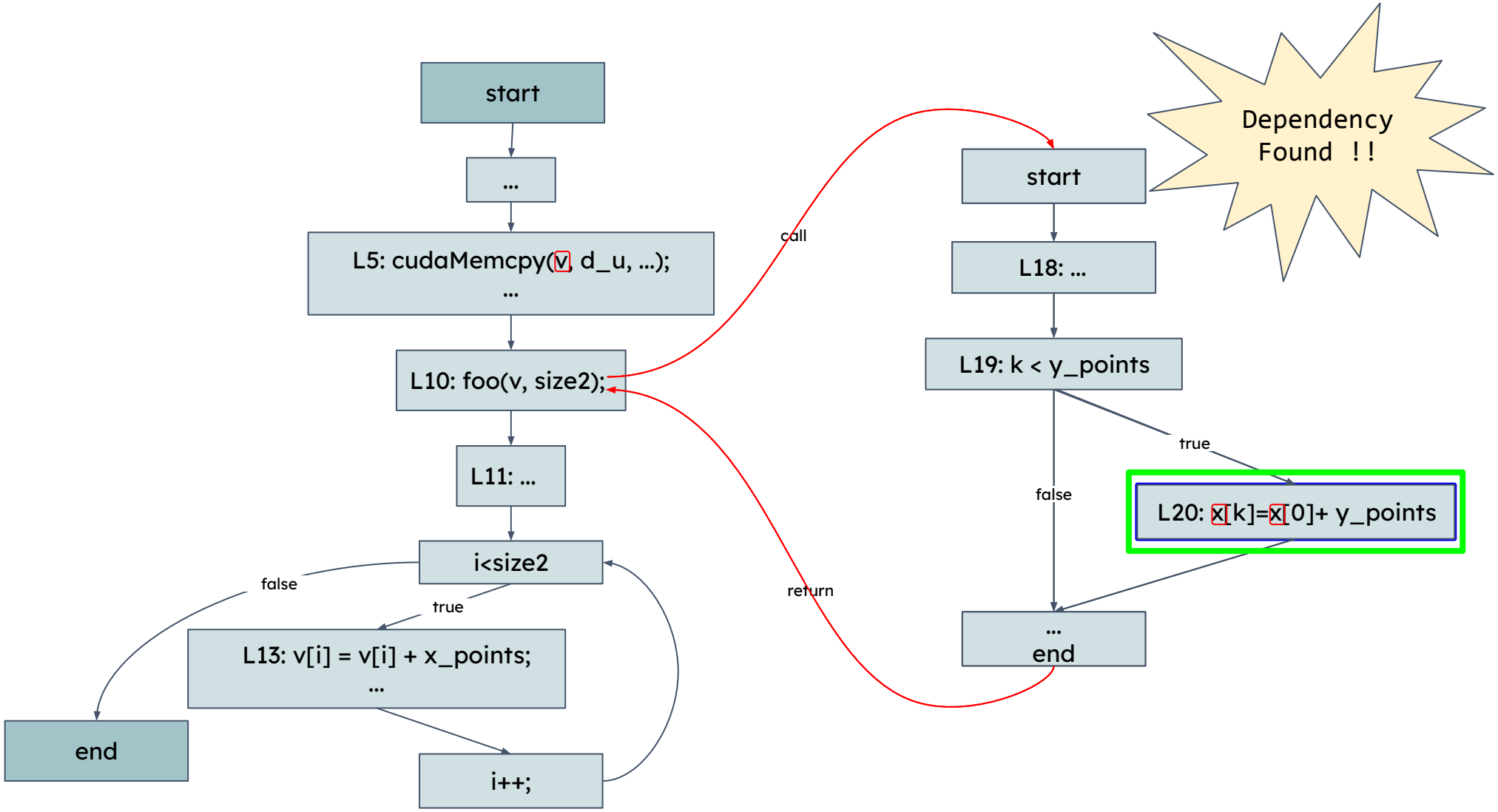
Step 1: Finding Target Locations (Contd.)



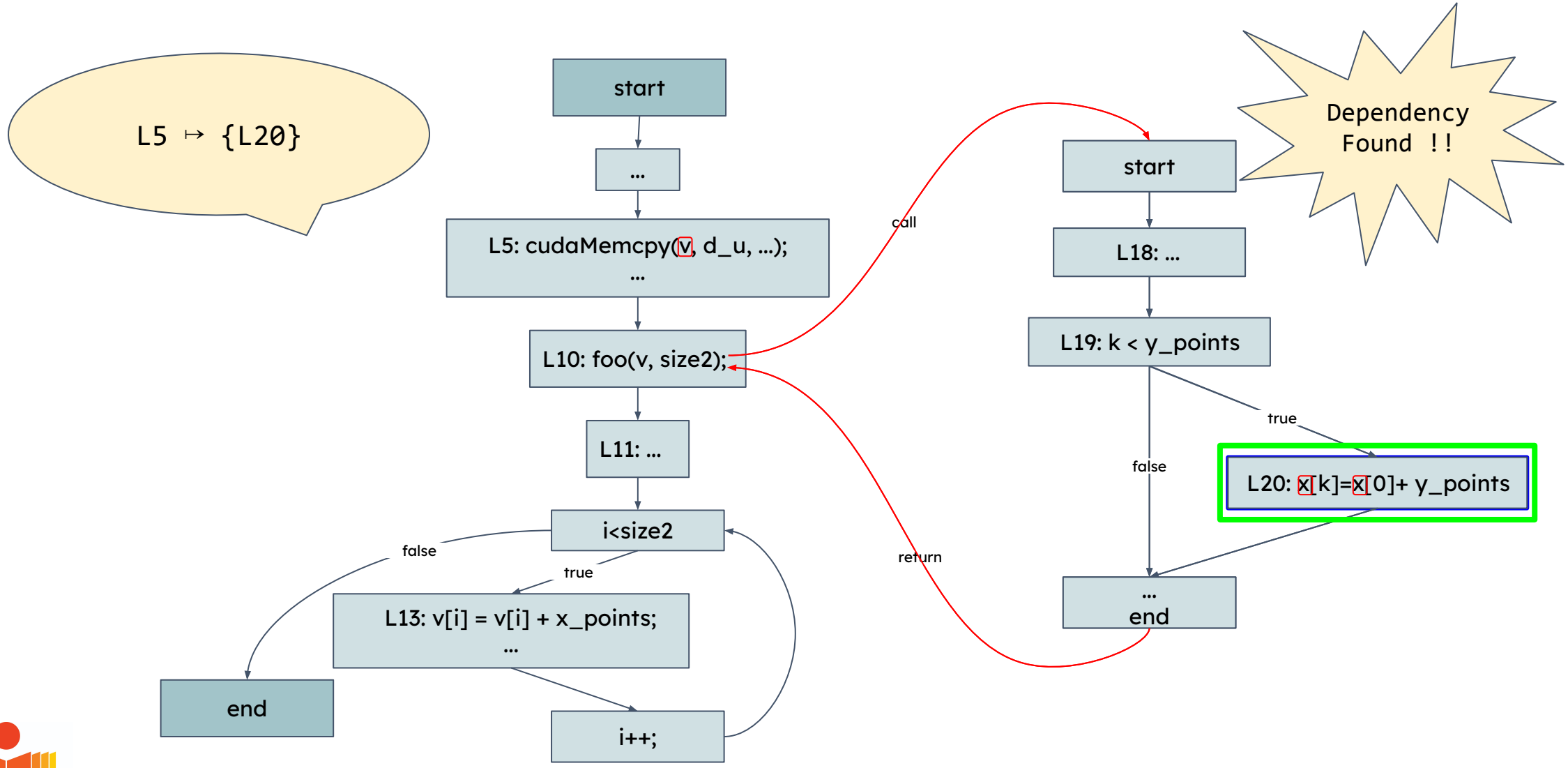
Step 1: Finding Target Locations (Contd.)



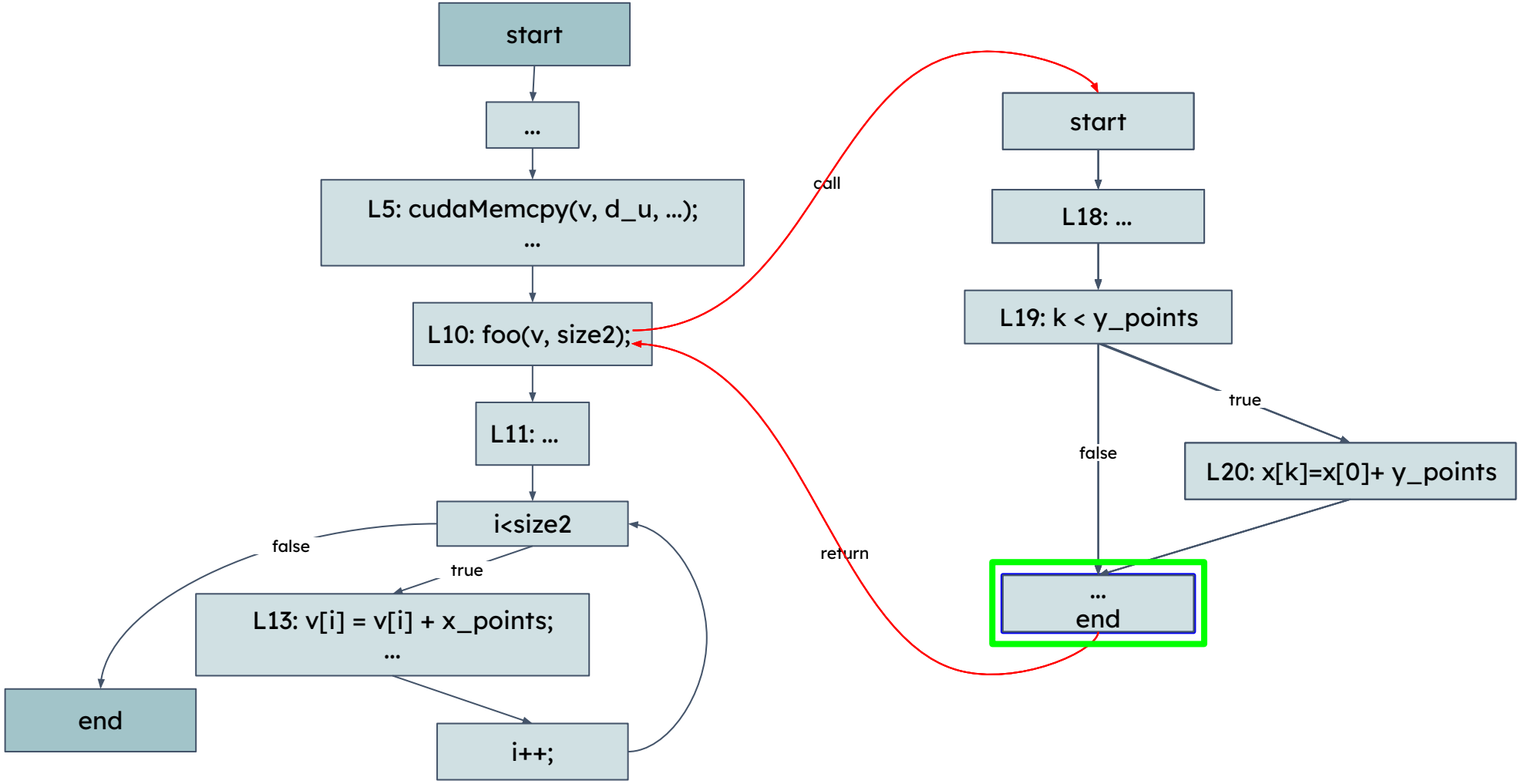
Step 1: Finding Target Locations (Contd.)



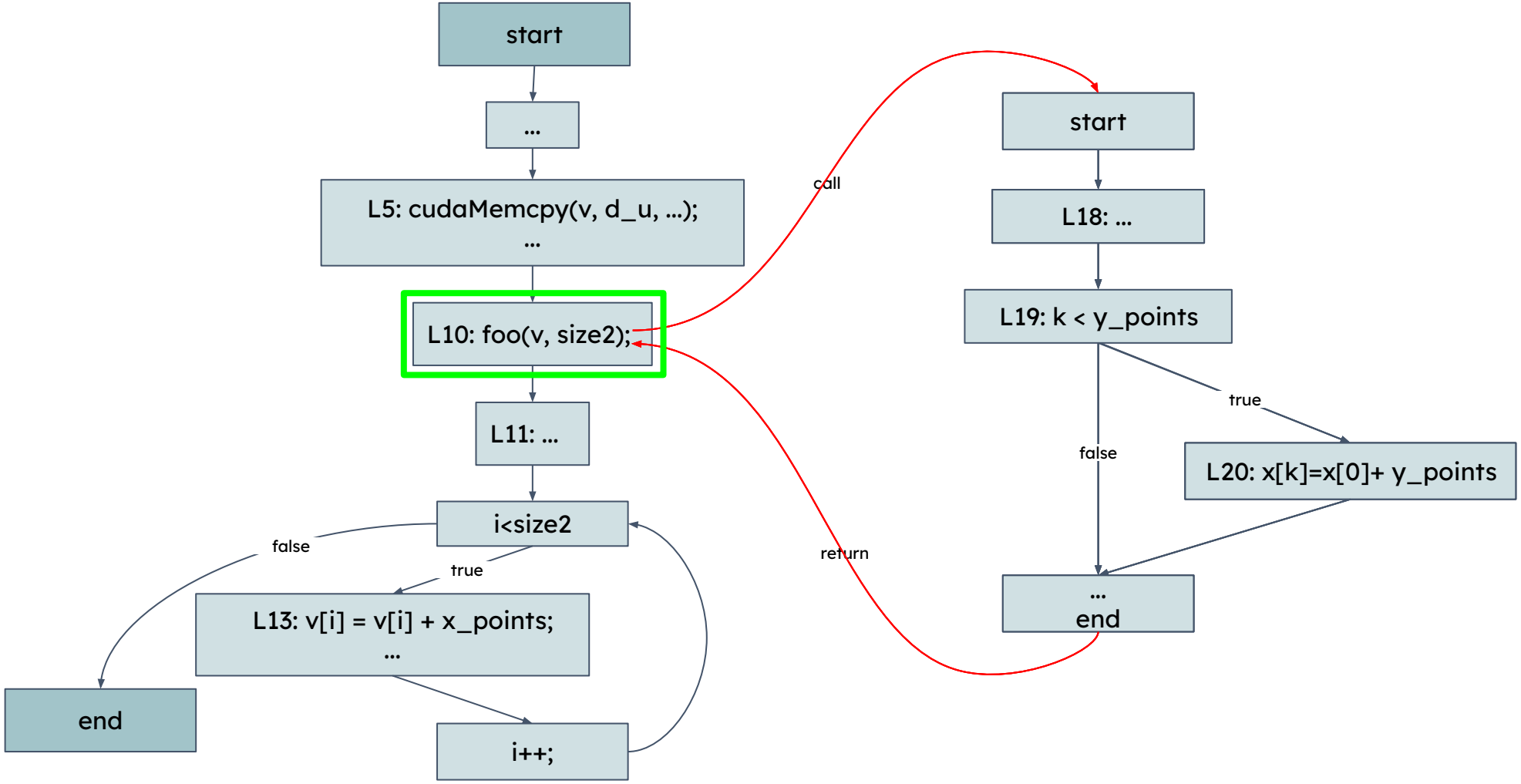
Step 1: Finding Target Locations (Contd.)



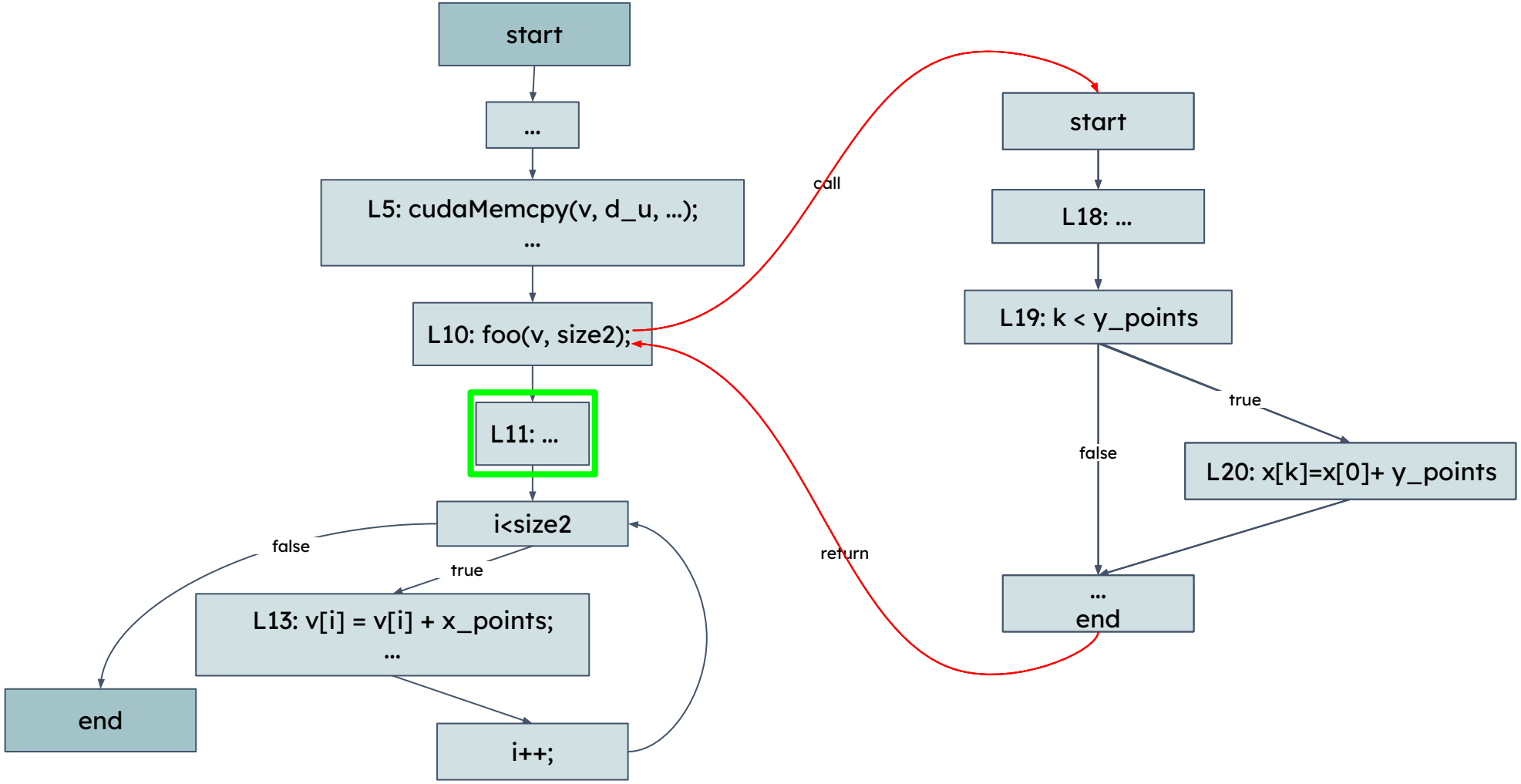
Step 1: Finding Target Locations (Contd.)



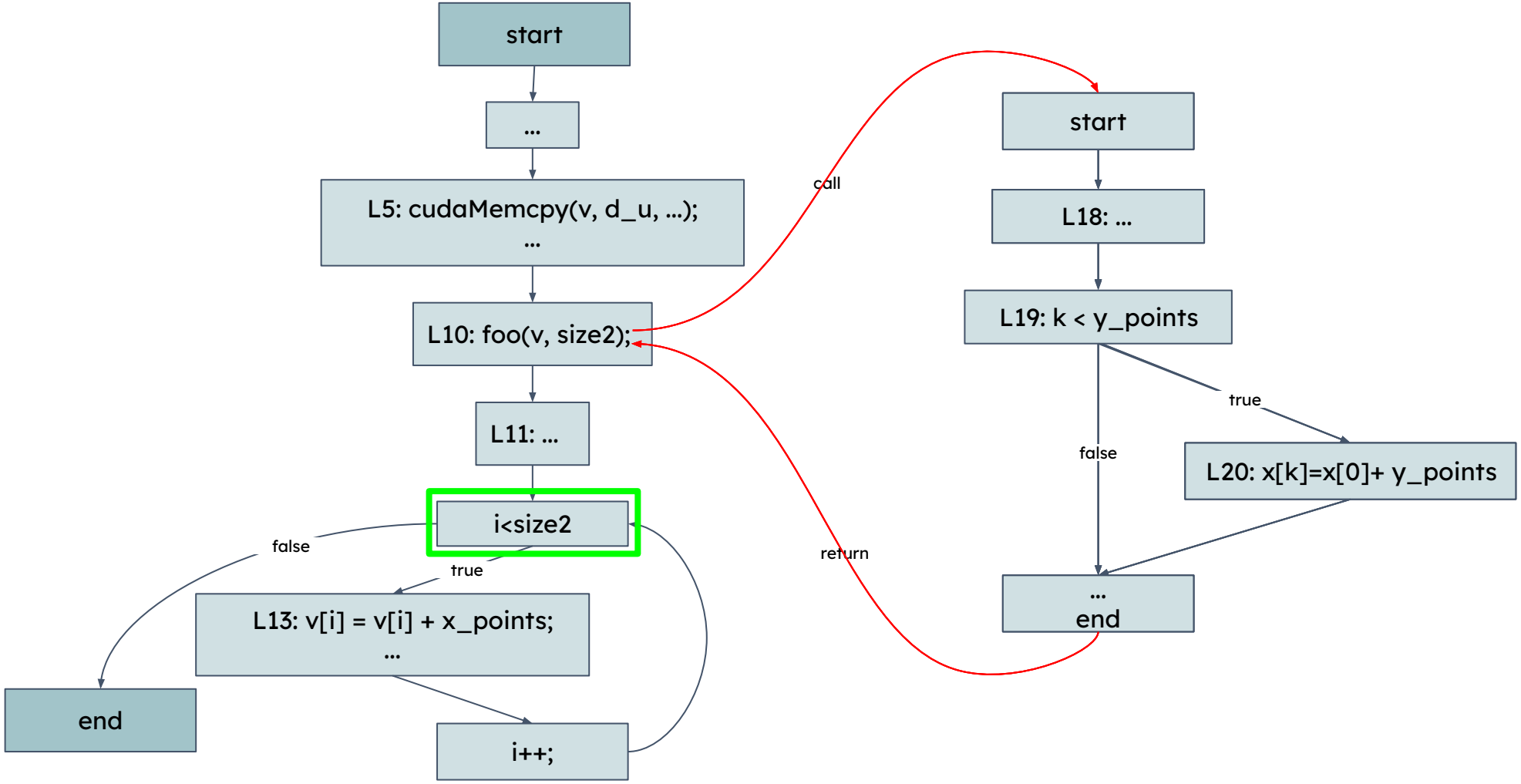
Step 1: Finding Target Locations (Contd.)



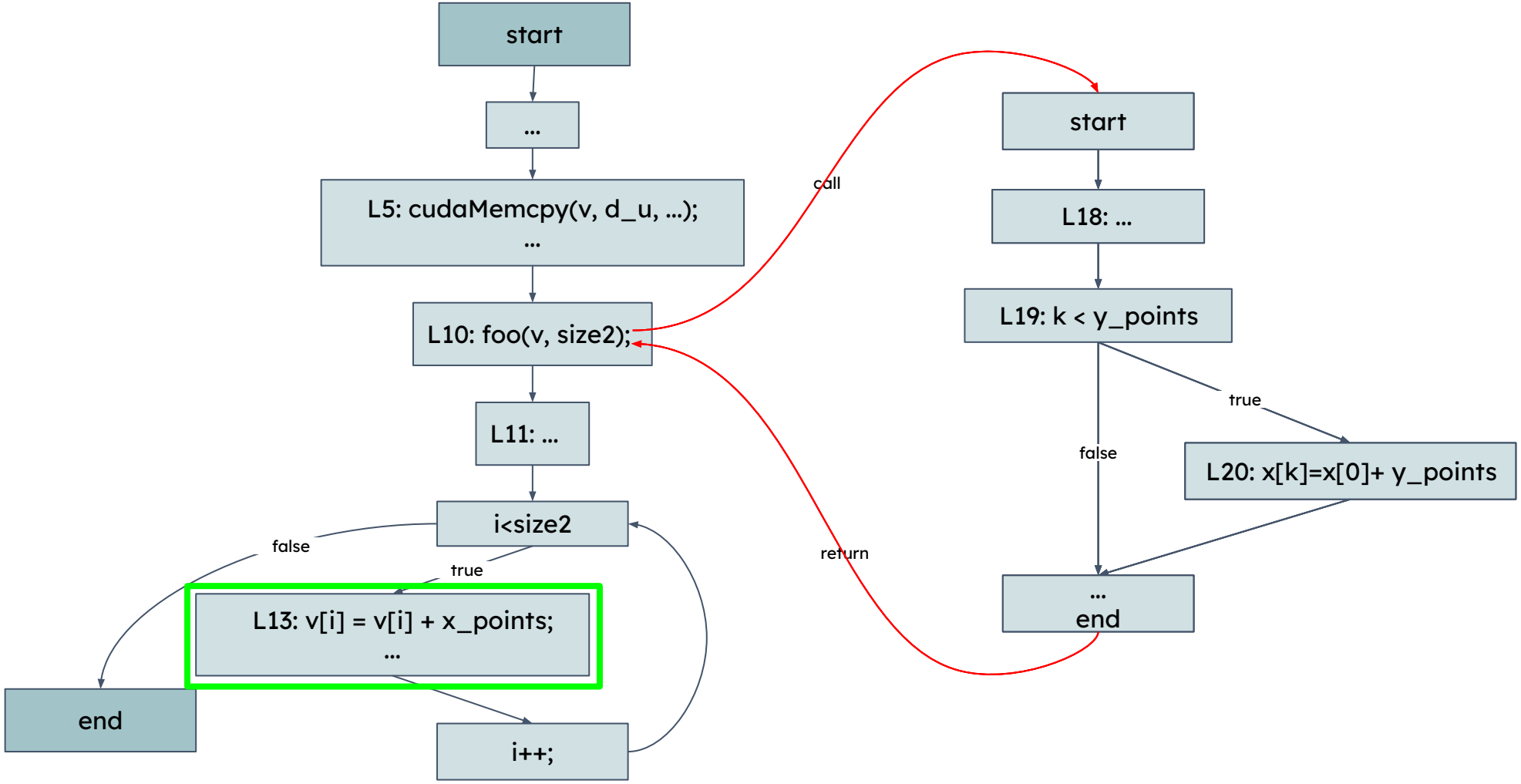
Step 1: Finding Target Locations (Contd.)



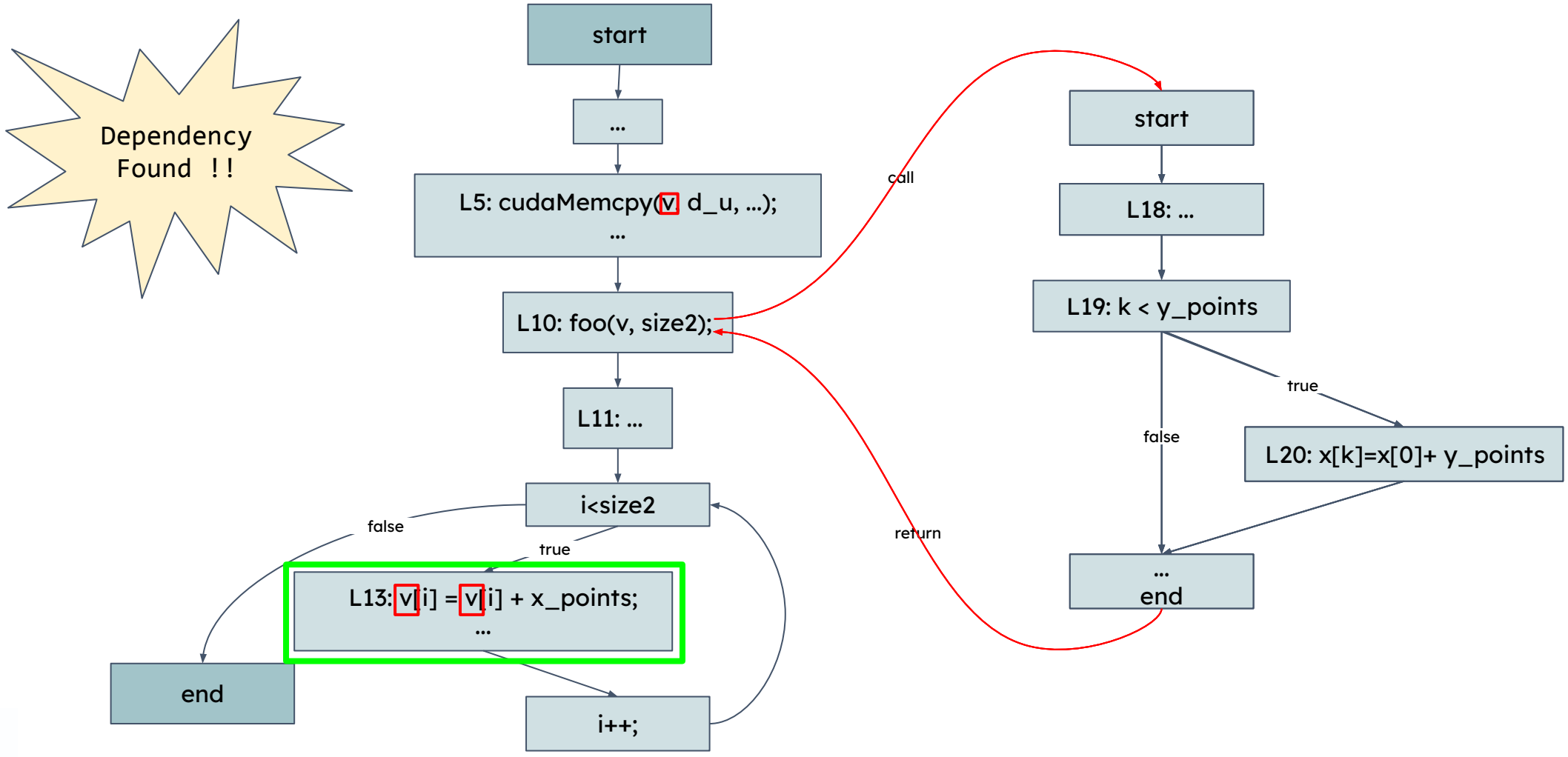
Step 1: Finding Target Locations (Contd.)



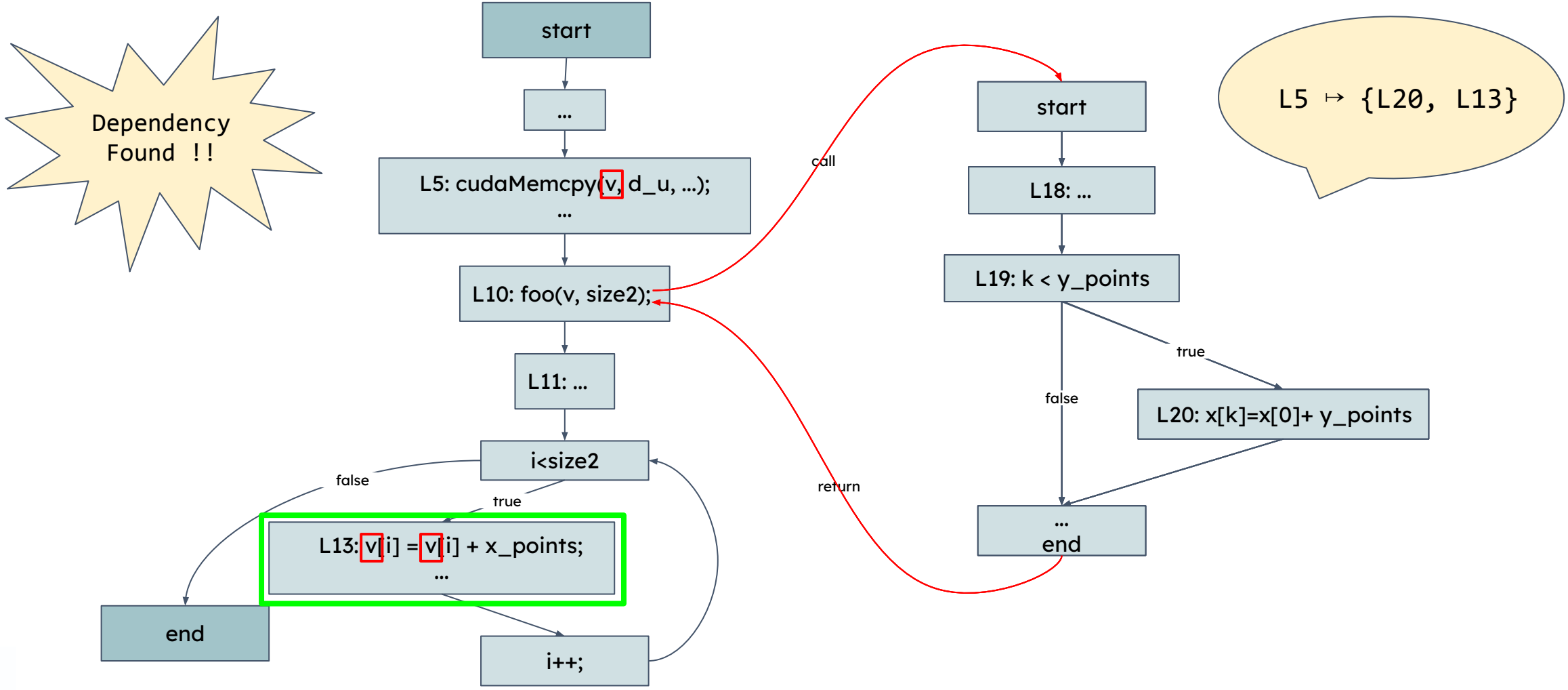
Step 1: Finding Target Locations (Contd.)



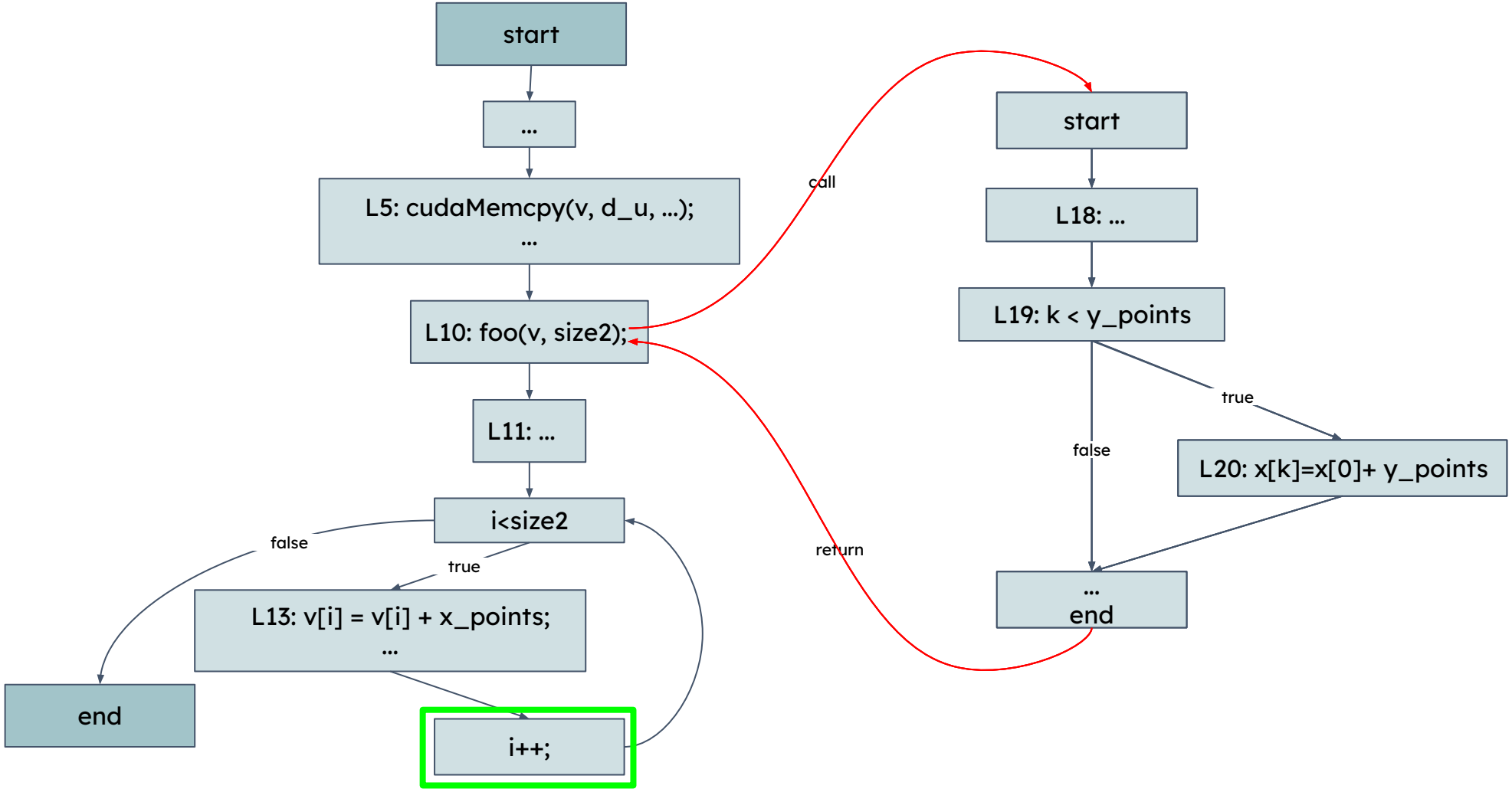
Step 1: Finding Target Locations (Contd.)



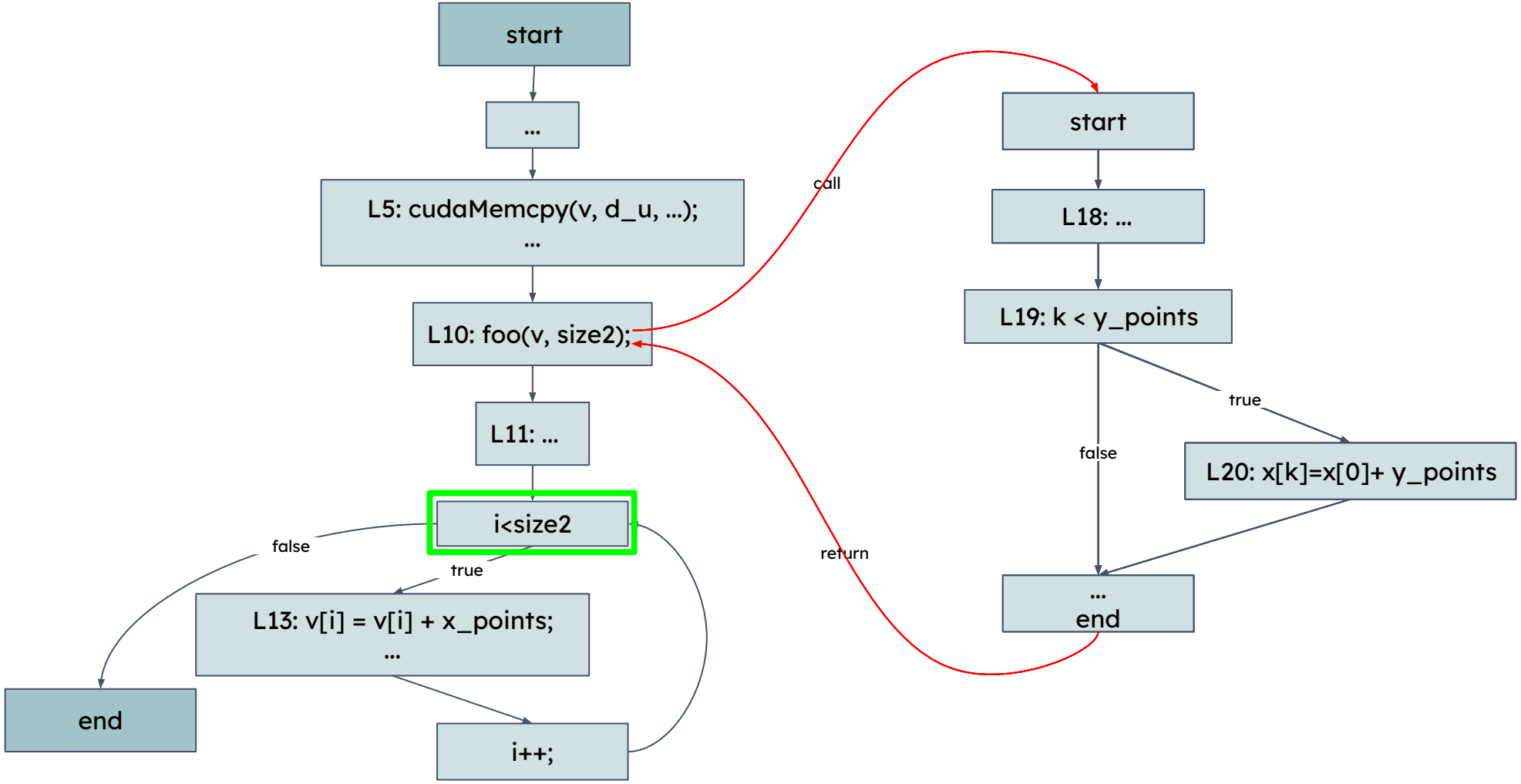
Step 1: Finding Target Locations (Contd.)



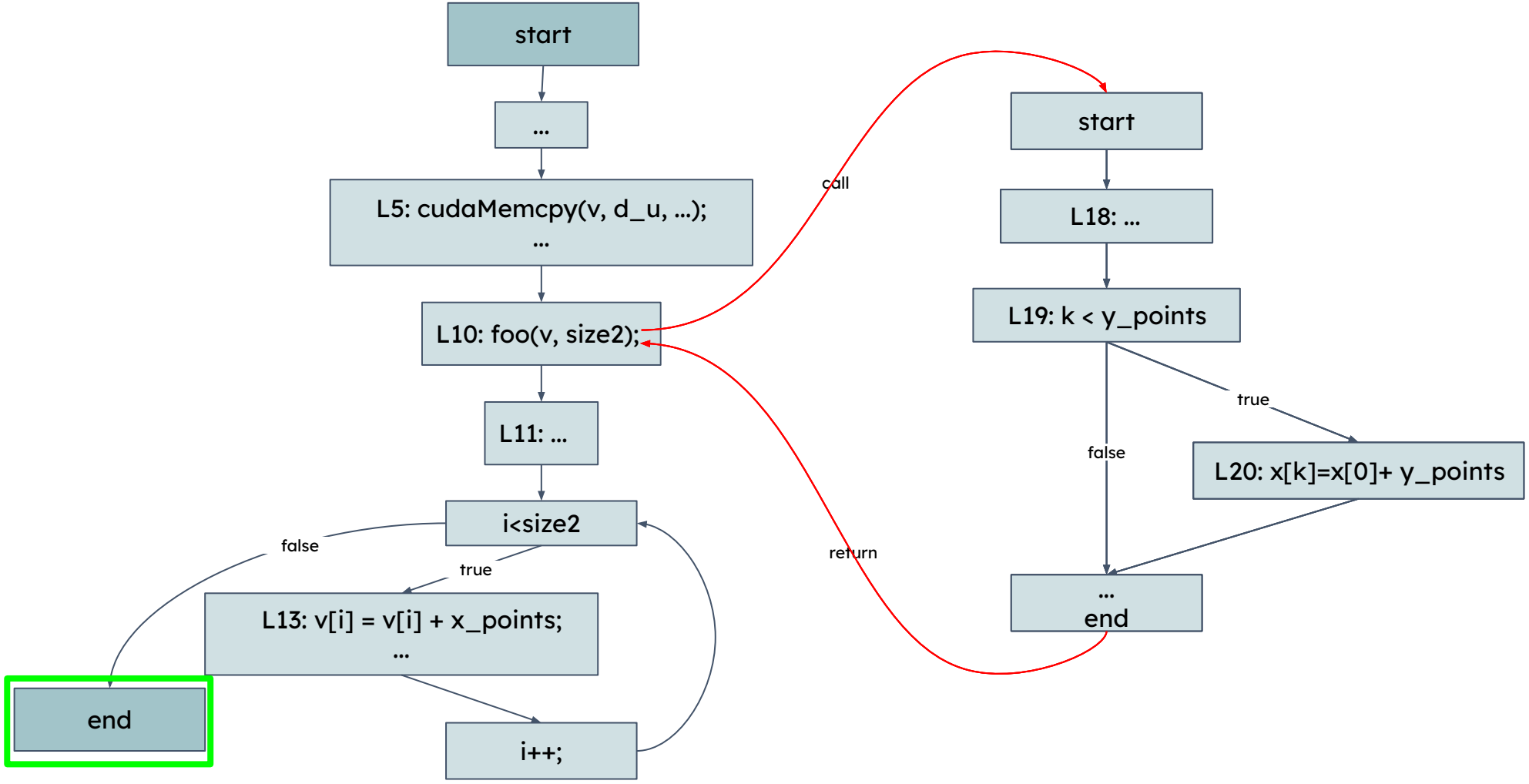
Step 1: Finding Target Locations (Contd.)



Step 1: Finding Target Locations (Contd.)



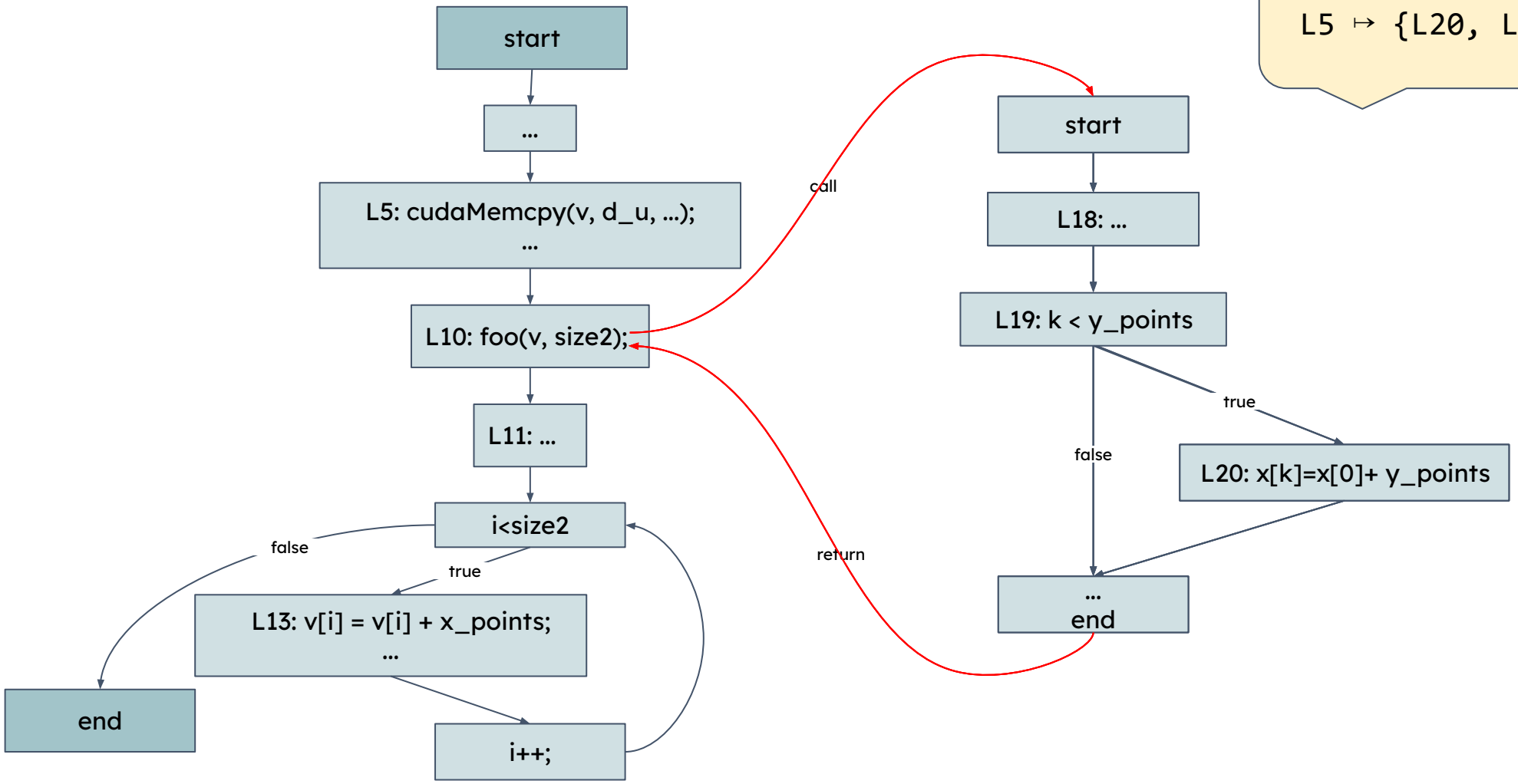
Step 1: Finding Target Locations (Contd.)



Step 2: Updating Target Locations: Motivation

Target Map

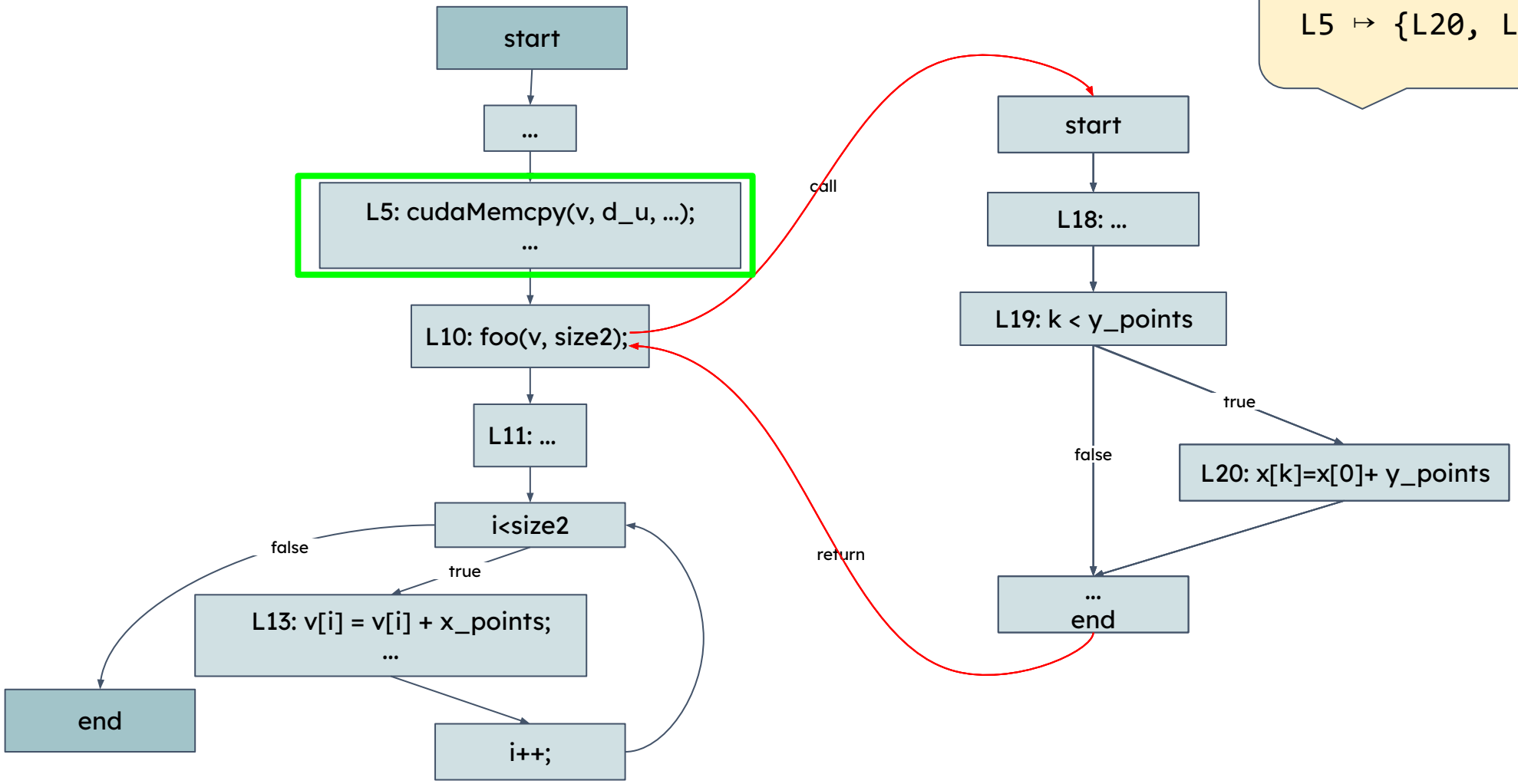
L5 ↦ {L20, L13}



Step 2: Updating Target Locations: Motivation

Target Map

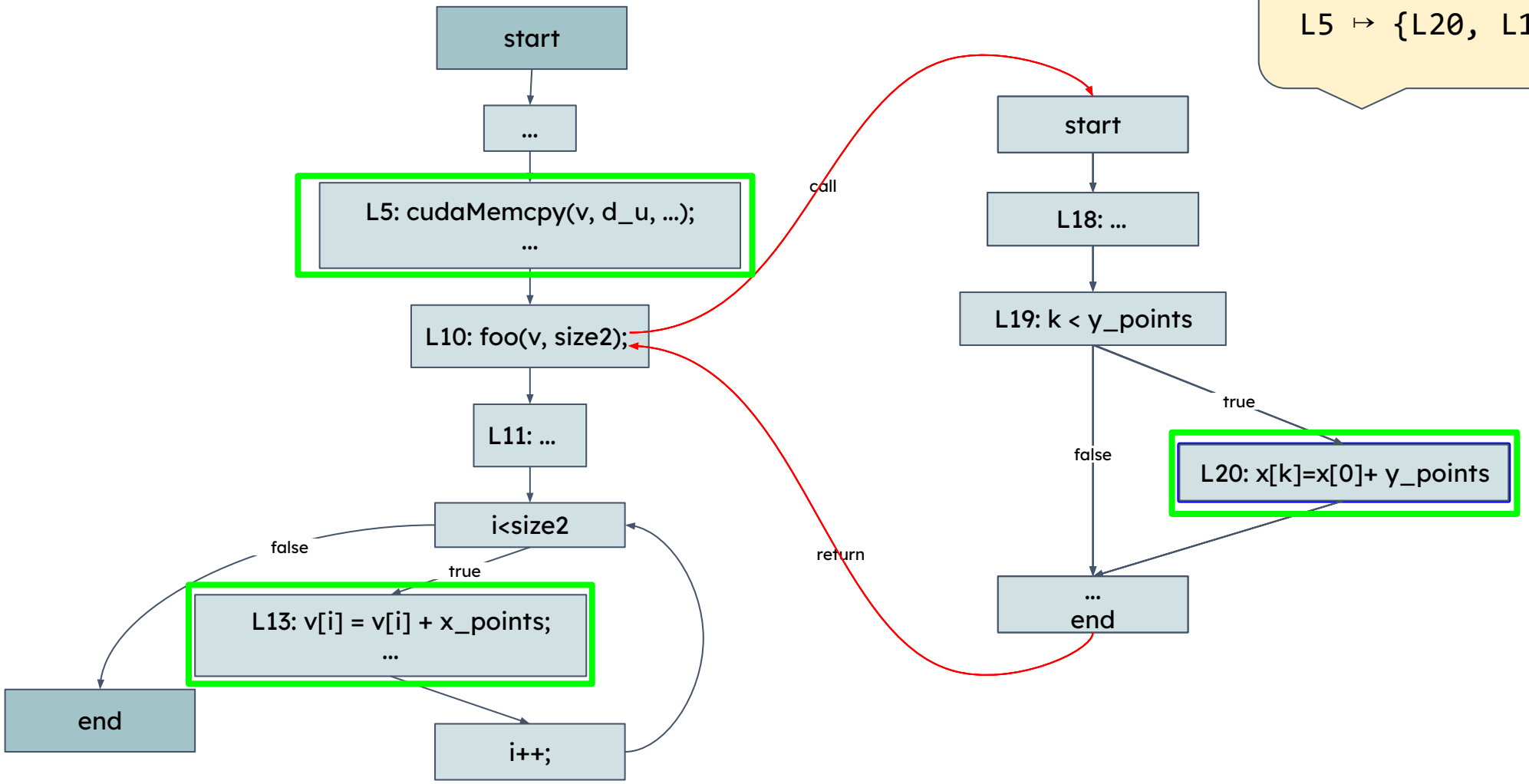
L5 ↦ {L20, L13}



Step 2: Updating Target Locations: Motivation

Target Map

L5 ↦ {L20, L13}

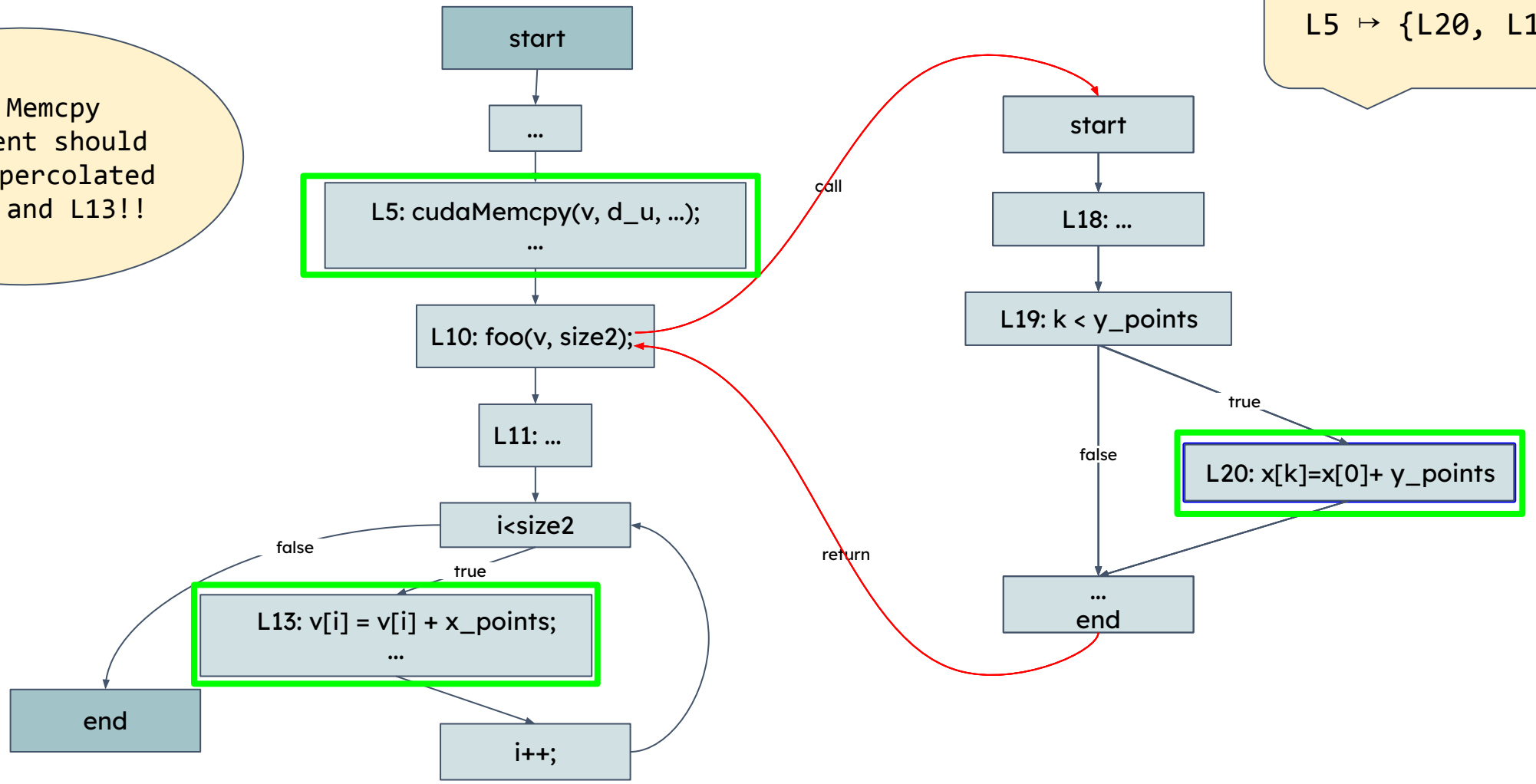


Step 2: Updating Target Locations: Motivation

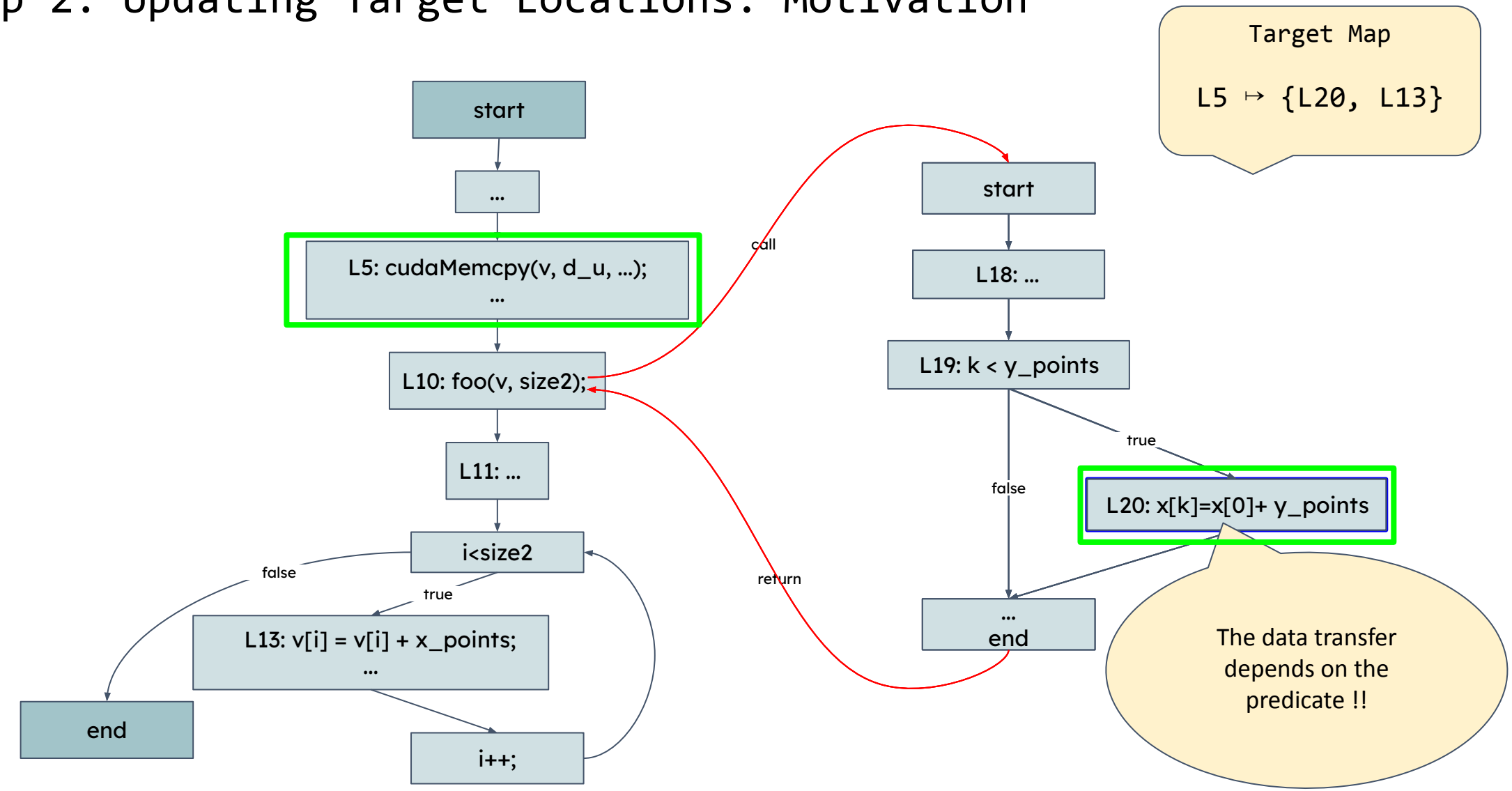
The Memcpy statement should not be percolated to L20 and L13!!

Target Map

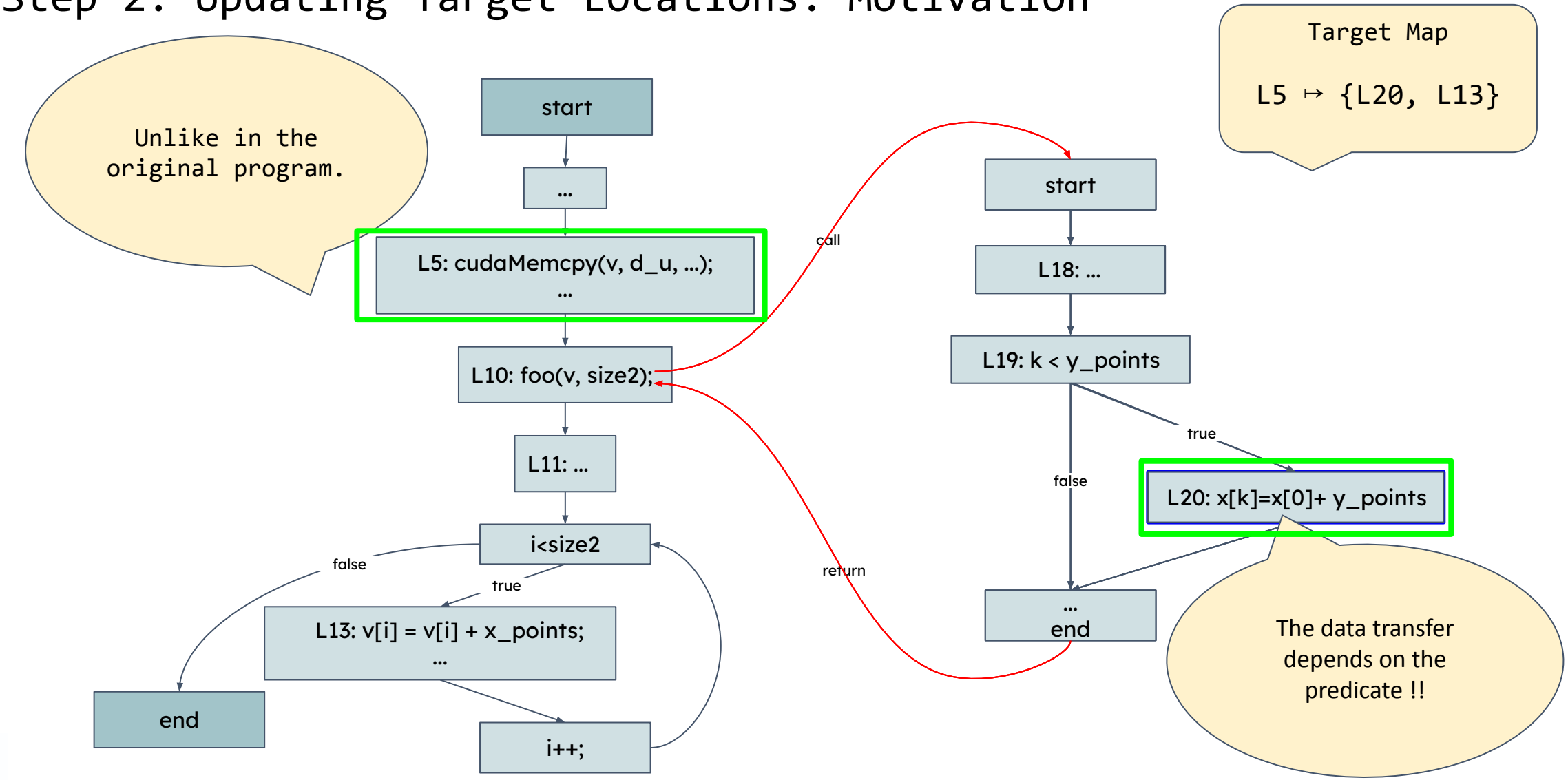
L5 ↦ {L20, L13}



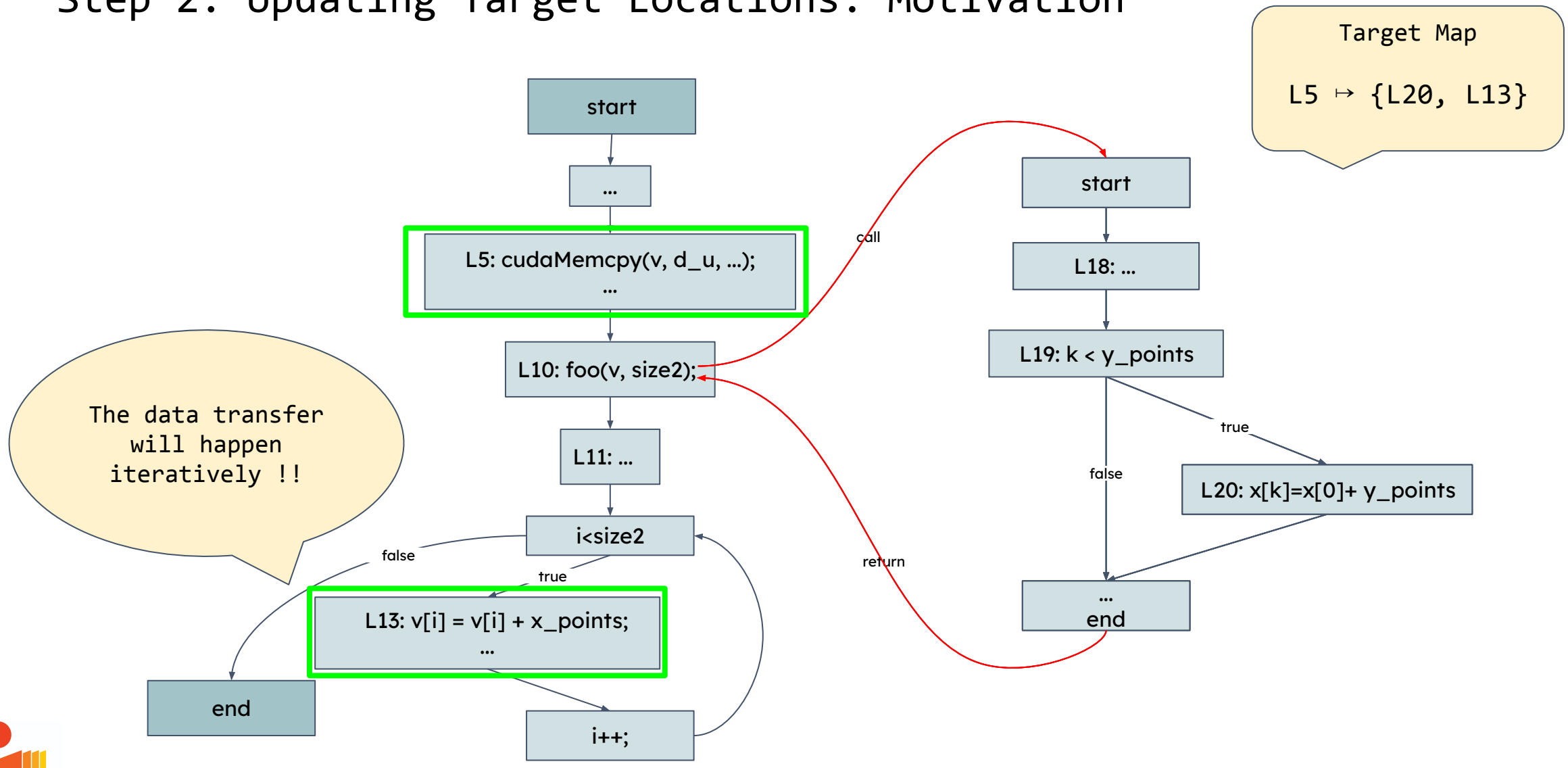
Step 2: Updating Target Locations: Motivation



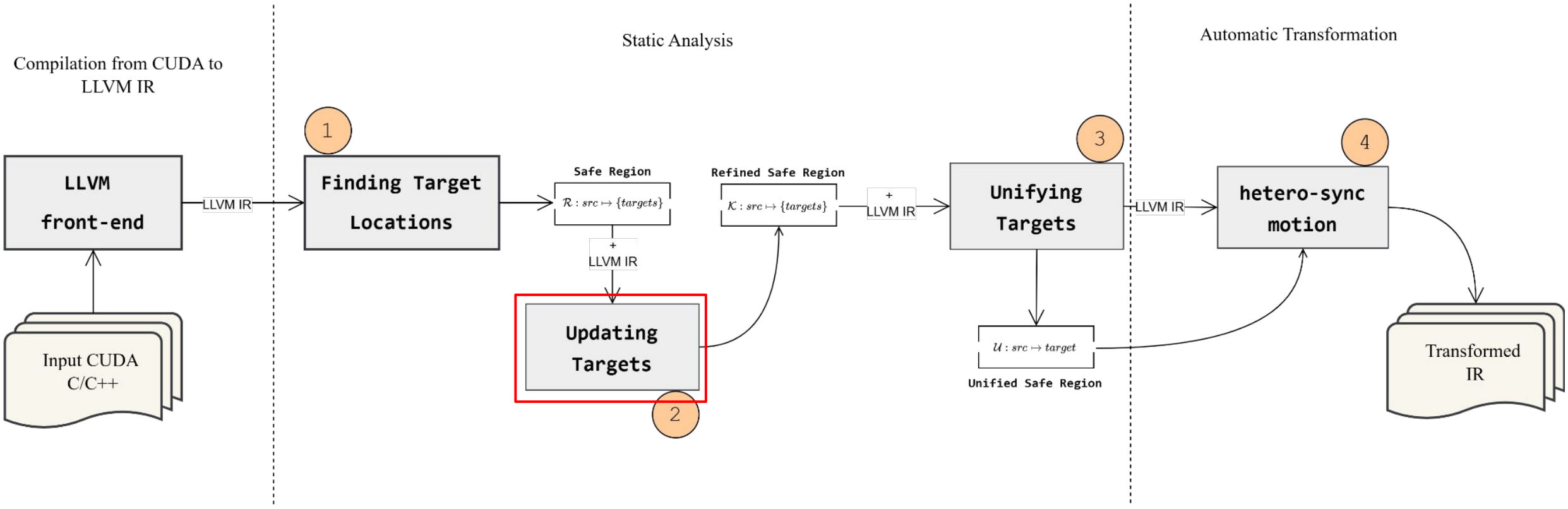
Step 2: Updating Target Locations: Motivation



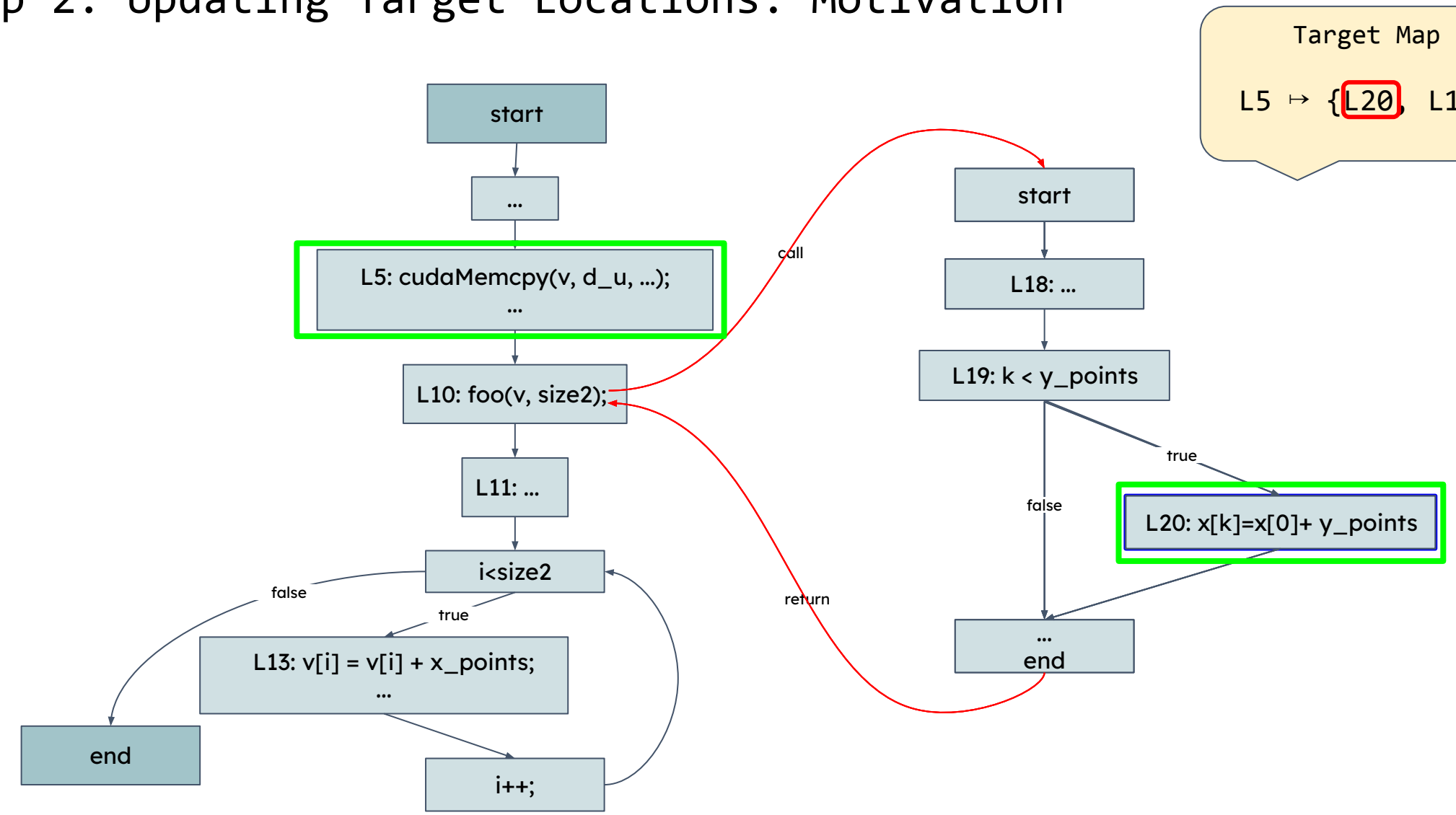
Step 2: Updating Target Locations: Motivation



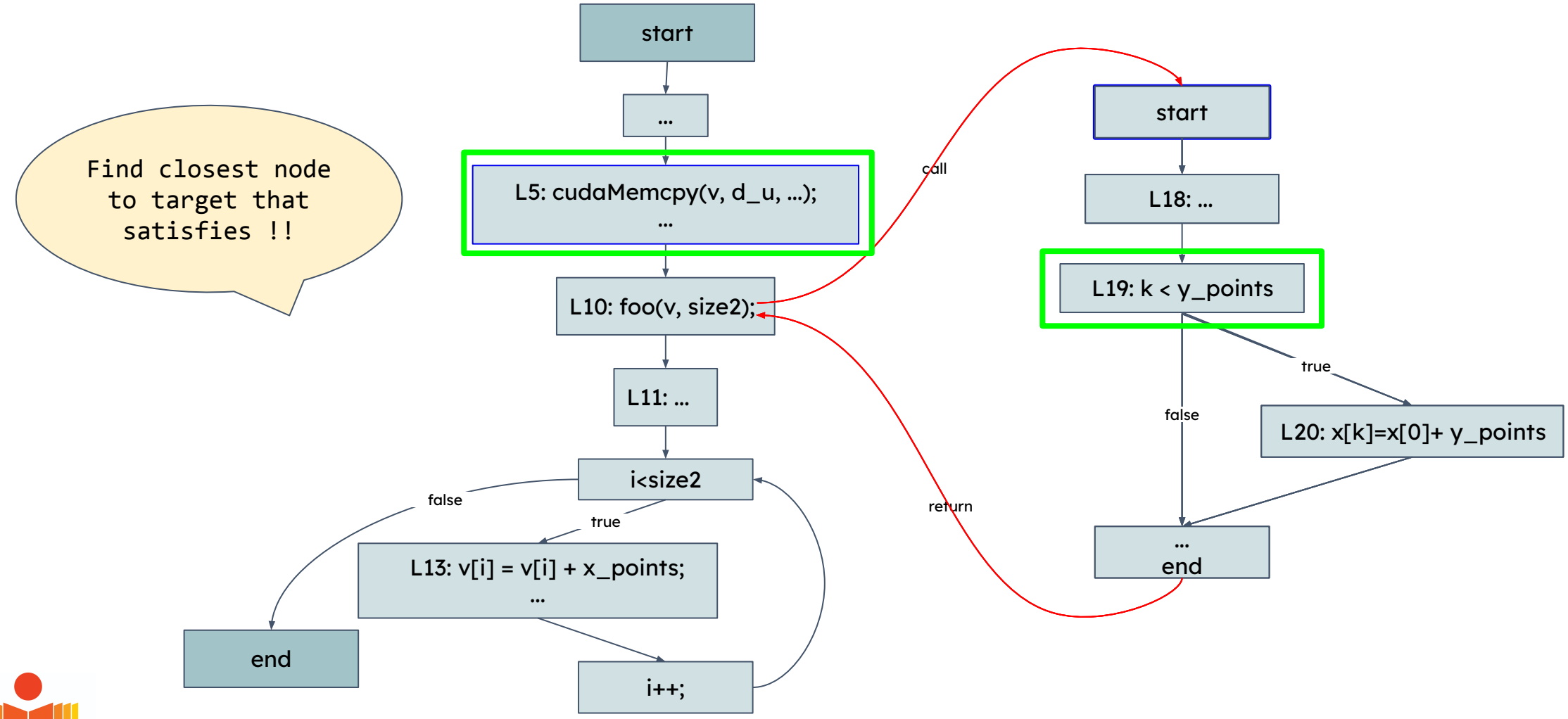
Step 2: Updating Target Locations



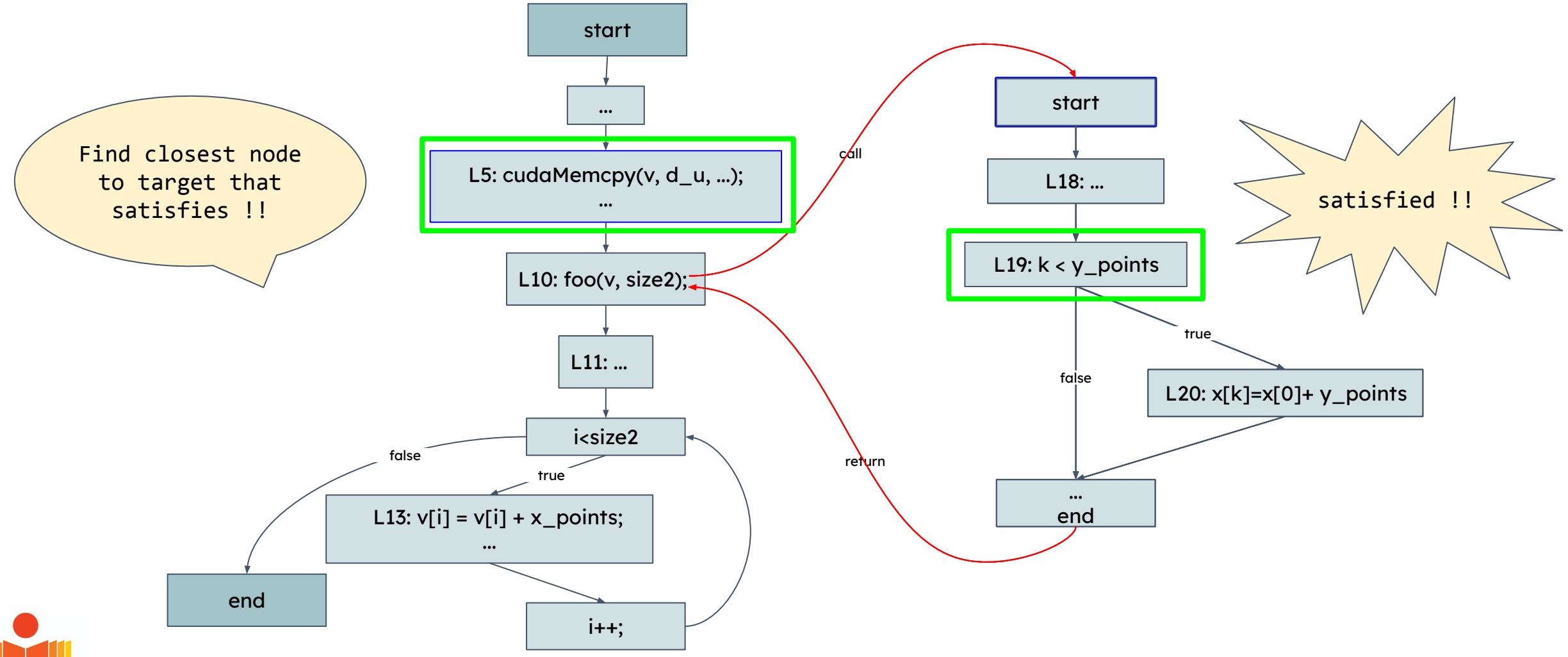
Step 2: Updating Target Locations: Motivation



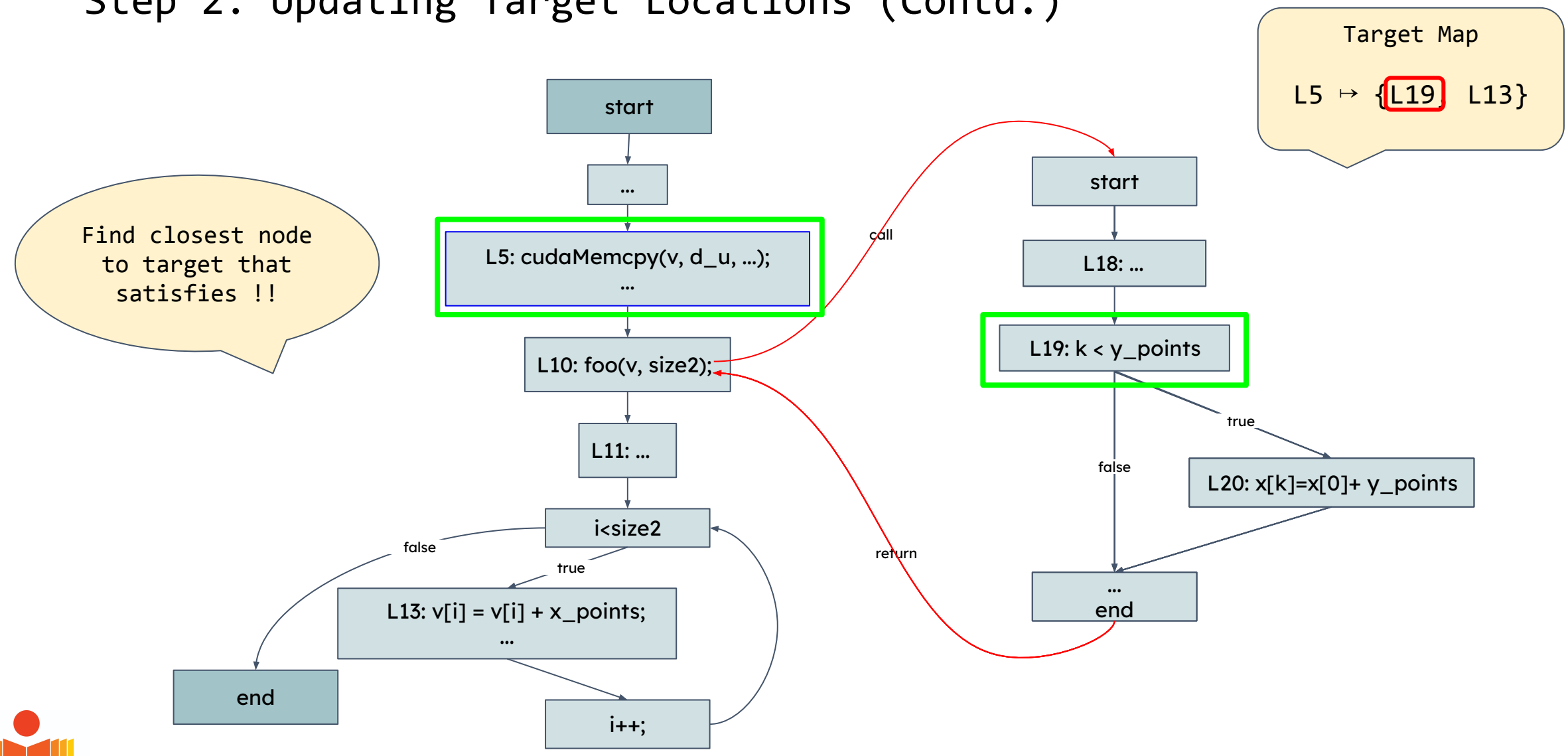
Step 2: Updating Target Locations (Contd.)



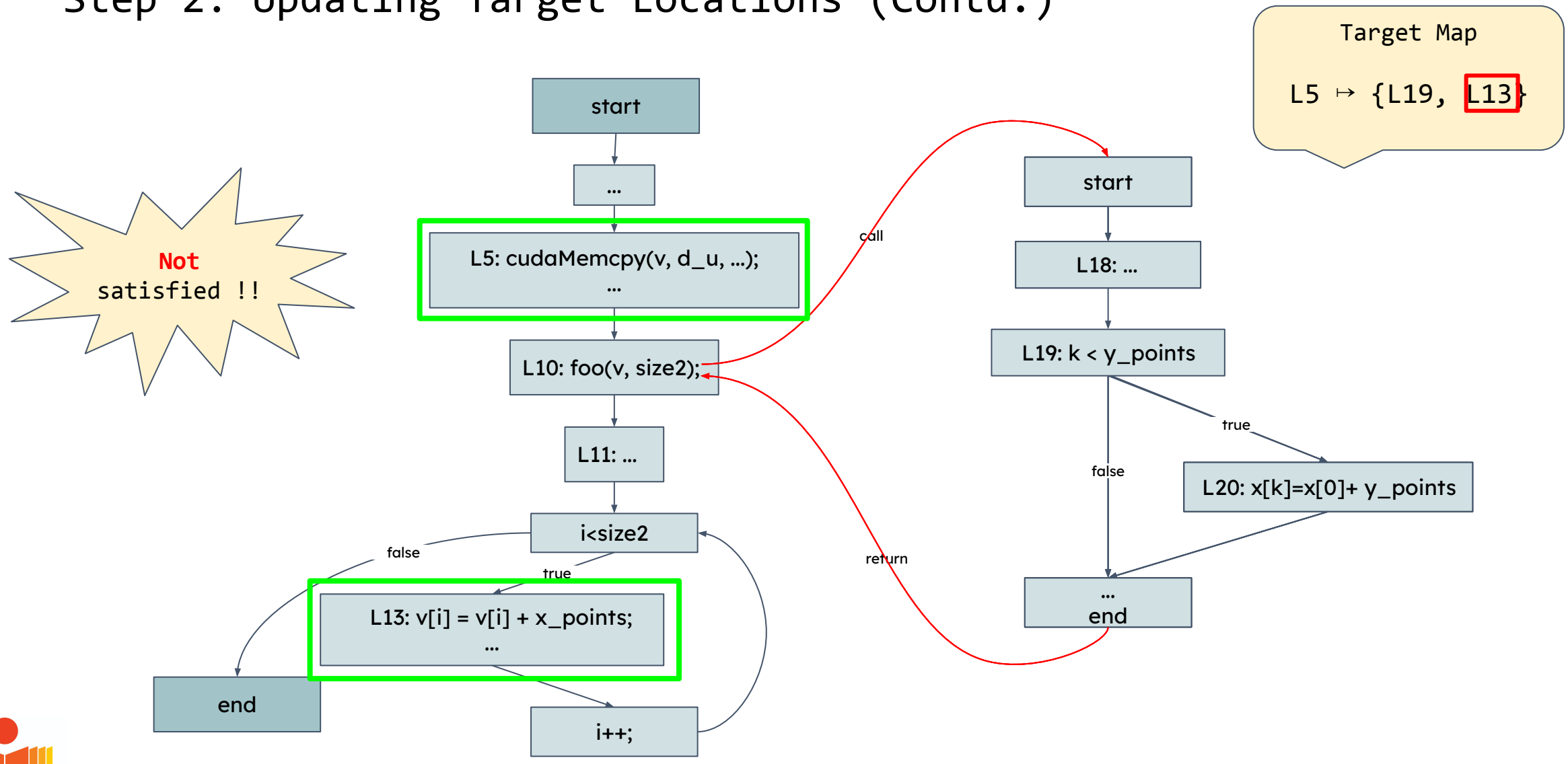
Step 2: Updating Target Locations (Contd.)



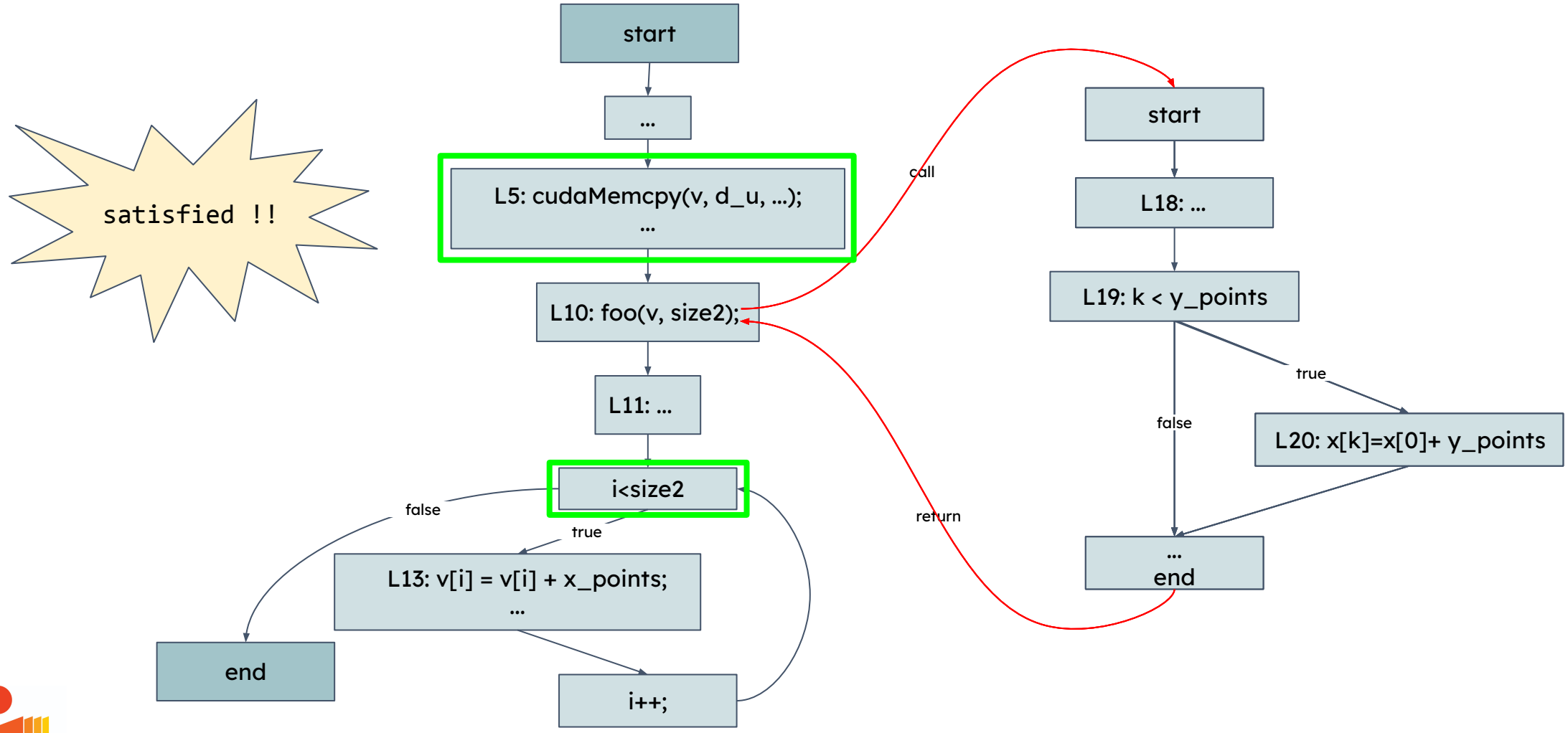
Step 2: Updating Target Locations (Contd.)



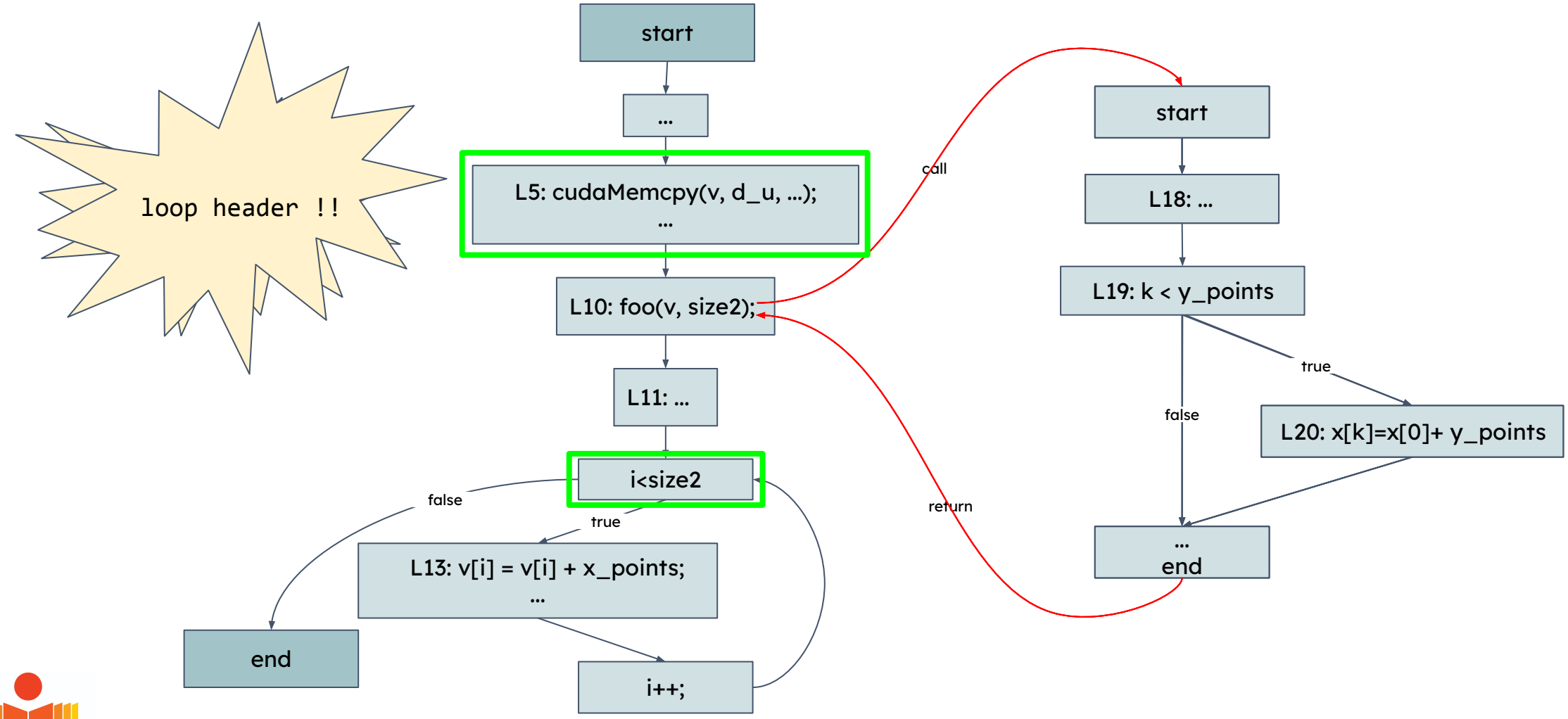
Step 2: Updating Target Locations (Contd.)



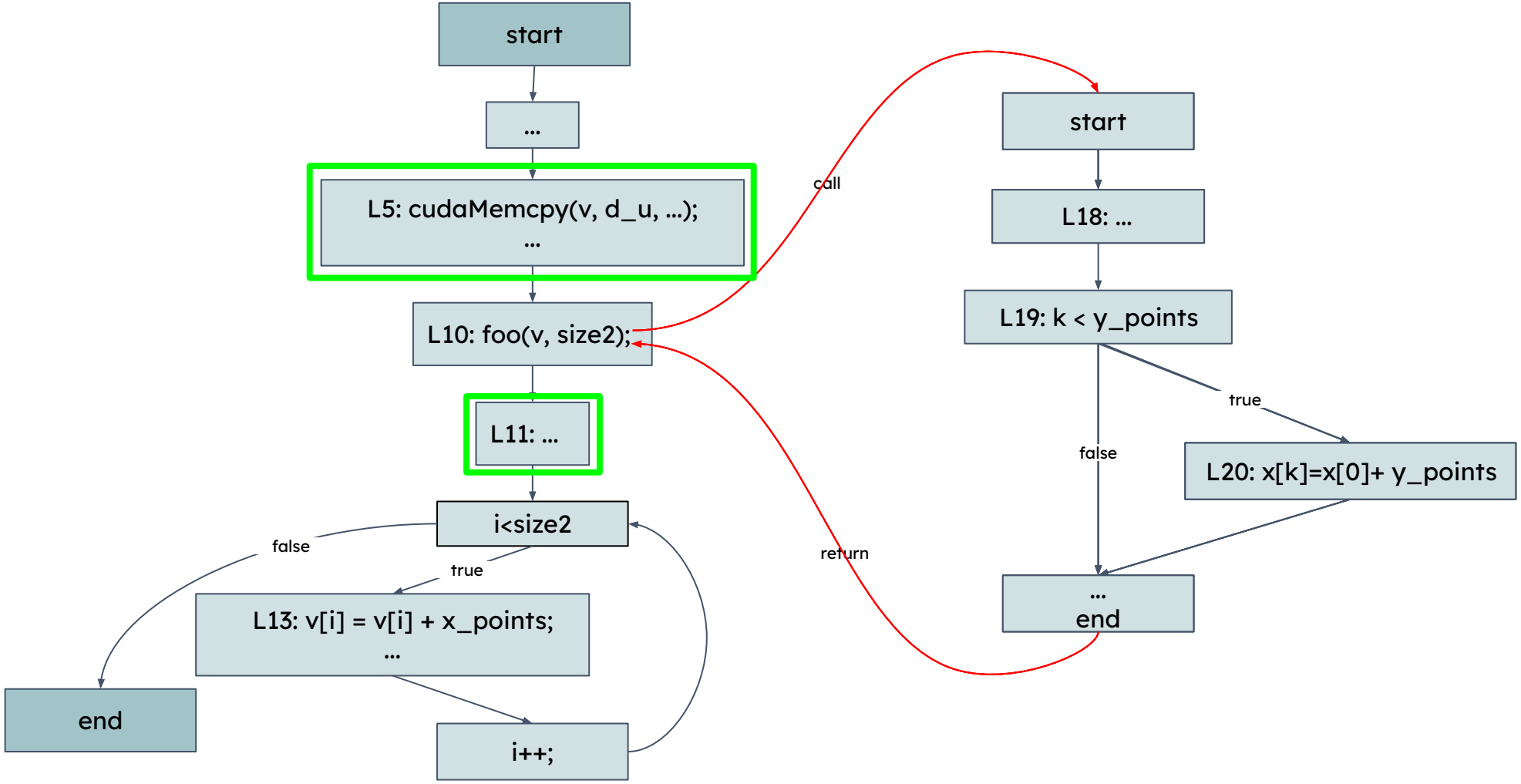
Step 2: Updating Target Locations (Contd.)



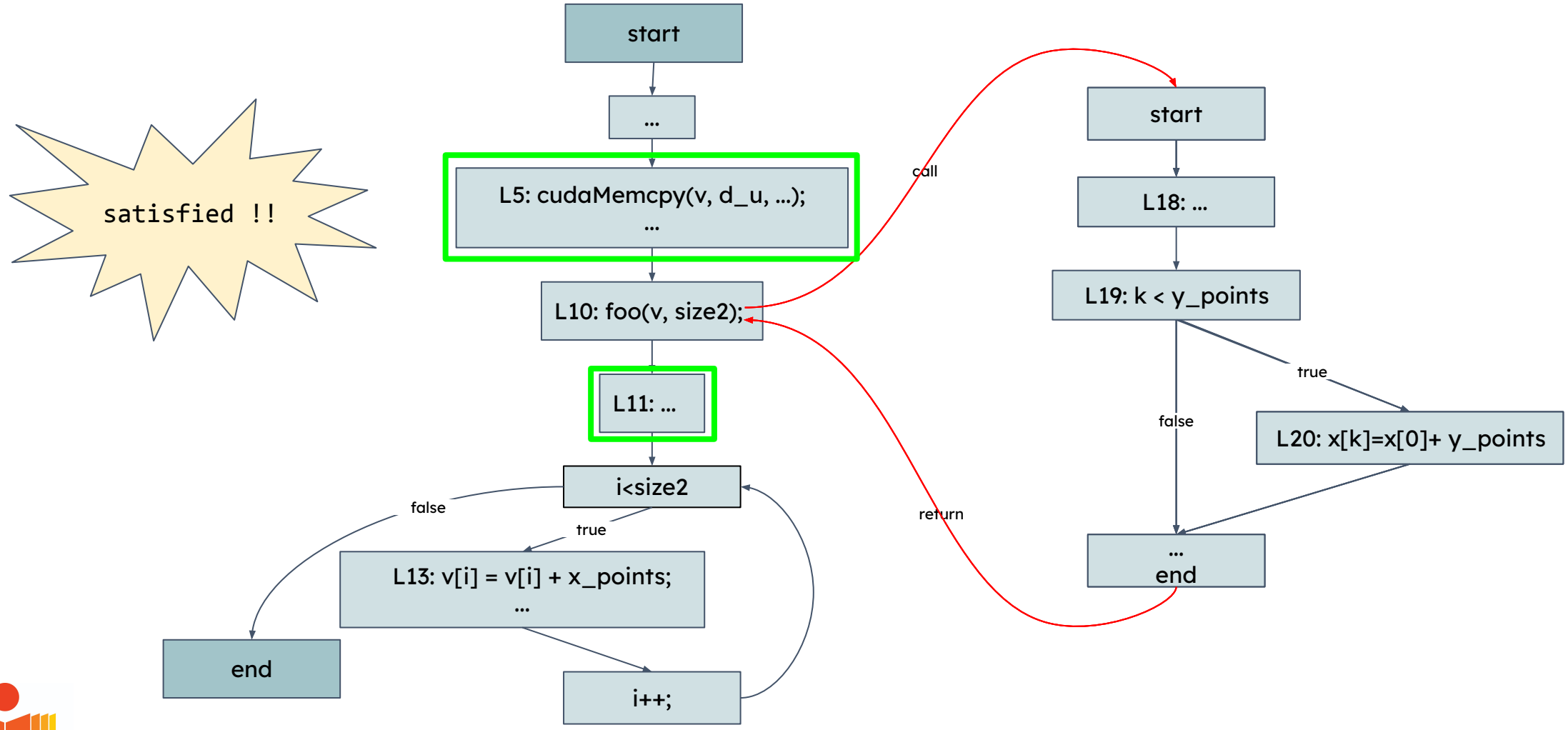
Step 2: Updating Target Locations (Contd.)



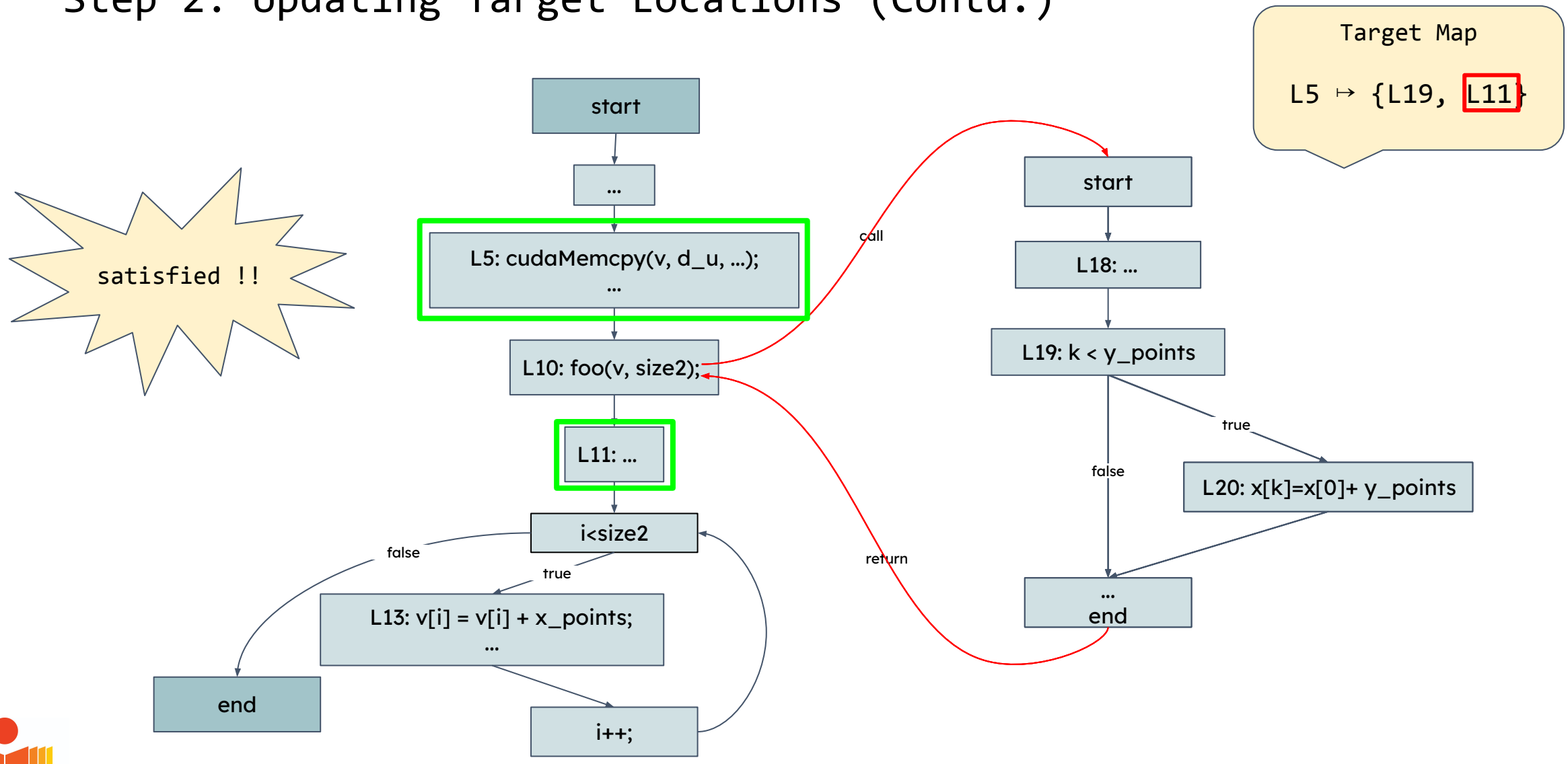
Step 2: Updating Target Locations (Contd.)



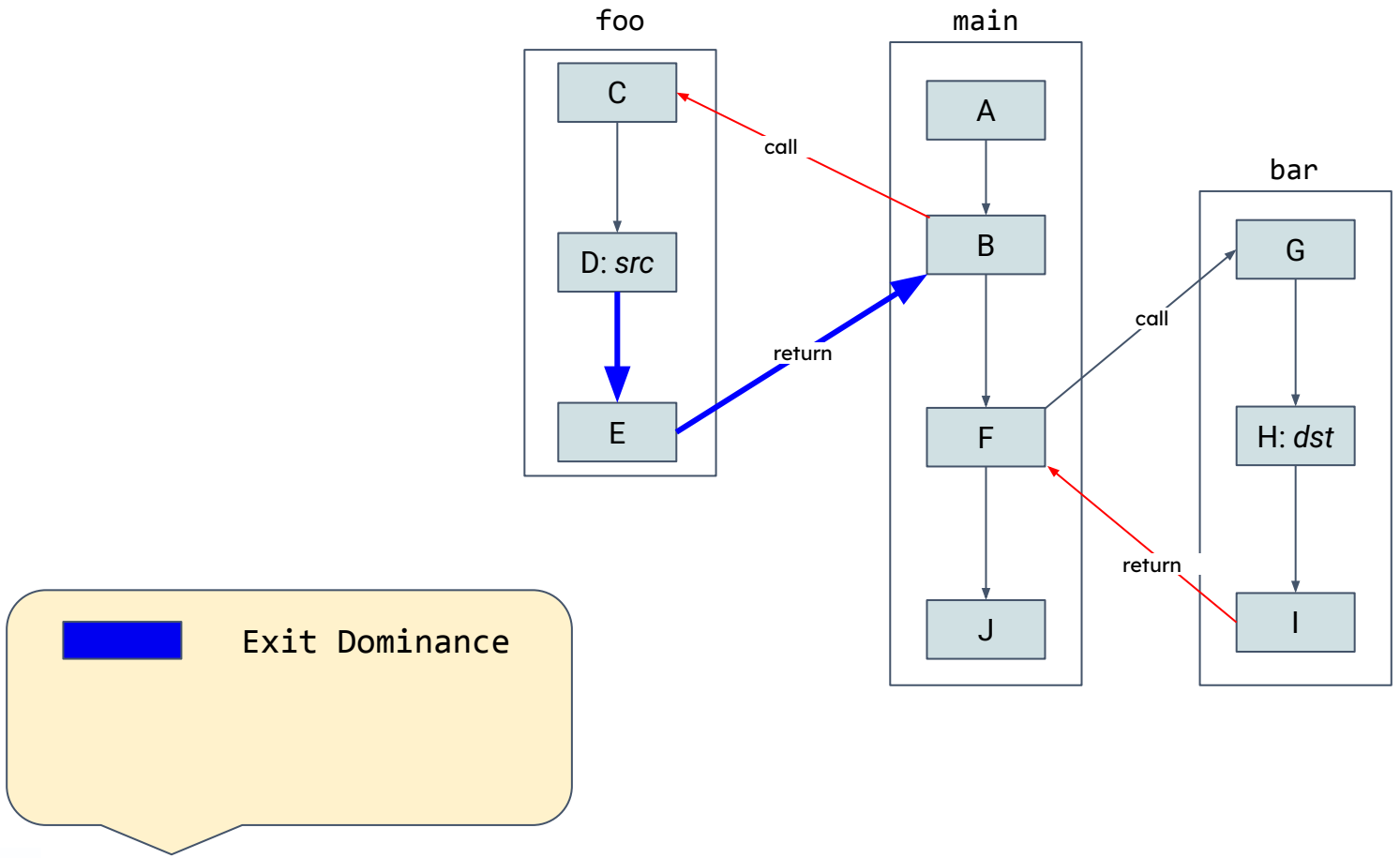
Step 2: Updating Target Locations (Contd.)



Step 2: Updating Target Locations (Contd.)

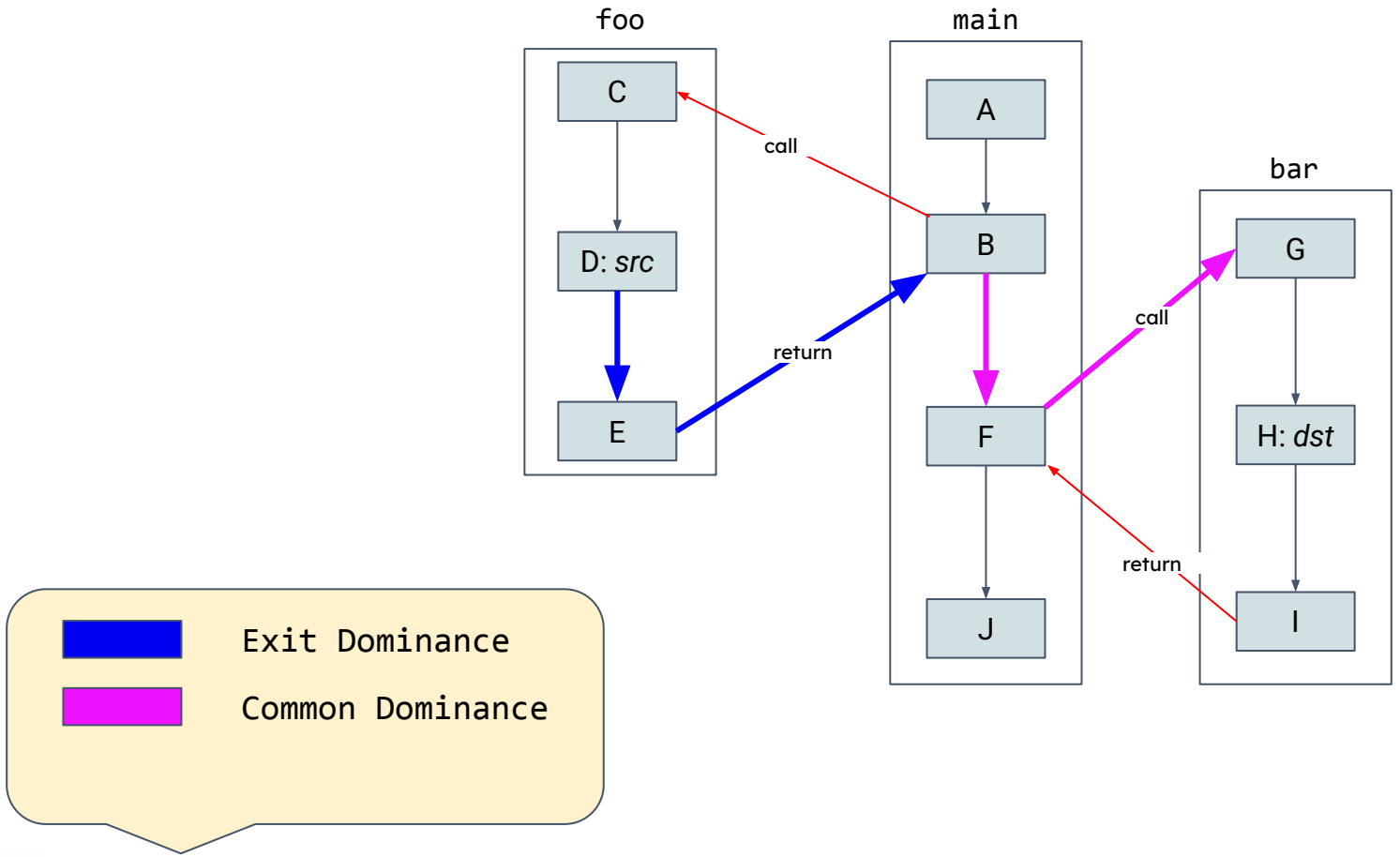


Step 2: Updating Target Locations: Inter-Procedural dom-pdom



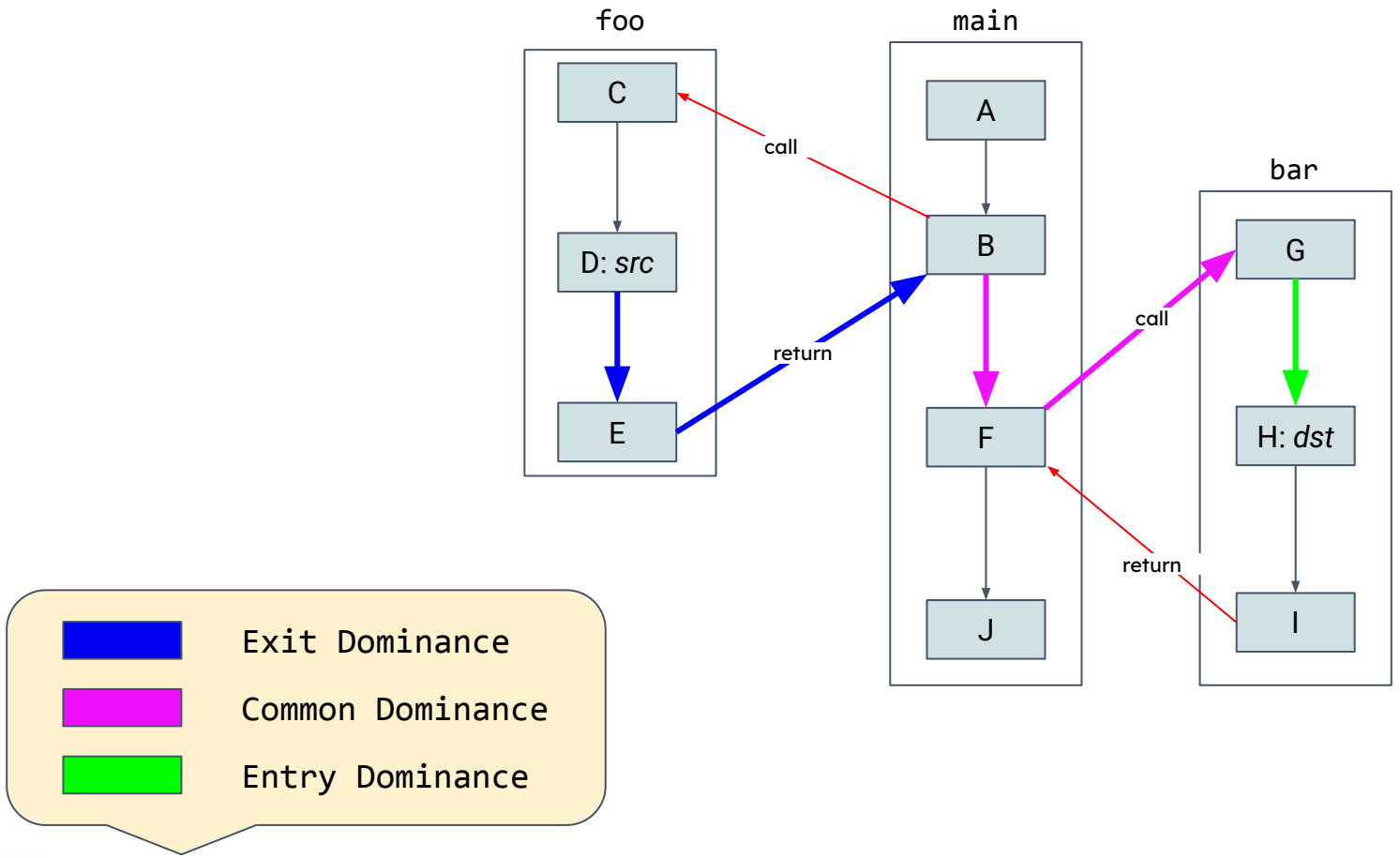
- Demand-driven inter-procedural dom-pdom analysis
- Demand-driven query is *<D:src, H:dst>*
- Based on transitivity of dom-pdom

Step 2: Updating Target Locations: Inter-Procedural dom-pdom



- Demand-driven inter-procedural dom-pdom analysis
- Demand-driven query is *<D:src, H:dst>*
- Based on transitivity of dom-pdom

Step 2: Updating Target Locations: Inter-Procedural dom-pdom

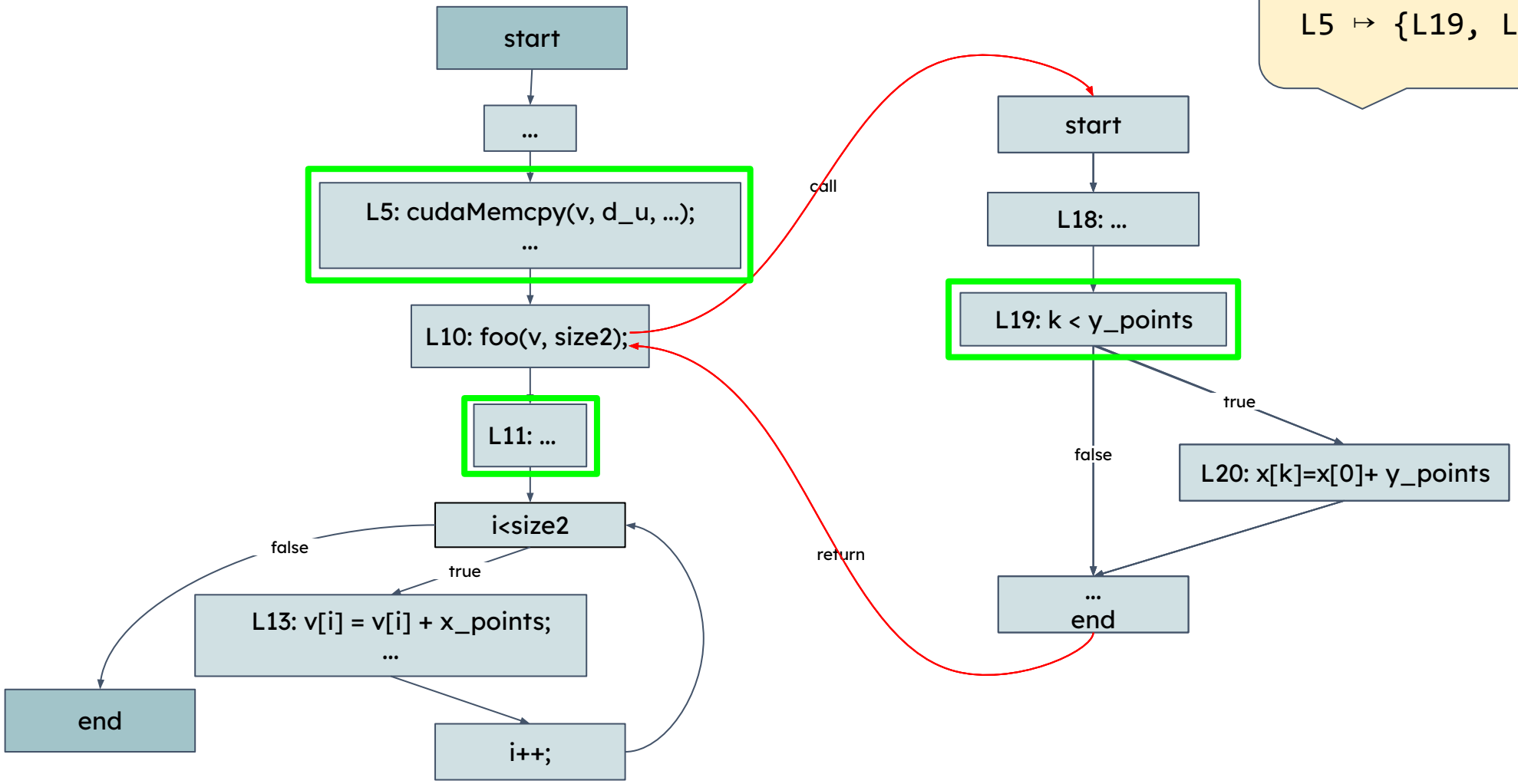


- Demand-driven inter-procedural dom-pdom analysis
- Demand-driven query is *<D:src, H:dst>*
- Based on transitivity of dom-pdom

Step 3: Unifying Target Locations: Motivation

Target Map

L5 ↦ {L19, L11}

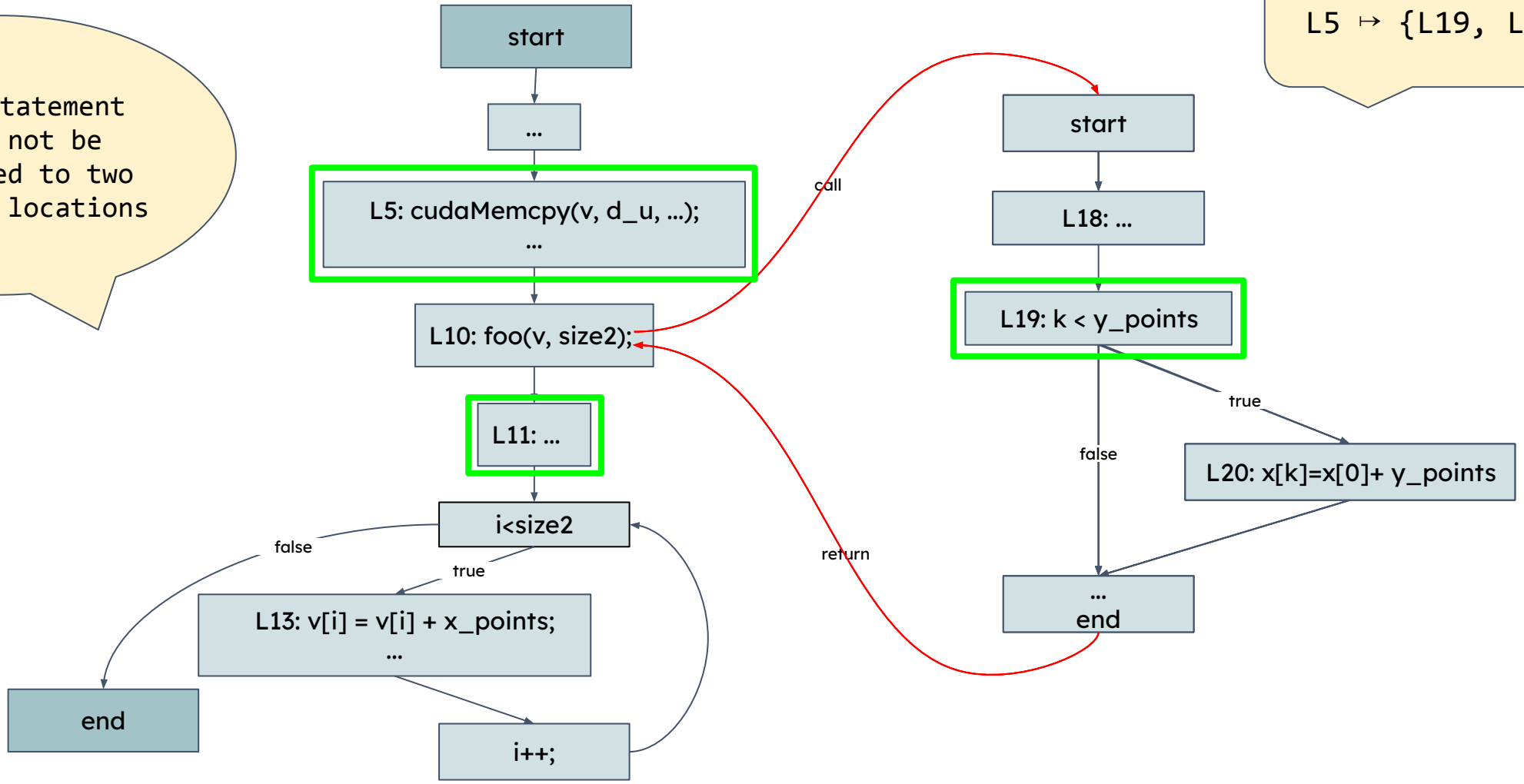


Step 3: Unifying Target Locations: Motivation

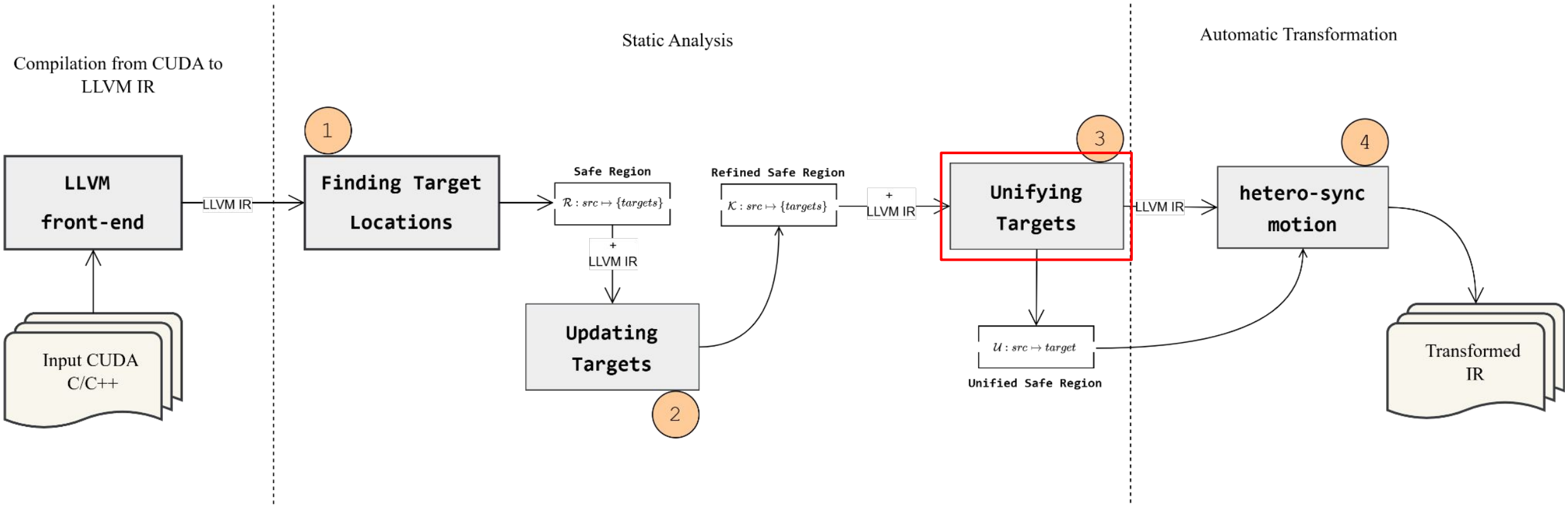
Memcpy statement should not be percolated to two different locations

Target Map

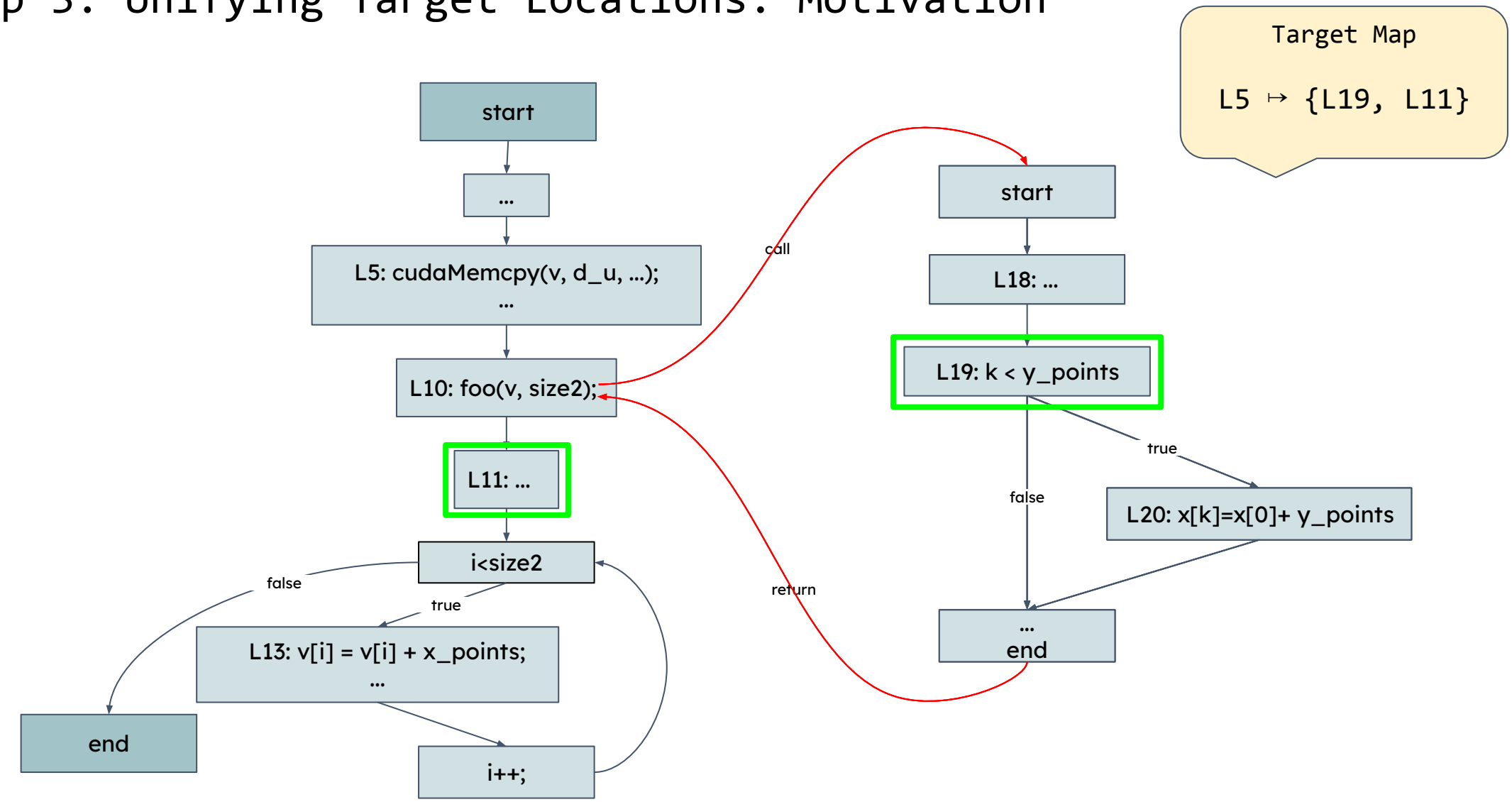
L5 ↦ {L19, L11}



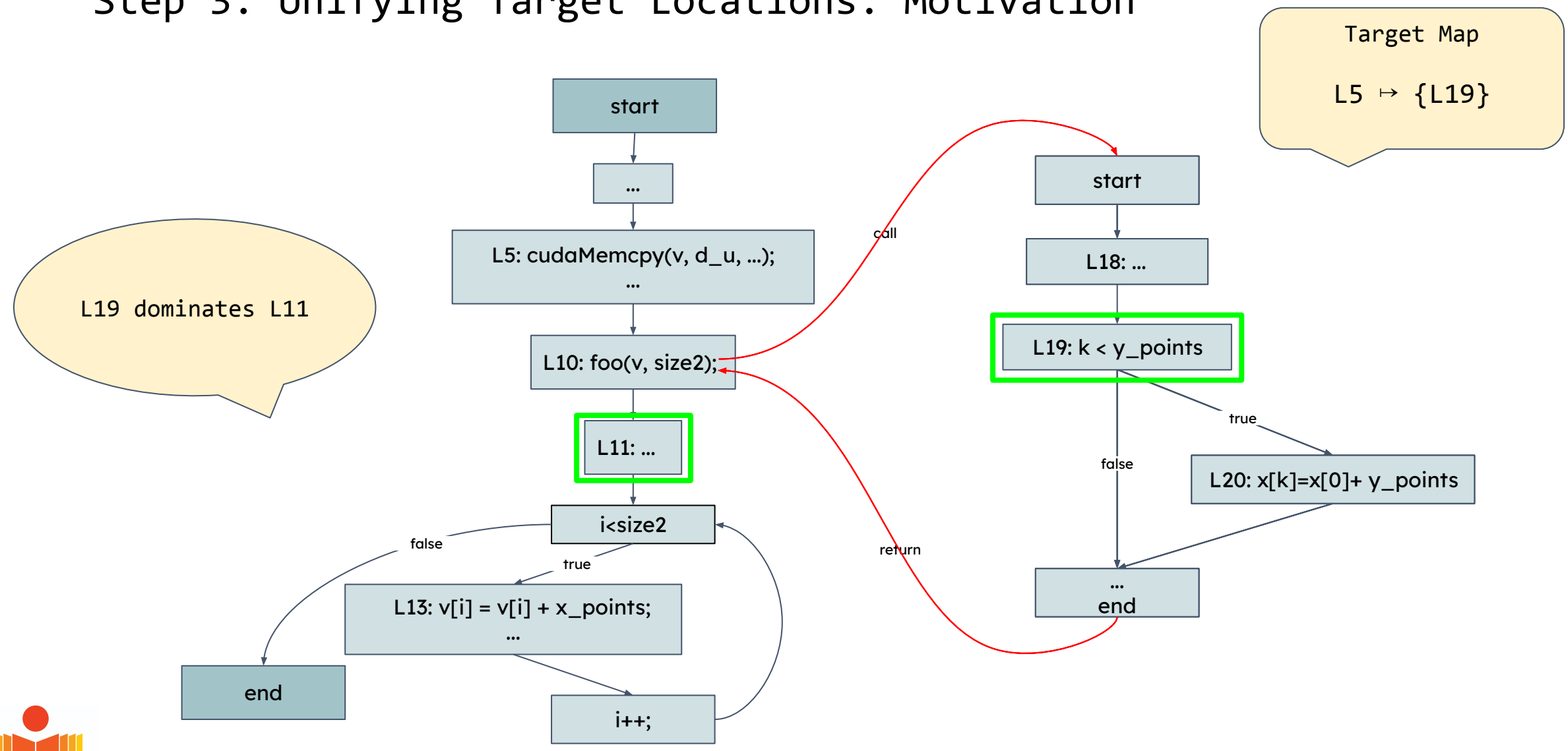
Step 3: Unifying Target Locations



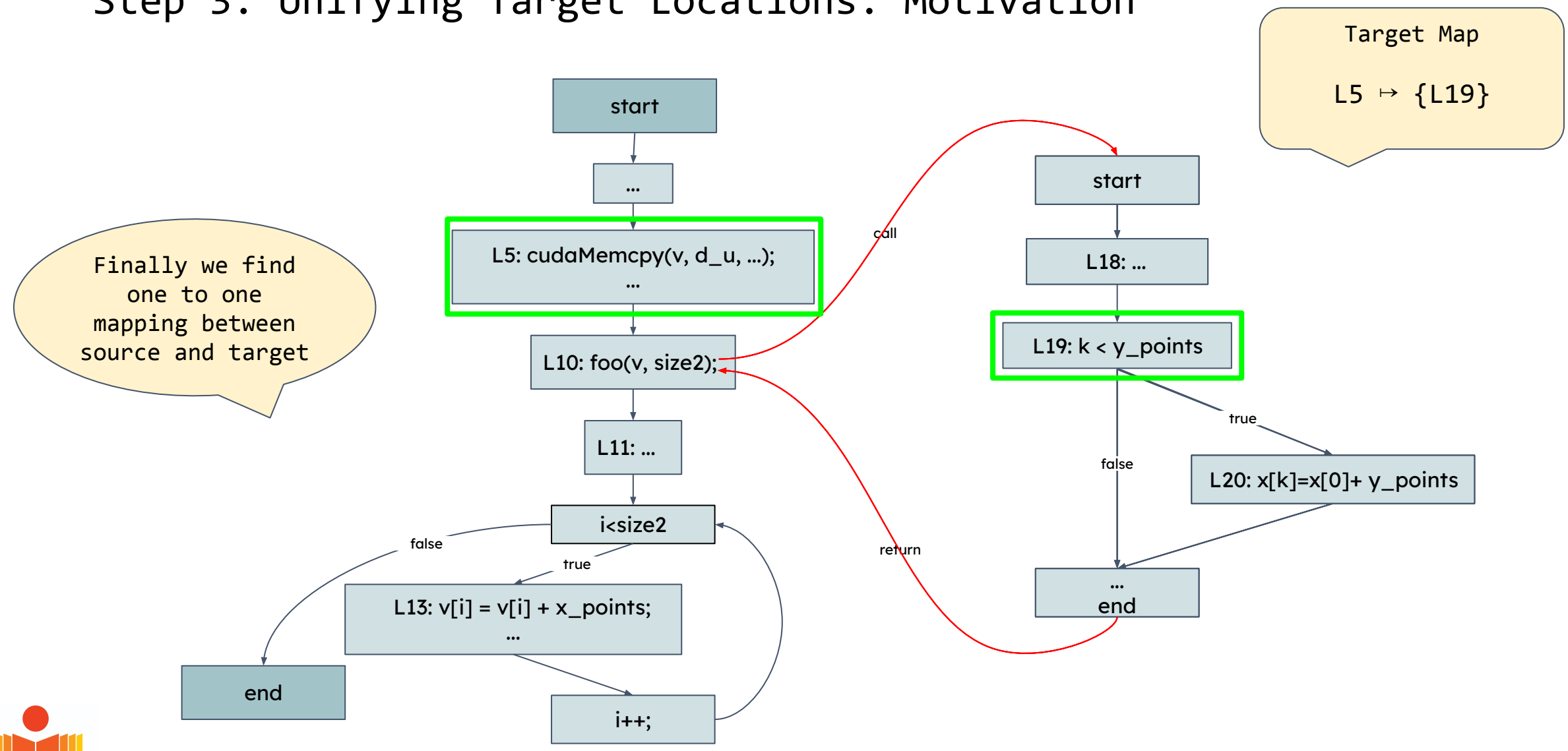
Step 3: Unifying Target Locations: Motivation



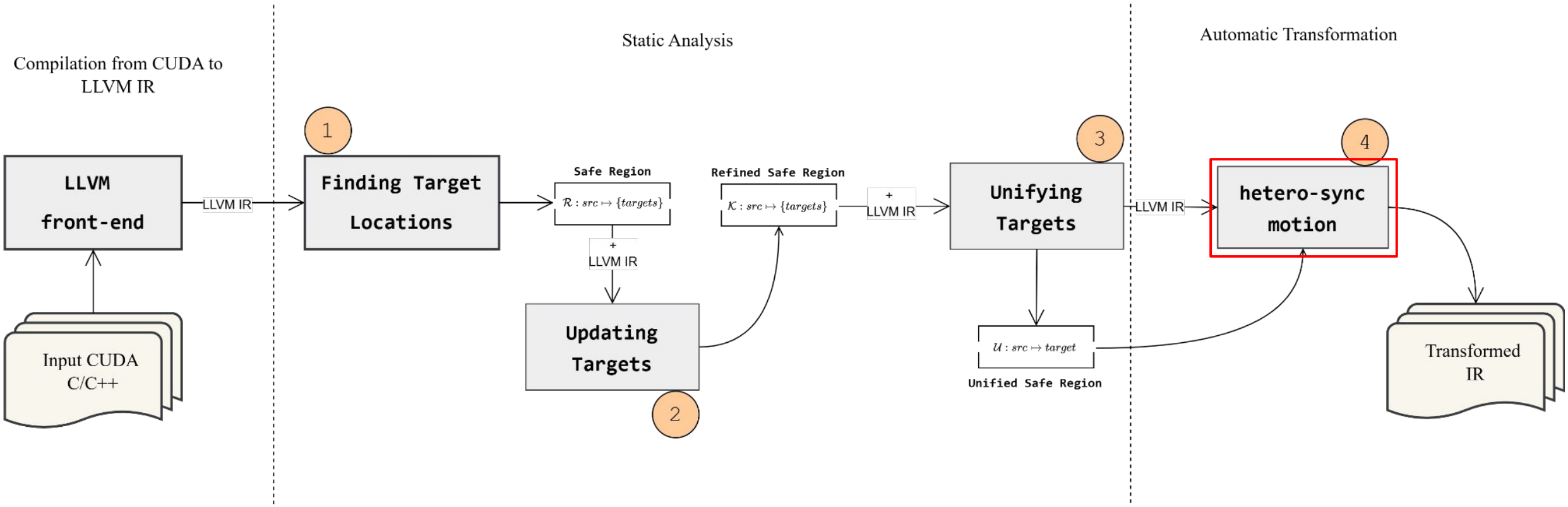
Step 3: Unifying Target Locations: Motivation



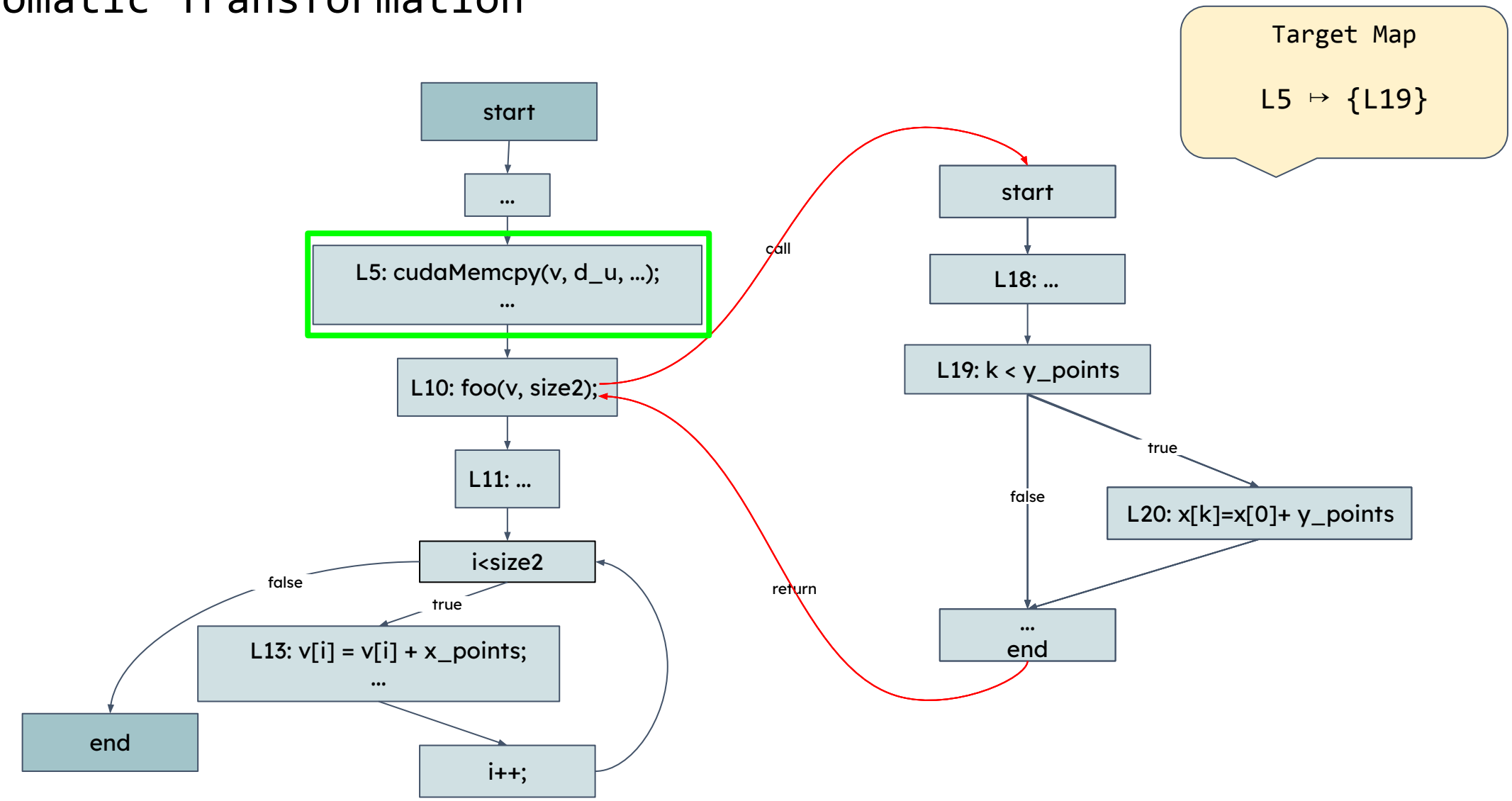
Step 3: Unifying Target Locations: Motivation



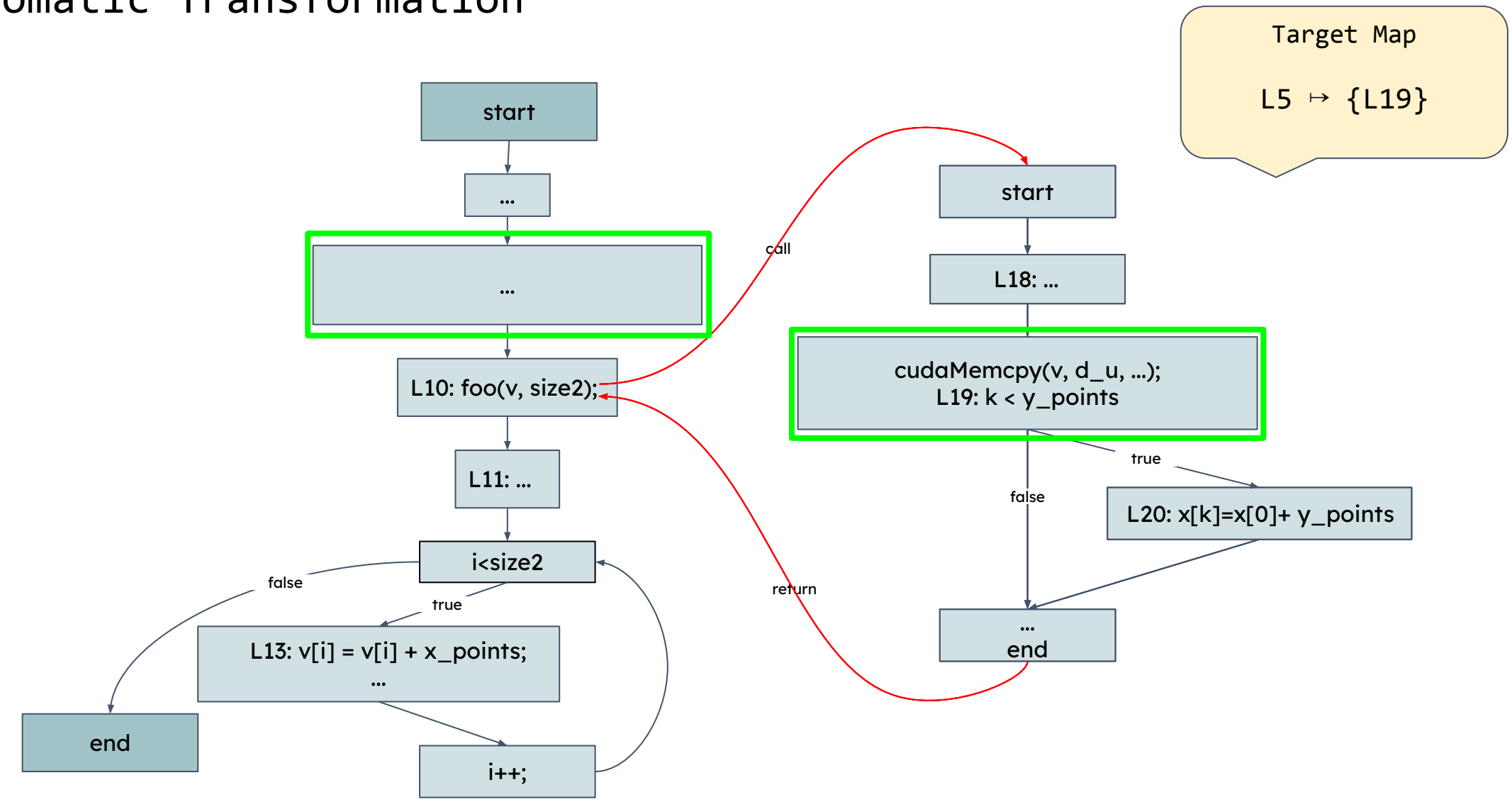
Automatic Transformation



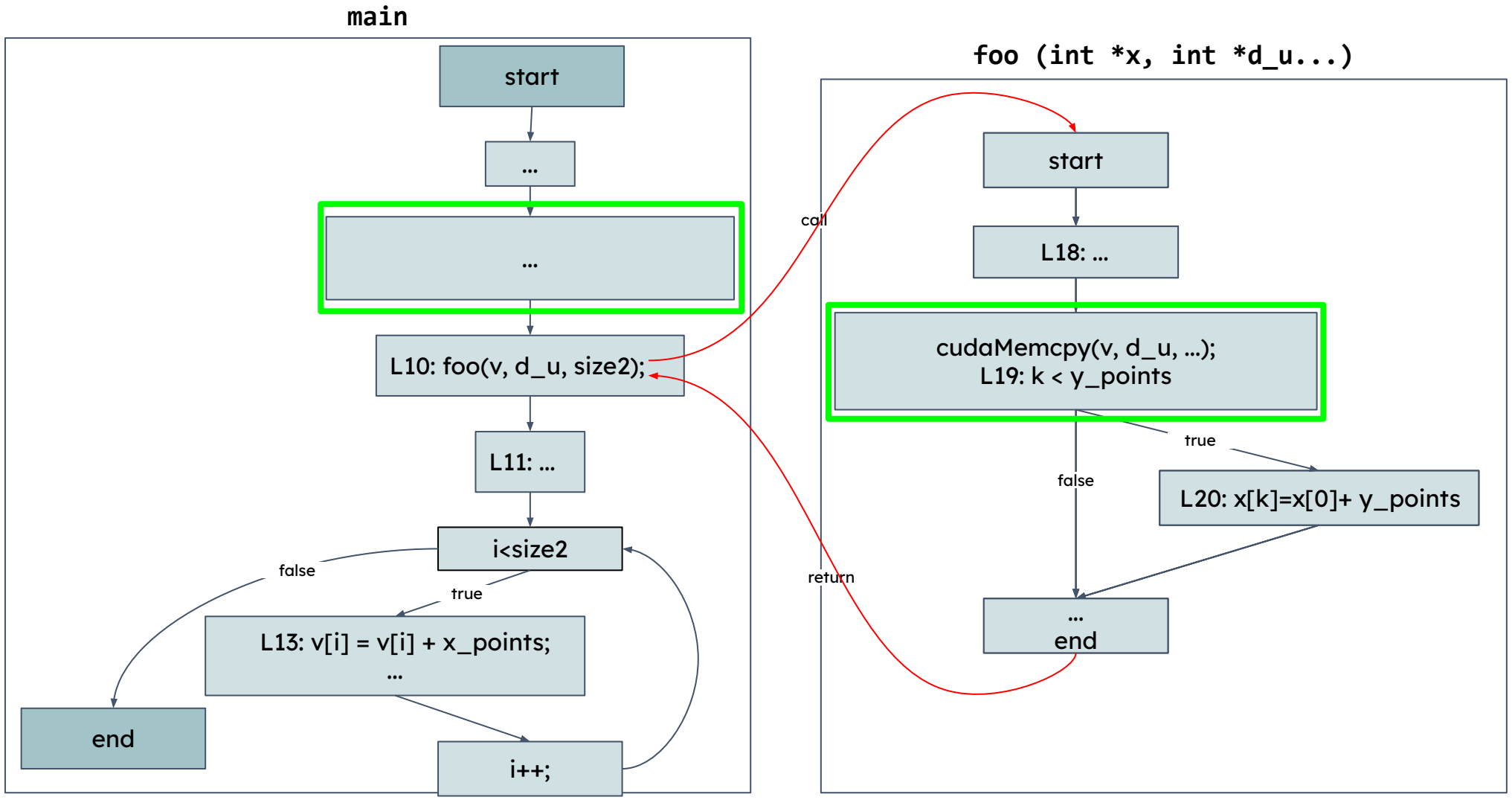
Automatic Transformation



Automatic Transformation



Automatic Transformation



Evaluation

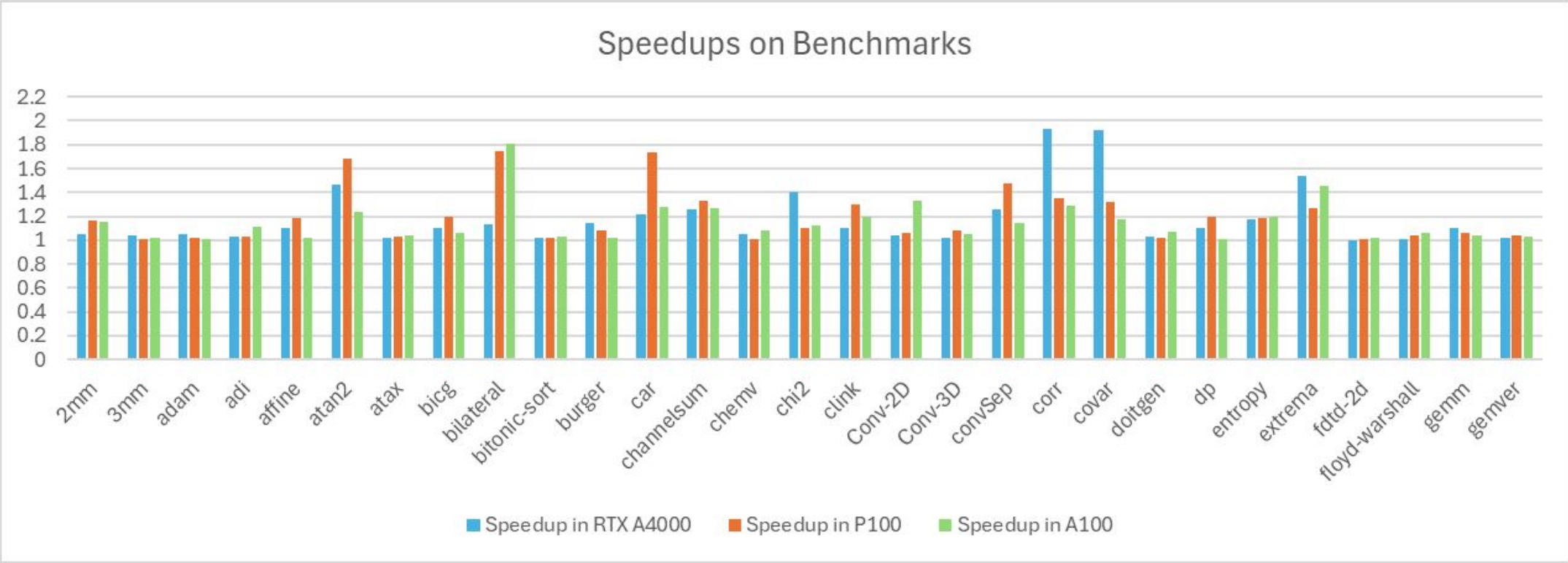
Evaluation: Experimental Setup

Component	Platform 1 *(RTX A4000)*	Platform 2 *(P100)*	Platform 3 *(A100)*
GPU (DRAM Size)	NVIDIA RTX A4000 (16 GB)	NVIDIA P100 (16 GB)	NVIDIA A100 (84 GB)
CPU	64 × AMD EPYC @ 2.20GHz	20 × Intel Xeon Silver @ 2.20GHz	64 × Intel Xeon Gold @ 3.50GHz
OS	Ubuntu 22.04 LTS	Debian GNU/Linux 11	Ubuntu 22.04 LTS
CPU RAM	128 GB	189 GB	84 GB

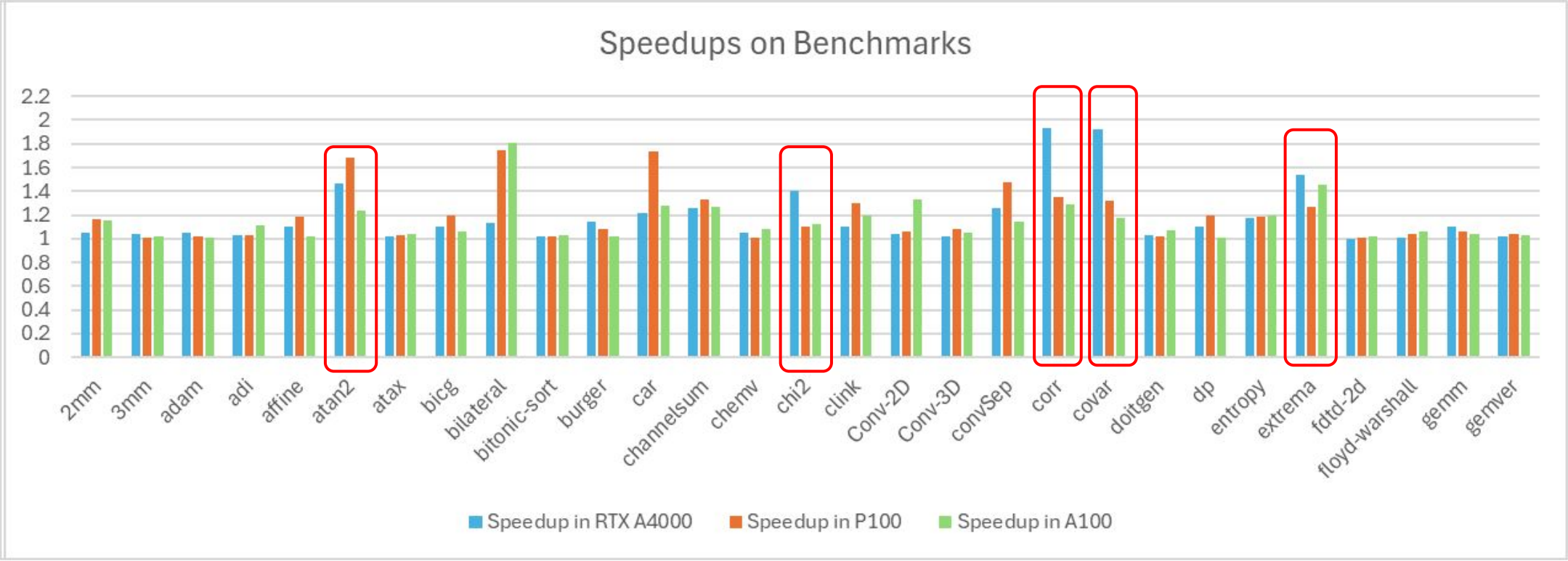
Evaluation: Efficiency of GSOHC

- Found **361** opportunities in **56** benchmark programs!!
- Around **48%** of the benchmarks have inter-procedural opportunities
- Benchmark LOC ranges from few hundreds to **18k**
- **Analysis time**
  - ranges from few milliseconds to 291 milliseconds
  - depends less on program size but more on control flow

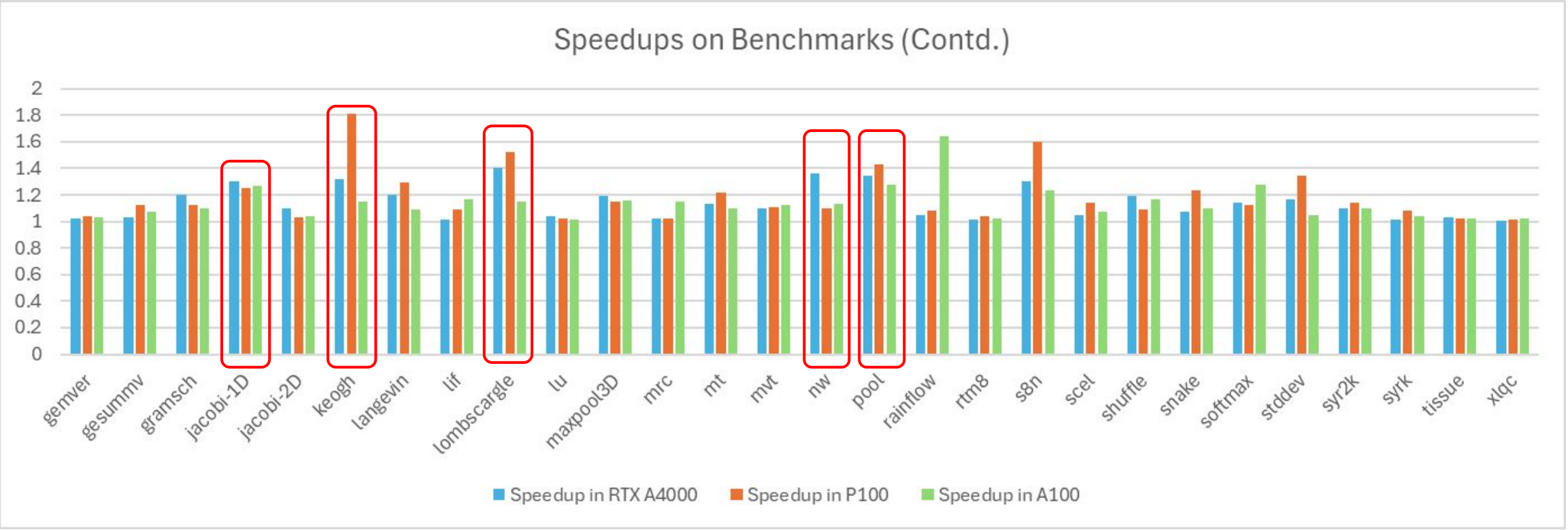
Evaluation: Effectiveness of GSOHC



Evaluation: Effectiveness of GSOHC

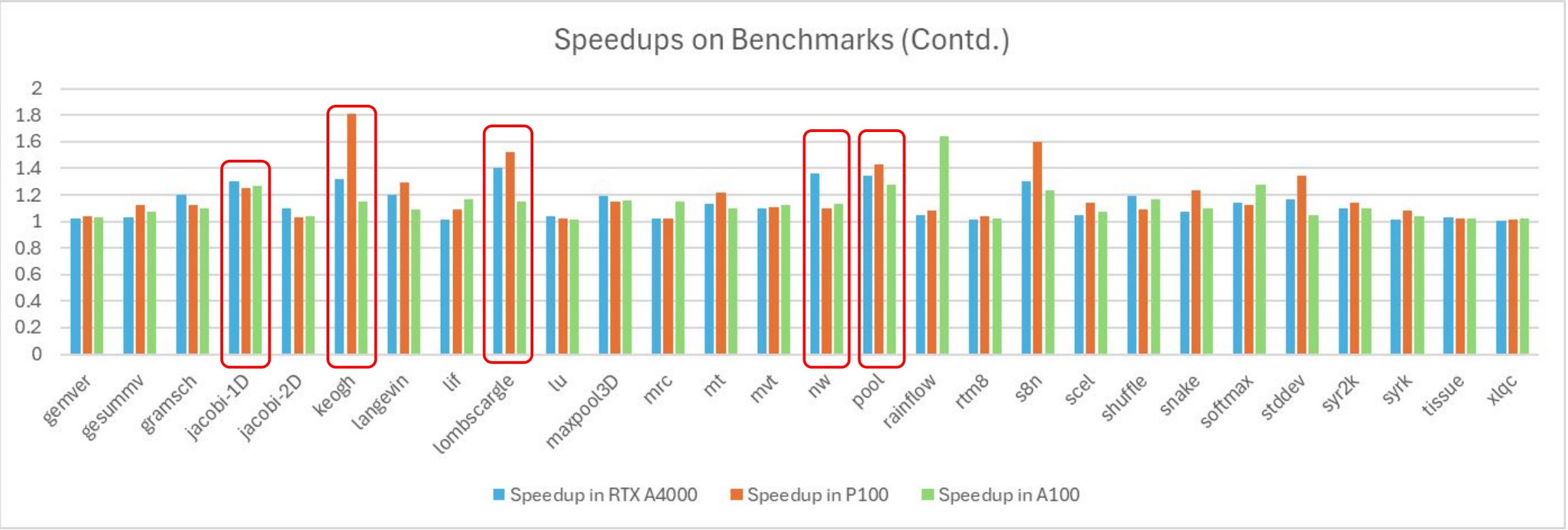


Evaluation: Effectiveness of GSOHC (Contd.)



भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

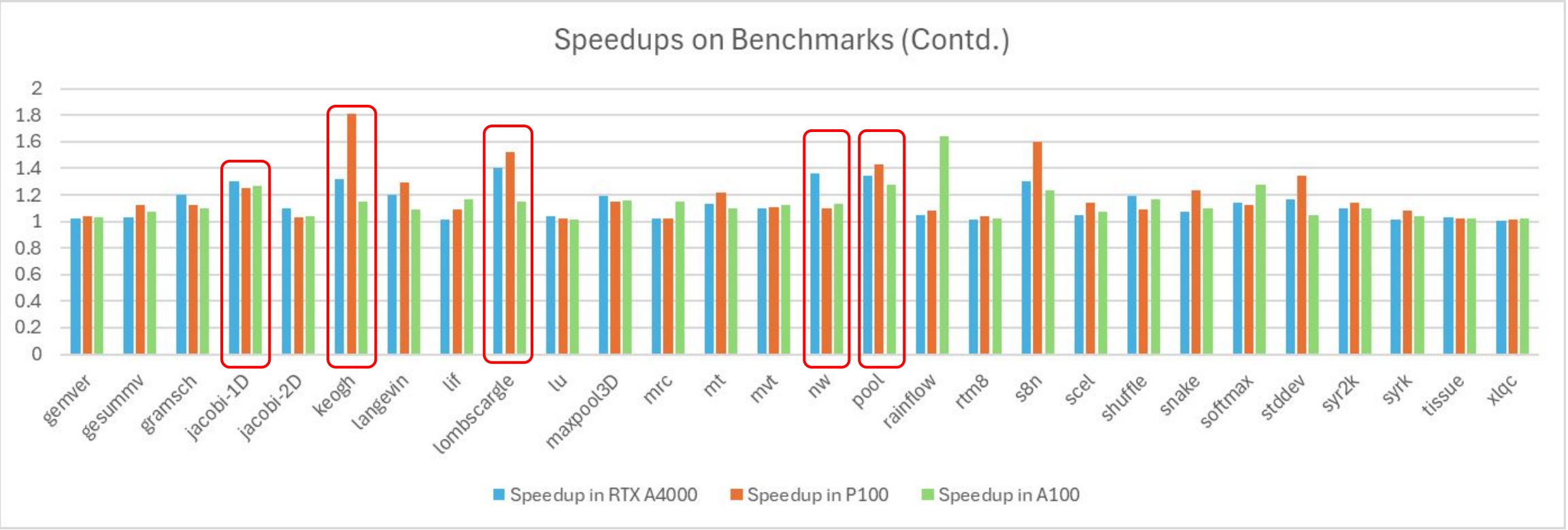
Evaluation: Effectiveness of GSOHC



Performance Improvement: Speedup upto 1.9x and 34% improvement with an average of 11%

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Evaluation: Effectiveness of GSOHC



Insight: The optimization yields best performance if the host-side computation time, and the device-side computation time, are significant and are comparable to one another.

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Summary

- **Hetero-Sync Motion:** Moves synchronization statements to improve CPU-GPU overlap → better performance.

- **GSOHC**
  - A novel compiler framework for optimizing CPU-GPU synchronization
  - 3-phased framework
  - Interprocedural, context- and flow-sensitive data-flow analysis

- **Comprehensive Evaluation**
  - Tested on 56 benchmark programs
  - Identified all optimization opportunities
  - Achieved up to 1.9× speedup and 34% improvement with an average of 11%
  -

Summary

- **Hetero-Sync Motion:** Moves synchronization statements to improve CPU-GPU overlap → better performance.
- **GSOHC**
  - A novel compiler framework for optimizing CPU-GPU synchronization
  - 3-phased framework
  - Interprocedural, context- and flow-sensitive data-flow analysis
- **Comprehensive Evaluation**
  - Tested on 56 benchmark programs
  - Identified all optimization opportunities
  - Achieved up to 1.9× speedup

Webpage

Artifact

Thank You!!

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

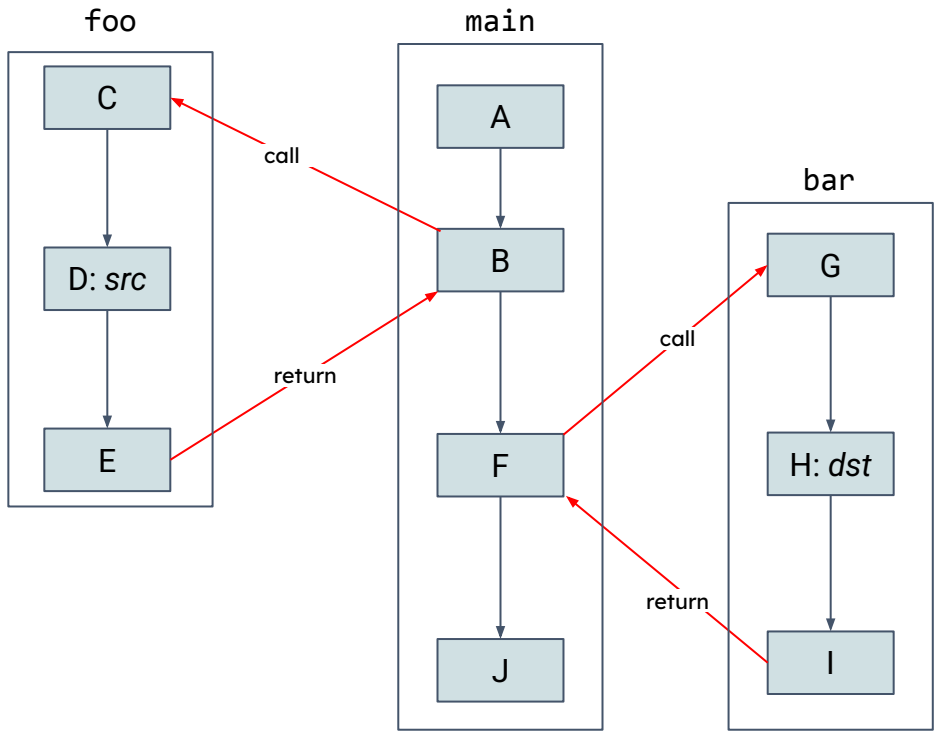
Webpage

LinkedIn

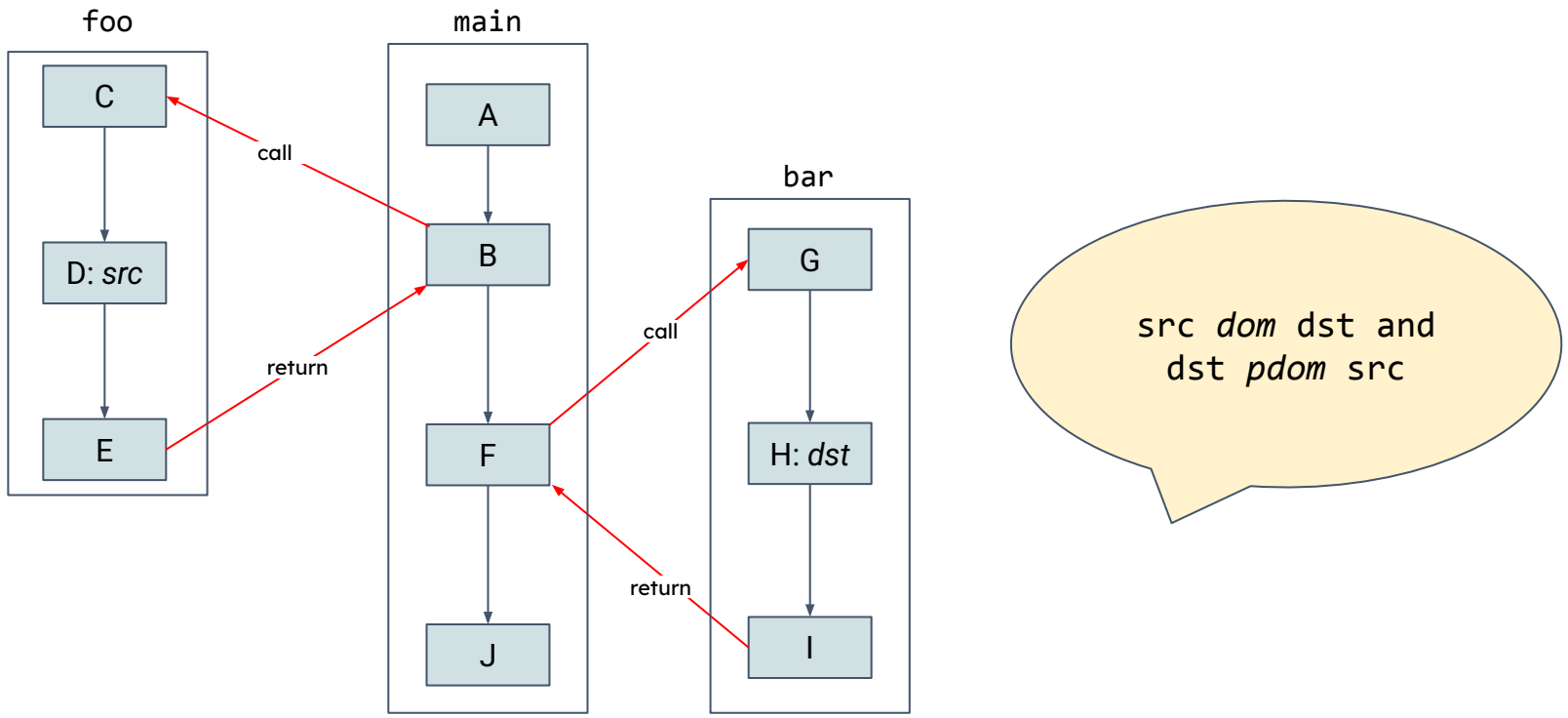
Artifact

Thank You

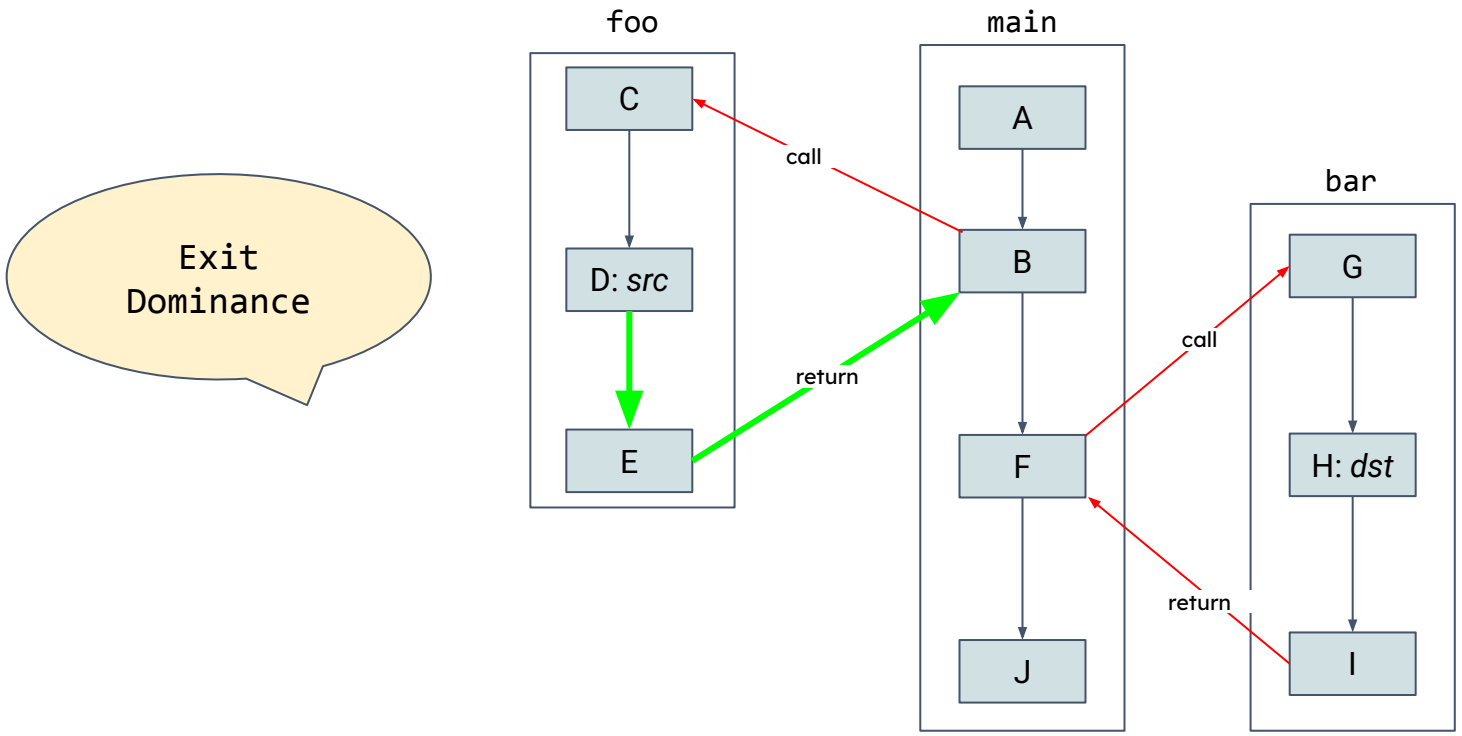
Step 2: Updating Target Locations: Inter-Procedural dom - pdom



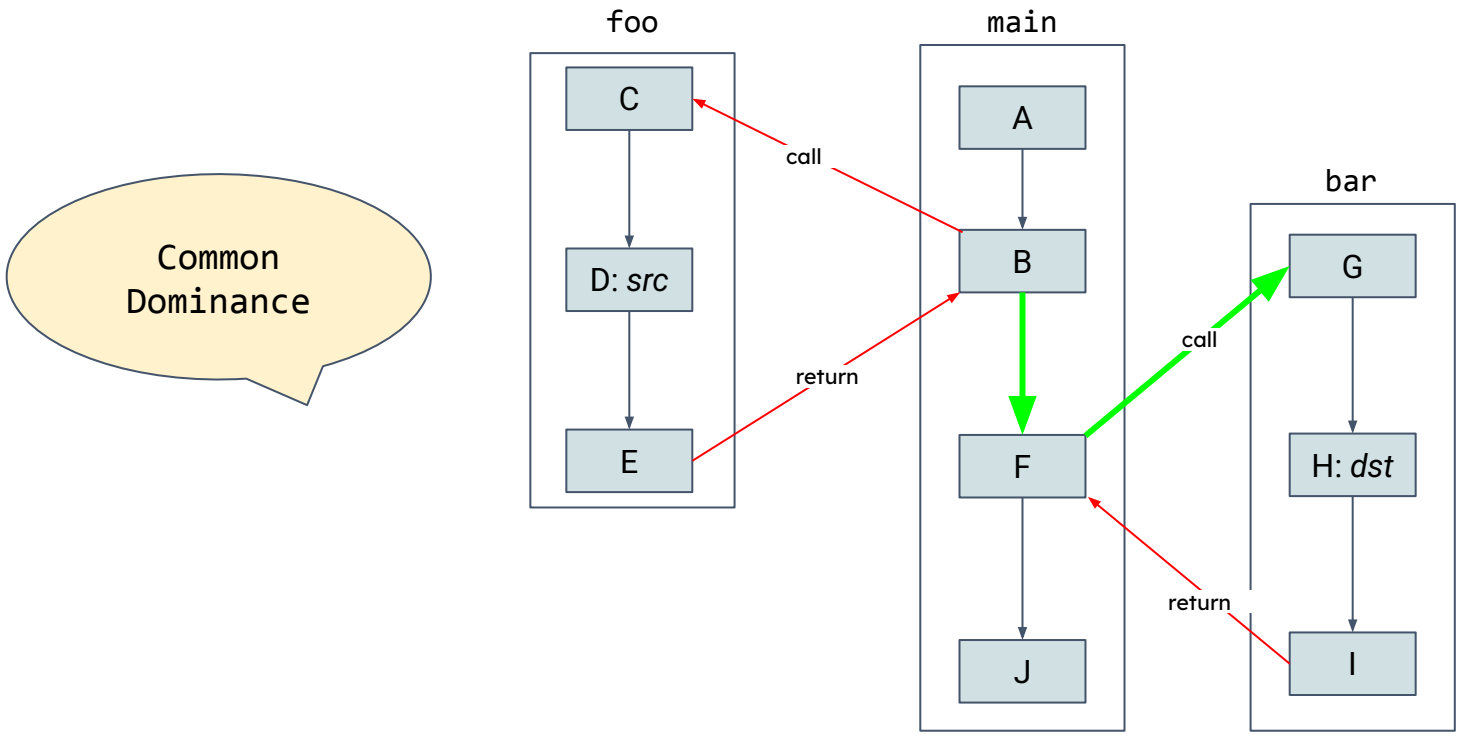
Step 2: Updating Target Locations: Inter-Procedural dom - pdom



Step 2: Updating Target Locations: Inter-Procedural dom - pdom



Step 2: Updating Target Locations: Inter-Procedural dom - pdom



Step 2: Updating Target Locations: Inter-Procedural dom - pdom