COL380

Introduction to
Parallel & Distributed Programming

Programming Models

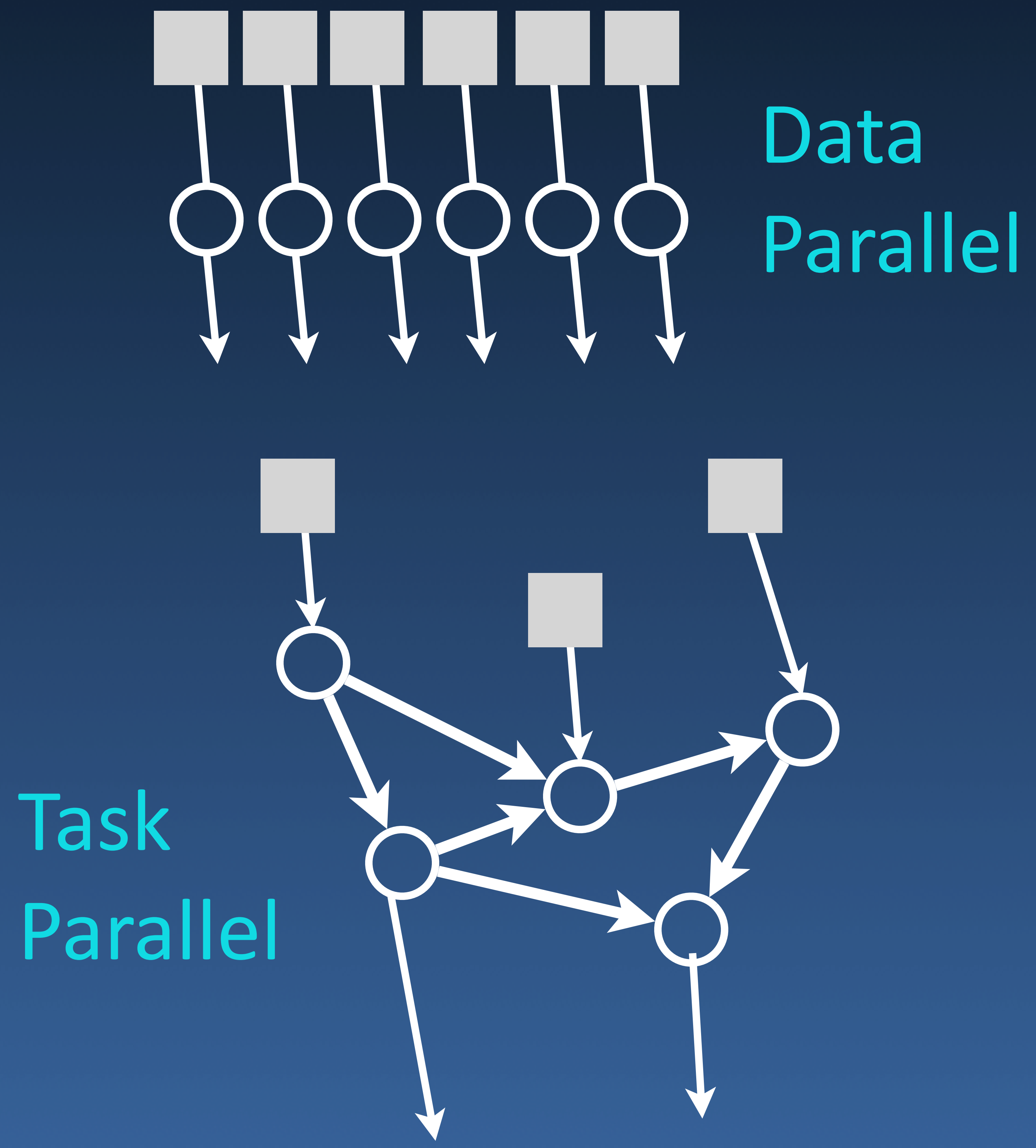
- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

Programming Models

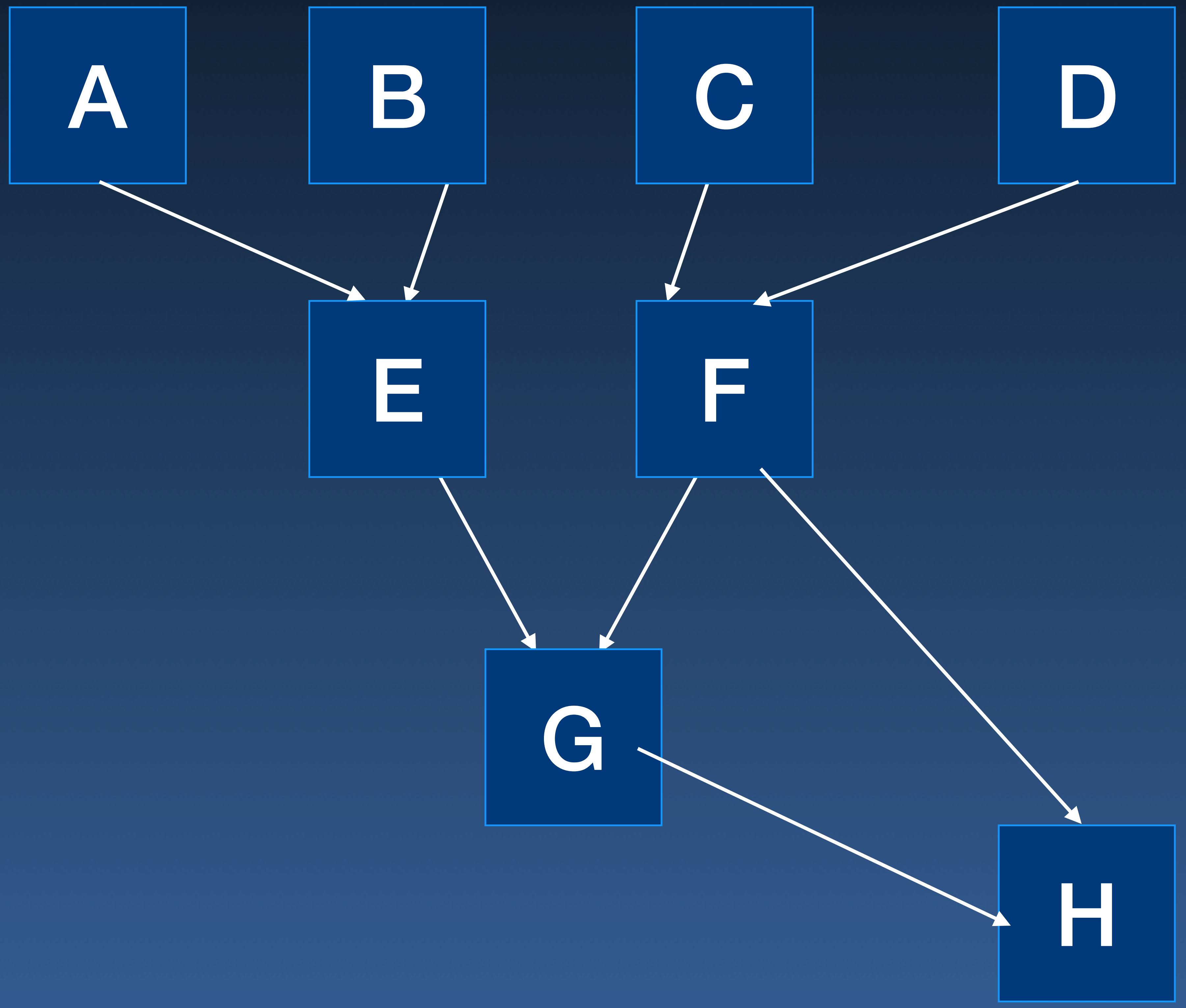
- Shared Memory
  - ➡ Tasks share a common address space they access asynchronously
  - ➡ Synchronization used to control access to the shared memory
  - ➡ Data may be cached on the processor that works on it
  - ➡ Compiler translates user variables into “global” memory addresses
- Message Passing
  - ➡ Tasks use their own local memory
  - ➡ Data transfer usually requires cooperation: send matched by a receive

Task Decomposition

- Data Parallel
  - ➡ Perform $f(x)$ for many x
- Task Parallel
  - ➡ Perform many functions f_i
- Pipeline



Task Graph



Granularity

Critical Path Length

Maximum Concurrency

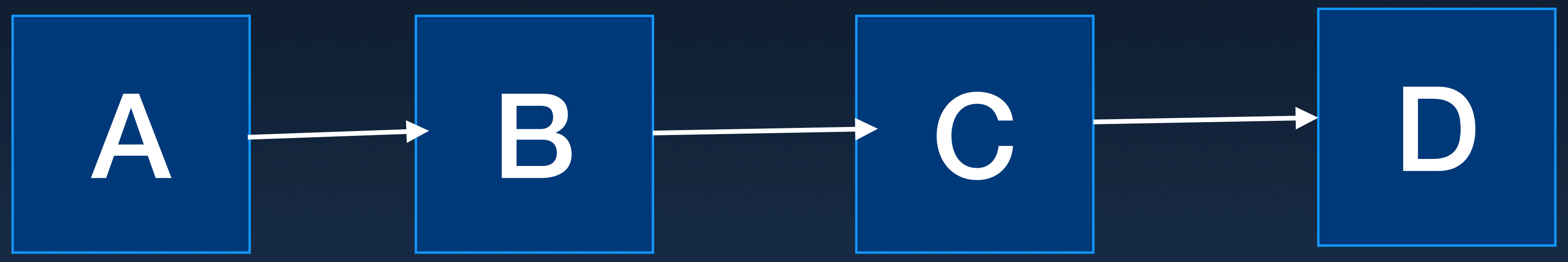
Average Concurrency

$$= \frac{\textbf{\#tasks}}{\textbf{Critical path length}}$$

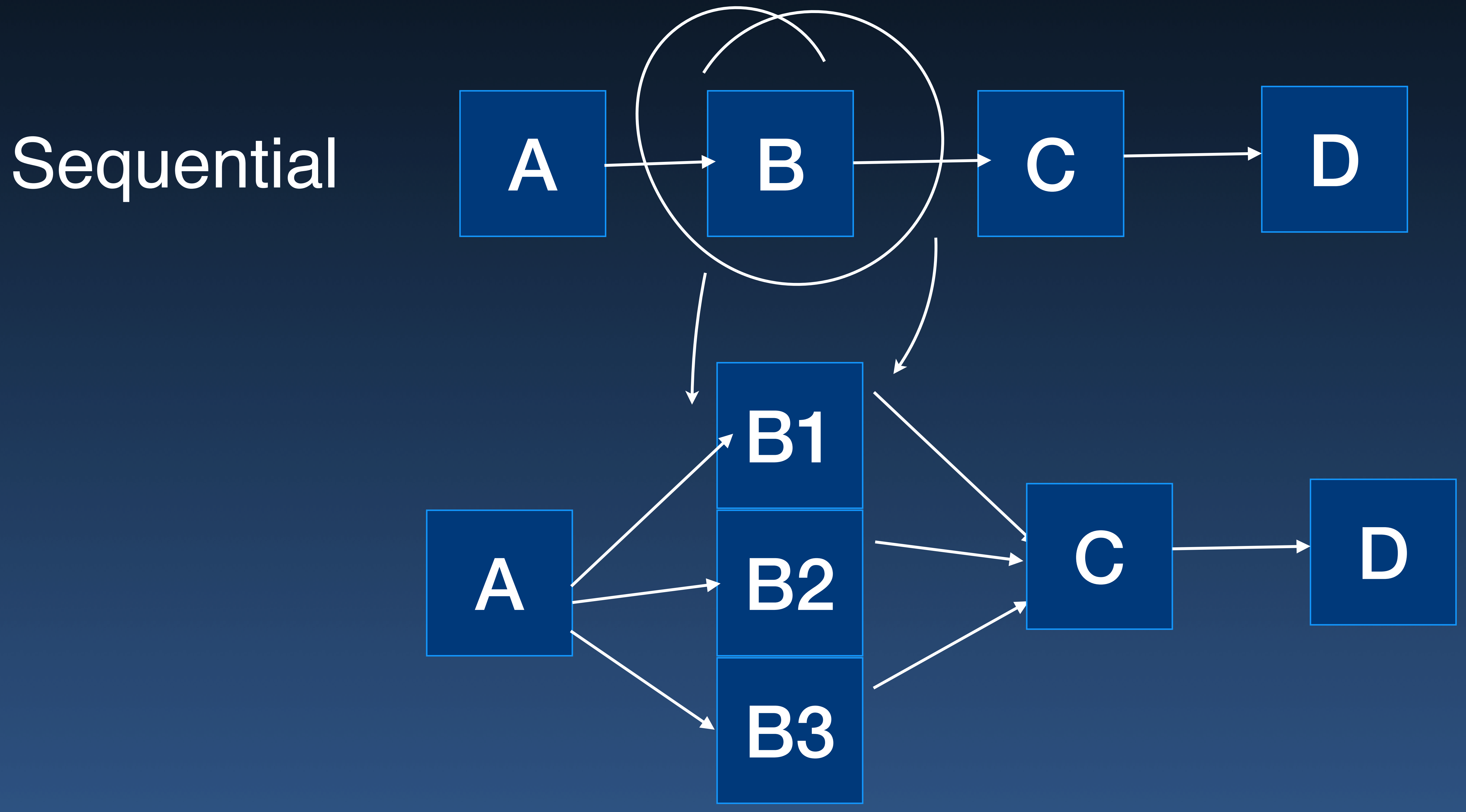
Subodh Kumar

Task Graph

Sequential



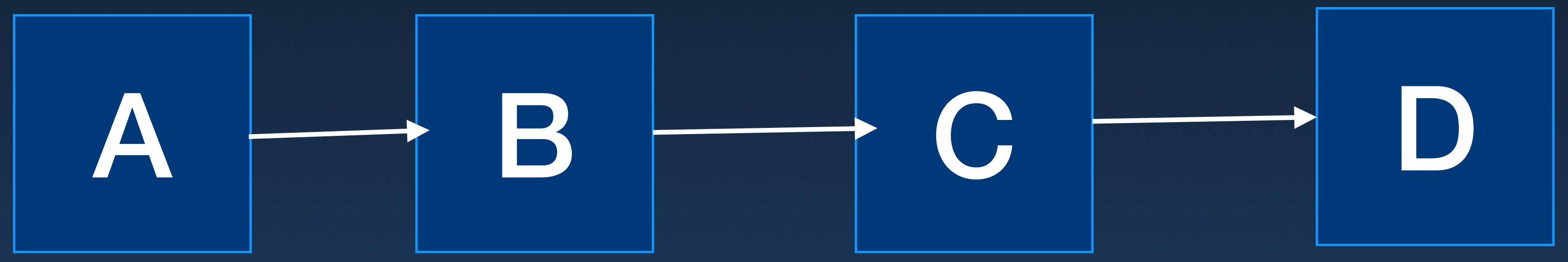
Task Graph



Can be at different levels of detail

Task Graph

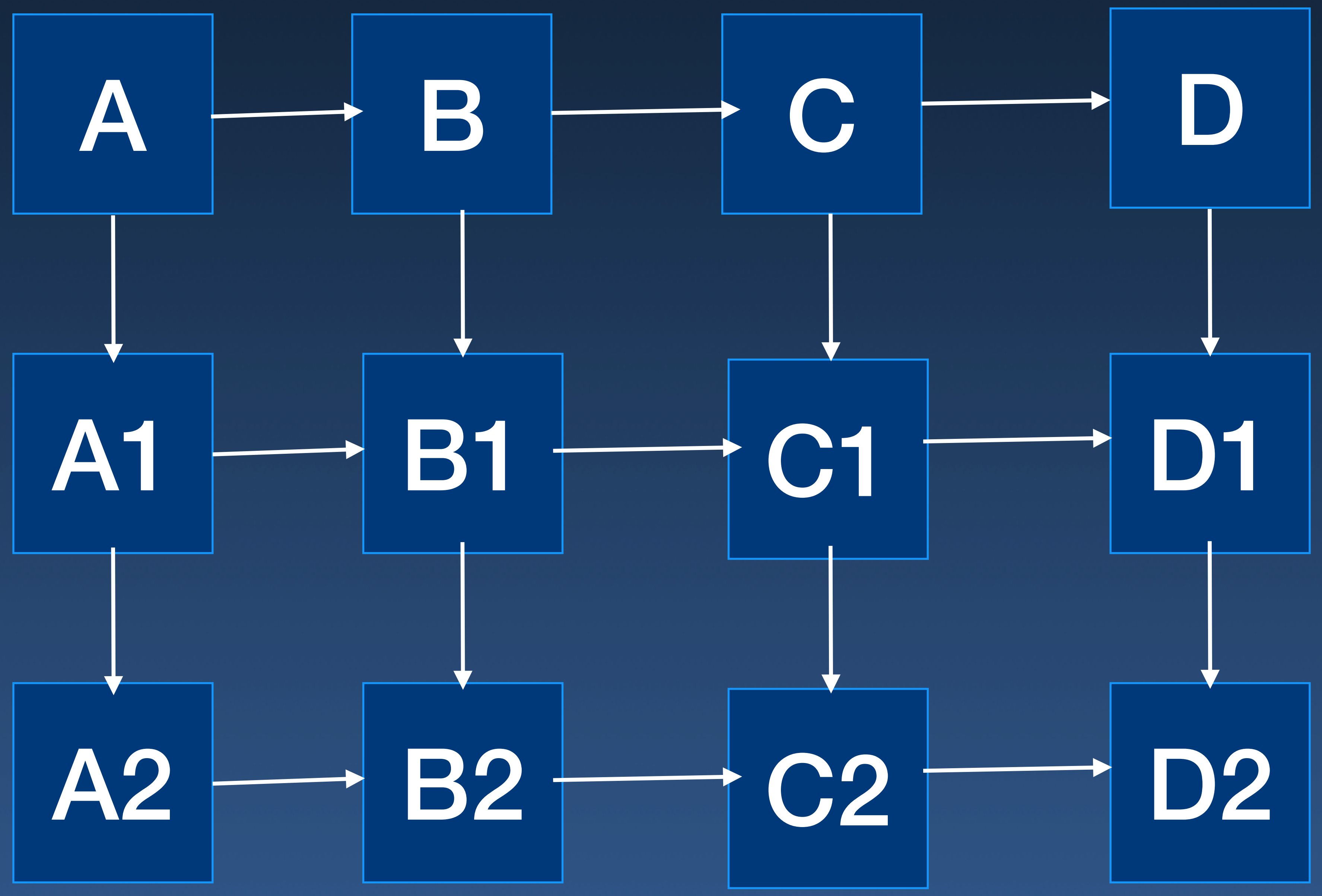
Pipeline



Subodh Kumar

Task Graph

Pipeline



Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream proces
- Map-reduce m
- Client-server model

```
cudaGraph_t graph;
cudaGraphExec_t instance;
cudaStreamBeginCapture(stream, cudaStreamCaptureModeGlobal);
…
cudaStreamEndCapture(stream, &graph);
cudaGraphInstantiate(&instance, graph, NULL, NULL, 0);
cudaGraphLaunch(instance, stream);
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

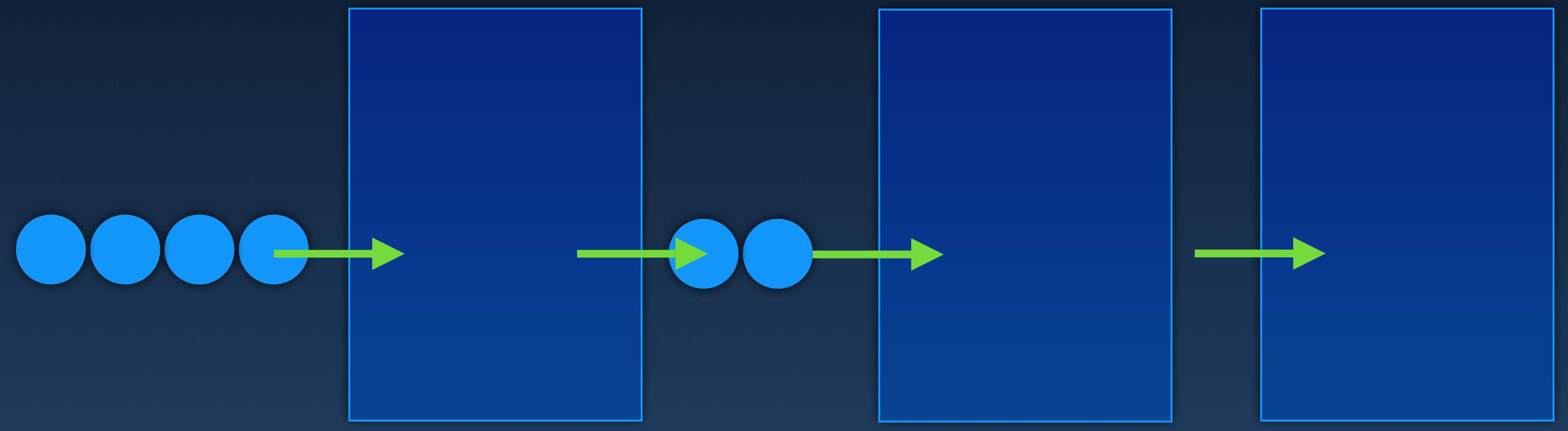
```
q = create_queue(args);
…

t = create_task(args);
update_task1(args)
…
q.submit(t);
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

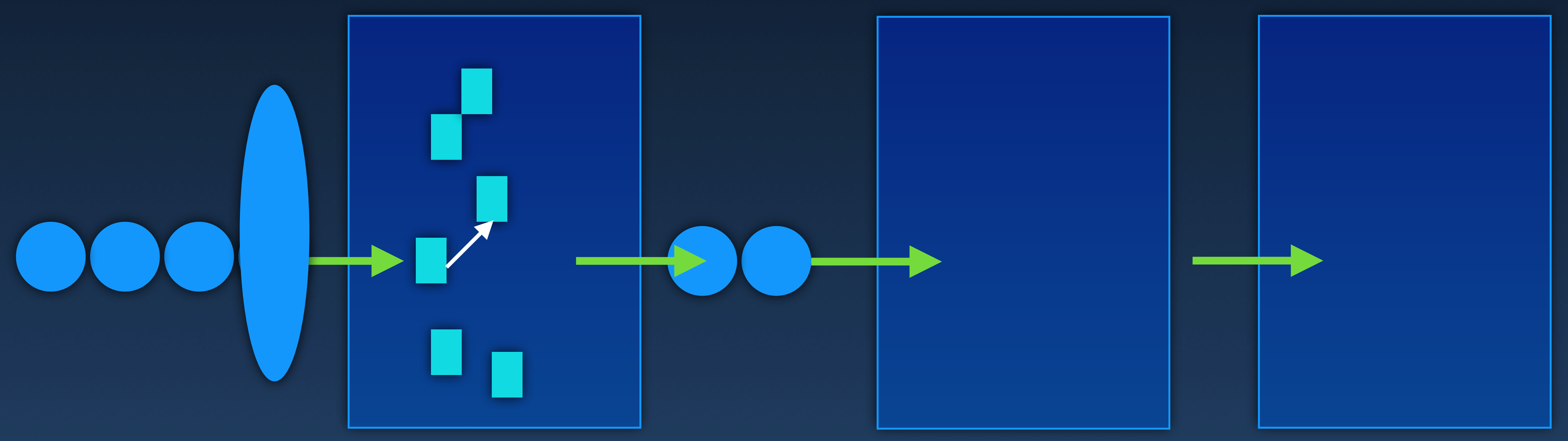
Computational pipeline



Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

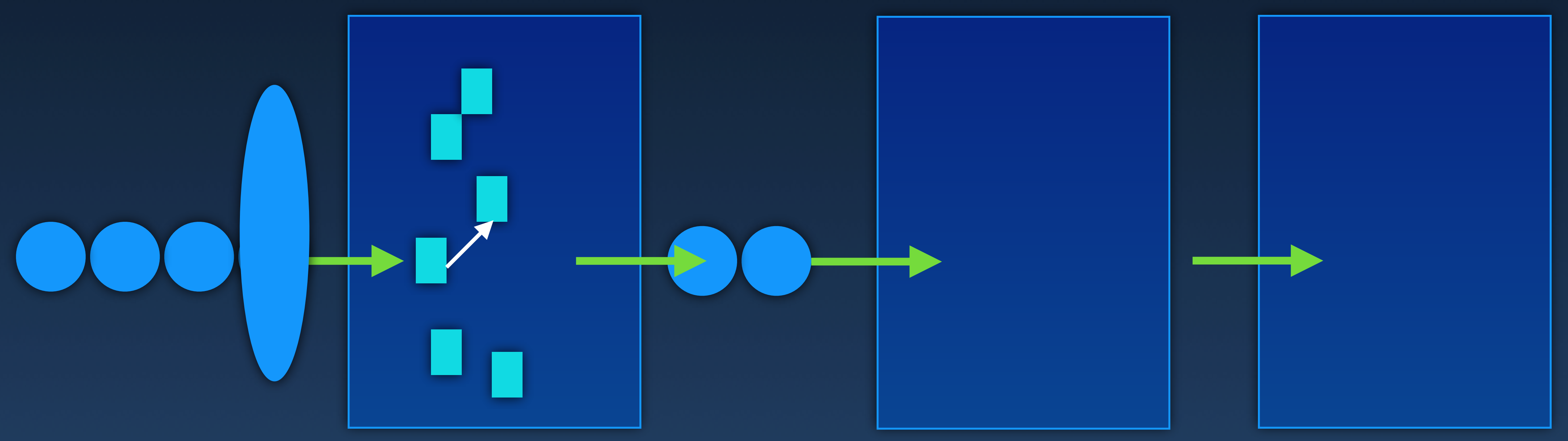
Computational pipeline



Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

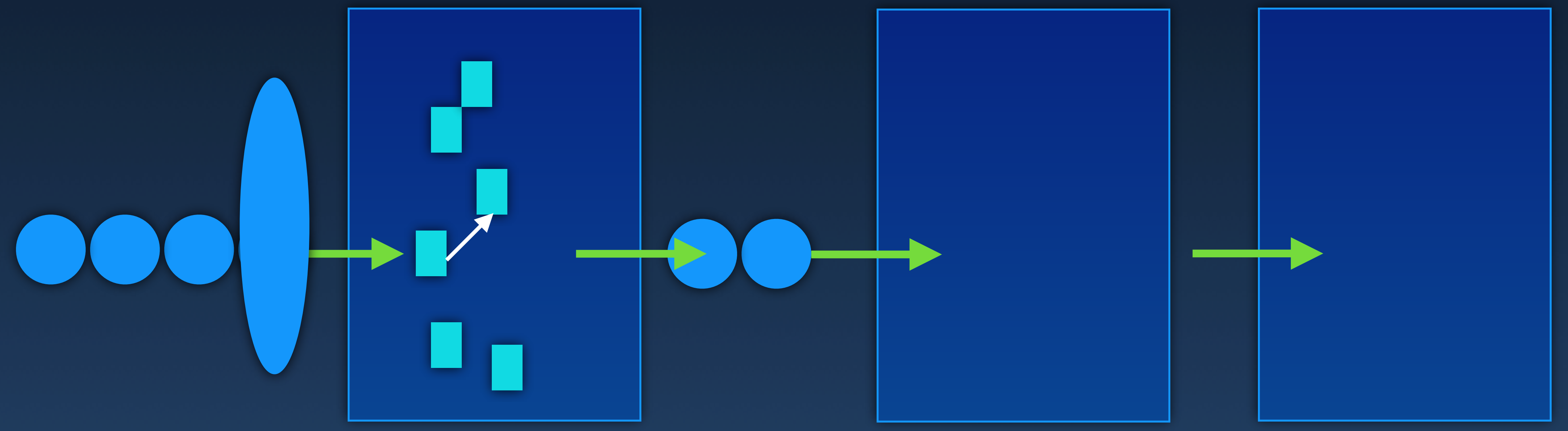
Computational pipeline



```
List outputlist =
        inputlist.stream()
                   .filter(i -> i.x > i.v).           // filter if x > v
                   .map(i -> i.y)                      // fetch y
                   .collect(Collectors.toList());// collect to list
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model



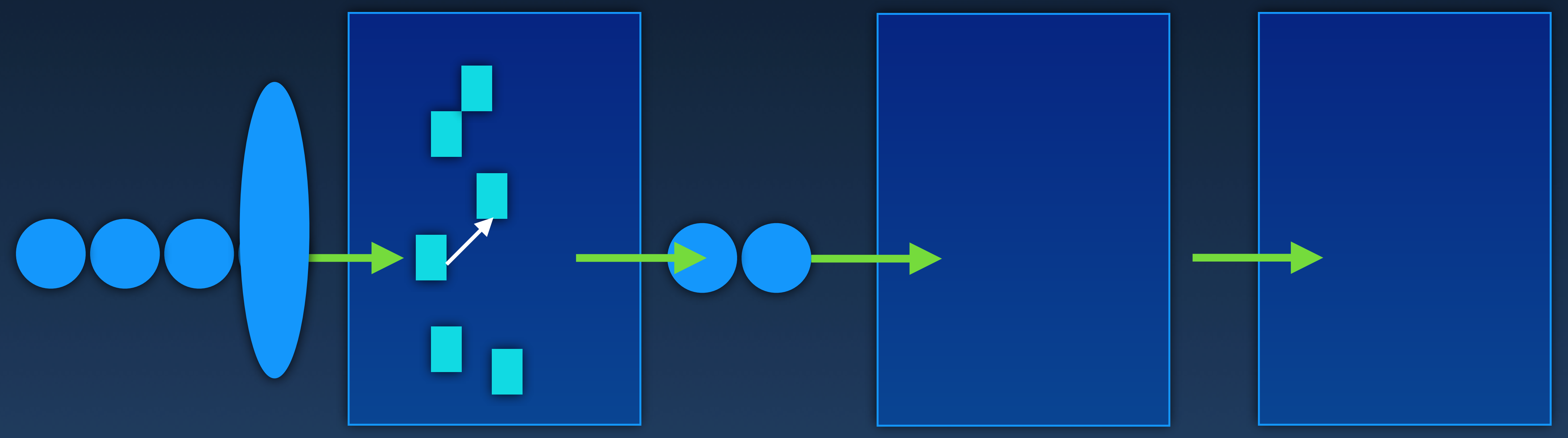
Computational pipeline

```
Float value =
        inputlist.stream()
                   .filter(i -> i.x > i.v).            // filter if x > v
                   .map(i -> i.y)                      // fetch y
                   .reduce(0f, (sum, y) -> sum+y); // reduce
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

Computational pipeline

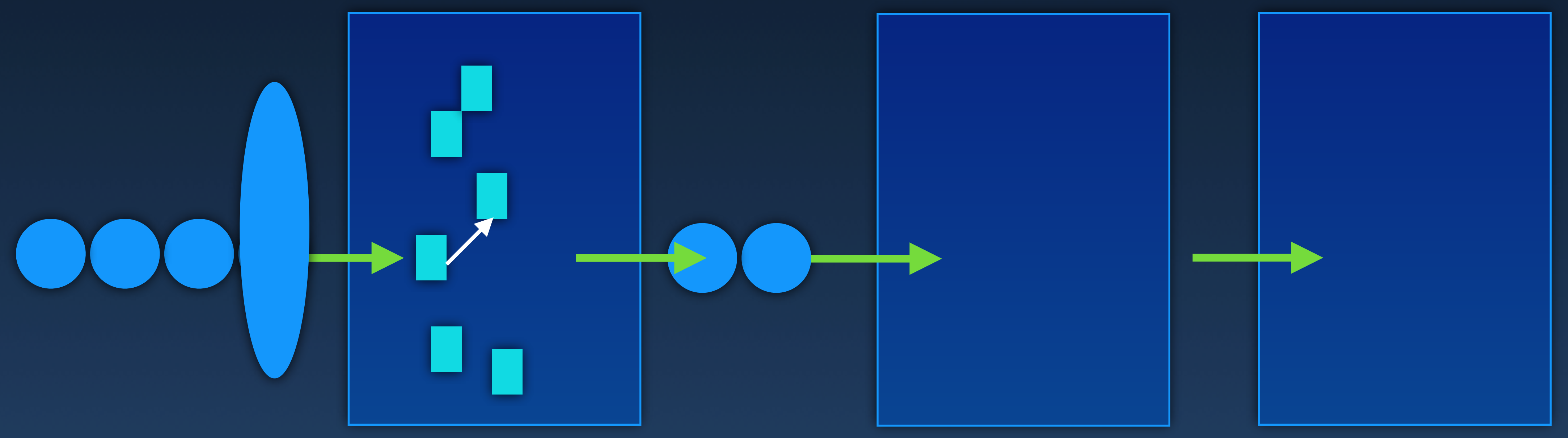


```
Float value =
        inputlist.stream().parallel()
                .filter(i -> i.x > i.v).              // filter if x > v
                .map(i -> i.y)                        // fetch y
                .reduce(0f, (sum, y) -> sum+y); // reduce
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

Computational pipeline

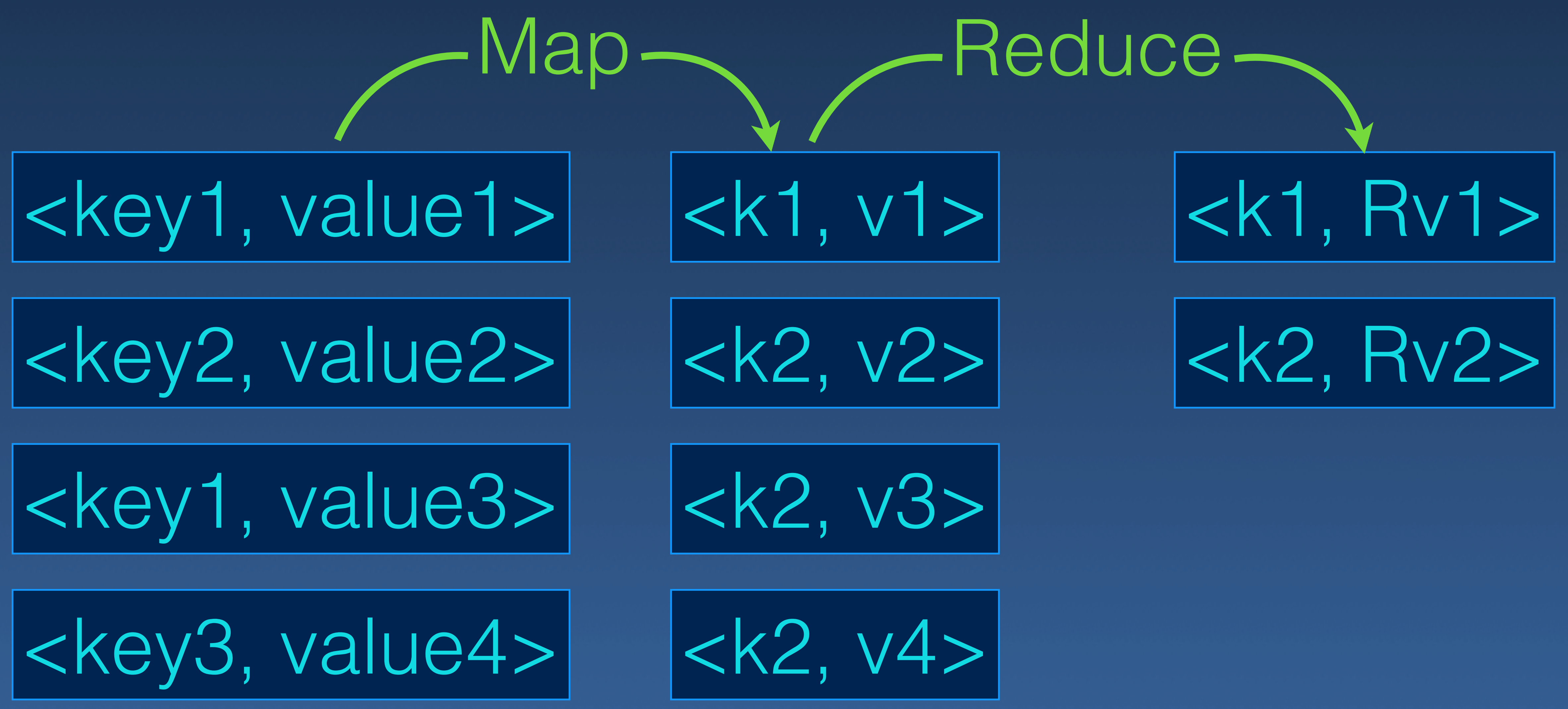


```
Float value =
        inputlist.stream().parallel()
                .filter(i -> i.x > i.v).            // filter if x > v
                                                   etch y
                                                   ); // reduce
```

```
inputlist.stream().parallel().forEach(e -> f(e));
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model



Map

Reduce

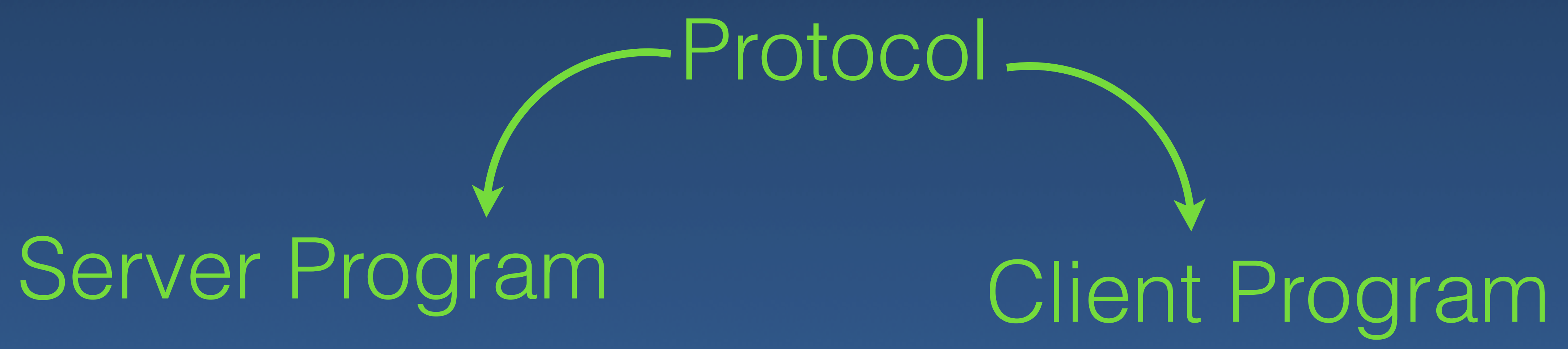
<key1, value1>	<k1, v1>	<k1, Rv1>
<key2, value2>	<k2, v2>	<k2, Rv2>
<key1, value3>	<k2, v3>	
<key3, value4>	<k2, v4>	

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based m
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

JAVA RMI:

```
Registry registry = LocateRegistry.getRegistry(hostString);
Someclass stub = (Someclass) registry.lookup("somename");
String response = stub.somemethod();
```



```
c1 = Accept()
Fork process(c1)
```

```
Request(s1)
```