COL380

Introduction to
Parallel & Distributed Programming

Agenda

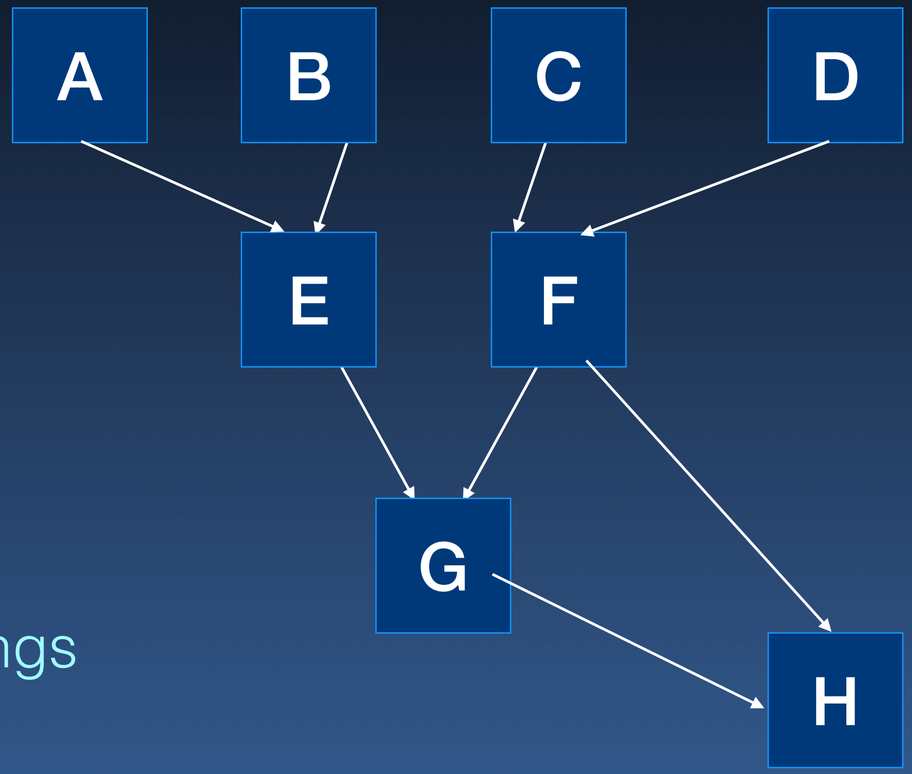
- Programming Models
- Computational Models

Programming Models

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

Task Graph

- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things



Programming Models

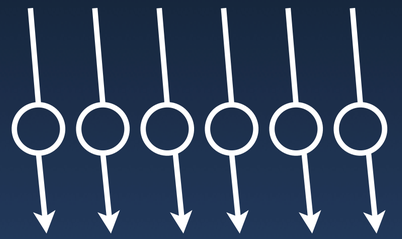
- Shared Memory model
- Distributed Me
- Task-graph ba
- Stream proces
- Work-queue m
- Map-reduce m
- Client-server m

```
cudaGraph_t graph;
cudaGraphCreate(&graph);
cudaGraphAddNode(graph, kernel_a, {}, ...);
cudaGraphAddNode(graph, kernel_b, { kernel_a }, ...);
cudaGraphAddNode(graph, kernel_c, { kernel_a }, ...);
cudaGraphAddNode(graph, kernel_d, { kernel_b, kernel_c }, …);

cudaStreamBeginCapture(stream, cudaStreamCaptureModeGlobal);
…
cudaStreamEndCapture(stream, &graph); // Convert stream to graph

cudaGraphExec_t graphinstance;
cudaGraphInstantiate(&graphinstance, graph, NULL, NULL, 0);
cudaGraphLaunch(graphinstance, stream);
```

Data Decomposition

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

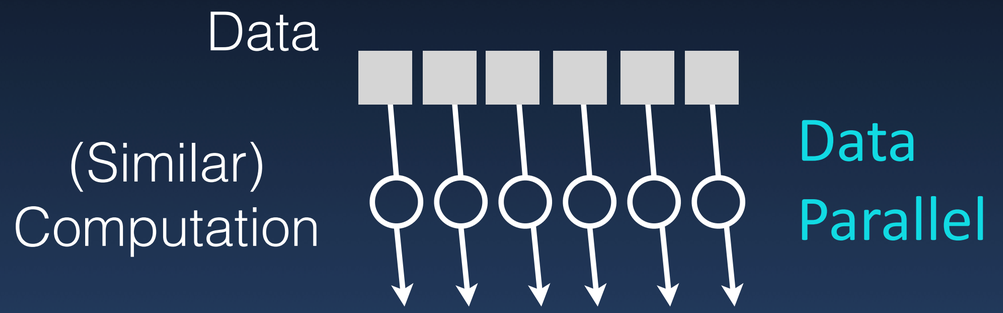
(Similar) Computation



Data Parallel

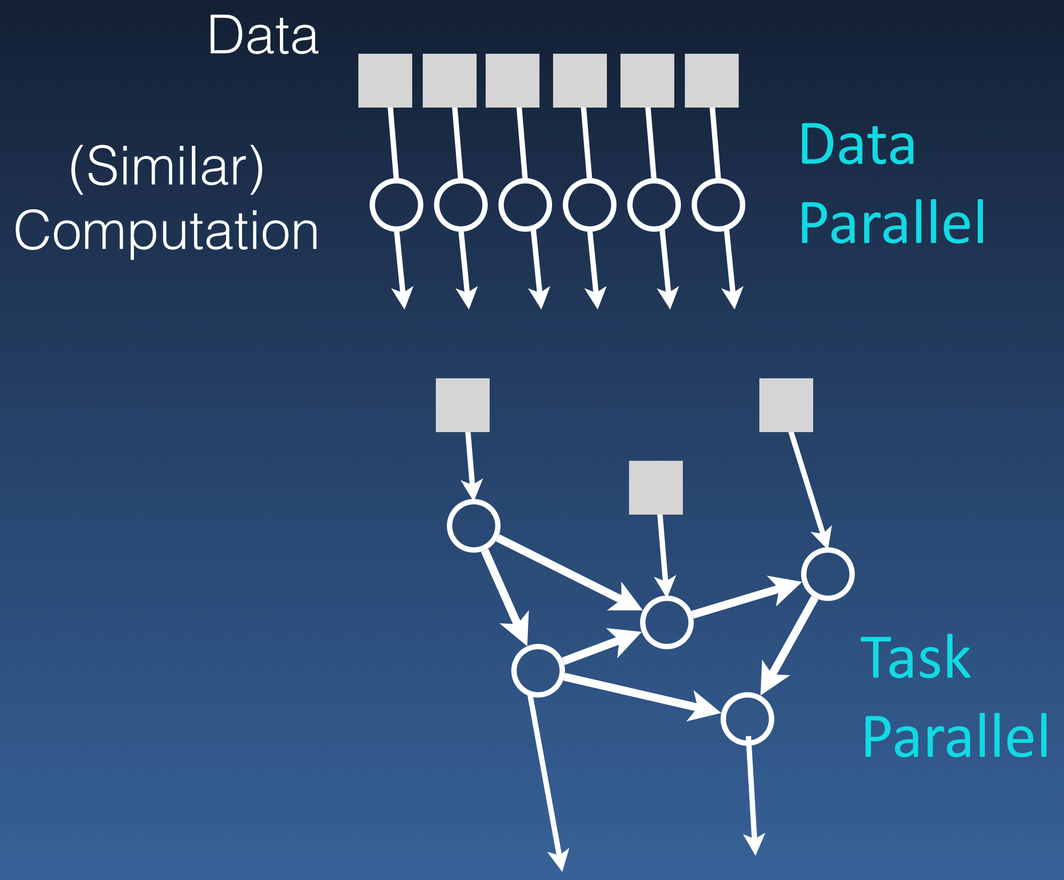
Data Decomposition

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



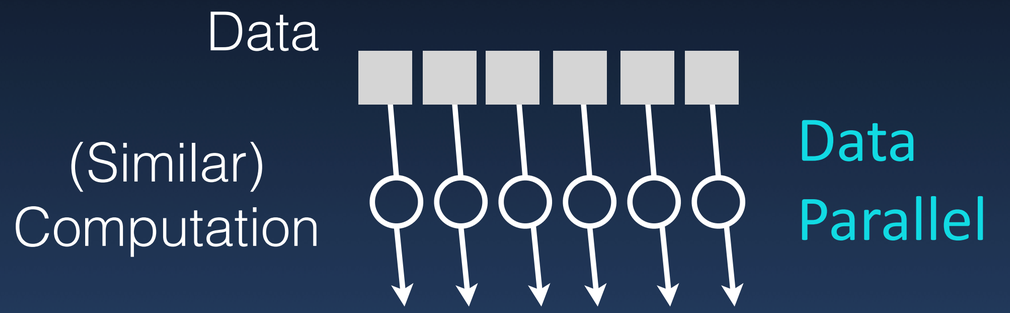
Data Decomposition

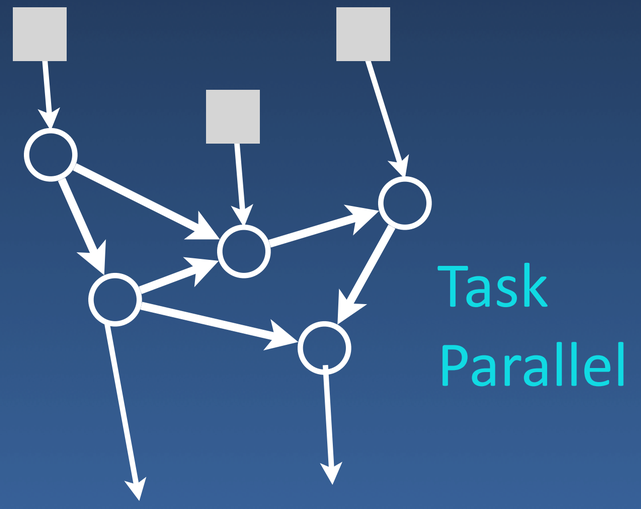
- Data Parallel
  - ➡ Perform $f(x)$ for many x
- Task Parallel
  - ➡ Perform many functions f_i
- Pipeline

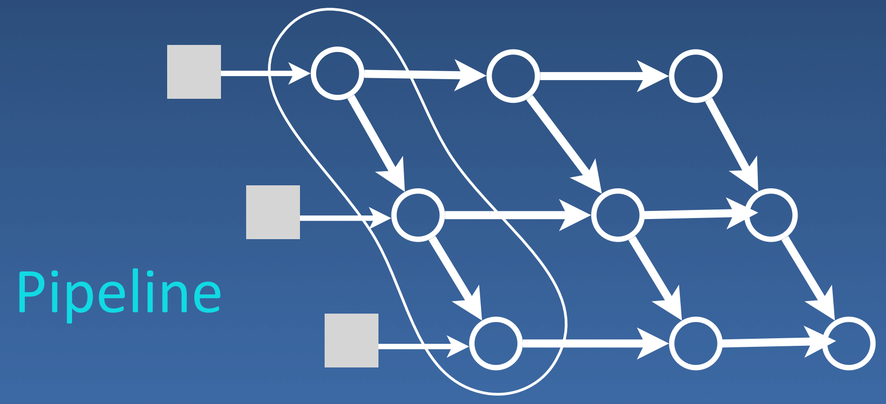


Data Decomposition

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



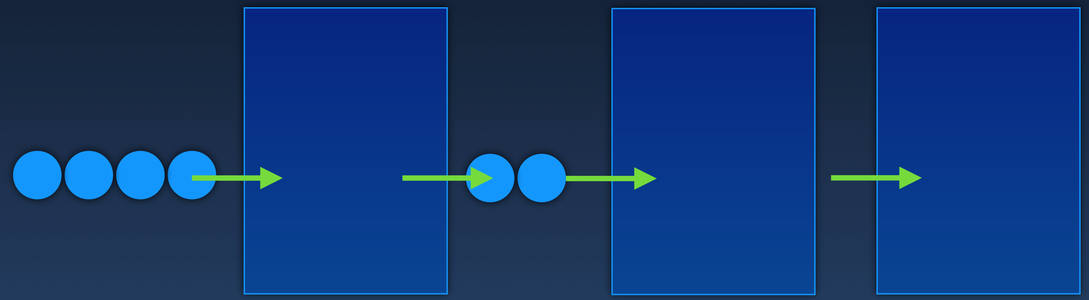




Programming Models

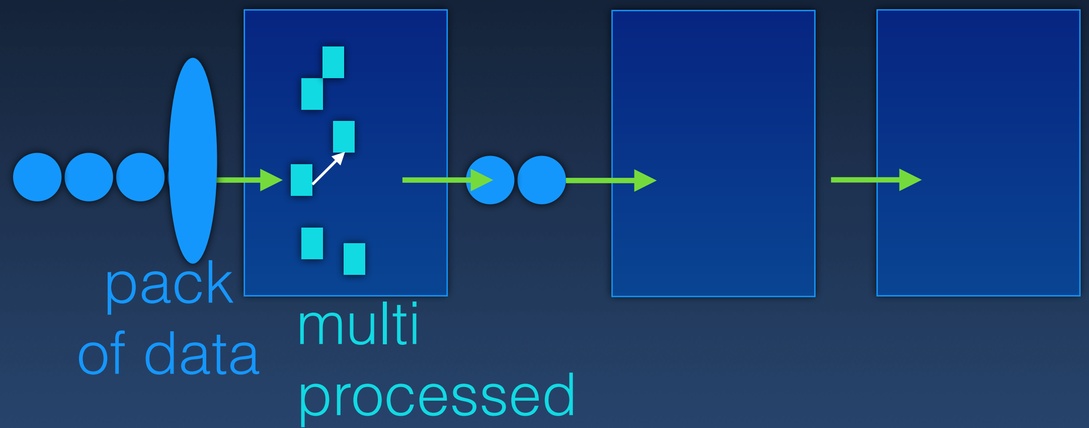
- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Stream processing model
- Work-queue model
- Map-reduce model
- Client-server model

Computational pipeline



Programming Models

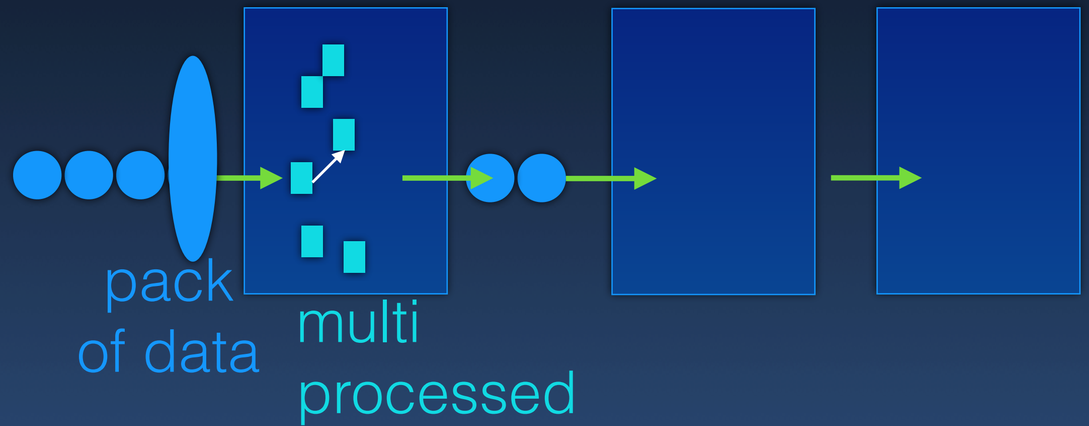
- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Stream processing model
- Work-queue model
- Map-reduce model
- Client-server model



Programming Models

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

Computational pipeline

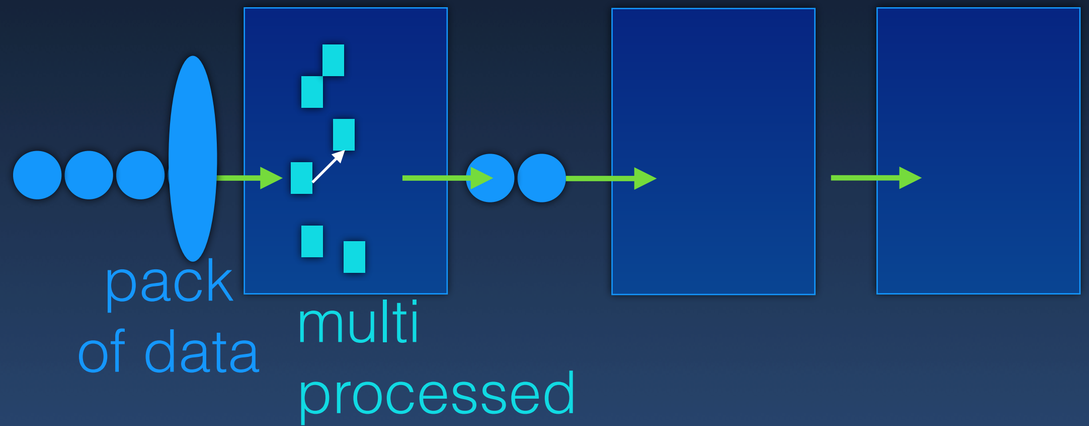



```
List outputlist =
        inputlist.stream()
                .filter(i -> i.x > i.v).          // filter if x > v
                .map(i -> i.y)                   // fetch y
                .collect(Collectors.toList());// collect to list
```

Programming Models

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

Computational pipeline

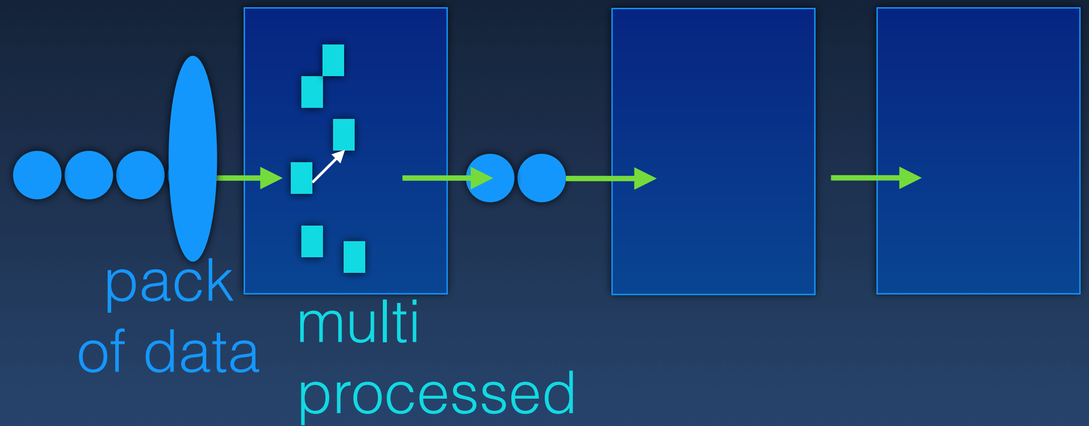



```
Float value =
        inputlist.stream()
                .filter(i -> i.x > i.v).          // filter if x > v
                .map(i -> i.y)                   // fetch y
                .reduce(0f, (sum, y) -> sum+y); // reduce
```

Programming Models

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

Computational pipeline

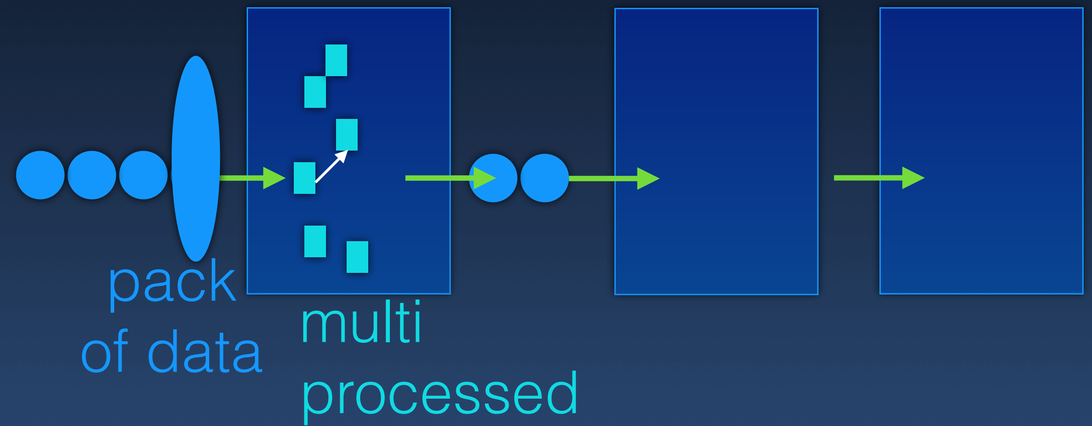



```
Float value =
        inputlist.stream().parallel()
                .filter(i -> i.x > i.v).          // filter if x > v
                .map(i -> i.y)                   // fetch y
                .reduce(0f, (sum, y) -> sum+y); // reduce
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Stream processing model
- Work-queue 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
- Stream processing model
- Work-queue 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

<key1, value1>

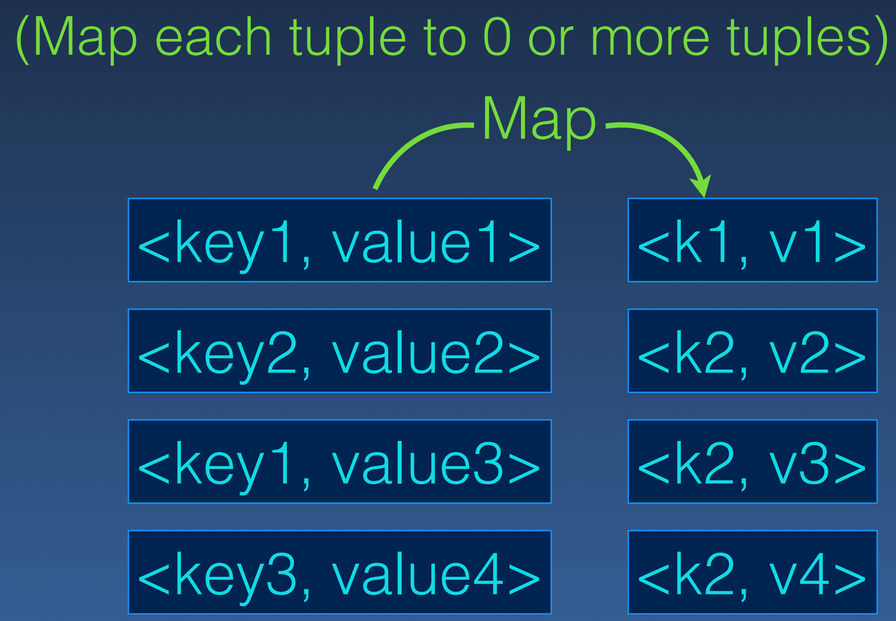
<key2, value2>

<key1, value3>

<key3, value4>

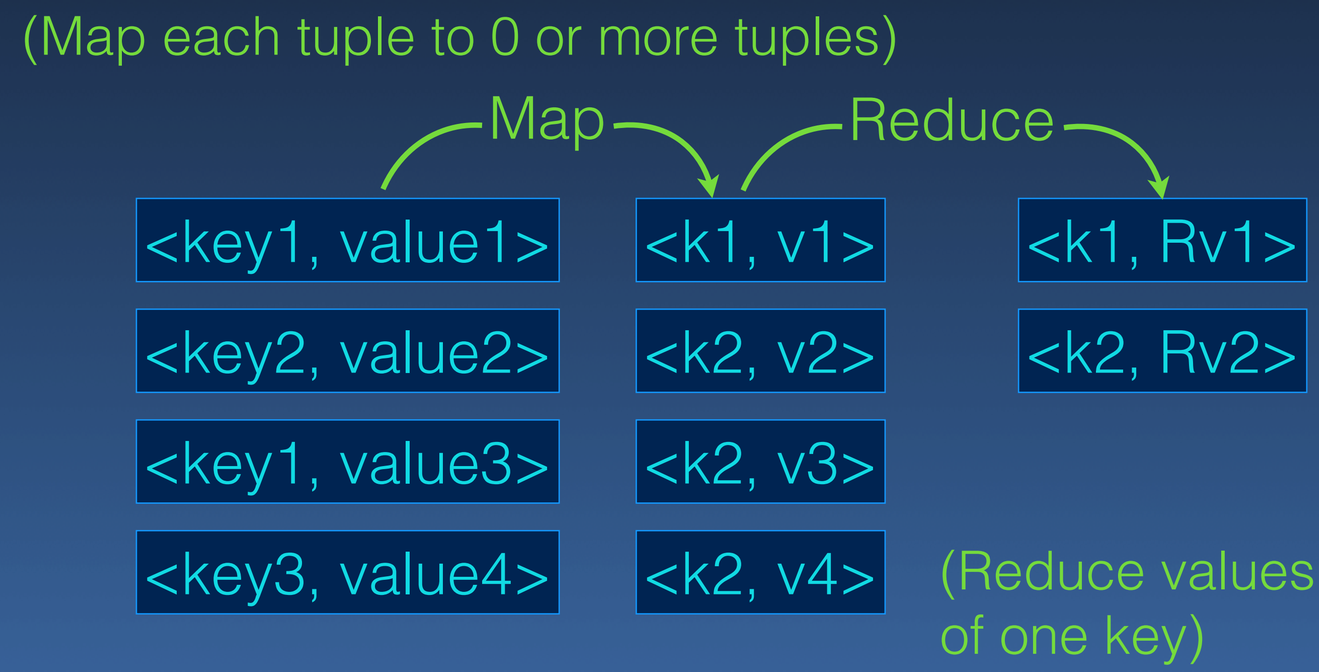
Programming Models

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



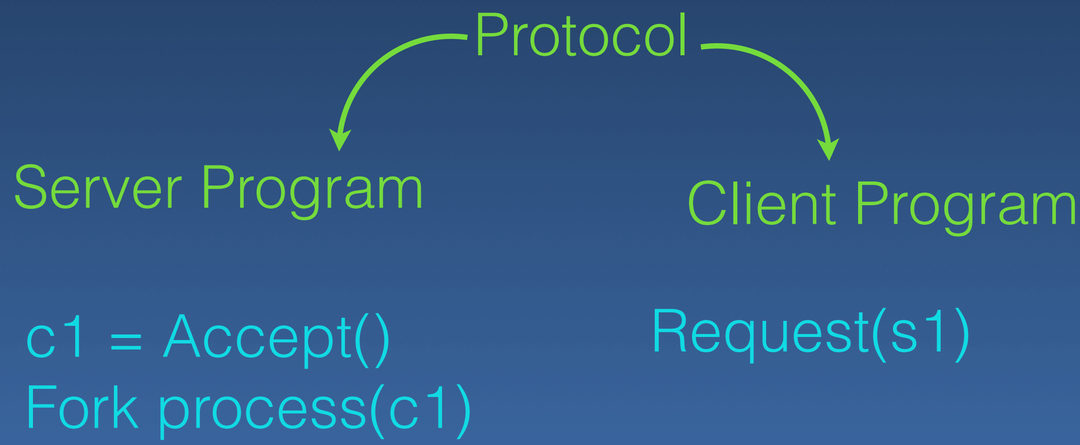
Programming Models

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

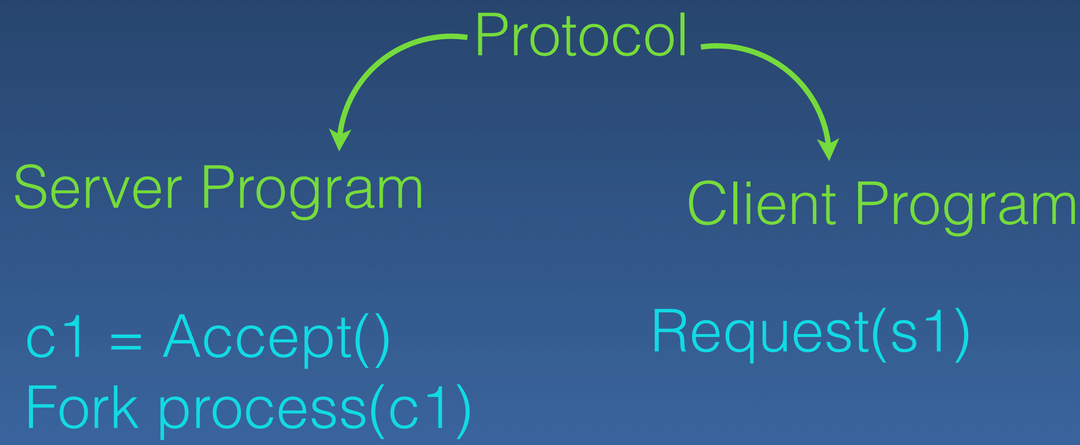


Programming Models

- Shared Memory 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();
```



Review

- Examples of programming using
  - Task-graph based model
  - Stream processing model
  - Work-queue model
  - Map-reduce model
  - Client-server model