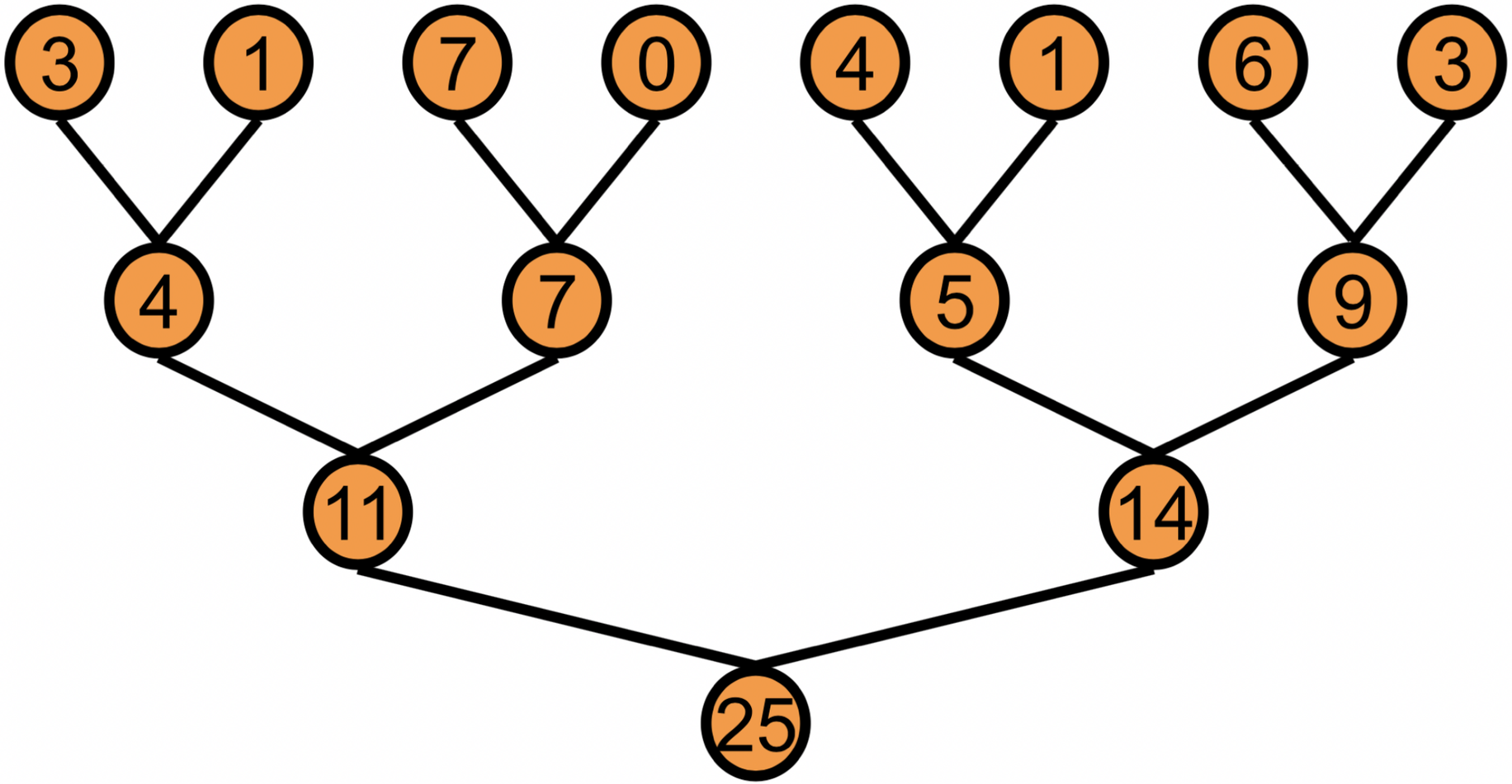
CME 213, ME 339—Spring 2021

Eric Darve, ICME, Stanford

“The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.” (Stephen Hawking)

Group activity with prefix scan

Parallel reduction or scan



Cumulative sum or prefix scan

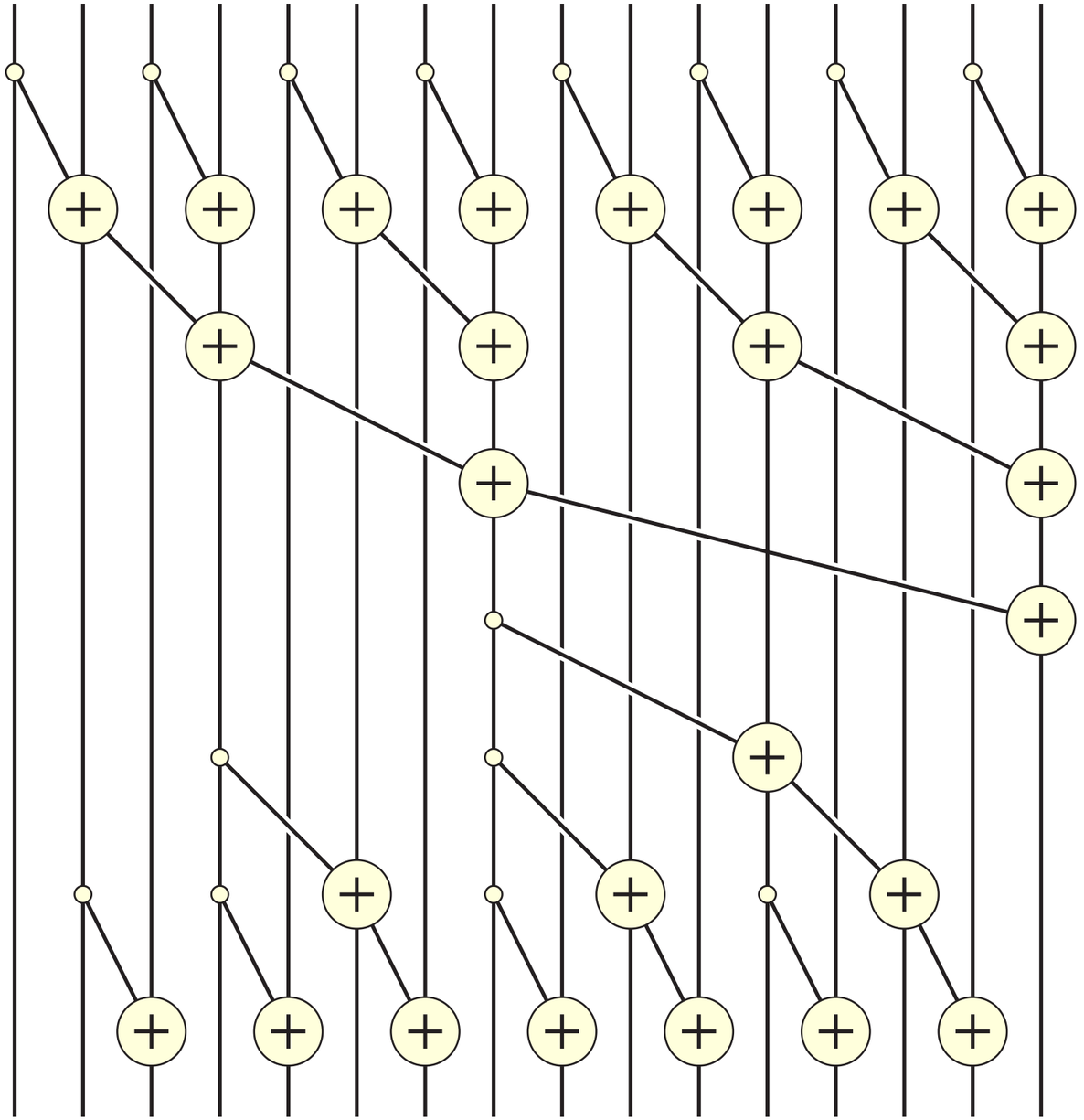
```
3   5   6   2   4
3   8  14  16  20
```

Algorithm 1

Work efficient

Two phases

1. Collect or reduce
2. Distribute



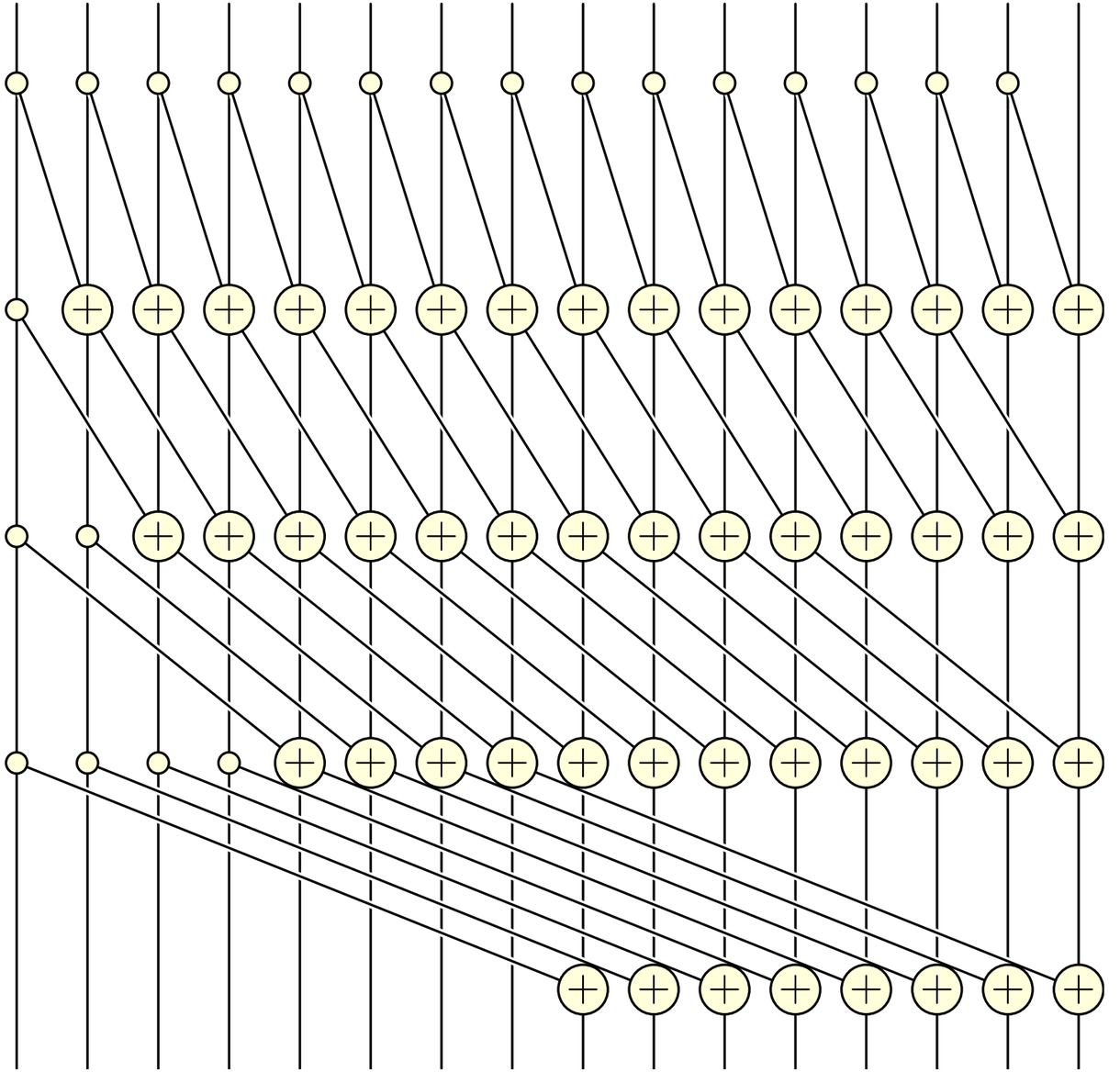
Number of passes: $2 \log_2 n - 2$

Amount of work: ~ 2 x sequential flops

Can we reduce the running time by adding more flops?

Algorithm 2: Hillis–Steele

Main idea: concurrent tree reductions



Number of passes: $\log_2 n$ vs $2 \log_2 n$

Amount of work: ~ $\log_2 n$ x sequential flops

When a lot of processors are available, Hillis–Steele algorithm is superior.

Shorter span vs work-efficient

Game time

Form teams of 10–14 players

Goal of activity: parallel prefix scan using human processors

Download the code from class web page:

generate_sequence.cpp

Compile and run

```
$ g++ -std=c++11 generate_sequence.cpp; ./a.out
```

Each group of players is assigned a unique group ID

```
$ g++ -std=c++11 generate_sequence.cpp; ./a.out
```

Enter your group number

```
Enter your group number (an integer greater or equal to 1)
1
Selected group ID: 1
Row 1: index
Row 2: random value

Index  1 to 10
1           2           3           4           5           6           7           8           9           10
92042       30656       29306       78086       18200       58661       37315       62538       18682       55136

Index 11 to 20
11          12          13          14          15          16          17          18          19          20
58612       97333       91698       92309       76746       59943       89398       82595       12042       20990

Index 21 to 30
21          22          23          24          25          26          27          28          29          30
76933       24332       48451       73520       86703       44385       45908       76778       92724       71110

Index 31 to 40
31          32          33          34          35          36          37          38          39          40
31160       69749       83981       42199       15489       60934       71592       97890       98748       71890

Index 41 to 50
41          42          43          44          45          46          47          48          49          50
14235       47311       47343       45712       61789       60090       86268       92795       16769       54642

Index 51 to 60
51          52          53          54          55          56          57          58          59          60
10333       12637       27953       47918       93868       81824       91842       56957       55775       84172
```

The code returns a sequence of random numbers.

This is the sequence you use for the cumulative sum.

We are going to build a computer using humans.

There are 3 types of players in this game: **mem**, **net**, **pu**.

Each player has only one type.

You can have as many players of each type as you want.

In previous years, we were doing this exercise on the lawn on the Stanford oval.

This year we will use zoom and emails instead and try to replicate a similar experience.

All players in the same team will be in the same breakout room on zoom.

You can communicate in the breakout room.

However, to compute the prefix sum you need to communicate exclusively through emails following the rules below.

This process mimics the time taken to execute instructions on a computer.

In the example below, we will assume that we want to add

```
12 + 23 = 35
```

Being able to add two integers is all we need to compute the prefix sum in parallel.

The thinkers

mem player: they can send emails to the **net** players.

The content of the email should follow the pattern:

```
Add the numbers 12 and 23
```

The runners (they used to run on the lawn from **mem** to **pu**)

net players follow these steps:

(1) Compile the commands from the user. The assembly code is:

```
LD R1, [12];
LD R2, [23];
IADD R3, R1, R2;
```

(2) Use this online tool to convert the string above to a binary code.

Output

```
01001100 01000100 00100000 01010010 00110001 00101100 00100000 01011011 00110001 00110010 01011101 00111011 000
```

(3) Send the binary code to the **pu** player by email.

*You cannot send another message to that **pu** before you receive a reply back.*

The machines

pu player:

1. Decode the binary message using the decoder.
2. Compute the result.
3. Email the result, 35, to the **net** player.

The **net** player now performs the following tasks:

(1) Generate the assembly for the result

```
ST [35], R3;
```

(2) Encode the message:

```
01010011 01010100 00100000 01011011 00110010 00110011 01011101 00101100 00100000 01010010 00110011 00111011
```

(3) Email the binary code to the **mem** player

The **mem** player decodes the message.

Encoder:

https://www.rapidtables.com/convert/number/ascii-to-binary.html

Decoder:

https://www.rapidtables.com/convert/number/binary-to-ascii.html

Make sure you have everyone's email!

Pick a team name


GET READY
get set
GO

Discussion

How did you organize your group?

What was the best strategy?

What were the main bottlenecks?

What would you do differently?