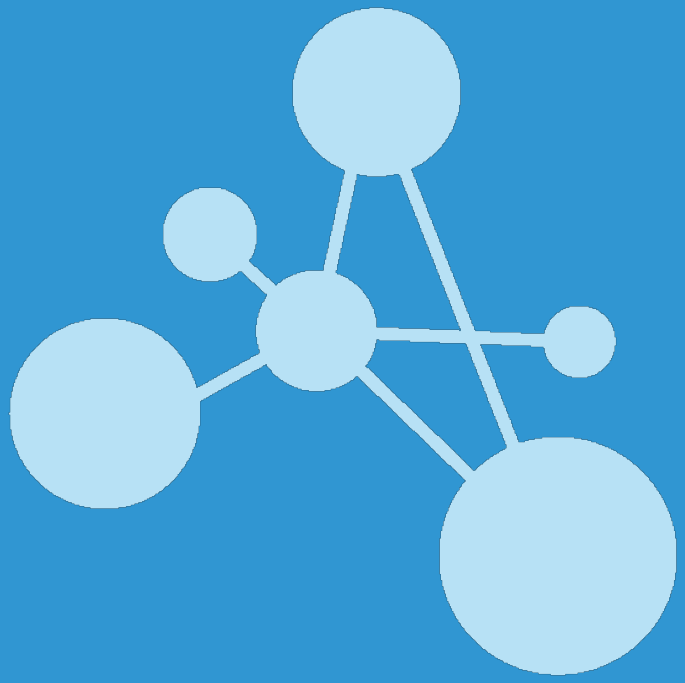
Distributed VS Parallel implementations of graph algorithms



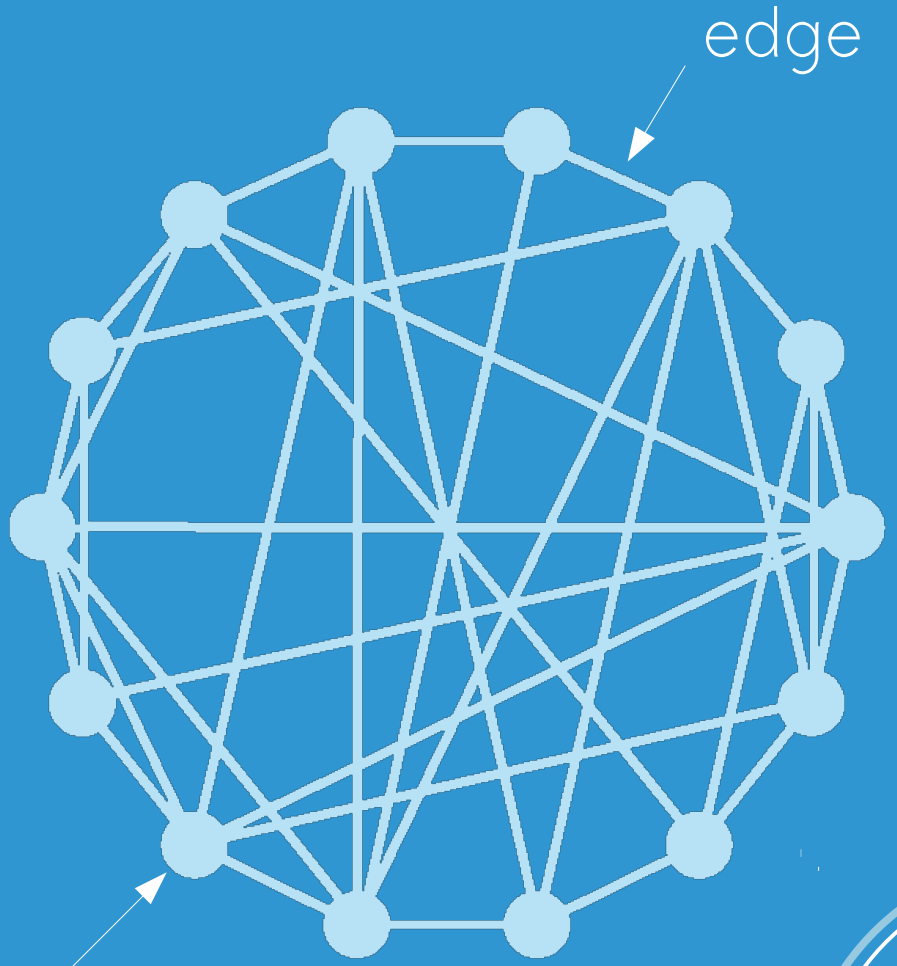
Alexis SIRETA,Lazar PETROV

Outline

About graph computing

What is a graph ?

A graph is a set of nodes connected to each other by edges



What kind of graphs ?

Edges can be :

Unweighted

weighted

Directed

Undirected

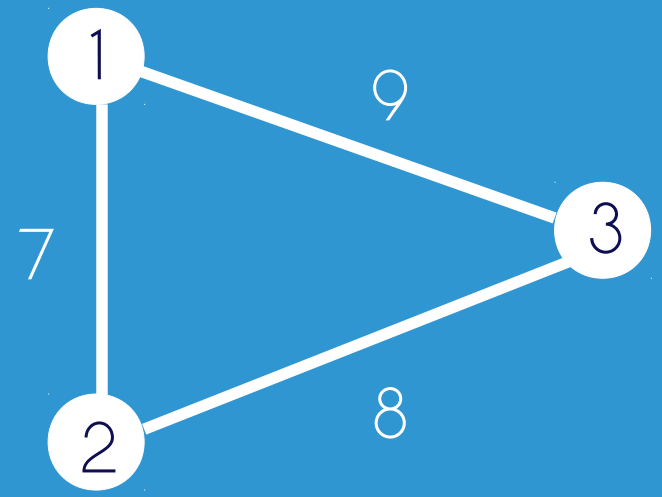
Connected graph

A connected graph is a graph in which there is a path between every pair of nodes

How to represent a graph ?

Adjacency matrix

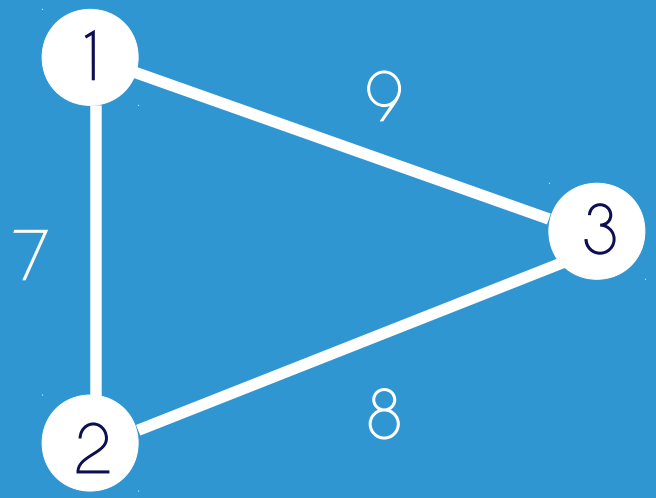
	1	2	3
Node1	0	7	9
Node2	7	0	8
Node3	9	8	0



How to represent a graph ?

Edge list

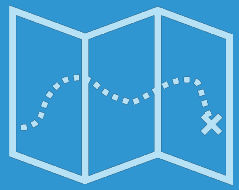
Nodea	Nodeb	W
Node1	Node2	7
Node1	Node3	9
Node2	Node3	8
Node2	Node1	7
Node3	Node1	9
Node3	Node2	8



What are graphs used for ?

Data representation of a wide range problems :

- Finding shortest path from A to B



- Representing database

- Find related topics

...and plenty more !

Problem !

Graphs are getting **VERY** big :

Example :

Directed network of hyper links between the articles of the Chinese online encyclopedia Baidu.

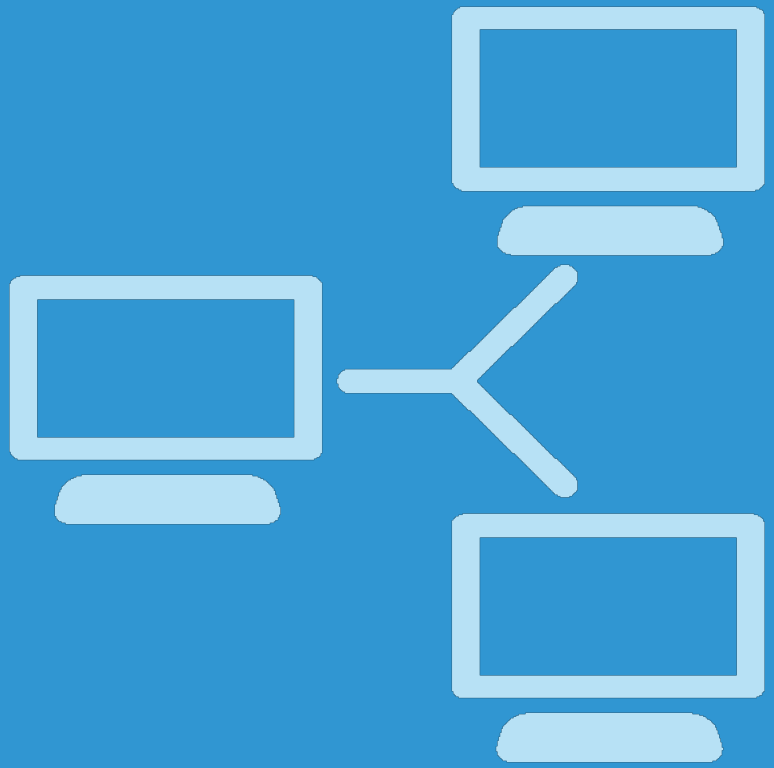
17 643 697 edges

source : http://konect.uni-koblenz.de/networks/zhishi-baidu-internallink

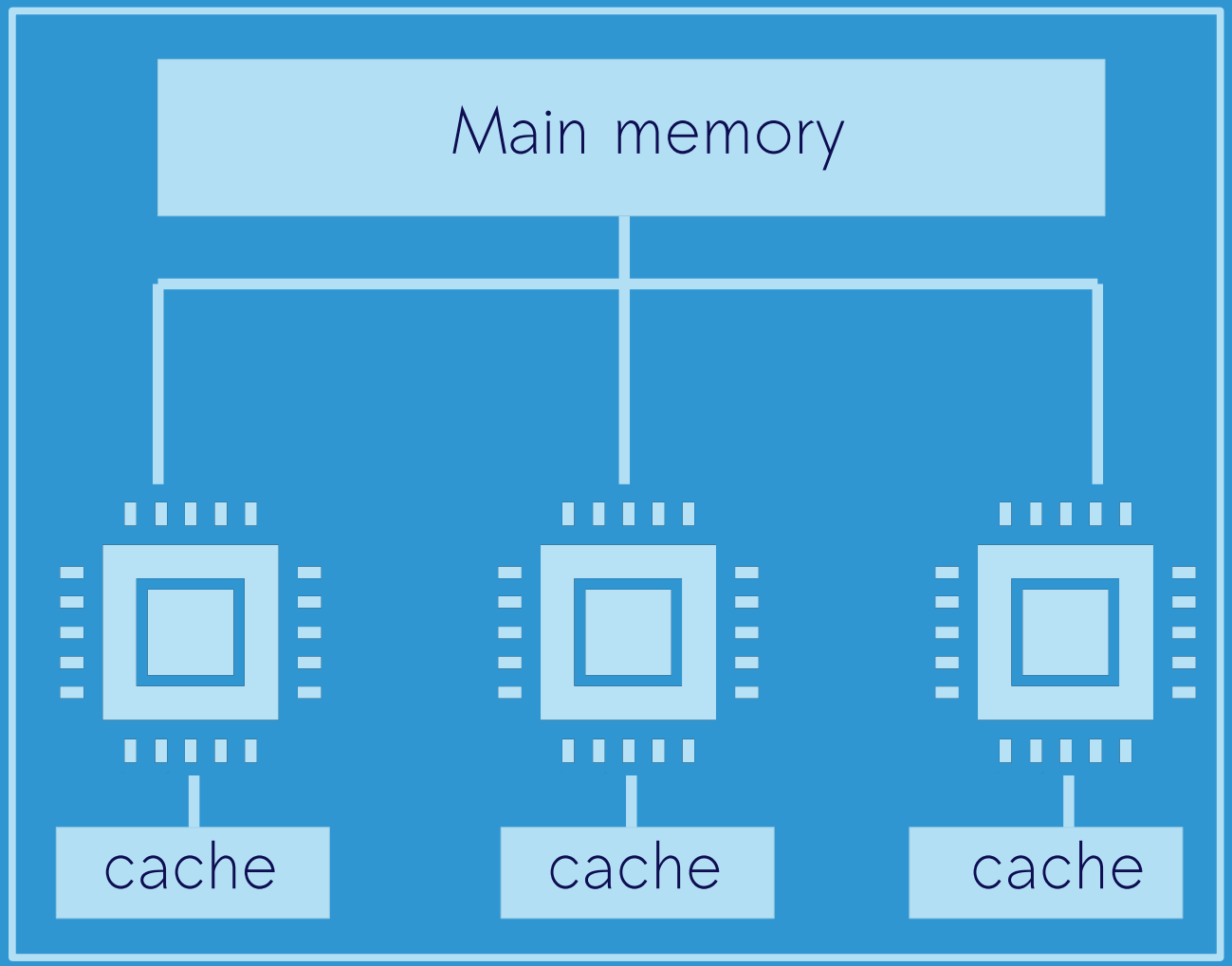
Solution !

Use **Parallel** or **Distributed** systems

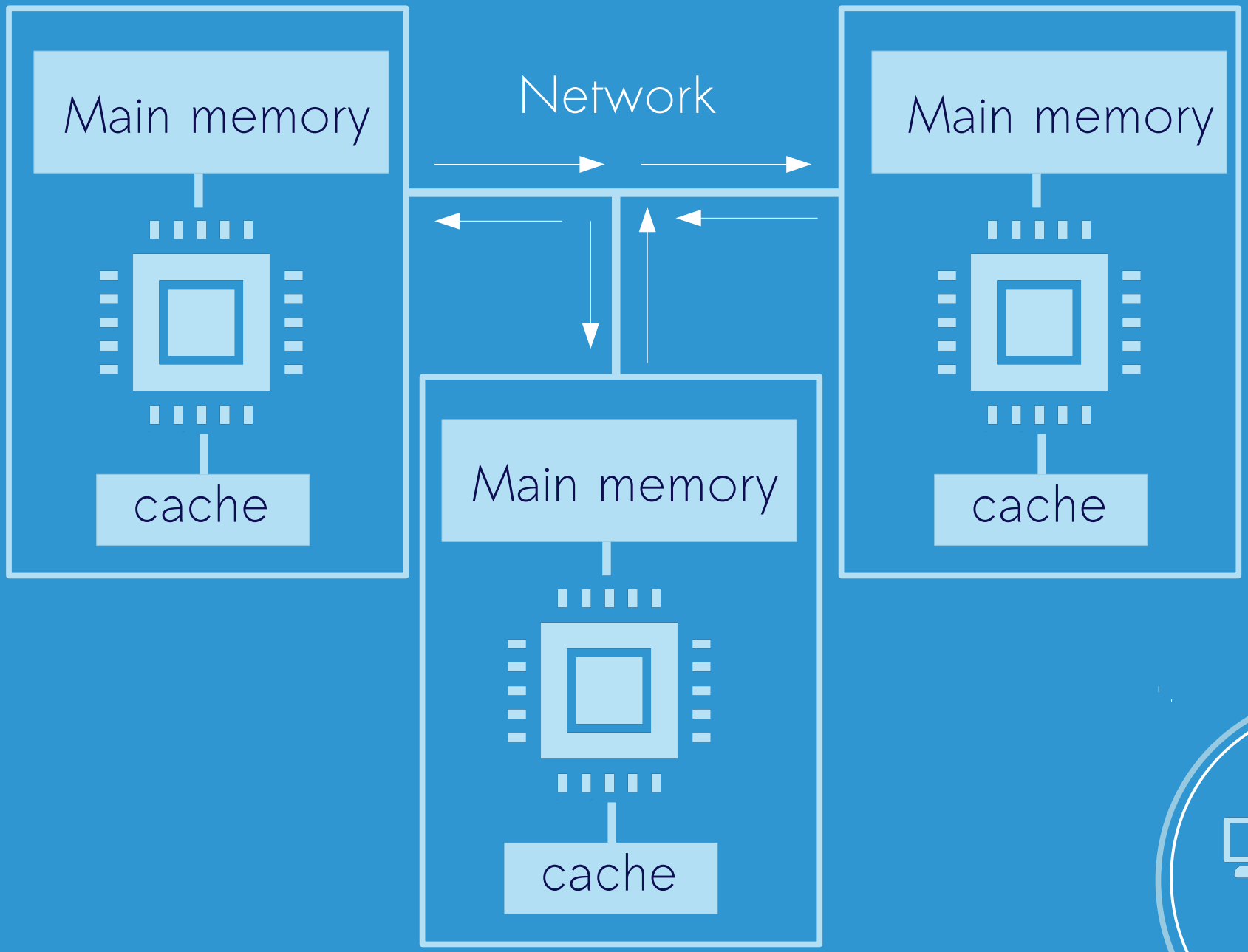
Distributed and Parallel systems



Parallel System



Distributed System



Our Research Project

Goal and Questions

Compare the performances of parallel and distributed implementations of a graph algorithm

Questions:

Can we really compare algorithms running on different architectures ?

How do the algorithms scale ?

How do they adapt to other architectures ?

Hypothesis

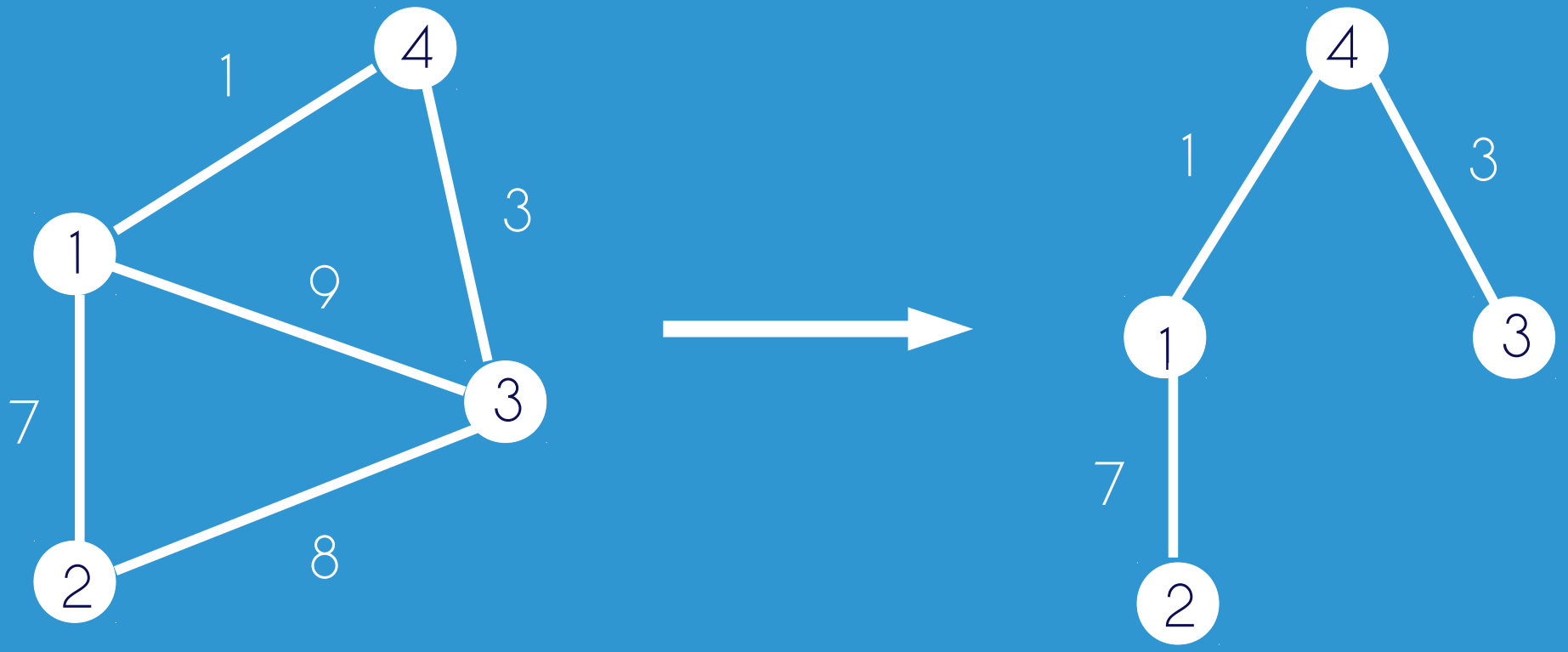
Hypothesis: *Distributed will run slower than parallel for small graphs because of communication latency but will run faster for big graphs because of memory access time*

Procedure

- Choose two implementations of one graph algorithm
- Build a theoretical model of the execution time
- Run the algorithms on the Uva cluster
- Explain the results and adapt the theoretical model if needed

Minimum Spanning Tree

What is it ?



Is relevant for connected undirected graphs

Which algorithm choose ?

Several classical algorithms : Prim, Kruskal, **Boruvka**

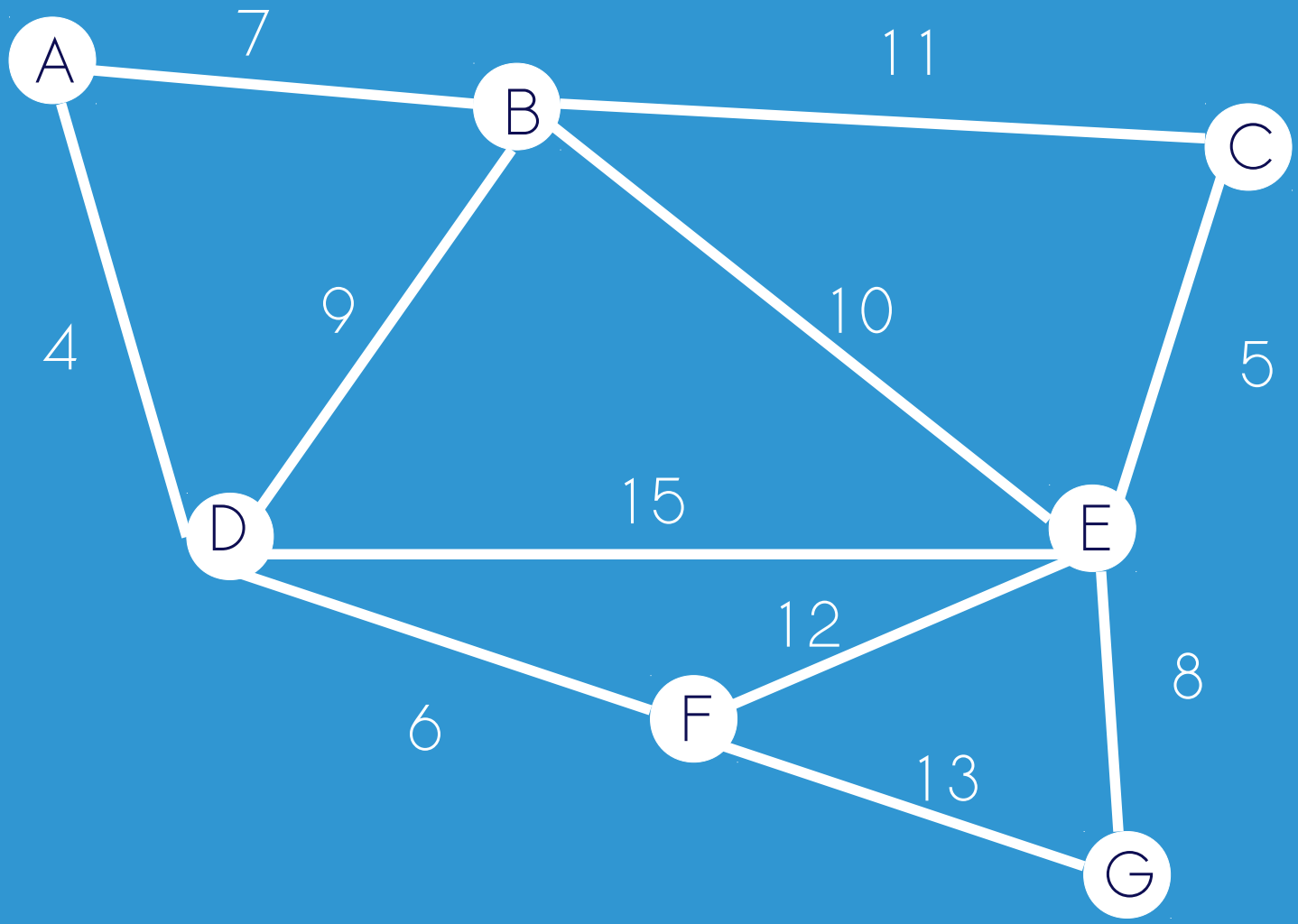
Boruvka : This is the most used for parallel and distributed implementations, therefore this is the one we chose

Parallel implementation : Bor-el, described in the paper "*Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs*" by David A. Bader and Guojing Cong

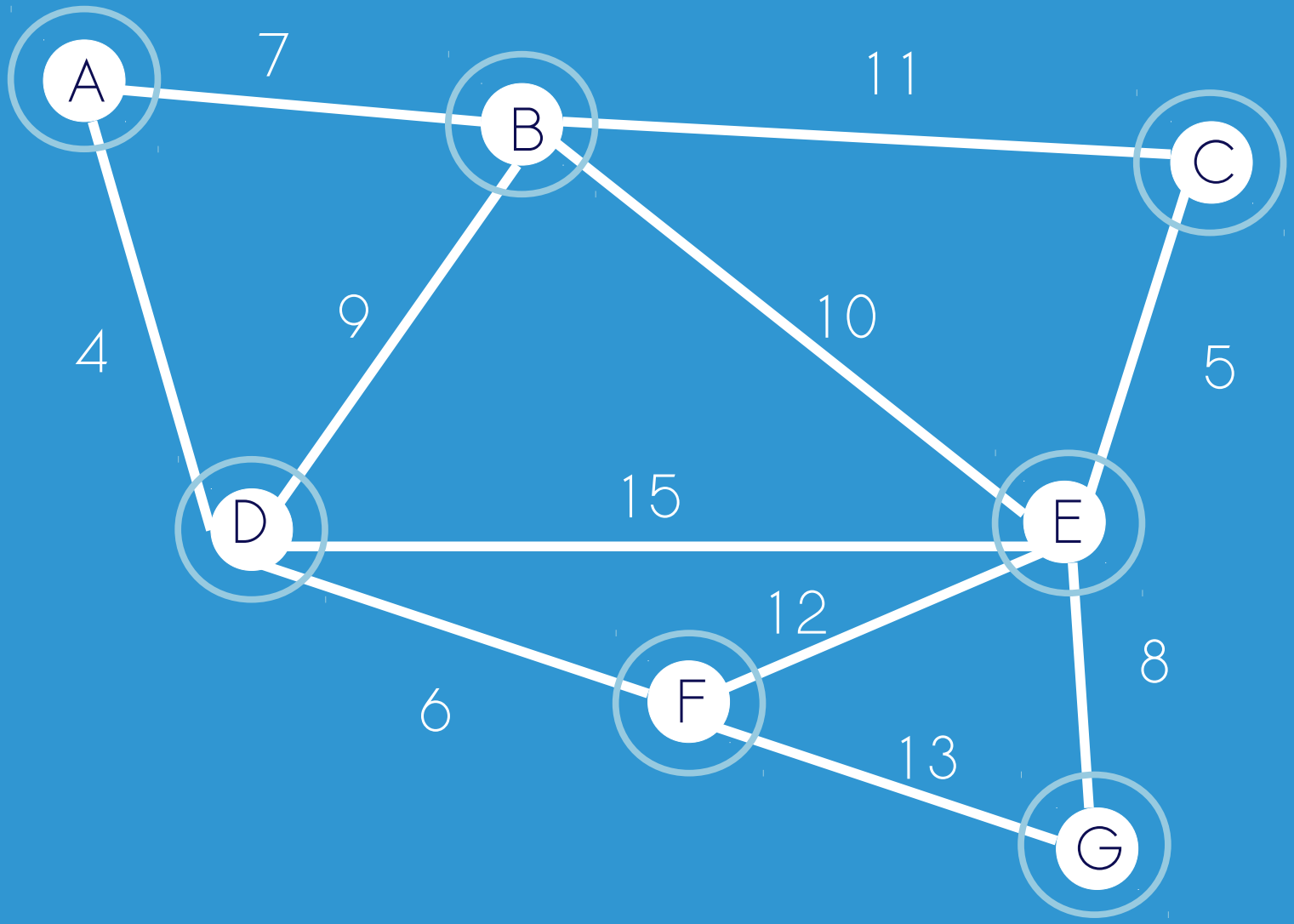
Distributed implementation : GHS, described in "*A distributed algorithm for minimum weight spanning trees*" by R. G. Gallager, P. A. Humblet and P. M. Spira

Sequential algorithm

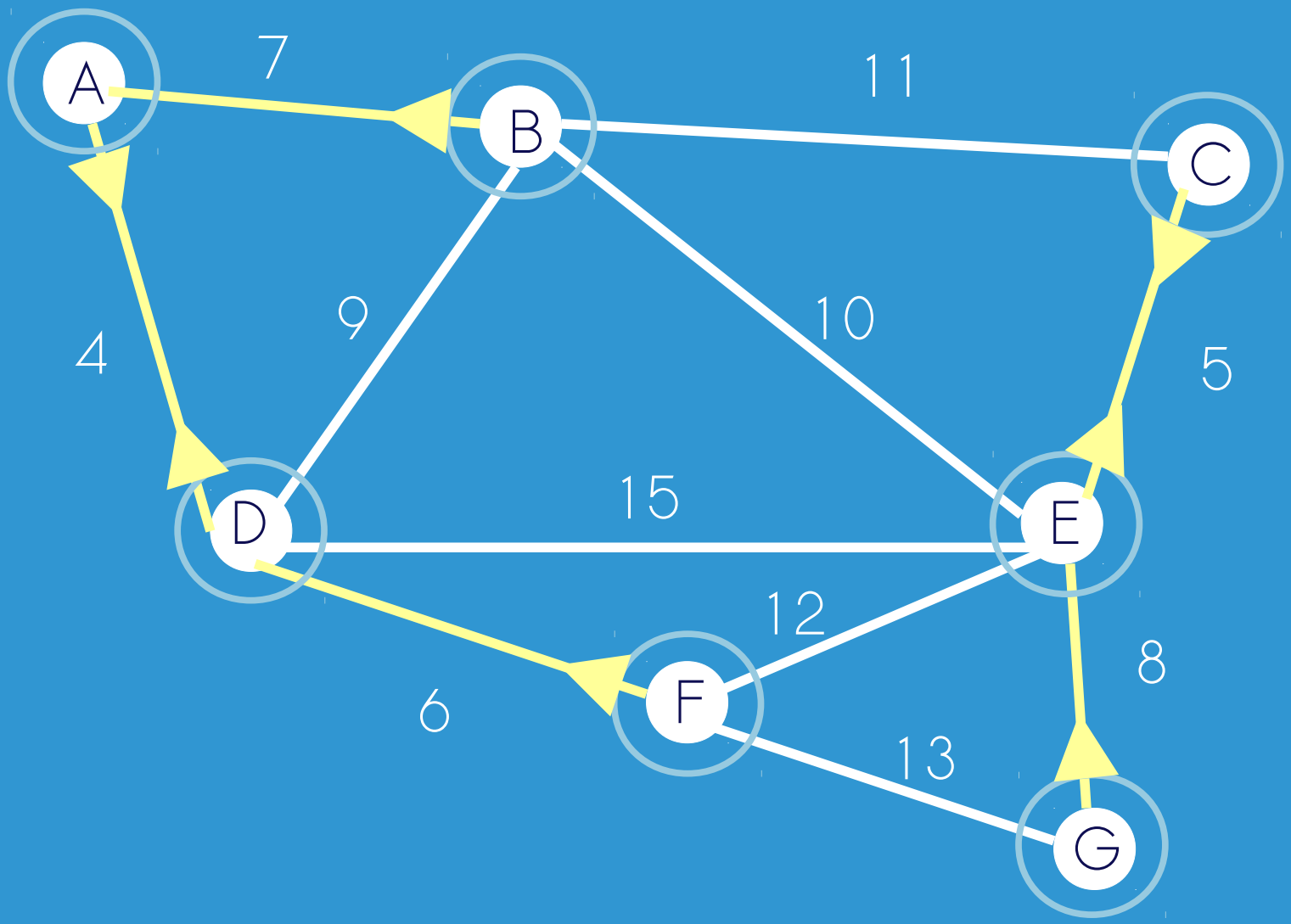
Example Graph



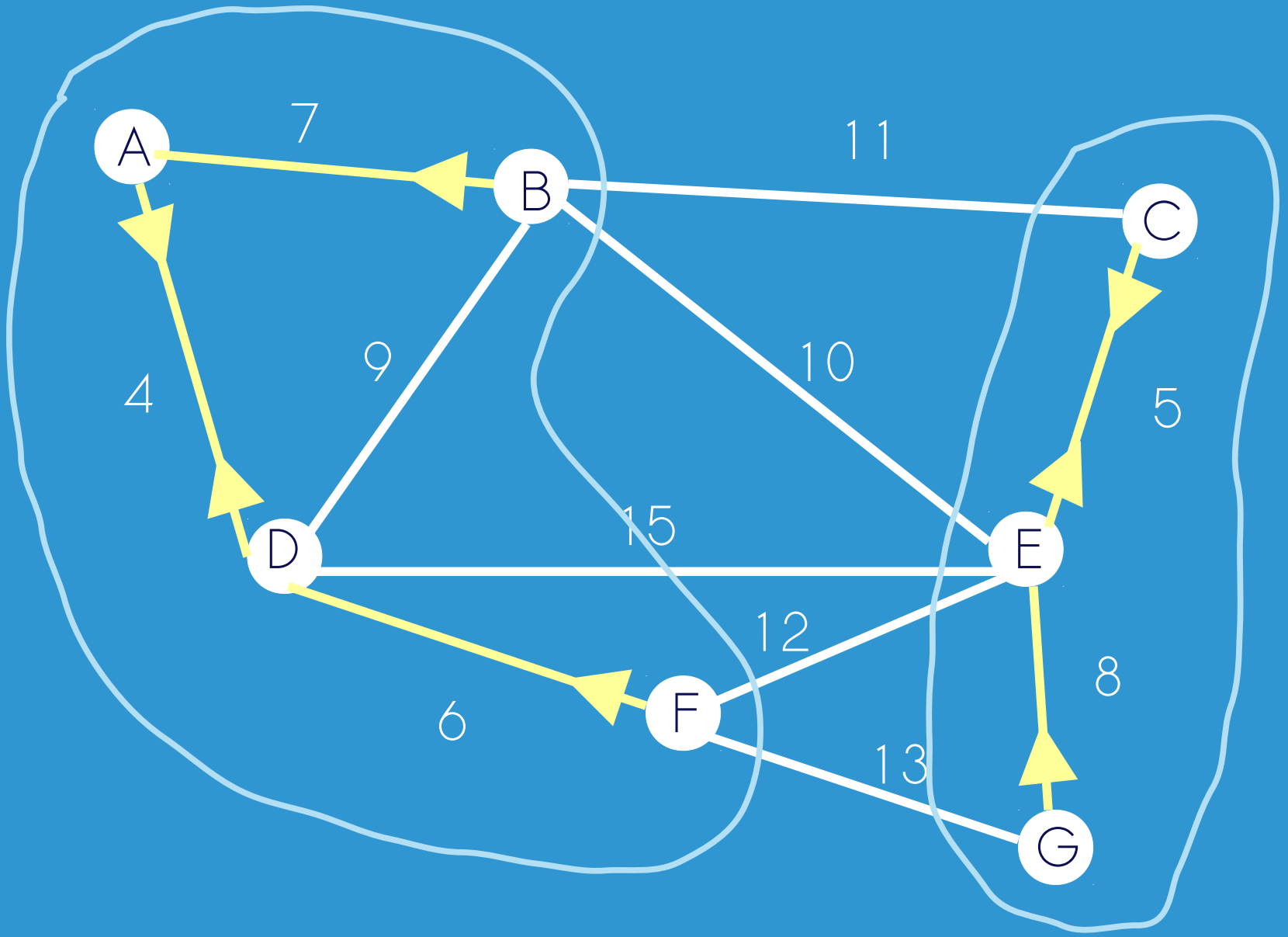
Initialize components



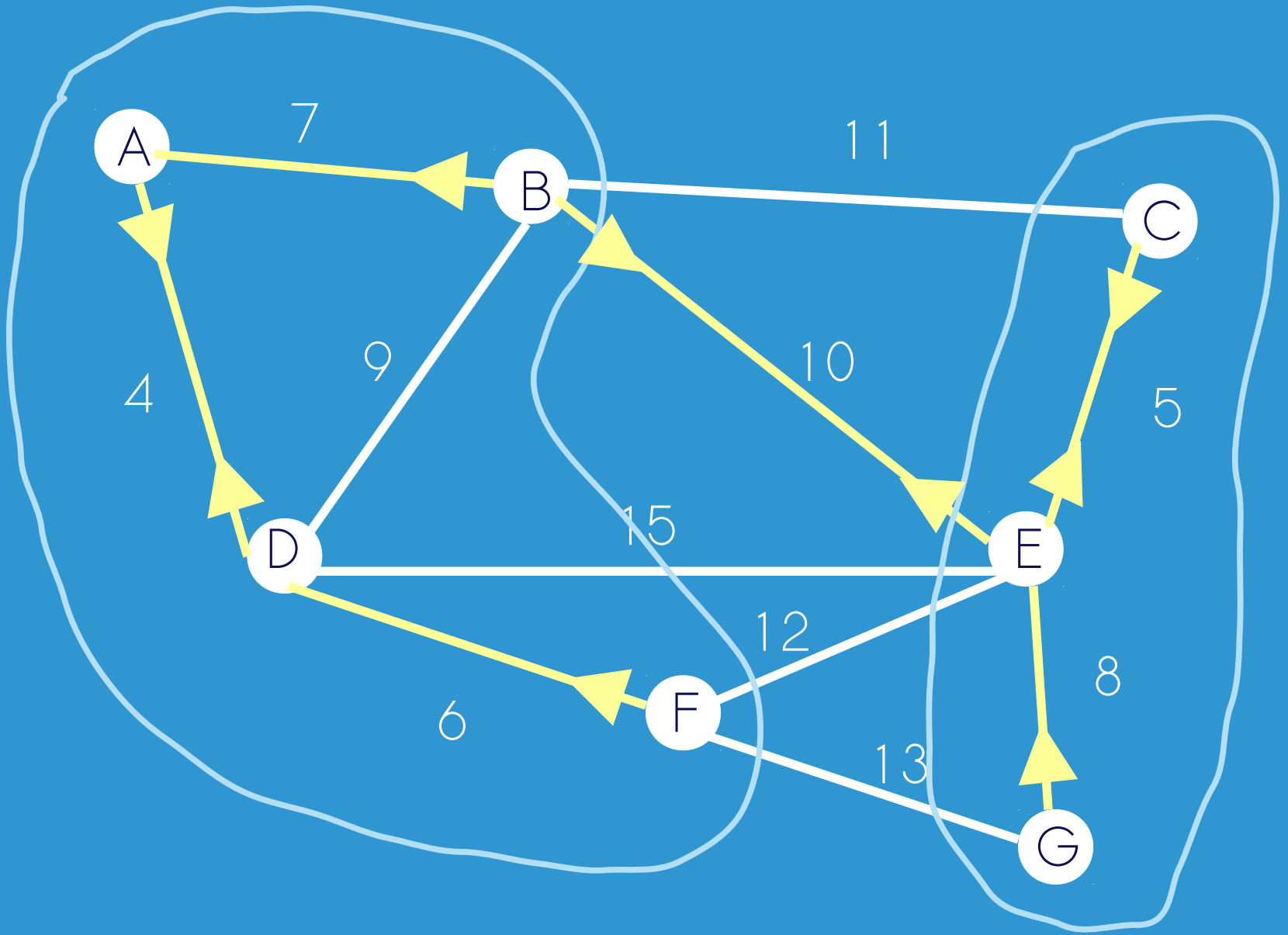
Finding MWOE



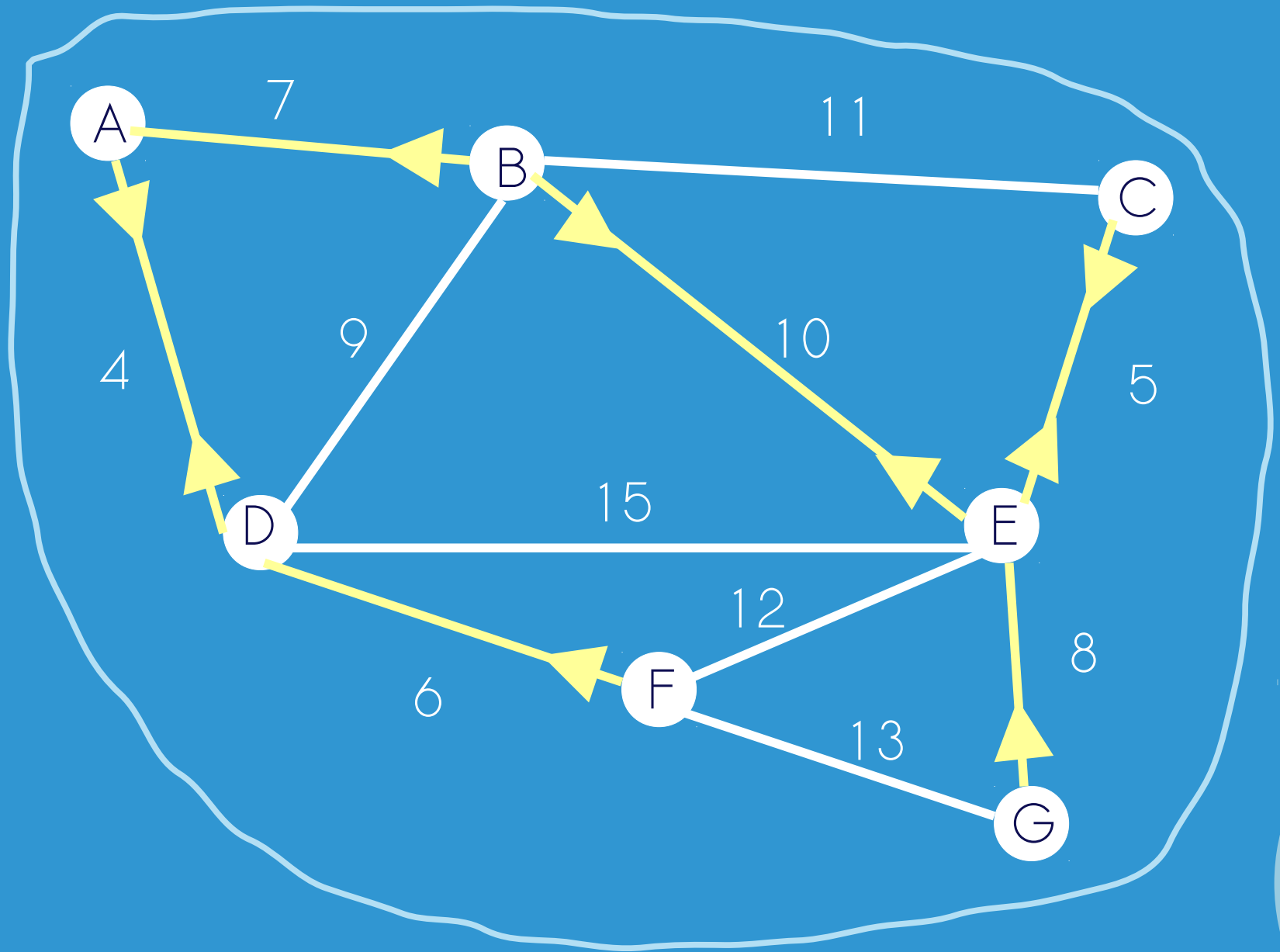
Creating new components



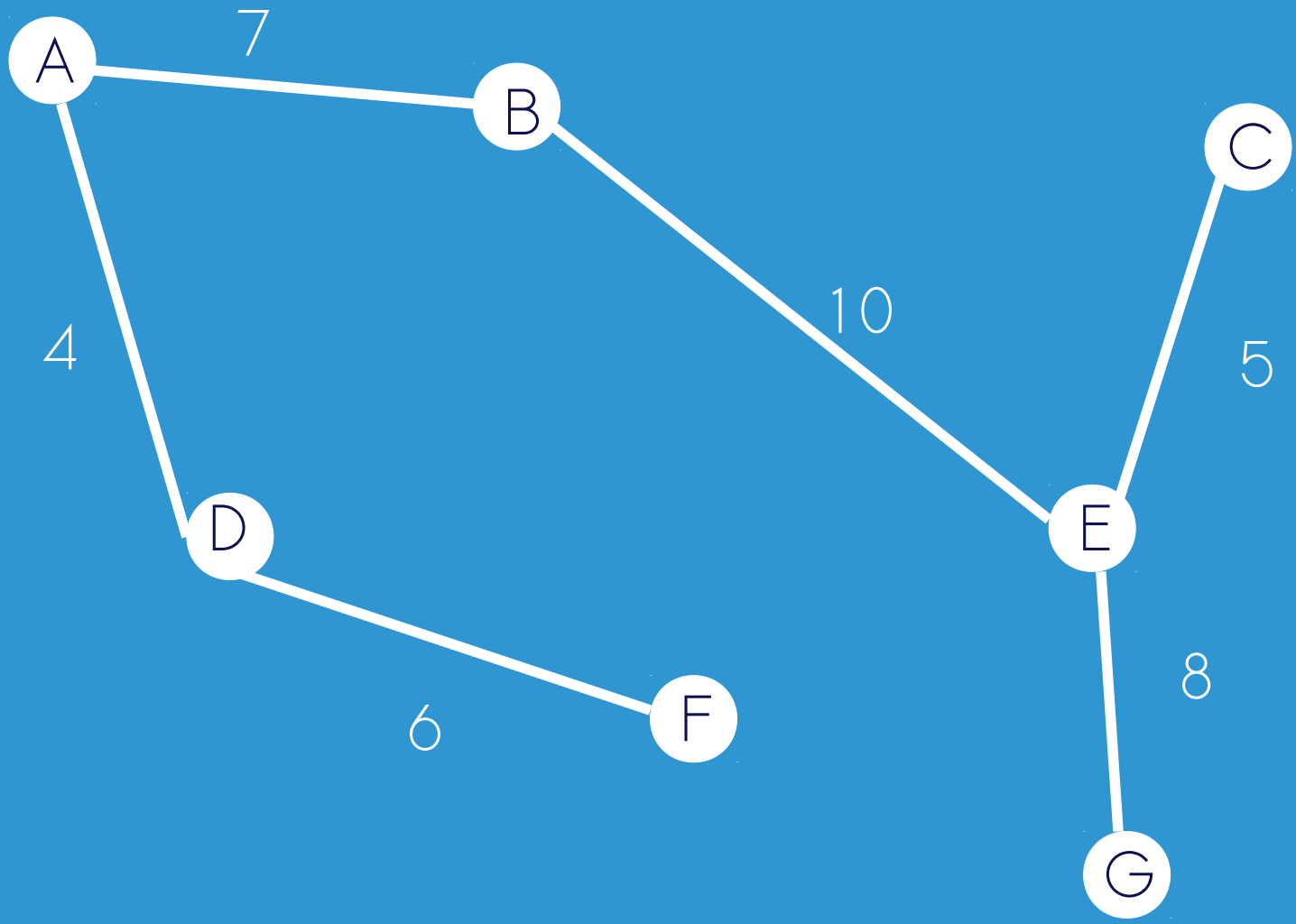
Finding MWOE



Creating new component

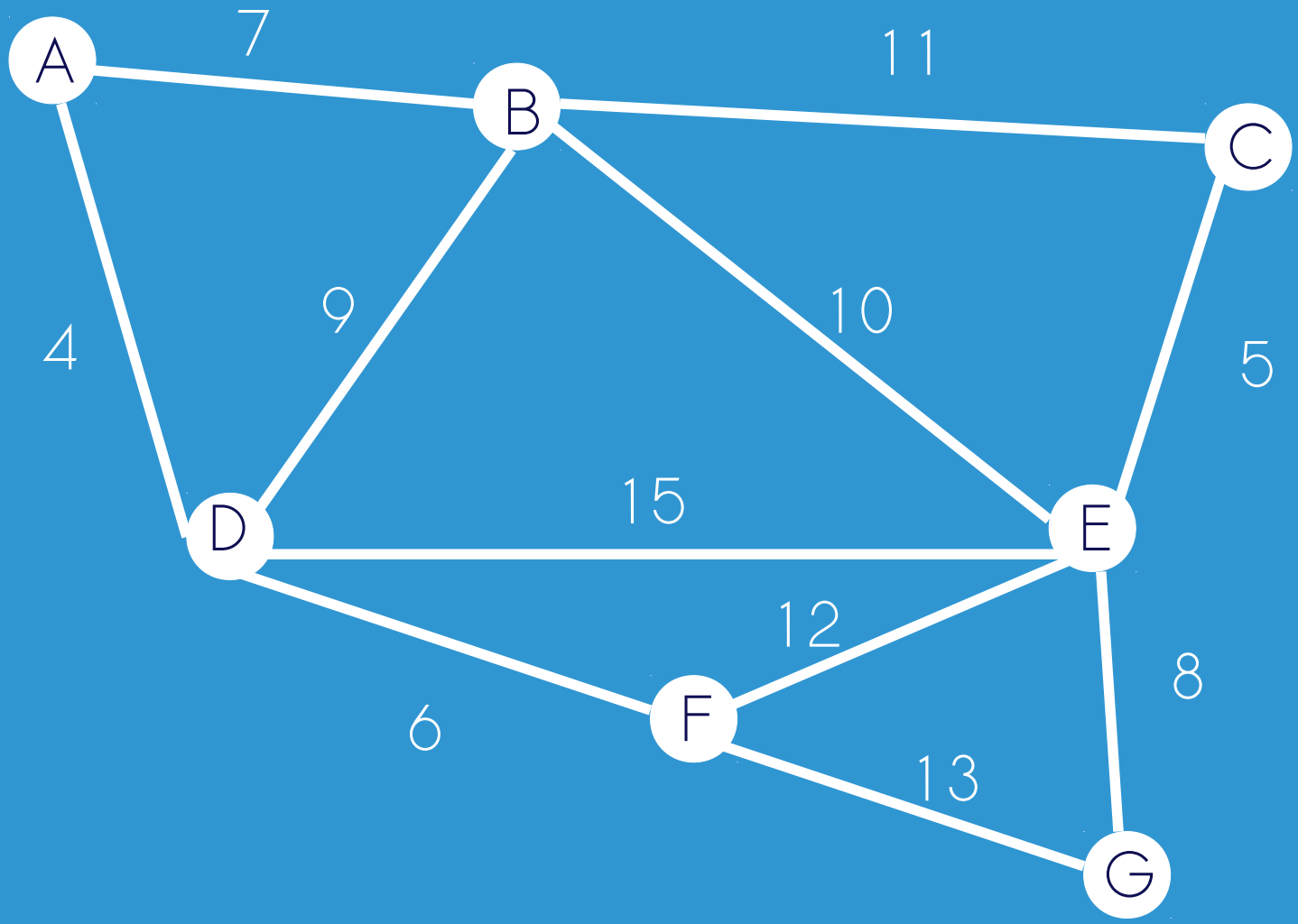


Here is the Minimum spanning tree



Bor-el algorithm (Parallel)

Example Graph



Edge list representation

MST

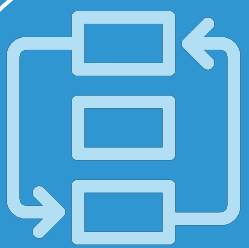
A B 7
A D 4
B A 7
B C 11
B D 9
B E 10
C B 11
C E 5
D A 4
D B 9
D E 15
D F 6
E B 10
E C 5
E D 15
E F 12
E G 8
F D 6
F E 12
F G 13
G E 8
G F 13

Select MWOE

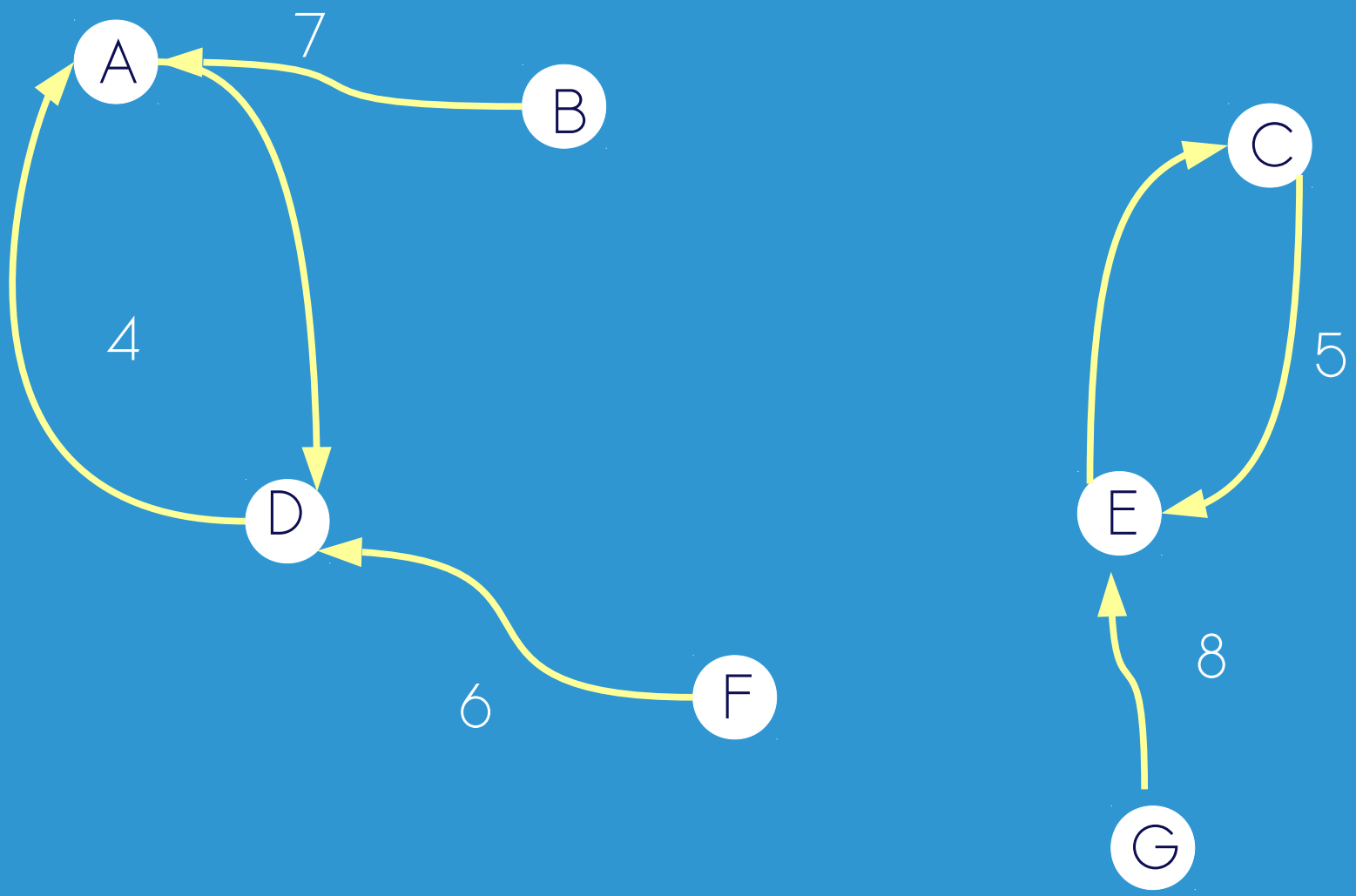
MST

A D 4
B A 7
C E 5
D A 4
E C 5
F D 6

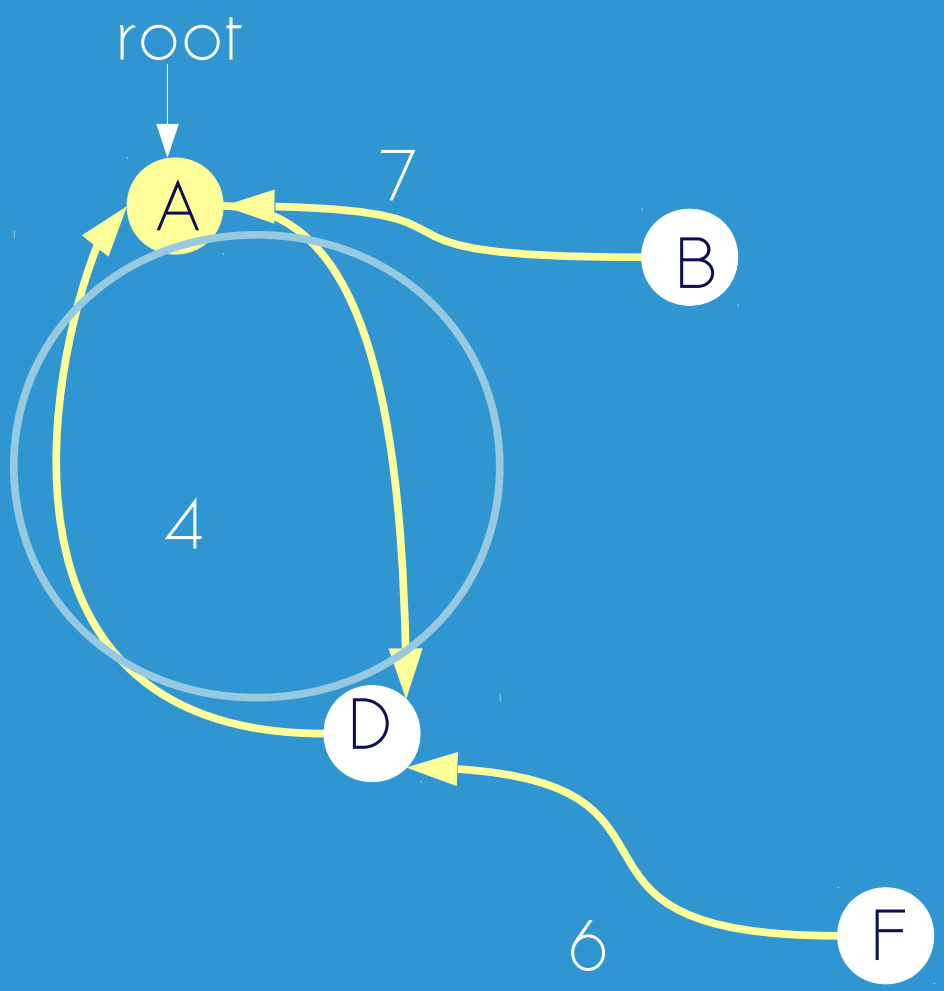
A B 7
A D 4
B A 7
B C 11
B D 9
B E 10
C B 11
C E 5
D A 4
D B 9
D E 15
D F 6
E B 10
E C 5
E D 15
E F 12
E G 8
F D 6
F E 12
F G 13
G E 8
G F 13

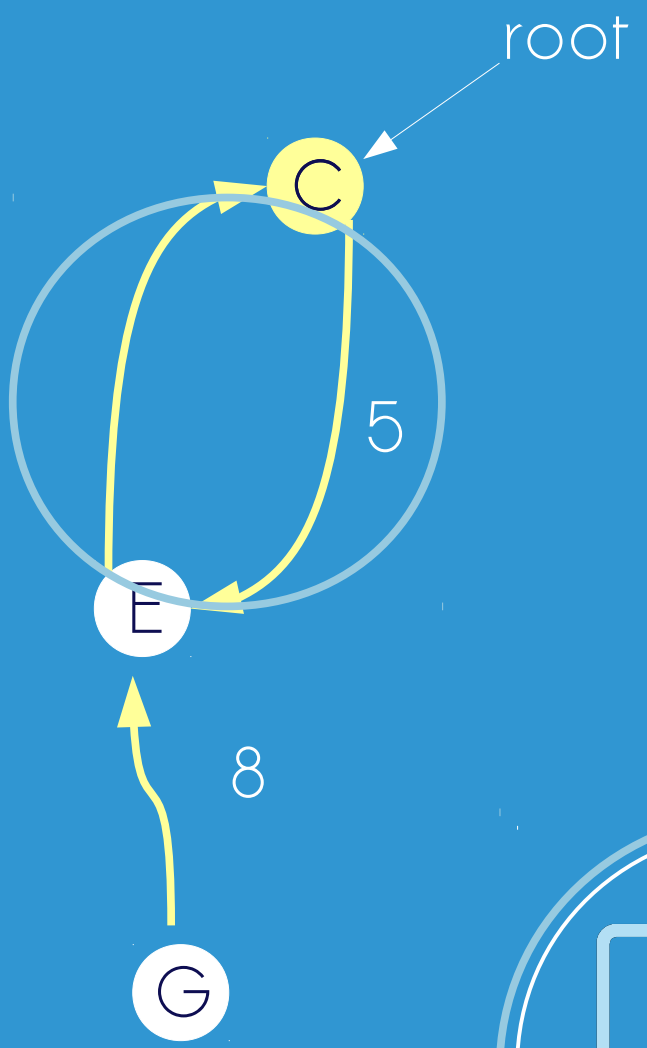


These are the edges we selected

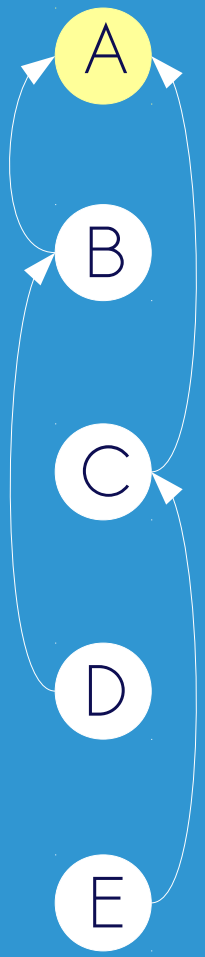


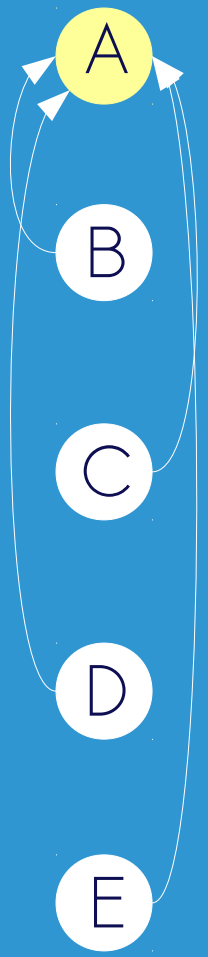
These are the edges we selected



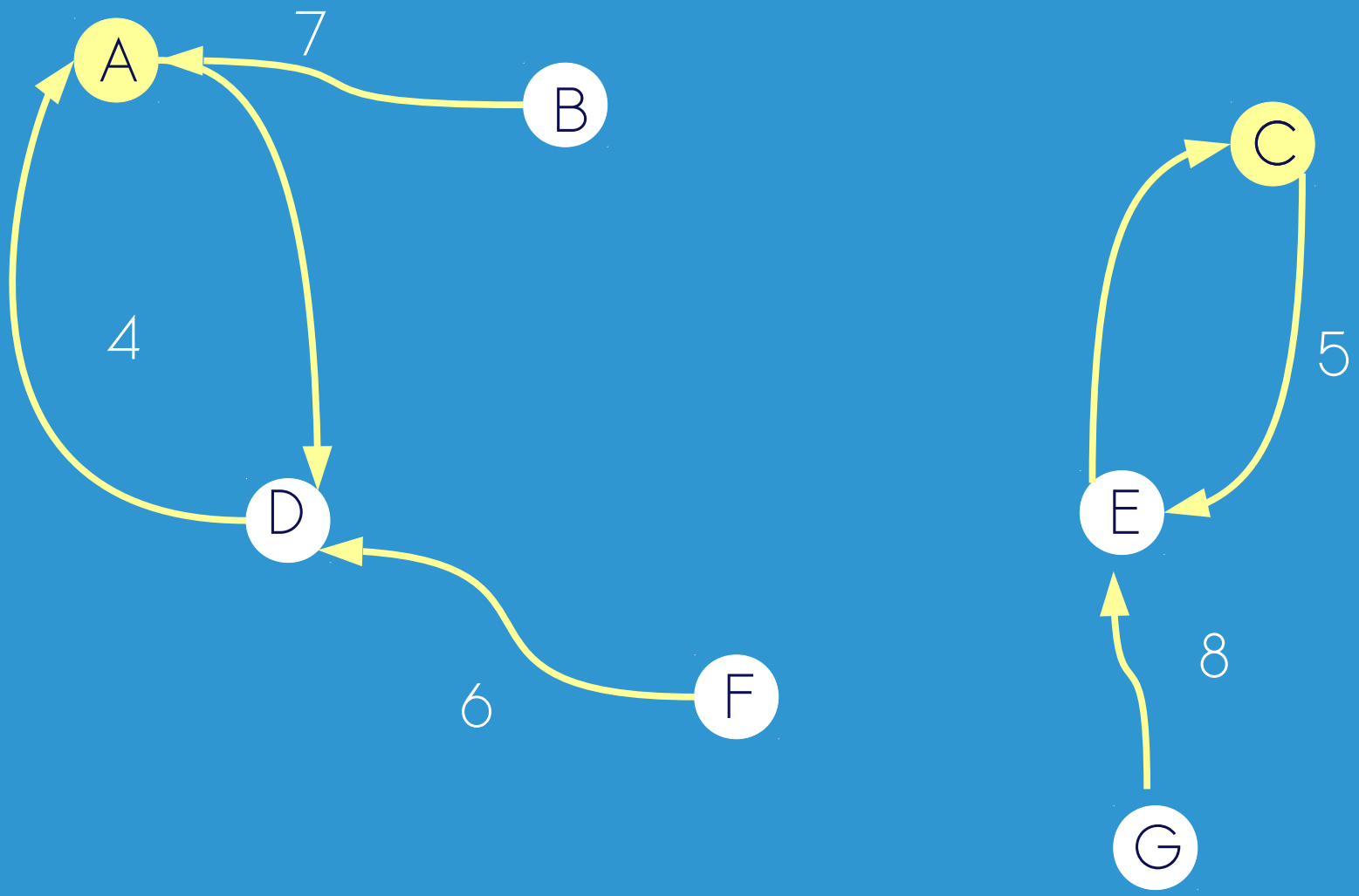


Pointer jumping example

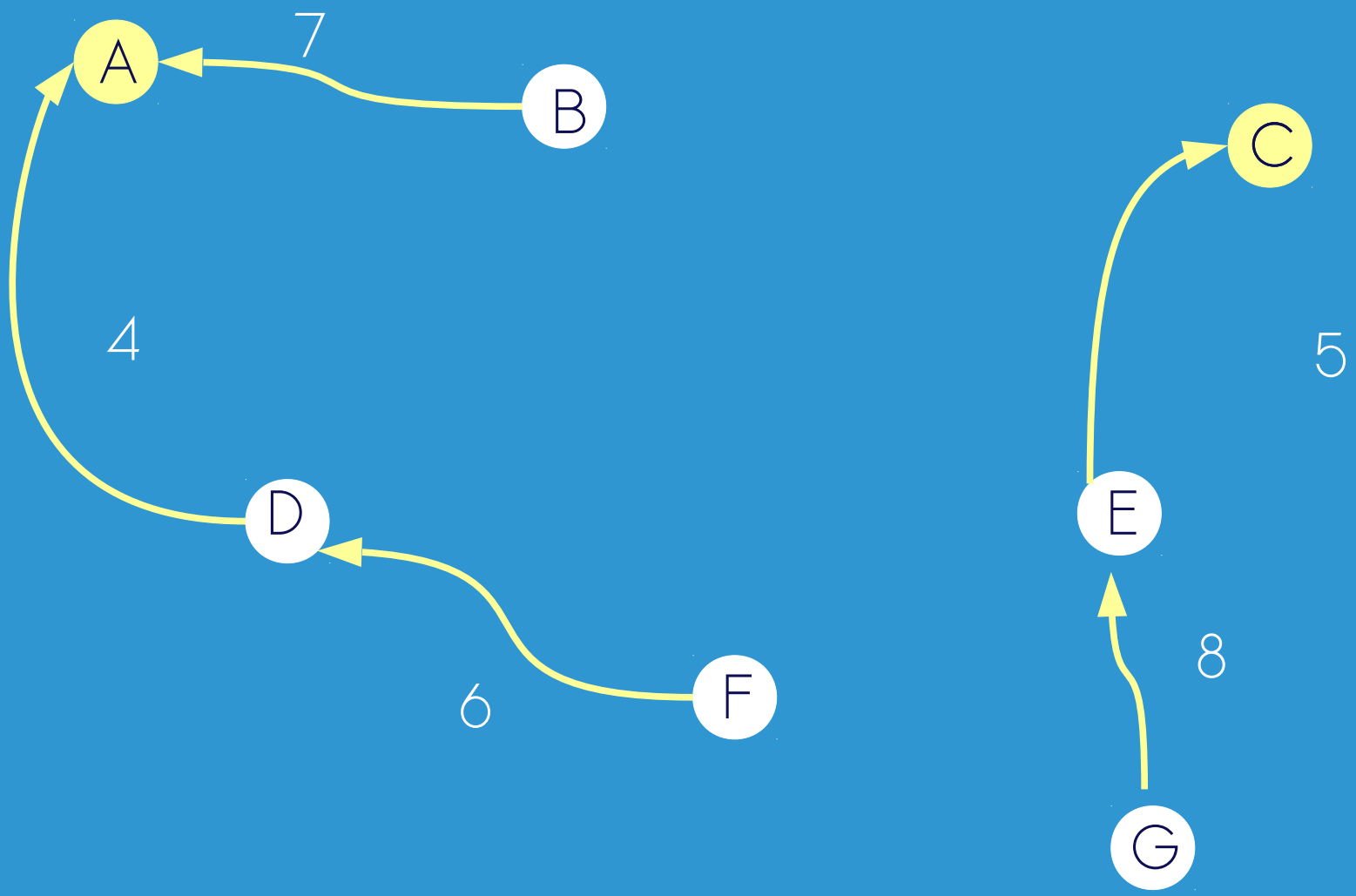




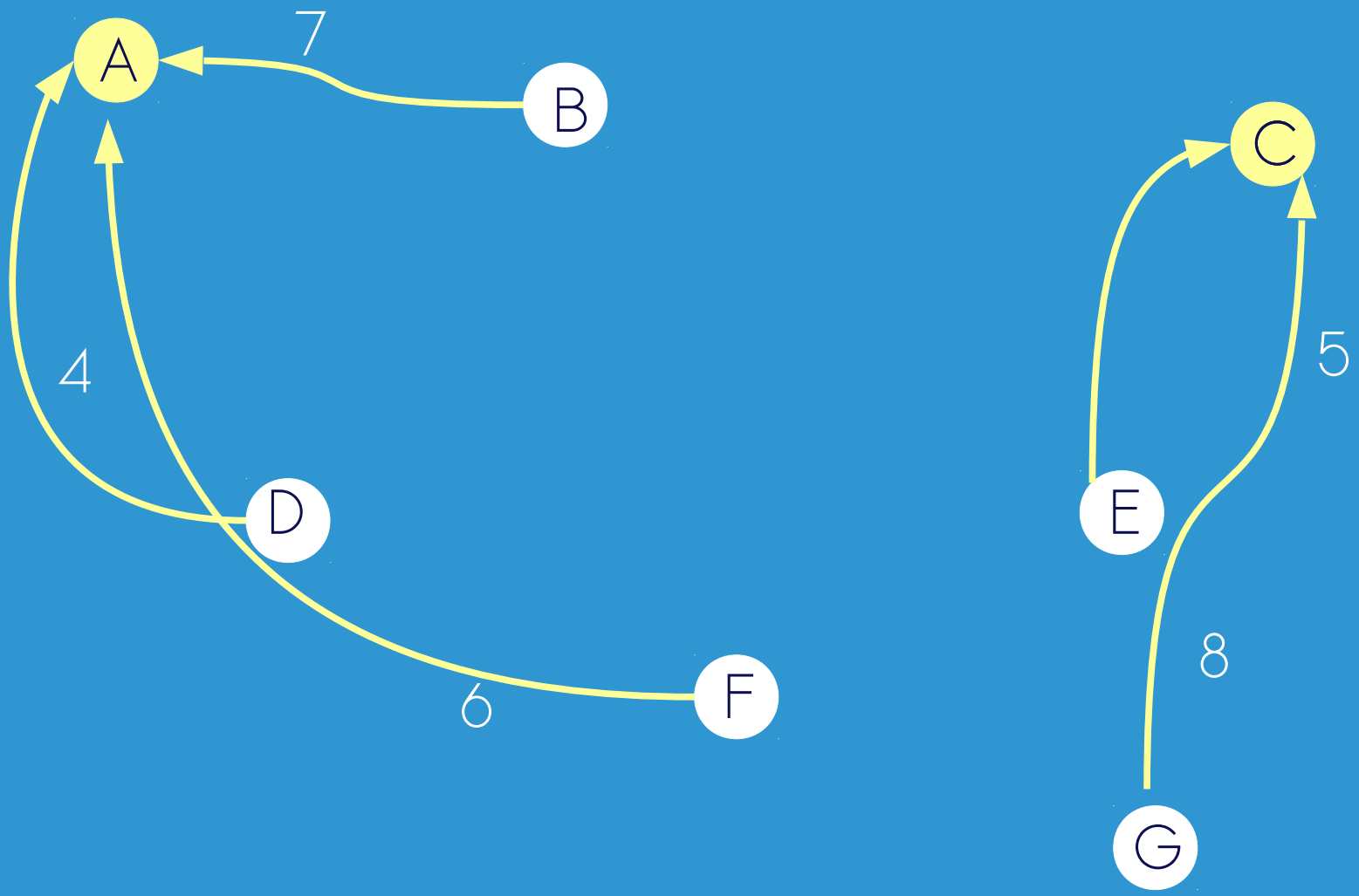
Pointer jumping



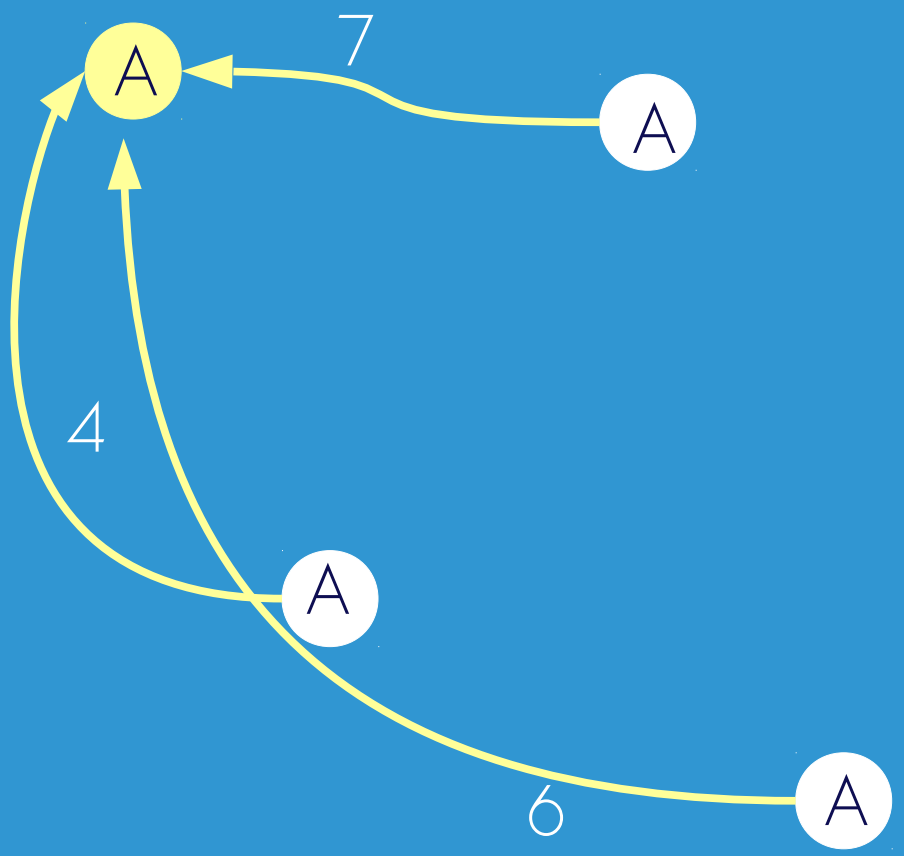
Pointer jumping

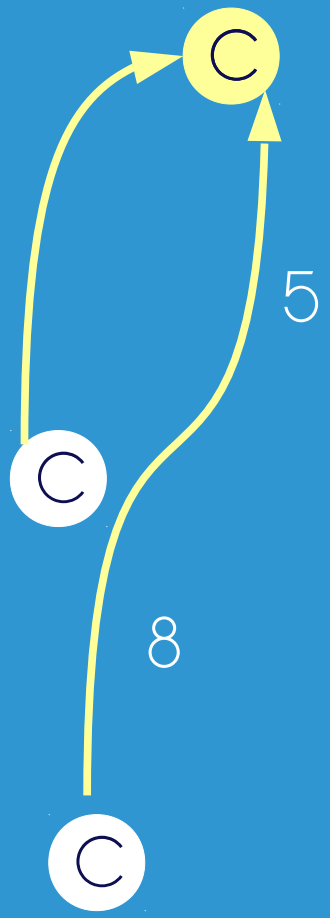


Pointer jumping



Create supervertex





In the edge list

MST

A D 4
B A 7
C E 5
D A 4
E C 5
F D 6

A B 7
A D 4
B A 7
B C 11
B D 9
B E 10
C B 11
C E 5
D A 4
D B 9
D E 15
D F 6
E B 10
E C 5
E D 15
E F 12
E G 8
F D 6
F E 12
F G 13
G E 8
G F 13

In the edge list

MST

A D 4
B A 7
C E 5
D A 4
E C 5
F D 6

A A 7
A A 4
A A 7
A C 11
A A 9
A C 10
C A 11
C C 5
A A 4

A A 9
A C 15
A A 6
C A 10
C C 5
C A 15
C A 12
C C 8
A A 6

A C 12
A C 13
C C 8
C A 13

Compact

MST

A D 4
B A 7
C E 5
D A 4
E C 5
F D 6

A C 11
A C 10
C A 11

A C 15
C A 10
C A 15
C A 12

A C 12
A C 13
C A 13

Find Mwoe

MST

A D 4
B A 7
C E 5
D A 4
E C 5
F D 6
B E 10

A C 11
A C 10
C A 11

A C 15
C A 10
C A 15
C A 12

A C 12
A C 13
C A 13

Found Spanning tree

A D 4
B A 7
C E 5
D A 4
E C 5
F D 6
B E 10

Theoretical analysis of Bor-el

Size of graph in memory

Number of edges

N : number of nodes

log(N) size of one node in memory

$$\frac{4 * E * \log(N)}{p} + \frac{2 * E * \log(E)}{p}$$

Number of processors

2 times each edge

2 nodes id per edge

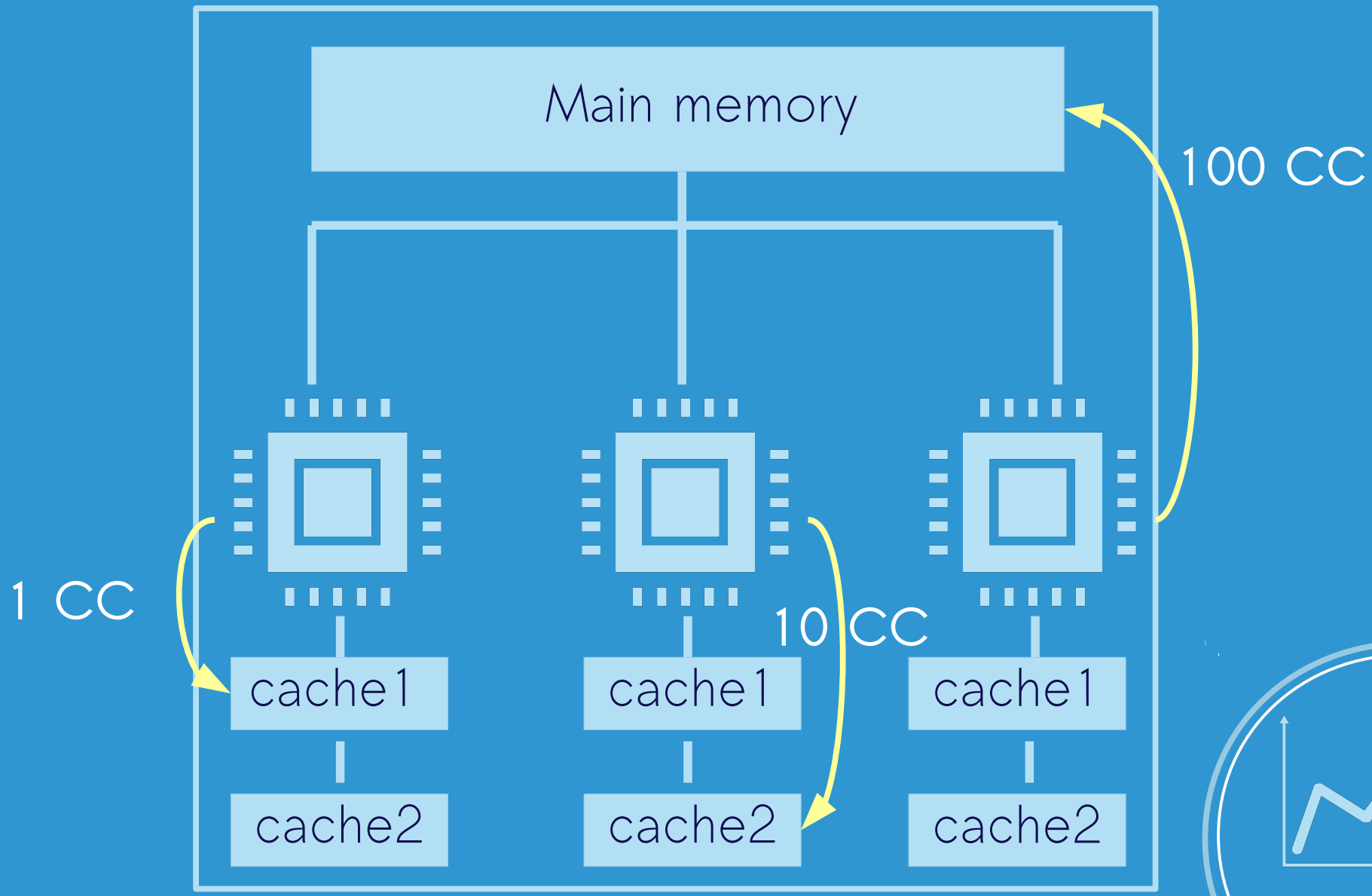
Size of weights in memory

Average number of edges

E decreases of at least N/2 each iteration. Lets say E = kN

$$E = k \cdot N - \frac{N}{4} \cdot (2 \cdot k + 1)$$

Memory access time



Memory access time

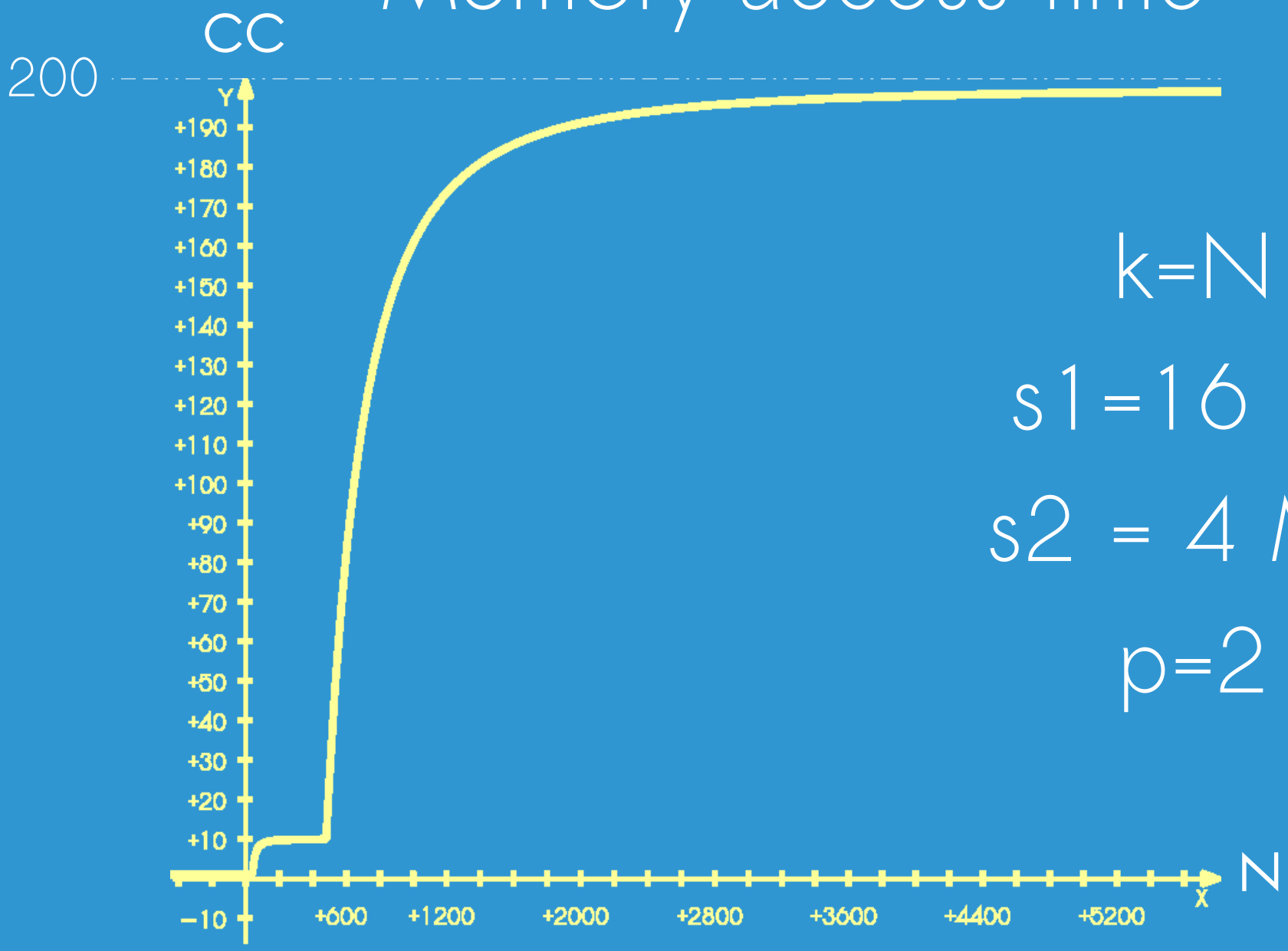
Size of cache 1

Size of cache 2

$$min\left(1,\frac{s1}{S}\right)\cdot 1+\left(1-min\left(1,\frac{s1}{S}\right)\right)\cdot\left(min\left(1,\frac{s2}{S}\right)\cdot 10+\left(1-min\left(1,\frac{s2}{S}\right)\right)\cdot 100\cdot p\right)$$

Size of graph in memory

Memory access time



k=N

s1=16 kb

s2 = 4 Mb

p=2

Number of memory accesses

Formula given by the paper on bor-el

$$\left(\frac{8E+N+Nlog(N)}{p}+\frac{4EClog\left(\frac{2E}{P}\right)}{p}\right)\log(N)$$

C is an **unknown constant** : using their **experimental results**
we fount it is around **3.21**

Computation complexity

Formula given by the paper on bor-el

$$\left(\frac{E}{p}\right)\log\left(E\right)\log\left(N\right)$$

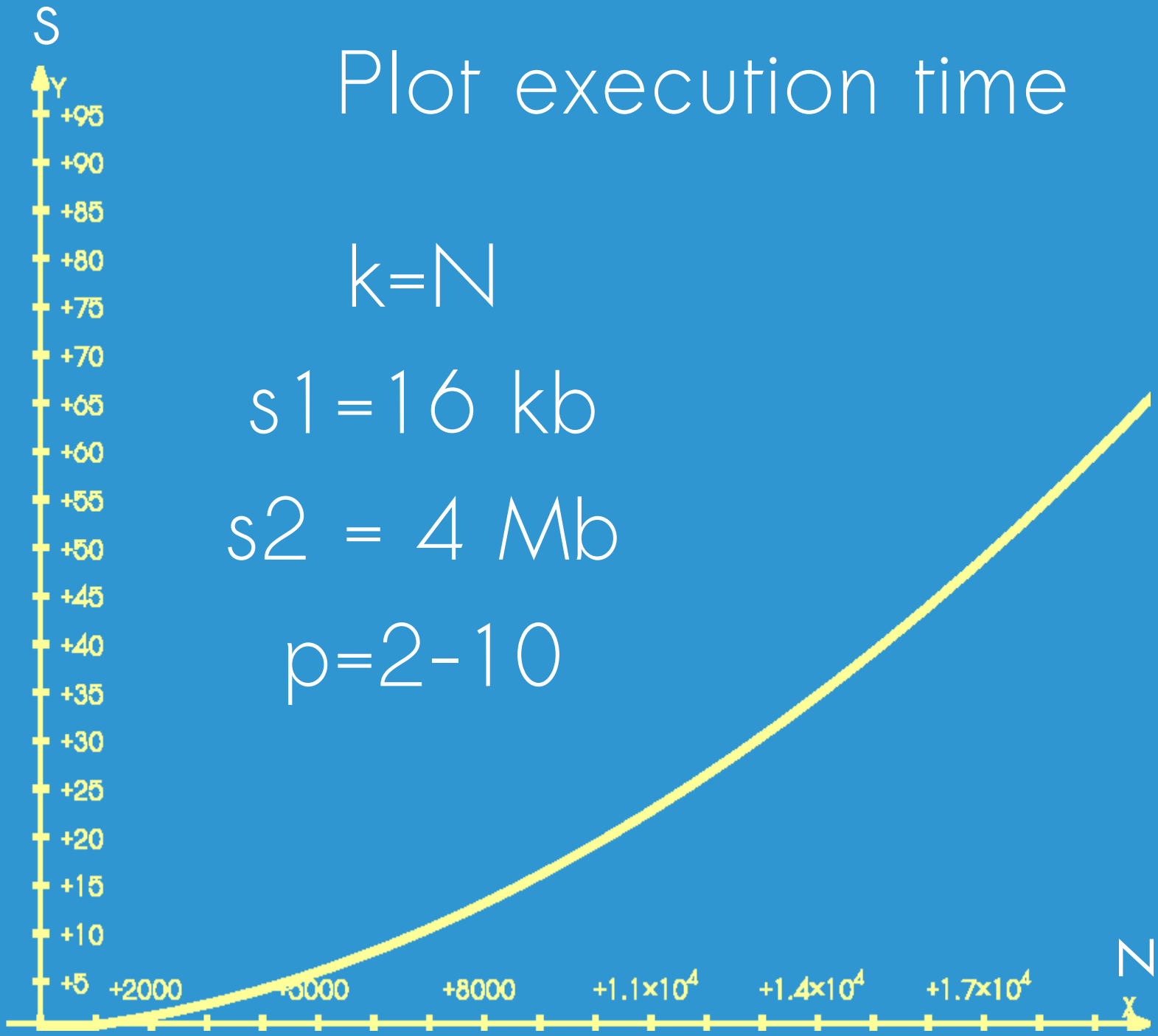
Plot execution time

k=N

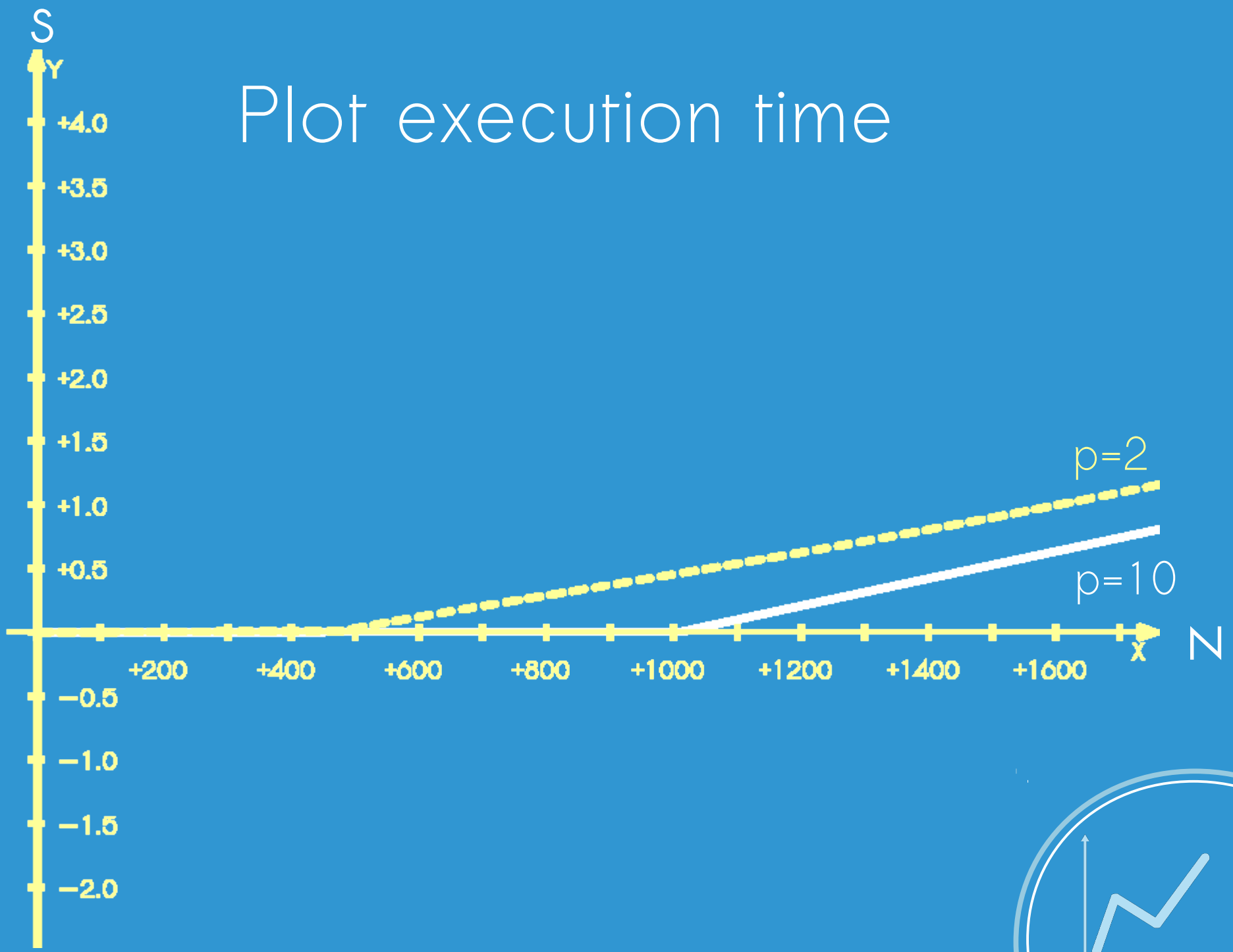
s1=16 kb

s2 = 4 Mb

p=2-10



Plot execution time

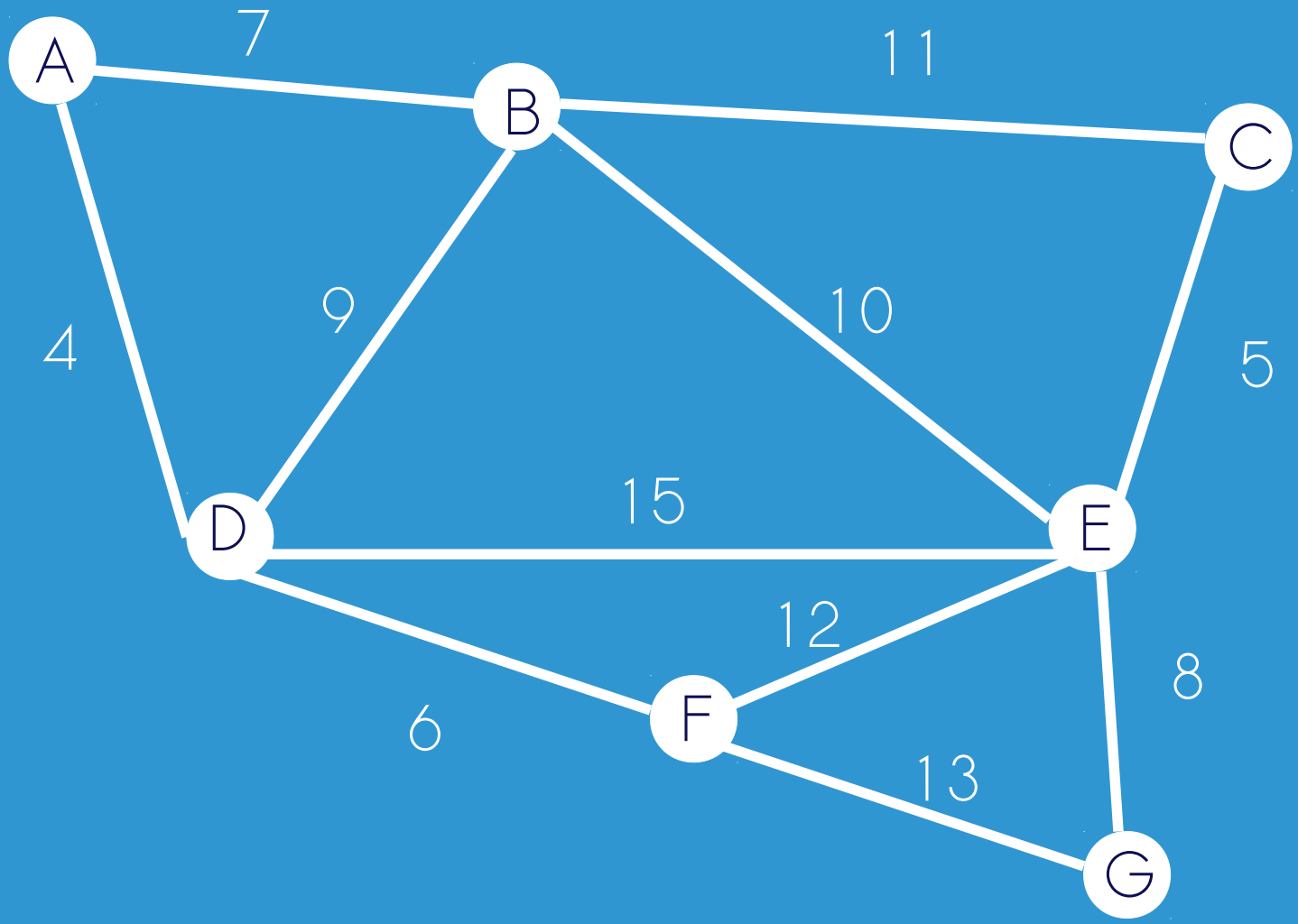


Analysis

Plot does not vary with p because time highly dominated by memory access for very big graphs

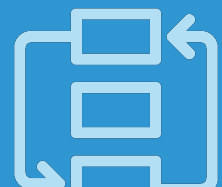
GHS algorithm (Distributed)

Example graph



State of each edge

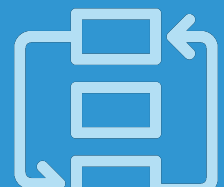
- **Branch** edges are those that have already been determined to be part of the MST.
- **Rejected** edges are those that have already been determined not to be part of the MST.
- **Basic** edges are neither branch edges nor rejected edges.



State of each edge

Each processor stores:

- The state of any of its incident edges, which can be either of {basic, branch, reject}
- Identity of its fragment (the weigth of a core edge - for single-node fragments, the proc. id)
- Local MWOE
- MWOE for each branching-out edge
- Parent channel (route towards the root)
- MWOE channel (route towards the MWOE of its appended subfragment)

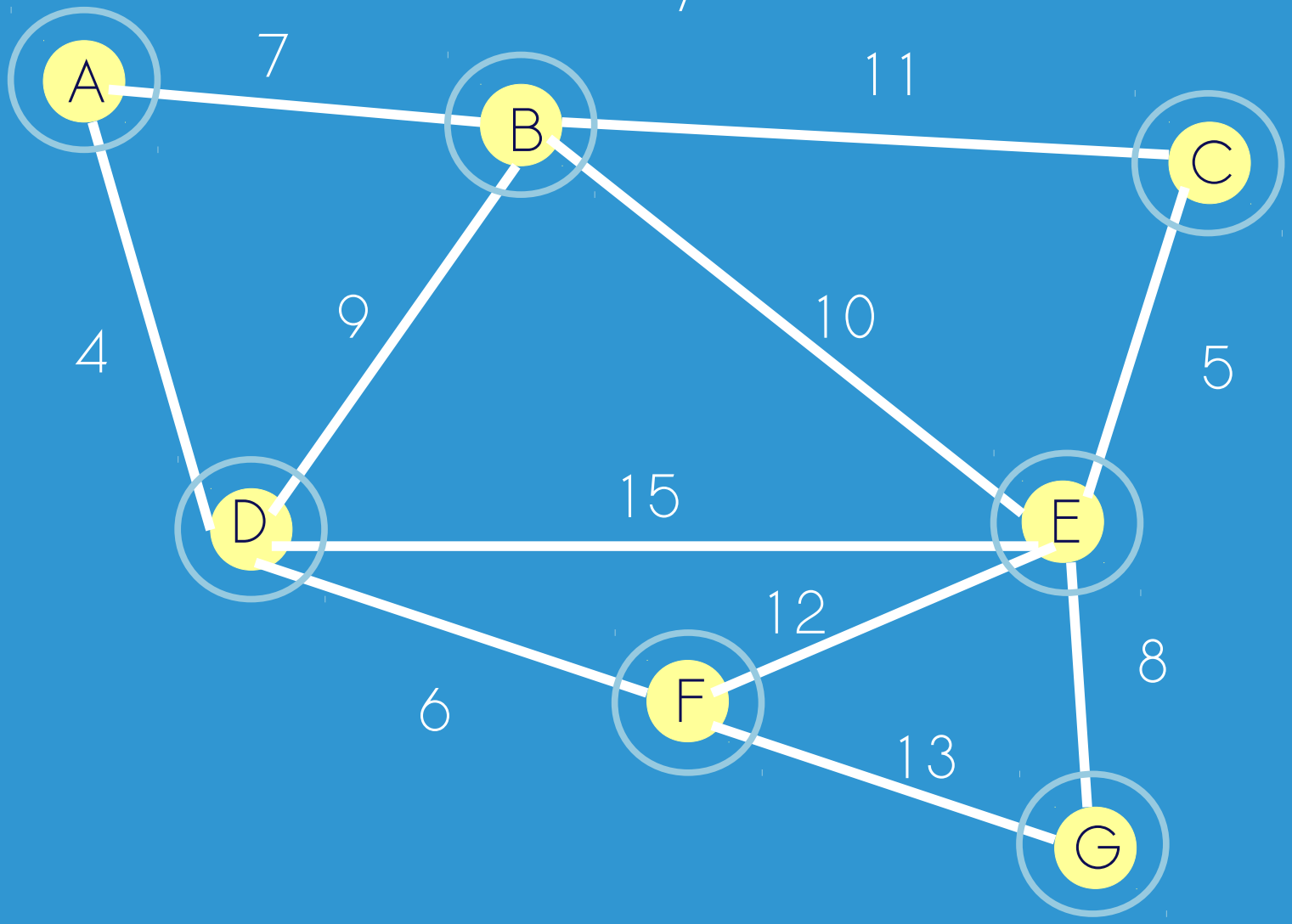


Type of messages

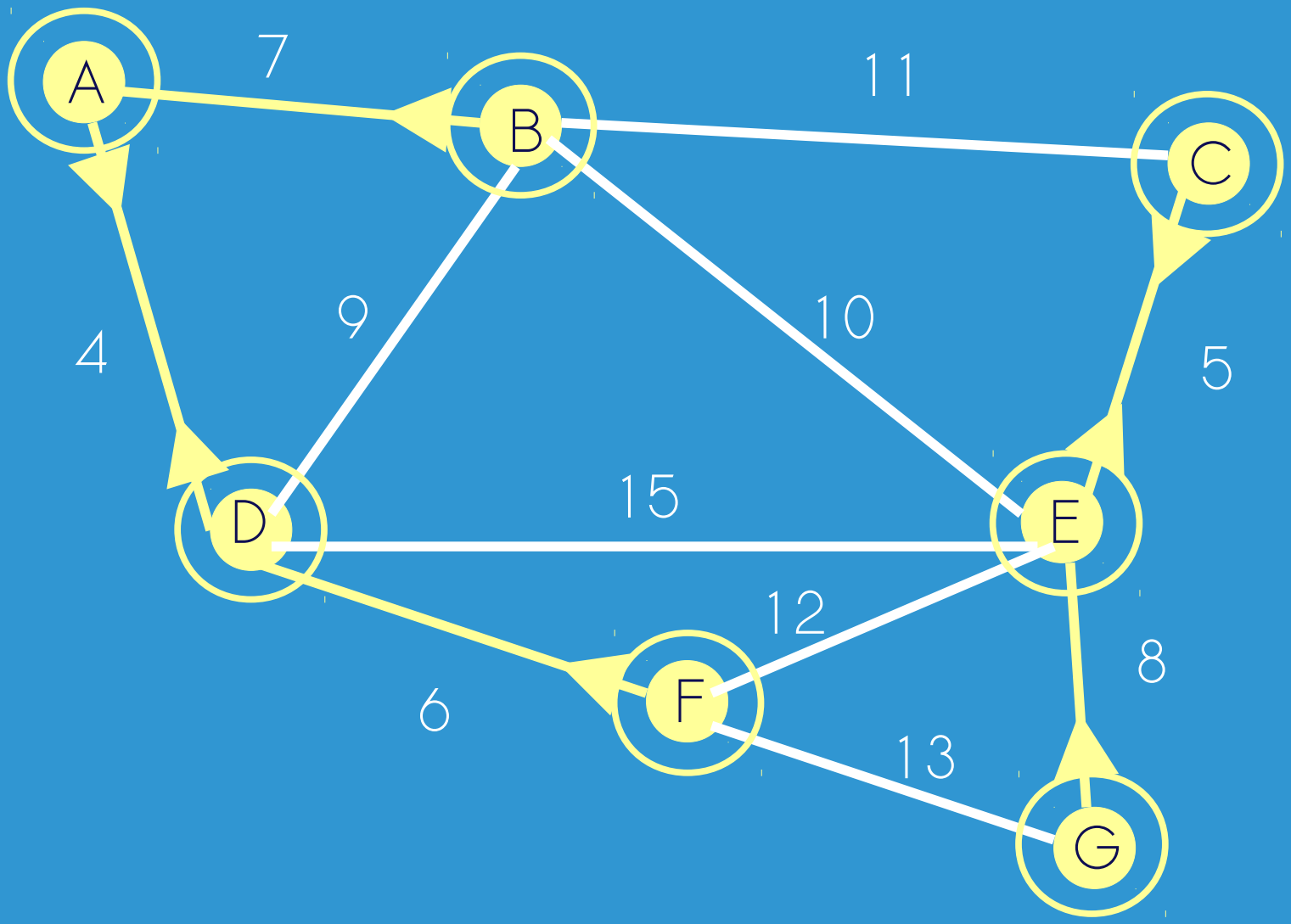
- **New fragment(identity):** coordination message sent by the root at the end of a phase
- **Test(identity):** for checking the status of a basic edge
- **Reject**, **Accept**: response to **Test**
- **Report(weight):** for reporting to the parent node the MWOE of the appended subfragment
- **Merge:** sent by the root to the node incident to the MWOE to activate union of fragments
- **Connect(My Id):** sent by the node incident to the MWOE to perform the union

Phase 0 : Every node is a fragment

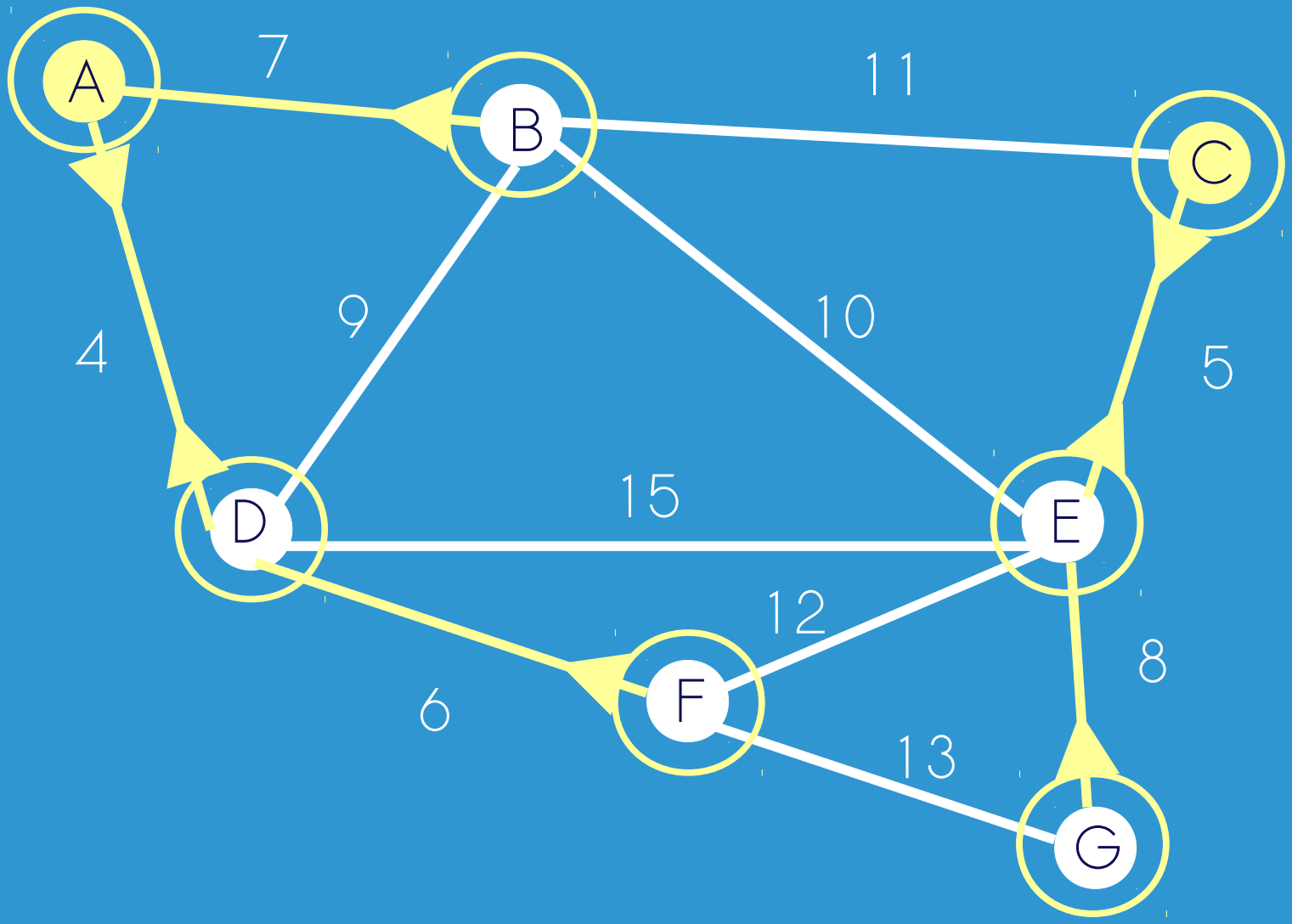
...And every node is the root of its fragment



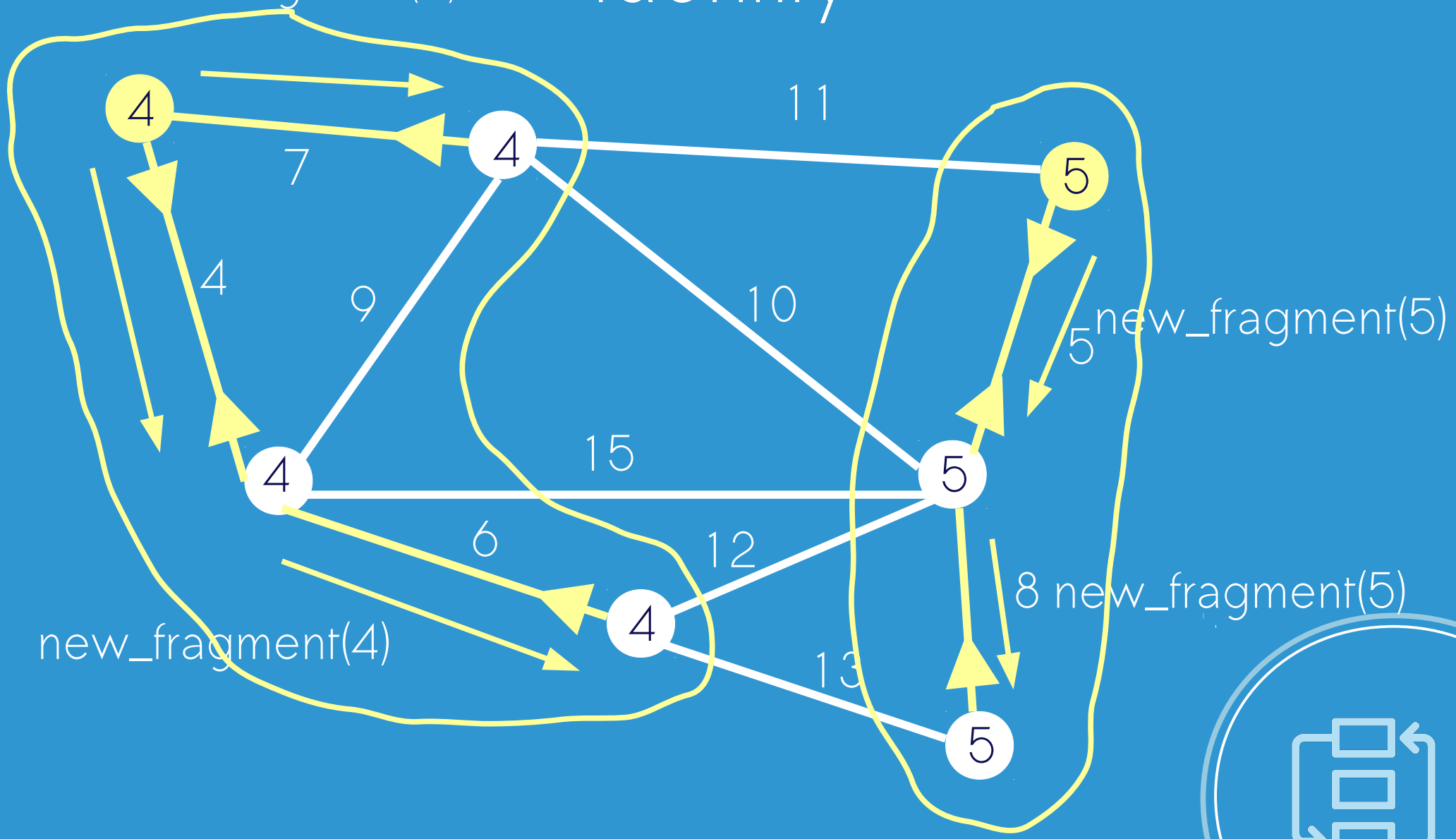
Phase 1 : Find MWOE



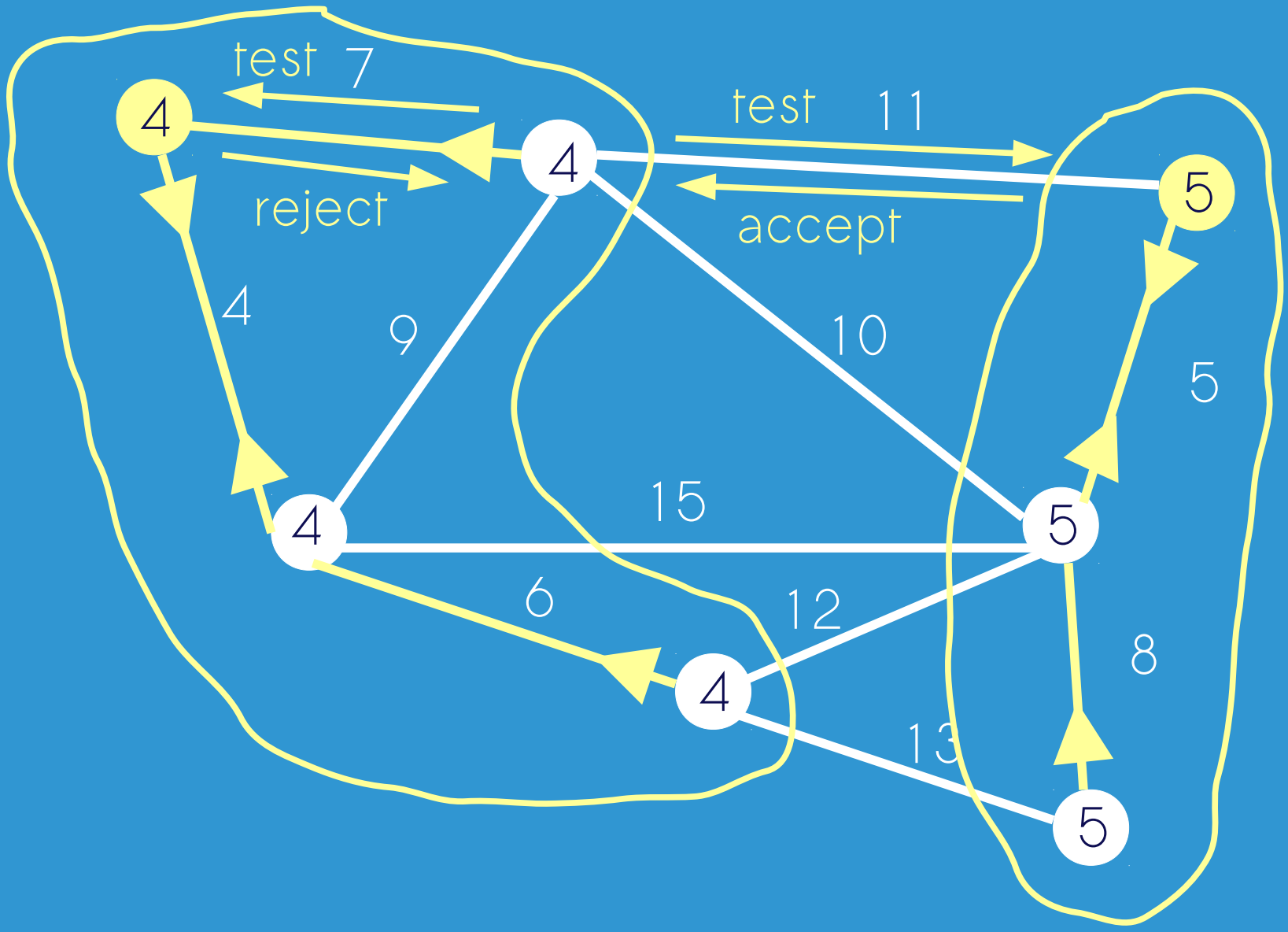
Phase 1 : select new root



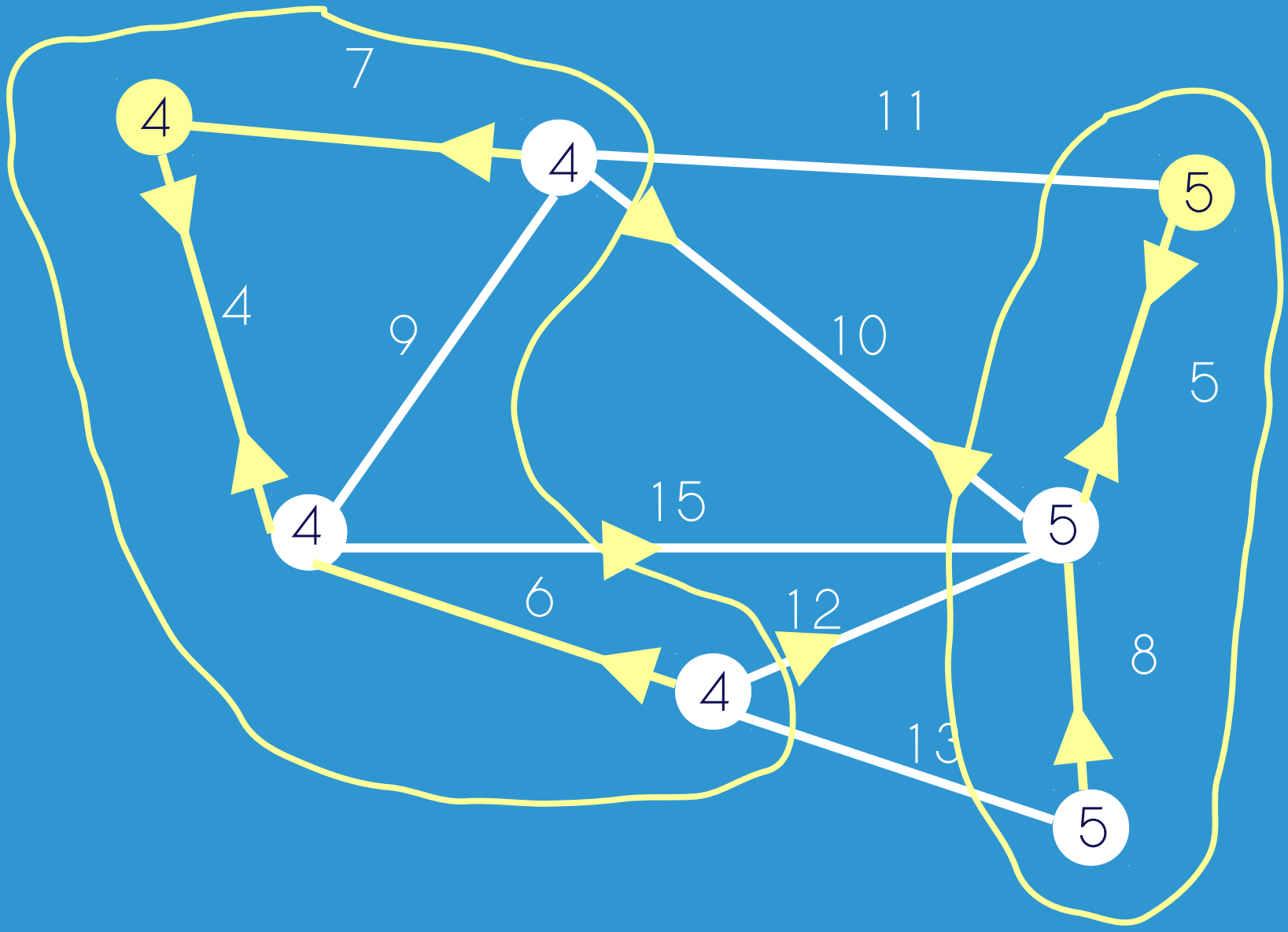
Phase 1 : root broadcast new identity



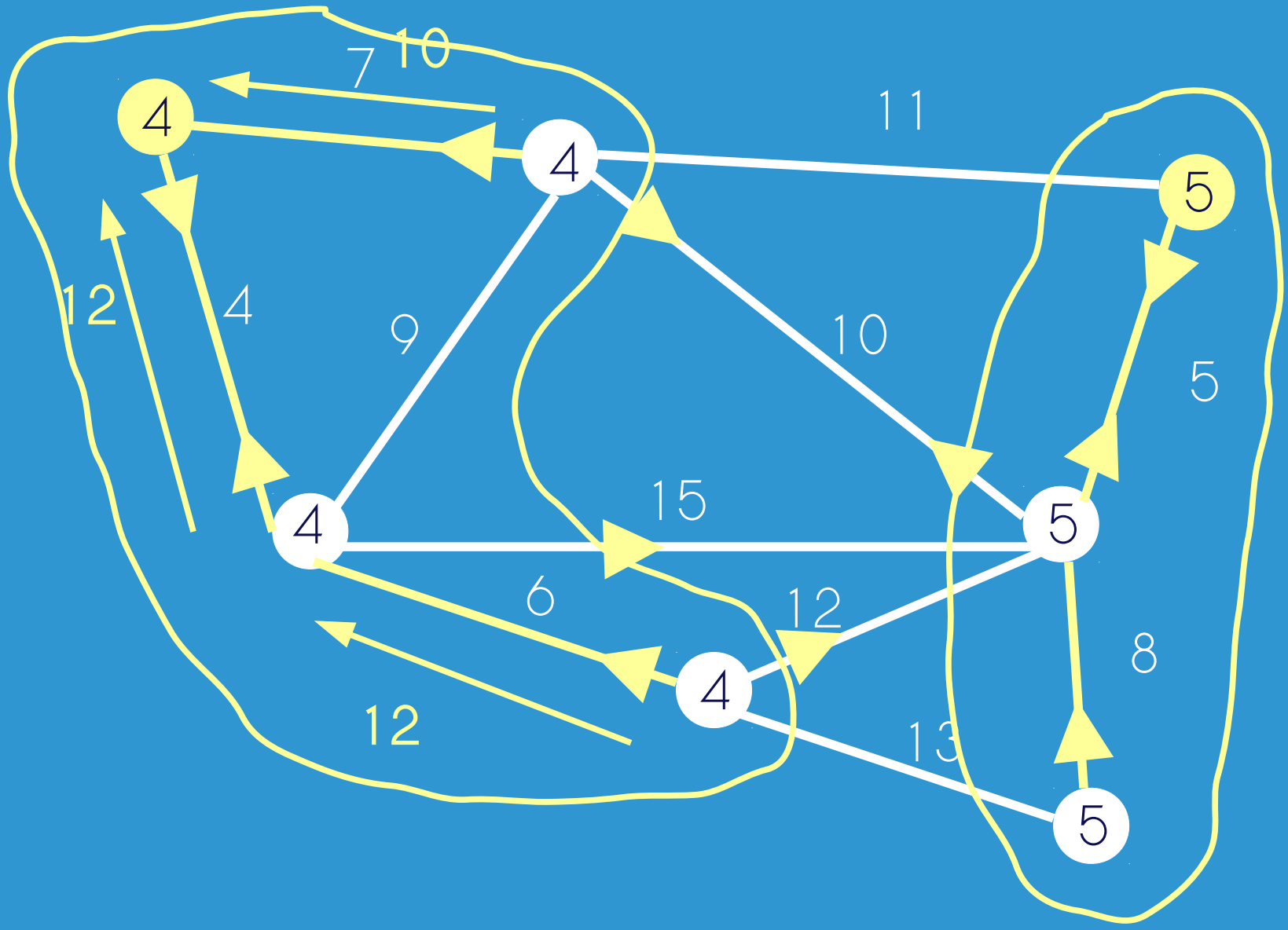
Phase 1 : Find MWOE



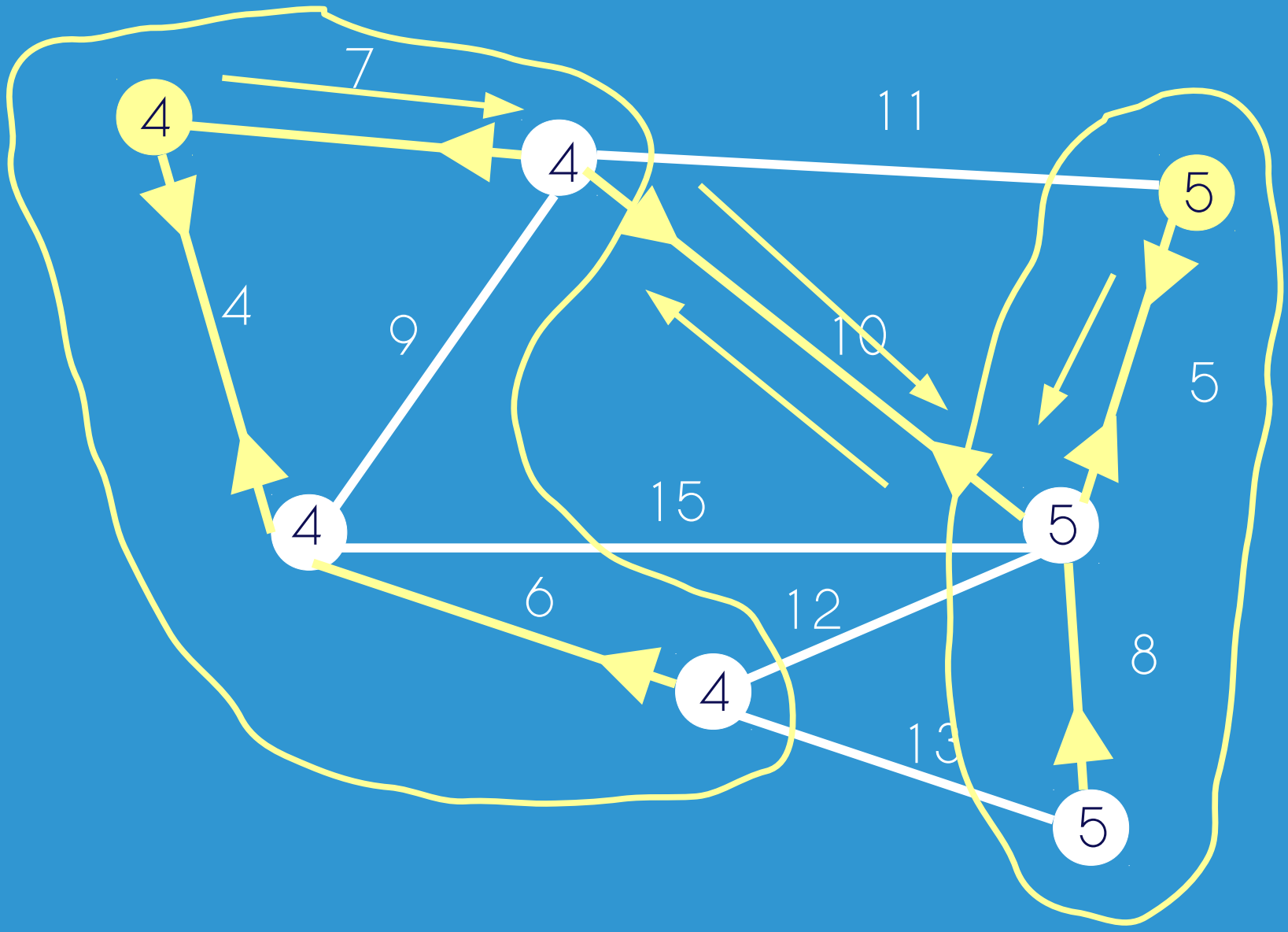
Phase 1 : Find MWOE



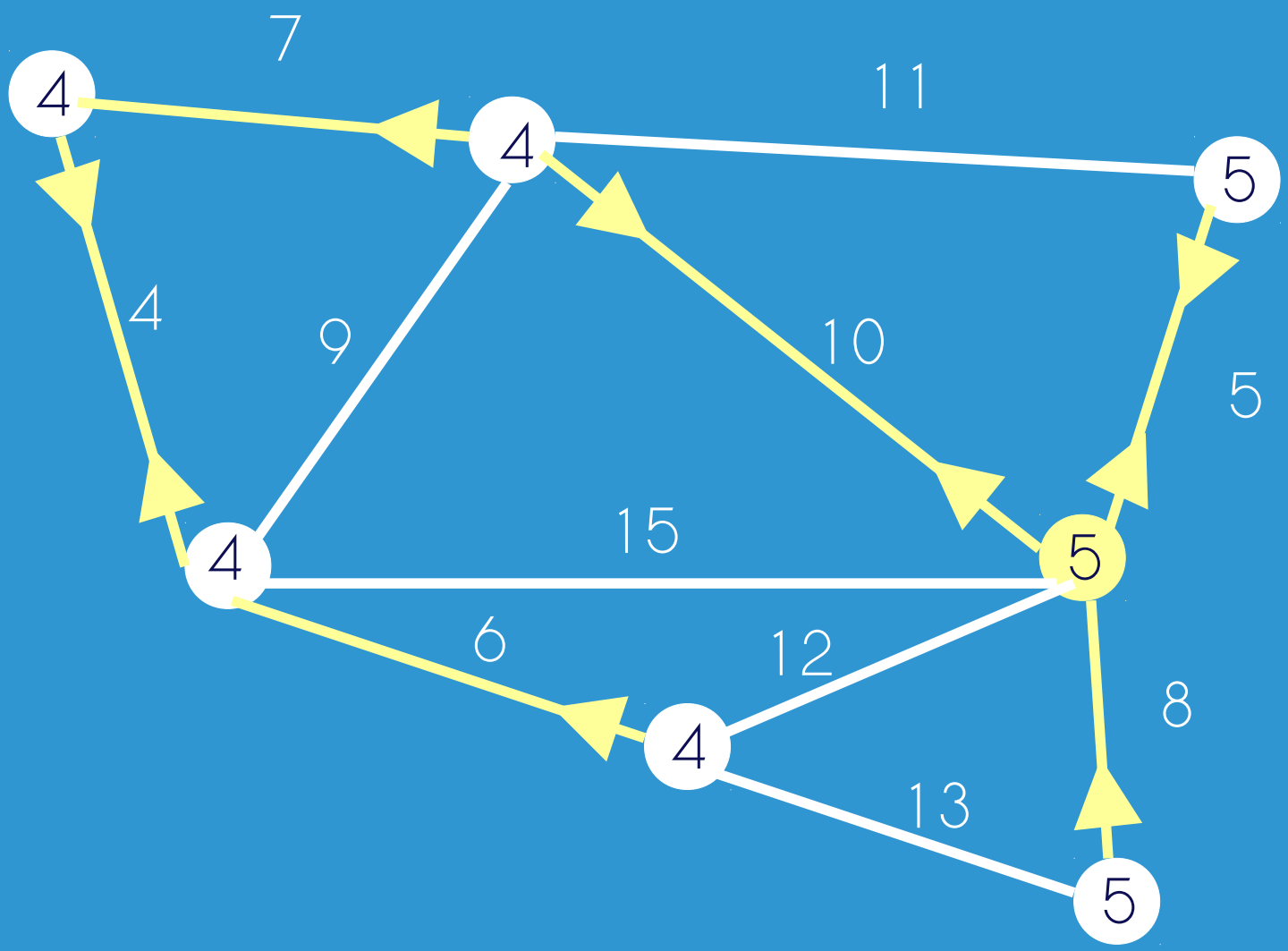
Phase 1 : Report to root



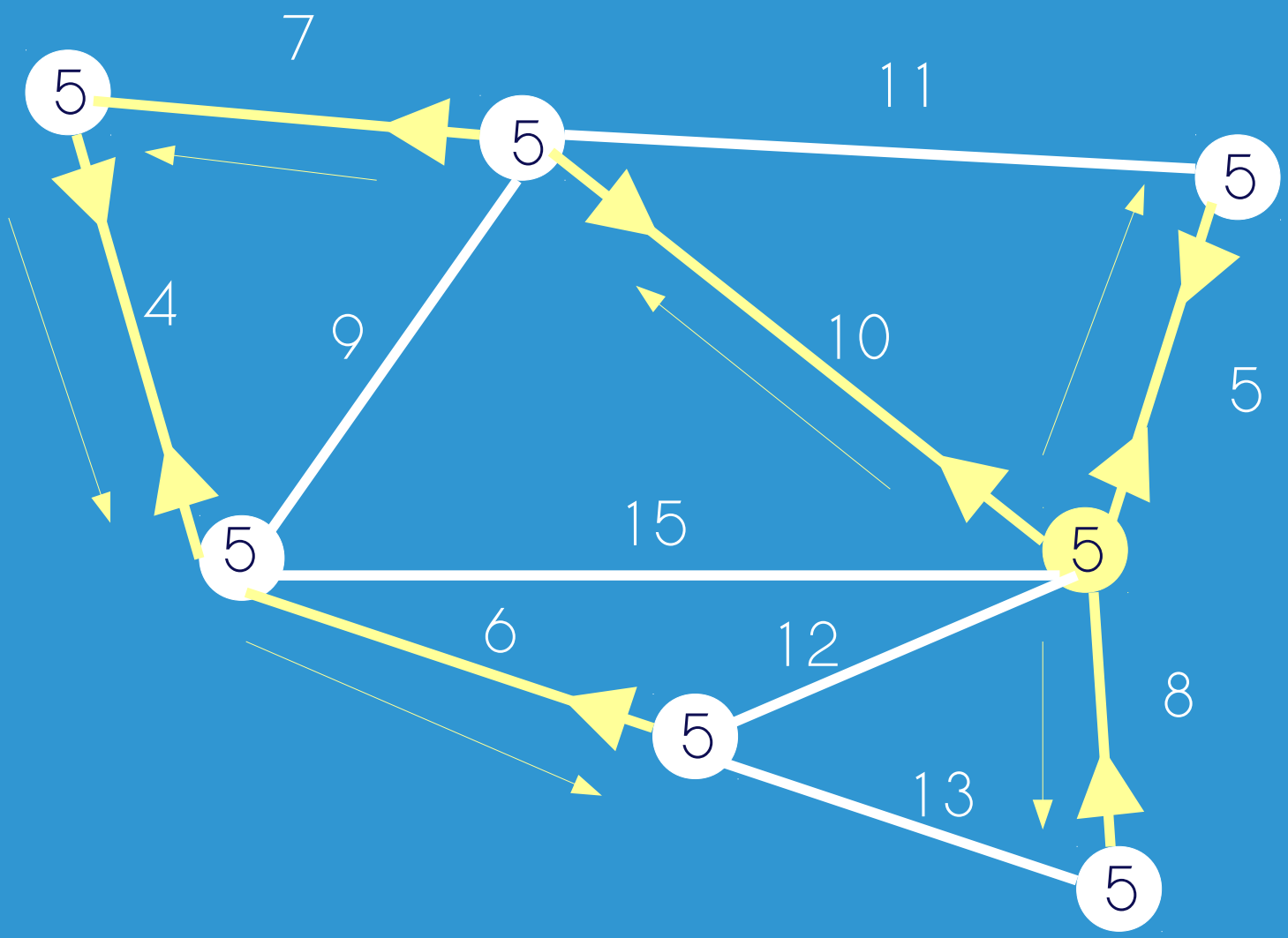
Phase 1 :Send connect



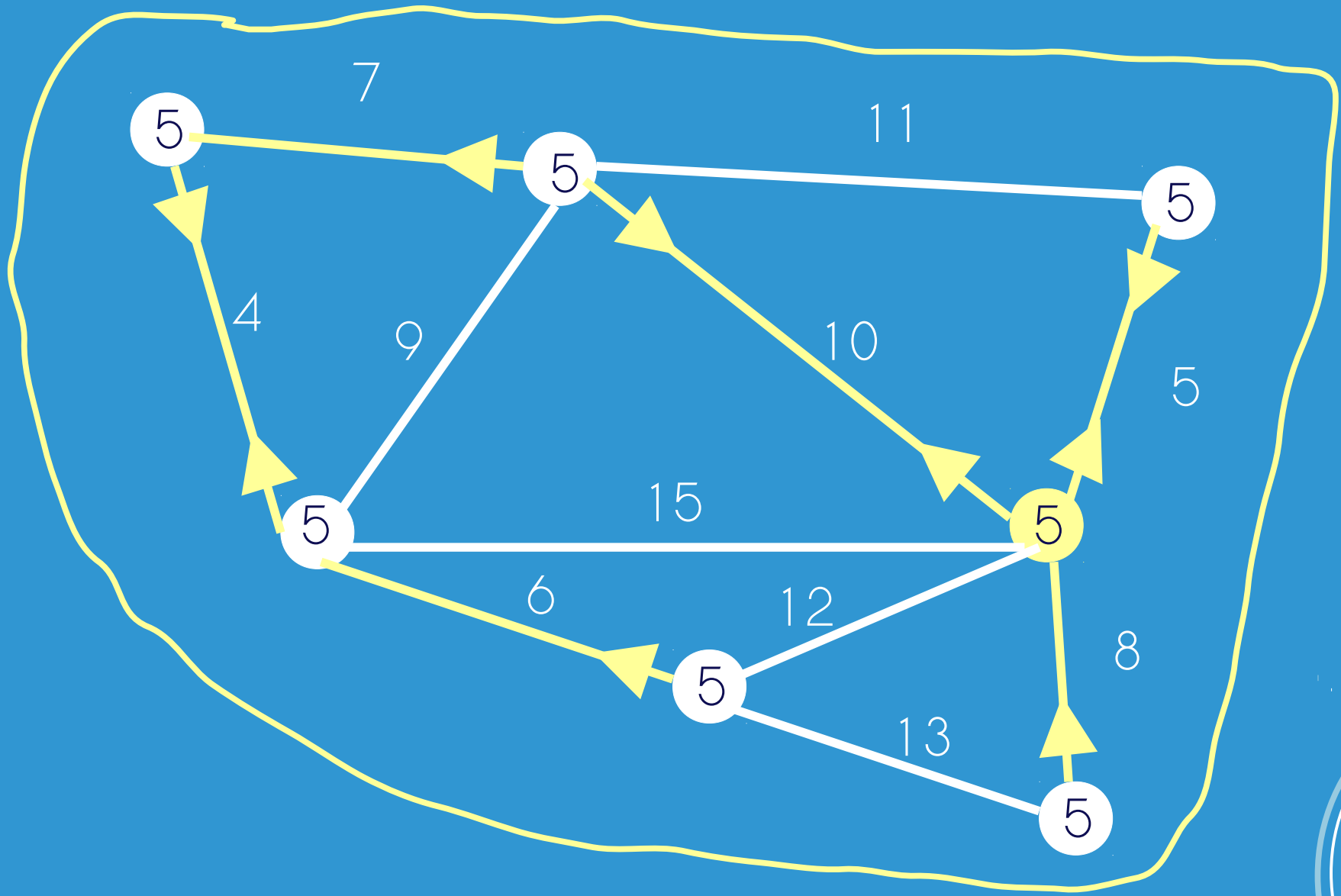
Phase 1 :New root



Phase 1 :Broadcast ID



Phase 1 :MST !

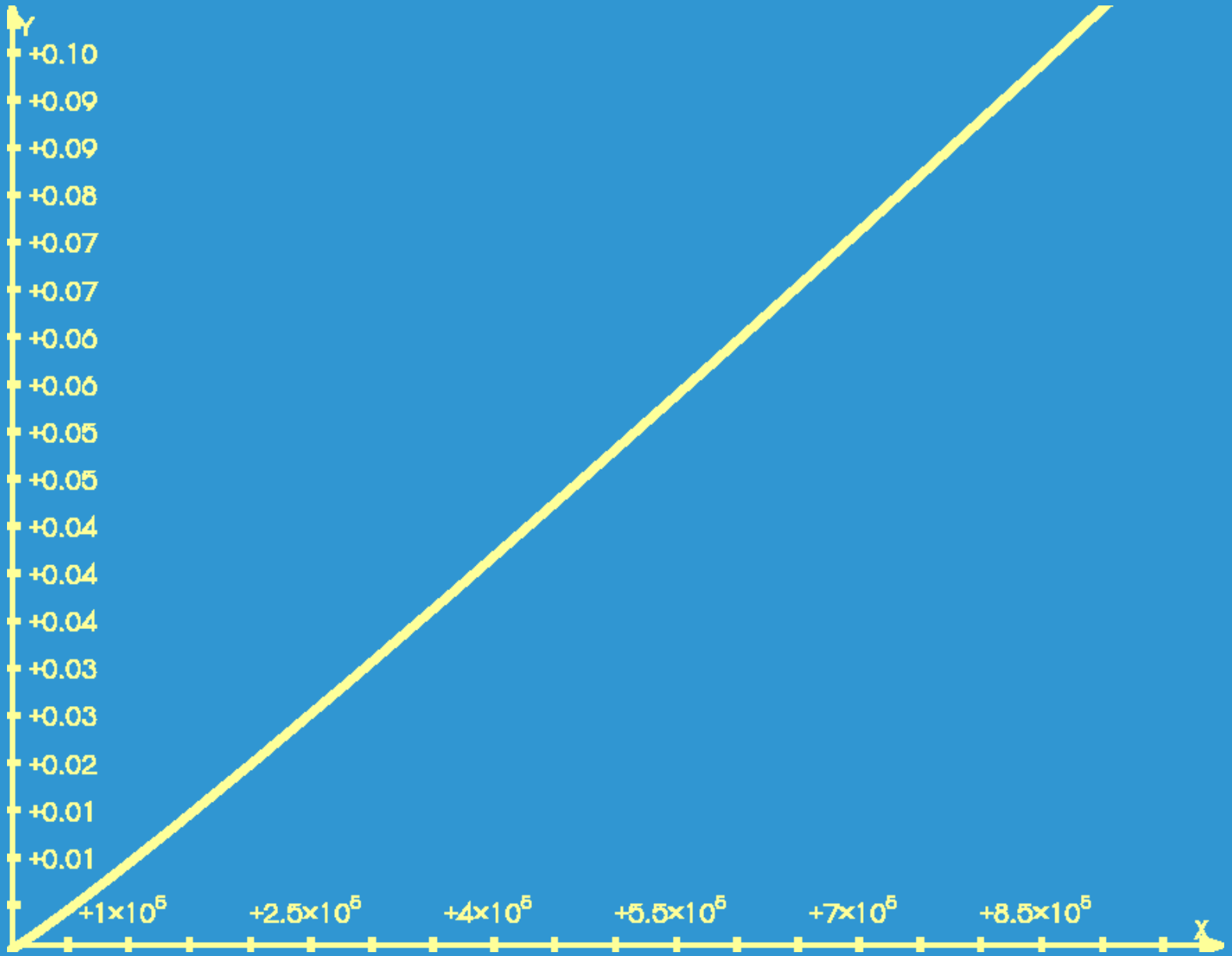


Theoretical analysis of GHS

Theoretical execution time

Number of messages sent per node:	$(2E + 5N(\log(N) - 1) + 3N)/N$
Max size of messages sent:	$\log(E)+\log(8N)$
Speed of connection:	1 Gb/s

Plot


Y
+0.10
+0.09
+0.09
+0.08
+0.07
+0.07
+0.06
+0.06
+0.05
+0.05
+0.04
+0.04
+0.04
+0.03
+0.03
+0.02
+0.01
+0.01
+1×10^5
+2.5×10^5
+4×10^5
+5.5×10^5
+7×10^5
+8.5×10^5
X

Analysis

Theoretically the distributed algorithm is ALWAYS way faster

than the parallel one

This is true with our hypothesis of a network without latencies

and one host per node

Experiments

The Uva cluster

18 nodes with 16 cores each

Max graph size = 82656 edges

Ghs implementation : Python

Initially chose a python implementation : Did not run properly on the cluster

Ran N times (in parallel) the whole algorithm

Ghs implementation : C with MPI

Then chose a C implementation using MPI (Message Passing Interface) to communicate between processes

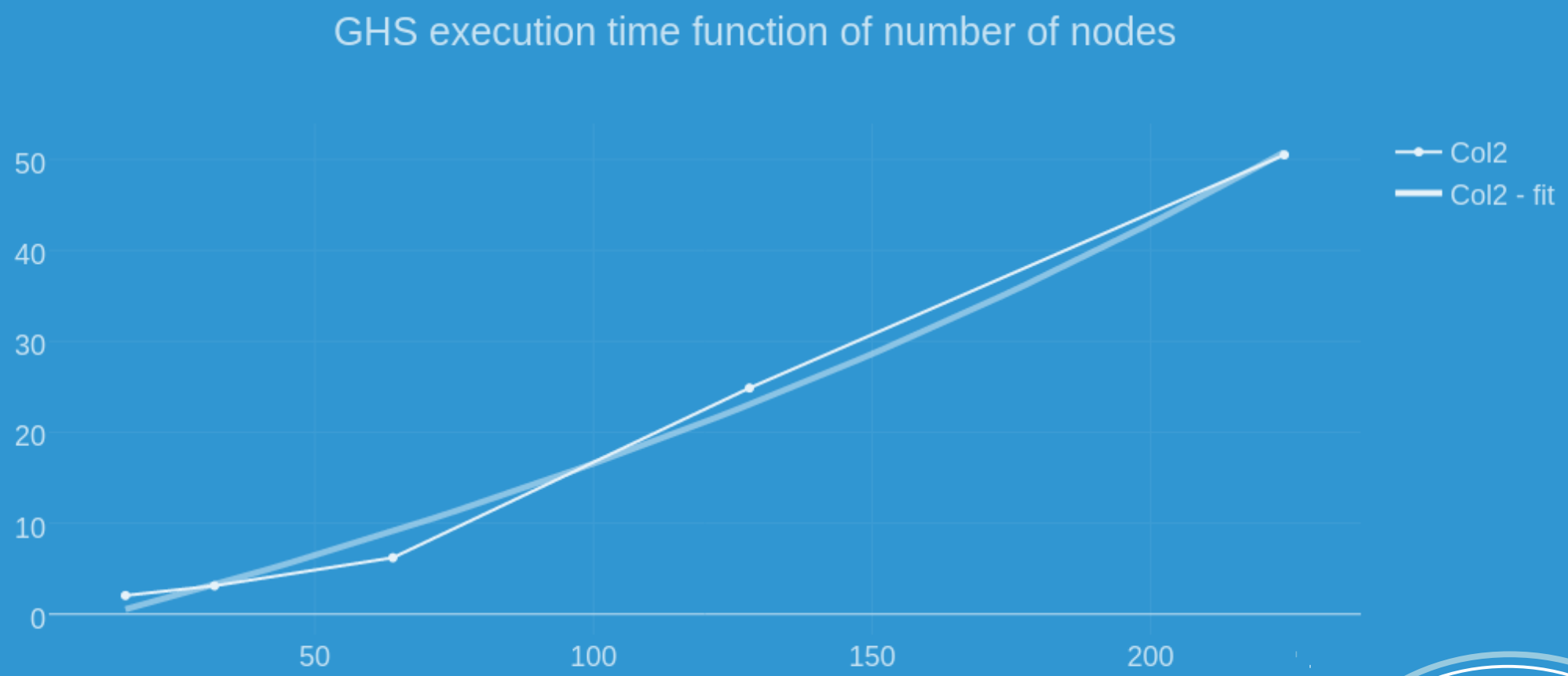
Did not run the algorithm until the end

Making it work

The C algorithm worked for a specific type of graphs

0	1	2	3
1	0	4	5
2	4	0	6
3	5	6	0

Results


GHS execution time function of number of nodes
50
40
30
20
10
0
50
100
150
200
Col2
Col2 - fit

Reasons for such different results

Very badly written algorithm

Message queues

Communication latency

Check if algorithm does not send too many messages

Number of nodes	Theoretical value (msg sent)	Experimental value (msg sent)
224	110410	216712
128	37100	56717
64	10250	8200
32	2710	1573

Check if not a queuing problem

Number of nodes	Number of cores	Time (s)
2	16	3.153
8	4	3.583

Communication latency

Add a latency every time a process sends a message

Theoretical latency needed :
0.1 s

Empirical latency found :
0.025 s

Don't forget that the implementation sends twice the theoretical amount of messages !

Communication latency

Add a latency every time a process sends a message

Theoretical latency needed :
0.1 s

Empirical latency found (between two nodes) :
0.025 s

Don't forget that the implementation sends twice the theoretical amount of messages !

Communication latency

There is no latency if we run the algorithm on one node

Possibly if we run the algorithm on a N core node we match the theoretical speed

Further work

Investigate the other factors that caused the bad performance

Investigate the best architectures to run the distributed algorithm

Conclusion

Parallel algorithm way faster than the distributed one

Causes of bad performances of GHS is communication latency caused by MPI and bad implementation of the algorithm

Uva cluster is not optimized for algorithms that require a lot of communication

Nevertheless it is possible to find implementations and architectures that will make GHS outperform bor-el and this should be investigated