Name: Ekta Arora

Roll No: 161500207 (21)

REPORT ON JOB SEQUENCING AND COLLISION PREVENTION

INTRODUCTION:

Linear pipelining

Pipelining is a technique of that decompose any sequential process into small subprocesses, which are independent of each other so that each subprocess can be executed in a special dedicated segment and all these segments operates concurrently. Thus whole task is partitioned to independent tasks and these subtask are executed by a segment. The result obtained as an output of a segment (after performing all computation in it) is transferred to next segment in pipeline and the final result is obtained after the data have been through all segments. Thus it could understand if take each segment consists of an input register followed by a combinational circuit. This combinational circuit performs the required sub operation and register holds the intermediate result. The output of one combinational circuit is given as input to the next segment.

The concept of pipelining in computer organization is analogous to an industrial assembly line. As in industry there different division like manufacturing, packing and delivery division, a product is manufactured by manufacturing division, while it is packed by packing division a new product is manufactured by manufacturing unit. While this product is delivered by delivery unit a third product is manufactured by manufacturing unit and second product has been packed. Thus pipeline results in speeding the overall process. Pipelining can be effectively implemented for systems having following characteristics:

- A system is repeatedly executes a *basic function*.
- A basic function must be divisible into independent *stages* such that each stage have minimal overlap.
- The complexity of the stages should be roughly similar.

The pipelining in computer organization is basically flow of information. To understand how it works for the computer system lets consider an process which involves four steps /

segment and the process is to be repeated six times. If single steps take t nsec time then time required to complete one process is 4 t nsec and to repeat it 6 times we require 24t nsec.

Now let's see how problem works behaves with pipelining concept. This can be illustrated with a space time diagram given below figure 3.1, which shows the segment utilization as function of time. Lets us take there are 6 processes to be handled (represented in figure as P1, P2, P3, P4, P5 and P6) and each process is divided into 4 segments (S1, S2, S3, S4). For sake of simplicity we take each segment takes equal time to complete the assigned job i.e., equal to one clock cycle. The horizontal axis displays the time in clock cycles and vertical axis gives the segment number. Initially, process1 is handled by the segment 1. After the first clock segment 2 handles process 1 and segment 1 handles new process P2. Thus first process will take 4 clock cycles and remaining processes will be completed one process each clock cycle. Thus for above example total time required to complete whole job will be 9 clock cycles (with pipeline organization) instead of 24 clock cycles required for non pipeline configuration.

	1	2	3	4	5	6	7	8	9
P1	S1	S2	S3	S4					
P2		S1	S2	S3	S4				
P3			S1	S2	S3	S4			
P4				S1	S2	S3	S4		
P5					S1	S2	S3	S4	
P6						S1	S2	S3	S4

Figure 3.1 Space –time diagram for pipeline

Speedup ratio : The speed up ratio is ratio between maximum time taken by non pipeline process over process using pipelining. Thus in general if there are n processes and each process is divided into k segments (subprocesses). The first process will take k segments to complete the processes, but once the pipeline is full that is first process is complete, it will take only one clock period to obtain an output for each process. Thus first process will take k clock cycles and remaining n-1 processes will emerge from the pipe at the one process per clock cycle thus total time taken by remaining process will be (n-1) clock cycle time.

Let tbe the one clock cycle time.
p

The time taken for n processes having k segments in pipeline configuration will be

$= k*t_p + (n-1)*t_p = (k+n-1)*t_p$

the time taken for one process is t_n thus the time taken to complete n process in non pipeline = $n*t_n$ configuration will be

Thus speed up ratio for one process in non pipeline and pipeline configuration is = $n*t_n / (n+k-1)*t_p$

if n is very large compared to k than

$=t_n / t_p$

if a process takes same time in both case with pipeline and non pipeline configuration than $t_n = k*t_p$

Thus speed up ratio will $S_k = k*t_p/t_p = k$

Theoretically maximum speedup ratio will be k where k are the total number of segments in which process is divided. The following are various limitations due to which any pipeline system cannot operate at its maximum theoretical rate i.e., k (speed up ratio).

a. Different segments take different time to complete there suboperations, and in pipelining clock cycle must be chosen equal to time delay of the segment with maximum propagation time. Thus all other segments have to waste time waiting for next clock cycle. The possible solution for improvement here can if possible subdivide the segment into different stages i.e., increase the number of stages and if segment is not subdivisible than use multiple of resource for segment causing maximum delay so that more than one instruction can be executed in to different resources and overall performance will improve.

b. Additional time delay may be introduced because of extra circuitry or additional software requirement is needed to overcome various hazards, and store the result in the intermediate registers. Such delays are not found in non pipeline circuit.

c. Further pipelining can be of maximum benefit if whole process can be divided into suboperations which are independent to each other. But if there is some resource conflict or data dependency i.e., a instruction depends on the result of pervious instruction which is not yet available than instruction has to wait till result become available or conditional or non conditional branching i.e., the

bubbles or time delay is introduced.

Efficiency : The efficiency of linear pipeline is measured by the percentage of time when processor are busy over total time taken i.e., sum of busy time plus idle time. Thus if n is number of task , k is stage of pipeline and t is clock period then efficiency is given by $\eta = n/ [k + n -1]$

Thus larger number of task in pipeline more will be pipeline busy hence better will be efficiency.

It can be easily seen from expression as $n \rightarrow \infty$, $\eta \rightarrow 1$.

$\eta = S/k$
k

Thus efficiency η of the pipeline is the speedup divided by the number of stages, or one can say actual speed ratio over ideal speed up ratio. In steady stage where $n >> k$, η approaches 1.

Throughput: The number of task completed by a pipeline per unit time is called throughput, this represents computing power of pipeline. We define throughput as

$W= n/[k*t + (n-1) *t] = \eta/t$

In ideal case as η -> 1 the throughout is equal to 1/t that is equal to frequency. Thus maximum throughput is obtained is there is one output per clock pulse.

Que 3.1. A non-pipeline system takes 60 ns to process a task. The same task can be processed in six segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speed up that can be achieved? Soln. Total time taken by for non pipeline to complete 100 task is = 100 * 60 = 6000 ns Total time taken by pipeline configuration to complete 100 task is

= (100 + 6 –1) *10 = 1050 ns

Thus speed up ratio will be = 6000 / 1050 = 4.76

The maximum speedup that can be achieved for this process is = 60 / 10 = 6

Thus, if total speed of non pipeline process is same as that of total time taken to complete a process with pipeline than maximum speed up ratio is equal to number of segments.

Que 3.2. A non-pipeline system takes 50 ns to process a task. The same task can be processed in a six segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speed up that can be achieved? Soln.

Total time taken by for non pipeline to complete 100 task is = 100 * 50 = 5000 ns Total time taken by pipeline configuration to complete 100 task is = (100 + 6 –1) *10 = 1050 ns

Thus speed up ratio will be = 5000 / 1050 = 4.76

The maximum speedup that can be achieved for this process is = 50 / 10 = 5

The two areas where pipeline organization is most commonly used are arithmetic pipeline and instruction pipeline. An arithmetic pipeline where different stages of an arithmetic operation are handled along the stages of a pipeline i.e., divides the arithmetic operation into suboperations for execution of pipeline segments. An instruction pipeline operates on a stream of instructions by overlapping the fetch, decode, and execute phases of the instruction cycle as different stages of pipeline. RISC architecture supports pipelining more than a CISC architecture does. There are three prime disadvantages of pipeline architecture.

1. The first is complexity i.e., to divide the process into dependent subtask
2. Many intermediate registers are required to hold the intermediate information as output of one stage which will be input of next stage. These are not required for single unit circuit thus it is usually constructed entirely as combinational circuit
3. The third disadvantage is its inability to continuously run the pipeline at full speed, i.e. the pipeline stalls for some cycle. There are phenomena called pipeline hazards which disrupt the smooth execution of the pipeline if these hazards are not handled properly they may gave wrong result. Often it is required insert delays in the pipeline flow in order to manage these hazards such delays are called bubbles. Often it is managed by using special hardware techniques while sometime using software techniques such as compiler or code reordering, etc. Various types of pipeline hazards include:
    - structural hazards that happens due to hardware conflicts
    - data hazards that happen due to data dependencies
    - control hazards that happens when there is change in flow of statement like due to branch, jump, or any other control flow changes conditions
    - Exception hazard that happens due to some exception or interrupt occurred while execution in a pipeline system.

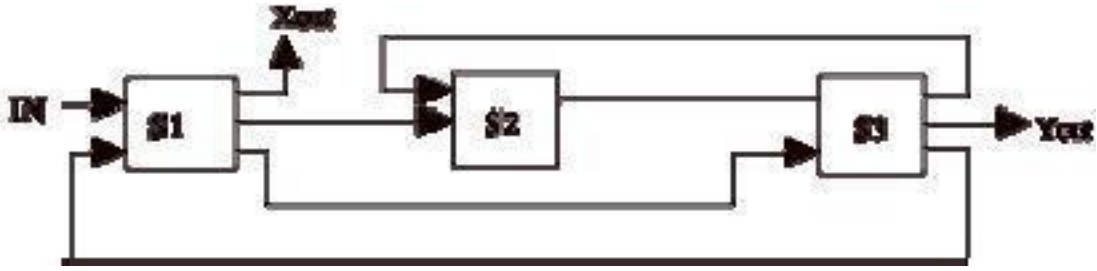
3.1 Non linear pipeline

A dynamic pipeline can be reconfigured to perform variable function at different times. The traditional linear pipelines are static pipeline because they used to perform

fixed function. A dynamic pipeline allows feed forward and feedback connections in addition to streamline connection. A dynamic pipelining may initiate tasks from different reservation tables simultaneously to allow multiple numbers of initiations of different functions in the same pipeline.

3.3.1 Reservation Tables and latency analysis

Reservation tables are used how successive pipeline stages are utilized for a specific evaluation function. These reservation tables show the sequence in which each function utilizes each stage. The rows correspond to pipeline stages and the columns to clock time units. The total number of clock units in the table is called the evaluation time. A reservation table represents the flow of data through the pipeline for one complete evaluation of a given function. (For example, think of X as being a floating square root, and Y as being a floating cosine. A simple floating multiply might occupy just S1 and S2 in sequence.) We could also denote multiple stages being used in parallel, or a stage being drawn out for more than one cycle with these diagrams.



S1	X					X		X
S2		X		X				
S3			X		X		X	

S1	Y				Y	
S2			Y			
S3		Y		Y		Y

We determine the next start time for one or the other of the functions by lining up the diagrams and sliding one with respect to another to see where one can fit into the open slots. Once an X function has been scheduled, another X function can start after 1, 3 or 6 cycles. A Y function can start after 2 or 4 cycles. Once a Y function has been scheduled, another Y function can start after 1, 3 or 5 cycles. An X function can start after 2 or 4 cycles. After two functions have been scheduled, no more can be started until both are complete.

Job Sequencing and Collision Prevention

Initiation the start a single function evaluation collision may occur as two or more initiations attempt to use the same stage at the same time. Thus it is required to properly schedule queued tasks awaiting initiation in order to avoid collisions and to achieve high throughput. We can define collision as:

1. A collision occurs when two tasks are initiated with latency (initiation interval) equal to the column distance between two "X" on some row of the reservation table.
2. The set of column distances F ={l1,l2,…,lr} between all possible pairs of "X" on each row of the reservation table is called the forbidden set of latencies.
3. The collision vector is a binary vector C = (Cn…C2 C1), Where Ci=1 if i belongs to F (set of forbidden latencies) and Ci=0 otherwise.

Some fundamental concepts used in it are**:**

Latency - number of time units between two initiations (any positive integer 1, 2,…)

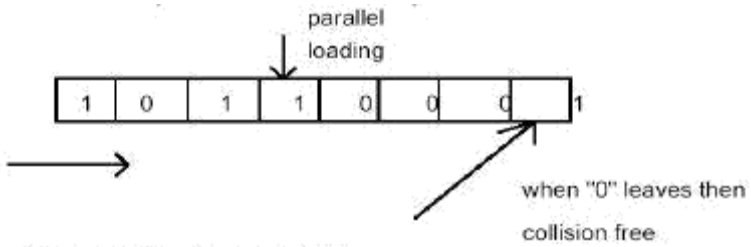
Latency sequence – sequence of latencies between successive initiations

Latency cycle – a latency sequence that repeats itself

Control strategy – the procedure to choose a latency sequence

Greedy strategy – a control strategy that always minimizes the latency between the current initiation and the very last initiation

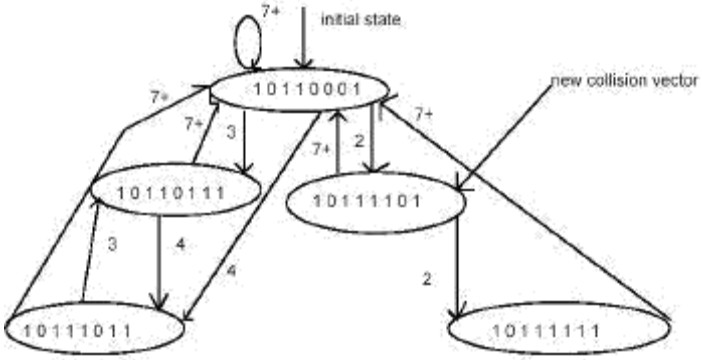
Example: Let us consider a Reservation Table with the following set of forbidden latencies F and permitted latencies P (complementation of F).



Forbidden list = F = {1,5,6,8}
Collision vector: C={1 0 1 1 0 0 0 1)
8 7 6 5 4 3 2 1

	0	1	2	3	4	5	6	7	8
1	X								X
2		X	X					X	
3				X					
4					X	X			
5							X	X	

1 0 1 1 0 0 0 1
1 0 1 1 0 1 1 1
OR 1 0 1 1
1 0 1 1 1 0 1 1



1 0 1 1 0 0 0 1 initial
(−)0 0 1 0 1 1 0 0 →2 (initial)
1 0 1 1 1 1 0 1

1 0 1 1 0 0 0 1 initial
0 0 0 1 0 1 1 0 →2 (initial)
1 0 1 1 0 1 1 1

1 0 1 1 0 0 0 1 initial
0 0 0 0 1 0 1 1 →4 (initial)

1 0 1 1 0 0 0 1 initial
0 0 0 0 0 0 0 1 →7 (initial)
1 0 1 1 0 0 0 1

It has been observed that

1. The collision vector shows both permitted and forbidden latencies from the same reservation table.

2. One can use n-bit shift register to hold the collision vector for implementing a control strategy for successive task initiations in the pipeline. Upon initiation of the first task, the collision vector is parallel-loaded into the shift register as the initial state. The shift register is then shifted right one bit at a time, entering 0's from the left end. A collision free initiation is allowed at time instant t+k a bit 0 is being shifted at of the register after k shifts from time t.

A **state diagram** is used to characterize the successive initiations of tasks in the pipeline in order to find the shortest latency sequence to optimize the control strategy. A **state** on the diagram is represented by the contents of the shift register after the proper number of shifts is made, which is equal to the latency between the current and next task initiations.

3. The successive collision vectors are used to prevent future task collisions with previously initiated tasks, while the collision vector C is used to prevent possible collisions with the current task. If a collision vector has a "1" in the ith bit (from the right), at time t, then the task sequence should avoid the initiation of a task at time t+i.

4. Closed logs or cycles in the state diagram indicate the steady – state sustainable latency sequence of task initiations without collisions. The **average latency** of a cycle is the sum of its latencies (period) divided by the number of states in the cycle.

5. The throughput of a pipeline is inversely proportional to the reciprocal of the average latency. A latency sequence is called **permissible** if no collisions exist in the successive initiations governed by the given latency sequence.

6. The maximum throughput is achieved by an optimal scheduling strategy that achieves the (MAL) minimum average latency without collisions.

Simple cycles are those latency cycles in which each state appears only once per each iteration of the cycle. A single cycle is a **greedy cycle** if each latency contained in the cycle is the minimal latency (outgoing arc) from a state in the cycle. A good task- initiation sequence should include the greedy cycle.

Procedure to determine the greedy cycles

1. From each of the state diagram, one chooses the arc with the smallest latency label unit; a closed simple cycle can formed.
2. The average latency of any greedy cycle is no greater than the number of latencies in the forbidden set, which equals the number of 1's in the initial collision vector.
3. The average latency of any greedy cycle is always lower-bounded by the MAL in the collision vector

Two methods for improving dynamic pipeline throughput have been proposed by Davidson and Patel these are

- The reservation of a pipeline can be modified with insertion of non complete delays
- Use of internal buffer at each stage.

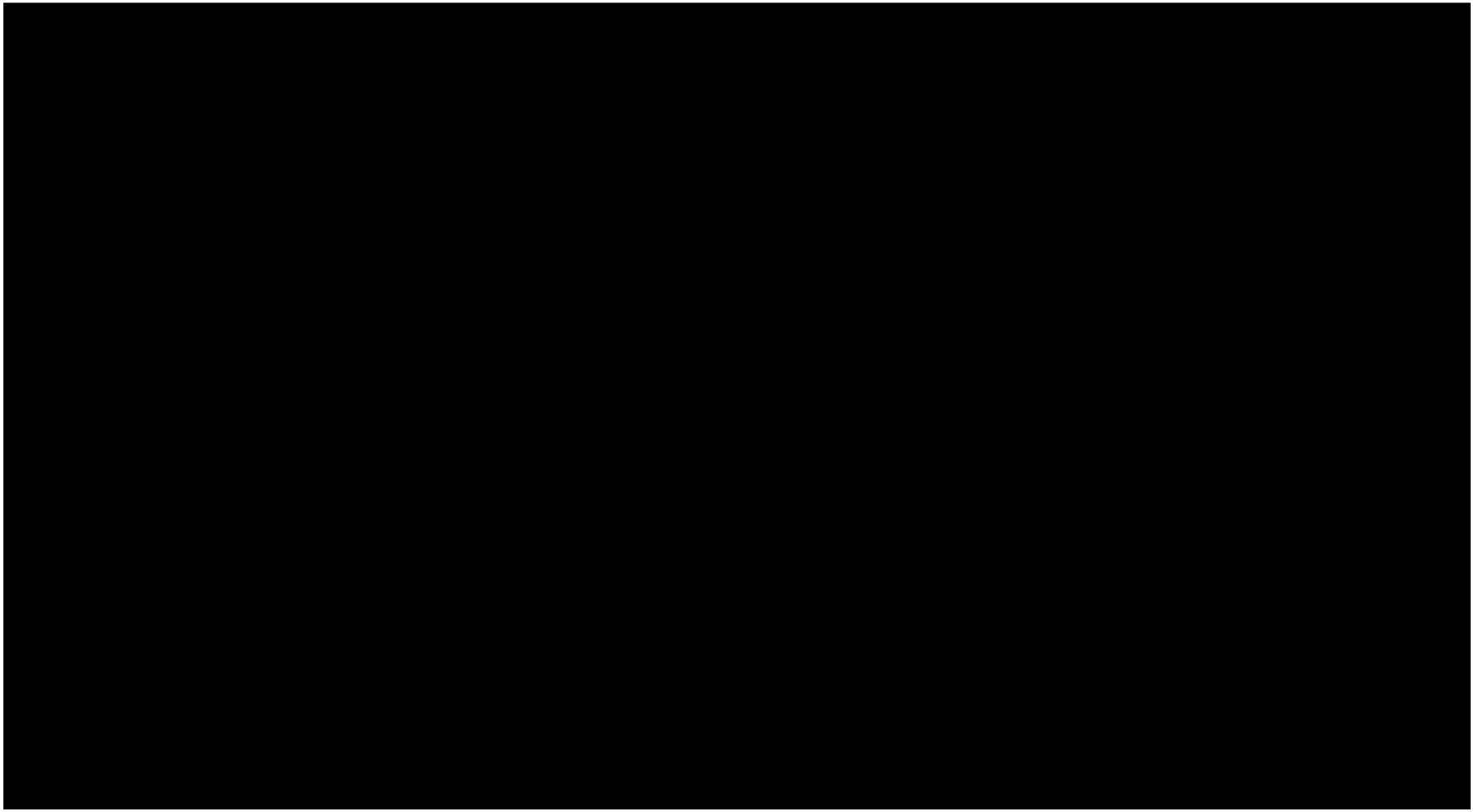
Thus high throughput can be achieved by using the modified reservation table yielding a more desirable latency pattern such the each stage is maximum utilized. Any computation can be delayed by inserting a non compute stage.

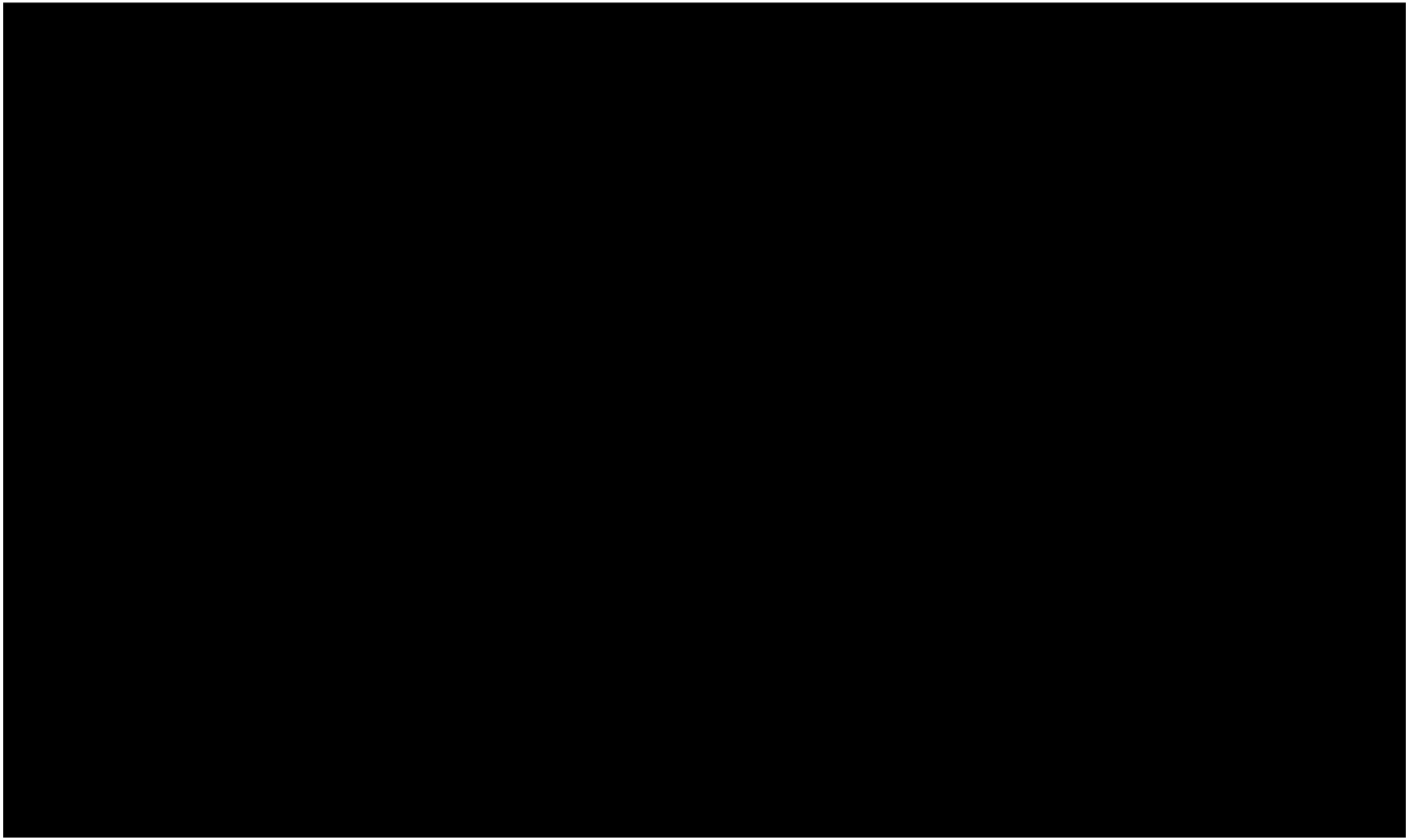
Reconfigurable pipelines with different function types are more desirable. This requires an extensive resource sharing among different functions. To achieve this one need a more complicated structure of pipeline segments and their interconnection controls like bypass techniques to avoid unwanted stage.

A dynamic pipeline would allow several configurations to be simultaneously present like

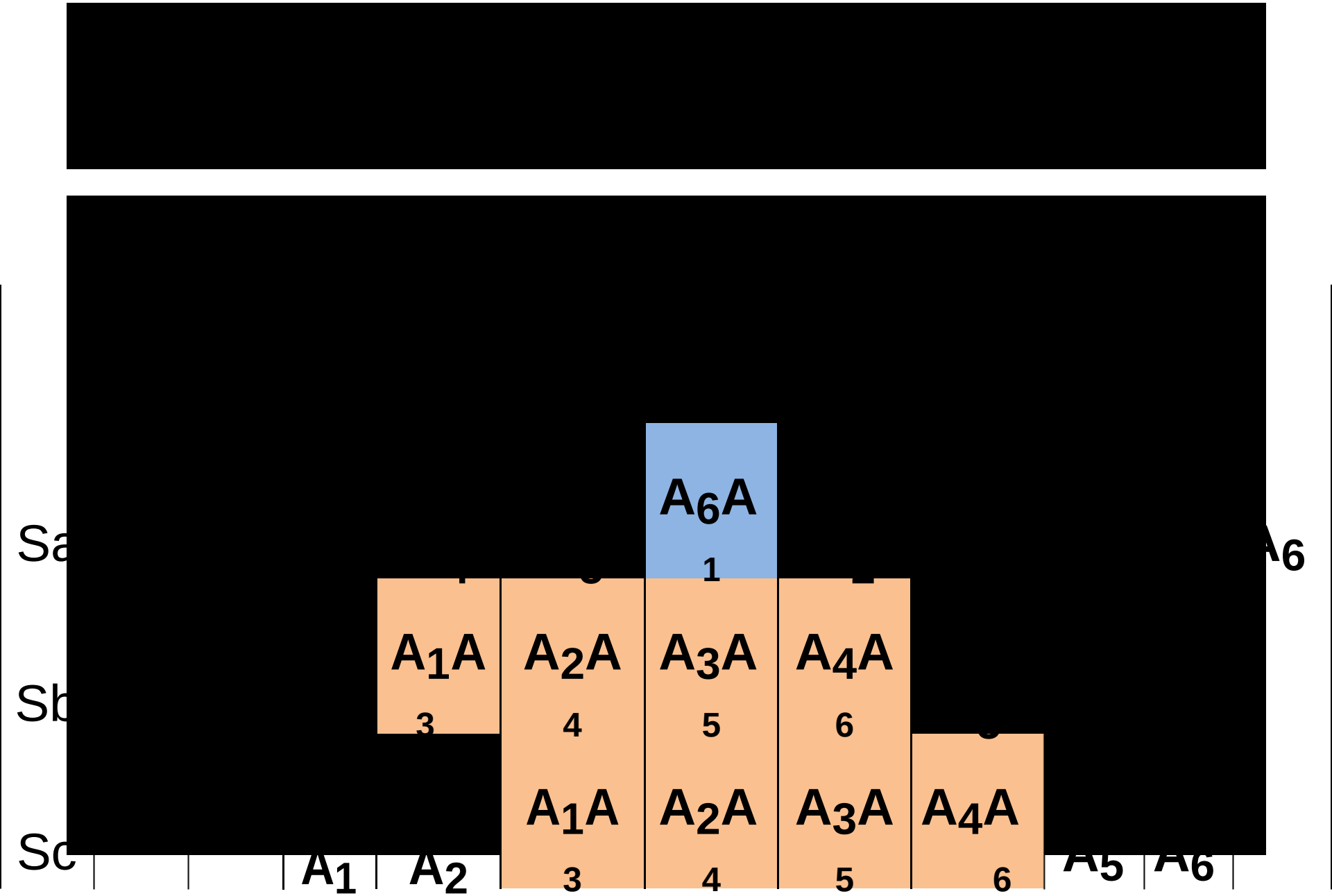
arithmetic unit performing both addition as well as multiplication at same time. But to

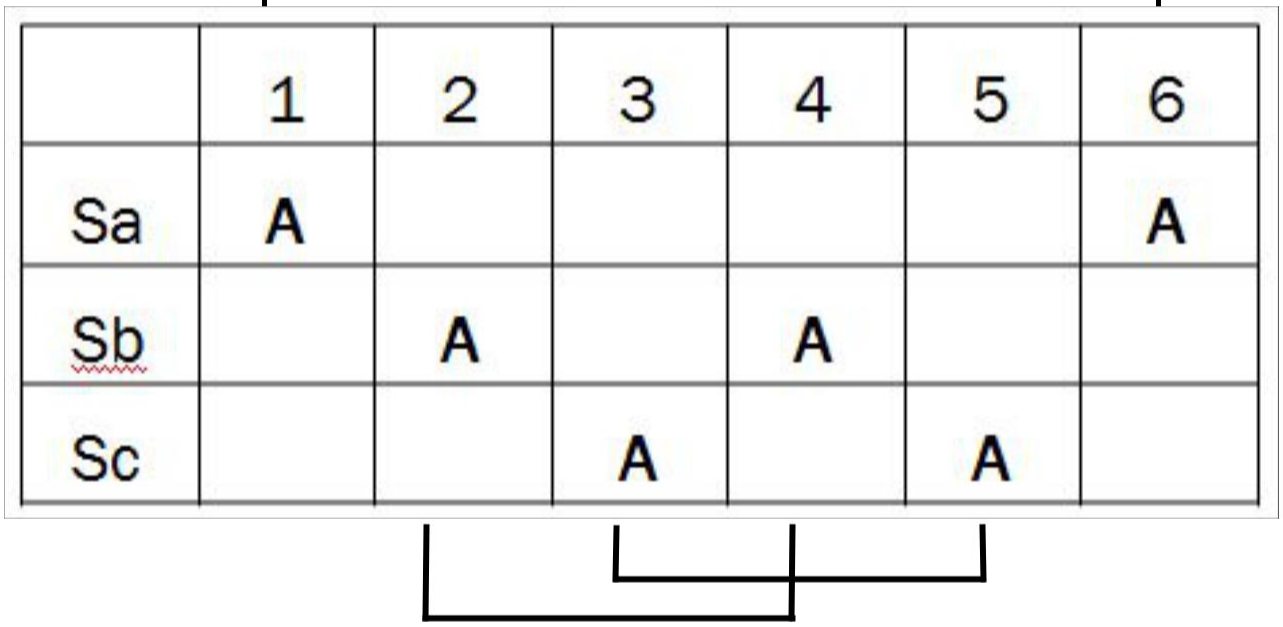
achieve this tremendous control overhead and increased interconnection complexity would be expected.





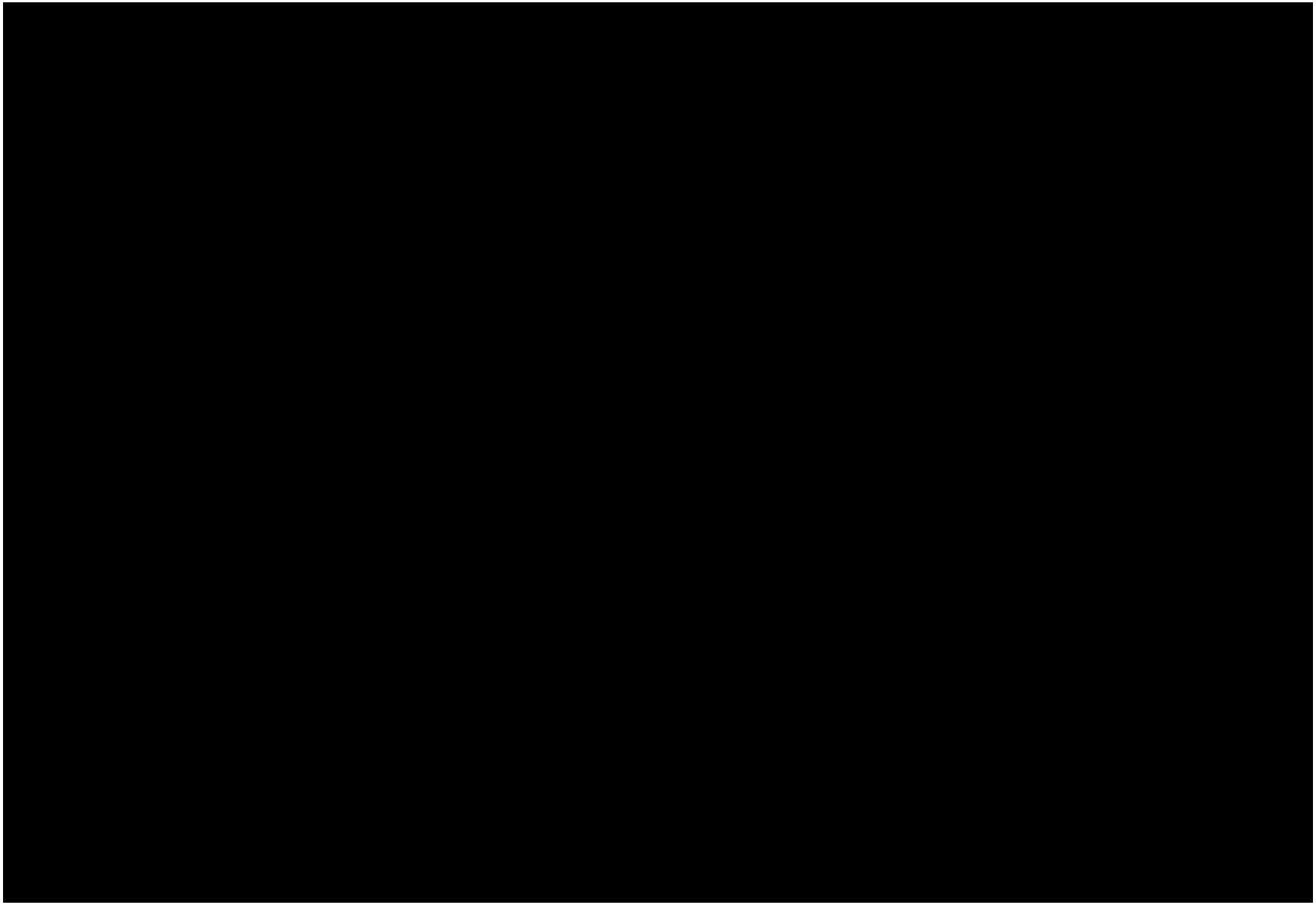
Latency:

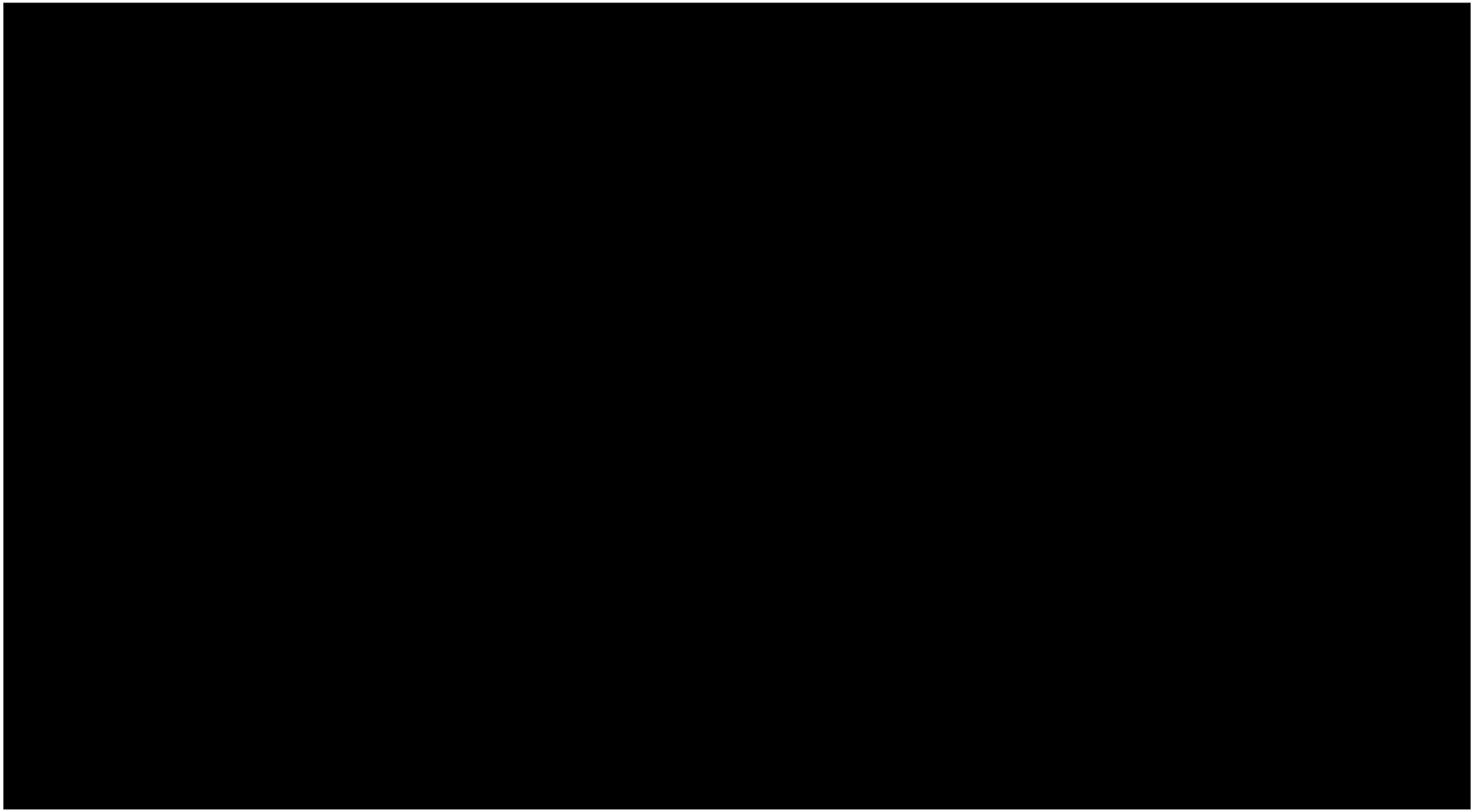

A6A
1
A1A
3
A2A
4
A3A
5
A4A
6
A1A
3
A2A
4
A3A
5
A4A
6
A1
A2
A5
A6

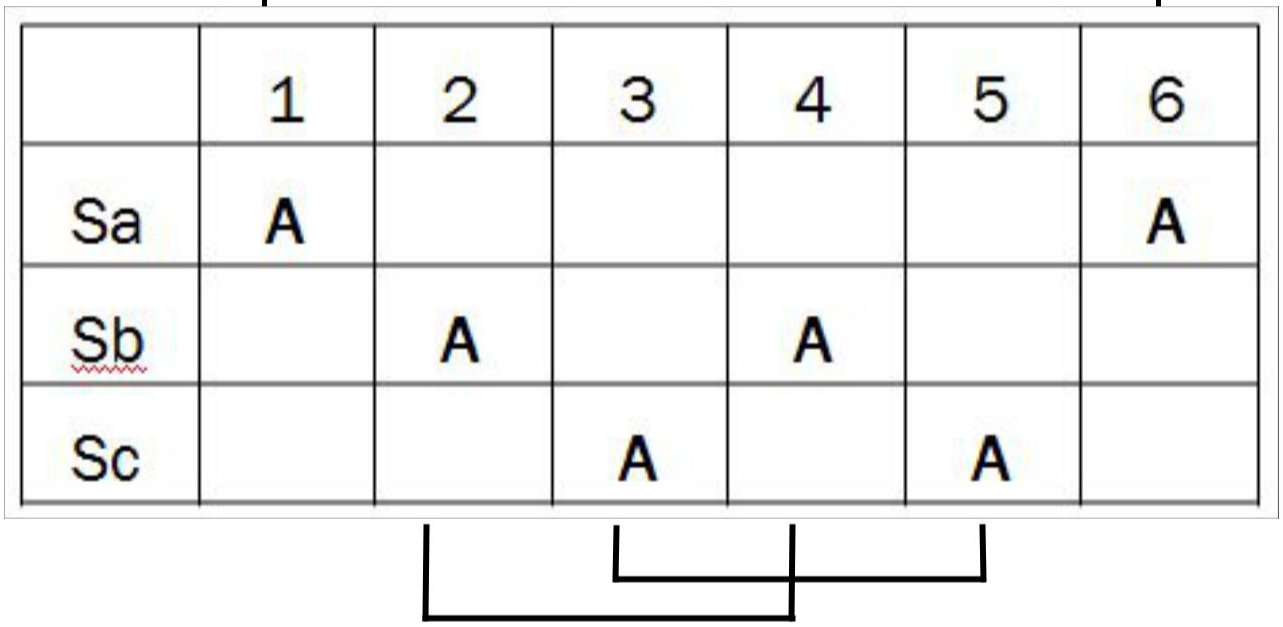


	1	2	3	4	5	6
Sa	A					A
Sb		A		A		
Sc			A		A	

}



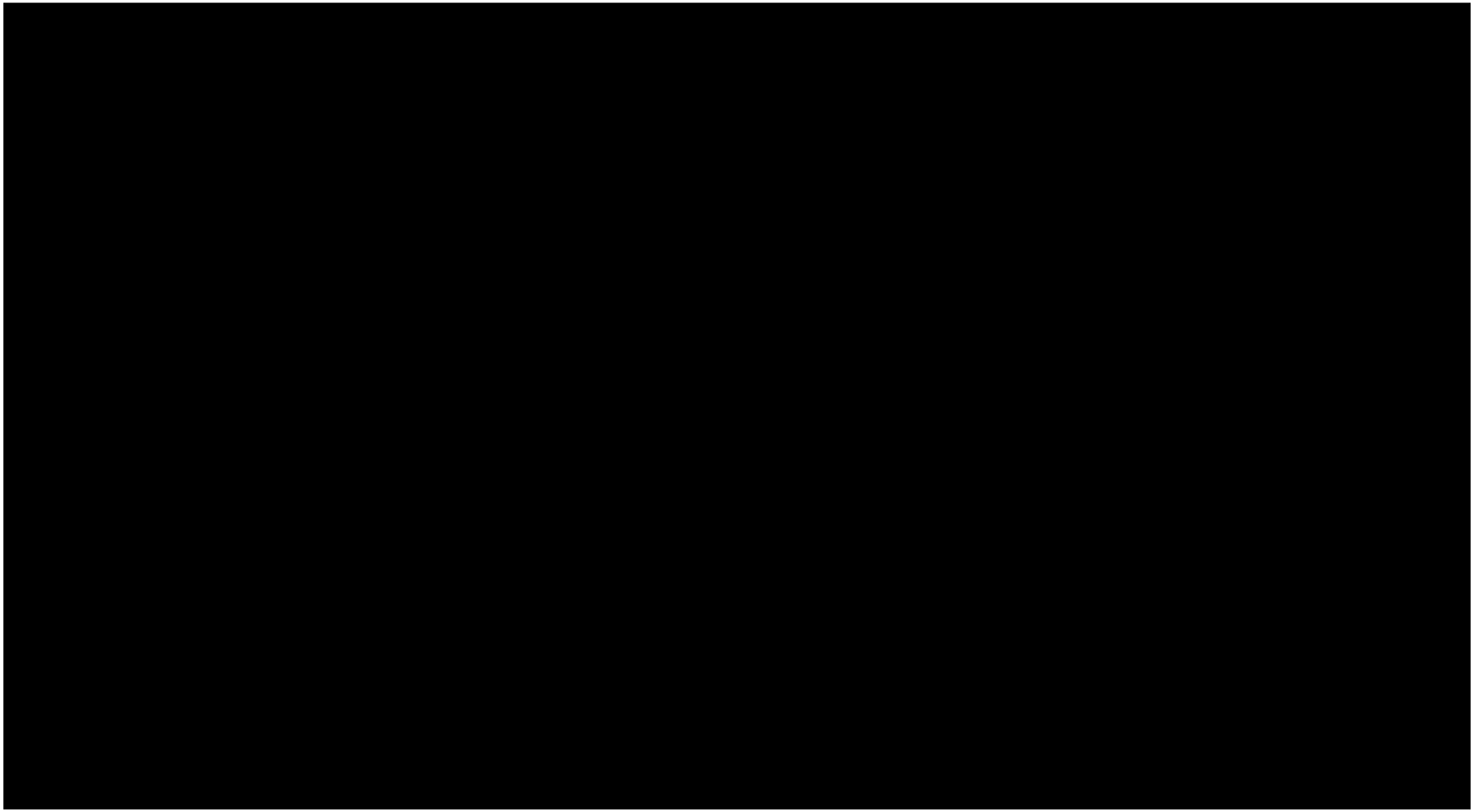


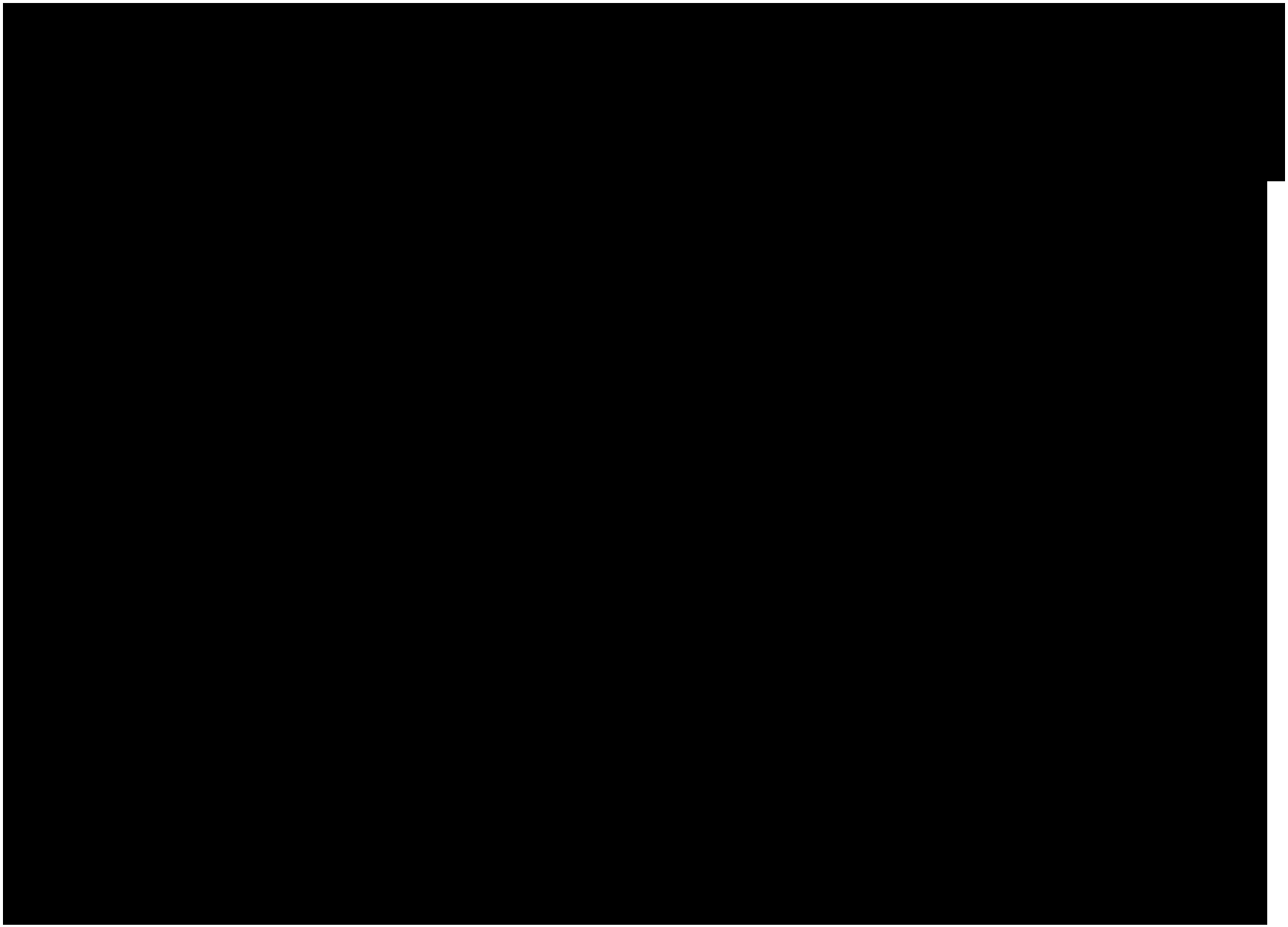


	1	2	3	4	5	6
Sa	A					A
Sb		A		A		
Sc			A		A	

C = (C . C

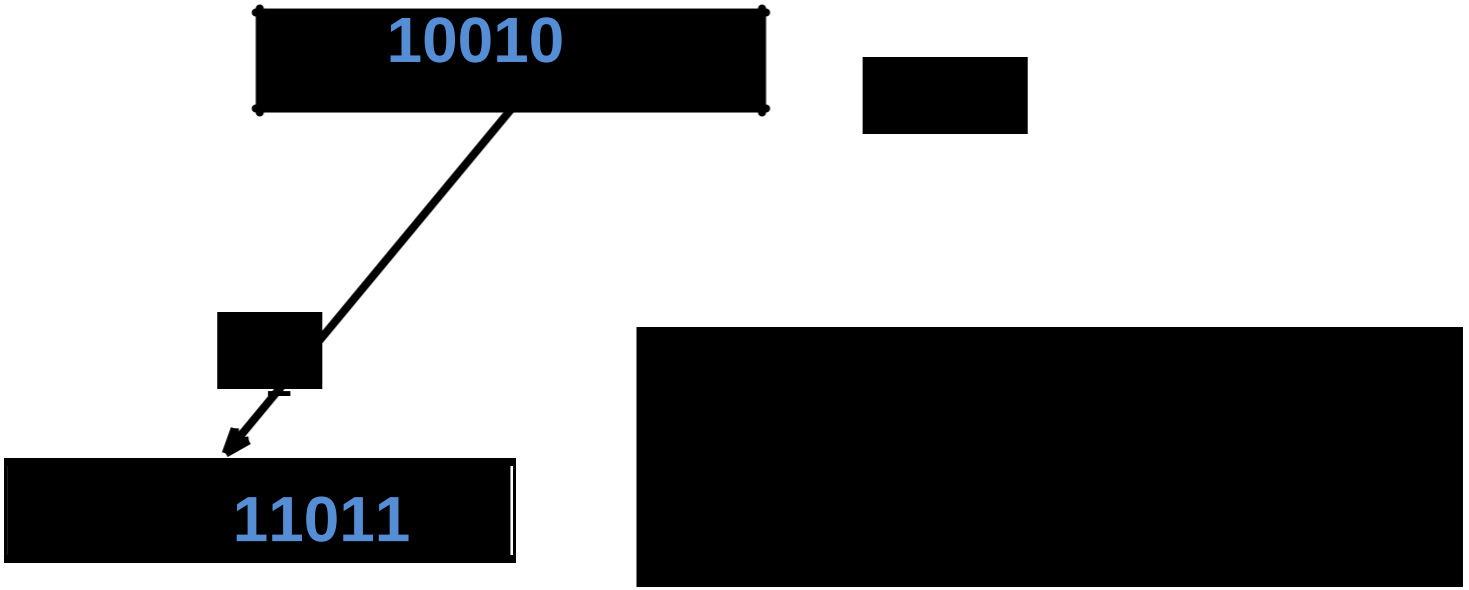
F and C





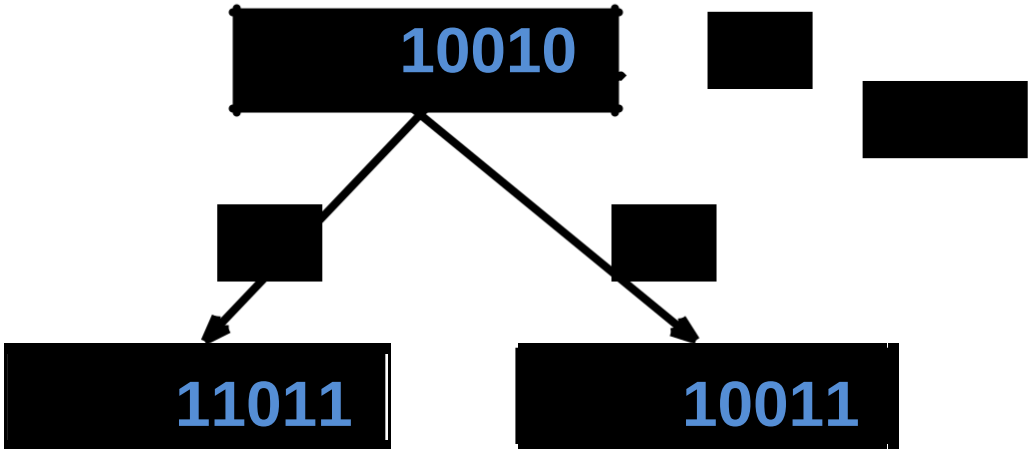
10010

[redacted]

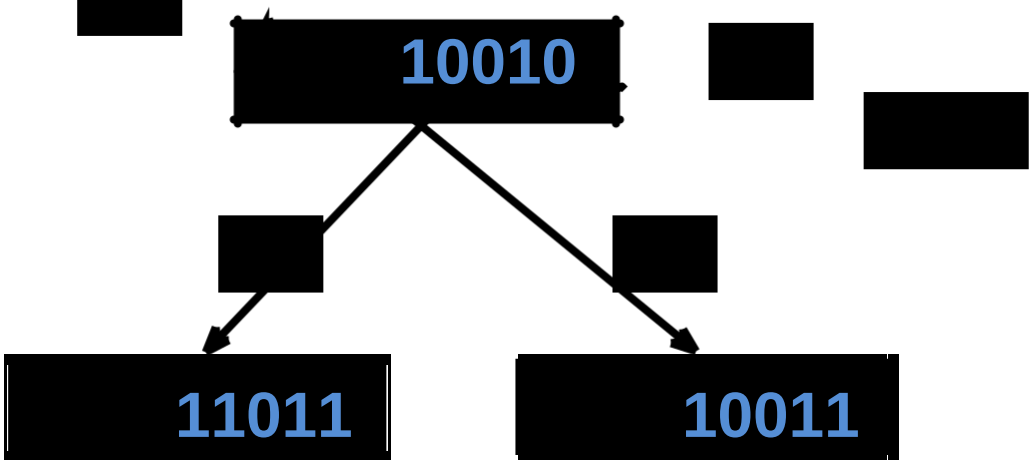



10010
11011

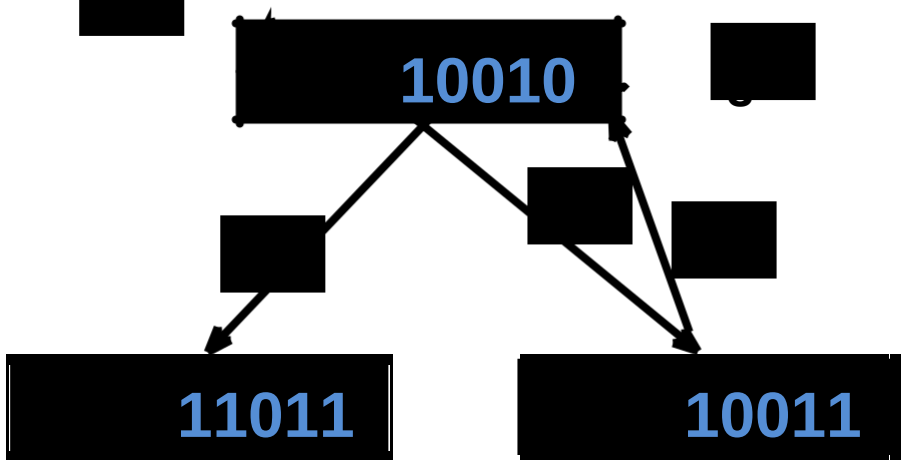
[redacted]



[redacted]

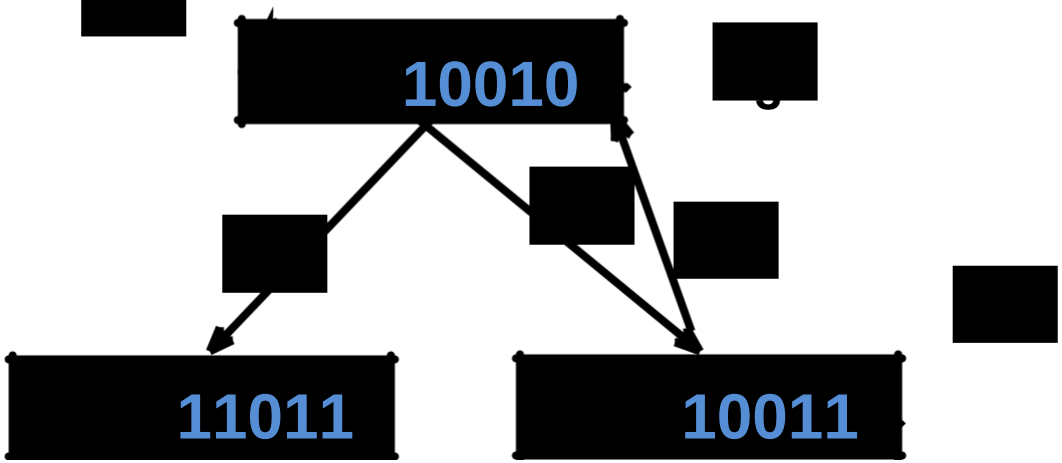

10010
11011
10011

[redacted]



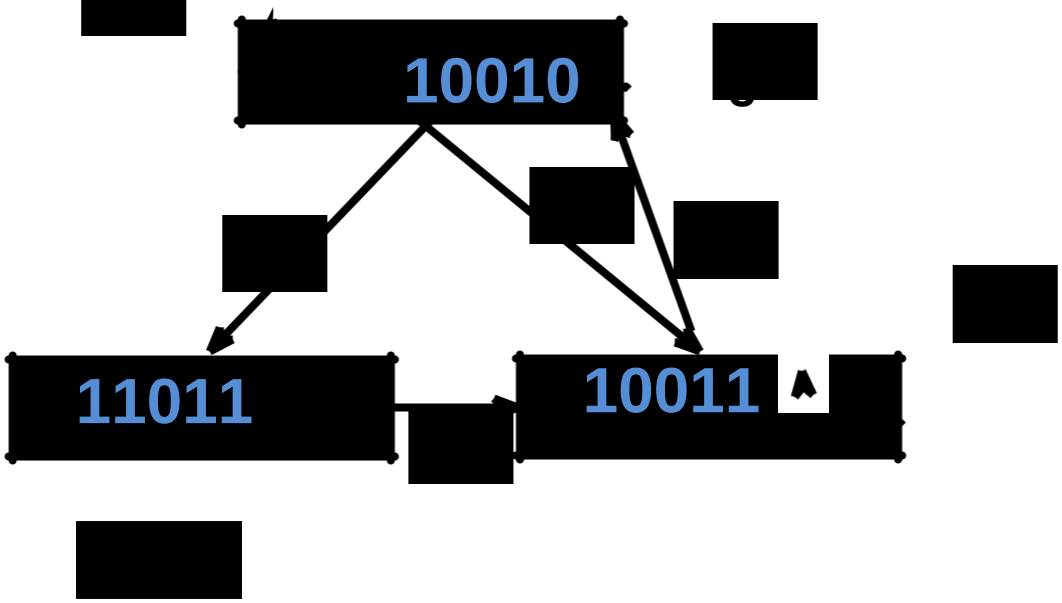
[redacted]

[redacted]



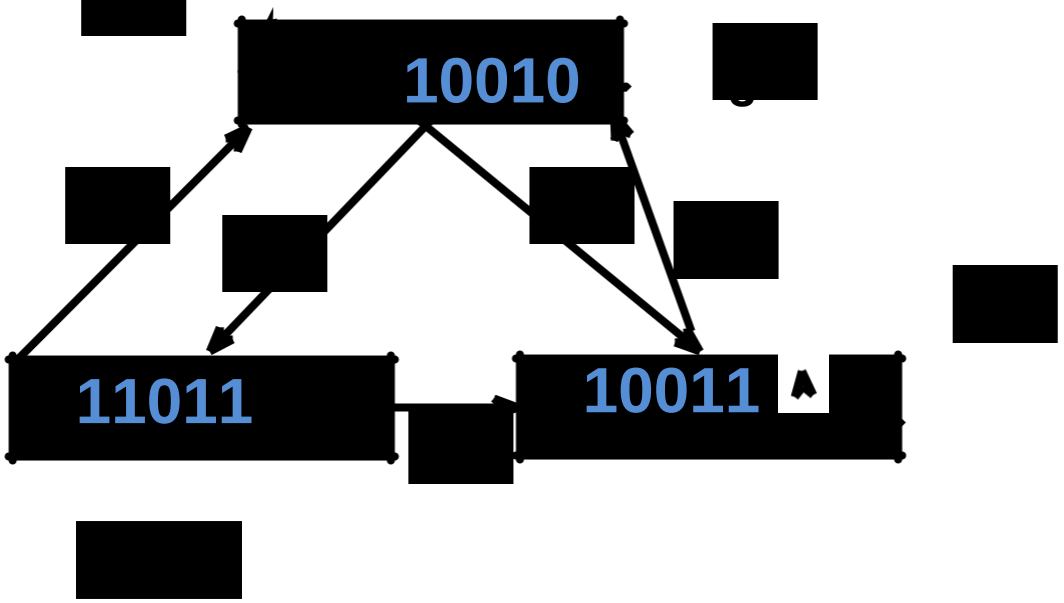
[redacted]

[redacted]



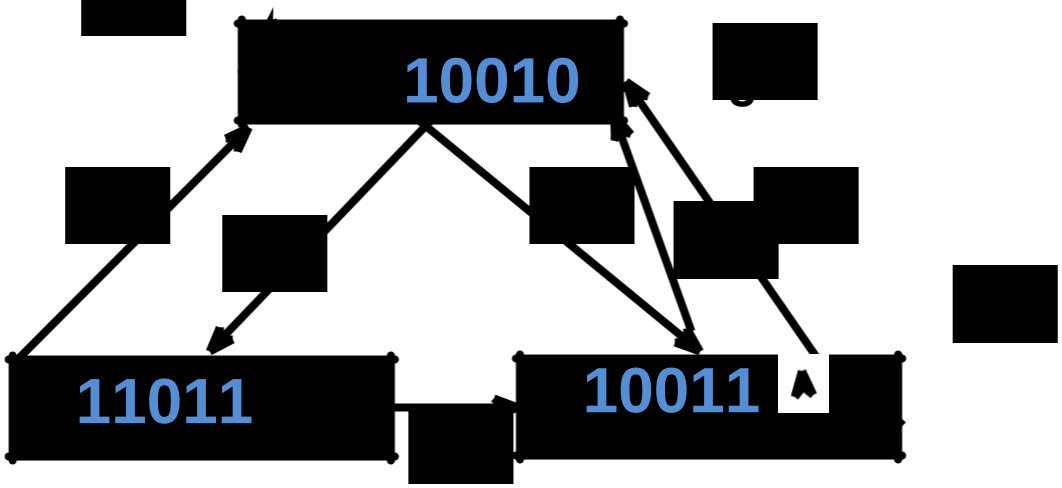
[redacted]

[redacted]

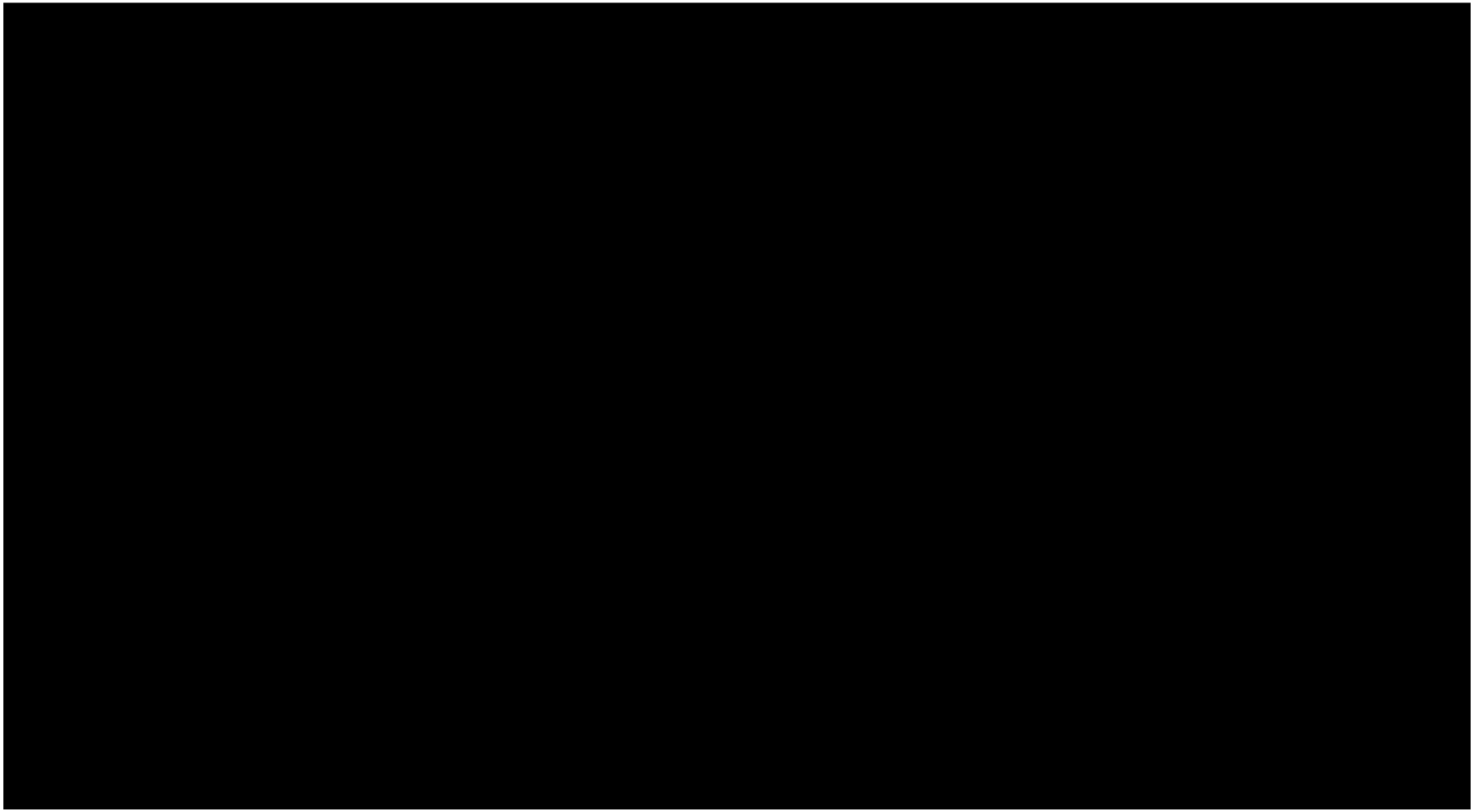


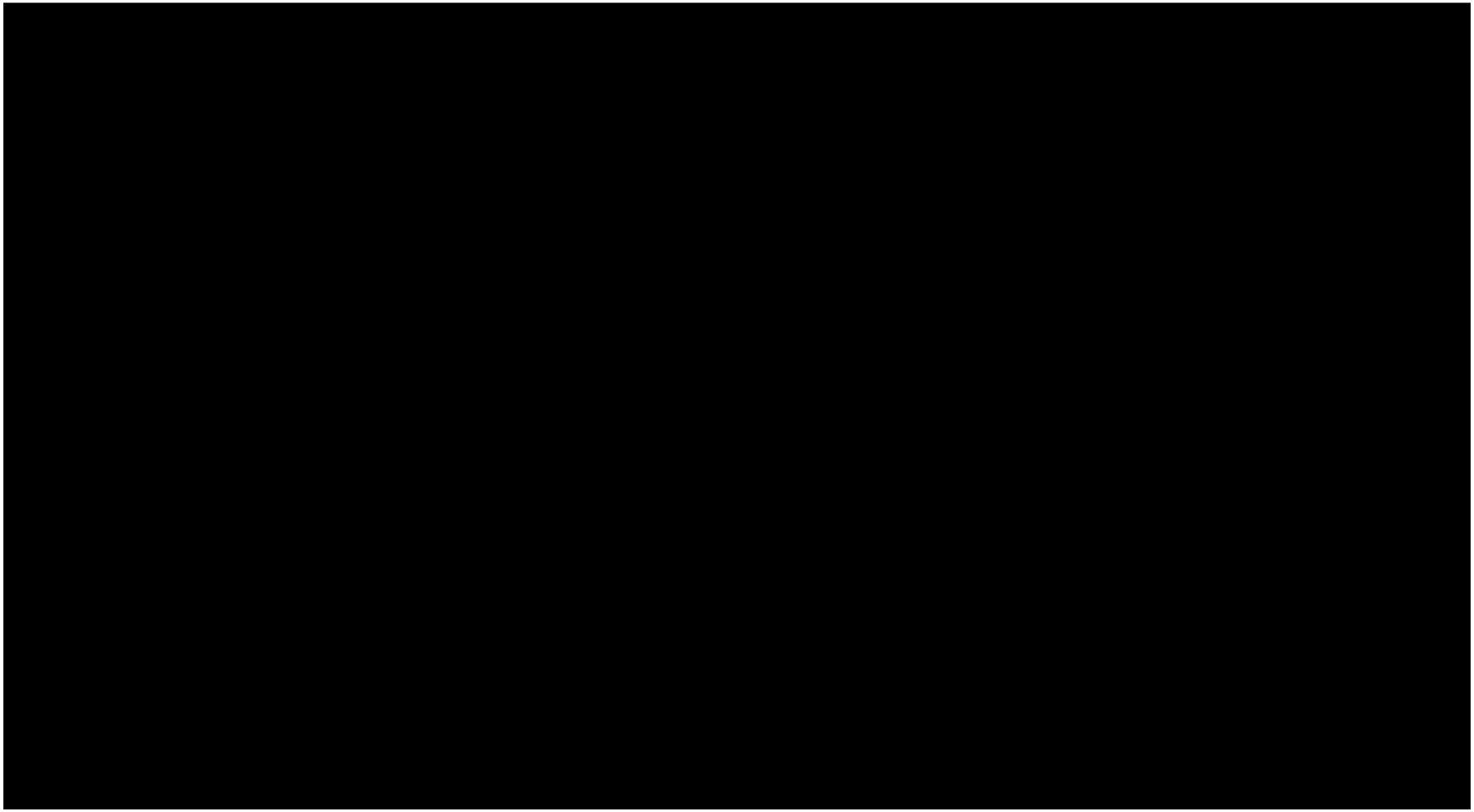
[redacted]

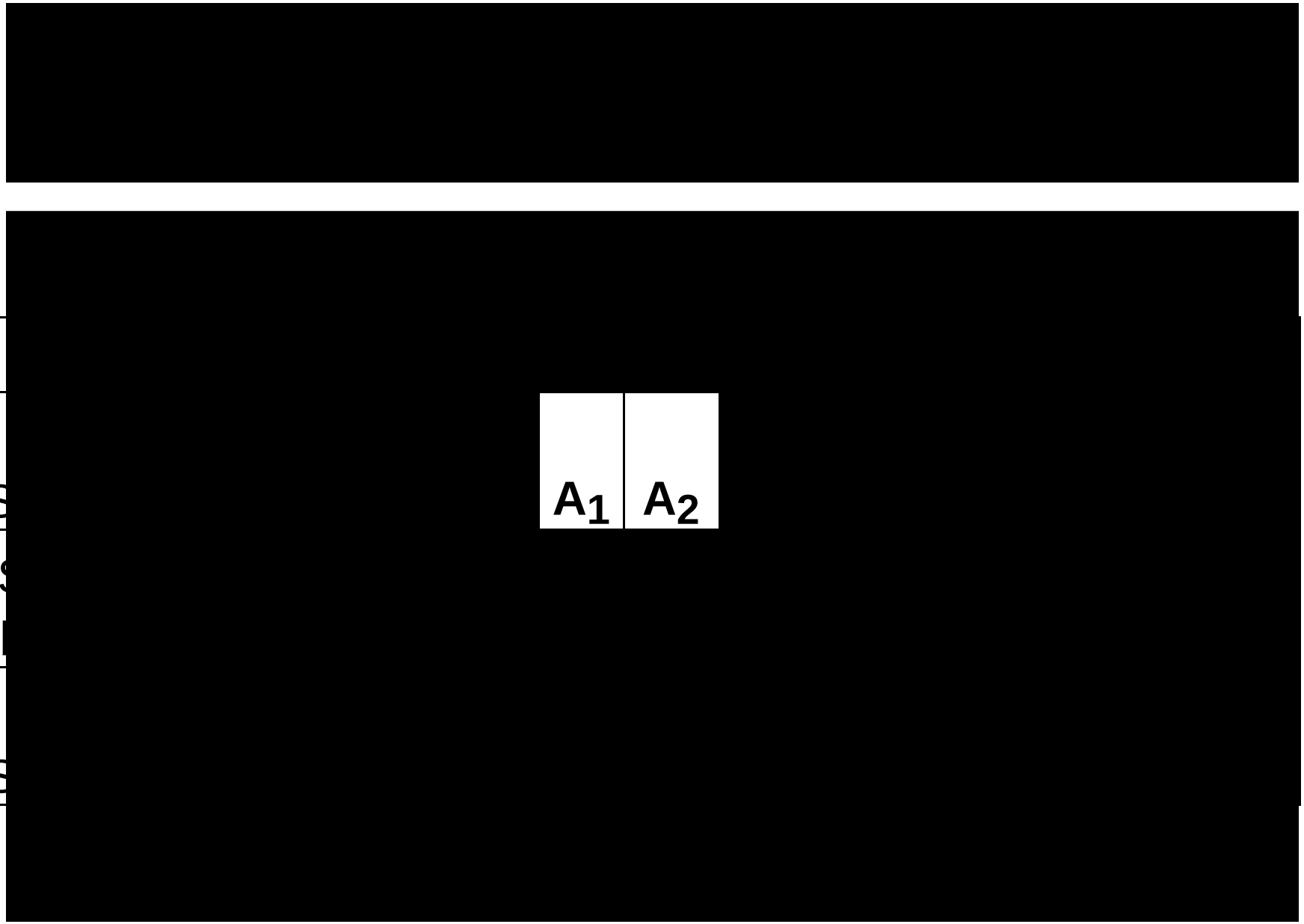
[redacted]



[redacted]






A_1
A_2