Unit 12
CONCURRENT PROCESSES

Announcements

- Lab Exam
- PA10
- Final Exam

Concurrency in Real Life

- Concurrency is the simultaneous occurrence of events.

- Most complex tasks can be broken down into a set of simpler activities
  - Building a house: bricklaying, carpentry, plumbing, electrical installation, roofing
  - Some of them can overlap and take place concurrently

Concurrency in Computing

- On the **Internet**: independent, autonomous agents try to achieve individual and shared goals.
- Our **local machines** can do more than one thing at a time.
  - While using a word processor, we can download files, manage the print queue, and stream audio.
- **Sequential programs:** Subprograms do not overlap in time
- **Concurrent programs**: Subprograms may overlap in time, their executions proceed concurrently

Why Do We Need It?

- Everything happens at once in the world.
  - For example, traffic control, airline seat reservation, process control, banking
- Performance gain from multiprocessing hardware
  - For example, Google, Yahoo, divide each query into thousands of little queries and use thousands of small computers.
  - For example, a supercomputer with thousands of processors can compute a weather prediction much faster than a single processor.
- Increased application throughput (amount of work that a computer can do in a given time period).
  - When one application is waiting for I/O another can continue its execution.

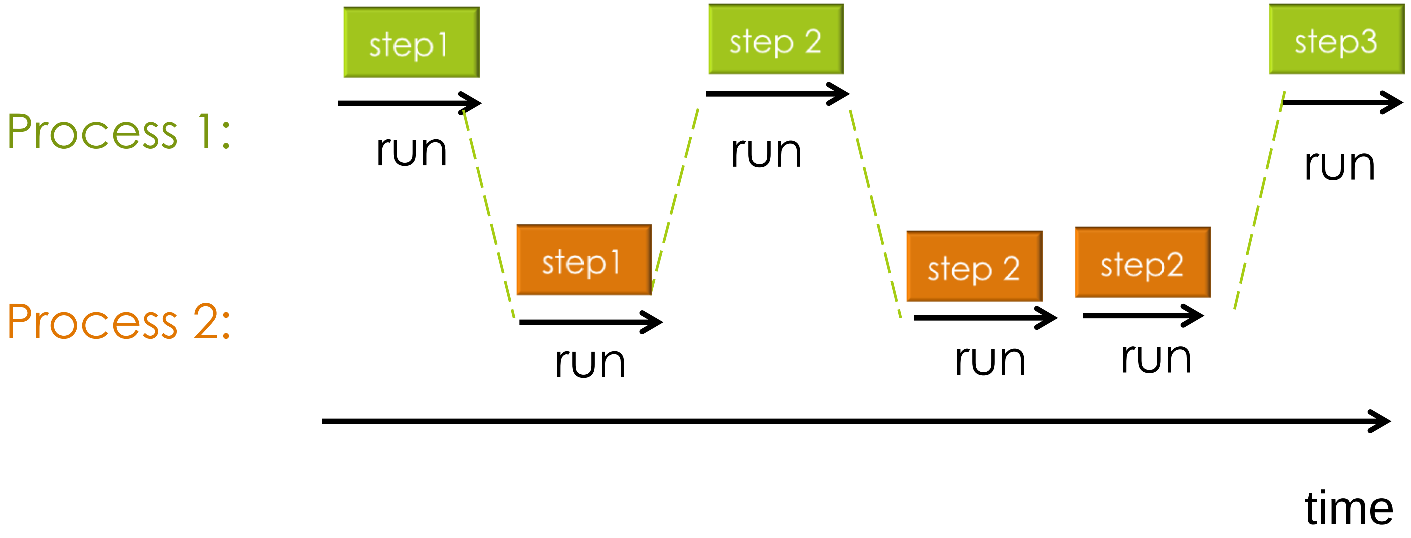
A Useful abstraction: Process

- Process: A program in execution
  - Program along with its data in memory, open files, open communication channels etc.

- Concurrency involves
  - multiple processes running simultaneously on **multiple processors** or
  - on a single processor **time-sharing** the processor.

Sharing a Processor

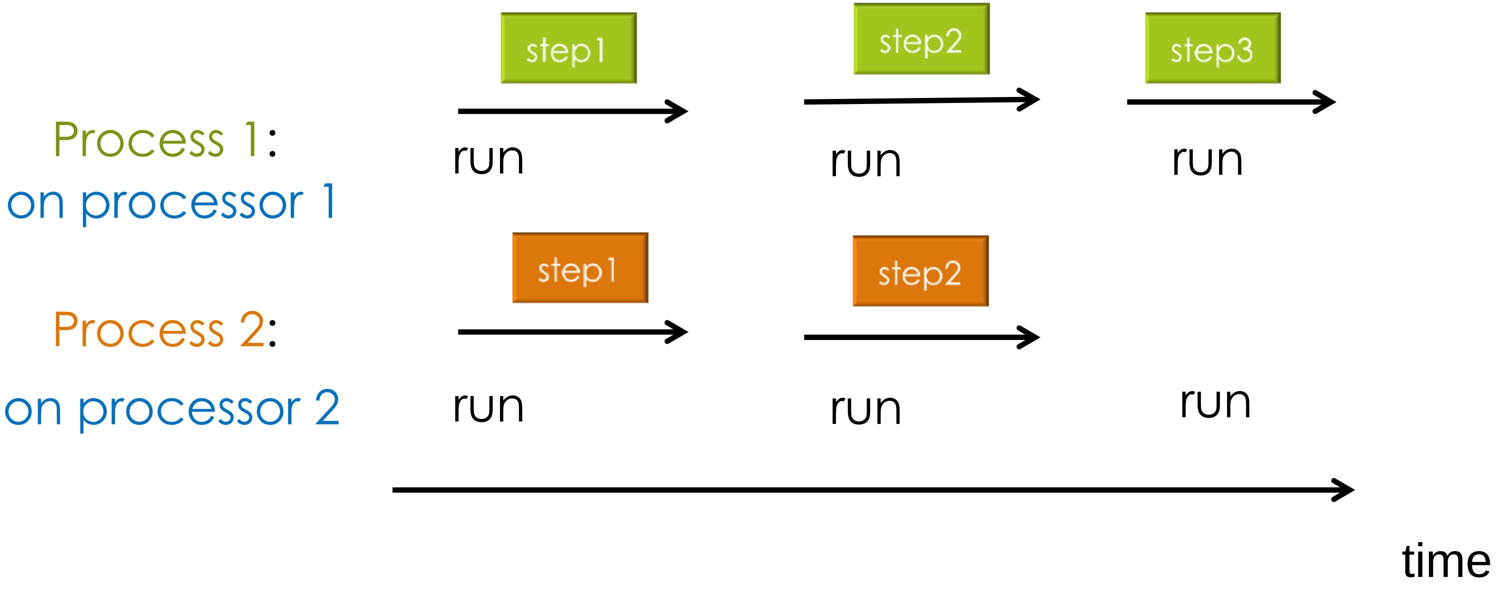
If only **one processor** (CPU) is available, the only way to run multiple processes is by **switching** between them.



Only one process is using the CPU at a given time even though they look like they are running in parallel to an observer.

Multiple Processors

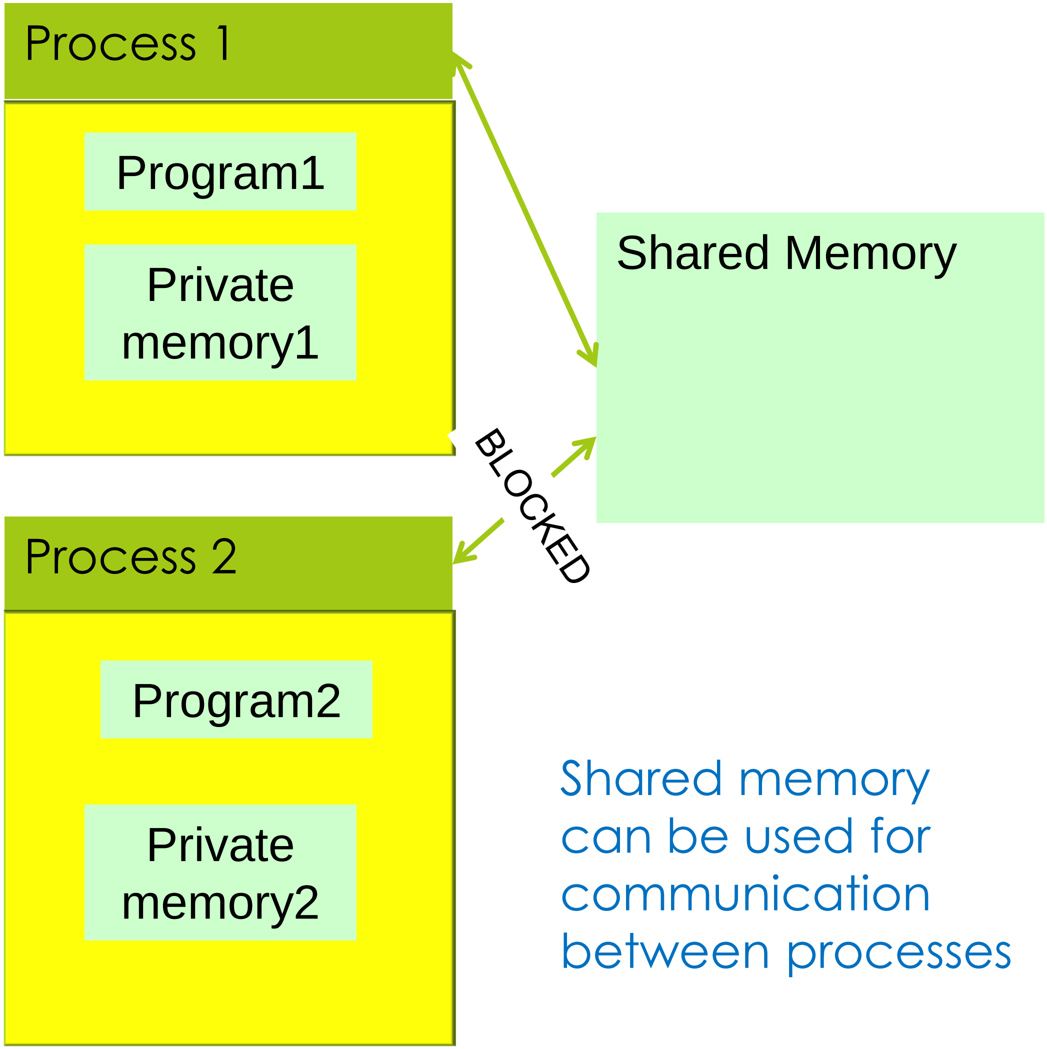
If you have **multiple CPUs**, you may execute multiple processes in parallel (simultaneously). Really!



Sharing Memory

- Processes may share resources such as memory

- For example, only one processor at a time may execute an instruction that touches the shared memory. The memory hardware makes the others wait.

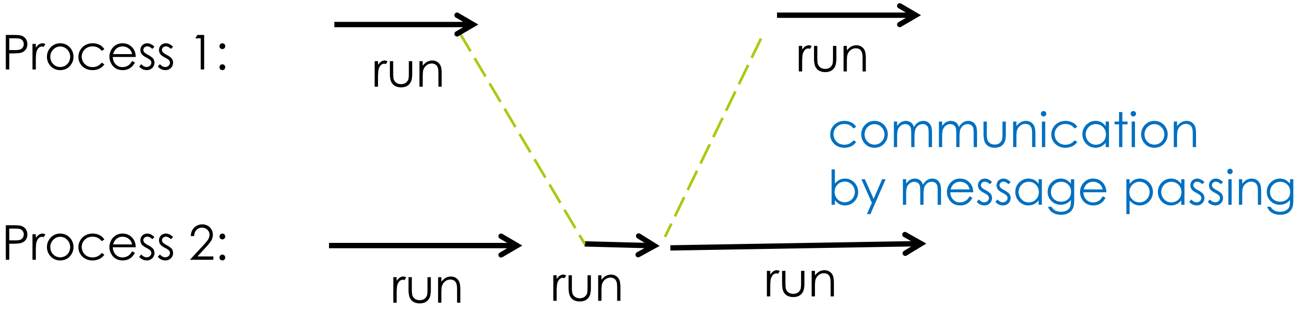


Shared memory can be used for communication between processes

Distributed Computing

Processes may be running on distributed systems

- For example, a cluster of workstations, communicating via sockets



Process 1:

run

run

communication by message passing

Process 2:

run

run

run

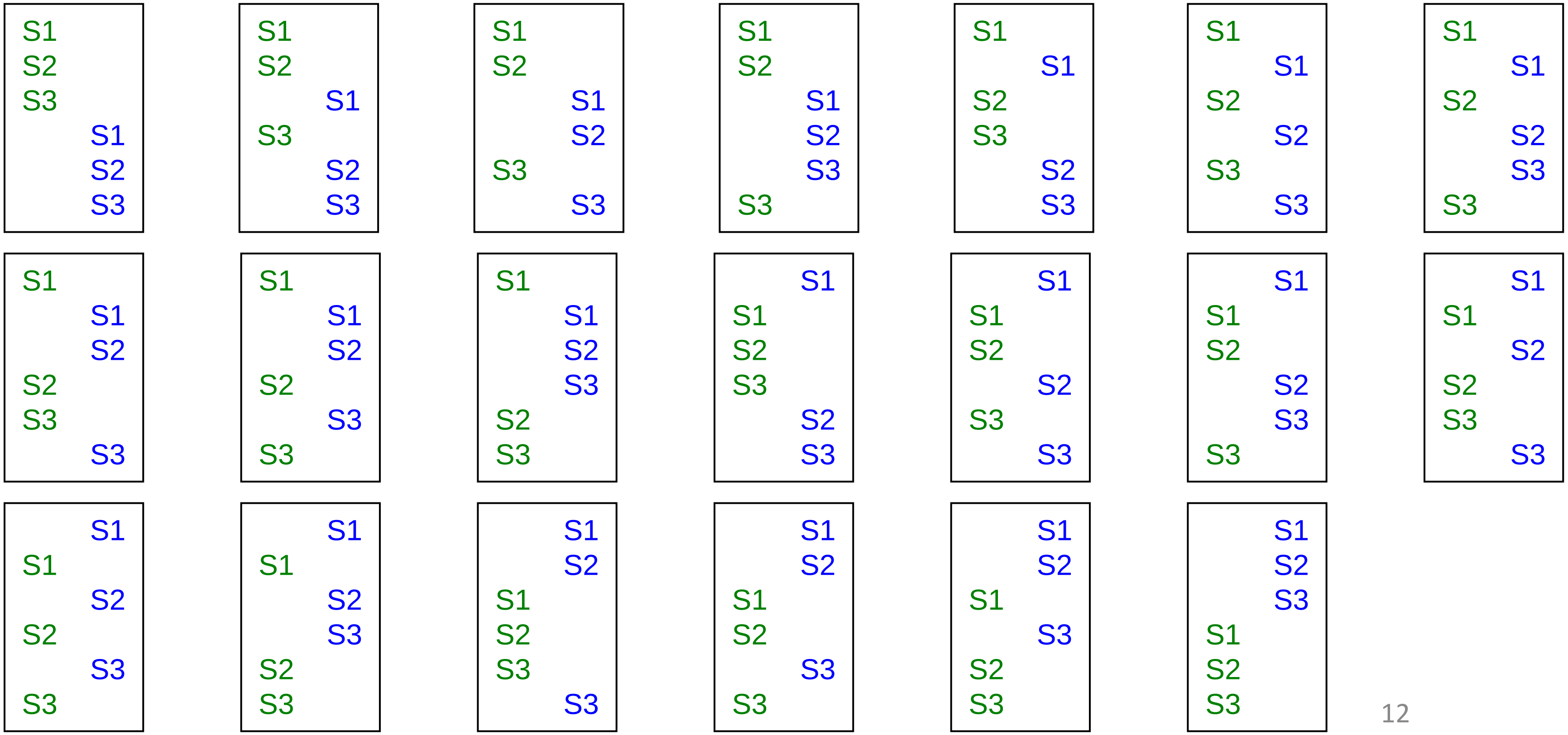
Some steps are executed simultaneously but some are dependent on another

There are many ways to execute two processes concurrently.

The green process executes steps S1 S2 S3 in the given order.
The blue process executes steps S1 S2 S3 in the given order.

S1 S2 S3	S1 S2 S3

Several possible interleavings of steps.



S1	
S2	
S3	
	S1
	S2
	S3

S1	
S2	
	S1
S3	
	S2
	S3

S1	
S2	
	S1
	S2
S3	
	S3

S1	
S2	
	S1
	S2
	S3
S3	

S1	
	S1
S2	
S3	
	S2
	S3

S1	
	S1
S2	
	S2
S3	
	S3

S1	
	S1
S2	
	S2
	S3
S3	

S1	
	S1
	S2
S2	
S3	
	S3

S1	
	S1
	S2
S2	
	S3
S3	

S1	
	S1
	S2
	S3
S2	
S3	

	S1
S1	
S2	
S3	
	S2
	S3

	S1
S1	
S2	
	S2
S3	
	S3

	S1
S1	
S2	
	S2
	S3
S3	

	S1
S1	
	S2
S2	
S3	
	S3

	S1
S1	
	S2
S2	
	S3
S3	

	S1
S1	
	S2
	S3
S2	
S3	

	S1
	S2
S1	
S2	
S3	
	S3

	S1
	S2
S1	
S2	
	S3
S3	

	S1
	S2
S1	
	S3
S2	
S3	

	S1
	S2
	S3
S1	
S2	
S3	

Critical Sections

- Often, a process really needs exclusive access to some data.
- A **critical section** is a sequence of steps that have exclusive access to the shared memory.
- Real Life Examples
  - Crossing a traffic intersection
  - A bank with many ATMs
  - Making a ticket reservation

Critical Section Example

- Consider a bank with multiple ATM's.
- At one, Mr. J requests a withdrawal of $10.
- At another, Ms. J requests a withdrawal of $10 from the same account.
- The bank's computer executes:
  1. For Mr. J, verify that the balance is big enough.
  2. For Ms. J, verify that the balance is big enough.
  3. Subtract 10 from the balance for Mr. J.
  4. Subtract 10 from the balance for Ms. J.

- The balance went negative if it was less than $20!

Vocabulary Reminder

- Race condition: A behavior in concurrent processing where proper functioning depends on the timing of other uncontrollable events

- A critical section is a piece of code that accesses a shared resource that must **not** be concurrently accessed by more than one process

Deadlock

- Deadlock is the condition when two or more processes are all waiting for some shared resource, but no process actually has it to release, so all processes to wait forever without proceeding.
- It’s like gridlock in real traffic.

PIPELINING

Pipelining

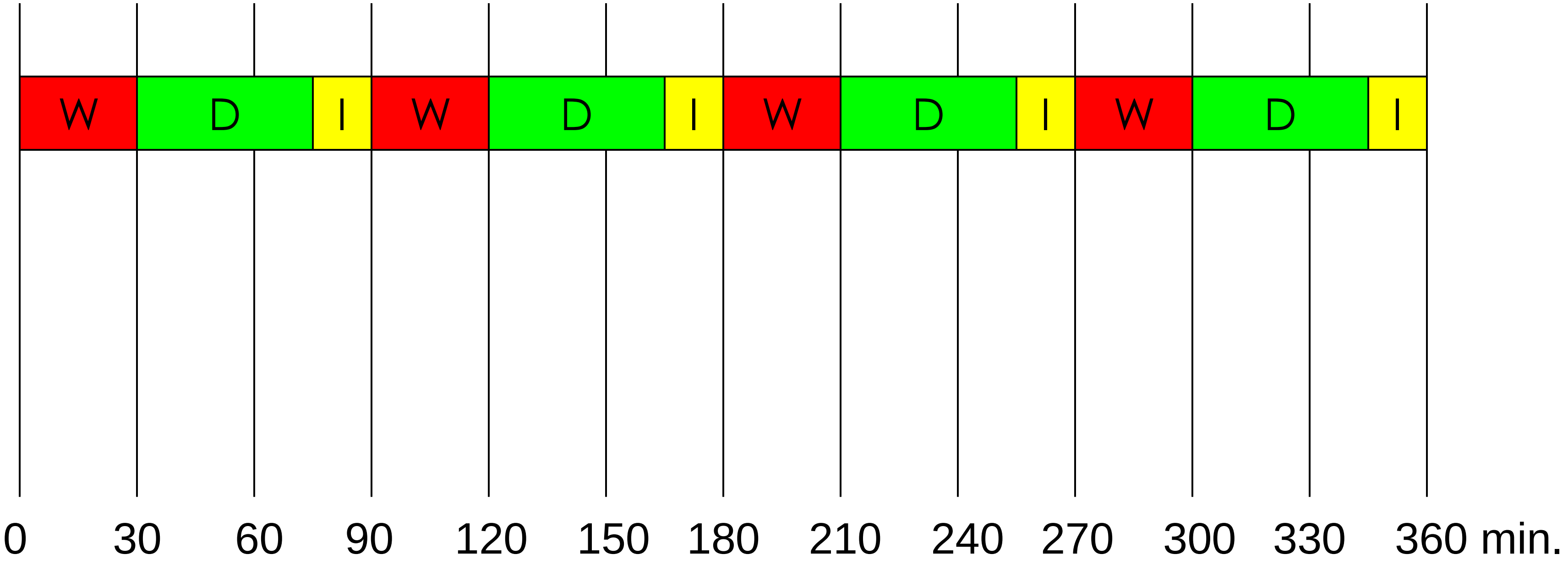
- Pipelining is similar to an assembly line.
  - Instead of completing one computation before starting another, each computation is split into simpler sub-steps, and computations are started as others are in progress.

Laundry Without Pipelining

Washing, Drying and Ironing four loads of laundry.

W	D	I
Washing (30 min)	Drying (45 min)	Ironing (15 min)

WITHOUT PIPELINING: 6 hr



Laundry With Pipelining

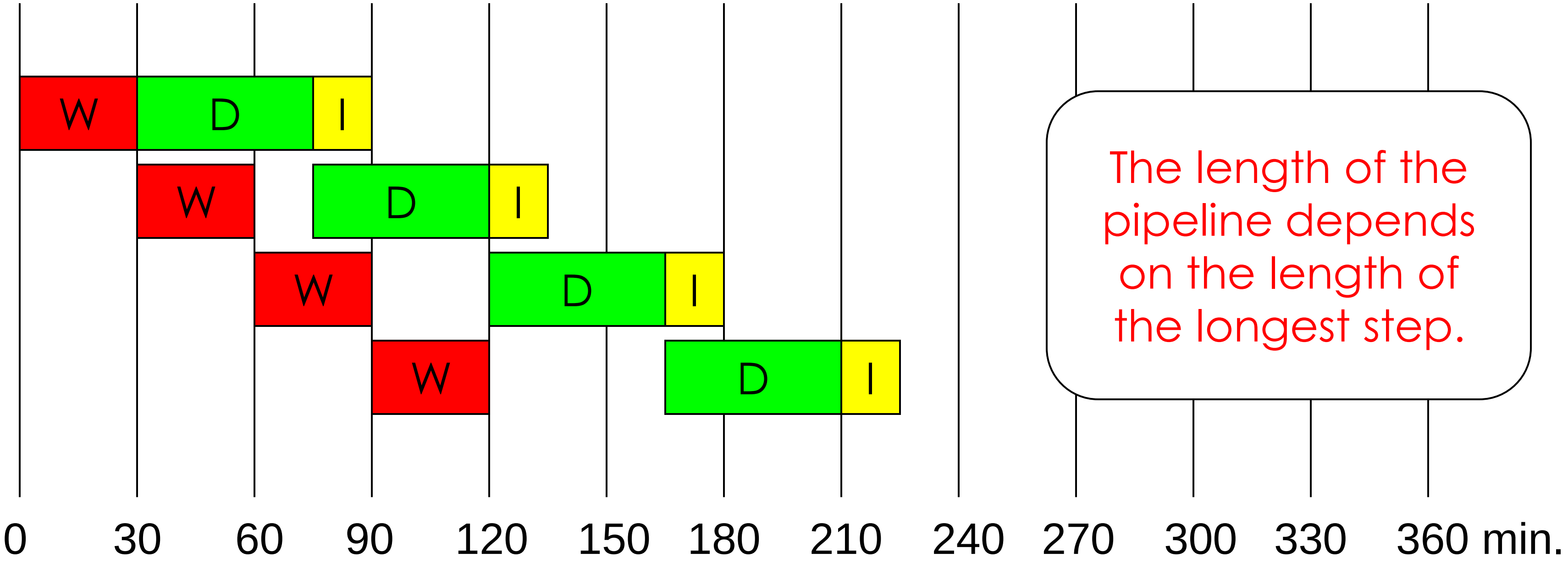
Washing, Drying and Ironing four loads of laundry.

W Washing (30 min)

D Drying (45 min)

I Ironing (15 min)

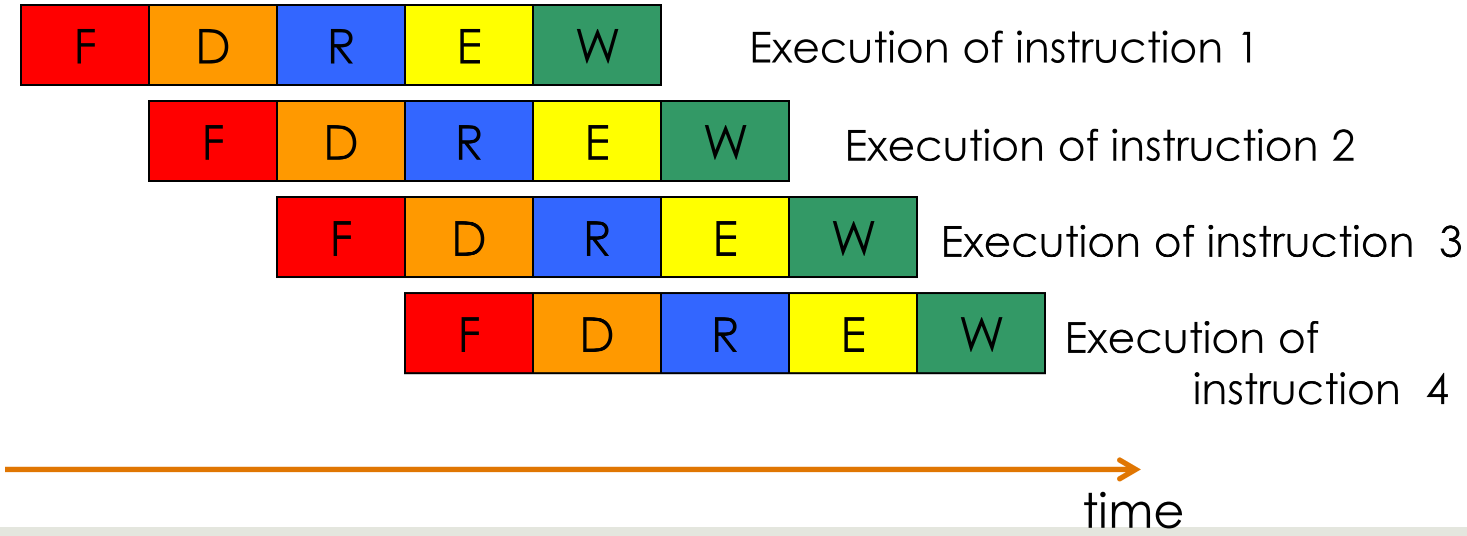
WITH PIPELINING: 3 hr 45 min



The length of the pipeline depends on the length of the longest step.

Pipelining in Computing

- **F**etch instruction from memory
- **D**ecode the instruction
- **R**ead data from registers
- **E**xecute the instruction
- **W**rite the result into a register



F D R E W Execution of instruction 1

F D R E W Execution of instruction 2

F D R E W Execution of instruction 3

F D R E W Execution of instruction 4

time

Dealing with Dependencies

```
ADD R3, R1
ADD R5, R3
ADD R8, R7
ADD R11, R10
```

"Add the contents of R1 and R3 and store the results in R3.”

This instruction depends on the result of the previous instruction.

What does this mean for pipelining?

Dealing with Dependencies

ADD R3, R1

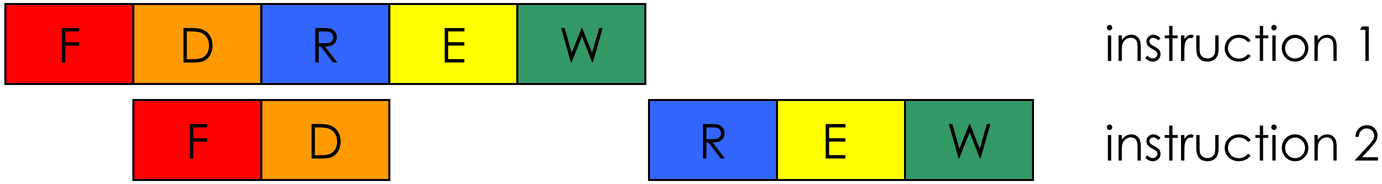
"Add the contents of R1 and R3 and store the results in R3."

ADD R5, R3

This instruction depends on the result of the previous instruction. (This will hold up the pipeline. We cannot do the R step for the second instruction before finishing the W step for the first instruction.)

ADD R8, R7

ADD R11, R10



Dealing with Dependencies

ADD R3, R1

ADD R5, R3

ADD R8, R7

ADD R11, R10

"Add the contents of R1 and R3 and store the results in R3."

This instruction depends on the result of the previous instruction. (This will hold up the pipeline. We cannot do the R step for the second instruction before finishing the W step for the first instruction.)

ADD R3, R1

ADD R8, R7

ADD R11, R10

ADD R5, R3

Reorder the instructions to minimize the delay on the pipeline due to the dependency, if possible.

F	D	R	E	W			
	F	D	R	E	W		
		F	D	R	E	W	
			F	D	R	E	W

Matrix Multiplication

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	

Matrix Multiplication

0 + 95*0.15 + 90*0.1 + 93*0.15 + 91*0.15 + 85*0.15 + 92*0.3 = 91.2

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	grade
student1	91.2
student2	
student3	
student4	
student5	
student6	
student7	
student8	

Matrix Multiplication

0 + 73*0.15 + 80*0.1 + 75*0.15 + 63*0.15 + 79*0.15 + 75*0.3 = 74.0

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	grade
student1	91.2
student2	74.0
student3	
student4	
student5	
student6	
student7	
student8	

Matrix Multiplication

0 + 85*0.15 + 73*0.1 + 80*0.15 + 85*0.15 + 88*0.15 + 91*0.3 = 85.3

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	grade
student1	91.2
student2	74.0
student3	85.3
student4	
student5	
student6	
student7	
student8	

....and so on...

Matrix Multiplication

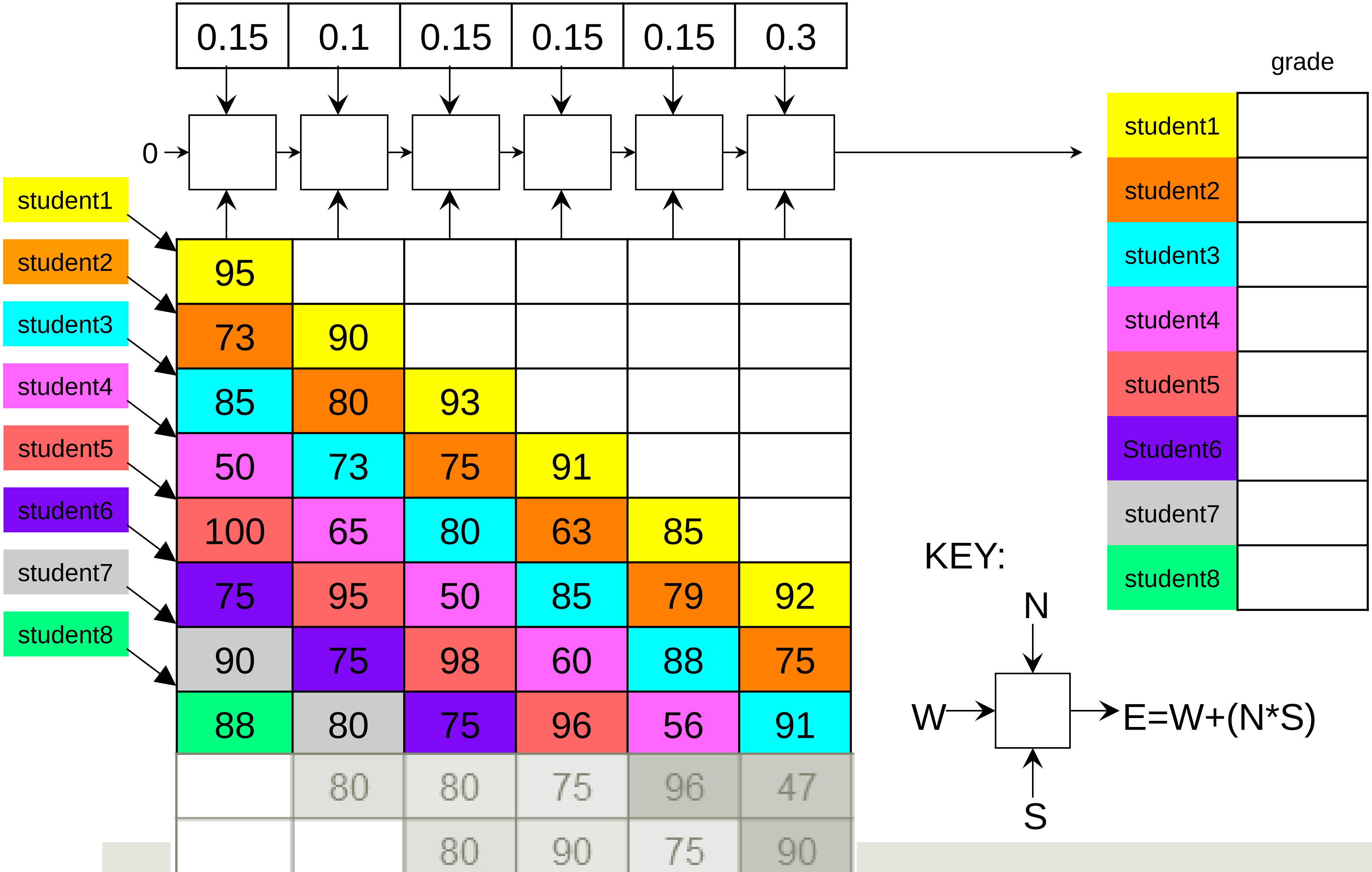
**If each multiply/add takes 1 time unit,
this non-pipelined matrix multiplication takes 48 time units.**

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

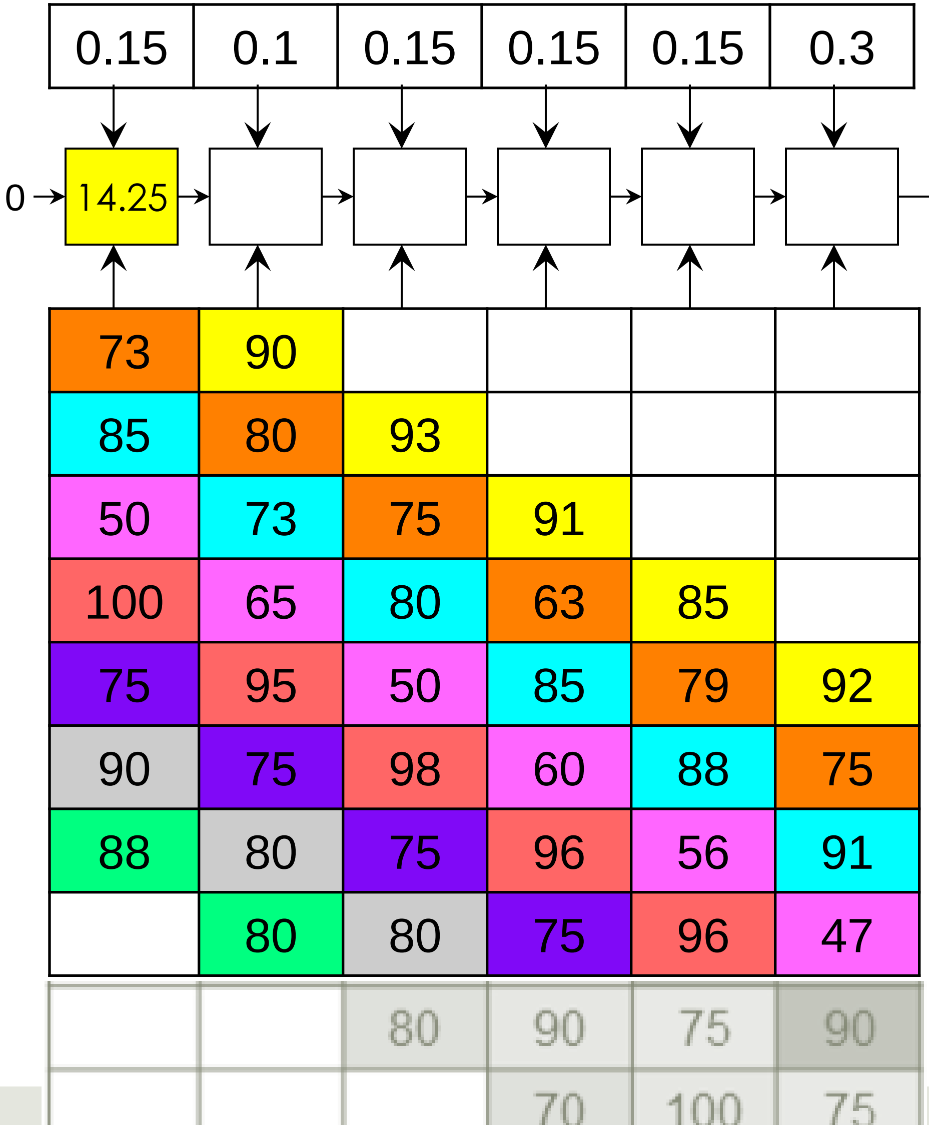
	grade
student1	91.2
student2	74.0
student3	85.3
student4	53.0
student5	95.0
student6	75.0
student7	92.0
student8	69.2

Faster Matrix Multiplication
using Pipelining



0.15	0.1	0.15	0.15	0.15	0.3
95					
73	90				
85	80	93			
50	73	75	91		
100	65	80	63	85	
75	95	50	85	79	92
90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90

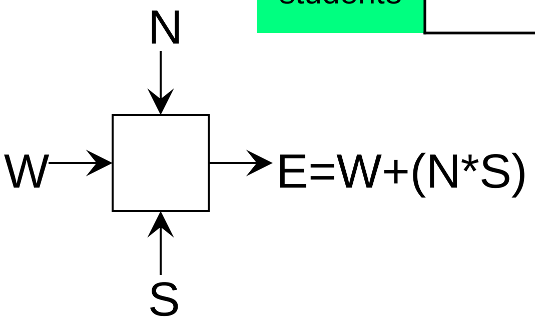
	grade
student1	
student2	
student3	
student4	
student5	
Student6	
student7	
student8	

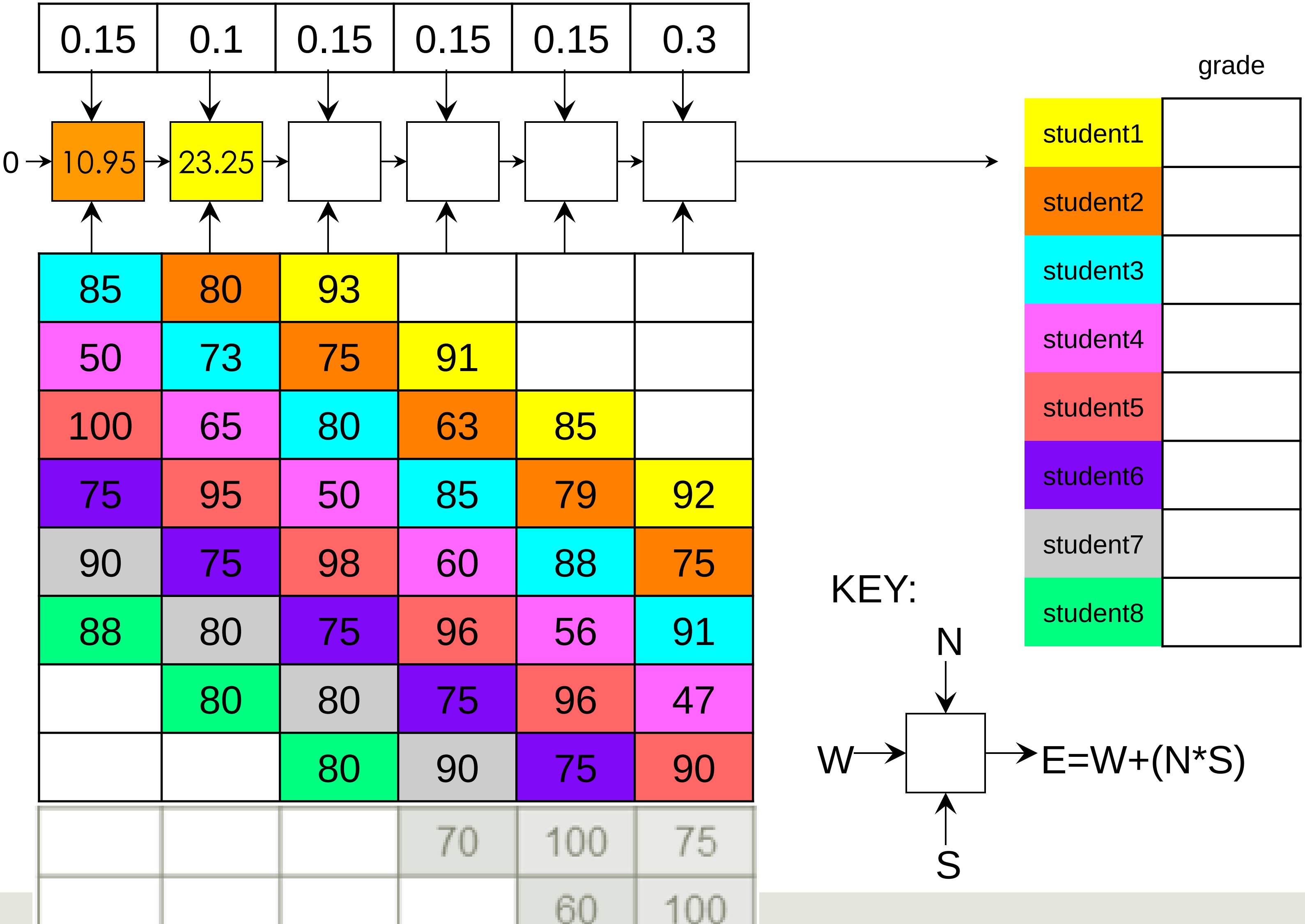


73	90				
85	80	93			
50	73	75	91		
100	65	80	63	85	
75	95	50	85	79	92
90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75

	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	

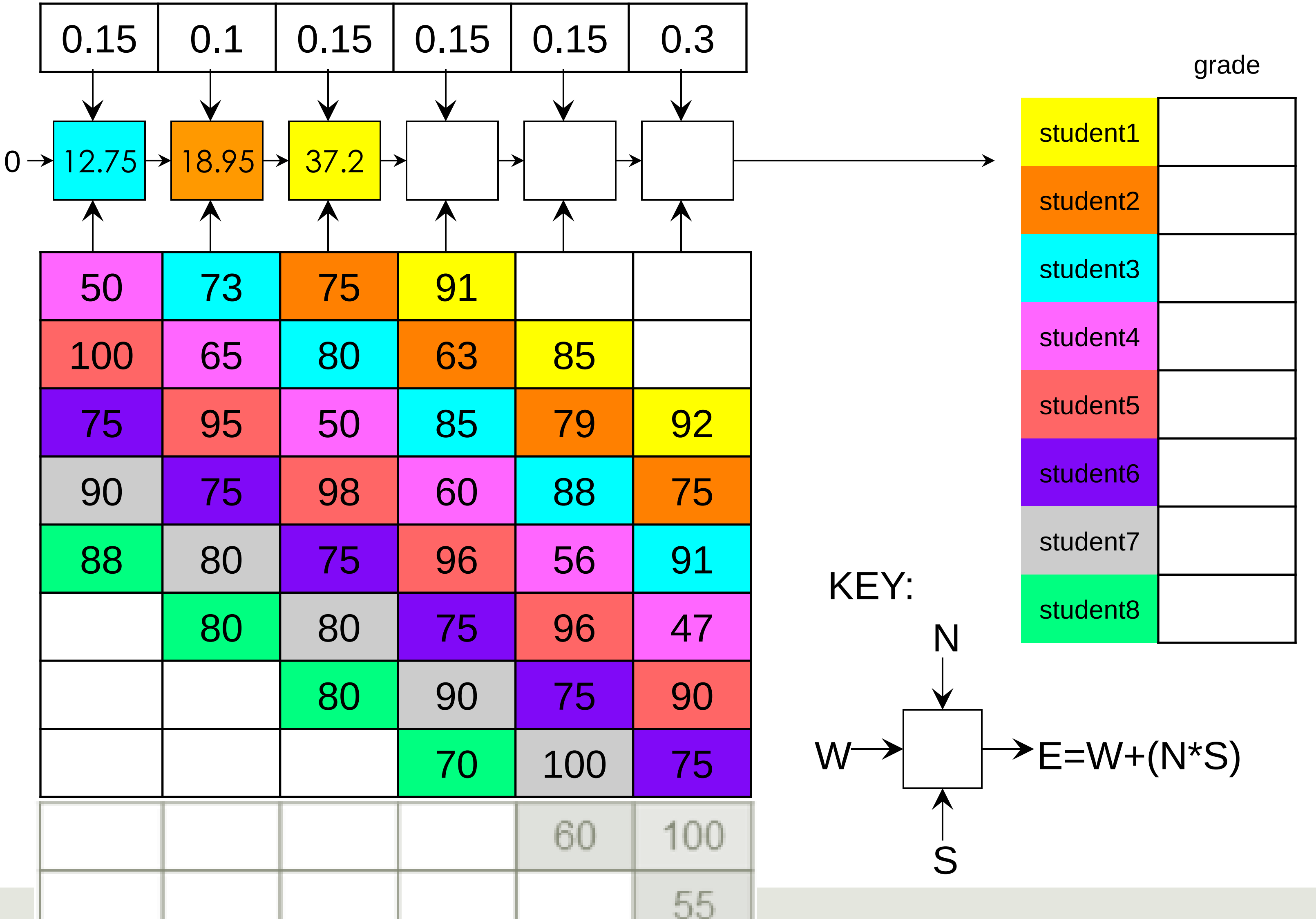
KEY:





0.15	0.1	0.15	0.15	0.15	0.3
85	80	93			
50	73	75	91		
100	65	80	63	85	
75	95	50	85	79	92
90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100

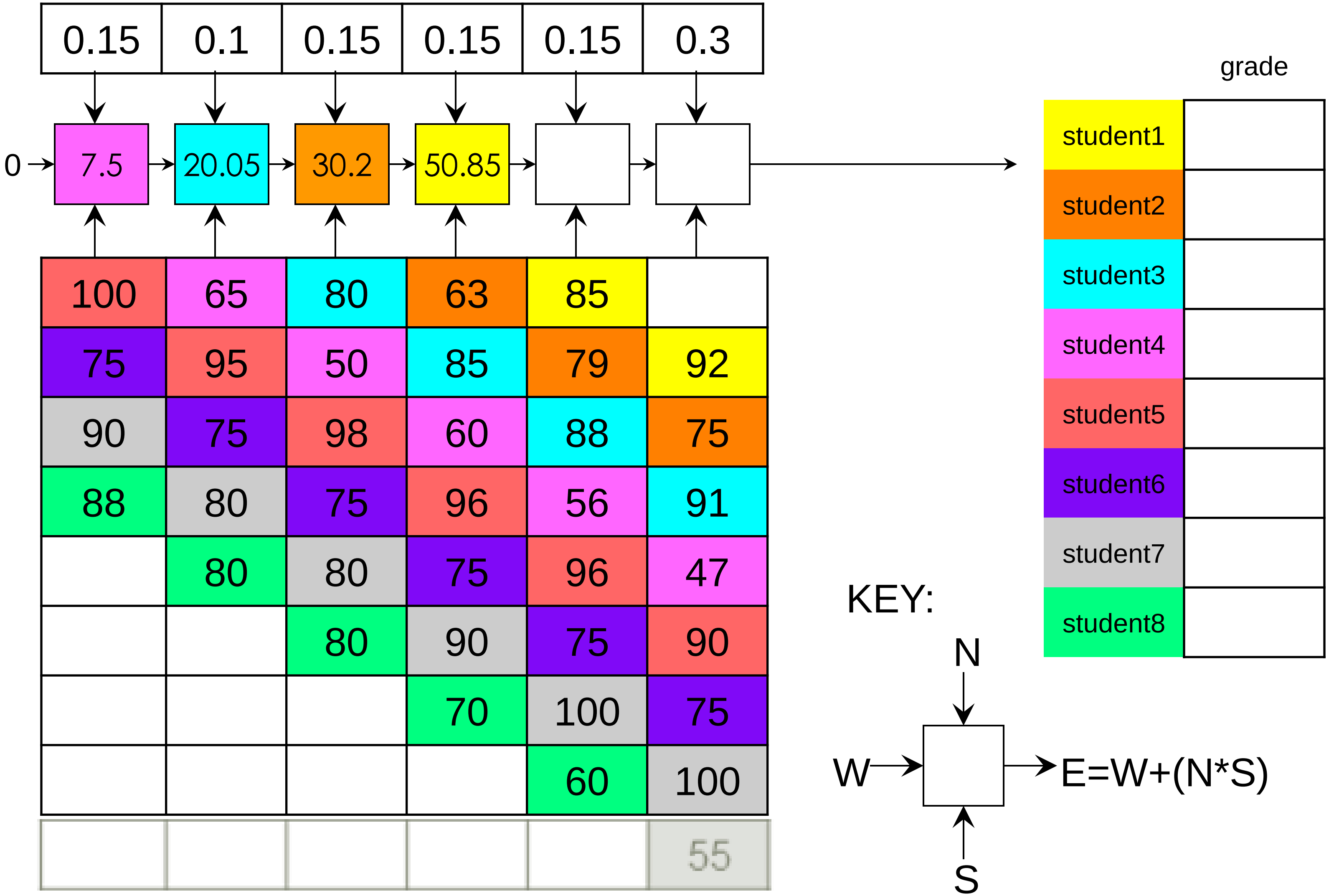
	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	



0.15	0.1	0.15	0.15	0.15	0.3

50	73	75	91		
100	65	80	63	85	
75	95	50	85	79	92
90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100
					55

	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	



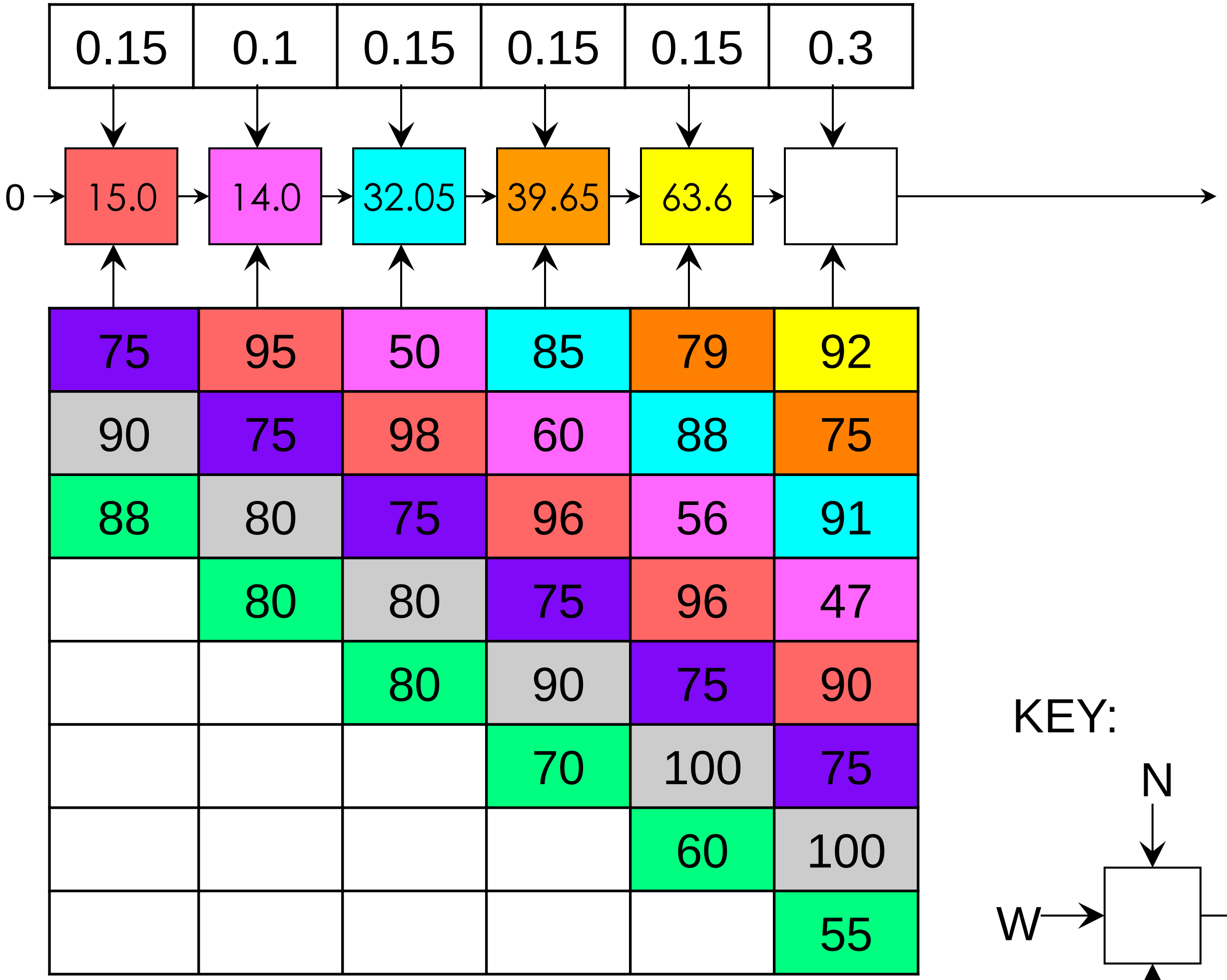
0.15	0.1	0.15	0.15	0.15	0.3

0 → 7.5 → 20.05 → 30.2 → 50.85 → → →

100	65	80	63	85	
75	95	50	85	79	92
90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100
					55

	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	

KEY:



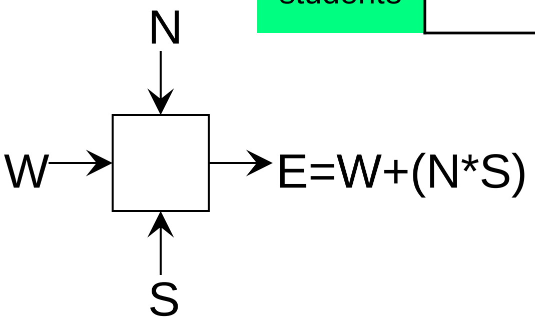
0.15	0.1	0.15	0.15	0.15	0.3

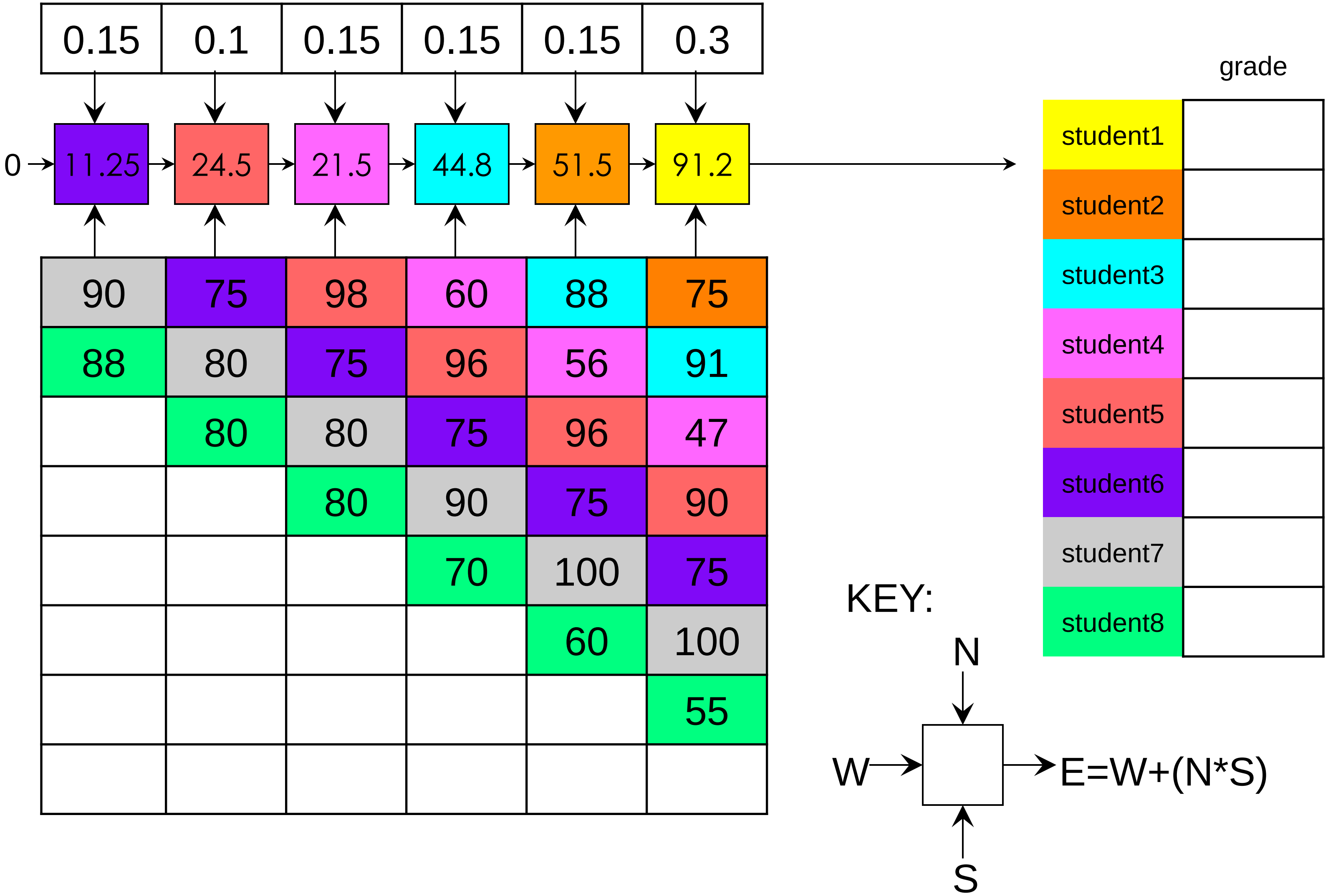
15.0	14.0	32.05	39.65	63.6	

75	95	50	85	79	92
90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100
					55

	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	

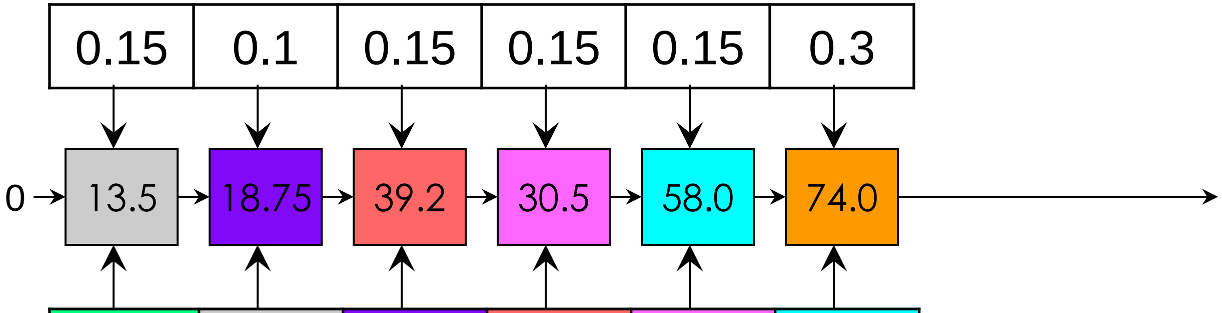
KEY:





90	75	98	60	88	75
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100
					55

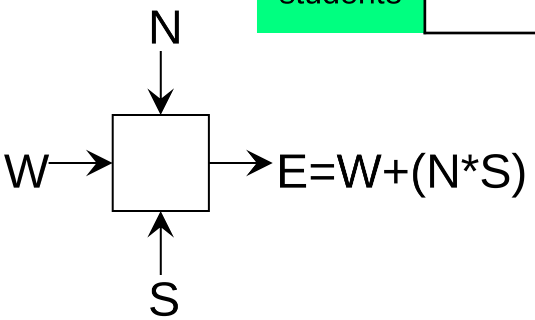
	grade
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	



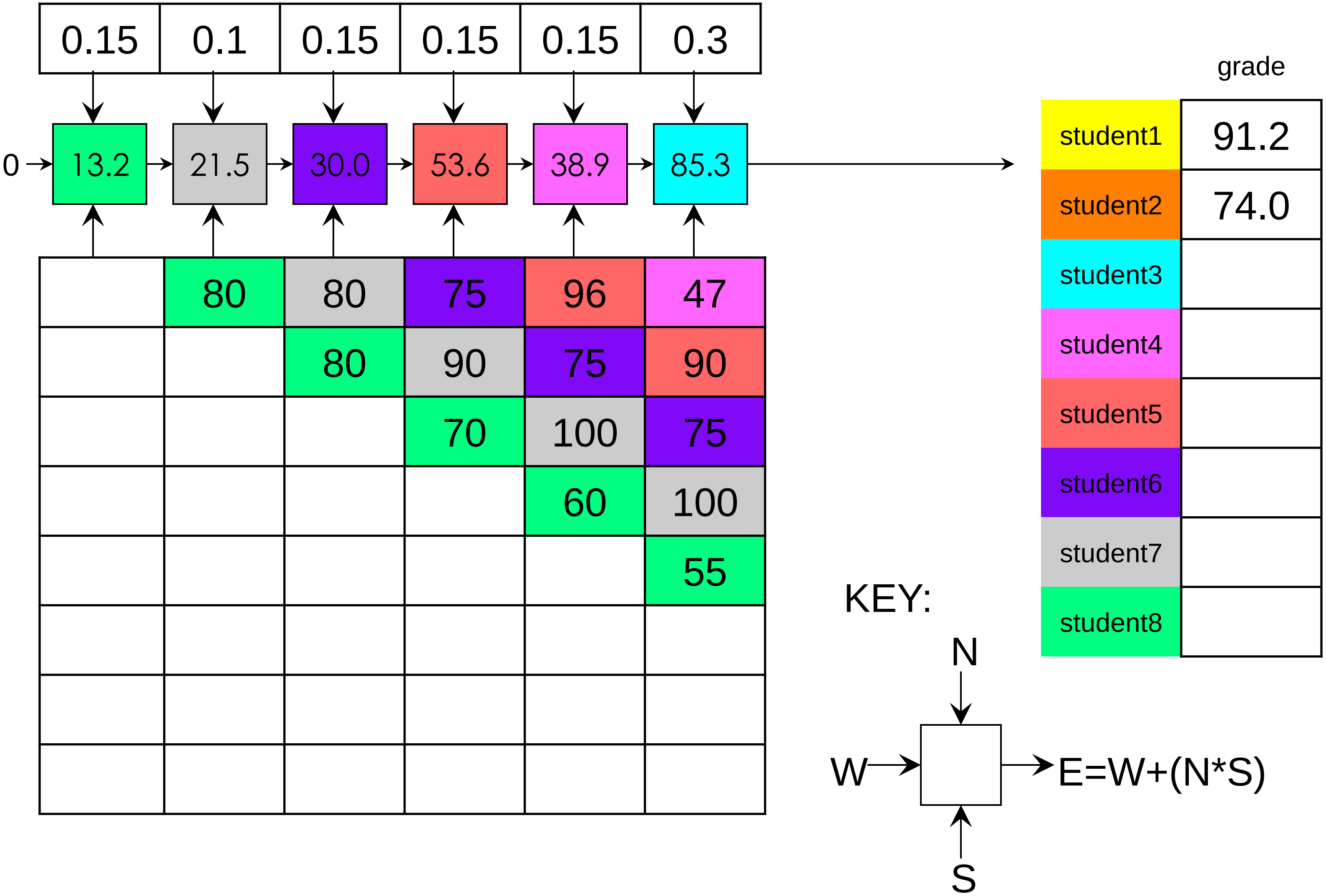
88	80	75	96	56	91
	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100
					55

	grade
student1	91.2
student2	
student3	
student4	
student5	
student6	
student7	
student8	

KEY:



E=W+(N*S)



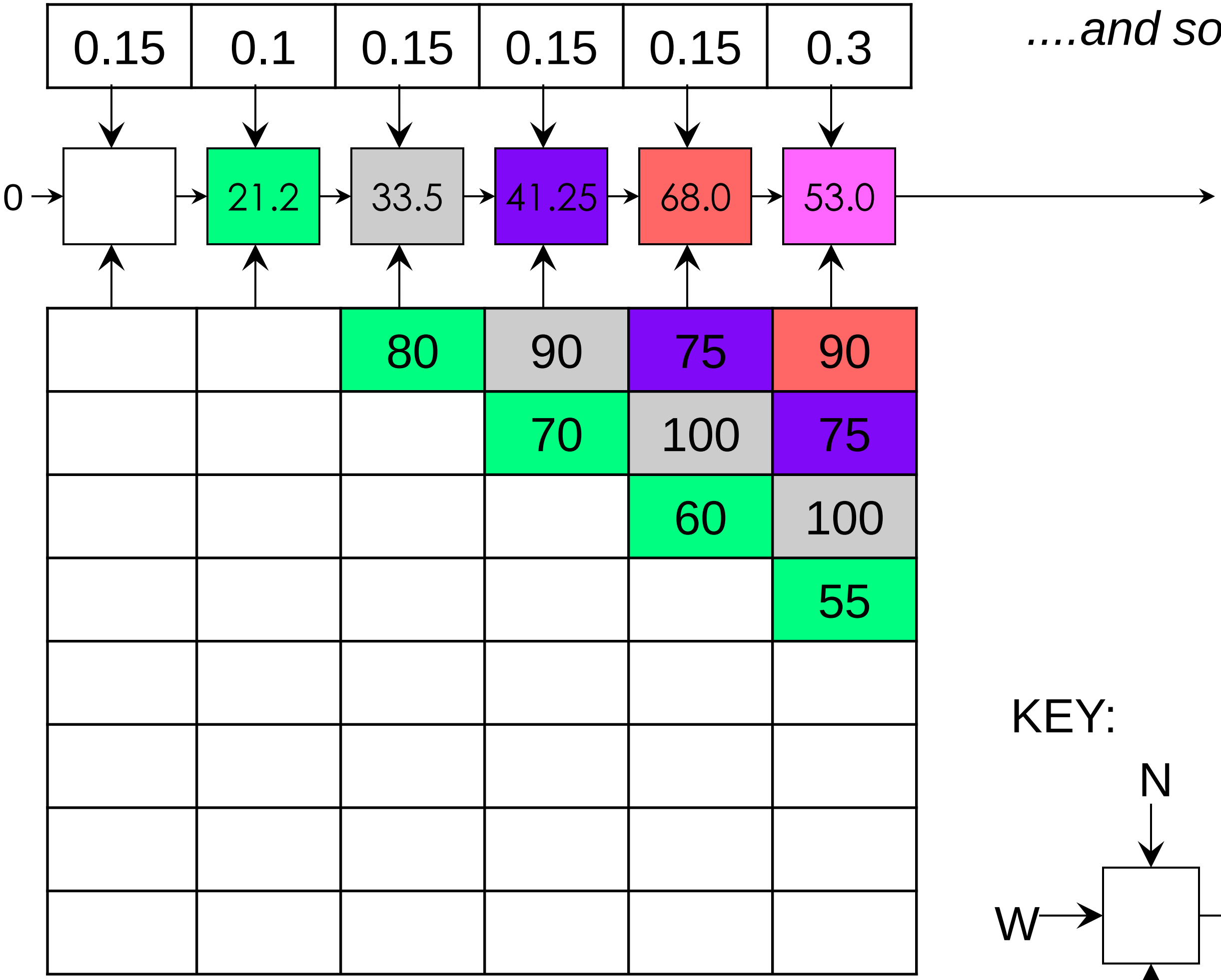
0.15	0.1	0.15	0.15	0.15	0.3

	80	80	75	96	47
		80	90	75	90
			70	100	75
				60	100
					55

	grade
student1	91.2
student2	74.0
student3	
student4	
student5	
student6	
student7	
student8	

KEY:

E=W+(N*S)

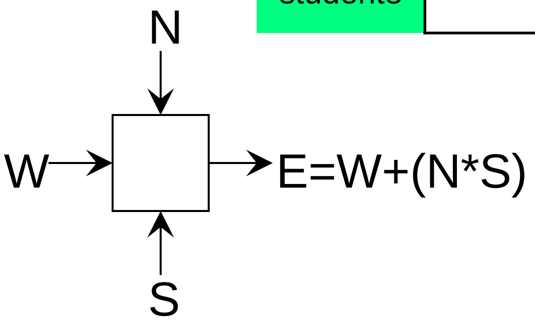


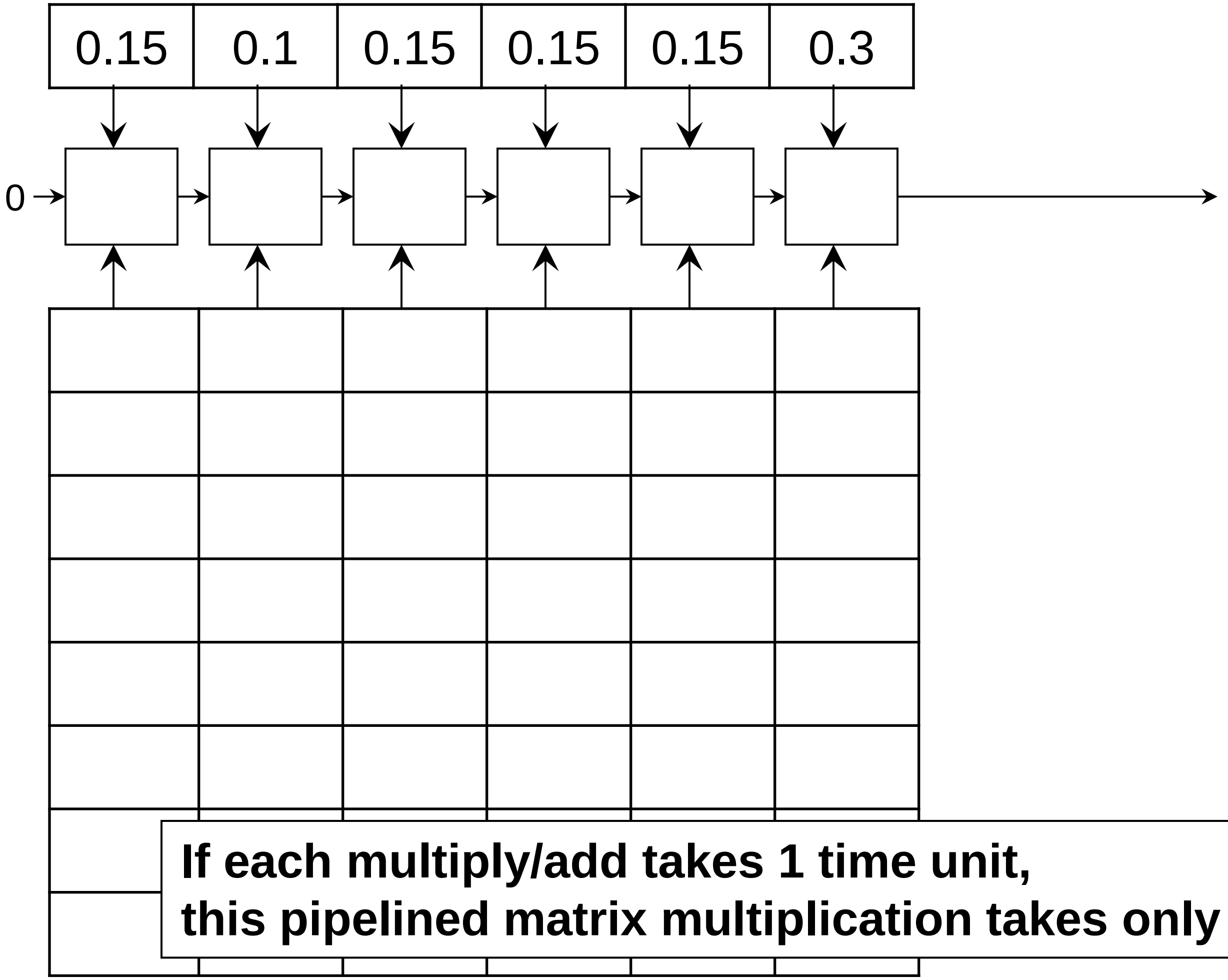
		80	90	75	90
			70	100	75
				60	100
					55

....and so on...

	grade
student1	91.2
student2	74.0
student3	85.3
student4	
student5	
student6	
student7	
student8	

KEY:





	grade
student1	91.2
student2	74.0
student3	85.3
student4	53.0
student5	95.0
student6	75.0
student7	92.0
student8	69.2

If each multiply/add takes 1 time unit,
this pipelined matrix multiplication takes only 13 time units.

Summary

- Concurrency means execution of multiple processes at the same time. It may be implemented by interleaving steps of processes on a single processor or using multiple processors. The way we reason about concurrency in both scenarios is similar. We cannot make arbitrary assumptions about timings of steps so we have to consider all interleavings of steps to be possible.

- Processes may interact and coordinate in complex ways. Care must be taken when they share common resources, to deal with race conditions and to avoid deadlocks.

- Pipelining is a method that increases the throughput of a system when processing a stream of data. It is an example of using concurrency to make processing faster by processing parts of the data in parallel.