UNIT 10B
Concurrency: Pipelining

REVIEW: MULTI-TASKING , MULTIPROCESSING, DEADLOCK

Concurrency

- Concurrency is the process of performing more than one process at a time.
- Why do we need it?
  - Some computations require concurrency by their nature: for example, banking , airline seat reservation, process control etc.
  - Some computations can be sped up if we can figure out how to decompose computation we can also speed-up computing process.

Different Flavors

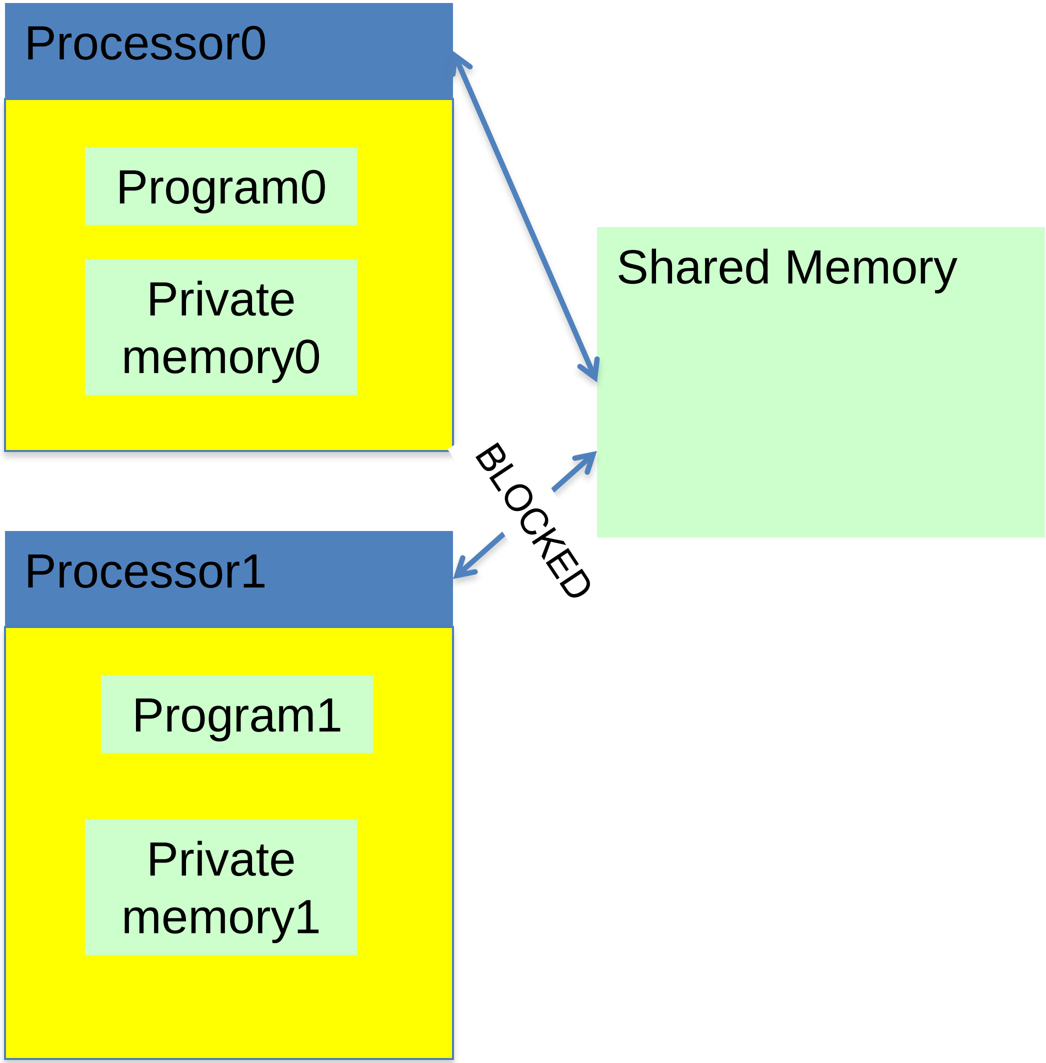
- Computing has many ways to implement concurrency:
  - Parallel processing (multiple processes, each process on a single processor)
  - Multitasking (multiple processes sharing a single processor. Think of running many apps at the same time. In fact only one of them is executed at a given time point)
  - Pipelining
  - Distributed computing

Concurrency and Ruby

- In this course, we only focus on writing sequential programs.
- Newer versions of Ruby support concurrency constructs such as threads.

A Ruby Multiprocessor Model

- The processors (thread) run independently.
- The shared memory is used for communication.
- Only one processor at a time may execute a line of Ruby that touches the shared memory. The memory hardware makes the others wait.



We reason about concurrent executions using all possible interleavings of the steps of sequential threads of execution.

Program 1	Program 2
S1	S1
S2	S2
S3	S3

S1	S1	S1	S1	S1	S1	S1
S2	S2	S2	S2	S1	S1	S1
S3	S1	S1	S1	S2	S2	S2
S1	S3	S2	S2	S3	S2	S2
S2	S2	S3	S3	S2	S3	S3
S3	S3	S3	S3	S3	S3	S3

S1	S1	S1	S1	S1	S1	S1
S1	S1	S1	S1	S1	S1	S1
S2	S2	S2	S2	S2	S2	S2
S2	S2	S3	S3	S2	S2	S2
S3	S3	S2	S2	S3	S3	S2
S3	S3	S3	S3	S3	S3	S2

S1	S1	S1	S1	S1	S1
S1	S1	S2	S2	S2	S2
S2	S2	S1	S1	S1	S3
S2	S3	S2	S2	S3	S1
S3	S2	S3	S3	S2	S2
S3	S3	S3	S3	S3	S3

Concurrent programming is hard.

- Only a tiny percentage of practicing programmers can do it.
- It requires art and mathematics.
  - It’s like digital hardware design.
  - It needs proofs.
- Conventional debugging doesn’t work.
  - If you stop the program to observe, you change the behavior.
  - Testing is futile because the number of possible execution sequences for the same input explodes.

Critical Section

- A **critical section** is a section of computer code that must only be executed by one process or thread at a time.
- Examples:
  - A printer queue receiving a file to be printed
  - Code to set a seat as reserved
  - Web server that generates and returns a web page showing current registration in course

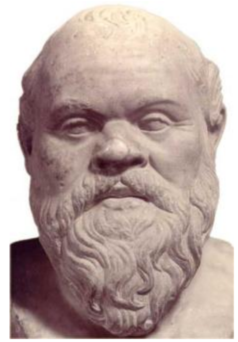
Shared Computing Resources

- memory
- tape drives
- disk drives
- printers
- communication ports
- input devices (keyboard, mouse)

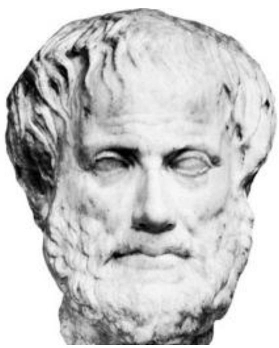
Deadlock

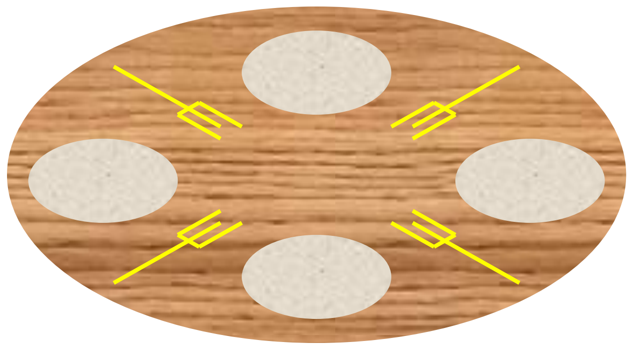
- Deadlock is the condition when two or more processes are all waiting for some shared resource that other processes of the group hold, causing all processes to wait forever without proceeding.

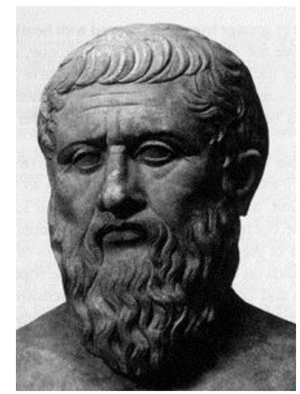
Dining Philosopher's Problem



Socrates







Aristotle

Plato

Homer

The Dining Philosopher’s

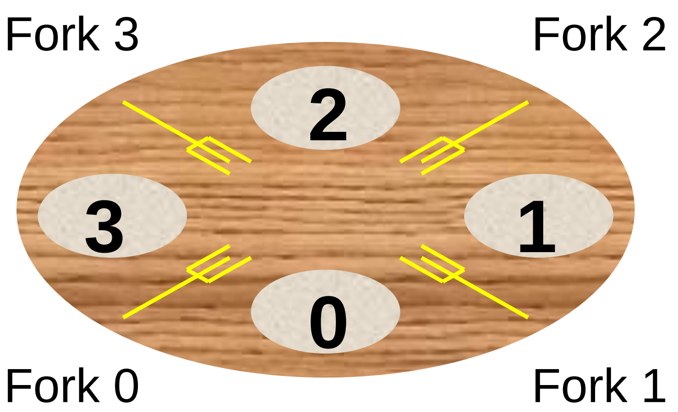
- Each philosopher thinks for a while, then picks up his left fork, then picks up his right fork, then eats, then puts down his left fork, then puts down his right fork, thinks for a while...
  - We assume here that each philosopher thinks and eats for random times, and a philosopher cannot be interrupted while he picks up or puts down a single fork.
- Each fork models a "resource" on a computer controlled by an OS.
- Original problem proposed by Edsgar Dijkstra.

Dining Philosopher's Problem

- There are N philosophers.
- Philosopher i does the following:

1. THINK
2. Pick up fork i.
3. Pick up fork (i+1) modulo N.
4. EAT
5. Put down fork i.
6. Put down fork (i+1) modulo N.
7. Go to step 1.

N=4



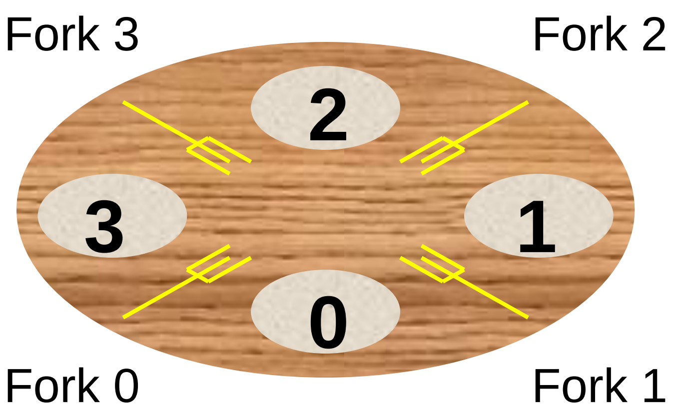
NOTE: (i+1) modulo N = i+1 , if $0 \leq i < N-1$
(i+1) modulo N = 0, if $i = N-1$

Dining Philosopher's Problem

- There are N philosophers.
- Philosopher i does the following:

1. THINK
2. Pick up fork i.
3. Pick up fork (i+1) modulo N.
4. EAT
5. Put down fork i.
6. Put down fork (i+1) modulo N.
7. Go to step 1.

N=4



How can deadlock occur here?

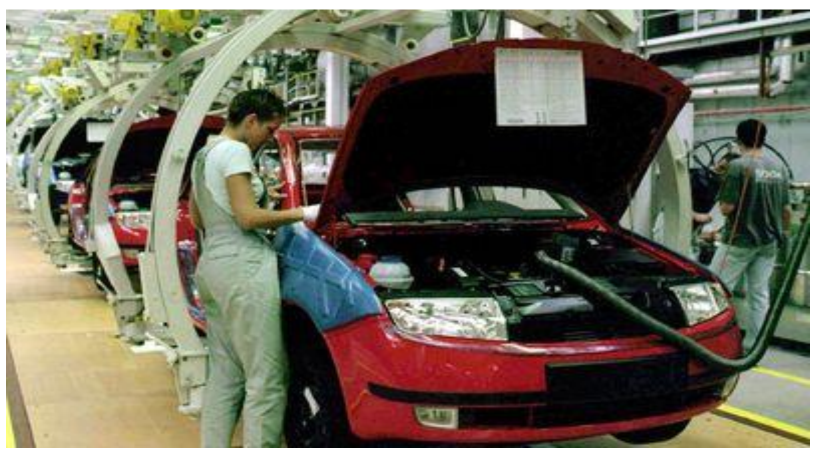
Removing the Deadlock

- Think about how to remove the deadlock. This is a part of the assignment released today.

PIPELINING

Pipelining

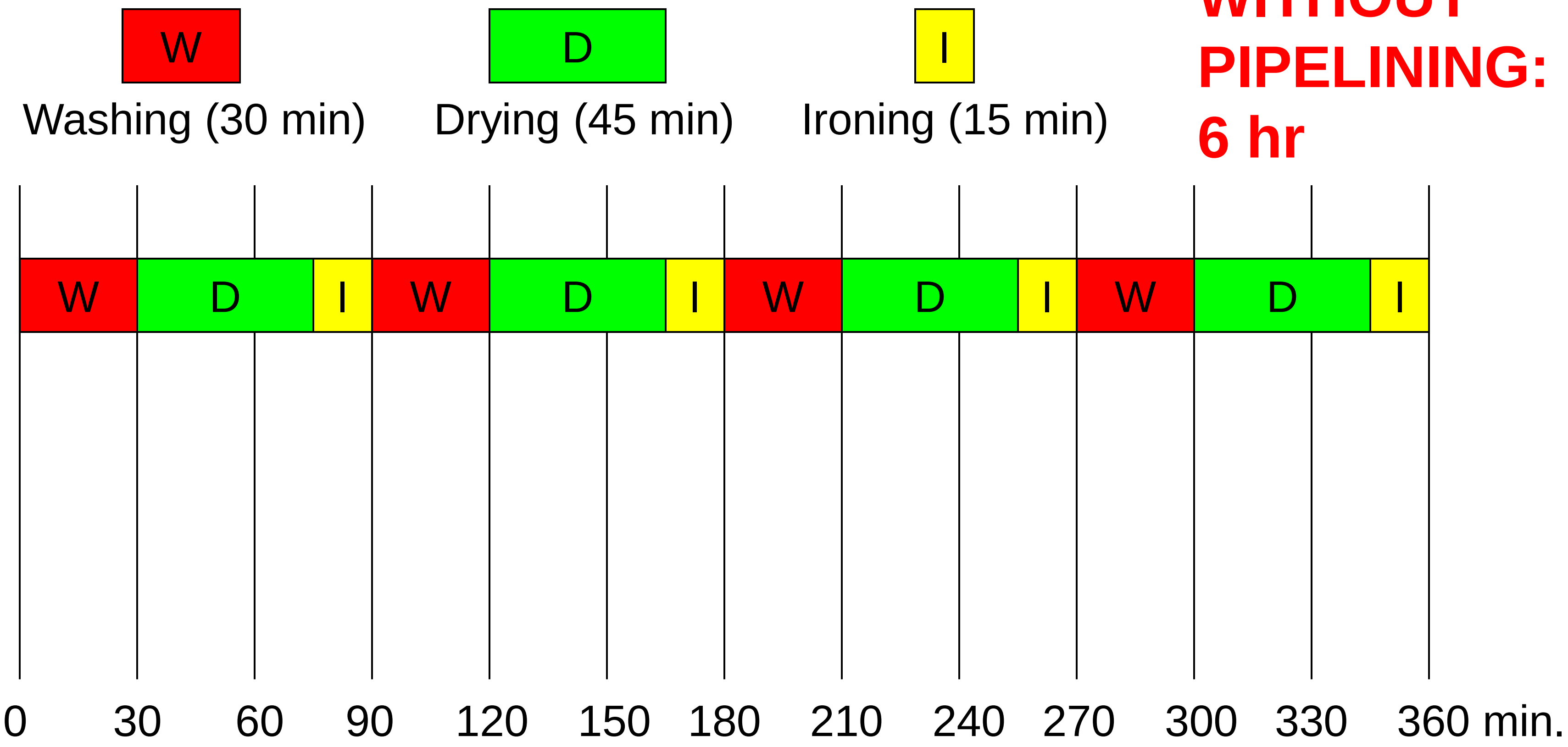
- Pipelining is similar to an assembly line.
  - Instead of completing one computation before starting another, each computation is split into simpler sub-steps, and computations are started as others are in progress.



Laundry Without Pipelining

Washing, Drying and Ironing four loads of laundry.

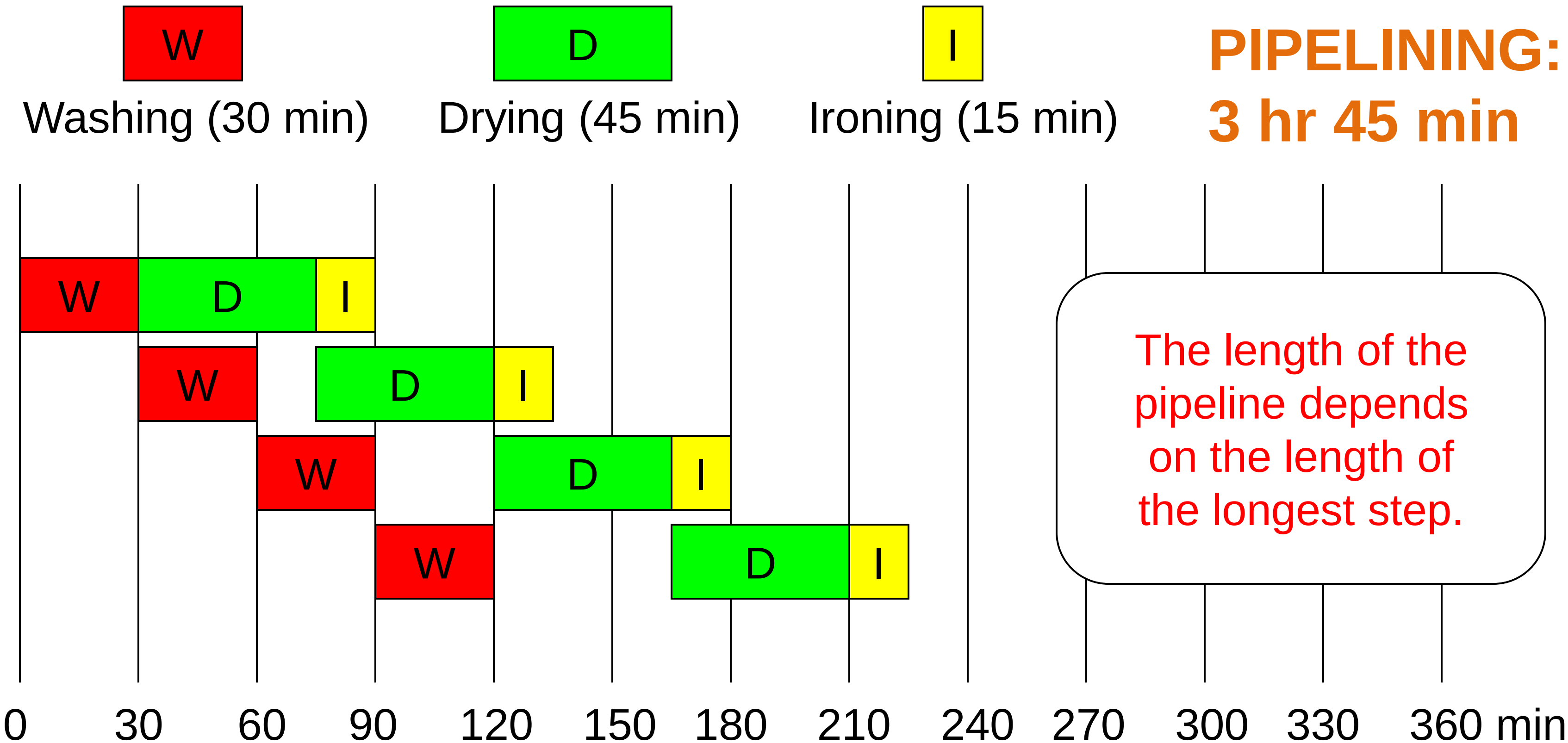
WITHOUT PIPELINING: 6 hr



Laundry With Pipelining

Washing, Drying and Ironing four loads of laundry.

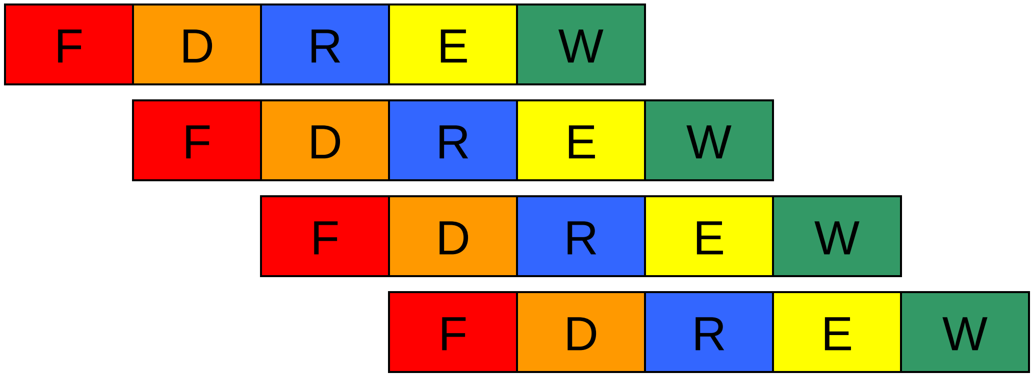
WITH PIPELINING: 3 hr 45 min



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

Pipelining in Computing

- Fetch instruction from memory
- Decode the instruction
- Read data from registers
- Execute the instruction
- Write the result into a register



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 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.

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

	average
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

	average
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

	average
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

	average
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.

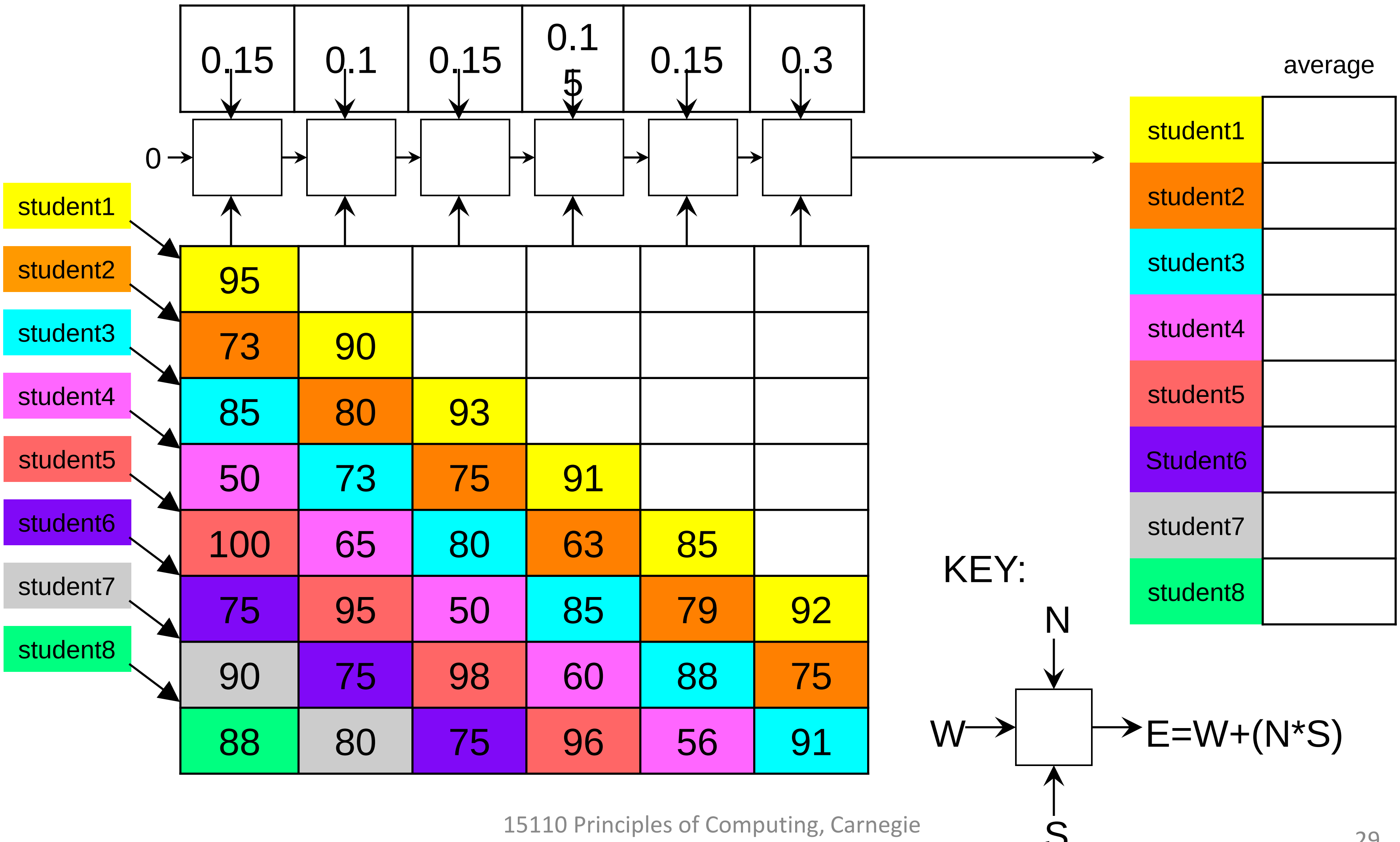
	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

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

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

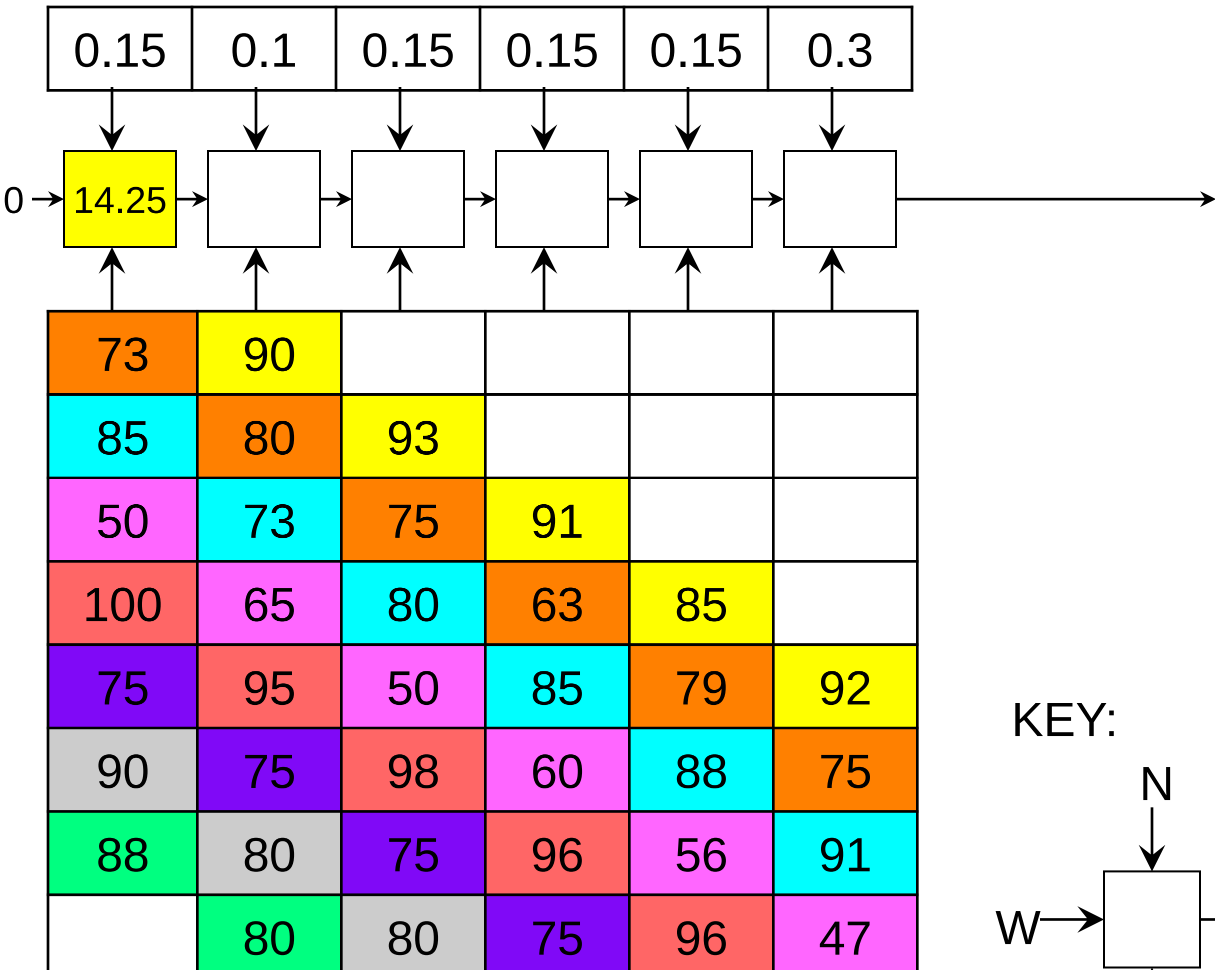
Faster Matrix Multiplication

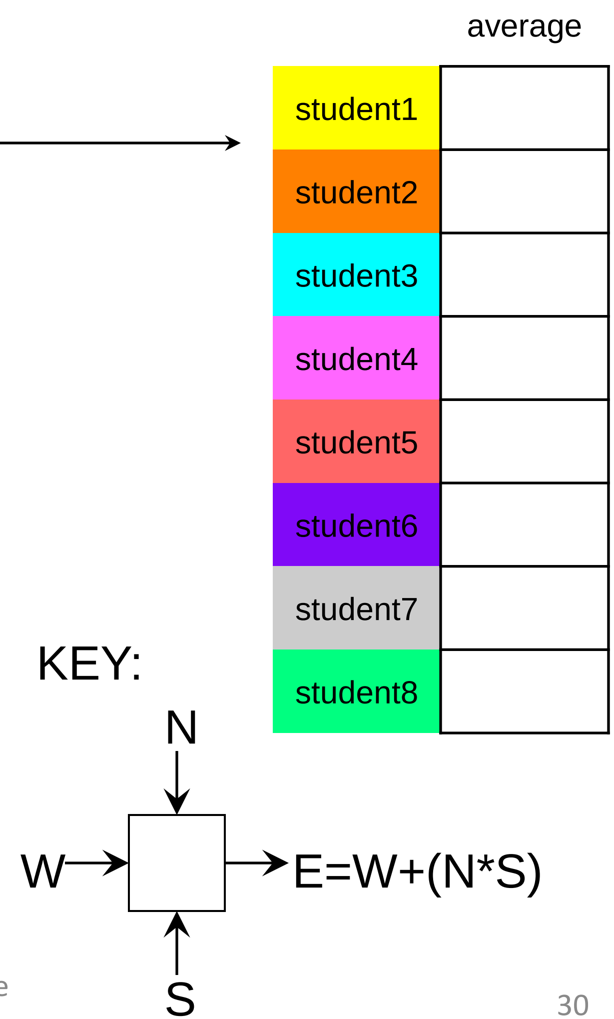
using Pipelining



Faster Matrix Multiplication

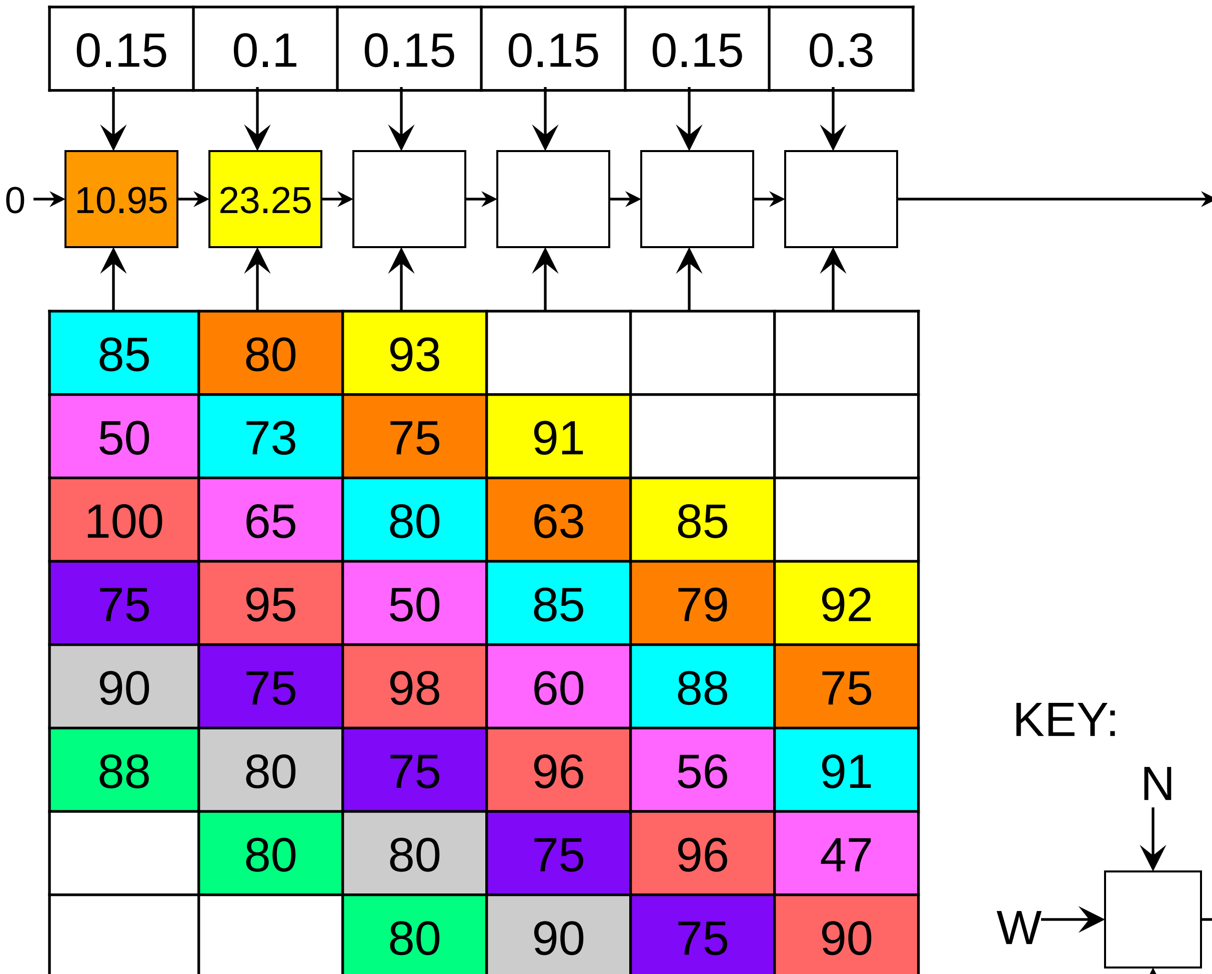
using Pipelining

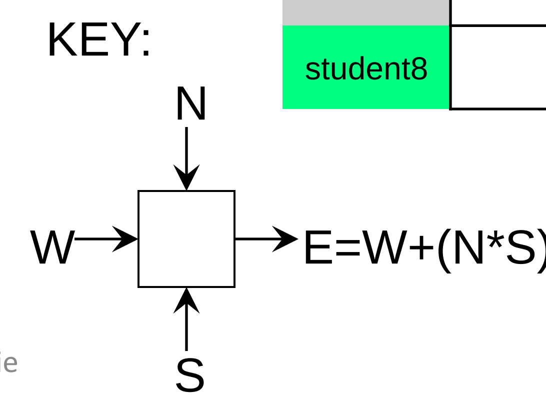




Faster Matrix Multiplication

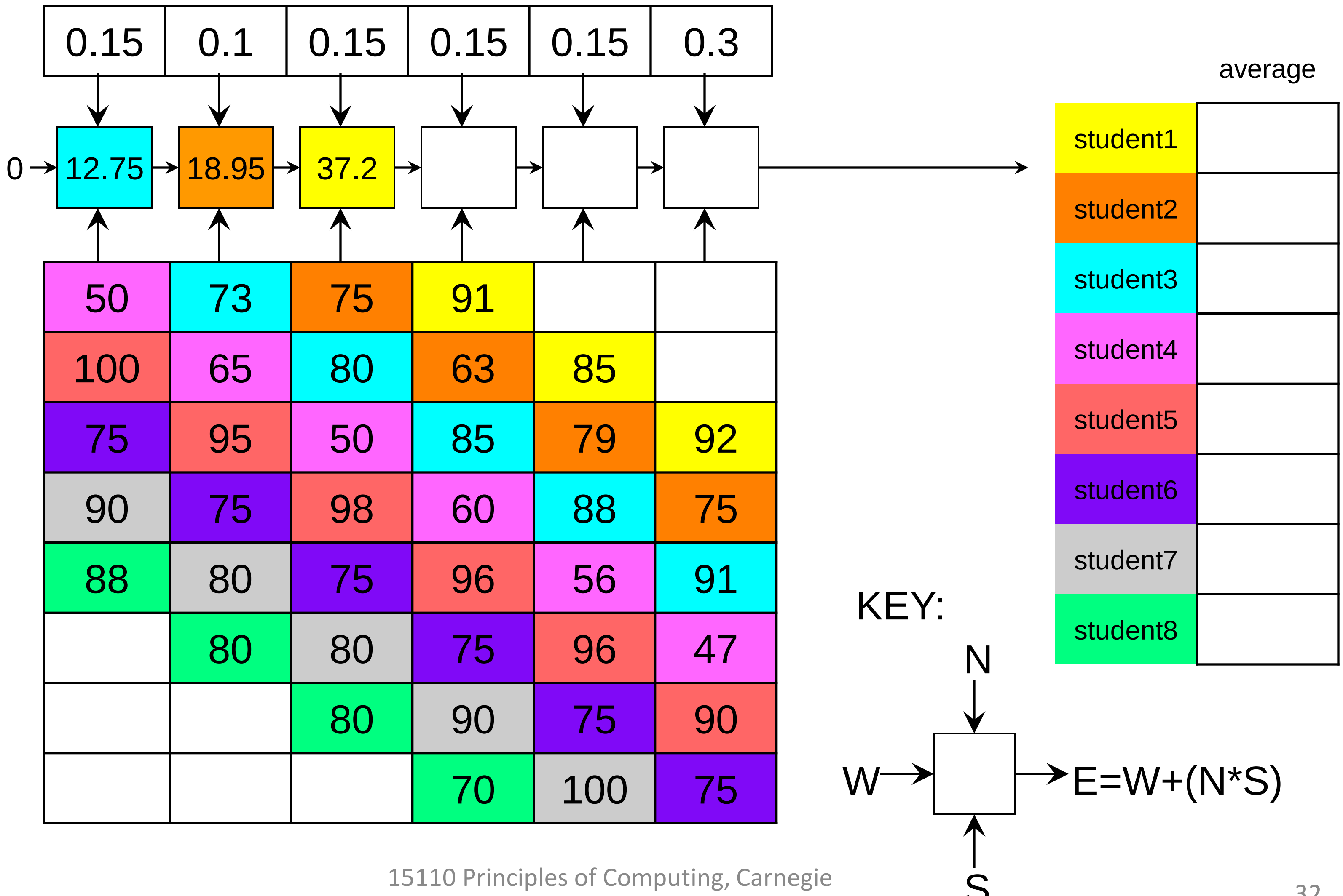
using Pipelining





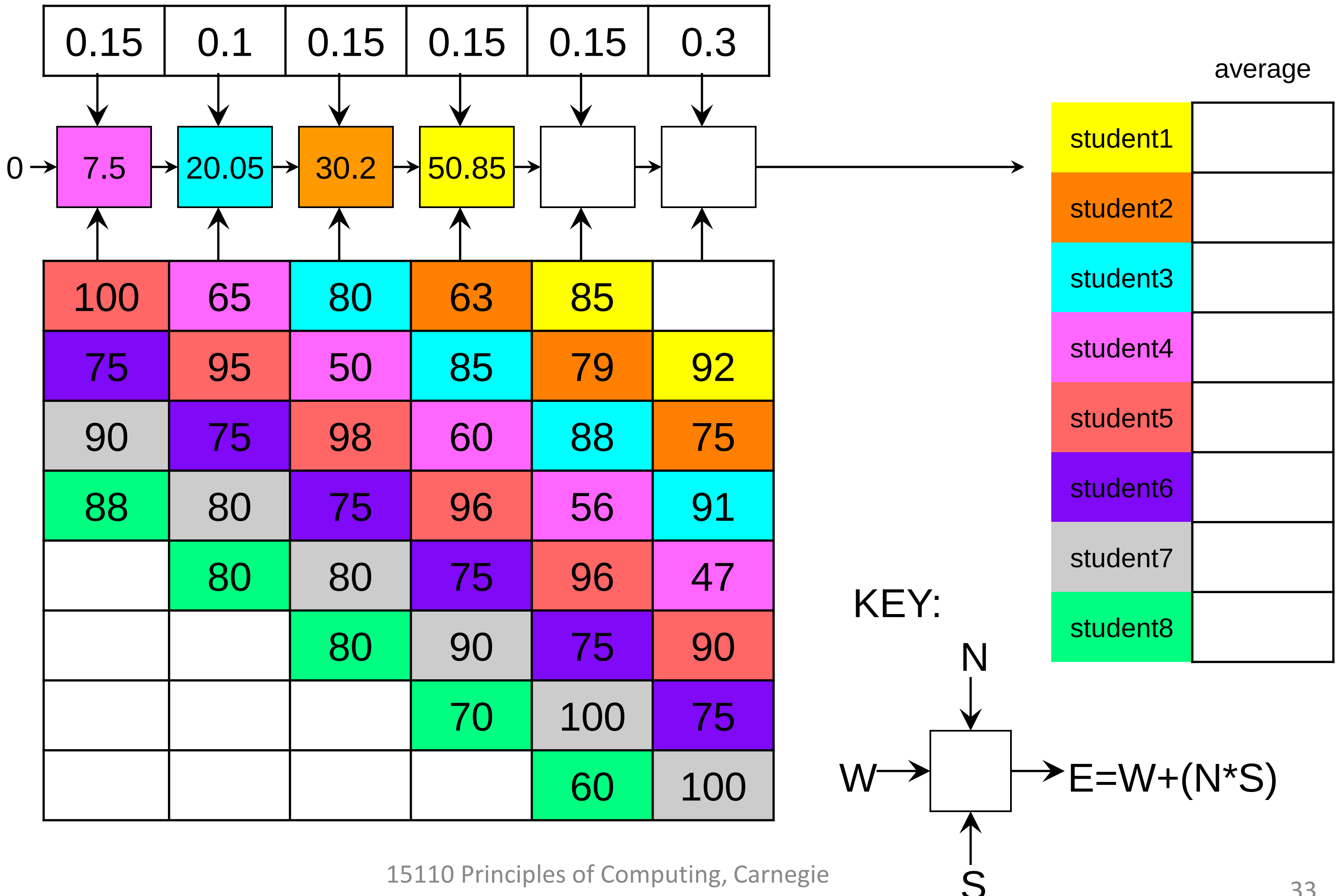
Faster Matrix Multiplication

using Pipelining



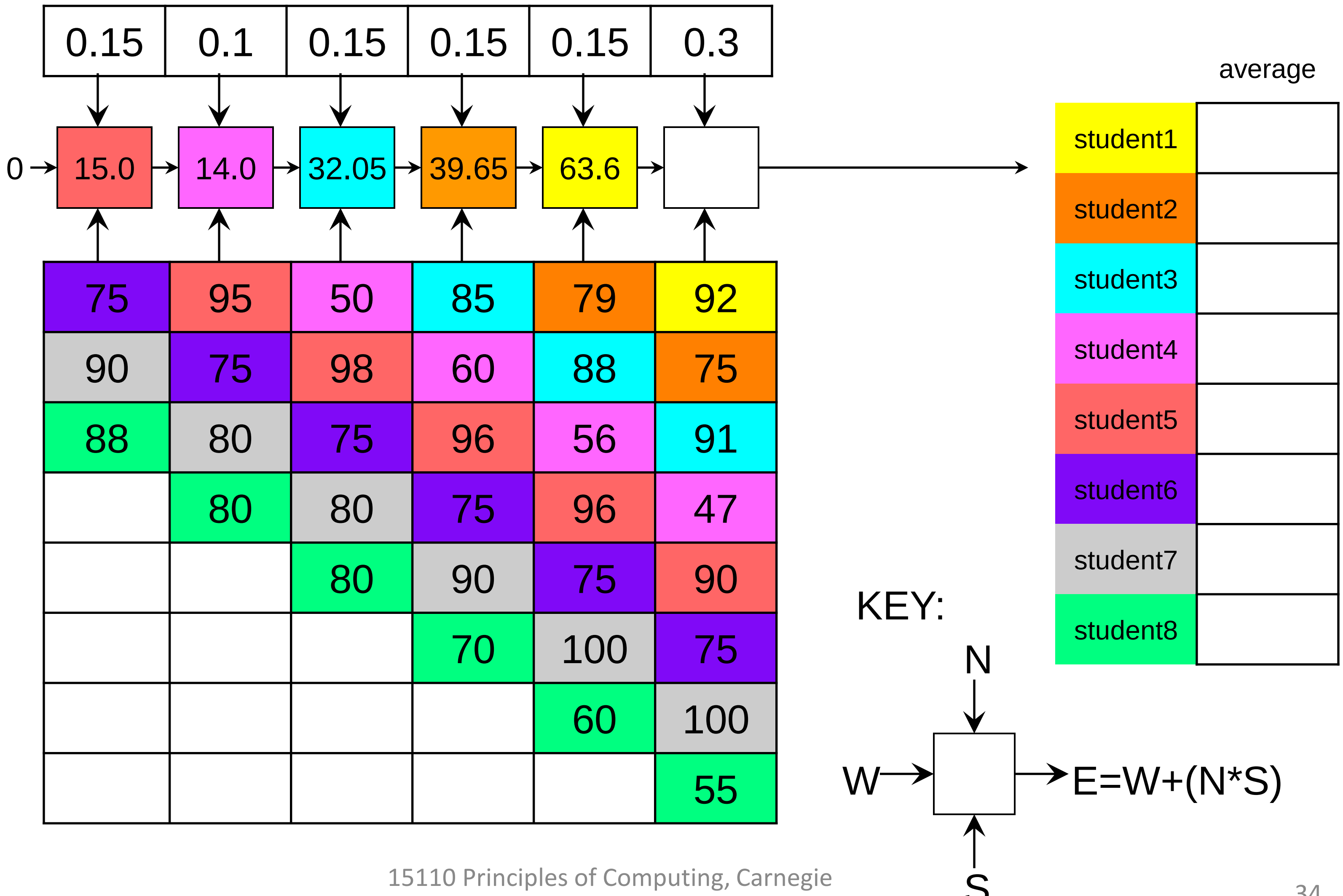
Faster Matrix Multiplication

using Pipelining



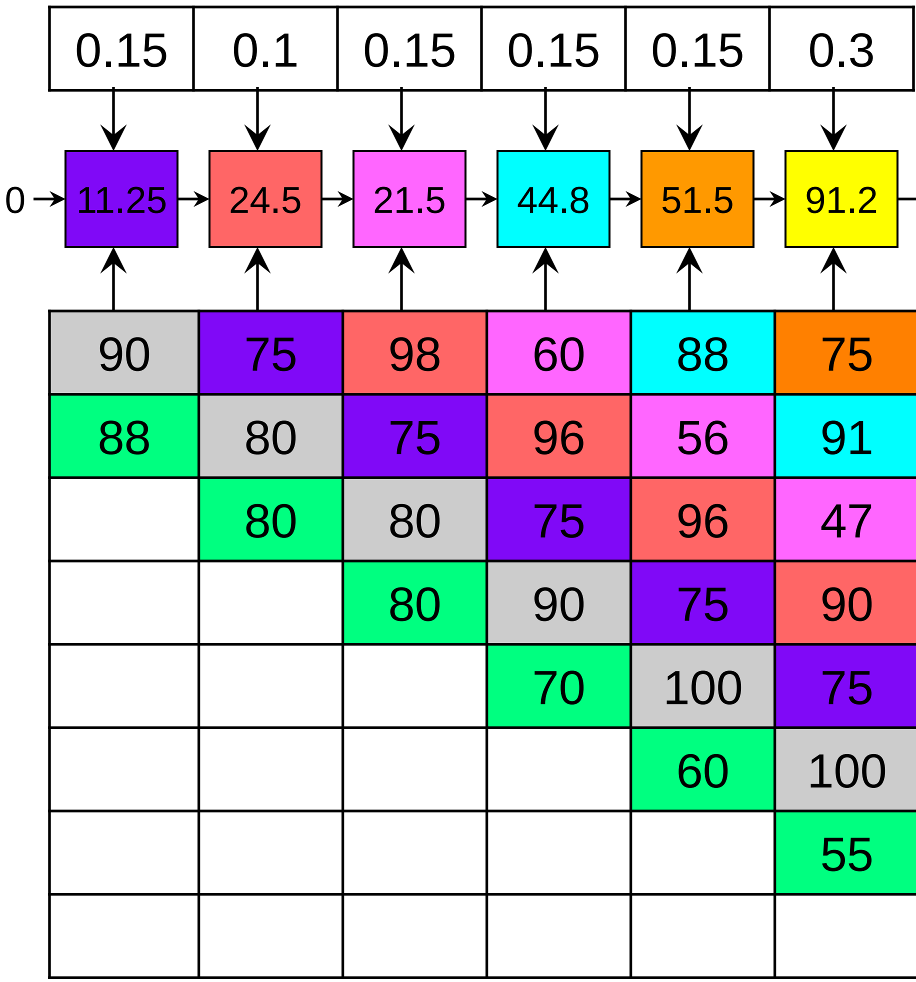
Faster Matrix Multiplication

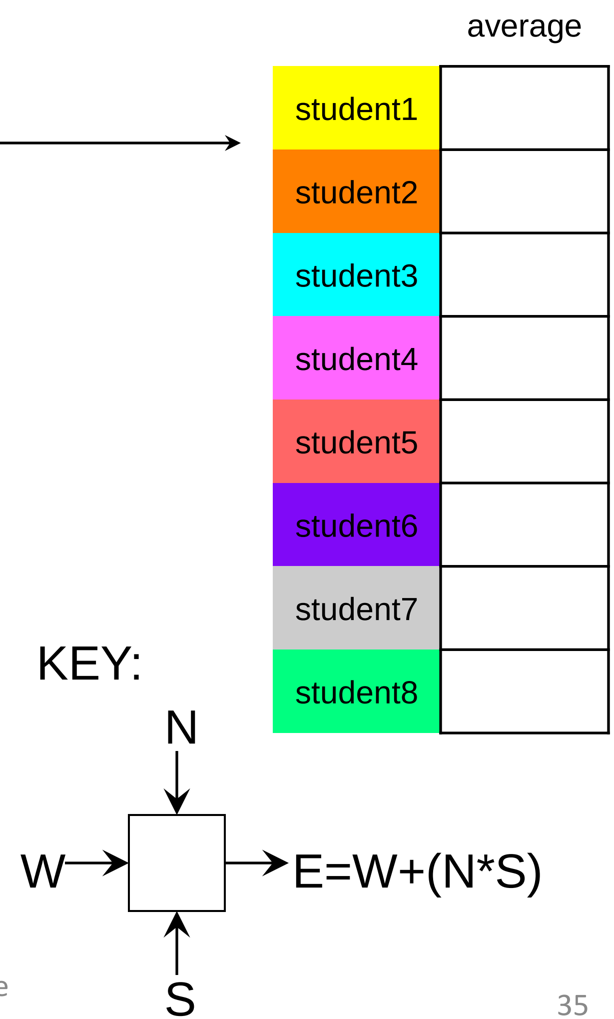
using Pipelining



Faster Matrix Multiplication

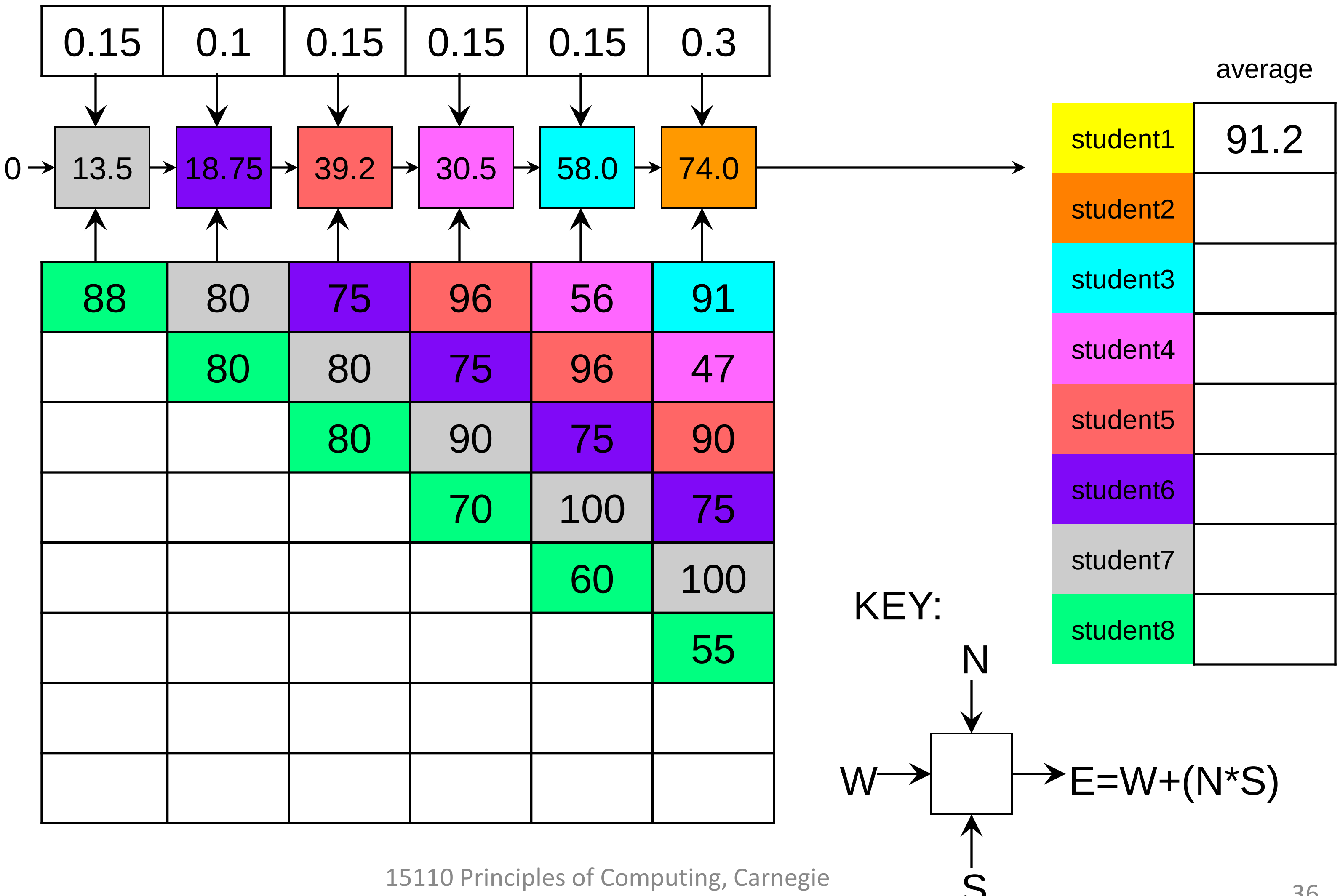
using Pipelining





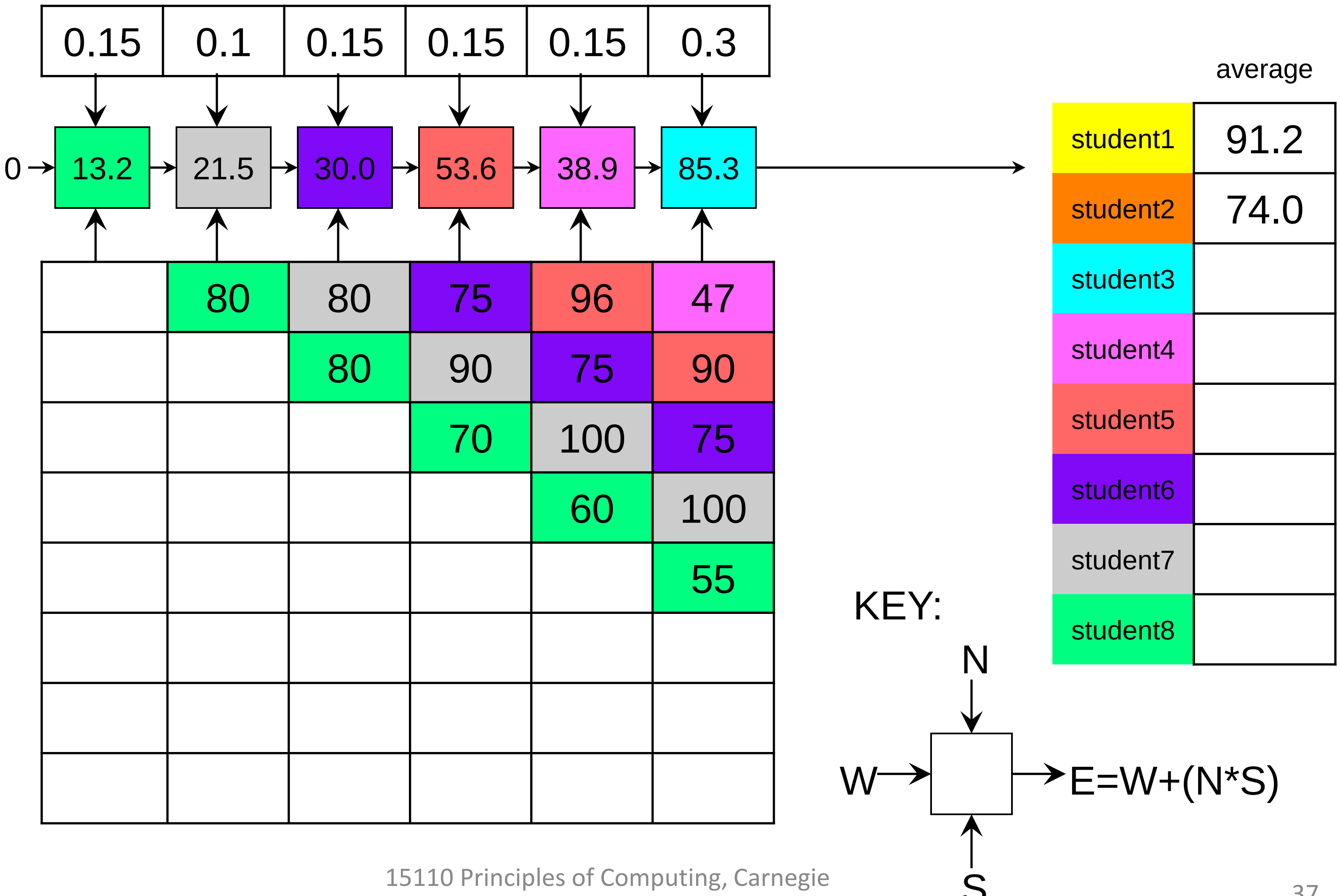
Faster Matrix Multiplication

using Pipelining



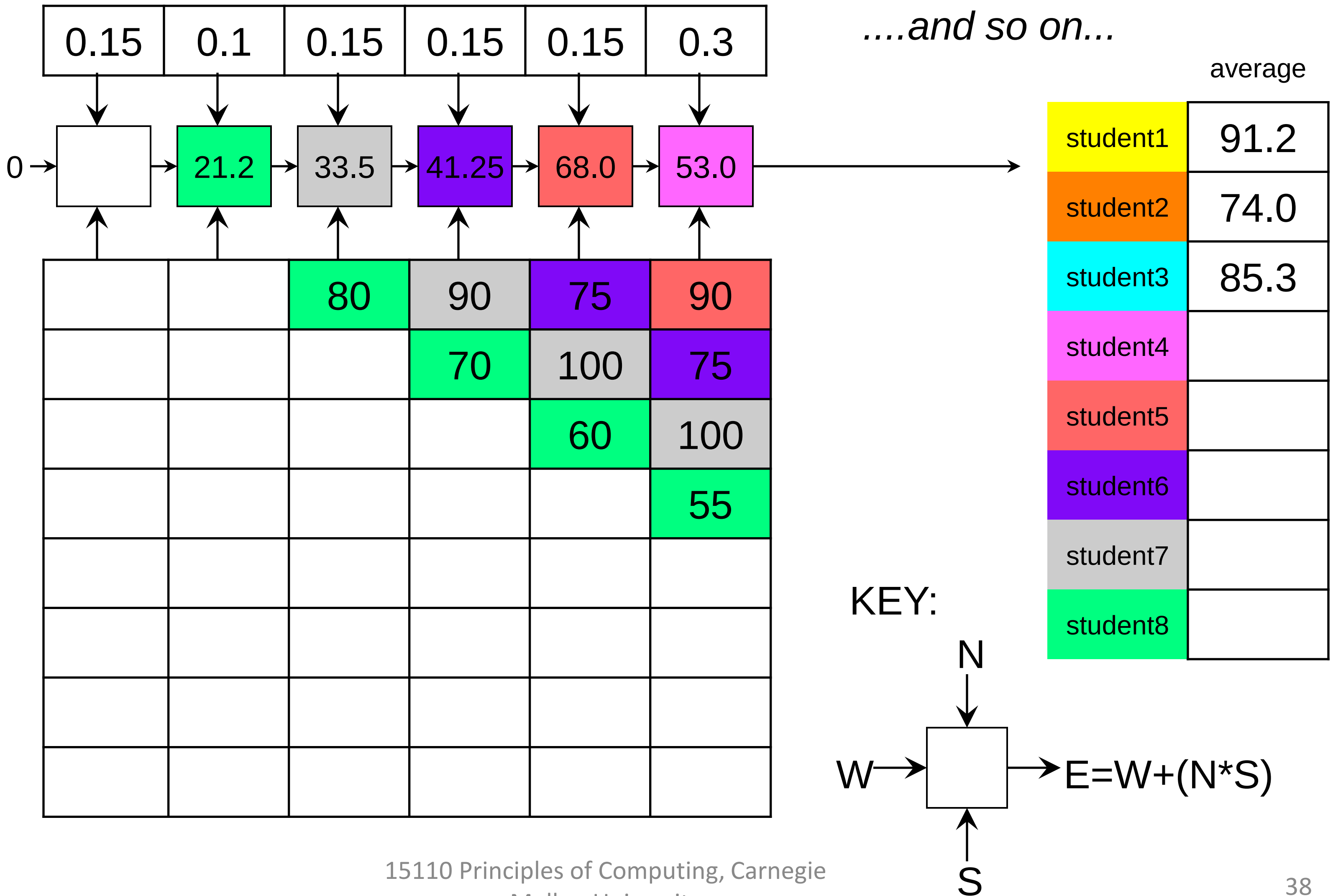
Faster Matrix Multiplication

using Pipelining



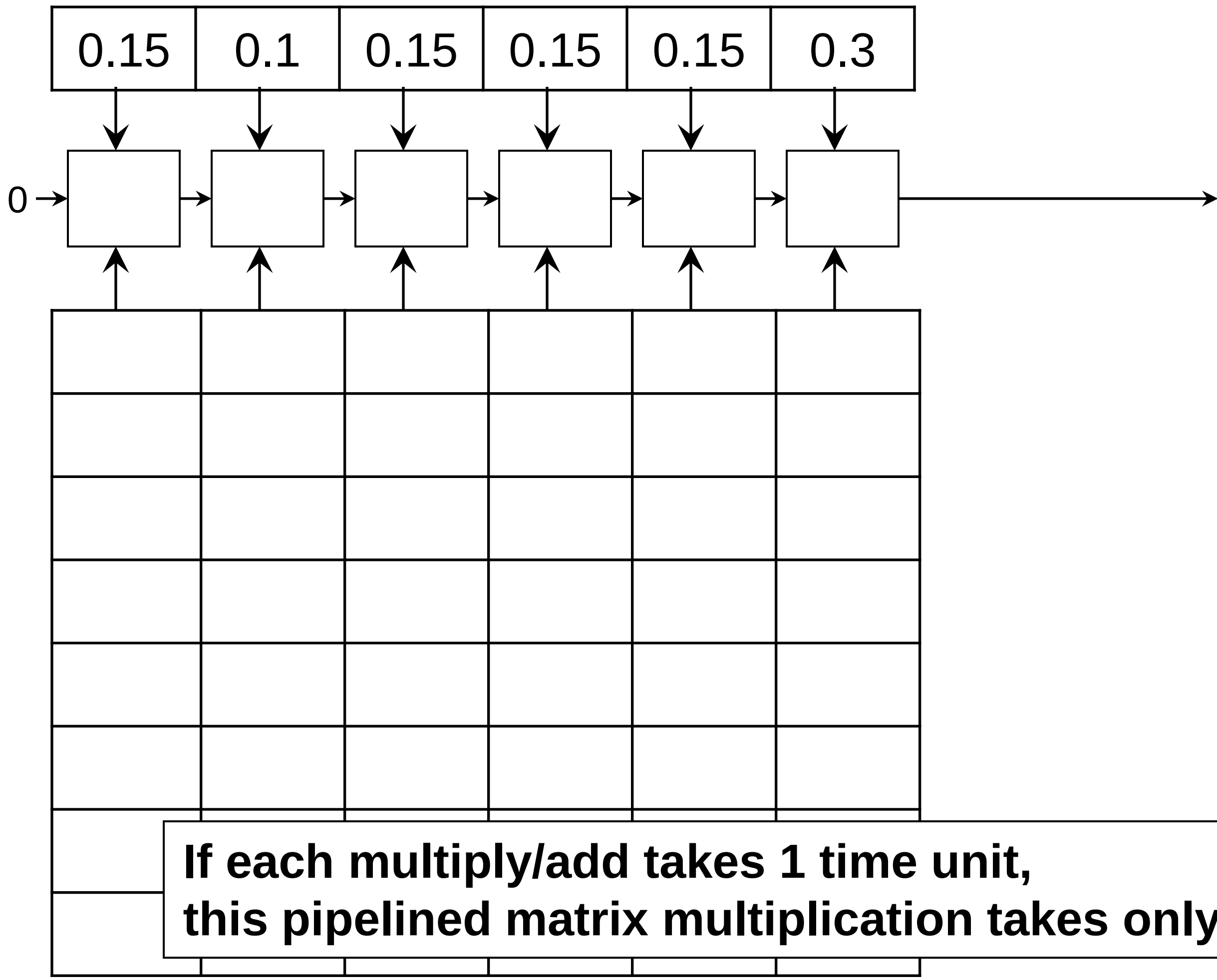
Faster Matrix Multiplication

using Pipelining



Faster Matrix Multiplication

using Pipelining



	average
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.

Distributed Systems

- A distributed system is an application that consists of processes that
  - execute on multiple computers connected through a network, and
  - cooperate to accomplish a task.
- Advantages
  - Reconfigurable: add or rearrange new parts
  - Geographically distributed: Low communication delays for remote users
  - Scalable: can add more processors as demand increases

Examples of Failures

- Failures happen all the time and distributed systems must cope with it
  - permanent network failures
  - dropped messages
  - between sender and receiver
  - an individual computer breaks
  - a process crashes or goes into an infinite loop

Can We Fix These Failures?

- Replication/Redundancy
- Using transaction logs to do recovery
  - Transaction log is a history of actions executed by a database management system to guarantee backup over crashes or hardware failures.

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.
- Processes may interact and coordinate in complex ways. Care must be taken when they share common resources, to deal with race conditions, to avoid deadlocks etc.
- We did not introduce any new programming construct in this unit.