# Concurrency: Pipelining and Distributed systems

Jeffery von Ronne[1]

Department of Computer Science
Carnegie Mellon University

April 4, 2012

[1] Pipelining material is from Tom Cortina.



## Outline

# What is pipelining?

pipelining is like an assembly line

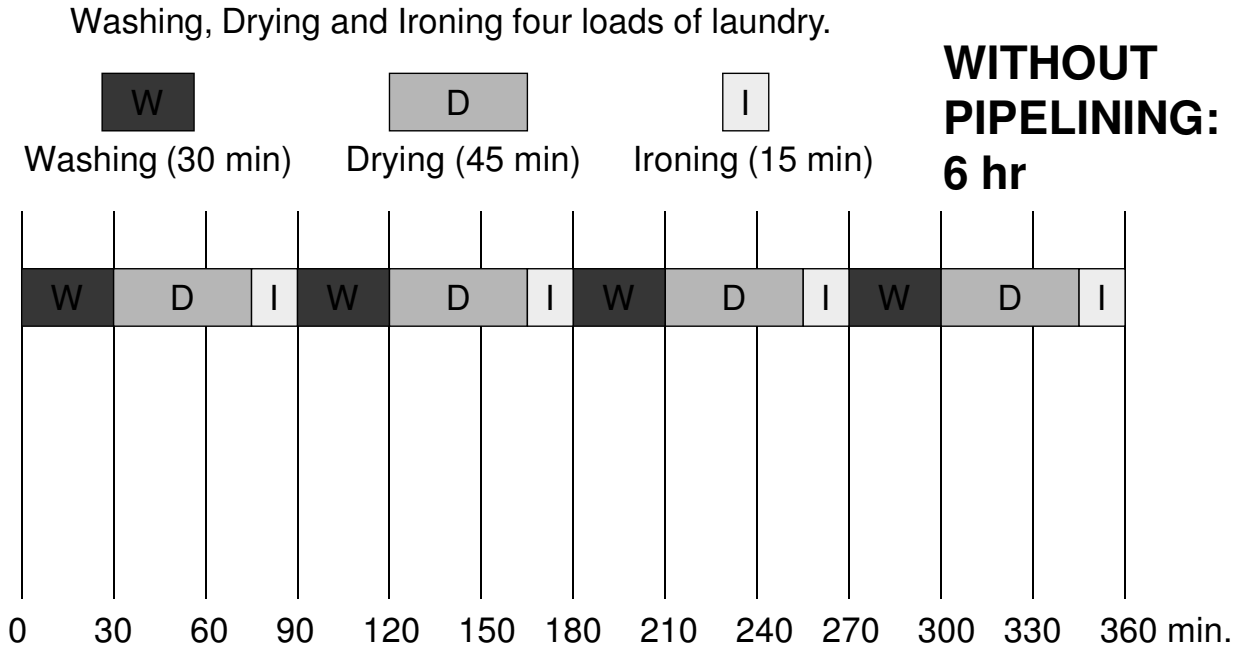
- sequence of similar tasks
- before one task finishes, start the next
- each task split into simpler sub-steps
  - once one sub-step is completed
    - begin that sub-step on the next task
  - separate resource for each sub-step

1943 Vultee Valiant assembly line
[AFHRA photo 080129-f-3927s-303]

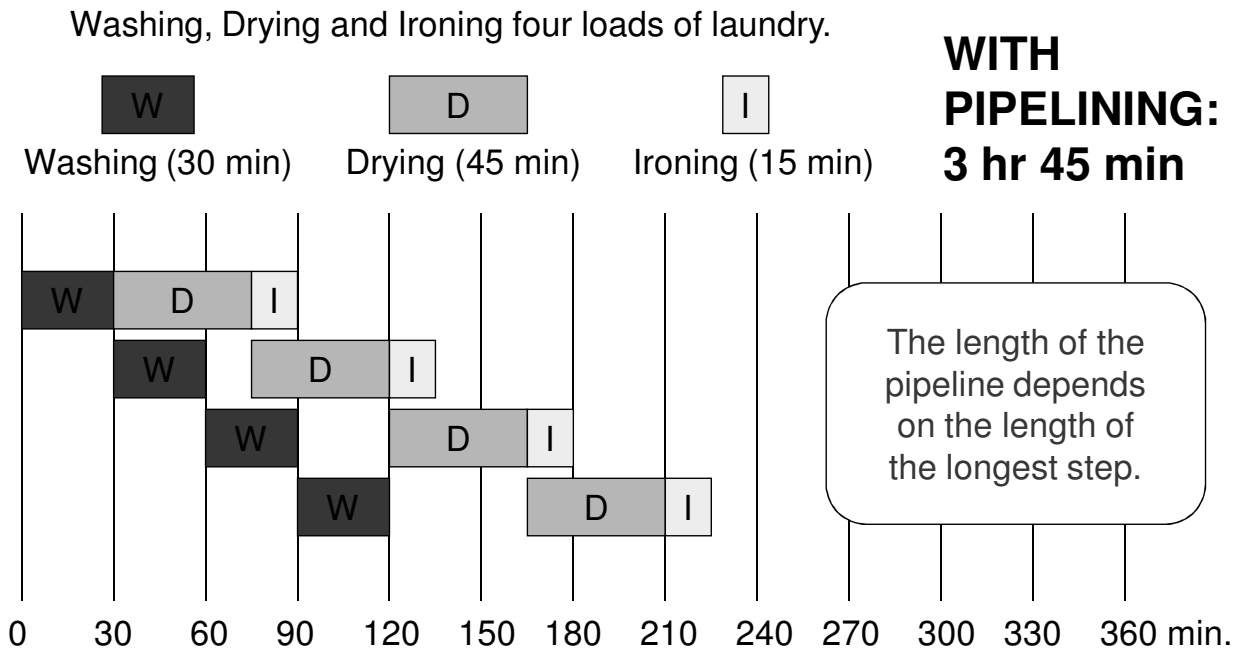




# Laundry Without Pipelining



# Laundry With Pipelining

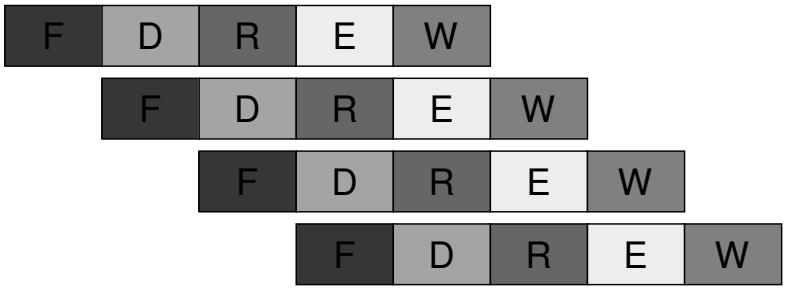







# Processor Pipeline Stages

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



# Processor Pipeline Stall

```
ADD R3, R2, R1
ADD R5, R4, R3
ADD R8, R7, R6
ADD R11, R10, R9
```

"Add the contents of R1 and R2 and store the results in R3."

This instruction depends on the result of the previous instruction. (This will hold up the pipeline.)

```
ADD R3, R2, R1
ADD R8, R7, R6
ADD R11, R10, R9
ADD R5, R4, R3
```

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





# Processor Pipeline with Conditional Branches

```
A: ADD R3, R2, R1
   SUB R6, R5, R4
   BEQ R6, R3, A
   MOV R2, R1
```

"Branch to label A if R3 = R6."

The BEQ instruction will stall in the pipeline since the final values of R3 and R6 are not known yet.

Possible solutions:

1. Assume the branch occurs. If we find later that R3 is not equal to R6, clear the pipeline and begin computing with the MOV instruction.
2. Start decoding the ADD and MOV instructions. When we know if R3 is equal to R6 or not, send the appropriate instructions into the pipeline for completion.

# Matrix Multiplication without Pipelining

**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 without Pipelining

**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 without Pipelining

**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 without Pipelining

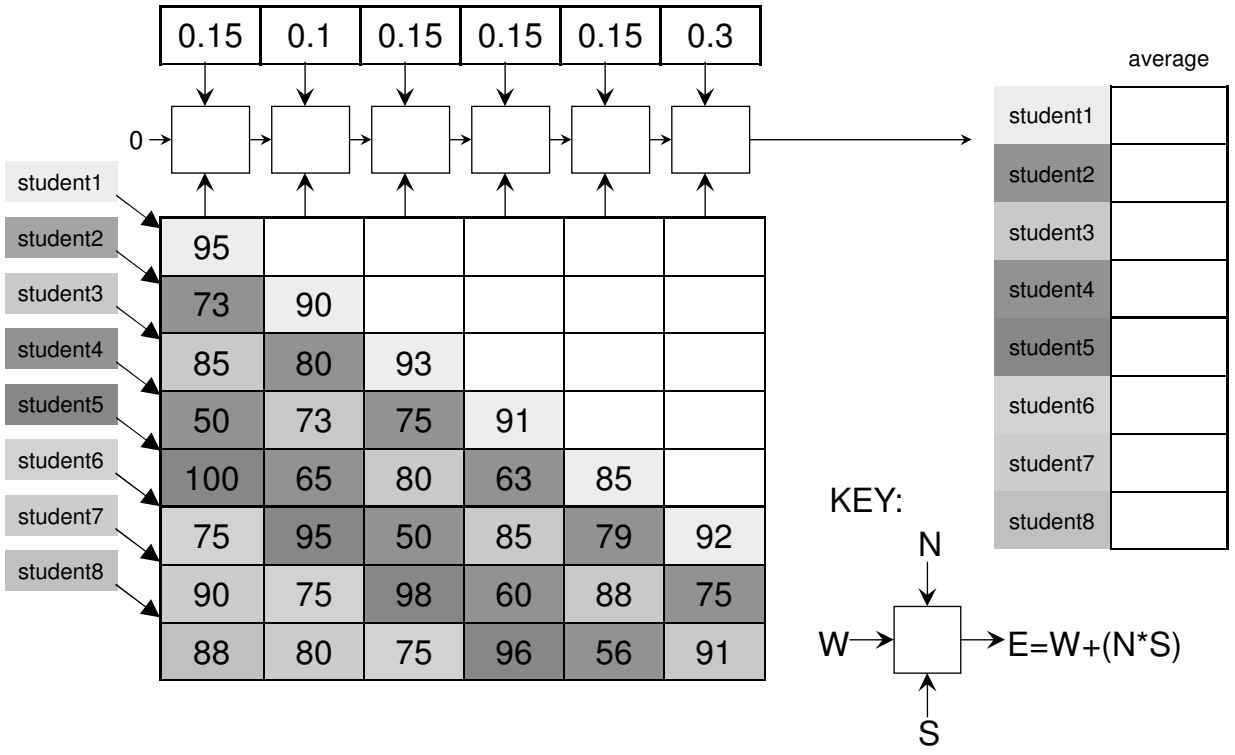
**If each multiply/add takes 1 time unit,**
**this non-pipelined matrix multiplication takes 48 time units.**

| | 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 |

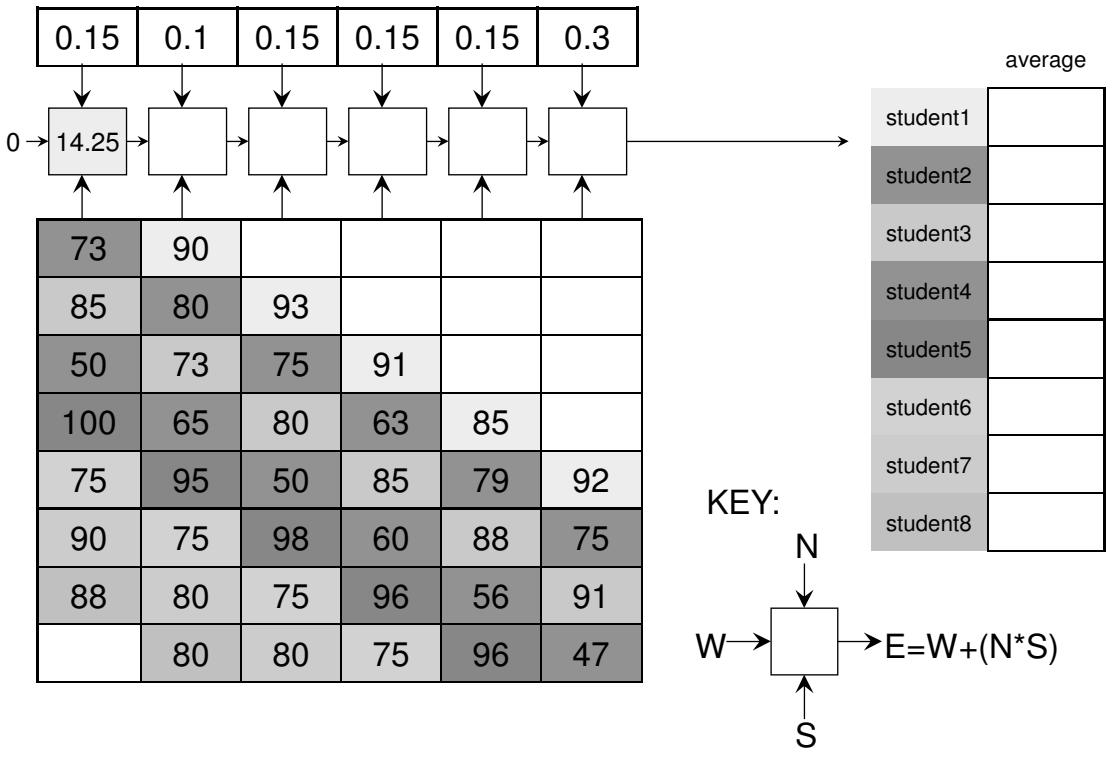
# Matrix Multiplication with Pipelining



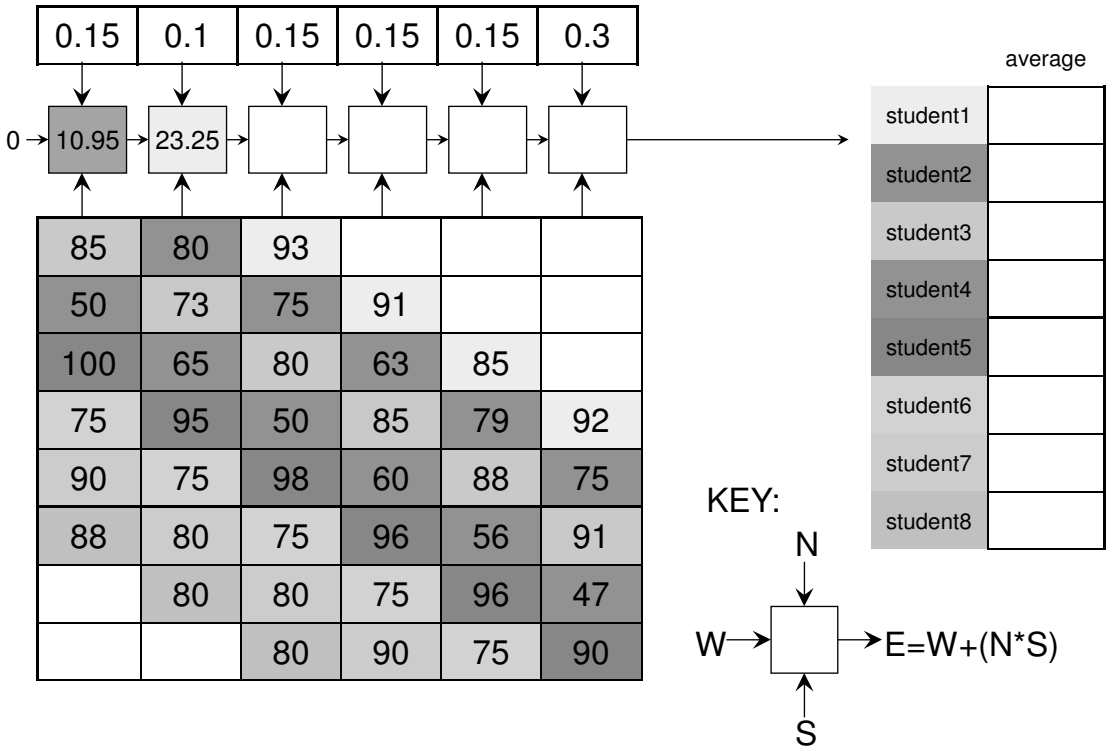





# Matrix Multiplication with Pipelining



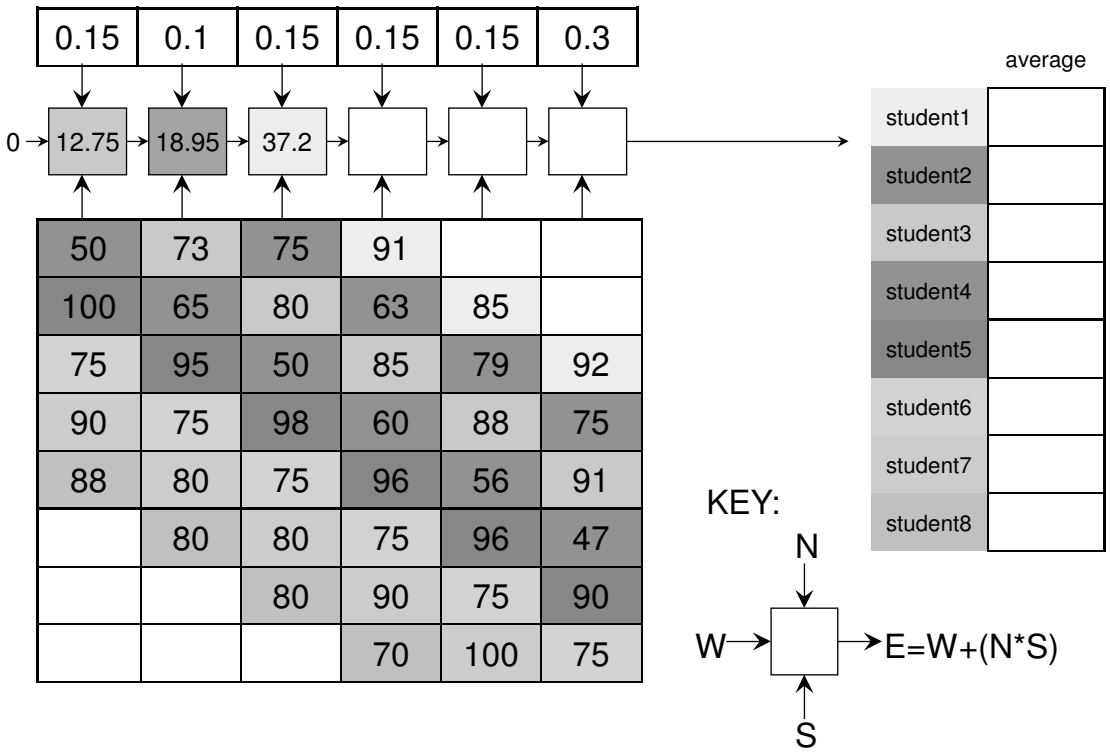
# Matrix Multiplication with Pipelining



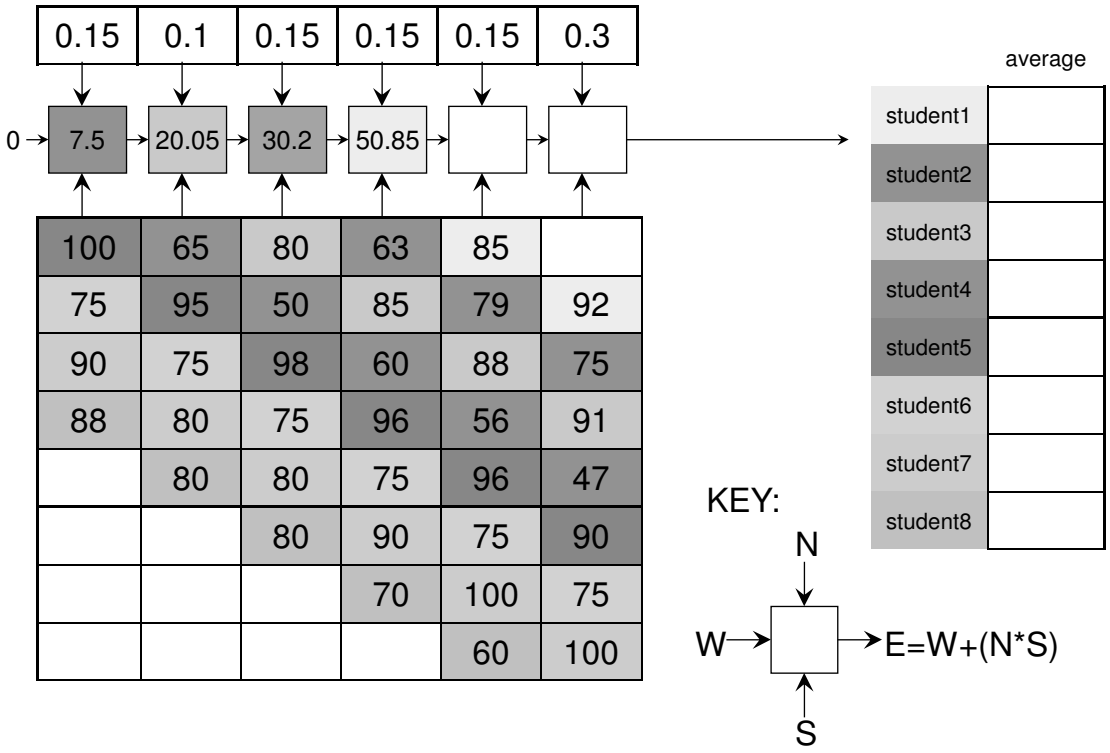





# Matrix Multiplication with Pipelining



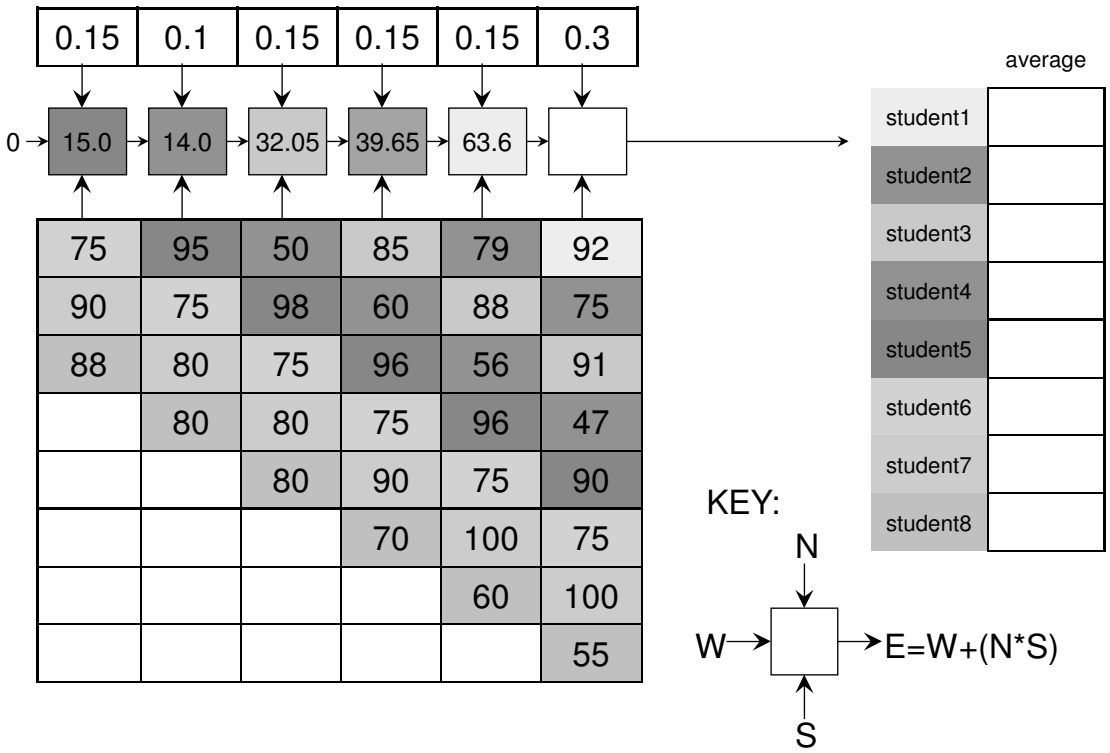
# Matrix Multiplication with Pipelining



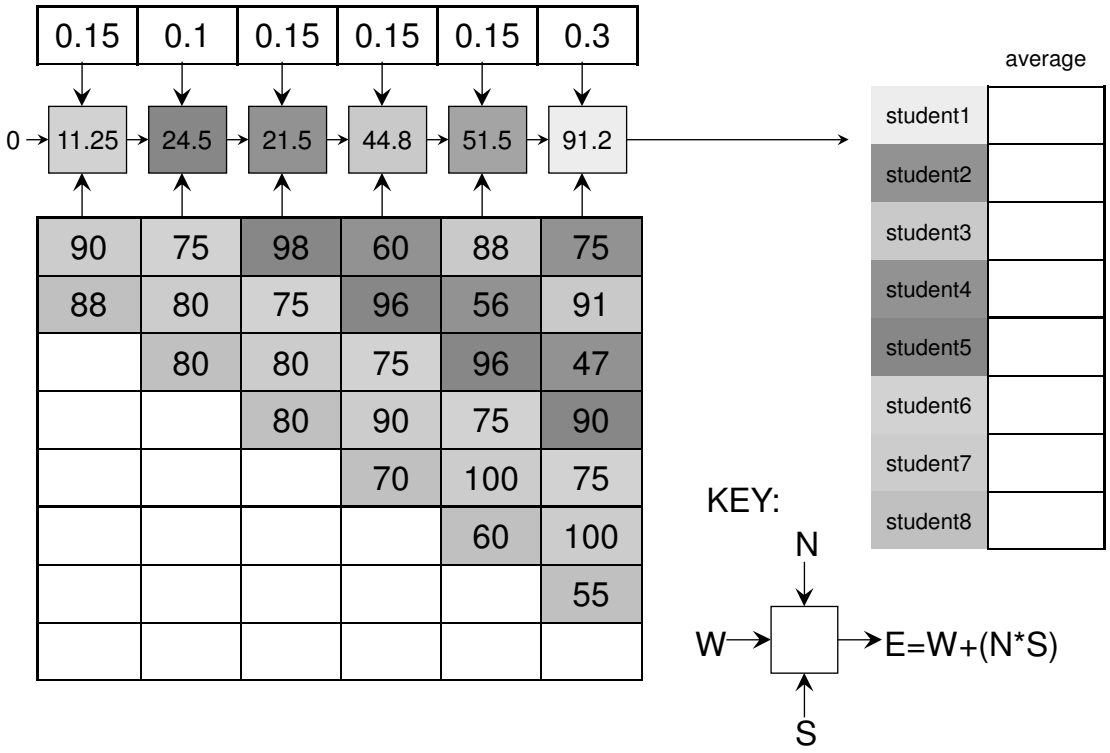





# Matrix Multiplication with Pipelining



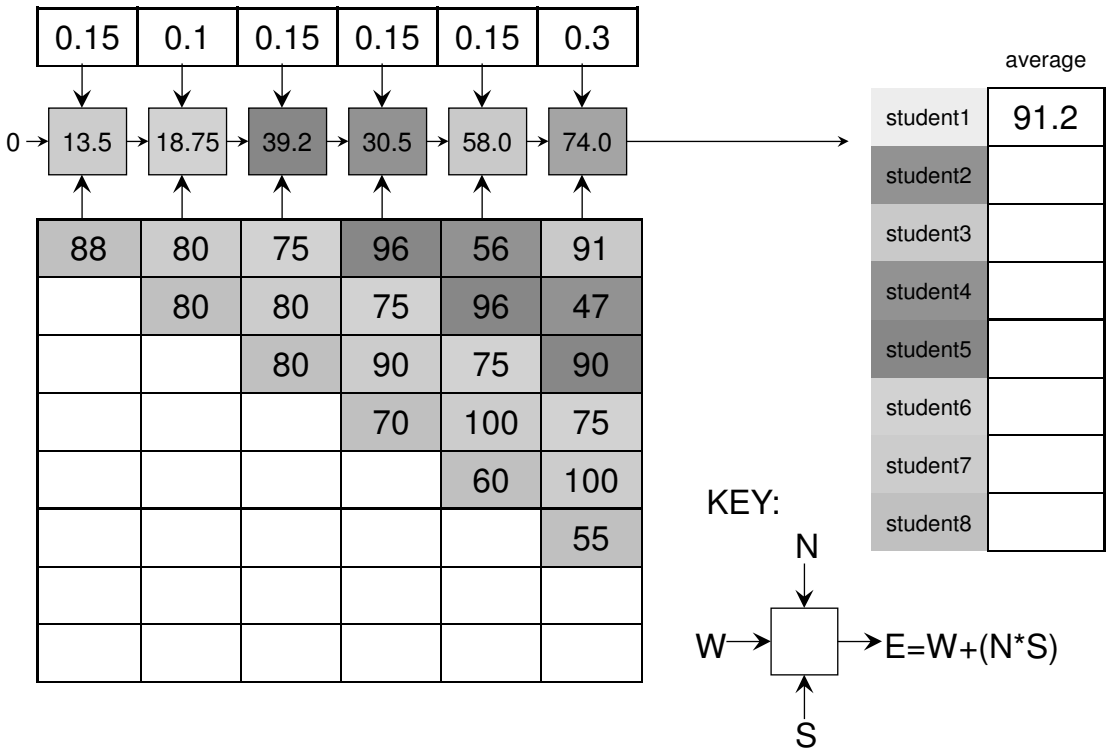
# Matrix Multiplication with Pipelining



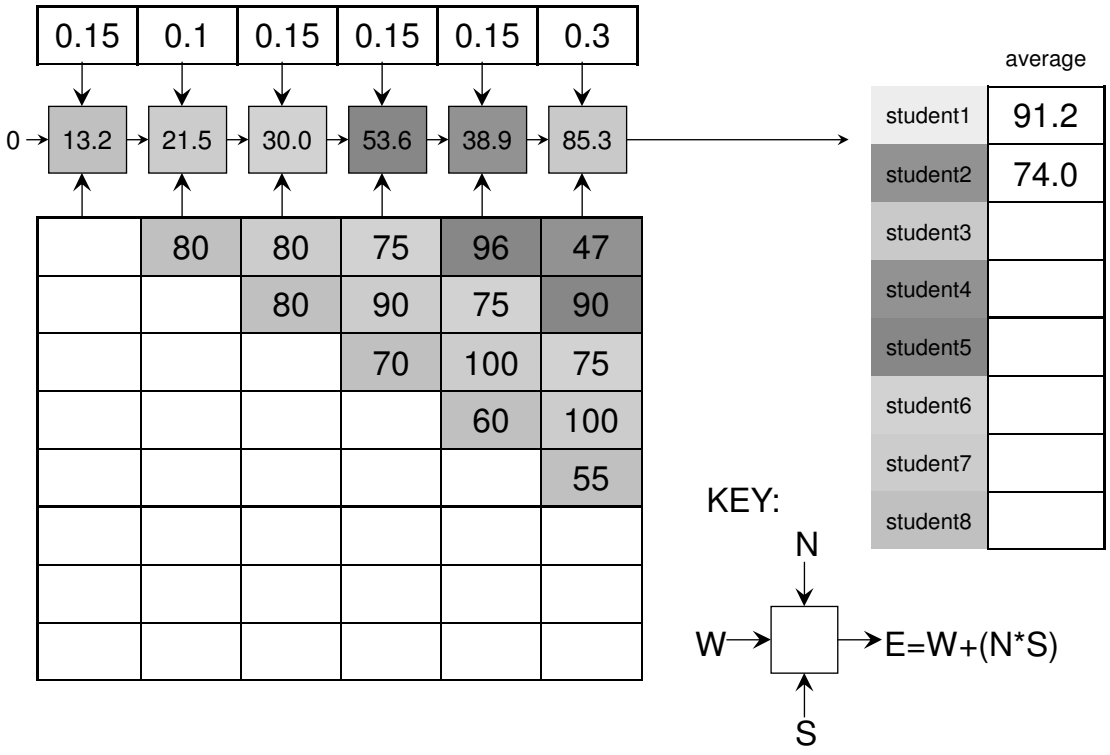





# Matrix Multiplication with Pipelining



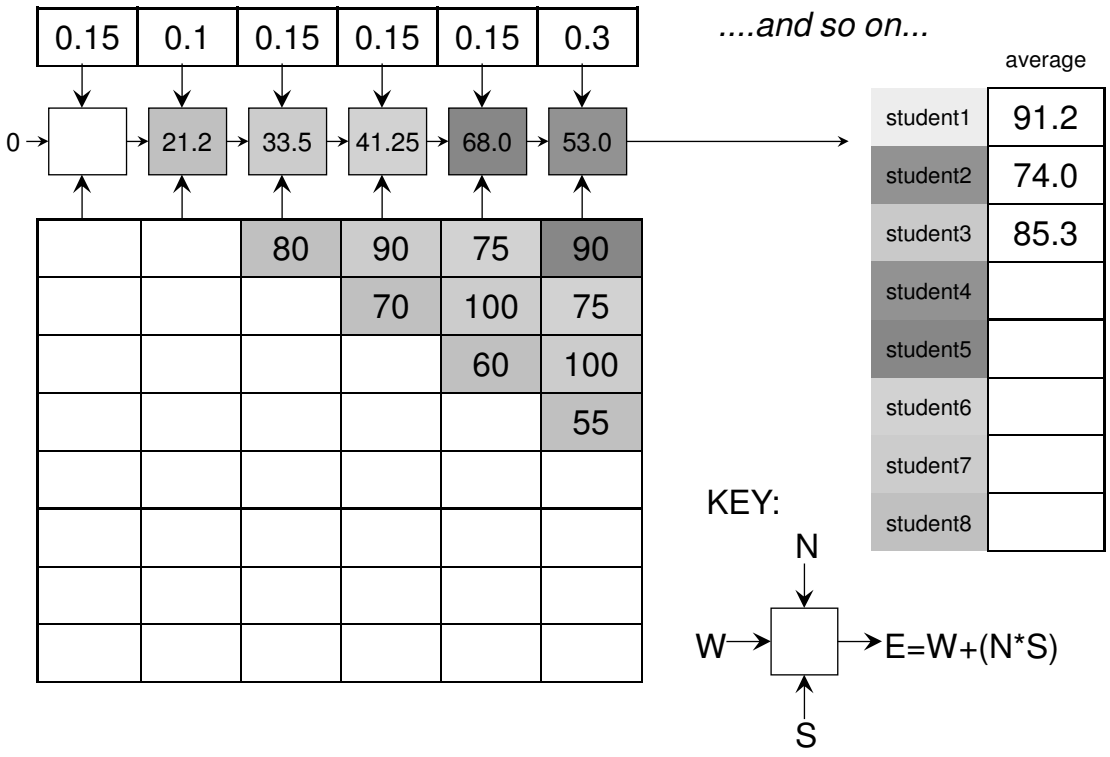
# Matrix Multiplication with Pipelining



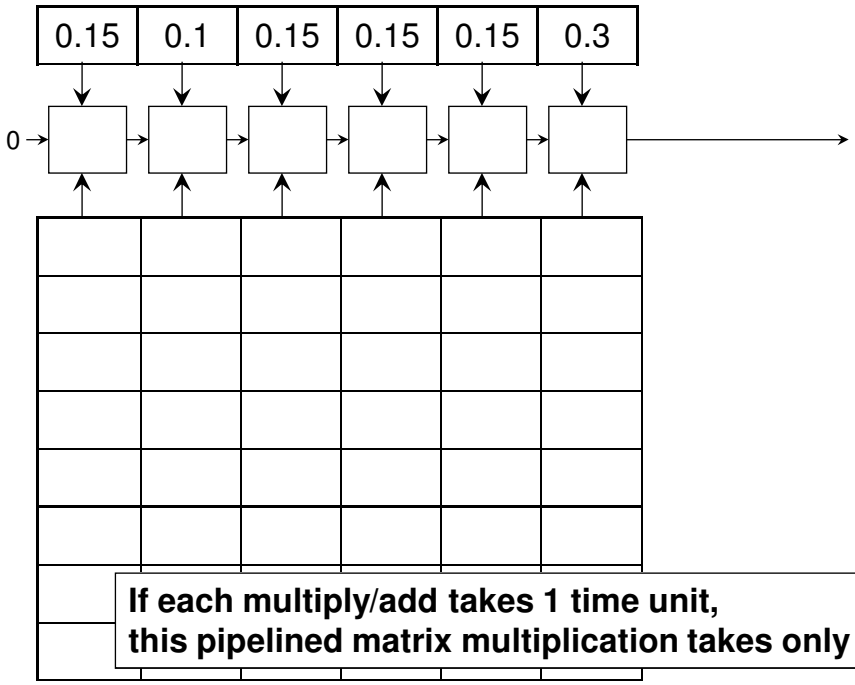





# Matrix Multiplication with Pipelining



# Matrix Multiplication with 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.**





# The 'What' and 'Why' of Distributed Systems

*A distributed system is an application that consists of processes that*

a) *execute on multiple computers connected through a network, and*
b) *cooperate to accomplish a task.*

## Advantages

- open
  - add new parts and interact with remote users
- scalable
  - system can be altered to accommodate changes in numbers of users, resources, computer systems

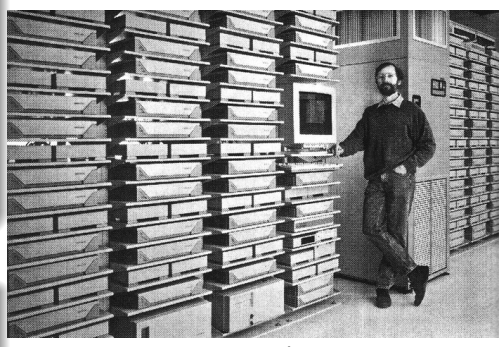
# Render Farms

## Rendering

- flatten a 3-d space to a 2-d
- lighting (raytracing)
- potential concurrency
  - frames
  - pixels within a frame

## Toy Story (1995)

- about 4 hours for each frame
- 80 SPARCstation 20 systems
- 1 SPARCserver 1000 system



Toy Story Render Farm





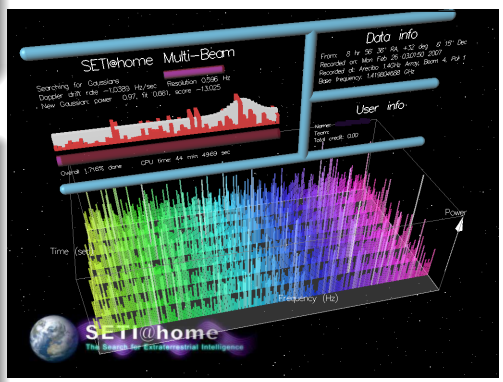
# SETI@home

## Search for Extraterrestrial Intelligence

- telescopes pointed at the sky
- scan for “artificial signals”

## SETI@Home

- splits data into work units
  - frequency: 10 kHZ
  - time: 107s
- SETI@home screensaver
  1. receives work unit
  2. processes work unit
  3. returns work unit



# Challenge of Distributed Computing: Reliability in Context of Failure

> *Failure is the defining difference between distributed and local programming, so you have to design distributed systems with the expectation of failure. Imagine asking people, "If the probability of something happening is one in* $10^{13}$*, how often would it happen?" Common sense would be to answer, "Never." That is an infinitely large number in human terms. But if you ask a physicist, she would say, "All the time. In a cubic foot of air, those things happen all the time." When you design distributed systems, you have to say, "Failure happens all the time." So when you design, you design for failure. It is your number one concern.*
>
> — Ken Arnold





# Examples of Failure

- permanent network failures
- dropped messages between sender and receiver
- an individual computer breaks
- a process crashes or goes into an infinite loop

# Summary: Pipelining and Distributed Systems

## pipelining

- assembly line: different sub-steps run concurrently
- Processor Pipelining:
  - runs a sequence of instructions faster
  - splits each into, e.g., fetch, decode, read, execute, write
  - separate hardware for each pipeline stage

## Distributed Systems

- multiple processes distributed across multiple machines
- Examples:
  - render farms
  - SETI@home
  - google