

# HPC & Parallel Programming

## Introduction

**Kun Suo**

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

# Outline

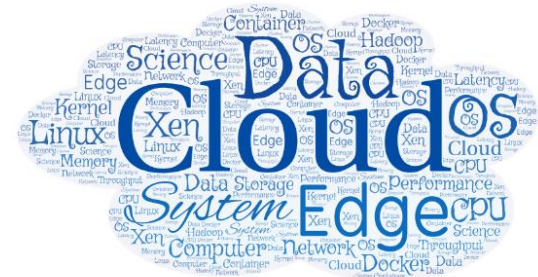
---

- Why study HPC & parallel programming?
- What to learn?
- Course structure
- Course policy
- An example of HPC & parallel programming



# Self Introduction

- Kun Suo, Ph.D.
  - Homepage, <https://kevinsuo.github.io/>
- Research interests:
  - Cloud computing and virtualization;
  - Parallel and Distributed Computation, containers and kubernetes;
  - Software defined network (SDN) and network function virtualization (NFV)
  - Big data systems and machine learning systems
- Projects you may be interested in:
  - Several projects in Cloud & Data & Edge
  - <https://kevinsuo.github.io/code-lab.html>



# Now it's your turn

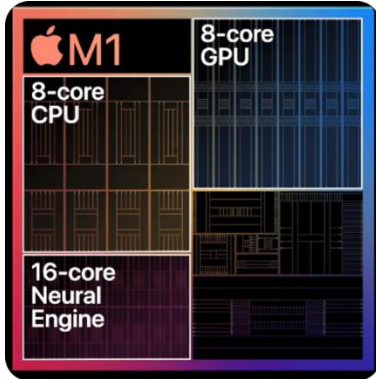
---

- Name, program/year, where from <https://www2.eecs.berkeley.edu/Research/Areas/CS/>
- Your interests in Computer Science
- Have you ever used or heard of parallel and distributed system? Can you name some of them? What do you expect from this course?

If you are in the online course, introduce yourself in D2L, Discussions → Self-Introduction



# Example of HPC & Parallel Programming



personal computer



internet



cloud



Social system



# An Example of Parallel Computing

---

- CPU v.s. GPU

Single core

[NVIDIA: Adam and Jamie explain parallel processing on GPU's \(youtube.com\)](https://www.youtube.com/watch?v=93108364160)



# Course Information

---

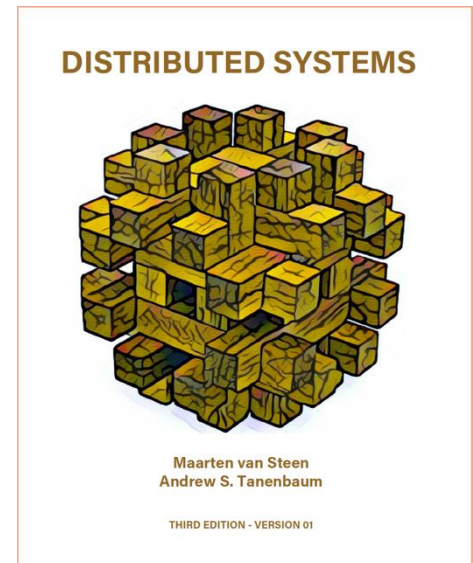
- Instructor: Dr. Kun Suo
- Office: J-3230
- Email: [ksuo@kennesaw.edu](mailto:ksuo@kennesaw.edu)
  - Only reply to e-mails that are sent from KSU student email accounts and title the course number [CS4522]
- Office Hours:
  - Email or Microsoft Teams
  - By appointment
- Course Materials
  - Homework assignments, lecture slides, and other materials will be posted in the webpage (<https://kevinsuo.github.io/teaching.html>) and D2L.



# Reference Book

---

- “Distributed Systems 3rd edition (2017)” by M. van Steen and A.S. Tanenbaum:
  - ISBN-13: 978-1543057386
  - You can get a digital copy of this book for free: <https://www.distributed-systems.net/index.php/books/ds3/>





# Prerequisites

---

- Computer basics that are supposed to be covered in *(CS 3502) Operating Systems*, *(CS 3503) Computer Organization and Architecture* course, *(CS 4504) Parallel System* course.
- **C** programming (code reading, kernel development and debugging). ([Famous projects in C](#))
- **Linux** command line environment (compiling, Makefile, debugging, simple shell programming).



# For C and Linux beginners

---

- C tutorial

- <https://www.tutorialspoint.com/cprogramming/>
- <https://www.learn-c.org>
- <https://www.cprogramming.com/tutorial/c-tutorial.html>

- Linux tutorial

- <https://ryanstutorials.net/linuxtutorial/>
- <http://www.ee.surrey.ac.uk/Teaching/Unix/>
- <https://www.tutorialspoint.com/unix/>



# Project Environment

- Recommend project environment (local)

- VirtualBox + Ubuntu + Linux

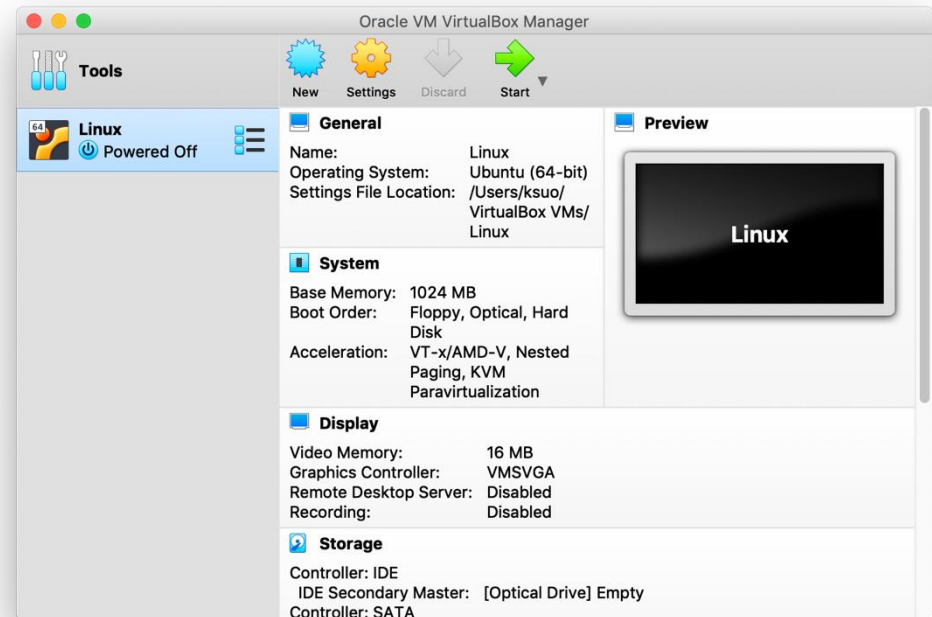
Virtual machine

<https://www.virtualbox.org/>

VM OS

<https://ubuntu.com/download/desktop>

VM OS Kernel



# Project Environment

---

- Recommend project environment (local)
  - VirtualBox + Ubuntu + Linux
- New to VirtualBox?
  - <https://oracle-base.com/articles/vm/virtualbox-creating-a-new-vm>
  - [https://www.youtube.com/watch?v=sB\\_5fqiySi4](https://www.youtube.com/watch?v=sB_5fqiySi4)
  - <https://youtu.be/GDoCrfPma2k> (MacOS)
- You can access to VMs in KSU data centers (cloud) through <https://cseview.kennesaw.edu/>,
  - username: administration; password: linuxadmin



# Why study HPC & Parallel Programming?

---

- Most computer systems today are a certain form of HPC/parallel/distributed systems
  - Internet, datacenters, super computers, mobile devices
  - Most of the applications are parallel or even distributed apps (example: debug decompress file app, [link](#), starts at 2:33)
- To learn useful techniques to build large systems
  - A system with 10,000 nodes is different from one with 100 nodes
- How to deal with imperfections
  - Machines can fail; network is slow; topology is not flat



# What to learn

---

- HPC & Parallel Programming:
  - Parallel hardware
  - Matrix multiplication optimization
  - Pthread programming
  - MPI programming
  - OpenMP programming
  - GPU programming



# Expected Outcomes

---

- Familiar with popular parallel programming libraries (Pthread, OpenMP, MPI, GPU)
- Familiar with fundamentals of program optimization
- The ability to
  - Evaluate the performance of parallel and HPC systems
  - Write simple parallel and HPC programs
  - Understand the tradeoffs in program design



# Course Structure

---

- Lectures
  - Time/Location
  - D2L/Course website
- Projects
  - 5 programming assignments
  - 1 paper/project presentation
- Exams (open books)
  - Midterm: online D2L, TBA.
  - Final: online D2L, TBA





# Course Policy

---

- Grading scale

Percentage	Grade
90 - 100	A
80 - 89	B
70 - 79	C
60 - 69	D
Below 60	F



# Grading Policy (cont.)

---

- Grading percentage
  - Projects (x5): 50%
  - Presentation: 10%, including project or paper presentation
  - Midterm: 20%
  - Final exam: 20%

Late submission policy: late submission will **not be accepted** and **no credits**



# Academic Integrity

---

- Academic dishonesty

[https://scai.kennesaw.edu/KSU\\_Codes\\_of\\_Conduct\\_2019-2020.pdf](https://scai.kennesaw.edu/KSU_Codes_of_Conduct_2019-2020.pdf)

- Cheating

Receiving, attempting to receive, knowingly giving or attempting to give unauthorized assistance...

- Plagiarism

- Collusion

Do not upload course documents to 3<sup>rd</sup> party website without author's permission

- The submission for credit of any work or materials that are attributable in whole or in part to another person

- Taking an examination for another person

- Any act designed to give unfair advantage to a student or the attempt to commit



# Where to go for help ?

---

- Ask questions in class
- Ask questions outside class
  - Classmates and friends
- Attend office hours
  - Send Dr. Kun Suo emails or leave message on teams
- Search on the web
  - Stand on the shoulder of giants



# HPC & Parallel Programming

## Start from An Example

**Kun Suo**

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

# An example of HPC & Parallel Programming: Matrix Multiplication

---

$$\begin{array}{c} \vec{a}_1 \rightarrow \\ \vec{a}_2 \rightarrow \end{array} \begin{array}{c} \vec{b}_1 \quad \vec{b}_2 \\ \downarrow \quad \downarrow \end{array} \begin{array}{c} \left[ \begin{array}{cc} 1 & 7 \\ 2 & 4 \end{array} \right] \cdot \left[ \begin{array}{cc} 3 & 3 \\ 5 & 2 \end{array} \right] = \left[ \begin{array}{cc} \vec{a}_1 \cdot \vec{b}_1 & \vec{a}_1 \cdot \vec{b}_2 \\ \vec{a}_2 \cdot \vec{b}_1 & \vec{a}_2 \cdot \vec{b}_2 \end{array} \right] \end{array}$$

$A \qquad B \qquad C$



# Matrix multiply

<https://github.com/kevinsuo/CS7172/blob/master/matrix.c>

```
int main()
{
    initMatrix();

    double time_spent = 0.0;
    clock_t begin = clock();

    matrixMultiply();

    clock_t end = clock();
    time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
    printf("Time elapsed is %f seconds", time_spent);

    return 0;
}
```



# Matrix multiply

<https://github.com/kevinsuo/CS7172/blob/master/matrix.c>

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define N 1000

double A[N][N], B[N][N], C[N][N];

void initMatrix()
{
    int i, j = 0;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            A[i][j] = rand() % 100 + 1; //generate a number between [1, 100]
            B[i][j] = rand() % 100 + 1; //generate a number between [1, 100]
        }
    }
}
```



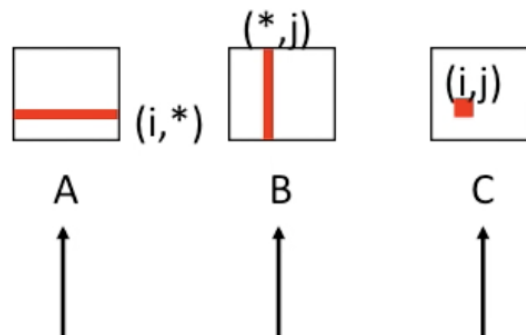


# Matrix multiply

<https://github.com/kevinsuo/CS7172/blob/master/matrix.c>

```
void matrixMultiply() {  
    int i, j, k = 0;  
    for (i = 0; i < N; i++) {  
        for (j = 0; j < N; j++) {  
            for (k = 0; k < N; k++) {  
                C[i][j] += A[i][k] * B[k][j];  
            }  
        }  
    }  
}
```

Inner loop:



Row-wise

Column-  
wise

Fixed



# Matrix multiply

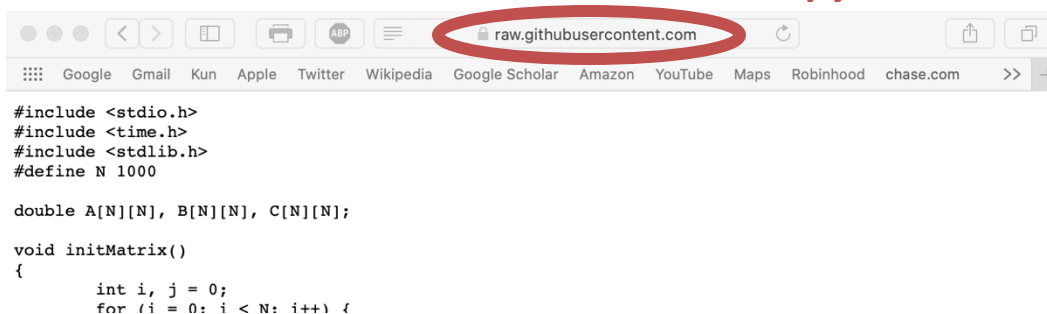
<https://github.com/kevinsuo/CS7172/blob/master/matrix.c>



1. **Raw**

```
46 lines (36 sloc) | 775 Bytes
1  #include <stdio.h>
2  #include <time.h>
3  #include <stdlib.h>
4  #define N 1000
5
6  double A[N][N], B[N][N], C[N][N];
7
8  void initMatrix()
9  {
10     int i, j = 0;
11     for (i = 0; i < N; i++) {
12         for (j = 0; j < N; j++) {
13             A[i][j] = rand() % 100 + 1; //generate a number between [1, 100]
14             B[i][j] = rand() % 100 + 1; //generate a number between [1, 100]
15         }
16     }
17 }
18
```

2. Copy the URL

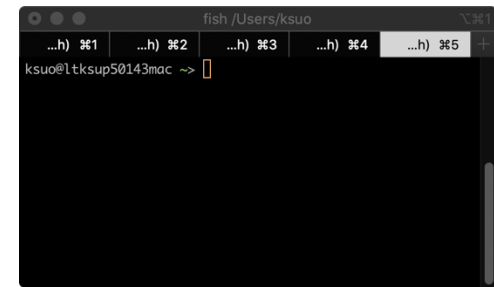


[raw.githubusercontent.com](https://raw.githubusercontent.com/kevinsuo/CS7172/master/matrix.c)

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define N 1000

double A[N][N], B[N][N], C[N][N];

void initMatrix()
{
    int i, j = 0;
    for (i = 0; i < N; i++) {
```



3.

\$ wget URL

\$ gcc filename.c -o filename.o

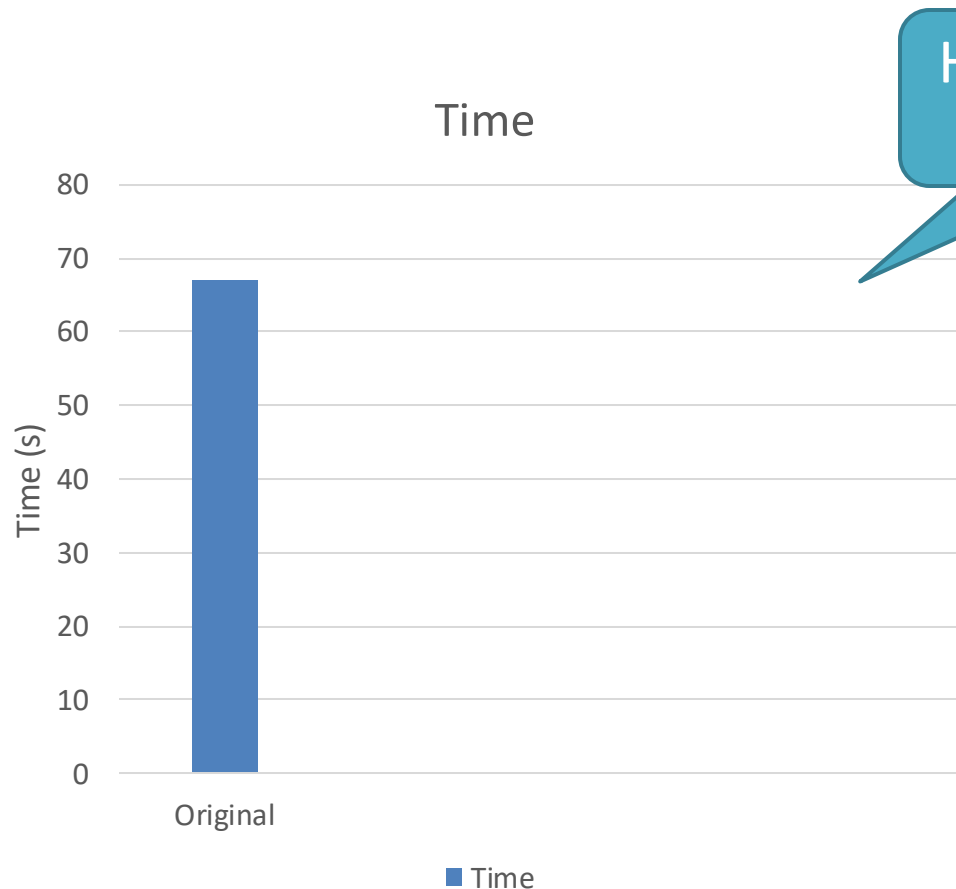
\$ ./filename.o

(if no wget/gcc,

\$ sudo apt install wget, gcc)



# Matrix multiply



How to run it faster?

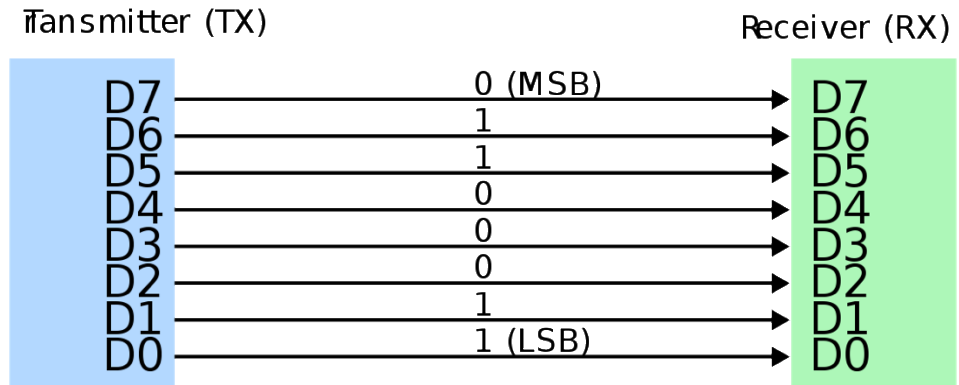
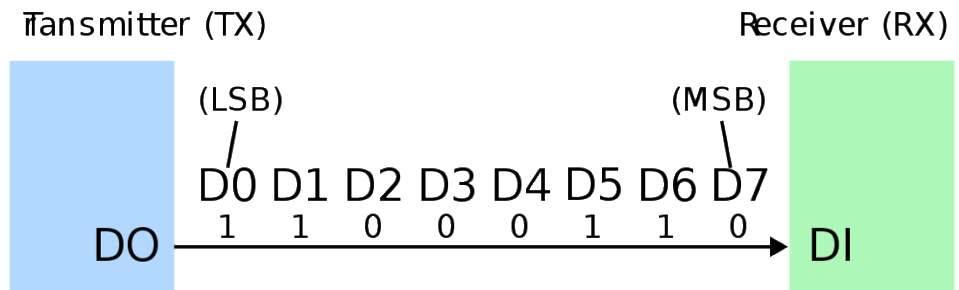
1. Accelerate serial execution
2. Accelerate in parallel



# How to run it faster?

1. Accelerate serial execution  
Reduce unnecessary steps

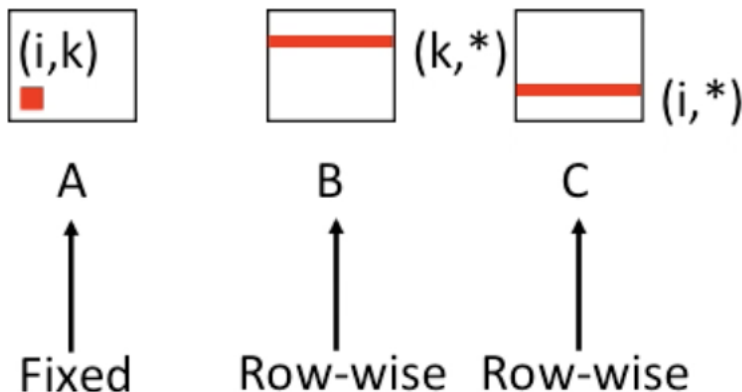
2. Accelerate in parallel



# Option 1: Optimization using locality

```
void matrixMultiply() {  
    int i, j, k = 0;  
    for (k = 0; k < N; k++) {  
        for (i = 0; i < N; i++) {  
            for (j = 0; j < N; j++) {  
                C[i][j] += A[i][k] * B[k][j];  
            }  
        }  
    }  
}
```

Inner loop:



<https://github.com/kevinsuo/CS7172/blob/master/matrix-opt.c>

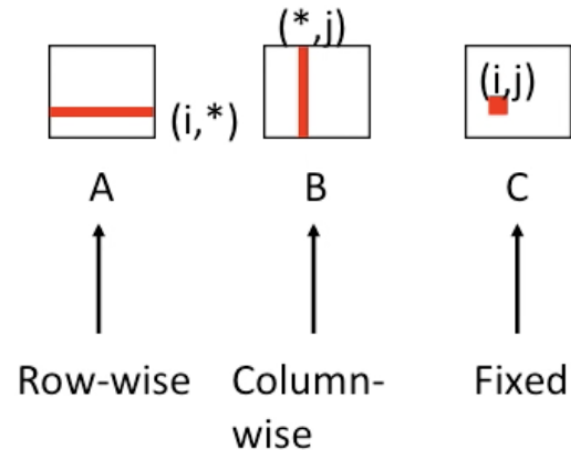
# Option 1: Optimization using locality

```
ksuo@ksuo-VirtualBox ~/cs7172> ./a.o
Time elapsed is 67.452589 seconds
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172> ./a2.o
Time elapsed is 18.149353 seconds
```

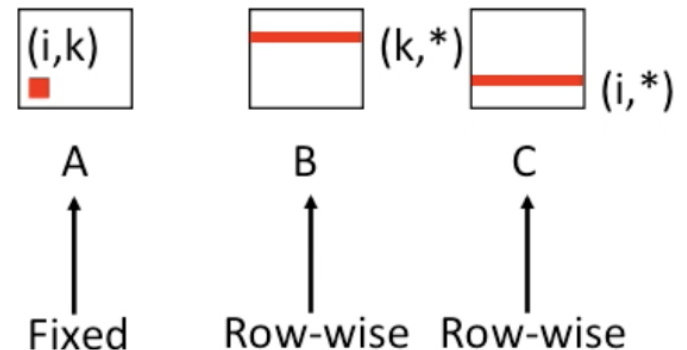
N=2000

3.7x

Inner loop:

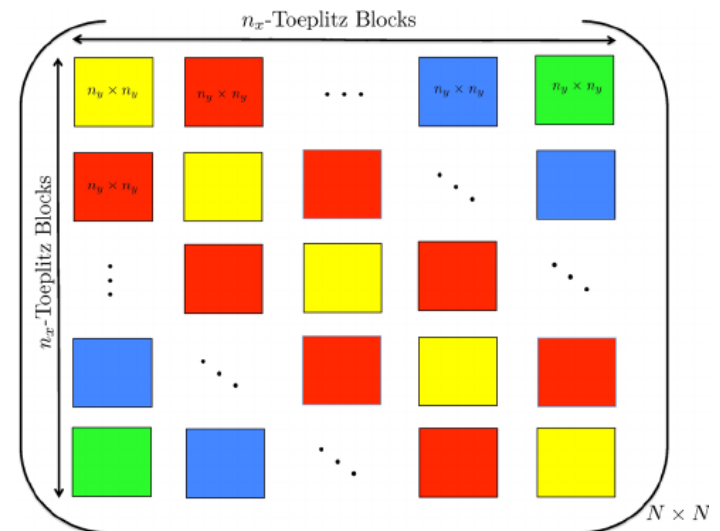
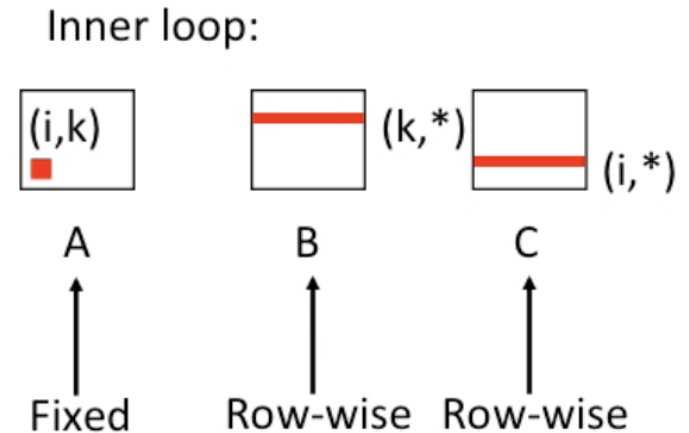


Inner loop:



# Option 1: Optimization using locality

- Temporal locality
  - Every inner loop reuse the value of  $A[i, k]$
- Spatial locality
  - Divide the large matrix into smaller ones and put it inside the cache during calculation



# Option 1: Optimization using locality

```
void matrixMultiply() {  
    int i, j, k = 0;  
    int i2, j2, k2 = 0;  
  
    for (k2 = 0; k2 < N; k2+=BLOCK_SIZE) {  
        for (i2 = 0; i2 < N; i2+=BLOCK_SIZE) {  
            for (j2 = 0; j2 < N; j2+=BLOCK_SIZE) {  
                //inside each block  
                for (k = k2; k < k2+BLOCK_SIZE; k++) {  
                    for (i = i2; i < i2+BLOCK_SIZE; i++) {  
                        for (j = j2; j < j2+BLOCK_SIZE; j++) {  
                            C[i][j] += A[i][k] * B[k][j];  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

<https://github.com/kevinsuo/CS7172/blob/master/matrix-opt2.c>

N = 2000

$$\begin{pmatrix} J_1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 \\ 0 & 0 & J_2 & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & 0 & 0 & J_3 \end{pmatrix}$$





# Option 1: Optimization using locality

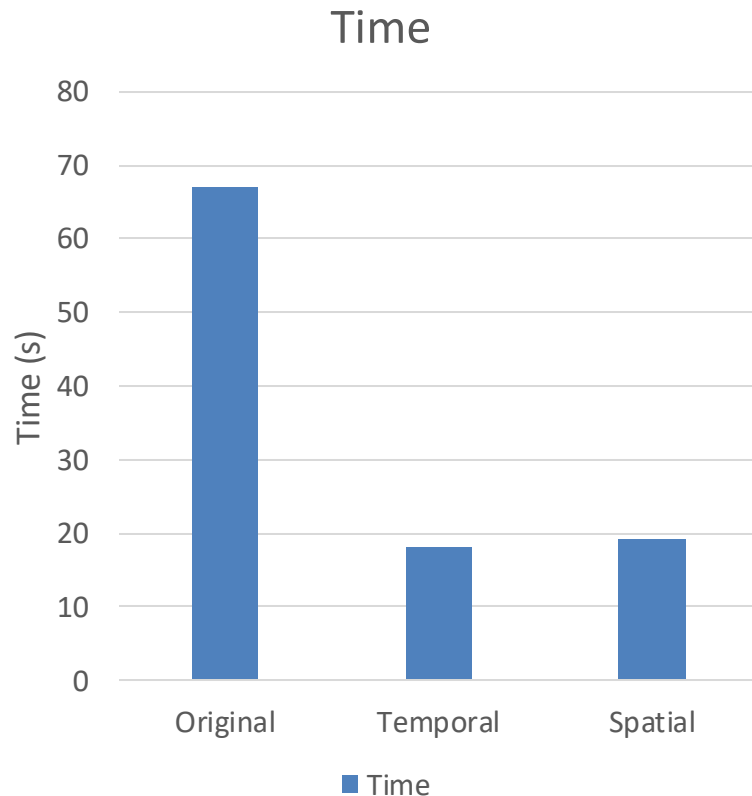
$$A = \left( \begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) \Rightarrow \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$A_{11} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, A_{12} = \begin{pmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \end{pmatrix}$$

$$A_{21} = \begin{pmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \end{pmatrix}, A_{22} = \begin{pmatrix} a_{33} & a_{34} \\ a_{43} & a_{44} \end{pmatrix}$$



# Option 1: Optimization using locality



```
ksuo@ksuo-VirtualBox ~/cs7172> ./a.o
Time elapsed is 67.845517 seconds
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172>
ksuo@ksuo-VirtualBox ~/cs7172> ./a3.o
Time elapsed is 19.115410 seconds
```



# Optimal 2: Optimization using parallel

---

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

Task 1:  $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$

Task 2:  $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$

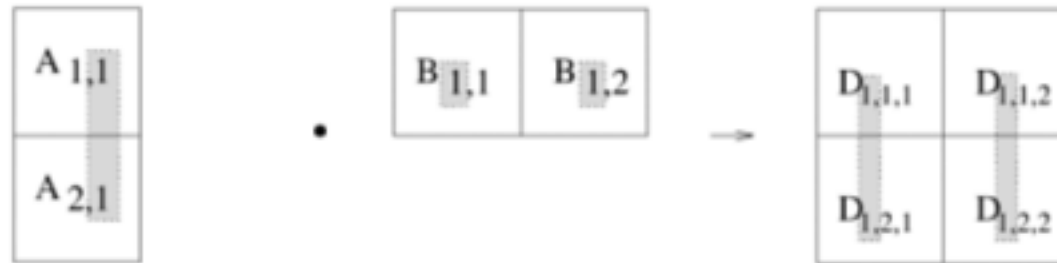
Task 3:  $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$

Task 4:  $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

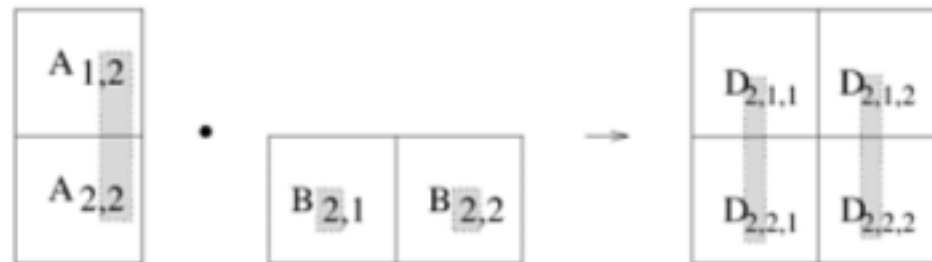


# Optimal 2: Optimization using parallel

Thread 1:

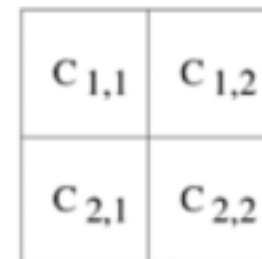


Thread 2:



+

↓



# Reading Materials

---

- Strassen algorithm
  - In 1969, Volker Strassen proposed a matrix multiplication algorithm with a complexity of  $O(n^{2.807})$ : [Link](#). This was the first time in history that computational complexity of matrix multiplication was reduced below  $O(n^3)$ .
- Coppersmith–Winograd algorithm
  - In 1990, Don Coppersmith and Shmuel Winograd made a groundbreaking achievement by reducing the complexity to  $O(n^{2.3727})$ . Paper: [Link](#)



# Optimization and Speedup

---

	N=200	N=400	N=800	N=1600
matrix				
matrix-opt1				
matrix-opt2				

	N=200	N=400	N=800	N=1600
matrix				
matrix-opt1				
matrix-opt2				



# Single thread app demo

<https://youtu.be/dlsBhvhQ9mA>



A terminal window titled "fish /home/administrator" with a tab labeled "fish /home/administrator (ssh)". The terminal shows the command "fish" being executed, resulting in the message "Welcome to fish, the friendly interactive shell" and the prompt "administrator@ubuntu1804vm ~->". A blue speech bubble points to the terminal with the text "Single thread program".

```
fish /home/administrator (ssh) %1
administrator@ubuntu1804vm:~$ fish
Welcome to fish, the friendly interactive shell
administrator@ubuntu1804vm ~->
```

Single thread  
program



# Multi-thread app demo

<https://youtu.be/ubLB2fb8cdc>

```
fish /home/administrator
fish /home/administrator/Downloads (ssh) %1
fish /home/administrator (ssh) %2 +

See 'snap info htop' for additional versions.

administrator@ubuntu1804vm ~-> sudo apt install htop -y
[sudo] password for administrator:
No logon servers
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  efibootmgr libegl1-mesa libfwpup1 libllvm9 linux-hwe-5.4-headers-5.4.0-42 linux-hwe-5.4-headers-5.4.0-45
  linux-hwe-5.4-headers-5.4.0-47 linux-hwe-5.4-headers-5.4.0-48 linux-hwe-5.4-headers-5.4.0-51
  linux-hwe-5.4-headers-5.4.0-52
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  htop
0 upgraded, 1 newly installed, 0 to remove and 88 not upgraded.
Need to get 80.0 kB of archives.
After this operation, 221 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 htop amd64 2.1.0-3 [80.0 kB]
Fetched 80.0 kB in 0s (554 kB/s)
Selecting previously unselected package htop.
(Reading database ... 288356 files and directories currently installed.)
Preparing to unpack .../htop_2.1.0-3_amd64.deb ...
Unpacking htop (2.1.0-3) ...
Setting up htop (2.1.0-3) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
administrator@ubuntu1804vm ~-> htop
administrator@ubuntu1804vm ~-> █
```

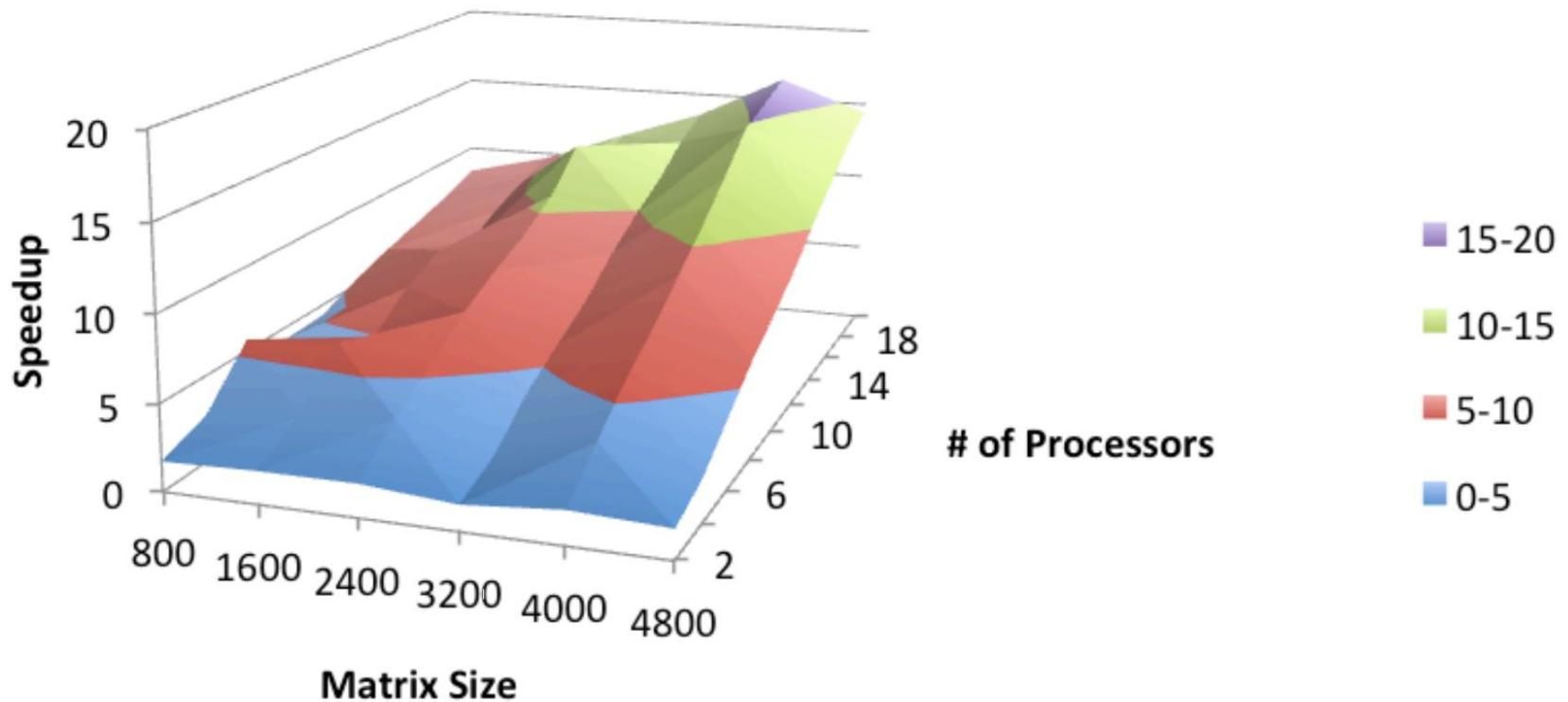
Multi-thread  
program





# Optimal 2: Optimization using parallel

---



[https://www.cse.unr.edu/~fredh/class/415/Nolan/matrix\\_multiplication/writeup.pdf](https://www.cse.unr.edu/~fredh/class/415/Nolan/matrix_multiplication/writeup.pdf)



# Example of distributed system: sorting

- Sorting on a single machine, e.g., Database

```
select field_a from table_b order by field_a limit 100, 10;
```

```
db.collection_b  
.find()  
.sort({"field_a":1})  
.skip(100)  
.limit(10);
```

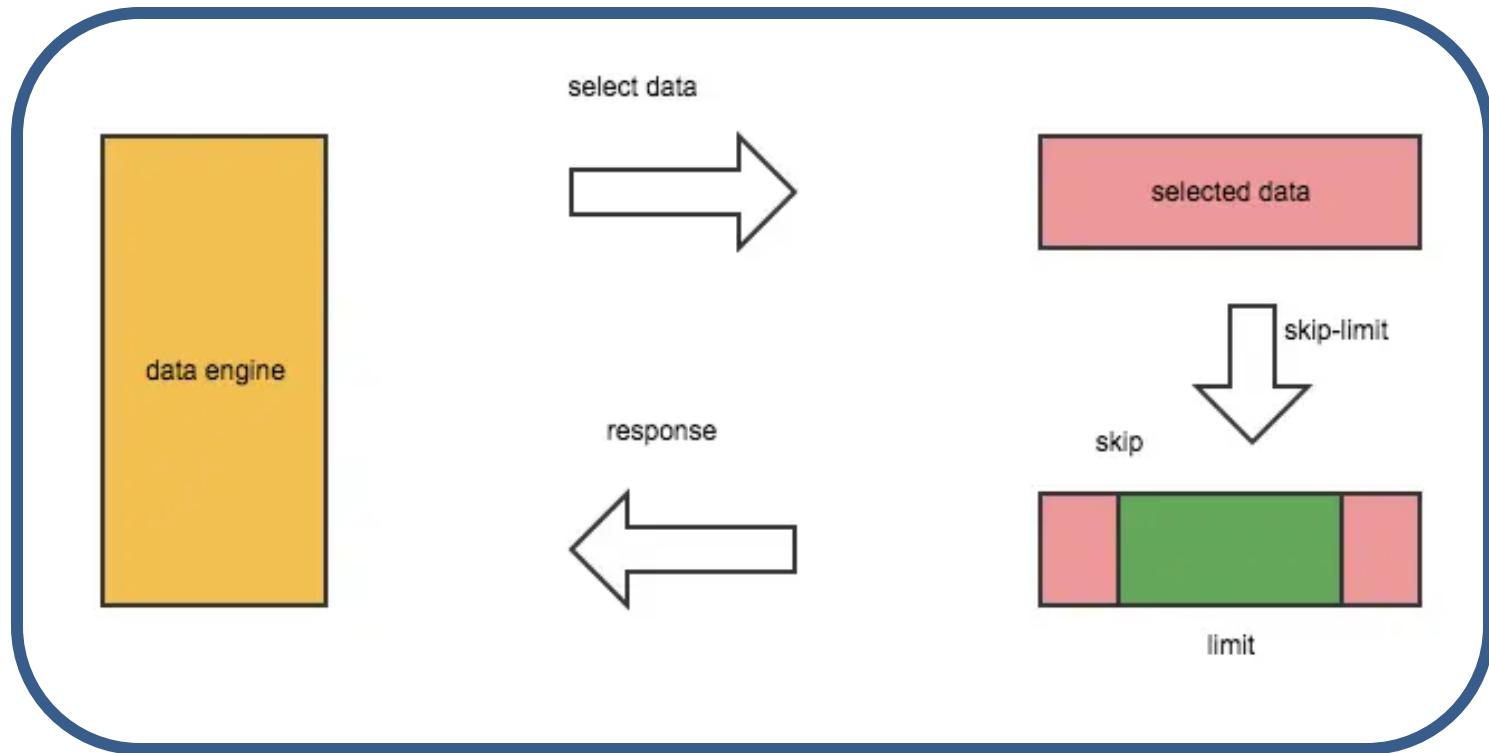
	field_a	field_b	field_c
100			
...			
...			
110			

From line 100  
to the next 10  
lines of data

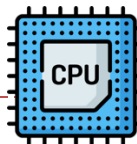


# Example of distributed system: sorting

- Workflow on a single node



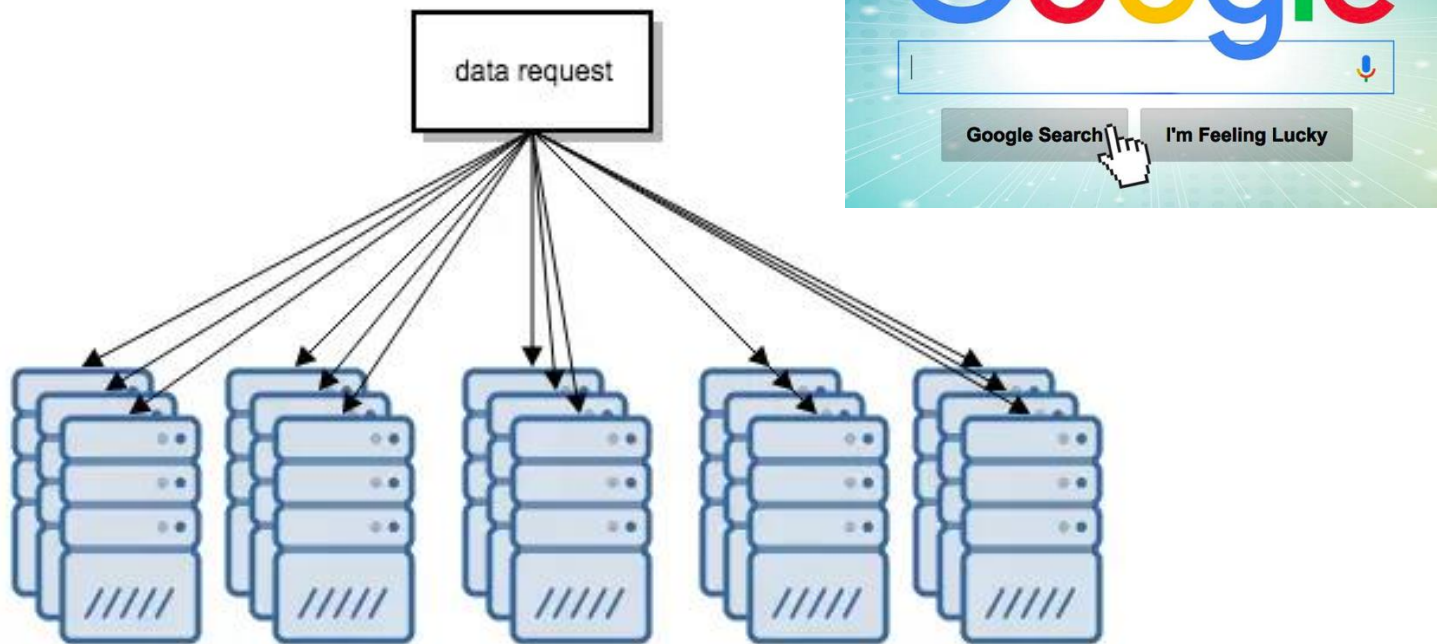
One server



# Example of distributed system: sorting

---

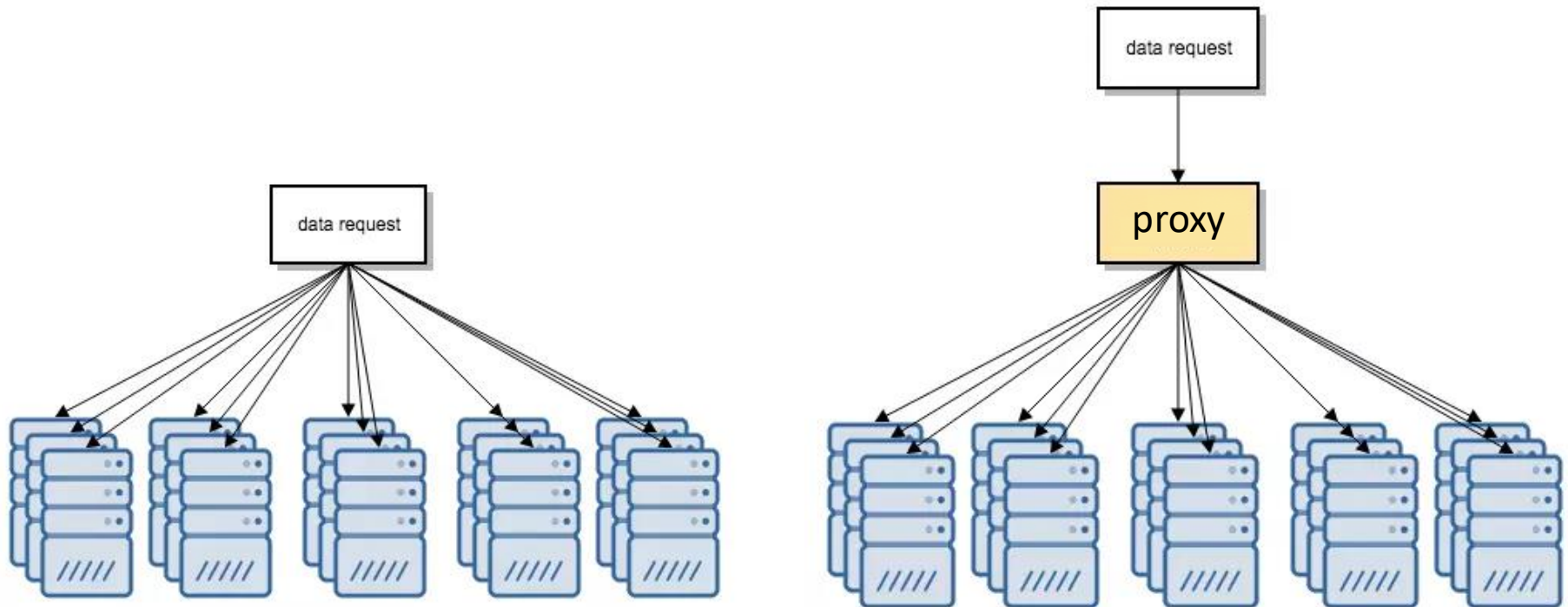
- If the data is too much and single node cannot hold



# Example of distributed system: sorting

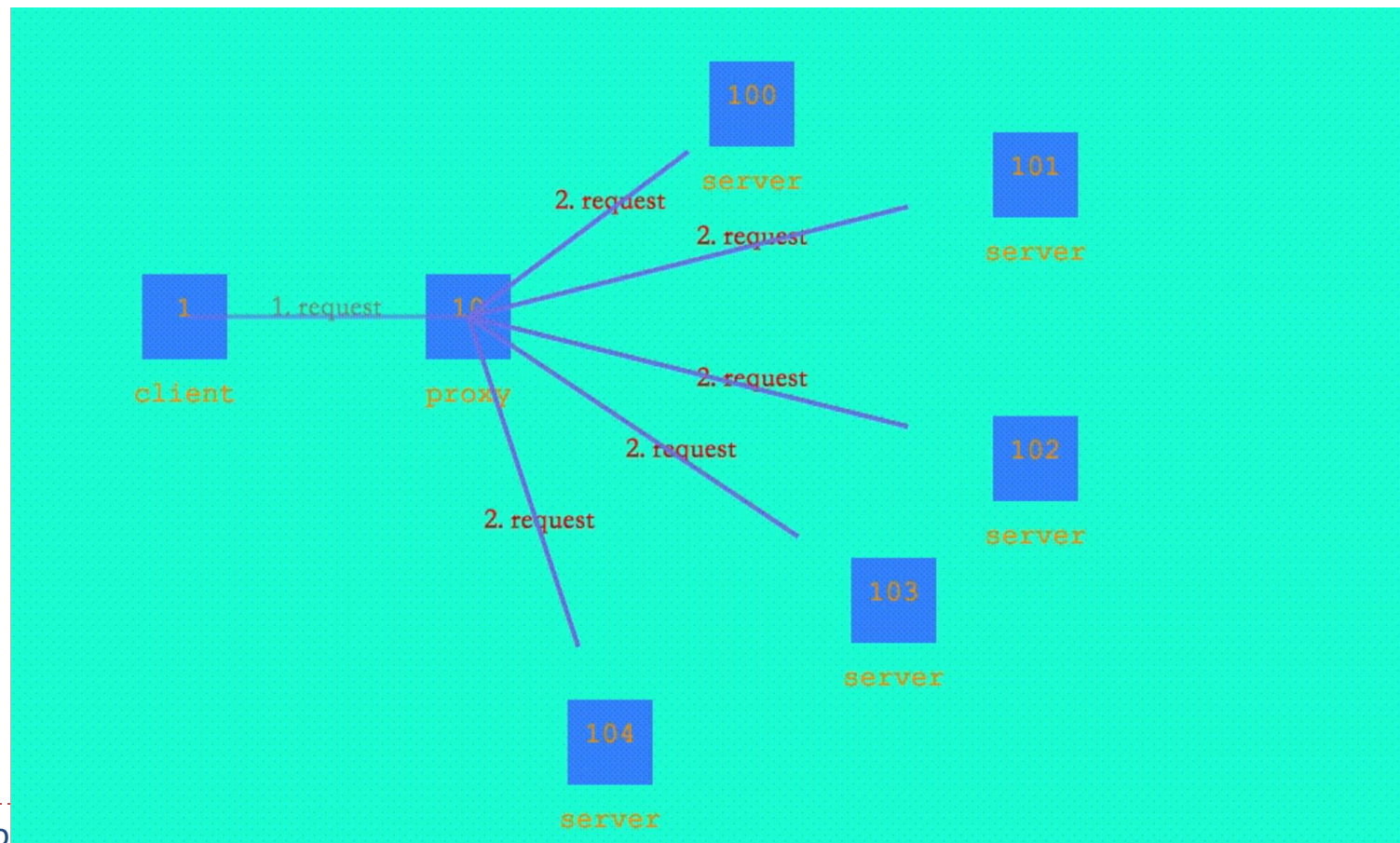
---

- Choose a node for merge processing



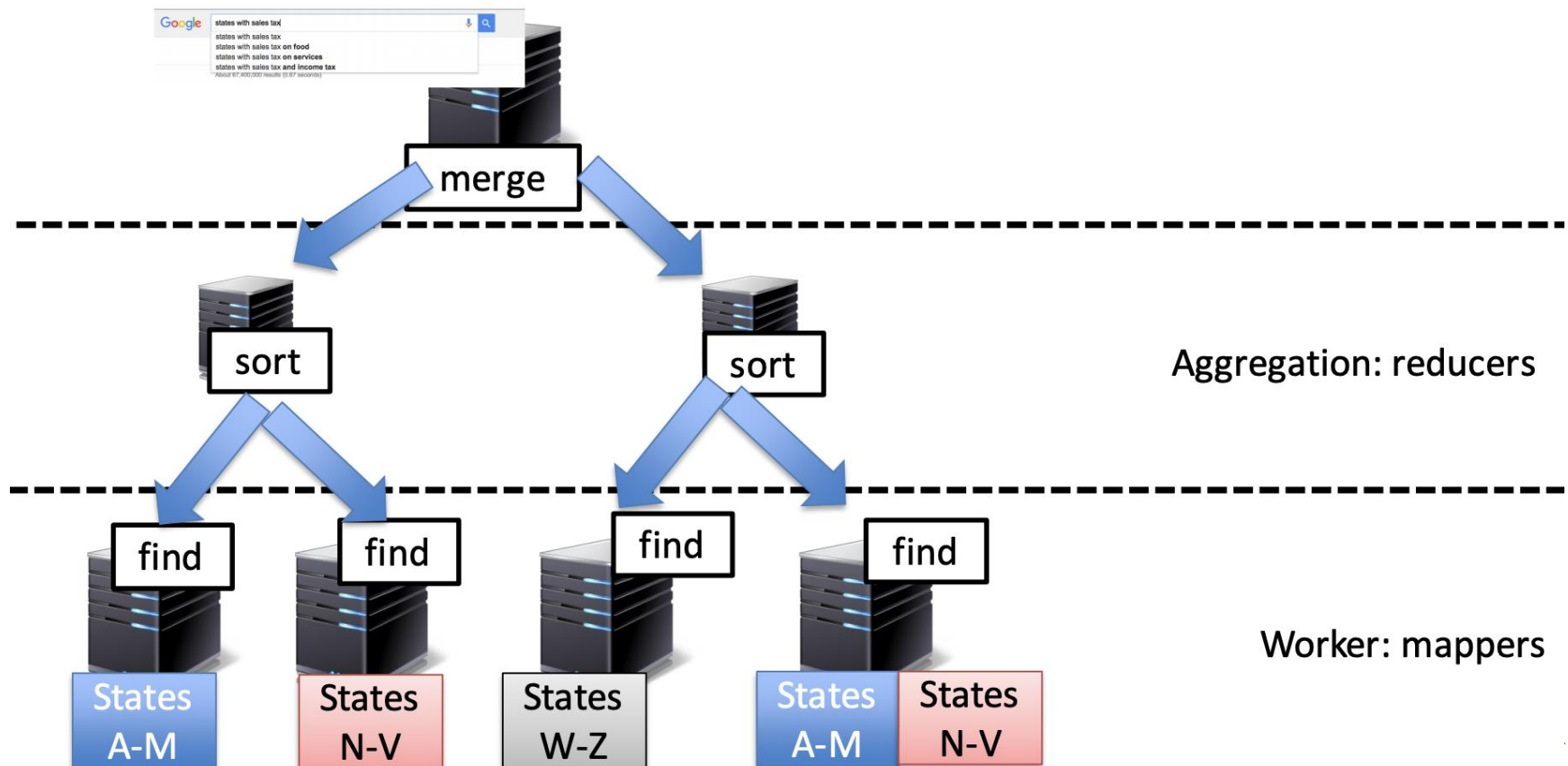
# Example of distributed system: sorting

- Workflow



# Example of distributed system: sorting

- How Do Request Get Processed in a Data Center





# Example of distributed system: sorting

---

- How Google Search Works
- <https://www.youtube.com/watch?v=0eKVizvYSUQ>





# Conclusion

---

- Why study HPC & parallel programming?
- What to learn?
- Course structure
- Course policy
- An example of HPC & parallel programming

