2110625
Data Science Architecture

System architecture for data science solutions

Parallel computing

รศ.ดร. วีระ เหมืองสิน

Assoc. Prof. Veera Muangsin

Department of Computer Engineering

Chulalongkorn University

veera.m@chula.ac.th

Why study computer architecture?

- To understand *how features of computer architecture impact applications*

  and, conversely, *how requirements of applications lead to the emergence of new hardware and architectural support*

- For system architect: *to design a system that can run applications efficiently*
- For SW developer: *to develop software that efficiently utilizes the underlying system*
- For data scientist: *to make informed decisions about selecting hardware and programming tools for data processing and analysis tasks.*

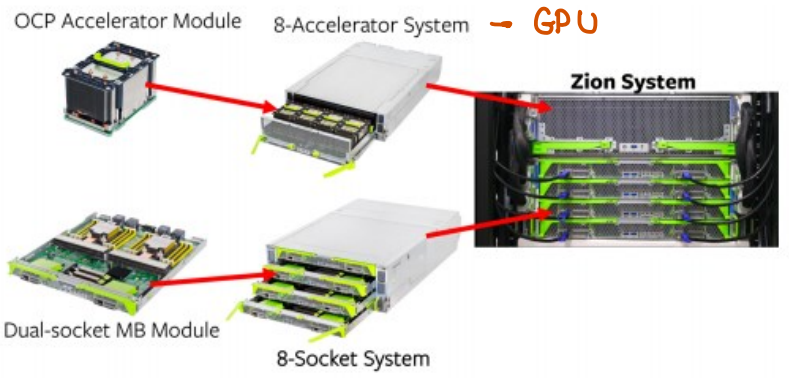
- Examples
  - Software requirement/behavior: *Most memory accesses are to recently used data or nearby data.*
  - Architecture change: *memory hierarchy*
  - cache memory: *small and fast memory for the to-be-used-soon data*
  - Software change: *Programmers write program to optimize cache usage.*

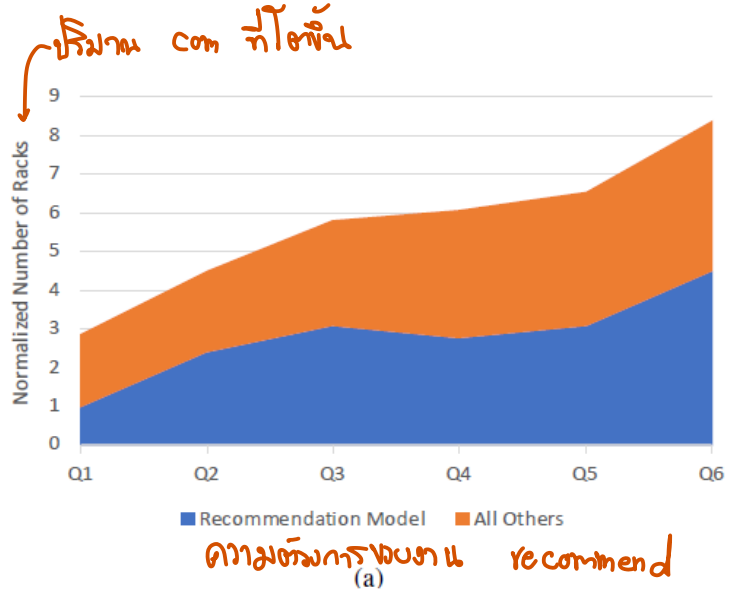
Requirements and architecture of a machine learning system

Deep Learning Training i
Design of Scale-up a

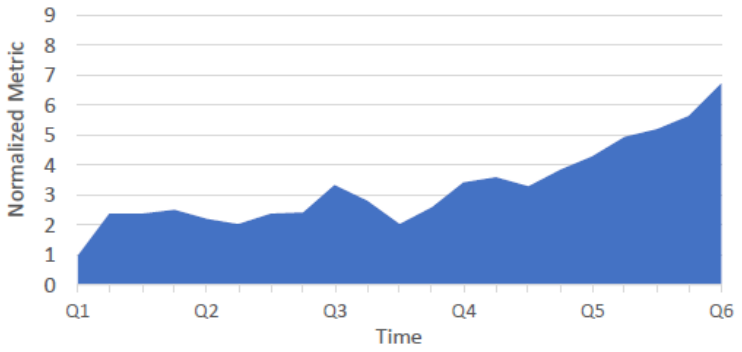
Maxim Naumov*, John Kim†, Dheevatsa M
Whitney Zhao, Serhat Yilmaz, Changkyu Kim,
Isabel Gao, Bor-Yiing Su, Jiya
Facebook, 1 Hacker

***Abstract*—Large-scale traini**
performance and accuracy of m
book we use many different m
video and language models. Ho
the deep learning recommenda
responsible for more than 50% of the training demand in our data centers. Recommendation models present unique challenges in training because they exercise not only compute but also memory capacity as well as memory and network bandwidth. As model size and complexity increase, efficiently scaling training becomes a challenge. To address it we design Zion – Facebook's next-generation large-memory training platform that consists of both CPUs and accelerators. Also, we discuss the design requirements of future scale-out training systems.





(a)



(b)

Fig. 1. (a) Server compute demand for training and (b) number of distributed training workflows across Facebook data centers.

https://research.facebook.com/publications/deep-learning-training-in-facebook-data-centers-design-of-scale-up-and-scale-out-systems/

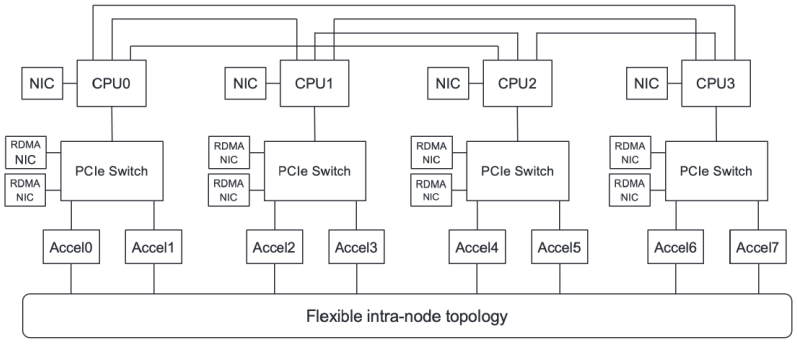
Requirements and architecture of a machine learning system

Neo ใช้ Parallelism 4 วิธี เป็นการทำ distributed training model ใหม่ต้องกระจาย load ไปยัง cluster

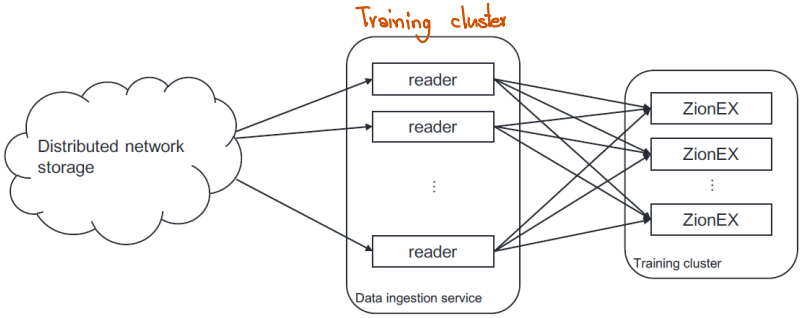
Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models

Dheevatsa Mudigere[†‡], Yuchen Hao[†‡], Jianyu Huang[†‡], Zhihao Jia[§], Andrew Tulloch[‡], Srinivas Sridharan[‡], Xing Liu[‡], Mustafa Ozdal[‡], Jade Nie[‡], Jongsoo Park[‡], Liang Luo[‡], Jie (Amy) Yang[‡], Leon Gao[‡], Dmytro Ivchenko[‡], Aarti Basant[‡], Yuxi Hu[‡], Jiyan Yang[‡], Ehsan K. Ardestani[‡], Xiaodong Wang[‡], Rakesh Komuravelli[‡], Ching-Hsiang Chu[‡], Serhat Yilmaz[‡], Huayu Li[‡], Jiyuan Qian[‡], Zhuobo Feng[‡], Yinbin Ma[‡], Junjie Yang[‡], Ellie Wen[‡], Hong Li[‡], Lin Yang[‡], Chonglin Sun[‡], Whitney Zhao[‡], Dimitry Melts[‡], Krishna Dhulipala[‡], KR Kishore[‡], Tyler Graf[‡], Assaf Eisenman[‡], Kiran Kumar Matam[‡], Adi Gangidi[‡], Guoqiang Jerry Chen[‡], Manoj Krishnan[‡], Avinash Nayak[‡], Krishnakumar Nair[‡], Bharath Muthiah[‡], Mahmoud khorashadi[‡], Pallab Bhattacharya[‡], Petr Lapukhov[‡], Maxim Naumov[‡], Ajit Mathews[‡], Lin Qiao[‡], Mikhail Smelyanskiy[‡], Bill Jia[‡], Vijay Rao[‡]

[‡]Meta Platforms, [§]Carnegie Mellon University



(b) *ZionEX*



(c) Overall training architecture.

ABSTRACT

Deep learning recommendat…been used across many business-critic…e the single largest AI application in …and in its data-centers. In this paper, we present Neo, a software-hardware co-designed system for high-performance distributed training of large-scale DLRMs. Neo employs a novel 4D parallelism strategy that combines table-wise, row-wise, column-wise, and data parallelism for training massive embedding operators in DLRMs. In addition, Neo enables extremely high-performance and memory-efficient embedding computations using a variety of critical systems optimizations, including hybrid kernel fusion, software-managed caching, and quality-preserving compression. Finally, Neo is paired with *ZionEX*, a new hardware platform co-designed with Neo's 4D parallelism for optimizing communications for large-scale DLRM training. Our evaluation on 128 GPUs using 16 *ZionEX* nodes shows that Neo outperforms existing systems by up to 40× for training 12-trillion-parameter DLRM models deployed in production.

ทำงานร่วมกัน new HW (ZionEX) รองรับ model Neo

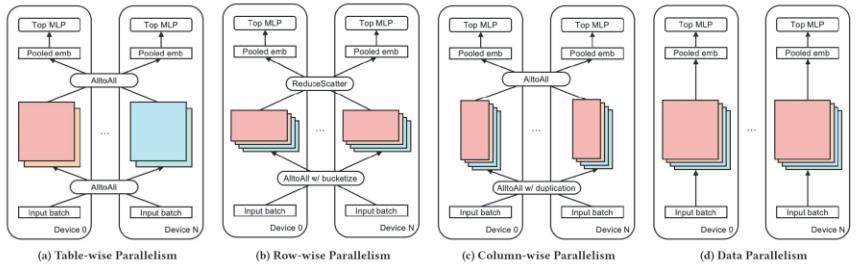
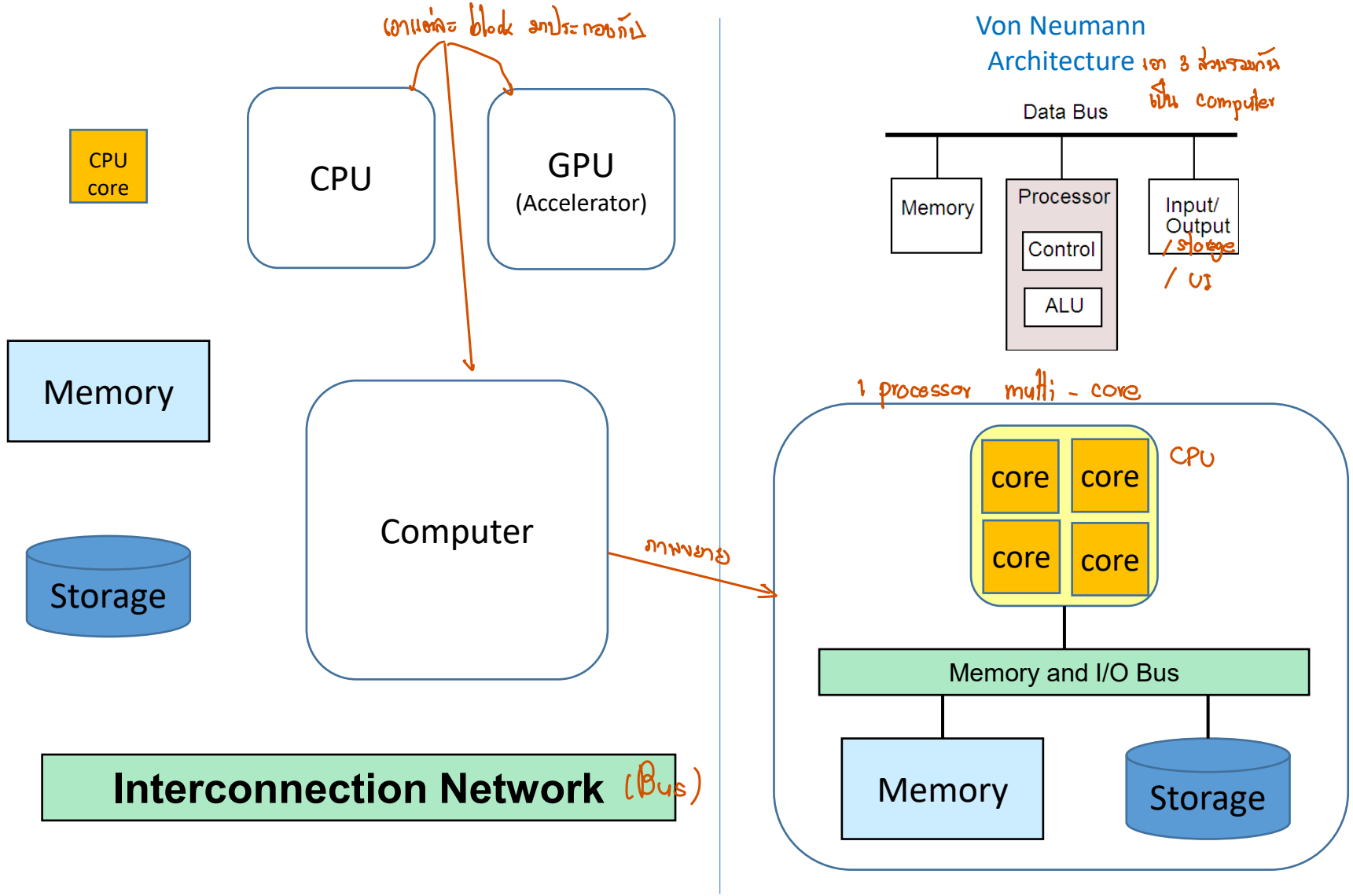


Figure 5: Embedding table sharding schemes with different implications on the communication cost, load balancing and memory requirement. Bottom MLP is omitted in this figure for simplicity of illustration.

https://research.facebook.com/publications/software-hardware-co-design-for-fast-and-scalable-training-of-deep-learning-recommendation-models/

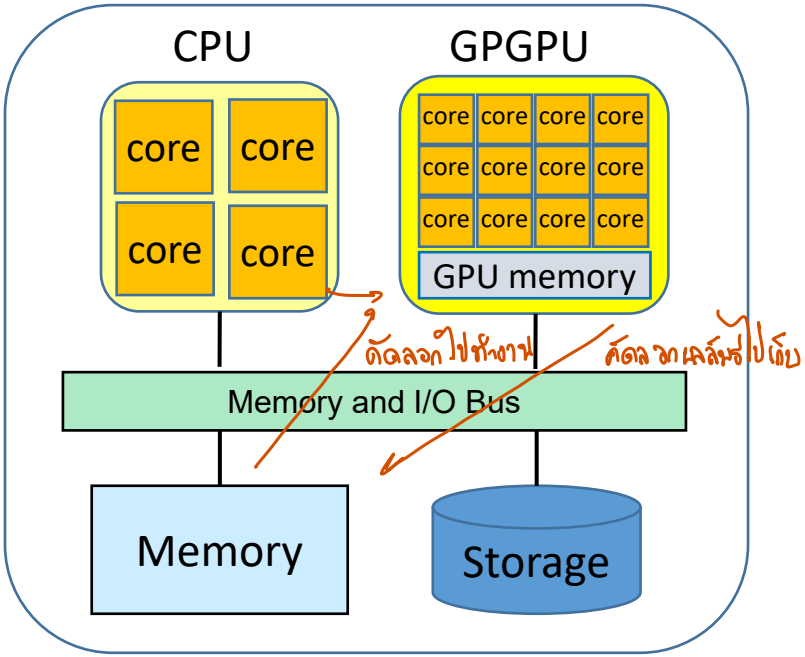
Processing Architecture

Building blocks of a computer system



GPGPU ทำงานต่างกว่า CPU ปกติ

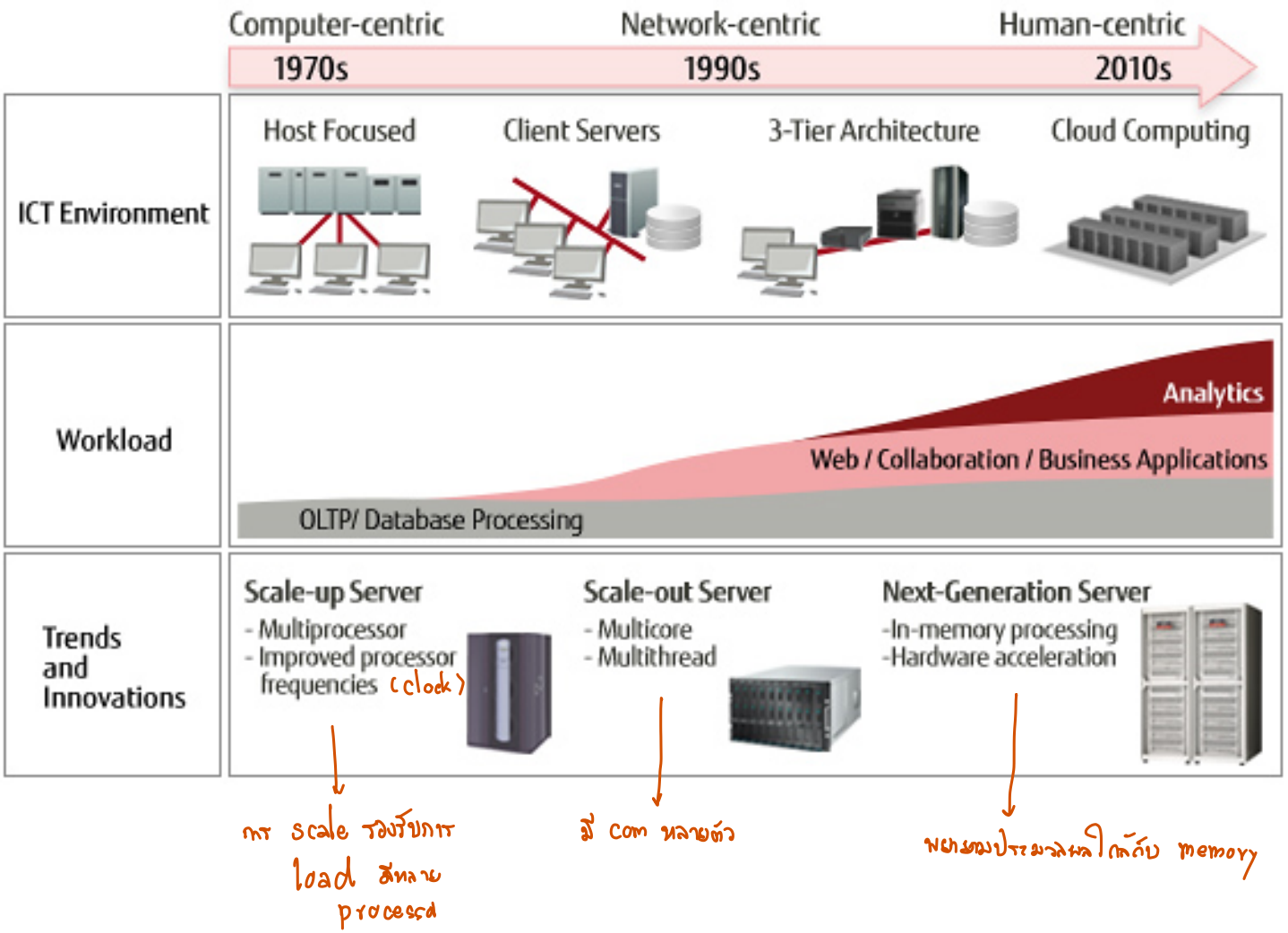
เมื่อก่อนใช้กับ Video card อย่างเดียว



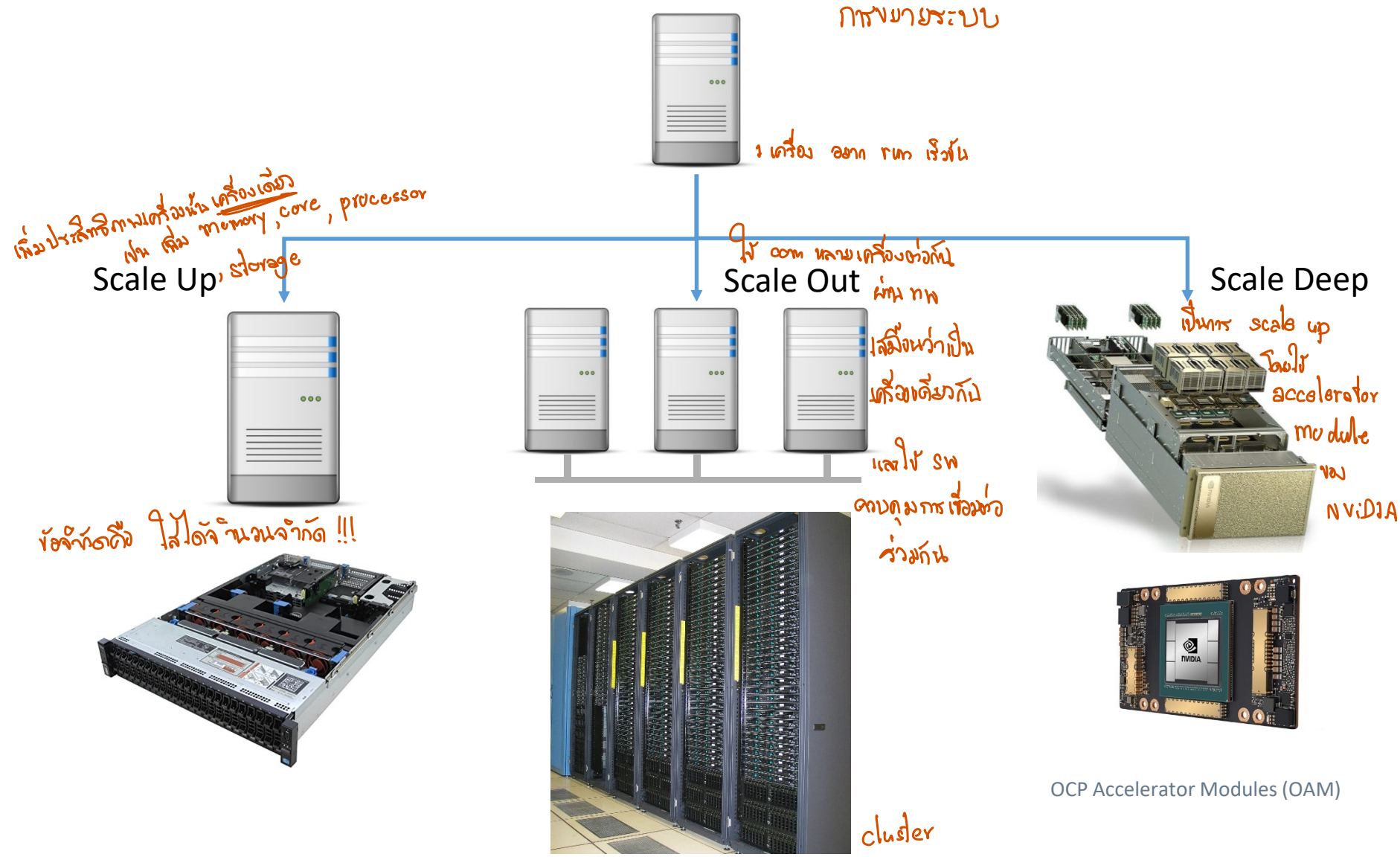
- GPU is an accelerator with vector processing architecture for graphics and image processing. GPU มีความสามารถประมวลผล graphic,
- GPU offers more parallelism via threads and vectorization. matrix
- General-purpose GPU (GPGPU) provides programming model (e.g. CUDA, OpenCL) for other applications that exhibit data parallelism. นอกจากโปรแกรมที่ใช้ความสามารถ GPGPU อย่างอื่น ที่ไม่ใช่แค่เรื่อง CG
- CPU is needed to run OS and programs. CPU ยังต้องมีอยู่ เพราะชุดคำสั่งต่างจาก GPU
- CPU offloads some computation tasks to GPU. ทำการส่ง code และ data ให้กับ GPU ประมวลผล
- GPU has its own memory. GPU มี memory ของตัวเอง
- Data must be transferred between system memory and GPU memory

การก็อปลอกข้อมูลมี overhead อยู่

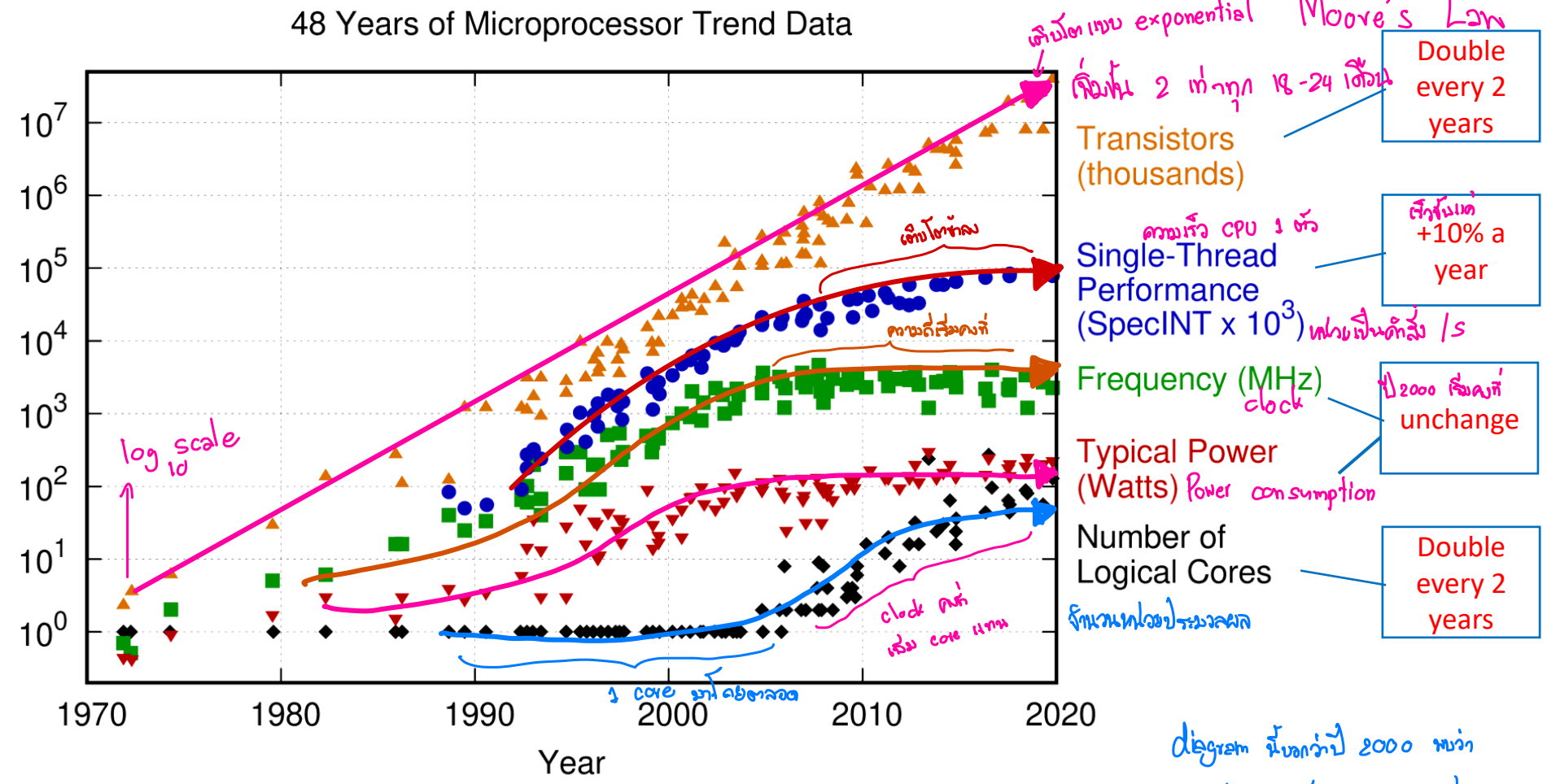
Evolution of Computer Systems



Scale Up, Scale Out, Scale Deep



Microprocessor Trends



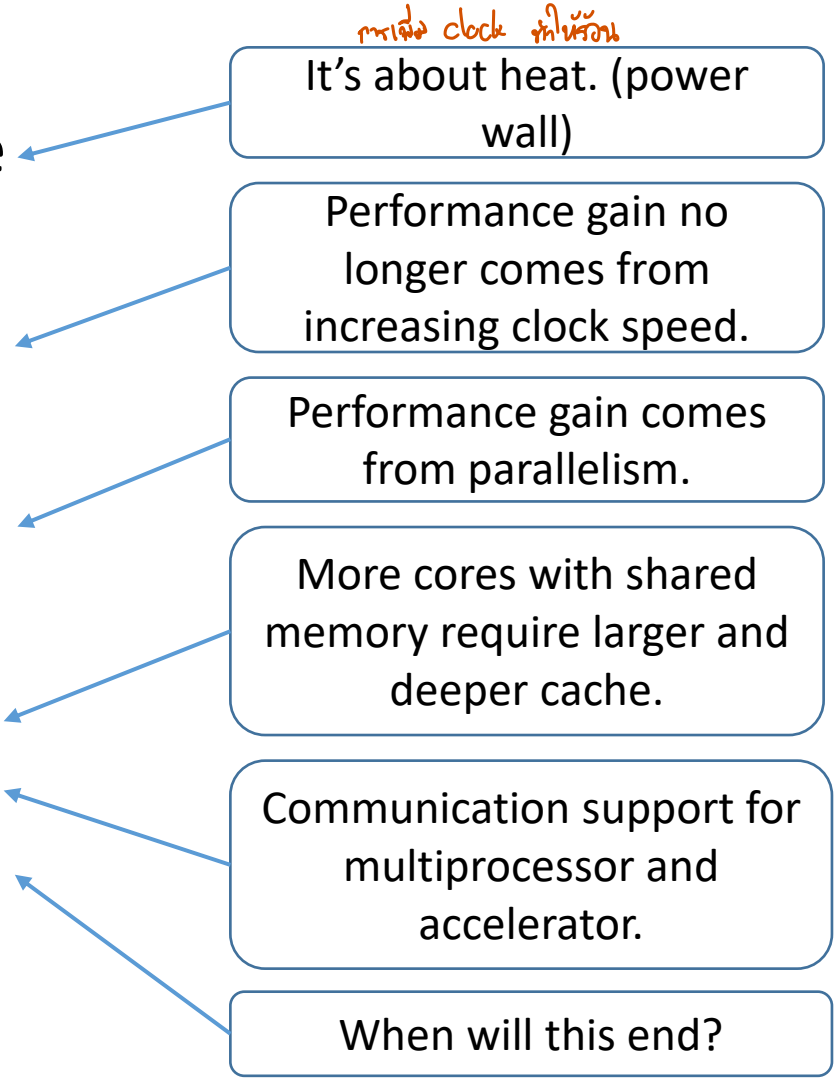
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Power Wall + Memory Wall + ILP Wall = Brick Wall

https://github.com/samhatfield/microprocessor-trend-data

Microprocessor Trends

- Clock speed and power have stopped increasing.
- Single core speed still increase but slowly.
- Number of logical cores grows exponentially.
- Number of transistors still grows exponentially.

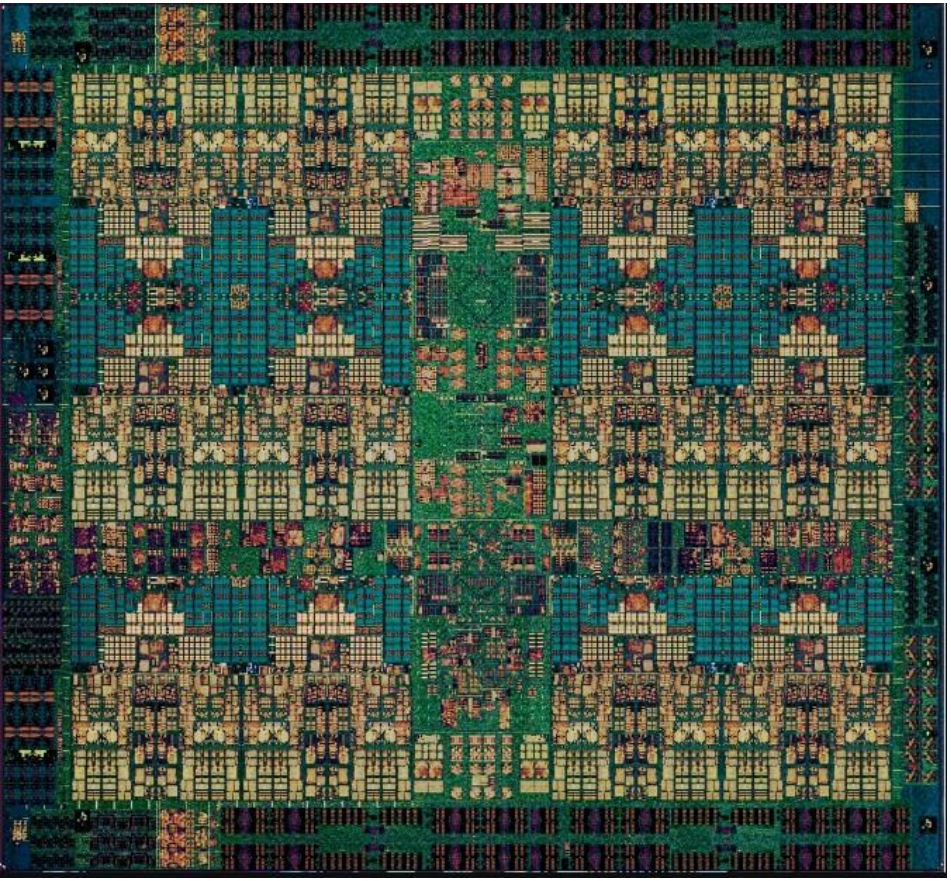


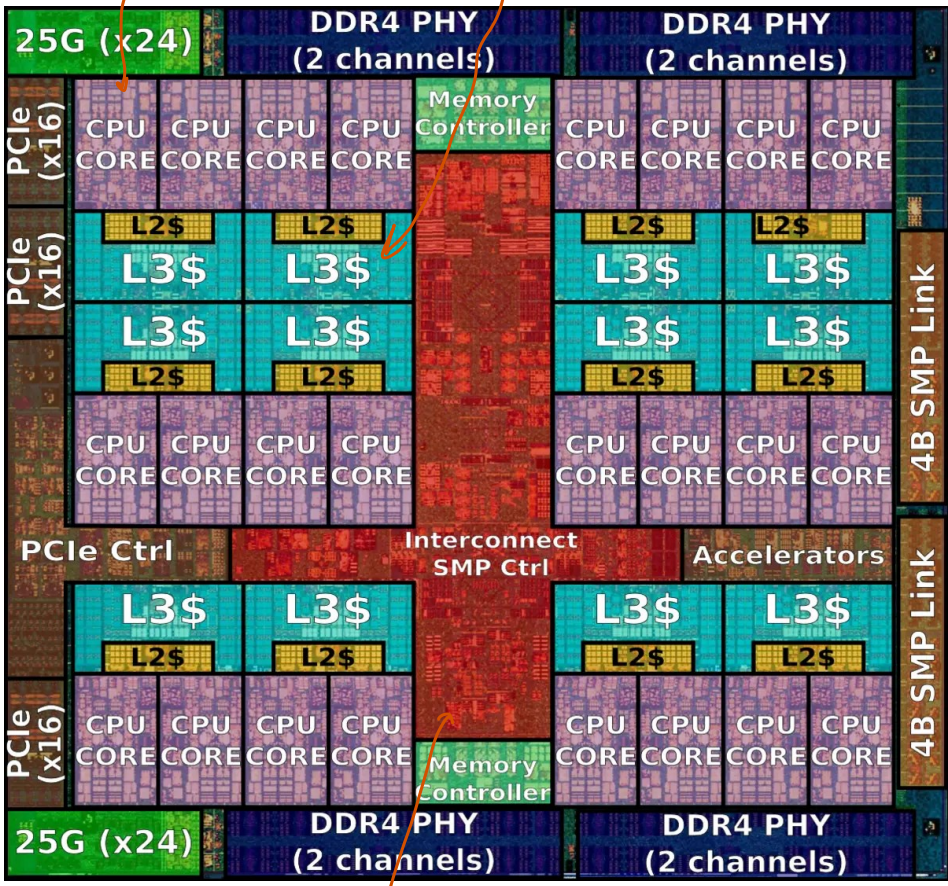
Other trends

- Accelerator processors with more parallelism
  - GPGPU (General Purpose Graphics Processing Unit)
    - Vector and array processing

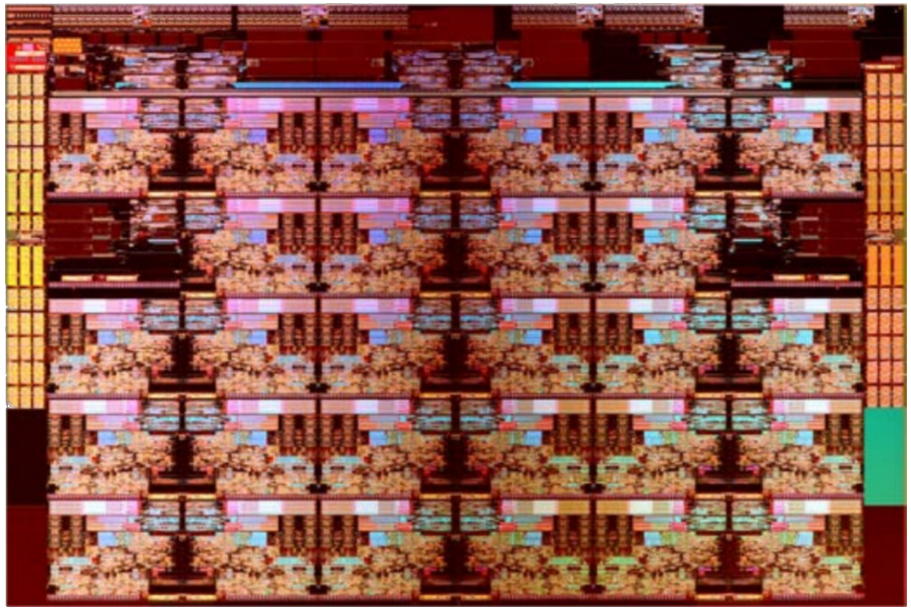
- AI Accelerator (aka Deep Learning Processor, etc.)
  - TPU (Tensor Processing Unit) ทำชุดคำสั่งที่เหมาะกับ workload ด้าน NN
    - Google
  - Optical Neural Network

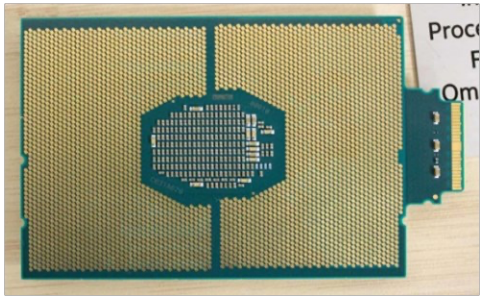
IBM Power9 24-core Processor (2017)





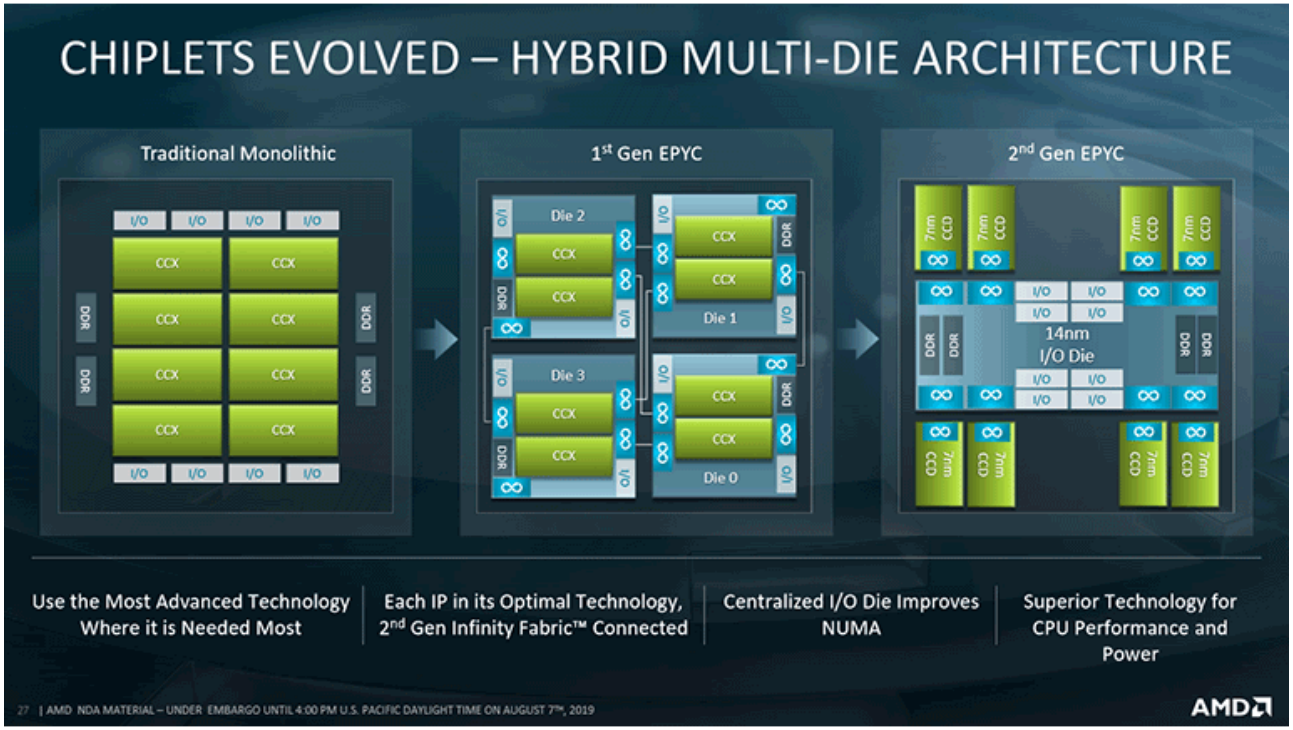
Intel Xeon Platinum 28-core Processor (2017)

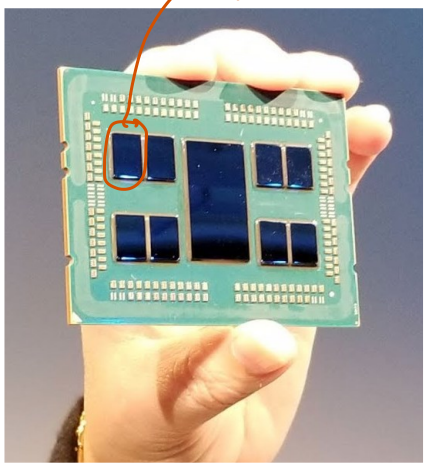




Intel Xeon 8176F, Launch Date Q3'17, $8874

Multi-Die Architecture



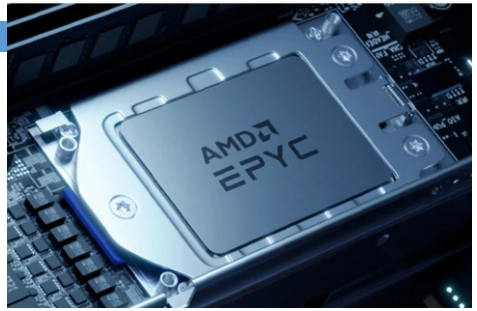


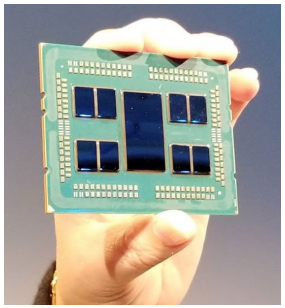
ถ้าหาก core เยอะ ถ้าหาก chip อยู่บนตัวเดียว จะผลิตยาก

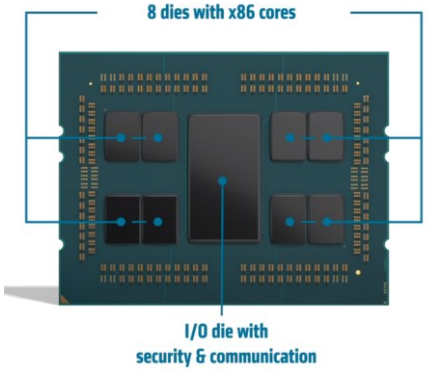
Making a multi-die CPU is more economical than a monolithic-die CPU.

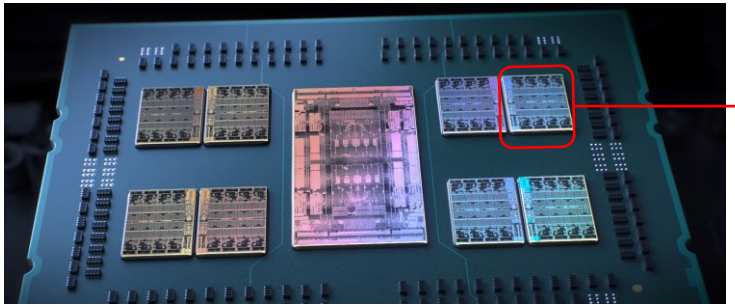
AMD EPYC 7763 Processor (2021)

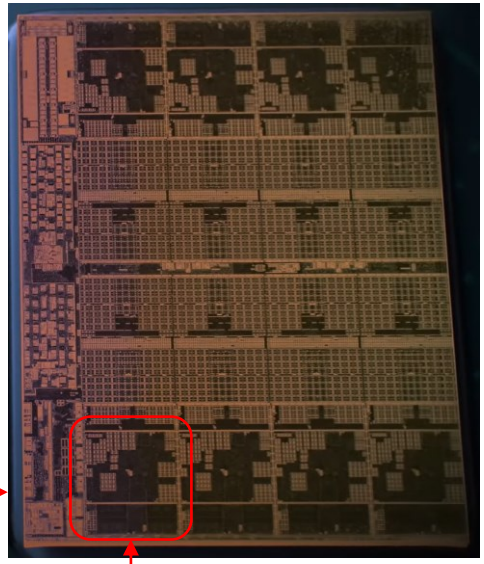
- 64 cores / 128 threads
- Multi-die architecture
- 8 dies, 8 cores per die
- Each core has 32KB L1 cache, and 512 KB L2 cache.
- Each die has 32 MB L3 cache per die → total 256 MB L3 cache
- Transistor size 7nm
- 280 Watt
- Launch 2021, $8000



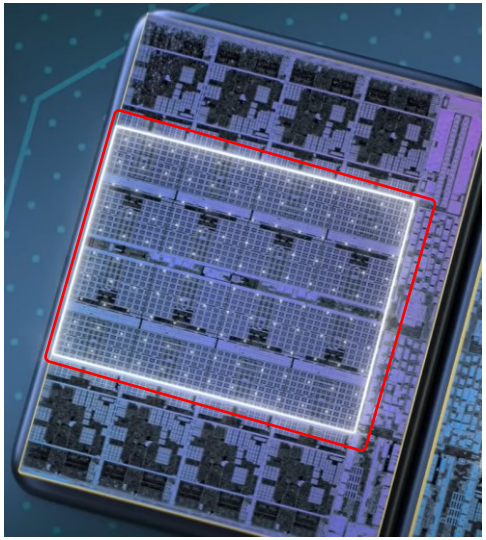


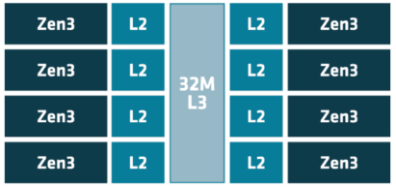






CPU core





Total 256 MB L3-cache

https://www.amd.com/system/files/documents/overview-amd-epyc7003-series-processors-microarchitecture.pdf

Nvidia GeForce RTX 3090 GPU (2020)

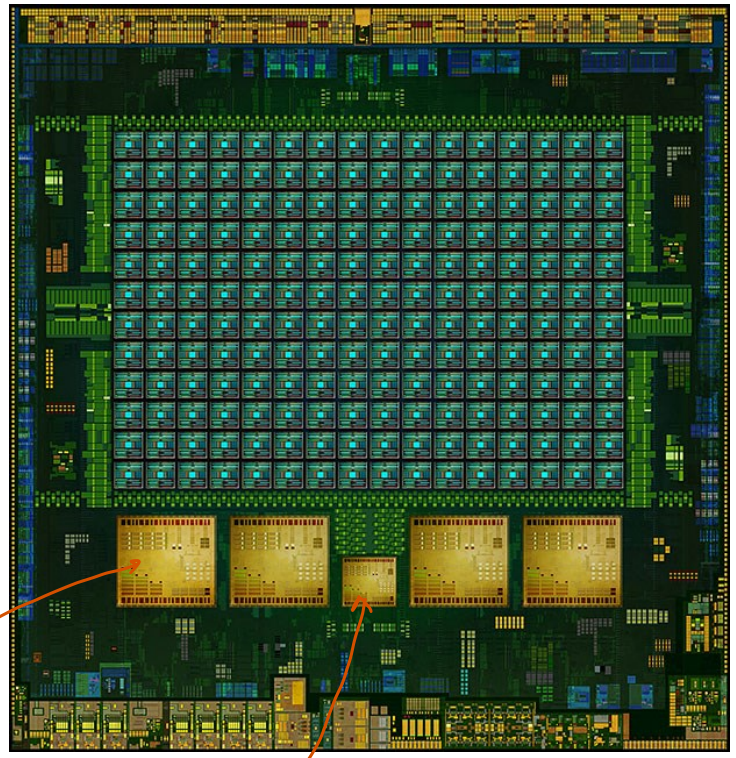
- 10,496 CUDA cores
- 82 SM (streaming multiprocessors)
- 24 GB RAM
- ~36 TFLOPS

Integrated CPU GPU (2014)

สี CPU + GPU ใน chip เดียวกัน

- Nvidia Tegra K1
  - Mobile processor
  - System-on-a-chip: SoC
  - 4-Plus-1CPU cores
  - 192 GPU cores
  - 5W กินไฟ 5 watt

เริ่มสร้างสำหรับ mobile device เพื่อประหยัดพลังงาน
ตัวทำงานทุก core จะเปลืองไฟ



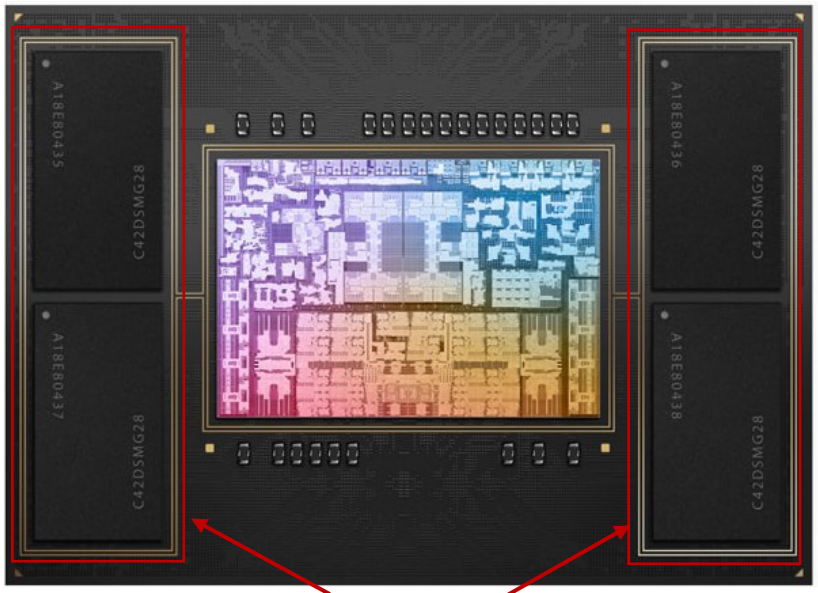
CPU

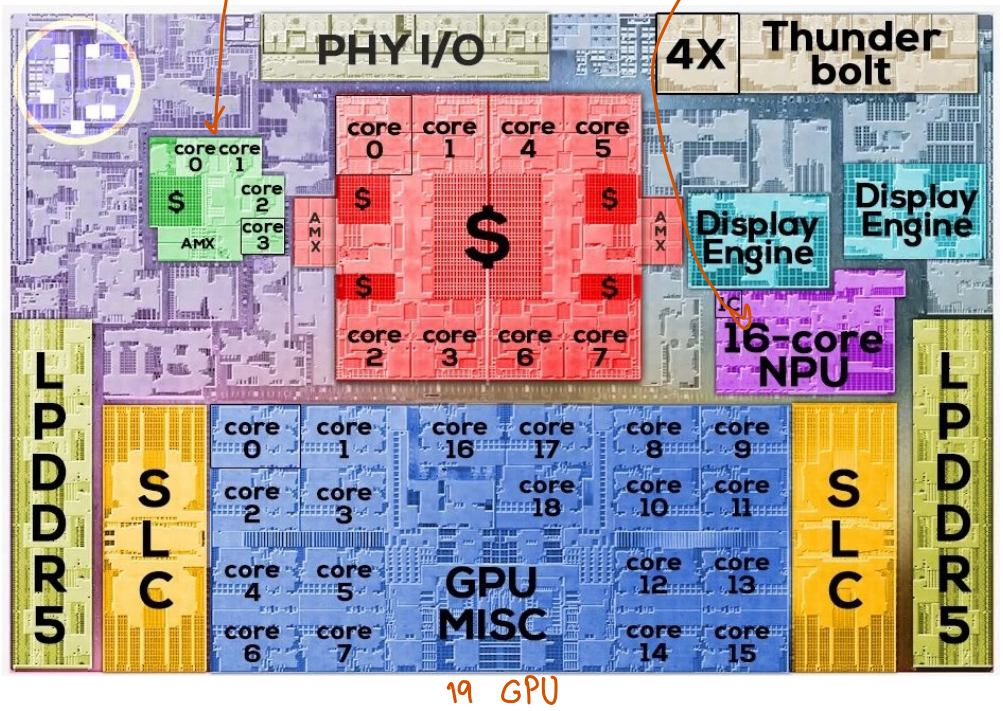
ทำงานตลอดเวลา 1 CPU
แม้ปิดหน้าจอ
ในขณะที่ GPU, CPU ตัวอื่นไม่ทำงาน

System-on-Chip

- Apple M2 Pro
- 12 CPU cores (8 high-performance cores + 4 high-efficiency cores)
  + 19 GPU cores + 16-core Neural Engine
- 32 GB Unified Memory (shared memory between CPU and GPU)

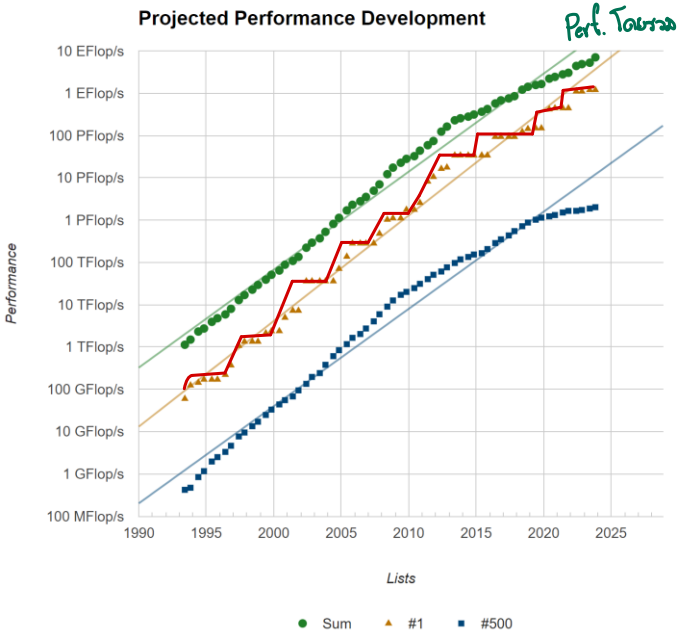
ลด overhead การ transfer ระหว่าง CPU กับ GPU





State of the Art: Top500 Nov 2023

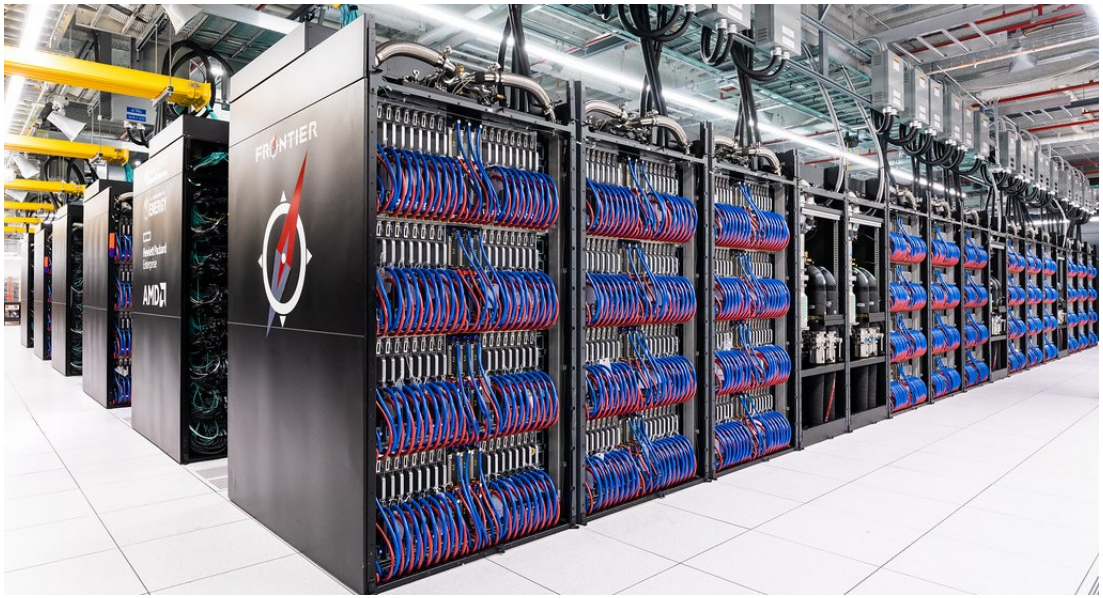
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	4,742,808	585.34	1,059.33	24,687
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Microsoft Azure United States	1,123,200	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
94	**LANTA** - HPE Cray EX235n, AMD EPYC 7713 64C 2GHz, NVIDIA A100 40GB, Slingshot-11, HPE NSTDA Supercomputer Center (ThaiSC) Thailand	87,296	8.15	13.77	310
500	**TX-Green2** - PowerEdge C6420, Xeon Platinum 8260 24C 2.4GHz, 25G Ethernet, ACTION MIT Lincoln Laboratory Supercomputing Center United States	43,200	2.02	53.08	

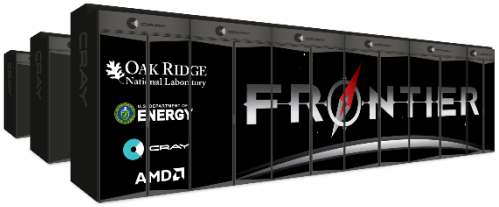


https://top500.org/lists/top500/list/2023/11/

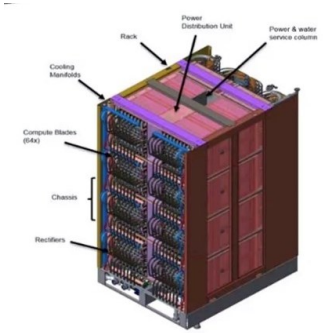
https://top500.org/statistics/perfdevel/

Frontier Supercomputer





system

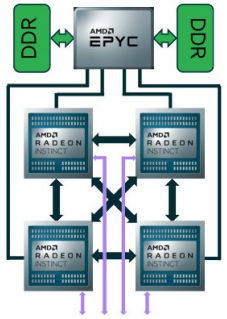


rack

Compute blade

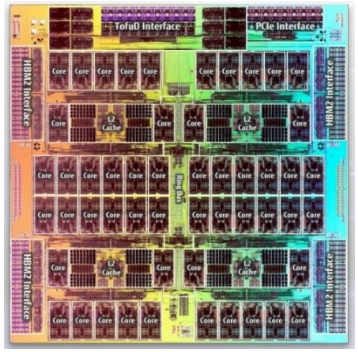
- 2 AMD nodes

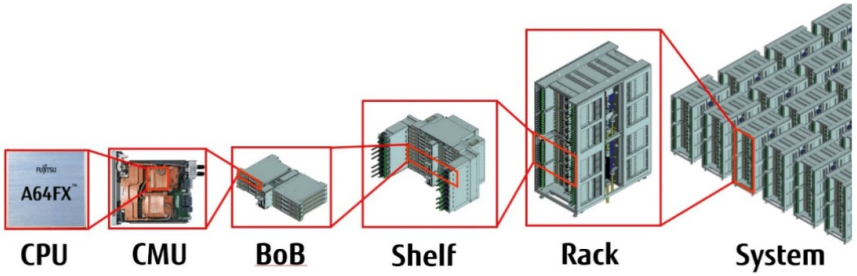
node



CPU + GPU

Fugaku Supercomputer





IEEE SPECTRUM Topics Reports Blogs Multimedia

Tech Talk | Computing | Hardware

29 Jun 2020 | 16:00 GMT

Japan's Fugaku Supercomputer Completes First-Ever Sweep of High-Performance Benchmarks

Fugaku, based on the Arm architecture, races ahead in processing speed, running real-world applications, and in a new benchmark: evaluating calculations used in artificial intelligence

By **John Boyd**

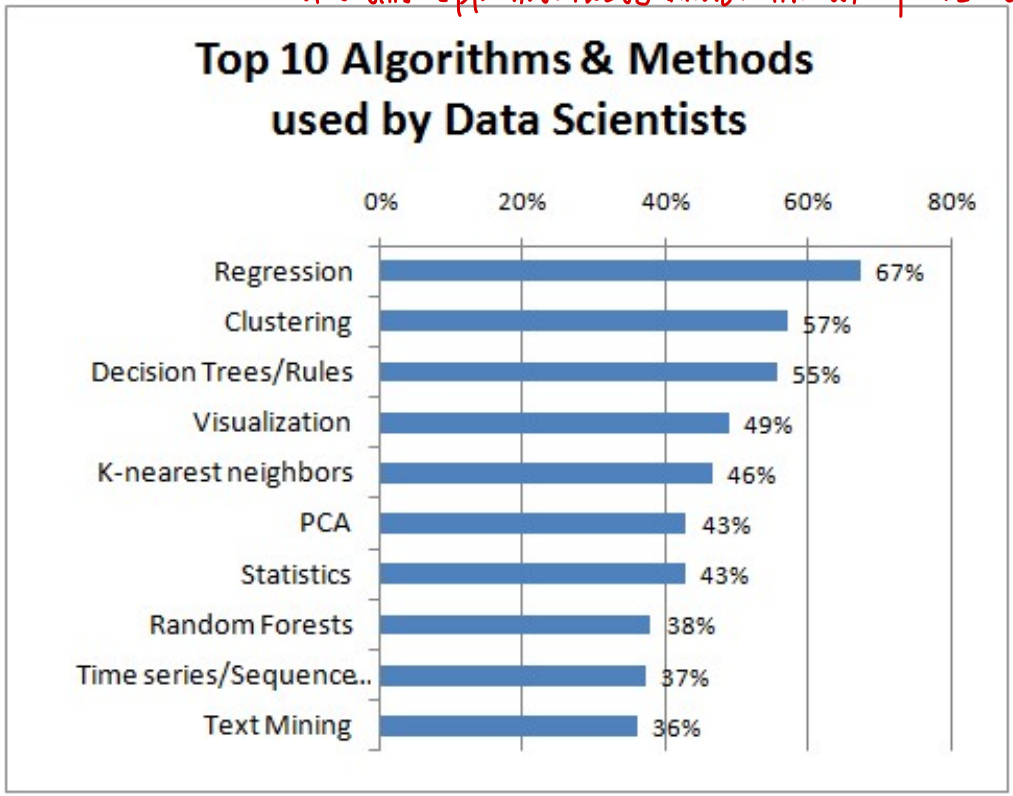
https://spectrum.ieee.org/tech-talk/computing/hardware/japans-fugaku-supercomputer-is-first-in-the-world-to-simultaneously-top-all-high-performance-benchmarks

System Architecture ในยุค Software

Application Requirements

Requirements of Data Science Applications

พิจารณา ก. ต้องการ app ที่ใช้งานบ่อย แล้วเอา HW มา optimize ออกแบบ



https://www.kdnuggets.com/2016/09/poll-algorithms-used-data-scientists.html

Which methods/algorithms you used in the past 12 months for an actual Data Science-related application?

Common Data Analysis Methods

- Regression
  - least squares method technique ย่อยๆ ที่ใช้บ่อยๆ
- Clustering
  - k-means clustering, hierarchical clustering
- Classification
  - decision tree, k-nearest neighbors, support vector machine, neural network
- Recommendation
  - collaborative filtering
- Text analysis
  - sentiment analysis, translation
- Network Analysis
  - centrality, community detection

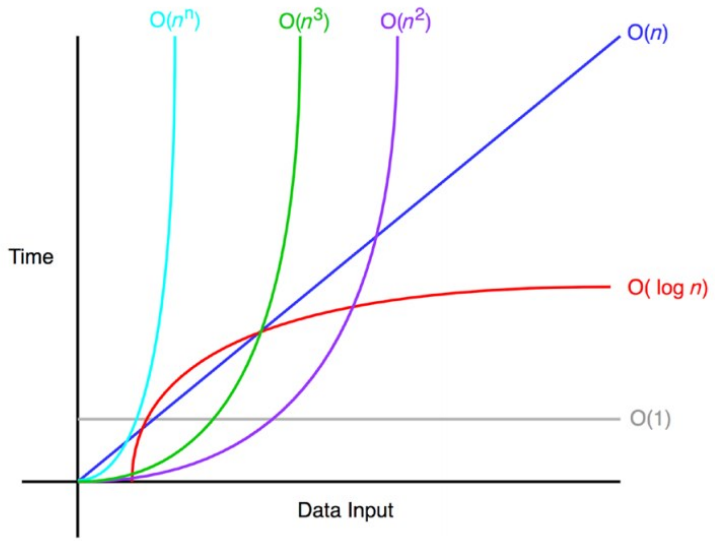
Basic data structures and algorithms

All analysis methods are constructed from basic data structures and algorithms: ทุก method ถูกสร้างโดยโครงสร้างข้อมูลพื้นฐานและอัลกอริทึม

- Array
  - sort, search, average
- Matrix
  - dot product, matrix multiplication
- Tree
  - search
- Graph
  - shortest path

Computational Complexity

แทนความซับซ้อนของปัญหา, ความยากในการคำนวณ



- $O(1)$
  - Read an element in an array
- $O(\log n)$ ข้อมูลเพิ่มขึ้น เวลาไม่ได้เพิ่มขึ้นมาก
  - Binary search
- $O(n)$
  - Average, vector-vector addition, vector-vector dot product
- $O(n \log n)$
  - QuickSort, merge sort
- $O(n^2)$
  - Insertion Sort, matrix-vector multiplication
- $O(n^3)$
  - All-pair shortest path, matrix-matrix multiplication

Computing Intensive vs. Memory Intensive Problems

$$O(n^3) \text{ vs. } O(N)$$

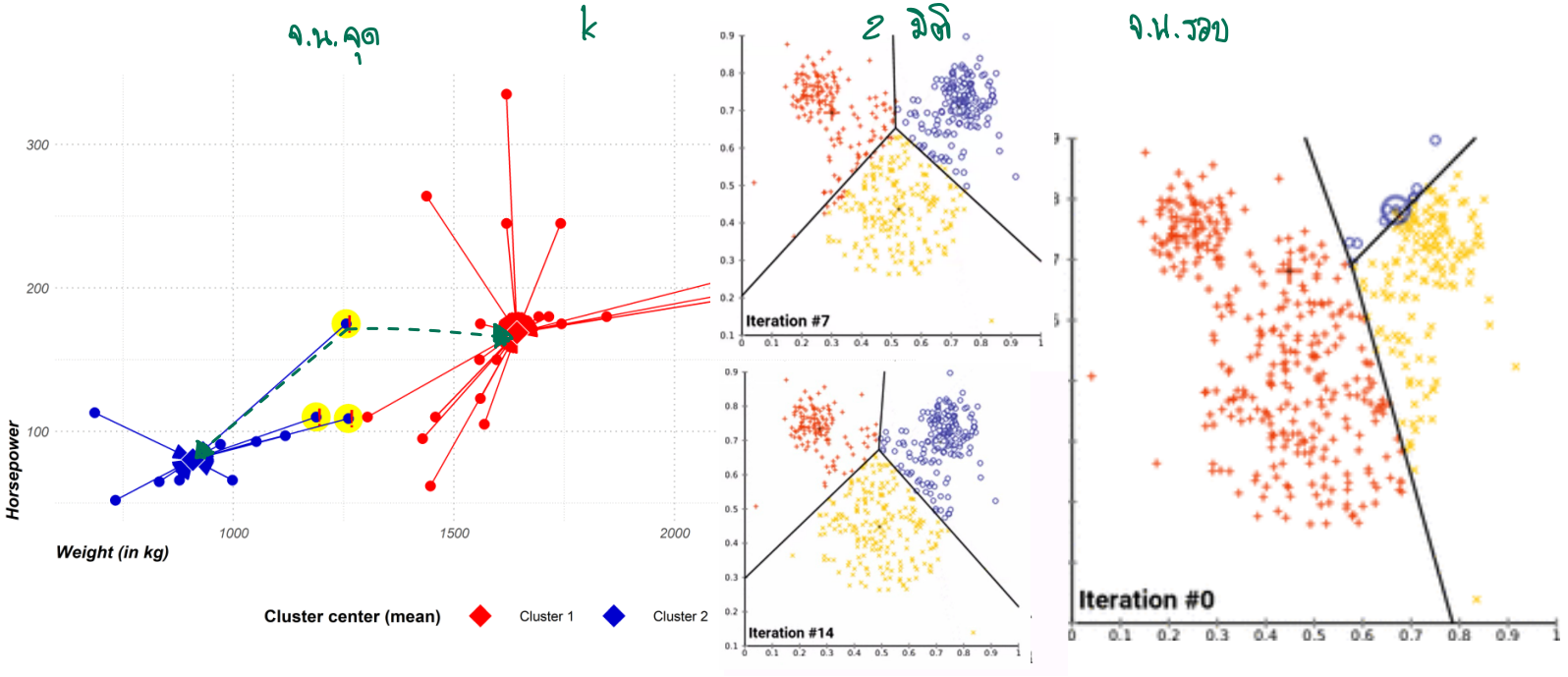
ความซับซ้อนต้องพิจารณาที่ขนาดของข้อมูล ดังนั้น $O(N)$ มีโอกาสใหญ่กว่า $O(n^3)$ ได้

ปัญหา big data

A data science problem can be both computing intensive and memory intensive.

Example: K-Means Clustering

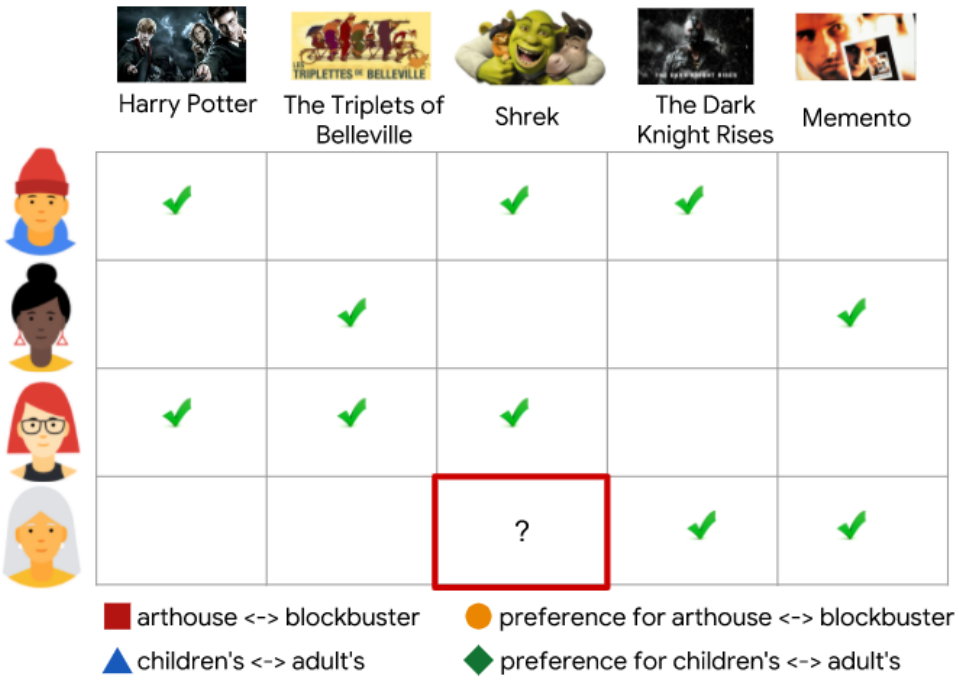
- An algorithm to partition data into clusters (groups of similar data points)
- The difference, or distance, between a pair of data points can be calculated from their properties.
- Each cluster has a centroid point representing average of each property of its members.
- The algorithm repeats these two steps until the result does not change:
  - Assign each point to the cluster with the nearest centroid.
  - Calculate centroids of each cluster.
- Complexity → O(no.points * no.clusters * no.dimensions * no.iterations)



Example: Recommendation Systems based on Matrix Factorization

- Feedback matrix contain users' respond to movies. Most cells are empty (sparse matrix).
- Goal: predict users' feedback on movies. In other words, fill the empty cells in *feedback*.

Recommendation คือปัญหาการเติมค่าในช่องว่าง

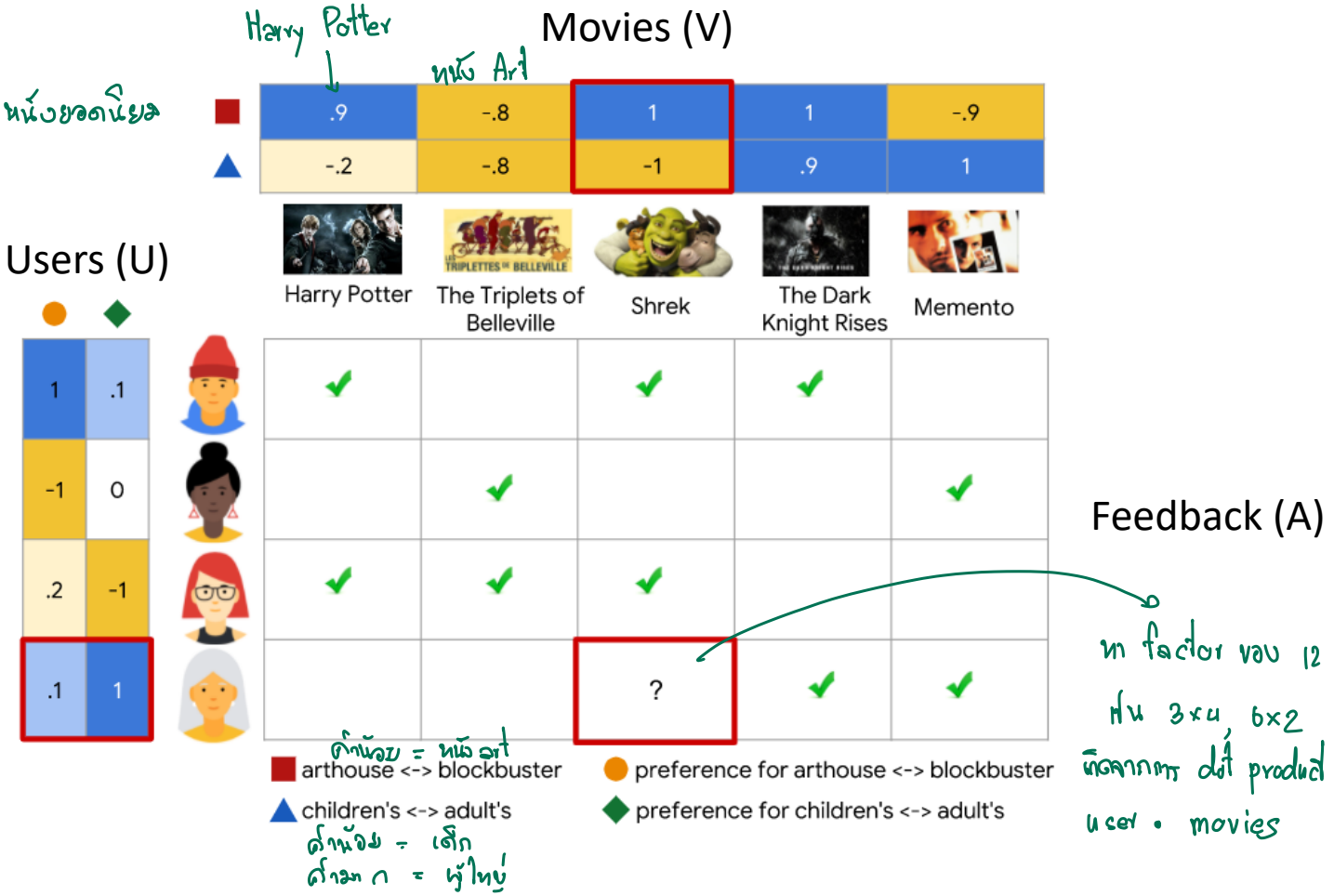


Feedback

https://developers.google.com/machine-learning/recommendation/collaborative/basics

Example: Recommendation Systems based on Matrix Factorization

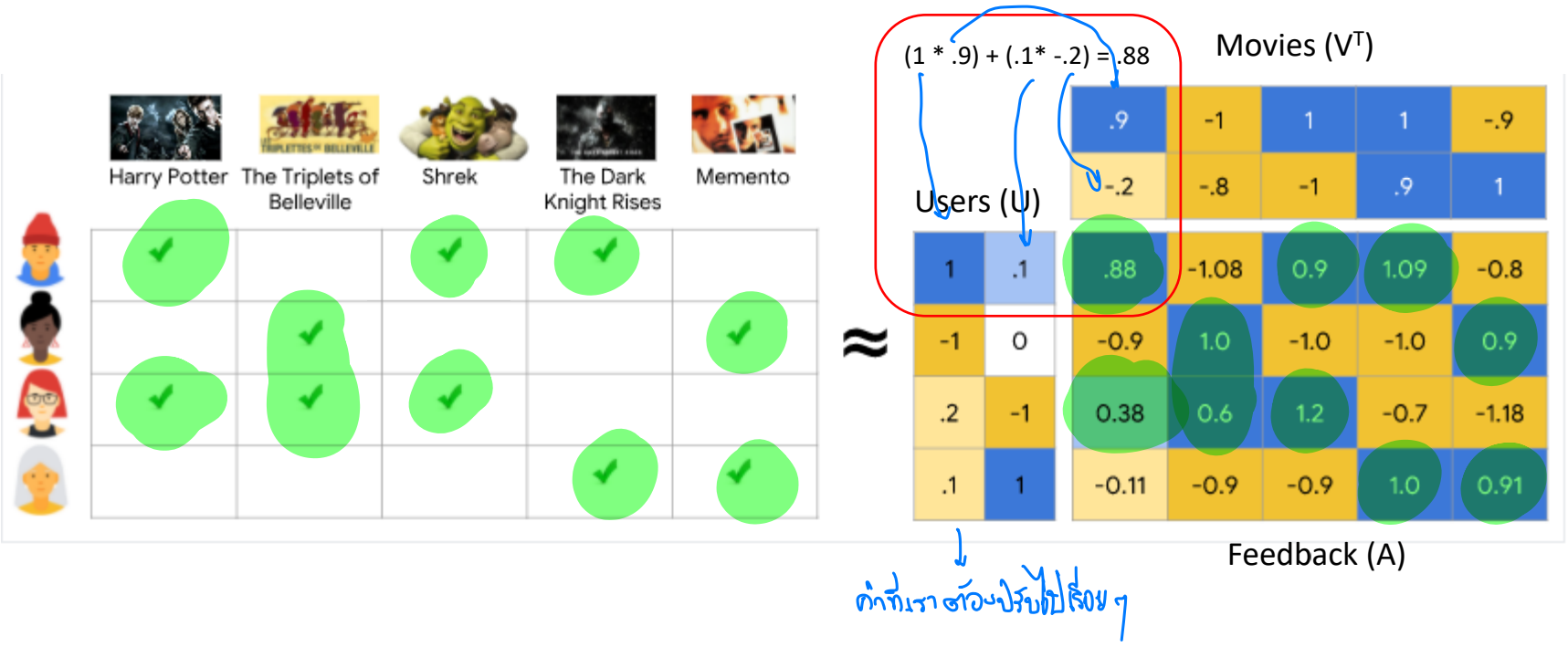
- *Movies* matrix represents characteristics of each movie.
- Users matrix represents preference on each characteristic.



https://developers.google.com/machine-learning/recommendation/collaborative/basics

Example: Recommendation Systems based on Matrix Factorization

- Every cell in the *rating* matrix can be calculated from the dot product of *user* and *movie* matrices.
- Matrix Factorization: Find U and V such that $A \approx UV^T$
- Iteratively adjust the value in *user* and *movie* matrices to minimize rating error (objective function).
- Complexity → O(no.rating * no.features * no.iteration)



https://developers.google.com/machine-learning/recommendation/collaborative/matrix

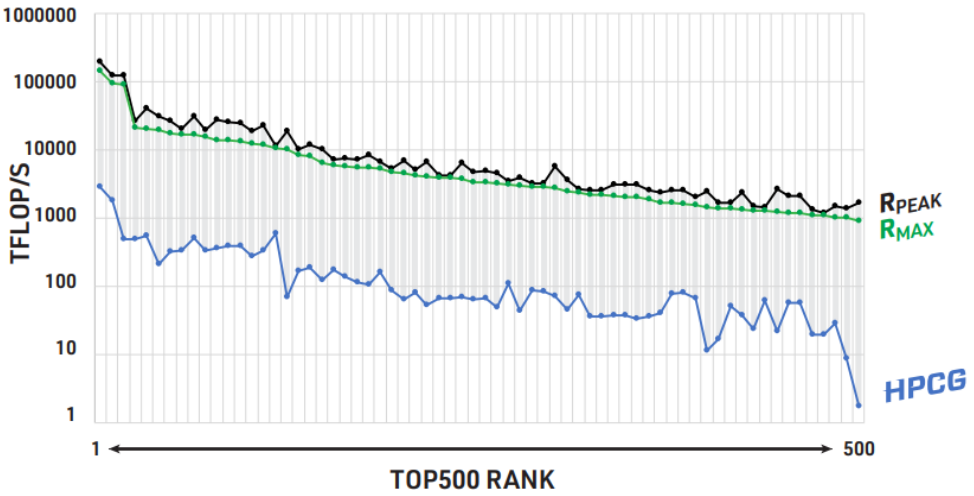
HPCG Benchmark (used by Top500)

PRECONDITIONED CONJUGATE GRADIENT SOLVER

Code วัด benchmark โดยพยายามสร้างเครื่องที่ทำ dot product เร็วๆ

$p_0 \leftarrow x_0, r_0 \leftarrow b - Ap_0$
for $i = 1, 2,$ to max_iterations **do**
  $z_i \leftarrow M^{-1} r_{i-1}$
  if $i = 1$ **then**
    $p_i \leftarrow z_i$
    $\alpha_i \leftarrow \text{dot\_prod}(r_{i-1}, z_i)$ ← dot product ผมว่ามีการวัดโดยการคำนวน matrix
  else
    $\alpha_i \leftarrow \text{dot\_prod}(r_{i-1}, z_i)$
    $\beta_i \leftarrow \alpha_i / \alpha_{i-1}$
    $p_i \leftarrow \beta_i p_{i-1} + z_i$
  end if
  $\alpha_i \leftarrow \text{dot\_prod}(r_{i-1}, z_i) / \text{dot\_prod}(p_i, Ap_i)$
  $x_{i+1} \leftarrow x_i + \alpha_i p_i$
  $r_i \leftarrow r_{i-1} - \alpha_i Ap_i$
  if $\|r_i\|_2 <$ tolerance **then**
    STOP
  end if
end for

- The benchmark is designed to model data access patterns of real-world applications such as sparse matrix calculations.
- It tests the limitations of memory system and interconnect better than TOP500's Linpack benchmark.



http://hpcg-benchmark.org/
https://www.icl.utk.edu/files/print/2019/hpcg-sc19.pdf

จัดอนาคต รพ มีปัญหาขาดทุน demand เยอะขึ้น
HW เร็วขึ้น

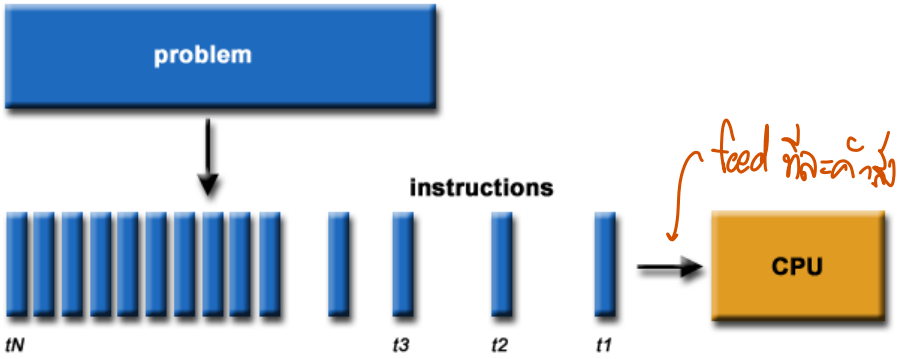
Scalability & Parallelism

Parallelism

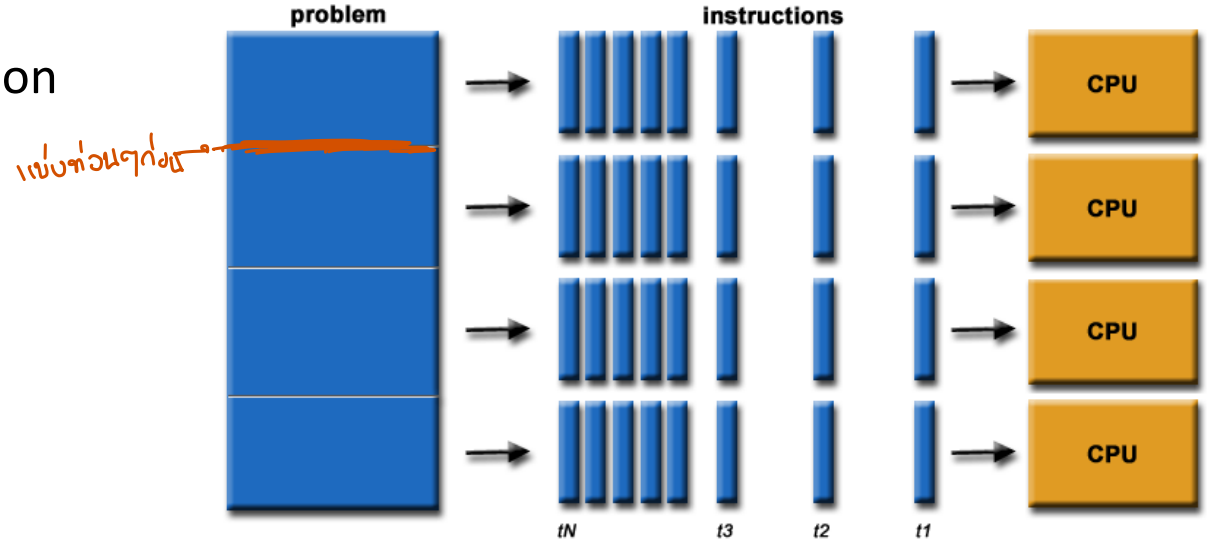
- Parallelism is the *potential to divide a task into sub-tasks and perform them in parallel*.
- Parallel computing involves how to *exploit parallelism in software with parallelism in hardware (parallel architecture)*.

Serial vs. Parallel Computation

Serial Computation



Parallel Computation



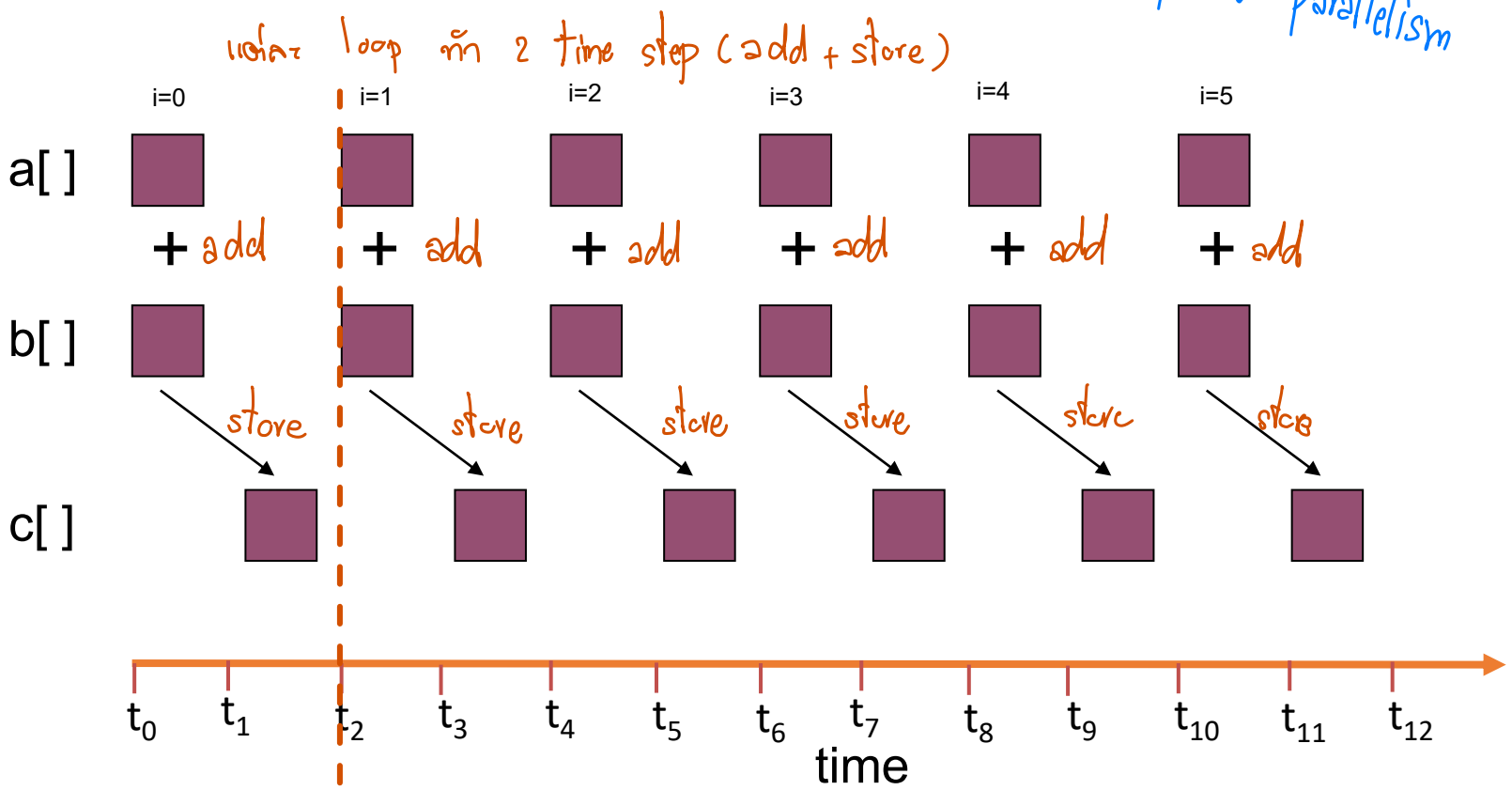
Sequential Computation

ถ้าสลับลำดับได้ แสดงว่าทำ Parallelism ได้ เช่น ทำ i=5 ก่อน i=4 ได้

```
for (i=0; i<6; i++)
    c[i] = a[i] + b[i];
```

- Primitive operations (i.e. load, add, store) are performed sequentially.

loop-level parallelism



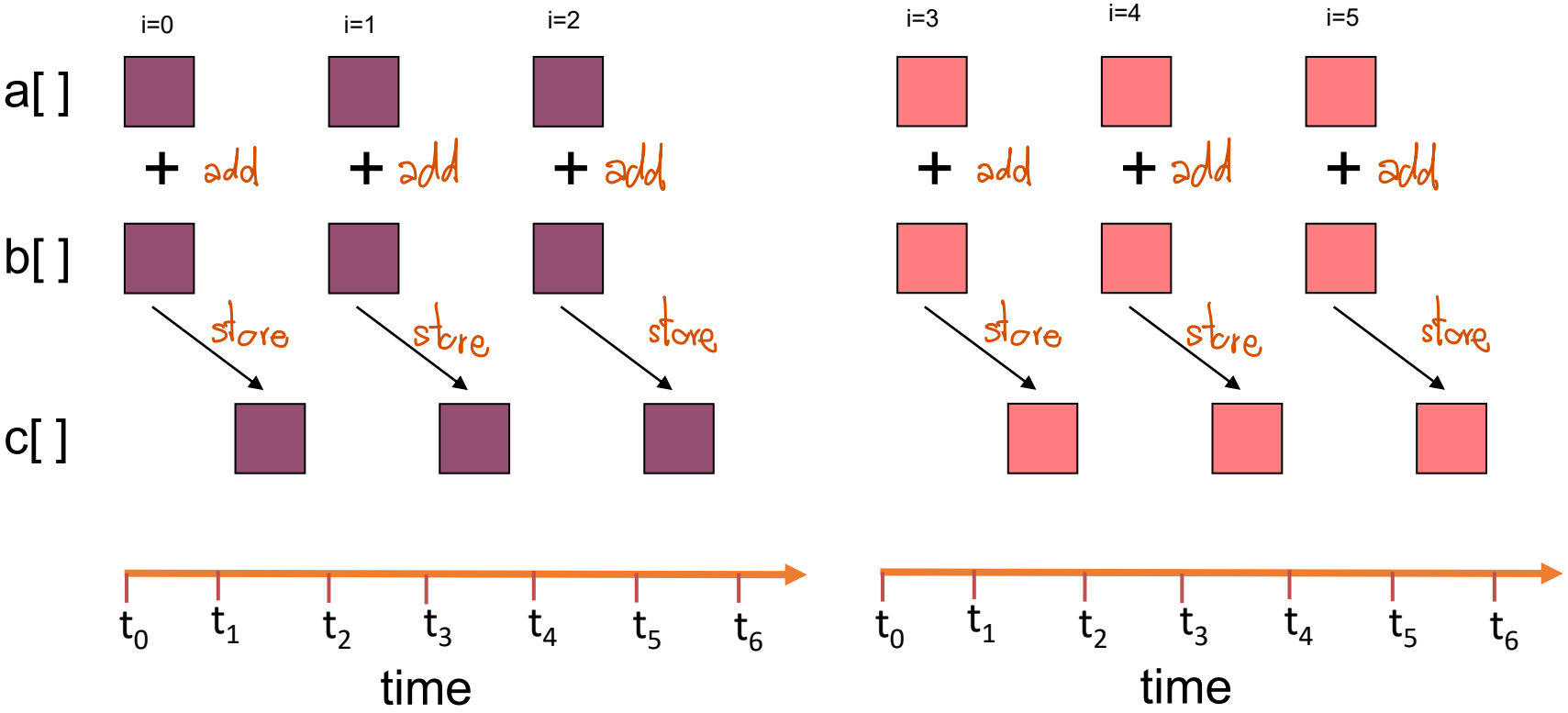
2 cores

Parallel computation with 2 processing elements

- A task is divided and performed in parallel. แบ่งงานข้อมูล

```
for (i=0; i<3; i++)
    c[i] = a[i] + b[i];
```

```
for (i=3; i<6; i++)
    c[i] = a[i] + b[i];
```



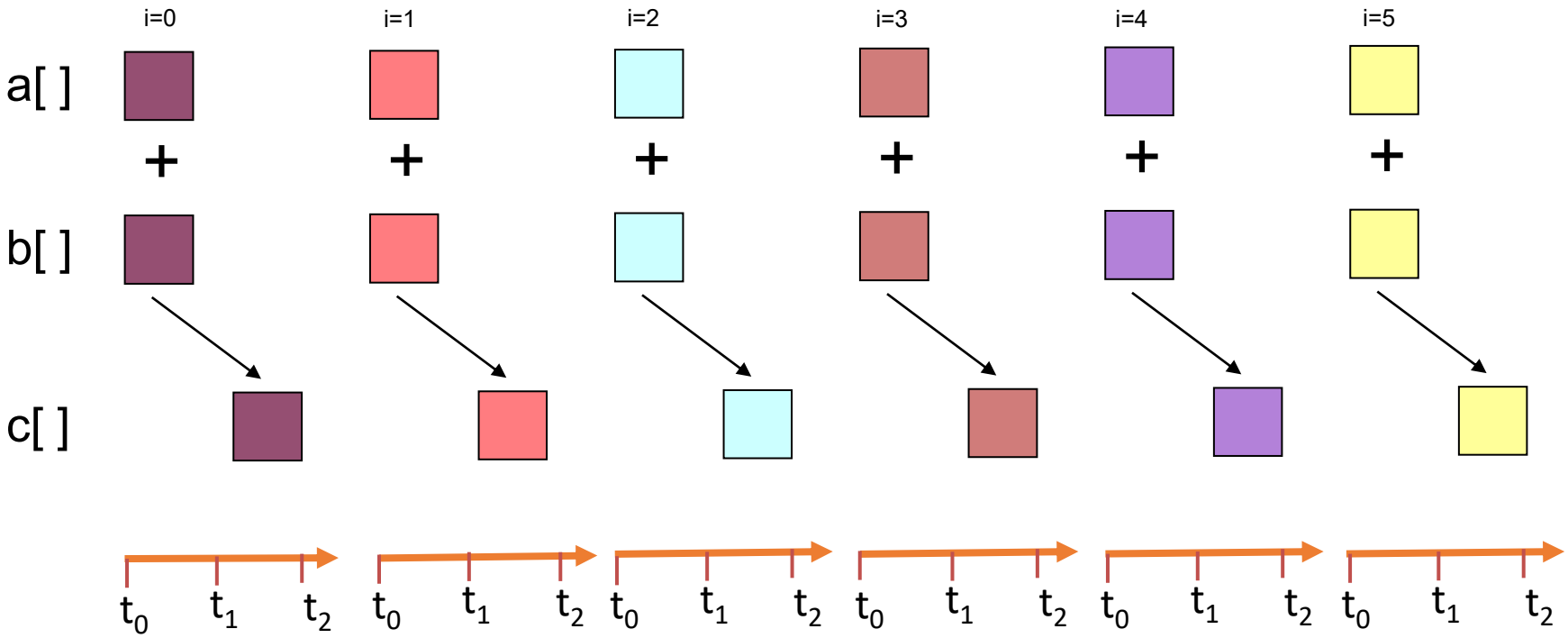
Parallel computation with 6 processing elements

6 cores

- Exploiting all parallelism in software.

ใช้ parallelism ทั้งหมดที่มี

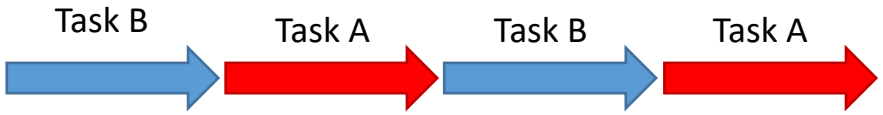
```
c[i] = a[i] + b[i];
```



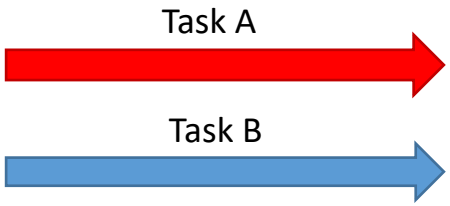
Parallelism vs. Concurrency

- Concurrency is when multiple tasks logically (actually or appear to) run simultaneously. คือการทำงานที่งานจะไม่ได้ทำงานจริงๆ
- Parallelism is when multiple tasks physically (actually) run simultaneously.
- Concurrency is more general than parallelism.
  - Parallel processes always run concurrently, but concurrent processes do not always run in parallel.

Concurrency but not parallelism



Concurrency and parallelism



Scalability

- Scalability is the ability of a system or process to handle growing amount of work or its ability to be enlarged to accommodate that growth.

  ความสามารถในการเติบโตของระบบ ปัญหามากขึ้น arch สามารถรองรับได้?

- Scalability of a parallel system is relative to *the ability to exploit parallelism*.

- Scalable system
  - The performance improves after adding hardware, proportionally to the added capacity.
  - Support wide range of system sizes
  - Use parallel computing or distributed systems techniques

Levels of parallelism in software

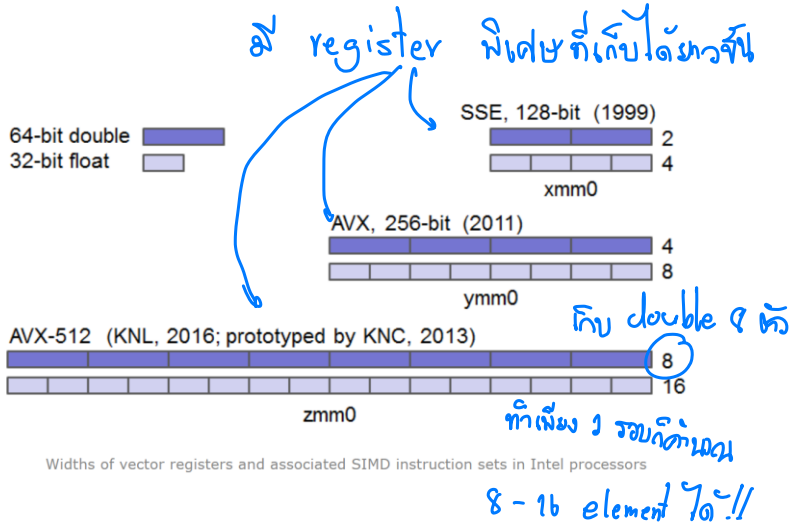
การ match parallel ใน SW กับ HW

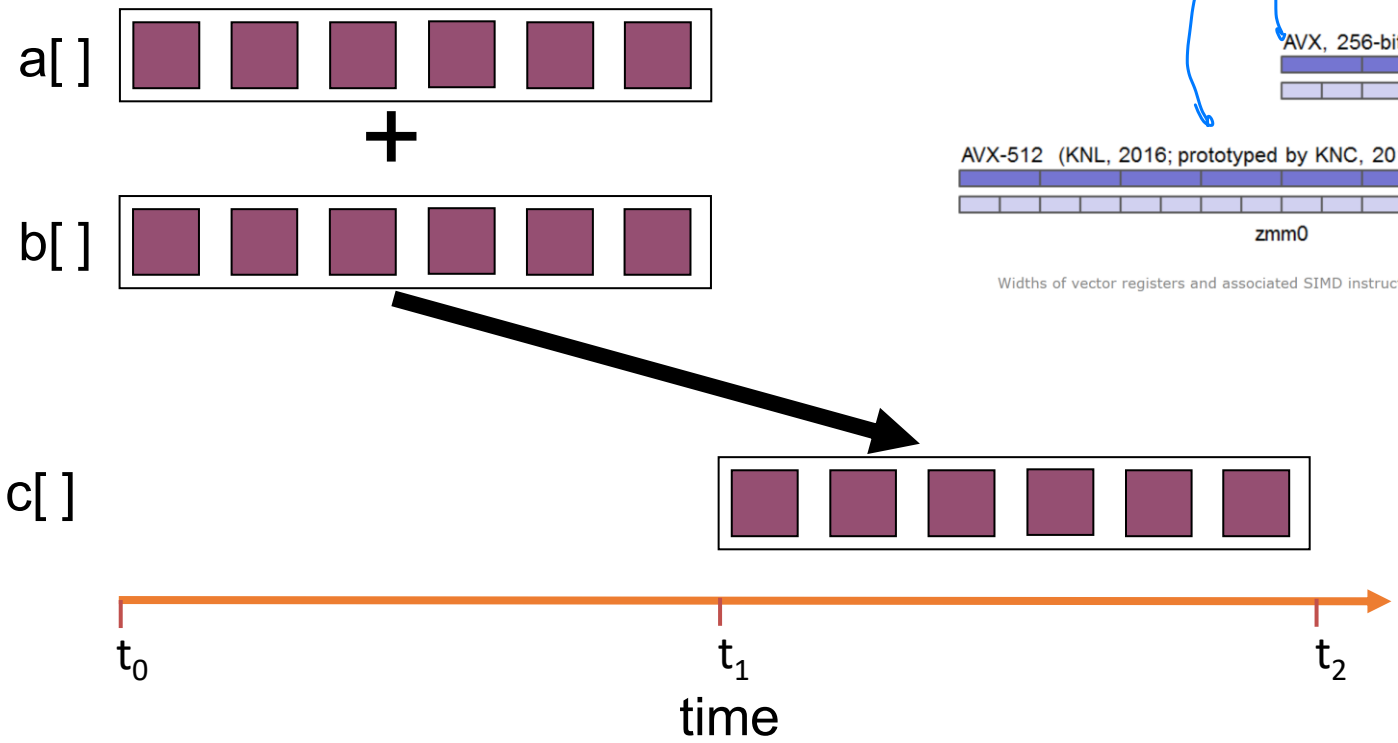
- Instruction-level parallelism ทำ oper การบวกพร้อมกันได้ a+b ก่อน x+y ได้ เช่น C[i] = a[i] + b[i] + x[i] + y[i]
  - Executing multiple CPU instructions in parallel
- Loop-level parallelism
  - Executing individual iterations of a loop in parallel
- Thread-level
  - Executing multiple threads in parallel
- Process-level ใหญ่กว่าระดับ Thread
  - Executing multiple processes in parallel

Run ใน hadoop cluster หลายเครื่องทำพร้อมกัน

Vector Processing

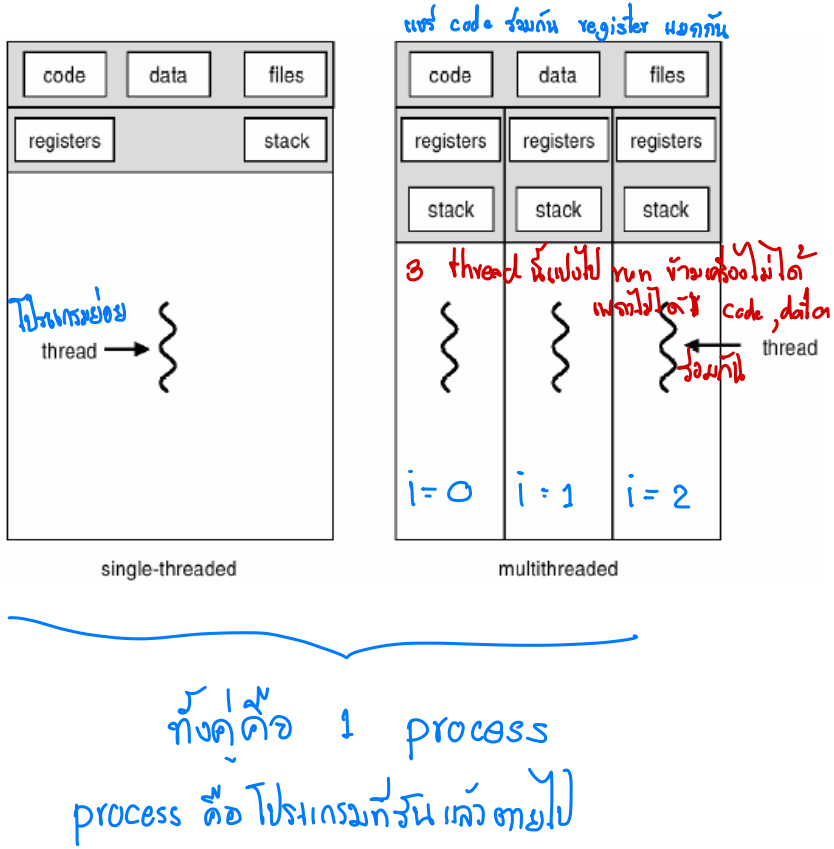
- Each vector instruction performs on all elements in vector operands at once.
- A vector register has limited size (e.g. 32, 64 elements)



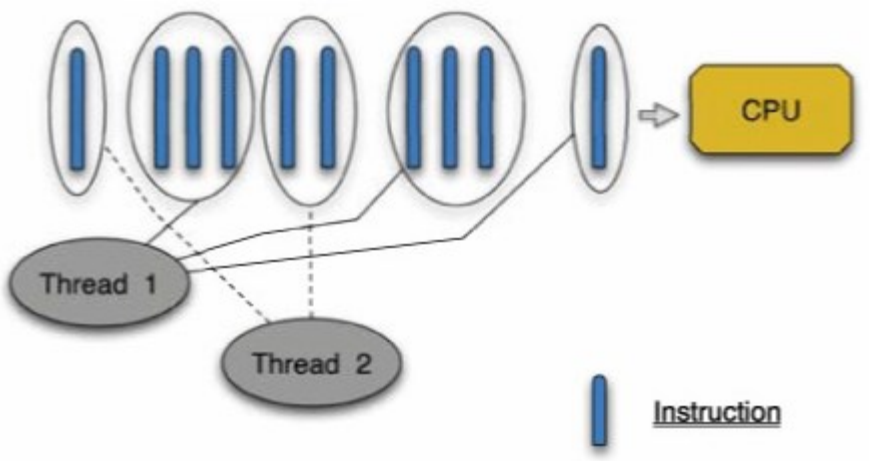


Multi-thread Processing Model

- A process is a running instance of a program
- A process can have multiple threads.
- Processes have separate memory spaces.
- Threads in the same program share memory space.

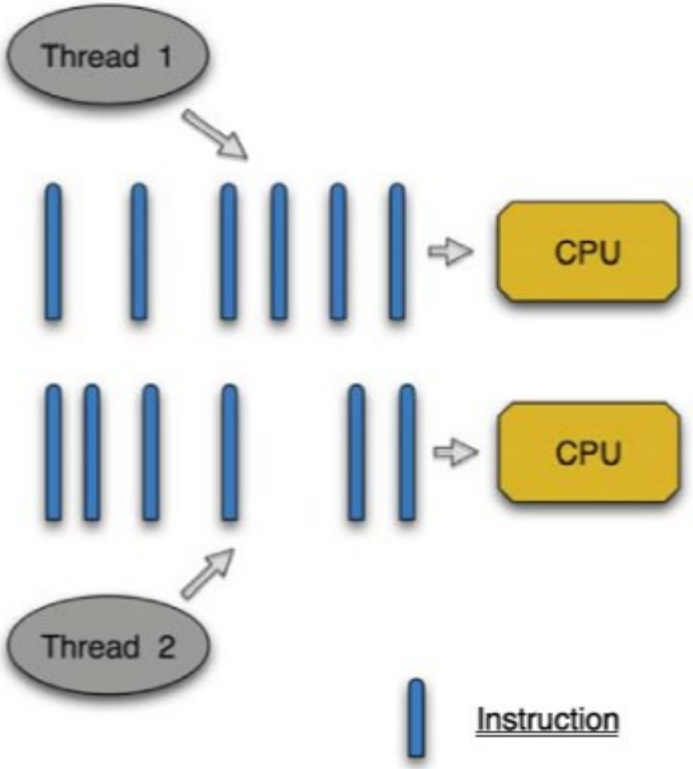


Threads



ไม่ได้เร็วขึ้น แต่ทำงานพร้อมกัน

Multiple threads running concurrently, but not in parallel, on one CPU core (no performance gain)



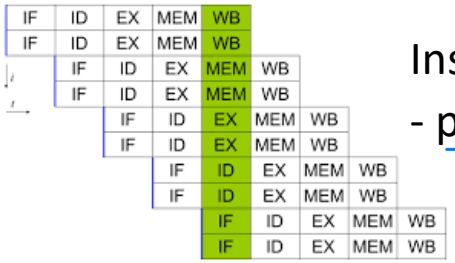
มี CPU หลายตัว ก็ทำงาน parallel จริงๆ

Multiple threads running in parallel on separate CPU cores

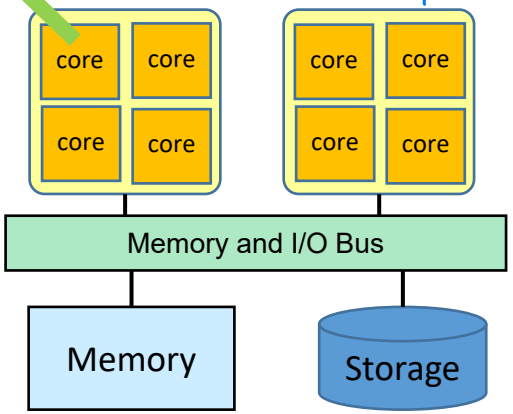
Level of parallelism in hardware

งาน data science มักจะ run ผ่าน high-level language (library)
มันจะพยายามทำ ILP โดยอัตโนมัติ เราแค่เลือก lib ให้ถูกและ optimize

Instruction Level Parallelism (ILP) (อยู่ใน HW)
- pipelining, superscalar, vector processing

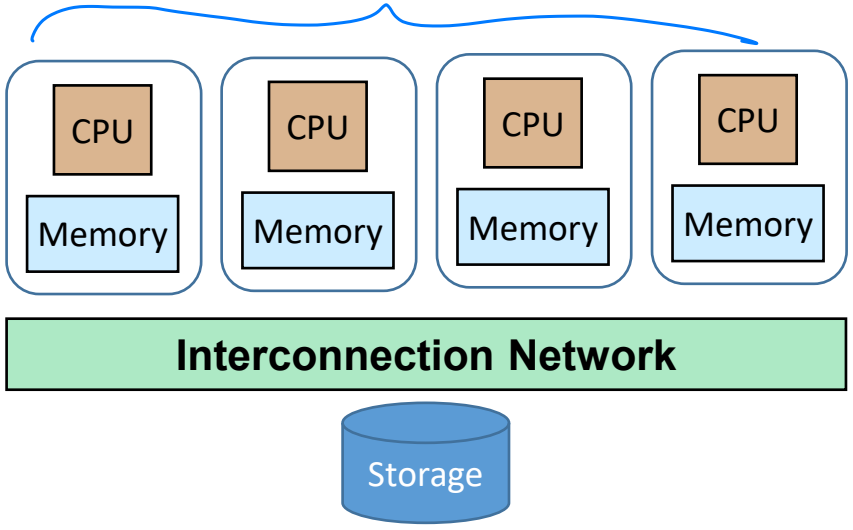


เสมือน run
หลายคำสั่งพร้อมกัน
แต่มีการแบ่ง execute ทีละส่วนๆ



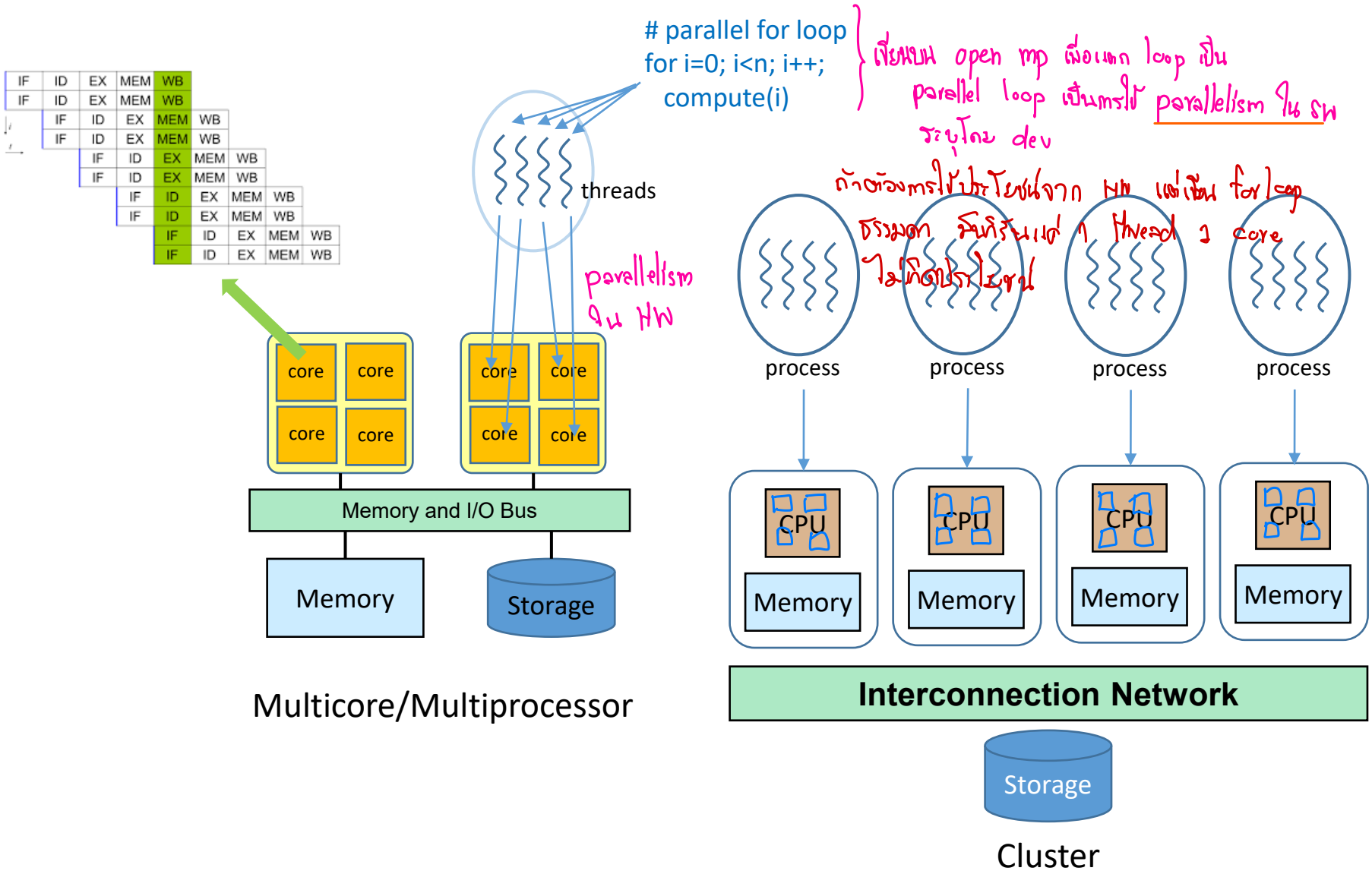
Multicore/Multiprocessor

Multi Processor

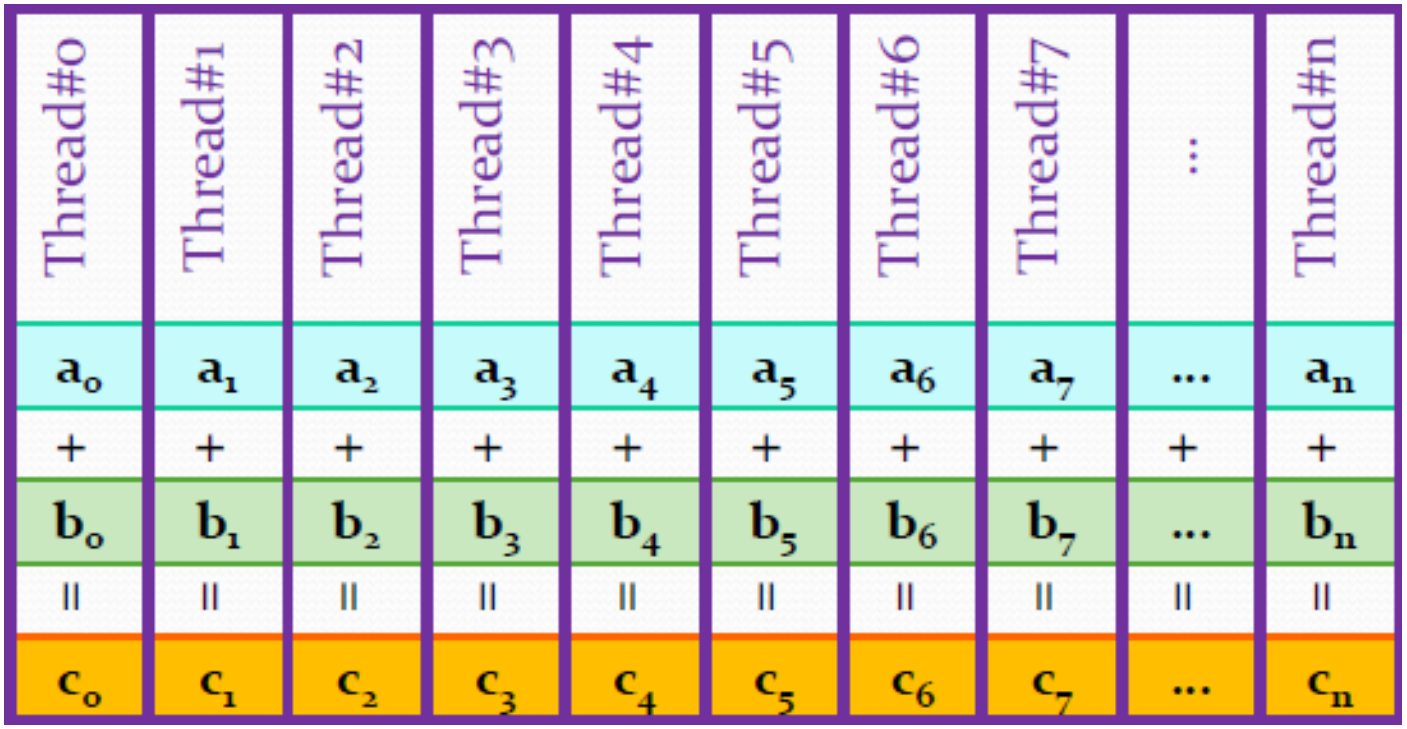


Cluster

Exploiting parallelism in software with parallel architecture



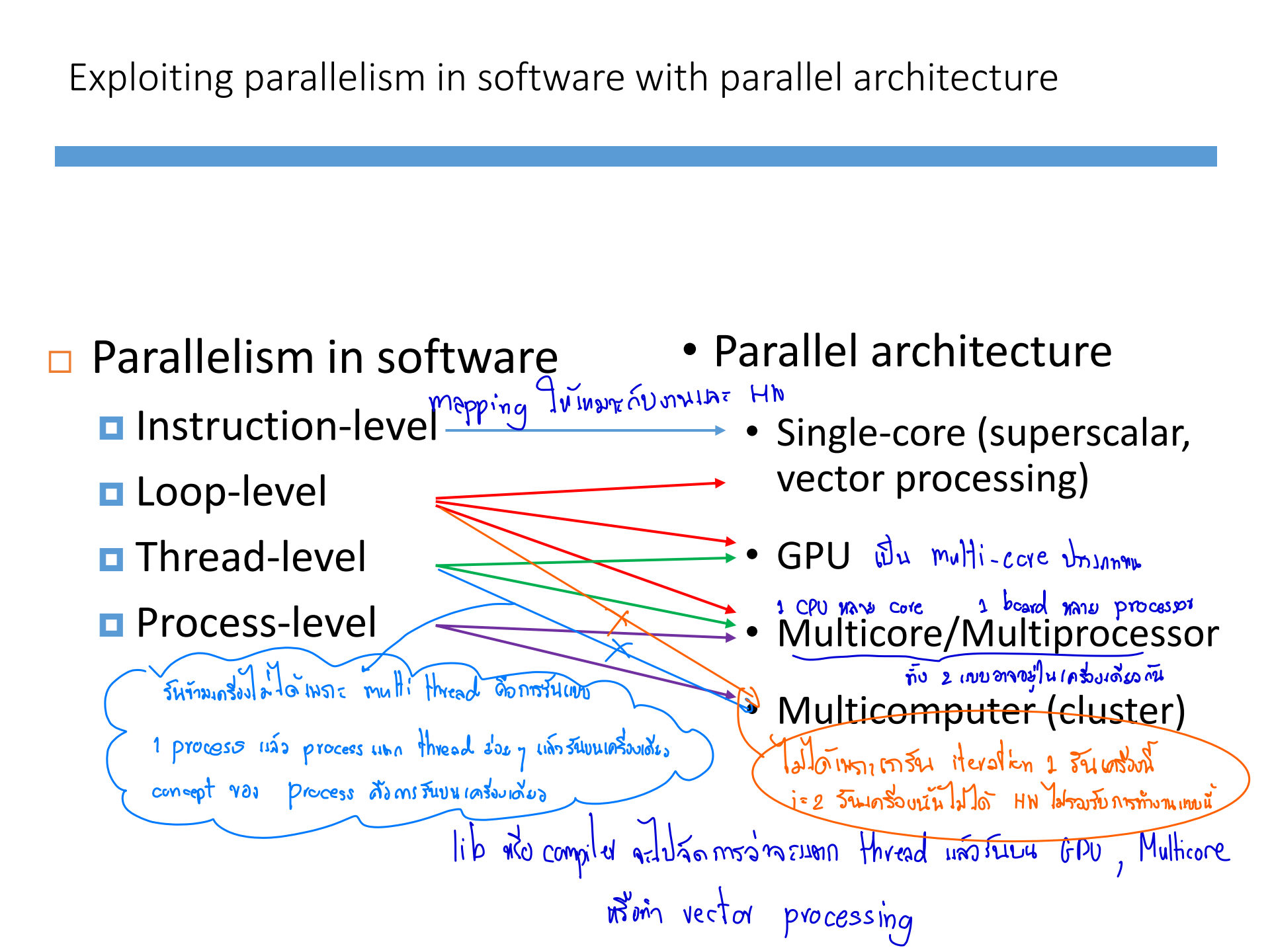
Data-Parallelism and Multithreading



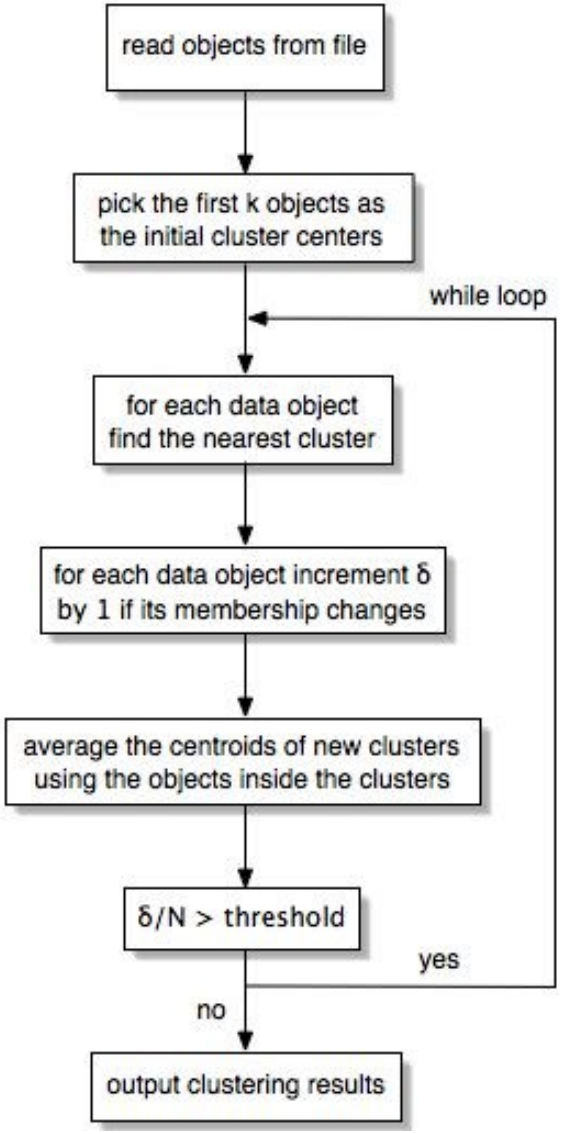
Exploiting parallelism in software with parallel architecture

- Parallelism in software
  - Instruction-level
  - Loop-level
  - Thread-level
  - Process-level

- Parallel architecture
  - Single-core (superscalar, vector processing)
  - GPU
  - Multicore/Multiprocessor
  - Multicomputer (cluster)



Sequential K-Means Clustering

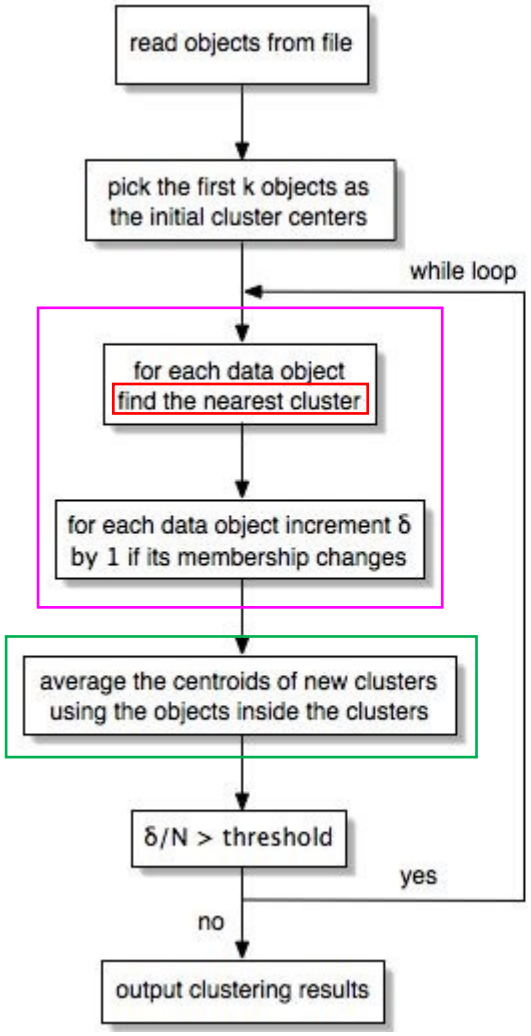


N: *number of data objects*
K: *number of clusters*

objects[N]: *array of data objects*
clusters[K]: *array of cluster centers*
membership[N]: *array of object memberships*

```
kmeans_clustering( )
1   while δ/N > threshold
2     δ ← 0
3     for i ← 0 to N-1          loop ตามจำนวน object
4       for j ← 0 to K-1        loop ตามจำนวน cluster
5         distance ← | objects[i] - clusters[j] |
6         if distance < dmin
7           dmin ← distance
8           n ← j
9       if membership[i] ≠ n
10        δ ← δ + 1
11        membership[i] ← n
12      new_clusters[n] ← new_clusters[n] + objects[i]
13      new_cluster_size[n] ← new_cluster_size[n] + 1
14    for j ← 0 to K-1          คำนวณหา centroid ใหม่ของ cluster
15      clusters[j][*] ← new_clusters[j][*] / new_cluster_size[j]
16      new_clusters[j][*] ← 0
17      new_cluster_size[j] ← 0
```

Parallel K-Means Clustering



N: *number of data objects*
K: *number of clusters*

objects[N]: *array of data objects*
clusters[K]: *array of cluster centers*
membership[N]: *array of object memberships*

ทำ loop เพื่อพิจารณา parallelism ว่าทำ parallel ได้หรือไม่ ?

```
kmeans_clustering( )
1   while δ/N > threshold
2     δ ← 0
3     for i ← 0 to N-1
4       for j ← 0 to K-1
5         distance ← | objects[i] - clusters[j] |
6         if distance < dmin
7           dmin ← distance
8           n ← j
9       if membership[i] ≠ n
10        δ ← δ + 1
11        membership[i] ← n
12      new_clusters[n] ← new_clusters[n] + objects[i]
13      new_cluster_size[n] ← new_cluster_size[n] + 1
14    for j ← 0 to K-1
15      clusters[j][*] ← new_clusters[j][*] / new_cluster_size[j]
16      new_clusters[j][*] ← 0
17      new_cluster_size[j] ← 0
```

ทำขนานไม่ได้ สถานะเปลี่ยนไปเรื่อยๆ ปล่อยเป็น sequential

ทำขนานได้ เพราะไม่ขึ้นต่อกัน สามารถเกิดขึ้นพร้อมกันได้ parallelism

Parallelizable loops

2 อันนี้ทำขนานไม่ได้ เพราะเราต้องใช้ centroid เดิม เพื่อจัดกลุ่มให้เสร็จก่อน จึงจะหาตำแหน่ง cluster ใหม่

Parallelism Issues

Perfectly Parallel Problem

โปรแกรมที่ทำงานแยกกันสมบูรณ์ parallel ได้ง่ายๆ

- Also known as “embarrassingly parallel problem”
- A problem that is easy to divide into parallel subtasks without dependency or communication among subtasks. เช่น การบวกค่าใน array
- Unfortunately, most parallel problems are not in this class.

Issues in Parallelism (1)

- Dependency
  - A task can be executed only after all tasks that it depends on have already been executed.
  - Dependency limits parallelism.

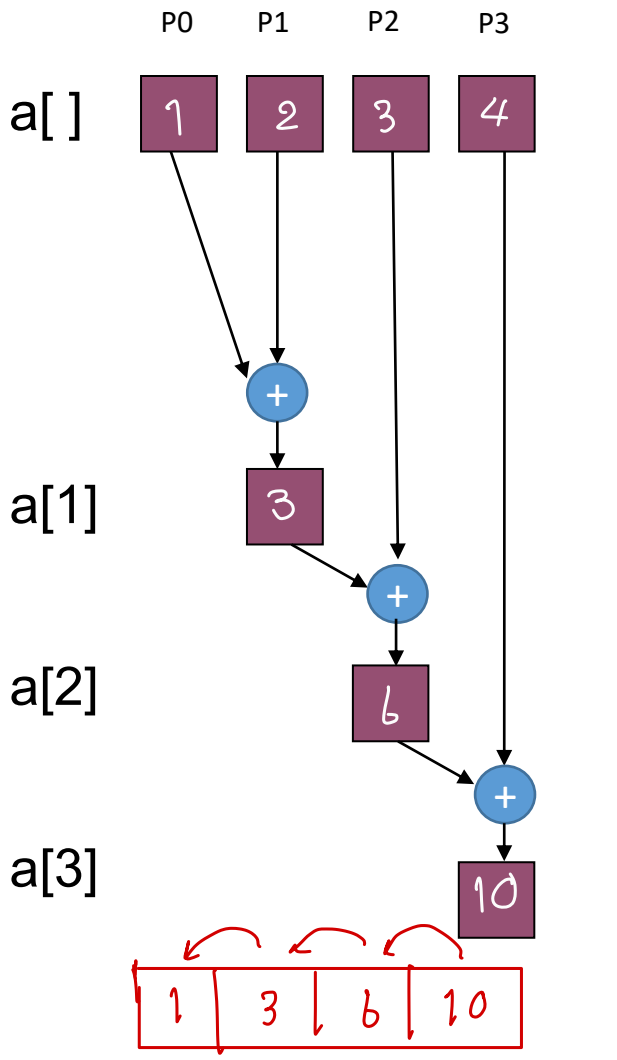
การที่ บางขั้นตอนอาศัย ผลลัพธ์จากขั้นก่อนหน้า จึงไม่ใช่ parallelism

Example: Prefix sum

```
for (i=1; i<n; i++)
        a[i] = a[i] + a[i-1];
```

dependencies

- `a[i]` depends on `a[i-1]`
- Subtasks cannot run in parallel.
- Time complexity: O(n)

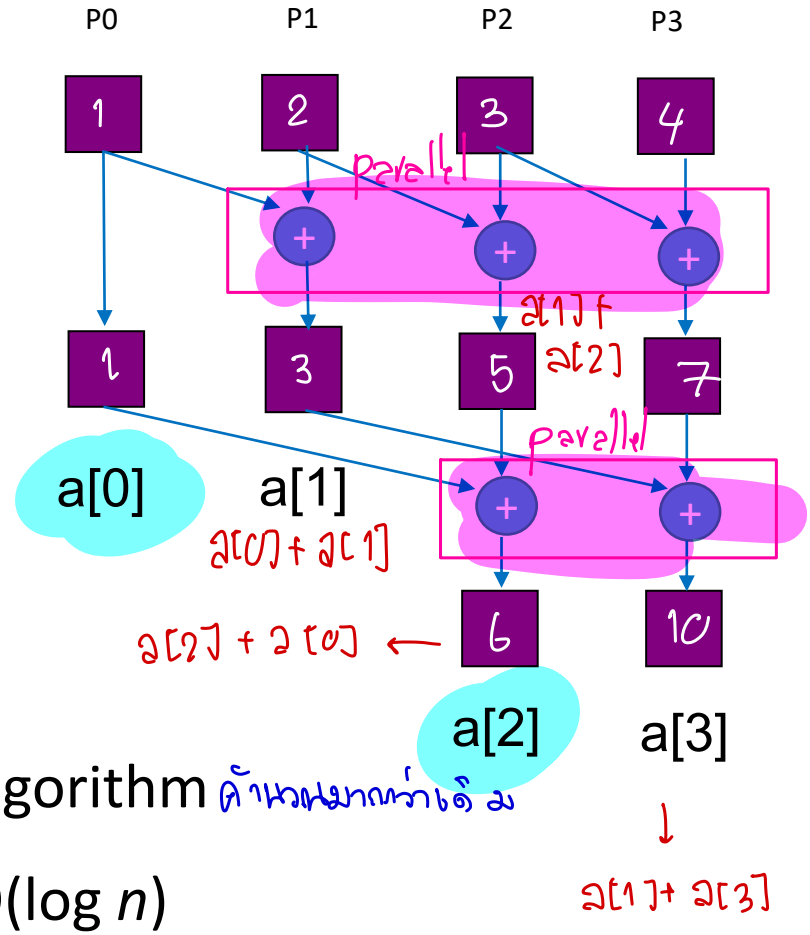


Parallel Prefix Sum Algorithm

j : จำนวน phase การคำนวณ

```
for (j=0; j < log2(n); j++)
  parallel for (i=2^j; i < n; i++)
    a[i] = a[i] + a[i-2^j];
```

เท่ากับจำนวน phase ขนานที่ได้

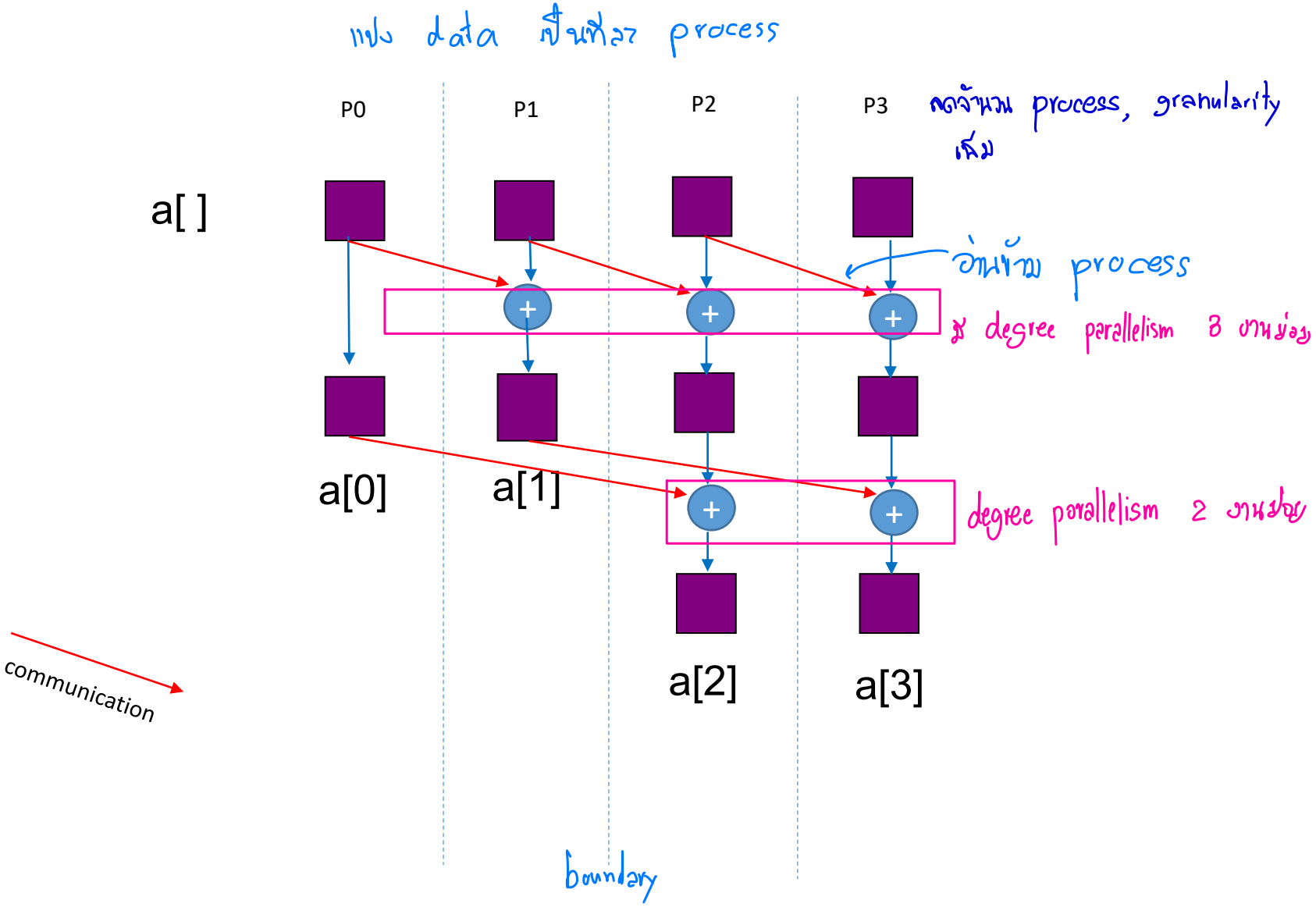


- More computation than sequential algorithm คำนวณมากกว่าเดิม
- Time complexity with *n* processors: O(log *n*)

Issues in Parallelism (2)

- Communication and synchronization
  - Needs for transferring information among subtasks
  - Communication and synchronization cause delay.
  - Communication method depends on level of parallelism
    - Threads communicate via shared memory
    - Processes on different machines communicate via messages or remote procedure call (RPC)

Communication



Issues in Parallelism (3)

- Degree of parallelism
  - The number of parallel tasks จำนวน parallel task ของงานย่อย
  - Degree of parallelism in software is usually greater than degree of parallelism in hardware. Degree ใน SW > HW

    แบ่ง parallelism มาใช้ก็ได้ไม่ใช้ ขึ้นกับว่ามี processing unit แค่ไหน

    ถ้าใช้ก็ได้ไม่ดี เพราะ CPU core ไม่คุ้มค่า
- Granularity ขนาดของงานย่อย
  - sizes of sub-tasks
  - Different levels of parallelism deal with different granularity.

$$\text{degree of parallelism} \propto \frac{1}{\text{Granularity}}$$

Issues in Parallelism (4)

- Other overheads
  - Creating and destroying subtasks / managing worker pool
  - Context switching between tasks that share the same computing element
  - Overheads may relate to degree of parallelism and granularity.

overhead การสร้าง thread มากเกินไป