Parallelism in Software

Minsoo Ryu

Department of Computer Science and Engineering
Hanyang University

Outline

1. Parallelism in Software
2. Creating a Multicore Program
3. Multicore Design Patterns
4. Q & A

Types of Parallelism

- **Parallelism in hardware (implicit parallelism)**
  - **Pipelining**
  - **Superscalar, VLIW**

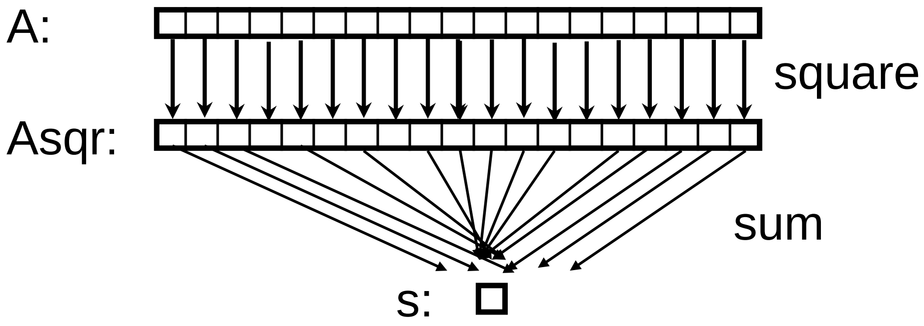
    } instruction level parallelism (ILP)
  - **SIMD processing** } data level parallelism (DLP)
  - **HW multithreading** } thread level parallelism (TLP)

- **Parallelism in software**
  - **Data parallelism**
  - **Task parallelism**
    - Also known as control parallelism or function parallelism

Parallelism in Software: Data Parallelism

- Data parallelism
  - A data set can be partitioned into multiple subsets
  - A set of tasks perform the same computation, but operate on different data
- Consider applying a function square to the elements of an array A and then computing its sum

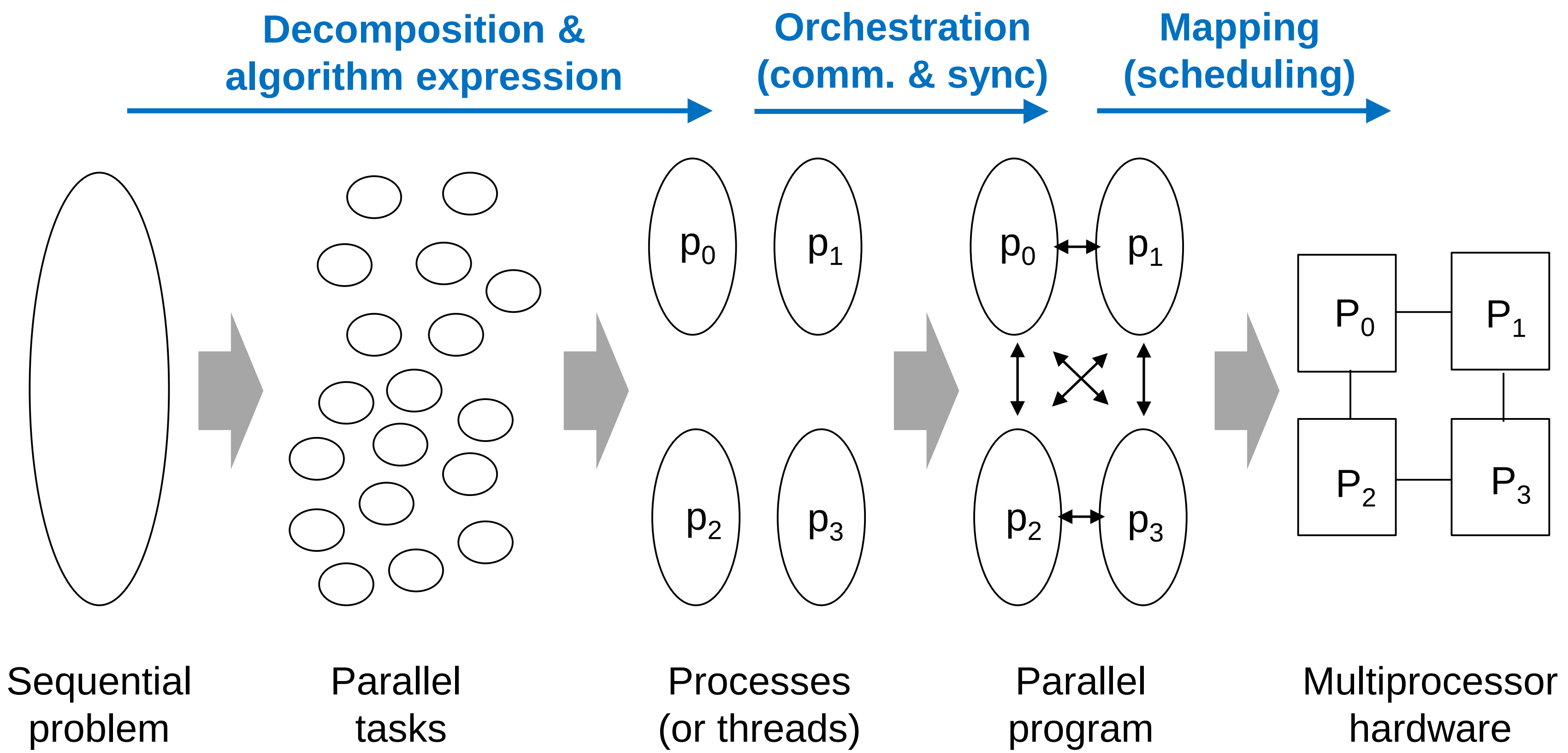
A = array of all data
Asqr = map(square, A)
s = sum(Asqr)



Parallelism in Software: Task Parallelism

- Task parallelism
  - An application can be partitioned into multiple different tasks that can be executed simultaneously
  - Task parallelism often involves data parallelism

Creating a Multicore Program



Four Design Spaces

Algorithmic Expression

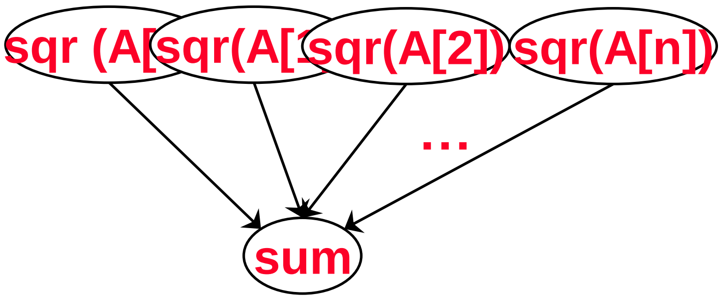
- Finding concurrency
  - Decompose the problem (expose concurrent tasks)
- Algorithm structure
  - Map tasks to processes to exploit parallel architecture

Software Construction

- Supporting structures
  - Code and data structuring
- Implementation mechanisms
  - Low level mechanisms used to write parallel programs

Problem Decomposition

- Decompose the problem into smaller tasks
  - Determine which can be done in parallel with each other, and which must be done in some order
- Conceptualize a decomposition as a task dependency graph with:
  - nodes corresponding to tasks
  - edges indicating dependencies



- A given problem may be decomposed into tasks in many different ways
  - Tasks may be of same, different, or indeterminate sizes

Multicore Design Patterns

- **What is a design pattern?**
  - A design pattern is a general reusable solution to a commonly occurring problem within a given context
  - A design pattern is not a finished solution, but a description or template (a formalized best practice)

- **Books on design patterns**
  - "Design Patterns: Elements of Reusable Object-Oriented Software," E. Gamma, R. Helm, R. Johnson and J. Vlissides
  - "Patterns for Parallel Programming," T.G. Mattson, B.A. Sanders, and B.L. Massingill

Input/Output Data Decomposition

- Data decomposition for matrix multiplication
  - Matrices are partitioned into four submatrices

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

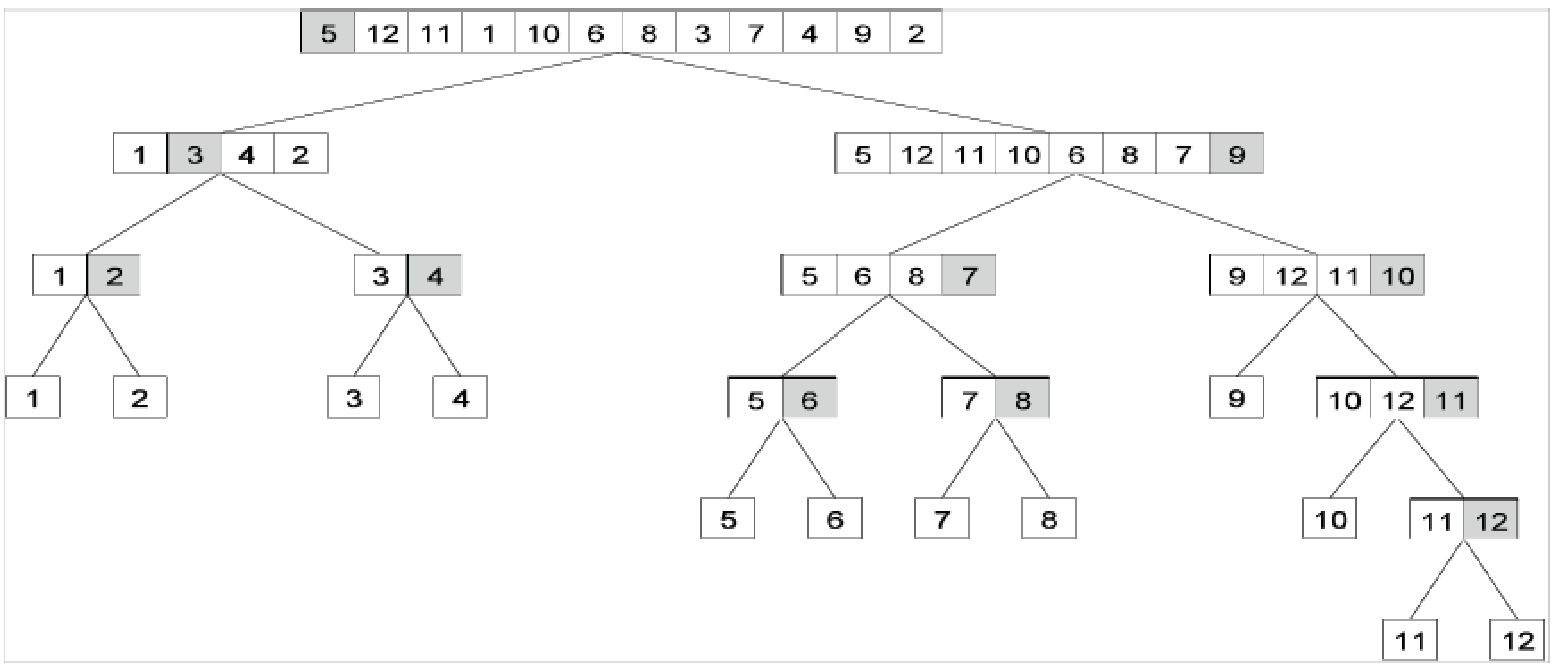
(a)

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

(b)

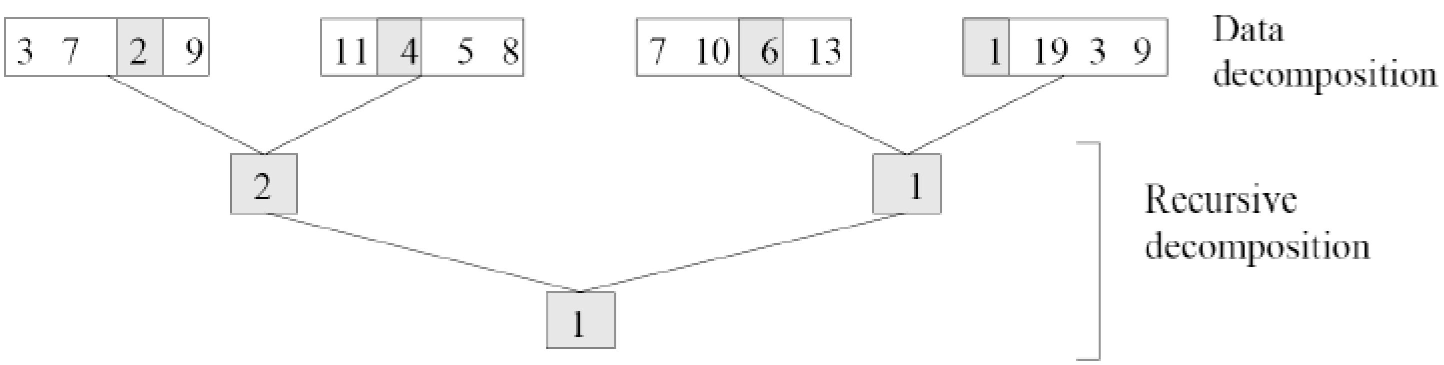
Recursive Decomposition

- **Recursive decomposition for Quicksort**
  - dived-and-conquer algorithm



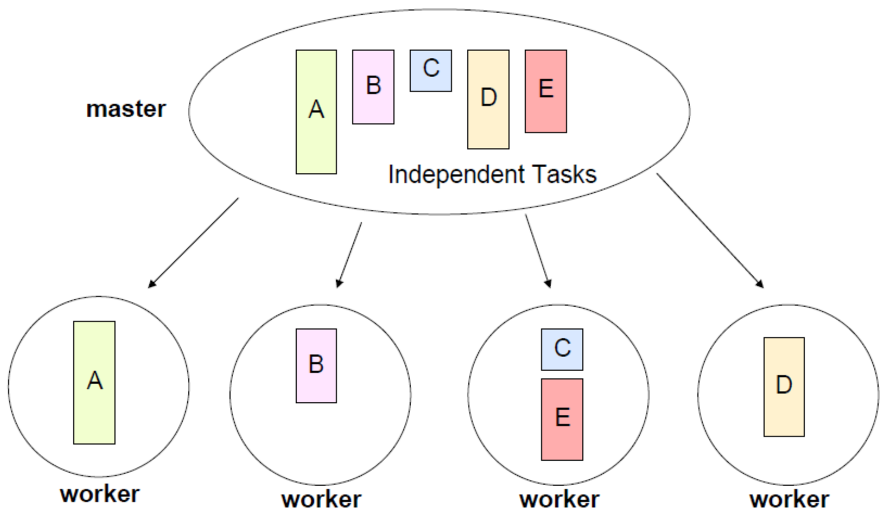
Hybrid Decomposition

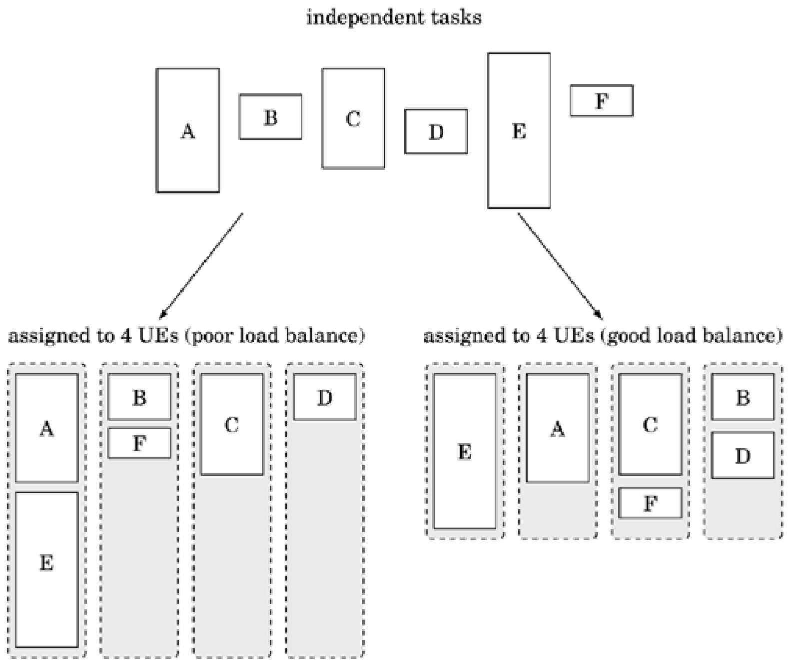
- **Hybrid decomposition for finding the minimum**
  - **A mix of decomposition techniques is often needed**



Direct Task Decomposition

- Organize the system with independent tasks
  - Decomposition into independent tasks
  - Replication of a task

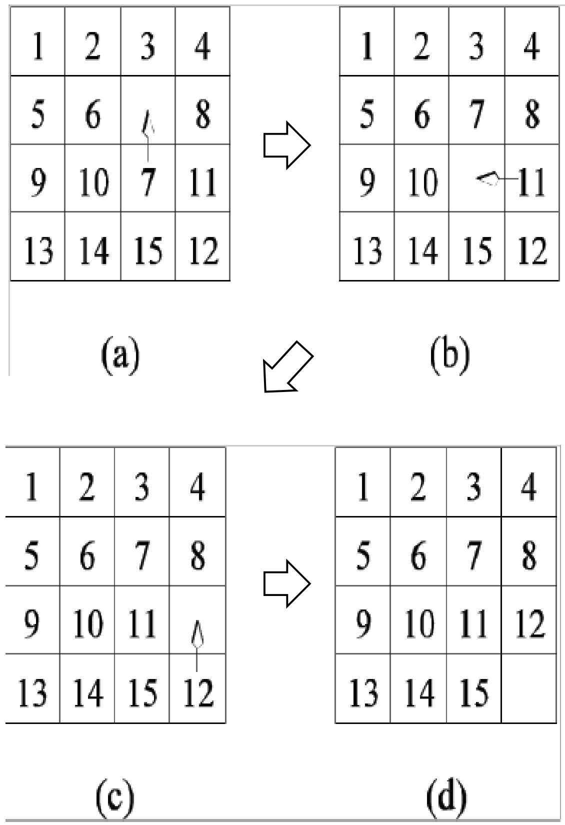


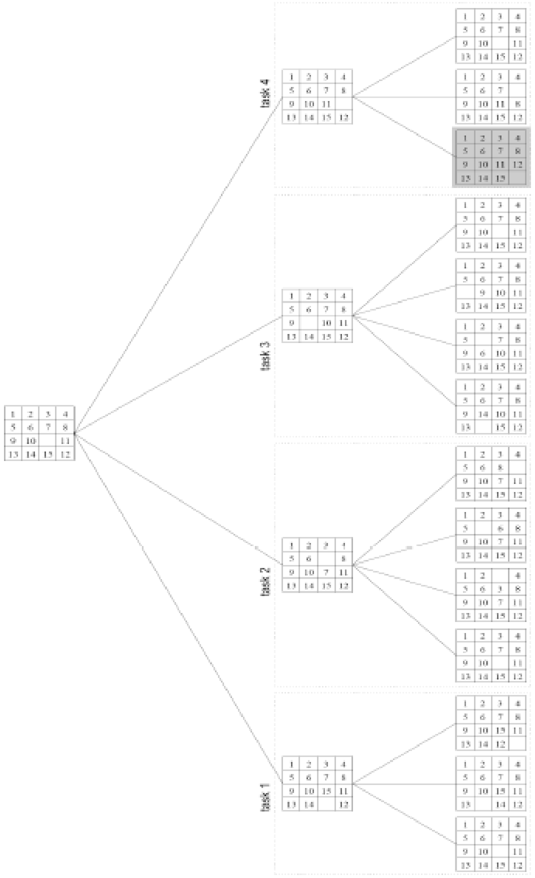


  - Tasks can be executed in a master/worker manner

Exploratory Decomposition

- **Exploratory decomposition for a tile puzzle**
  - **Search of a state space of solutions**



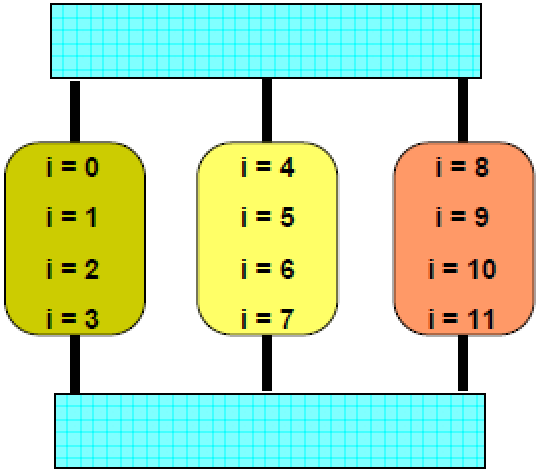


Loop Parallelism

Loop parallelism pattern

- Many programs are expressed using iterative constructs
- We can divide iterations into multiple sets of iterations and execute them in parallel

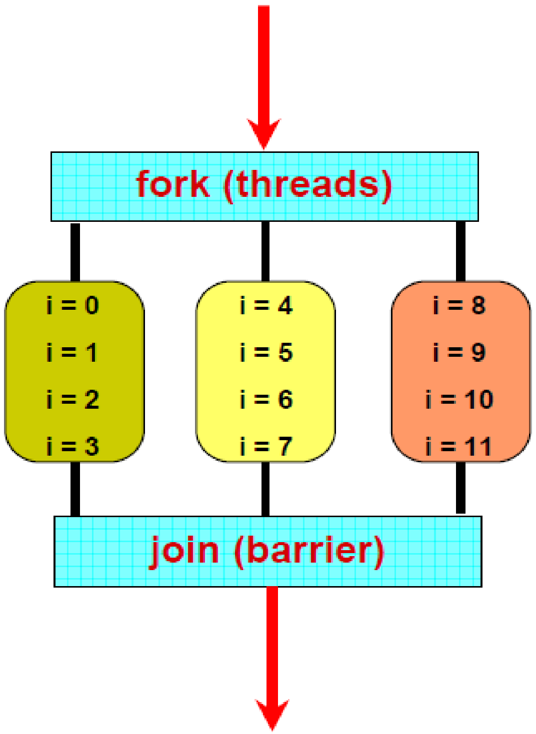
```
for (i = 0; i < 12; i++)
        C[i] = A[i] + B[i];
```



Fork and Join

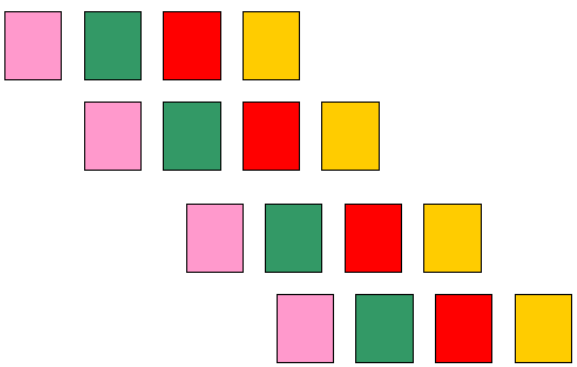
- Fork/Join pattern
  - Parent task creates new tasks (fork) then waits until they complete (join) before continuing on with the computation

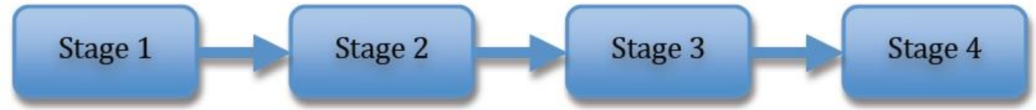
```
for (i = 0; i < 12; i++)
      C[i] = A[i] + B[i];
```



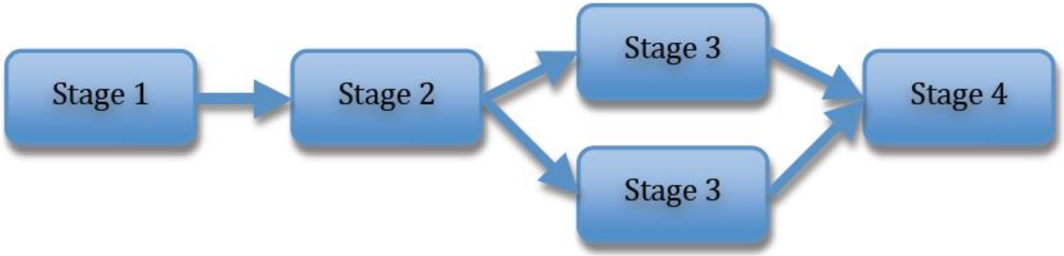
Pipeline

- **Pipeline pattern**
  - **Tasks are organized in terms of the flow of data**
  - **Examples**
    - Ray tracing, molecular dynamics simulation, video codec





Linear pipeline

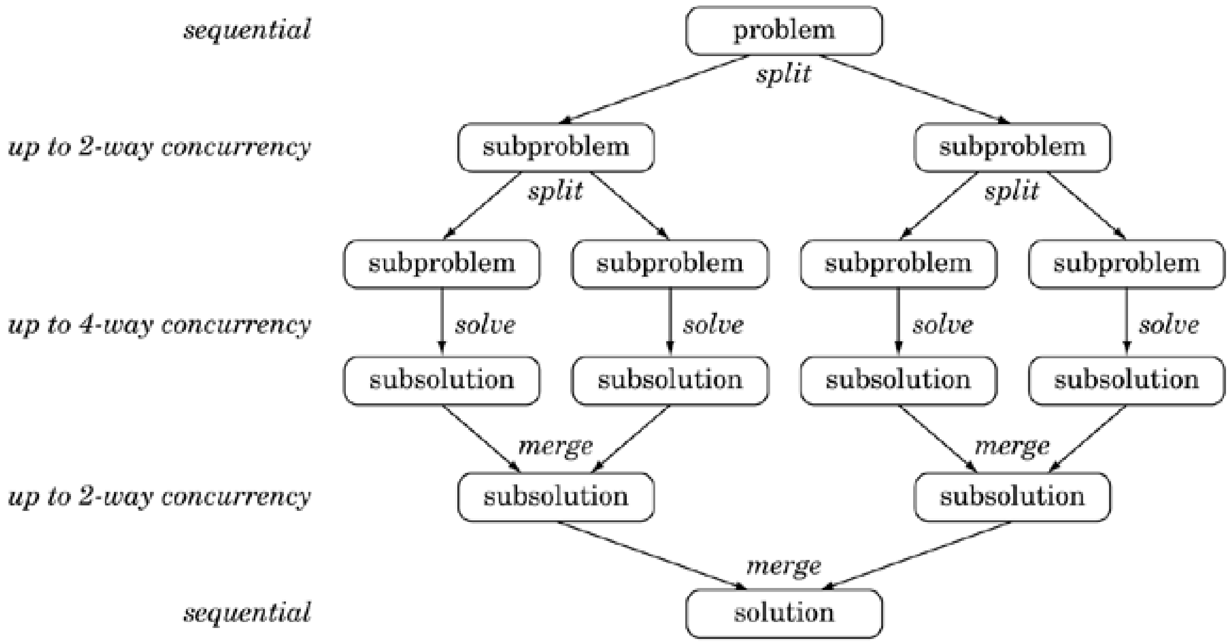


Nonlinear pipeline

Divide and Conquer

➢ Divide and conquer pattern

- A problem is solved by splitting it into a number of smaller subproblems, solving them independently, and merging the subsolutions
- Examples: Merge sort, matrix diagonalization



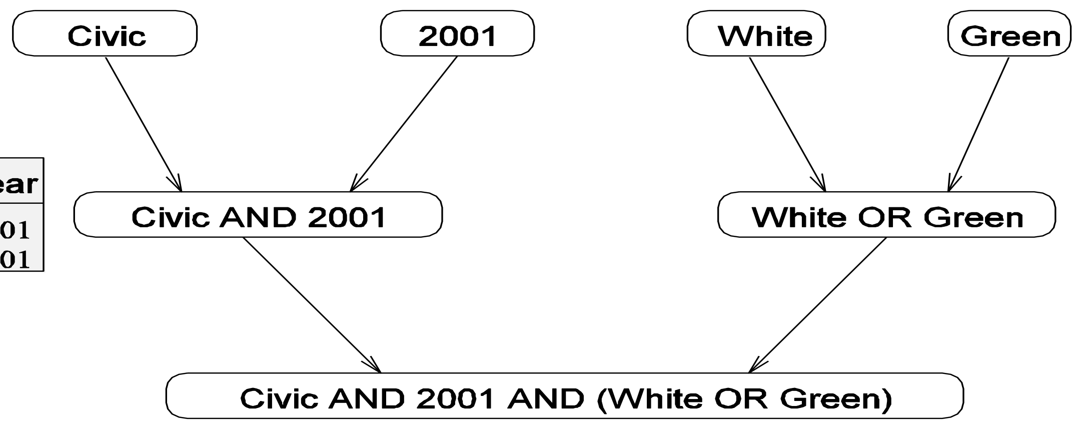
Example 1. Database Query Processing

- **Consider the following query**
  - **MODEL = ''CIVIC'' AND YEAR = 2001 AND COLOR = ''GREEN'' OR COLOR = ''WHITE ''**

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Example 1. Database Query Processing

➢ **One possible decomposition**



ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

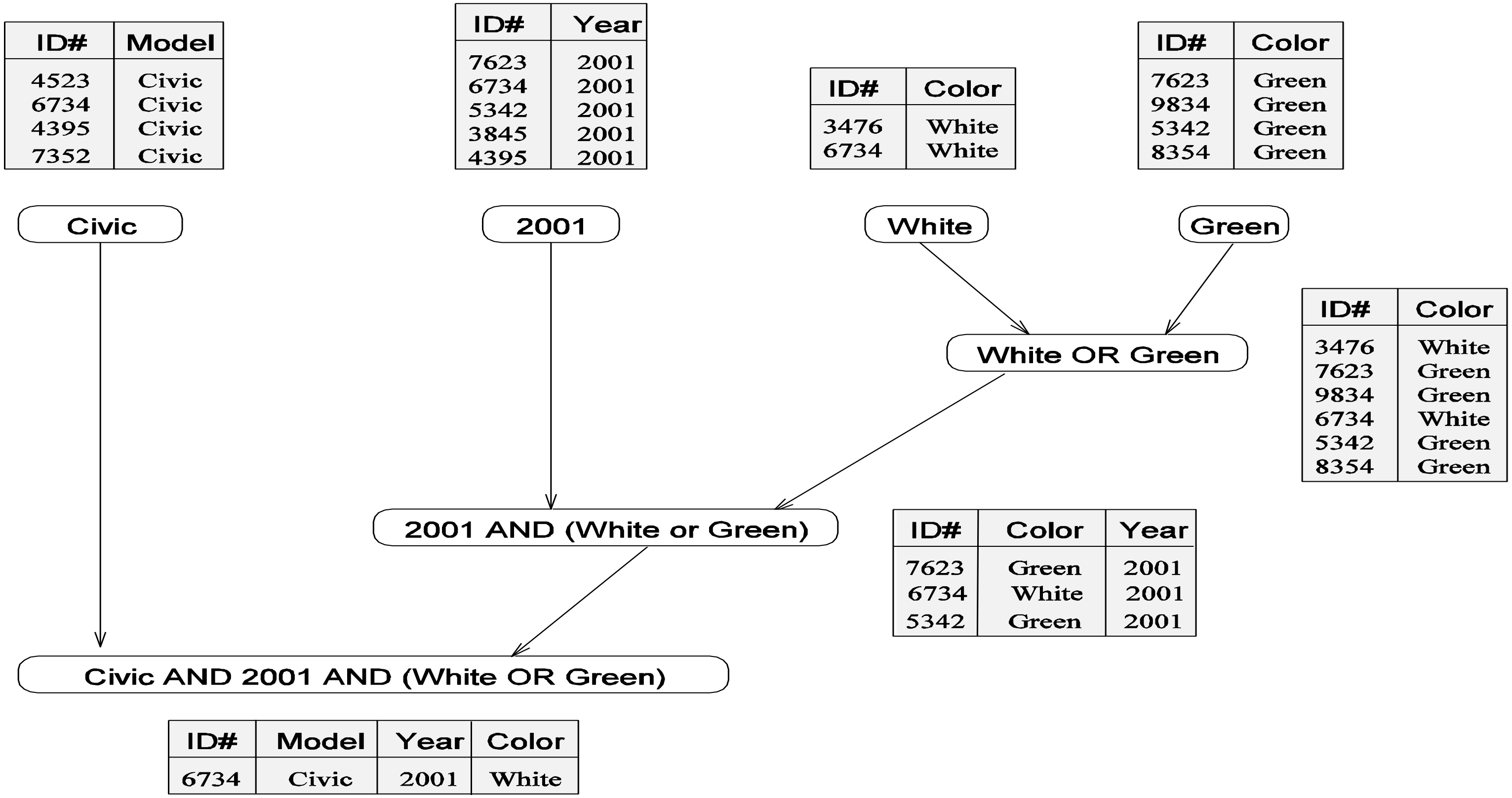
ID#	Model	Year
6734	Civic	2001
4395	Civic	2001

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Model	Year	Color
6734	Civic	2001	White

Example 1. Database Query Processing

- A different approach



ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

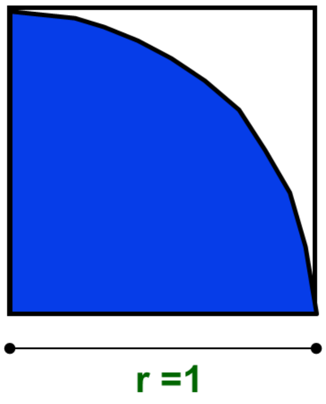
ID#	Color	Year
7623	Green	2001
6734	White	2001
5342	Green	2001

ID#	Model	Year	Color
6734	Civic	2001	White

Example 2. Monte Carlo Pi Calculation

- Estimate Pi by throwing darts at a unit square
- Calculate percentage that fall in the unit circle
  - Area of square = $r^2 = 1$
  - Area of circle quadrant = $¼ * \pi r^2 = \pi/4$

- Randomly throw darts at x, y positions
- If $x^2 + y^2 < 1$, then point is inside circle
- Compute ratio:
  - # points inside / # points total
  - $\pi = 4*\text{ratio}$



Example 2. Monte Carlo Pi Calculation

```
main(int argc, char **argv) {
  int i, hits, trials = 0;
  double pi;

  if (argc != 2)trials = 1000000;
  else trials = atoi(argv[1]);

  srand(0);  // see hw tutorial

  for (i=0; i < trials; i++) hits += hit();
  pi = 4.0*hits/trials;
  printf("PI estimated to %f.", pi);
}
```

Example 2. Monte Carlo Pi Calculation

```
int hit(){
   int const rand_max = 0xFFFFFF;
   double x = ((double) rand()) / RAND_MAX;
   double y = ((double) rand()) / RAND_MAX;
   if ((x*x + y*y) <= 1.0) {
        return(1);
   } else {
        return(0);
   }
}
```


Q&A