Computer and Automation Research Institute

Hungarian Academy of Sciences

Pipelined computation (in P-GRADE)

Péter Kacsuk

Laboratory of Parallel and Distributed Systems

MTA SZTAKI Research Institute

kacsuk@sztaki.hu

www.lpds.sztaki.hu

TYPES OF AVAILABLE PARALLELISM

Data and Computation Decomposition

a) How ?

- **Data Parallelism**

  Decompose and distribute data. All the processors execute the same code but operating on local data

- **Function Parallelism**

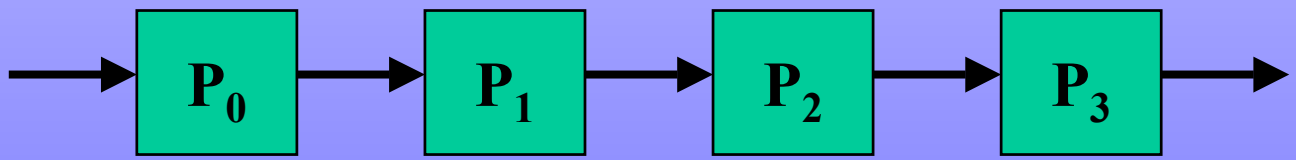
  Decompose COMPUTATIONS into pieces (functions).
  Data are taken to the tasks where they are needed.

- **Pipelining**

  A given (complex) computation must be performed on a large set of data. Decompose the COMPUTATION into functions.

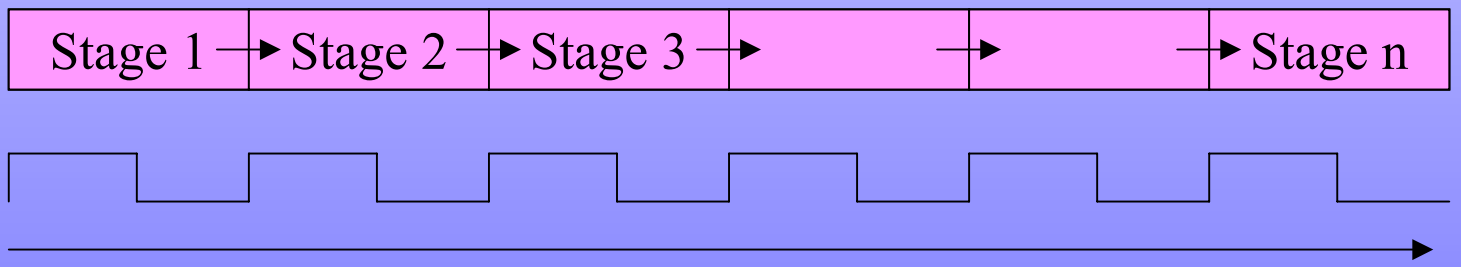
Basic principles of pipelining

- The problem is divided into a number of successive tasks
- A pipeline stage is associated with each task
- Each stage *outputs* to the *input* of the next stage
- Same time required for each stage

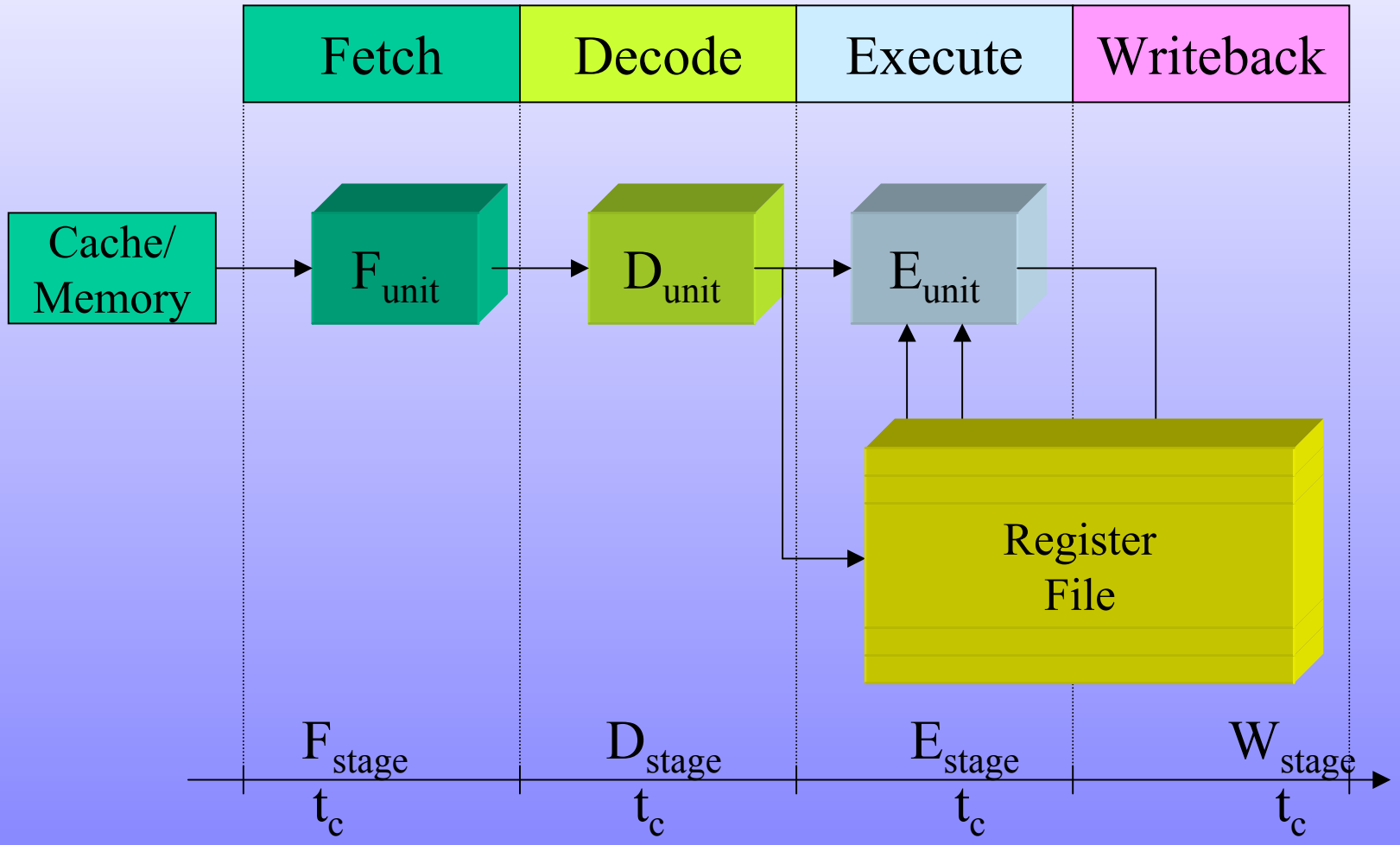


Heterogeneous pipelines

- Represent functional decomposition: each stage is a different function
- Equal timing of stages is not trivial
- Example: pipeline processors
- Pipeline is *clocked* synchronously in hardware.



Pipelining in processors



Homogeneous pipelines

- Represents data decomposition: each stage has the same function
- Equal timing of stages is easier but still not trivial
- Examples:
  - Adding numbers
  - Frequency filter
  - Sorting numbers
  - Prime number generation

Adding numbers

- Consider a simple loop:

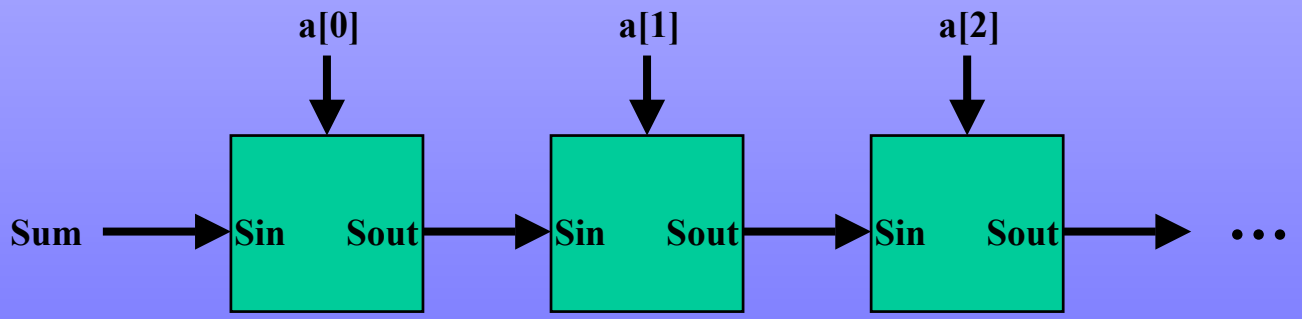
```
for (i = 0; i < n; i++)
        sum = sum + a[i];
```

- The loop could be unfolded

```
sum = sum + a[0];
sum = sum + a[1];
sum = sum + a[2];   ...
```

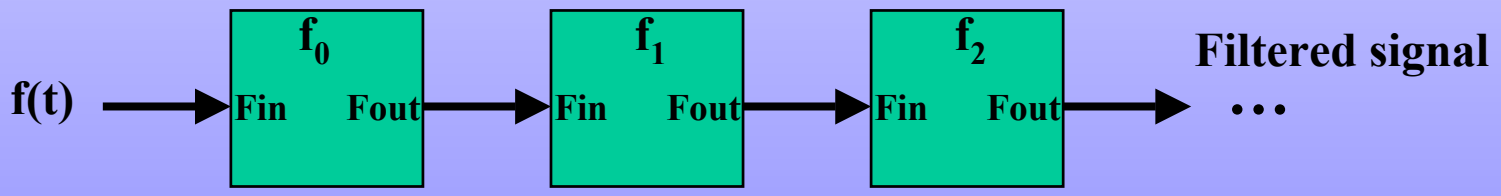
- Stage i performs:

$$s_{out} = s_{in} + a[i];$$



Frequency filter

- The objective is to remove specific frequencies (say f0, f1, f2, f3, … fn) from a digitized signal f(t).
- Each stage is responsible for removing one of the frequencies



Speed-up in pipelines

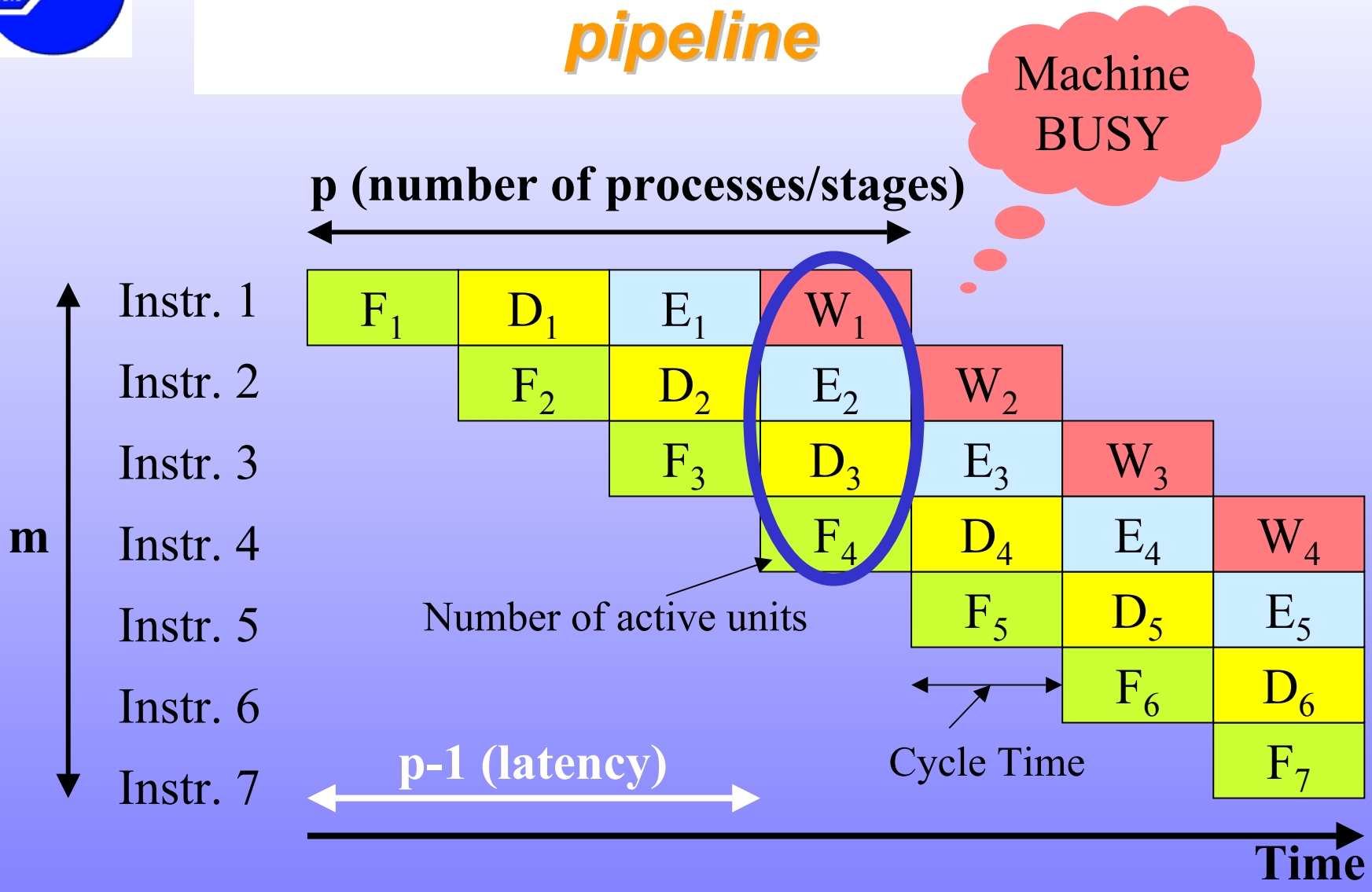
- The pipeline approach can provide speed-up for the following **three types** of computation:

  Type 1. If **more than one instance** of the complete problem is to be executed (example: pipeline processors)

  Type 2. If a **series of data items** must be processed, each requiring multiple operations (example: adding numbers)

  Type 3. If information to start the next process can be passed forward before the process has completed all its internal operations (example: Prime number generation)

Space-time diagram of Type 1 pipeline



Analysis of pipelines type 1 and 2

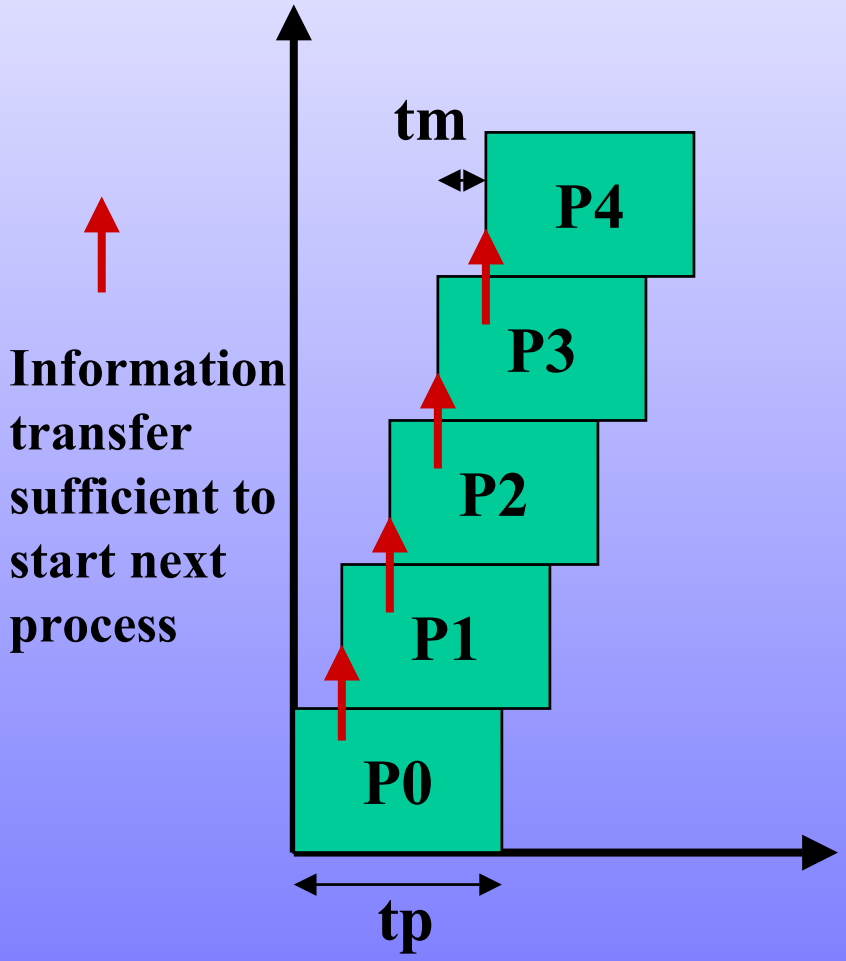
- The number of pipeline cycles to execute
  - **m instances** of the problem
  - with **p processes (stages)** constituting the pipeline

  is

  p - 1 + m cycles
- The average number of cycles for the m instances:

  (p - 1 + m) / m -> 1 **if m >> p**
- One instance of the problem will be completed in each pipeline cycle after **the first p -1 cycles** (called the **pipeline latency**)
- Szekvenciális végrehajtási idő: m * p
  Párhuzamos végrehajtási idő: **(p-1) + m**
  Gyorsítás: m*p/(p-1+m) -> p if m>>p

Space-time diagram of pipelines type 3



- Sequential execution time:

  $Ts = p * tp$

- Parallel execution time:

  $Tp = (p - 1) * tm + tp$

- Computing speed-up:

  $1/S = Tp/Ts =$

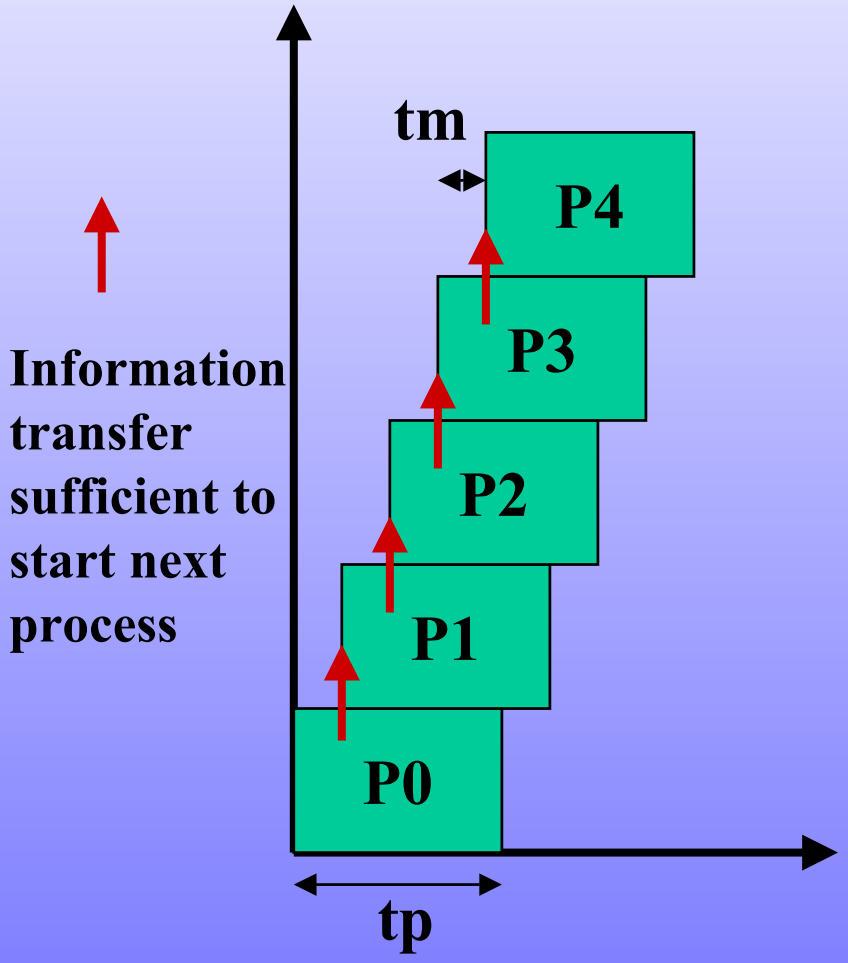
  $= tm/tp - tm/p*tp + 1/p$

For example:

S = 5 if p=10, tm/tp = 1/10

$1/S = 1/10 - 1/100 + 1/10 = 1/5$

Space-time diagram of pipelines type 3



1/S = tm/tp - tm/p*tp + 1/p ->

-> tm/tp + 1/p

a/ 1/S = 2/p if tm/tp = 1/p

S = p/2

S =50 if p=100, tm/tp = 1/100

b/ 1/S = 1/p if 1/p > 10tm/tp

S = p if p > tp/10tm

S =10 if p=10, tm/tp = 1/100

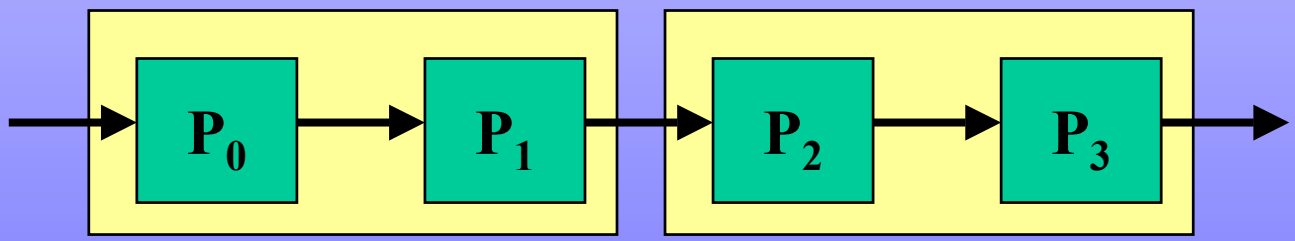
b/ 1/S = tm/tp if tm/tp > 10/p

S = tp/tm if tp/tm > 10p

S =10 if p=100, tm/tp = 1/10

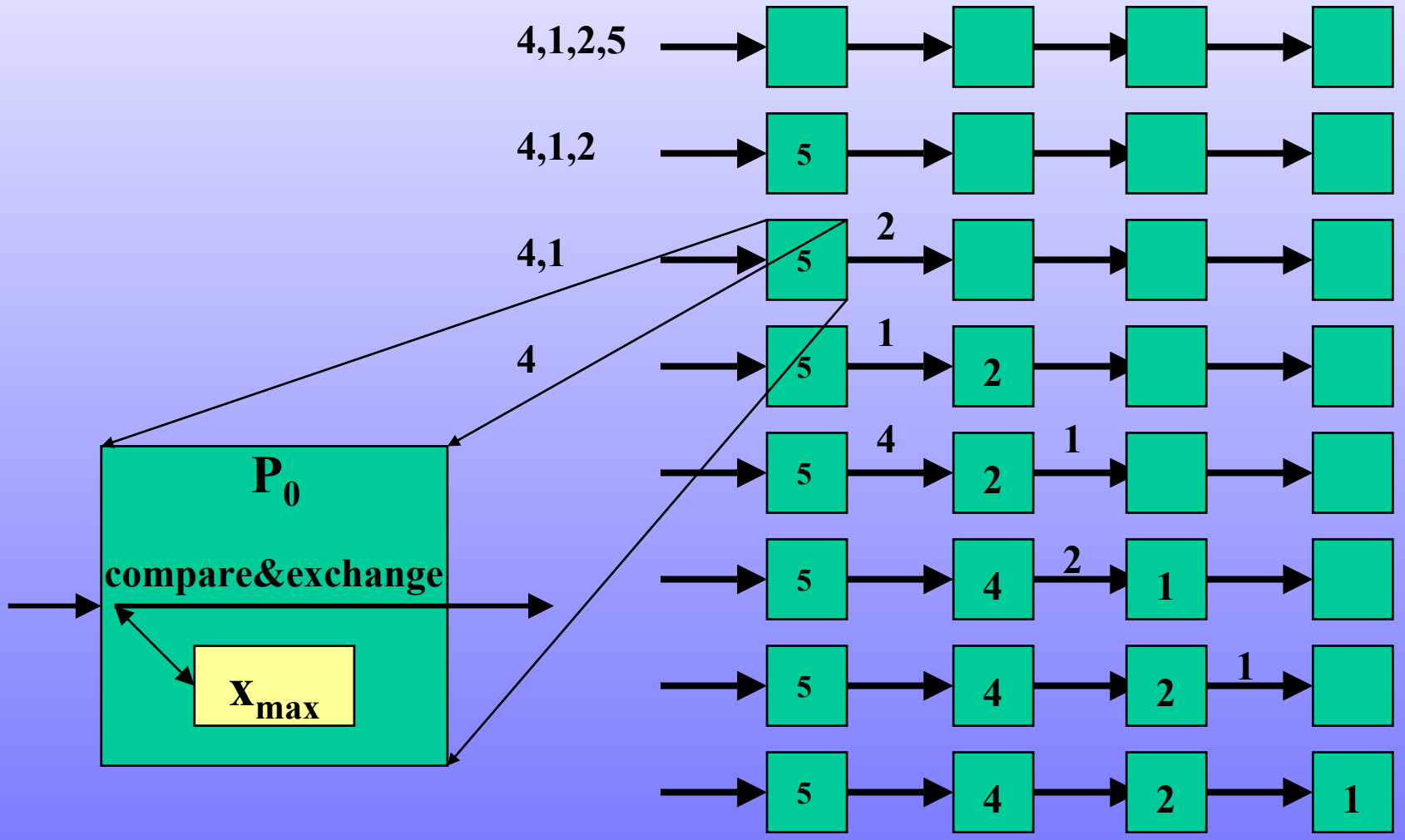
Mapping

- If the number of stages larger than the number of processes, a group of stages can be assigned to each processor.
- Execution of stages within one processor is sequential.

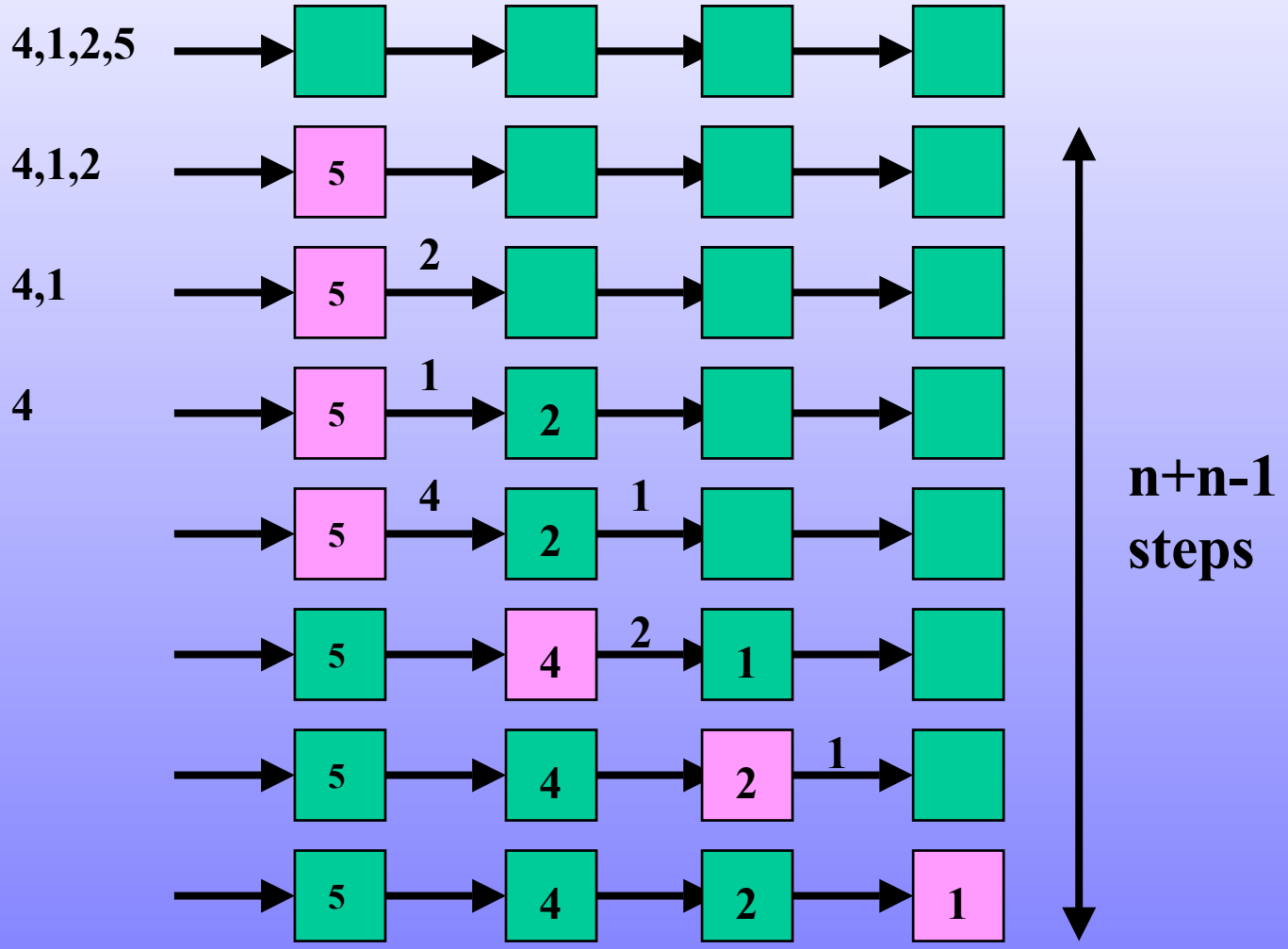


Sorting numbers

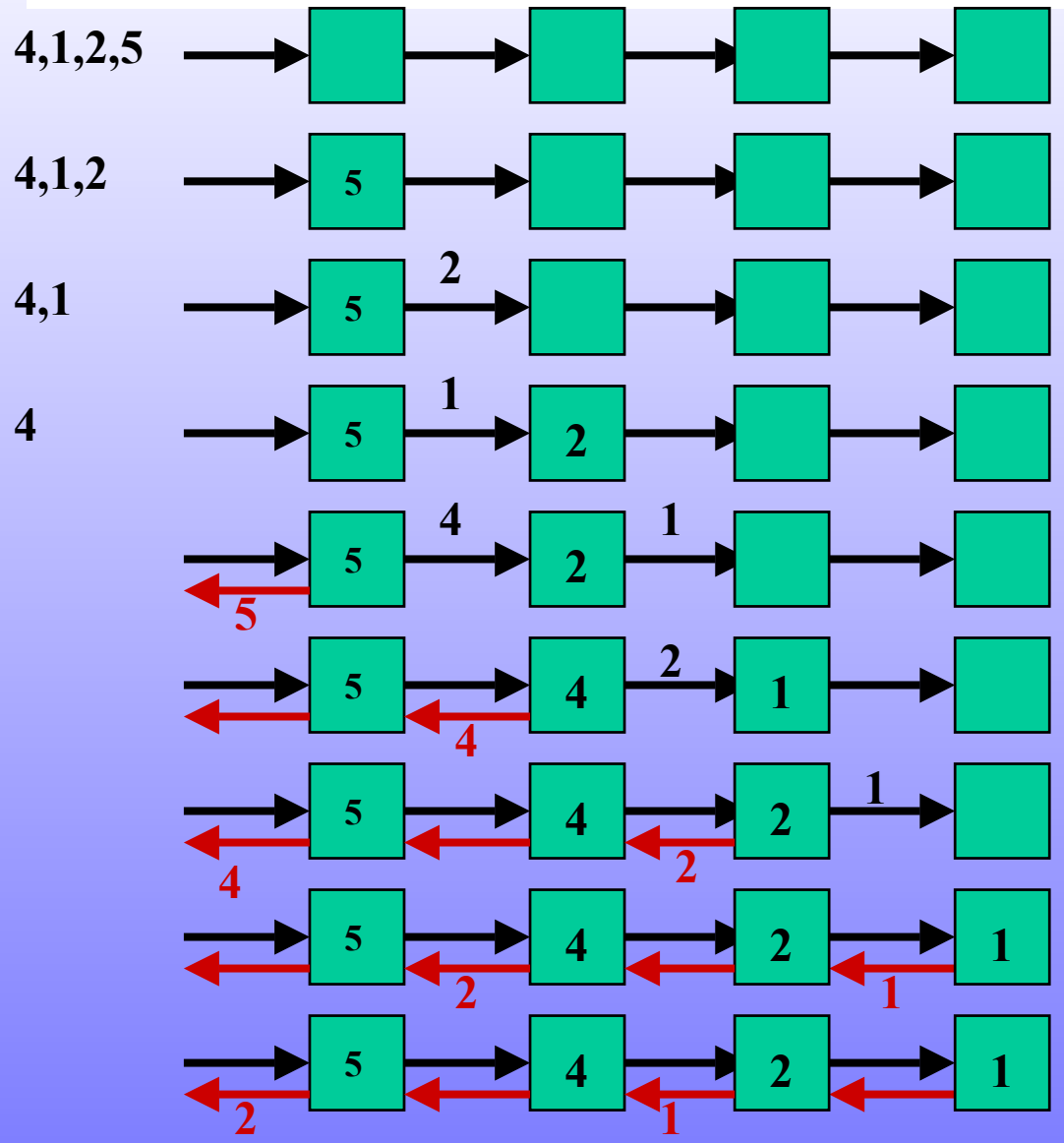
- The objective is to reorder a set of numbers in increasing order



Sorting numbers



Sending back results



Analysis of sorting numbers

If there are **n** numbers to sort, the number of sequential steps is:

$ts = (n - 1) + (n - 2) + \ldots + 2 + 1 = n * (n - 1) / 2$ **$O(n^2)$**

regarding the compare&exchange operation as one computational step.

It requires n-1 steps to find the largest number,

n-2 steps to find the next largest number, etc.

The parallel implementation has

$n + n - 1 = 2n - 1$ cycles **$O(n)$**

Algorithm speed-up ~ O(n)

Yet, parallelisation is not efficient!

WHY?

Analysis of sorting numbers II.

The parallel implementation requires:

n processes to sort n numbers.

In order to achieve the algorithm speedup

we need n processors as well.

Speed-up if n >10:

$S = t_{seq} / t_{par} = n(n-1) / 2*(2n-1) \sim n^2/4n = n/4$

Efficiency:

$E = S/N = n(n-1) / 2n*(2n-1) = (n-1) / (4n-2) < 1/4$

Each processor does useful work in maximum 25% of its working time.

Analysis of sorting numbers III.

The parallel implementation has 2n - 1 cycles:

Compution at each cycle consists of

1 compare&exchange operation

Communication at each cycle consists of

1 recv + 1 send operations

Total parallel execution time:

$t_{par} = (t_{comp} + t_{comm})(2n - 1)$ where

$t_{comp} = 1$ and $t_{comm} = 2\,(t_{startup} + t_{data})$

Typically

$t_{startup} + t_{data} = 10\, t_{comp}$

Analysis of sorting numbers IV.

The sequential implementation has n(n - 1)/2 cycles:

Total sequential execution time:

$$t_{seq} = n(n - 1)/2 * t_{comp} = n(n - 1)/2$$

Speed-up:

$$S = t_{seq} / t_{par} = n(n - 1) / 2*(2n -1)(1+ 2 (t_{startup} + t_{data}))=$$

$$n(n - 1) / 2*(2n -1)(1+ 20) \sim n^2/84n = n/84$$

Efficiency:

$$E = S/N = 1/84$$

Each processor does useful work in about 1% of its working time.

Prime number generation - sieve of Eratosthenes

Definition:

The algorithm removes nonprimes, leaving only primes:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, …

After selecting 2, we get:

2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15, ~~16~~, …

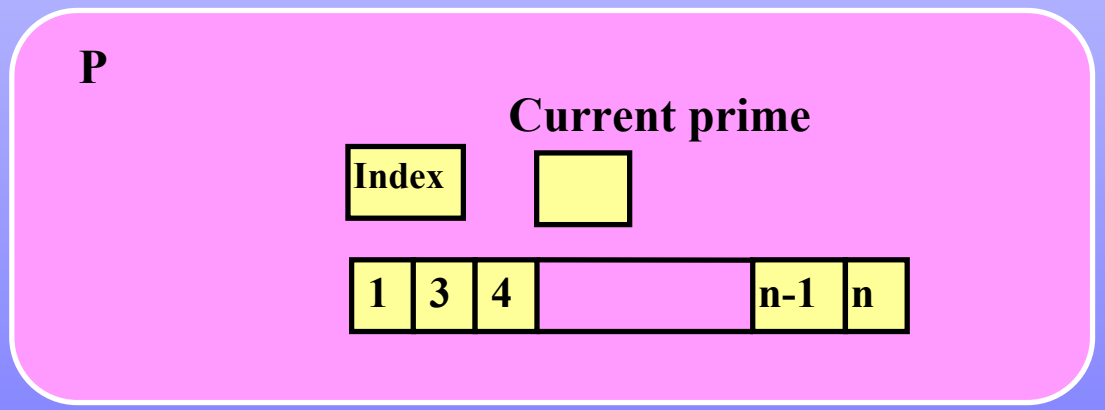
After selecting 3, we get:

2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, …

SEQUENTIAL SOLUTION OF THE SIEVE OF ERATOSTHENES

Three key data structures are used:

- A boolean array whose elements correspond to the natural numbers being sieved
- an integer representing the latest prime number found
- an integer used as a loop index incremented as multiples of the current prime are marked as composite numbers



Analysis of prime number generation

There are $\lfloor n/2 - 1 \rfloor$ multiples of 2

$\lfloor n/3 - 1 \rfloor$ multiples of 3, etc.

Total sequential time:

$$t_s = \left\lfloor \frac{n}{2} - 1 \right\rfloor + \left\lfloor \frac{n}{3} - 1 \right\rfloor + \left\lfloor \frac{n}{5} - 1 \right\rfloor + \ldots + \left\lfloor \frac{n}{\sqrt{n}} - 1 \right\rfloor$$

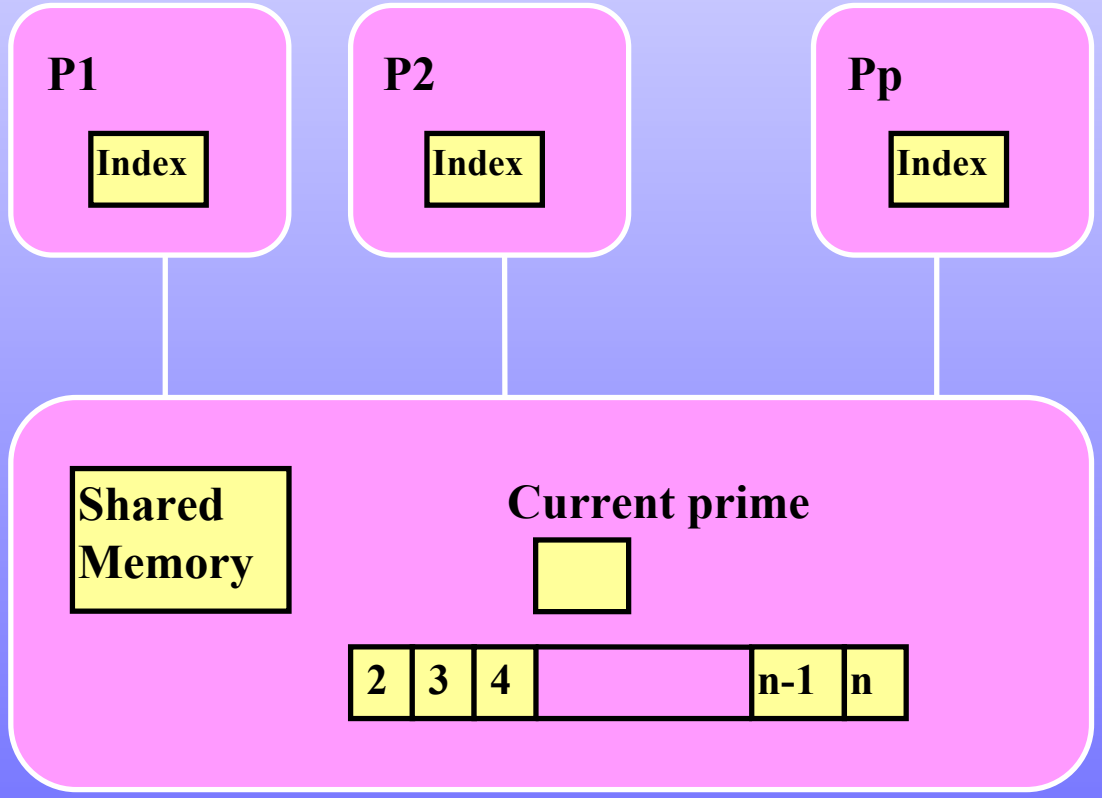
Sequential time complexity: **$O(n^2)$**

FUNCTION PARALLEL SOLUTION OF THE SIEVE OF ERATOSTHENES

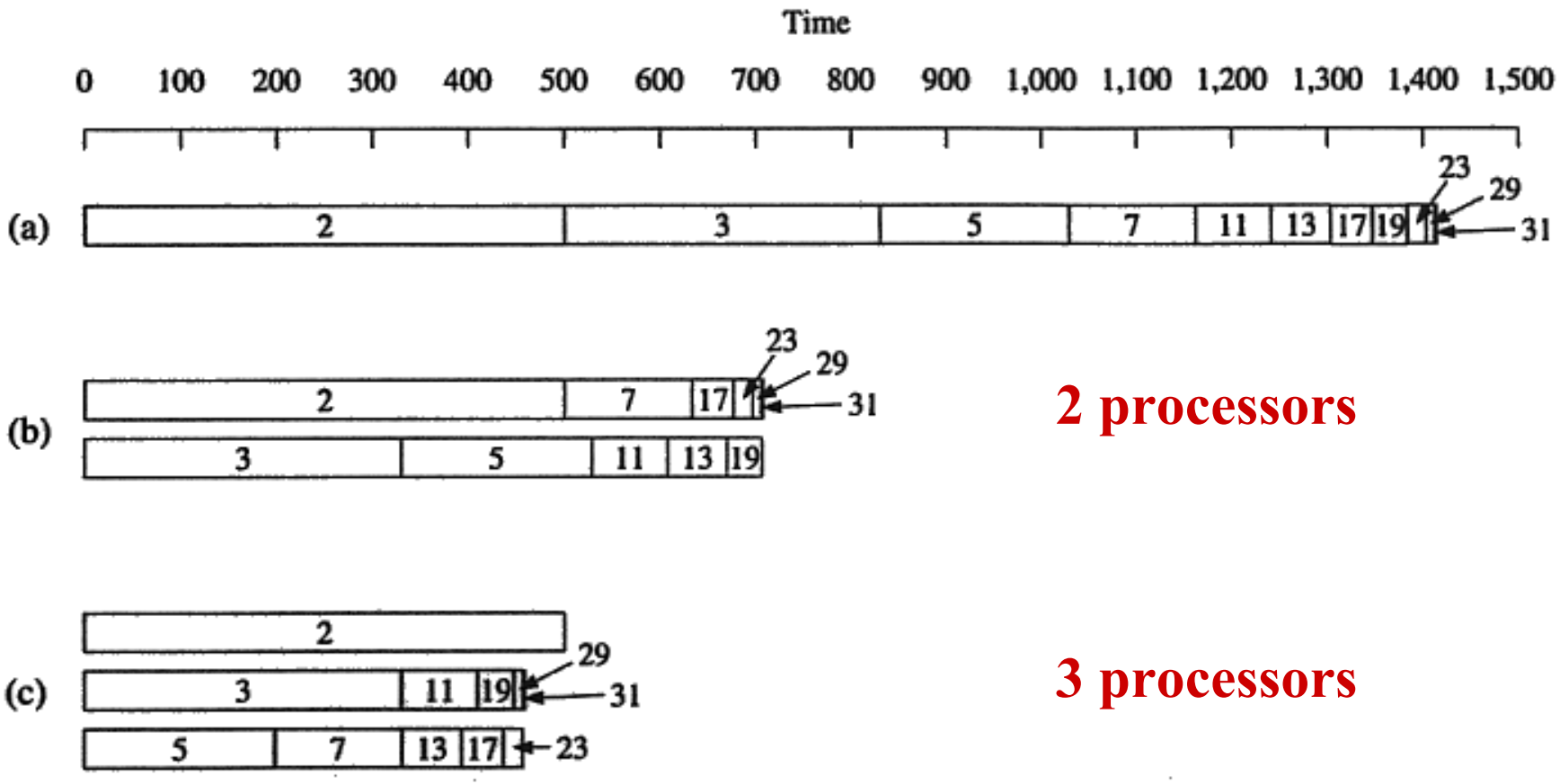
Shared memory model

for function parallel Sieve of Erathostenes algorithm

Each processor has its own private loop index and shares access to other variables with all the other processors



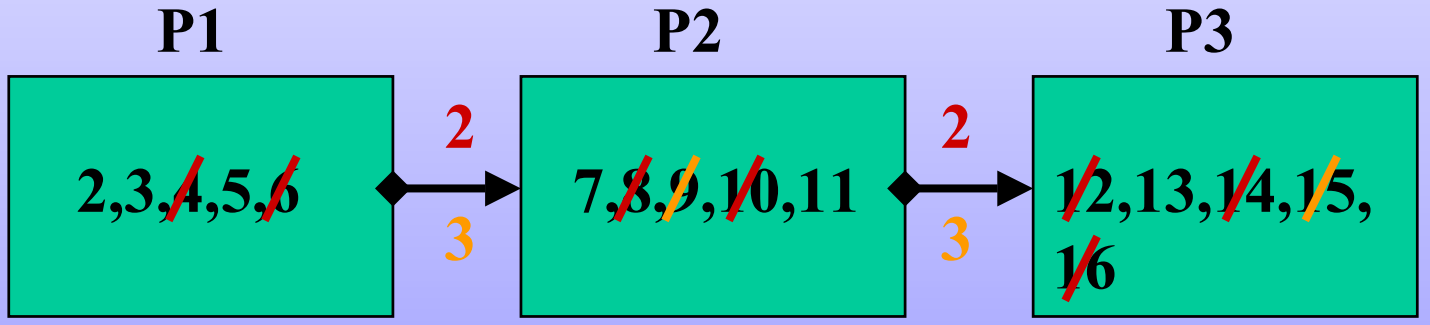
LIMITATIONS OF THE FUNCTION PARALLEL SOLUTION



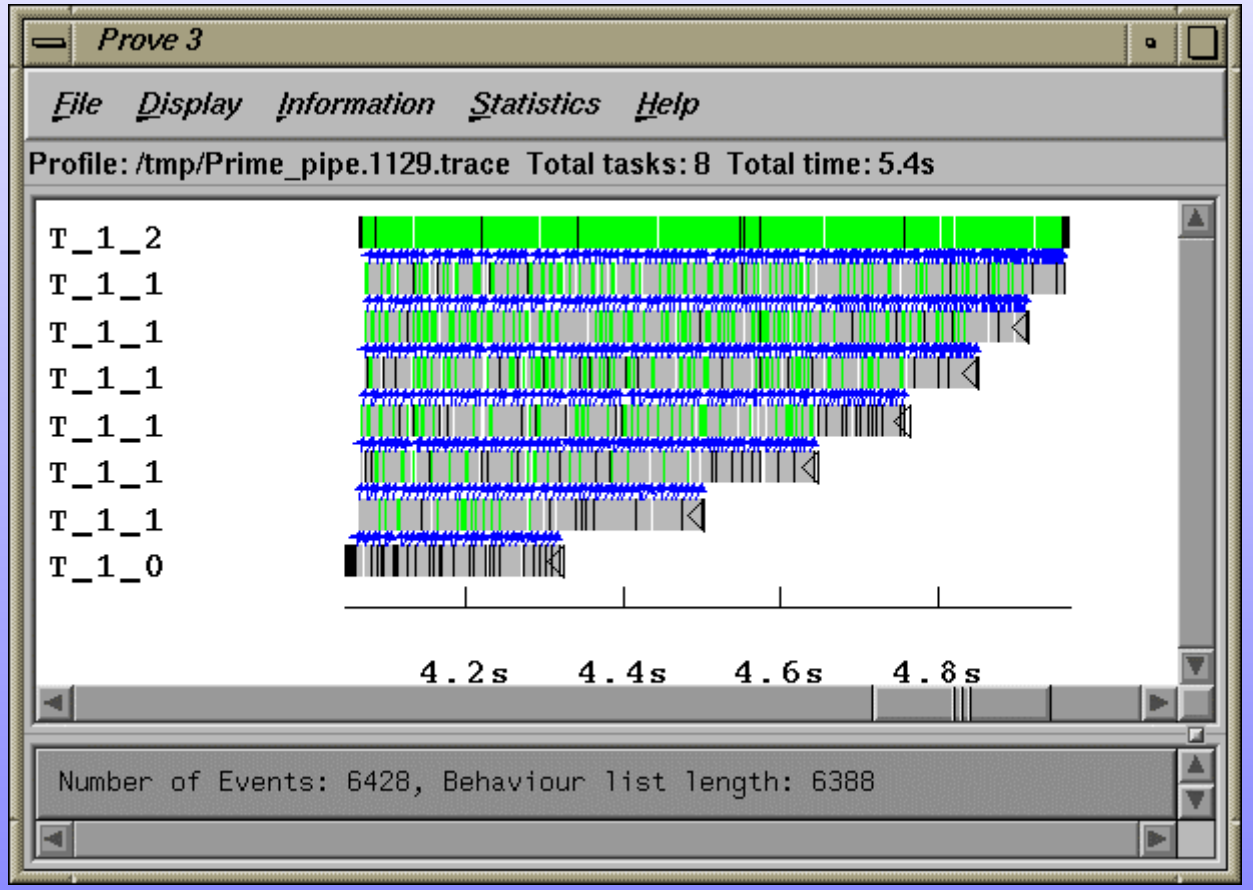
2 processors

3 processors

Pipeline implementation of sieve of Eratosthenes



Pipeline implementation of sieve of Eratosthenes



Thank You ...

Gyakorlat

Login: student1

:

:

student10

password: tanulo123

$ssh n0

$pgrade

$cp /usr/local/GRADE/config/pvmhosts .

$vi pvmhosts

---------- ha a PVM nem indul ----------

$killallpvm

Gyakorlat

1. Futtasd le a Prime_pipe példaprogramot 4 és 8 processzel, 4 ill. 8 processzorral. Van-e különbség a végrehajtási sebességben?
2. Hasonlitsd össze a szinkron és aszinkron (Prime_pipe_asyn) megoldás sebességét. Mi a különbség oka?
3. A jelen verzióban a keresési tartomány egyenletesen van elosztva a pipe tagjai között, de a processzek végrehajtási ideje jelentősen eltér. Írj egy olyan verziót, ahol a kiosztás nem egyenletes, de a végrehajtási idő kb. azonos minden proceszben.