Adaptive Algorithms for new Parallel Supports

Bruno Raffin, Jean-Louis Roch, Denis Trystram

MOAIS

ID Lab, INRIA, France

Laboratoire Informatique et Distribution

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Institut National Polytechnique de Grenoble

INRIA INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

GRENOBLE 1 UNIVERSITÉ JOSEPH FOURIER SCIENCES, TECHNOLOGIE, MÉDECINE

Overview

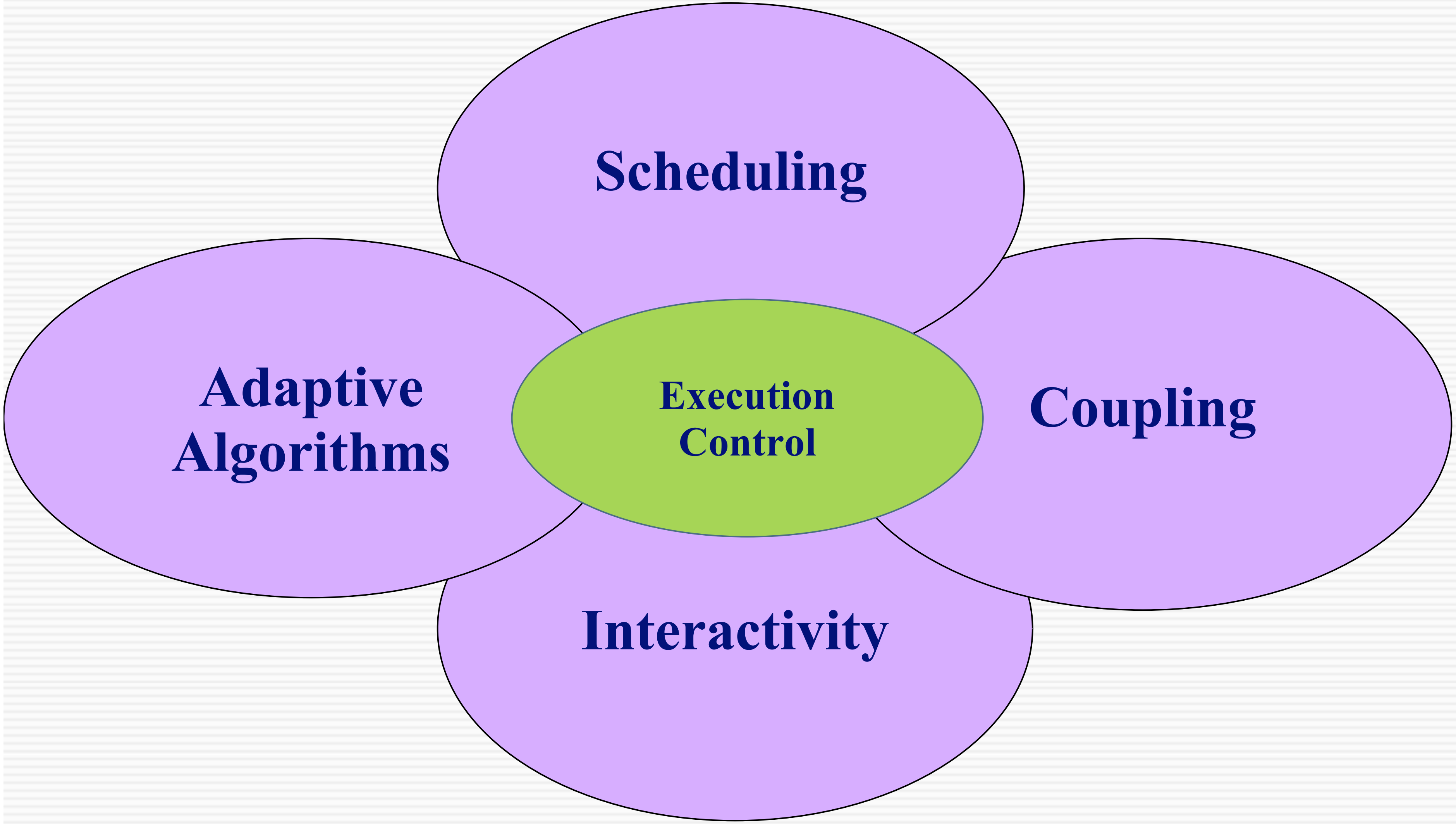
Today:

- Introduction
- Some Basics on Scheduling Theory
- Multicriteria Mapping/scheduling

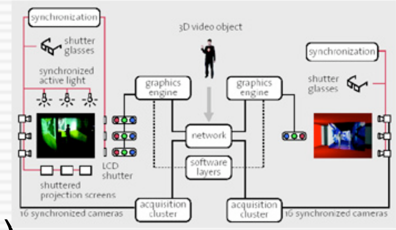
Tomorrow:

- Adaptive Algorithms: a Classification
- Work Stealing: basics on Theory and Implementation
- Processors oblivious parallel algorithms
- Anytime Work Stealing

The Moais Group

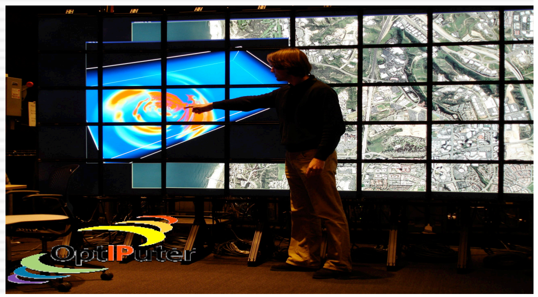


New Parallel Supports (Large ones)



- Clusters:
  - 72% of top 500 machines
  - Trends: more processing units, faster networks (PCI- Express)
  - Heterogeneous (CPUs, GPUs, FPGAs)

- Grids:
  - Heterogeneous networks
  - Heterogeneous administration policies
  - Resource Volatility
- Virtual Reality/Visualization Clusters:
  - Virtual Reality, Scientific Visualization and Computational Steering
  - PC clusters + graphics cards + multiple I/O devices (cameras, 3D trackers, multi-projector displays)
- Interactive Grids:
  - Grid + very high performance networks (optical networks) + high prformance I/O devices (Ex. Optiputer)

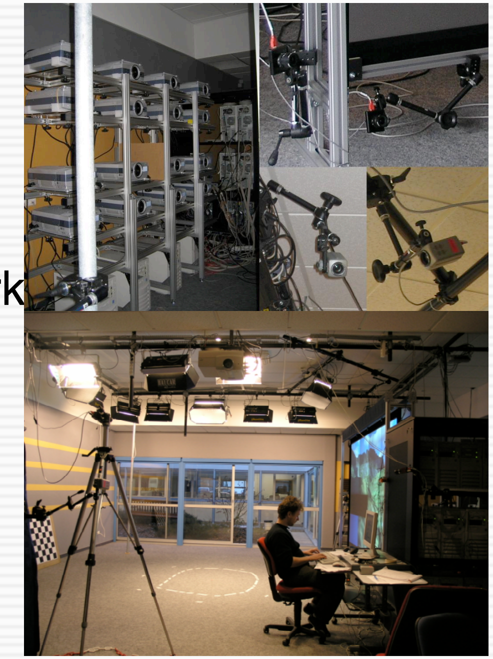


New Parallel Supports (small ones)

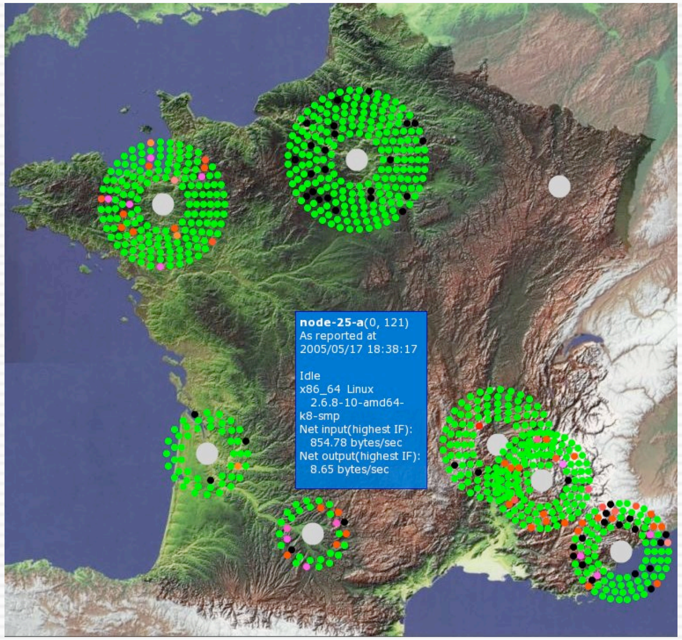
- Commodity SMPs:
  - 8 way PCs equipped with multi-core processors (AMD Hypertransport)

- Multi-core architectures:
  - Dual Core processors (Opterons, Itanium, etc.)
  - Dual Core graphics processors (and programmable: Shaders)
  - Heteregoneous multi-cores (Cells)
  - MPSoCs (Multi-Processor Systems-on-Chips)

Moais Plateforms

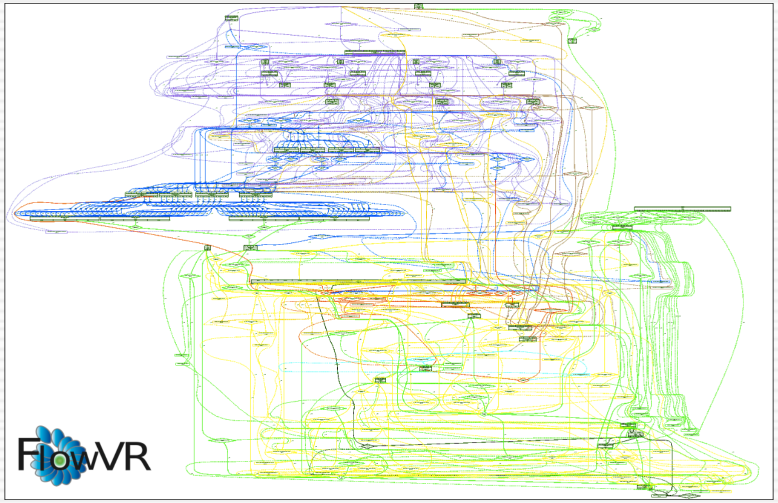


- Icluster 2 :
  - 110 dual Itanium 2 processors with Myrinet network
- GrImage (“Grappe” and Image):
  - Camera Network
  - 54 processors (dual processor cluster)
  - Dual gigabits network
  - 16 projectors display wall



- Grids:
  - Regional: Ciment
  - National: Grid5000
    - Dedicated to CS experiments
- SMPs:
  - 8-way Itanium (Bull novascale)
  - 8-way dual-core Opteron + 2 GPUs
- MPSoCs
  - Collaborations with ST Microelectronics

Moais Softwares

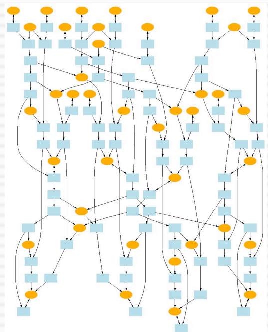


FlowVR (flowvr.sf.net)

- Dedicated to interactive applications
- Static Macro-dataflow
- Parallel Code coupling

Kaapi (kaapi.gforce.inria.fr)

- Work stealing (SMP and Clusters)
- Dynamics Macro-dataflow
- Fault Tolerance (add/del resources)



Kaapi

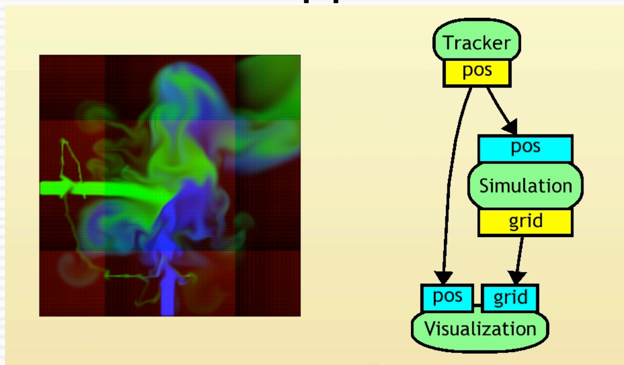
Oar (oar.imag.fr)

- Batch scheduler (Clusters and Grids)
- Developed by the Mescal group
- A framework for testing new scheduling algorithms

Some Basic on Scheduling Theory

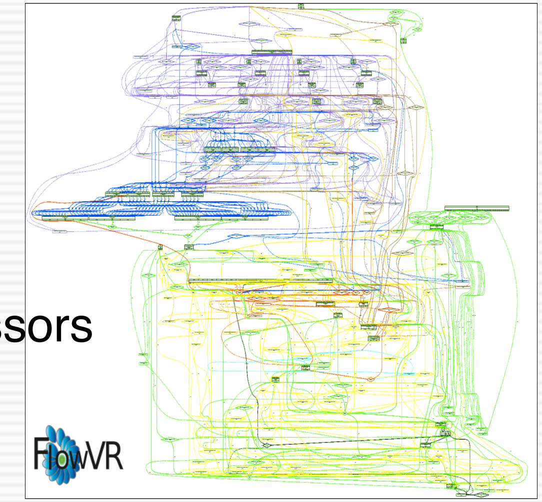
Parallel Interactive App.

- Human in the loop
- Parallel machines (cluster) to enable large interactive applications
- Two main performance criteria:
  - Frequency (refresh rate)
    - Visualization: 30-60 Hz
    - Haptic : 1000 Hz
  - Latency (makespan for one iteration)
    - Object handling: 75 ms



- A classical programming approach: data-flow model
  - Application = static graph
    - Edges: FIFO connections for data transfert
    - Vertices: tasks consuming and producing data
    - Source vertices: sample input signal (cameras)
    - Sink vertices: output signal (projector)
- One challenge:

  Good mapping and scheduling of tasks on processors



Video

Frequency and Latency

Question

Can we optimize the frequency and latency independently ?

Theorem

For an unbounded number of identical processors, no communiction cost, any mapping with one task per processor is optimal for both the latency and frequency.

Idea of Proof

Frequency: given by the slowest module

Latency: length of the critical path

A Multicriteria Problem

Theorem

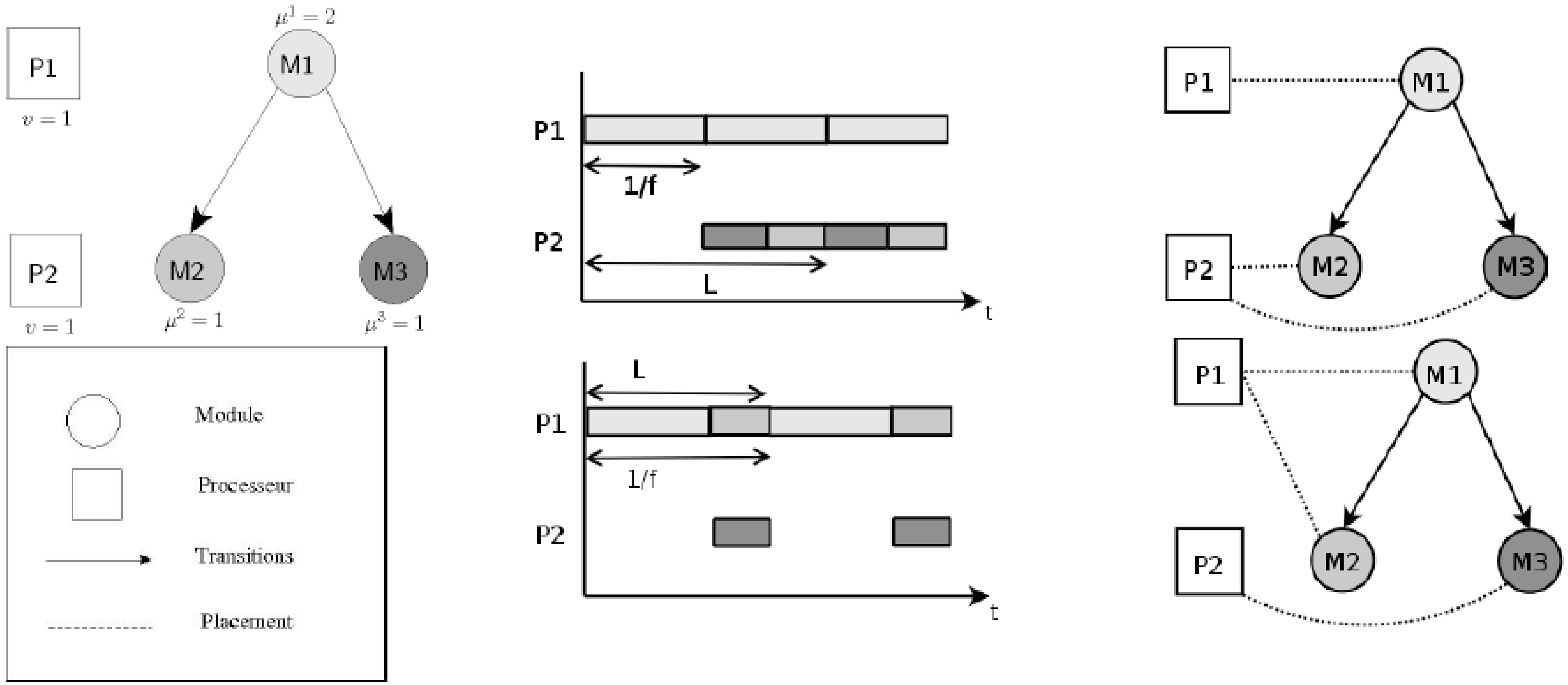
If at least one of the following holds:

- Bounded number of processors
- Processors have different speeds
- Communication cost between processors is not nul

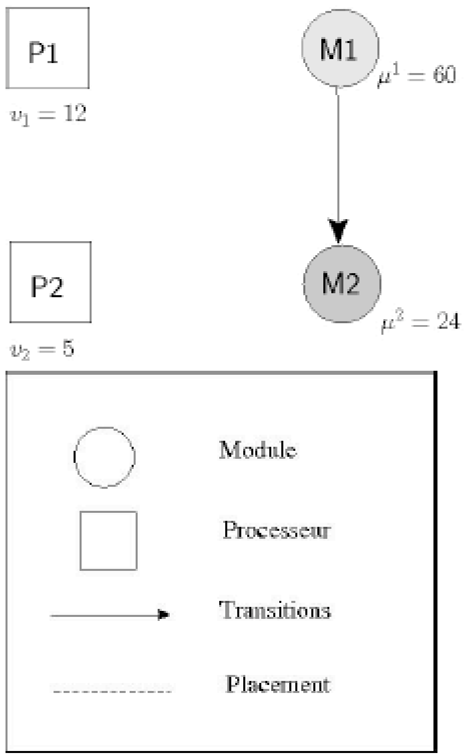
then for some applications there exist no mapping that optimize both, the latency and the frequency.

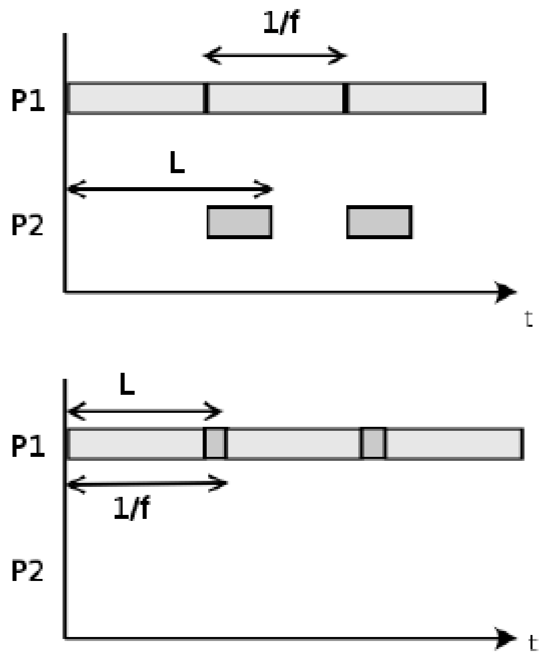
Proof : We just have to identify three examples.

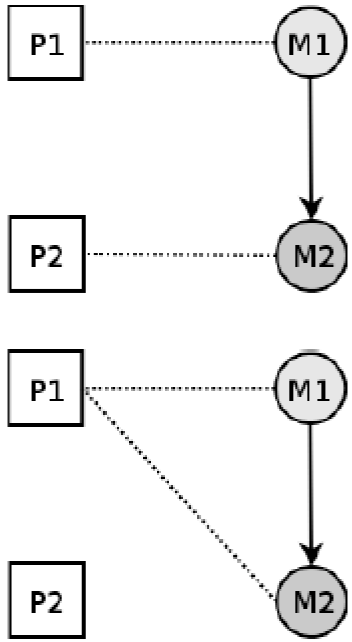
Bounded Number of Proc.



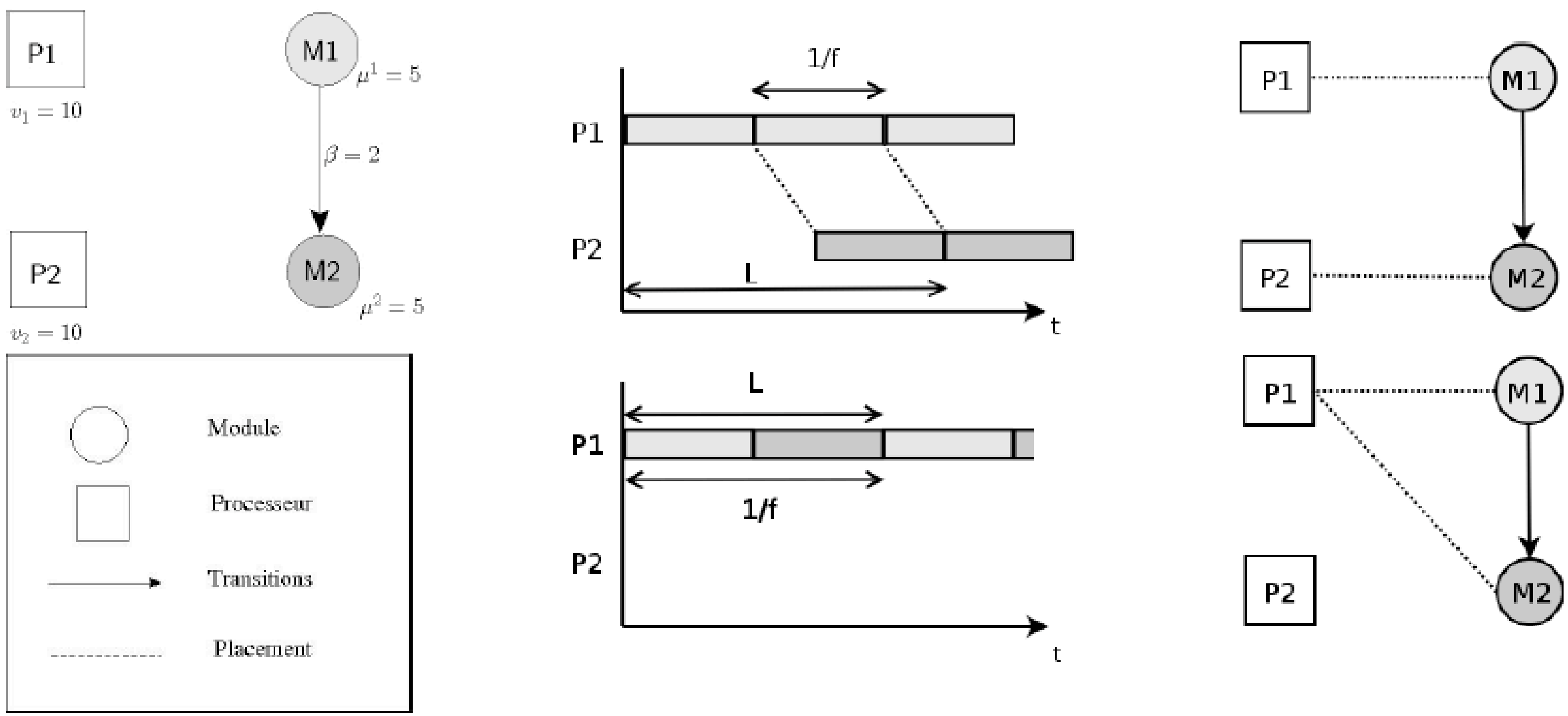
Different Processor Speeds







Communication Cost

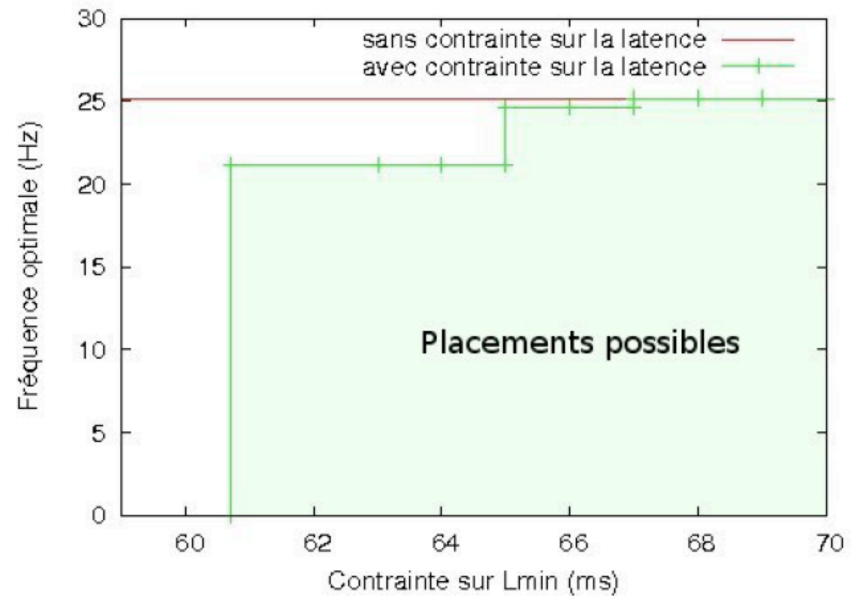


Mapping

Solving the multicriteria mapping:

Optimize one parameter while a bound is set on the other.

How to chose the “best” Latency/frequency tradeoff: A user decision.



Preliminary results on a simple example using simple heuristics

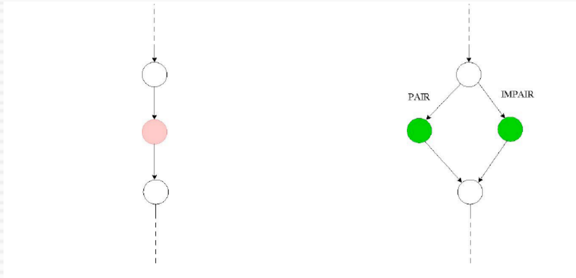
Perspectives

Today we are far from being able to compute mappings for real applications (hundred of tasks)

Other parameters the mapping could take advantage of:

Stateless tasks:

- Duplicate the tasks if idle resources
- Improve frequency but not latency



Parallel Tasks:

- Give the mapping algorithm the ability to decide the number of processors assigned
- Can improve both frequency and latency (if parallelisation efficient)

Tasks implementing level of detail algorithms:

- The task adapt the quality of the result to the execution time it has been allowed to execute
- Can improve latency and frequency **but impair quality** (an other cirteria to take into account?)

Static mapping on an “average work load” but work load vary over time

(2 users bellow the camera network instead of one for instance).

Adaptive/Hybrid Algorithms: a Classification

- **What adaptation is ?**
- **Example 1: List Scheduling**
- **Example 2:**
  - Several algorithms to solve a same problem **f** : ***algo_f***$_{1}$, ***algo_f***$_{2}$, … ***algo_f***$_{k}$
  - Each *algo_f*$_{k}$ is recursive

algo_f$_{i}$ (n, …) {

….

f (n - 1, …) ;

….

f (n / 2, …) ;

…

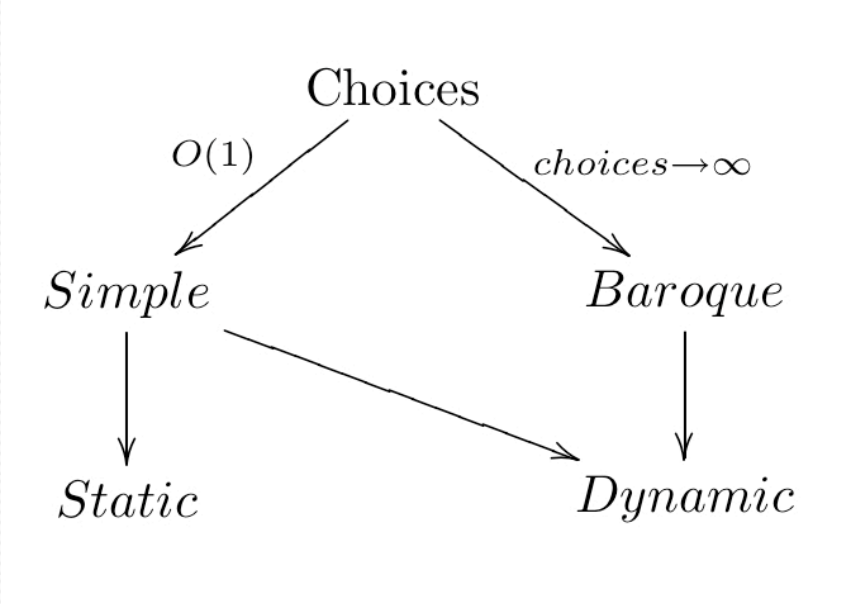
}

Adaptation:
choose *algo_f*$_{j}$ for each call to f

- Adaptation choice can be based on a variety of parameters: data size, cache size, number of processors, etc.

Adaptation has an overhead: how to manage it ?

Classification (1/2)

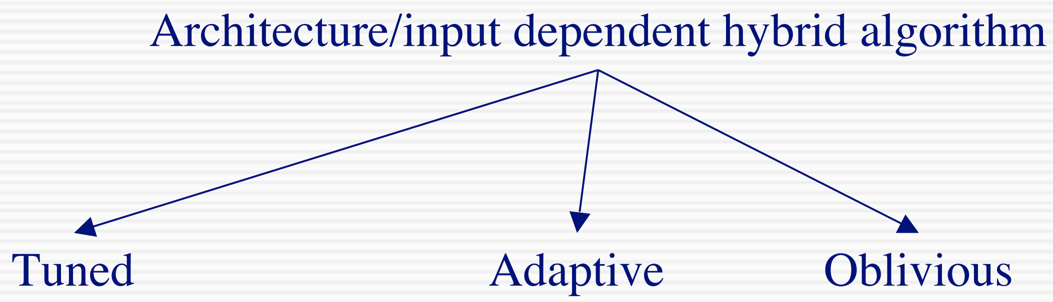


- ***Simple*** *hybrid* if bounded number of choices independent on the input size
  □ [eg parallel/sequential, block size in Atlas, …]
  Choices are either dynamic or pre-computed based on architecture properties.

- ***Baroque*** *hybrid* if unbounded number of choices (based on input sizes)
  [eg message size for hybrid collective communications, recursive splitting factors in FFTW]

  Choices are dynamic

Classification (2/2)



- **Tuned**: Strategic choices are based on ***static*** resource properties

[eg cache size, # processors,…]

[eg ATLAS and GOTO libraries, FFTW, LinBox/FFLAS]

- **Adaptive**:
  - Choices based on input properties or resource availability discovered at run-time
  - No machine or memory specific parameter analysis

[eg : idle processors, …]

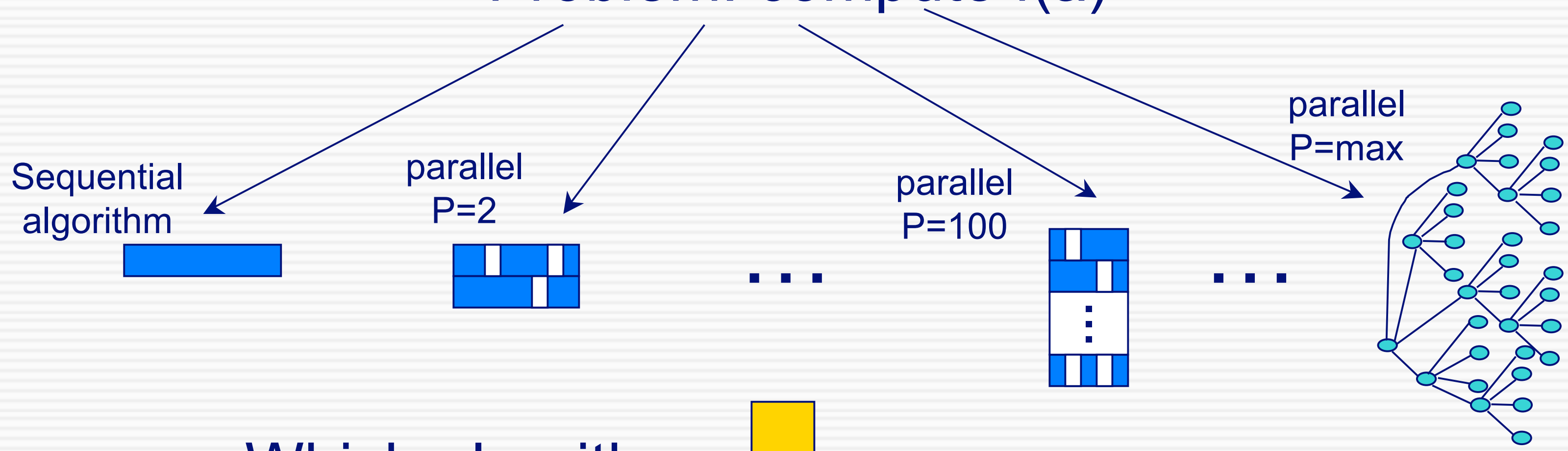
[eg work stealing]

- **Oblivious**: Control flow depends neither on particular input data values nor static properties of the resources

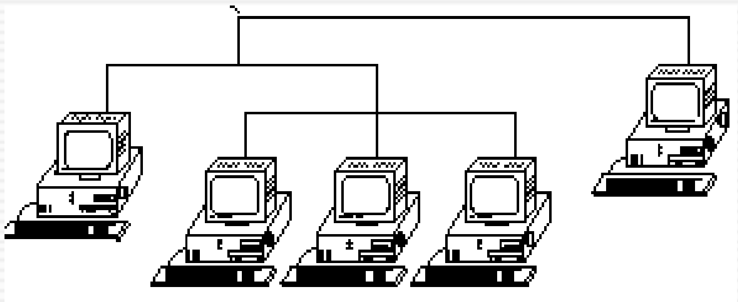
[eg cache-oblivious algorithm]

Adaptation in parallel algorithms

Problem: compute f(a)



Which algorithm to choose ?



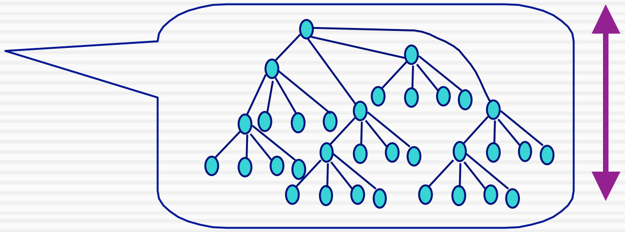
Heterogeneous network

Multi-user SMP server

Grid

Parallelism and efficiency

« **Work** »
W_1 = #operations
Time on 1 proc.



«**Depth** »
W_∞ = #ops on a critical path
Time on ∞ proc.

Problem : how to adapt the potential parallelism to the resources ?

Scheduling

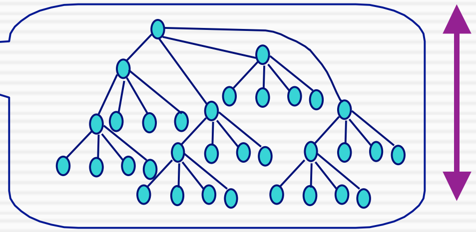
efficient policy (close to optimal) ⟷ control of the policy (realisation)

Difficult in general (coarse grain) **But easy if W_∞ small (fine grain)** $W_p = W_1/p + W_\infty$ [List scheduling, Graham69]	Expensive in general (fine grain) **But small overhead if coarse grain**

=> to have T_∞ small with coarse grain control

Work-stealing (1/2)

« **Work** »
W_1 = #total operations performed

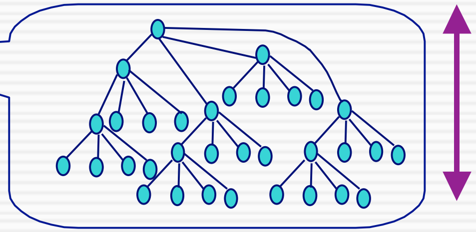


«**Depth** »
W_∞ = #ops on critical path

- **List scheduling :** processors get their work from a centralized list
- **Workstealing :** distributed and randomized list scheduling
  - Each processor manages locally the tasks it creates
  - When idle, a processor steals the oldest ready task on a remote -non idle- victim processor (randomly chosen)

Work-stealing (2/2)

« **Work** »
W_1 = #total operations performed



«**Depth** »
W_∞ = #ops on a critical path

(parallel time on ∞ resources)

- **Guarantees :**

$$T_p \leq \frac{W_1}{p.\Pi_{ave}} + O\left(\frac{W_\infty}{\Pi_{ave}}\right)$$

Π_{ave}: Processor average speeds [Bender-Rabin02]

$$\#\text{success steals} \leq O(pW_\infty)$$

[Blumofe 98, Narlikar 01, Bender 02]

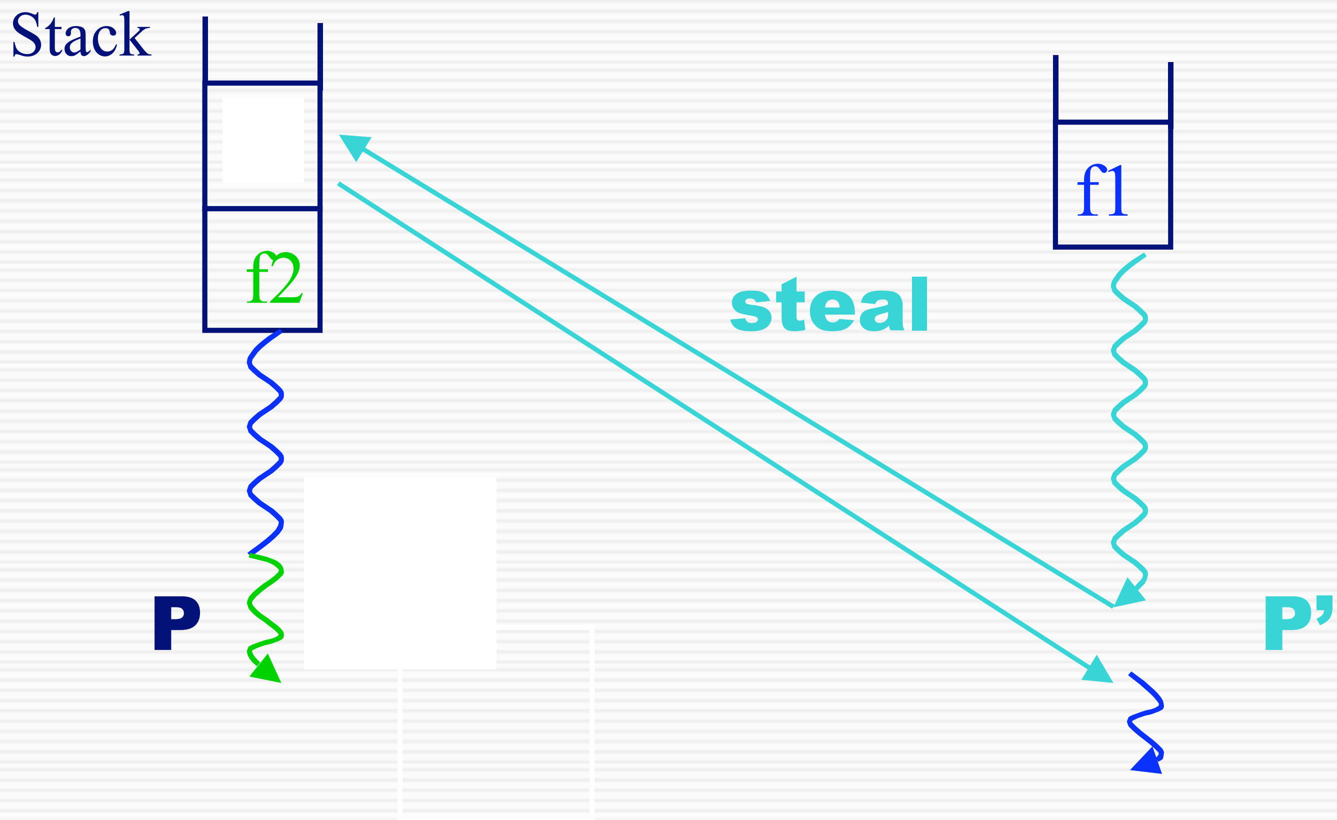
Near-optimal adaptive schedule if $W_\infty \lll W_1$ (with a good probability)

Implementation of Work Stealing

f1() { ….

fork f2 ; …

}



Implementation of Work-stealing

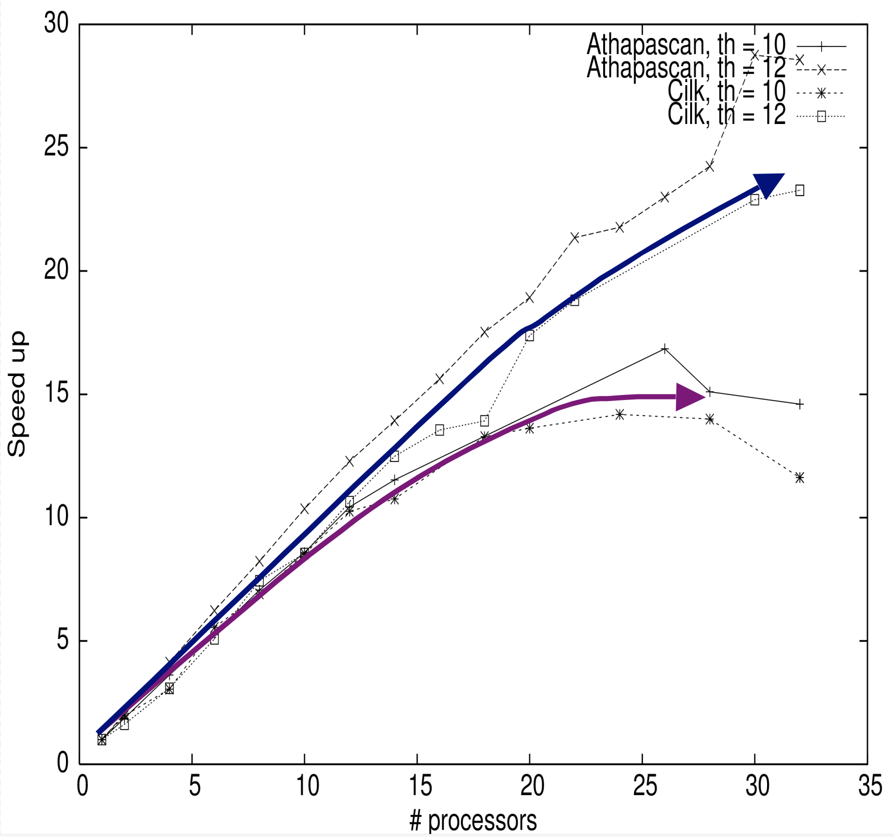
- Goal: Reduce the overheads
  - **Stealing overheads**
  - **Local task queue management overheads**

  - **Work first principle: scheduling overhead on the steal operations (only $O(pW_\infty)$ steals)**
  - **Depth first local computation to save memory**
  - **Compare&Swap atomic operations**

- Some work stealing libraries:

  Cilk, Charm ++, Satin, Kaapi

Experimentation: knary benchmark



SMP Architecture

Origin 3800 (32 procs)

Cilk / Athapascan

#procs	Speed-Up
8	7,83
16	15,6
32	30,9
64	59,2
100	90,1

Distributed Archi.

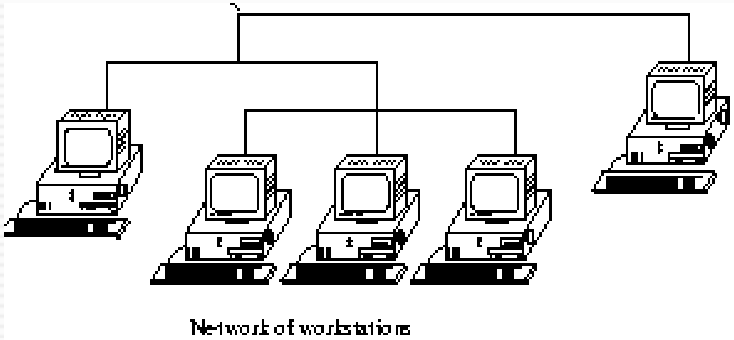
iCluster

Athapascan

$T_s = 2397$ s $\approx$ $T_1 = 2435$

Processor-oblivious algorithms

Dynamic architecture : non-fixed number of resources, variable speeds
eg: grid, SMP server in multi-users mode,….



=> motivates « **processor-oblivious** » parallel algorithm that :

+ is **independent** from the underlying architecture:
  no reference to *p* nor $\Pi_i(t)$ = *speed of processor i at time t nor …*

+ on a given architecture, has **performance guarantees** :
  behaves as well as an optimal (off-line, non-oblivious) one

Work-stealing and adaptability

- **Work-stealing ensures allocation of processors to tasks transparently to the application with provable performances**
  - Support to addition of new resources
  - Support to resilience of resources and fault-tolerance (crash faults, network, …)
    - Checkpoint/restart mechanisms with provable performances [Porch, Kaapi, …]

- **“Baroque hybrid” adaptation:** there is an -implicit- dynamic choice between two algorithms
  - **a sequential (local) algorithm :** depth-first (default choice)
  - **A parallel algorithm :** breadth-first
  - Choice is performed at runtime, depending on resource idleness

- **Well suited to applications where a fine grain parallel algorithm is also a good sequential algorithm** [Cilk]:
  - Parallel Divide&Conquer computations
  - Tree searching, Branch&X …

-> suited when both sequential and parallel algorithms perform (almost) the **same number** of operations

Processor Oblivious Algorithm

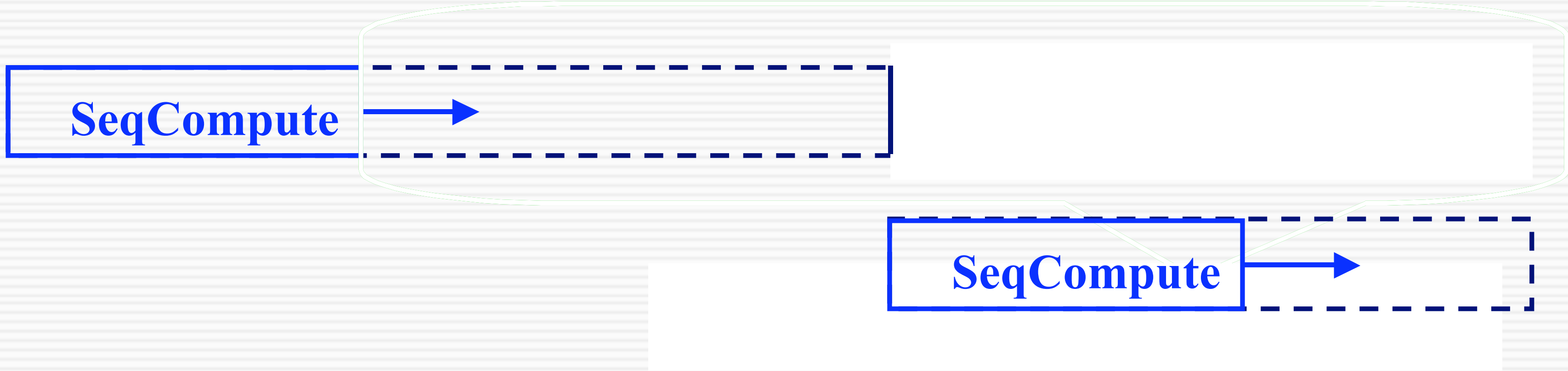
Based on the Work-first principle :

Executes always a sequential algorithm to reduce parallelism overhead

⇒ use parallel algorithm only if a processor becomes **idle** (ie *steals*) by ***extracting parallelism*** from a sequential computation

Hypothesis : two algorithms :

- - 1 sequential : ***SeqCompute***
  - 1 parallel : ***LastPartComputation*** : at any time, it is possible to extract parallelism from the remaining computations of the sequential algorithm



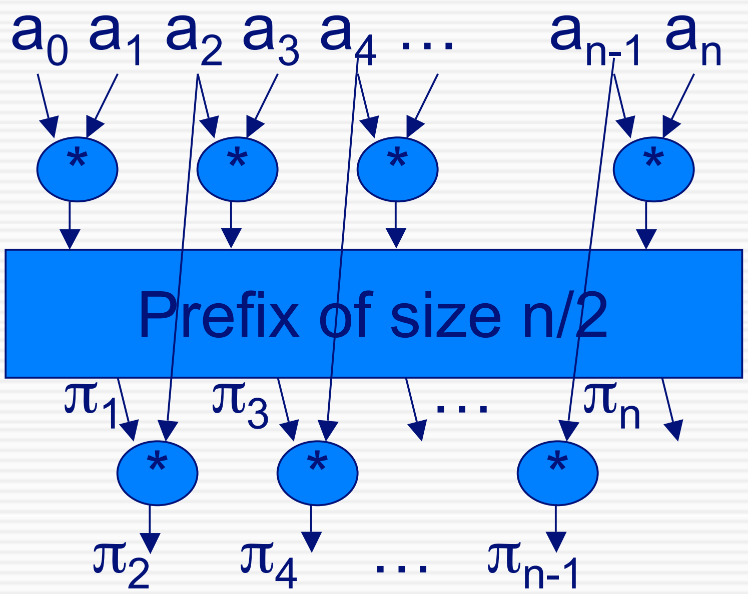
Prefix computation

- Prefix problem :
  - input : $a_0, a_1, \ldots, a_n$
  - output : $\pi_0, \pi_1, \ldots, \pi_n$ with $\pi_i = \prod_{k=0}^{i} a_k$
- Sequential algorithm :

```
for (i= 0 ; i <= n;  i++ )  π[ i ] = π[ i – 1 ] * a [ i ] ;
```

performs $W_1 = W_\infty = $ **n** *operations*

- Fine grain optimal parallel algorithm [Ladner-Fischer]:



Critical path W_∞ =2. *log* n

but performs W_1 = **2.n ops**

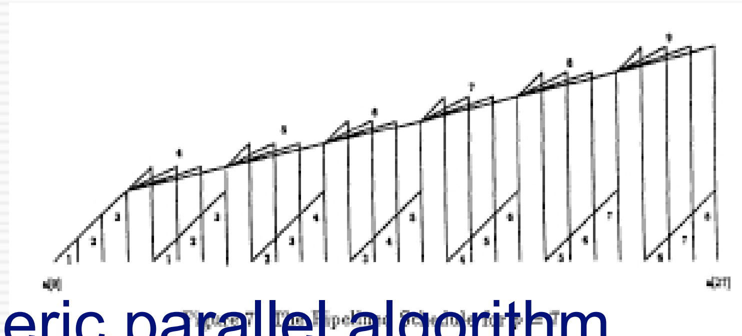
Twice more expensive than the sequential …

Prefix computation

- Lower bound: any parallel prefix algorithm runs on p processors in time at least:

$$T_p \geq \frac{2n}{p+1}$$

lower bound : block algorithm + pipeline [Nicolau&al. 1996]

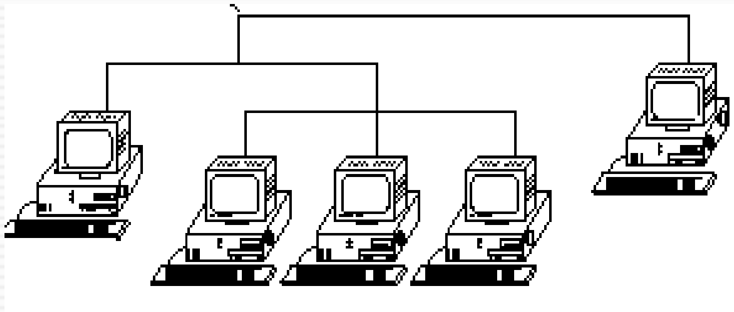


–**Question** : How to design a generic parallel algorithm, independent from the architecture, that achieves optimal performance on any given architecture ?

–> to design a processor oblivious hybrid **algorithm** where **scheduling** suits the number of operations performed to the architecture

Architecture model

- Heterogeneous processors with changing speed [Bender-Rabin02]



Network of workstations

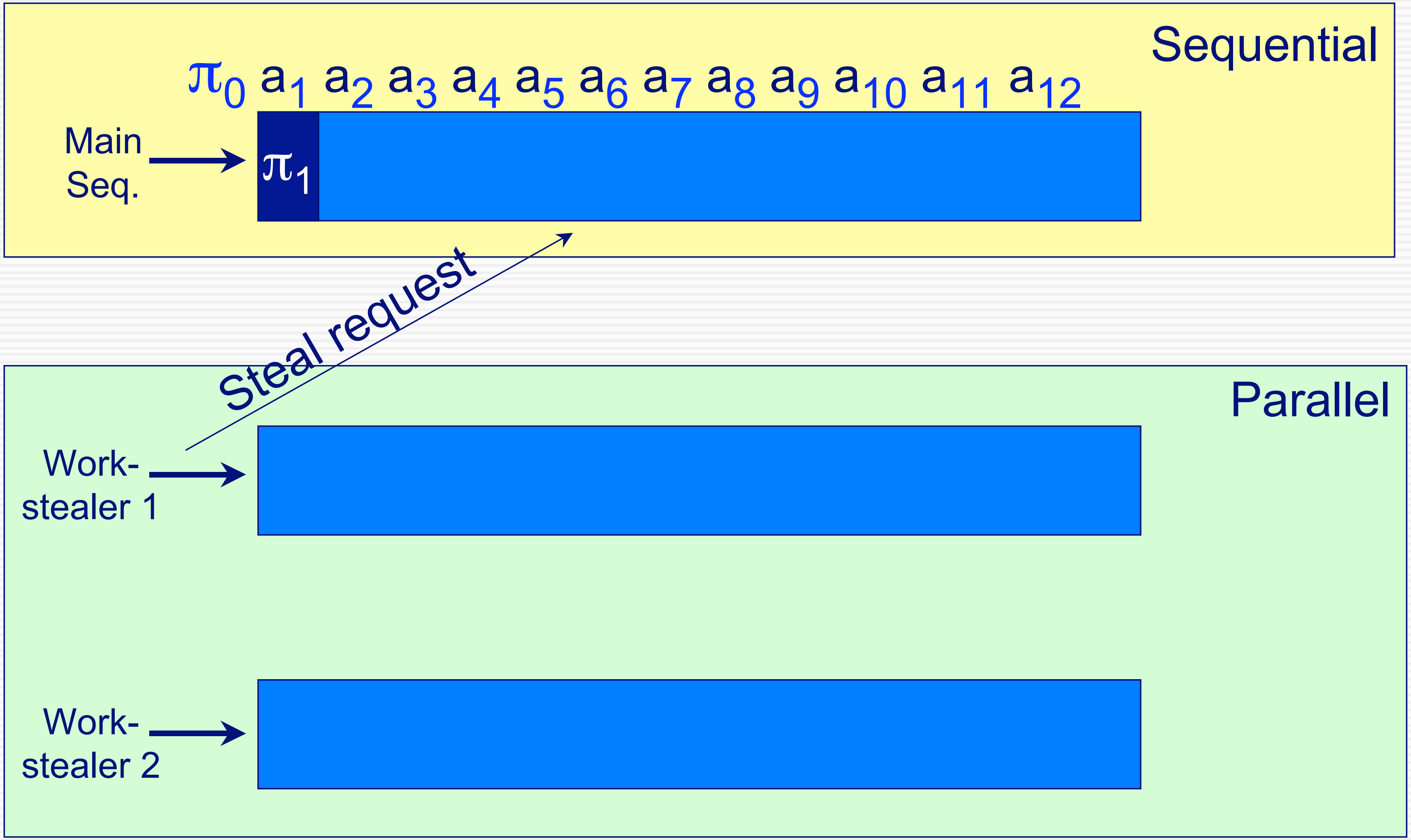
=> ***$\Pi_i(t)$ = instantaneous speed of processor i*** *at time t in #operations per second*

- Average speed per processor for a computation with duration T :

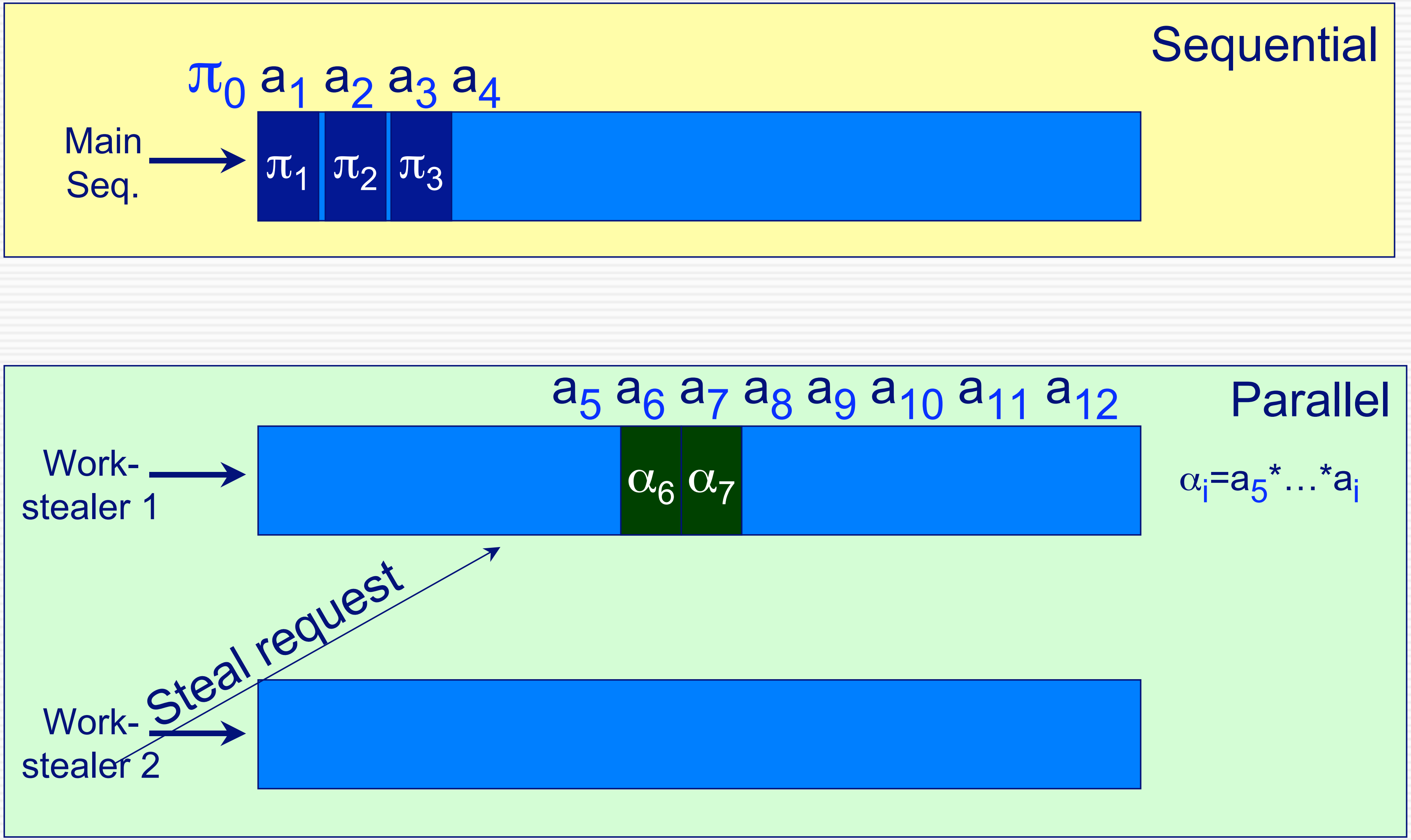
$$\pi_{ave} = \frac{\sum_{i=1..p} \sum_{t=0..T} \Pi_i(t)}{p.T}$$

- Lower bound for the time of prefix computation :

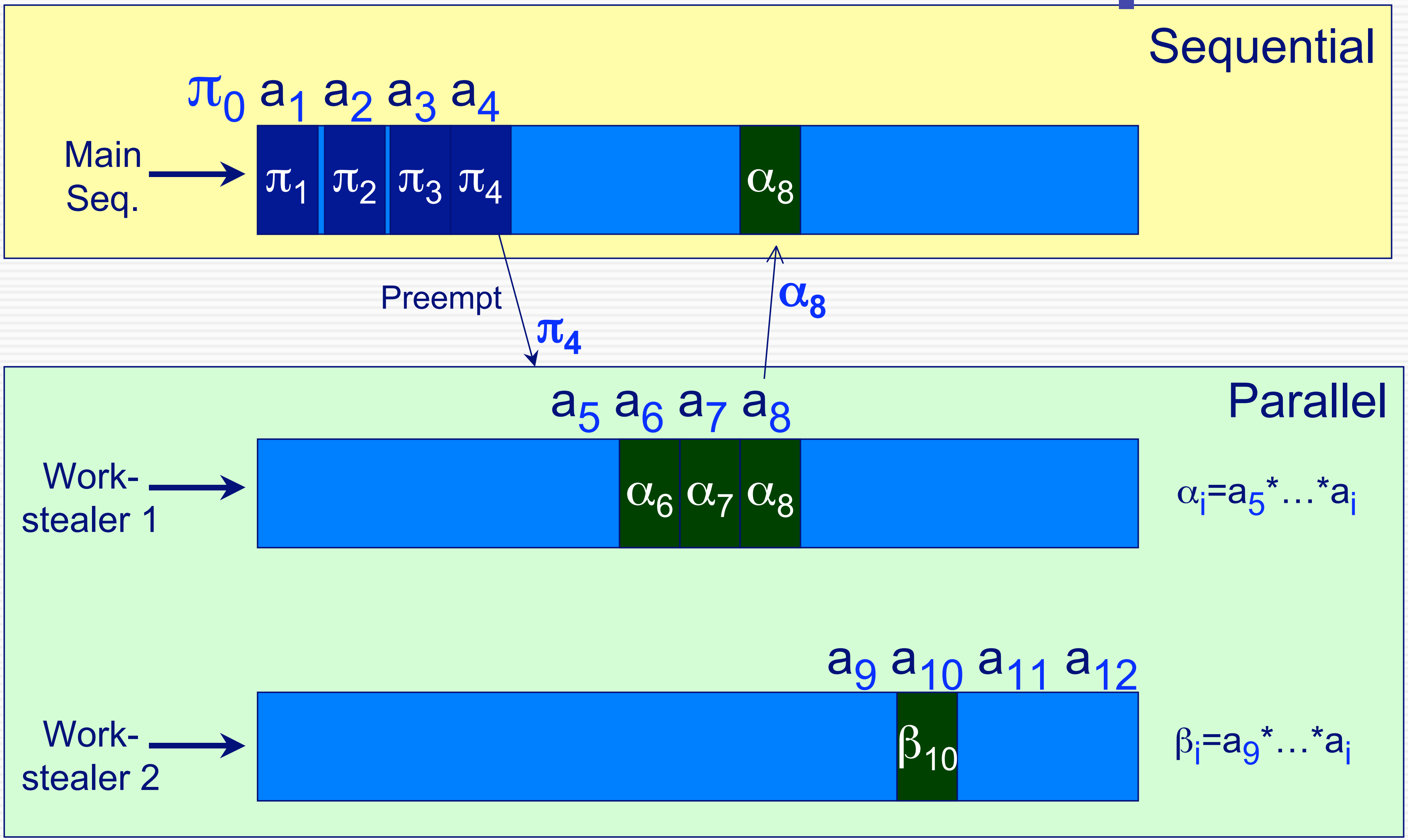
P-Oblivious Prefix on 3 proc.



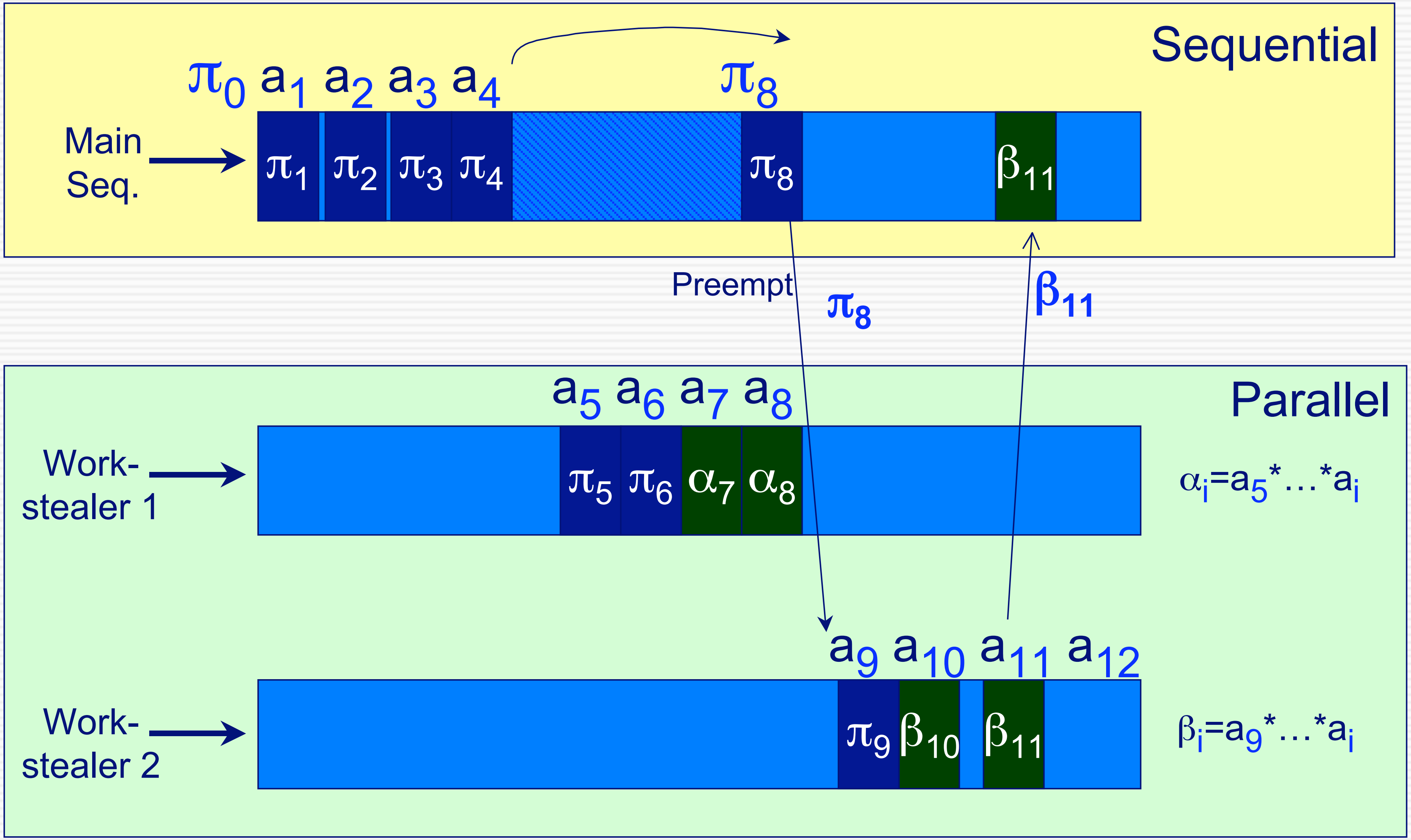
P-Oblivious Prefix on 3 proc.



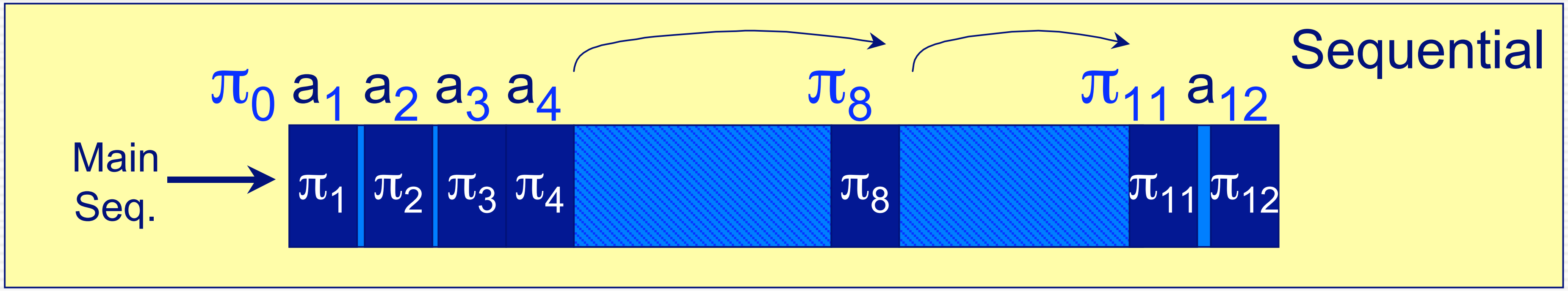
P-Oblivious Prefix on 3 proc.

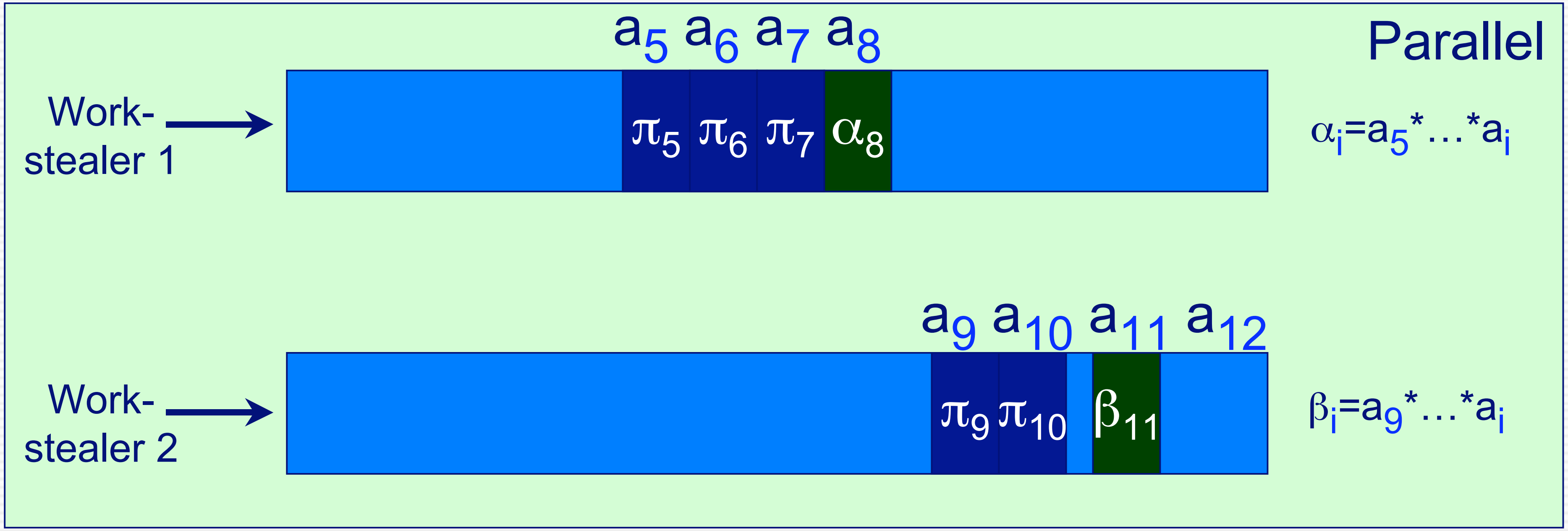


P-Oblivious Prefix on 3 proc.

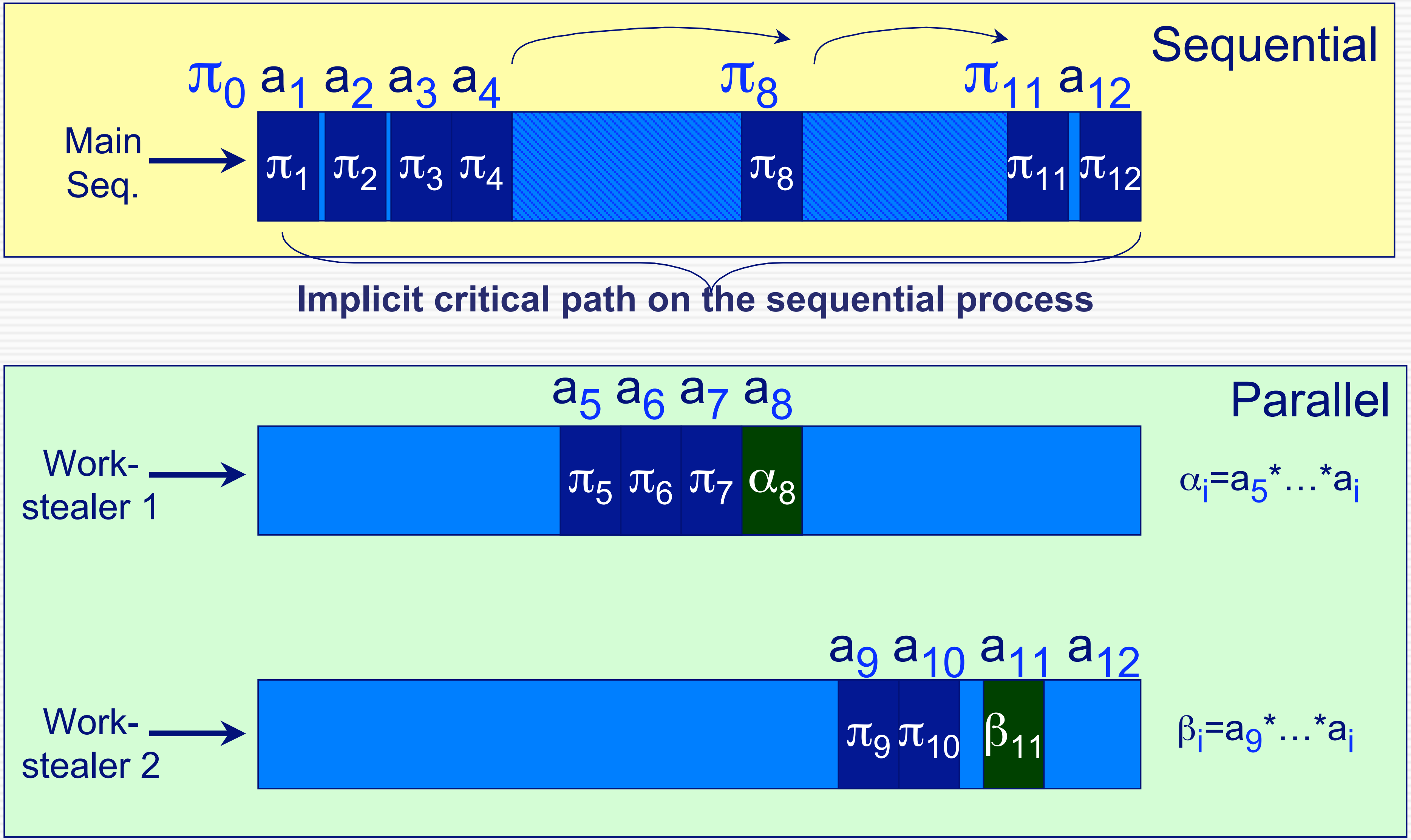


P-Oblivious Prefix on 3 proc.





P-Oblivious Prefix on 3 proc.



Analysis of the algorithm

- Execution time $\leq \frac{2n}{(p+1).\Pi_{ave}} + O\left(\frac{\log n}{\Pi_{ave}}\right)$

  Lower bound

- Sketch of the proof :

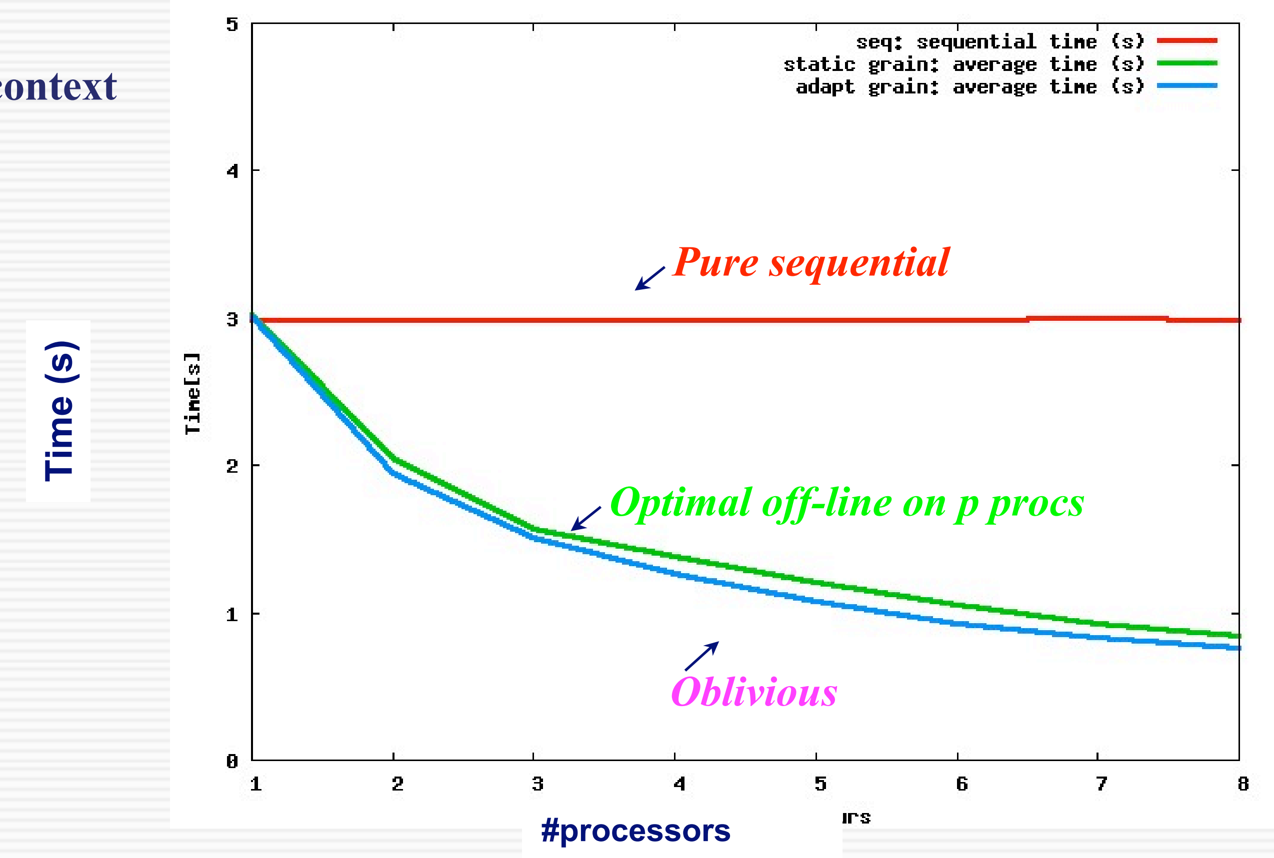
  Dynamic coupling of two algorithms that complete simultaneously:
  - Sequential: (optimal) number of operations S on one processor
  - Parallel : minimal time but performs X operations on other processors
    - dynamic splitting always possible till finest grain BUT local sequential
      - Critical path small (eg : log X)
      - Each non constant time task can potentially be splitted (variable speeds)

  - Algorithmic scheme ensures $T_s = T_p + O(\log X)$
    => enables to bound the whole number X of operations performed
    and the overhead of parallelism = (s+X) - #ops_optimal

Results 1/2

Prefix sum of 8.10^6 double on a SMP 8 procs (IA64 1.5GHz/ linux)

Single user context



Single-usercontext : processor-oblivious prefix achieves near-optimal performance :

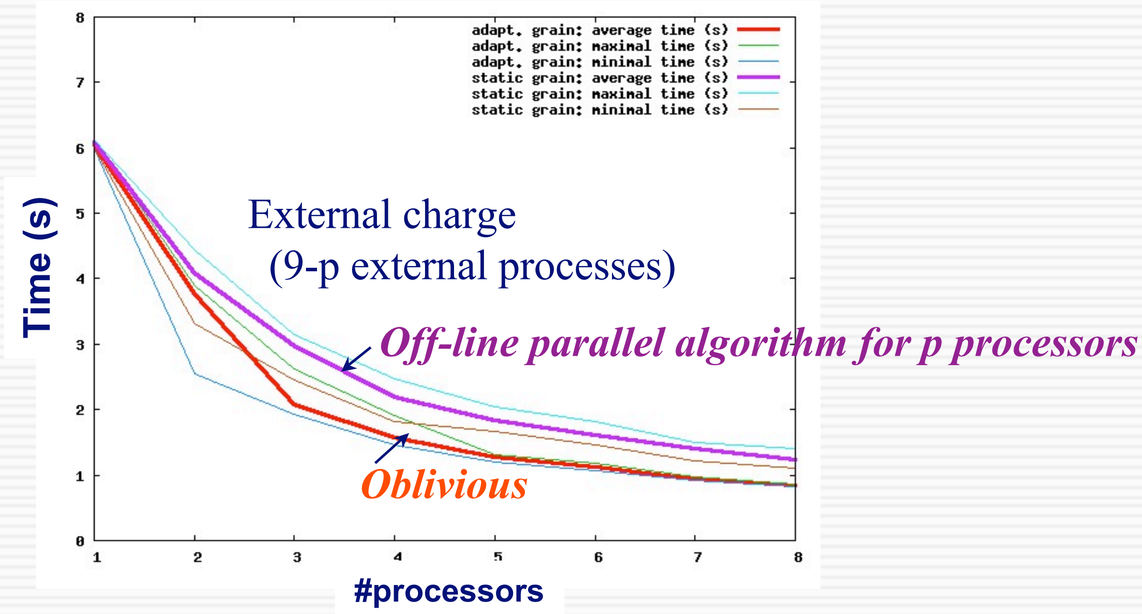
- close to the lower bound both on 1 proc and on p processors

- Less sensitive to system overhead : even better than the theoretically "optimal" off-line parallel algorithm on p processors

Results 2/2

Prefix sum of 8.10^6 double on a SMP 8 procs (IA64 1.5GHz/ linux)

Multi-user context :



Multi-user context :

Additional external charge: (9-p) additional external dummy processes are concurrently executed

Processor-oblivious prefix computation is always the fastest

15% benefit over a parallel algorithm for p processors with off-line schedule,

Work Stealing: Summary

- **Classical work stealing: Adaptive hybrid algorithm**
  - Implicitly mix a parallel and sequential algorithm
  - Efficient if parallel and sequential algorithms perform about the same amount of operations

- **Processor Oblivious**
  - Explicit mix a parallel and sequential algorithm (may execute different amount of operations)
  - Oblivious: optimal whatever the execution contect is.

Other oblivious parallel algorithms:

Iterated product, gzip / compression, MPEG-4 / H264

Anytime Work Stealing

Anytime Algorithm:

- Can be stopped at any time (with a result)
- Result quality improve has more time is allocated

In Computer graphics anytime algorithms are common:

Level of Detail algorithms (time budget, triangle budget, etc…)

Example: Progressive texture loading, triangle decimation (Google Earth)

Anytime Work Stealing:

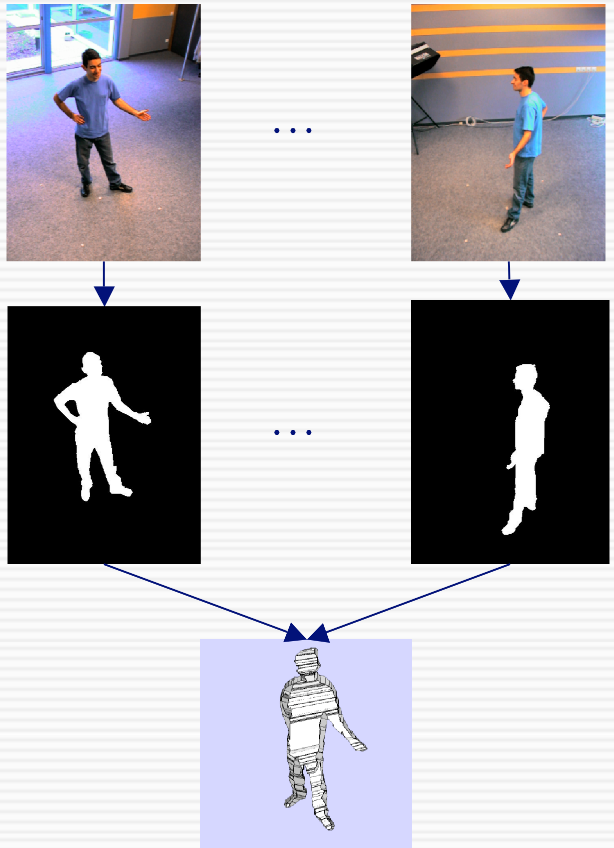
- Use parallelism to get faster, but keep anyway the ability to stop computations at anytime.
- Work stealing: adapt to input irregularities.

Example: Parallel Octree computation for 3D Modeling

Parallel 3D Modeling

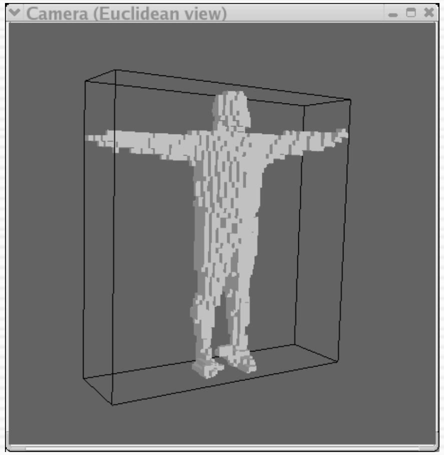
3D Modeling :
build a 3D model of a scene from a set of calibrated images

On-line 3D modeling for interactions: 3D modeling from multiple video streams (30 fps)



Octree Carving

A classical recursive anytime 3D modeling algorithm.



Init: 1 grey cube (cover the acquisition space)

Iterate:

while (grey cubes available && time left)
Select a grey cube
Project cube in each image
If inside each silhouette, cube is black
if outside one silhouette, cube is transparent
else split the cube in 8 grey su-cubes
end

Tree shape depends on input data.

Octree Carving

Parallel Octree:

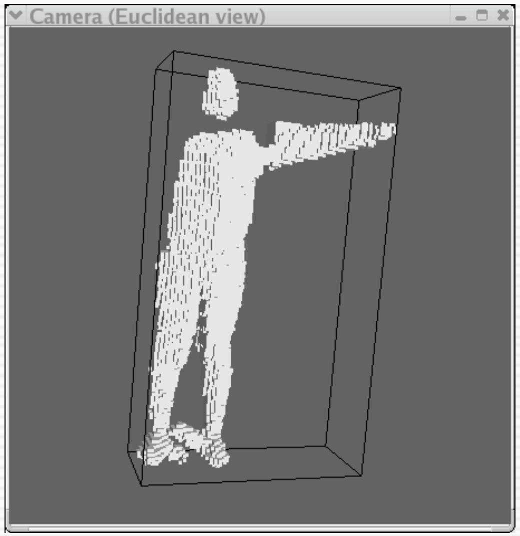
-Work stealing to avoid idle processors (adapt to data irregularities)

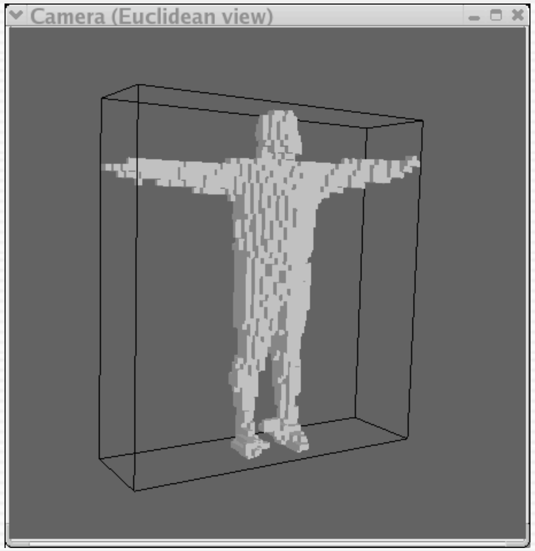
-Small critical path, while huge amount of work (eg. $W_\infty = 8$, $W_1 = 164\ 000$)

-Same amount of work for sequential and parallel algorithms

- Octree need to be “balanced” when stopping:
  - Width first stealing
  - Width first local computations
  - Synchronization barriers locking processors when progressing along W_∞

Unbalanced

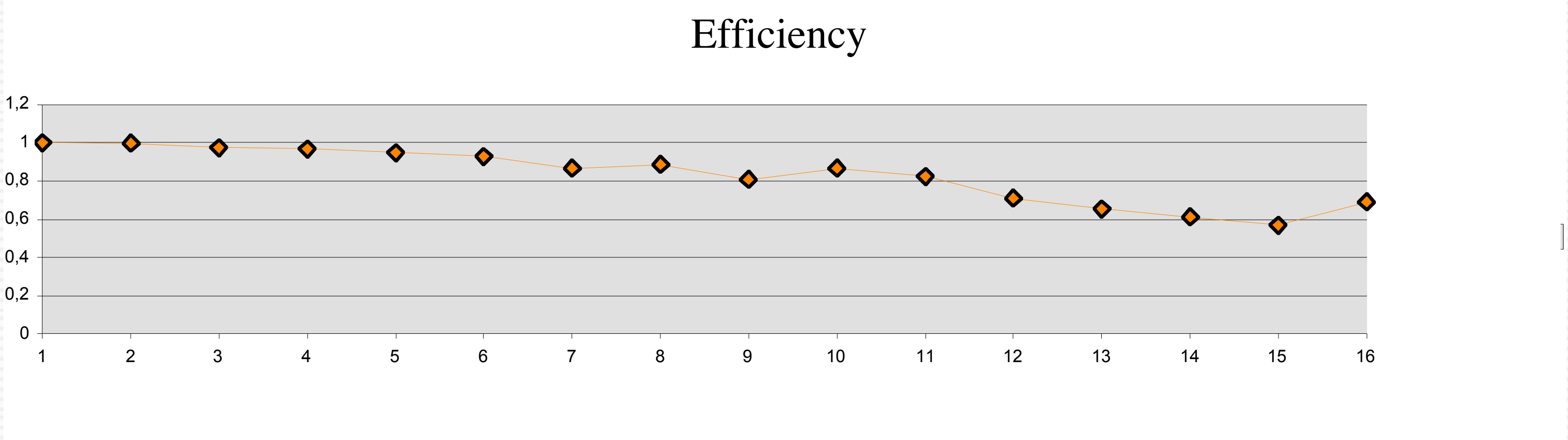


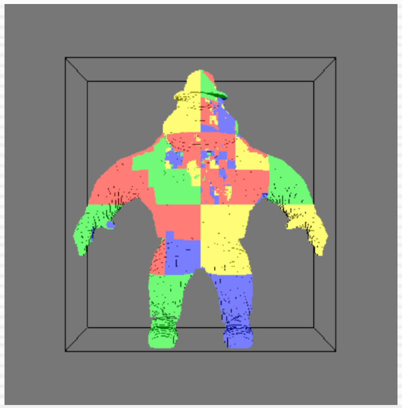


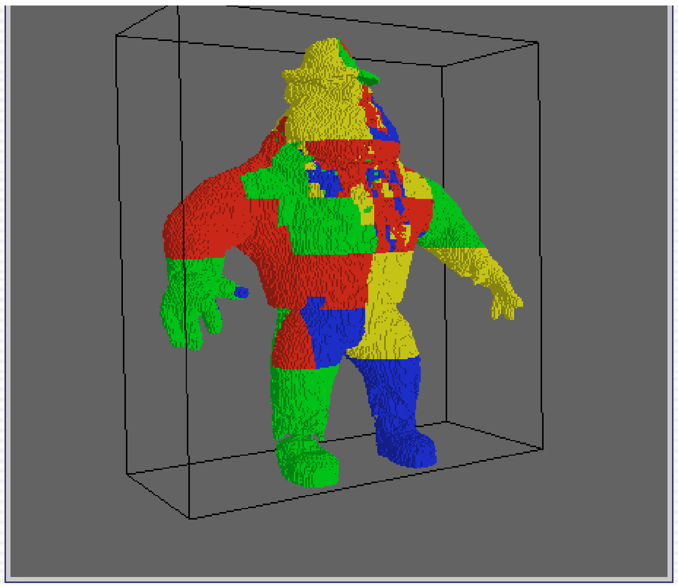
Balanced

Results

- 16 core Opteron machine, 64 images
- Sequential: 269 ms, 16 Cores: 24 ms
- 8 cores: about 100 steals (167 000 grey cells)







Conclusion

Classical Parallel algorithms (MPI-1):

Not well adapted to new supports:

- Resource volatility (grid, large clusters, multi-user environments)
- Data irregularities (interactive applications)

List Scheduling:

Adaptive algorithm with performance guarantee

But centralized ready task queue

Work Stealing:

Distributed task queues + Random steals

Efficient if

$$W_\infty <<< W_{1\ parallel} \text{ and } W_1 \approx W_{sequential}$$

Processor oblivious algorithm:

When W_1 very different from $W_{sequential}$

Hybrid a sequential and a parallel algorithm with a work sealing approach