Interactive computations and outsourcing

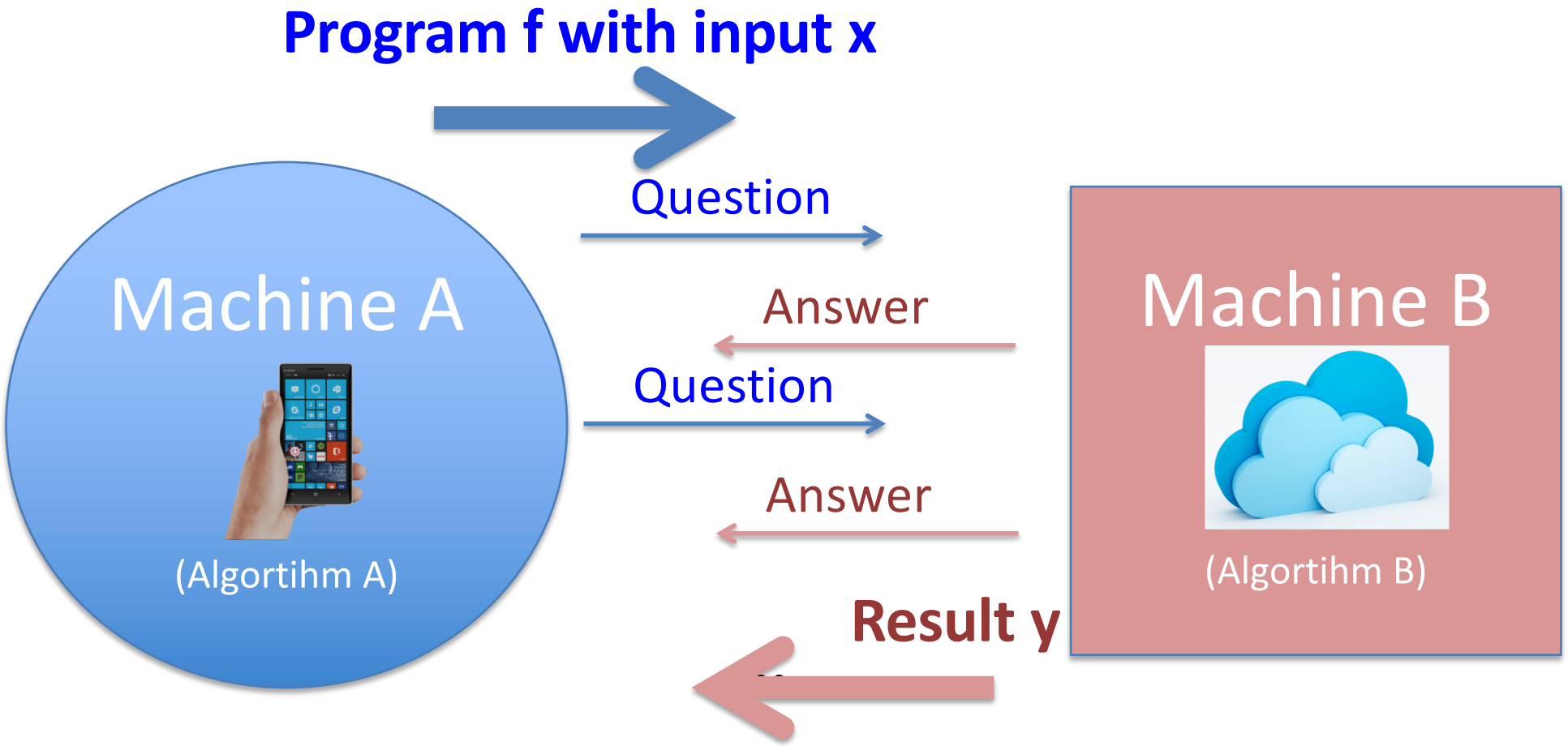
Jean-Louis Roch
Grenoble University, France

Grenoble INP -Ensimag, Univ. Grenoble Alpes, 2021/10/22

Distributed, heterogeneous (hybrid)

- Various computing abilities and levels of trust

Outsourcing



A kind of parallel algorithm, but heterogeneous "computing models"

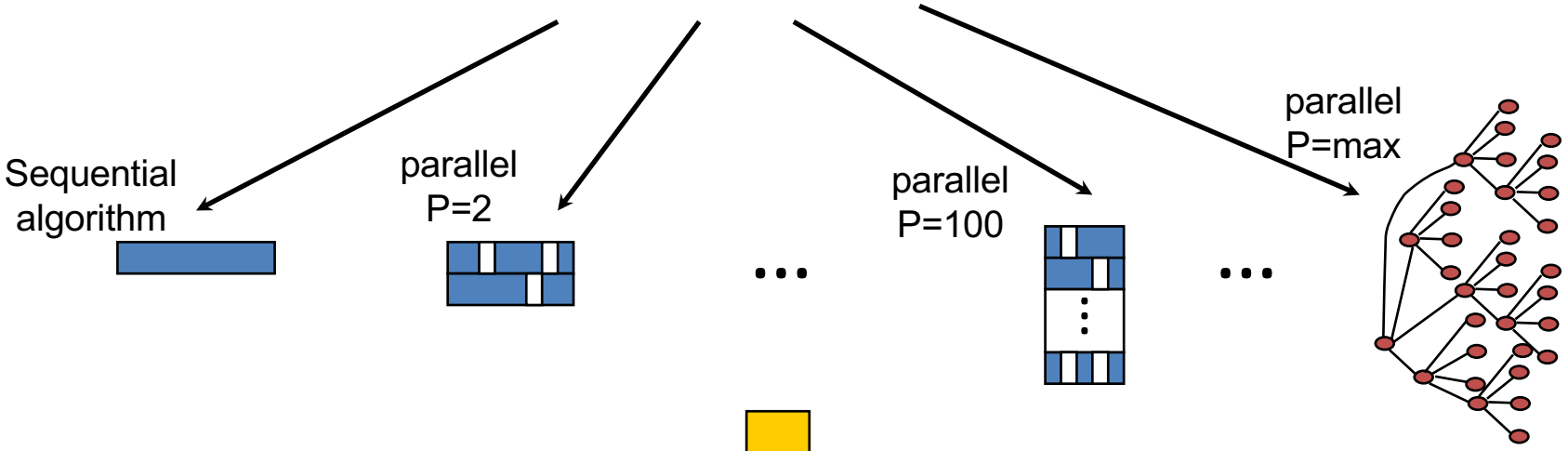
Why interactive algorithms ?

- For provable oblivious performances,

- For provable security (result correctness)

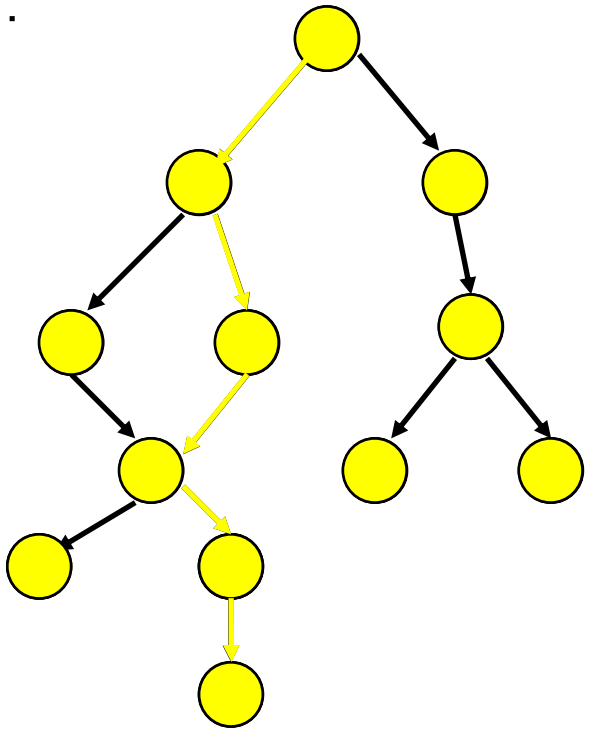
Towards oblivious algorithms

To design a single efficient algorithm
with provable performances on an arbitrary architecture



Which algorithm
to choose ?

:



Basic notations: Work and depth

"Work" W = #total number operations performed

"Depth" D = #operations on a critical path

(~parallel "time" on ∞ resources)

Relation to execution time T(p, Π)

For any greedy *maximum utilization* schedule Graham69, Brent70, Jaffe80, Bender-Rabin02]

$$\frac{1}{\Pi_{ave}} Max\left(\frac{W}{p};D\right) \le Time(p,\Pi) \le \frac{1}{\Pi_{ave}}\left(\frac{W}{p}+D\right)$$

- **There exist schedulers that reach the bound :** $\frac{1}{\Pi_{ave}}\left(\frac{W}{p}+O(D)\right)$

Work-Stealing: a basis to design processor-oblivious algorithm

- **Work-stealing =** *"A decentralized thread scheduler: whenever a processor runs out of work, it steals work from a randomly chose processor." […, Leiserson 1998/Cilk, …]*
  interaction between idle resources and busy ones

- Scheduler that reach: $\frac{1}{\Pi_{ave}}\left(\frac{W}{p}+O(D)\right)$

  Moreover: if D small, few steals request [O(p.D) w.h.p.]

- *When both W and D are small, leads to oblivious parallel algorithm*
  - *both theoretical and practical [Cilk, TBB, Kaapi, …]*
- *But, in general, W and D are antagonist …*

Example: Parallel prefix

- **Prefix problem** :
  - input : $a_0, a_1, \ldots, a_n$
  - output : $\pi_1, \ldots, \pi_n$ with

$$\pi_i = \prod_{k=0}^{i} a_k$$

- **Sequential** algorithm :
  - for ($\pi[0] = a[0]$, i = 1 ; i <= n; i++) $\pi[\,i\,] = \pi[\,i-1\,] * a\,[\,i\,]$;

performs only* n *operations
(and minimal cache misses)

- Optimal parallelization on **p identical processors**:

Optimal time $T_p = 2n / (p+1)$

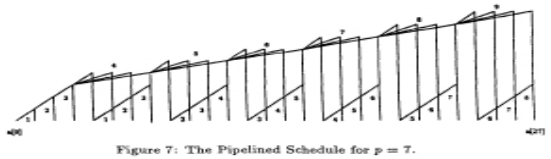
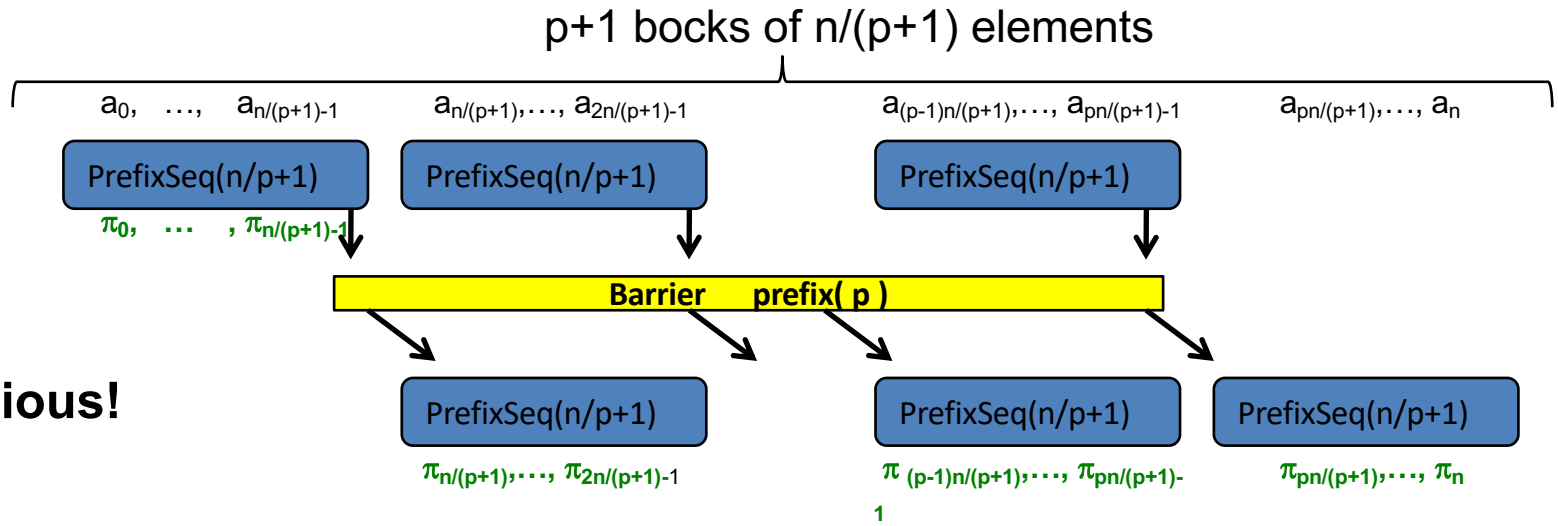


Figure 7: The Pipelined Schedule for $p = 7$.



- **Non oblivious!**

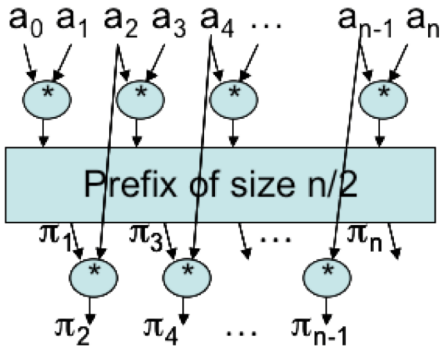
Application to parallel prefix

- **Prefix problem** :
  - input : $a_0, a_1, \ldots, a_n$
  - output : $\pi_1, \ldots, \pi_n$ with

$$\pi_i = \prod_{k=0}^{i} a_k$$

- **Oblivious parallel** algorithm : recursive to minimize the depth D

[Ladner-Fisher-81]



Depth D = 2. *log* n
but performs W = **2.n ops**

Oblivious

but

- Non oblivious tight lower bound on **p identical processors**:

[Nicolau&al. 1996]

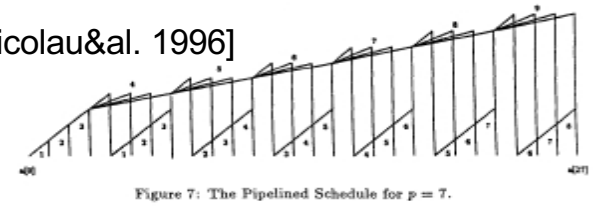


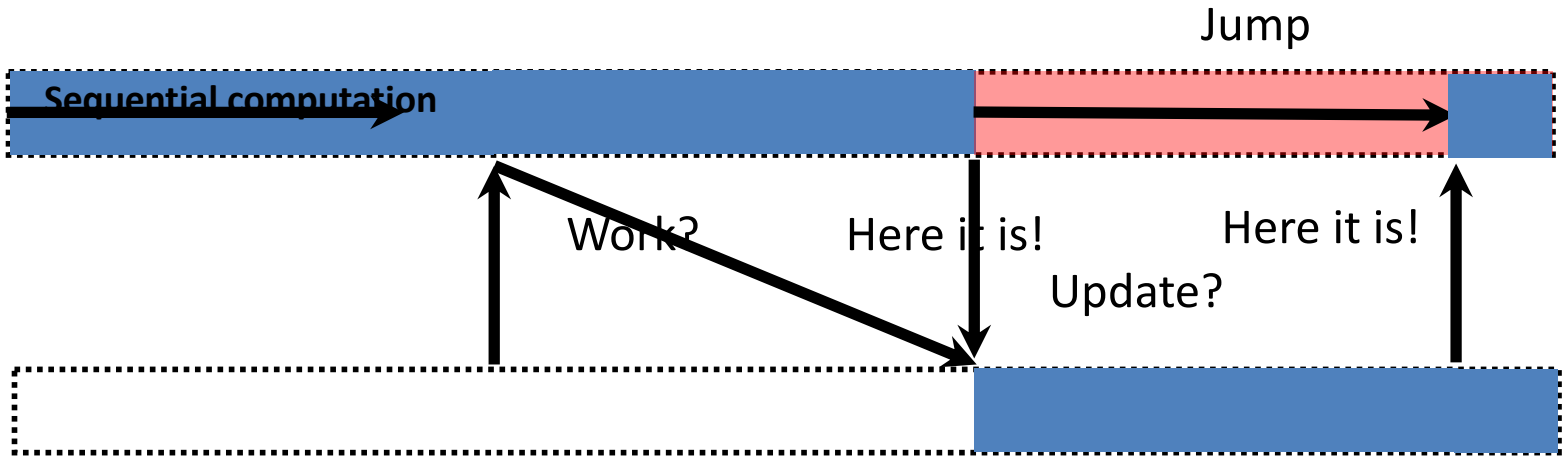
Figure 7: The Pipelined Schedule for p = 7.

Optimal time T_p = 2n / (p+1)
but performs **2.n.p/(p+1) ops**

non optimal

Interactive parallel algorithm oblivious design

- Each resource performs a specialized sequential algorithm
  - other resources act as co-processors
  - at any time, sequential computation is in progress



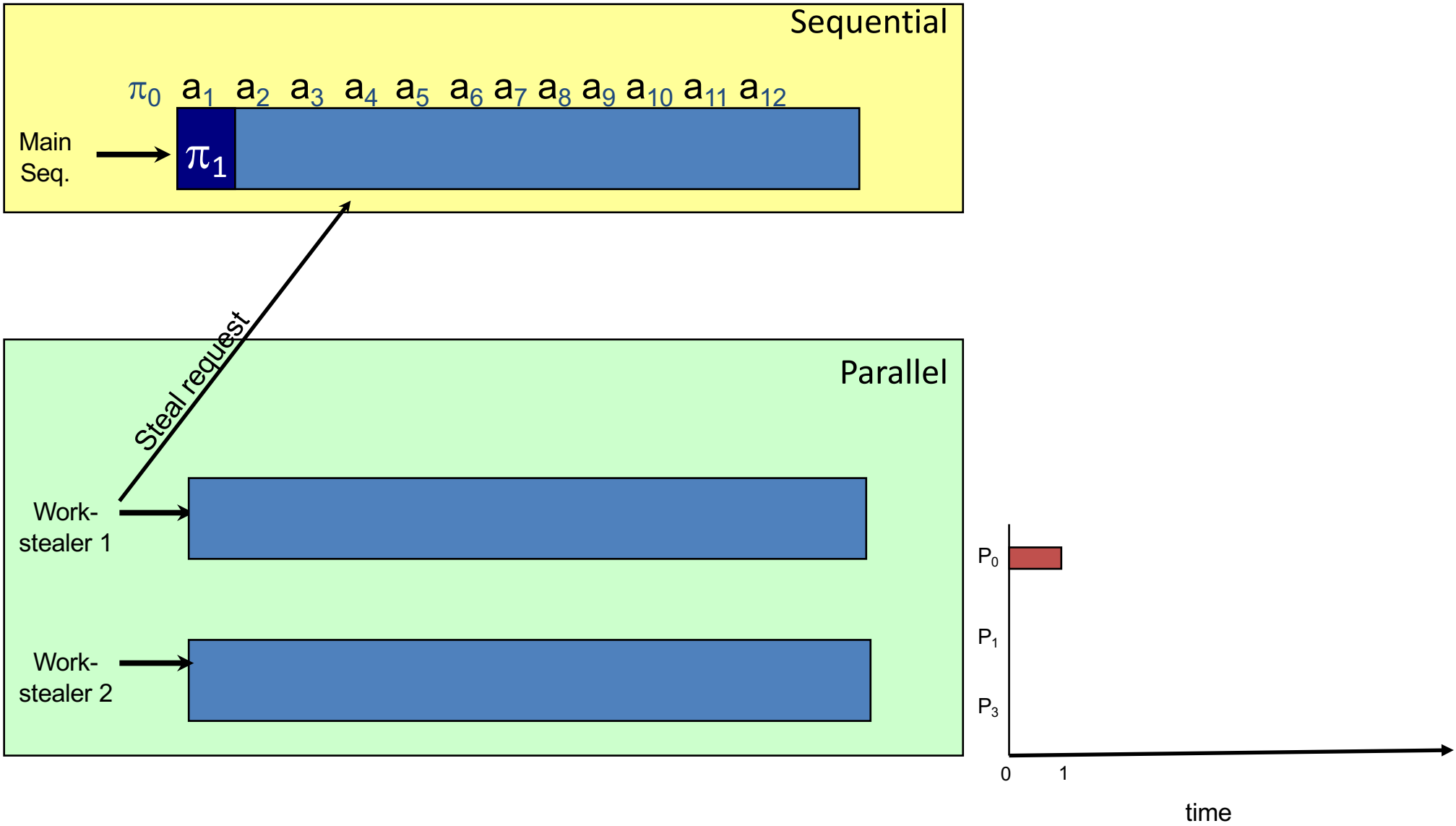
- When a stealer appears, it extracts some work (steal).

- When the work is completed, partial results are merged (eventually, preemption of the stealer).

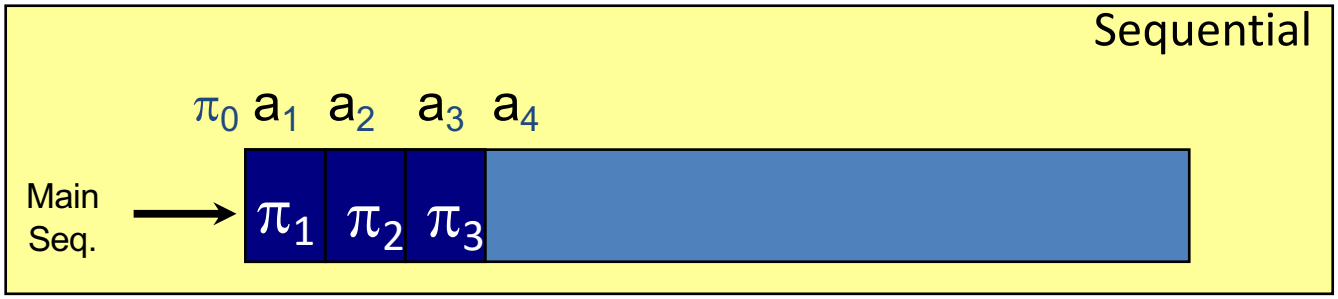
Provable performances in various contexts

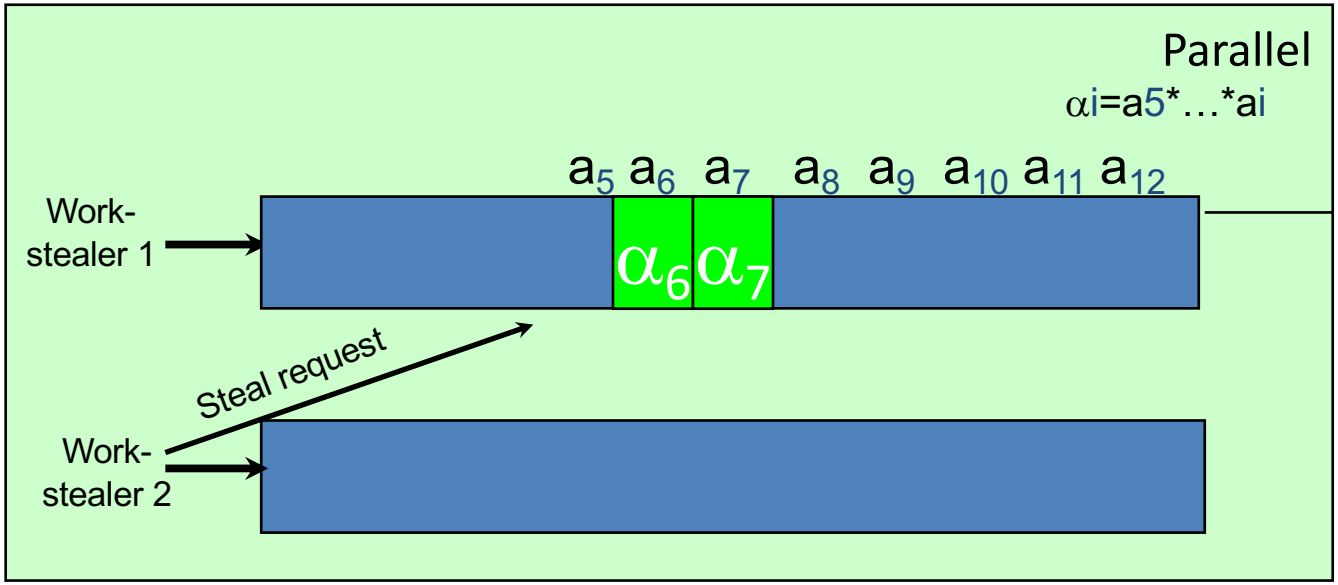
- Computations with iterators: C++ STL algorithms
  - [Traoré, Roch, Maillard, Gautier, Bernard EUROPAR 08]
- Block Cipher Chaining modes (eg AES-CTR)
  - [Danjean,Gillard, Guelton, Roche, Roch PASCO 07]
- Anytime computation
  - [Soares, Raffin, Menier, Roch EGPV 07]
- Deterministic random number generator
  - [Mor, Roch, Maillard EUROPAR 14]

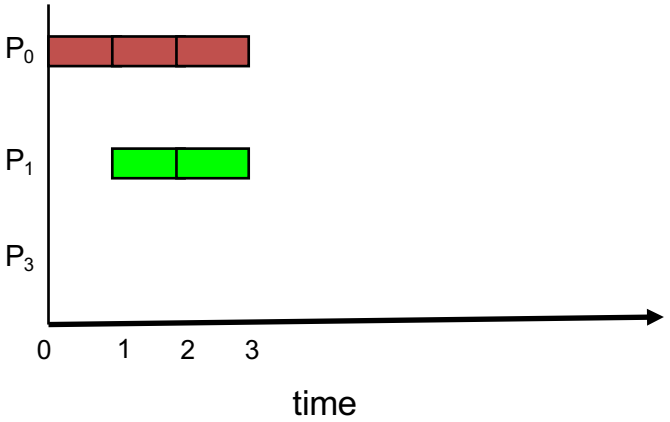
P-Oblivious Prefix on 3 proc.



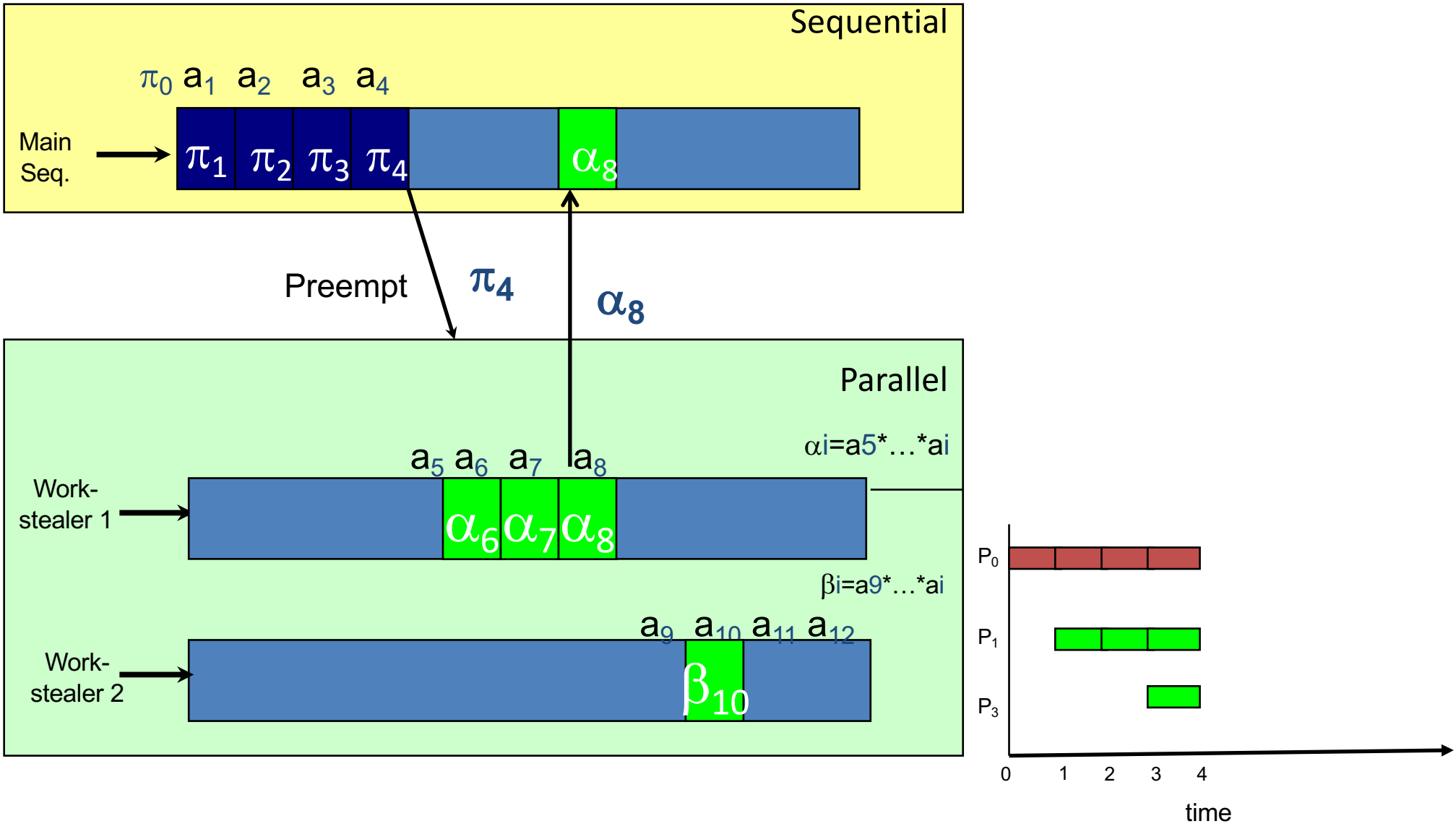
P-Oblivious Prefix on 3 proc.



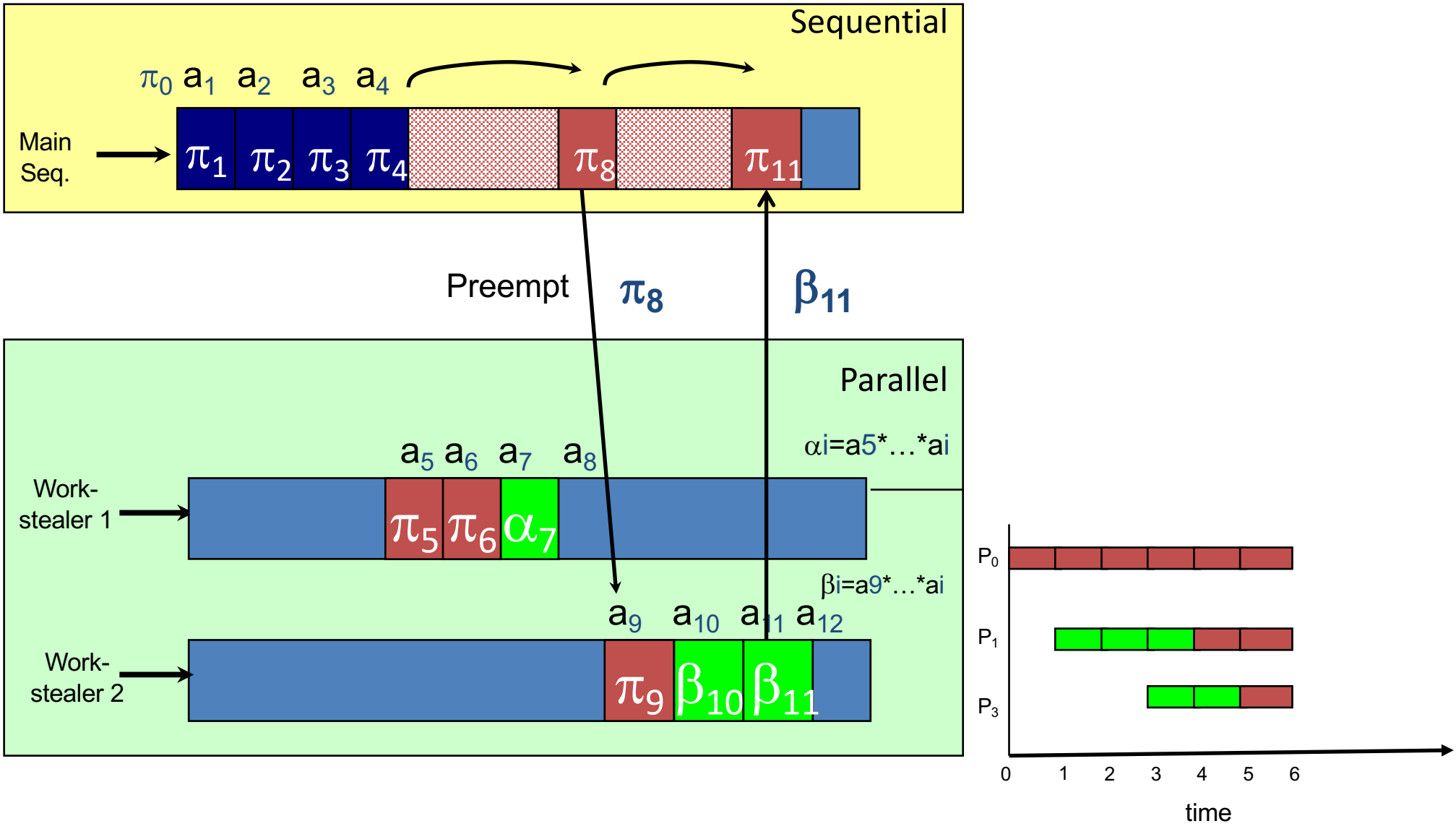




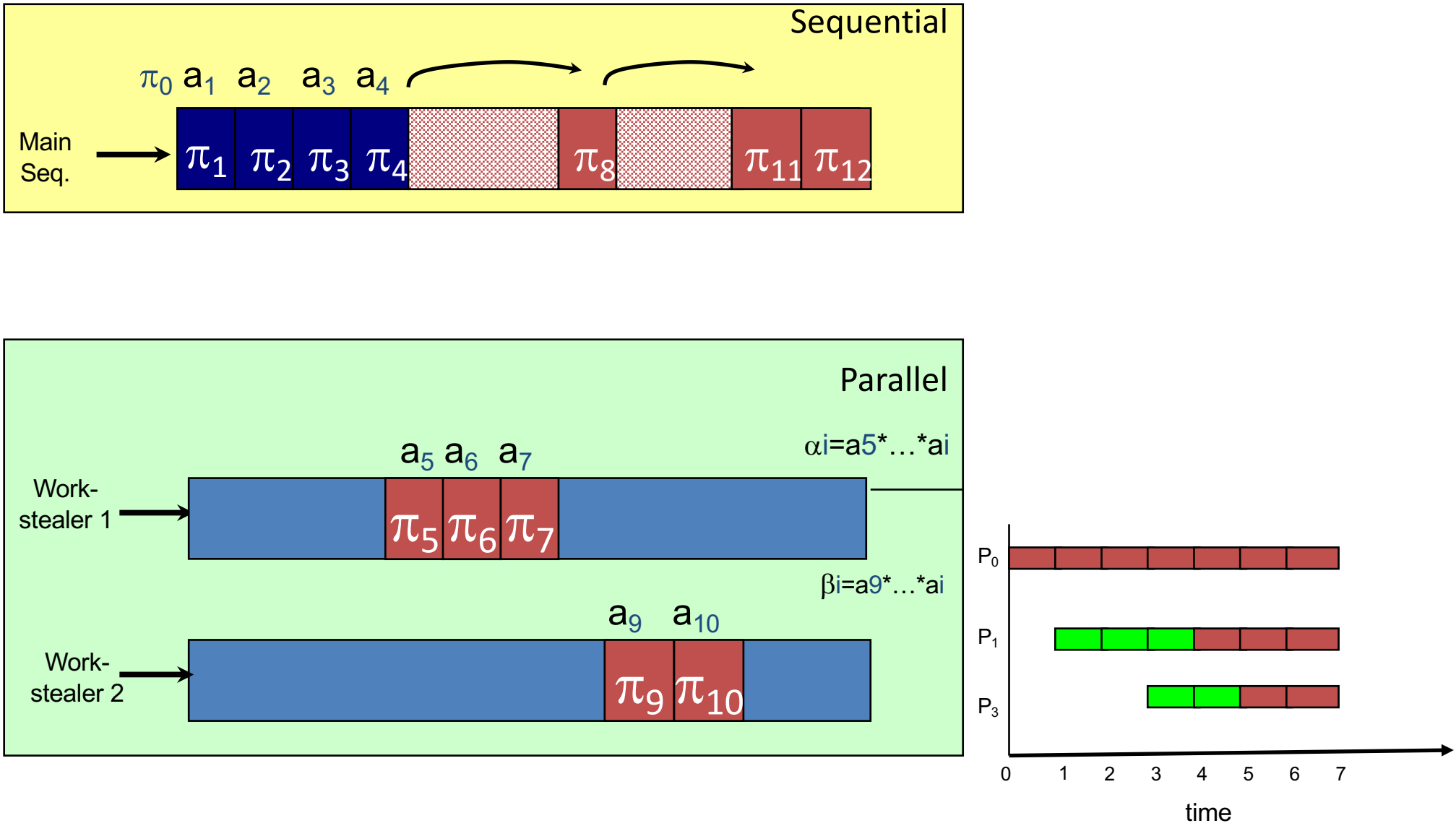
P-Oblivious Prefix on 3 proc.



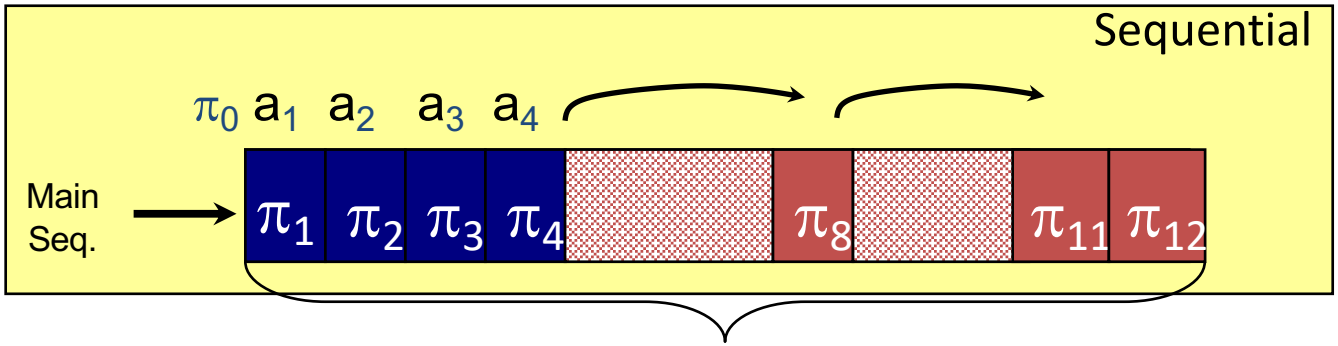
P-Oblivious Prefix on 3 proc.



P-Oblivious Prefix on 3 proc.

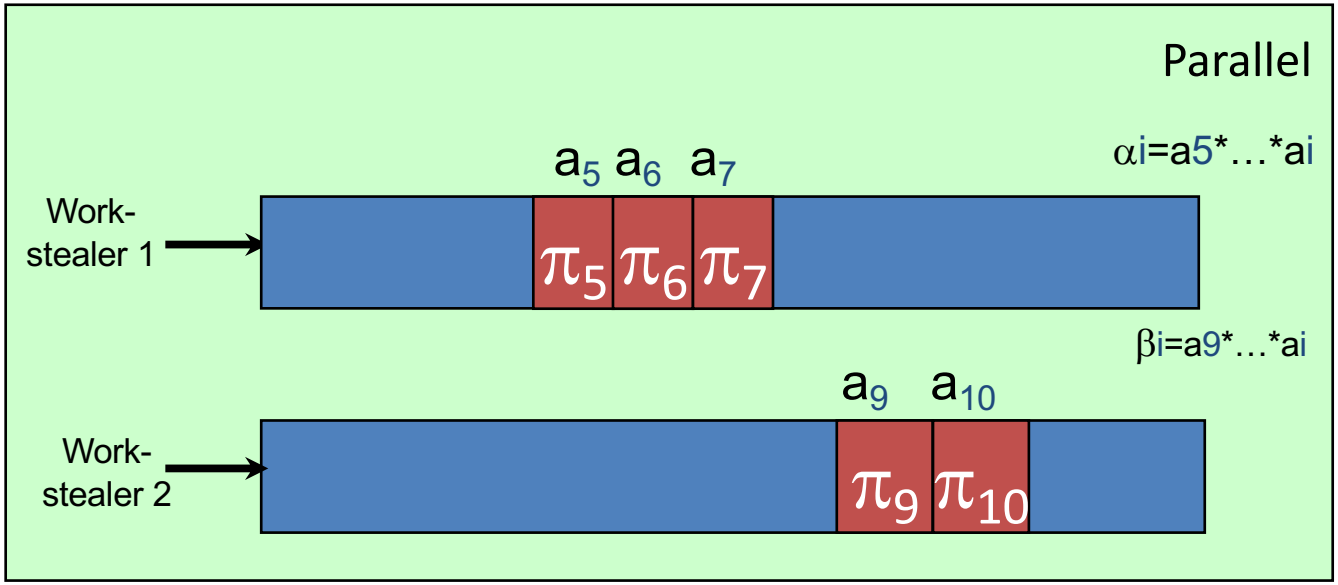


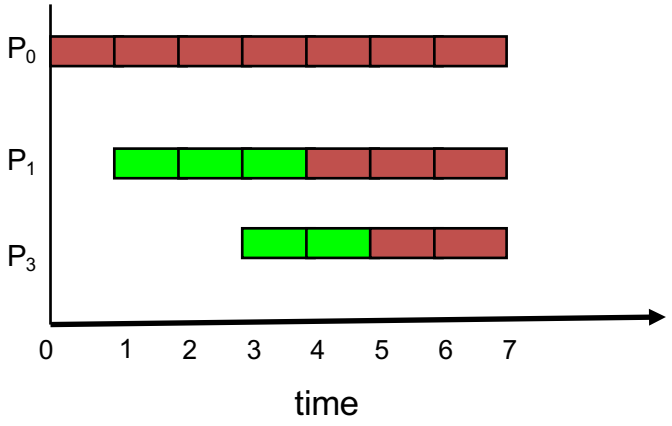
P-Oblivious Prefix on 3 proc.



Implicit critical path on the sequential process

$T_p = 7$ $T_p^* = 6$





Provable performances

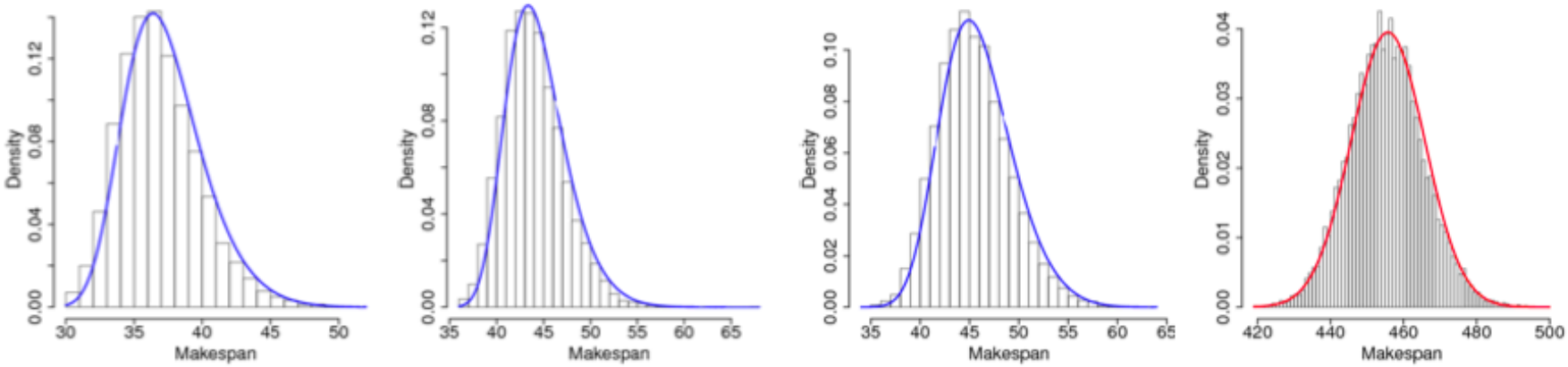
- Model parallel computation by its trace(s) (history)
  - Each computation is modelled a tree, which nodes are sequential blocks
  - #steals request = O(p. Depth) w.h.p.
  - Non determinstic: family of all possible trees

- #Successfull steals, with « steal the oldest » strategy:
  - Let M = maximum number of forks on a parallel
  - Determinstic strategy: #succ_steals ≤ M.(p-1)
  - Random steal strategy:
    $$E[\#succ\_steals] \leq (M-1)p.\ \ln\ln p + p.\ \ln p + O(P)$$
  - The finer the parallel grain => the fewer the tasks
    => the larger the effective granularity !!!

- Randomized Depth-First work-stealing : DAG with arity 2
- #average steals_requests per process **= $O(T_\infty)$** [Arora&al 2001]
- Tighter analysis based (Distributed List Scheduling model) **< 5.5 T_∞** + O(1)

[Tchiboukdjia, Trystram, Roch, Beranrd 2009, ISAAC 2010, A. Oper. Res. 2012]

- independent unit tasks : # average steals_requests **< 3.65 $\log_2 W$** + 3.64
- with initial random allocation **< 1.83 $\log_2 W$** + 3.64
- with cooperative work stealing **< 2.88 $\log_2 W$** + 3.4

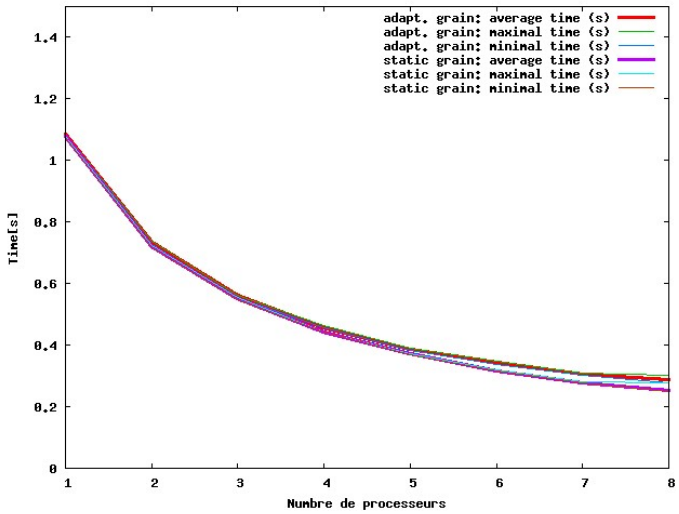


Provable perfomance of the Processor-oblivious parallel prefix

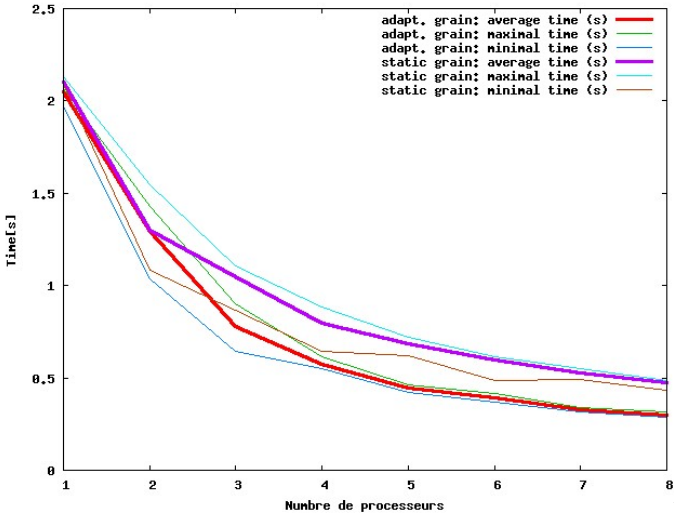
Execution time $\leq \frac{2n}{(p+1).\Pi_{ave}} + O\left(\frac{\log^2 n}{\Pi_{ave}}\right)$

Lower bound

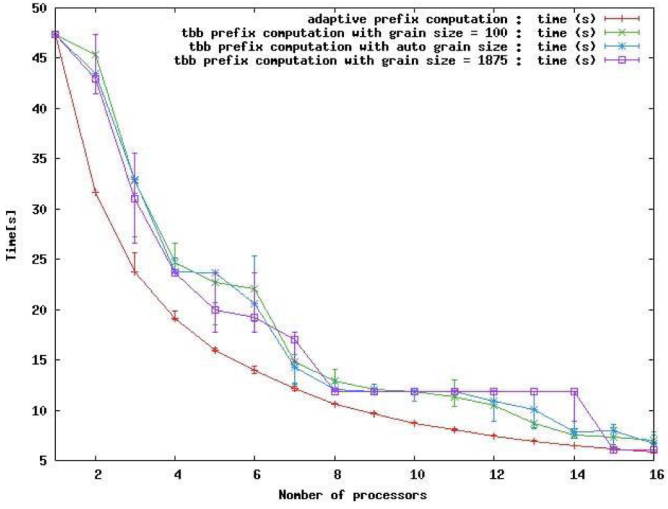
Practical performance [* = double, finest "grain" = 2048 double] [TRMGB, Europar'08]



Single user



Processors with small variations



Comparison with TBB

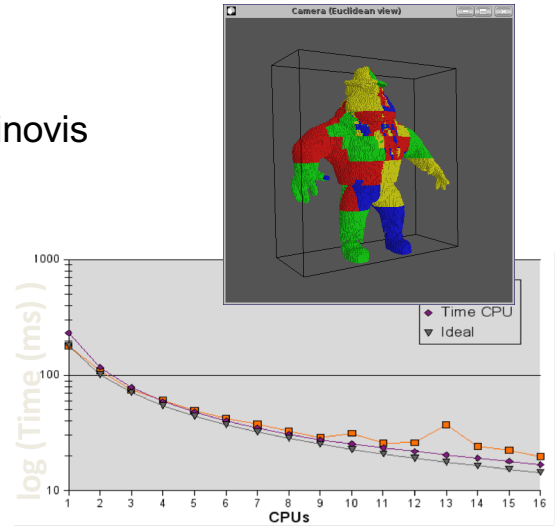
+ Cache-friendly (from the sequential algorithm)

Programming model

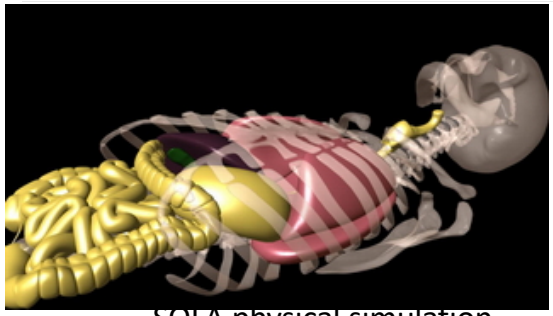
- Athapascan / Kaapi
  - Macro-data flow
  - Tasks with input-output dependencies
    - Dynamic task graph
    - Lazy computation of the DAG of tasks

Hybrid computing CPUs + GPUs

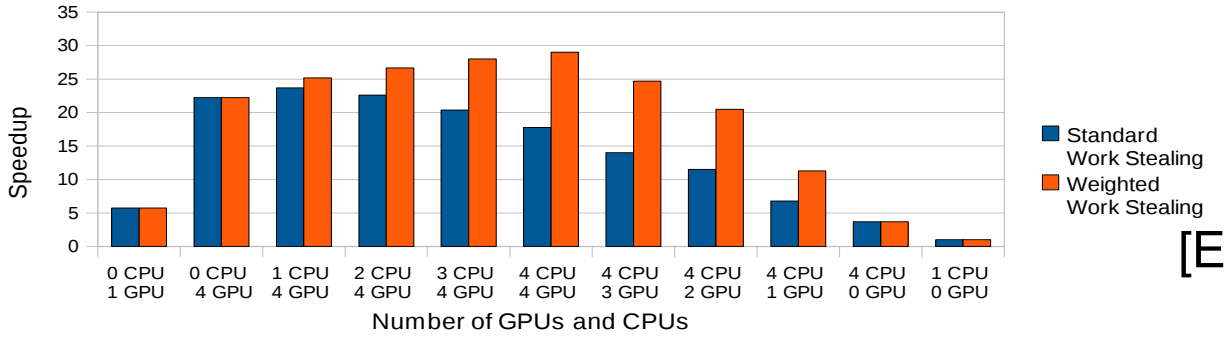
– parallel 3-D reconstruction [Grimage 2007, Kinovis 2015]
adaptive precision wrt frame rate



– Each task has 2 implementations: CPU and GPU
  - Work-stealing + data-partitioning KAAPI embedded in SOFA



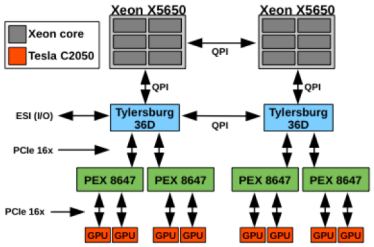
SOFA physical simulation
[Evasion / Imagine]

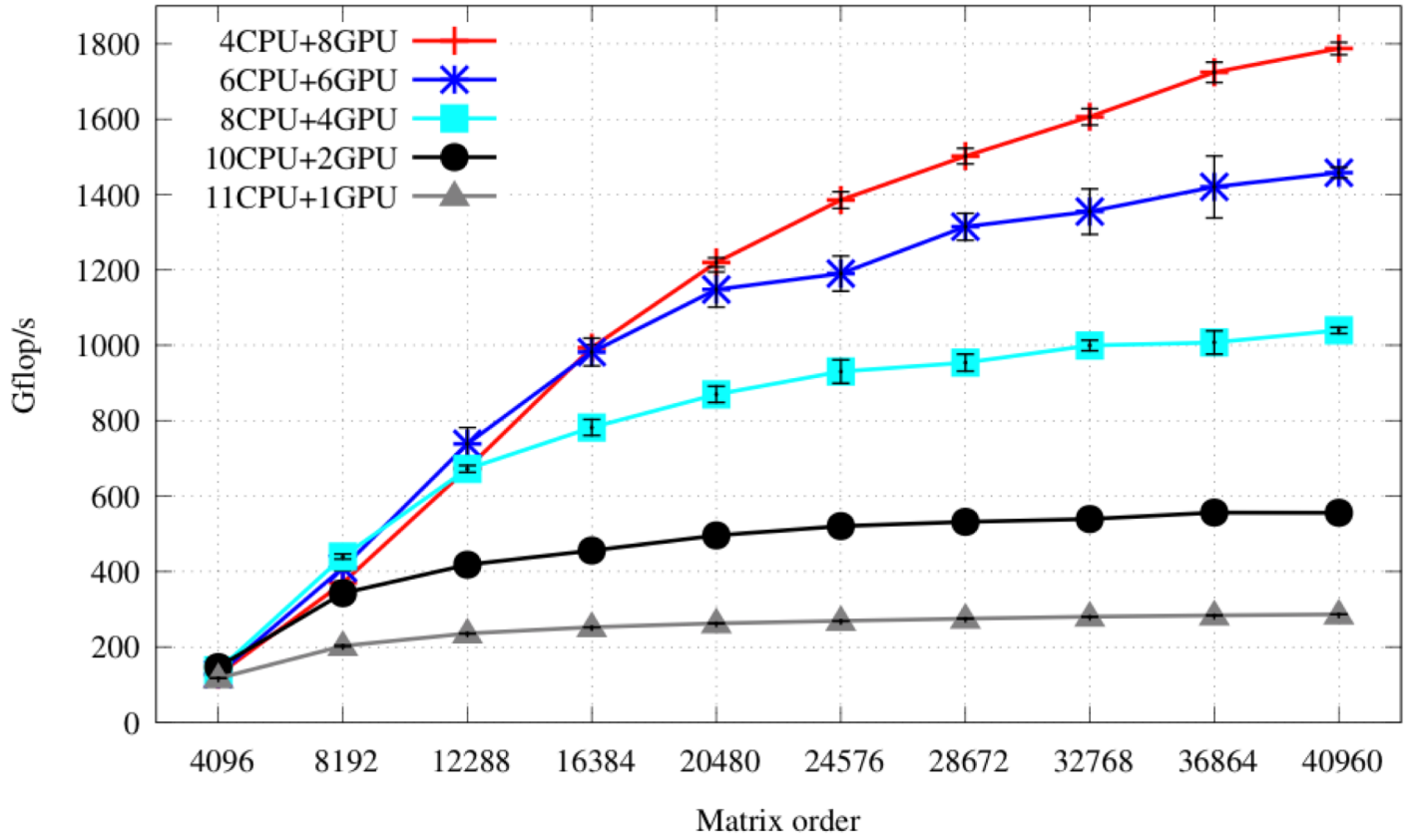


[E Hermann 2010]

DPOTRF multi-CPUs + Multi-GPUs

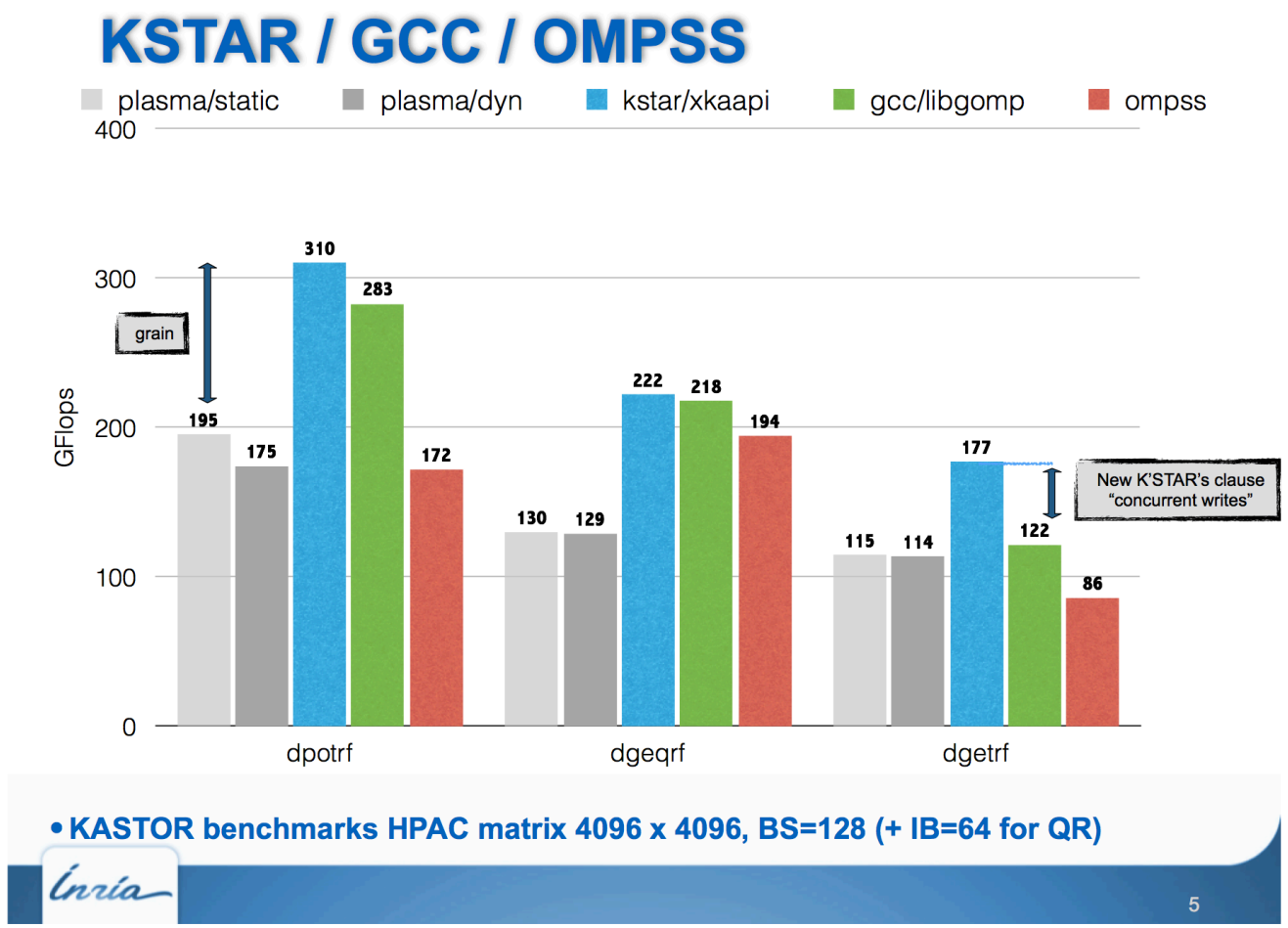
2 hexacores INTEL X5650 (2.66Ghz)
8 GPUs Tesla C2050 (fermi GPU, 1.15Ghz, 3GBytes)





Gautier &al.
[IPDPS 2013]

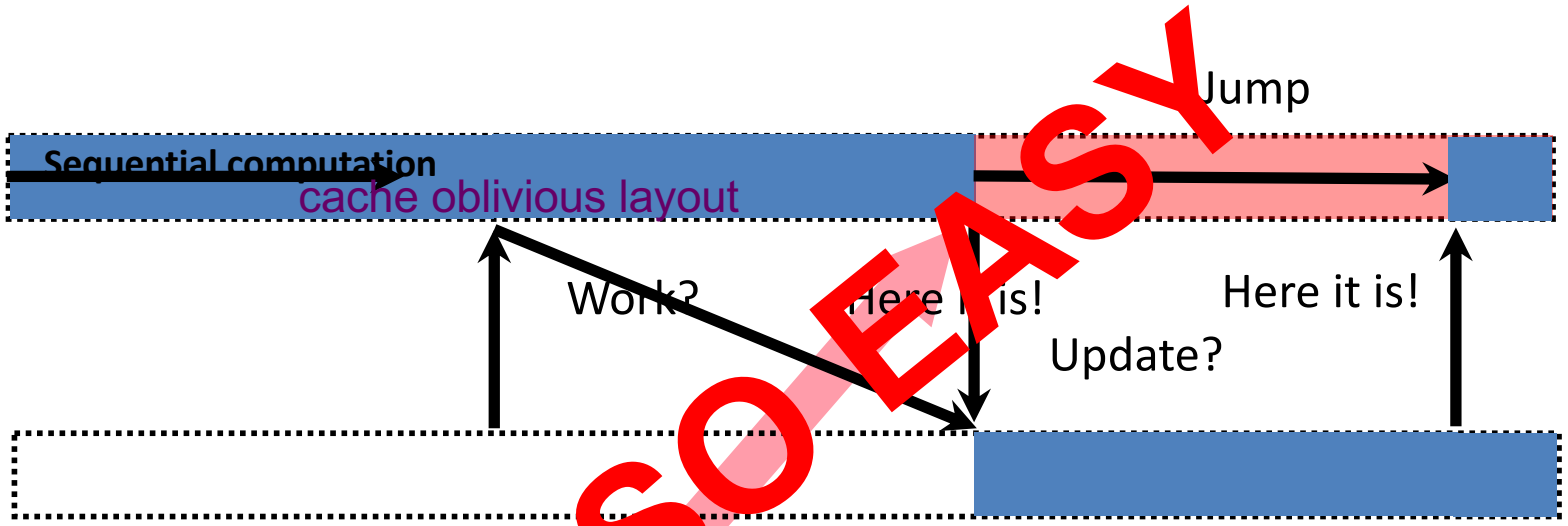
Integration in OpenMP-4



[Broquedis, Gautier &al. IWOMP 2014]

HPAC Machine
Intel Xeon CPU E5-4620 0 @ 2.20GHz
- 4 NUMA nodes
- 32 coeurs

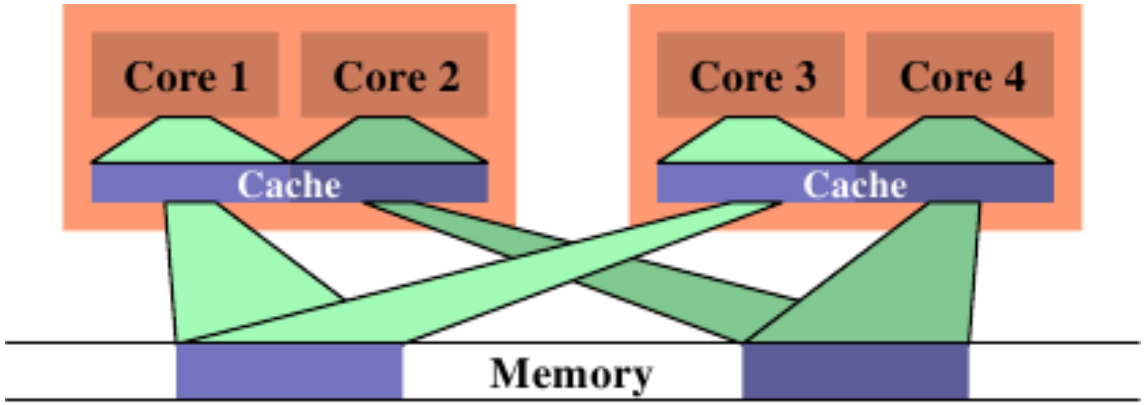
What about locality?



NOT SO EASY

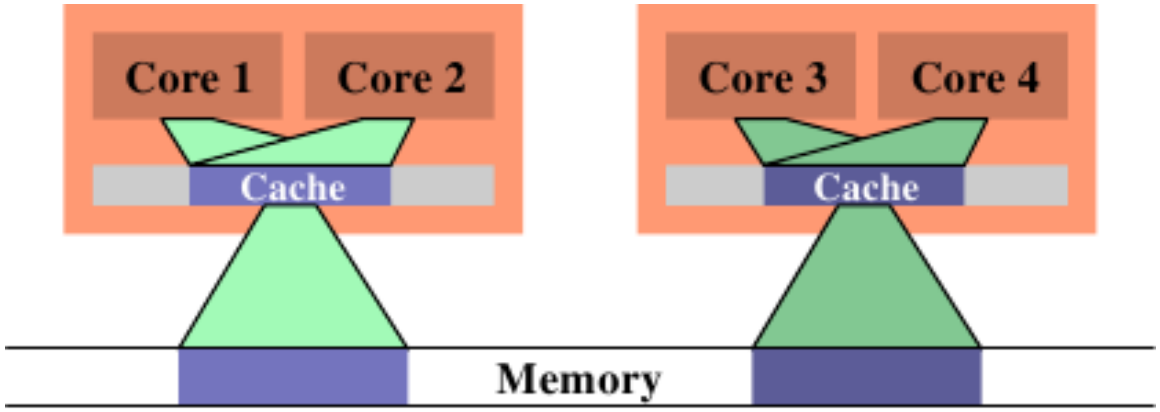
Few steals so few cache misses/synchro

Interaction cache / scheduling



Can be worse than
each core has a cache of half size

Cache sharing



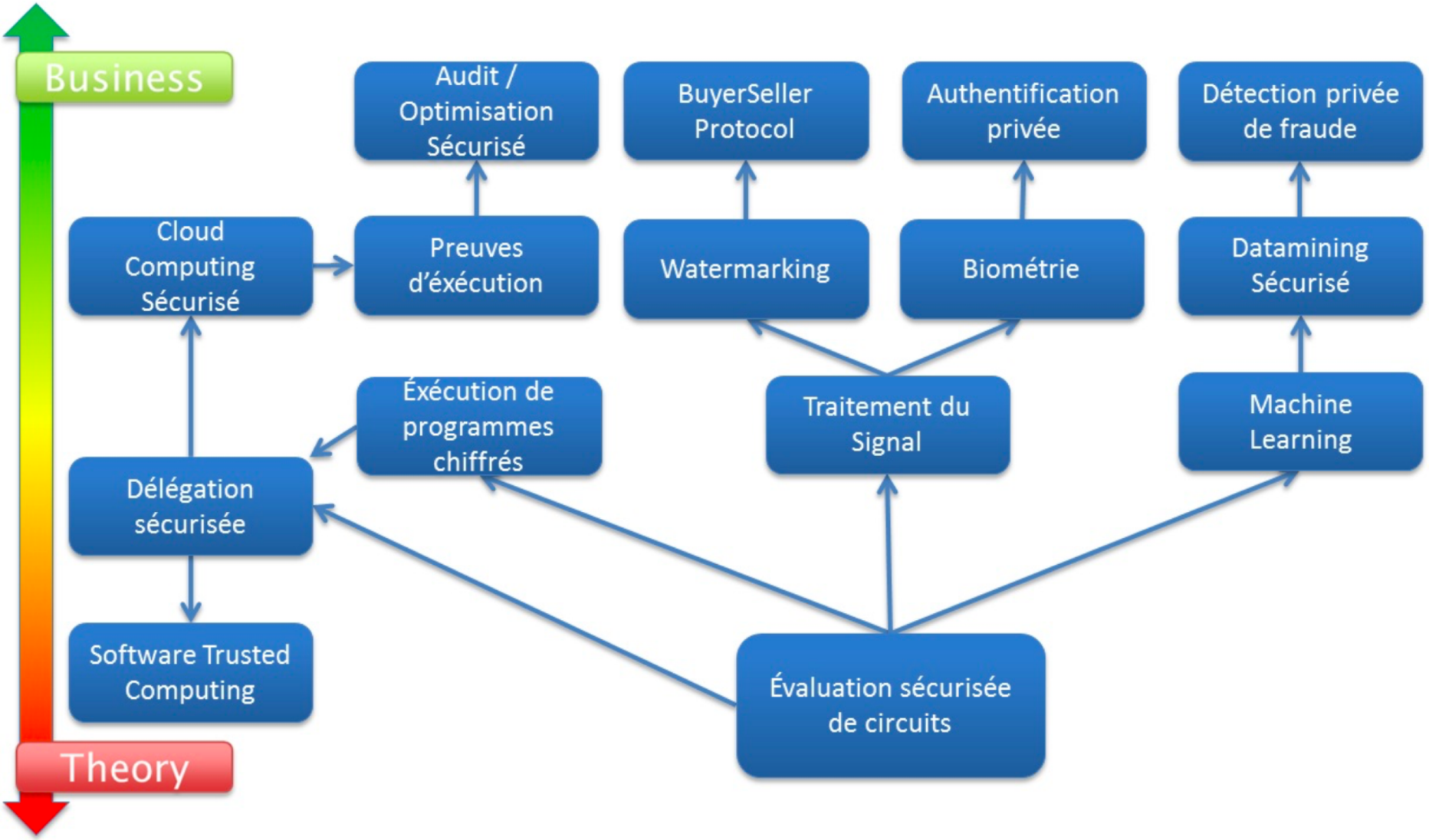
Can be as good as
each core has a cache of full size

Plan

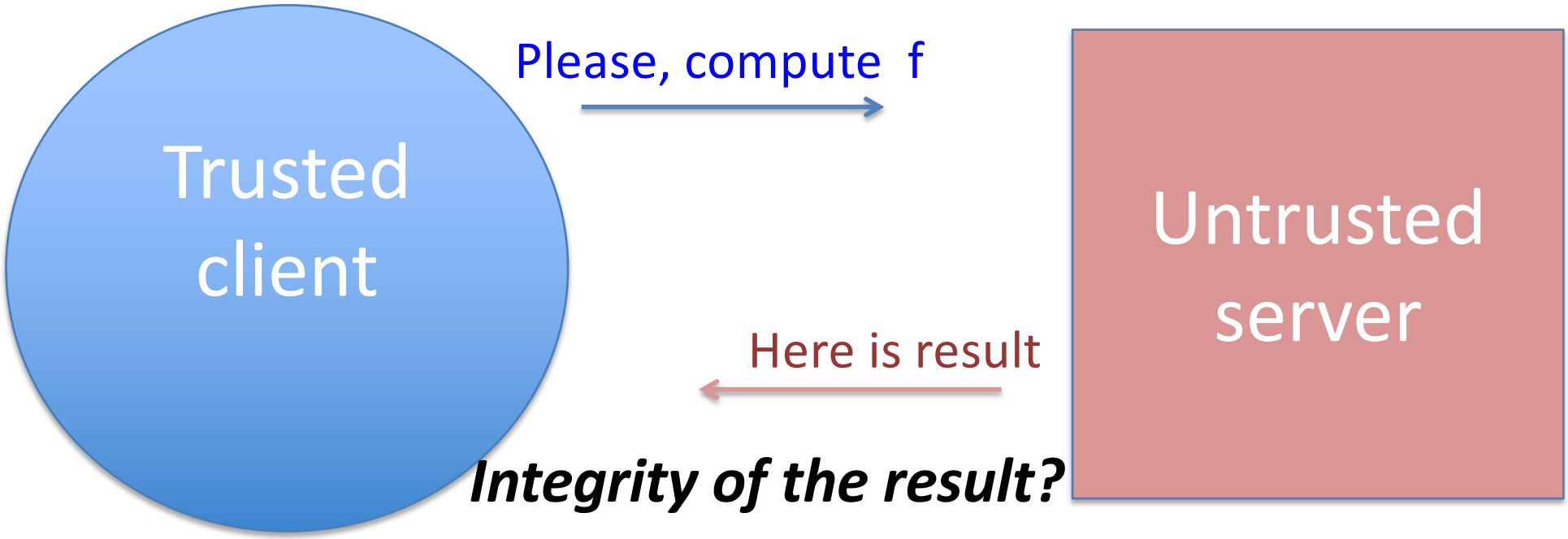
Interactive computations and outsourcing

1. For oblivious parallel provable performances
   - Work-stealing
   - Hybrid computing

2. For provable security
   - Result correctness (integrity)


Business
Theory
Audit / Optimisation Sécurisé
BuyerSeller Protocol
Authentification privée
Détection privée de fraude
Cloud Computing Sécurisé
Preuves d'éxécution
Watermarking
Biométrie
Datamining Sécurisé
Éxécution de programmes chiffrés
Traitement du Signal
Machine Learning
Délégation sécurisée
Software Trusted Computing
Évaluation sécurisée de circuits

Delegating computation



- Contexts
  - Co-processor (overclocked…)
  - Supercomputer (soft errors)
  - Cloud computing
  - Volunteer computing

Motivating example

- Peggy has developed a nice application that efficiently solves Traveling Salesman Problem
- Victor sends Peggy the location (map) of his clients and pays her for the shortest Hamiltonian circuit
- How does Victor check he gets the shortest?

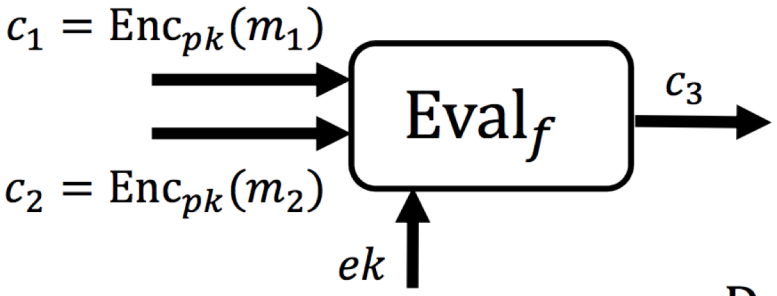
Outsourcing and privacy

- Previous example: Victor provides the location of his clients => how to keep this secret

- Other example: digital doctor

Outsourcing and privacy

- Outsourcing computation with secret input
  - Computation is performed on encrypted data
  - Based on asymnetric encryption (eg thanks to fully **homomorphic encryption**) [Gentry 2009]



$$\mathrm{Dec}_{sk}\left(\mathrm{Eval}_f(ek, c_1, c_2)\right) = f(m_1, m_2)$$

Homomorphic encryption: El Gamal e-vote

- **Remind El Gamal** (in cyclic group G with g a generator):
  - Bob has private key b and public key $B=g^b$
  - Alice: $C=E(M) = (c_1=g^r , c_2=M . B^r)$ Bob= $D(C)=c_1^{-b} . c_2 = M$

- **El Gamal enables homomorphic addition** *(note Alice encrypts g^M instead of M)*:
  - $C=E(M) = (g^r , g^M . B^r)$ (encode g^M instead of M)
  - $C'=E(M')= (g^{r'} , g^{M'} . B^{r'})$
  - Multiplication of ciphetxt in G matches addition of plaintext *(e.g.integers plaintext)*
    $C.C' = (g^r . g^{r'}, g^M . B^r . g^{M'} . B^{r'}) = (g^{r+r'}, g^{M+M'} . B^{r+r'})$
    $= E(M+M')$
  - Enables anyone to compute as many additions of ciphertexts as desired

- **Application: electronic vote by homomorhic addition**
  - Each voter (Alice) sends her encrypted vote v (0 or 1) to the voting machine (Bob) :
    - $C(0)=(g^r , B^r)$ $C(1)=(g^{r'} , g.B^{r'})$: each Alice checks her encrypted vote is correctly stored
  - Each one can compute the encrypted score of the vote : $\Pi_{C\ voter}(C) = (g^{\Sigma r} , g^{\Sigma v}.B^{\Sigma r})$
  - The voting machine knows secret b : it decrypts the score $\Sigma_{voters}v$ and publishes it !
    - Why the voting machine does not need to compute discrete log ?

Fully homomorphic encryption

- How to perform AND and XOR on ciphered booleans ?
- [Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan]
  - Secret p : a large odd integer (eg thousands of digits)
  - For x in {0,1} : E(x) = pq +2.r + x
    with random q ~ million of digits and r ~ twenty digits (the ***noise***)
  - Knowing p: (E (x) mod p) mod 2 = (2.r+x) mod 2 = x
  - Without knowing p: E(x) seems to give no information

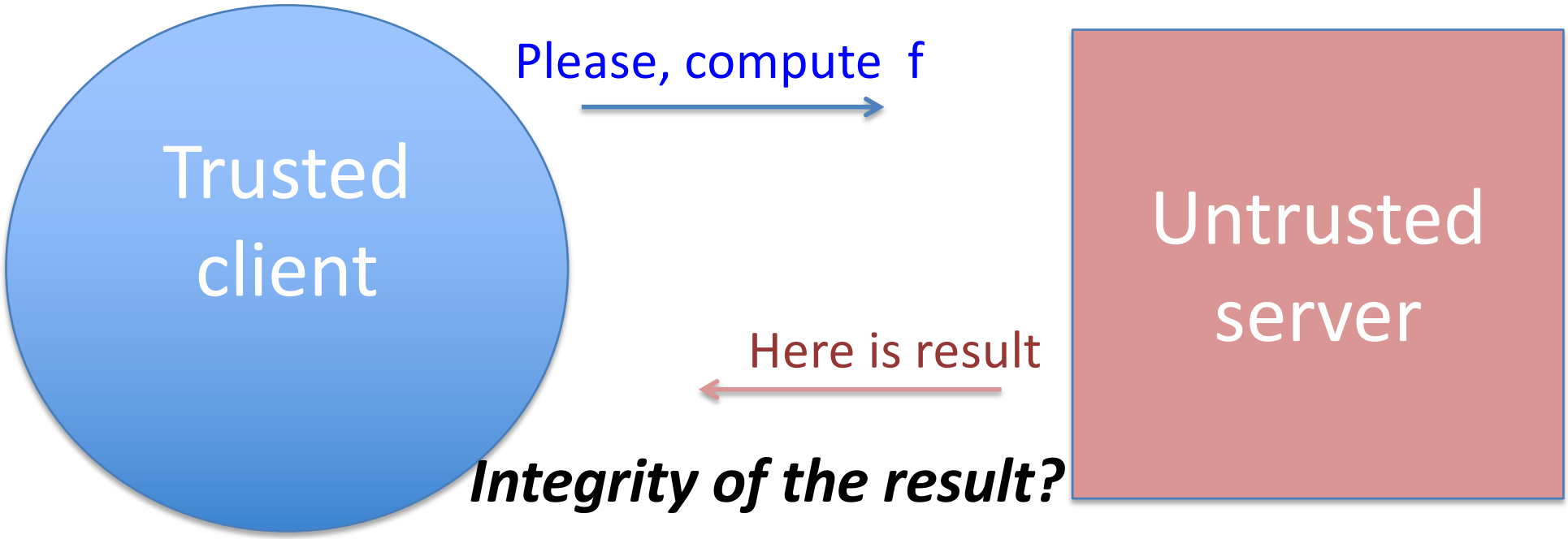
- Fully homomorphic with @x and y booleans:
  - E(x)+E(x')=p.(q+q') + 2(r+r ')+ x+x' => mod p mod 2 = x XOR x'
  - E(x).E(x')=p (pqq'+q(2r'+x')+q'(2r+x)) + 2(2rr'+rx'+r'x) + x.x'
    => mod p mod 2 = x AND x'
- But with operations, the noise increases may become larger than p
- Key Gentry's idea: bootstrapping
  - Eliminate the noise by outsourcing remander by p on the cipher domain
    - by homomorphic computation with AND and XOR (so without revealing p, only its ciphering !)
  - Care : the number of AND and XOR keeps the noise bounded
    - chooice of p and the q's large enough!)

Outsourcing and privacy

- Homomorphic scheme enables to outsource encryption with secret key (or signature)

- Homomorphic encryption enables publicly Verifiable computation [Fiore, Gennaro 2012, …]
  - Server computes on private data and produces a verifiable digest of the computation
  - Verification of the computation
    - Different from a result certification

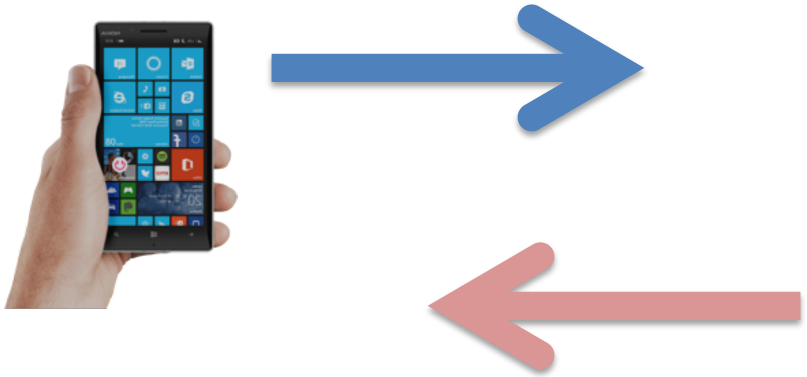
Delegating computation

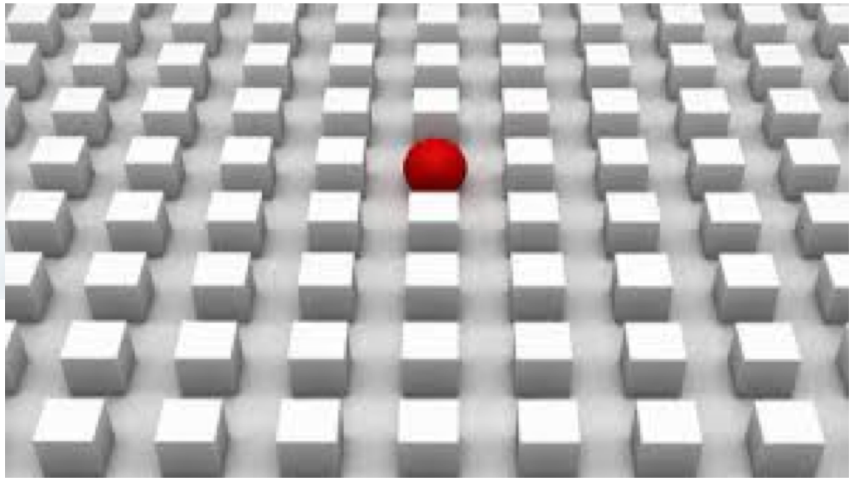


- Contexts
  - Co-processor (overclocked…)
  - Supercomputer (soft errors)
  - Cloud computing
  - Volunteer computing

Attack models

- No attack [current HPC and grid computing platform]
  - Failure (MTBF)

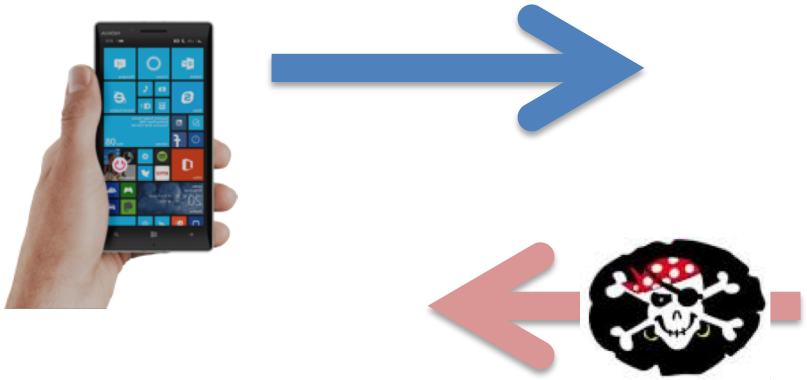


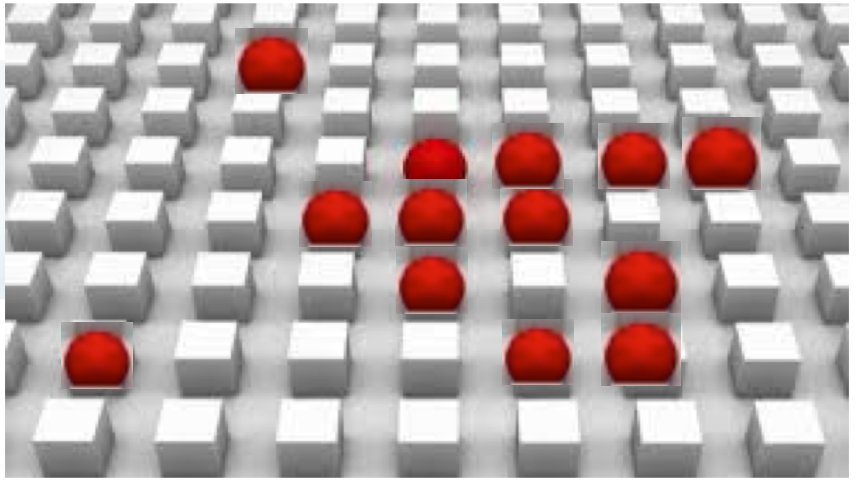


- **Attack on few isolated resources**
  - Soft errors - corruption of part of the computation

Attack models

- No attack [current HPC and grid computing platform]
  - Failure (MTBF)



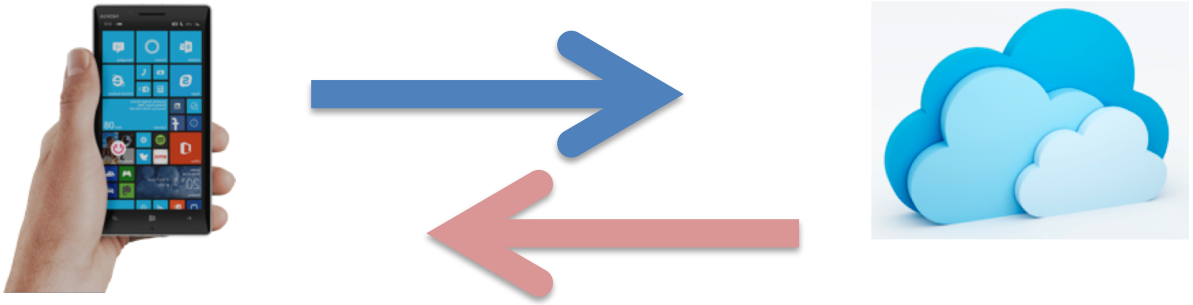


- **Attack on few isolated resources**
  - Soft errors - corruption of part of the computation
- **Massive attacks**

Countermeasures against such attacks (detect/correct)

Verifiable [outsourced] computation

- **Trusted but slow Client** (Verifier, *Victor*)
  sends a function F with input x to the server

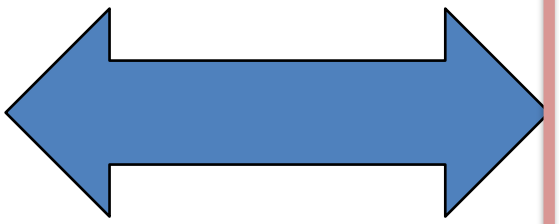


- **Fast but untrusted Server** (Prover, *Peggy*)
  returns y = F(x) and a proof Π that y is correct.

Computing Π should take almost same time than F

Verifying Π should take less time than computing F.

The power of interaction



Verifier
(Victor)

Prover
(Peggy)

Example : Graph isomorphism

- Input : two graphs G_1 and G_2
- Output : Yes iff G_1 and G_2 are isomorphic

- Graph isomorphism is in NP
  - not known to be neither in P nor NP-complete;
  - not known to be in co-NP.

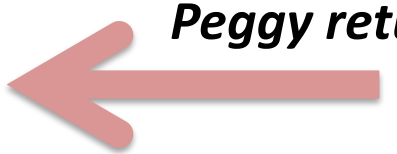
- Proof of isomorphism : Π = permutation that maps E_1 to E_2
  - Then a verifier can check the proof in almost linear time.

- *But no polynomial proof yet known for graph non-isomorphism.*

Interactive graph [non]-isomorphism

- Victor
  - Toss b := rand{1,2}
  - H := random_pemutation (E_b)
  - Asks Peggy: to which H is isomorphic to : E_1 or E_2 ?

Peggy returns y and Π



  - Victor checks Π and if OK
    - If y ≠b : Victor has a proof that E_1 isomorphic to E_2
    - Else y = b : Victor states that E_1 is not isomorphic to E_2 with error probability ½
  - Error probability ≤ 2-k after k rounds

Efficient verifiable computing by spot checking

- Check polynomial equality by random evaluation

  [Schwartz-Zippel]

  - Choose $r_1, \ldots, r_n$ at random in a subset S of a field
  - If $Q(r_1, \ldots, r_n)=0$ then $Q == 0$ with error probability $\leq \deg(Q) / \#S$

- Example: *Verifying matrix multiplication* (**Friedval's algorithm**)
  - To check C = A.B, choose a random vector r
    and verify C . r = A . (B . r)

    Cost : linear in size(A) + size(B) + size(C)

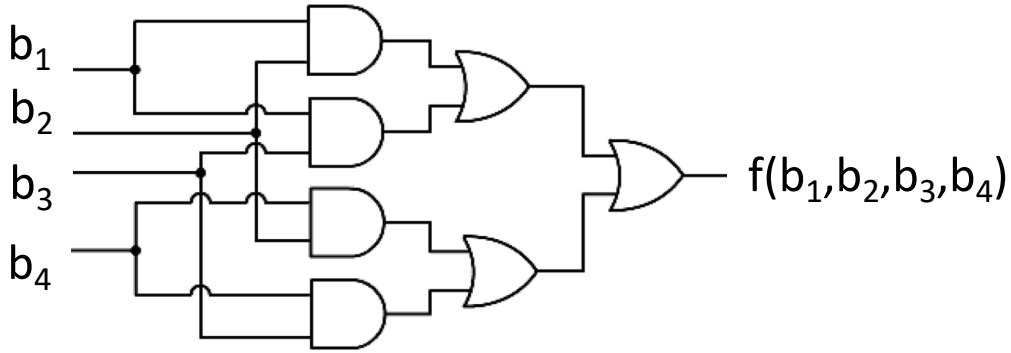
Interactive linear algebra

- Most dense linear algebra reduces to Matrix multiplication
  - Locally compute the (recursive) scheme in $O(n^2)$ while outsourcing all Matrix Multiplications
  - [Algorithm-Based Secure and Fault Tolerant Outsourcing of Matrix Computations, A Kumar, JL Roch, HAL 2013]

- Alternatively provide efficient certificates for sparse linear algebra
  - [Interactive certificates for linear algebra, JD Dumas, E Kaltofen, ISSAC 2014]

Verifying general circuits

- Inputs : $b_1 \ldots b_n$ Outputs : $y_1 \ldots y_m$
- How to verify $y_1 \ldots y_m = f(b_1 \ldots b_n)$



The power of interaction

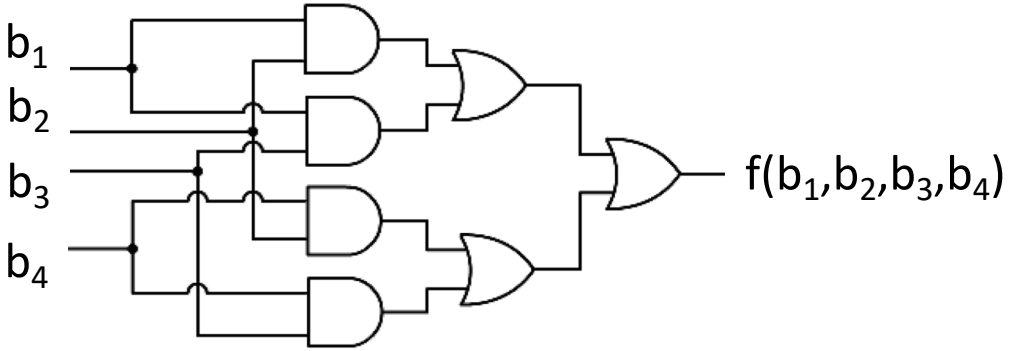
- Any problem in PSPACE has a polynomial verifier
  - TQBF (quantified Boolean formula problem)

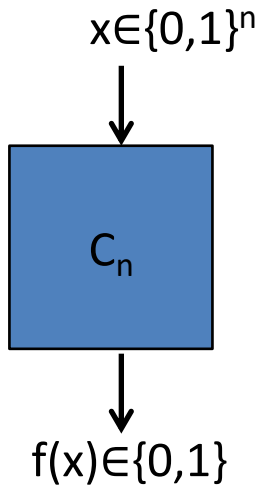
- A polynomial interactive scheme for #SAT

[Lund, Fortnow, Karloff, Nisan 1992]

A key tool: the **sum-check** protocol

- **Input** : a (boolean) circuit C_n of depth δ that implements a function f with n bits in input :





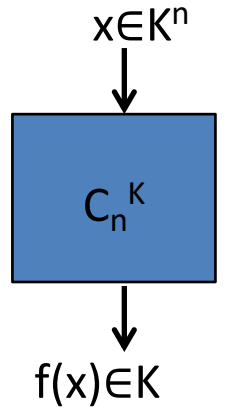
- **Output** : $S_n = \sum_{b_1=0,1} \dots \sum_{b_n=0,1} f(b_1, \dots, b_n)$

- Let $d=2^{\delta}$: #usefull gates ≤ d.
  Theorem: The verifier *interactively* computes S_n in polynomial time $(n+d)^{O(1)}$. (if δ=O(log n), polynomial in n)

- Application: number of elements that verify a predicate (#SAT)

Key 1: Arithmetization

- Transform the boolean circuit C_n in an arithmetic circuit C_n^2 in any field K (eg mod *p*) :
  - x *and* y = x $._K$ y *not*(x) = 1 – x
  - x *or* y = not (not(x) *and* not(y)) = 1 $-_K$ (1-$_K$x). $_K$(1-$_K$y)
- Transform the circuit C_n^2 in a circuit C_n^K with input in a (large) field K.
  - Gates are + and x in K
  - When inputs are 0 or 1, the output is the same than C_n



- Now, the circuit can be seen as a polynomial in n variables (the input) with degree d
  - For m=log #K, the circuit can be evaluated in time $(nm)^{O(1)}$, polynomial for any [random] input in K^n.

- **Key 2: induction on the number of sum**
  - Each sub-sum is verified with Schwartz-Zippel

Interactive verification of #3-SAT

- Let: Φ = (c_1 and … and c_m) be a 3-SAT CNF formula
- Arithmetization of Φ gives $g(X_1, \ldots, X_n) = Q(c_1).Q(c_2)\ldots. Q(c_m)$
- Deg(g) ≤ 3m (small)
  Polynomial-size circuit to evaluate g at any $(b_1, \ldots, b_n)$
- To prove #SAT(Φ)=K reduces to a sequence of sum-check

$$\Sigma_{b_1=0,1} \ldots \Sigma_{b_n=0,1}\, g(b_1, \ldots, b_n)$$

  - computation in F_p with p prime > 2^n

[Goldwasser, Kalai, Rothblum 2008][Thaler Crypto 2015]

Outsourcing general circuits

- Circuits C with n inputs and outputs,
  - Work W, depth D
  - Each level is of degree 1 (multilinear extension)
- Computation is valid iff all levels are corrects
  - Verified by a sum-check at each level

- Cost = $(N + D) \log^{O(1)} (N + W)$

- Optimization when the computation resumes to a reduction of independent parallel computations

Illustration on Matrix Multiplication

[Thaler, Crypto 2013, Crypto 2015]

- Let A and B matrices (n,n) in K with $m = \log_2 n$
- A is a (boolean) function $\{0,1\}^m x \{0,1\}^m x \rightarrow K$:

  $A(i_1,\dots i_m, j_1, \dots, j_m) = A(i,j)$

- Let g_A be the polynomial multilinear extension of A
- The g_C verifies

  $g_C(i_1,\dots i_m, j_1, \dots, j_m) = \sum_{k=0..n} g_A(i_1,\dots i_m, k_1, \dots, k_m).g_B(k_1,\dots k_m, j_1, \dots, j_m)$

- With the sum-check protocol, this sum of *n* elements is verified in O(log *n*)
- Generalizes to parallel compuations with logarithmic depth (NC1)

Practical efficiency ?

- Further improvements [Thaler]
  - Sum of products only
  - Same circuit for any coefficient

Problem Size	Naïve MatMult Time	Additional P time	V Time	Rounds	Protocol Comm
1024 x 1024	2.17 s	0.03 s	0.67 s	11	264 bytes
2048 x 2048	18.23 s	0.13 s	2.89 s	12	288 bytes

  - Yet far from Fiedvald's verification

Conclusion on outsourcing

- Verifying delegated computation
  - Interaction between models provides power
  - Enables the provable use of untrusted platforms
    - Overclocked processors, algorithms with faults, quantum computing, …
  - Fully Homomorphic encryption (yet inefficient)
  - Current research to improve FHE efficiency

- On going research - Applications
  - Cloud computing. (web services)
  - Outsourced fault-tolerant computation
  - Secure remote storage (privacy)
  - Secure control-command for critical infratscture (SCADA)
  - A promising market (eg digital doctor)

Questions ?

- S Arora, B Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009
- C. Gentry, Fully homomorphic encryption using ideal lattices, 2009
- J Thaler. Time-optimal intercative proofs for circuit evaluation. Crypto 2013,
- D Fiore, R Gennaro, Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications, 2013
- B Parno, C Gentry. Pinocchio: nearly practical verifiable computation. Security&Privacy 2013
- M Walfish, AJ BlumbergVerifying computations without reexecution them. Comm. ACM 2/2015
- I Chillotti, N Gama, M Georgieva, M Izabachene. "Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping". 2017.