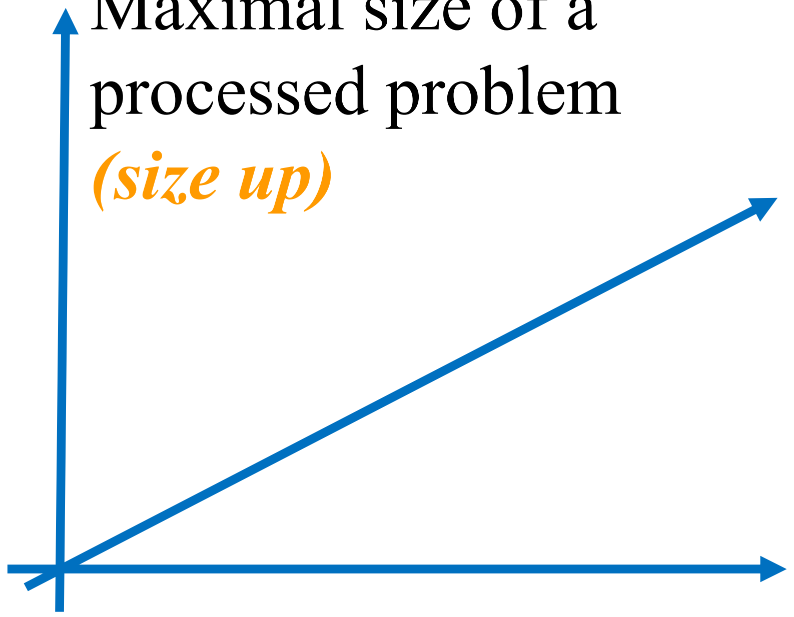
Big Data

Performance, efficiency and scalability metrics

Stéphane Vialle & Gianluca Quercini

What to do with more computer resources?



Maximal size of a processed problem *(size up)*

Nb of independent problems processed per seconde *(flow strength increase)*

Processing speed of a problem *(speedup)*

Passage à l'échelle / Scaling up : size up + speedup + cost control

1 – Preliminary analysis of the execution time

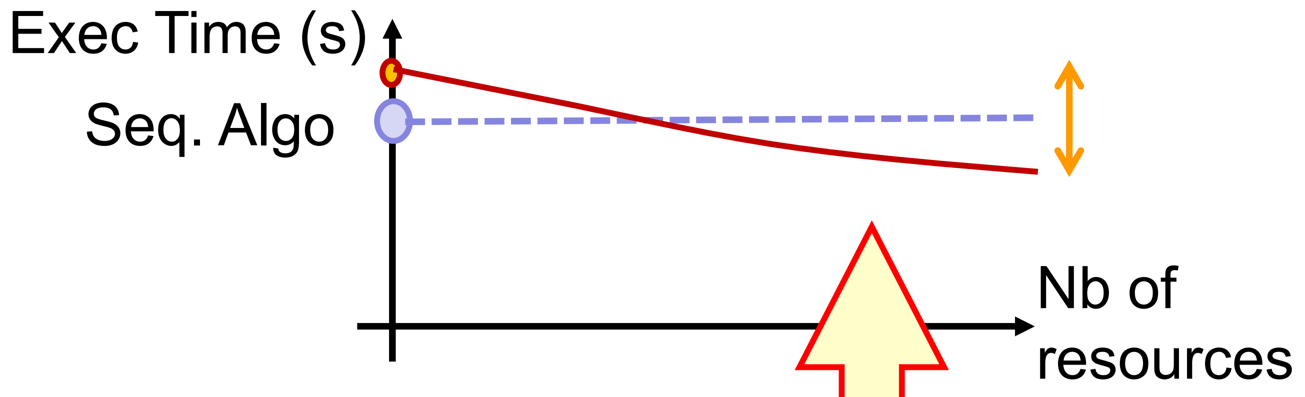
- Should performance analysis be pursued?
- Choice of an adapted representation

Should perf. analysis be pursued?

Disappointing time curve

When the additional costs of managing parallelism (*Tsynchro, Tcomm*) are high ...

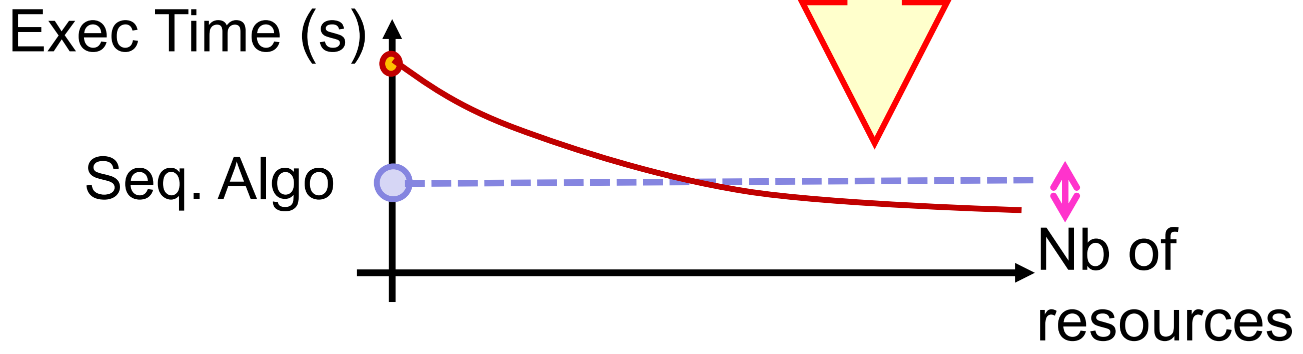
...the initial acceleration is low!



When the parallel algorithm is much more complex than the best sequential algorithm...

Stop the analysis!
Improve the algorithm!

...the final acceleration remains low!



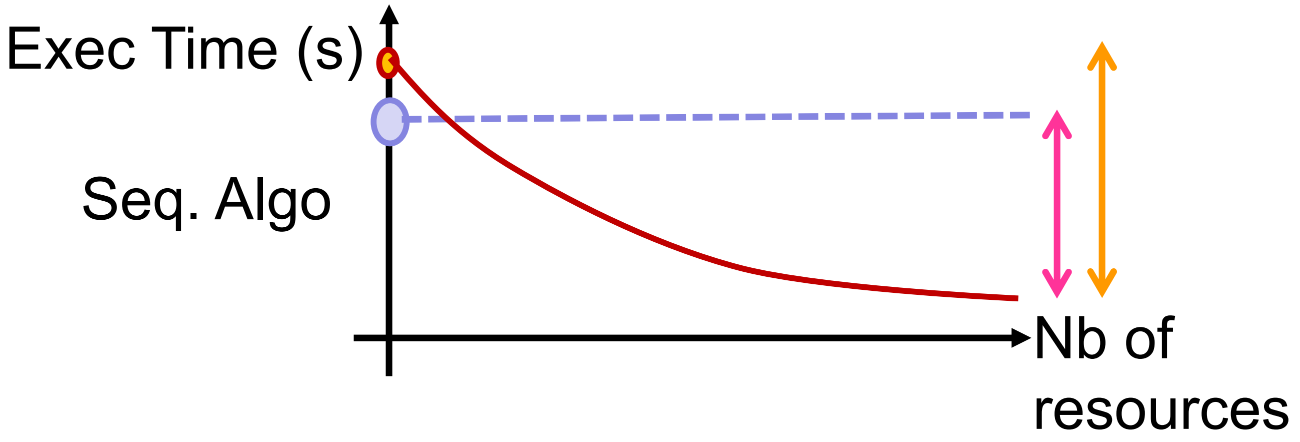
Should perf. analysis be pursued?

Promising time curve

When the parallel algorithm has :

- an execution time that decreases significantly
- a limited additional cost on a single resource

All accelerations will be great !

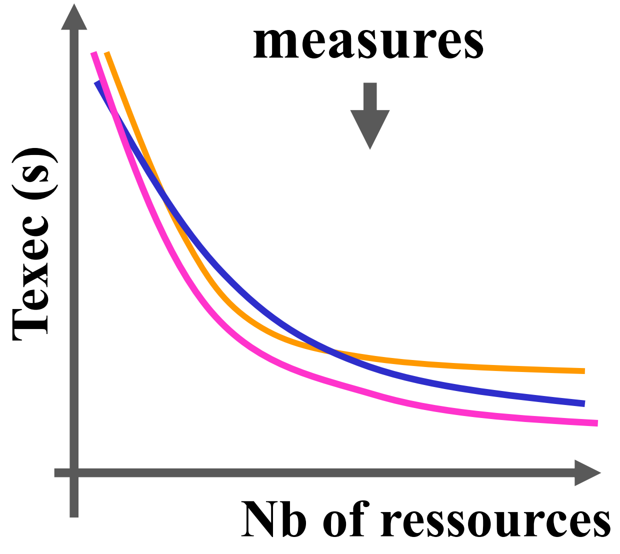


Interesting solution! Continue the analysis

Texec *vs* Texec ideal
Speedup
Efficiency
Size up
Scalability…

Choice of an adapted representation

Which representation to adopt for an execution time curve?



What is the « best » experimental curves?

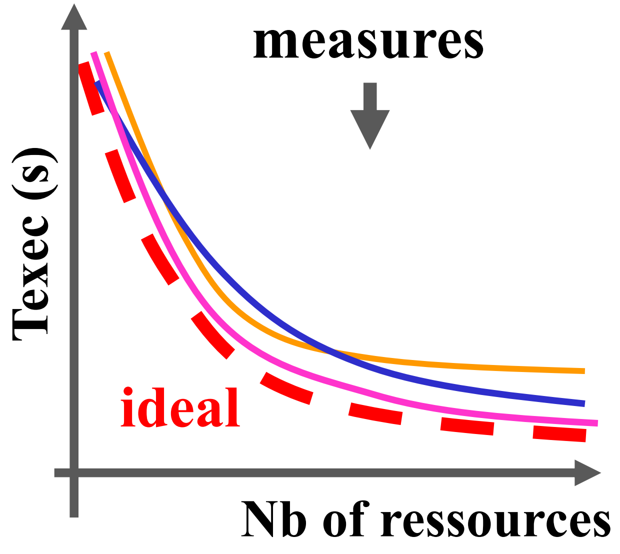
- We must compare the measurements to the theory, to the *ideal curve*
- We need a *representation* that is easy to verify *for human eyes*

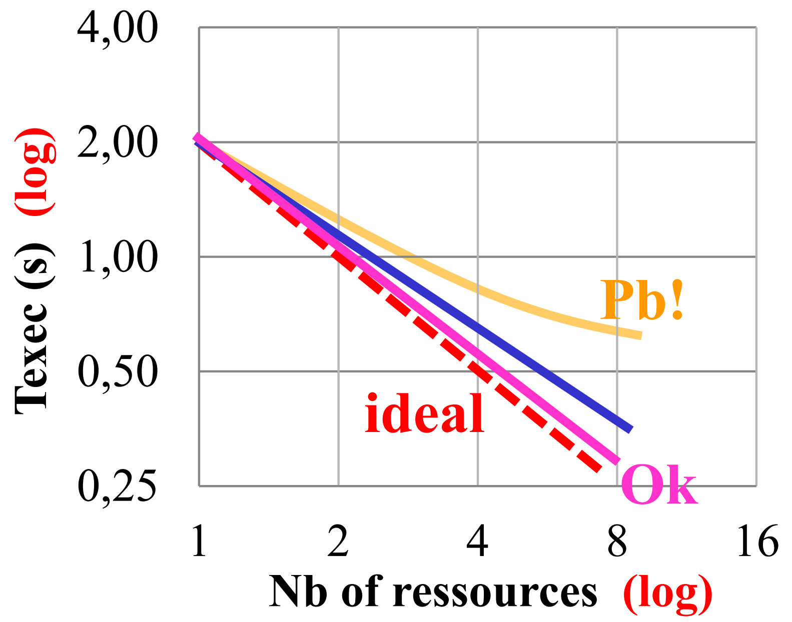
Case of a parallel execution time:

- Ideal curve: T(n ressources) = T(1)/n : a hyperbole
  → Check if we get a hyperbole
- But human eyes do not detect hyperboles
  → It is easily confused with a curve from another law!

Choice of an adapted representation

Which representation to adopt for an execution time curve?





Ideal curve: T(n) = T(1)/n

$$Y = \log(T(n)) = \log(T(1)) - \log(n)$$
$$Y = C^{te} - X$$

- With logarithmic scales, *ideal Texec* is a straight line of slope -1
- So you can easily detect:
  - a close mathematical law : $T(n) = T(1)/n^a$ → slope –a
  - a complete deviation from the theory

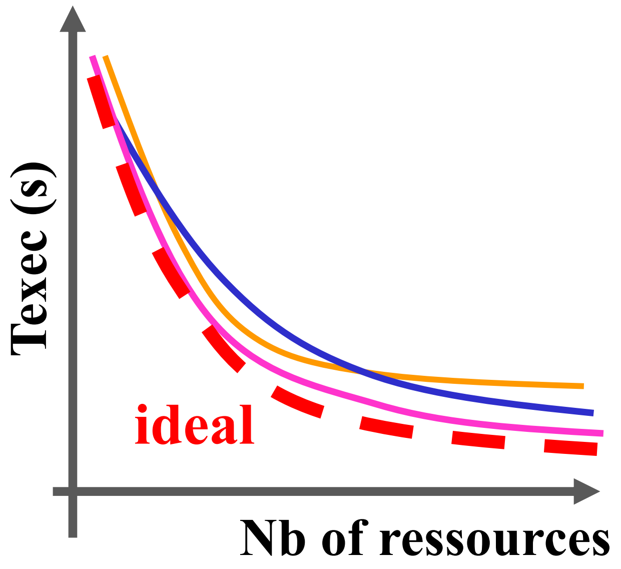
Choice of an adapted representation

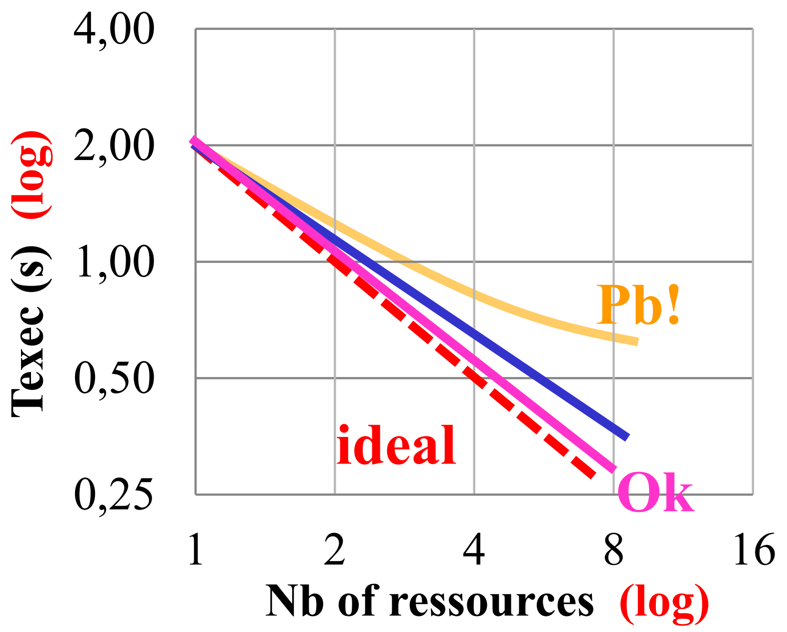
Which representation to adopt for an execution time?

Summary:

It is preferable to know the ideal/theoretical curve ...

...and to choose a representation that allows to visualize straight lines or simple geometric shapes.



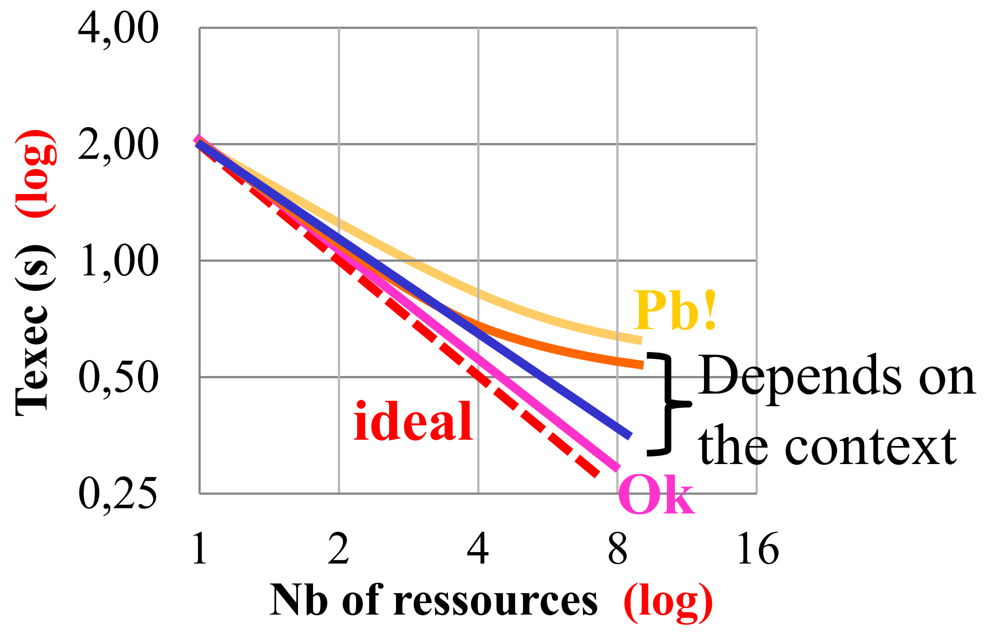


2 – Performance metrics (*T*, *S*, *e*)

- Execution time metrics
- Speed up metrics
- Efficiency metrics
- Sequential reference issue

Execution time metric

Execution time of a parallel/distributed application:



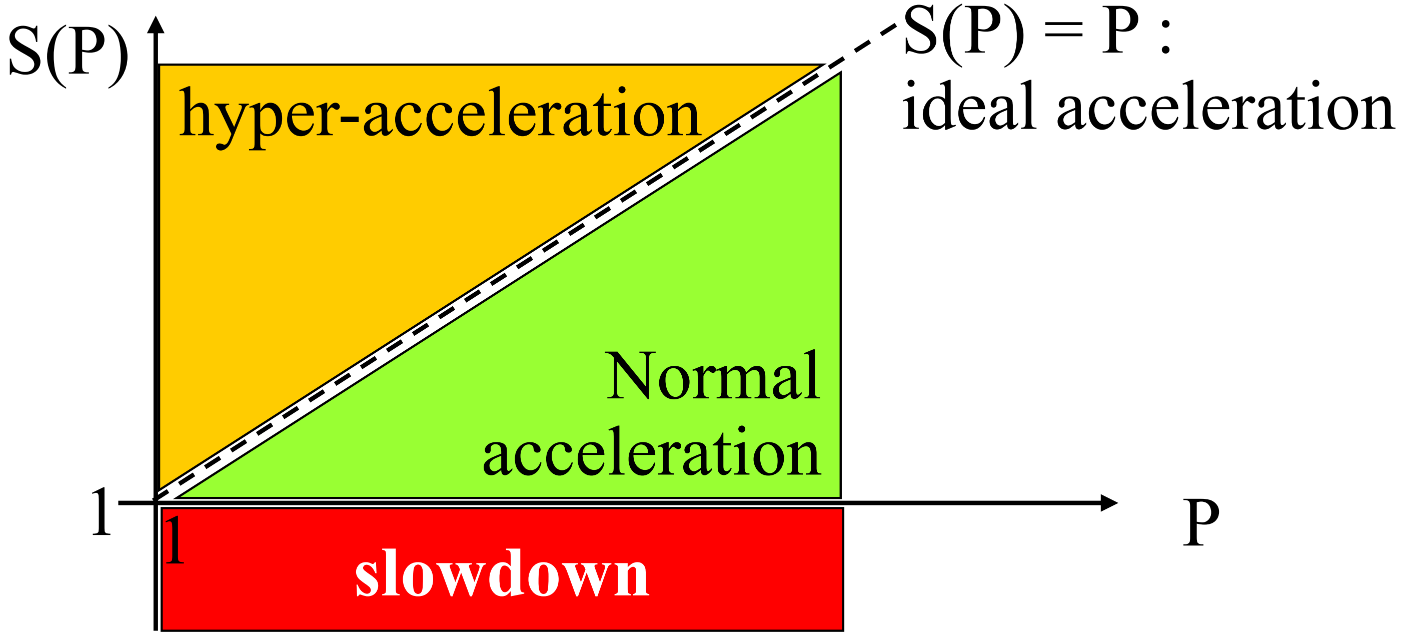
1. Is the execution time steadily decreasing?
   - As far as the number of resources exploited?
2. Is the deviation from the ideal curve acceptable?

Maintaining a constant decrease over a large number of resources, and close to the ideal decrease is very difficult.

Acceleration metric

Speedup:

$$\boxed{S(P) = \frac{T(1)}{T(P)}} \longrightarrow \begin{cases} S(P) < 1 : \text{we're slowing down! bad parallelization} \\ 1 < S(P) < P : \text{“normal”} \\ P < S(P) : \text{hyper-acceleration analyze \& justify} \end{cases}$$

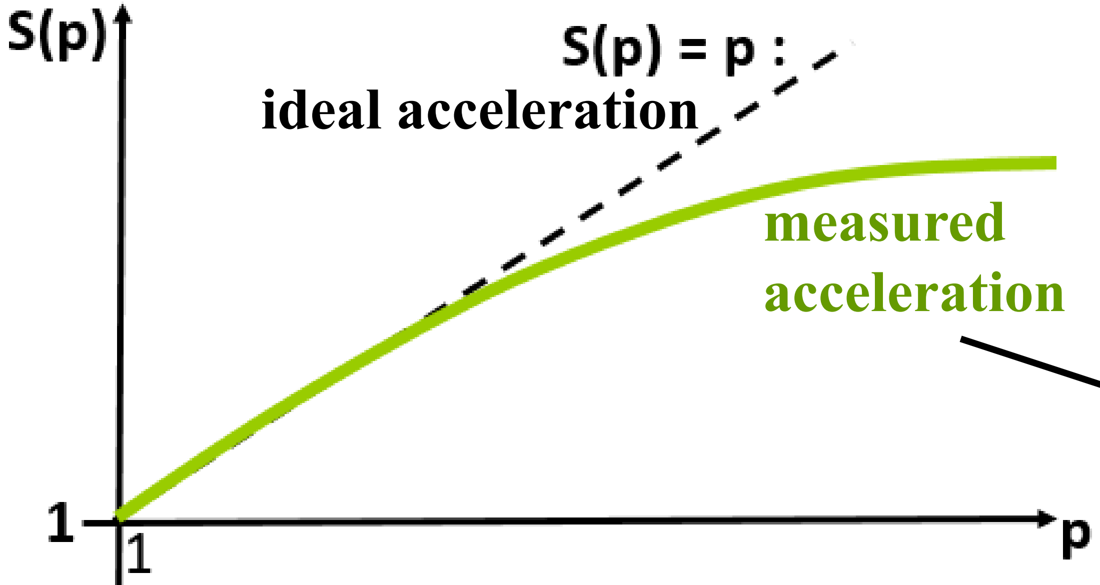


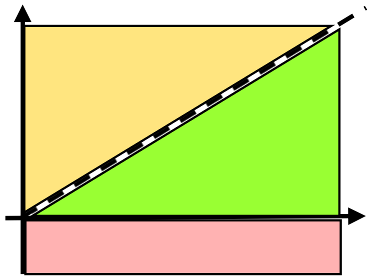
Acceleration metric

Speedup:

$$\boxed{S(P) = \frac{T(1)}{T(P)}} \longrightarrow \begin{cases} S(P) < 1 : \text{we're slowing down! bad parallelization} \\ 1 < S(P) < P : \text{“normal”} \\ P < S(P) : \text{hyper-acceleration analyze \& justify} \end{cases}$$

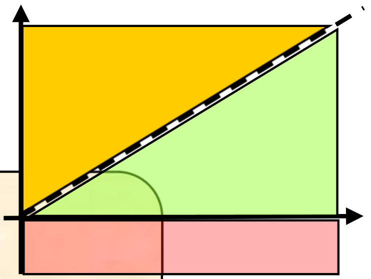
Standard case:





there are always sources of performance loss...

Acceleration metric

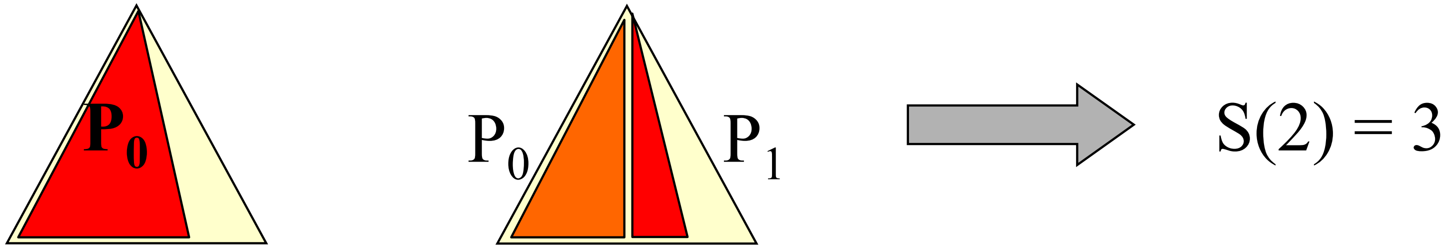


Cases of hyper-acceleration:

It's not magic, and it's not normal !
→ The phenomenon must be analyzed and explained
→ Correct an error or exploit an optimization

Examples of explanations :

- we no longer do the right operations (false result)
- the data fits in the total cache of the P processors
- we have modified the starting algorithm and converge faster (e.g. optimized genetic algorithm!).
- we look for a solution in a tree and stop the pgm



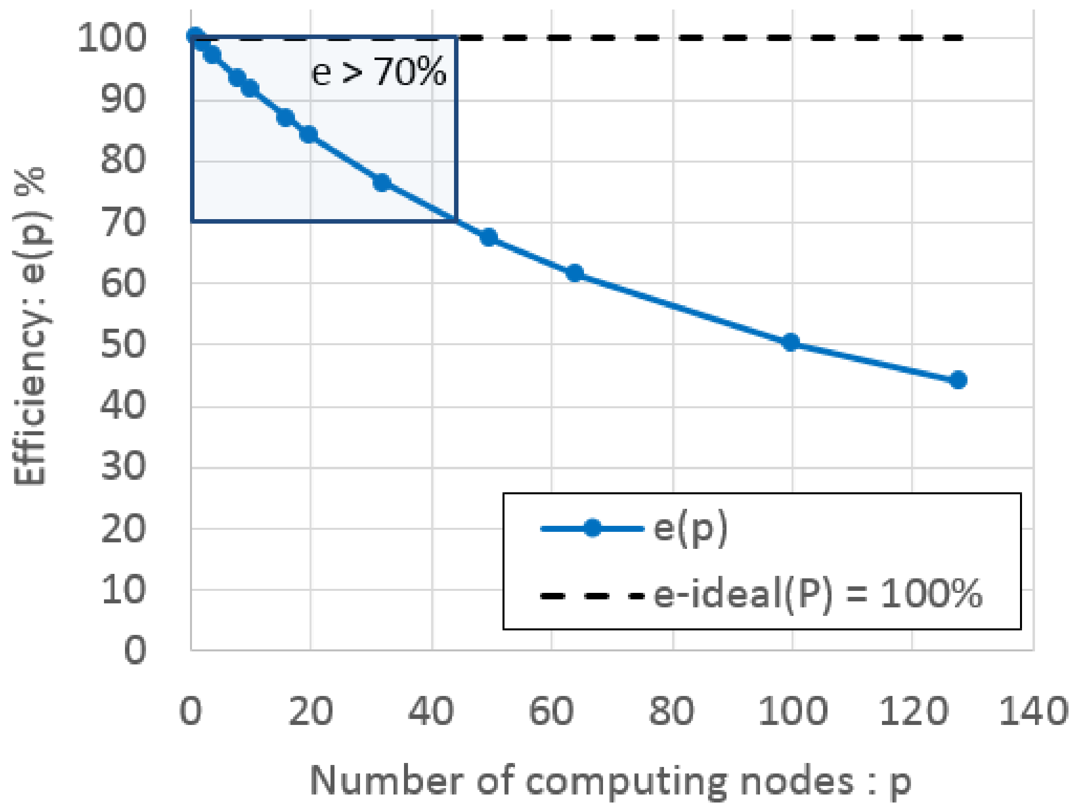
Efficiency metric

Efficacité :

$$\boxed{e(P) = \frac{S(P)}{P}}$$

Resource utilization rate, or
fraction obtained of the ideal acceleration

- $e(P) \in [0;1], \in [0\%;100\%]$
- $e(P) > 100\% \Leftrightarrow$ hyper-acceleration



The user is interested in the acceleration achieved

The buyer of the machine is interested in the efficiency of the applications

The **developer** is interested in **both**

Choice of the sequential reference

Which sequential execution to choose as a reference?

Same program running on a single processor?
Same algorithm implemented sequentially?
Best known sequential algorithm ?

Sequential compilation with the same compiler?
Compiling with the best sequential compiler ?

Sequential optimizations allowed by parallelization ?
Maximum sequential optimizations ?

Execution on a single processor of the parallel machine?
Execution on the best sequential machine ?

Choice of the sequential reference

Each sequential reference choice corresponds to:

- a different point of view
- a different business objective
- a different analysis objective

→ Make the choice corresponding to your problem

→ Clearly state this choice

Examples:

Final user → chooses HIS sequential pgm on HIS sequential machine

Parallel Code Developer → chooses HIS parallel pgm in ONE process of HIS parallel machine

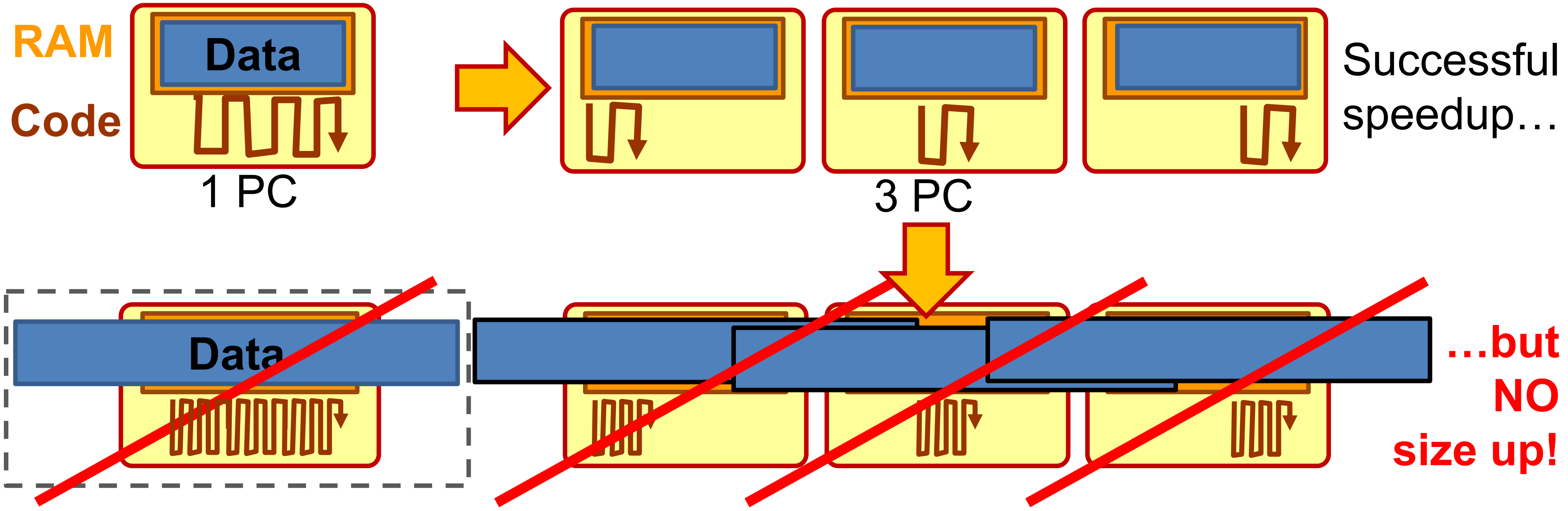
3 – *Size Up* metrics

- Design of a code suitable for size up
- Size up metrics in execution time
- Size up metrics in time and resources
- Size up in 3 successive objectives
- Experimental examples

Design of a code suitable for *size up*

1st objective: to be able to deal with bigger problems on more rsrcs

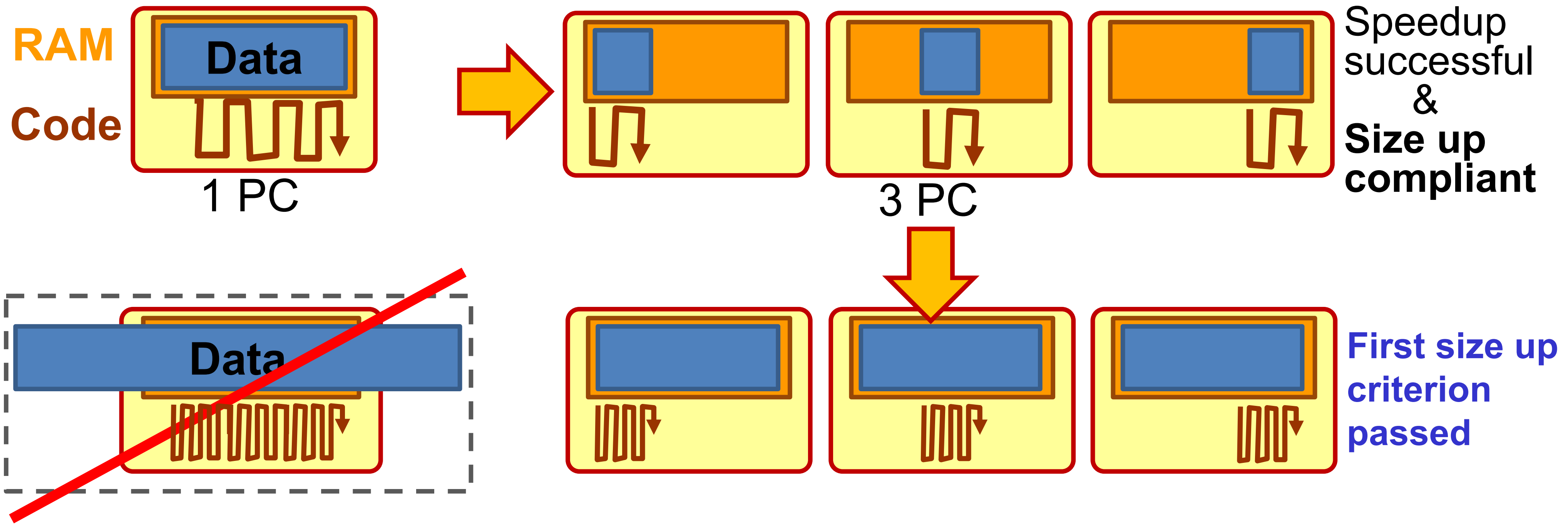
A code that replicates most of its data on all machines will always be limited by a machine's memory size...and will not be suitable for size up



Design of a code suitable for *size up*

1st objective: to be able to deal with bigger problems on more rsrcs

A code that distributes most of its data across all machines will be able to store more data on more machines...and will be suitable for size up

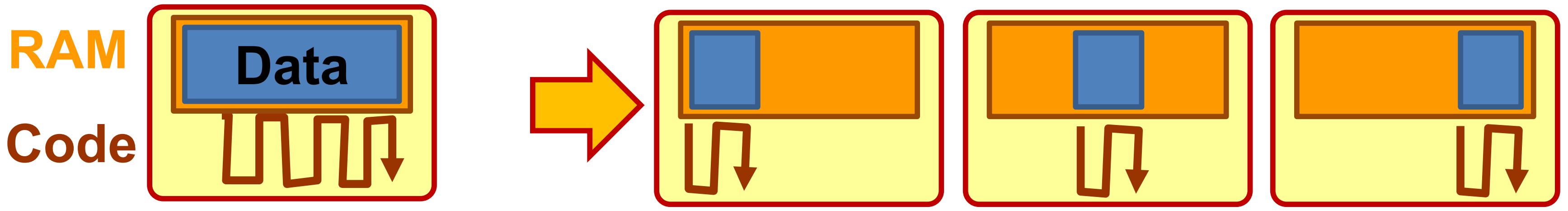


RAM
Code
Data
1 PC
3 PC
Speedup successful & Size up compliant
Data
First size up criterion passed

Design of a code suitable for *size up*

1st objective: to be able to deal with bigger problems on more rsrcs

→ Design of an initial distribution of data, and a minimum volume communication scheme



- **Need to circulate initial data between nodes:** so that a node can continue calculations on data other than its own.
- **Need to circulate intermediate results between nodes**: so that one node can continue another node's calculations with its own data

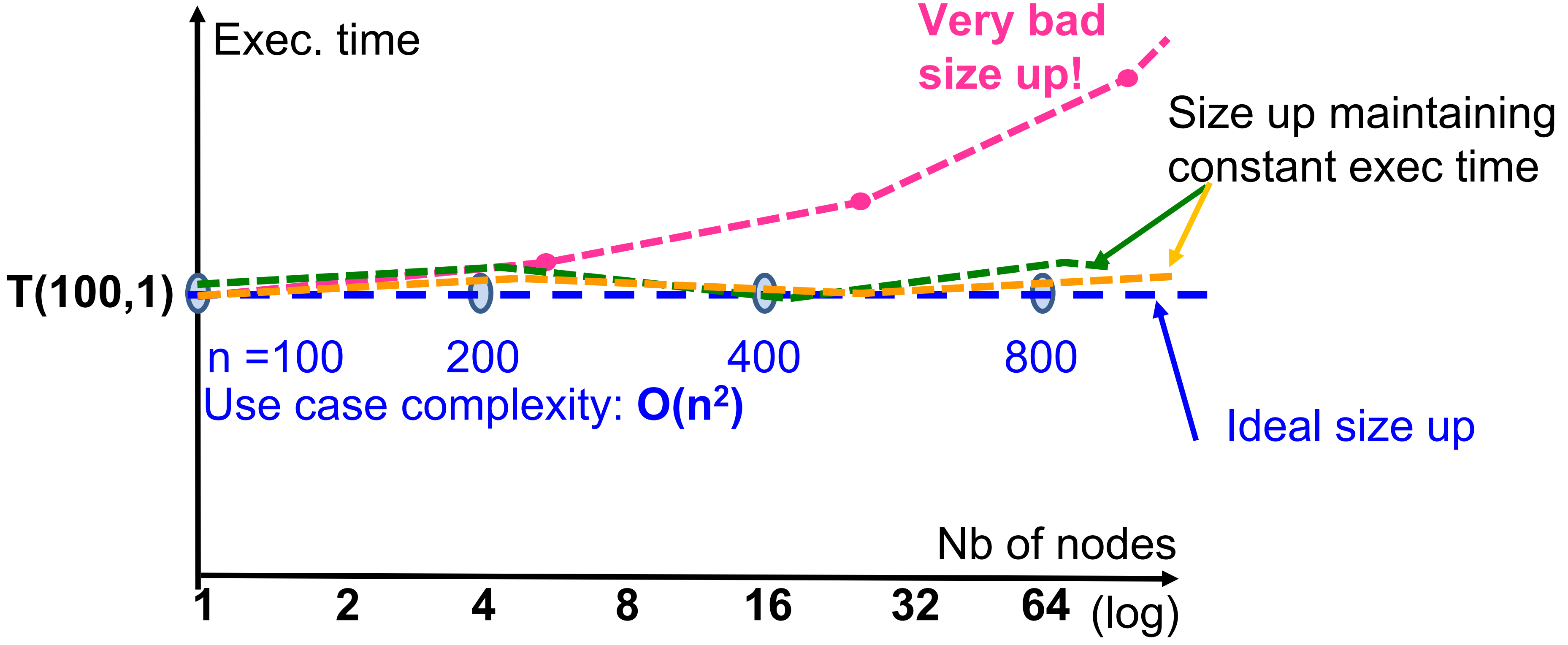
Metric of *size up* in T_{exec}

2nd objective: keep T_{exec} constant

$$T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$$

with: T(pb size, rsrc nb)

→ **Goal not achieved!**



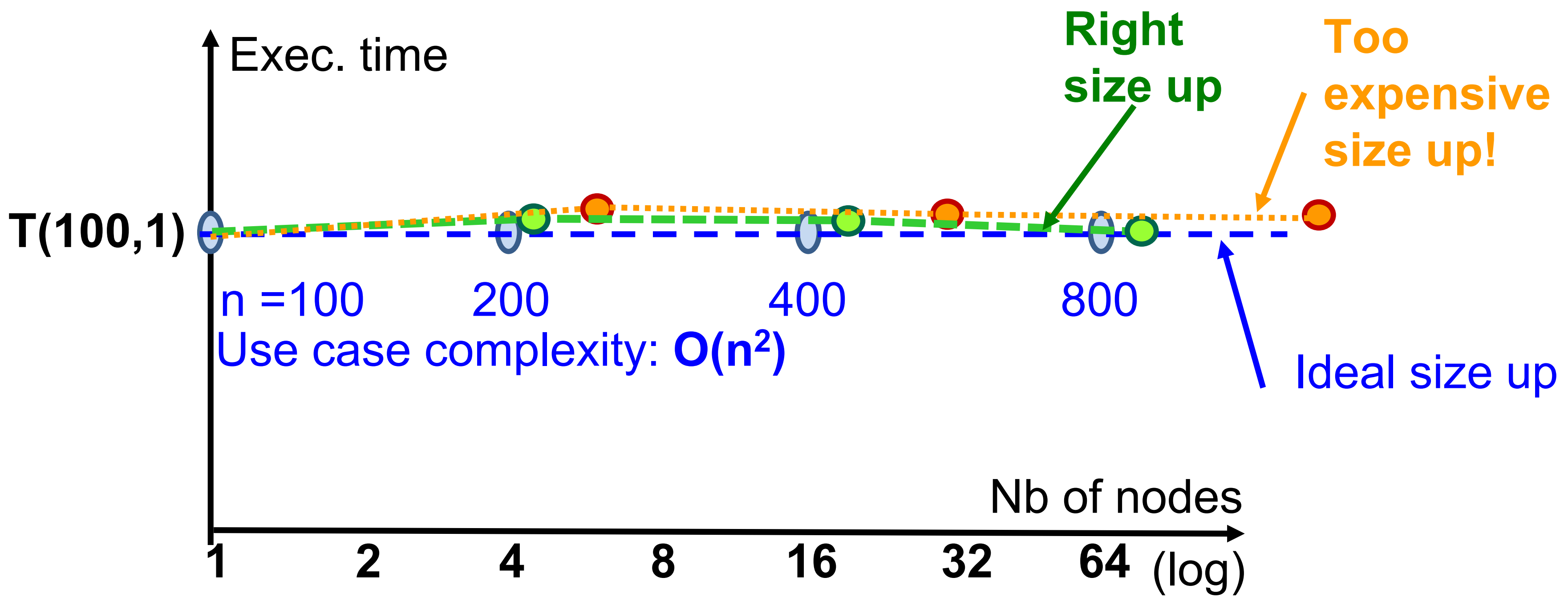
Metric of *size up* in T_{exec} <u>and</u> rsrcs

3rd objective : keep T_{exec} constant, with the minimum number of rsrc

$$\mathbf{T(1\times n_1,p_1) = T(2\times n_1,p_2) = T(k\times n_1,p_k) = C^{te}}$$

with: T(pb size, rsrc nb)

with: $O(p_k) = O(calcul(k\times n_1))$



→ **Goal not achieved!** → **Goal achieved**

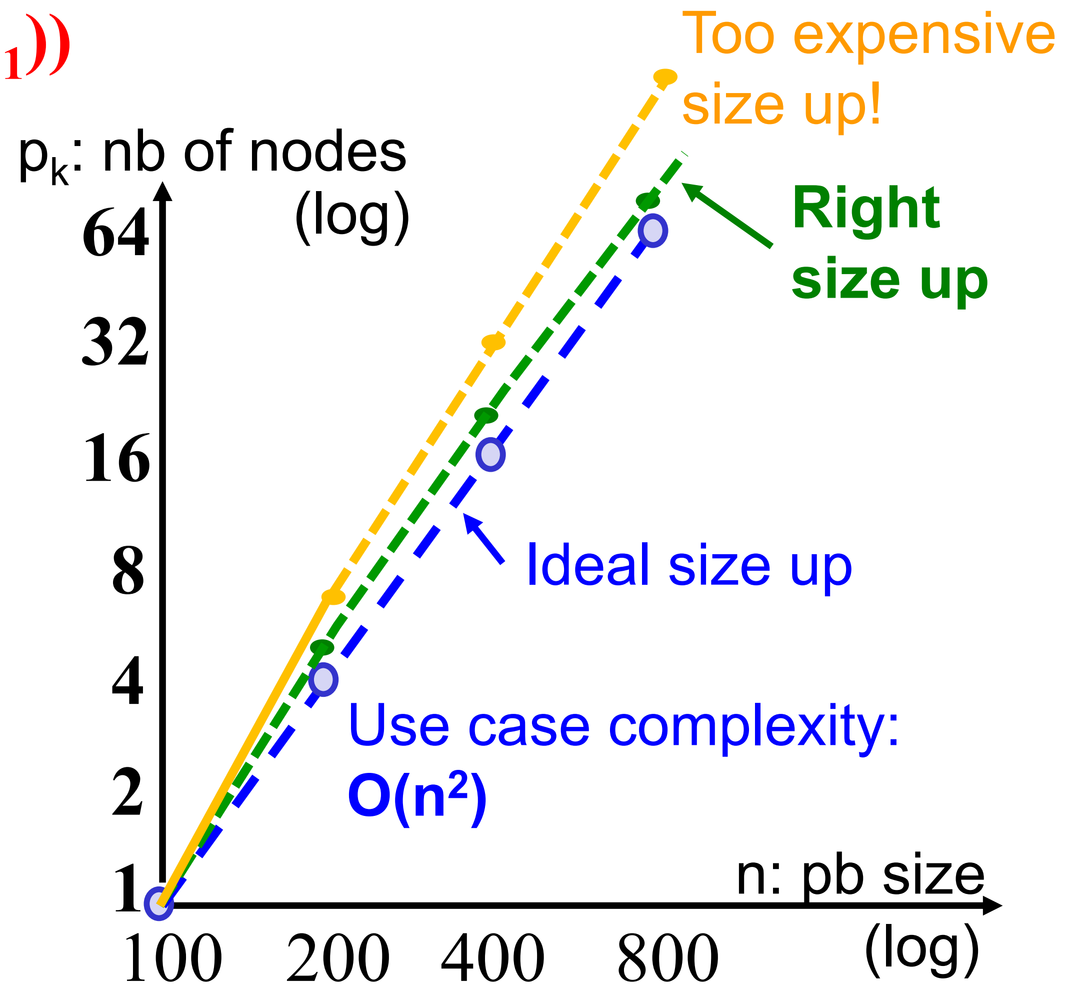
Metric of *size up* in T_{exec} and rsrcs

3rd objective : keep T_{exec} constant, with the minimum number of rsrc

$$T(1\times n_1,p_1) = T(2\times n_1,p_2) = T(k\times n_1,p_k) = C^{te}$$

with: T(pb size, rsrc nb)

with: $O(p_k) = O(calcul(k\times n_1))$

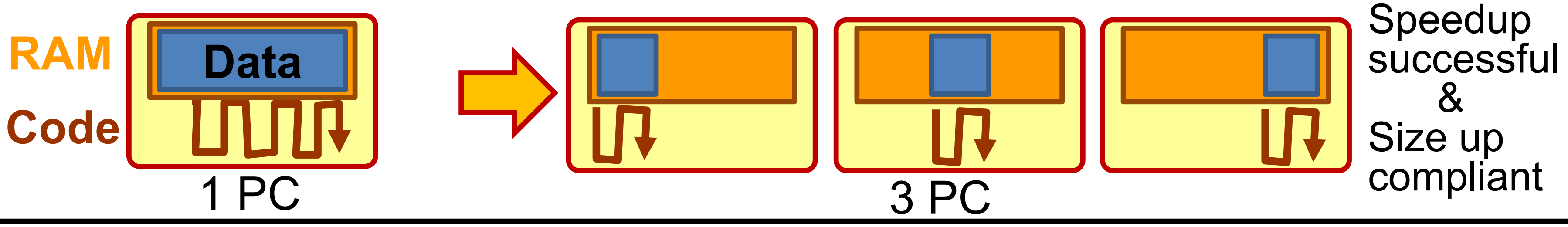


→ **Goal not achieved!**

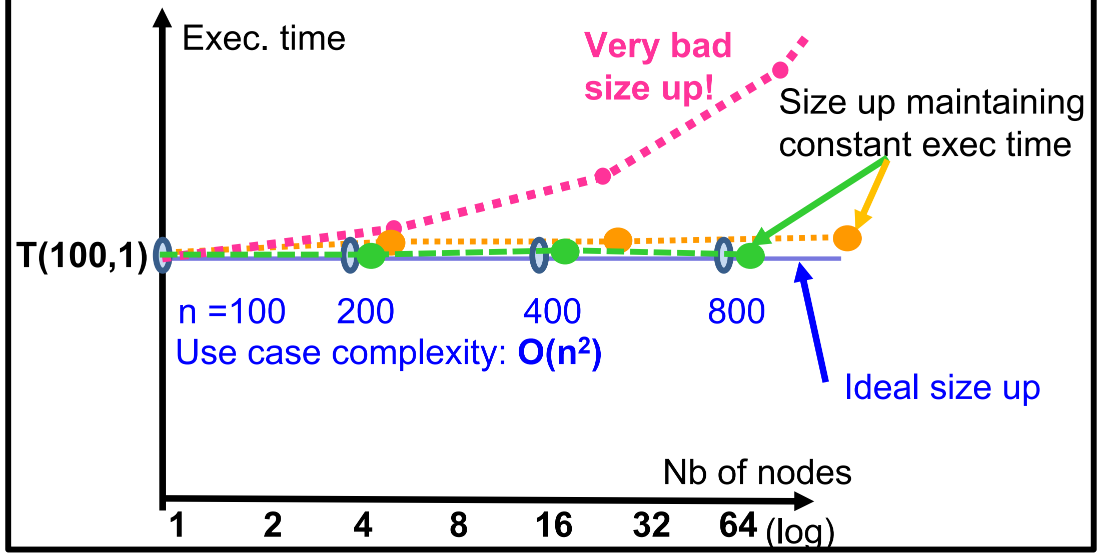
→ **Goal achieved**

Size up in 3 successive objectives

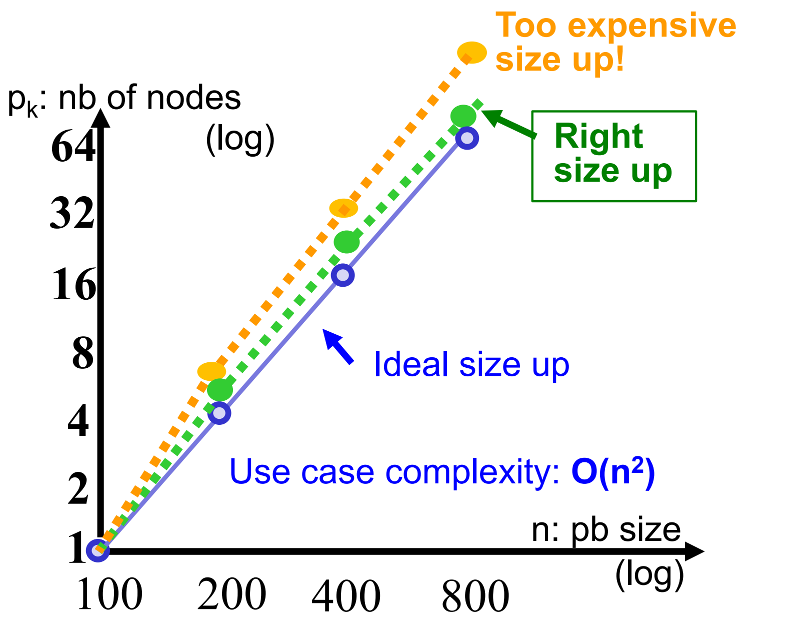
1st objective: to be able to deal with bigger problems on more rsrcs



2nd objective: keep $T_{exec} = C^{te}$



3rd objective: $T_{exec} = C^{te}$ with the min nb of rsrc



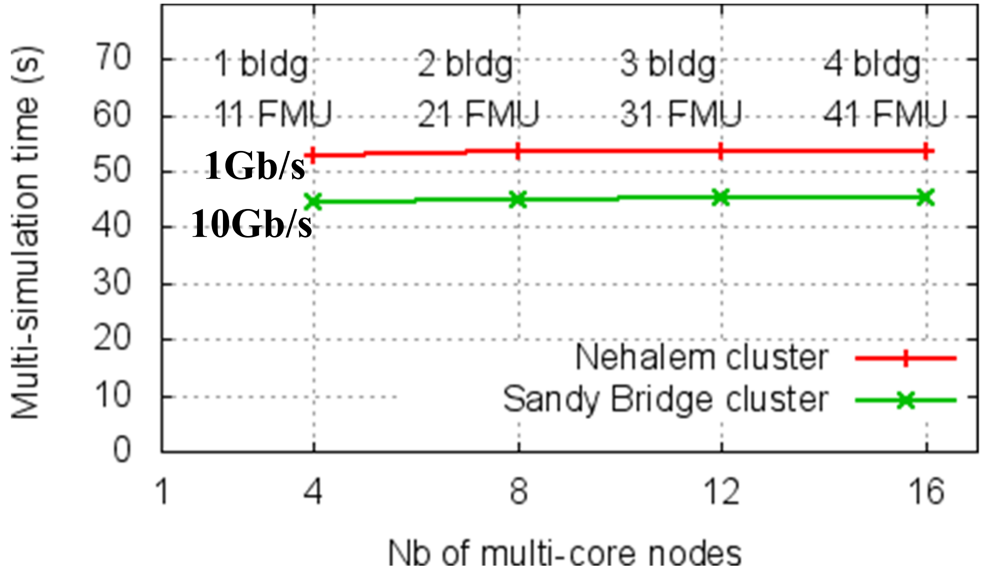
Experimental example

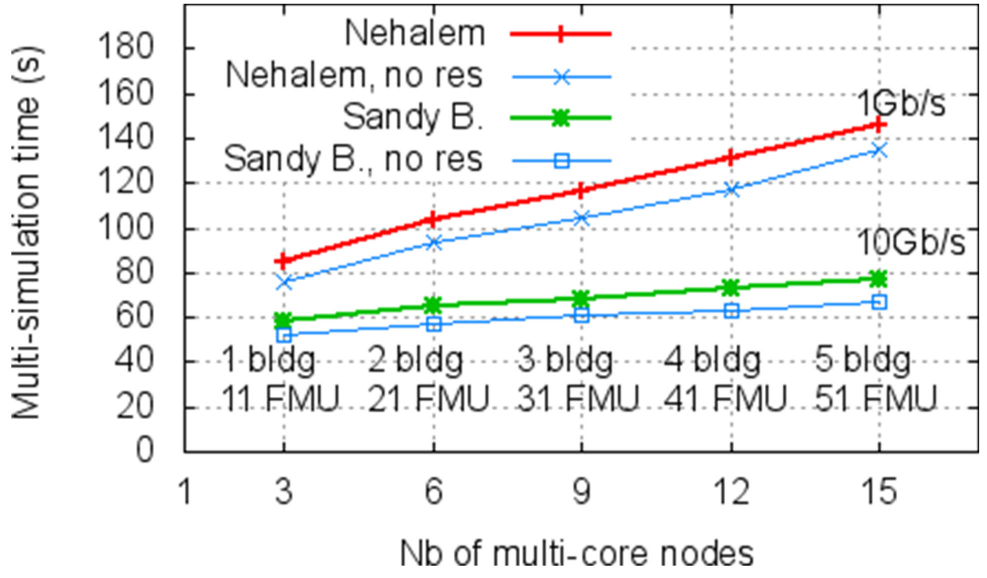
Co-simulation of heat exchanges within buildings weakly coupled (quasi-independent calculations)

Objective: size up at constant time with minimum nb of rsrc

$T(1\times 1\text{building},p_1) = T(2\times 1\text{bldg},p_2) = T(k\times 1\text{bldg},p_k) = C^{te}$

with: $O(p_k) = O(\text{calcul}(k\times 1\text{bldg})) \approx O(k)$





A lot of calculations and little comm. → Size up from 1Gb/s

Few calculations and little comm. → Size up from 10Gb/s

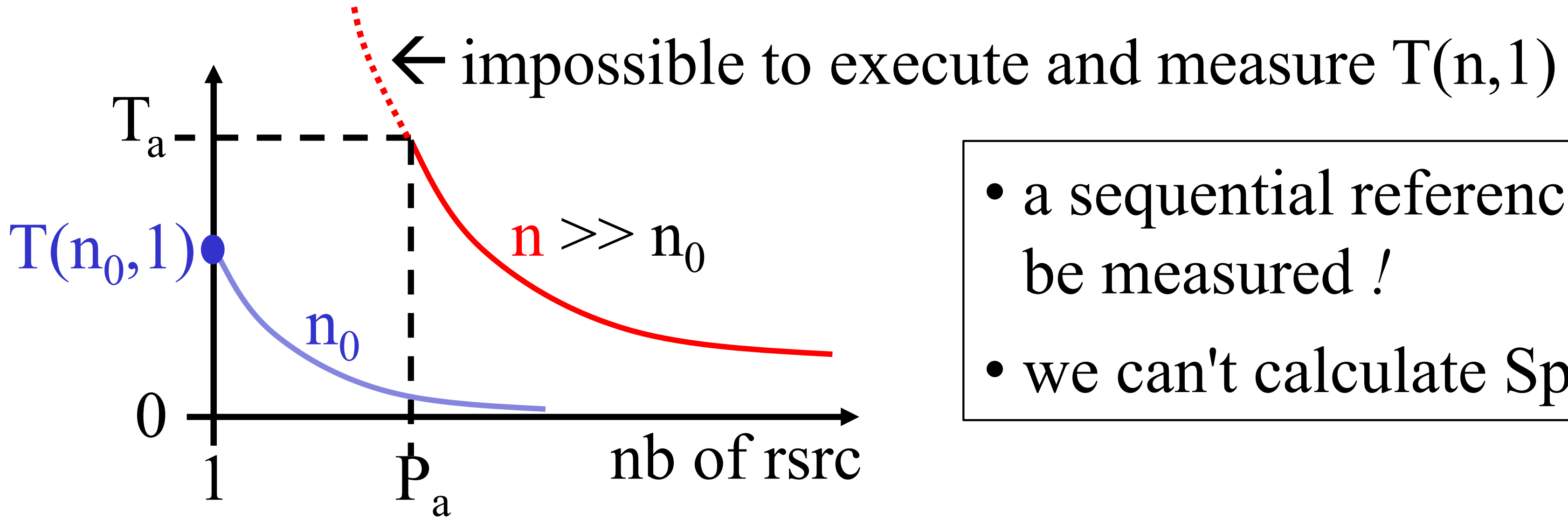
4 – Criteria for scaling

- A metric based on execution time
- *Size Up* + *Speedup* combined approach
- Scaling graphs

A metric based on Texec

As the size of the pb. grows, sequential exec. is no longer possible:

- not enough RAM, not enough disk...
- execution too long, resources cannot be monopolized...



- a sequential reference cannot be measured !
- we can't calculate Speedup !!

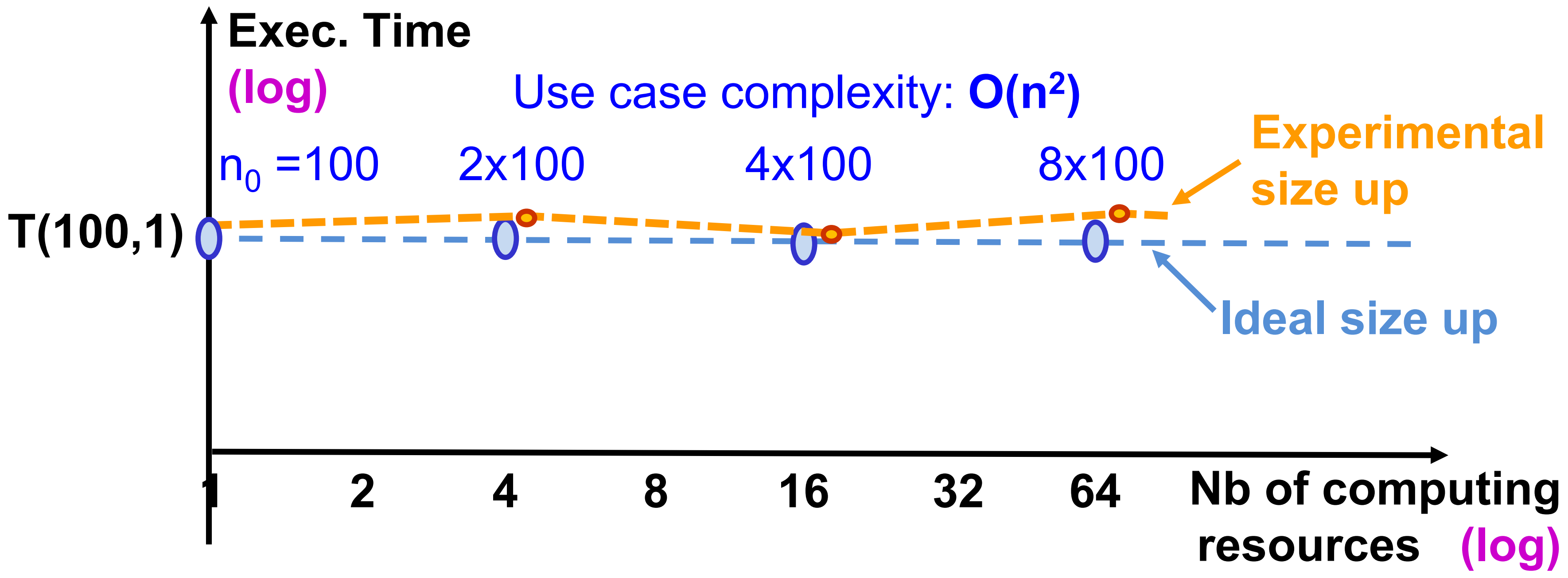
→ A *scaling* analysis should be built only on measures of T_{exec}

Size Up & Speedup approach

Definition and criteria for "simple" scaling:

- *Enable a "size up", in constant time,*
- *and in an economically bearable way*

Level-3 size up



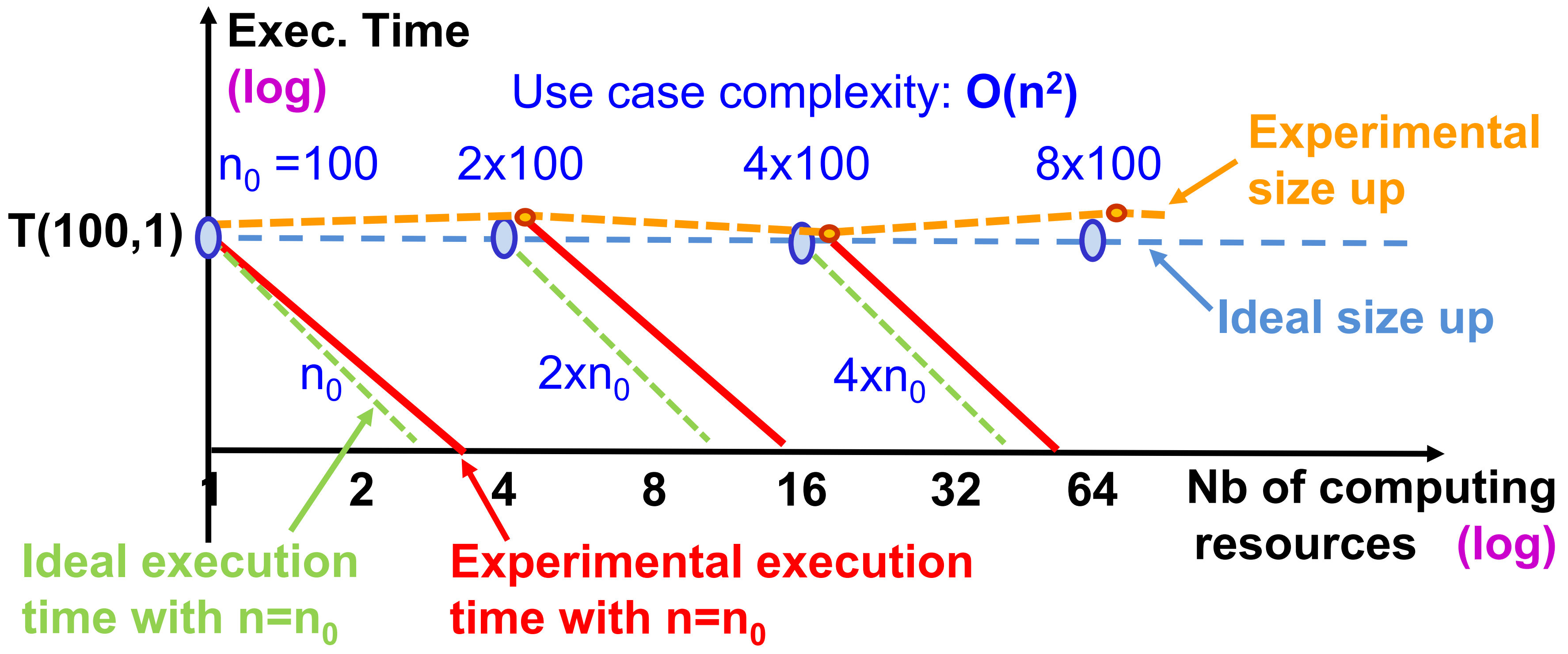
Size Up & Speedup approach

Definition and criteria for "complete" scaling:

- *Enable a "size up", in constant time,*
- *and in an economically bearable way*

&

- *Enable to "speedup" for all problem sizes*
- *with always the same profile of decrease in execution time*

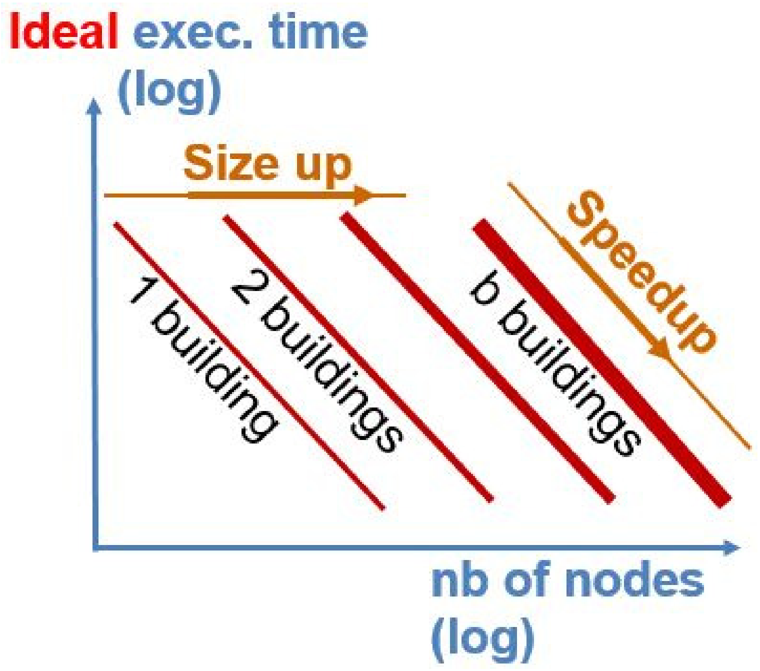


Scaling graph

Creation and use of a scaling graph:

- Measure T(n,p) for different sizes of pb (n) and rsrc (p)
- Draw T(n,p) curves in logarithmic scale

Ideal case:
parallel straight lines!



“ To be able to deal with problems of unlimited size in the future ”

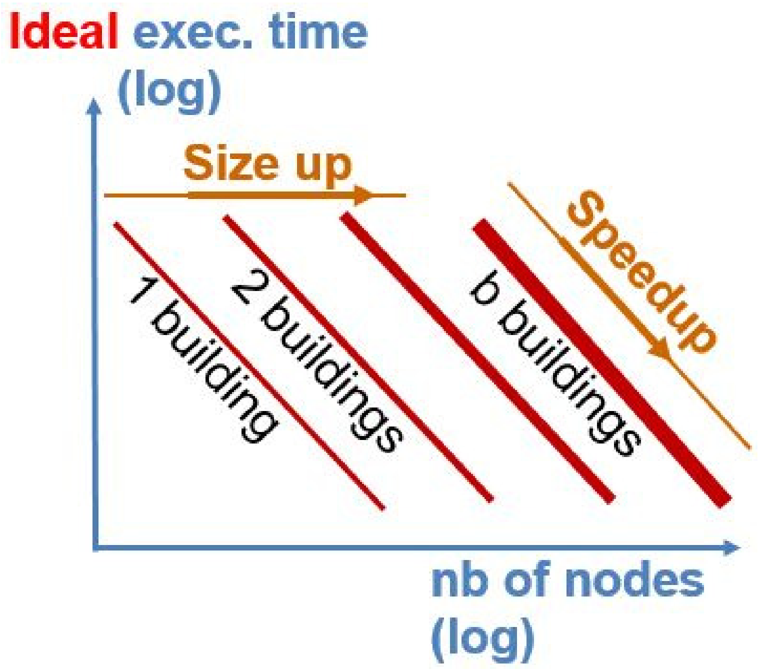
Scaling graph

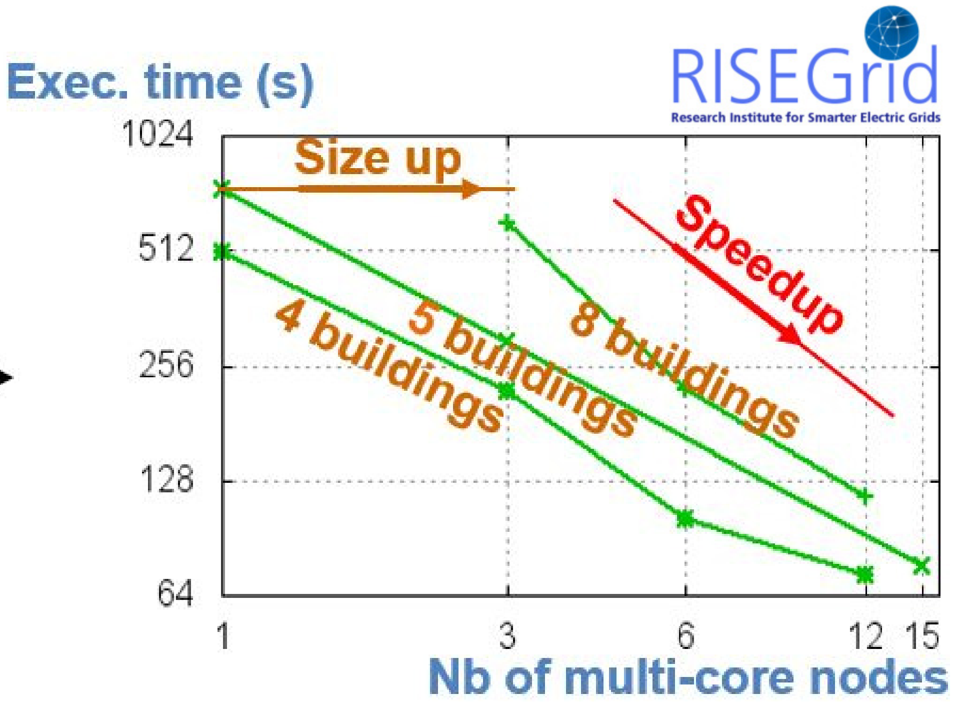
Creation and use of a scaling graph:

- Measure T(n,p) for different sizes of pb (n) and rsrc (p)
- Draw T(n,p) curves in logarithmic scale

Ideal case:
parallel straight lines!

Real case:
roughly parallel curves…



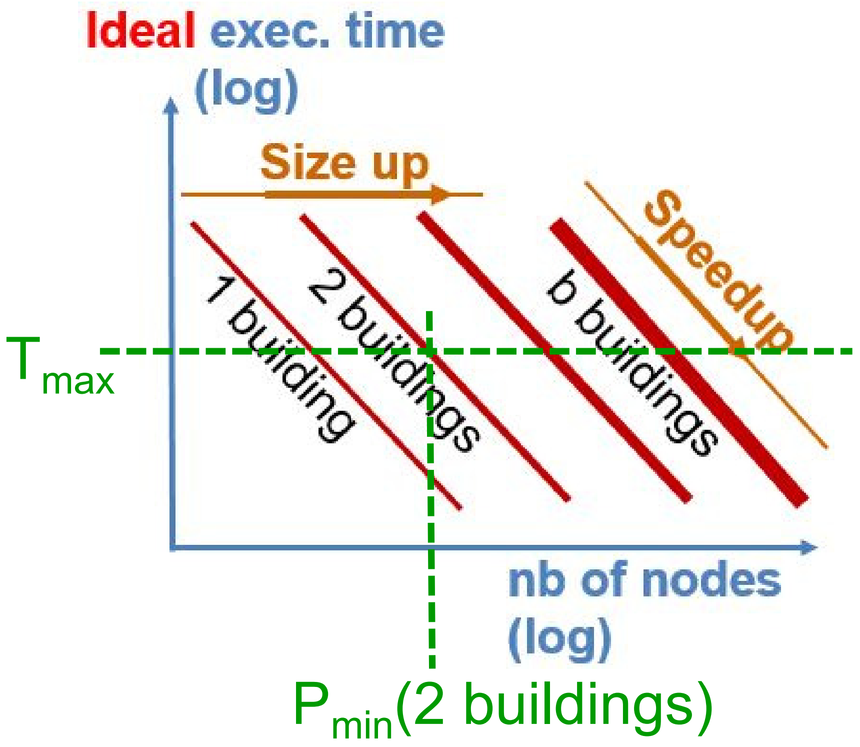


Scaling graph

Creation and use of a scaling graph:

- Measure T(n,p) for different sizes of pb (n) and rsrc (p)
- Draw T(n,p) curves in logarithmic scale

Use in "abacus" mode: For a given problem size, identify the number of rsrcs to use in order to respect a maximum T-exec



A fully scalable solution makes it possible to :

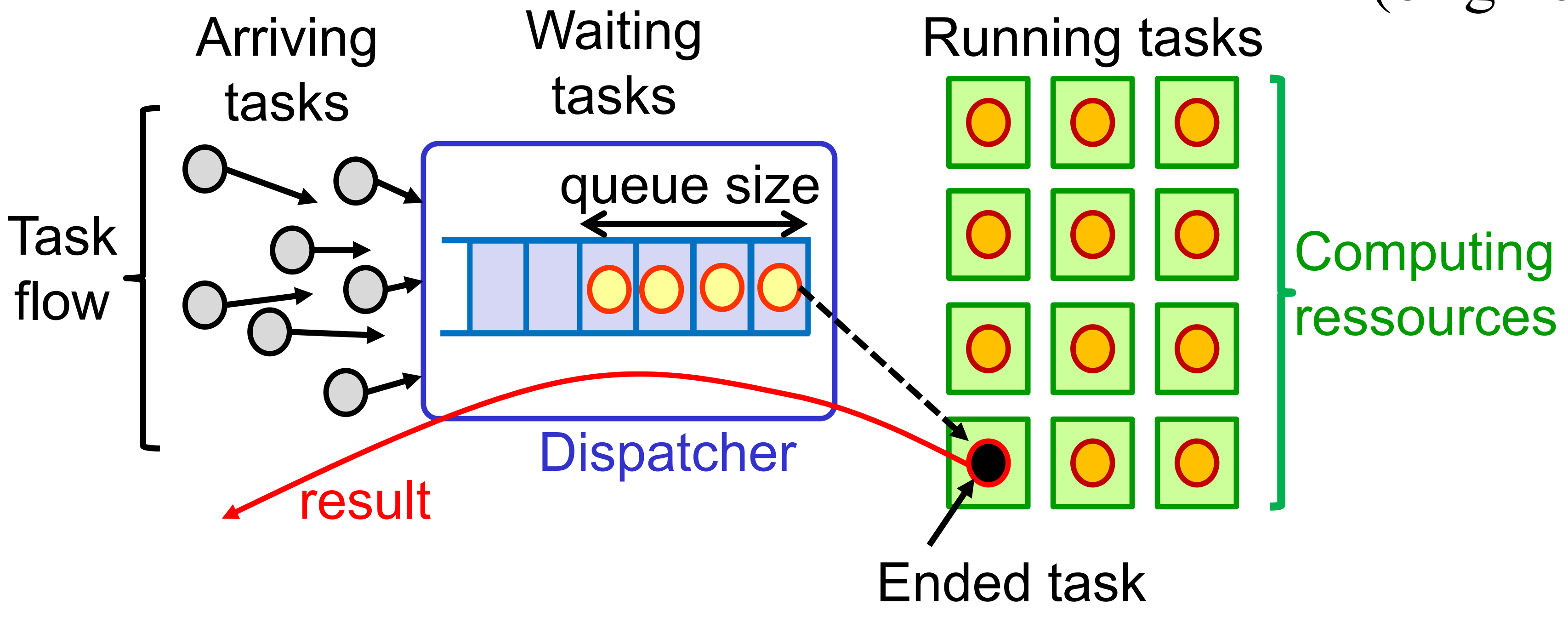
- meet the needs of calculations
- require the minimum amount of resources
- quantify the necessary resources
- plan the associated expenses

5 – Métriques d’analyse de flux de requêtes

- Architecture logicielle de flux
- Métrique de flux
- Analyse de pics de charge

Architecture logicielle de flux

(origine HTC)



Système de base:

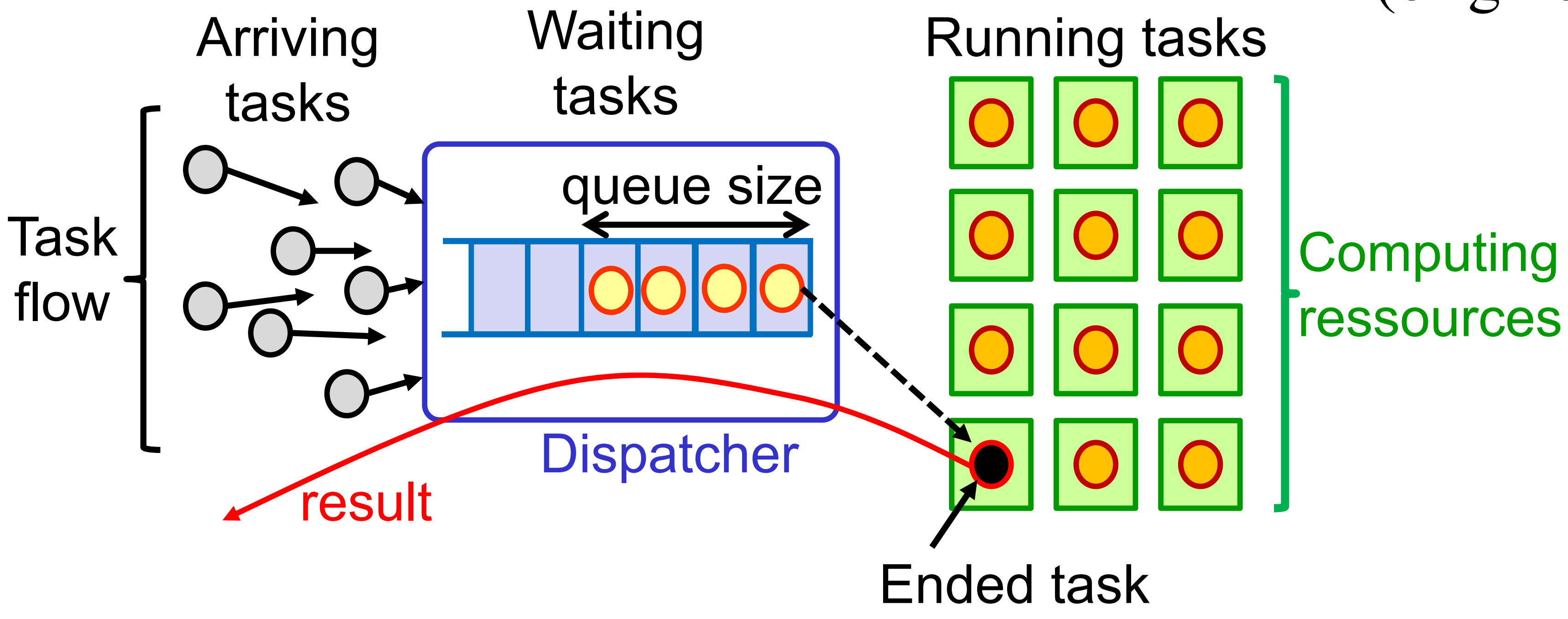
- Tâches identiques (mêmes calculs, données différentes)
- Tâches séquentielles

Systèmes plus complexe :

- Tâches différentes (hétérogènes)
- Taches séquentielles ou *multithreads*

Métriques de flux

(origine HTC)



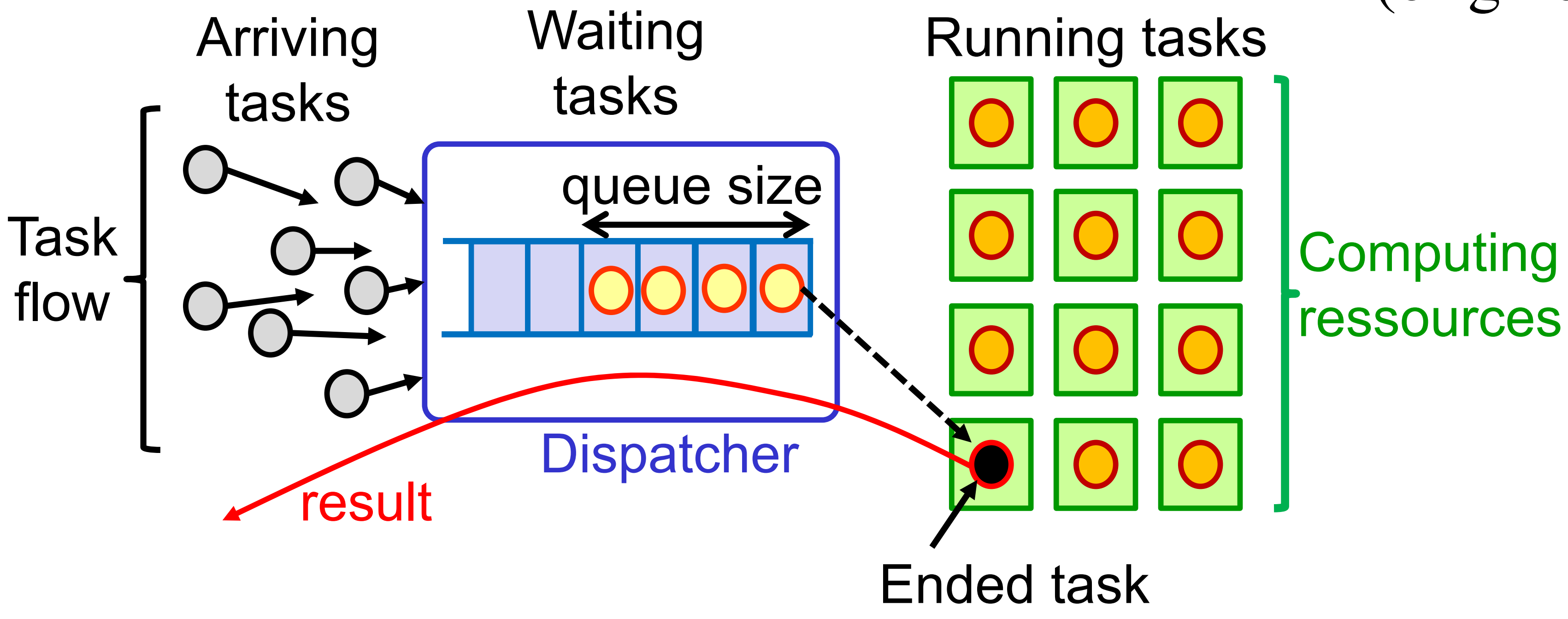
Métriques de flux :

1. Vitesse de traitement des tâches (tâches/s, tâches/min…)
2. Temps de traitement d’une tâche (T_{max}, T_{moy} …)

→ Maintenir $T < T_{max}$, et $V > V_{flow}$

Métriques de flux

(origine HTC)



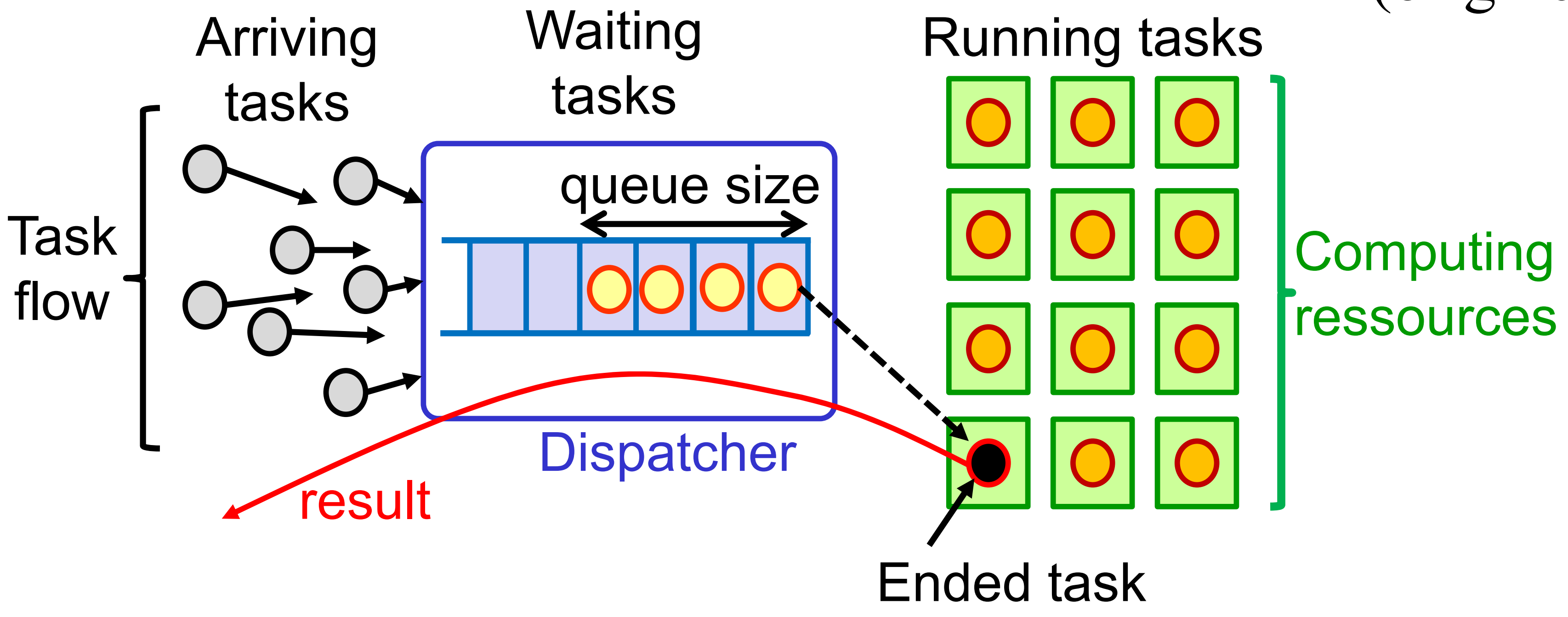
Métriques de flux :

3. Taille de la file d'attente (en nbr de tâches)
4. Temps de latence d'une tâche ($t_{\text{démarrage}} - t_{\text{soumission}}$)

→ Si *Taille de file* ou *Temps de latence* augmente
Alors augmenter le nombre de ressources de calcul
(ou améliorer le *dispatcher*…si pb)

Analyse de pics de charge

(origine HTC)



Pour étudier les pics de charge :

Métrique de moyenne ⇨ Métrique instantanée

- V → V(t)
- T_{max} → $T_{max}(t)$
- Taille → Taille(t)
- $T_{latence}$ → $T_{latence}(t)$

6 – Distributed execution of a Spark task graph

- Execution trace
- Performance measures & analysis

Execution trace

Spark job trace: on 10 Spark executors, with 3GB input file

```
DAGScheduler: Submitting 24 missing tasks from ShuffleMapStage 0 ...
TaskSchedulerImpl: Adding task set 0.0 with 24 tasks
...
TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 172.20.10.14, executor 0, partition 1, ...)
TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, 172.20.10.11, executor 7, partition 2, ...)
...
TaskSetManager: Starting task 10.0 in stage 0.0 (TID 10, 172.20.10.11, executor 7, partition 10, ...)

TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 18274 ms … (executor 7) (1/24)
TaskSetManager: Starting task 11.0 in stage 0.0 (TID 11, 172.20.10.7, executor 8, partition 11, ...)
TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 18459 ms … (executor 8) (2/24)
...

TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks
                   have all completed, from pool
...
```

Beggining of task graph execution

Submitting the 10 first tasks on the 10 Spark executor processes

Submitting a new task when a previous one has finished

End of task graph execution

Performance measures & analysis

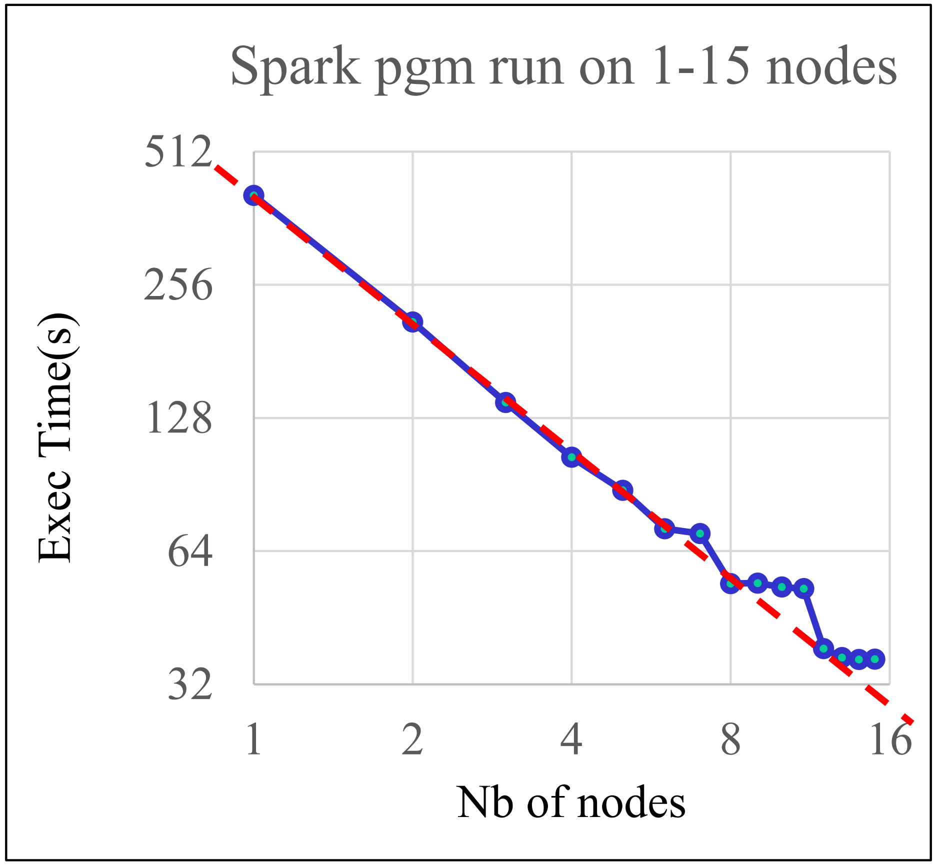
Execution time as a function of the number of Spark executors

Ex. of Spark application run:

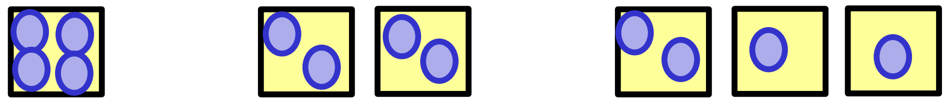
- from 1 up to 15 executors
- with 1 executor per node

Good overall decrease but plateaus appear !

Probable **load balancing problem**…



Ex: a graph of 4 parallel tasks



on 1 node: T

on 2 nodes: T/2

on 3 nodes: T/**2**

→ A plateau appears

QUIZ

QUIZ

1. Vous devez acheter une application distribuée « A » pour la faire tourner sur 4 noeuds, et vous disposez de l’information suivante :
    - « l’application A1 exhibe un speedup de 3.9 sur 4 nœuds »
    - « l’application A2 exhibe un speedup de 3.0 sur 4 nœuds »

   Que faites-vous ?

2. Vous devez acheter une application distribuée « A », et vous disposez de l’information suivante :
    - « l’application A1 traite 2.10^4 documents/s sur 10 nœuds »
    - « l’application A1 traite 3.10^4 documents/s sur 10 nœuds »

   Que faites-vous si vous avez 10 nœuds ?

   Que faites-vous si vous avez 20 nœuds ?

QUIZ

3. Citez des métriques de performances absolues et de performances relatives

4. Qu’est ce qui peut empêcher un système (hard & soft) de passer à l’échelle ?

5. On considère une application sous la forme d’un graphe de tâches très hétérogènes (des traitements longs et d’autres courts)
   - Si on augmente le nombre de nœuds sans augmenter le nombre de tâches (i.e: la taille du pb), va-t-on obtenir un bon speedup ?
   - Un « passage à l’échelle » reste-t-il possible ?

QUIZ

6. Vous devez augmenter le nombre de nœuds de calcul pour traiter des problèmes plus importants dans le même temps.

   Au lieu de cela on vous propose de passer tous vos nœuds de 1 à 2 processeurs, sans autre changement dans votre cluster (solution moins onéreuse…).

   Que devez-vous vérifier avant d’opter pour un passage à l’échelle avec cette solution ?

Appendix

A - Methodology for measurement of execution time

Méthodologie de mesures

Mesures externes :

```
>time myAppli
>/usr/bin/time myAppli
>times myAppli
>timex myAppli
......
```

```
12.002u   user
 0.128s   system
12.150    total
```

Nom et fonctionnement variables selon le système utilisé !!

Fréquemment :
total > user + system !!

Simple à utiliser
Pas de modifications des codes sources

Peu précis: ± 0.5s

Méthodologie de mesures

Mesures internes :

```
time()
clock()
gettimeofday()
omp_get_wtime()
```

$\longrightarrow$ Compte les clicks d'horloge

$\longrightarrow$ Compte le temps écoulé en s

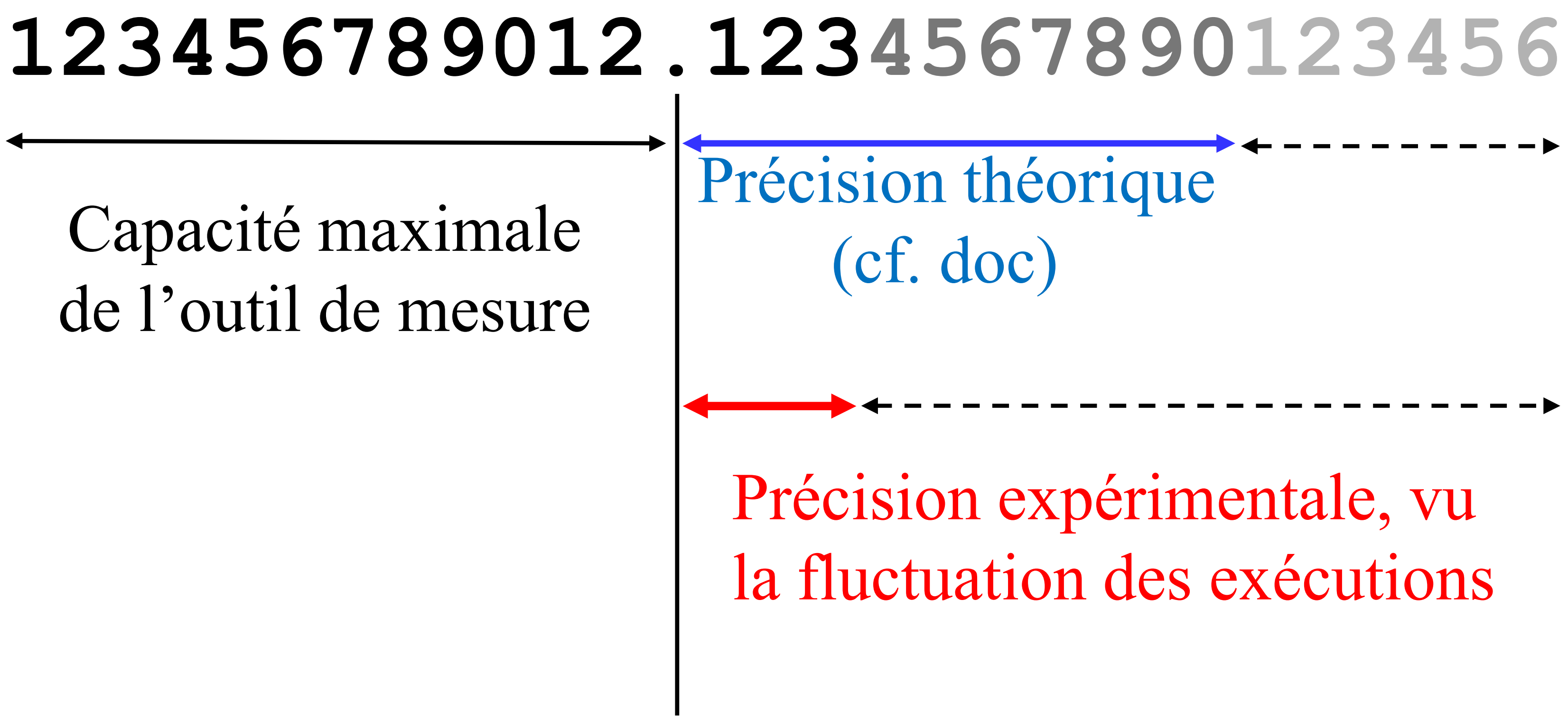
- Toutes ces routines ne sont pas toujours disponibles !
- "gettimeofday" est en général une bonne solution.
- Parfois il existe des outils plus précis pour mesurer de petites durées.

Plus précis que les mesures externes

Besoin de modifier le code source
Pas toujours totalement portable

Méthodologie de mesures

Précision des outils et des mesures :


123456789012.1234567890123456
Capacité maximale de l’outil de mesure
Précision théorique (cf. doc)
Précision expérimentale, vu la fluctuation des exécutions

- Ne pas tenir compte de trop de décimales!
- Faire attention à ne pas déborder la capacité de mesure!

Méthodologie de mesures

Problème fréquent :

Test en mode exclusif (mono-user).
Outil de mesure à 1ms de précision.

$\Rightarrow$

Fluctuation de 500ms d’une exécution à l’autre !!

Et plus encore avec la montée en fréquence automatique des procs. (effet de “chauffe”)

Démarche conseillée :

- Mesurer les fluctuations, ne pas les ignorer
  (le *warm up* des processeurs peut perturber les premières)
- Ne pas donner que les valeurs moyennes
- Mesurer des temps > 10s si possible

Méthodologie de mesures

Stocker des meta-données sur les conditions de mesure :

- **Date de l’exécution**
- **Auteur(s) du test**
- Outil(s) de mesure utilisé(s)
- Caractéristiques de la machine : RAM, Cache, Processeurs, …
- OS utilisé (nom et version)
- Compilateur utilisé (nom et version)
- Options de compilation utilisées
- Test en multi-user/mono-user ?
- Présence d’IO dans le test ?
- Configuration du programme de test : taille des données, …

On oublie souvent (et rapidement) à quel benchmark se réfère une série de mesures !

On manque souvent de détail sur les conditions de réalisation d’une série de mesures !

Performance, efficiency and scalability metrics