High-Performance Scientific Computing: Introduction to Parallel Programming

Erik Spence

SciNet HPC Consortium

11 March 2014

Why Parallel Programming?

- **Faster**
  There's a limit to how fast one computer can compute.
- **Bigger**
  There's a limit to how much memory, disk, *etc.*, can be put on one computer.
- **More**
  We want to do the same thing that was done on one computer, but *thousands of times*.
- So use more computers!

SciNet compute • calcul CANADA

Why is it necessary?

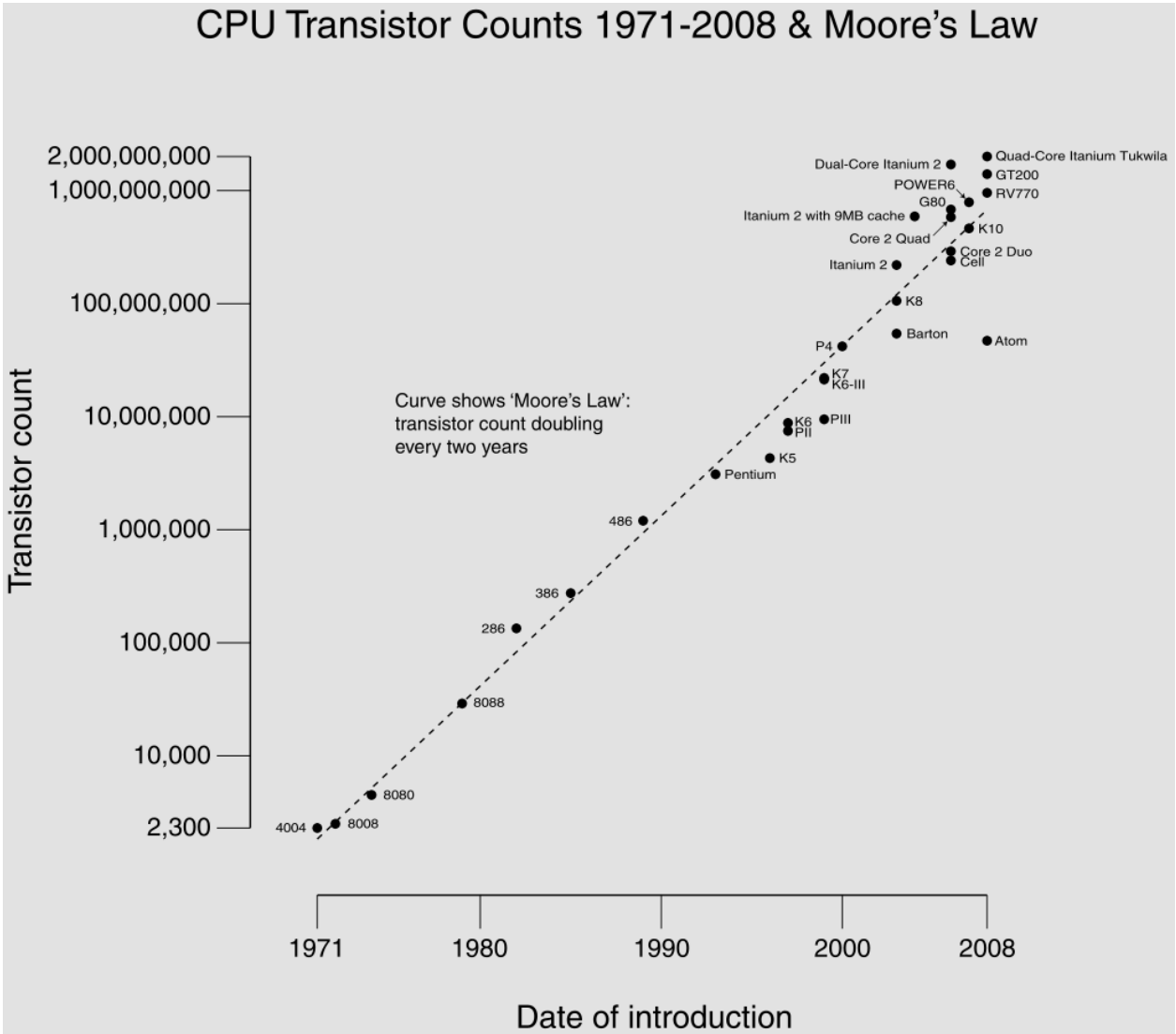
- **Big Data:** Modern experiments and observations yield vastly more data to be processed than in the past.
- **Big Science:** As more computing resources become available (SciNet), the bar for cutting edge simulations is raised.
- **New Science:** which before could not even be done, now becomes reachable.

However:

- Advances in clock speeds, bigger and faster memory and disks have been lagging as compared to ten years ago. We can no longer "just wait a year" and get a better computer.
- So more computing resources here means: more cores running *concurrently*.
- Even most laptops now have 2 or more cpus.
- So parallel computing is necessary.

SciNet compute • calcul CANADA

Wait, what about Moore's Law?



(source: Transistor Count and Moore's Law - 2008.svg, by Wgsimon, wikipedia)

Wait, what about Moore's Law?

Moore's Law:

> *. . . describes a long-term trend in the history of computing hardware. The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years.*
>
> *(source: Moore's law, wikipedia)*

But. . .

- Moore's Law didn't promise us increasing clock speed.
- We've gotten more transistors but it's getting hard to push clock-speed up. Power density is the limiting factor.
- So we've gotten more cores at a fixed clock speed.

SciNet compute • calcul CANADA

Concurrency

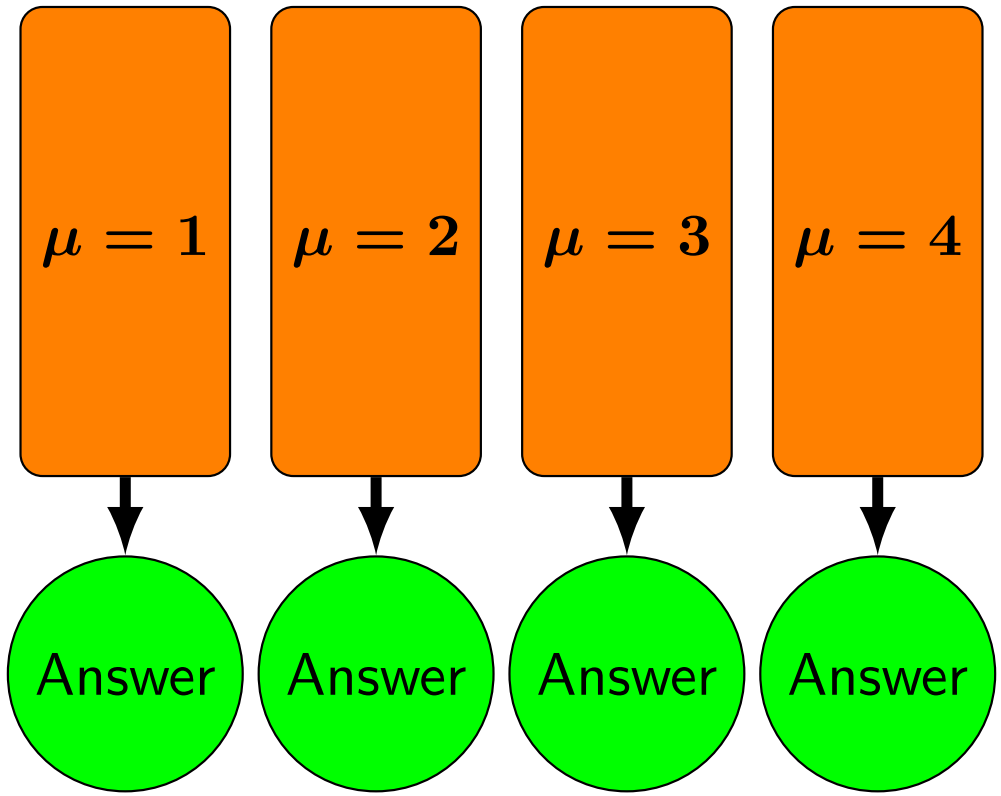
- All these cores need something to do.
- We need to find parts of the program that can done independently, and therefore on different cores concurrently.
- We would like there to be many such parts.
- Ideally, the order of execution should not matter either.
- However, data dependencies limit concurrency.

(source: http://flickr.com/photos/splorp)

SciNet compute • calcul CANADA

Parameter study: best case scenario

- Suppose the aim is to get results from a model as a parameter varies.
- We can run the serial program on each processor at the same time.
- Thus we get 'more' done.



SciNet compute • calcul CANADA

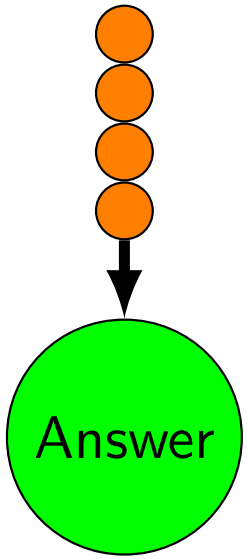
Throughput

- How many tasks can you do per unit time?

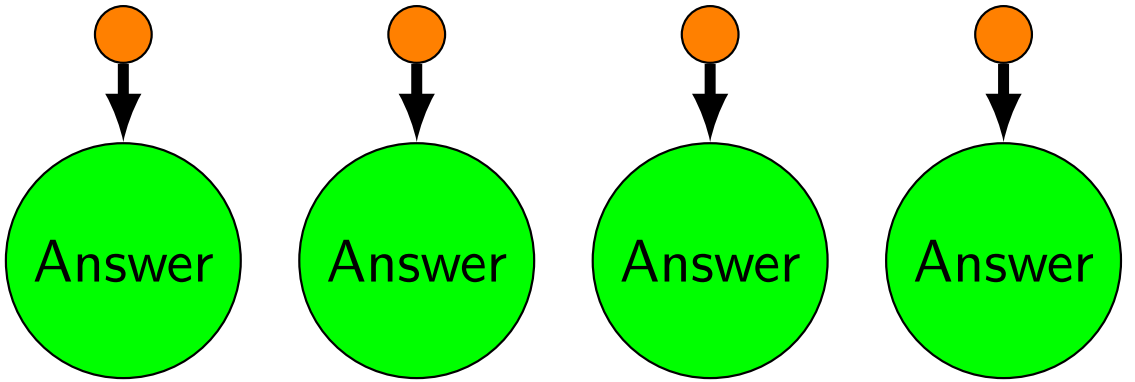
$$\textbf{throughput} = H = \frac{N}{T}$$

  N is the number of tasks, T is the total time.

- Maximizing H means that you can do as much as possible.
- Independent tasks: using P processors increases H by a factor of P.



$$T = NT_1$$
$$H = 1/T_1$$



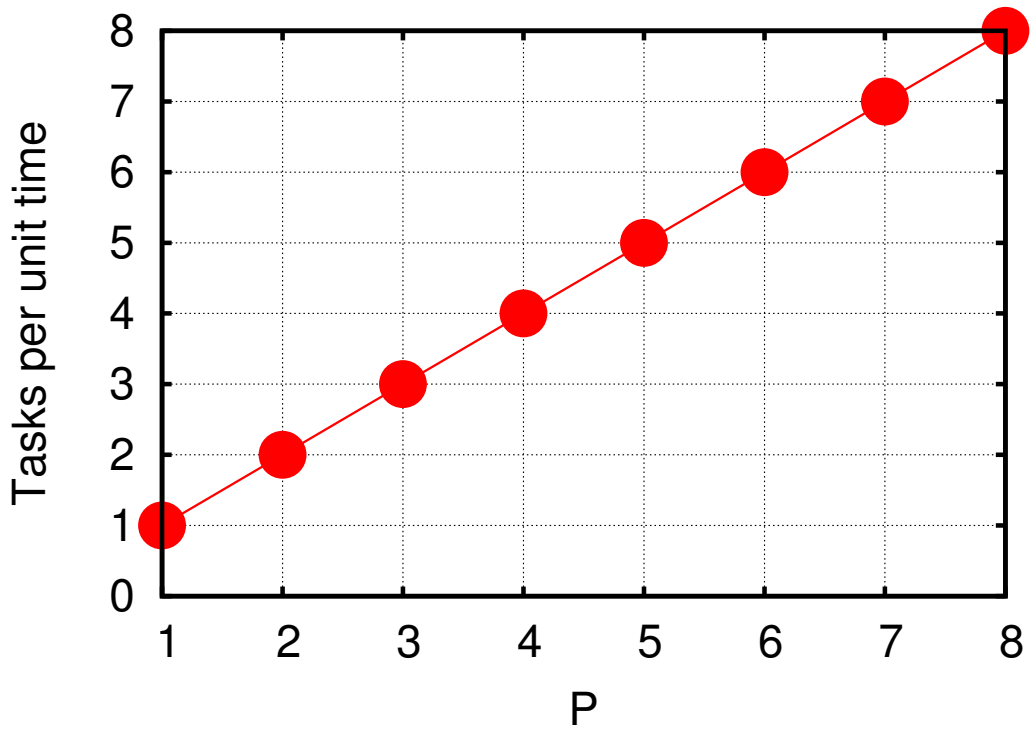
$$T = NT_1/P$$
$$H = P/T_1$$

Scaling — Throughput

- How a given problem's throughput scales as processor number increases is called "strong scaling".
- In this case, linear scaling:

$$H \propto P$$

- This is perfect scaling.

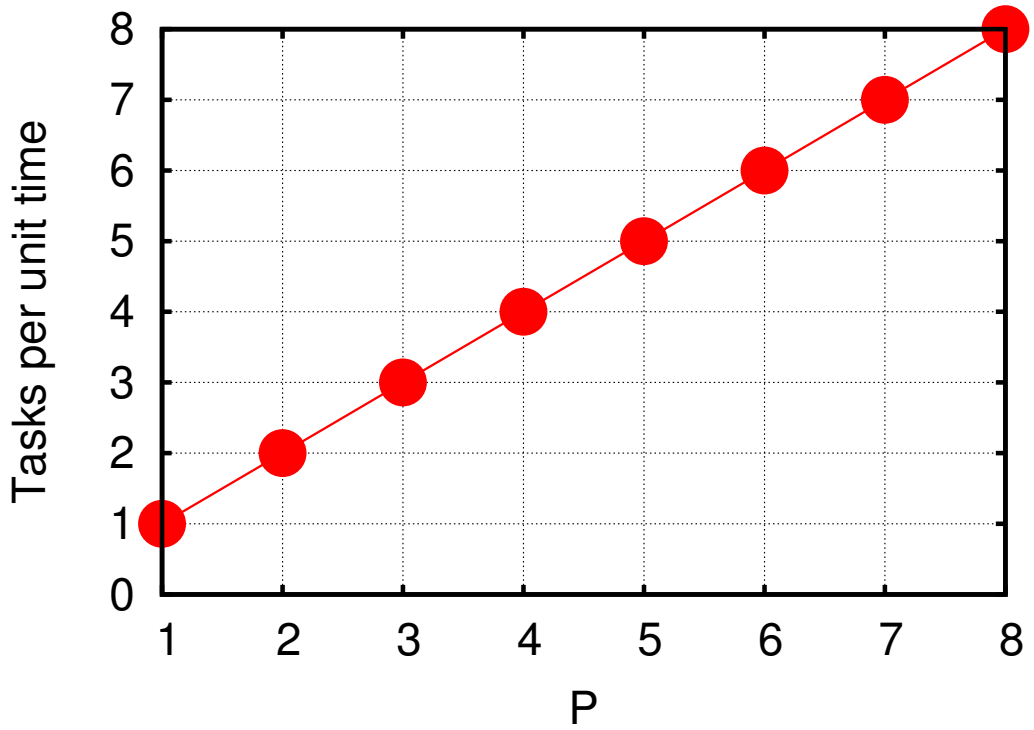


Scaling — Speedup

- Speedup: how much faster the problem is solved as processor number increases.
- This is measured by the serial time divided by the parallel time

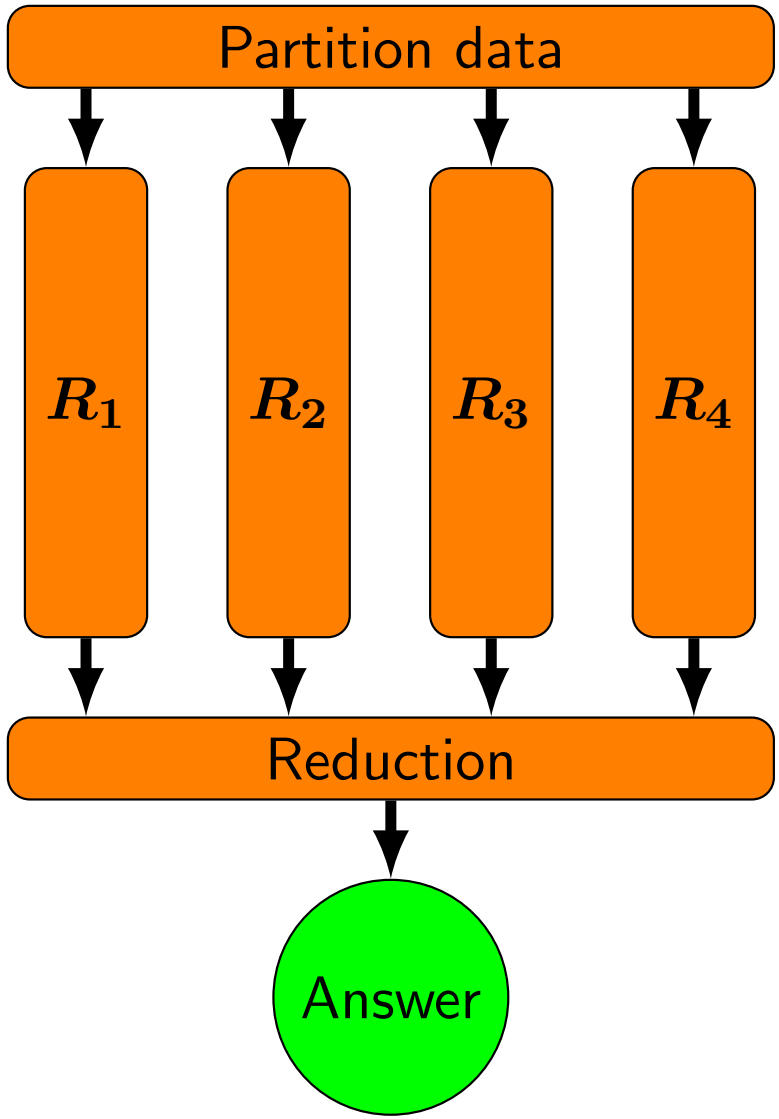
$$S = \frac{T_{\text{serial}}}{T(P)}$$

- For embarrassingly parallel applications, $S \propto P$: Linear speed up.

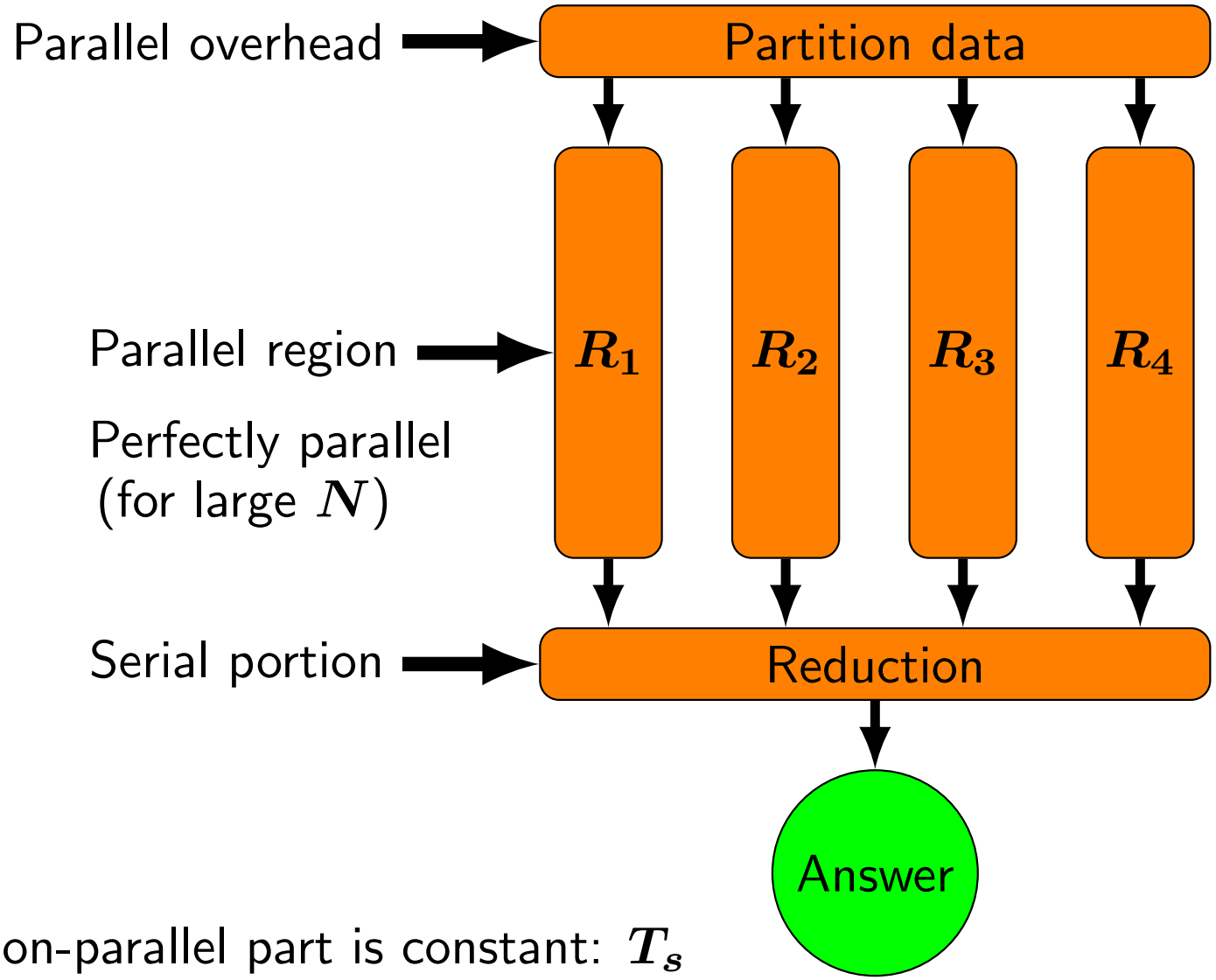


Non-ideal cases

- Say we want to integrate some tabulated experimental data.
- Integration can be split up, so different regions are summed by each processor.
- Non-ideal:
  - We first need to get data to each processor.
  - At the end we need to bring together all the sums: 'reduction'.



Non-ideal cases



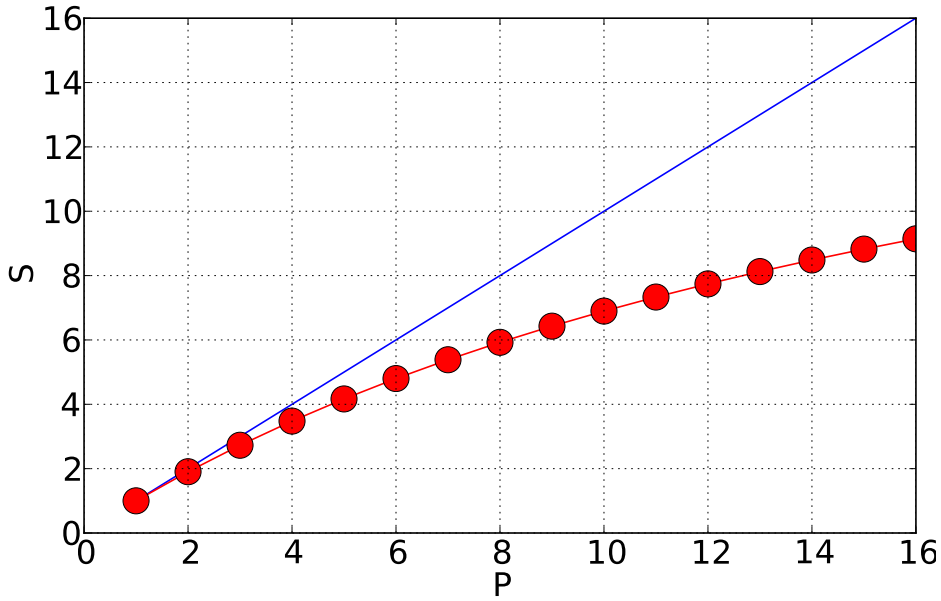
Suppose non-parallel part is constant: T_s

Amdahl's law

Speed-up (without parallel overhead): $S = \dfrac{T_{\text{serial}}}{T(P)} = \dfrac{NT_1 + T_s}{\frac{NT_1}{P} + T_s}$

or, calling $f = T_s/(T_s + NT_1)$ the serial fraction,

$$S = \frac{1}{f + (1-f)/P} \xrightarrow{P\to\infty} \frac{1}{f}$$



The serial part dominates asymptotically. The speed-up is limited, no matter what size of P. $f = 5\%$ above.

Scaling efficiency

Speed-up compared to ideal factor P:

$$\textbf{Efficiency} = \frac{S}{P}$$

This will invariably fall off for larger P, except for embarrassingly parallel problems.

$$\textbf{Efficiency} \sim \frac{1}{fP} \overset{P\to\infty}{\longrightarrow} 0$$

You cannot get 100% efficiency in any non-trivial problem.

All you can aim for here is to make the efficiency as least low as possible.

SciNet compute • calcul CANADA

Less-ideal case of Amdahl's law

We assumed that the non-parallel part is constant. But it will in fact increase with P, from the sum of the results of all the processors

$$T_s \approx PT_1$$

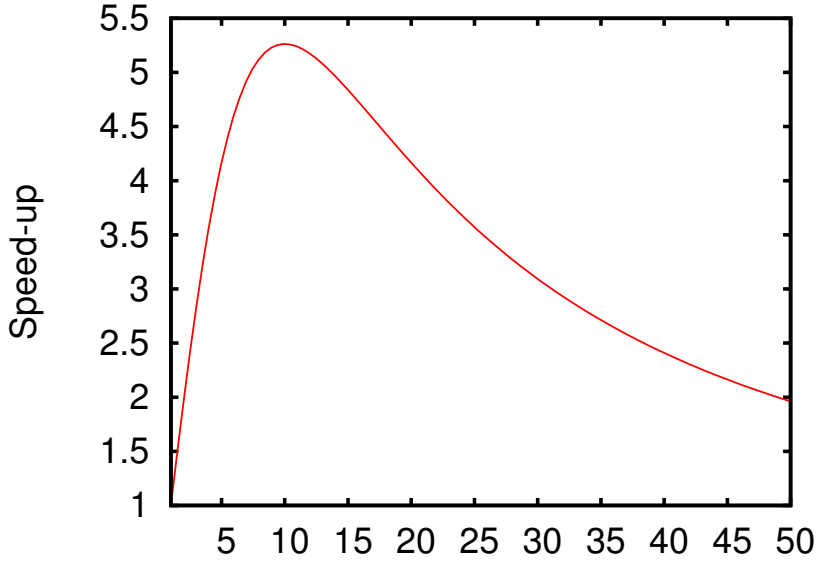
Serial fraction is now a function of P:

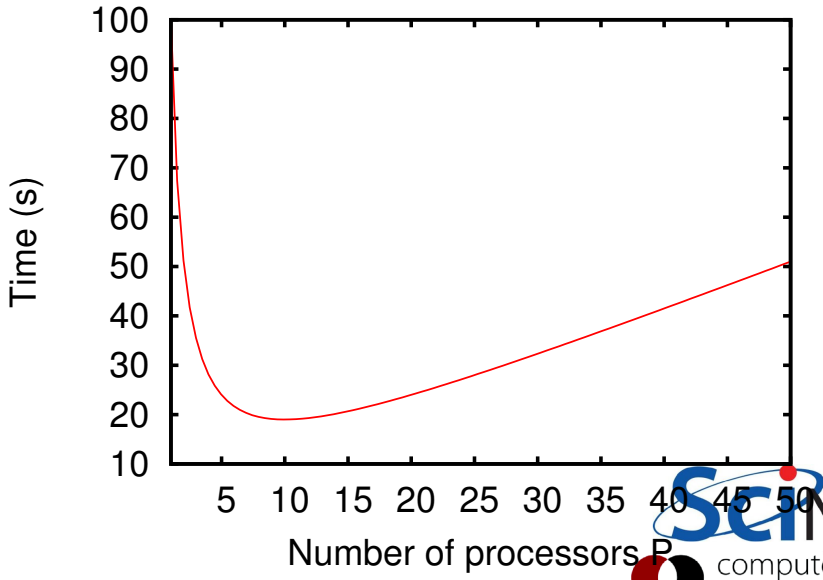
$$f(P) \sim \frac{P}{N}$$

Amdahl:

$$S(P) = \frac{1}{f(P) + [1 - f(P)]/P}$$

Example: $N = 100$, $T_1 = 1s \ldots$





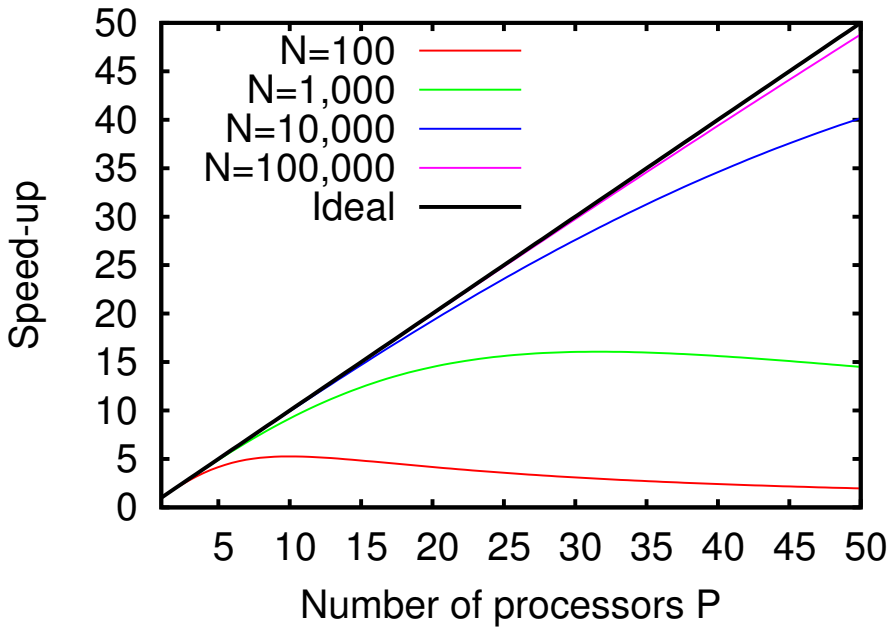
Trying to beat Amdahl's law I

Scale up!

The larger N, the smaller the serial fraction:

$$f(P) = \frac{P}{N}$$

$$S = \frac{1}{f + (1-f)/P}$$



Weak scaling: increase the problem size while increasing P:

$$\mathbf{Time_{weak}}(P) = \mathbf{Time}(N = n \times P, P)$$

Good weak scaling means the time approaches a constant for large P.

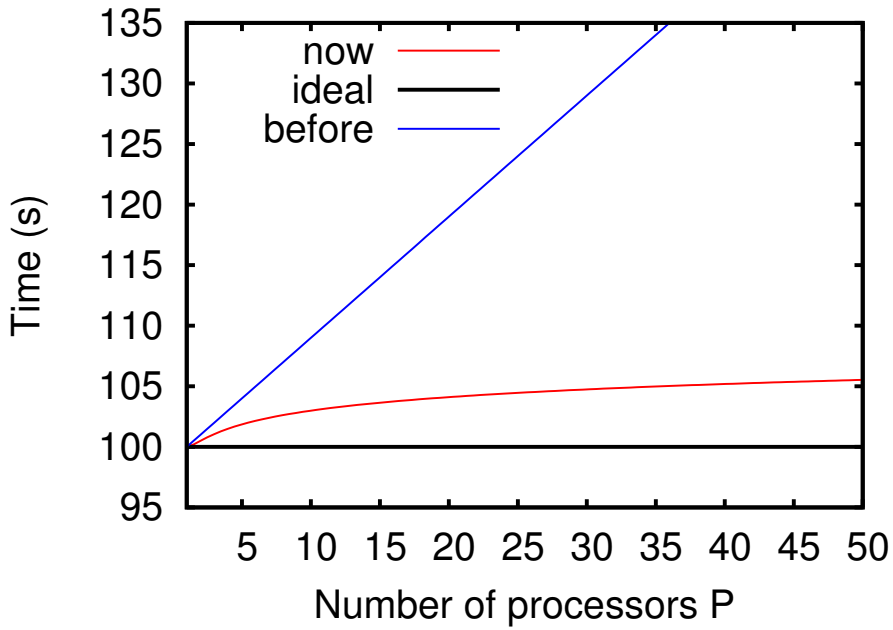
Trying to beat Amdahl's law I, continued

Weak scaling

$$\mathbf{T_{weak}}(P) = \mathbf{T}(N = n \times P, P)$$

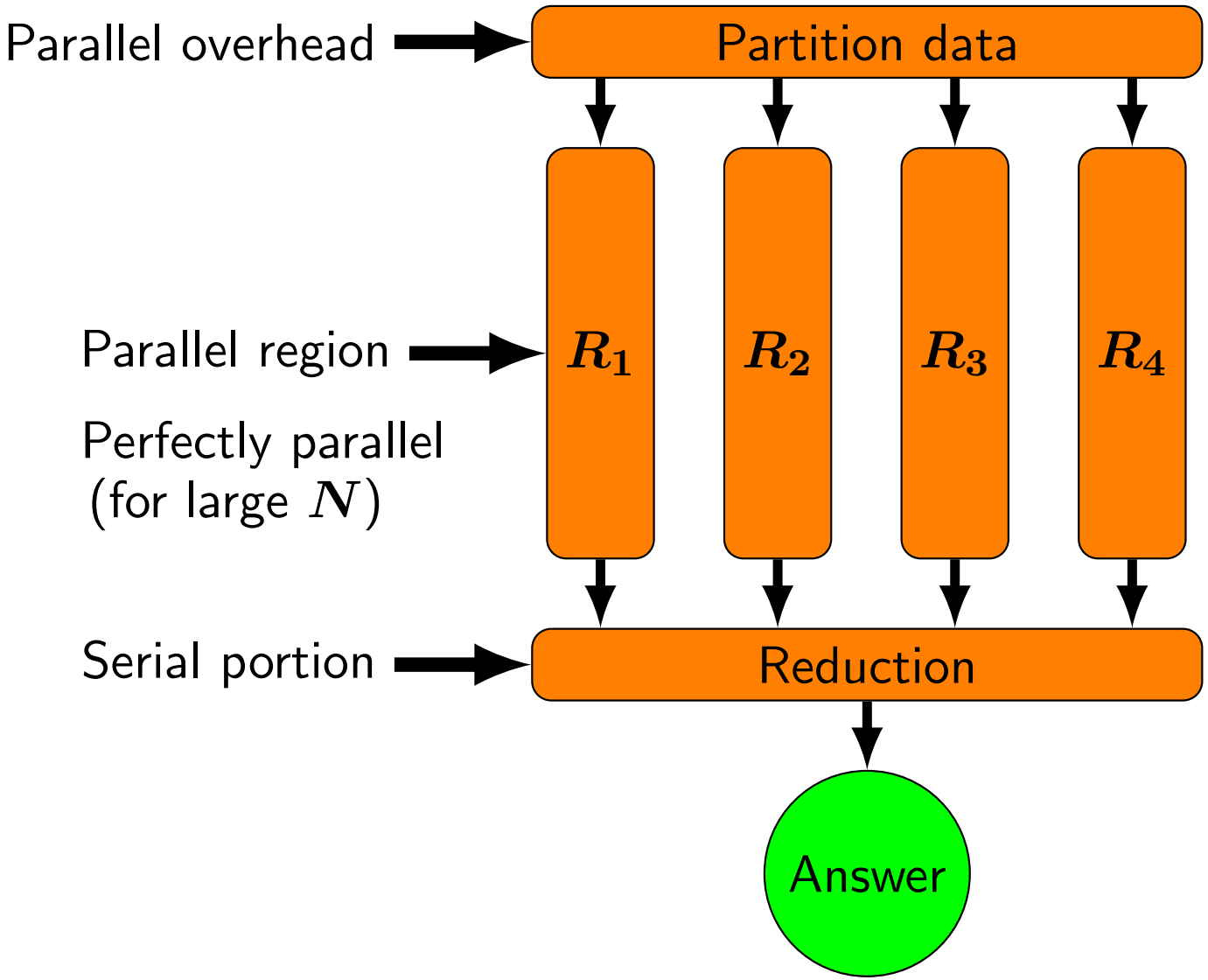
In theory we should approach a constant for large P.

Not quite, but a significant improvement over before.

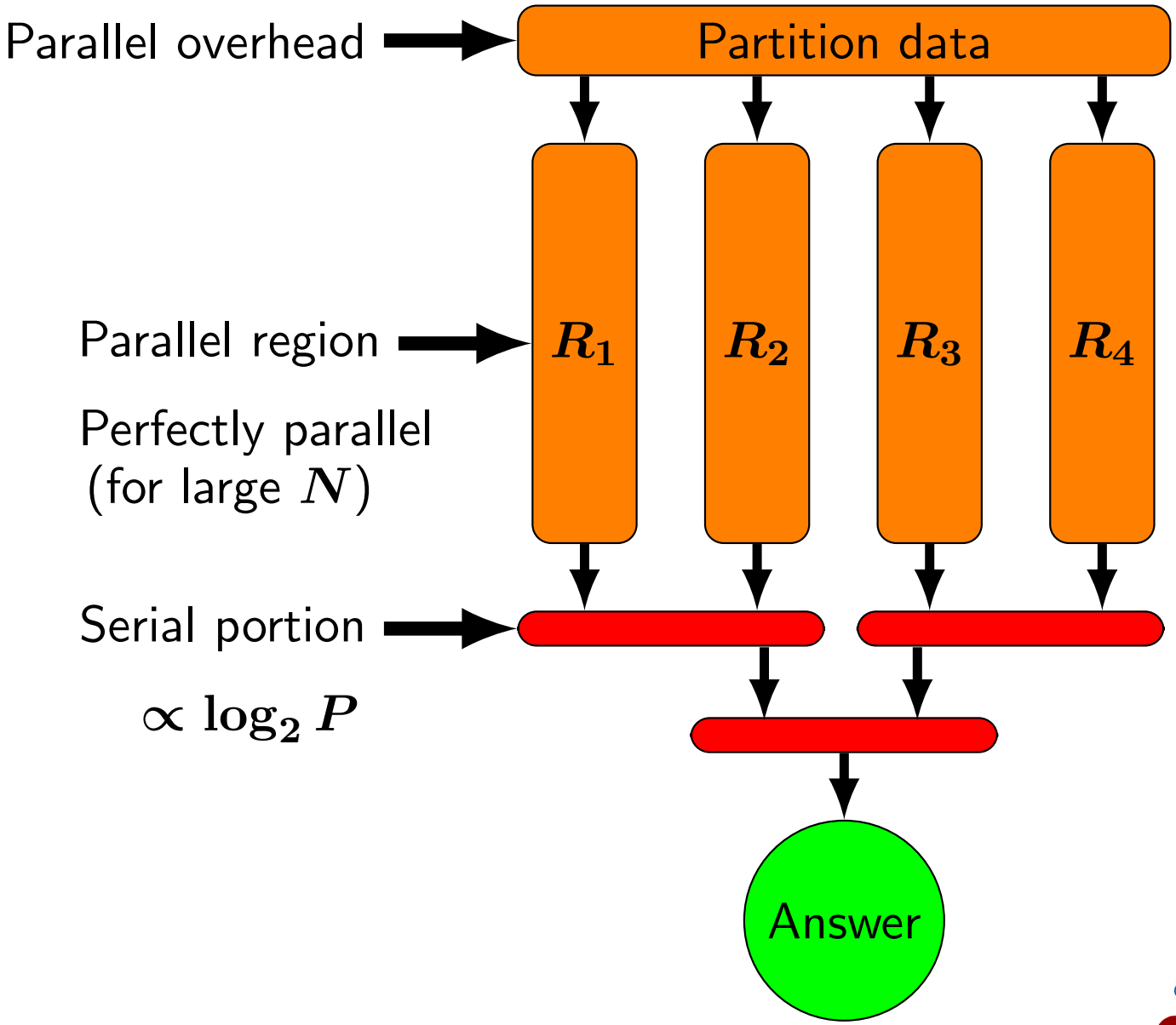


Really not that bad, and other other algorithms can do better.

Trying to beat Amdahl's law II



Trying to beat Amdahl's law II



Trying to beat Amdahl's law II, continued

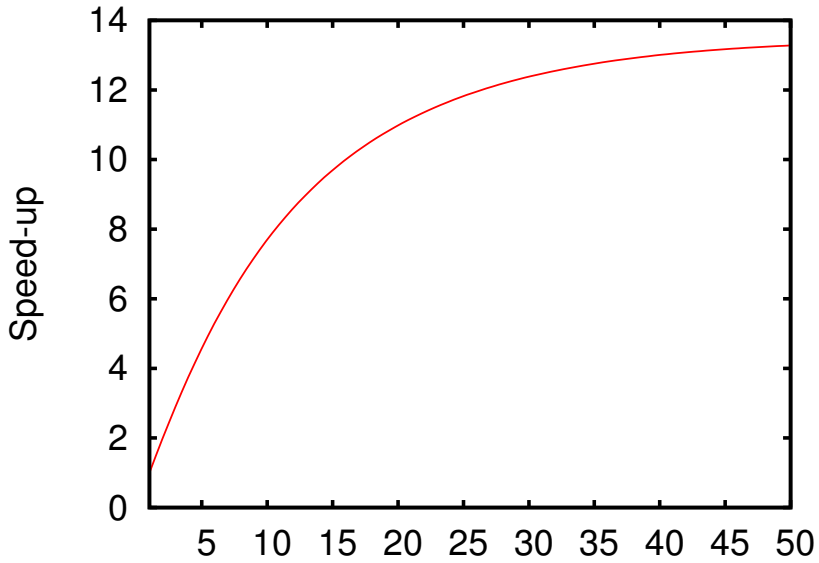
Serial fraction is now a different function of P:

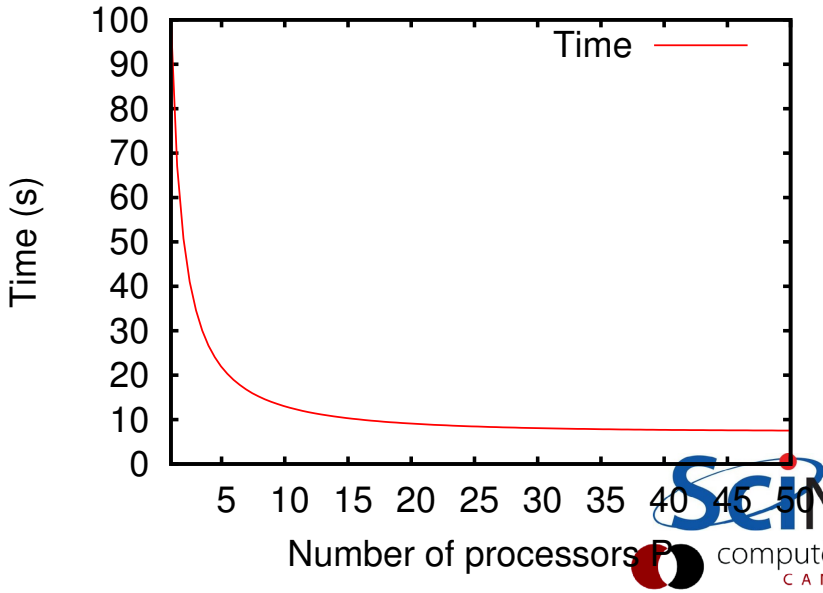
$$f(P) = \frac{\log_2 P}{N}$$

Amdahl:

$$S(P) = \frac{1}{f(P) + [1 - f(P)]/P}$$

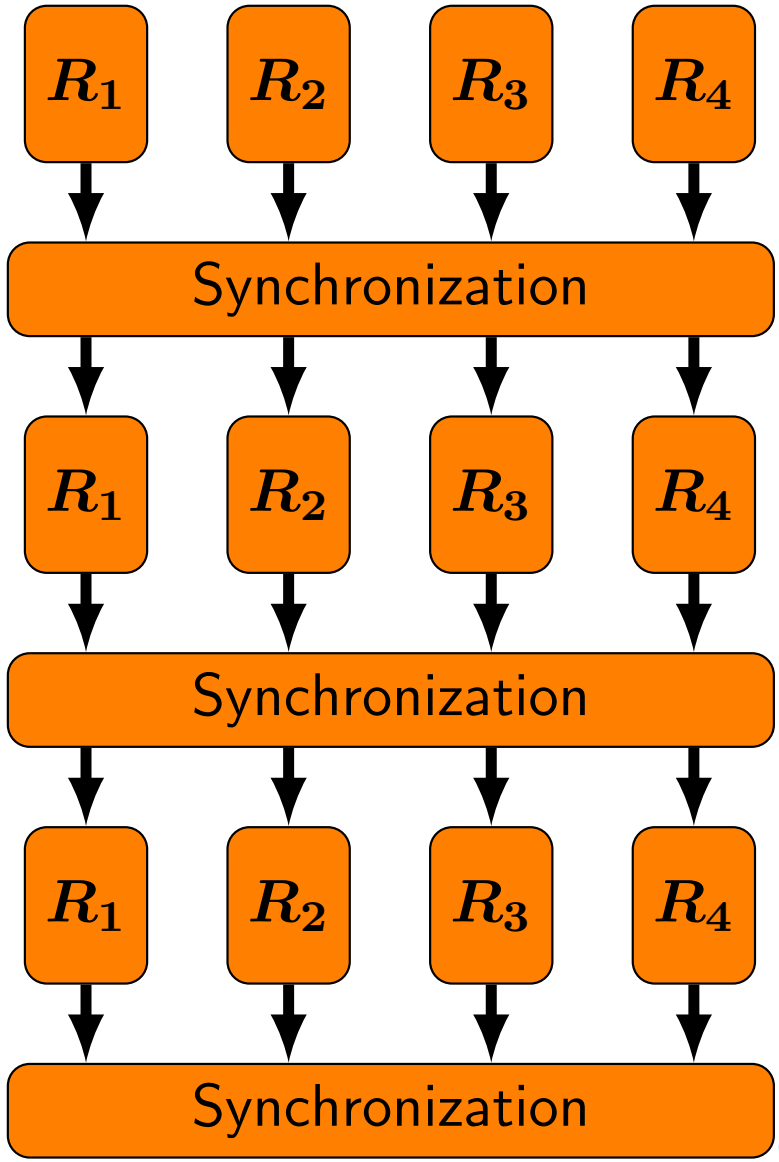
Example: $N = 100$, $T_1 = 1s \ldots$





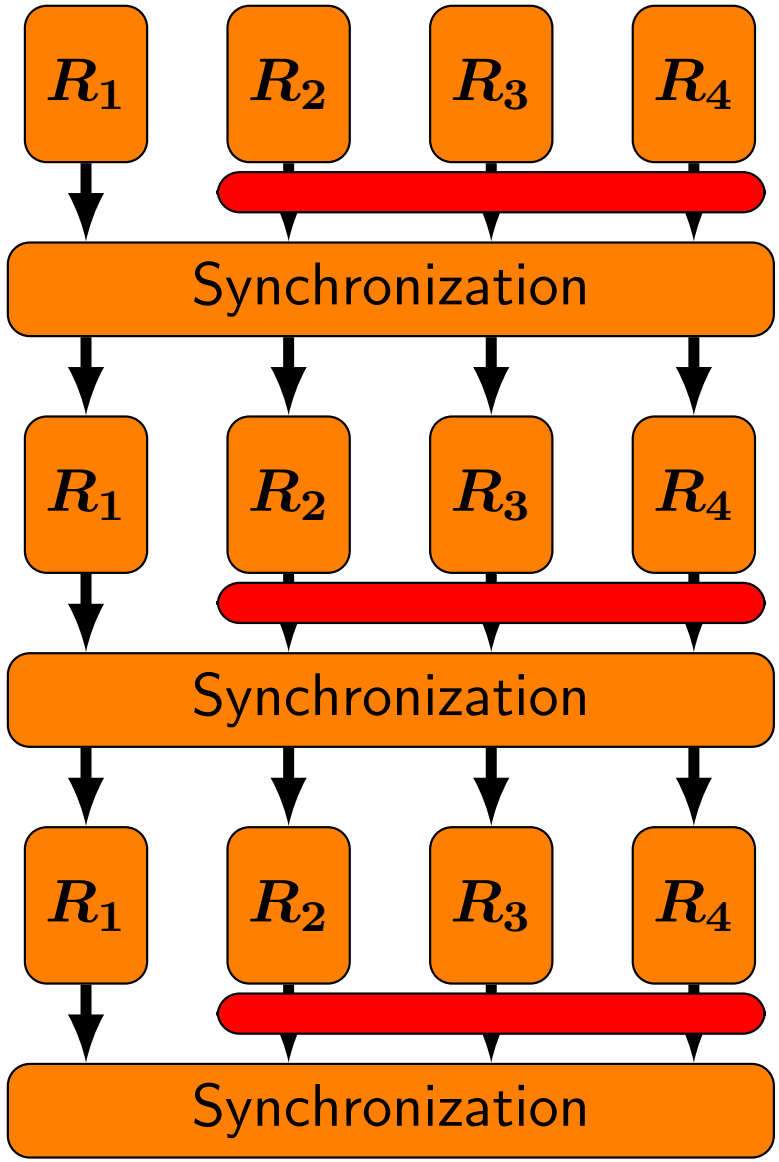
Synchronization

- Most problems are not purely concurrent.
- Some level of synchronization or exchange of information is needed between tasks.
- While synchronizing, nothing else happens: increases Amdahl's f.
- And the synchronizations themselves are costly.



Load balancing

- The division of calculations among the processors may not be equal.
- Some processors could already be done, while others are still going.
- Effectively using fewer than P processors: reduced efficiency.
- The aim is for load-balanced algorithms.

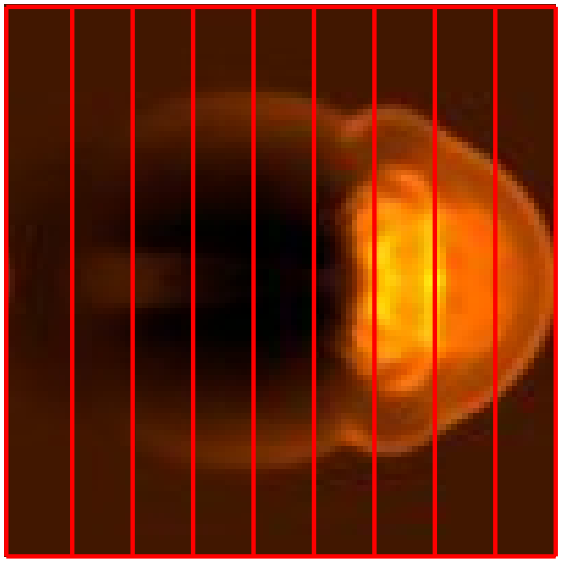


SciNet compute • calcul CANADA

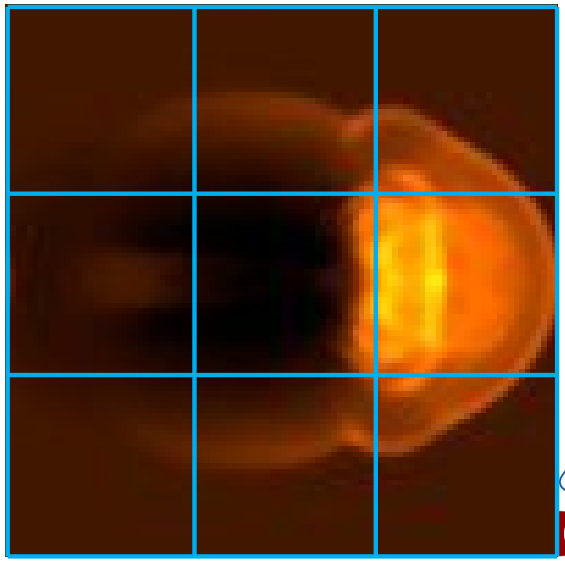
Locality

- So far we have neglected communication costs.
- But communication costs are more expensive than computation!
- To minimize communication-to-computation ratio:
  * Keep the data where it is needed.
  * Make sure as little data as possible is to be communicated.
  * Make shared data as local to the correct processors as possible.
- Local data means lower need for syncs, or smaller-scale syncs.
- Local syncs can alleviate load balancing issues.

incorrect



correct



Take home message

The big lesson here: Parallel algorithm design is about finding as much concurrency as possible, and arranging it in a way that maximizes locality.