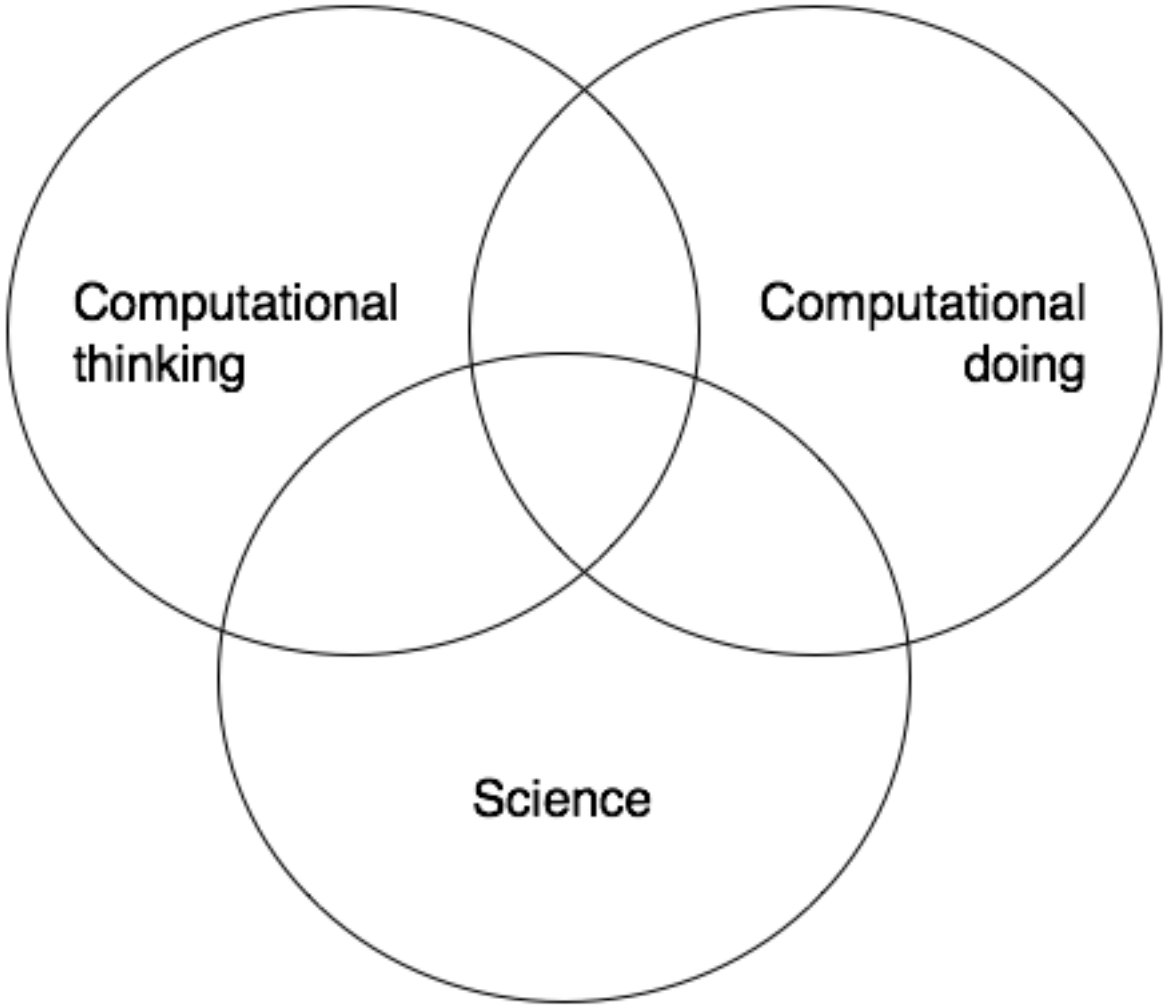
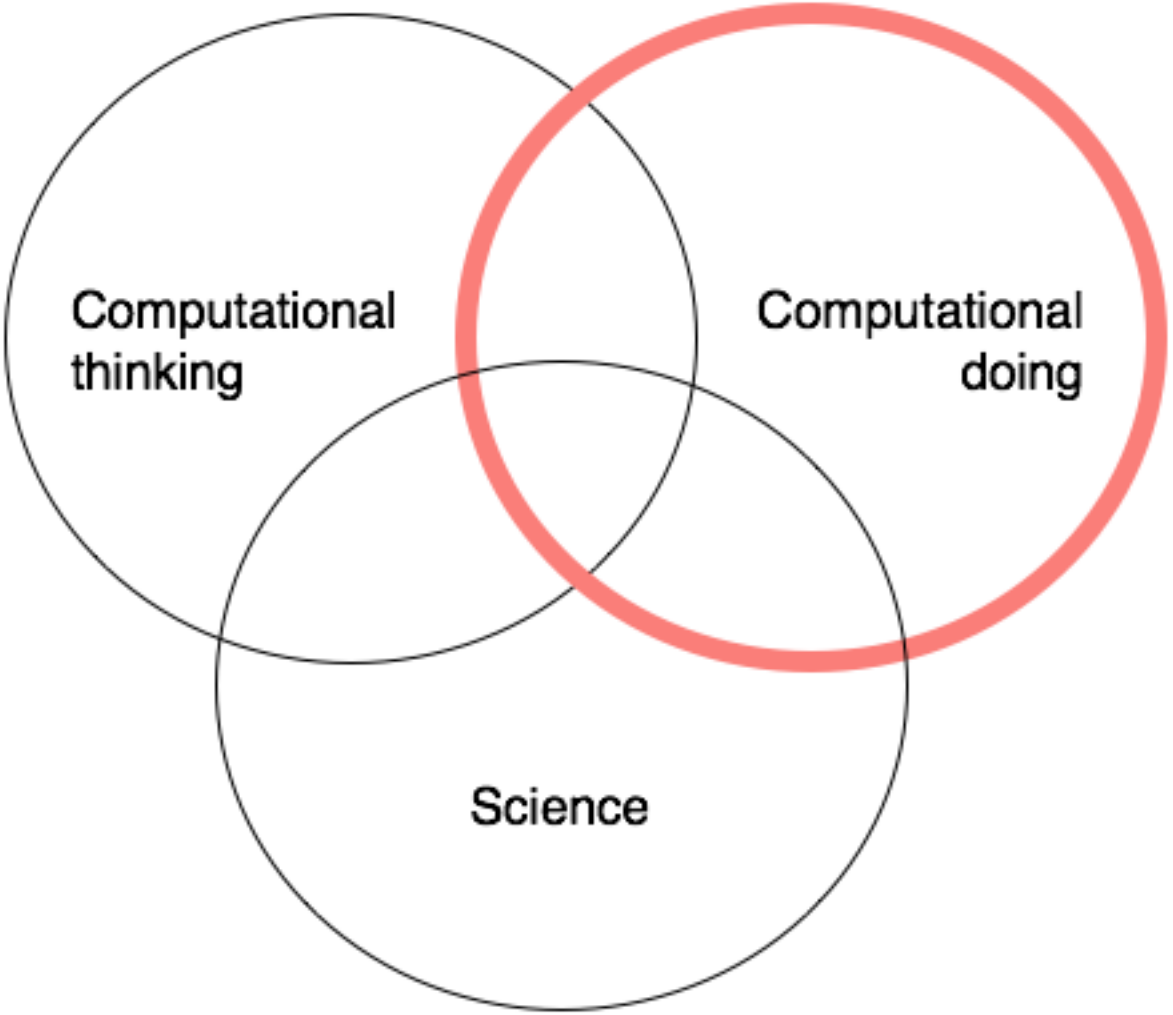
Hardware, software, and thinking.

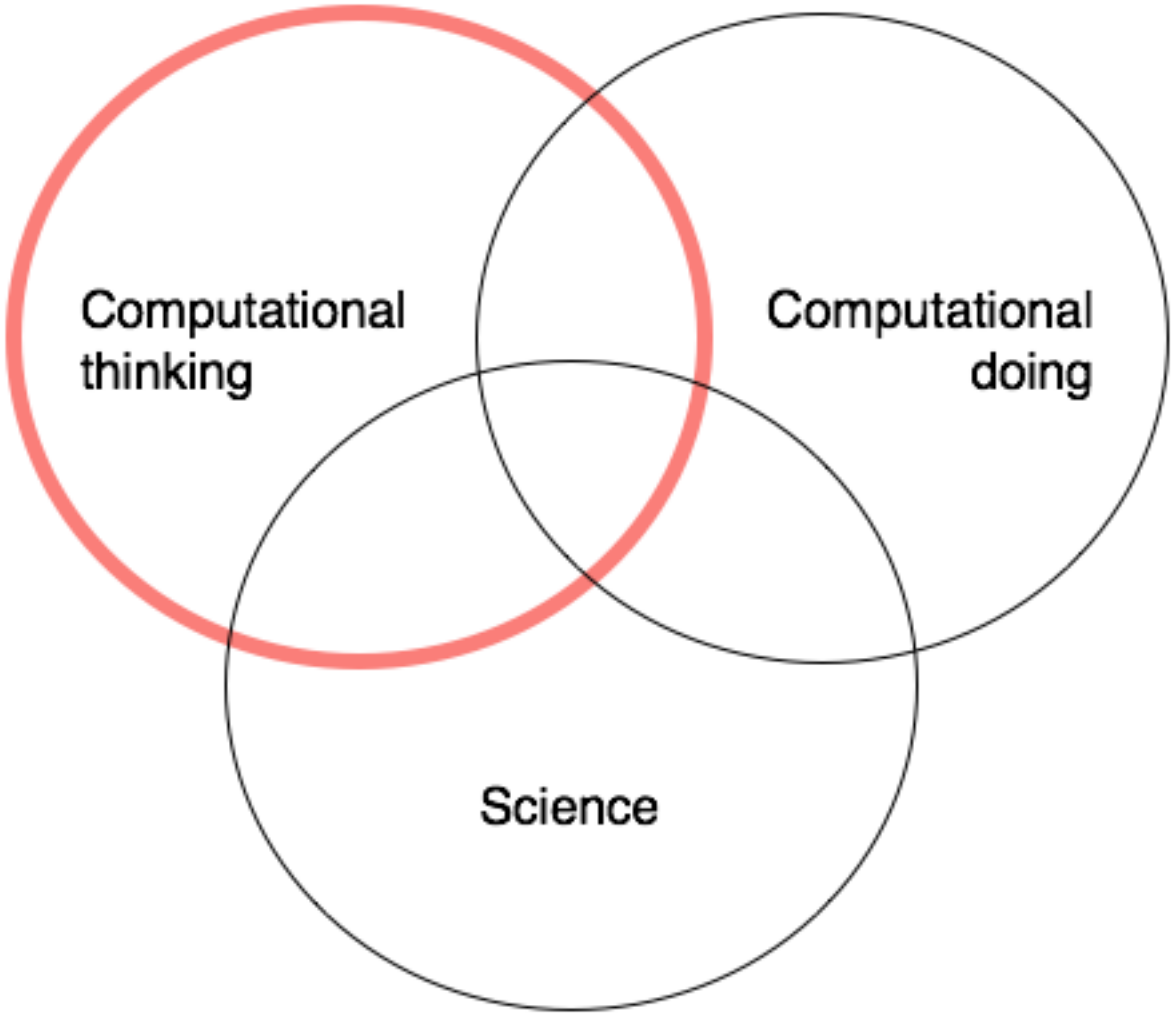
Computational science



Computational science



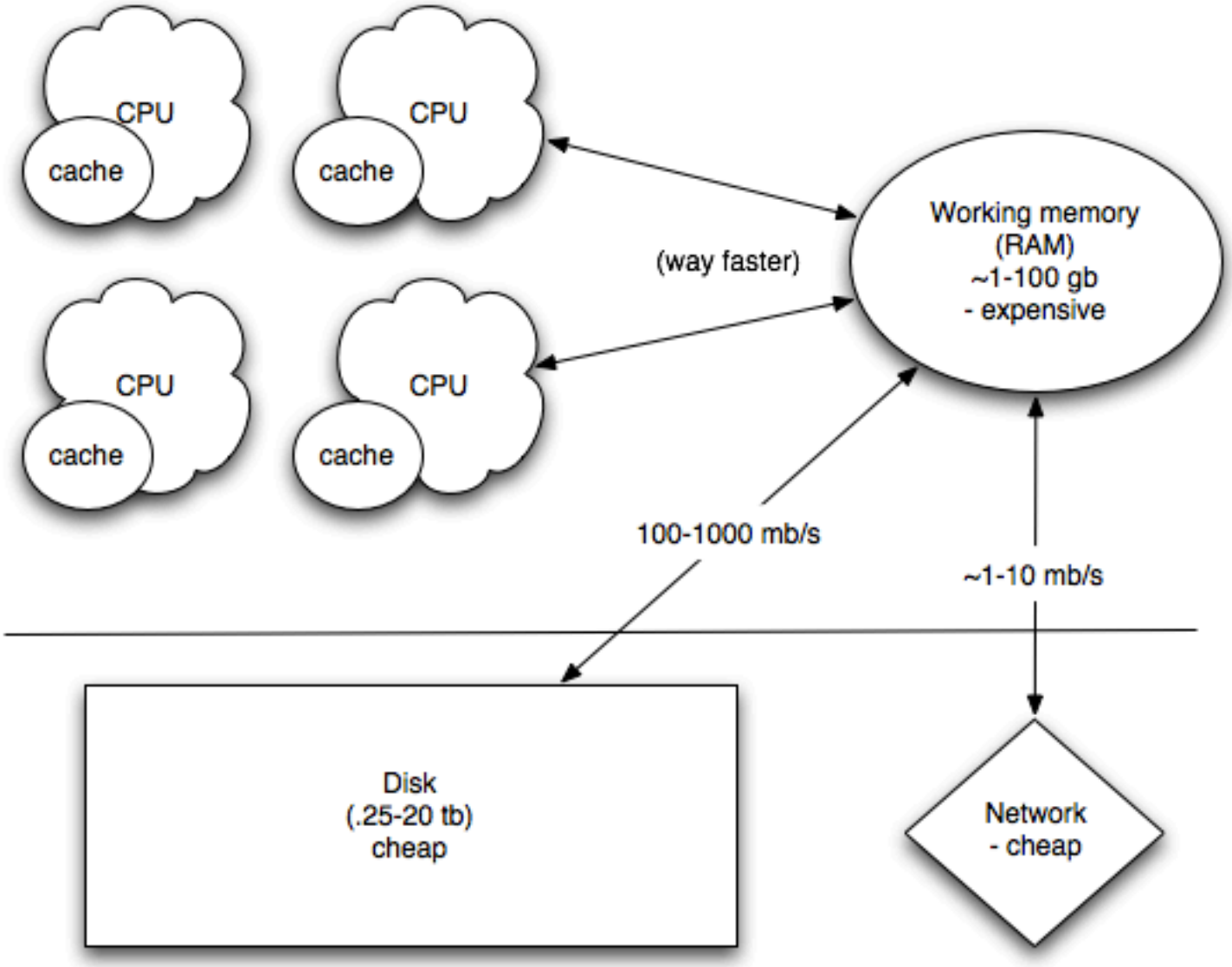
Computational science



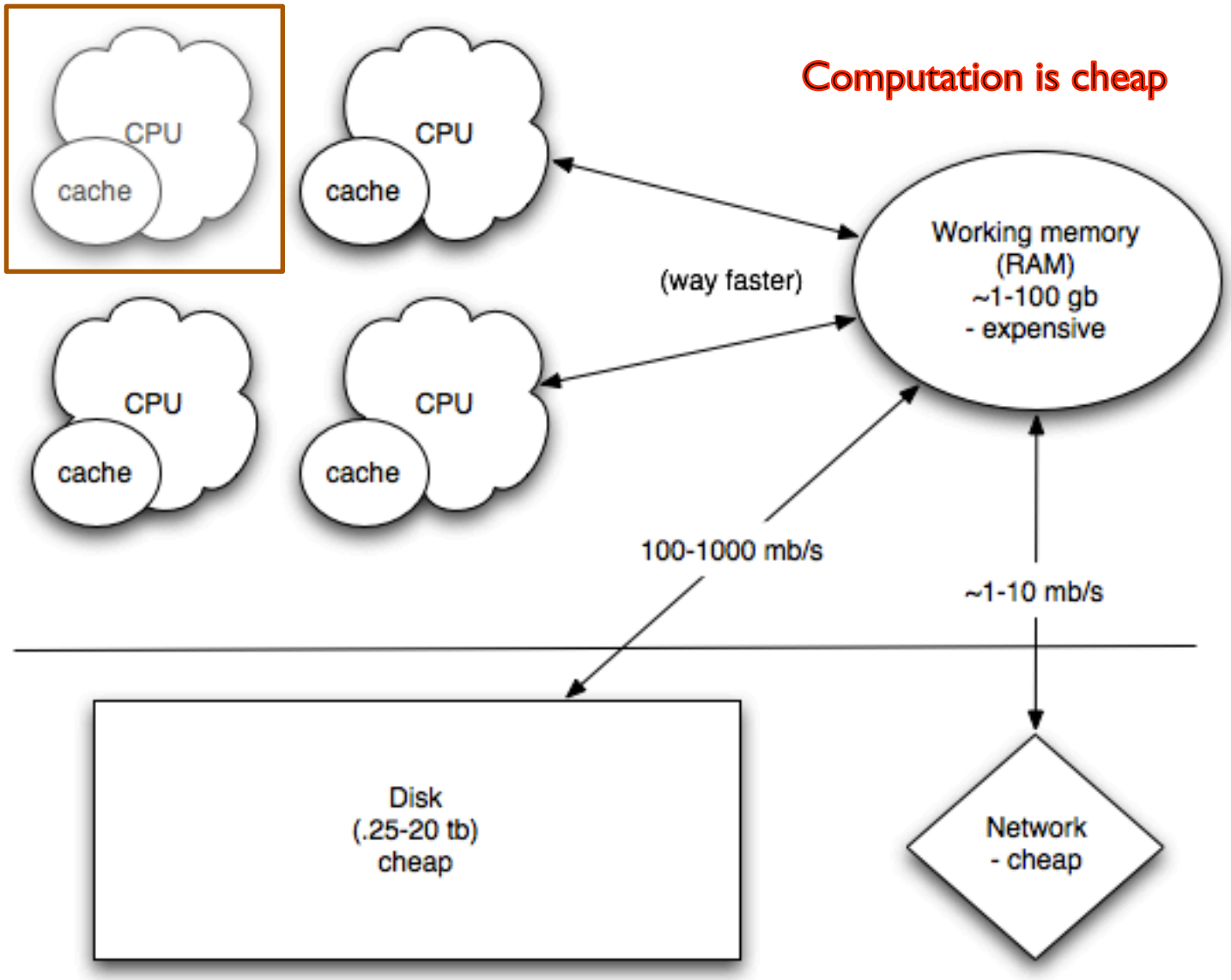
Outline

- Some basic thoughts on hardware
- Implementing your software
- Mowing your lawn
- Outsmarting the evil dean
  - Heuristics in computation
- Algorithms and scaling
  - Two ways to calculate prime numbers
- Breaking down algorithms
  - Sorting
- Software reliability

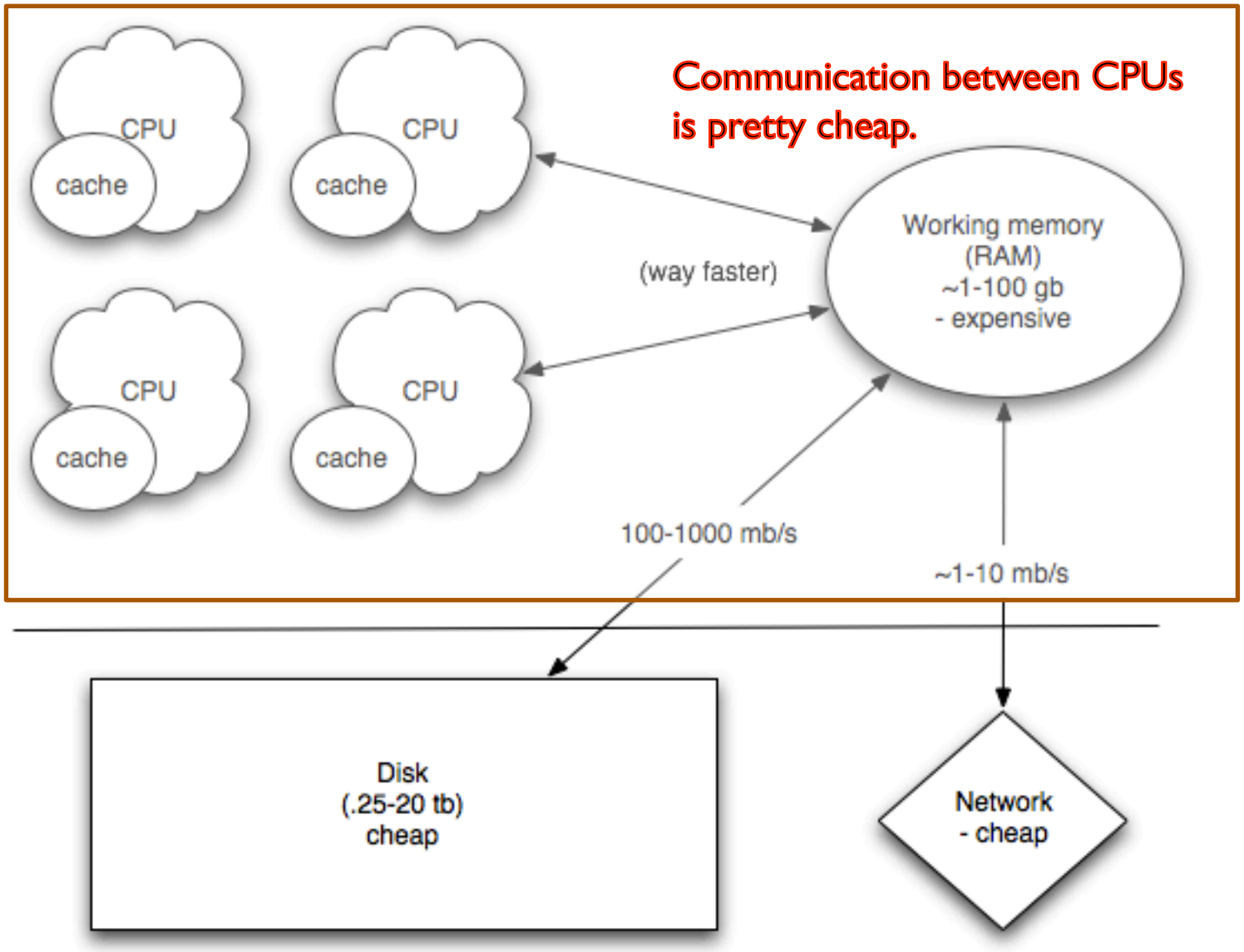
Computer architecture



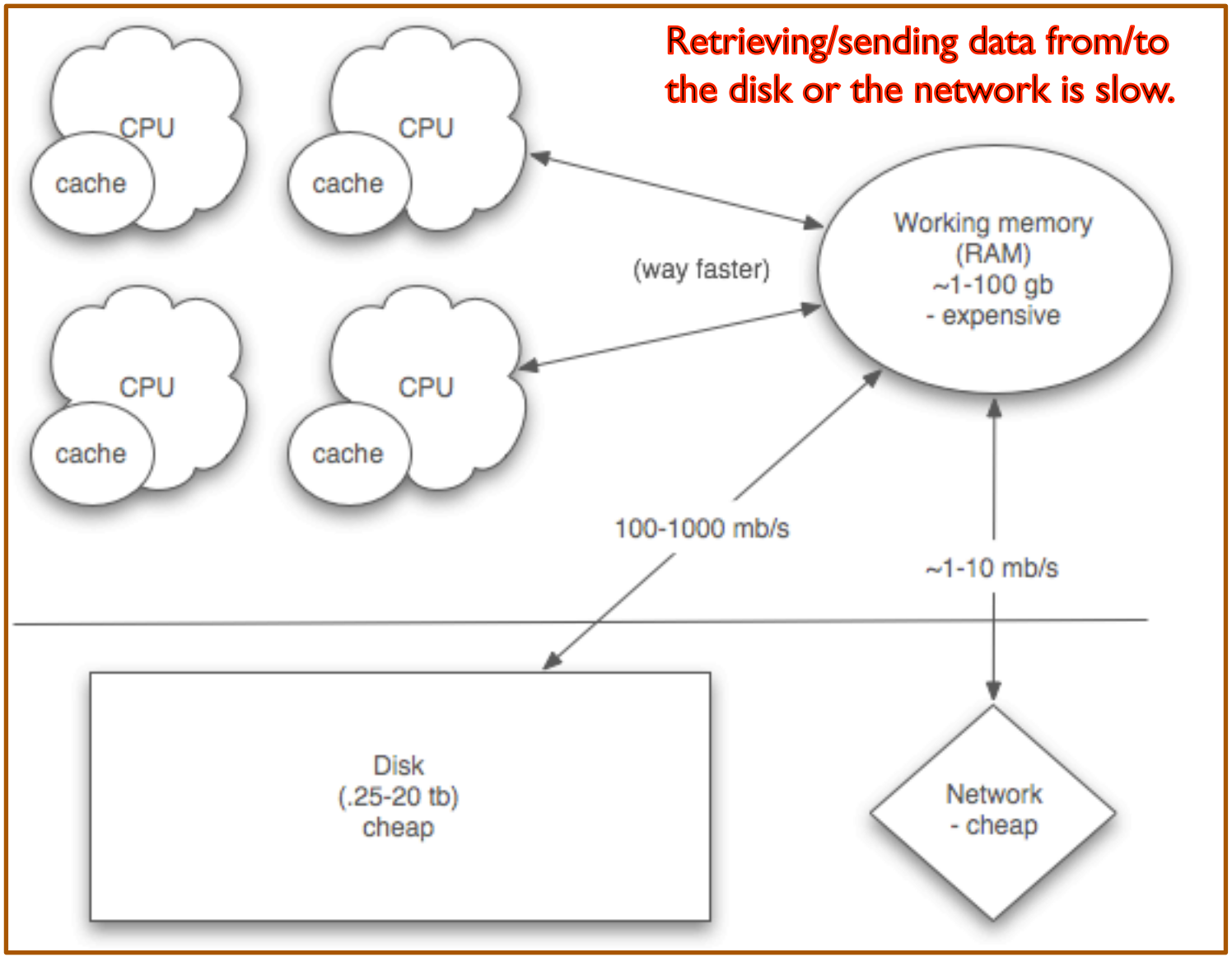
Computer architecture



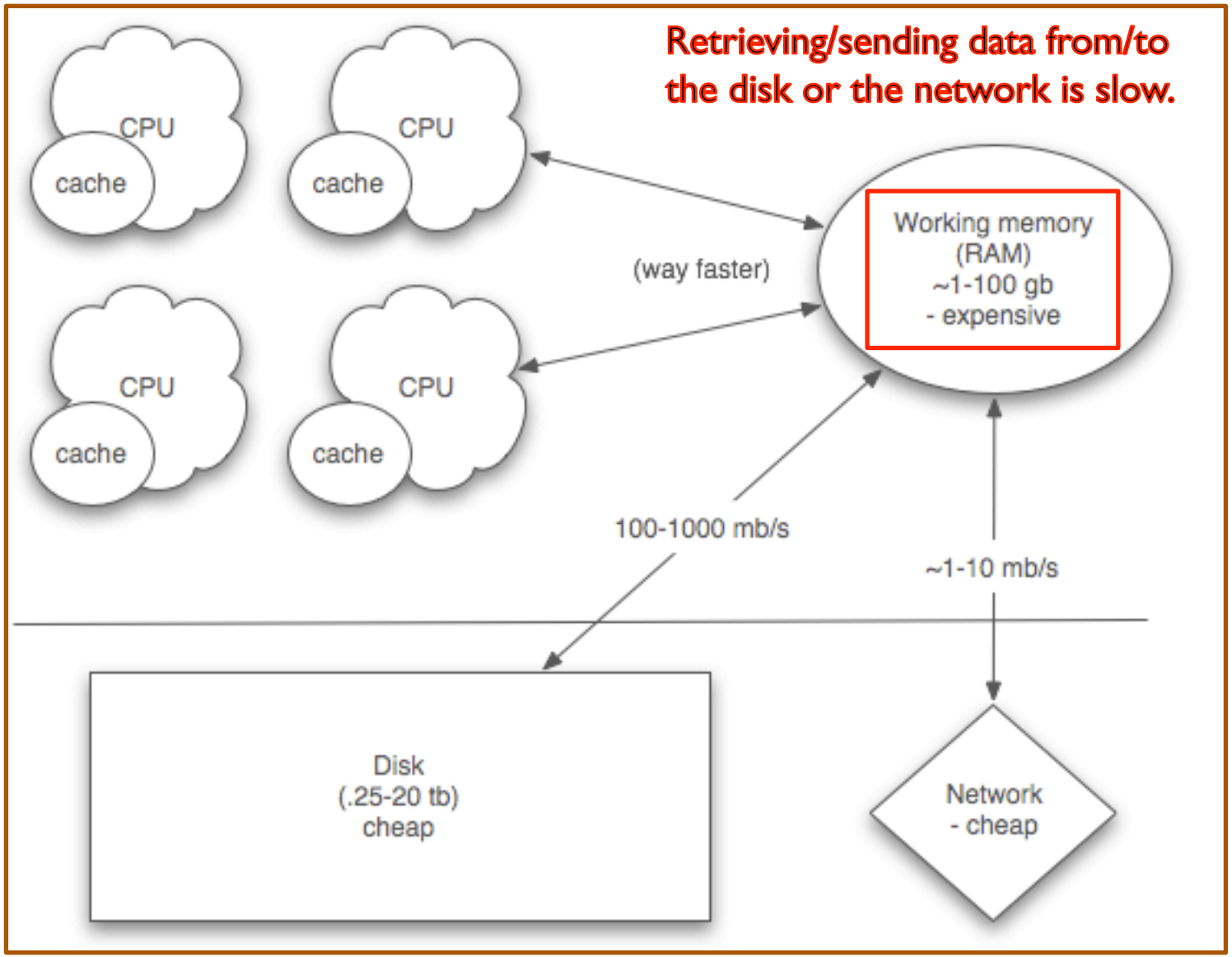
Computer architecture



Computer architecture



Computer architecture



Questions to ask

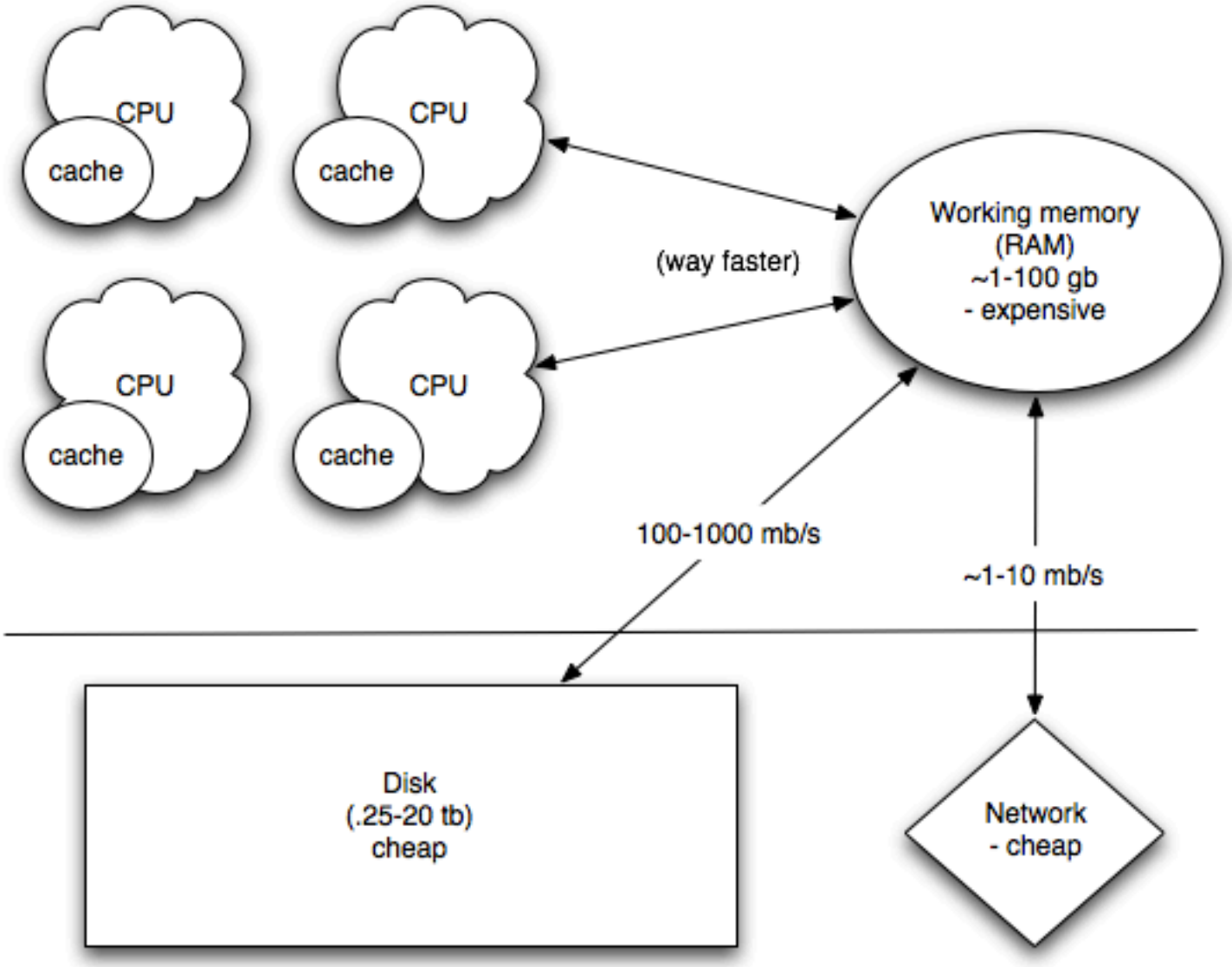
- Can I split my problem up into small chunks?

  (because, if so, I can use multiple, small computers effectively. Small computers are cheap!)

- How does my computation scale?

- How does my memory use scale?

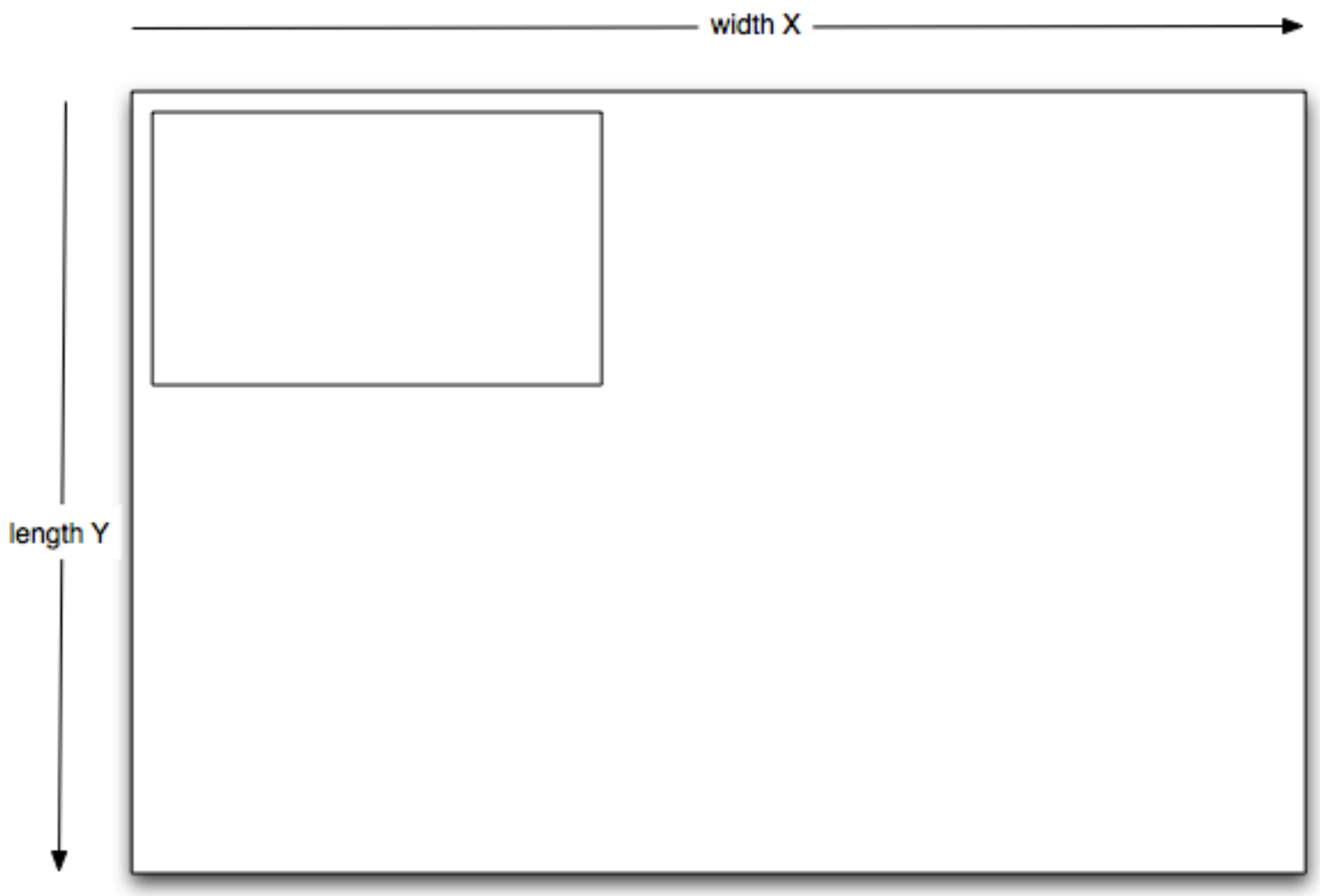
Communication between CPUs/computers slows you down; this is main factor in splitting up tasks.



More important than raw computer speed…

- How does your approach *scale* with the problem size?
- If your approach scales well, then bigger problems can be tackled by throwing more computation at it, or by "micro optimizing".
- If not, well… you may need to think more. Or it may be *impossible* to scale.

Example: mowing the lawn



Mowing the lawn scales… how?

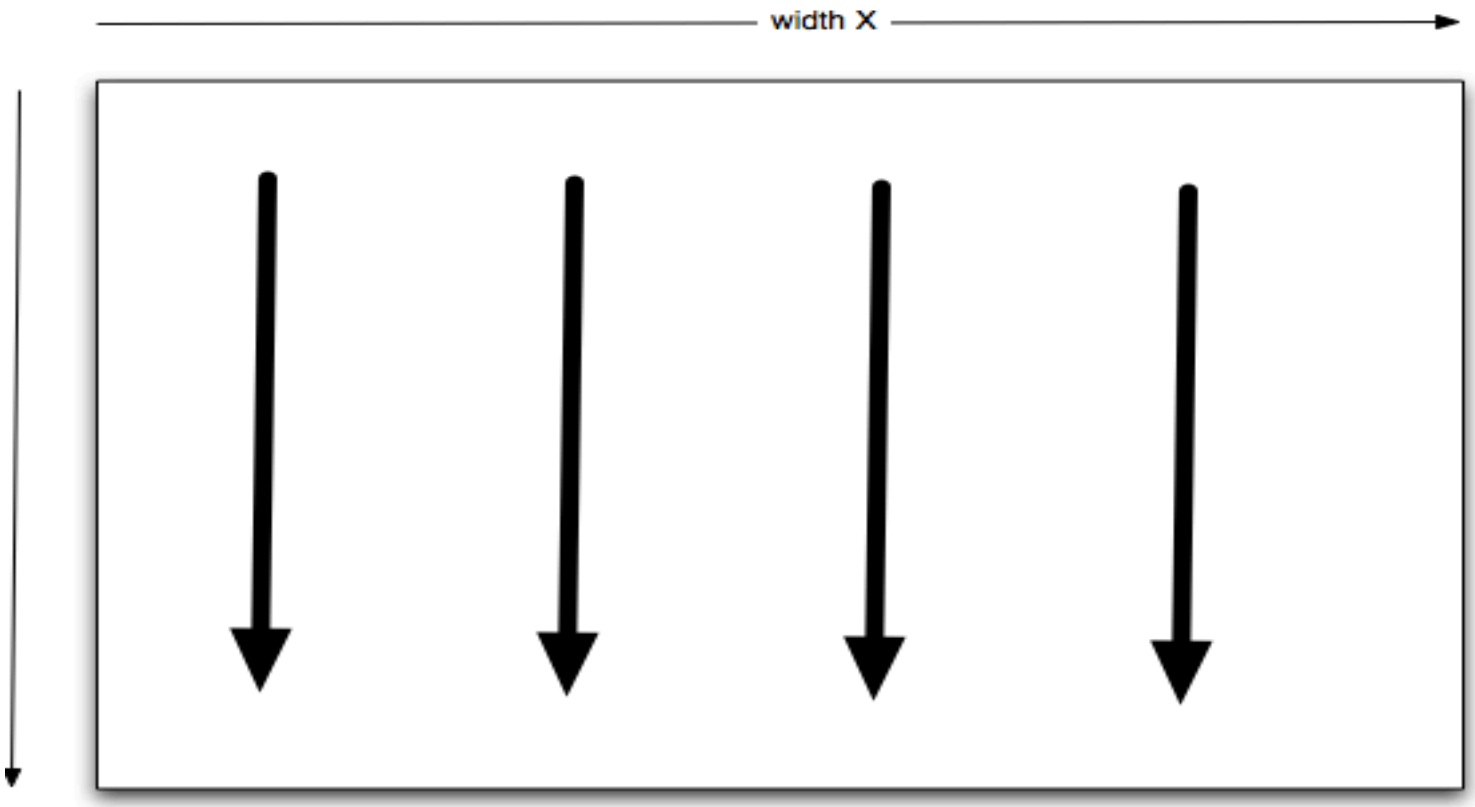
- The time it takes to mow the lawn is related to the *area* of the lawn, X * Y.
- We say O(mowing) = X*Y, therefore.
- Note: this is independent of the speed of your mower, the width of your mower blades, the height of the grass, etc. Those are all *details* that tell you how fast you can *actually* mow a lawn. Unrelated to scaling.

Can you parallelize lawn mowing?

- If you have four times as many lawn mowers, can you mow the lawn five times as fast?
- **YES.**
- But why??

Can you parallelize lawn mowing?

- Yes, because you can subdivide the task in advance.

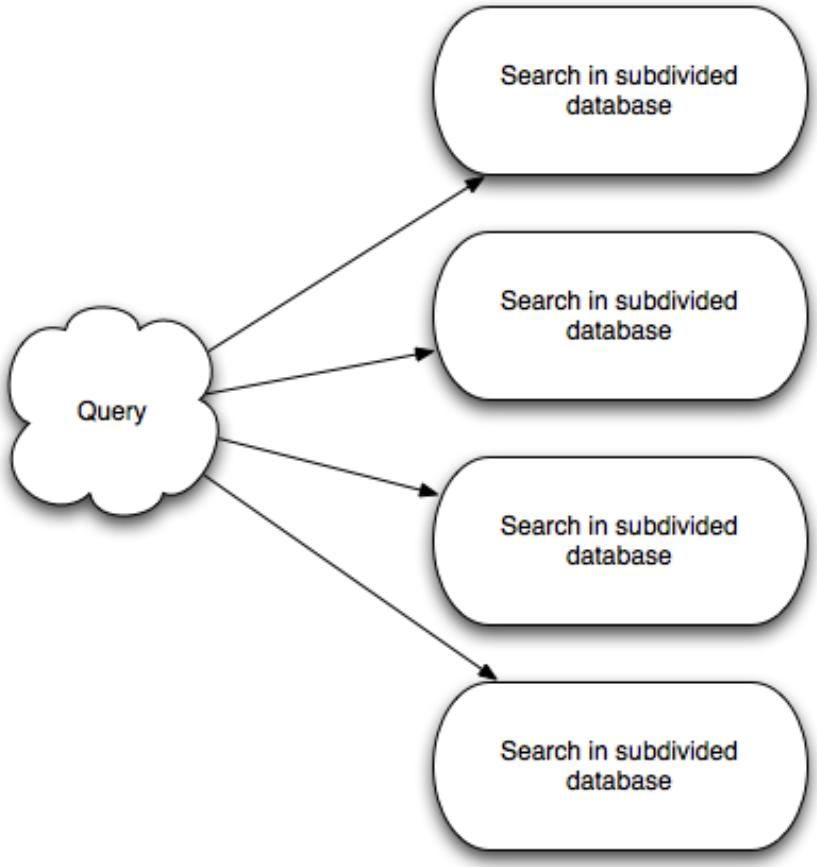


Terminology

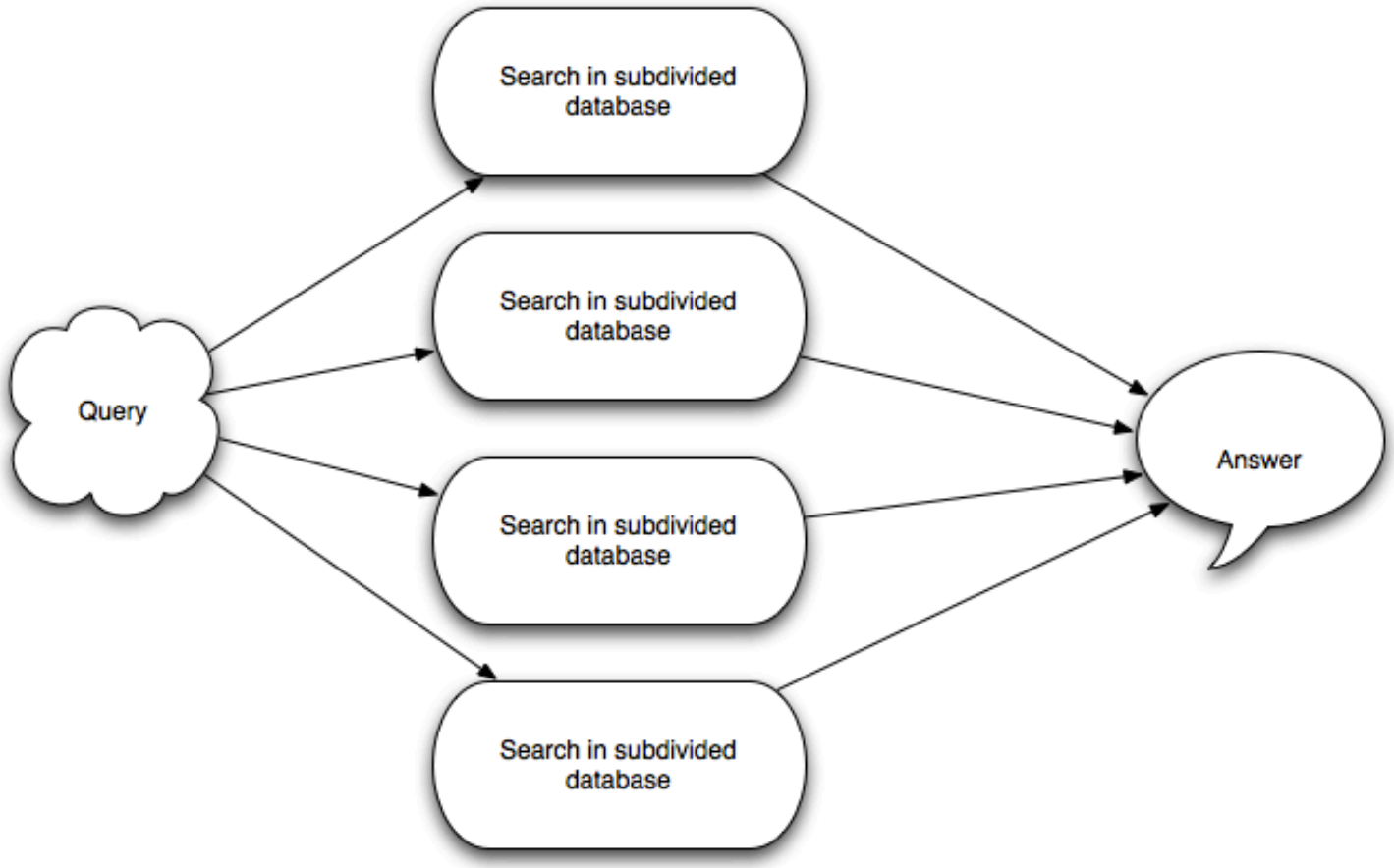
- "Massively parallel" or "pleasantly parallel" – if you throw M resources at an N-sized problem, it becomes N/M-sized for each resource. Think lawn mowing.
- Google has figured out how to do something close to this with search.
- (Does anyone have any intuition as to why *Web search* isn't actually pleasantly parallel?)

Is Google search massively parallel?

Google search



Google search

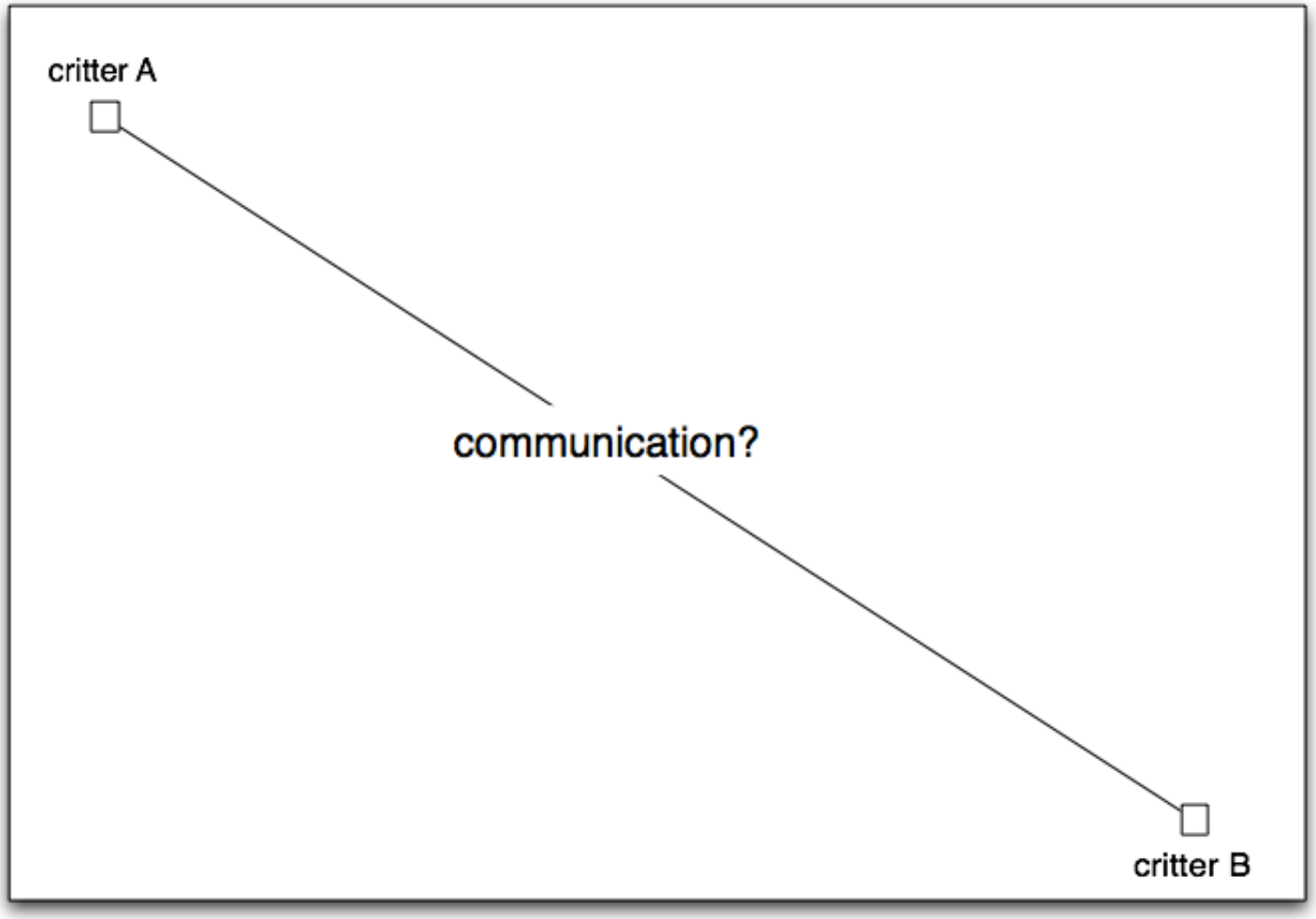


Why is this relevant?

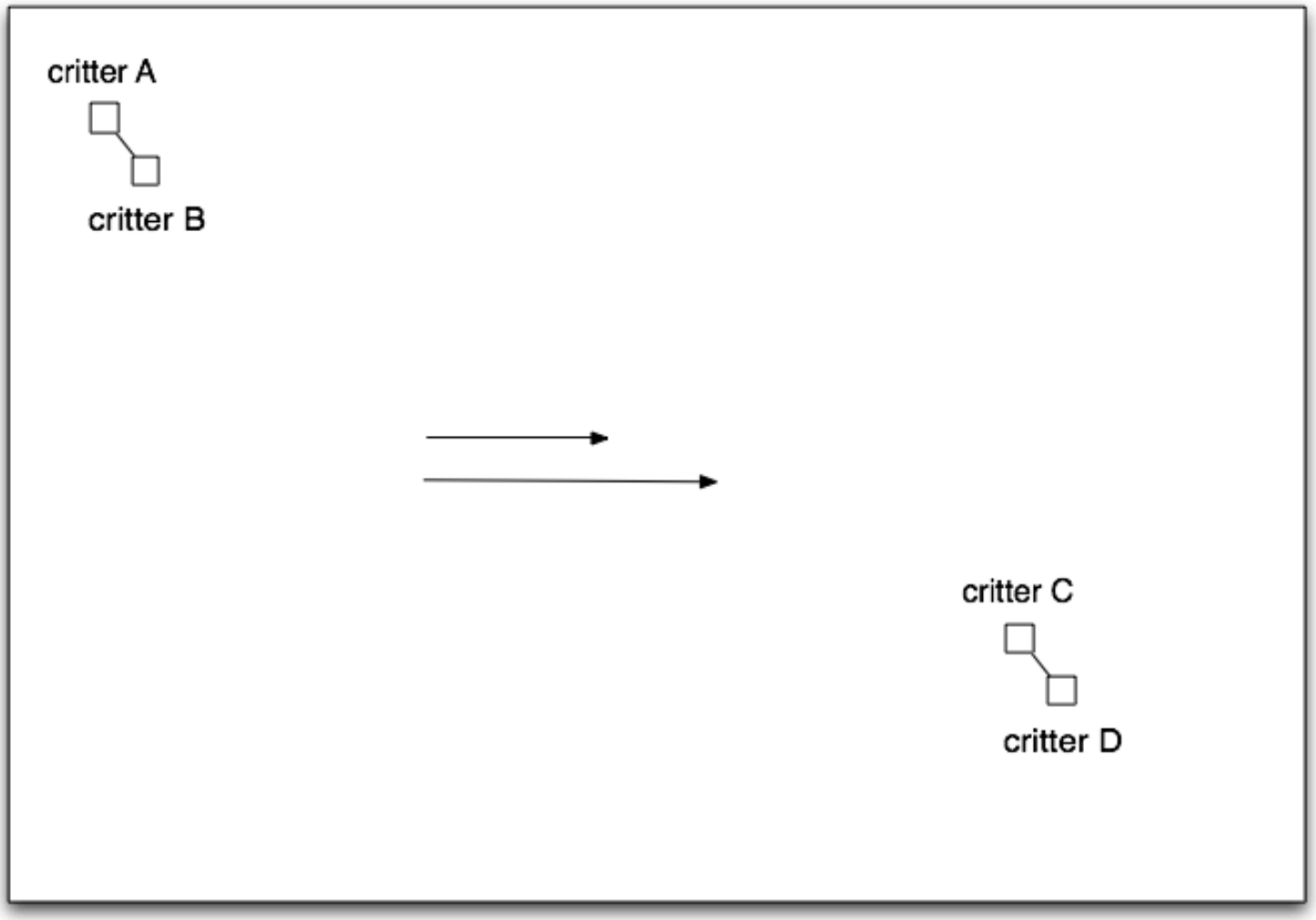
- First, because I'm trying to teach you how to recognize mean & nasty problems.
  - This is especially important when someone tells you that they've managed to scale a difficult problem well. *What tradeoffs?*
- Second: is *Avida* parallelizable?
  - (If you already know the answer, keep quiet.)

Is Avida parallelizable?

Avida parallelization



Avida parallelization



Avida scaling

- Note that computation/generation scales with the size of the world, which is X*Y.
- Parallelizing Avida is therefore possible but not trivial.
- How big a world can you run usefully/ easily? And how can you figure it out without trying to run Avida?

Reminder

- *Algorithm* scaling is independent of the actual time and resources it takes to run the algorithm.
- Scaling tells you how time-to-run (or linked resources) scales as the problem size changes, nothing more.
- Language note:
  - "non trivial" means "wow, that's hard" in computer-speak.
  - "hmmmmmmmm" means "wow, that's reallllly hard."

Combinatorics is the devil

- Think about chess…
- The number of possible moves from any given position is usually *quite* large. Say it's N.
- So to explore all paths (to find the best next move) out to 3, you need to:
  - Explore N moves
    - For each of those, explore N moves
      - For each of those, explore N moves
  - N * N * N => scales badly!

Easy-to-check vs easy-to-find

Given a number, factor it into only prime numbers.

This is hard.

Given a set of prime numbers, verify that they multiply to yield a particular number.

This is easy.

(This is actually the basis of modern encryption…)

Is a number prime?

- Dumb algorithm:
  - For N < P:
    - Does N divide P evenly? If so, not prime.
- Smarter algorithm:
  - For N < sqrt(P):
    - Does N divide P evenly? If so, not prime.
  - (try "100". If N > 10 divides it, then it must be multiplied by a number *smaller* than 100, too)
- Even smarter algorithm:
  - For prime p < sqrt(P):
    - Does p divide P evenly? If so, not prime.

Is a number prime?

- Smarter algorithm:
  - For N < sqrt(P):
    - Does N divide P evenly? If so, not prime.
- Even smarter algorithm:
  - For prime p < sqrt(P):
    - Does p divide P evenly? If so, not prime.
- Which one is faster?
- Which one requires more memory?

The Evil Dean

Suppose:

50 dorm rooms, two students per room

100 students can be admitted, of 400 total

Dean provides list of students that cannot be paired.

The Evil Dean

Suppose:

50 dorm rooms, two students per room

100 students can be admitted, of 400 total

Dean provides list of students that cannot be paired.

It is *easy to check* any particular list of student/room combinations for validity.

In general, it is *extremely hard* to quickly find a guaranteed solution.

Checking your "evil dean" solution

- For each pair, you need to verify that they're not on the list together.

- This is *not* combinatorial –
  - 50 pairs
  - However big a list
  - At worst, you can take each pair and look them up on the list. 50 x length of list.

Finding a guaranteed solution strategy for your evil dean…

- That's much harder to do quickly. *THAT's* combinatorial.
- First assignment: 400, choose 2. (79,800)
- Second assignment: 398, choose 2. (79003 ways to do that.)
- Or, 400*399 possible pairs to check against list… find 50 (of 156k) and you're good! ("Greedy algorithm")
- Etc. Scales REALLY badly with size of dorm, size of rooms, size of list, number of students.

Finding a *probably-correct* solution for your evil dean…

- That might be much easier. (Randomly pick roommates? Or by alphabetical last?)
- …you can probably rely on roommates to tell you if they hate each other, and then you switch just those. Odds are you won't get in too much trouble.
- Unless you end up with a murder, of course.

Heuristics

- "Heuristics" are short cuts that usually work (but can go horribly wrong).

- Not all problems are amenable.
  - Prime numbers? No good, *fast* short cut.
  - Evil dean & housing? Sure – start with a random solution, eliminate one of each pair that conflicts, until you find a non-conflict..

- Heuristics rely on assumptions about the specific type of problem you're going to tackle, and don't always work.
  - If the Dean is evil, he can construct a list of incompatible roommates that breaks your process.
  - Or he can just gives you a really long list of incompatible roommates.

Heuristics, again

- You can choose the problem so that any given heuristic fails, *and* it's just as difficult to figure out what the best heuristic is as it is to… run the approach.

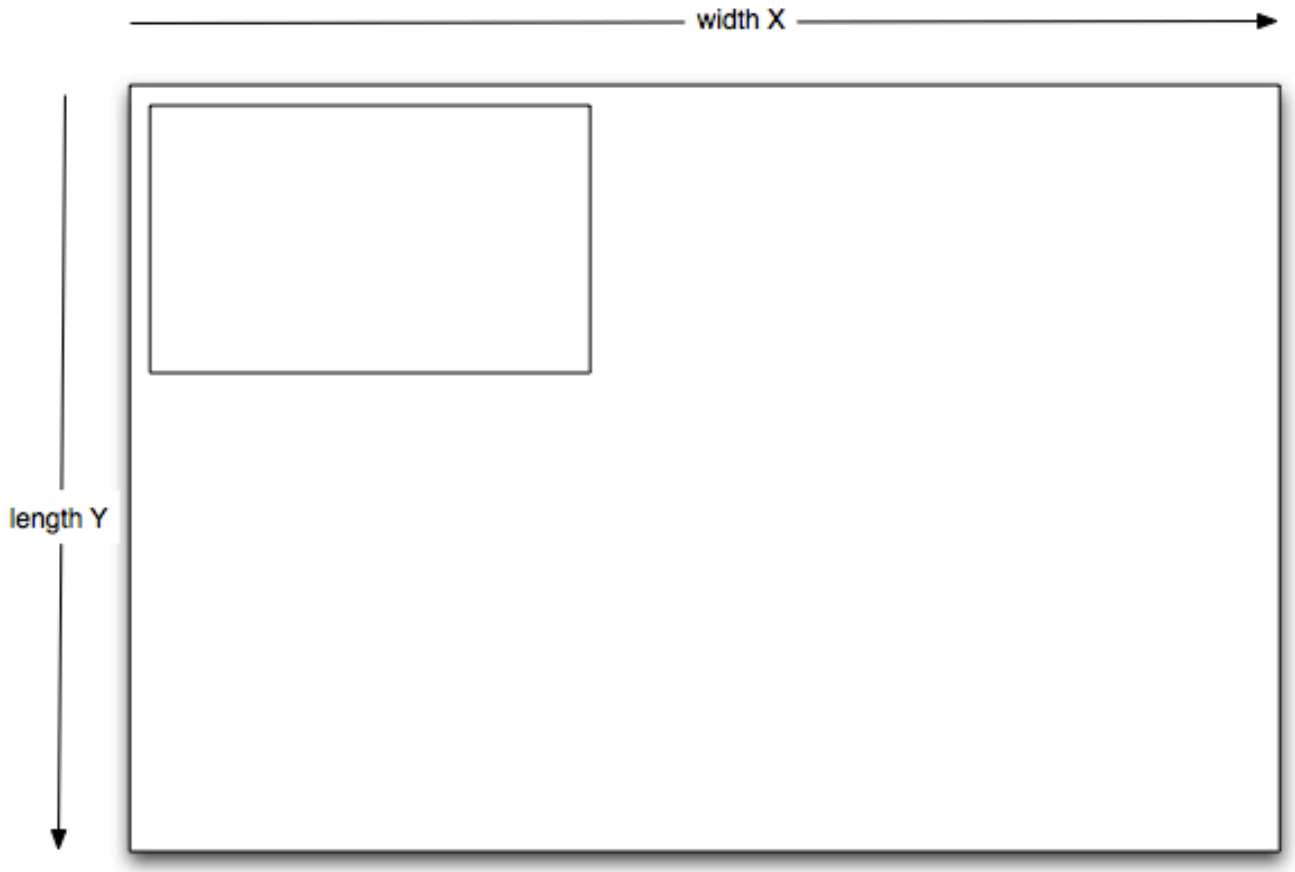
In summary, about algorithms

- Many useful problems are also NP-complete, because of the way they scale.

- Heuristic solutions (planning for the best!) or approximate solutions (it's ok if we're wrong some small amount of the time) are much easier than exact solutions.

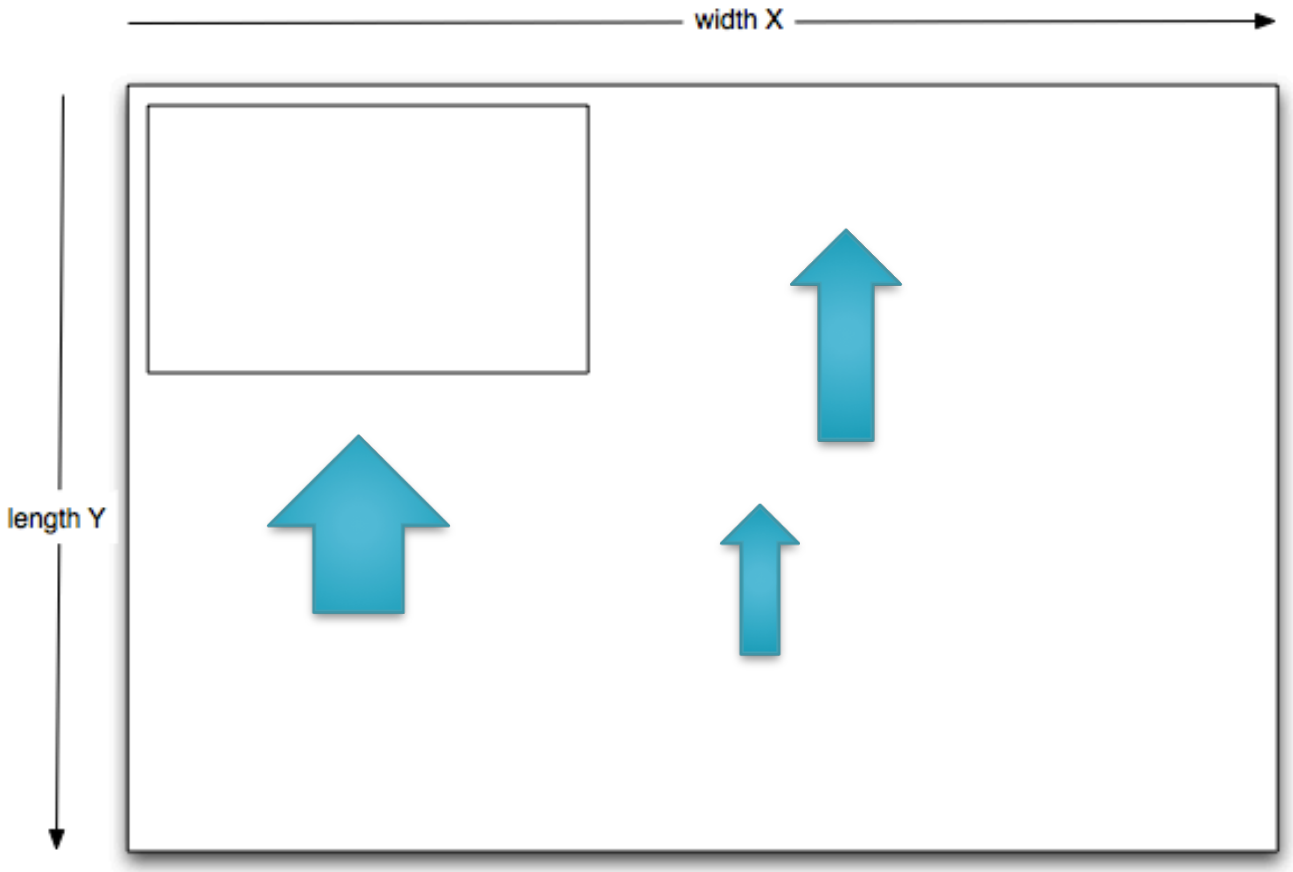
What (some) Computer Scientists do: re lawn mowing

- How does the problem scale?



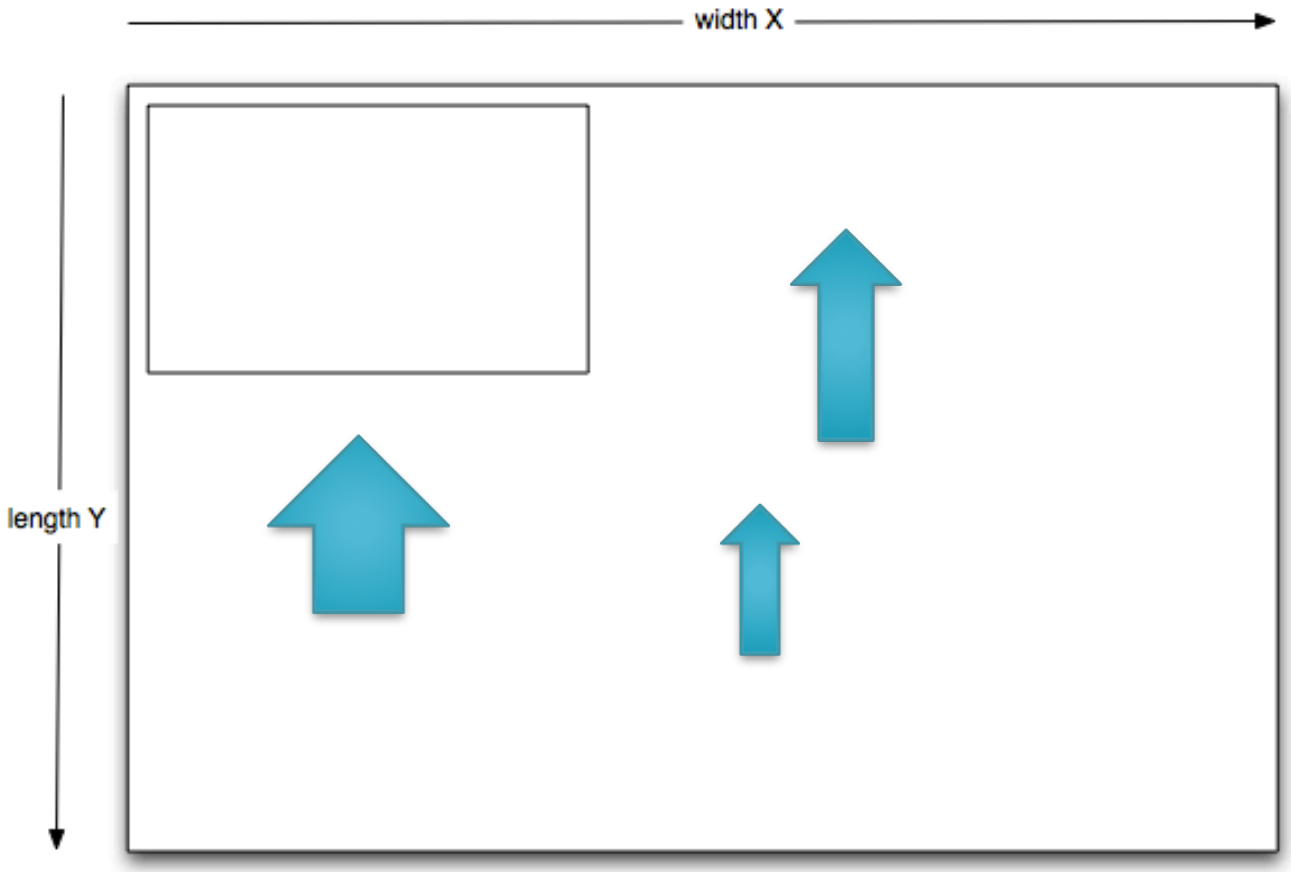
What (some) Computer Scientists do: re lawn mowing

- If you have trees, what's the right way to subdivide?



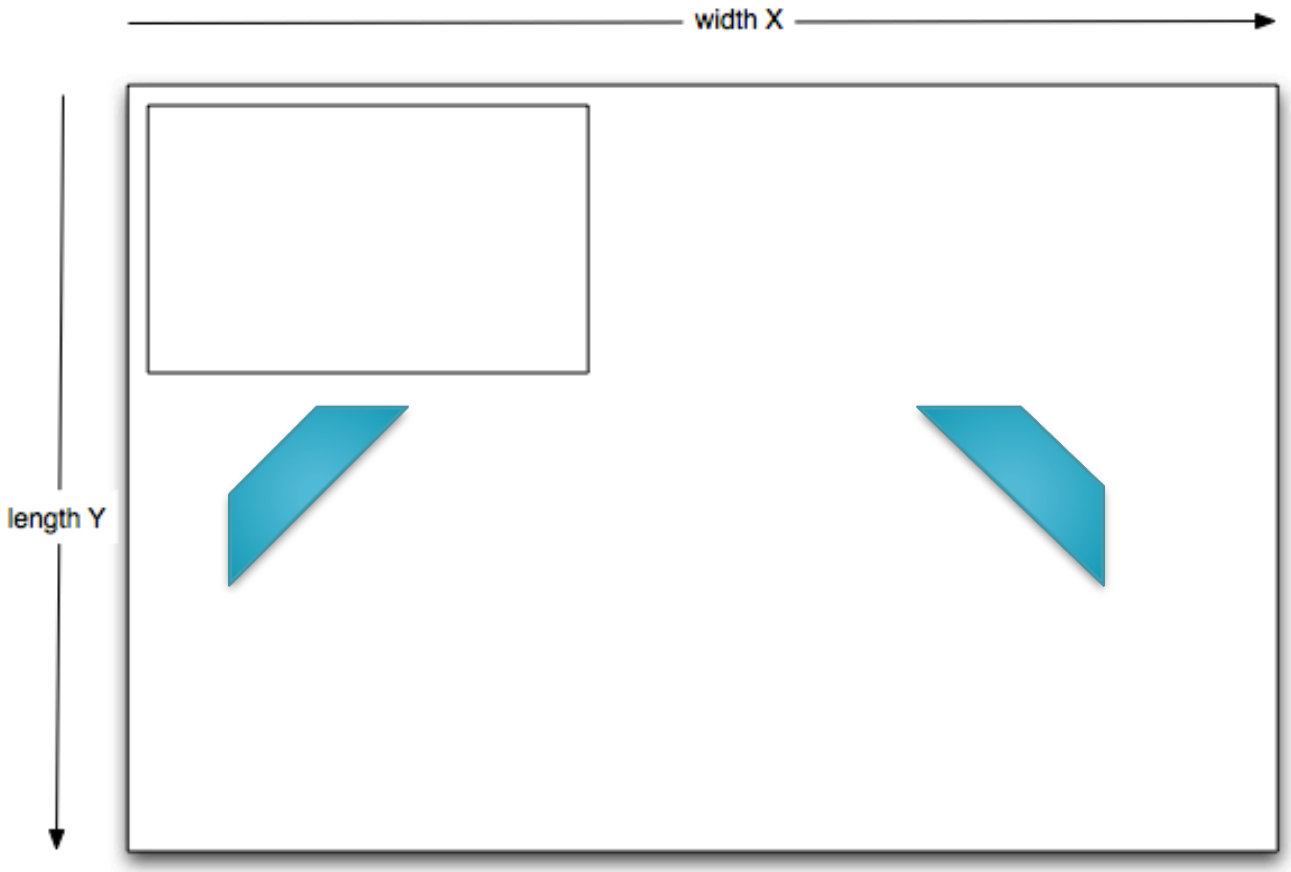
What (some) Computer Scientists do: re lawn mowing

- Suppose you have a Mad Gardener planting new trees *while* you mow?



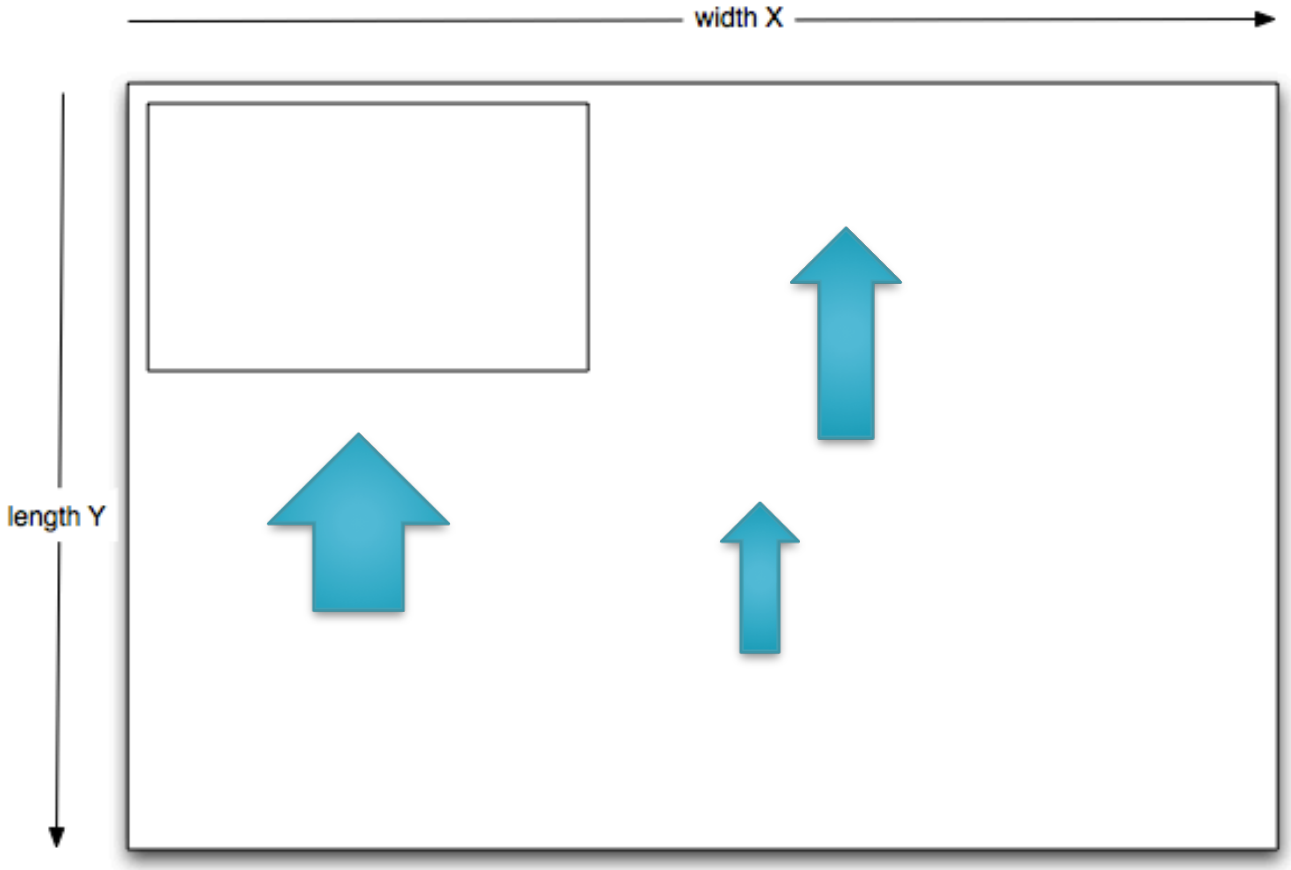
What (some) Computer Scientists do: re lawn mowing

- Or, suppose the lawn is on a hill – how do you subdivide?



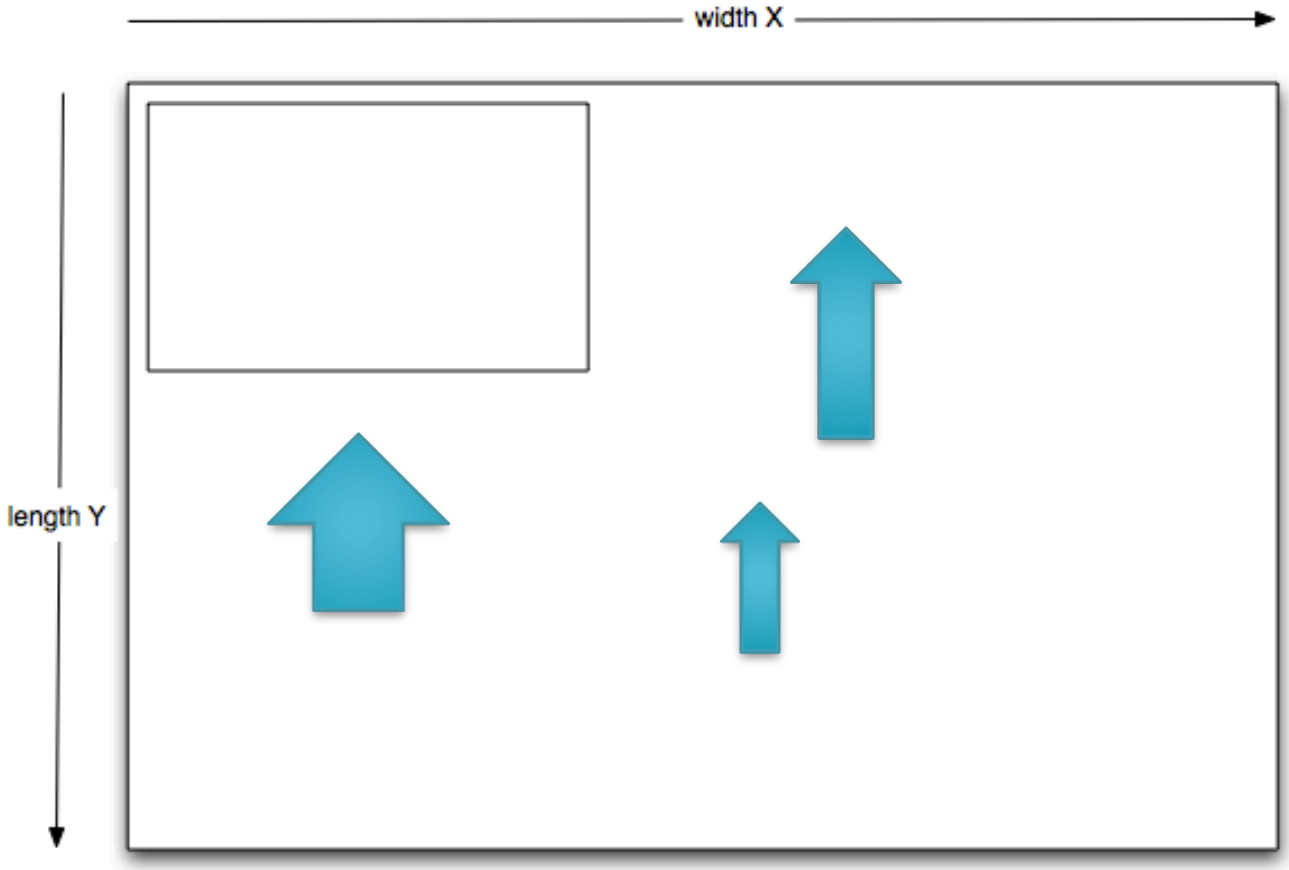
What (some) Computer Scientists do: re lawn mowing

- Suppose you run a lawn mowing *company*, and you face all of the above problems? What's the best *approach?*



What (some) Computer Scientists do: re lawn mowing

- Suppose you run a *space habitat lawn mowing* company? How do you mow arbitrarily shaped 2-D spaces?



What (some) Computer Scientists do: re lawn mowing

- Suppose you run a *space habitat lawn mowing* company? How do you mow arbitrarily shaped 2-D spaces?

- (You know you're a mathematician/computer scientist if you can't go to sleep at night because you're thinking about how to do that.)

What (some) Computer Scientists do: re lawn mowing

- Suppose you run a *space habitat lawn mowing* company? How do you mow arbitrarily shaped 2-D spaces?

- (You know you're a mathematician/computer scientist if you can't go to sleep at night because you're thinking about how to do that.)

- (You're a writer if you immediately think, neat! I could write a sci fi story about a space habitat lawn mowing company!)

Classifying problems

- The main classification you need to know is: NP-complete.
  - Easy to verify.
  - Hard to compute.
  - *Asymmetric.*

- Our dean problem is NP-complete.
- Many (most?) hard problems I run across are NP-complete, I think.

Black box nature of algorithms

- When you listen to a computational scientist explain their clever algorithm…
- …it's a mistake to think that they necessarily *know* what's going on.
- Software is full of bugs and unintended consequences.

Tracking the process

- How do you know that your software *today* is doing the same thing it was doing last month? Or last year? Or in the hands of that other graduate student?

- There are some tools & techniques for dealing with change in software. We'll talk about them in a week or two.

Avida stochasticity: why?

Sarah Jones:

UD: 500 Gen: 38.53015 Fit: 0.2479998 Merit: 97.69530

Robert Heckendorn:

UD: 500 Gen: 39.12720 Fit: 0.2561943 Merit: 98.58119

Max Maliska:

UD: 500 Gen: 39.35769 Fit: 0.2472624 Merit: 94.19189

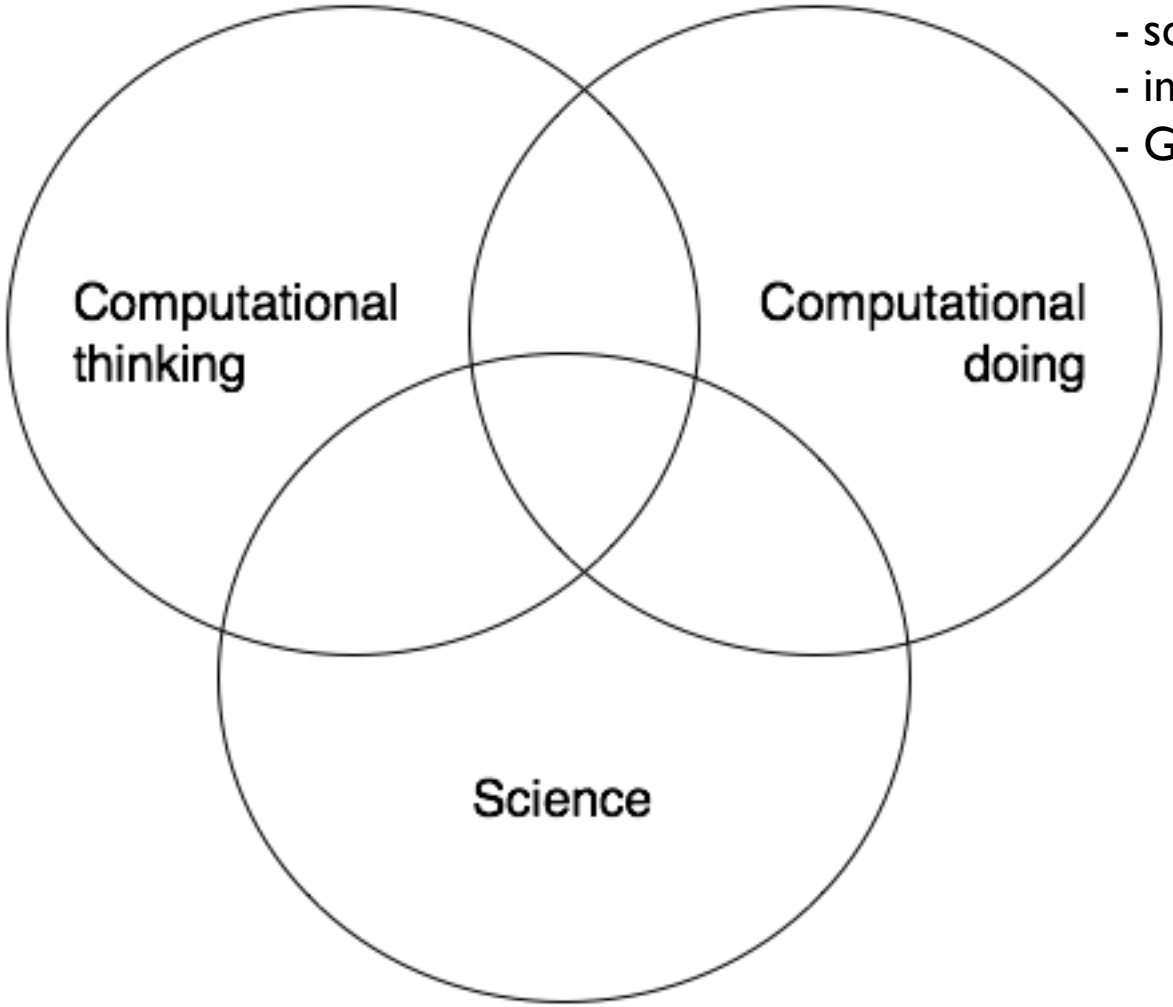
Jeannine McLean:

UD: 500 Gen: 37.32764 Fit: 0.2461677 Merit: 99.99727

Avida stochasticity: uh-oh?

- Read the Nature paper.
- How easy would it be to replicate those results today?

Computational science



Who gets what wrong?
- scaling
- implementation details
- Good Science