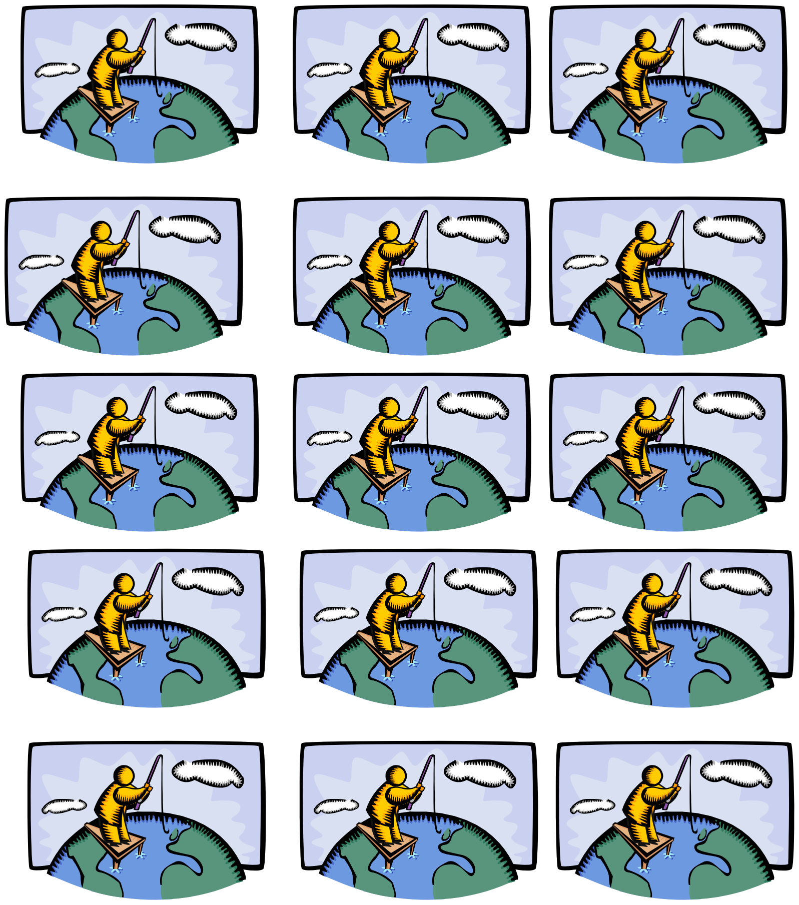
# Parallelism

# What is a Parallelism - Simple Analogy

# Parallelism

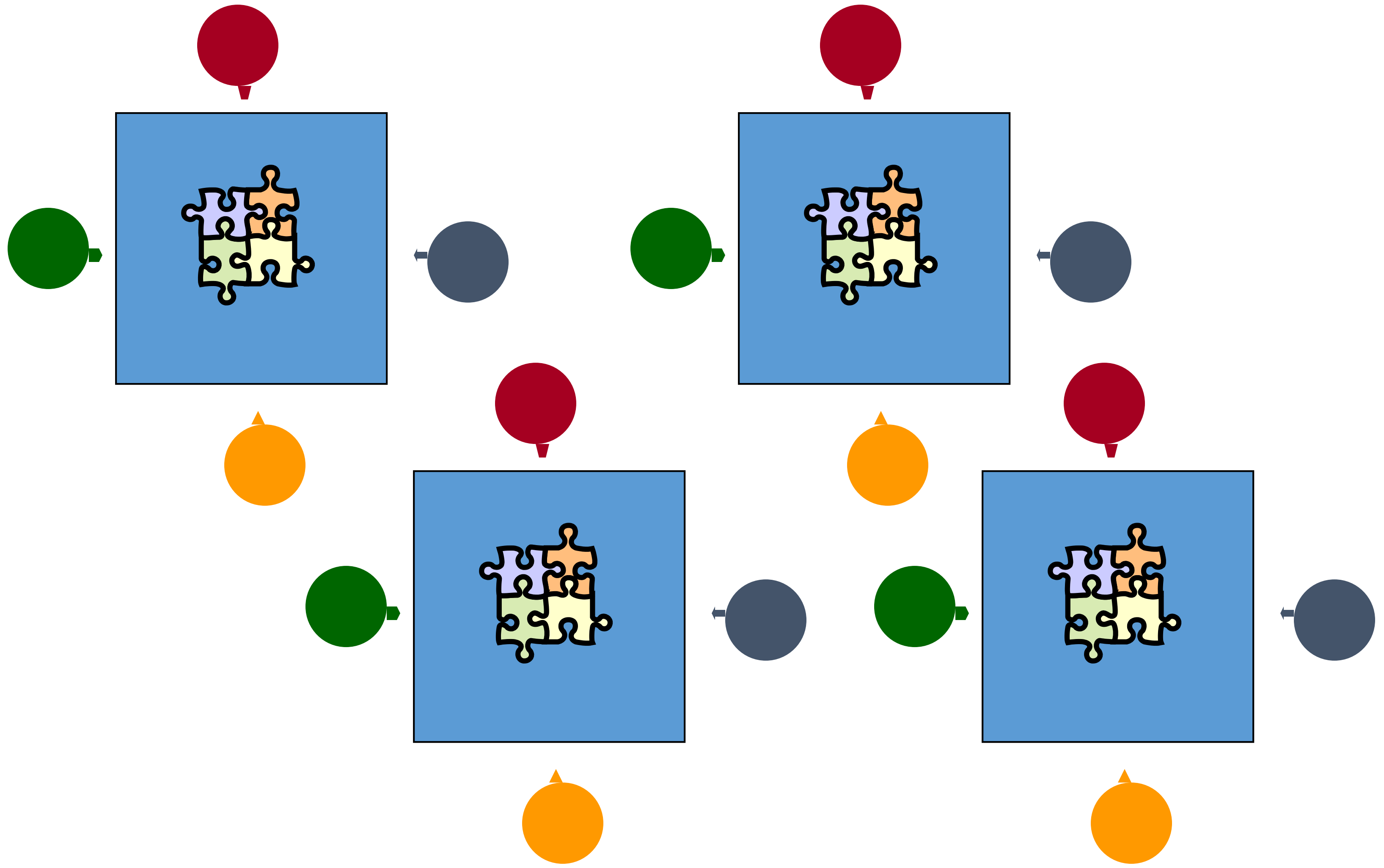
***<u>Parallelism</u>*** means doing multiple things at the same time; you can get more work done in the same time.

Less fish …

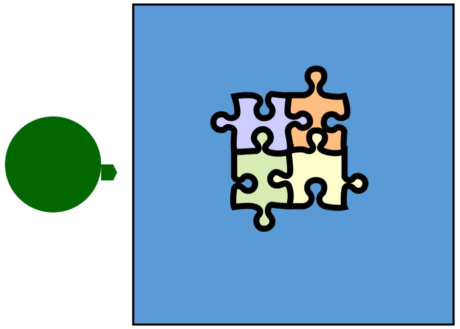


More fish!

# The Jigsaw Puzzle Analogy



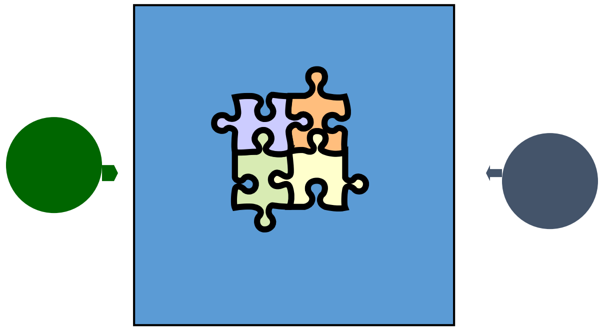
## Serial Computing



Suppose you want to do a jigsaw puzzle that has, say, a thousand pieces.

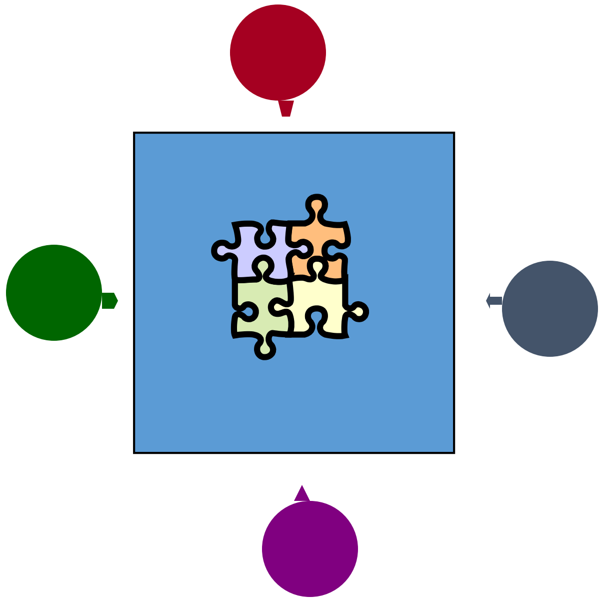
We can imagine that it'll take you a certain amount of time. Let's say that you can put the puzzle together in an hour.

## Shared Memory Parallelism



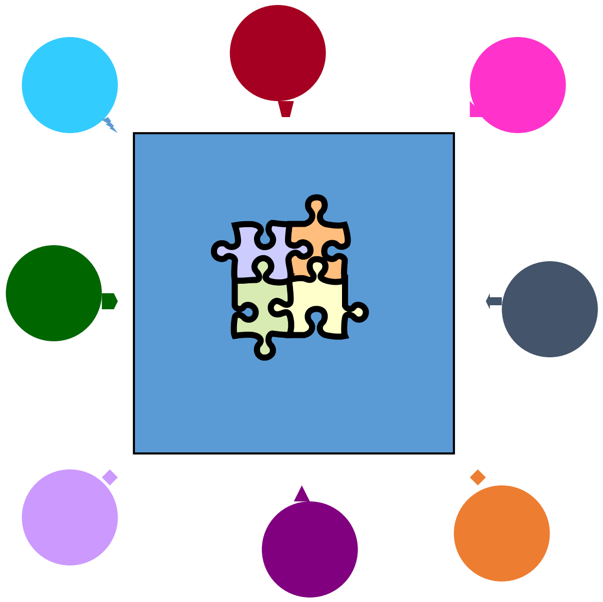
If Scott sits across the table from you, he can work on his half of the puzzle, and you can work on yours. Once in a while, you'll both reach into the pile of pieces at the same time (you'll ***contend*** for the same resource), which will cause a little bit of slowdown. And from time to time, you'll have to work together (***communicate***) at the interface between his half and yours. The speedup will be nearly 2-to-1; you might take 35 minutes instead of 30.

## The More the Merrier?



Now, let's put Paul and Charlie on the other two sides of the table. Each of you can work on a part of the puzzle, but there'll be a lot more contention for the shared resource (the pile of puzzle pieces) and a lot more communication at the interfaces. So, you will get noticeably less than a 4-to-1 speedup, but you'll still have an improvement - maybe something like 3-to-1. The four of you can get it done in 20 minutes instead of an hour.

## Diminishing Returns



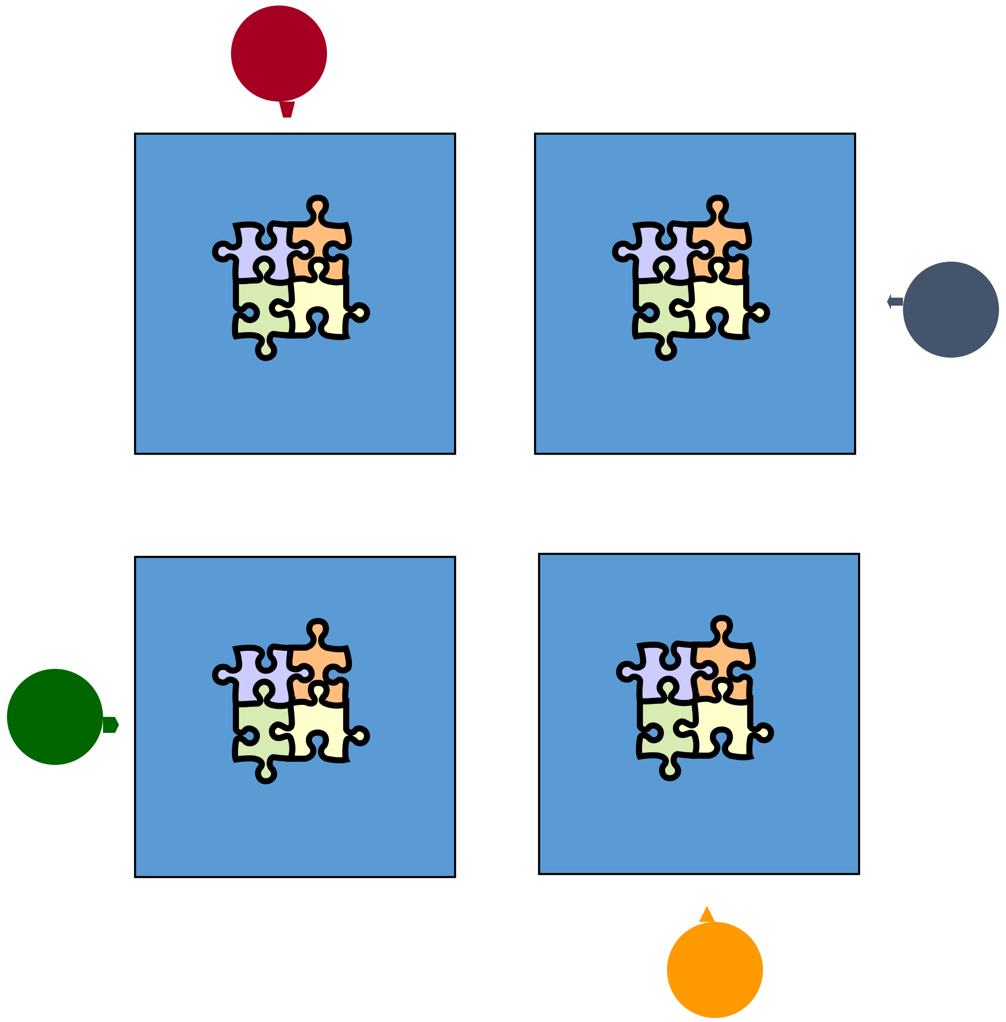
If we now put Dave, Tom, Horst, and Brandon on the corners of the table, there's going to be a whole lot of contention for the shared resource and a lot of communication at the many interfaces. So, the speedup you'll get will be much less than we'd like; you'll be lucky to get 5-to-1.

So, we can see that adding more and more workers onto a shared resource is eventually going to have a diminishing return.

## Distributed Parallelism

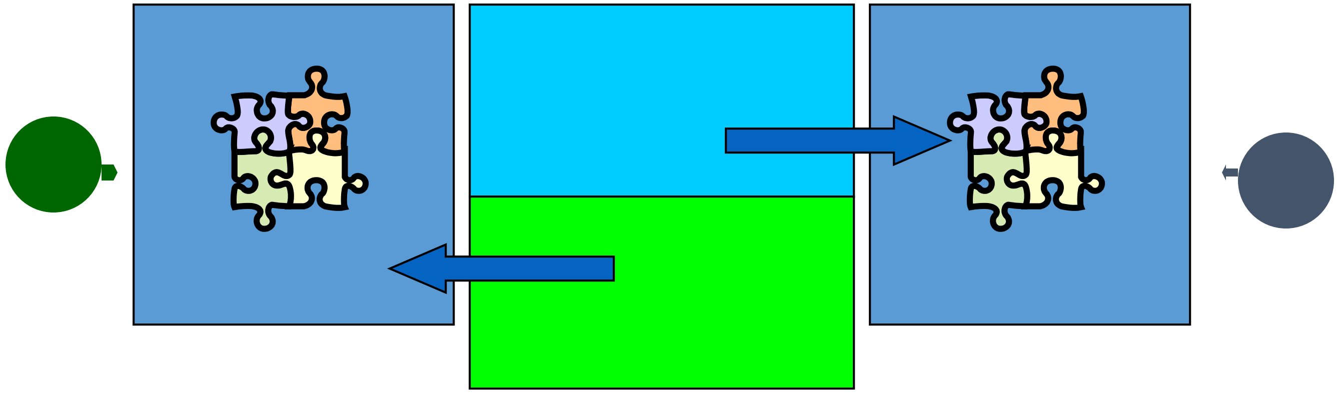
Now let’s try something a little different. Let’s set up two tables, and let’s put you at one of them and Scott at the other.  Let’s put half of the puzzle pieces on your table and the other half of the pieces on Scott’s table. Now you can work completely independently, without any contention for a shared resource. **BUT**, the cost per communication is **MUCH** higher (you have to scootch your tables together), and you need the ability to split up (***decompose***) the puzzle pieces reasonably evenly, which may be tricky to do for some puzzles.

# More Distributed Processors



It's a lot easier to add more processors in distributed parallelism. But, you always have to be aware of the need to decompose the problem and to communicate among the processors. Also, as you add more processors, it may be harder to ***load balance*** the amount of work that each processor gets.

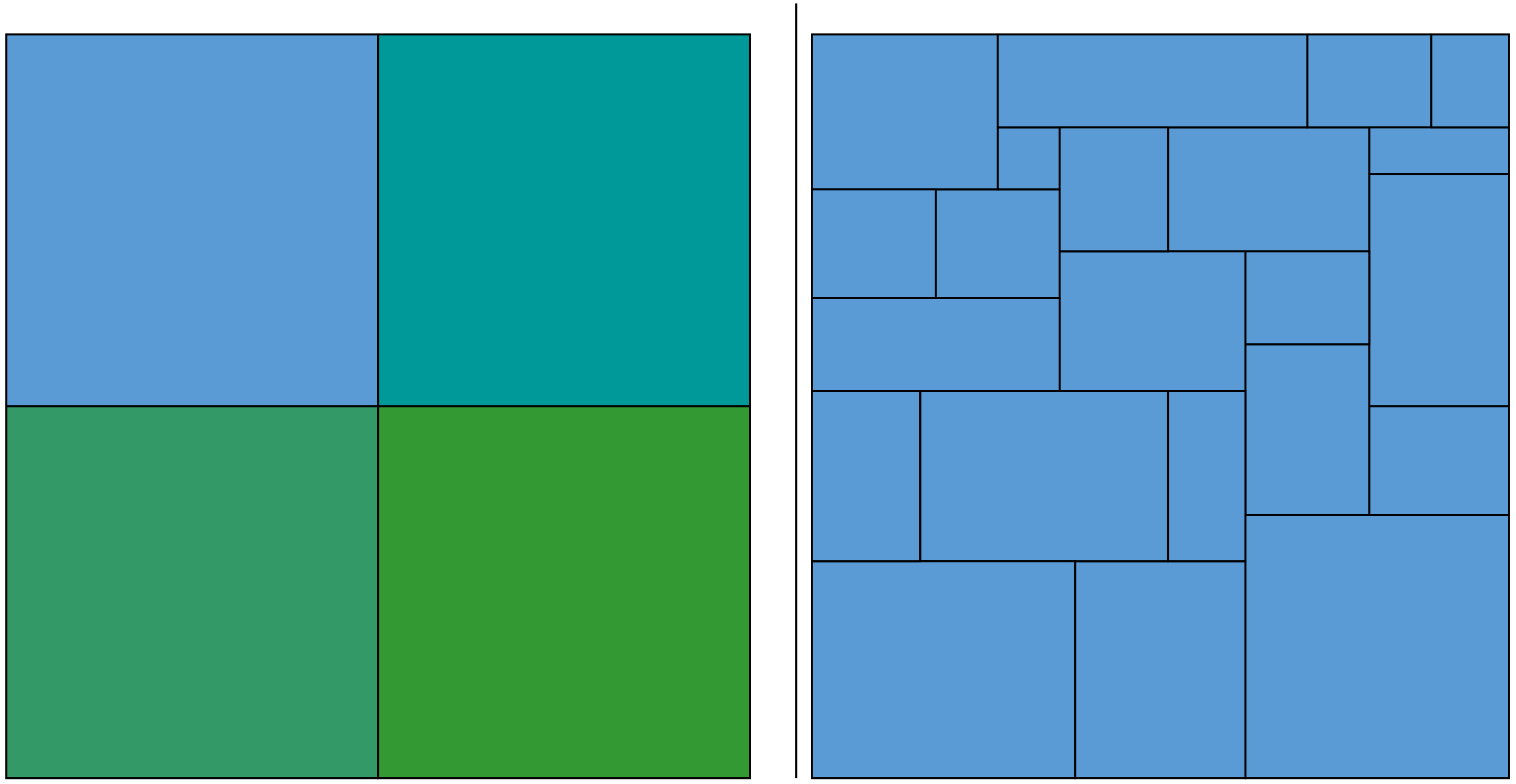
## Load Balancing



***Load balancing*** means ensuring that everyone completes their workload at roughly the same time.

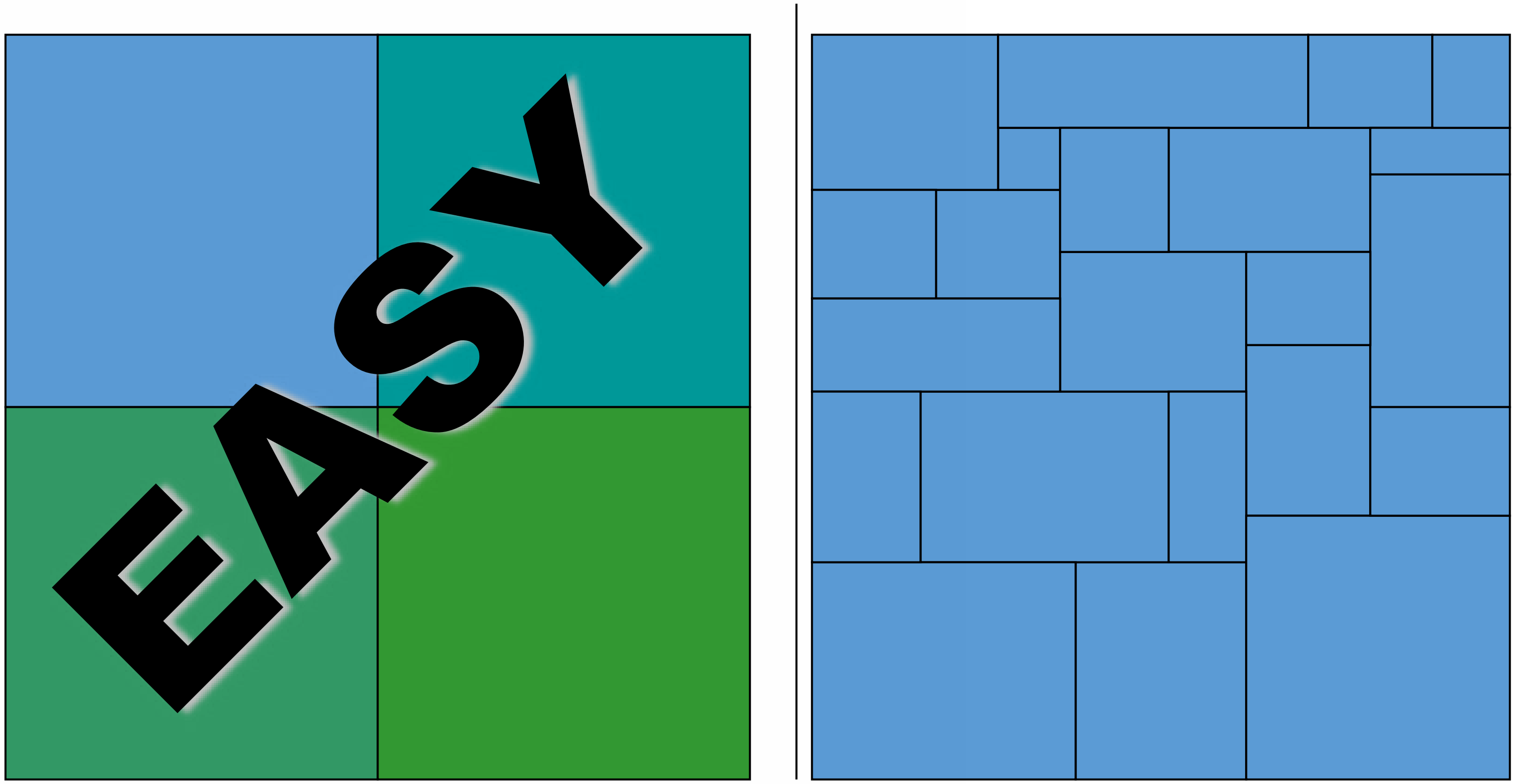
For example, if the jigsaw puzzle is half grass and half sky, then you can do the grass and Scott can do the sky. Then you'll only have to communicate at the horizon – and the amount of work that each of you does on your own is roughly equal. So you'll get pretty good speedup.

## Load Balancing



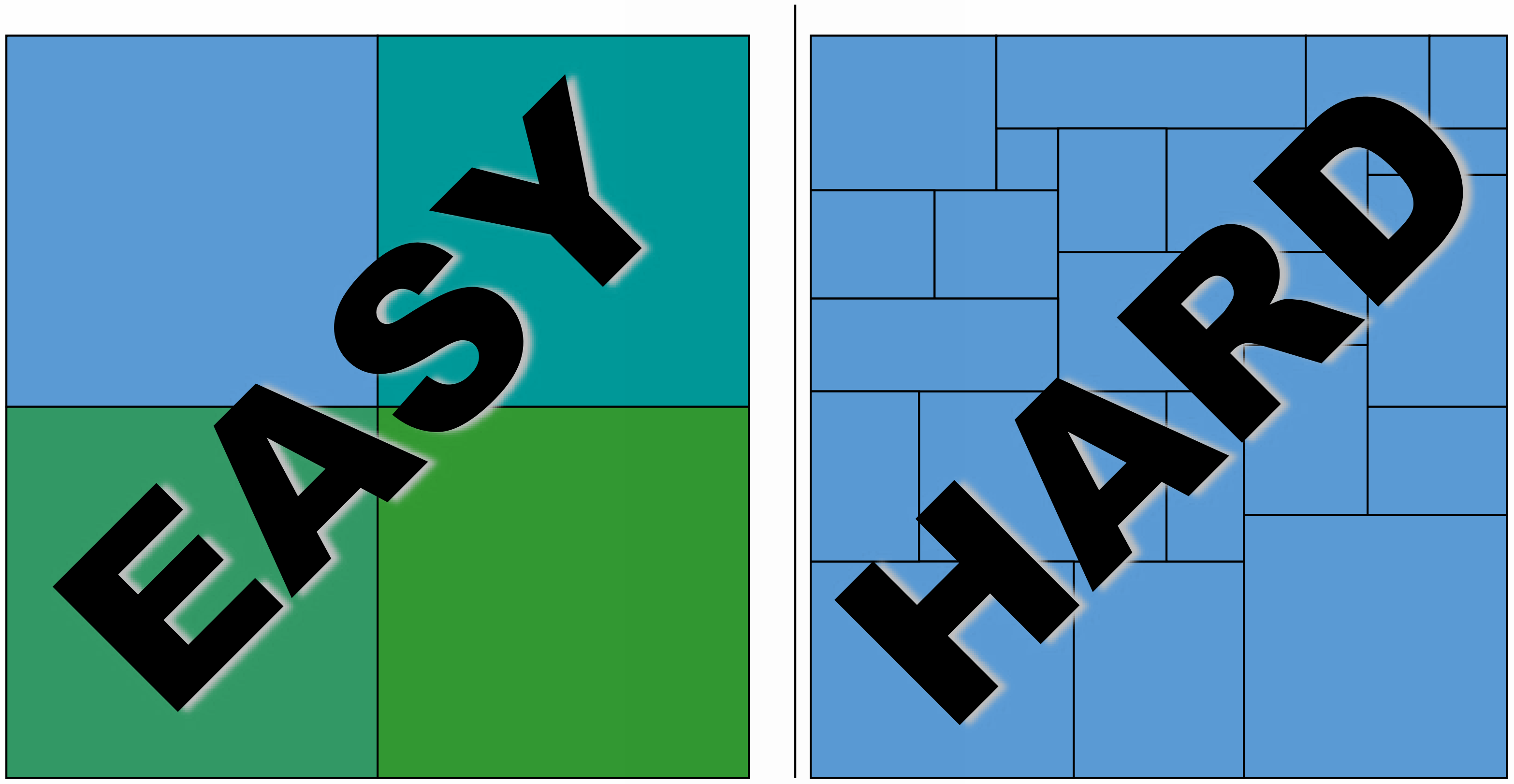
Load balancing can be easy if the problem splits up into chunks of roughly equal size with one chunk per processor. Or, load balancing can be very hard.

## Load Balancing



Load balancing can be easy if the problem splits up into chunks of roughly equal size with one chunk per processor. Or, load balancing can be very hard.

## Load Balancing



Load balancing can be easy if the problem splits up into chunks of roughly equal size with one chunk per processor. Or, load balancing can be very hard.

## Why Bother?

## Why Bother with HPC at All?

- It’s clear that making effective use of HPC takes quite a bit of effort, both learning how and developing software.
- That seems like a lot of trouble to just get your code to run faster.
- It’s nice to have a code that used to take a day and now runs in an hour. But, if you can afford to wait a day, what’s the point of HPC?
- Why go to all that trouble just to get your code to run faster?

## Why HPC is Worth the Bother

- What HPC gives you that you won't get elsewhere is the ability to do **bigger, better, and more exciting science**. If your code can run faster, that means that you can tackle much bigger problems in the same amount of time that you used to need for smaller problems.
- HPC is important not only for its own sake, but also because what happens in HPC today will be on your desktop in about 10 to 15 years and on your cell phone in 25 years; it puts you **ahead of the curve**.

## The Future is Now

- Historically, this has always been true:
  - **Whatever happens in supercomputing today will be on your desktop down the road.**
- So, if you have experience with supercomputing, you'll be ahead of the curve when things get to the desktop.
- Exascale

# Exa-scale Challenges

- Processor architecture
- Facility power is the primary constraint for the exascale system
  - A Xeon based 5.34 petaflop system (Cheyenne) consumes 1.72 MW, so an exaflop computer would require 320 MW, which is untenable. GPU and KNL have lower power footprints.
  - The target is 20-40 MW in 2020 for 1 exaflop.
- Memory bandwidth and capacity are not keeping pace with the increase in flops
- Clock frequencies are expected to decrease to conserve power
- Cost of data movement
- A new programming model will be necessary
- The I/O system will be much harder to manage
- Reliability and resiliency will be critical at the scale (Component mean-time-to-failure)
- Cost

## Thanks for your attention!

Questions?
Irfan@ucar.edu

## References

[1] Image by Greg Bryan, Columbia U.

[2] "Update on the Collaborative Radar Acquisition Field Test (CRAFT): Planning for the Next Steps." Presented to NWS Headquarters August 30 2001.

[3] See http://hneeman.oscer.ou.edu/hamr.html for details.

[4] http://www.dell.com/

[5] http://www.vw.com/newbeetle/

[6] Richard Gerber, The Software Optimization Cookbook: High-performance Recipes for the Intel Architecture. Intel Press, 2002, pp. 161-168.

[7] RightMark Memory Analyzer. http://cpu.rightmark.org/

[8] ftp://download.intel.com/design/Pentium4/papers/24943801.pdf

[9] http://www.samsungssd.com/meetssd/techspecs

[10] http://www.samsung.com/Products/OpticalDiscDrive/SlimDrive/OpticalDiscDrive_SlimDrive_SN_S082D.asp?page=Specifications

[11] ftp://download.intel.com/design/Pentium4/manuals/24896606.pdf

[12] http://www.pricewatch.com/

*Special Thanks to Henry Neeman, University of Oklahoma for the use of his slides of LCI Workshop, Monday, May 18, 2015.*

## Acknowledgements

- Henry Neeman, University of Oklahoma for his 2015 LCI Slides.
- Erik Scott (Harris) , Jared David Baker (University of Wyoming) , Pamela Hill (NCAR), Shilo Hall (NCAR), Nathan Rini (NCAR), Ben Matthews (NCAR), Jon Roberts (NCAR),  Thomas Engel (NCAR), , Jeffrey R. Lang (University of Wyoming) , Jonathan Anderson (University Colorado, Boulder), Brian Dale Haymore (University of Utah), Leslie Ann Froeschl (University of Illinois) ,Tim Brewer (University of Wyoming),  Stormy Knight (NCAR), Robert McLay (TACC) for reviewing and providing feedback on the slides.
- LCI for the opportunity.