



UC Berkeley  
Sr Lecturer SOE  
Dan Garcia

# The Beauty and Joy of Computing

## Lecture #18 Distributed Computing



**FASTER THAN 50M LAPTOPS!**

By the end of the decade, we're going to see computers that can compute one exaFLOP (recall kilo, mega, giga, tera, peta, exa), and we've just hit 10 petaFLOPs!





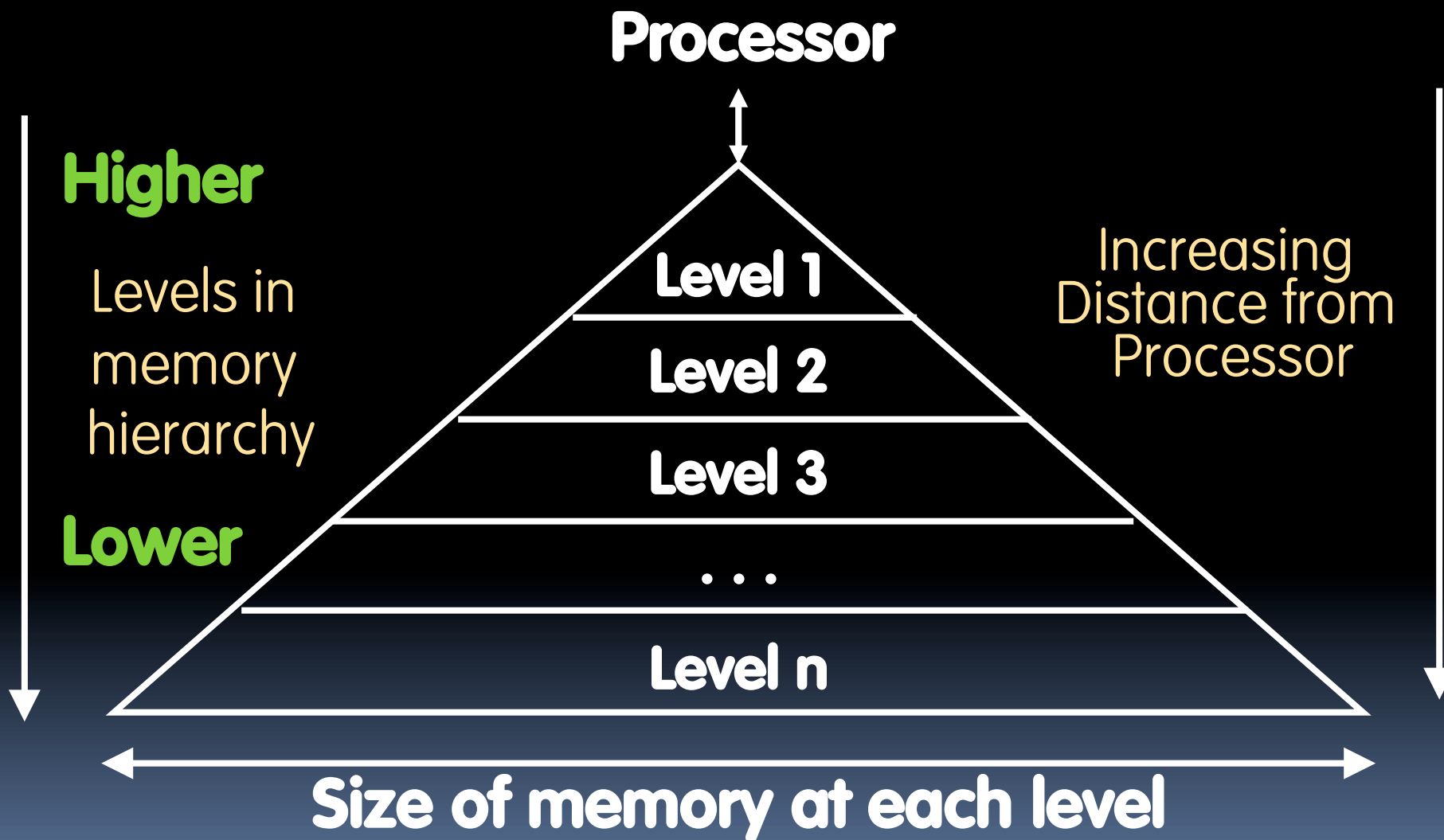
# Lecture Overview

- **Basics**
  - Memory
  - Network
- **Distributed Computing**
  - Themes
  - Challenges
- **Solution! MapReduce**
  - How it works
  - Our implementation





# Memory Hierarchy





# Memory Hierarchy Details

---

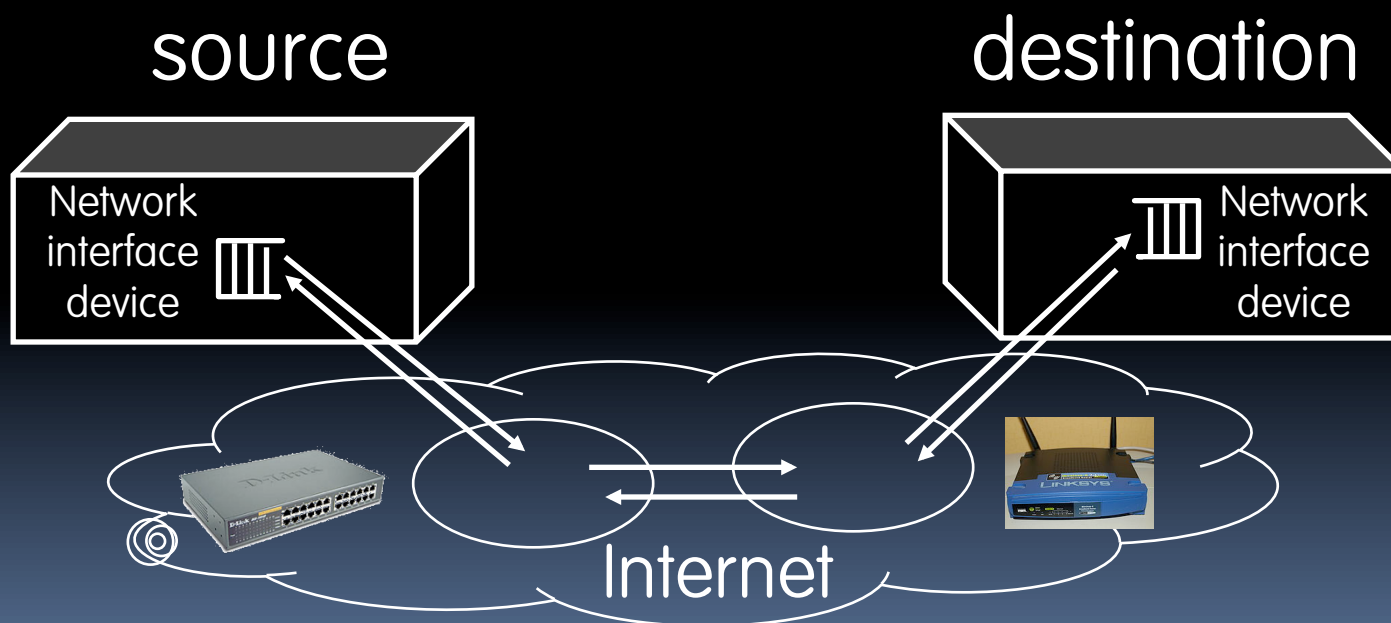
- If level closer to Processor, it is:
  - Smaller
  - Faster
  - More expensive
  - subset of lower levels
    - ...contains most recently used data
- **Lowest Level (usually disk) contains all available data (does it go beyond the disk?)**
- Memory Hierarchy **Abstraction** presents the processor with the illusion of a very large & fast memory





# Networking Basics

- source **encodes** and destination **decodes** content of the message
- **switches** and **routers** use the destination in order to deliver the message, dynamically





# Networking Facts and Benefits

- **Networks connect computers, sub-networks, and other networks.**
  - Networks connect computers all over the world (and in space!)
  - Computer networks...
    - support asynchronous and distributed communication
    - enable new forms of collaboration

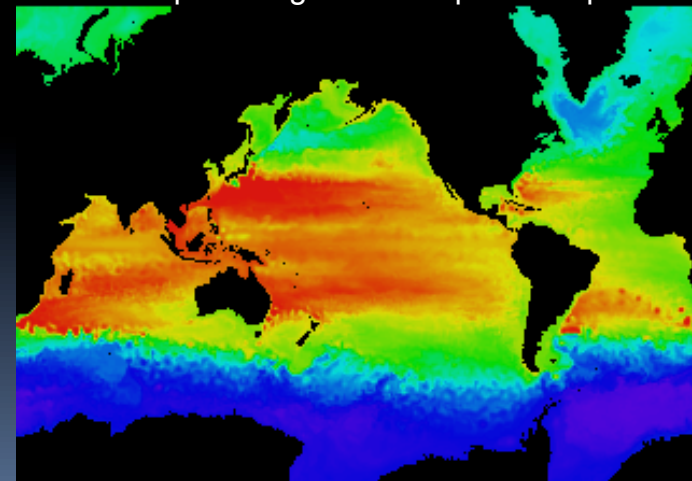




# Performance Needed for Big Problems

- **Performance terminology**
  - the FLOP: Floating point Operation
  - “flops” = # FLOP/second is the standard metric for computing power
- **Example: Global Climate Modeling**
  - Divide the world into a grid (e.g. 10 km spacing)
  - Solve fluid dynamics equations for each point & minute
    - Requires about 100 Flops per grid point per minute
  - Weather Prediction (7 days in 24 hours):
    - 56 Gflops
  - Climate Prediction (50 years in 30 days):
    - 4.8 Tflops
- **Perspective**
  - Intel Core i7 980 XE Desktop Processor
    - ~100 Gflops
    - Climate Prediction would take ~5 years

[www.epm.ornl.gov/champp/champp.html](http://www.epm.ornl.gov/champp/champp.html)



Garcia





# What Can We Do? Use Many CPUs!

- **Supercomputing – like those listed in top500.org**
  - Multiple processors “all in one box / room” from one vendor that often communicate through shared memory
  - This is often where you find exotic architectures
- **Distributed computing**
  - Many separate computers (each with independent CPU, RAM, HD, NIC) that communicate through a network
    - Grids (heterogenous computers across Internet)
    - Clusters (mostly homogeneous computers all in one room)
      - Google uses commodity computers to exploit “knee in curve” price/performance sweet spot
  - It’s about being able to solve “big” problems, not “small” problems faster
    - These problems can be data (mostly) or CPU intensive

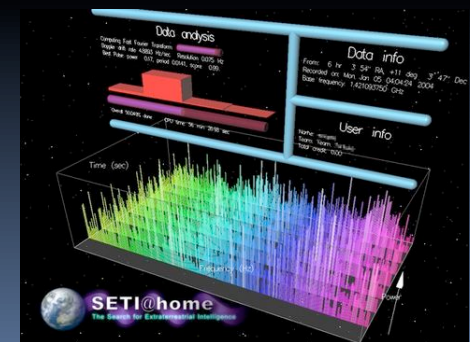






# Distributed Computing Themes

- Let's network many disparate machines into one compute cluster
- These could all be the same (easier) or very different machines (harder)
- Common themes
  - "Dispatcher" gives jobs & collects results
  - "Workers" (get, process, return) until done
- Examples
  - SETI@Home, BOINC, Render farms
  - Google clusters running MapReduce



Garcia





# Peer Instruction



1. Writing & managing **SETI@Home** is relatively straightforward; just hand out & gather data
2. The majority of the world's computing power lives in supercomputer centers

	12
a)	FF
b)	FT
c)	TF
d)	TT

Garcia





# Peer Instruction Answer



1. The heterogeneity of the machines, handling machines that fail, falsify data. **FALSE**
2. Have you considered how many PCs + game devices exist? Not even close. **FALSE**

1. Writing & managing **SETI@Home** is relatively straightforward; just hand out & gather data
2. The **majority of the world's computing power** lives in supercomputer centers

	12
a)	<b>FF</b>
b)	<b>FT</b>
c)	<b>TF</b>
d)	<b>TT</b>

Garcia





# Distributed Computing Challenges

- **Communication is fundamental difficulty**
  - Distributing data, updating shared resource, communicating results, handling failures
  - Machines have separate memories, so need network
  - Introduces inefficiencies: overhead, waiting, etc.
- **Need to parallelize algorithms, data structures**
  - Must look at problems from parallel standpoint
  - Best for problems whose compute times  $\gg$  overhead



"Up" (L-R) Dug, Russell, Carl Fredrickson ©Disney/Pixar. All Rights Reserved.



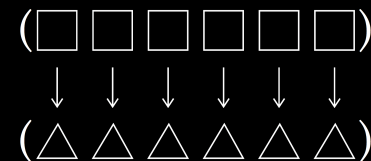


# Review

- Functions as Data
- Higher-Order Functions
- Useful HOFs (you can build your own!)

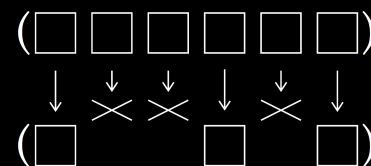
- **map** Reporter over List

- Report a new list, every element  $E$  of `List` becoming `Reporter(E)`



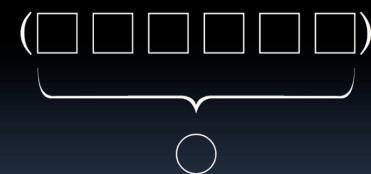
- **keep items such that** Predicate from List

- Report a new list, keeping only elements  $E$  of `List` if `Predicate(E)`



- **combine with** Reporter over List

- Combine all the elements of `List` with `Reporter(E)`
    - This is also known as “reduce”



- Acronym example

- **keep**  $\rightarrow$  **map**  $\rightarrow$  **combine**



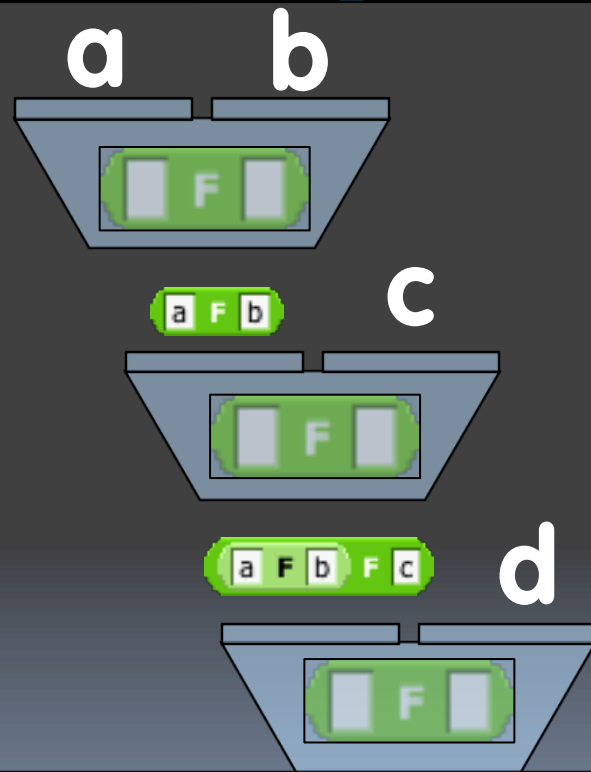


List

1	a
2	b
3	c
4	d

+ length: 4

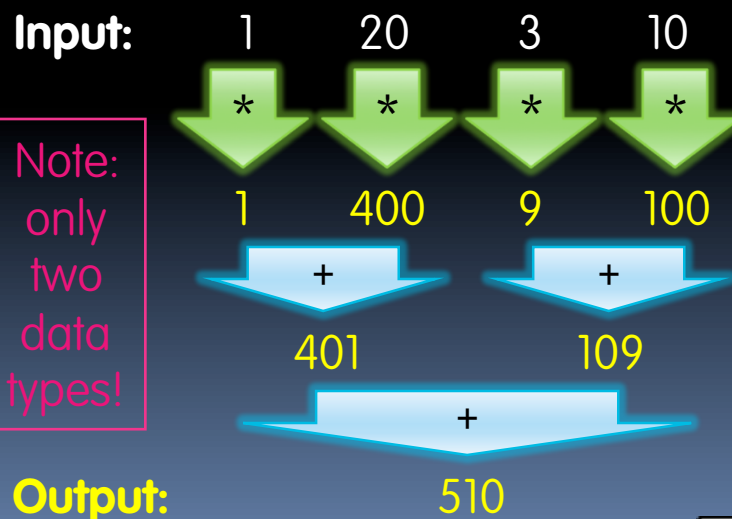
combine with Reporter over List





# Google's MapReduce Simplified

- We told you “the beauty of pure functional programming is that it’s easily parallelizable”
  - Do you see how you could parallelize this?
  - Reducer should be associative and commutative
- Imagine 10,000 machines ready to help you compute anything you could cast as a MapReduce problem!
  - This is the abstraction Google is famous for authoring
  - It hides LOTS of difficulty of writing parallel code!
  - The system takes care of load balancing, dead machines, etc.



Note: only two data types!





# MapReduce Advantages/Disadvantages

- **Now it's easy to program for many CPUs**
  - Communication management effectively gone
  - Fault tolerance, monitoring
    - machine failures, suddenly-slow machines, etc are handled
  - Can be much easier to design and program!
  - Can cascade several (many?) MapReduce tasks
- **But ... it might restrict solvable problems**
  - Might be hard to express problem in MapReduce
  - Data parallelism is key
    - Need to be able to break up a problem by data chunks
  - Full MapReduce is closed-source (to Google) C++
    - Hadoop is open-source Java-based rewrite







# What contributes to overhead the most?

---



- a) Dividing problem up
- b) Shipping it out / Getting it back
- c) Verifying the result
- d) Putting results together
- e) Depends on problem





# Summary

- Systems and networks enable and foster computational problem solving
- **MapReduce** is a great distributed computing abstraction
  - It removes the onus of worrying about load balancing, failed machines, data distribution from the programmer of the problem
  - (and puts it on the authors of the MapReduce framework)

