CS 262A:
Advanced Topics in Computer Systems

Ali Ghodsi and Ion Stoica

August 26, 2020

Course Information

Course website:

- https://ucbrise.github.io/cs262a-fall2020/
  - It is on Github so you can contribute content!

Schedule and reading list are subject to change

Who are we?

Ali Ghodsi

- Adjunct Professor, UC Berkeley (Systems, Databases)
- CEO, co-founder, Databricks
- Assistant Professor, KTH (2008-2009)
- OSS involvement: Apache Spark, Apache Mesos, Alluxio, Delta, MLFlow

Ion Stoica

- Professor, UC Berkeley (Systems, Networking)
- Co-founder: Databricks, Anyscale, Conviva
- OSS involvement: Apache Spark, Ray, Apache Mesos, Alluxio, Clipper

We have been working together since 2009!

Who are we?

Michael Whittaker

- TA
- PhD Student (Databases)
- Research focused on replication, consistency, and consensus algorithms for distributed systems

What is System Research about?

Manage resources:

- Memory, CPU, storage
- Data (database systems)

Provide abstractions to applications:

- File systems
- Processes, threads
- VM, containers
- Naming system
- …

This Class

Learn about systems research by

- Reading several seminal papers
- Doing it: work on an exciting project

Hopefully start next generation of impactful systems

Appreciate what is Good Research

Problem selection

Solution & research methodology

Presentation

What do you need to do?

Research oriented class project

- Groups of 2-3

Paper reading

Paper presentations

Research Project

Investigate new ideas and solutions in a class research project

- Define the problem
- Execute the research
- Write up and present your research

Ideally, best projects will become conference papers (e.g., OSDI/SOSP, NSDI, EuroSys)

Research Project: Steps

We'll distribute a list of projects

- You can either choose one or come up with your own

Pick your partner(s) and submit a one-page proposal describing:

- The problem you are solving
- Your plan of attack with milestones and dates
- Any special resources you may need

Poster session

Submit project report

Paper Reading

All students are required to read all papers

All students are required to submit “reviews” for each paper:

- Answer four short questions on google form (see next slide)
- We will send out how to signup for papers later today

Paper Reading: Key Questions

- What is the problem?
- What is the solution's main idea?
- Why did it succeed or failed?
- Does the paper (or do you) identify any fundamental/hard trade-offs?

Submissions: Will send out a google form for every paper that you need to fill in (will close it 10min before the class)

Distributed Shared Memory

Countless papers in 1990s:

- Very compelling abstraction
- Many hard challenges, so many researchers worked on it

Today

- Few systems using distributed shared memory, if any
  - Note: Comeback in the context of disaggregated memory?
- Message passing (e.g., MPI) or bulk synchronous processing (e.g., Spark) prevalent

Why did it fail?

Virtual Machine

Many papers in 1990s:

- Very compelling abstraction
- Many hard challenges, so many researchers worked on it

Today

- VMs everywhere
- Containers (e.g., docker) take this concept to the next level

Why did it succeed?

What are Hard/Fundamental Tradeoffs?

- Brewer's CAP conjecture: "Consistency, Availability, Partition-tolerance", you can have only 2/3 in a distributed system

- Tradeoff between latency and throughput for arbitrary updates in distributed systems
  - Batch request to increase throughput, but hurts latency

Paper Presentation

Each student must present at least one paper

You must submit slides for review one week before presentation for feedback

- Please use Google slides

Lecture Format (starting with 9/7)

Faculty will provide context of the topic (10-15min)

Next, papers presented by students

- Two presentations of 15min each, followed by 20min discussion

Grading Policy

20% Class Participation

- Answer questions, join discussion, and present papers

20% Paper reviews

60% Project

Grading

Project: 60%

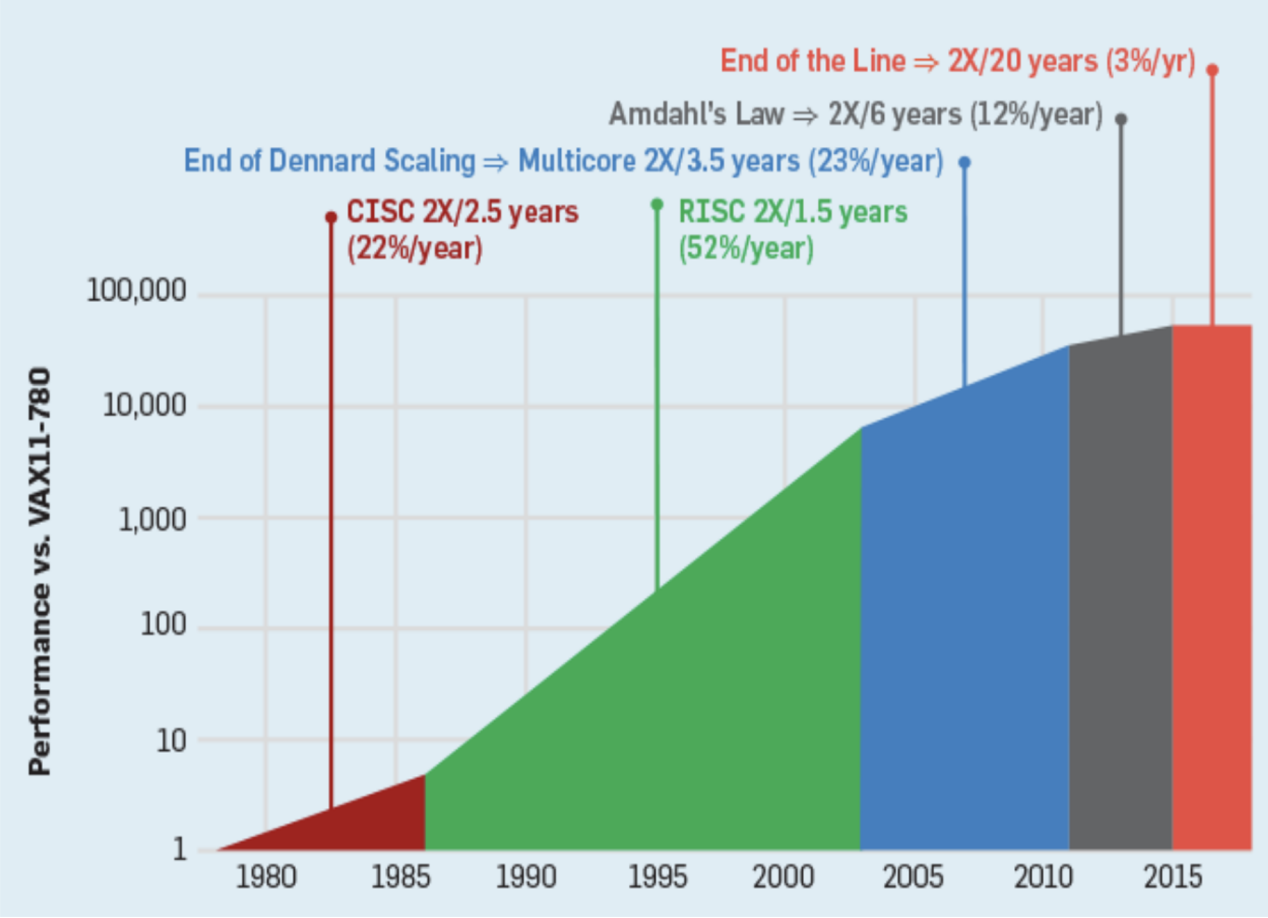
Paper blogs: 20%

Class participation: 20%

Exciting times in systems research

- Moore's law ending → many challenges
- Huge servers
- Specialized hardware
- Distributed computing
- Serverless computing
- …

Moore's Law ending



Huge Servers

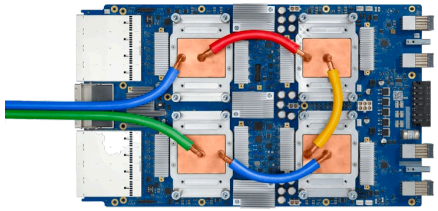
AWS high-memory instances

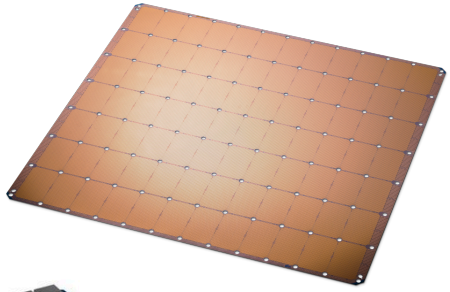
Name	Logical Processors*	RAM (GiB)	Network Perf (Gbps)	Dedicated EBS Bandwidth (Gbps)
u-6tb1.metal**	448	6144	100	38
u-9tb1.metal**	448	9216	100	38
u-12tb1.metal**	448	12288	100	38
u-18tb1.metal	448	18432	100	38
u-24tb1.metal	448	24576	100	38

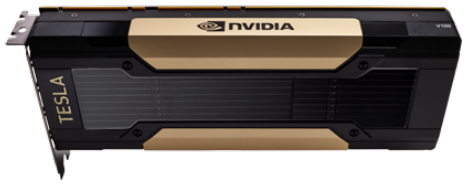
24TB !

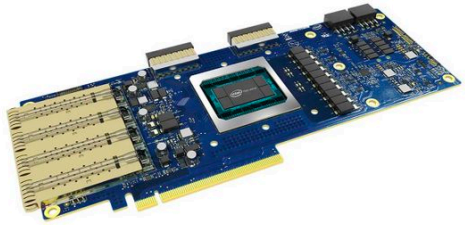
Specialized hardware

Trade generality for performance

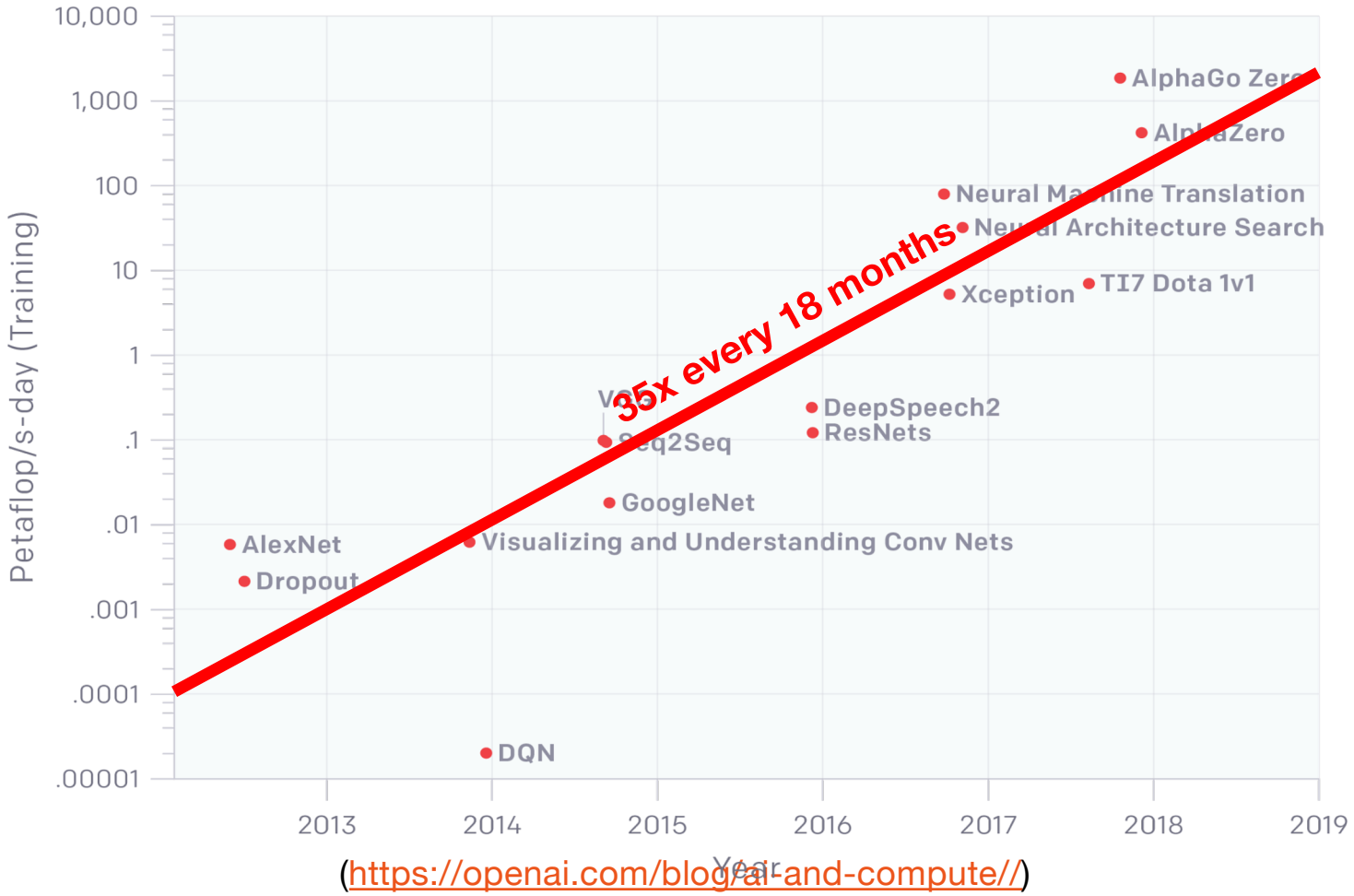






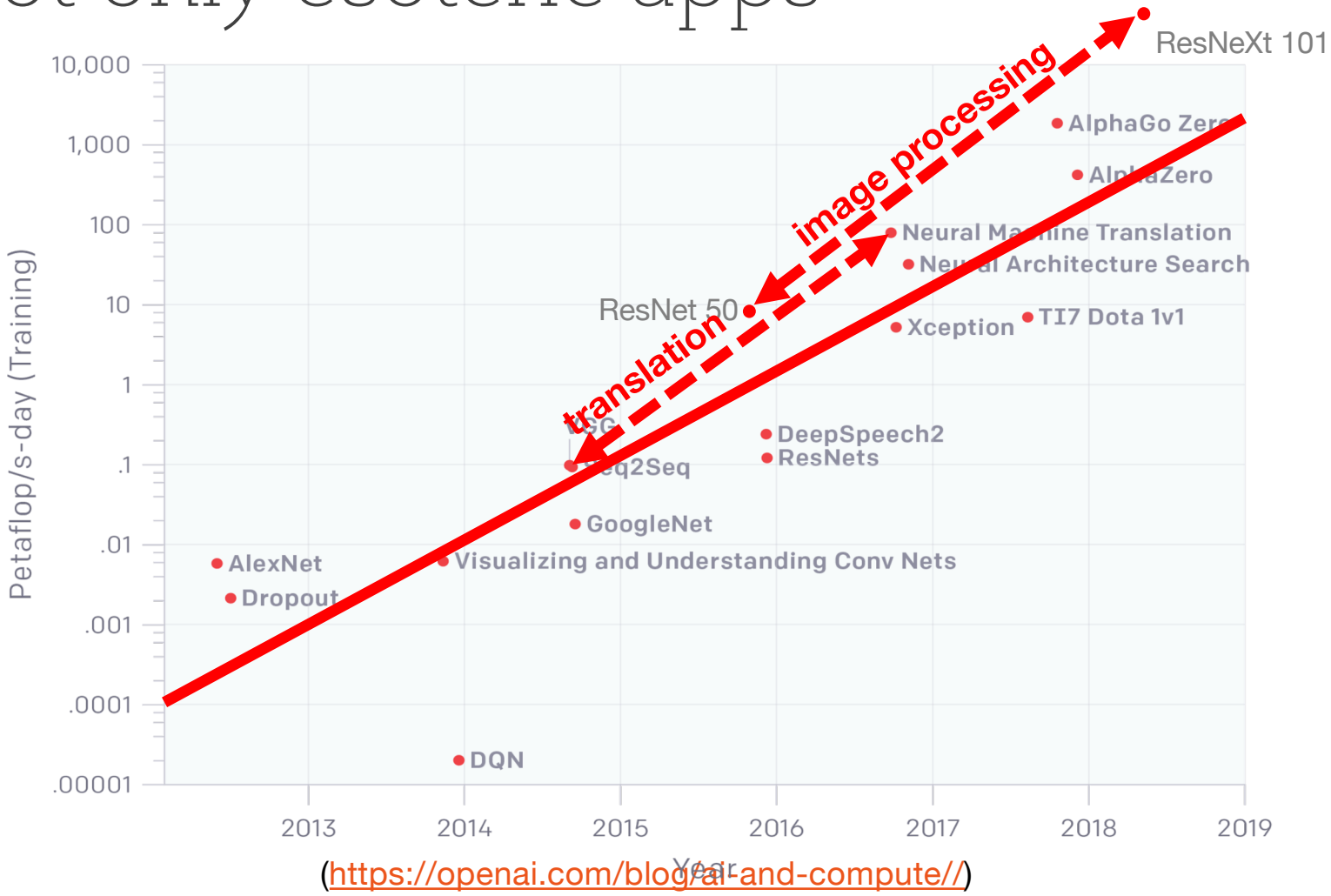


Deep Learning Demands



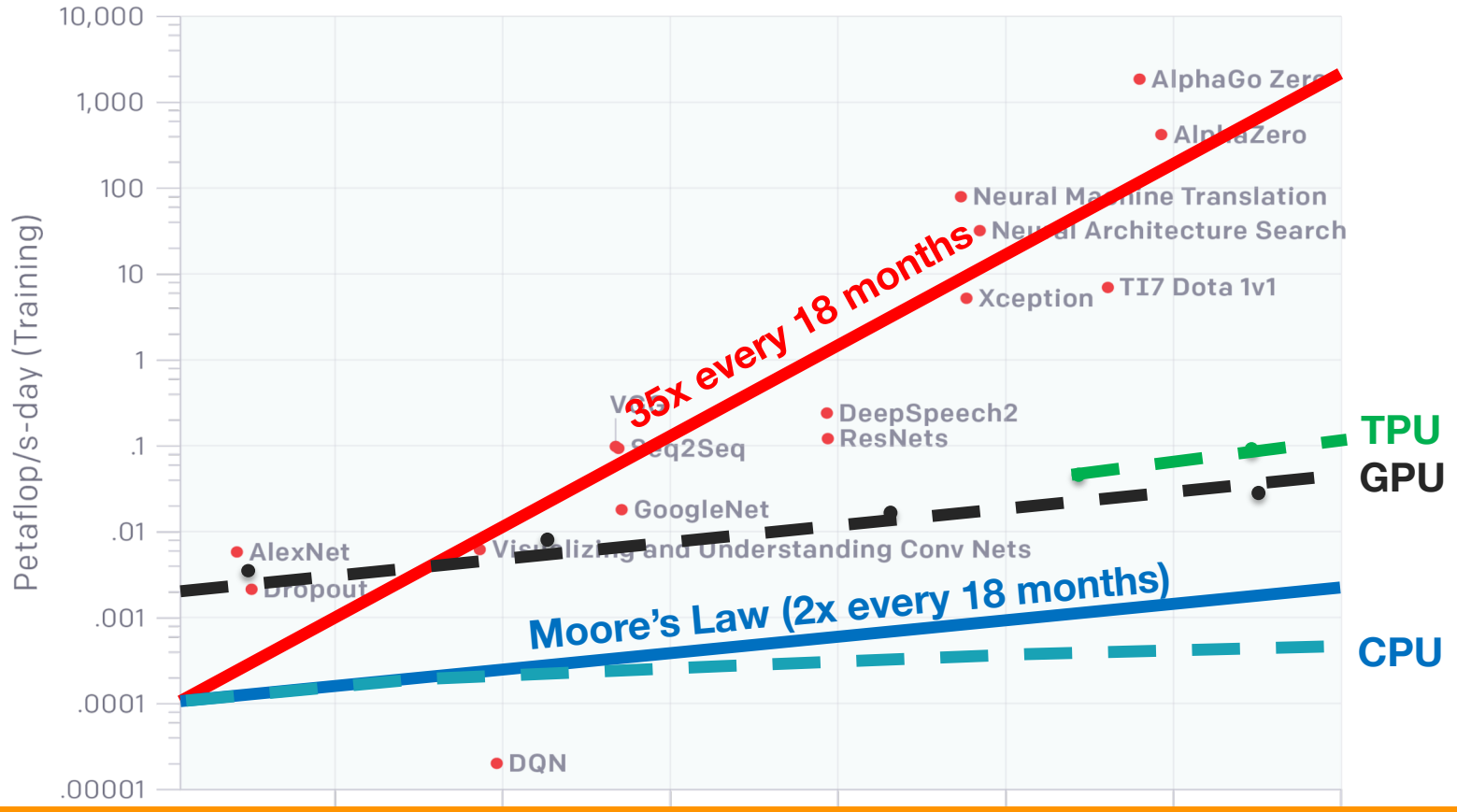
(https://openai.com/blog/ai-and-compute//)

Not only esoteric apps



(https://openai.com/blog/ai-and-compute//)

Specialized hardware not enough



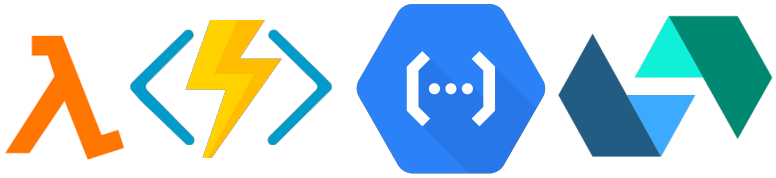
No way out but distributed!

Serverless

Make distributed applications easy

Serverless:

- Abstract away servers/clusters
- Pay-per-use (e.g., 100ms increments)



Natural progression: cloud agnostic!

What is a good problem to work on?

Impact: someone cares about solving the problem

- The more people care, the better

Priority: you are among the first to work on the problem

- Increases the chances of being first to solve this problem
- Caveat: This is just "nice to have". If you believe it's an important problem and you are passionate bout it, go for it!

Edge: You have an expertise (edge) to solve the problem

Incremental: can be partitioned into sub-problems

- Help you make rapid progress, iterate, better understand the problem

Impact (1/2)

Do things that have not been done before

- “It always seems impossible until is done” – Nelson Mandela
- Examples: zero-knowledge proof, beating chess/go world champion, …

Do things much better than before (e.g., 10x)

- Examples: GPUs, Distributed Hash Tables (routing efficiency), in-memory databases, CNNs, …

Impact (2/2)

Theory

- Improve space/time complexity of wide used algo (e.g., SGD)
- Extend existing techniques to new domains/environments, e.g., fair scheduler from network to CPU, distributed graph algorithms
- Prove new desirable properties for popular techniques (increase practitioners confidence in using them), e.g., show an allocation algo is strategy proof, show an algo is guaranteed to converge

Systems

- Support new workloads impractical before, e.g., big data processing
- Significantly improve efficiency/speed/cost, e.g., GPUs, dist algos
- “Democratize” technology, e.g., cloud computing, serverless, TensorFlow

Priority (1/2)

You experience problem first-hand

- One of the main reasons we are building systems/artifacts
  1. Build system solving a "need"
  2. Have people use it
  3. Understand things people want to do with your system, you haven't thought about
  4. Now you have a problem, you've "seen" first!

Take a bet on a new area and dive in

- Because it's new area, few people are working on it
- This is one thing Berkeley has been excelling at, e.g.,
  - Complexity theory, approximate algorithms, databases, networking, sensor networks, OSes (UNIX), NOW, Big Data, ML, …

Priority (2/2)

Talk with people from other areas to find new problems, e.g.,

- Spark motivated by Matei's trying to solve a ML postdoc's (Lester Mackey) problem

Berkeley's labs are set up to encourage you do exactly this!

> "*I notice that if you have the door to your office closed, you get more work done today and tomorrow, and you are more productive than most. But 10 years later somehow you don't know quite know what problems are worth working on; all the hard work you do is sort of tangential in importance. . . . I can say there is a pretty good correlation between those who work with the doors open and those who ultimately do important things*."
>
> – Richard Hamming, Turing Award Winner
>
> ("You and Your Research" : http://www.cs.virginia.edu/~robins/YouAndYourResearch.html)

Edge

If you do not have expertise to solve the problem, talk with people who do!

Again, labs like RISELab are set up to encourage you do exactly this!

Incrementality

Reach the first meaningful milestone fast

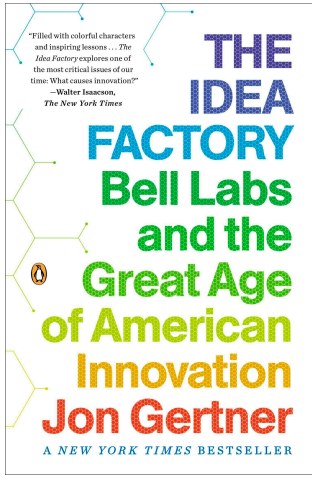
- Critical to get the project off the ground, maintain the excitement of the students and, if a system, start growing community
- Examples: Mesos, Spark provided an useful system within 1 year

Decompose project in a series of milestones providing increasing value

- Examples: Spark core → Shark (SQL) → Streaming → Mllib
- Developing a chess algorithm to first beat an amateur, then a master, then a grand master, and finally the world champion

Holly grail example: Bell Labs achieving its vision of "enabling any two people in the world communicate"

- Enable two people to talk by phone between Boston and NY
- Enable two people to talk by phone across US between NY and SF
- Enable two people to talk by phone across Atlantic between NY and London…. (see "The Idea Factory" book)



Questions to ask when picking a problem

What will happen if I solve this problem? Will anyone care? Will anything change in the way people do things?

- If you hope for a quantitative gain, simplify the problem and see improvement; if improvement not what you hopped for, reconsider

Who else is working on the problem? (Again, big caveat here; if you believe the problem is important an you can solve it go for it!)

Why me? Do I have any edge on solving this problem? Am I expert in some techniques I think it will help me solve it? Did I solve similar problems?

Is there a meaningful milestone I can get to fast?

Some problems I worked on

Problem	***Impact*** ***(problem formulation)***	***Priority***	***Edge***	***Incremental***	***Comments***
Chord	Need for decentralized (no liability), efficient P2P system	Among several groups working on it	Experts in distributed systems and algorithms	Followed by many refinements	At some point, most cited paper **13,600+ citations**

Some problems I worked on

Problem	***Impact*** ***(problem formulation)***	***Priority***	***Edge***	***Incremental***	***Comments***
Chord	Need for decentralized (no liability), efficient P2P system	Among several groups working on it	Experts in distributed systems and algorithms	Followed by many refinements	At some point, most cited paper **13,600+ citations**
Spark	Need for fast, general-purpose big data analytics	Among first to see ML use cases (AMPLab) and interactive queries (industry; Conviva, FB)	Already experts in big data (in academia); started working on this 3 years earlier	Core Spark (3,000 LoC); Shark, Streaming, ML libs	High impact in industry **3,940+ citations**

Some problems I/we worked on

Problem	***Impact*** ***(problem formulation)***	***Priority***	***Edge***	***Incremental***	***Comments***
Chord	Need for decentralized (no liability), efficient P2P system	Among several groups working on it	Experts in distributed systems and algorithms	Followed by many refinements	At some point, most cited paper **14K+ citations**
Spark	Need for fast, general-purpose big data analytics	Among first to see ML use cases (AMPLab) and interactive queries (industry; Conviva, FB)	Already experts in big data (in academia); started working on this 3 years earlier	Core Spark (3,000 LoC); Shark, Streaming, ML libs	High impact in industry **5K+ citations**
Dominant Resource Fairness (DRF)	Need for better utilization for multi resource types workloads	Among first to see the problem (Mesos)	Experts in scheduling algorithms	Followed by other papers in different areas	Intellectual impact (microeconomy); implemented in a few systems **1K+ citations**