Extracting Parallelism is key to performance.

Key goal of hardware, systems, and for more than a decade. The only way to get performance.

Old Slides from ~2017.
But these are main ideas, we'll see them at multiple scales.

Biased by my own work because I have slides and am lazy… not because I think it's best.

Message

Statistical algorithms *have* ***relaxed*** *notions of correctness leads to* ***new opportunities*** *for:*

- Algorithms,
- Systems, and
- Hardware.

The Key Balance

Key Issue: Balance
Statistical versus Hardware Efficiency.

- **Statistical efficiency** how many steps you take
- **Hardware efficiency** how efficiently you take each of those steps

Ce Zhang, CR *DimmWitted: A Study of Main-Memory Statistical Analytics.* VLDB14.

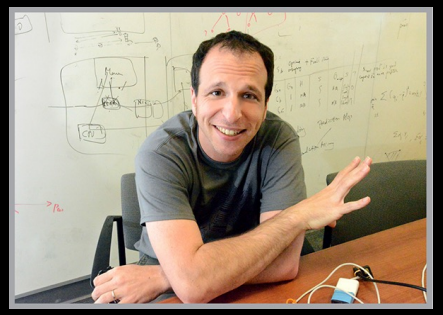
Three driving trends in **hardware**

(1) **Lots of smaller cores**,
(2) **Non-Uniform Memory (NUMA)**, and
(3) **Single-Instruction Multiple Data (SIMD)** (and SIMT)

Approximation allows **major performance improvements**.

Trend 1: **Many different Cores**

Steve Wright

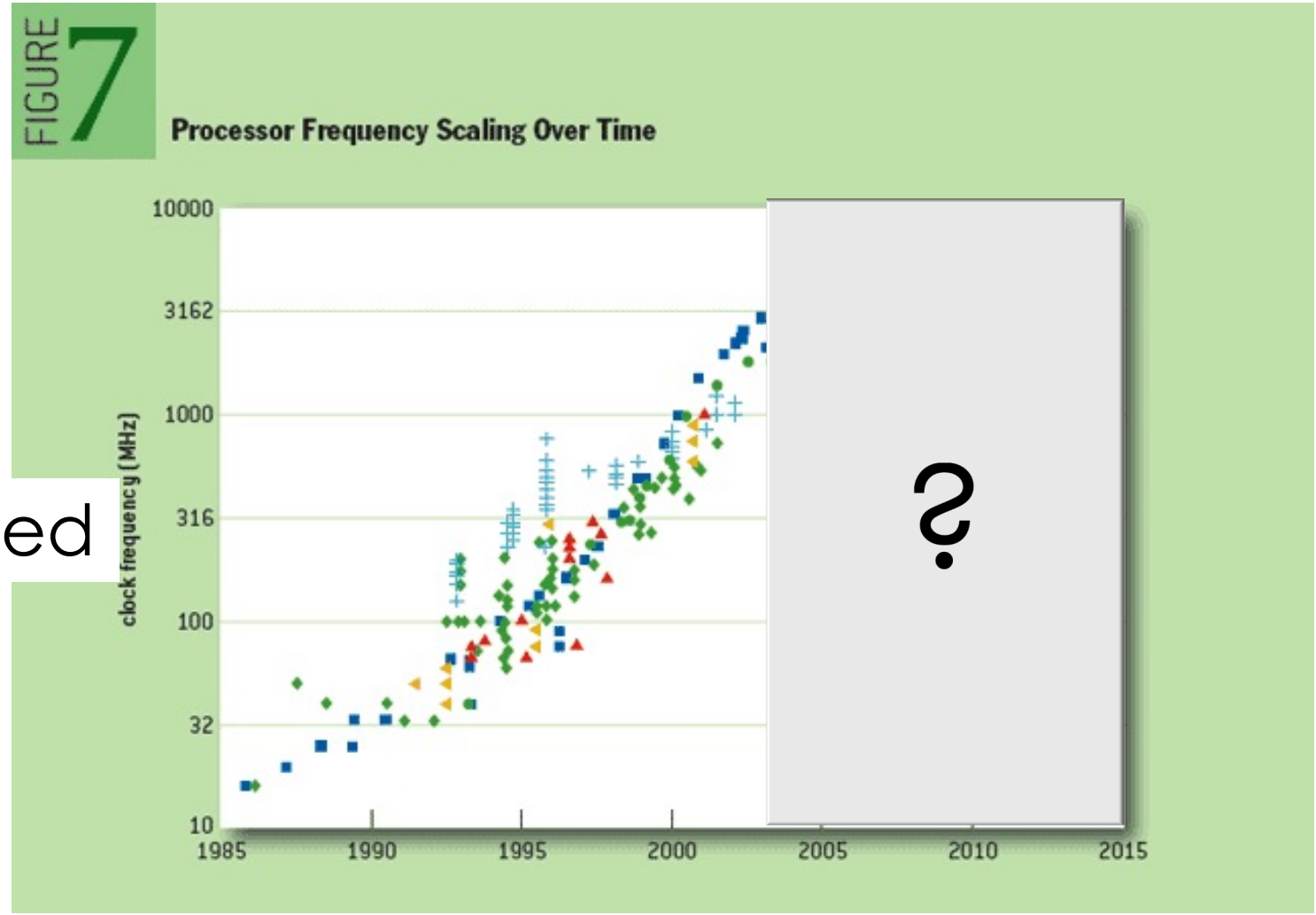


Ben Recht

Feng Niu

Single Cores are not getting faster.

Speed



Chips now contain many cores, so throughput is increasing… but need to rewrite algos!

Statistical Analytics Crash Course

Staggering amount of machine learning/stats can be written as:

$$\min_x \sum_{i=1}^{N} f(x, y_i)$$

N (number of y_is, data) typically in the billions
Ex: Classification, Recommendation, Deep Learning.

De facto iteration to solve large-scale problems: **SGD**.

$$x^{k+1} = x^k - \alpha N \nabla f(x^k, y_j)$$

Select one term, j, and estimate gradient.

Billions of tiny iterations.

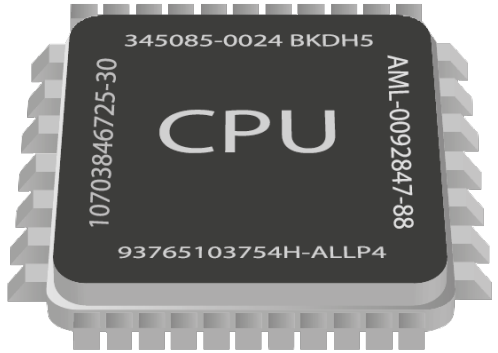
Multicore: Independent Case

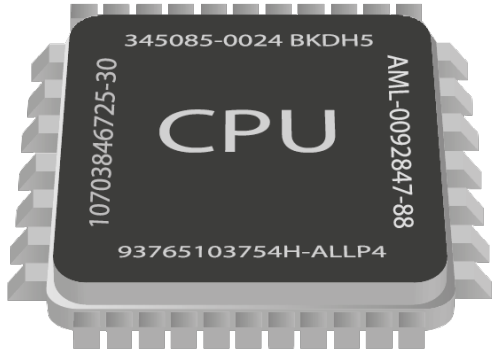
Job 1

Job 2

Job 3

Job 4





Jobs with little communication, 2 cores executes twice as faster!

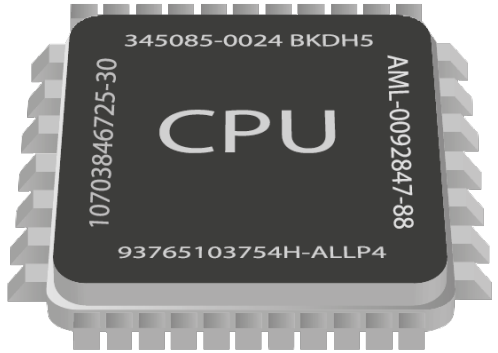
Multicore: Dependent Case

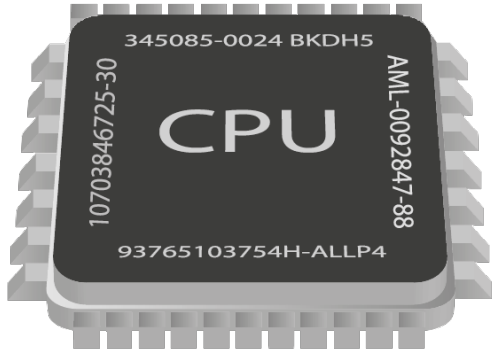
Job 1

Job 2

Job 3

Job 4





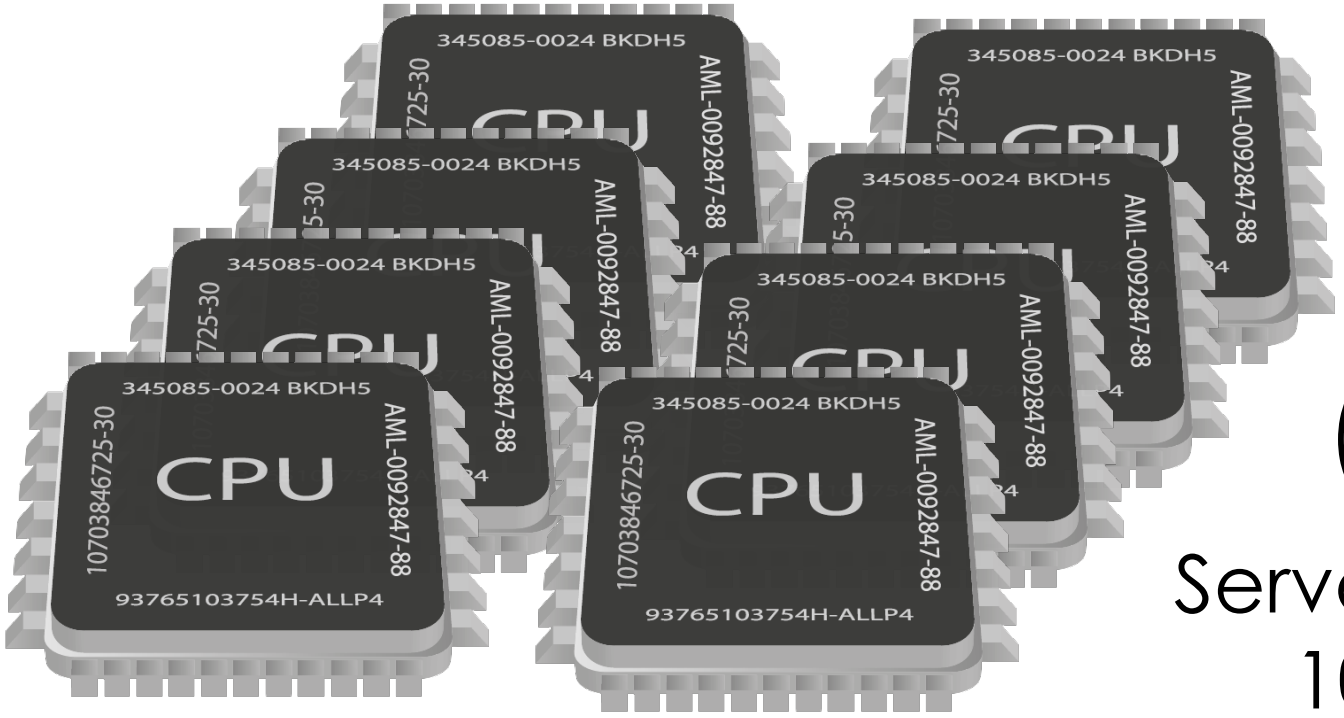
Is it my turn?

Protocol for "whose turn," called **locking**, takes 100 cycles.

Communication scales quadratically

Suppose it takes 1 second to communicate with 2 cores.

4 cores takes 4 second



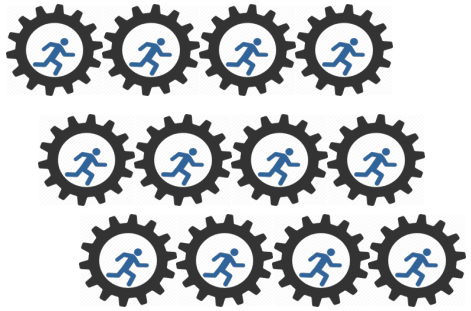
8 cores takes 16 seconds.

k cores takes $(k/2)^2$ seconds.

Server may have 100+ cores

The key algorithm in machine learning

The core algorithm of modern learning called is **Stochastic Gradient Descent (SGD)**



SGD consists of **BILLIONS** of tiny jobs!

Implemented in a classical way (locking) SGD actually gets *slower* with more cores

So what can we do?

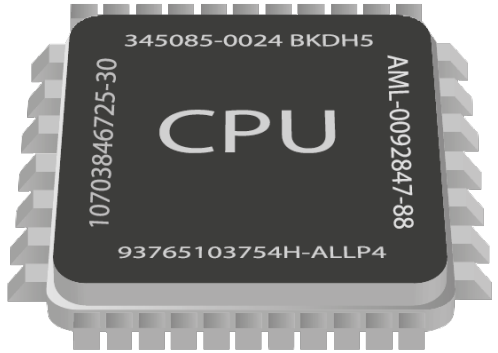
Multicore: Hogwild! Case

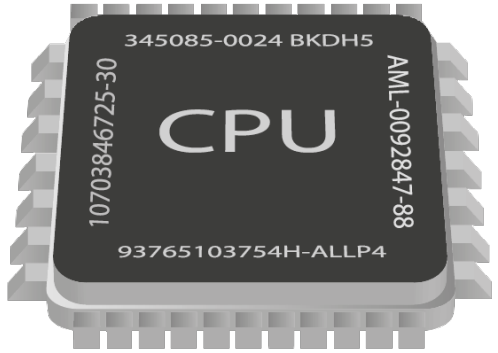

Job 1


Job 2


Job 3


Job 4





Is it my turn? Yes!

Ignore the locks!

How do we run SGD in Parallel?

Just ignore the locking protocol…
As we say, go **Hogwild**!

*This is computer science **heresy!***

Theorem (roughly, NIPS11): If we do ***no locking***, SGD converges to *correct* answer—*at essentially the same rate*!

Hogwild! [Niu, Recht, Ré, Wright NIPS11]
AsySCD [Liu, Wright et al. ICML14, JMLR14]
Buckwild! [De Sa, Olukotun, Ré NIPS15]

Cortana: Microsoft's Digital Assistant

WIRED

AI breakthrough: Microsoft's 'Project Adam' identifies dog breeds, points to future of machine learning

All web companies have similar: image rec, voice, mobile, search, etc.

"...using a technology called, of all things, ***Hogwild!"***

http://www.wired.com/2014/07/microsoft-adam/
http://www.geekwire.com/2014/artificial-intelligence-breakthrough-microsofts-project-adam-identifies-dog-breeds/

A larger trend?

PYTORCH

TensorFlow

Caffe2

Relaxing **consistency** to be **architecturally aware** can be a big performance win.

Microsoft

Google

ORACLE®

MADlib

Asynchrony in Deep Learning

Image: Google Research Blog

Sergül Aydöre
@sergulaydore

Follow

Heard at #icml2016: "SGD is so robust that your bugs in your SGD implementation will work as regularizers."

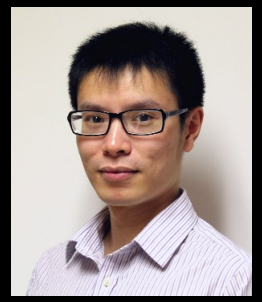
RETWEETS 77 LIKES 92

2:52 PM - 23 Jun 2016

A regularizer is a (sane) statistical penalty…
Bugs in your implementation are not helpful

Trend 2: **NUMA**
Non-Uniform Memory Access

Steve Wright

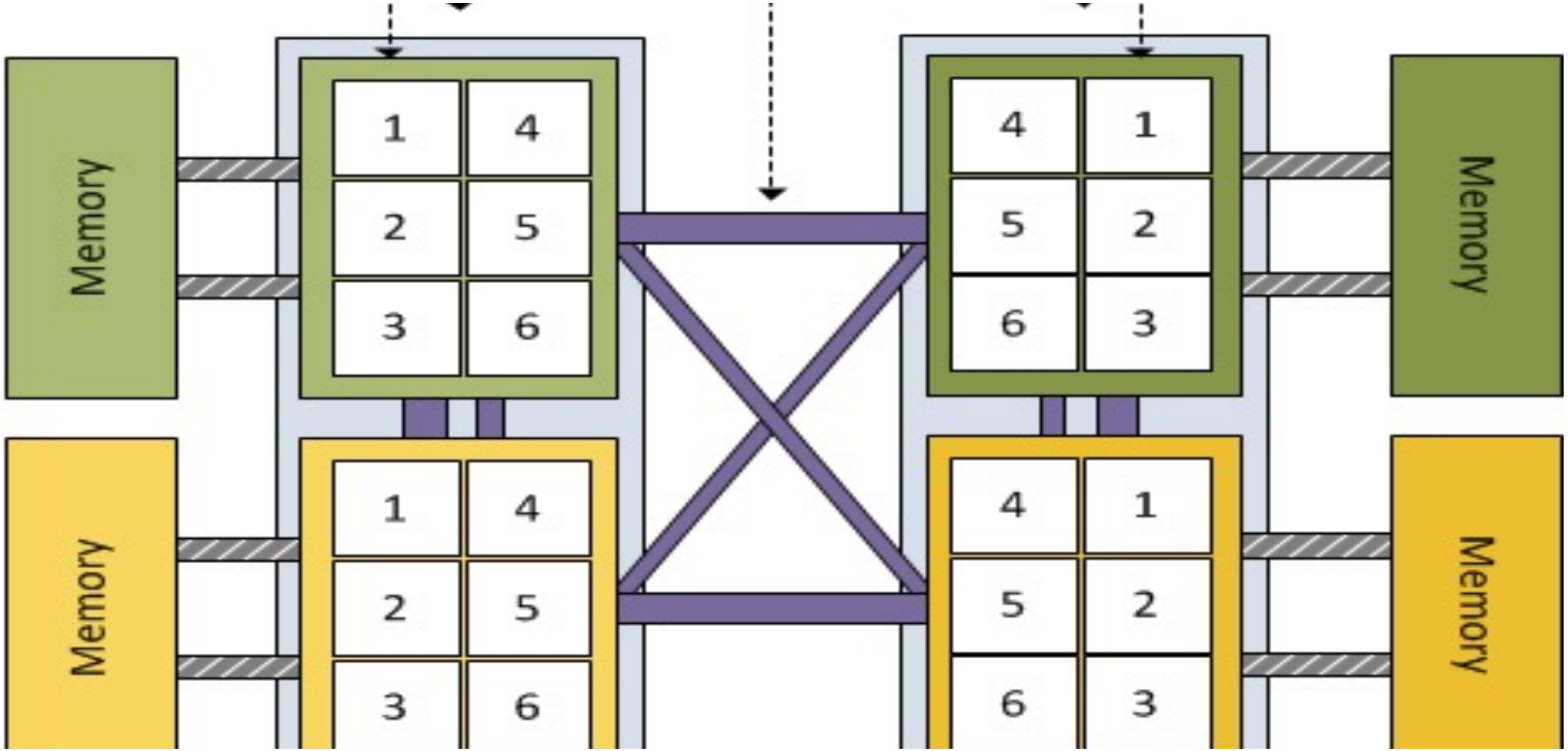


Ji Liu

Ce Zhang

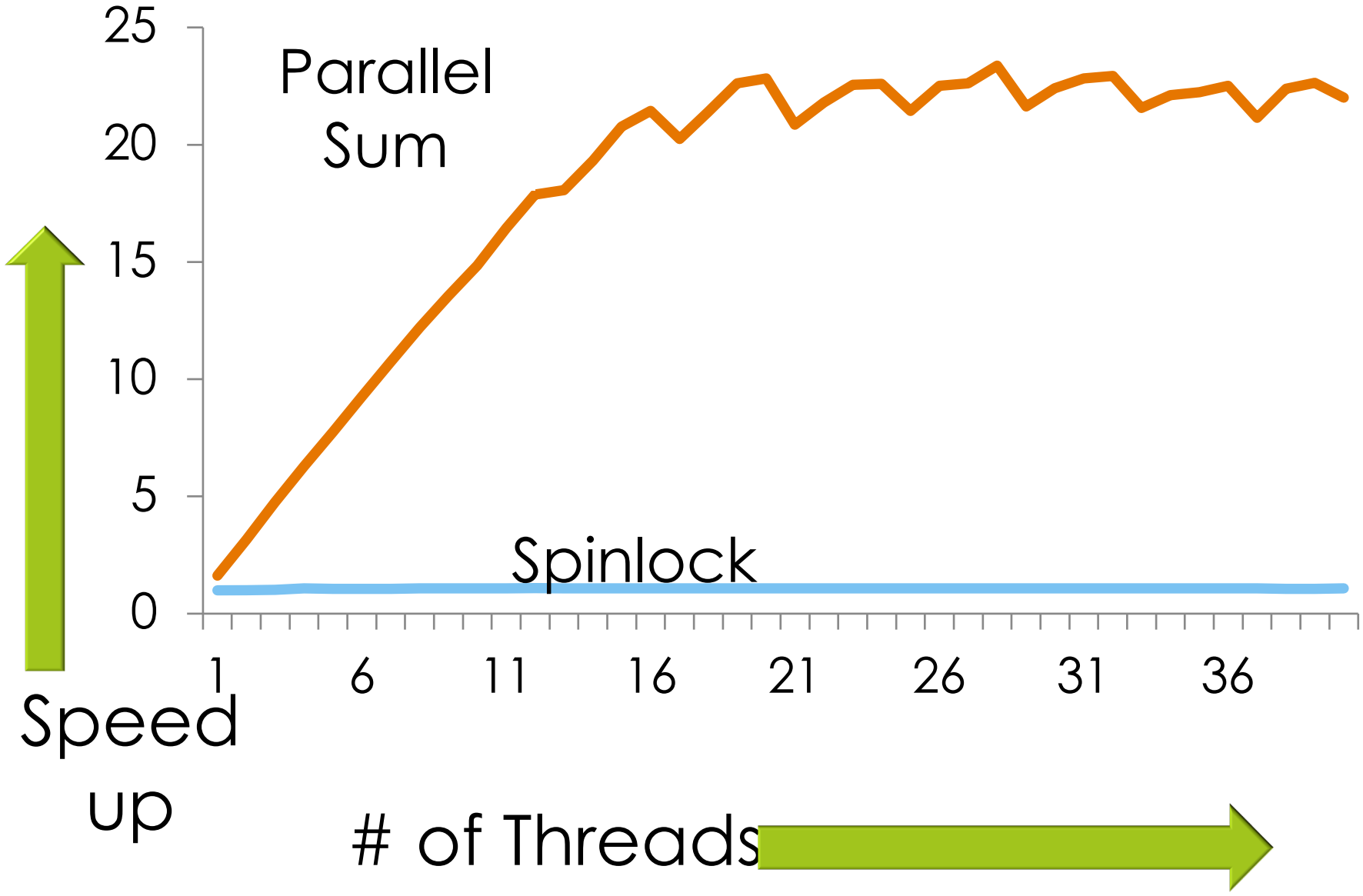
Krishna Sridhar

A view inside a box... (more later

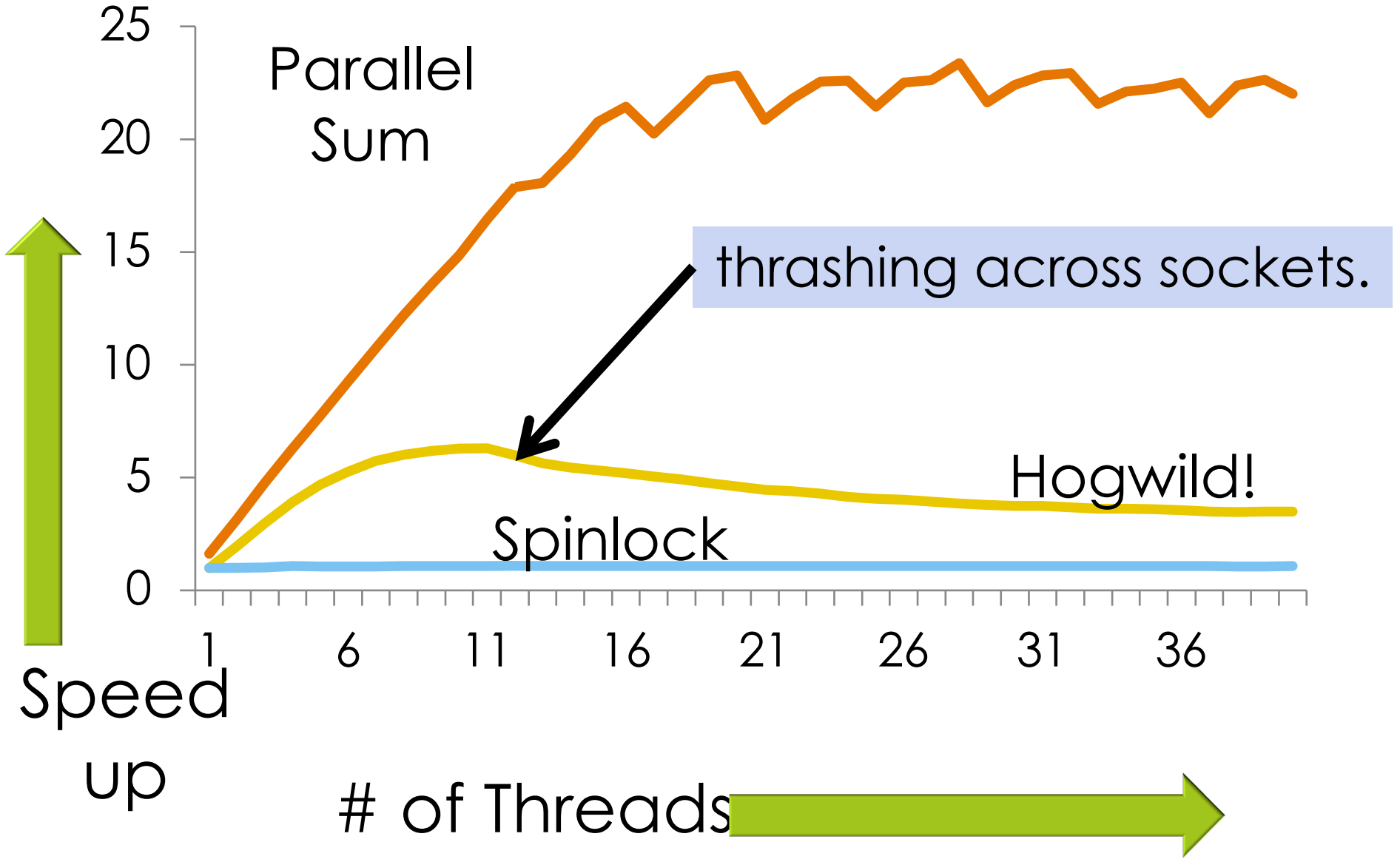


Modern version: Thousands of cores with close by memory
(Called high-bandwidth memory, called HBM)

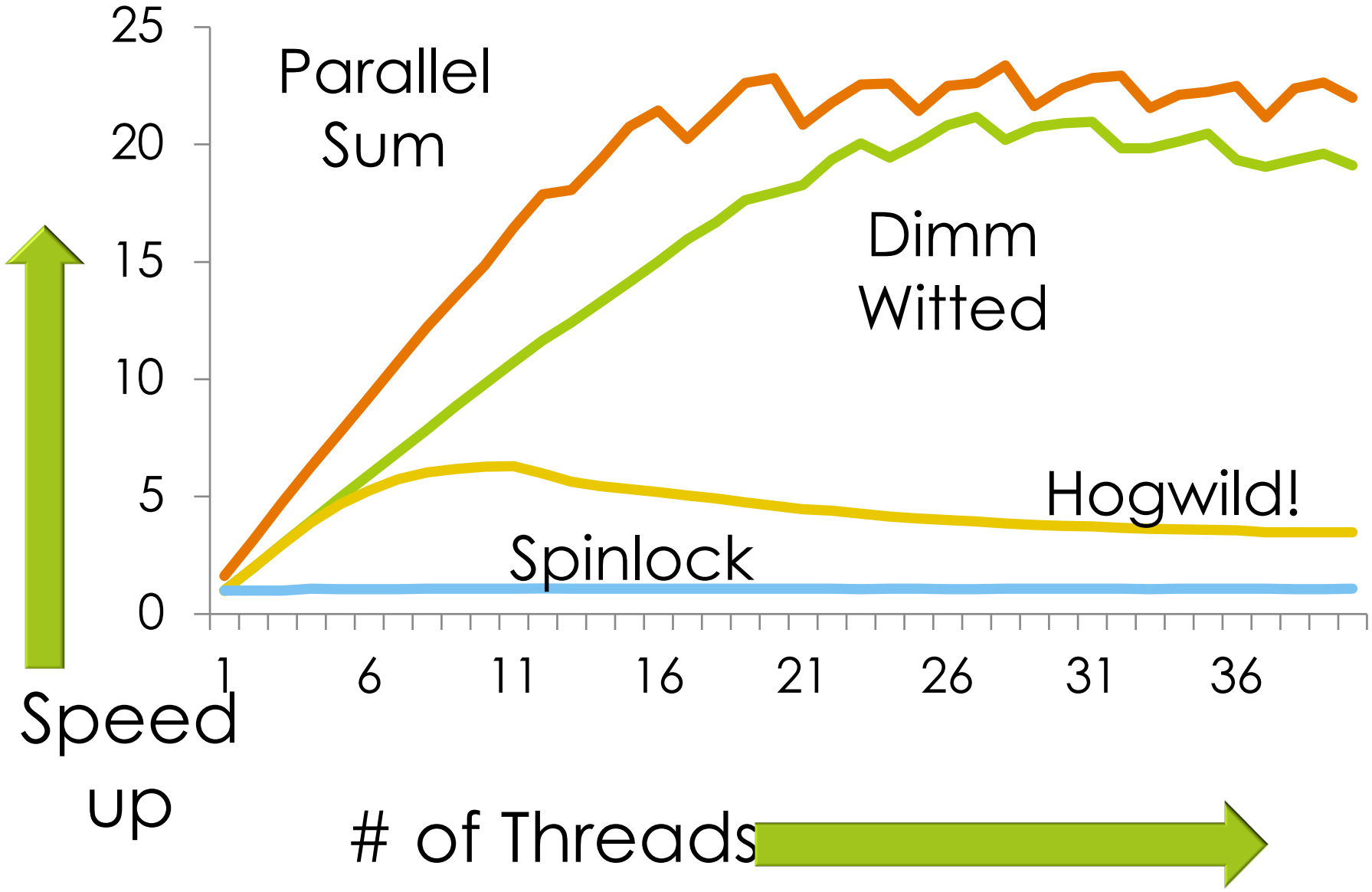
One Example: Quadratic Programming with Orthant Constraints (on cpu, same tradeoffs)



One Example.

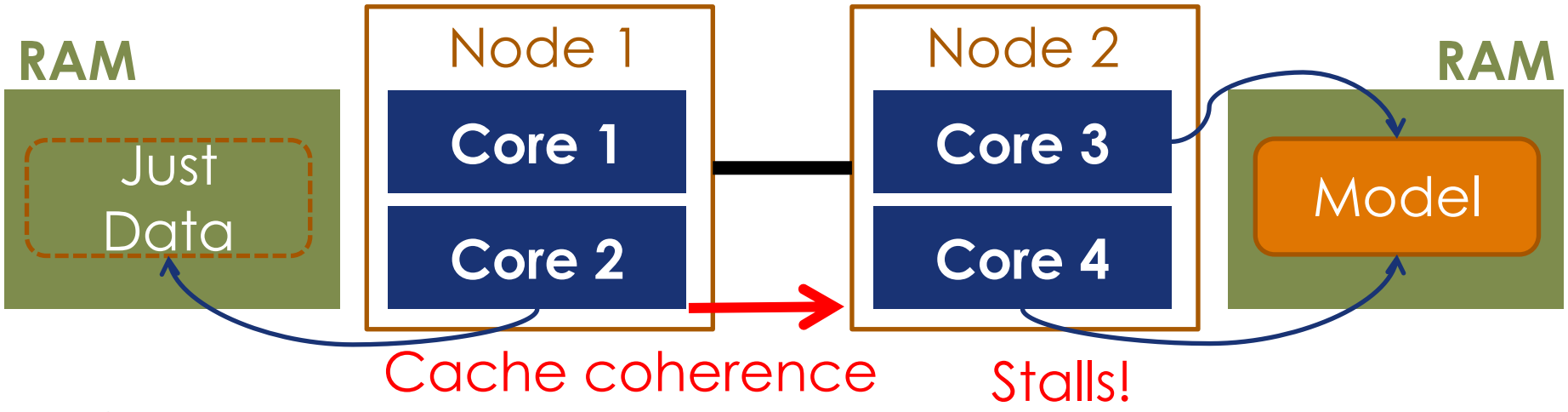

25
20
15
10
5
0
1
6
11
16
21
26
31
36
Parallel Sum
thrashing across sockets.
Hogwild!
Spinlock
Speed up
# of Threads

What about multiple sockets?

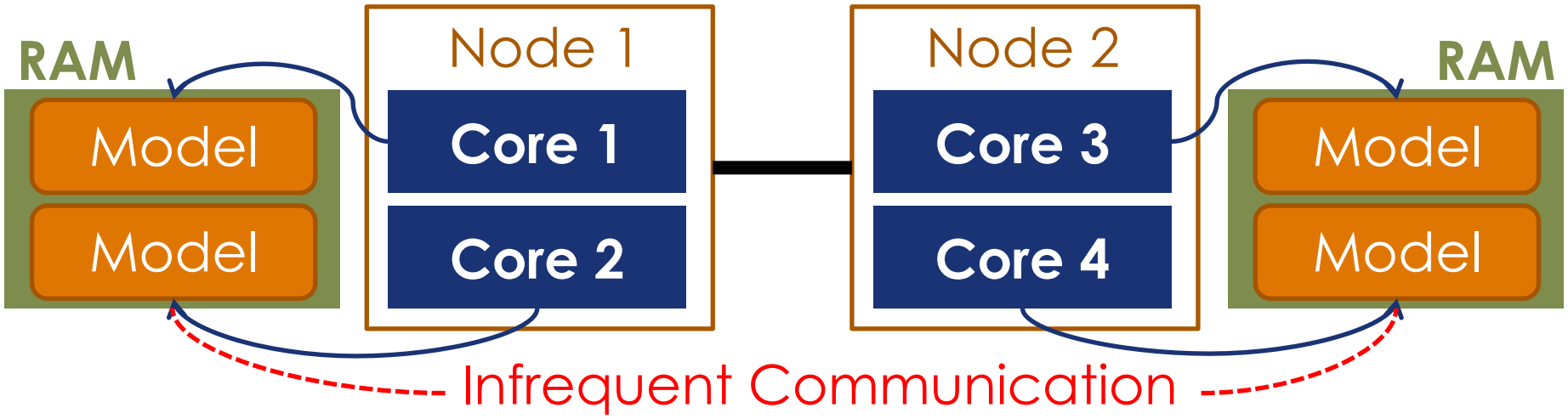


Model Replication

PerMachine (Hogwild!)



PerCore

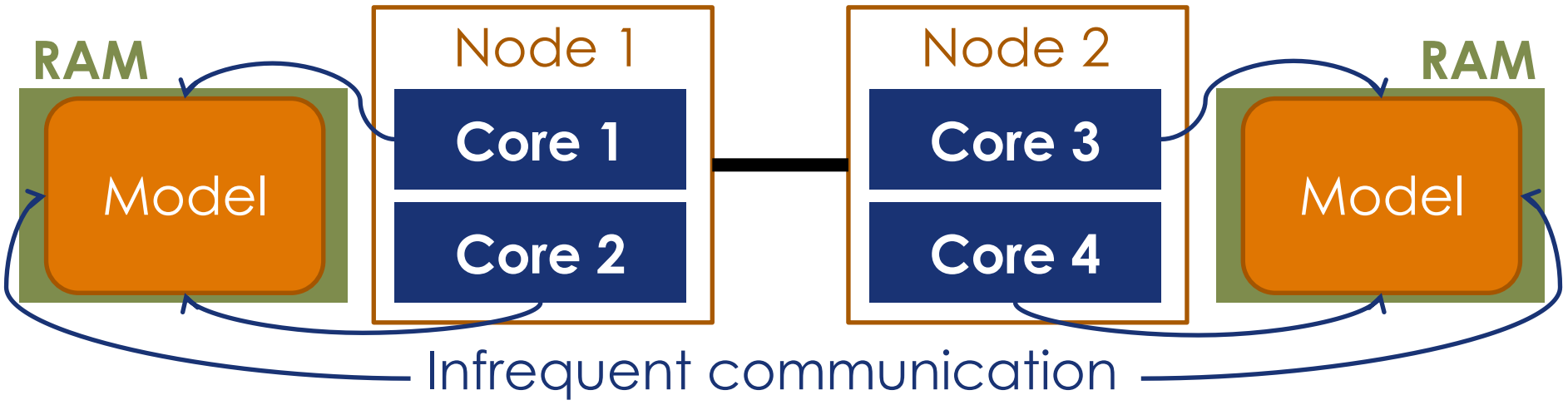


Model Replication

	Statistical Efficiency	Hardware Efficiency
Hogwild!	High	Low
PerCore	Low	High

In between both **Hogwild!** and **PerCore**?

PerNode



Statistical versus Hardware Efficiency

Relaxing consistency results in new tradeoffs.

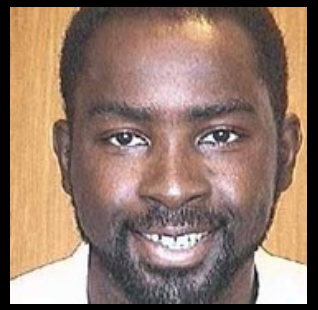
1. Access methods
   - {Row, Column, Row-col}
2. Model Replication
   - {Core, Node, Machine}
3. Data Replication
   - {Full, Importance, Shard}

Can be 100x faster than classical choices

DimmWitted: A Study of Main-Memory Statistical Analytics. VLDB14.

Trend 3: Single Instruction Multiple Data (**SIMD**)

Chris De Sa



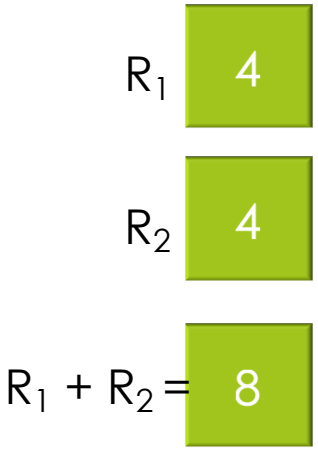
Kunle Olukotun

Modern processors offer **fine-grained** parallelism. [NIPS15]

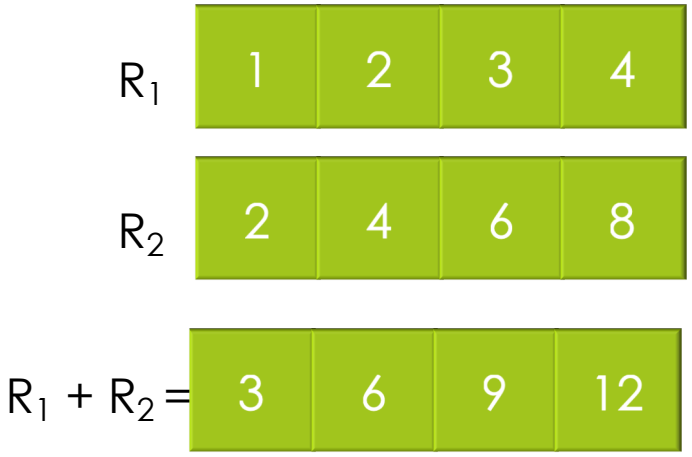
SIMD Processing: Fine-grained parallelism

Single instruction multiple data (SIMD)

Standard Addition (Two registers)



SIMD Addition (4 way)



Same operation on
multiple data points in parallel

SIMD: Doubling again!

SIMD bandwidth has **doubled each** of the last four generations.

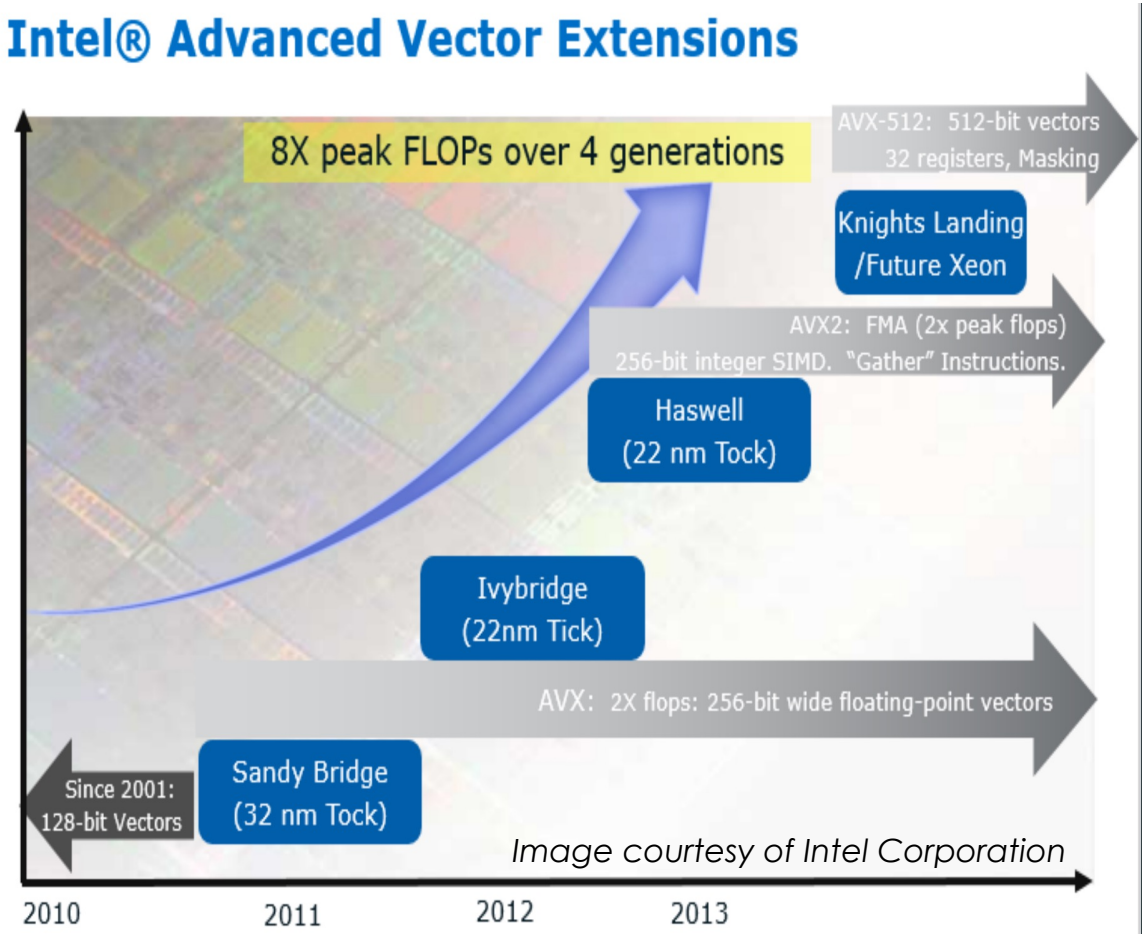
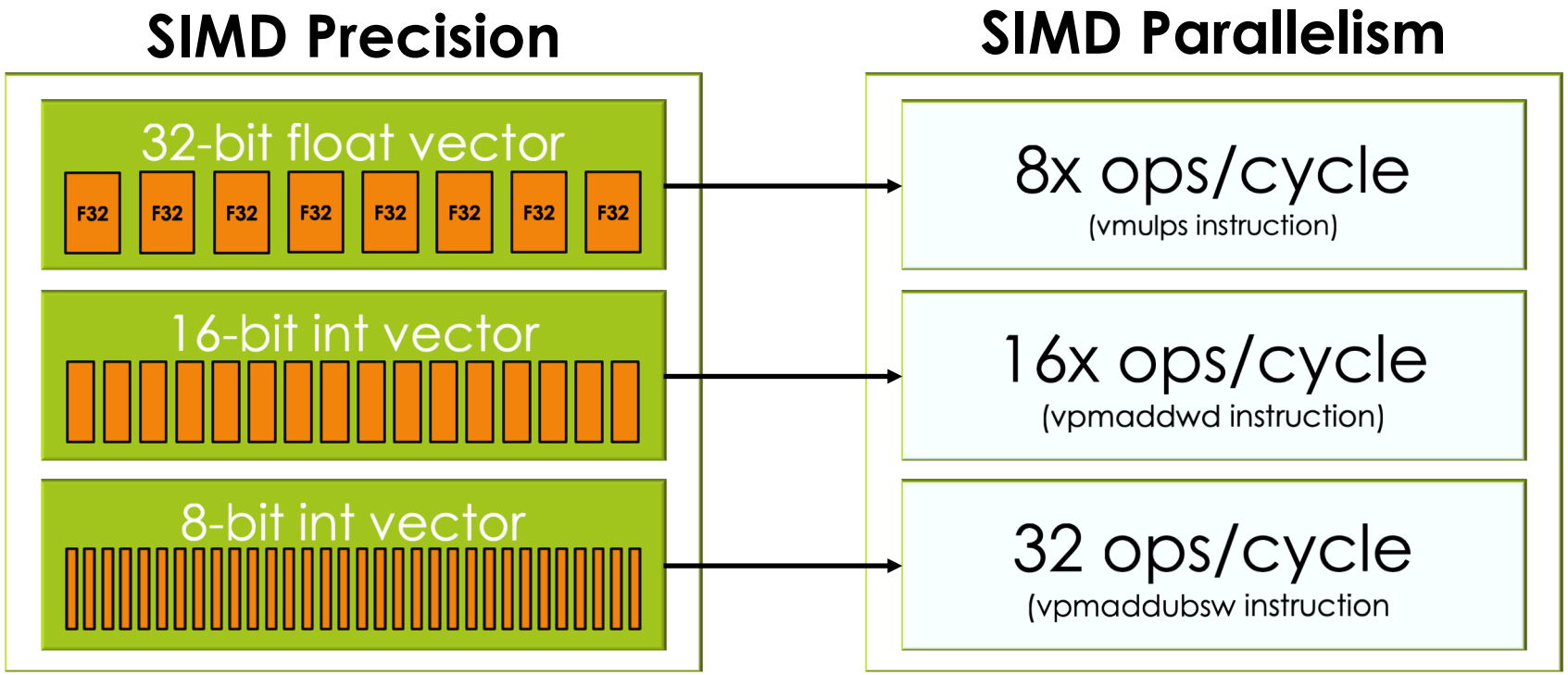


Image courtesy of Intel Corporation

Good old days of Moore's Law! …

If we can take advantage of fine-grained parallelism

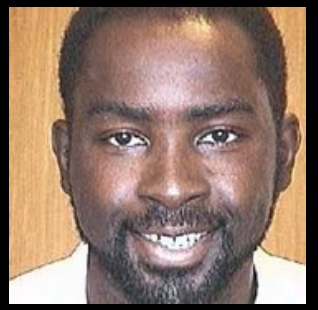
Precision vs. Parallelism



Tradeoff between precision & parallelism

A hardware model for precision [ISCA17]

Chris De Sa



Kunle Olukotun

Four Classes of Numbers

- **D**ataset numbers
  - used to store the immutable input data
- **M**odel numbers
  - used to represent the vector we are updating
- **G**radient numbers
  - used as intermediates in gradient computations
- **C**ommunication numbers
  - used to communicate among parallel workers

Quantize classes independently

- Using low-precision for different number classes has **different effects on performance**.
  - e.g. quantizing the **gradient numbers** improves compute throughput, but has little effect on memory

- Existing work often quantizes some classes, but **doesn't consider the others**.

The **DMGC** Model

- Idea: associate each implementation with a **DMGC signature** that displays its precision for all four number classes
  - Lets us **classify previous work** and future systems

$$D^8 M^{16} G^{32f} C^{16}$$

- D^8: The algorithm uses 8-bit numbers to store the dataset.
- M^{16}: It uses 16-bit numbers for the model.
- G^{32f}: It computes gradients as 32-bit floats.
- C^{16}: It communicates among workers with 16-bit numbers.

Be warned:
Your learning **parameters** depend on the hardware and those numbers.
(e.g. momentum and delay are connected)

Ioannis Mitliagkas

Jian Zhang

What shook my belief in progress through optimization…

- Turns out Optimization is a **leaky** abstraction for deep learning.
  - There are approaches that cause the loss to go down more slowly (worse optimization) but generalizing better (better test performance).

- Happy to give examples if you ask, so many out there it's bizarre….

- This is so much more interesting than it should be!