Algorithmic Frontiers of Modern Massively Parallel Computation

Introduction

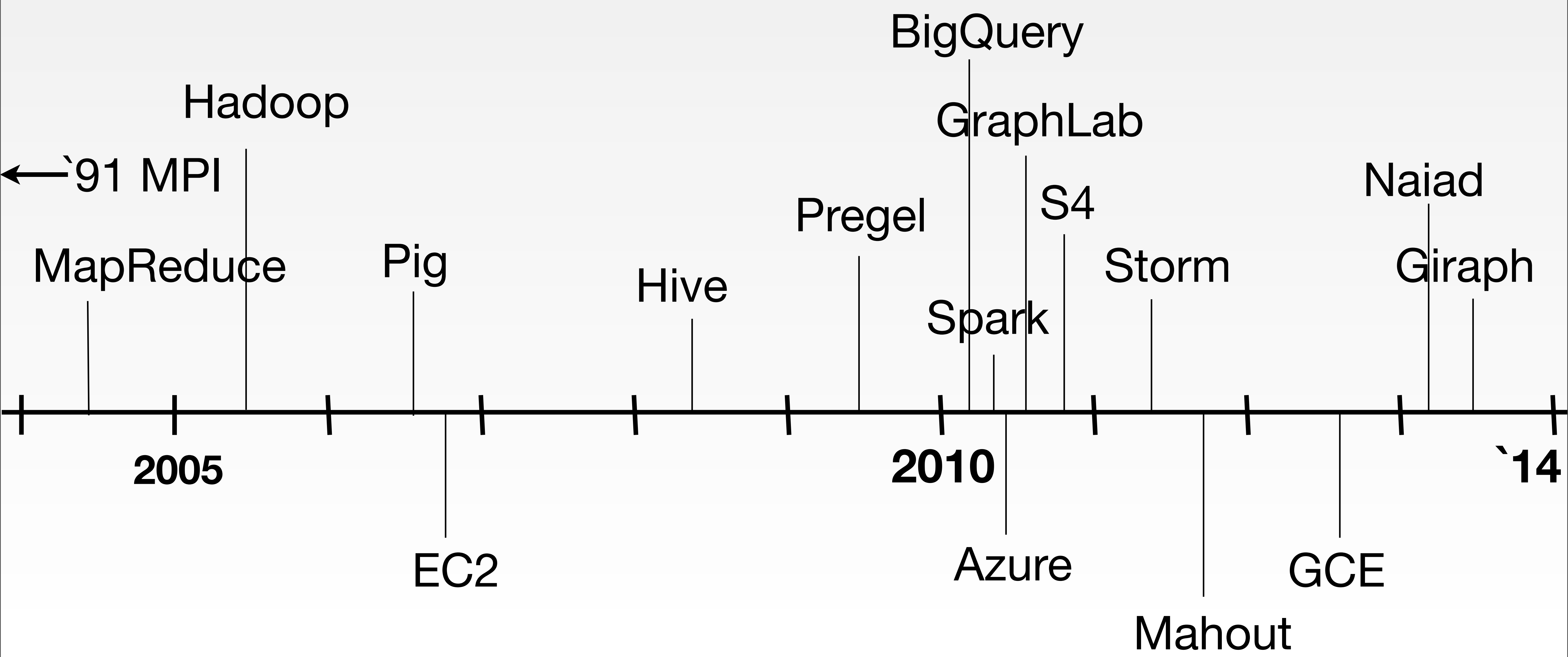
Ashish Goel, Sergei Vassilvitskii, Grigory Yaroslavtsev

June 14, 2015

Schedule

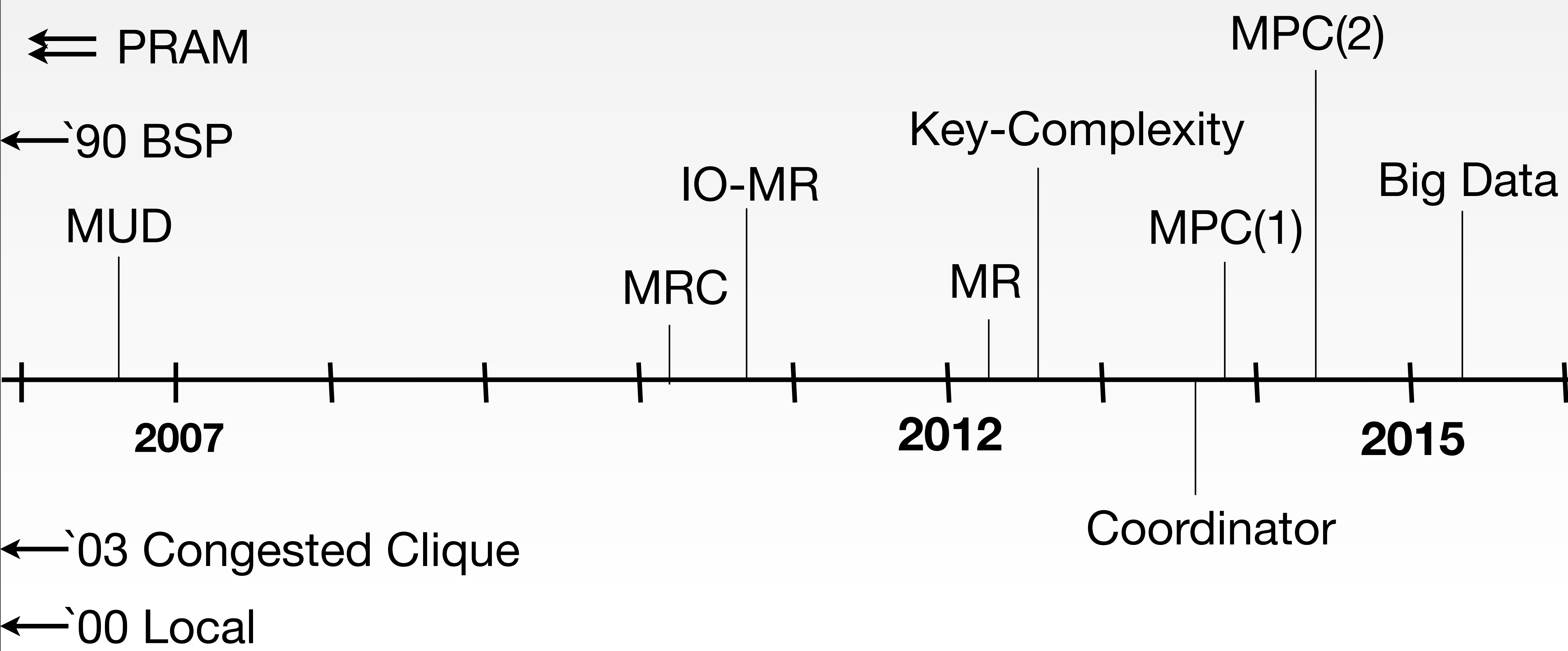
9:00 - 9:30	Introduction
9:30 - 10:15	Distributed Machine Learning (Nina Balcan)
10:15 - 11:00	Randomized Composable Coresets (Vahab Mirrokni)
11:00 - 11:30	Coffee Break
11:30 - 12:15	Algorithms for Graphs on V. Large Number of Nodes (Krzysztof Onak)
12:15 - 2:15	Lunch (on your own)
2:15 - 3:00	Massively Parallel Communication and Query Evaluation (Paul Beame)
3:00 - 3:30	Graph Clustering in a few Rounds (Ravi Kumar)
3:30 - 4:00	Coffee Break
4:00 - 4:45	Sample & Prune: For Submodular Optimization (Ben Moseley)
4:45 - 5:00	Conclusion & Discussion

Modern Parallelism (Practice)



*All dates approximate

Modern Parallelism (Theory)



* Plus Streaming, External Memory, and others

Bird’s Eye View

– 0. Input is partitioned across many machines

Bird's Eye View

- 0. Input is partitioned across many machines

Computation proceeds in synchronous rounds. In every round, every machine:

- 1. Receives data
- 2. Does local computation on the data it has
- 3. Sends data out to others

Bird's Eye View

- 0. Input is partitioned across many machines

Computation proceeds in synchronous rounds. In every round, every machine:

- 1. Receives data
- 2. Does local computation on the data it has
- 3. Sends data out to others

Success Measures:

- Number of Rounds
- Total work, speedup
- Communication

Devil in the Details

0. Data partitioned across machines
   - Either randomly or arbitrarily
   - How many machines?
   - How much slack in the system?

Devil in the Details

0. Data partitioned across machines

1. Receive Data

- How much data can be received?
- Bounds on data received per link (from each machine) or in total.
- Often called ‘memory,’ or ‘space.’
- Denoted by $M, m, \mu, s, n/p^{1-\epsilon}$

- Has emerged as an important parameter.
- Lower and upper bounds with this as a parameter

Devil in the Details

0. Data partitioned across machines
1. Receive Data
2. Do local processing
   - Relatively uncontroversial

Devil in the Details

0. Data partitioned across machines
1. Receive Data
2. Do local processing
3. Send data to others
   - How much data to send? Limitations per link? per machine? For the whole system?
   - Which machines to send it to? Any? Limited topology?

Devil in the Details

0. Data partitioned across machines
1. Receive Data
2. Do local processing
3. Send data to others

Different parameter settings lead to different models.

- Receive $\tilde{O}(1)$, poly machines, all connected: PRAM
- Receive, send unbounded, specific network topology: LOCAL
- Receive $\tilde{O}(1)$, send $\tilde{O}(1)$, n machines, specific topology: CONGEST
- Receive $s = n/p^{1-\epsilon}$, p machines, all connected: MPC(1)
- Receive $s = n^{1-\epsilon}$, $n^{1-\epsilon}$ machines, all connected: MRC
- ...

Details: Success Metrics

Number of Rounds:

- Well established
- Few (if any?) trade-offs on number of rounds vs. computation per round

Work Efficiency

- Important !
- See “Scalability! But at What COST? [McSherry, Isard, Murray `15]

Communication

- Matrix transpose -- linear communication yet very efficient
- Care more about skew, limited by input size

Consensus Emerging:

Parameters:

- Problem size : n
- Per machine, per round input size : s

Metric:

- Number of rounds: $r(s,n)$
- Ideal: $O(1)$ - e.g. group by key
- Sometimes $\Theta(\log_s n)$: sorting, dense connectivity
- Less ideal $O(\operatorname{poly}\log n)$: sparse connectivity

Simulations

Theorem: Every round of an EREW PRAM Algorithm can be simulated with two rounds.

- Direct extensions to CREW, CRCW Algorithms

Proof Idea:

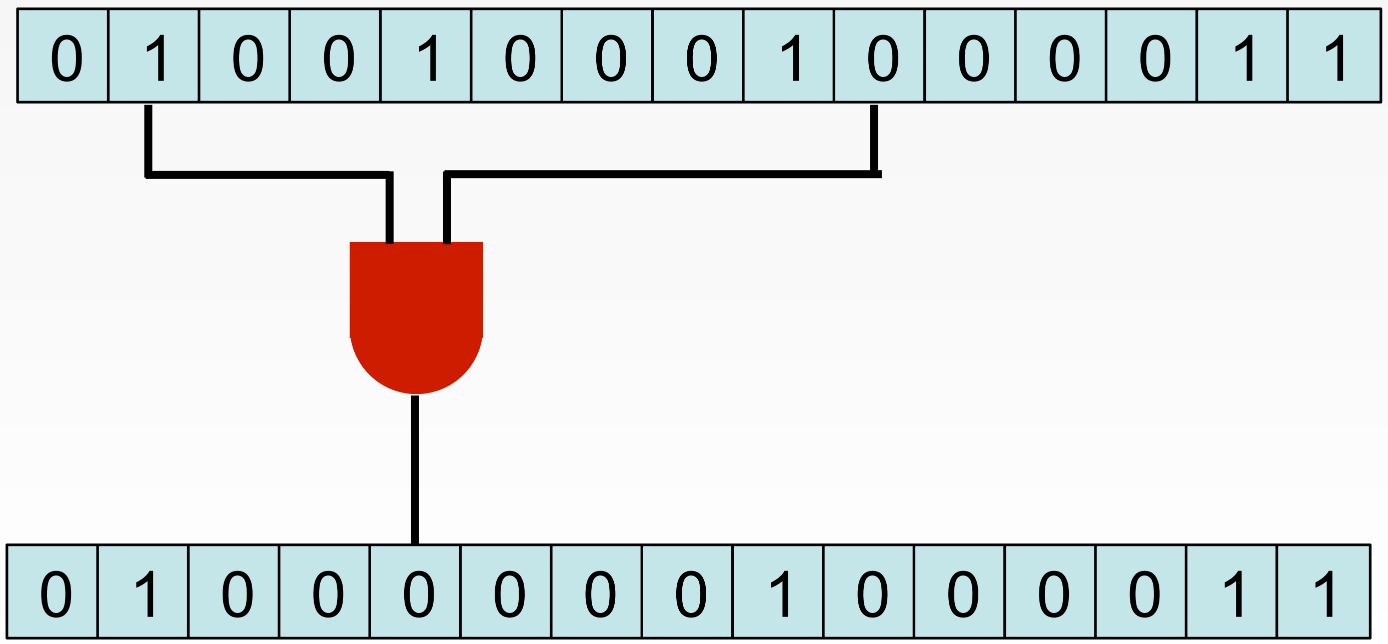
- Divide the shared memory of the PRAM among the machines, and simulate updates.

Simulations (cont)

Proof Idea:

- Divide the shared memory of the PRAM among the machines. Perform computation in one round, update memory in next.

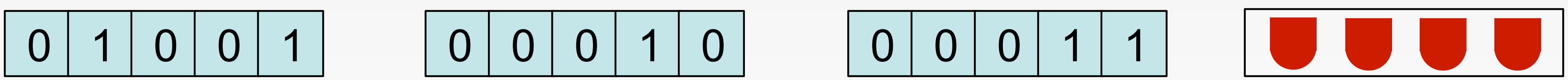
Memory:



Simulations (cont)

Proof Idea:

- Have “memory” machines and “compute machines.”
- Memory machines simulate PRAM’s shared memory
- Compute machines update the state

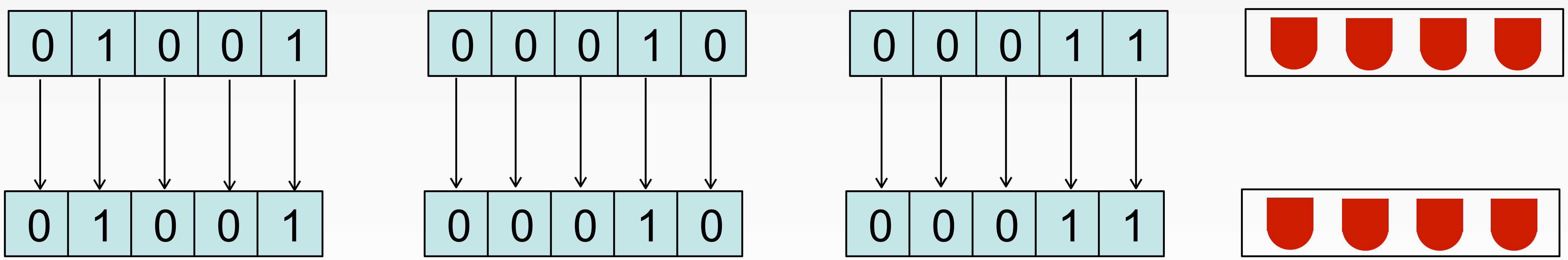


- EREW PRAM: Every at most two outputs & inputs (one for memory, one for compute)

Simulations (cont)

Proof Idea:

- Have “memory” machines and “compute machines.”
- Memory machines simulate PRAM’s shared memory
- Compute machines update the state

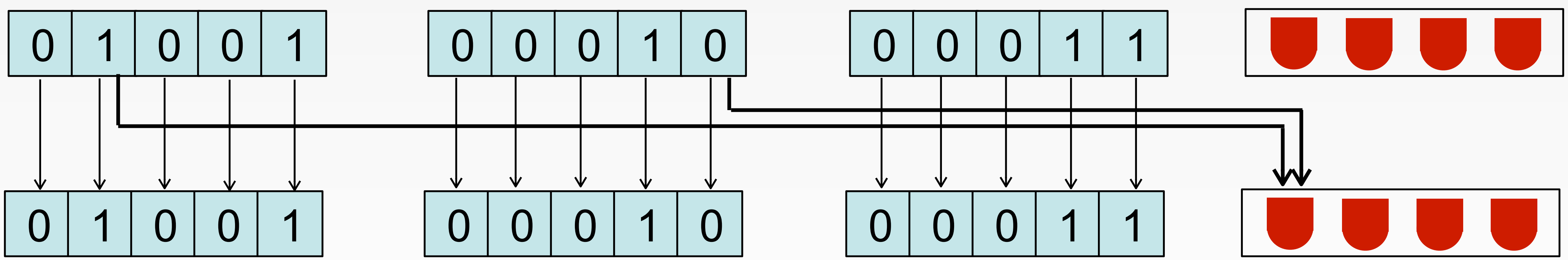


- EREW PRAM: Every at most two outputs & inputs (one for memory, one for compute)

Simulations (cont)

Proof Idea:

- Have “memory” machines and “compute machines.”
- Memory machines simulate PRAM’s shared memory
- Compute machines update the state

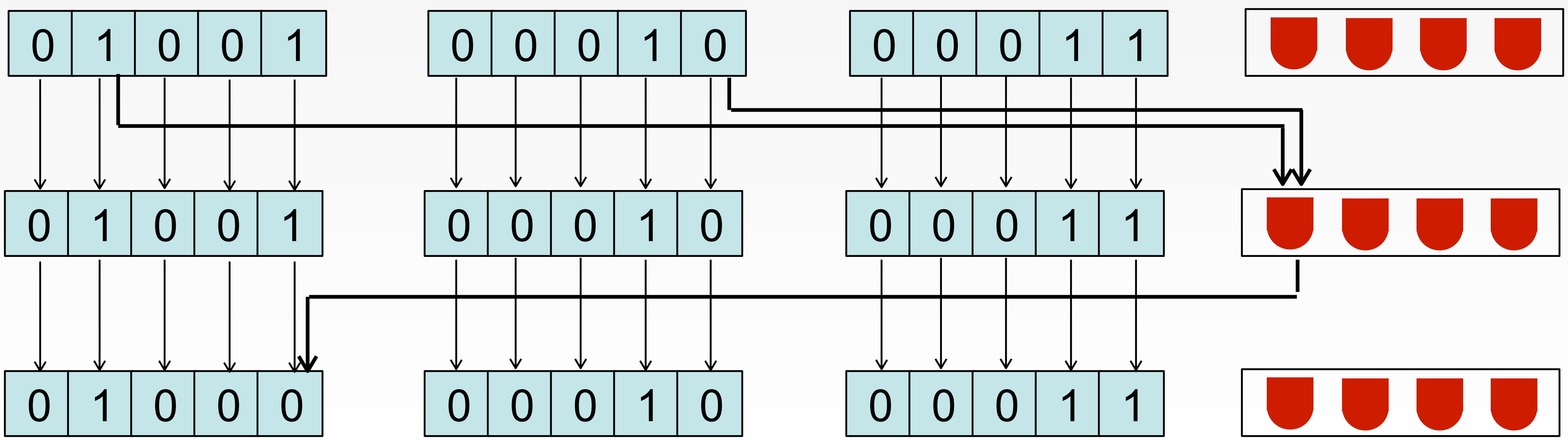


- EREW PRAM: Every at most two outputs & inputs (one for memory, one for compute)

Simulations (cont)

Proof Idea:

- Have “memory” machines and “compute machines.”
- Memory machines simulate PRAM’s shared memory
- Compute machines update the state



- EREW PRAM: Every at most two outputs & inputs (one for memory, one for compute)

Simulations

Theorem: Every round of an EREW PRAM Algorithm can be simulated with two rounds.

- Direct extensions to CREW, CRCW Algorithms

But, stronger than PRAMs.

- Subset sum. Given an array A, compute $B[i] = \sum_{j=0}^{i} A[j]$ for all i.
- Requires $O(\log n)$ rounds in PRAM
- Can be done in $O(\log_s n)$ rounds with space s

Algorithms

One Technique: Coresets!

- Reduce input size from n to s in parallel
- Solve the problem in a single round on one machine

Very Practical!

- n : Peta/Tetabytes
- $s \approx \sqrt{n}$: Giga/Megabytes

Talks today about coresets for:

- Clustering: k-means, k-median, k-center, correlation
- Graph Problems: connectivity, matchings
- Submodular Maximization

Lower Bounds

Some progress!

- Good bounds on what is computable in one round
- Multi-round lower bounds for restricted models (talks today)

Canonical problem:

- Given a two-regular graph, decide if it is connected or not.
- Best upper bounds $O(\log n)$ for $s = o(n)$
- Best lower bounds $\Omega(\log_s n)$ by circuit complexity reductions.
  - To improve must take number of machines into consideration

Schedule

9:00 - 9:30	Introduction
9:30 - 10:15	Distributed Machine Learning (Nina Balcan)
10:15 - 11:00	Randomized Composable Coresets (Vahab Mirrokni)
11:00 - 11:30	Coffee Break
11:30 - 12:15	Algorithms for Graphs on V. Large Number of Nodes (Krzysztof Onak)
12:15 - 2:15	Lunch (on your own)
2:15 - 3:00	Massively Parallel Communication and Query Evaluation (Paul Beame)
3:00 - 3:30	Graph Clustering in a few Rounds (Ravi Kumar)
3:30 - 4:00	Coffee Break
4:00 - 4:45	Sample & Prune: For Submodular Optimization (Ben Moseley)
4:45 - 5:00	Conclusion & Discussion

References: Models

BSP: Valiant. A bridging model for parallel computation. Communications ACM 1990.

MUD: Feldman, Muthukrishnan, Sidiropoulos, Stein, Svitkina. On Distributing Symmetric Streaming Computations. ACM TALG 2010.

MRC: Karloff, Suri, Vassilvitskii. A Model of Computation for MapReduce, SODA 2010.

IO-MR: Goodrich, Sitchinava, Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. ISAAC 2011.

Key-Complexity: Goel, Munagala. Complexity Measures for MapReduce, and Comparison to Parallel Sorting. ArXiV 2012.

MR: Pietracaprina, Pucci, Riondato, Silvestri, Upfal. Space Round Tradeoffs for MapReduce Computations. ICS 2012

MPC(1): Beame, Koutris, Suciu. Communication Steps for Parallel Query Processing. PODS 2013.

MPC(2): Andoni, Nikolov, Onak, Yaroslavtsev. Parallel Algorithms for Geometric Graph Problems. STOC 2014.

Big Data: Klauck, Nanongkai, Pandurangan, Robinson. Distributed Computation of Large Scale Graph Problems. SODA 2015