Graph Connectivity in MapReduce...

...How Hard Could it Be?

Sergei Vassilvitskii

+Karloff, Kumar, Lattanzi, Moseley, Roughgarden, Suri, Vattani, Wang

August 28, 2015

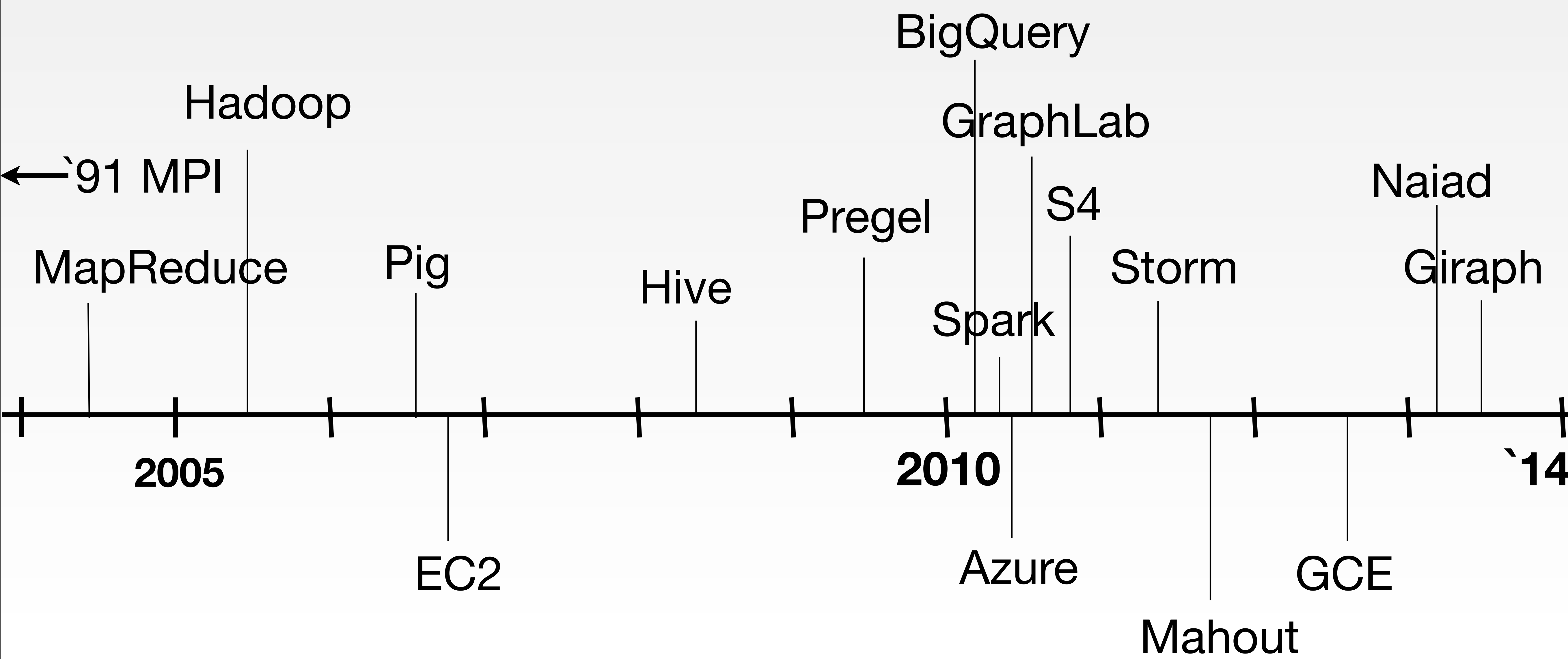
Google NYC

Maybe Easy...Maybe Hard...

Outline:

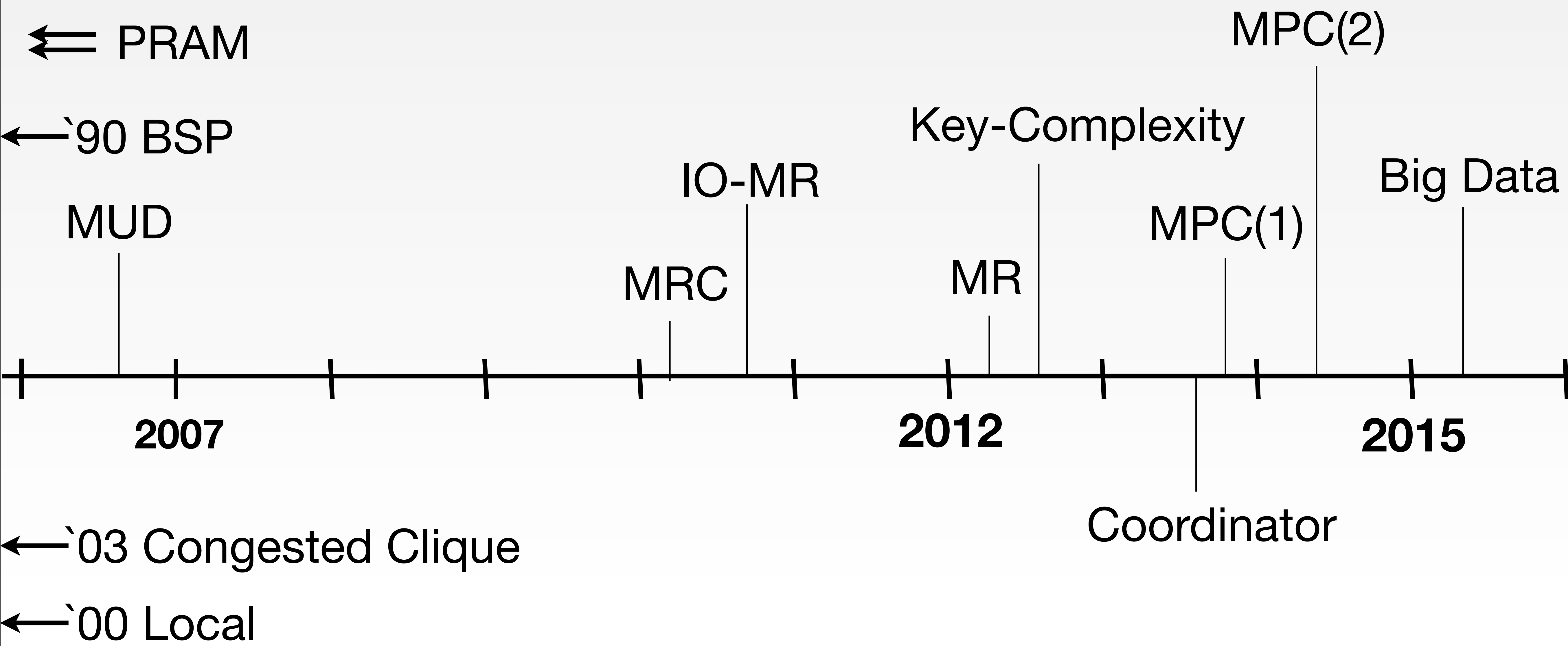
- Modeling:
  - MapReduce at its core
  - Comparison to PRAMs, BSP, Big Data, Local, Congest, ...
- Connectivity: Pretty Easy:
  - (1) PRAM Simulations & Algorithms
  - (2) Connectivity Coresets
  - (3) Unbounded width algorithms
- Connectivity: Pretty Hard:
  - (1) Merging paths
  - (2) s-Shuffles
  - (3) Circuit complexity implications
- State of the world

How did we get here?



*All dates approximate

Understanding where we are



* 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 (no memory across rounds)
- 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 (no memory across rounds)
- 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 $M, m, \mu, s, n/p^{1-\epsilon}$

Devil in the Details

0. Data partitioned across machines
1. Receive Data
2. Do local processing
   - Decide whether local processing is ‘free’ or not
   - Limitations on kinds of processing?

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

The Art of Modeling

Today: Three Models

Classical:

– PRAM. Polynomial M processors, receive constant s per round.

Today: Three Models

Classical:

– PRAM. Polynomial M processors, receive constant s per round.

Modern:

– MRC: Sublinear M processors, receive sublinear s per round

– Can simulate PRAMs with O(1) rounds for round when $M_{PRAM} < s \cdot M_{MRC}$

Today: Three Models

Classical:

- PRAM. Polynomial M processors, receive constant s per round.

Modern:

- MRC: Sublinear M processors, receive sublinear s per round
- Can simulate PRAMs with O(1) rounds for round when $M_{PRAM} < s \cdot M_{MRC}$

Abstract

- s-SHUFFLE: Unlimited M processors, receive sublinear s per round
- Can simulate both PRAMs and MRC with O(1) rounds for round

Connectivity is Easy

Many approaches:

- PRAM Algorithms
- PRAM Simulations & MR Friendly approaches
- Coresets
- Unbounded Width Algorithms

PRAM Algorithms (sample)

Model	Rounds	Processors	Reference
CRCW Deterministic	log n	m + n	Shiloach & Vishkin 82
CRCW Deterministic	log n	(m + n)α(n) / log n	Cole & Vishkin '91
CRCW Randomized	log n	(m+n) / log n	Gazit '91
EREW Deterministic	$\log^{1.5}$ n	m + n	Johnson & Metaxas '92
EREW Randomized	log n	m + $n^{1+\epsilon}$	Karger, Nisan Parnas' 92
EREW Randomized	log n log log n	(m + n) / log n	Karger, Nisan Parnas' 92
EREW Deterministic	log n log log n	m + n	Chong & Lam '95
EREW Randomized	log n	m + n	Radzik '94
EREW Randomized	log n	(m + n) / log n	Halperin & Zwick '96
EREW Deterministic	log n	m + n	Chong, Ham, Lam '01
EREW Deterministic	log n		Reingold '04

MRC

Theorem: Every EREW Algorithm using M processors can be simulated in MRC using $M/n^{1-\epsilon}$ machines.

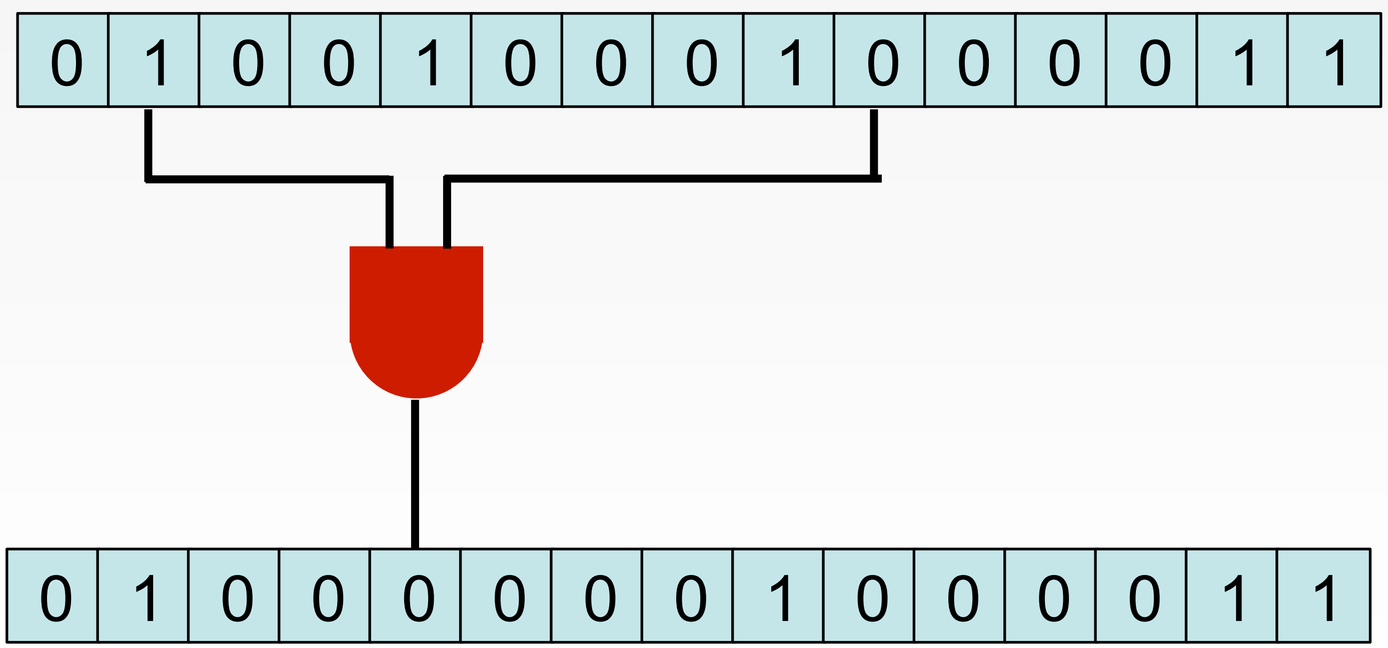
Corollary: Can compute connectivity in $O(\log n)$ rounds.

Simulations (cont)

Proof Idea:

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

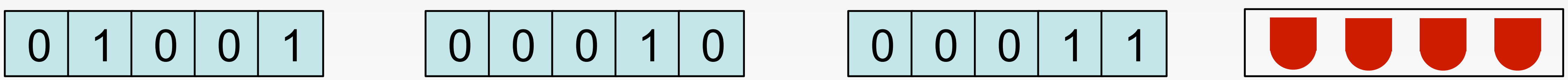
Memory:



Simulations (cont)

Proof Idea:

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

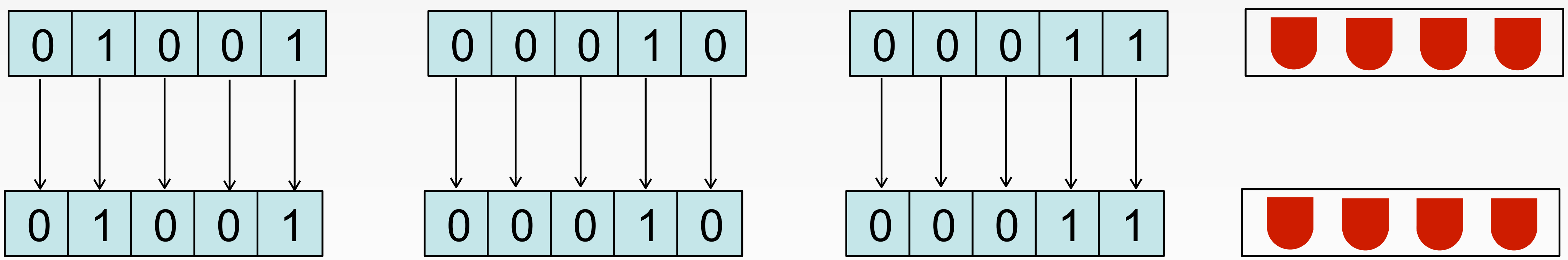


- EREW PRAM: Every processor needs at most two outputs & inputs (one for memory, one for compute)

Simulations (cont)

Proof Idea:

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

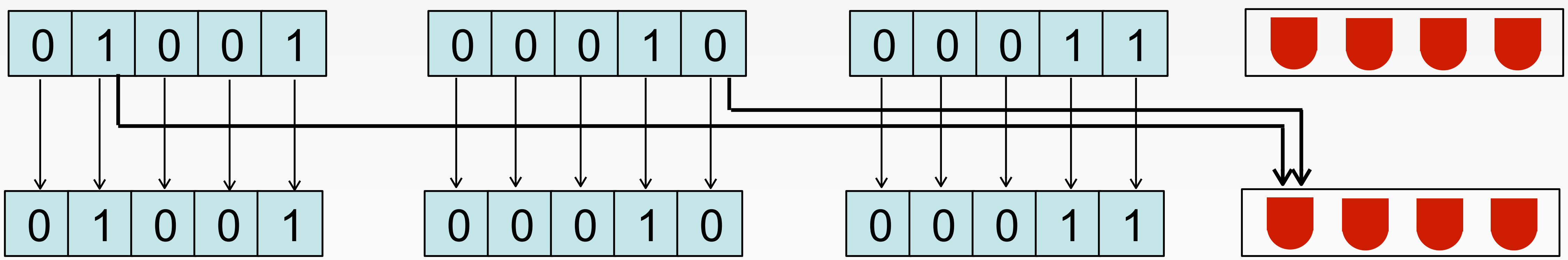


- EREW PRAM: Every processor needs at most two outputs & inputs (one for memory, one for compute)

Simulations (cont)

Proof Idea:

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

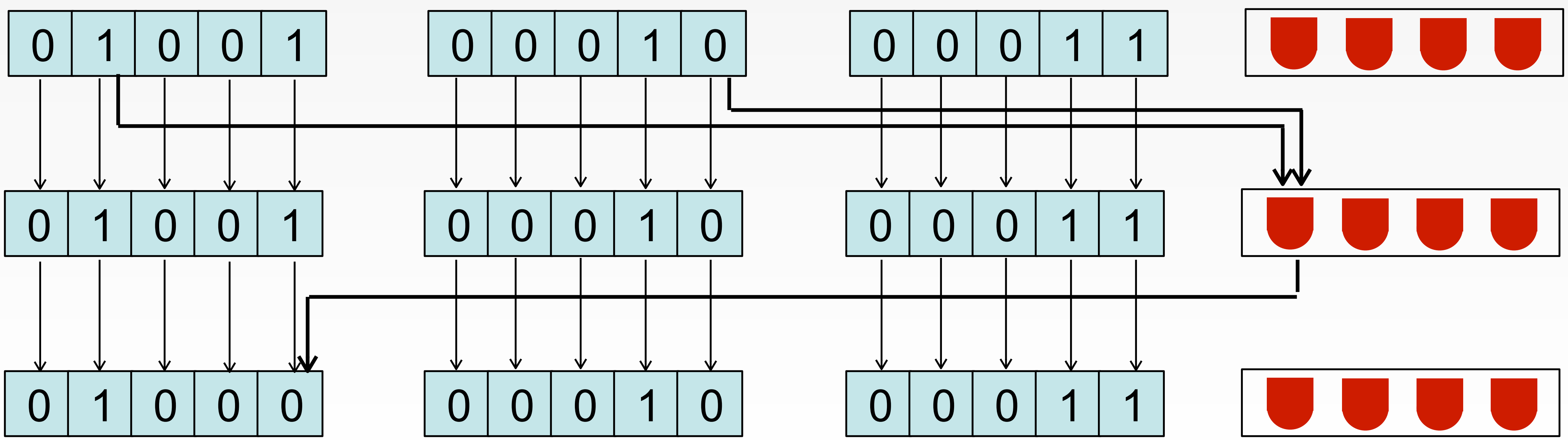


- EREW PRAM: Every processor needs at most two outputs & inputs (one for memory, one for compute)

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 processor needs at most two outputs & inputs (one for memory, one for compute)

So PRAMs?

So is MapReduce just a practical PRAM, or can it do more?

Biggest Difference:

- Input of size s
- Arbitrary computation at each node

Building Short Summaries

Main idea:

- Summarize parts of input while preserving the answer

Examples:

- Clustering: transform to a weighted instance
- Set cover: look for sets that cover many elements

Formally:

- Find the **coreset**: a smaller version of the problem so that the optimal solution is unchanged.

Coresets for Connectivity

Coreset:

- A spanning tree preserves all of the connectivity structure
- Spanning tree has at most $n << m$ edges.

Goal

- Make the graph as tree like as possible in parallel
- Load the tree onto a single machine
- Compute number of components

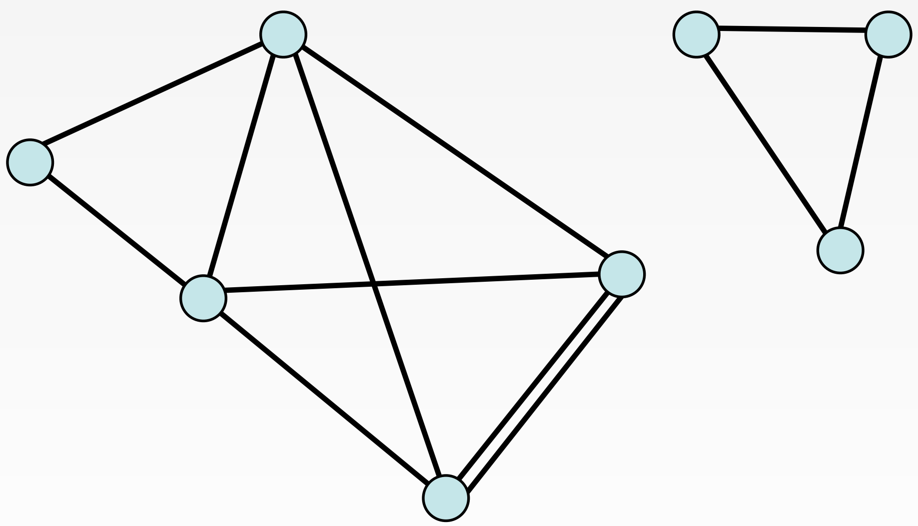
Making the graph tree-like

What makes the edge redundant

- If we already know the end points are connected

Connected Components

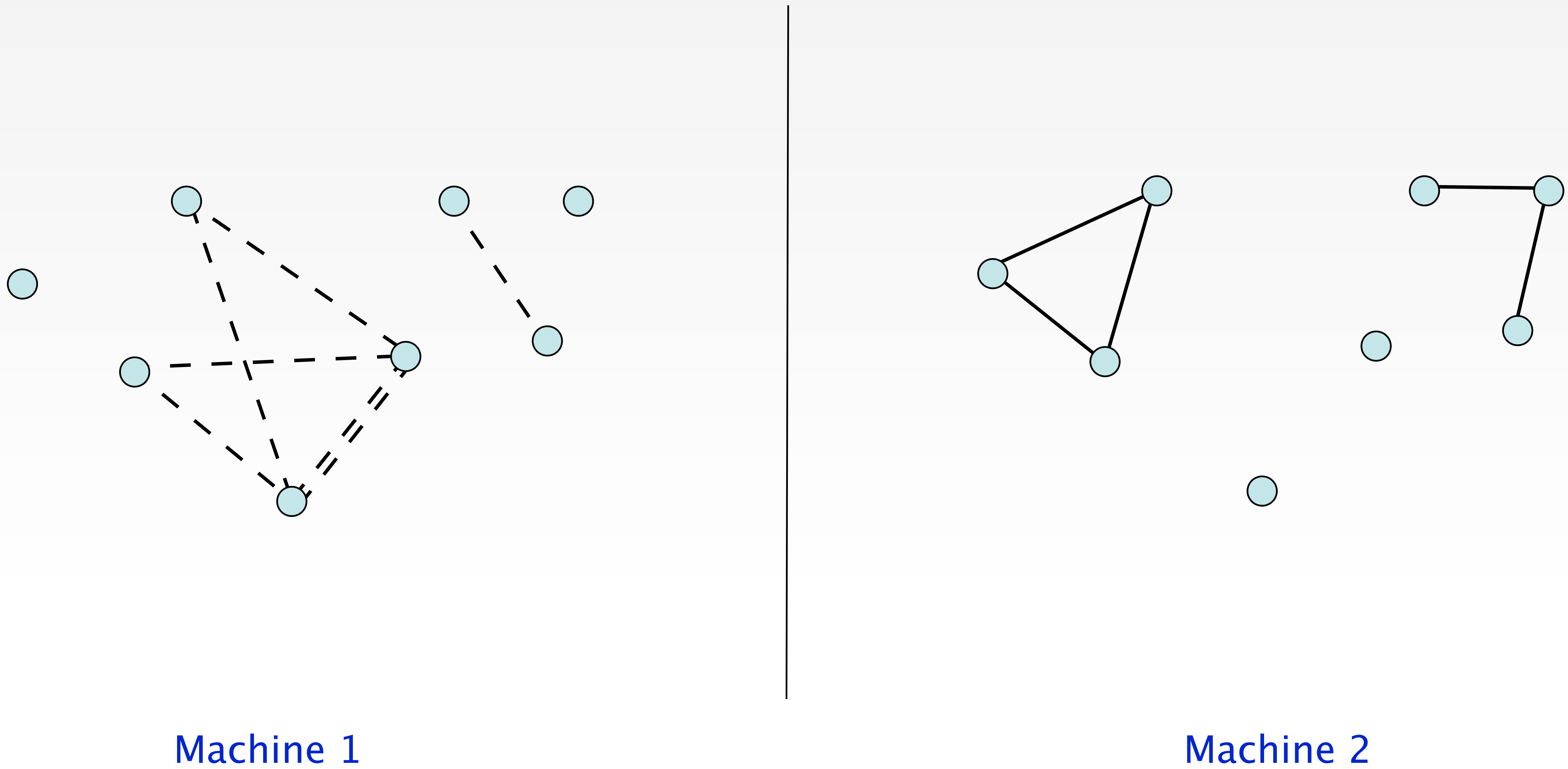
Given a graph:



Connected Components

Given a graph:

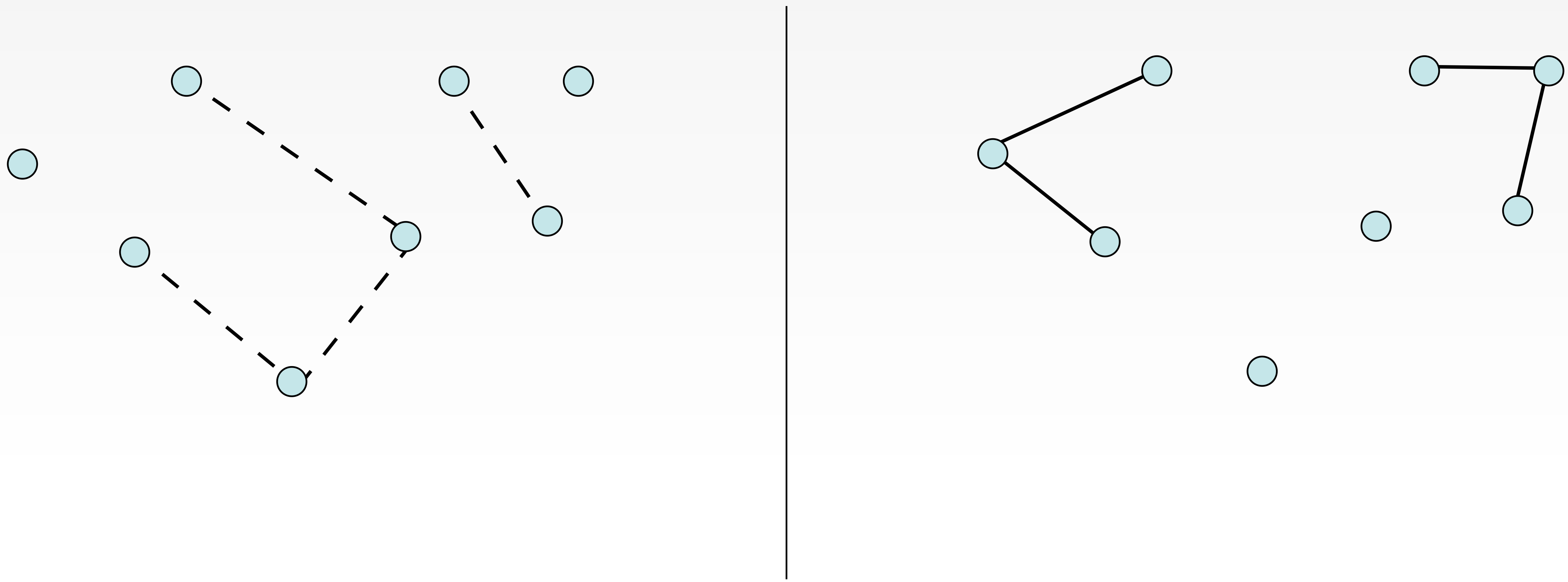
1. Partition edges (randomly)



Connected Components

Given a graph:

1. Partition edges (randomly)
2. Summarize (keep $\leq n-1$ edges per partition)



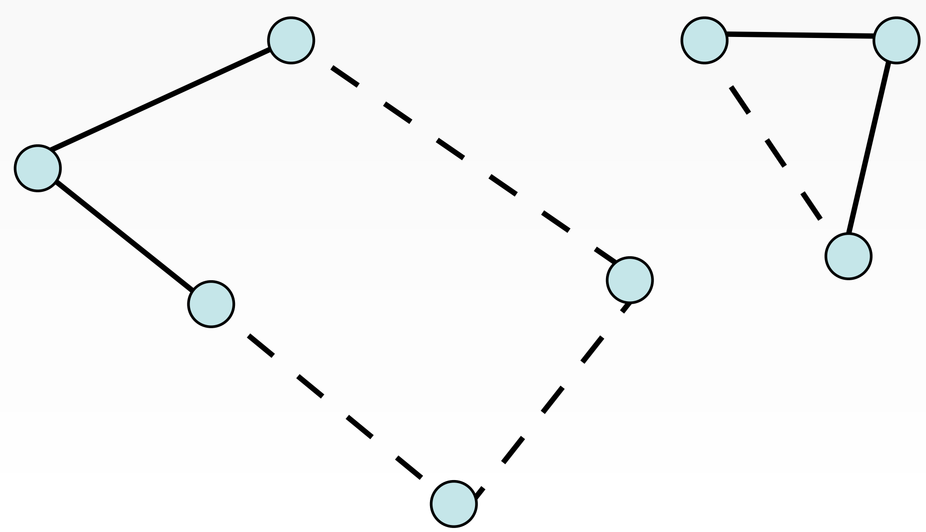
Machine 1

Machine 2

Connected Components

Given a graph:

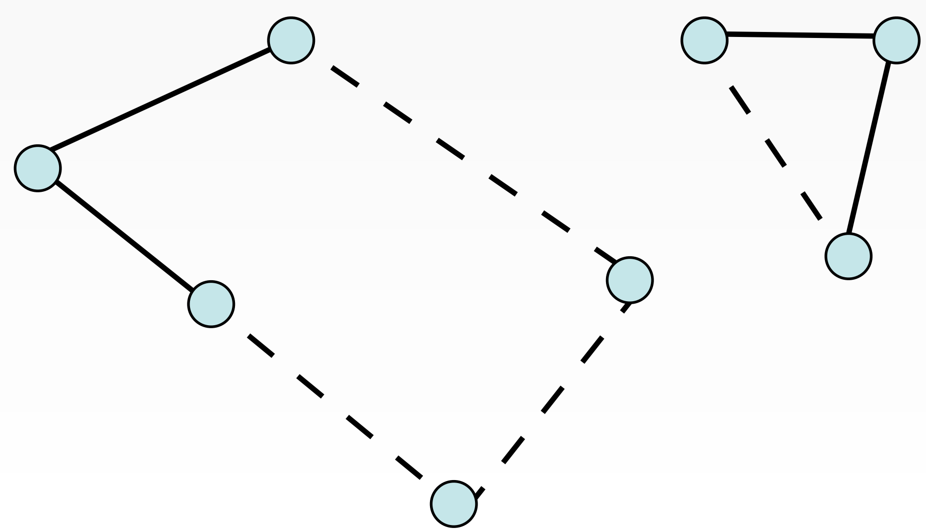
1. Partition edges (randomly)
2. Summarize (keep $\leq n-1$ edges per partition)
3. Recombine



Connected Components

Given a graph:

1. Partition edges (randomly)
2. Summarize (keep $\leq n-1$ edges per partition)
3. Recombine
4. Compute CC's



Analysis

Number of rounds:

- Suppose have n^{1+c} memory per node ($c > 0$)
- After one round keep: $m/(n^{1+c})$ trees each of size $n - 1$
- Therefore, total output: m/n^c
- After $1/c$ rounds, input size is less than n^{1+c}, can load the data on a single machine.

Analysis

Number of rounds:

- Suppose have n^{1+c} memory per node ($c > 0$)
- After one round keep: $m/(n^{1+c})$ trees each of size $n - 1$
- Therefore, total output: m/n^c
- After $1/c$ rounds, input size is less than n^{1+c}, can load the data on a single machine.

Gives smooth trade off on rounds vs. memory:

- Small c leads to more rounds, but less memory per node
- Large c leads to fewer rounds, but more memory per node
- Still need more than n memory per node (semistreaming regime)

Generalizing the idea

Coresets are a very useful MR primitive

- Graph properties:
  - Matchings
  - MSTs
- Submodular optimization
  - Set covers
  - Matroid, p-system constraints
- Clustering
  - k-center, k-means, ..
- ...

Connectivity Recap

Very small space:

- PRAM simulations require $s = O(1)$

Very large space:

- Coresets requires $s = n^{1+c}$

Connectivity Recap

Very small space:

– PRAM simulations require $s = O(1)$

Very large space:

– Coresets requires $s = n^{1+c}$

Medium space?

– $s = o(n) = \omega(1)$

Connectivity Recap

Very small space:

- PRAM simulations require $s = O(1)$

Very large space:

- Coresets requires $s = n^{1+c}$

Medium space?

- $s = o(n) = \omega(1)$
- start with many machines

Many many many machines

Easier problem:

- Given n, guess the input graph, check if the input is correct. Output #CCs
  - Partition the check across all of the machines. Each machine checks $\binom{n}{2}/s$ edges.

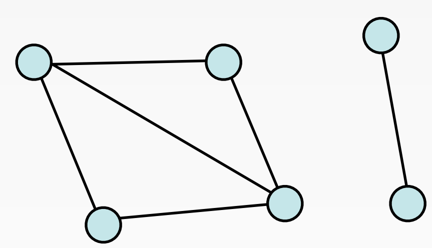
Many many many machines

Easier problem:

- Given n, guess the input graph, check if the input is correct. Output #CCs
  - Partition the check across all of the machines. Each machine checks $\binom{n}{2}/s$ edges.

Example:

- n = 6, s = 3
- Guess graph:



- Each machine gets three potential edges to check

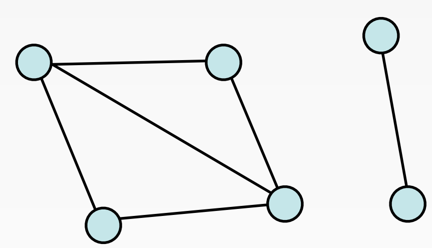
Many many many machines

Easier problem:

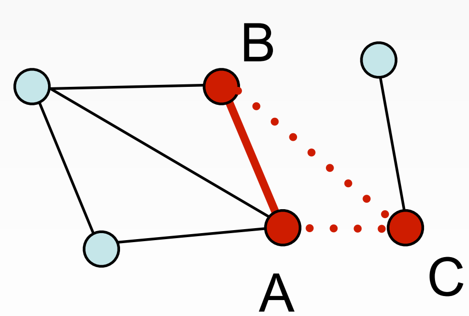
- Given n, guess the input graph, check if the input is correct. Output #CCs
  - Partition the check across all of the machines. Each machine checks $\binom{n}{2}/s$ edges.

Example:

- n = 6, s = 3
- Guess graph:



- Each machine gets three potential edges to check



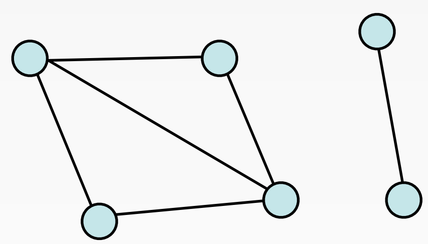
Many many many machines

Easier problem:

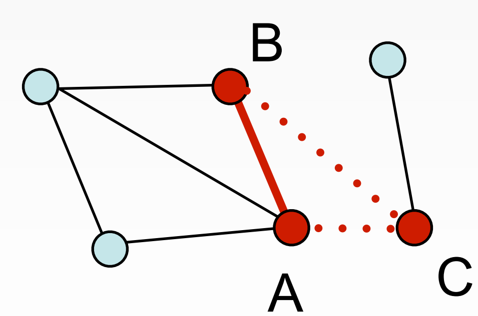
- Given n, guess the input graph, check if the input is correct. Output #CCs
  - Partition the check across all of the machines. Each machine checks $\binom{n}{2}/s$ edges.

Example:

- n = 6, s = 3
- Guess graph:



- Each machine gets three potential edges to check
- Outputs 1 if there's a match, stays silent otherwise



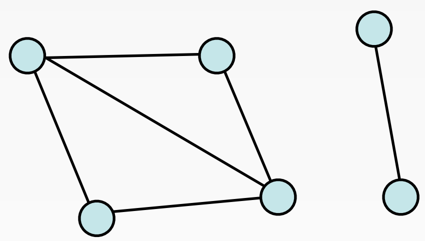
Many many many machines

Easier problem:

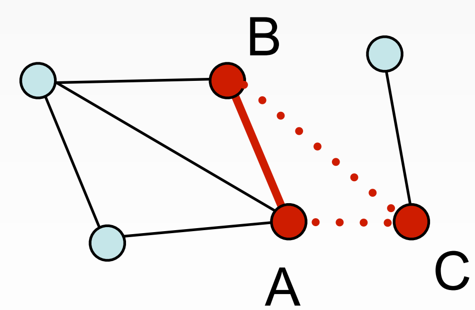
- Given n, guess the input graph, check if the input is correct. Output #CCs
  - Partition the check across all of the machines. Each machine checks $\binom{n}{2}/s$ edges.

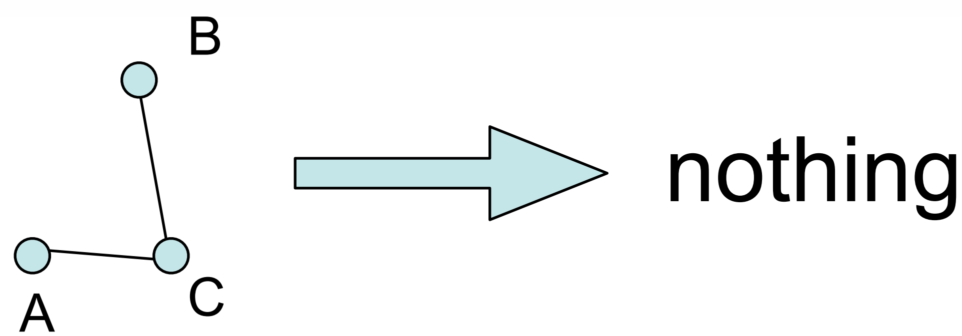
Example:

- n = 6, s = 3
- Guess graph:



- Each machine gets three potential edges to check
- Outputs 1 if there's a match, stays silent otherwise





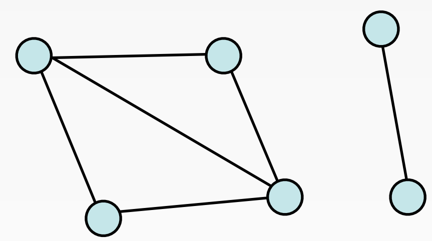
Many many many machines

Easier problem:

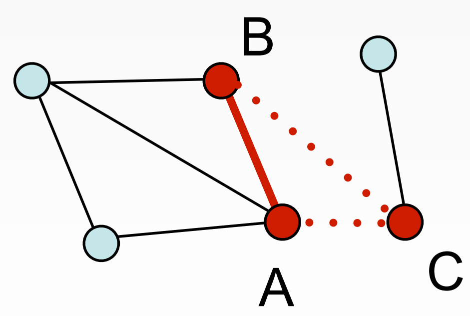
- Given n, guess the input graph, check if the input is correct. Output #CCs
  - Partition the check across all of the machines. Each machine checks $\binom{n}{2}/s$ edges.

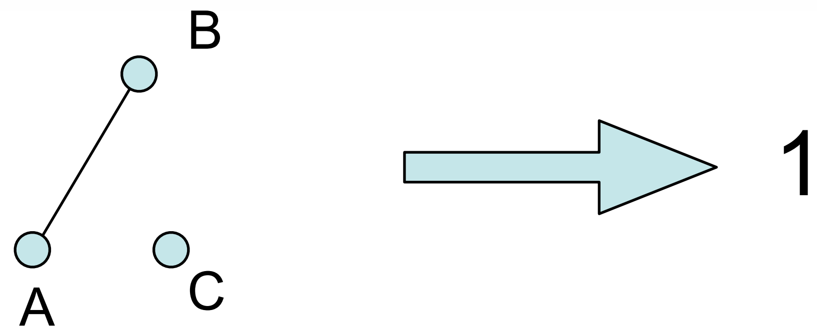
Example:

- n = 6, s = 3
- Guess graph:



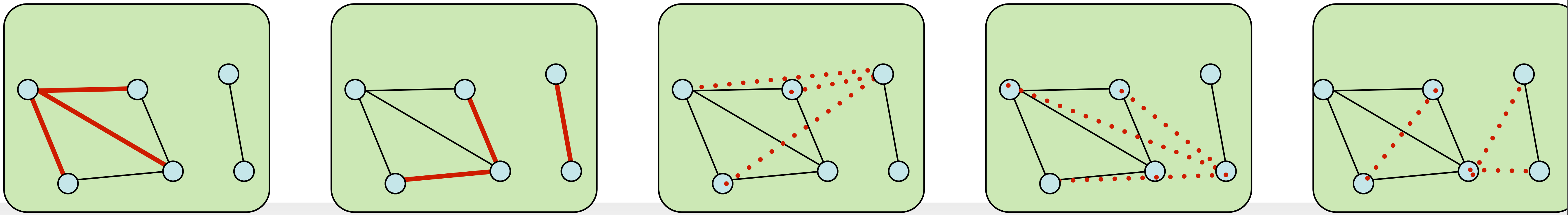
- Each machine gets three potential edges to check
- Outputs 1 if there's a match, stays silent otherwise





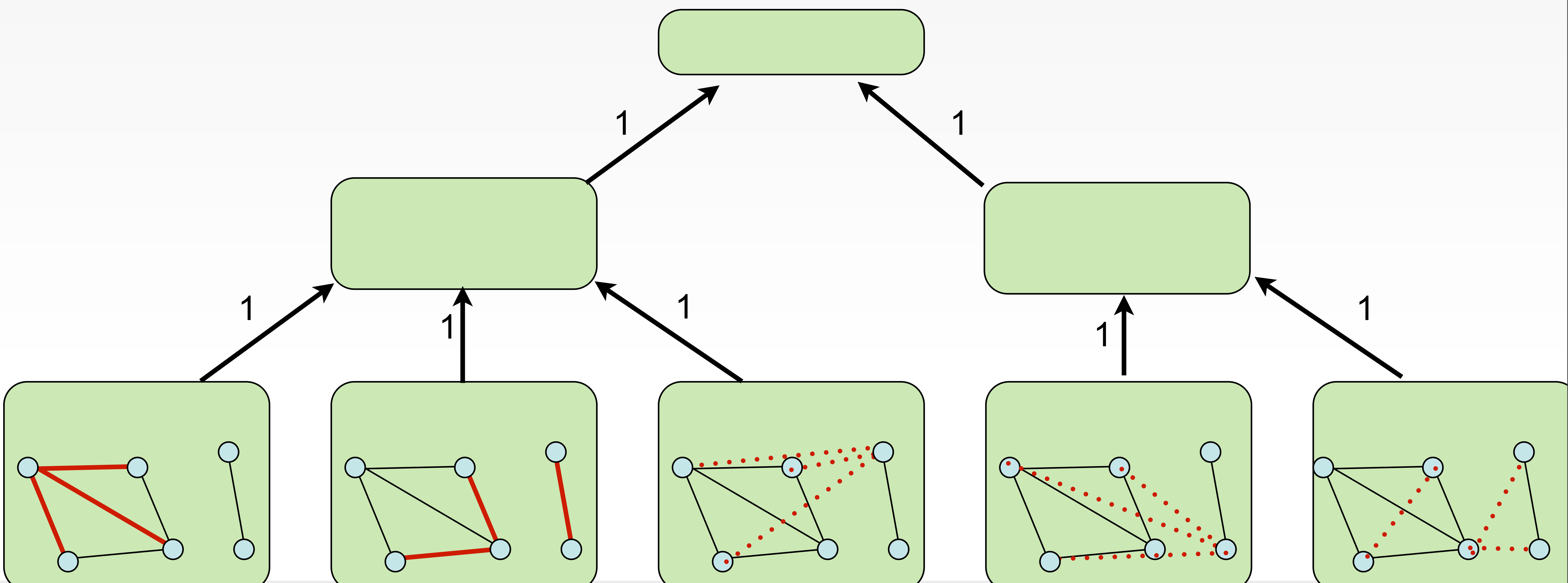
Example (Guessed Correctly)

- n = 6, s = 3, m at most 15 (simple graphs)
- First layer: each machine checks existence of the potential edges assigned to it



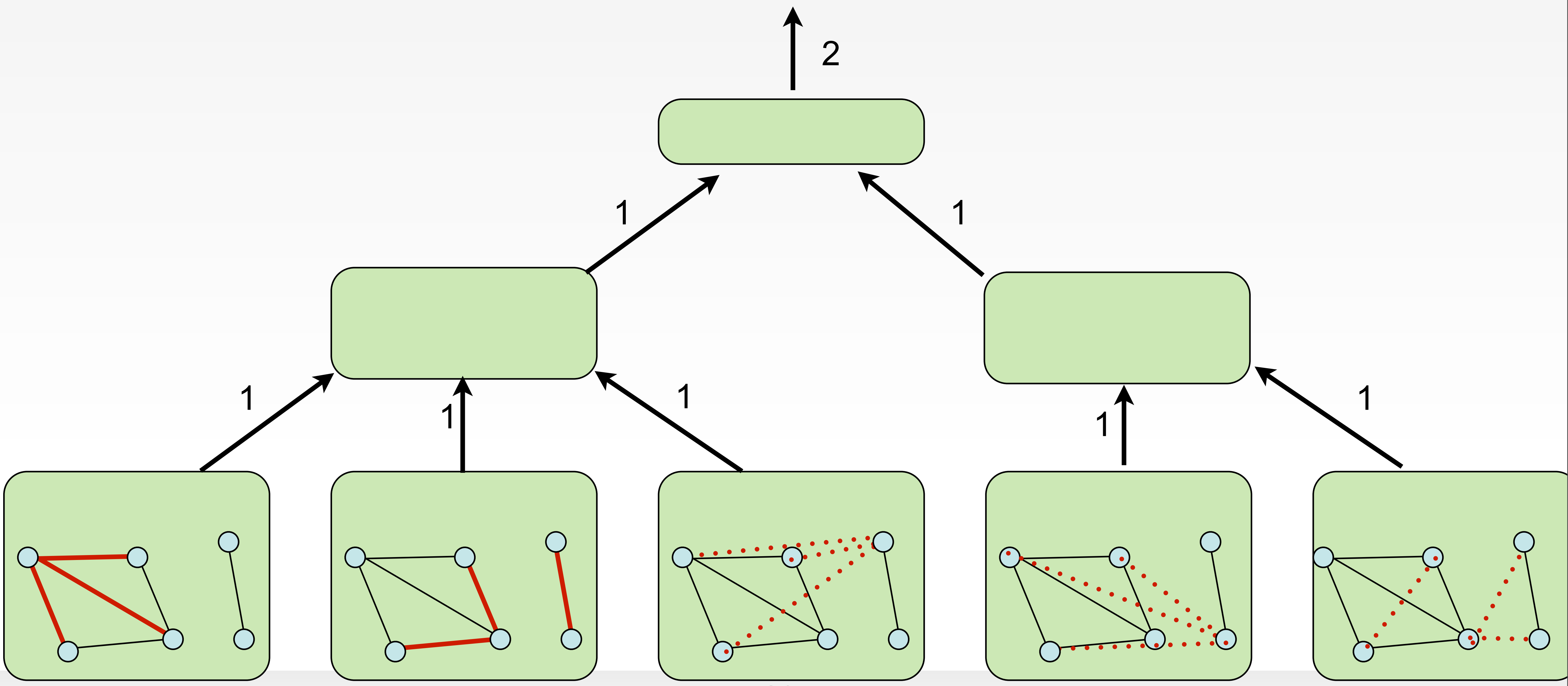
Example (Guessed Correctly)

- n = 6, s = 3, m at most 15 (simple graphs)
- First layer: each machine checks existence of the potential edges assigned to it
- Later layers: aggregate the checks, only continue if input is consistent



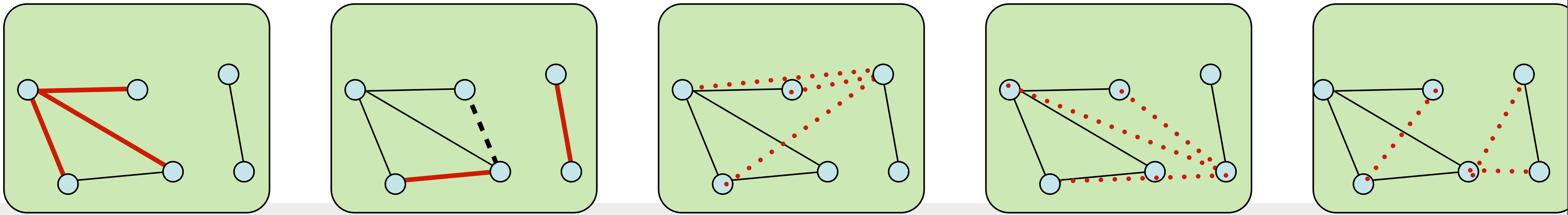
Example (Guessed Correctly)

- n = 6, s = 3, m at most 15 (simple graphs)
- First layer: each machine checks existence of the potential edges assigned to it
- Later layers: aggregate the checks, only continue if input is consistent
- Last layer: output # of CCs if consistent, silent otherwise



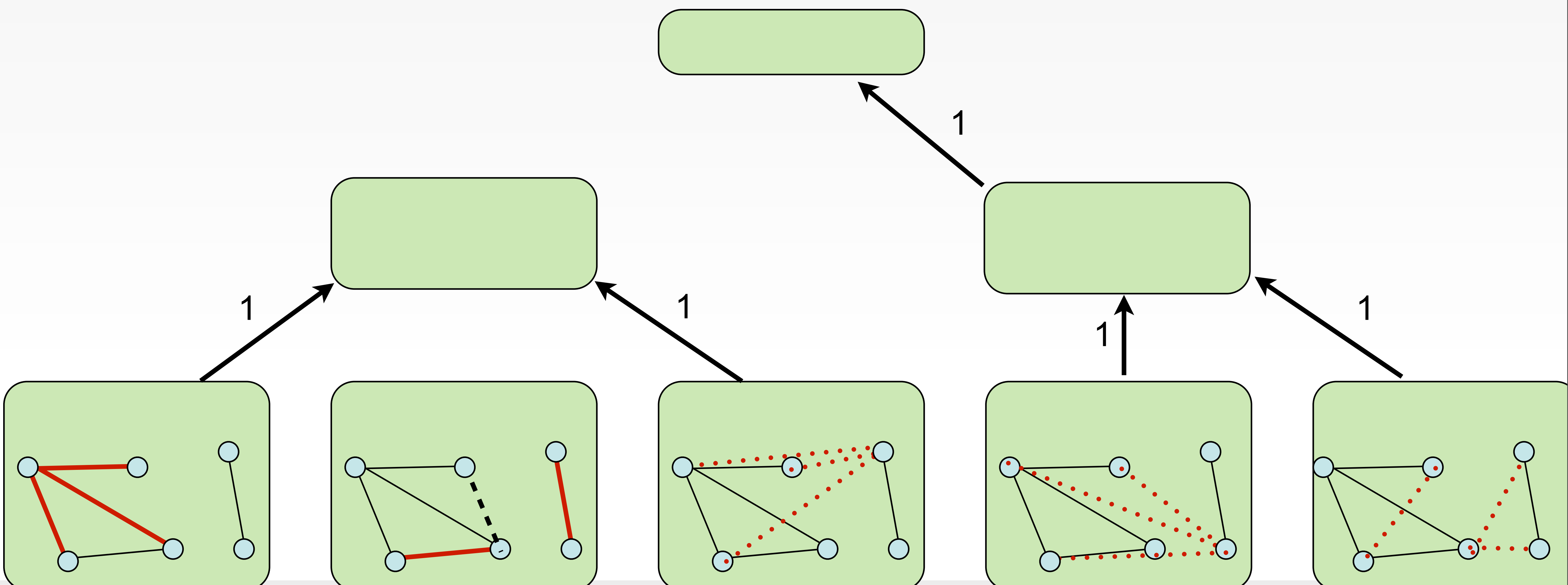
Example (Guessed Incorrectly)

- n = 6, s = 3, m at most 15 (simple graphs)
- First layer: each machine checks existence of the potential edges assigned to it



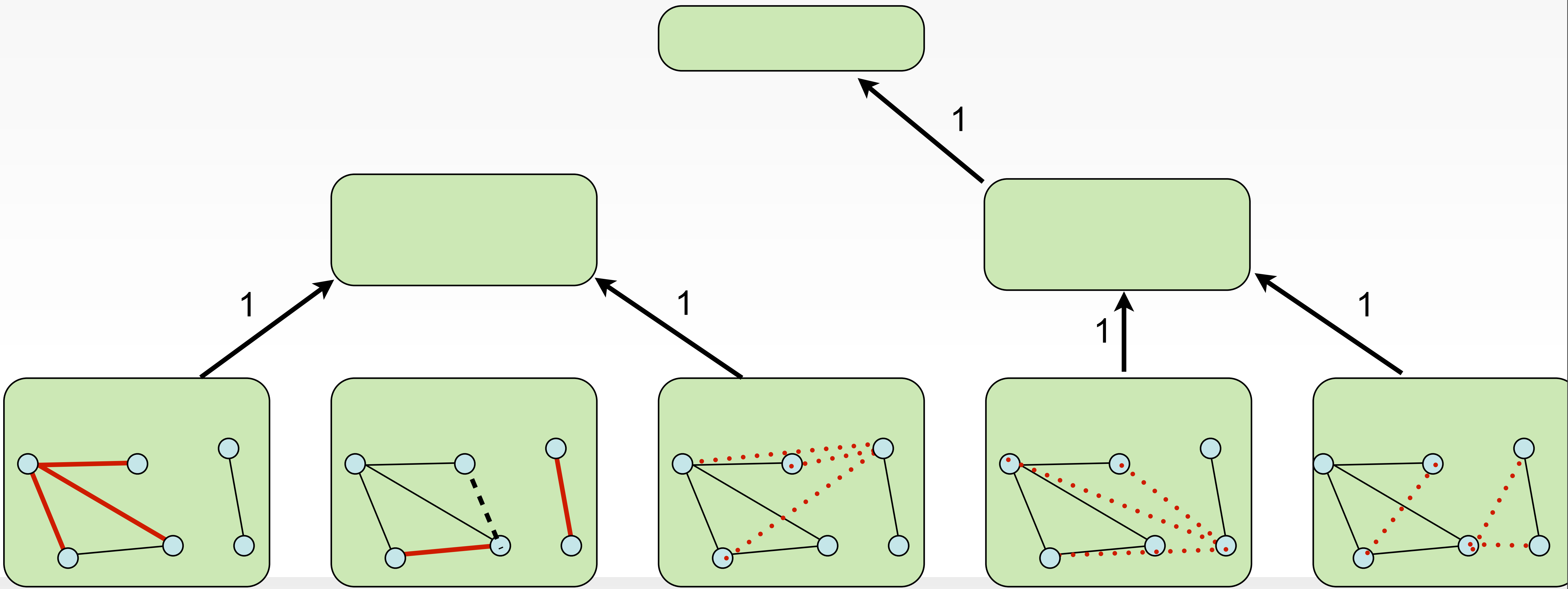
Example (Guessed Incorrectly)

- n = 6, s = 3, m at most 15 (simple graphs)
- First layer: each machine checks existence of the potential edges assigned to it
- Later layers: aggregate the checks, only continue if input is consistent



Example (Guessed Incorrectly)

- n = 6, s = 3, m at most 15 (simple graphs)
- First layer: each machine checks existence of the potential edges assigned to it
- Later layers: aggregate the checks, only continue if input is consistent
- Last layer: output # of CCs if consistent, silent otherwise



Analysis

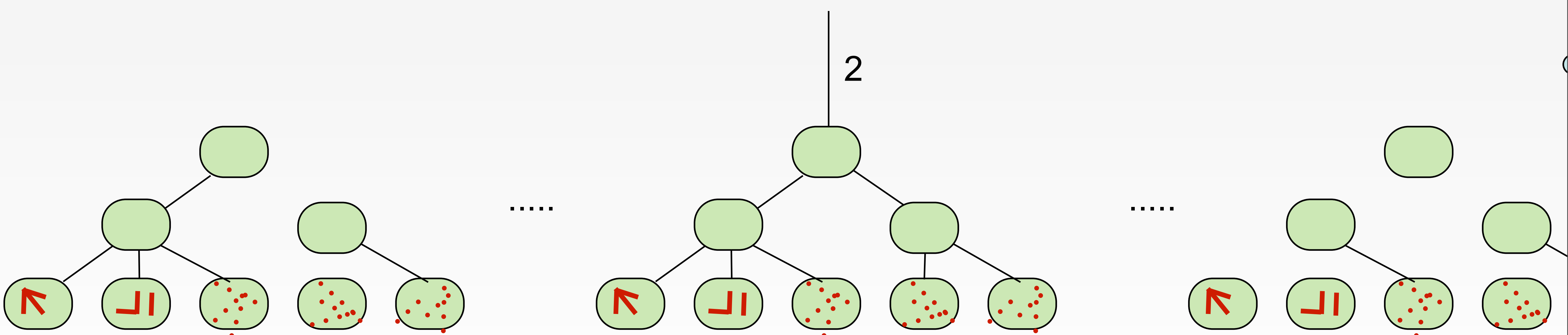
Rounds:

- One round to check
- $\log_s m$ rounds to aggregate

Analysis

Use the ‘many many many’ machines to parallelize the guessing

– Only one guess is correct, so only a single machine talks in the last round



– Total number of machines: exponential (compare to all graphs on n nodes)

Solving Connectivity

What do we know?

	Rounds	Space	Machines
PRAM	$O(\log n)$	$s = O(1)$	$n + m$
PRAM Sim	$O(\log n)$	$s = n^{\epsilon}$	$(n+m)/s$
Coreset	$O(\log_{s/n} n)$	$s = n^{1+c}$	m/s
Brute Force	$O(\log_{s} n)$	s	many (exponential)

Solving Connectivity

What do we know?

	Rounds	Space	Machines
PRAM	$O(\log n)$	$s = O(1)$	$n + m$
PRAM Sim	$O(\log n)$	$s = n^{\epsilon}$	$(n+m)/s$
Coreset	$O(\log_{s/n} n)$	$s = n^{1+c}$	m/s
Brute Force	$O(\log_{s} n)$	s	many (exponential)
??	$O(\log_{s} n)$	s	$o(m+n)$

The puzzle (circa 2010)

Given:

- 2 regular graph, n nodes, n edges
- $o(n)$ machines each with $s = o(n)$ memory

Decide:

- Is this one cycle or two cycles?

Best known:

- PRAM simulation, $O(\log n)$ rounds

Connectivity is hard!

Maybe it's impossible?

- Try Restricted Classes of Algorithms
- Find lower bounds in the abstract s-Shuffle setting
- Connections to circuit complexity

Restrictions

Idea 1. Make problem slightly harder

- If required to output the path, st-Connectivity requires $\Omega(\log n)$ rounds. [BKS13].
- Input revealed as part of a game, not all at once: then st-Connectivity requires $\Omega(\log n)$ rounds [JLS14].

Restrictions

Idea 1. Make problem slightly harder

- If required to output the path, st-Connectivity requires $\Omega(\log n)$ rounds. [BKS13].
- Input revealed as part of a game, not all at once: then st-Connectivity requires $\Omega(\log n)$ rounds [JLS14].

Idea 2. Limit the algorithm's power

- "Atomic" algorithms.
- Grow edges into paths, all decisions made as function of the path. Non intersecting paths don't leak information.
- Any MRC algorithm (total memory $o(n^2)$) needs $\Omega(\log n)$ rounds [IM].

No Restrictions

Memory s, Machines: unbounded

- Earlier saw an algorithm using $O(\log_s n)$ rounds
- Is that the best possible?

Isn't it obvious?

- The output depends on all n edges!

Obvious bounds

Simple function E on 3 bits [N91]:

- Return 1 if exactly one or two bits are set to 1.
- E(000) = 0, E(001) = 1, E(010) = 1, ..., E(110) = 1, E(111) = 0.

Square of the function on 9 bits:

- Apply E(.) on bits 1-3, 4-6, 7-9. Apply E(.) on the results
- $E^2(110010111) = E(E(110), E(010), E(111)) = E(110) = 1$.

Obvious bounds

Simple function E on 3 bits [N91]:

- Return 1 if exactly one or two bits are set to 1.
- E(000) = 0, E(001) = 1, E(010) = 1, ..., E(110) = 1, E(111) = 0.

Square of the function on 9 bits:

- Apply E(.) on bits 1-3, 4-6, 7-9. Apply E(.) on the results
- E^2(110010111) = E(E(110), E(010), E(111)) = E(110) = 1.

Obvious circuit bounds:

- Suppose $s = 2$
- Function depends on 9 bits
- Therefore requires circuit of depth $\lceil \log_2 9 \rceil = 4$.

Obvious? MR Bounds

Claim: E^2 can be implemented in 3 MR rounds with $s = 2$.

Obvious? MR Bounds

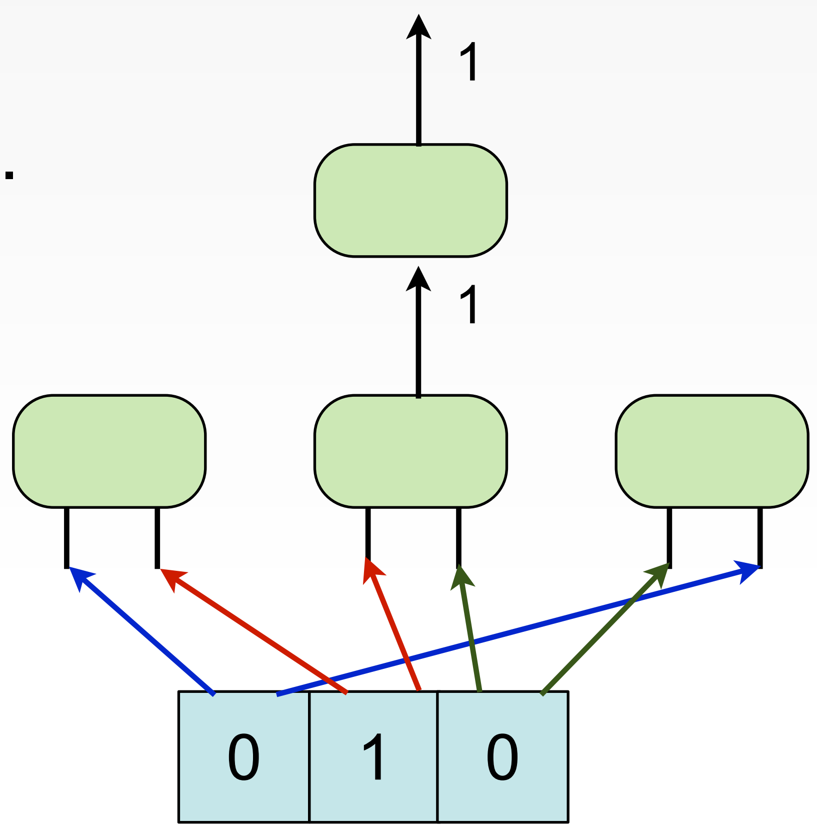
Claim: E^2 can be implemented in 3 MR rounds with $s = 2$.

Proof:

- Main idea: communication topology depends on the input.

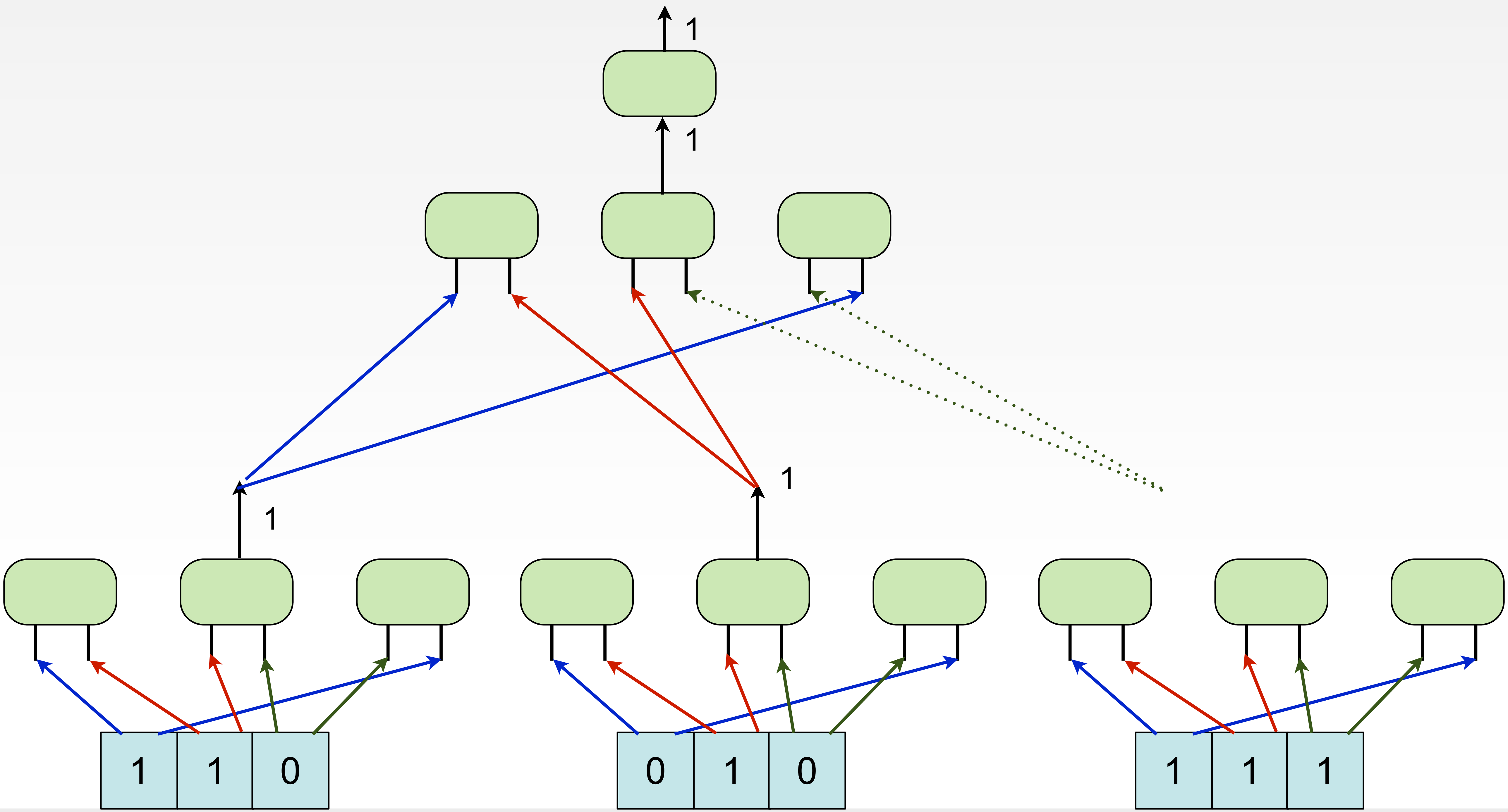
Details:

- Machine receives an ordered pair of inputs
- Output 1 if the first input is 1, the second is 0.
- Otherwise output nothing.



Obvious? MR Bounds

– Implementing E²: Just layer these gadgets, only aggregate in last step.



Silence is golden

Simple function E on 3 bits [N91]:

- Return 1 if exactly one or two bits are set to 1

Square of the function on 9 bits:

- Apply E(.) on bits 1-3, 4-6, 7-9. Apply E(.) on the results

Obvious circuit bounds:

- Suppose s = 2
- Function depends on 9 bits
- Therefore requires circuit depth of 4 rounds

Non-obvious MR bounds:

- Can be implemented in $3 < \log_2 9$ rounds!
- Staying silent says something

What’s really going on?

What is special about E^2? What about AND or OR?

What's really going on?

What is special about E^2? What about AND or OR?

Theorem[RVW]: Let $x_i \in \{0, 1\}$ be the i-th bit of the input. Then the output of any r-round s-shuffle computation can be expressed as a polynomial of degree s^r.

- Can generalize beyond decision problems. Every bit of output can be represented by a polynomial of this degree.

The degree of E is 2, even though it's on three inputs:

$$E(x_1, x_2, x_3) = x_1(1 - x_2) + x_2(1 - x_3) + x_3(1 - x_1)$$

So what?

Corollary: Any function of degree d requires $\lceil \log_s d \rceil$ rounds to compute using deterministic s-Shuffles (and therefore MR).

So what?

Corollary: Any function of degree d requires $\lceil \log_s d \rceil$ rounds to compute using deterministic s-Shuffles (and therefore MR).

Theorem(randomization): Any function of approximate degree $\tilde{d}$ requires at least $\lceil \log_s \tilde{d} \rceil$ rounds to compute using randomized s-Shuffles (and therefore MR).

Approximate Degree:

- For Boolean functions: $d \leq 216 \cdot \tilde{d}^6$
- Therefore, randomization does not help (much).

Back to connectivity

What is the degree of connectivity?

- undirected connectivity: $\binom{n}{2}$
- undirected st-connectivity: $\binom{n}{2}$

Therefore:

- Any s-Shuffle algorithm must take $\log_s \binom{n}{2} = \Omega(\log_s n)$ rounds.
- Any MapReduce algorithm must take $\Omega(\log_s n)$ rounds.
- Need $\Omega(1/\epsilon)$ rounds when $s = n^\epsilon$

Where are we now?

	Rounds	Space	Machines
PRAM	$O(\log n)$	$s = O(1)$	$n + m$
PRAM Sim	$O(\log n)$	$s = n^{\epsilon}$	$(n+m)/s$
Coreset	$O(\log_{s/n} n)$	$s = n^{1+c}$	m/s
Brute Force	$O(\log_{s} n)$	s	many (exponential)

– PRAM Simulations are rounds-optimal with a few restrictions.

– Brute Force is Rounds/Space optimal. Can we improve the machines bound?

Further lower bounds are hard

Theorem: Suppose some problem in $\mathcal{P}$ requires $\Omega(\log_s n)$ rounds using an s-shuffle with polynomial number of machines.

Then: $\mathcal{NC}^1 \subsetneq \mathcal{P}$.

Further lower bounds are hard

Theorem: Suppose some problem in $\mathcal{P}$ requires $\Omega(\log_s n)$ rounds using an s-shuffle with polynomial number of machines.

Then: $\mathcal{NC}^1 \subsetneq \mathcal{P}$.

Proof[sketch]:

- Any circuit of depth $O(\log n)$ can be simulated using an s-shuffle in $O(\log_s n)$ rounds.
- Each round corresponds to $\log s$ layers of the circuit.
- Since the circuit has fan-in 2, the total input need to simulate each round is at most s.

Further lower bounds are hard

Theorem: Suppose some problem in $\mathcal{P}$ requires $\Omega(\log_s n)$ rounds using an s-shuffle with polynomial number of machines.

Then: $\mathcal{NC}^1 \subsetneq \mathcal{P}$.

Barebone requirements of an s-Shuffle:

- Number of machines is polynomial
- Computation is synchronous
- Each machine reads s bits from input or previous rounds of computation
- Each machine needs to compute Boolean functions of its inputs

How hard is connected components?

Ideal:

- $o(m)$ machines each with $s = o(m)$ memory
- Runs in constant rounds

How hard is connected components?

Ideal:

- o(m) machines each with s = o(m) memory
- Runs in constant rounds

Status:

- PRAM Algorithms: run in log m rounds
- Dense graphs: use coresets

Further improvement:

- ‘Path stitching’ algorithms won’t work
- ‘Small space’ algorithms won’t work
- Puzzle still unresolved.

Thank You