Cooperative Computing for Autonomous Data Centers

Jon Berry (Sandia National Laboratories)

Mike Collins (Christopher Newport University)

Aaron Kearns (U. New Mexico)

Cynthia A. Phillips (Sandia National Laboratories)

Jared Saia (U. New Mexico)

Randy Smith (Sandia National Laboratories

LDRD
LABORATORY DIRECTED RESEARCH & DEVELOPMENT

NNSA
National Nuclear Security Administration

CCR
Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

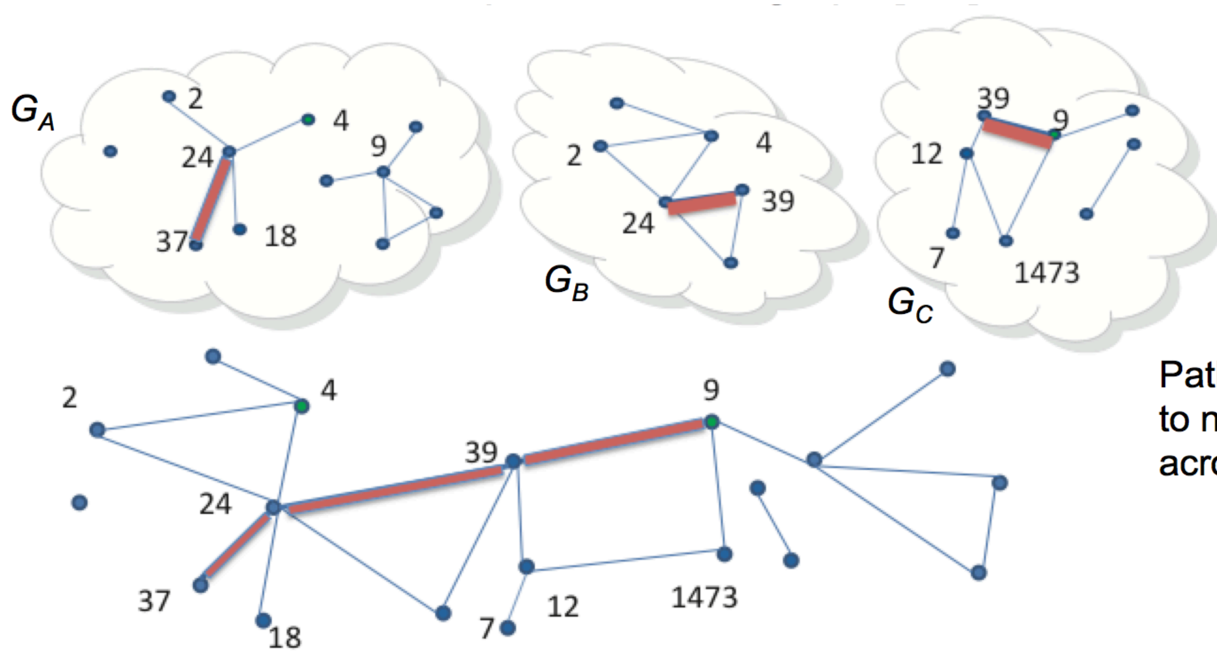
Sandia National Laboratories

A New Distributed Computing Model

Alice and Bob (or more) independently create social graphs G_A and G_B.

- Alice and Bob each know nothing of the other's graph.
- Shared namespace. Overlap at nodes.

Goal: Cooperate to compute algorithms over G_A union G_B with limited sharing: $O(\log^k n)$ total communication for size n graphs, constant k



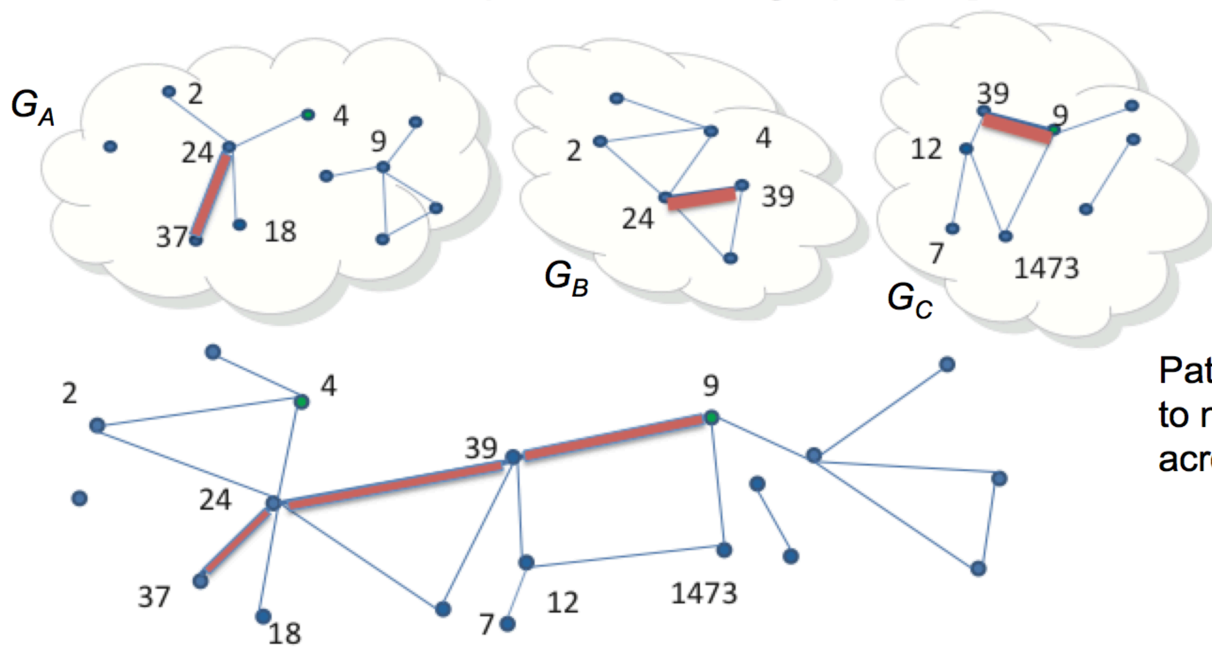
s-t connectivity
For 3 centers

Path from node s=37
to node t=9 is split
across all three centers

Another Limited Sharing Model

Goal: Cooperate to compute algorithms over $G_A \cup G_B(\cup G_C \ldots)$
Alice gets no information beyond answer in honest-but-curious model.

- Secure multiparty computation
  - Few players, large data



s-t connectivity
For 3 centers

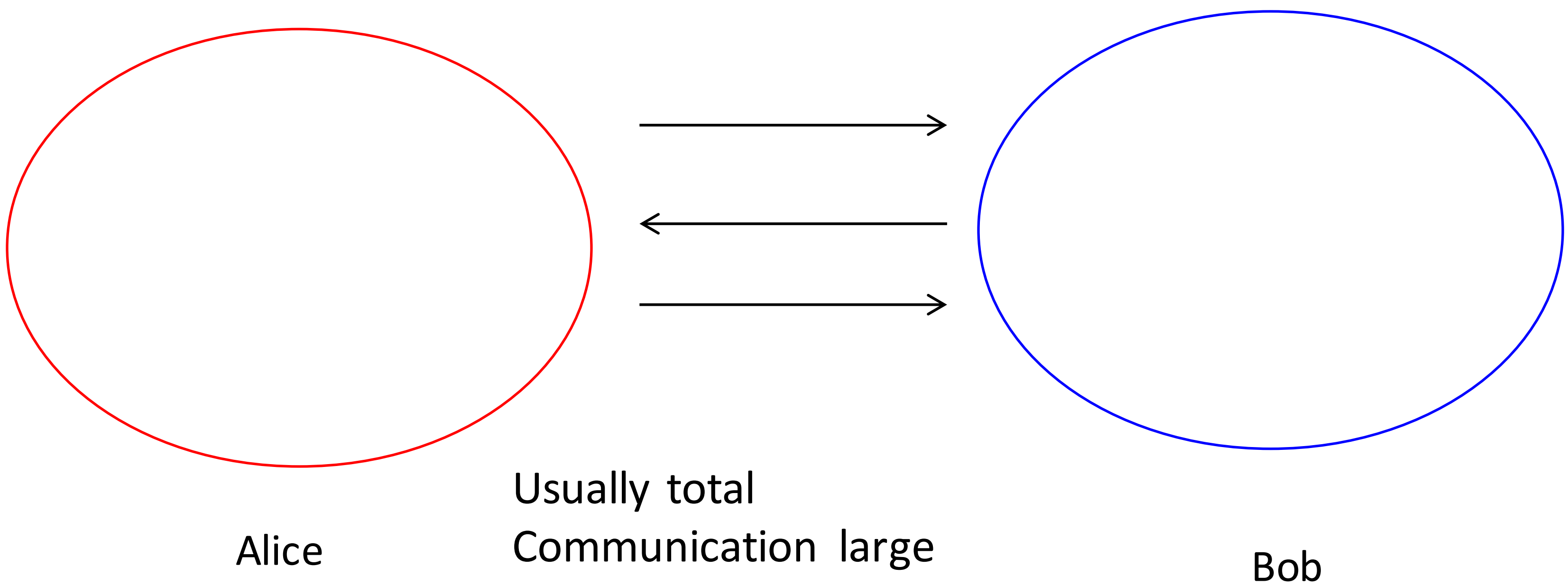
Path from node s=37
to node t=9 is split
across all three centers

Motivation

- Company mergers (Brickell and Shmatikov)
  - B&S algorithm assumes node names are known
- National security: connect-the-dots for counterterrorism
- Nodes are people
  - Exploit structure of social networks

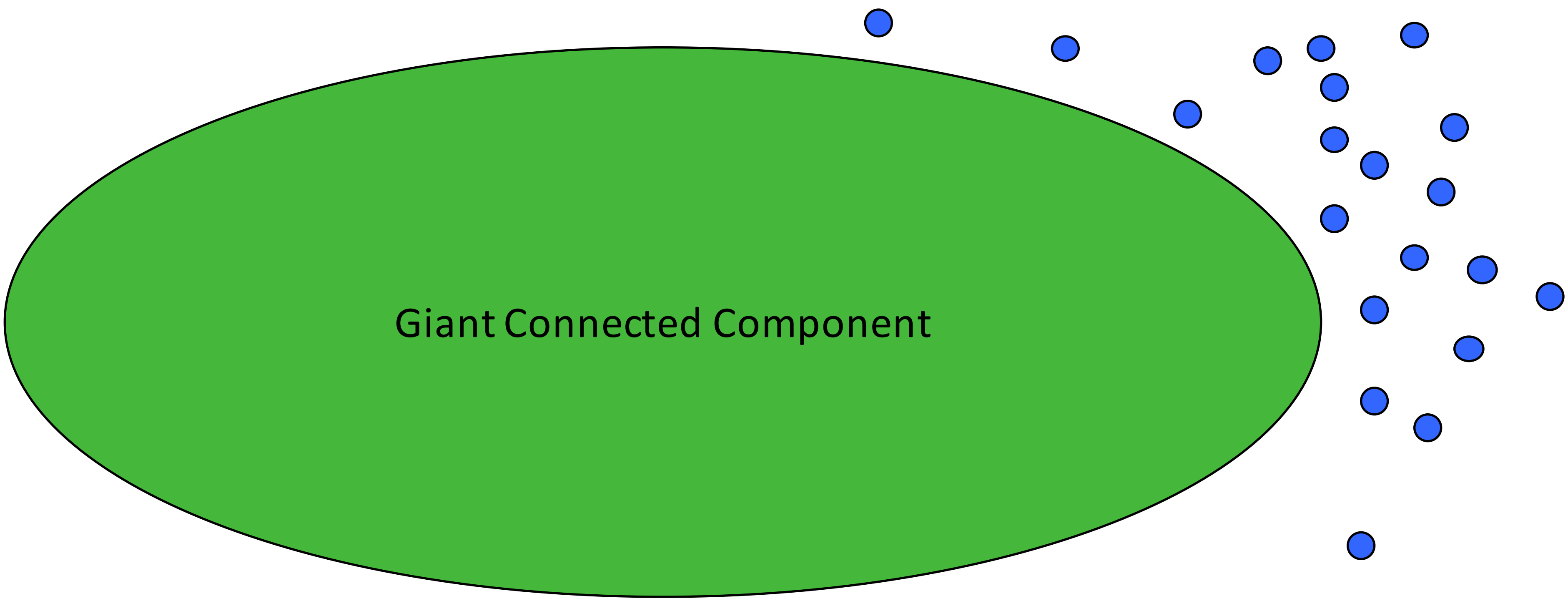
Result: Low-Communication s-t Connectivity

- s-t connectivity for social graphs: $O(\log^2 n)$ bits for n-node social networks
- $\Omega(n \log n)$ lower bound for general graphs (Hajnal, Maass, Turàn)
  - Edges partitioned, 2 parties



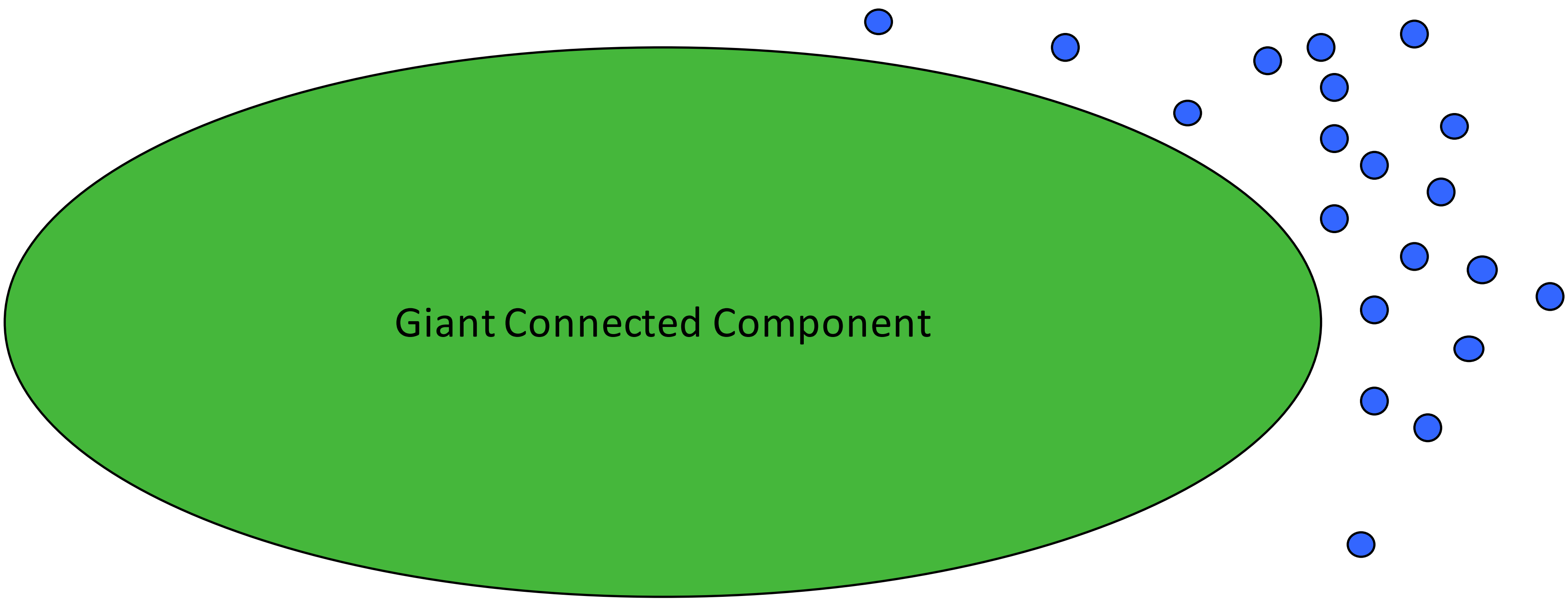
Social Network Structure

- Social networks have a giant component: second smallest component of size O(log n)



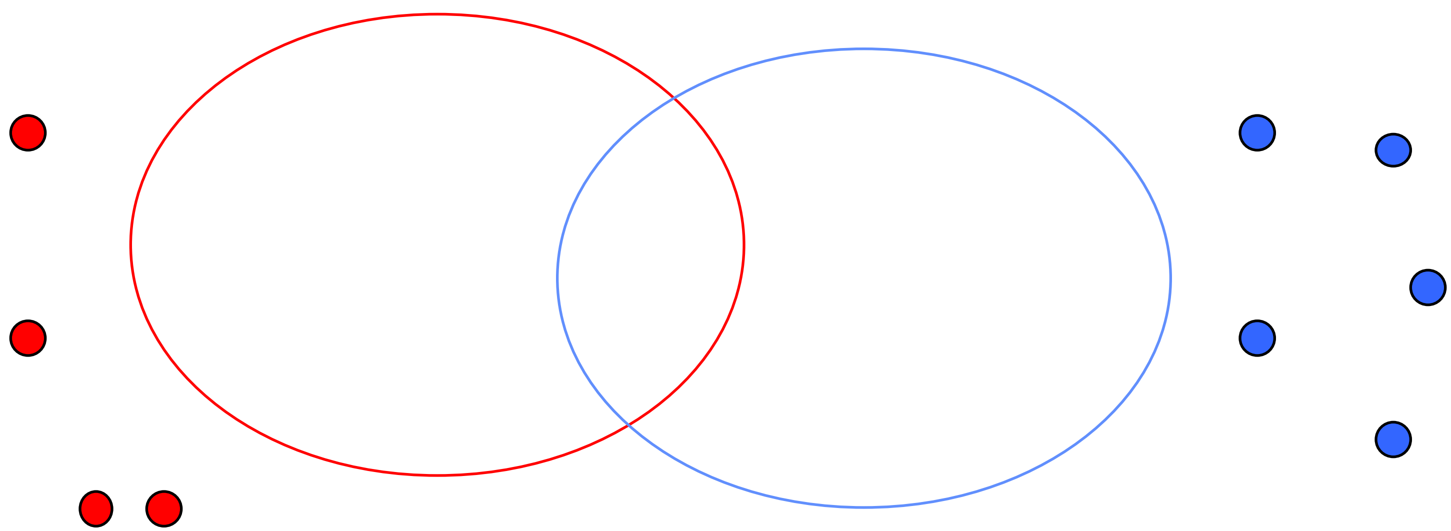
Social Network Structure

- Normal connection growth (Easley and Kleinberg)
- Observed in social networks (long distance phone call, linkedin, etc)
- Theoretically in Chung-Lu graphs with power-law exponent between 1+ε and 3.47



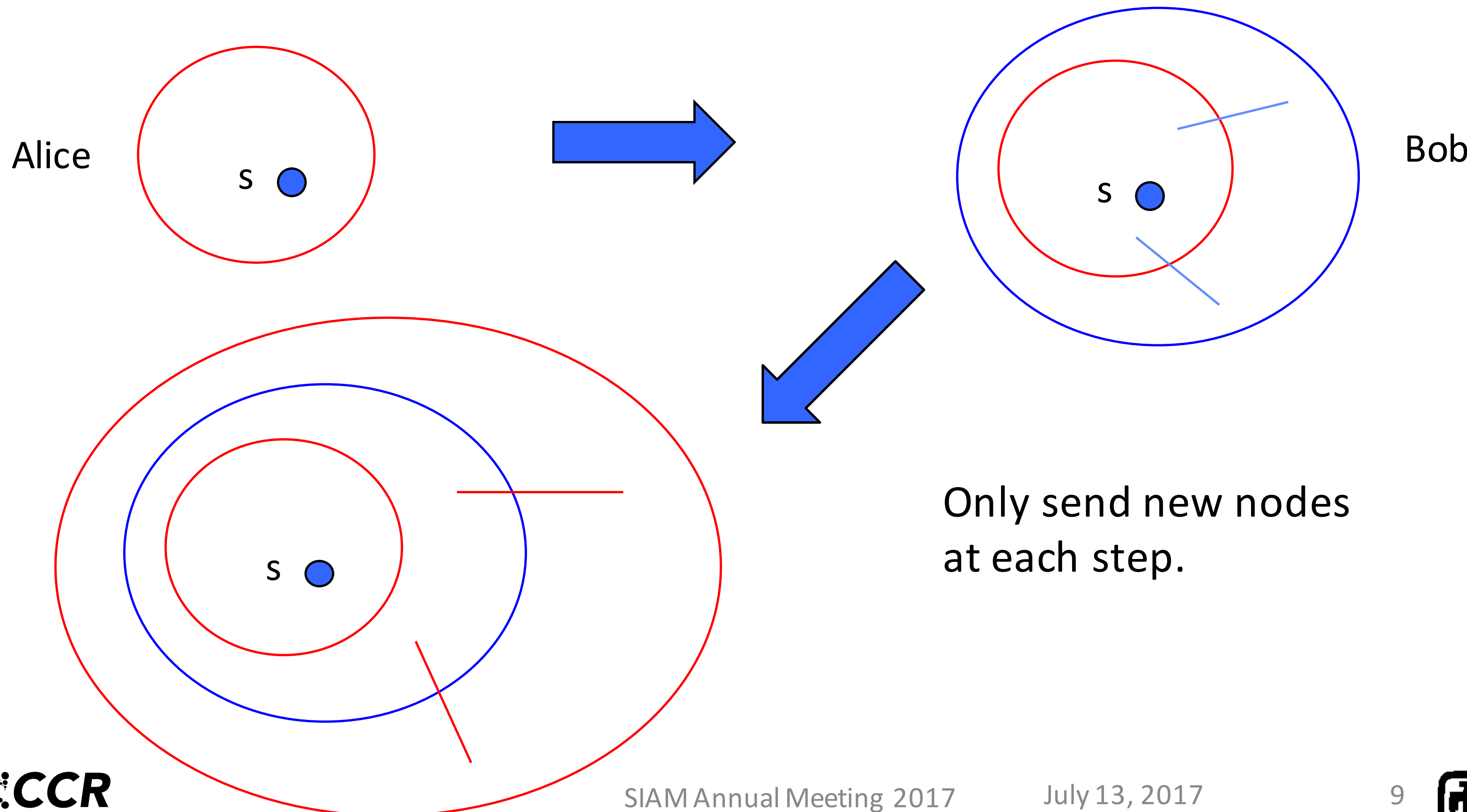
Assumptions

- Alice's graph G_A and Bob's graph G_B both have giant components
- These giant components intersect
  - Can verify with $O(\log^2 n)$ communication with high probability if intersect by a constant fraction (say 1%)



Shell Expansion

- Like breadth-first-search, "layer" is connected piece in G_A or G_B
- Key: don't explore too much of the graph(s)



Only send new nodes at each step.

Low-Sharing s-t Connectivity Algorithm

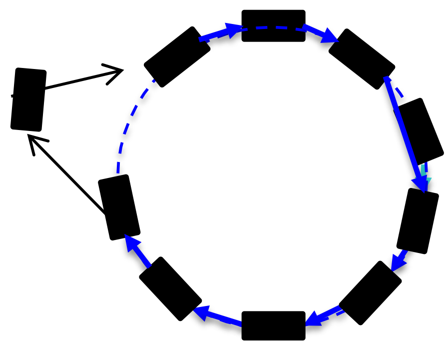
- Alice and Bob agree on a value γ (polylog in n)
  - Algorithm is correct iff γ at least size of 2nd largest component
- Do shell expansion (BFS) from both s and t
- Stopping criteria:
  1. s shell merges with t shell (yes)
  2. No new nodes added in some step (no)
  3. Shell merges with giant component of G_A or G_B (yes)
  4. Shell size exceeds γ . Stop before sending. (yes)

- With a good guess, $\gamma = O(\log n)$, so $O(\log^2 n)$ bits communicated

More Than Two Centers

- Do shell expansion in a loop
- Center that adds a node removes it when it comes back (so each center sees it once)

Query processor



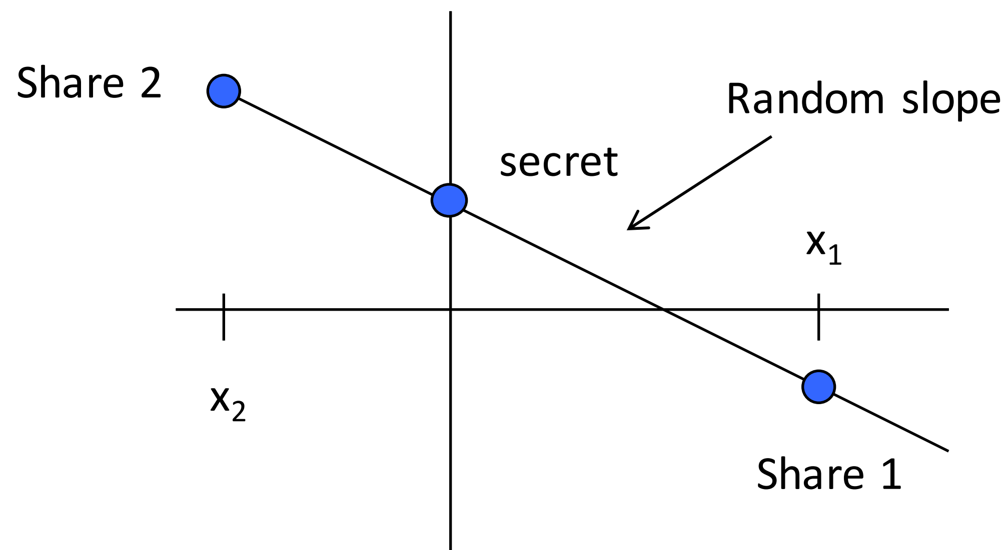
- The query processor starts both the s and t shells (containing only the one node if necessary
- Looks like the 2-processor protocol with all the other processors merged.

Secure Multiparty Computation Version

- Alice and Bob can determine that a path connects s and t without revealing anything about: the path, nodes seen by either party
- Similar to a model used by Brickell and Shmatikov
  - They assume known node names (shared customer lists)
- Secure multiparty computation
  - Usually many parties, small data (circuits, oblivious RAM)
  - We have small number of parties, large data

Tool #1

- Secret sharing
  - Secrets are in a finite field
  - Use a polynomial of degree d to encode a value, d+1 shares
    - All shares reveal secret, d reveals nothing
    - Solution is y intercept, secrets are polynomials at other x
- Key: Given a share of x (called $[x]_i$) and a share of y (called $[y]_i$), can get a share of the sum by adding shares: $[x+y]_i = x_i + y_i$



Tool #2: Secure MUX

$$\mathrm{MUX}(c,a,b) = \begin{cases} a, & c \neq 0, \\ b, & \text{otherwise.} \end{cases}$$

- Need to be able to securely compute shares of MUX(c,a,b), given shares of a,b,c
- Information-theoretically secure protocols if at least 3 centers (Ben-or, Goldwasser, Wigderson)
- For 2 centers need Yao's garbled circuits (crytographic)
  - This is expensive, requires communication

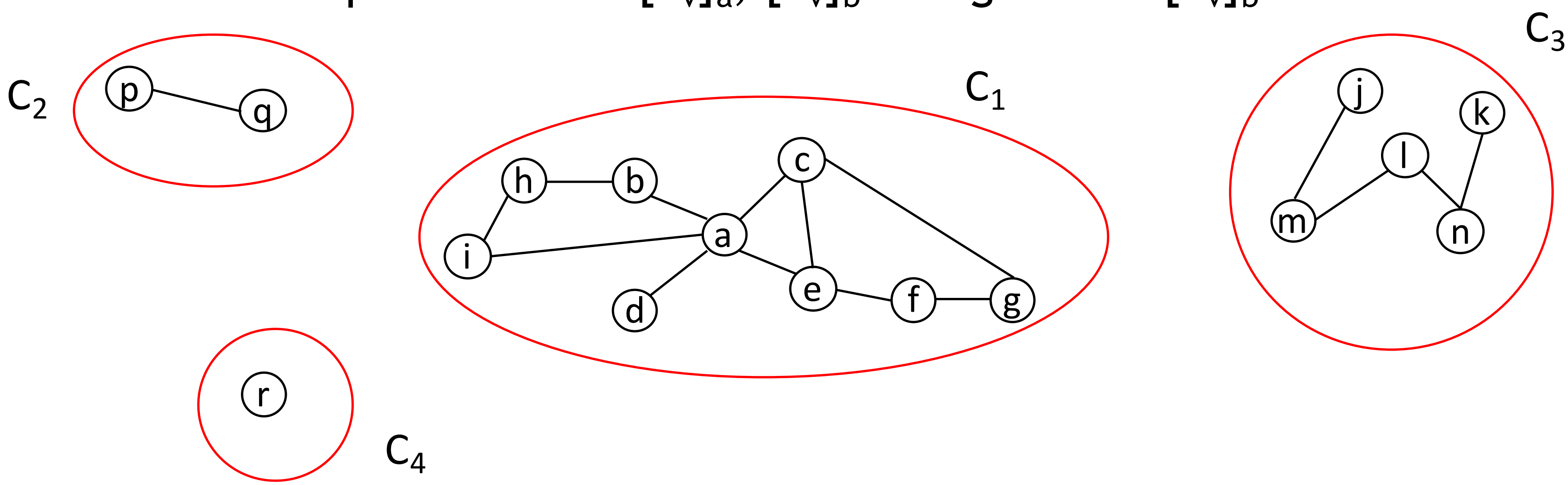
Algorithm Overview

- Secret share component names for each node (both Bob and Alice)
- Secret-shared shell expansion from s
- For each node compute secret-shared binary variable:
  - P(v) is 1 if node v in same component as s, else 0
- In end reveal P(t) by combining secret shares

- Can do this with hidden names

First Version: Shared Node Names

- Alice computes connected components
- x_v is component label for node v
  - x_b=1, x_p=2, x_j=3, x_r = 4
- Alice computes shares $[x_v]_a$, $[x_v]_b$ and gives all $[x_v]_b$ to Bob.



- Bob does the same. His node labels are y_v, shares $[y_v]_a$, $[y_v]_b$. He gives $[y_v]_a$ to Alice.

Constraint on Component Labels

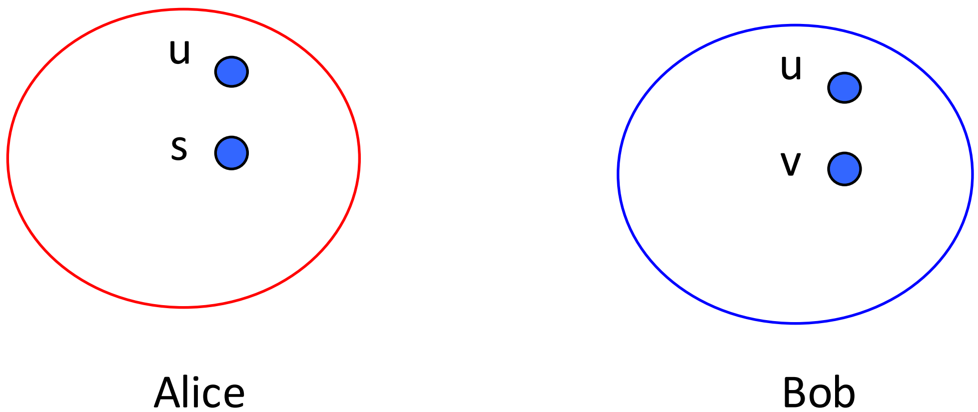
- Let P be a large prime, $P > n^2$ (n is # nodes). Field is integers mod P.
- Pick an $M > n$ such that $M^2 < P$. Require $1 < x_v < M$ for Alice. Bob's labels are tM for some $1 < t < M$.

Key: Alice's labels are different order(s) of magnitude from Bob's:

- Alice's components: 1,2,3
- Bob's components: 1000, 2000, 3000

Propagating Connectivity Information

- P_v is a binary variable set to 0 iff there exists a node u such that $x_u = x_s$ and $y_u = y_v$.

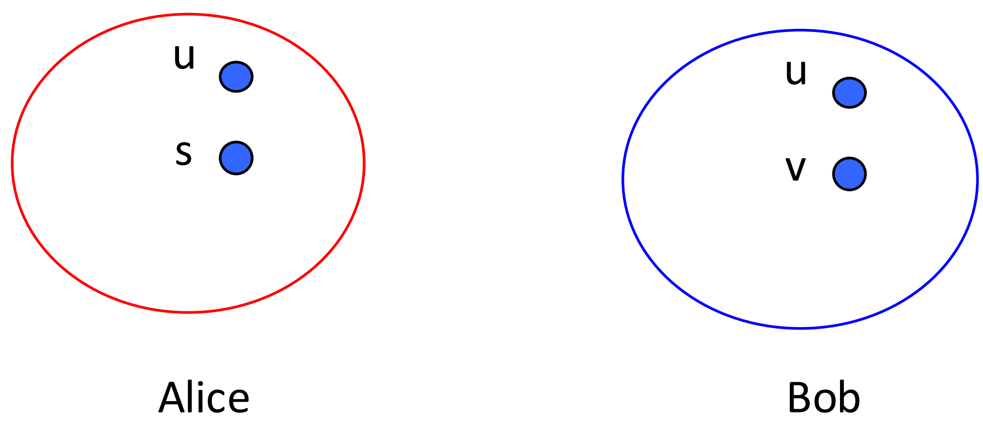


Algorithm 1 OddStep

$P_v = 1$
for node u **do**
    $P_v \leftarrow \mathrm{MUX}((x_s - x_u + y_u - y_v), P_v, 0)$
end for

Propagating Connectivity Information

- Pv is a binary variable set to 0 iff there exists a node u such that $x_u=x_s$ and $y_u=y_v$.

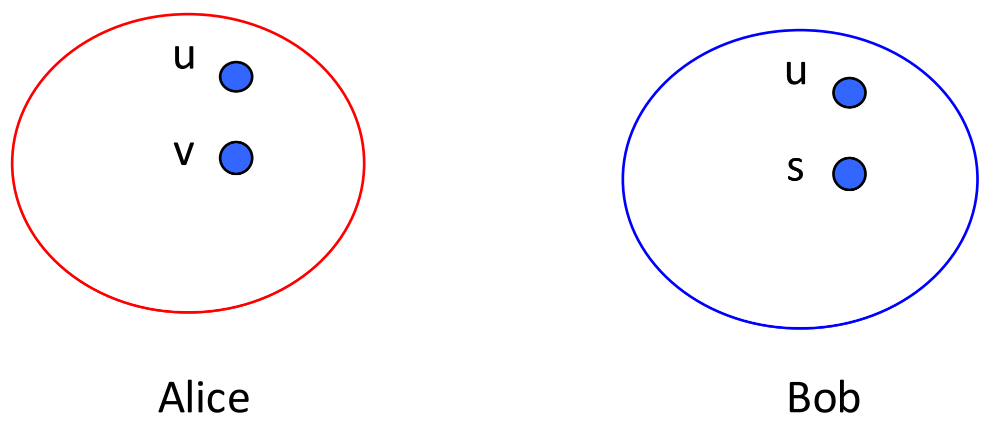


- Update the y_v, to show connectivity to s

$$y_v \leftarrow \mathrm{MUX}(P_v, y_v, y_s)$$

Propagating Other Way Too

- Pv is a binary variable set to 0 iff there exists a node u such that x_u=x_s and y_u=y_v.



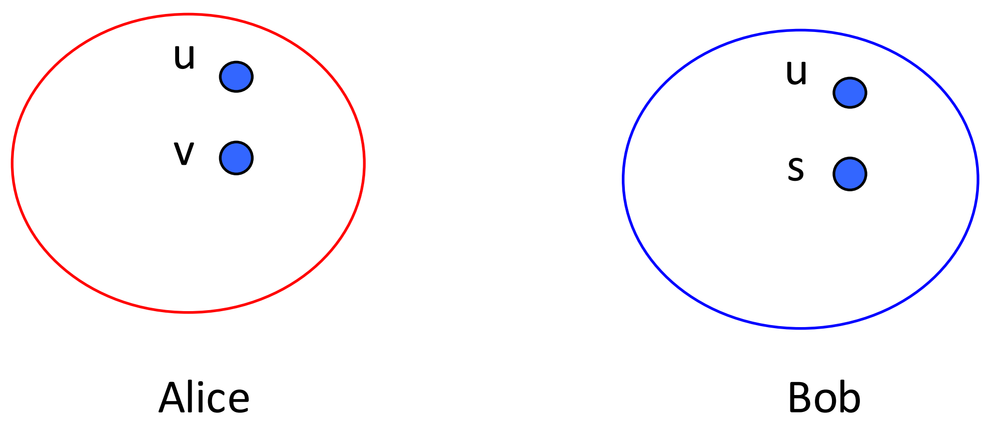
Algorithm 2 EvenStep

```
P_v = 1
for node u do
    P_v ← MUX((y_s − y_u + x_u − x_v), P_v, 0)
end for
```

$P_v = 1$
for node u **do**
$\quad P_v \leftarrow \text{MUX}((y_s - y_u + x_u - x_v), P_v, 0)$
end for

Propagating Connectivity Information

- Pv is a binary variable set to 0 iff there exists a node u such that $x_u=x_s$ and $y_u=y_v$.

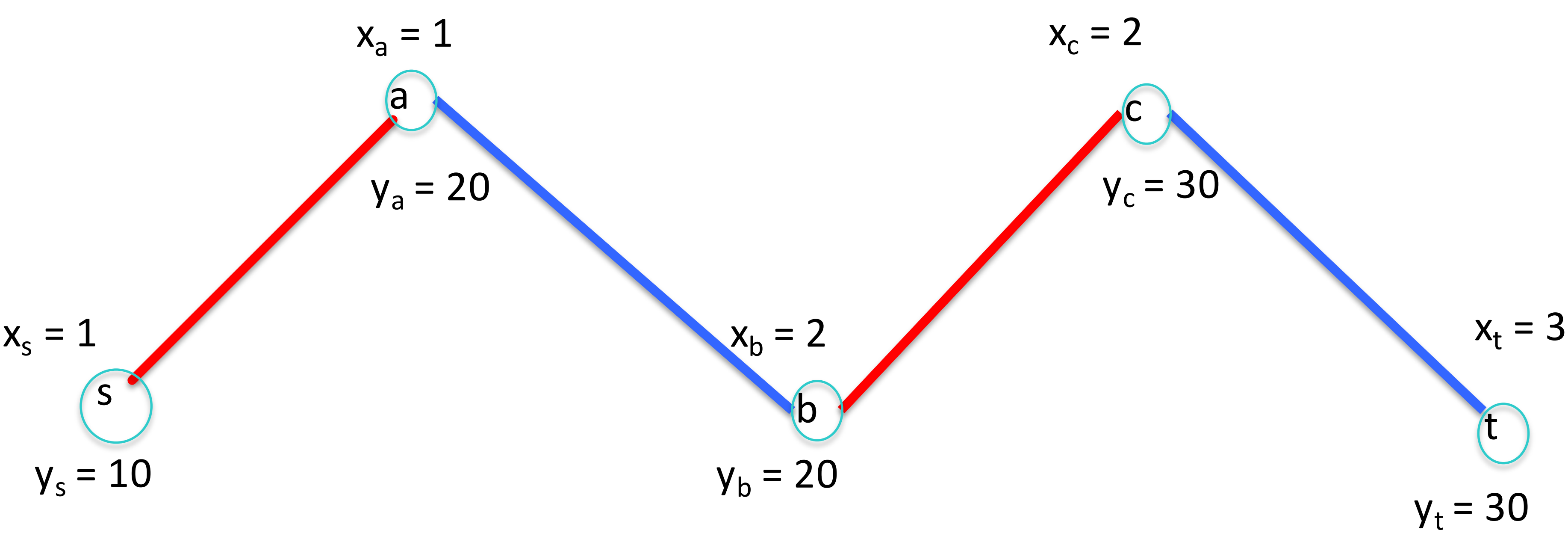


- Update the x_v, to show connectivity to s

$$x_v \longleftarrow \mathrm{MUX}(P_v,\ x_v,\ x_s)$$

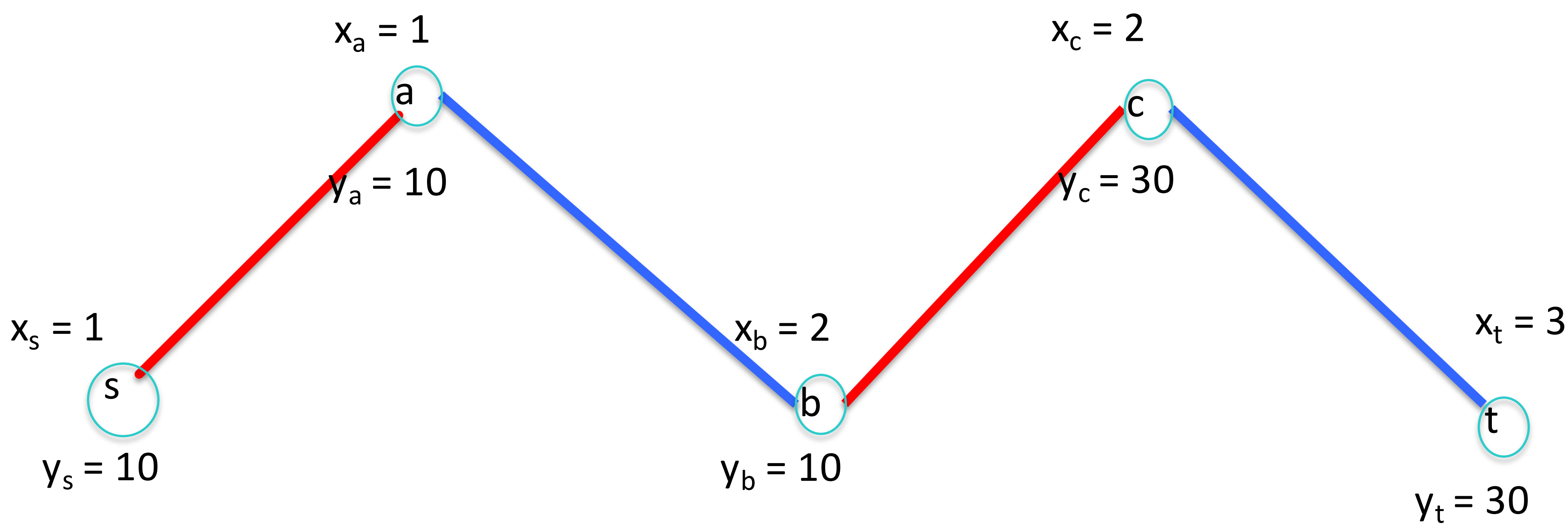
Example

- Here are the labels at the start:



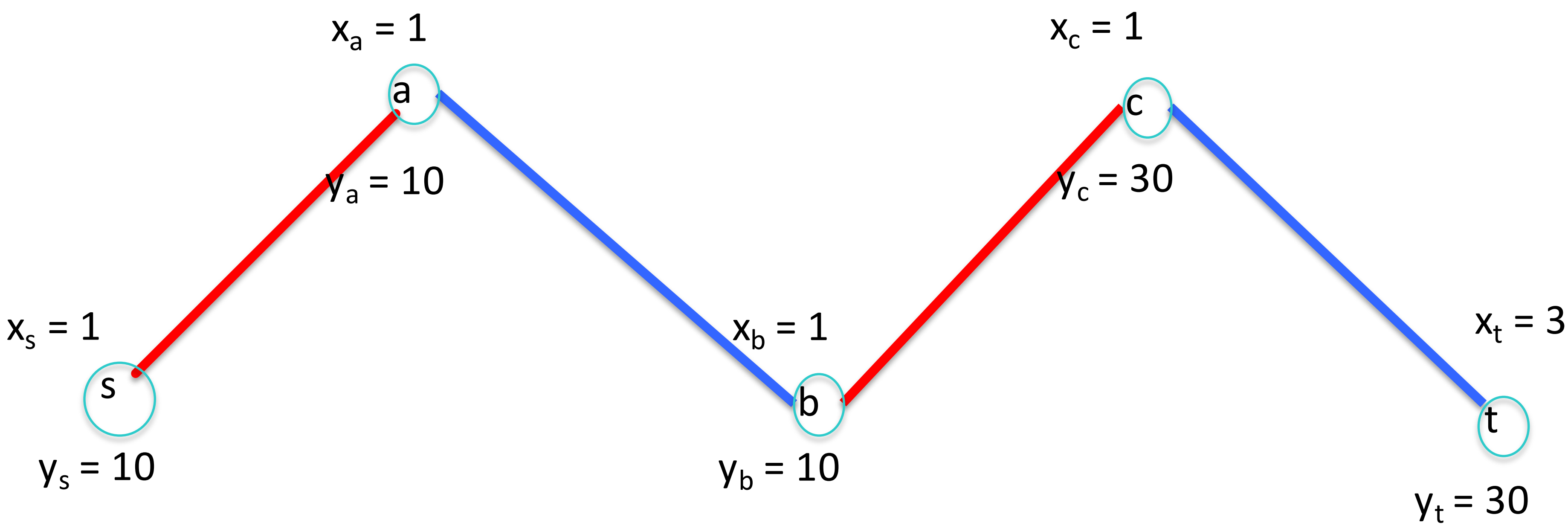
- $P_a = 0$ because $x_s - x_a + y_a - y_a = 0$ $(u = a)$
- $P_b = 0$ because $x_s - x_a + y_a - y_b = 0$ $(u = a)$
- So y_a and y_b are set to y_s

Example



- $P_b = 0$ because $y_s - y_b + x_b - x_b = 0$ $(u = b)$
- $P_c = 0$ because $y_s - y_b + x_b - x_c = 0$ $(u = b)$
- So x_b and x_c are set to x_s

Example



- The next step sets $y_t = 10 = y_s$
- From that point on $P_t = 0$
- After enough steps, compare shares to decode P_t.
- Enough steps: diameter (at most n-1), or j if only care about paths of length at most j

Complexity

- If there are n nodes and (known) diameter d
  - $O(d)$ major steps
  - $O(n^2)$ work (MUXs) per major step
  - But can do work for intermediate node u in parallel so O(n) communication rounds per major step

Hiding Names

- Arrays of names and labels
  - Arbitrary, except s, t are first

Dummy node

		0	1	2	3	4	5	6	7	8	9
Alice	Names	s	t	c	b	q	a	e	α	β	δ
	Labels	x_s	x_t	x_c	x_b	x_q	x_a	x_e	x_α	x_β	x_δ
Bob	Names	s	t	a	q	g	e	h	b	ζ	μ
	Labels	y_s	y_t	y_a	y_q	y_g	y_e	y_h	y_b	y_ζ	y_μ

Secret-Shared Permutation

- Secret-shared y' array effectively permutes Bob's labels to match

Alice

s	t	c	b	q	a	e	α	β	δ	Names

y_s	y_t	0	y_7	y_3	y_2	y_5	0	0	0	Bob's Permuted Names y'

Bob

s	t	a	q	g	e	h	b	ζ	μ	Names
y_s	y_t	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	Labels

Secret Names

- Compute using MUX (just comparisons of unknown objects)
- Then use y' instead of y in previous algorithm

Secret-shared names

for j **do**
  $y'_j \leftarrow 0$
  for i **do**
    $y'_j \leftarrow y'_j + \text{MUX}(\hat{x}_j - \hat{y}_i,\ 0,\ y_i)$
  end for
end for

Then the parties compute shares of P_k as

$P_k \leftarrow 1$
for j **do**
  $P_k \leftarrow \text{MUX}(x_s - x_j + y'_j - y'_k,\ P_k,\ 0)$
end for

Concluding Thoughts

- Exploiting social network structure
  - Degree distribution
  - Community structure (clustering coefficients)
  - Etc
- We have considered evolutionary/social properties
  - Beware of non-human behavior in online social networks

J. Berry, M. Collins, Aaron Kearns, C. Phillips, J. Saia, R. Smith, “Cooperative computing for autonomous data centers,” Proceedings of the IEEE International Parallel and Distributed Processing Symposium, May 2015.