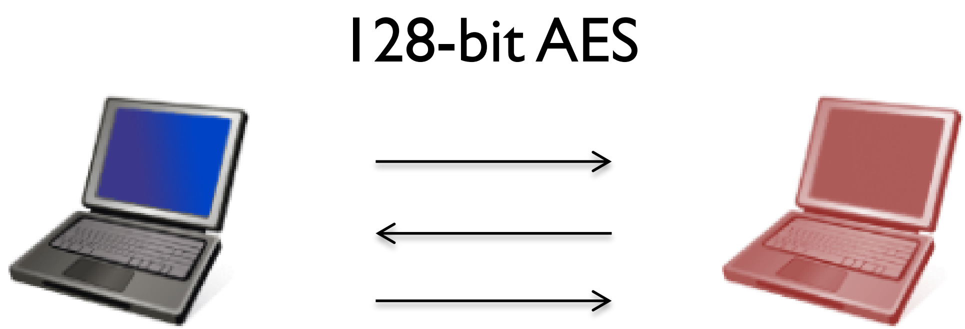
LARGE-SCALE SECURE COMPUTATION

MPC FOR PARALLEL RAM PROGRAMS

Elette Boyle
Technion

Based on join works with Kai-Min Chung and Rafael Pass

Multi-Party Computation (MPC)

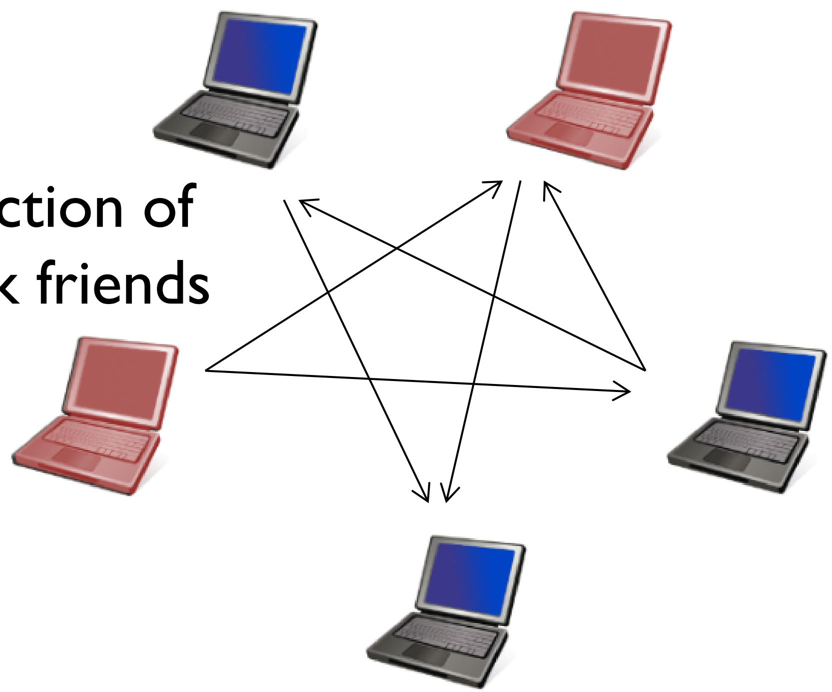


[GMW87] – Computational Setting

[BGW88, CCD88] – Information Theoretic Setting with Secure Channels

This Talk: Large-Scale MPC

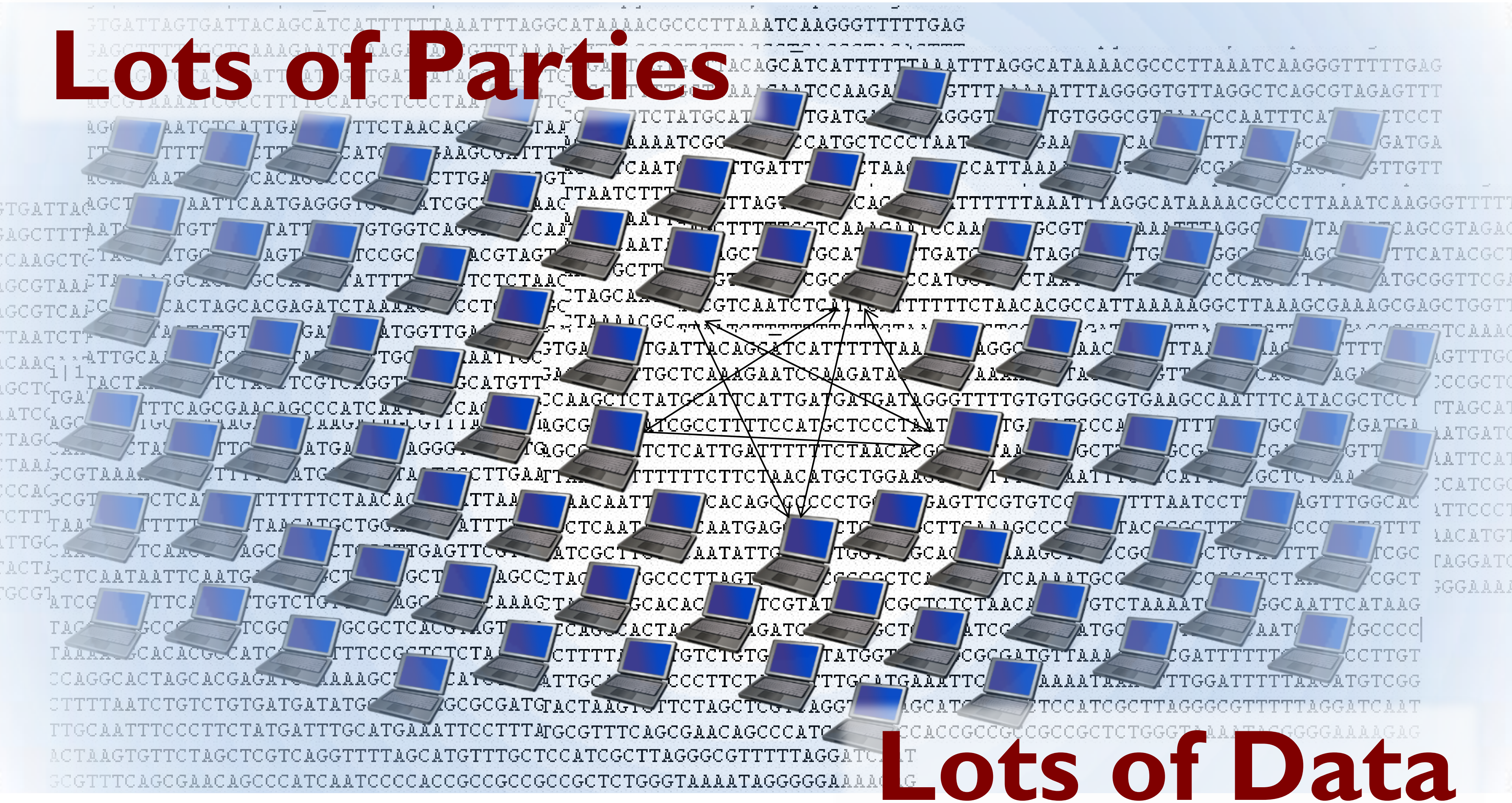
Set intersection of
Facebook friends



[GMW87] – Computational Setting

[BGW88, CCD88] – Information Theoretic Setting with Secure Channels

This Talk: Large-Scale MPC



MPC Efficiency Metrics

Communication Memory Computation

How are these affected in the large-scale setting?

Costs of Communication

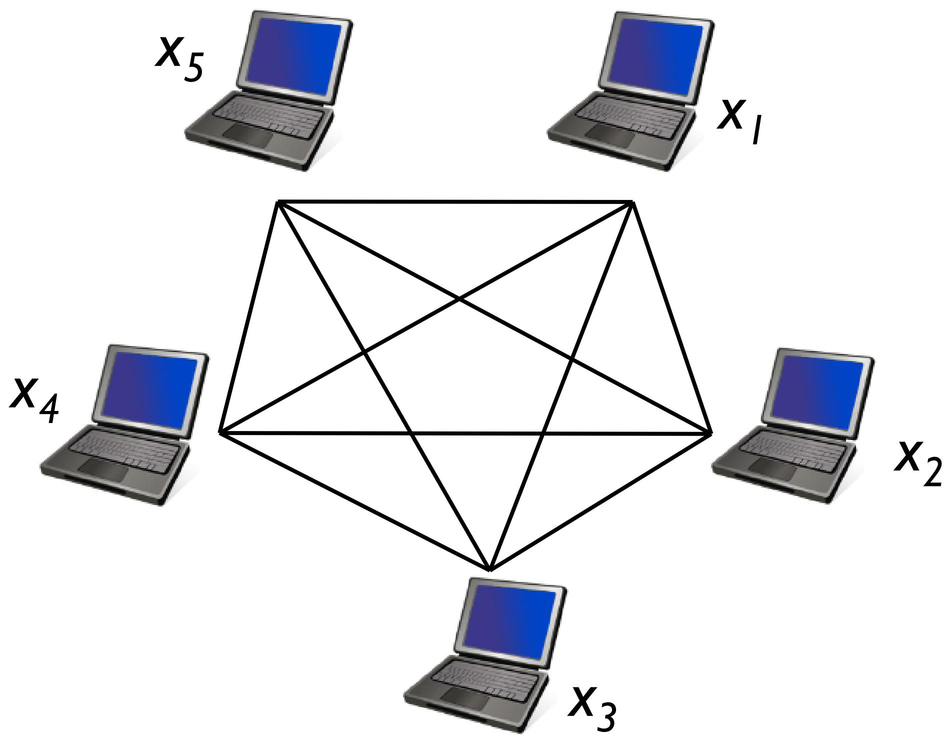
- # of bits communicated
- # of sequential rounds
- …Who a party is speaking to

Nearly all protocols:
Every party speaks to every party

Communication: *Locality* Metric

[BGT13]

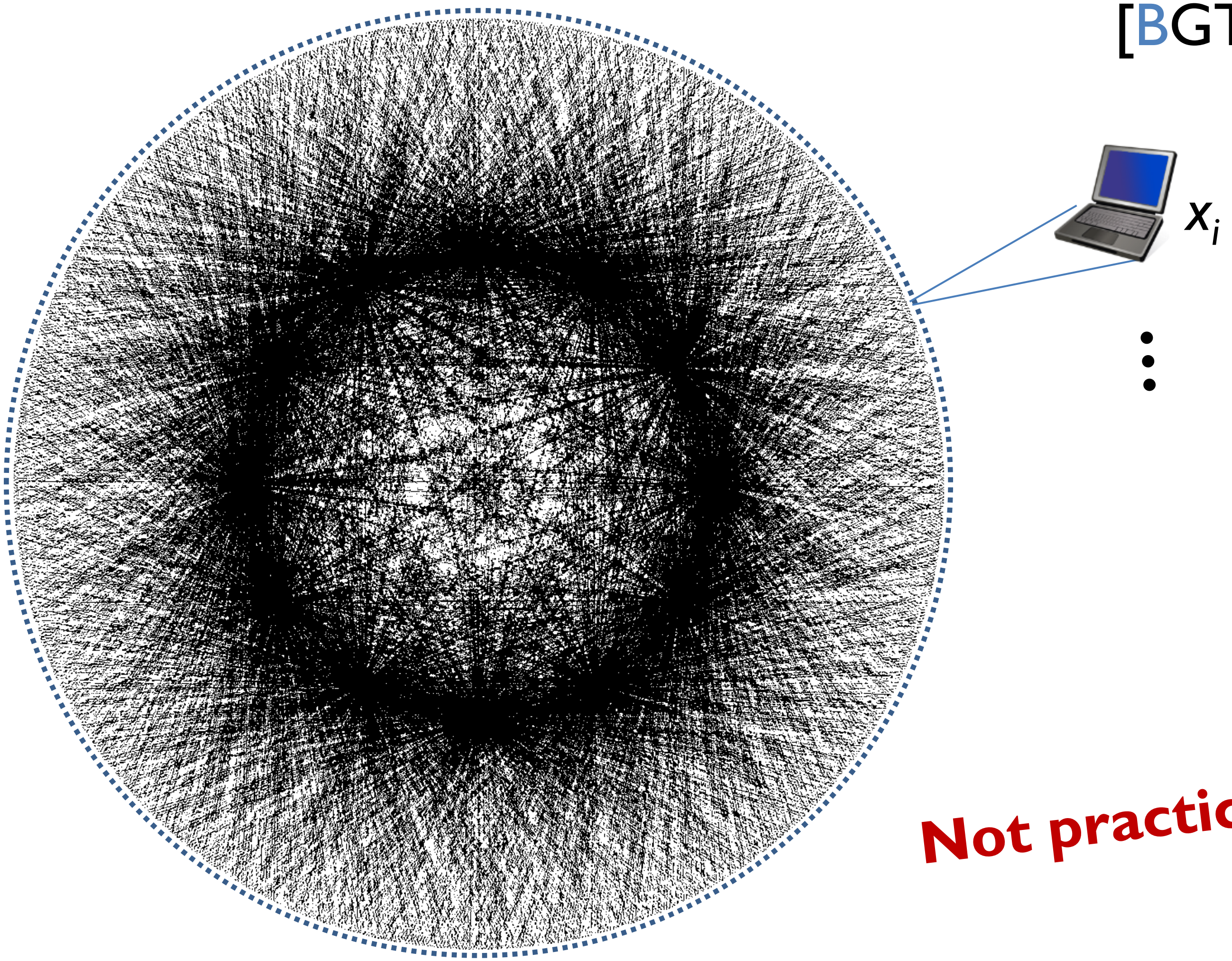
parties:
$n = 5$



Communication: *Locality* Metric

[BGT13]

\# parties:
$n = 10{,}000$

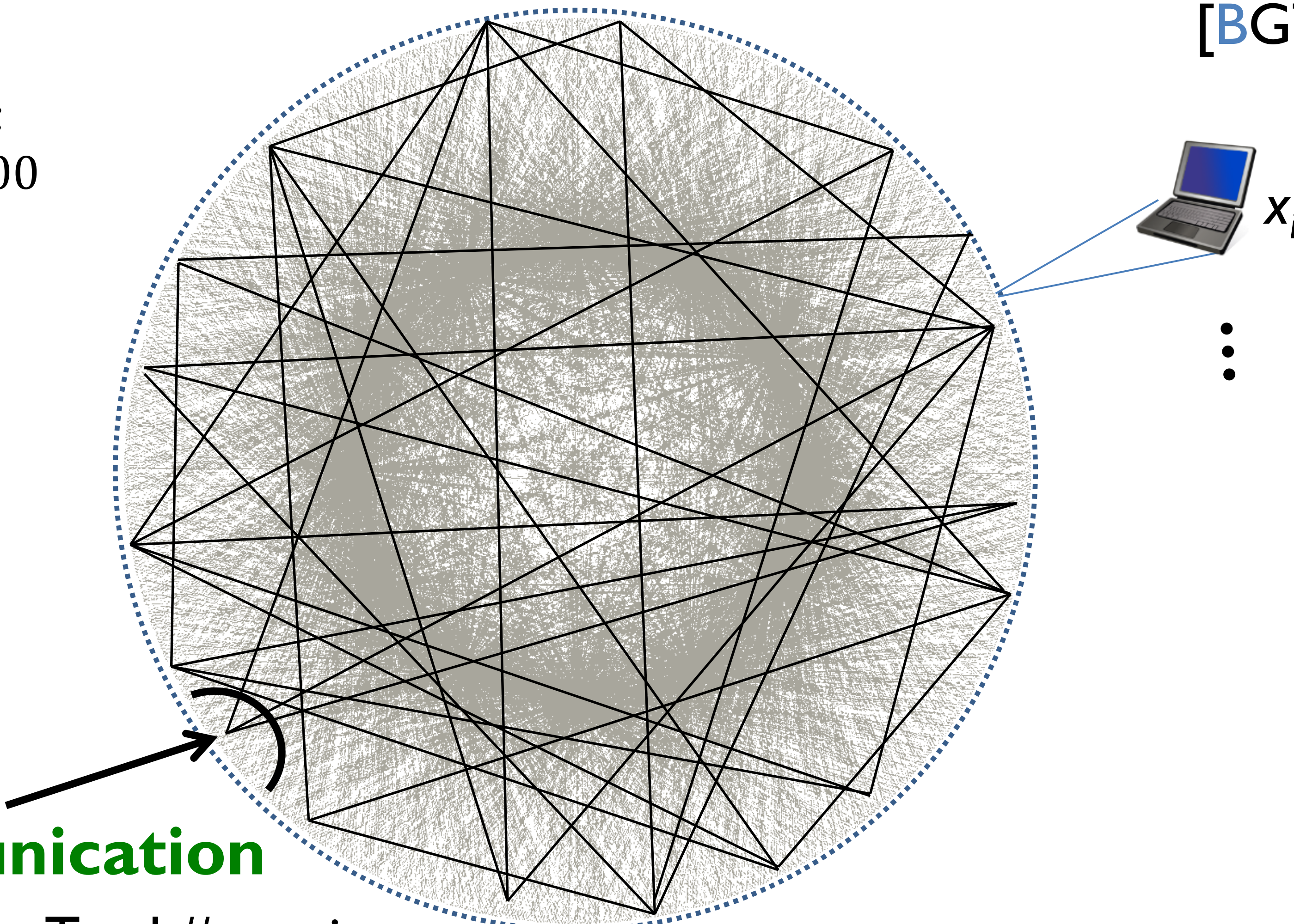


Not practical!

Communication: *Locality* Metric

[BGT13]

parties:
$n = 10{,}000$



Communication Locality: Total # parties each party communicates with throughout protocol lifetime

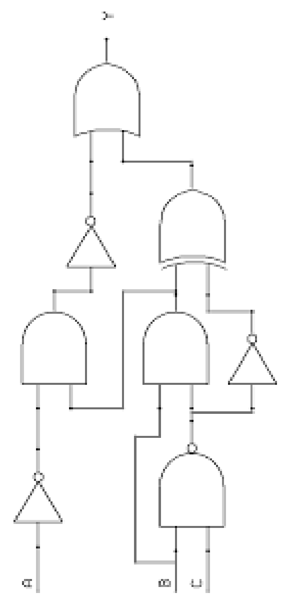
Memory: Balancing the Burden

- Combined data size is huge!
- Want: Memory requirement per party

$\approx$ (his input + Space(Π)/n)

Computation: Going Beyond Circuits

f



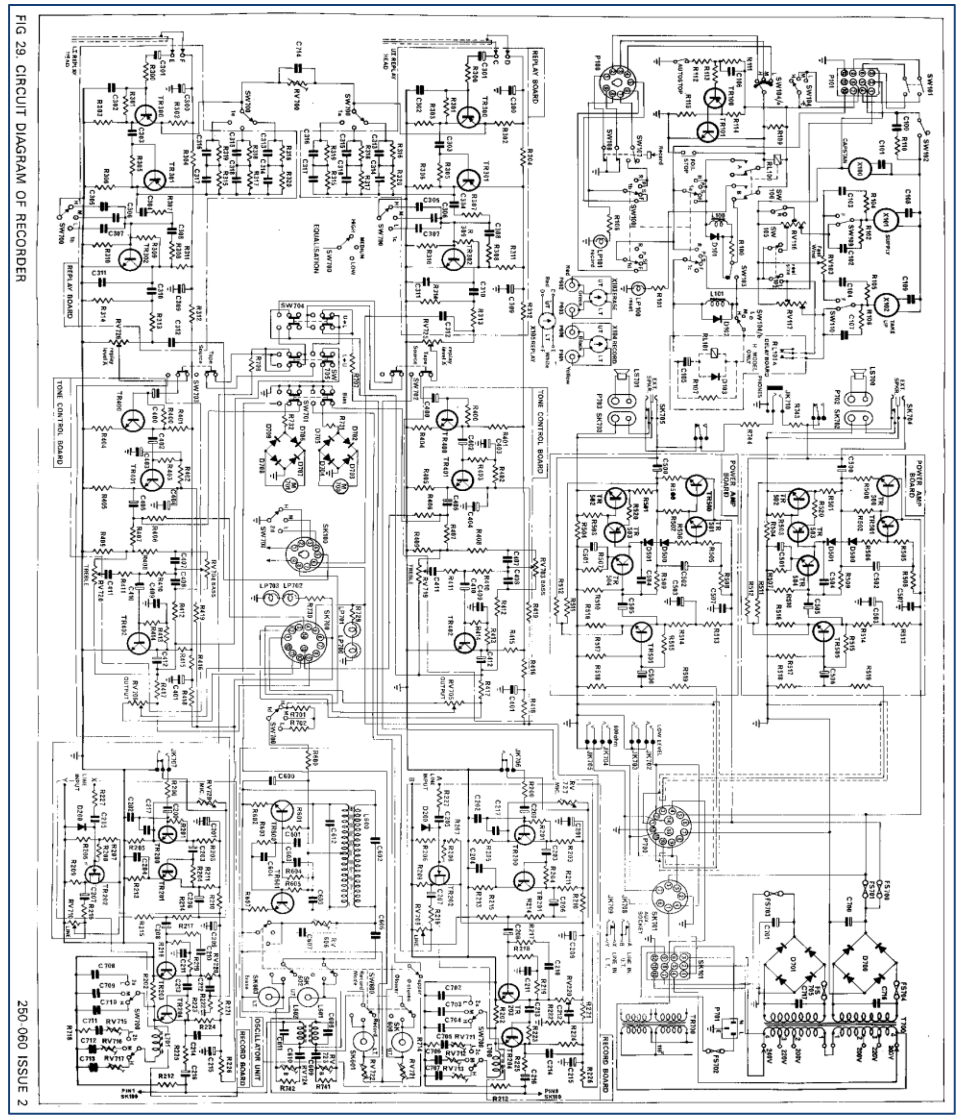
MPC protocol for securely computing the circuit

Optimizing this transformation yields better MPC efficiency

Computation: Going Beyond Circuits

Program

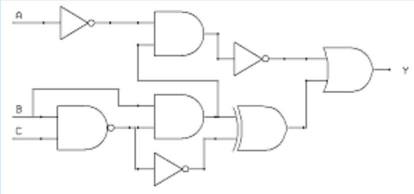
```
BINARY-SEARCH(x, T, p, r)
low = p
high = max(p, r + 1)
while low < high
    mid = ⌊(low + high)/2⌋
    if x ≤ T[mid]
        high = mid
    else low = mid + 1
return high
```



Generically:
Blow up by factor of
entire database size!

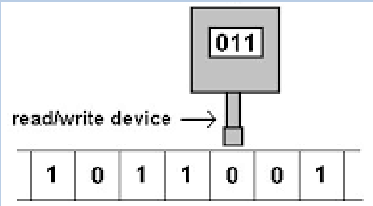
Models of Computation 101

- Circuits

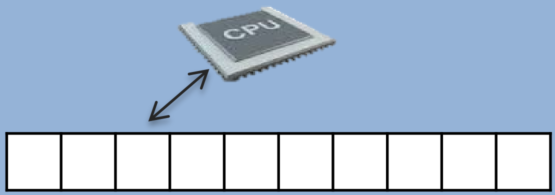


AND, OR, NOT gates

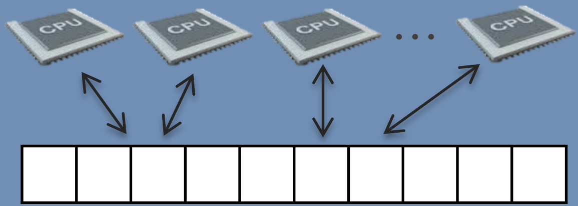
- Turing Machines



- RAM Machines



- Parallel RAM Machines



Computation: Going Beyond Circuits

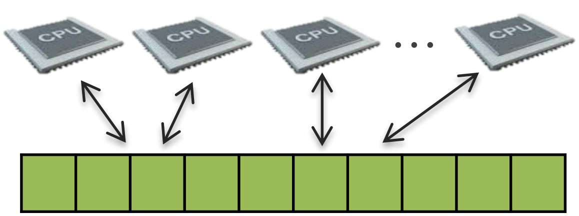
Large-scale computations *f* leverage
random access and ***parallelism***

- Circuit (and TM) model for *f* not appropriate!

Computation: Going Beyond Circuits

Large-scale computations *f* leverage
random access and ***parallelism***

- Circuit (and TM) model for *f* not appropriate!
- RAM model for *f* loses parallelism!
- **Parallel RAM (PRAM) Model**



Rough History of Prior MPC Work

- Circuits model

  E.g.: Original protocols [GMW87, BGW88, CCD88,…], Scalable MPC [DI06, DN07, DIK+08, DIK10, DKMS12, ZMS14], MPC on incomplete networks [CGO10, CGO12], MPC based on FHE / Obfuscation [Gen09, AJL+12, MSS13, GGHR14], Optimized MPC for practice [BNP08, KS08, LPS08, NO09, LP11, BDOZ11, DPSZ12, NNOS12, L13, FJN+13, ALSZ13, DZ13, LR14, ZRE15,…]

- RAM model
  - 2-PC [OS97, GKK+11, LO13, GGHJ+13, GHRW14, WHHSS14]
  - Extensions to MPC [DMN11] **don't scale with *n***

- PRAM model (nothing)

Eg: Per-party memory requirement ~ size of *all parties'* inputs

Asymptotically

The Goal:
Efficient MPC for PRAM

n-party MPC for PRAMs Π

Needed for security

Time Steps - *Parallel* Time(Π)

Per-party Computation - Comp(Π)/n + His input

Per-party Memory - His input + Space(Π)/n

Comm Locality - 1

Theorem [BCP14,BCP15]:

n-party MPC for PRAMs Π

$\tilde{O}$ = polylog(n)

Rounds - $\tilde{O}$(*Parallel* Time(Π))

Per-party Computation - $\tilde{O}$(Comp(Π)/n)

Per-party Memory - $\tilde{O}$(His input + Space(Π)/n)

Comm Locality - $\tilde{O}$(...)

$\tilde{O}$(His input) + BC /party

Given a 1-time (reusable) preprocessing stage

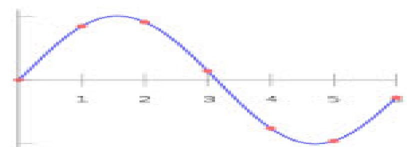
Static corruptions, 2/3+ε honest parties, Unconditional security

The Construction

For Large Data, Many Parties…

- Step 1: *Secret Share* inputs across parties

  Eg: evaluations of random polynomial st $p(0)=s$ [Sha79]



Problem 1: Everyone talks to everyone
Problem 2: Everyone stores all inputs

- Step 2: Evaluate gate-by-gate on shares

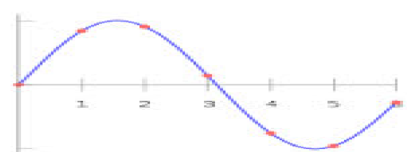
  (sometimes with communication)

Problem 3: Computation ~ Circuit Size

Consider a Simpler Problem: Large Data, Few Parties

- <u>Step 1</u>: *Secret Share* inputs across parties

  Eg: evaluations of random polynomial st *p(0)=s* [Sha79]



Problem 1: Everyone talks to everyone

Problem 2: Everyone stores all inputs

- <u>Step 2</u>: Evaluate gate-by-gate on shares

  (sometimes with communication)

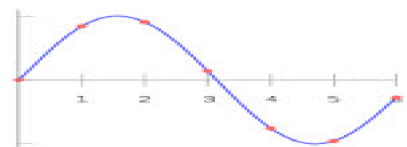
Problem 3: Computation ~ Circuit Size

Consider a Simpler Problem:
Large Data, Few Parties

- Step 1: *Secret Share* inputs across parties

  Eg: evaluations of random polynomial st $p(0)$=s [Sha79]

$p(0)$=secret



$p(i)$=Party i's share

Problem 1: Everyone talks to everyone
Problem 2: Everyone stores all inputs

These are ok!

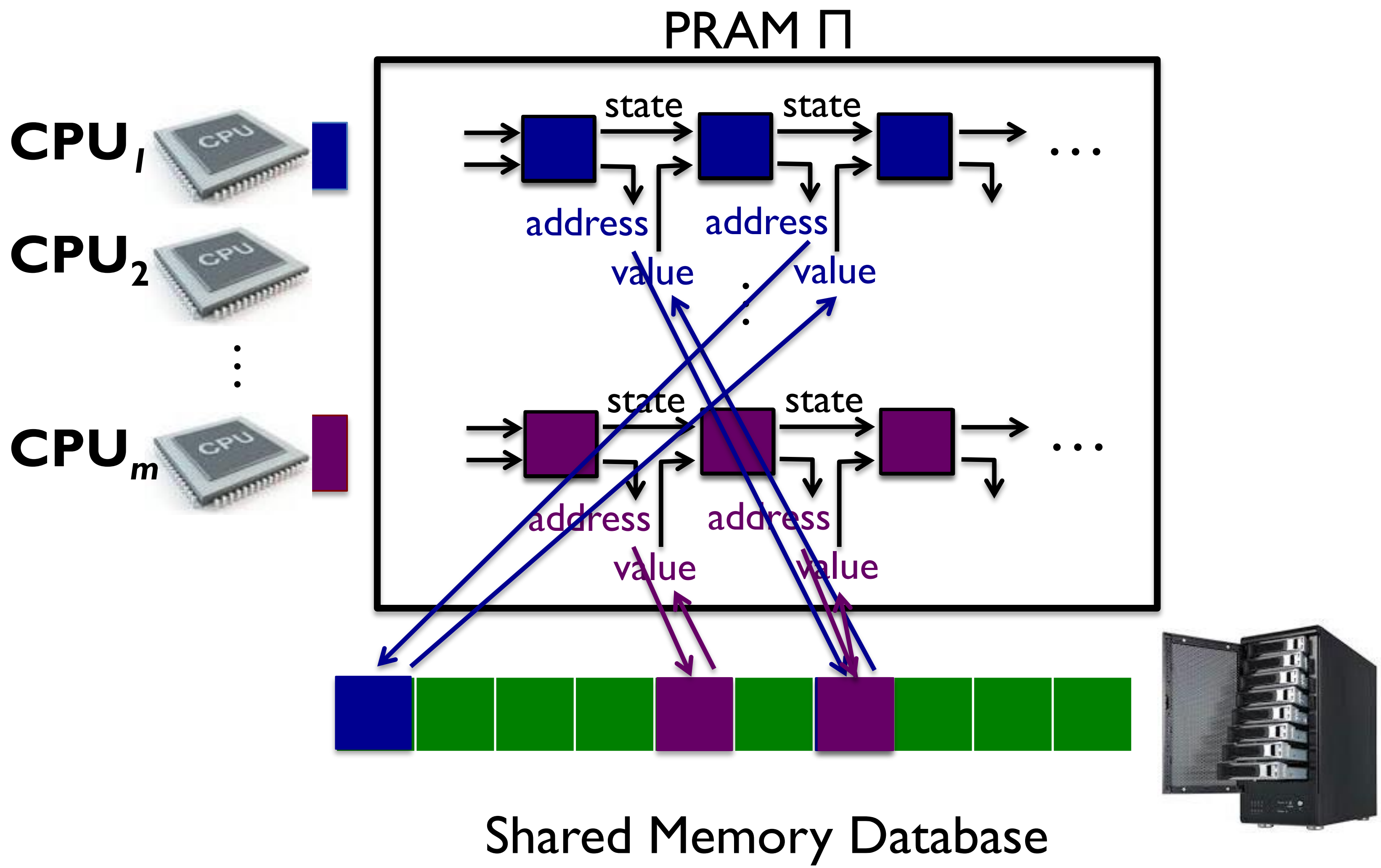
- Step 2: Evaluate gate-by-gate on shares

  (sometimes with communicati

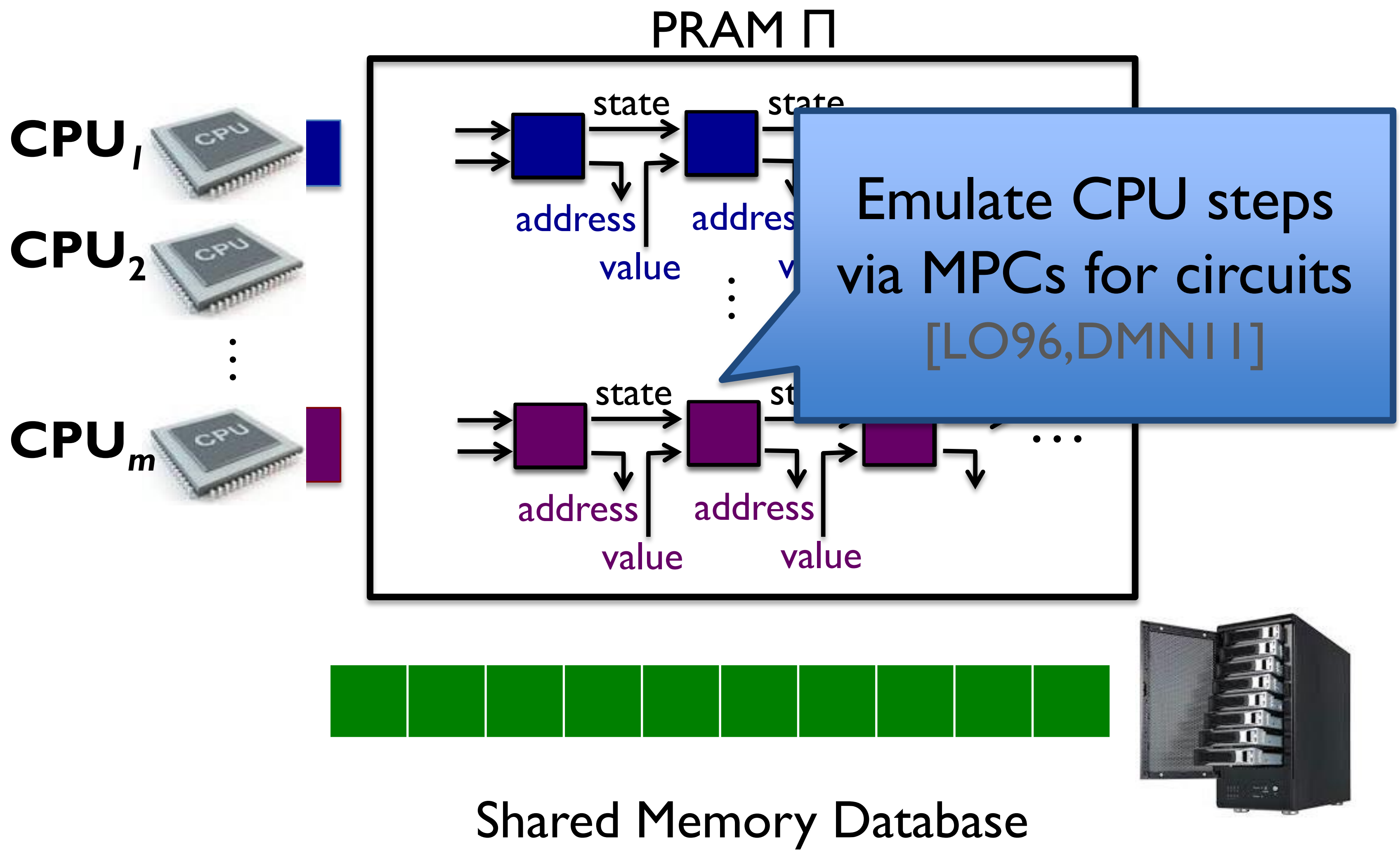
Problem 3: Computation ~ Circuit Size

Wanted:
Comp ~ |PRAM|

How PRAM Works



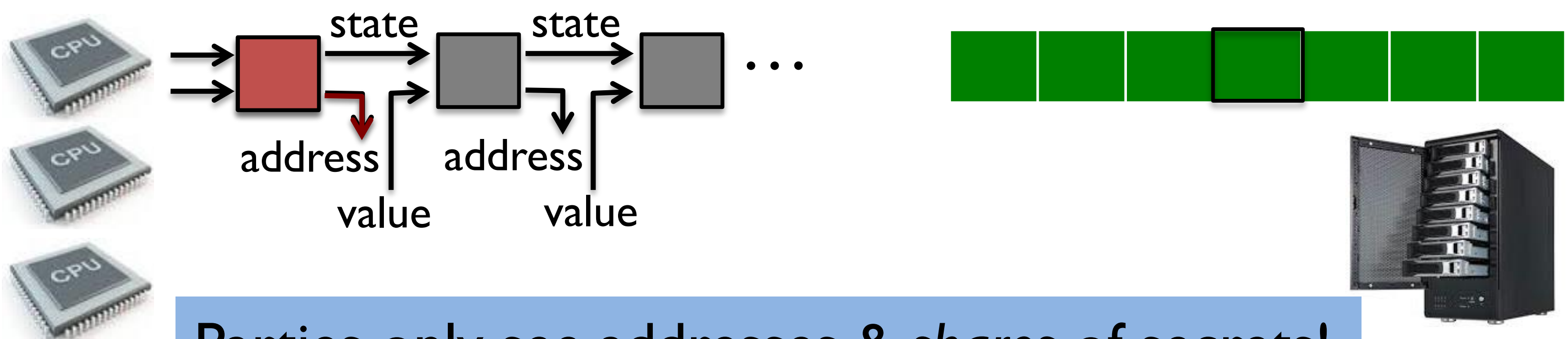
MPC for PRAM: First Idea



MPC for PRAM: First Idea

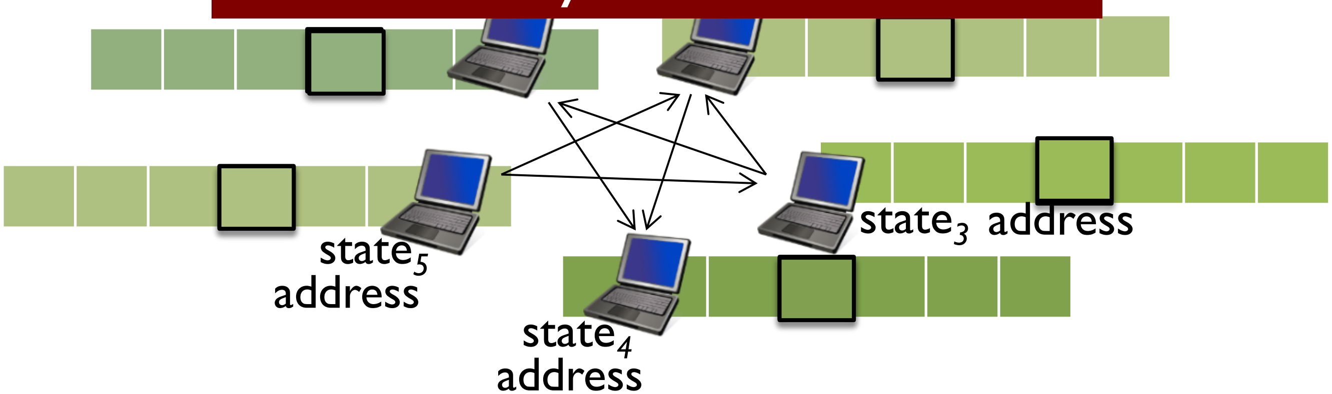
à la [LO06, DMN11]

- Step 1: *Secret Share* inputs across parties
- Step 2: Emulate PRAM CPU steps via small-scale MPCs



Parties only see addresses & *shares* of secrets!

Addresses may leak information!

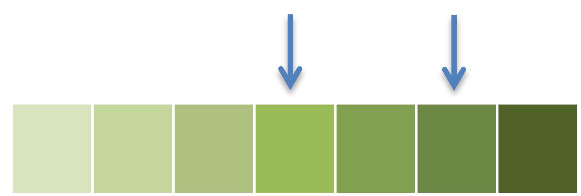


Memory Access Patterns May Leak Information!

Toy Example: Binary Search for 100

Leaks Information!

Access mem 4
Access mem 6
Stop



Reveals: $x_4 < 100$
$x_6 = 100$

Wanted:
PRAM $\rightarrow$ **Oblivious** PRAM

"Oblivious" = memory access patterns appear independent of data

Toy Example: Binary Search for 100

Oblivious Program Compilers

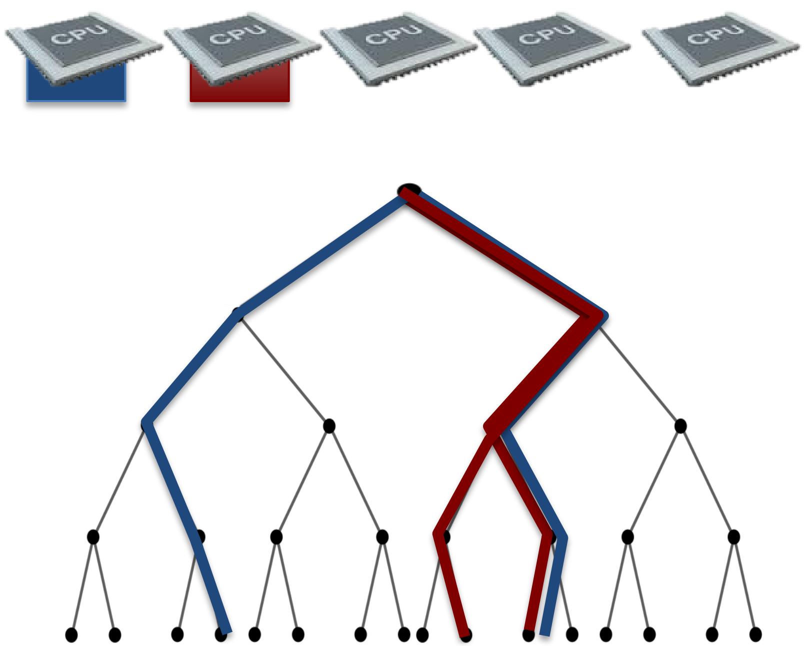
Program from class C → ***Oblivious*** Program from class C

History:

M = memory size

- Turing Machines: log(M) overhead [PF 79]
- RAM programs: polylog(M) overhead [Gol86, Ost90, GO96, Ajt10, DMN11, SCSL11, CP13, GGHJ+13, SDSF+13]
- PRAM: polylog(M) overhead [BCP14]

Core Problem: Supporting Parallel Accesses!



Can't afford for CPUs to take turns!

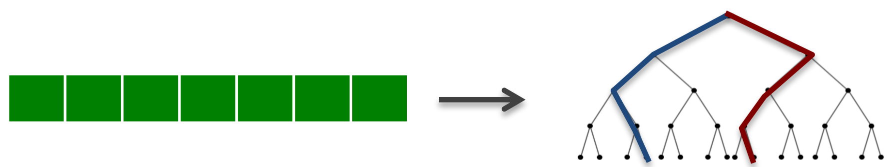
Storing multiple copies causes consistency issues!

Reveals lookup collision!

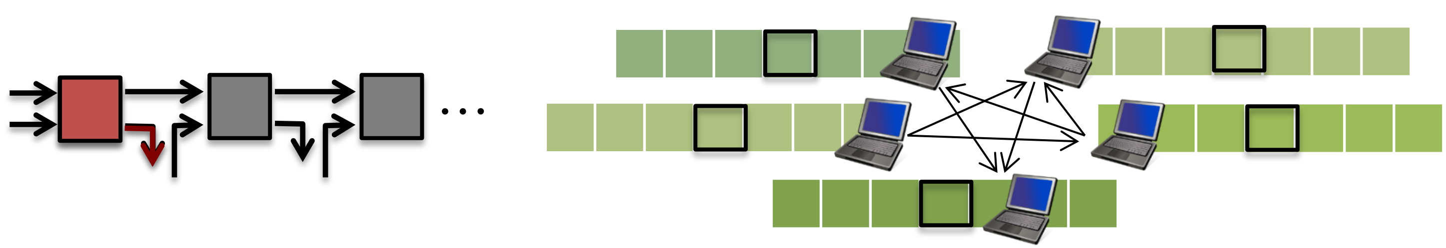
New Protocol:
(Few-Party) **MPC for PRAM**

- <u>Step 1</u>: *Secret Share* inputs across parties

- <u>Step 2</u>: PRAM → Oblivious PRAM



- <u>Step 3</u>: Emulate OPRAM via small-scale MPCs



And for Large Data and **Many Parties**…

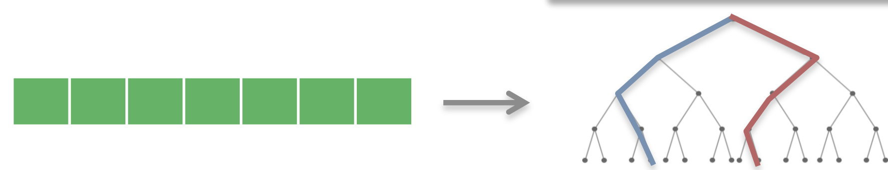
- Step 1: *Secret Share* inputs across parties

Problem 1: Everyone talks to everyone
Problem 2: Everyone stores all inputs

For another time…

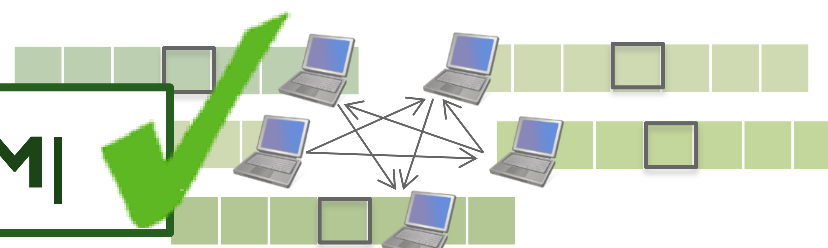
… while load balancing!

- Step 2: PRAM → Oblivious PR



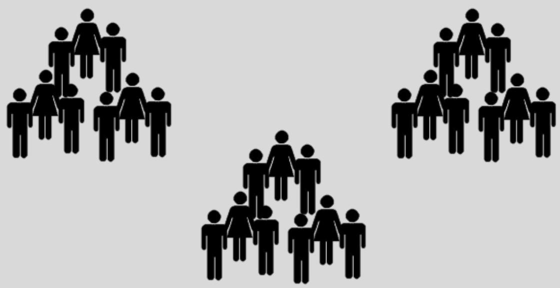
- Step 3: Emulate OPRAM via small-scale MPCs

Computation ~ |PRAM|

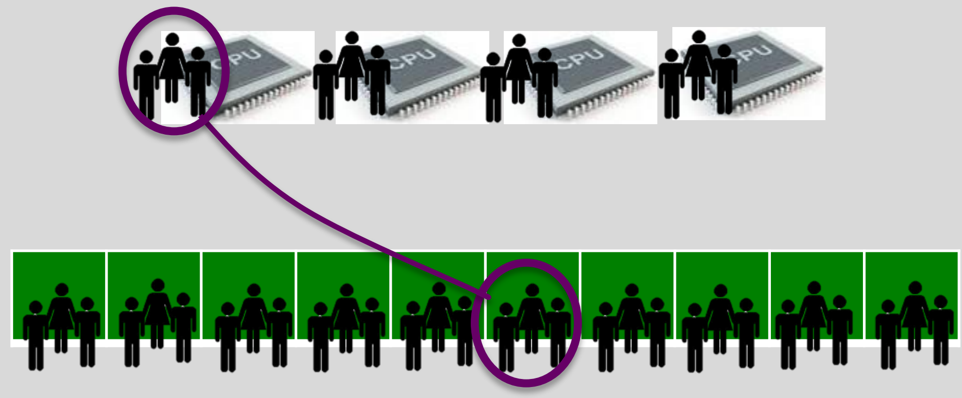


Teaser of Additional Techniques

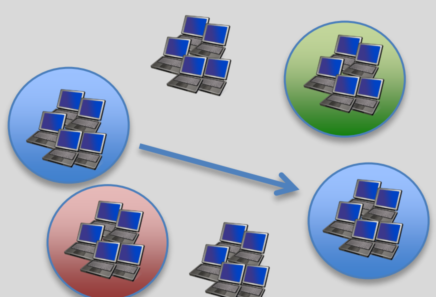
Electing Committees



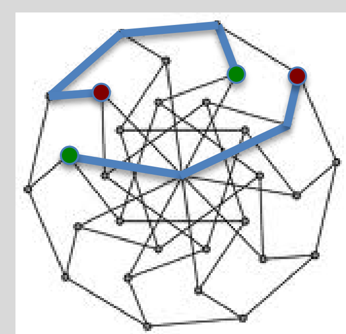
***Distributed* OPRAM**



Load-Balancing via Job Passing



Load-Balanced Routing over Expander Graphs



Future Directions

- "O**P**RAM is the new ORAM"

  \- me

- Pushing Large-Scale MPC toward Practicality

Leveraging computational assumptions? Adaptive security?

Improving broadcast with locality? Honest minority? Targeted protocols?

MPC for MapReduce? Asynchronous models?