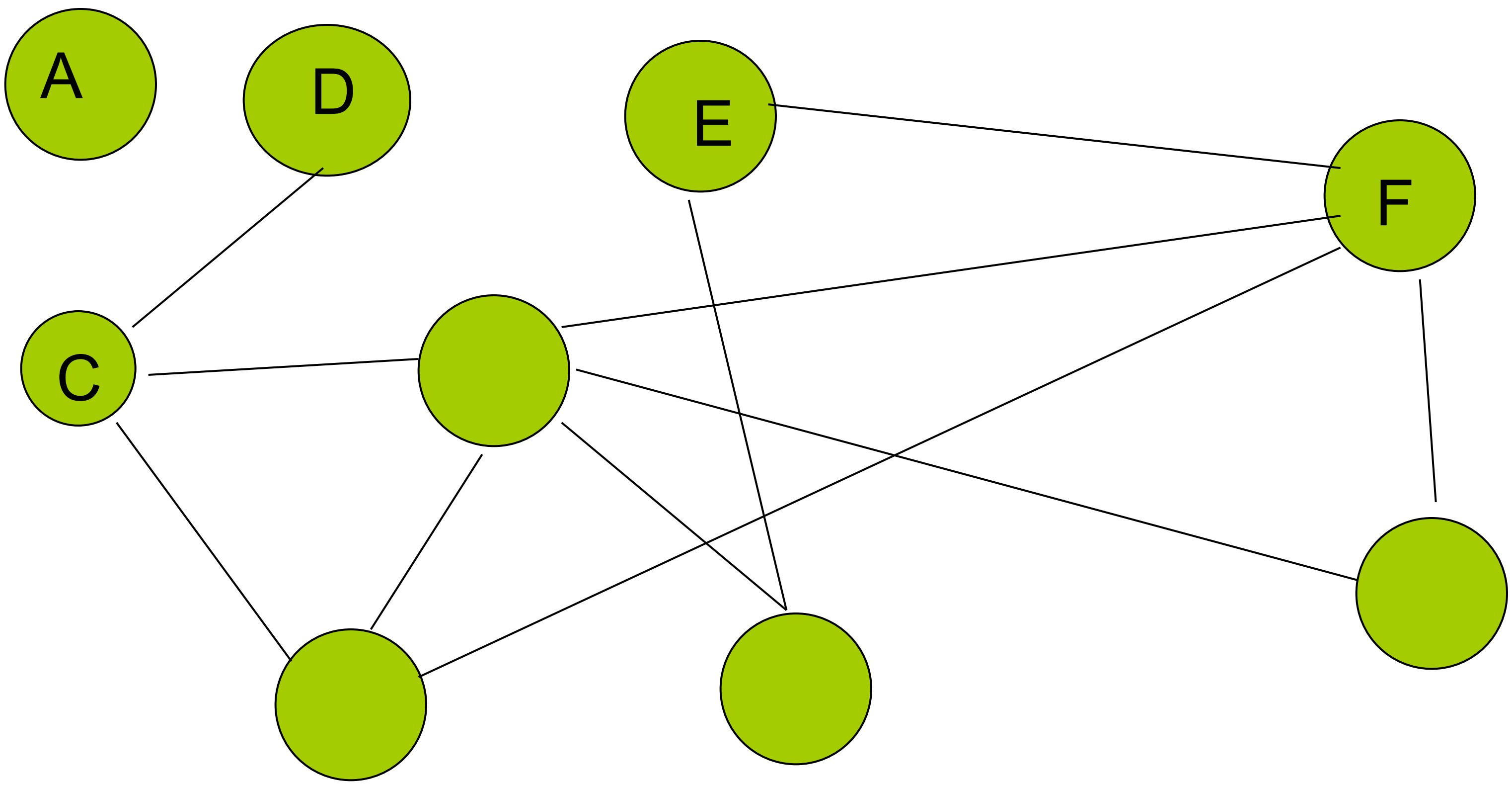
Dynamic Graph Connectivity in polylogarithmic worst case time

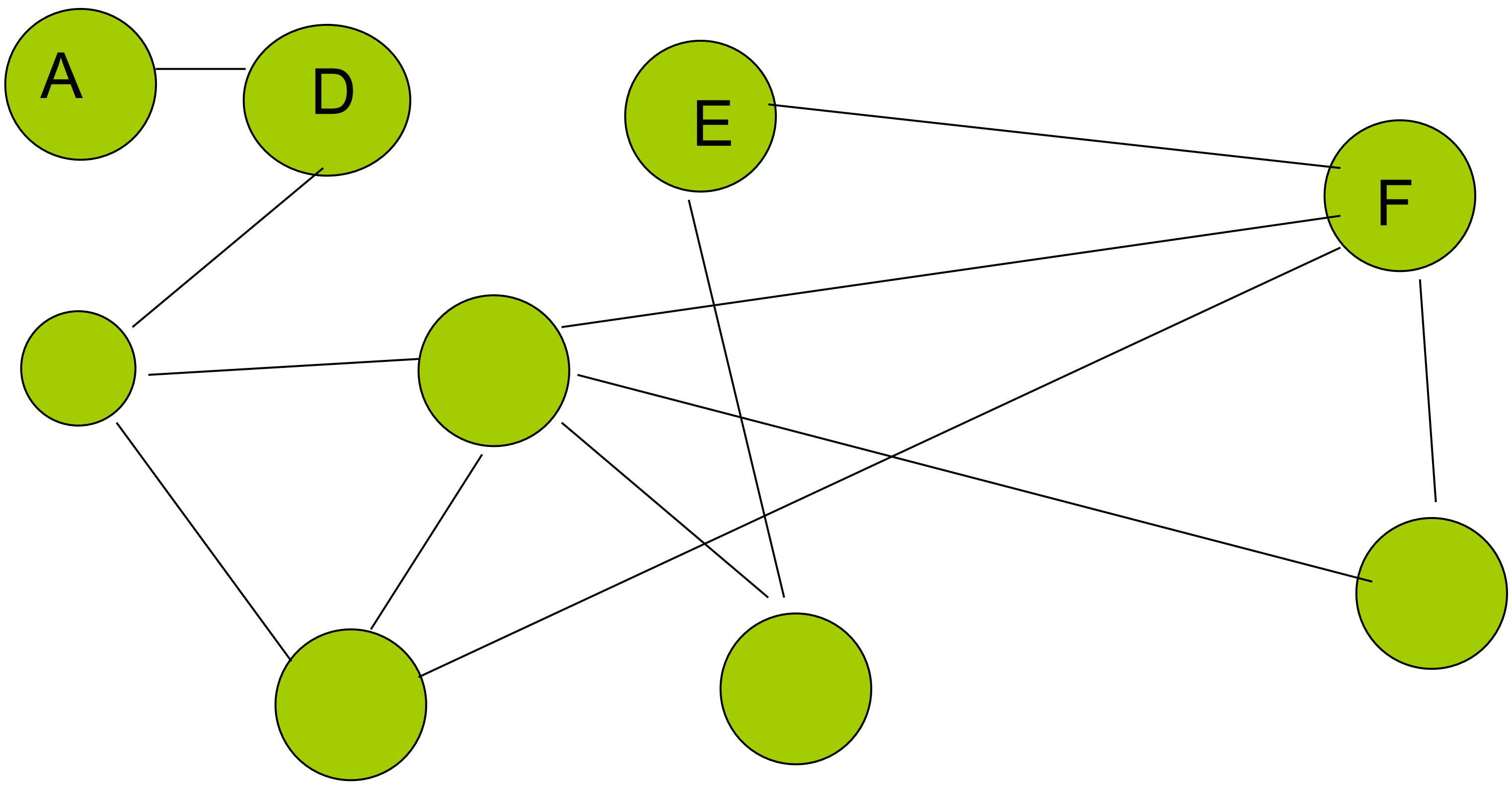
Bruce Kapron, Valerie King and Ben Mountjoy
University of Victoria,
Victoria,Vancouver Island, BC

Graph with n nodes

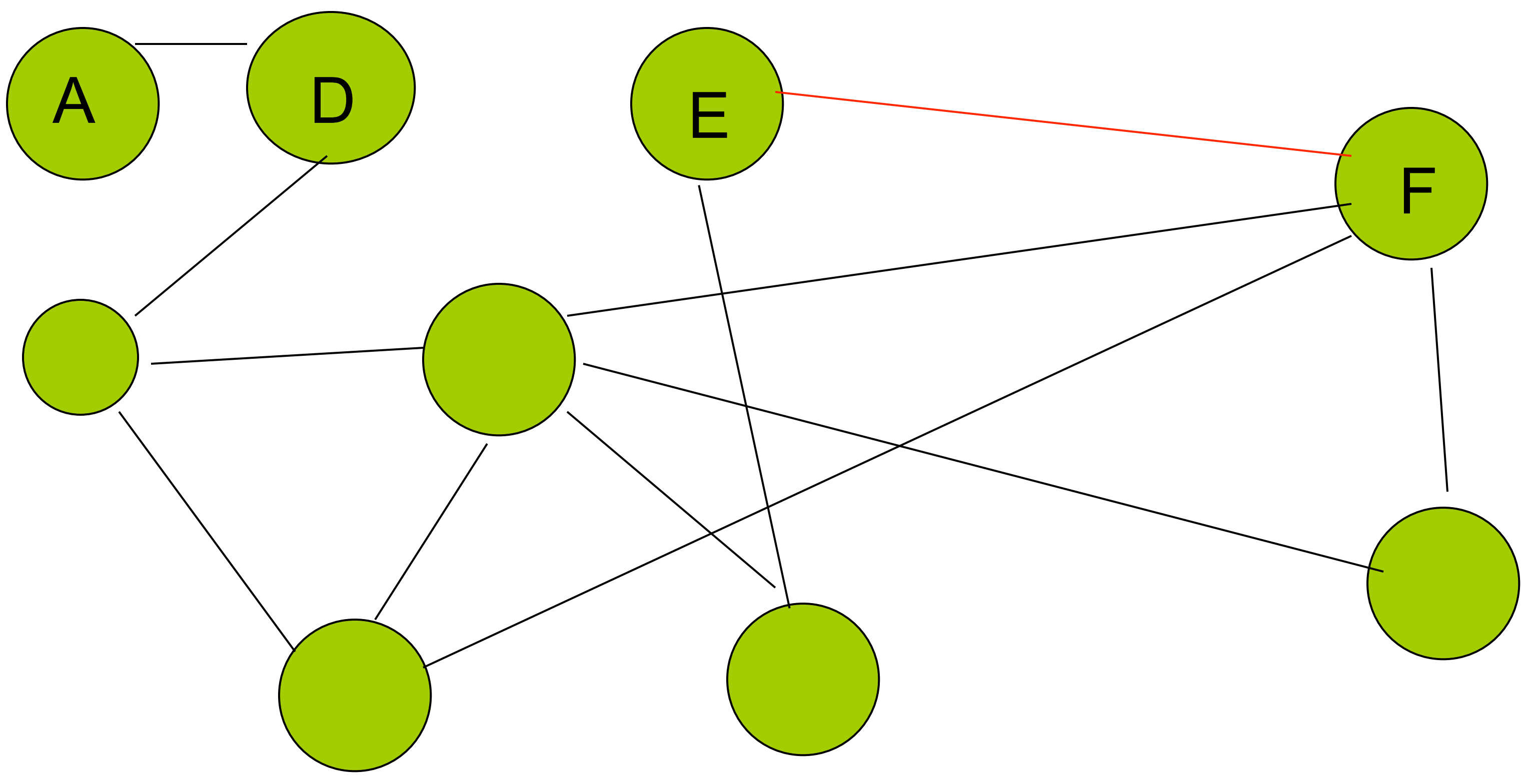
Sequence of online updates and queries



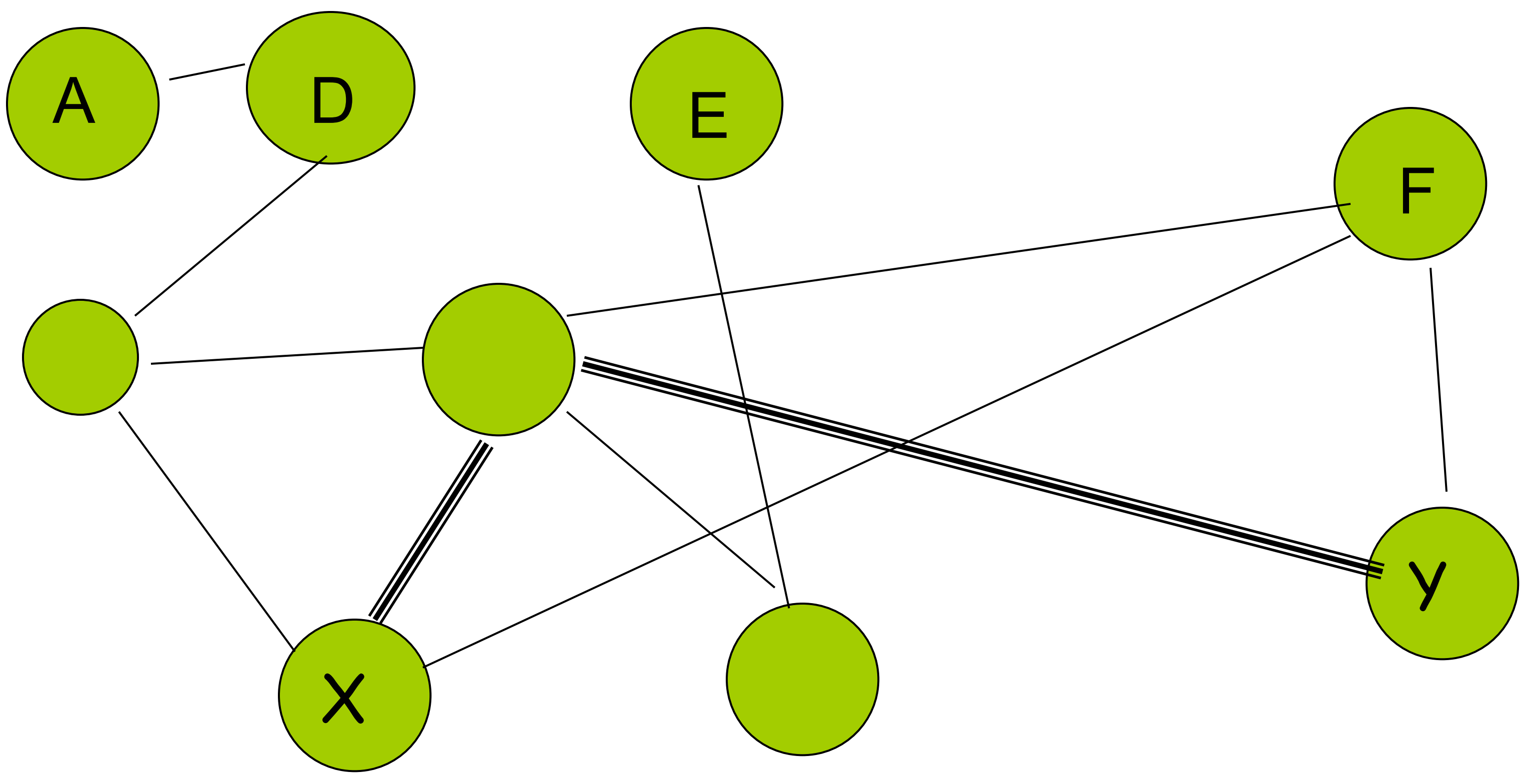
Update: Insert {A,D}



Update: Delete edge {E,F}



QUERY(X,Y): Is there a path between X and Y?



How to avoid O(m) cost of recomputing spanning forest with each update or running O(m) search for each query?

m=number of edges

A Simple problem , but lots of interesting ideas….

Early 60's-70's: partially dynamic **amortized**:

- insertions only:

  Union-find; Tarjan's $\alpha(m,n)$ analysis

- 1981: edge deletions only Even $O(mn)$

Fully Dynamic (Update times)

- 1983: $O(\sqrt{m})$ **worst** case Fredrickson
- 1992,7: $O(\sqrt{n})$ Sparsification Eppstein, Galil, Italiano, Nissenzweig

POLYLOG Amortized time updates

Update time / Query time

- 1995 $O(\log^3 n)$ / $O(\log n/\log\log n)$.
(expected time) Henzinger, King

- 1998 $O(\log^2 n)$ / $O(\log n/\log\log n)$
Holm, de Lichtenberg, Thorup

- 2000 $O(\log n\,(\log\log n)^3)$ / $O\left(\frac{\log n}{\log\log\log n}\right)$
Thorup

All with $\theta(n)$ worst case update time

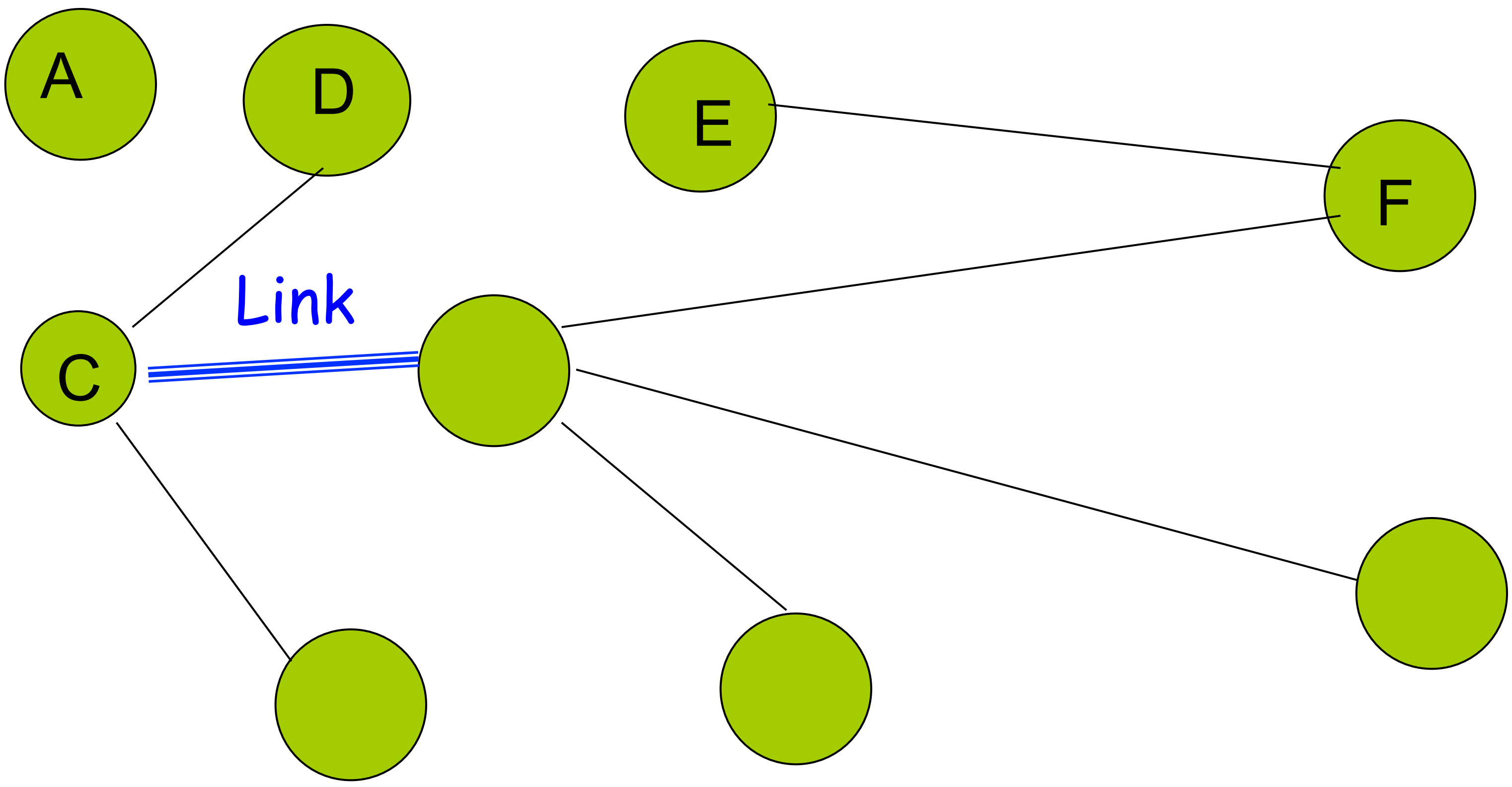
SODA 2013:

$O(\log^5 n)$ worst case update time
$O(\log n/\log\log n)$ query time
1-sided error:

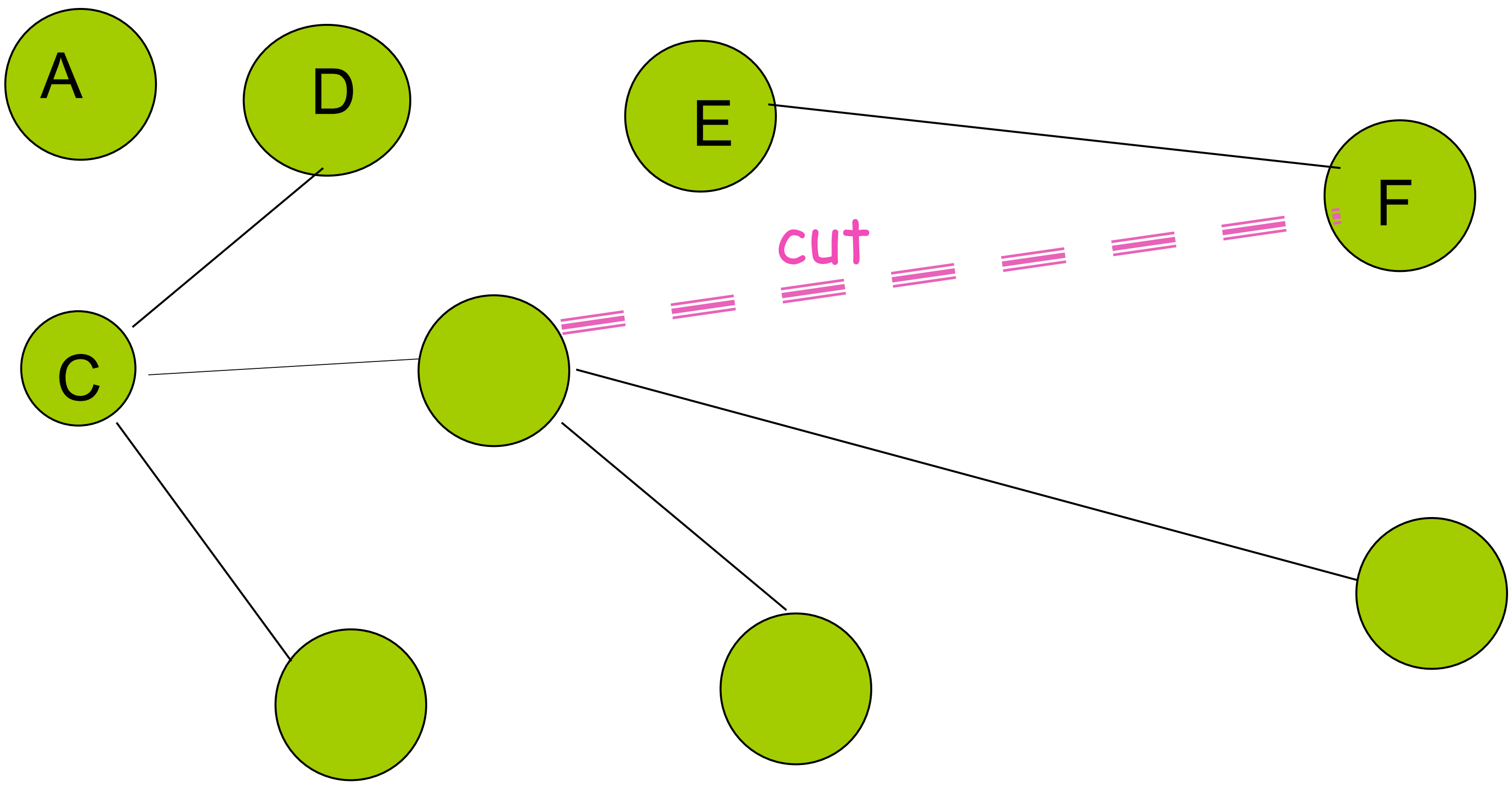
"Yes" always correct
"No" prob. $1/n^c$ error

All known techniques rely on maintaining a spanning forest

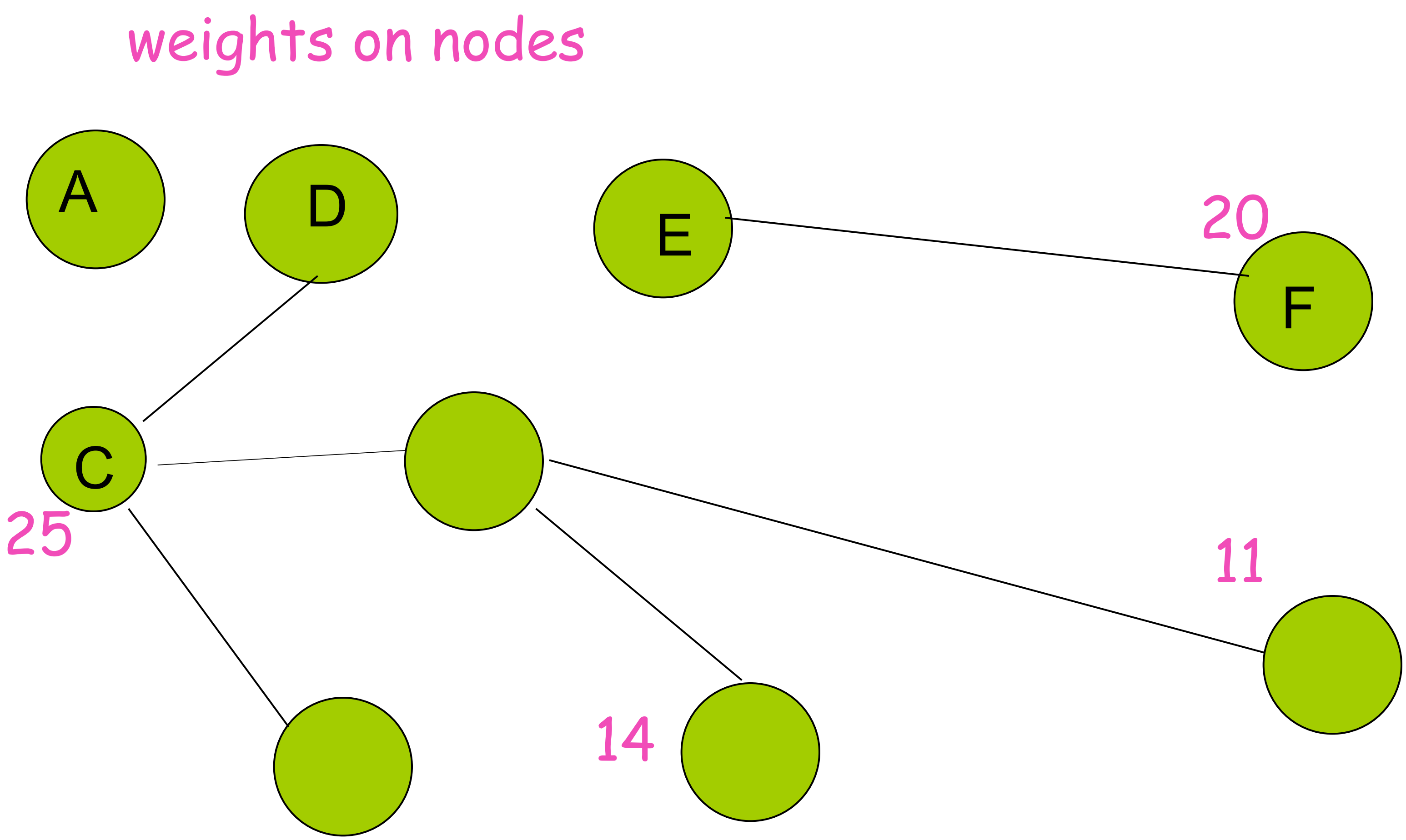
Dynamic Trees (ET-trees, H-K 1995)



Dynamic Trees (ET-trees, H-K 1995)

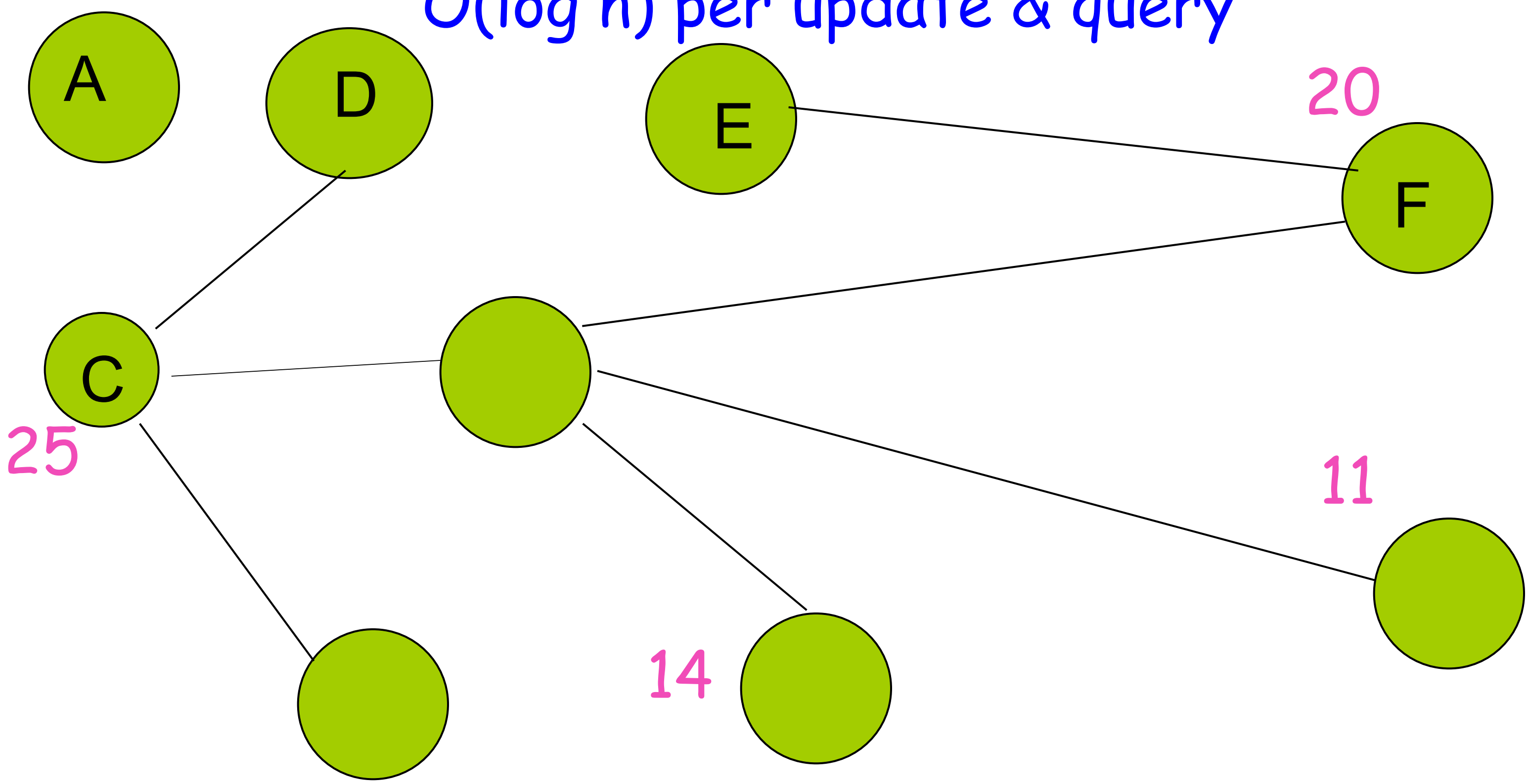


Dynamic Trees (ET-trees, H-K 1995)

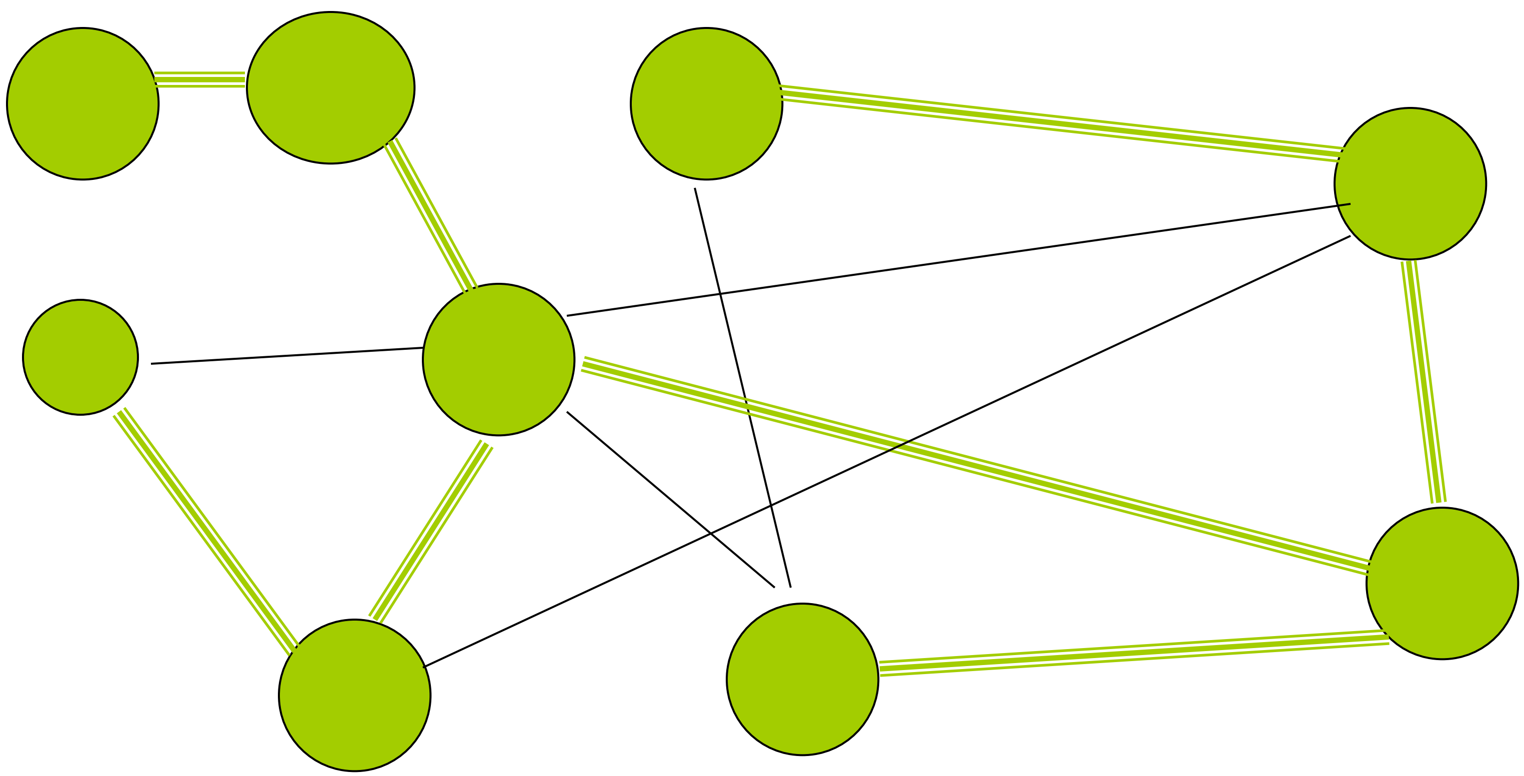


Dynamic Trees (ET-trees, H-K 1995)

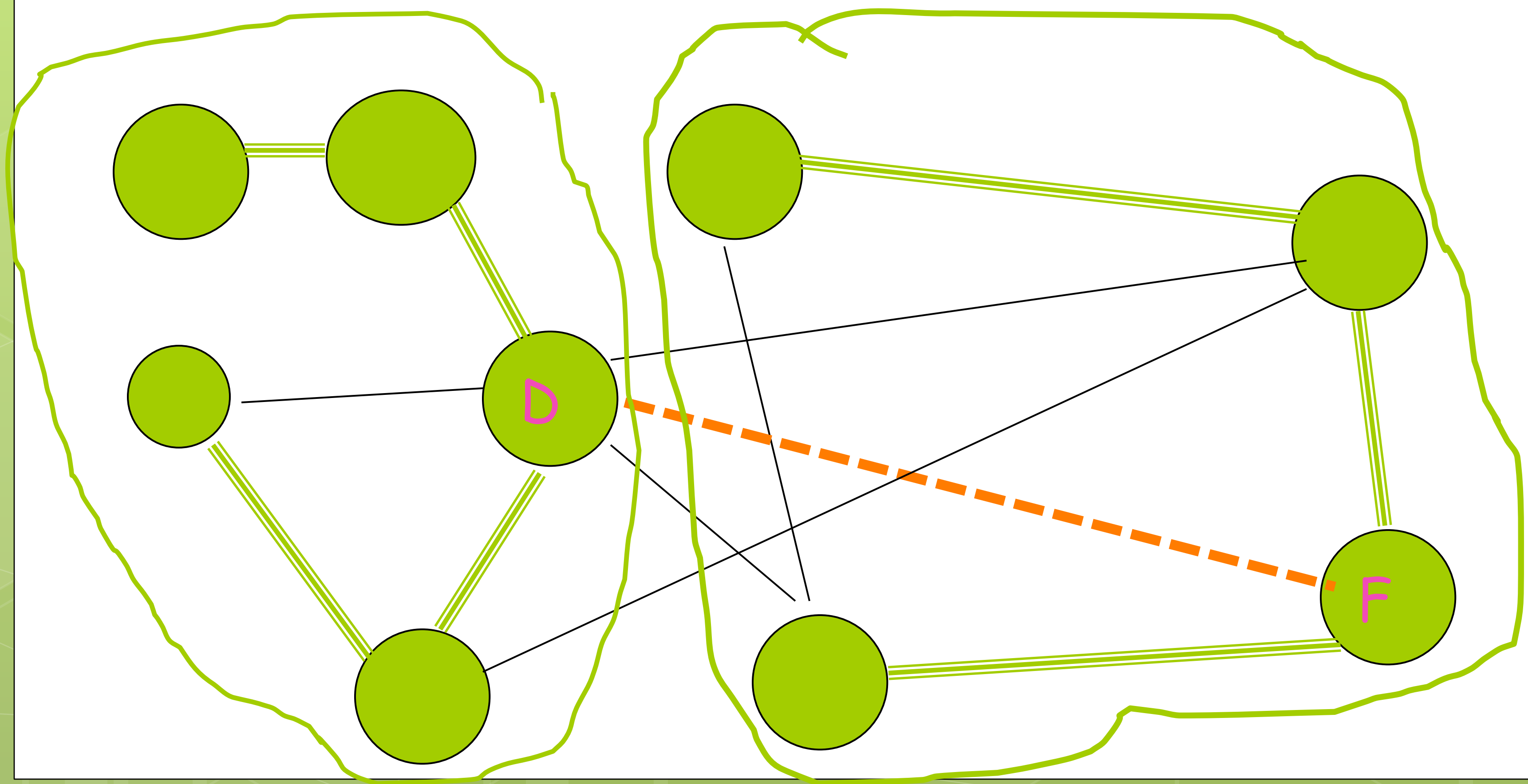
Query: Find tree containing node C
Query: Return sum of wts in tree
O(log n) per update & query



We maintain a spanning forest



When tree edge is deleted, how to find replacement edge?



Here, bitwiseXOR method:

V={1,2,…,n}

Form the name of {a,b}, a<b:

a (as a lg n bit number) followed by

b (as a lg n bit number)

“<ab>”

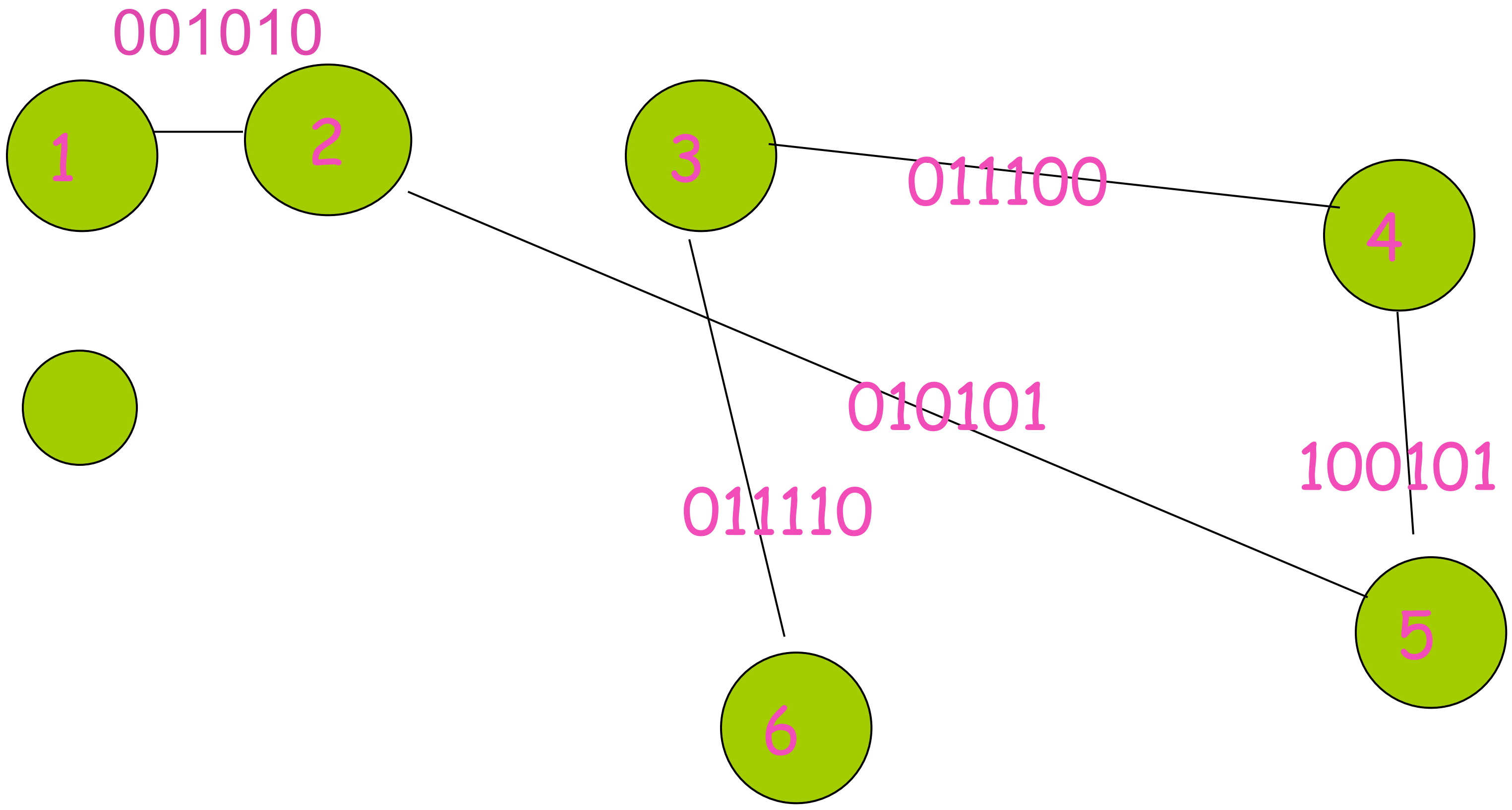
For each node a, keep a vector of bits v(a),

v(a)=bitwise XOR of names <ab> of edges

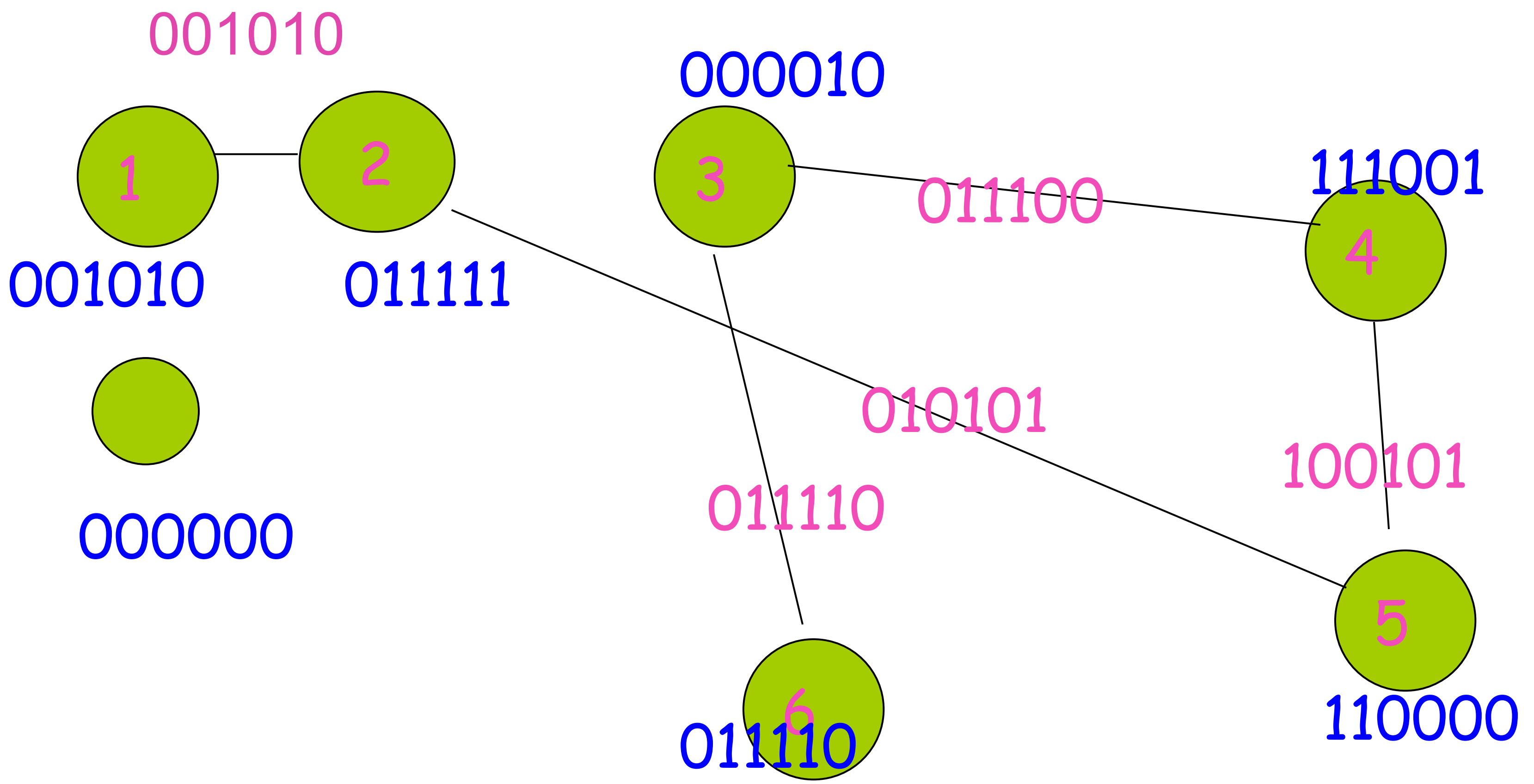
For any cut (S, V\S), if there is exactly one edge {x,y} in its cutset then

$$\mathrm{XOR}_{a\ \mathrm{in}\ S}\ v(a) = \langle xy \rangle$$

Example:

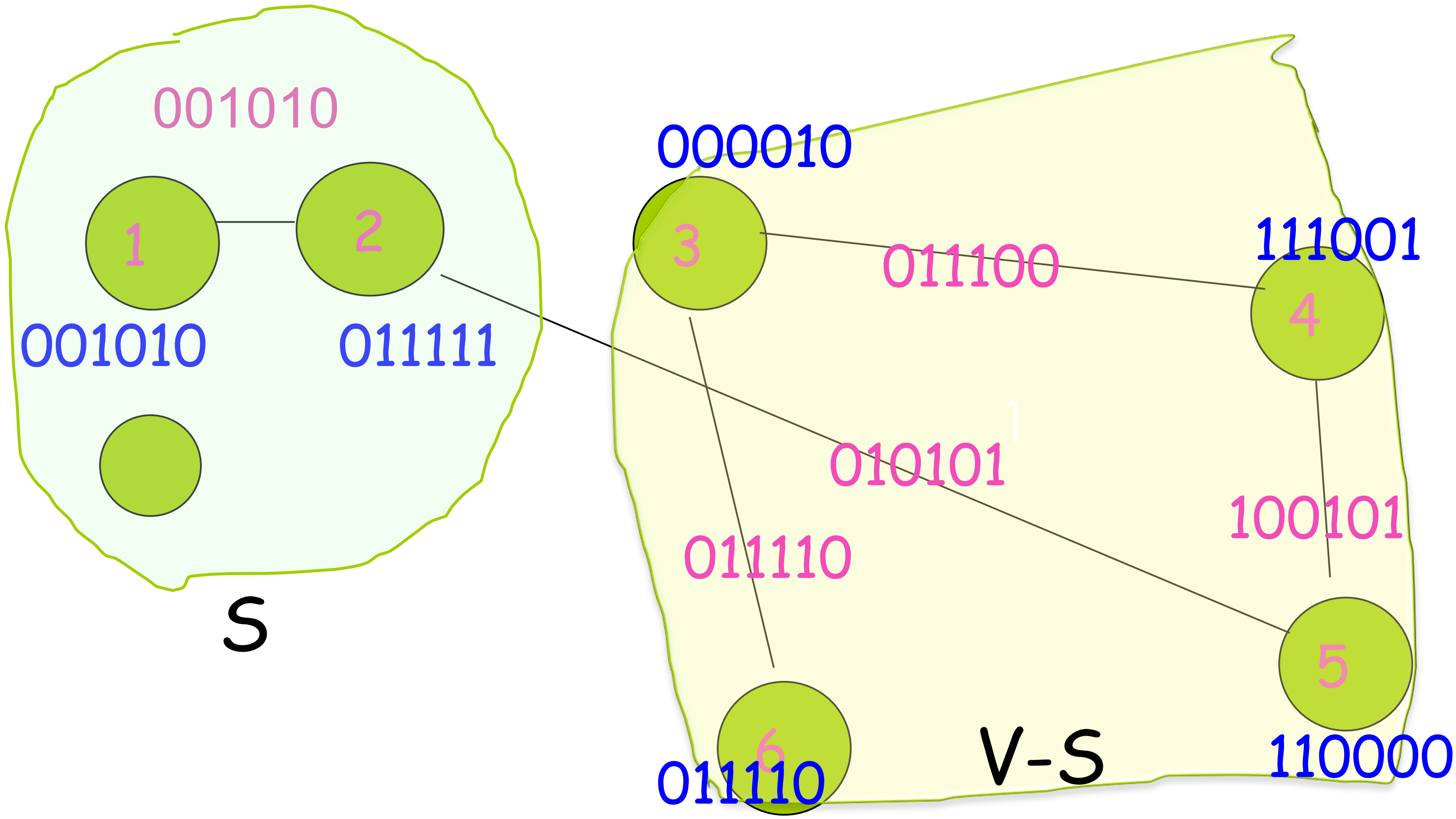


v(a)



001010
000010
1
2
3
111001
011100
4
001010
011111
010101
100101
011110
000000
5
6
011110
110000

XOR of v(a) = 001010 = XOR of v(a) in V-S
in S + 011111
=010101



Dealing with larger cutsets

To insert:

- Add <ab> to v(a,i) and v(b,i) with prob. $1/2^i$, for i=0.,2,…,2lg n
- Keep record of additions for each a and i.

To delete: Add again if it was added before

Dealing with larger cutsets

To insert:

- Add <ab> to v(a,i) and v(b,i) with prob. $1/2^i$, for i=0.,2,…,2lg n
- Keep record of additions for each a and i.

To delete: Add again if it was added before

Observe: C cutset of (S,V-S). For i ~lg |C|,

Pr[Adding an edge {a,b} in C to v(a,i)]~=1/|C|

and

Pr[Exactly one edge in C was added to some v(a,i)

=Pr[$\text{bitwiseXOR}_{a \text{ in } S}$ v(a,i) = name of edge in C]

= a const.

Dealing with larger cutsets

To insert:

- Add <ab> to v(a,i) and v(b,i) with prob. $1/2^i$, for i=0.,2,…,2lg n
- Keep **record** of additions for each a and i.

To delete: Add again if it was added before

Observe:

C cutset of (S,V-S). For i ~lg |*C*|,

Pr[bitwiseXOR$_{a \text{ in } S}$ v(a,i) = edge in *C*] = a const.

Repeat for log n versions. Then for some version, the name of exactly one edge in *C* appears with prob $1-1/n^c$

Over a sequence of updates:

Union bound gives small error over polynomial length sequence, *provided the choice of updates are independent of the random bits*

Record enables incremental rebuilding and periodic correction of data structure to maintain prob. of error.

Solution to dynamic connectivity?? (not quite)

Problems:

A. Can't let adversary know the spanning tree edges

B. Adversary sees answers to queries
--Update sequence is independent of random bits while all queries correctly answered, as they are then determined by the graph itself.

C. Choice of *cut searched* depends on random bits!

XOR method solves easier problem:

"CUTSET" DataStructure (DS)

Maintain a forest F of dynamic disjoint trees in graph G:

Updates: insert-edge, delete-edge,
insert-tree- edge, delete-tree-edge.

Query (S) returns an edge in the cutset (S, V\S)

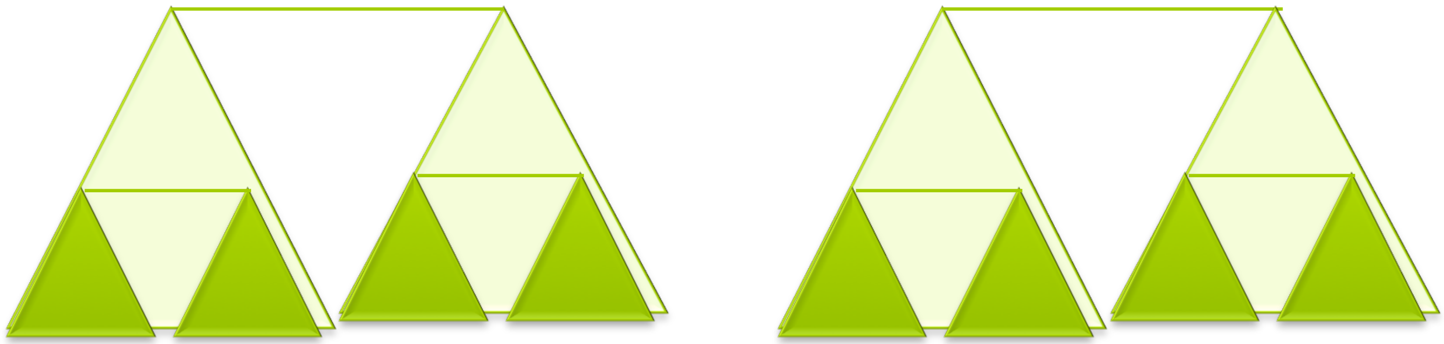
Updates are independent of random bits.

Maintain spanning forest using Cutset $DS_{i,}$ i=0…lg n =TOP

Random bits from Cutset DS_i used to pick edges in F_{i+1} joining trees from F_i

"Tier i+1 edge"

Query(T,k) returns a k+1 edge if it exists



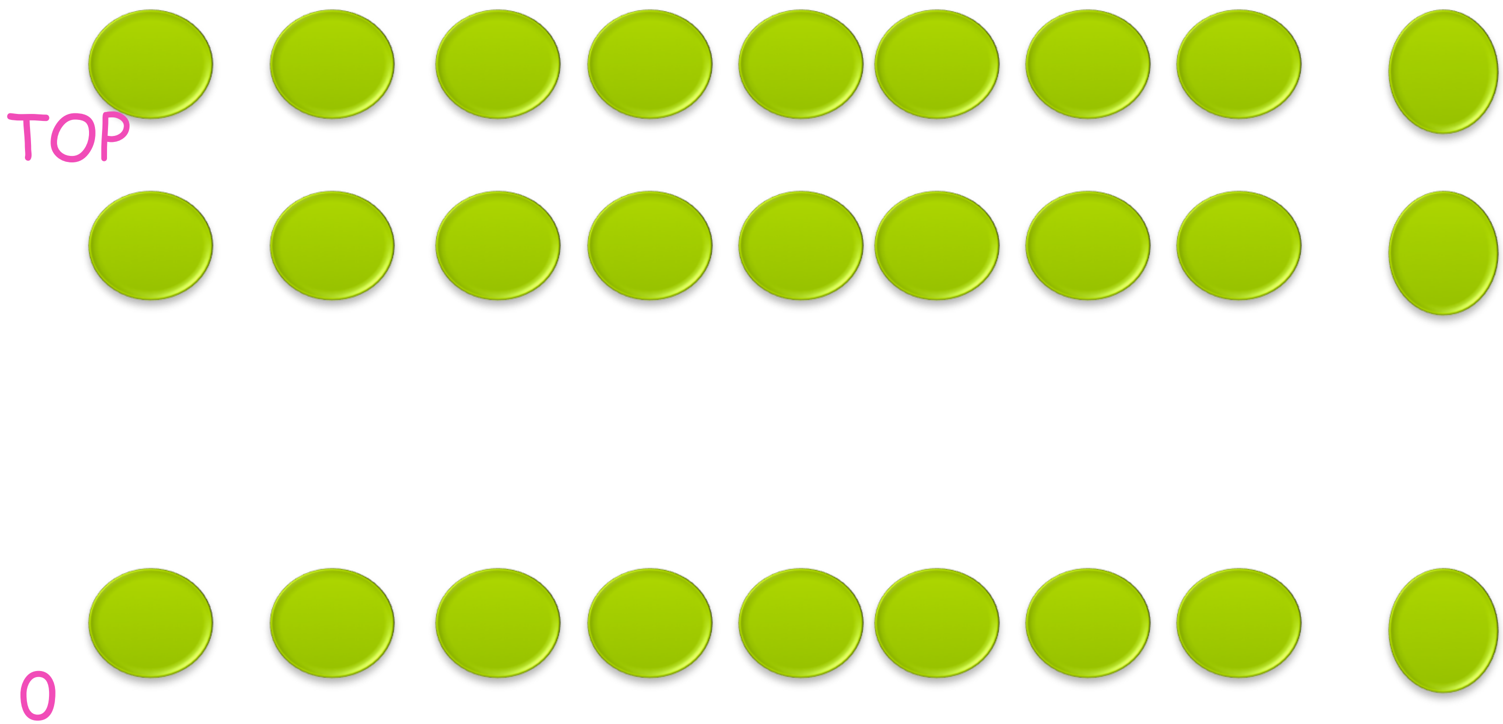
INVARIANTS:

-Structure of F_i is independent of random bits from tiers i and higher.

-Every tree on tier i is matched (linked) to another tree on tier i by a tier i+1 edge unless it's maximal in G

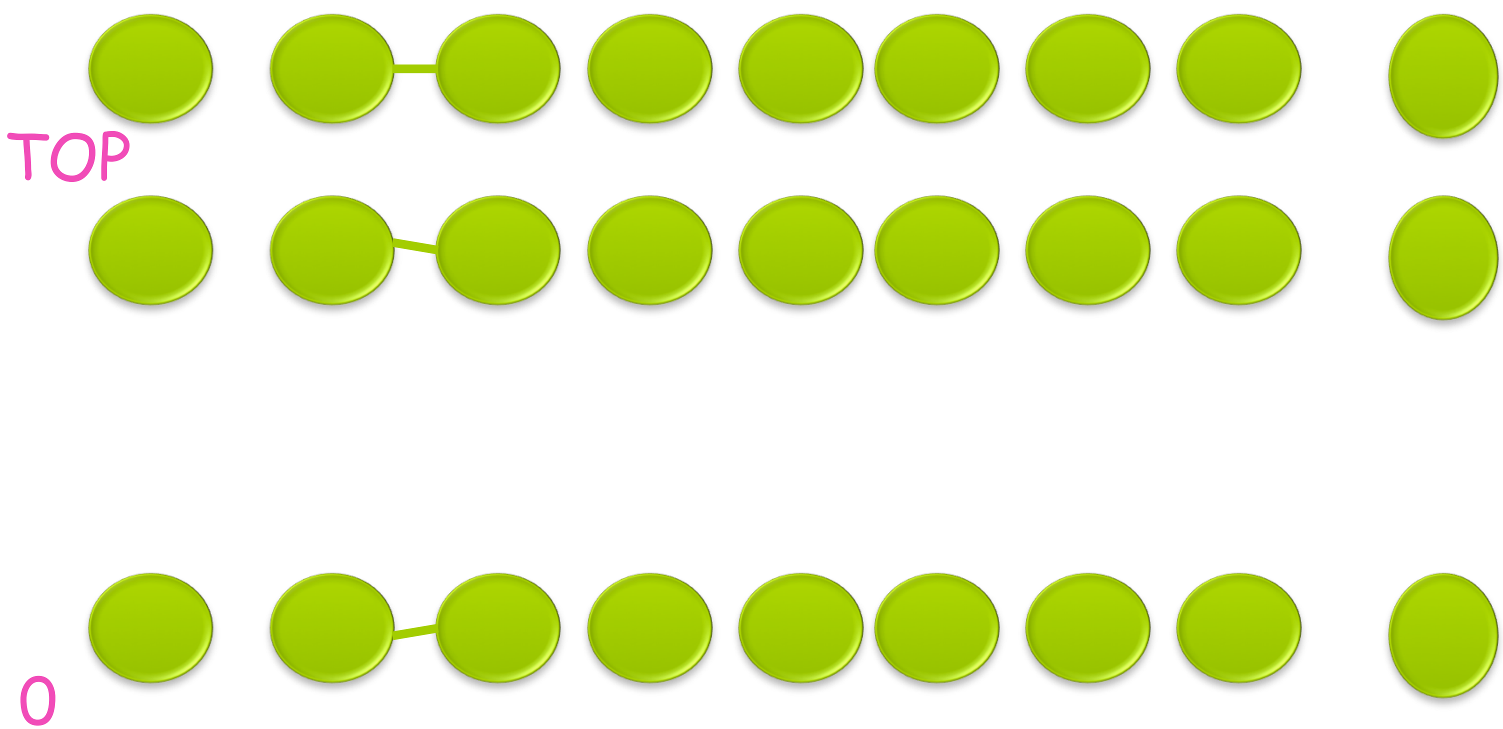
→ spanning forest by TOP tier

Initially, all F_i are singleton nodes



Insert edge: insert into all Cutset DS_i

If edge joins unconnected trees in F_{top} insert edge as tree edge into all F_i

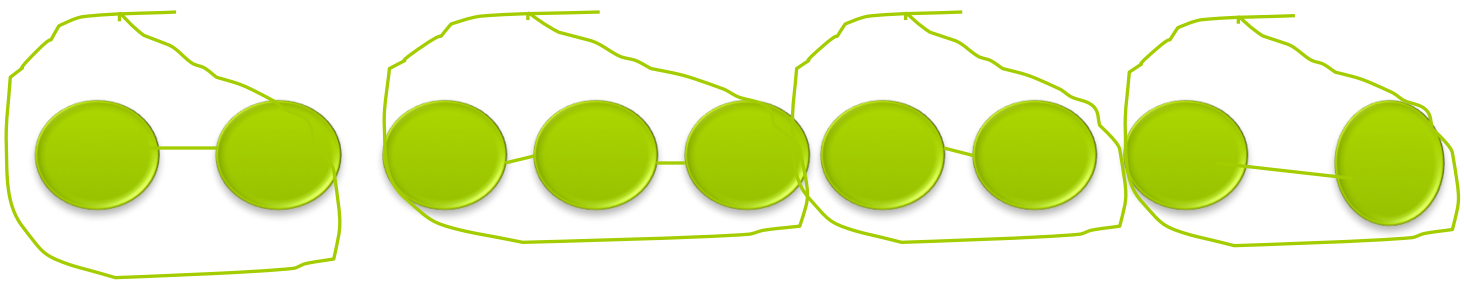


Delete edge: delete from all Cutset DS_i

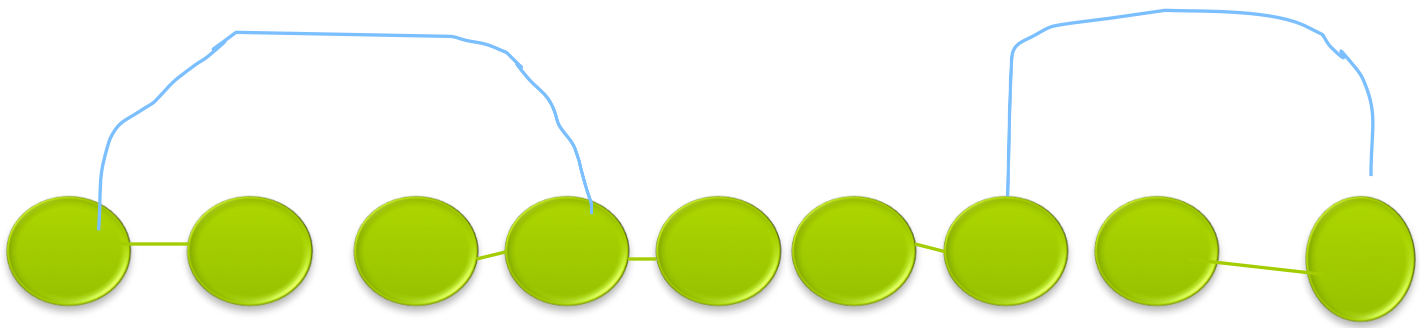
Restore Invariants using Cutset DS_i

Example: F_0

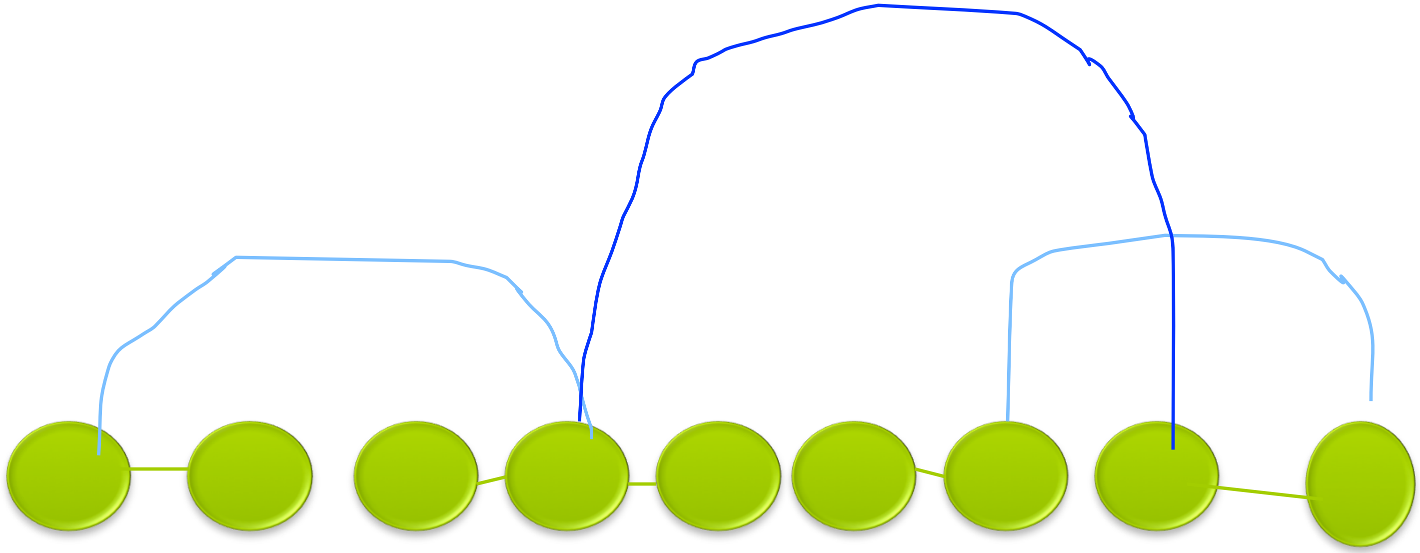
Example: F_1



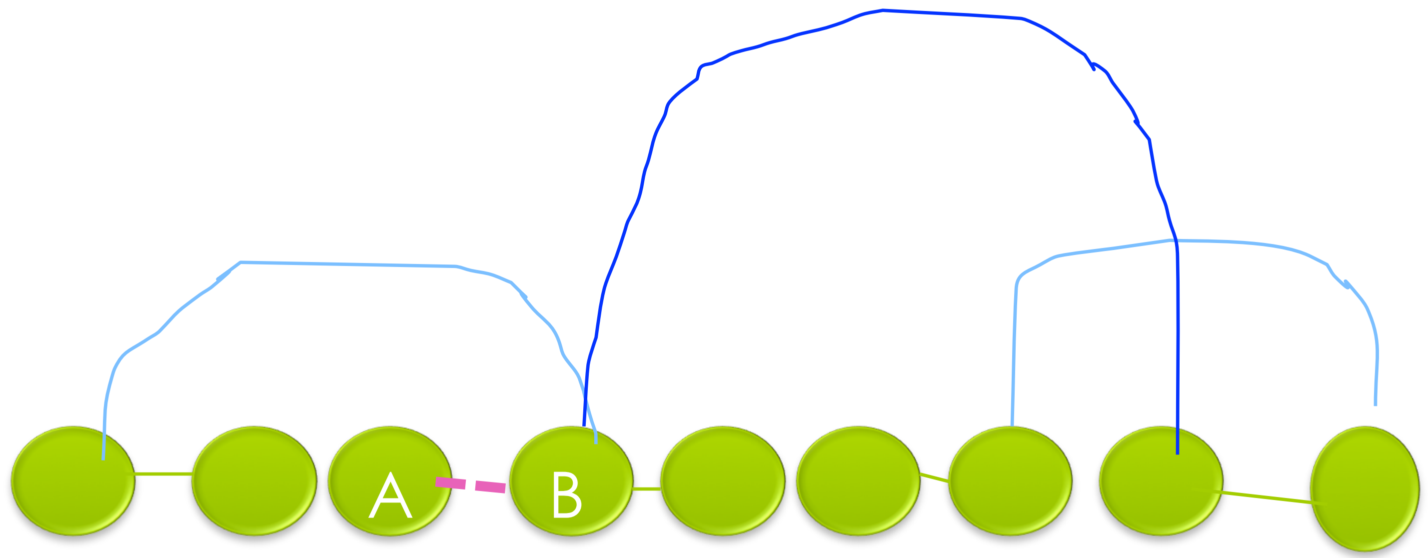
F_2



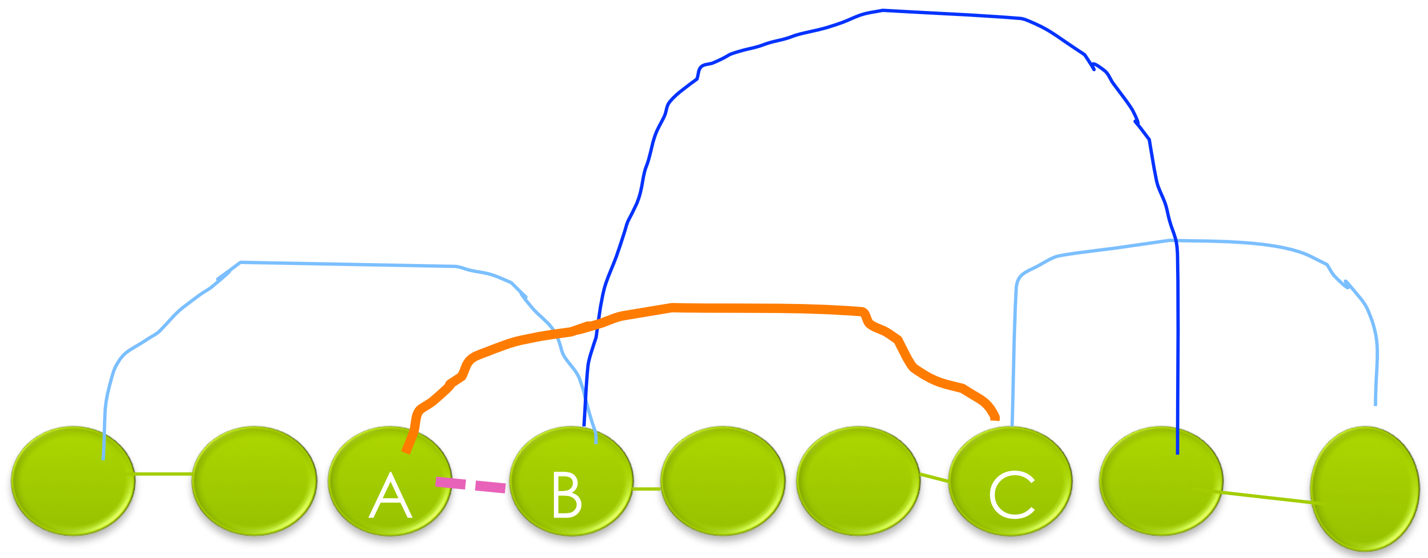
F_3



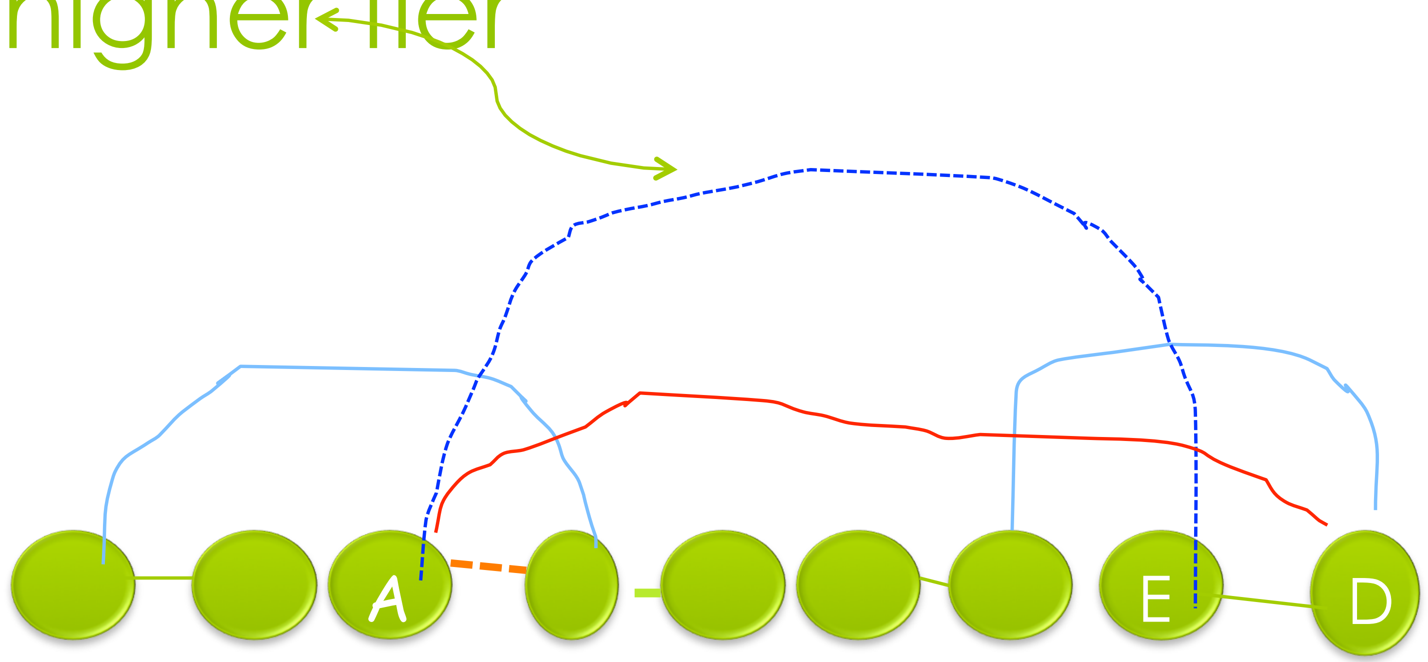
Deletion of a tier 1 edge:



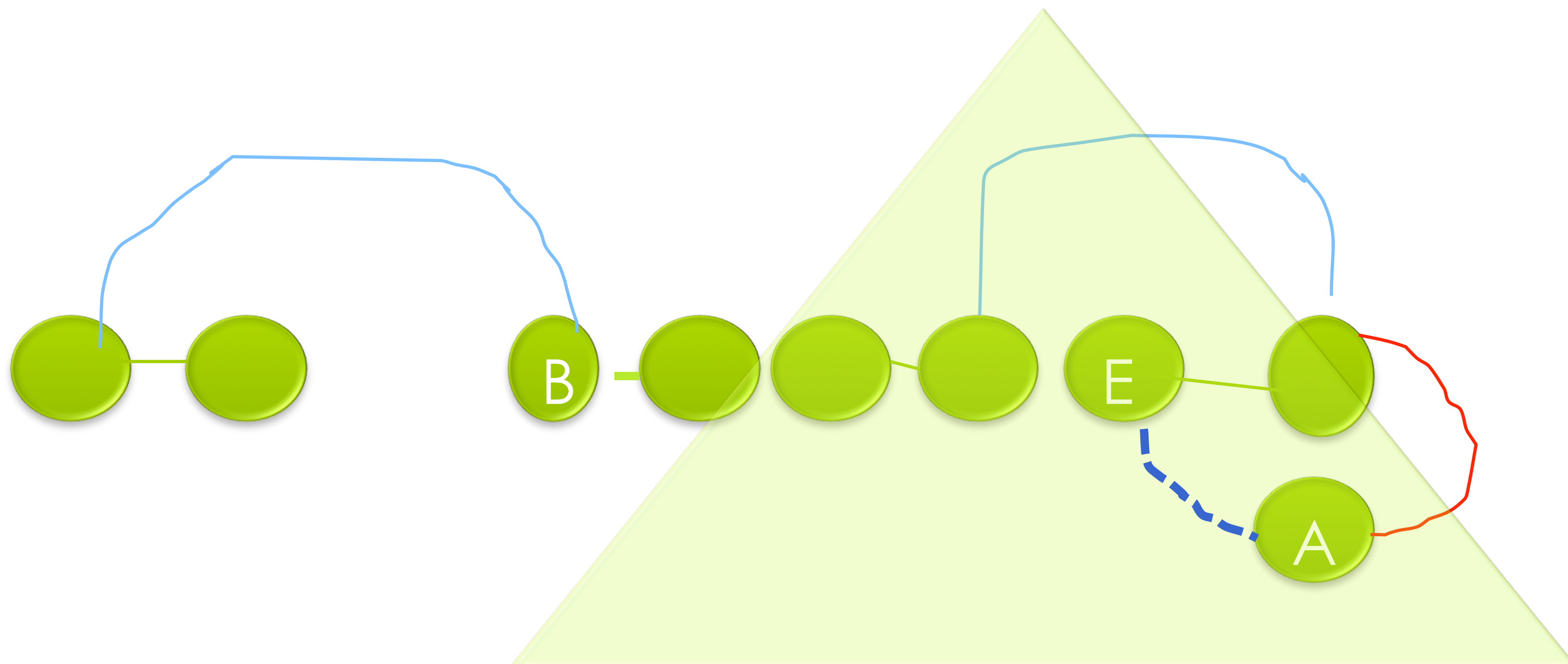
Deletion: If unmatched tree T in tier i, find new edge in Cut (T,V-T) and insert into all $F_{i'}$, i'>i



But new tree edge may cause an unmatched tree on a higher tier



Unmatched tree in F_2



Delete (x,y)

Delete(x, y)

remove {x,y} from all $CutSet_i$ containing it.

for u in {x,y} do
    while u has an unmatched ancestor in the
            Boruvka tree **do**
        A ← the lowest unmatched ancestor of u
        k ← (tier of A)
        Reconnect(A, k)

Reconnect(A, k)

e = {v,w} ← Query(A,k) (assume that v is the endpoint of e in A)

if e = null then mark A as maximal

else {remove higher edge from F to break cycle}

if there is a path from v to w in F_{top} **then do**

e' ← maximum tier edge on the path between v and w.

Remove e' from all F_i that contain it

Add e to $F_{k'}$ for all k' > k

To implement:

“**if** there is a path from v to w in F_{top} **then do**

e’ ← maximum tier edge on the path between v and w.”

Use S-T dynamic trees:

Maintain F_{TOP} with edges labeled by their tier number.

Find maximum weighted edge in path from v to w, O(log n) per operation.

Other Implementation details:

Use ET-Trees to maintain XOR sums:

- $O(\log^2 n)$ size vectors, → $O(\log^3 n)$ cost to change a tree edge
- 2 tree edges per tier inserted per deletion
- Each edge insertion affects forests in up to lg n tiers
- → $O((\log^3 n)(2 \log n)(\log n))$

--> $O(\log^5 n)$ overall cost per deletion

Space

Record of insertions requires $\tilde{O}(m)$. Omit by using hash function for randomness, but then can only be run for poly time.

See Graph Sketches paper, Ahn, Guha, McGregor, SODA 2012, which uses similar ideas to ours, but for a somewhat different problem.

Open Problems

Reduce update cost: lots of possibilities, or modify goal to reduced worst case **expected** cost.

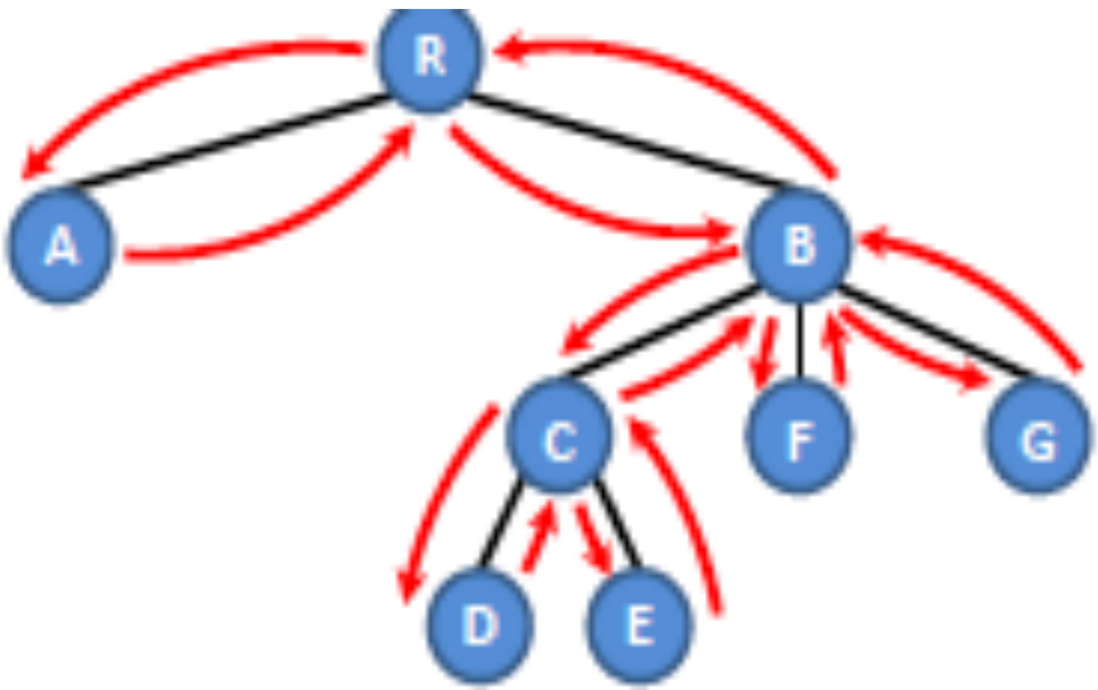
Is there a Las Vegas or deterministic alg with polylog worst case time?

Is there a polylog worst case alg. for dynamic MST?

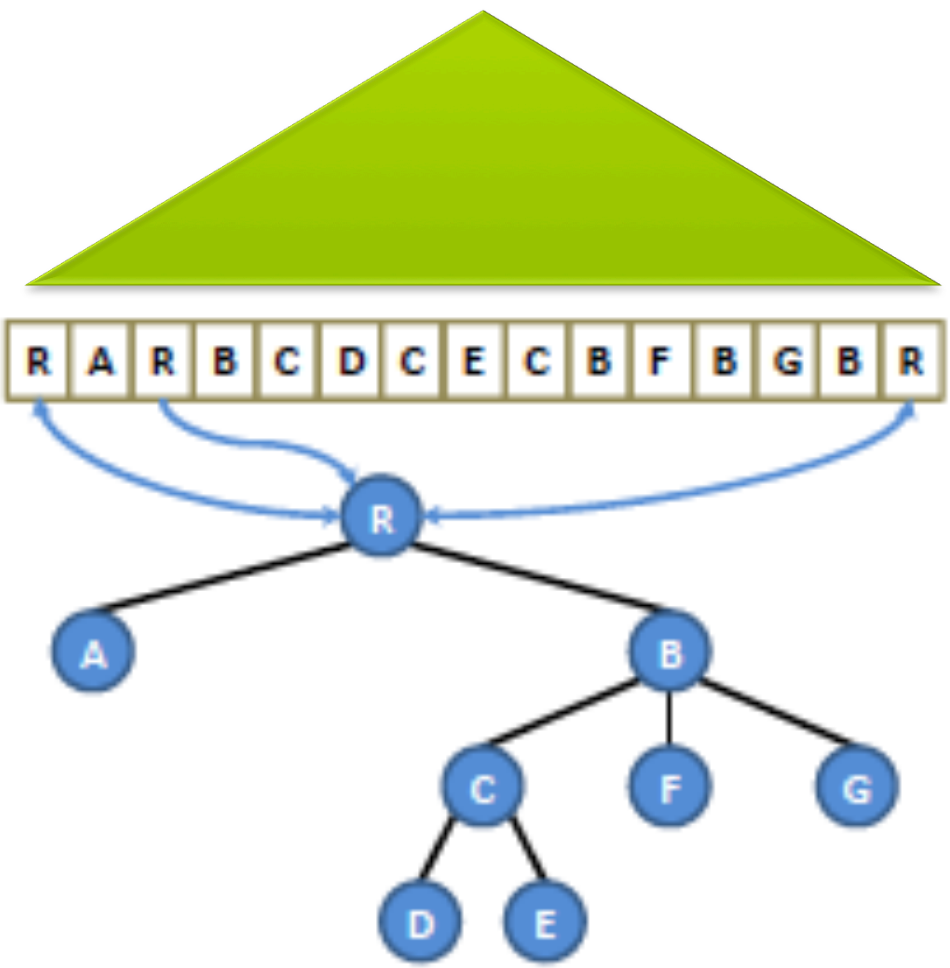

Come visit us
in Victoria
Questions?
Photograph © The Butchart Gardens L

1995,98
ET trees
used

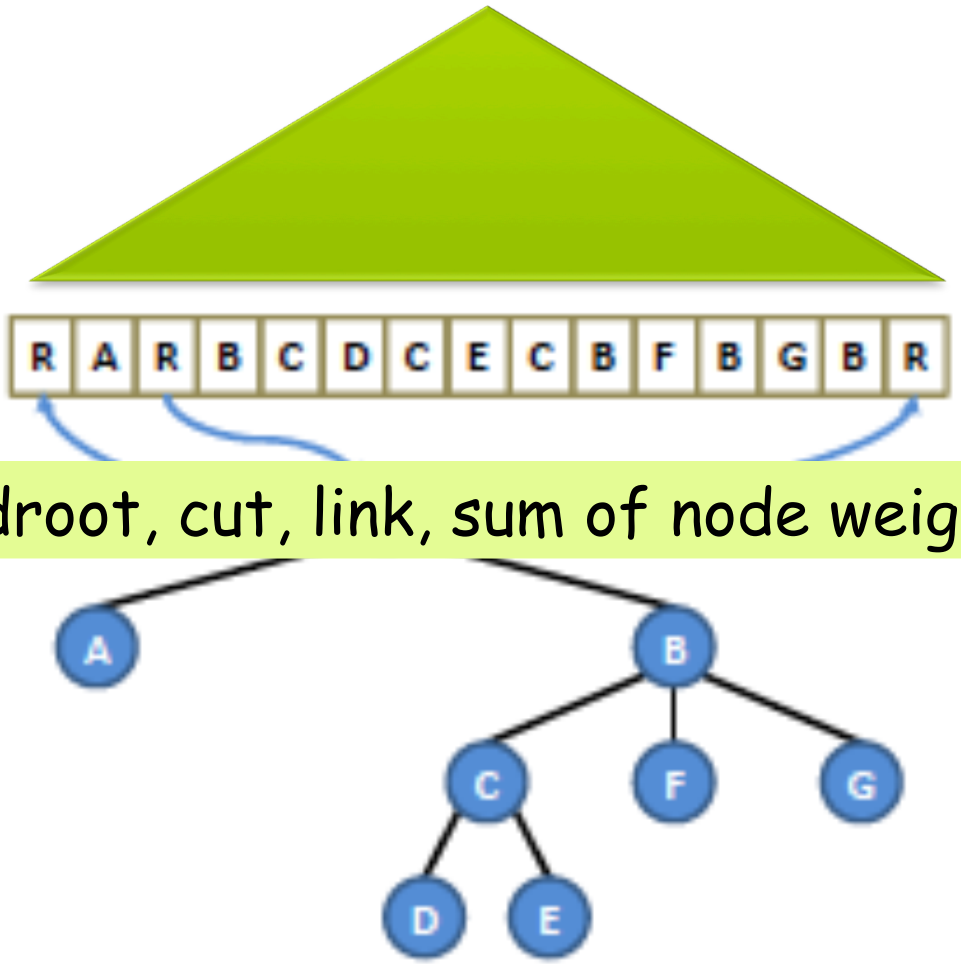
Euler Tour Tree
(from Erik Demaine.'s class notes)



Euler Tour Tree



Euler Tour Tree: augmented balanced search tree



findroot, cut, link, sum of node weights in tree

Lower Bounds for Dynamic Connectivity

$\Omega(\log n)$ time per operation (Patrascu, Demaine 2004)
in the
Cell probe model=#memory accesses
(where each word contains log n bits)

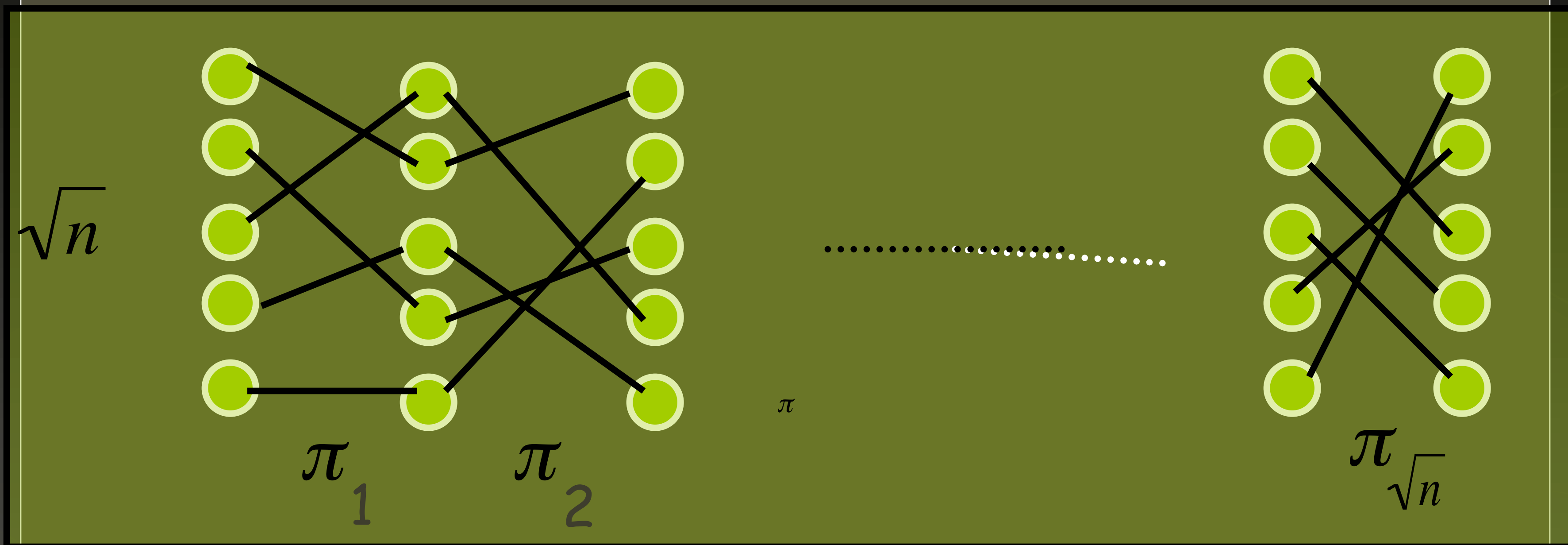
Also lower bounds on tradeoffs between query time and update time, e.g.:
query time * lg(update time/query time)= $\Omega(\log n)$

I would like to take a moment to remember Mihai Patrascu a very talented young colleague in this area whom I will miss

July 17,1982-

June 5, 2012

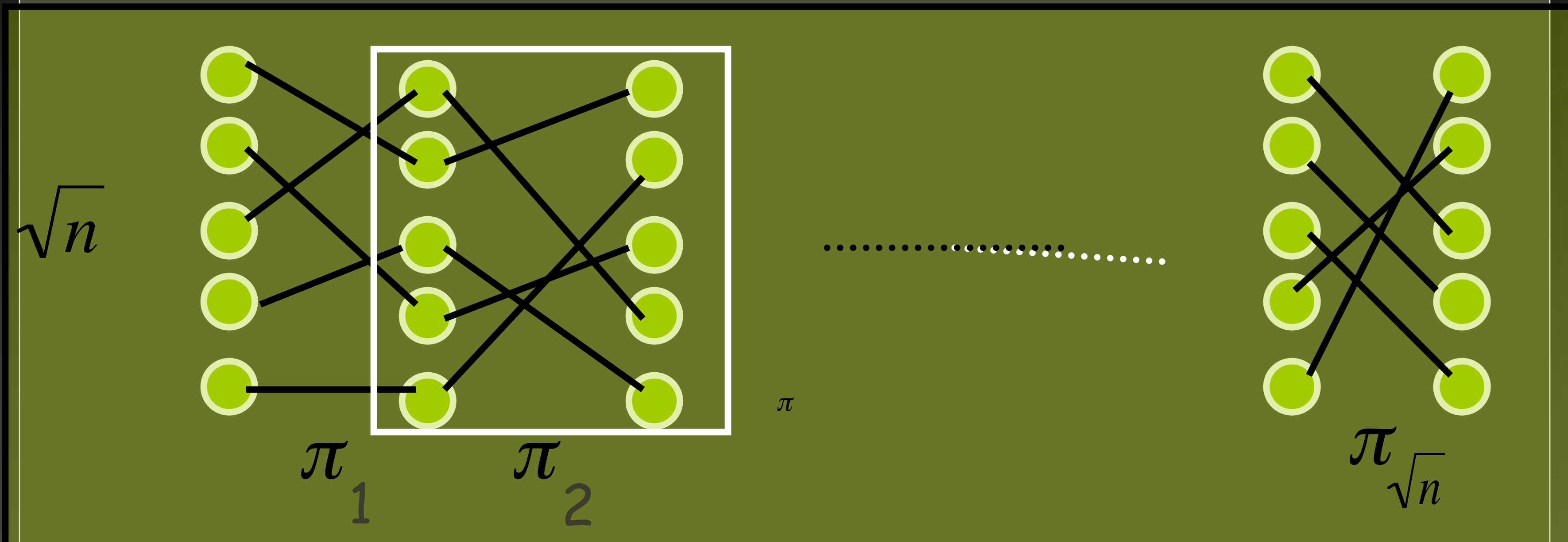
Lower Bound for Connectivity



Random distribution of BATCH updates and queries:

Prob. 1/2: replace a randomly chosen Π_k by a random Π

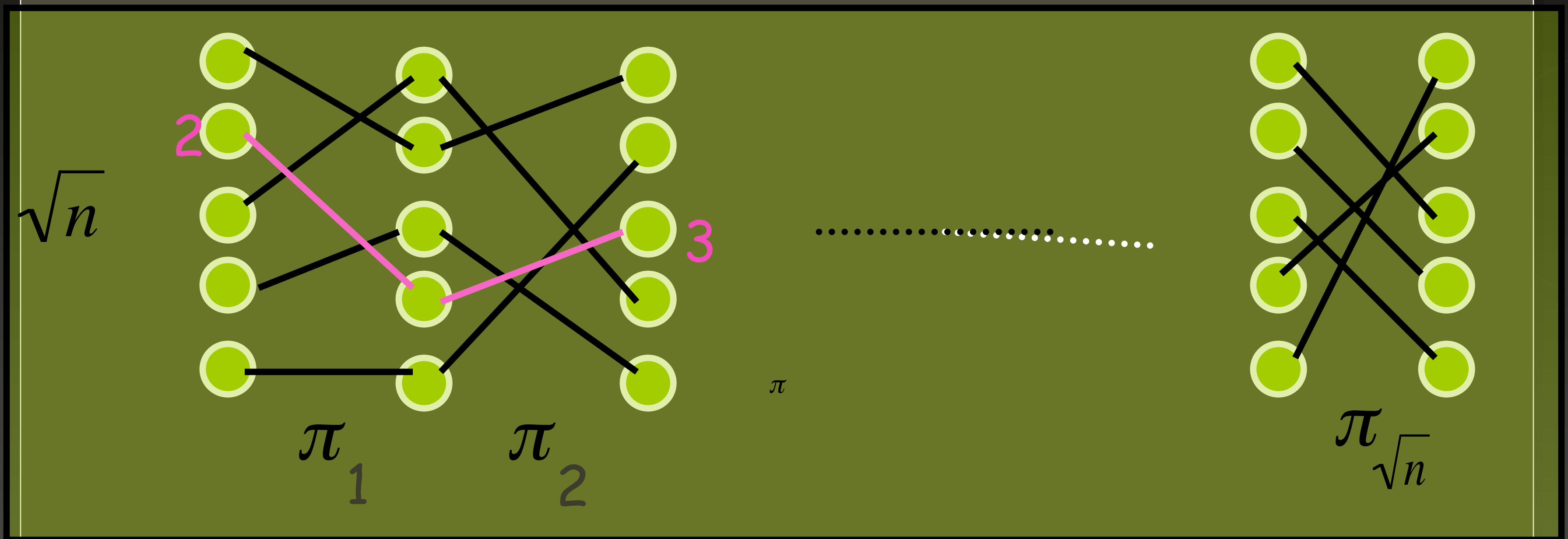
Lower Bound for Connectivity



Random distribution of BATCH updates and queries:

Prob. 1/2 do update (k) : replace a randomly chosen π_k by a random π

Lower Bound for Connectivity



Random distribution of BATCH updates and queries:

Prob. 1/2: update (k): replace a randomly chosen π_k by a random π

Prob. 1/2: query (k): $\forall$ rows i, random column k, test $\pi_k(\ldots(\pi_2(\pi_1(i))))$

Lower Bound for Connectivity

Sequence of batch operations

Split into two time intervals

i .. j-1

Updates here
sorted by type

$U_1 < U_2 < \ldots < U_{k-1}$

j .. k
Queries here
sorted by type:

$Q_1 < Q_2 < \ldots < Q_k$

Note: High expected number L of interleaves:

$U_1 < Q_{1,} < U_2 < U_3 < Q_2 < \ldots < U_{k-1}$

To answer Q_2 need to know U_2 ,U_3
-->Need to know a different U for each interleaving

Lower Bound for Connectivity

Sequence of batch operations

Split into two time intervals

i .. j-1
Updates here
sorted by type

$U_1 < U_2 <\ldots< U_{k-1}$

WRITES ⟷ READS

j .. k
Queries here
sorted by type:
$Q_1 < Q_2 <\ldots< Q_k$

Number of READS of these WRITES must be sufficient to provide enough bits to encode L U's.

Paper shows method for concise encoding of info from READS from which U's can be reconstructed.

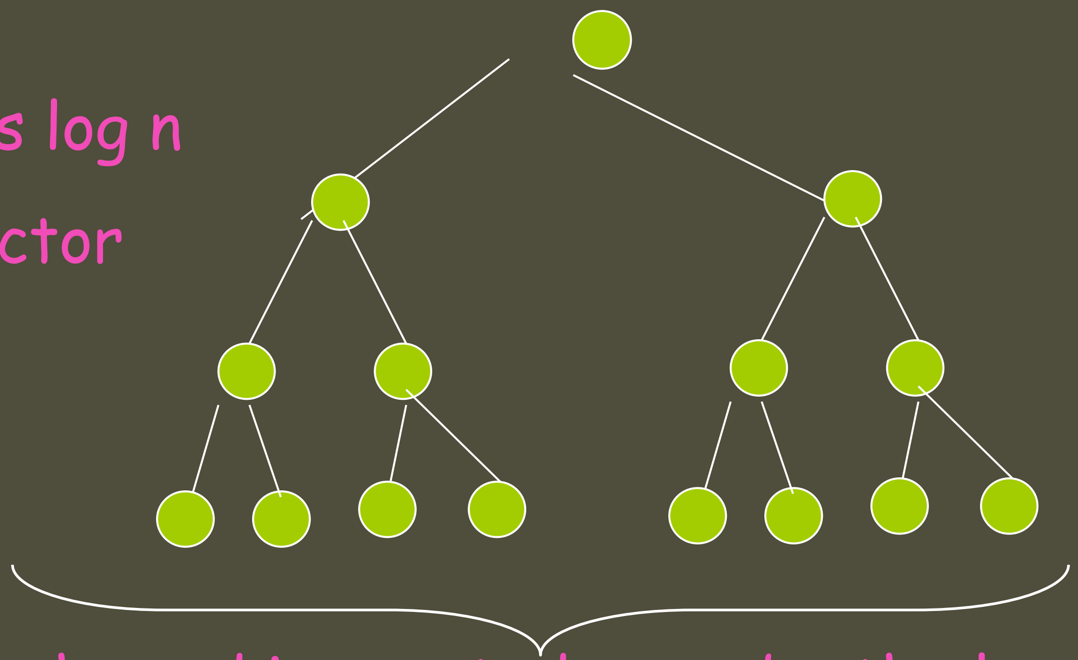
Lower Bound for Connectivity

Sum up expected costs over intervals given by binary tree,

Parent interval = union of children intervals.

Adds log n

factor



Note: Each read is counted once, by the lowest common Ancestor of the read and most recent preceding write time.