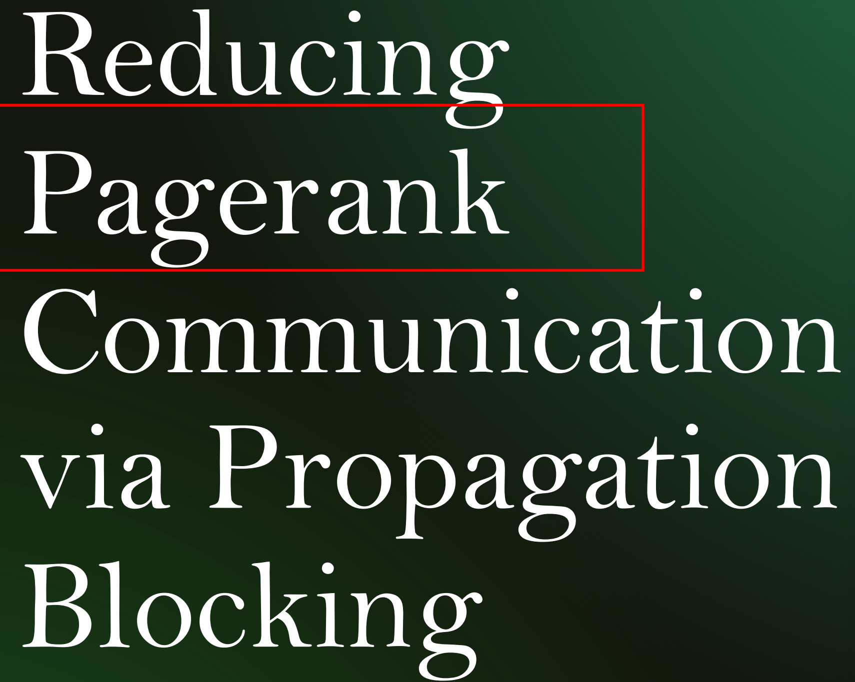


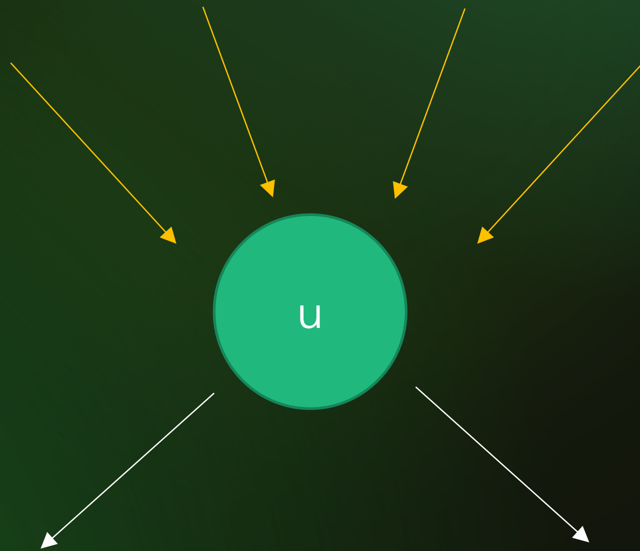
Reducing Pagerank Communication via Propagation Blocking

BEAMER, ASANOVIC & PATTERSON

By: Isabelle Quaye

Pagerank

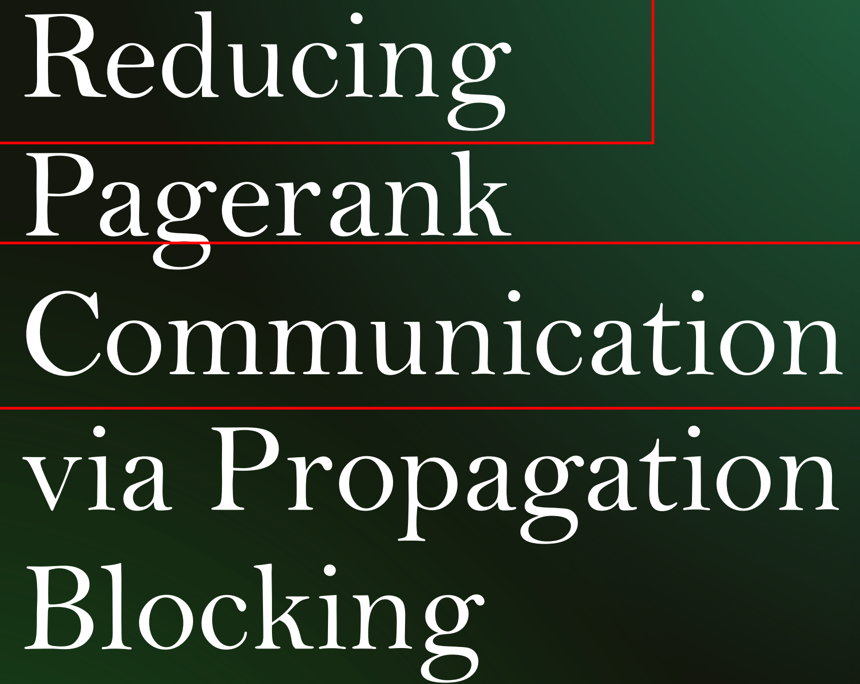
- Algorithm for ranking vertices by importance/popularity
- Model the web such that webpages are vertices and links connecting webpages are edges
- The algorithm progresses in rounds/iterations



Dampening factor: introduces randomness

$$Importance(u)_t = \frac{1-d}{|V|} + d * \sum_{v \in N^-(u)} \frac{Importance(v)_{t-1}}{|N^+(v)|}$$

$N^+(v)$: outgoing neighbours
$N^-(v)$: incoming neighbours



Reducing Pagerank Communication via Propagation Blocking

BEAMER, ASANOVIC & PATTERSON

Isabelle Quaye

What does “communication” mean here?

- Communication here refers to the movement of data between the cache and memory
- When processing large graphs, input, output and intermediate values may not all fit into cache
- So we may incur cache misses as we read and write data during execution
- Poor locality in reading/writing data = Lots of cache misses = High communication costs
- Reducing Pagerank communication = improving locality when executing Pagerank algorithm on large graphs

Reducing Pagerank Communication via Propagation Blocking

BEAMER, ASANOVIC & PATTERSON

Isabelle Quaye

What existed before propagation blocking?

- Reordering graphs by relabelling
- Processing vertices in certain orders
- Graph compression
- Cache Blocking/Tiling

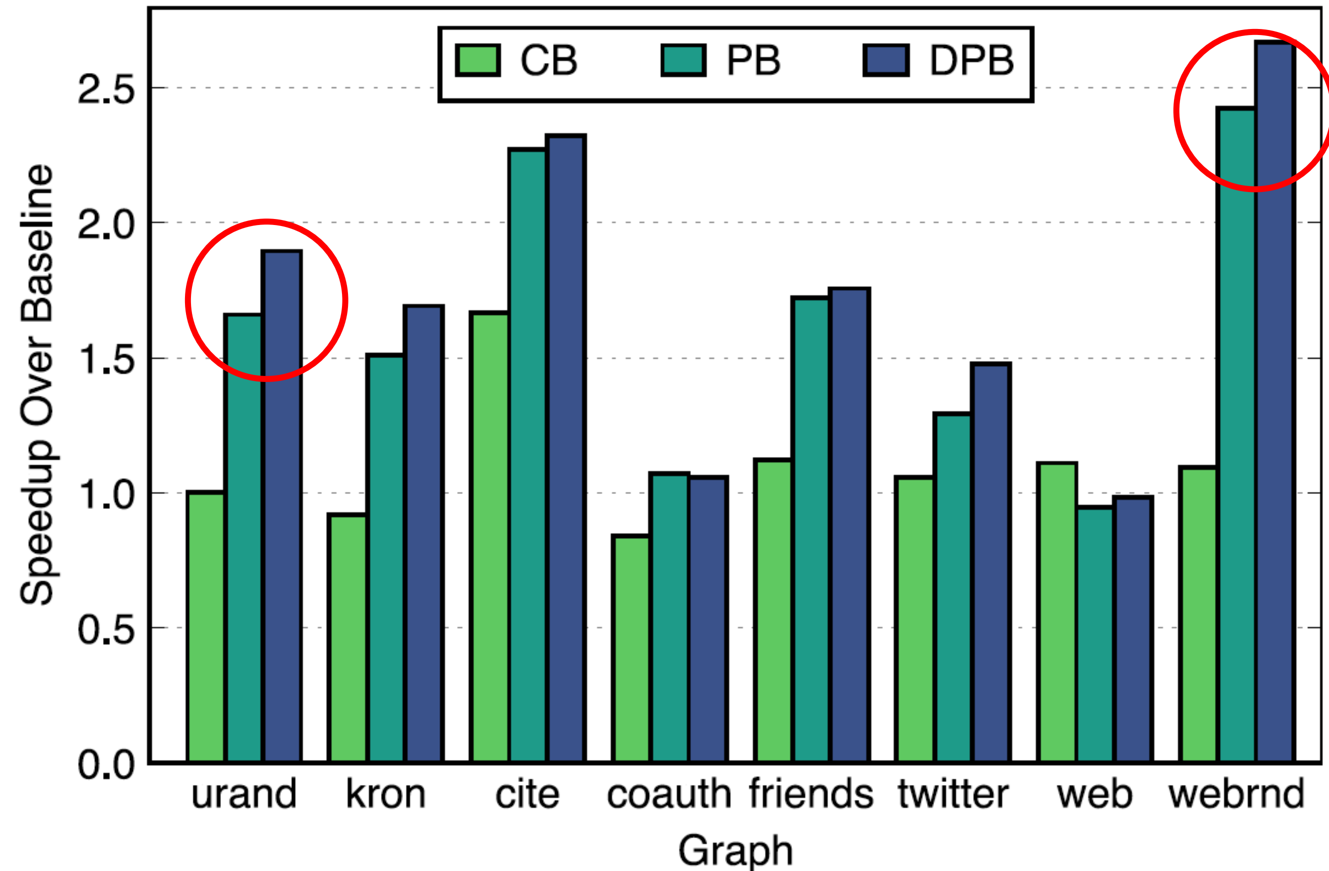
Results for Propagation Blocking(PB)

Legend

CB: Cache blocking

PB: Propagation blocking

DPB: Deterministic propagation blocking

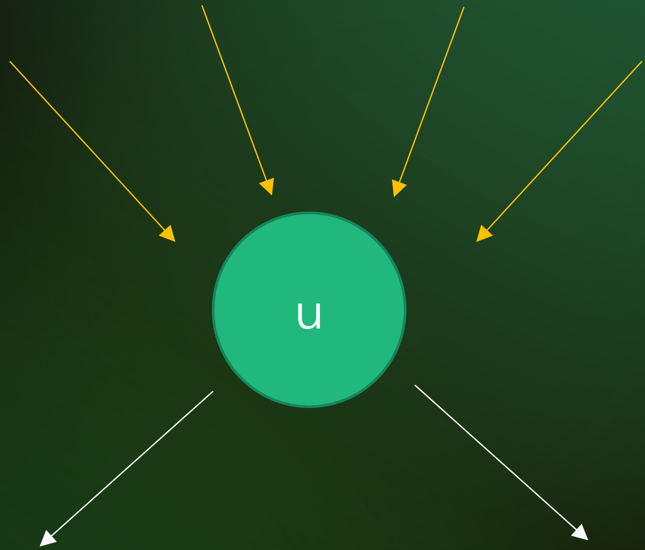


Presentation Outline

- Pagerank & the problem of locality
- Idea 1: Using cache blocking
- Idea 2: Propagation blocking
- Evaluation of Propagation Blocking
- Generalization to other applications

Pagerank & Locality: Pagerank variants

PageRank Terminology from paper

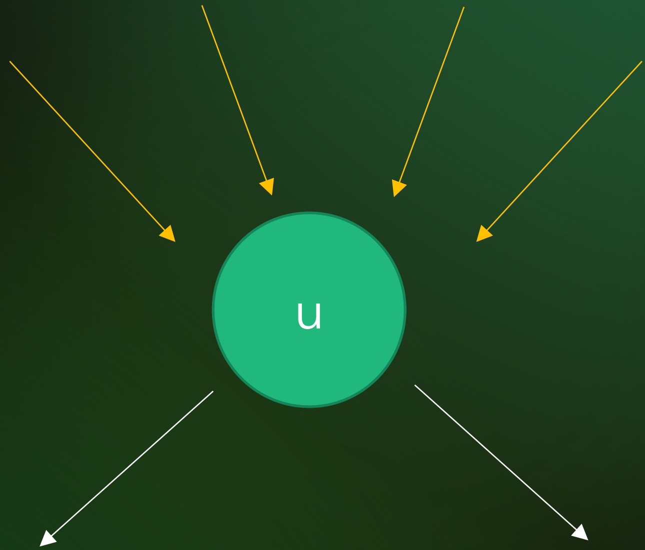


$$Importance(u)_t = \frac{1-d}{|V|} + d * \sum_{v \in N^-(u)} \frac{Importance(v)_{t-1}}{|N^+(v)|}$$

$N^+(v)$: outgoing neighbours
$N^-(v)$: incoming neighbours

Called **sum(u)**

PageRank Terminology from paper



$$Importance(u)_t = \frac{1-d}{|V|} + d * \sum_{v \in N^-(u)} \frac{Importance(v)_{t-1}}{|N^+(v)|}$$

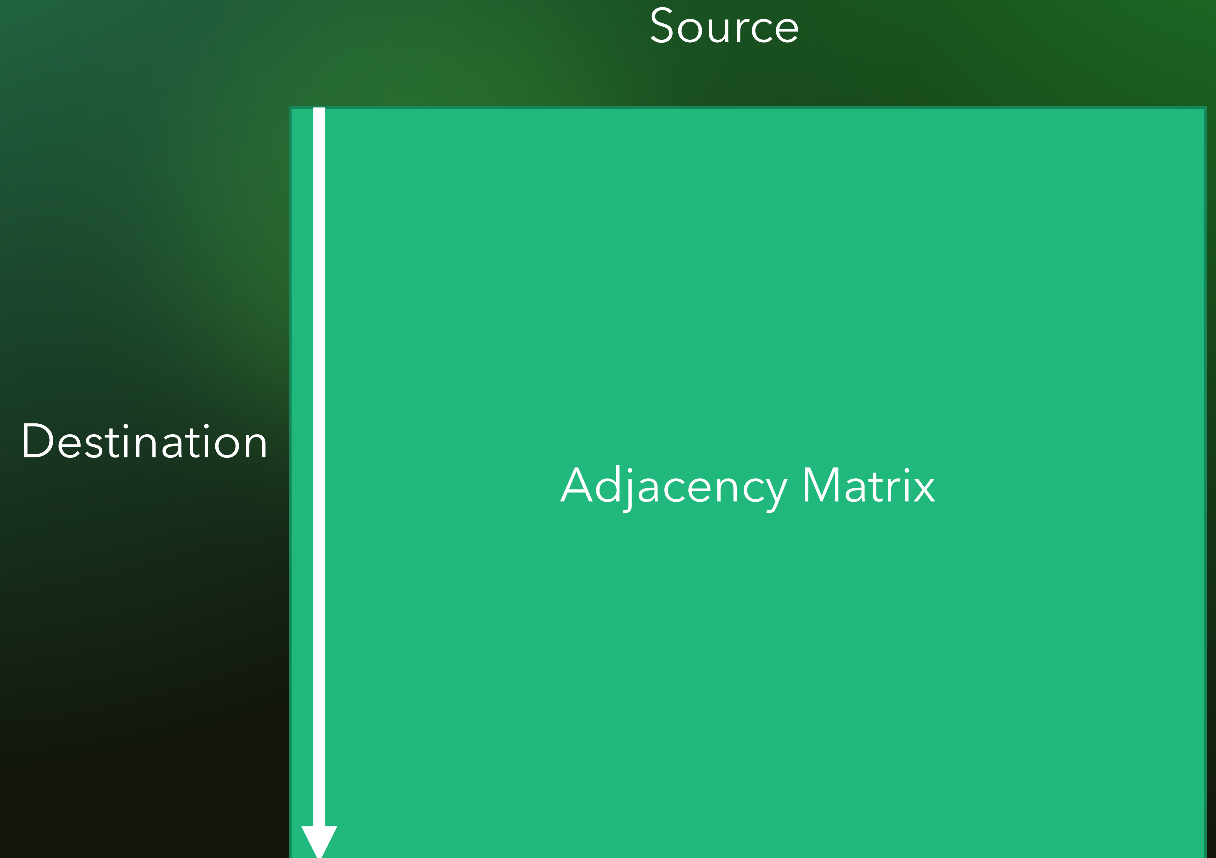
$N^+(v)$: outgoing neighbours
$N^-(v)$: incoming neighbours

Single term called the **contribution of v** to **sum(u)**

PageRank Pull Implementation

- First iterate over vertices and compute their contributions to their outgoing neighbours

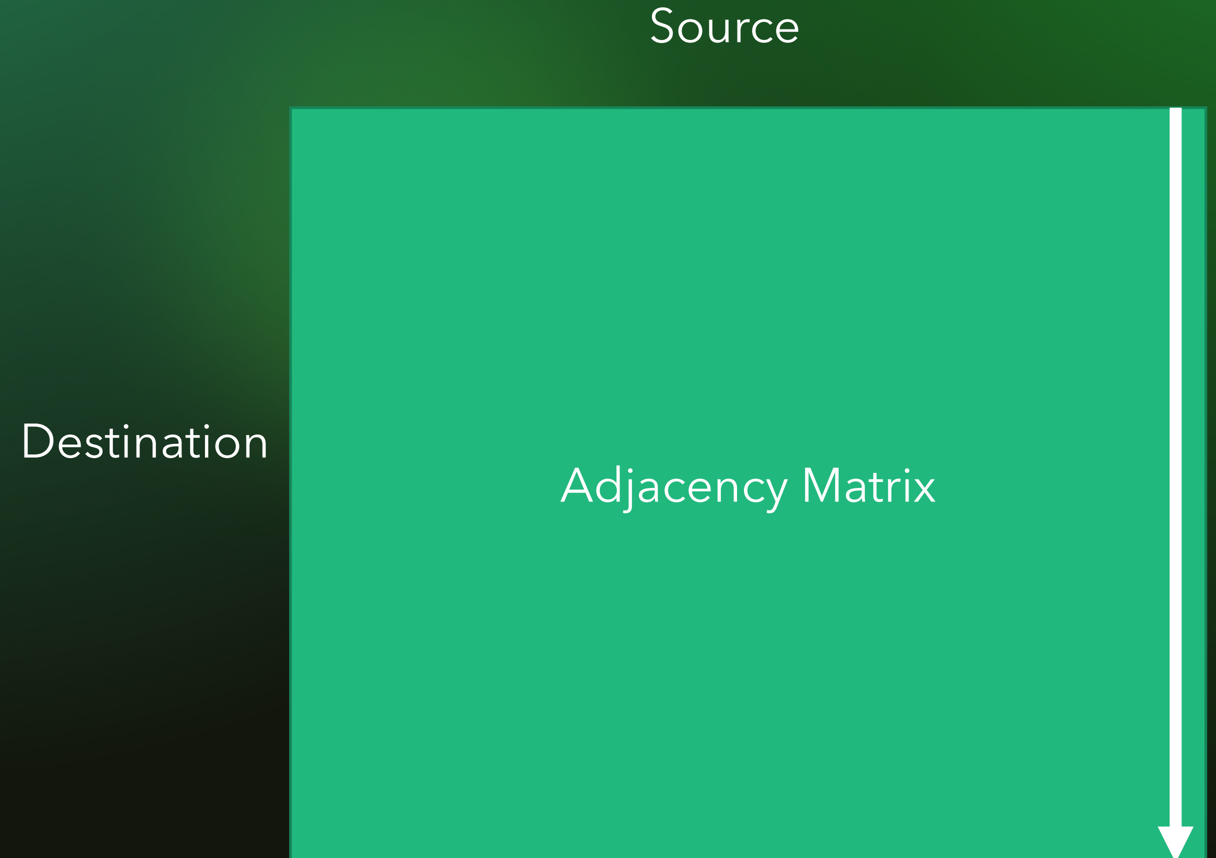
Contributions



PageRank Pull Implementation

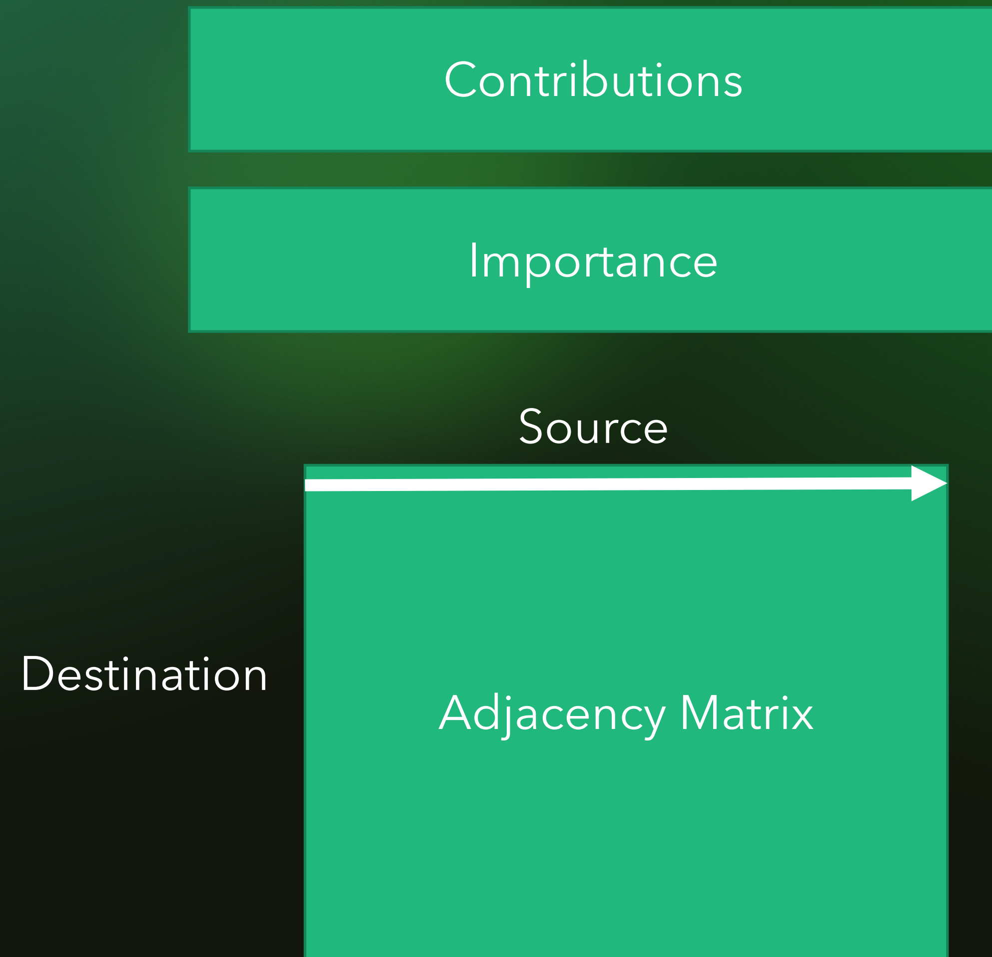
- First iterate over vertices and compute their contributions to their outgoing neighbours

Contributions



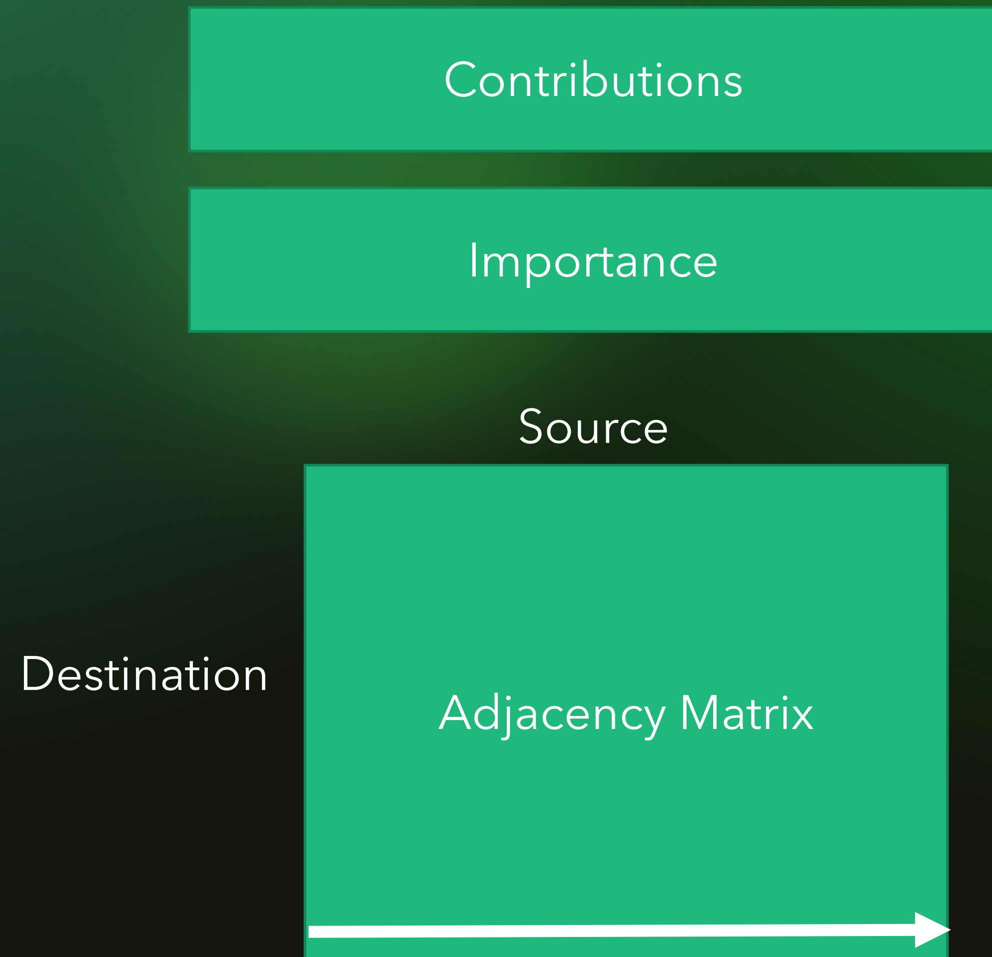
PageRank Pull Implementation

- First iterate over vertices and compute their contributions to their outgoing neighbours
- Next, iterate through each vertex and use the contributions computed to calculate the sum and importance



PageRank Pull Implementation

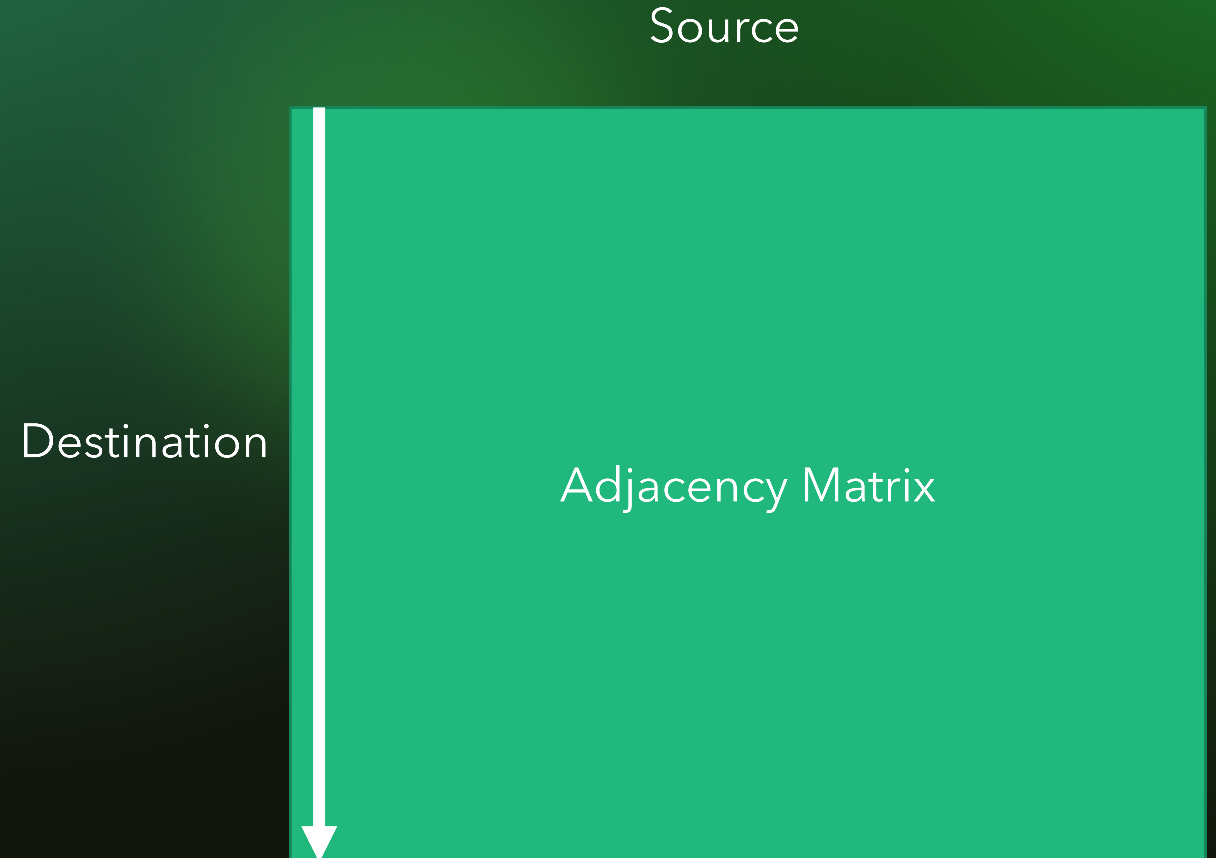
- First iterate over vertices and compute their contributions to their outgoing neighbours
- Next, iterate through each vertex and use the contributions computed to calculate the sum and importance



PageRank Push Implementation

- First iterate through each vertex and add its contribution to each outgoing neighbour's sum in the sums array

Sums



PageRank Push Implementation

- First iterate through each vertex and add its contribution to each outgoing neighbour's sum in the sums array

Sums

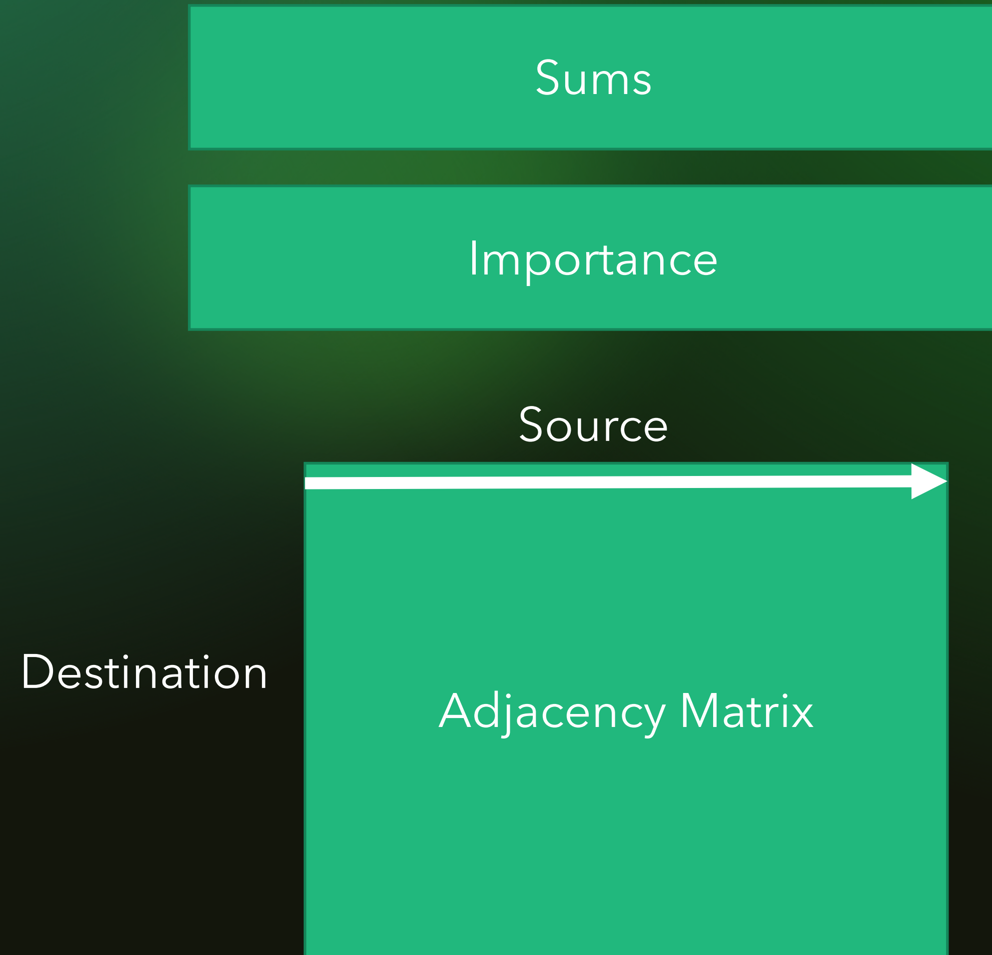
Source

Destination

PageRank Push Implementation

- First iterate through each vertex and add its contribution to each outgoing neighbour's sum in the sums array

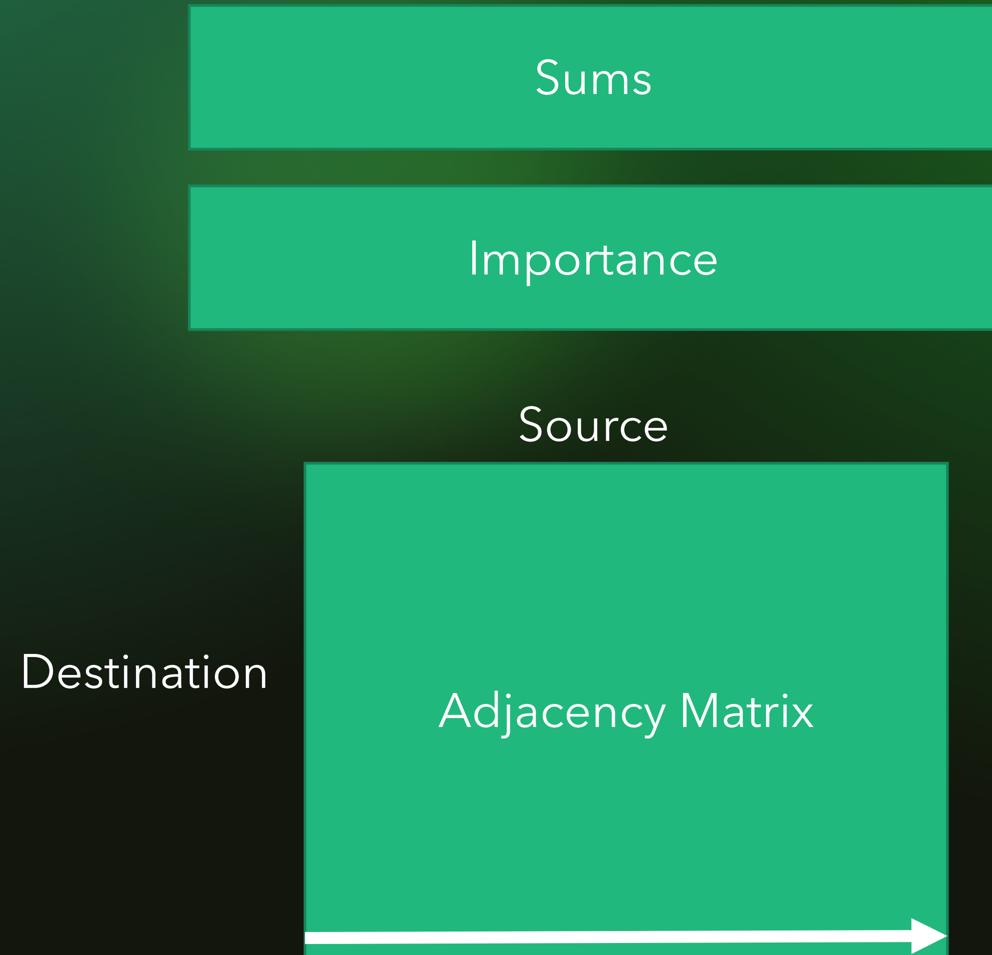
- Next, iterate through each vertex and compute it's importance using the computed sum



PageRank Push Implementation

- First iterate through each vertex and add its contribution to each outgoing neighbour's sum in the sums array

- Next, iterate through each vertex and compute it's importance using the computed sum



Pagerank & Communication: The problem of locality

Why Pagerank can incur high communication cost

- Both the contributions array and the sums array do not fit into cache*
- This means non-contiguous accesses to these arrays can lead to high communication costs because we encounter more cache misses
- Notice we don't have to worry about the adjacency matrix because the sparse matrix representation guarantees that we achieve good locality

*= for the graphs we are looking at at least

What’s the solution to this?

But what technique can we use when we have a 2D array and want to maximize locality?

What's the solution to this?

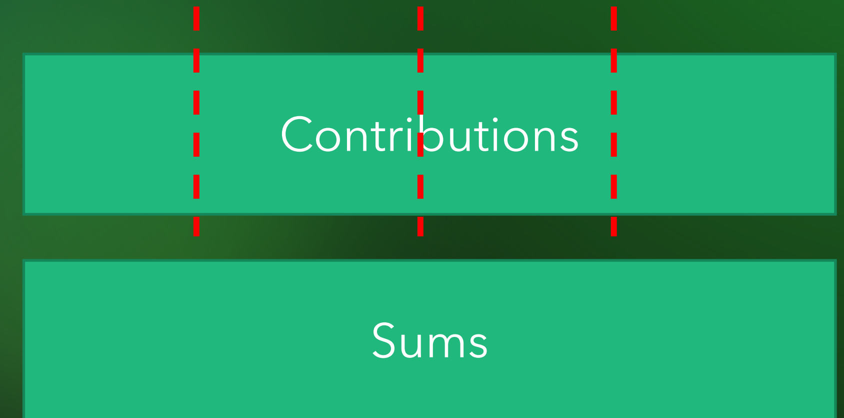
But what technique can we use when we have a 2D array and want to maximize locality?

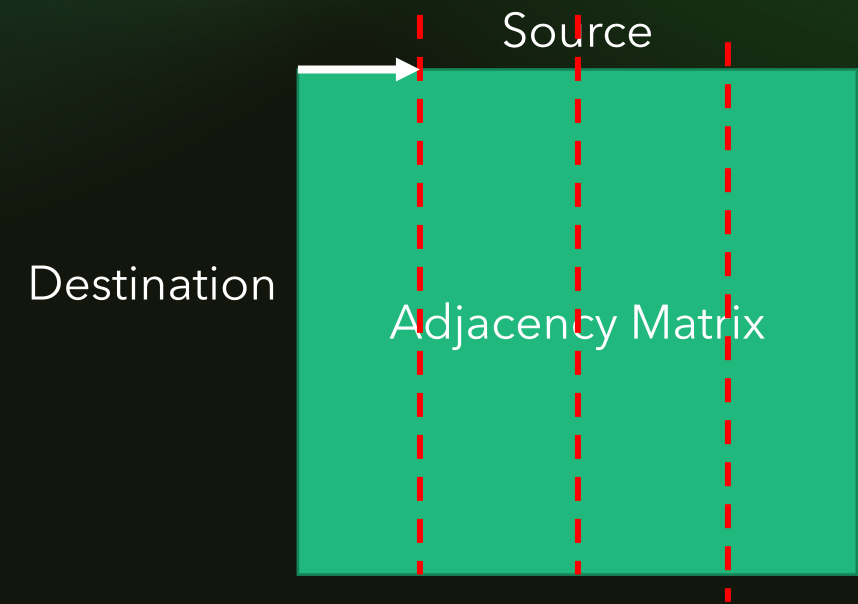
Blocking to improve locality/reduce communication costs

Idea 1: Cache Blocking

Cache Blocking to improve locality in pull direction

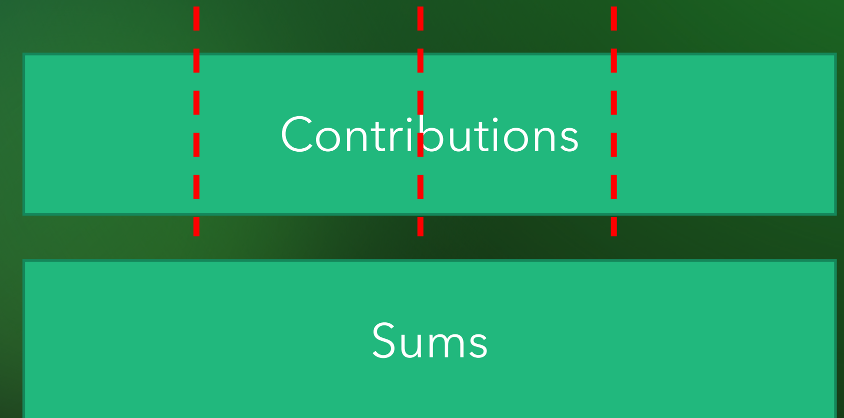
- When reading from the contributions array, first break up the array into blocks/tiles
- Create a sums array
- Go block by block reading the contribution array and adding it to sums array

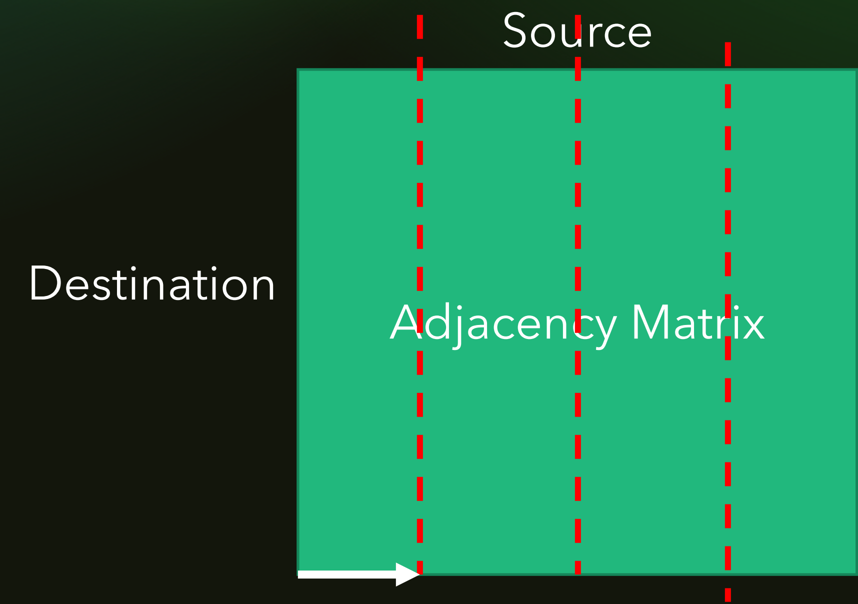




Cache Blocking to improve locality in pull direction

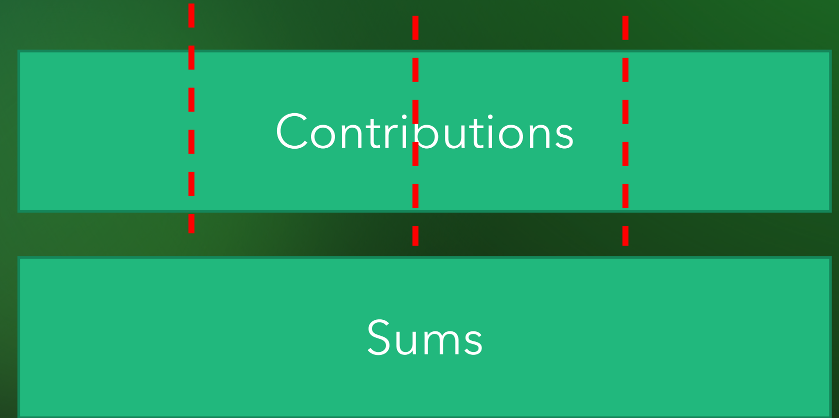
- When reading from the contributions array, first break up the array into blocks/tiles
- Create a sums array
- Go block by block reading the contribution array and adding it to sums array

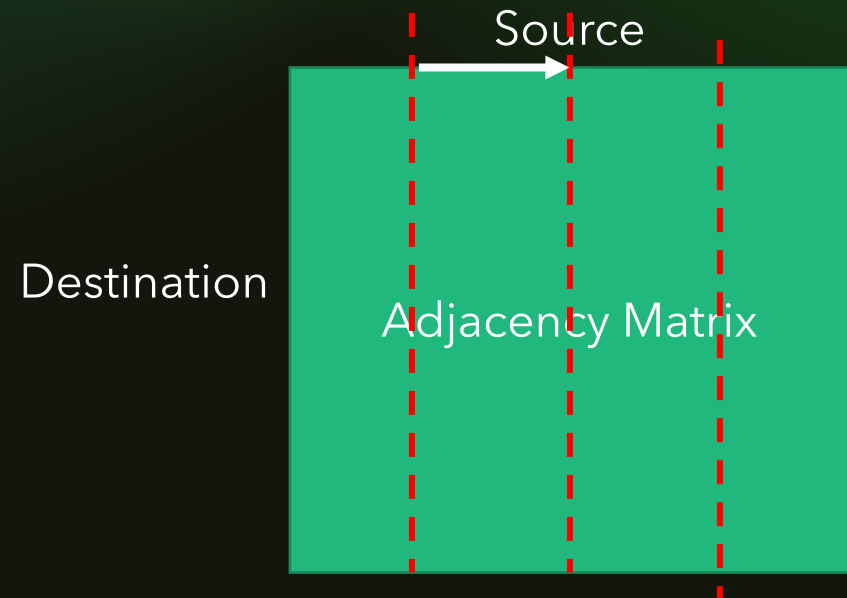




Cache Blocking to improve locality in pull direction

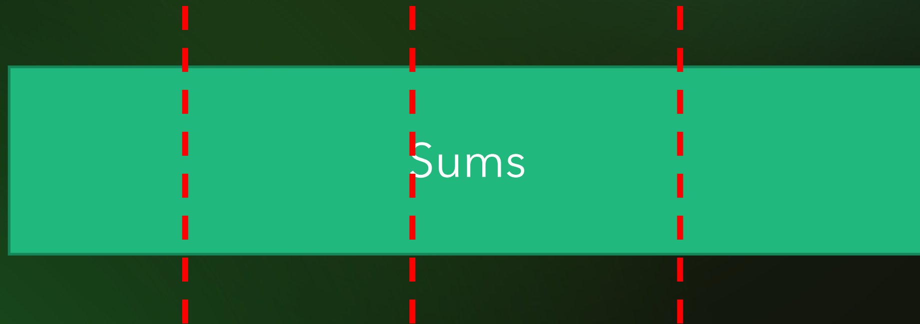
- When reading from the contributions array, first break up the array into blocks/tiles
- Create a sums array
- Go block by block reading the contribution array and adding it to sums array

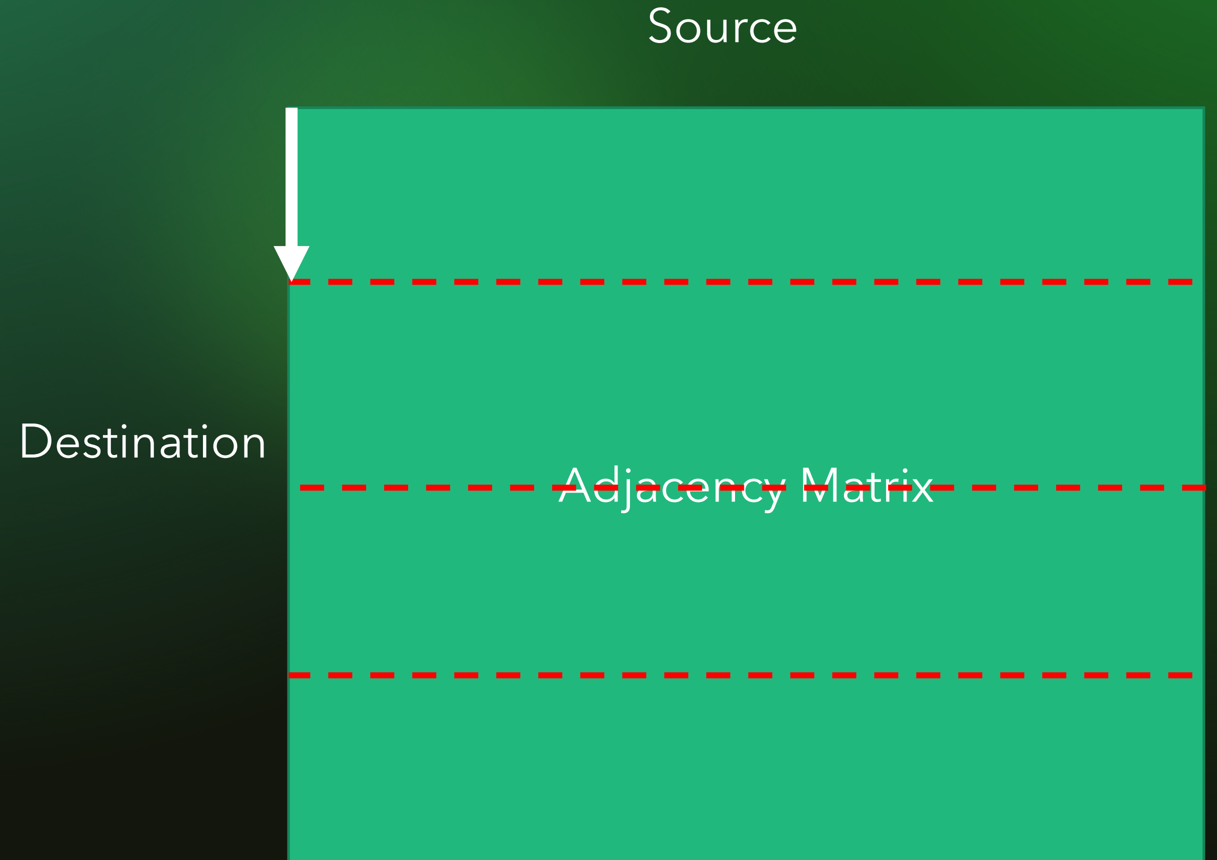




Cache Blocking to improve locality in push direction

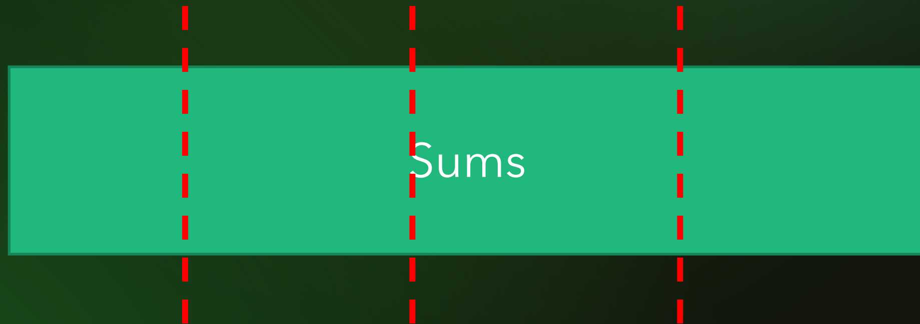
- When computing the sums array, break up the graph into blocks and compute sums for each vertex block by block

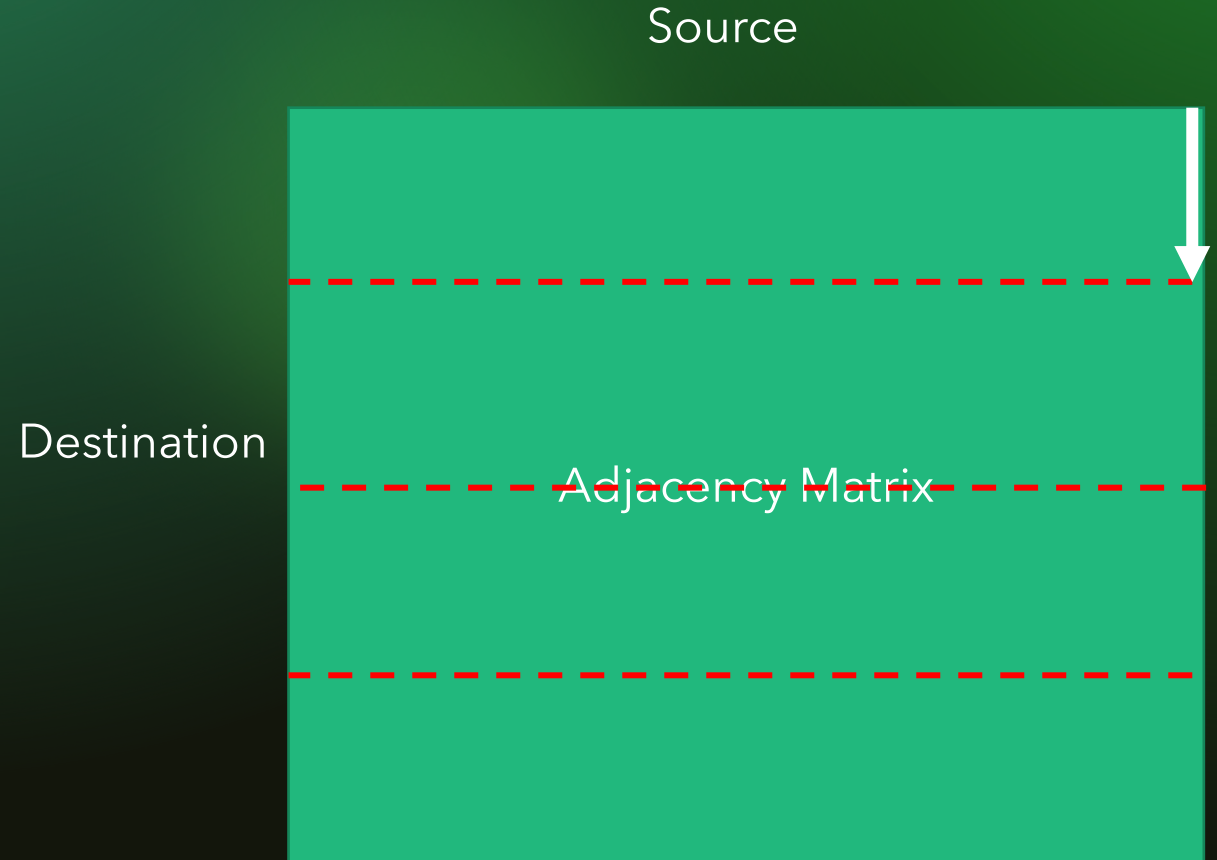




Cache Blocking to improve locality in push direction

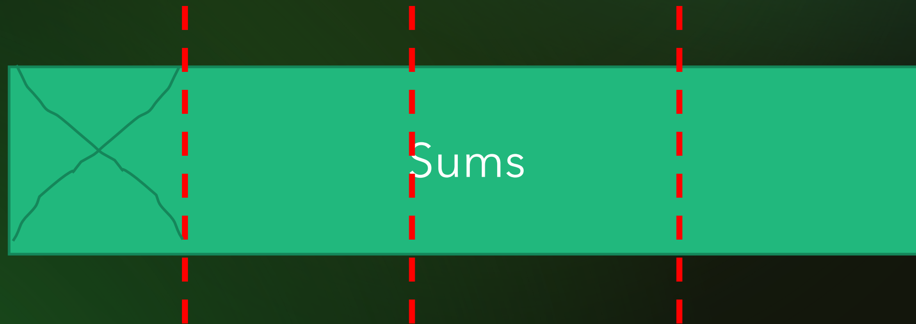
- When computing the sums array, break up the graph into blocks and compute sums for each vertex block by block

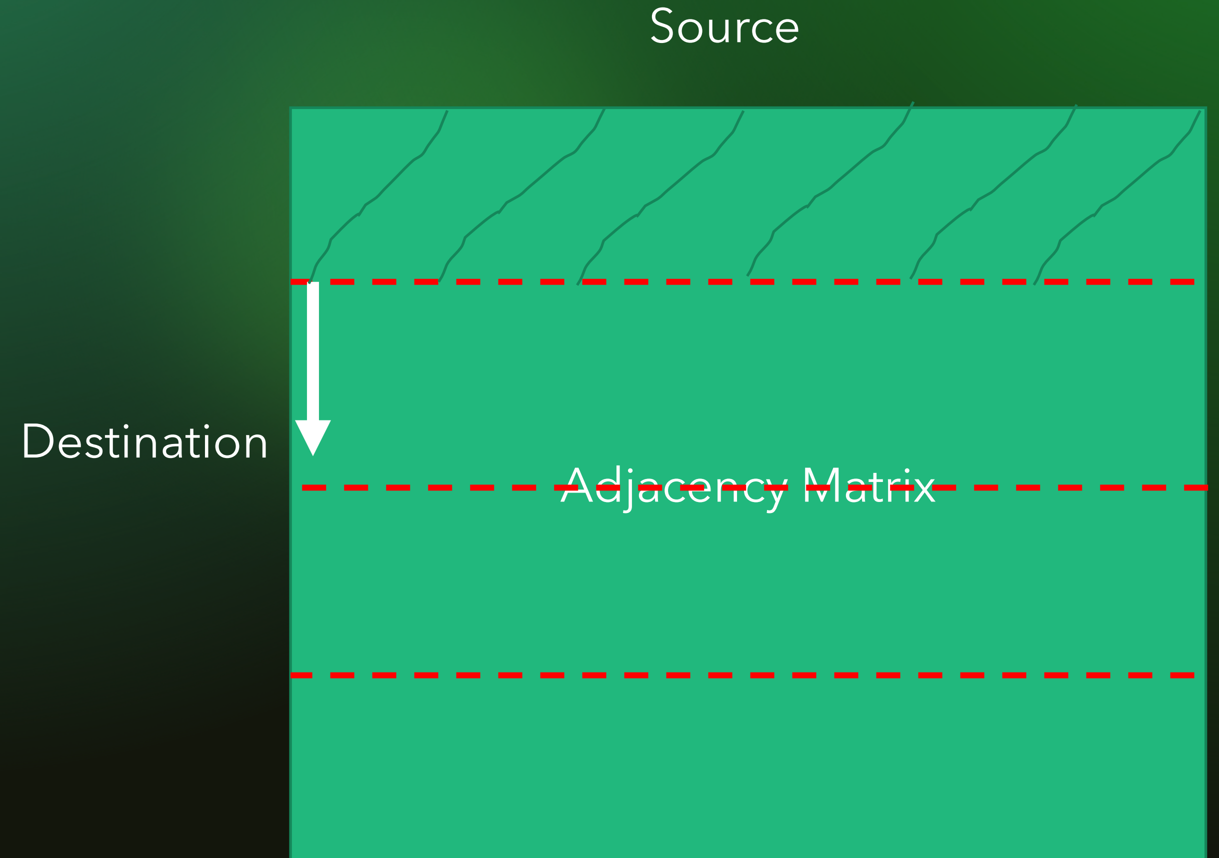




Cache Blocking to improve locality in push direction

- When computing the sums array, break up the graph into blocks and compute sums for each vertex block by block





We still have a problem!

- For the pull direction we made it better for reading values from the contributions array but made it worse for calculating sums
- For the push direction we made it better for writing values to the sums array but may have made it worse for calculating contributions.
- Also cache blocking doesn't scale!

Idea 2: Propagation Blocking

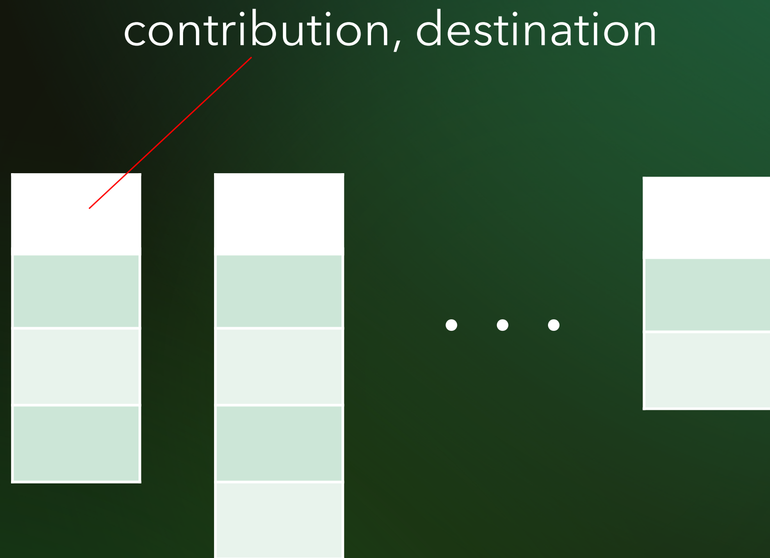
Propagation blocking definition

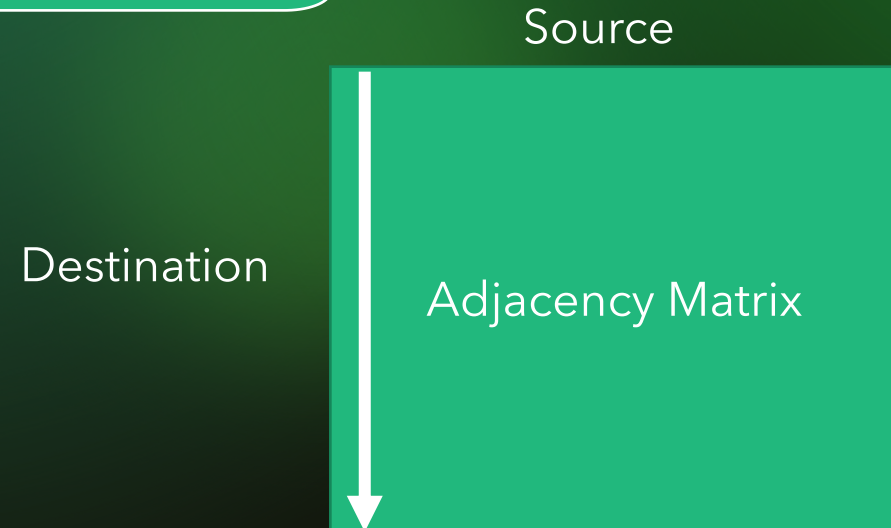
- We will block propagations rather than the graph!
- Propagations here are the contribution each vertex makes to its outgoing neighbours' sums.
- This way our blocking **scales with the updates per round** and not the graph size per se.

Propagation blocking stages

Propagation blocking first phase

Binning





- Sub-divide your destination vertices into bins
- Note that multiple destination vertices will map to a bin
- Vertices next to each other are in the same bin

- As we compute the contribution of a source to it's destination vertices, we do not add this to the sums array
- We first put it in the corresponding bin of that destination vertex
- Because multiple vertices map to a bin, you must include the destination vertex of the contribution

Propagation blocking second phase

- Process each bin consecutively
- Adding the contribution to the sum of the destination vertex in the sums array

Propagation blocking second phase

- Process each bin consecutively
- Adding the contribution to the sum of the destination vertex in the sums array

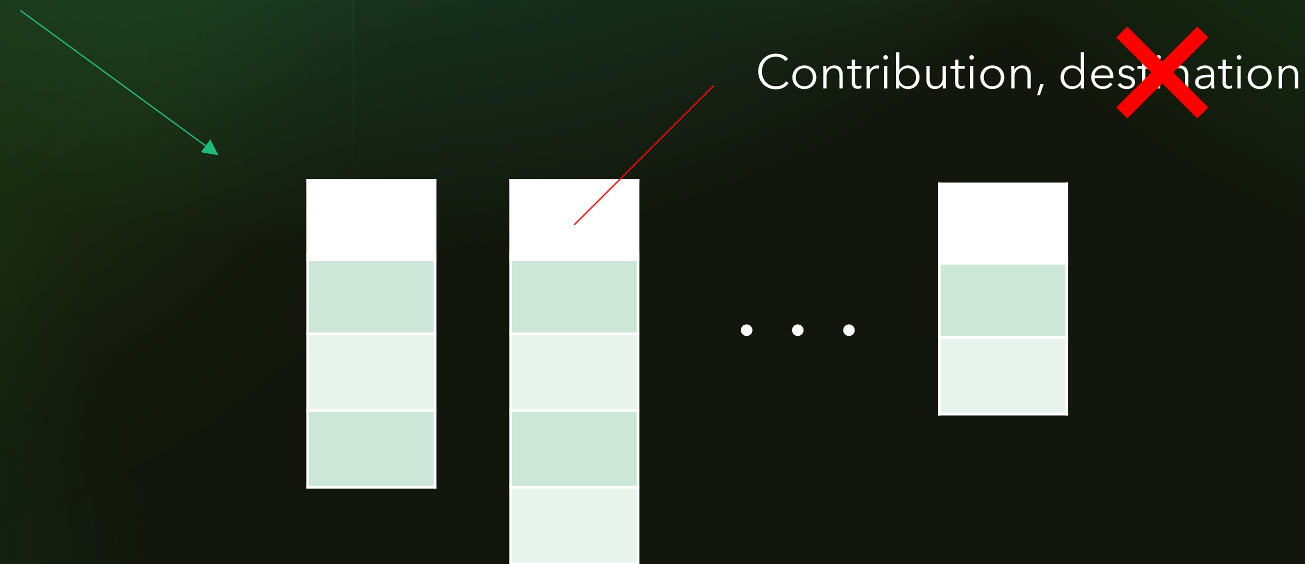
Why is propagation blocking a good idea?

- The paper focuses on running Pagerank on a sparse graph and so the number of updates to vertices is relatively small (low edge traffic)
- This means the space taken up by buckets << the number of vertices
- So we are better off writing to buckets first than directly to the sums array
- This way, when we write back to the sums array we will enjoy high spatial locality and subsequently lower communication cost

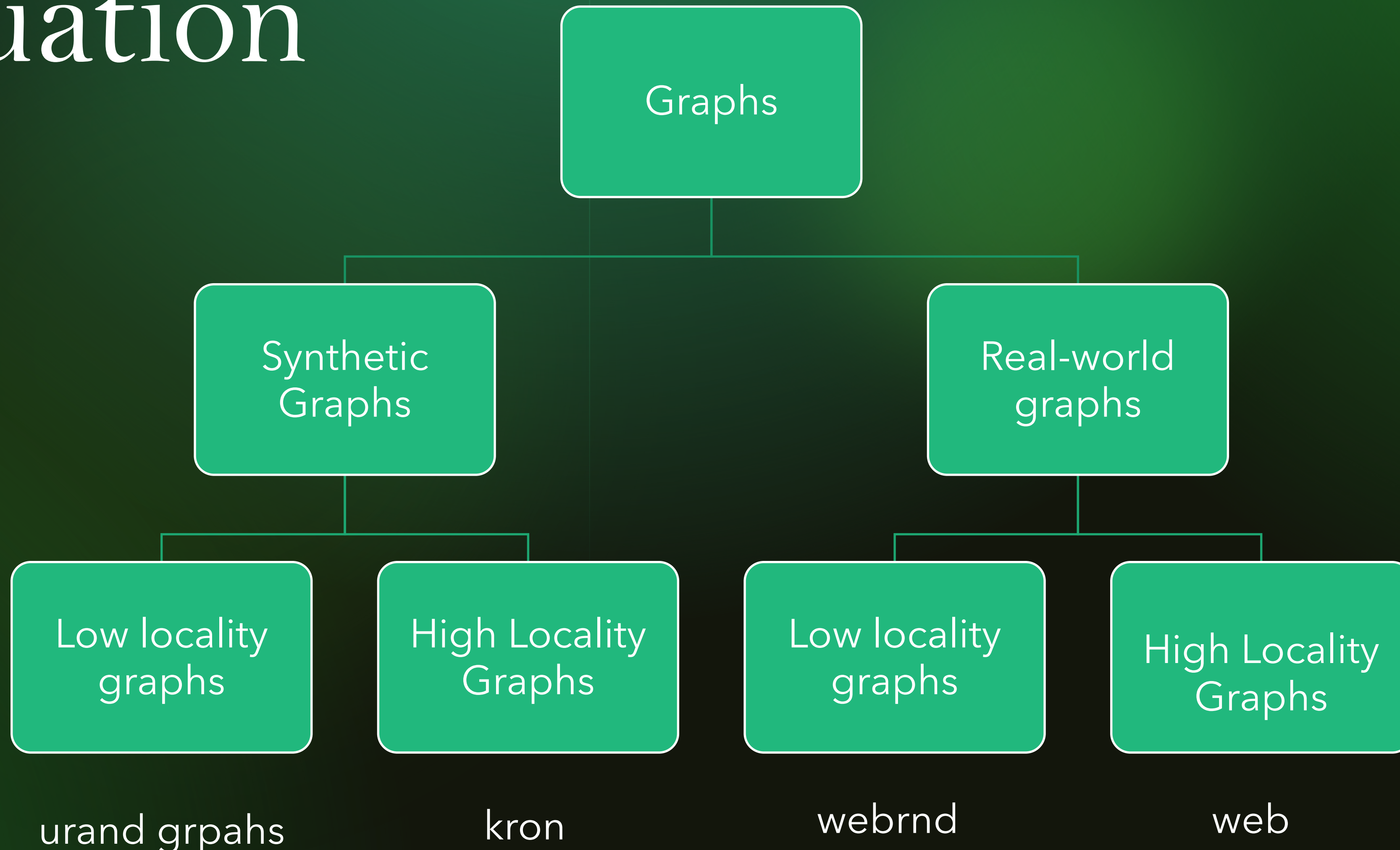
Evaluating Propagation Blocking

What do we compare?

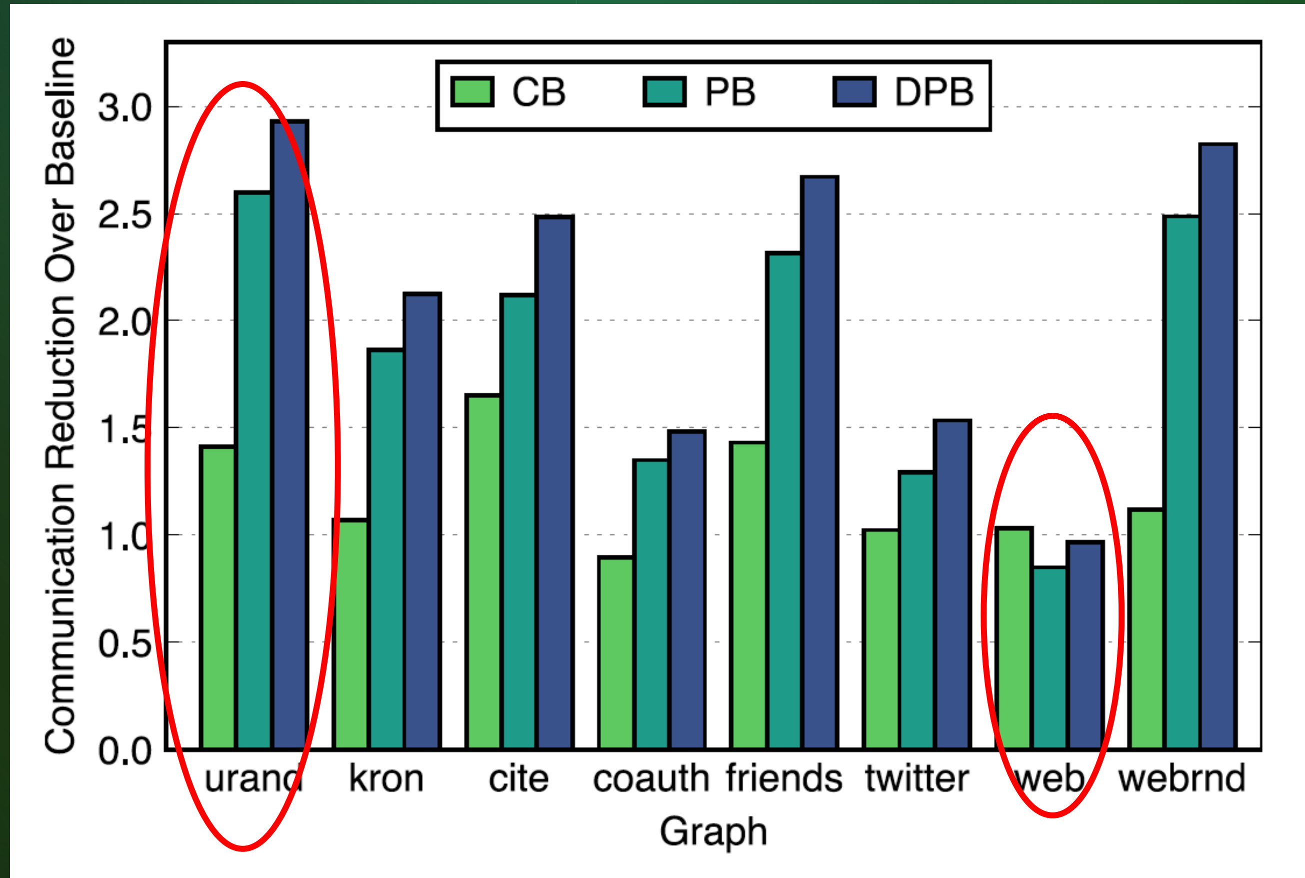
- They compare the performance of four different PageRank implementations:
  - Baseline(PageRank implementation from existing graph processing libraries)
  - Cache Blocking
  - Propagation Blocking
  - Deterministic Propagation Blocking



Taxonomy for graphs used in evaluation

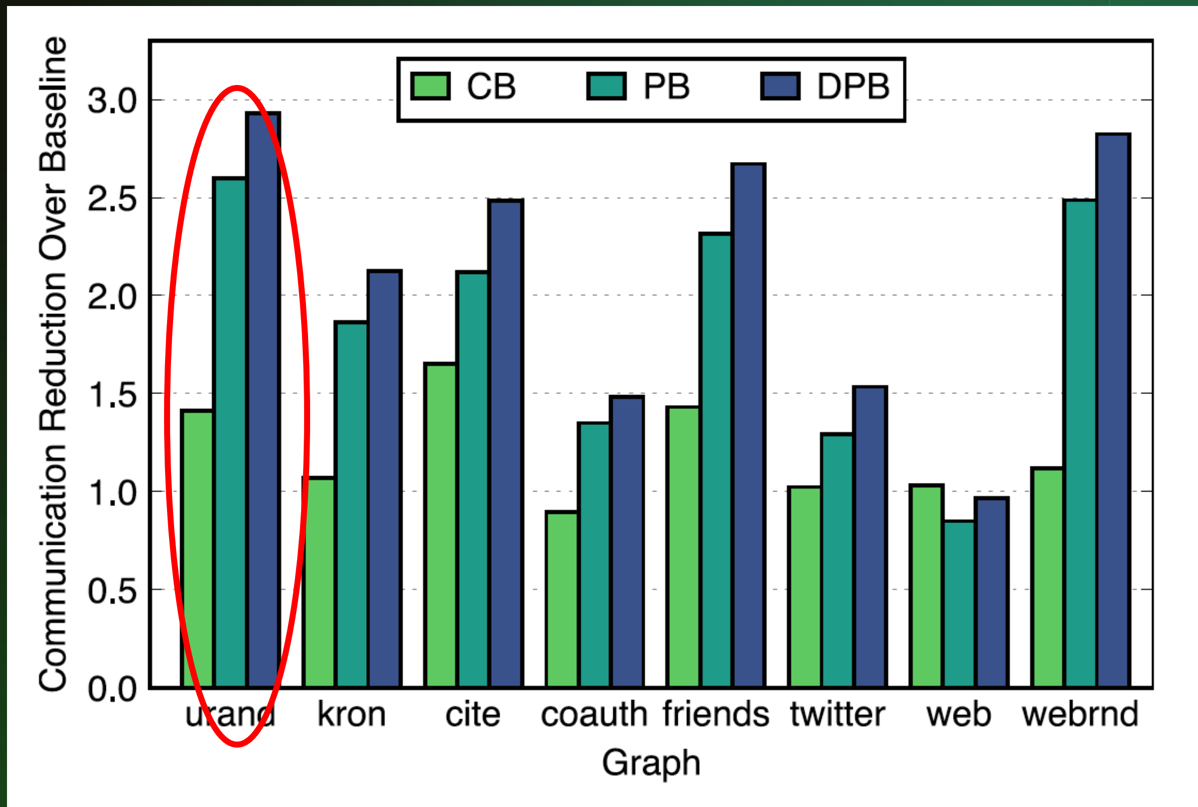


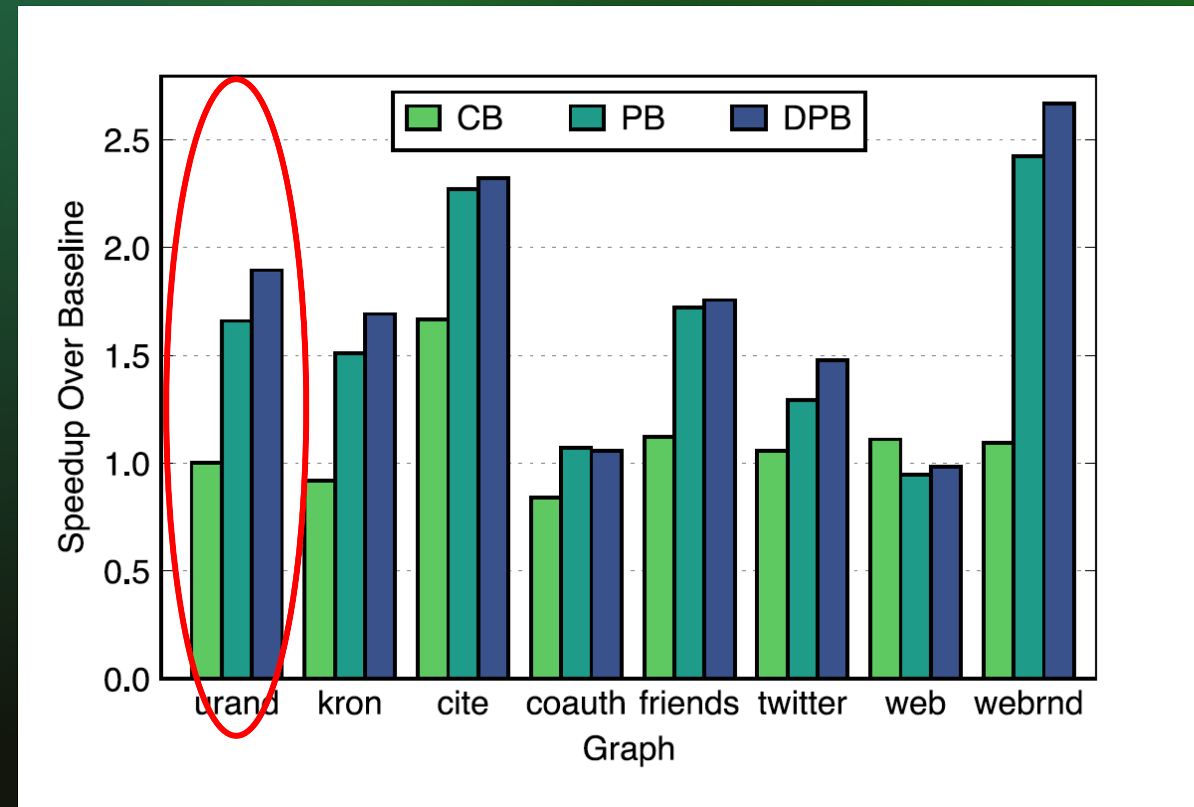
Communication reduction for graphs



Significant speedups for those graphs that suffer from low locality.

Communication reduction vs. Speedup





The decrease in communication cost does not translate to an equivalent decrease in runtime.

Best technique depends on the size of the graph

- For small graphs, baseline implementation without blocking provides the best performance
- For medium sized graphs, cache blocking with the push implementation provides the best performance
- For large graphs, propagation blocking works best and scales best.
- Here small, medium and large is relative to size of cache.

Generalization of Propagation Blocking

Applications beyond PageRank

- Not limited to PageRank
- In fact propagation blocking is a powerful idea that tries to scale communication cost with the amount of actual computation work we do
- Idea also applicable to SpMV(sparse matrix vector multiplication) since most graph problems share a duality with matrix computation problems