Big Data Computing

Master's of Science in Computer Science

Sapienza University of Rome

2026/2027

Gabriele Tolomei

tolomei@di.uniroma1.it

Module 3: Scaling Classical Tasks

Lecture 3.2: Link Analysis and PageRank

Philosophy of Link Analysis

> *"A page is important if important pages point to it."*
> *— The recursive idea behind PageRank*

Lecture 3.1 answered "which pages match this query?" exactly. But thousands of pages may match equally well. Today we ask a different question about the same crawl: ***which pages deserve to be on top?*** *The answer is hidden not in the text, but in the link structure.*

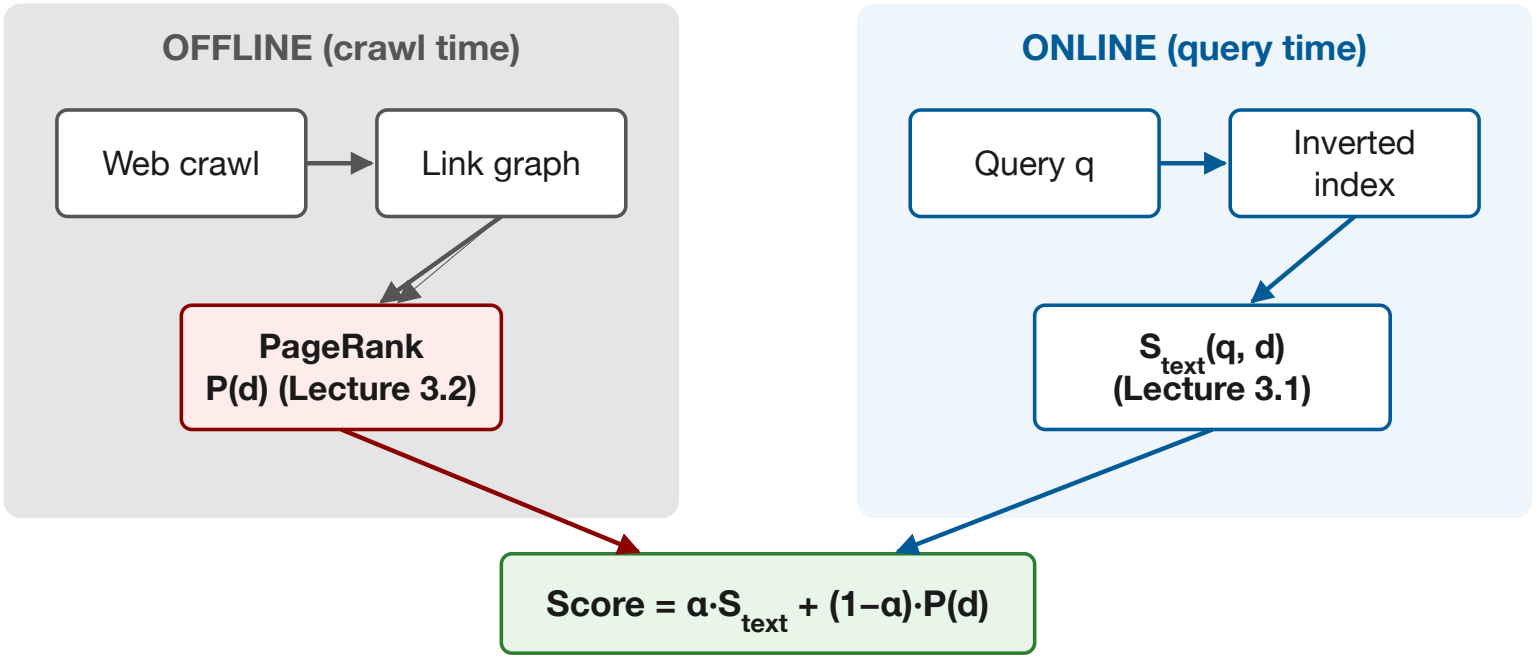
Lecture Roadmap

We build **PageRank** one small step at a time, in **7 sections**:

1. **The Importance of a Node** — links as votes
2. **From Votes to Equations** — the "flow" model
3. **Matrix, Eigenvector, Power Iteration** — a first algorithm
4. **The Random Surfer** — the probabilistic view
5. **Google's Fix** — dead ends, spider traps, teleporting
6. **Computing PageRank at Scale** — sparse iteration
7. **Synthesis** — take-home and bridge to Lecture 3.3

> *One tiny graph (3 pages) runs through the whole lecture. Proofs, derivations, numerical traces, MapReduce/Spark code, personalized PageRank, HITS and exam problems are in the **Appendix**.*

Why This Matters: Authority Is the Missing Signal



- $P(d)$ is **query-independent** and computed **offline**: where does it come from? **PageRank.**

Same crawl, new question: not "which pages match?", but "which pages matter?"

§ 1 — The Importance of a Node

Links as votes

📎 *Extended material → Appendix A.7*

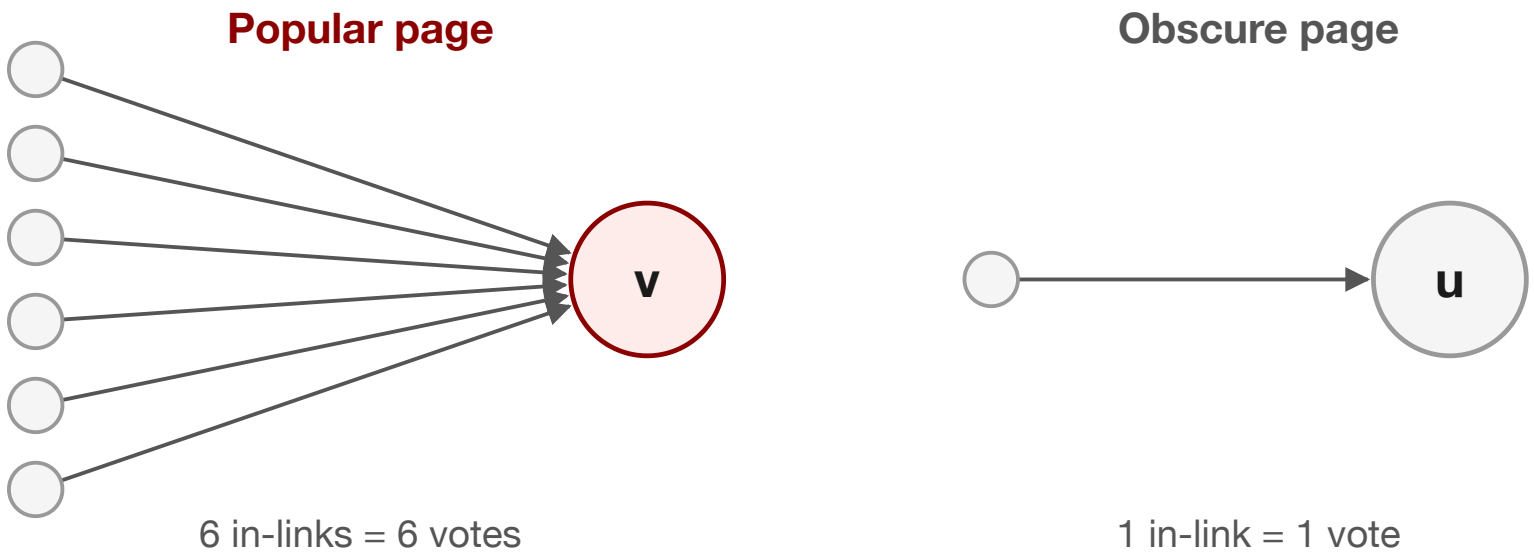
Goal: The "Importance" of a Node

- Give each page a **score** that says how **important** it is, using the **structure of the graph only** (**link analysis**).
- **PageRank**: introduced in **1998** by Sergey Brin and Larry Page, the core of the **Google** search engine.
- The Web is a **scale-free network** (Lecture 2.4): a few hubs, a huge number of pages with one or two in-links.

Brin & Page (1998), "The Anatomy of a Large-Scale Hypertextual Web Search Engine", Computer Networks 30(1–7).

PageRank's Intuition (1/2): Links as Votes

Intuition 1. The more incoming links a page has, the more important it is.

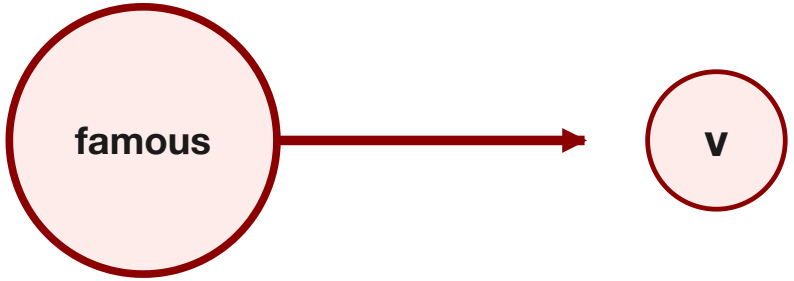


*Each link from w to v is a **vote** by w for v.* `www.stanford.edu` *has more than 23K in-links.*

PageRank's Intuition (2/2): Not All Votes Are Equal

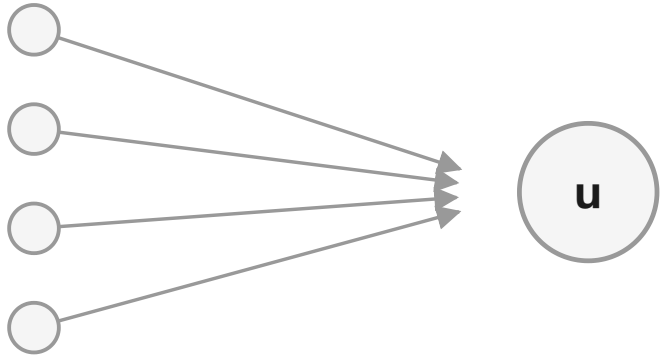
Intuition 2. A vote from an **important** page counts **more**.

One vote from an important page…



…is worth a lot

…versus several votes from nobodies

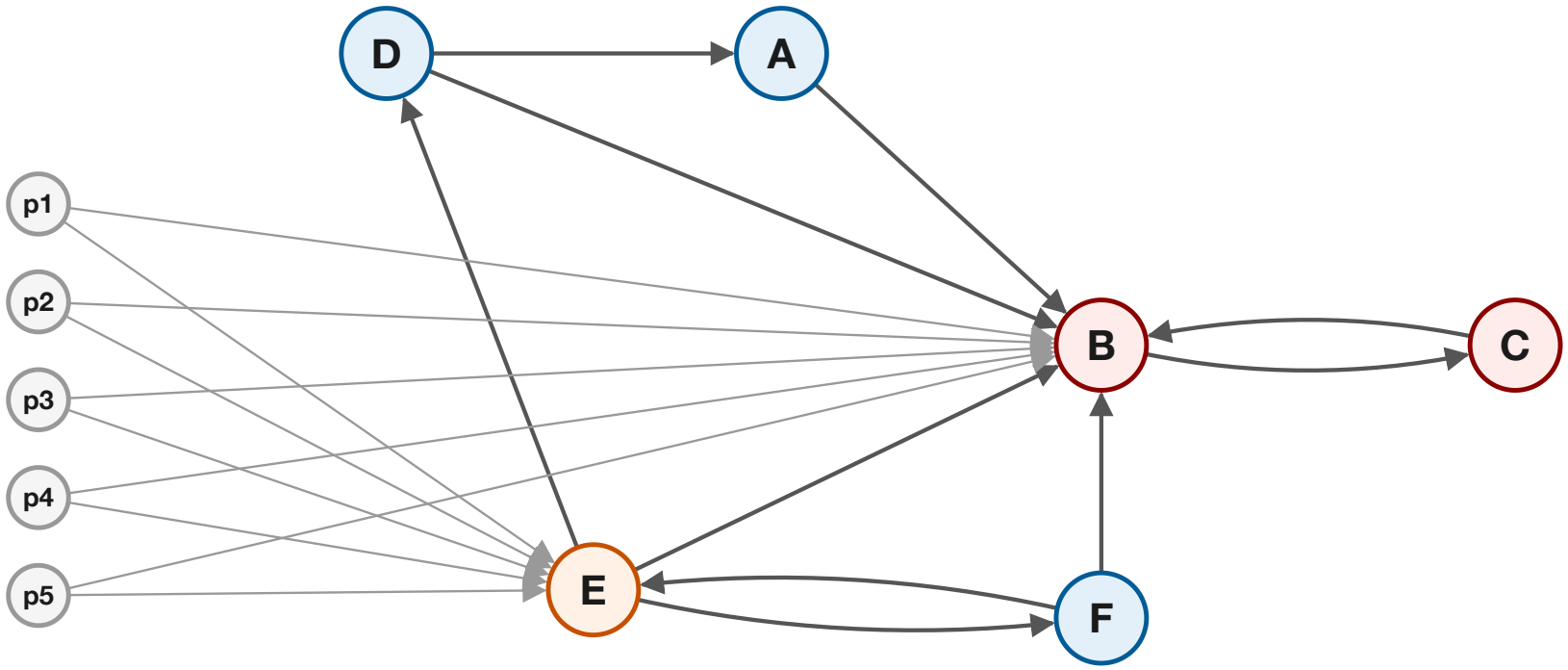


…are worth little each

*But which pages are important? Those with important pages voting for them: a **recursive definition**.*

Example: A Small Web

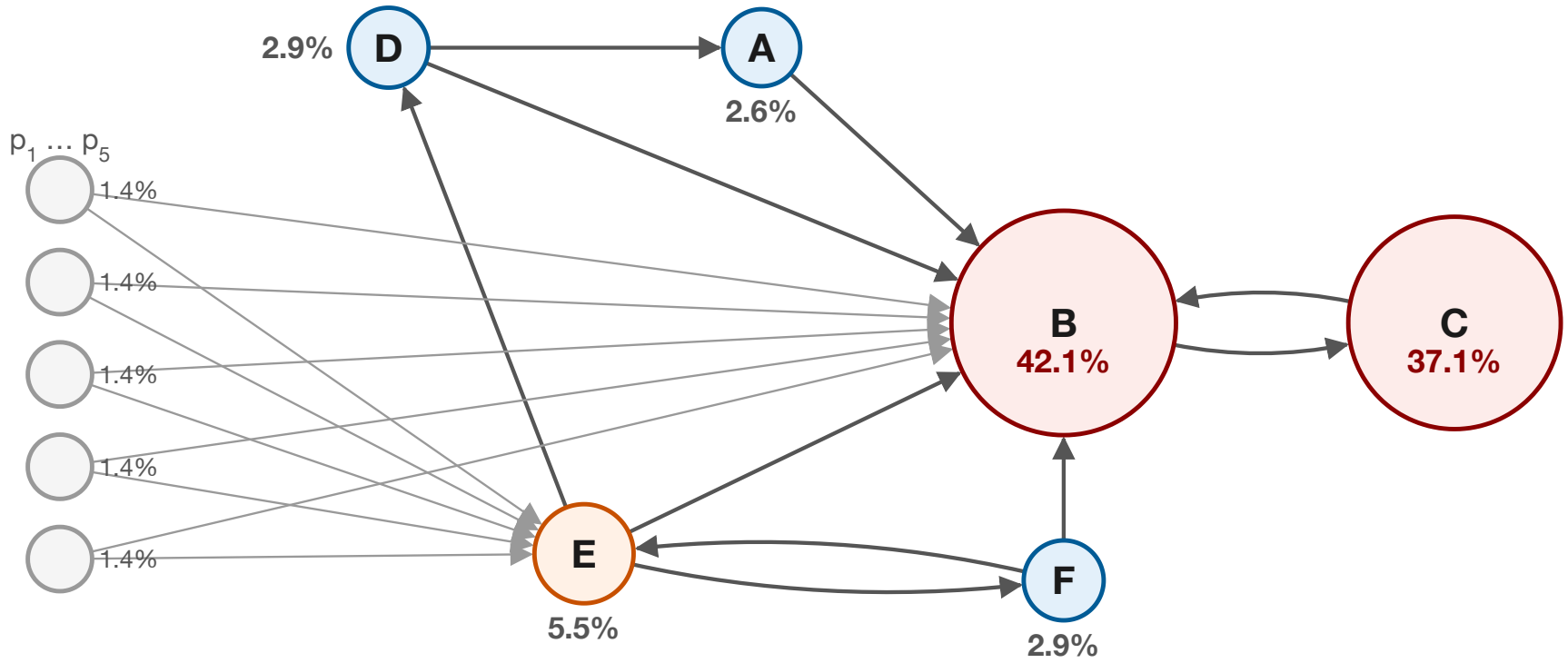
Eleven pages. Before computing anything: **who do you expect on top?**



B has 10 in-links, E has 6, every other page has at most 1.

Example: The PageRank Scores

Node size = PageRank score (full method of §5, $d = 0.85$).



*C is second with only **one** in-link, but it comes from the important page B.*

✅ Checkpoint 1 — The Importance of a Node

Test your understanding:

1. Why is $P(d)$ computed offline while S_{text} is computed online?
2. State the two intuitions behind PageRank in your own words.
3. In the example, why does C outrank E, although E has six in-links and C only one?
4. Why is the definition "recursive"?

Consolidate your answers before proceeding.

§ 2 — From Votes to Equations

The "flow" model

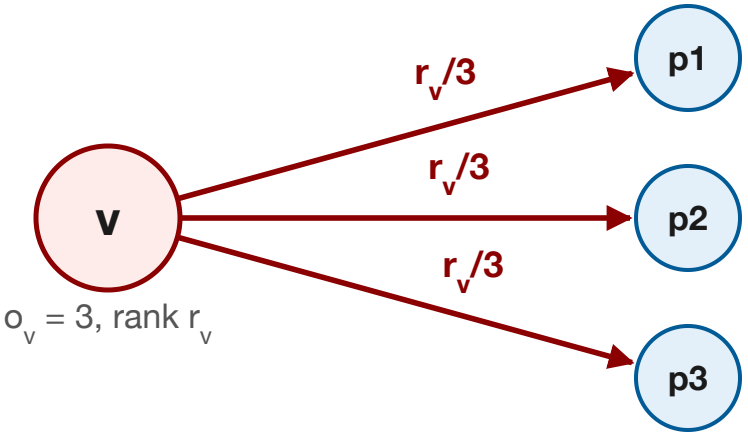
📎 *Extended material → Appendix A.8*

Notation

Symbol	Meaning
(V, E)	the Web graph; $\lvert V \rvert = N$ pages
O_v	set of pages that v **links to**
$o_v = \lvert O_v \rvert$	**out-degree** of v
I_v	set of pages that **link to** v
r_v	**rank** (importance) of page v

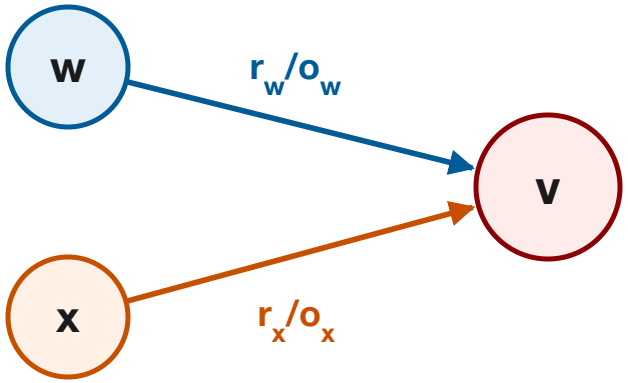
A Vote Is Shared; a Rank Is a Sum

A page splits its rank



each out-link carries an equal share

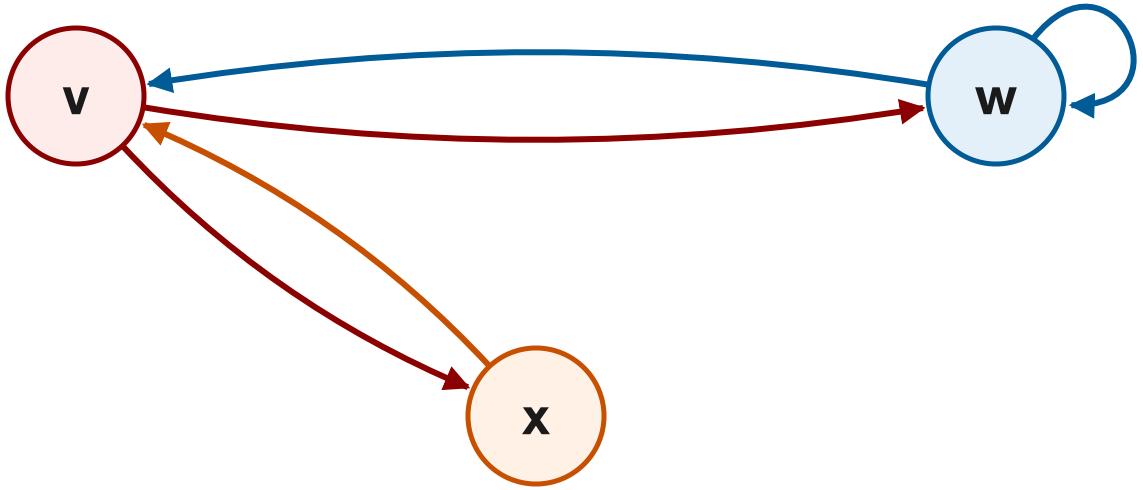
A page sums what it receives



$r_v = r_w/o_w + r_x/o_x$

$$r_v = \sum_{w \in I_v} \frac{r_w}{o_w}$$ (one equation per page: N equations, N unknowns)

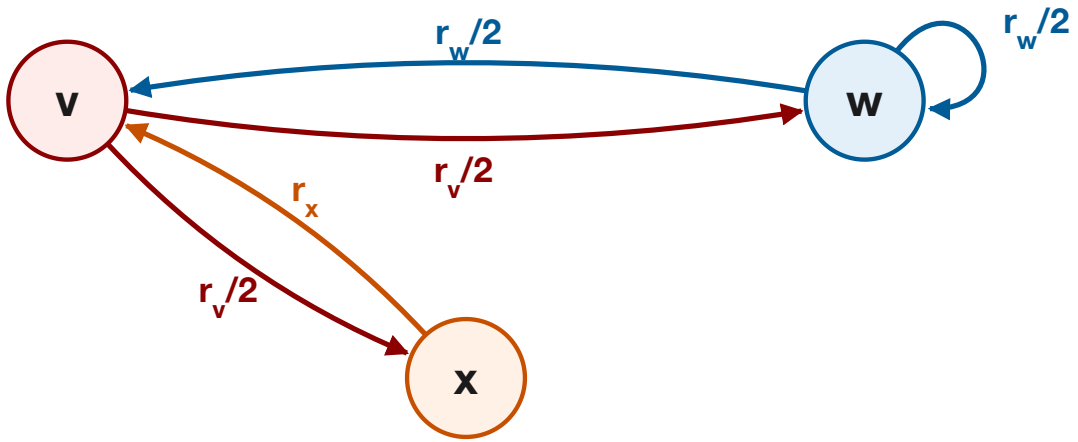
Running Example: Three Pages



- Out-degrees: $o_v = 2$, $o_w = 2$, $o_x = 1$. Page w links to itself (a self-loop).

We use this graph until the end of the lecture.

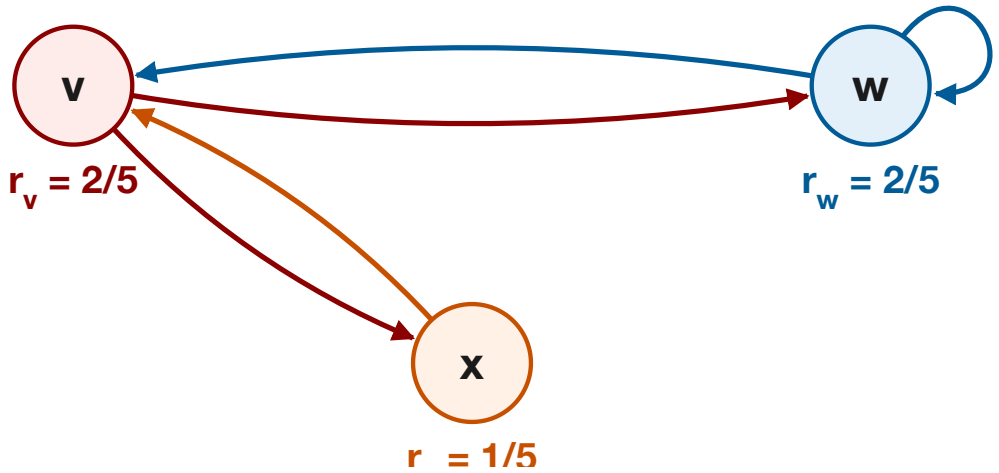
The "Flow" Equations



$$\begin{cases} r_v = r_w/2 + r_x \\ r_w = r_v/2 + r_w/2 \\ r_x = r_v/2 \end{cases}$$

Solving the System

The equations are **not independent**: they only fix the **proportions** between ranks (Appendix A.8). Add the constraint $r_v + r_w + r_x = 1$:



- On the Web: **hundreds of billions** of equations. Solving them directly is hopeless. *We need a new formulation.*

✅ Checkpoint 2 — From Votes to Equations

Test your understanding:

1. In the formula $r_v = \sum_{w \in I_v} r_w / o_w$, why do we divide by o_w?
2. Write the flow equation of a page with in-links from a (out-degree 4) and b (out-degree 1).
3. Why does the system of flow equations have infinitely many solutions without the constraint $\sum r = 1$?
4. Why can't we just solve the system with Gaussian elimination on the Web?

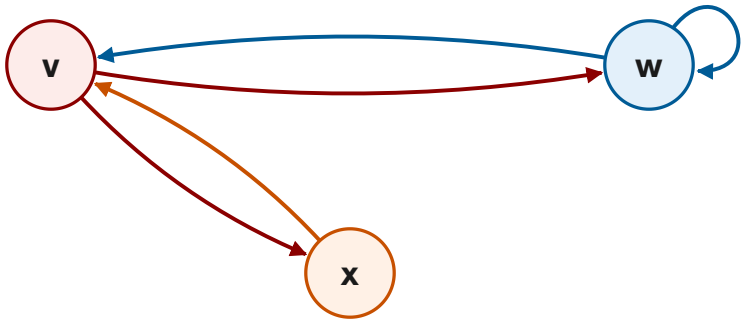
Consolidate your answers before proceeding.

§ 3 — Matrix, Eigenvector, Power Iteration

A first algorithm

📎 *Extended material → Appendix A.1, A.2, A.8*

The Matrix Formulation: One Column per Page



M =

from page …

	v	**w**	**x**
to v	0	1/2	1
to w	1/2	1/2	0
to x	1/2	0	0

column w: how w splits its rank

every column sums to 1

- $M_{v,w} = 1/o_w$ if w links to v, else 0: each column sums to 1 (**column stochastic**).
- With $\mathbf{r} = (r_v, r_w, r_x)^T$, the three flow equations become **one**: $\mathbf{M}\,\mathbf{r} = \mathbf{r}$.

(Assumption for now: every page has at least one out-link. We drop it in §5.)

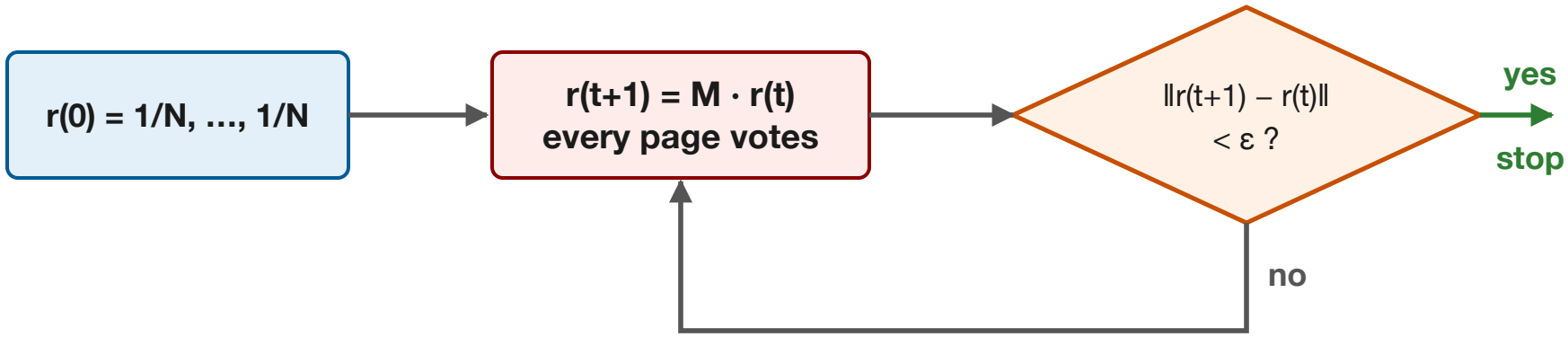
The Eigenvector Formulation

$$\mathbf{M}\,\mathbf{r} = \mathbf{r}$$

- $\mathbf{r}$ is an **eigenvector** of $\mathbf{M}$ with **eigenvalue 1**: the **principal eigenvector**, since 1 is the largest eigenvalue of any column-stochastic matrix.
- Solve it by **power iteration**: no equations, we just keep "voting".

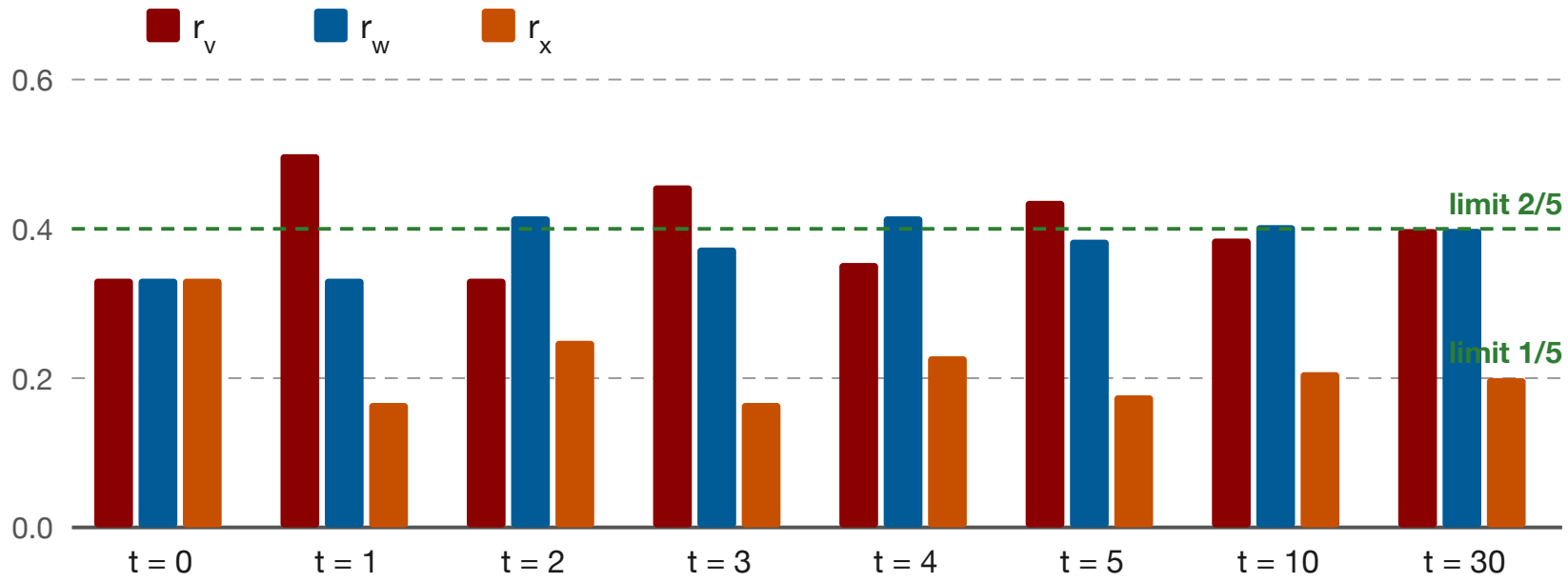
Does it always exist, and is it unique? Not always: see §4–§5.

Power Iteration



- Start with **equal** rank for everybody; at each step, every page **passes its rank along its out-links**.
- Stop when the vector no longer changes (up to ϵ).

Power Iteration on the Example



- First step: $r_v(1) = \frac{1}{2} \cdot \frac{1}{3} + 1 \cdot \frac{1}{3} = \frac{1}{2},\ r_w(1) = \frac{1}{3},\ r_x(1) = \frac{1}{6}$ (total preserved). Full table in Appendix A.8.

Same Answer, Without Solving Equations

	r_v	r_w	r_x
Flow equations + constraint (§2)	2/5	2/5	1/5
Power iteration (limit)	6/15 = 2/5	6/15 = 2/5	3/15 = 1/5

- Each iteration touches each link **once**: cheap, even for huge graphs.
- *Does it always converge? Always to the same vector?* (§4–§5)

Take-Home Message of Part I

- **Goal:** an importance score for each page, from links only.
- Represent the Web as a matrix $\mathbf{M}$: a link from w to v is a **vote** from w to v.
- PageRank is the **principal eigenvector** of $\mathbf{M}$: $\mathbf{Mr} = \mathbf{r}$.
- **Power iteration** finds it by repeated multiplication.
- Next: the **probabilistic** view, and what can go wrong.

✅ Checkpoint 3 — Matrix, Eigenvector, Power Iteration

Test your understanding:

1. Why does every column of $\mathbf{M}$ sum to 1?
2. Write $\mathbf{M}$ for a graph with edges $a \to b$, $a \to c$, $b \to c$, $c \to a$.
3. Why is $\mathbf{Mr} = \mathbf{r}$ an eigenvector equation, and for which eigenvalue?
4. Why do we start power iteration from the uniform vector, and when do we stop?

Consolidate your answers before proceeding.

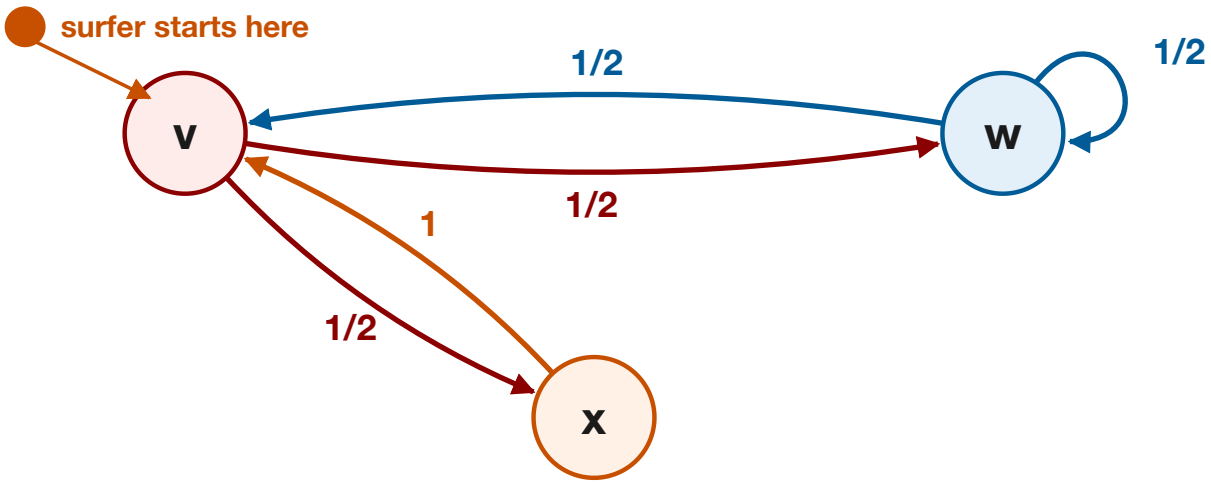
§ 4 — The Random Surfer

The probabilistic view

📎 *Extended material → Appendix A.1, A.5, A.6*

The Random Surfer

Imagine a person browsing the Web **forever**: pick **one outgoing link uniformly at random**, follow it, repeat.



*A **random walk** on the Web graph; the labels on the arrows are now **probabilities**.*

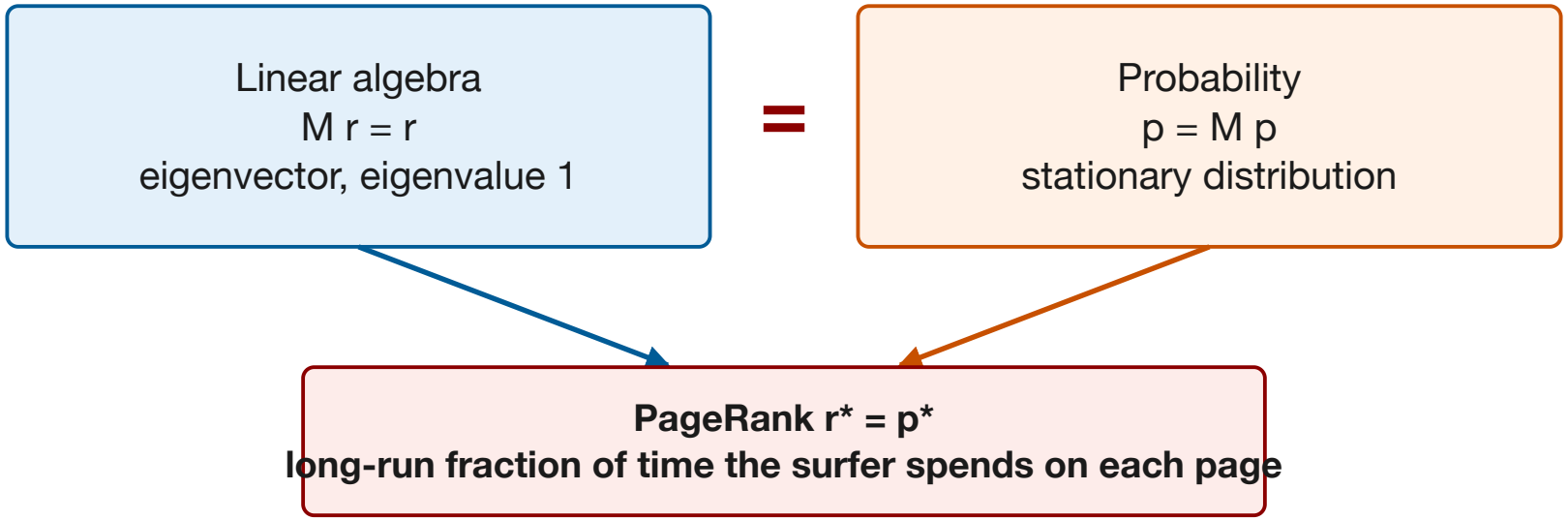
The Transition Matrix Is $\mathbf{M}$

- $M_{v,w}$ = probability that the surfer **moves from** w **to** v: exactly $1/o_w$ if w links to v.
- A **Markov chain**: the next page depends **only on the current page**.
- With $\mathbf{p}(t)$ the distribution over pages at time t:

$$\mathbf{p}(t+1) = \mathbf{M}\,\mathbf{p}(t)$$

Same formula as power iteration: starting uniform, $\mathbf{p}(1) = (1/2, 1/3, 1/6) = \mathbf{r}(1)$.

Two Views, One Vector



- If the walk reaches a **steady state**, $\mathbf{p}^* = \mathbf{M}\mathbf{p}^*$: the **stationary distribution**.
- A page visited often is a page that matters.

When Does It Work?

Does power iteration always converge? Is $\mathbf{r}^*$ unique? Does the walk reach a steady state?

> ***Perron–Frobenius (informal).*** *If $\mathbf{M}$ is column stochastic* ***and strictly positive*** *(no zero entries), then $\mathbf{r}^*$* ***exists****, is* ***unique****, and power iteration converges to it* ***from any starting vector****. (Formal statement: Appendix A.1.)*

- **Problem:** our $\mathbf{M}$ is stochastic, but it usually contains **zeros** (most pairs of pages are not linked).

Should we give up? No: this is where Brin and Page fixed the definition.

✅ Checkpoint 4 — The Random Surfer

Test your understanding:

1. What does $M_{v,w}$ mean in the random-surfer model?
2. What is the Markov property, in one sentence?
3. Why is the PageRank vector the same as the stationary distribution?
4. Which assumption of the Perron–Frobenius theorem does our $\mathbf{M}$ violate?

Consolidate your answers before proceeding.

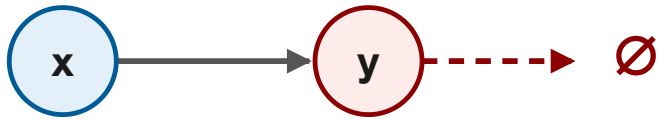
§ 5 — Google's Fix

Dead ends, spider traps, teleporting

📎 *Extended material → Appendix A.1, A.2, A.8*

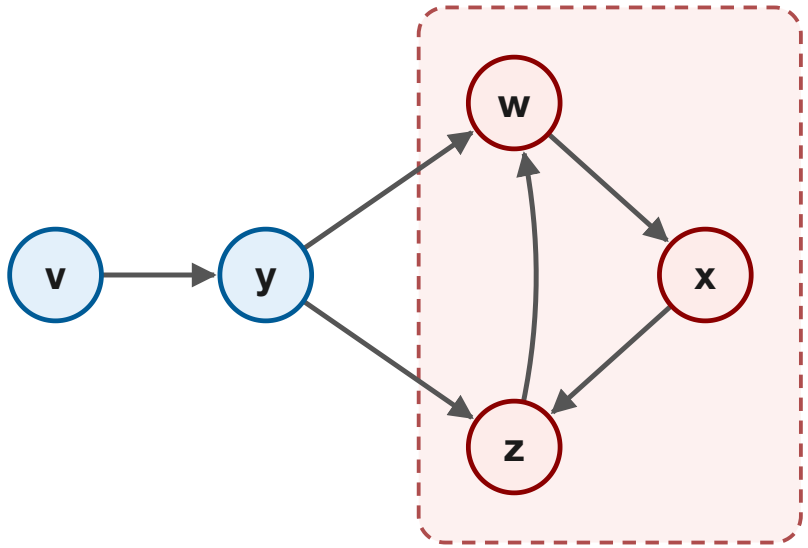
Two Problems with the Web Graph

Dead end



no out-links: rank leaks out

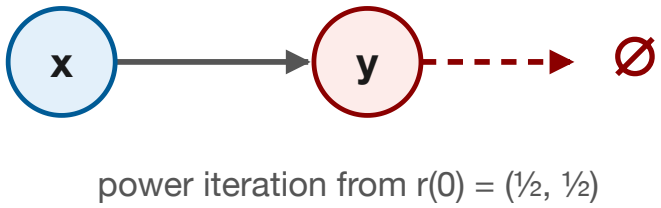
Spider trap



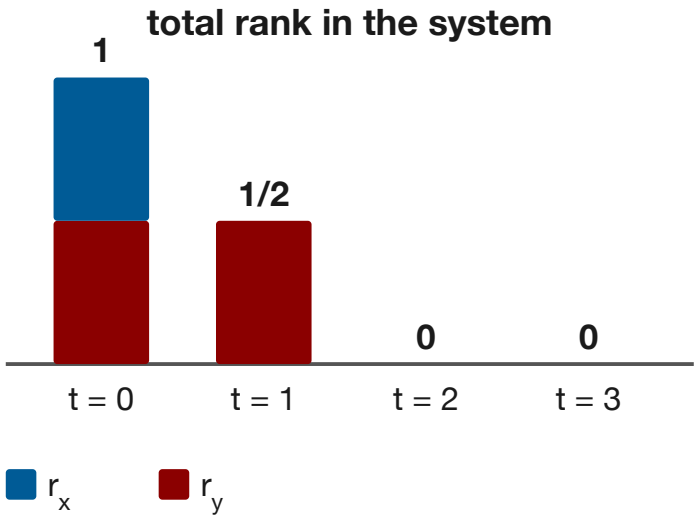
links stay inside: rank gets absorbed

- **Dead end:** $\mathbf{M}$ is **not stochastic**. **Spider trap:** $\mathbf{M}$ is stochastic but **not positive**.

Problem 1: A Dead End Swallows the Rank

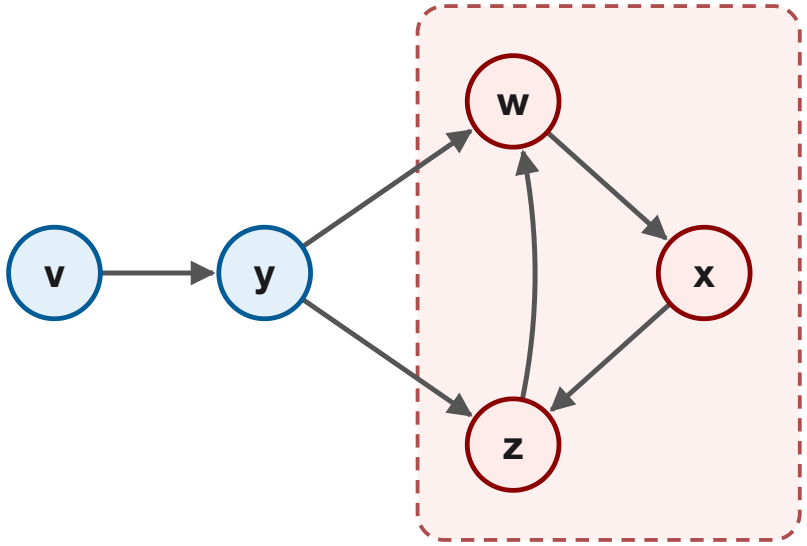


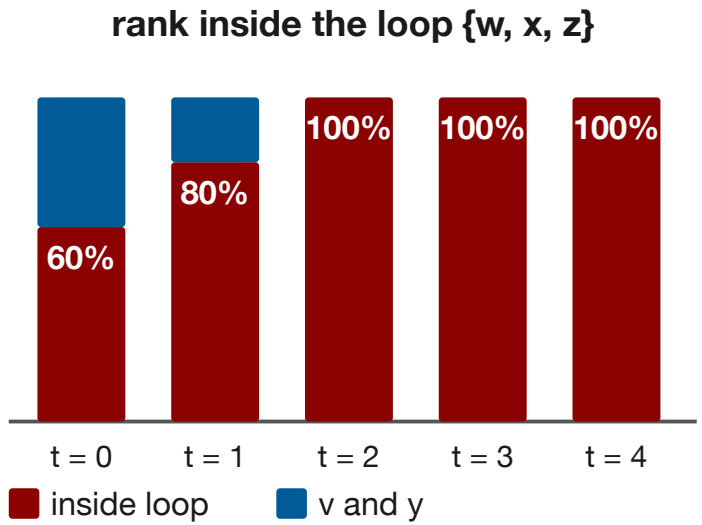
power iteration from r(0) = (½, ½)



- Column y of $\mathbf{M}$ is all zeros: all rank **leaks out** and the PageRank vector **vanishes to 0** (trace in Appendix A.8).

Problem 2: A Spider Trap Absorbs the Rank

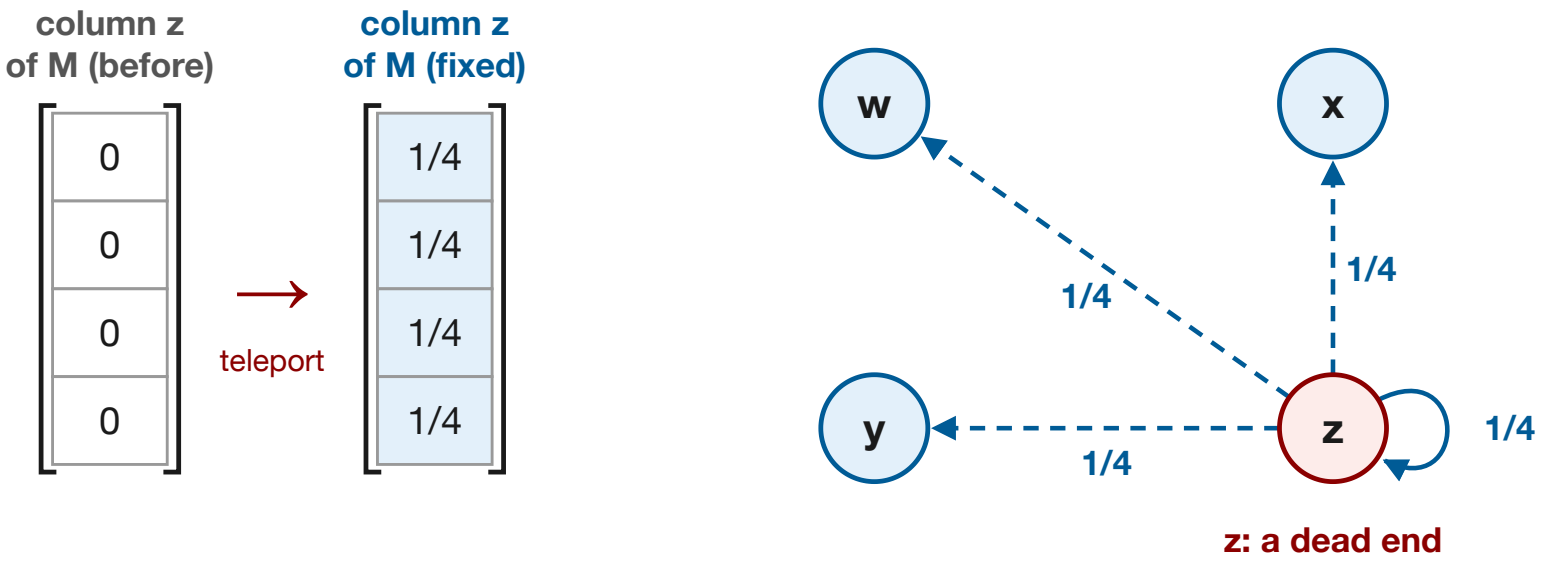




- Rank can **enter** the loop but never leave: after two steps v and y hold nothing, and the vector keeps rotating inside the loop (trace in Appendix A.8).

Fix 1: Teleporting from Dead Ends

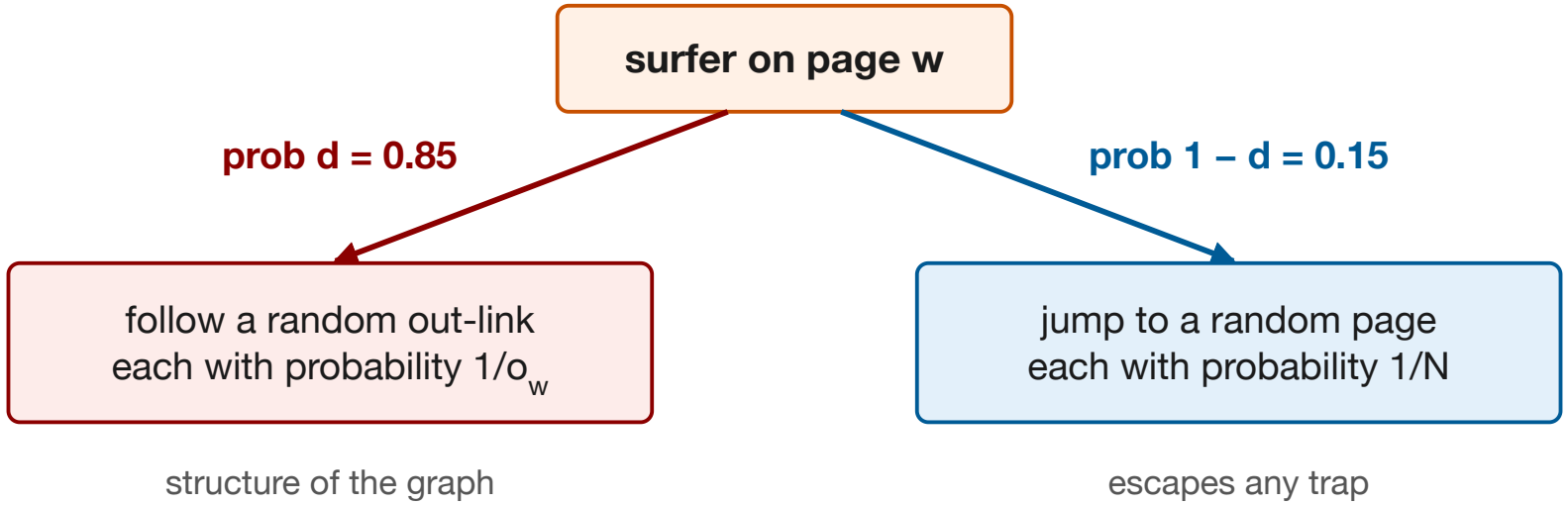
Pretend a dead end links to **every** page, with equal probability $1/N$:



- The dead end's zero column becomes a column of $1/N$: $\mathbf{M}$ **is column stochastic again**.
- From now on, $\mathbf{M}$ always includes this fix.

Fix 2: Teleporting from Everywhere

To escape spider traps, the surfer **sometimes jumps**; d is the **damping factor** (Brin and Page: $d = 0.85$).



The surfer can no longer be trapped forever: at every step it can leave.

The Google Matrix

$$\mathbf{G} = d\,\mathbf{M} + \frac{1-d}{N}\,\mathbf{1}\mathbf{1}^T$$

d · M

0	0.425	0.85
0.425	0.425	0
0.425	0	0

the links: mostly zeros

+

(1–d)/N · 1 1ᵀ

0.05	0.05	0.05
0.05	0.05	0.05
0.05	0.05	0.05

teleport: the same everywhere

=

G

0.05	0.475	0.9
0.475	0.475	0.05
0.475	0.05	0.05

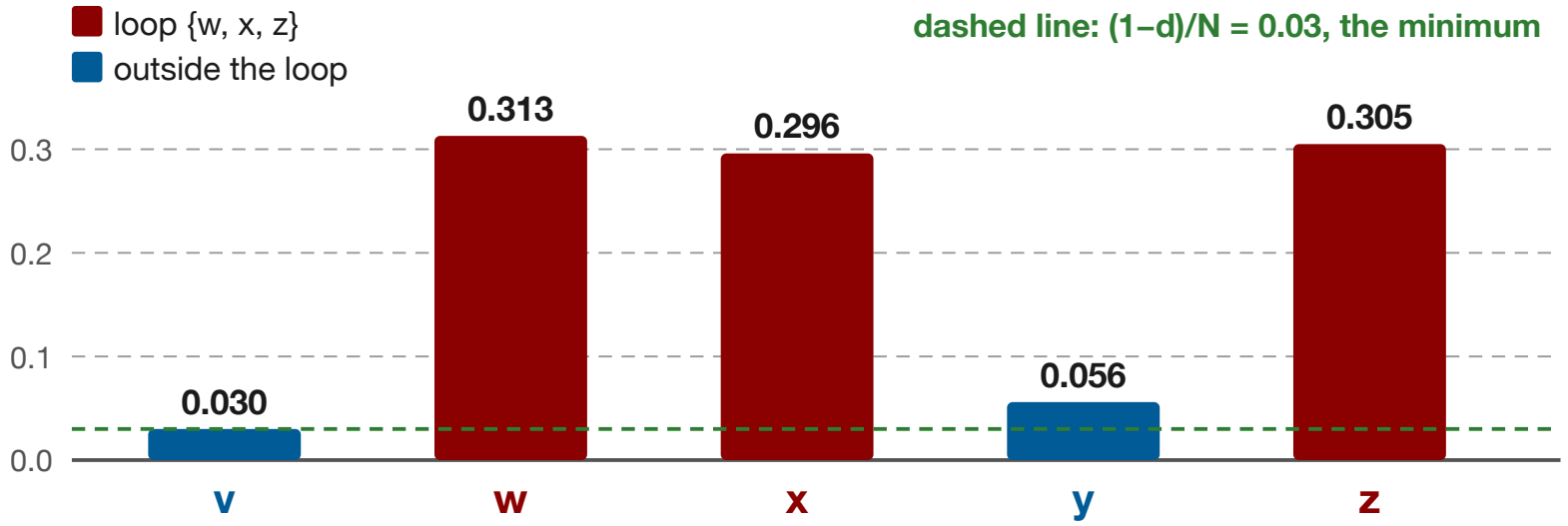
every entry > 0

three-page example, d = 0.85, N = 3

- $d\,\mathbf{M}$ is the surfer **following links**; the second term is the **jump**. $\mathbf{G}$ is column stochastic and **strictly positive**: **Perron–Frobenius applies**. PageRank is the stationary distribution of $\mathbf{G}$.

Back to the Spider Trap

$\mathbf{G}$ for the 5-page example ($d = 0.85$, $N = 5$):



*Nobody is at 0; the loop still holds most of the rank but **no longer swallows everything**. The vector **converges**.*

✅ Checkpoint 5 — Google's Fix

Test your understanding:

1. What is a dead end, and what happens to the rank vector without a fix?
2. What is a spider trap, and why does it break uniqueness and convergence?
3. Explain the two kinds of teleporting and which problem each solves.
4. Why does $\mathbf{G}$ satisfy the hypotheses of Perron–Frobenius?

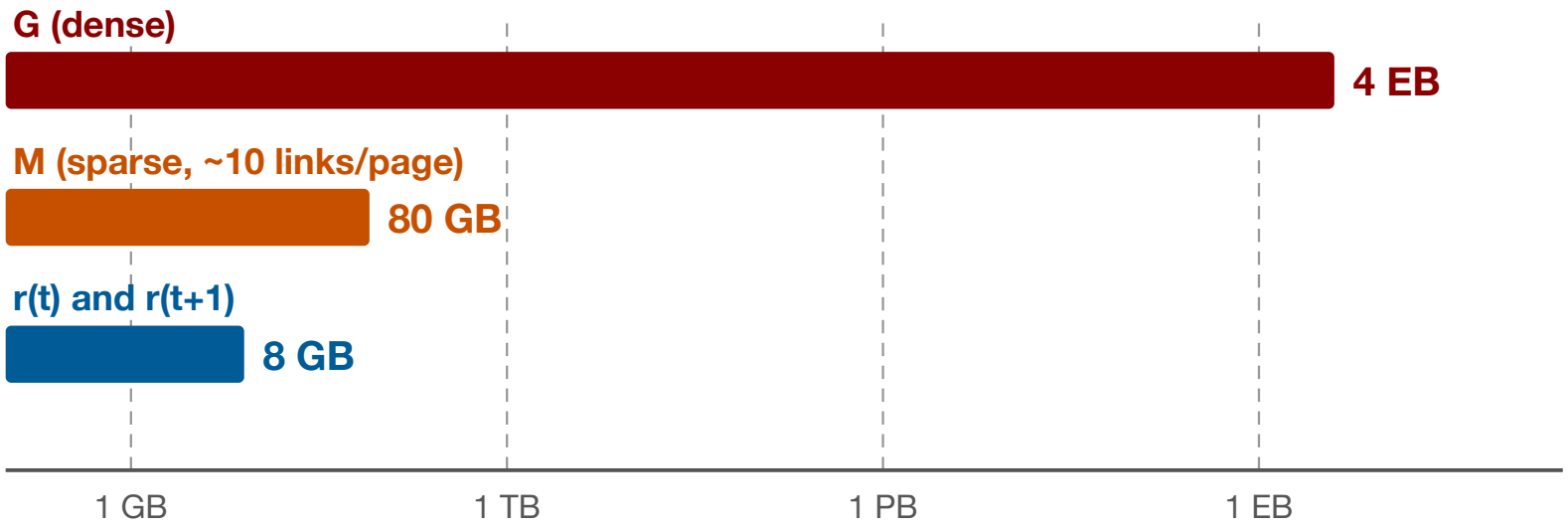
Consolidate your answers before proceeding.

§ 6 — Computing PageRank at Scale

Keep the matrix sparse

📎 *Extended material → Appendix A.3, A.4, A.5, A.6, A.8*

We Can Never Build $\mathbf{G}$



N = 10^9 pages, 4 bytes per entry. Log scale: each tick is ×1000.

$\mathbf{G}$ *(and the* $1/N$ *dead-end columns of* $\mathbf{M}$*) are* ***conceptual****: we never build them. We store only the real links, the sparse* $\mathbf{M}_{\text{links}}$*, and add the rest back as a single constant.*

The Update We Actually Run

$$\mathbf{r}(t+1) \;=\; d\,\mathbf{M}_{\text{links}}\,\mathbf{r}(t) \;+\; \frac{1-S}{N}\,\mathbf{1}, \qquad S = \text{total of the first term}$$

- $d\,\mathbf{M}_{\text{links}}\,\mathbf{r}(t)$: only the **links** are touched (the matrix stays sparse).
- The constant $(1-S)/N$ re-inserts **both** teleporting and the rank lost in dead ends.
- Same result as multiplying by $\mathbf{G}$ (derivation in Appendix A.8).

PageRank: Pseudocode

Algorithm 1 PageRank by sparse power iteration

Input: out-links O_w for every page w; damping d; tolerance ε **Output:** rank vector $\mathbf{r}$

$\mathbf{r} \leftarrow (1/N, \ldots, 1/N)$

repeat

  $\mathbf{r}' \leftarrow \mathbf{0}$

  for each page w **do** — *one pass over the links*

    for each $v \in O_w$ **do** $r'_v \leftarrow r'_v + d\, r_w / o_w$

  $S \leftarrow \sum_v r'_v$ — *rank that stayed on the links*

  for each page v **do** $r'_v \leftarrow r'_v + (1 - S)/N$ — *teleport + dead-end rank*

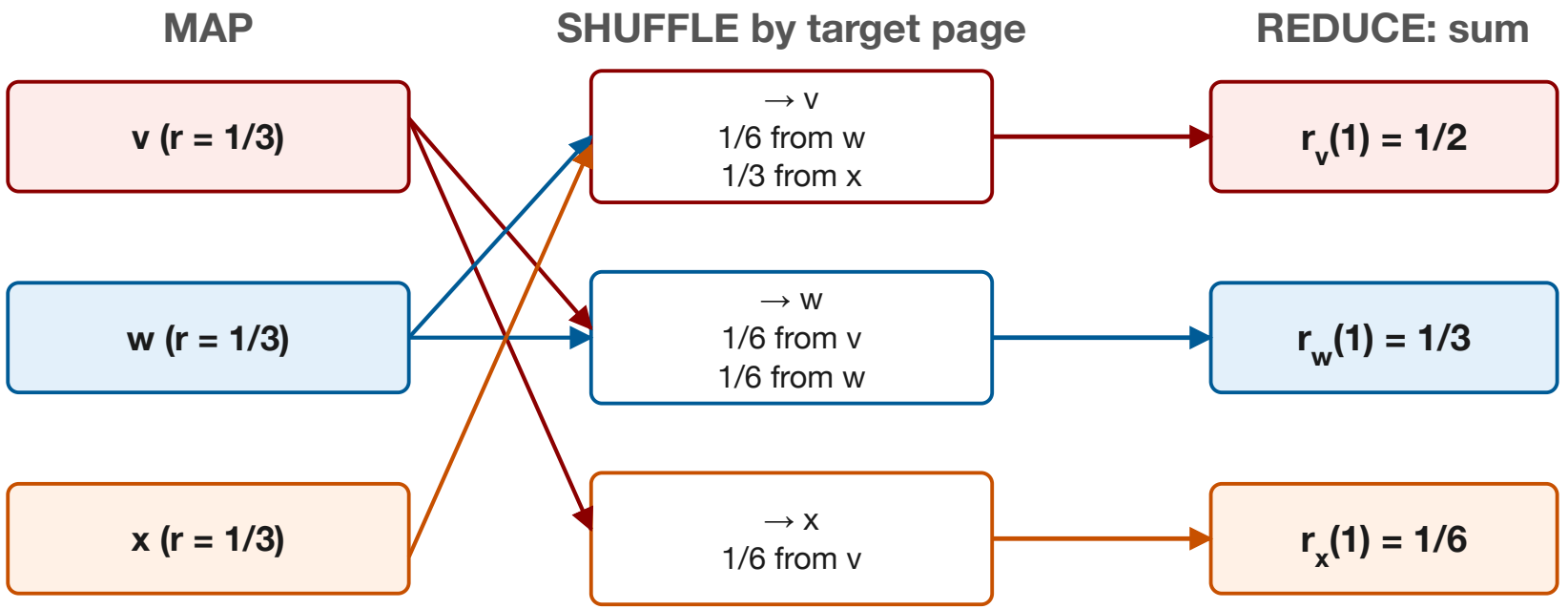
  $\delta \leftarrow \|\mathbf{r}' - \mathbf{r}\|_1$; $\mathbf{r} \leftarrow \mathbf{r}'$

until $\delta < \varepsilon$

return $\mathbf{r}$

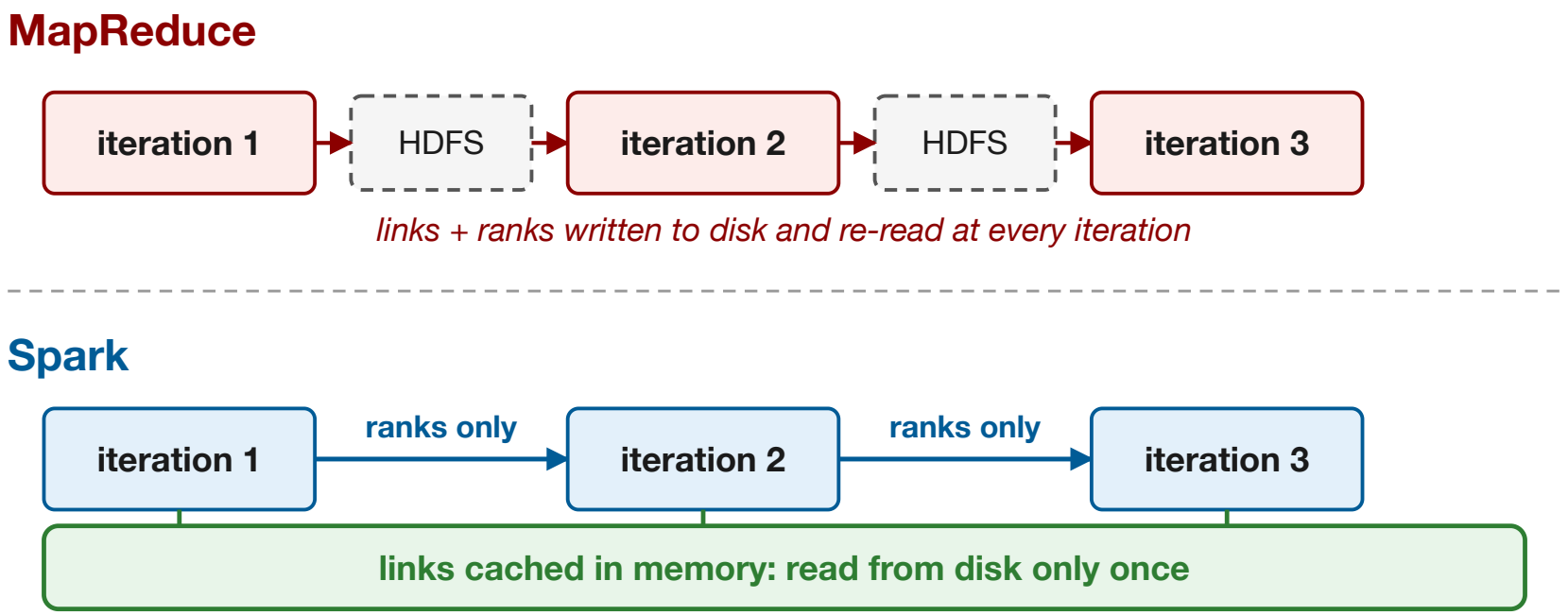
Verified on the 4-page dead-end example: converges to $(0.206,\ 0.186,\ 0.186,\ 0.422)$.

One Iteration at Scale



- One iteration = a **sparse matrix–vector product**: **map**, **shuffle** by target page, **reduce**. About **90 iterations** give $\epsilon = 10^{-6}$ (Appendix A.2).
- Hubs (Lecture 2.4) make some reducers much busier than others.

MapReduce vs Spark



MapReduce (Lecture 1.2) re-reads the graph at every iteration; Spark (Lecture 1.3) caches the links and iterates on the ranks.

✅ Checkpoint 6 — Computing PageRank at Scale

Test your understanding:

1. Why can't we store $\mathbf{G}$, and what do we store instead?
2. What does the constant $(1 - S)/N$ account for in the pseudocode?
3. Which two operations make up one iteration in MapReduce or Spark?
4. Why is Spark better suited than MapReduce for this computation?

Consolidate your answers before the synthesis.

Take-Home Message of Part II

- Existence and uniqueness of PageRank hold only for **strictly positive stochastic** matrices (Perron–Frobenius).
- The Web has **dead ends** and **spider traps**.
- **Google's solution:** teleporting, with damping factor $d = 0.85$.
- It is still computed efficiently from the **sparse** link matrix, by power iteration.
- PageRank is the **offline authority prior** $P(d)$ of Lecture 3.1's blended score.

Lecture 3.2 Complete

Recommended Reading

- Leskovec, Rajaraman & Ullman (2020). *Mining of Massive Datasets*, Ch. 5.
- Brin & Page (1998). *The anatomy of a large-scale hypertextual Web search engine*. Computer Networks 30(1–7).
- Page, Brin, Motwani & Winograd (1999). *The PageRank citation ranking: bringing order to the web*. Stanford InfoLab Technical Report.

Contact & Q&A

- **Course:** Big Data Computing
- **Office Hours:** Drop me a message, and we'll arrange a time to meet, either in person or online! 🙂

Thank you for your attention!

§ Appendix

Appendix Map

Section	Extends	What you will find	Slides
A.1	§ 2–§ 5	Formal view of $\mathbf{M}$; statement of Perron–Frobenius	2
A.2	§ 3, § 5, § 6	Convergence rate d^t with a short proof; iteration counts	2
A.3	§ 6	MapReduce version	1
A.4	§ 6	PySpark version	1
A.5	§ 4, § 6	Personalized PageRank	1
A.6	§ 4, § 6	Monte Carlo PageRank	1
A.7	§ 1	HITS: hubs and authorities	1
A.8	§ 2, § 3, § 5, § 6	Derivations and numerical traces moved out of the main lecture	5
A.9	all	Exam-style problems and solutions	3

Supplementary material for deeper study. Not required during the main lecture.

A.1 · Formal View of $\mathbf{M}$

Let $\mathbf{A}$ be the adjacency matrix with $A_{v,w} = 1$ if $w \in O_v$ (row v lists the out-links of v), and let $\mathbf{L} = \mathrm{diag}(o_1, \ldots, o_N)$ be the diagonal matrix of out-degrees. Then:

$$\mathbf{M} \;=\; \left(\mathbf{L}^{-1}\mathbf{A}\right)^{T}, \qquad M_{v,w} = \begin{cases} 1/o_w & \text{if } v \in O_w \\ 0 & \text{otherwise} \end{cases}$$

- $\mathbf{L}^{-1}\mathbf{A}$ divides each row of $\mathbf{A}$ by its out-degree: **row** stochastic.
- Transposing makes it **column** stochastic, so that $\mathbf{r}(t+1) = \mathbf{M}\mathbf{r}(t)$ multiplies on the right.
- $\mathbf{L}^{-1}$ exists only if every $o_v \geq 1$: again, no dead ends.

A.1 · Perron–Frobenius, Stated

> ***Theorem.*** *Let $\mathbf{G}$ be a column-stochastic matrix with **all entries strictly positive**. Then (i) $\lambda = 1$ is an eigenvalue and every other eigenvalue has $|\lambda| < 1$; (ii) the eigenspace of $\lambda = 1$ has dimension one; (iii) it contains a unique vector $\mathbf{r}^*$ with positive entries summing to 1; (iv) for every probability vector $\mathbf{r}(0)$: $\mathbf{G}^t\mathbf{r}(0) \to \mathbf{r}^*$.*

- (i)–(iii): **existence** and **uniqueness**. (iv): **convergence** of power iteration.
- Why positivity matters: it makes every page **reachable from every page in one step** (irreducible) and rules out **cycles that force periodic behavior** (aperiodic).
- Proof: see Horn & Johnson, *Matrix Analysis*, or Leskovec et al., *MMDS*, Ch. 5.

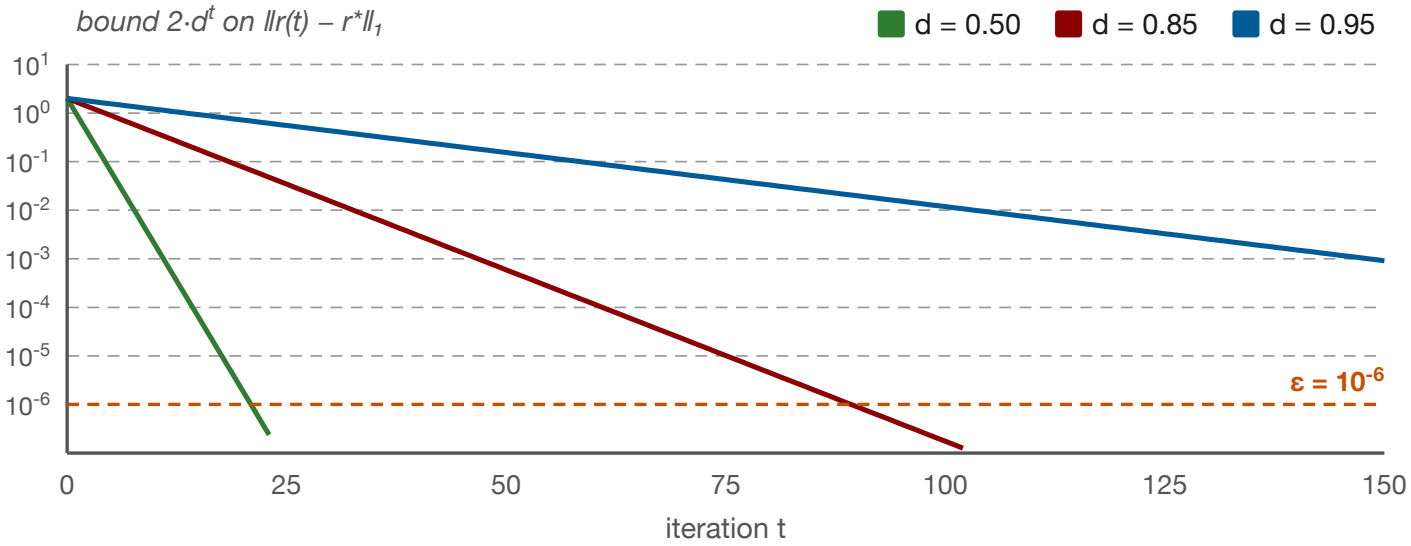
A.2 · Convergence Rate of Power Iteration

> ***Claim.*** *With* $\mathbf{r}(t+1) = \mathbf{G}\,\mathbf{r}(t)$ *and* $\mathbf{G} = d\mathbf{M} + \frac{1-d}{N}\mathbf{1}\mathbf{1}^T$*:* $\|\mathbf{r}(t) - \mathbf{r}^*\|_1 \le 2\,d^{\,t}$.

Proof. Let $\mathbf{e}(t) = \mathbf{r}(t) - \mathbf{r}^*$. Both vectors sum to 1, so $\mathbf{1}^T\mathbf{e}(t) = 0$.

1. Hence $\mathbf{G}\,\mathbf{e}(t) = d\,\mathbf{M}\mathbf{e}(t)$: the all-ones term vanishes.
2. A column-stochastic matrix never increases the 1-norm: $\|\mathbf{M}\mathbf{e}\|_1 \le \|\mathbf{e}\|_1$.
3. So $\|\mathbf{e}(t+1)\|_1 \le d\,\|\mathbf{e}(t)\|_1$, and $\|\mathbf{e}(t)\|_1 \le d^t\,\|\mathbf{e}(0)\|_1 \le 2\,d^t$. ■

A.2 · How Many Iterations?



- Guarantee $2d^t \le 10^{-6}$ for $d = 0.85$: $t \ge \ln(5 \times 10^{-7}) / \ln 0.85 \approx 89.3$, so **90 iterations**; for $d = 0.99$ about **1444**.
- The rate does **not** depend on N: the Web graph is just as fast as a toy graph.

A.3 · MapReduce Version (One Iteration)

```
Input record:  ( w , r_w , O_w )

Map( w , r_w , O_w ):
    emit( w , O_w )                              # pass the graph structure through
    if O_w is empty: emit( "DEAD" , r_w )        # dead-end rank (handled by the driver)
    for each v in O_w:
        emit( v , d · r_w / o_w )                # the share of w for v

Reduce( v , values ):
    O_v ← the list among the values;   s ← sum of the numeric values
    emit( v , s , O_v )                          # s = d · (rank received on links)

Driver: S ← sum of all s;  add (1 − S)/N to every rank;  test convergence;  next job.
```

*The whole link graph is shuffled and written to HDFS at **every** iteration: expensive (Lecture 1.2).*

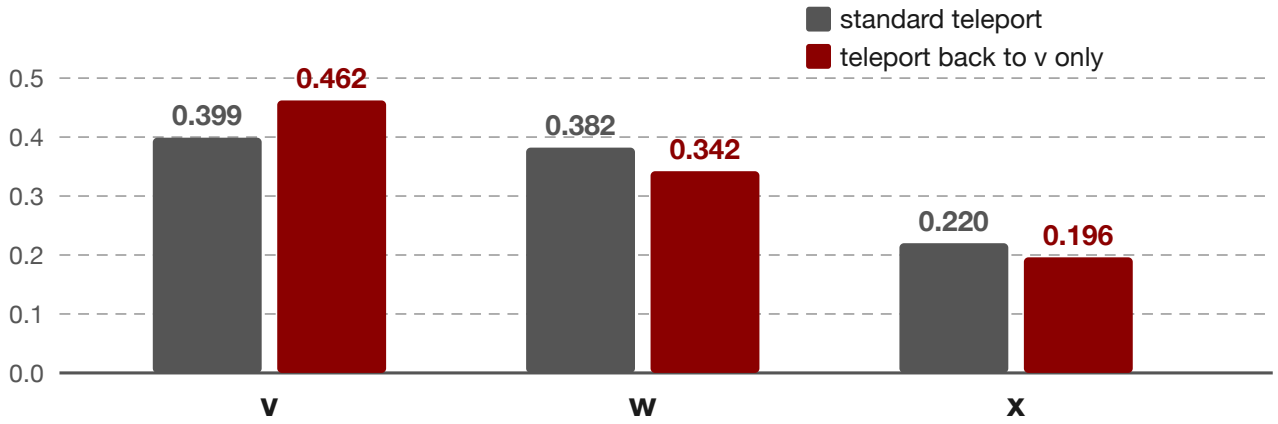
A.4 · PySpark Version

```
links = sc.textFile(path).map(parse).cache()        # (page, [out-links]); ALL pages present
N = links.count()
ranks = links.mapValues(lambda _: 1.0 / N)
for _ in range(iters):
    contribs = links.join(ranks).flatMap(           # w sends r_w / o_w to each v in O_w
        lambda kv: [(v, kv[1][1] / len(kv[1][0])) for v in kv[1][0]])
    summed = contribs.reduceByKey(lambda a, b: a + b)
    total = d * summed.values().sum()               # rank that stayed on the links
    ranks = links.leftOuterJoin(summed).mapValues(  # pages with no in-links get only teleport
        lambda p: d * (p[1] or 0.0) + (1.0 - total) / N)
```

`links` *is cached: it is read from disk once, then reused at every iteration (Lecture 1.3). Same update as the pseudocode of §6; checked on the 4-page dead-end graph.*

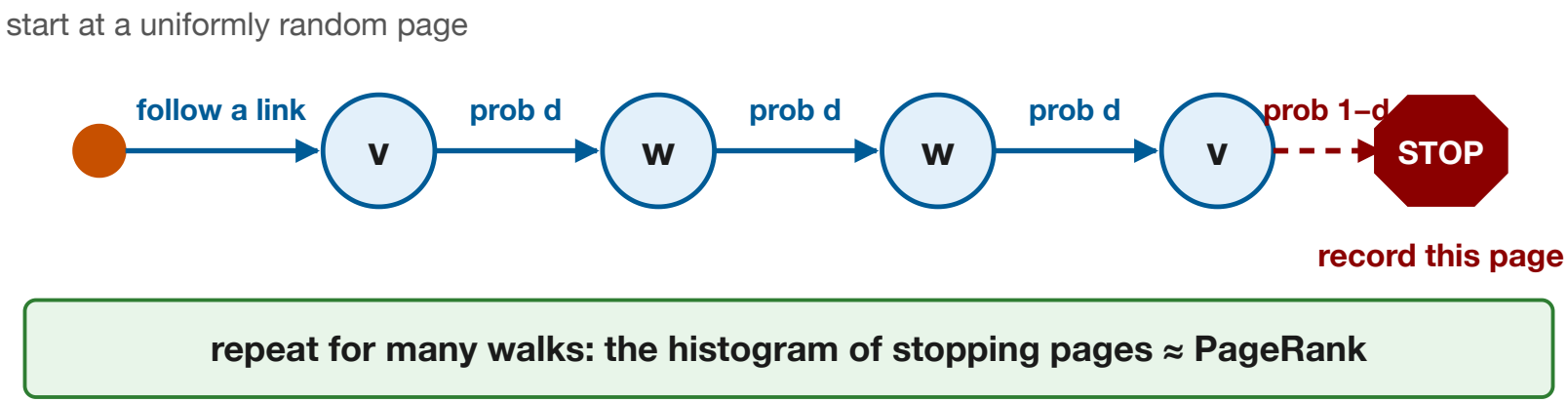
A.5 · Personalized PageRank

Teleport not to a uniform page, but back to a **chosen page** s (or a set of pages): $\mathbf{r}_s = d\,\mathbf{M}\,\mathbf{r}_s + (1-d)\,\mathbf{e}_s$



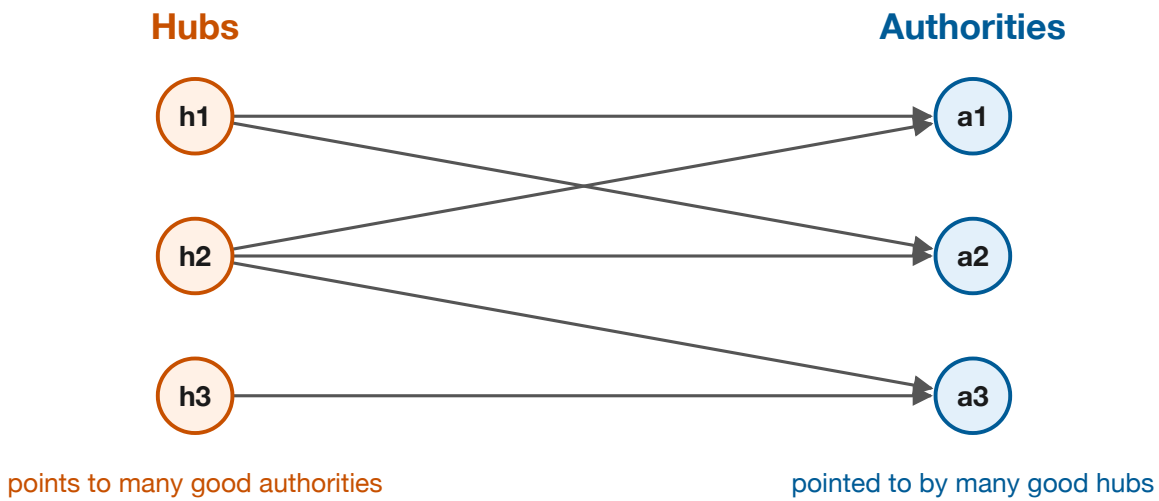
- $\mathbf{e}_s$: all zeros except a 1 at s. Importance **from the point of view of** s; uses: recommendations, "pages related to this one", topic-sensitive ranking.

A.6 · Monte Carlo PageRank



- Running example, 20,000 walks: $(0.397,\ 0.384,\ 0.219)$ versus exact $(0.399,\ 0.382,\ 0.220)$.
- Walks are independent: **embarrassingly parallel**; error shrinks as $1/\sqrt{\#\text{walks}}$ (Lecture 3.4).

A.7 · HITS: Hubs and Authorities



- Iterate: $\mathbf{a} \leftarrow \mathbf{A}^T\mathbf{h}$, $\mathbf{h} \leftarrow \mathbf{A}\mathbf{a}$, normalizing each time.
- HITS is computed **per query** on a small subgraph; PageRank is **global** and query-independent.

Reference: Kleinberg (1999), "Authoritative sources in a hyperlinked environment", Journal of the ACM 46(5).

A.8 · Why the Flow Equations Fix Only Proportions

(moved from § 2) 3 equations, 3 unknowns — but the equations are **not independent**.

- Substitute $r_x = r_v/2$ in the first equation: $r_v = r_w/2 + r_v/2$, i.e. $r_v = r_w$, which is exactly the second equation.
- So there are **infinitely many** solutions, all multiples of each other: the recursion fixes only the **proportions** between ranks.
- The constraint $r_v + r_w + r_x = 1$ pins down the scale: $r_v = r_w = 2/5$, $r_x = 1/5$.
- Check: $r_x = r_v/2 = 1/5$ ✓ and $r_v = r_w/2 + r_x = 1/5 + 1/5 = 2/5$ ✓.

A.8 · Power Iteration: The Full Trace

(moved from § 3)

t	0	1	2	3	4	5	10	30
r_v	1/3	1/2	1/3	11/24	17/48	7/16	0.387	0.3998
r_w	1/3	1/3	5/12	3/8	5/12	37/96	0.405	0.4001
r_x	1/3	1/6	1/4	1/6	11/48	17/96	0.208	0.2001

- First step, in full: $r_v(1) = \frac{1}{2}\cdot\frac{1}{3} + 1\cdot\frac{1}{3} = \frac{1}{2}$, $r_w(1) = \frac{1}{2}\cdot\frac{1}{3} + \frac{1}{2}\cdot\frac{1}{3} = \frac{1}{3}$, $r_x(1) = \frac{1}{2}\cdot\frac{1}{3} = \frac{1}{6}$; check $\frac{1}{2} + \frac{1}{3} + \frac{1}{6} = 1$.
- The vector **zigzags** around the limit $(0.4, 0.4, 0.2) = (6/15, 6/15, 3/15)$ and settles down.

A.8 · Dead End and Spider Trap: Numerical Traces

(moved from § 5)

Dead end $x \to y$, $\mathbf{M} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$, start $(1/2, 1/2)$:

t	0	1	2	3
r_x	1/2	0	0	0
r_y	1/2	1/2	0	0
total	**1**	**1/2**	**0**	**0**

Spider trap $v \to y$, $y \to w, z$, $w \to x$, $x \to z$, $z \to w$; order (v, w, x, y, z):

t	0	1	2	3	4
r_v, r_y	0.2, 0.2	0, 0.2	0, 0	0, 0	0, 0
r_w, r_x, r_z	0.2, 0.2, 0.2	0.3, 0.2, 0.3	0.4, 0.3, 0.3	0.3, 0.4, 0.3	0.3, 0.3, 0.4

A.8 · Why the Rearranged Update Equals $\mathbf{G}\,\mathbf{r}$

(moved from § 6) Let $\mathbf{r}$ sum to 1 and $S = \mathbf{1}^T(d\,\mathbf{M}_{\text{links}}\,\mathbf{r})$ be the rank that stayed on the links.

1. A dead-end column of $\mathbf{M}$ is $\mathbf{1}/N$ (the real links are in $\mathbf{M}_{\text{links}}$), so $\mathbf{M}\mathbf{r} = \mathbf{M}_{\text{links}}\mathbf{r} + \frac{\delta}{N}\mathbf{1}$ with δ the total rank sitting on dead ends.
2. Since $\mathbf{1}\mathbf{1}^T\mathbf{r} = \mathbf{1}$ (the entries of $\mathbf{r}$ sum to 1): $\mathbf{G}\,\mathbf{r} = d\,\mathbf{M}_{\text{links}}\,\mathbf{r} + \frac{d\,\delta + (1-d)}{N}\,\mathbf{1}$.
3. $\mathbf{G}\mathbf{r}$ sums to 1, so $d\,\delta + (1-d) = 1 - S$ and the constant is exactly $(1-S)/N$. ■

*Cost: about 10 out-links per page and $N = 10^9$ give 10^{10} entries, about **80 GB** for $\mathbf{M}_{\text{links}}$, against 4 EB for $\mathbf{G}$.*

A.8 · The Worked Example on $\mathbf{G}$

(moved from § 5) Three-page example, $d = 0.85$, $N = 3$ (no dead ends):

$$\mathbf{G} = \begin{pmatrix} 0.05 & 0.475 & 0.9 \\ 0.475 & 0.475 & 0.05 \\ 0.475 & 0.05 & 0.05 \end{pmatrix}, \qquad \mathbf{r}^* \approx (0.399,\, 0.382,\, 0.220)$$

- Spider-trap example ($N = 5$): $\mathbf{r}^* = (0.030,\, 0.313,\, 0.296,\, 0.056,\, 0.305)$; the minimum $(1 - d)/N = 0.03$ is reached by v, which has no in-links.

A.9 · Exam-Style Problems

P1. Graph $a \to b,\ a \to c,\ b \to c,\ c \to a$. Write $\mathbf{M}$ (order a, b, c), compute $\mathbf{r}(1)$, $\mathbf{r}(2)$ from the uniform start, and find $\mathbf{r}^*$.

P2. Graph $p \to q,\ p \to r,\ q \to r$ (r is a dead end), $d = 0.85$. Write $\mathbf{M}$ (with the dead-end fix) and $\mathbf{G}$, and compute $\mathbf{r}(1)$ from the uniform start.

P3. How many iterations guarantee $\|\mathbf{r}(t) - \mathbf{r}^*\|_1 \le 10^{-6}$ for $d = 0.85$? And for $d = 0.99$?

A.9 · Solutions (1/2)

P1. Columns: a sends $1/2$ to b and c; b sends 1 to c; c sends 1 to a.

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix}$$

$\mathbf{r}(1) = (1/3,\, 1/6,\, 1/2)$, $\mathbf{r}(2) = (1/2,\, 1/6,\, 1/3)$. Stationary: $r_a = r_c$, $r_b = r_a/2$, sum 1, so $\mathbf{r}^* = (2/5,\, 1/5,\, 2/5)$.

A.9 · Solutions (2/2)

P2. Dead end r: its column becomes $(1/3, 1/3, 1/3)$. With $(1-d)/N = 0.05$:

$$\mathbf{G} = \begin{pmatrix} 0.05 & 0.05 & 0.333 \\ 0.475 & 0.05 & 0.333 \\ 0.475 & 0.90 & 0.333 \end{pmatrix}, \qquad \mathbf{r}(1) = (0.144,\ 0.286,\ 0.569)$$

(limit: $(0.198,\ 0.282,\ 0.521)$).

P3. $2d^t \le 10^{-6} \Rightarrow t \ge \ln(5\times10^{-7})/\ln d$: **90** iterations for $d = 0.85$, **1444** for $d = 0.99$.