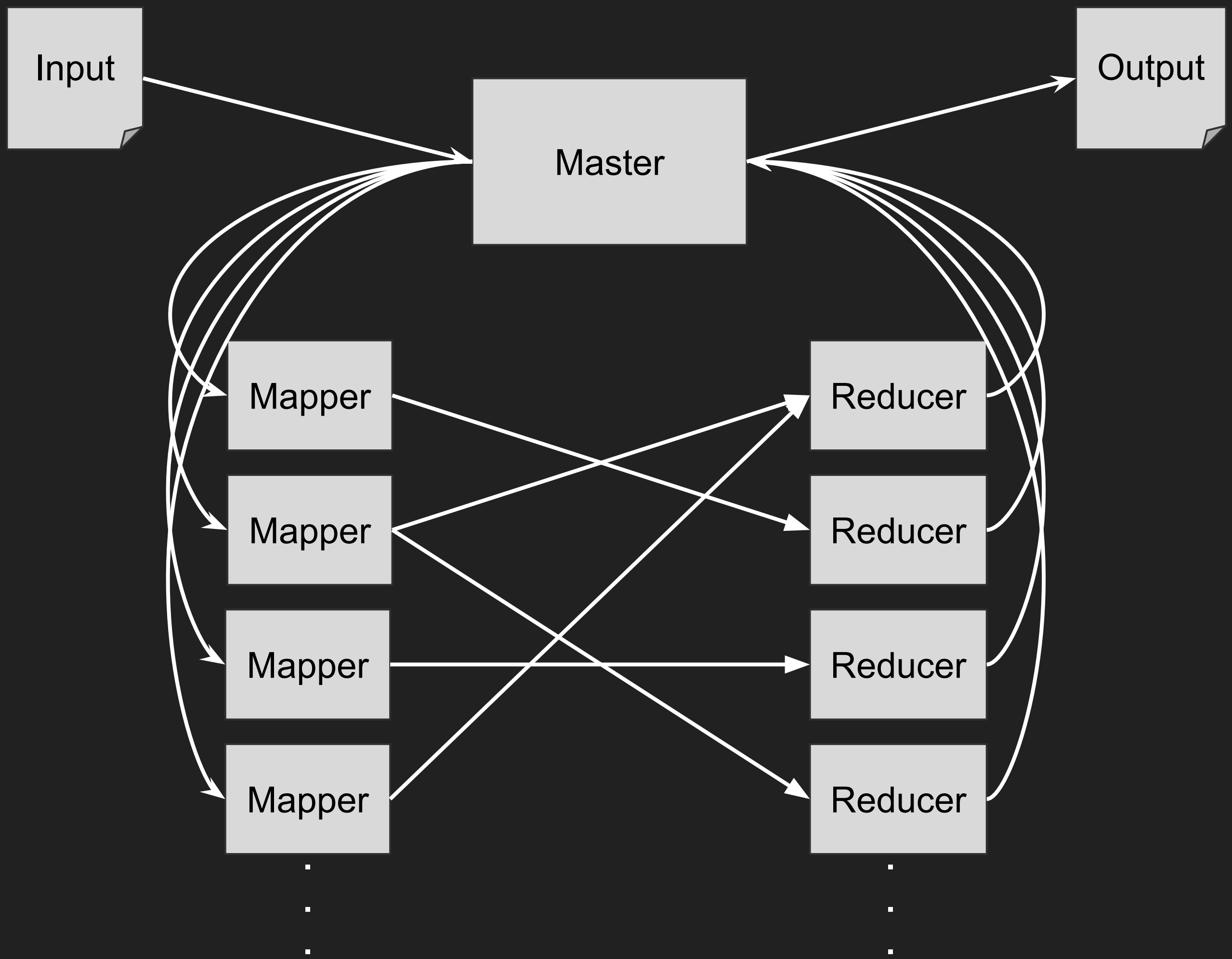
Distributed Computing

Map/Reduce

- Developed at Google, many other implementations since (Apache Hadoop, etc.)
- Designed for processing big data sets in parallel
- Framework provides
  - Parallelization
  - Redundancy
  - Fault-tolerance
  - Speed…?*
- Abstracts the split-apply-combine analysis strategy
- Works for a multi-threaded environment or a networked cluster environment

Map/Reduce

- Master/Controller node reads input and feeds (k_1, v_1) pairs to worker nodes (mappers)
- Map
  - Apply a user-provided map function on data keyed by k_1 and generates a list of outputs
  - $(k_1, v_1) \rightarrow [(k_2, v_2)]$
- Shuffle
  - Redistributes map outputs so that similarly key'd data are on the same shard
  - $hash(k_2)\ \%\ N_R$
  - $[(k_2, v_2)] \rightarrow (k_2, [v_2])$
- Reduce
  - Apply a user-provided reduce function to combine the list of v_2 values into a final output
  - $(k_2, [v_2]) \rightarrow [v_3]$
- Master/Controller node collates reducer outputs into a single final output


Input
Master
Output
Mapper
Mapper
Mapper
Mapper
Reducer
Reducer
Reducer
Reducer

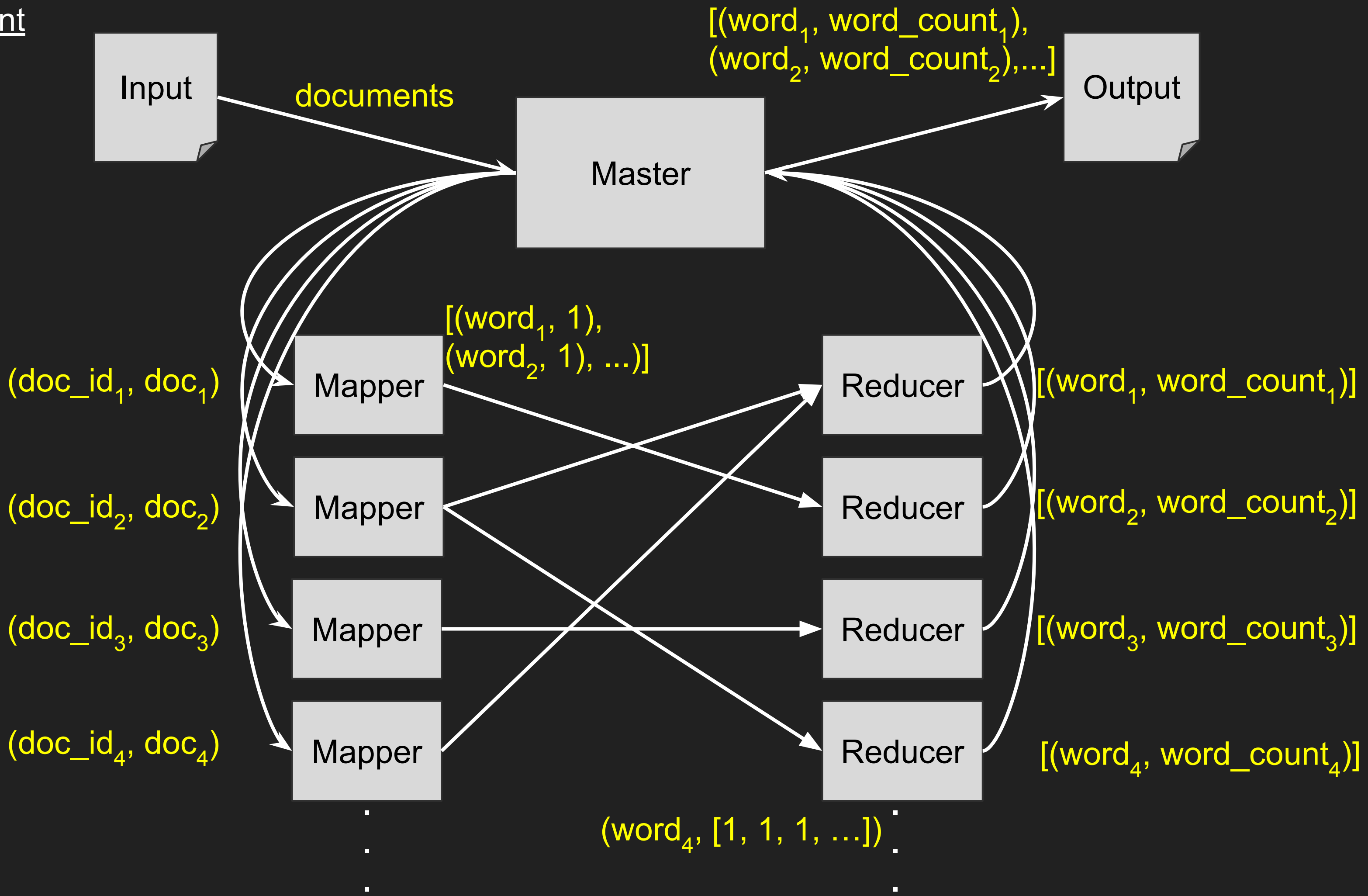
Map/Reduce

- Framework is essentially the same for every application
  - User only needs to supply the Map and Reduce functions
- What happens if a Mapper dies?
  - Controller keeps track of keys that have been successfully mapped
  - Can always send key to a new mapper
- What happens if Reducer dies?
  - Controller keeps track of the keys that have been successfully reduced
  - Mappers save their results on disk in the shard, so any other machine can start from there
- Not necessarily fast
  - “Shuffle” stage actually hides some complication about combining everything
  - Network/communication latency also a factor

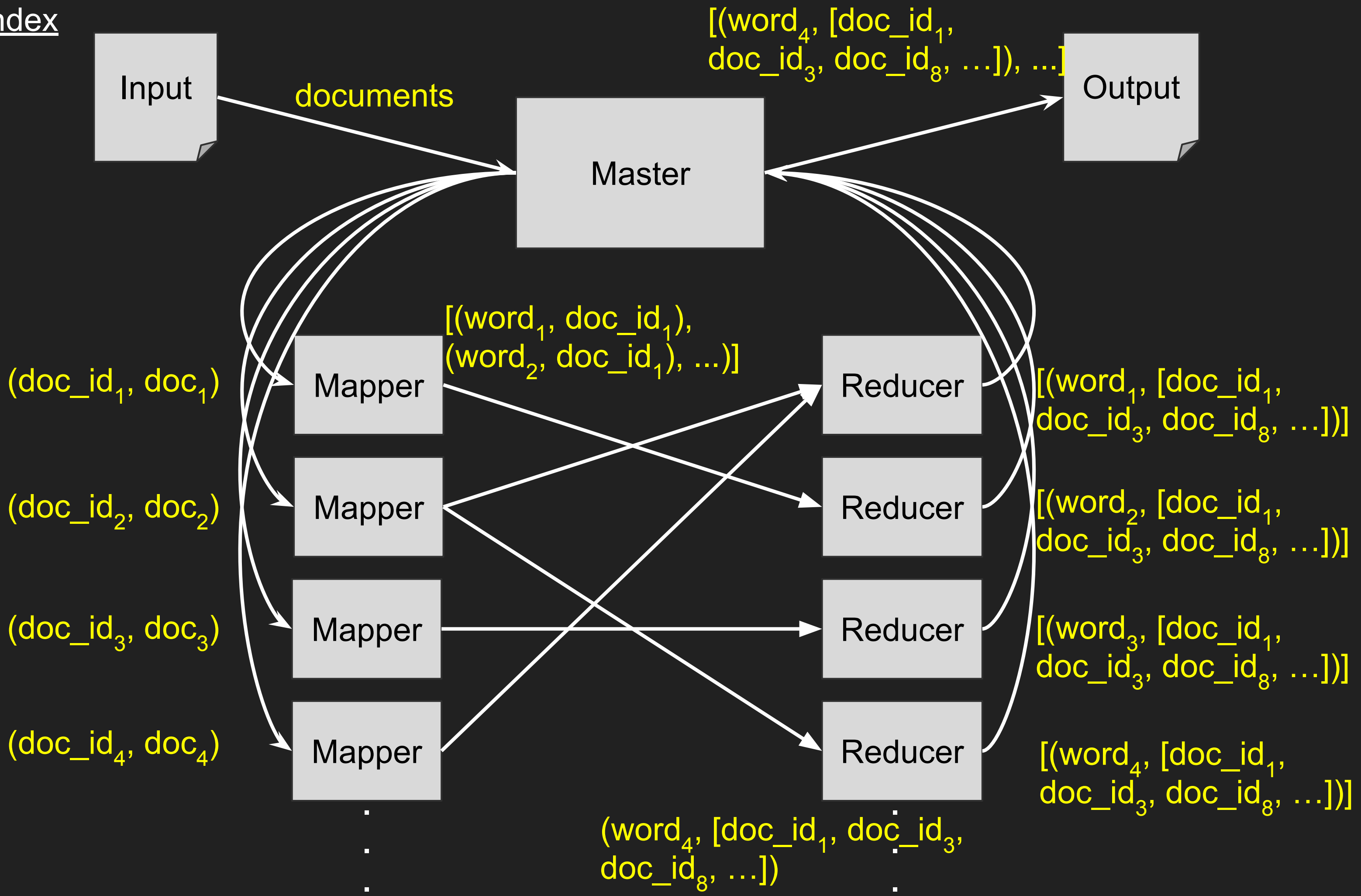
Map/Reduce Example Applications

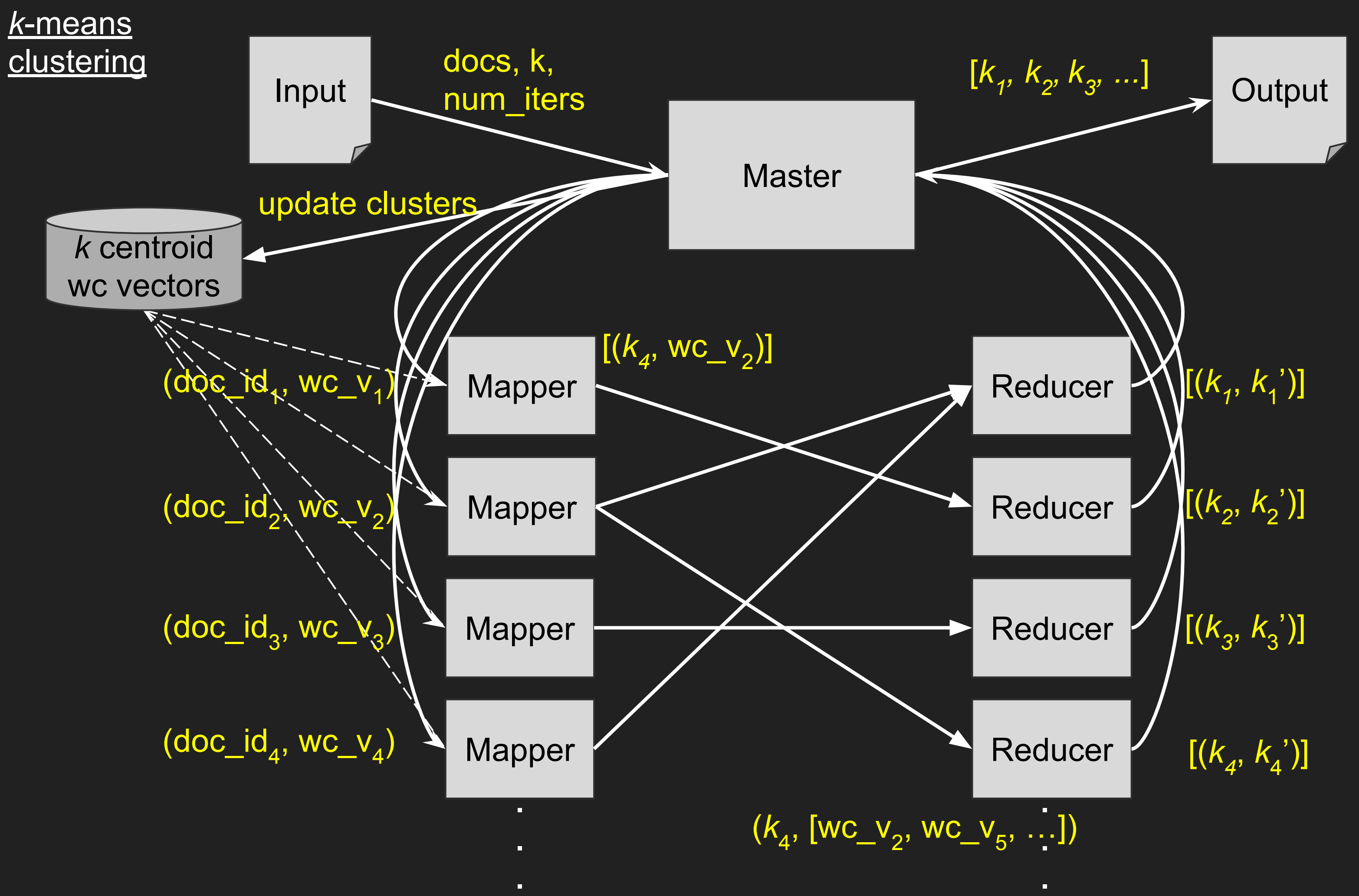
- Word-count
  - Given a list of documents, for every word in any document output how many times each word appears across all documents
- Inverted Index
  - Given a list of documents, for every word in any document output for each word which documents those words appear in
- *k*-means clustering
  - Given *n* points in space, find *k* clusters that best-fit the data set
  - Specifically word_count vectors for documents; using word counts to cluster similar documents
  - Spoilers: requires multiple map/reduce runs!
- PageRank
  - Given a graph of web links, find the probability that a user will end up on any given page
  - Spoilers: requires multiple map/reduce runs!

Word Count

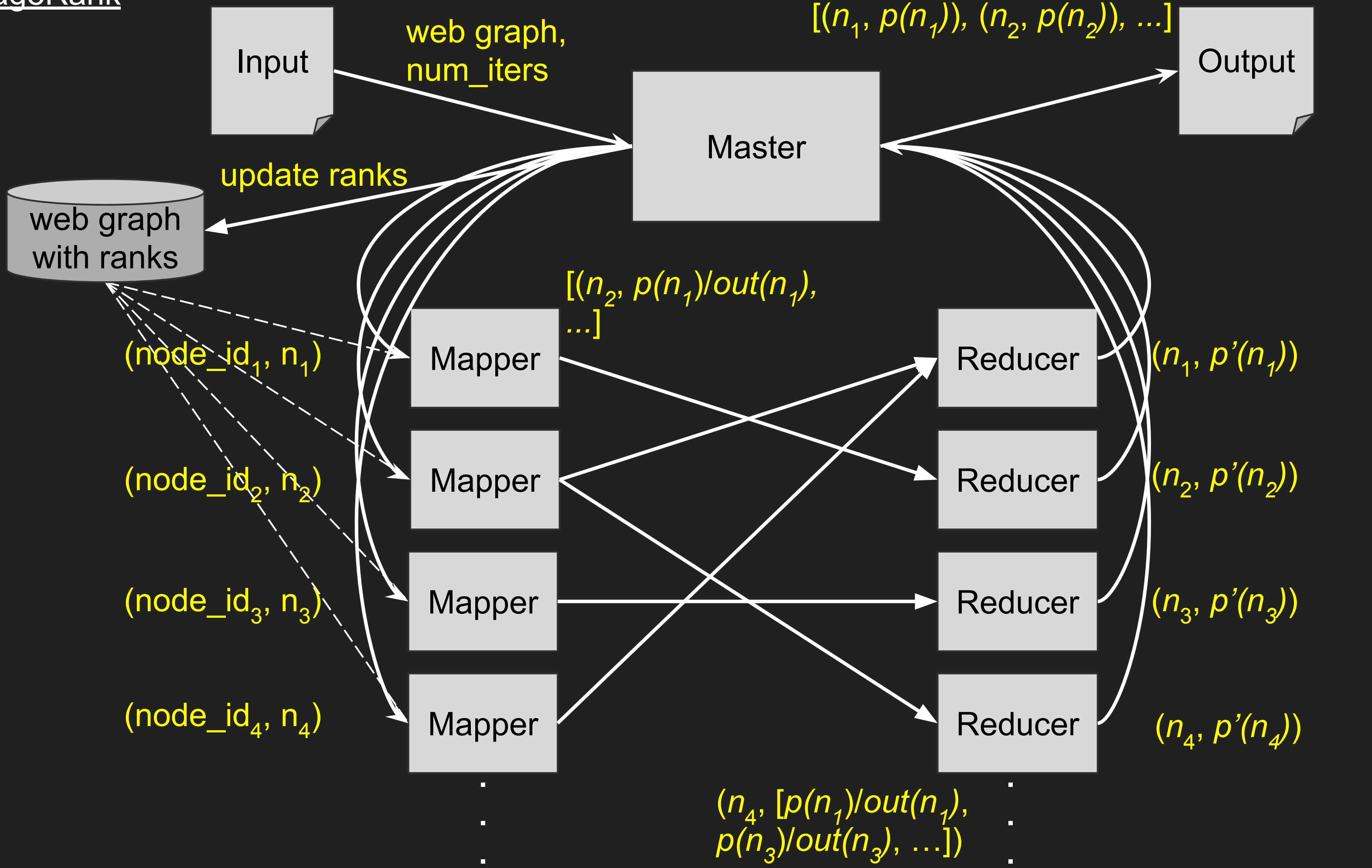

Input
documents
Master
[(word$_1$, word_count$_1$), (word$_2$, word_count$_2$),...]
Output
(doc_id$_1$, doc$_1$)
(doc_id$_2$, doc$_2$)
(doc_id$_3$, doc$_3$)
(doc_id$_4$, doc$_4$)
Mapper
Mapper
Mapper
Mapper
[(word$_1$, 1), (word$_2$, 1), ...)]
Reducer
Reducer
Reducer
Reducer
(word$_4$, [1, 1, 1, …])
[(word$_1$, word_count$_1$)]
[(word$_2$, word_count$_2$)]
[(word$_3$, word_count$_3$)]
[(word$_4$, word_count$_4$)]

Inverted Index




k-means clustering
Input
docs, k, num_iters
Master
[k1, k2, k3, …]
Output
update clusters
k centroid wc vectors
(doc_id1, wc_v1)
(doc_id2, wc_v2)
(doc_id3, wc_v3)
(doc_id4, wc_v4)
Mapper
Mapper
Mapper
Mapper
[(k4, wc_v2)]
Reducer
Reducer
Reducer
Reducer
[(k1, k1')]
[(k2, k2')]
[(k3, k3')]
[(k4, k4')]
(k4, [wc_v2, wc_v5, …])

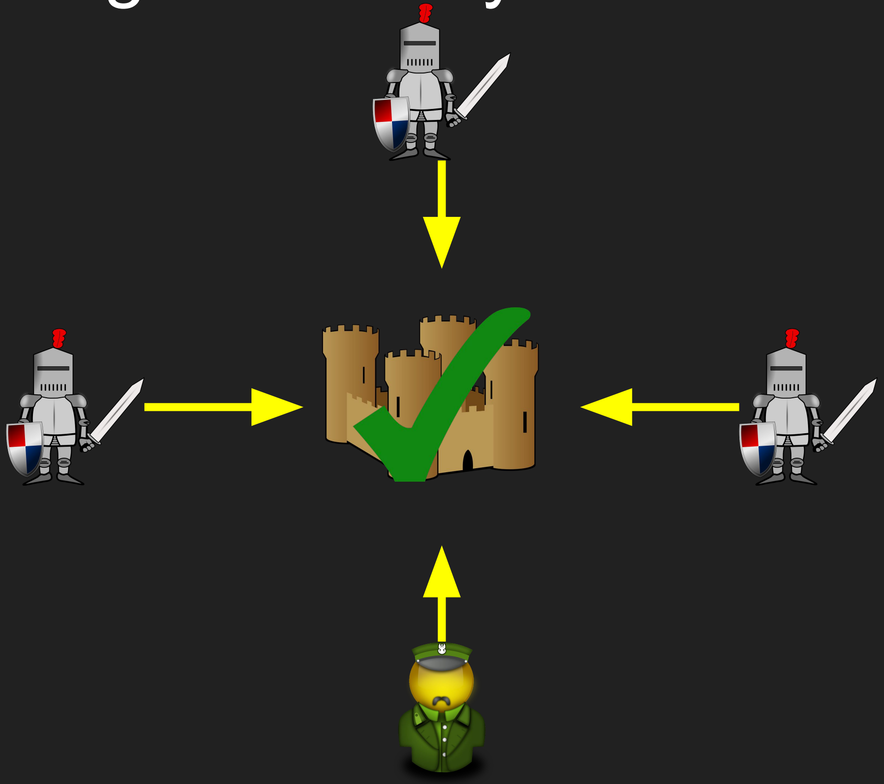
PageRank



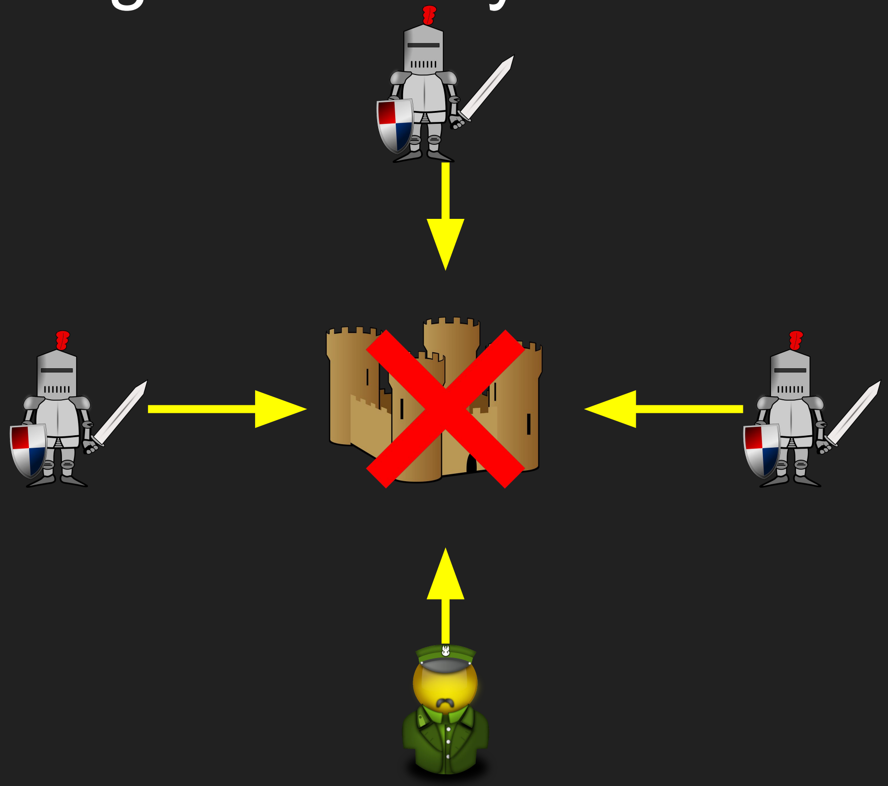
Input
web graph,
num_iters
Master
[(n1, p(n1)), (n2, p(n2)), ...]
Output
update ranks
web graph
with ranks
[(n2, p(n1)/out(n1),
...]
(node_id1, n1)
(node_id2, n2)
(node_id3, n3)
(node_id4, n4)
Mapper
Mapper
Mapper
Mapper
Reducer
Reducer
Reducer
Reducer
(n1, p'(n1))
(n2, p'(n2))
(n3, p'(n3))
(n4, p'(n4))
(n4, [p(n1)/out(n1),
p(n3)/out(n3), ...])

Consensus

Consensus Algorithms: Byzantine Generals



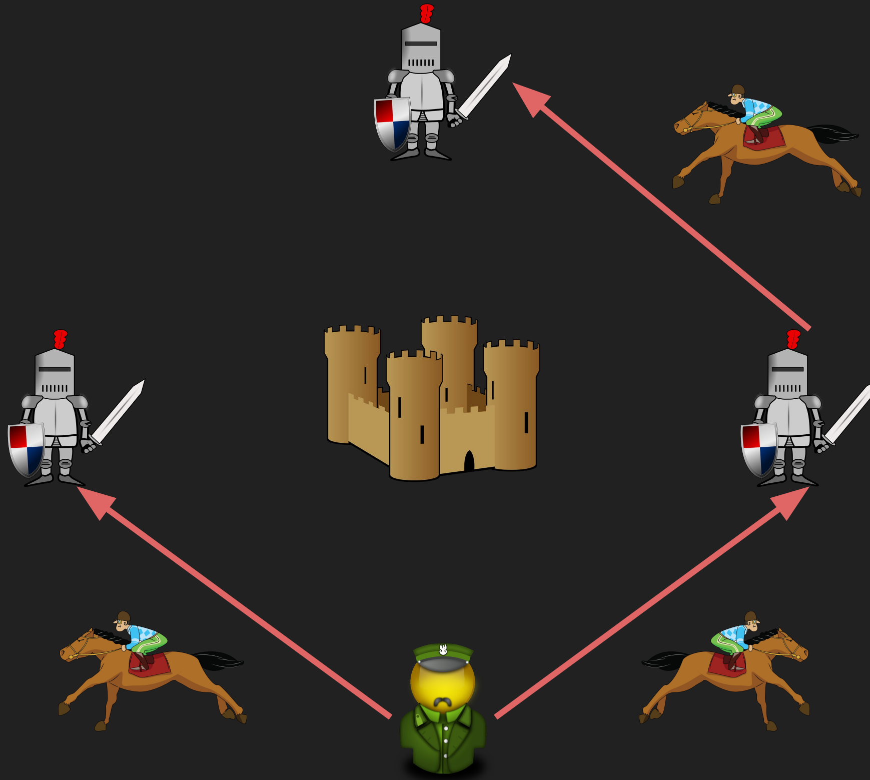
Consensus Algorithms: Byzantine Generals



Consensus Algorithms: Byzantine Generals

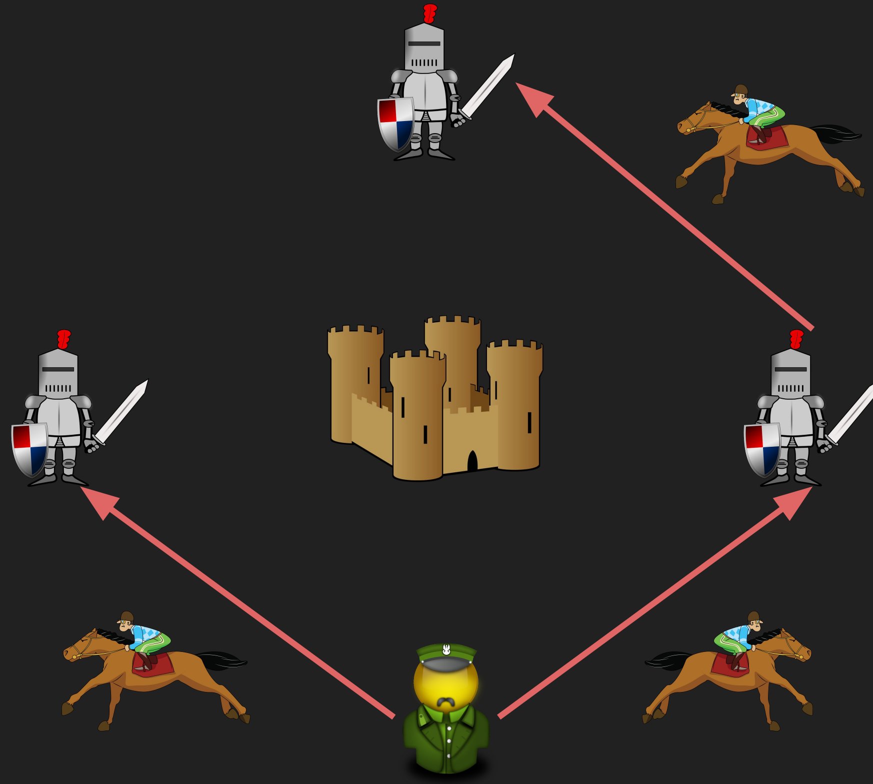
- Key Question: How do we coordinate with all the other generals at once?

- Assume we can't send signals the enemy can see (like torches)
- We're going to have to send messengers



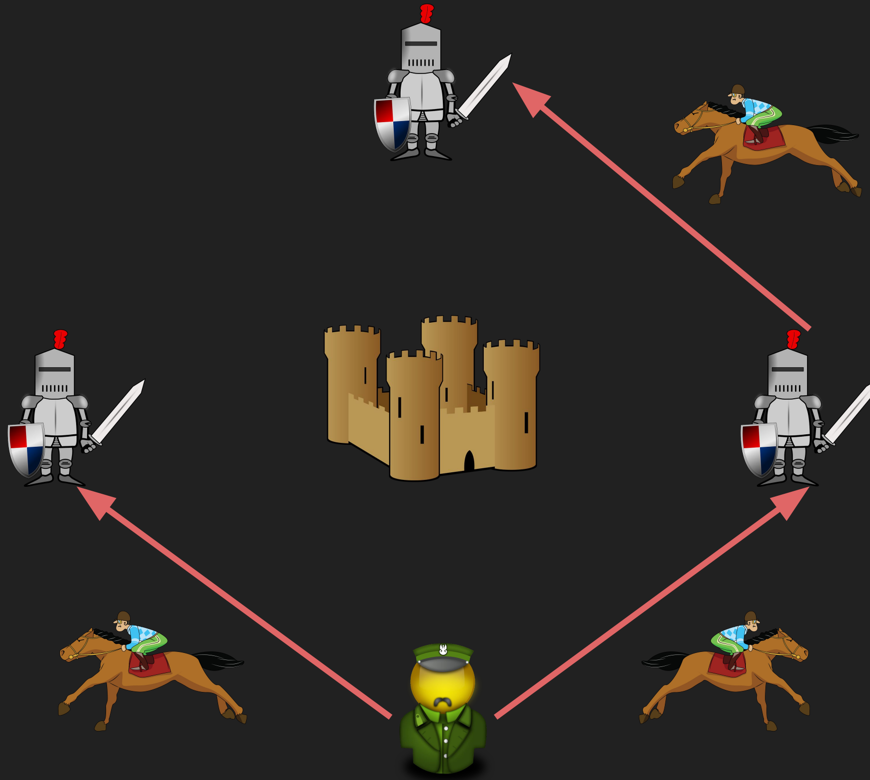
Consensus Algorithms: Byzantine Generals

- What issues might we have?
  - How do we know the messengers made it?
  - How do we know that the message wasn't intercepted and replaced?
    - Same for the response
  - How do we know that the generals will even go along with the plan?



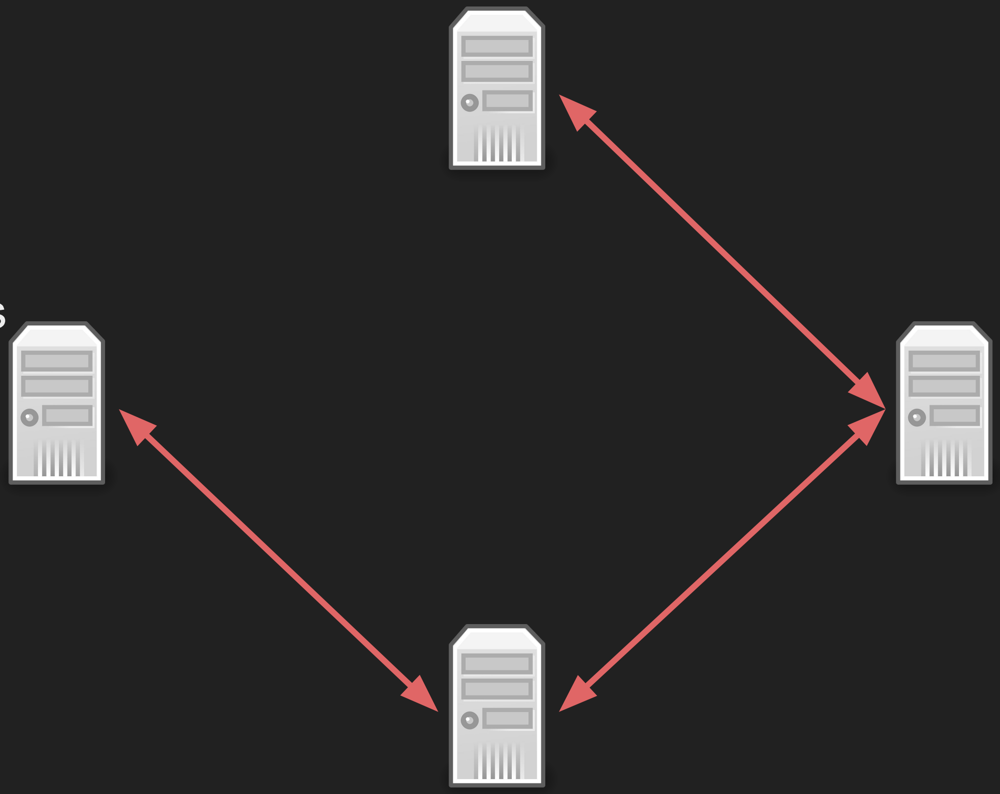
Consensus Algorithms: Byzantine Generals

- How does this relate to Computer Science?
  - Replace generals with computers and messengers with network packets



Consensus Algorithms: Byzantine Generals

- How does this relate to Computer Science?
  - Replace generals with computers and messengers with network packets



Consensus Algorithms: Byzantine Generals

- What issues might we have?
  - How do we know the packets made it?
    - Networking (TCP)
  - How do we know that the packet wasn't intercepted and replaced?
    - Encryption, Digital Signatures, etc.
  - How do we know that the other computers aren't working against us?