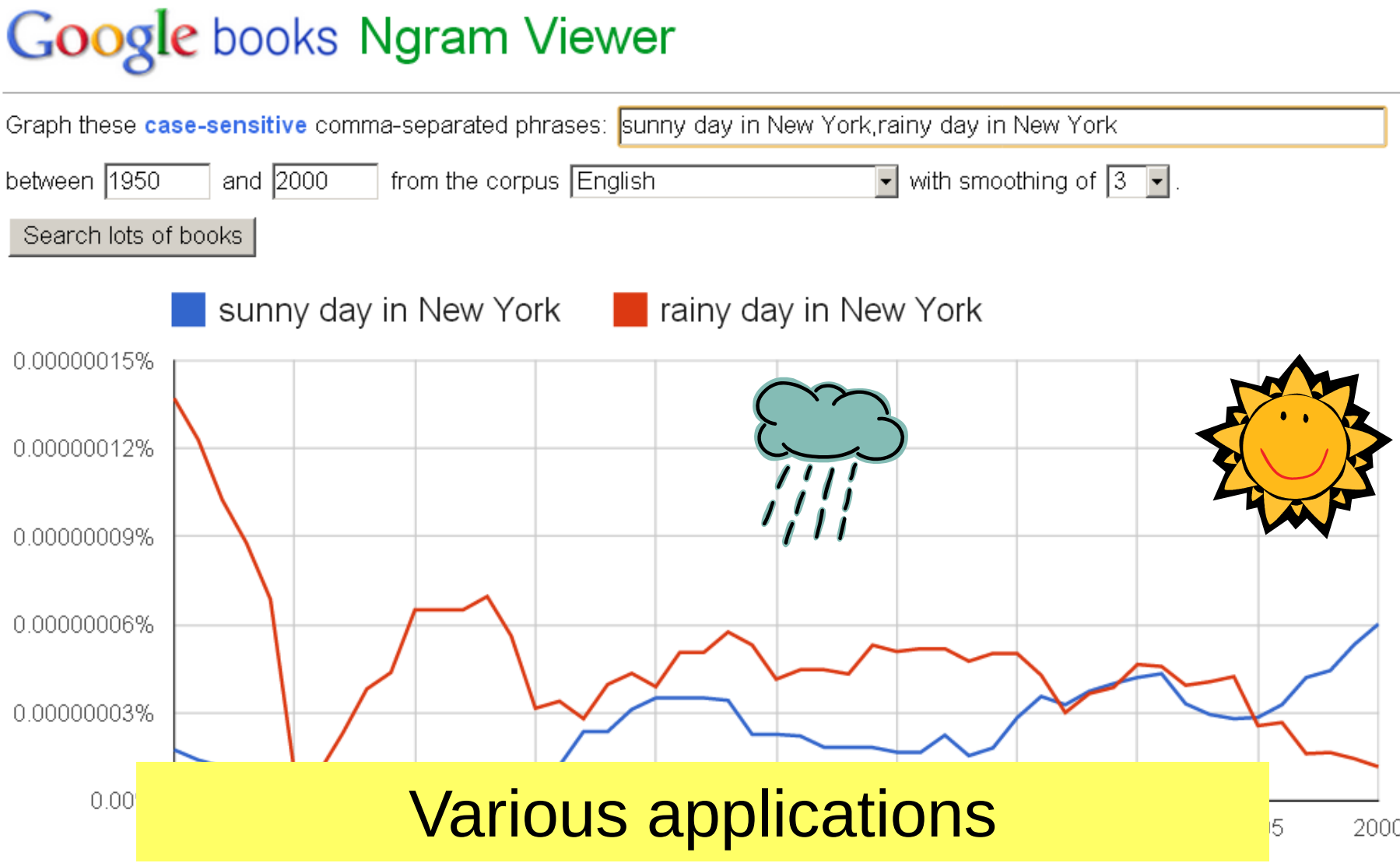
Mind the Gap: Large-Scale Frequent Sequence Mining

Iris Miliaraki Klaus Berberich Rainer Gemulla Spyros Zoupanos

Max Planck Institute for Informatics

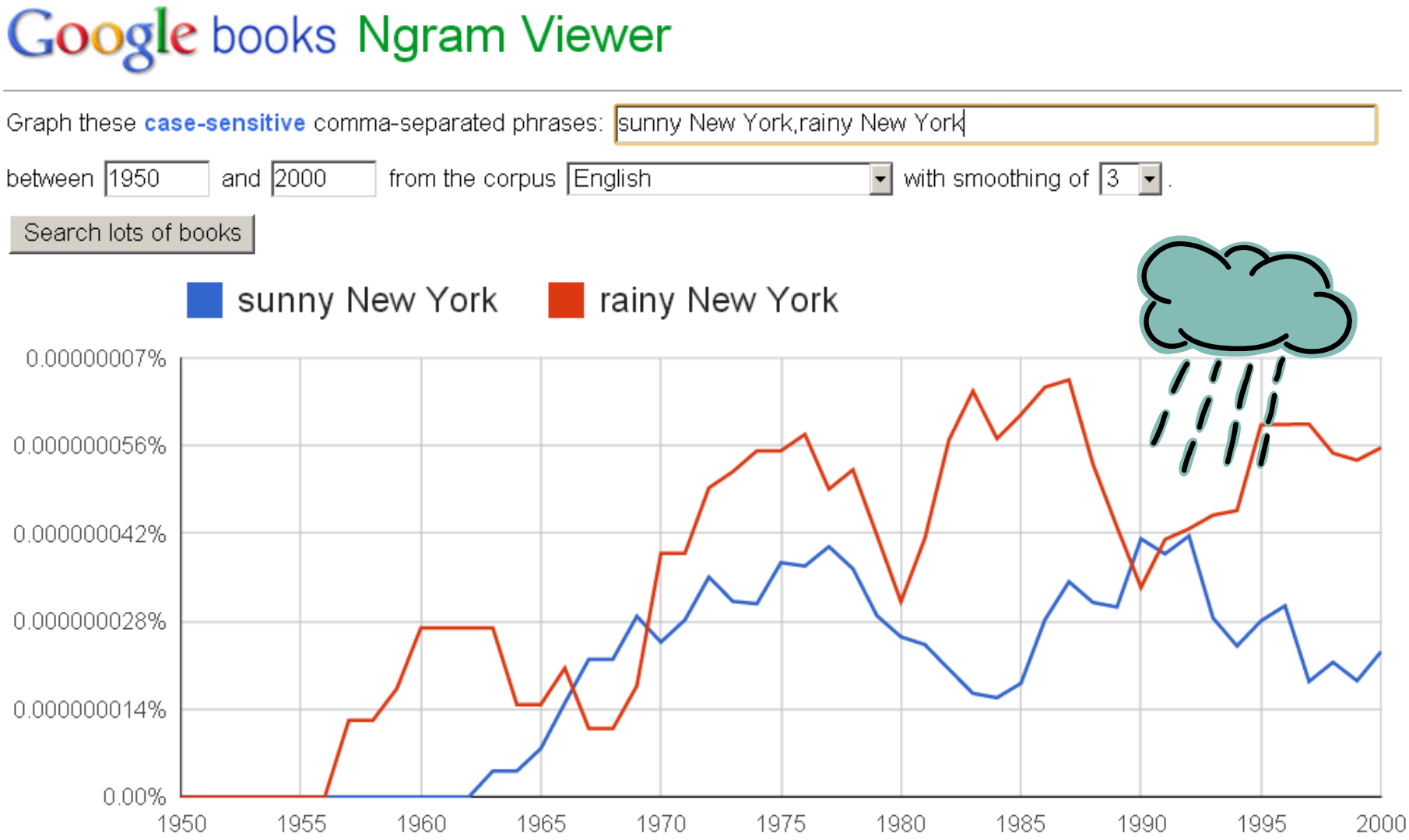
Saarbrücken, Germany

Why are sequences interesting?



Various applications

Why are sequences interesting?



Sequences with gaps

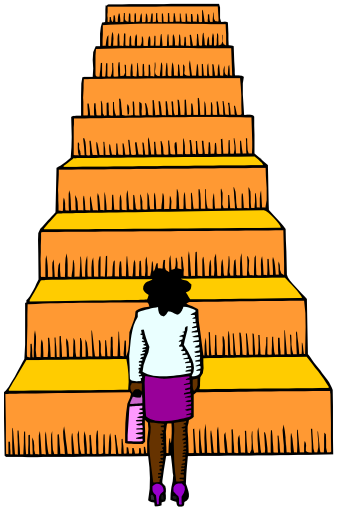
- Generalization of n-grams to sequences with gaps
  - sunny [...] New York
  - rainy [...] New York

- Exposes more structure
  - ✓ Central Park is the best place to be on a **sunny** day in **New York**.
  - ✓ It was a **sunny**, beautiful **New York** City afternoon.

More applications....

- **Text analysis** (e.g., linguistics or sociology)
- **Language modeling** (e.g, query completion)
- **Information extraction** (e.g,relation extraction)
- Also: web usage mining, spam detection, ...

Challenges



Huge collections of sequences

Computationally intensive problem

- $O(n^2)$ n-grams for sequence S where |S| = n
- $O(2^n)$ subsequences for sequence S where |S| = n and gap > n

Sequences with small support can be interesting

Potentially many output patterns

How can we perform frequent sequence mining at such large scales?

Outline

- Motivation & challenges
- **Problem statement**
- The MG-FSM algorithm
- Experimental Evaluation
- Conclusion

Gap-constrained frequent sequence mining

Input: Sequence database

Output: Frequent subsequences that

- Occur in at least σ sequences (support threshold)
- Have length at most λ (length threshold)
- Have gap at most γ between consecutive items (gap threshold)

Central Park is the best place to be on a sunny day in New York.

Monday was a sunny day in New York.

It was a sunny, beautiful New York City afternoon.

Frequent n-gram sunny day in New York for $\sigma = 2$, $\gamma = 0$, $\lambda = 5$

Gap-constrained frequent sequence mining

Input: Sequence database

Output: Frequent subsequences that

- Occur in at least σ sequences (support threshold)
- Have length at most λ (length threshold)
- Have gap at most γ between consecutive items (gap threshold)

Central Park is the best place to be on a sunny day in New York.
Monday was a sunny day in New York.
It was a sunny, beautiful New York City afternoon.

Frequent n-gram sunny day in New York for $\sigma = 2, \gamma = 0, \lambda = 5$

Frequent n-gram New York for $\sigma = 3, \gamma = 0, \lambda = 2$

Gap-constrained frequent sequence mining

Input: Sequence database

Output: Frequent subsequences that

- Occur in at least σ sequences (support threshold)
- Have length at most λ (length threshold)
- Have gap at most γ between consecutive items (gap threshold)

Central Park is the best place to be on a sunny day in New York.
Monday was a sunny day in New York.
It was a sunny, beautiful New York City afternoon.

Frequent n-gram sunny day in New York for $\sigma = 2, \gamma = 0, \lambda = 5$

Frequent n-gram New York for $\sigma = 3, \gamma = 0, \lambda = 2$

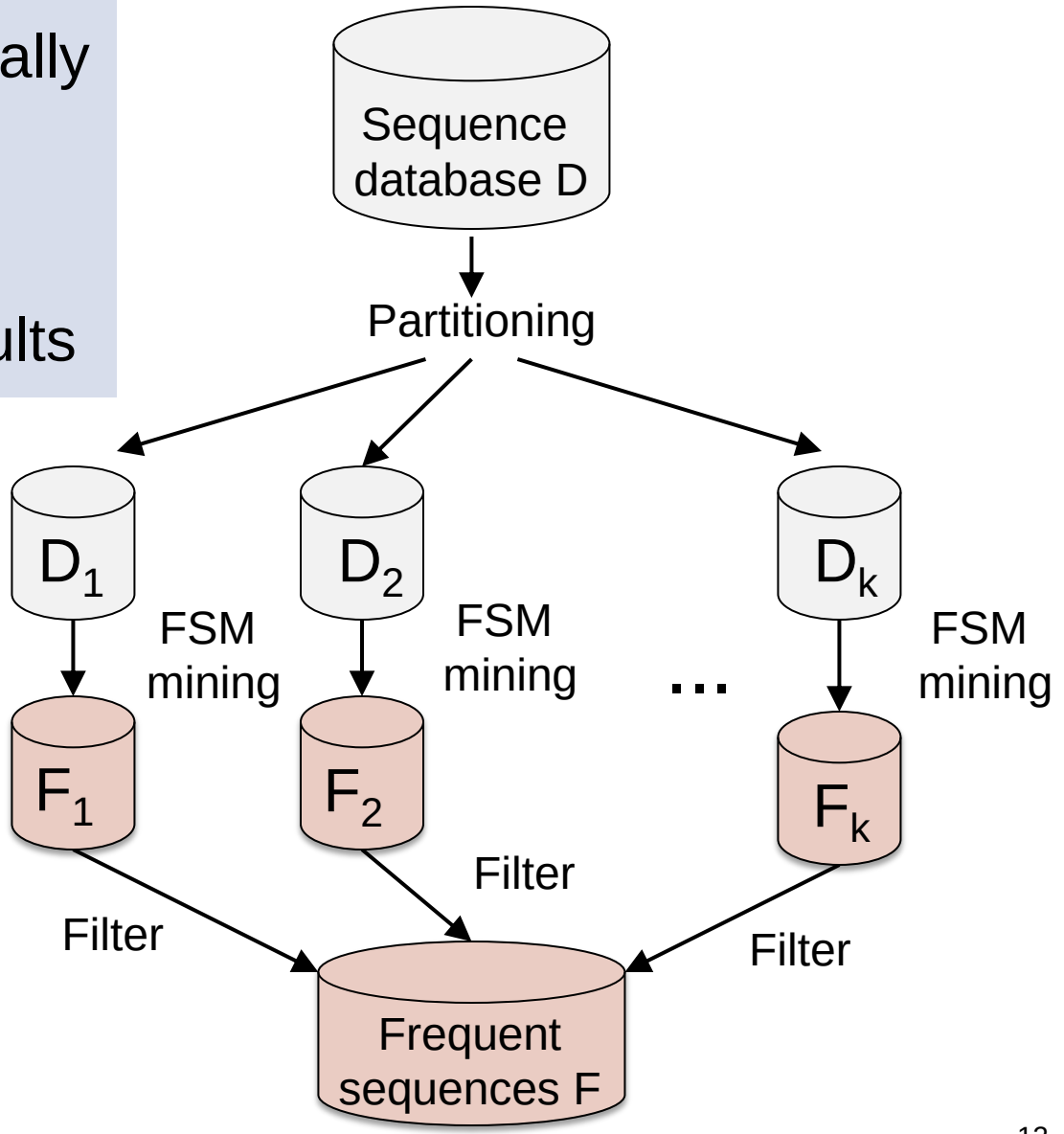
Frequent subsequence sunny New York for $\sigma = 3, \gamma \geq 2, \lambda = 3$

Outline

- Motivation & challenges
- Problem statement
- **The MG-FSM algorithm**
- Experimental Evaluation
- Conclusion

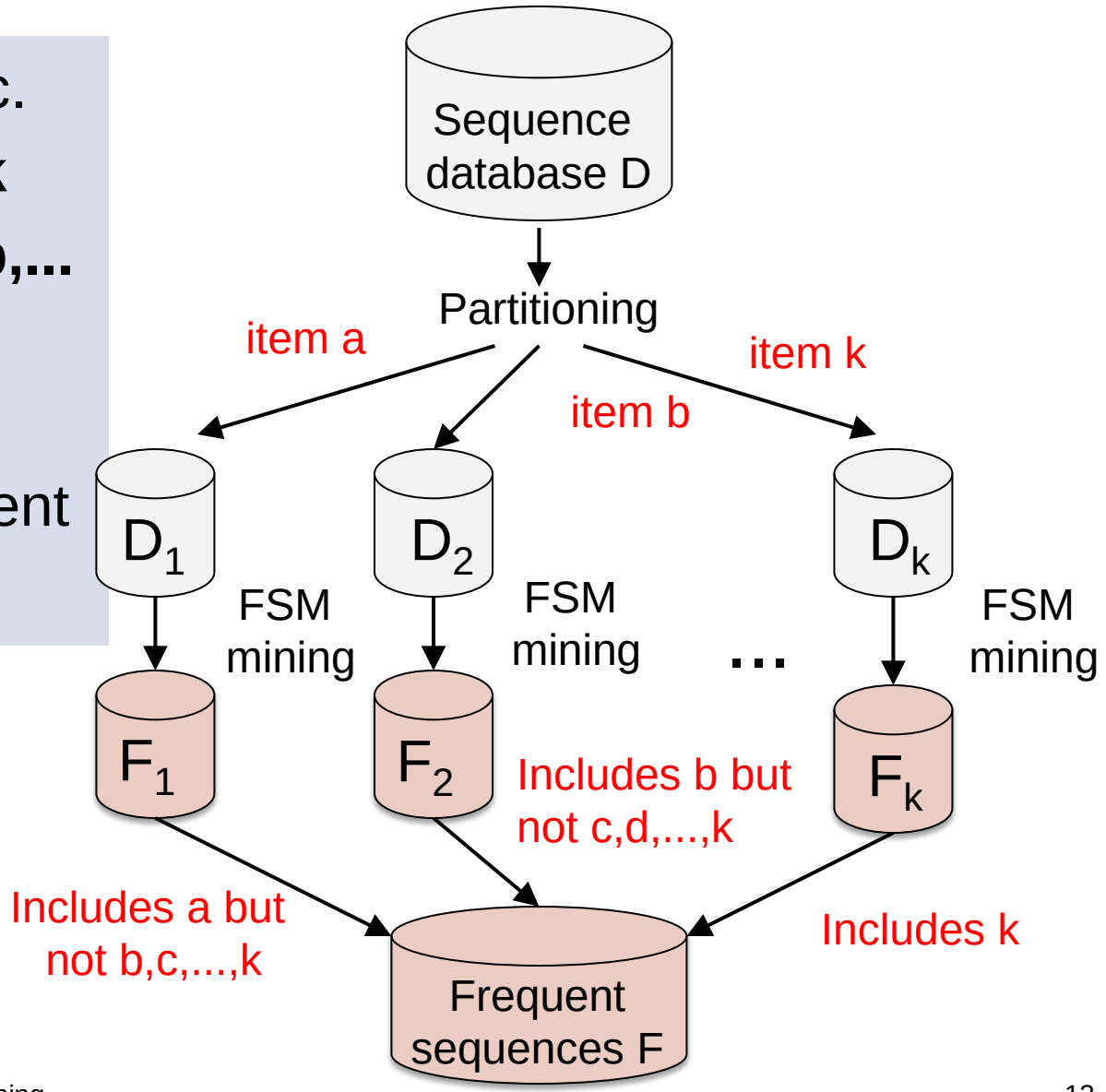
Parallel frequent sequence mining

1. Divide data into potentially overlapping partitions
2. Mine each partition
3. Filter and combine results



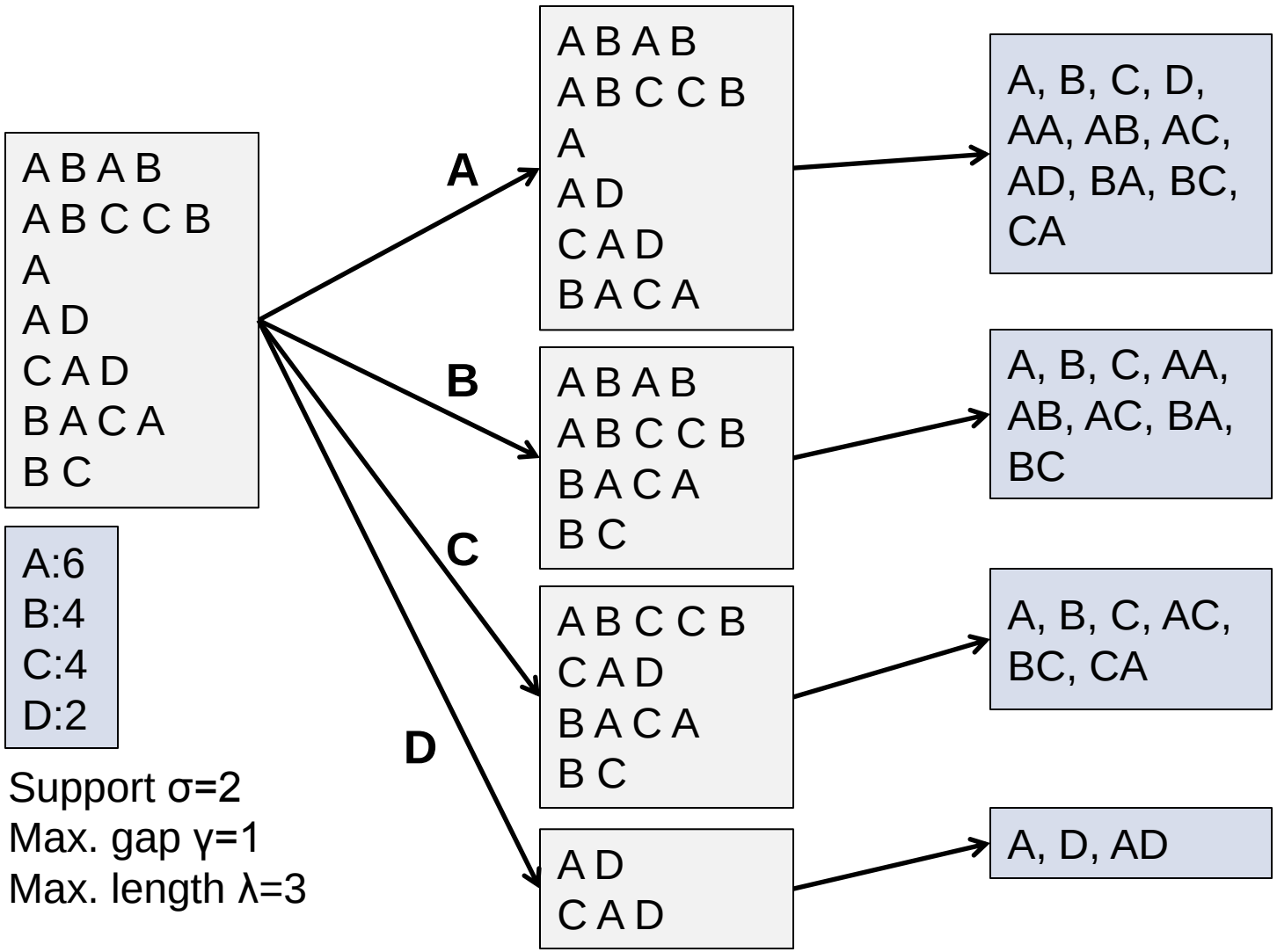
Using item-based partitioning

1. Order items by desc. frequency **a > ... > k**
2. Partition by item **a,b,...** (called **pivot** item)
3. Mine each partition
4. Filter: no less-frequent item

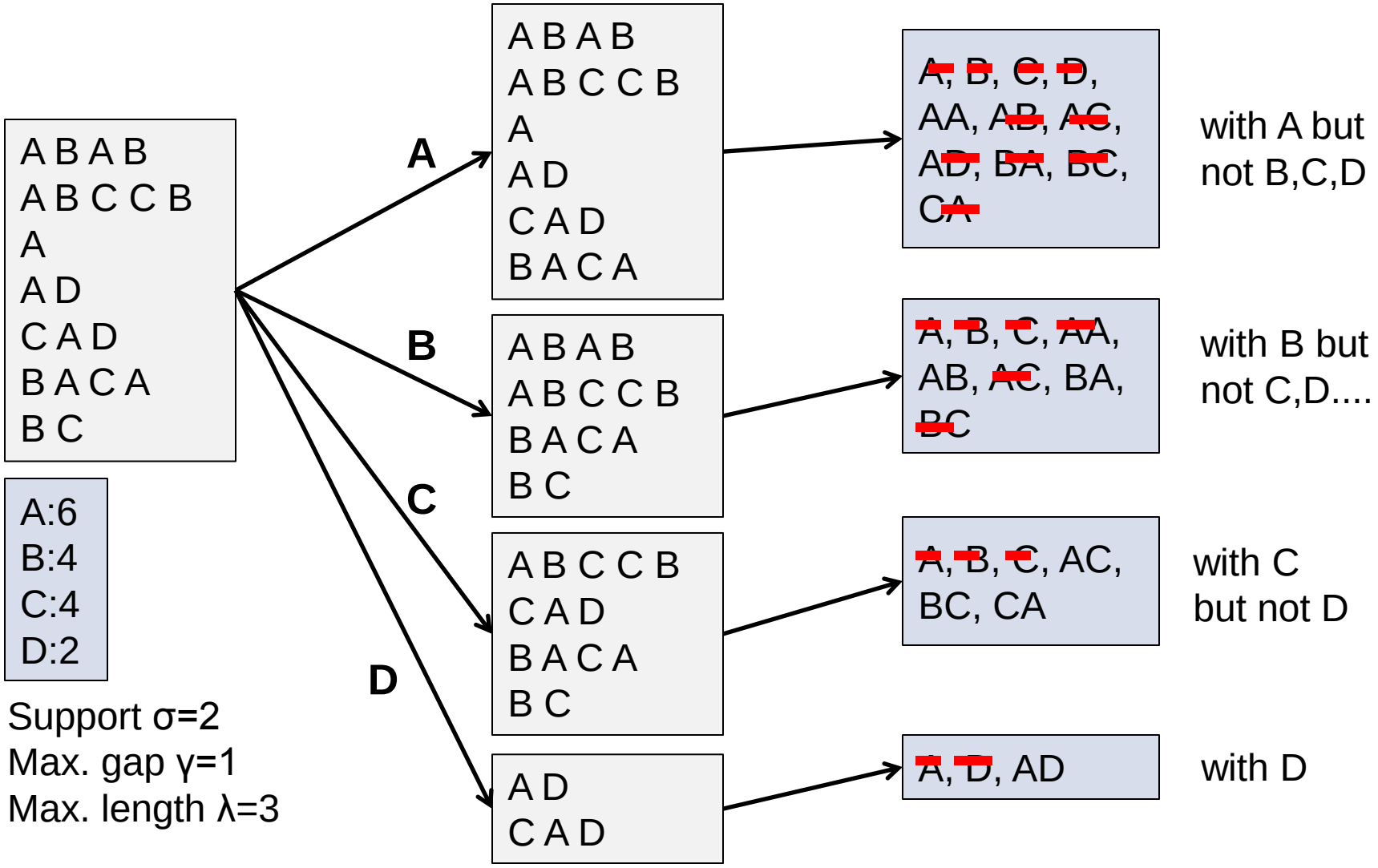


Disjoint subsequence sets computed in-parallel and independently

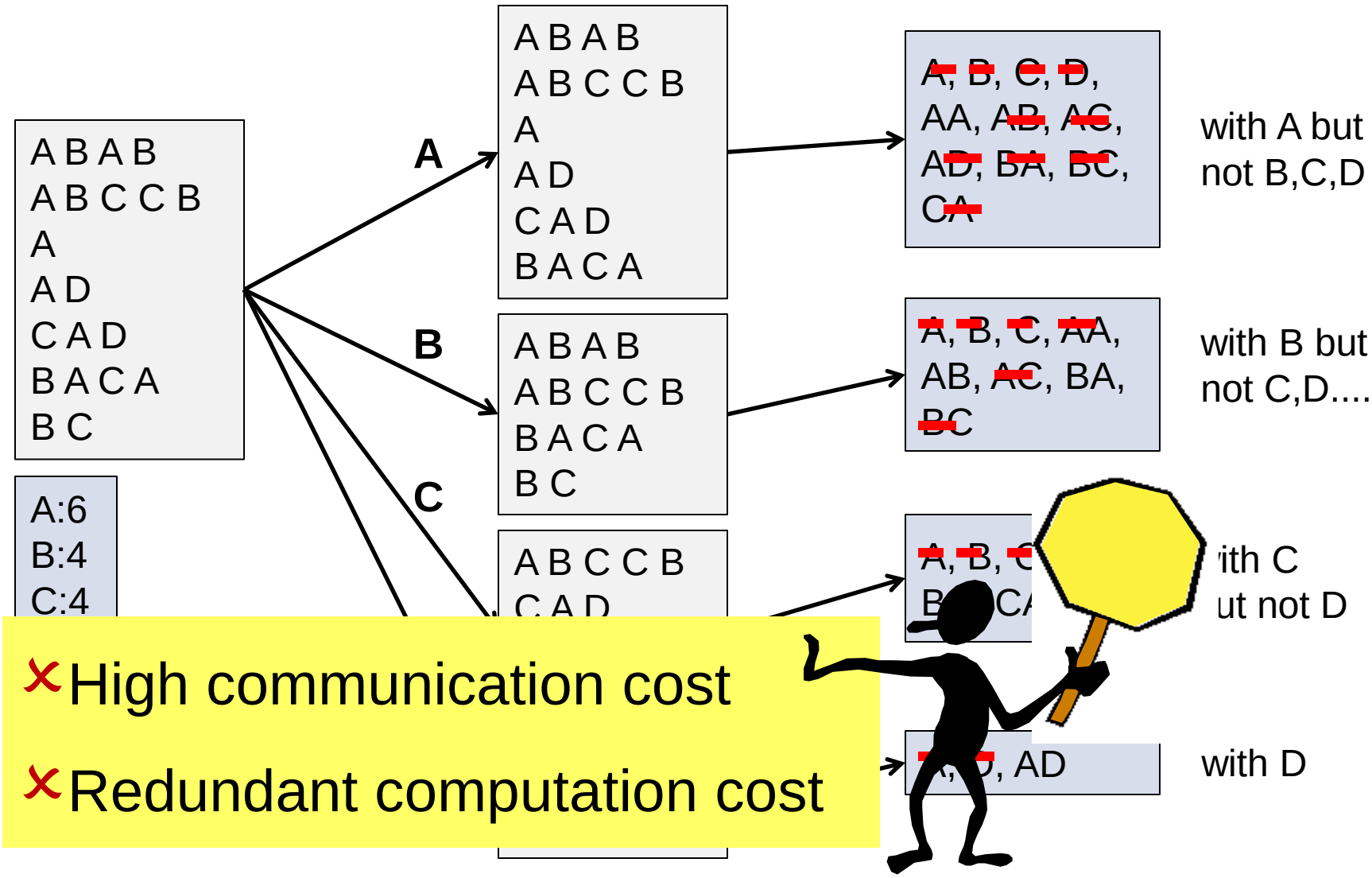
Example: Naive partitioning



Example: Naive partitioning

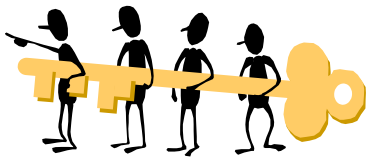


Example: Naive partitioning



Improving the partitioning

Traditional approach



- Derive a partitioning rule (“projection”)
- Prove correctness of the partition rule

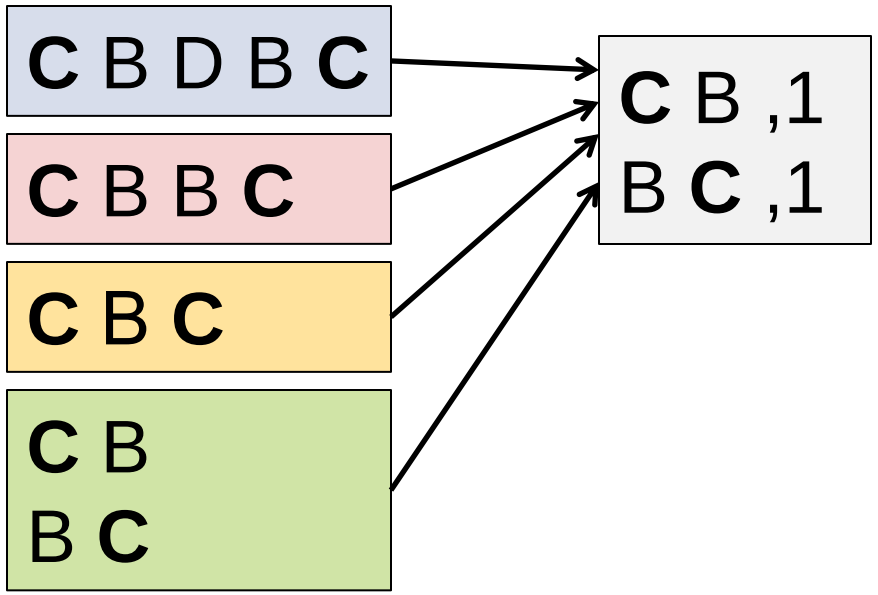
MG-FSM approach

- Use *any* partitioning satisfying correctness
- Rewrite the input sequences →

  ensuring each w-partition generates the set of **pivot sequences** for w

Which is the optimal partition?

Max. gap γ=0
Max. length λ=2
pivot **C**



Many short sequences? Few long sequences?
Optimal partition not clear!

Aim for a “good” partition
using inexpensive rewrites

Trade-off cost & gain

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
D A **C** B D
D A **C** B D D B **C** A
B **C** A D D B D
A D D **C** D

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
D A **C** B **D**
D A **C** B **D** **D** B **C** A
B **C** A **D** **D** B **D**
A **D** **D** **C** **D**

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)

Rewriting partitions

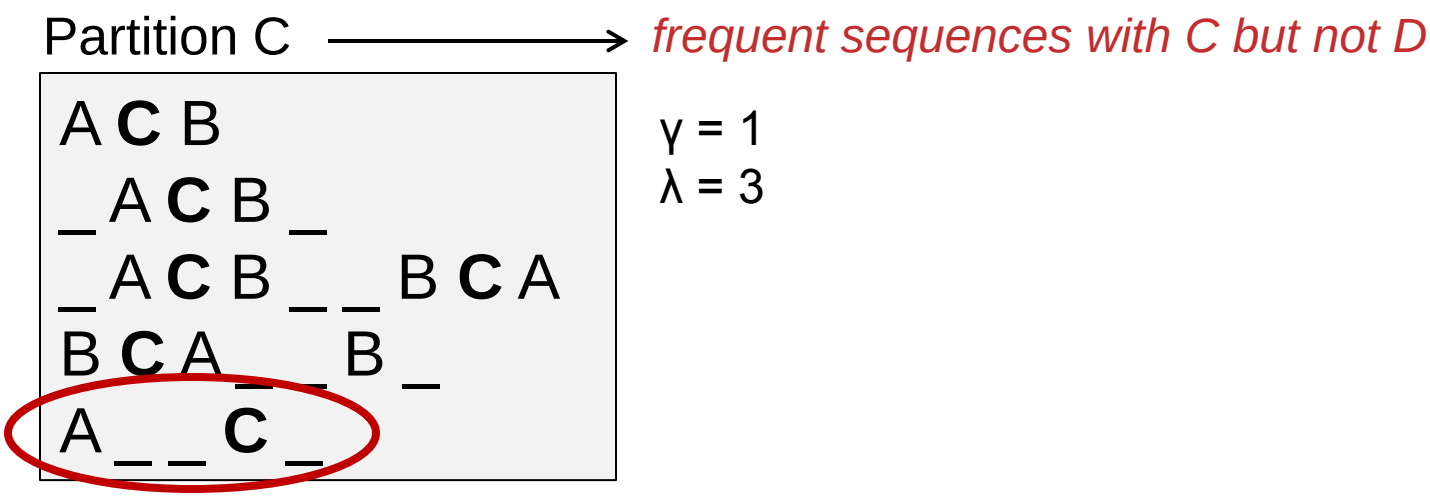
Partition C → *frequent sequences with C but not D*

A **C** B
_ A **C** B _
_ A **C** B _ _ B **C** A
B **C** A _ _ B _
A _ _ **C** _

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)

Rewriting partitions



1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)

2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
_ A **C** B _
_ A **C** B _ _ B **C** A
B **C** A _ _ B _
~~A _ _ **C** _~~

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
_ A **C** B _
_ A **C** B _ _ B **C** A
B **C** A _ _ **B** _

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
_ A **C** B _
_ A **C** B _ _ B **C** A
B **C** A _

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
_ A **C** B _
_ A **C** B _ _ B **C** A
B **C** A _

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)
4. Remove trailing and leading blanks

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
A **C** B
A **C** B _ _ B **C** A
B **C** A

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)
4. Remove trailing and leading blanks

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
A **C** B
A **C** B _ _ B **C** A
B **C** A

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)
4. Remove trailing and leading blanks
5. Break up sequence at split points (i.e., sequences of γ+1 blanks)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B
A **C** B
A **C** B
B **C** A
B **C** A

γ = 1
λ = 3

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)
4. Remove trailing and leading blanks
5. Break up sequence at split points (i.e., sequences of γ+1 blanks)

Rewriting partitions

Partition C → *frequent sequences with C but not D*

A **C** B **: 3**
~~A **C** B~~
~~A **C** B~~
~~B **C** A~~ **: 2**
B **C** A

γ = 1
λ = 3

A **C** B
D A **C** B D
D A **C** B D D B **C** A
B **C** A D D B D
A D D **C** D

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)
4. Remove trailing and leading blanks
5. Break up sequence at split points (i.e., sequences of γ+1 blanks)
6. Aggregate repeated subsequences

Rewriting partitions

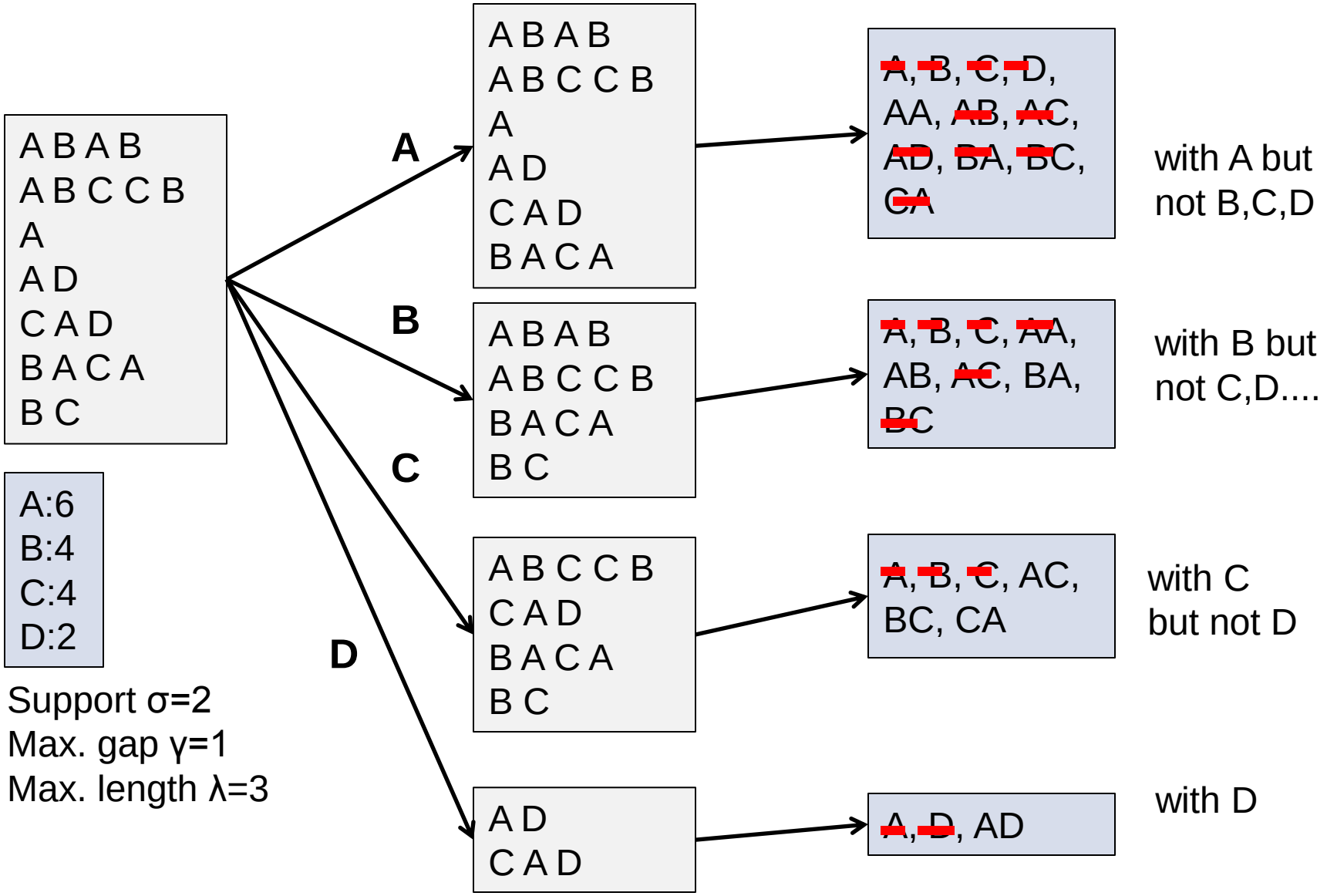
Partition C → *frequent sequences with C but not D*

A **C** B : 3 B **C** A : 2	γ = 1 λ = 3

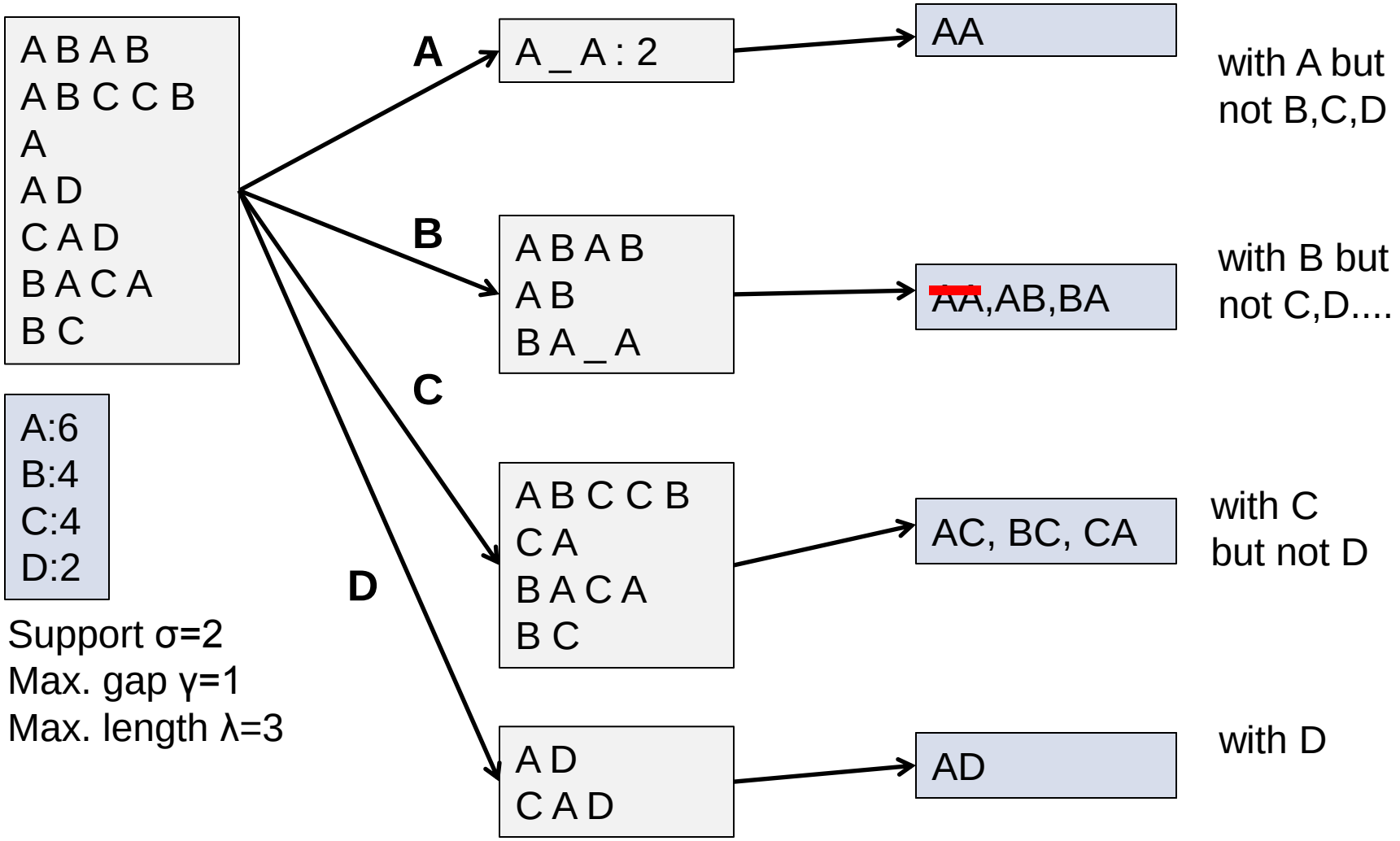
A **C** B
D A **C** B D
D A **C** B D D B **C** A
B **C** A D D B D
A D D **C** D

1. Replace irrelevant items (i.e., less frequent) by special blank symbol (_)
2. Drop irrelevant sequences (i.e., cannot generate any pivot sequence)
3. Remove all unreachable items (provably correct)
4. Remove trailing and leading blanks
5. Break up sequence at split points (i.e., sequences of γ+1 blanks)
6. Aggregate repeated subsequences

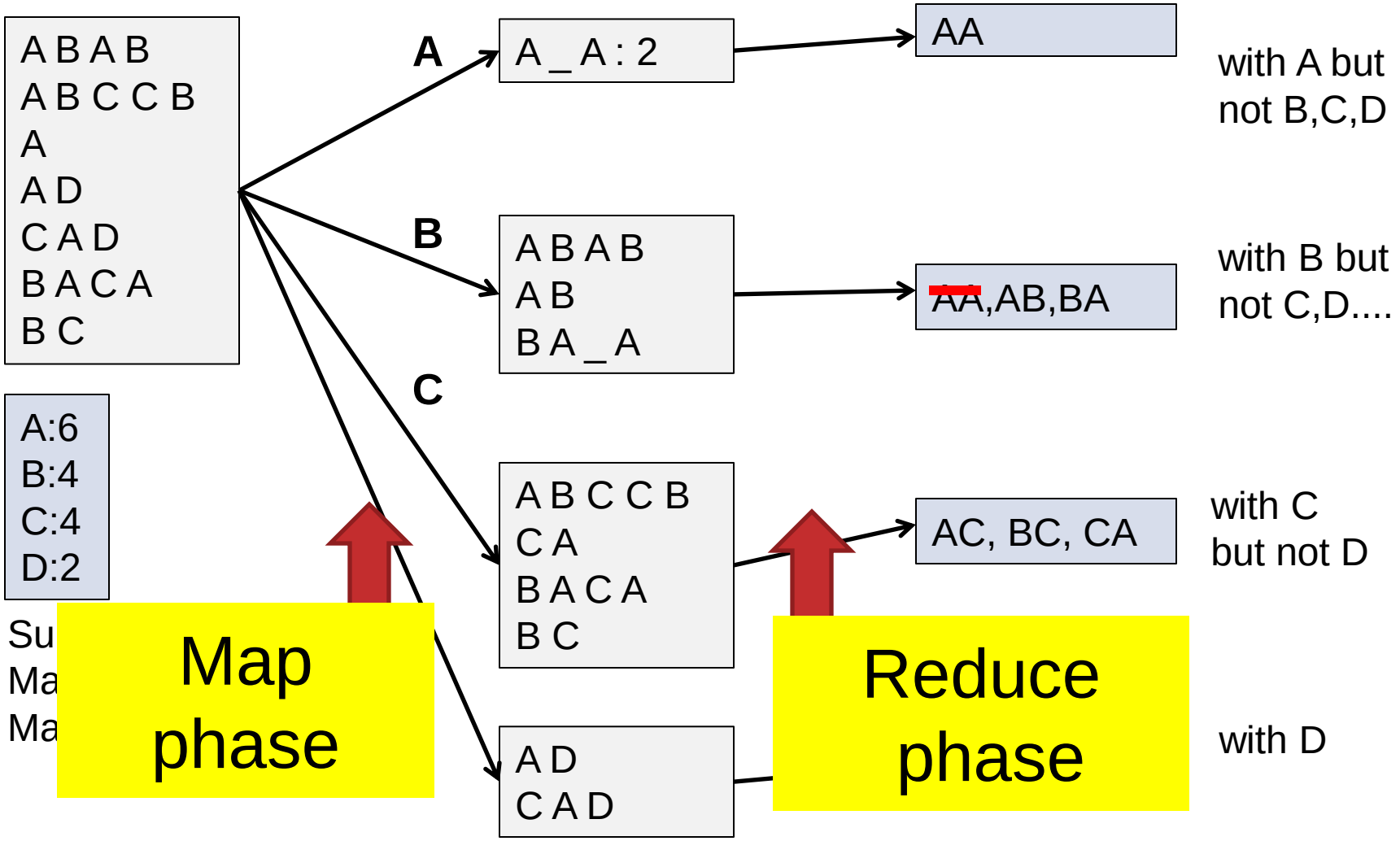
Revisiting example: Naive partitioning



Revisiting example: MG-FSM partitioning



Revisiting example: MG-FSM partitioning



Outline

- Motivation & challenges
- Problem statement
- The MG-FSM algorithm
- **Experimental Evaluation**
- Conclusion

Experimental evaluation: Setup

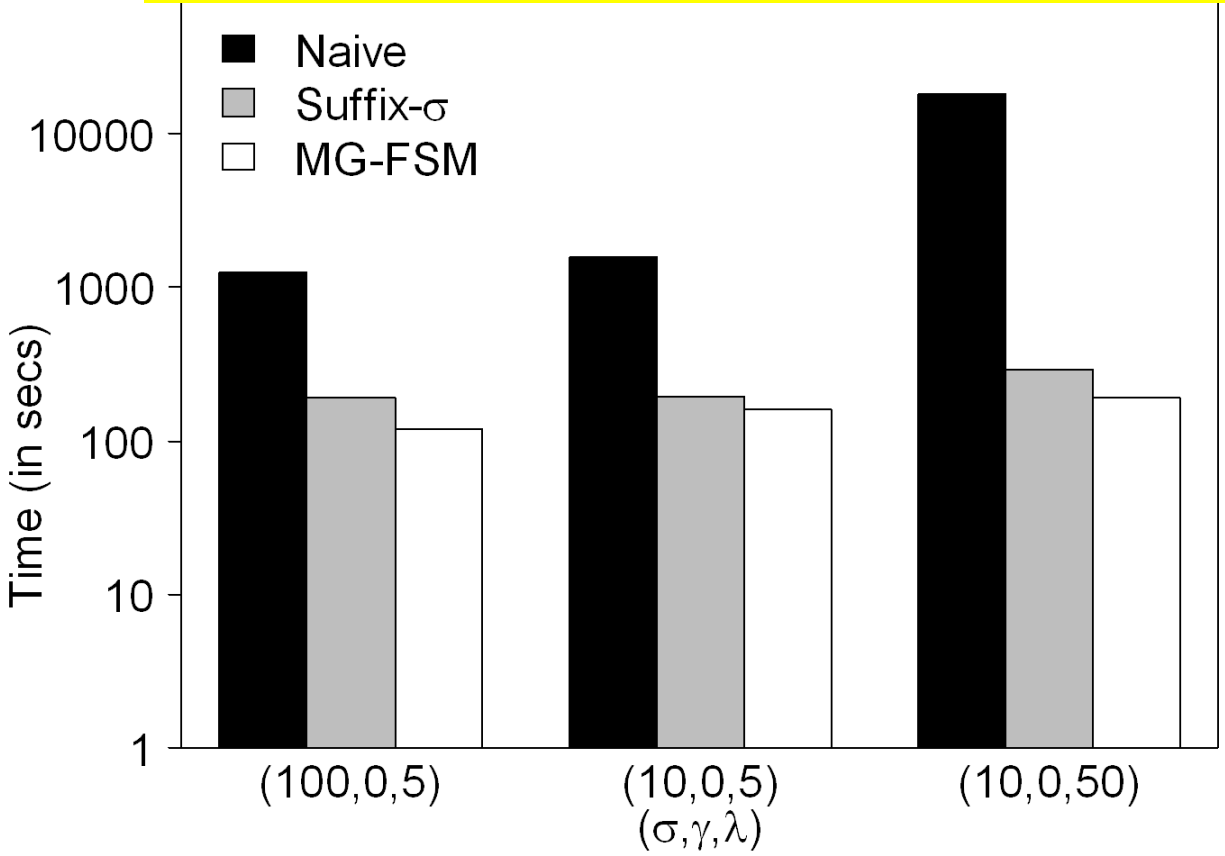
- Algorithms
  - MG-FSM
  - Naive algorithm for MapReduce
  - Suffix-σ (state-of-the-art n-gram miner)
- Setting
  - 10-machine local cluster
  - 10 GBit/64GB of main memory/eight 2TB SAS 7200 RPM hard disks/2 Intel Xeon E5-2640 6-core CPUs
  - Cloudera cdh3u0 distribution of Hadoop 0.20.2.

Experimental evaluation: Datasets

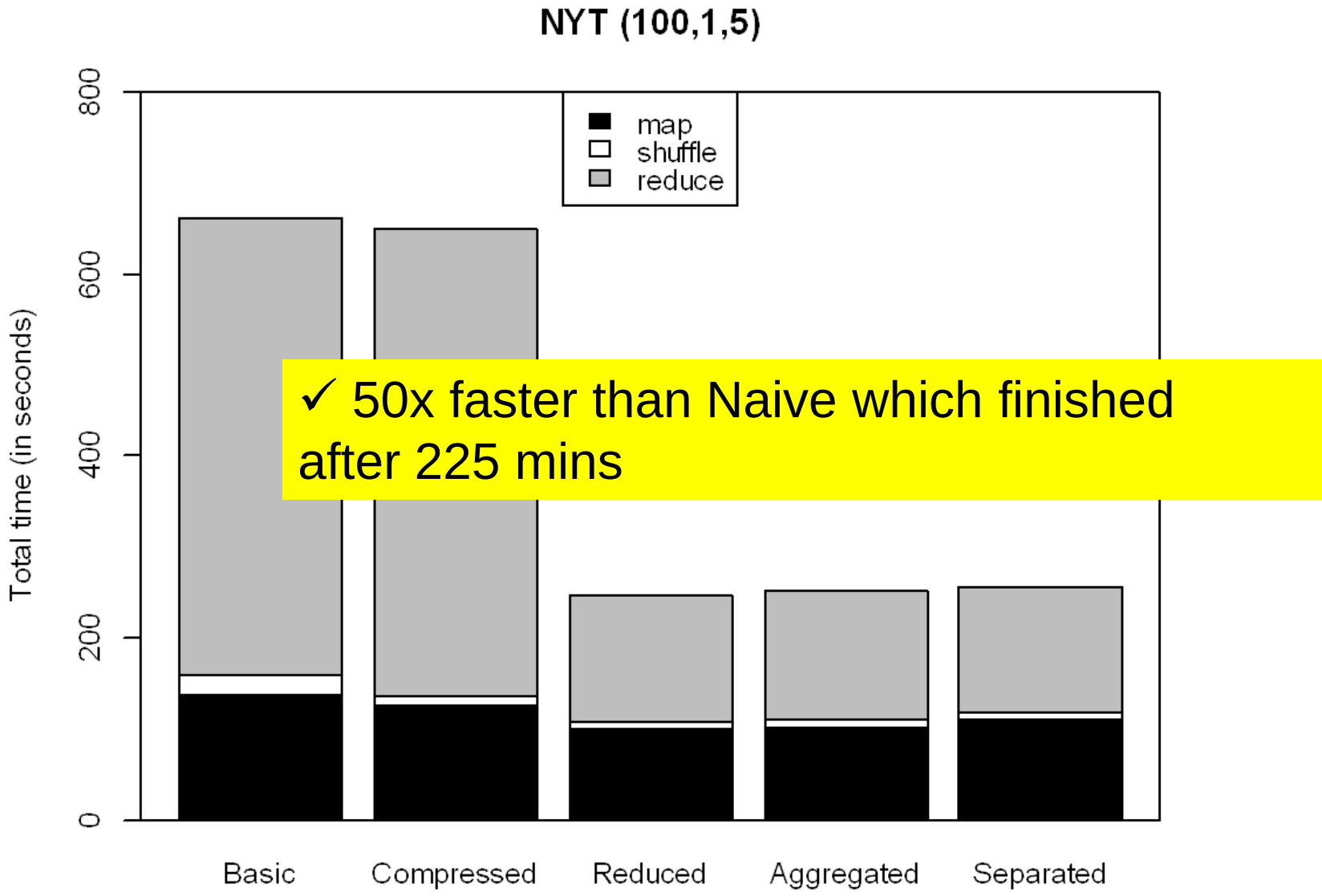
	ClueWeb	New York Times
Average length	19	19
Maximum length	20 993	21 174
Total sequences	1 135 036 279	53 137 507
Total items	21 565 723 440	1 051 435 745
Distinct items	7 361 754	1 577 233
Total bytes	66 181 963 922	3 087 605 146

n-gram mining (γ=0)

✓ Orders of magnitude faster than Naive
✓ Competitive to state-of-the-art n-gram miners



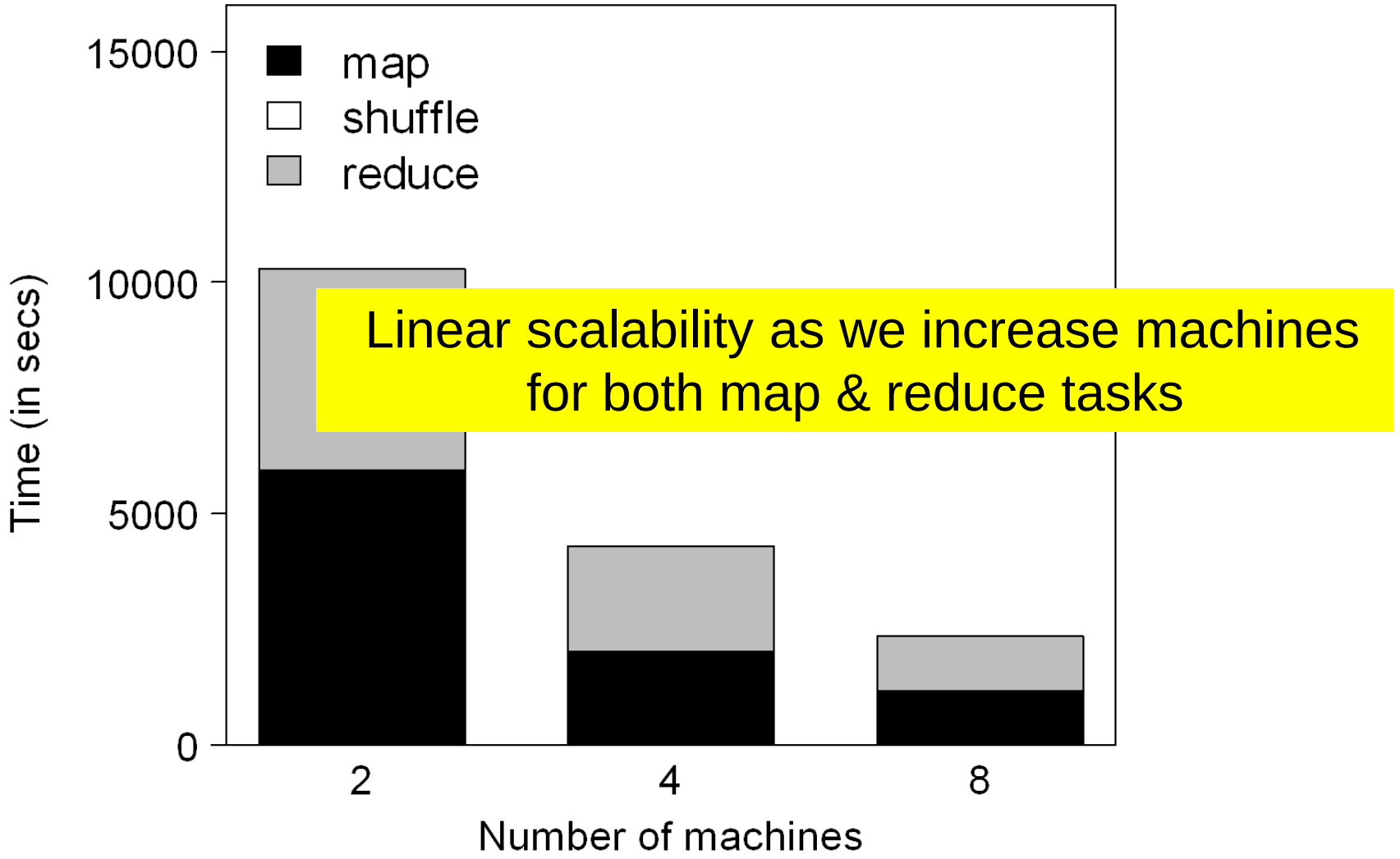
MG-FSM partition optimizations (time)



✓ 50x faster than Naive which finished after 225 mins

Strong scalability

σ=1000,γ=1,λ=5
50% ClueWeb



Linear scalability as we increase machines for both map & reduce tasks

Outline

- Motivation & challenges
- Problem statement
- The MG-FSM algorithm
- Experimental Evaluation
- Conclusion

Summary & Contributions

- MG-FSM mines frequent sequences with gap constraints
- Uses item-based partitioning → partitions can be mined independently and in parallel using *any FSM algorithm*
- Instead of “optimal” partitioning, MG-FSM uses efficient, inexpensive rewrites that ensure correctness
- Fast, low communication cost, scalable

Questions?