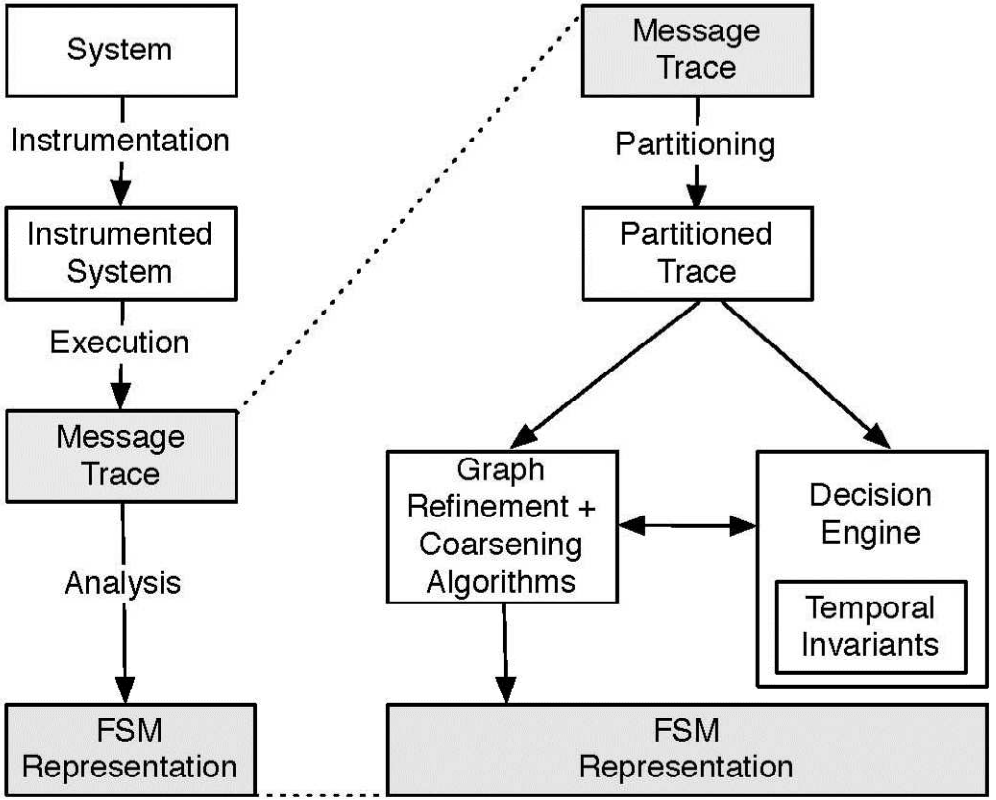
The Synoptic Analysis Pipeline



Trace Partitioning

- User choses a predicate
  - Examples (Source, Timestamp range, Parameter, etc)
- Trace is partitioned on the predicate into a set of trace partitions.
- Crucial to encode domain knowledge.
  - Example: For 2-Phase Commit each partition is an instance of the protocol execution.

Trace Partitioning Example with NFS

Unparitioned:

```
Lookup  file1
Lookup  file2
Access  file2
Access  file1
Access  file1
Read    file2
Read    file1
Getattr file2
Read    file1
Write   file2
```

Could infer invariant:
"lookup file1"
always precedes
"access file 2"
But this doesn't make sense.

Partitioned by file ID:

p_0

```
Lookup  file1
Access  file1
Access  file1
Read    file1
Read    file1
```

p_1

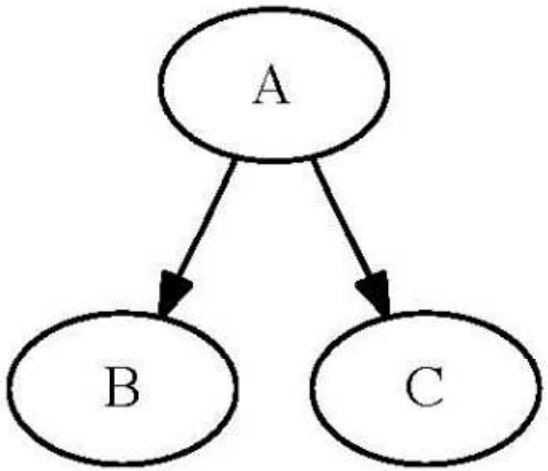
```
Lookup  file2
Access  file2
Read    file2
Getattr file2
Write   file2
```

Invariants inferred from these traces will be over access patterns to a particular file.

Interpreting Traces As Graphs

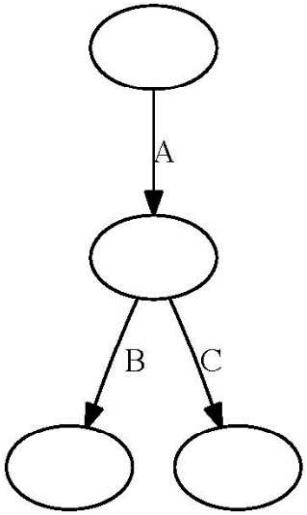
Relational Representation

- Captures the notion that messages are temporally related.



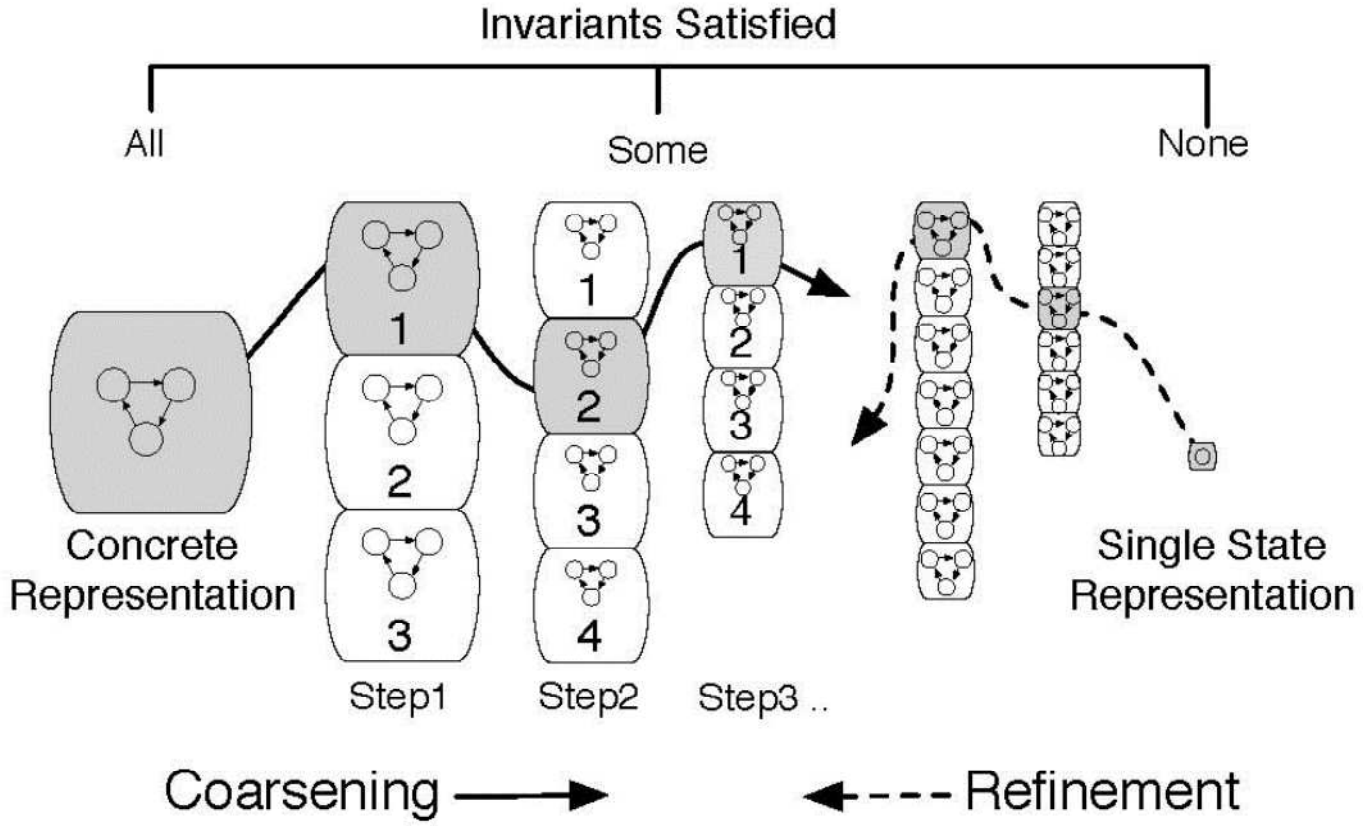
System State Representation

- Captures the notion that messages represent change in system state.



Note that we can convert one representation to the other

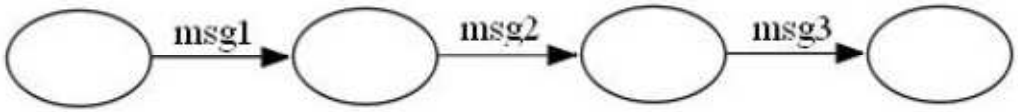
Exploring the Representations



Converge on the answer using either graph coarsening or graph refinement

Coarsening

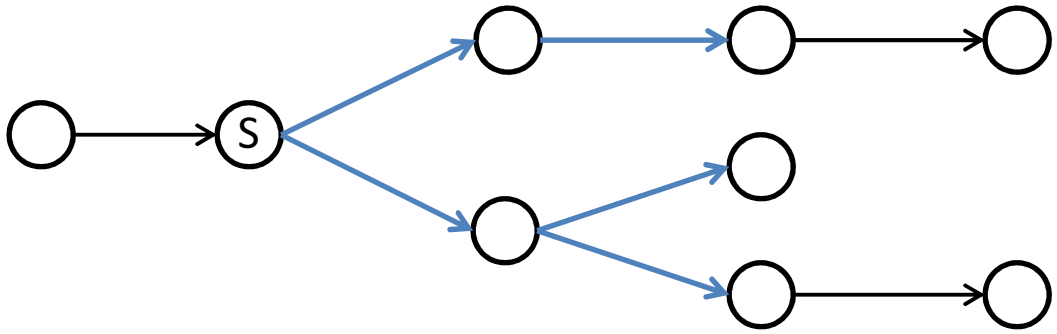
- Starts of with the most precise trace description - a linear graph for each trace partition:



- Graph is the system state representation of the trace.
- Tries to compress the graph to get a concise representation.

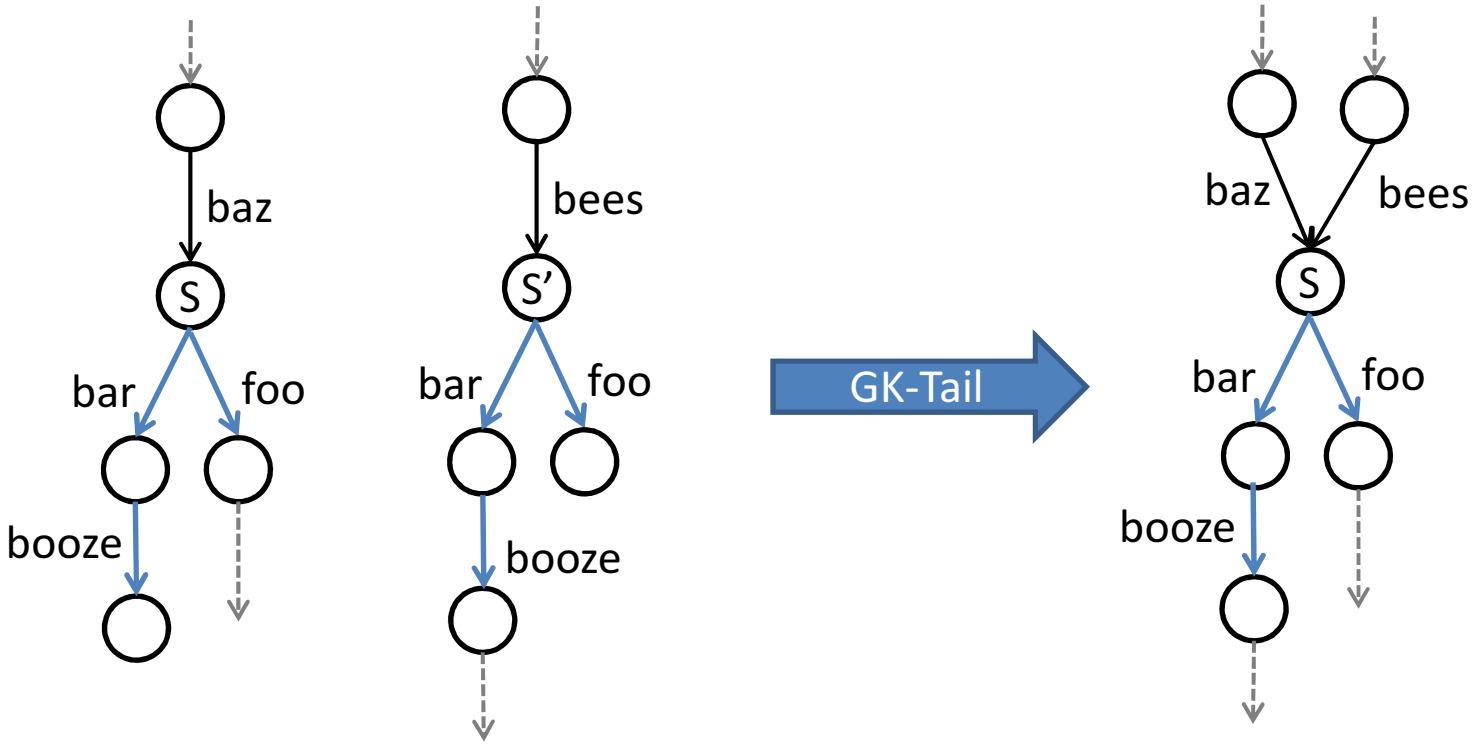
Coarsening Algorithm: GK-Tail [1]

- Given a state in the graph we define its k-Tail as the subgraph made by the frontier k edges out
- The 2-tail of the state S in the graph below is in blue:



Coarsening Algorithm: GK-Tail [2]

- GK-Tail algorithm idea: if two states have *equivalent* k-Tails, we merge them.
- Repeat until there are no more states to merge.

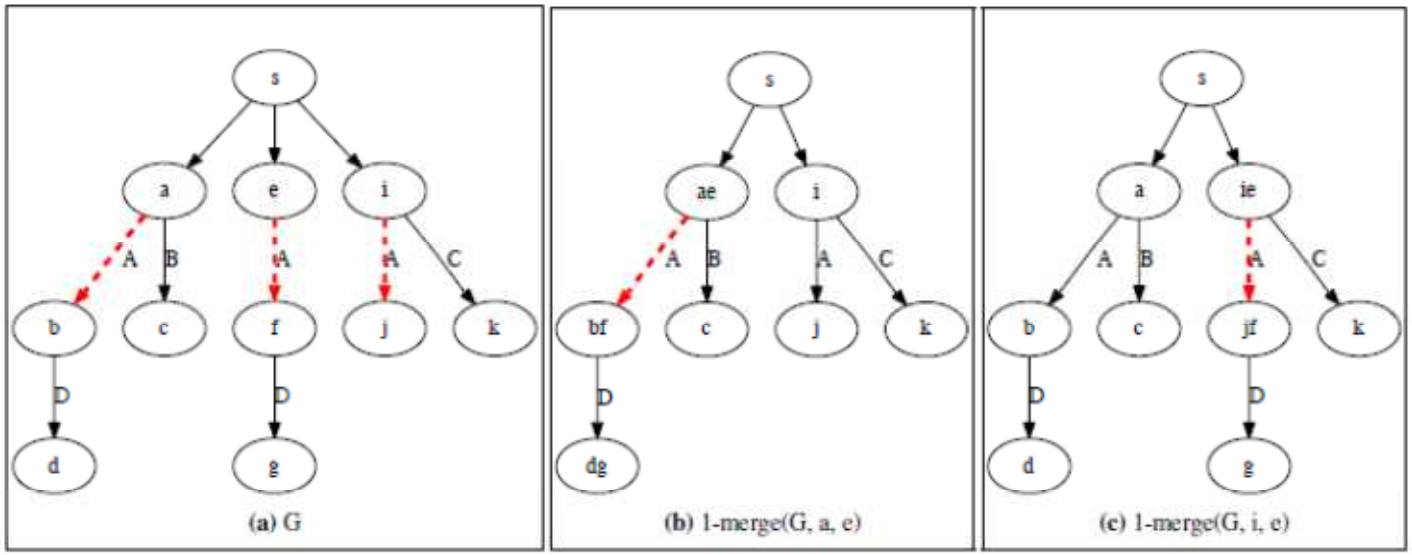


GK-Tail: What Is Equivalent?

- Strict notion of equivalence: k-Tails exactly equal.
  - Advantage: True equivalence relation – algorithm result is deterministic.
  - Disadvantage: Often too strict for meaningful data reduction.
- Subsumption equivalence: k-Tail of S' contained within the k-Tail of S
  - Advantage: Reduces representation much more efficiently
  - Disadvantage: Not an equivalence relation – output of algorithm is a local optimum, may not be global.

GK-Tail: Subsumption Nondeterminsim

- Multiple merge candidates for *e*: *a* and *i* both work.
- After merge, algorithm produces distinct results - both local optima.



- Result of 1-merge on (a) can be either (b) or (c)
- If we chose to merge at *e* and *a* we get (b)
- If we chose to merge at *e* and *i* we get (c)

GK-Tail: When can we merge? [1]

- The notion of k-Tails does not capture all desired properties of the trace.
  - Example: Messages more than k apart are temporally related.
- One solution: Make k bigger
  - Problem: If we make k too big summarization will be weak
  - Another problem: There could be other relationships.
- How do we chose the right value of k?
  - In general, this is hard – encodes domain knowledge.

GK-Tail: When can we merge? [2]

- Our Solution: Compute and encode *invariants* about the trace
- We call a graph *valid* if it satisfies the invariants of the original trace.
- When performing the GK-Tail algorithm make sure that each merge produces a valid graph.
- GK-Tail now repeats until there are no more states to merge with matching k-Tails OR no ones that will lead to a valid graph.

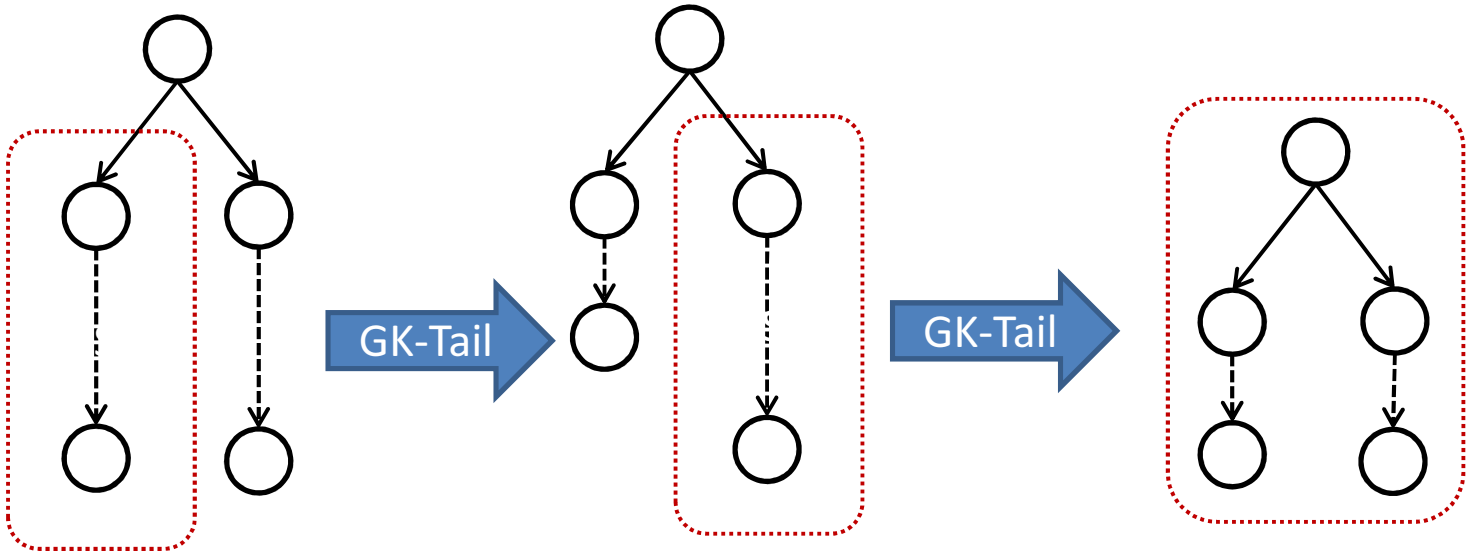
GK-Tail Performance

- Trivial implementation is $O(n^3)$
  - For current graph have to consider at most merging every state to every other state (n^2)
  - Each merge removes a state – can do at most n merges.

- Can we do better?

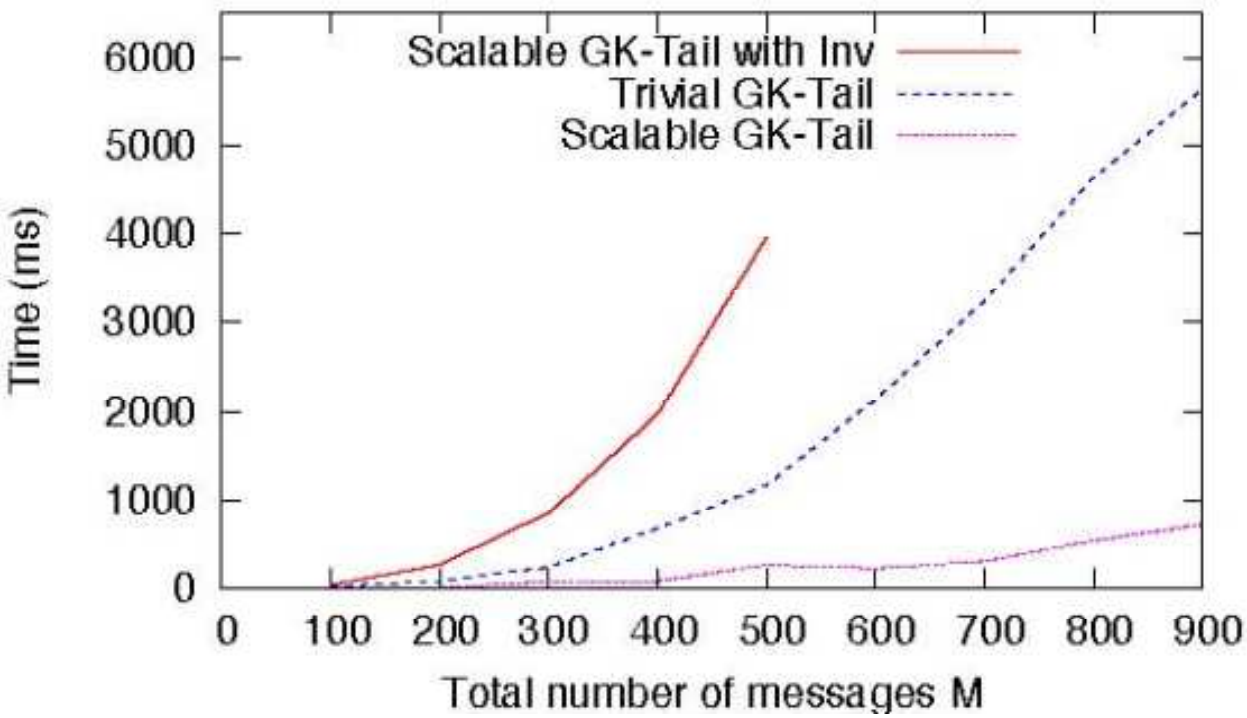
Scalable GK-Tail

- Key Idea: Divide and Conquer.
- We can perform GK-Tail on subgraphs and then perform GK-Tail on the resulting graph if we pick the subgraphs carefully.



- Theoretical complexity is the same – $O(n^3)$
  - Worst case: no chosen sub-graph is mergeable.
- In practice, the expectation is that each sub-graph will have approximately the same distribution of messages as the complete graph.

GK-Tail Performance



- Input is a repeated sequence of messages
- GK-Tail performance is polynomial
- Scalable GK-Tail performance is roughly linear
- Still slow when we use Invariants.