Cloud-based Software Verification

Srđan Krstić
Politecnico di Milano

Joint work with Carlo Ghezzi, Domenico Bianculli, Marcello Bersani and Pierluigi San Pietro

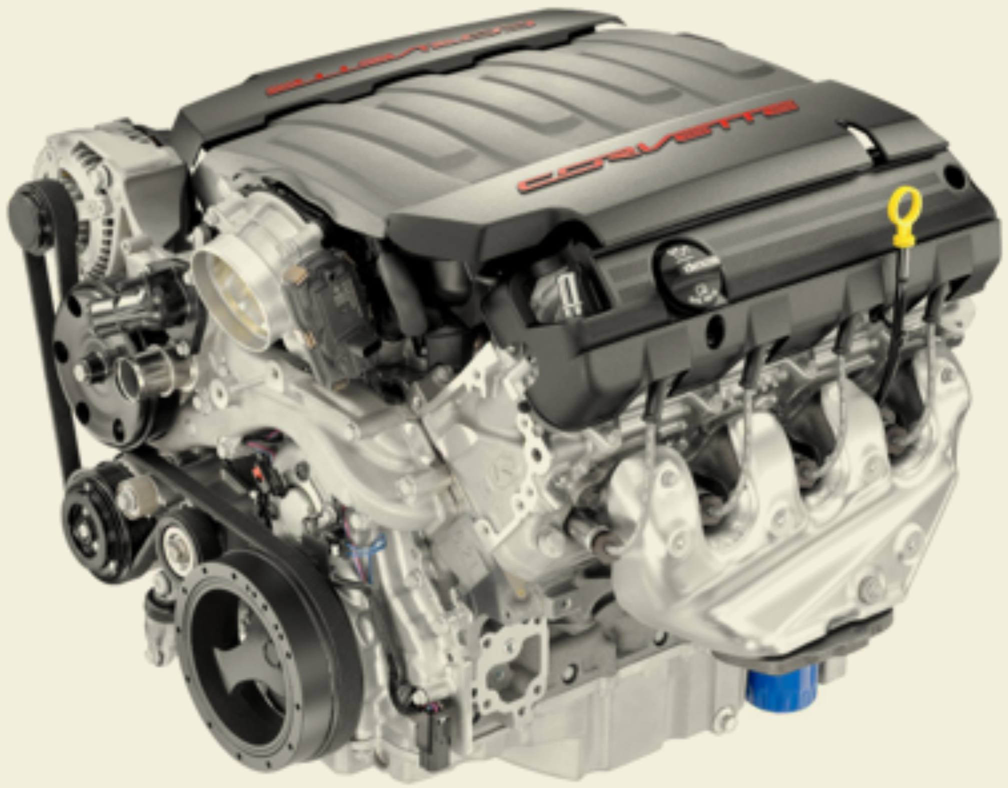
Cloud-based Software Verification

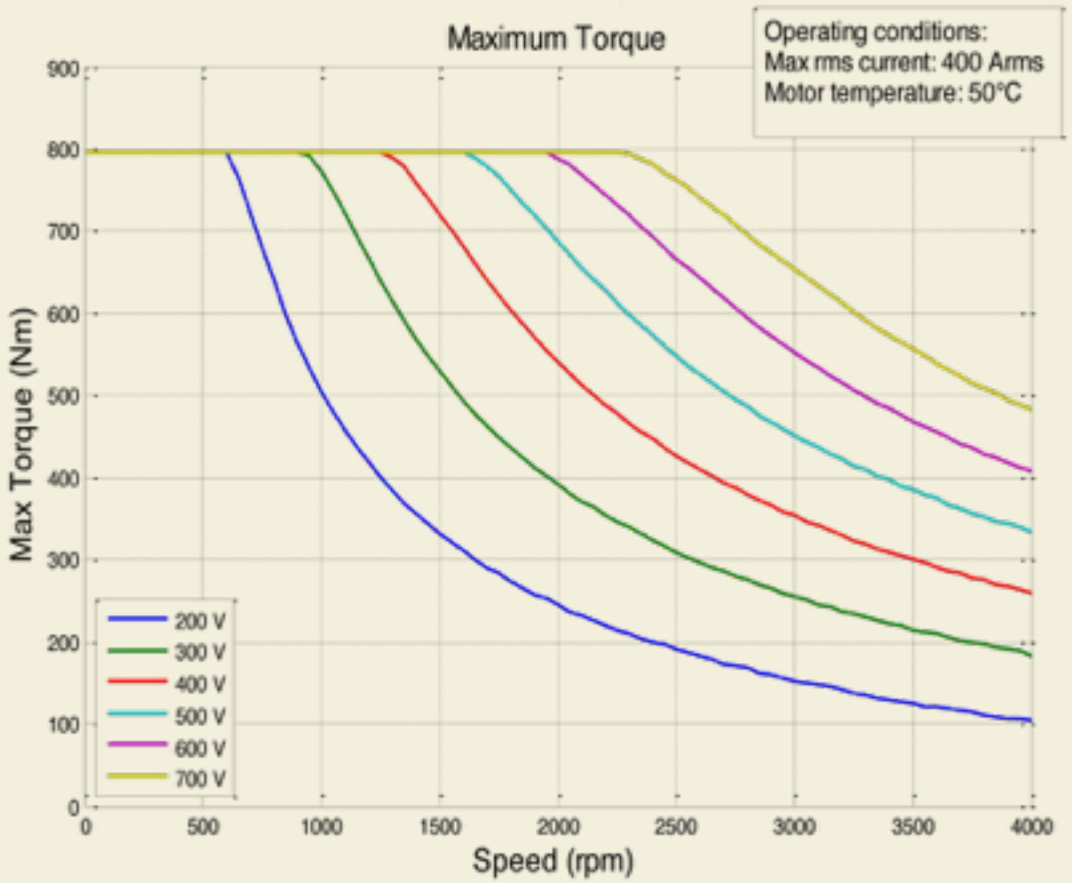
Srđan Krstić
Politecnico di Milano

Joint work with Carlo Ghezzi, Domenico Bianculli, Marcello Bersani and Pierluigi San Pietro

Software Verification

System ⊨ Specification

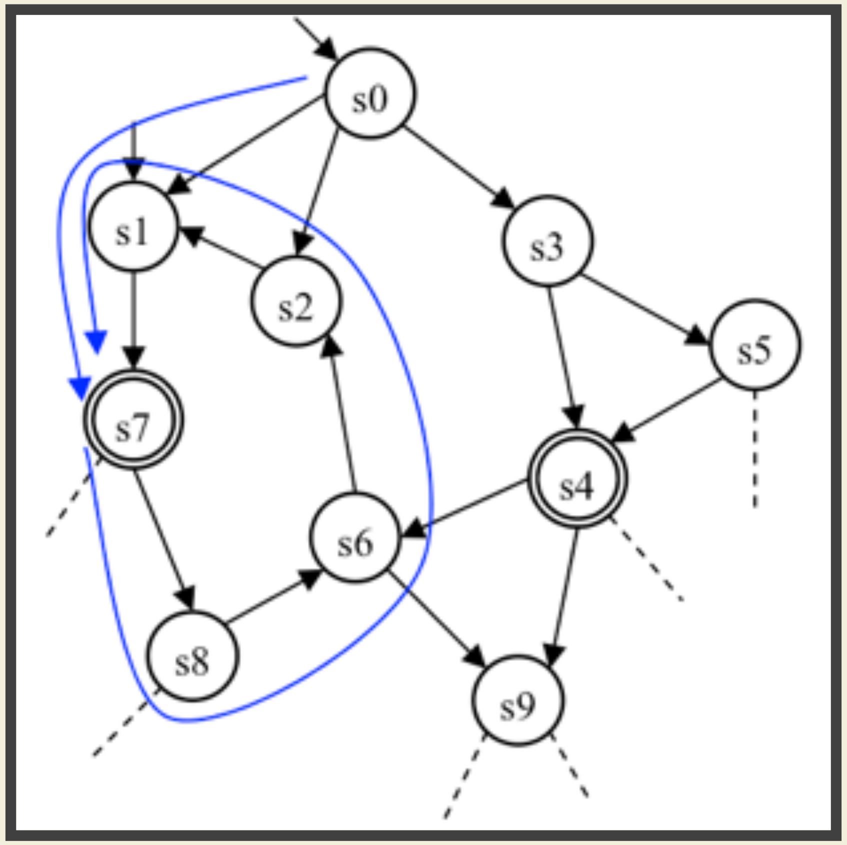




Theorem proving

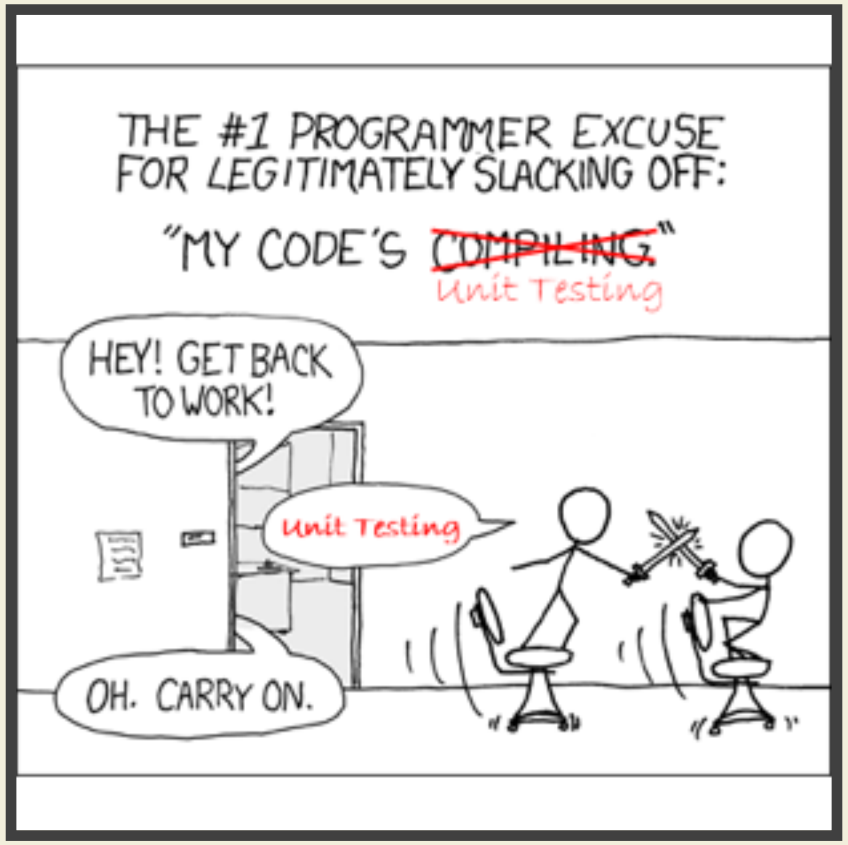
- High guarantees
- Undecidable
- Requires expert knowledge

Model checking



- High guarantees
- State explosion problem
- Finite models, mostly

Testing



- Low guarantees
- Scalable
- Simple

Modern Software

Dynamic behavior

Component-based

Decentralized

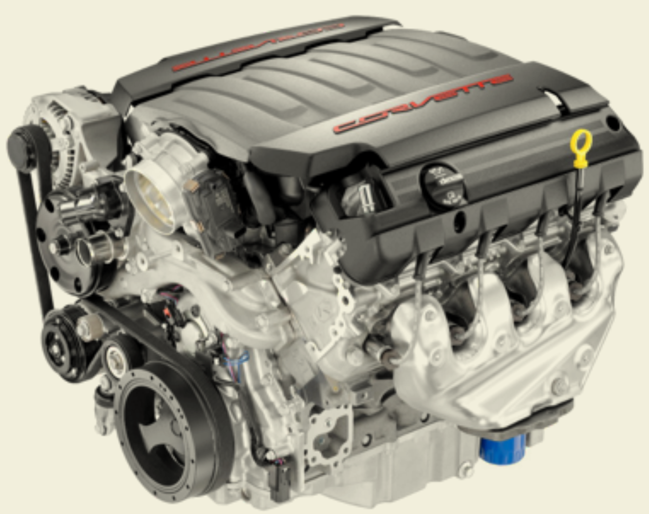
Dynamic Interaction

Third-party functionality

Monitoring

Instrumentation

Property



System

Execution trace

Offline Monitoring

- Trace/History checking

Online Monitoring

- Runtime verification

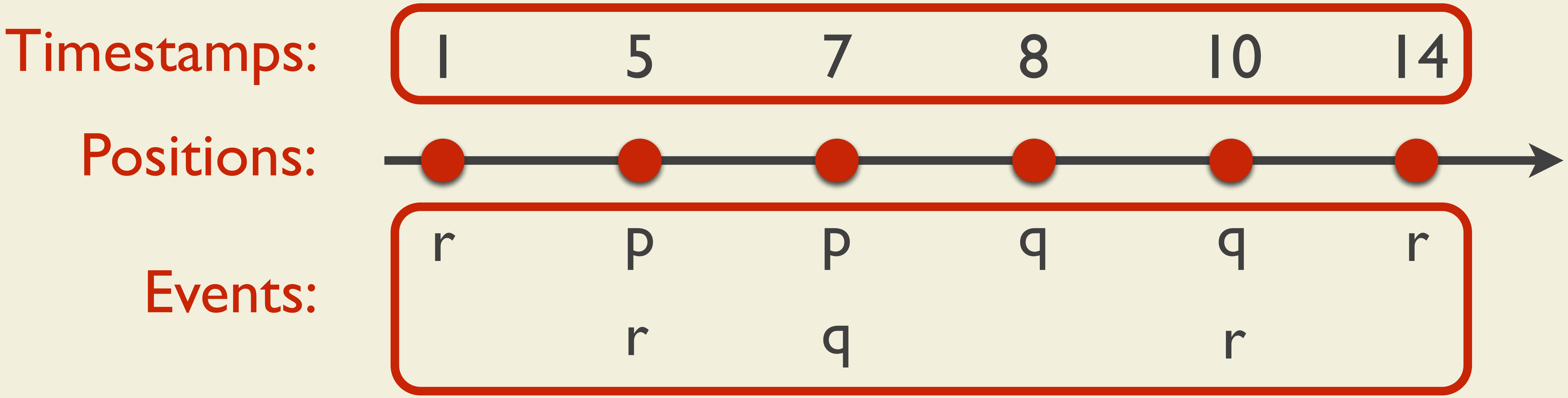
Offline vs Online Monitoring

- Execution overhead
- Scanning direction
- Preprocessing
- Specification language semantics

<1s

```
2015-05-15 11:13:04,155 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] recei
2015-05-15 11:13:04,155 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] hand
2015-05-15 11:13:05,155 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] recei
2015-05-15 11:13:05,155 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] hand
2015-05-15 11:13:06,155 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] recei
2015-05-15 11:13:06,155 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] hand
2015-05-15 11:13:06,693 DEBUG org.apache.hadoop.ipc.Client: The ping interval is 60000 ms.
2015-05-15 11:13:06,694 DEBUG org.apache.hadoop.ipc.Client: Connecting to localhost/127.0.0.1:9000
2015-05-15 11:13:06,695 DEBUG org.apache.hadoop.ipc.Client: IPC Client (1198532806) connection to localhost
2015-05-15 11:13:06,695 DEBUG org.apache.hadoop.ipc.Client: IPC Client (1198532806) connection to localhost
2015-05-15 11:13:06,696 DEBUG org.apache.hadoop.ipc.Client: IPC Client (1198532806) connection to localhost
2015-05-15 11:13:06,697 DEBUG org.apache.hadoop.ipc.ProtobufRpcEngine: Call: renewLease took 4ms
2015-05-15 11:13:06,697 DEBUG org.apache.hadoop.hdfs.LeaseRenewer: Lease renewed for client DFSClient_NO
2015-05-15 11:13:06,697 DEBUG org.apache.hadoop.hdfs.LeaseRenewer: Lease renewer daemon for [DFSClient_
2015-05-15 11:13:06,903 DEBUG org.apache.spark.util.Utils: Shutdown hook called
2015-05-15 11:13:06,903 DEBUG org.apache.spark.storage.DiskBlockManager: Shutdown hook called
2015-05-15 11:13:06,908 ERROR org.apache.hadoop.hdfs.DFSClient: Failed to close inode 16409
ava.io.IOException: All datanodes 127.0.0.1:50010 are bad. Aborting...
	at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.setupPipelineForAppendOrRecovery(DFSOutput
	at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.processDatanodeError(DFSOutputStream.java:1
	at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:548)
2015-05-15 11:13:06,912 DEBUG org.apache.hadoop.ipc.Client: stopping client from cache: org.apache.hadoop.i
2015-05-15 11:13:06,913 DEBUG org.apache.hadoop.ipc.Client: removing client from cache: org.apache.hadoop.
2015-05-15 11:13:06,913 DEBUG org.apache.hadoop.ipc.Client: stopping actual client because no more referenc
2015-05-15 11:13:06,913 DEBUG org.apache.hadoop.ipc.Client: Stopping client
2015-05-15 11:13:06,914 DEBUG org.apache.hadoop.ipc.Client: IPC Client (1198532806) connection to localhost
2015-05-15 11:13:06,914 DEBUG org.apache.hadoop.ipc.Client: IPC Client (1198532806) connection to localhost
2015-05-15 11:13:06,931 DEBUG org.apache.spark.repl.SparkILoop$SparkILoopInterpreter: parse("      sc.stop()
2015-05-15 11:13:07,153 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] recei
2015-05-15 11:13:07,153 DEBUG org.apache.spark.scheduler.cluster.SparkDeploySchedulerBackend: [actor] hand
```

Formalizing Execution Traces: Timed words



Example Properties

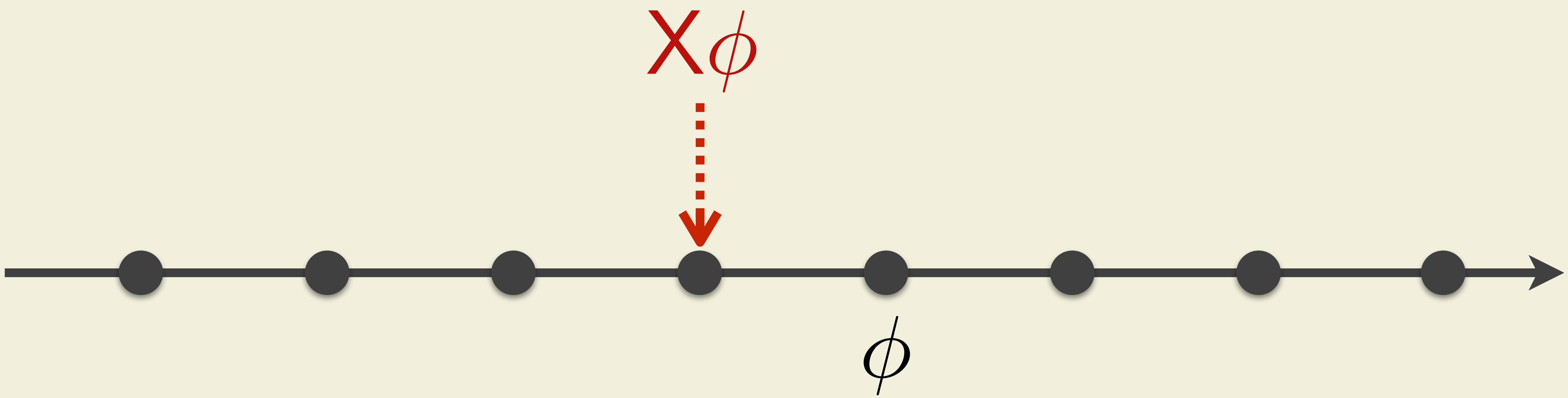
- P1: “At any time, the user must be logged in before executing a withdraw operation”;
- P2: “ At any time, the user must be logged out by the system 5 minutes after logging in or after his last withdraw operation”;
- P3: “The number of withdrawal operations performed within 10 minutes before customer logs out must be less than or equal to 3, at any time”;
- P4: “It is always the case that the total amount of money withdrawn by any user in the last 30 days does not exceed 5000 EUR, except if the user has previously received a higher credit limit”.

Formalizing Properties: Temporal Logic

- Linear Temporal Logic
- Metric Temporal Logic
- Metric Temporal Logic with Aggregations
- Metric First-Order Temporal Logic

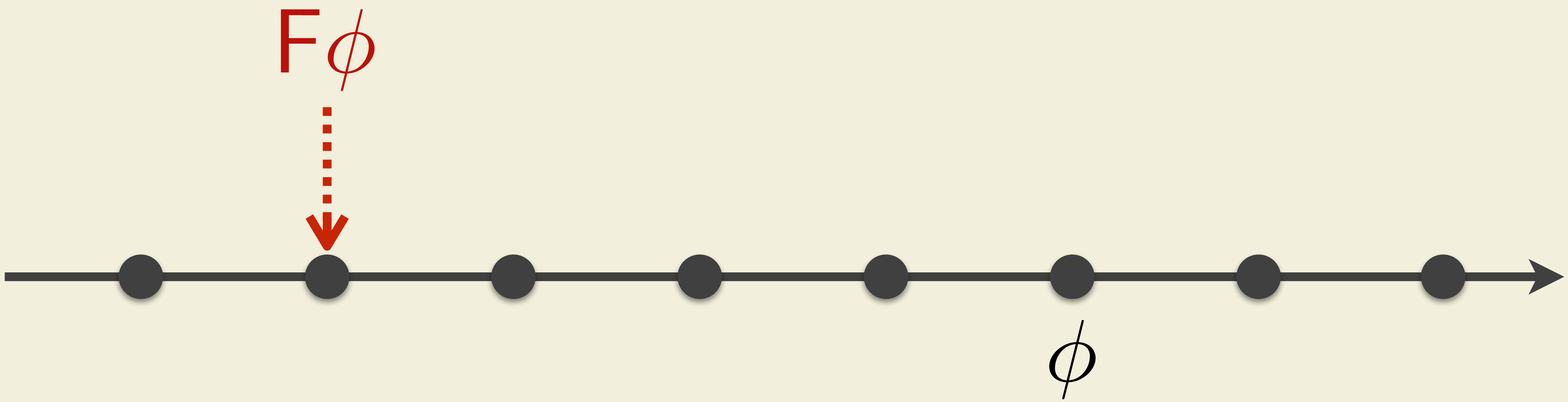
Linear Temporal Logic (LTL)

Next



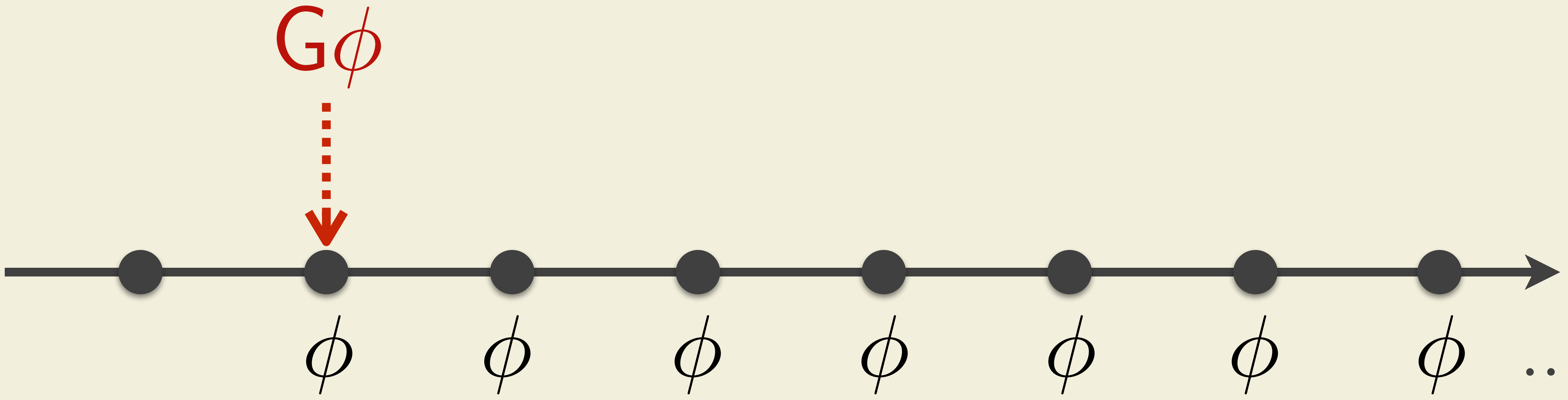
Linear Temporal Logic (LTL)

Eventually

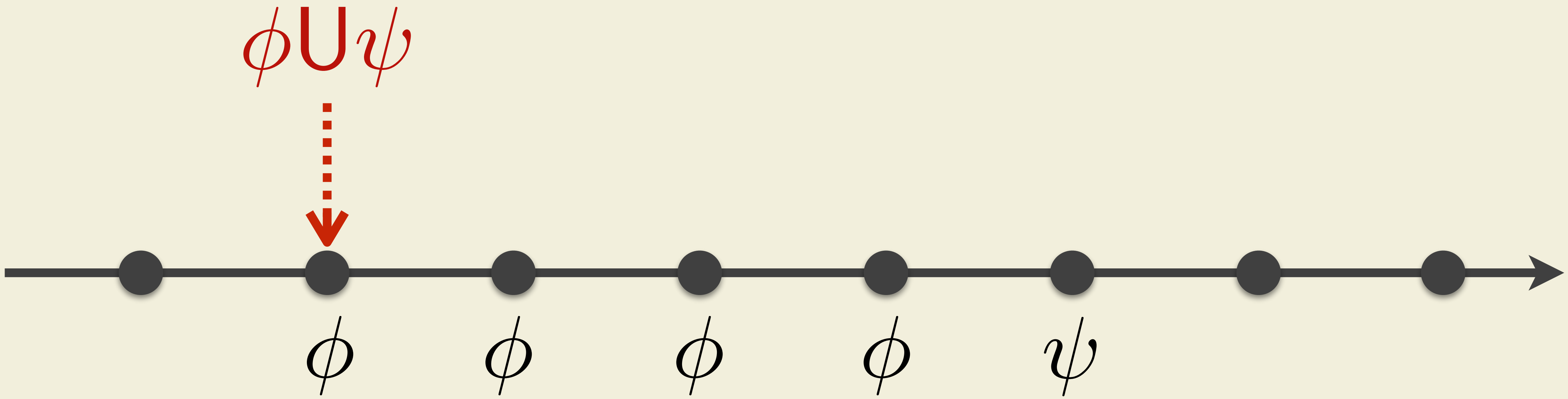


Linear Temporal Logic (LTL)

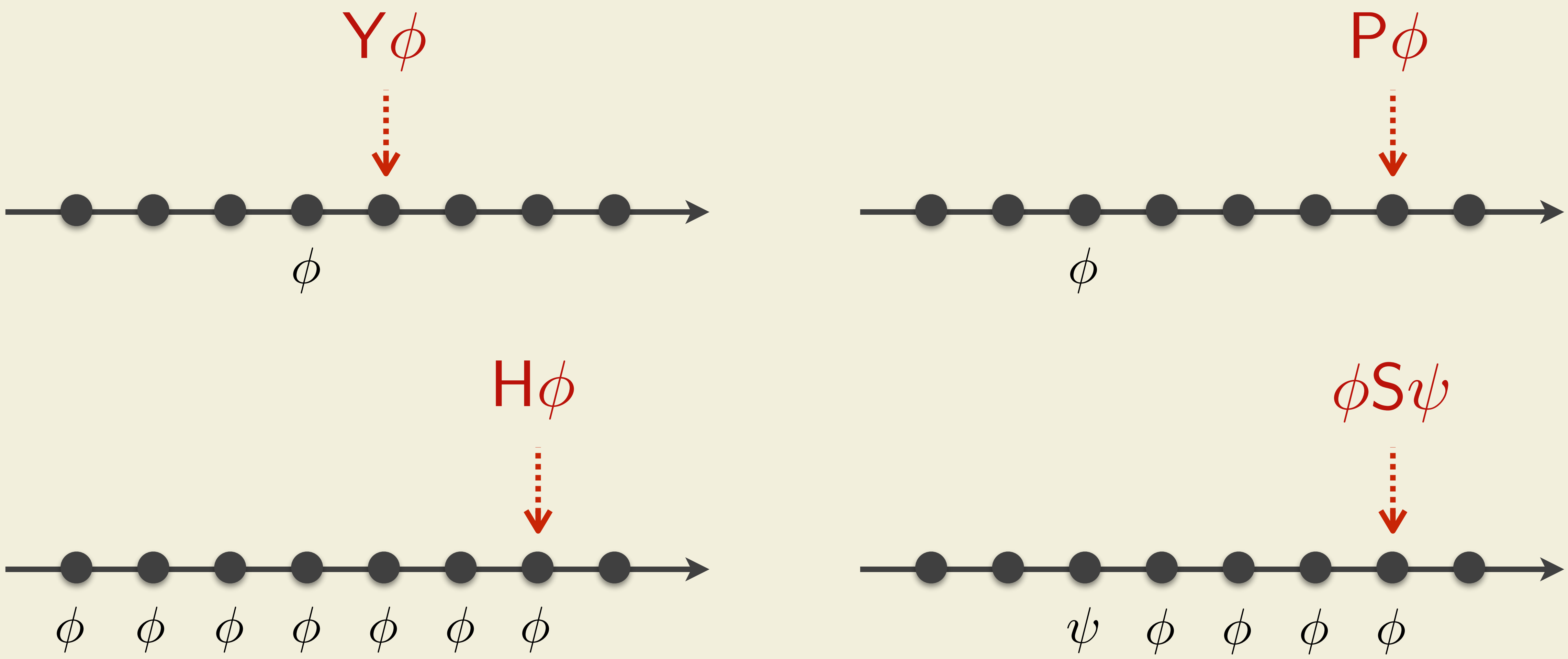
Globally



Linear Temporal Logic (LTL)
Until

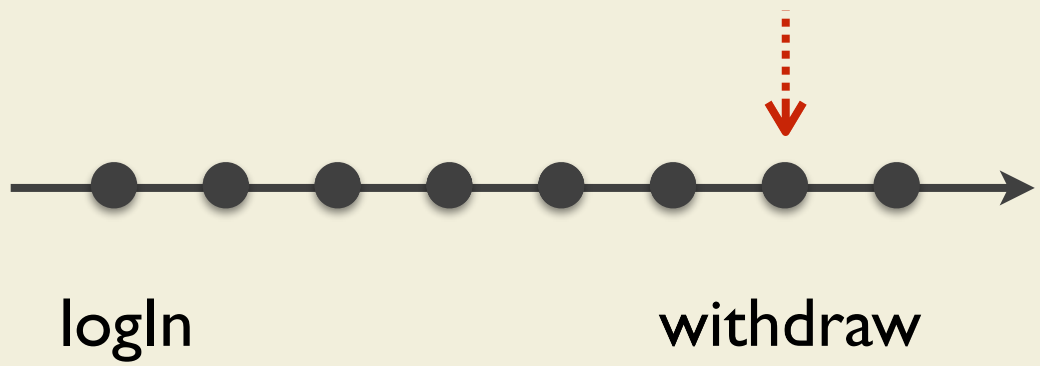


Linear Temporal Logic (LTL)
Past operators



P1: "**At any time,** the user must be **logged in** **before** executing a **withdraw** operation"

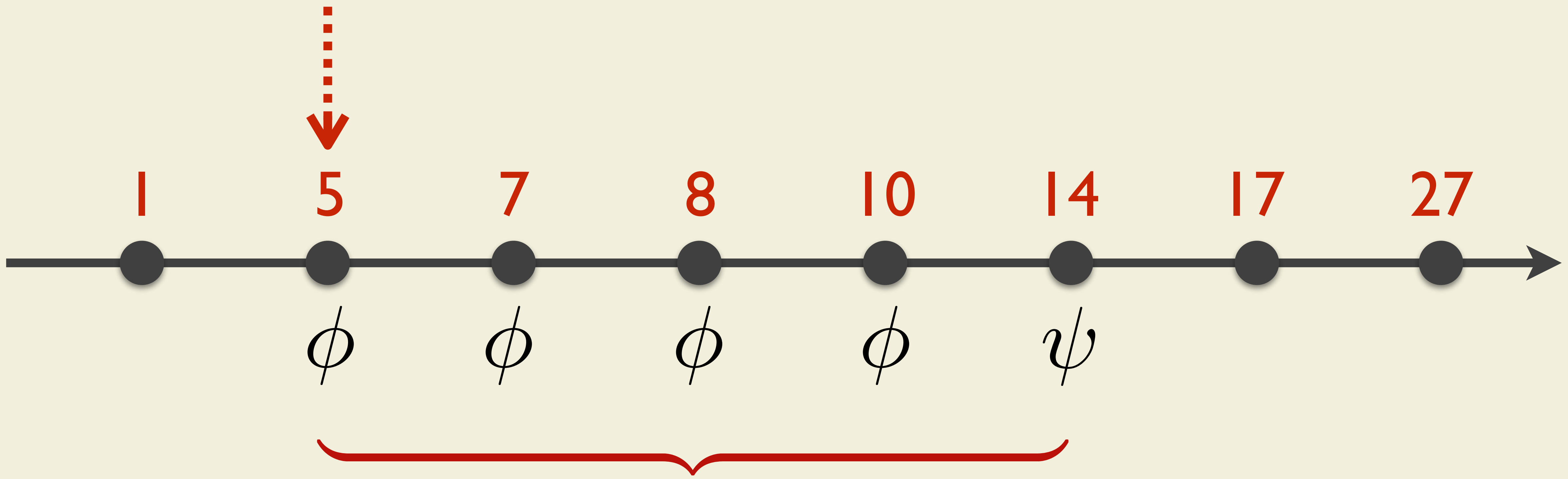
$$\mathbf{G}(\mathit{withdraw} \rightarrow \neg \mathit{logOff}\ \mathbf{S}\ \mathit{logIn})$$



P2: “**At any time**, the user must be **logged out** by the system **5 minutes after logging in** or after his last **withdraw** operation”

Metric Temporal Logic (MTL)

$\phi \mathsf{U}_{(4,12]} \psi$

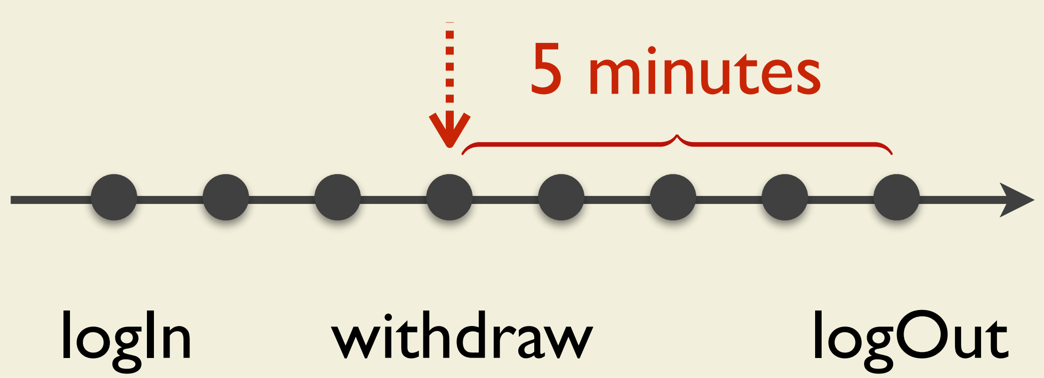


$4 < 14 - 5 \leq 12$

P2: "**At any time**, the user must be **logged out** by the system **5 minutes after logging in** or after his last **withdraw** operation"

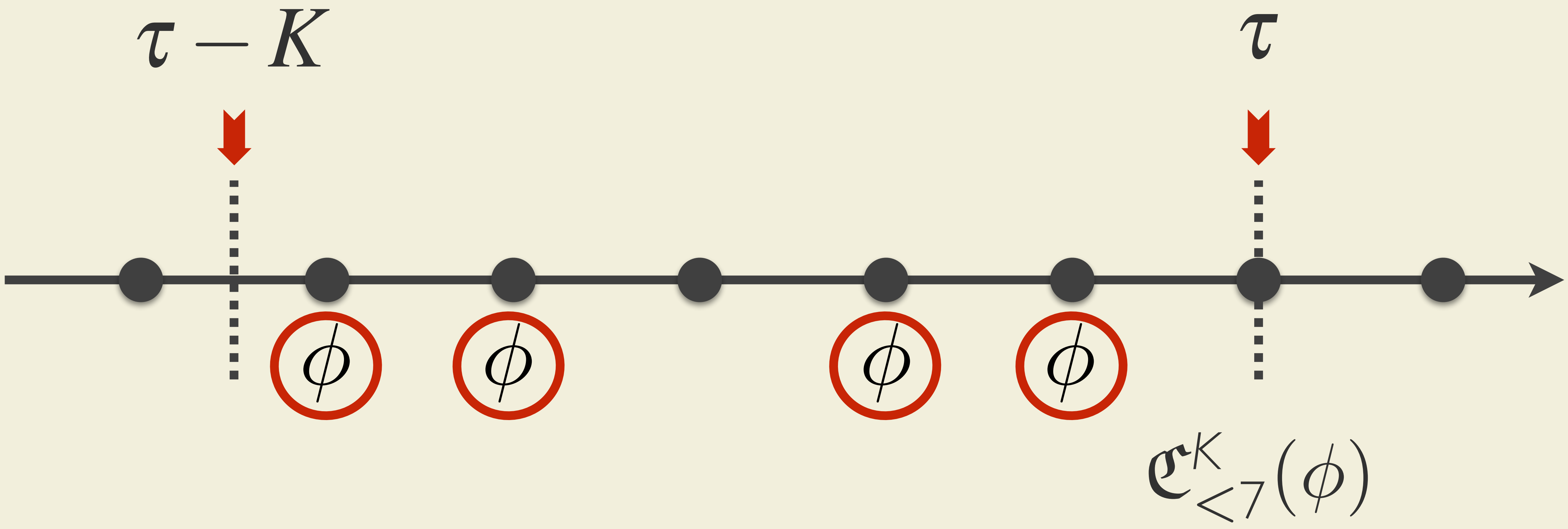
$$G(\mathit{logIn} \rightarrow F_{[300,300]}\ \mathit{logOut}\ \underline{\vee}\ F_{(0,300]}\mathit{withdraw})$$

$$G(\mathit{withdraw} \wedge G_{(0,300]}\neg \mathit{withdraw} \rightarrow F_{[300,300]}\ \mathit{logOut})$$



P3: “The number of **withdrawal** operations performed within **10 minutes** before customer **logs out is less than or equal to 3,** at any time”

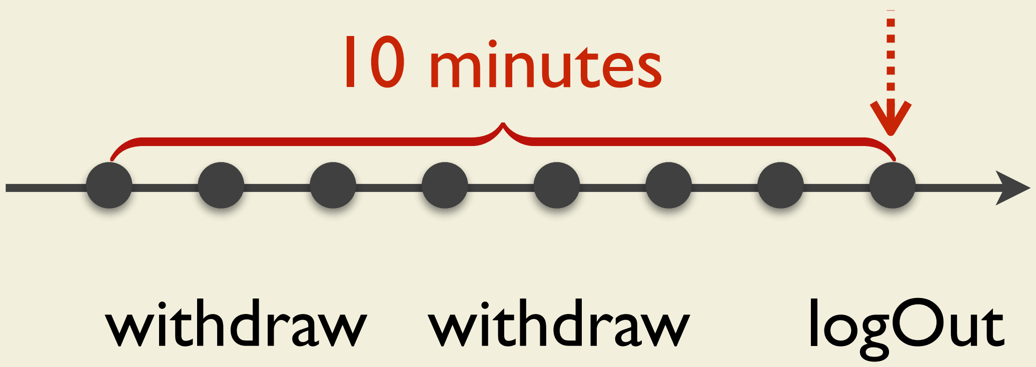
MTL with Aggregations



$$4 < 7$$

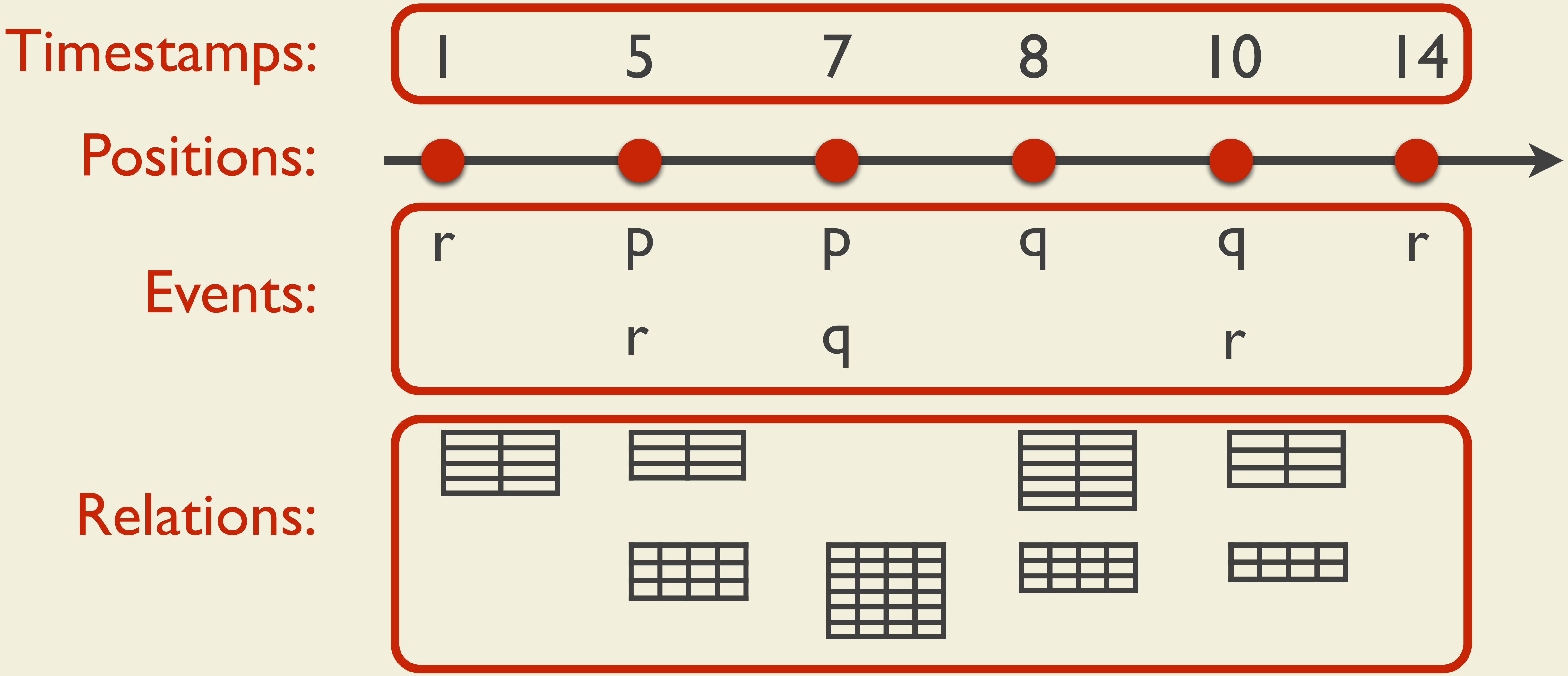
P3: “The number of **withdrawal** operations performed within **10 minutes** before customer **logs out is less than or equal to 3,** at any time”

$$G(logOut \rightarrow \mathfrak{C}^{600}_{\leq 3}(withdraw))$$

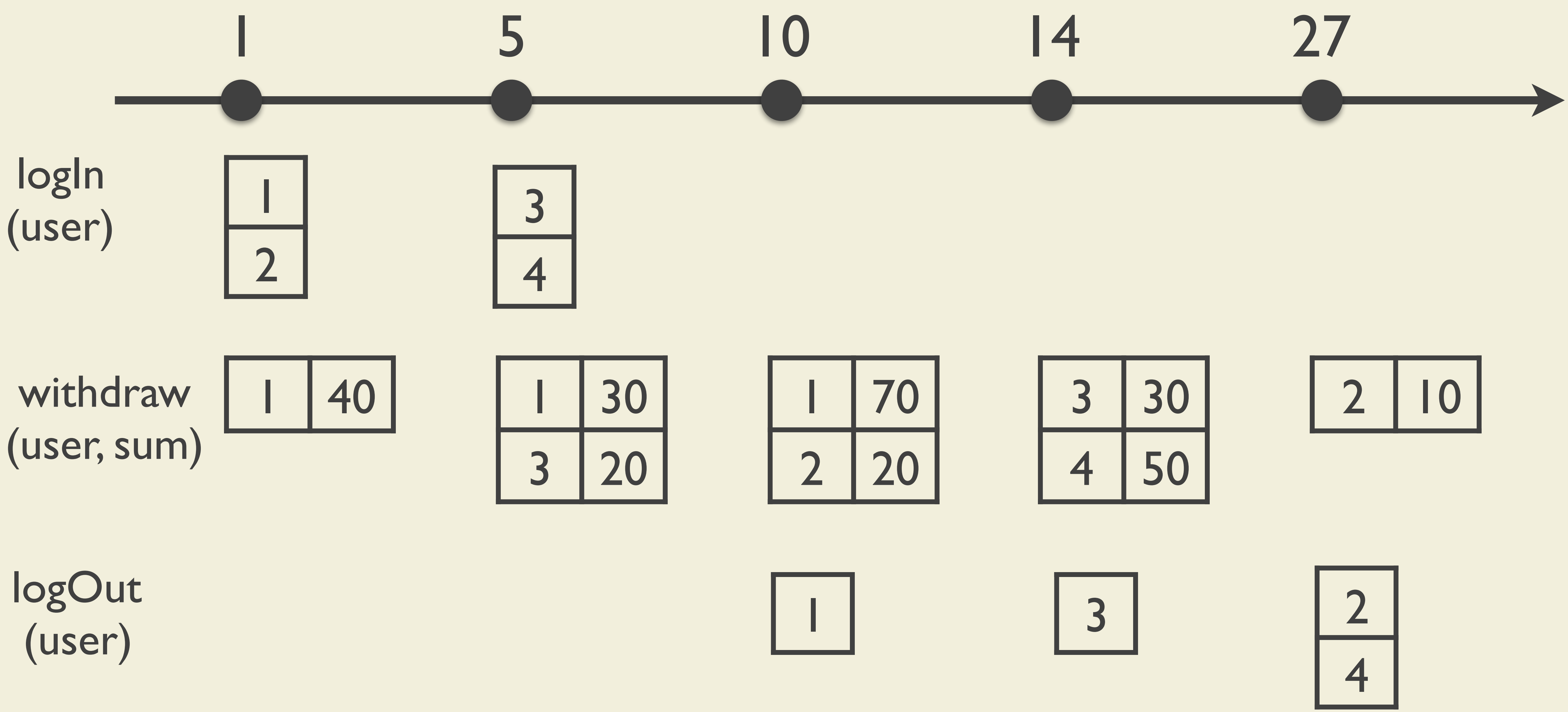


P4: “The amounts of money **withdrawn** by **any user** does not exceed 5000 EUR, except if the user has **previously received a higher credit limit**”.

Timed Word with Relations



Timed Word with Relations



P4: “The amounts of money **withdrawn** by **any user** does not exceed 5000 EUR, except if the user has **previously received a higher credit limit**”.

$$G(\forall u.\forall a.(withdraw(u, a) \rightarrow a \leq 5000 \vee \mathsf{P}\ creditLimit(u)))$$

Monitoring Algorithms

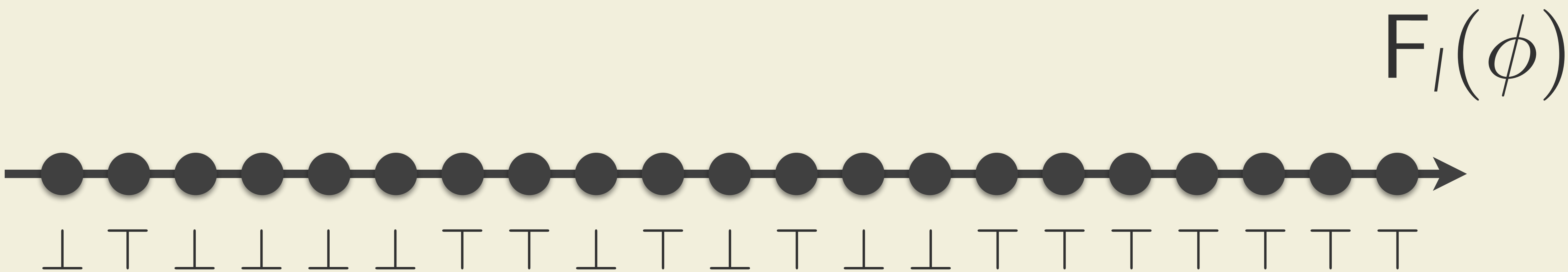
LTL Monitoring Algorithm

Eventually operator

Arbitrary nesting

Reverse scanning

Incremental verdict



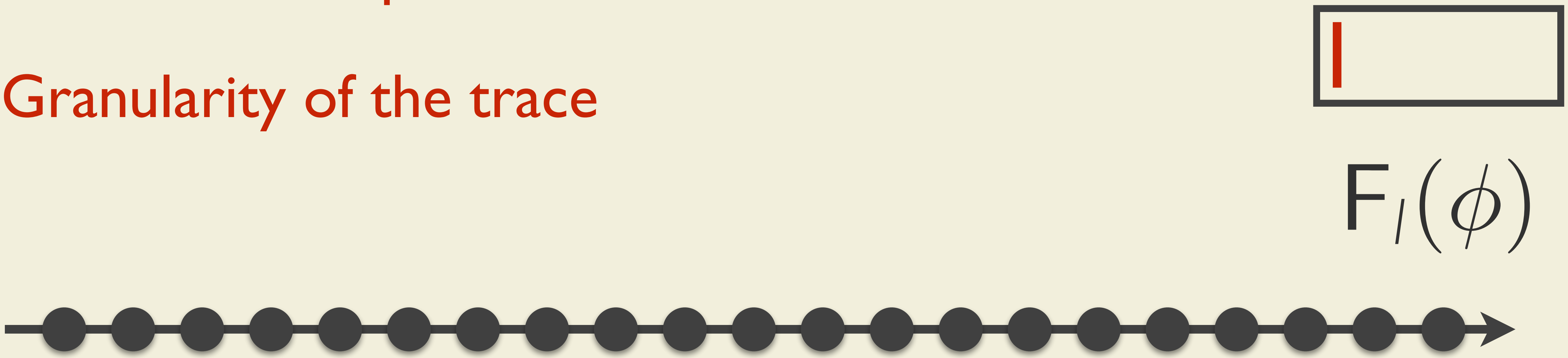
MTL Monitoring Algorithm

Metric Eventually operator

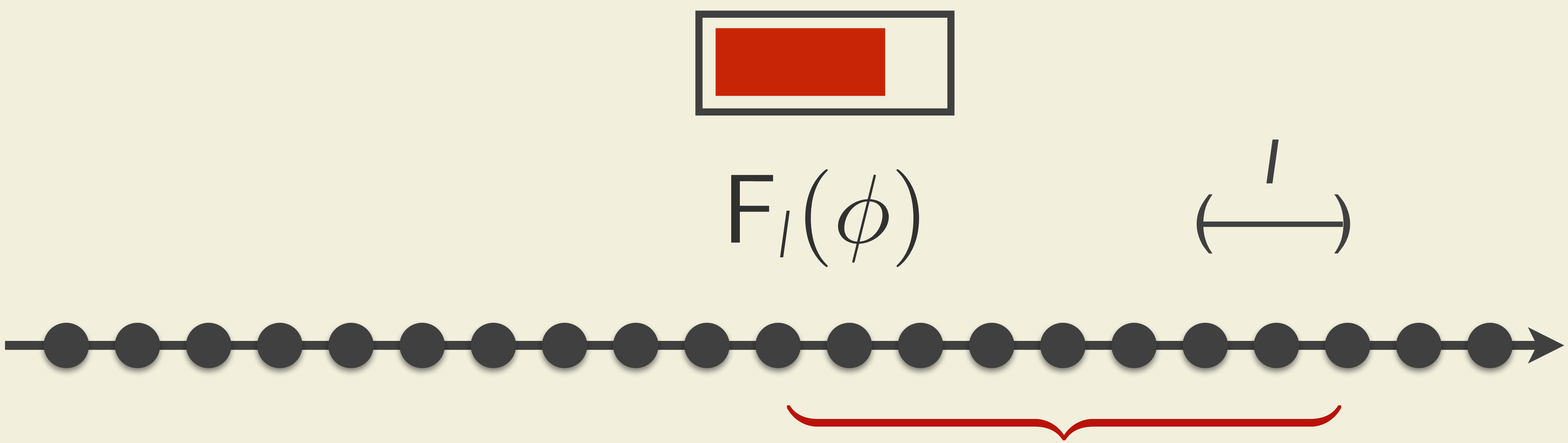
Queue-like data structure

Size of the temporal interval

Granularity of the trace



MTL Monitoring Algorithm



RV 2011

Algorithms for Monitoring Real-time Properties*°

David Basin, Felix Klaedtke, and Eugen Zălinescu

Computer Science Department, ETH Zurich, Switzerland

Abstract. We present and analyze monitoring algorithms for a safety fragment of metric temporal logics, which differ in their underlying time model. The time models considered have either dense or discrete time domains and are point-based or interval-based. Our analysis reveals differences and similarities between the time models for monitoring and highlights key concepts underlying our and prior monitoring algorithms.

1 Introduction

Real-time logics [2] allow us to specify system properties involving timing constraints, e.g., every request must be followed within 10 seconds by a grant. Such specifications are useful when designing, developing, and verifying systems with hard real-time requirements. They also have applications in runtime verification, where monitors generated from specifications are used to check the correctness of system behavior at runtime [10]. Various monitoring algorithms for real-time logics have been developed [4,5,7,12,14,15,17,20] based on different time models. These time models can be characterized by two independent aspects. First, a

Distributed Monitoring

Wikipedia Page Traffic Statistics Dataset

Contains 7 months of hourly page view statistics for all articles in Wikipedia

Size: 320 GB

Created On: June 9, 2009

DARPA Scalable Network Monitoring (SNM) Program Traffic

Contains 9 days of captured network traffic

Size: 7083.4 TB

Created On: November 12, 2009

Edward Snowden
@Snowden

Follow

I forgot to turn off notifications. Twitter sent me an email for each:

Follow
Favorite
Retweet
DM

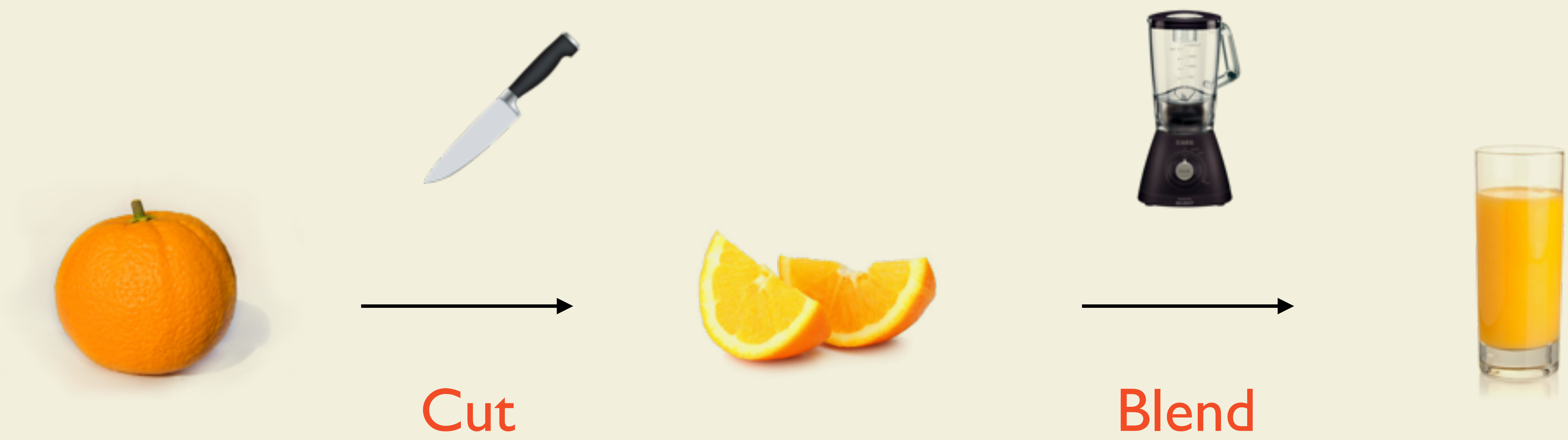
47 gigs of notifications. #lessonlearned

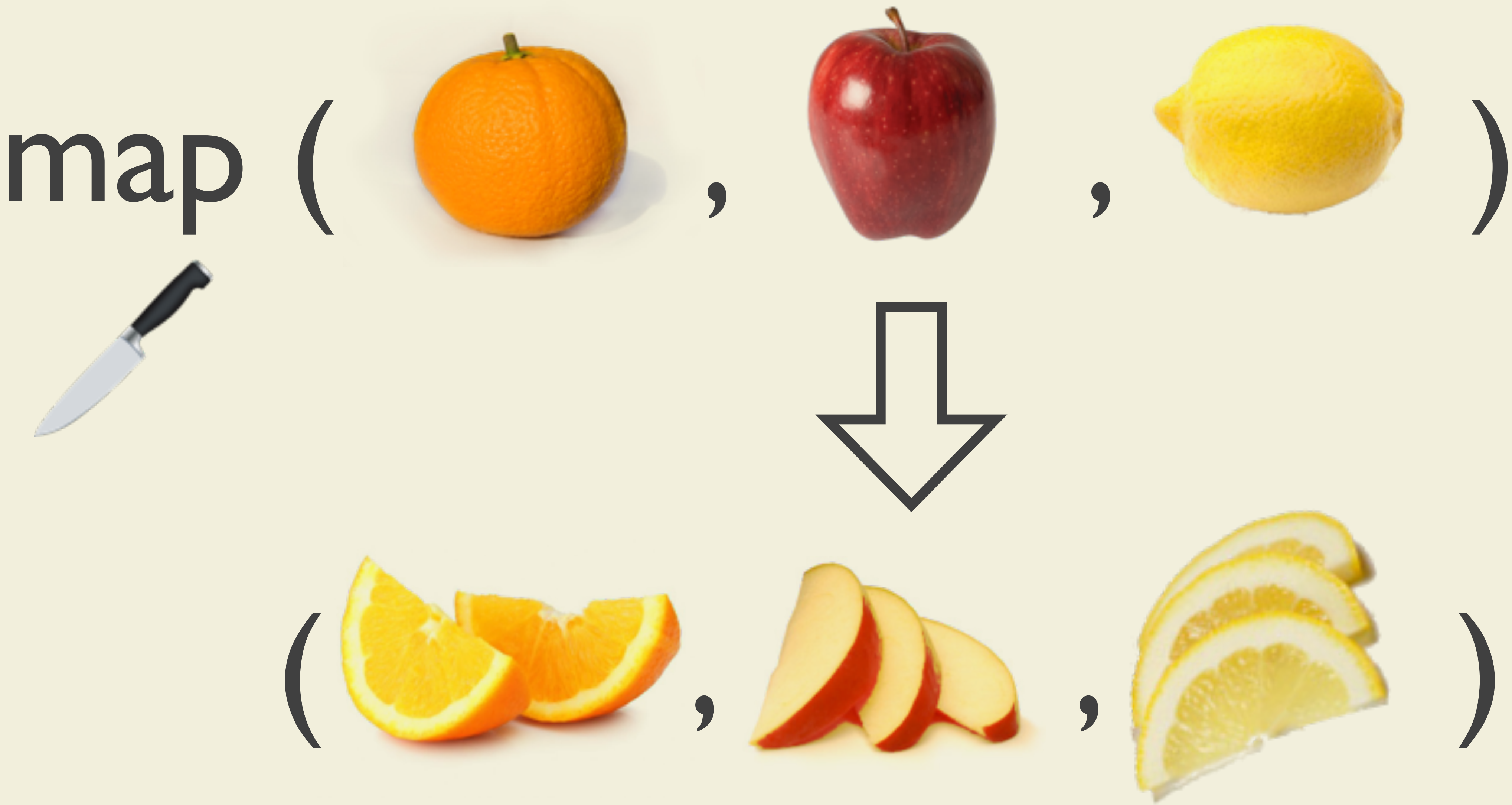
Main Challenge

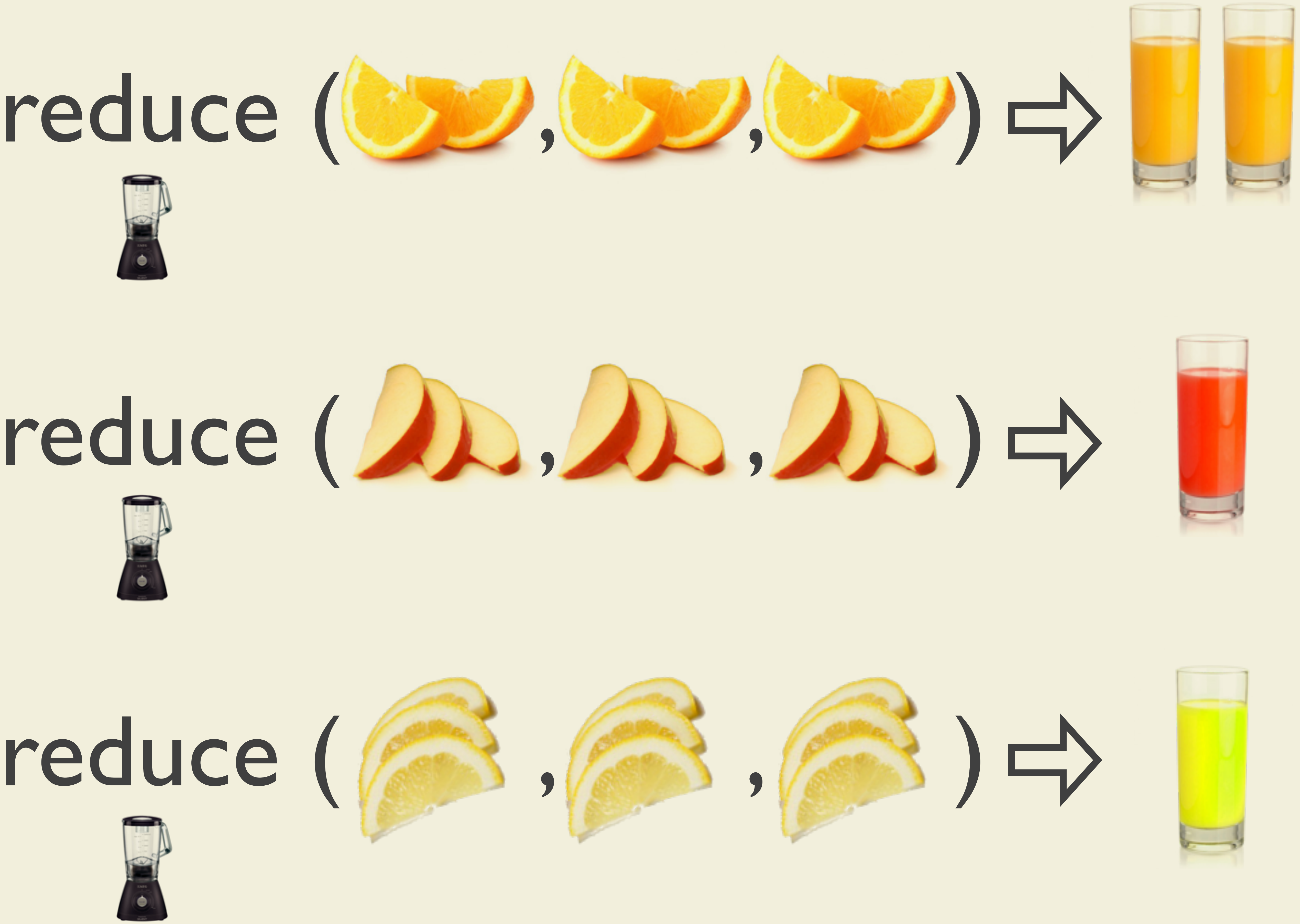
Large traces that cannot be collected, stored and processed on a single machine.

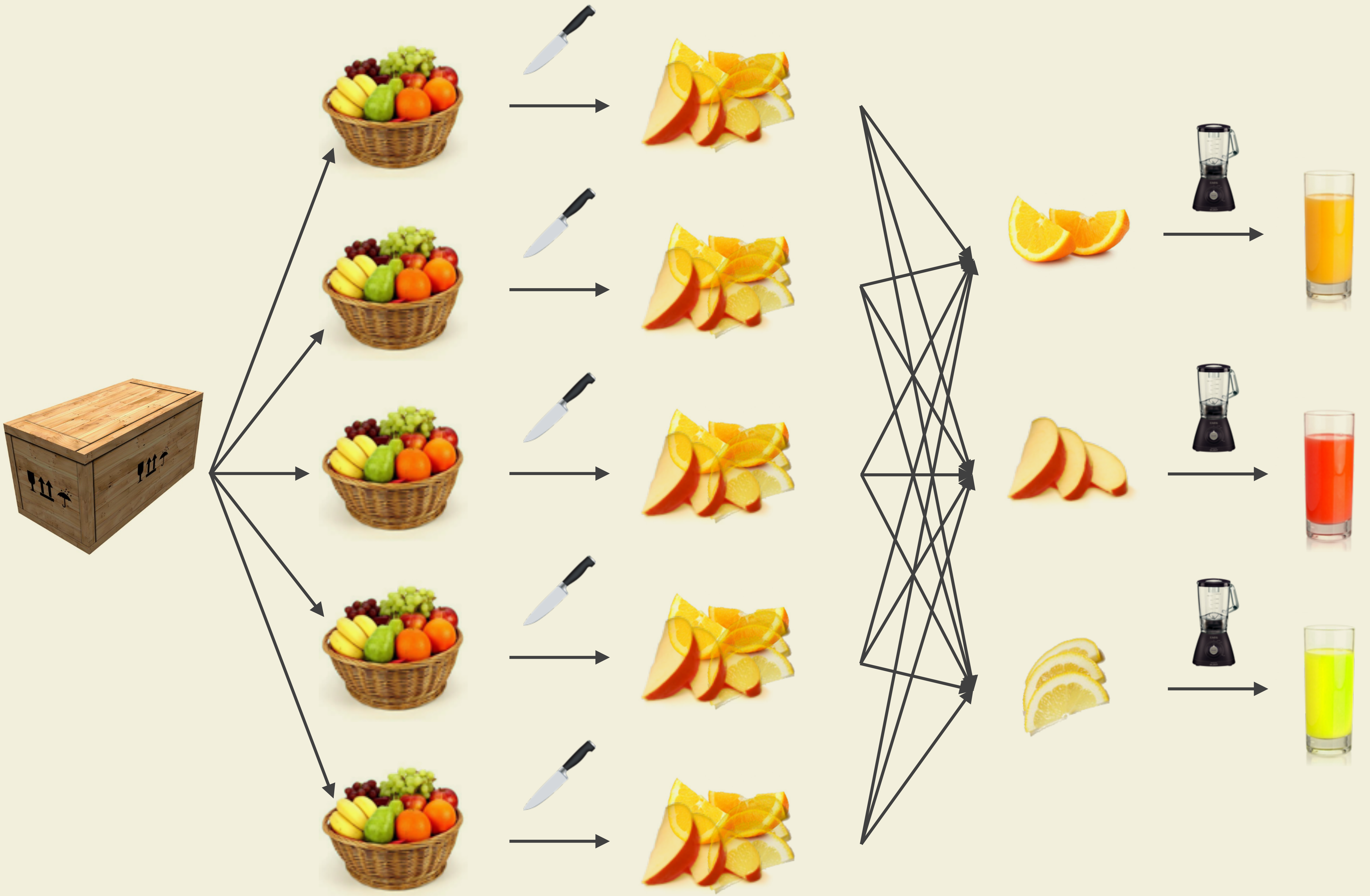
Solution: Distributed Monitoring using MapReduce

MapReduce




map


reduce (, ,)
reduce (, ,)
reduce (, ,)



How to use MapReduce for Monitoring?

Parallelization Strategies

Splitting the formula (general, limited parallelization)

- Parallel sub-formula processing
- Temporal operator decomposition

Splitting the trace (heuristic, high parallelization)

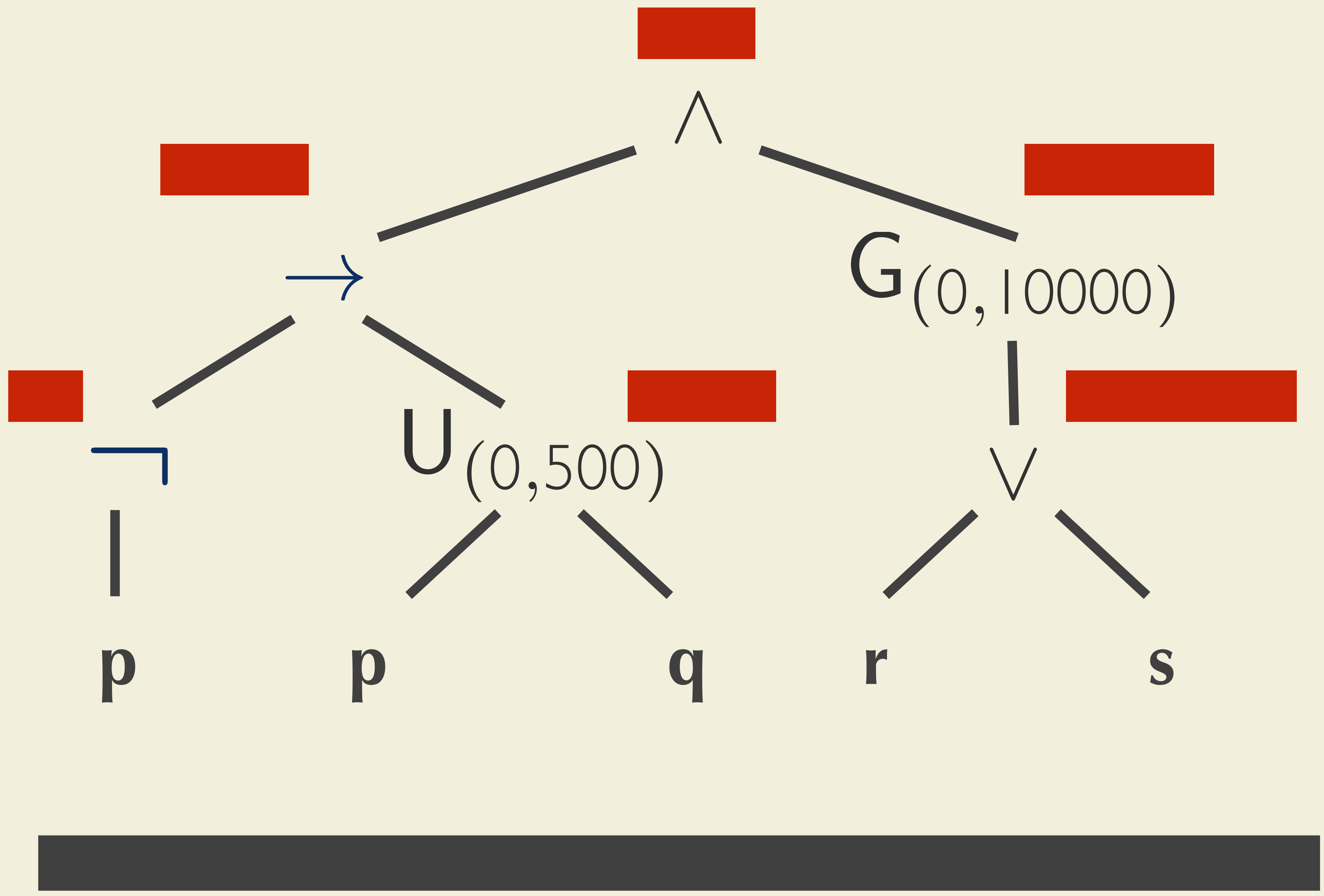
- Time-wise trace splitting
- Data-wise trace splitting

Parallel sub-formula processing

First position?

Iterative Trace Subsets

Formula Syntax Tree



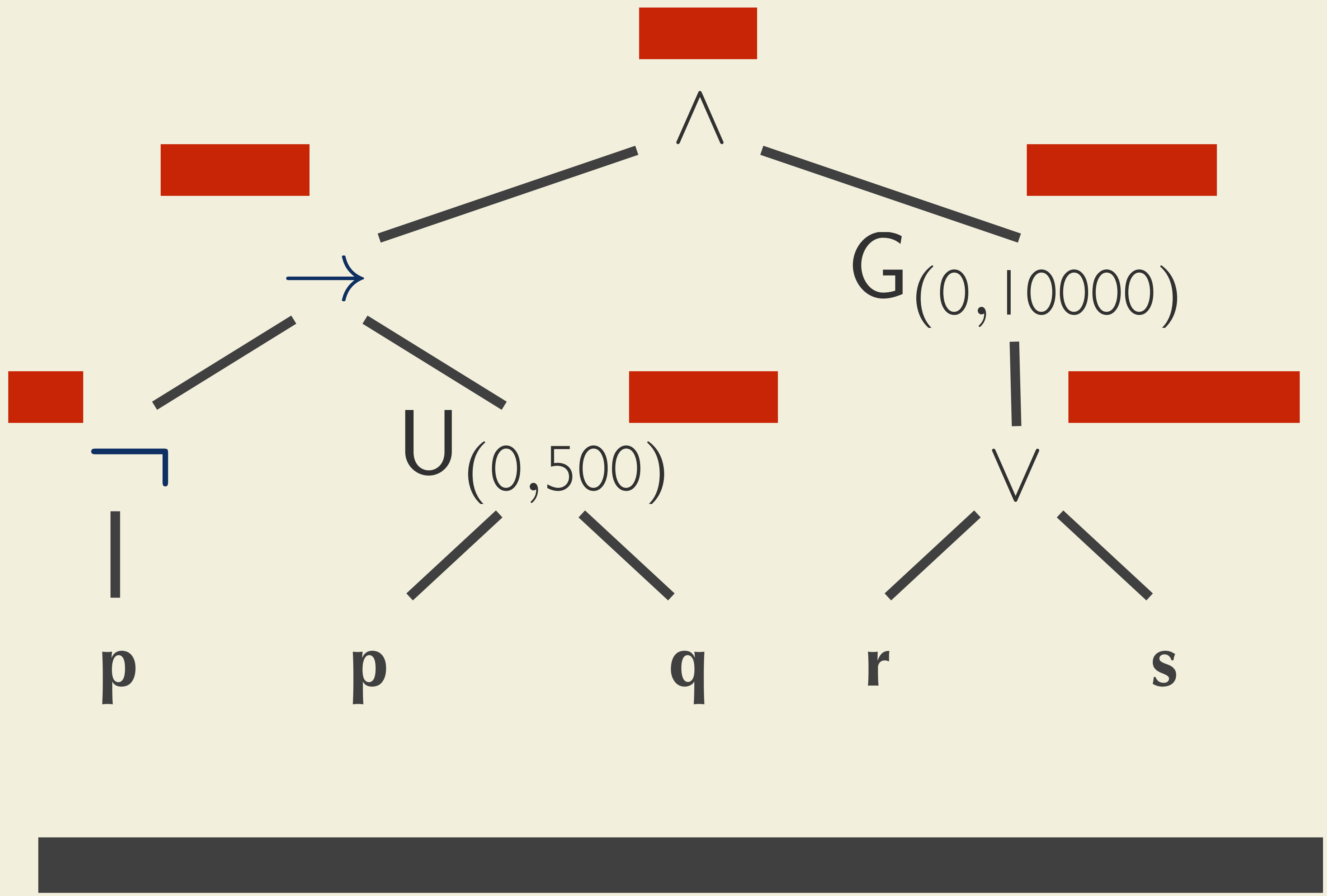
Trace

Parallel sub-formula processing

MAP: associate sub-formulae with super-formulae

REDUCE: monitor the sub-formulae

REPEAT



SEFM 2014

Trace checking of Metric Temporal Logic with Aggregating Modalities using MapReduce

Domenico Bianculli[1], Carlo Ghezzi[2], and Srđan Krstić[2]

[1] SnT Centre - University of Luxembourg, Luxembourg
domenico.bianculli@uni.lu
[2] DEEP-SE group - DEIB - Politecnico di Milano, Italy
{ghezzi,krstic}@elet.polimi.it

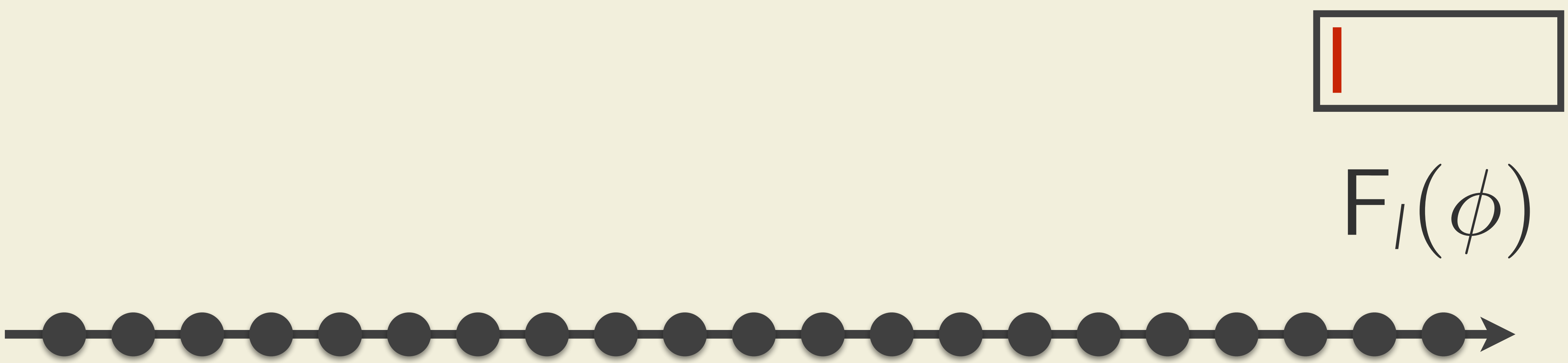
Abstract. Modern, complex software systems produce a large amount of execution data, often stored in logs. These logs can be analyzed using trace checking techniques to check whether the system complies with its requirements specifications. Often these specifications express quantitative properties of the system, which include timing constraints as well as higher-level constraints on the occurrences of events, expressed using aggregate operators. In this paper we present an algorithm that exploits the MapReduce programming model to check specifications expressed in a metric temporal logic with aggregating modalities, over large execution traces. The algorithm exploits the structure of the formula to parallelize the evaluation, with a significant gain in time. We report on the evaluation of the implementation—based on the Hadoop framework—of the proposed algorithm and comment on its scalability.

(SBA), are built accord-

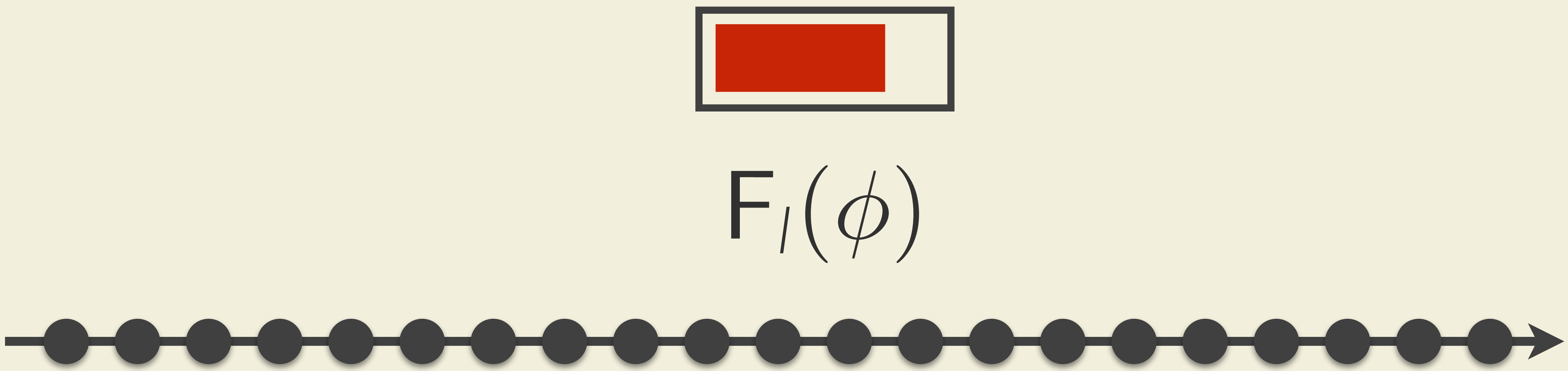
Health Insurance Portability and Accountability Act of 1996

"Retain the documentation [...] for **6 years** from the date of its creation or the date when it last was in effect, whichever is later"

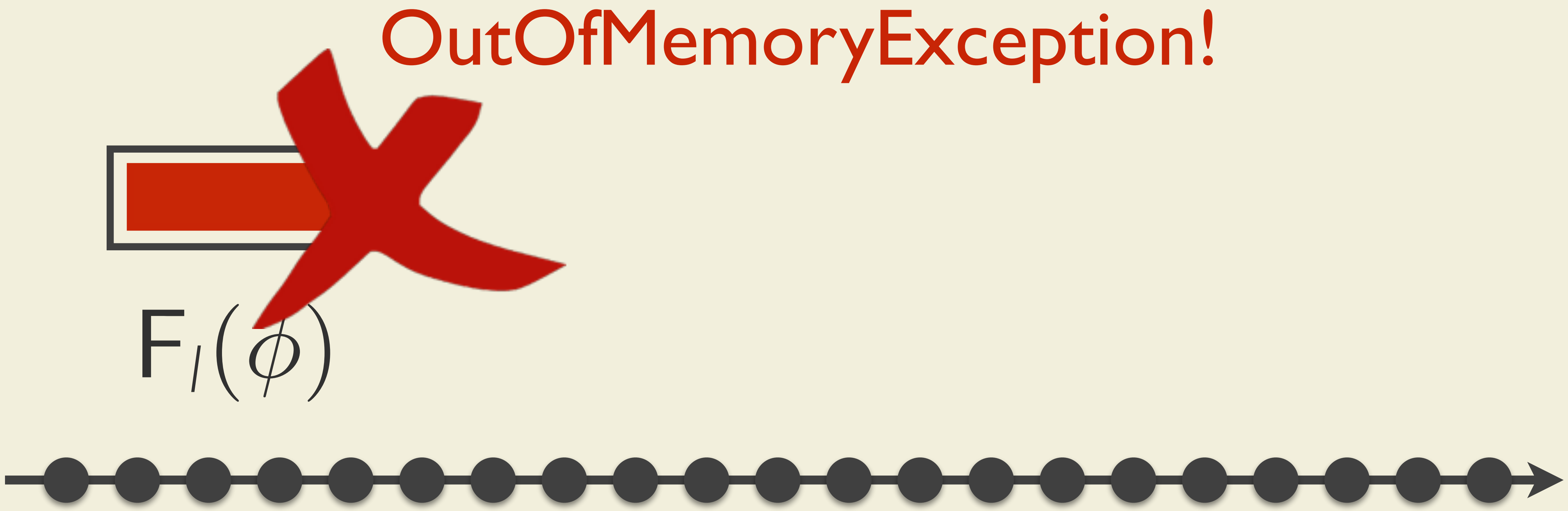
Trace Checking Temporal operators



Trace Checking Temporal operators



Trace Checking Temporal operators



Parallelization Strategies

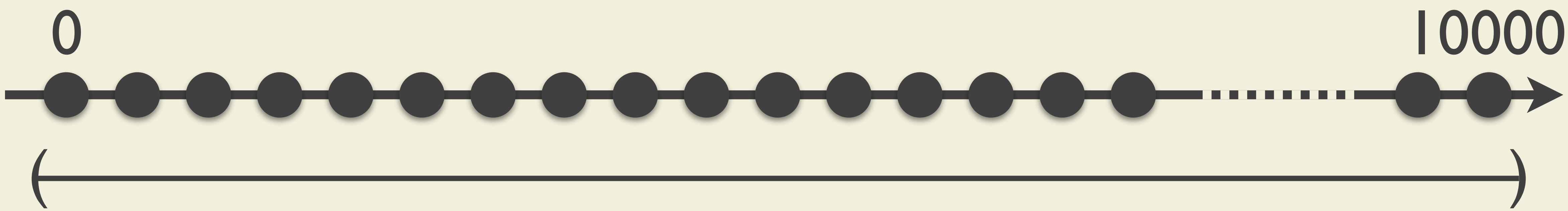
Splitting the formula

- Parallel sub-formula processing
- Temporal operator decomposition

Splitting the trace

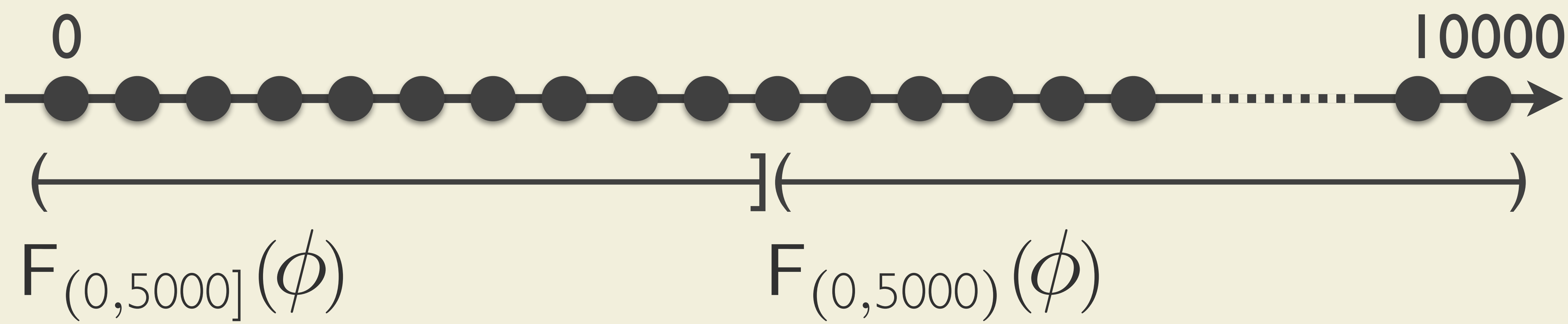
- Time-wise trace splitting (vertical)
- Data-wise trace splitting (horizontal)

Temporal operator decomposition



$F_{(0,10000]}(\phi)$

Temporal operator decomposition



$$F_{(0,10000)}(\phi) \equiv F_{(0,5000]}(\phi) \vee F_{=5000}(F_{(0,5000)}(\phi))$$

Temporal operator decomposition

$$F_{(0,10000)}(\phi) \equiv F_{(0,5000]}(\phi) \vee F_{=5000}(F_{(0,5000)}(\phi))$$

Equivalent?

Yes*

*** if we slightly tweak the MTL semantics**

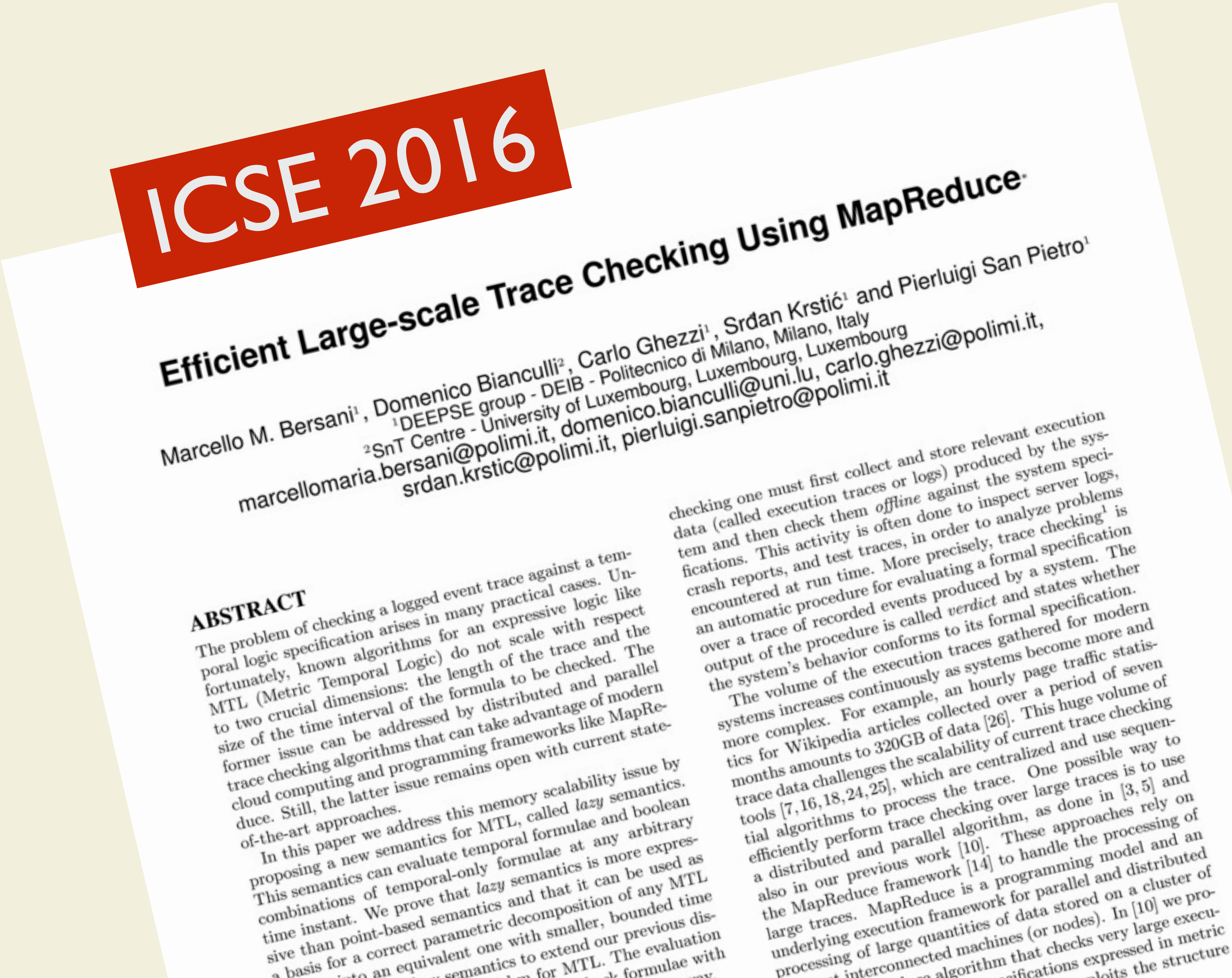
Temporal operator decomposition

1. Analyze the temporal operators in the formula

2. If all intervals are small enough*, apply the parallel sub-formula processing

3. Otherwise, decompose the formula to be small enough* and then apply the parallel sub-formula processing

ICSE 2016



Efficient Large-scale Trace Checking Using MapReduce

Marcello M. Bersani[1], Domenico Bianculli[2], Carlo Ghezzi[1], Srđan Krstić[1] and Pierluigi San Pietro[1]
[1]DEEPSE group - DEIB - Politecnico di Milano, Milano, Italy
[2]SnT Centre - University of Luxembourg, Luxembourg, Luxembourg
marcellomaria.bersani@polimi.it, domenico.bianculli@uni.lu, carlo.ghezzi@polimi.it,
srdan.krstic@polimi.it, pierluigi.sanpietro@polimi.it

ABSTRACT

The problem of checking a logged event trace against a temporal logic specification arises in many practical cases. Unfortunately, known algorithms for an expressive logic like MTL (Metric Temporal Logic) do not scale with respect to two crucial dimensions: the length of the trace and the size of the time interval of the formula to be checked. The former issue can be addressed by distributed and parallel trace checking algorithms that can take advantage of modern cloud computing and programming frameworks like MapReduce. Still, the latter issue remains open with current state-of-the-art approaches.

In this paper we address this memory scalability issue by proposing a new semantics for MTL, called *lazy* semantics. This semantics can evaluate temporal formulae and boolean combinations of temporal-only formulae at any arbitrary time instant. We prove that *lazy* semantics is more expressive than point-based semantics and that it can be used as a basis for a correct parametric decomposition of any MTL

checking one must first collect and store relevant execution data (called execution traces or logs) produced by the system and then check them *offline* against the system specifications. This activity is often done to inspect server logs, crash reports, and test traces, in order to analyze problems encountered at run time. More precisely, trace checking[1] is an automatic procedure for evaluating a formal specification over a trace of recorded events produced by a system. The output of the procedure is called *verdict* and states whether the system's behavior conforms to its formal specification.

The volume of the execution traces gathered for modern systems increases continuously as systems become more and more complex. For example, an hourly page traffic statistics for Wikipedia articles collected over a period of seven months amounts to 320GB of data [26]. This huge volume of trace data challenges the scalability of current trace checking tools [7,16,18,24,25], which are centralized and use sequential algorithms to process the trace. One possible way to efficiently perform trace checking over large traces is to use a distributed and parallel algorithm, as done in [3,5] and also in our previous work [10]. These approaches rely on the MapReduce framework [14] to handle the processing of large traces. MapReduce is a programming model and an underlying execution framework for parallel and distributed processing of large quantities of data stored on a cluster of

Parallelization Strategies

Splitting the formula

- Parallel sub-formula processing
- Temporal operator decomposition

Splitting the trace

- Time-wise trace splitting
- Data-wise trace splitting

Splitting the trace

Goal: Split the trace T based on a particular formula Φ into trace slices $T^1, T^2, \ldots T^k$ such that:

if $T \models \Phi$ then for all $\forall i. T^i \models \Phi$

if $T \not\models \Phi$ then there exists i such that $T^i \not\models \Phi$

Parallelization Strategies

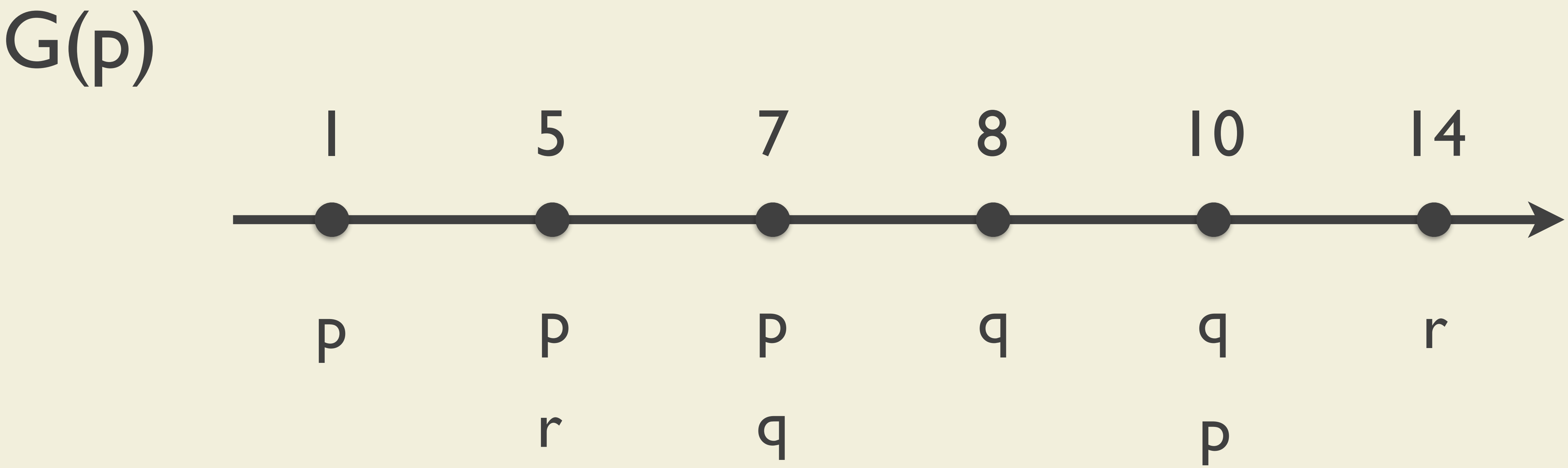
Splitting the formula

- Parallel sub-formula processing
- Temporal operator decomposition

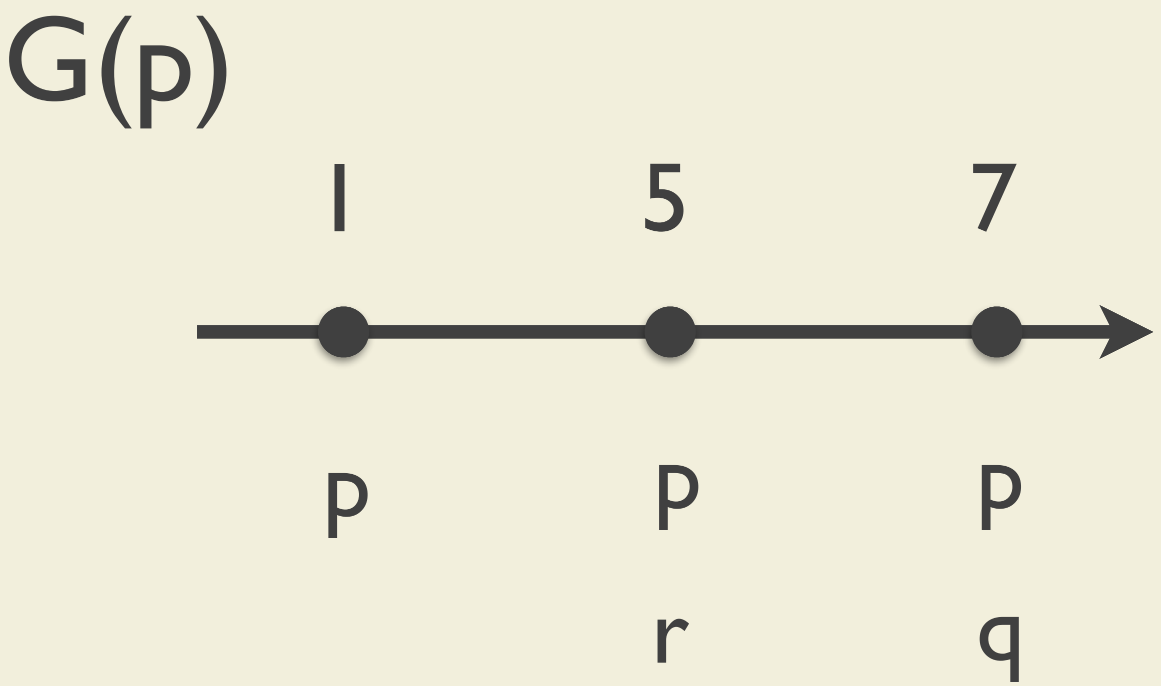
Splitting the trace

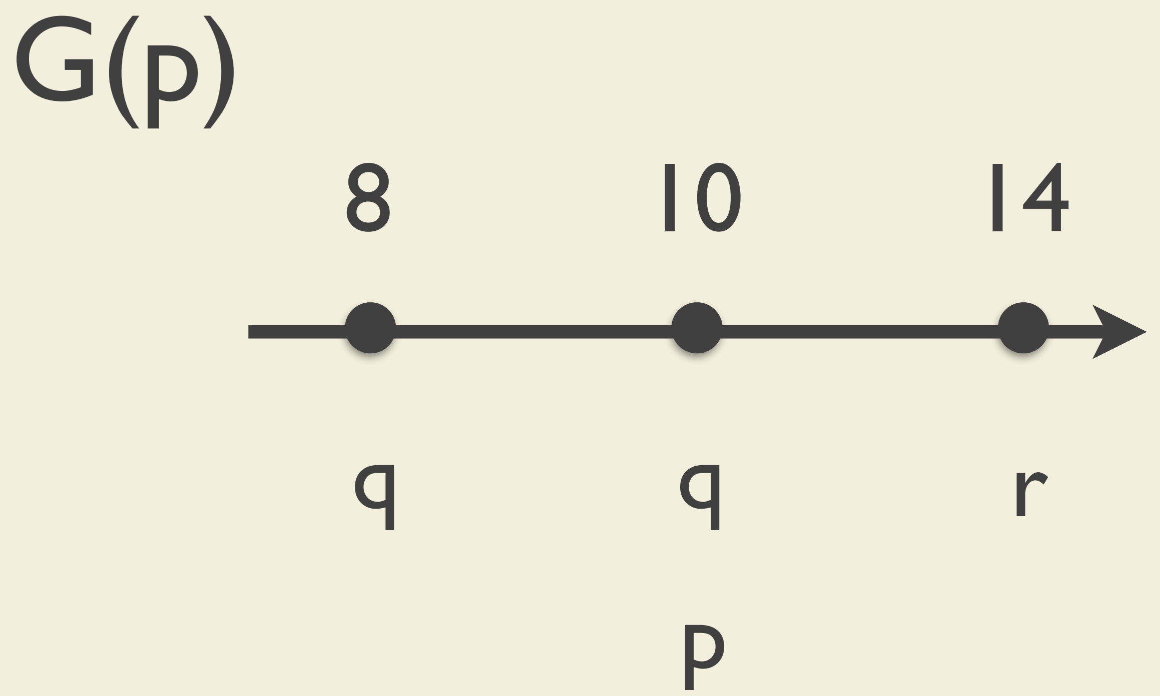
- Time-wise trace splitting
- Data-wise trace splitting

Time-wise trace splitting



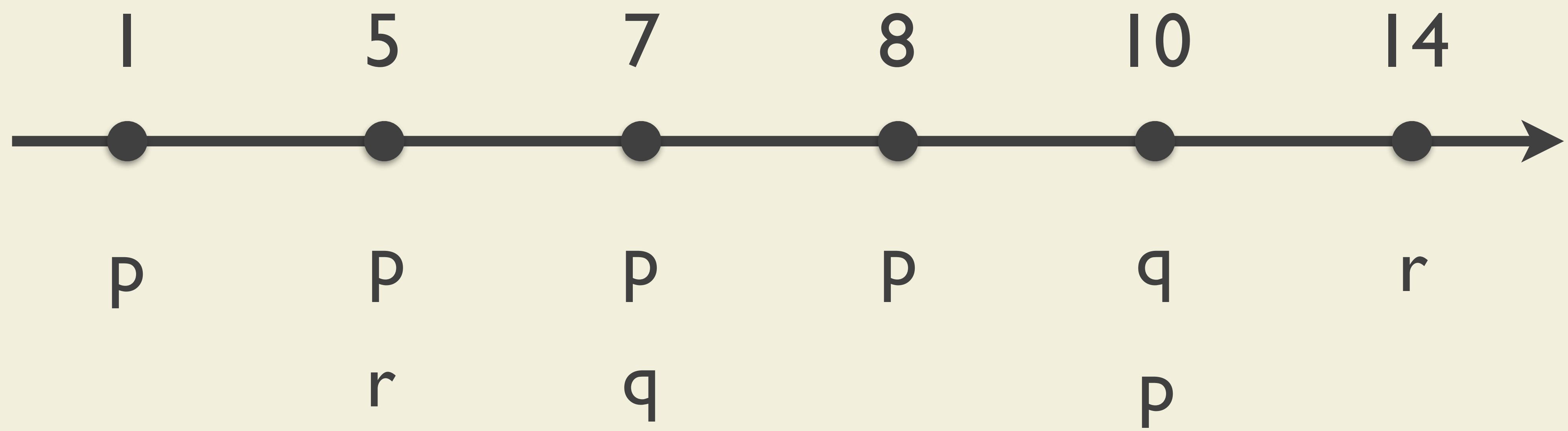
Time-wise trace splitting





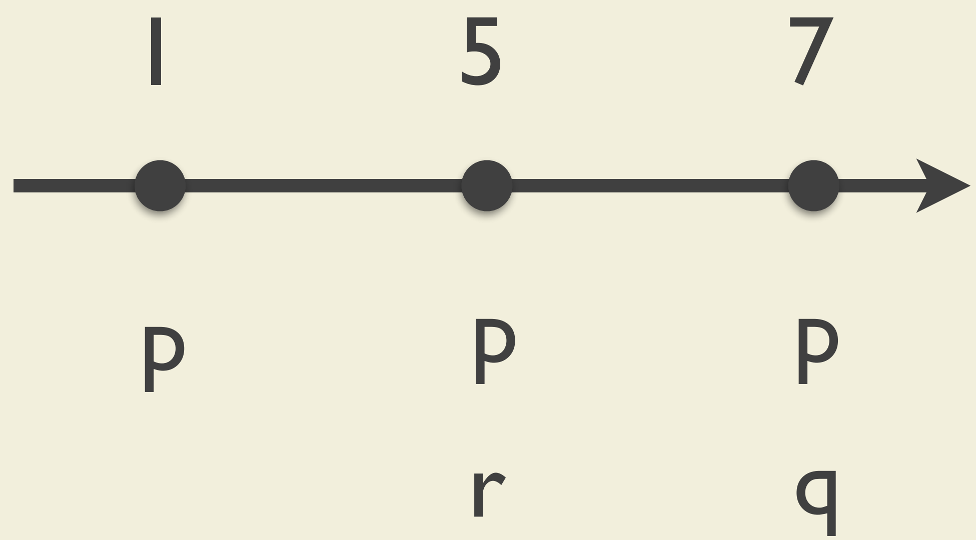
Time-wise trace splitting

$G_{[0,3]}(p)$

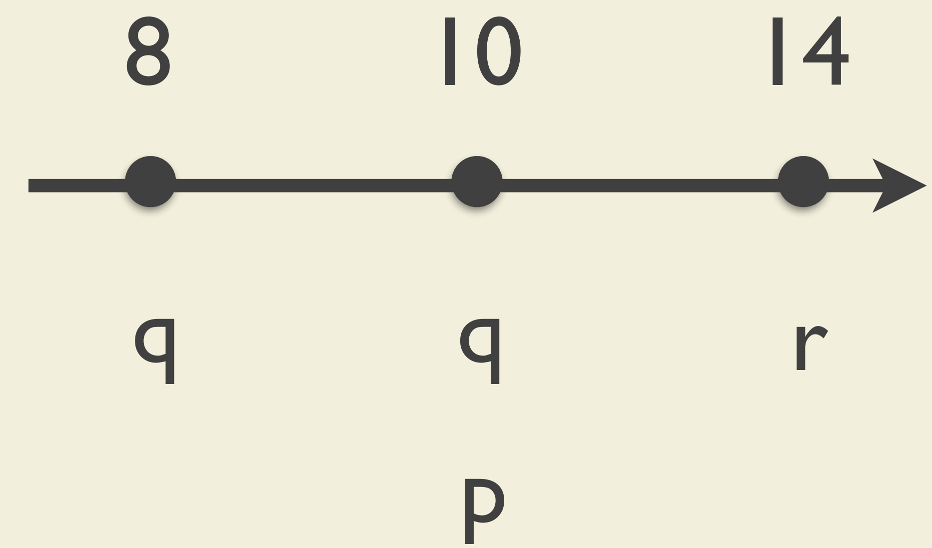


Time-wise trace splitting

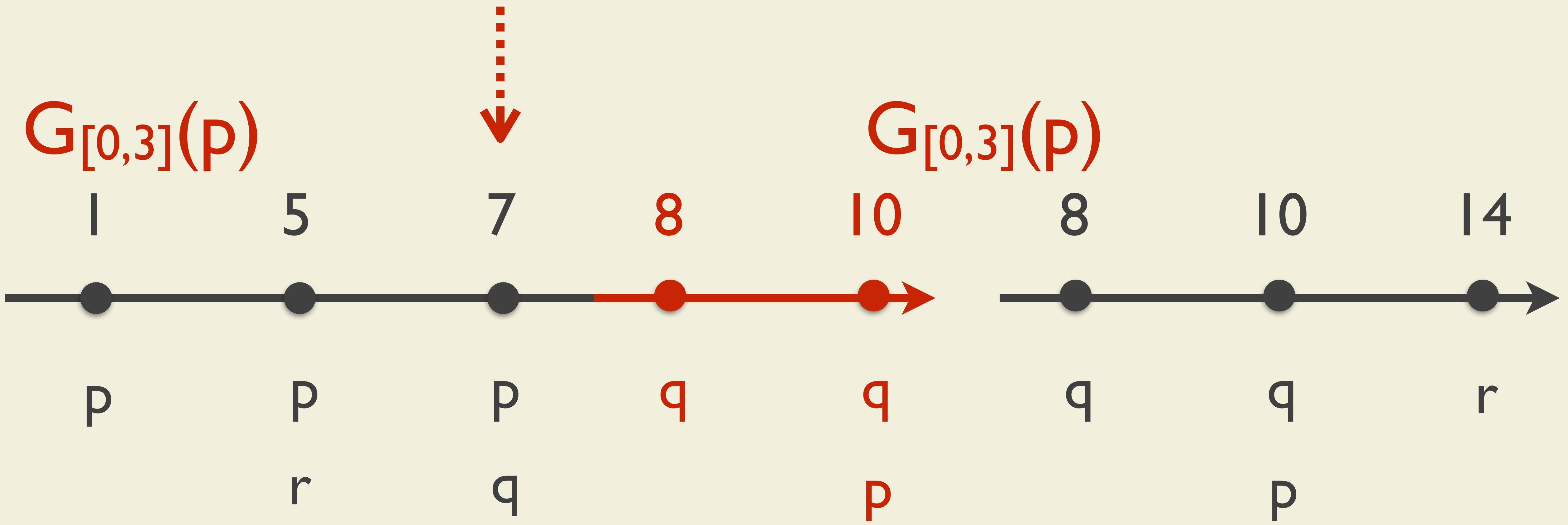
$G_{[0,3]}(p)$



$G_{[0,3]}(p)$



Time-wise trace splitting



Splitting: The Temporal Structure

$\Phi \quad \Rightarrow \quad \mathrm{range}(\Phi)$

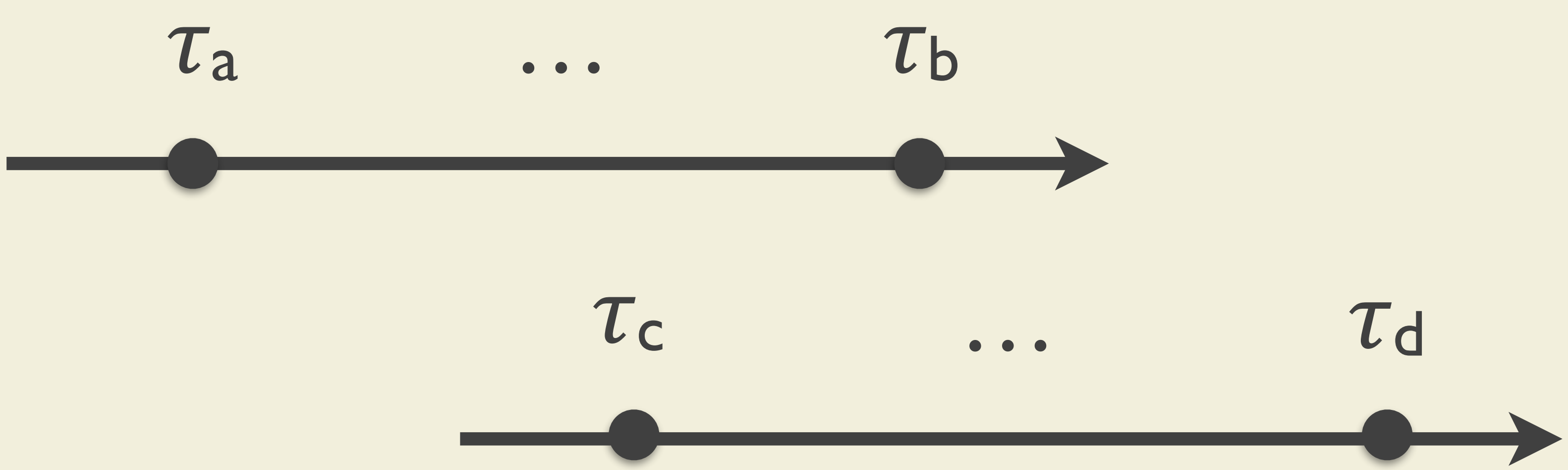
[0,0] [0,7] [0,3]

$(p \wedge q)\ U_{[2,4]}\ (F_{[1,3]}p)$

[0,0] [0,0] [0,0]

Splitting: The Temporal Structure

$$\tau_b - \tau_c = \text{range}(\Phi)$$



Parallelization Strategies

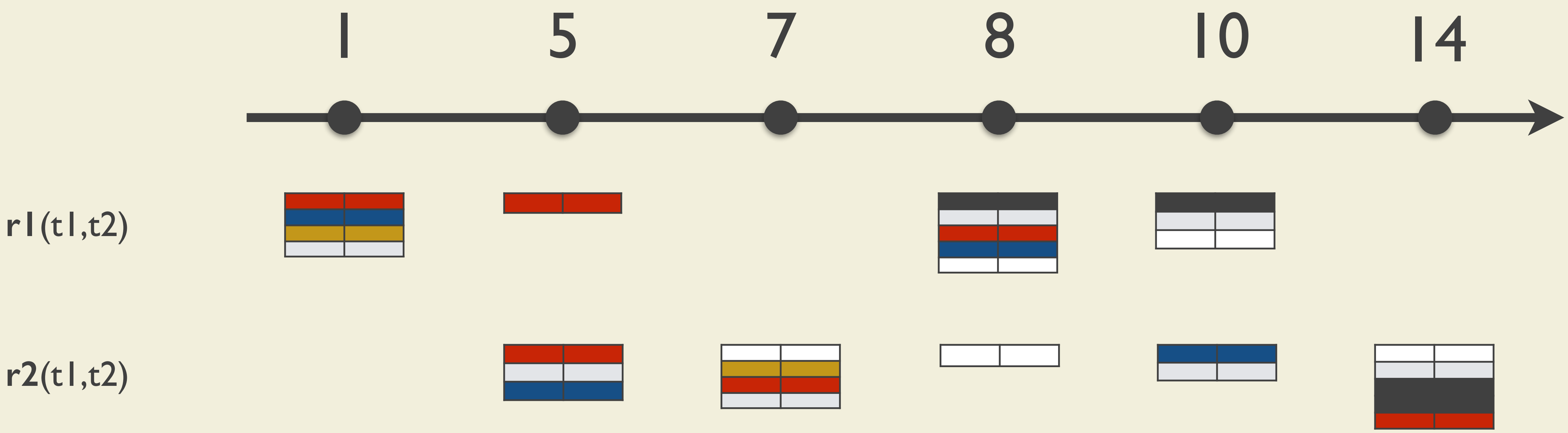
Splitting the formula

- Parallel sub-formula processing
- Temporal operator decomposition

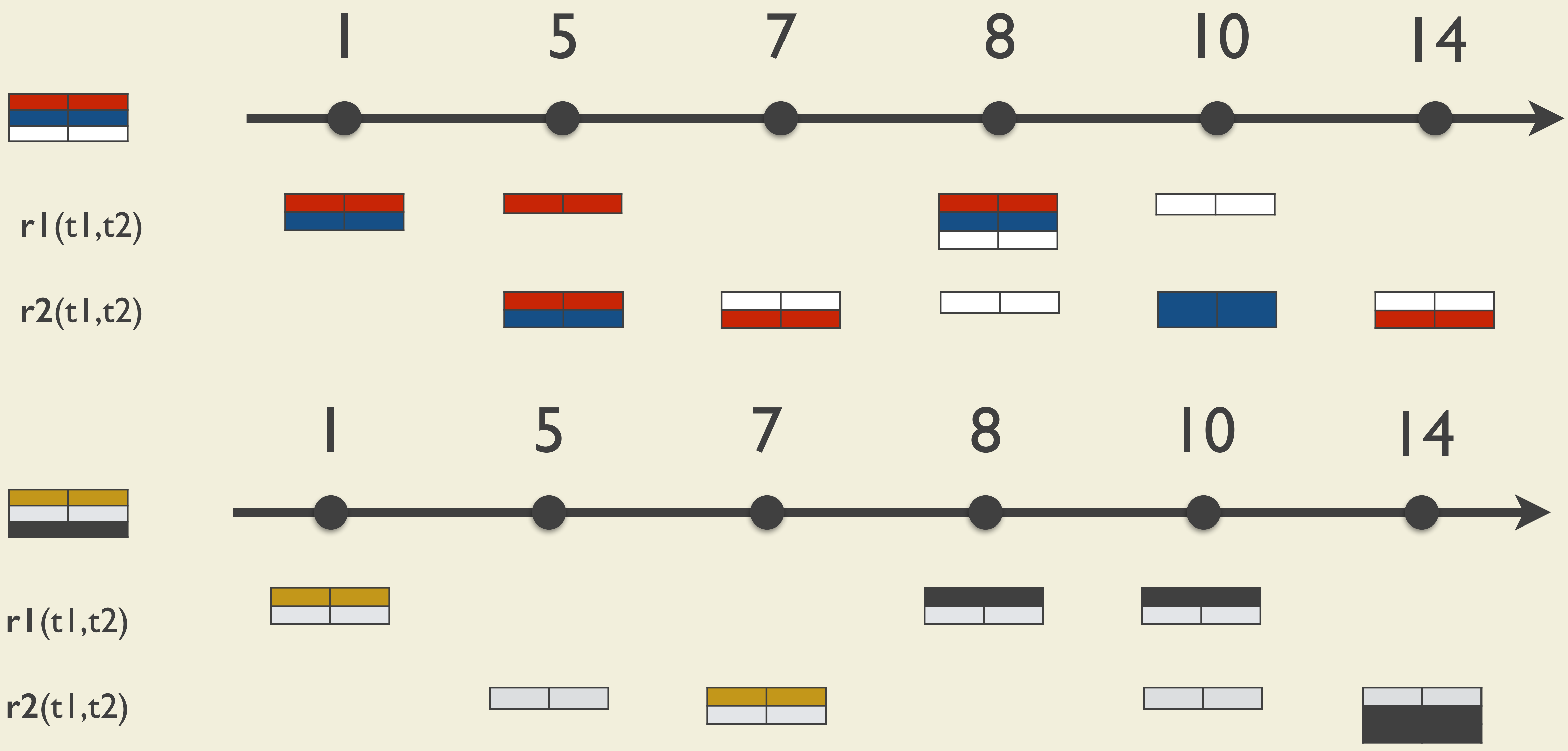
Splitting the trace

- Time-wise trace splitting
- Data-wise trace splitting

Data-wise trace splitting

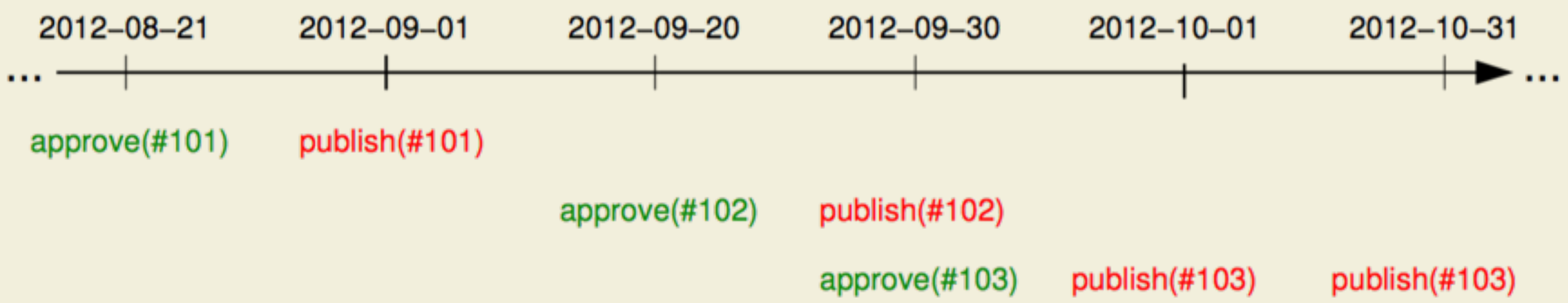


Data-wise trace splitting



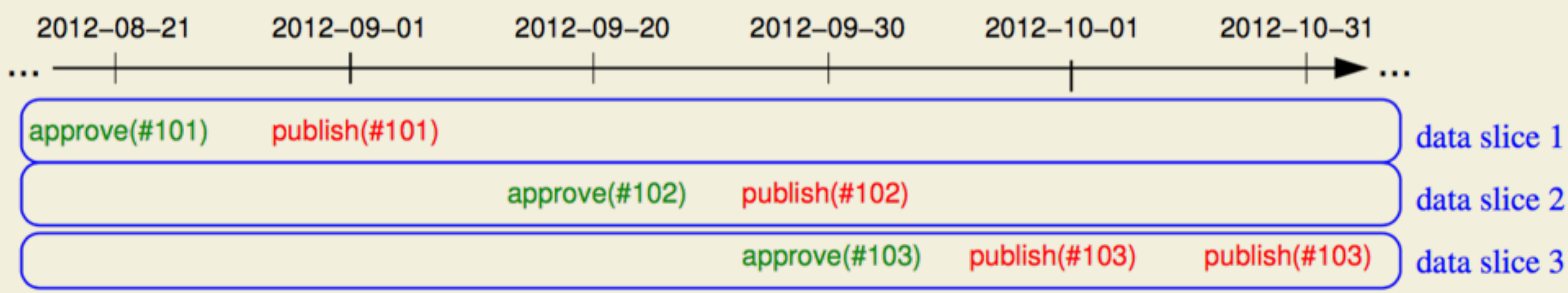
Example

- Split log based on parameters of log events
- Formula: $\mathsf{G}(\forall r.(\mathit{publish}(r) \rightarrow \mathsf{P}(\mathit{approve}(u))))$
- Slices cover different reports:



Example

- Split log based on parameters of log events
- Formula: $\mathsf{G}(\forall r.(\mathit{publish}(r) \to \mathsf{P}(\mathit{approve}(u))))$
- Slices cover different reports:



Example

- What about these formulae?

$$\mathsf{G}(\forall r.(\mathit{publish}(r) \rightarrow \neg\exists r'.(\mathsf{P}\ \mathit{publish}(r') \wedge r' > r)))$$

$$\mathsf{G}(\forall r.(\mathit{publish}(r) \rightarrow \mathsf{F}\ \mathit{publish}(\mathit{summary})))$$

- Bottom line: it's a heuristic

RV 2014

Scalable Offline Monitoring*

David Basin[1], Germano Caronni[2], Sarah Ereth[3], Matúš Harvan[4], Felix Klaedtke[5], and Heiko Mantel[3]

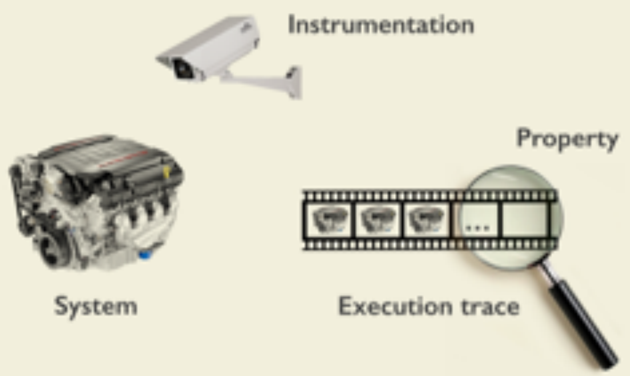
[1] Institute of Information Security, ETH Zurich, Switzerland
[2] Google Inc., Switzerland
[3] Department of Computer Science, TU Darmstadt, Germany
[4] ABB Corporate Research, Switzerland
[5] NEC Europe Ltd., Heidelberg, Germany

Abstract. We propose an approach to monitoring IT systems offline, where system actions are logged in a distributed file system and subsequently checked for compliance against policies formulated in an expressive temporal logic. The novelty of our approach is that monitoring is parallelized so that it scales to large logs. Our technical contributions comprise a formal framework for slicing logs, an algorithmic realization based on MapReduce, and a high-performance implementation. We evaluate our approach analytically and experimentally, proving the soundness and completeness of our slicing techniques and demonstrating its practical feasibility and efficiency on real-world logs with 400 GB of relevant data.

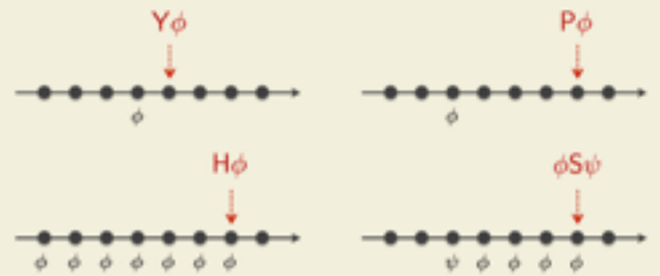
1 Introduction

Summary

Monitoring



Linear Temporal Logic (LTL)
Past operators



MTL Monitoring Algorithm

Main Challenge

Large traces that cannot be collected, stored and processed on a single machine.

Solution: Distributed Monitoring using MapReduce

Parallelization Strategies

Splitting the formula (general, limited parallelization)

- Parallel sub-formula processing
- Temporal operator decomposition

Splitting the trace (heuristic, high parallelization)

- Time-wise trace splitting
- Data-wise trace splitting

Open problems

- Combine the orthogonal parallelization strategies
- Provide a general distributed framework for online monitoring
- Generalize the monitoring approach: given a formula, the least general (hence, the least complex) algorithm is used to monitor it

Cloud-based Software Verification

Srđan Krstić
Politecnico di Milano

Joint work with Carlo Ghezzi, Domenico Bianculli, Marcello Bersani and Pierluigi San Pietro

References

- [1] David Basin, Felix Klaedtke, and Eugen Zalinescu: Algorithms for Monitoring Real-time Properties
- [2] Domenico Bianculli, Carlo Ghezzi and Srdjan Krstic. Trace checking of Metric Temporal Logic with Aggregating Modalities using MapReduce.
- [3] Marcello Maria Bersani, Domenico Bianculli, Carlo Ghezzi, Srdjan Krstic, and Pierluigi San Pietro. Efficient Large-scale Trace Checking Using MapReduce
- [4] Martin Leucker, Christian Schallhart: A brief account of runtime verification
- [5] Srdjan Krstic: Trace Checking of Quantitative Properties
- [6] Martin Leucker: Teaching Runtime Verification
- [7] David A. Basin, Felix Klaedtke, Samuel Müller, Eugen Zalinescu: Monitoring Metric First-Order Temporal Properties
- [8] Marcello Maria Bersani, Domenico Bianculli, Carlo Ghezzi, Srdjan Krstic, and Pierluigi San Pietro: SMT-based checking of SOLOIST over sparse traces
- [9] Domenico Bianculli, Carlo Ghezzi, Srđan Krstić, and Pierluigi San Pietro: Offline trace checking of quantitative properties of service-based applications
- [10] David Basin, Germano Caronni, Sarah Ereth, Matus Harvan, Felix Klaedtke, and Heiko Mantel: Scalable Offline Monitoring