UNIVERSITÀ DEGLI STUDI DI MILANO
Dept. of Computer Science

Formal Verification Problems in a Bigdata World: Towards a Mighty Synergy

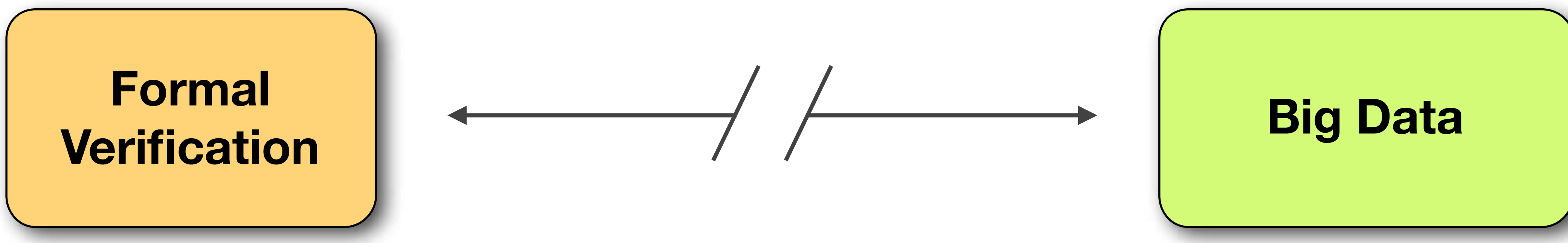
Matteo Camilli
matteo.camilli@unimi.it
http://camilli.di.unimi.it

ICSE 2014
Hyderabad, India
June 3, 2014

Outline

- Introduction, Motivations, Objectives
- Background
- Some details on:
  - MapReduce
  - Techniques, Frameworks and Tools
- Experiments
- Conclusion
- Planned work

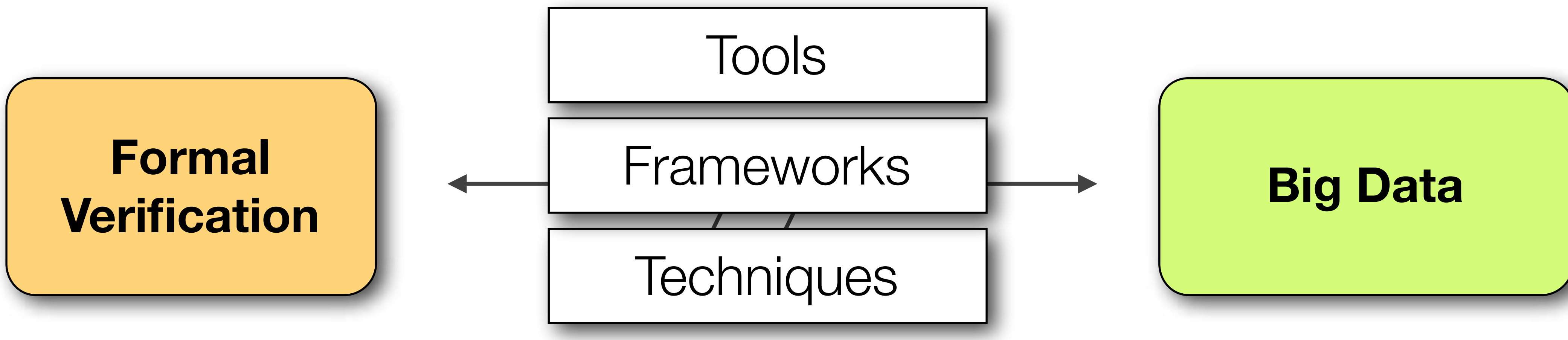
Introduction



- background on formal methods
    - Modeling
    - Interpreting

- deploy techniques into software tools able to analyze large amount of data very reliably and efficiently
- adapting an application for exploiting the scalability provided by cloud computing facilities.

Introduction

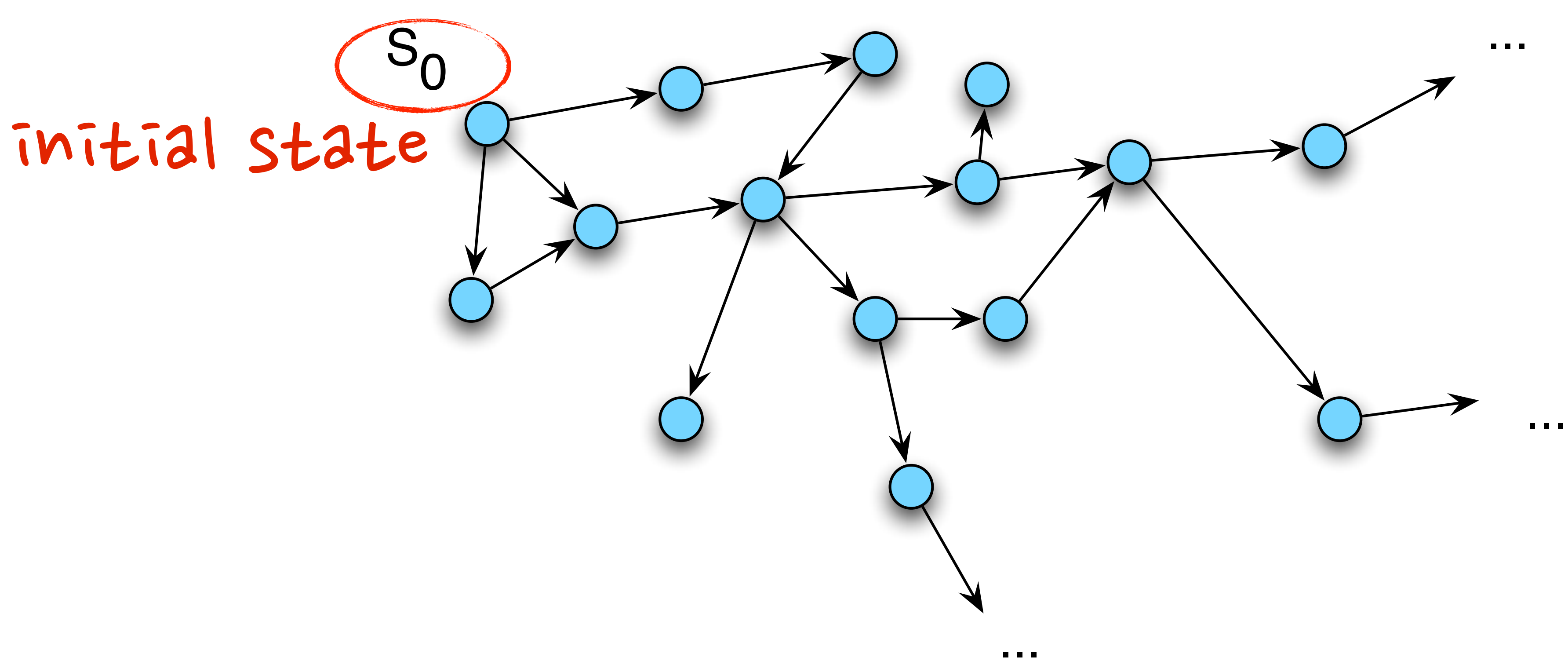


- background on formal methods
    - Modeling
    - Interpreting

- deploy techniques into software tools able to analyze large amount of data very reliably and efficiently
- adapting an application for exploiting the scalability provided by cloud computing facilities.

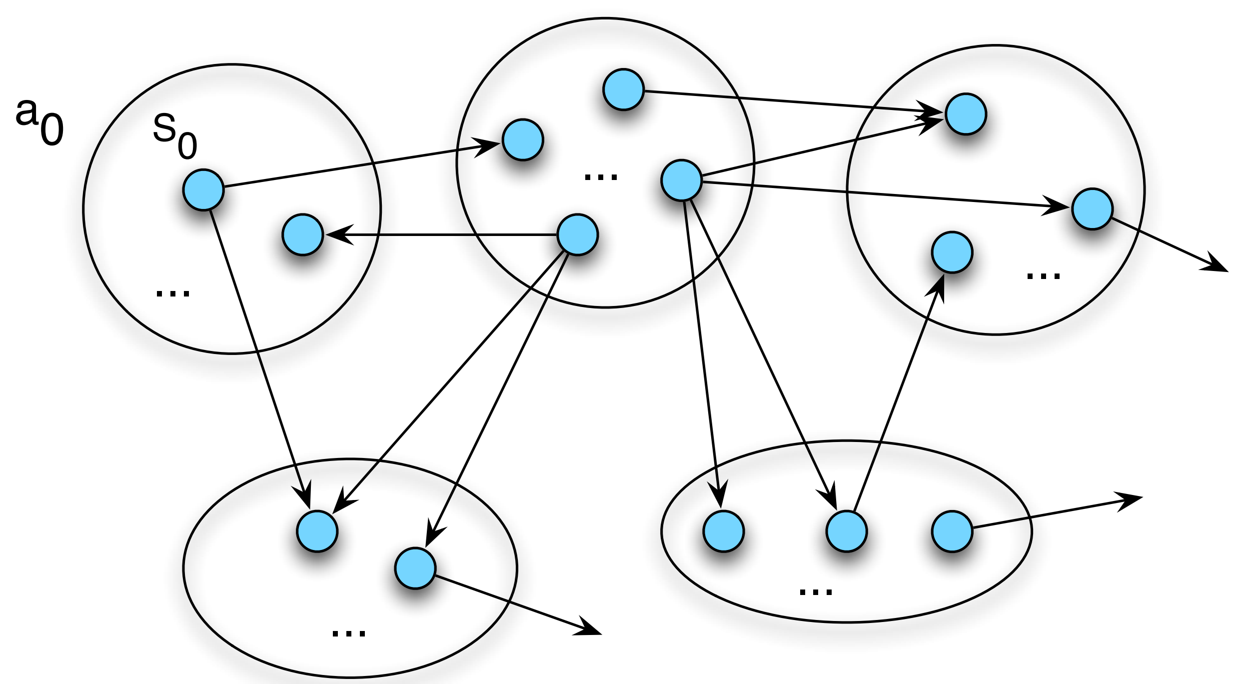
Background

- The behavior of a discrete-event dynamic system is formally given in terms of a labeled state transition system: $(S, \Lambda, \rightarrow)$
- Λ is a set of labels
- $\rightarrow \subseteq S \times \Lambda \times S$ s.t. $(s,\lambda,s') \in \rightarrow$ iff s' reachable from s (written as $s \xrightarrow{\lambda} s'$)



Background

- In general S may be infinite, or even uncountable. Some abstraction techniques are required in order to be able to enumerate the whole state space.
- Abstract State Space: (A, L,⇒)
- Where A is a coverage of S, and $\Rightarrow \subseteq A \times L \times A$ s.t. exists a morphism f which maps Λ labels into L labels.

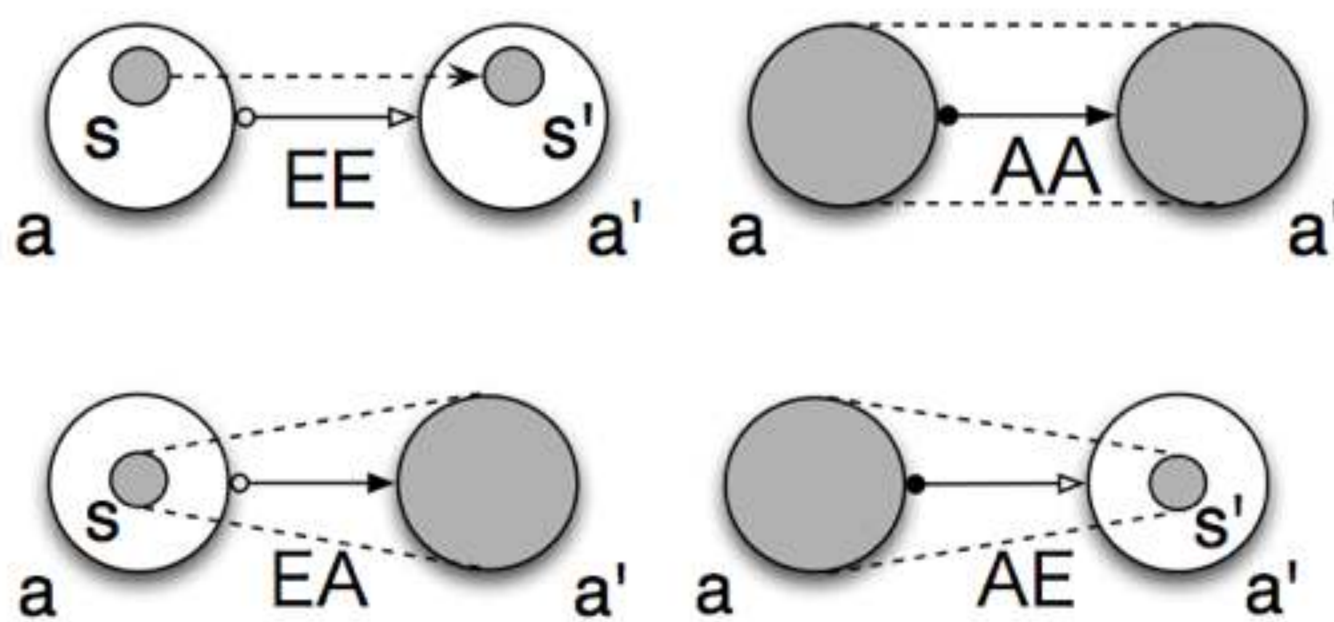


Background

- The relation $\Rightarrow$ satisfies the condition **EE:**

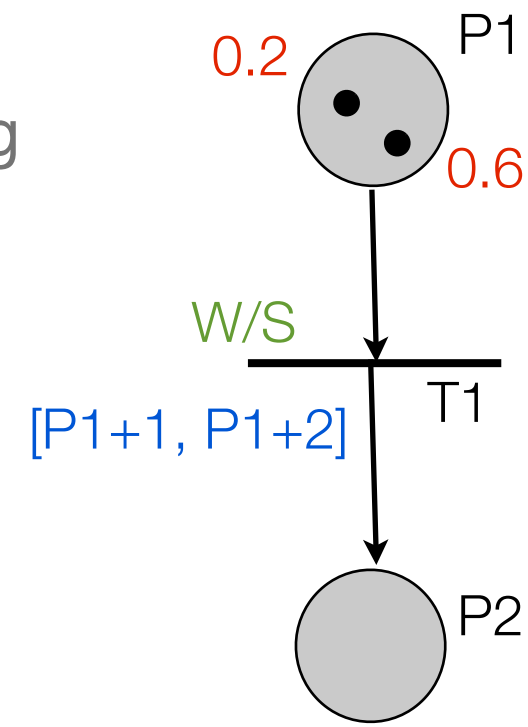
  (1) if $a \overset{l}{\Rightarrow} a'$, then $\exists s \in a,\ s' \in a' : s \overset{\lambda}{\longrightarrow} s'$ with $\lambda \in f^{-1}(l)$

  (2) if $s \overset{\lambda}{\longrightarrow} s'$, then $\forall a \in A$ s.t. $s \in a$, $\exists a' \in A$ s.t. $s' \in a' \wedge a \overset{f(\lambda)}{\Rightarrow} a'$



Time Basic nets - Reachability analysis

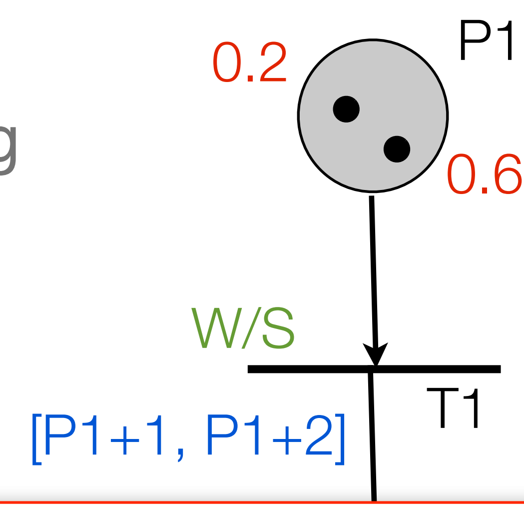
- Three key points of the Time Reachability Graph building algorithm allow in many cases the termination.
  - Identification of inclusions between classes of states
  - Erasure of absolute times
  - Identification of anonymous timestamps



Bellettini, C.; Capra, L.; , "Reachability Analysis of Time Basic Petri Nets: A Time Coverage Approach," *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2011 13th International Symposium on* , vol., no., pp.110-117, 26-29 Sept. 2011 doi: 10.1109/SYNASC.2011.16 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6169509&isnumber=6169489

Time Basic nets - Reachability analysis

- Three key points of the Time Reachability Graph building algorithm allow in many cases the termination.
  - Identification of inclusions between classes of states
  - Erasure of absolute times



Execution of the Gas Burner example:

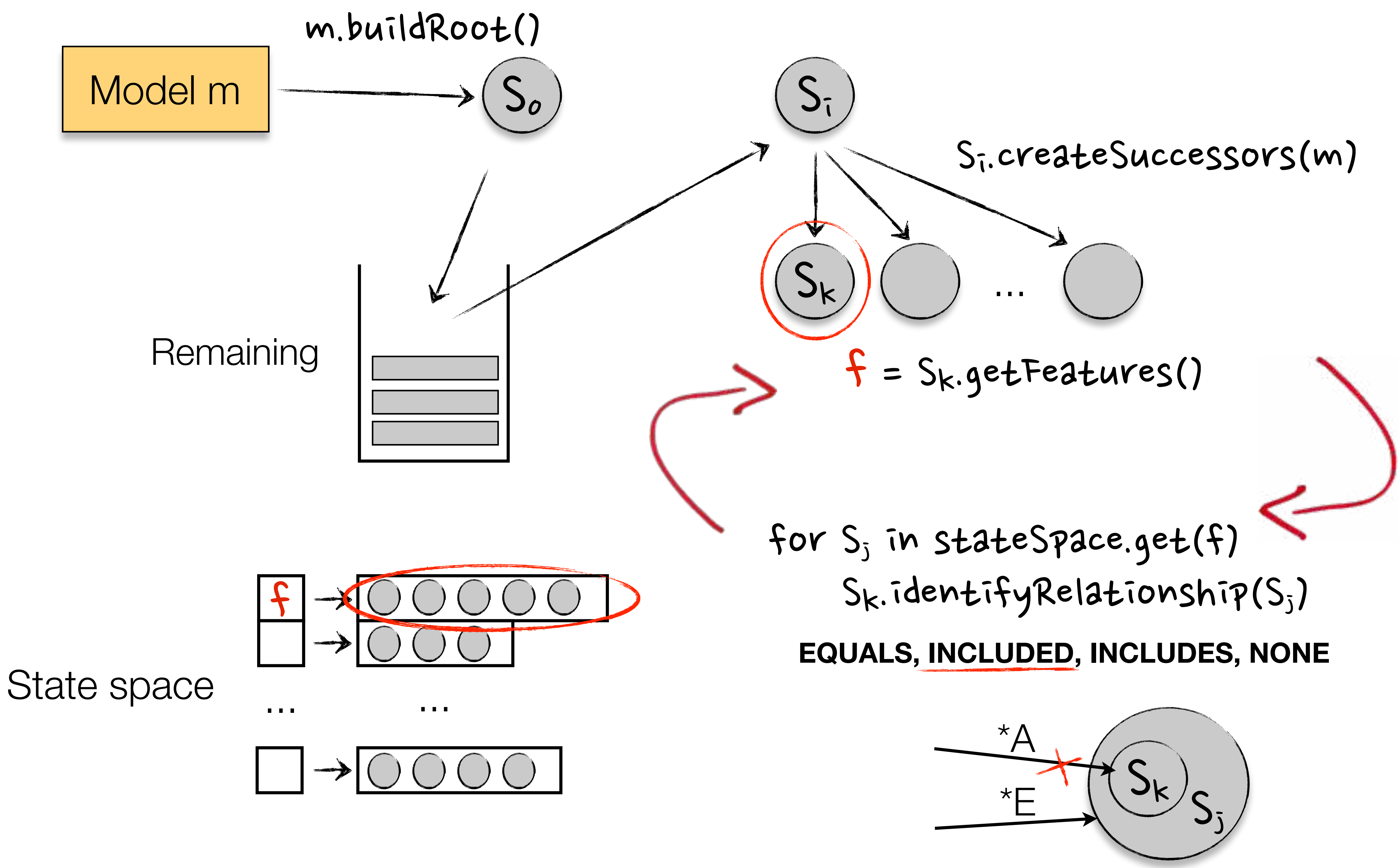
Total built abstract states: 22.978

Final abstract state space: 14.563

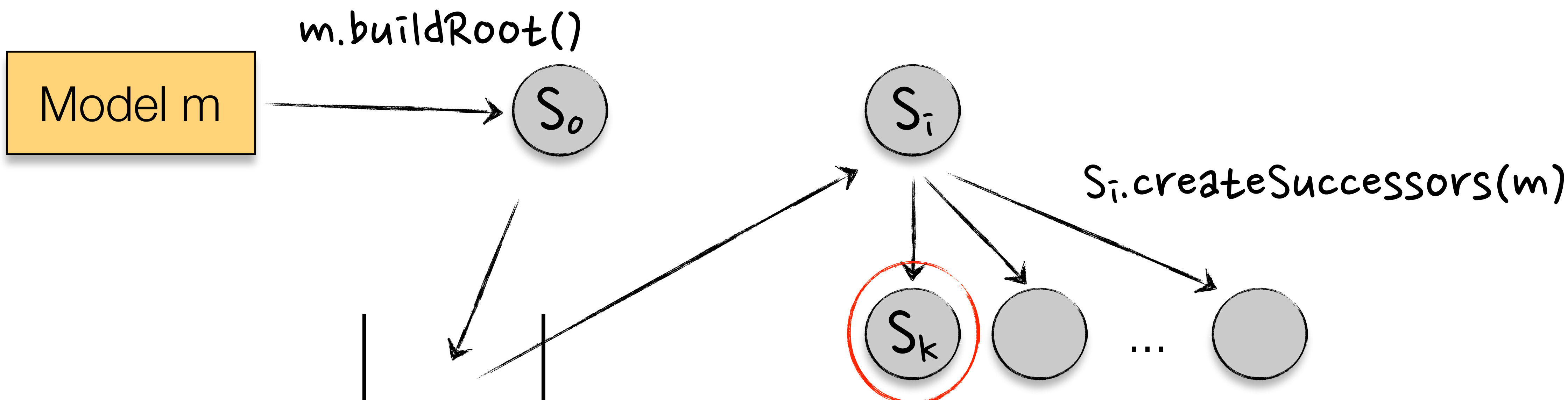
architecture	# CPUs	tool version	compute model	T	H	f	exec. time
2.4Ghz Intel Core 2 Duo, 2GB RAM	1×2 cores	sequential	local (single machine)	-	-	(2)	~7.5 hrs

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6169509&isnumber=6169489

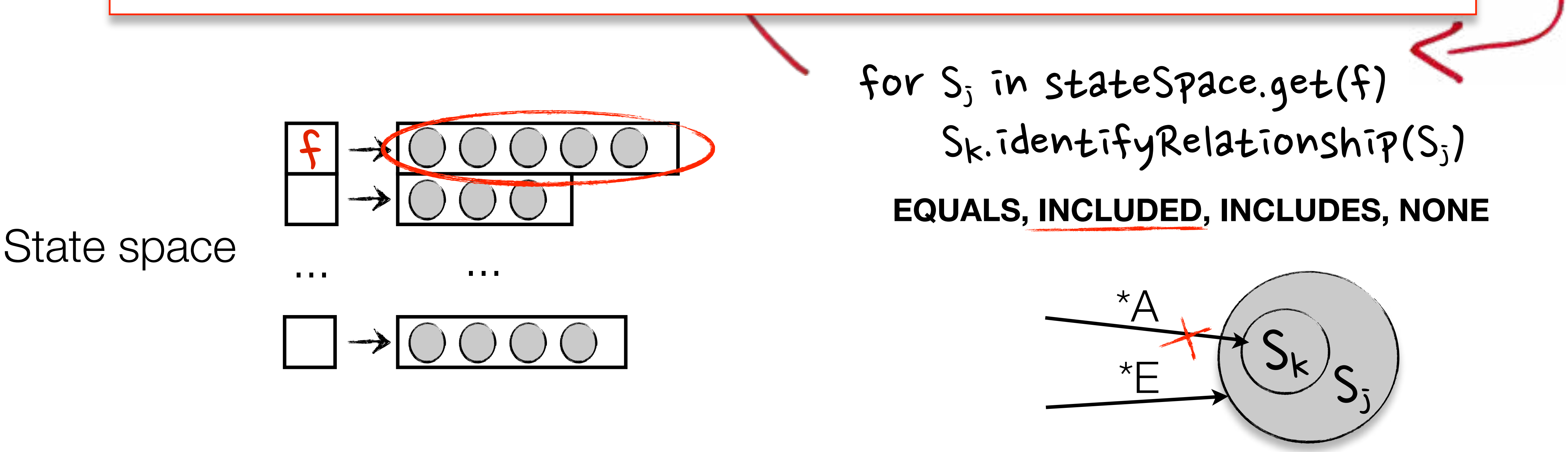
Sequential algorithm



Sequential algorithm

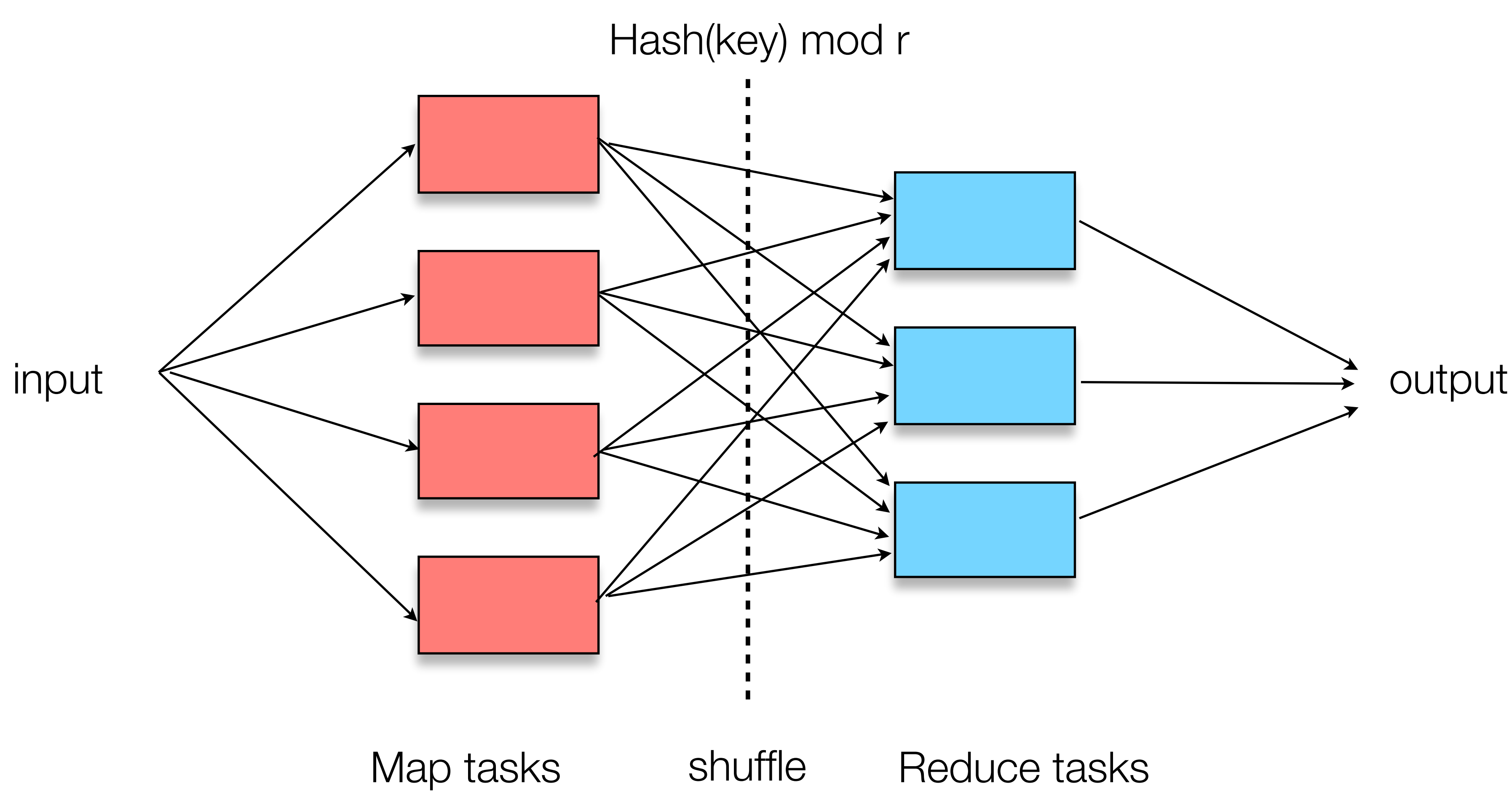


Straightforward, but because of the **state explosion problem** sequential tools may become very slow or even crash.



Map-Reduce

- Map-Reduce job =
  - **Map** function (inputs -> key-value pairs) +
  - **Reduce** function (key and list of values -> outputs)
- Map and Reduce tasks apply Map and Reduce function to many inputs in parallel.

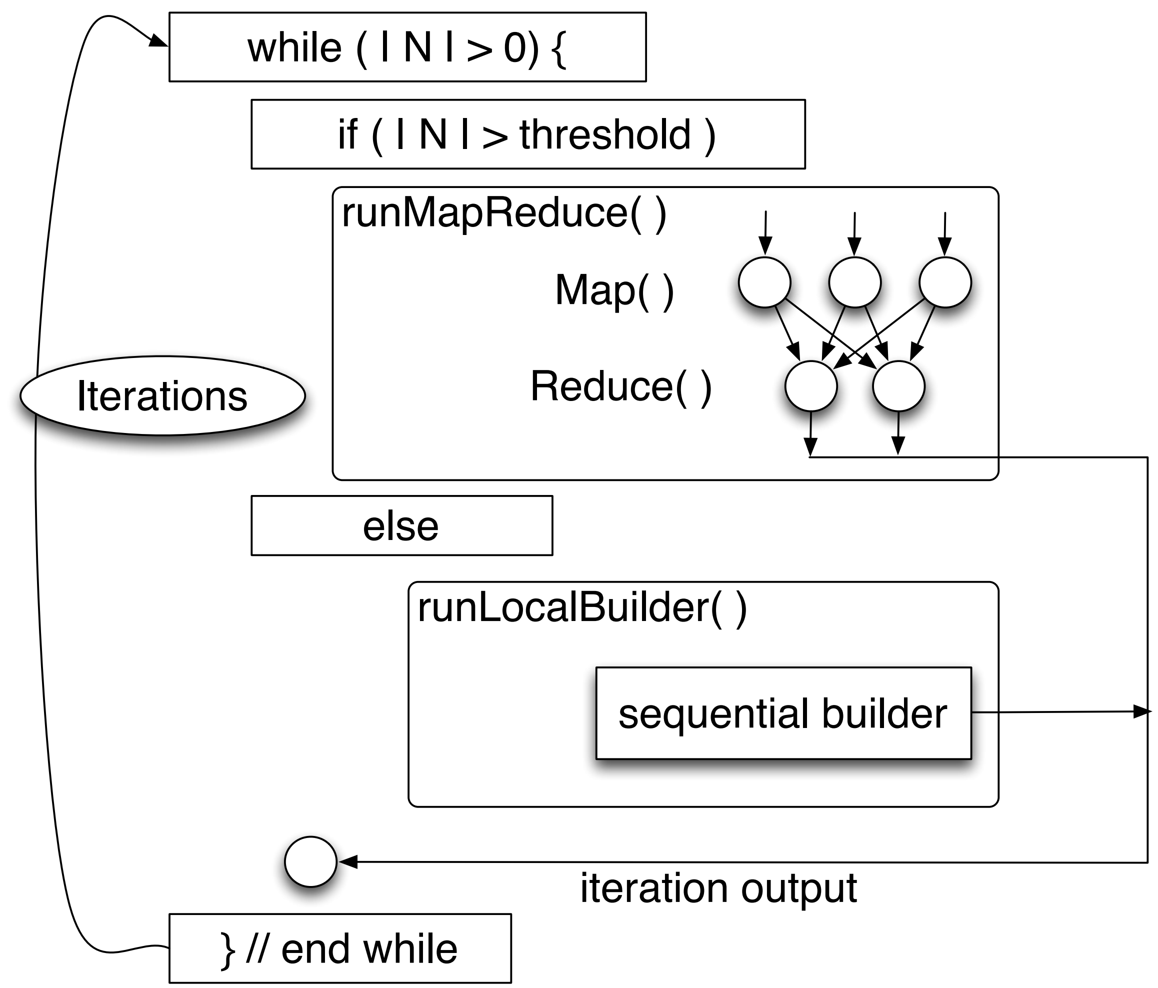


Map-Reduce TB nets analysis tool

- Map step =
  - given an unexplored state, it applies the createSuccessors function. Incoming transitions are stored into destination states by a list of identifiers.
- Shuffle step =
  - Gathers together states potentially related: This is done by using as intermediate keys the evaluation of the getFeatures function.
- Reduce step =
  - given a set of states potentially related, it applies the identifyRelationship function foreach pair of states.
- Building blocks =
  - State = <M,C> pair. M marking, C constraint.
  - identifyRelationship computes the actual relationship between two states according to the following rule: $a \subseteq a' \iff \sigma(M) = \sigma(M') \land C \Rightarrow C'$
  - getFeatures returns just the topological part of $M \equiv \sigma(M)$.

Hybrid Iterative Map-Reduce

- A single Map-Reduce job is not enough: Iterative Map-Reduce
- During the first and last iterations of the algorithm the set of *states* is quite small. Thus a MapReduce job over a large cluster of machines is useless and expensive in term of time and resources.
- The computation starts with a sequential algorithm and goes on until the state space size passes a configurable threshold. After that we distribute the computation over a big cluster.



Hybrid Iterative Map-Reduce

- A single Map-Reduce job is not enough: **Iterative Map-Reduce**
- During the first and last iterations of

Gas Burner example:

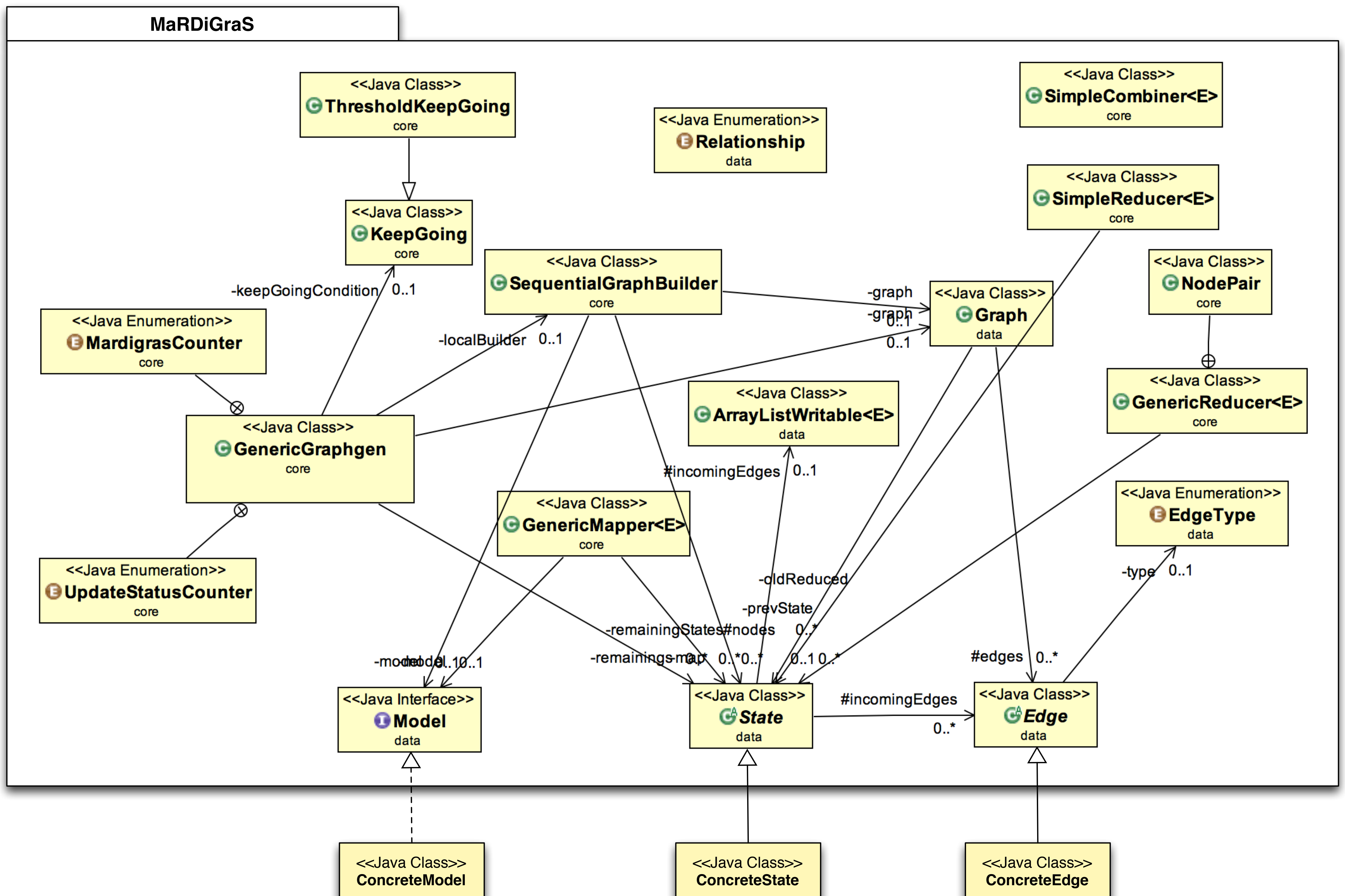
#machines	machine type	#**abstract** states	threshold	time (m)
1	m2.2xlarge	1.456x10	200	175
4	m2.2xlarge	1.456x10	200	95
8	m2.2xlarge	1.456x10	200	39

- The execution with 8 machines is almost 80% faster than the sequential algorithm

we distribute the computation over a big cluster.

MaRDiGraS

MapReduce-based Distributed building of reachability GraphS



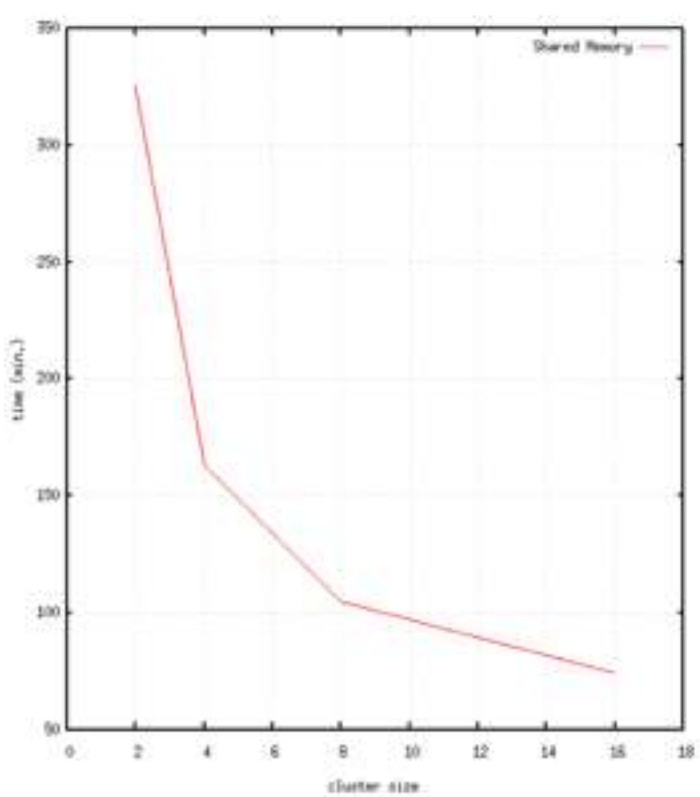
Use Cases

- P/T nets
  - State = <M> marking, associates places with natural numbers.
  - s = s' ⟺ M = M' thus we can use the optimized Reduce phase.
- In order to prove the effectiveness of using MaRDiGraS to improve legacy tools, we adapted an existing P/T nets tool: PIPE.
- To adapt the sequential algorithm of PIPE into a distributed one, we just needed 290 lines of code: a very small number also if compared with the dimension of the effectively used PIPE modules (~6500 lines of code).

Use Cases

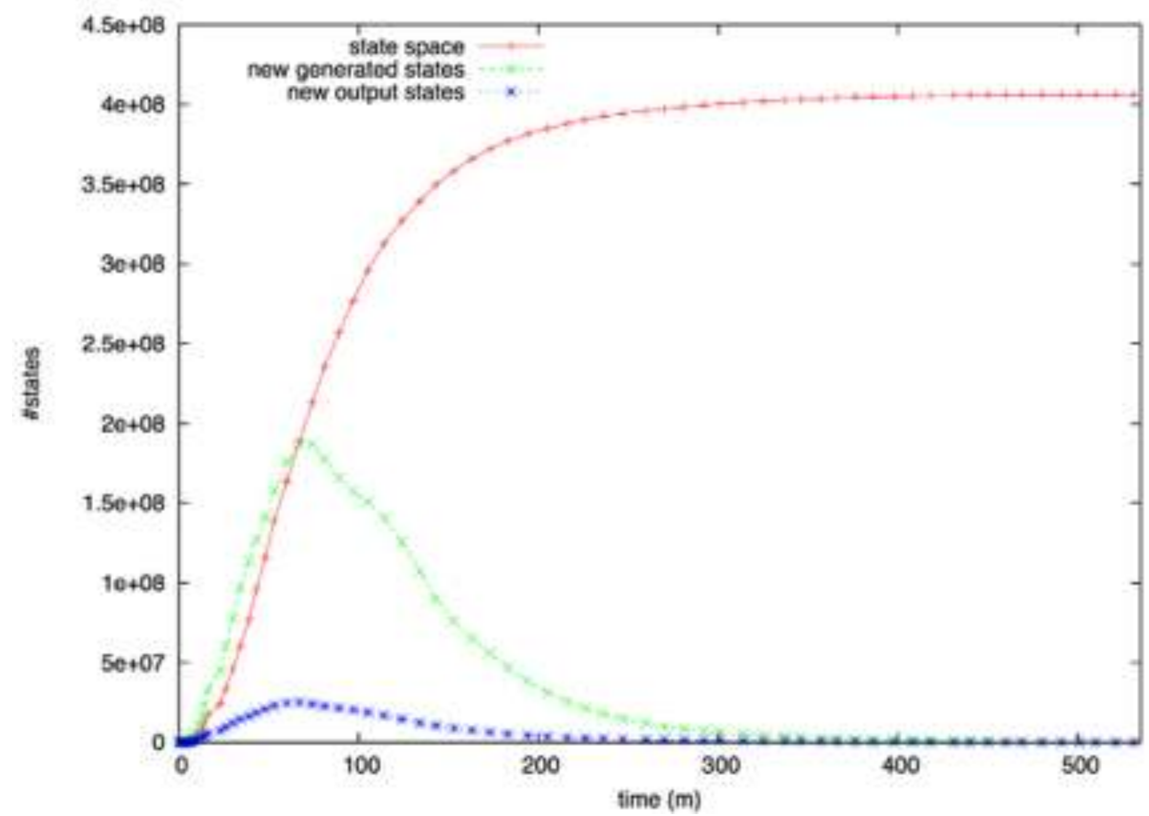
Shared Memory example:

- 1.831×10^6 reachable states
- The PIPE tool takes more than 20 hours to complete the computation.
- The adapted version takes 74 min to complete the same computation, using 16 machines.



Simple Load Balancing example:

- 4.060×10^8 states
  3.051×10^9 transitions
  120GB of data
- execution time = 530 min. using 20 machines.



CTL model checking in the cloud

- We developed a software tool which exploits the MaRDiGraS computed graphs by applying iterative map-reduce algorithms based on fixpoint characterizations of the basic temporal operators of CTL (Computational Tree Logic).
- Given a state transition system $T=\langle S,s_0,R,L\rangle$, and a set of states that satisfy the ϕ formula ($[\phi]_T$)
  - $[EX\phi]_T = R^{-}([\phi]_T)$
  - $[EG\phi]_T = \nu_X([\phi]_T \cap R^{-}(X))$
  - $[E[\phi U\psi]]_T = \mu_X([\psi]_T \cup ([\phi]_T \cap R^{-}(X)))$

Computation Tree Logic

- CTL is a branching-time logic which models time as a tree-like structure where each moment can be followed by several different possible futures. In CTL each basic temporal operator (i.e., either X, F, G) must be immediately preceded by a path quantifier (i.e., either A or E). In particular, CTL formulas are inductively defined as follows

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid A\psi \mid E\psi \ (state\ formulas)$$

$$\psi ::= X\phi \mid F\phi \mid G\phi \mid \phi U \phi \ (path\ formulas)$$

- The interpretation of a CTL formula is defined over a Kripke structure (i.e, a state transition system).

Definition 1 (Kripke structure): A Kripke structure T is a quadruple $\langle S, S_0, R, L \rangle$, where:

1) S is a finite set of states.
2) S_0 is the set of initial states.
3) $R \subseteq S \times S$ is a a total transition relation, that is: $\forall s \in S\ \exists s' \in S$ such that $(s, s') \in R$
4) $L : S \rightarrow 2^{AP}$ labels each state with the set of atomic propositions that hold in that state.

Computation Tree Logic

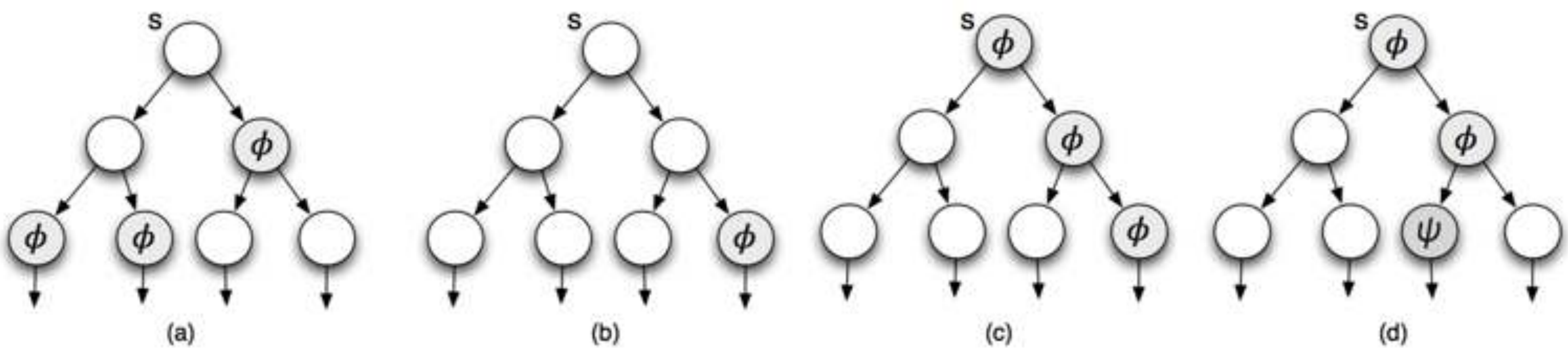


Fig. 1: (a) $T \models_s AF\phi$; (b) $T \models_s EF\phi$; (c) $T \models_s EG\phi$; (d) $T \models_s E[\phi U \psi]$

- It can be shown that any CTL formula can be written in terms of ¬, ∨, **EX**, **EG**, and **EU**

$$R^{-}(W) := \{s \in S \ : \ \exists s'(R(s,s') \wedge s' \in W)\}$$

$$[\![EX\phi]\!]_T = R^{-}([\![\phi]\!]_T)$$

greatest fixed point

$$[\![EG\phi]\!]_T = \nu_X([\![\phi]\!]_T \cap R^{-}(X))$$

monotonic predicate transformer

$$[\![E[\phi U \psi]]\!]_T = \mu_X([\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^{-}(X)))$$

least fixed point

MapReduce EX evaluation

$$[\![EX\phi]\!]_T = R^-([\![\phi]\!]_T)$$

Algorithm 2 MapReduce algorithm for evaluating $EX\phi$

function MAP(k, s)
    if $s \in [\![\phi]\!]_T$ **then**
        for $e \in R^-(s)$ **do**
            $emit(e, \perp)$
        end for
    end if
    $emit(k, s)$
end function
function REDUCE($k, list := [s_1, s_2, ...]$)
    if $\perp \in list$ **then**
        $s := s' \in list \ s.t. \ s' \neq \perp$
        $emit(k, s)$
    end if
end function

MapReduce EG evaluation

$$[\![EG\phi]\!]_T = \nu_X([\![\phi]\!]_T \cap R^-(X))$$

Algorithm 3 MapReduce for evaluating $EG\phi$

function MAP(k, s)
    if $s \in X$ **then**
        for $e \in R^-(s)$ **do**
            $emit(e, \perp)$
        end for
    end if
    if $s \in [\![\phi]\!]_T$ **then**
        $emit(k, s)$
    end if
end function
function REDUCE($k, list := [s_1, s_2, ...]$)
    if $\perp \in list \ \wedge \ (s \neq \perp \in list)$ **then**
        $emit(k, s)$
    end if
end function

MapReduce EU evaluation

$$[\![E[\phi U \psi]]\!]_T = \mu_X([\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^-(X)))$$

Algorithm 4 MapReduce algorithm for evaluating $E[\phi U \psi]$

function MAP(k, s)
    if $s \in X$ **then**
        for $e \in R^-(s)$ **do**
            $emit(e, \perp)$
        end for
    end if
    if $s \in [\![\phi]\!]_T \vee s \in [\![\psi]\!]_T$ **then**
        $emit(k, s)$
    end if
end function
function REDUCE$(k, list := [s_1, s_2, ...])$
    $s := s' \in list \ s.t. \ s' \neq \perp$
    if $(\perp \in list \ \wedge \ s \neq null) \vee (s \in [\![\psi]\!]_T)$ **then**
        $emit(k, s)$
    end if
end function

CTL experiments

- Models:
  - Shared memory ($\sim 10^6$ states, $\sim 10^7$ transitions)
  - Dekker ($\sim 10^7$ states, $\sim 10^8$ transitions)
  - Simple load balancing ($\sim 10^8$ states, $\sim 10^9$ transitions)

Table 1: Shared memory report

property	$\lvert [\![property]\!]_T \rvert$	# machines	time (s)
$EX[\phi]$	2.135×10^5	1	70
$EX[\phi]$	2.135×10^5	2	67
$EX[\phi]$	2.135×10^5	4	50
$EX[\phi]$	2.135×10^5	8	38
$EG[\psi]$	0	1	67
$EG[\psi]$	0	2	55
$EG[\psi]$	0	4	58
$E[\omega\ U \rho]$	1.831×10^6	1	1898
$E[\omega\ U \rho]$	1.831×10^6	2	1124
$E[\omega\ U \rho]$	1.831×10^6	4	839
$E[\omega\ U \rho]$	1.831×10^6	8	564
$E[\omega\ U \rho]$	1.831×10^6	16	509

Table 2: Dekker report

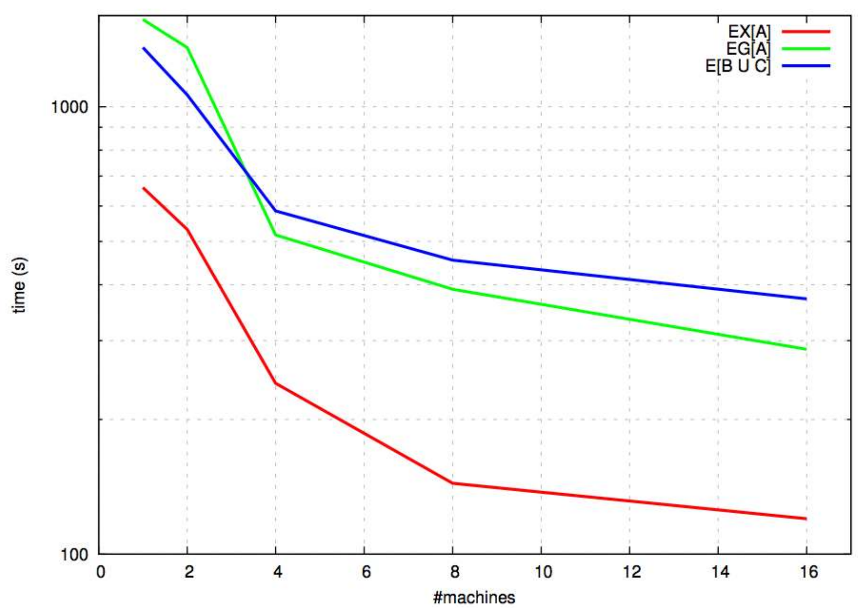
property	$\lvert [\![property]\!]_T \rvert$	# machines	time (s)
$EX[\phi]$	1.153×10^7	1	660
$EX[\phi]$	1.153×10^7	2	532
$EX[\phi]$	1.153×10^7	4	241
$EX[\phi]$	1.153×10^7	8	144
$EX[\phi]$	1.153×10^7	16	120
$EG[\psi]$	7.405×10^6	1	1567
$EG[\psi]$	7.405×10^6	2	1356
$EG[\psi]$	7.405×10^6	4	517
$EG[\psi]$	7.405×10^6	8	391
$EG[\psi]$	7.405×10^6	16	287
$E[\omega\ U \rho]$	5.767×10^6	1	1357
$E[\omega\ U \rho]$	5.767×10^6	2	1063
$E[\omega\ U \rho]$	5.767×10^6	4	585
$E[\omega\ U \rho]$	5.767×10^6	8	454
$E[\omega\ U \rho]$	5.767×10^6	16	372

Table 3: Simple load balancing report

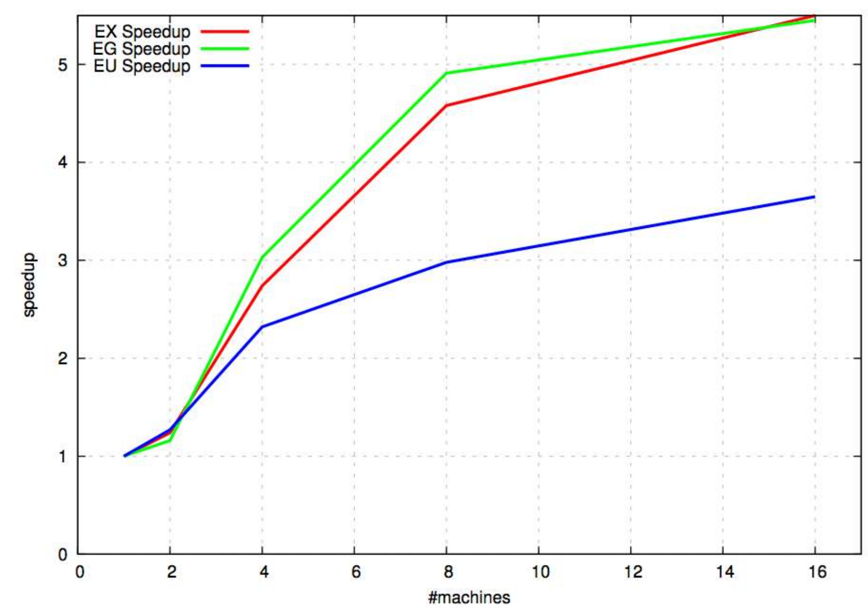
property	$\lvert [\![property]\!]_T \rvert$	# machines	time (s)
$EX[\phi]$	1.716×10^8	1	2908
$EX[\phi]$	1.716×10^8	2	2401
$EX[\phi]$	1.716×10^8	4	937
$EX[\phi]$	1.716×10^8	8	693
$EX[\phi]$	1.716×10^8	16	251
$EG[\psi]$	4.060×10^8	1	21678
$EG[\psi]$	4.060×10^8	2	17147
$EG[\psi]$	4.060×10^8	4	6525
$EG[\psi]$	4.060×10^8	8	2983
$EG[\psi]$	4.060×10^8	16	1226
$E[\omega\ U \rho]$	7.524×10^7	1	1821
$E[\omega\ U \rho]$	7.524×10^7	2	1714
$E[\omega\ U \rho]$	7.524×10^7	4	602
$E[\omega\ U \rho]$	7.524×10^7	8	377
$E[\omega\ U \rho]$	7.524×10^7	16	203

CTL experiments

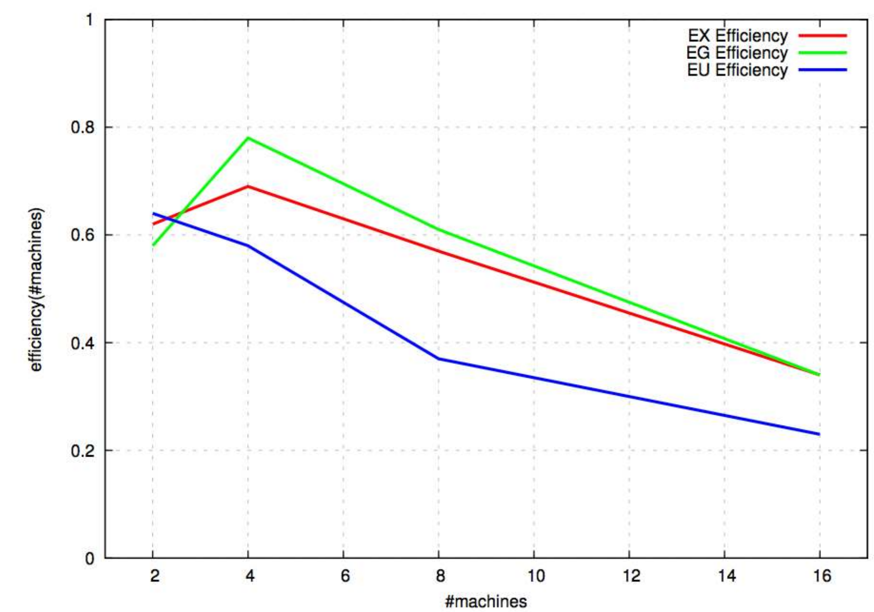
(a) Dekker model checking time



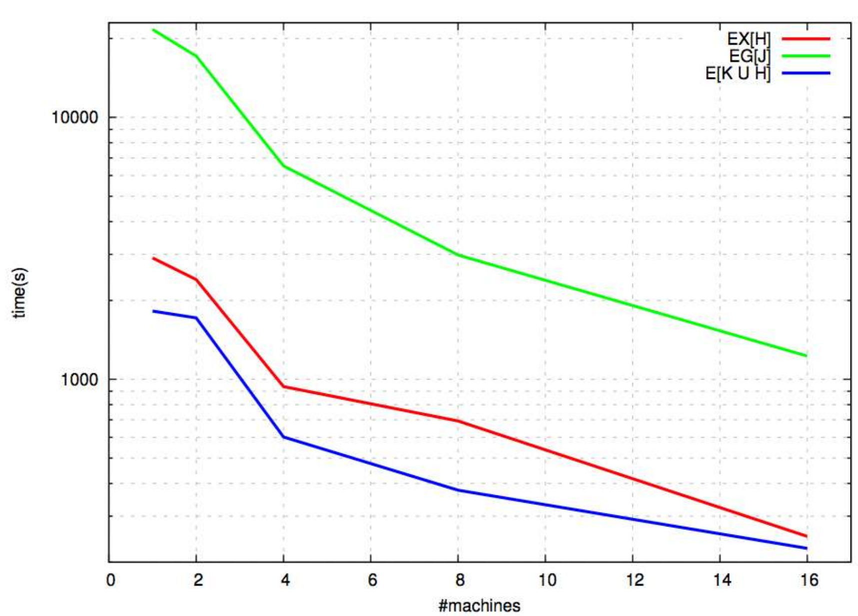
(b) Dekker Speedup



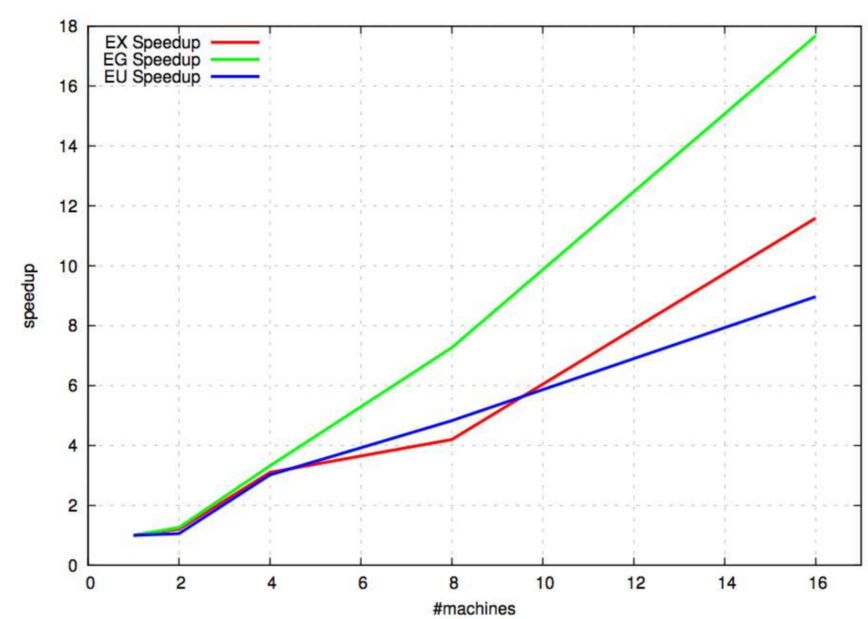
(c) Dekker efficiency



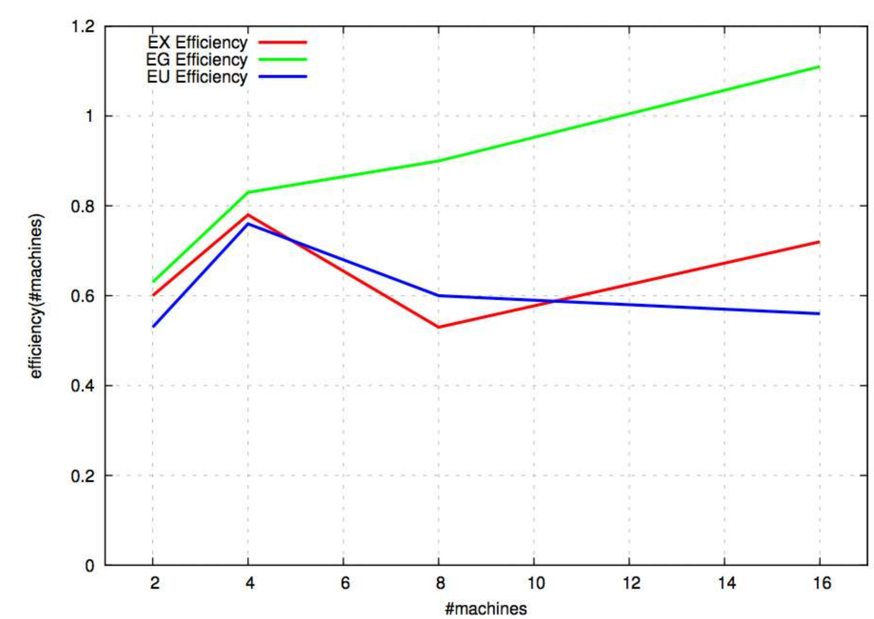
(d) Simple-lb model checking time



(e) Simple-lb Speedup



(f) Simple-lb efficiency



Conclusion

- MaRDiGraS + CTL verification in the cloud allow users to implement distributed reachability graph builders and verification tools for different formalisms without care about all non functional aspects.
    - They apply techniques typically used by the big data community and so far poorly explored for this kind of issues.
- We believe that this work could be a first step towards a synergy between two very different, but related communities: the formal verification community and the big data community.
- Open Questions
    - How it can be optimized when the remaining set gets very small?
    - How to choose the optimal threshold dynamically?
    - Are there classes of formalisms for which this approach cannot be used? And how can we adapt it to these classes?
    - ... ?

Planned Work

- Development of a technique for tackling topologically infinite TB net models
  - computation of minimal coverability sets (so far unexplored)
  - this provides a means to decide several important properties also for real time systems:
    - *coverability*: is it possible to reach a marking dominating a given marking?
    - *boundedness*: is the set of reachability markings finite?
    - *place boundedness*: is it possible to bound the number of tokens in a given place?
    - *semi-liveness*: is there a reachable marking in which a given transition is enabled?

References

- Matteo Camilli. 2012. Petri nets state space analysis in the cloud. In Proceedings of the 2012 International Conference on Software Engineering (ICSE 2012). IEEE Press, Piscataway, NJ, USA, 1638-1640.

- Carlo Bellettini, Matteo Camilli, Lorenzo Capra, and Mattia Monga. 2012. Symbolic State Space Exploration of RT Systems in the Cloud. In Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '12). IEEE Computer Society, Washington, DC, USA, 295-302. DOI=10.1109/SYNASC.2012.18 http://dx.doi.org/10.1109/SYNASC.2012.18

- C. Bellettini, M. Camilli, L. Capra, and M. Monga. Mardigras: Simplified building of reachability graphs on large clusters. In P. Abdulla and I. Potapov, editors, Reachability Problems, volume 8169 of LNCS, pages 83–95. Springer Berlin Heidelberg, 2013.

- Matteo Camilli. 2014. Formal verification problems in a big data world: towards a mighty synergy. In Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014). ACM, New York, NY, USA, 638-641. DOI=10.1145/2591062.2591088 http://doi.acm.org/10.1145/2591062.2591088