Parallel Analysis of Parallelism

Verifying Concurrent Software System Designs Using GPUs

GTC 2015

Anton Wijs

Correctness of Concurrent Systems

- Distributed, concurrent systems common-place, but very difficult to develop
  - **network applications, communication protocols, multi-threaded applications**
- Systems may contain bugs such as *deadlocks* and *livelocks*
  - *Deadlock:* computation not finished, but system cannot progress
  - *Livelock:* system repeats computation steps without progressing

- Given a model of a concurrent system, these, and other functional properties can be checked using **model checking**
  - All states in which the system (design) can end up are inspected
  - It is **automatic**
  - Provides useful feedback (**counter-examples**)

Model Checking

- Exhaustively interpret all potential functional behaviour of a model of a (concurrent) system, and automatically check whether that behaviour meets a given specification
    - **Deadlock freedom**
    - **Race freedom**
    - … *safety* and *liveness* properties

Safety:
"two processes can never simultaneously access the same critical section"
Liveness:
"When a message has been sent, it is eventually received"

- Formal models describe **hardware or software designs**, requirements specified using **temporal logics** (*CTL, LTL, mu-calculus*)

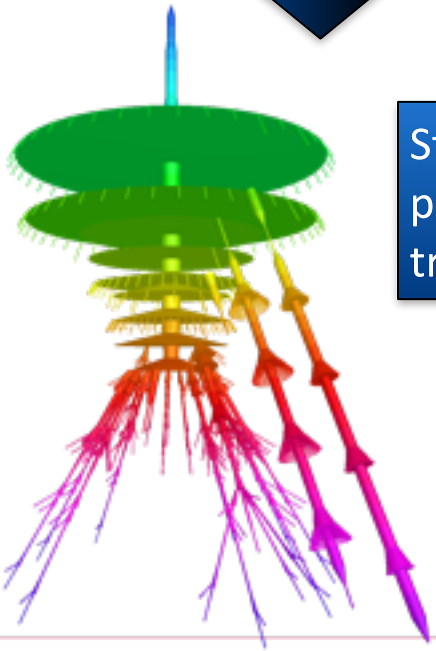
2007: pioneers **E.M. Clarke, J. Sifakis, E.A. Emerson** (early 80's) receive *Turing award*

Model Checking

(Dining Philosophers Problem)

(Deadlock freedom as mu-calculus formula)

[True] <True> True



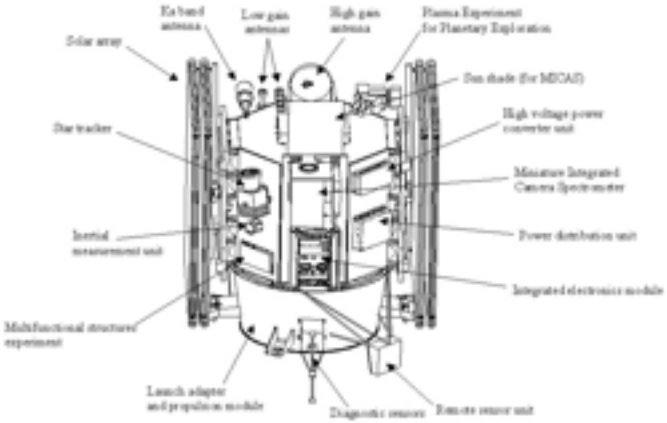
State Space is a map showing all possible system states and transitions between them

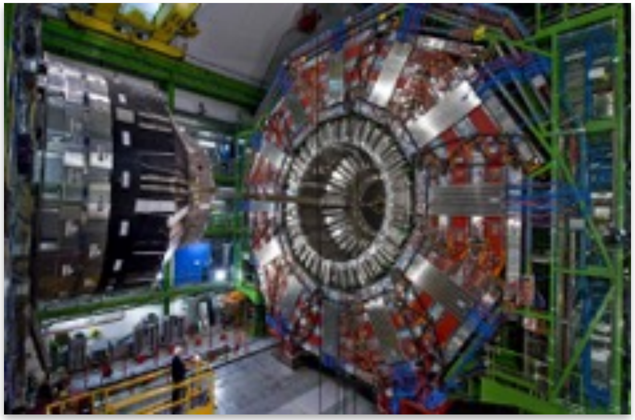
Dining Philosophers System can deadlock!

(Produced with the LTSview tool of the mCRL2 toolset)

Model Checking Success Stories

- **Deadlocks** detected in airline reservation systems
- Modern e-commerce protocols have been verified
- Studies of IEEE standards for in-house communication of appliances has led to **significant improvements**
- Errors found in *Deep Space 1 spacecraft controller* model ('98)
- **TU/e** with **mCRL2**: Control software of the Compact Muon Solenoid Experiment at the *Large Hadron Collider*: 27.500 finite state machines, **livelock** and **reachability bugs** found

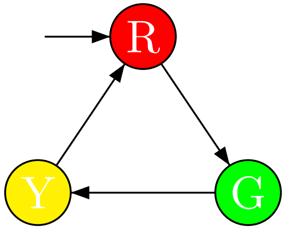

Ka band antenna
Low gain antennas
High gain antenna
Plasma Experiment for Planetary Exploration
Solar array
Sun shade (for MICAS)
Star tracker
High voltage power converter unit
Miniature Integrated Camera Spectrometer
Inertial measurement unit
Power distribution unit
Integrated electronics module
Multifunctional structures experiment
Launch adapter and propulsion module
Diagnostic sensors
Remote sensor unit

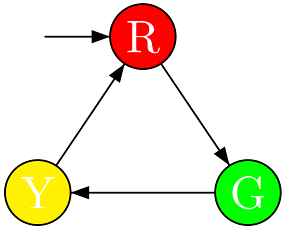


Drawback: state space explosion

Running example: Traffic light control system

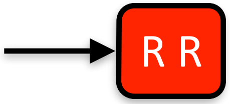
Specification
Controller
Timer

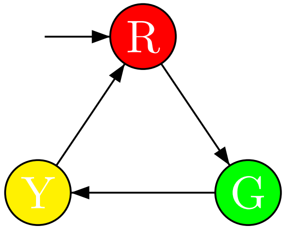


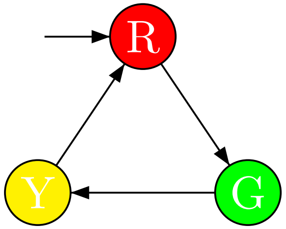


Drawback: state space explosion

Running example: Traffic light control system

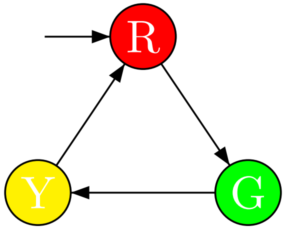


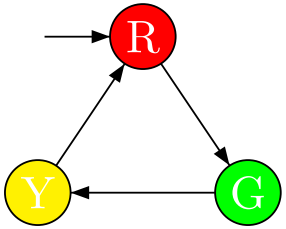


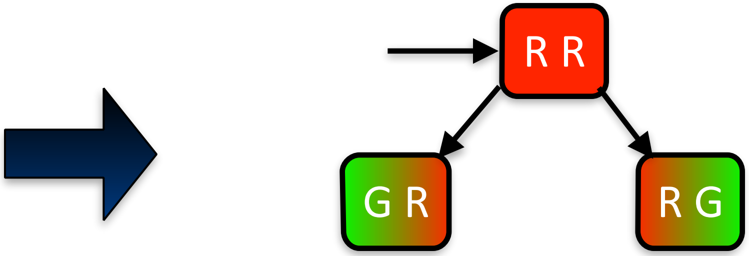


Drawback: state space explosion

Running example: Traffic light control system

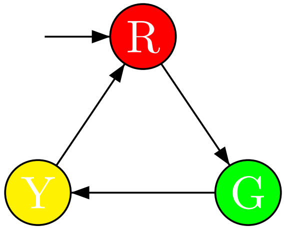


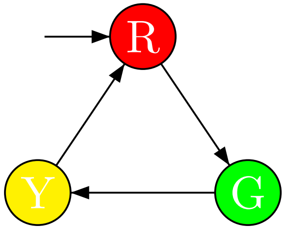


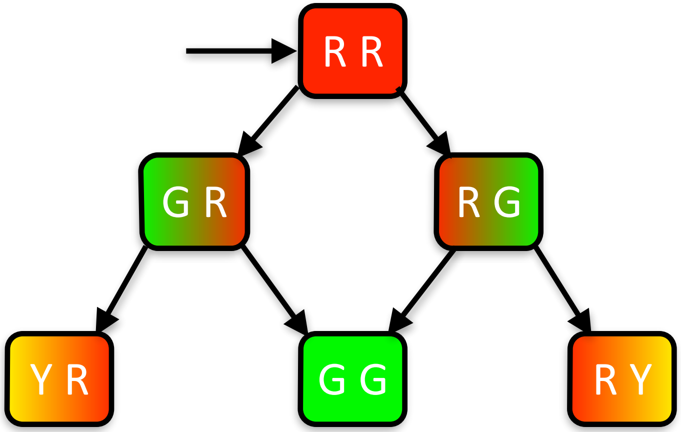


Drawback: state space explosion

Running example: Traffic light control system

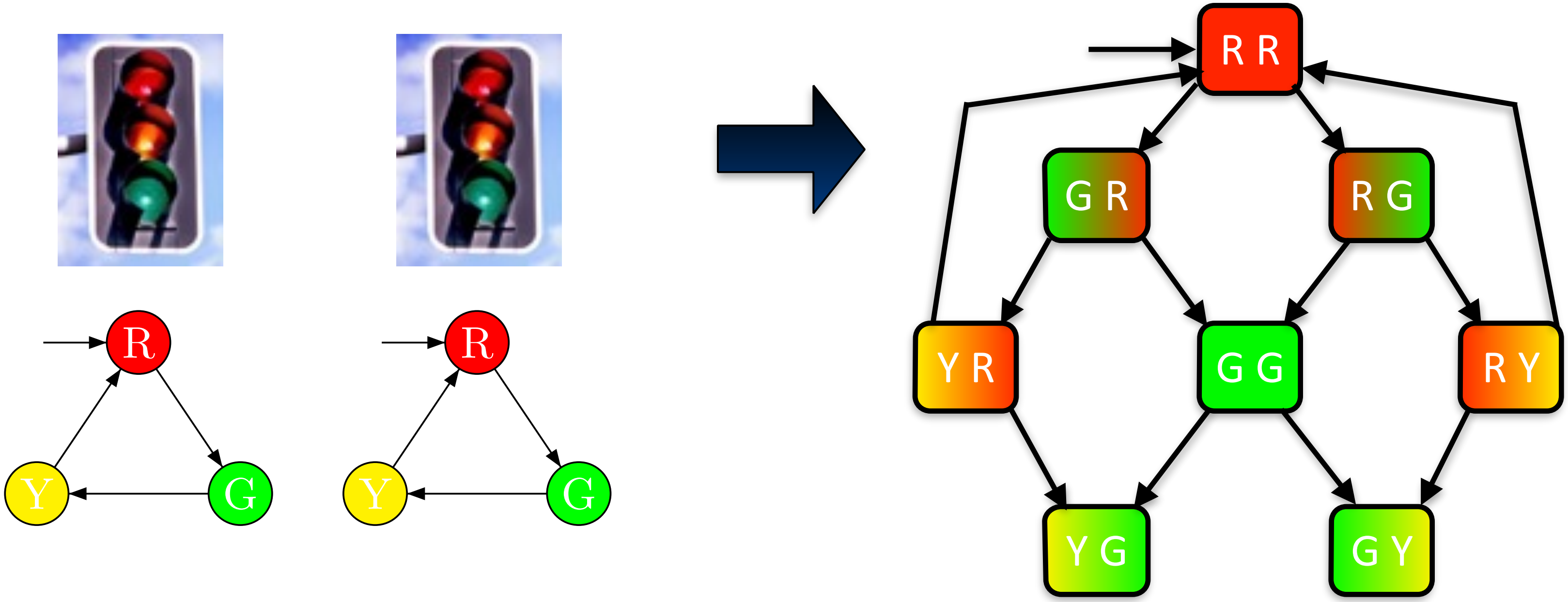






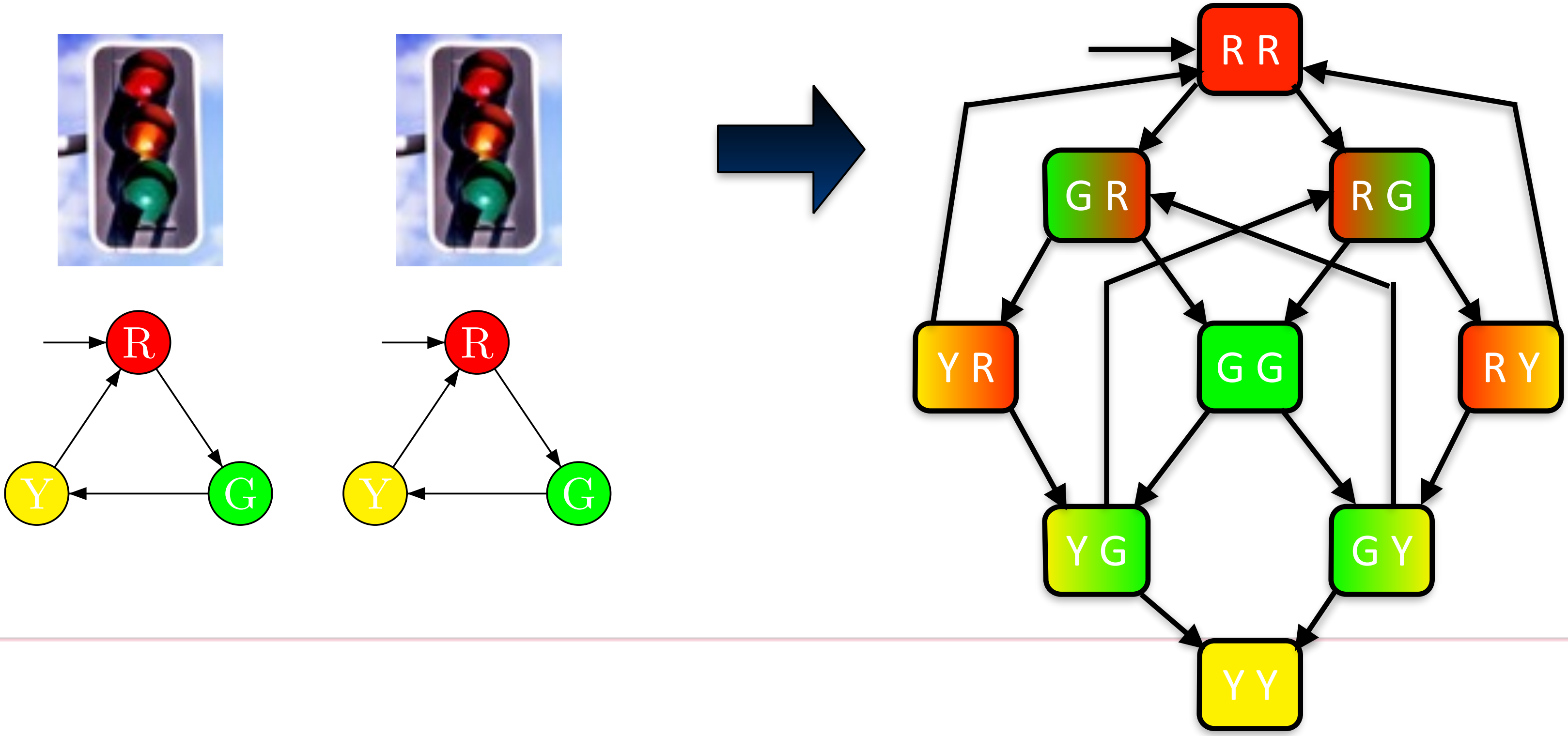
Drawback: state space explosion

Running example: Traffic light control system



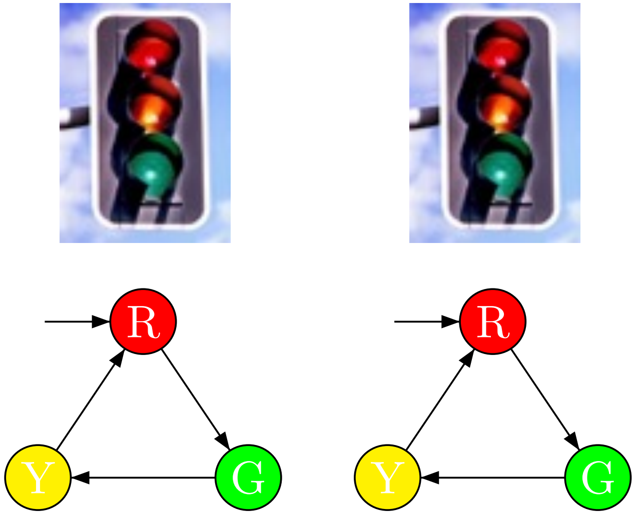
Drawback: state space explosion

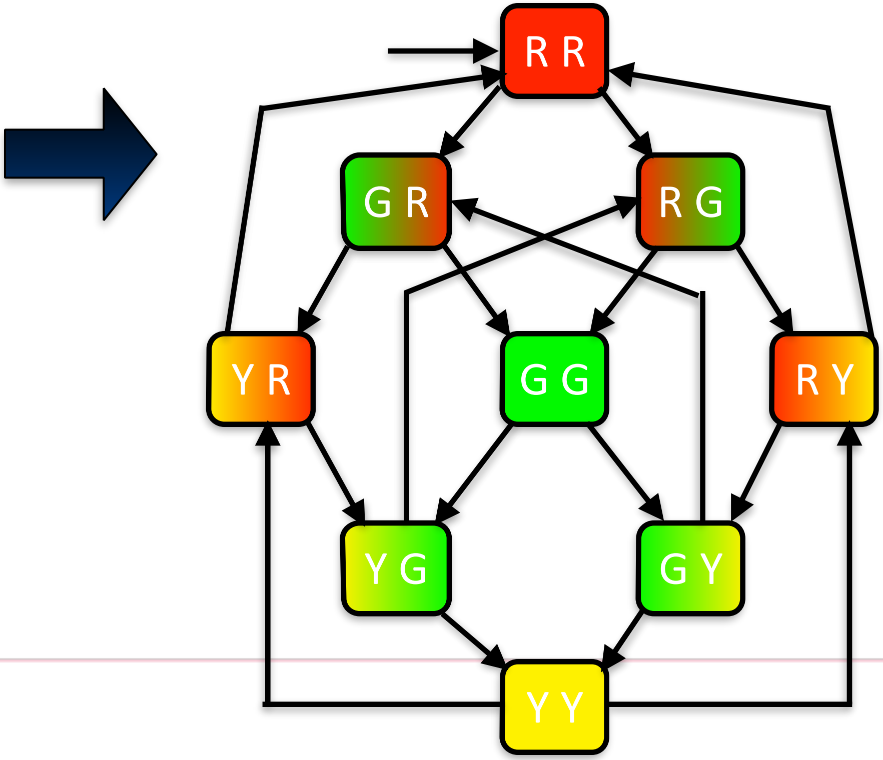
Running example: Traffic light control system



Drawback: state space explosion

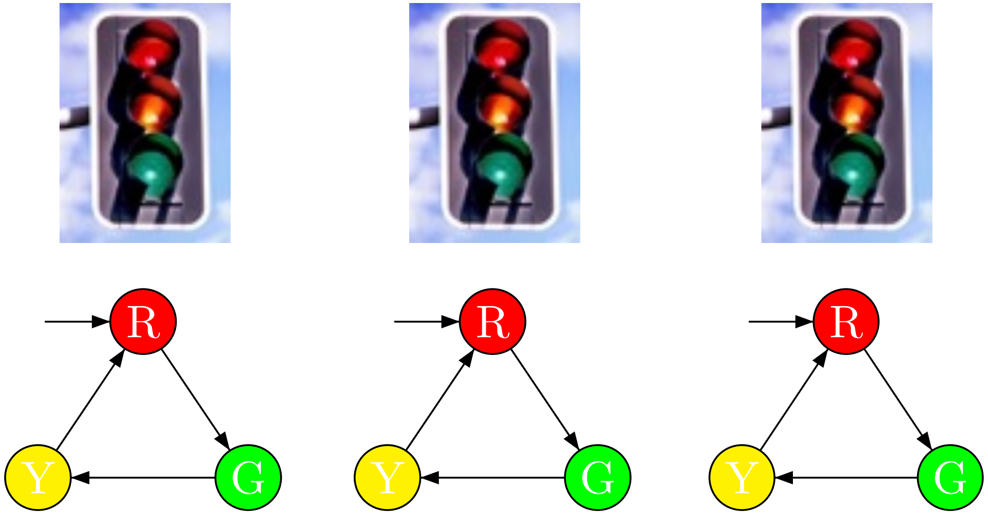
Running example: Traffic light control system





Drawback: state space explosion

Running example: Traffic light control system

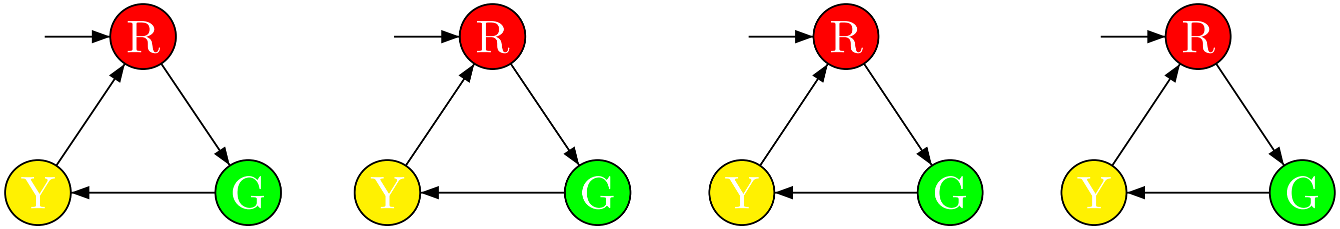


27 states

Drawback: state space explosion

Running example: Traffic light control system

81 states



Drawback: state space explosion

Running example: Traffic light control system

13 traffic lights	**1.59 million states**
14 traffic lights	**4.78 million states**
15 traffic lights	**14.35 million states**
16 traffic lights	**43.05 million states**
17 traffic lights	**129.14 million states**

Linear growth of model leads to ***exponential*** growth of state space!

Current state-of-the-art (explicit-state) model checking: reason about ~ ***3 billion states***

- Common operations in model checking:
  - **Generating** state spaces (+ **on-the-fly checking properties**)
  - **Analysing the structure** of states spaces (e.g., **strongly connected components,** relevant for more complex properties)
  - Comparing states and transitions
    - **Minimising** state spaces for more efficient analysis

- **Can GPUs be used for this?**

- **Yes, but far from trivially**

Harnessing the power of GPUs for model checking

Anton Wijs
Eindhoven University of Technology

joint work with

Dragan Bošnački
Eindhoven University of Technology

Stefan Edelkamp & Damian Sulewski
University of Bremen

Joost-Pieter Katoen
RWTH Aachen University

On-The-Fly State Space Exploration

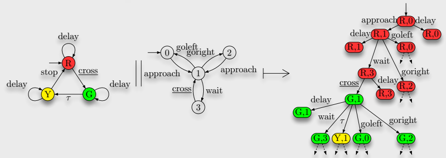
Construct a state space, given a model of a concurrent system [3]
Model = set of interacting finite-state Labelled Transition Systems

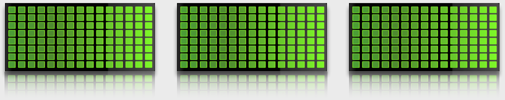
New hash-table design for GPUs, with fine-grained parallelism
Elements are placed in buckets using *warp-the-line* technique

Threads work in groups to generate state successors
Parallelism at state-level

Block-local shared memory used for state caches
Local duplicate detection reduces global hash table access

Work forwarding per block from one search iteration to the next
Speeds up fetching new work for the next iteration





10-100x speedup

State Space Structural Analysis

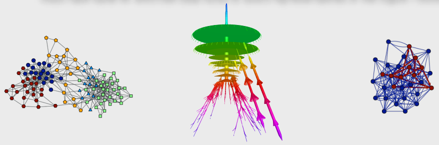
Decompose explicit graph into Strongly Connected Components
&
Decompose graph of Markov Decision Process into Maximal End Components [5]

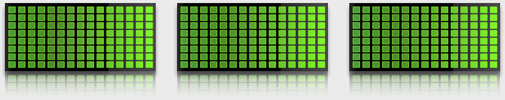
Decompositioning based on Forward/Backward Breadth-First Search
Novel combined forward/backward thread kernel with *trimming* of trivial SCCs

Check equivalence of states for state space comparison and minimisation [2]
Efficiently checks *strong* and *branching* bisimilarity of states

Equivalence determined via many-core partition refinement
Bisimilar states end up in the same block in final partition

Both operations use a new pivot selection procedure for each region / block
Uses hash table for enforced data races to select representatives of the region / block





10-79x speedup

Probability Computations

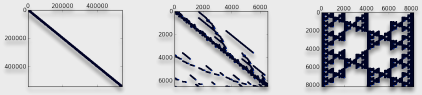
Perform numerical computations for probabilistic model checking [1, 4]
Needed to check if a probabilistic property holds in a discrete or continuous time Markov Chain

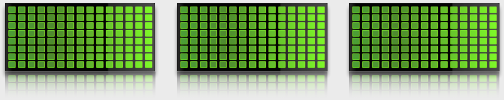
Solving systems of linear equations and performing matrix-vector multiplication
Parallel matrix-vector multiplication used in Jacobi method for solving equation systems

Parallel termination checking achieves significant speedup
Fast checking if next iteration is needed

Novel restructuring of input ensures coalesced memory access by threads
Faster reading of input reduces multiplication run time up to four times

States / transitions are grouped in segments of 16 and 32 states
Coincides with a half and a full *warp* of threads





20-35x speedup

References

[1] ***Parallel Probabilistic Model Checking on General Purpose Graphics Processors***
D. Bošnački, S. Edelkamp, D. Sulewski, and A.J. Wijs
International Journal on Software Tools for Technology Transfer 13(1) 21-35 (2011)

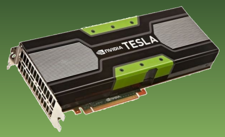
[2] ***GPU Accelerated Strong and Branching Bisimilarity Checking***
A.J. Wijs
in *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, accepted for publication (2015)

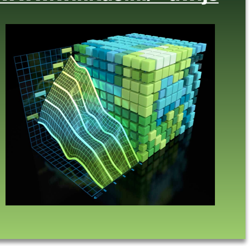
[3] ***GPUexplore: Many-Core On-The-Fly State Space Exploration Using GPUs***
A.J. Wijs and D. Bošnački
in *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of LNCS, pp. 233-247 (2014)

[4] ***Improving GPU Sparse Matrix-Vector Multiplication for Probabilistic Model Checking***
A.J. Wijs and D. Bošnački
in *Proceedings of the 19th International SPIN Workshop on Model Checking of Software (SPIN'12)*, volume 7385 of LNCS, pp. 98-116 (2012)

[5] ***GPU-Based Graph Decomposition into Strongly Connected and Maximal End Components***
A.J. Wijs, J.-P. Katoen, and D. Bošnački
in *Proceedings of the 26th International Conference on Computer Aided Verification (CAV'14)*, volume 8559 of LNCS, pp. 309-325 (2014)

Tools available at http://www.win.tue.nl/~awijs





The central image in "State Space Structural Analysis" shows the state space of a Bounded Retransmission Protocol model, and was created using the LTSview tool of the mCRL2 toolset (http://www.mcrl2.org)

Harnessing the power of GPUs for model checking

Anton Wijs
Eindhoven University of Technology

joint work with

Dragan Bošnački
Eindhoven University of Technology

Stefan Edelkamp & Damian Sulewski
University of Bremen

Joost-Pieter Katoen
RWTH Aachen University

On-The-Fly State Space Exploration

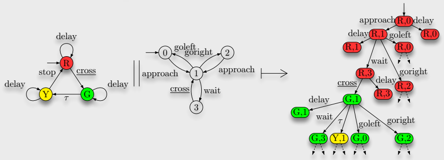
Construct a state space, given a model of a concurrent system [3]
Model = set of interacting finite-state Labelled Transition Systems

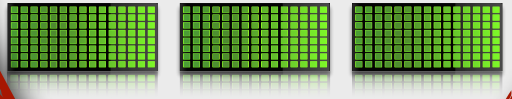
New hash-table design for GPUs, with fine-grained parallelism
Elements are placed in buckets using *warp-the-line* technique

Threads work in groups to generate state successors
Parallelism at state-level

Block-local shared memory used for state caches
Local duplicate detection reduces global hash table access

Work forwarding per block from one search iteration to the next
Speeds up fetching new work for the next iteration





10-100x speedup

State Space Structural Analysis

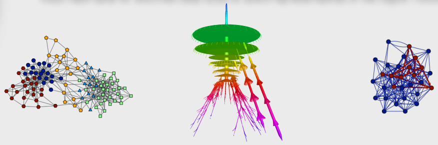
Decompose explicit graph into Strongly Connected Components
&
Decompose graph of Markov Decision Process into Maximal End Components [5]

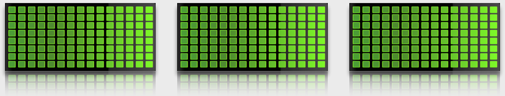
Decompositioning based on Forward/Backward Breadth-First Search
Novel combined forward/backward thread kernel with *trimming* of trivial SCCs

Check equivalence of states for state space comparison and minimisation [2]
Efficiently checks *strong* and *branching* bisimilarity of states

Equivalence determined via many-core partition refinement
Bisimilar states end up in the same block in final partition

Both operations use a new pivot selection procedure for each region / block
Uses hash table for enforced data races to select representatives of the region / block





10-79x speedup

Probability Computations

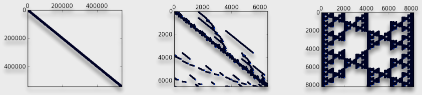
Perform numerical computations for probabilistic model checking [1, 4]
Needed to check if a probabilistic property holds in a discrete or continuous time Markov Chain

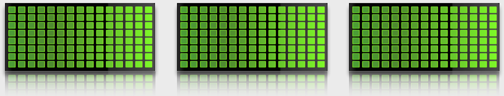
Solving systems of linear equations and performing matrix-vector multiplication
Parallel matrix-vector multiplication used in Jacobi method for solving equation systems

Parallel termination checking achieves significant speedup
Fast checking if next iteration is needed

Novel restructuring of input ensures coalesced memory access by threads
Faster reading of input reduces multiplication run time up to four times

States / transitions are grouped in segments of 16 and 32 states
Coincides with a half and a full *warp* of threads





20-35x speedup

References

[1] ***Parallel Probabilistic Model Checking on General Purpose Graphics Processors***
D. Bošnački, S. Edelkamp, D. Sulewski, and A.J. Wijs
International Journal on Software Tools for Technology Transfer 13(1) 21-35 (2011)

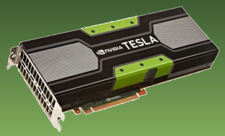
[2] ***GPU Accelerated Strong and Branching Bisimilarity Checking***
A.J. Wijs
in *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, accepted for publication (2015)

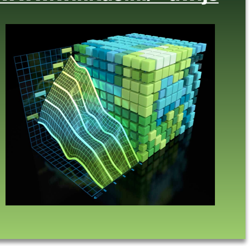
[3] ***GPUexplore: Many-Core On-The-Fly State Space Exploration Using GPUs***
A.J. Wijs and D. Bošnački
in *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of LNCS, pp. 233-247 (2014)

[4] ***Improving GPU Sparse Matrix-Vector Multiplication for Probabilistic Model Checking***
A.J. Wijs and D. Bošnački
in *Proceedings of the 19th International SPIN Workshop on Model Checking of Software (SPIN'12)*, volume 7385 of LNCS, pp. 98-116 (2012)

[5] ***GPU-Based Graph Decomposition into Strongly Connected and Maximal End Components***
A.J. Wijs, J.-P. Katoen, and D. Bošnački
in *Proceedings of the 26th International Conference on Computer Aided Verification (CAV'14)*, volume 8559 of LNCS, pp. 309-325 (2014)

Tools available at http://www.win.tue.nl/~awijs

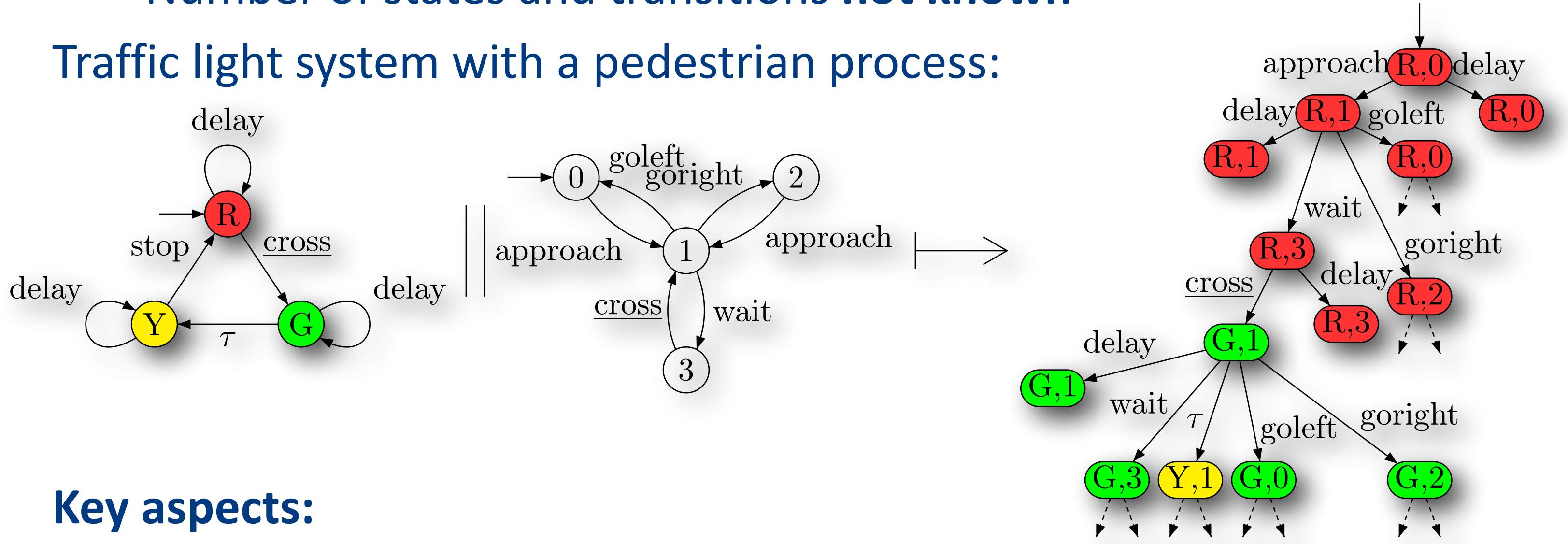




The central image in "State Space Structural Analysis" shows the state space of a Bounded Retransmission Protocol model, and was created using the LTSview tool of the mCRL2 toolset (http://www.mcrl2.org)

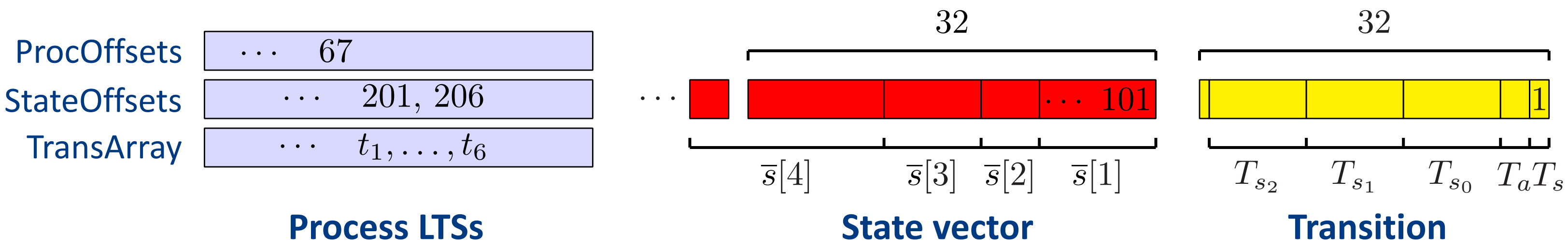
State Space Generation

- **Graph traversal** is a very important operation
  - Much work on GPU graph traversal (also at GTC 2015)
- **However**, for model checking, many approaches are not suitable, since the graph (state space) is not known **a priori**
  - Number of states and transitions **not known**
- Traffic light system with a pedestrian process:



- **Key aspects:**
  - Next-state computation (compute new state vectors)
  - Keeping track of which state vectors have been visited / explored

Model encoding

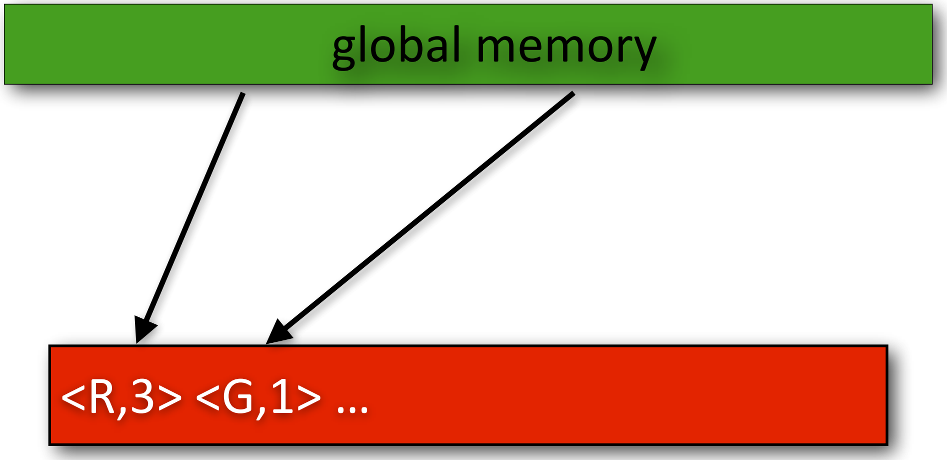


- In addition: synchronisation rules are encoded as bit sequences

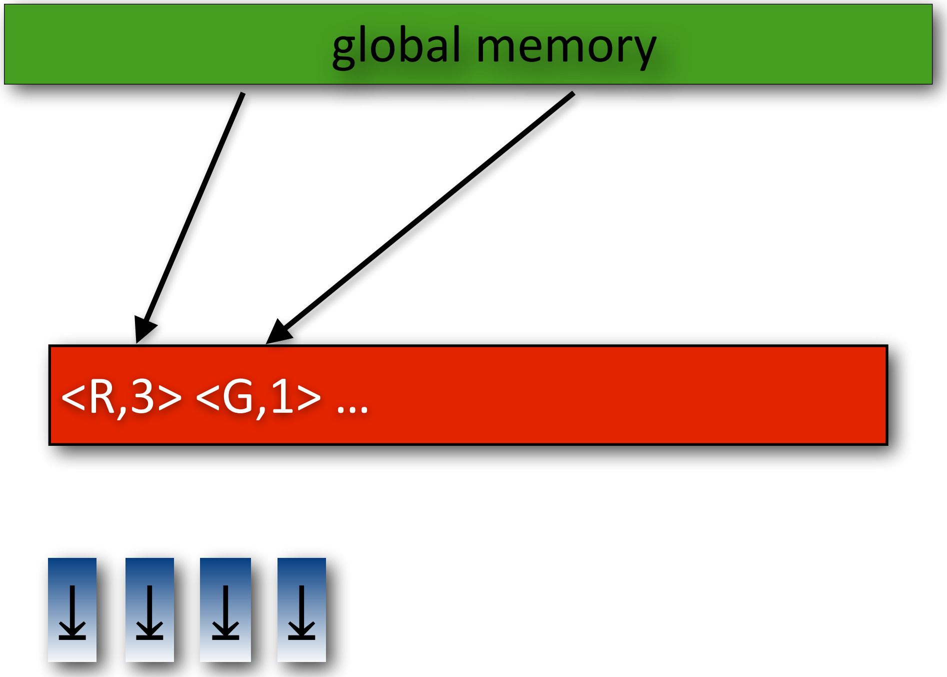
Input has a known size, and never changes: can be stored in **texture memory**

global memory

- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



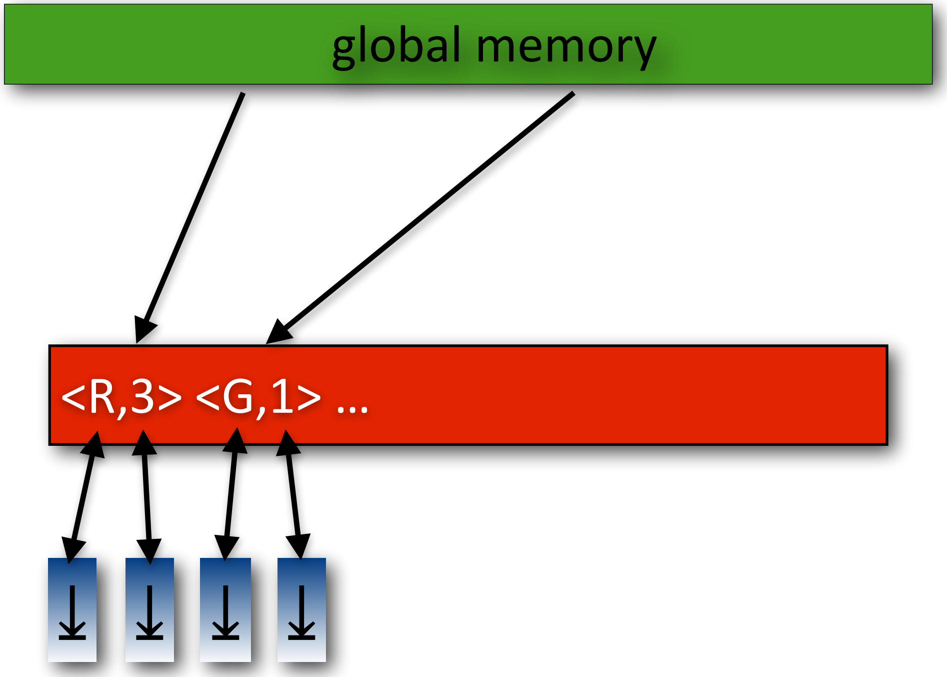
- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



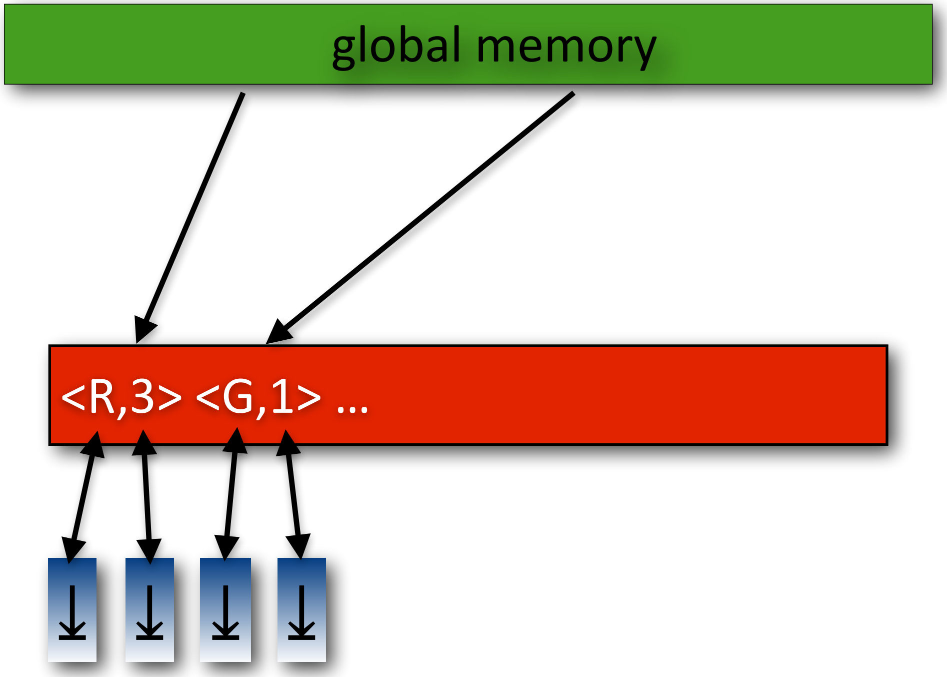
global memory

<R,3> <G,1> …

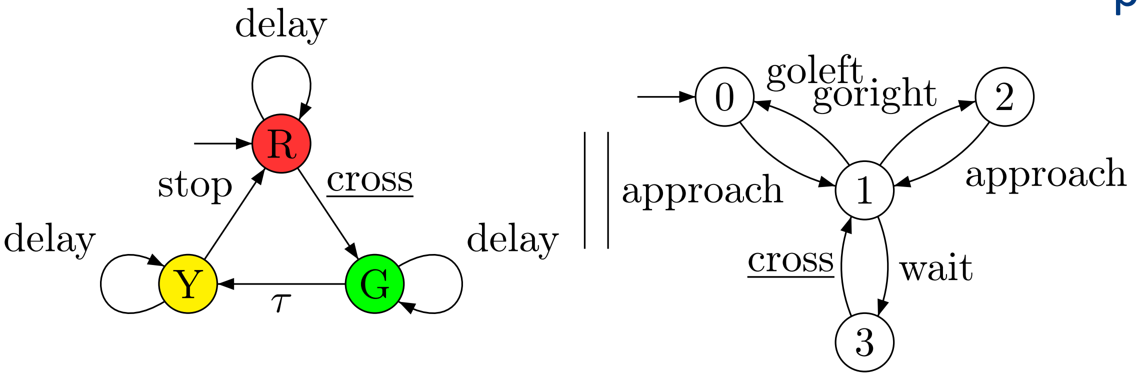
- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress

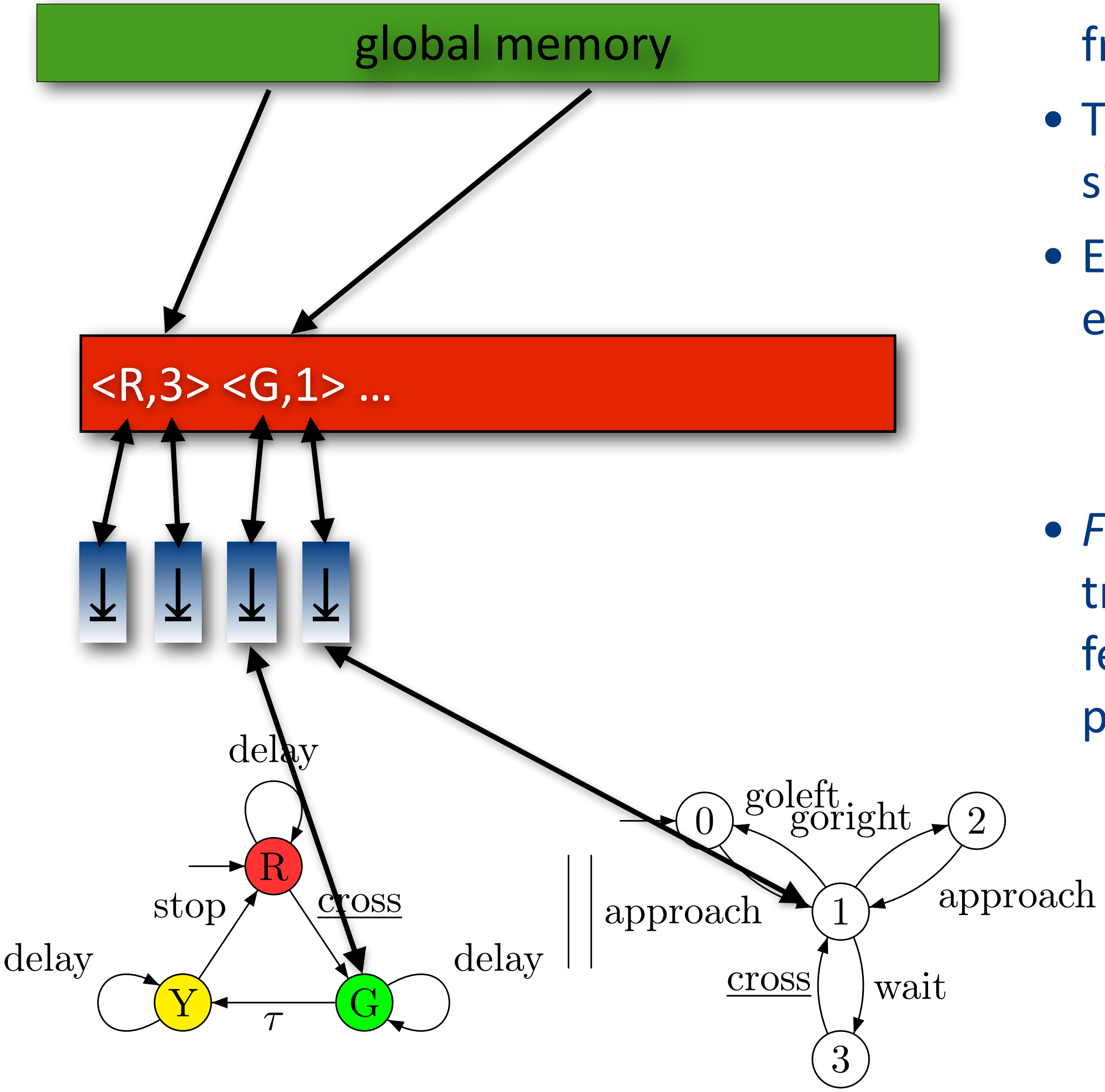


- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
    - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress

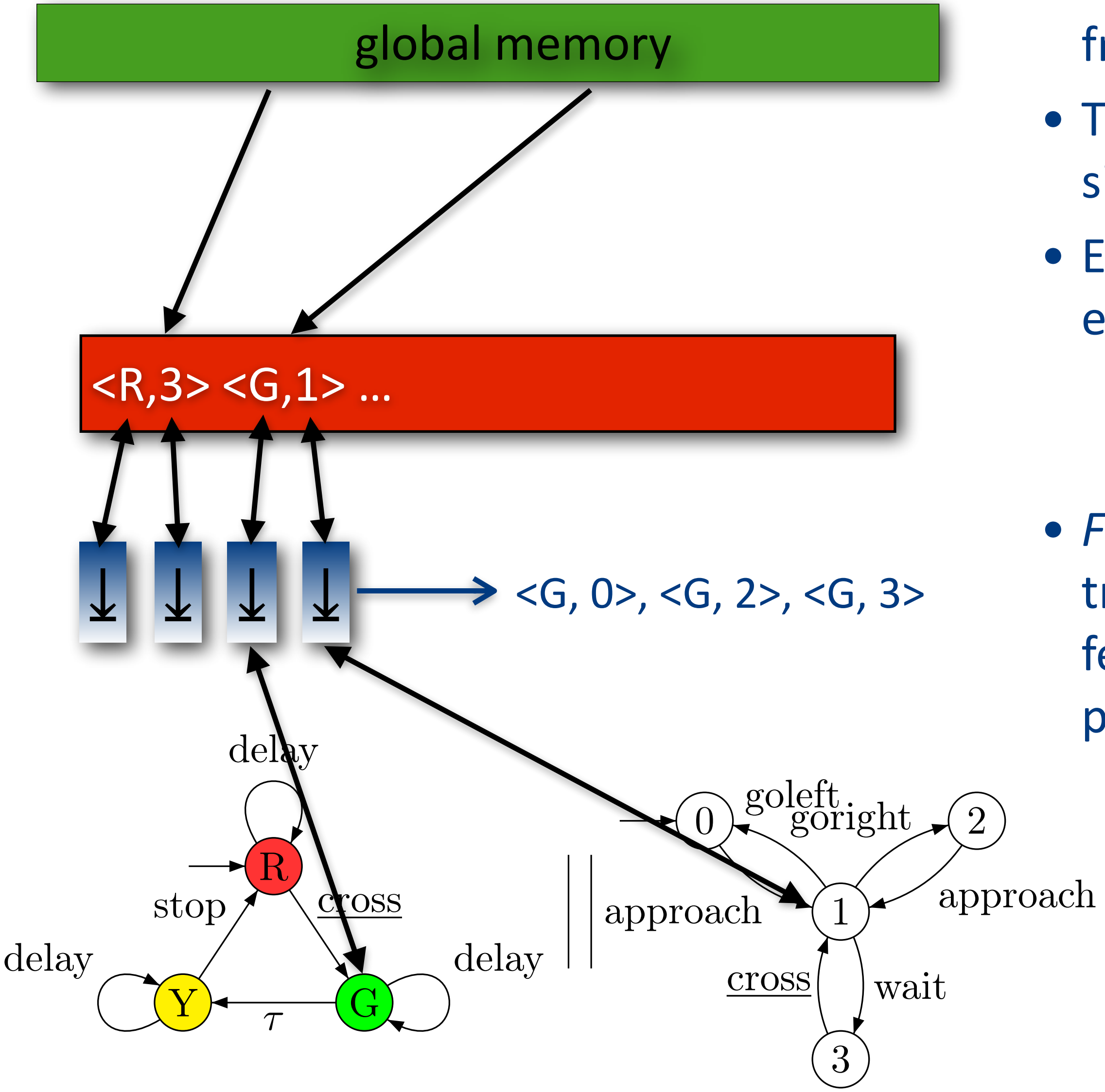


- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress

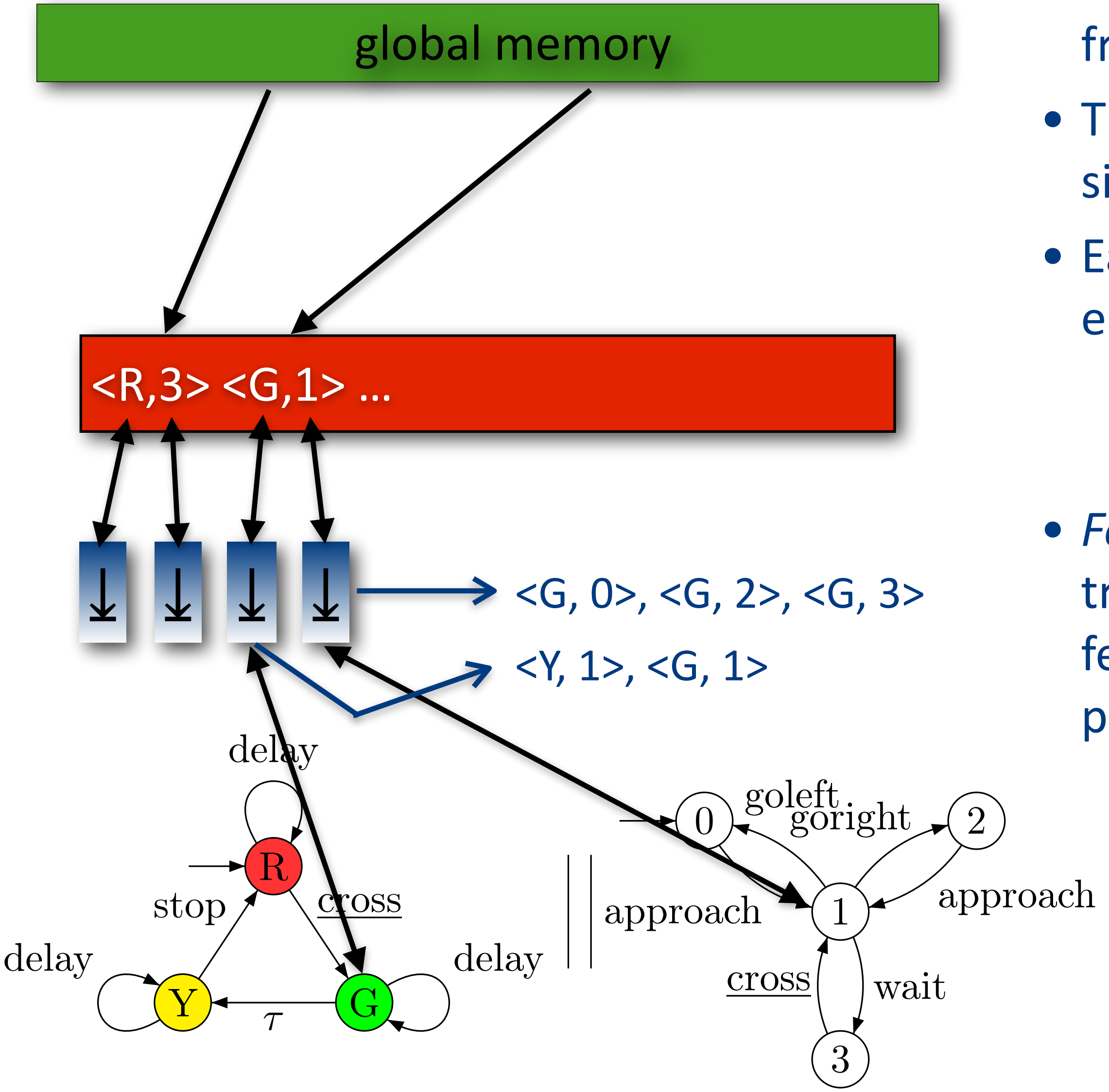




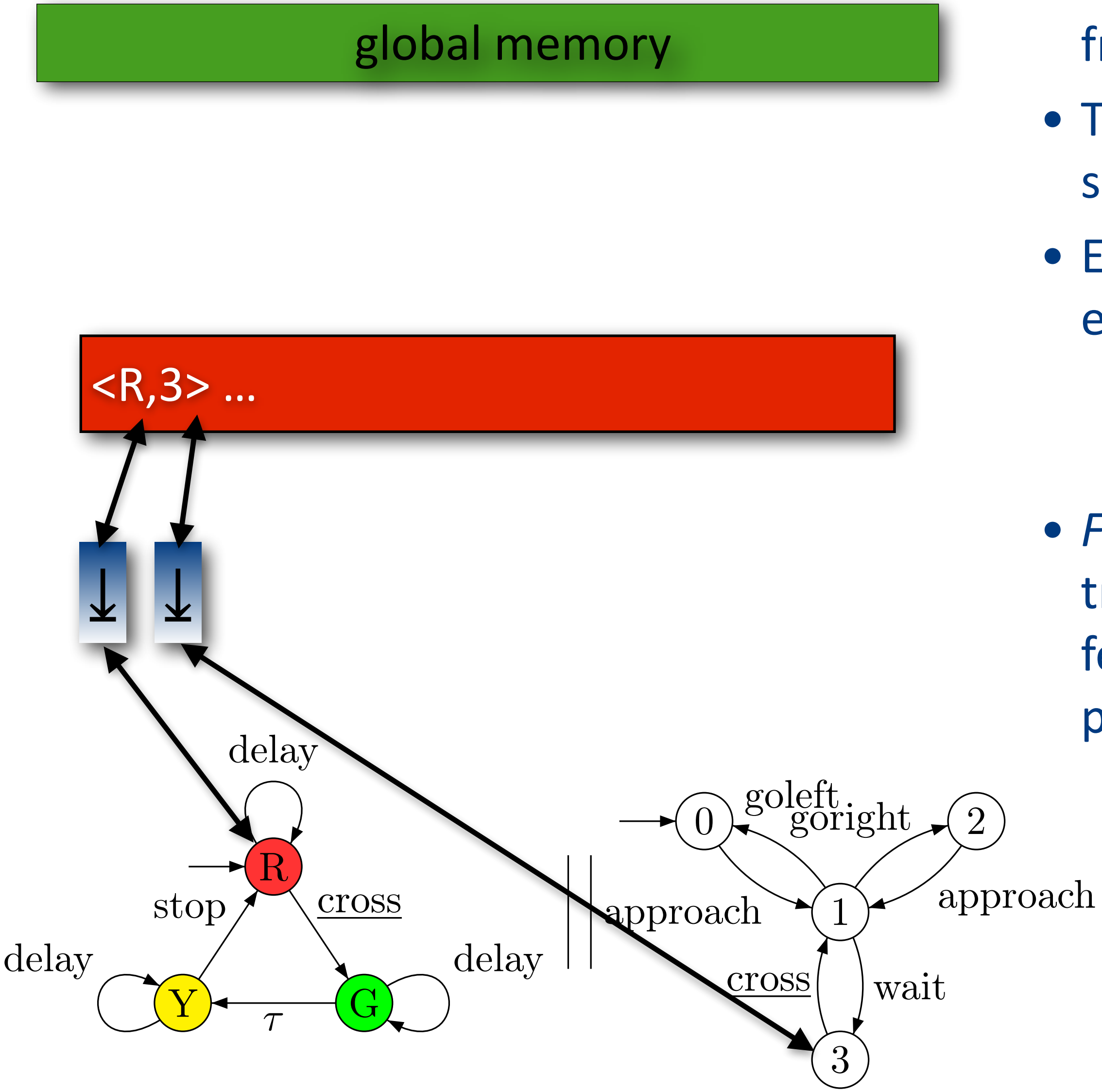
- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
    - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



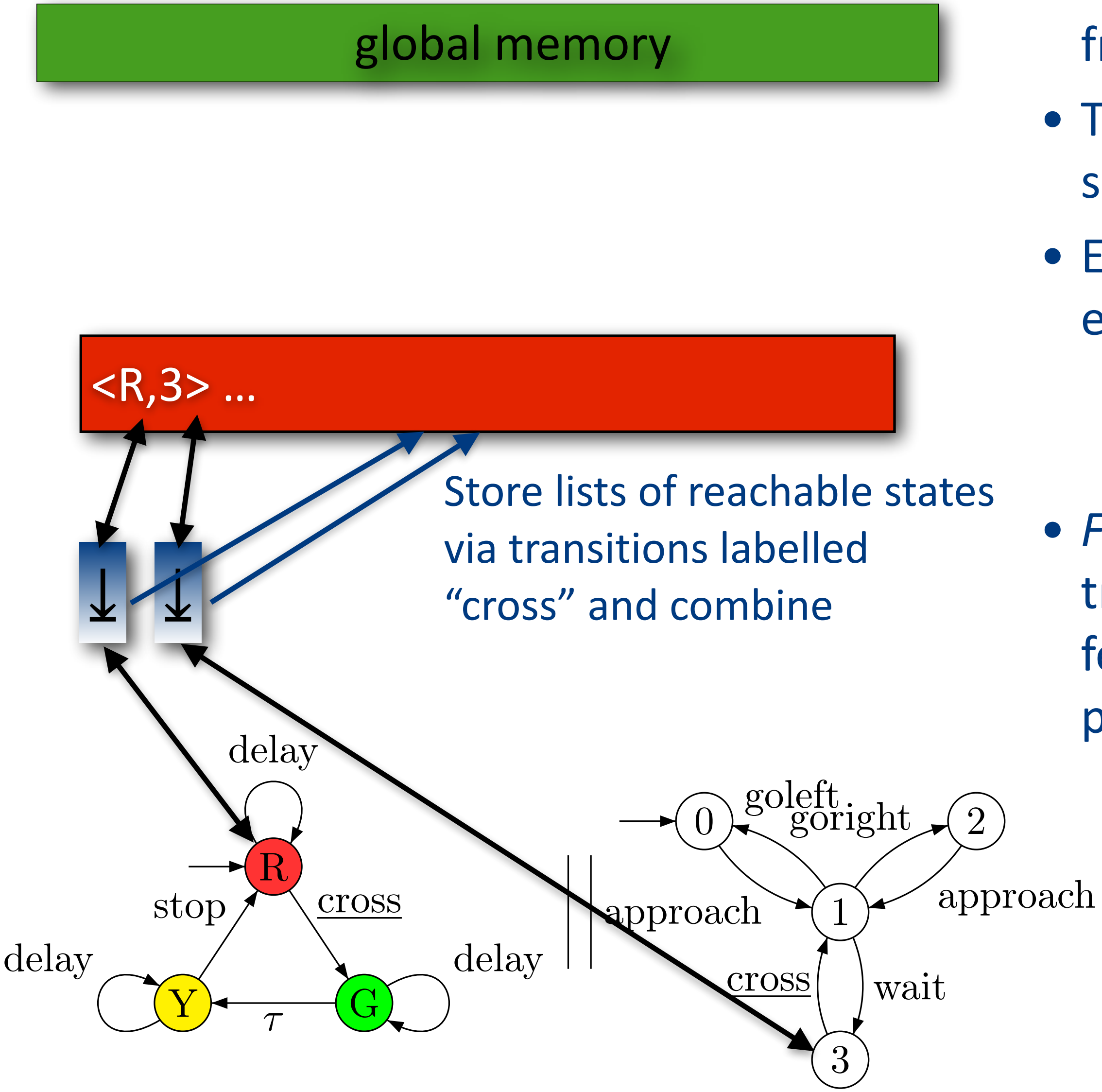
- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



global memory

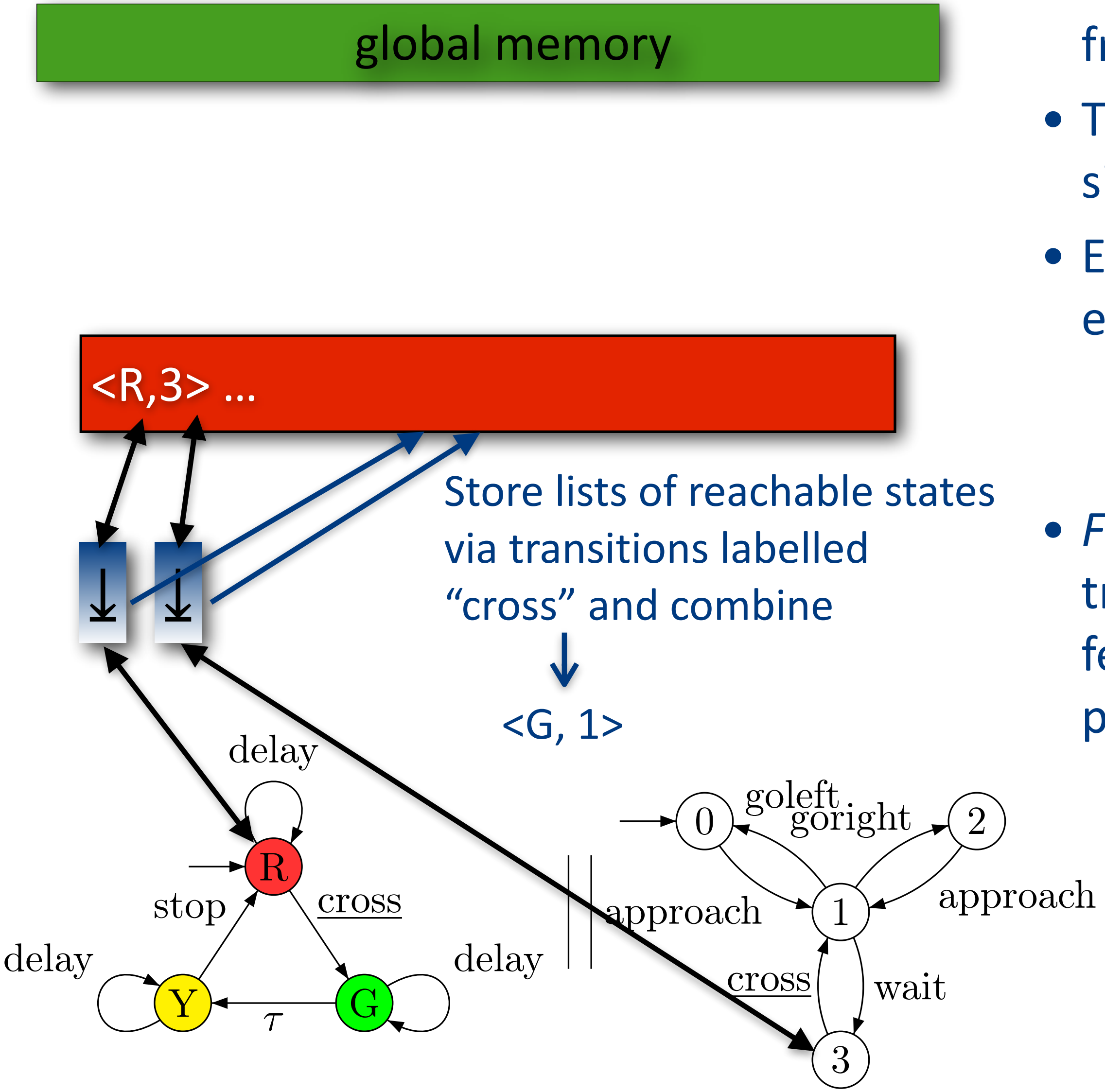
<R,3> ...

- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
    - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



Store lists of reachable states via transitions labelled “cross” and combine

- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
    - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress



global memory

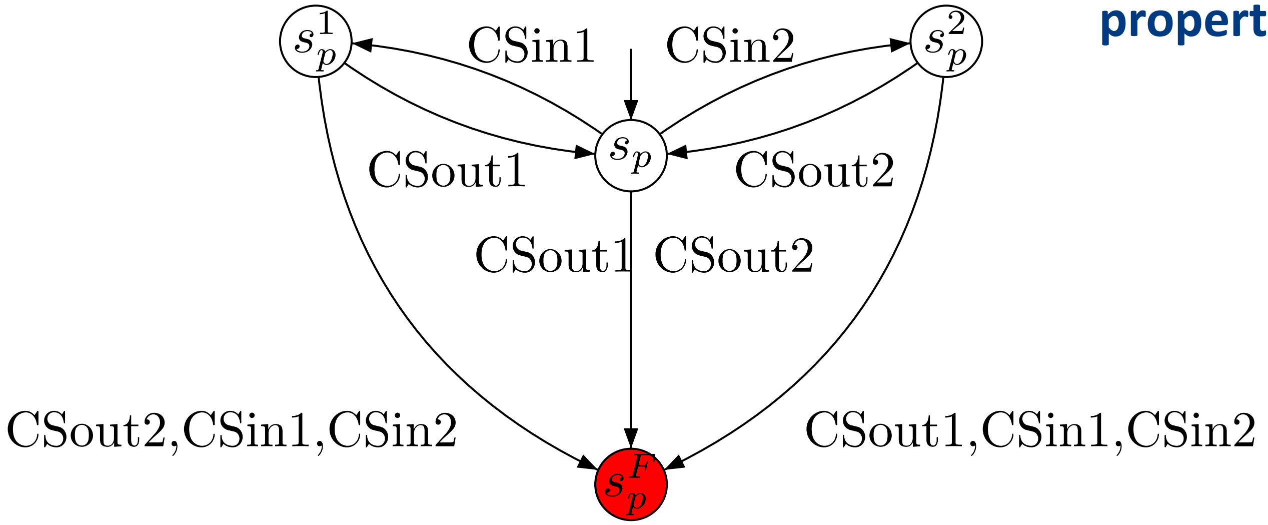
<R,3> ...

Store lists of reachable states via transitions labelled “cross” and combine

<G, 1>

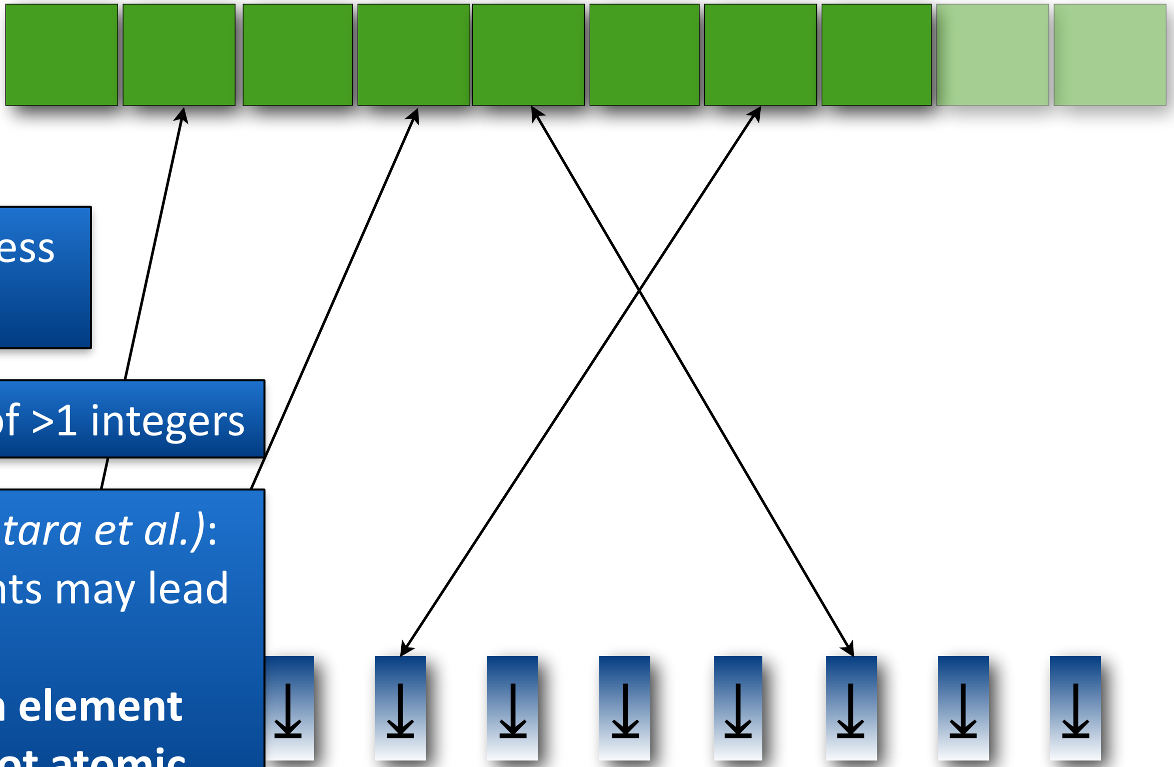
- Block fetches **unexplored vectors** from global to shared memory
- Threads are placed in *groups* of size *n* (= state vector length)
- Each thread fetches transition entries of its process / state
  - independent transitions are *immediately processed*
- *For synchronisations:* all transitions of next label are fetched, group leader manages progress

Property checking



- Add another automaton to the model network representing the property
- Example: **mutual exclusion property**

State storage



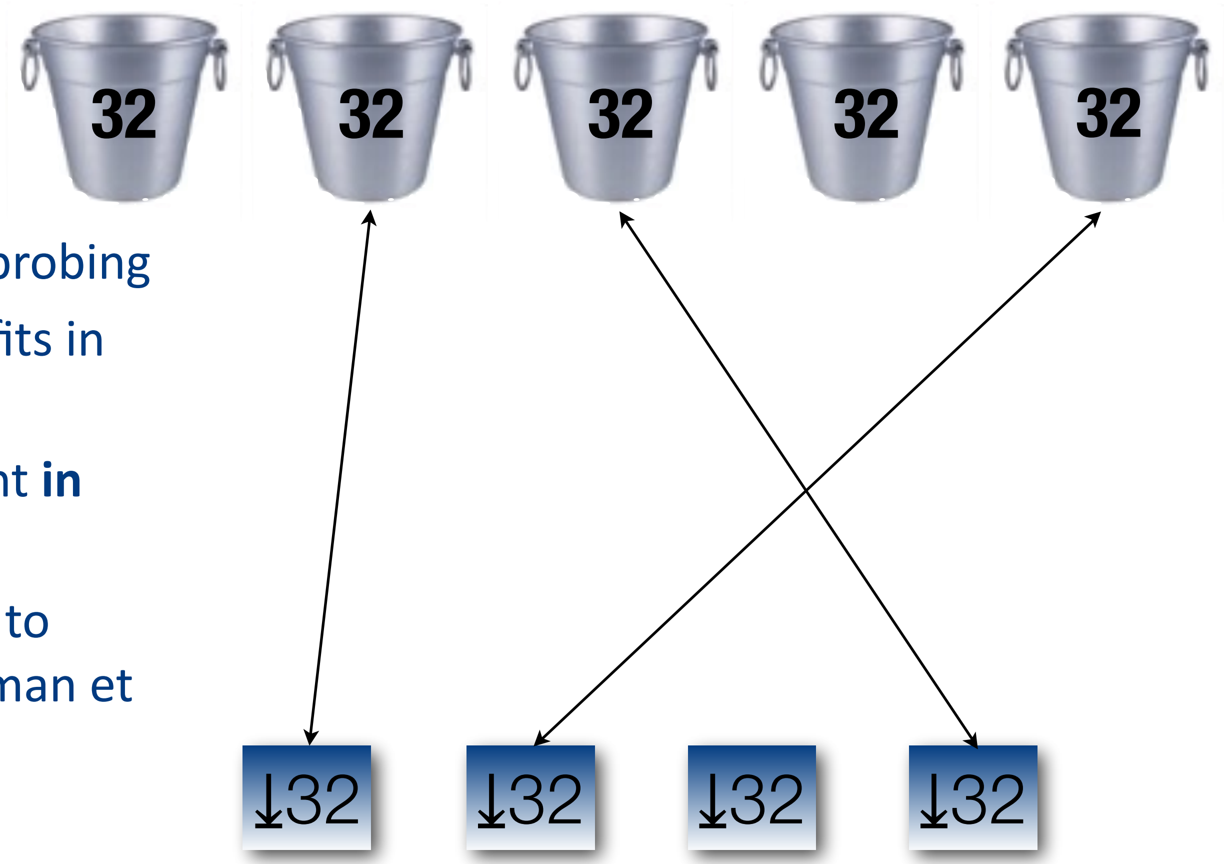
In a warp, random memory access ⇒ *bad for performance*

Worse when elements consist of >1 integers

Cuckoo hashing on GPUs *(Alcantara et al.)*:

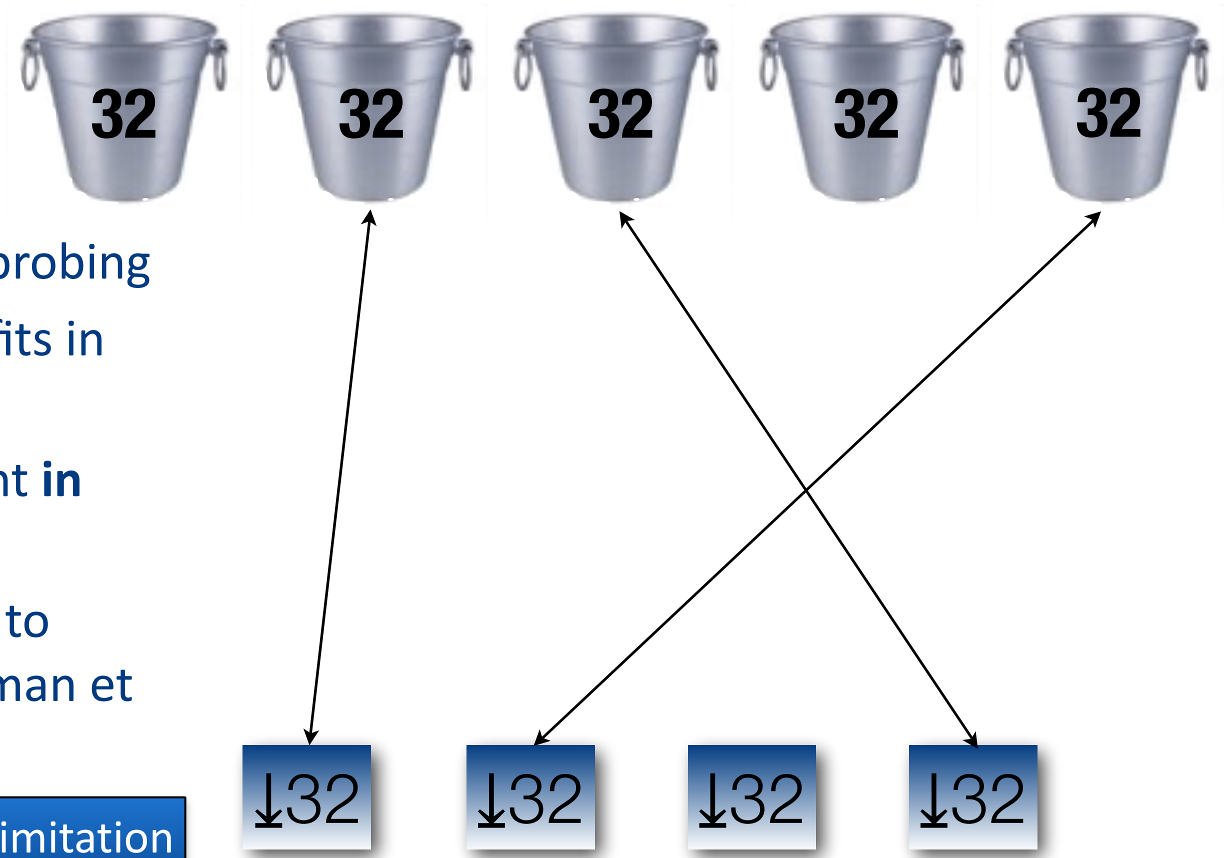
- moving around of elements may lead to duplicate entries
- **Drastically more so when element insertion and lookup is not atomic**
- Need of another hash table design

State storage



- Hash table with linear probing
- Buckets of 32 integers fits in **cache line**
- Scanning bucket content **in parallel**
    - **warp-the-line** (nod to walk-the-line [Laarman et al.,'10])

State storage



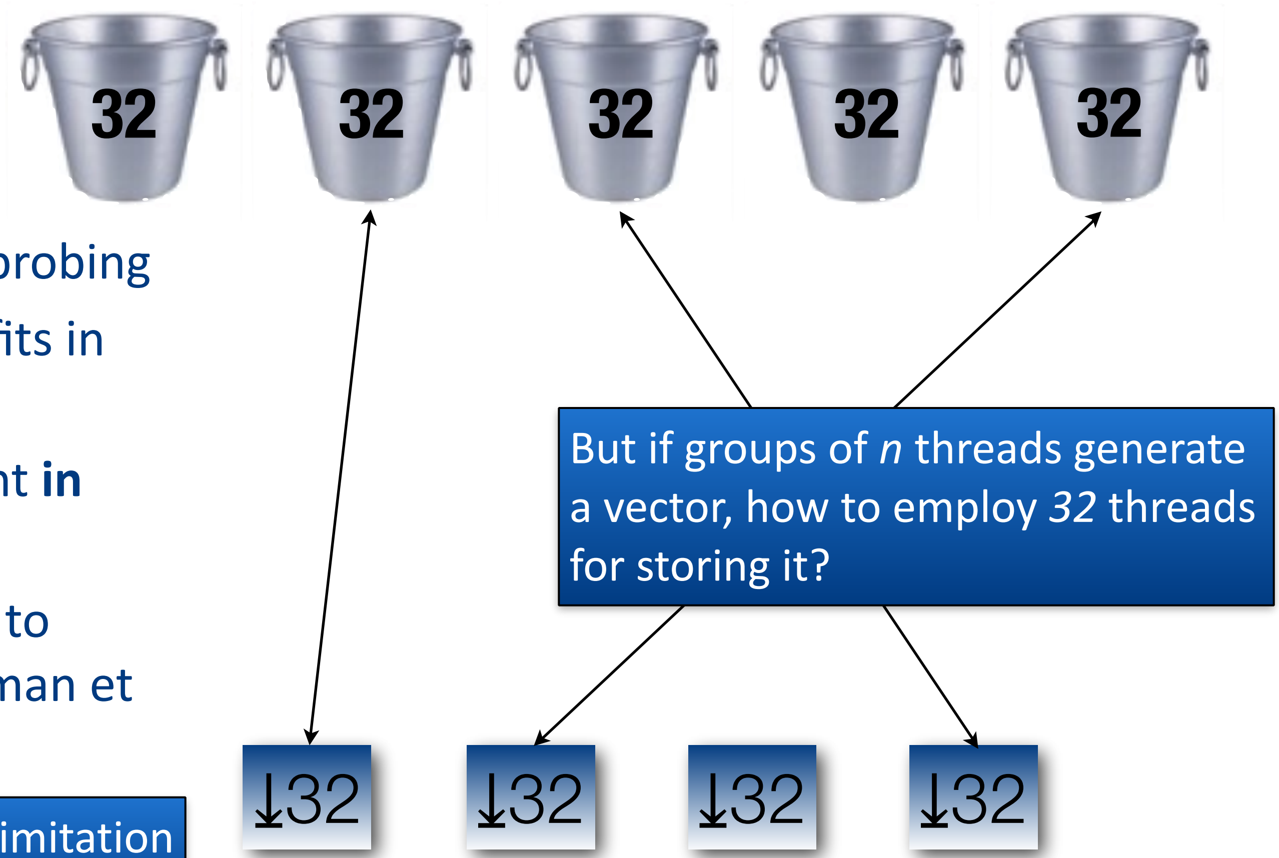
- Hash table with linear probing
- Buckets of 32 integers fits in **cache line**
- Scanning bucket content **in parallel**
    - **warp-the-line** (nod to walk-the-line [Laarman et al.,'10])

Assumes vector size < 32, limitation can be overcome

State storage

- Hash table with linear probing
- Buckets of 32 integers fits in **cache line**
- Scanning bucket content **in parallel**
  - **warp-the-line** (nod to walk-the-line [Laarman et al.,'10])

Assumes vector size < 32, limitation can be overcome

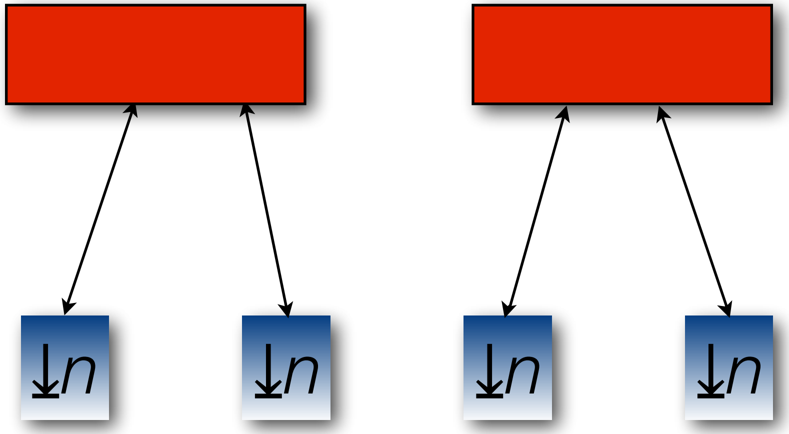


But if groups of *n* threads generate a vector, how to employ *32* threads for storing it?

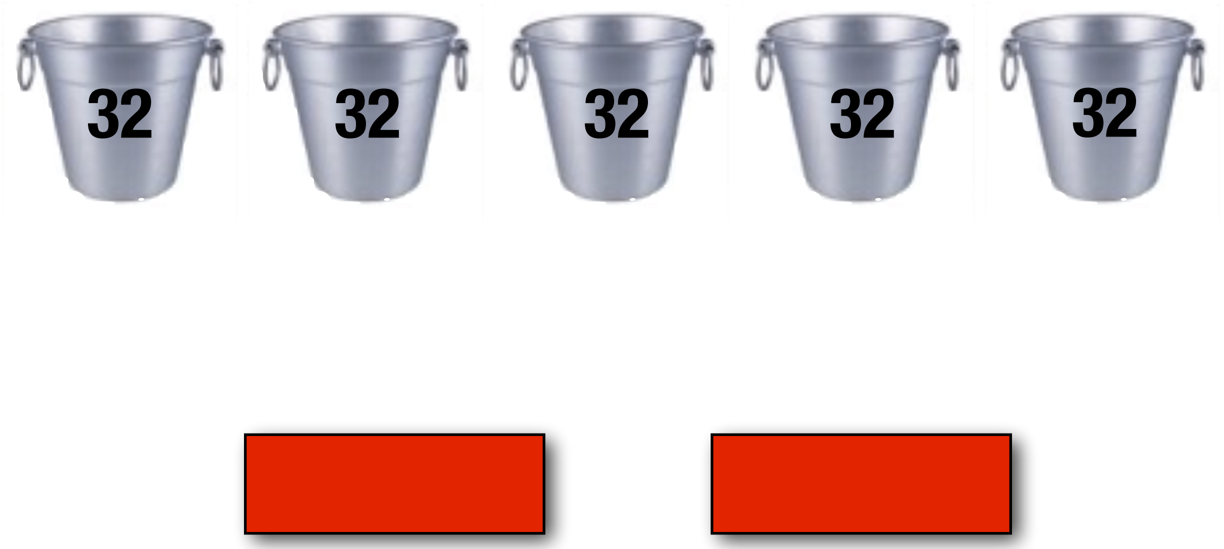
State storage

32
32
32
32
32

- Shared memory hash table used for temporary storage
  - block-local **partial duplicate detection**



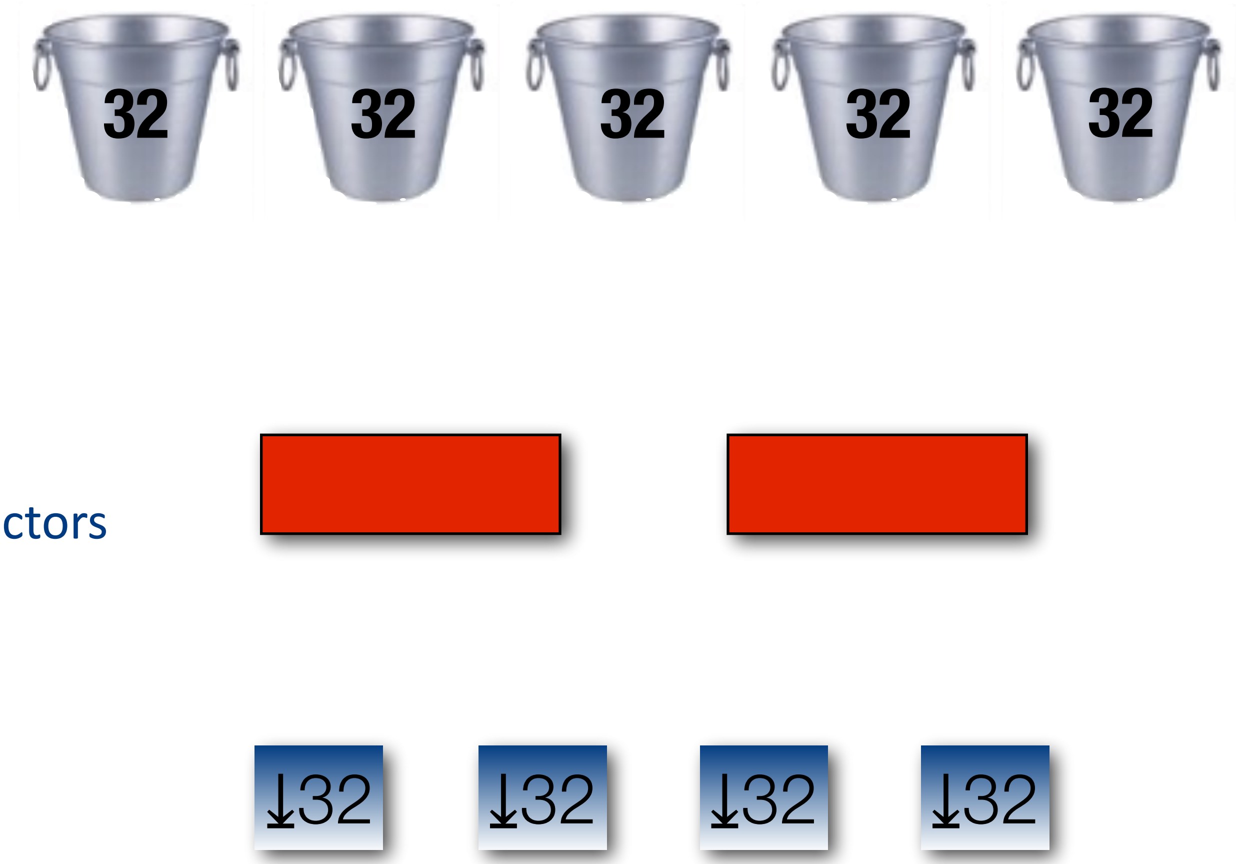
State storage



State storage

32
32
32
32
32

State storage

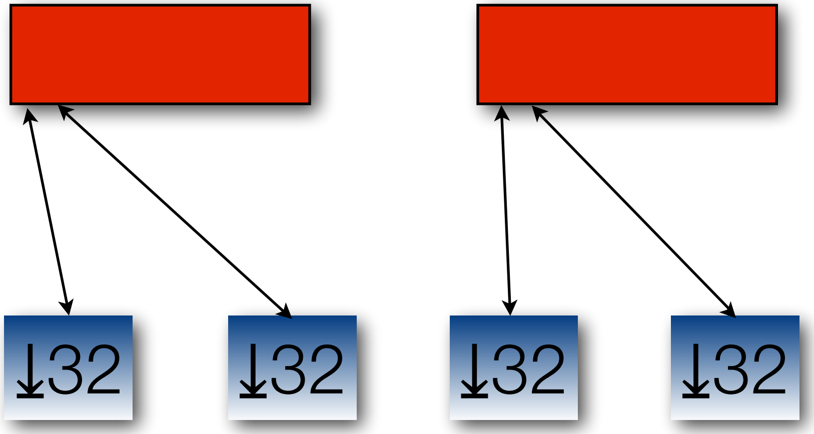


- Warp scans shared memory
- Warp stores new vectors in buckets

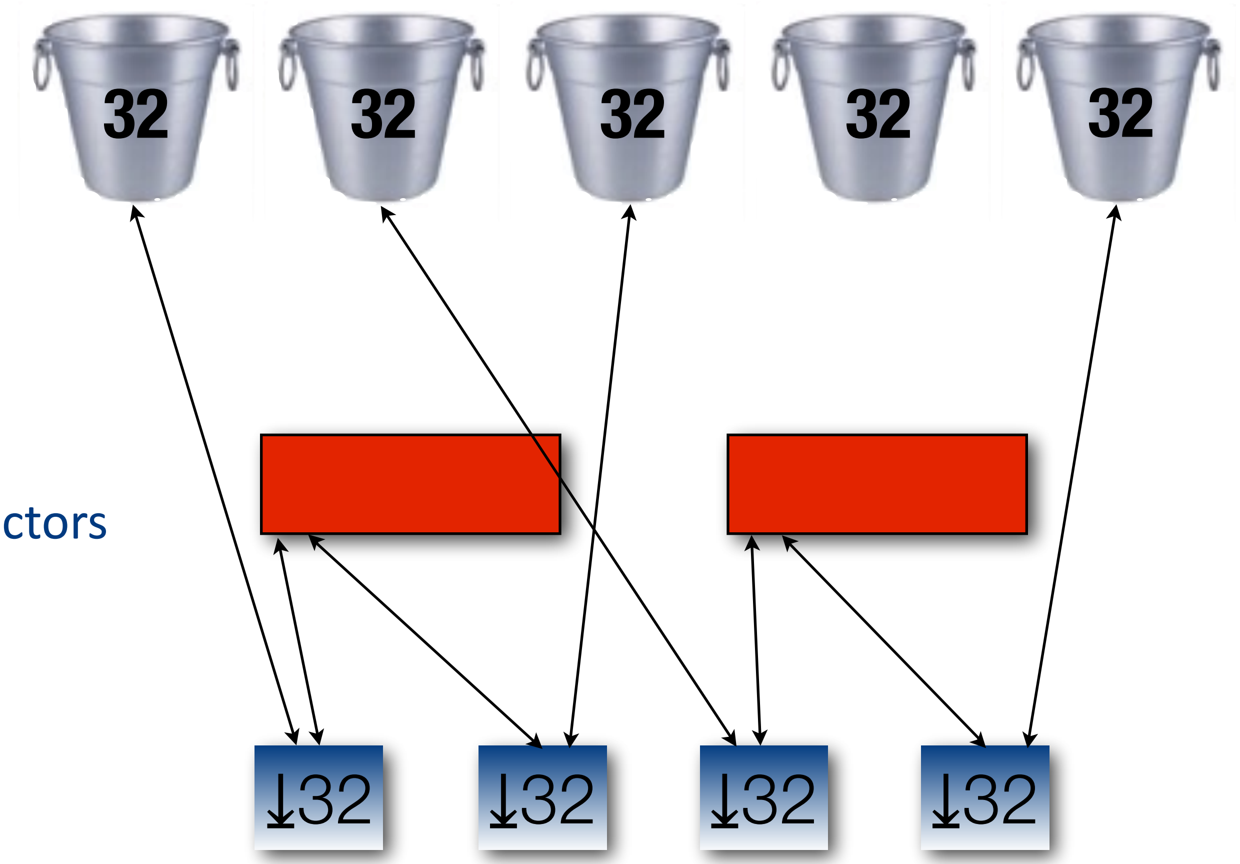
State storage

32
32
32
32
32

- Warp scans shared memory
- Warp stores new vectors in buckets



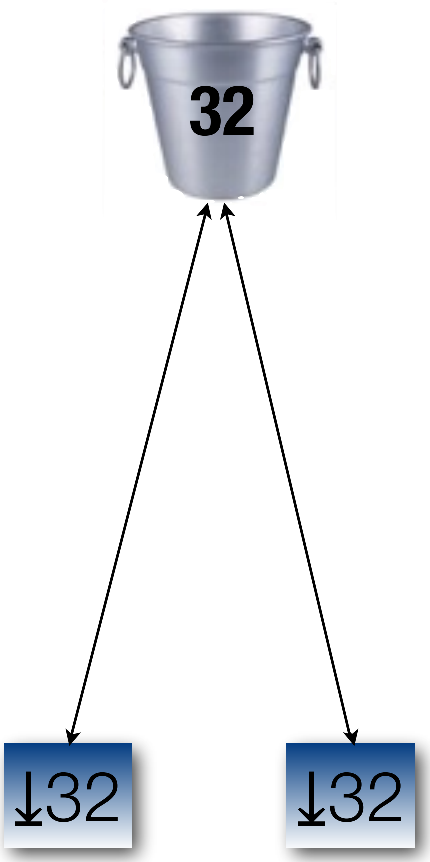
State storage



- Warp scans shared memory
- Warp stores new vectors in buckets

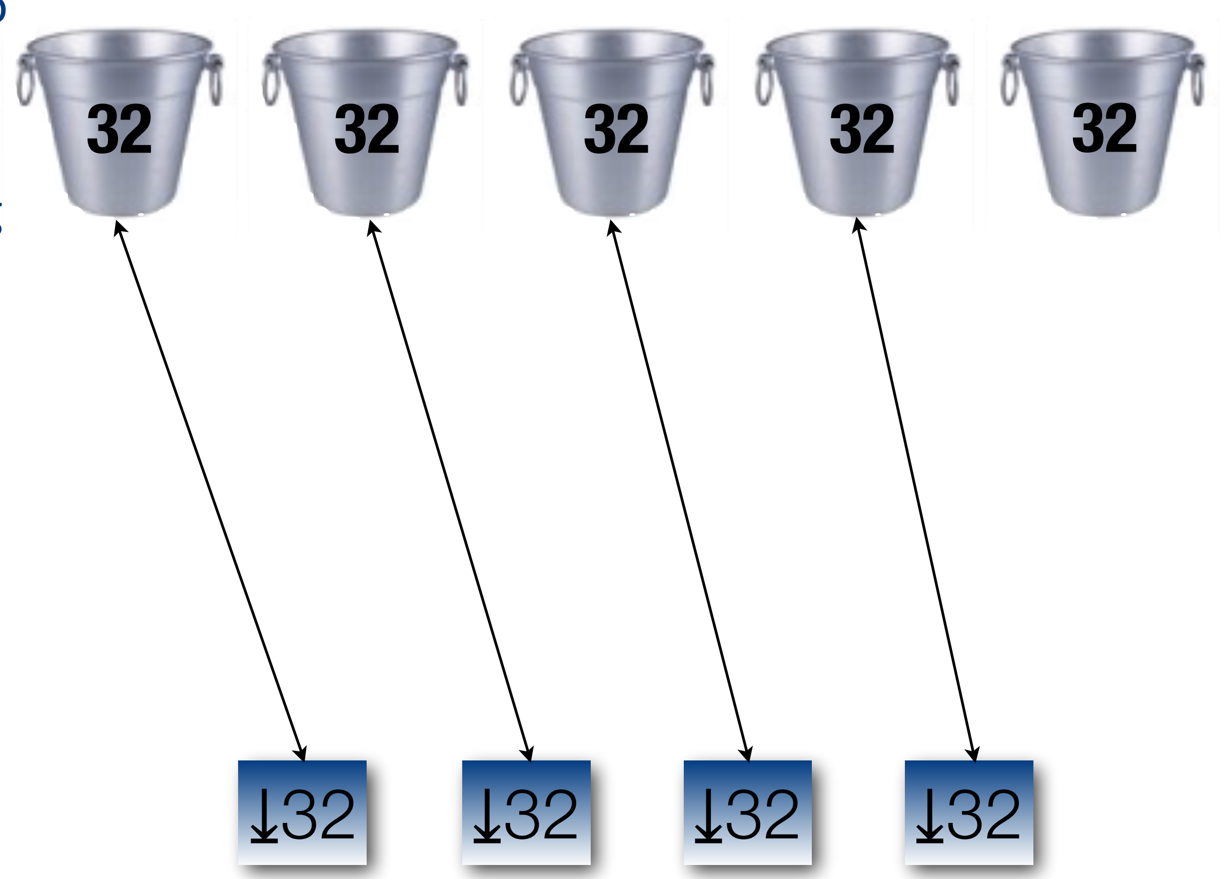
Data races

- For vectors in multiple integers
  - Warp W1 can be writing vector *v* while warp W2 reads
- False positives
  - W2 concludes that *v* is **not** in hash table
- However: results in **redundant work**, not in ignoring states
  - On average 2% redundant work



State retrieval

- Global hash table also serves for **state retrieval**
  - Requires scanning hash table for work
- **Work claiming:**
  - When a group generates new vector, it is **claimed by block** for next iteration

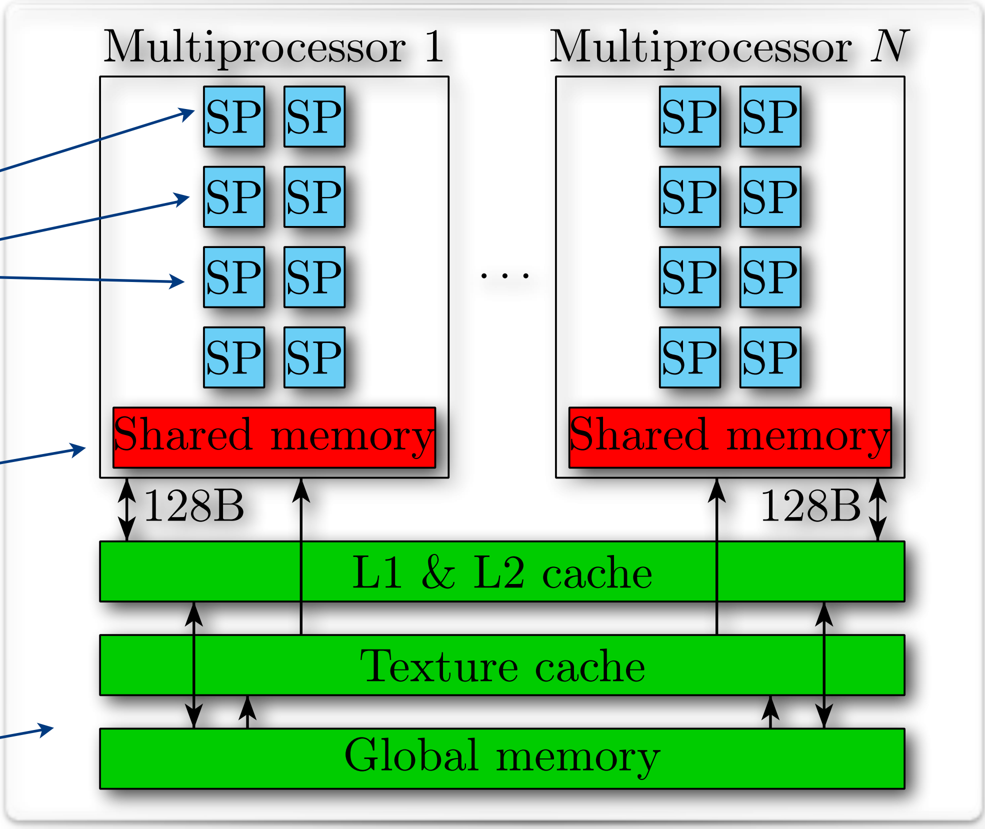


Model representation

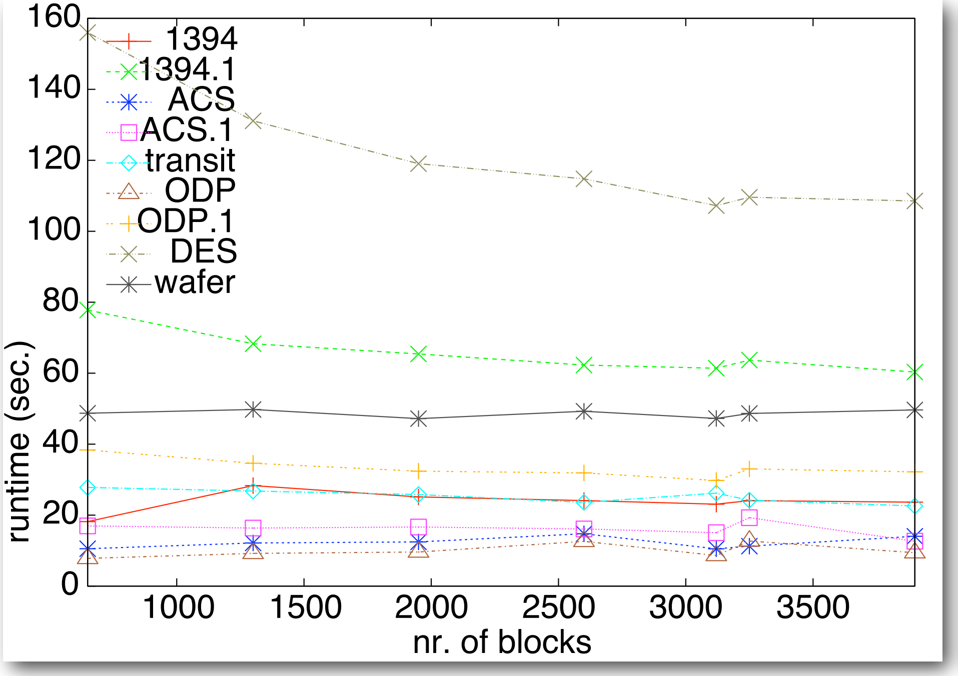
Next-state computation

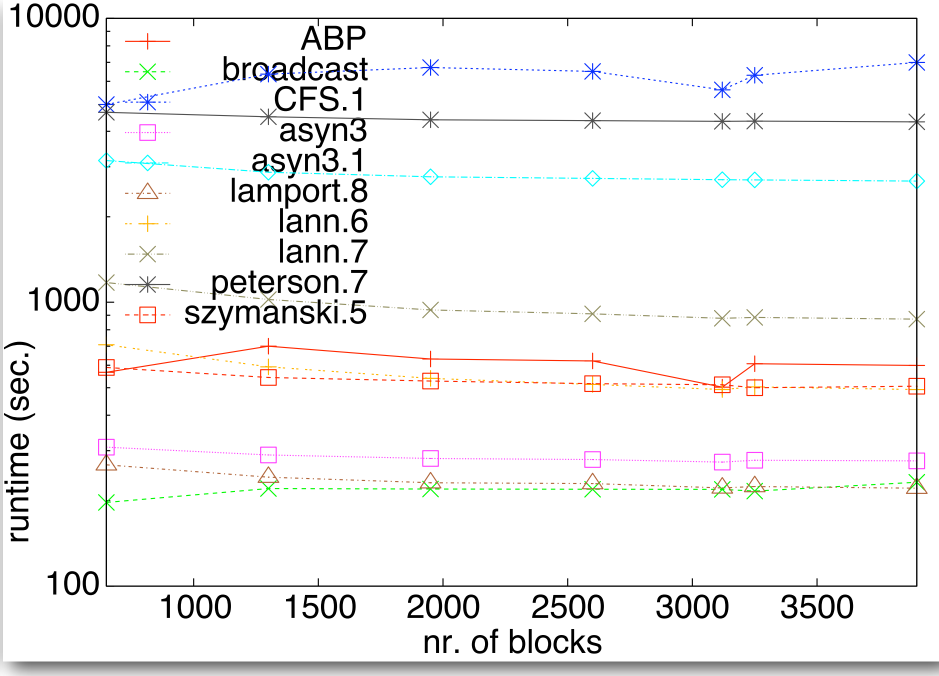
Block-local state caches

Global hash table

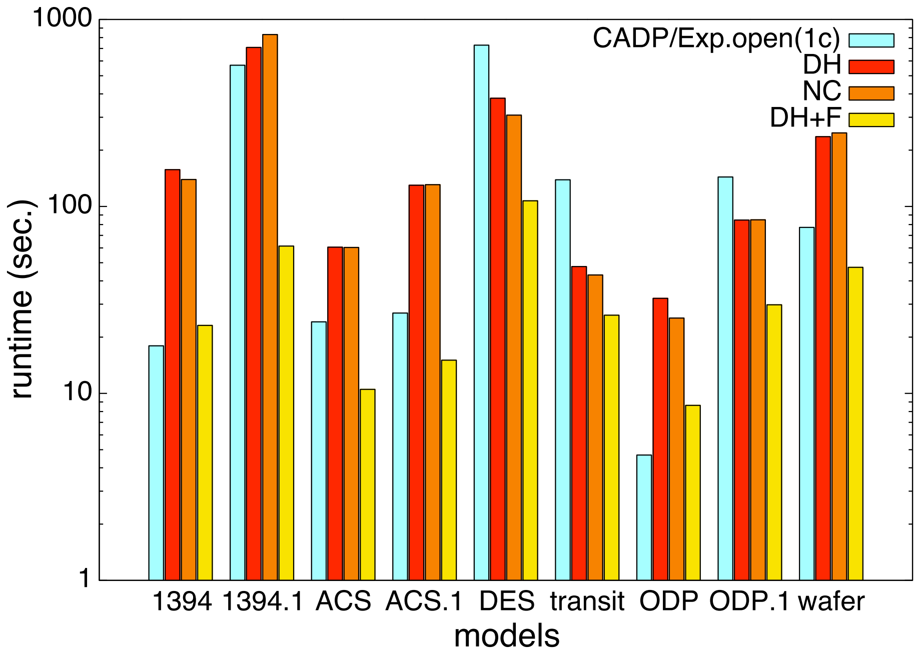


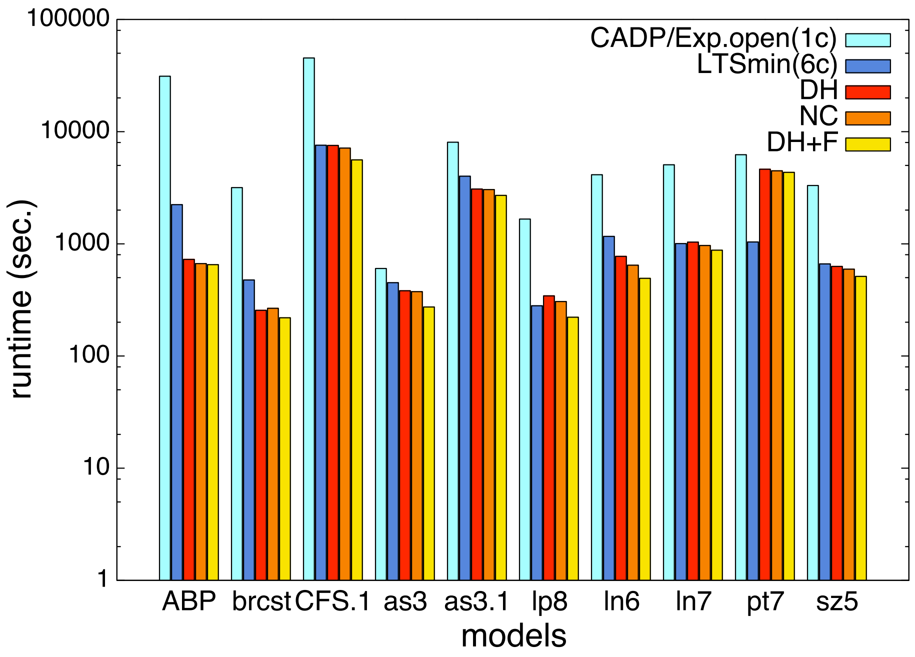
Parameter experiments - blocks



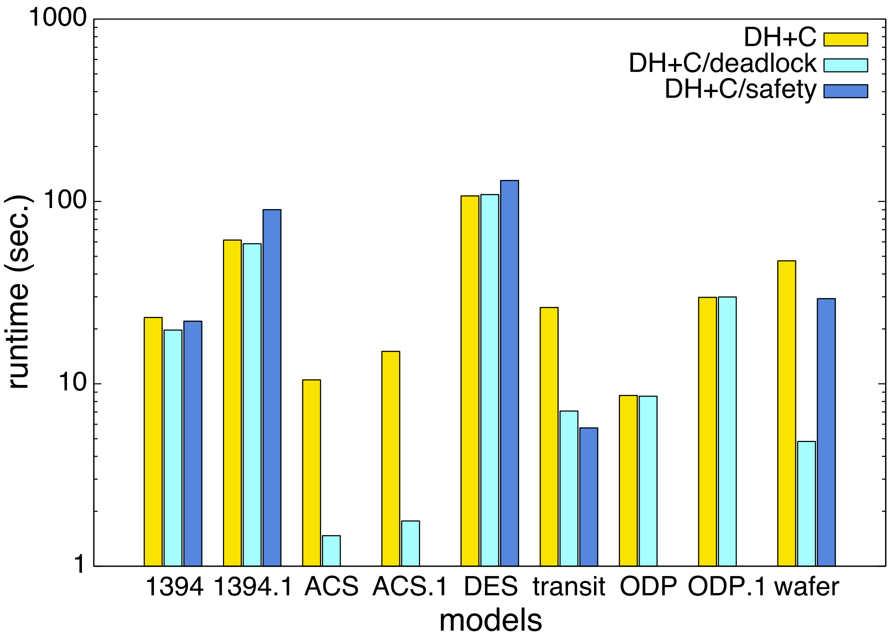


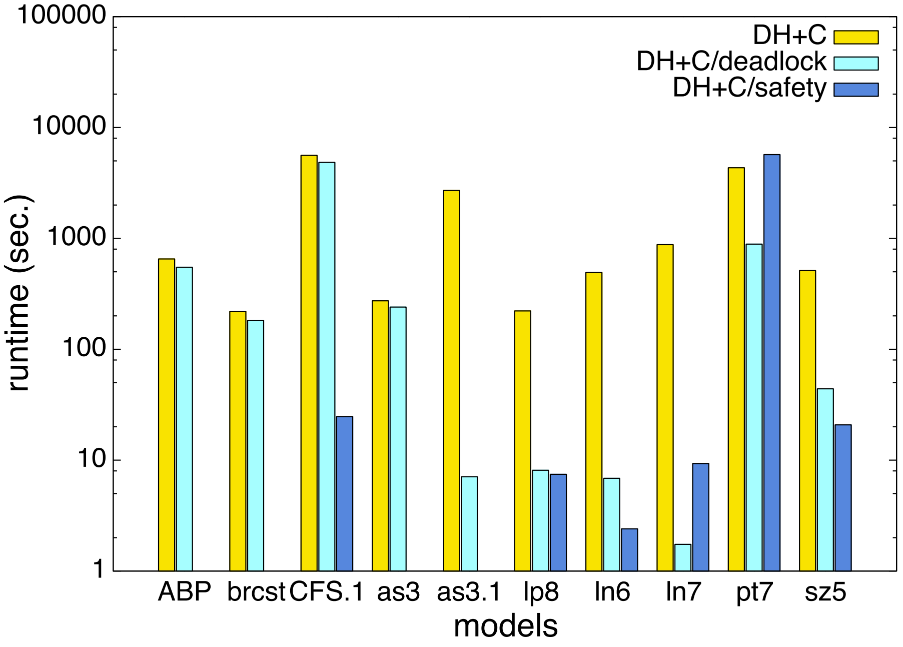
Runtimes - exploration





Runtimes - property checking





Further material

- **GPUexplore, GPUdecompose, GPUreduce** tools online
  - http://www.win.tue.nl/~awijs/software.html
- **Publications Model Checking & GPUs:**
  - ***Parallel Probabilistic Model Checking on General Purpose Graphics Processors***, D. Bošnački, S. Edelkamp, D. Sulewski and A.J. Wijs. *International Journal on Software Tools for Technology Transfer, Volume 13, Issue 1, pp. 21-35, Springer (January 2011)*
  - ***Improving GPU Sparse Matrix-Vector Multiplication for Probabilistic Model Checking***, A.J. Wijs and D. Bošnački. *In Proc. 19th International SPIN Workshop on Model Checking of Software (SPIN'12), Oxford, Great Britain, volume 7385 of Lecture Notes in Computer Science, pp. 98-116, Springer (2012)*
  - ***GPUexplore: Many-Core On-The-Fly State Space Exploration Using GPUs***, A.J. Wijs and D. Bošnački. *In Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14), Grenoble, France, volume 8413 of Lecture Notes in Computer Science, pp. 233-247, Springer (2014)*
  - ***GPU-Based Graph Decomposition into Strongly Connected and Maximal End Components***, A.J. Wijs, J.-P. Katoen and D. Bošnački. *In Proc. 26th International Conference on Computer Aided Verification (CAV'14), Vienna, Austria, volume 8559 of Lecture Notes in Computer Science, pp. 309-325, Springer (2014)*
  - ***GPU Accelerated Strong and Branching Bisimilarity Checking***, A.J. Wijs. *In Proc. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), London, UK, to appear*
- **Poster P5185 - Harnessing the Power of GPUs for Model Checking**

Structure of the talk

- Automatic formal verification: what is it and why use it?
    - ***State space generation and analysis***
- GEM Toolbox: Model Checking on GPUs
    - What does it offer?
    - How is it implemented?
        - Range of techniques specifically designed for state space structures
    - What speedups can it achieve?

Dining Philosophers Problem

- 5 Philosophers at a dining table
- A philosopher needs two forks to eat (on the right and left)
- *Can a philosopher starve?*
- *Can all philosophers starve?*

- Try out possibilities or …

- Make a formal specification of the situation (what is there and what can happen?)
- Automatically check all possible events and states of the system
- *Model checking*
- Allows you to check all kinds of properties

- *State space*: involves all possible *states* of system, and *transitions* between those states
- Image of the state space of a **Bounded Retransmission Protocol** model
- Model checking can guarantee that a system is correct or can reach undesired states (*the dining philosophers can starve*)
- But…
- Model checking is computationally very demanding, due to *state space explosion problem*
  - Linear growth of model tends to lead to exponential growth of state space