Formal Verification and Synthesis for Quality-of-Service in On-Chip Networks

August 27, 2013

Daniel E. Holcomb
UC Berkeley
Prof. Sanjit A. Seshia

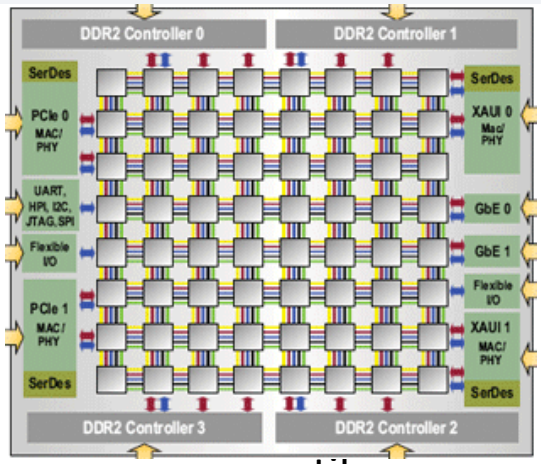
Acknowledgments

- Bryan Brady
- Michael Kishinevsky
- Alexander Gotmanov
- Yuriy Viktorov
- Satrajit Chatterjee
- Bob Brayton's group
- Jiang and Alan
- etc

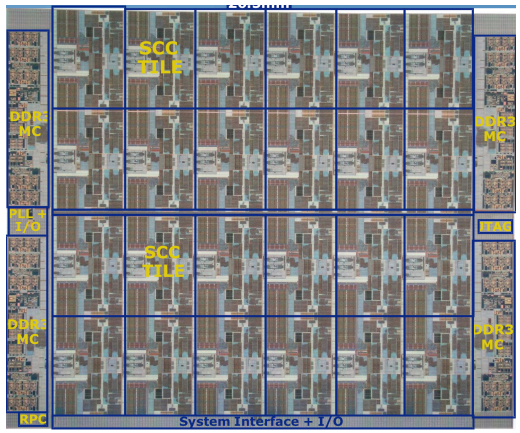
Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

Motivation

- On-chip networks due to scaling
  - Intel Teraflops Research chip
    - 80 simple cores
  - Tilera TILE-Gx100
    - 100 general purpose cores
  - Intel Single-Chip cloud computer
    - 48 IA cores
- Significant Cost/Performance considerations
- Complex to reason about
  - Arbitration -- including unfair policies
  - Reservations
- State of the art for Quality of Service (QoS)
  - Reasoning through extensive simulation
  - Analytical techniques in some simple cases
  - Can we do better with formal methods?



www.tilera.com



www.intel.com

Thesis Statement

Leverage model checking for solving NoC QoS latency problems. Address capacity limitations by extending well-known formal techniques including abstraction and compositional reasoning into the NoC domain

- This talk concerns 3 specific QoS contributions
  - Workload abstraction of traffic models
  - Latency proofs by property strengthening
  - Optimal buffer sizing for QoS

Compositional Latency Verification

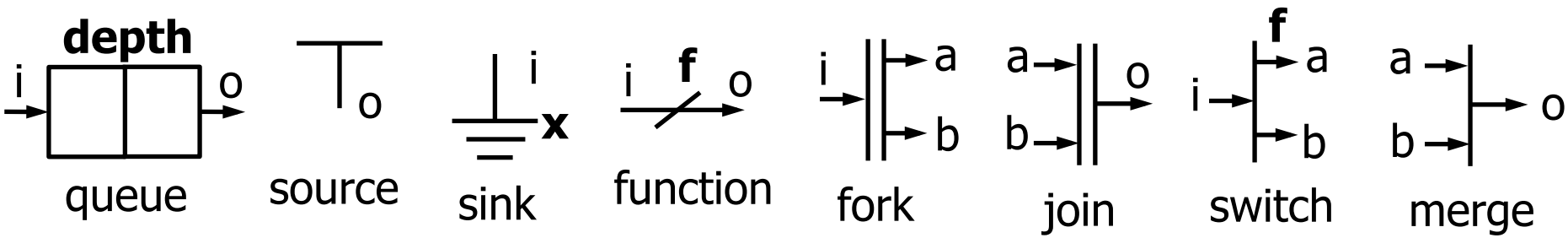
Outline

- **Background**
  - **xMAS: Formal model of NoC**
  - **Verification technology**
- Compositional latency verification
  - Abstraction and traffic modeling
  - Inductive proof by property strengthening
- Buffer sizing using counterexamples

xMAS Modeling Language

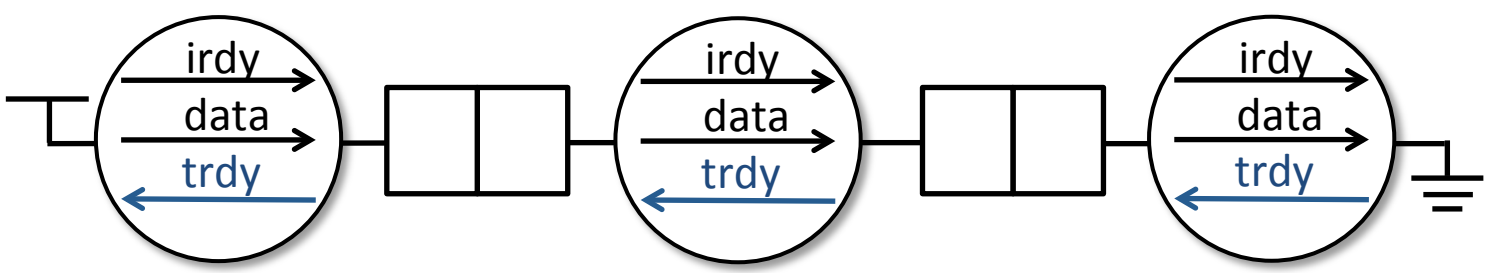
Executable Microarchitectural Specifications (xMAS)

- Represent communication fabrics as compositions of kernel primitives [Chatterjee et al., HLDVT'10]



- Primitives connected by channels

transfer = irdy & trdy



Why use xMAS Formalism?

- Effective abstraction for real designs
  - Expressive enough to model interesting behaviors
  - Hides messy details of arbitrary RTL
- Finite set of primitives allows reuse of reasoning
- Convenient entry point for verification tools
  - RTL or UCLID modeling language
  - Can augment xMAS with arbitrary sequential logic as needed

Formal Verification / Model Checking

- Model checking
  - Given a finite state model with initial state(s), inputs and transition relation
  - Formally check whether any sequence of inputs can drive the model to a bad state
- Used when bugs are unacceptable
  - Safety-critical applications
  - VLSI designs
- Adoption limited by lack of appropriate models, and inability to scale

Model Checking an xMAS network

- SAT-based verification using And-Inverter-Graphs (AIGs)
- State-of-the-art model checking engines in ABC [Brayton and Mishchenko, CAV'10]
  - Bounded Model Checking (BMC)
  - Induction / K-induction
  - Property directed reachability (PDR) [Een and Mishchenko, IWLS'11]
    - ABC's Implementation of IC3 [Bradley VMCAI'11]
- VeriABC converts verilog to AIG [Long et al.,IWLS'11]

Model Checking an xMAS network

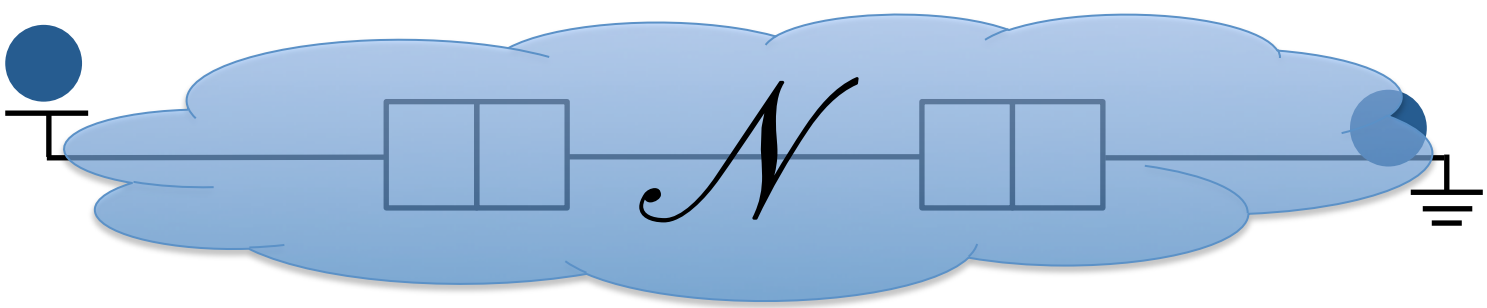
- Satisfiability Modulo Theories (SMT) solving
  - An approach for coping with state explosion
  - Decide validity of term-level formulas
- UCLID model checker [Bryant et al., CAV'02]
  - Verifier for systems expressed in a combination of logical theories
  - Symbolic simulation
  - SMT solving as the underlying engine

Verifying Latency Bounds

- xMAS network $\mathcal{N}$
- Denote latency property Φ
- Property for global latency bound T

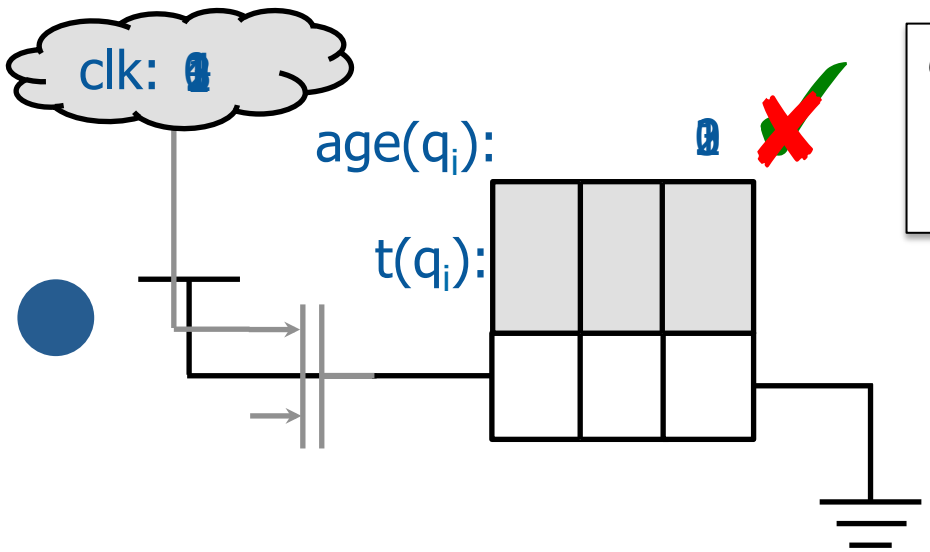
$$\Phi_T^G :\approx \mathbf{G}\left(src \implies \mathbf{F}^{<T}\ sink\right)$$

  – A "prompt-LTL" property [Kupferman et al. '09]
- Goal is to verify $\mathcal{N} \models \Phi_T^G$



Checking Latency as Simple Safety Property

- Represent latency since injection as **age** of packet
  - Global clock (n-bit counter)
  - Injection timestamps on packets (widen queue slots by n-bits)



$$age(q_i) := \quad clk - t(q_i) \mod t_{max}$$

Outline

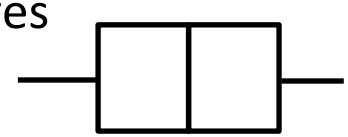
- Background
  - xMAS: Formal model of NoC
  - Verification technology
- **<u>Compositional latency verification</u>**
  - Abstraction and traffic modeling
  - Inductive proof by property strengthening
- Buffer sizing using counterexamples

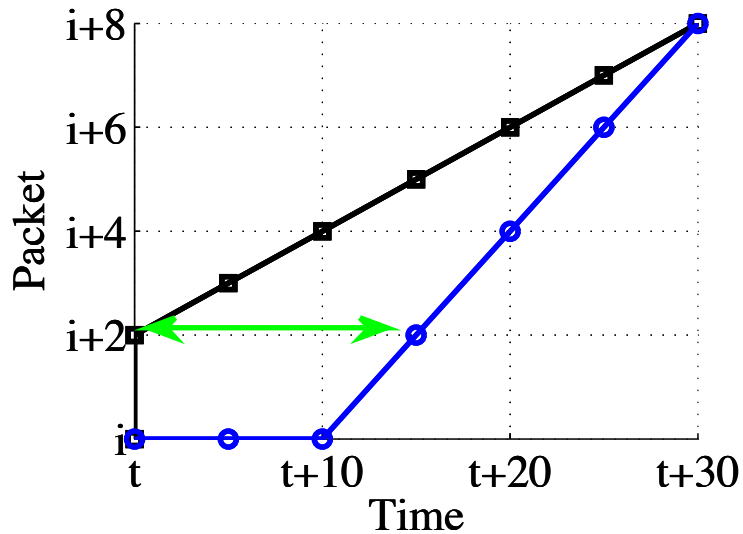
Compositional Latency Verification

- Size of interesting latency problems
  - 10s or 100s of cycles
  - Similar number of queues
- Without compositional reasoning, verification difficulty grows with both
  - Beyond scope of current tools
- We give two approaches to compositional latency verification of NoC

Related Work

- **Network Calculus** [Cruz '91] for analyzing networks with traffic rates bounded by linear envelopes
  - **Arrival** and **service** curves
  - **Latency** bounds
  - Algebraic composition rules





- Model checking to verify of Router Latency [Krishna et al. Haifa '11]

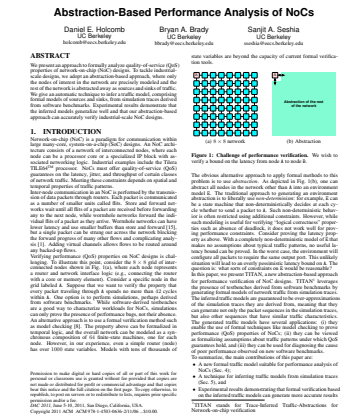
- **Static Timing Analysis** of combinational circuits
  - Use graph as abstraction of gate-level timing

Outline

- Background
  - xMAS: Formal model of NoC
  - Verification technology
- Compositional latency verification
  - **Abstraction and traffic modeling**
  - Inductive proof by property strengthening
- Buffer sizing using counterexamples

Traffic Abstraction

- Decompose both model and property
- Reduce scope of verification problem by focusing on one router in isolation
- Abstract rest of network into precise environment model



Abstraction-Based Performance Analysis of NoCs

Daniel E. Holcomb, UC Berkeley, holcomb@eecs.berkeley.edu
Bryan A. Brady, UC Berkeley, bbrady@eecs.berkeley.edu
Sanjit A. Seshia, UC Berkeley, sseshia@eecs.berkeley.edu

ABSTRACT

We present an approach to formally analyze quality-of-service (QoS) properties of network-on-chip (NoC) designs. To tackle industrial-scale NoCs, we use a decomposition of both model and property. Only the nodes of interest in the network are precisely modeled and the rest of the network is abstracted away as sources and sinks of traffic. We give an automatic technique to infer a traffic model, comprising formal models of sources and sinks, from simulation traces derived from software benchmarks. Experimental results demonstrate that the inferred models generalize well and that our abstraction-based approach can accurately verify industrial-scale NoC designs.

1. INTRODUCTION

Network-on-chip (NoC) is a paradigm for communication within VLSI chip multiprocessor (CMP) designs. An NoC architecture consists of a network of interconnected nodes, where each node can be a processor core or a specialized IP block with associated networking logic. Industrial examples include the Tilera TILE64™ processor. NoCs must offer quality-of-service (QoS) guarantees to ensure that the system meets performance requirements of network traffic. Meeting these constraints depends on spatial and temporal properties of traffic patterns.

Inter-node communication in an NoC is performed by the transmission of data packets through routers. Each packet is communicated as a number of smaller units called flits. Store and forward networks wait until all flits of a packet are received before forwarding the packet to the next node, while wormhole routers forward individual flits of a packet as they arrive. Wormhole networks can have lower latency and use smaller buffers than store and forward [15], but a single packet can be strung out across the network blocking the forward progress of many other flows and complicating analysis. Use of virtual channels allows flows to be routed around any backed-up flows.

Verifying performance (QoS) properties on NoC designs is challenging. To illustrate this point, consider the 8×8 grid of interconnected nodes shown in Fig. 1(a), where each node represents a router and network interface logic (e.g., connecting the router with a core or memory element). Consider a specific node in the center of the network. Suppose we want to prove that the latency of every packet traveling through it spends no more than 10 cycles within it. One option is to perform simulations, perhaps drawn from software benchmarks. While software-derived testbenches are a good way to characterize workloads for NoCs, simulations do not provide exhaustive coverage of possible traffic patterns. An alternative approach is to use a formal verification method such as model checking [8]. The property above can be formalized in temporal logic, and the overall network can be modeled as a synchronous composition of 64 finite-state machines, one for each node. However, in our experience, even a simple router (node) has over 1000 state variables. Models with tens of thousands of state variables are beyond the capacity of current formal verification tools.

(a) 8×8 network (b) Abstraction

Figure 1: Challenge of performance verification. We wish to verify a bound on the latency from node i to node k.

The obvious alternative approach to apply formal methods to this problem is to use abstraction. As depicted in Fig. 1(b), one can abstract the rest of the network into a simple environment model E. The traditional approach to generating an environment abstraction is to liberally add non-deterministic behaviors. For example, E can be a state machine that non-deterministically decides at each cycle whether to send a packet to k. Such non-deterministic behaviors can be used to prove safety properties of interest. Although such modeling is useful for verifying "logical correctness" properties such as absence of deadlock, it does not work well for proving performance constraints. Consider proving the latency property as above. With a completely non-deterministic model of E that makes no assumptions about typical traffic patterns, no useful latency bound can be proved. In the worst case, the environment will always send packets to node k and block its outgoing links, which in this situation will lead to an overly pessimistic latency bound on k. The question is: what sorts of constraints on E would be reasonable?

In this paper, we pursue TITAN, a new abstraction-based approach for performance verification of NoC designs. TITAN[1] leverages recent work on mining specifications from simulation traces and generating formal models of network traffic from simulation traces. The inferred traffic models are guaranteed to be over-approximations of the simulation traces they are derived from, meaning that they can generate not only the packet sequences in the simulation traces, but also other sequences that have similar traffic characteristics. The generated traffic models have several applications: (i) they can be used to verify performance properties of NoCs, (ii) they can be used as formalizing assumptions about traffic patterns under which QoS guarantees hold, and (iii) they can be used for diagnosing the cause of poor performance observed on new software benchmarks.

To summarize, the main contributions of this paper are:

- A new formal traffic model suitable for performance analysis of NoC designs;
- A technique for inferring traffic models from simulation traces (Sec. 5); and
- Experimental results demonstrating that formal verification based on the inferred traffic models can generate more accurate results

[1] TITAN stands for Trace-Inferred Traffic-Abstractions for Network-on-chip verification

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2011 June 5-10, 2011, San Diego, California, USA.
Copyright 2011 ACM 978-1-4503-0636-2/11/06 ... $10.00.

Daniel Holcomb, Bryan Brady, and Sanjit A. Seshia. **Abstraction-Based Performance Analysis of NoCs**. In *Proceedings of the Design Automation Conference (DAC)*, pp. 492–497, June 2011.

http://www.eecs.berkeley.edu/~holcomb/dac11-noc.pdf

Abstraction Enables Formal Approach

- Goal: prove a global latency bound

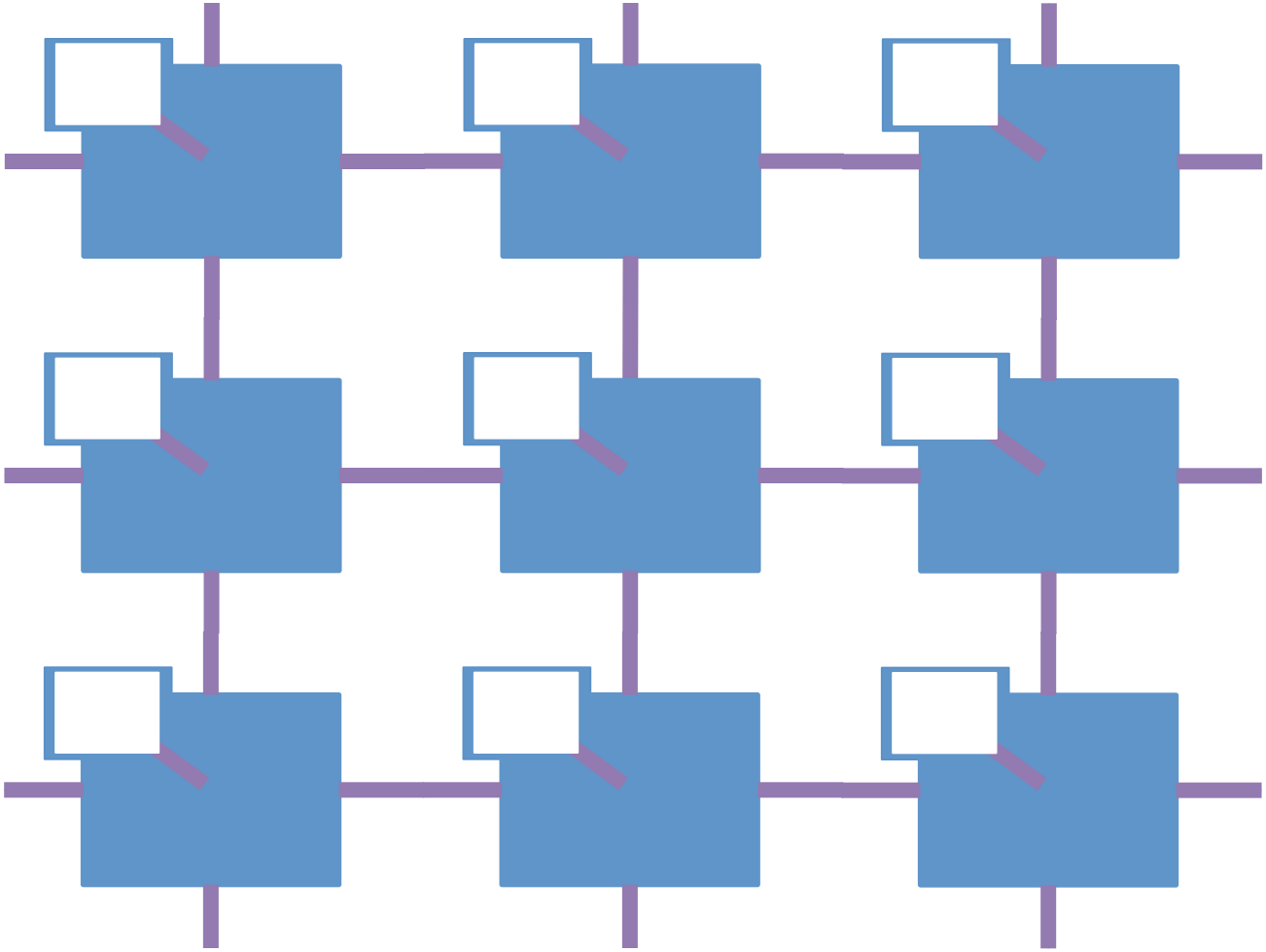
- ~~Model in xMAS~~
- ~~Find bounds by sweeping latency T and checking each~~
- Model too large!

- Prove conservative latency bounds for each router
- Compose using path routing constraints

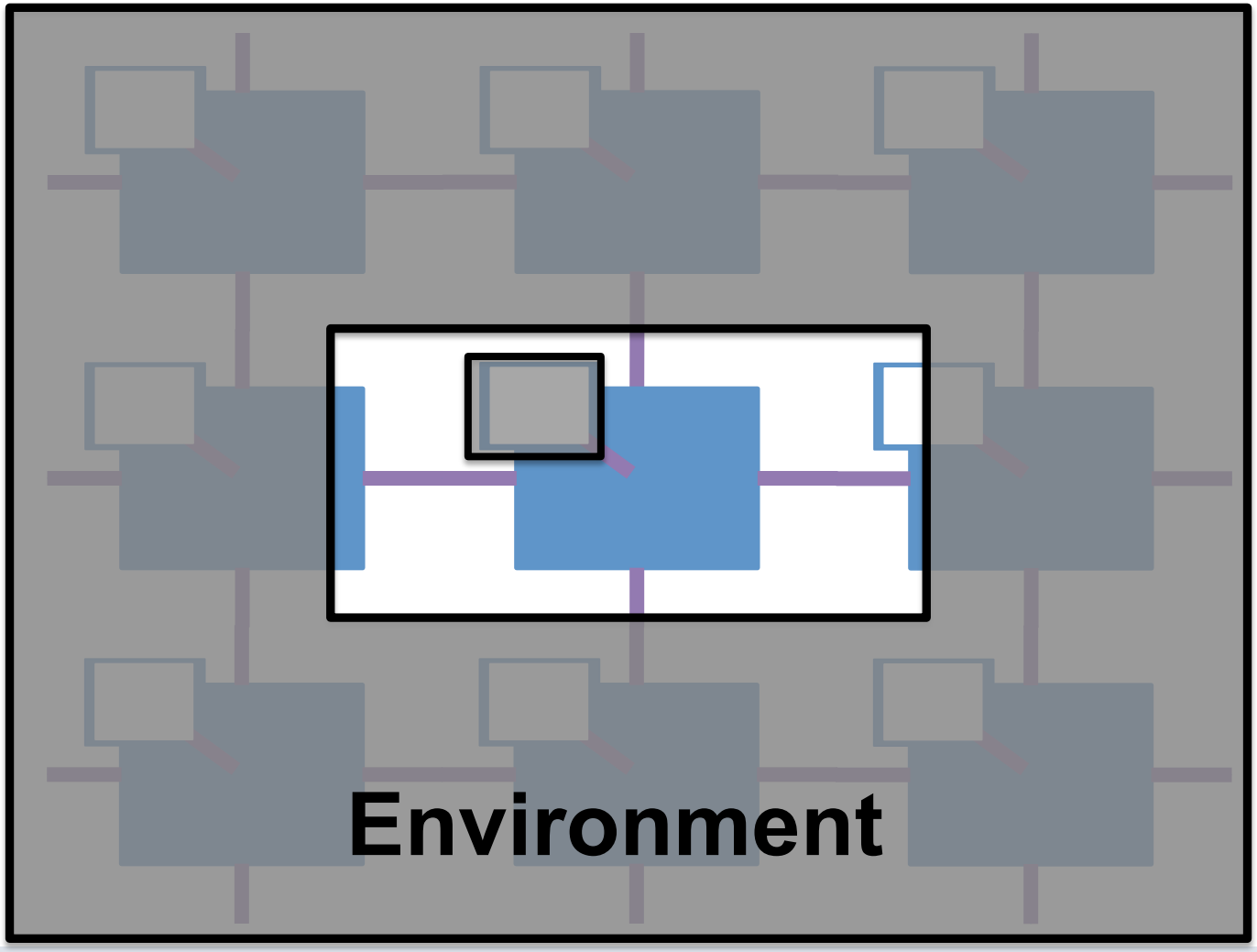
$$T := \sum_{i \in path} t_i$$

$$\left(\bigwedge_i \mathcal{N}_i \models \Phi_{t_i} \right) \implies \mathcal{N} \models \Phi_T^G$$

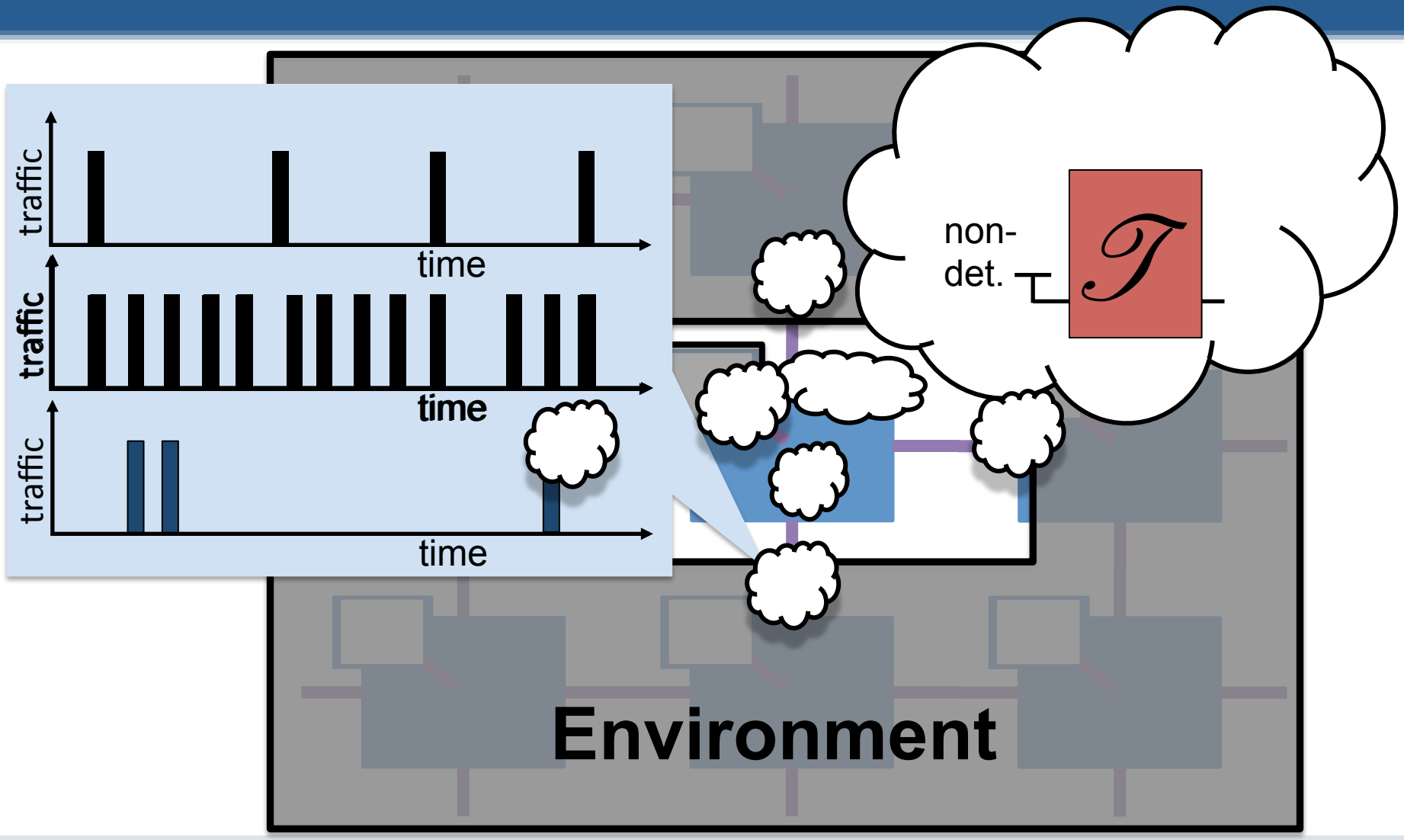
Abstraction Enables Formal Approach



Abstraction Enables Formal Approach

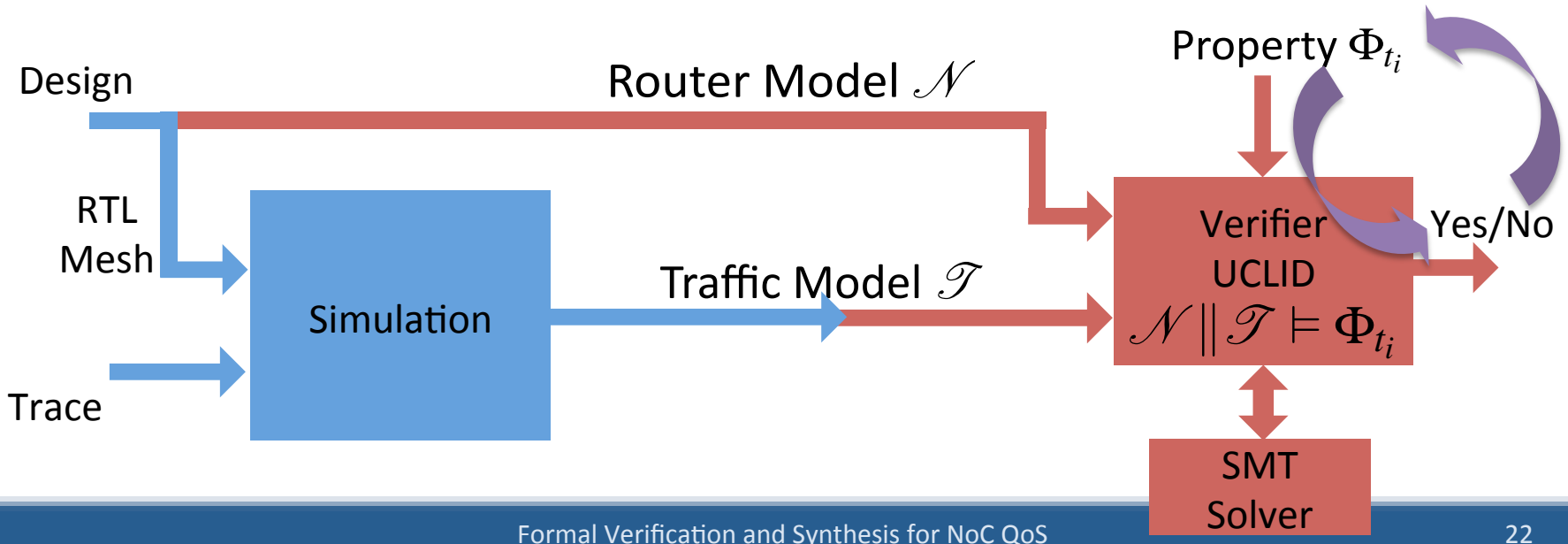


Abstraction Enables Formal Approach



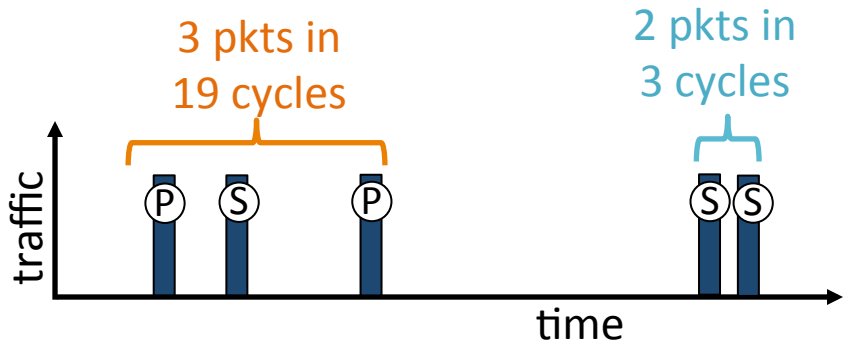
Contributions

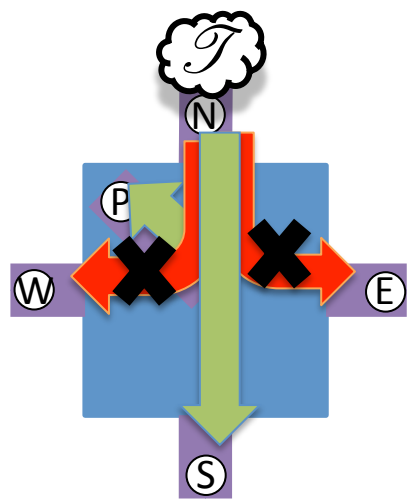
- Precise abstraction for formal NoC analysis
  - Extension of network calculus [Cruz, Tran. Info Theory '91]
- Inferring traffic model from simulation data
- Verifying performance using inferred traffic models

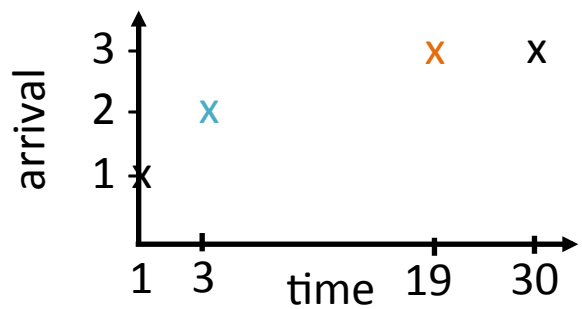


Inferring Formal Traffic Model

- Inferred from RTL simulation
  - CMP router
    [Peh PhD Thesis 2002]
  - Random samples from PARSEC traces
    [Soteriou et al., MASCOTS 2006]
- Predicates on channel behaviors
  - Boolean constraints
  - Conjunctions of rate constraints

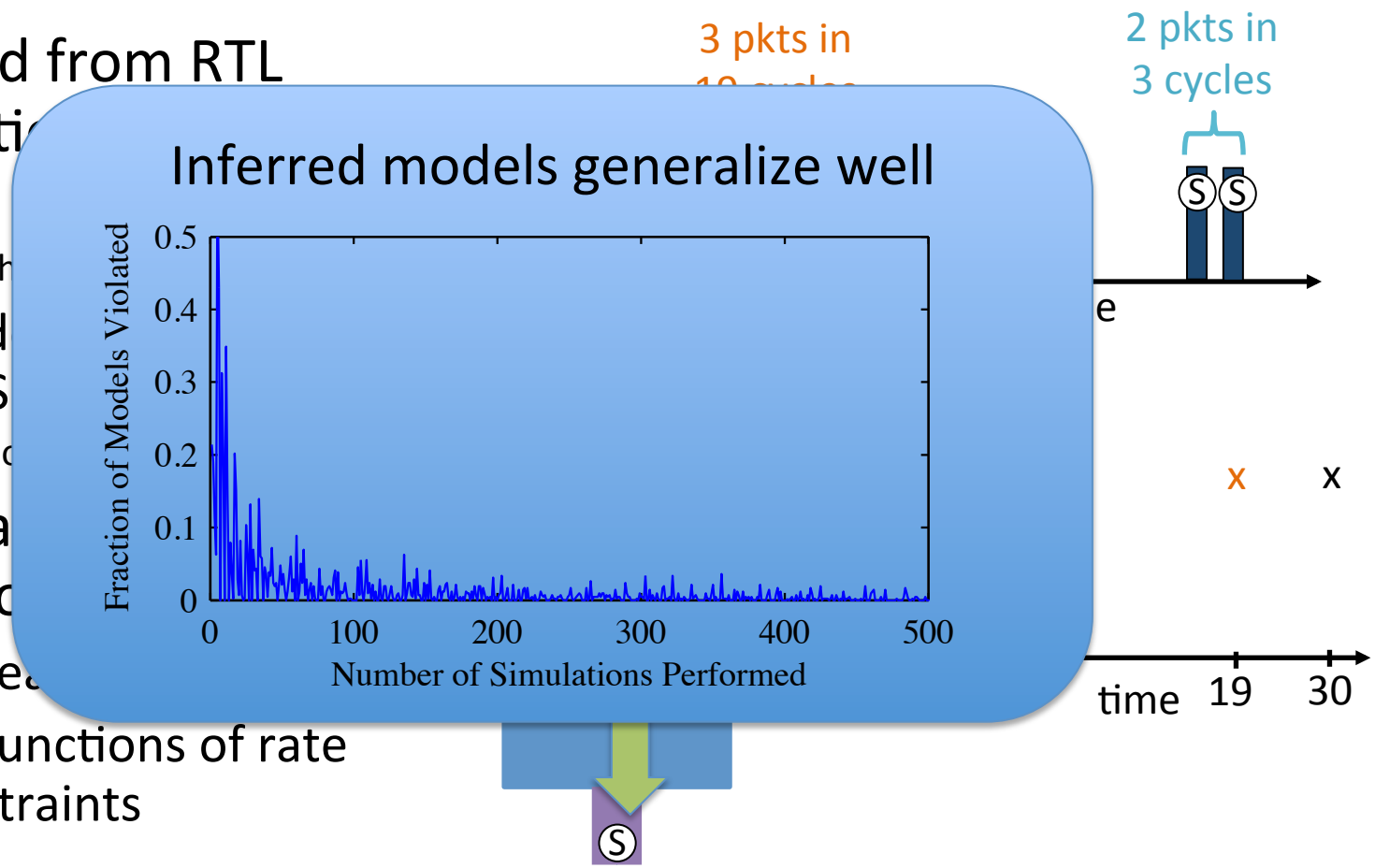






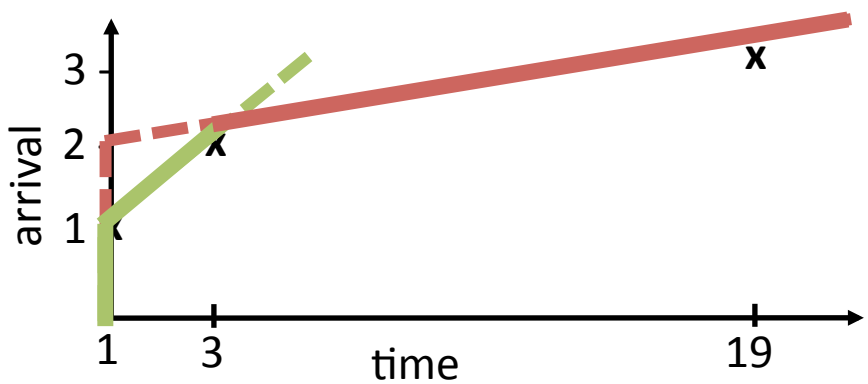
Inferring Formal Traffic Model

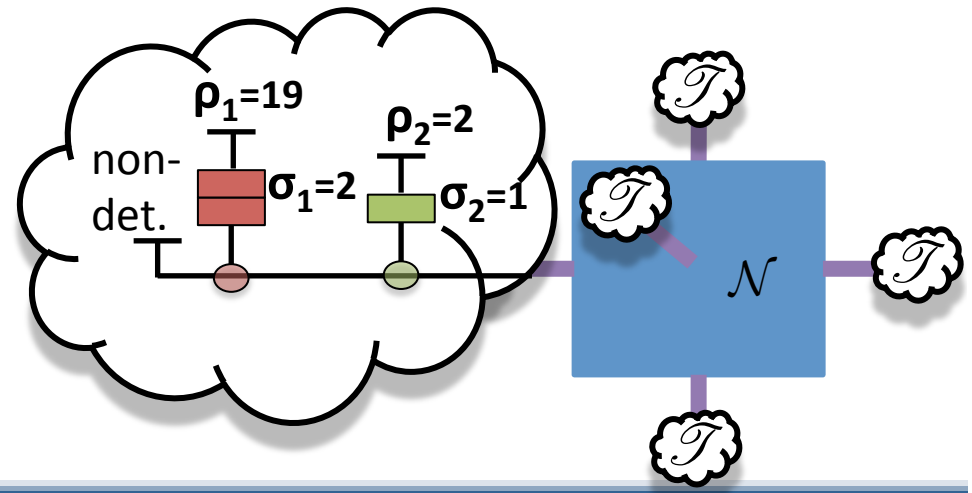
- Inferred from RTL simulati
  - CMP
    [Peh Ph
  - Rand
    PARS
    [Soteric
- Predica
  behavic
  - Boole
  - Conjunctions of rate constraints



Applying Formal Traffic Model

- Non-deterministic choice of allowed destinations
- Rate constraints in xMAS enforced by **token-bucket** regulation of non-deterministic sources
  - Size of token bucket (σ) constrains traffic bursts
  - Token injection frequency (ρ) constrains long-time average rate (1/ρ)

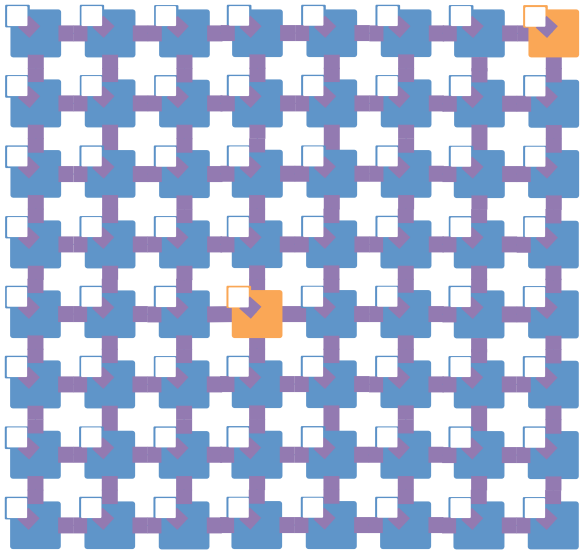




Traffic modeling -- Conclusions

- UCLID Symbolic Simulation – SMT theory of Bitvectors
  - 30 cycle BMC with traffic model for sources and eager sinks

Node i	Simple Model		Fortified Model	
	Runtime [seconds]	Latency [cycles]	Runtime [seconds]	Latency [cycles]
0,0	2160	18	3083	15
1,1	7626	23	17454	20
2,2	5413	24	4444	24
3,3	**12060**	**24**	**16014**	**23**
4,4	2851	25	4880	24
5,5	5848	24	18555	20
6,6	6486	24	9927	23
7,7	**3372**	**17**	**4468**	**14**
Non-Det	**1621**	**25**	**1621**	**25**



Runtimes using Boolector 1.4.1 as solver
[Brummayer & Biere '09]

Traffic model leads to tighter verified latency bounds

Summary of Approach

- One strategy for compositional reasoning
  - Decompose both model and latency property along router boundaries
  - Traffic models as interface specs
- Limitations
  - Finding router latency bound is brute force
  - Bounded model checking for local proofs
  - Traffic abstraction has no guarantee of soundness
- Limitations addressed in next work
  - Automated, sound compositions
  - Inductive proofs of latency

Outline

- Background
  - xMAS: Formal model of NoC
  - Verification technology
- Compositional latency verification
  - Abstraction and traffic modeling
  - **<u>Inductive proof by property strengthening</u>**
- Buffer sizing using counterexamples

Compositional Verification

- Strengthen latency property with subgoals
- Reduce required unrolling
- Make latency bounds inductive
- Orders of magnitude speedup

Daniel Holcomb, Alexander Gotmanov, Michael Kishinevsky, and Sanjit A. Seshia. **Compositional Performance Verification of NoC Designs**. In *Proceedings of the 10th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, July 2012.
http://www.eecs.berkeley.edu/~holcomb/memocode12-noc.pdf

Compositional Performance Verification of NoC Designs

Daniel E. Holcomb, *UC Berkeley* — Alexander Gotmanov, *Intel* — Michael Kishinevsky, *Intel* — Sanjit A. Seshia, *UC Berkeley*

Abstract—We present a compositional approach to formally verify quality-of-service (QoS) properties of network-on-chip (NoC) designs. A major challenge to scalability is the need to verify latency bounds for hundreds to thousands of cycles, which are beyond the capacity of state-of-the-art model checkers. We address this challenge by a compositional form of k-induction. The overall latency bound problem is divided into a number of sub-problems, termed *latency lemmas*. Each latency lemma states that a packet spends a smaller number of cycles at a particular "stage" of progress. We present a partially-automated method of computing these stages based on the topology of the network and a subset of relevant state, and verify the latency lemmas using k-induction. The effectiveness of this compositional technique is demonstrated on illustrative examples as well as an industrial ring interconnection network.

I. Introduction

Network on chip (NoC) is a paradigm for communication within large many-core systems on chip (SoC) designs. An NoC architecture is a network of interconnected nodes, where each node has networking logic and is associated with a processor core, memory controller, specialized IP block, etc. Industrial examples include the Tilera TILE64™ processor [1], STI Cell BE [2], and Intel Larrabee [3]. NoCs typically offer quality of service (QoS) guarantees on worst case latency, jitter, and throughput for some classes of network traffic. QoS violations are often similar to starvation and deadlock scenarios and can be easily missed in performance simulation.

High-level modeling of NoCs [4], [5] and automatic abstraction [6] help to hide unnecessary details, easing the way for formal analysis. Even so, verifying QoS properties can be challenging for industrial NoC designs, both due to the scale of the design and the property to be verified. Consider for instance the problem of proving an upper bound on the latency of sending a packet from one node in the network to another. In principle, this property can be expressed in linear temporal logic (LTL), and the problem can be solved using model checking. The LTL property expresses a *bounded liveness property*, written in English as "every packet from source A gets to its destination B within N cycles." Bounded liveness is essentially a safety assertion where one adds some extra logic to track the progress of time. One can use state-of-the-art model checking strategies such as k-induction [7], interpolation [8], and IC3/property-directed reachability (PDR) [9], [10] to verify this property. However, regardless of the strategy, it is necessary to analyze at least N consecutive cycles to either prove or disprove the latency bound, assuming that N is tight. Typical latency bounds for NoCs can be in hundreds or thousands of clock cycles. Unrolling of model transition relation to such depth is beyond the capacity of state-of-the-art model checking engines. Property-directed reachability [9], [10], while avoiding explicit unrolling of the transition relation, still does not scale past tens of clock cycles.

In order to address this challenge, in this paper we use a tried-and-tested approach in formal verification: *compositional reasoning*. In compositional reasoning, one breaks up the overall proof obligation (proving a latency bound of N cycles) into a number of "smaller" proof sub-goals, which are much easier to verify, such that if all of the sub-goals are proved, then so is the original property. The key is to devise a decomposition that is well-suited to the verification task at hand. A natural approach for latency bounds is to first prove smaller bounds on a packet's progress through the network; *e.g.*, how long does it take to inject a packet into the network, how much time does it spend *along* a particular subpath, etc. We term these proof sub-goals *latency lemmas*. Methods to discover and apply them are the core contributions of this paper.

Specifically, we show that for some common network topologies, one can enumerate finitely many *stages* that a packet can go through. Each location in the network belongs to at least one stage at every time moment. Stages are arranged into a directed, acyclic *stage graph*, to capture the order in which they can be visited by a packet. A latency lemma bounds the number of cycles between when a packet is injected into the network and when the same packet exits some particular stage. By proving the latency lemmas for all stages, one proves the bounds corresponding to all paths through the network.

To summarize, we make the following novel contributions in this paper:

- A compositional approach to proving latency bound properties in NoCs by decomposition into latency lemmas.
- Methods of formulating latency lemmas using a *stage graph* based on the topology of the network and control state.
- Experimental results on two illustrative examples show that our approach can reduce the runtime of inductive verification of latency bounds by 4x-55x, and reduce the runtime of the state-of-the-art IC3/PDR technique by 2x. However using k-induction to verify latency with lemmas added gives a 8x-15x overall reduction in runtime relative to verifying the same property using PDR.
- Experimental results on an industrial-style ring interconnection network show that latency lemmas give a significant speedup, and in all cases allow latency bounds to be proved inductively. Several variants of the ring can only be verified within the allotted runtime when latency lemmas are used.

The rest of the paper is organized as follows. Sec. II introduces basic terminology and sketches our approach using a simple example. Sec. III describes our compositional approach in detail. Sec. IV describes our strategy for creating a stage graph. Results for illustrative examples are presented in Sec. V, and for the ring

Scalability of Induction

- Inductive proofs are efficient because of reduced unrolling
  - No model decomposition

$$p(s_i) \wedge R(s_i, s_{i+1}) \implies p(s_{i+1})$$

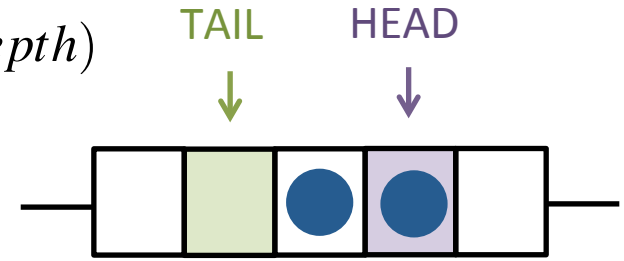
$$\left(\bigwedge_{i \in [j, j+k-1]} p(s_i) \wedge \bigwedge_{i \in [j, j+k-1]} R(s_i, s_{i+1}) \right) \implies p(s_{j+k})$$

- <u>Two challenges:</u>
  - General initial state
  - Proof may require large k

Auxiliary Invariants

- Auxiliary invariant Ψ blocks off unreachable states from verifier
  - Helps restrict general initial state to good states
  - Otherwise can include deadlock
- Simplicity of xMAS enables helps create Ψ
  - Numeric invariants [Chatterjee et al., CAV'10]
  - Channel persistency [Gotmanov et al., VMCAI'11]
  - Structural queue invariants

$$tail = head + num \pmod{depth}$$
$$tail < depth$$
$$head < depth$$



Verifying Global Latency Bounds

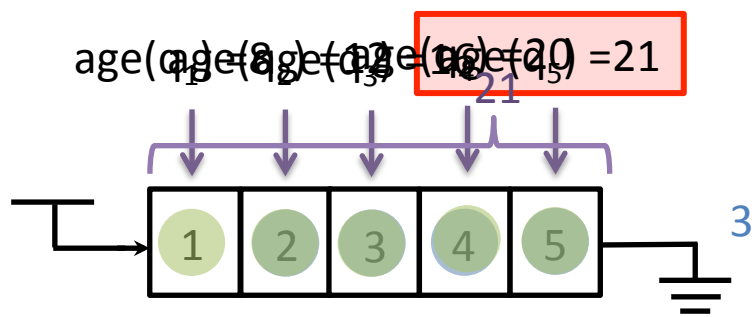
- Prove end-to-end latency of 21 cycles

$$\Phi_{21}^{G} := \bigwedge_{used_i} age(q_i) < 21$$

- Strengthened with auxiliary invariant

$$\mathcal{N} \models \Phi_{21}^{G} \wedge \Psi$$

Induction depth k is too large to be of much use



Strengthening Latency Property

- Strengthen property with Age Lemmas
  - Impose "**stage graph**" onto xMAS network
  - Precise composable subgoals
- Each age lemma ϕ_j defined by:
  - t_j Precise age bound for stage j
  - $p_{i,j}$ =True iff packet in slot i maps to stage j

$$\mathcal{N} \models \Phi^G \wedge \Psi \wedge \Phi^L$$

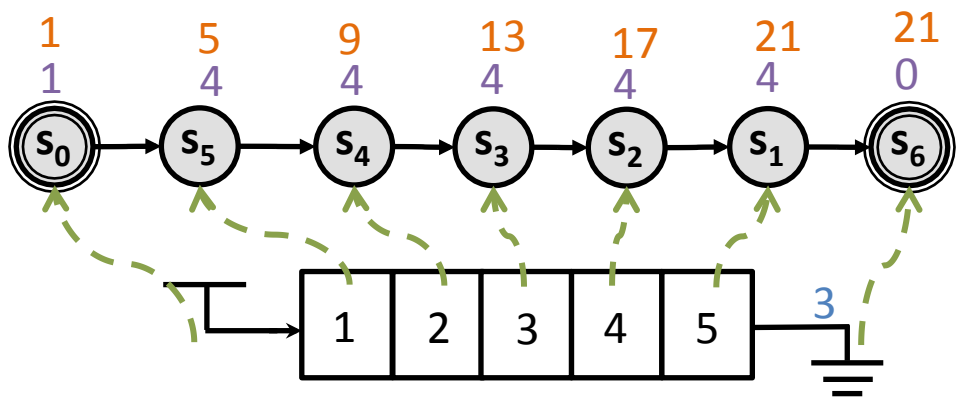
$$\phi_j := \bigwedge_i (p_{i,j} \implies age(q_i) < t_j)$$

$$\Phi^L := \bigwedge_j \phi_j$$

$$T_L := \max_{j \in stages} (t_j)$$

$$\Phi^L \implies \Phi^G_{T_L}$$

d_j = max time at stage j

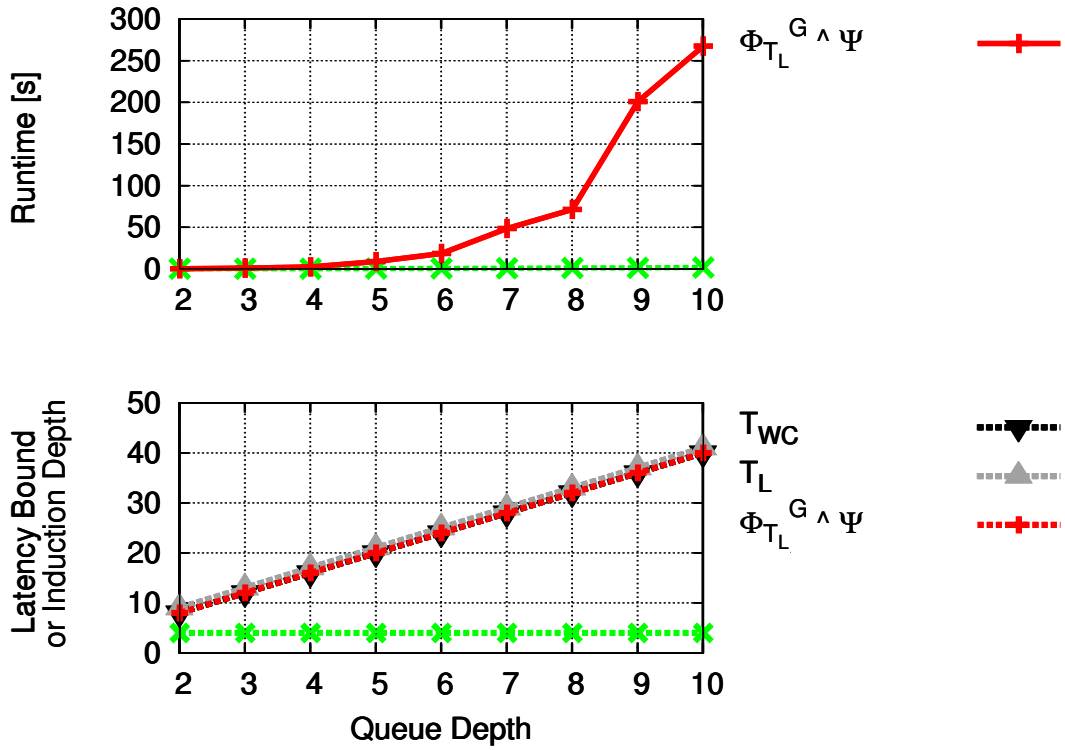


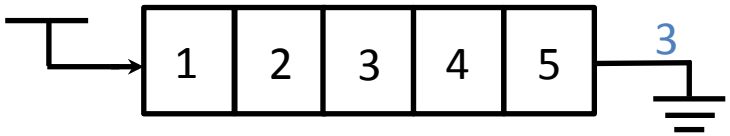
0.37s, k=4
vs
8.70s, k=20

Why Compositional?

Global bound: Induction depth proportional to worst-case latency

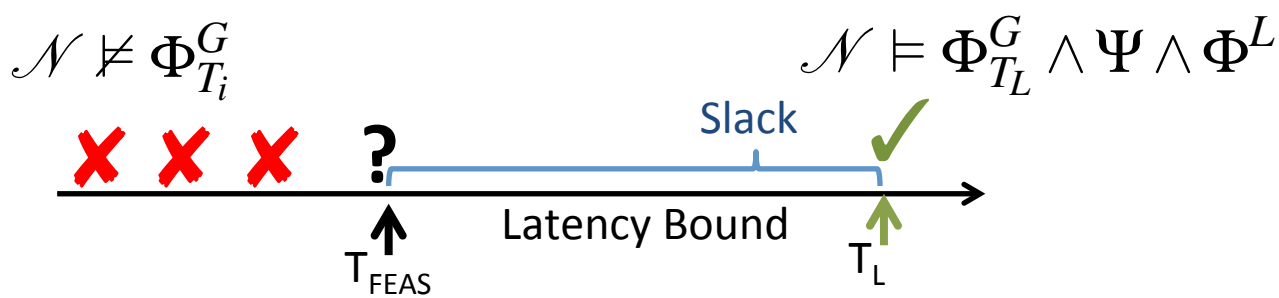
Compositional: Induction depth proportional to size of subgoal





Tightness of Bound from Stage Graph?

- Age lemmas are conservative with respect to time at each stage of progress
  - But how loose is implied bound T_L?
- Use BMC to disprove smaller latency bounds
  - Sweep to find largest disprovable
  - Conservative estimate of slack



Timing Implications of xMAS Transfer Signals

Boolean condition for channel b target ready?

~~b.trdy~~

~~a.irdy and o.trdy~~

expand readiness conditions

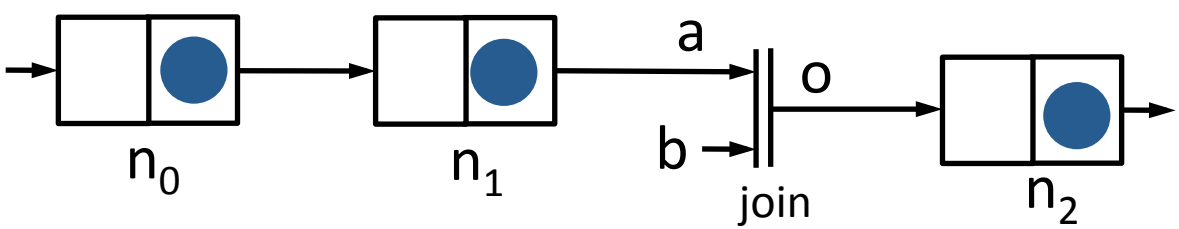
How long until b.trdy asserted?

$n_1 \neq 0 \wedge n_2 \neq 2$ ⟷ 0 cycles

$n_0 \neq 0 \wedge n_1 = 0 \wedge n_2 \neq 0$ ⟷ 1 cycles

otherwise ⟷ ∞ cycles

otherwise ⟷ ∞ cycles



Definition of Join:

a.trdy := o.trdy ^ b.irdy

b.trdy := o.trdy ^ a.irdy

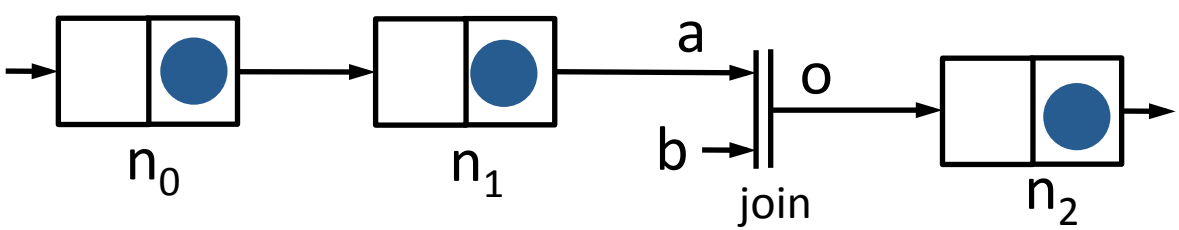
o.irdy := a.irdy ^ b.irdy

Timing Implications of xMAS Transfer Signals

Idea: reason about future readiness of signals in way that is analogous to current readiness of Boolean signals

conditions

$n_1 \neq 0 \wedge n_2 \neq 2$	⟷	0 cycles
$n_0 \neq 0 \wedge n_1 = 0 \wedge n_2 \neq 0$	⟷	1 cycles
otherwise	⟷	∞ cycles
otherwise	⟷	∞ cycles



Definition of Join:

a.trdy := o.trdy ^ b.irdy

b.trdy := o.trdy ^ a.irdy

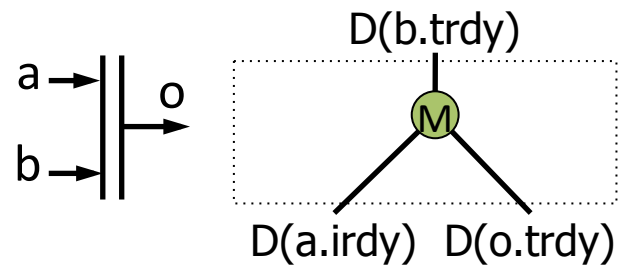
o.irdy := a.irdy ^ b.irdy

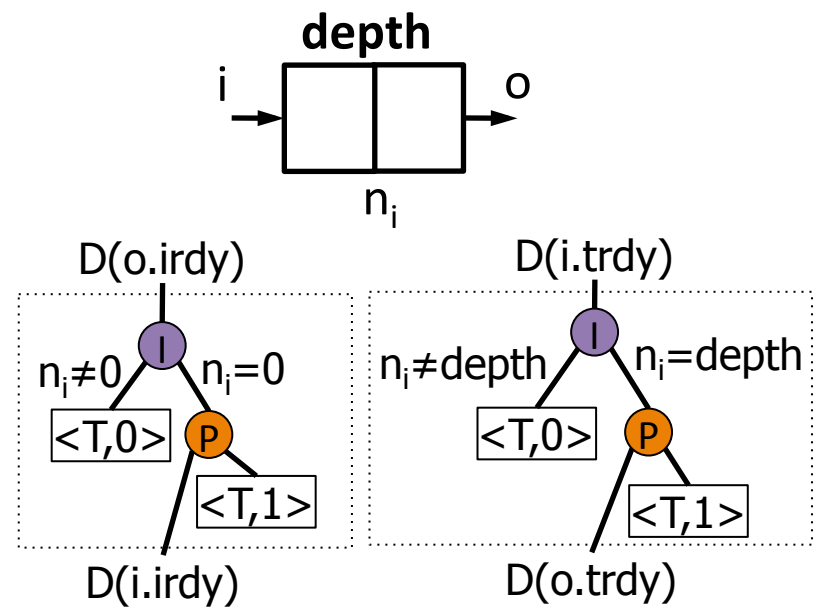
Future Readiness Bounds

- Want to reason about future readiness time of irdy/trdy
  - Depends on network state
- Represent conditionally:
  - D(trdy) = {<g_0,δ_0>, <g_1,δ_1>,…, <g_N,δ_N>}
  - g_i is a conjunction of queue predicates
  - δ_i is a numeric bound on readiness from state satisfying g_i

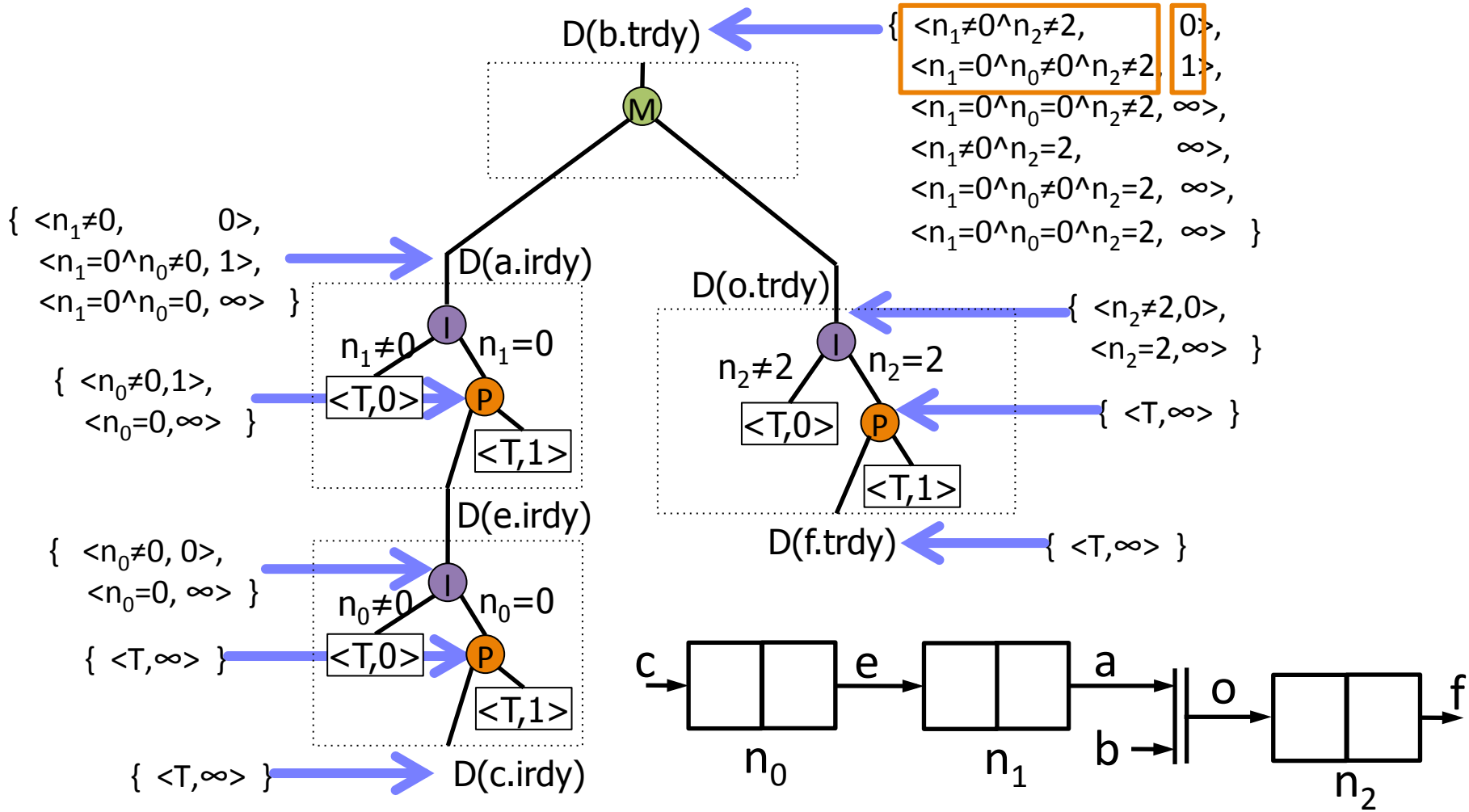
$$g_i \implies \mathbf{F}^{<\delta_i}\ trdy$$

- Rule-based propagation
  - Analogous to Boolean transfer formulas for current readiness in xMAS
  - Manipulated using operations Max,Plus,ITE

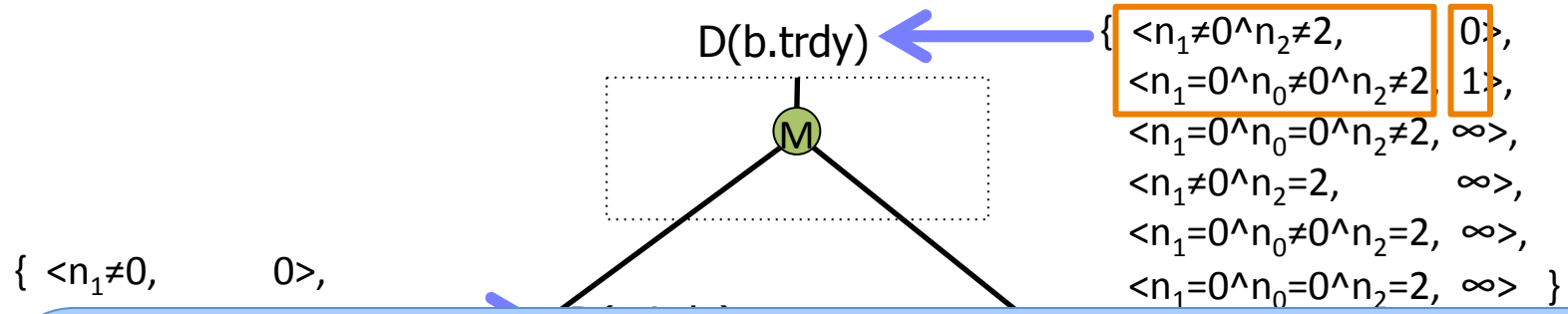




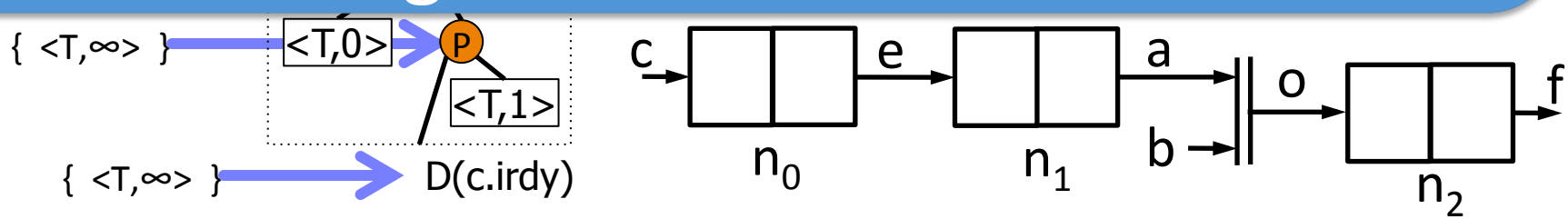
Deriving Progress Lemmas



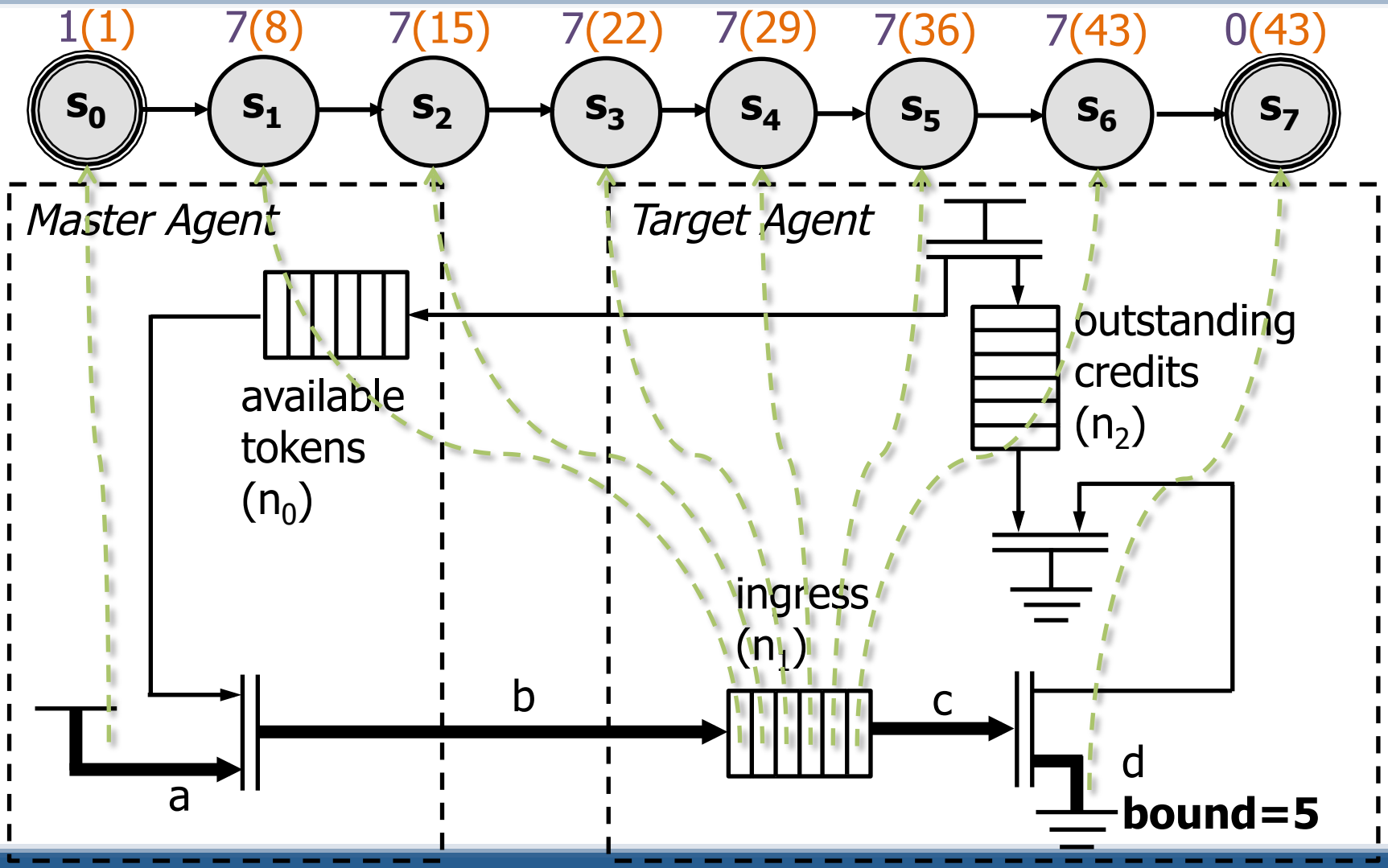
Deriving Progress Lemmas



- Property Θ asserts that network state satisfies a condition with a finite bound
- If Θ is valid, then no more than 1 cycles of blocking occur

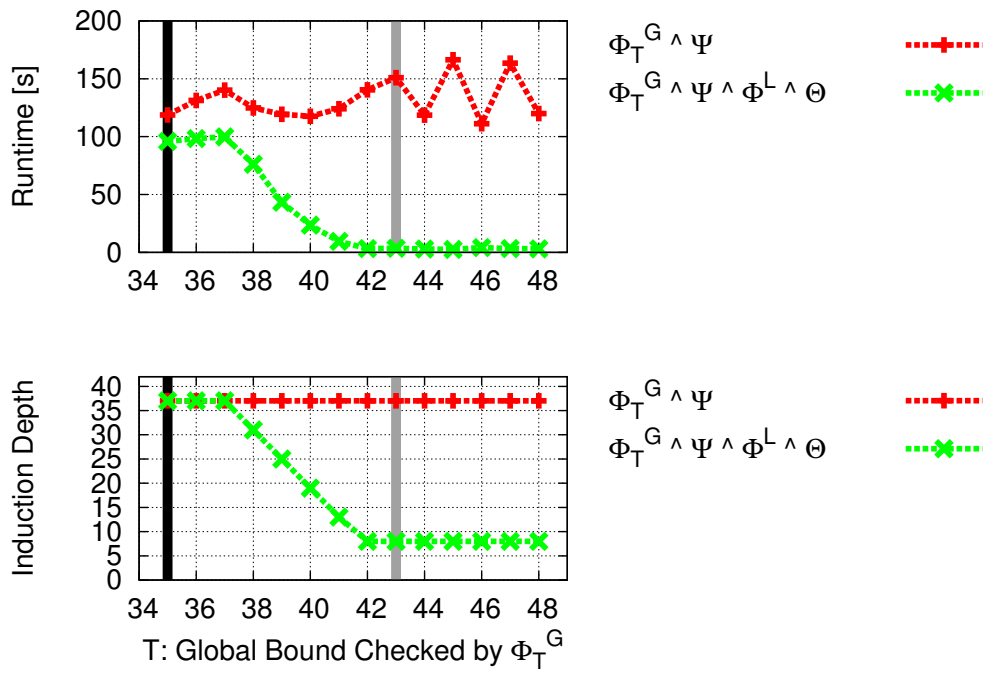


Credit Loop Network

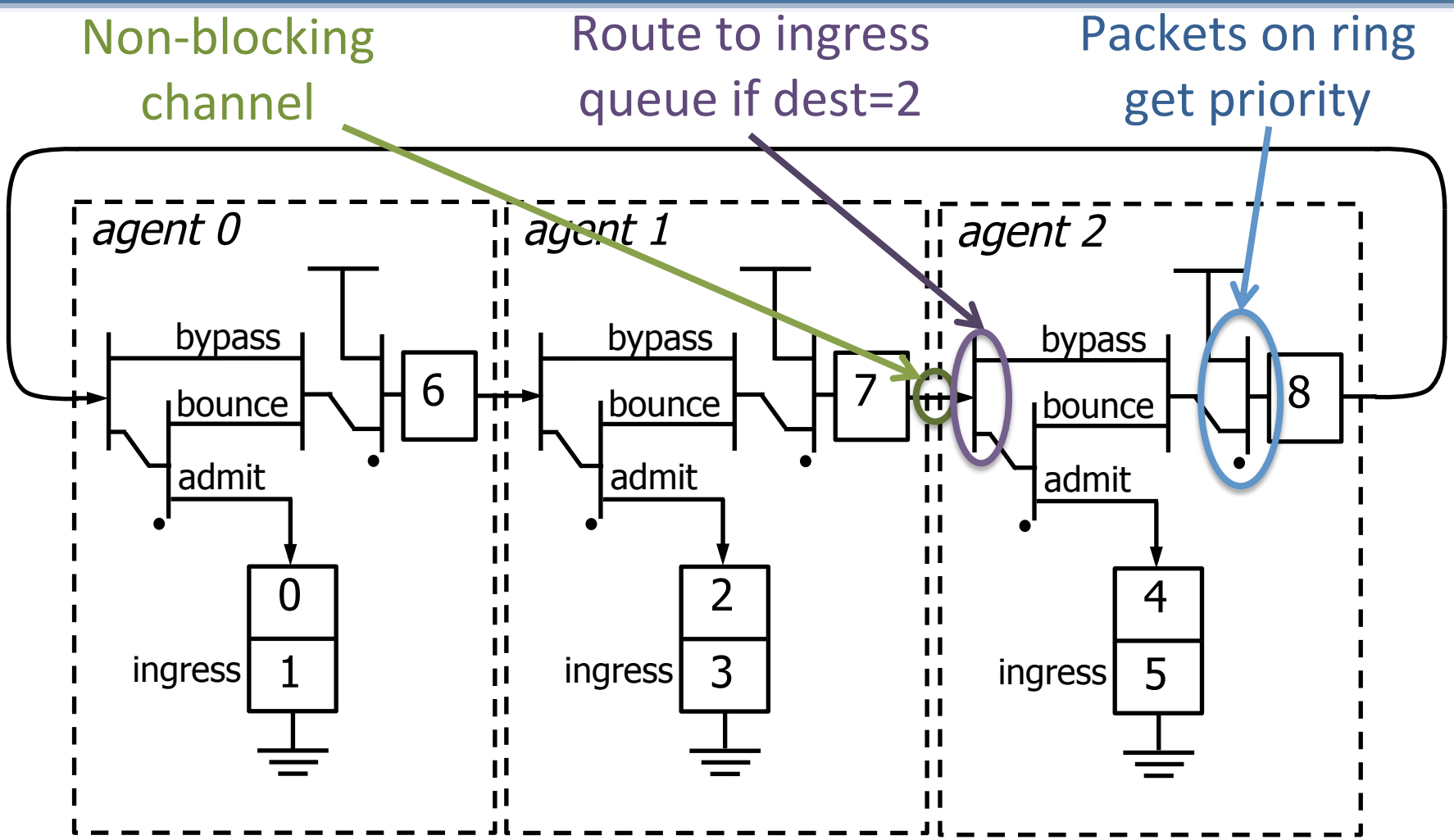


Trading Looseness against Runtime

find bound $T_{FEAS} \equiv 35$	Runtime (s)	Frames	Cex	Engine	Property
	52.79	42	Y	bmc	Φ_{34}^{G}
	1045.62	200	-	bmc	Φ_{35}^{G}

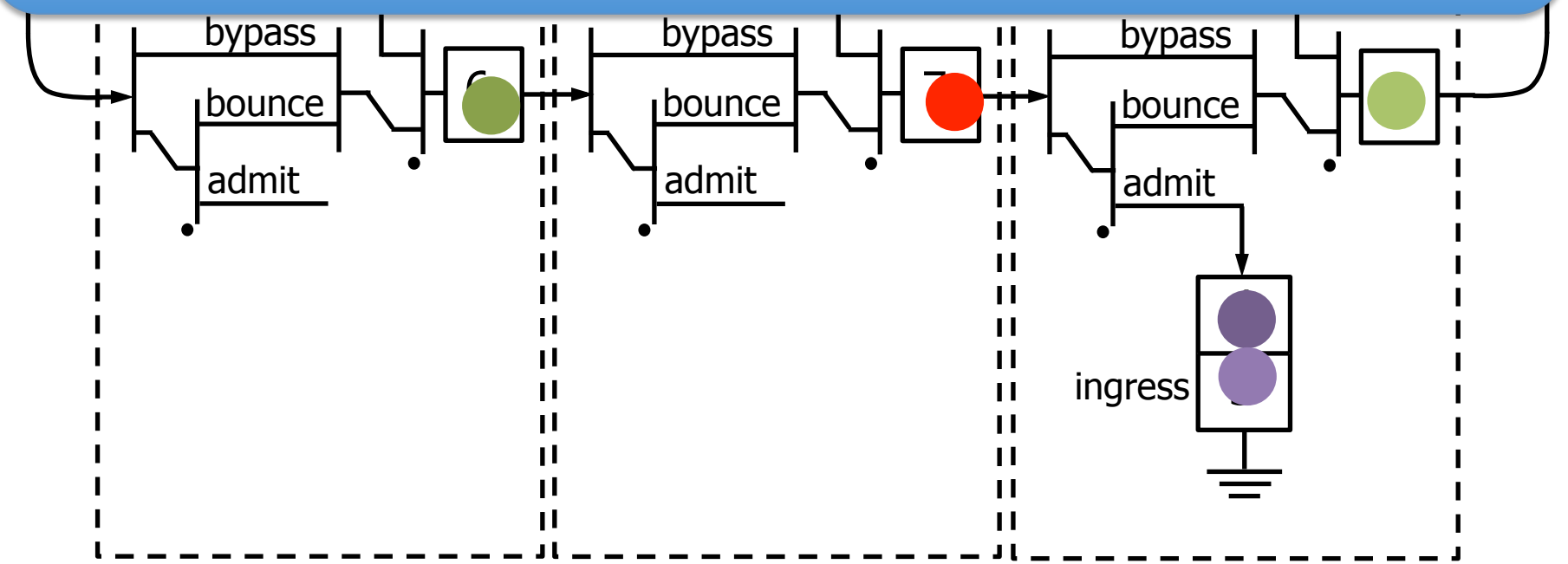


Non-Stalling Ring Interconnect



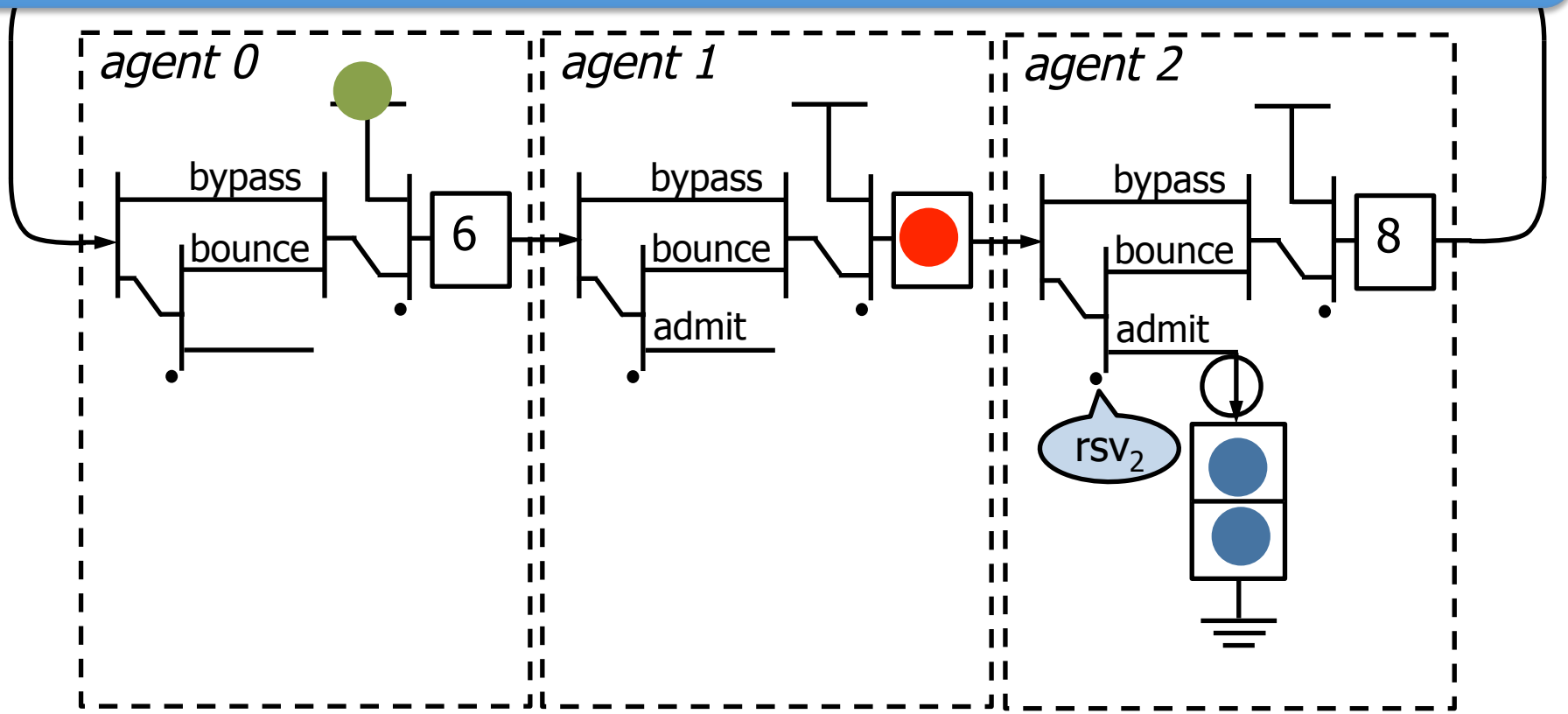
Non-Stalling Ring Interconnect

- No progress without reservations

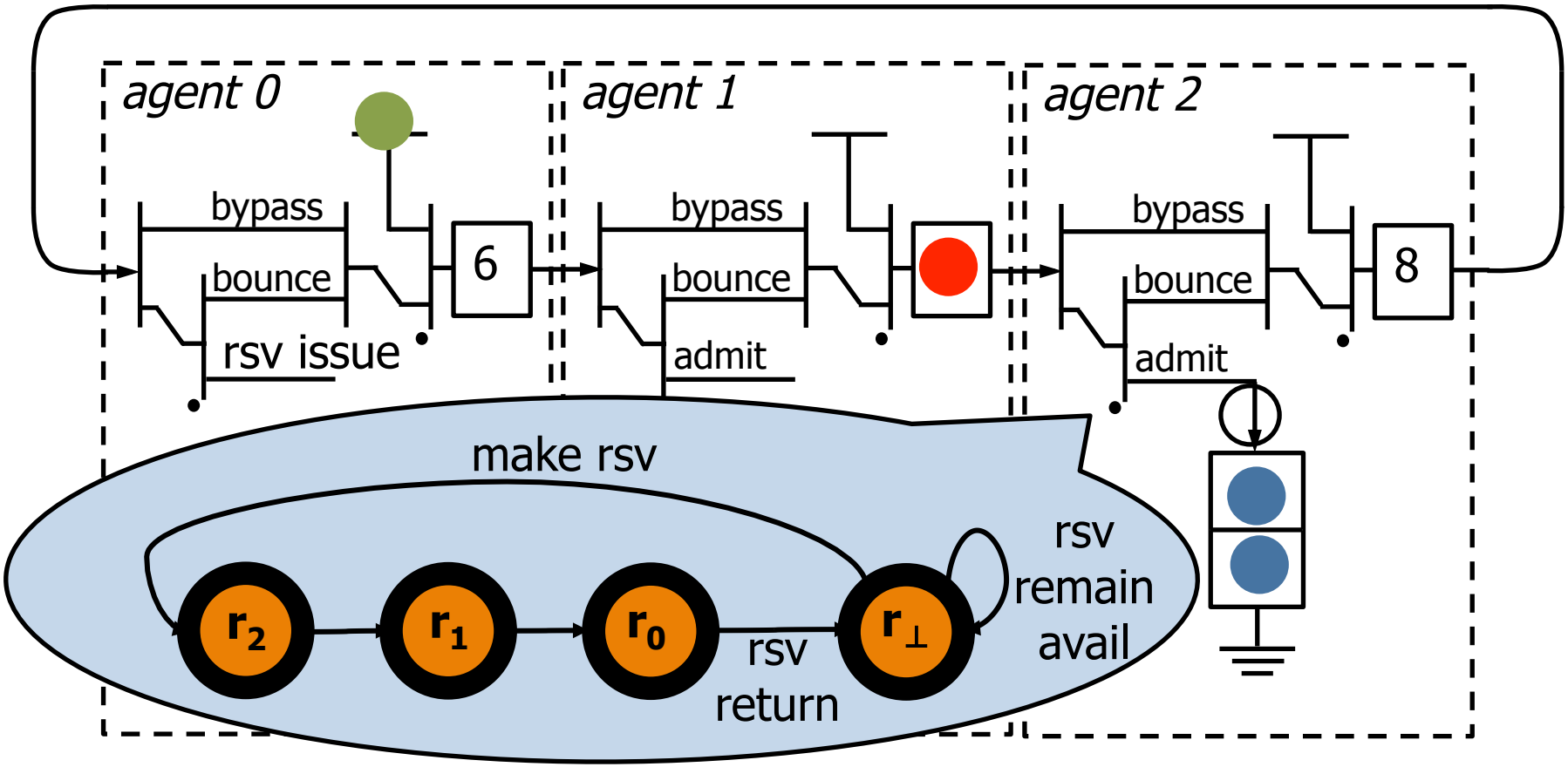


Receive Reservation in Ring

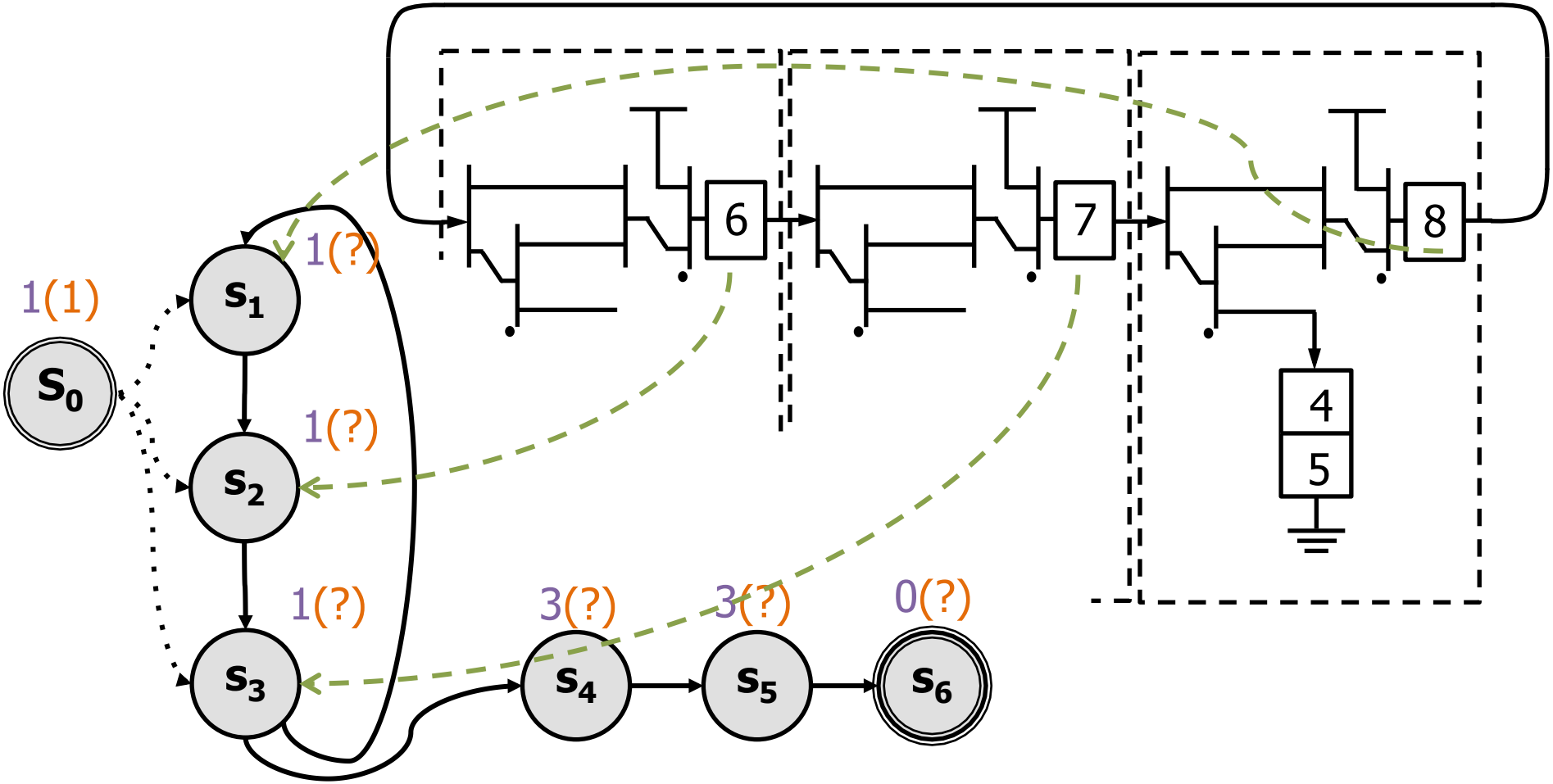
- Livelock prevention to ensure finite latency [Mattina et al]
- Fair ingress admission with respect to packets on the rings



Receive Reservation in Ring

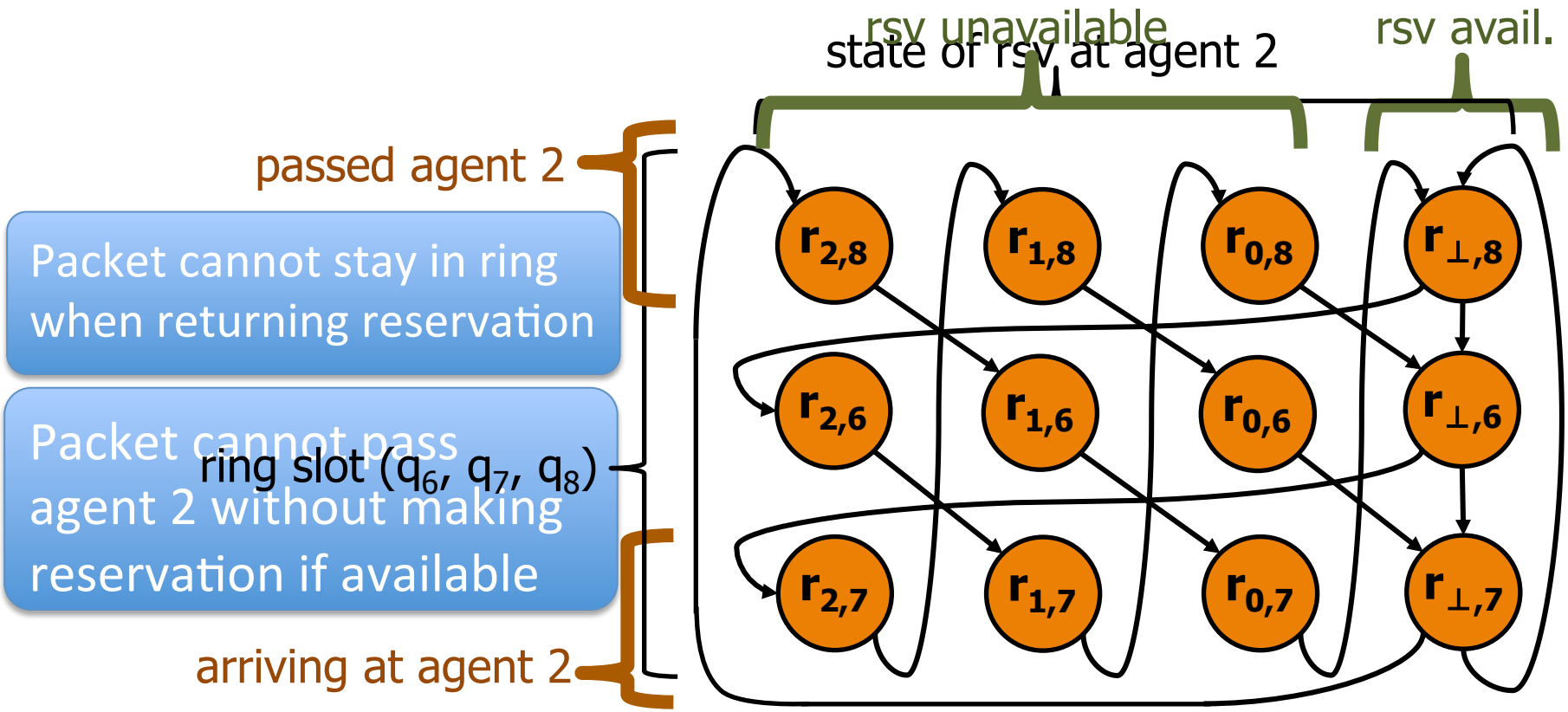


Naïve Stage Graph of Ring

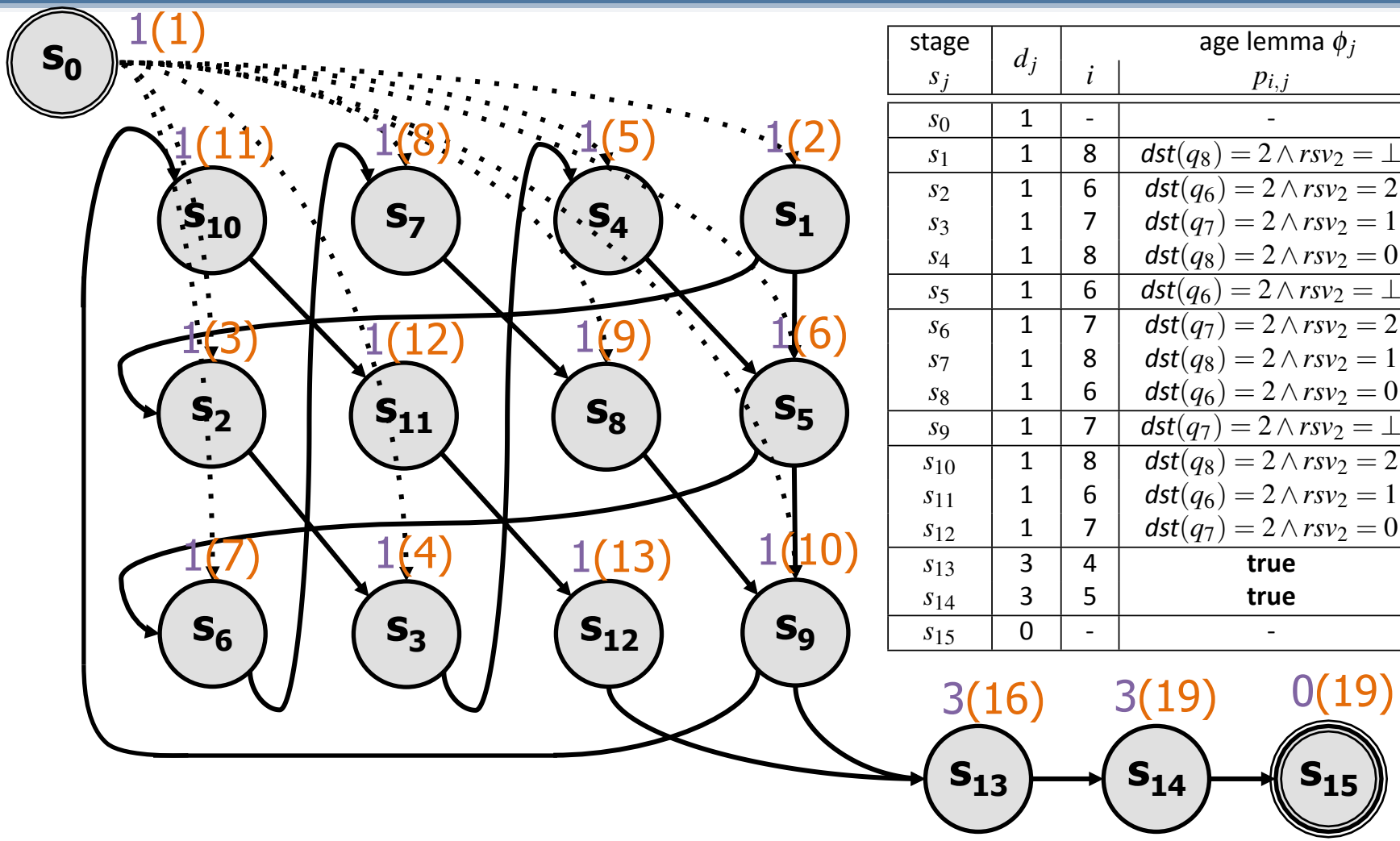


Toward Acyclic Stage Graph for Ring

- Product automaton of ring and reservation logic



Stage Graph for 3-Agent Ring



stage s_j	d_j	age lemma ϕ_j i	$p_{i,j}$	t_j
s_0	1	-	-	1
s_1	1	8	$dst(q_8) = 2 \wedge rsv_2 = \perp$	2
s_2	1	6	$dst(q_6) = 2 \wedge rsv_2 = 2$	3
s_3	1	7	$dst(q_7) = 2 \wedge rsv_2 = 1$	4
s_4	1	8	$dst(q_8) = 2 \wedge rsv_2 = 0$	5
s_5	1	6	$dst(q_6) = 2 \wedge rsv_2 = \perp$	6
s_6	1	7	$dst(q_7) = 2 \wedge rsv_2 = 2$	7
s_7	1	8	$dst(q_8) = 2 \wedge rsv_2 = 1$	8
s_8	1	6	$dst(q_6) = 2 \wedge rsv_2 = 0$	9
s_9	1	7	$dst(q_7) = 2 \wedge rsv_2 = \perp$	10
s_{10}	1	8	$dst(q_8) = 2 \wedge rsv_2 = 2$	11
s_{11}	1	6	$dst(q_6) = 2 \wedge rsv_2 = 1$	12
s_{12}	1	7	$dst(q_7) = 2 \wedge rsv_2 = 0$	13
s_{13}	3	4	**true**	16
s_{14}	3	5	**true**	19
s_{15}	0	-	-	19

Ring Verification Results

- Prove 79 cycle bound in 10 frames of induction
- Proved bound is 1 cycle loose
- Speedup of 65x to >130x

find bound $T_{FEAS} \equiv 18$	Runtime (s)	Frames	Cex	Engine	Property
	24.12	20	Y	bmc	Φ^G_{17}
	868.28	200	-	bmc	Φ^G_{18}

verify bound $T_L \equiv 19$	Runtime (s)	Frames	Proved	Engine	Property
	62.34	18	Y	kind	$\Phi^G_{19} \wedge \Psi$
	1.31	**4**	**Y**	**kind**	$\Phi^G_{19} \wedge \Psi \wedge \Phi^L \wedge \Theta$
	88.39	12	Y	pdr	$\Phi^G_{19} \wedge \Psi$
	6.57	14	Y	pdr	$\Phi^G_{19} \wedge \Psi \wedge \Phi^L \wedge \Theta$

3-agent ring

find bound $T_{FEAS} \equiv 78$	Runtime (s)	Frames	Cex	Engine	Property
	3901.95	80	Y	bmc	Φ^G_{77}
	10,000.00	111	-	bmc	Φ^G_{78}

verify bound $\equiv T_L \equiv 79$	Runtime (s)	Frames	Proved	Engine	Property
	10,000.00	-	-	kind	$\Phi^G_{79} \wedge \Psi$
	75.28	**10**	**Y**	**kind**	$\Phi^G_{79} \wedge \Psi \wedge \Phi^L \wedge \Theta$
	10,000.00	-	-	pdr	$\Phi^G_{79} \wedge \Psi$
	662.15	73	Y	pdr	$\Phi^G_{79} \wedge \Psi \wedge \Phi^L \wedge \Theta$

8-agent ring

Inductive Proof via Strengthening -- Conclusions

- Inductive verification of latency property on a bit-level RTL network from xMAS model
  - ✓ Orders of magnitude verification speedup
  - ✓ Models complex arbitration behaviors
  - ✓ Sound composition
  - ✗ Not automated for cyclic networks

Knowledge of higher-level network structure produces useful invariants for bit-level verification

Outline

- Background
  - xMAS: Formal model of NoC
  - Verification technology
- Compositional latency verification
  - Abstraction and traffic modeling
  - Inductive proof by property strengthening
- **<u>Buffer sizing using counterexamples</u>**

Buffer Sizing

- Simulate with symbolic-sized queues
- Assert that a QoS property is violated
- Counterexample reveals valid buffer size

Bryan Brady, Daniel Holcomb, and Sanjit A. Seshia. **Counterexample-Guided SMT-Driven Optimal Buffer Sizing**. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 329–334, March 2011.

http://www.eecs.berkeley.edu/~holcomb/date11-noc.pdf

Counterexample-Guided SMT-Driven Optimal Buffer Sizing

Bryan A. Brady[1] Daniel Holcomb[1] Sanjit A. Seshia
UC Berkeley, EECS Department
{bbrady,holcomb,sseshia}@eecs.berkeley.edu

Abstract—The quality of network-on-chip (NoC) designs depends crucially on the size of buffers in NoC components. While buffers impose a significant area and power overhead, they are essential for ensuring high throughput and low latency. In this paper, we present a new approach for minimizing the cumulative buffer size in on-chip networks. Our approach is based on counterexample-guided synthesis, given high-level specifications on traffic behavior. Our approach uses model checking based on satisfiability modulo theories (SMT) solvers, within an overall counterexample-guided synthesis loop. We demonstrate the effectiveness of our technique on NoC designs involving arbitration, credit logic, and virtual channels.

I. Introduction

Network-on-chip (NoC) is a promising paradigm for communication within large system-on-a-chip (SoC) designs. An NoC architecture consists of a network of interconnected nodes, where each node can be a processor core or a specialized IP block. Inter-node communication is performed by the transmission of data packets through routers, which typically have a number of buffers. In this paper, we address a key problem in the design and implementation of NoCs — the minimization of buffer size needed to guarantee a particular quality of service.

Buffers play a critical role in NoC design: increasing the sizes of the buffers can significantly reduce the average latency of packets and hence increase the overall throughput. However, even with scalable communication architectures where routers only exchange data with their neighbors (e.g. [1]), the size of each input-channel buffer has a serious impact on the overall area and power of a NoC router design. For example, Ho and Mascarenco [2] indicate that changing the buffer size at each input channel from 2 words to 3 words will increase the overall area by 30% for a 4 × 4 network. The buffer sizing problem is further complicated by the heterogeneity of traffic patterns in NoCs. For example, one needs to allocate more buffers in a more heavily loaded channel. Hence, we need a way to judiciously allocate buffer capacity for each channel to match the traffic patterns characterizing various applications.

In this paper, we propose a formal technique for *minimizing the cumulative buffer size* while meeting design and performance constraints with respect to specified traffic patterns. Our approach has the following key characteristics. First, we describe an NoC design formally using components drawn from a small set of primitives, similar to the recent XMAS approach [3]. Second, we employ *term-level modeling* [4], [5] to synthesizing expressions for buffer sizes in terms of symbolic variables. We use a technique for representing designs as suitable fragments of first-order logic, which can then be analyzed using satisfiability modulo theories (SMT) solvers [6]. In our model, the traffic injected into the network is non-deterministic but obeys bounds on average rate and burst size. The destination addresses are described using uninterpreted functions. Given this formal model, the third characteristic of our approach is the use of SMT-based model checking to find the minimal buffer sizes that guarantee some throughput and latency for specific traffic patterns. The approach is based on counterexample-guided synthesis, where we repeatedly invoke an SMT-based model checker both for synthesizing buffer sizes and for checking whether the synthesized buffer sizes guarantee the performance property for specified traffic patterns. Buffer sizings found with our approach guarantee performance for all specified traffic patterns, without having to explicitly enumerate each one.

In summary, we make the following novel contributions in this paper.

1. We propose a new SMT-based model checking technique, based on counterexample-guided synthesis, for minimizing the cumulative buffer size in an NoC design.
2. We show how a formal term-level approach can be effective for modeling general NoC designs as well as traffic patterns.
3. We demonstrate the effectiveness of our technique on NoC designs involving arbitration, credit logic, and virtual channels.

Our paper is organized as follows. We review related literature in Section II. Section III describes our formal model for NoC designs and defines the buffer minimization problem. We describe our SMT-based approach in Section IV. Experimental results are presented in Section V and we conclude in Section VI.

II. Related Work

The problem of buffer minimization has been widely studied in the digital signal processing (DSP) community. Synchronous dataflow (SDF) models [7] in particular are used to reason about the minimum buffer size required for a feasible schedule to exist in order for the model to be deadlock-free and to conform to timing constraints [8]. In general, this buffer minimization problem is NP-complete [9].

Various techniques have been applied to address the NP-completeness. Prayfurko et al. [10] use an SDF model of an NoC to compute sizings for a fixed-size internal buffer. Stuijk et al. [11] use model checking to determine whether there exists a buffer sizing smaller than some bound that admits a deadlock-free schedule; the minimal sizing is found using iterated calls to the model checker. Stuijk et al. [8] present a dynamic-programming algorithm that generates a set of candidate buffer sizings that is guaranteed to contain all Pareto-optimal points in the buffer-size versus throughput space; the Pareto-optimal points among the set are then found by self-timed simulation. Wiggers et al [12] approximate minimal buffer sizing for a given throughput using a network-flow formulation, but the closeness of approximation is not bounded.

[1]Joint first authors.
978-1-9618893-7-9/DATE11 © 2011 EDAA

Buffer Sizing

Unsized network $\mathcal{N}$

Traffic model $\mathcal{T}$

Performance property ϕ

CEBUS

Minimal sizing S such that $\mathcal{N}[S] \| \mathcal{T} \models \phi$

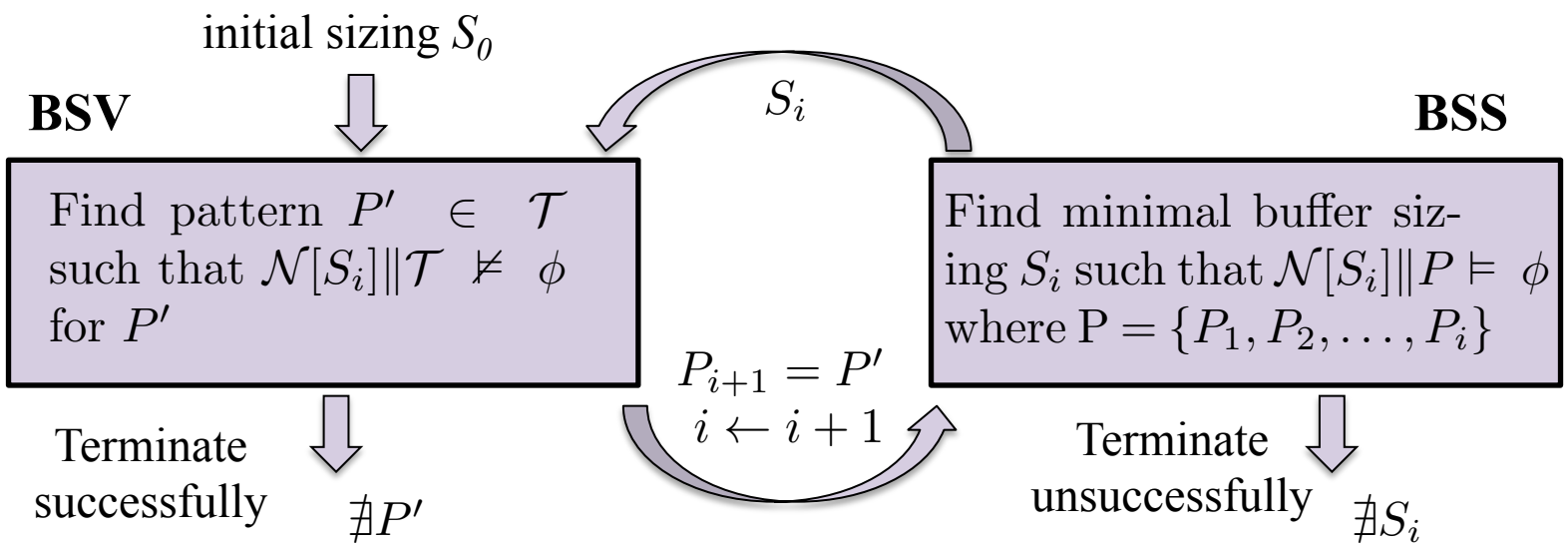
$$\phi = \phi_{latency} \wedge \phi_{non-blocking}$$

[Clarke et al. CAV'00]
[Frehse et al. HSCC'08]
[Solar-Lezama et al. ASPLOS'06]

- Need to address quantifier alternation

$$\exists S_i \, \forall P' \in \mathcal{T}. \;\; (\mathcal{N}[S_i] \| P' \models \phi)$$

Quantifier Instantiation



Check validity of formula

$$\mathcal{N}[S_i]\|\mathcal{T} \models \phi$$

- **Valid:** sizing S_i ensures ϕ
- **Invalid:** counterexample is traffic pattern P' that causes $\neg\phi$

Check validity of formula

$$|S_i| < size \implies \neg(\mathcal{N}[S_i]\|P \models \phi)$$

- **Valid:** no S_i ensures ϕ for all P
- **Invalid:** Counterexample is sizing S_i that ensures ϕ

Buffer Sizing -- Conclusions

Completeness has a cost -- even for network of 3 queues

Verification

- Checking same formula at each iteration i
- More difficult SAT problem at later iterations
  - Few satisfying assignments to be found

Iteration i	*Buffer-Size Verification (BSV)*					
	CNF Size		Runtime (sec)			
	Vars	Clauses	SAT	Enc	SSim	Total
0	158K	473K	1.4	17.5	1.3	20.2
2	158K	473K	0.9	17.7	1.2	19.9
4	158K	473K	8.9	17.7	1.3	28.0
6	158K	473K	16.8	17.7	1.3	35.8
8	158K	473K	91.7	17.9	1.3	111.0
10	158K	473K	242.8	17.7	1.3	261.8
11	158K	473K	106.0	17.9	1.2	125.3
12*	158K	473K	373.5	16.2	2.7	392.5

Synthesis

- Problem size linear in i
- Binary search to minimize
- Difficulty of SAT problem not necessarily proportional to size
  - Symbolic sim. and decision proc. encoding dominate runtime

Iteration i	*Buffer-Size Synthesis (BSS)*					
	CNF Size		Runtime (sec)			
	Vars	Clauses	SAT	Enc	SSim	Total
0	79K	236K	5.6	69.0	11.0	85.7
2	100K	300K	2.8	60.8	16.0	79.7
4	217K	650K	5.7	79.7	25.1	110.6
6	259K	778K	15.0	195.3	60.2	270.6
8	345K	1037K	27.2	234.5	91.4	353.2
10	429K	1287K	39.7	342.2	126.7	508.8
11	486K	1458K	61.0	392.9	155.6	609.6

Summary

- Formal methods are promising approach for synthesizing and verifying NoC QoS
- Achieved several orders of magnitude speedup over monolithic model checking of latency properties without strengthening
- Not yet a push-button solution for general network models, but xMAS methodology helps

Thank you

Leverage model checking for solving NoC QoS latency problems. Address capacity limitations by extending well-known formal techniques including abstraction and compositional reasoning into the NoC domain

- 3 specific QoS contributions
  - Workload abstraction of traffic models
  - Latency proofs by property strengthening
  - Optimal buffer sizing for QoS

Compositional Latency Verification