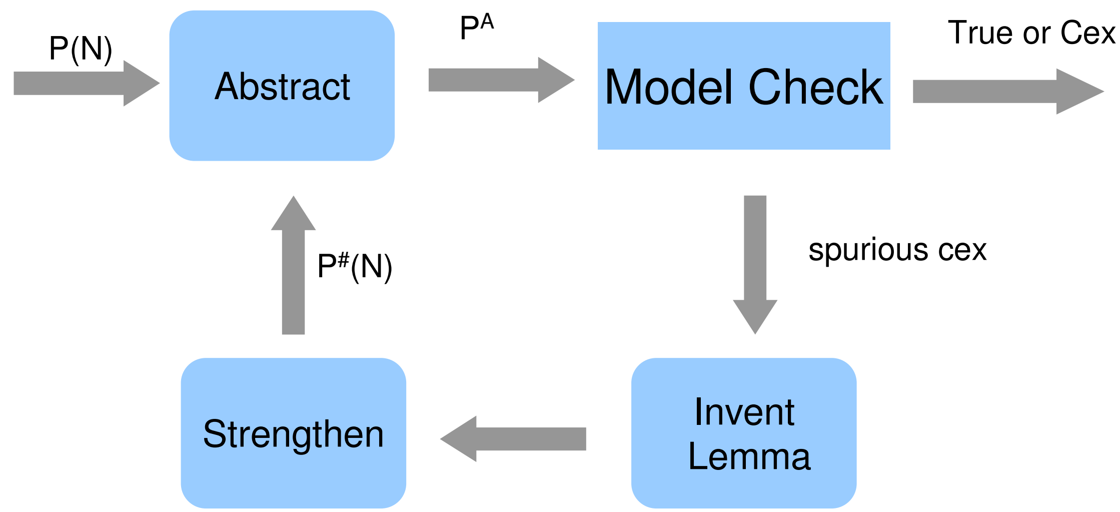
# Protocol Verification using Flows: An Industrial Experience

Murali Talupur

Joint work with
John O' Leary, Mark R. Tuttle
SCL, Intel Corp

# Parametric Verification using Flows



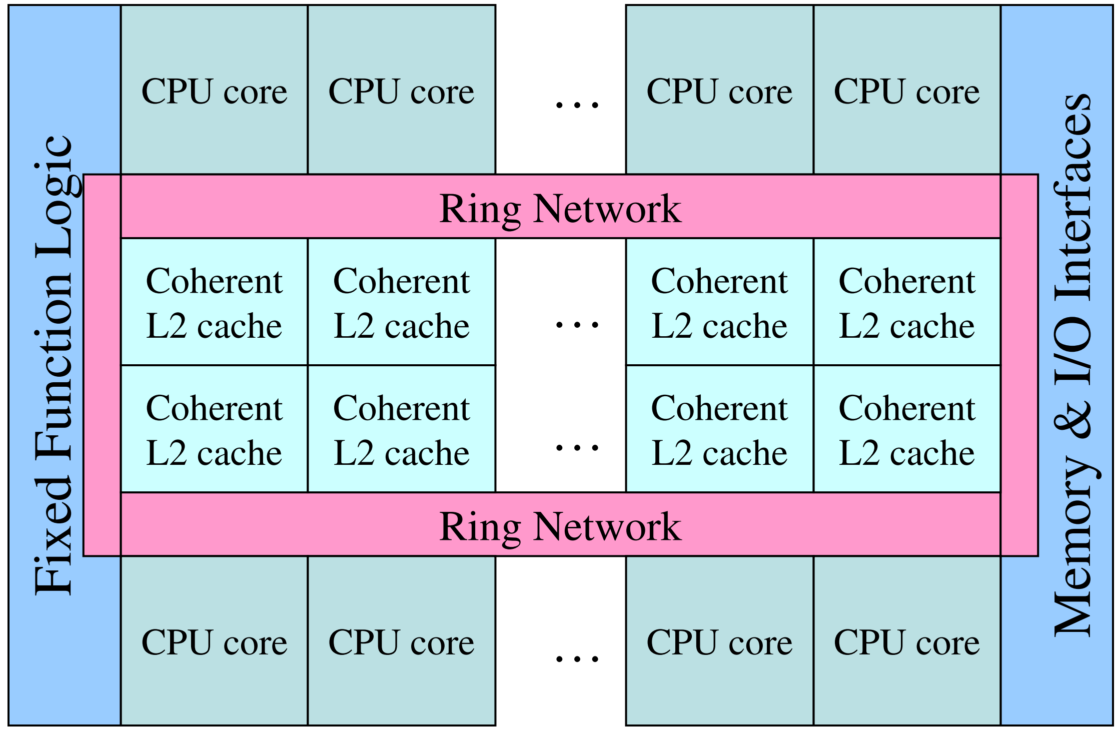

Last year we introduced the ***CMP + Flows*** method for parametric protocol verification *[FMCAD08]*

1. CMP is an abstraction & compositional reasoning based method
   1. Uses Model Checker as a proof assistant
   2. Requires user guidance
2. Demonstrated that "flows" yield powerful invariants
   1. Partial orders on "events"
   2. Available for free
3. Applied it to German and Flash

# Verification of Larrabee Cache Protocol

This year we applied the method to a ***real, state-of-the-art cache protocol***

*To be used in Intel's Larrabee processors*




***LRB is several orders of magnitude larger than Flash***
*(which is considered hard to verify)*

50 message types vs 16 messages
70 Boolean state variables vs 10

# Lessons from our Effort

- A significant milestone
  - To our knowledge no protocol of this size has been verified at this level of automation
  - Proof required just 5 manual lemmas by hand
    - *Dramatic reduction compared to 25 lemmas required for McOP protocol using just CMP method[DCC08]*
  - *CMP+Flows scales very well in terms of protocol size and manual effort required*

- *Demonstrates that powerful invariants, namely those from flows, are available essentially for free*

- Ideas from our work will be useful in other contexts
  - Other message passing systems
  - Shared memory systems, concurrent software verification as well

# Extensions Required

*Notion of Flows had to be generalized*

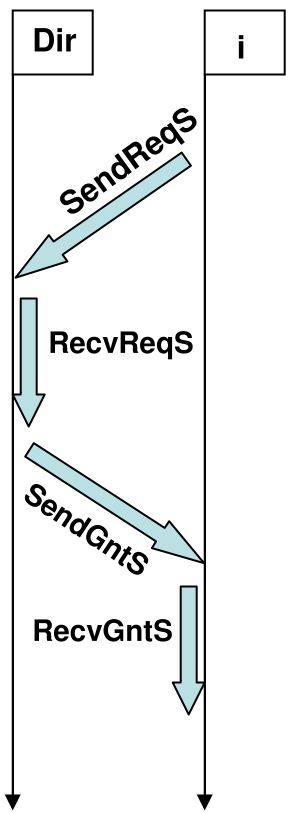
- From simple linear flows to directed acyclic graphs

*Additional invariants from flows*

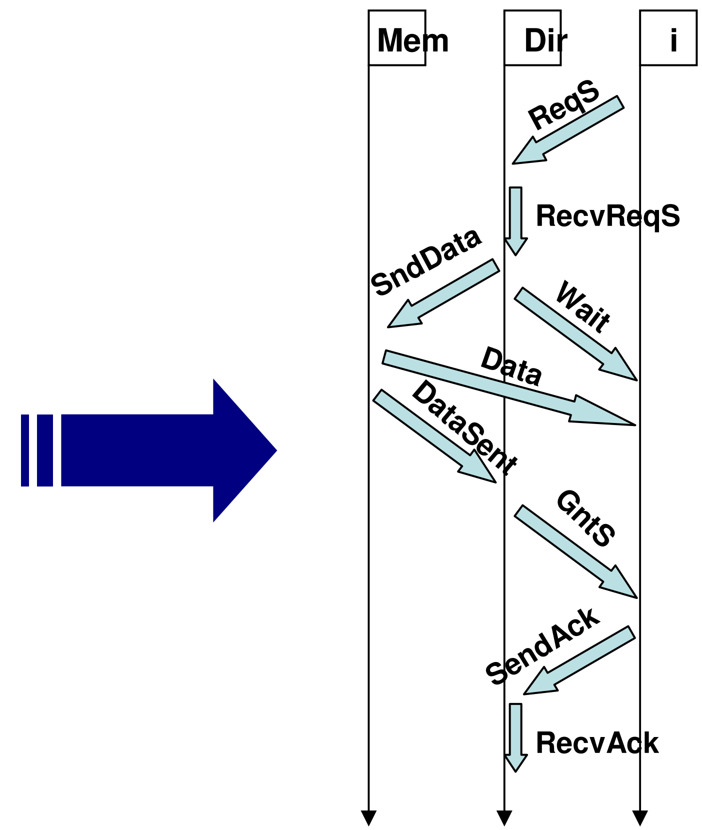
- Conflict between flows

*Criteria to choose which flows to use*

- Using all flows leads to state explosion




***Linear Flows***




***DAG Flows***

# Outline

*Background*

- *CMP method*
- *Flows*

Extensions

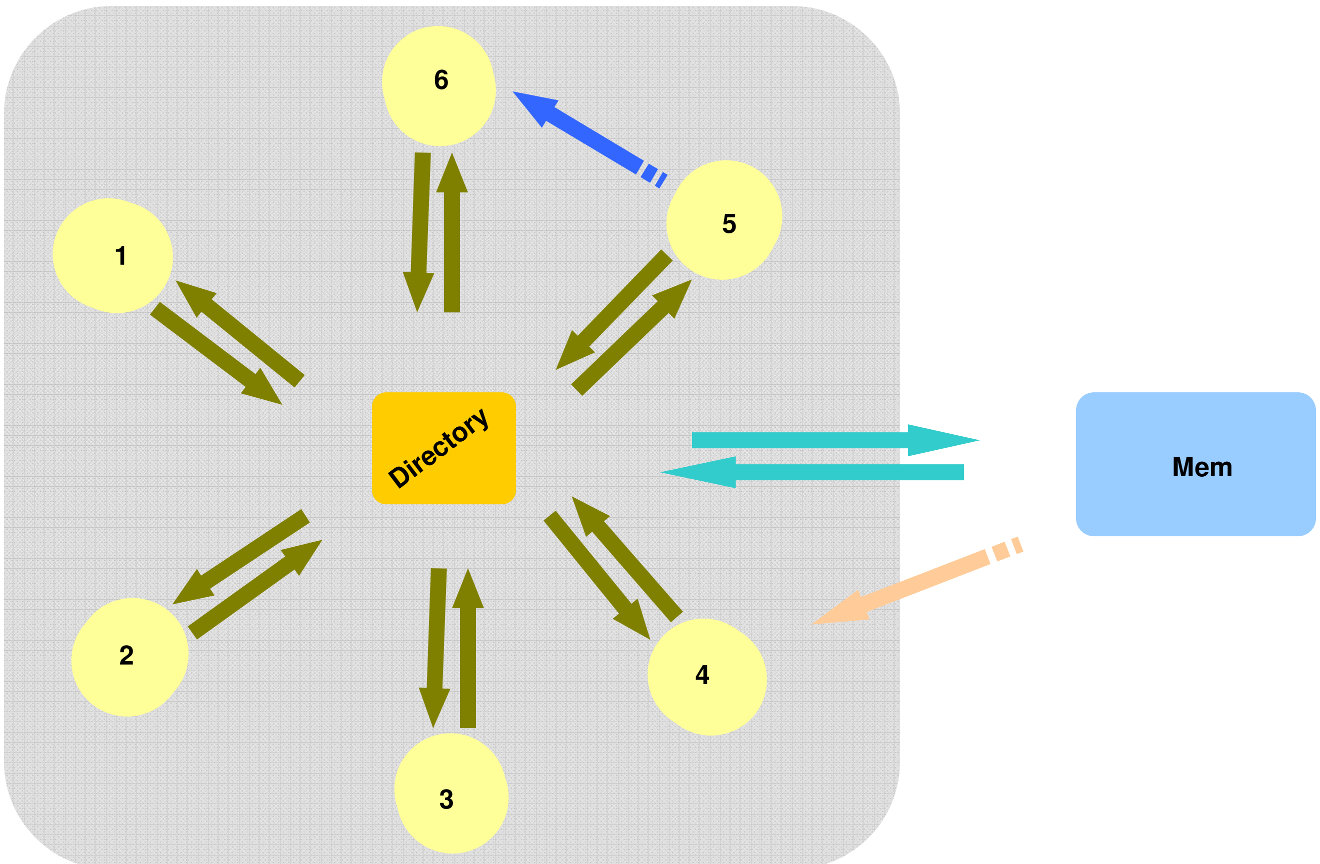
- Linear flows to dags
- New language
- New constraints

LRB verification

- Details
- Lessons

Conclusion

# Logical Model of a Cache Protocol



# CMP+Flows Approach

Consists of two key elements

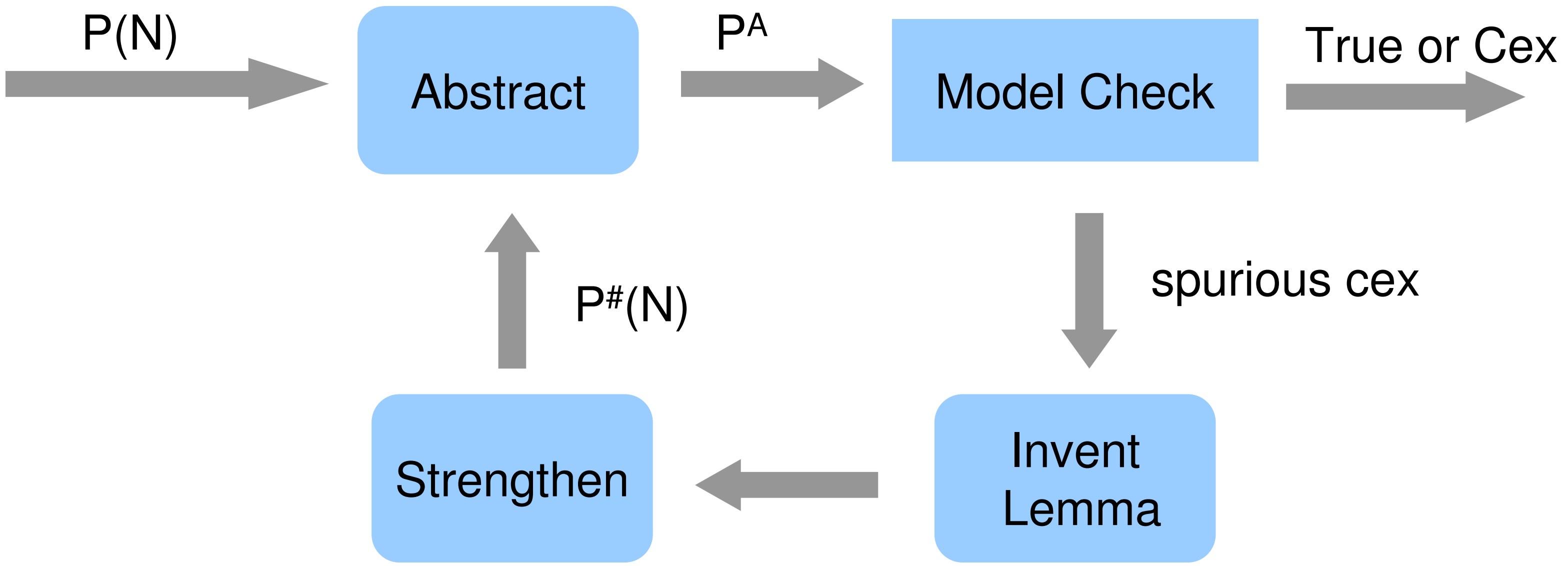
***CMP Method:***

- ***A general framework for verifying systems with replication based on abstraction & compositional reasoning***
- ***We simplified and generalized the method***

***Flow based Invariants:***

- ***A new method for discovering system invariants***
- ***Implicit partial orders on system events yield valuable invariants***

# CMP Method




**Strengthening with L:**

Each rule is of the form:

***rname: guard → action***

Strengthened rule:

***rname: guard & L → action***

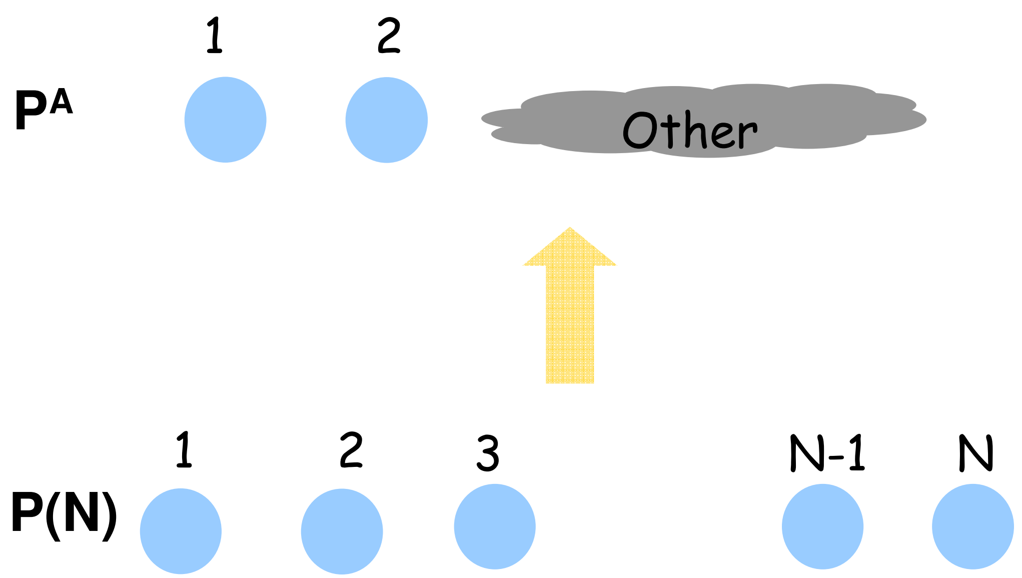
# Abstraction in CMP

*Data Type Reduction*

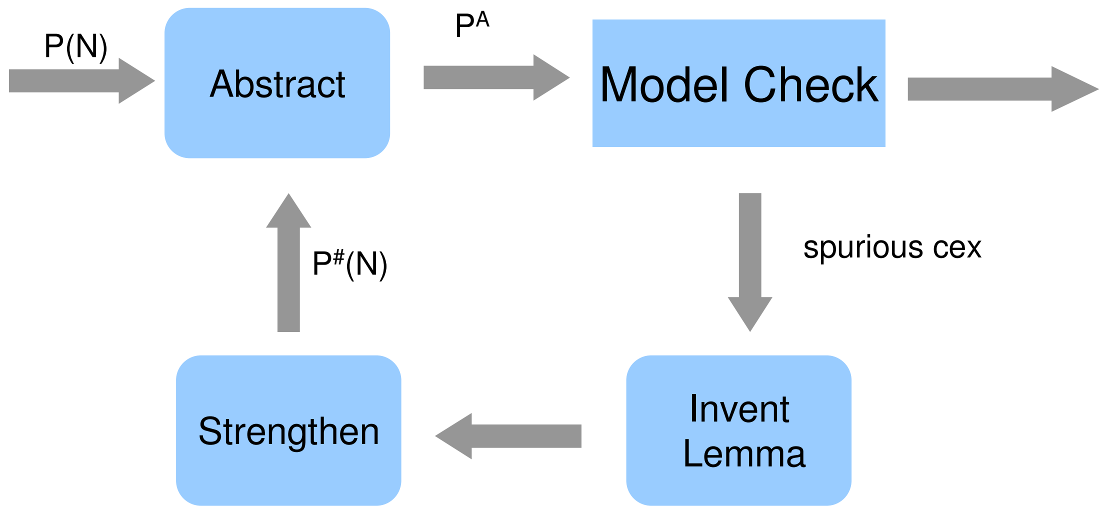
Throws away the state spaces of agents 3..N

Any condition involving them is conservatively over-approximated

Syntactic & fast but leads to very abstract models



# Inventing Lemmas



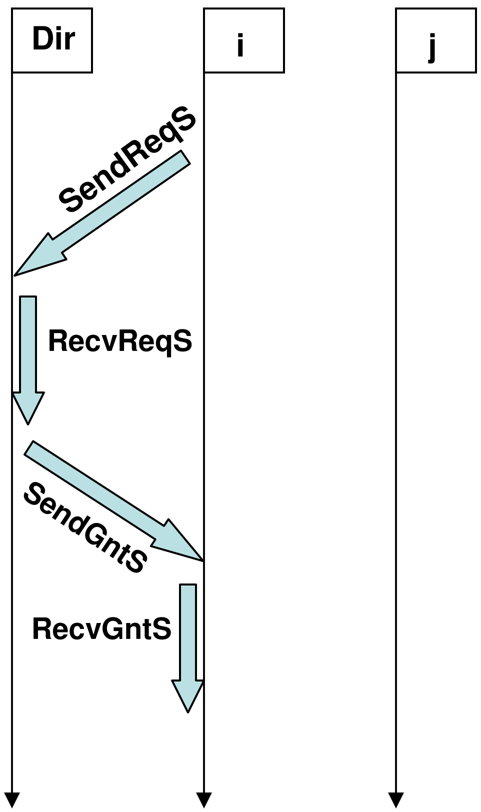

Manual process (by examining spurious cexs)

Time consuming and requires insight

*Drawback of all theorem proving style methods*

***Flows can drastically reduce the "lemma burden"***

# Flows




**Process i intiates a *Request Shared* transaction: Case 1**

Partial orders on system events

- For cache protocols, sending and receiving of messages by various agents are "events"

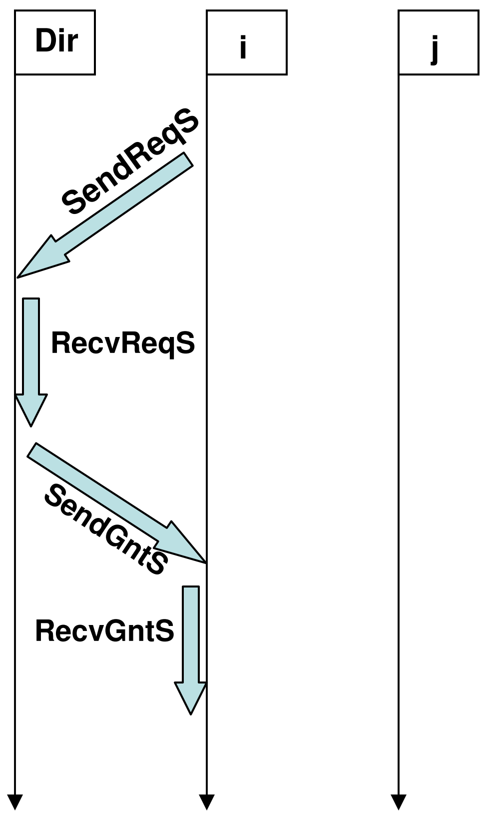
Each event corresponds to a well defined syntactic block of protocol code

- For cache protocols written in Murphi, events are essentially rule names

*For the rest of the talk:*

**Rule names ⇔ Events**

# Constraints from Flows



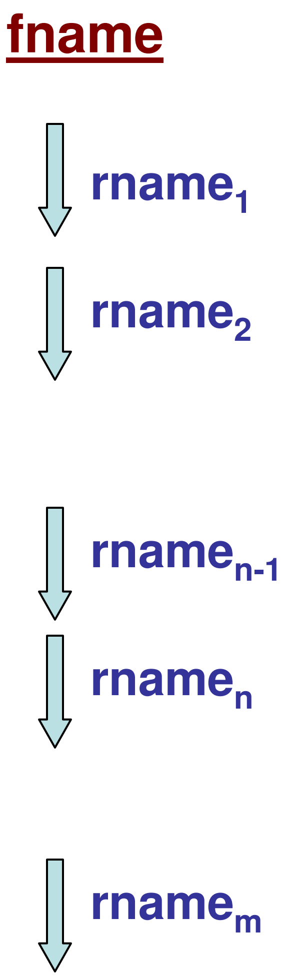

*Precedence between events*:
*For instance, for process i, action* ***RecvReqS(i)*** *must happen before* ***SendGntS(i)***

*Sample invariant:*
If guard for ***SendGntS*** is true then
*history variables must record firing of* ***RecvReqS***

*Flows are used and also validated*

***Wrong/incomplete flows are caught by the method***

# Tracking Flows




A set **Aux(i)** of *auxiliary variables* to track

1) all the flows that a process i is involved in
2) for each such flow the last rule that was fired

Each aux $\in$ **Aux(i)** is initially ( *no_flow, no_rule* )

If process i fires rule $rname_n$ in *fname* update aux = (f,$rname_{n-1}$) to (f,$rname_n$)

If $rname_n$ is the last rule reset the aux variable

# Outline

Background

- CMP method
- Flows

*Extensions*

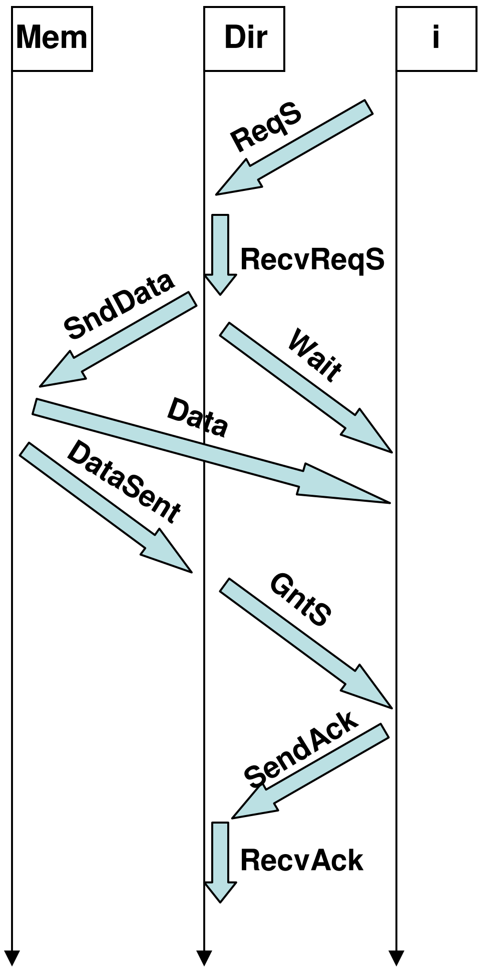
- *Linear flows to dags*
- *New language*
- *New constraints*

LRB verification

- Details
- Lessons

Conclusion

# Typical LRB flow




**A transaction for requesting shared access**

Flows are DAGs in real protocols unlike "academic" protocols

***SendAck*** depends on two other events: ***Data*** and ***GntS***

***RecvReqS*** enbles two other events: ***SndData*** and ***Wait***

Order between all events not specified: For eg., ***GntS*** and ***Data***

Flattening out partial orders leads to an explosion in the number of flows

# Language for new Flows

Each flow is given by:

$\mathbf{fname}$: {$\mathbf{prec}_1$, $\mathbf{prec}_2$,..,$\mathbf{prec}_n$}

Name of the flow

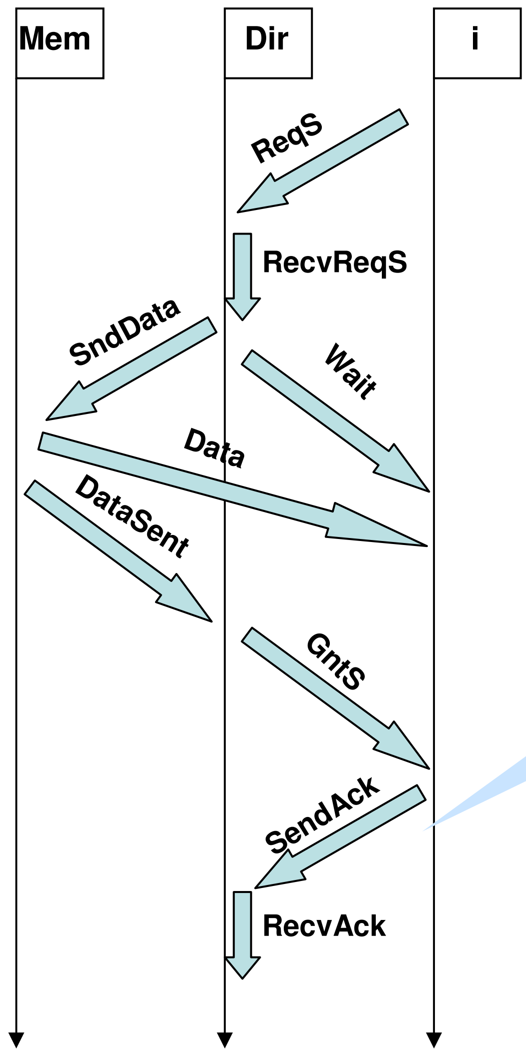
where each $\mathbf{prec}_i$ is an entry of the form

$\mathbf{rname}$: $\mathbf{rname}_1$, .., $\mathbf{rname}_m$

Name of the rule firing

Names of the preceding rules

# Example




***ReqShar: {prec$_1$,.., prec$_9$}***

One of the 'prec's:
**SendAck**: **GntS, Data**

# Conflict sets

- Many flows are mutually exclusive
  - For example, ***ReqShar*** cannot happen when ***ReqExcl*** is happening and vice-versa
    - Because the directory can participate in only one of these at a time
- Further, many flows are such that only a single instance can be alive at any time
  - ***ReqShar***, ***ReqExcl*** for example
- With each flow we also associate a conflict set

# Language for flows

We need event ids to distinguish occurrences of same event in multiple flows.

Each flow is given by:

**fname, conflict_set**: {$\mathbf{prec_1}$, $\mathbf{prec_2}$,..,$\mathbf{prec_n}$}

where each $\mathbf{prec_i}$ is an entry of the form

(**rname, id**): ($\mathbf{rname_1}$,$\mathbf{id_1}$), .., ($\mathbf{rname_m}$,$\mathbf{id}_m$)

Name of the flow and conflict set

Name of the rule firing & id

Names of the preceding rules & ids

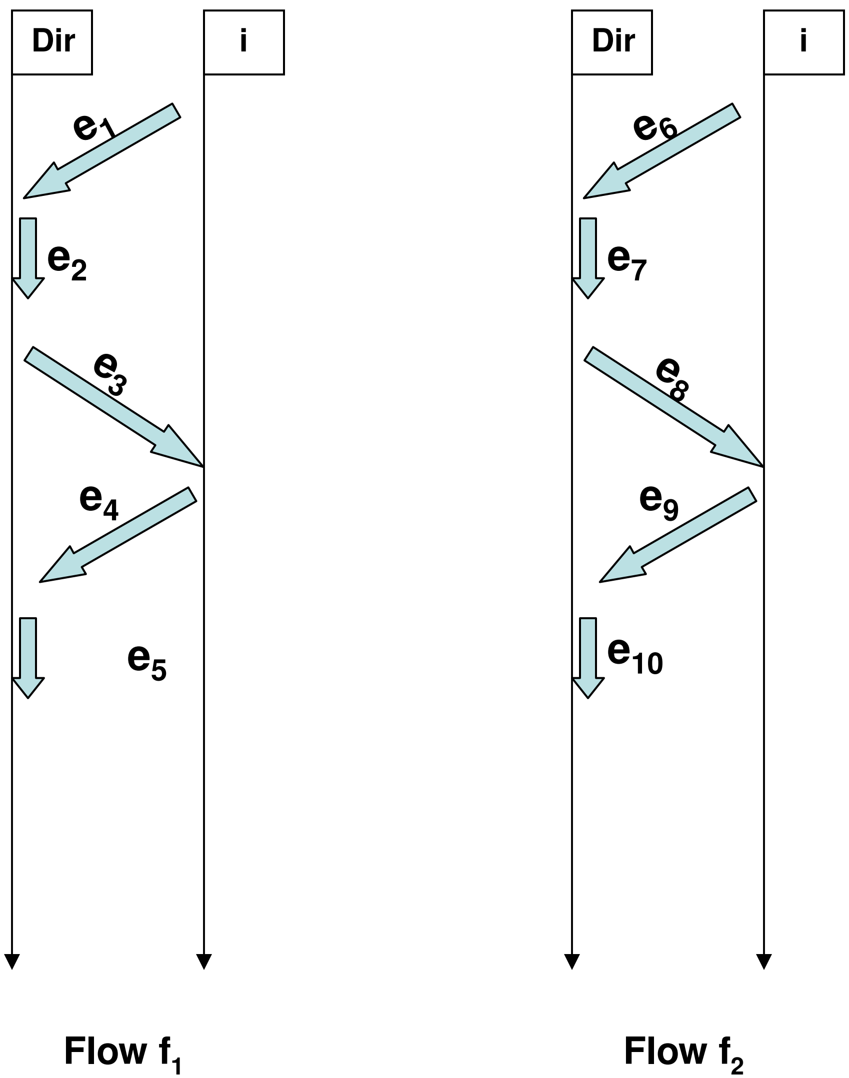
# Invariants from Flows

- Invariants from ***precedence constraints***:
  - Constraints on events within a flow
  - Extension to new language straight-forward

This is new!

- Invariants from ***conflict constraints:***
  - Constraints on events across multiple flows

# Conflict constraints




Suppose $f_1$ and $f_2$ conflict

**Conflict constraint:**

*If f1 is active then f2 cannot become active*

***Equivalently:***

*If there exists an aux variable recording firing of an event from $f_1$ then $e_6$ should not be enabled*

Rest of events in $f_2$ are disabled by the precedence constraints

# Outline

Background

- CMP method
- Flows

Extensions

- Linear flows to dags
- New language
- New constraints

*LRB verification* ←

- *Details*
- *Lessons*

Conclusion

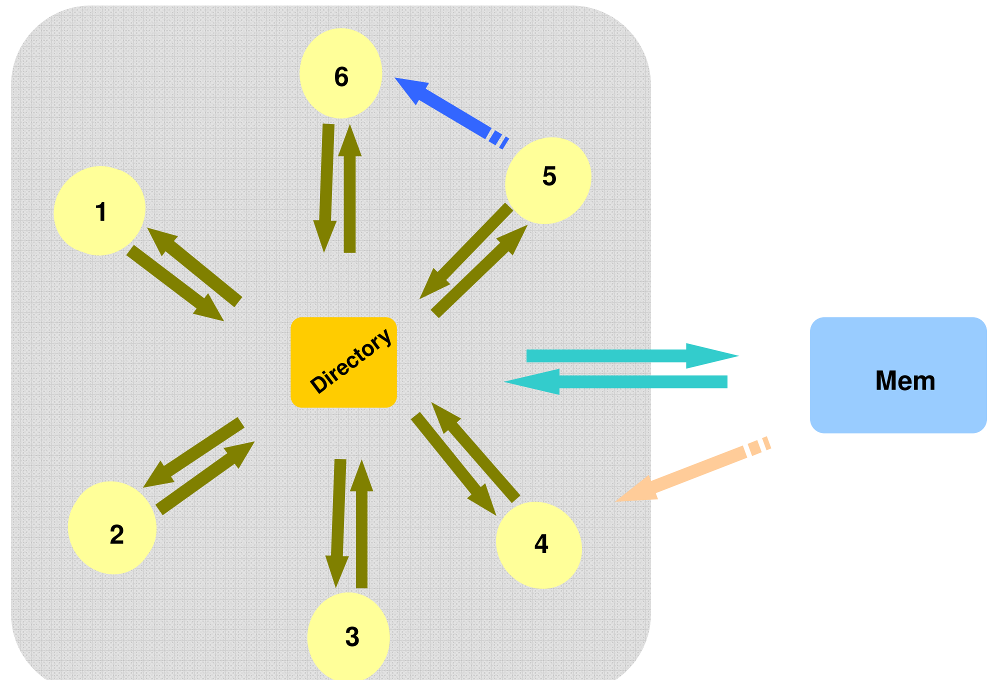
# LRB Cache Model

High level model written Murphi

Constructed semi-automatically from tabular description

Retained all the relevant details

- The various in- and out-buffers and L1 cache states




Retaining all the internal structure made characterizing when an agent has access difficult

# LRB Proof (1)

- ***Property:*** If cache i has exclusive access to an item then no other cache j has access to the same item.

- The rules in Murphi model were very large covering multiple “events”
  - Single rule for “Receive” would cover different types of incoming messages
    - Even though they belonged to completely different transactions

  - We needed to break up the rules into smaller rules to get closer rule-event correspondence
    - Done using simple rewriting procedures

- Model had some quirks
  - Many directory variables referred to using terms that had process ids
    - Though they were essentially constants
  - Leads to an unnecessarily abstract model

# LRB Proof (2)

- Abstraction was carried out using ***Abster***
  - We need to specify how many agents to keep concrete in the abstract model
    - 2 agents for LRB since we were verifying two indexed safety properties

- Flows are also given as an input
  - We used about 15 flows from the design documents
    - Covering transactions for shared and exclusive access
      - Left out flows for write backs and invalidates
  - Flow invariants generated automatically
  - These led to 36 lemmas
    - 25 from precedence constraints and 11 from conflict constraints

# LRB Proof (3)

- 5 manual lemmas on top to complete the proof
  - *Huge reduction compared to the 25 lemmas used for McOP [DCC08]*
- Architects were more impressed with flow validation than with the global properties verified!
- Murphi running time: 5.5 hrs
  - Time taken for whole proof not clear
    - Methodology development and proof went hand in hand

# State explosion from flows

- It does not help to track all the flows that we can get from the design documents!

- ***Only flows that appear in their own conflict sets should be used***

  – The rest lead to blow up in state space of the abstract model
    - Multiple instantiations of a flow can be active at the same time
    - Thus, the “other” agent can saturate the auxiliary variables

  – Unexpected because the concrete model with auxiliary variables does not suffer from the same problem

# Outline

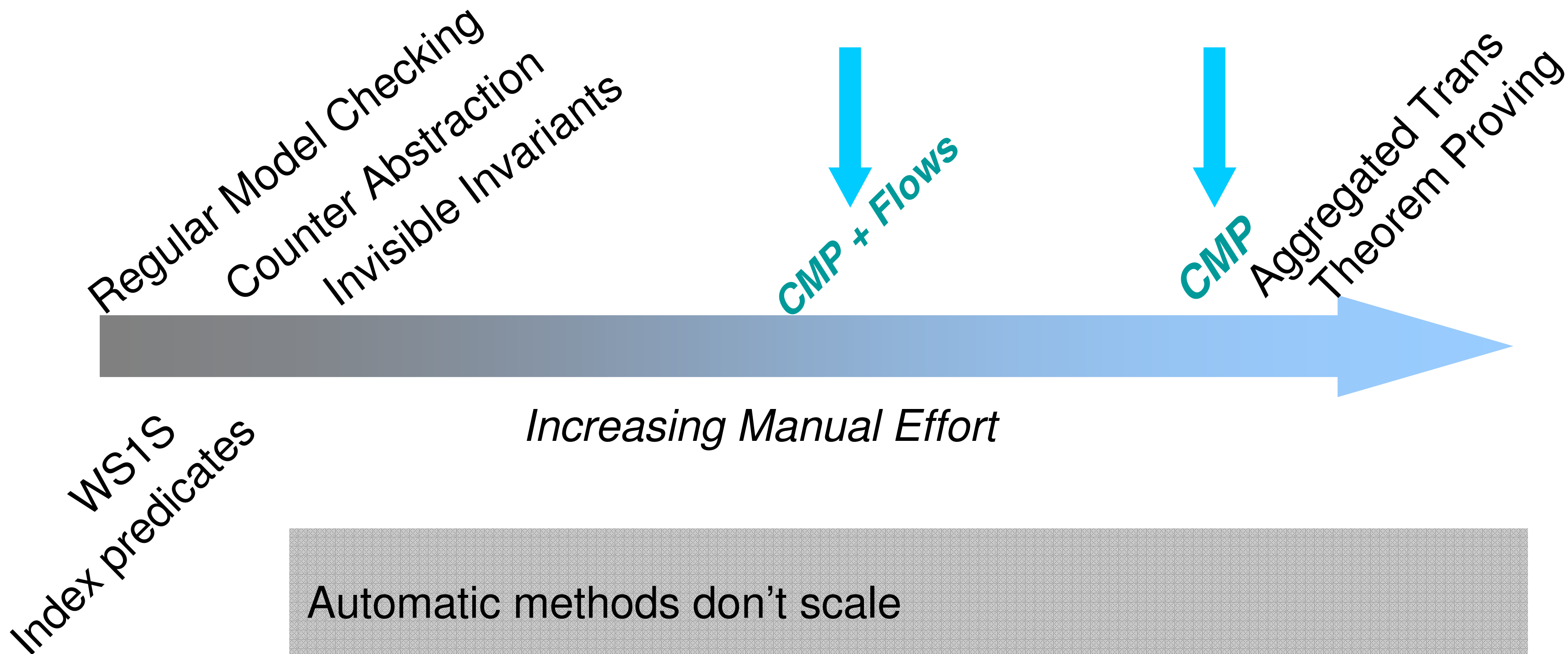
Background
- CMP method
- Flows

Extensions
- Linear flows to dags
- New language
- New constraints

LRB verification
- Details
- Lessons

*Conclusion*

# Existing Methods




Automatic methods don’t scale

Theorem Proving style methods require human guidance but scale

# Conclusion

- *CMP + Flows* method is highly scalable and easy to use
  - Perhaps the only method available for large protocols
- Ideas generally applicable
  - Not limited to cache protocols
  - Flows open up a new avenue to taming verification complexity
    - By providing a way to harness informal high level reasoning in a precise manner

# Future Work

- Extend flows to other kinds of systems
  - Shared memory systems
  - Concurrent software
- Investigate other uses of flows
  - Run time monitoring
  - Refinement checking between high level model and RTL implementation
  - Speeding up model checking