Verification of Temporal Properties for Communicating Datalog Programs

Diego Calvanese
Free University of Bozen-Bolzano, Italy
Umeå University, Sweden

21st International Workshop on Nonmonotonic Reasoning (NMR)
2 Sep. 2023 — Rhodes (Greece)

Our starting point

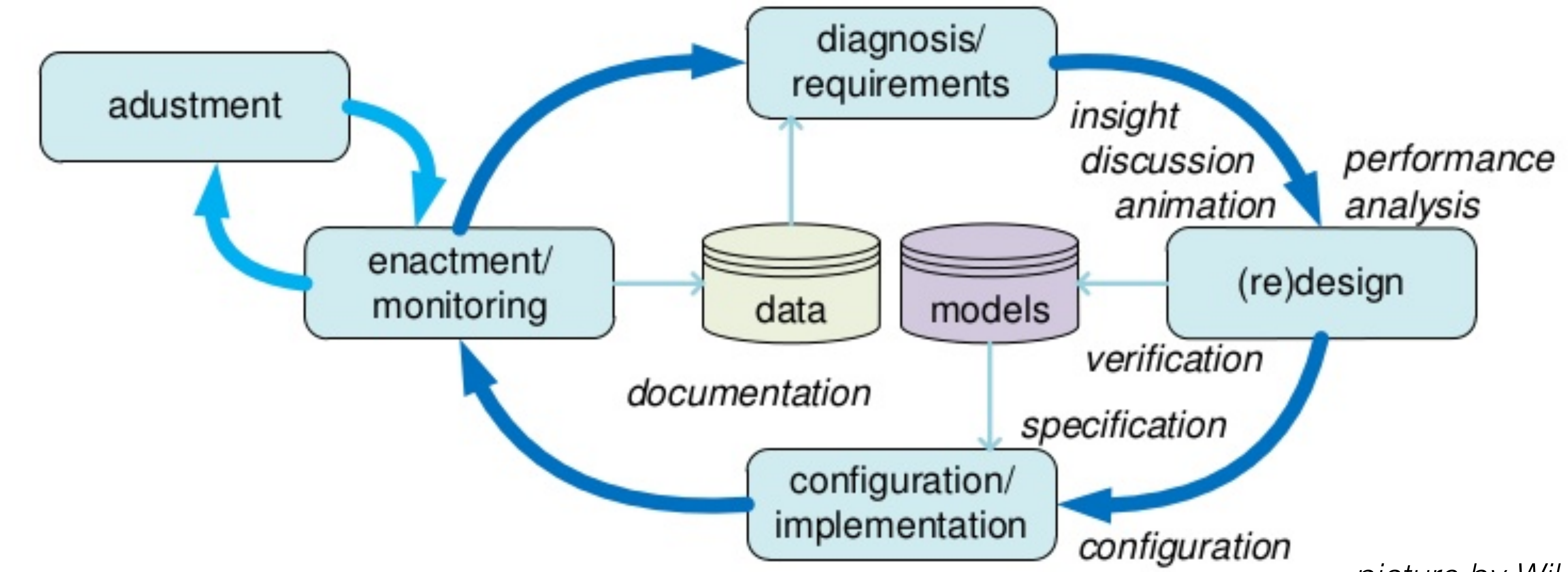
The information assets of an organization consist of:

- data, and
- processes, that determine how data changes and evolves over time.

The underlying dynamic systems come in different forms:

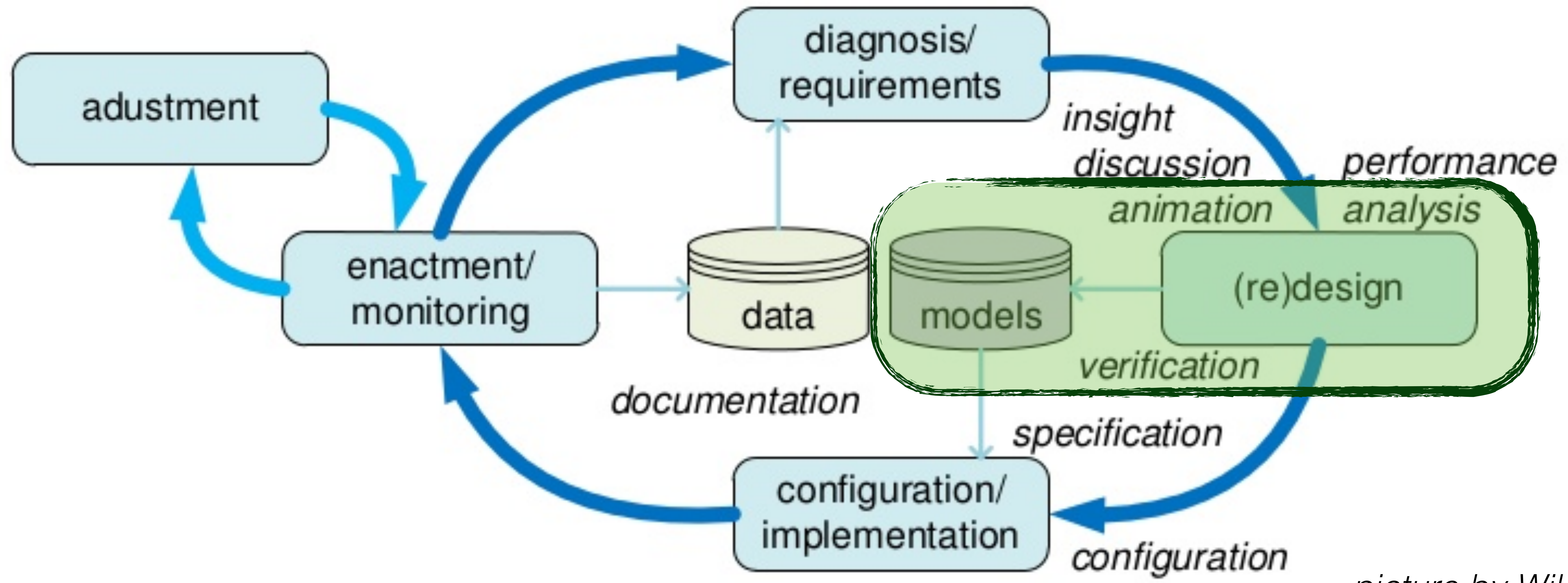
- business processes
- industrial processes
- distributed systems
- multiagent systems

Complex systems lifecycle



picture by Wil van der Aalst

Formal verification



picture by Wil van der Aalst

Automated analysis
of a formal model of the system
against a property of interest,
considering all possible system behaviors

A claim …

Logic-based languages and
knowledge representation and reasoning
can provide powerful techniques and tools to
model and understand complex systems, and
verify in an automated way their behaviour
along their entire lifecycle

… and an appeal

Towards this goal, we should:

- Foster **cross-fertilization with related fields** such as database theory, formal methods, business process management, information systems
- Understand the **decidability / undecidability frontier**, classifying the **sources of complexity**, so as to attack them when developing concrete tools
- Use requirements coming from **practice** to guide and validate foundational results

Practice

Practice

BPMN

OWL ORM

UML

YAWL

EPC

SQL FCL

CMMN

GSM Dedalus

Declare

DAML-S JSON

WebdamLog AUML

ACM JADE

BPEL Bloom

ER SBVR DMN

+ methodologies

How do we reconcile this situation?

How do we reconcile this situation?

Strong theory

How do we reconcile this situation?

Strong theory

A formally grounded design methodology

1. Develop **formal models** of the systems we deal with
2. Show that they can capture **requirements coming from practice**
3. Understand sources of **undecidability** and **complexity**
4. Find **robust conditions** for decidability/tractability
5. Understand how they **apply in practice**
6. Implement proof-of-concept **prototypes** for verification

A formally grounded design methodology

1. Develop **formal models** of the systems we deal with
2. Show that they can capture **requirements coming from practice**
3. Understand sources of **undecidability** and **complexity**
4. Find **robust conditions** for decidability/tractability
5. Understand how they **apply in practice**
6. Implement proof-of-concept **prototypes** for verification

Data vs. processes

To understand how complex systems operate, we need to take into account both **data** and **processes**, and how they **interact with each other**

“Sometimes I just feel like processing some data, but I have no data to process—other times I have the data, but I have nothing to process it with.”

A dichotomy

- We need to overcome the dichotomy between data and processes
- Warning: the **dichotomy is deeply rooted** in industrial practice, and in the adopted methodologies and tooling
  - *BPM professionals*: think that **data are subsidiary to processes**, and neglect the importance of data quality
  - *Master data managers*: claim that **data are the main driver** for the company's existence, and they only focus on data quality

Overcoming the data-process dichotomy

Strong need for

- formalisms supporting the **integrated modeling of processes and data**
- design methodologies based on such formalisms
- systems and tools that implement formalisms and methodologies

In line with our methodology, we follow a **foundational approach**

Dichotomy addressed in different communities

Data Management

Knowledge Representation

Business Processes

Dichotomy addressed in different communities

Data Management

- Dynamic Relational Model [Vianu 1980s] — constraints between DB states
- Active DBs [Vianu, Abiteboul 1980s, 1990s] — ECA rules over DBs
- Temporal DBs [Snodgrass 1980s, 1990s] — contraint-based [Kabanza & al. 1990s], temporal deductive DBs [Chomicki & Imielinski 1980s] — Queried via timestamped FO or FO-LTL
- Relational transducers [Abiteboul & al., late 1990s] and ASM [Spielmann 2000s]
- Active XML (AXML) [Abiteboul 2000s]
- Data-driven Web Systems [Deutsch & al. 2000s]
- Data-centric Dynamic Systems (DCDSs) [Bagheri-Hariri & al. 2010s]

Knowledge Representation

Business Processes

Dichotomy addressed in different communities

Data Management

Knowledge Representation

- Reasoning about actions and Sitcalc [McCarthy 1960s; Reiter 1990s]
- Temporal extensions of DLs [Wolter & Zakharyaschev 1990s; Artale & al. 2000s]
- Temporal logics (LTL) over DLs [Baader & Lutz 2010s]
- Combining DLs and action formalisms [Milicic 2000s]
- Semantic Web Services — OWL-S, WSMO, … [mid 2000s]
- Adopting Levesque’s functional approach and boundedness [C., De Giacomo, Lomuscio, Patrizi, Montali, et al. 2010s]

Business Processes

Dichotomy addressed in different communities

Data Management

Knowledge Representation

Business Processes

- Workflow formalisms and systems
- Artifact-centric approach [2000s at IBM, 2010s] — with data representation and lifecycle components
- variants of Petri Nets with data — colored PNs, PNs with names, DB-nets [Montali & Rivkin, late 2010s]
- “triple crown” of process improvement: BPMN + CMMN + DMN — somewhat loose coupling, addressing practical requirements

Formal verification — The propositional case

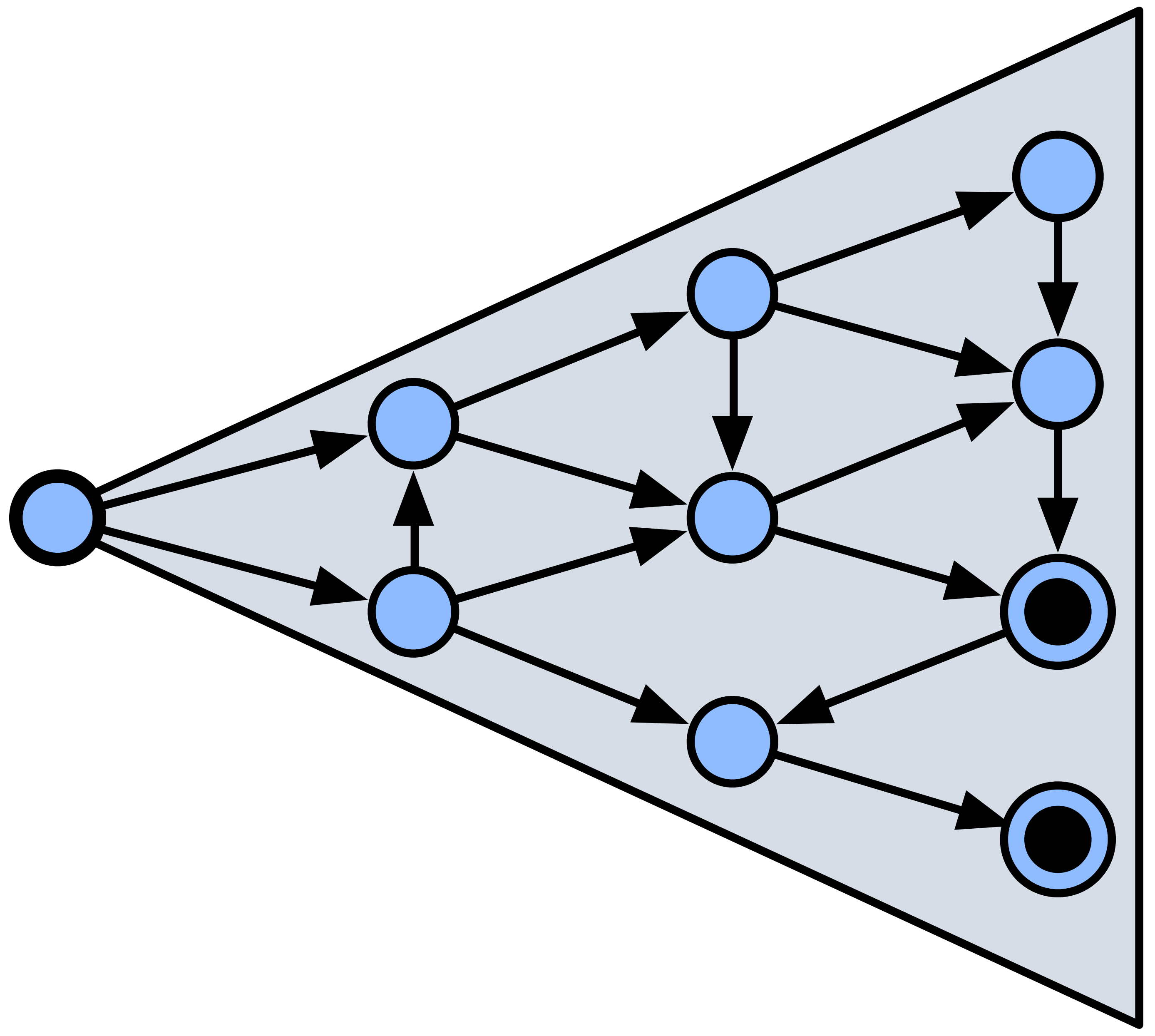
Process control-flow
Agent behaviors / protocols

(Un)desired property

Formal verification — The propositional case

Process control-flow
Agent behaviors / protocols

Finite-state
transition
system



Φ

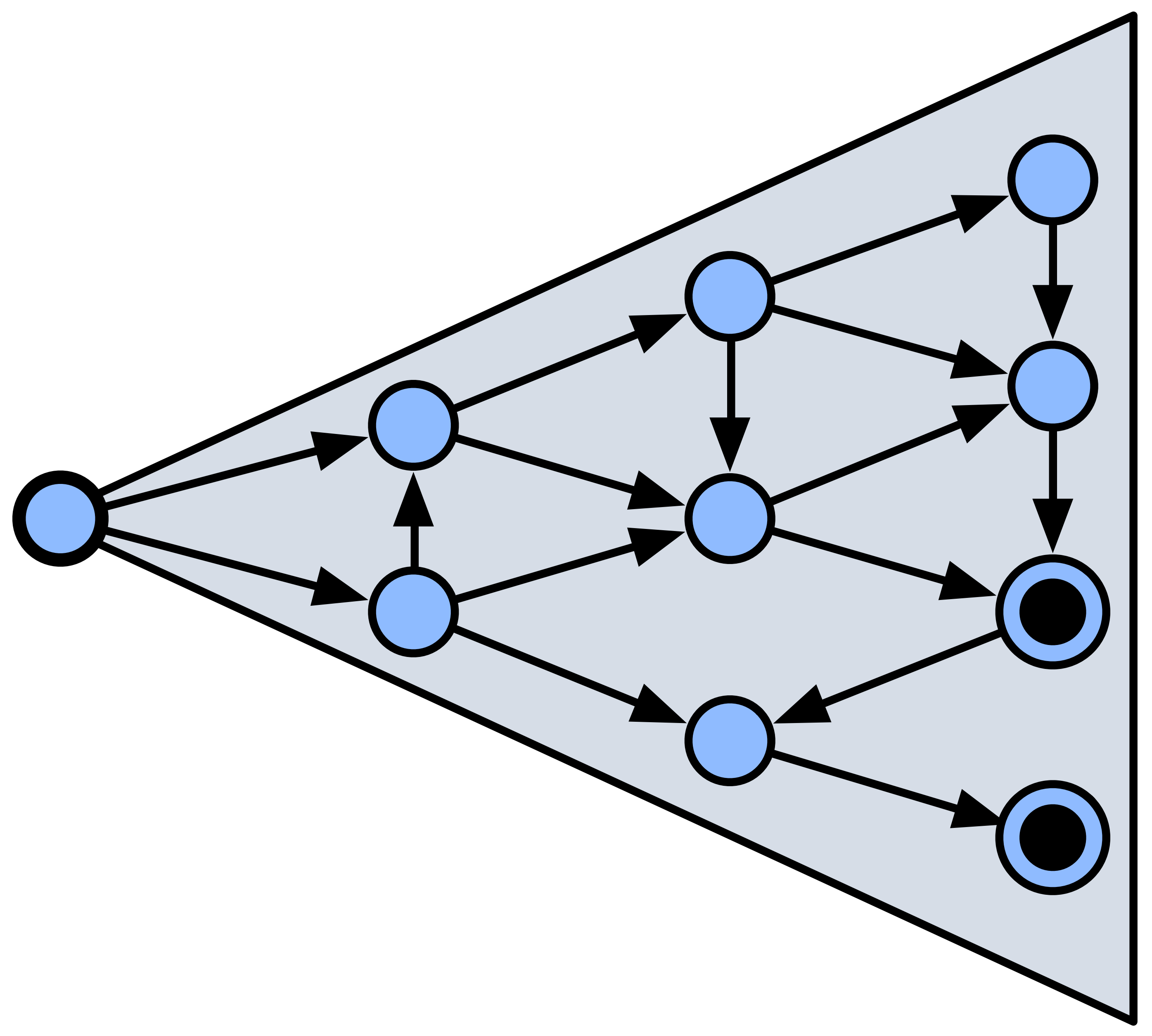
Propositional
temporal formula

(Un)desired property

Formal verification — The propositional case

Process control-flow
Agent behaviors / protocols

Finite-state transition system



$\models \Phi$

Verification via model checking
2007 Turing award:
Clarke, Emerson, Sifakis

Propositional temporal formula

(Un)desired property

Formal verification — The data-aware case

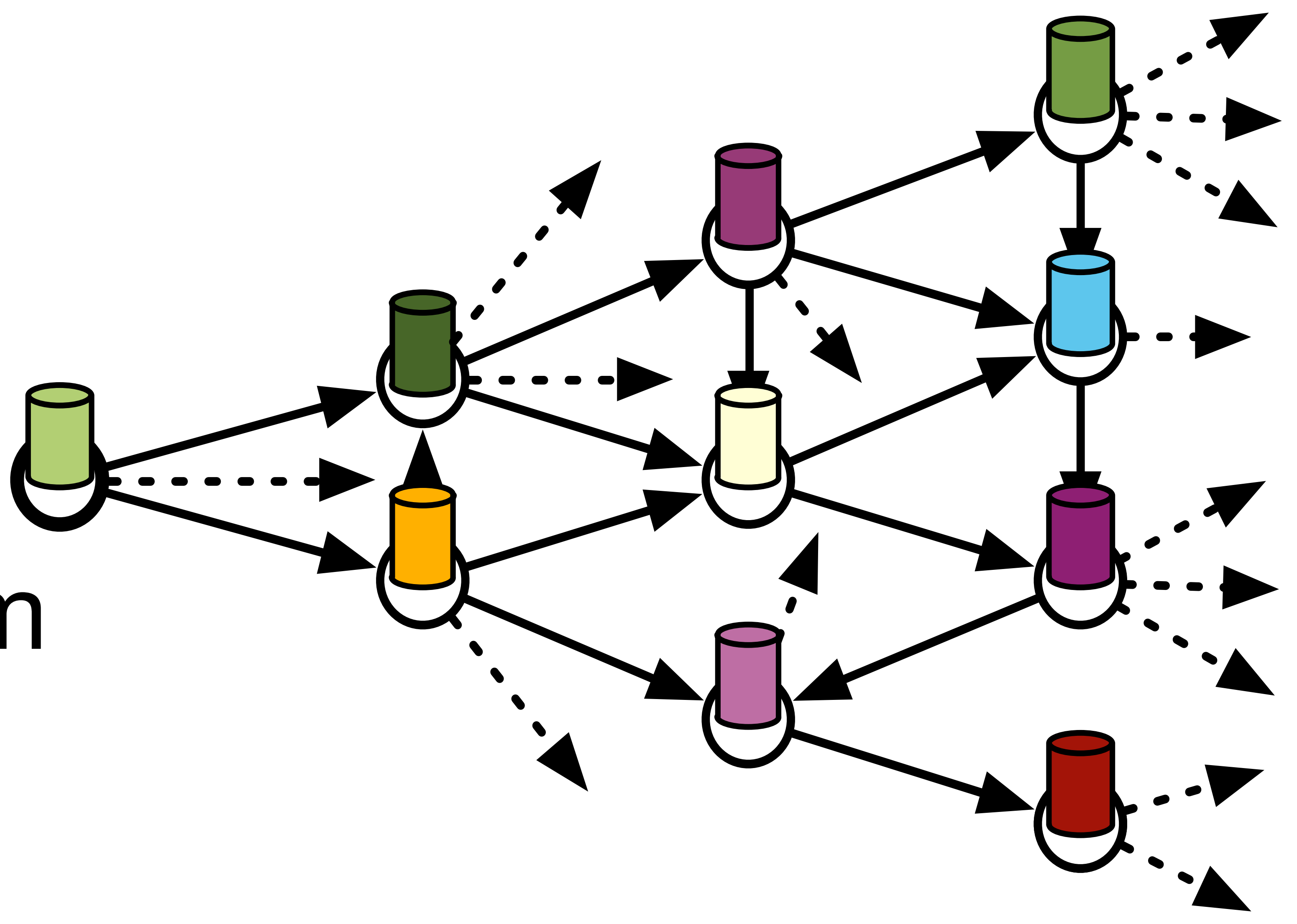
Process+Data
Data-aware agent behaviors / protocols

(Un)desired property

Formal verification — The data-aware case

Process+Data
Data-aware agent behaviors / protocols

Infinite-state, relational transition system



Φ

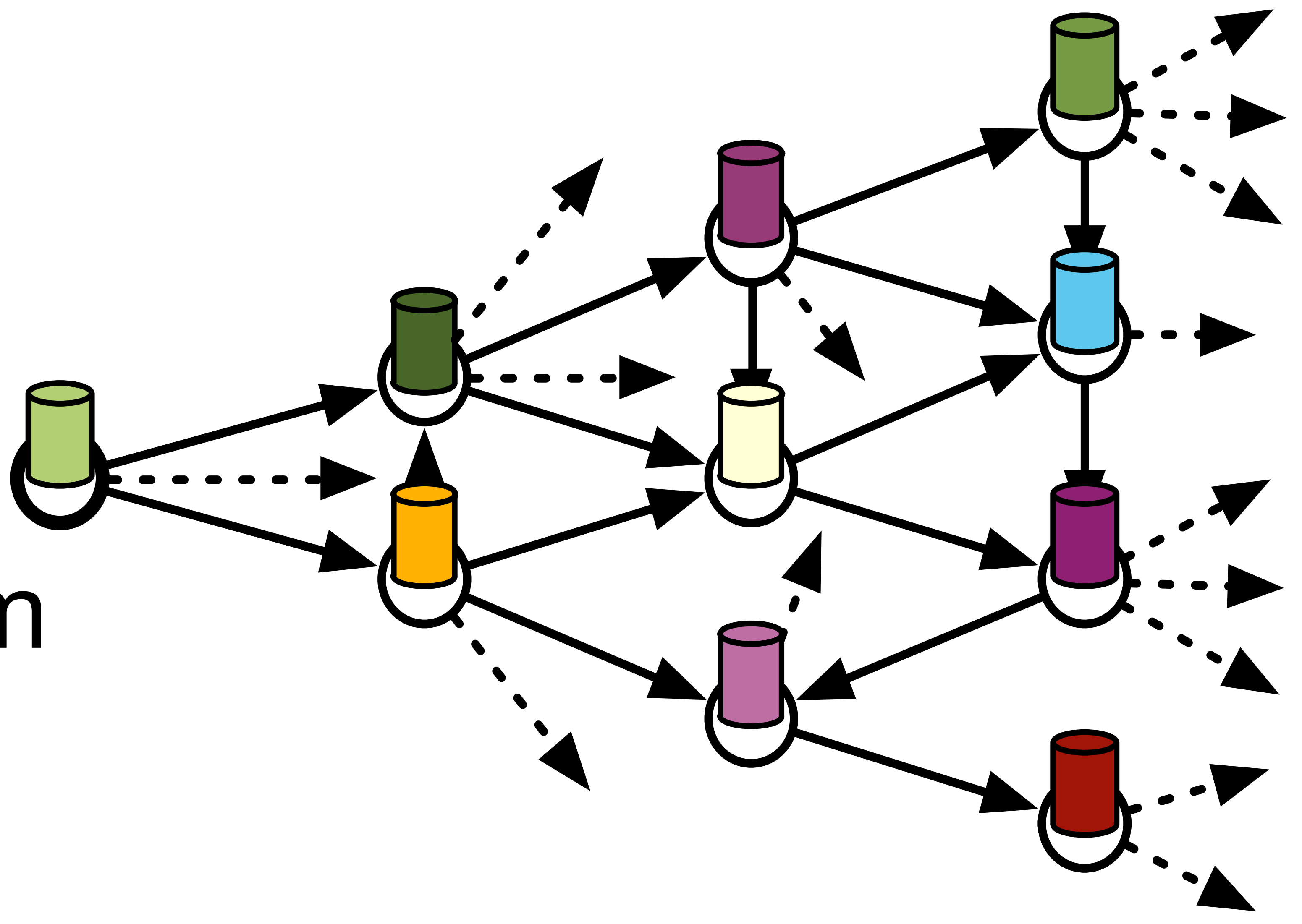
First-order temporal formula

(Un)desired property

Formal verification — The data-aware case

Process+Data
Data-aware agent behaviors / protocols

Infinite-state, relational transition system



$\models \Phi$

First-order temporal formula

(Un)desired property

Why FO temporal logic

- To inspect data → **FO queries**
- To capture system dynamics → **temporal modalities**
- To track evolution of objects → FO **quantification across** states

Example: It is always the case that every order is eventually either cancelled or paid and then delivered

Problem dimensions

Data component	Relational DB	Description logic KB	OBDA system	Inconsistency tolerant KB	…
Process component	Condition-action rules	ECA-like rules	Golog program	…	
Task modeling	Conditional effects	Add/delete assertions	Logic programs	…	
External inputs	None	External services	Input DB	Fixed input	…
Network topology	Single orchestrator	Full mesh	Connected, fixed graph	Dynamic graph	…
Interaction mechanism	None	Synchronous	Asynchronous and ordered	…	

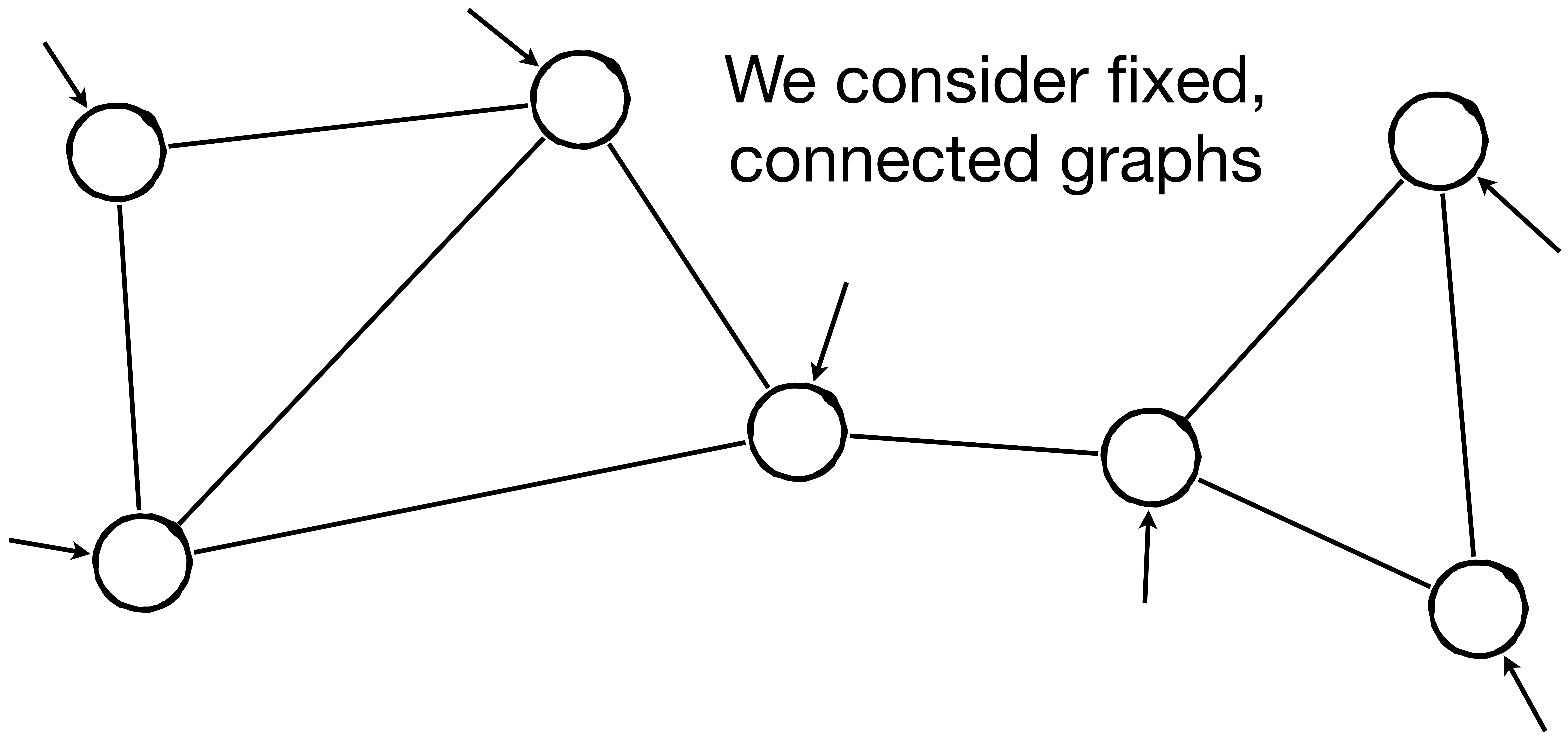
We pick one specific model

Declarative distributed computing

Distributed, data-centric computations
with extensions of Datalog

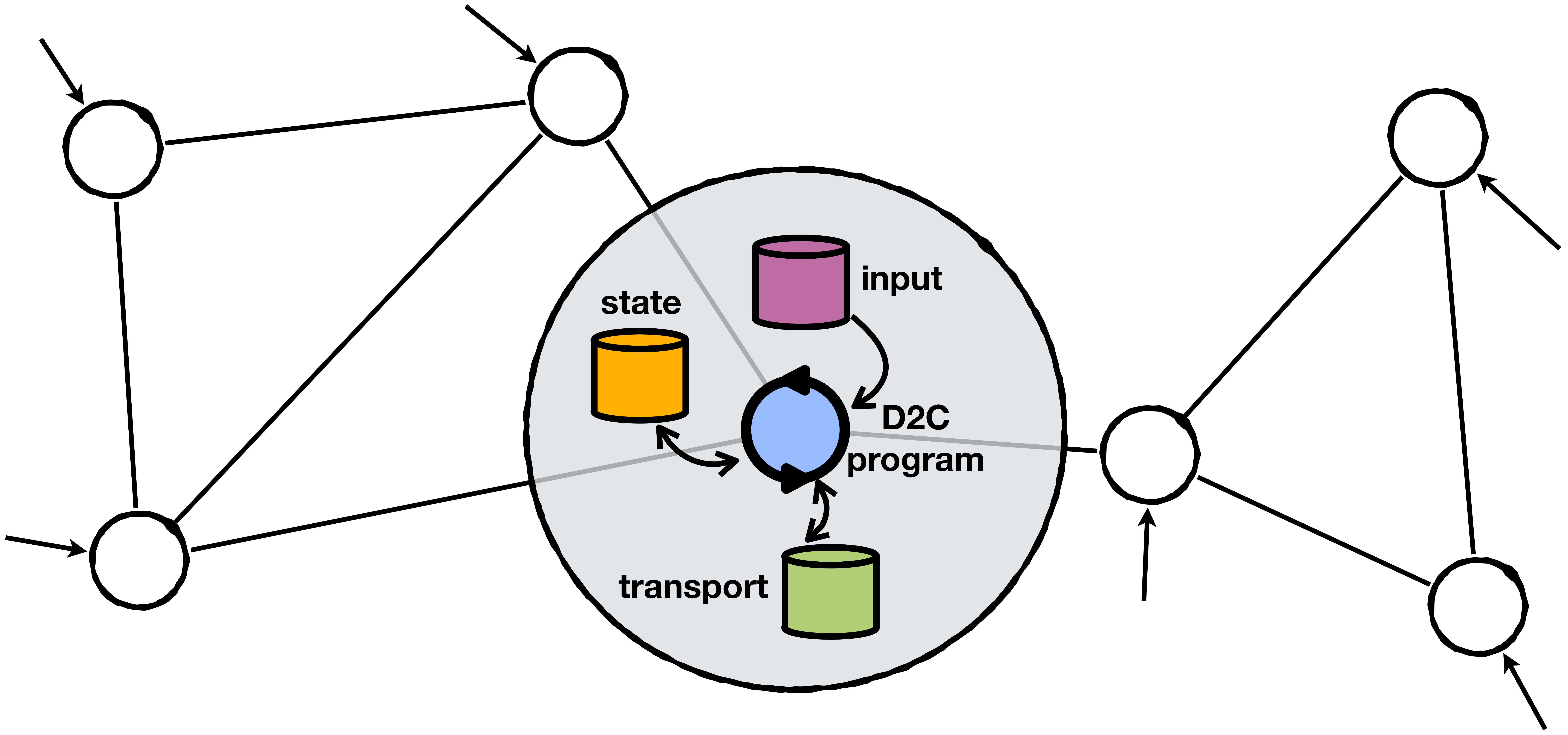
- Many applications: distributed query processing, distributed business processes, web data management, routing algorithms, software-defined networking, …
- Contributed to the renaissance of Datalog [Loo & al., 2009; Hellerstein, 2010]
- Compares well with standard approaches [Loo & al., 2005]

Communicating Datalog Programs (CDPs)



We consider fixed, connected graphs

Communicating Datalog Programs (CDPs)



D2C program running in a node

- Datalog programs extended with
  - **time**: ***prev* construct** to refer to the **previous state**
  - **location**: **@ construct** to refer the **sender / receiver nodes**
  - **non-determinism**: *choice* construct [Saccà & Zaniolo, 1990]
- Stable model semantics
- Each node has initial knowledge about its neighbors and its identity, and starts with a given state DB that is the same for all nodes
- **Input** relations are read-only, and **may inject fresh data from an infinite data domain** (strings, pure names, …)

Reactive behaviour of nodes

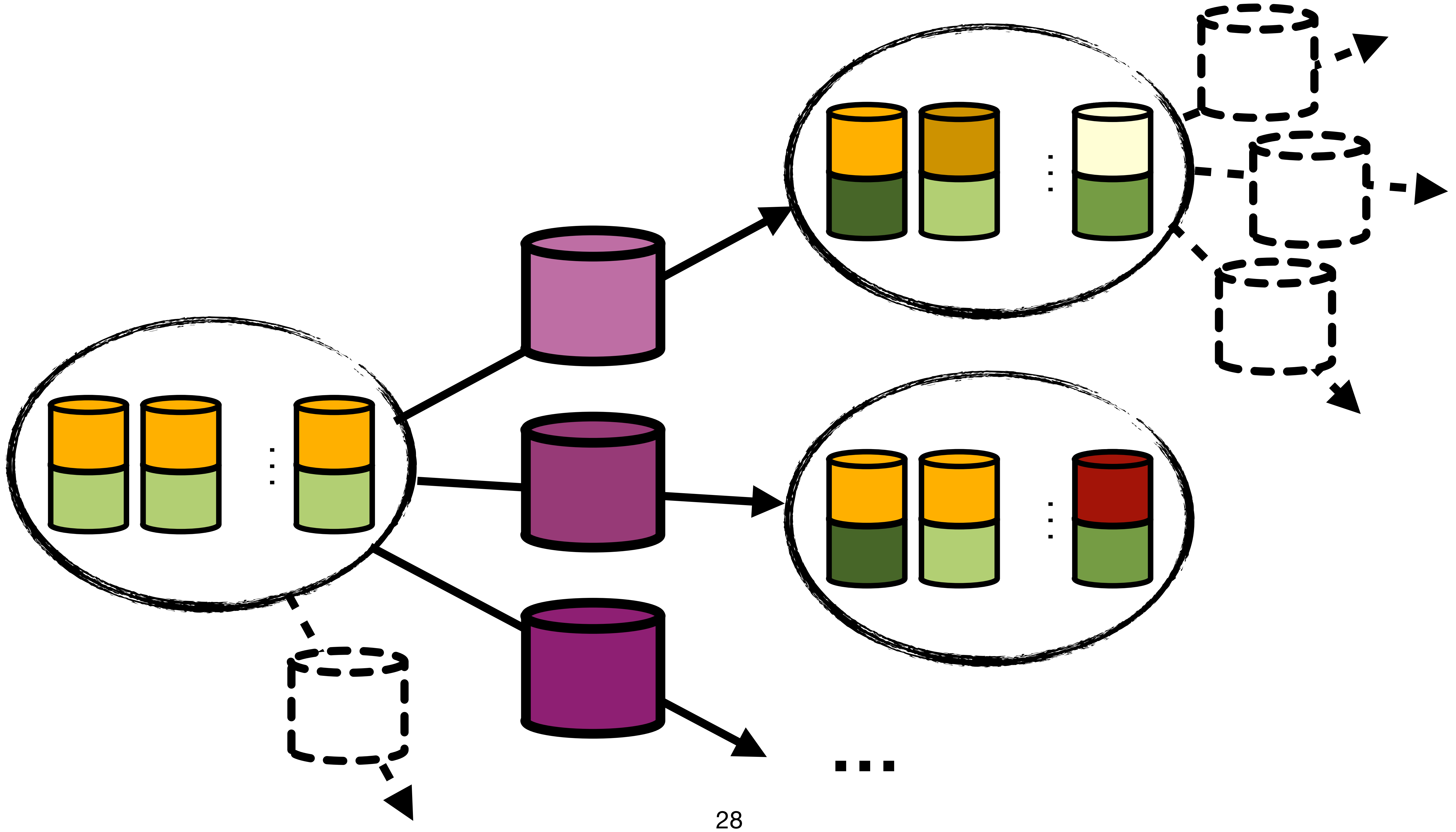
Whenever a node receives (a set of) incoming messages, it performs **a transition**:

1. *Received messages* form the **current transport DB**
2. The current **input DB** is incorporated
3. *Stable models* of the program are computed
4. The node *nondeterministically* evolves by *updating* its **state** and **transport** DBs with the content of one of the stable models
5. The messages contained in the **newly computed transport DB** *are sent* to the destination nodes

Execution semantics

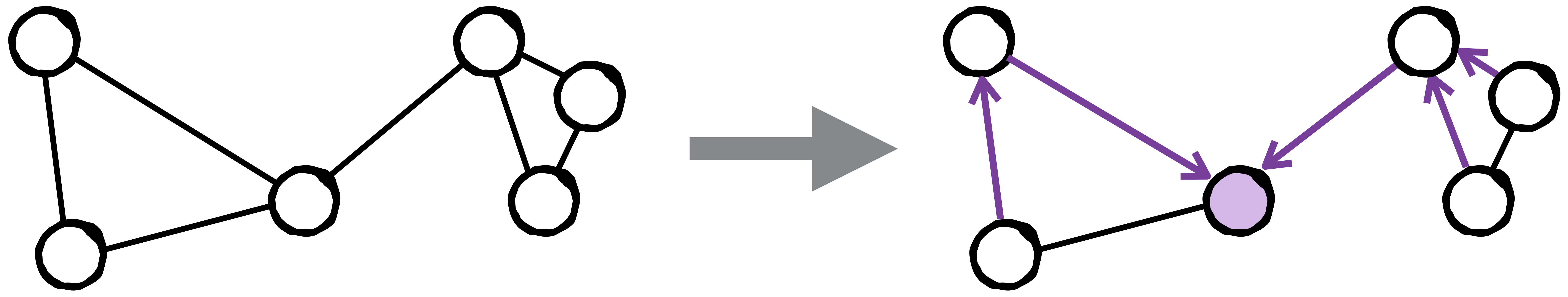
Relational transition systems with node-indexed databases

Successors are constructed considering **all possible input DBs** and **all possible internal choices of nodes**



Example

Construction of a rooted spanning tree of the network



- State schema: keeps neighbors and parent
- Transport schema: asks neighbor to become a child

Example

- When multiple neighbors request to join, pick one as a parent if you don't already have one:

```
parent(P) if choice(X,P), join@X,
             prev not parent(_).
```

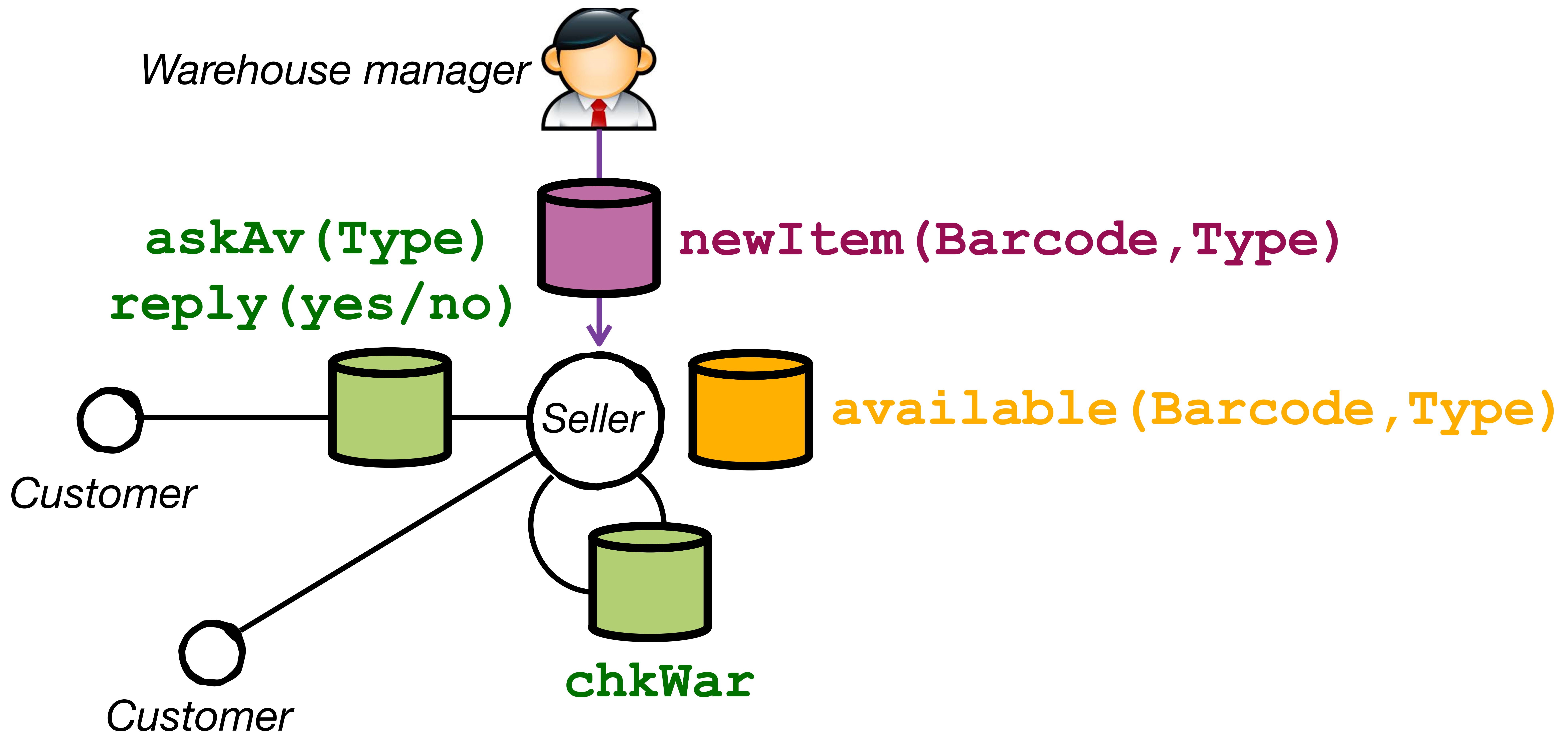
- If you have just joined the tree, flood the join request to neighbors (the parent will ignore it):

```
    join@N if parent(_), neighbor(N),
             prev not parent(_).
```

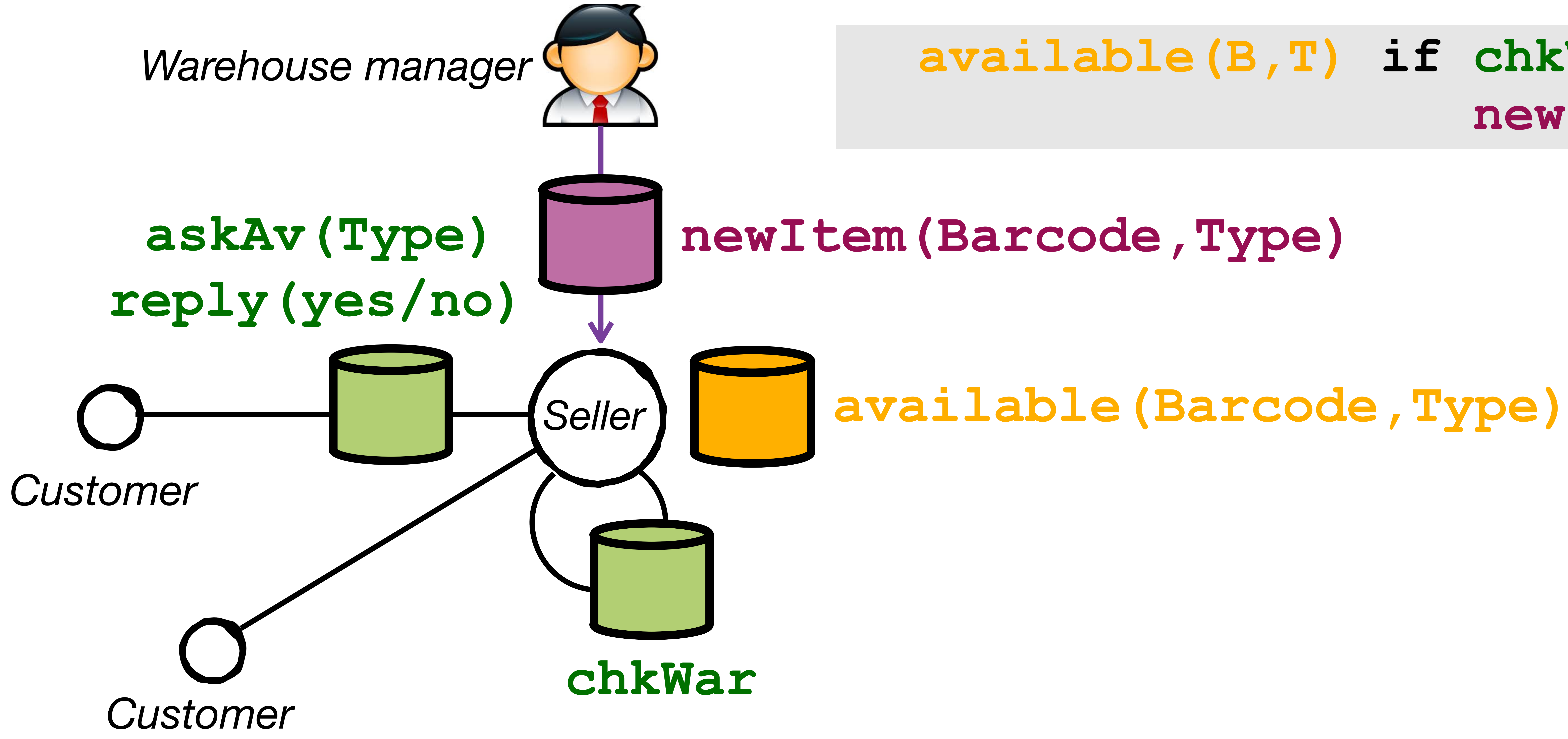
- Parent information is kept:

```
parent(P) if prev parent(P).
```

Another example

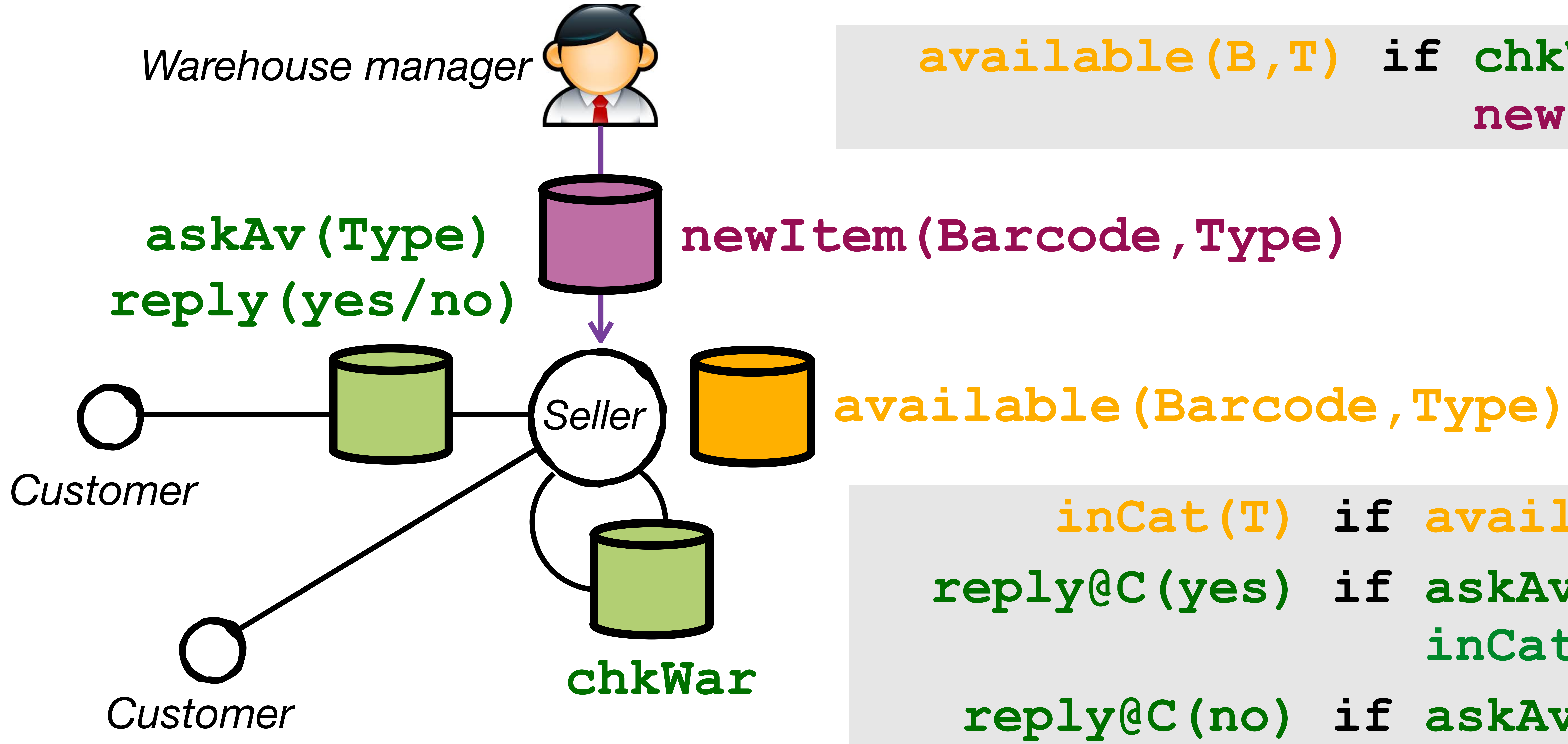


Another example



```
    available(B,T) if chkWare@self,
                      newItem(B,T).
```

Another example



```
available(B,T) if chkWare@self,
                  newItem(B,T).
```

```
      inCat(T) if available(_,T).
 reply@C(yes) if askAv@C(T),
                  inCat(T).
  reply@C(no) if askAv@C(T),
                  not inCat(T).
```

Pure declarative semantics

- Runs of closed CDPs can be simulated using standard ASP solvers
- D2C programs are compiled into Datalog by:
  - priming relations for simulating **prev**
  - transforming @ into an additional predicate argument
  - transforming transport predicates into send/receive predicates
- Additional rules for causality via vector clocks
- Additional rules for the semantics of the communication model

Which properties to verify

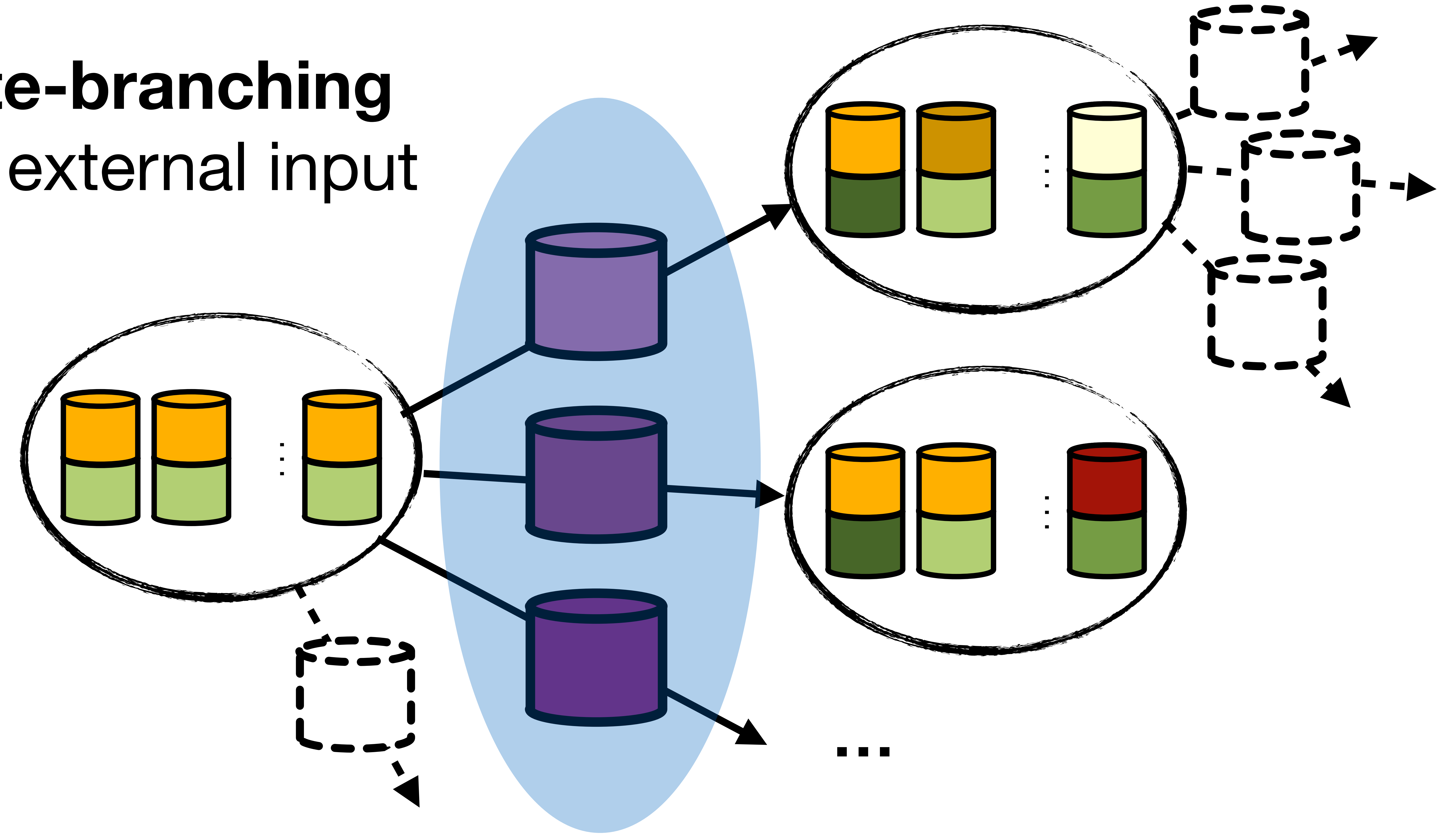
- Domain-specific properties: **CTL-FO** or LTL-FO with active domain quantification
  - Maintain: $\mathbf{AG}(\forall n, p \,.\, Parent@n(p) \rightarrow \mathbf{AG} Parent@n(p))$
  - Broadcast: $\mathbf{AG}(\forall x \,.\, (\exists n \,.\, R@n(\vec{x})) \rightarrow \mathbf{AF} \forall n' \,.\, R@n'(\vec{x}))$

- Generic properties: **convergence**

  Check whether the system
  always / sometimes reaches quiescence
  with some / all nodes in a non-faulty state

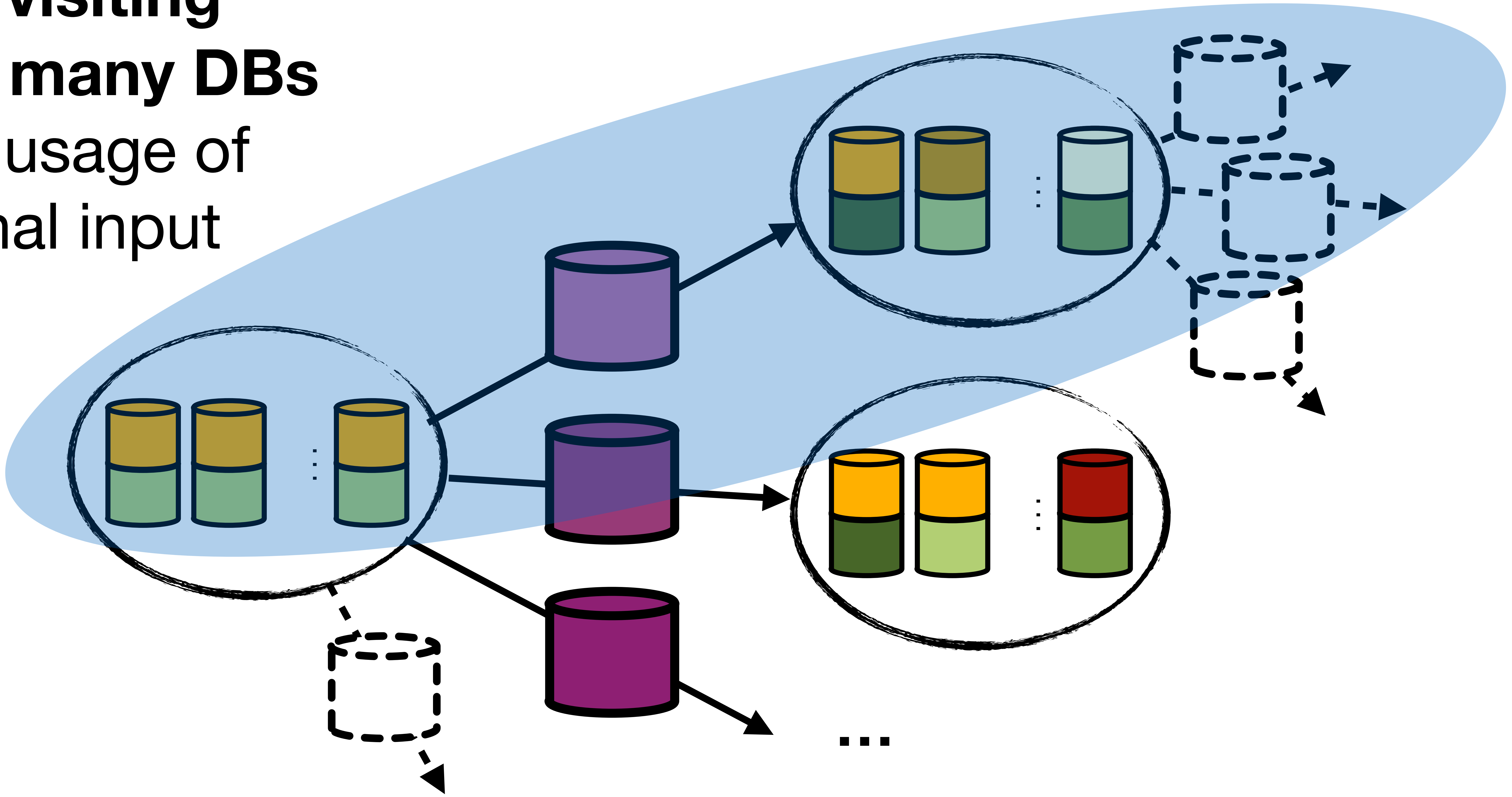
Sources of infinity

Infinite-branching
due to external input



Sources of infinity

Runs visiting infinitely many DBs due to usage of external input



Problem space is still large

- **Input-policy**:
    - closed: no inputs
    - autonomous: input DB is given at the beginning and then not changed
    - interactive: input DB changes at each computation step
- **Channel behaviour**: sets / multisets / queues, lossy / non lossy
- Data **boundedness**:

    a DB is *b*-bounded if, in each reachable configuration, it contains at most *b* objects. It is bounded if it is b-bounded for some b.
- **Message expressiveness**: propositional only / unary only / arbitrary
- Prev-awareness: allow / disallow the use of *prev* on the input

An easy case: closed CDPs with set-channels

Closed CDPs with set-channels

There is

- no injection of data from the external world
- no component of the system that can grow indefinitely

$\rightarrow$ FO becomes syntactic sugar, and we can rely on traditional model checking

Closed CDPs with set-channels

There is

- no injection of data from the external world
- no component of the system that can grow indefinitely

→ FO becomes syntactic sugar, and we can rely on traditional model checking

Still, convergence is PSPACE-hard, without any assumption on the network topology:

1. Elect a leader
2. Construct a tree rooted in the leader
3. Linearize the tree
4. Use it to simulate a corridor-tiling problem

The hardest case: Interactive CDPs without restrictions

The hardest case: Interactive CDPs without restrictions

A node is a computing device with a finite-state control and unbounded memory
… This is also called a Turing Machine!

→ Even the simplest form of verification, i.e., reachability, is undecidable

Size-boundedness

Put an a-priori bound *b* on the number of constants that can simultaneously appear in the DB.

- Studied extensively over the past 10 years (under the name of “state-boundedness”)
- In general, the resulting transition system is still infinite-state
- The bound *b* might be known or not-known
- In CDPs, boundedness can be applied to the different types of DBs:
  - state-boundedness
  - transport-boundedness
  - input-boundedness

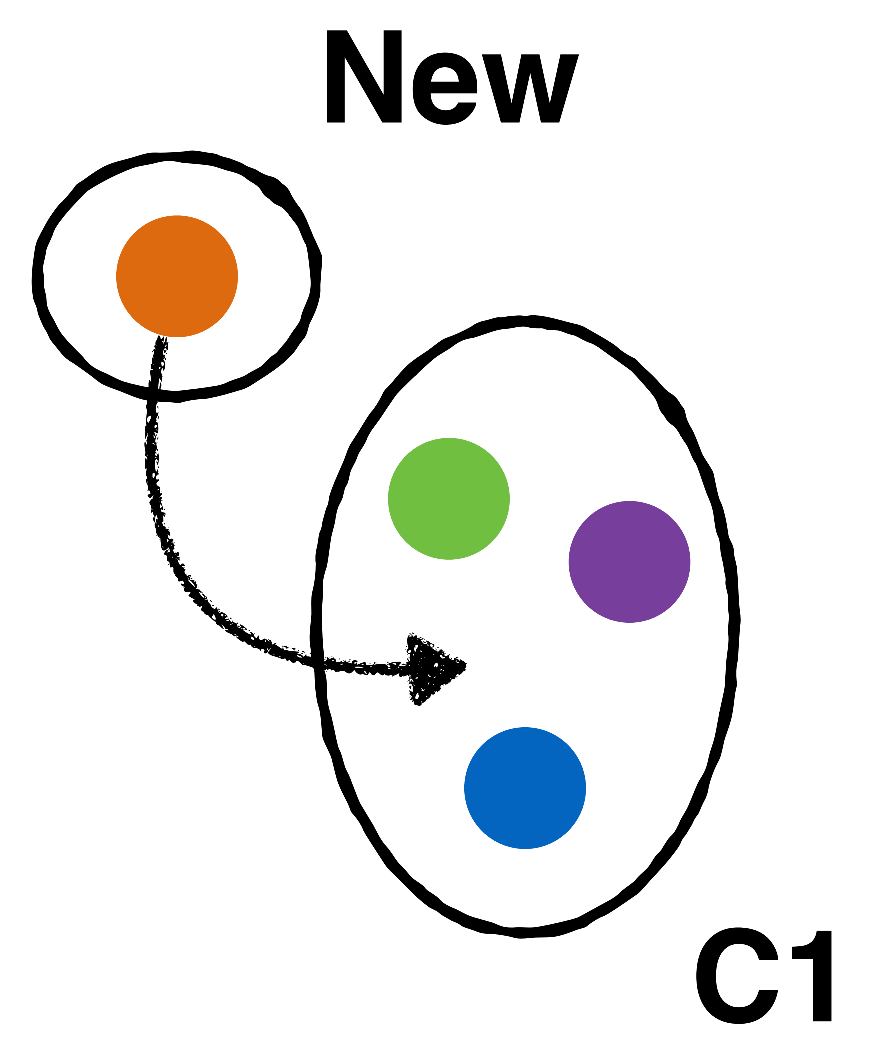
Results for FO-CTL verification

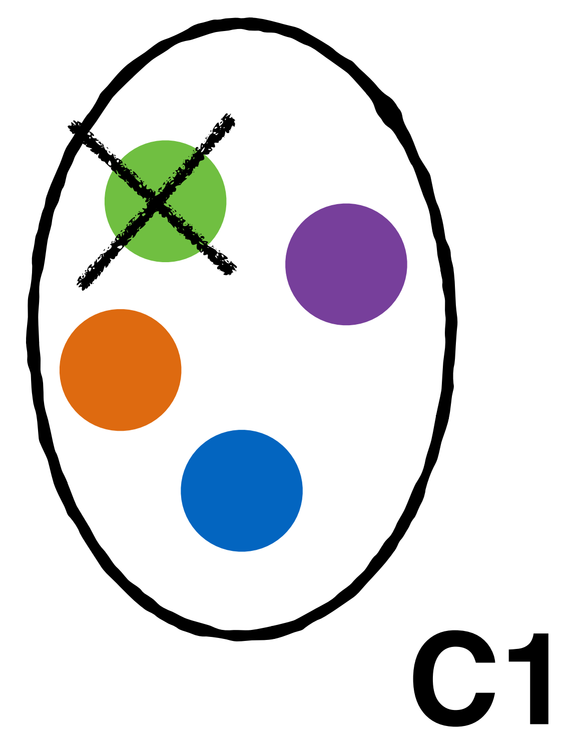
Type of CDP	Input bounded	State / Channel bounded		
		N / Y	Y / N	Y / Y
Interactive	N	U	U	PSPACE-c
	Y	U	U	PSPACE-c
Autonomous	N	U	U	U
	Y	N.A.	N.A.	PSPACE-c
Closed	N.A.	N.A.	N.A.	PSPACE-c

Undecidability — State unbounded

Simulation of a 2-counter Minsky machine

- Single node with 2 unary relations **C1** and **C2**
- A single unary input relation **New**
- Increment counter 1:
  - check whether **New** contains an object not in **C1**
  - if not, enter into an error state
  - if so, insert it in **C1**
- Decrement counter 1: pick an object in **C1** and remove it
- Test counter 1 for zero: check that **C1** is empty





Undecidability — Transport unbounded

Again, by simulation of a 2-counter Minsky machine

- Single node with self-loop channel with 2 **binary** relations **C1** and **C2**, each of which represents a cyclic graph of length equal to the counter value + 1, initialized to (**`me`**,**`me`**)
- A single unary input relation **New**
- Increment counter 1:
  - check whether **New** contains an object not in **C1**, by "working off" the cycle from the channel
  - if not, or if unexpected tuples are obtained from the channel, enter into an error state
  - if so, extend **C1** by incorporating the new object
- Decrement counter 1: remove from **C1** the initial tuple (**`me`**,x), with x ≠ **`me`**, thus shortening the cycle
- Test counter 1 for zero: check that **C1** contains the tuple (**`me`**,**`me`**)

Towards decidability — Key properties of CDPs

CDPs (and other similar logic-based formalisms for data-aware processes) enjoy two key properties, since they are:

- **Markovian**: Next state only depends on the current state + input. Two states with identical node DBs are bisimilar.
- **Generic**: Datalog (as all query language) does not distinguish structures that are identical modulo uniform renaming of data objects

→ Two isomorphic CDP snapshots are bisimilar

Exploited to prune infinite branching

- Consider a system snapshot and its node DBs
- Input is bounded → Only boundedly many isomorphic types relating the input objects and those in the active domain of the CDP
- Input configurations in the same isomorphic type produce isomorphic snapshots
- Keep only one representative successor state per isomorphic type
- The “pruned” transition system is finite-branching and bisimilar to the original one

Compacting infinite runs via recycling

- **Key observation**: under restricted quantification (persistence-preserving, or no quantification across), the logic cannot distinguish local from global freshness
- Hence, we modify the transition system construction:
  whenever we need to consider a fresh representative object …
  - … if there is an old object that can be recycled → use that one
  - … if not → pick a globally fresh object
- Properties of this **recycling technique**:
  - under size-boundedness, we need only a bounded number of objects (we do not need to know the bound)
  - it preserves bisimulation!

→ We obtain a **finite-state transition system**

Resulting verification algorithm

- **Input**: Interactive CDP all of whose DBs are size-bounded
- Construct the **abstract transition** system that works over **isomorphic types** and **recycles objects**. The abstract transition system is:
  - finite-state
  - bisimilar to (i.e., a faithful representation of) the original one
- Use the abstract system to model-check FO-LTL or FO-CTL formulas (with suitably restricted quantification) using conventional techniques

→ PSPACE upper bound

Decidability with unbounded size

The undecidability results leave little margin for relaxing boundedness

- state-boundedness is unavoidable
- input-boundedness is not required, but unboundedness cannot be really exploited
- transport-unboundedness leads to undecidability, provided we can use a binary relation in the channel

→ This leaves the case open, where we are **not transport-bounded**, but the channels contain unary relations only — Still a significant case

- Exploiting an encoding into coverability for nu-Petri Nets, we have shown decidability of verification of convergence properties in this case
- The general case of FO-CTL (and fragments of it) stays undecidable

Conclusions

- The formal analysis of dynamic systems considering data and distribution is still a major challenge that needs to be addressed
- The problem space is very large, since many different factors crucially affect decidability/complexity of verification
- Significant progress has been made, through a rather fine-grained analysis, but the overall picture is still very fragmented
- Interesting contributions and techniques have come from many diverse areas, and you are all welcome to collaborate with your favourite toolbox!