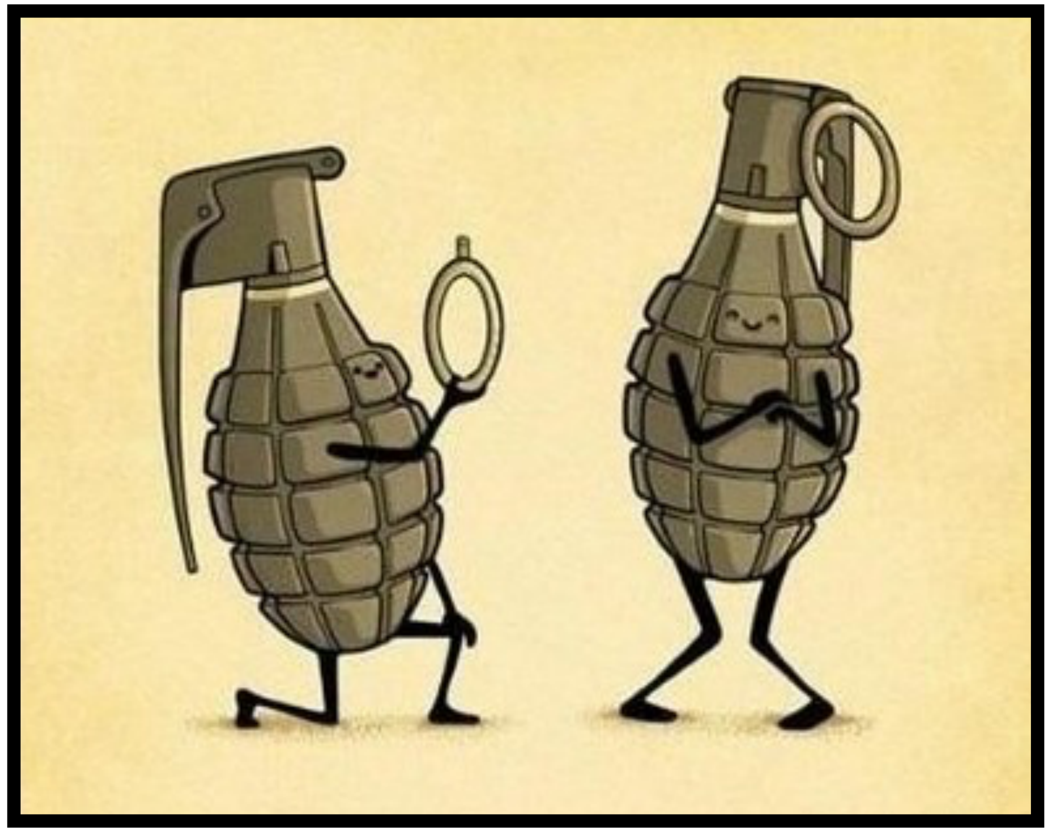
Data and Processes:



A Challenging, though Necessary, Marriage

Marco Montali

Free University of Bozen-Bolzano

"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."

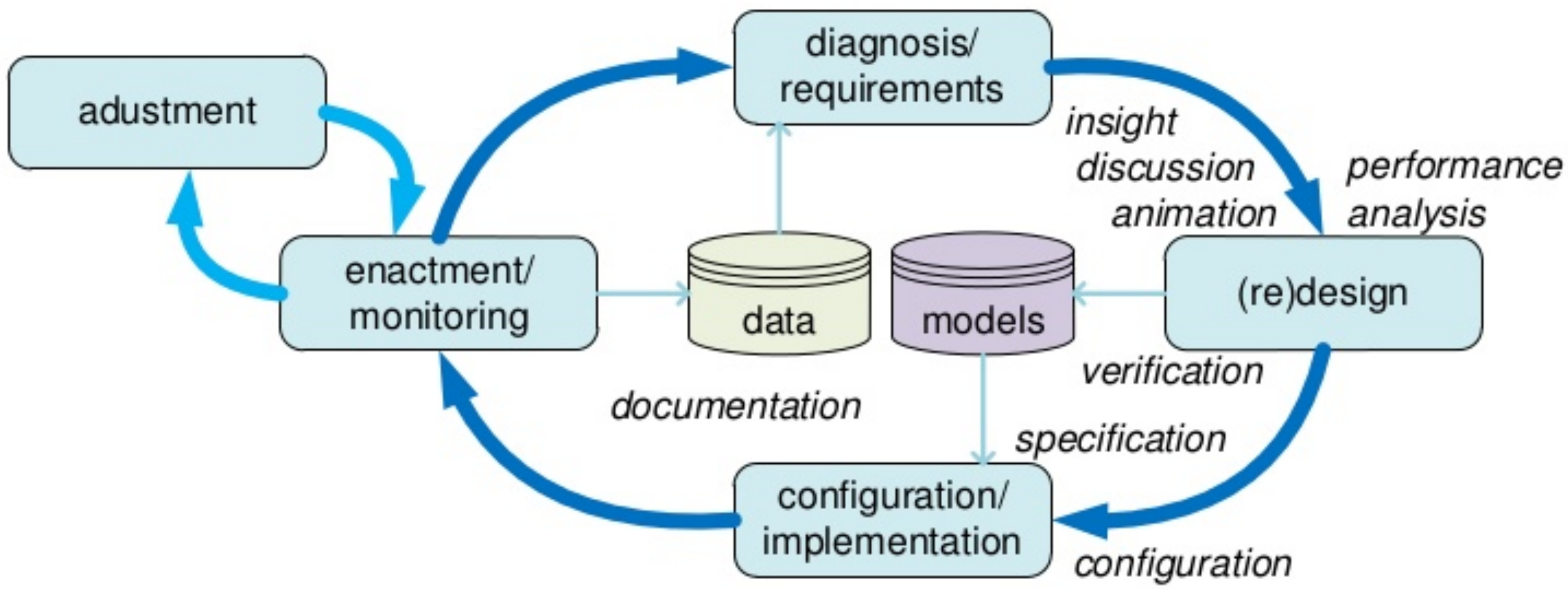
Our Starting Point

Marrying processes and data is a must if we want to really *understand* how complex dynamic systems operate

Dynamic systems of interest:

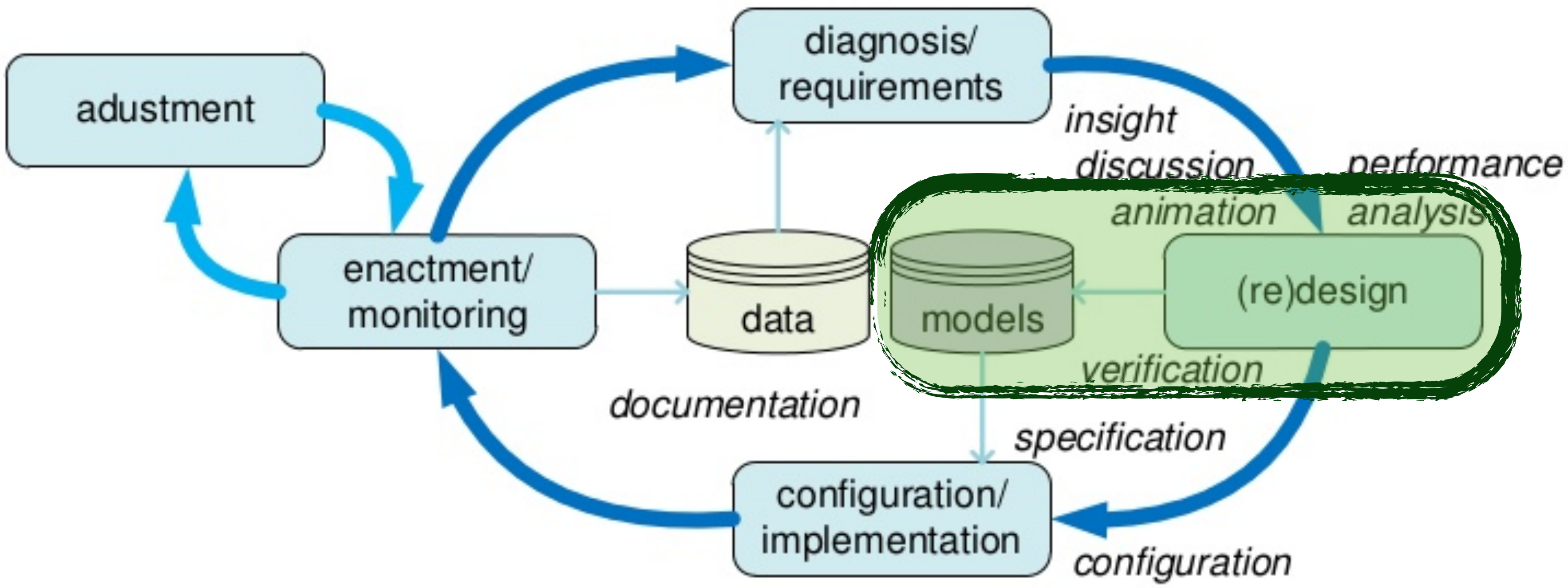
- business processes
- multiagent systems
- distributed systems

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

Our Thesis

Knowledge representation and computational logics

can become a swiss-army knife to

understand data-aware dynamic systems,
and
provide automated reasoning and verification capabilities along their entire lifecycle

Warning!

Towards this goal, I believe we have to:

- *foster cross-fertilization* with related fields such as database theory, formal methods, business process management, information systems
- systematically classify the *sources of undecidability and complexity*, so as to attack them when developing concrete tools
- continuously *validate* how foundational results relate to practice

Practice

Practice

+ methodologies

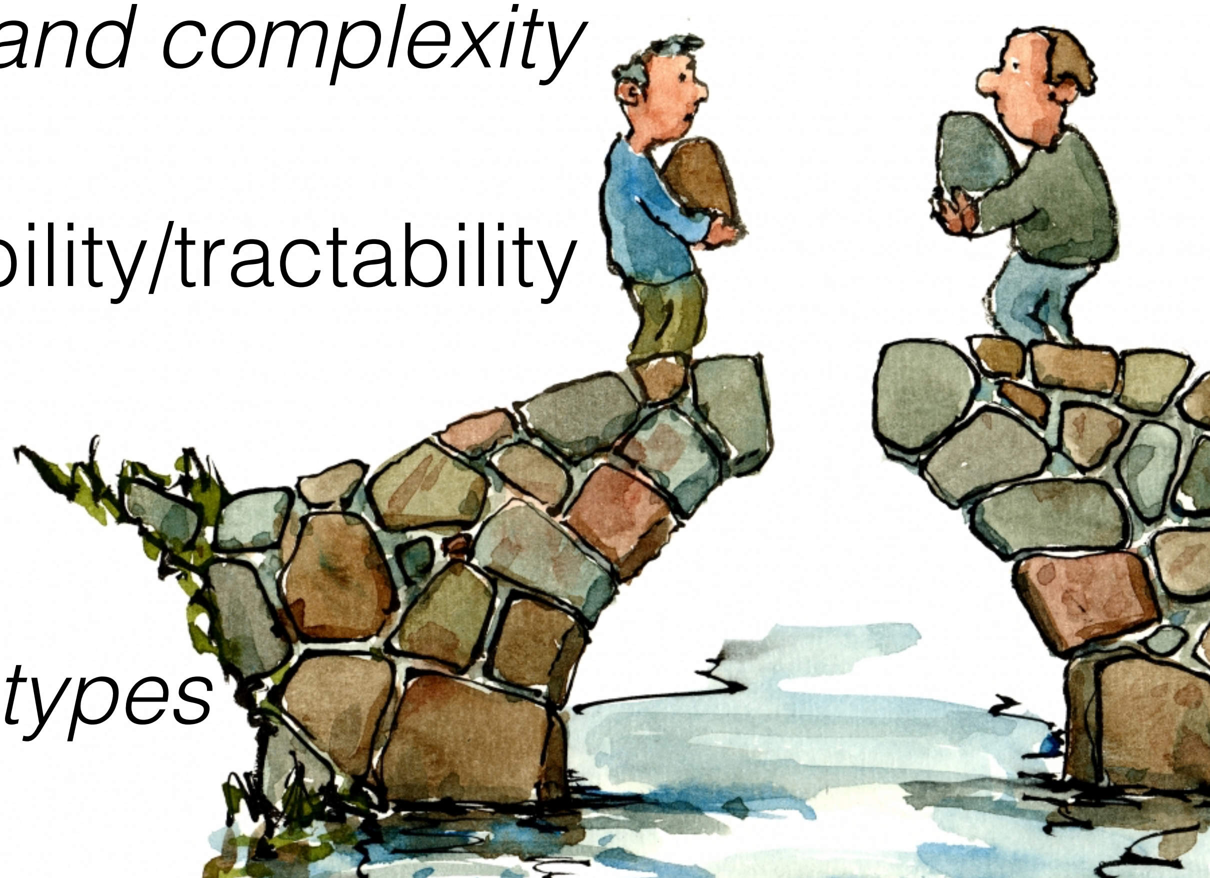
Theory

Theory

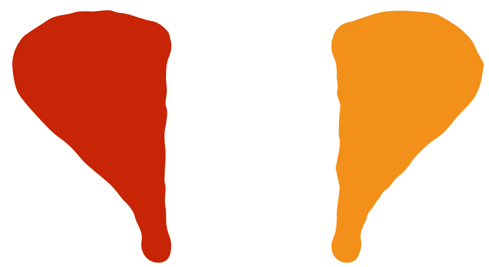

Theorem
Theorem
Theorem
Theorem
Theorem
Theorem
Theorem
Theorem
Theorem
Theorem
Theorem

Our Approach

1. Develop *formal models* for data-aware dynamic systems
2. Show that they can capture *concrete modeling languages*
3. Outline a *map of (un)decidability and complexity*
4. Find *robust conditions* for decidability/tractability
5. Bring them *back into practice*
6. Implement proof-of-concept *prototypes*



Outline: 3 Acts

1. Loneliness 

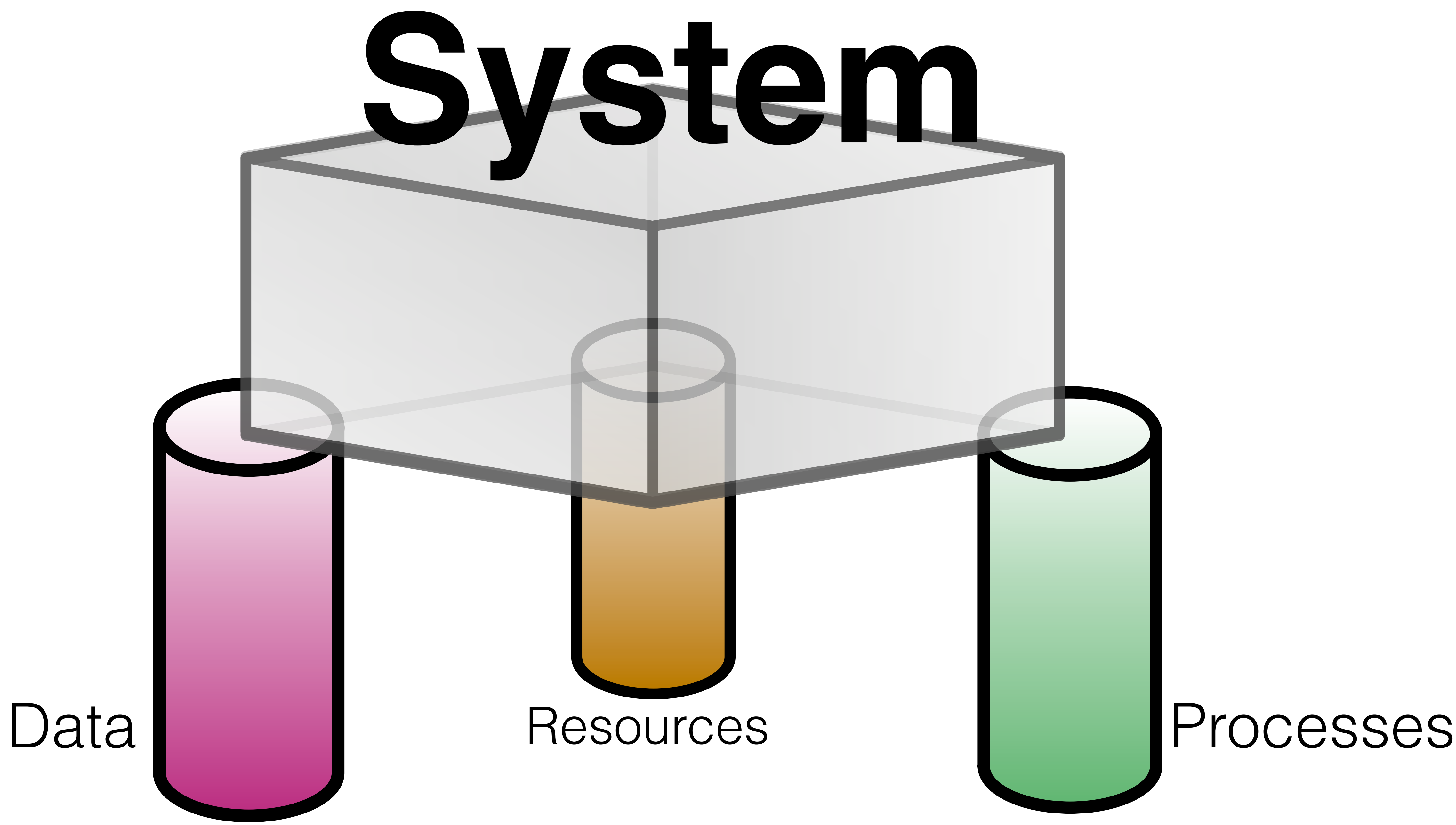
2. Marriage 

3. Hate and love 

Act 1

Loneliness

The Three Pillars of Complex Systems



In AI and CS, we know **a lot** about each pillar!

Information Assets

- **Data**: the main information source about the history of the domain of interest and the relevant aspects of the current state of affairs
- **Processes**: how work is carried out in the domain of interest, leading to evolve data
- **Resources**: humans and devices responsible for the execution of work units within a process

We focus on the first two aspects!

State of the Art

- Traditional isolation between processes and data
- Why? To attack the complexity (*divide et impera*)
- **AI has greatly contributed to these two aspects**
    - *Data*: knowledge bases, conceptual models, ontologies, ontology-based data access and integration, inconsistency-tolerant semantics, …
    - *Processes*: reasoning about actions, temporal/dynamic logics, situation/event calculus, temporal reasoning, planning, verification, synthesis, …

Application Domains

	Data	Process
Business Process Management	• Information system	• Activities + events • Control-flow constraints • External inputs
Multiagent Systems	• Knowledge of agents • Institutional knowledge	• Speech acts • Creation of new objects • Interaction protocols
Distributed Systems	• Facts maintained by the system nodes	• Exchanged messages • Application-level inputs

Loneliness in BPM

Data/Process Fragmentation

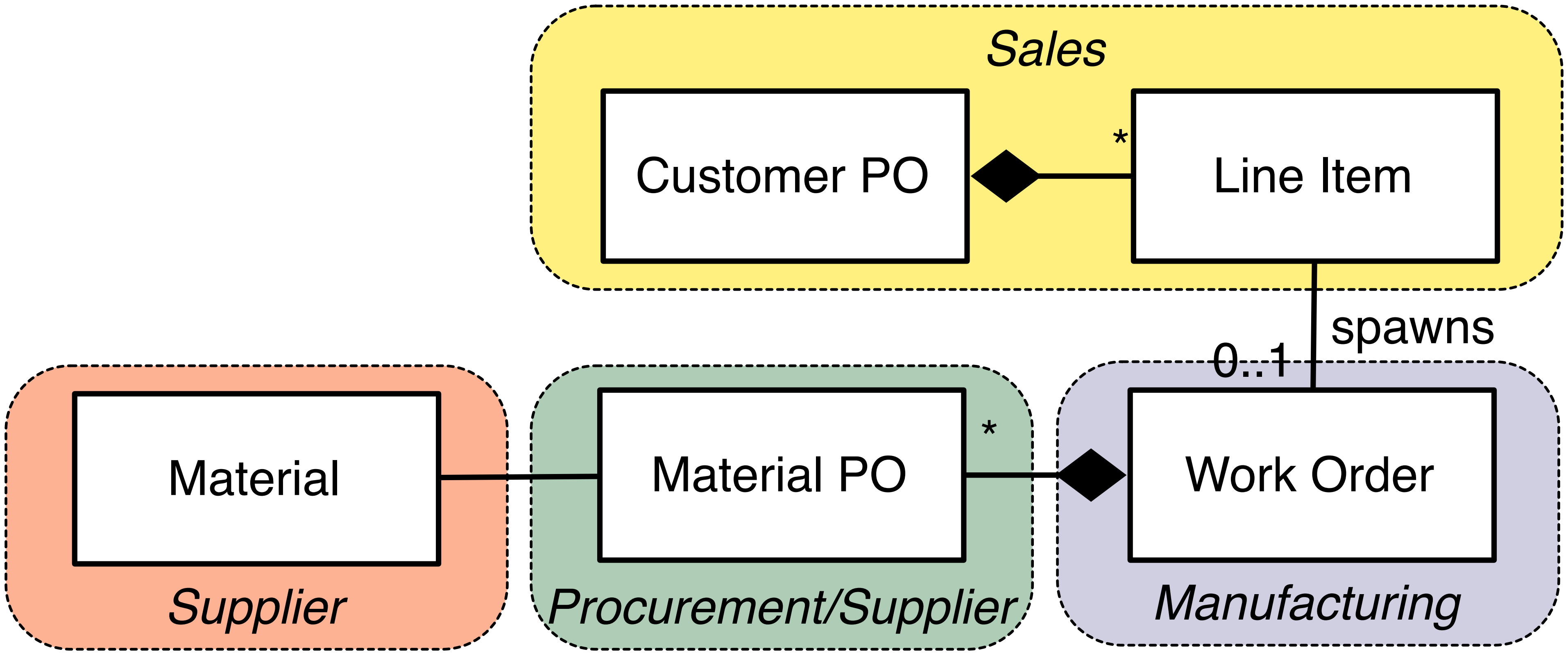
- A business process consists of a **set of activities that are performed in coordination** in an organizational and technical environment [Weske, 2007]
- **Activities change the real world**
  - The corresponding updates are reflected into the organizational information system(s)
- **Data trigger** decision-making, which in turn determines **the next steps to be taken in the process**
- Survey by *Forrester* [Karel et al, 2009]: **lack of interaction between data and process experts**

Experts Dichotomy

- *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
- Forrester: in 83/100 companies, **no interaction at all** between these two groups
    - *This isolation propagates to languages and tools*, which never properly account for the process-data connection

Conventional Data Modeling

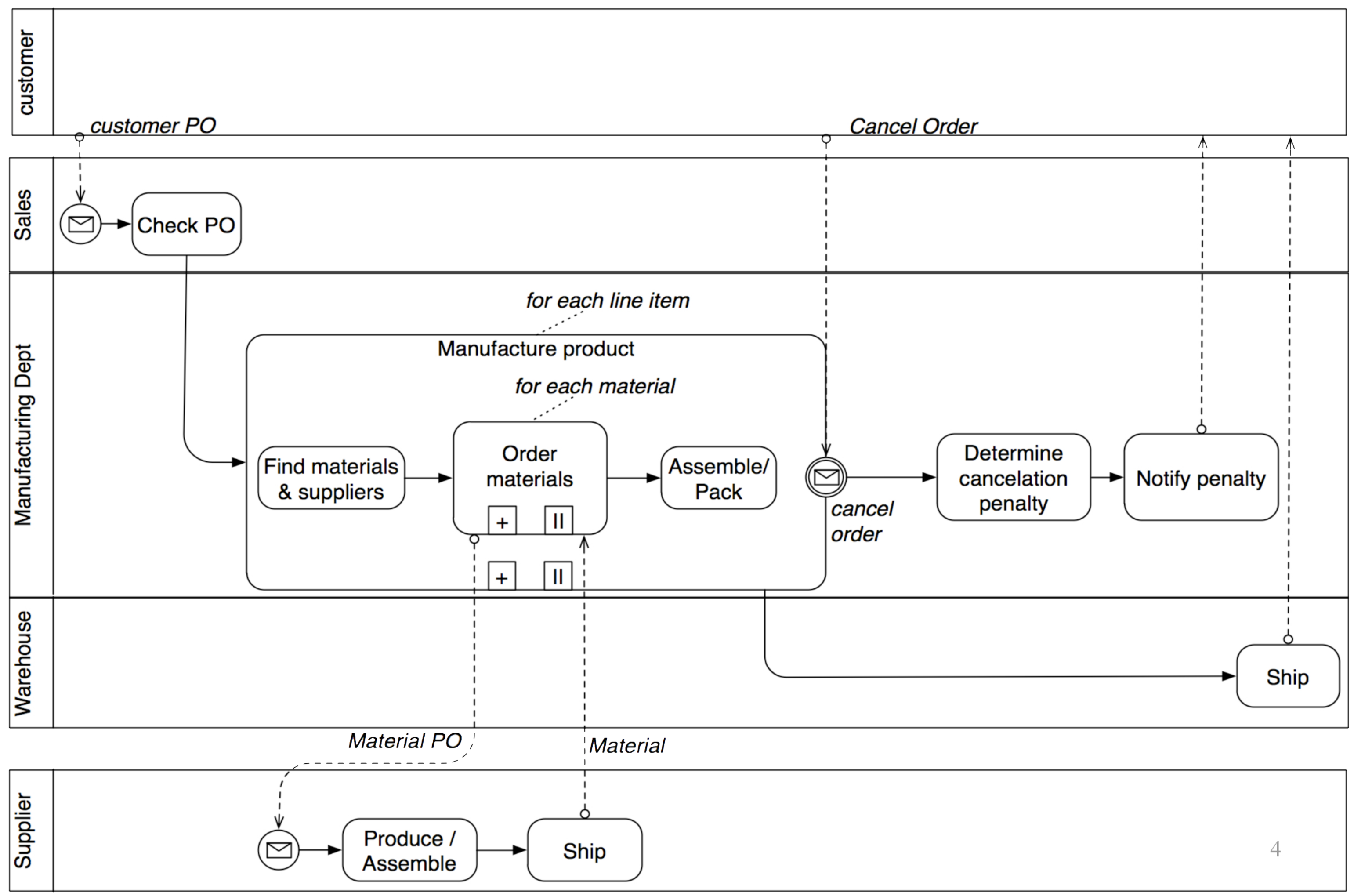
Focus: revelant entities, relations, static constraints



But… how do data evolve?
Where can we find the "state" of a purchase order?

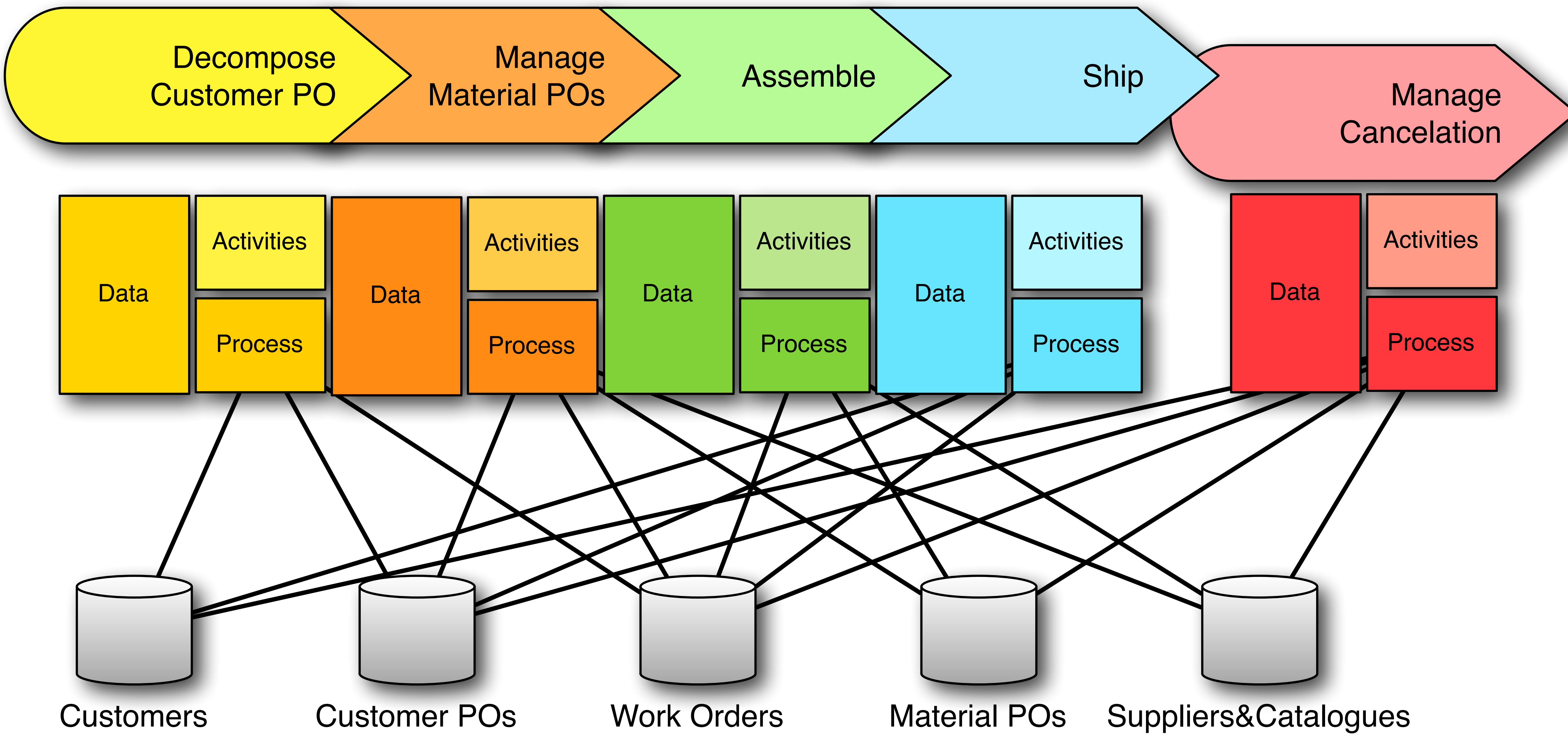
Conventional Process Modeling

Focus: control-flow of activities in response to events



But… how do activities update data?
What is the impact of canceling an order?

Do you like Spaghetti?



IT integration: difficult to manage, understand, evolve

The Need of Conceptual Integration

- [Meyer et al, 2011]: data-process integration crucial to assess the **value of processes** and **evaluate KPIs**
- [Dumas, 2011]: data-process integration crucial to **aggregate** all relevant **information**, and to suitably **inject business rules** into the system
- [Reichert, 2012]: **“Process and data are just two sides of the same coin”**

Business Entities/Artifacts

Data-centric paradigm for process modeling

- First: *elicitation of relevant business entities* that are evolved within given organizational boundaries
- Then: definition of the *lifecycle* of such entities, and how *tasks trigger the progression* within the lifecycle
- Active research area, with concrete languages (e.g., IBM GSM, OMG CMMN)
- Cf. **EU project ACSI** (completed)

Loneliness in Social Commitments

Social Commitments

Semantics for agent interaction that abstracts away from the internal agent implementation

- [Castelfranchi 1995]: social commitments as a *mediator between an individual and its "normative" relation with other agents*
- Extensively adopted for flexible specification of multiagent interaction protocols, business contracts, interorganizational business processes (cf. work by Singh et al)

Conditional Commitments

`CC(debtor,creditor,ϕ,ψ)`

- When condition ϕ holds, the debtor agent *becomes committed* towards the creditor agent to make condition ψ true
- Agents change the state of affairs implicitly causing conditions to become true/false
- Commitments are consequently progressed reflecting the normative state of the interaction

Literature Example

- Contract between Bob (seller) and Alice (customer):

```
CC(bob,alice,item_paid,item_owned)
```

- Actions available to agents:

```
pay_with_cc causes item_paid
send_by_courier causes item_owned
deliver_manually causes item_owned
```

Literature Example

- Contract between Bob (seller) and Alice (customer):

```
CC(bob,alice,item_paid,item_owned)
```

- Actions available to agents:

```
pay_with_cc causes item_paid
send_by_courier causes item_owned
deliver_manually causes item_owned
```

Is this satisfactory???

Reality

- *Multiple* customers, sellers, items
  —> *Many-to-many* business relations established as *instances* of the same contractual commitment
- Need of **co-referencing commitment instances** through agents and the exchanged data
  - If **Bob** gets paid by **Alice** for **a laptop**, then **Bob** is commitment to ensure that **Alice** owns **that laptop**
- More in general, see work by Ferrario and Guarino on service foundations

From the Literature to Reality

(At least) two fixes required [Montali et al, 2014]:

1. Agent actions/messages must carry an **explicit data payload** (Alice pays *an item* with cc)
2. **Commitments** and dynamics have to become **data-aware**

```
forall Seller S, Customer C, Item I.
  CC(S,C,Paid(C,I,S),Owned(C,I))
```

Formal Verification

The Conventional, Propositional Case

Process control-flow
Agent behaviors/protocols

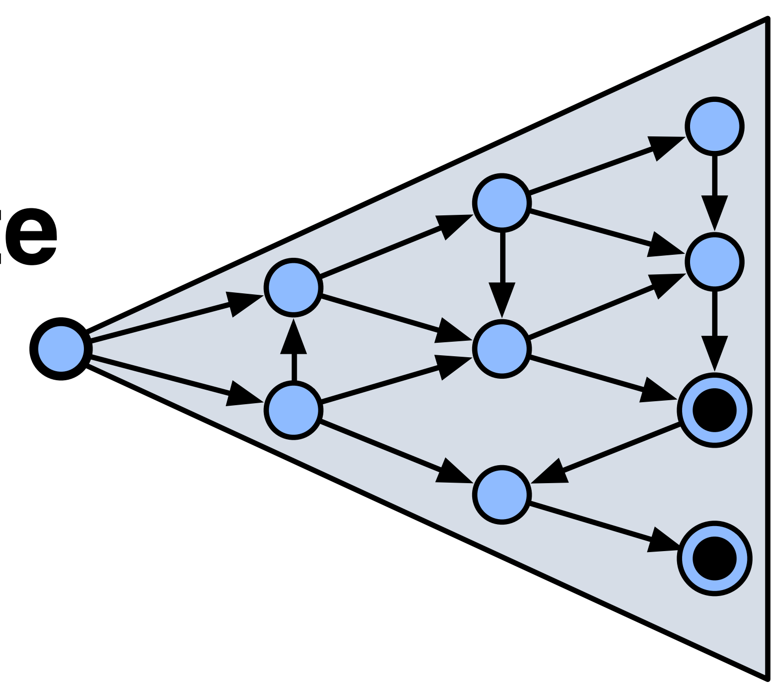
(Un)desired property

Formal Verification

The Conventional, Propositional Case

Process control-flow
Agent behaviors/protocols

Finite-state
transition
system



Φ **Propositional**
temporal formula

(Un)desired property

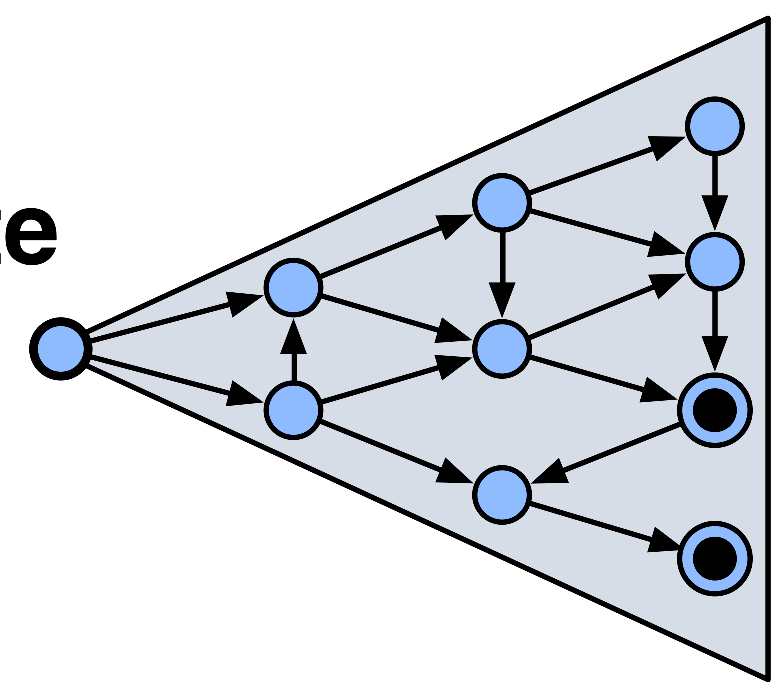
Formal Verification

The Conventional, Propositional Case

Process control-flow
Agent behaviors/protocols

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

Finite-state transition system



$\models \Phi$

Propositional temporal formula

(Un)desired property


Marriage
Act 2

Formal Verification

The Data-Aware Case

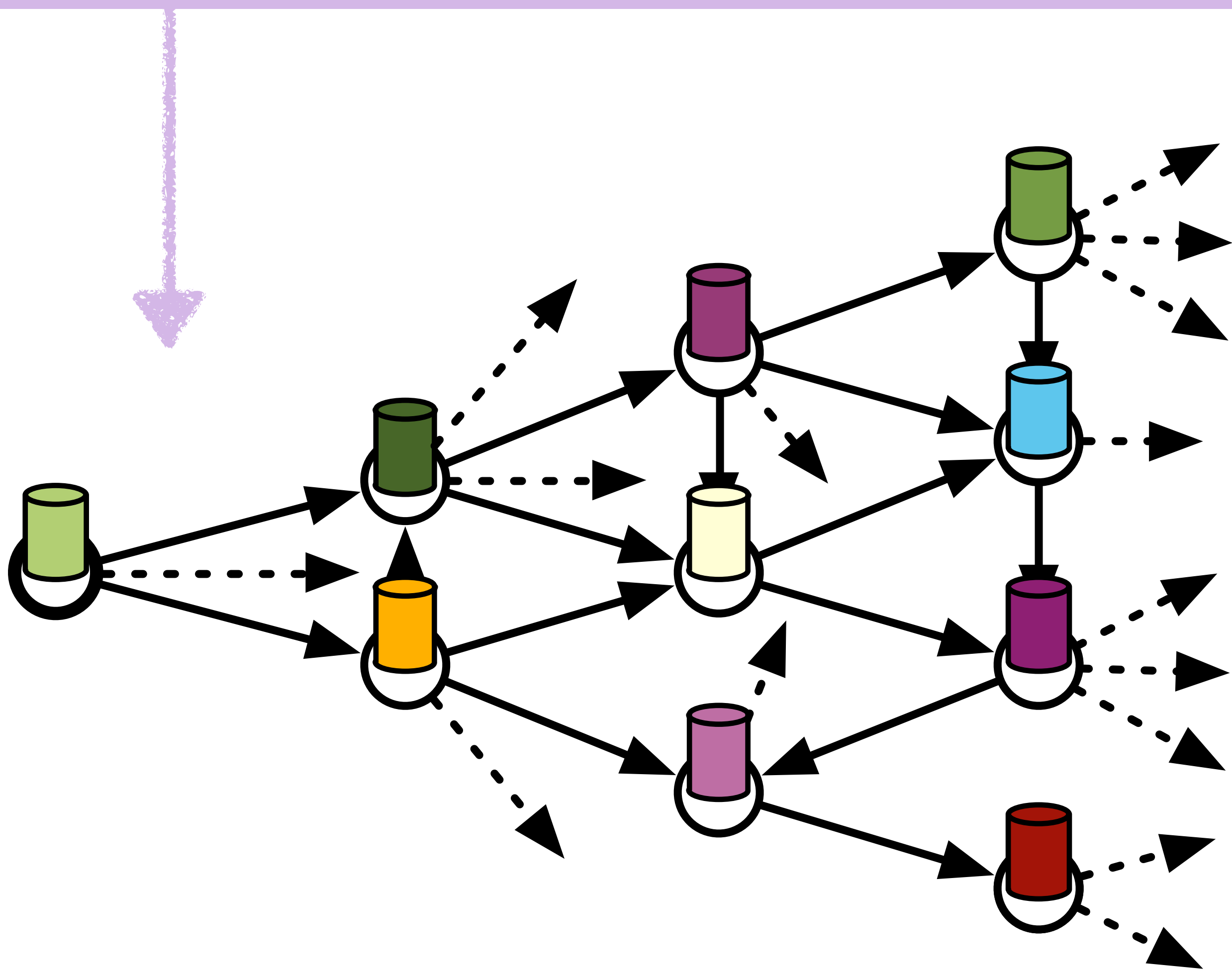
Process+Data
Data-aware agent behaviors/protocols

(Un)desired property

Formal Verification

The Data-Aware Case

Process+Data
Data-aware agent behaviors/protocols



Infinite-state, relational
transition system [Vardi 2005]

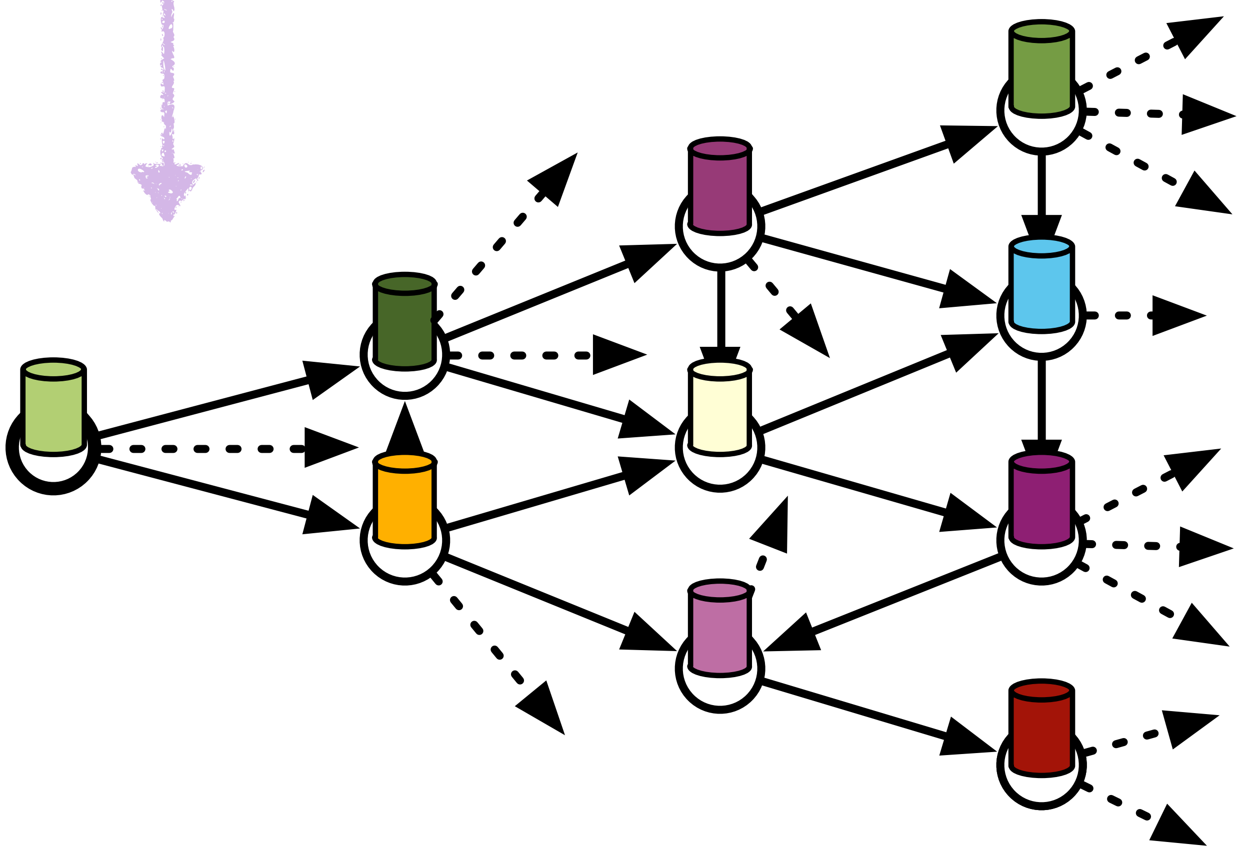
Φ **First-order**
temporal formula

(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 Logics

- To inspect data: FO queries
- To capture system dynamics: temporal modalities
- To track the 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

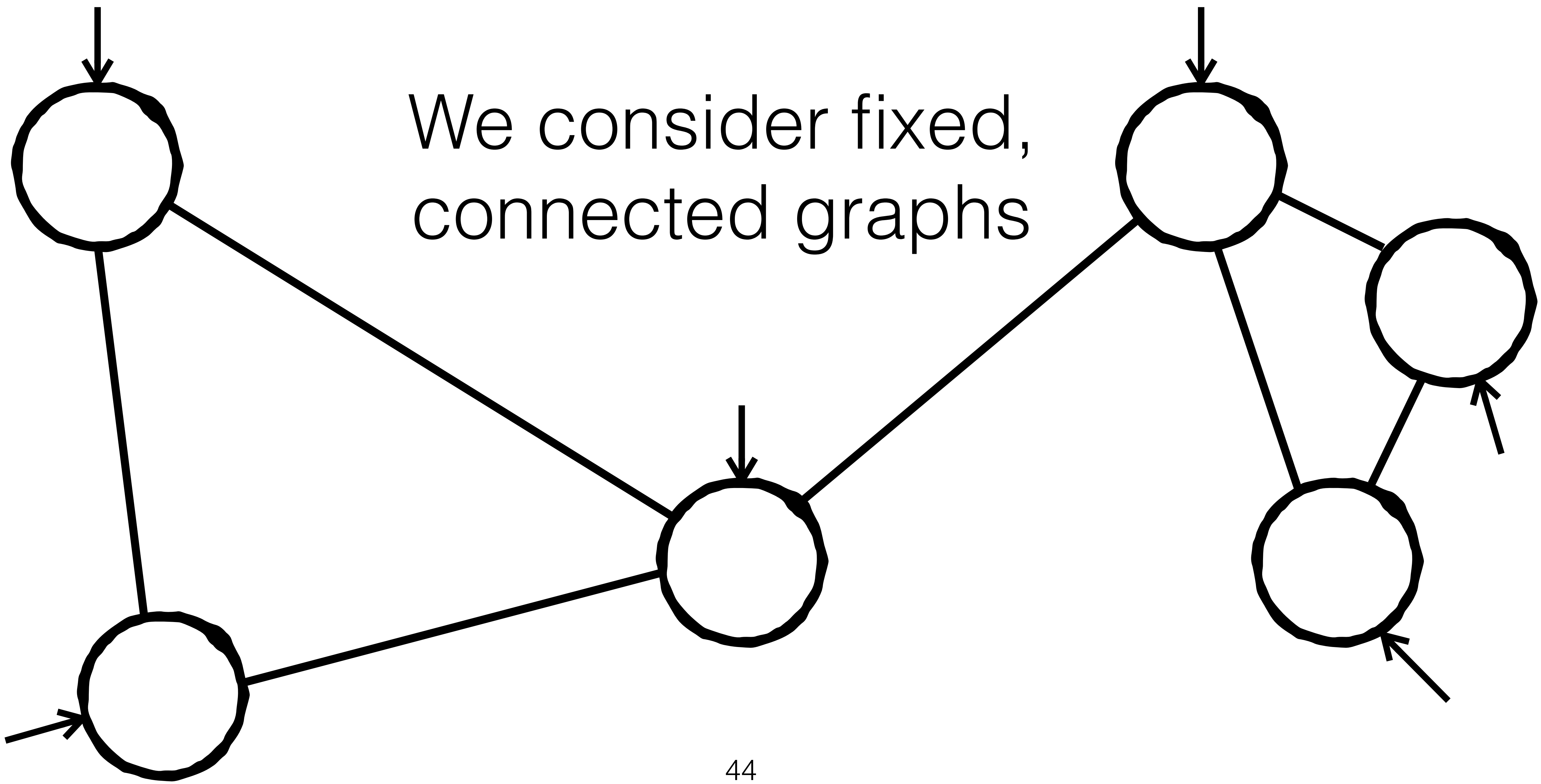
Data component	Relational DB	Description logic KB	OBDA system	Inconsistency tolerant KB	…
Process component	condition-action rules	ECA-like rules	Golog program	…	
Task modeling	Conditional effects	Add/delete assertions	Logic programs	…	
External inputs	None	External services	Input DB	Fixed input	…
Network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

Declarative Distributed Computing

Distributed, data-centric computing with extensions of Datalog

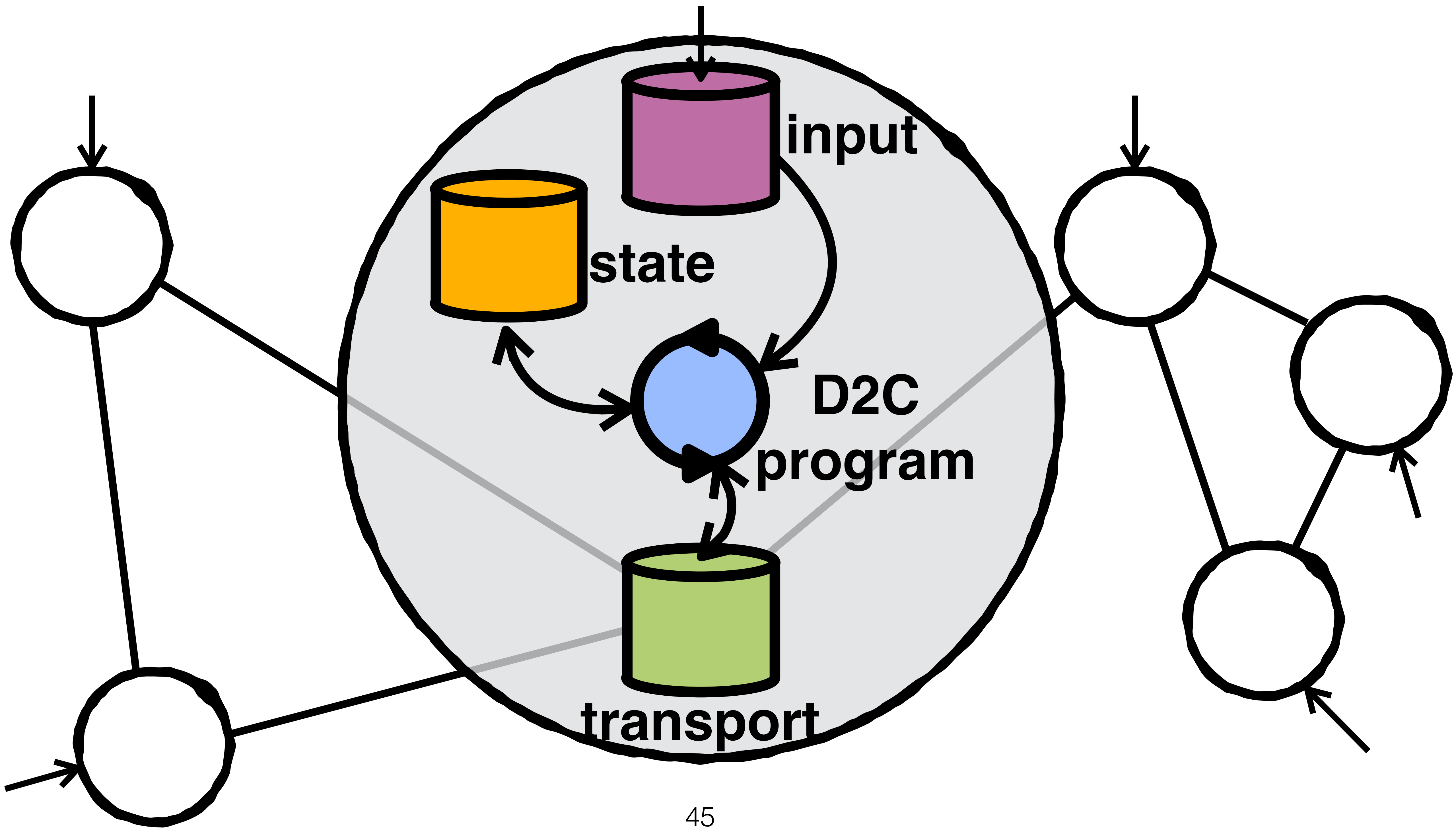
- Pushed the renaissance of Datalog [Loo et al, 2009] [Hellerstein, 2010]
- Compares well with standard approaches [Loo et al, 2005]
- Many applications: distributed query processing, distributed business processes, web data management, routing algorithms, software-defined networking, …

Declarative Distributed Systems (DDS)



We consider fixed, connected graphs

Declarative Distributed Systems (DDS)



D2C Programs

- Datalog programs extended with
  - **non-determinism**: *choice* construct
    [Saccà and Zaniolo, 1990]
  - **time**: *prev* construct to refer to the previous state
    location: @ construct to refer the sender/receiver nodes
- *Stable model semantics*

- Each node has initial knowledge about its neighbors, and starts with a given state DB

- **Input** relations are read-only, and **may inject fresh data from an infinite data domain** (strings, pure names, …)

Node Reactive Behavior

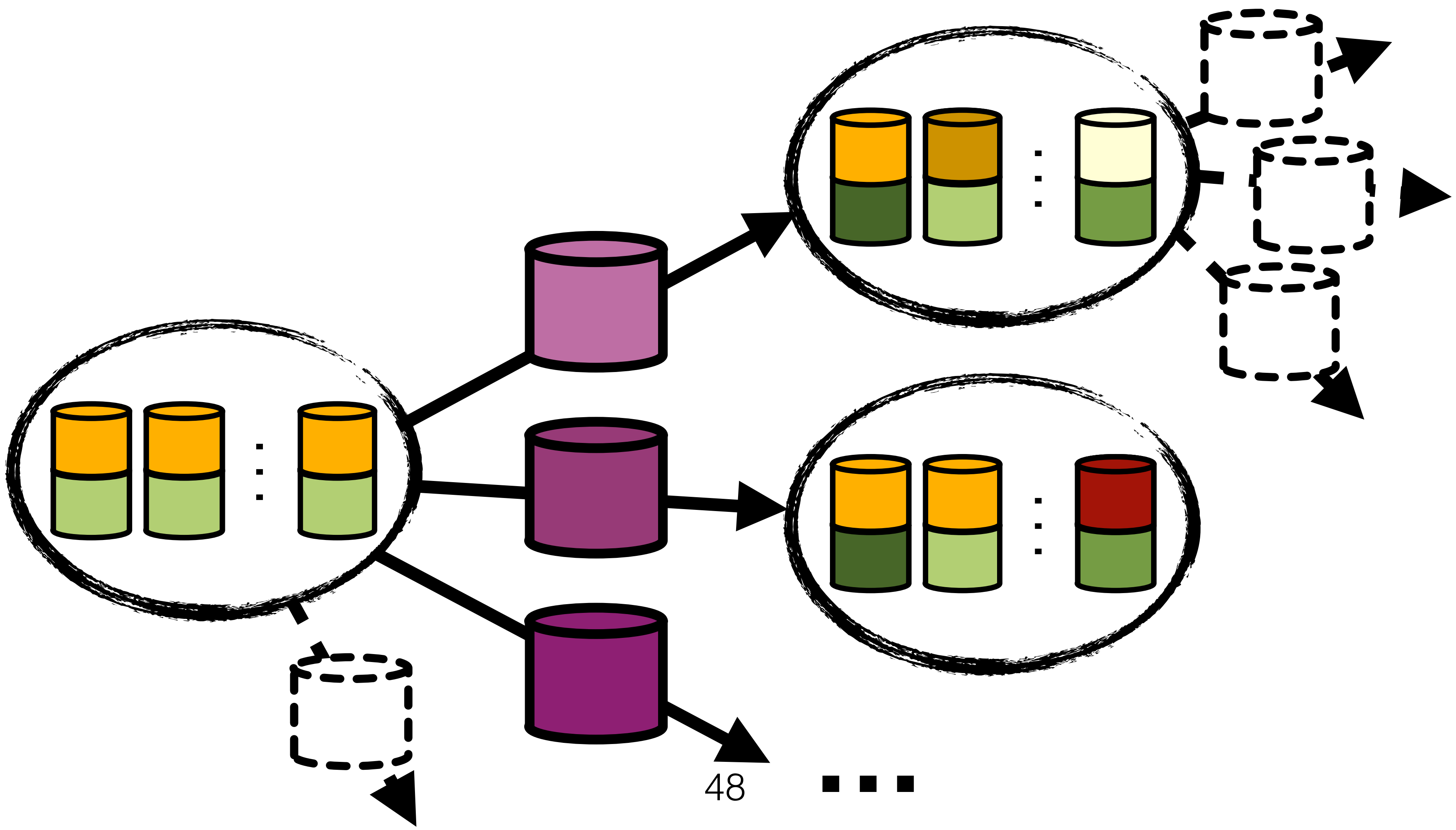
Whenever a node receives (a set of) incoming messages, it performs **a transition**:

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

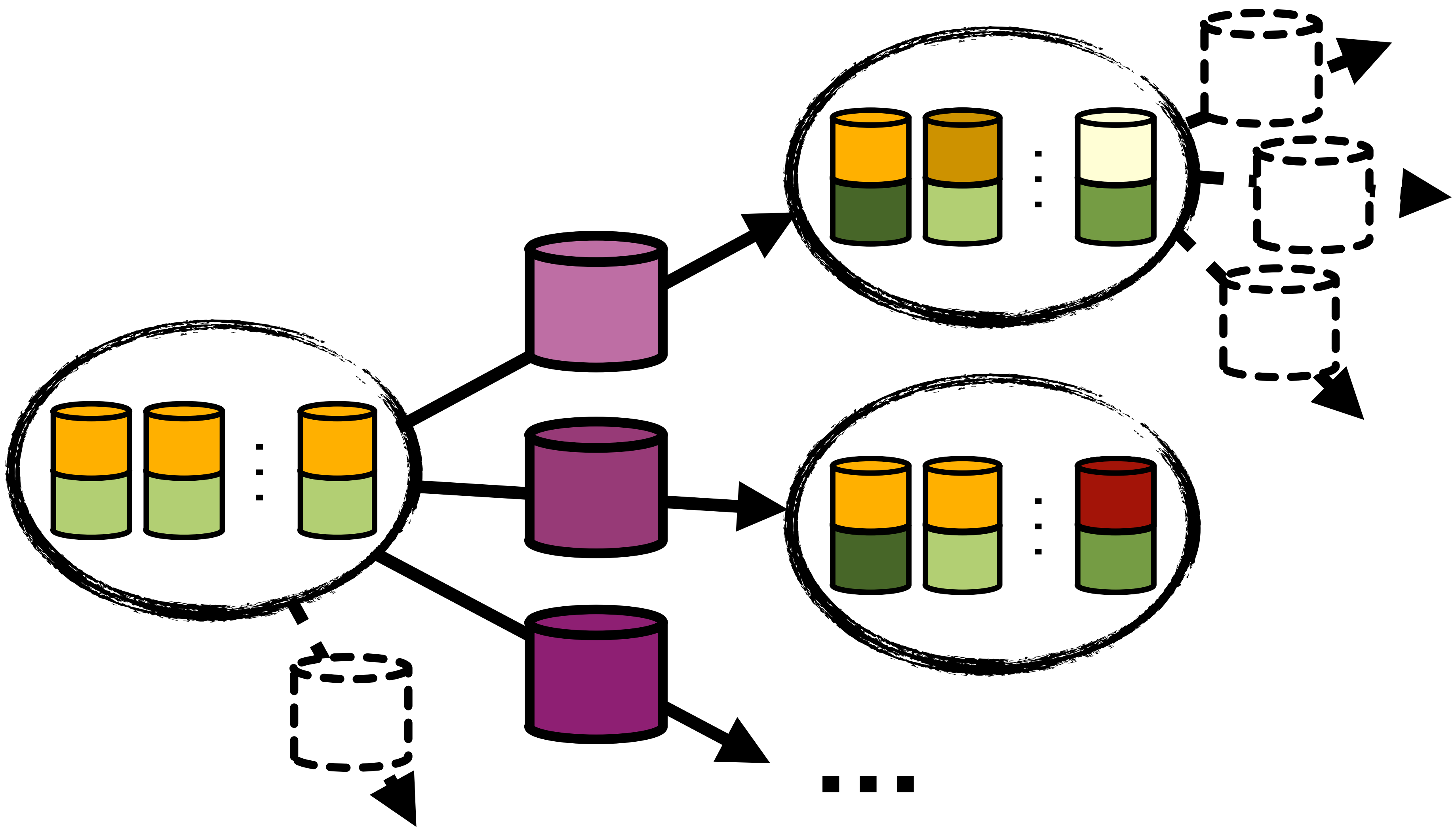
Execution Semantics

Relational transition systems with node-indexed databases

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

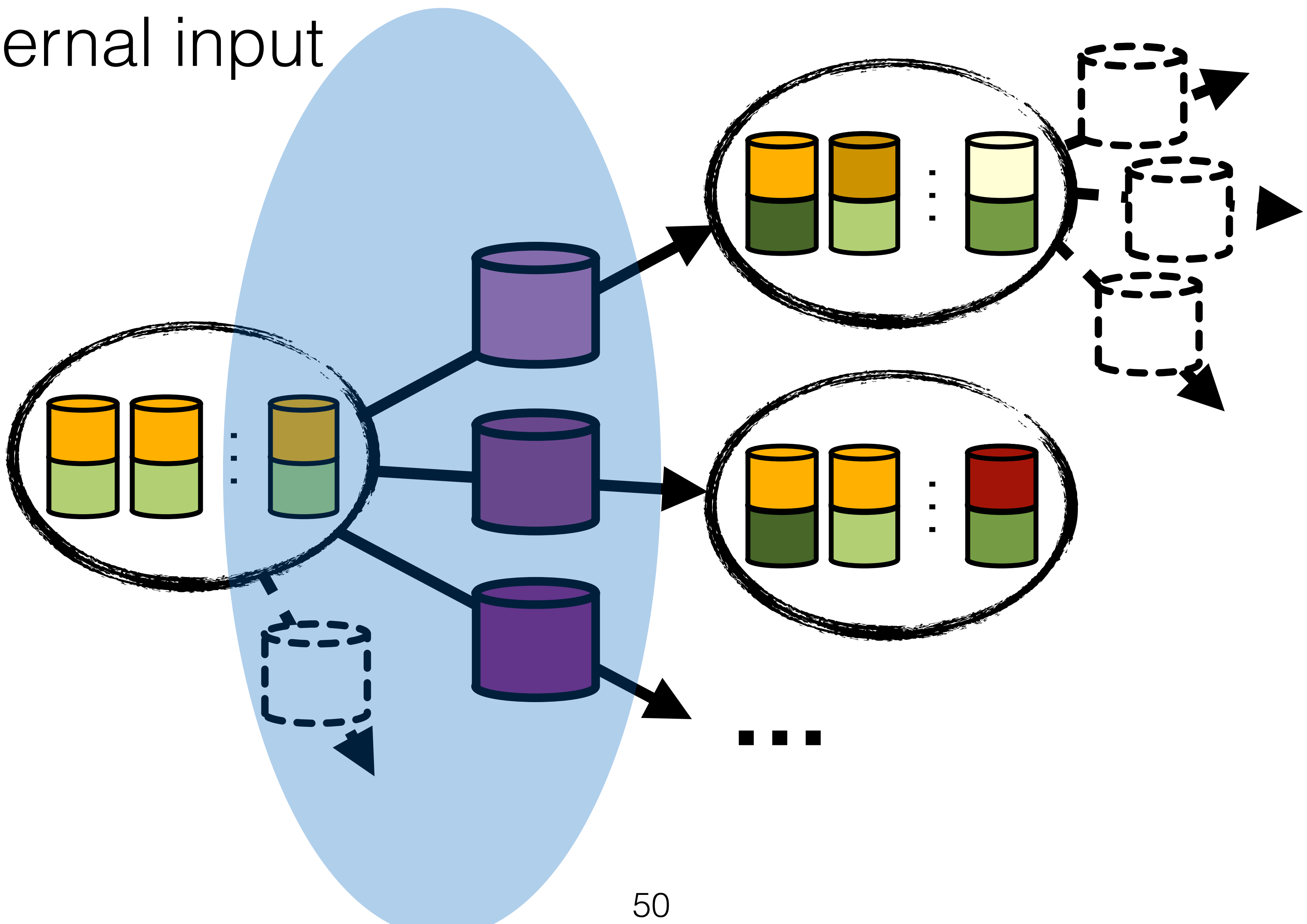


Sources of Infinity



Sources of Infinity

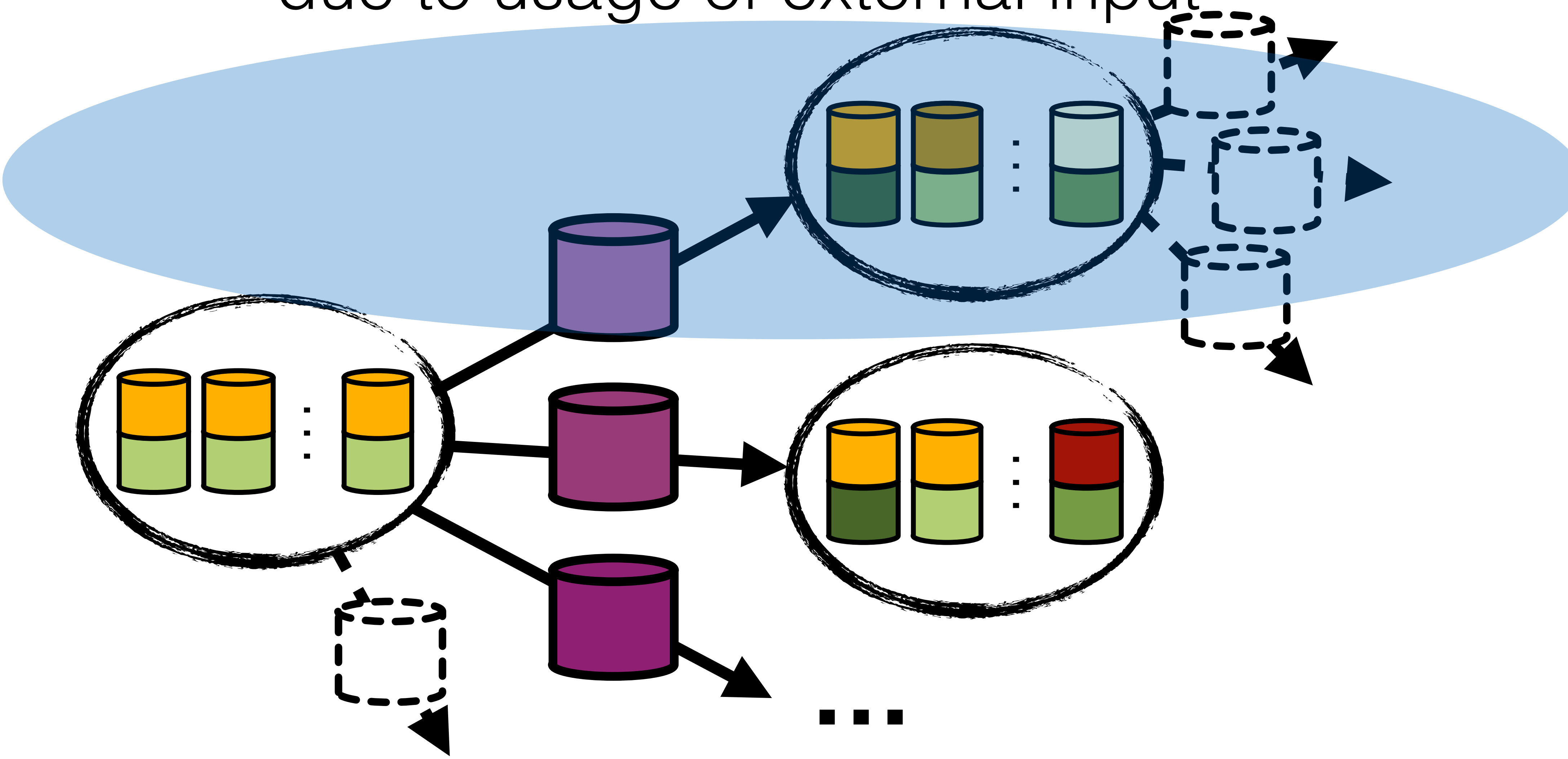
Infinite-branching
due to external input



Sources of Infinity

Runs visiting infinitely many DBs

due to usage of external input



Pure Declarative Semantics

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

Classes

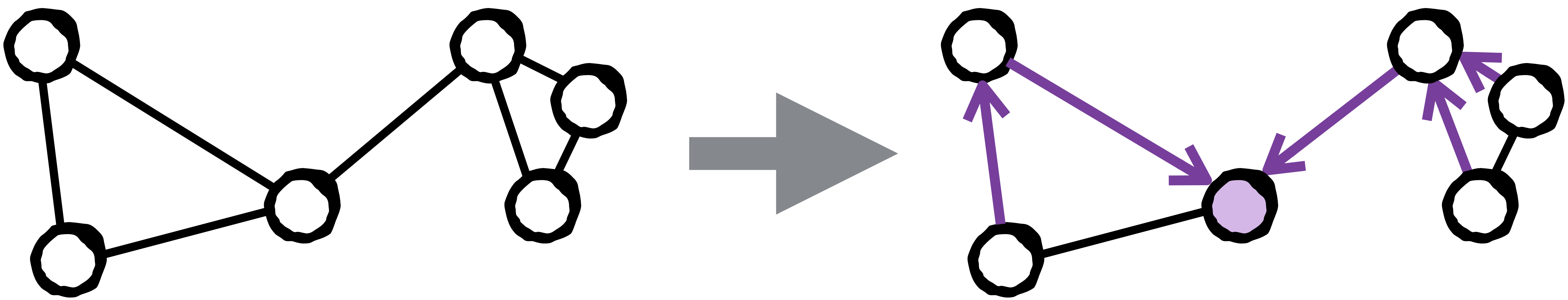
	synchronous global clock	**asynchronous ordered** interleaving semantics
closed no input	finite-state transition system	infinite-state transition system
interactive continuous input	infinite-state transition system	infinite-state transition system

Classes

	synchronous global clock	**asynchronous ordered** interleaving semantics
closed no input	**finite-state transition system**	infinite-state transition system
interactive continuous input	**infinite-state transition system**	infinite-state transition system

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(_).
```

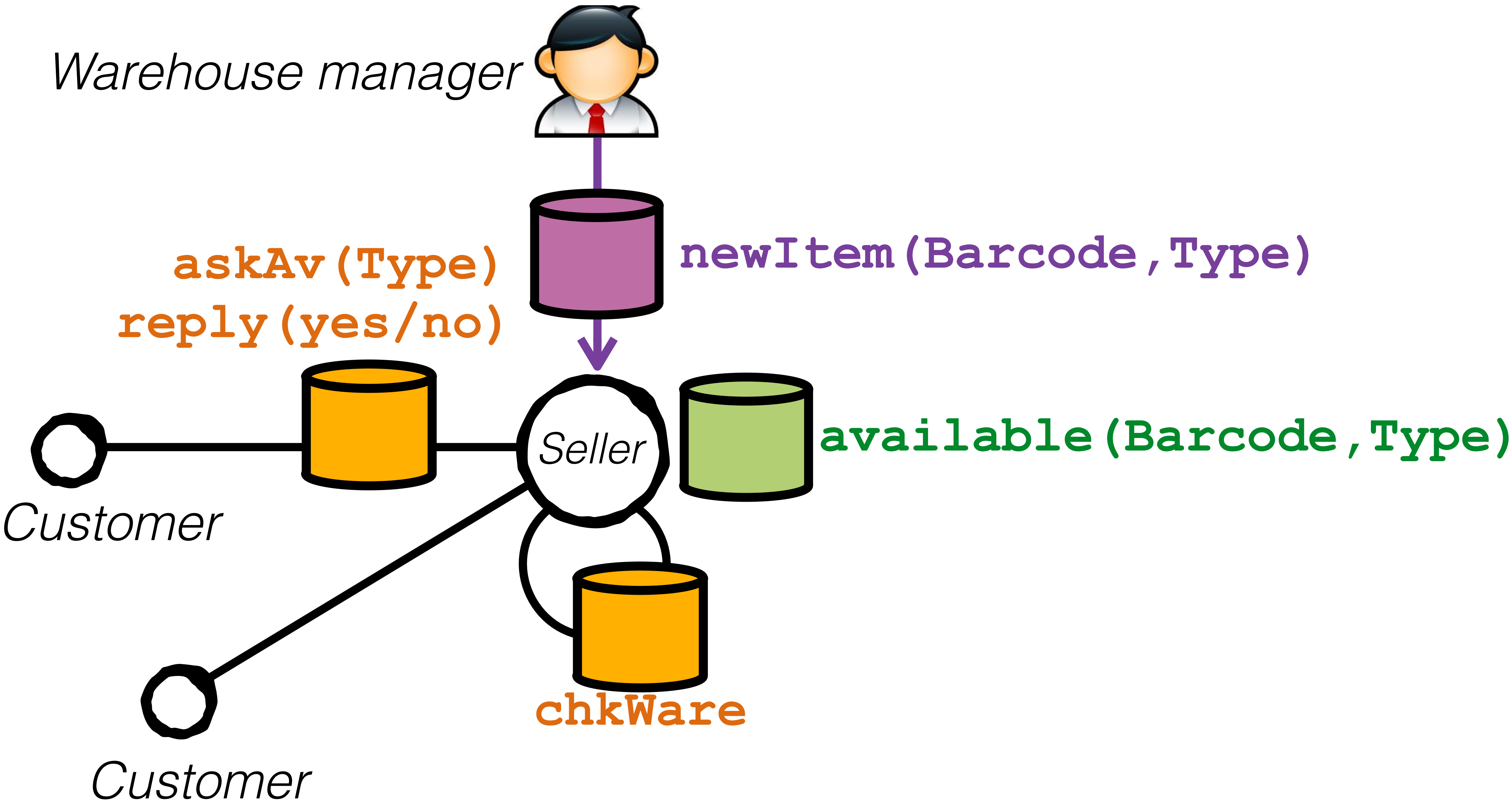
- If you have just joined the tree, flood the join request to neighbors (the parent will ignore it):

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

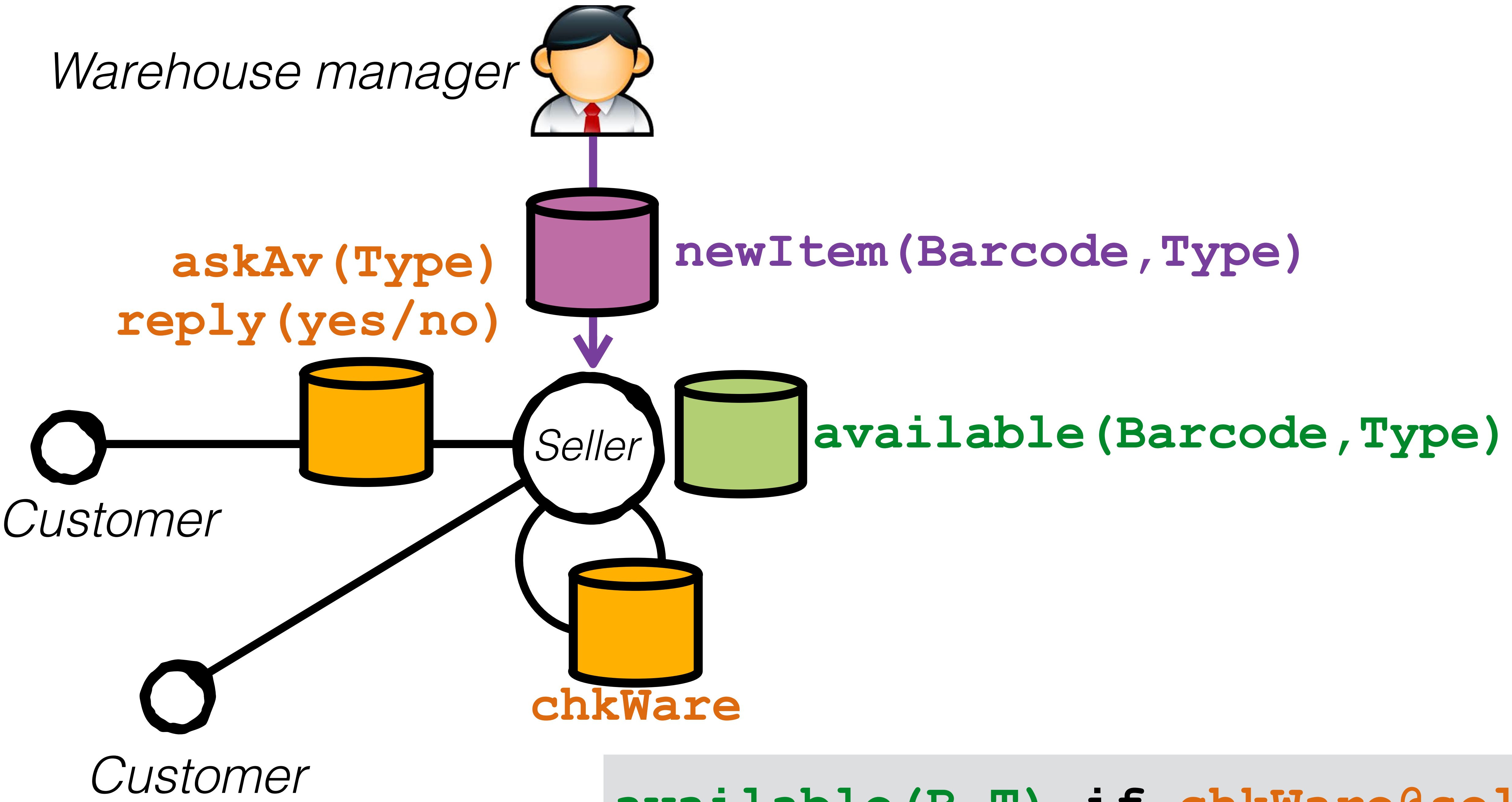
- Parent information is kept:

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

Another Example

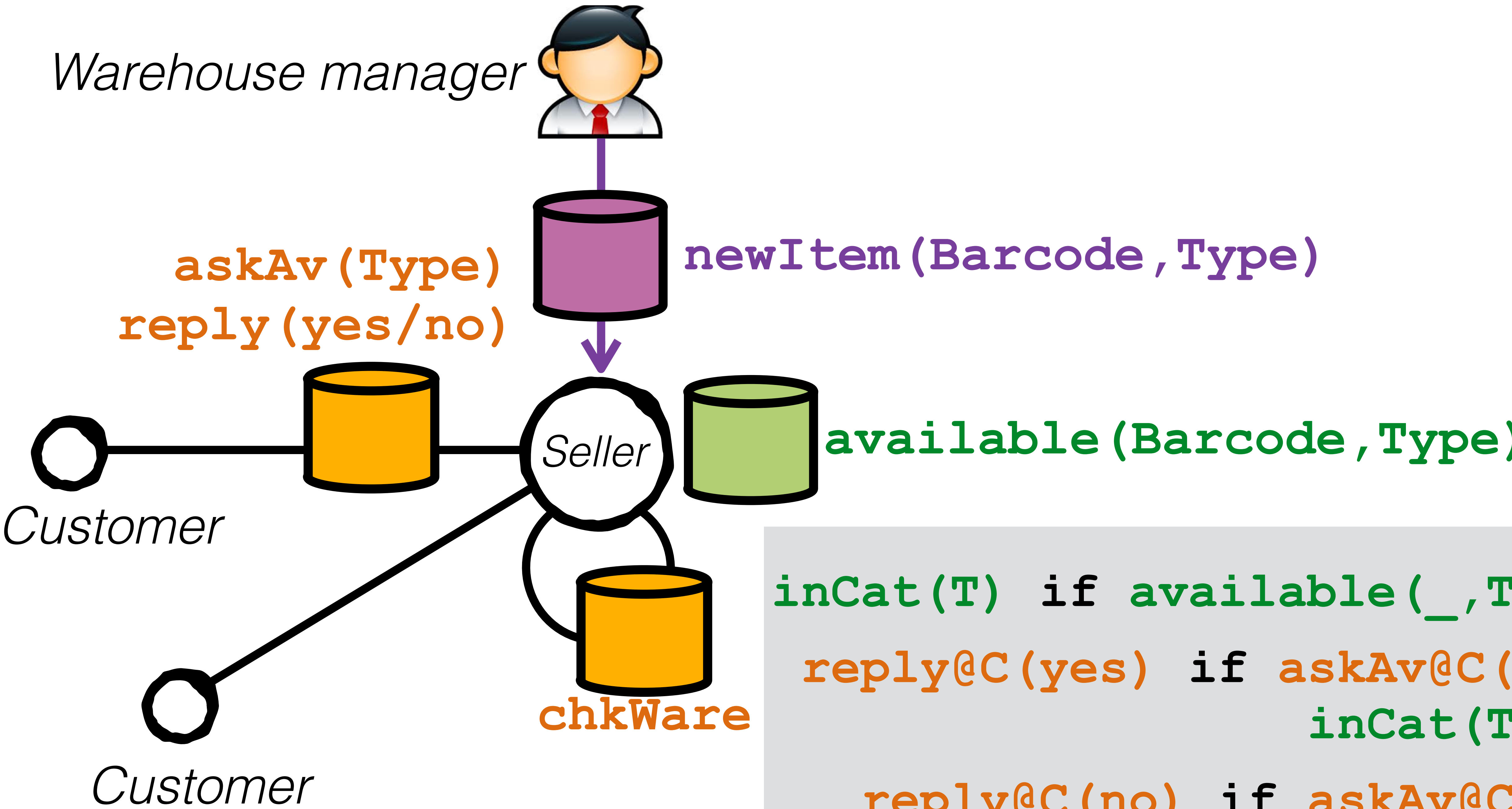


Another Example




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

Another Example




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

Interesting Questions

Domain-specific properties: CTL-FO or **LTL-FO** with **active domain quantification**

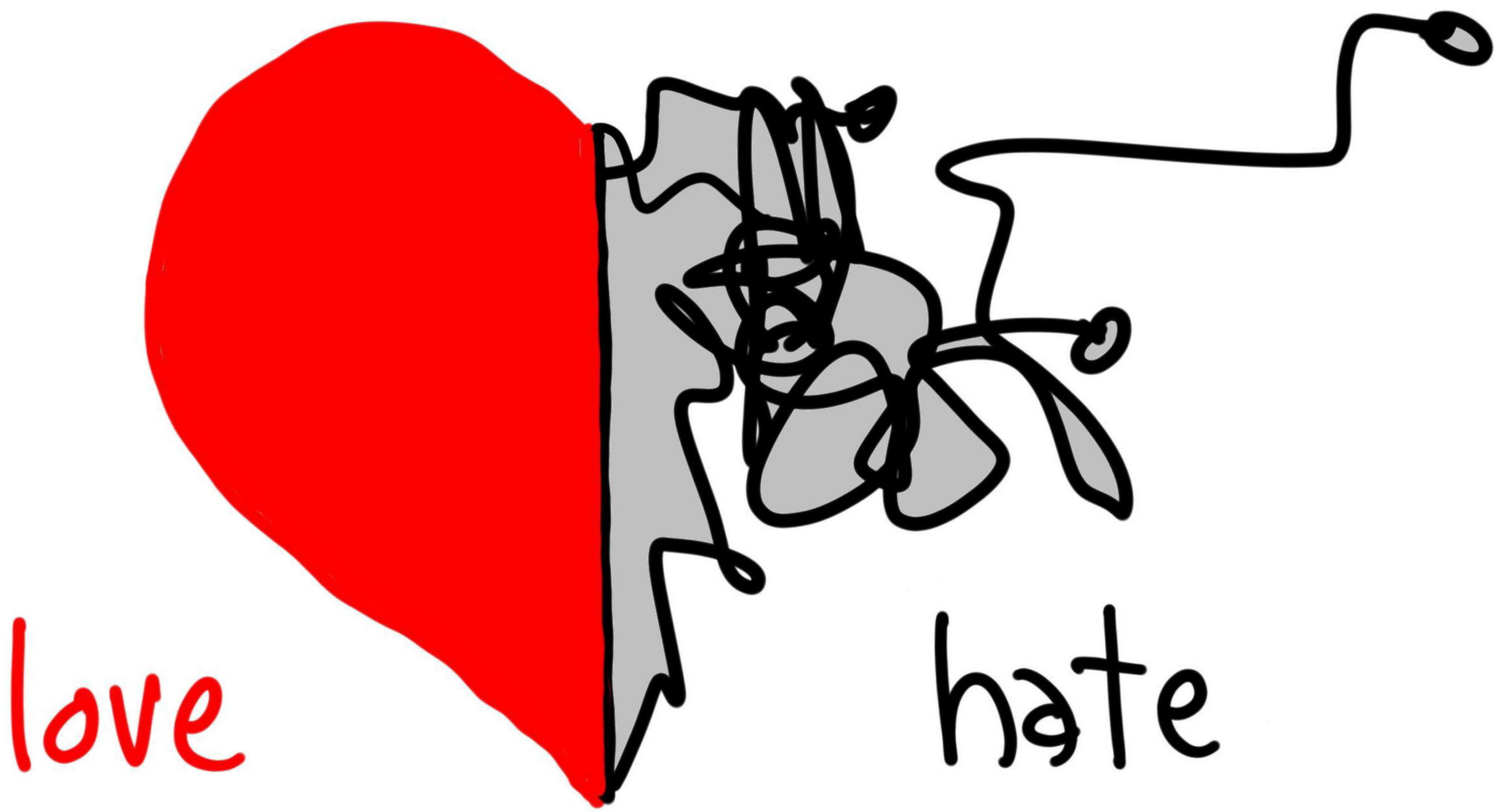
- Maintain: $\mathbf{G}(\forall n, p. Parent@n(p) \rightarrow \mathbf{G} Parent@n(p))$
- Broadcast: $\mathbf{G}(\forall x.(\exists n. R@n(\vec{x})) \rightarrow \mathbf{F} \forall n'. R@n'(\vec{x}))$

Generic properties: **convergence**

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

Act 3

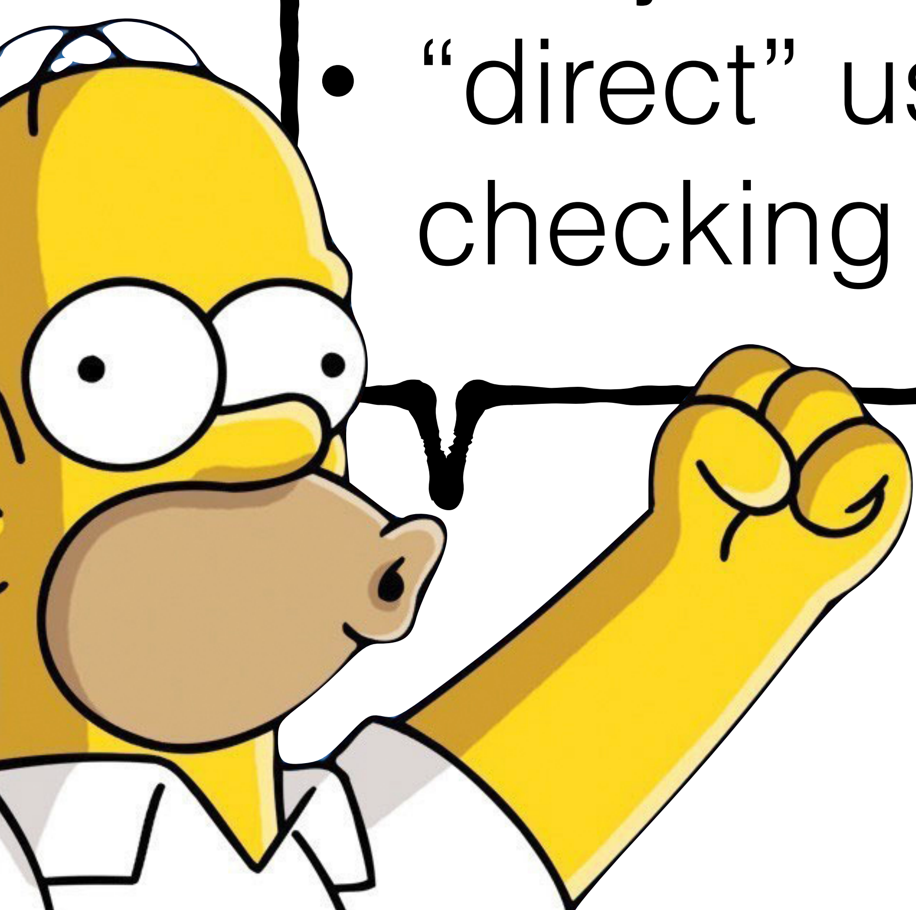
Hate and Love



Closed DDS: the "Easy" Case

No injection of data from the external world:

- system inherently finite-state
- FO: just a nice "surface syntax"
- "direct" usage of conventional model checking techniques



Closed DDS: the “Easy” Case

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. Compute a corridor tiling problem

Interactive DDS: the Hard Case

A node is computing machine with a finite-state control process and an unbounded memory.
So what is it?

Interactive DDS: the Hard Case

A node is computing machine
with a finite-state control process
and an unbounded memory.
So what is it?

A Turing machine
I.e., You are doomed to
undecidability, even for
propositional reachability!

Size-Boundedness

Intuition: put a pre-defined bound on the DB size

- Extensively studied over the last years - cf. ACSI project (under the name of “state-boundedness”)
- In general, the resulting transition system is still infinite-state (even when all relations are 1-bounded)
- *In DDS we can selectively bound state, transport, input!*

Does Size-Boundedness Help?

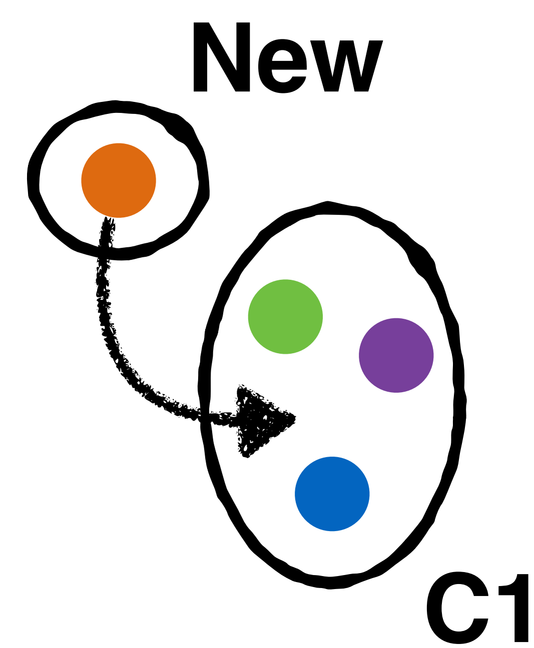
Interactive DDS, linear-time case

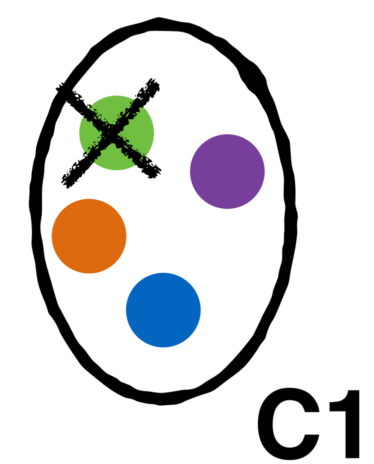
input bounded	state/transport bounded		
	N/Y	**Y/N**	**Y/Y**
N	convergence **undecidable**		model checking FO-LTL **undecidable**
Y			

Reasons for Undecidability (State Unbounded)

Simulation of a 2-counter Minsky machine

- Single node with 2 unary relations **C1** and **C2**
- 1-bounded, single unary input relation **New**
- Increment counter1:
  - check whether **New** contains an object not in **C1**
  - if not, enter into an error state
  - if so, *insert* it in **C1**
- Decrement counter1: pick an object in **C1** and *remove* it
- Test counter1 for zero: check that **C1** *is empty*





Reasons for Undecidability (State/Transport/Input Bounded)

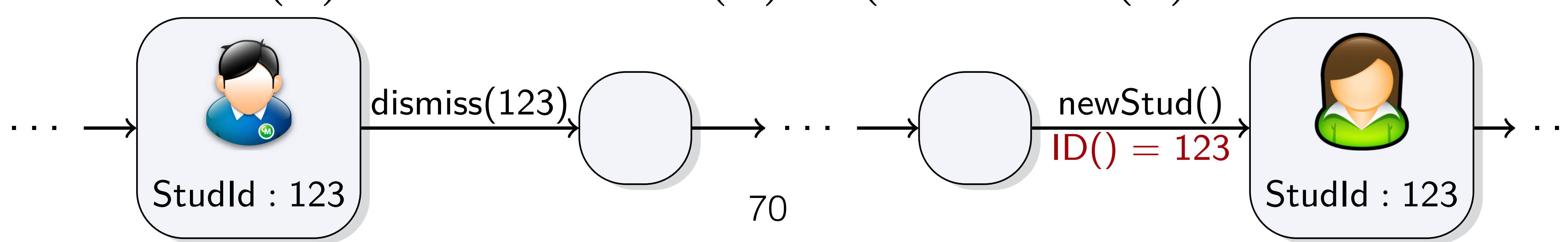
- Take a DDS with:
  - a single node
  - two unary, 1-bounded relations: one for input, one for state
  - a D2C program that just overwrites the state with the input
- It generates all *infinite data words* over the infinite data domain
- Satisfiability of LTL with freeze quantifier is undecidable [Demri and Lazic, 2006], and can be encoded as FO-LTL model checking over this DDS

- Undecidability comes from the extreme power of FO quantification across snapshots: *variables can store unbounded information!*

FO-LTL with Persistent Quantification

- Intuition: control the ability of the logic to quantify across snapshots
- Only **objects that persist** in the active domain of some node can be tracked
- When an object is lost, the formula *trivializes* to *true* or *false*
- E.g.: "guarded" until

$$\mathbf{G}(\forall s.Student(s) \rightarrow Student(s)\mathbf{U}(Retired(s) \vee Graduated(s)))$$



Size-Boundedness to the Rescue

**Interactive DDS, linear-time case
with persistent quantification**

input bounded	state/transport bounded		
	N/Y	Y/N	Y/Y
N	convergence **undecidable**		model checking FO-LTL with persistence **PSPACE-complete**
Y			

DDS Key Properties

DDS (and other similar data-aware dynamic systems) enjoy two key properties: 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 which are identical modulo uniform renaming of data objects.

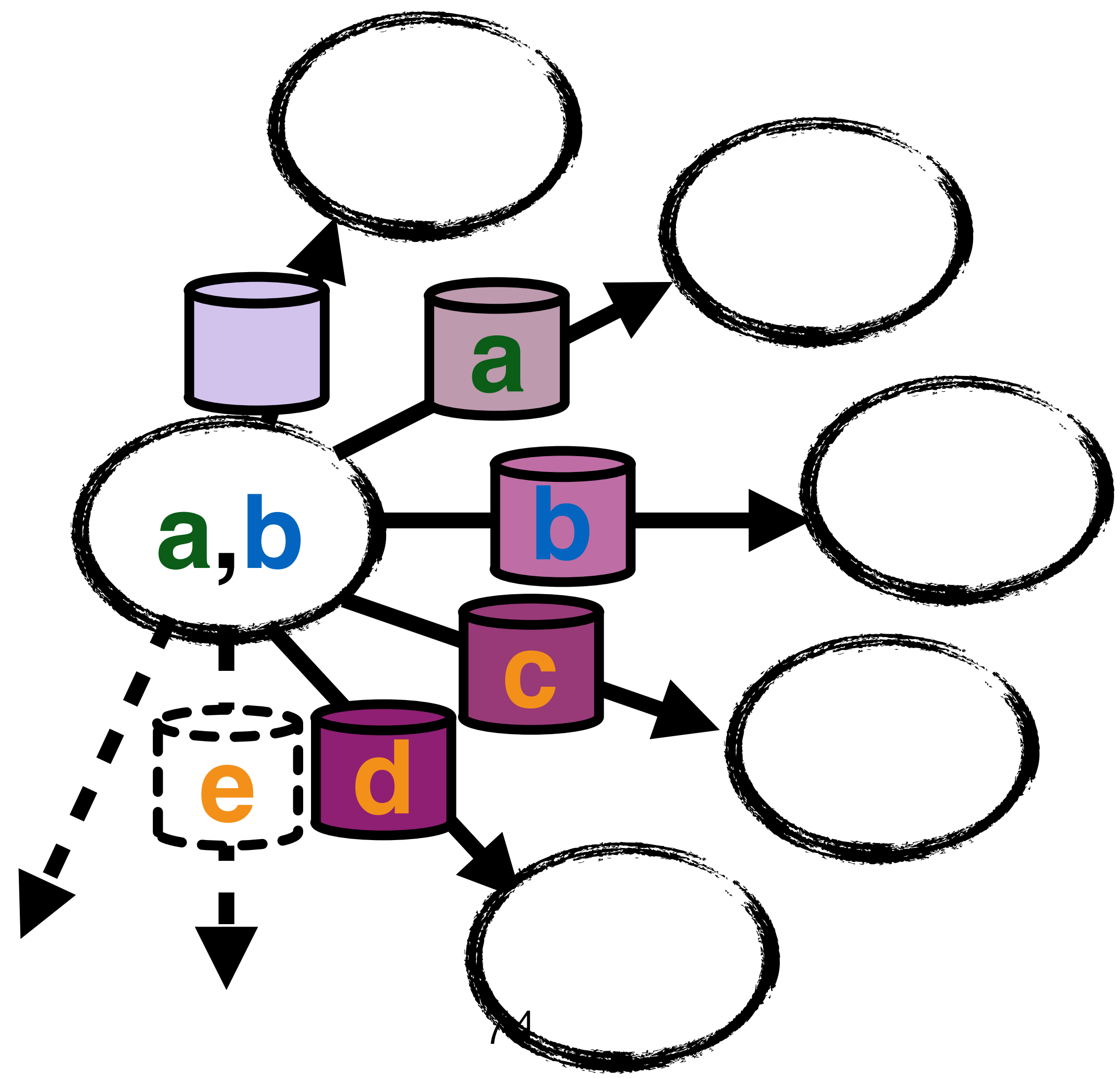
—> Two isomorphic DDS snapshots are bisimilar

Pruning 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 DDS active domain
- 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**

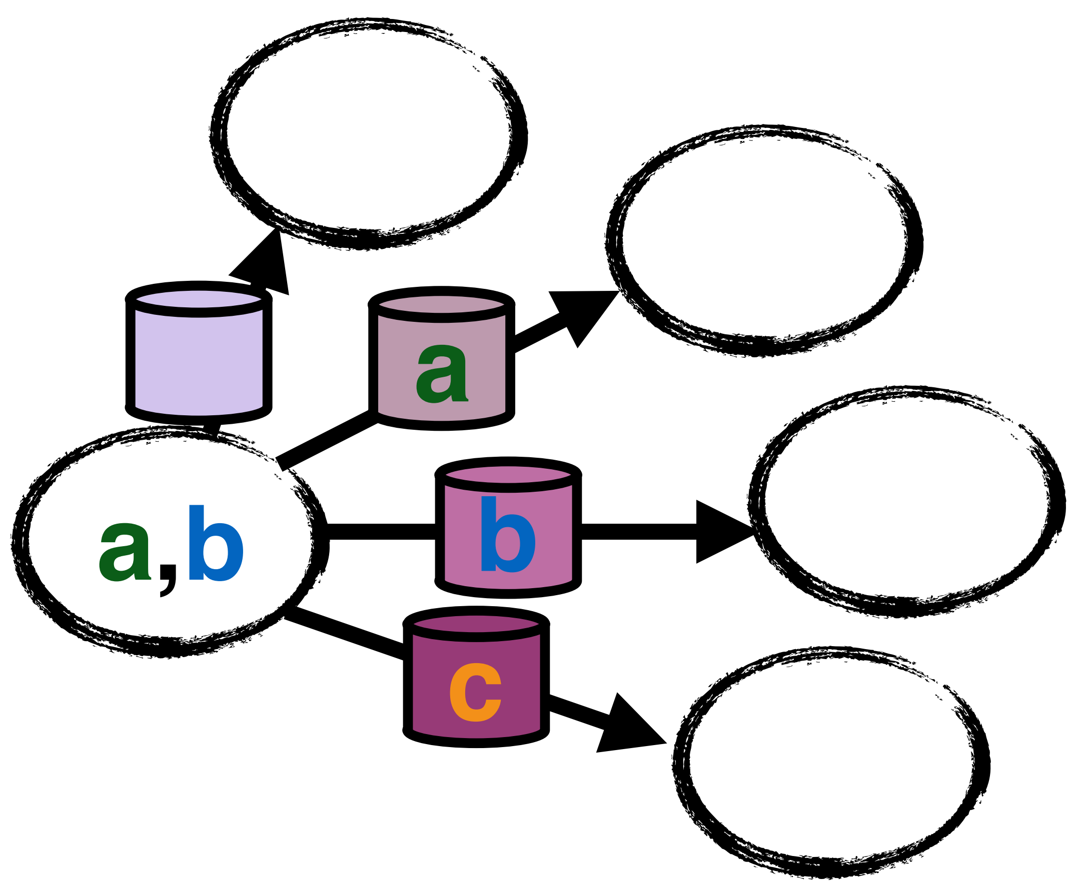
Example

- Input: single unary relation, 1-bounded
- Current state: two objects



Example

- Input: single unary relation, 1-bounded
- Current state: two objects



Compacting Infinite Runs

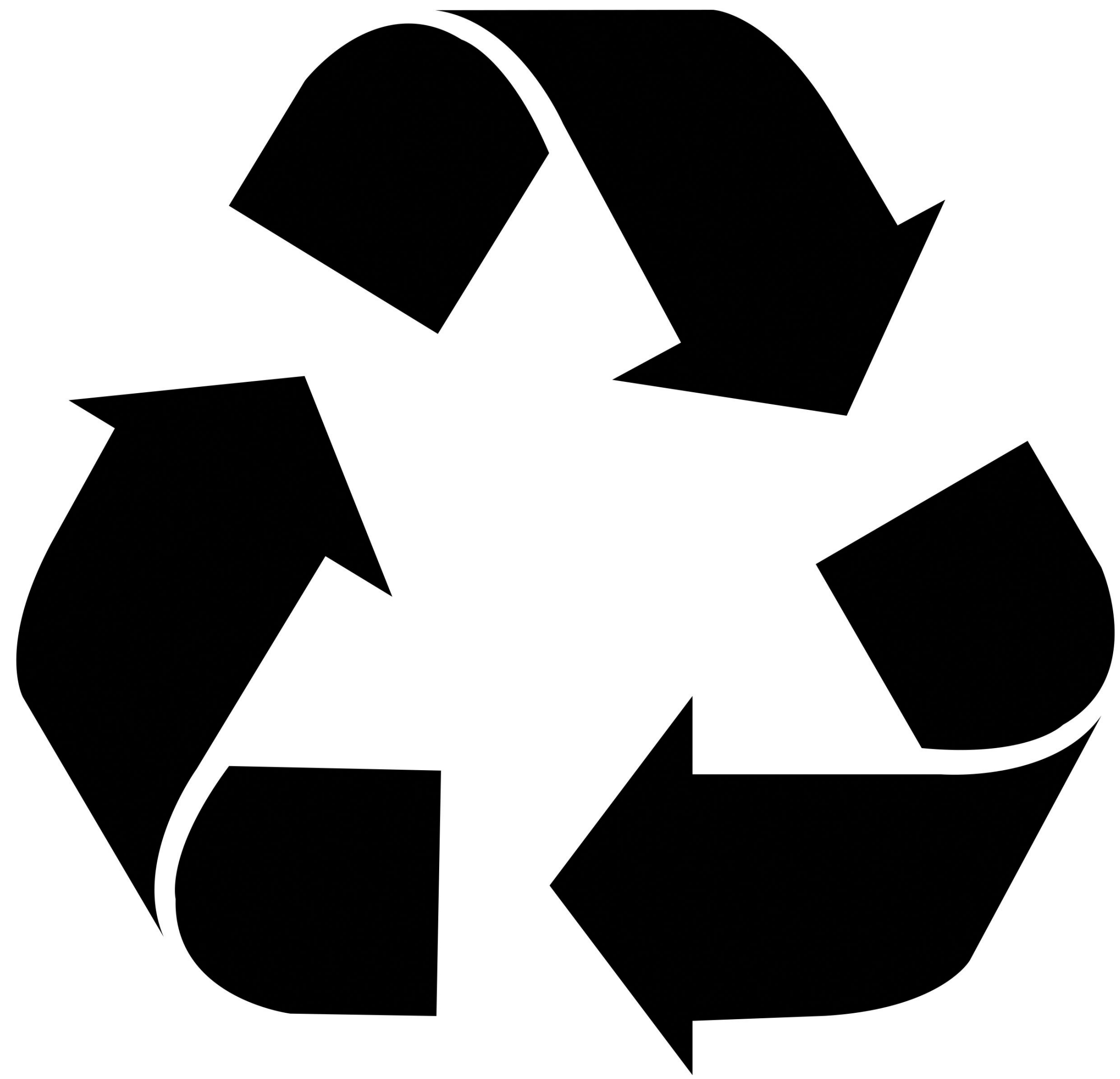
- Key observation: due to persistent quantification, the logic is **unable to distinguish local freshness from global freshness**
- So we modify the transition system construction: whenever we need to consider a fresh representative object…
    - … if there is an old object that can be recycled —> use that one
    - … if not —> pick a globally fresh object
- **This recycling technique preserves bisimulation!**

Compacting Infinite Runs

- [Calvanese et al, 2013]: if the system is size-bounded, the recycling technique reaches a point were no new objects are needed **—> finite-state transition system**
- N.B.: *the technique does not need to know the value of the bound*

Recap

Prune



Recycle

Recap

- Input: interactive DDS whose node DBs are all size-bounded
- Construct the abstract transition system that works over isomorphic types and recycles old objects
- The abstract transition system is
  - finite-state
  - a faithful representation of the original one
- Use the abstract system to model check "persistent" FO-LTL formulae using conventional techniques (PSPACE upper bound)

Conclusion

Marriage between processes and data is challenging, though necessary

- *Size-boundedness is a robust condition* towards the effective verifiability of such systems
    - The same results hold in by enriching the data component (ontologies, constraints, inconsistency-tolerance, …)

- *Same* formal model for execution and verification

Current and Future Work

- Implementations, leveraging the long-standing literature in data management and formal verification
- Emphasis on other reasoning services: monitoring, planning, adversarial synthesis
- Relaxations of size-boundedness, with the help of
  - Parameterized verification
  - Verification via underapproximation
  - Conceptual conditions that hold in practice

Acknowledgments

All coauthors of this research,
in particular

Diego Calvanese
Giuseppe De Giacomo
Alin Deutsch
Jorge Lobo
Fabio Patrizi

Acknowledgments

AI*IA

The AI*IA “2015 Somalvico Award” Committee

The external supporters of my nomination:
Wil van der Aalst
Thomas Eiter
Munindar Singh

Acknowledgments

Paola Mello
Diego Calvanese

The AI group @ DISI-UNIBO
The KRDB Group @ UNIBZ

My colleagues in
Ferrara, Rome, Eindhoven, Tartu, Uppsala

Acknowledgments

My
(unbounded)
family