Choreography Revisited

Jianwen Su
University of California at Santa Barbara

Computing

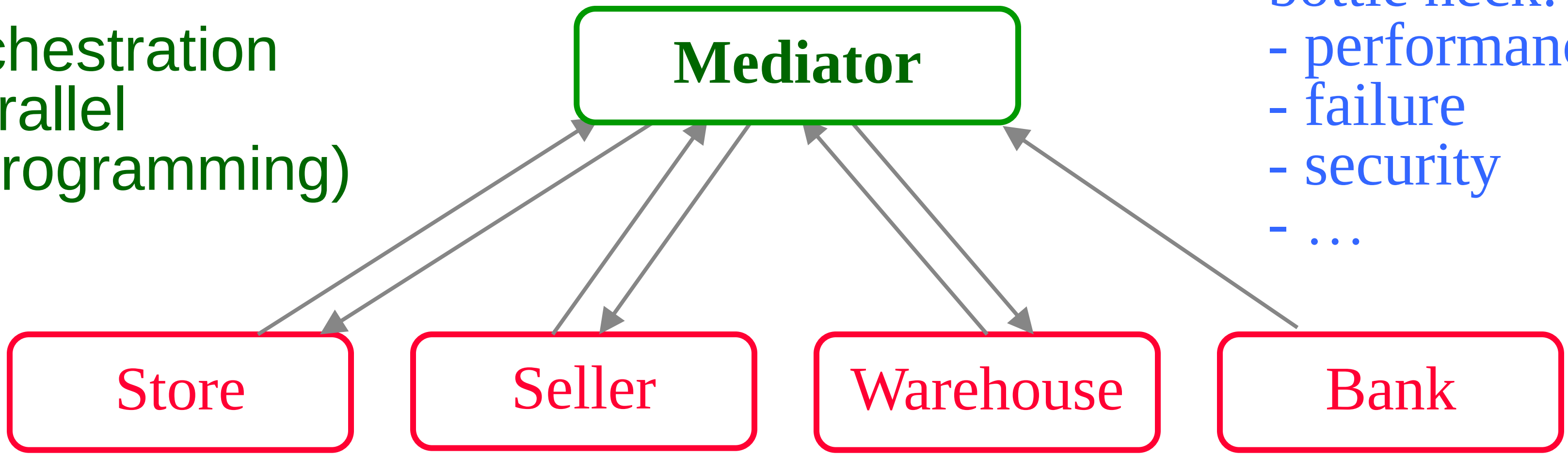
- Foundation for Science, Technology, Engineering
  - Modeling & abstraction
  - Algorithmic thinking
- This talk concerns business processes
  - Retail industry, legal & government, health care, …

  BPs could be helped by CS in
  - Management of data and processes
  - Techniques for modeling & design, automation
  - Business informatics (as a new sector?)
- Focus: collaboration between business processes

Collaboration Models (Extreme Cases)

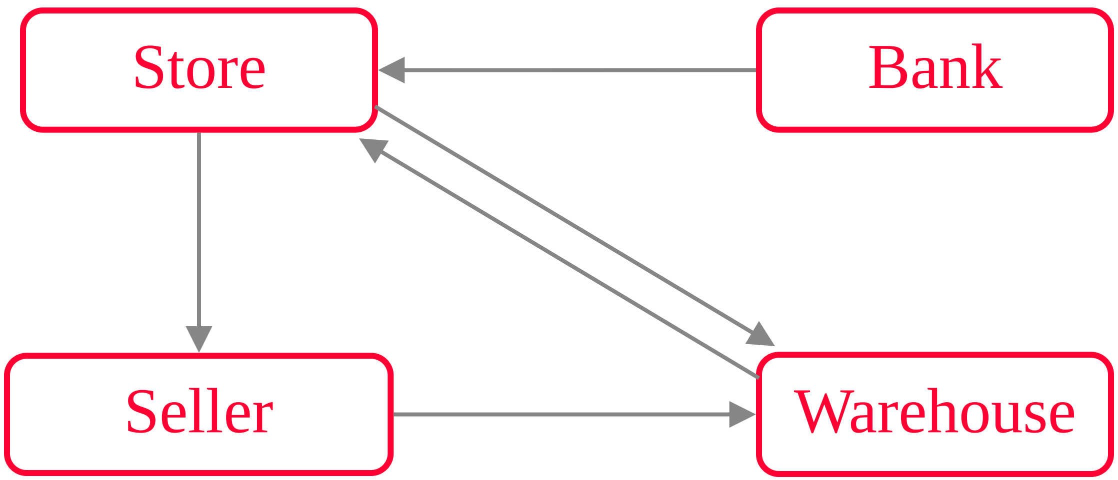
- Hub-and-spoke or mediated

Orchestration (parallel programming)



✓ global state
✗ bottle neck:
- performance
- failure
- security
- …

- Peer to peer (communicate when needed)

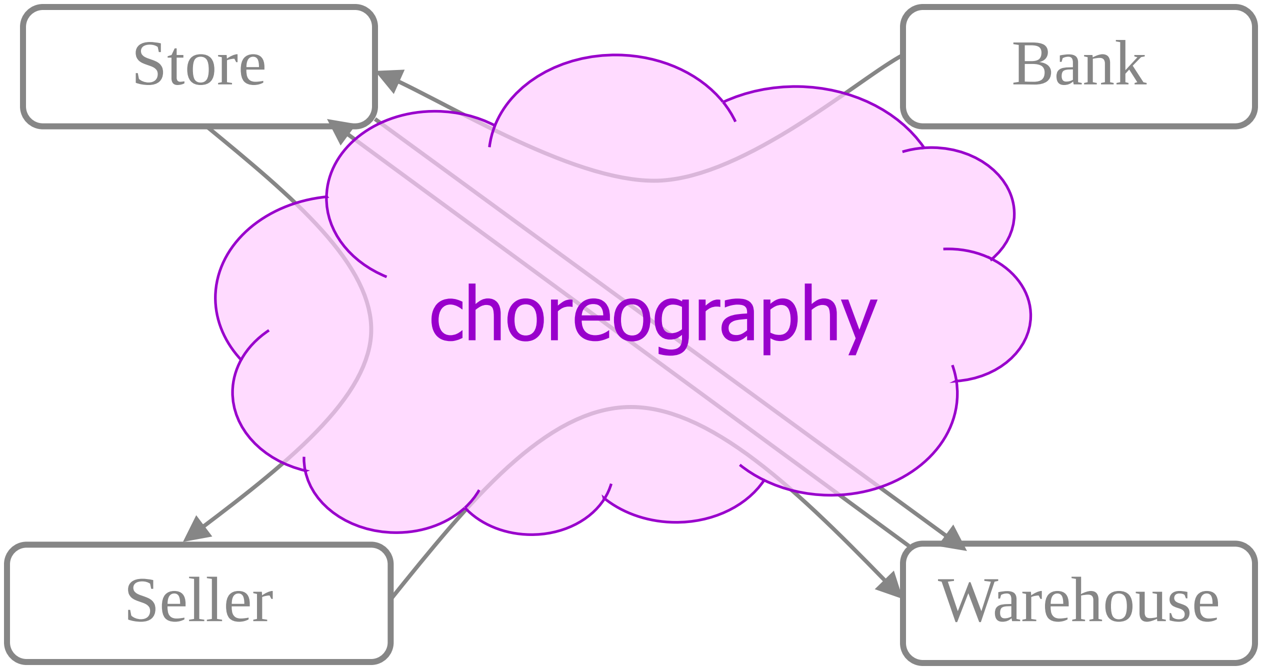


✓ no bottle neck
✗ global state

CS can contribute here

Choreography

- A choreography defines how biz processes should collaborate to achieve a business goal

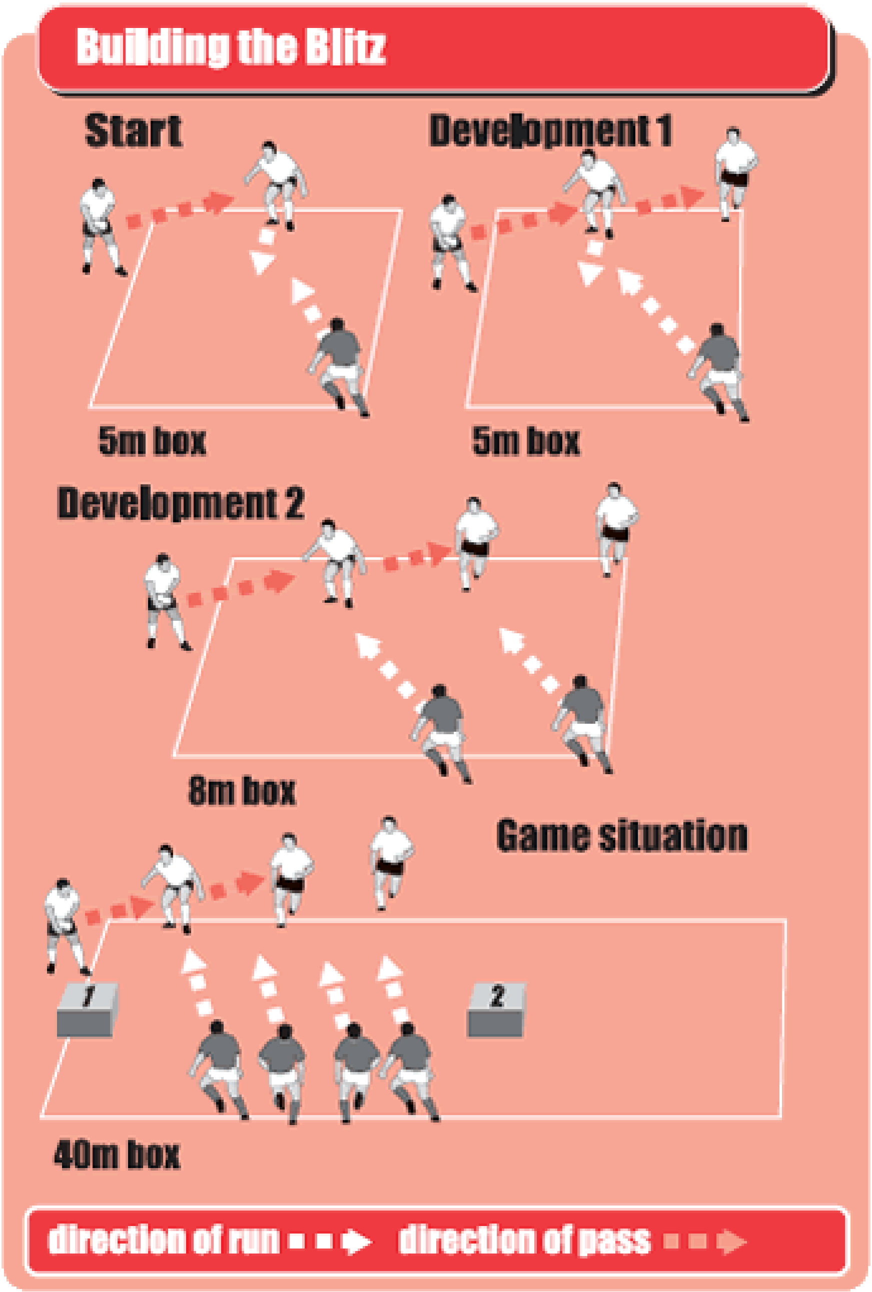


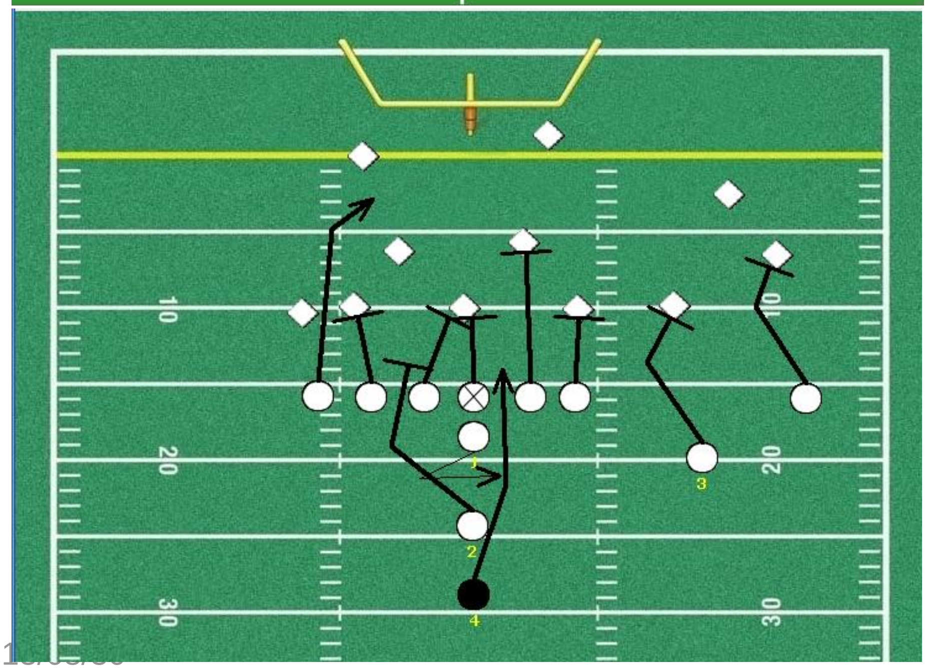
- Goal: Support for choreography languages:
  - Design "correctness", auto realization, mechanisms for monitoring, …

Outline

- Choreography & biz processes
- Key Aspects of choreography specification
  - Weaknesses of existing choreography languages
- Ingredients of our approach
  - Artifacts as biz processes
  - Correlations
  - Message diagrams
- Snapshots and temporal (choreography) constraints
- Realization
- Conclusions

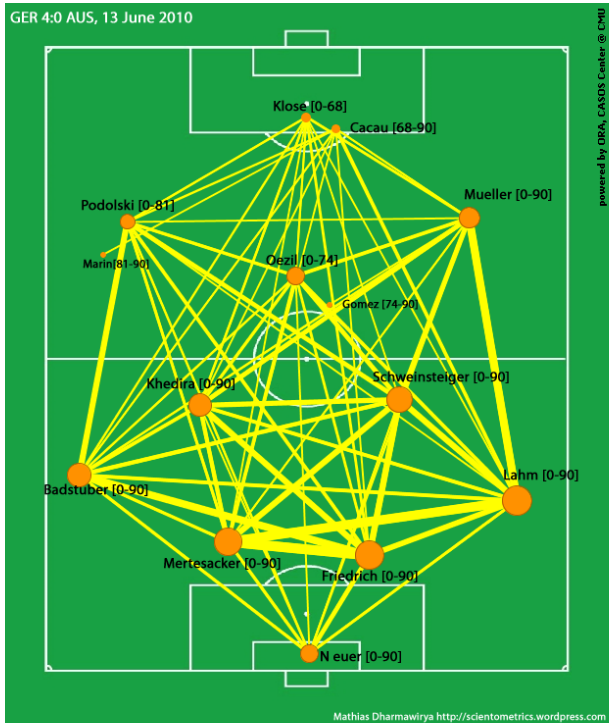
Examples: Choreographies for Soccer





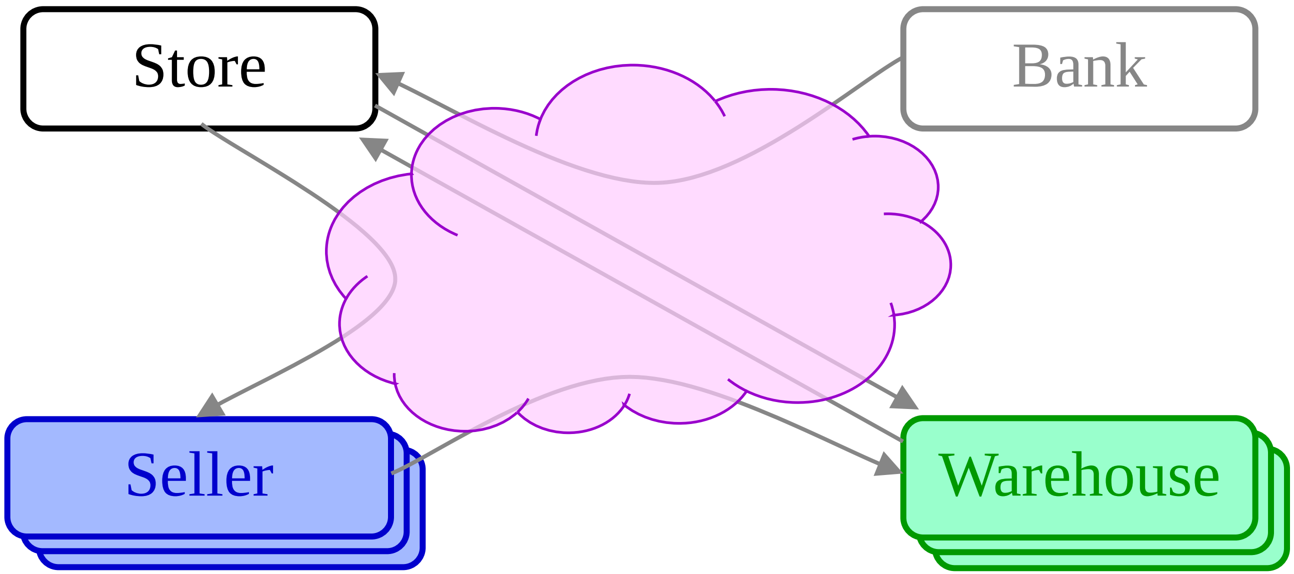
Choreographies for BPs Are More Complex

Views (for Analytics)



Correlation of Process Instances

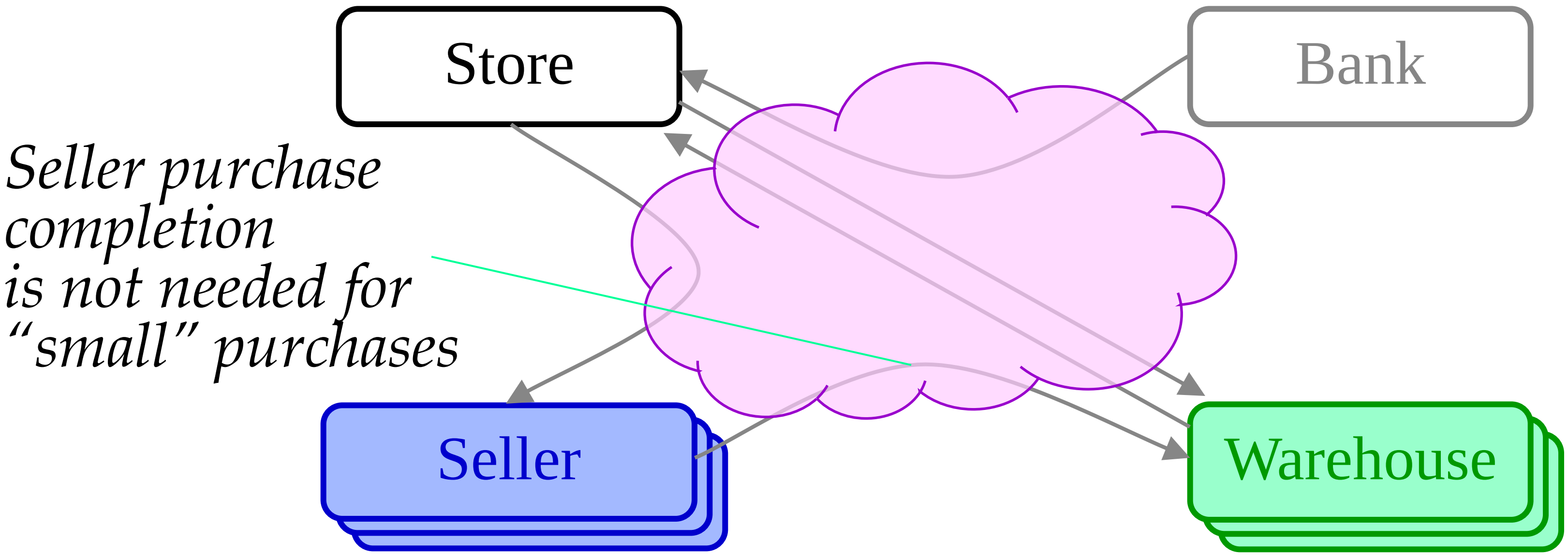
- A choreography should be aware of process instances not just biz process types



- Existing languages? None support such correlations: WS-CDL, BPMN, process algebras, conversation protocols, Let's Dance, (BPEL,) …

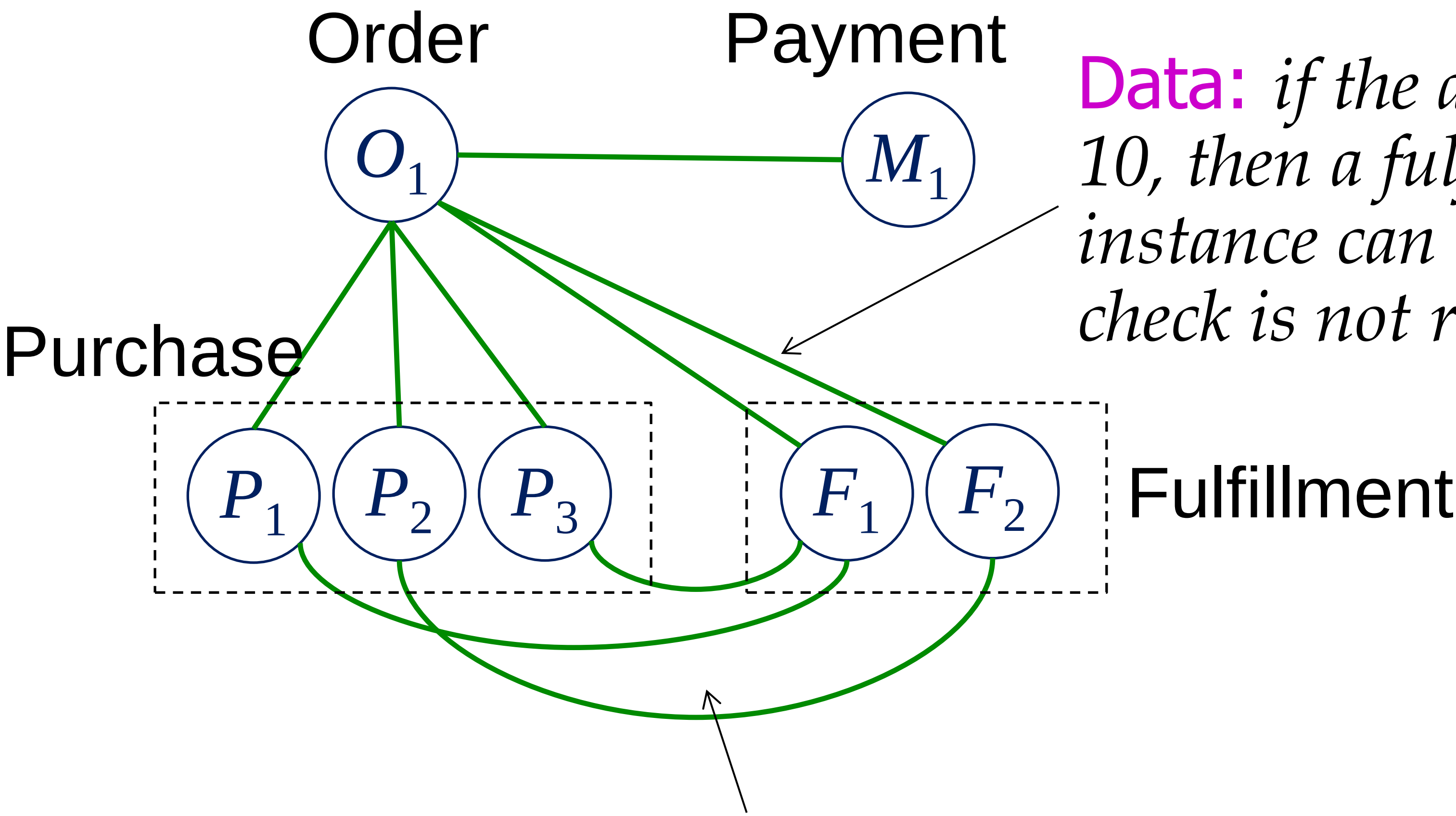
Data in Messages and Process Instances

- Choreography constraints may depend on message contents and data from process instances



- Most choreography languages support no data, or no general models for data

What are Needed?



Data: *if the amount is less 10, then a fulfillment instance can ship even the check is not received*

Instance-level correlation: *Which instances are correlated during the runtime? Who sends messages to whom?*

Existing Choreography Languages

	Instance correlation	Schema correlation	Data
Conversation model [Fu et al 2004]	no	yes	no
WS-CDL [W3C 2005]	no	yes	message variables*
Let's Dance [Zaha et al 2006]	no	yes	no
BPEL4Chor [Decker et al 2007]	1-to-m only	yes	message variables*
Artifact-centric choreography [Lohmann-Wolf 2010]	no	yes	no
Our model	yes	yes	yes

*no clear linkage between variables and processes

Outline

- Choreography & Biz Processes
- Key Aspects of Choreography Specification
  - Weaknesses of existing choreography languages
- Ingredients of Our Approach
  - Artifacts as Biz Processes *who*
  - Correlations *to whom*
  - Message Diagrams *sends what* *at what time*
- Snapshots and Temporal (Choreography) Constraints
- Realization
- Conclusions

Four Types of Data in Biz Processes

- Essential business data for the process logic: items, shipping addresses, ...
- Current execution or enactment states: order sent, shipping request made, ...
- Resource usage and states: cargo space reserved, truck schedule to be determined, ...
- Correlation between processes instances: 3 warehouse fulfillment process instances for a customer order instance, ...

- All data should be persistent (maintained properly)
- Traditional biz process modeling languages are weak in modeling related data

BP Models: Data Abstraction to Artifacts

Four classes of Biz process models:

- Data agnostic models: data mostly not present
  - WF nets (Petri nets), BPMN, …
- Data-aware models: data (variables) present, but storage and management hidden
  - BPEL, YAWL, …
- Storage-aware models: schemas for persistent stores, data mappings to/from BPs defined/managed manually
  - jBPM, …
- Artifact-centric models: logical modeling for biz data, automated: modeling other 3 types, data-storage mapping
  - GSM, EZ-Flow

Artifacts As Process Models

- Should support: instances, process contents, messages
- Artifact class or interface, data attributes, attribute types may be relational or other artifact classes

Store: *Order*

ID	C*Name*	*Addr*	Cart				...
			INo	*Seller*	WH*ouse*	*Price*	

Bank: *Payment*

ID	*Ord*ID	...

Seller: *Purchase*

ID	...

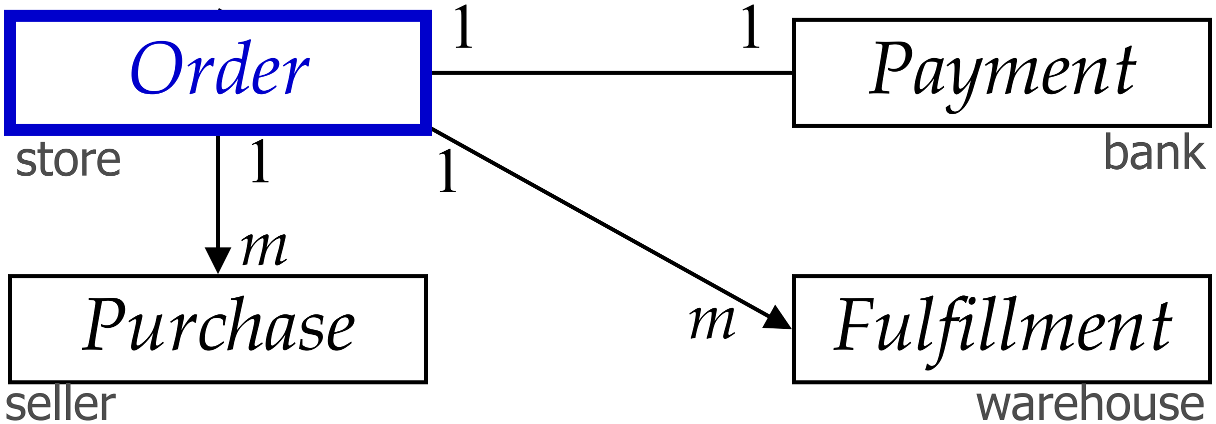
Warehouse: *Fulfillment*

ID	...

Lifecycle specifications not shown

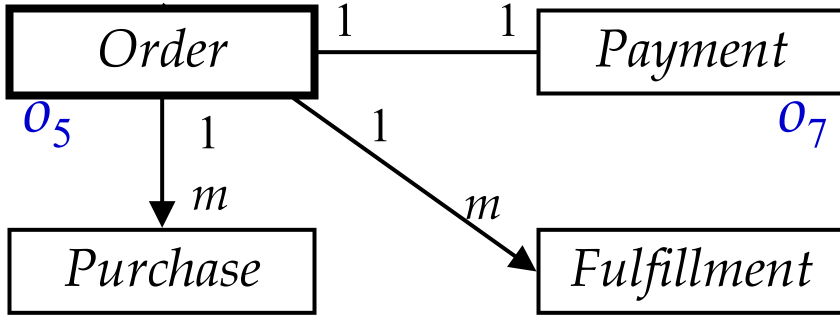
Correlation Diagrams

- Two process instances are correlated if they are involved in a common collaborative BP instance
  - *Messaging only between correlated instances*
- Correlations of a CBP are defined in a diagram, with one BP as the root or primary process



  - Directed edge indicates creation of BP instance(s)
  - Cardinality constraints are also defined
  - Some syntactic restrictions (acyclic, “1” on root, …)

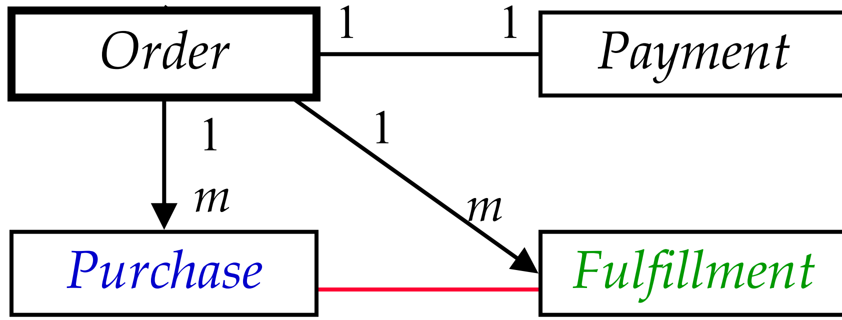
Referencing Correlated BP Instances



- Skolem notations reference correlated instances
- *Fulfillment*$\langle o_5 \rangle$ is the set of all *Fulfillment* instances IDs that are correlated to an *Order* instance with ID o_5

  Order$\langle o_7 \rangle$ is the *Order* instance correlated to a *Payment* instance with ID o_7
- Path expressions used to access contents of artifact attributes, o_5.Cart.Seller denotes all sellers of items in the cart of order o_5

Derived Correlations



- A *Purchase* instance and a *Fulfillment* instance is correlated if both correlated to the same *Order* instance and share at least one item

$$\text{CORRELATE } (\mathit{Purchase}, \mathit{Fulfillment}) \text{ if}$$
$$\mathit{Order}\langle \mathit{Purchase} \rangle = \mathit{Order}\langle \mathit{Fulfillment} \rangle \wedge$$
$$\mathit{Purchase}.\text{Items.INo} \sqcap \mathit{Fulfillment}.\text{Items.INo}$$

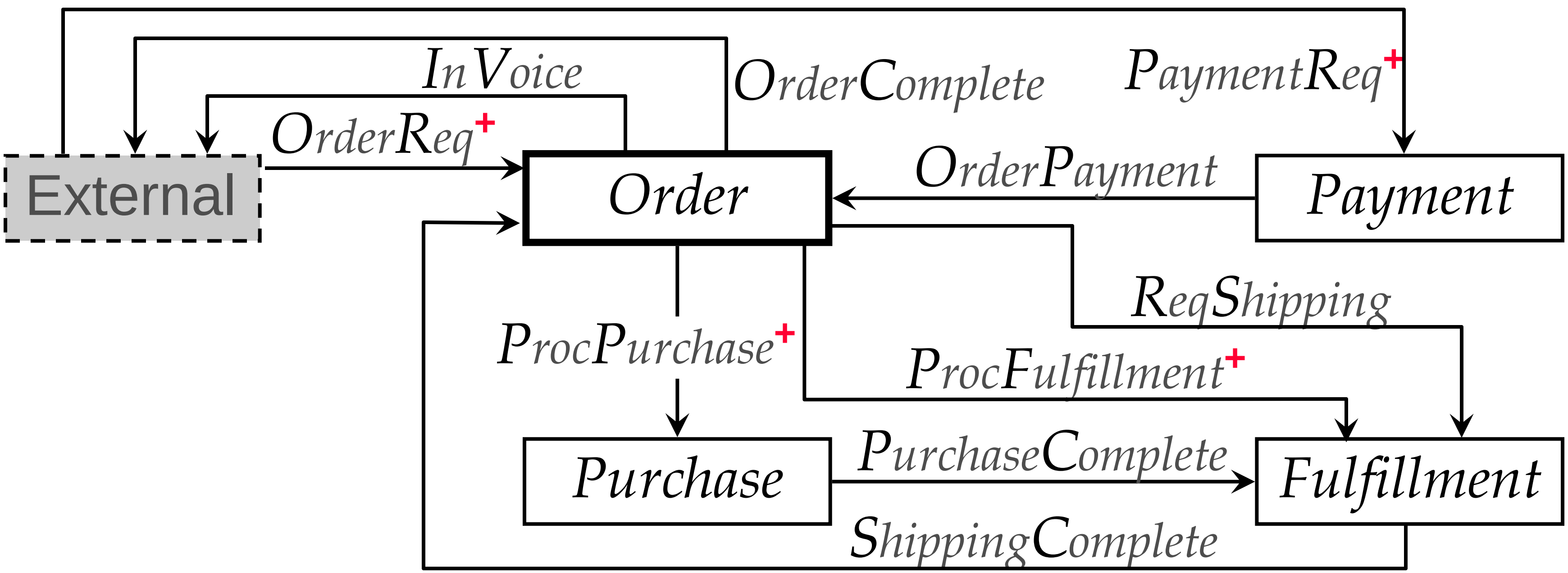
- Derived correlations have no cardinality constraints specified, nor instance creation

"Managing" Correlations

- Correlations are generated at runtime
- Some correlations are generated within collaborative BP execution, e.g., creating *Fulfillments* by *Order*
- Some correlations are obtained through external means, e.g., *Payment* & *Order*
- Need to know messaging "patterns"

- Runtime management of BP instance correlations using Petri nets: [Zhao-Liu CAiSE 07]

Messages Diagrams



- A message diagram defines message types and sender/receiver of each type
  - "External" denotes the environment
  - "+" means creation of new BP instance
- Message may have data attributes
  - Path expressions are used to access data contents

Outline

- Choreography & Biz Processes
- Key Aspects of Choreography Specification
  - Weaknesses of existing choreography languages
- Ingredients of Our Approach
  - Artifacts as Biz Processes
  - Correlations
  - Message Diagrams
- Snapshots and Temporal (Choreography) Constraints
- Realization
- Conclusions

System Snapshots (States)

- A system snapshot is a triple $(\mathbf{A}, \mathbf{M}, m)$

  $\mathbf{A}$: a set of "active" artifact instances,
  $\mathbf{M}$: a set of messages that are already sent, and
  m : the current message sent

  - Note that data contents are included

- Also "tracked":
  - Artifact instance correlations
  - Message-artifact dependencies
    (a message creates an artifact instance)
  - Message-message dependencies
    (a message replies to the previous message)

Message Predicates and Data Atoms

- Message predicates: $M(\mu, a, b)$
  - M: message type, μ message instance ID, a, b: ID of artifact instance (sender, receiver)
- With a data atom:

  $ProcPurchase(\mu, a, b) \wedge \mu.\text{cart.price}>100$
  - data atoms can involve artifacts (e.g., a, b)

Message-message dependencies

- $M[\gamma]$: ID of the message of type M in response to γ
- $M(M[\gamma], a, b)$ abbreviated as $M[\gamma](a, b)$

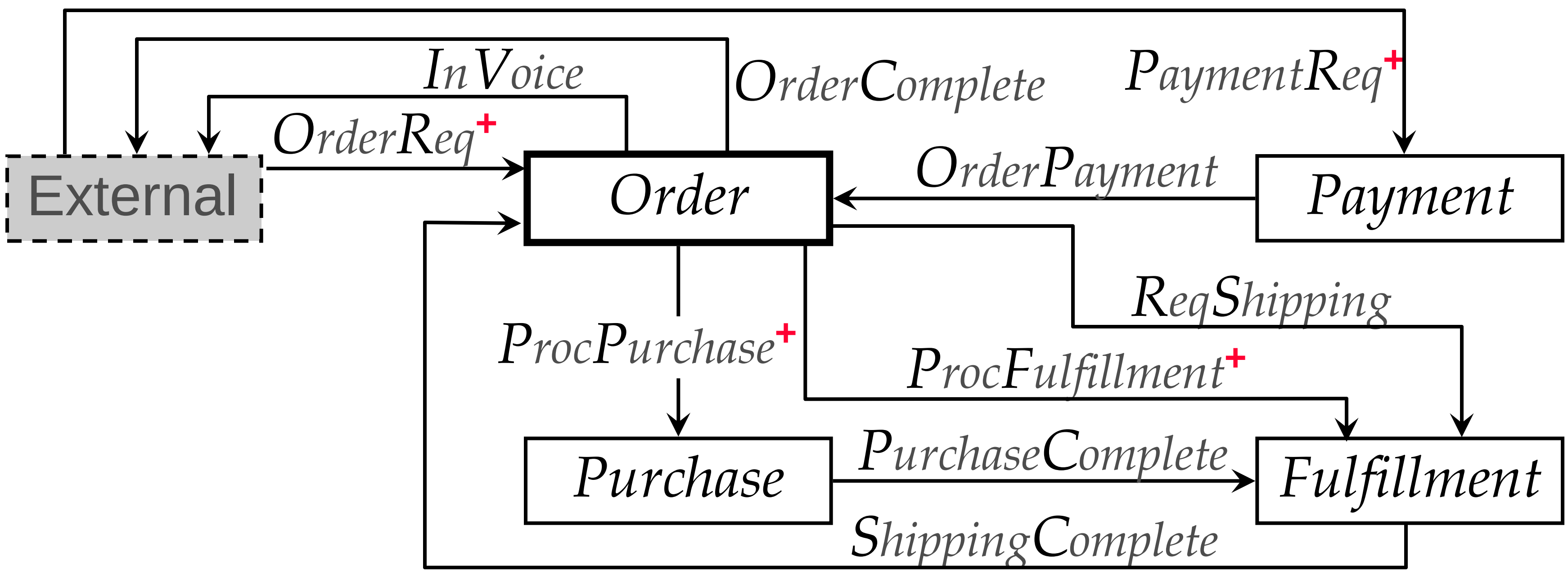
- A snapshot formula: a message predicate with one or more data atoms

Choreography Constraints

- General form: ψ_1 *op* ψ_2
  - ψ_1, ψ_2 : snapshot formulas
  - *op* : binary operators from DecSerFlow: [van der Aalst et al, 2006]

    (co-)exists, SUCCession (resp., prec.), etc. (11 kinds)
- Examples

Messages Diagram for the Example



Choreography Constraints

- General form: ψ_1 *op* ψ_2
  - ψ_1, ψ_2 : snapshot formulas
  - *op* : binary operators from DecSerFlow:
    [van der Aalst et al, 2006]

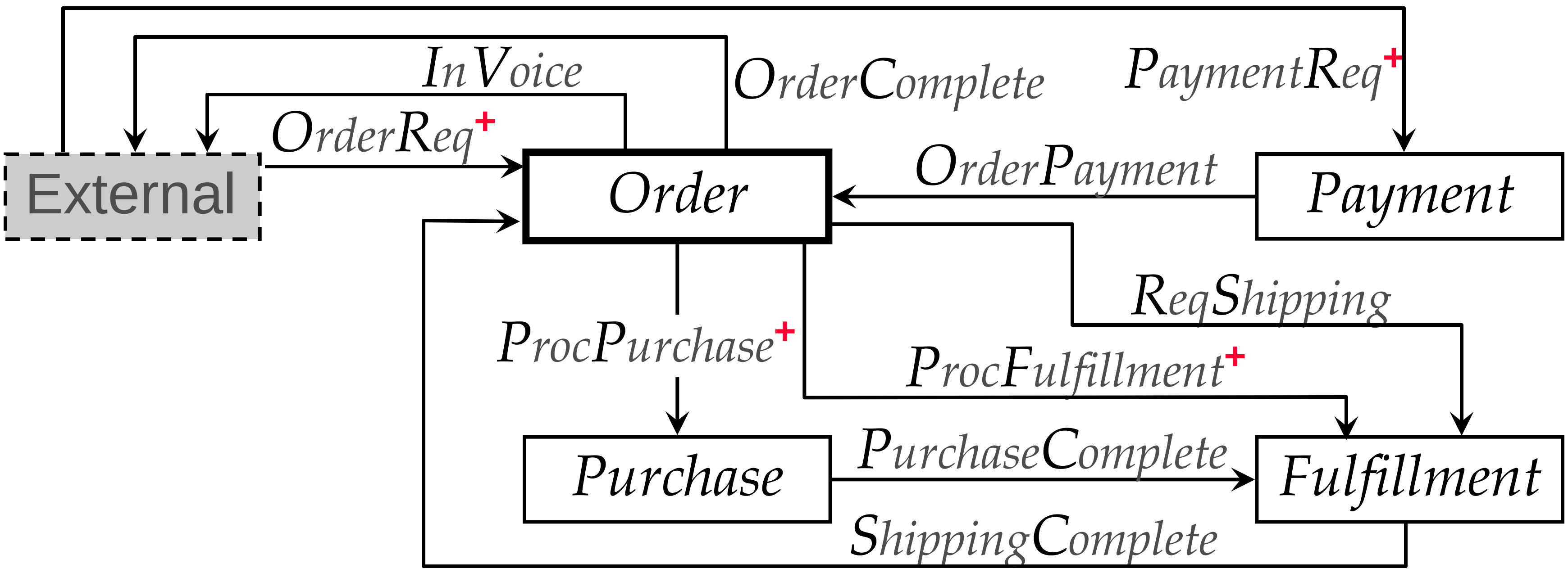
    (co-)exists, SUCCession (resp., prec.), etc.
    (11 kinds)
- An example:

$$OrderReq(\mu,\mathrm{EXT},x) \wedge \mu.\mathrm{amount}>10 \xrightarrow{\mathrm{SUCC}} ProcPurchase[\mu](x,Purchase\langle x\rangle)$$

  Each order request over 10 should be followed by one (or more) processing purchase messages
- Free (artifact/message ID) variables are universally quantified

Another Example

- $\forall x \in Fulfillment\ \forall y \in Purchase\langle x \rangle$
  $PurchaseComplete(\mu, y, x) \wedge y.\text{cart.price} > 100 \xrightarrow{\text{SUCC}} ReqShipping[\mu](Order\langle x \rangle, x)$
- If there is an item priced >100, shipping request is after all purchasing completion



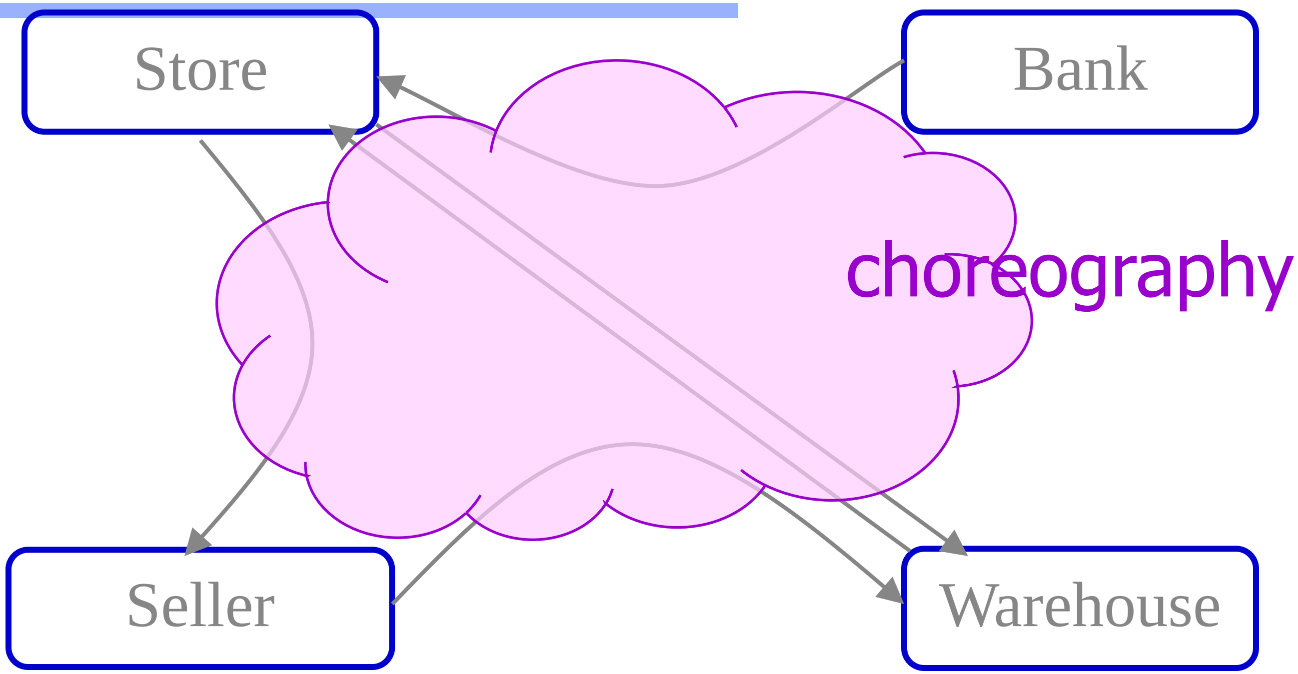
Semantics Based on FO-LTL

- DecSerFlow operators are expressible in Linear-Time Logic (LTL)
- Choreography constraints can be translated to first-order LTL
- Semantics of FO-LTL is based on sequences of system snapshots

Outline

- Choreography & Biz Processes
- Key Aspects of Choreography Specification
  - Weaknesses of existing choreography languages
- Ingredients of Our Approach
  - Artifacts as Biz Processes
  - Correlations
  - Message Diagrams
- Snapshots and Temporal (Choreography) Constraints
- Realization
- Conclusions

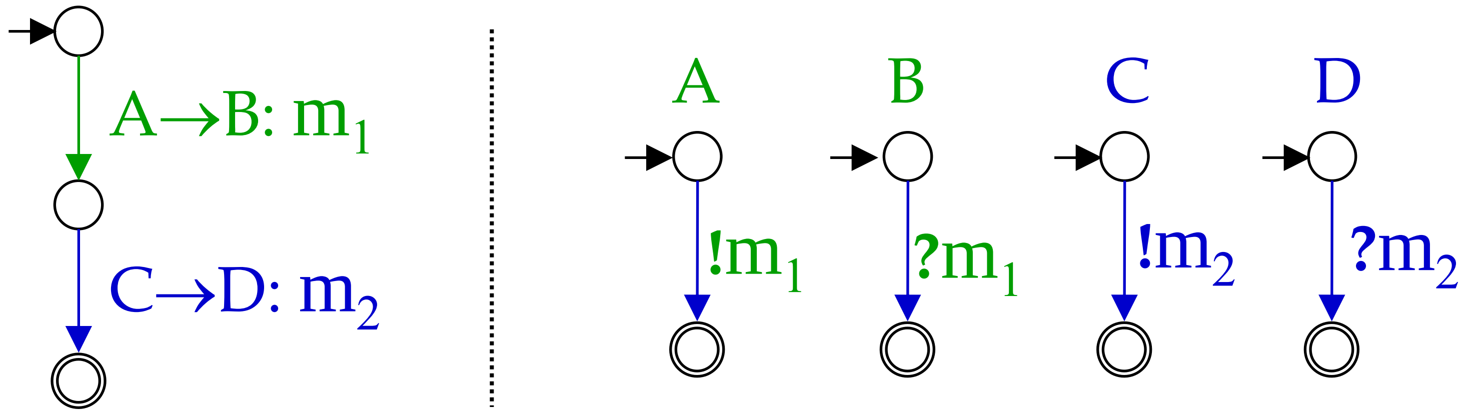
Realization



- A choreography = a set C of snapshot sequences that satisfy constraints
- Executable system = a set E of snapshot sequences that may be produced
- The choreography is realized by the executable system if $C = E$

Choreography Decision Problem

- Problem: Given a choreography, is it realizable?
  - Raised in [Bultan-Fu-Hull-S. WWW 03]
  - Studied in many contexts, especially with process algebras since 2004 [S.-Bultan-Fu-Zhao, WS-FM 07]
- Crux of the problem:

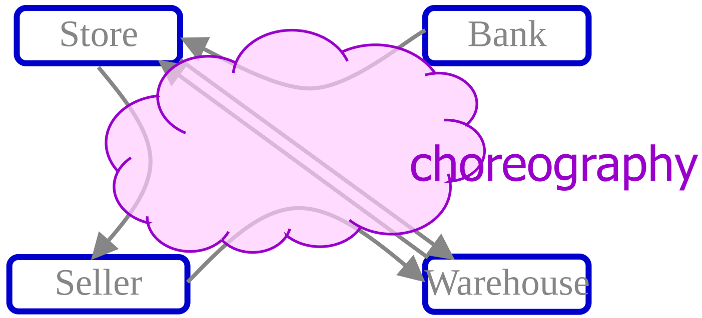


Choreography and projection to 4 peers

- When A, B, C, D operate autonomously, $m_2 m_1$ is possible

Choreography Realization Problem

- Given a choreography,
  how do we design
  an executable system to realize it?



- More practical:
  - Choreography design is a business decision
  - System design is software engineering problem

- Preliminary result: If a choreography has only 1-1 correlations, it can be realized
  - The executable system uses a small number of auxiliary messages to synchronize

Outline

- Choreography & Biz Processes
- Key Aspects of Choreography Specification
  - Weaknesses of existing choreography languages
- Ingredients of Our Approach
  - Artifacts as Biz Processes
  - Correlations
  - Message Diagrams
- Snapshots and Temporal (Choreography) Constraints
- Realization
- Conclusions

Conclusions

- BPM is a rich research area for CS:
  modeling, analytics, interoperation, evolution, …
- Collaborative BPs an interesting & very relevant thread in BPM
  - CS techniques helpful for orchestration
- CS techniques necesssary for choreography
  - This talk: trying to get to the technical details

development of specification languages, realization techniques, runtime monitoring and support, making changes, etc.

Future Problems

- Choreography specification with instance and data
  - FO+LTL semantics [Sun 2013]
  - Alternative framework? E.g., FSMs, process algebras, Petri nets, …
- Analysis of choreography
  - Satisfiability? (Seems undecidable for our language)
  - Finiteness? (Guarantee to terminate in finite steps, likely undecidable)
- Realization
  - Static compilation
  - Dynamic schemes