Trust Engineering

via

Cryptographic Protocols

Joshua D. Guttman

Jonathan C. Herzog   Jonathan K. Millen   John D. Ramsdell

Brian T. Sniffen   F. Javier Thayer

Supported by: **MITRE-Sponsored Research**

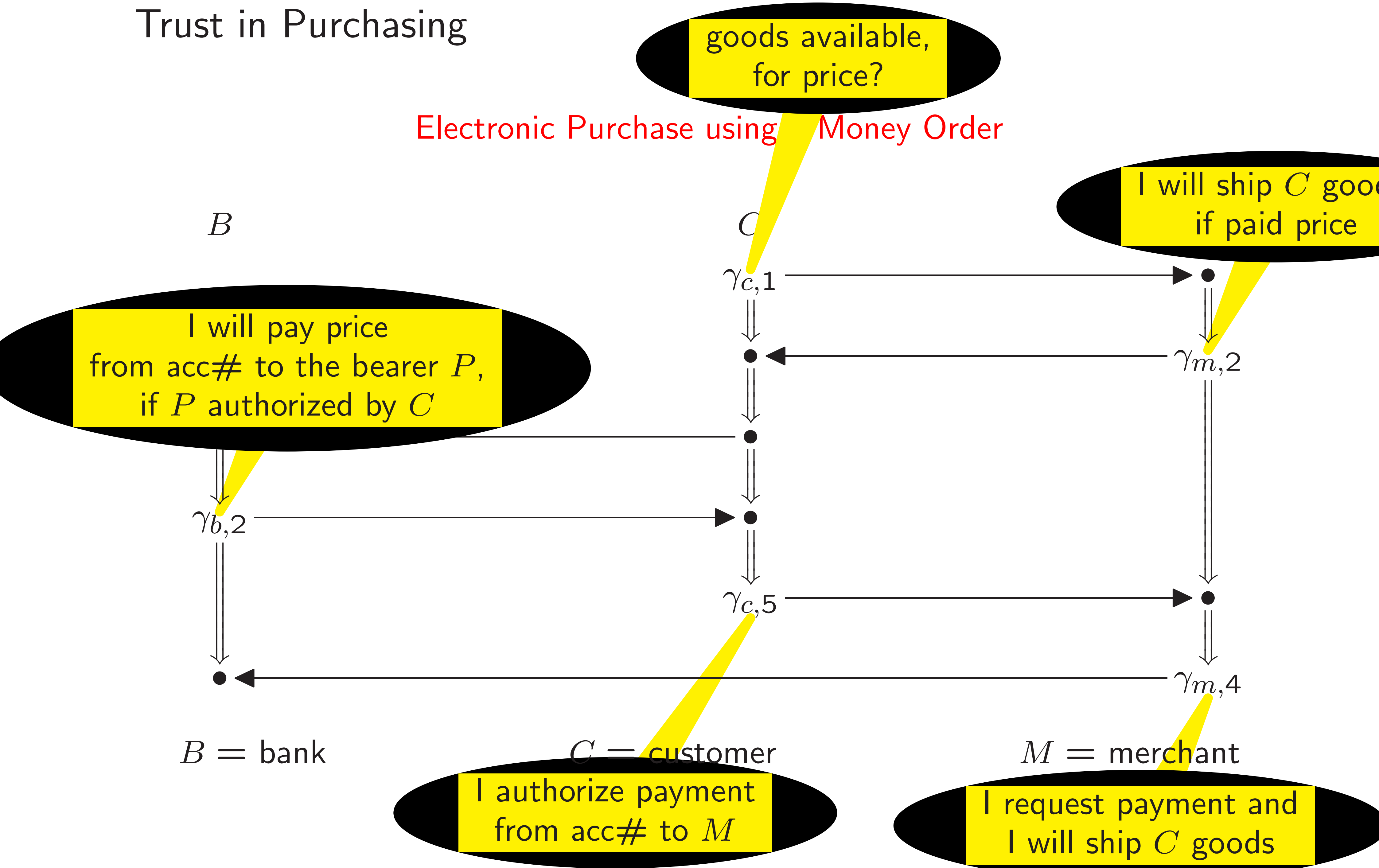
Trust Engineering

- Security in distributed systems must handle:
  - Many people, organizations, machines
  - Essentially different goals and policies
- Pervasive issues:
  - What principal is making a request?
  - If I respond, what action must I take?
  - What policy do I use to decide?
- Trust engineering goal:

  control global sequences of events via local decisions

  - My decisions suffice to prevent harm to me,
    even from actions taken elsewhere
  - I can appraise source, reliability of information from others
  - I can predict who might receive information I transmit

Trust in Purchasing



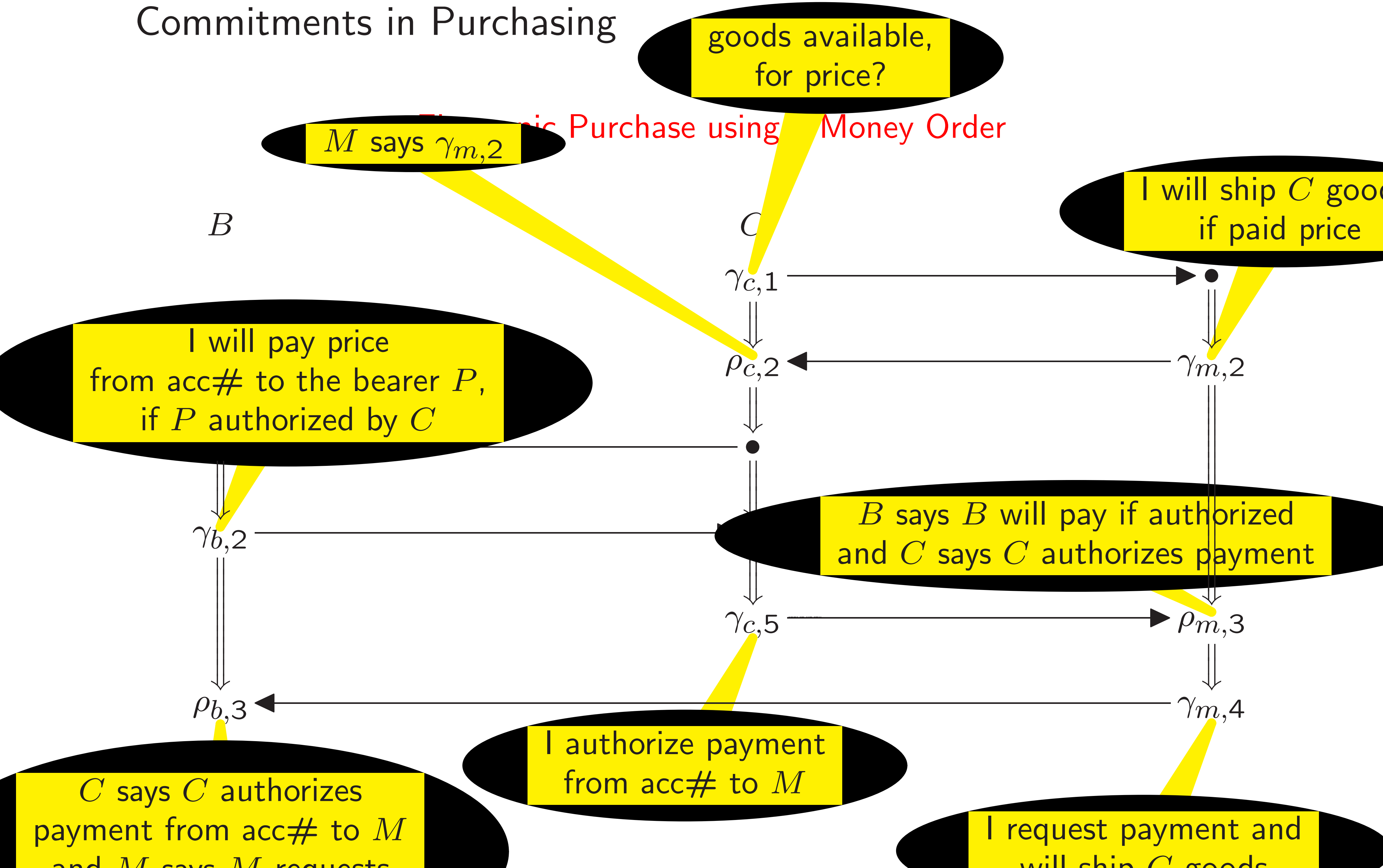
EPMO Goals

- At the end of a run, C, M, B agree on identities and price
  - B to transfer price from C's acct to M's
- C, M agree on goods
  - At the end of a run, M to ship goods to C
- Protocol preserves confidentiality:
  - M never learns C's account number
  - B never learns goods
  - Other parties never learn $C, M, B, \text{price}, \text{goods}$
  - B learns M's identity only if C decides to complete transaction

Types of goal:

- Authentication of identities
- Agreement on other parameters
- Confidentiality
- Agreement on commitments

Commitments in Purchasing



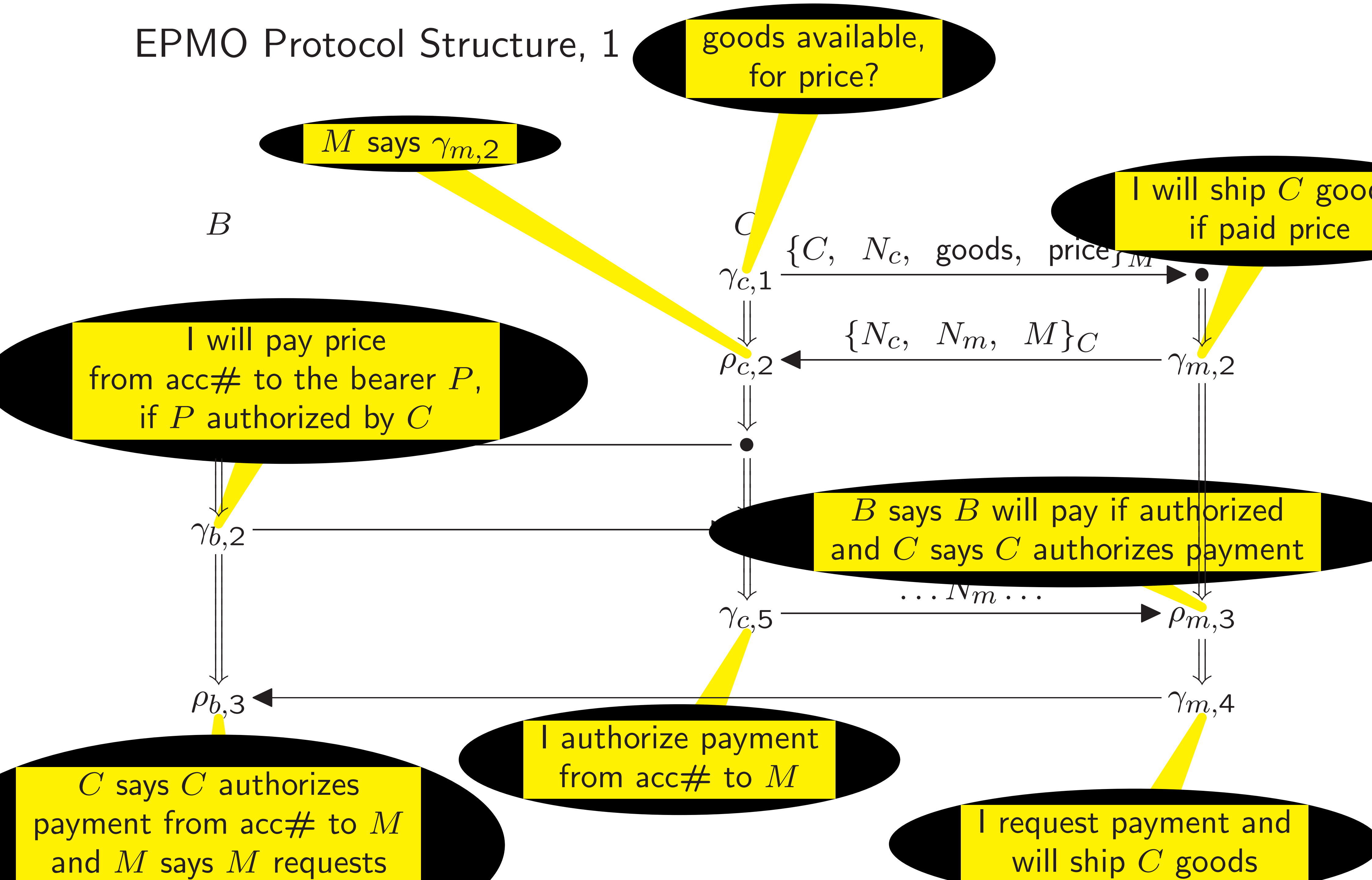
Trust Engineering

- Trust engineering goal:

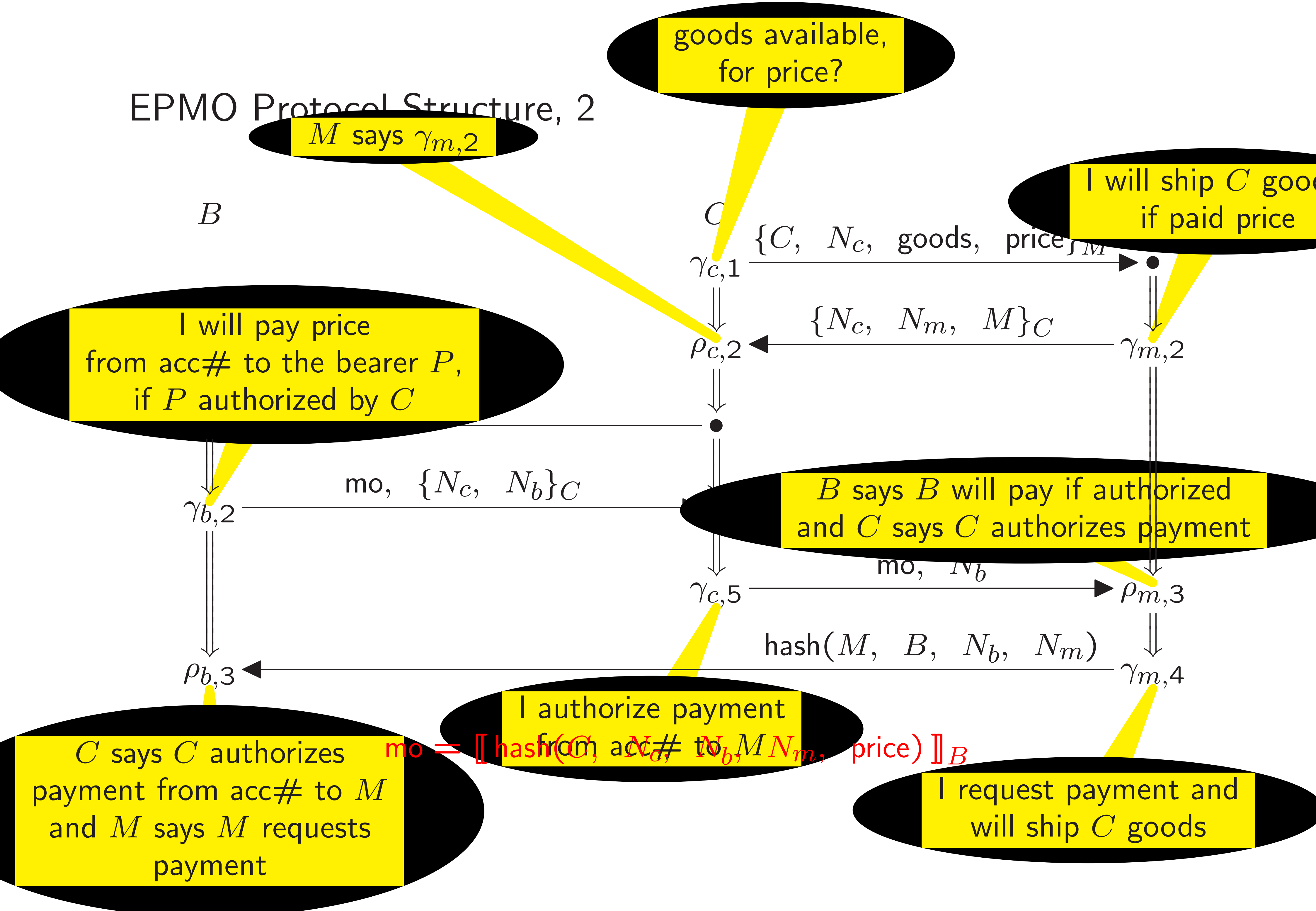
  control global sequences of events via local decisions

  - My decisions suffice to prevent harm to me, even from actions taken elsewhere
  - I can appraise source, reliability of information from others
  - I can predict who might receive information I transmit
- How to design new, application-specific protocols
  - Craft transactions in
    - Electronic commerce, web services, remote attestation
  - "Trust engineering:" Protocol to match trust goals of participants
- Goals of this talk: Explain
  - When is a protocol strong enough for its trust goals?
  - CPPL, a domain specific programming language for trust eng.

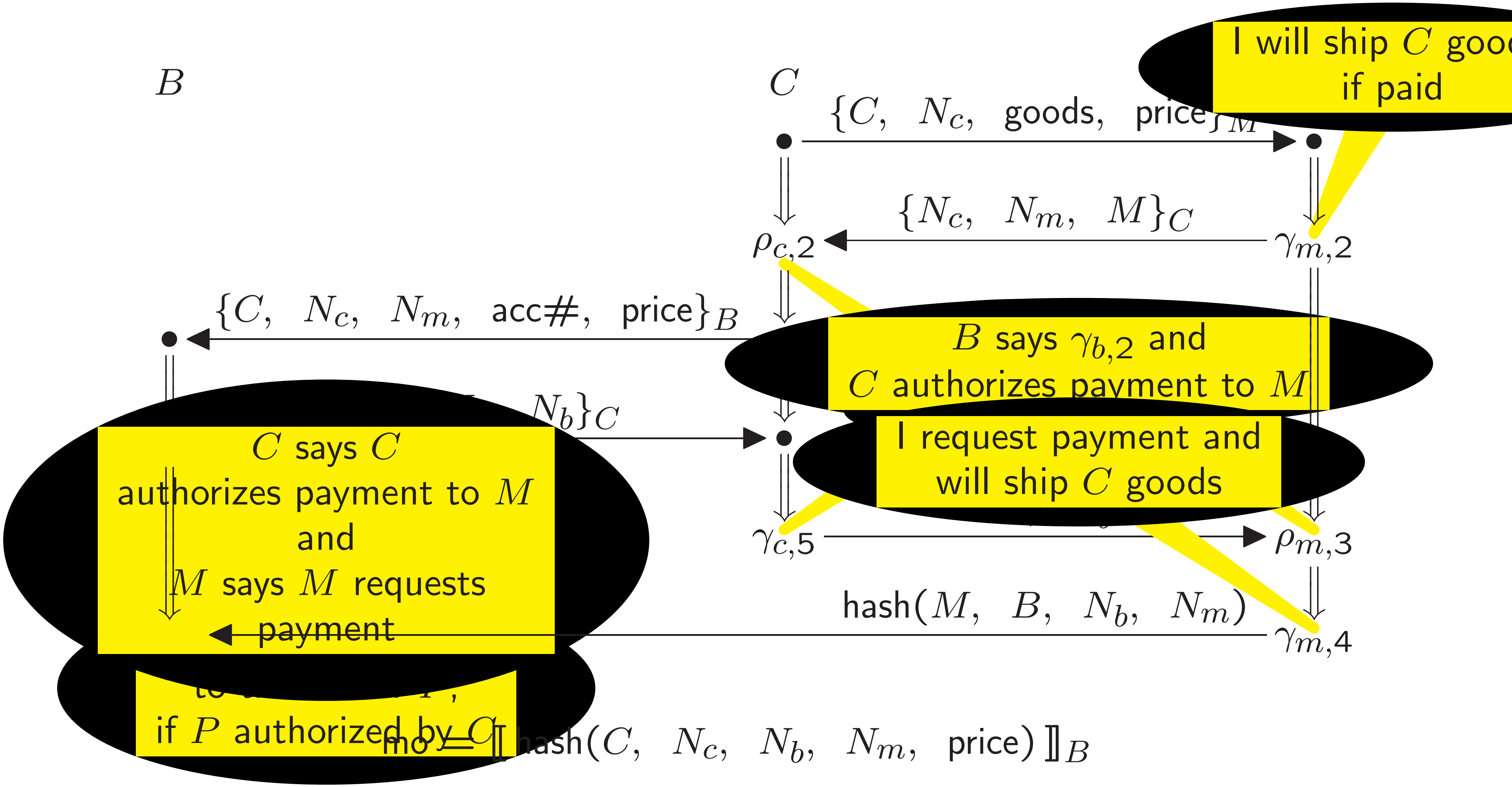
EPMO Protocol Structure, 1



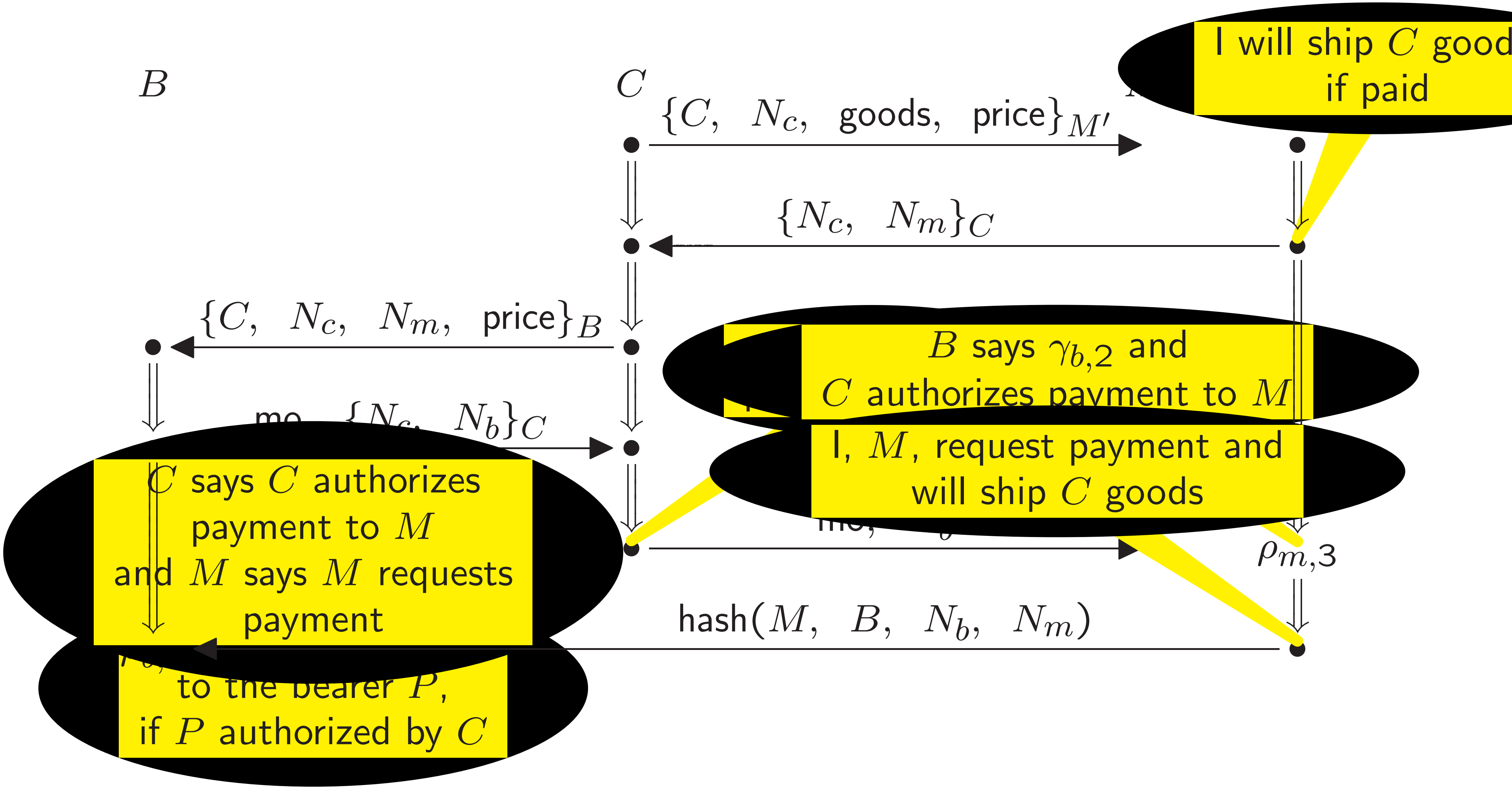
EPMO Protocol Structure, 2



EPMO Weakened



Lowe-style attack



Authentication Protocols: A Coordination Mechanism

- Causes principals to agree on certain parameters
- After a run, participant knows:
  - There is a protocol run by another principal
  - Some parameters match across runs
  - Some shared values are secrets
  - Other principal's run overlaps mine temporally
- Protocol design now tractable, based on a few theorems
  - "Authentication tests" determine extent of agreement
  - Formalize reasoning of previous slides via strands and bundles
- Formulas γ, ρ clarify real-world consequences of protocol run
  - When is customer committed to paying?
  - When is merchant committed to shipping?
  - Whose word did you depend on when deciding?
- Trust decisions constrain protocol runs ("business logic")

Trust management and protocols

- Each principal P
  - Reasons locally in initial theory Th_P, e.g. a theory in Datalog
  - Derives guarantee before transmitting message
  - Relies on assertions of others as premises
- Premises: formulas associated with message receptions
  - Specifies what recipient may rely on, e.g.
    "B says B will transfer funds if authorized"
  - Provides local representation of remote guarantee
  - Th_P determines whether ϕ follows from P' says ϕ
- Role of protocol
  - When I rely on you having asserted a formula,
    then you did guarantee that assertion
  - Coordination mechanism for rely/guarantees
  - Sound protocol: "relies" always backed by "guarantees"
    even with malicious adversary M'

Relevant notion of "trust"

Soundness

- Protocol Π is sound if:
  for all executions $\mathcal{B}$ of Π,
  and message receptions $n \in \mathcal{B}$

$$\{\text{prin}(m) \text{ says } \gamma_m \colon m \prec_{\mathcal{B}} n\} \quad \longrightarrow_{\mathcal{L}} \quad \rho_n$$

  where

  $\longrightarrow_{\mathcal{L}}$ is the consequence relation of the underlying logic

  $\prec_{\mathcal{B}}$ is the partial order generated by

  $m \to n$ implies $m \prec n$ (msg trans)
  $m \Rightarrow n$ implies $m \prec n$ (next step on strand, i.e. local run)

- Soundness follows from authentication properties
  - Strand space authentication methods work fine
  - Recency easy to incorporate
- Soundness:
  Criterion for Π to be strong enough for its trust interpretation

A Domain Specific Language

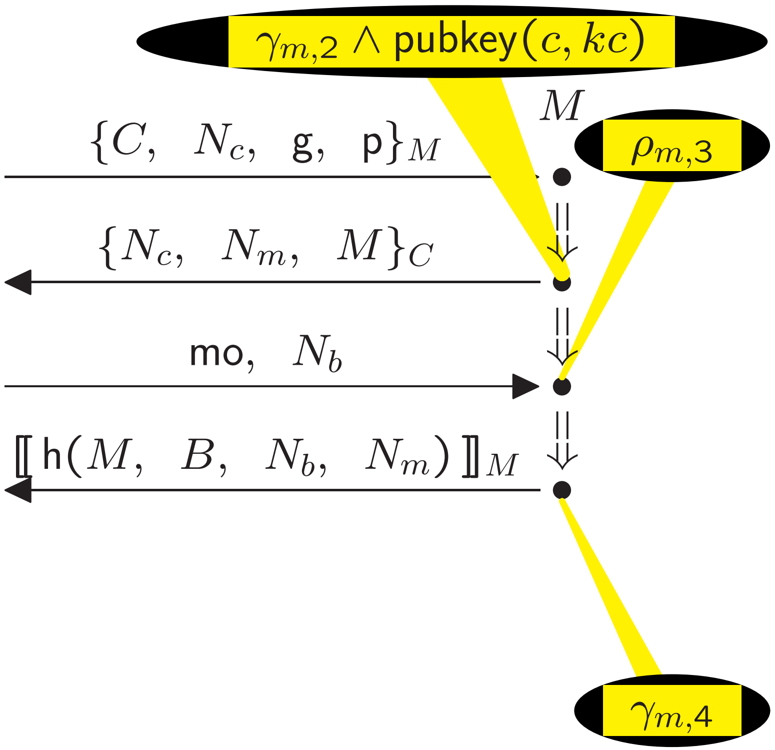
- CPPL, a Cryptographic Protocol Programming Language
  - Expresses cryptographic protocols
  - Programmer treats crypto primitives as black boxes
  - Controls behavior via trust queries
  - Equipped with a useful semantics
    - Useful for proving protocol security
    - Useful in structuring compiler we wrote

In the strand space framework

spi or applied pi would also work

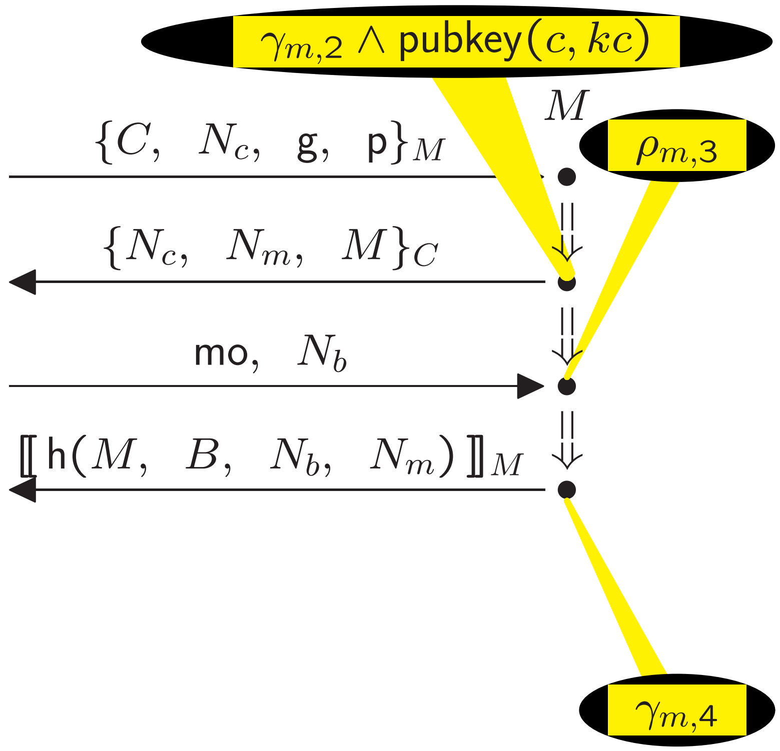
Coding the Merchant

```
let  chan = accept in
     receive chan
        {c, n_c, goods, price} km
       --> _
     let n_m = new nonce in
       send _
         --> chan {n_c, n_m, m} kc
       receive chan
           [[hash(c, n_c, n_b, n_m, price)]] skb, n_b
         --> _
       send _
         --> chan [[hash(B, n_b, n_m)]] skm
     return
```



Coding the Merchant: Trust Formulas

```
let  chan = accept in
     receive chan
        {c, n_c, goods, price} km
       --> true
     let n_m = new nonce in
       send γ_{m,2} and pubkey(c,kc)
         --> chan {n_c, n_m, m} kc
       receive chan
           [[hash(c, n_c, n_b, n_m, price)]] skb, n_b, b
         --> if sigkey(b,skb) then ρ_{m,3}
       send sigkey(b,skb) and γ_{m,4}
         --> chan [[hash(B, n_b, n_m)]] skm
     return
```



Semantics of CPPL

- A structured operational semantics
- Judgment:

$$\sigma \; ; \; \Delta \vdash \; \mathsf{c} \; : \; s$$

- Means:

  In environment σ,
  a principal holding theory Δ,
  executing code c,
  may unleash strand s

partial map
from identifiers
to values

“strand:”
purely local sequence
of sends, receives

Semantics of receive

$$\frac{\sigma_1 = \sigma \oplus \sigma' \qquad \sigma_1 \;;\; \Delta, \phi\,\sigma_1 \;\vdash\; c \;:\; s}{\sigma \;;\; \Delta \;\vdash\; (x \;\texttt{recv}\; m \;\phi\; c) \;:\; -(x,\; m)\,\sigma_1, \phi\,\sigma_1 \Rightarrow s}$$

dom$(\sigma') \subseteq$ vars(m) vars$(x, m, \phi) \subseteq$ dom(σ_1)

$\oplus$ means disjoint union of fns

$-(x,\; m), \phi \Rightarrow s$:

receive m from channel x, relying on ϕ; then do rest of strand s

Semantics of send

$$\frac{\sigma_1 = \sigma \oplus \sigma' \qquad \Delta \Vdash \phi\sigma_1 \qquad \sigma_1 \,;\, \Delta \vdash c : s}{\sigma \,;\, \Delta \vdash (\texttt{send}\ \phi\ x\ m\ c) : +(x,\ m)\sigma_1, \phi\sigma_1 \Rightarrow s}$$

$\mathrm{dom}(\sigma') \subseteq \mathrm{vars}(\phi)$ $\qquad$ $\mathrm{vars}(x, m, \phi) \subseteq \mathrm{dom}(\sigma_1)$

$+(x,\ m), \phi \Rightarrow s$:

transmit m on channel x, guaranteeing ϕ; then do rest of strand s

CPPL Principles

- Principal maintains an environment during run
  - Variables progressively become bound, never change value after
  - Values are atomic (nonce, name, key, etc)
- Message transmission, reception:
  - Reception:
    - Branch on form of message
    - New variables bound from msg components
    - Rely on assertion of sender
  - Transmission:
    - Branch on successful guarantee
    - New variables bound from successful guarantee (as in logic programming)
- Derive guarantees using:
  - Th_P, your initial theory
  - Values of variables bound up to this point
  - Rely formulas for earlier msg receptions

Subprotocols

- Subprotocols encapsulated by rely/guarantee
  - Callee relies on assertion of caller
    - Property of input parameters
  - Callee guarantees result for caller
    - Relation on input, output parameters
  - Caller and callee are same principal P (same theory Th_P)
- Subprotocol call, return: local message transmissions
  - Call: Message from caller to callee
  - Return: Message from callee to caller
  - RPC-like mechanism ("LPC")
- Flow of information on subprotocol call, return matches convention
  - Guarantee before transmitting
  - Rely when receiving

Protocol Headers

```
epmo_merchant_role(m, km, skm): (c, b, goods, price)
    rely pubkey(m, km) and sigkey(m, skm)
    guarantee supplied(c, goods, price, b)
in ... end
```

Subprotocol Headers

```
retrieve_pubkey (b, a, c, cver, d, kd) : (a, ka)
  rely       certifying_authority(c, a)
         and directory_service(d, c)
         and pubkey(d, kd)
         and sign_verification_key_of(c, cver)
  guarantee pubkey(a, ka)
 in
 ...
 end
```

Precondition/postcondition
specifies effect of
successful run of subprotocol

Subprotocol call site

```
call with
  pubkey(a, ka)
    --> null_protocol () ()
    true
  use key ka...
|
  certifying_authority(c, a)
    and directory_service(d, c)
    and pubkey(d, kd)
    and sign_verification_key_of(c, cver)
      --> retrieve_pubkey (b, a, c, cver, d, kd) (a, ka)
    pubkey(a, ka)
  use key ka...
```

Subprotocol Semantics

Invocation

$$\frac{\mathrm{dom}(\sigma') \subseteq \mathrm{ide}(pr,\ \ n,\ \ ai,\ \ x^*) \qquad \begin{array}{c}\sigma_1 = \sigma_{orig} \oplus \sigma' \\ \sigma_1 \ ;\ \Gamma_0, (\Psi\,\sigma) \ \vdash \ \ c \ \ :\ s, v\end{array}}{\sigma_{orig} \ ;\ \Gamma_0 \ \vdash \ \ \texttt{proc}\ n\ \ \Psi\ \ x^*\ c \ \ :\ \ -\mathsf{call}\ pr,\ \ n,\ \ ai,\ \ x^*\,\sigma_1,\ \Psi\,\sigma_1 \Rightarrow s, v}$$

Return

$$\frac{\sigma_1 = \sigma \oplus \sigma' \qquad \mathrm{dom}(\sigma') \subseteq \mathrm{ide}(\Phi) \qquad \Gamma \Vdash \Phi\,\sigma_1}{\sigma \ ;\ \Gamma \ \vdash \ \ \texttt{return}\ \Phi\ \ x^* \ \ :\ \langle +\mathsf{ret}\,(ai,\ \ x^*)\,\sigma_1,\ \Phi\,\sigma_1\rangle, \emptyset}$$

Trust and Protocols

- Crypto protocols coordinate principals
  - Agree on parameter values
  - Agree on assertions made
- Trust decisions at runtime can control protocol behavior
  - Stop protocol run if trust constraints fail
  - Choose branch conditional on successful trust constraint
  - Message transmissions and subprotocol calls
- Strand-based semantics
  - Provides good protocol verification methods, design heuristics
  - Motivated language design and implementation
- Status: Second version of compiler now complete
  - Datalog trust engine, Crypto library
- Trust engineering using cryptographic protocols

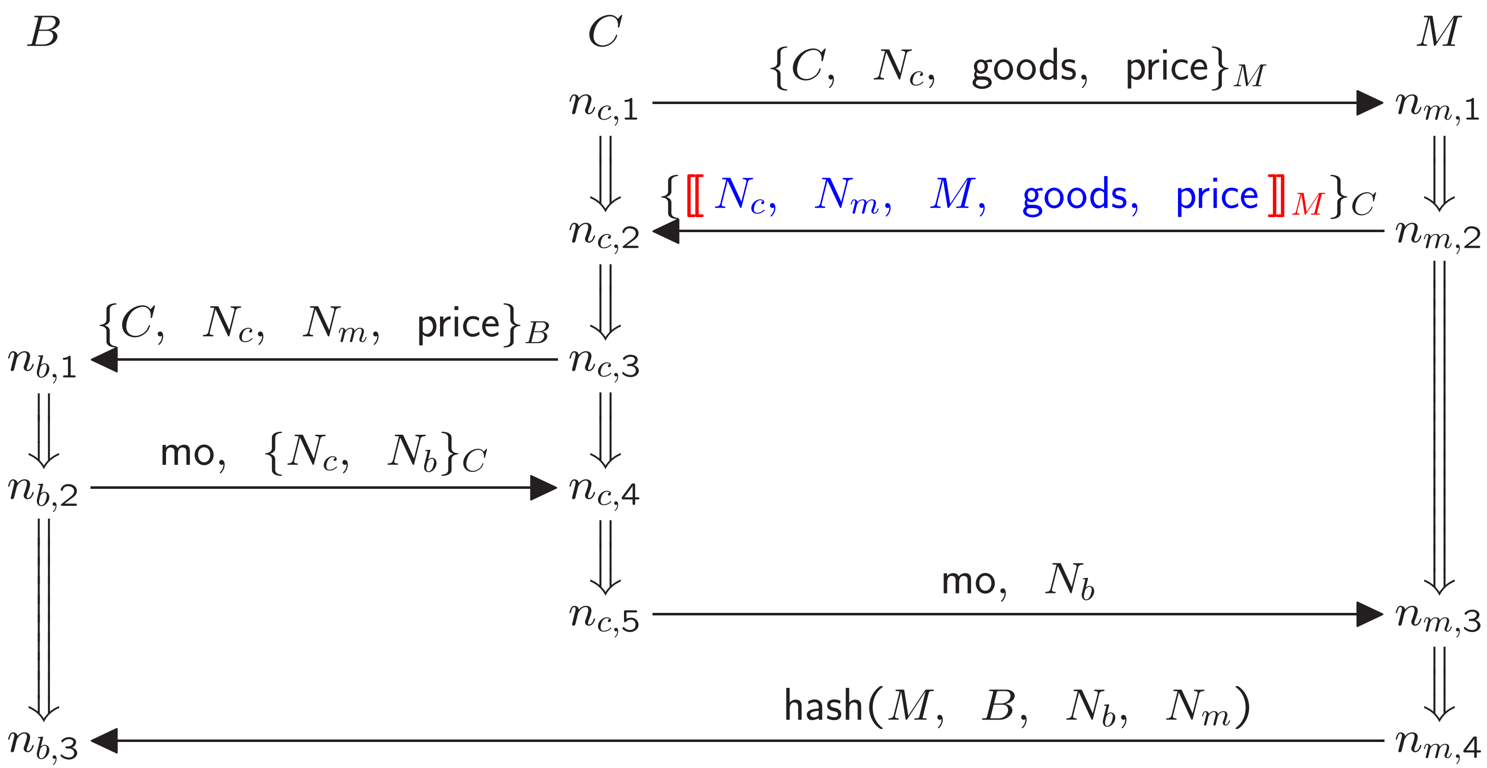
Contrast: Earlier Work

- The BAN tradition
  - Messages are formulas or formulas idealize messages
  - Who asserted the formulas?
  - Who drew consequences from formulas?
- Embedding formulas explicitly inside messages
  - Main view of logical trust mgt
  - Formulas parsed out of certificates
  - Problem of partial information?

starts with LAWB

- Our view: Formulas part of transmission/reception, not msg
  - Compatible with many insights of earlier views
  - Independent method to determine what events happened
  - Clarity about who makes assertions, who infers consequences
  - Partial information easy to handle
  - Rigorous notion of soundness

A Signed Alternate: SEPMO



Signed Electronic Purchase using Money Order

$\text{mo} = [\![\ \text{hash}(C,\ N_c,\ N_b,\ N_m,\ \text{price})\ ]\!]_B$