The Internet of Things and Cooperative Information Systems

Tutorial

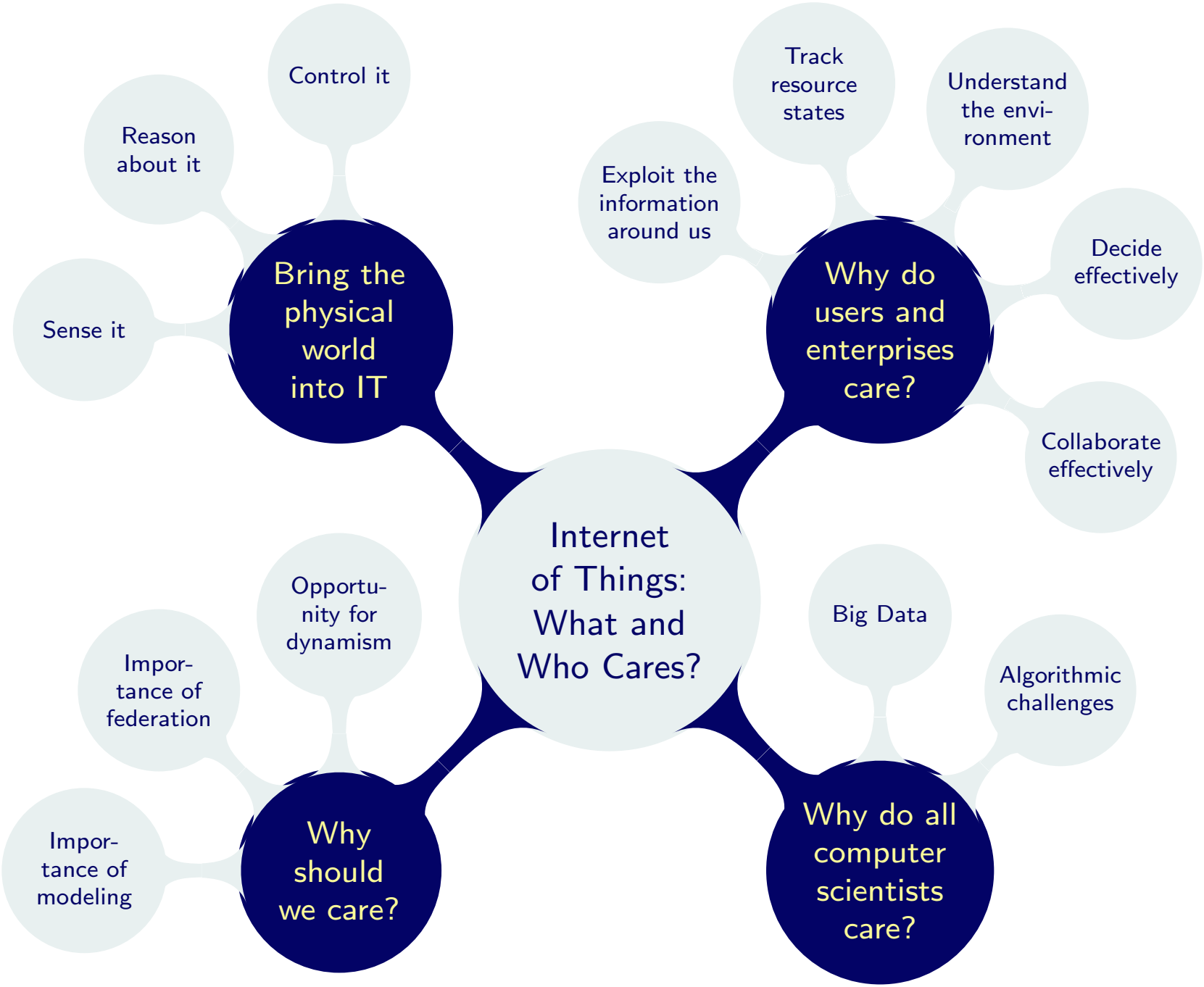
Munindar P. Singh and Amit K. Chopra

North Carolina State University
Lancaster University

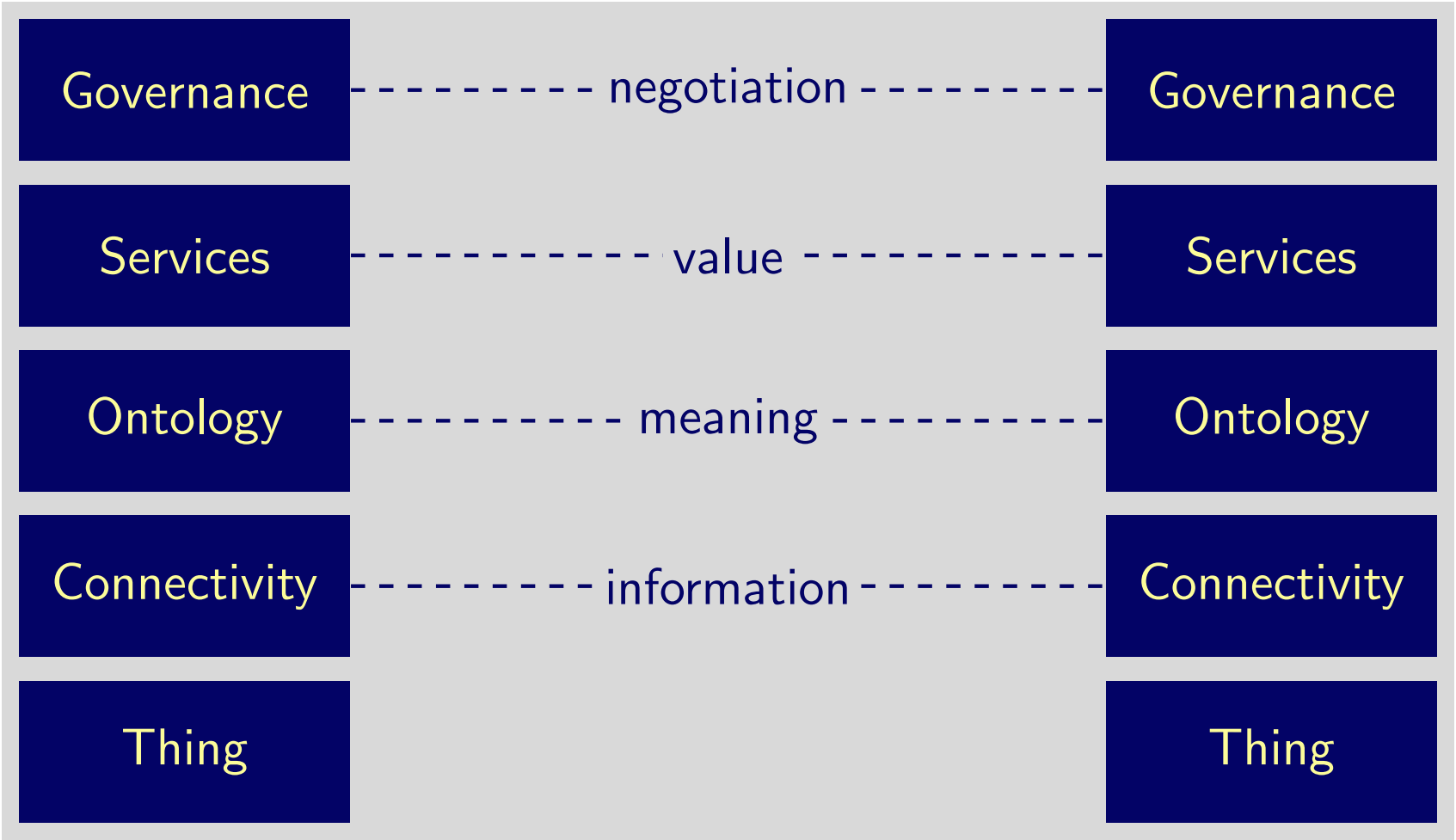
May 2015

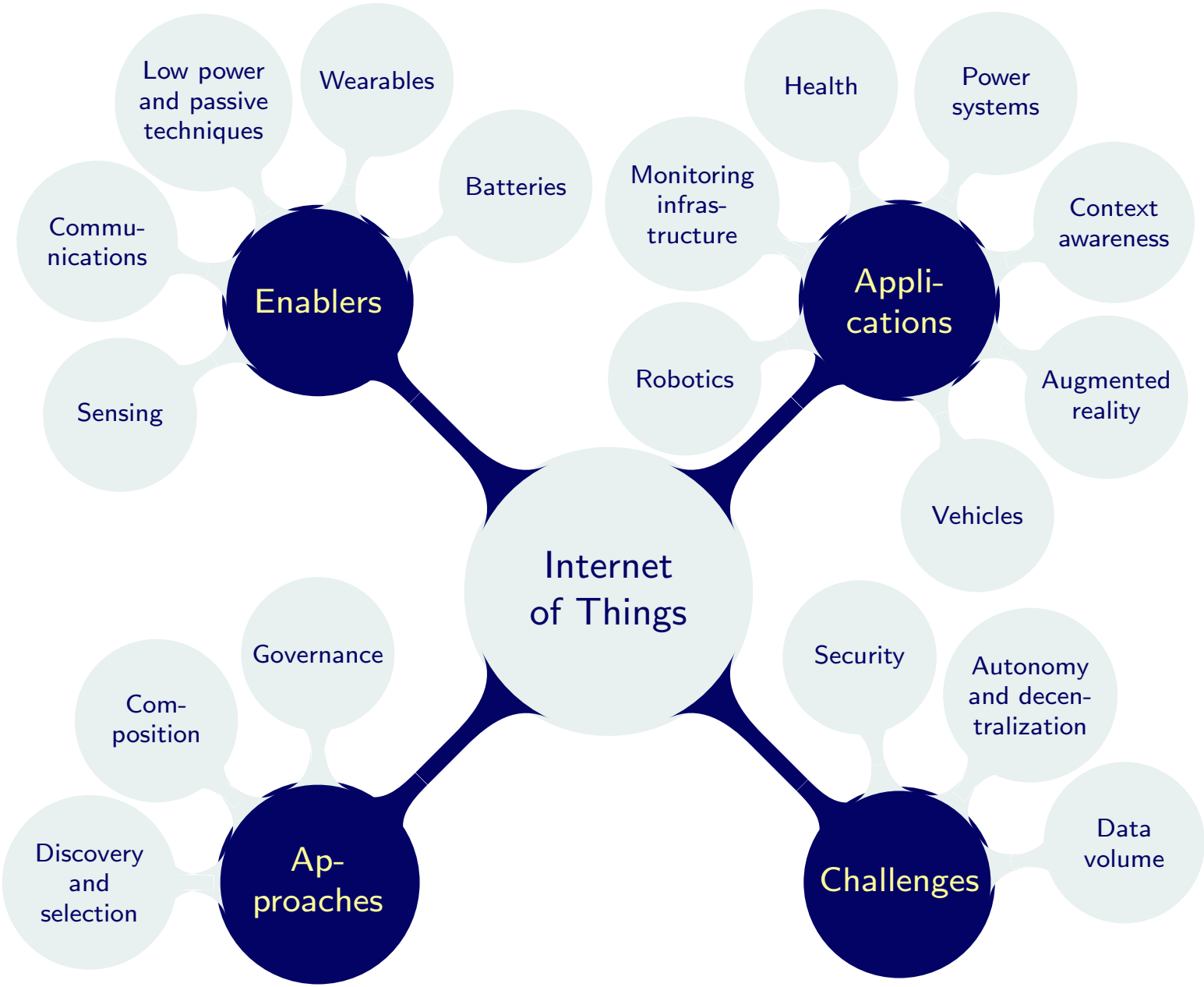
Outline

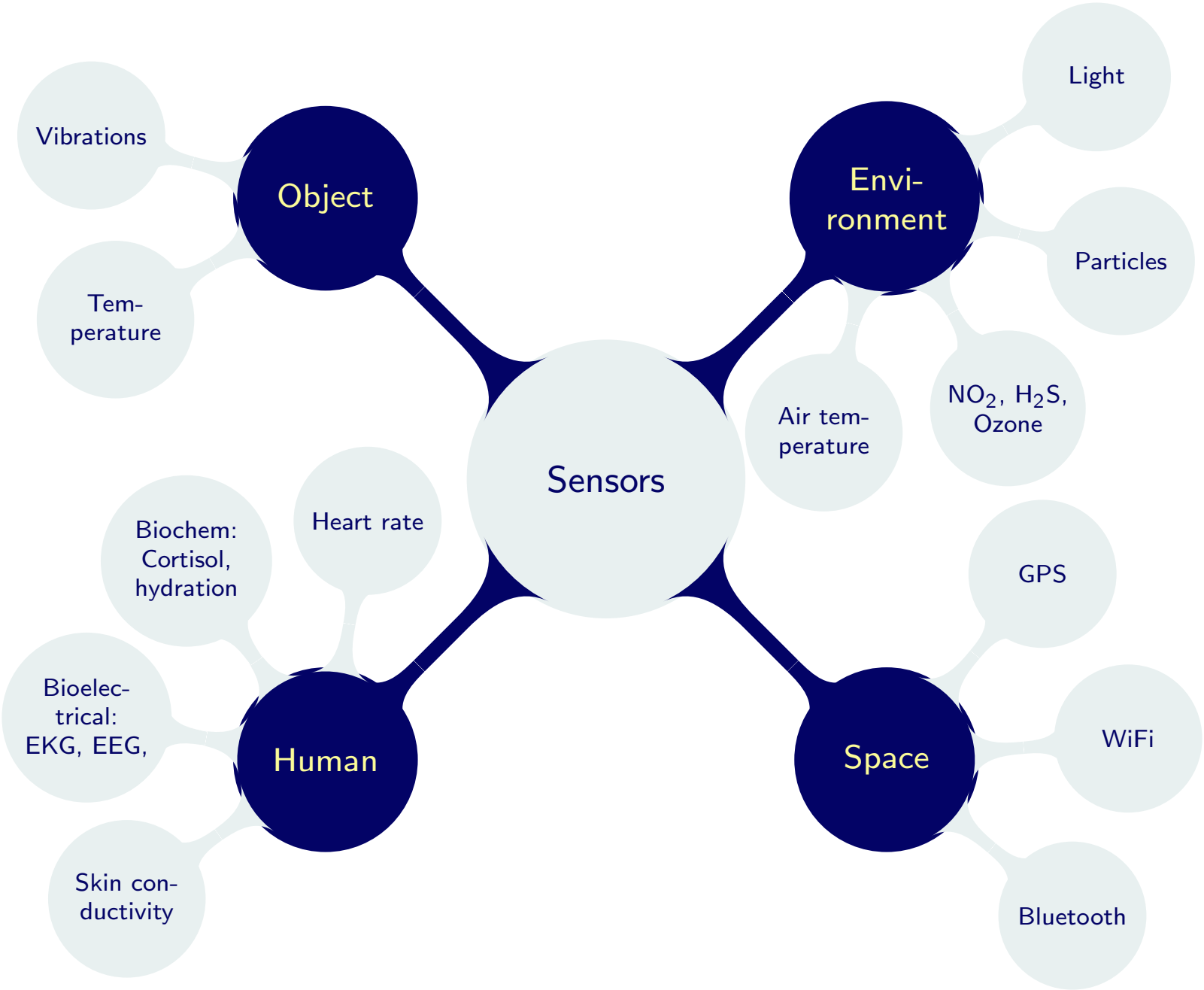

Scope and motivation of this tutorial
An introduction to a new area
From a sociotechnical perspective
Not a programming guide
With a hope to help build a research community
Focus on research challenges


Internet of Things: What and Who Cares?
Bring the physical world into IT
Control it
Reason about it
Sense it
Why do users and enterprises care?
Track resource states
Understand the environment
Exploit the information around us
Decide effectively
Collaborate effectively
Why should we care?
Opportunity for dynamism
Importance of federation
Importance of modeling
Why do all computer scientists care?
Big Data
Algorithmic challenges

IoT Federation Levels




Internet of Things
Enablers
Low power and passive techniques
Wearables
Batteries
Commu-nications
Sensing
Appli-cations
Health
Power systems
Monitoring infras-tructure
Context awareness
Robotics
Augmented reality
Vehicles
Ap-proaches
Governance
Com-position
Discovery and selection
Challenges
Security
Autonomy and decen-tralization
Data volume


Sensors
Object
Vibrations
Tem-
perature
Envi-
ronment
Light
Particles
Air tem-
perature
NO_2, H_2S,
Ozone
Human
Biochem:
Cortisol,
hydration
Heart rate
Bioelec-
trical:
EKG, EEG,
Skin con-
ductivity
Space
GPS
WiFi
Bluetooth

Core: Connectivity

Combination of low and high-bandwidth

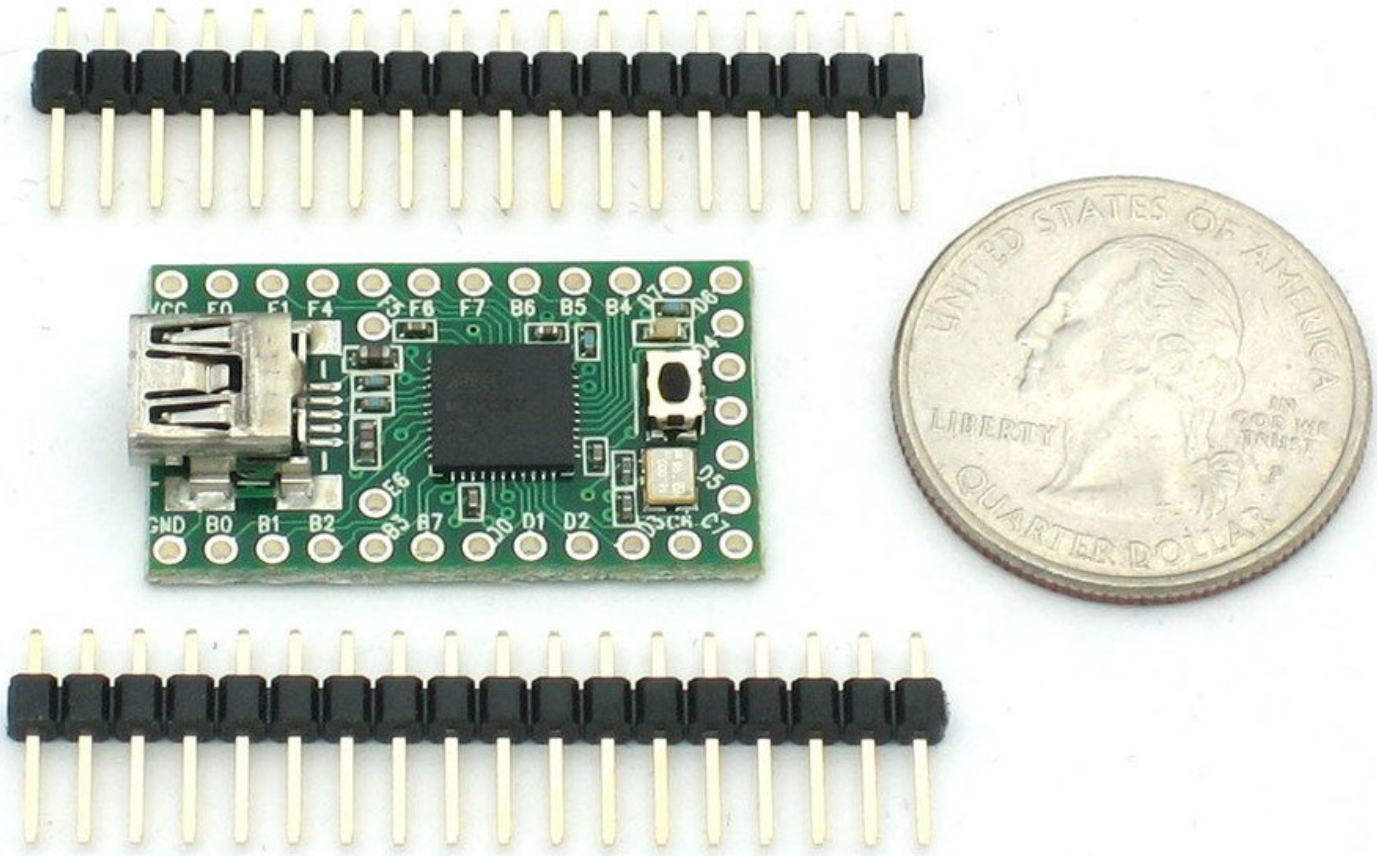
- Low-bandwidth connectivity
  - Between RFID tags and readers
  - Between sensors and base stations
- High-bandwidth
  - Wireless, with aggregator nodes such as smartphones and Arduino devices
  - Broadband, from aggregator nodes to store data in the cloud

Core: Passive and Low Power Technologies

Many dimensions

- Passive (batteryless)
  - No power
  - Require a proximal (within $\sim$10 cm) reader
- Active, battery-based
  - Can send information
  - Can last years but need battery replacement
- Active, self-powered
  - Harvest power from the
    - Environment, e.g., solar
    - Human body, temperature differential
    - Human body, movement
  - Enough power to transmit to a local, e.g., on-body, hub
    - Low-power radio: $10^{-7}W$ (versus Bluetooth: $10^{-3}W$)
    - Human body, temperature differential
    - Human body, movement
  - Can potentially last "forever"

Microcontrollers for Programmably Controlling Actuators



©Adafruit, New York

Power Effectiveness: Processors

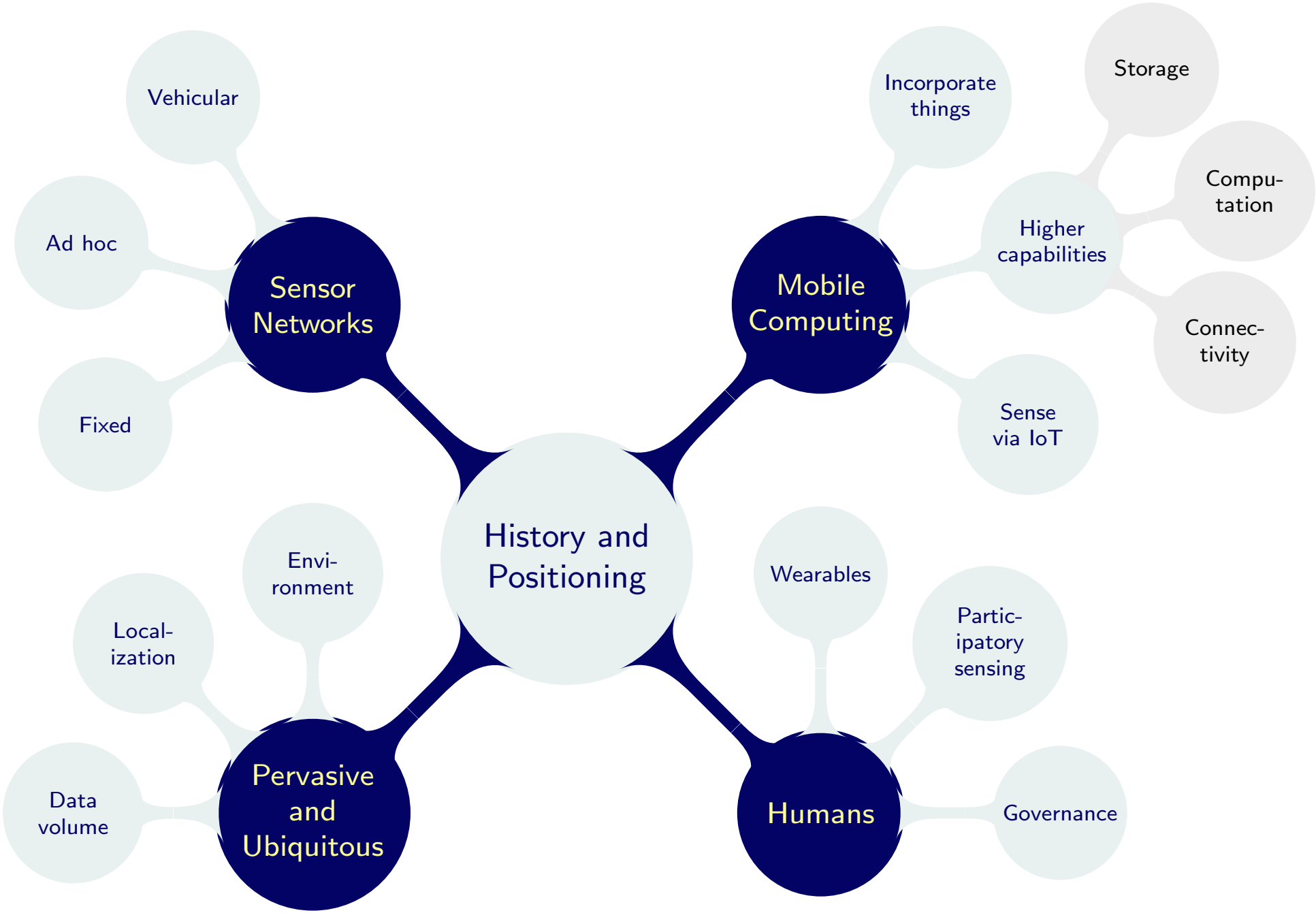
	Prototype	**Commercial 1**	**Commercial 2**
	NCSU ASSIST	EnOcean STM 31xC	Semtech SX1282
Voltage	0.5V	2.1V	1.0V to 1.6V
Processor	16b MSP430	Custom	8b CoolRISC
Power consumption	$< 1\mu W$@$200kHz$	5.1mA @ 3–5V	$1.2\mu W$ @ 32kHz; $600\mu W$ typical
Power harvesting	RF, solar thermoelectric	Yes	No

Credit: ASSIST Center, NCSU, ⟨http://assist.ncsu.edu/⟩

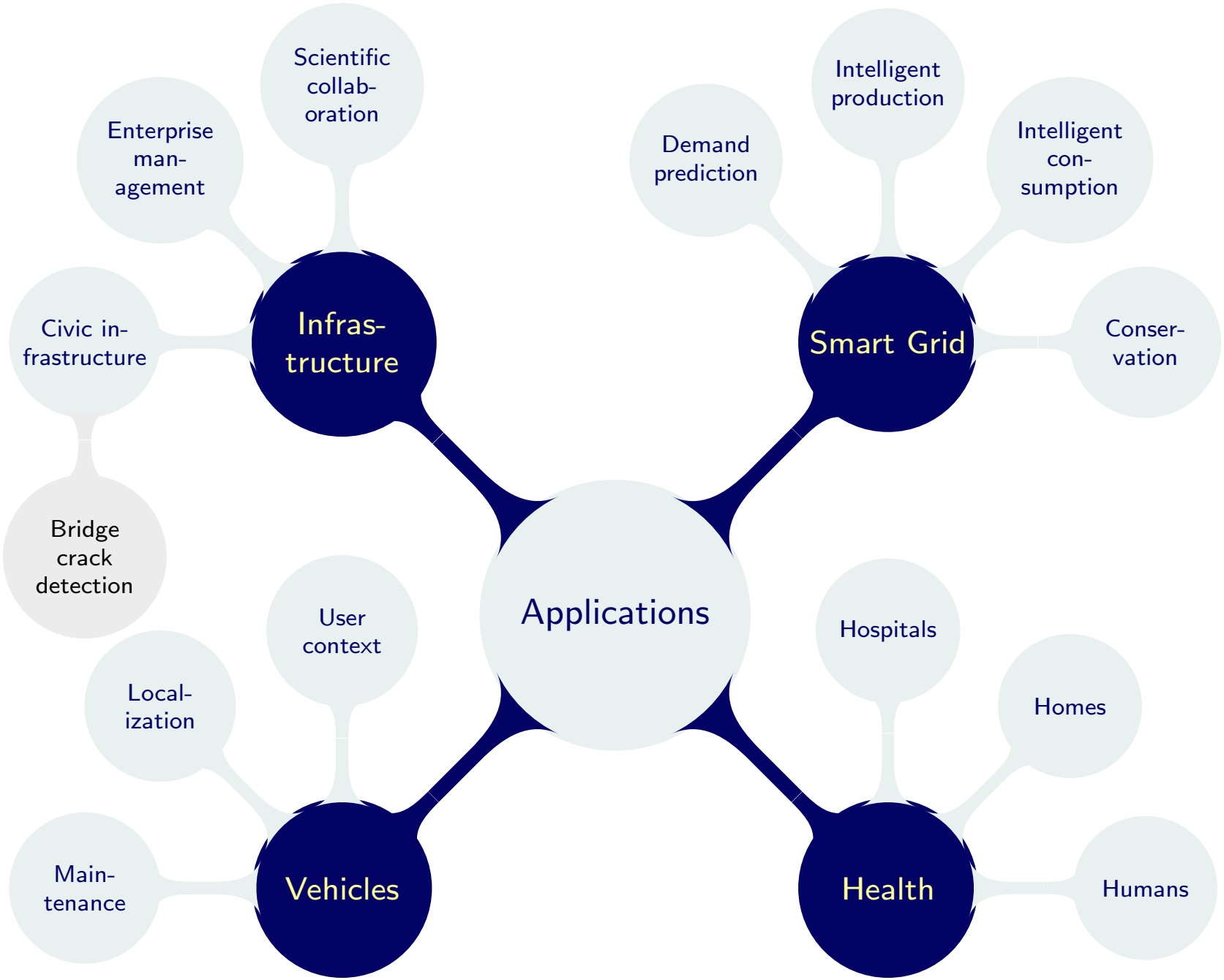
Power Effectiveness: Radios

	Prototype	**Commercial 1**	**Commercial 2**
	NCSU ASSIST	EnOcean STM 31xC	Semtech SX1282
Voltage	0.5–1.0 V	2.1V	1.0V to 1.6V
Transmission power	$6\mu W$ @ 200kbps	$30mW$ @125 kbps	$\sim 40mW$
Reception power	$200\mu W$ WBAN $120nW$ @ 12.5 kbps WU	$40mW$	$\sim 12mW$

Credit: ASSIST Center, NCSU, ⟨http://assist.ncsu.edu/⟩


History and Positioning
Sensor Networks
Vehicular
Ad hoc
Fixed
Mobile Computing
Incorporate things
Storage
Compu-tation
Higher capabilities
Connec-tivity
Sense via IoT
Pervasive and Ubiquitous
Envi-ronment
Local-ization
Data volume
Humans
Wearables
Partic-ipatory sensing
Governance

Outline


Applications
Infrastructure
Scientific collaboration
Enterprise management
Civic infrastructure
Bridge crack detection
Smart Grid
Intelligent production
Demand prediction
Intelligent consumption
Conservation
Vehicles
User context
Localization
Maintenance
Health
Hospitals
Homes
Humans

Internet of Trails?

Crabtree Lake Trail near Raleigh, North Carolina

©Munindar P. Singh

Internet of Hotels and Homes?

Internet of Oceans: Global Hybrid Profile Mooring Launch

©Tom Kleindinst, WHOI

Internet of Oceans: Glider Being Launched

Smart Grid

Demo setup at AAMAS 2015

©Munindar P. Singh

Internet of Lakes

Crabtree Lake near Raleigh, North Carolina

©Munindar P. Singh

Internet of Dogs

Tagged stray dog in Istanbul

©Munindar P. Singh

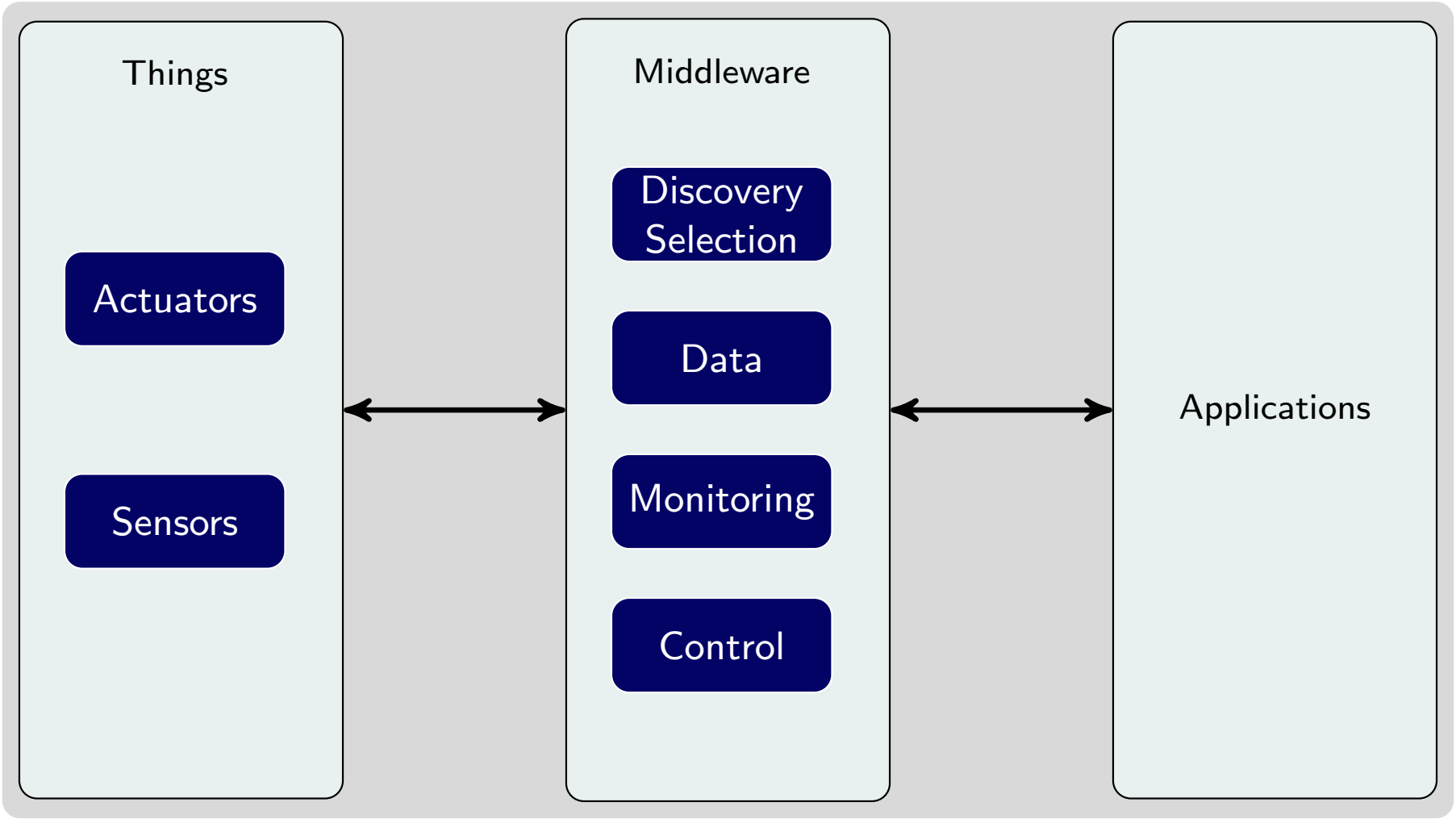
Internet of Monuments?

From the Acropolis; appears to be a visual reader, not an IoT sensor

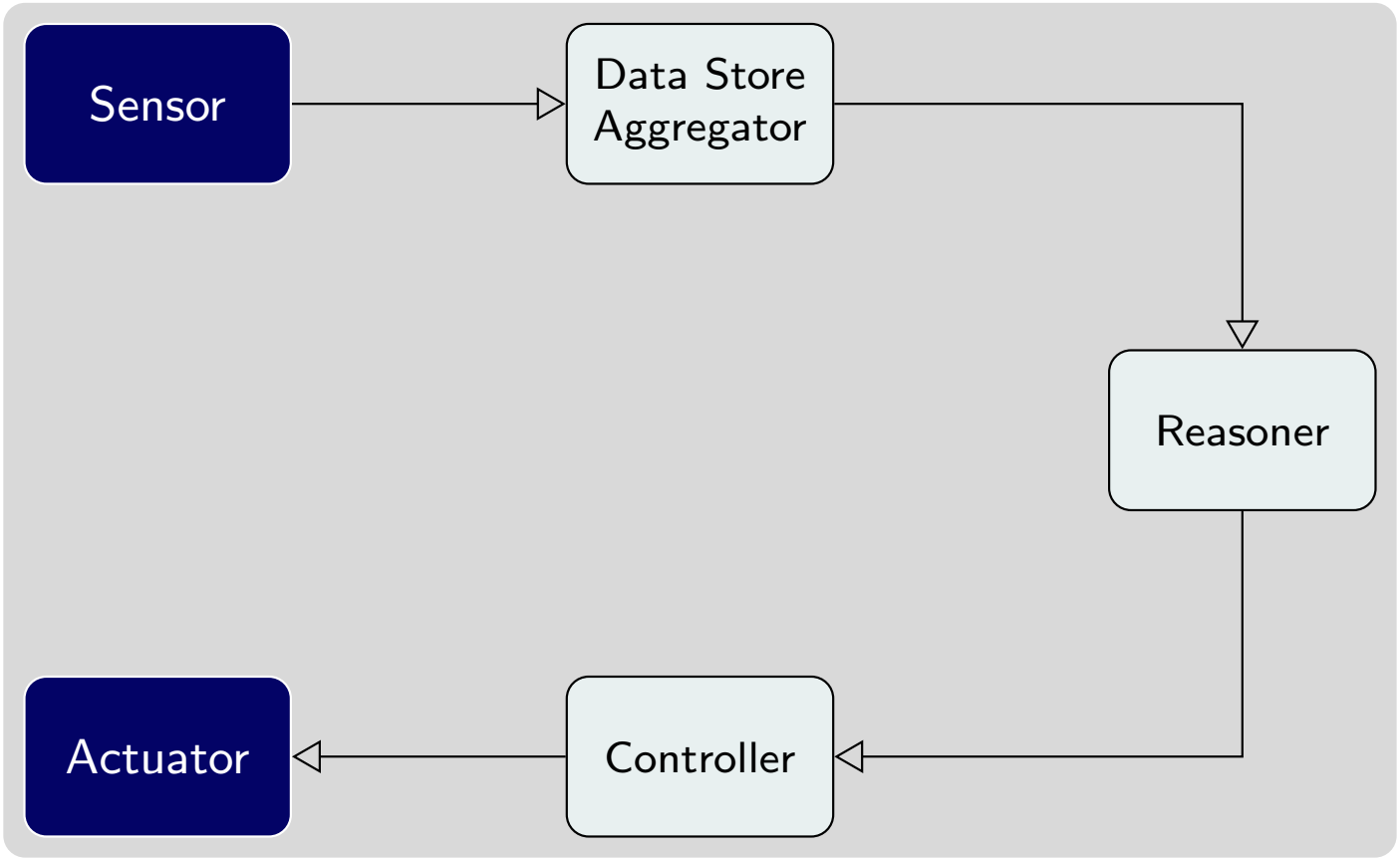
©Munindar P. Singh

Outline

Basic Architecture and Services



Architecture: Information Flow



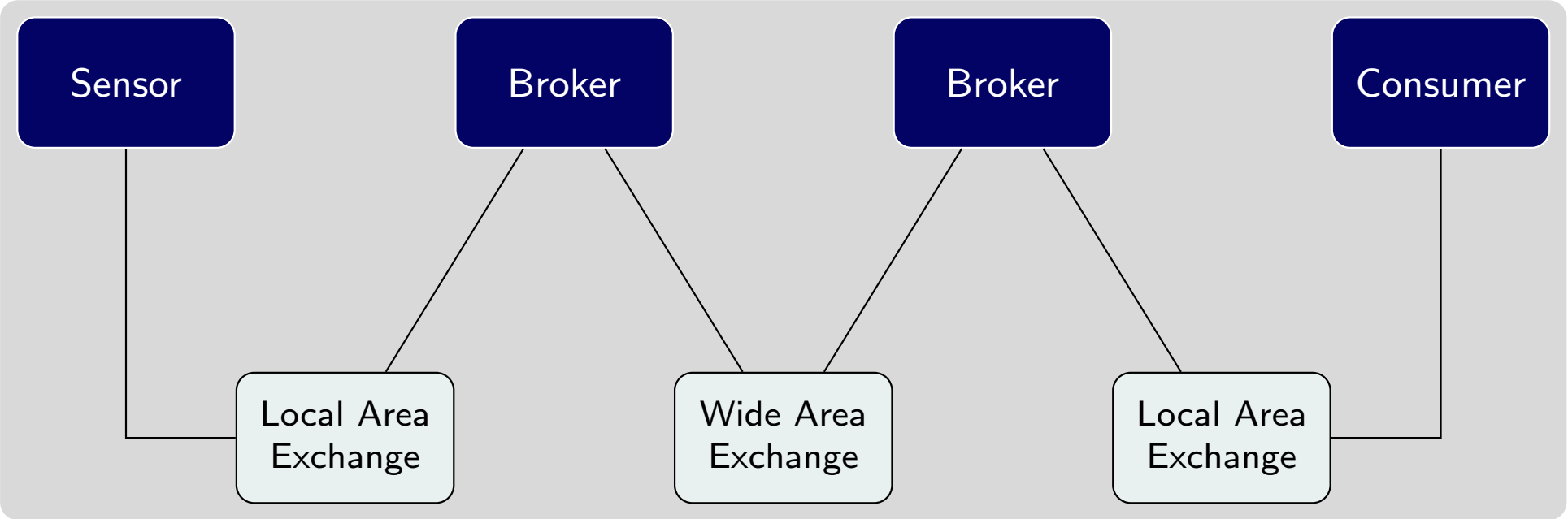
Enablers: Communications

Advanced Message Queuing Protocol

- Protocol, not like an API (e.g., JMS)
- Decouples communications from destination address
- Long-lived conversations
- Variety of communication patterns
  - Intercept
  - Delegate
  - Multiplex and demultiplex
- Upcoming improvements
  - Traffic flow and QoS
  - Decentralized deployment and governance
  - Multiple underlying protocols

Architecture: Information Exchange

AMQP style exchange space abstraction

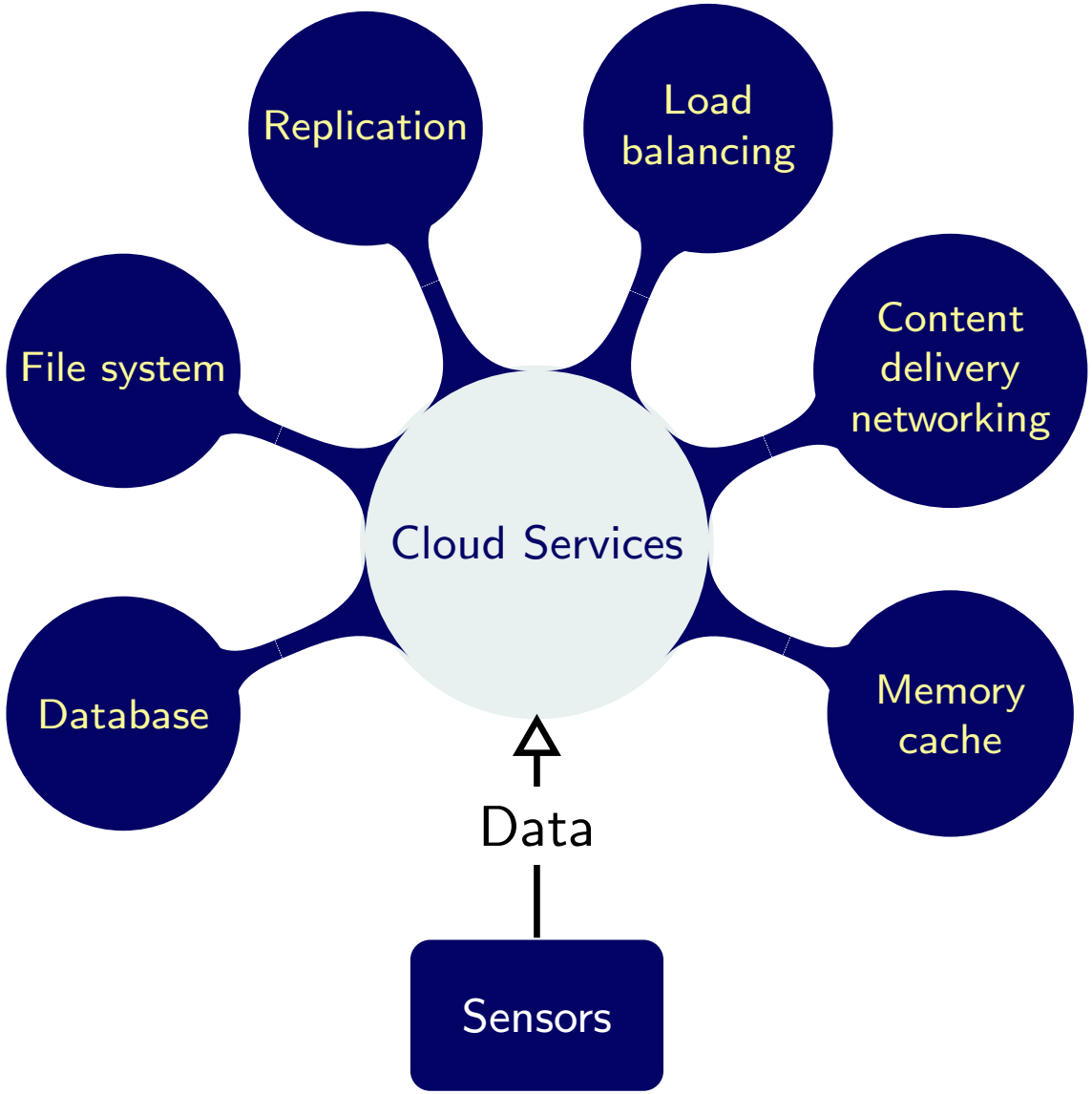


Based on work with Ocean Observatories Initiative, UCSD Scripps

Enablers: Cloud Storage

Eliminate need to set up complex storage services from scratch

- On demand
- Variety of data services
- *Never delete* operation
- NoSQL (and NewSQL) databases
  - Alternative to ACID databases
  - Tradeoff strong consistency for low latency



Enablers: Clouds

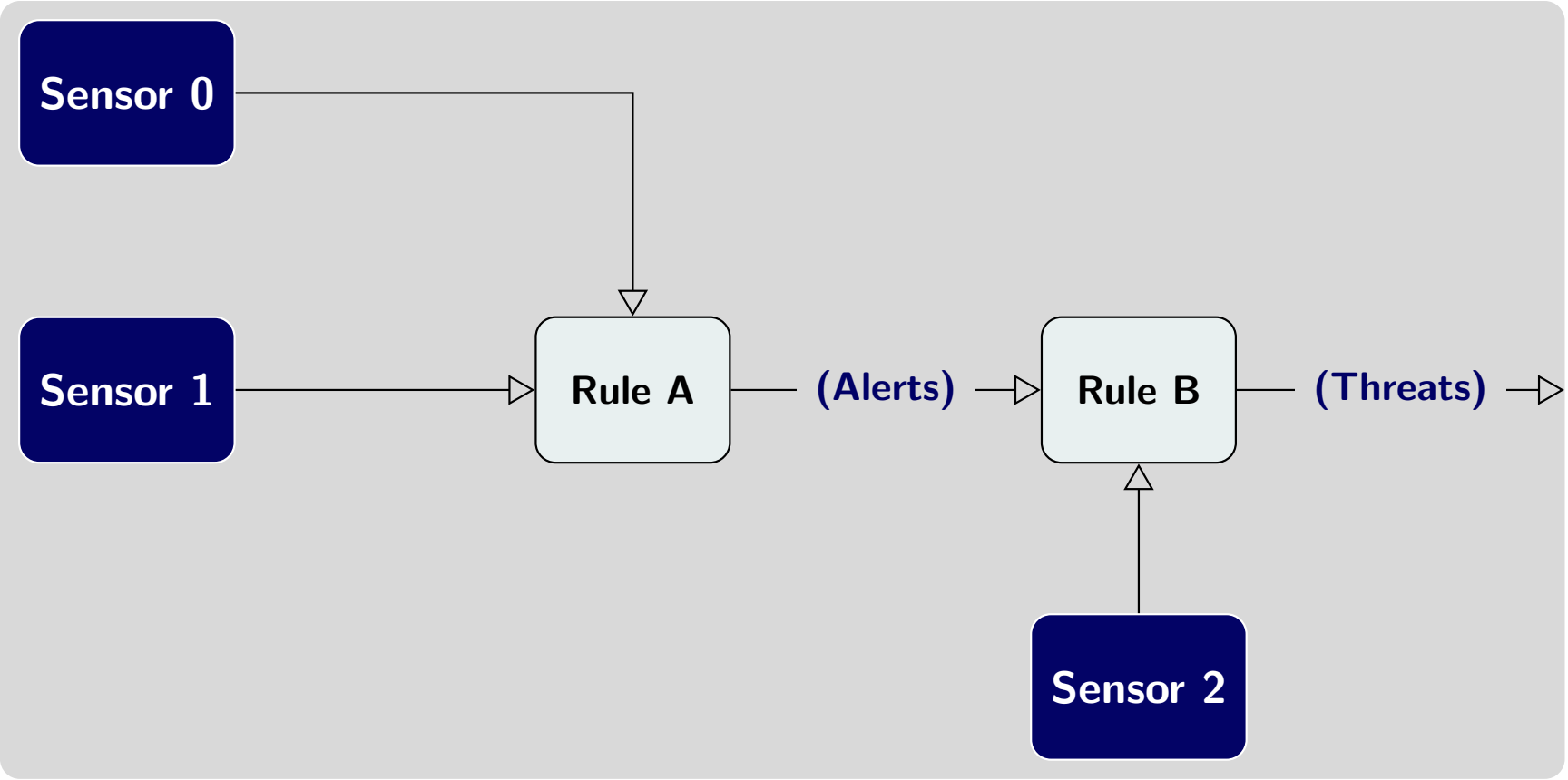
Querying and transforming data

- SQL
- Rule engines
- Complex event processing (CEP)
- Streaming SQL
  - (*Continuous* queries)
  - Provides SQL-interface for CEP
- Map reduce
- Custom programs

Event-Driven Architecture

Consume and produce event streams

- An event typically has an ID, timestamp, and other information



Enablers: Linked Data

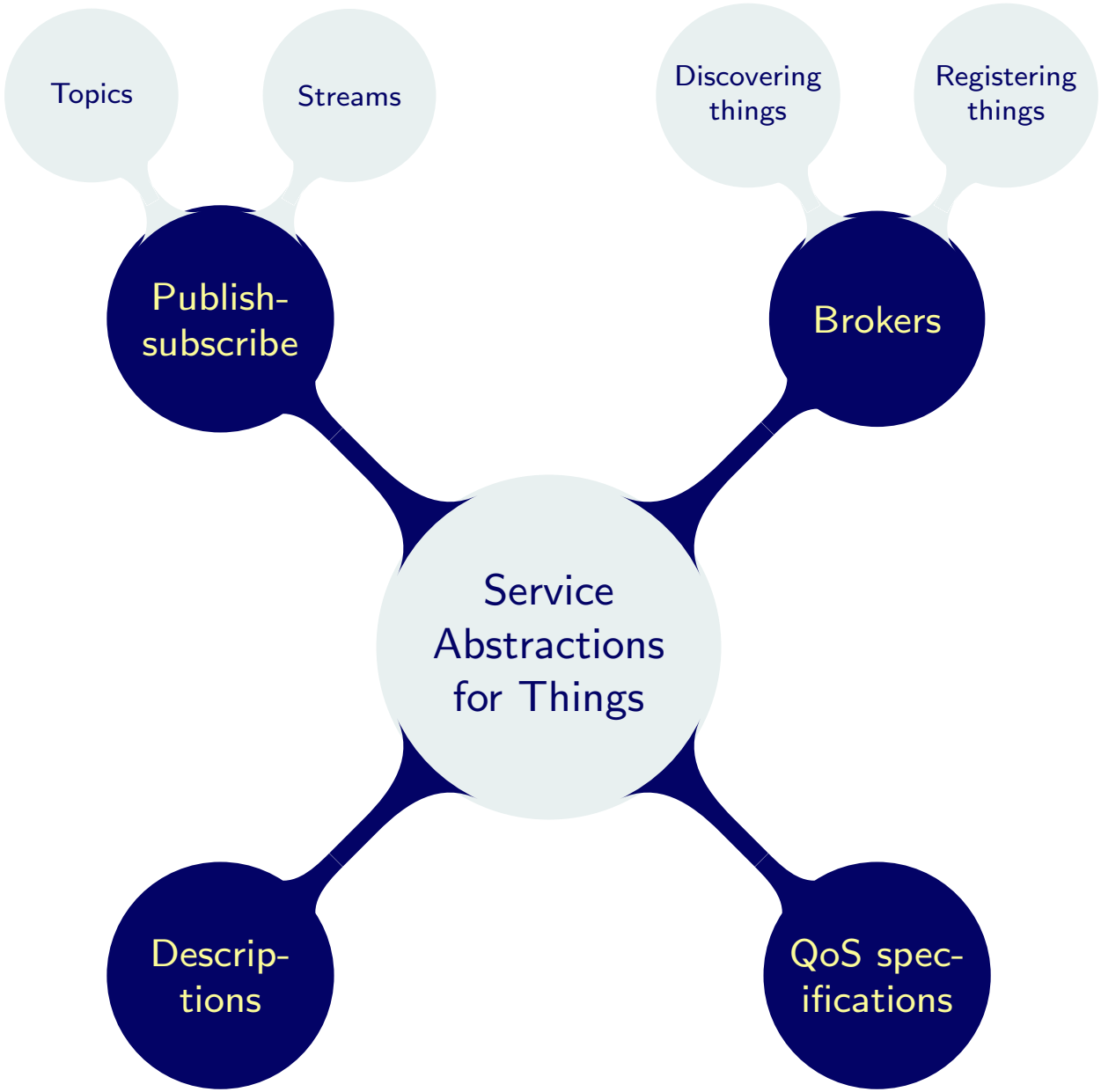
Publishing data

- URIs identify resources
- Deferencing a URI to produce a resource description
  - Encoded in a Semantic Web language, such as the Resource Description Framework, RDF
  - Including URIs for relevant resources
- Exploit underlying architecture, e.g., Domain Name System and Web servers, to locate resources

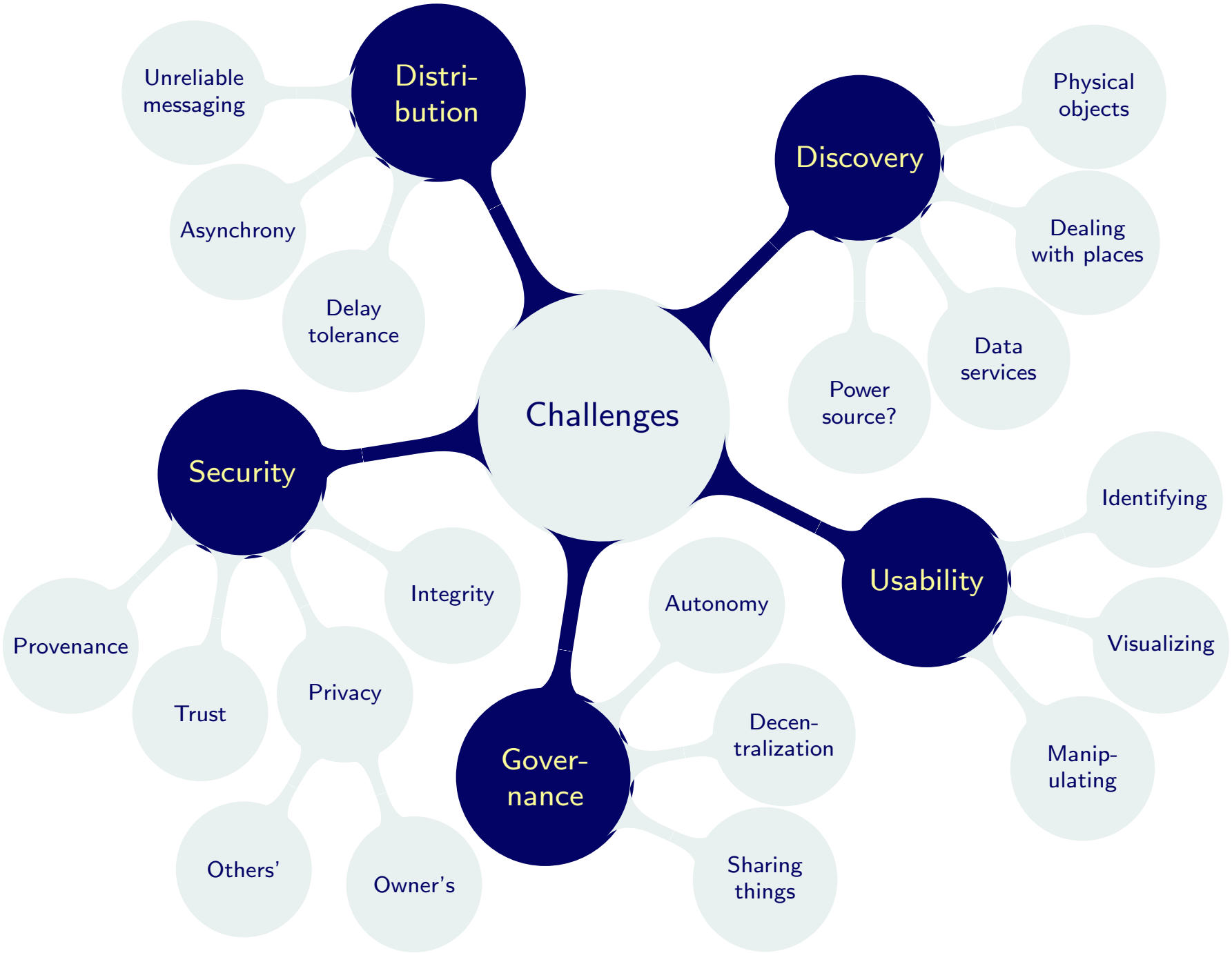
Enablers: Linked Data

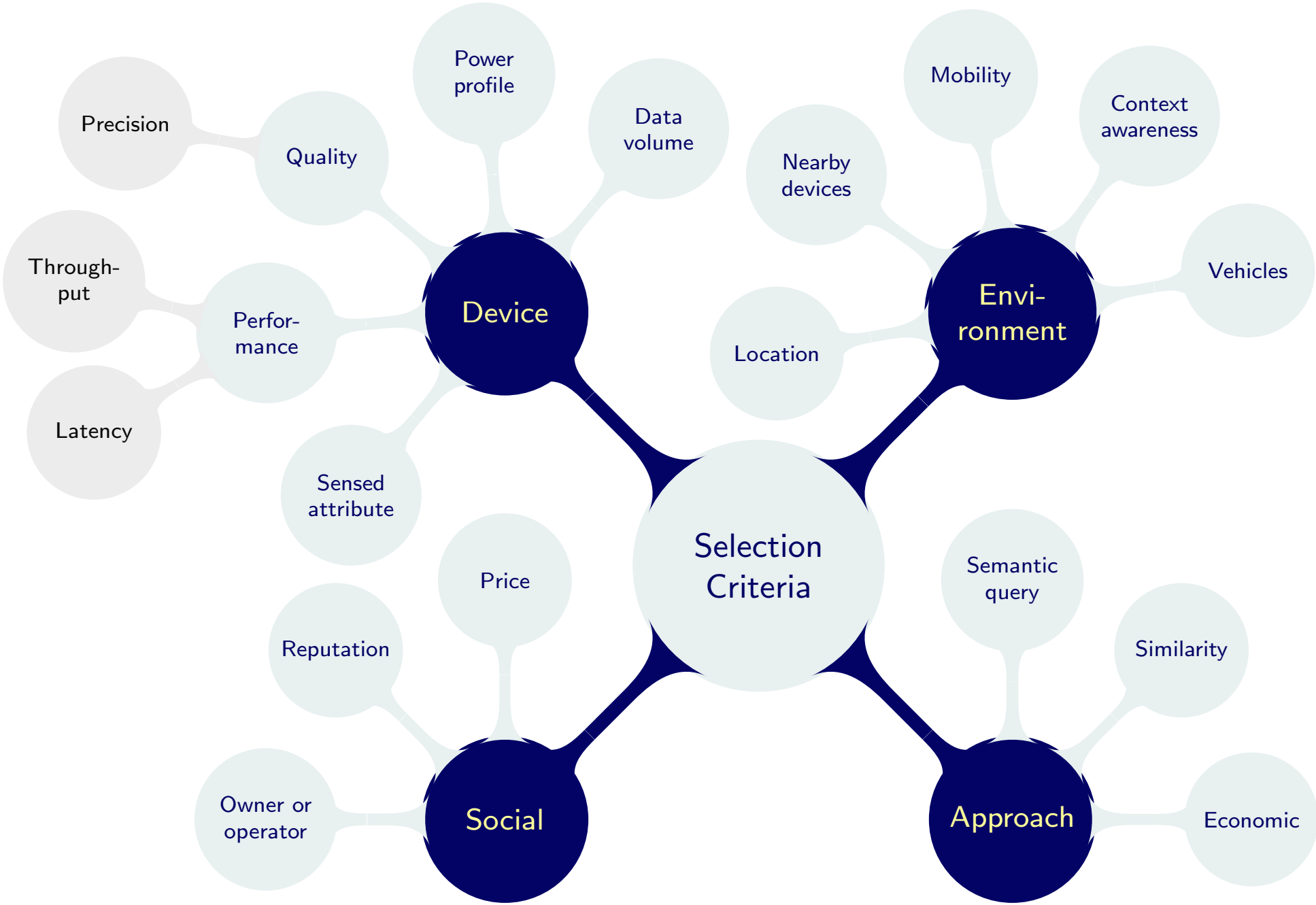
Publishing data

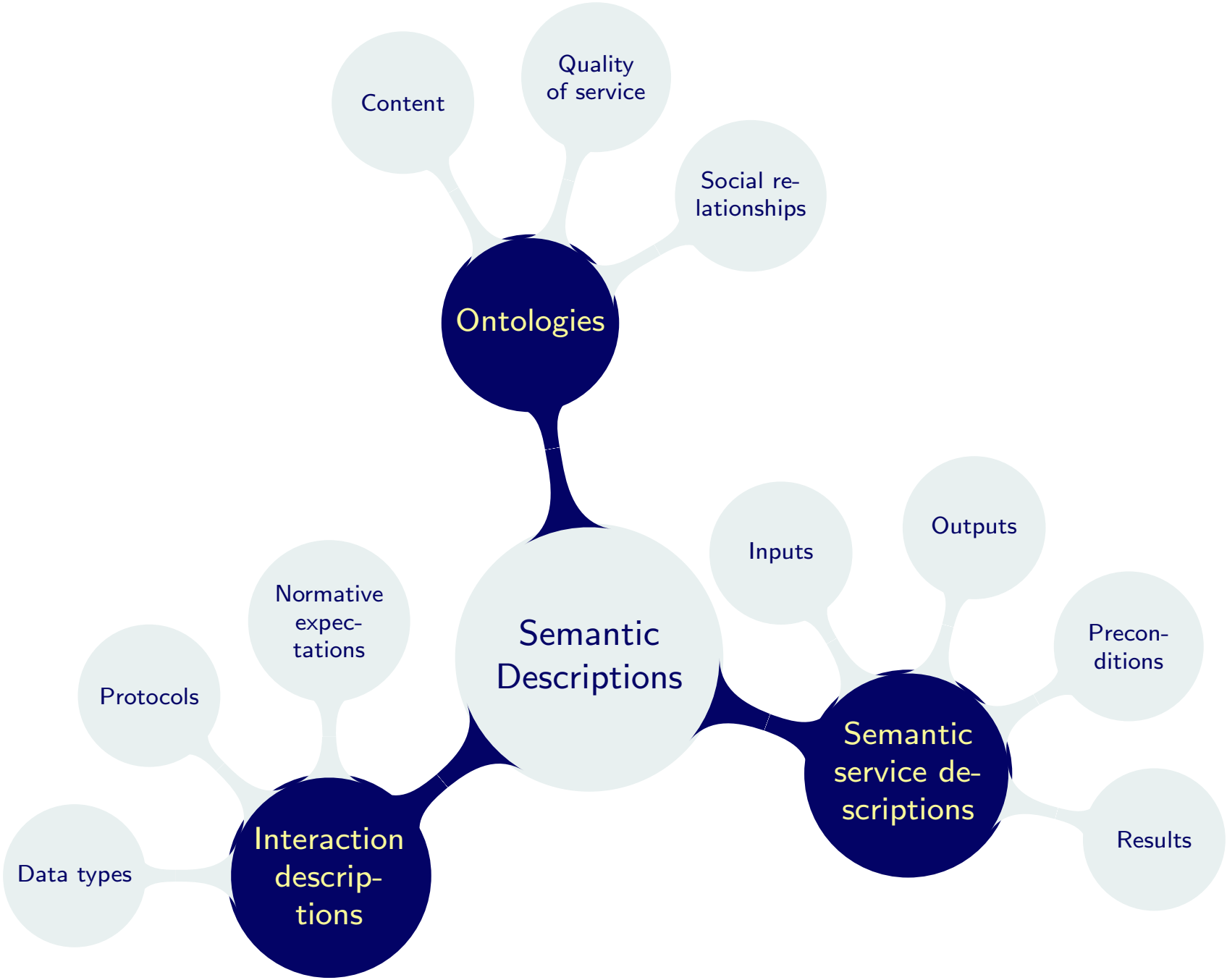
- Describe things and enable their discovery
- Describe the information produced by things
- Enable things to consume and exchange information
  - Enables reasoning
  - Promotes interoperability

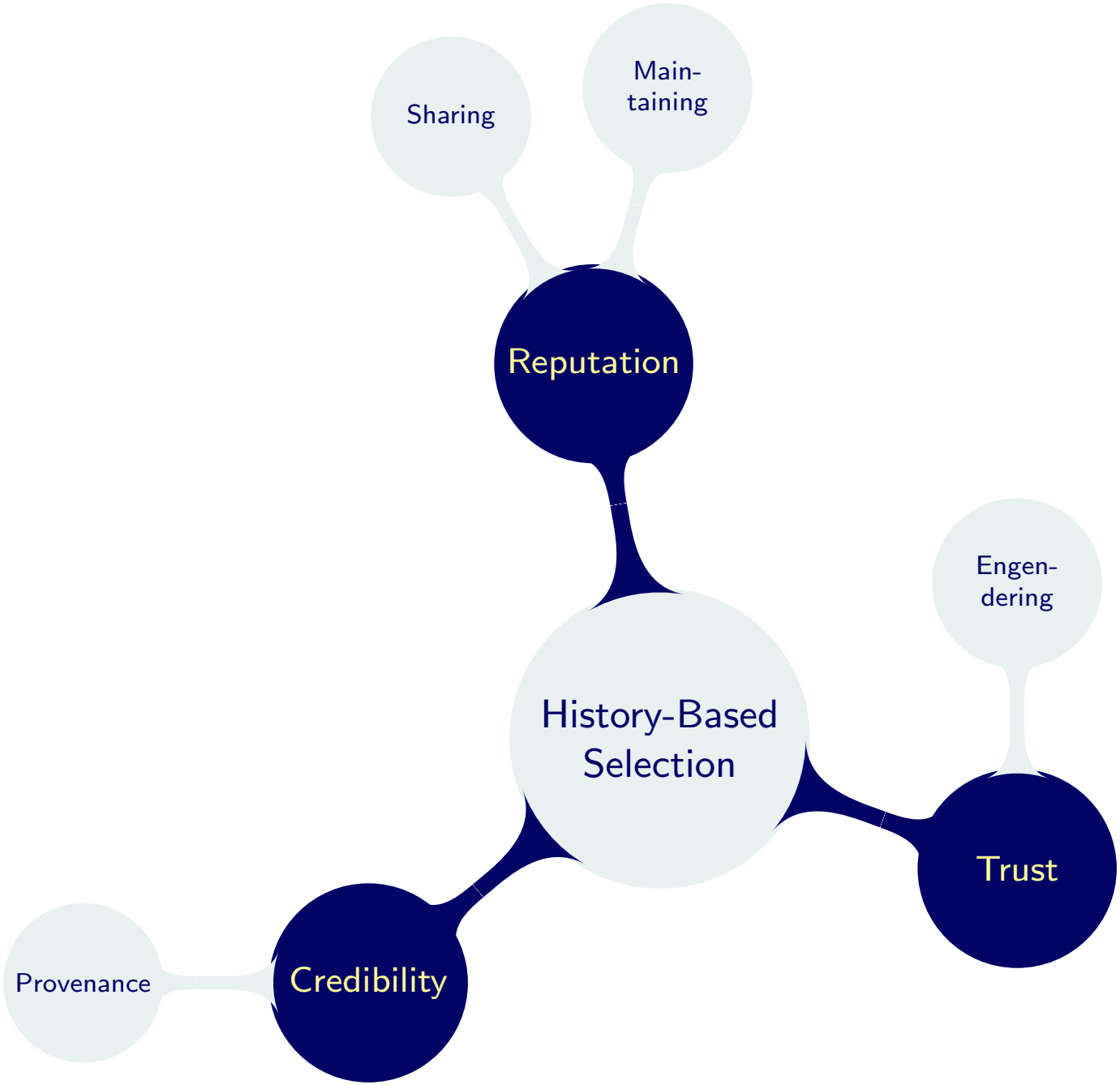

Topics
Streams
Discovering things
Registering things
Publish-subscribe
Brokers
Service Abstractions for Things
Descrip-tions
QoS spec-ifications

Outline


Challenges
Distri-bution
Unreliable messaging
Asynchrony
Delay tolerance
Discovery
Physical objects
Dealing with places
Data services
Power source?
Security
Integrity
Provenance
Trust
Privacy
Gover-nance
Autonomy
Decen-tralization
Sharing things
Others'
Owner's
Usability
Identifying
Visualizing
Manip-ulating


Selection Criteria
Device
Power profile
Data volume
Quality
Precision
Perfor-mance
Through-put
Latency
Sensed attribute
Environment
Mobility
Context awareness
Nearby devices
Vehicles
Location
Social
Price
Reputation
Owner or operator
Approach
Semantic query
Similarity
Economic


Content
Quality of service
Social relationships
Ontologies
Inputs
Outputs
Normative expectations
Semantic Descriptions
Protocols
Preconditions
Semantic service descriptions
Data types
Interaction descriptions
Results


Sharing
Main-
taining
Reputation
Engen-
dering
History-Based
Selection
Trust
Provenance
Credibility

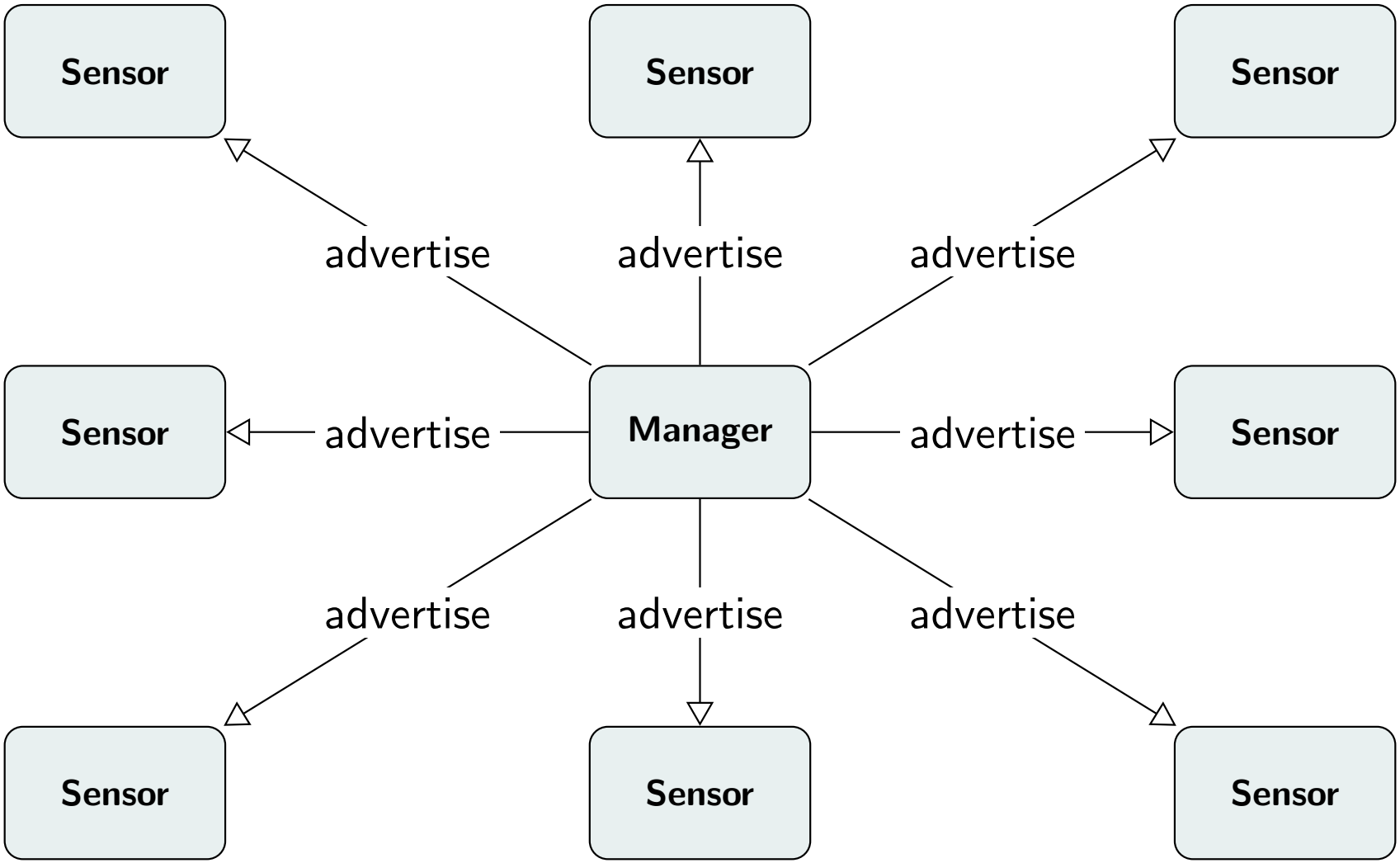
Outline

Sharing, Fusing, Revising Information

- Folk wisdom about aggregating knowledge
- Condorcet Jury Theorem
  - If each information source is better than 50% accurate (binary case)
  - Then their majority is even more accurate (depending on how many)
- Galton's crowdsourcing

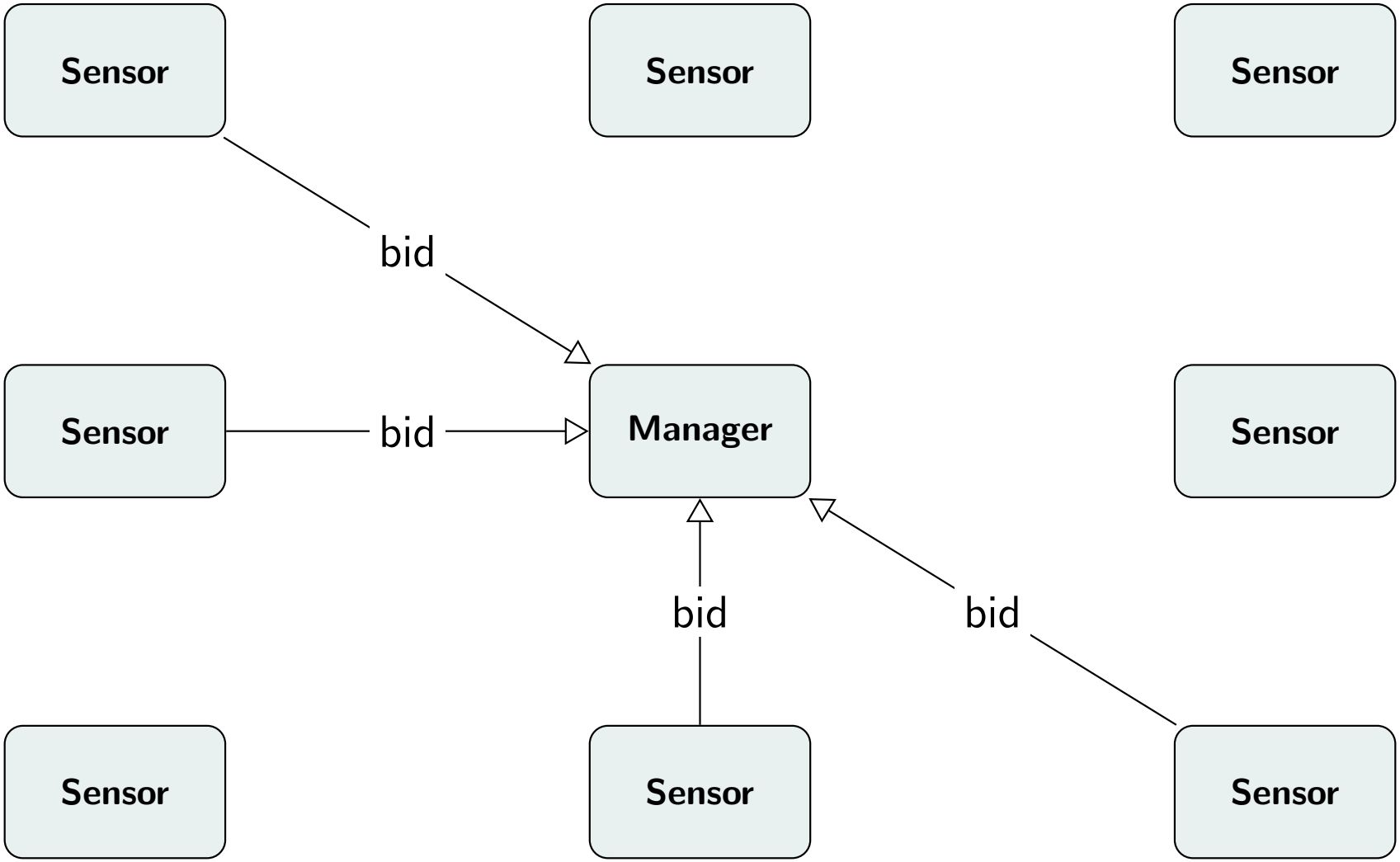
Contract Net Protocol: 1

Announce



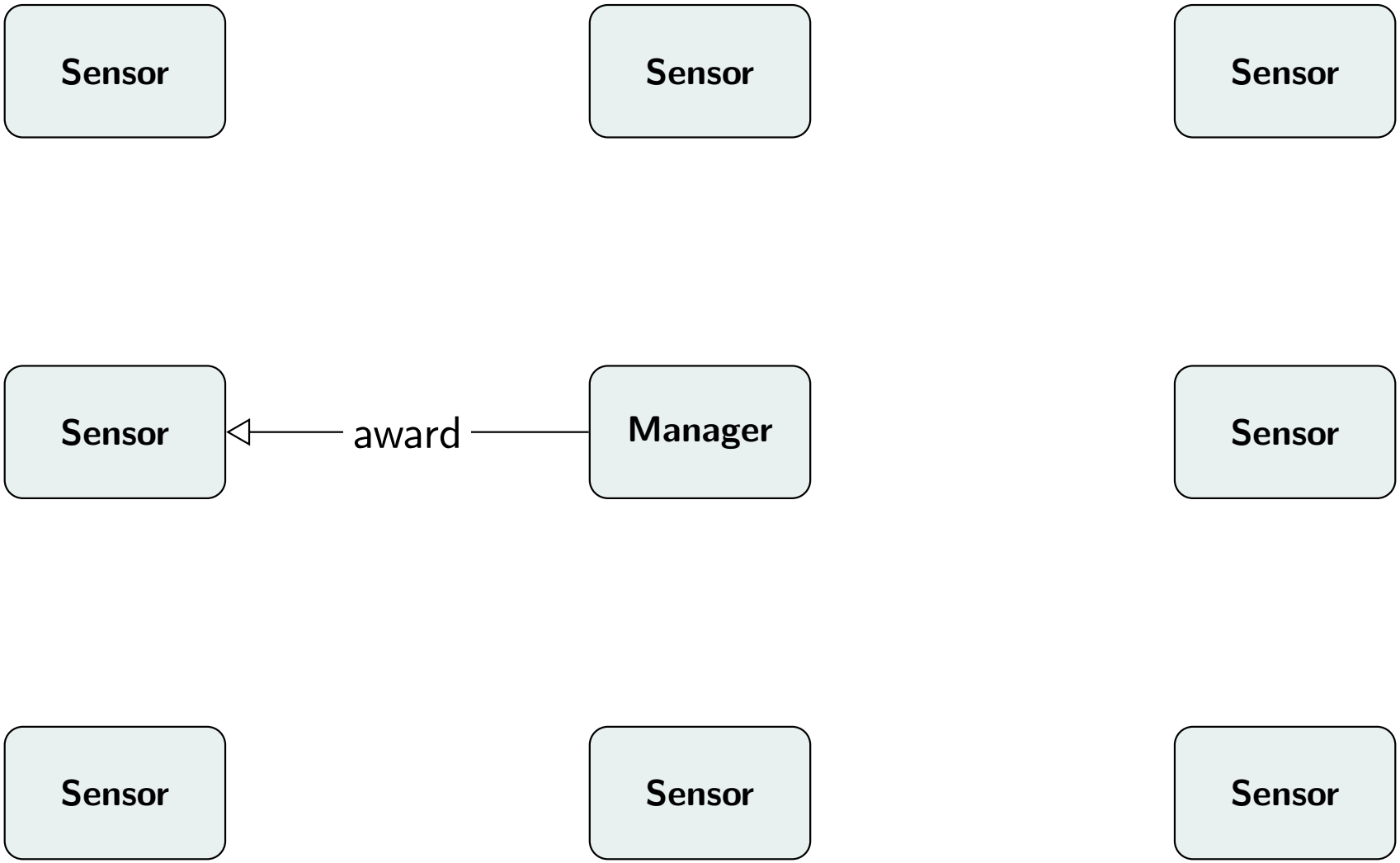
Contract Net Protocol: 2

Bid



Contract Net Protocol: 3

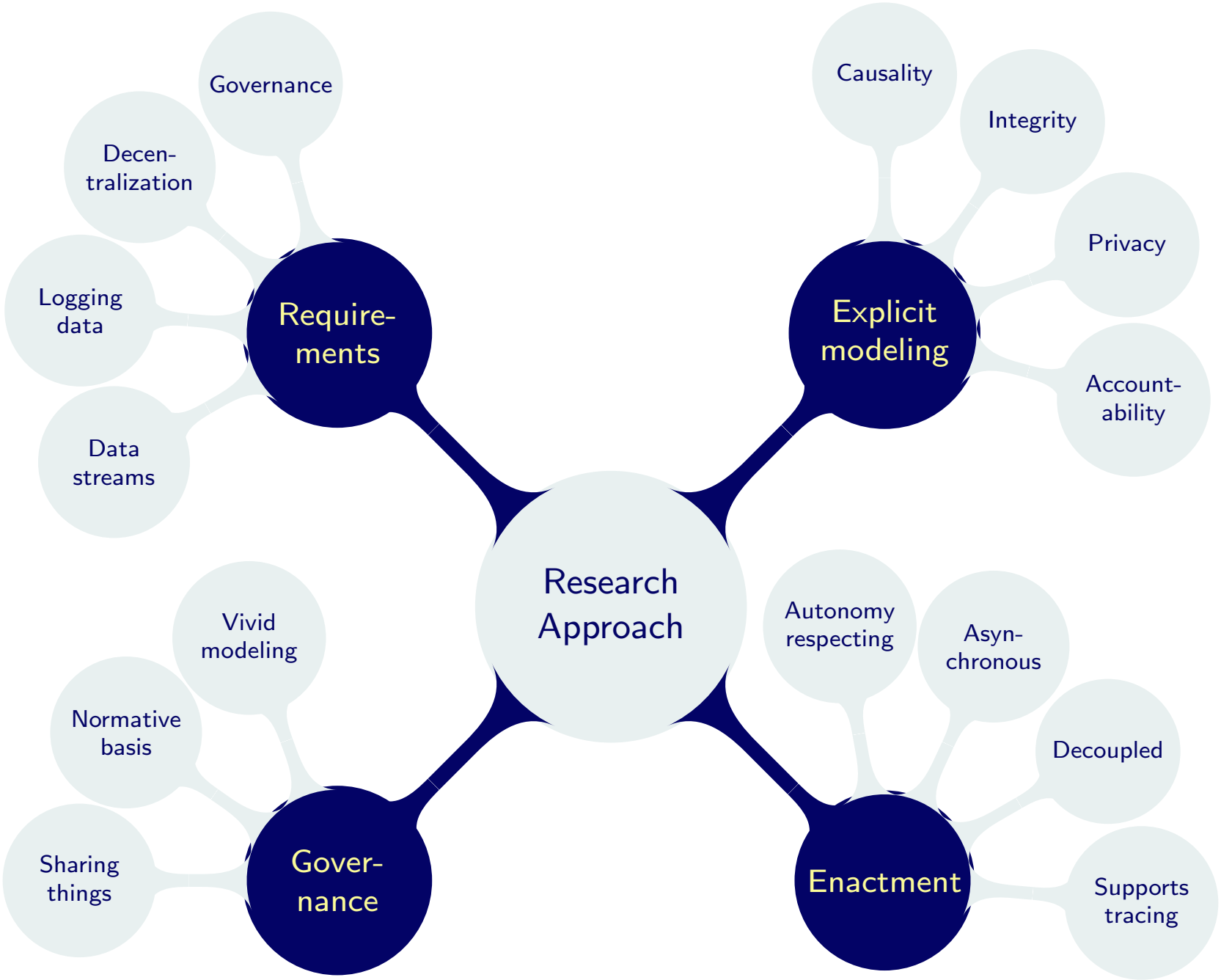
Award



Contract Net Protocol

- Generic protocol
  - Can apply recursively
- Supports important properties
  - Robustness against sensor failure
  - Robustness against connectivity failure
  - Incorporating economic features
- Not well specified
  - Confuses interaction and internal reasoning
  - Limited to two-party interactions

Outline

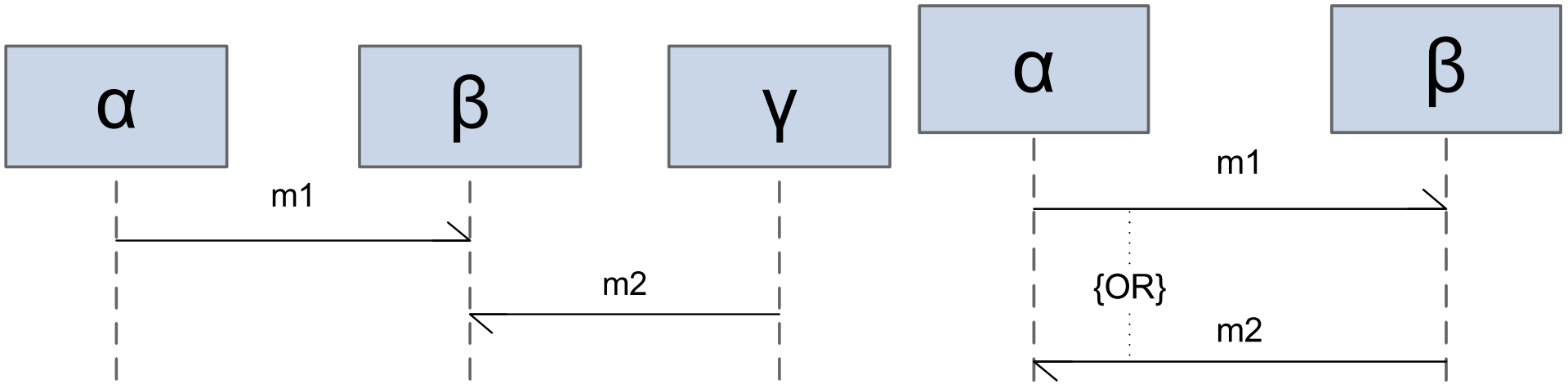

Research Approach
Require-ments
Governance
Decen-tralization
Logging data
Data streams
Explicit modeling
Causality
Integrity
Privacy
Account-ability
Gover-nance
Vivid modeling
Normative basis
Sharing things
Enactment
Autonomy respecting
Asyn-chronous
Decoupled
Supports tracing

Properties of Participants

- Autonomy
- Myopia
  - All choices must be local
  - Correctness should not rely on future interactions
- Heterogeneity: local $\neq$ internal
  - Local state (projection of global, which is stored nowhere)
    - Public or observable
    - Typically, must be revealed for correctness
  - Internal state
    - Private
    - Must never be revealed to avoid false coupling
- Shared nothing representation of local state
  - Enact via messaging

Traditional Specifications

Low-level, procedural approaches leading to over-specified protocols



- Traditional approaches
  - Emphasize arbitrary ordering and occurrence constraints
  - Then work hard to deal with those constraints
- Our philosophy: The Zen of Distributed Computing
  - Necessary ordering constraints fall out from *causality*
  - Necessary occurrence constraints fall out from *integrity*
  - Unnecessary constraints: simply *ignore* such

BSPL, the Blindingly Simple Protocol Language

Main ideas

- Only *two* syntactic notions
  - Declare a message schema: as an atomic protocol
  - Declare a composite protocol: as a bag of references to protocols
- Parameters are central
  - Provide a basis for expressing meaning in terms of bindings in protocol instances
  - Yield unambiguous specification of compositions through public parameters
  - Capture progression of a role's knowledge
  - Capture the completeness of a protocol enactment
  - Capture uniqueness of enactments through keys
- Separate structure (parameters) from meaning (bindings)
  - Capture many important constraints purely structurally

Key Parameters in BSPL

Marked as ⌜key⌝

- All the key parameters *together* form the key
- Each protocol must define at least one key parameter
- Each message or protocol reference must have at least one key parameter in common with the protocol in whose declaration it occurs
- The key of a protocol provides a basis for the uniqueness of its enactments

Parameter Adornments in BSPL

Capture the essential causal structure of a protocol (for simplicity, assume all parameters are strings)

- ⌜in⌝: Information that must be provided to instantiate a protocol
  - Bindings must exist locally in order to proceed
  - Bindings must be produced through some other protocol
- ⌜out⌝: Information that is generated by the protocol instances
  - Bindings can be fed into other protocols through their ⌜in⌝ parameters, thereby accomplishing composition
  - A standalone protocol must adorn all its public parameters ⌜out⌝
- ⌜nil⌝: Information that is absent from the protocol instance
  - Bindings must not exist

The *Hello* Protocol

```
Hello {
 role Self, Other
 parameter out greeting key

 Self ↦ Other: hi[out greeting key]
}
```

- At most one instance of *Hello* for each greeting
- At most one *hi* message for each greeting
- Enactable standalone: no parameter is ⌜in⌝
- The key of *hi* is explicit; often left implicit on messages

The *Pay* Protocol

```
Pay {
 role Payer, Payee
 parameter in ID key, in amount

 Payer ↦ Payee: payM[in ID, in amount]
}
```

- At most one *payM* for each ID
- Not enactable standalone: **why?**
- The key of *payM* is implicit; could be made explicit
- Eliding ⌜means⌝ clauses in this paper

The *Offer* Protocol

```
Offer {
 role Buyer, Seller
 parameter in ID key, out item, out price

 Buyer ↦ Seller: rfq[in ID, out item]
 Seller ↦ Buyer: quote[in ID, in item, out price]
}
```

- The key ID uniquifies instances of *Initiate Offer*, *rfq*, and *quote*
- Not enactable standalone: at least one parameter is ⌜in⌝
- An instance of *rfq* must precede any instance of *quote* with the same ID: **why?**
- No message need occur: **why?**
- *quote* must occur for *Offer* to complete: **why?**

The *Initiate Order* Protocol

```
Initiate-Order {
 role B, S
 parameter out ID key, out item, out price, out rID

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]

 B ↦ S: accept[in ID, in item, in price, out rID]
 B ↦ S: reject[in ID, in item, in price, out rID]
}
```

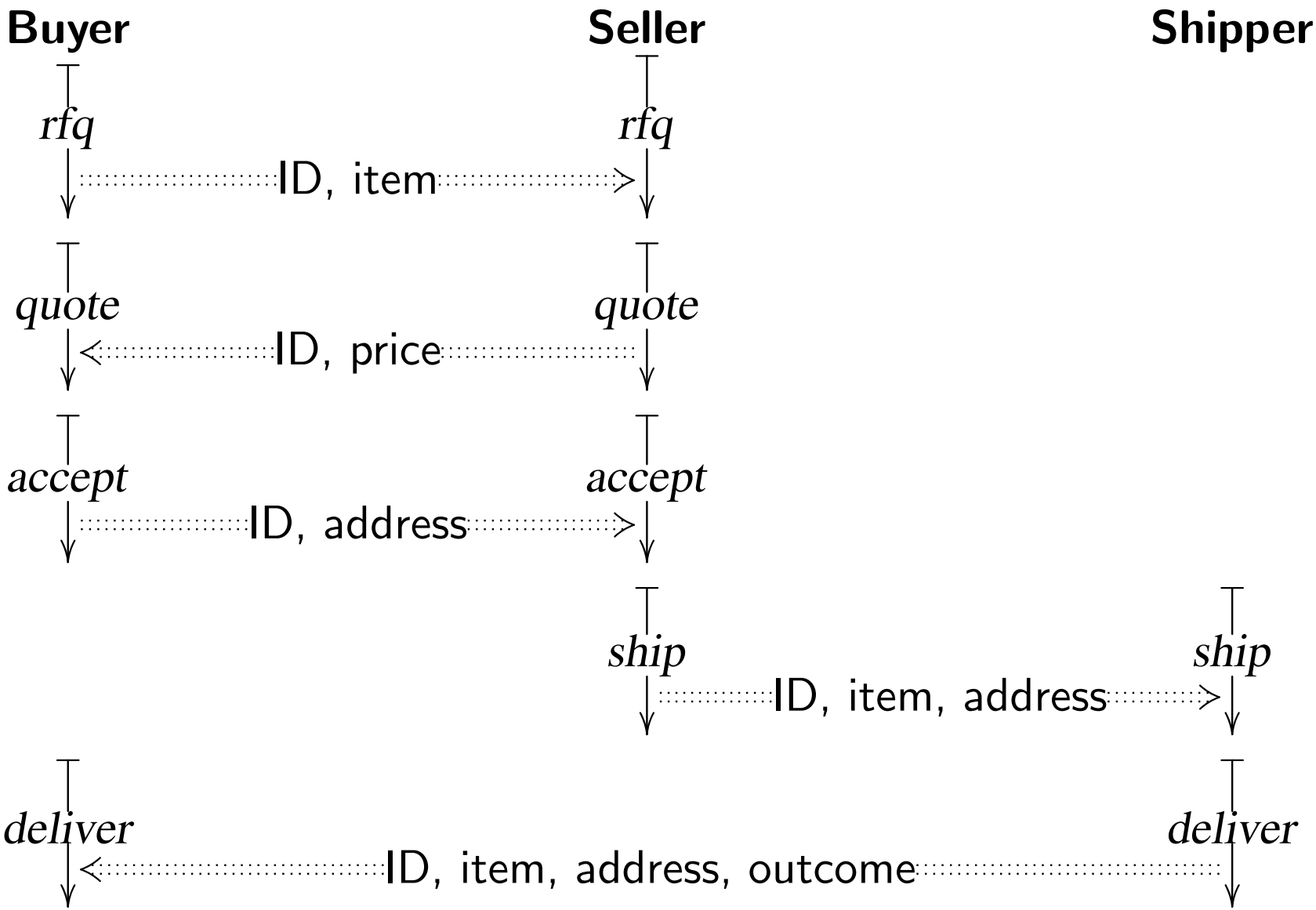
- The key ID uniquifies instances of *Order* and each of its messages
- Enactable standalone
- An *rfq* must precede a *quote* with the same ID
- A *quote* must precede an *accept* with the same ID
- A *quote* must precede a *reject* with the same ID
- An *accept* and a *reject* with the same ID *cannot* both occur: **why?**

The *Purchase* Protocol

```
Purchase {
 role B, S, Shipper
 parameter out ID key, out item, out price, out outcome
 private address, resp

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
 B ↦ S: accept[in ID, in item, in price, out address, out resp]
 B ↦ S: reject[in ID, in item, in price, out outcome, out resp]

 S ↦ Shipper: ship[in ID, in item, in address]
 Shipper ↦ B: deliver[in ID, in item, in address, out outcome]
}
```

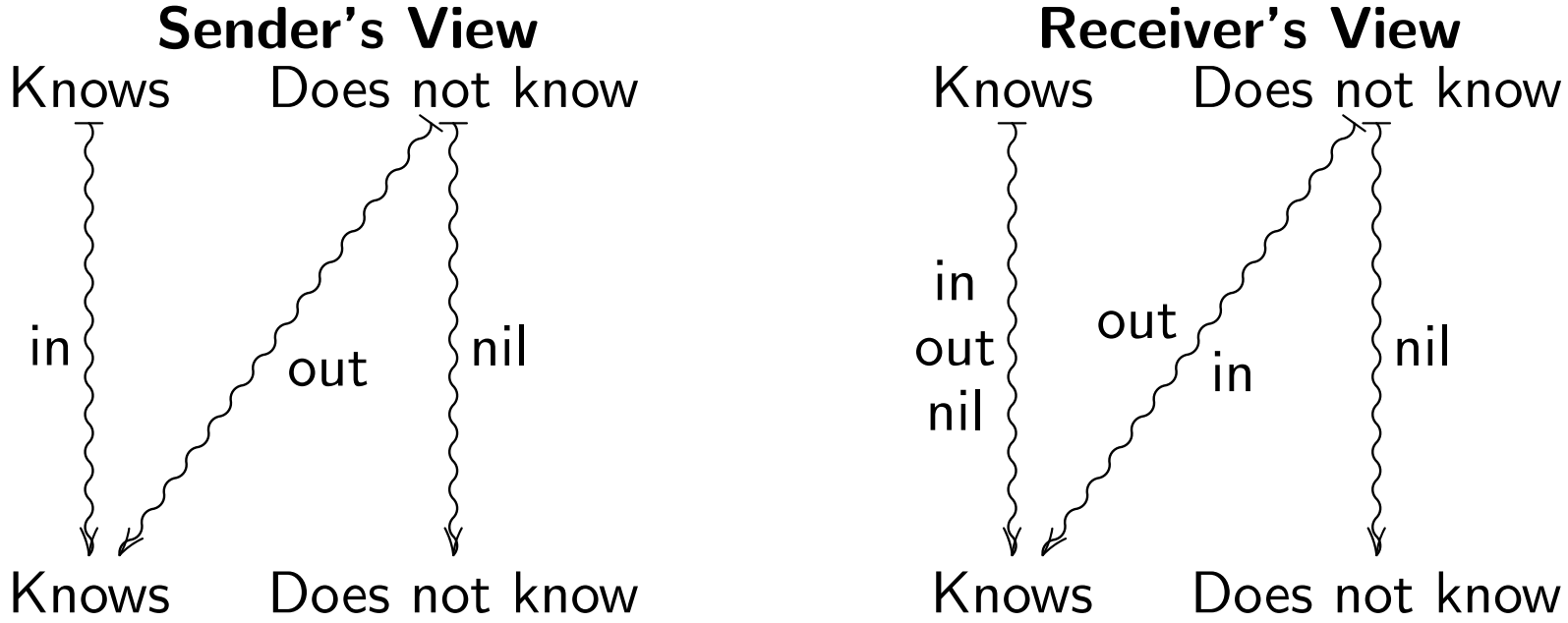
- At most one item, price, and outcome binding per ID
- Enactable standalone
- *reject* conflicts with *accept* on response (a *private* parameter)
- *reject* or *deliver* must occur for completion (to bind outcome)

Possible Enactment as a Vector of Local Histories



Knowledge and Viability

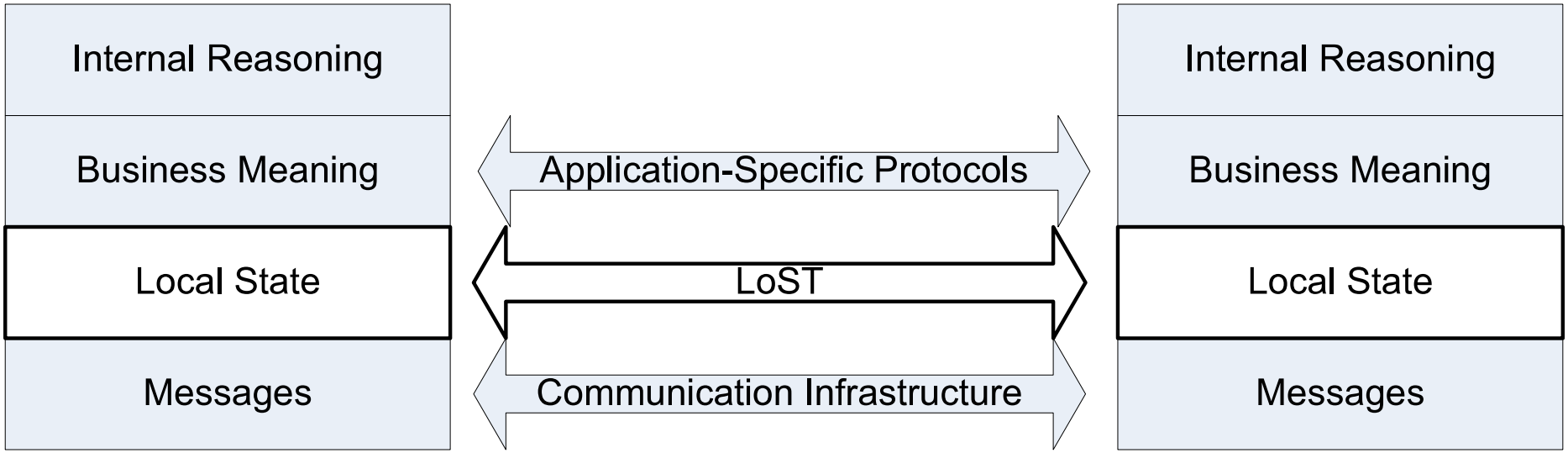
When is a message viable? What effect does it have on a role's local knowledge?



- Knowledge increases monotonically at each role
- An ⌜out⌝ parameter **creates** and transmits knowledge
- An ⌜in⌝ parameter transmits knowledge
- Repetitions through multiple paths are harmless and superfluous

Realizing BSPL via LoST

LoST = Local State Transfer



- Does *not* assume FIFO or reliable messaging
- Provides
  - Unique messages
  - Integrity checks on incoming messages
  - Consistency of local choices on outgoing messages

Implementing LoST

Think of the message logs you want

- For each role
  - For each message that it sends or receives
    - Maintain a *local* relation of the same schema as the message
- Receive and store any message provided
  - It is not a duplicate
  - Its integrity checks with respect to parameter bindings
  - Garbage collect expired sessions: requires additional annotations
- Send any unique message provided
  - Parameter bindings agree with previous bindings for the same keys for ⌜in⌝ parameters
  - No bindings for ⌜out⌝ and ⌜nil⌝ parameters exist

Comparing LoST and ReST

	ReST	*LoST*
Modality	Two-party; client-server; synchronous	Multiparty interactions; peer-to-peer; asynchronous
Computation	Server computes definitive resource state	Each party computes its definitive local state and the parties collaboratively and (potentially implicitly) compute the definitive interaction state
State	Server maintains no client state	Each party maintains its local state and, implicitly, the relevant components of the states of other parties from which there is a chain of messages to this party

Comparing LoST and ReST

	ReST	*LoST*
Transfer	State of a resource, suitably represented	Local state of an interaction via parameter bindings, suitably represented
Idempotent	For some verbs, especially GET	Always; repetitions are guaranteed harmless
Caching	Programmer can specify if cacheable	Always cacheable
Uniform interface	GET, POST, ...	⌜in⌝, ⌜out⌝, ⌜nil⌝
Naming	Of resources via URIs	Of interactions via (composite) keys, whose bindings could be URIs

Remark on Control versus Information Flow

- Control flow
  - Natural within a single computational thread
  - Exemplified by conditional branching
  - Presumes master-slave relationship across threads
  - Impossible between mutually autonomous parties because neither controls the other
  - May sound appropriate, but only because of long habit
- Information flow
  - Natural across computational threads
  - Explicitly tied to causality

Bliss Conceptual Model: Functions of Parameters

- Key
  - For interaction instantiation and uniqueness
- Payload
  - For interaction meaning
- Completion
  - To help determine when the interaction is over
- Integrity
  - For interaction integrity
- Control
  - To force certain preferred orders of enactment

BSPL

Taking a declarative, information-centric view of interaction to the limit

- Specification
  - A message is an atomic protocol
  - A composite protocol is a set of references to protocols
  - Each protocol is given by a name and a set of parameters (including keys)
  - Each protocol has *inputs* and *outputs*
- Representation
  - A protocol corresponds to a relation (table)
  - Integrity constraints apply on the relations
- Enactment via LoST: Local State Transfer
  - Information represented: local $\neq$ internal
  - Purely decentralized at each role
  - Materialize the relations *only* for messages

Information Centrism

Characterize each interaction purely in terms of information

- Explicit causality
  - Flow of information coincides with flow of causality
  - No hidden control flows
  - No backchannel for coordination
- Keys
  - Uniqueness
  - Basis for completion
- Integrity
  - Must have bindings for some parameters
  - Analogous to NOT NULL constraints
- Immutability
  - Durability
  - Robustness: insensitivity to
    - Reordering by infrastructure
    - Retransmission: one delivery is all it needs

Outline

Safety: *Purchase Unsafe*

Remove conflict between *accept* and *reject*

```
Purchase Unsafe {
 role B, S, Shipper
 parameter out ID key, out item, out price, out outcome
 private address, resp

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
 B ↦ S: accept[in ID, in item, in price, out address]
 B ↦ S: reject[in ID, in item, in price, out outcome]

 S ↦ Shipper: ship[in ID, in item, in address]
 Shipper ↦ B: deliver[in ID, in item, in address, out outcome]
}
```

- B can send both *accept* and *reject*
- Thus outcome can be bound twice in the same enactment

Liveness: *Purchase No Ship*

Omit *ship*

```
Purchase No Ship {
 role B, S, Shipper
 parameter out ID key, out item, out price, out outcome
 private address, resp

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
 B ↦ S: accept[in ID, in item, in price, out address, out resp]
 B ↦ S: reject[in ID, in item, in price, out outcome, out resp]

 Shipper ↦ B: deliver[in ID, in item, in address, out outcome]
}
```

- If B sends *reject*, the enactment completes
- If B sends *accept*, the enactment deadlocks

Encode Causal Structure as Temporal Constraints

- *Reception*. If a message is received, it was previously sent.
- *Information transmission* (sender's view)
  - Any ⌜in⌝ parameter occurs prior to the message
  - Any ⌜out⌝ parameter occurs simultaneously with the message
- *Information reception* (receiver's view)
  - Any ⌜out⌝ or ⌜in⌝ parameter occurs before or simultaneously with the message
- *Information minimality*. If a role observes a parameter, it must be simultaneously with *some* message sent or received
- *Ordering*. If a role sends any two messages, it observes them in some order

Verifying Safety

- Competing messages: those that have the same parameter as out
- *Conflict*. At least two competing messages occur
- *Safety* iff the causal structure $\wedge$ conflict is unsatisfiable

Verifying Liveness

- *Maximality*. If a role is enabled to send a message, it sends at least one such message
- *Reliability*. Any message that is sent is received
- *Incompleteness*. Some public parameter fails to be bound
- *Live* iff the causal structure $\wedge$ the occurrence is unsatisfiable

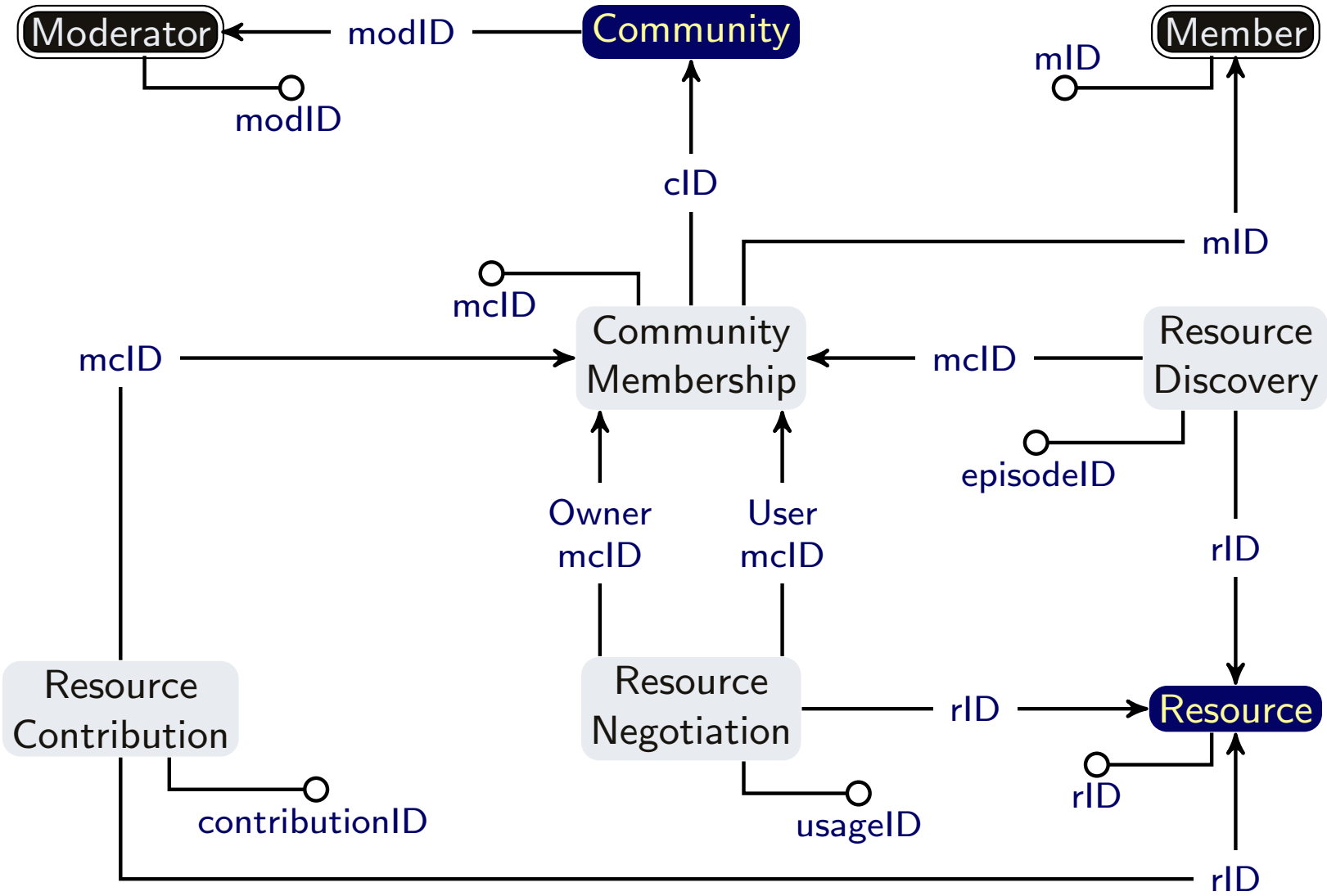
Bliss Methodology

Iterate over the following steps

1. Identify the roles needed in a protocol
2. Identify the conceptual social object computed
3. Identify the messages (or, recursively, subprotocols) to compute the social object
4. Identify each message as a component of the social object and any additional constraints
5. Introduce polymorphism of messages to support flexible sourcing of parameter bindings

Schema for IoT Resource Sharing

Maps to four protocols, naturally composed



The *Community Membership* protocol

```
Community Membership {
 role Mod, Mem // Moderator, Member
 parameter in cID key, in mID key, opt mcID, out outcome
 private request

 Mem ↦ Mod: requestAdmission[in cID, in mID, out request]
 Mod ↦ Mem: admit[in cID, in mID, in request, out mcID, out
     outcome]
 Mod ↦ Mem: deny[in cID, in mID, in request, nil mcID, out
     outcome]
}
```

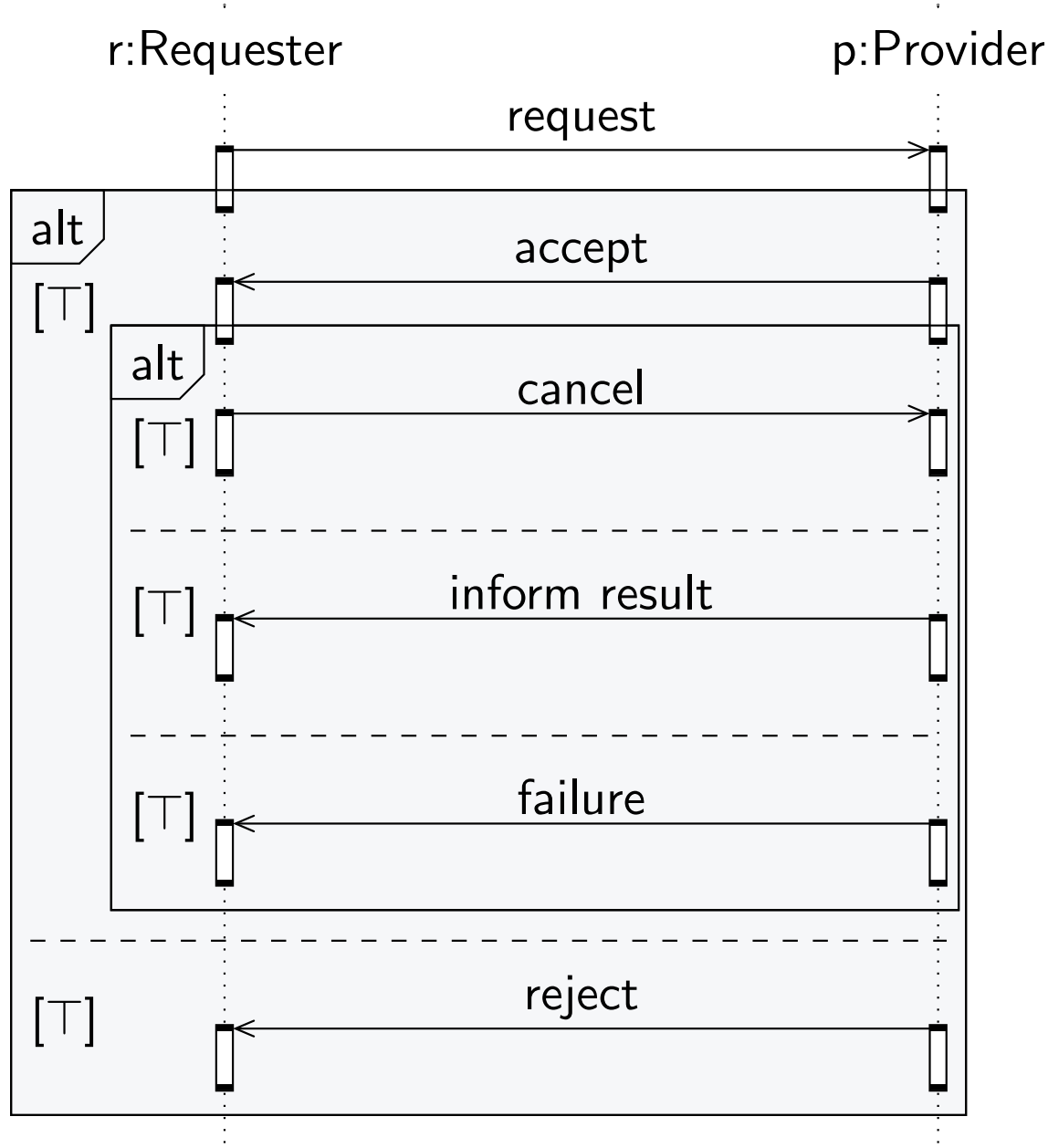
The *Resource Contribution* protocol

```
Resource Contribution {
 role Mod, Mem // Moderator, Member
 parameter in mcID key, in rID key, out contributionID key, out
     outcome
 private rlocation, rType

 Mem ↦ Mod: contribute[in mcID, in rID, out rLocation, out
     rType, out contributionID]
}
```

The *Resource Discovery* protocol

```
Resource Discovery {
 role Mod, Mem // Moderator, Member
 parameter out episodeID key, out rID key, out rOwnerID
 private rlocation, rType

 Mem ↦ Mod: search[out episodeID, out rLocation, out rType]
 Mod ↦ Mem: response[in episodeID, in rLocation, in rType, out
     rOwnerID, out rID]
}
```

Service Request Protocol (Erroneous: Unsafe)

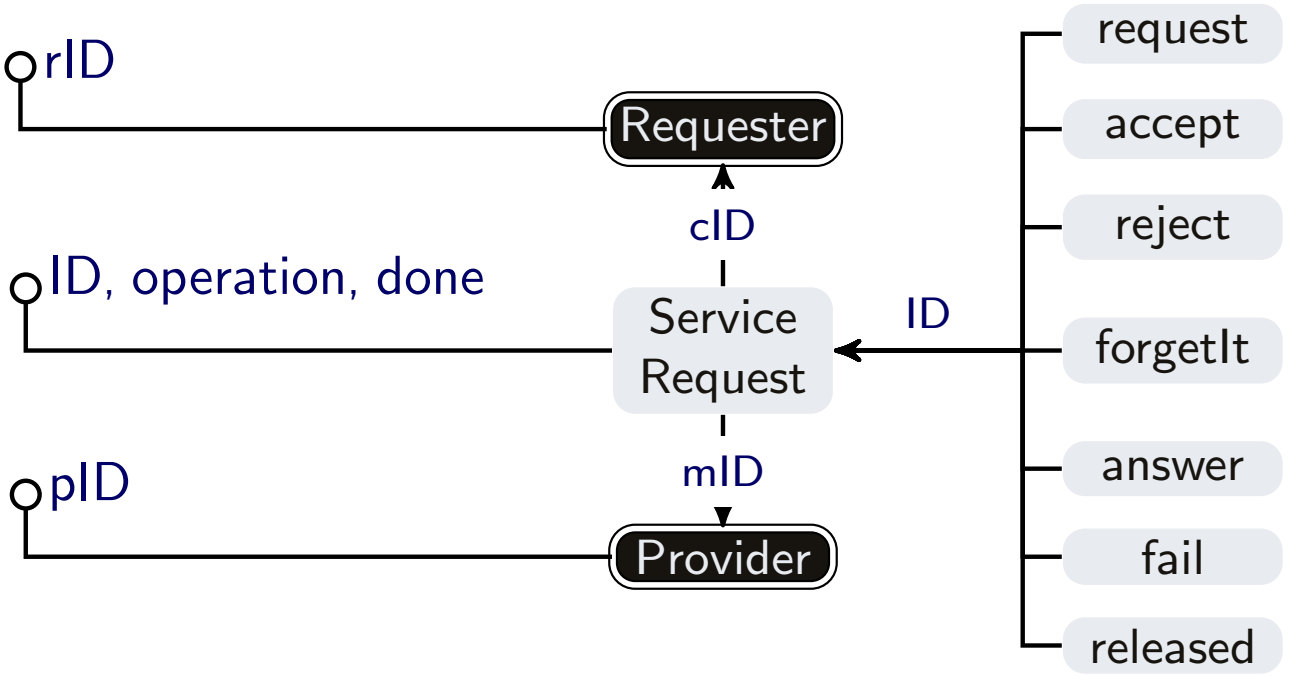


BSPL Reconstruction of Unsafe *Service Request*

Combining some parameters to reduce clutter

```
protocol OOI Service Request Unsafe {
 role R, P
 parameter out ID key, out operation, out result
 private confirmation

 R ↦ P: request[out ID, out operation]
 P ↦ R: accept[in ID, out confirmation]
 P ↦ R: reject[in ID, out confirmation, out result]
 R ↦ P: cancel[in ID, out result]
 P ↦ R: fail[in ID, out result]
 P ↦ R: answer[in ID, out result]
}
```

A Conceptual Schema for *Service Request*



The *Service Request* Protocol Via Bliss, Now Corrected

```
protocol OOI Service Request Corrected {
 role R, P
 parameter out ID key, out operation, out result
 private confirmation, releaseToken

 R ↦ P: request[out ID, out operation]
 P ↦ R: accept[in ID, in operation, out confirmation]
 P ↦ R: reject[in ID, in operation, out confirmation, out
     result]
 R ↦ P: forgetIt[in ID, in operation, in confirmation, out
     releaseToken]
 P ↦ R: answer[in ID, in operation, in confirmation, nil
     releaseToken, out result]
 P ↦ R: fail[in ID, in operation, in confirmation, nil
     releaseToken, out result]
 P ↦ R: released[in ID, in operation, in releaseToken, out
     result]
}
```

in-out Polymorphism

price could be ⌜in⌝ or ⌜out⌝

```
Flexible-Offer {
 role B, S
 parameter in ID key, out item, price, out qID

 B ↦ S: rfq[ID, out item, nil price]
 B ↦ S: rfq[ID, out item, in price]

 S ↦ B: quote[ID, in item, out price, out qID]
 S ↦ B: quote[ID, in item, in price, out qID]
}
```

- The price can be adorned ⌜in⌝ or ⌜out⌝ in a reference to this protocol

The *Bilateral Price Discovery* protocol

```
BPD {
  role Taker, Maker
  parameter out reqID key, out query, out result

  Taker ↦ Maker: priceRequest[out reqID, out query]
  Maker ↦ Taker: priceResponse[in reqID, in query, out result]
}
```

The *Generalized Bilateral Price Discovery* protocol

```
GBPD {
  role T, M
  parameter reqID key, query, res

  T ↦ M: priceRequest[out reqID, out query]
  T ↦ M: priceRequest[in reqID, in query]

  M ↦ T: priceResponse[in reqID, in query, out res]
  M ↦ T: priceResponse[in reqID, in query, in res]
}
```

The *Multilateral Price Discovery* protocol

```
MPD {
  role Taker, Exchange, Maker
  parameter out reqID key, out query, out res

  GBPD(Taker, Exchange, out reqID, out query, in res)
  GBPD(Exchange, Maker, in reqID, in query, out res)
}
```

Standing Order

As in IoT sensor subscription

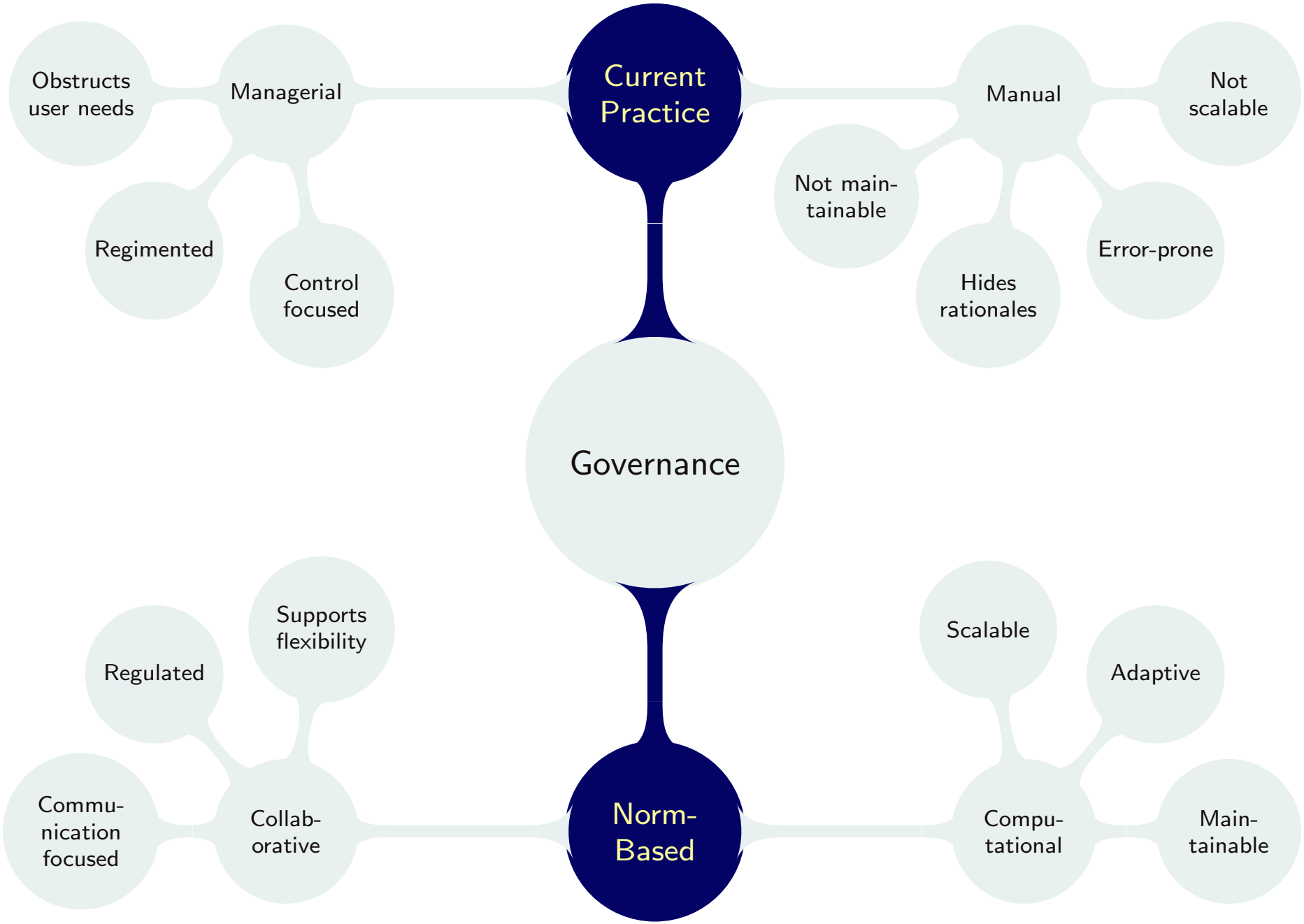
```
Subscription {
 role Sensor, Subscriber
 parameter out contractNO key, out request key, out dataResponse

 Sensor ↦ Subscriber: createContract[out contractNO]
 Subscriber ↦ Sensor: serviceReq[in contractNO, out request]
 Sensor ↦ Subscriber: dataService[in policyNO, in request, out
     dataResponse]
}
```

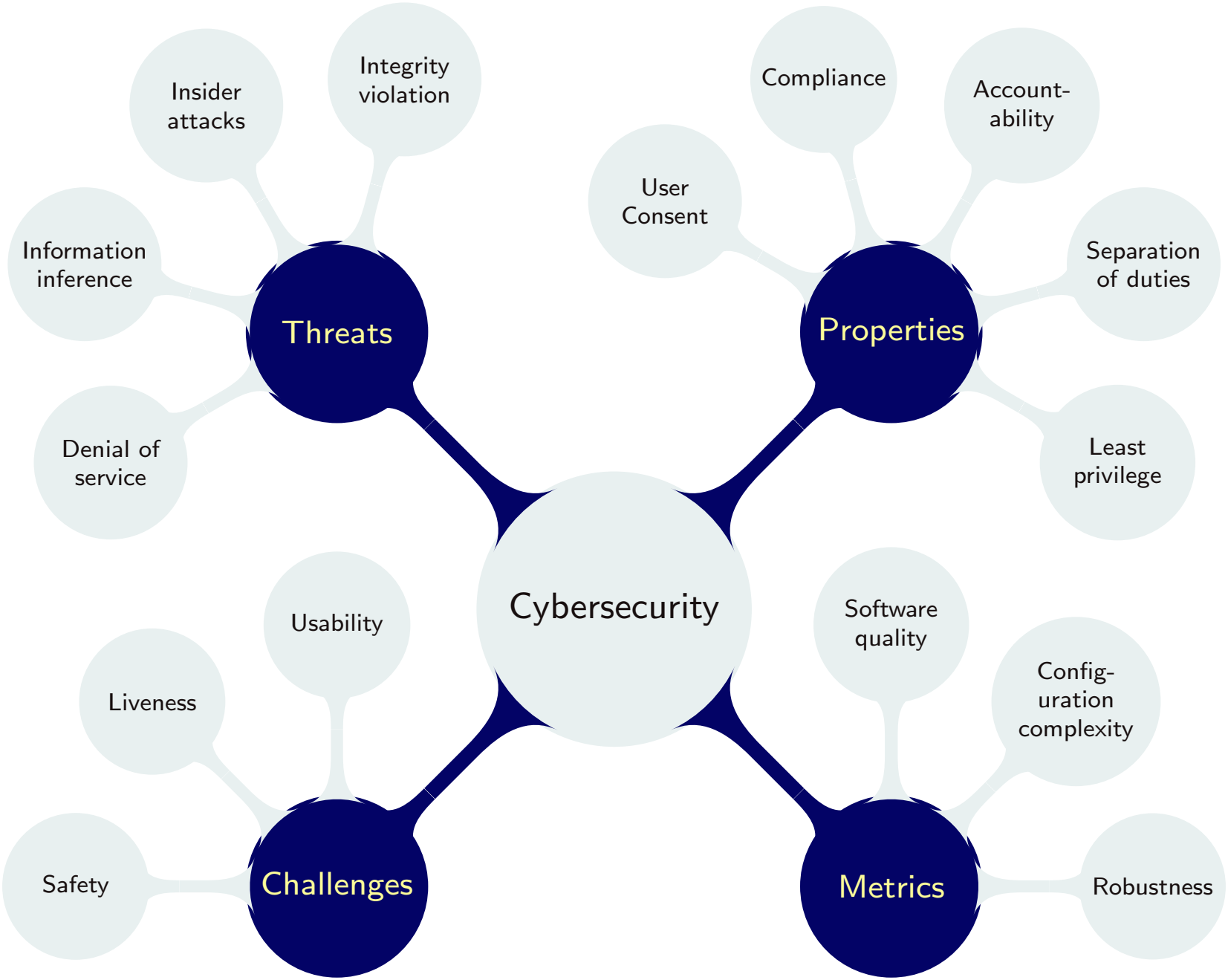
- Each data corresponds to a unique contract and has a unique request
- One contract produce many data responses
- Could make {contractNO, request} both key

The *Bid Offer* protocol

```
Bid Offer {
 role Coordinator uni, Bidder ⊒ Winner uni
 parameter out ID key, out request, out response, out decision

 Coordinator ↦ Bidder: CfB[out ID, out request]

 Bidder ↦ Coordinator: bid[in ID, in request, out response]

 Coordinator ↦ Winner: offer[in ID, in request, in response,
     out decision]
}
```

Outline


Governance
Current Practice
Obstructs user needs
Managerial
Regimented
Control focused
Manual
Not scalable
Not maintainable
Error-prone
Hides rationales
Norm-Based
Regulated
Supports flexibility
Communication focused
Collaborative
Scalable
Adaptive
Computational
Maintainable


Cybersecurity
Threats
Insider attacks
Integrity violation
Information inference
Denial of service
Properties
Compliance
Account-ability
User Consent
Separation of duties
Least privilege
Challenges
Usability
Liveness
Safety
Metrics
Software quality
Config-uration complexity
Robustness

Governance for Secure Collaboration

Broadly, administering sociotechnical systems to serve stakeholder needs

- Currently, automated support comes with managerial imposition: by superiors on subordinates
  - Control over managed resources
  - Necessary but not sufficient
  - Unsuited to many settings
    - When user needs aren't met, they subvert managerial diktats
    - Therefore, vulnerabilities
- Currently, governance is manual via out-of-band communications
  - Low productivity
  - Poor scalability to fine-grained, real-time governance decisions
  - Hidden, implicit considerations yield low confidence in correctness and poor maintainability
    - Lead to errors
    - Therefore, vulnerabilities

Governance Challenges in IoT

Accommodating autonomy, heterogeneity, and dynamism

- Support *configurational adaptation*
  - Resource sharing: Offer ocean instrument for sharing
  - Affiliation: Add new laboratories
  - Sanction: Allow external sharing of results to fulfill deliverables
- Support *Operational adaptation*
  - Resource sharing: Preempt low-priority users in case of oil spill
  - Affiliation: Forbid unilateral publishing of results
  - Sanction: Absolve researcher who reveals results to prevent public endangerment (extenuating circumstances)
- Research challenges
  - Abstractions to capture rules of encounter
  - Methods to design and analyze such abstractions
  - Methods to implement such abstractions

Foundations of Secure Collaboration over IoT

Social perspective that complements technical (data, application, infrastructure) perspectives

- *Normative basis:* Key relationships are reflected in norms
- *Management of social context:* An Org (as a microsociety) recursively provides the context for the norms among and policies of its members
- *Policy:* An implementation-independent model of decision making and operational semantics
- *Interaction orientation:* How agents apply policies to enter into, monitor, and enact normative relationships

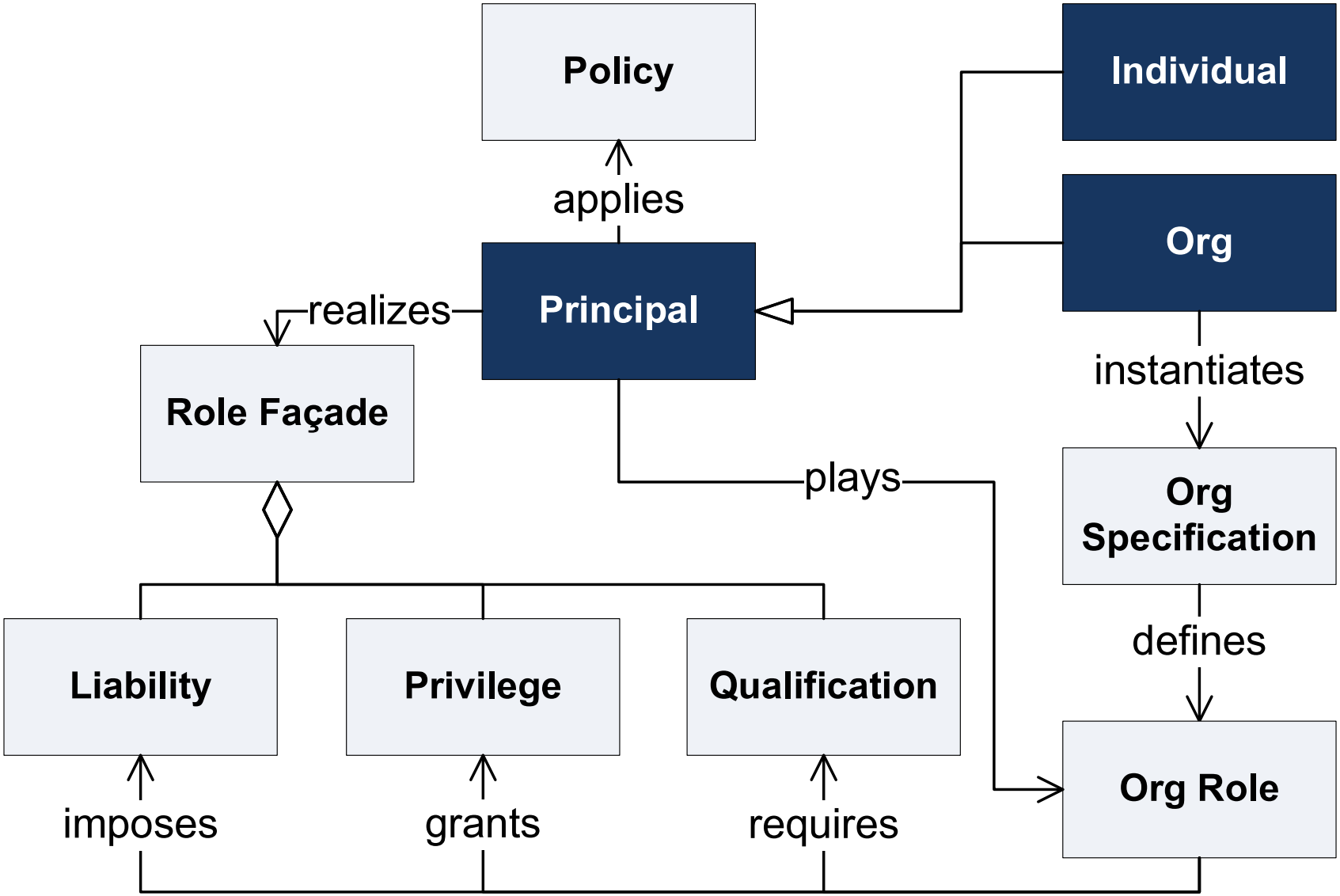
Principles of Governance: What Policies Give Us

Administration that is intelligent and intelligible

- Vividness of modeling
  - Grounded in applications; modeled entities are real
- Minimality of operational specifications
  - Leaving restrictions unstated except where essential to correctness
- Reification of representations
  - Explicit: hence, inspectable, sharable, and manipulable

Overview of Policy-Governed Secure Collaboration

Conceptual Model



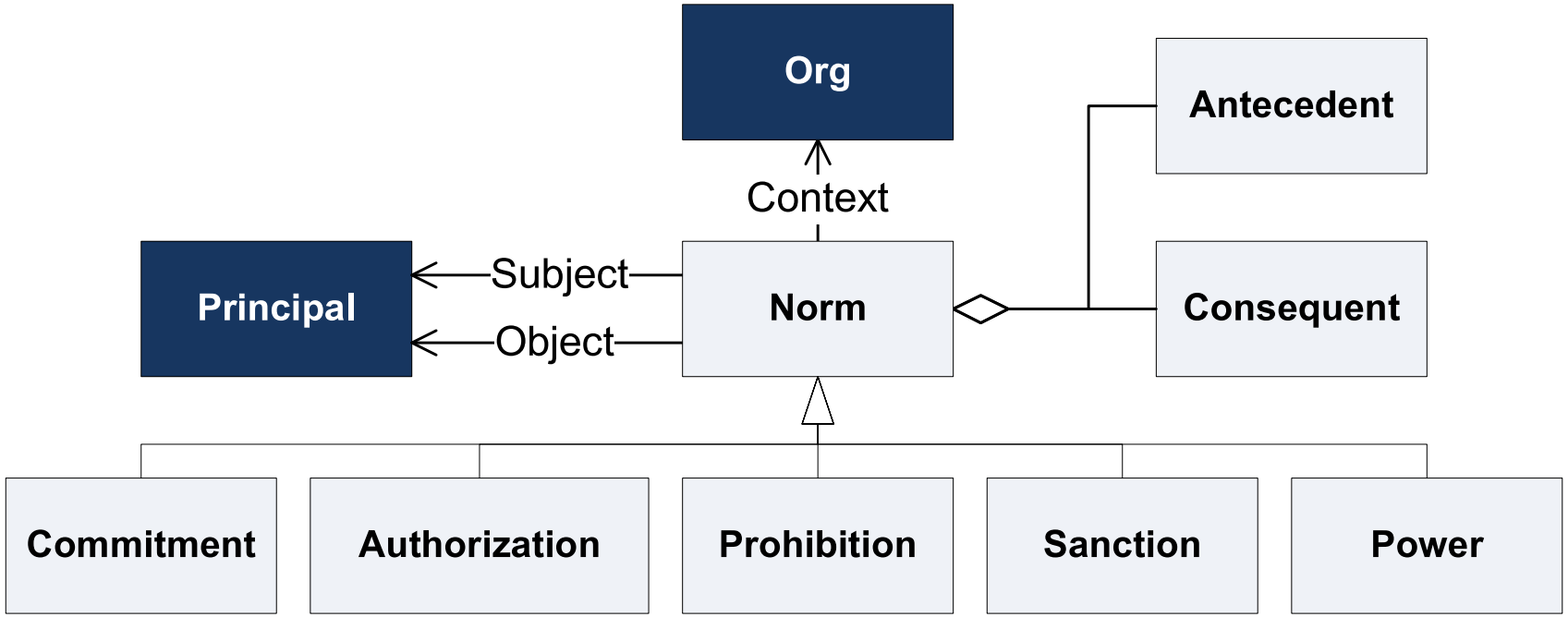
Achieving Governance: Principals and Orgs

Put collaboration in organizations center stage

- Principals are the stakeholders: people and organizations
  - Provide a locus for interaction
- Orgs are like *institutions:* have an identity and life time distinct from their members; also principals
  - Examples: NCSU, DoD, OOI, ...
  - Provide a locus for roles
  - Characterized via norms
  - Potentially enforce norms on members playing specific roles
    - An Org's main hold over its members is the threat of expulsion

Types of Norms

Unified logical form: Norm(subject, object, context, antecedent, consequent)

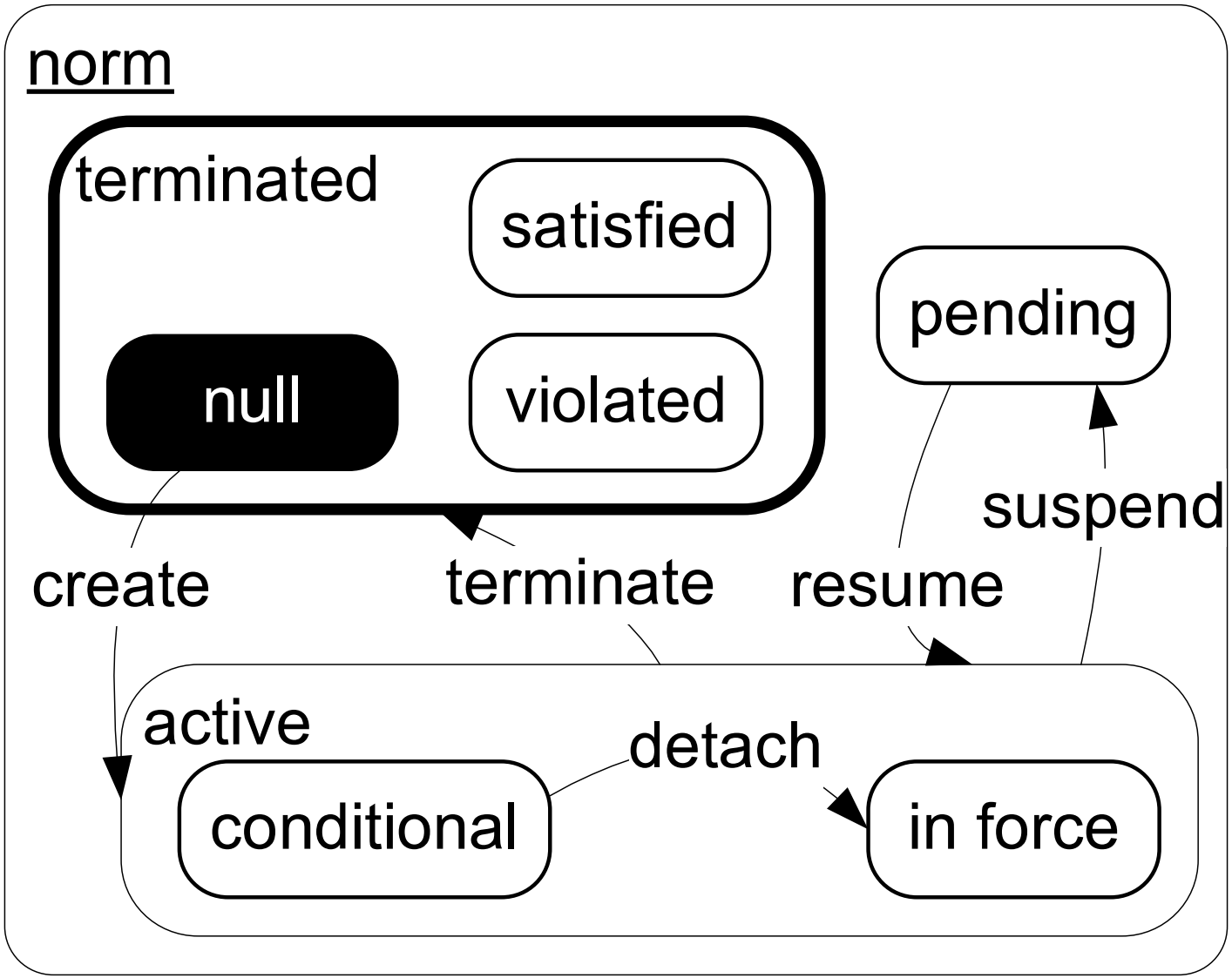


- Directed
- Declarative
- Composable
- Manipulable

Norms as Façades

Norm	**Subject's Façade**	**Object's Façade**
Commitment	Liability	Privilege
Authorization	Privilege	Liability
Power	Privilege	Liability
Prohibition	Liability	Privilege
Sanction	Liability	Privilege

Norm Life Cycle: 1



Norm Life Cycle: 2

Substate of a terminated norm

If terminated in		**Then**				
ant	**con**	**Com**	**Aut**	**Pro**	**San**	**Pow**
false	false	null	null	null	null	null
false	true	sat	vio	null	null	null
true	false	vio	null	sat	null	vio
true	true	sat	sat	vio	sat	sat

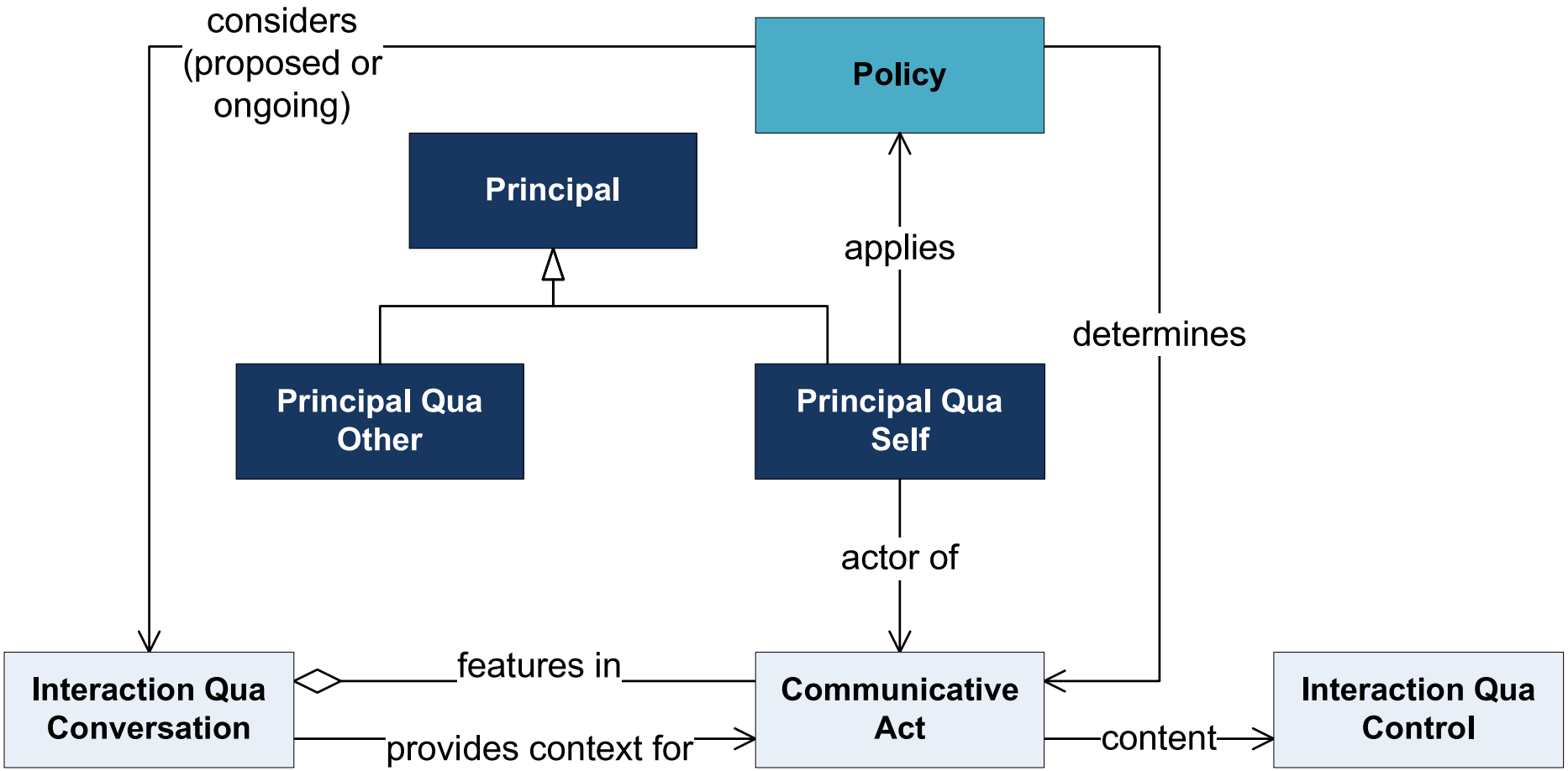
Unifying Norms and Policies for Governance

Promoting precision, verifiability, modularity, and reusability for secure collaboration

- Norms characterize interactions in terms of expectations and accountability
  - Provide the standards of correctness for governance
  - Packaged as role façades
  - Adopted by an agent to support its goals and concomitant policies
  - Help identify *policy points*: where policies apply
- An agent adopts policies that, given its role façades and goals,
  - Support discharging its liabilities
  - Potentially exploit its privileges
  - May not individually or collectively comply with norms
  - May thus violate some security expectations

Governance and Policies: Two Kinds of Interaction

Conversations with autonomous parties; control over resources



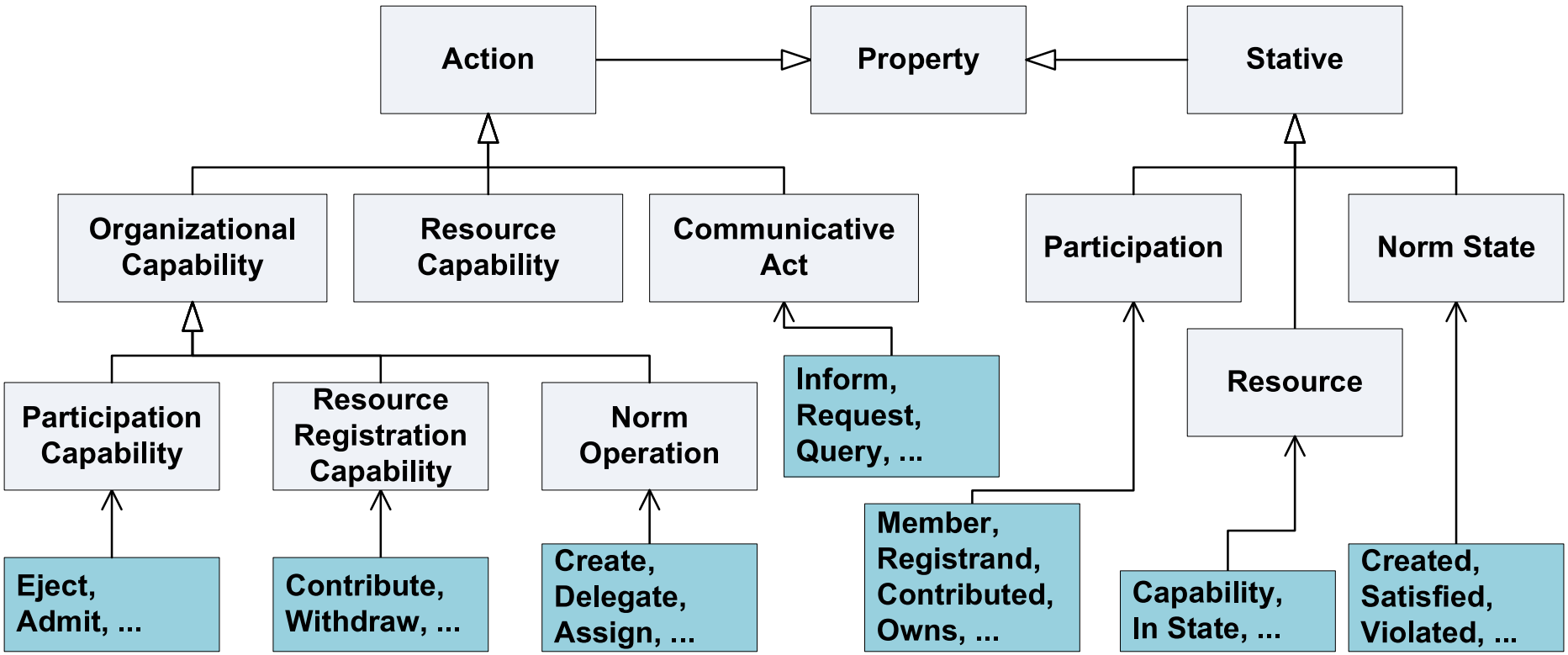
Governance and Policies: Information Model

Relevant information

- Attributes of the parties involved
  - Qualifications, affiliations
- Attributes of the capabilities involved
  - Interactions to be carried out upon resources
  - Collated as interaction types and resource types
- Attributes of the relationships among the parties involved
  - Participations in different Orgs
  - Arrangements among Orgs (captured as participations)
  - Ongoing interactions

Vocabulary for Governance and Policies

Norms and Orgs



An Information Model and Commitment Specification

```
TakeCharge(tcID, nuID, phID, patID, tcThreshold) key tcID
CardiacEvent(ceID, nuID, phID, patID, ceMagnitude) key ceID
CPR(cprID, nuID, phID, patID, cprDuration) key cprID

commitment CardioCare nuID to phID
create TakeCharge
detach CardiacEvent [, TakeCharge + 180]
  where ceMagnitude >= tcThreshold
discharge CPR [, CardiacEvent + 5]
```

A Cardio Care commitment from a nurse to a physician is

- *created* upon Take Charge;
- *detached* if a CardiacEvent for this patient happens above the specified threshold within 180 minutes
  - Else the commitment *expires*
- *discharged* if CPR on this patient happens within five minutes of the Cardiac Event (else *violated*)

Generate Log Schema

```
CREATE TABLE TakeCharge (
   tcID  VARCHAR(10), nuID  VARCHAR(10), phID  VARCHAR(10),
       patID  VARCHAR(10), tcThreshold  VARCHAR(10),
   stamp  DATETIME,
   PRIMARY KEY(tcID)
);

CREATE TABLE CardiacEvent (
   ceID  VARCHAR(10), nuID  VARCHAR(10), phID  VARCHAR(10),
       patID  VARCHAR(10), ceMagnitude  VARCHAR(10),
   stamp  DATETIME,
   PRIMARY KEY(ceID)
);

CREATE TABLE CPR (
   cprID  VARCHAR(10), nuID  VARCHAR(10), phID  VARCHAR(10),
       patID  VARCHAR(10), cprDuration  VARCHAR(10),
   stamp  DATETIME,
   PRIMARY KEY(cprID)
);
```

Generate Canonical Queries for Accountability Checking

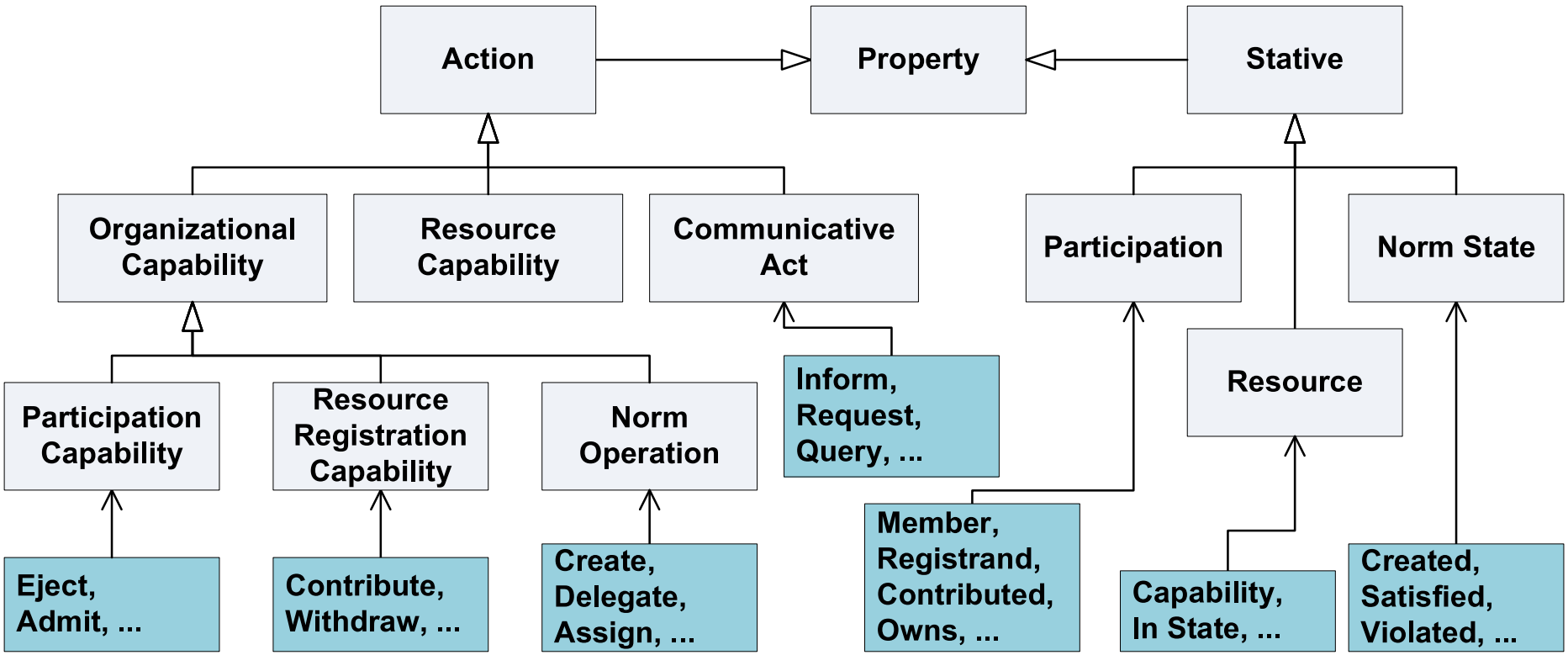
In relational algebra (Jun Yang's notation)

Query for which Cardio Care commitments are detached

```
((\select_{(stamp >= stamp38)} (
 (TakeCharge) \join
 (\rename_{ceID,nuID,phID,patID,ceMagnitude,stamp38}
  (\select_{ceMagnitude = tcThreshold} (CardiacEvent)))))
      \union
  (\select_{(stamp >= stamp37)}
    ((\select_{ceMagnitude = tcThreshold}
      (CardiacEvent)) \join
      (\rename_{tcID,nuID,phID,patID,tcThreshold,stamp37}
       (TakeCharge)))));
```

Vocabulary for Governance and Policies

Norms and Orgs



Policy Types

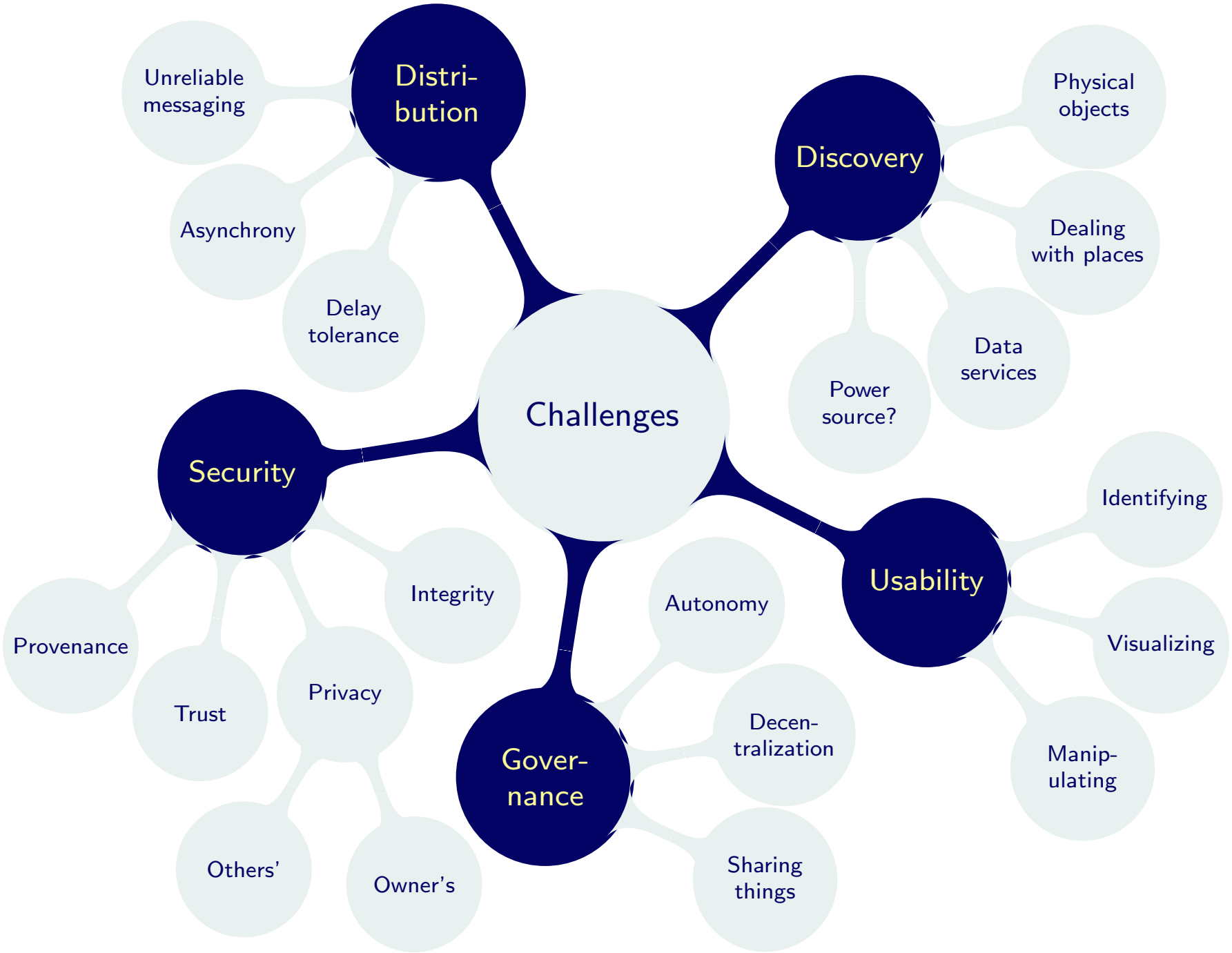
The policy interactions need to go beyond traditional access control

- Each policy can be understood in terms of its cause and its effect
- Cause
  - *Reactive:* triggered by a request from another stakeholder
  - *Proactive:* triggered by local observations
- Effect
  - *Authorization* of action to be taken on behalf of requester
  - *Enablement* of action, which would otherwise not be taken
  - *Expectation* of action, which would now be performed

Challenges and Partial Recent Progress

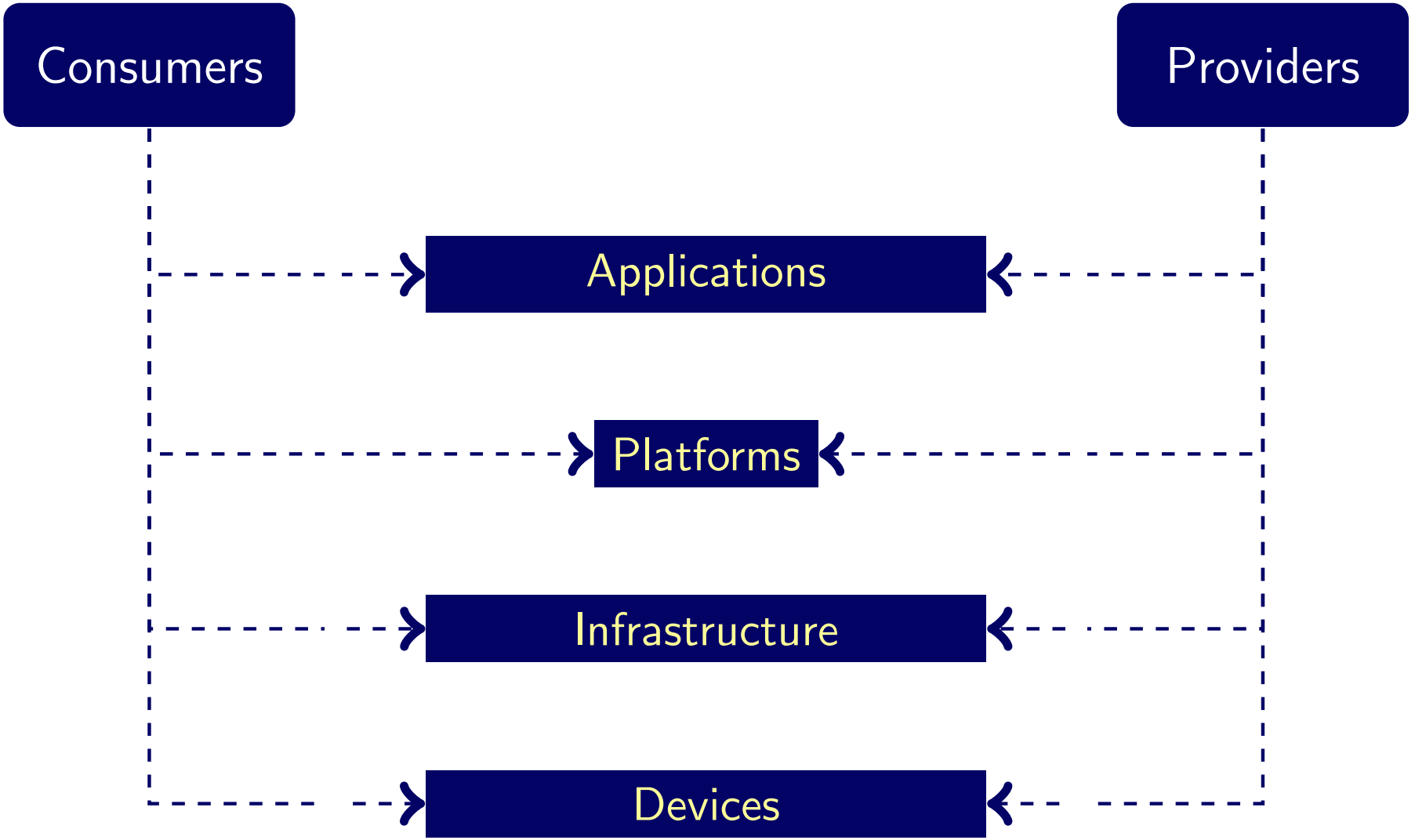
- Storing and retrieving events to determine the state of a norm
  - Mapping commitments to relational algebra [AAAI 2015]
- Maintaining alignment of views despite decentralization
  - Communications to guarantee (eventual) alignment [AAMAS 2015]
  - TBD: maximizing partial or "quick" alignment
- Designing protocols and Org contexts for monitorability
  - Failure of compositionality of monitorability [IJCAI 2015]
  - Automatically close a context to ensure monitorability
- Designing protocols and Org for robustness and resilience
  - Typology of sanctions and sanctioning processes [Draft]
  - TBD: Formalization of normative robustness and resilience
  - TBD: Reasoning about sanctions for design of Orgs
- Design processes conducive to autonomy
  - Abstract formal model of a sociotechnical design process [RE 2014]
  - TBD: Methodologies

Outline


Challenges
Distri-bution
Unreliable messaging
Asynchrony
Delay tolerance
Discovery
Physical objects
Dealing with places
Data services
Power source?
Security
Integrity
Provenance
Trust
Privacy
Others'
Owner's
Gover-nance
Autonomy
Decen-tralization
Sharing things
Usability
Identifying
Visualizing
Manip-ulating

IoT Layers: Desire a Few Platforms for Many Applications

Benefit from the upper layers; excitement from the lower layer



Summary and Directions

Exercise: Collective concept map

- What theme do you remember most from today?
- What additional information systems theme should we consider?
- What IoT research question would be worth pursuing?