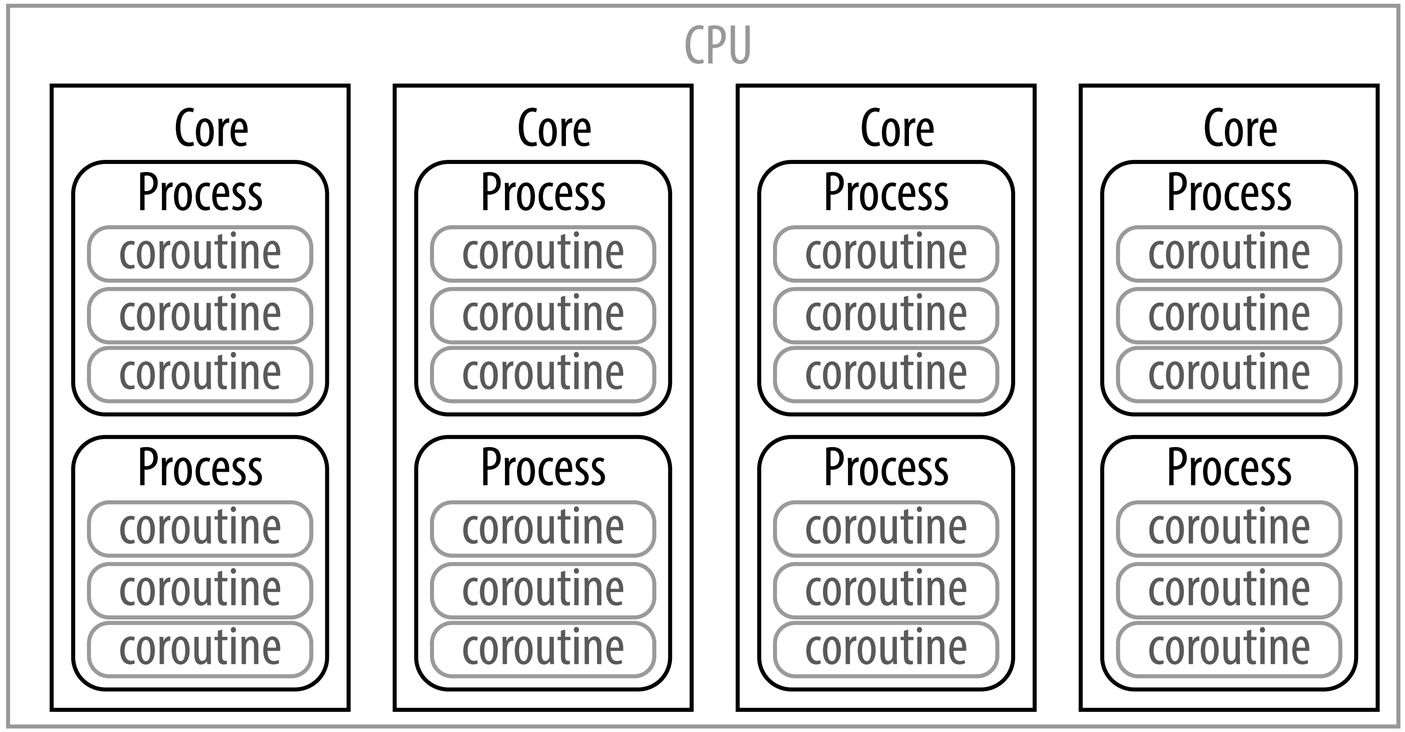
HIO
Hierarchical Asynchronous Coroutines & I/O

Samuel M. Smith Ph.D.
sam@samuelsmith.org

What is Asynchronous Processing?

Cooperative logical concurrency within a single process/thread plus nonblocking IO.



Advantages:

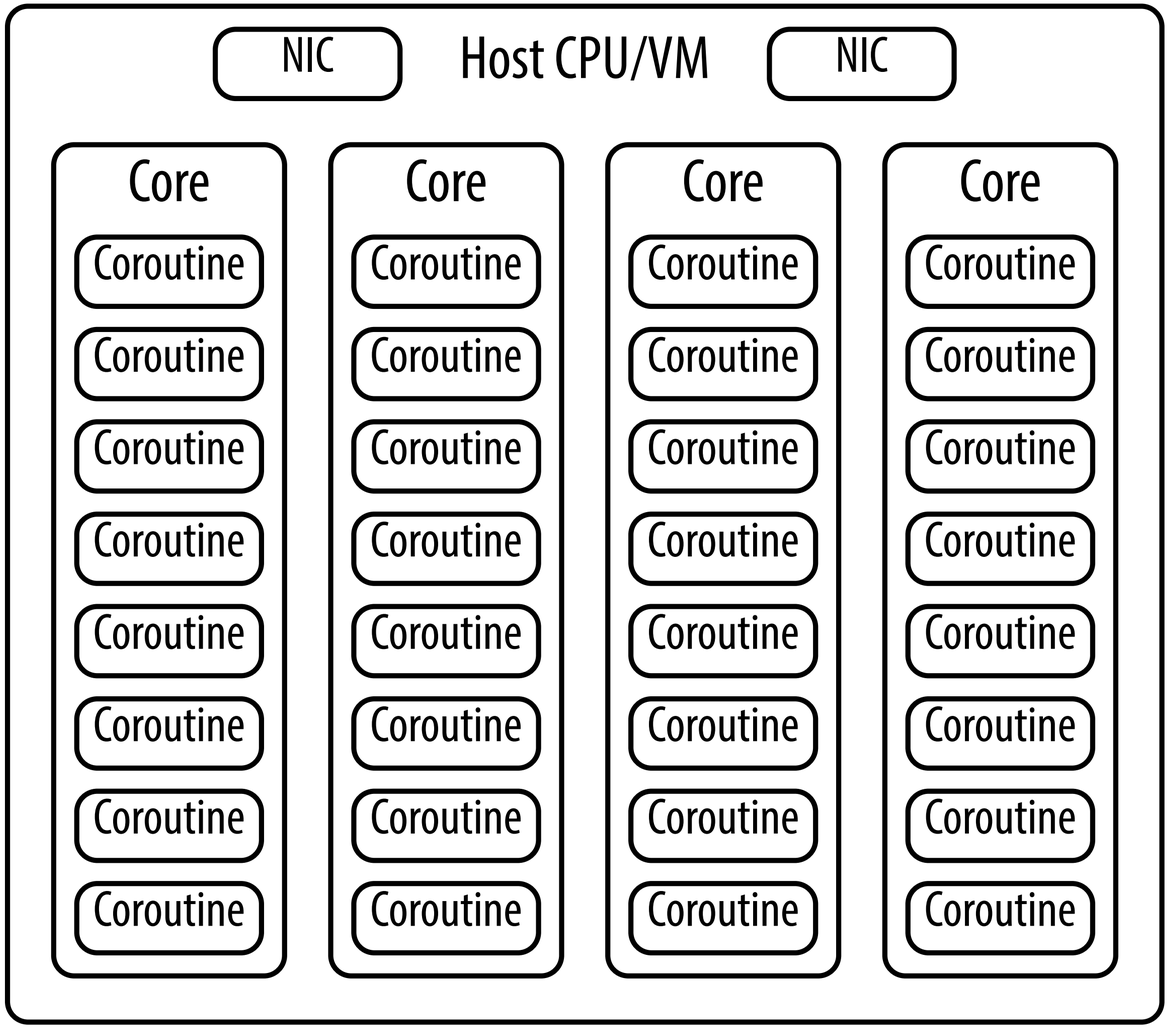
Performant, 1 process per core/hyperthread, minimizes context switch overhead usually >15%

Simple, no resources contentions, mutexes, locks, semaphores, everything is atomic wrt interruptions

Scalable, granularity matches logical concurrency not arbitrary host/process boundaries

Asynchronous Optimized Computation

Coroutines are the fundamental units of logically concurrent computation



Minimizes context switch overhead

Asynchronous Processing Approaches

Greenlets, Micro-threads, gevent, stackless python

Emulates conventional multi-thread/process model, join calculus, sleepy, continuous logic flow,

Event Loop Callbacks, Twisted, Node.js

Medusa/Reactor pattern, callback hell, discontinuous logic flow

Deferred, Python 3.5 asyncio futures, promises

Thread pool like scheduler, discontinuous logic flow

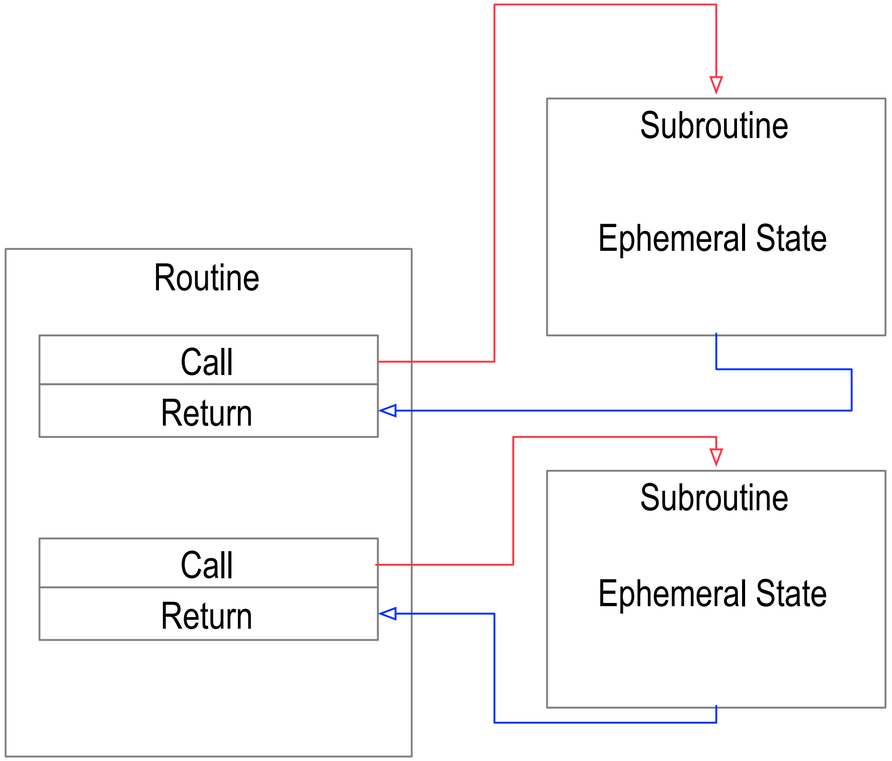
Yielded Delegated, generators/coroutines, Python 3.5 asyncio coroutines

Yielding scheduler, delegating coroutines, yieldy, continuous logic flow

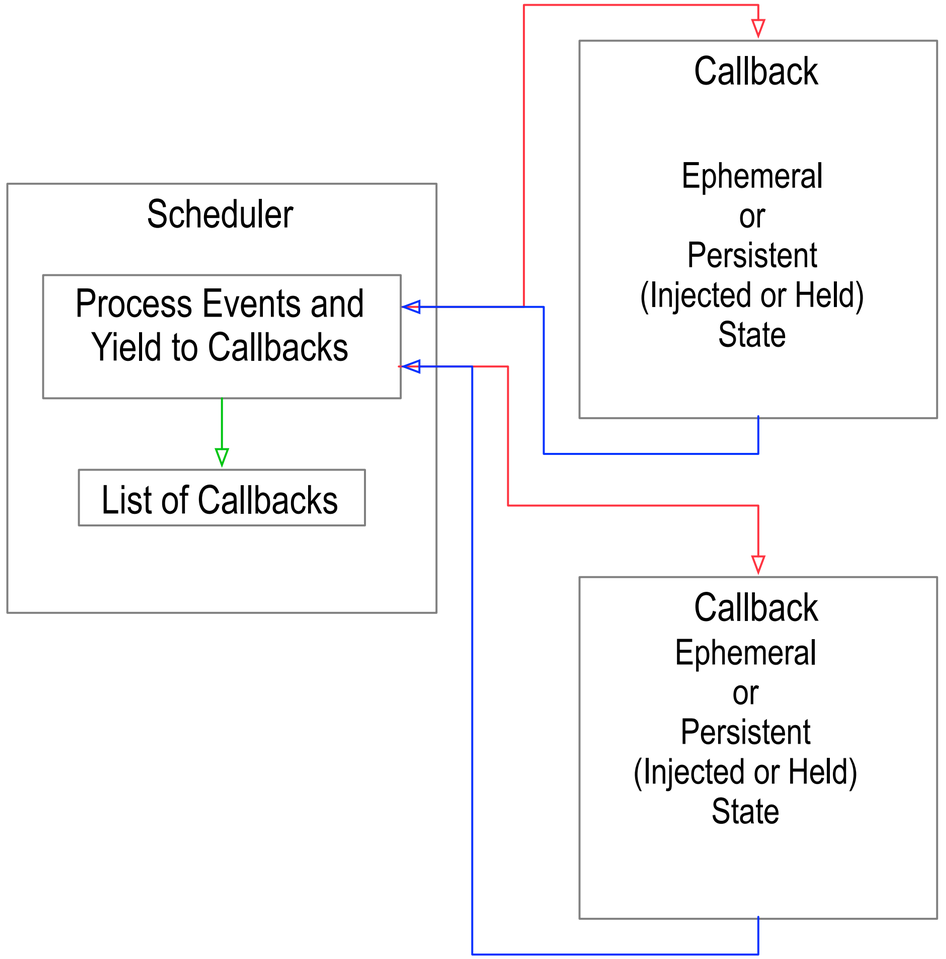
Nested hybrid coroutines, Ioflo

Nested contextual yielding/delegating schedulers, continuous logic flow

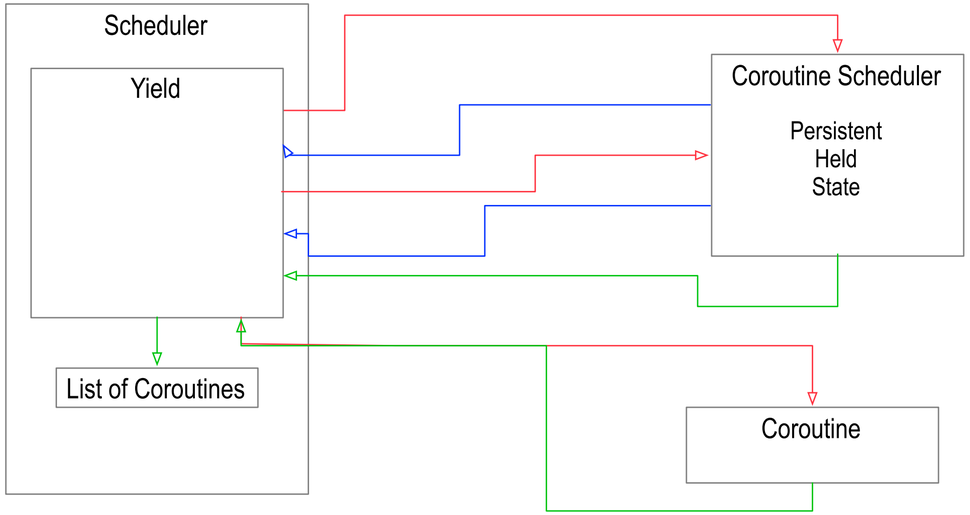
Synchronized



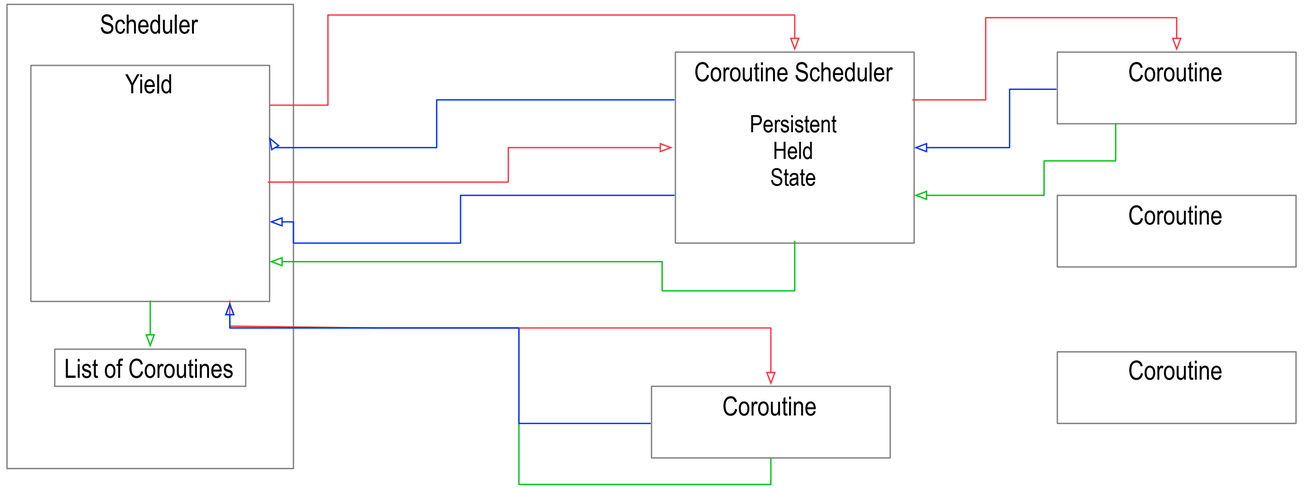
Async Callback



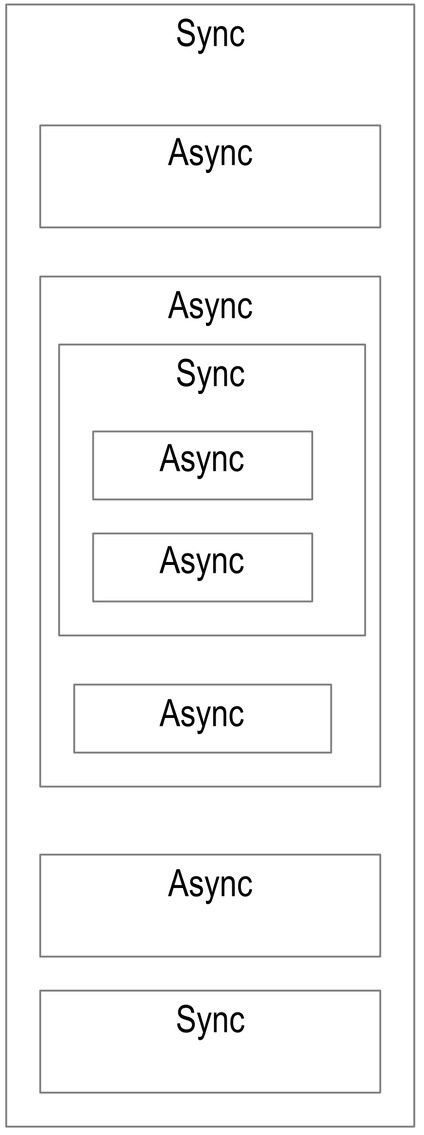
Coroutine



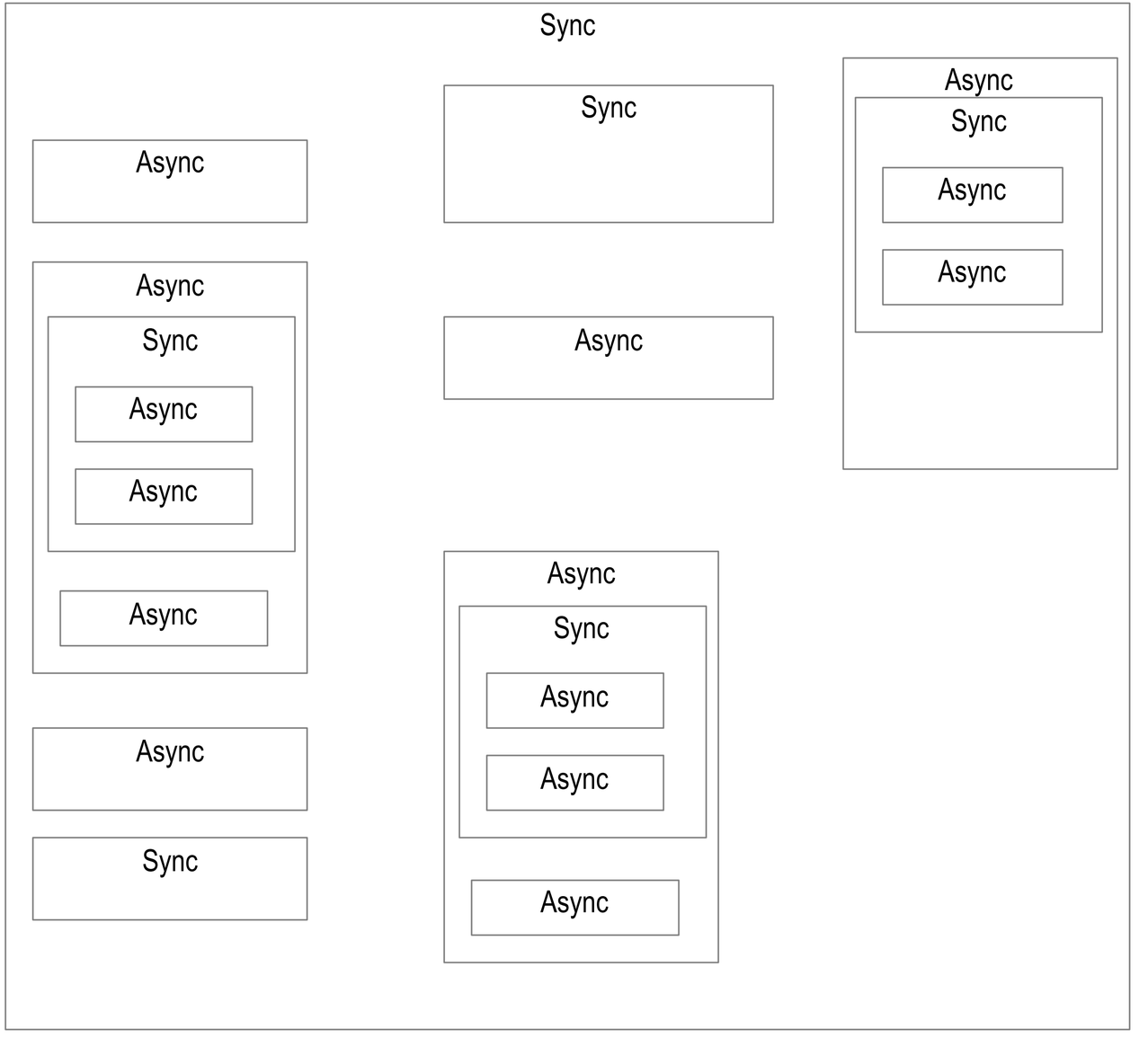
Hierarchical Coroutine



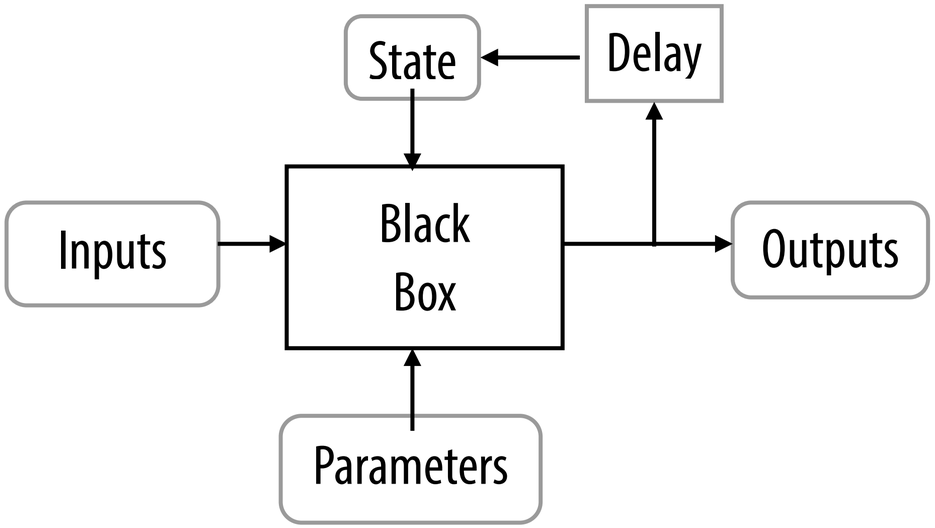
Hierarchical Heterarchical

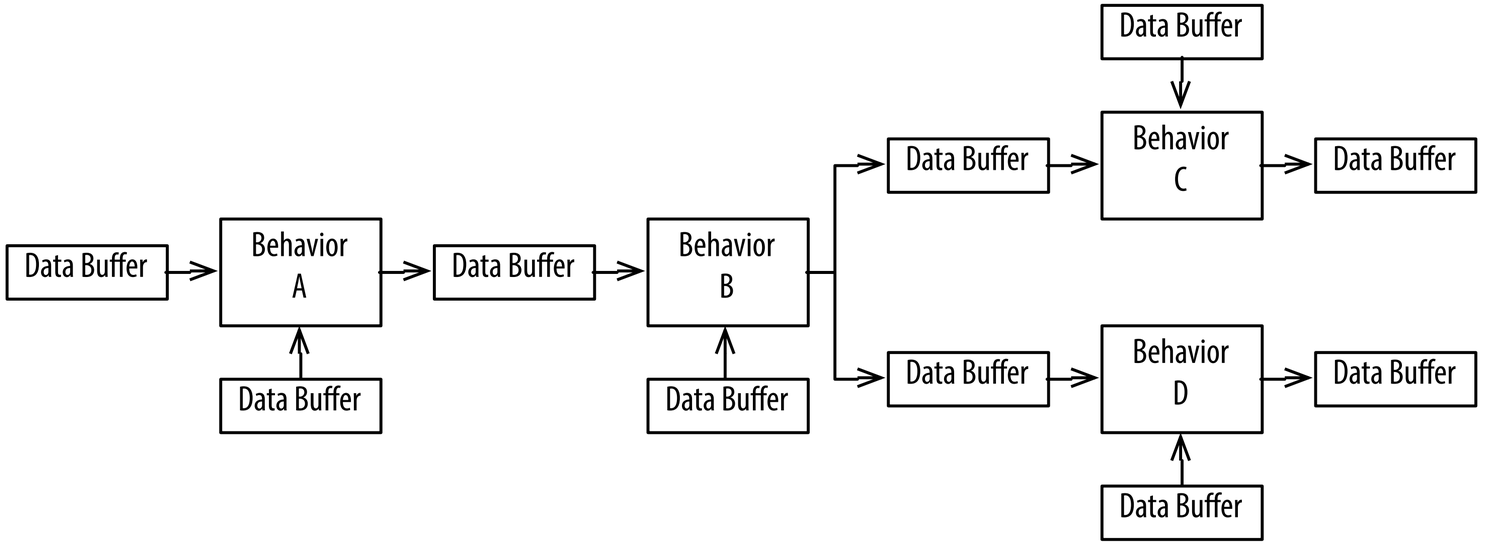


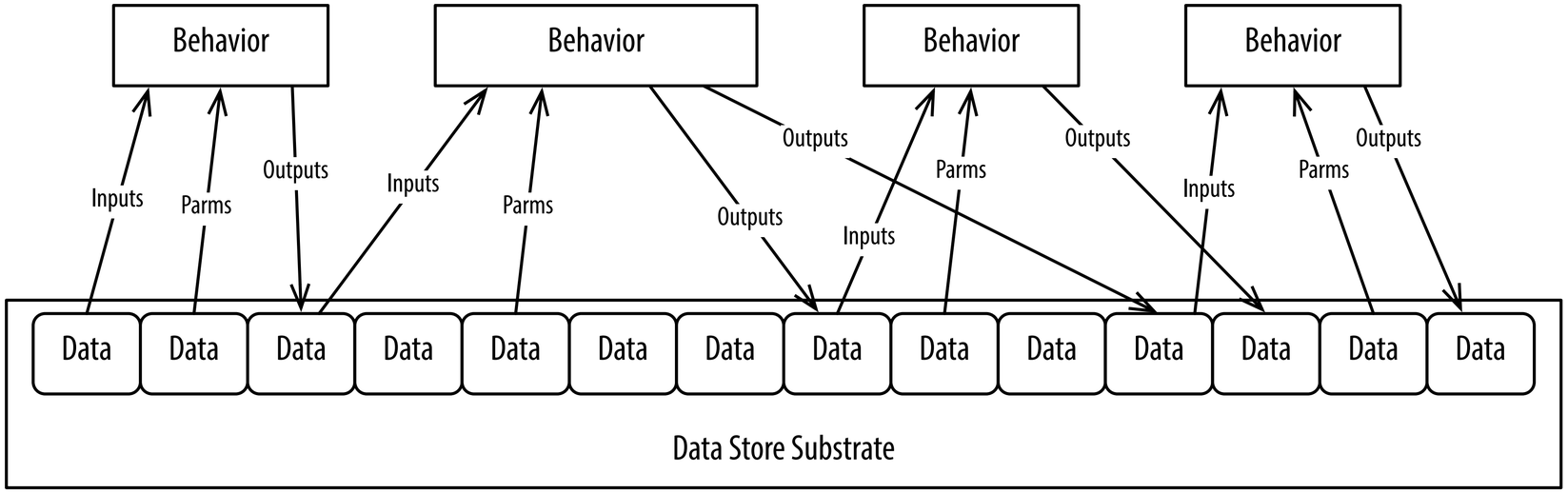
Complex Hierarchical Heterarchical



FLOW BASED PROGRAMMING

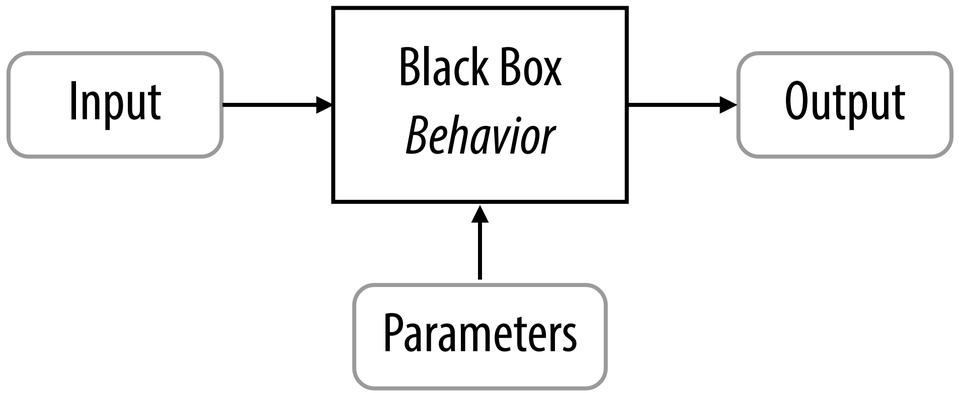






What is Flow-Based Programming (FBP)

A behavior transforms its input(s) into its output(s)

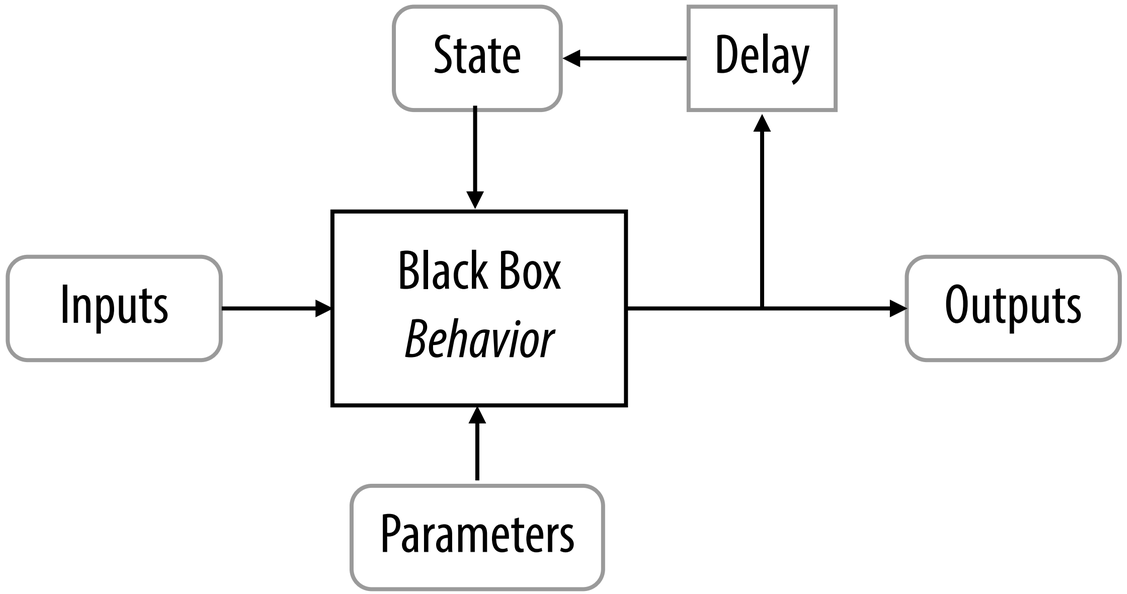


Inputs may also be parameters that modify the transformation

A behavior transforms its inputs governed by its parameters into its outputs

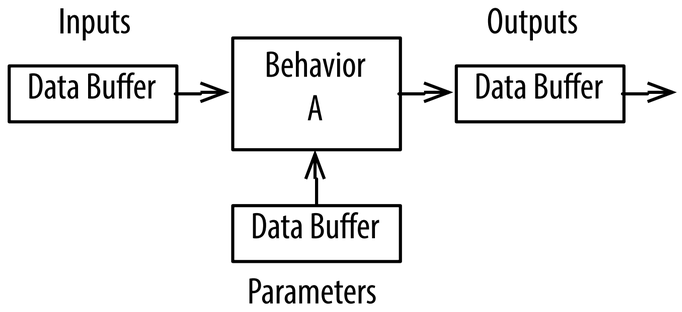
What is Flow-Based Programming (FBP)

Outputs may be fed back as state to modify the transformation

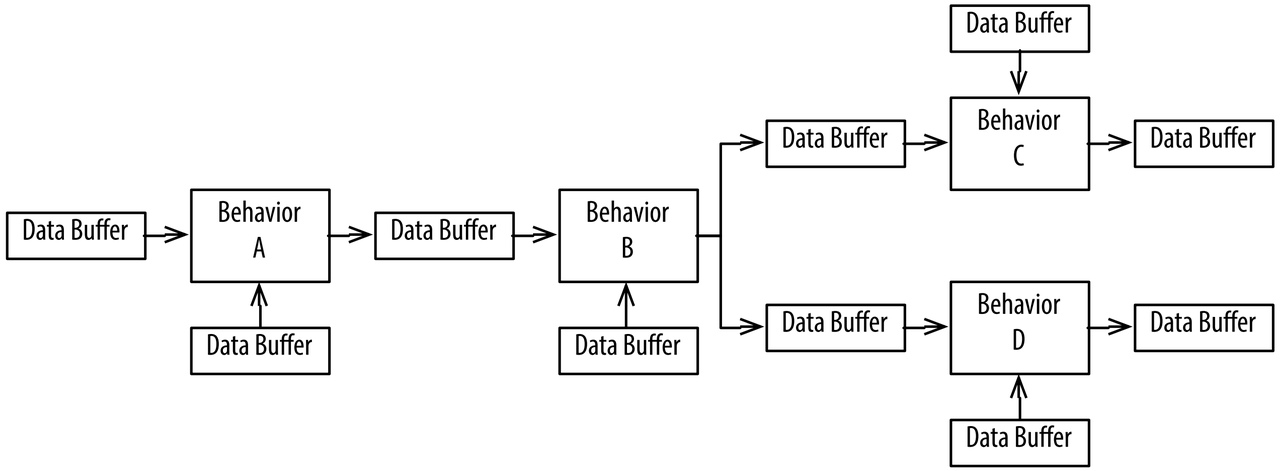


A behavior transforms its inputs and past outputs as governed by its parameters into its outputs

What is Flow-Based Programming (FBP)

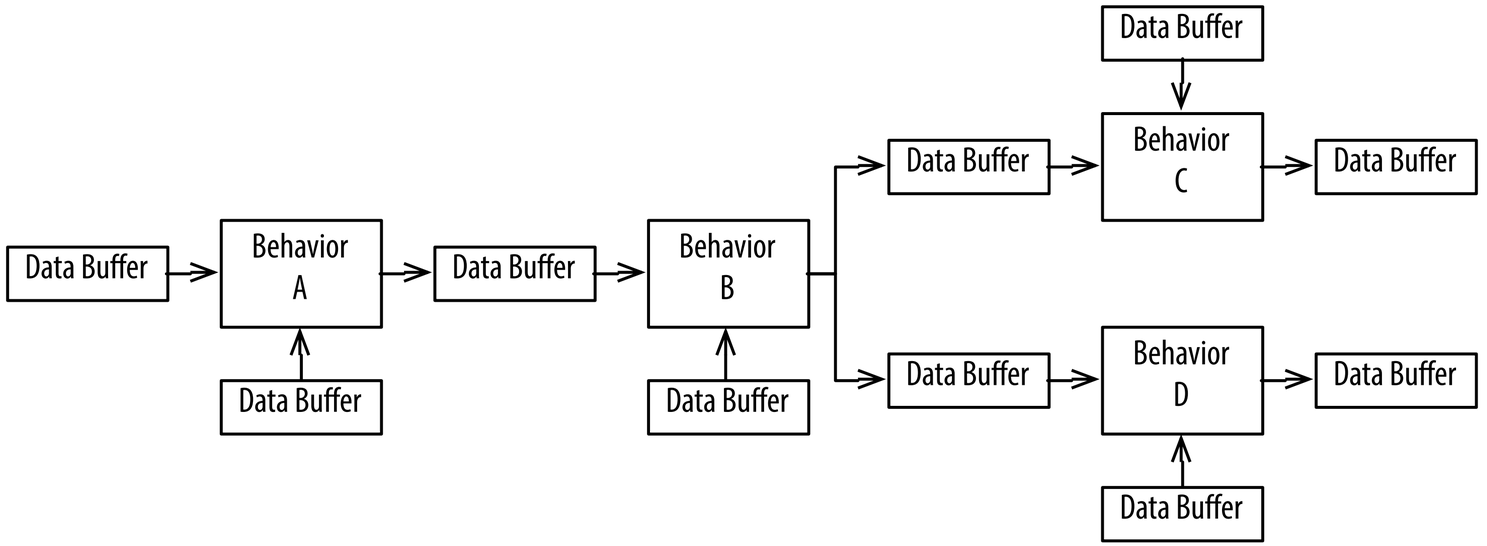


Behaviors can be reconnected endlessly in different networks without any internal changes



FBP applications (programs) are networks of asynchronous behaviors
which exchange data across externally defined connections

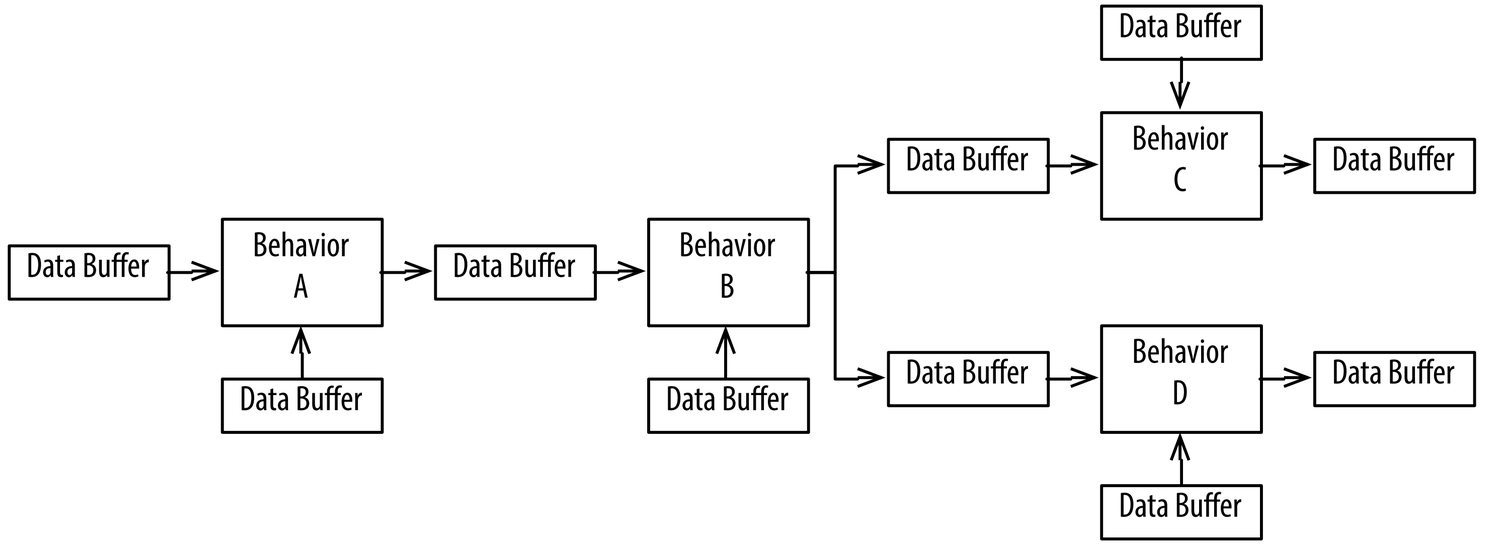
What is Flow-Based Programming (FBP)



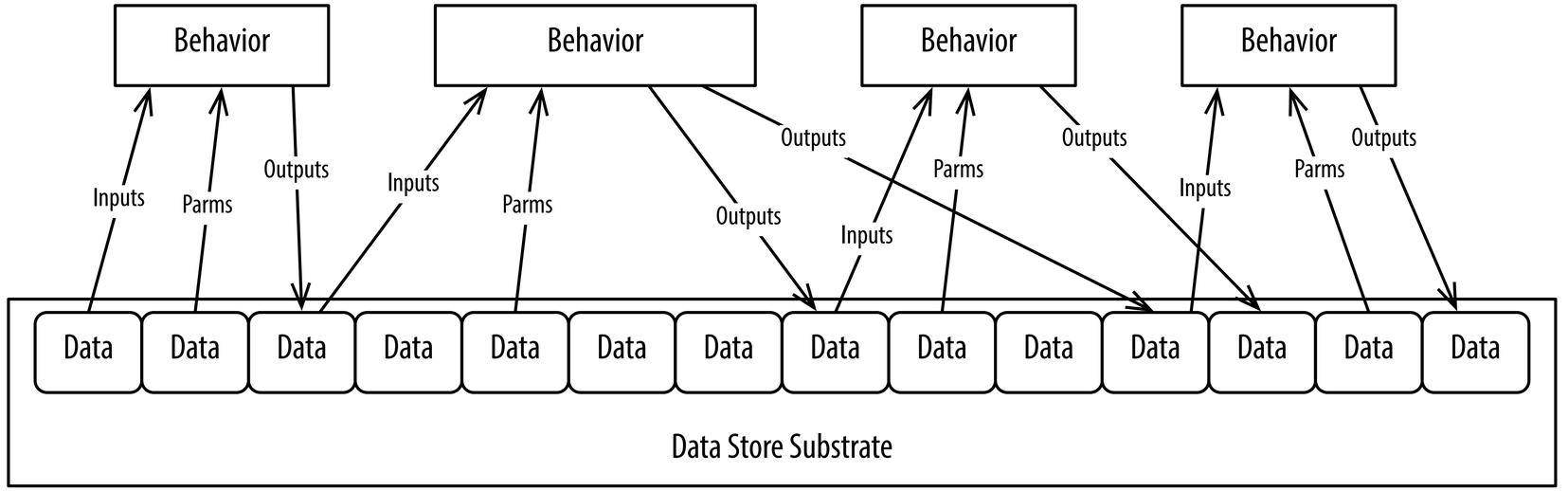
- Programming is the declarative composition of networks of behaviors
- FBP = Data Flow Oriented + Component Oriented
- FBP = FP-ish + OOP-ish

FBP Variant with DataStore Substrate

From this

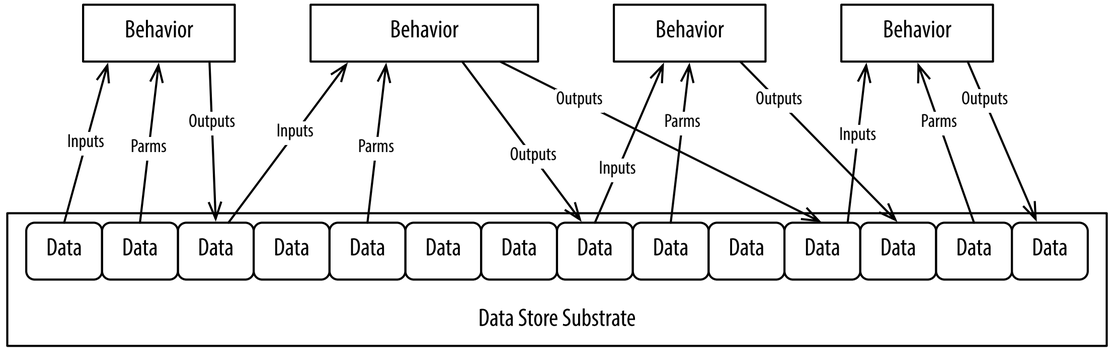


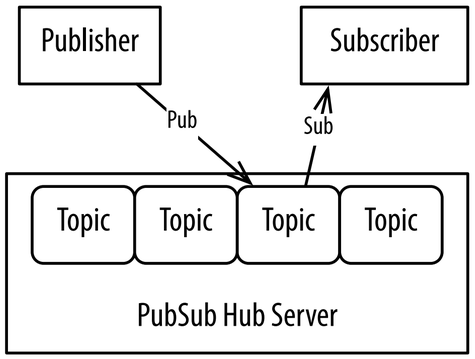
To this

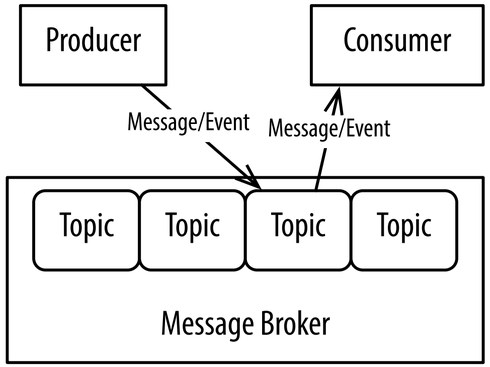


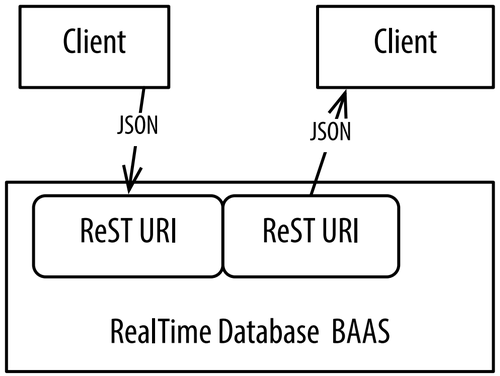
- Adds configurability, observability, traceability, replayability

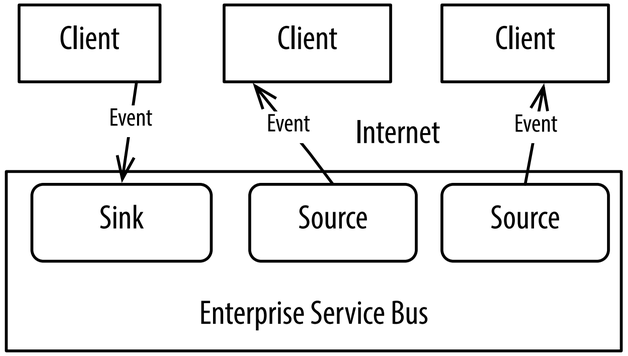
Meta Architecture Patterns

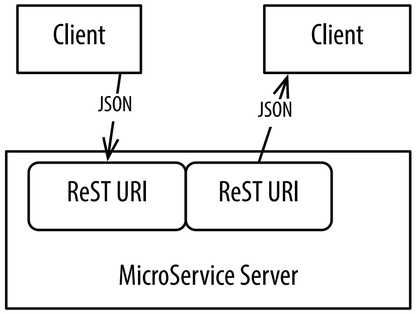


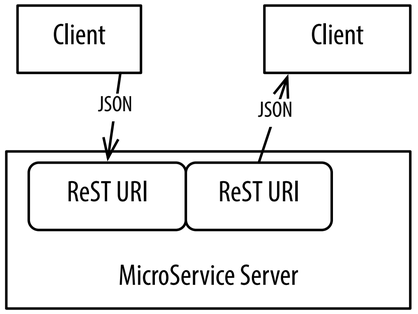


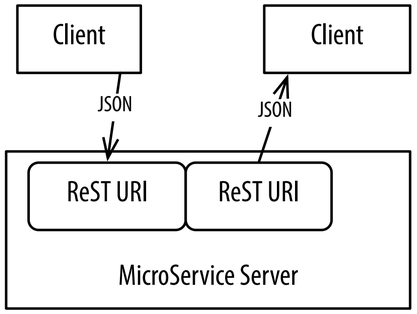


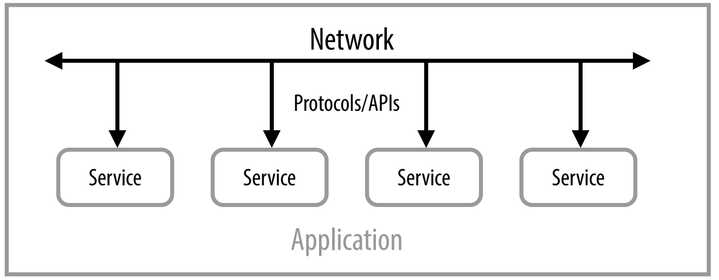












Dependency Inversion Principle

Dependency Inversion Principle, Martin 1996

Bad Software Design: Software that fulfills its requirements but exhibits any or all of:

Rigidity: Hard to change because each change affects other parts of the system.

Fragility: Making a change causes other parts of the system to break.

Immobility: Hard to disentangle in order to reuse in another application.

DIP:

HIGH LEVEL MODULES SHOULD NOT DEPEND UPON LOW LEVEL MODULES.

BOTH SHOULD DEPEND UPON ABSTRACTIONS.

ABSTRACTIONS SHOULD NOT DEPEND UPON DETAILS.

DETAILS SHOULD DEPEND UPON ABSTRACTIONS.

Its all about dependency management, duh !!!

Dependency Inversion Principle Transcendence

Bad Software Design: Software that fulfills its requirements but exhibits any or all of:

Rigidity: Hard to change because each change affects other parts of the system.

Fragility: Making a change causes other parts of the system to break.

Immobility: Hard to disentangle in order to reuse in another application.

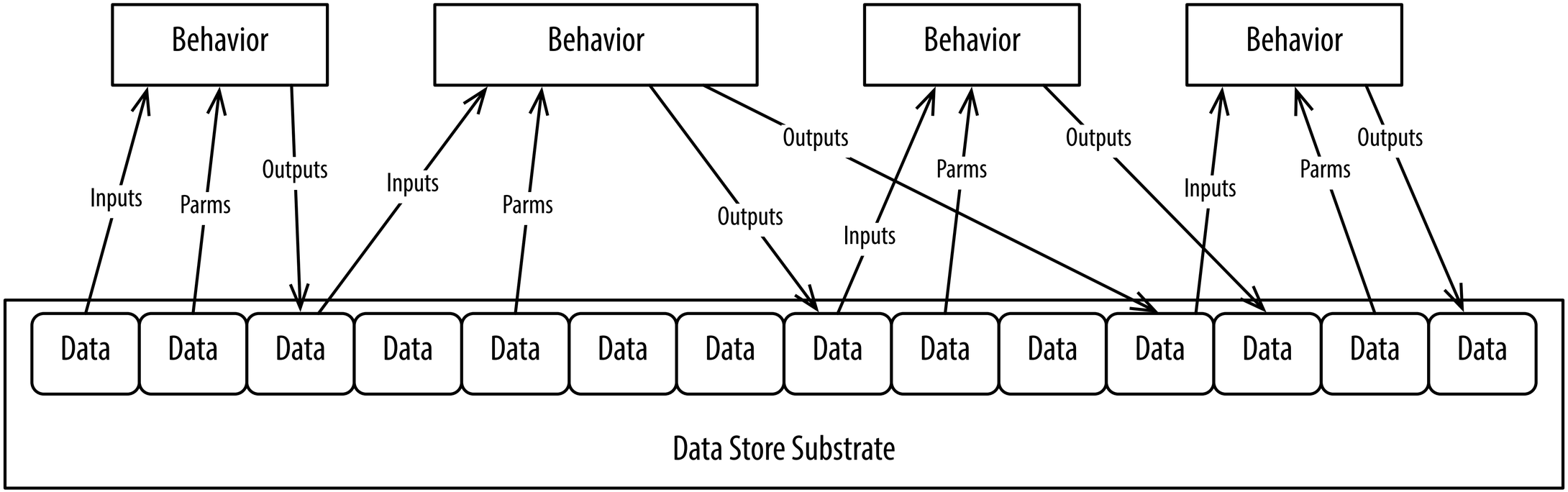
Flow-Based Programming transcends the DIP thusly:

THERE ARE NO MODULES, JUST COMPONENTS

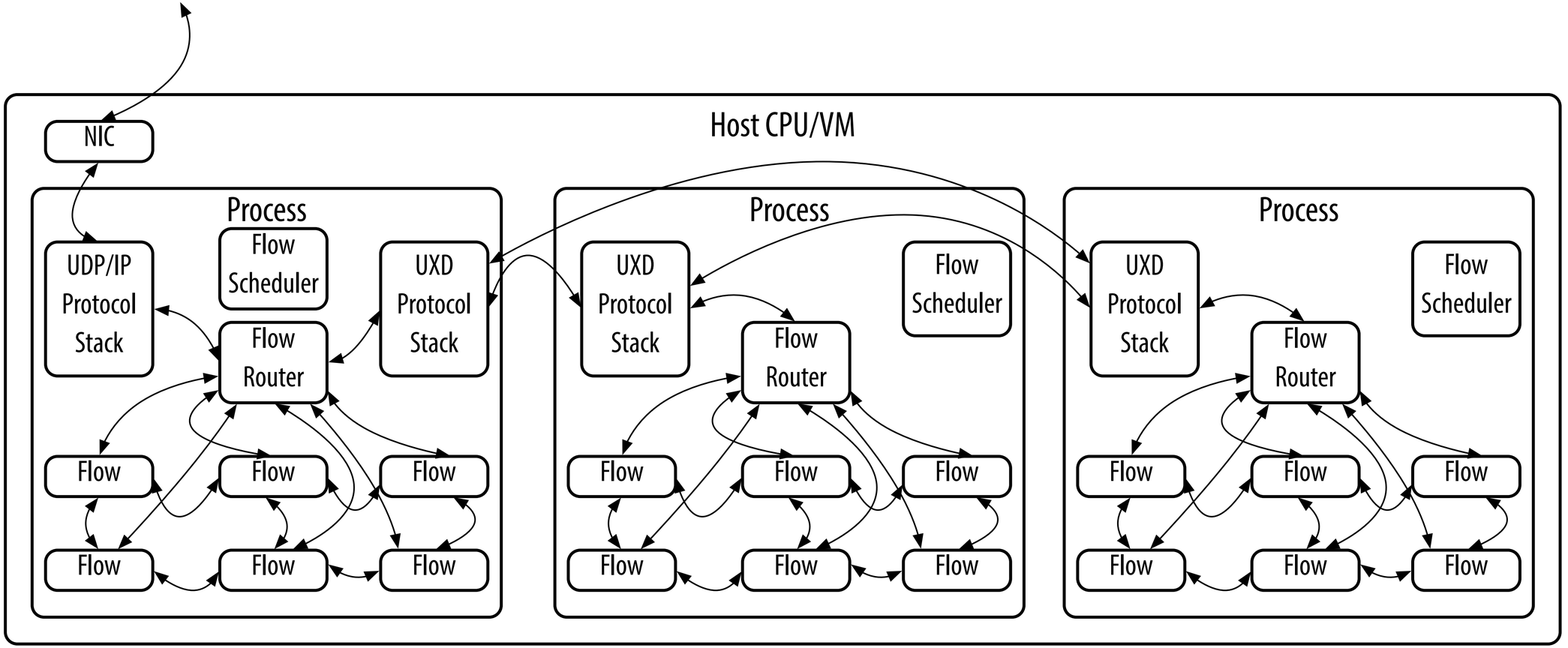
THERE ARE NO ABSTRACTIONS OR DETAILS JUST DATA

COMPONENTS DEPEND ON DATA, NOT OTHER COMPONENTS

One Dependency: DATA



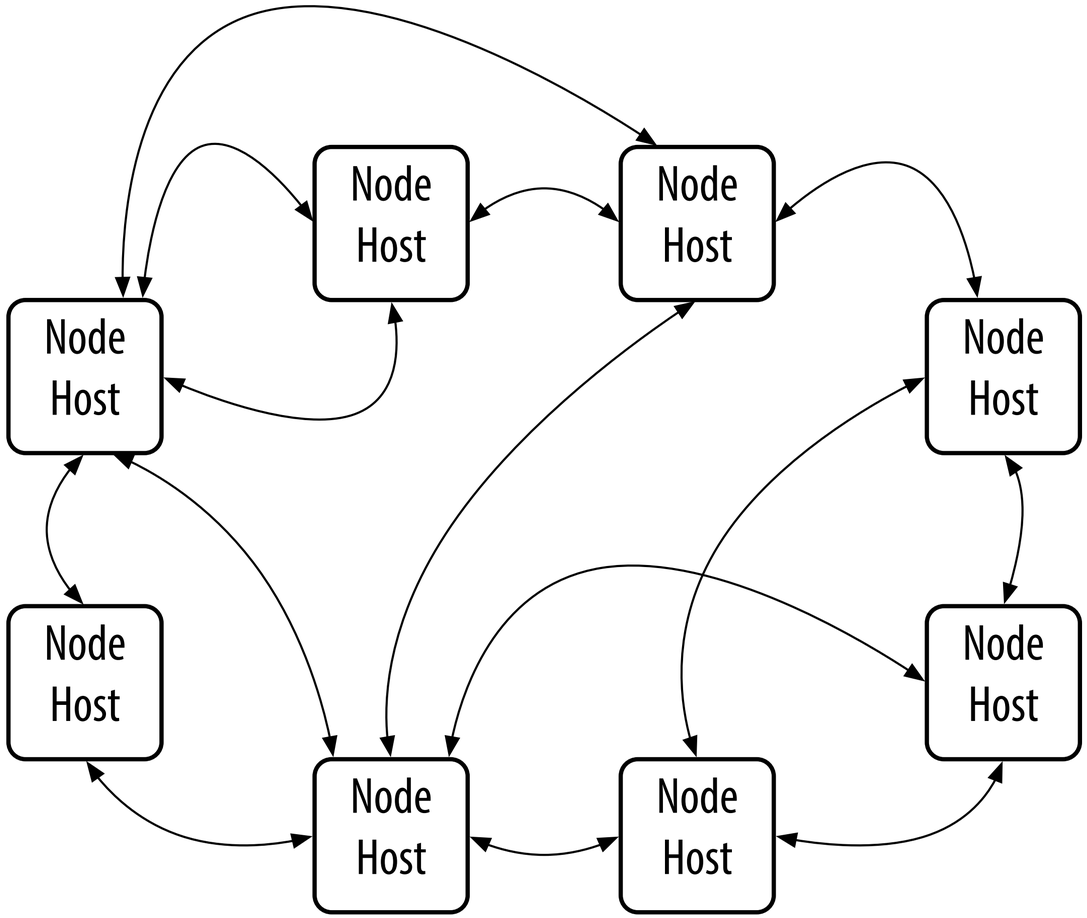
Data Flow Routing



Enables coroutine-to-coroutine communication
both intra-process, inter-process and inter-host

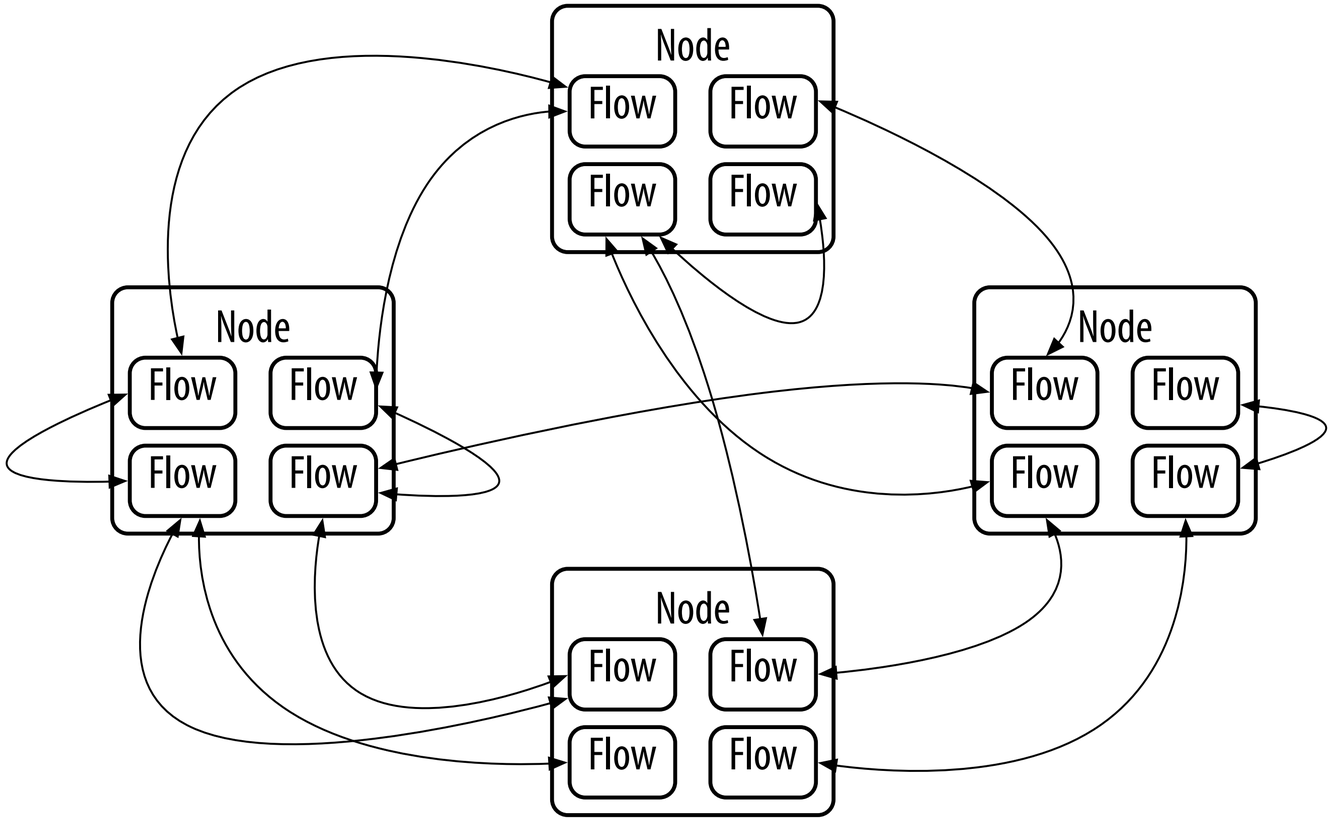
Many-to-Many-to-Many

Many-to-many



Performance and Reliability

Many-to-many-to-many



Logically concurrent flows of execution

Concurrent data flows

Service flows

Encapsulated Routing Data

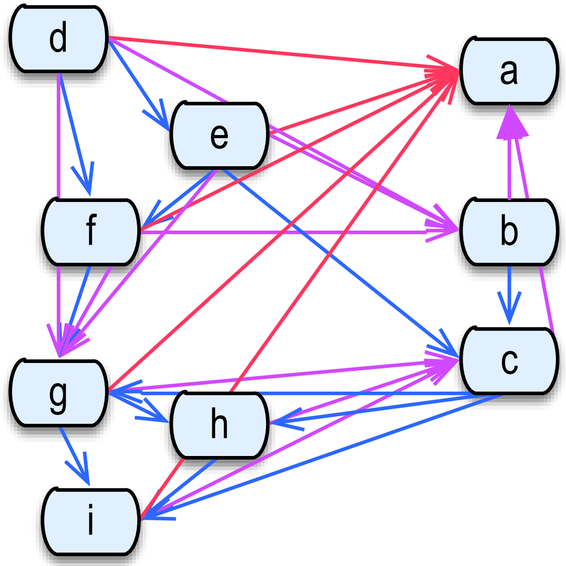
Tuple: (datashare name, UXD name, UDP name, channel name)

JSON:

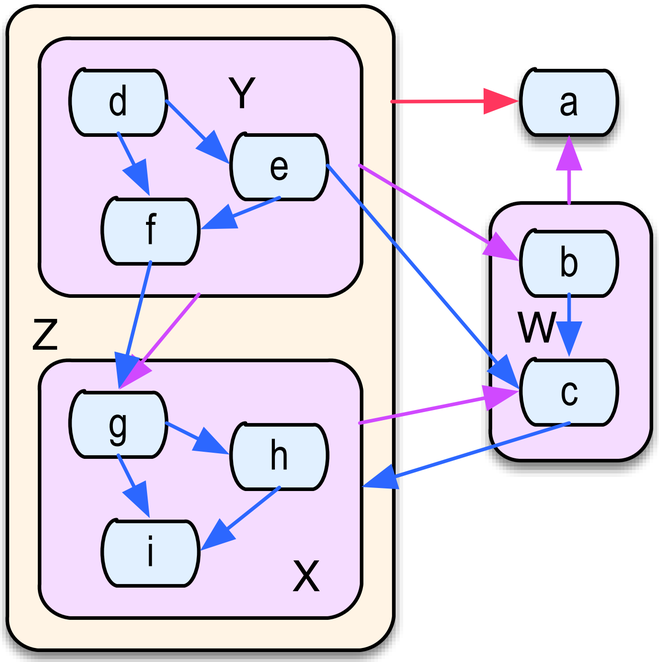
```
{
    "route":
    {
        "src": ["peter.piper.picked", "pepper", "peck", "garden"],
        "dst": ["suzy.sand.sell", "shell", "shore", "sea"]
    }
    "tag": "commerce.hot",
    "stamp": "20170125T13:45:62.340123UTC",
    "data":
    {
       "quantity": 5,
       "price": 1.50
    }
}
```

Hierarchical State

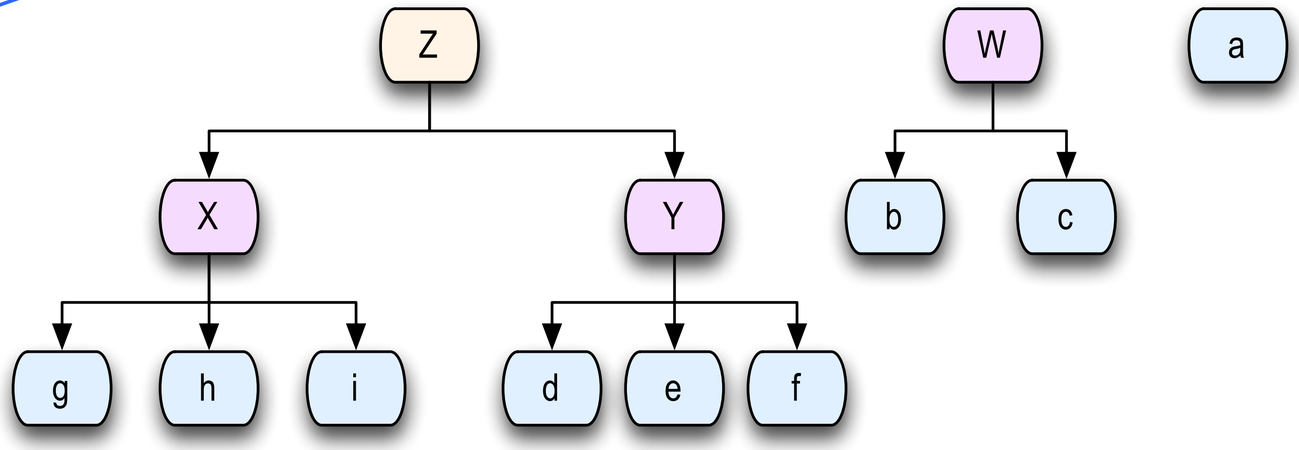
Flat State Machine
Relatively High Apparent Complexity



Hierarchical State Machine
Relatively Low Apparent Complexity

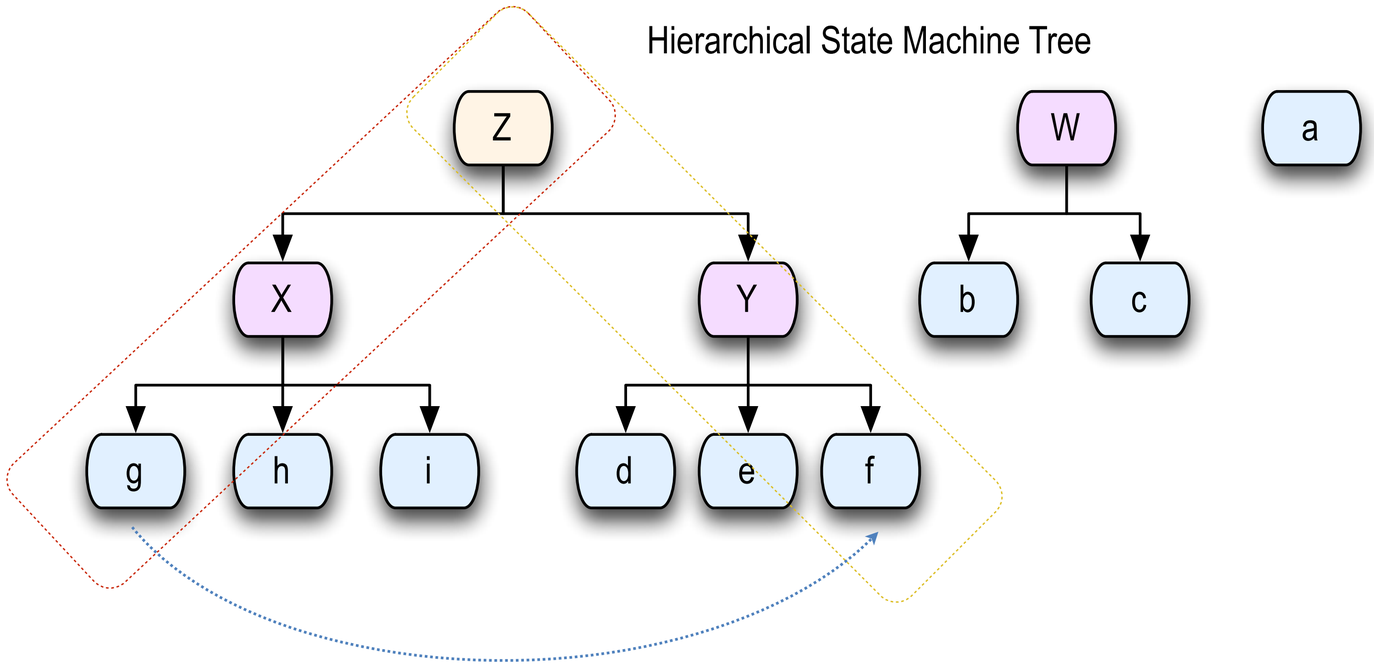


Hierarchical State Machine Tree



Transition = Change in Framing Context

```
go foobar if elapsed >= goal
```

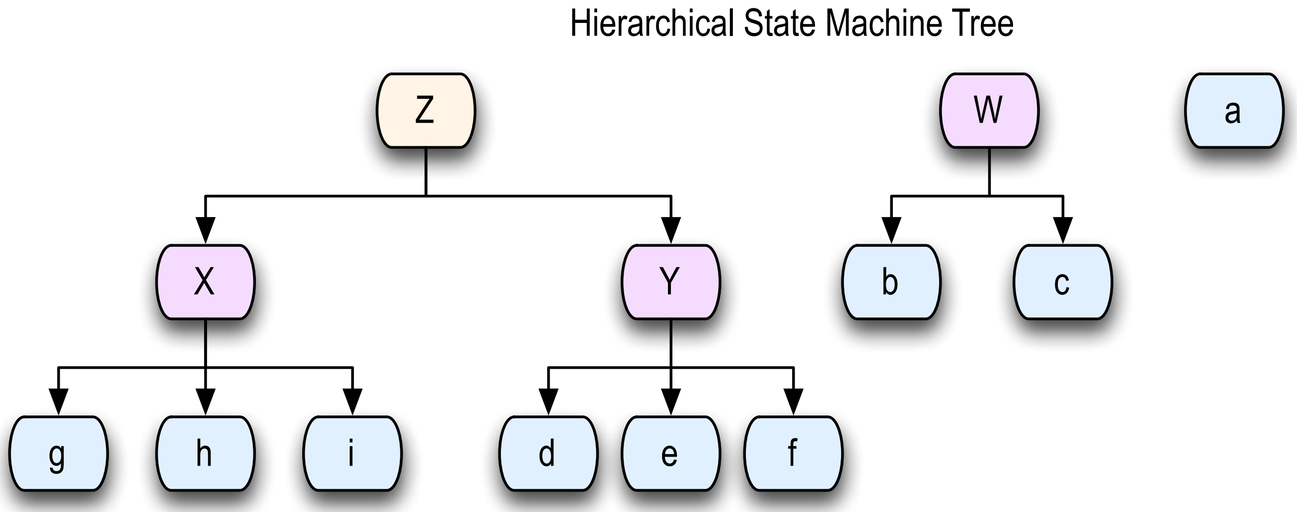


FloScript

Dichotomous Priorities

FloScript

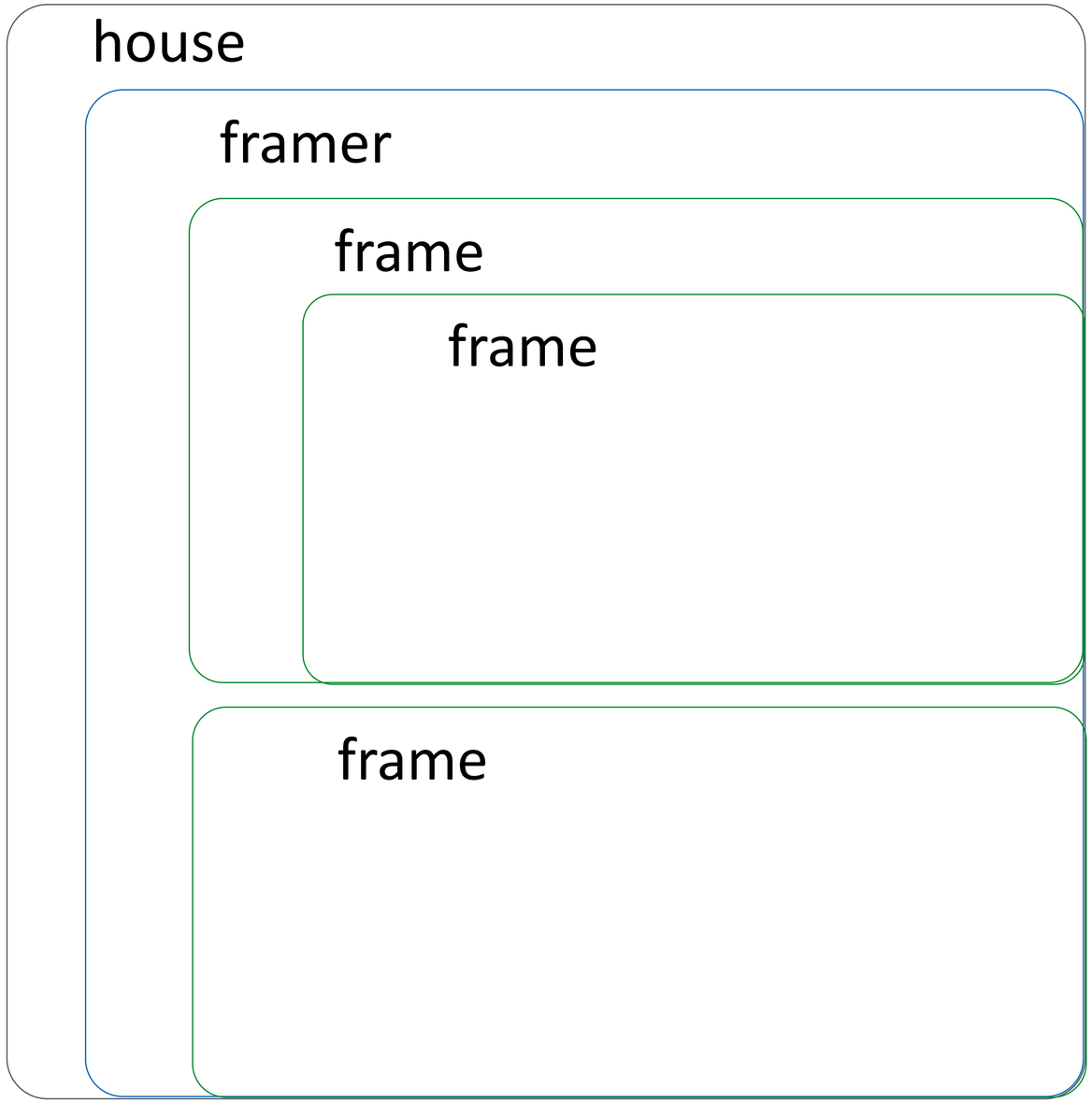
Transitions
- Change Context, Propriety
- Priority is Top-Down



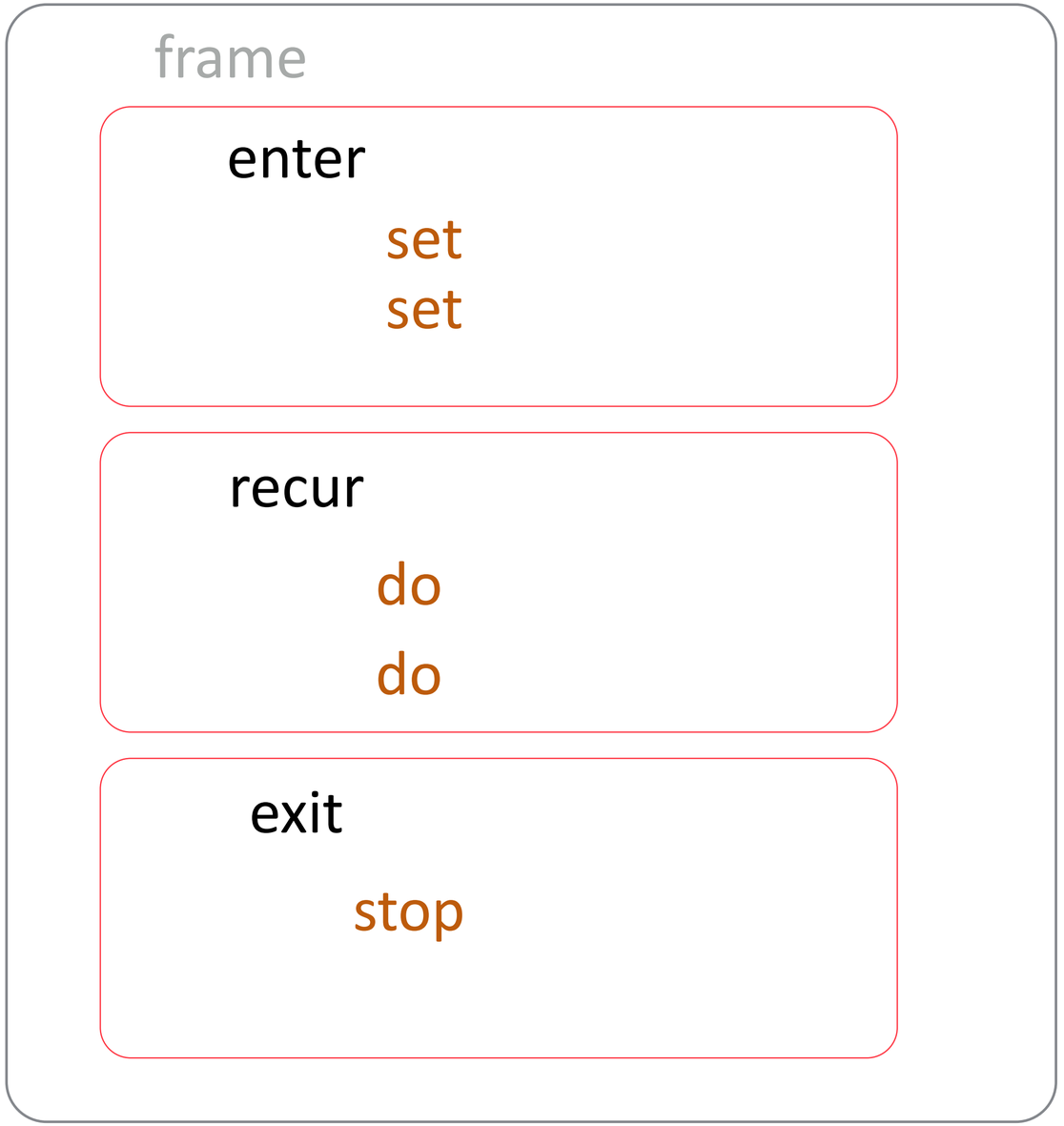
Actions
- Determine Contextual Behavior, Specificity
- Priority is Bottom-Up

Contexts:

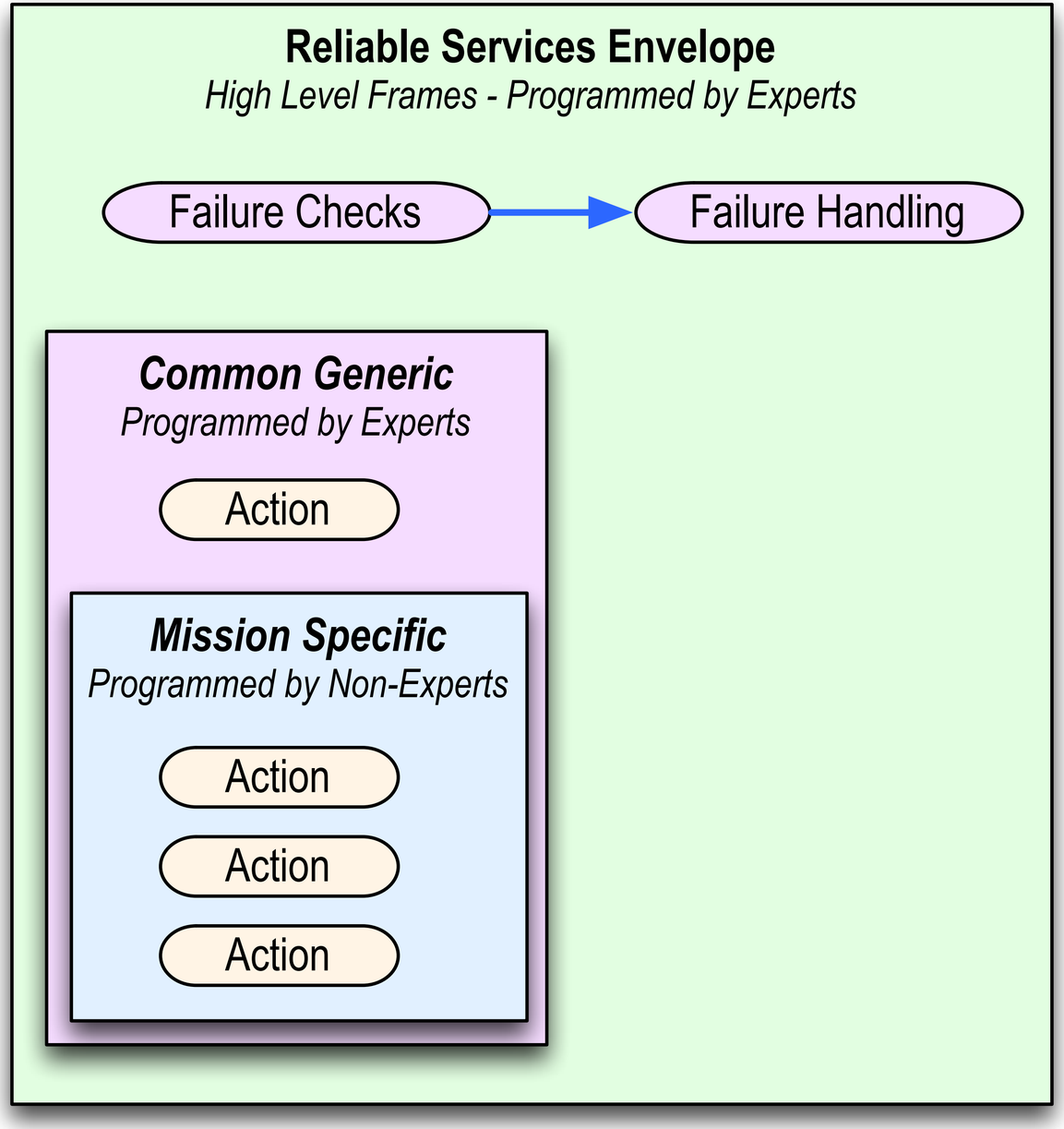
Frame of Reference "*Framing*"



Action Execution "*Actioning*"



Reliability



Universal Convenient Formalism for Program Synthesis

Universal *and* Convenient = Any meaningful program/algorithm can be practically represented, implemented and executed.

The formalism is Floscript which configures the automated reasoning engine.

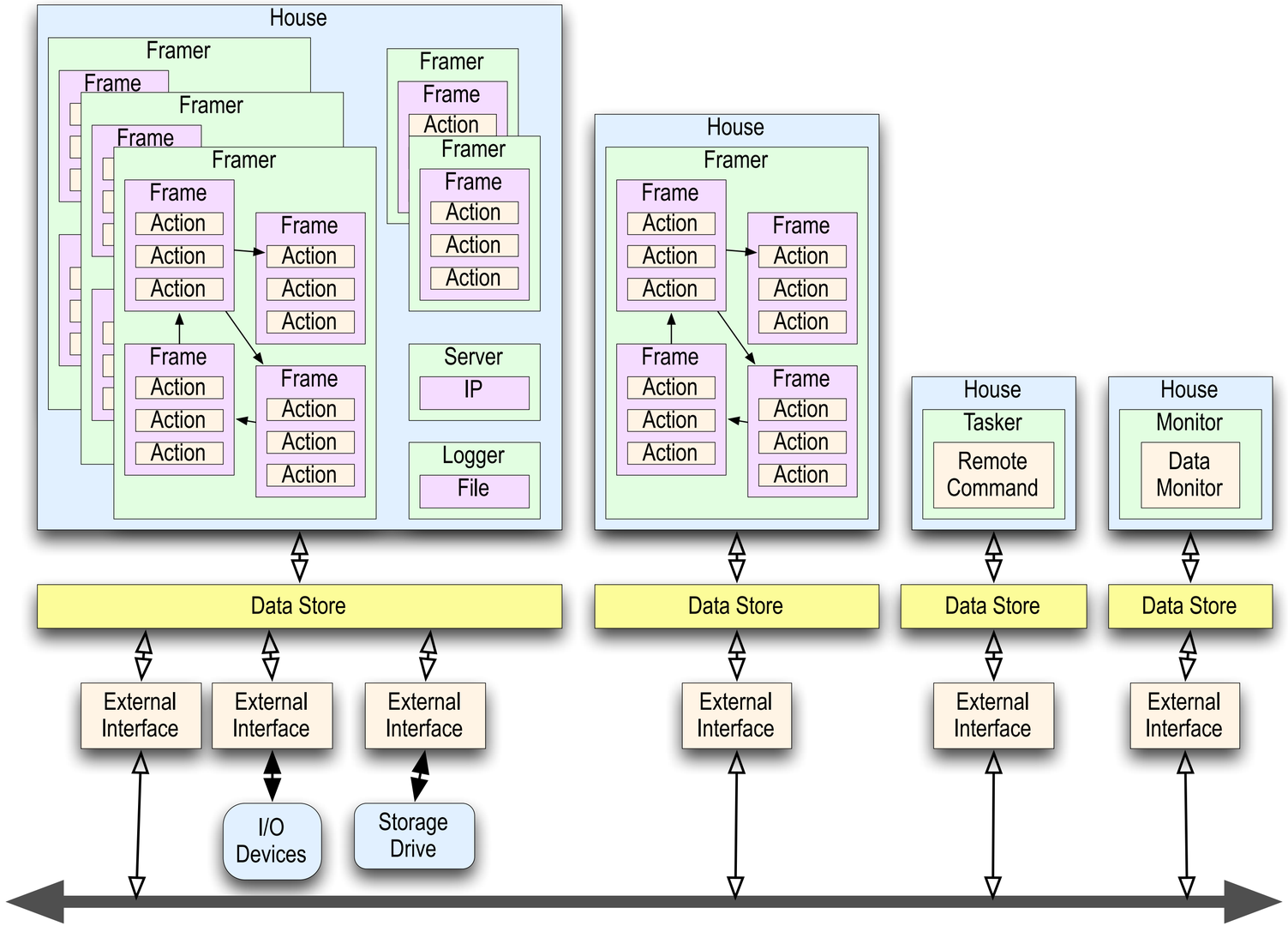
Floscript conveniently encodes everything needed to implement a decision trajectory generator with real-time UML-State Chart like expressive power:

Floscript includes, configuration, marshaling, concurrency, resources, hierarchical state transition, life cycle, staging contexts, communications, and persistence.

ioflo

FloScript

Big Picture



Framer
Pre Enter
for each Frame (top down)
Before Enter Actions

Transitions (except first time)
for each Frame (top down)
Eval Transitions

Enter
Implicit Actions (framer)
for each Frame (top down)
Entry Actions

Recur
Implicit Actions (framer)
for each Frame (top down)
Recur Actions

Exit
for each Frame (bottom up)
Exit Actions
Implicit Actions (framer)

Frame
Benter Actions
Explicit
Implicit (Aux Framers)

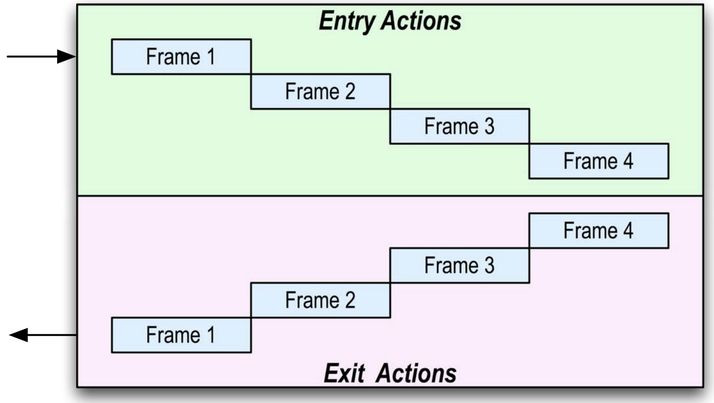
Transitions (except first time)
Explicit
Implicit (Aux Framers

Enter Actions
Explicit
Implicit (Aux Framers)

Recur Actions
Explicit
Implicit (aux framers)

Exit Actions
Implicit (Aux Framers)
Explicit

FloScript

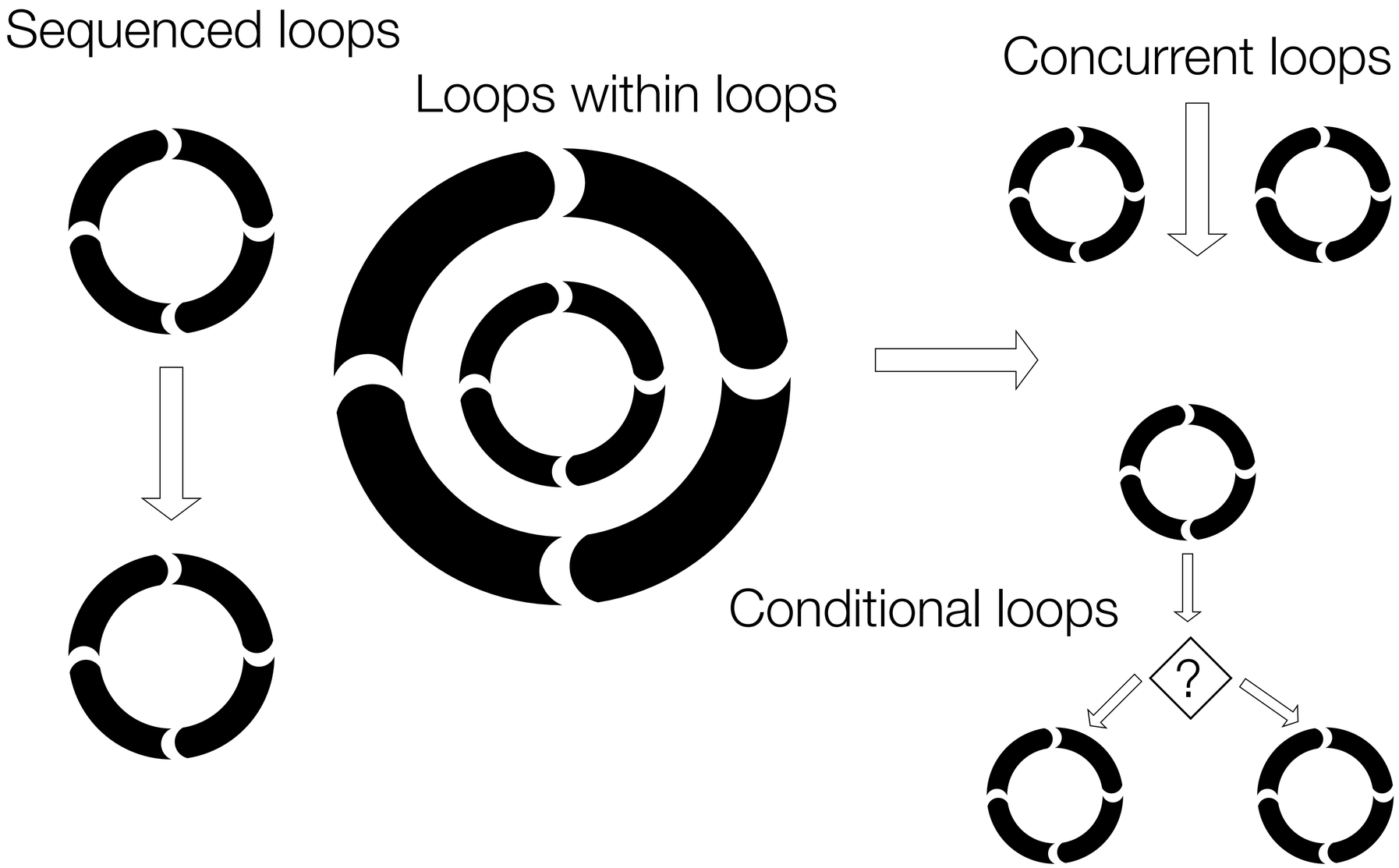


Autonomic Computing

Self-Managing

- Self-Configuring
- Self-Healing
- Self-Optimizing
- Self-Securing

Real persistently automated systems can have complex feedback loop topologies



Asynchronous Processing Approaches

Greenlets, Micro-threads, gevent, stackless python

Emulates conventional multi-thread/process model, join calculus, sleepy, continuous logic flow,

Event Loop Callbacks, Twisted, Node.js

Medusa/Reactor pattern, callback hell, discontinuous logic flow

Deferred, Python 3.5 asyncio futures, promises

Thread pool like scheduler, discontinuous logic flow

Yielded Delegated, generators/coroutines, Python 3.5 asyncio coroutines

Yielding scheduler, delegating coroutines, yieldy, continuous logic flow

Nested hybrid coroutines, Ioflo

Nested contextual yielding/delegating schedulers, continuous logic flow

Flow-Based Programming (FBP)

- Originated in 1970's by J.P. Morrison, contemporary with OOP & FP
- Relatively unknown but a lot of interest recently
- Think general-purpose data-flow programming
- Use it via a programming style, pattern, paradigm, or framework, not a language
- FBP is a simplifying unifying paradigm
- Distributed concurrent applications benefit most from FBP
- An FBP architecture may still be really useful even when not using an FBP Framework
- An FBP mindset may provide unique solution insights even when using OOP or FP

Flow-Based Programming Frameworks

Javascript

NoFlo

Flux

constructables.es

Python

ioflo

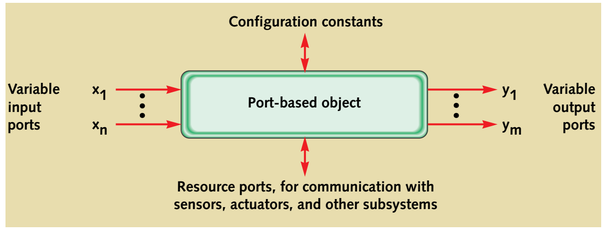
Pypes

PyF

Other

Expecco

PBO



More or less FBP

Java VM

Apache STORM

Spark

IBM InfoSphere Streams

Microsoft Azure Event Hubs

Python

ROS

Open Source Robotics Foundation

ERLANG

Other

MatLab Simulink

NATIONAL INSTRUMENTS LabVIEW

Replacement Independence

Behaviors (components) can be externally connected without any internal changes.

Composition of complex networks/graphs is conceptually simple

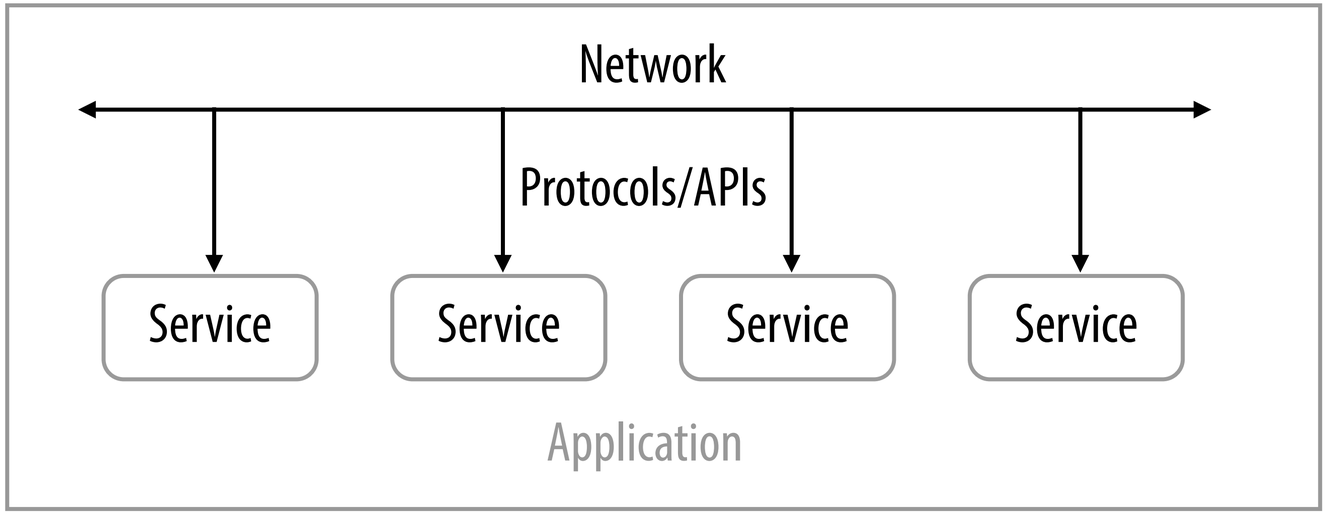
Because partitioning occurs intra-process/intra-host instead of inter-process/inter-host, distribution of behaviors across processor resources does not change behavior internals

Replacement Independence = Dependency Minimization

Replacement Independence = Flexibility, Robustness, Mobility

What is a Micro-Service Application?

Application composed of small single purpose services connected via lightweight protocols



Enables internal language agnosticism but external shared protocol (http/ReST/json)

Increased dev team autonomy and agility

Easier scaling

Decreased coupling Enhanced cohesion

Producer/Consumer architecture

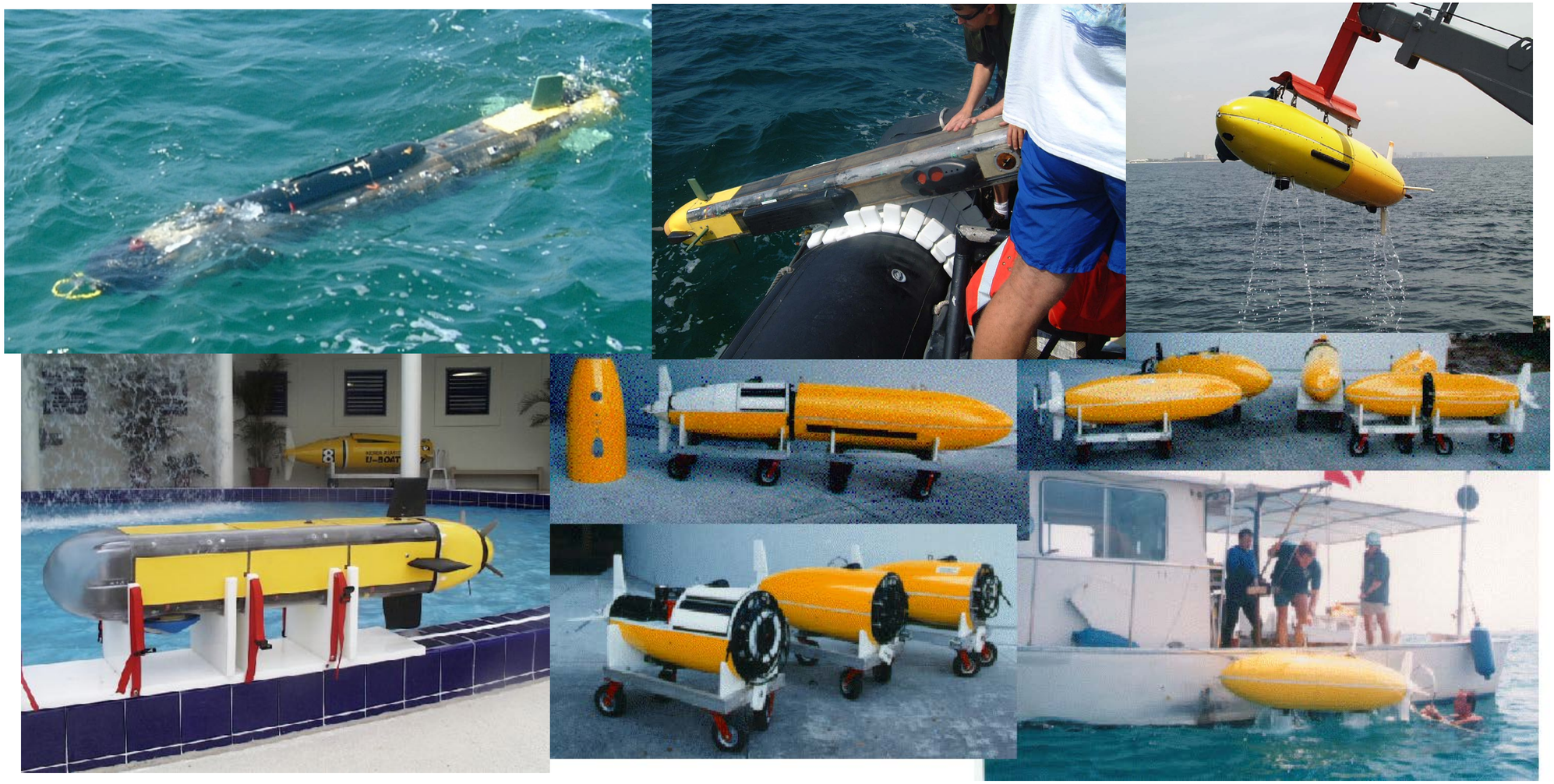
"High Replacement Independence"

Backup Slides

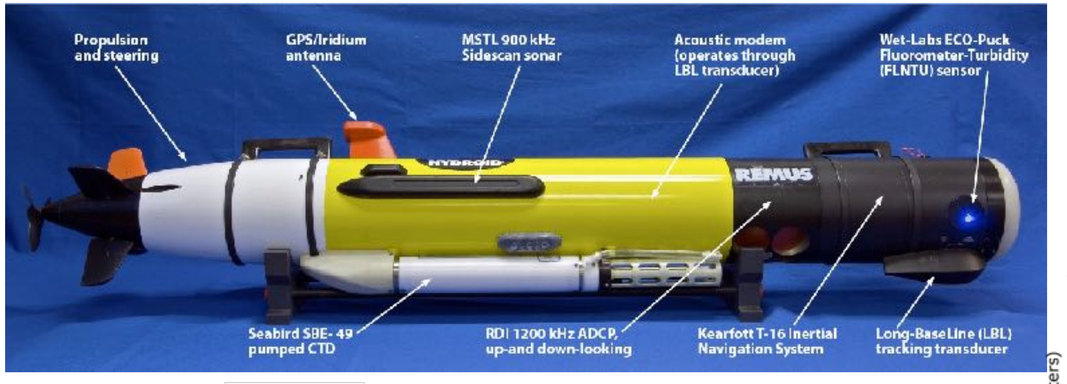
ASYNC ARCHITECTURE OVERVIEW

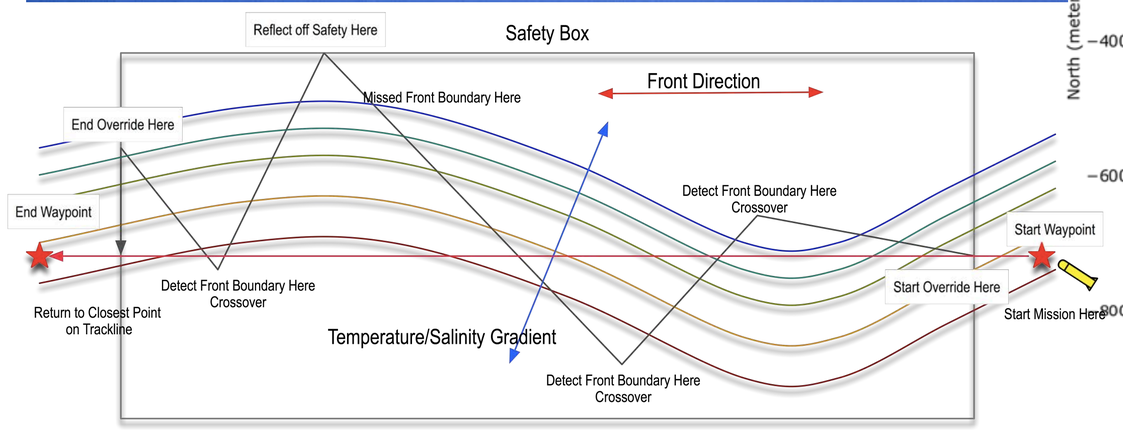
Samuel M. Smith Ph.D.
2019-08-30
sam@samuelsmith.org

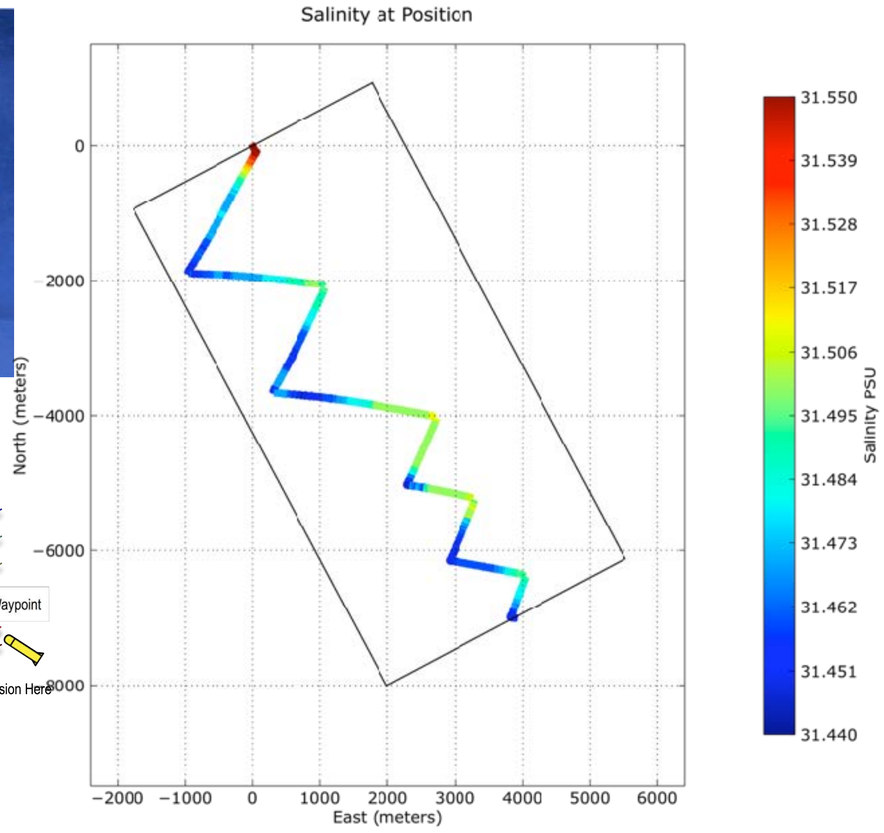
Autonomous Underwater Vehicles



Environmental Adaptive Sampling

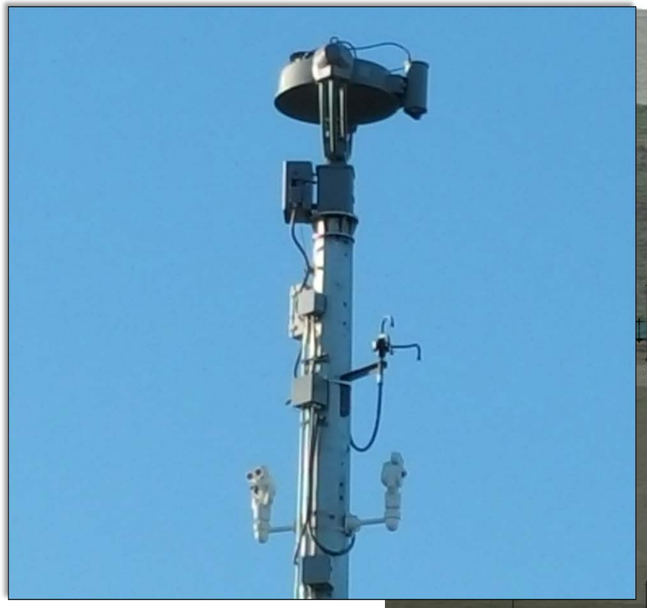


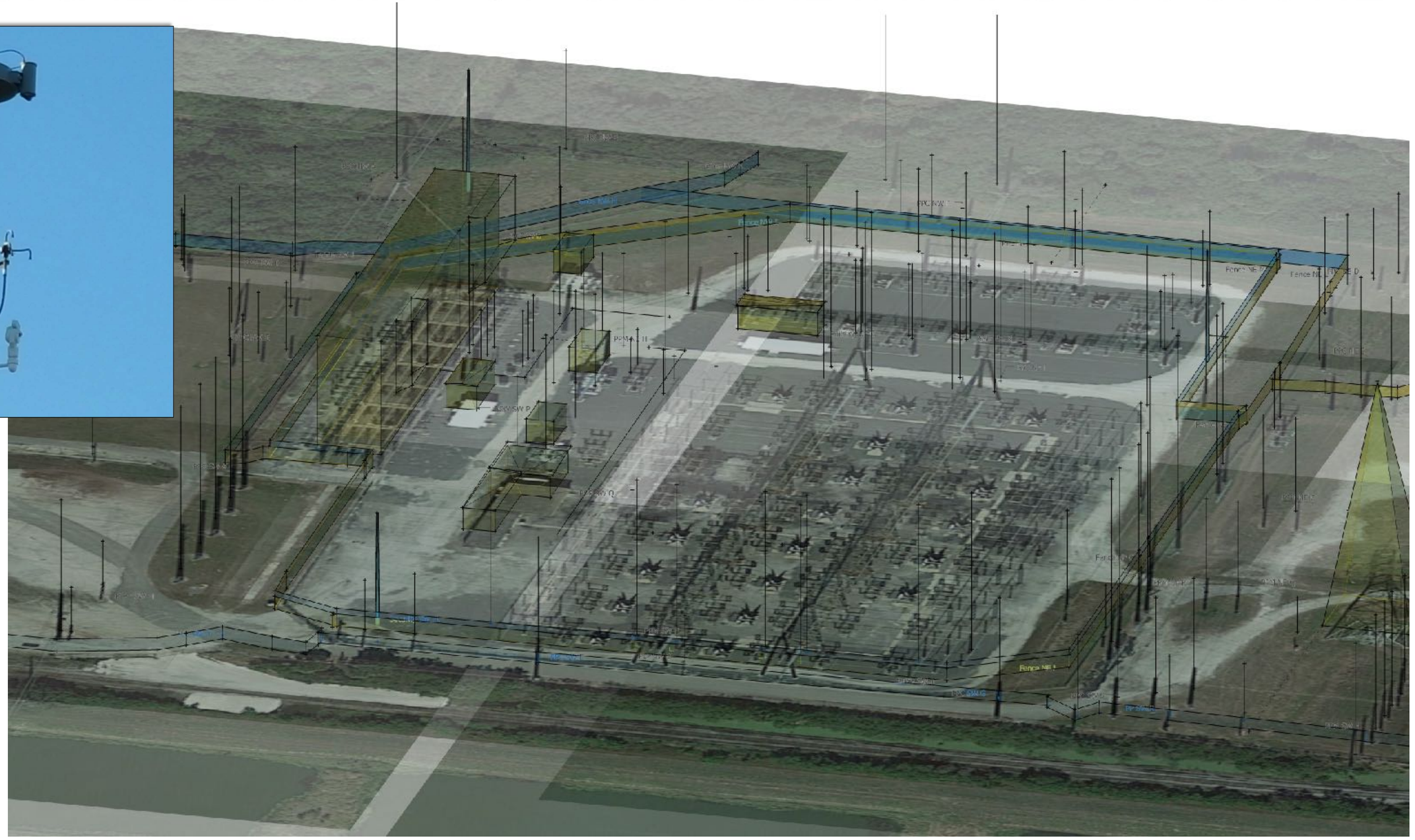




Shipboard & Building Automation

PHYSICAL SECURITY OF CRITICAL INFRASTRUCTURE

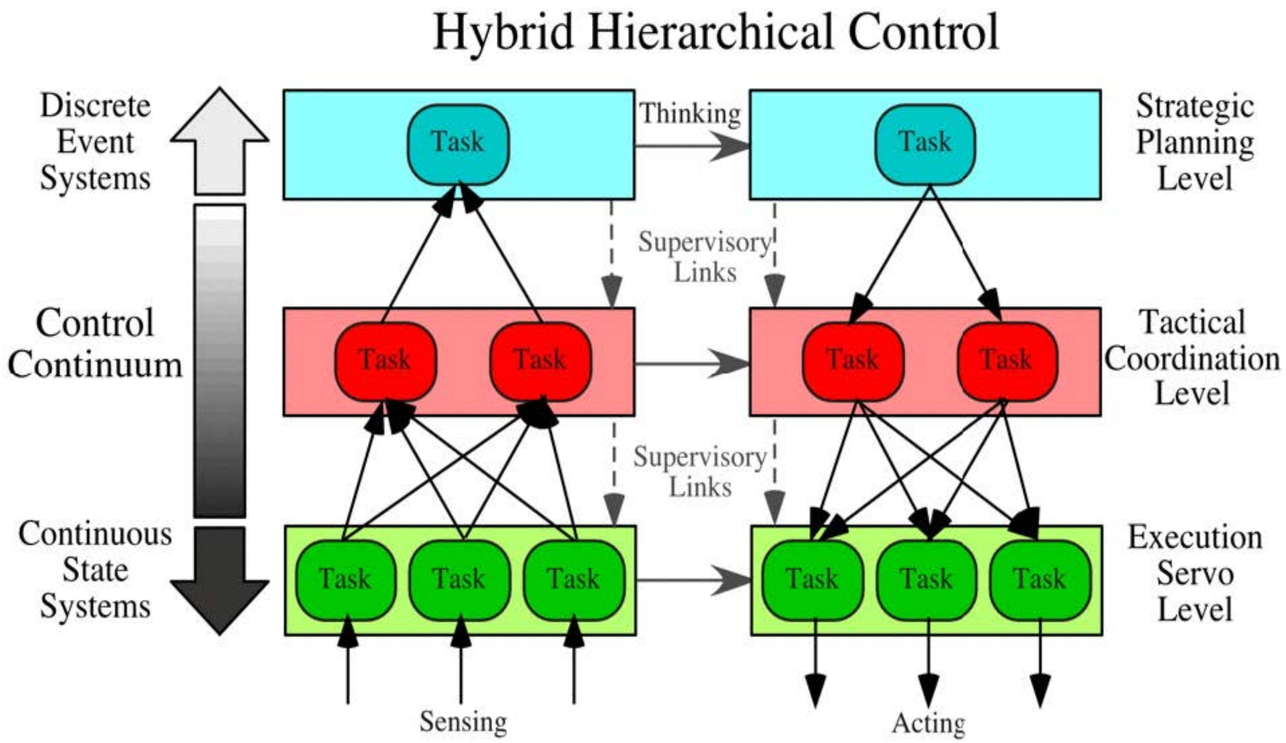


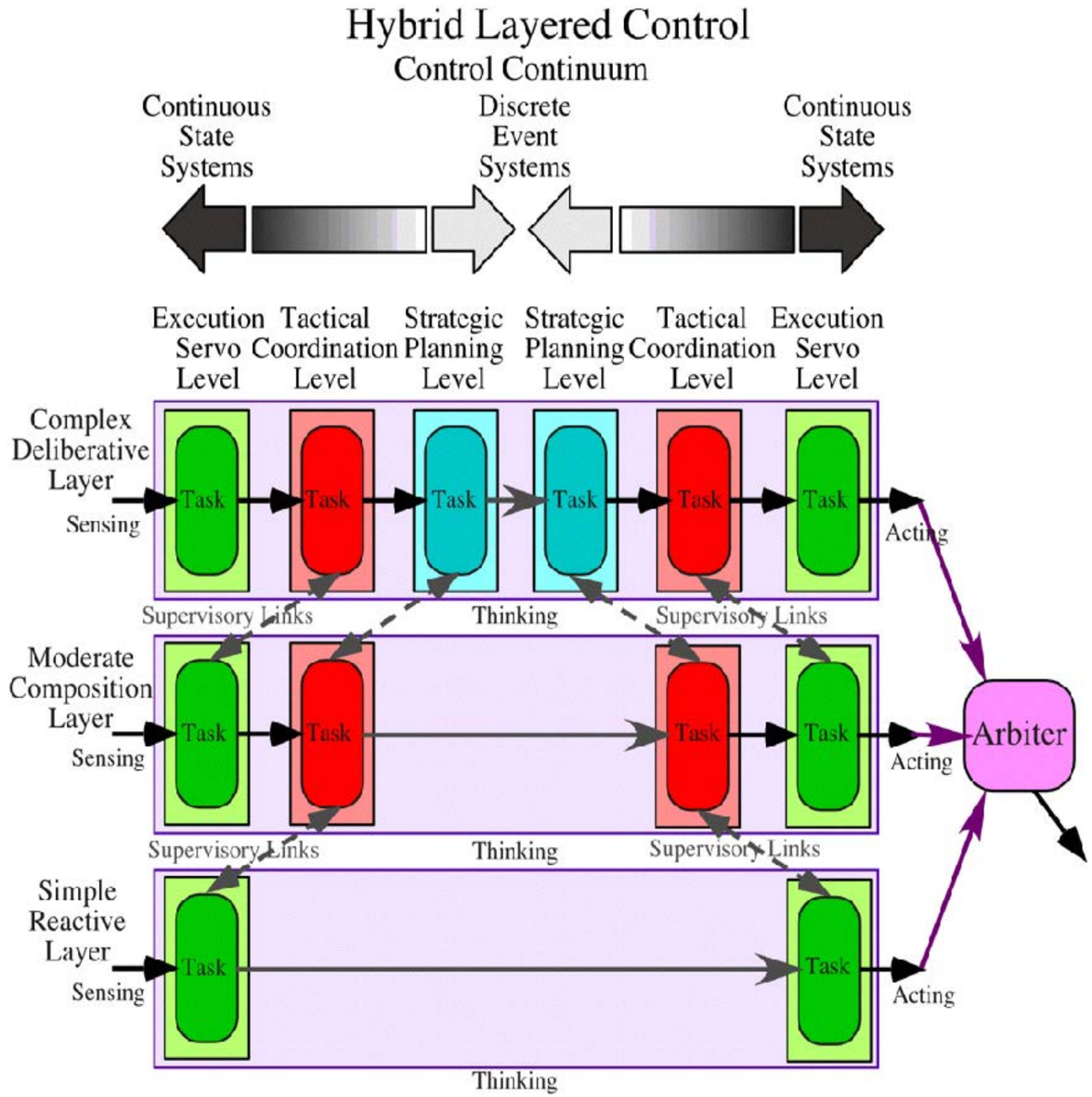


Complex System Control

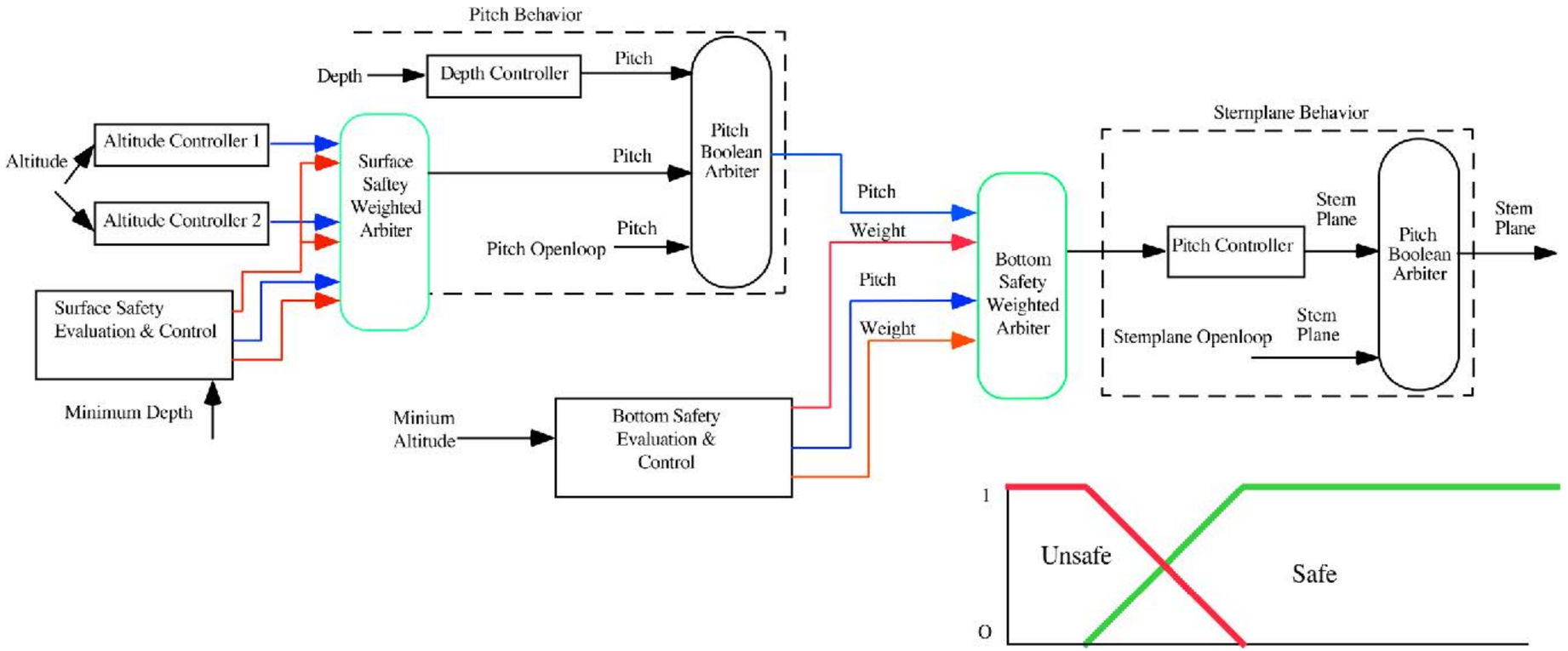
Strategic, Tactical, Executive

Deliberative, Compositive, Reactive





Automated Reasoning



Coordinated Control - Distributed Consensus

Groups of autonomous agents from different circumstances cooperating to achieve shared objective(s) in an environment of uncertainty

Think Decentralized Autonomous Organizations

Think Distributed Autonomic Services

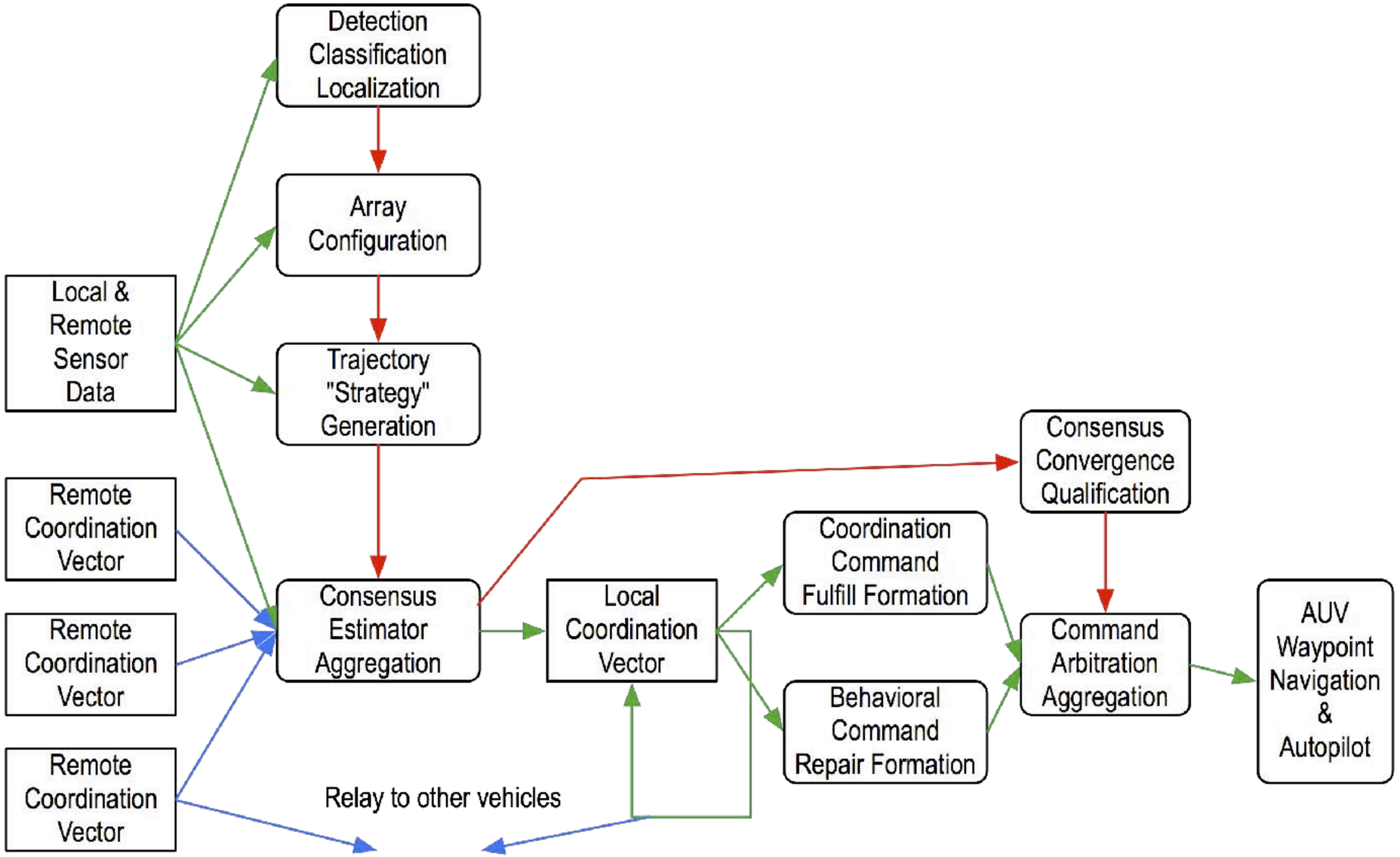
Coordinated Control Types

Leader-Follower

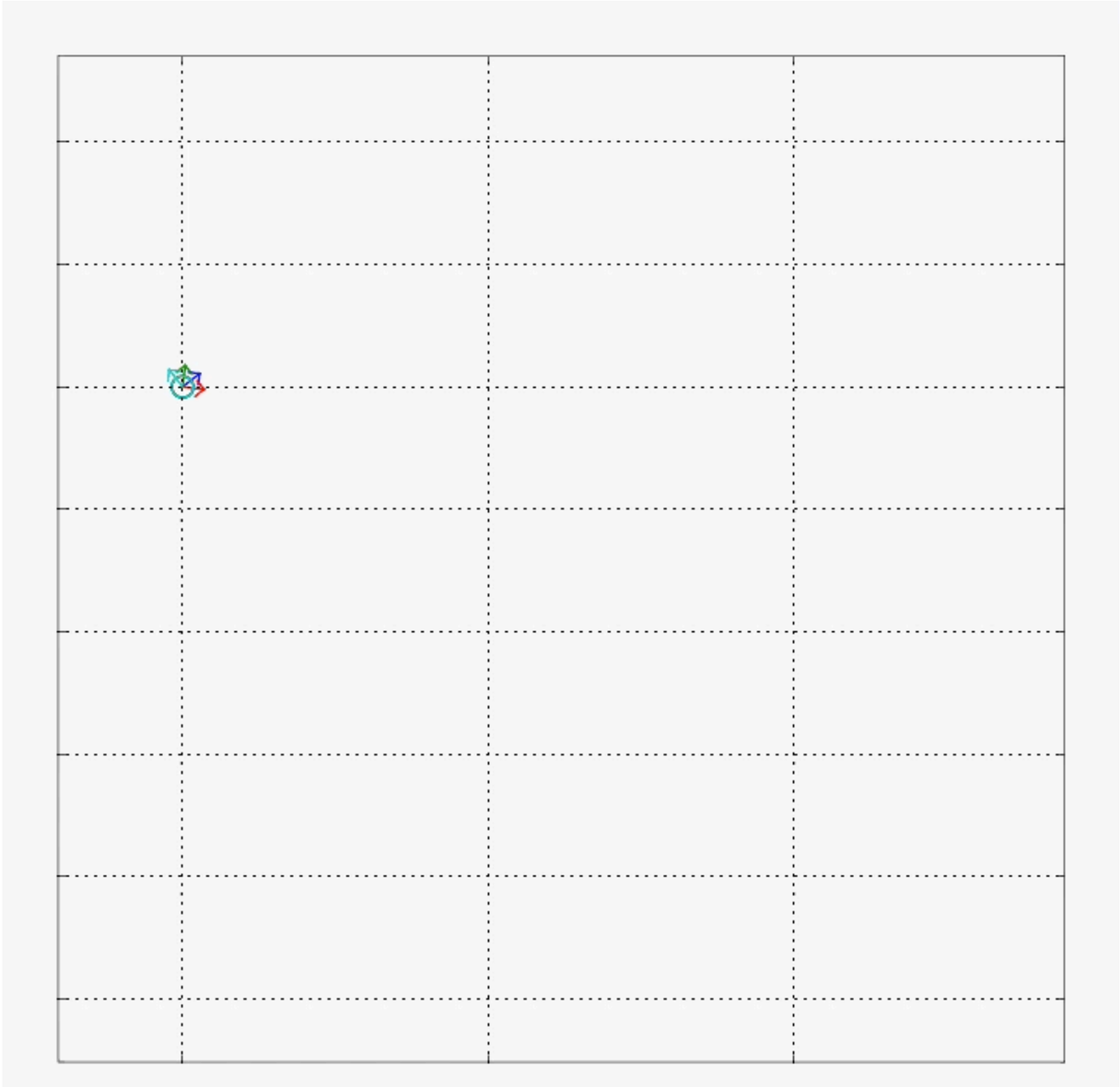
Behavioral

Role Based (Virtual Structure)

Role Based Coordinated Control



Cluster Control



Pattern Centric Reasoning/Learning

Humans deal with information uncertainty in an open world by using patterns or models to organize and distill information to take coherent action.

Patterns are flexible compositions of qualities, features and behaviors that can be matched against real world situations to generate an appropriate response.

Patterns bundle (generalize) disparate input cases (discontinuous decision/event space) into a finite set of flexible scenarios (models).

We want to synthesis programs from patterns.

A crucial problem is the *pattern representation problem*.

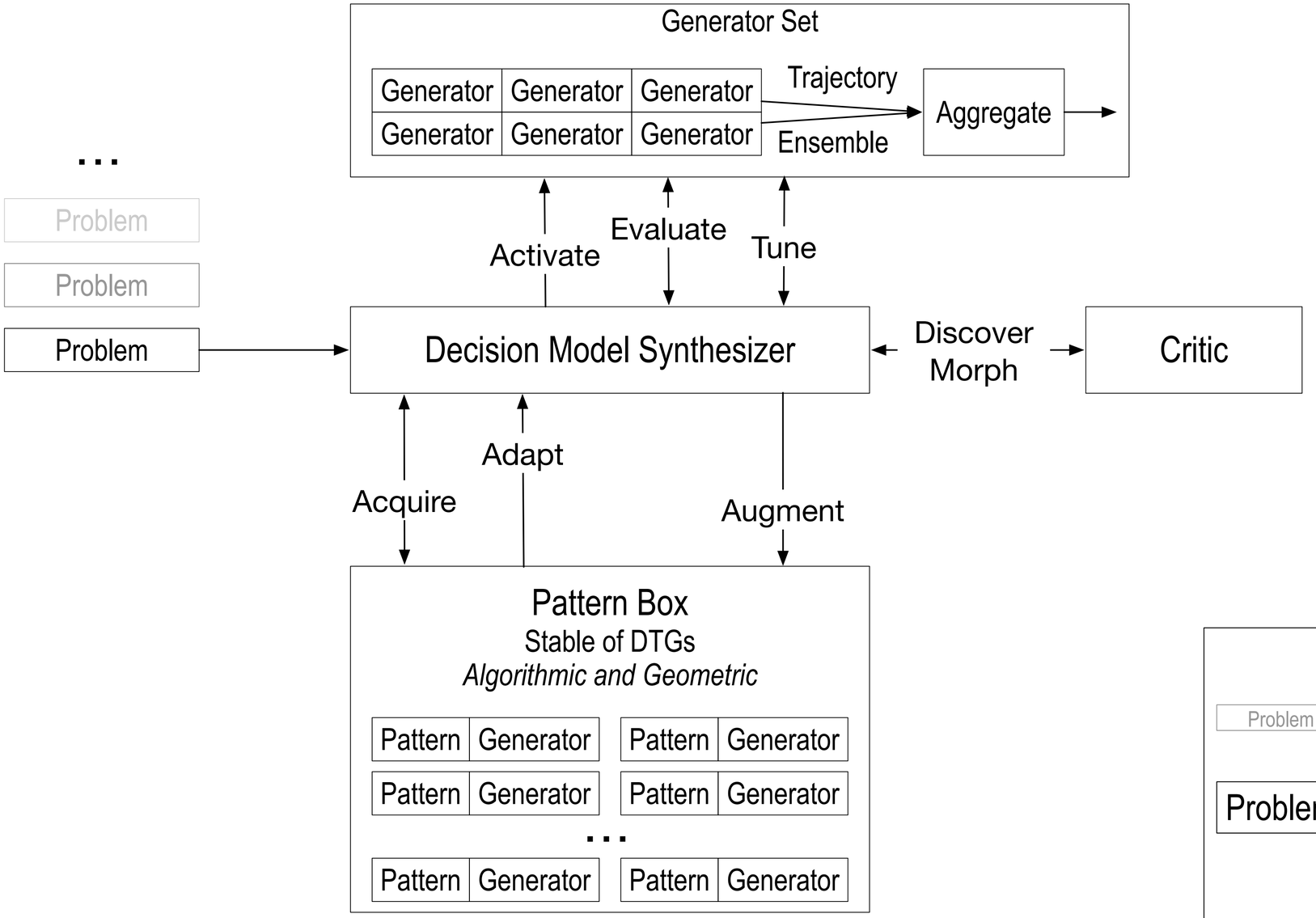
Need pattern richness. Only sufficiently rich pattern representations will achieve generalized learning.

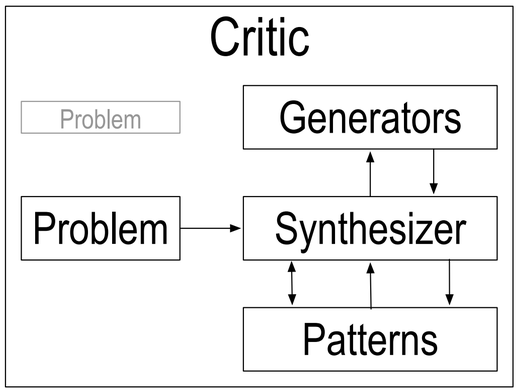
Insight: Humans match patterns not merely against *points* in decision/event space but against multi-step sequences of points in decision/events space. (ie models)

Deliberative reasoning evaluates predictions over multiple steps

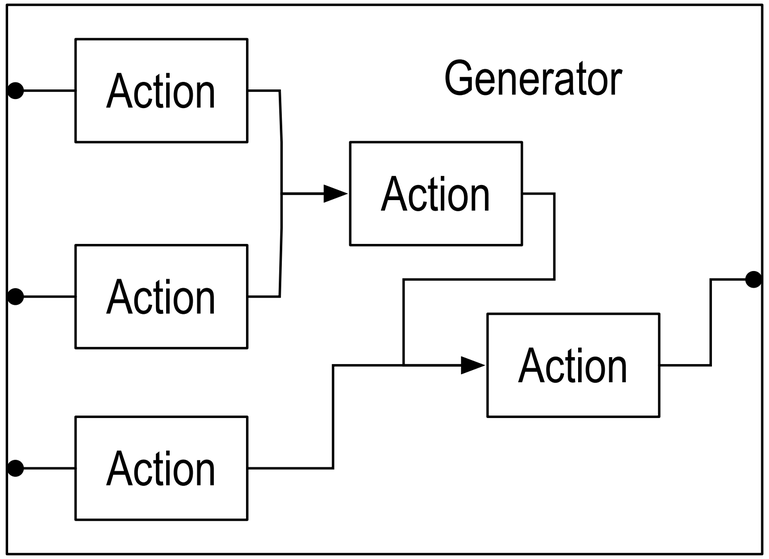
PCR

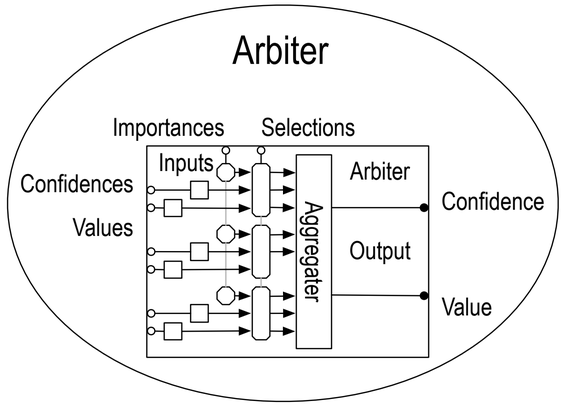
Pattern Centric Reasoning

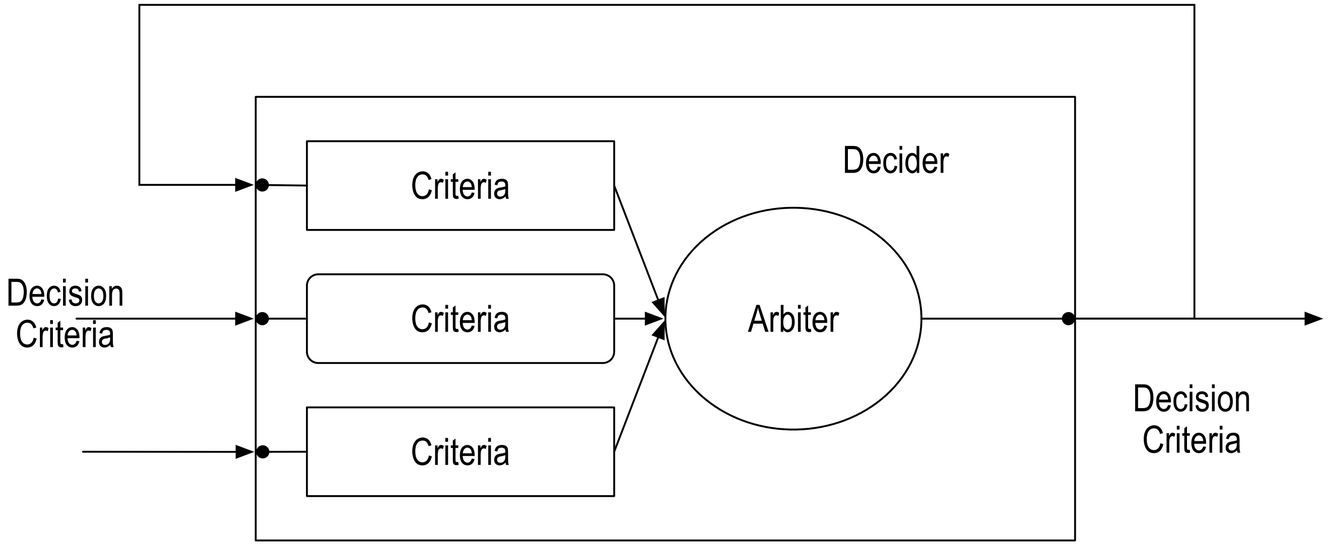


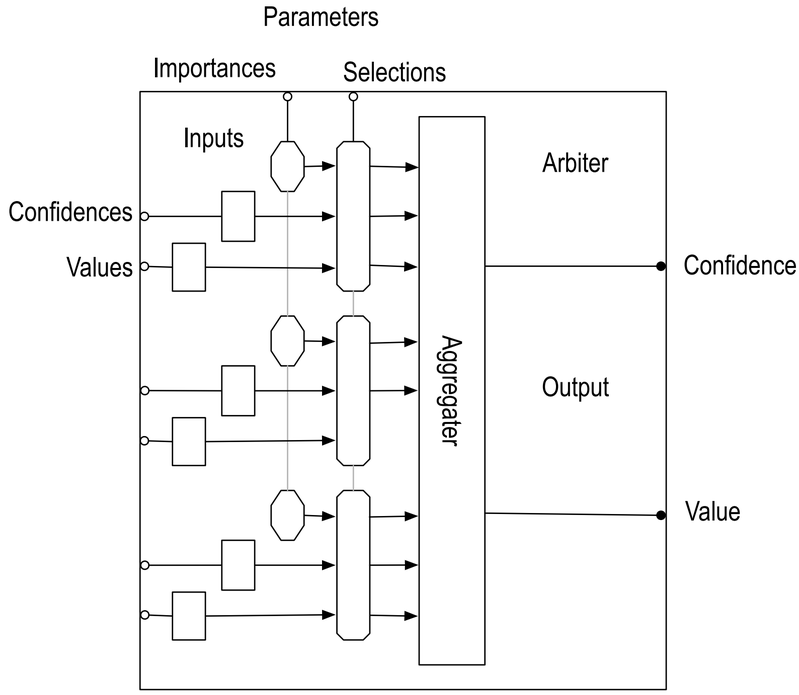


Components

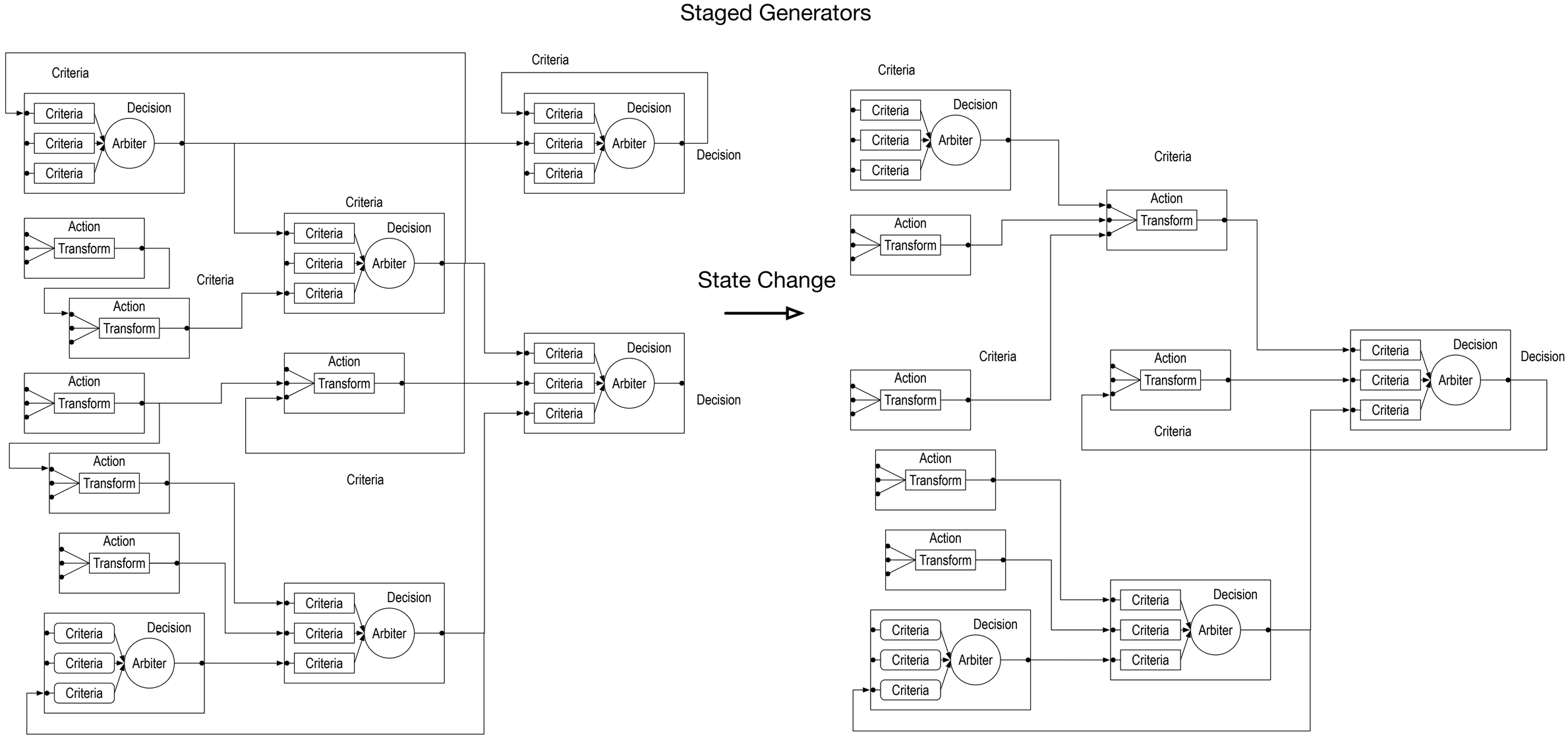

Generator
Action
Action
Action
Action
Action


Arbiter
Importances
Selections
Inputs
Confidences
Values
Aggregater
Arbiter
Confidence
Output
Value


Decider
Criteria
Criteria
Criteria
Decision
Criteria
Arbiter
Decision
Criteria

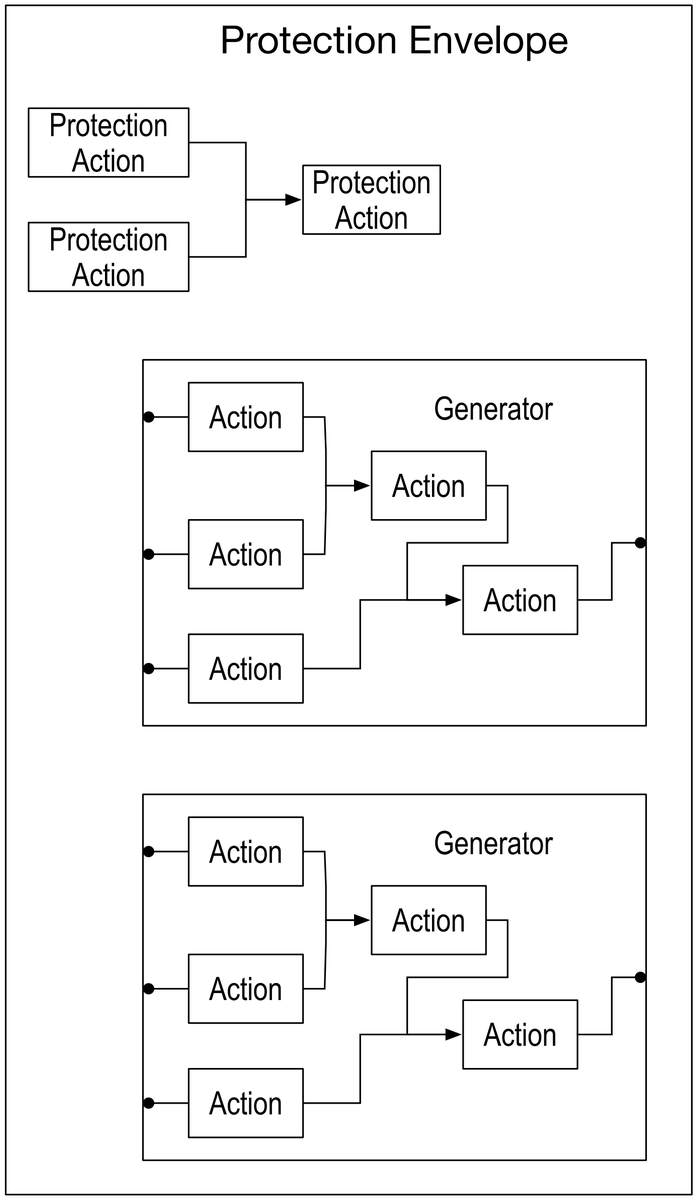

Parameters
Importances
Selections
Inputs
Confidences
Values
Aggregater
Arbiter
Confidence
Output
Value

Staged Generators



Autonomic = Self-**** Code

Protected Discovery



Complexity Management

Real Complexity = number of dependencies between elements of a software system

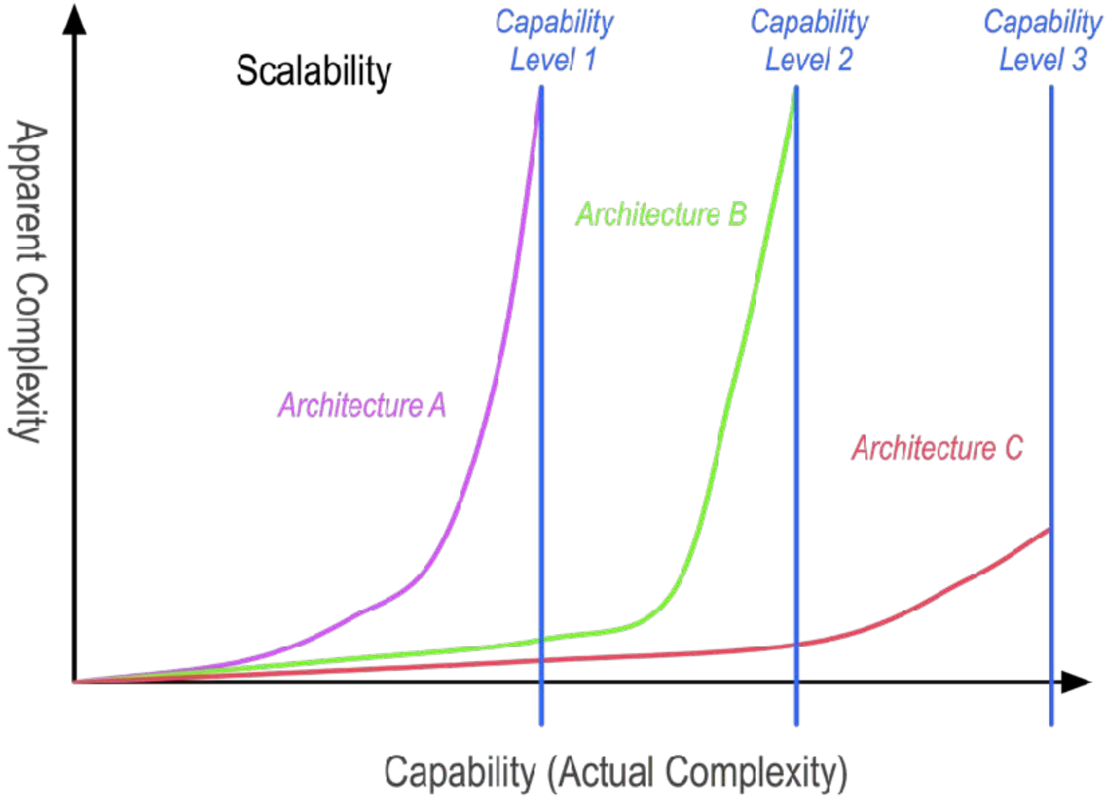
Apparent Complexity = number of dependencies that programmers must manage in order to make meaningful enhancements to software functionality

Perceived Risk = peril the programmer faces when attempting to add meaningful enhancements to software functionality.

Scalability

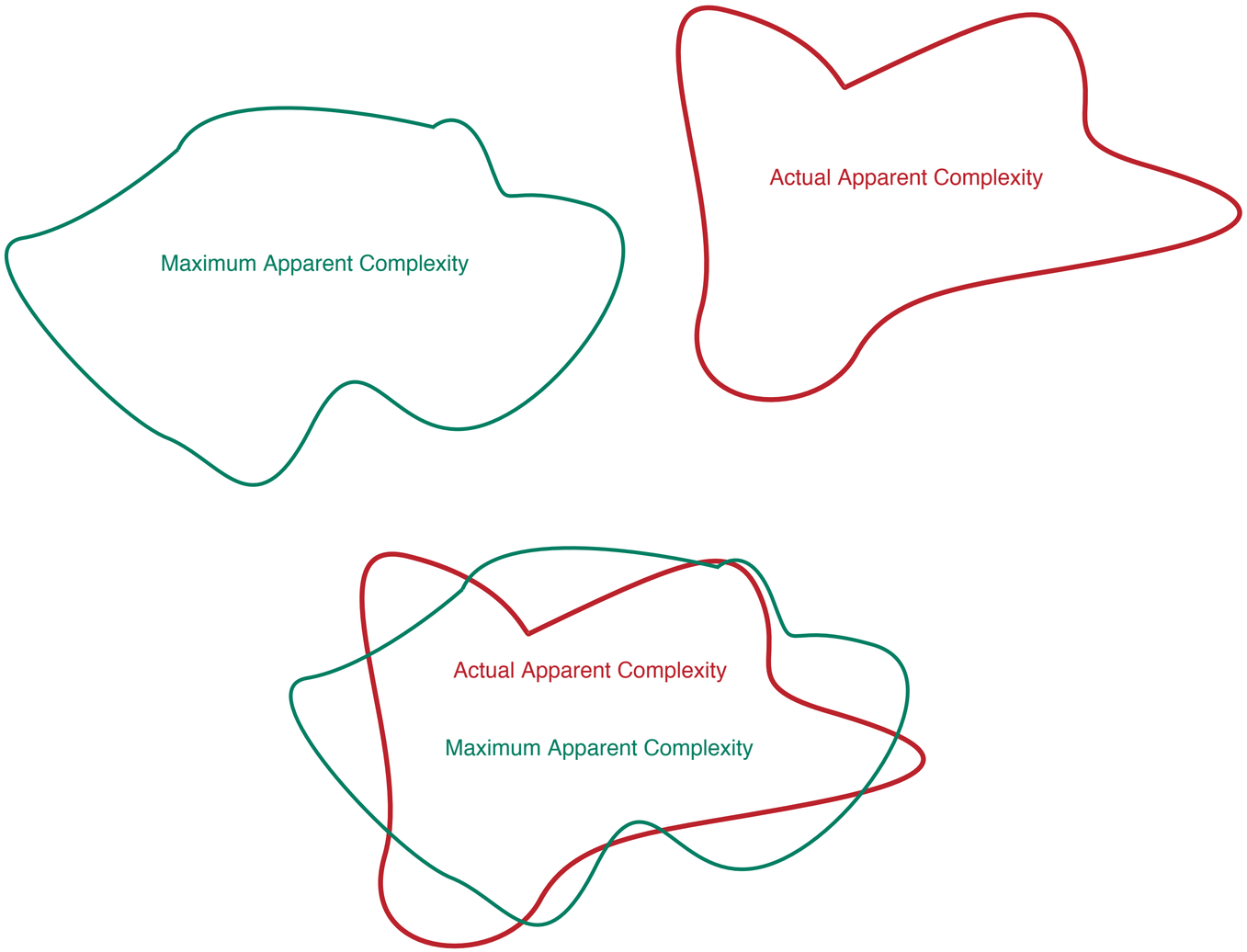
Higher levels of capability increase *real complexity* and may increase *apparent complexity*.

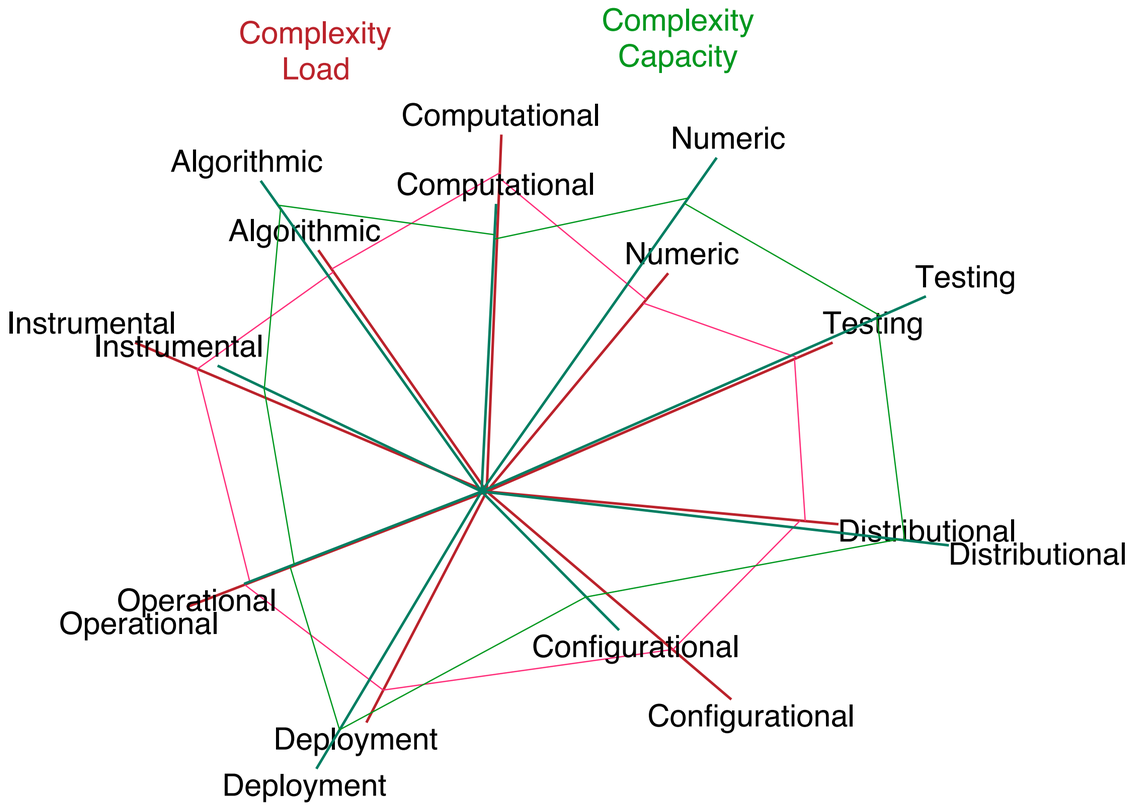
The principle limitation is programmer capacity not computational capacity



A given architecture/development team has a given maximal apparent complexity capacity.

When the actual apparent complexity starts to reach the maximal capacity, development becomes painful.





Replacement Independence

Behaviors (components) can be externally connected without any internal changes.

Composition of complex networks/graphs is conceptually simple

Because partitioning occurs intra-process/intra-host instead of inter-process/inter-host, distribution of behaviors across processor resources does not change behavior internals

Replacement Independence = Dependency Minimization

Replacement Independence = Flexibility, Robustness, Mobility

Online Program Synthesis

Contextualized generators

Adapting, tuning, training of generators

Interpolating generators

Composing new generators from sub-generators

Discovering new generators by predictive discovery of generator components and parameters.

Zero Trust Computing

Diffuse trust perimeter-less security model

Security, Privacy, Agency

Diffuse trust perimeter-less security principles

The network is always hostile, internally & externally; Locality is not trustworthy.

Every network interaction or data flow must be authenticated and authorized using best practice cryptography.

Inter-host communication must be end-to-end signed/encrypted and data must be stored signed/encrypted; Data is signed/encrypted in motion and at rest.

Policies for authentication and authorization must be dynamically modified based on behavior.

Policies must be governed by distributed consensus.

Decentralized Identity Inverts Service Architectures

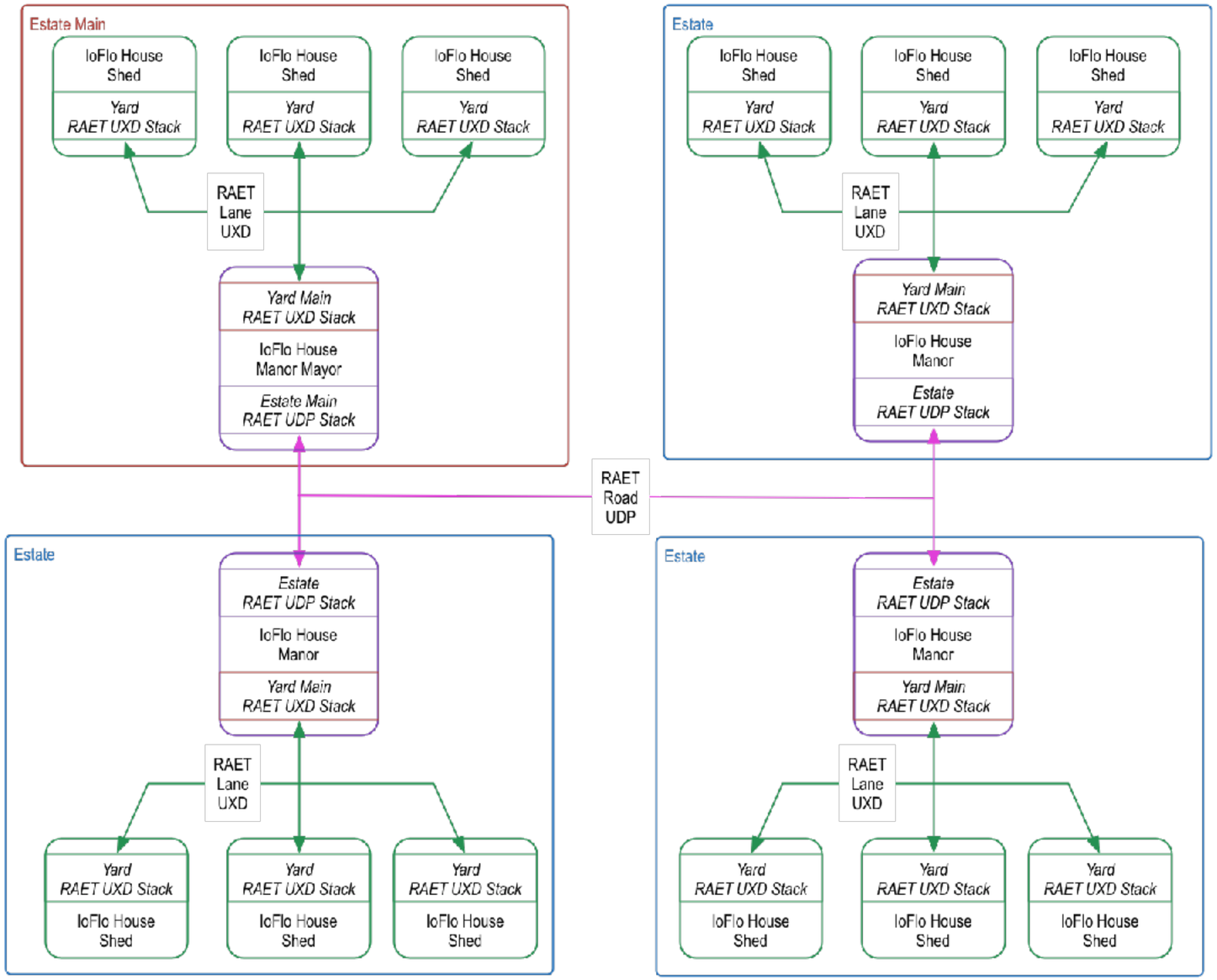
Conventional (centralized):

- Server creates identifiers (GUID, Database primary keys)
- Server timestamps
  - event ordering relative to server
- Server manages keys,
  - AuthN/AuthZ is indirect via client to server proxy
- Perimeter Security
- Server is source of truth
  - Server controls changes/updates to resources
  - Signed at rest problematic
  - Encrypted at rest problematic
- Server's role is 2nd party in two party transactions between client to server to client.

Unconventional (decentralized):

- Client creates identifiers (DIDs)
- Client timestamps
  - event ordering relative to client
- Client manages keys
  - AuthN/AuthZ is direct peer-to-peer
- Perimeterless Security
  - Client is source of truth
  - Client controls changes/updates to resources
  - Server cannot make changes
  - Client signs at rest
  - Client encrypts at rest
- Server's role is either:
  - Trusted 3rd party in 3 (multi) party transactions between 2 (or more) clients and server
  - Agent or proxy for a client in two party transaction with another client.

RAET Metaphor



RAET Features

https://github.com/RaetProtocol/raet

Peer-to-peer exchange of long term keys with multiple levels of control over how the exchange occurs.

Peer-to-peer implementation of the CurveCP handshake that allows for either side of a pair of hosts to initiate the handshake and resolves simultaneous CurveCP handshakes.

Exchanges name, role, and mode identifiers. Completed handshake authenticates the name for data flow routing, the role for authorization, and the mode for how the node operates on a given channel.

Extensible protocol that makes a trade-off on the side of simplicity and ease of debugging with a flexible header and a flexible data body.

The header by default is ascii readable and achieves compactness using a default value policy. This makes the ascii header on average about the same size as a full binary header.

Because the header is not fixed it is easily extensible and future proof.

Using routed data flows on top of the RAET channel credentials enables a data flow router to easily implement authorization policy. Where Authorization is per data flow address. Dynamic authorization policy can be easily implemented via a lookup table associated with each router.

RAET Reliable Asynchronous Event Transport

https://github.com/RaetProtocol/raet

End-to-End Encryption and Signing with:

NaCL, X22519, CurveCP Handshake, Curve22519, Salsa20/20 Poly1305

Python with LibNacl, Ioflo

UDP for interhost communication

Unix domain sockets for interprocess communications

Presence Support

Truly asynchronous architecture with non-blocking I/O

Peer-to-peer bidirectional Curve-CP handshake

FloScript

Incremental Encapsulation

Framing *concurrency*

Framers

Auxiliary Framers

Master/Slave Framers

Nested Frames

FloScript

Incremental Encapsulation

Actioning *components*

FloScript Verbs

Ioflo library Python Actions/Behaviors for do verb objects

Custom Python coded do verb objects

Custom coded Python C Extensions for do verb objects

References

http://ioflo.com
https://github.com/ioflo
https://github.com/ioflo/ioflo_manuals

http://www.jpaulmorrison.com/fbp/
http://www.jpaulmorrison.com/fbp/links_external.html
https://flowbasedprogramming.wordpress.com/article/flow-based-programming/
http://www.amazon.com/Flow-Based-Programming-2nd-Application-Development/dp/1451542321/ref=sr_1_1?ie=UTF8&qid=1427910581&sr=8-1&keywords=flow+based+programming

http://wiki.ros.org/ROS/Tutorials

Flow-Based Programming Resources

Flow-Based Programming, 2nd Ed. May 14, 2010

http://www.jpaulmorrison.com/fbp/

http://www.jpaulmorrison.com/fbp/links_external.html

https://flowbasedprogramming.wordpress.com/article/flow-based-programming/

Port Automata/ Port Based Objects

Port Automata and the Algebra of Concurrent Processes, Steenstrup & Arbib 1982

SoftwareComponentsForRealTime, Stewart 2000

FloScript Line Oriented, Contextual, Declarative Configuration Language

not block oriented

Declaration Sentence: `verb [object] [prepositional-phrases]`

```
done
frame build in setup
set speed to 5
framer create at 0.5 be active first build
```

`[adjectives …] kind`

Verb Object: `do salt eventer`

Intelligent Autonomy-Autonomic-Automation Programming Framework

Open Source: MIT License - ioflo.com - github.com/ioflo

Automation Operating System

Micro Threaded Concurrent Execution Engine With Non Blocking I/O

Automated Reasoning Engine

Flow Based Programming Framework

Hierarchical Action Framework

Discrete Event Simulation Framework

Dependency Injection Framework

Universal PubSub

FloScript: DSL for Convenient Configuration

Machine Learning - Machine Intelligence Infrastucture

Universal Name-Spaced Pub/Sub Interface for Intra application communication

high replacement independence

Addressing Modes:

direct indirect-absolute

```
put name sam eyes blue into .person.detail
```

indirect-relative-implied indirect-relative-root

```
set speed by blue in myapp.setup
```

direct indirect-relative-frame

```
put name sam eyes blue into detail of frame start
```

direct indirect-relative-framer

```
put true into good of framer
```

FloScript

RAET Reliable Asynchronous Event

https://github.com/RaetProtocol/raet

- RAET
  - Micro threaded architecture with non-blocking I/O
  - Micro threaded Event Pub/Sub separate from socket based transport layer
  - UDP sockets
  - Better observability and management of performance under load
  - Transactions
  - Unix domain sockets for interprocess communications

Creating a Do verb object/behavior "deed" with decorator

```
def doify(name,
          base=None,
          registry=None,
          parametric=None,
          inits=None,
          ioinits=None,
          parms=None):
    """ Parametrized decorator function that converts the decorated function
        into an Actor sub class with .action method and with class name name
        and registers the new subclass in the registry under name.
        If base is provided then register as subclass of base.
        Default base is Doer

        The parameters registry, parametric, inits, ioinits, and parms if provided,
        are used to create the class attributes for the new subclass
    """
```

Example

```
@doify('IoserveServerClose', ioinits=odict(valet=""))
def ioserveServerClose(self, **kwa):
    """
    Close server in valet

    Ioinit attributes:
        valet is a Valet instance

    Context: exit

    Example:
        do ioserve server close at exit
    """
    if self.valet.value:
        self.valet.value.servant.close()

        console.concise("Closed server '{0}' at '{1}'\n".format(
                                self.valet.name,
                                self.valet.value.servant.eha))
```

Valet WSGI HTTP Server

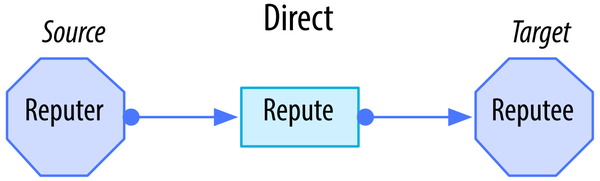
```
class Valet(object):
    """
    Valet WSGI Server Class
    """
    Timeout = 5.0  # default server connection timeout

    def __init__(self,
                 app=None,
                 reqs=None,
                 reps=None,
                 servant=None,
                 store=None,
                 name='',
                 bufsize=8096,
                 wlog=None,
                 ha=None,
                 host=u'',
                 port=None,
                 eha=None,
                 scheme=u'',
                 timeout=None,
                 **kwa):
        """
        Initialization method for instance.
        app = wsgi application callable
        reqs = odict of Requestant instances keyed by ca
        reps = odict of running Wsgi Responder instances keyed by ca
        servant = instance of Server or ServerTls or None
        store = Datastore for timers

        kwa needed to pass additional parameters to servant

        if servantinstances are not provided (None)
        some or all of these parameters will be used for initialization

        name = user friendly name for servant
        bufsize = buffer size
        wlog = WireLog instance if any
        ha = host address duple (host, port) for local servant listen socket
        host = host address for local servant listen socket, '' means any interface on host
        port = socket port for local servant listen socket
        eha = external destination address for incoming connections used in TLS
        scheme = http scheme u'http' or u'https' or empty

        """
```

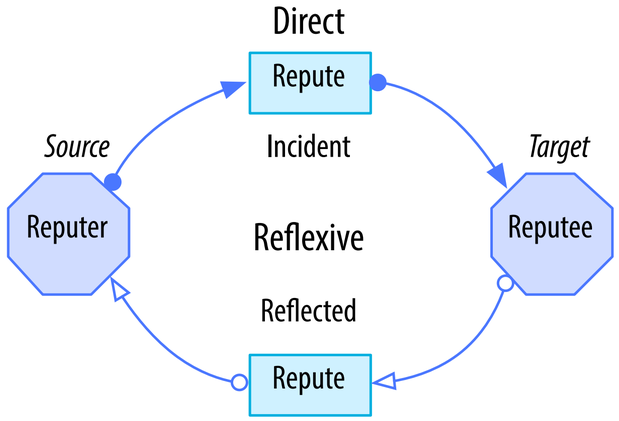
Patron HTTP Client

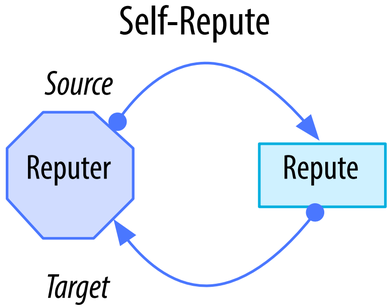
```
lass Patron(object):
    """
    Patron class nonblocking HTTP client connection manager
    """
    def __init__(self,
                 connector=None,
                 requester=None,
                 respondent=None,
                 store=None,
                 name='',
                 uid=0,
                 bufsize=8096,
                 wlog=None,
                 hostname='127.0.0.1',
                 port=None,
                 scheme=u'',
                 method=u'GET',  # unicode
                 path=u'/',  # unicode
                 headers=None,
                 qargs=None,
                 fragment=u'',
                 body=b'',
                 data=None,
                 fargs=None,
                 msg=None,
                 dictable=None,
                 events=None,
                 requests=None,
                 responses=None,
                 redirectable=True,
                 redirects=None,
                 **kwa):
```

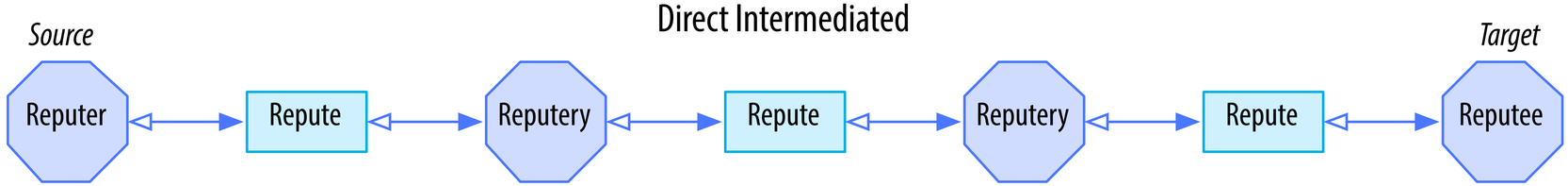
```
"""
        Initialization method for instance.
        kwa needed to pass other init parameters to connector

        connector = instance of Outgoer or OutgoerTls or None
        requester = instance of Requester or None
        respondent = instance of Respondent or None

        if either of requester, respondent instances are not provided (None)
        some or all of these parameters will be used for initialization

        name = user friendly name for connection
        uid = unique identifier for connection
        bufsize = buffer size
        wlog = WireLog instance if any
        host = host address or hostname of remote server
        port = socket port of remote server
        scheme = http scheme
        method = http request method verb unicode
        path = http url path section in unicode
               path may include scheme and netloc which takes priority
        qargs = dict of http query args
        fragment = http fragment
        headers = dict of http headers
        body = byte or binary array of request body bytes or bytearray
        data = dict of request body json if any
        fargs = dict of request body form args if any
        msg = bytearray of response msg to parse
        dictable = Boolean flag If True attempt to convert body from json
        events = deque of events if any
        requests = deque of requests if any each request is dict
        responses = deque of responses if any each response is dict
        redirectable = Boolean is allow redirects
        redirects = list of redirects if any each redirect is dict
        """
```

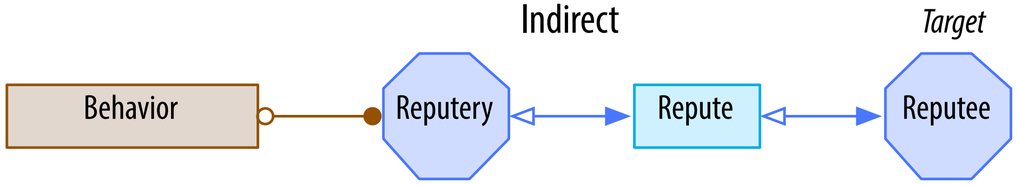
Reputing

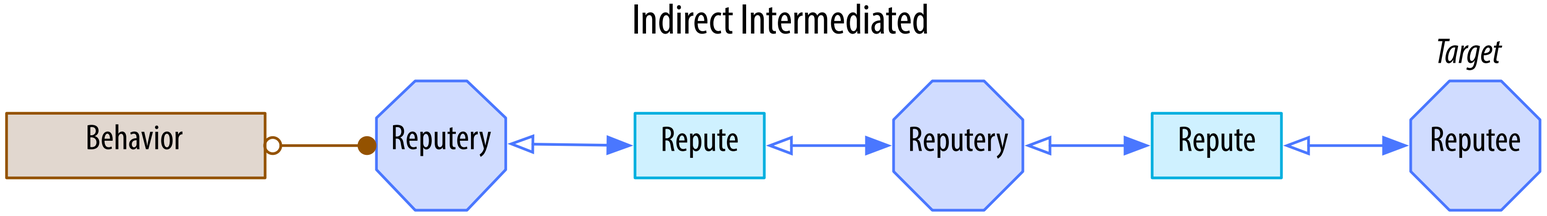






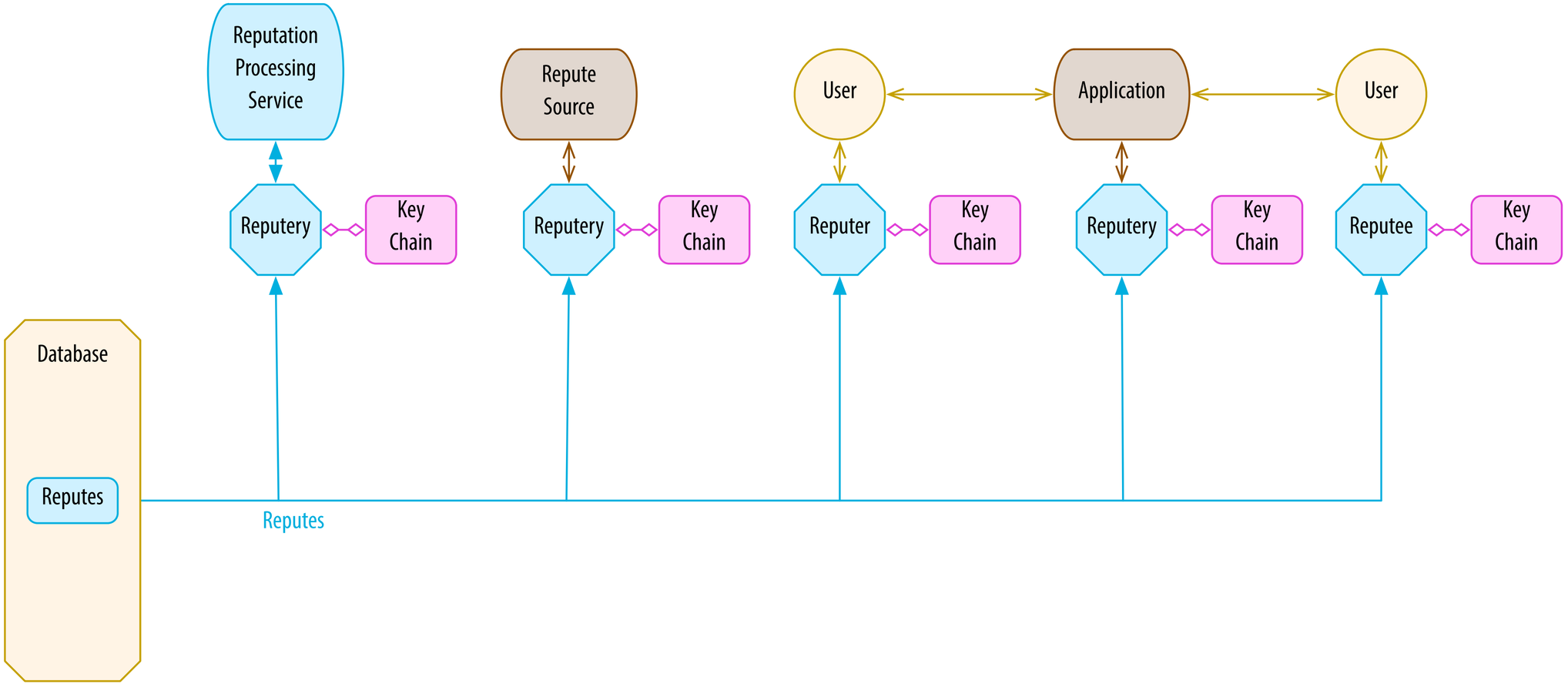






https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/open-reputation-low-level-whitepaper.pdf

Basic Reputation



Business Model

Reputation as a Service (RAAS)

DAS = Distributed Autonomic Service:

service based on distributed consensus + autonomic computing algorithms on decentralized computing infrastructure = *distributed AI*

RAAS on a DAS

Openness & Portability

Open Data Formats

Open Frameworks

Open Algorithms

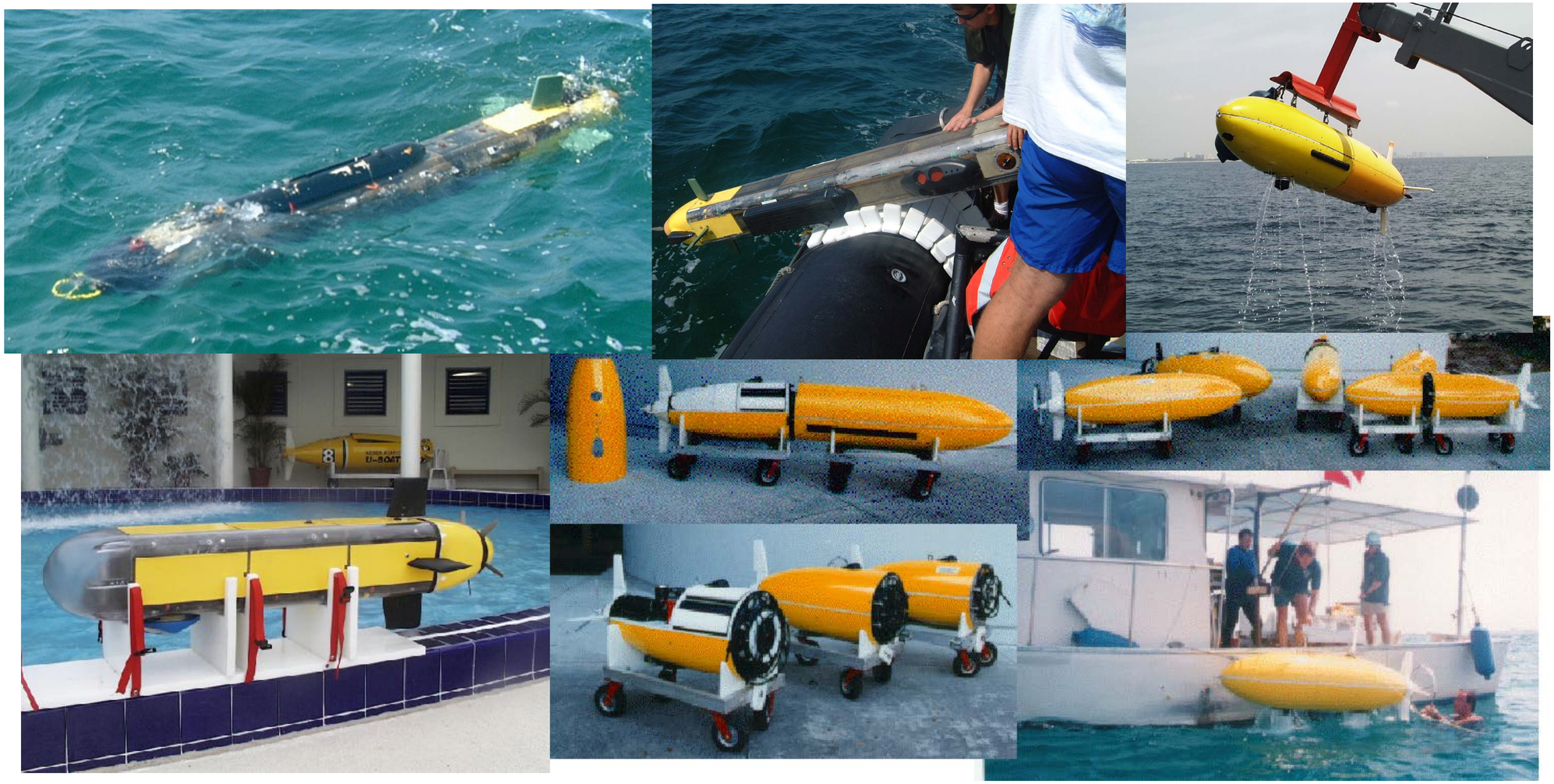
Open Governance

Proprietary Tuning, Parametrization, and Contextualization

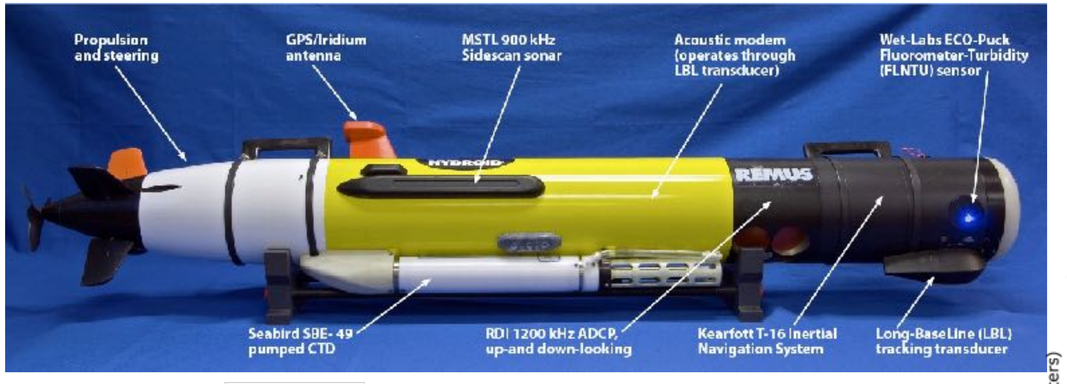
Algorithm

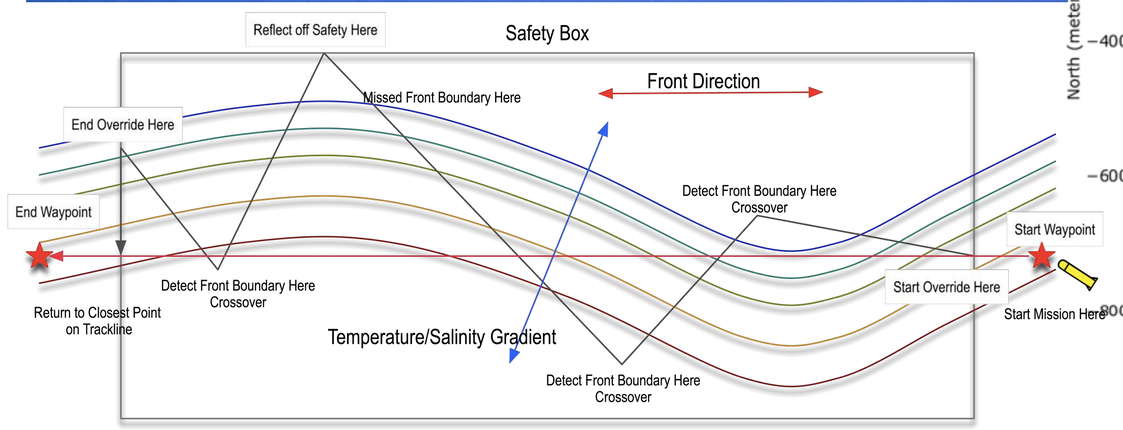
Elastic Constraint Networks with Reinforcement Learning

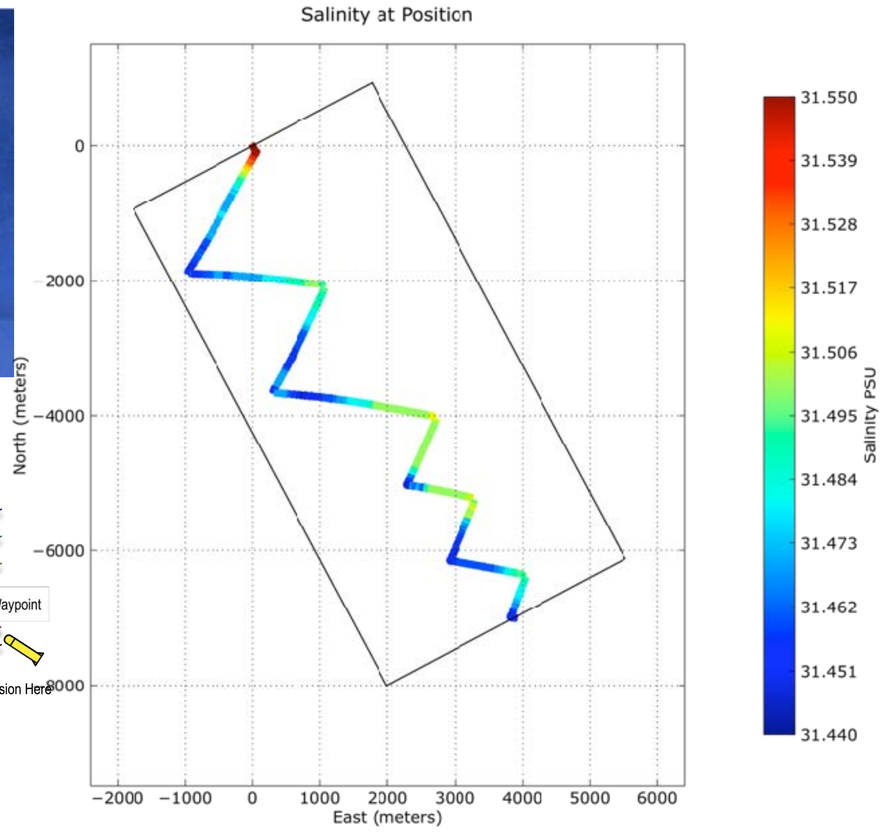
Autonomous Underwater Vehicles



Autonomous Underwater Vehicles







Shipboard & Building Automation

Measurement Theory

Scales

Nominal (Labeled but not ordered)

aggregation operations: none

Ordinal (Labeled and Ordered)

aggregation operations: median, mode

Interval (Ordered with Distance)

aggregation operations: median, mode, mean and moments about mean, std)

Ratio (Distance with True Zero)

aggregation operations: all

Uncertainty

Imprecision (Possibility Measures and operators)

Randomness (Probability Measures and operators)

Ambiguity (Similarity Measures and operators)

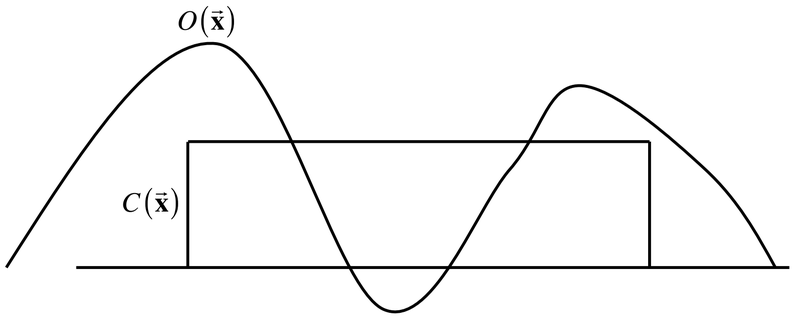
Hybrid (mixtures of imprecision, randomness, and ambiguity)

Symmetric Multiple Elastic Constraints

Conventional

Crisp constraints (intervals) plus objective function(s).

Maximize objective function(s) in area allowed by crisp constraints

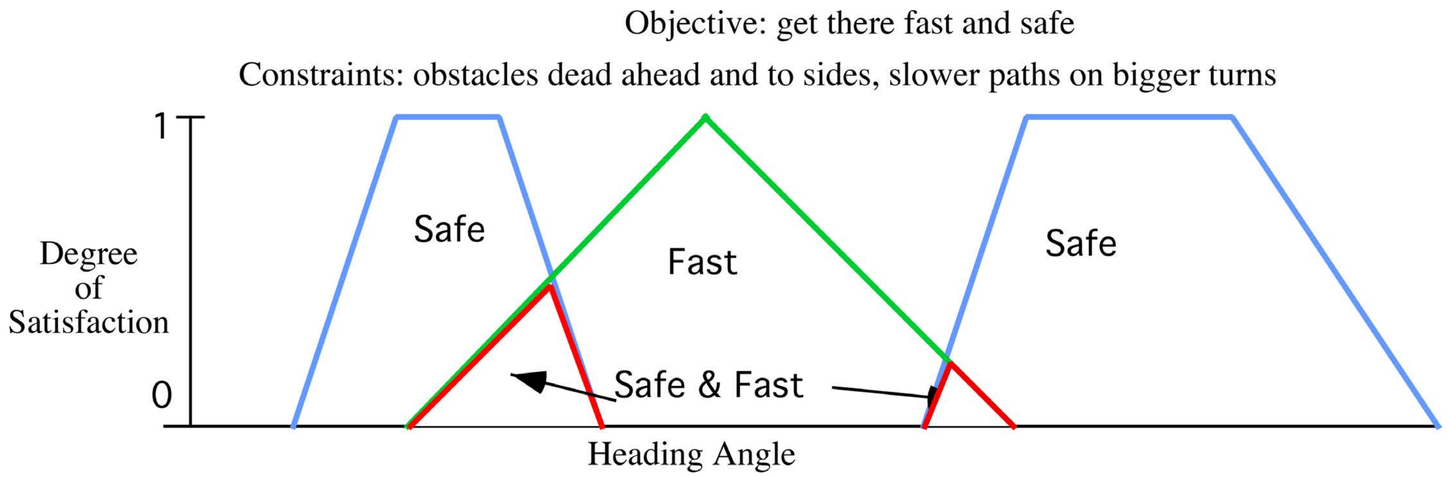


$$O(\vec{\mathbf{x}}) = f(x_1,\ldots,x_n)$$
$$C(\vec{\mathbf{x}}) = \{x_{1low} \le x_1 \le x_{1high},\ldots,x_{nlow} \le x_n \le x_{nhigh}\}$$
$$\max\big|_{C(\vec{\mathbf{x}})} O(\vec{\mathbf{x}})$$

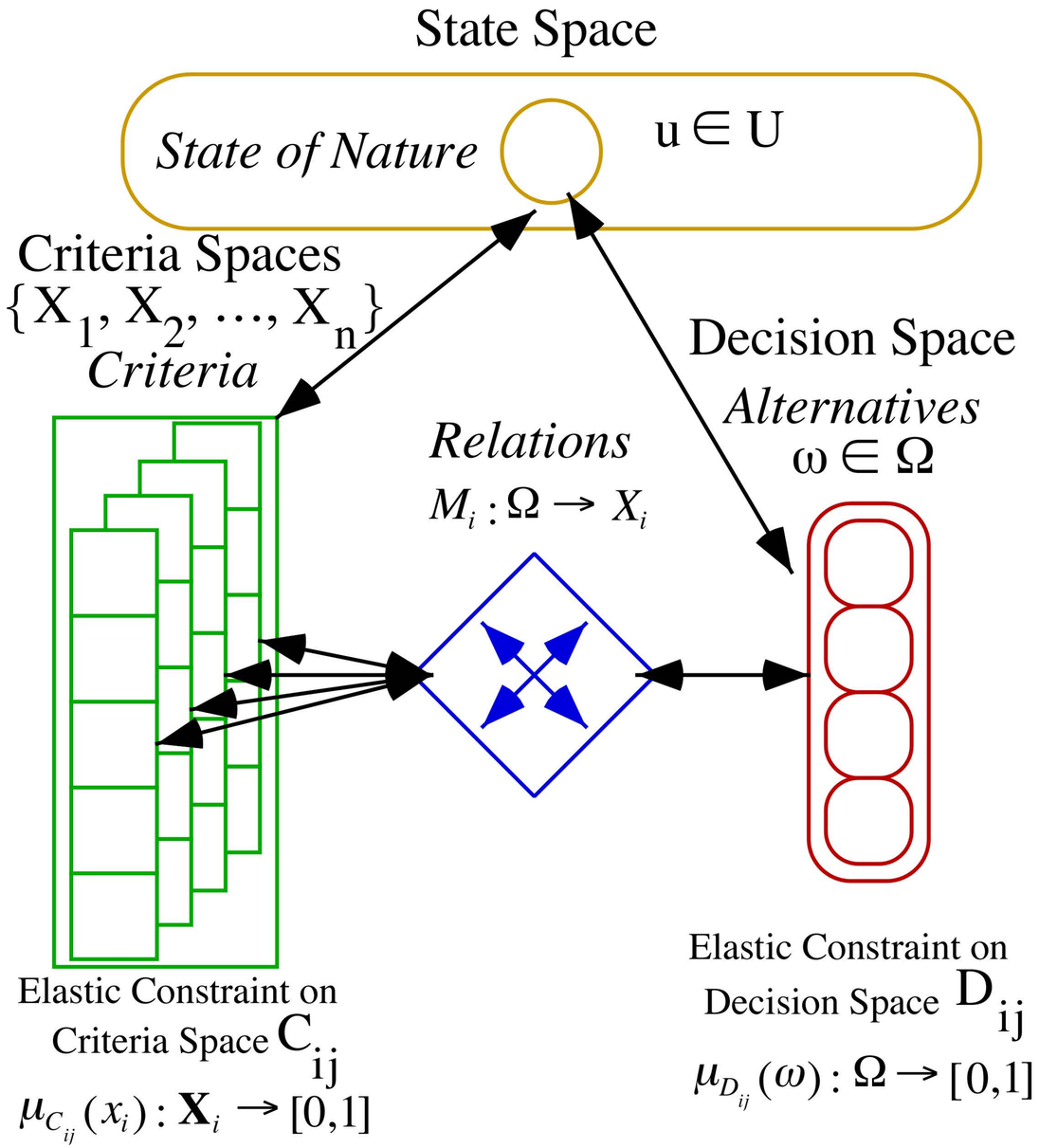
Symmetric

Both goals and constraints are expressed with elastic constraints.

Decision-making consists of finding the confluence of goals and constraints by aggregating the respective membership functions.



SMEC



state of nature $u \in \mathbf{U}$ state space

decision alternative $\omega \in \Omega$ decison space

set of criteria $\mathbf{C}$ defined on $\mathbf{X} = \{X_1, \ldots, X_m\}$ criteria spaces

elastic constraint C_{ij} = j^{th} constraint on i^{th} criteria space

membership function $\mu_{C_{ij}}(x_i) : \mathbf{X}_i \rightarrow [0,1]$

$M_i : \Omega \rightarrow X_i$ relation between decision space and i^{th} criteria space

C_{ij} induces a constraint D_{ij} on Ω through M

If $M_i \equiv m_i(\omega) \quad s.t. \quad x_i = m_i(\omega)$ Then

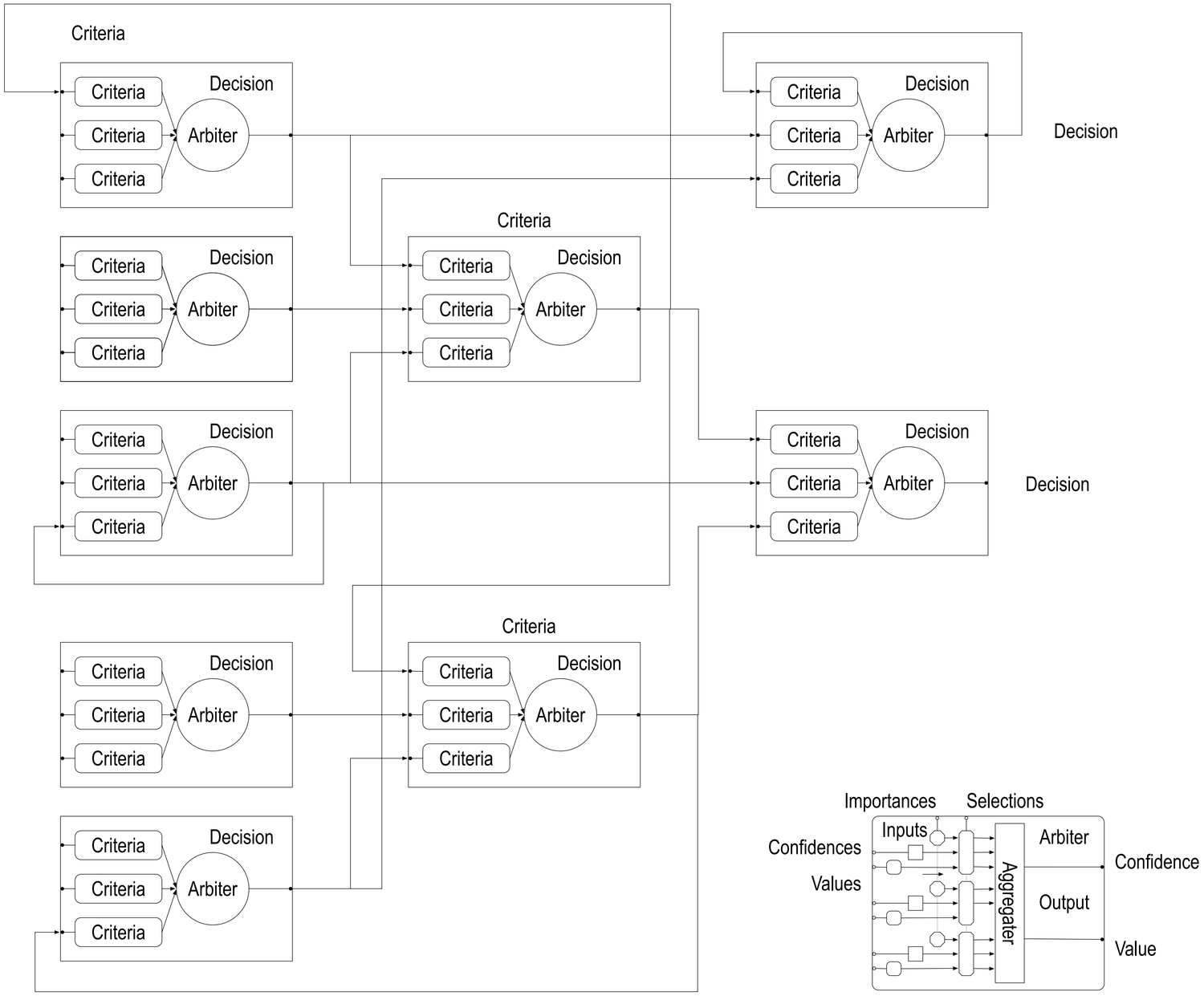
$$\mu_{D_{ij}}(\omega) = \mu_{C_{ij}}(x_i) = \mu_{C_{ij}}(m_i(\omega)) \quad \forall \omega \in \Omega$$

$$\mu_{D_{ij}}(\omega) : \Omega \rightarrow [0,1] \quad \mu_{C_{ij}}(x_i) : \mathbf{X}_i \rightarrow [0,1]$$

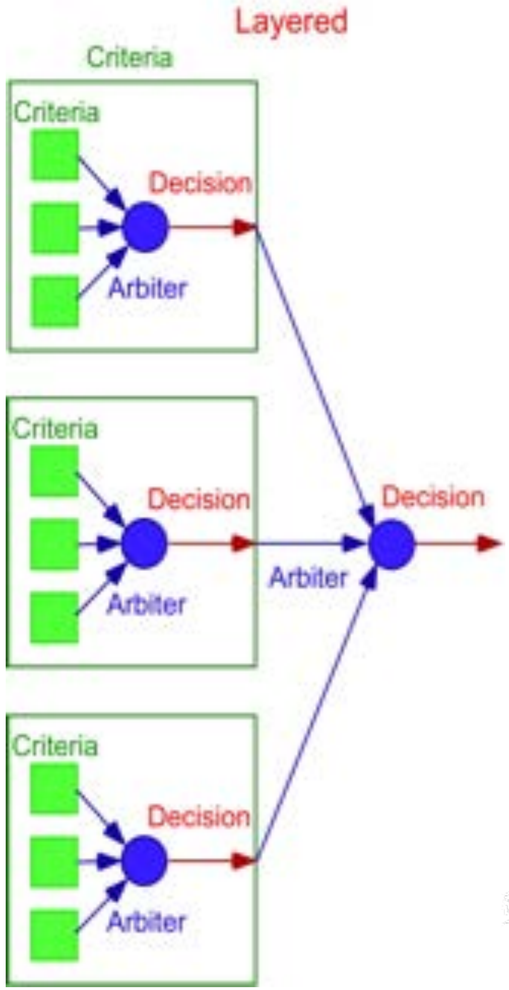
Final decision by aggregating constraints

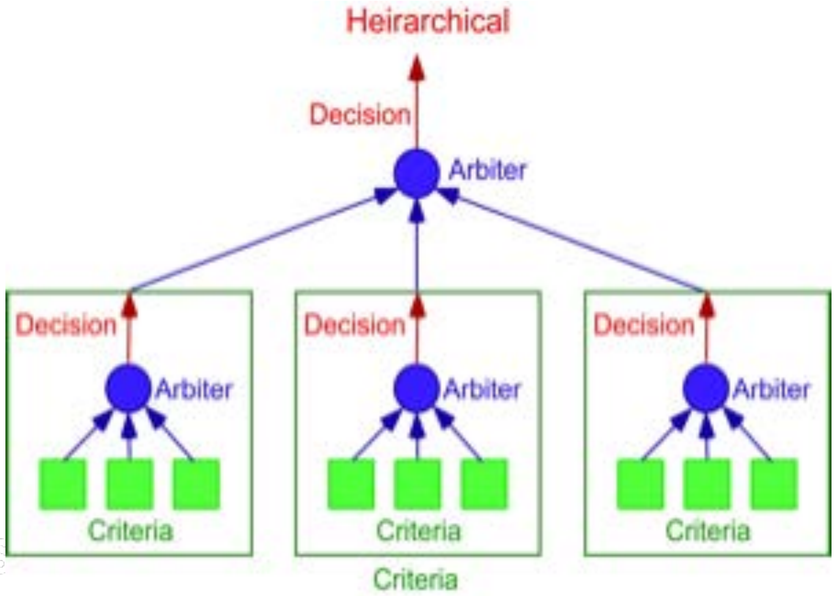
$$\mu_D(\omega) = Agg(\mu_{D_{11}}(\omega), \mu_{D_{12}}(\omega), \ldots, \mu_{D_{21}}(\omega), \ldots, \mu_{D_{mn}}(\omega))$$

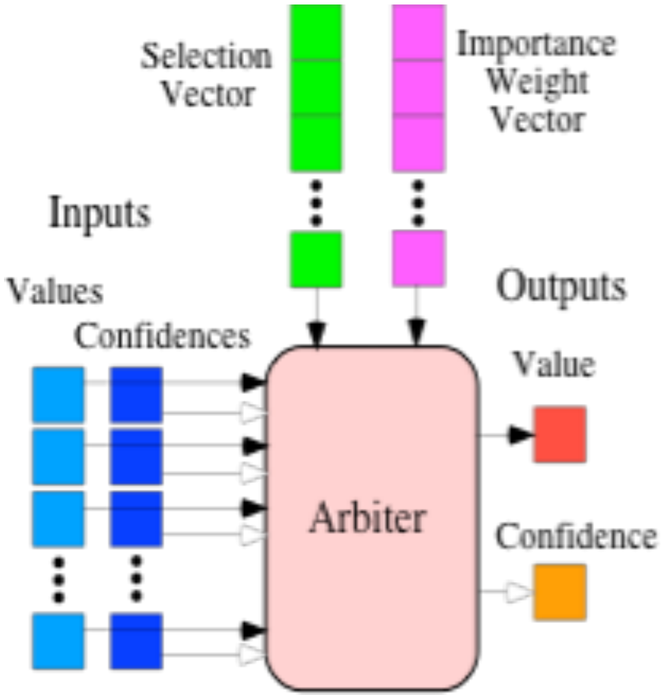
Symmetry Enables Networks of Elastic Constraints



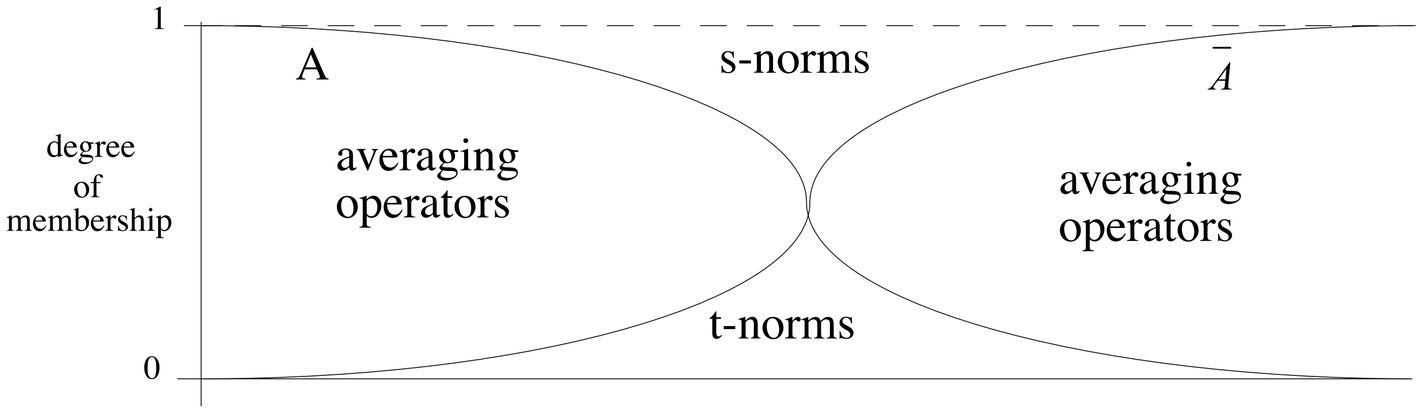
Elastic Constraint Network Building Blocks







Aggregation Operators



t-norms and s-norms,

weighted t-norms and s-norms

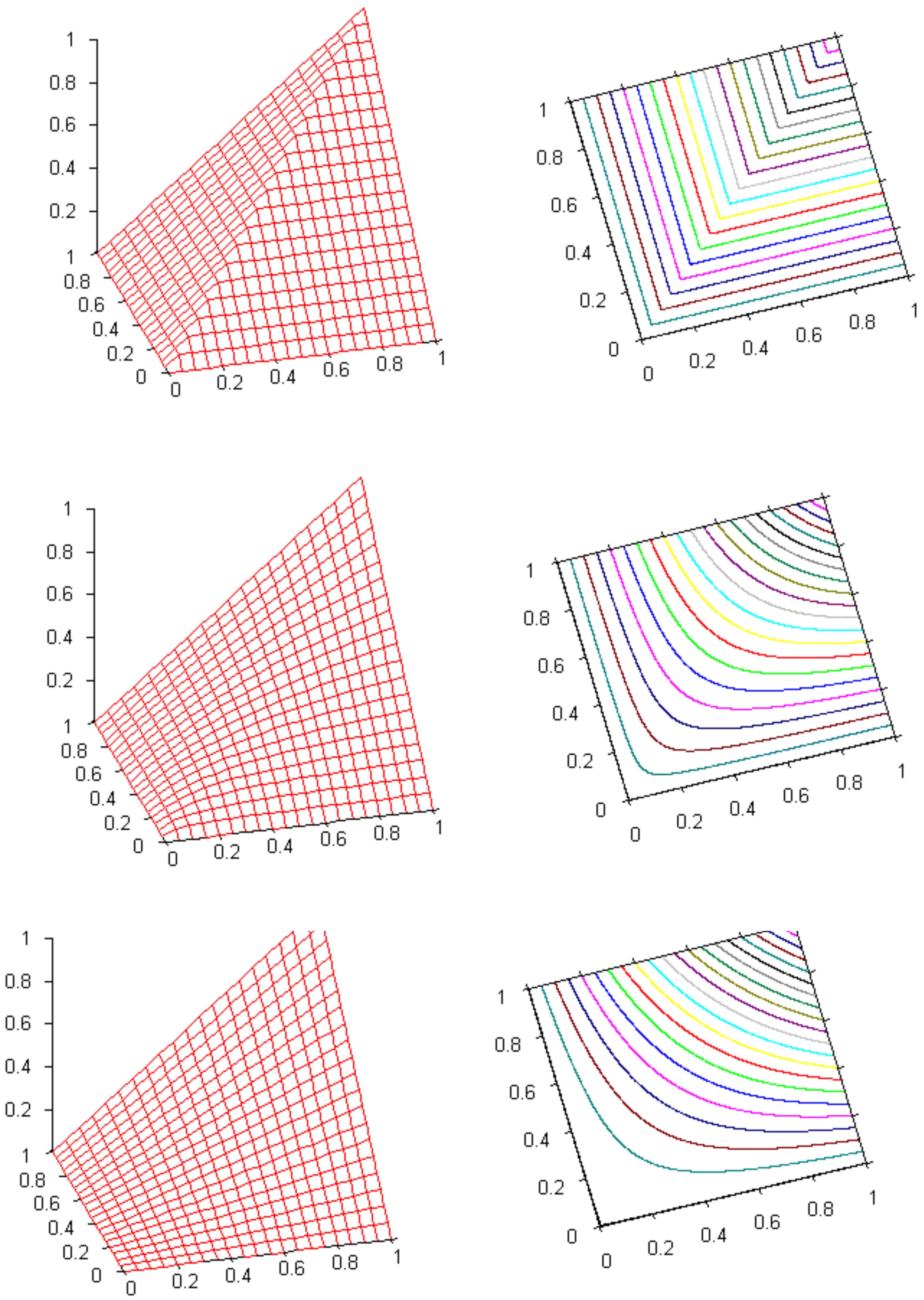
averaging operators

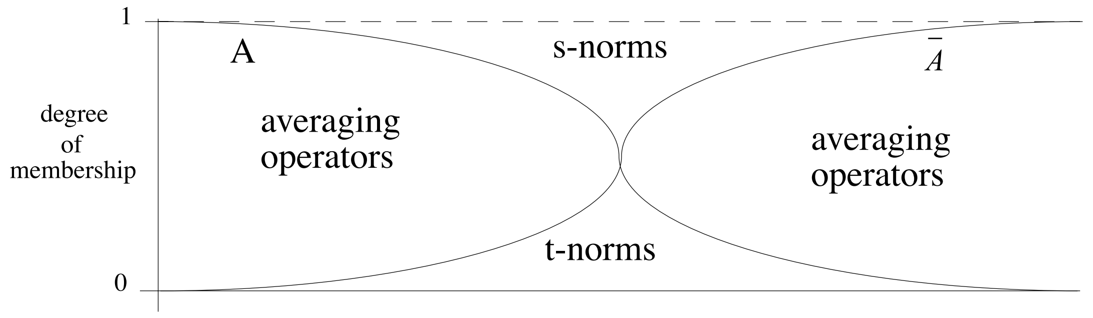
weighted aver- aging operators

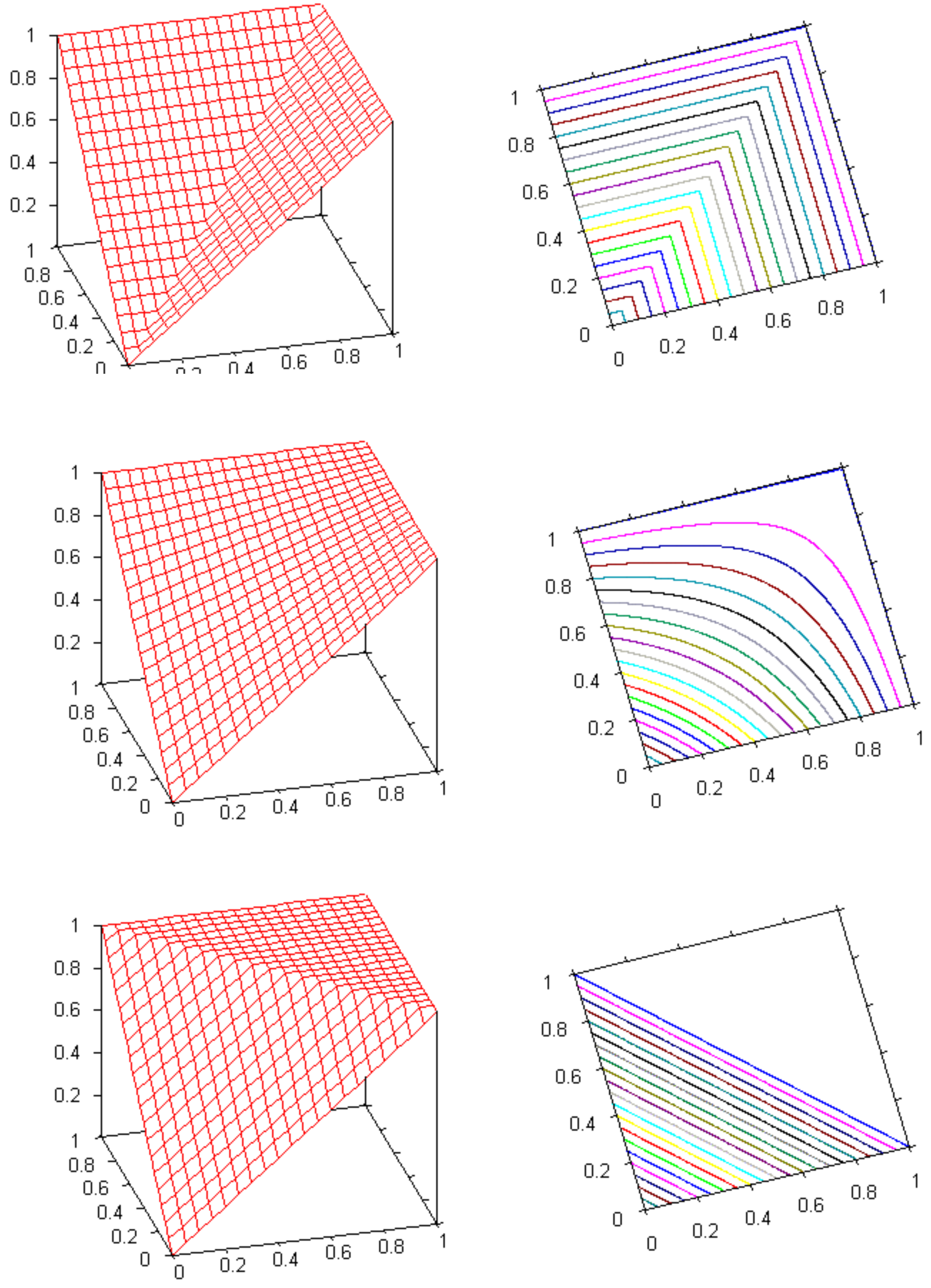
ordered weighted averaging operators (OWA)

hybrid operators.

Aggregation Operators

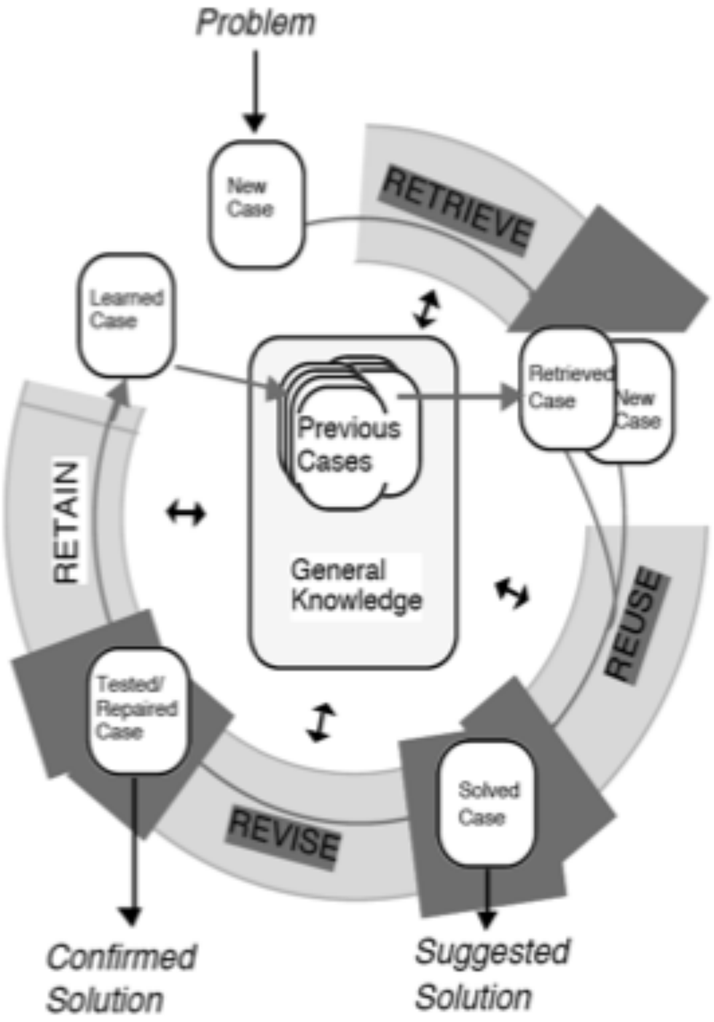


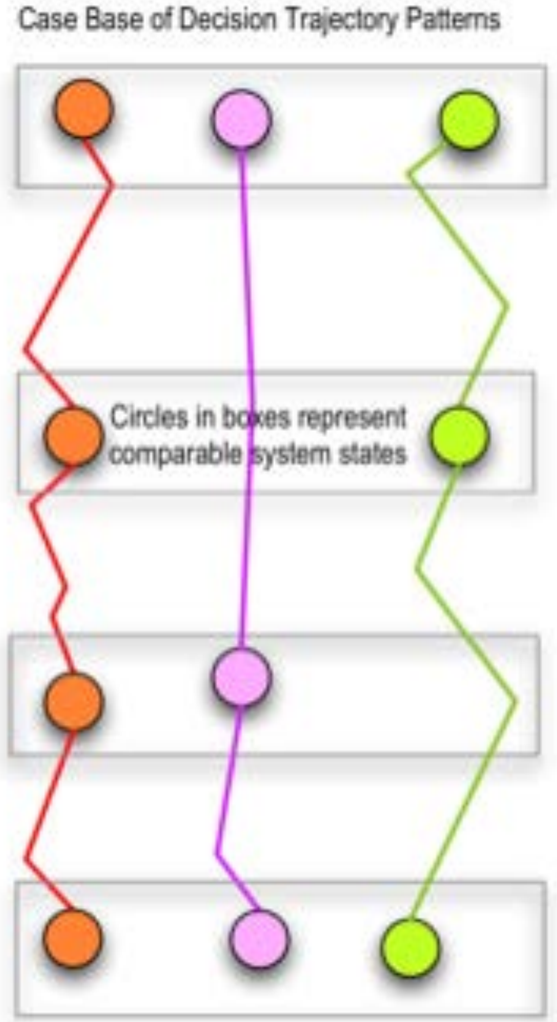


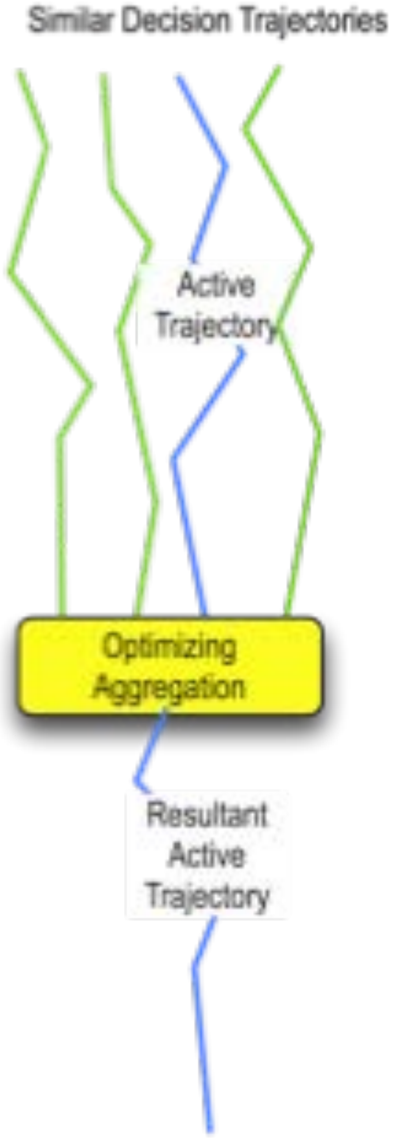


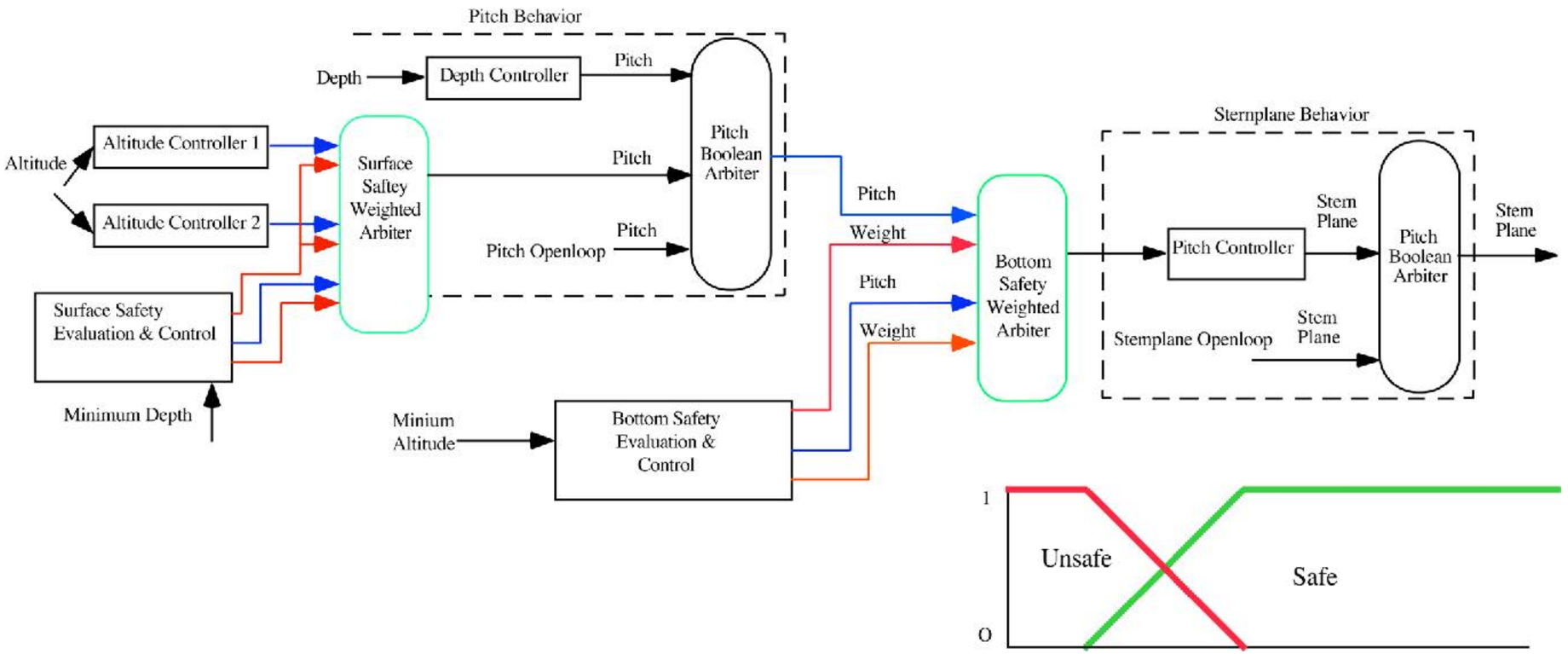
Decision Trajectory Based Optimization

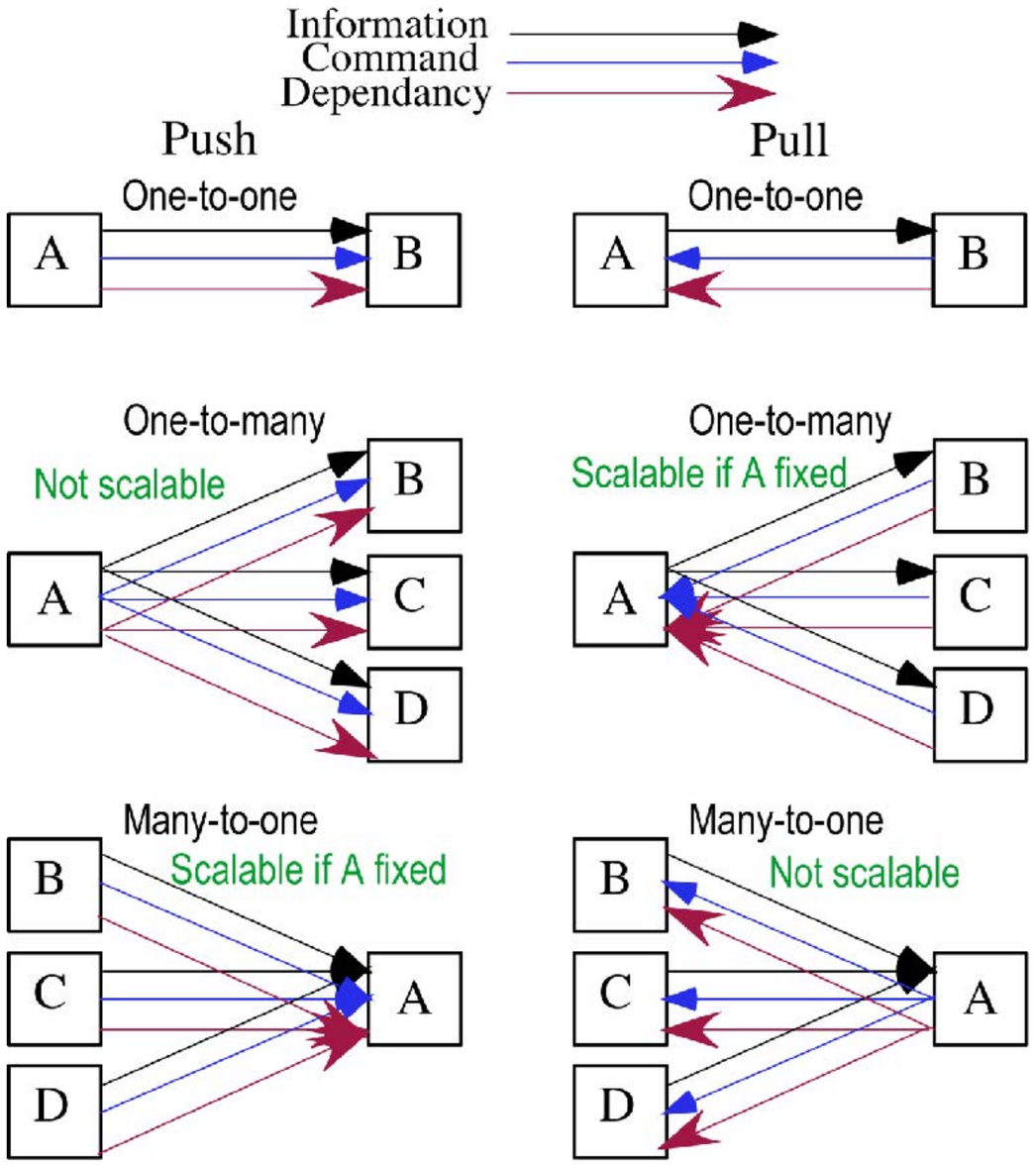
Predictive discrete event simulation with reinforcement learning to encode high level reasoning, strategy and tactics.

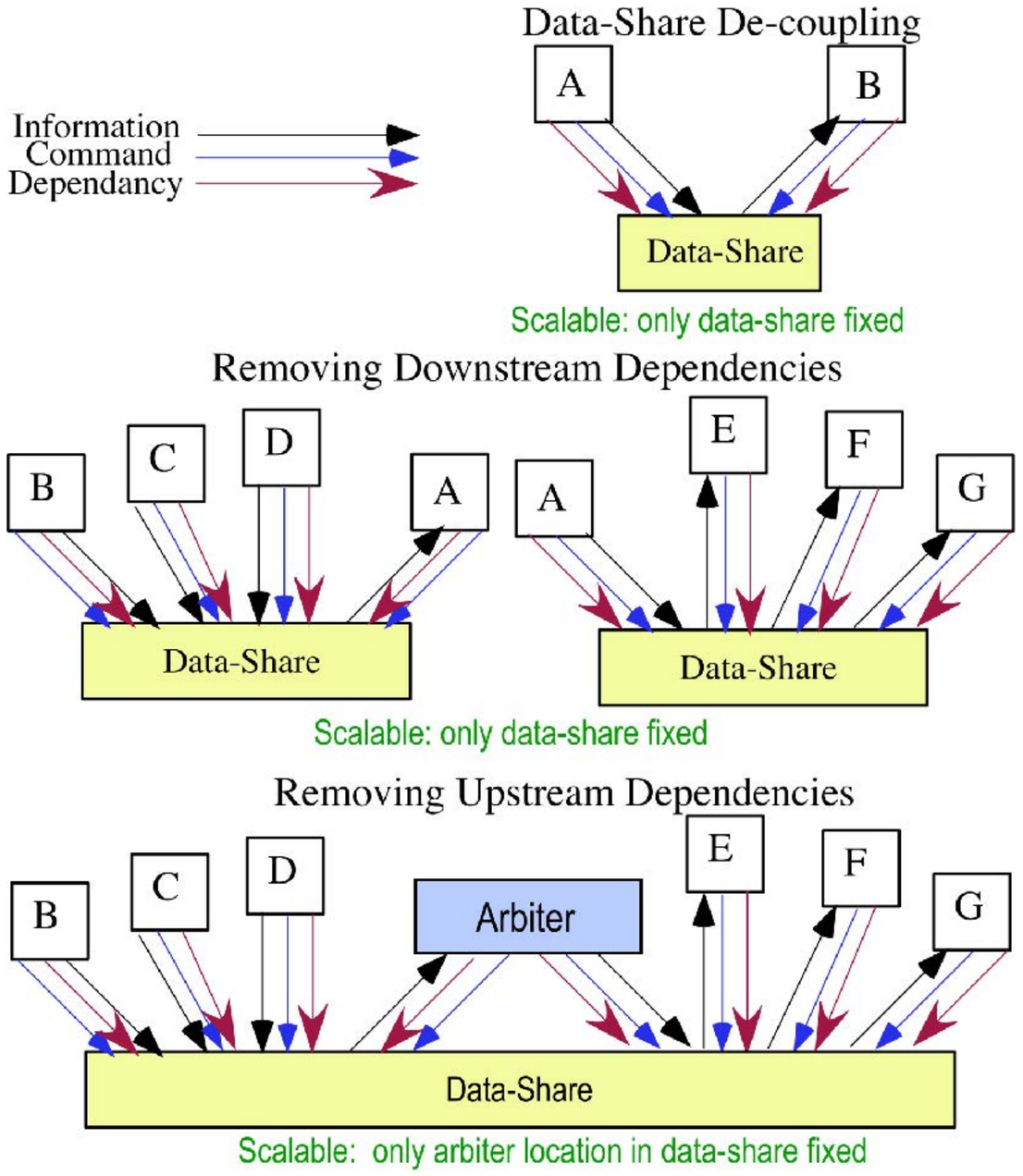







Pitch Behavior
Depth
Depth Controller
Pitch
Pitch Boolean Arbiter
Altitude
Altitude Controller 1
Altitude Controller 2
Surface Saftey Weighted Arbiter
Pitch
Pitch Openloop
Pitch
Surface Safety Evaluation & Control
Minimum Depth
Minium Altitude
Bottom Safety Evaluation & Control
Pitch
Weight
Pitch
Weight
Bottom Safety Weighted Arbiter
Sternplane Behavior
Pitch Controller
Stem Plane
Pitch Boolean Arbiter
Stem Plane
Stemplane Openloop
Stem Plane
1
Unsafe
Safe
0


Information
Command
Dependancy
Push
Pull
One-to-one
A
B
One-to-one
A
B
One-to-many
Not scalable
A
B
C
D
One-to-many
Scalable if A fixed
A
B
C
D
Many-to-one
Scalable if A fixed
B
C
D
A
Many-to-one
Not scalable
B
C
D
A


Data-Share De-coupling
Information
Command
Dependancy
A
B
Data-Share
Scalable: only data-share fixed
Removing Downstream Dependencies
B
C
D
A
Data-Share
A
E
F
G
Data-Share
Scalable: only data-share fixed
Removing Upstream Dependencies
B
C
D
Arbiter
E
F
G
Data-Share
Scalable: only arbiter location in data-share fixed

Complexity Management

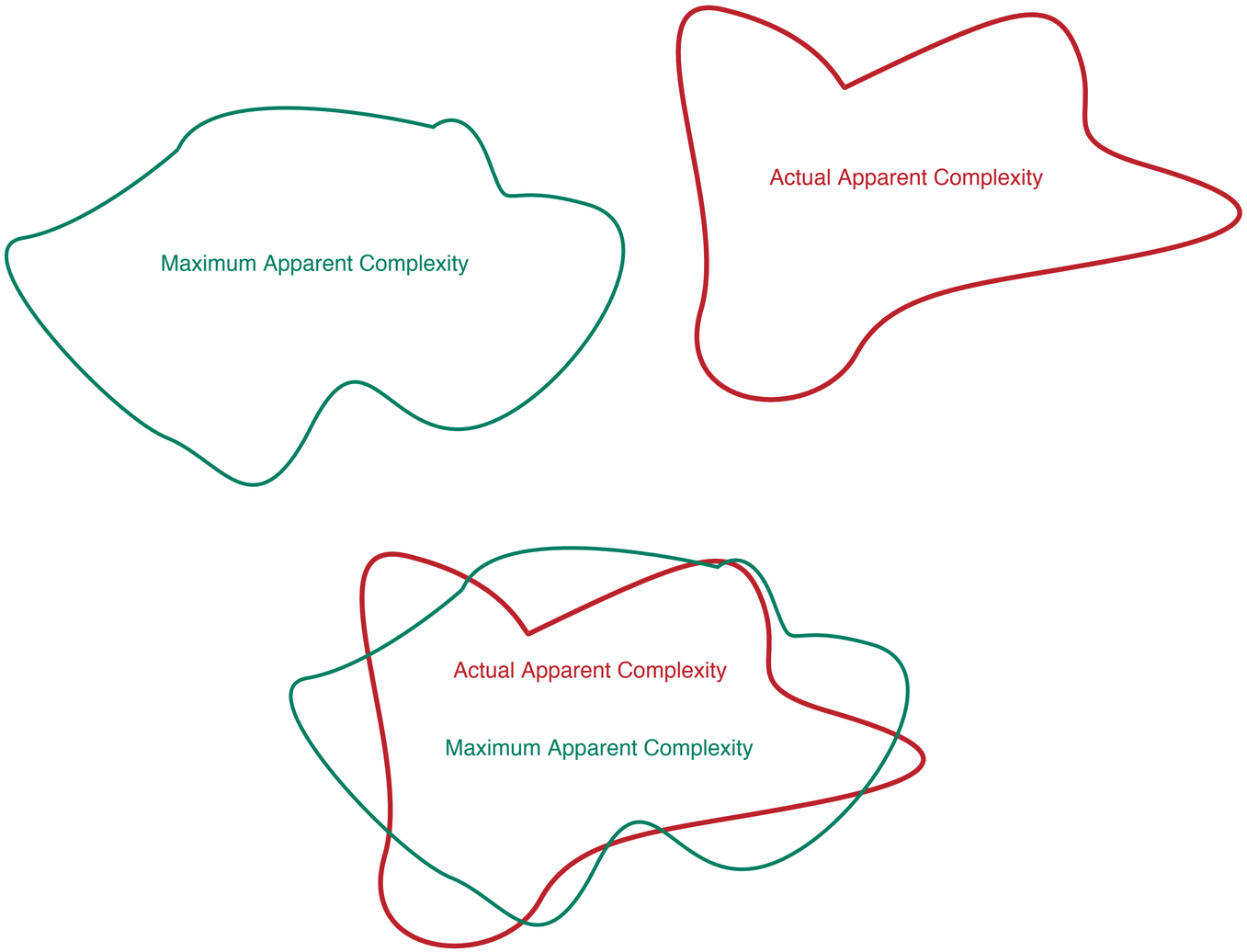
Real Complexity = number of dependencies between elements of a software system

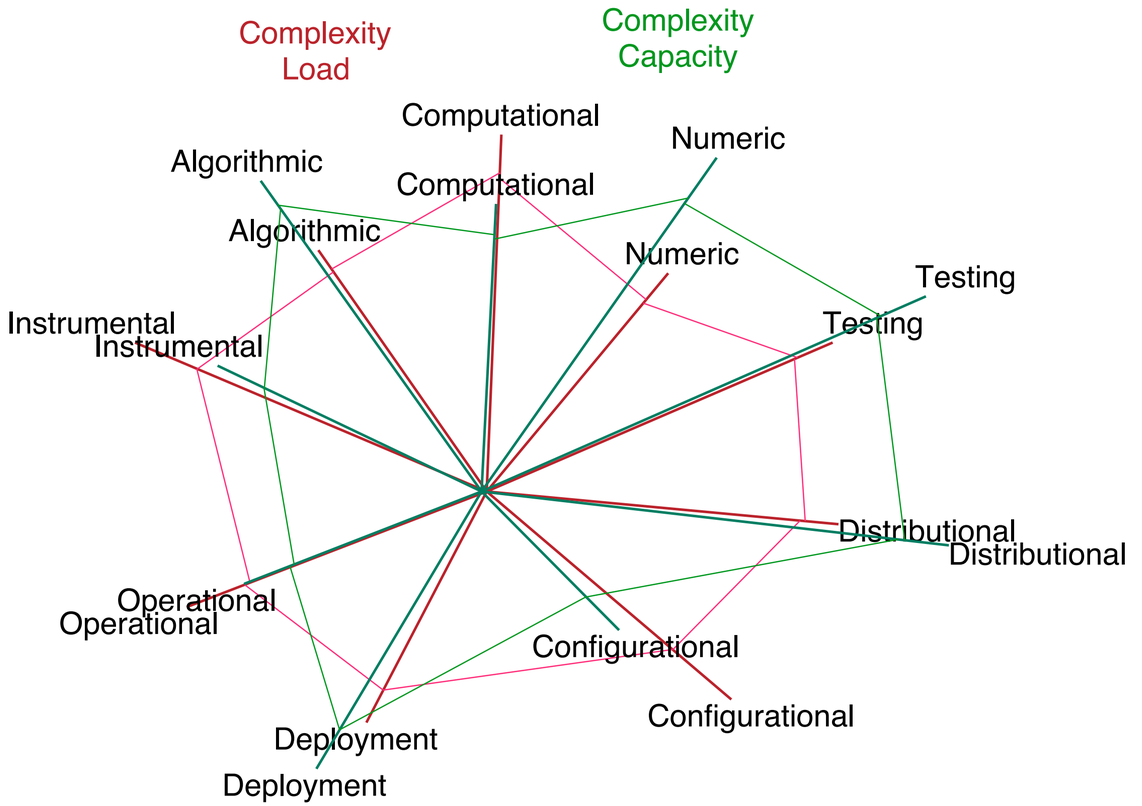
Apparent Complexity = number of dependencies that programmers must manage in order to make meaningful enhancements to software functionality

Perceived Risk = peril the programmer faces when attempting to add meaningful enhancements to software functionality.

A given architecture/development team has a given maximal apparent complexity capacity.

When the actual apparent complexity starts to reach the maximal capacity, development becomes painful.





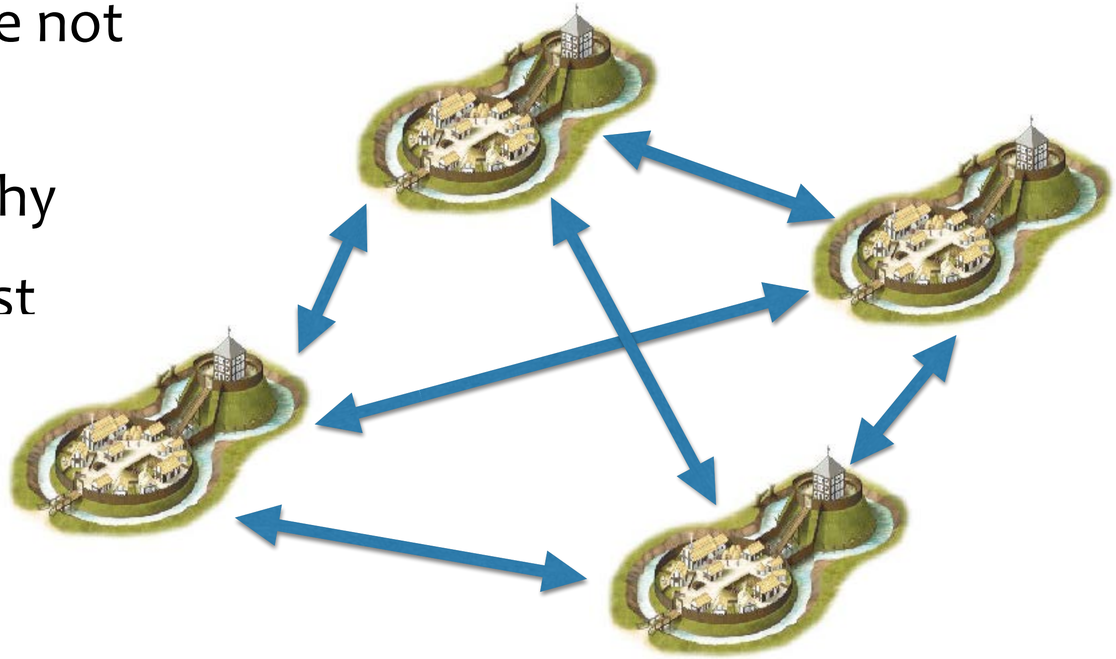
Diffuse Trust

- Cellular distributed topology.

  "Think clandestine spy ring or resistance organization"

- Defeating each node (cell, element) requires an independent exploit.
  - No single point of failure. *"Universal root privileges"*
  - No common mode failures. *"Exploit to attain root privileges"*
- As long as the majority or super majority of nodes are not exploited the system is trustworthy.
- Cooperation between peers vs. authoritarian hierarchy
- Distributed consensus is a way to achieve diffuse trust

VS



Why Distributed Consensus

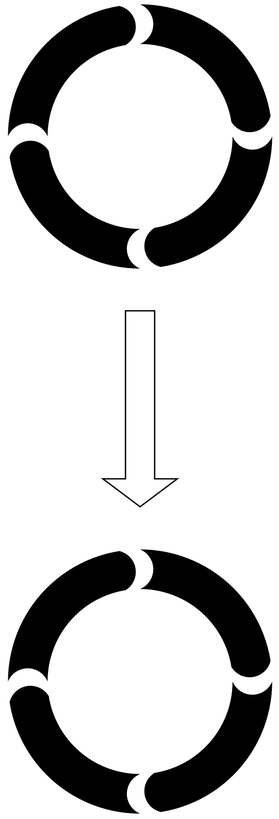
- Enables diffuse trust systems using de-centralized computing infrastructure
- Allows secure infrastructure outside a firewall by distributing the attack surface.
- Enables the replacement of strongly controlled computing infrastructure with weakly controlled computing infrastructure that may be more secure.
- Enables computation in the "*fog*" vs computation in the "*cloud*".
- Enables new disruptive business models

Distributed Consensus Applications

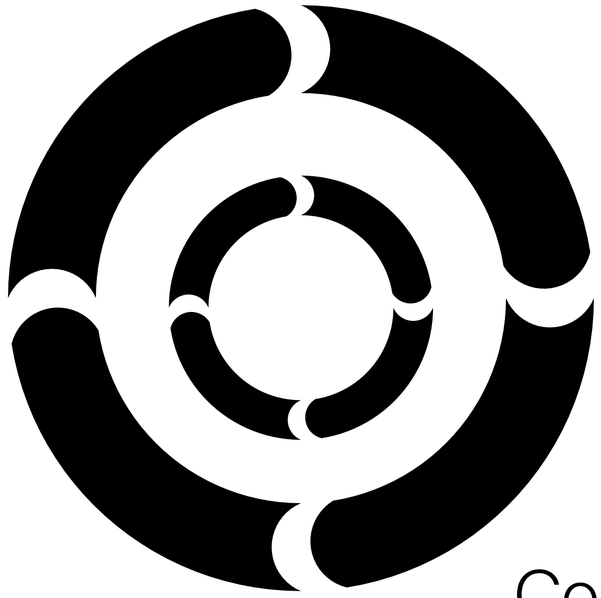
- Settlement, and other FinTech
- Exchanges
- IoT
- Neighborhood computing
- Open Platforms
- Smart Contracts
- Distributed AI

Real persistently automated systems can have complex feedback loop topologies

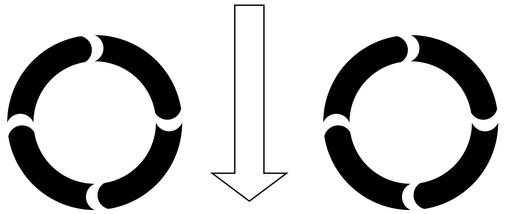
Sequenced loops



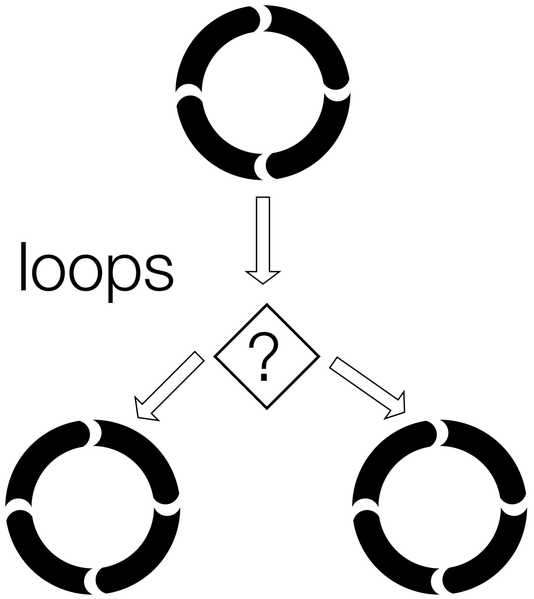
Loops within loops



Concurrent loops



Conditional loops



What Is Reputation

Contextual Behavior Based Predictor of Future Behavior

Measurement

Scoring, Rating and Ranking

Static vs Dynamic

Feedback effects

What Is Self-Sovereign Data

User Controlled and Managed

Portable Identifiers & Attributes

Decentralized (not in a silo)