charles river analytics

Probabilistic Programming: Past, Present, and Future

Keynote Presentation at the 22nd International Conference on Information Fusion (Fusion 2019), Ottawa, Canada (July 2019)

Avi Pfeffer, PhD

July 4, 2019

BLUF: Probabilistic Programming for Fusion

- Fusion systems take sensor and data inputs and perform useful reasoning with them
  - Predict future events
  - Infer current situation that led to observations
  - Learn how to predict and infer better
- Probabilistic reasoning can do all these things
- But with difficulty for all but the simplest models

Probabilistic programming makes it possible to develop probabilistic applications with much less effort and expertise

(for fusion applications and beyond)

Keynote Presentation at the 22nd International Conference on Information Fusion (Fusion 2019), Ottawa, Canada (July 2019)

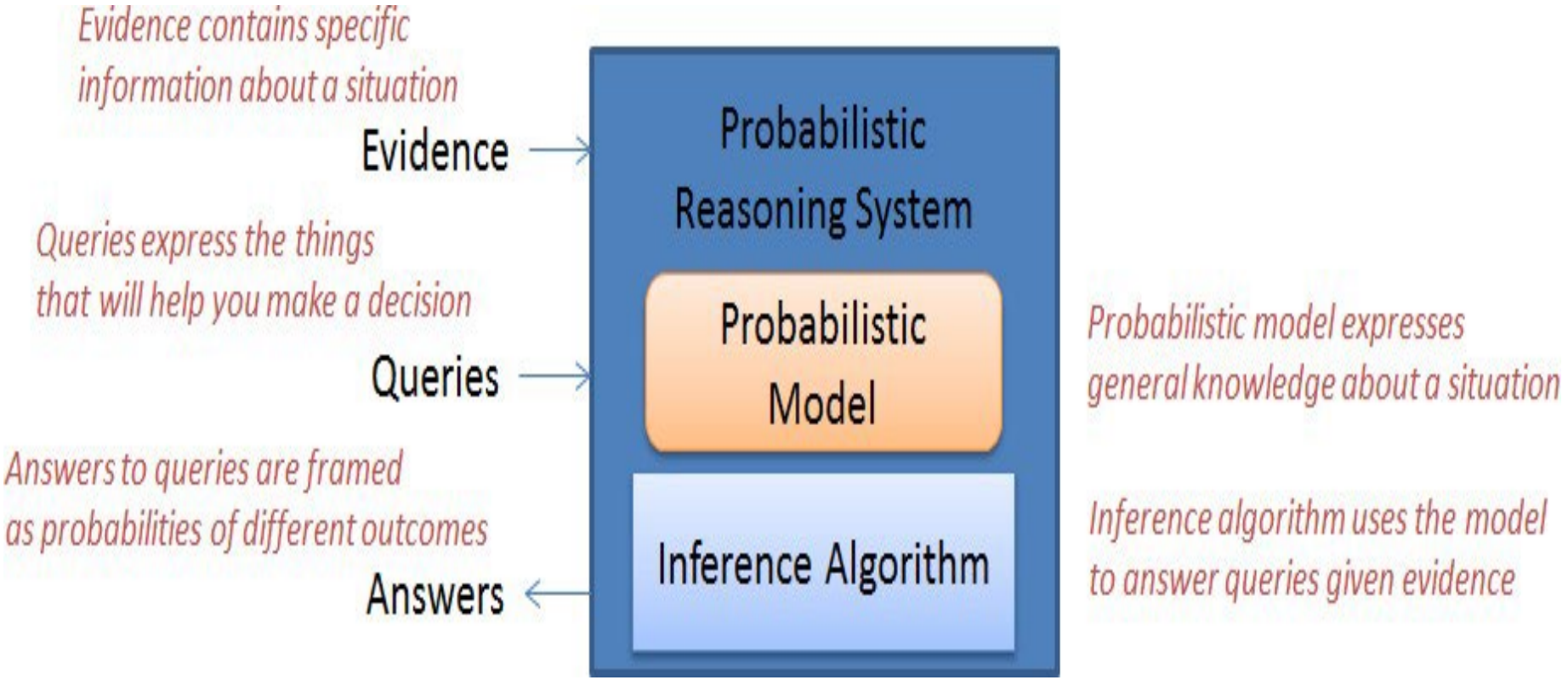
Overview

1) What is Probabilistic Programming?
2) Probabilistic Programming in Action
3) Probabilistic Programming Inference Algorithms
4) Probabilistic Programming for Long-Lived AI Systems

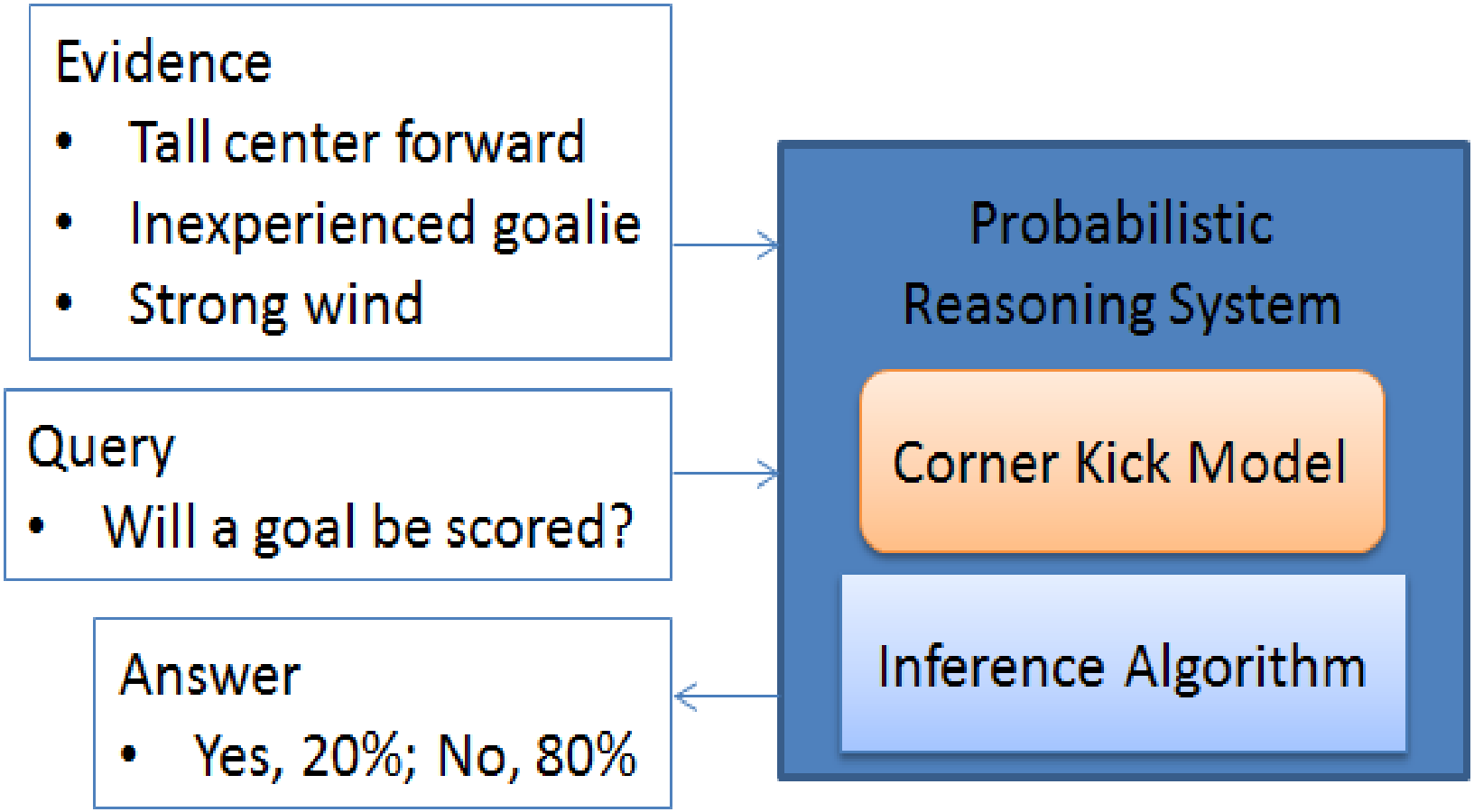
01

What is Probabilistic Programming?

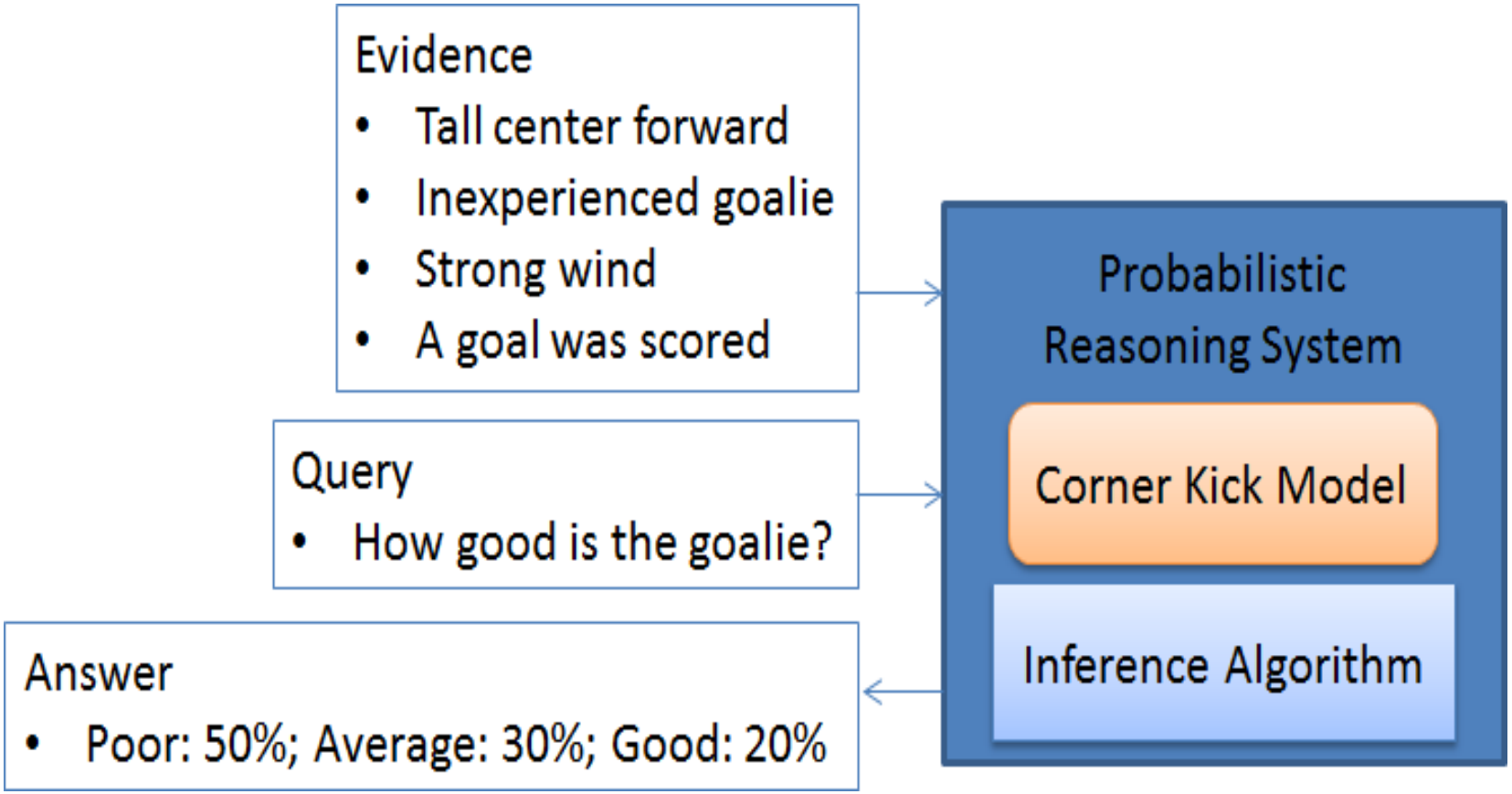
The Gist of Probabilistic Reasoning



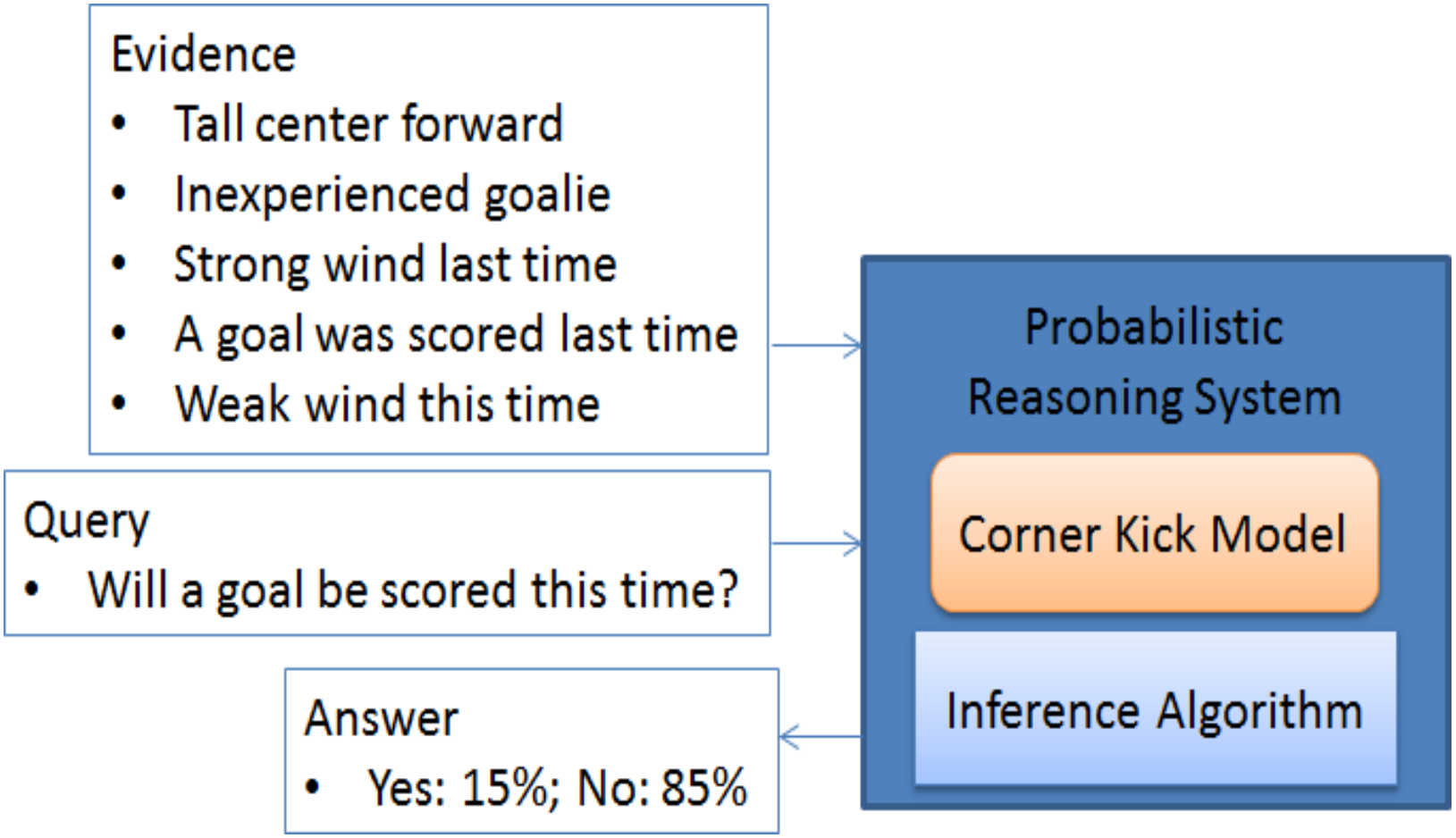
Probabilistic Reasoning: Predicting the Future



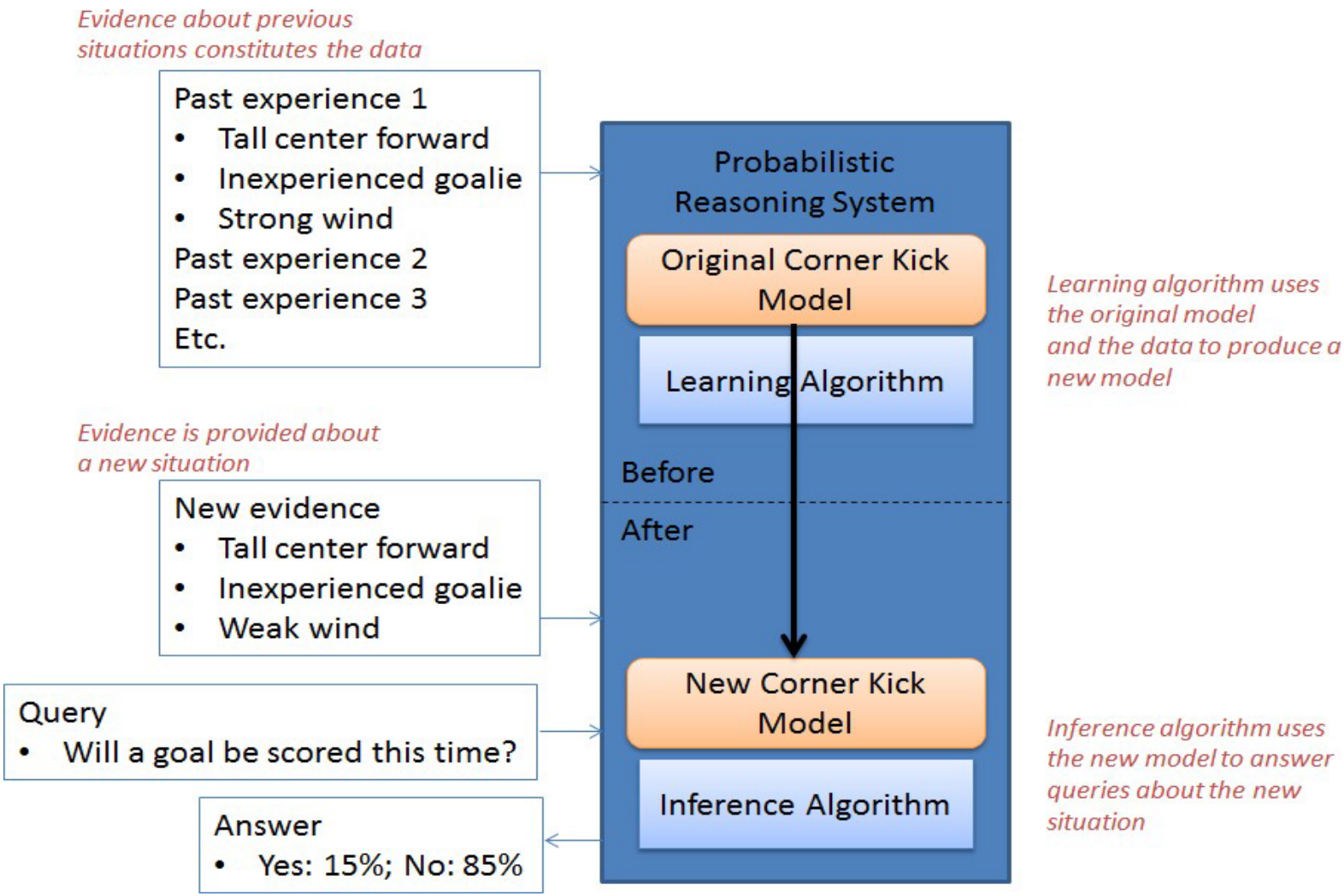
Probabilistic Reasoning: Inferring Factors that Caused Observations



Probabilistic Reasoning: Using the Past to Predict the Future



Probabilistic Reasoning: Learning from the Past



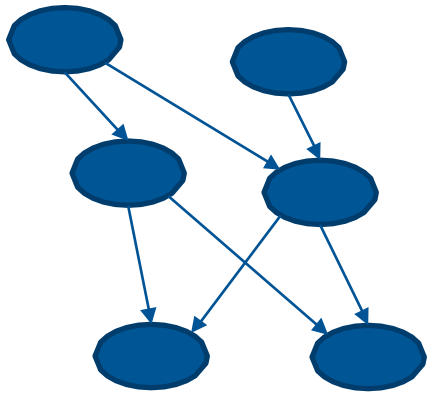
But Probabilistic Reasoning Is Hard!

You need to

- Implement the representation
- Implement the probabilistic inference algorithm
- Implement the learning algorithm
- Interact with data
- Integrate with an application

One Approach: Bayesian Networks

- Implement the representation
  - Bayesian networks
- Implement the probabilistic inference algorithm
  - Standard BN inference algorithms
- Implement the learning algorithm
  - Standard BN learning algorithms
- Interact with data
  - Use a package that supplies ability to read and store data
- Integrate with an application
  - Use a package's API



Limitations of this Approach

- Bayesian network models are flat and unstructured
- Bayesian networks have a fixed set of variables
- Variables have simple types
- Many applications do not satisfy these limitations
  - Models have natural structure that should be captured
  - Changing number of objects
  - Variables with structured and complex types
    - E.g., sequences, trees, graphs
- Before probabilistic programming, researchers invented representations for each individual application
- We want to make using probabilistic models for more complex applications as easy as Bayesian networks

Goals of Probabilistic Programming

Drastically reduce the work to create probabilistic reasoning applications

Expand the range of probabilistic applications that can be created

How Probabilistic Programming Achieves This

1. Expressive programming language for representing models
2. General-purpose inference and learning algorithms apply to models written in the language

All you have to do is represent the model in code and you automatically get the application

Basic Programming Concept: Functional Programming

- Non-probabilistic functional programming language: an expression describes a computation that produces a value

```
tallCenterForward = true
accurateCross = true
goodHeader = tallCenterForward && accurateCross
goodGoalie = false
goal = goodHeader && !goodGoalie
```

Functional Probabilistic Programming

- Probabilistic Functional programming language: an expression describes a *random* computation that produces a value

```
tallCenterForward = flip(0.3)
accurateCross = flip(0.5)
goodHeader =
   if (tallCenterForward && accurateCross) flip(0.8)
   else flip(0.1)
goodGoalie = flip(0.6)
goal =
   if (goodHeader && !goodGoalie) flip(0.7)
   else flip(0.3)
```

Sampling Semantics

- We imagine running the program many times
- Each run generates a value for each of the variables

Variable	Sample 1	Sample 2	Sample 3	Sample 4
Tall center forward	True	False	False	False
Accurate cross	False	True	True	False
Good header	True	False	False	True
Good goalie	True	True	True	False
Goal	False	False	False	True

- This process defines a joint probability distribution over all the variables

Probabilistic Program Inference Tasks

- Probability computation
  - Given observations about some variables (e.g. tallCenterForward)
  - Compute probability of values of other variables (e.g. Goal)
- Most probable explanation
  - Given observations about some variables
  - Compute most likely state of other variables
- Probability of evidence
  - Given observations about some variables
  - Compute probability of those observations
- Many different inference algorithms used

Probabilistic Programming Languages (PPLs)

- Most PPLs describe sampling process in a similar manner
- Variations:
  - Kinds of variables supported
    - E.g., discrete, continuous, or mixed
  - Kinds of models supported
    - E.g., finite structure vs infinitely recursive
  - Integration with ordinary programming language
    - E.g., library in host language vs separate language
  - Inference tasks supported
  - Algorithms used
  - Programming styles
    - Functional
    - Object-oriented
    - Logic
    - Imperative

02

Probabilistic Programming in Action

My Probabilistic Programming Languages

- 1997: Stochastic Lisp [Koller, McAllester, & Pfeffer 97]
  - First functional PPL and algorithm
  - Mainly theoretical
- 2001-2009: Integrated Bayesian Agent Language (IBAL)
  - First practical functional PPL [Pfeffer 01, 07]
  - Some interesting algorithms
  - But limited in its expressivity, algorithms, and integration
- 2009-2018: Figaro [Pfeffer 12, 16]
  - Object-oriented and functional
  - Highly expressive
  - Many algorithms
  - Easy to integrate with data and applications
  - Implemented as a Scala library
- 2018-: Scruff [Pfeffer & Lynn, 18]
  - Designed for long-lived AI applications

Our Example Program in Figaro

```
val tallCenterForward = Flip(0.3)
val accurateCross = Flip(0.5)
val goodHeader =
   If(tallCenterForward && accurateCross,
      Flip(0.8), Flip(0.1))
val goodGoalie = Flip(0.6)
val goal =
   If(goodHeader && !goodGoalie,
      Flip(0.7), Flip(0.3))

println(VariableElimination.probability(goal, true))
```

Figaro Language Concepts (1)

- Apply
  - Applies a function to a random variable that creates another random variable
    - Apply(Uniform(0, 1), x => x * 2)
  - Example: Centrality of a probabilistic graph

- Chain
  - Creates a new random variable that depends on the value of another random variable
    - Chain(Uniform(0, 1), x => Normal(x, 1))
  - Example: Random walk on a probabilistic graph

Figaro Language Concepts (2)

- Condition
  - Asserts that a variable must have a certain property
    - Uniform(0, 1).addCondition(x => x > 0.5)
    - Flip(0.7).observe(true)
  - Example: Observing that a probabilistic graph has 26 nodes

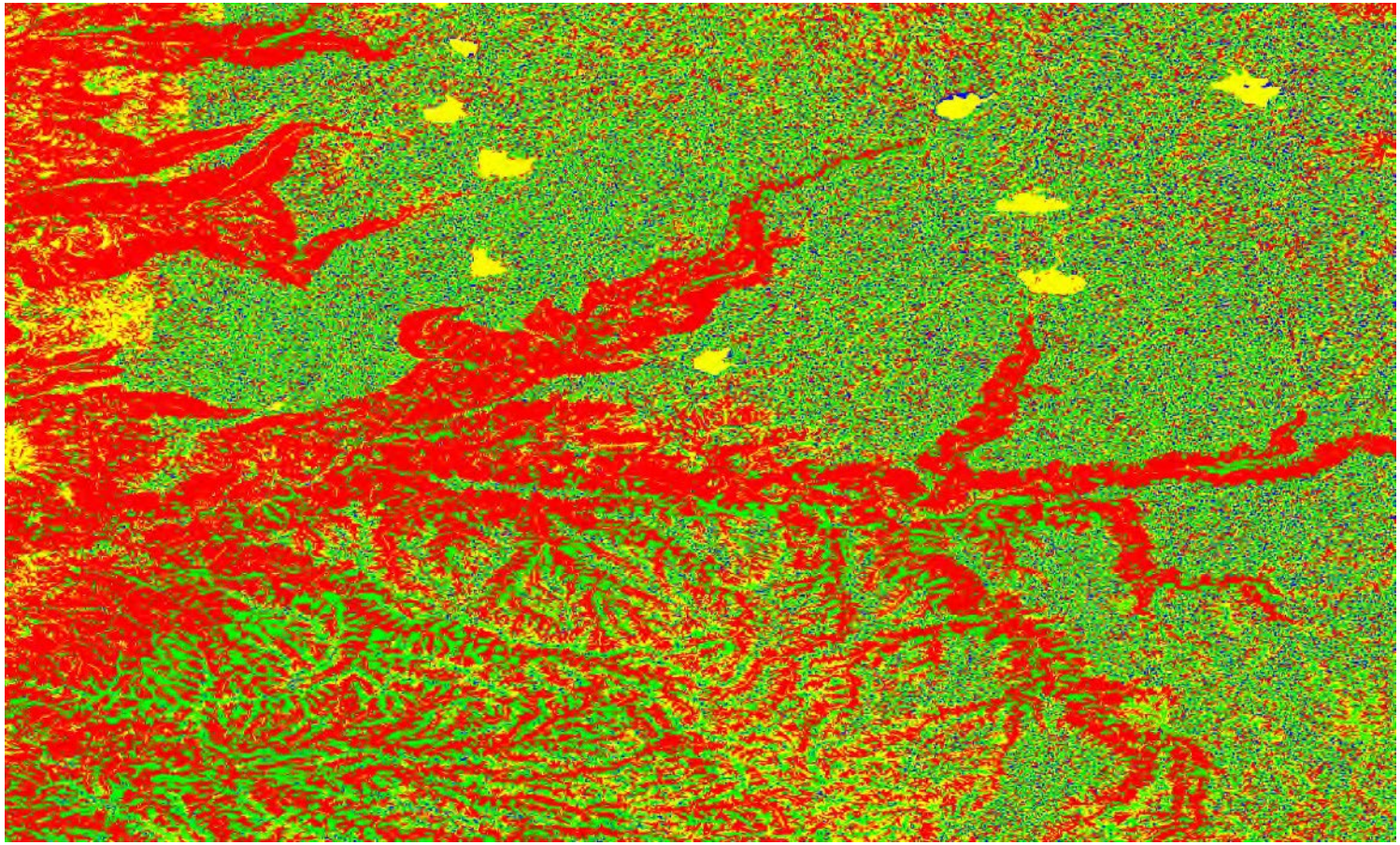
- Constraint
  - Provides a weighting function for the values of a variable
    - Uniform(0, 1).addConstraint(x => x)
  - Example: Asserting that nodes in a probabilistic graph tend to have fewer edges

Figaro Applications

- Over the last 10 years, we've created a large number of applications of Figaro
- I'll show you some representative examples to illustrate the sorts of things you can do with probabilistic programming

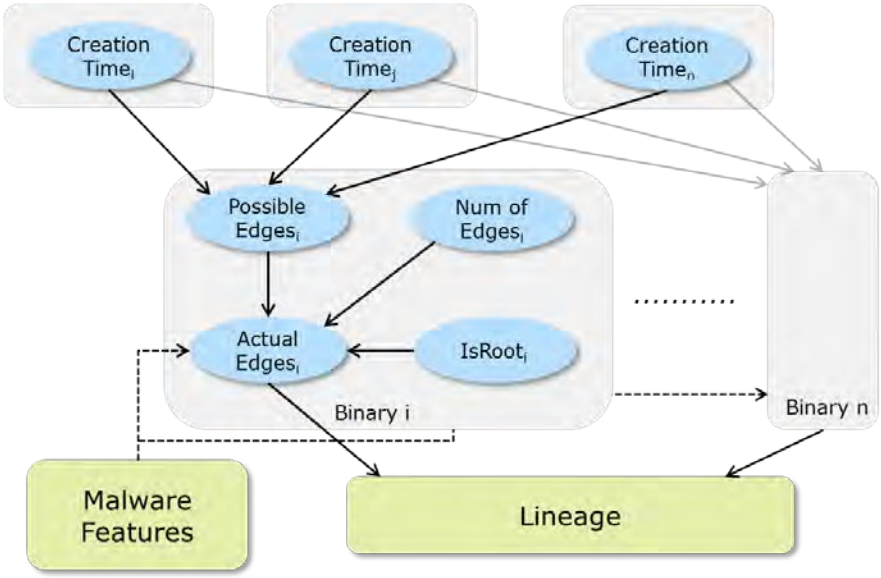
Hydrological Terrain Modeling for Army Logistics (TIDE)

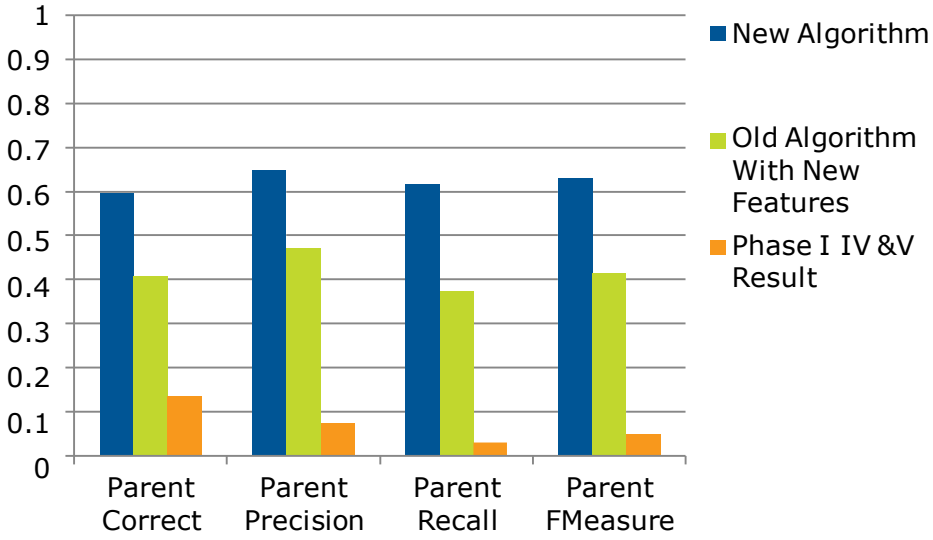
Figaro novices were able to quickly build up an integrated probabilistic reasoning application



Malware Lineage (DARPA Cyber Genome)

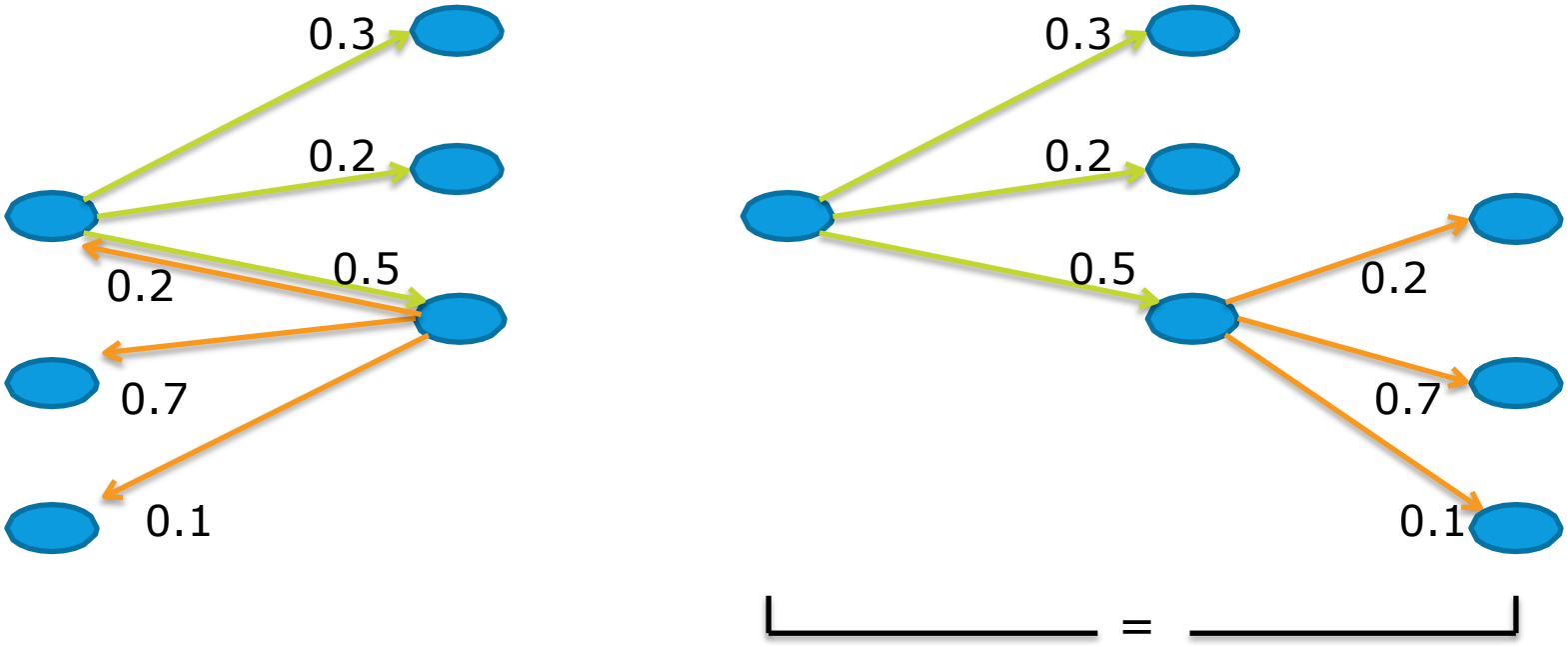
We were able to perform a sophisticated analysis far better than our previous non-probabilistic method





Tracklet Merging (DARPA PPAML Challenge Problem)

We came up with a new algorithm that we would not have thought of without probabilistic programming and expressed it in one slide



Tracklet Merging in Figaro

```
class Tracklet(
  toCandidates: List[(Double, Tracklet)],
  fromCandidates: List[(Double, Tracklet)]
){
  val next = Select(toCandidates: _*)
  val previous = Select(fromCandidates: _*)
}

for (source <- sources) {
  val nextPrevious =
    Chain(source.next,
          nextTracklet => nextTracklet.previous)
  nextPrevious.observe(source)
}
```

Workflow Activity Recognition (DARPA PPAML)

Based on DARPA PAL program, which turned into Siri

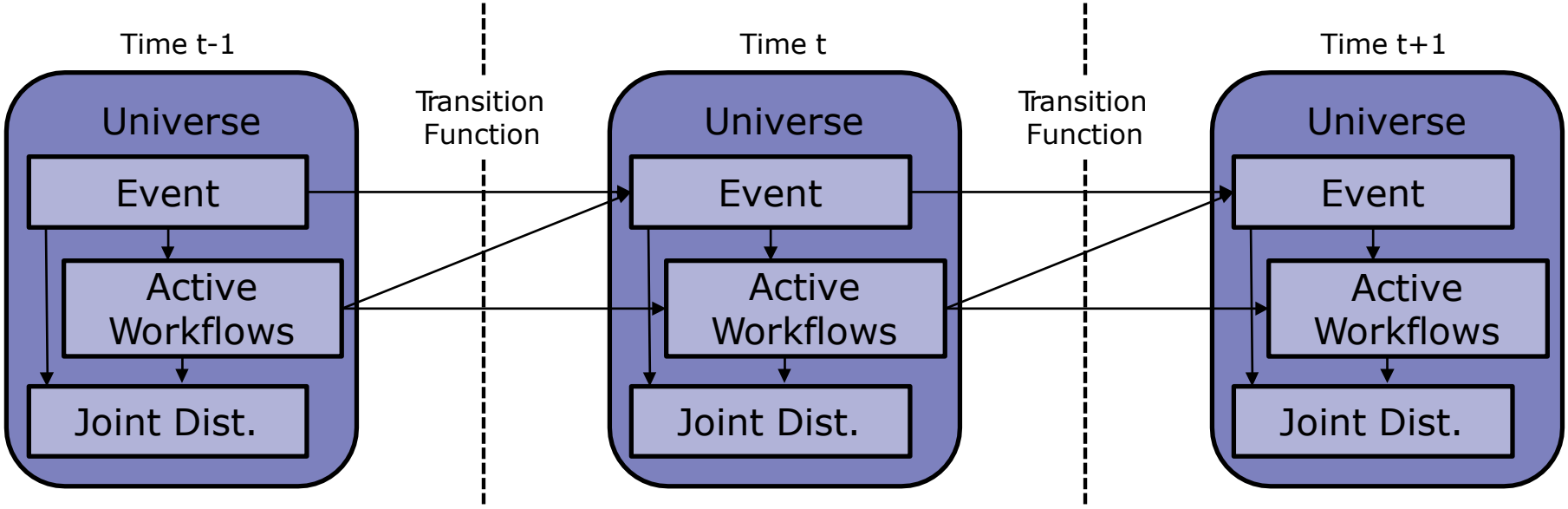
Problem

Given a sequence of desktop user events, determine the correct workflow, instance, and position of each event

Challenge: Workflows are interleaved!

Figaro Model

- Maintain a set of active workflows
- At each time point, the user can
  - Continue the current workflow
  - Switch to another active workflow
  - Start a new workflow
- Inference uses particle filtering



Workflow Identification Results

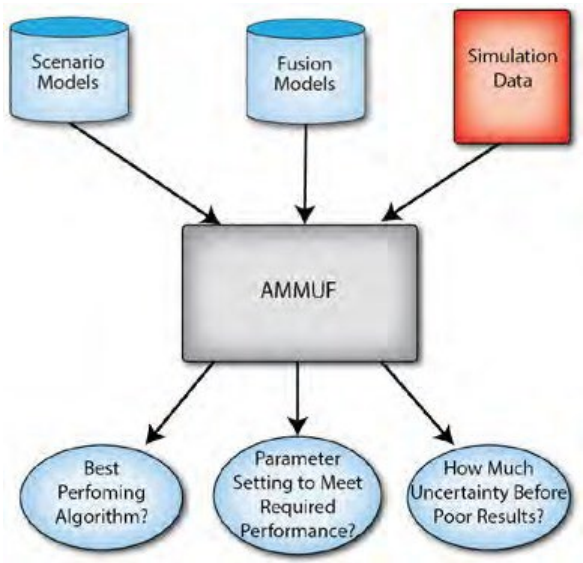
- 3rd-party evaluation:
  - Instrumented Windows desktop events
    - E.g., send email, open Word file, open URL
  - Six workflows
    - E.g., review document, compile report

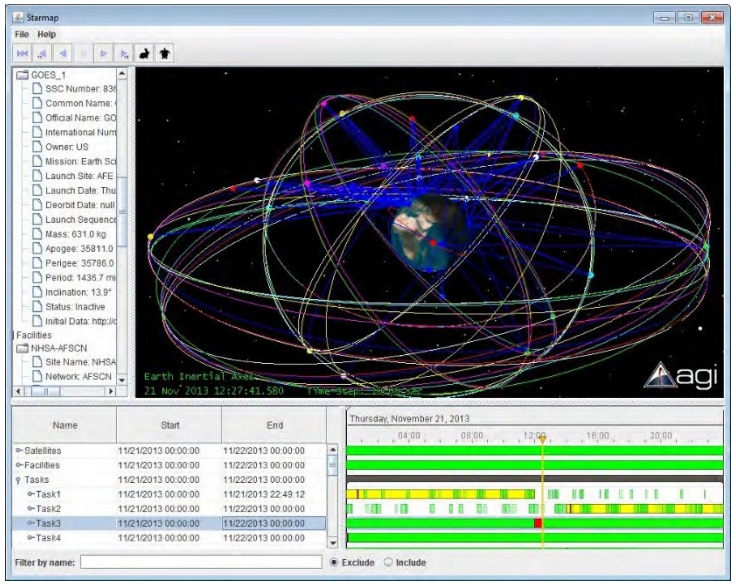
- Results
  - Over 70% of interleaved workflows correct
  - Over 80% of non-interleaved workflow correct

- The method from PAL got less than 50% and did not work with interleaved workflows

Some Other Fusion-Related Applications at Charles River

- Management of uncertainty in fusion for missile defense
- Hierarchical reasoning for space object classification
- Monitoring and predicting health of engineered systems





03

Probabilistic Programming Inference Algorithms

Probabilistic Program Inference Tasks

- Probability computation
  - Given observations about some variables (e.g., tallCenterForward)
  - Compute probability of values of other variables (e.g., Goal)
- Most probable explanation
  - Given observations about some variables
  - Compute most likely state of other variables
- Probability of evidence
  - Given observations about some variables
  - Compute probability of those observations

Algorithm Families

- Most probabilistic programming algorithms are generalizations of graphical models algorithms
- Sampling algorithms
  - Generate samples from the probability distribution
  - Compute statistics over those samples
- Factored algorithms
  - Represent model using tables called factors
  - Algorithms perform algebraic operations on factors
- Amortized inference
  - Run expensive, one-off compilation to produce fast inference model
  - Typically train a neural network to do inference with data generated from the probabilistic model

Sampling Algorithms: Rejection Sampling

- Generate samples from the program
- Delete the samples that disagree with the evidence
- Compute statistics on the remaining samples

Given our example corner kick program
With the observation that a goal was not scored

Variable	Sample 1	Sample 2	Sample 3	Sam ple 4
Tall center forward	True	False	False	Fals e
Accurate cross	False	True	True	Fals e
Good header	True	False	False	Tru e
Good goalie	True	True	True	Fals e
Goal	False	False	False	Tru e

P(accurate cross) = 2/3

Sampling Algorithms: Importance Sampling

- Similar to rejection sampling, but instead of crossing out samples, weights samples by how much they agree with the evidence
  - Works with soft evidence
  - Allows lookahead to avoid rejections
  - Variant: Sequential Monte Carlo

Soft evidence: goalie is 3 times more likely to be good

Variable	¾	¾	¾	¼
Tall center forward	True	False	False	False
Accurate cross	False	True	True	False
Good header	True	False	False	True
Good goalie	True	True	True	False
Goal	False	False	False	True

P(accurate cross) = (¾ + ¾) / (¾ + ¾ + ¾ + ¼)

Sampling Algorithms: Markov Chain Monte Carlo

- Repeatedly change the state of the system using some random process
- Every so often, collect a sample
- Variants
  - Gibbs sampling
  - Metropolis-Hastings
  - Hamiltonian Monte Carlo

Variable				
Tall center forward	True	True →	False	False
Accurate cross	False →	True	True →	False
Good header	True	True	True	True
Good goalie	True	True	True	True
Goal	False	False	False	False

Factored Algorithms

- Expresses computation as sum-of-products

P(good-goalie = True, goal = True) =

$\Sigma_{tcf}\ \Sigma_{ac}\Sigma_{gh}$

P(tall-center-forward = tcf)

P(accurate-cross = ac)

P(good-header = gh | tall-center-forward = tcf, accurate-cross = hc)

P(good-goalie = True)

P(goal = True | good-header = gh, good-goalie = True)

- Factored algorithms primarily work with discrete problems, but can be faster and more accurate than sampling algorithms

Factored Algorithms: Variable Elimination

- Rearranges computation of sums of products for maximum efficiency
- Produces exact answer!
- Complexity exponential in a derived property of the graph describing the computation

Factored Algorithms: Belief Propagation

- Solve computation by message-passing
- Exact for programs without loops
- Runs in linear time in size of model
- With loops, generally gives good answers, but no guarantees of convergence or accuracy

Two Big Issues for Probabilistic Program Inference

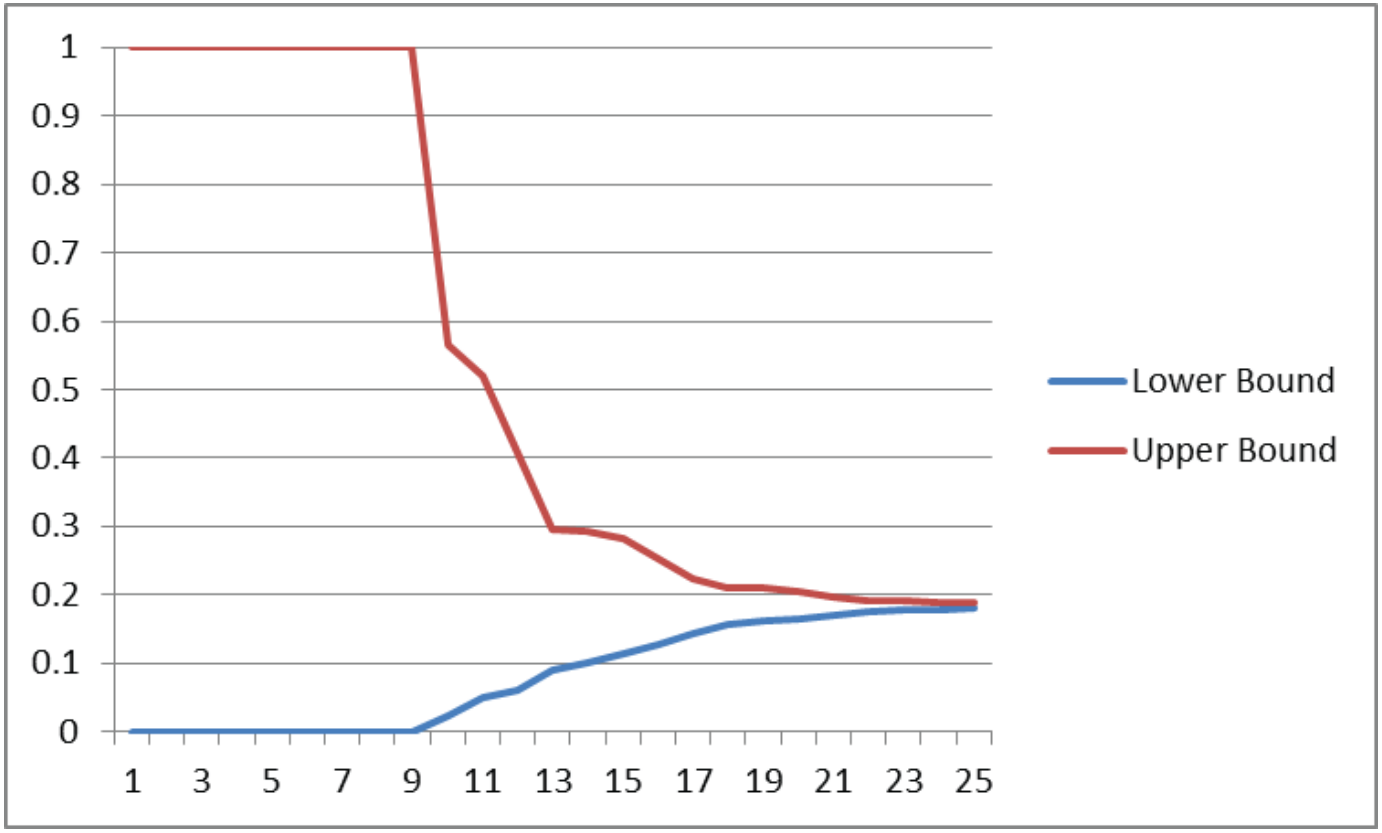
1. Probabilistic programming models can define a very large or infinite number of variables
   - Can't generate samples or create factor graph
2. We want to create an easy to use framework for building probabilistic applications, but there are so many algorithms to choose from and configure
   - Barrier to entry for non-ML expert users

Very Large Models: Lazy Inference [Pfeffer et al. 15]

- Expand only the most relevant parts of the model
- Quantify the effect of the unexpanded part on the query
- Use this to provide bounds on the answer
- Refine as desired to improve the bounds

Lazy Inference: Unbounded But Finite Grammar

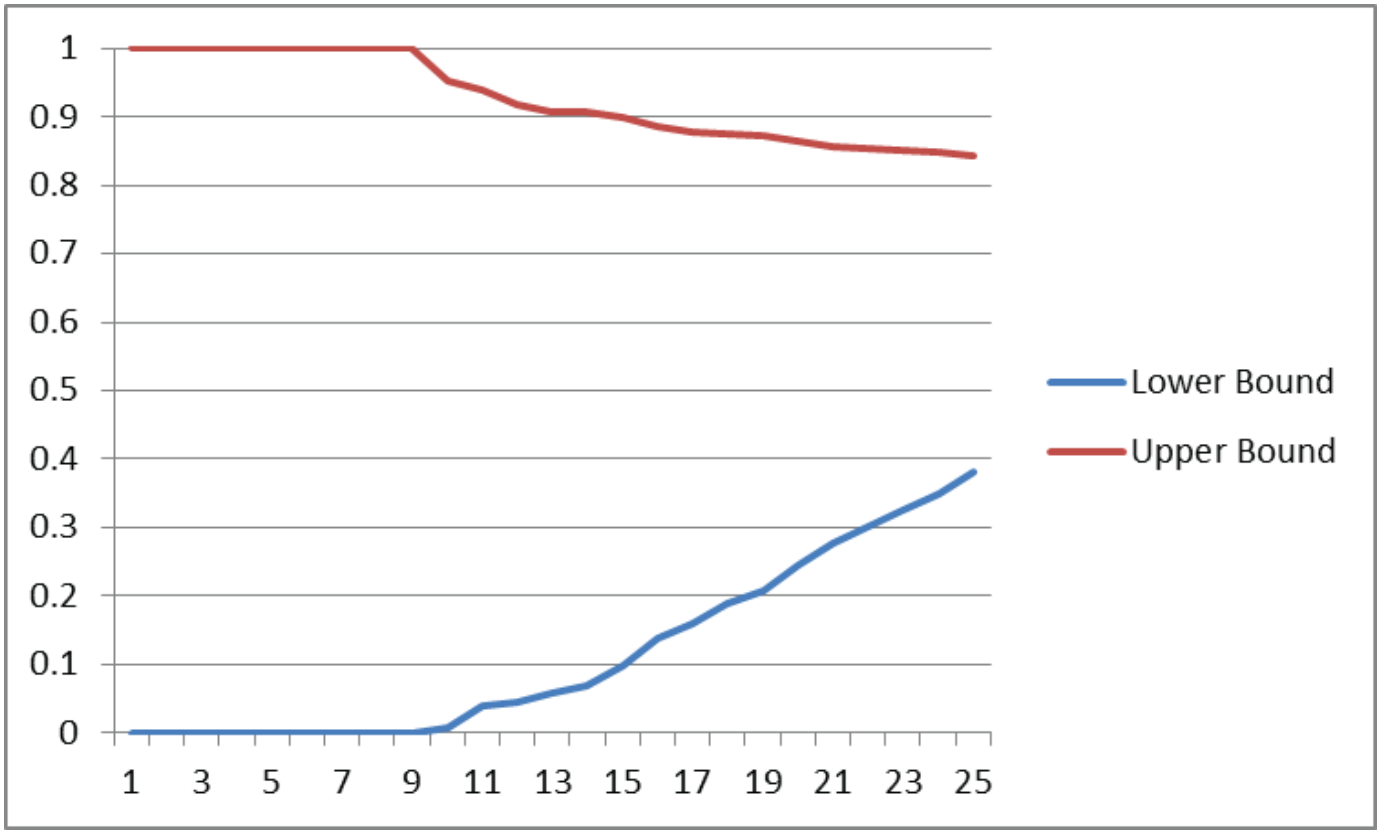
- Grammar generates sentences of any length
- Query is whether the sentence contains a specific subsentence



Lazy inference provides accurate answers on infinite models

Lazy Inference: Infinite Grammar

- Grammar generates infinite sentence with positive probability!
- No sampling or non-lazy method can produce any answer



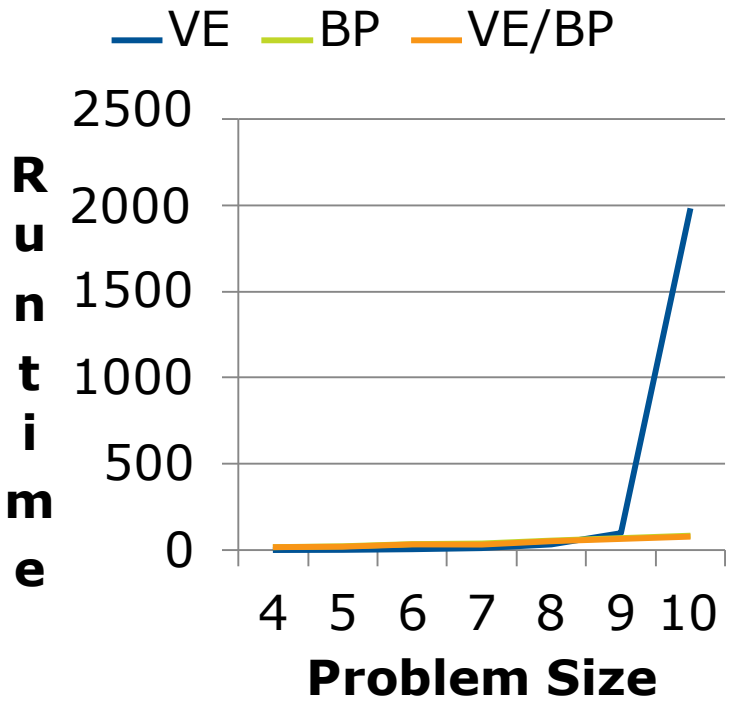
Lazy inference can answer queries no other method can

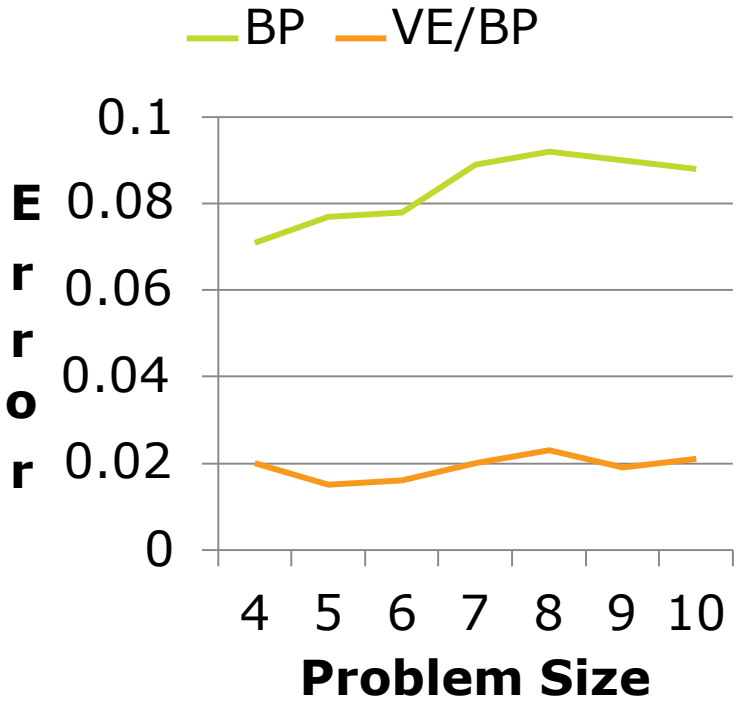
Automatic Optimization: Structured Factored Inference (SFI) [Pfeffer et al. 18]

- A method to automatically decompose a probabilistic program and optimize each part independently
  - Analogous to database query optimization
- **Step 1:** Decompose inference and use dynamic programming
- **Step 2:** Optimize each subproblem separately
- Two major advantages of this approach
  - It's much easier to decide what algorithm to use on a small subproblem than a large problem
  - We get to use different algorithms for different subproblems

Automatic Optimization: Medical Diagnosis Problem

- Based on QMR-DT benchmark
- Complexity of exact inference grows exponentially with problem size
- VE = variable elimination, BP = belief propagation
- VE/BP is SFI algorithm that automatically chooses between them





SFI is much faster than VE and much more accurate than BP

04

Probabilistic Programming for Long-Lived AI Systems

What do I Mean by Long-Lived AI?

- Agents interact with their environment through sensors and actuators
- Long-running interactions throughout the lifetime of the AI system
- Open-ended environments of particular interest

- Note: Long-lived AI does not necessarily imply physical robots
- Examples:
  - Chatbot conversing on open-ended series of topics
  - Virtual scientist formulating hypotheses, designing experiments, and developing theories
  - And of course, the household robot taking care of our various needs

Deep Q Learning for Long-Lived AI

- Impressive results, but some limitations:
- Extremely data hungry
  - Huge number of interactions required in simulated environment
- Limited capacity to transfer
  - E.g., Kansky et al., 2017: System trained on Breakout fails on minor variants of game
- Cannot distinguish causation from correlation
- Struggles with open-ended inference that requires information outside the specific input

Beyond Deep Learning

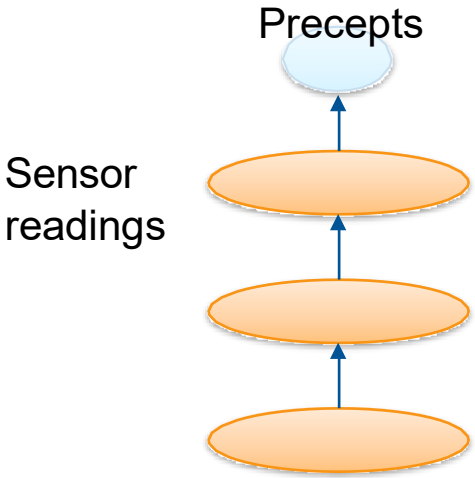
- Neural networks are bottom-up data structures
- If we could combine top-down knowledge with bottom-up sensor processing, maybe we can do better
- Useful knowledge can make systems less data-hungry
- General world knowledge can help transfer
- Knowledge can explicitly model causal relationships
- Open-ended world knowledge can be brought to bear

But we must not lose deep learning's ability to learn complex functions that can't be programmed!

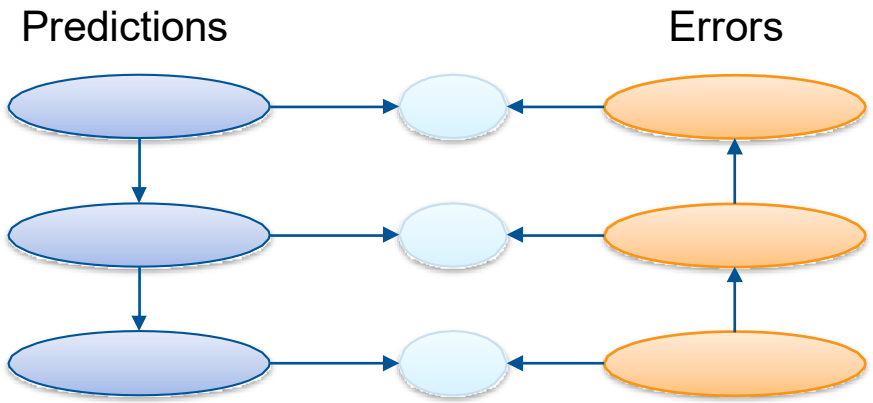
An Assist from Cognitive Science: Predictive Processing

- Traditionally:
  - Brain encodes sensory stimuli as they occur
  - Beliefs and concepts are result of perception
- Predictive processing (Friston, Hohwy, Clark, Rao & Ballard):
  - Beliefs about world yield predictions about sensory signals
  - Sensory cortex encodes prediction error
  - Perception results from combination of prediction and error

Traditional view



New view



Introducing Scruff [Pfeffer & Lynn 18]

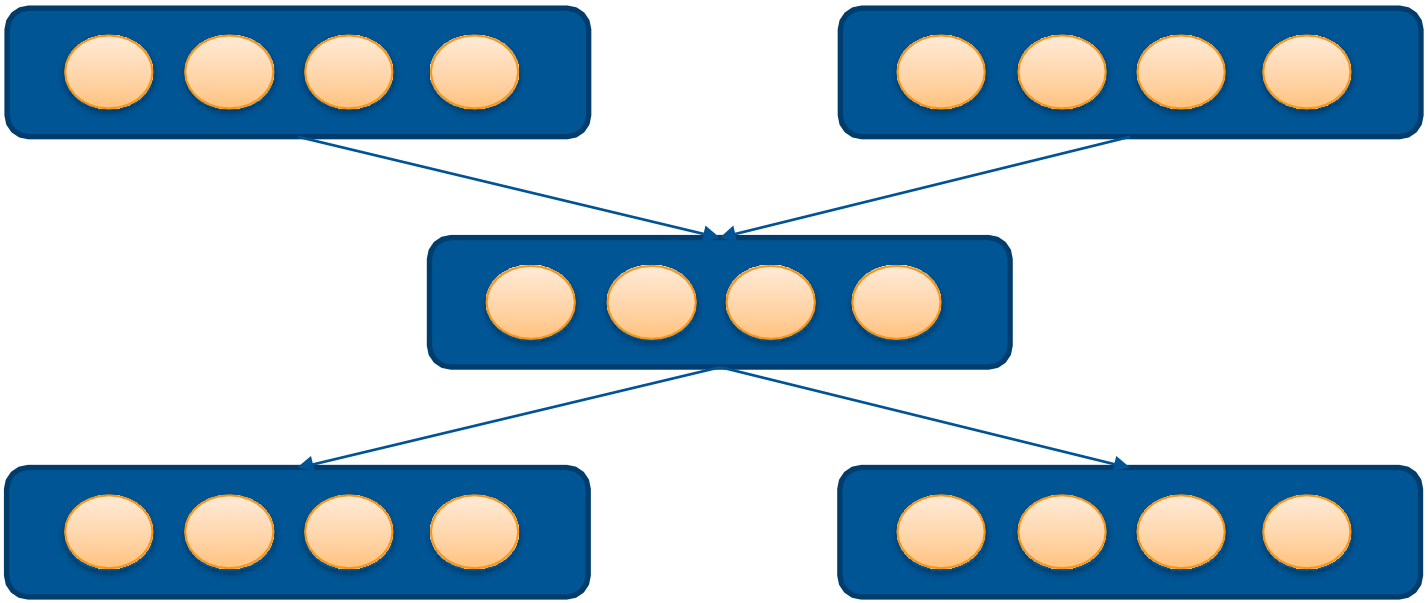
- A new probabilistic programming framework based on predictive processing
- Unlike traditional probabilistic programming, Scruff models are hierarchical networks with many layers of nodes
- Each node is represented by a probabilistic program

Scruff

- Bayesian models fit predictive processing well
  - Prediction = prior
  - Errors = likelihoods
  - Percepts = posterior
- Scruff lets us express prior domain knowledge using programs
- Take observations and reconcile with predictions to form posterior beliefs
- Programs let us work with more interesting data structures than just enumerated or continuous random variables

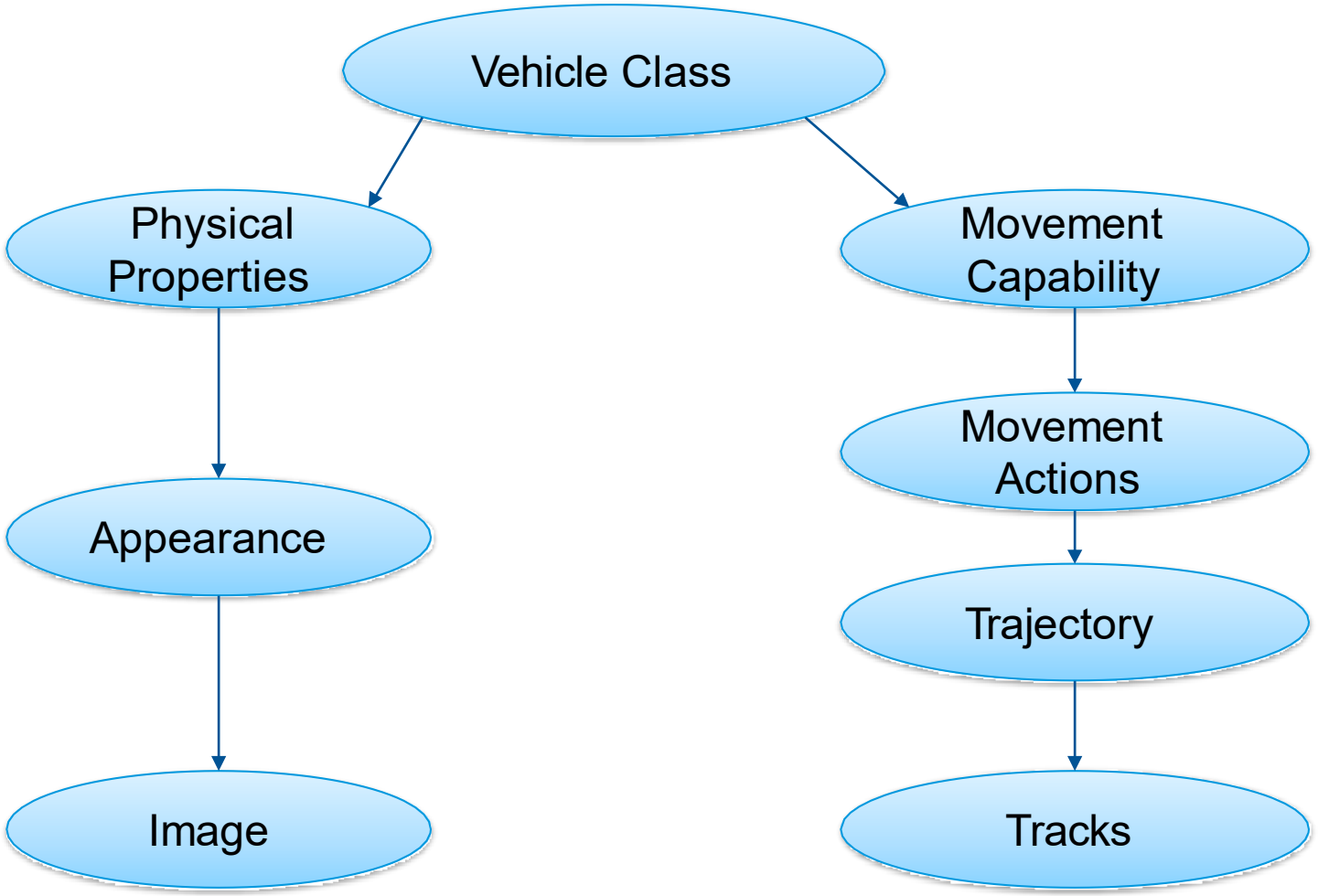
Hierarchical Representations

- Layer functions
  - Vector of nodes provide distributed encoding like neural net
  - Individual node conditional probabilities structured using programming language facilities



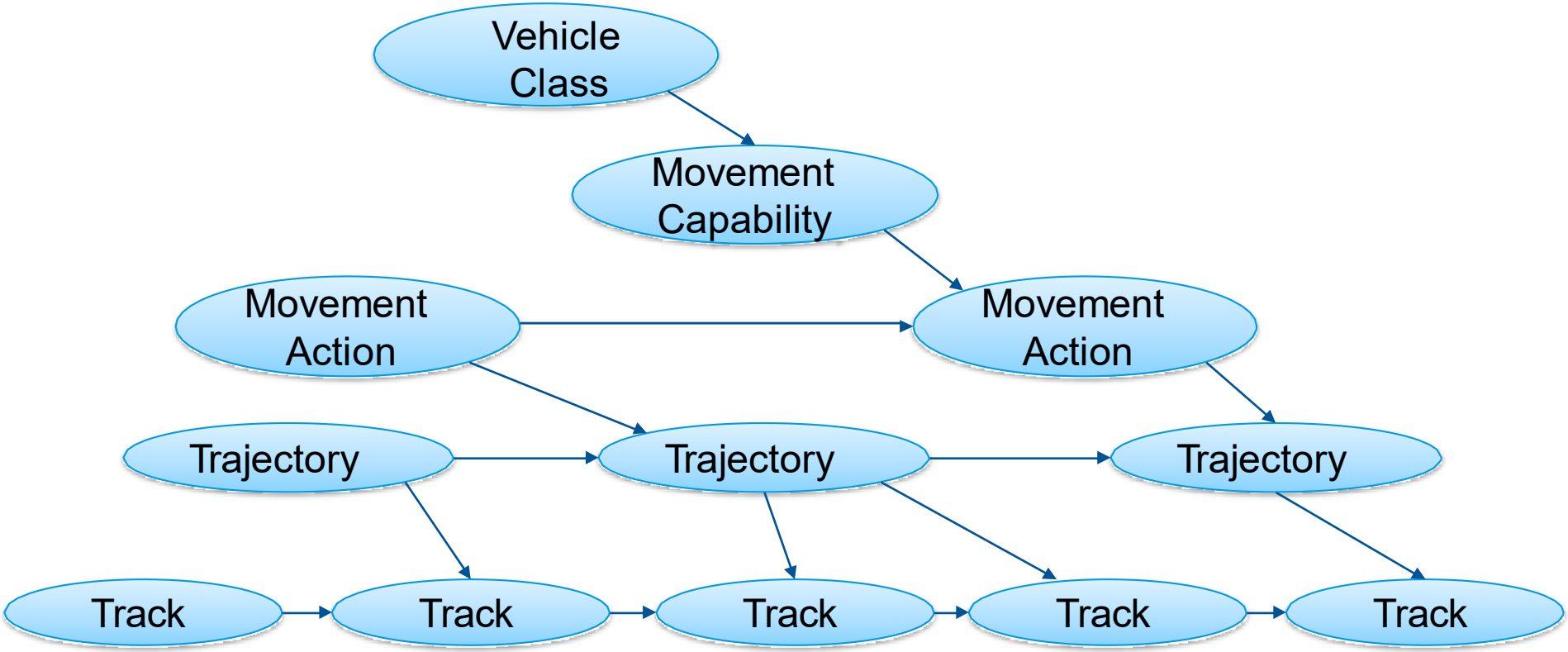
Higher layers provide: abstraction, aggregation, context Lower layers provide: specialization, decomposition

Example Hierarchy: Monitoring Vehicles



Modeling Different Time Rates

- Each variable takes a time argument
- State of layer depends on previous state of same layer and current state of parent layer
- Time deltas can be customized for each layer

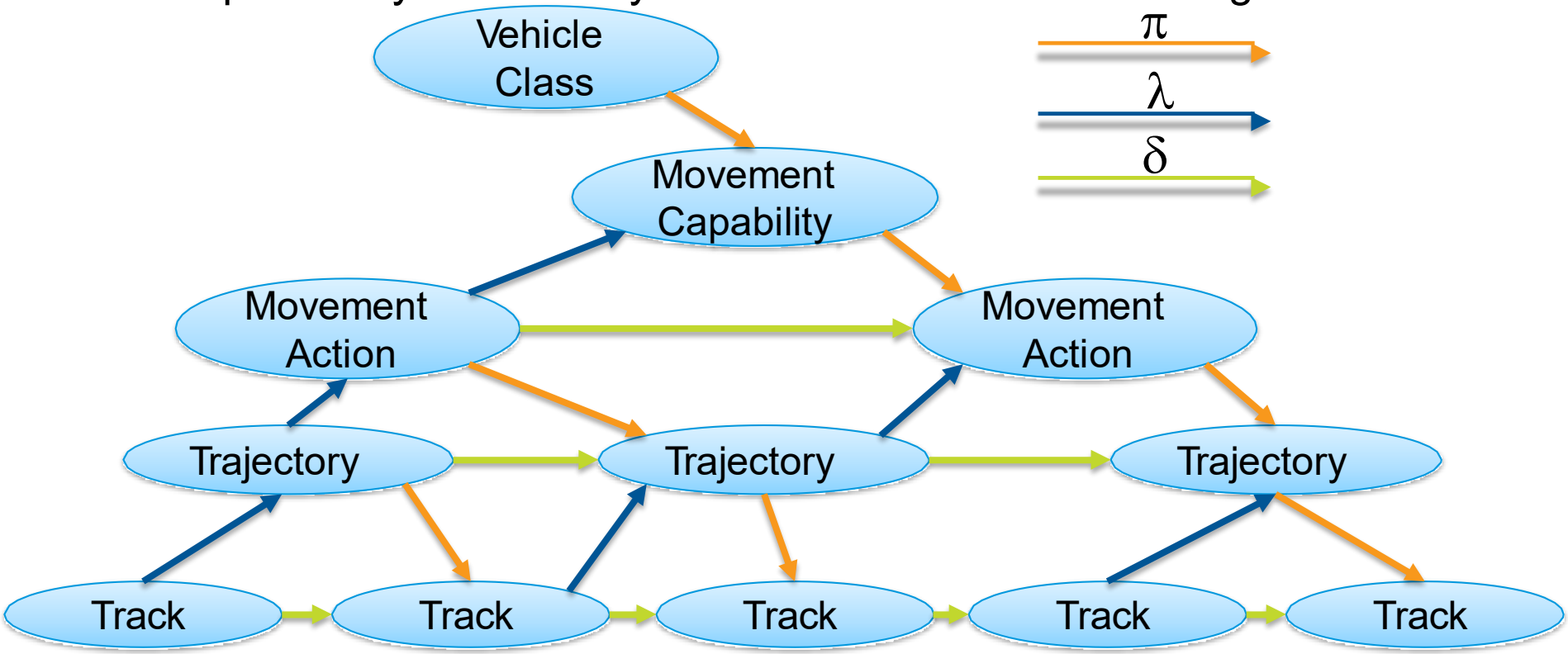


Goal of Monitoring and Learning

- Estimate the state of the system over time
  - Each node maintains beliefs about its state
  - Represented as probabilities of different hypotheses
  - Nodes update asynchronously at the appropriate rate
- Predict future developments
  - From the current state estimate, envision probable futures
- Learn to improve the system
  - Bayesian update
  - Gradient descent
  - Abductive hypothesis generation
- Goal is to develop a distributed, real-time system

Inference by Asynchronous Belief Propagation

- π messages encode predictions from parents
- λ messages encode likelihoods from children
- δ messages encode drift from same node at previous time
- Nodes update asynchronously based on most recent messages



Examples of Surprise and Adaptation (Planned)

- Case 1: Ordinary sensor noise
  - Truck moving at 100kph
  - Sensors provide noisy and intermittent observations
  - Scruff interprets this as a normal situation of truck maintaining 100kph
- Case 2: Surprising, but intermittent, sensor reading
  - Truck moving at 100kph, with momentary 120kph sensor reading
  - Scruff interprets this as sensor fault
  - Truck still held to be traveling at 100kph
- Case 3: Truck goes faster
  - Sensor gradually shifts from 100kph to 180kph
  - 180kph is faster than previously believed max speed of the truck
  - Scruff modifies the movement capability of the truck
- Case 4: Truck goes unbelievably fast
  - Sensor increases to 250kph
  - This is faster than any believed speed for a truck
  - Scruff maintains two hypotheses
    - Vehicle class is different – truck is disguise
    - Sensor is biased – but this is refuted by sensor reading of other vehicles

Status

- We've built a proof-of-concept demo of Scruff for a simple scenario
- We're now working on a robust, scalable implementation
- We plan to make it easy to use, with extensive representation, control, and reporting options
- We intend to make it open source

- Please send me an email if you would like to be notified when the first public version is available

Conclusion

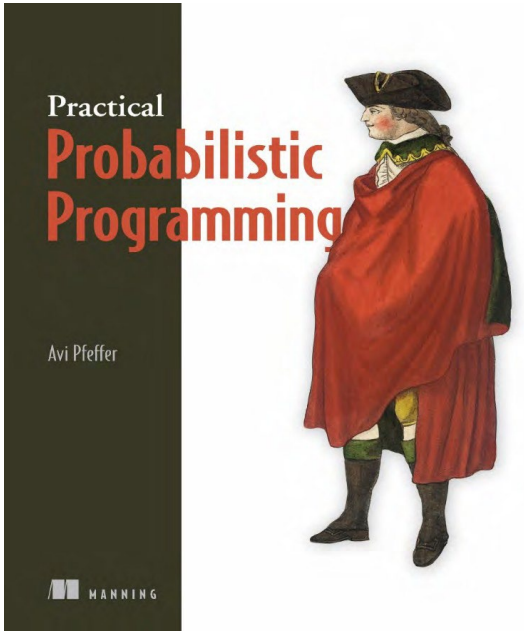
- Probabilistic programming makes it easier to develop applications to predict, infer, and learn for fusion and beyond
- Probabilistic programming languages are maturing and have a wide variety of applications
- Significant improvements have been made in probabilistic program inference in the last few years
- Scruff shows the way to a future of probabilistic programming as the basis for AI systems that interact with the environment over a long period of time

Acknowledgement

- This material is based upon work supported by the United States Air Force under Contract No. FA8750-14-C-0011.
- Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

More Information

- Figaro is open source
  - Contributions welcome!
  - Releases can be downloaded from www.cra.com/figaro
  - Figaro source is on GitHub at https://github.com/charles-river-analytics/figaro



- Charles River Analytics is hiring
- Employee-owned company

BOSTON BUSINESS JOURNAL

BPtW

2019 BEST PLACES TO WORK

Contact information: apfeffer@cra.com