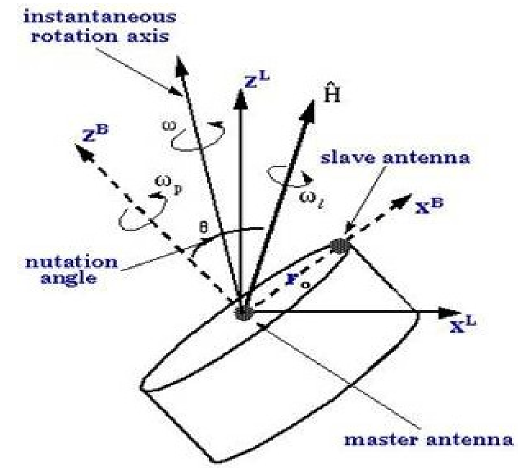
Using Software Transformation Systems as Program Generator Backends

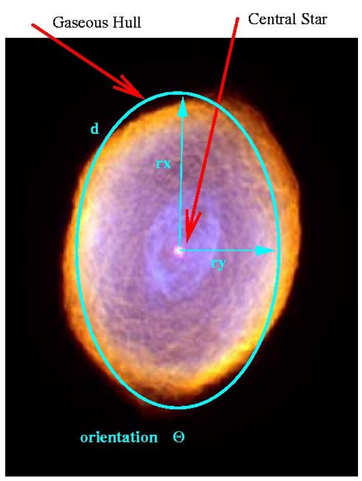
Ewen Denney Johann Schumann
USRA/RIACS, NASA Ames

Bernd Fischer
ECS, U Southampton

Application domains

- Two main application domains
  - Guidance, Navigation & Control
  - Data Analysis
- Two common characteristics
  - Concise mathematical models
  - Algorithmic variability
- Top two algorithm families
  - Kalman Filters → AutoFilter
  - Clustering → AutoBayes
- Highly *mathematical* domains





GN&C

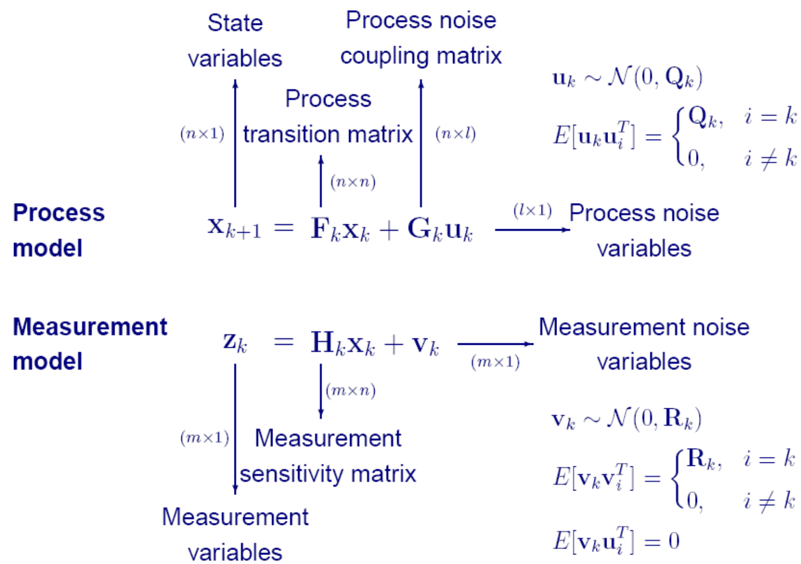
Spacecraft, aircraft, ships, and (increasingly) cars require methods for the accurate determination of *position* and *attitude*

- Equipment:
  - Compass, clock, GPS, INS
  - Radio navigation (DME, Radar)
- Problems:
  - Measurements are noisy
  - Each measurement contributes partial information
  - Sensor failures (degradation, transient, permanent)
- Overall task:

  Calculate the best possible state estimate using all available information

Synthesis for GN&C

- Standard technology: Kalman filters
- Commercial autocoders insufficient
  - algorithmic variability
  - not adaptable
- Specialized generator for GN&C: AutoFilter



- High-level domain-specific modeling language
  - differential equations
- Supports model-based development

Kalman Filters: Model

Process model

$$x_{k+1} = F_k x_k + G_k u_k$$

- State variables $(n\times 1)$: x_k
- Process transition matrix $(n\times n)$: F_k
- Process noise Coupling matrix $(n\times 1)$: G_k
- Process noise variables $(l\times 1)$: u_k

$$u_k \sim N(0, Q_k)$$

$$E[u_k u_i^T] = \begin{cases} Q_k, & i = k \\ 0, & i \neq k \end{cases}$$

Measurement model

$$z_k = H_k x_k + v_k$$

- Measurement variables $(m\times 1)$: z_k
- Measurement sensitivity matrix $(m\times n)$: H_k
- Measurement noise variables $(m\times 1)$: v_k

$$v_k \sim N(0, R_k)$$

$$E[v_k v_i^T] = \begin{cases} R_k, & i = k \\ 0, & i \neq k \end{cases}$$

$$E[v_k u_i^T] = 0$$

Kalman Filters: Model

Attitude, speed, etc.

Flight dynamics

$(n\times 1)$ $(n\times n)$

Process model

$$x_{k+1} = F_k x_k + G_k u_k$$

$(l\times 1)$ → Turbulences, headwind, etc.

Measurement model

$$z_k = H_k x_k + v_k$$

$(m\times 1)$ → Radar uncertainty, etc.

$(m\times 1)$ $(m\times n)$

Instrument wiring

Altimeter, radar, barometer, etc.

Kalman Filters: Model

Location, yaw, yaw rate

Rover dynamics

$(n \times 1)$ $(n \times n)$

Process model

$$x_{k+1} = F_k x_k + G_k u_k$$

$(l \times 1)$ Wheel slippage

Measurement model

$$z_k = H_k x_k + v_k$$

$(m \times 1)$ Gyro uncertainty, etc.

$(m \times 1)$ $(m \times n)$

Sensors

IMU, wheel odometry

Kalman Filters: Algorithm

1. Initialization: initialize all vectors and matrices.

$$K = P^- H^T (HP^- H^T + R)^{-1}$$

2. Measurement update: read and process measurement *z*.

$$x^+ = x^- + K(z - Hx^-)$$

$$P^+ = (I - KH)P^-$$

3. Temporal update: *estimate* one step ahead in time.

$$x^- = Fx^+$$

$$P^- = FP^+F^T + Q$$

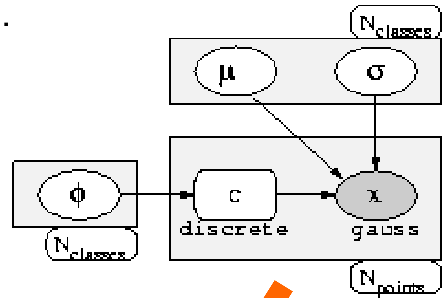
4. Go to 2.

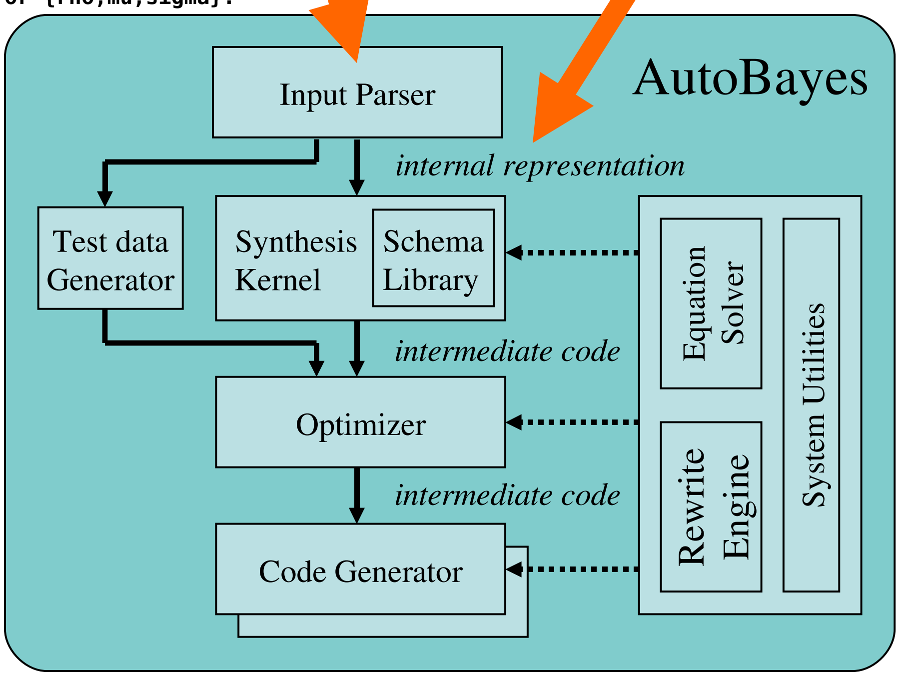
Synthesis architecture

```
model mog as 'Mixture of Gaussians'.

const nat n_points.
  where 0 < n_points.
const nat n_classes := 3.
  where n_classes << n_points.
...
double mu(0..n_classes-1).
double sigma(0..n_classes-1).
  where 0 < sigma(_).
...
data double x(0..n_points-1).
x(I) ~ gauss(mu(c(I)), sigma(c(I))).

max pr(x|{rho,mu,sigma})
for {rho,mu,sigma}.
```





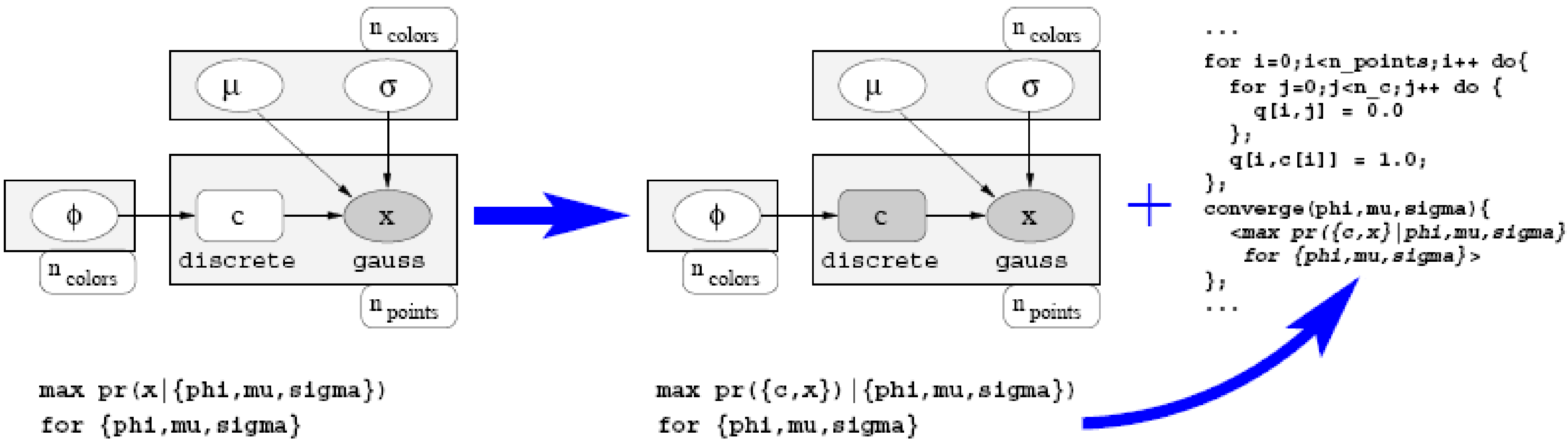
- Schema library
- Symbolic subsystem
  - rewrite engine
  - symbolic differentiation
  - (polynomial) equation solver
- Procedural intermediate language
- Multiple backends
  - C/C++ based: Octave, Matlab, CLARAty
- Multiple programs synthesized

Schemas

- Algorithmic knowledge encoded as *schemas*
  - Schema = Conditions + Code fragment
  - Recursively composed
  - Progressive instantiation of solution
  - Generates platform-independent intermediate code

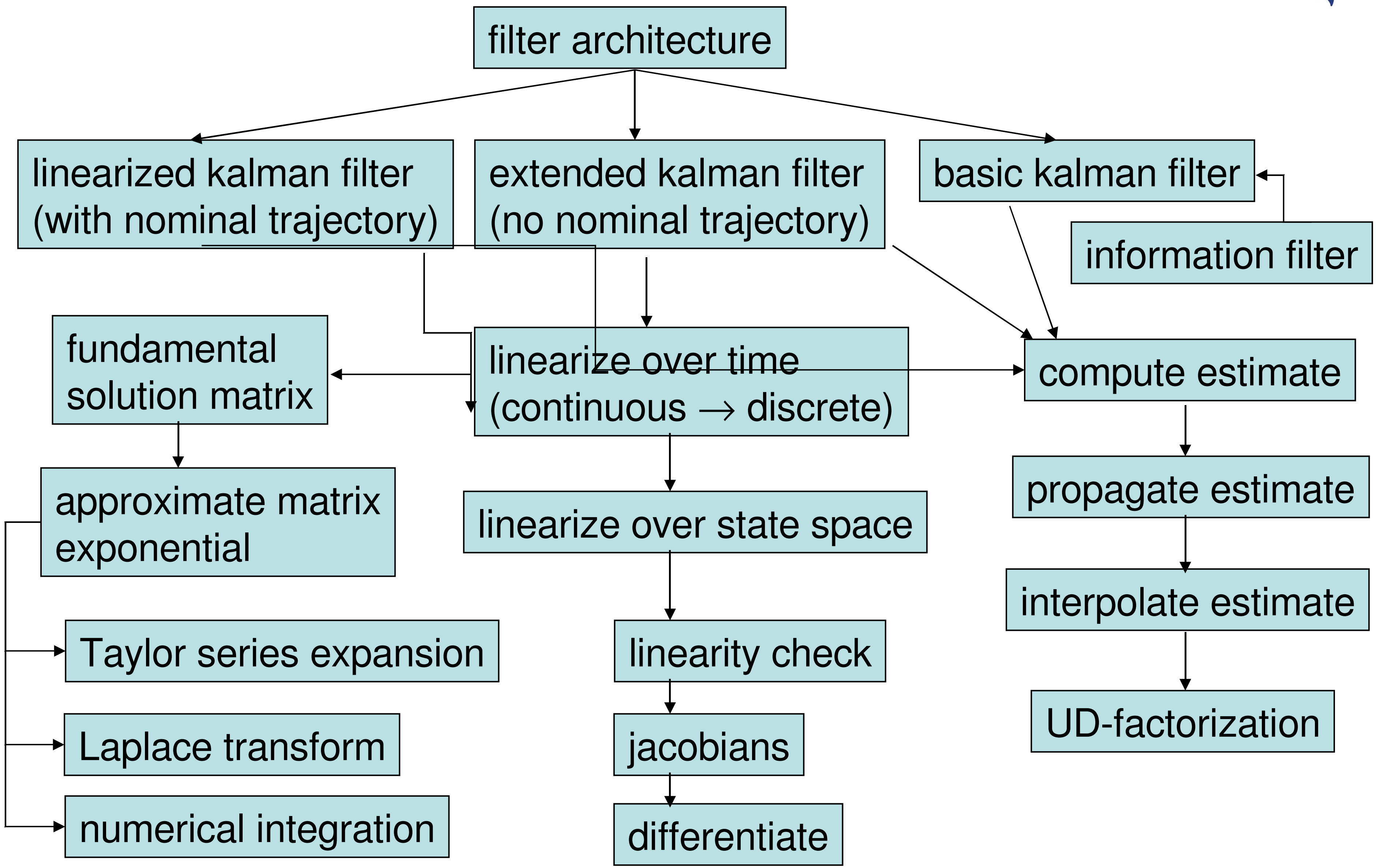
```
schema( max P(U|V) wrt V, Code_fragment↑ ) :-
    ...                                          (* applicability constraints *)
  → Code_fragment =
    begin
      ⟨guess values for c[i]⟩                     (* Initialize *)
      for i:=1 to N do for j:=1 to M do q[i,j] := 0;
      for k:=1 to N do q[k,c[k]] := 1;
      while-converging(V) do
        ⟨ max P({q,U}|V) wrt V⟩                    (* M-step *)
        q[i,j] := ⟨...⟩                            (* E-step: calculate P(q|{U,V})
      end                                          (* end while-converging *)
    end
```

Schemas as Transformations



- Big-step transformations
  - horizontal (model decompositions / transformations)
  - vertical (domain-specific algorthms)
- Implemented as combination of techniques
  - meta-program (check conditions)
  - graph rewriting (transform model)
  - templates (represent code fragments)

Schema Hierarchy



Rewrites as Transformations

Small-step transformations encoded as conditional rewrites: C => L = R

- Differentiation
- Discretization
- Taylor expansion
- Matrix identities
- Linearization of set of equations
- Approximations
- Trigonometry

(+ simple algebraic identities)

STS for Optimizations (I)

Observation: schema-based program construction offers *opportunities* for optimization

- based on use of independent building blocks
  - loop fusion
- based on instantiation of building blocks
  - loop unrolling
  - strength reduction
  - scalarization
- based on repeated use of specification information
  - constant propagation

STS for Optimizations (II)

Observation: schema-based program construction offers *support* for optimization

- exploits knowledge available at synthesis time

```
schema(max f(X,Y) wrt X, Prog) :-
   Prog = <numeric optimization routine>
```

  → can hoist Y out of loops without dataflow analysis

- similar in spirit to anticipatory optimization
- requires integration of optimization and synthesis

Conclusions

- Highly mathematical domains
  - rich structure for transformations
- Combination of synthesis and optimization promising
  - maximum effect requires tight integration