Graphical Models

Lecture 11:

Clique Trees

Andrew McCallum
mccallum@cs.umass.edu

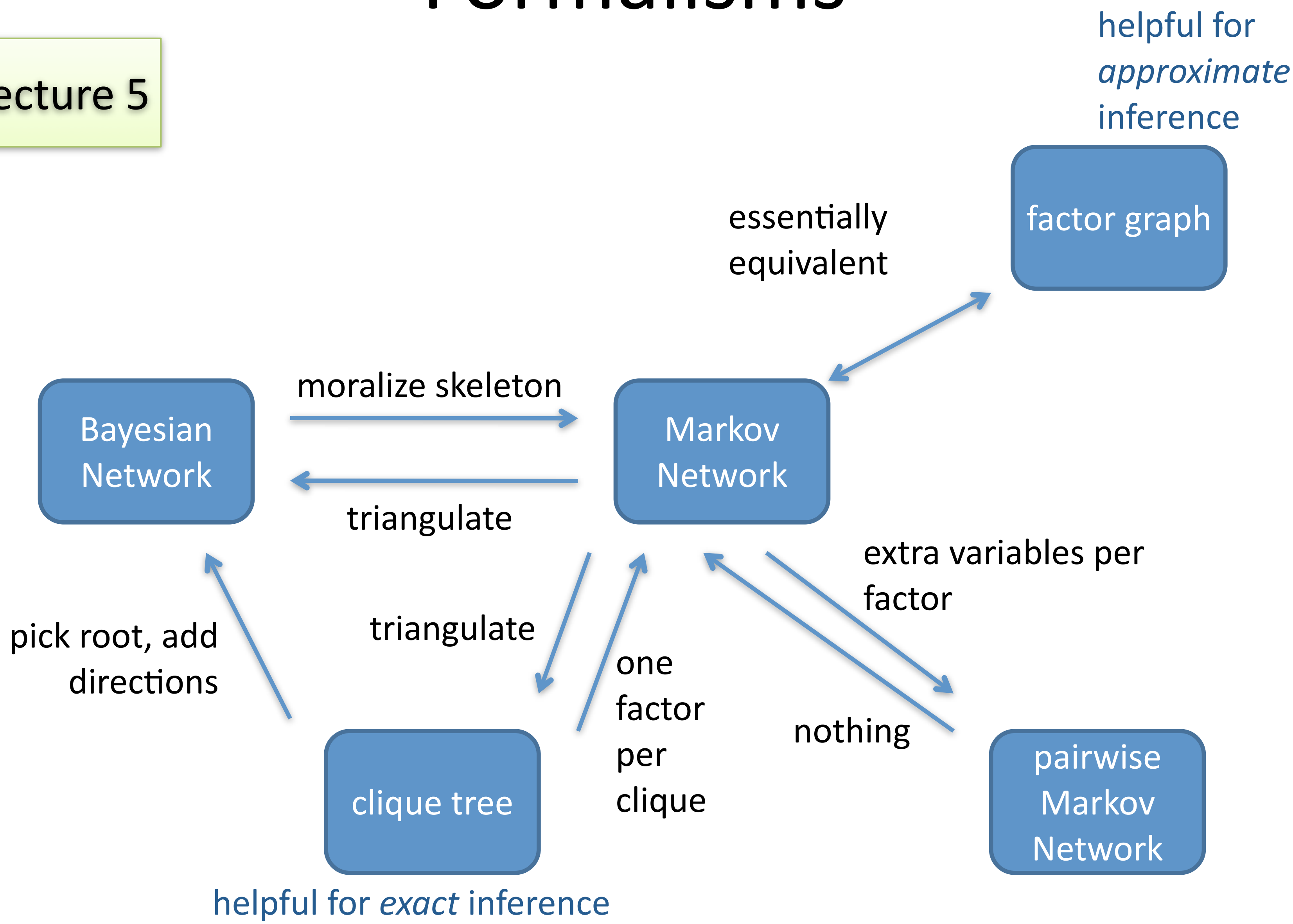
Thanks to Noah Smith and Carlos Guestrin for some slide materials.

Inference

- Last two lectures: variable elimination
  - Sum out variables one at a time.
- Today: alternative algorithms.
  - Also based on factors.
  - Central data structure: **clique tree**.

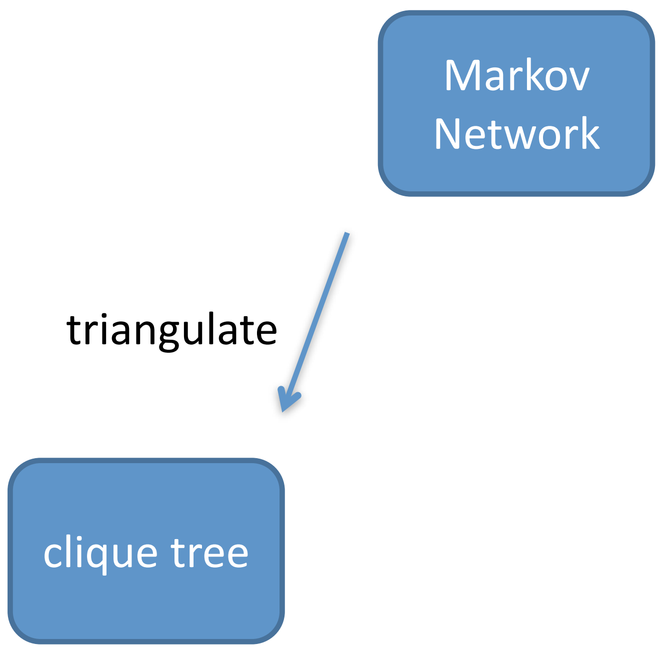
Formalisms

Lecture 5



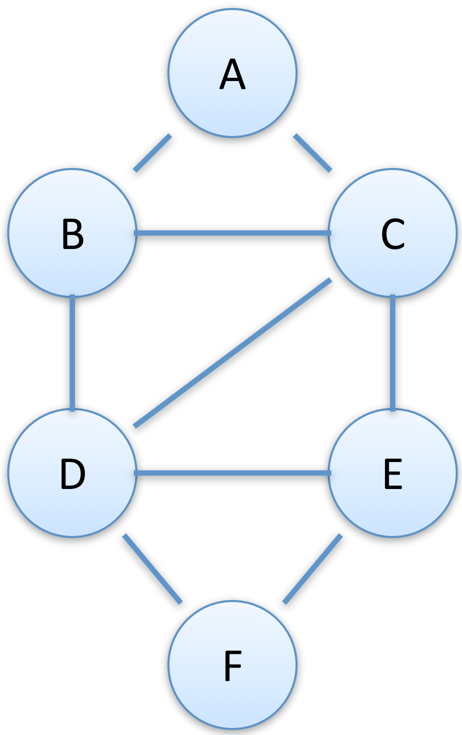
Formalisms

Lecture 5



Lecture 5

Clique Tree

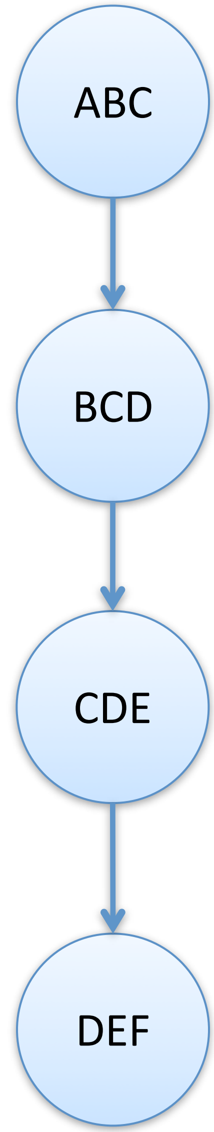


Every maximal clique becomes a vertex.

Connect vertices with overlapping variables

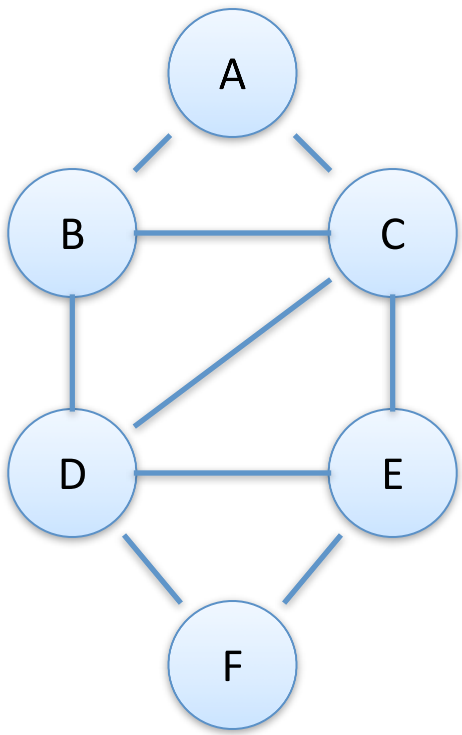
Tree structure? then “Clique Tree”

(If graph was triangulated, always get a tree structure.)



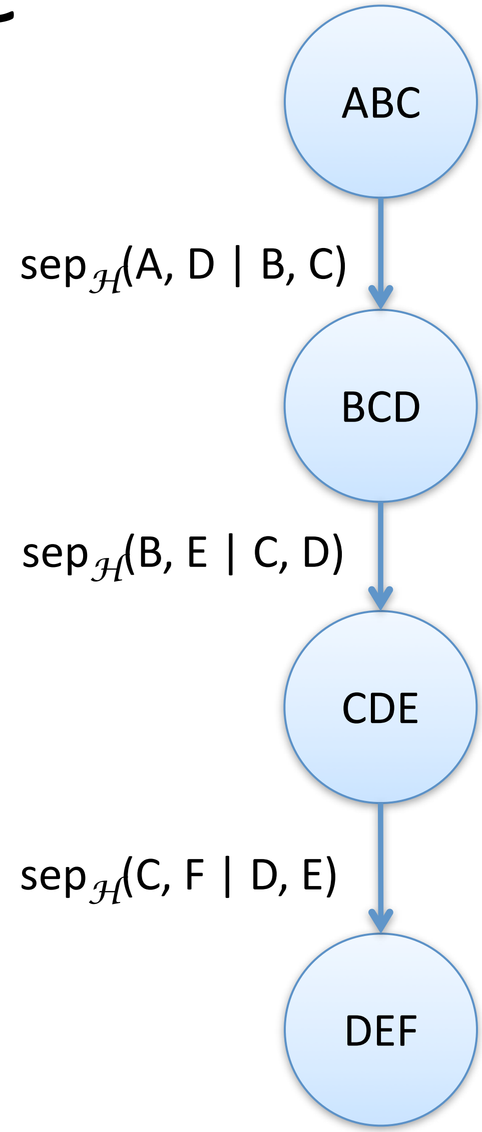
Lecture 5

Clique Tree



“Sep-set” = {A, D}

For each edge, intersection of r.v.s *separates* the rest in $\mathcal{H}$.

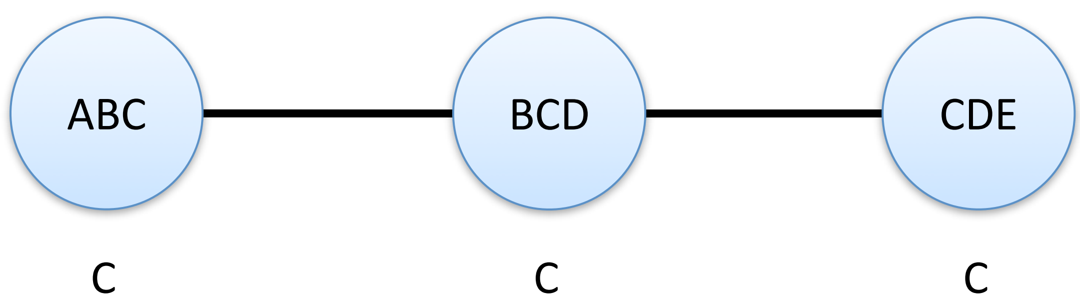


“Cluster Graph”

- A cluster graph for a set of factors Φ over $\mathcal{X}$ is an undirected graph, each of whose nodes *i* is associated with a subset $\mathbf{C}_i \subseteq \mathcal{X}$.
- A cluster graph must be “**family preserving**”: each factor $\phi \in \Phi$ must be associated with a cluster $\mathbf{C}_i$ (denoted $\alpha(\phi)$), such that $\text{Scope}[\phi] \subseteq \mathbf{C}_i$.
- Each edge between a pair of clusters $\mathbf{C}_i$ and $\mathbf{C}_j$ is associated with a **sepset** $S_{i,j} \subseteq \mathbf{C}_i \cap \mathbf{C}_j$.
- “**Cluster Tree**” a cluster graph that is a tree.

"**Running Intersection**" Property

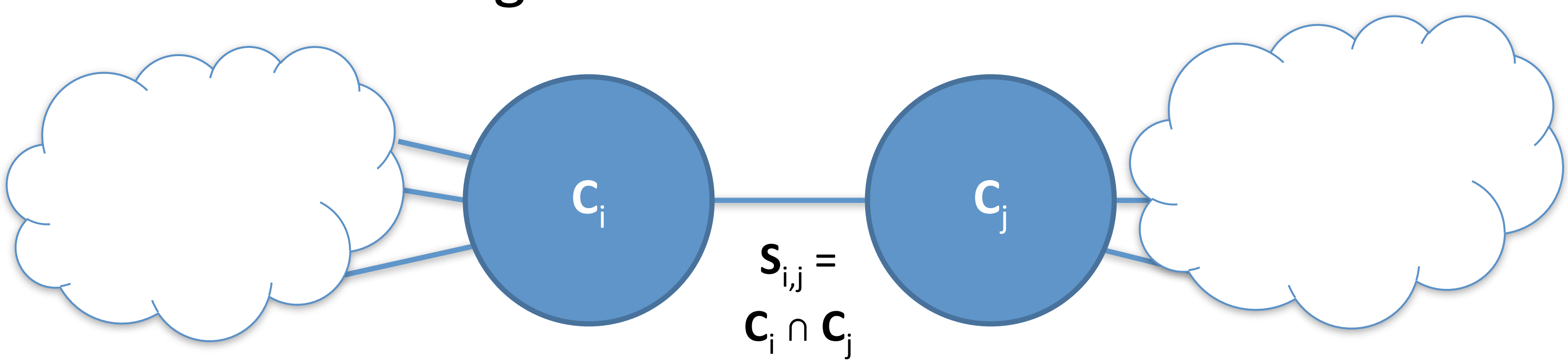
- Cluster tree $\mathcal{T}$ (with vertices C_i…) has the **running intersection property** if,
  whenever there is a variable X such as $X \in C_i$ and $X \in C_j$,
  then X is also in every cluster in the (unique) path in $\mathcal{T}$ between C_i and C_j.



"Clique Tree" Definition

aka *"Junction Tree,"* or *"Join Tree"*

- Given an undirected graph $\mathcal{H}$,
  a tree $\mathcal{T}$ is a **clique tree** for $\mathcal{H}$ if:
  - Each node in $\mathcal{T}$ corresponds to a clique in $\mathcal{H}$
  - Each maximal clique in $\mathcal{H}$ is a node in $\mathcal{T}$
  - Each sepset $\mathbf{S}_{i,j}$ separates the variables strictly on one side of the edge from the variables on the other side.



In other words

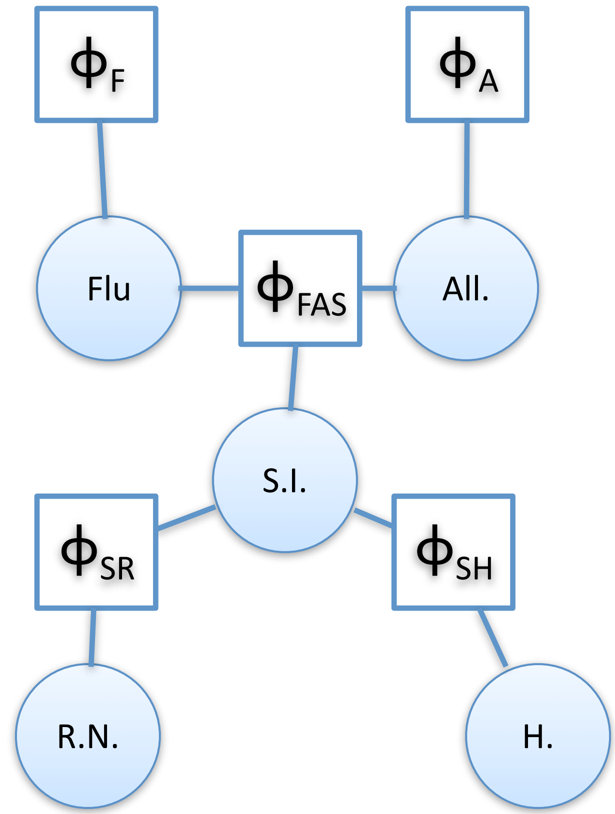
Clique Tree = cluster tree that satisfies running intersection property.

How to Obtain a Clique Tree
Option 1

- Start with factor graph or undirected graph.
- Do variable elimination. On iteration i:
  - Let ψ_i be the intermediate factor
  - Let τ_i be the marginalized factor
  - Let $\mathbf{C}_i$ be the set of variables involved in ψ_i
- One node per $\mathbf{C}_i$, edge $\mathbf{C}_i$-$\mathbf{C}_j$ when τ_i gets used in computing ψ_j.

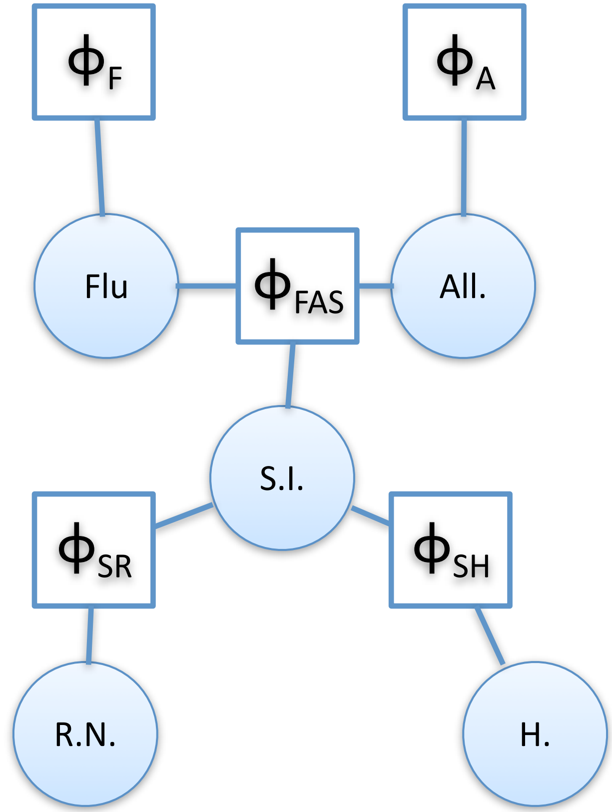
Example

- Eliminate S.



Example

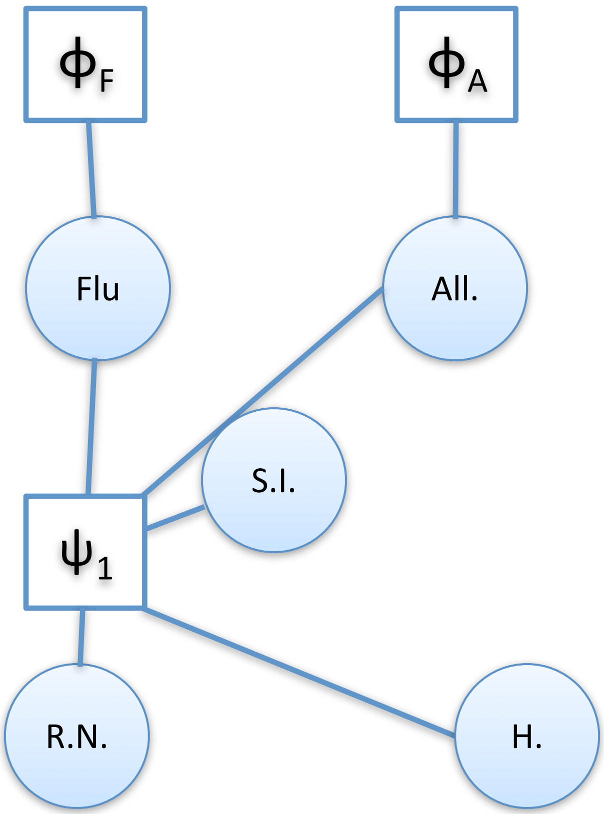
- Eliminate S.
- $\psi_1 = \phi_{FAS} \cdot \phi_{SR} \cdot \phi_{SH}$



Example

- Eliminate S.

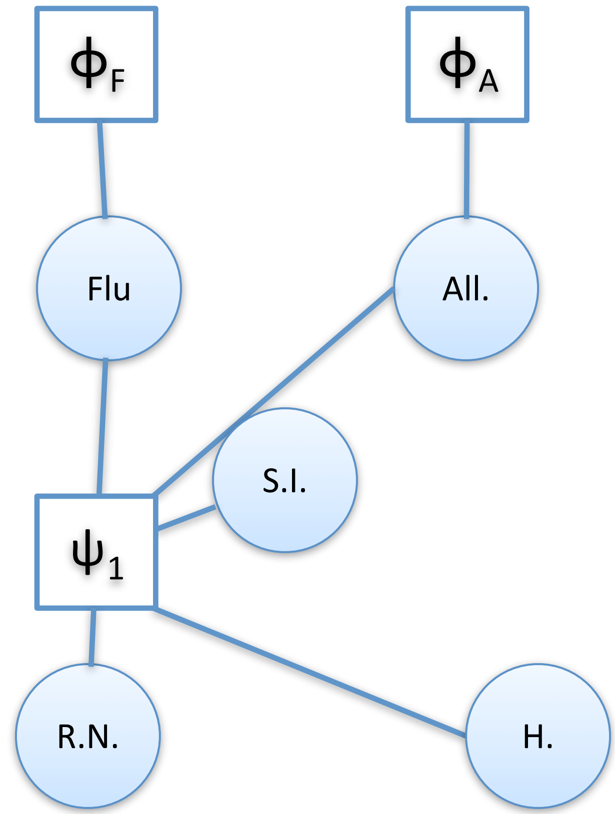
- $\psi_1 = \phi_{FAS} \cdot \phi_{SR} \cdot \phi_{SH}$



Example

- Eliminate S.

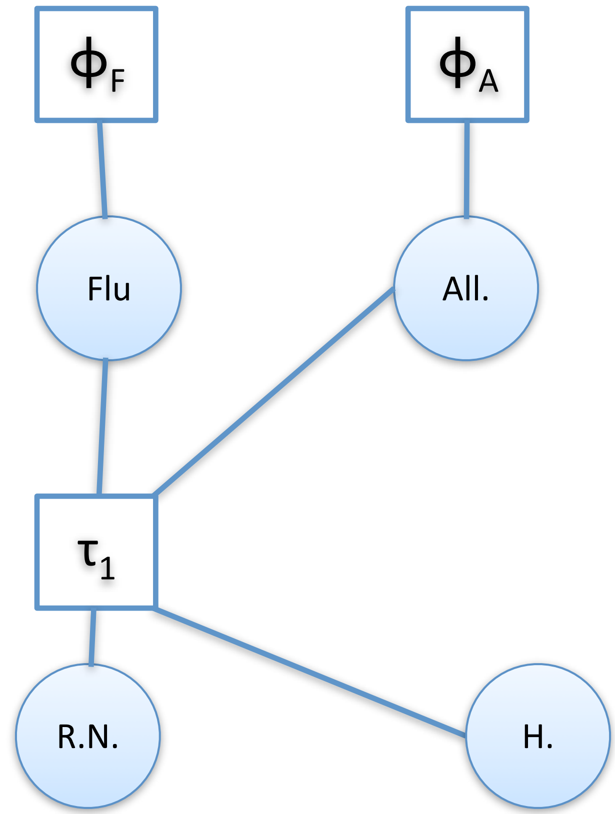
- $\psi_1 = \phi_{FAS} \cdot \phi_{SR} \cdot \phi_{SH}$
- $\tau_1 = \sum_S \psi_1$



Example

- Eliminate S.

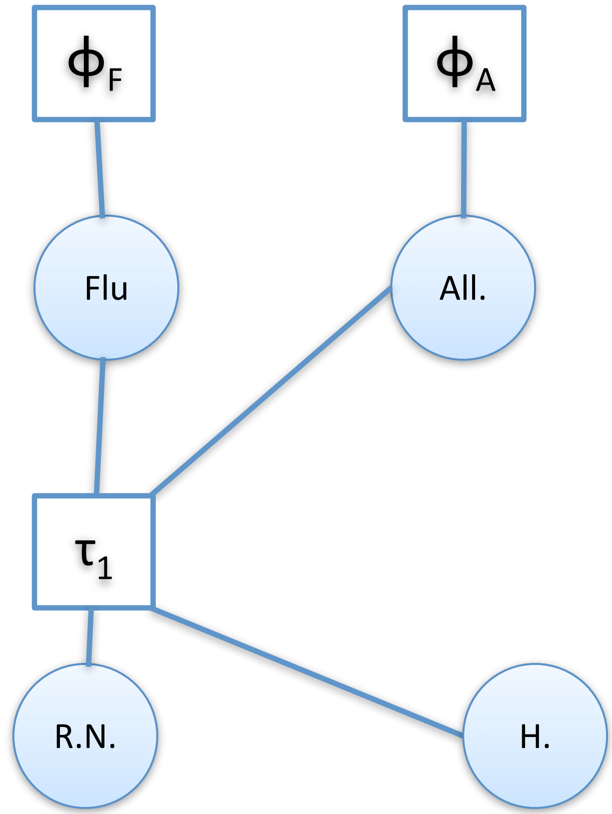
- $\psi_1 = \phi_{FAS} \cdot \phi_{SR} \cdot \phi_{SH}$
- $\tau_1 = \sum_S \psi_1$

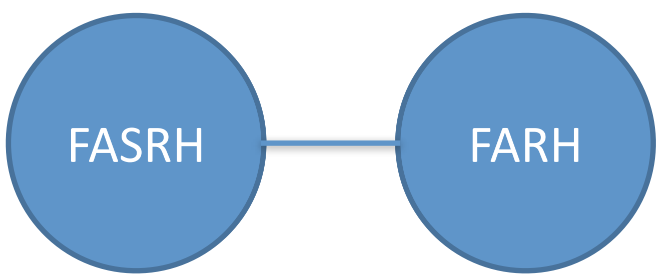


Example

- Eliminate R.

- $\psi_2 = \tau_1$
- $\tau_2 = \sum_R \psi_2$

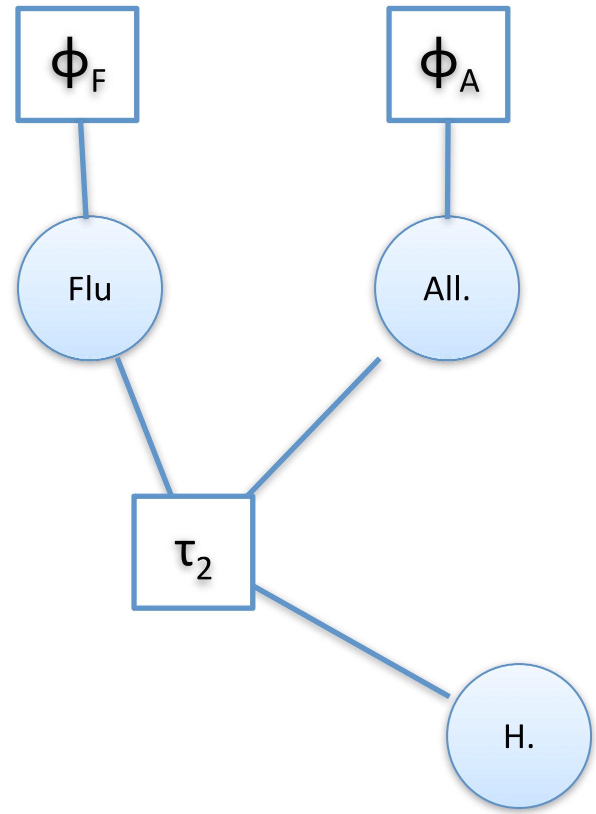


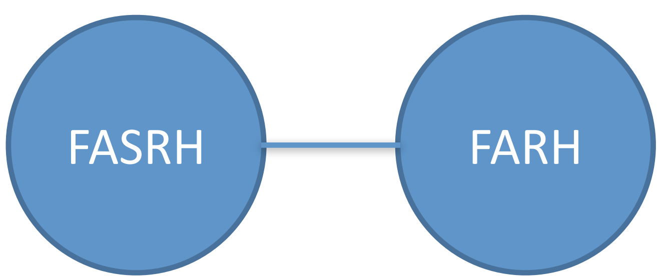


Example

- Eliminate R.

- $\psi_2 = \tau_1$
- $\tau_2 = \sum_R \psi_2$

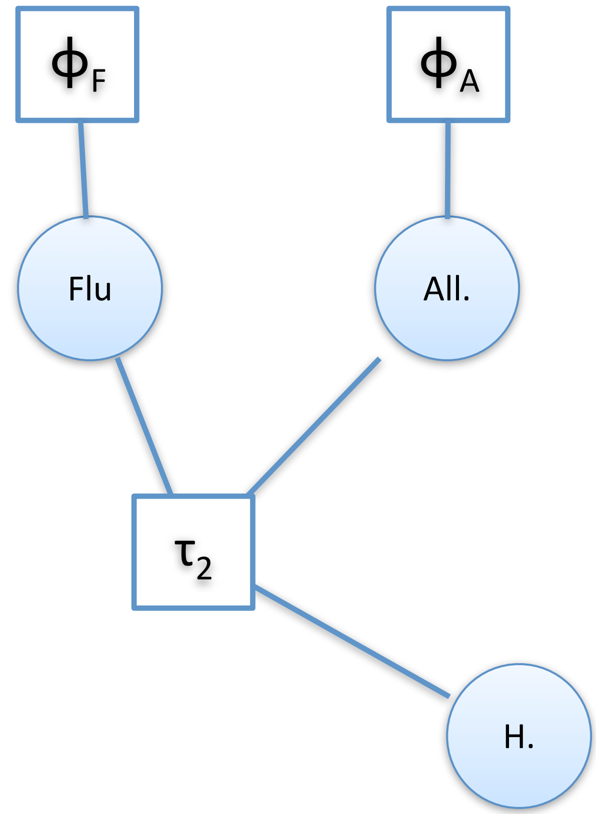




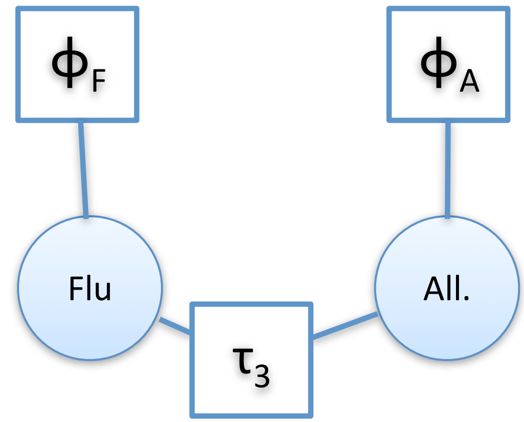
Example

- Eliminate H.

- $\psi_3 = \tau_2$
- $\tau_3 = \sum_H \psi_3$



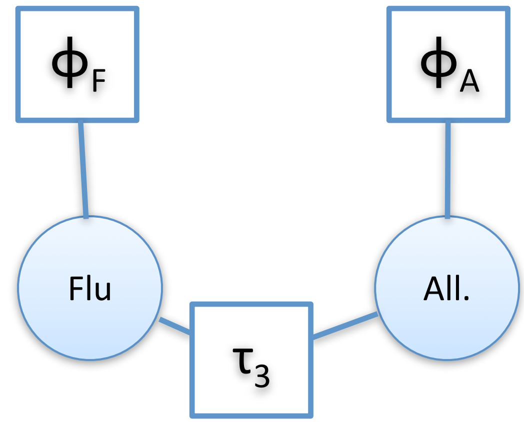
Example



- Eliminate H.

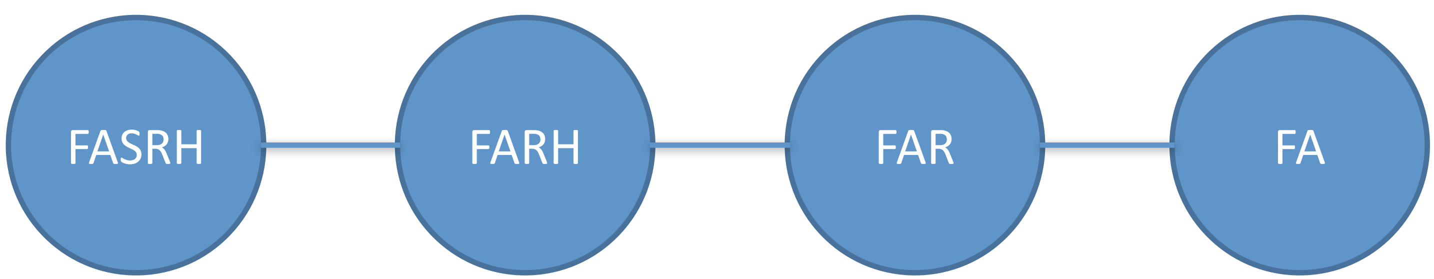
- $\psi_3 = \tau_2$
- $\tau_3 = \sum_H \psi_3$

Example



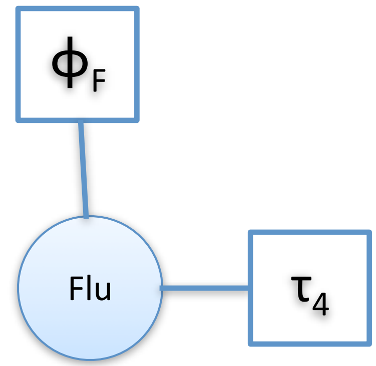
- Eliminate A.

- $\psi_4 = \tau_3\ \phi_A$
- $\tau_4 = \sum_A \psi_3$

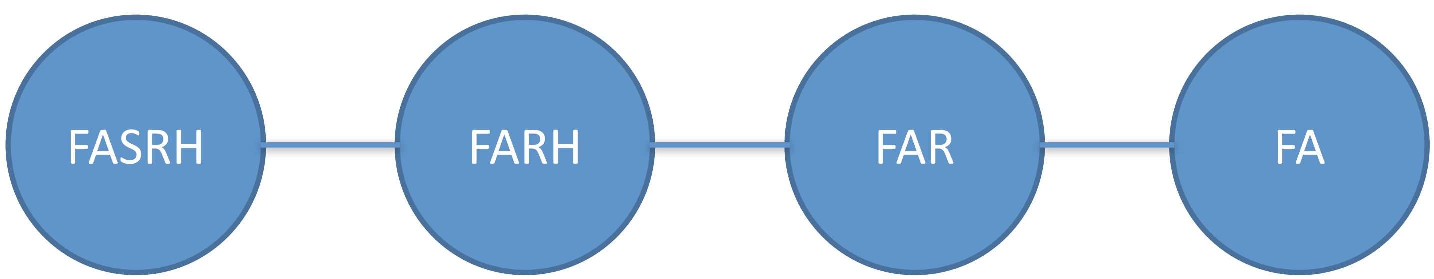


Example

- Eliminate A.

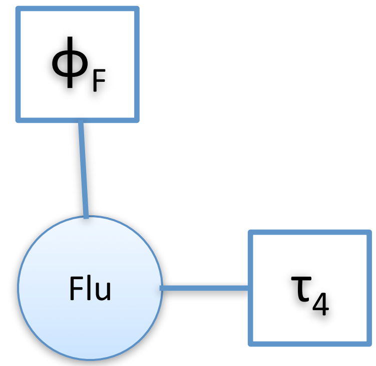


- $\psi_4 = \tau_3\ \phi_A$
- $\tau_4 = \sum_A \psi_3$

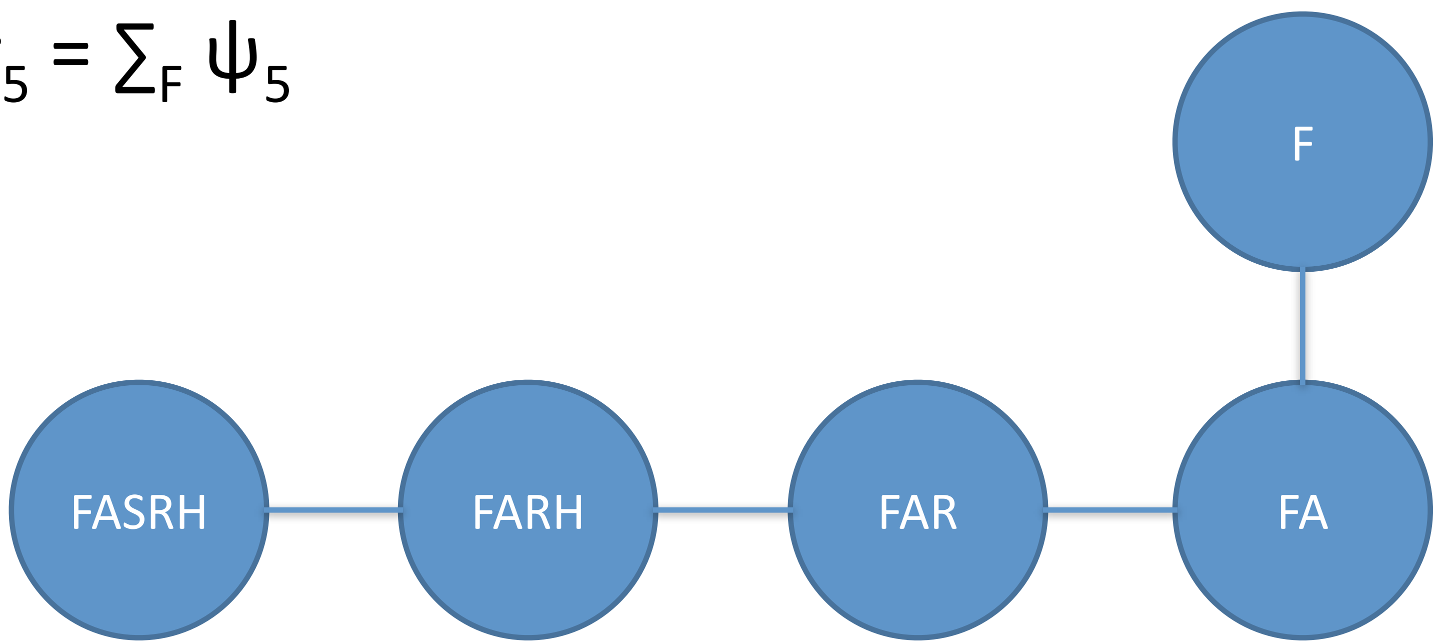


Example

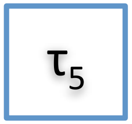
- Eliminate F.



- $\psi_5 = \tau_4 \, \phi_F$
- $\tau_5 = \sum_F \psi_5$

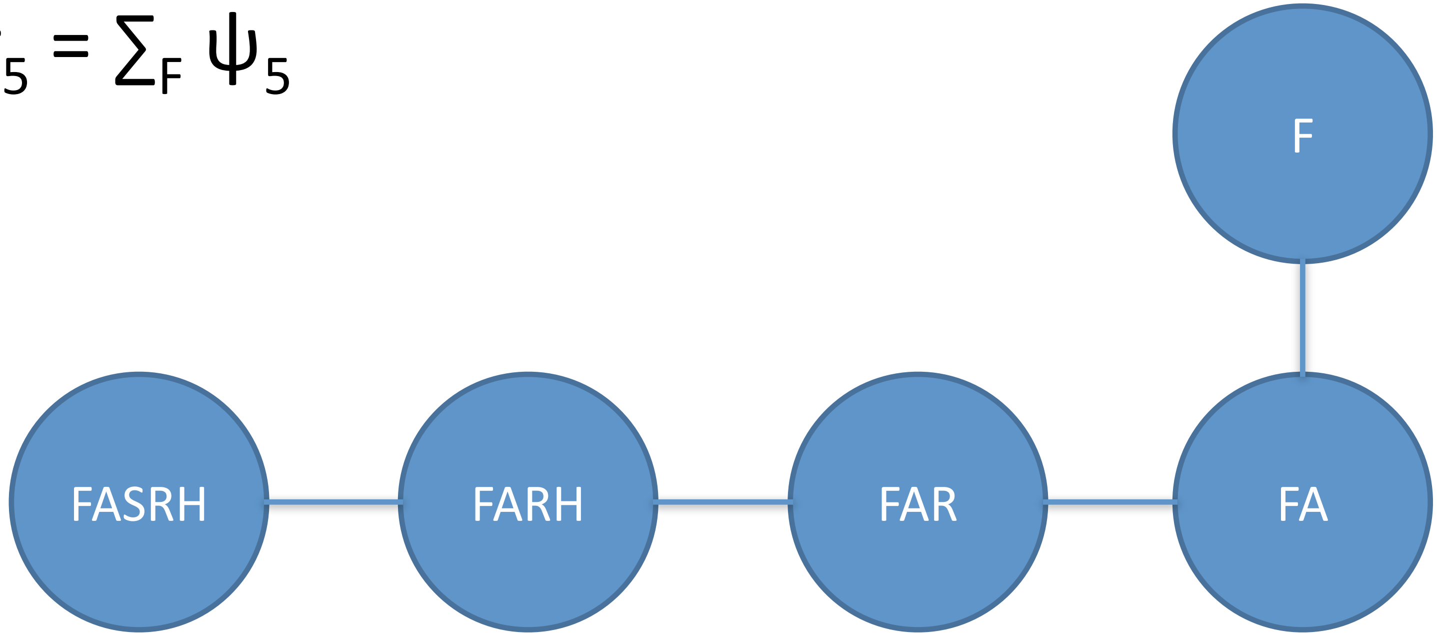


Example



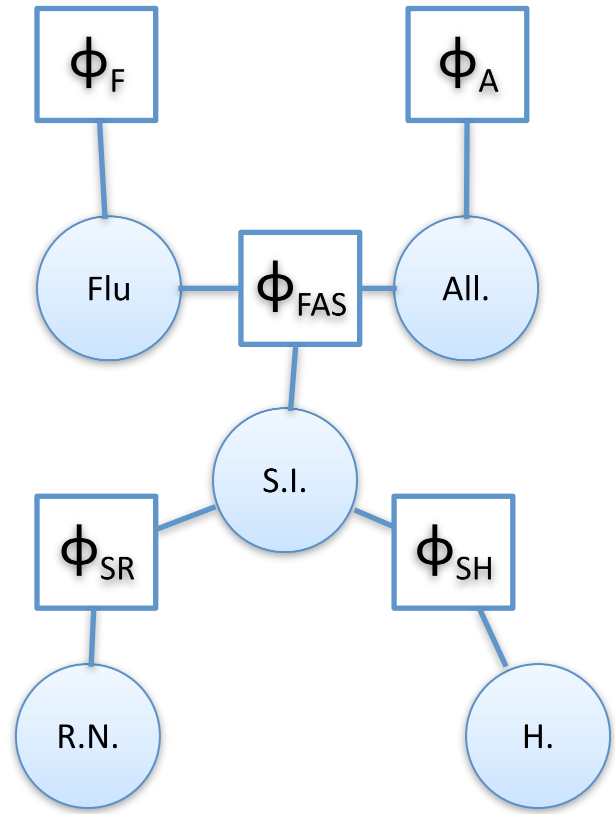
- Eliminate F.

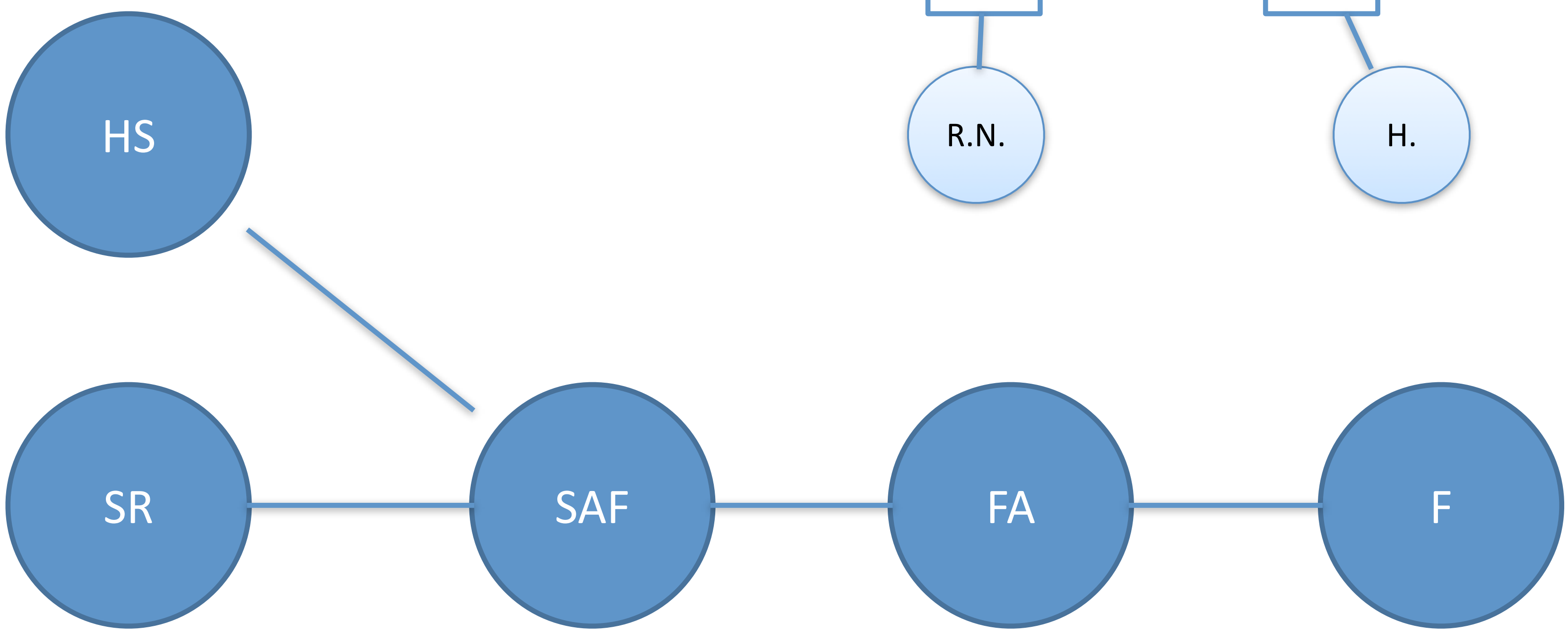
- $\psi_5 = \tau_4\, \phi_F$
- $\tau_5 = \sum_F \psi_5$



Alternative Ordering

- {H, R, S, A, F}





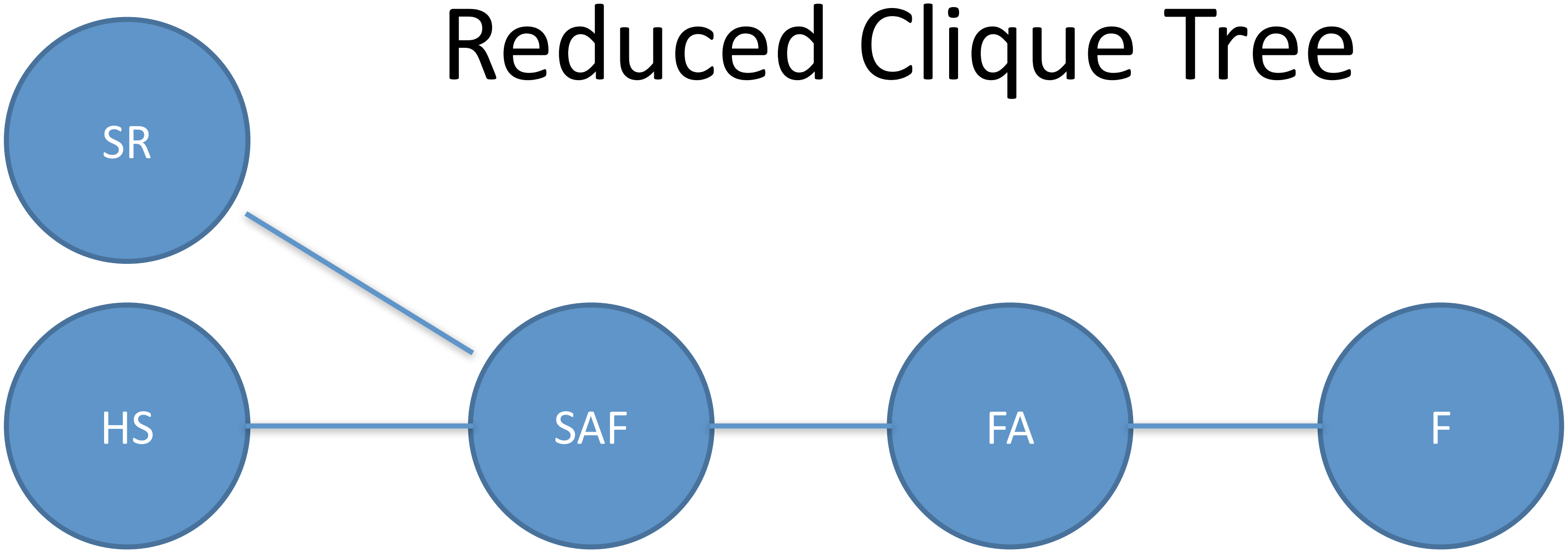
How to Obtain a Clique Tree
Option 1

- Start with factor graph or undirected graph.
- Do variable elimination.
- One node per $\mathbf{C}_i$, edge $\mathbf{C}_i$-$\mathbf{C}_j$ when τ_i gets used in computing ψ_j.
- Result is always a clique tree!
  - Running intersection property.
  - Different orderings will give different clique trees.

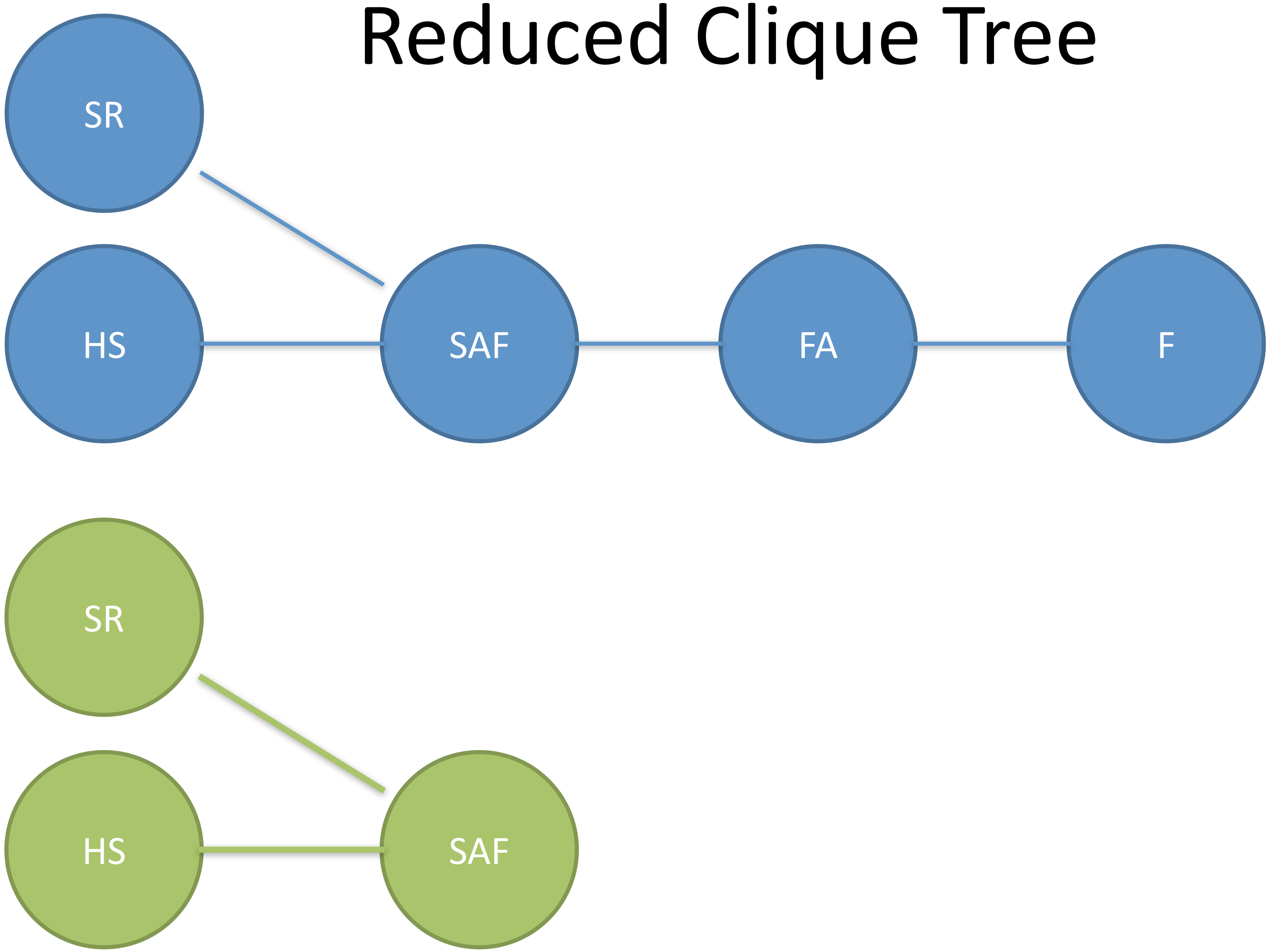
Reducing a Clique Tree

- Given a clique tree $\mathcal{T}$ for a set of factors Φ, there is always a reduced clique tree $\mathcal{T}'$ such that
  - all clique-vertices in $\mathcal{T}'$ are in $\mathcal{T}$
  - no clique vertex in $\mathcal{T}'$ is a subset of another.
- Construction exploits the running intersection property.

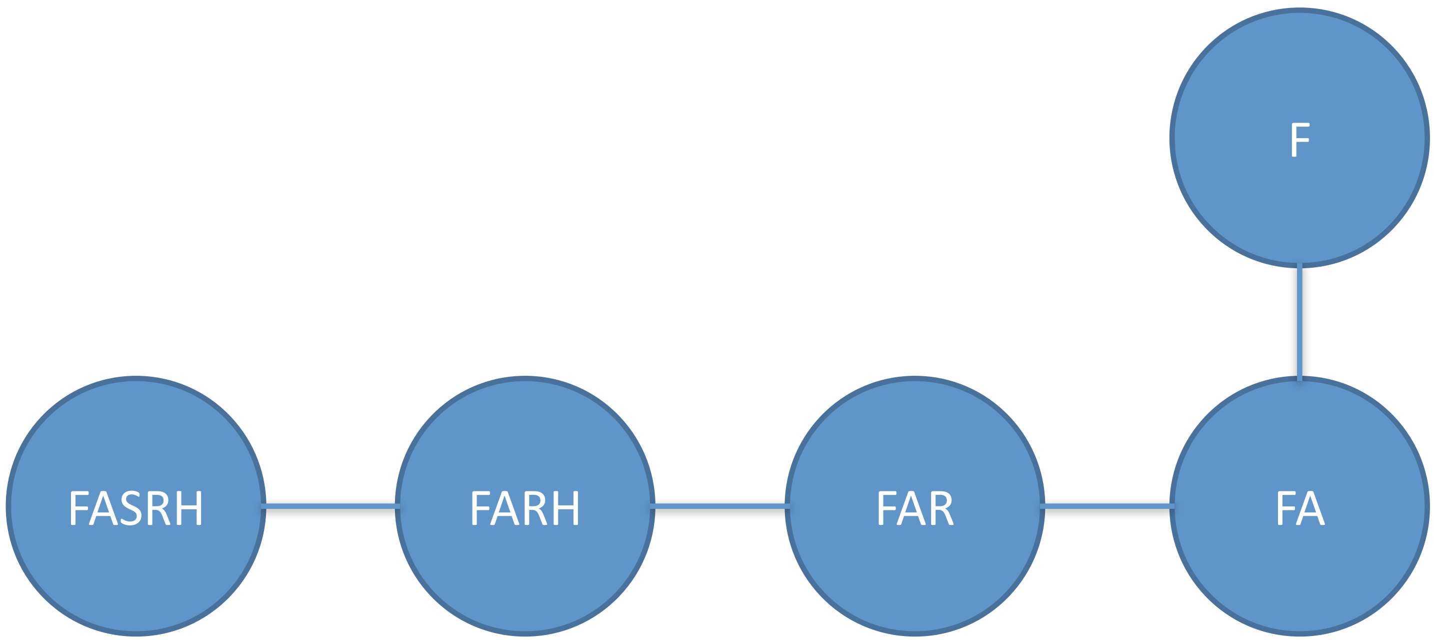
Reduced Clique Tree



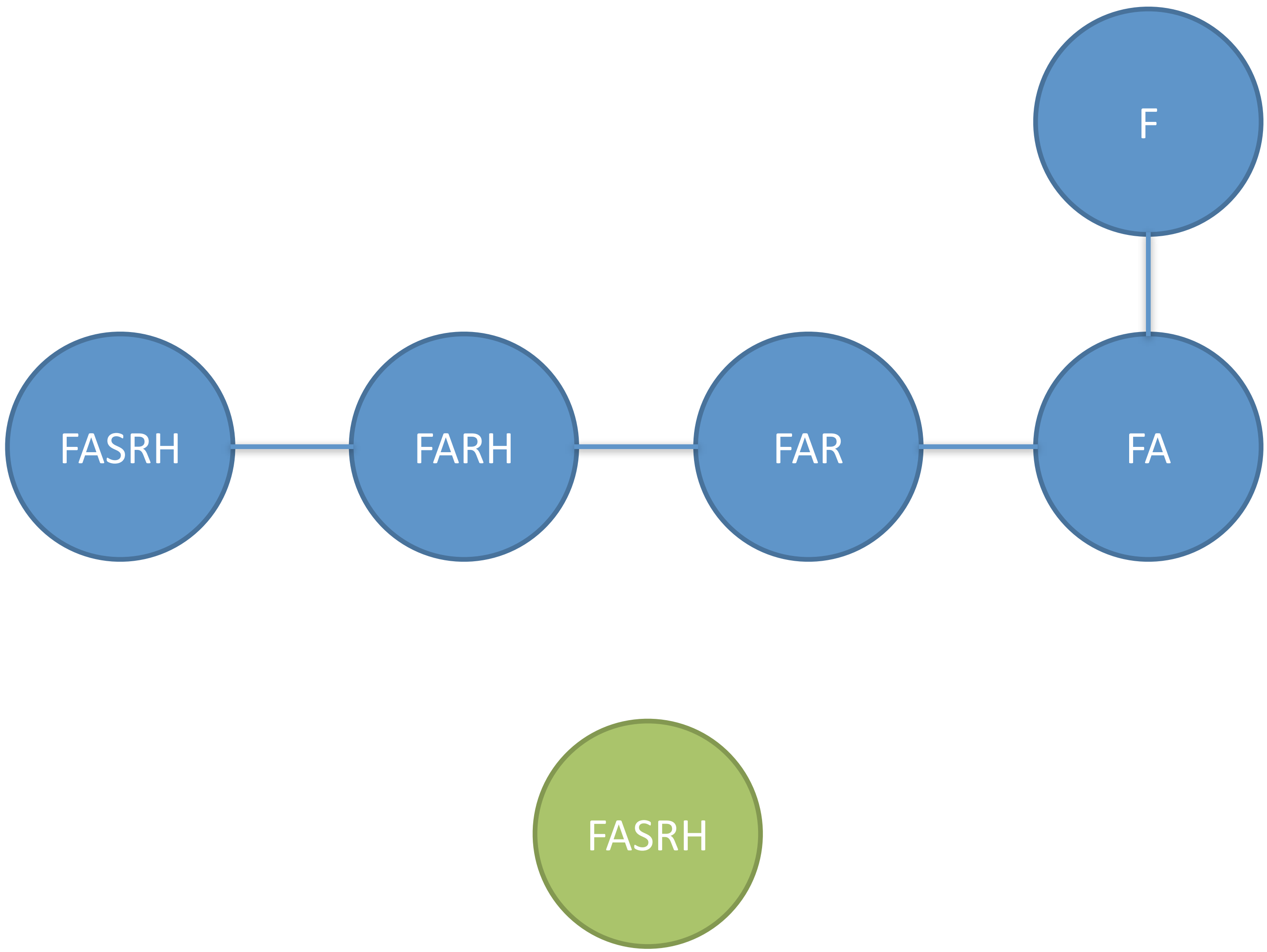
Reduced Clique Tree



Reduced Clique Tree



Reduced Clique Tree



How to Obtain a Clique Tree *Option 2*

1. Given factors, construct the undirected graph.
2. Triangulate to get a chordal graph.
3. Find maximal cliques in the chordal graph.
4. Construct a tree from the “cluster graph” of maximal cliques.

Actually the method we discussed in Lecture 5

Step 2: Triangulate

- NP-hard in general to get the smallest one.
- Use heuristics.

Step 3: Find Maximal Cliques

- Maximum cardinality search: gives an induced graph with no fill edges (assuming chordal graph as input, which we have).
- We proved last time that every maximal clique in the induced graph equates to the scope of an intermediate factor from VE.

- Other approaches exist.

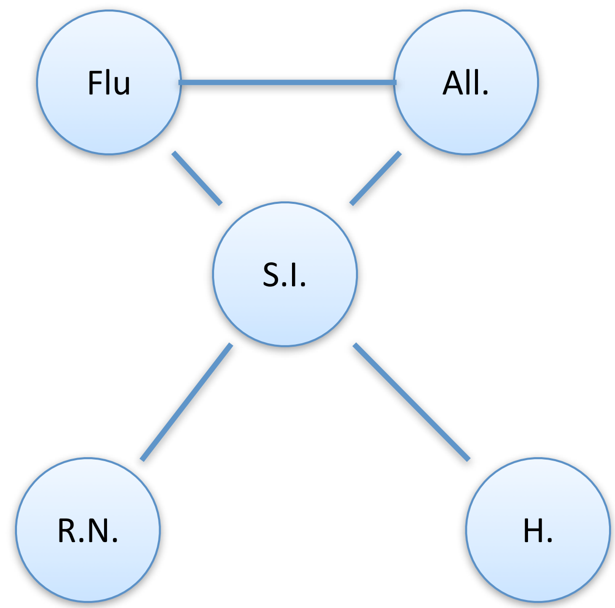
Maximum Cardinality Search for VE Ordering

- Start with undirected graph on **X**, all nodes unmarked.
- For i = |**X**| to 1:
  - Let Y be the unmarked variable in **X** with the largest number of *marked* neighbors
  - $\pi(Y) = i$
  - Mark Y.
- Eliminate using permutation π.

Lecture 10

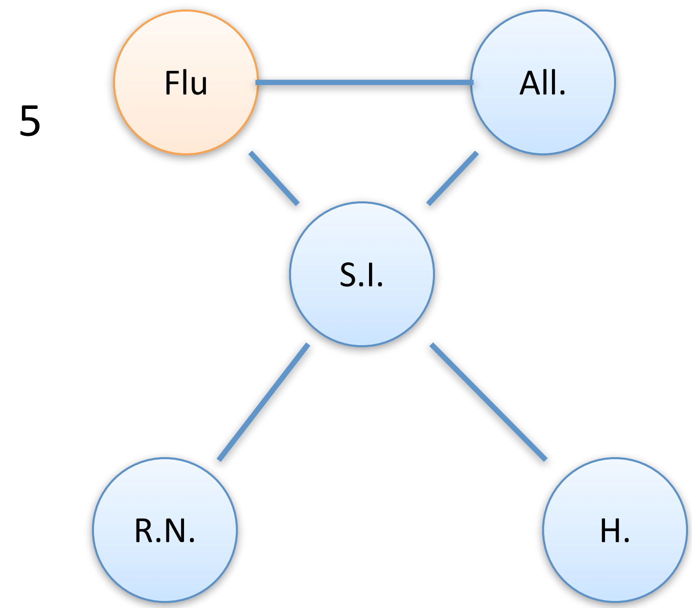
Example

- First node is arbitrary.



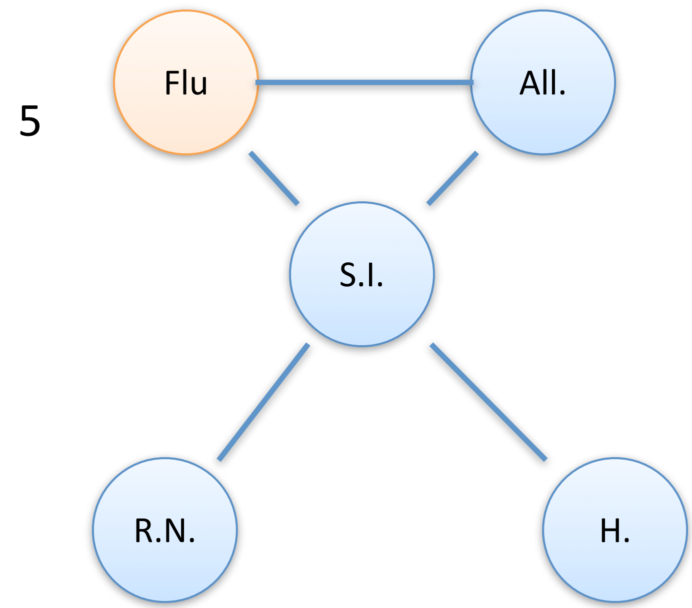
Example

- First node is arbitrary.



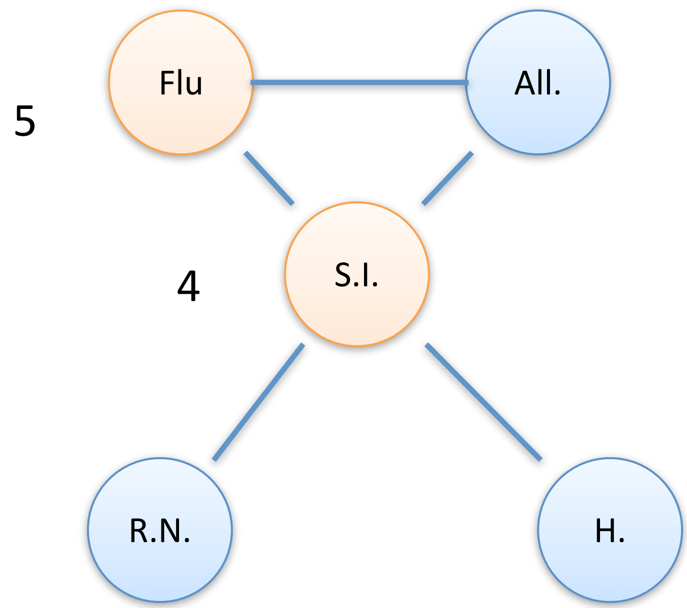
Example

- 1: A, S
- 0: R, H



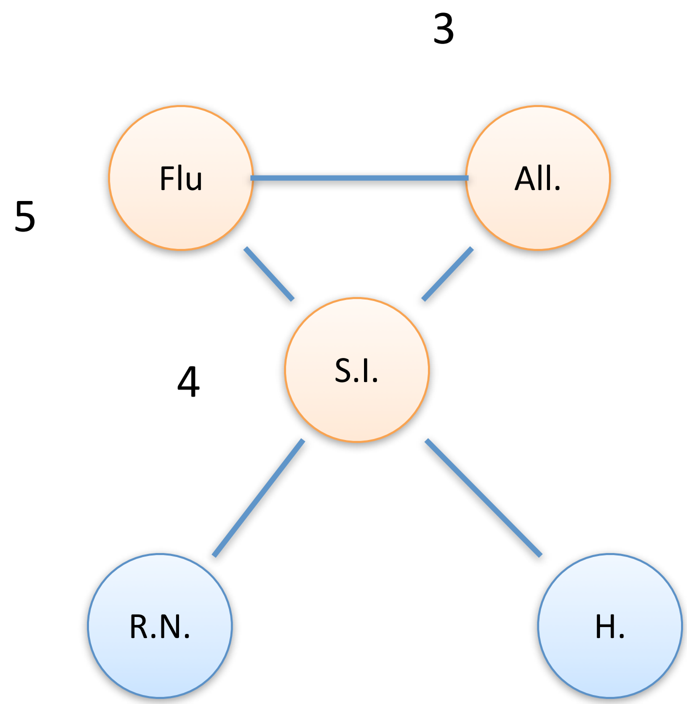
Example

- 2: A
- 1: R, H



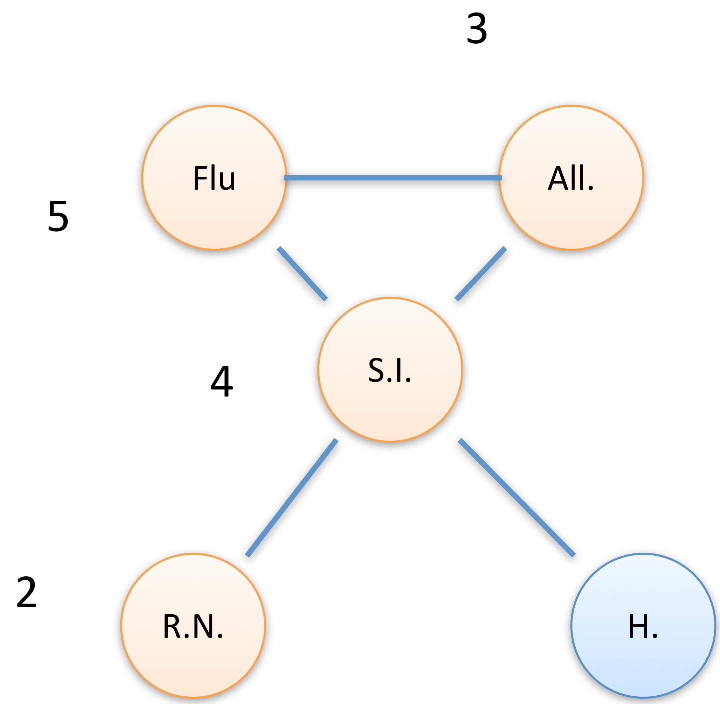
Example

- 1: R, H



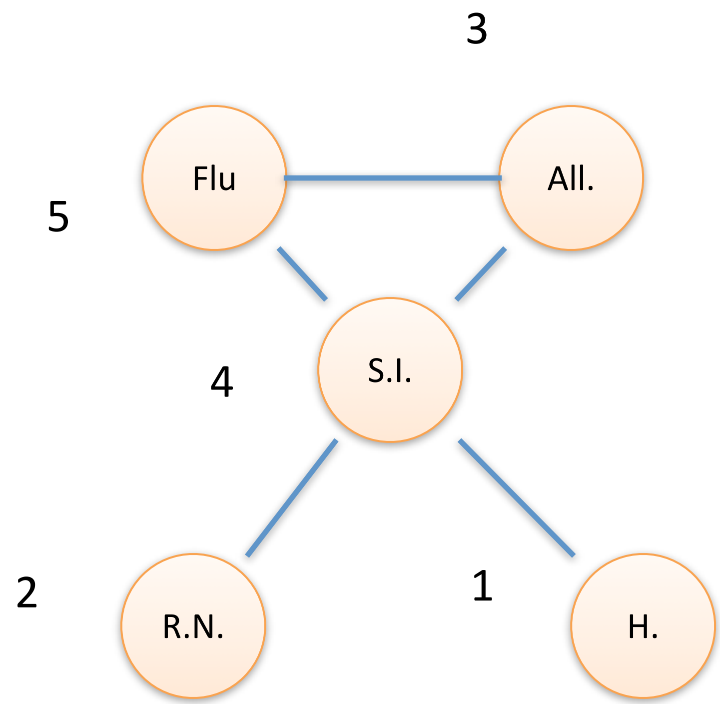
Example

- 1: H



Example

- Order: {H, R, A, S, F}

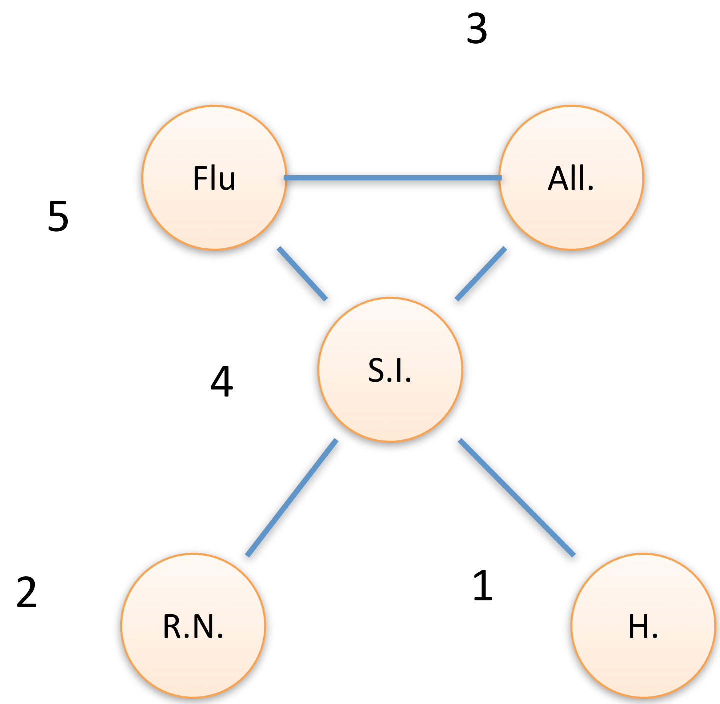


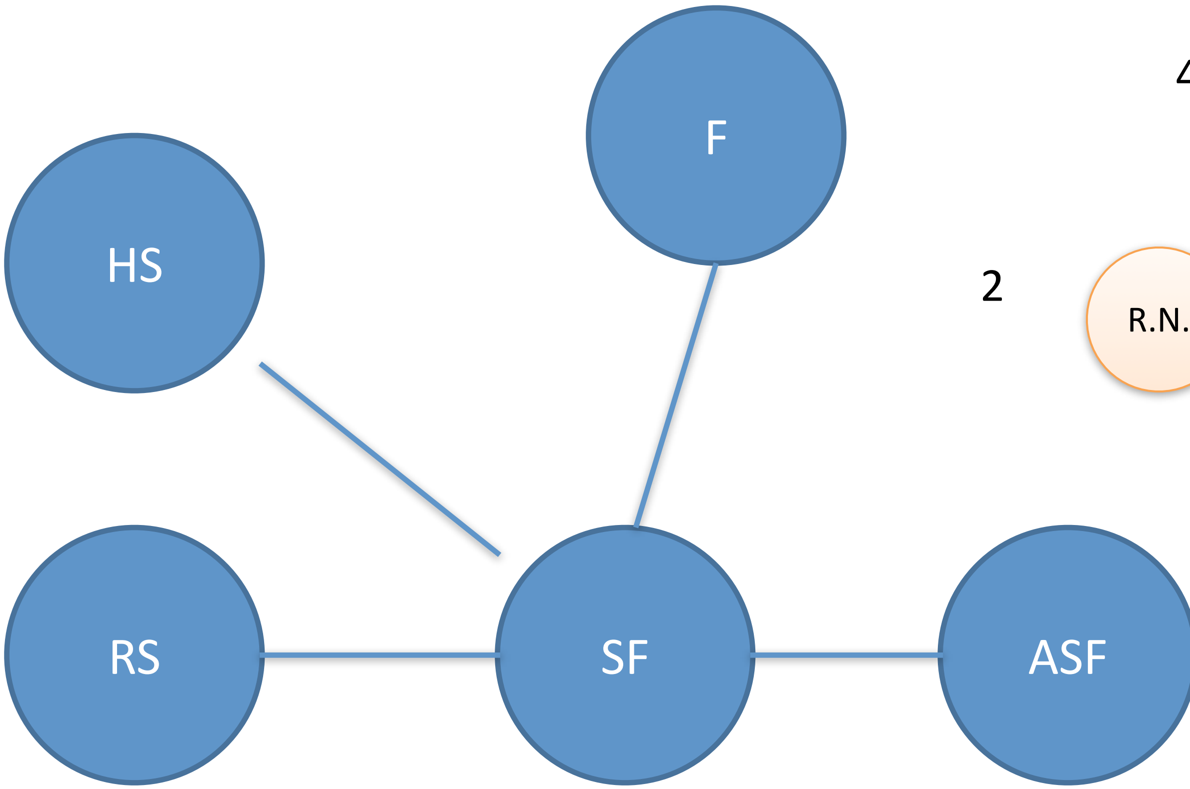
How to Obtain a Clique Tree
Option 2

1. Given factors, construct the undirected graph.
2. Triangulate to get a chordal graph.
3. Find maximal cliques in the chordal graph.
4. Construct a tree from the “cluster graph” of maximal cliques.
   – Results from VE with the ordering from maximum cardinality search; or use maximum spanning tree (edge weights = sepset sizes).

Example

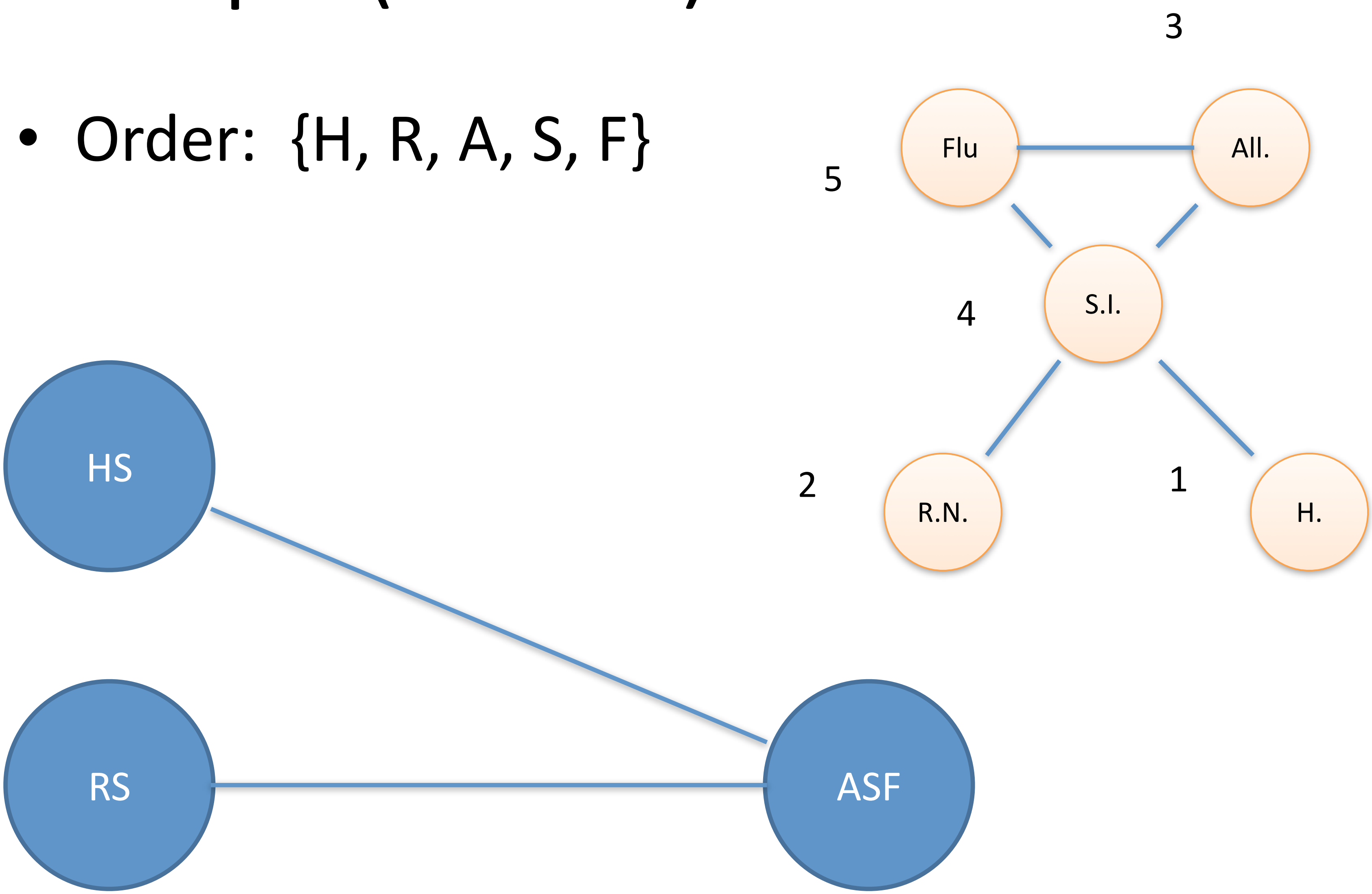
- Order: {H, R, A, S, F}





Example (Reduce)

- Order: {H, R, A, S, F}

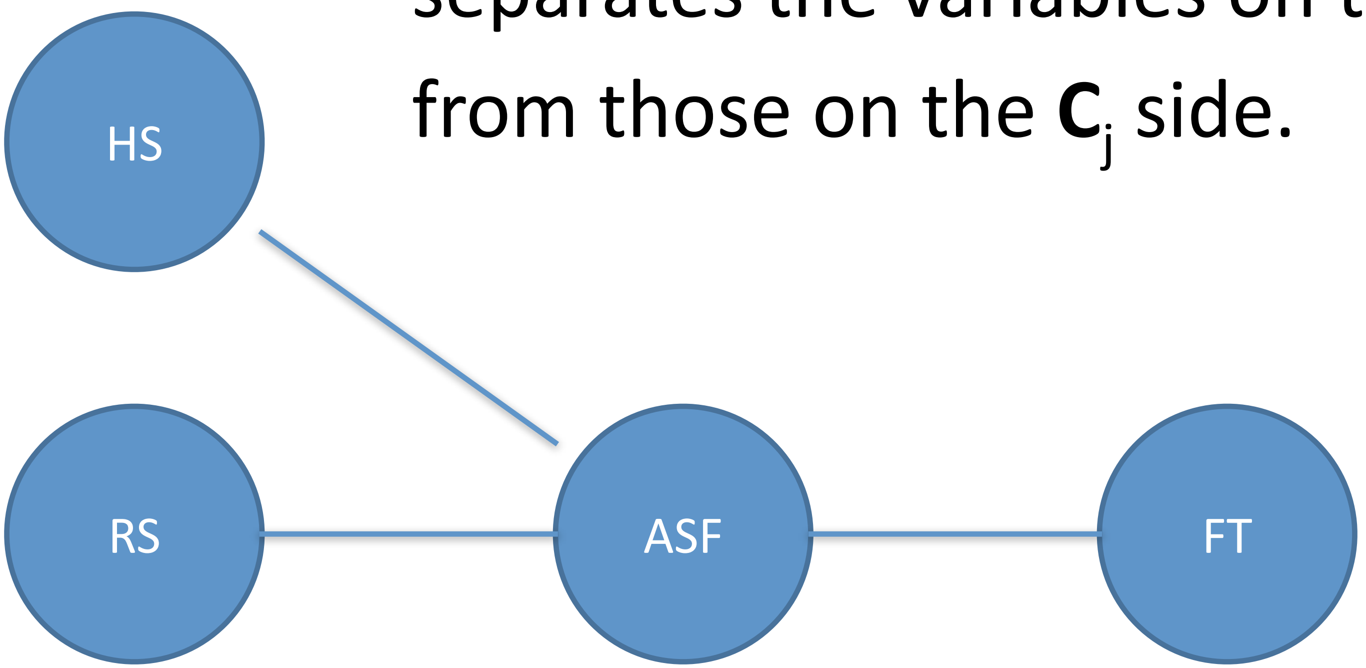


Now You Know …

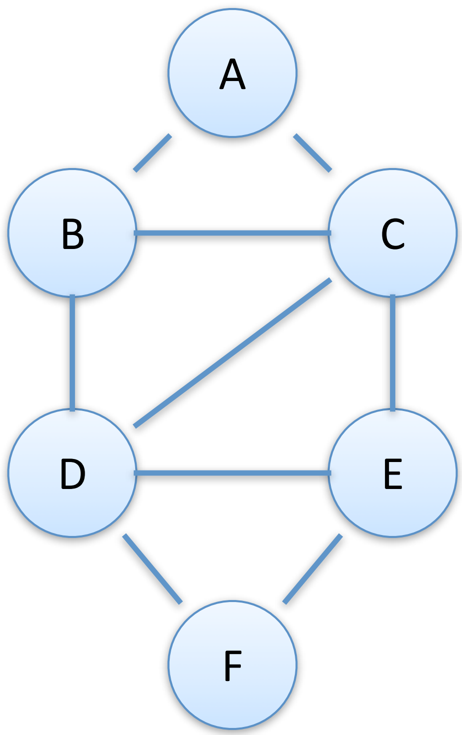
- How to construct a clique tree.
- Also called a “junction tree” or a “join tree.”

Running Intersection Property

- For connected clique-vertices $\mathbf{C}_i$ and $\mathbf{C}_j$, the sepset $\mathbf{S}_{i,j}$ is $\mathbf{C}_i \cap \mathbf{C}_j$, and it separates the variables on the $\mathbf{C}_i$ side from those on the $\mathbf{C}_j$ side.

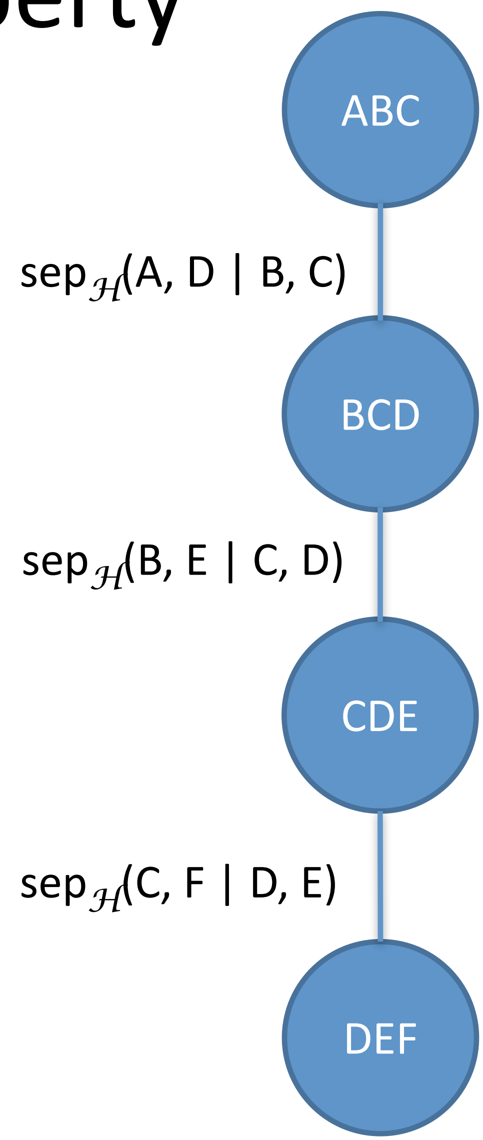


Running Intersection Property



Lecture 5

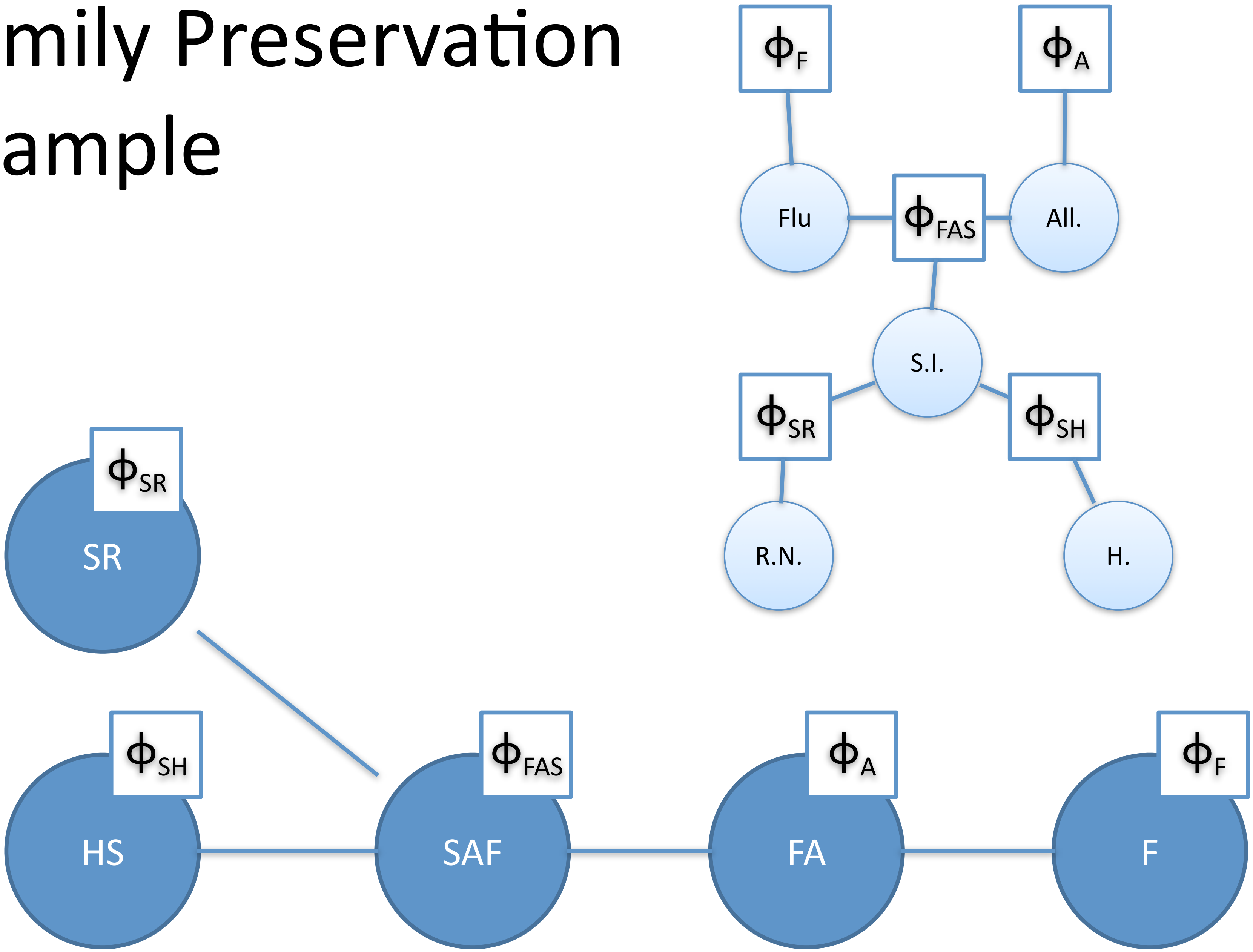
For each edge, intersection of r.v.s *separates* the rest in $\mathcal{H}$.



Family Preservation

- Every factor in the original graph is associated with one clique-vertex in the clique tree.

Family Preservation Example



Inference

- Last two lectures: variable elimination
  - Sum out variables one at a time.
- Today: alternative algorithms.
  - Also based on factors.
  - Central data structure: **clique tree**.

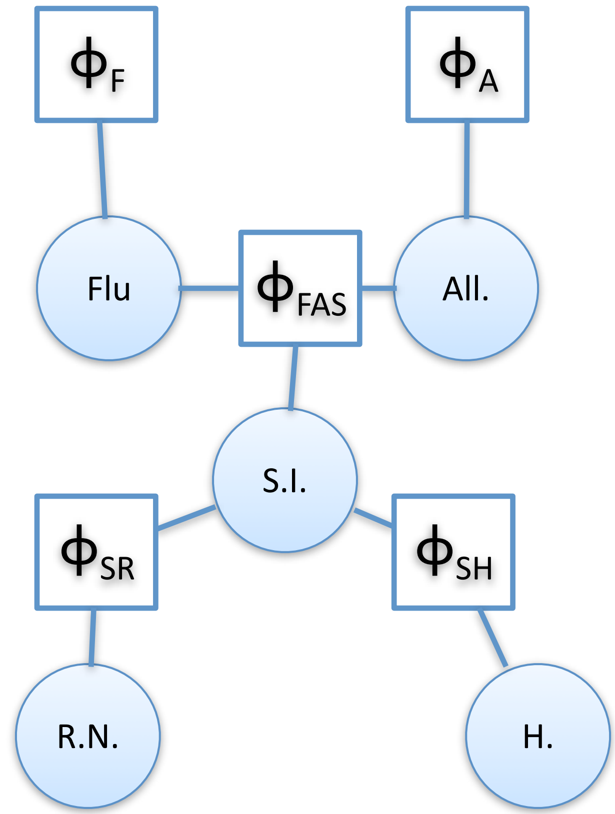
- What are these algorithms?

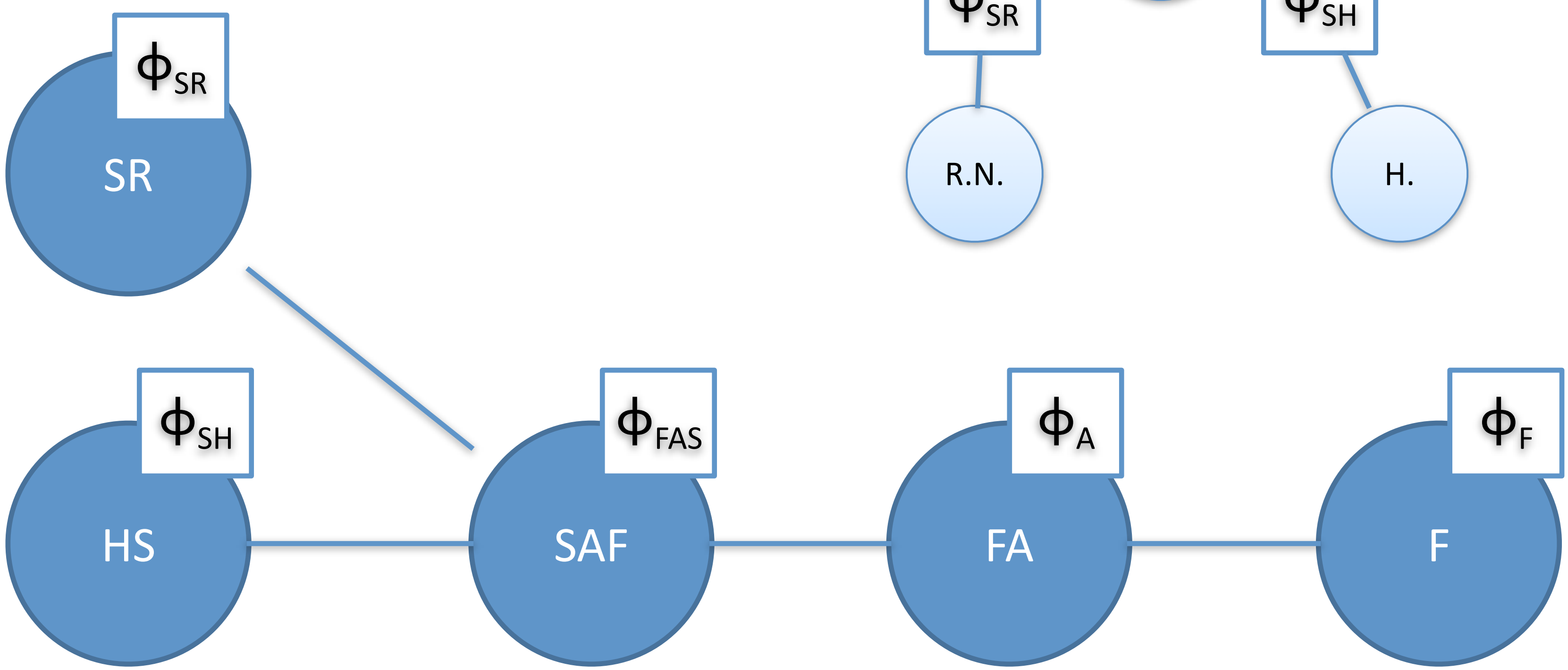
Clique Trees and Variable Elimination

- Each product-factor ψ belongs to a clique-vertex.
- Each τ is a "**message**"
  from one clique-vertex to another.

Example

- {H, R, S, A, F}

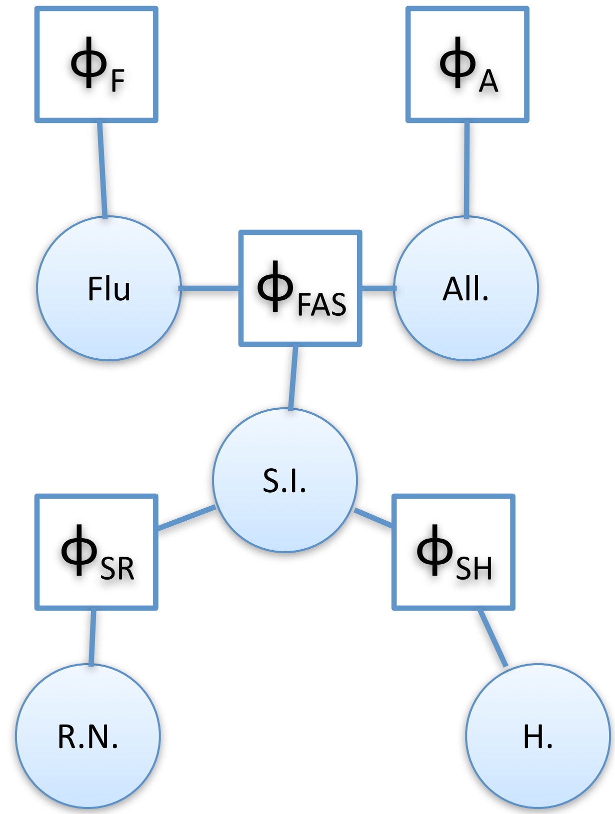


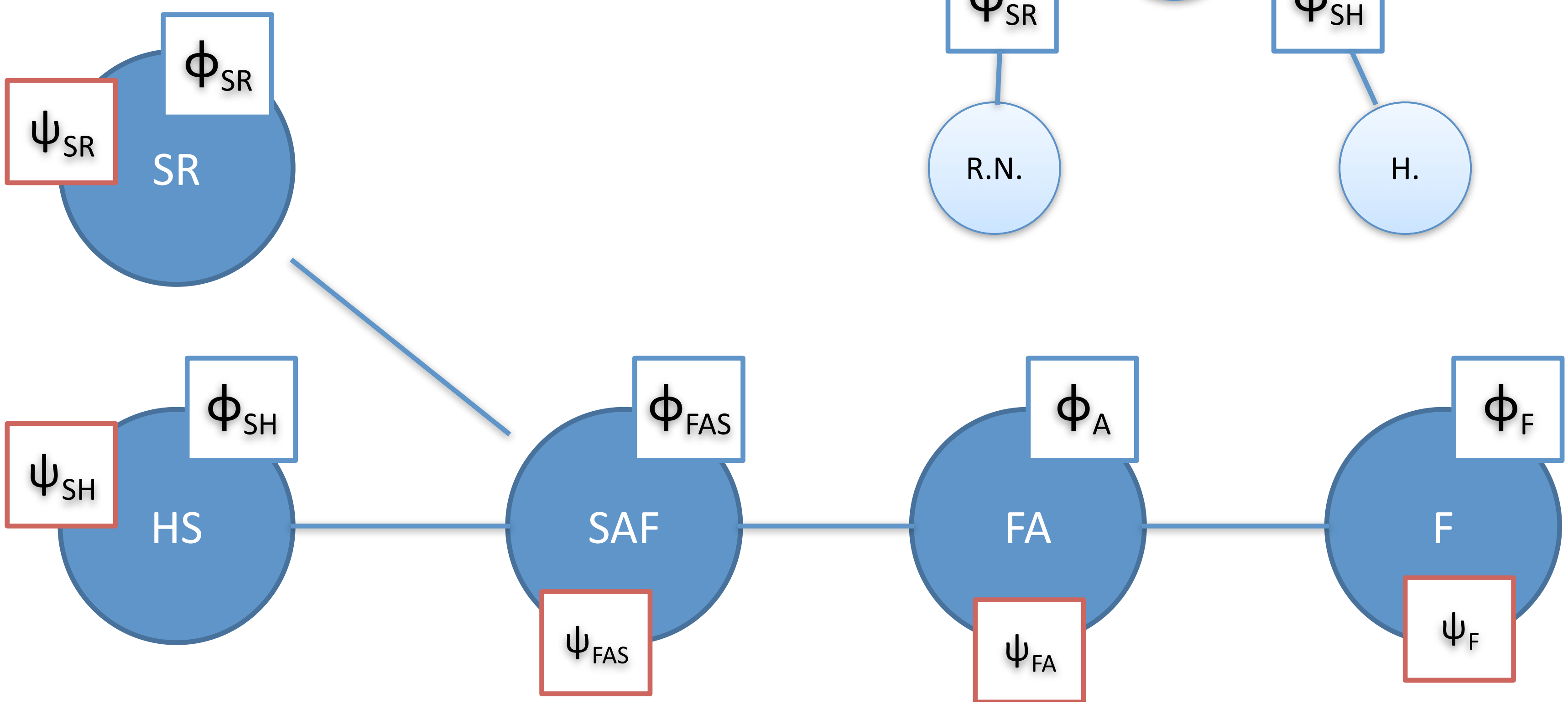


Just happens to be that in this graph with this ordering there is a one-to-one correspondence between original phis and product factors psi, because there was only one factor touching each variable-to-eliminate.

Example

- {H, R, S, A, F}





Messages

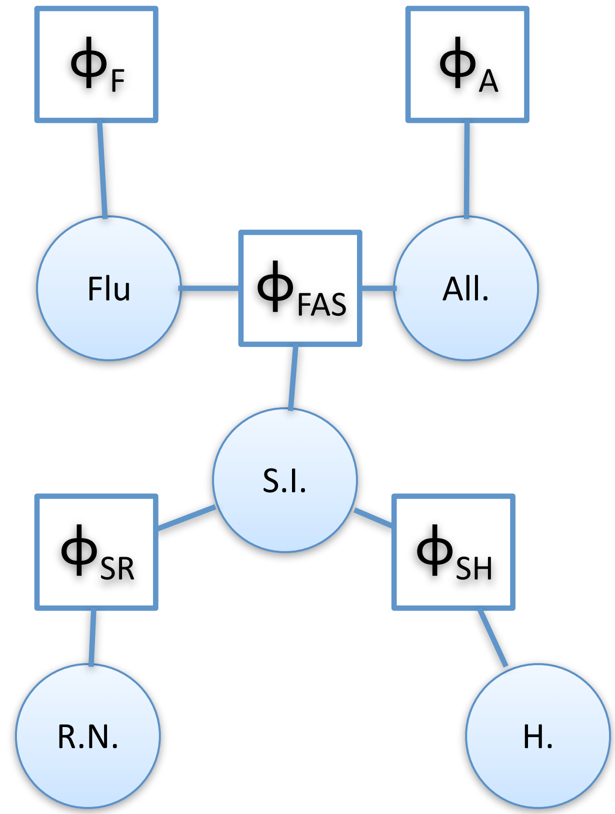
- Notice that the variable elimination ordering implies a direction to the edges.

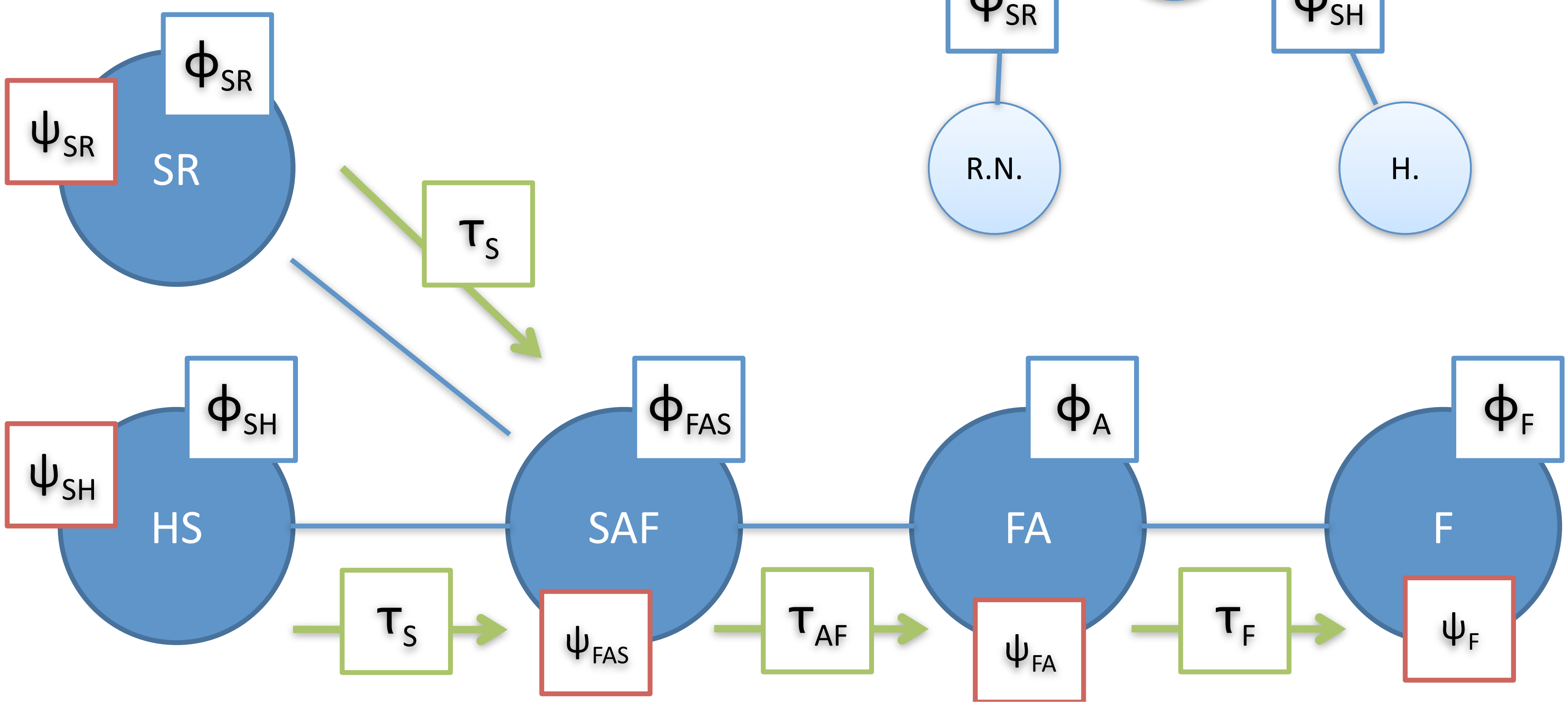
- Messages go "upstream."

- This leads to a clique-tree-centric view of VE, in which the same calculations happen as in VE.

Example

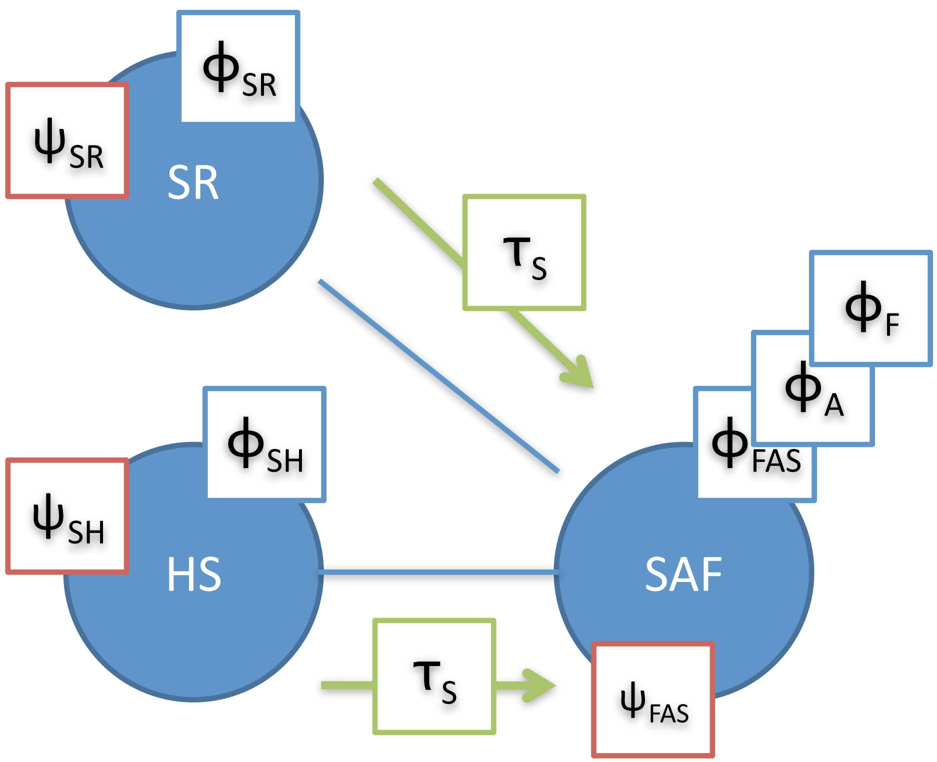
- {H, R, S, A, F}

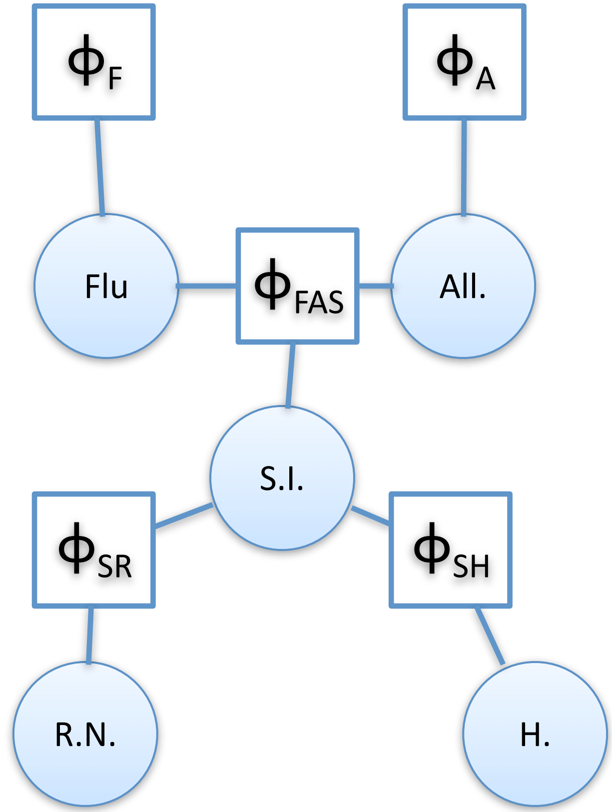




Example

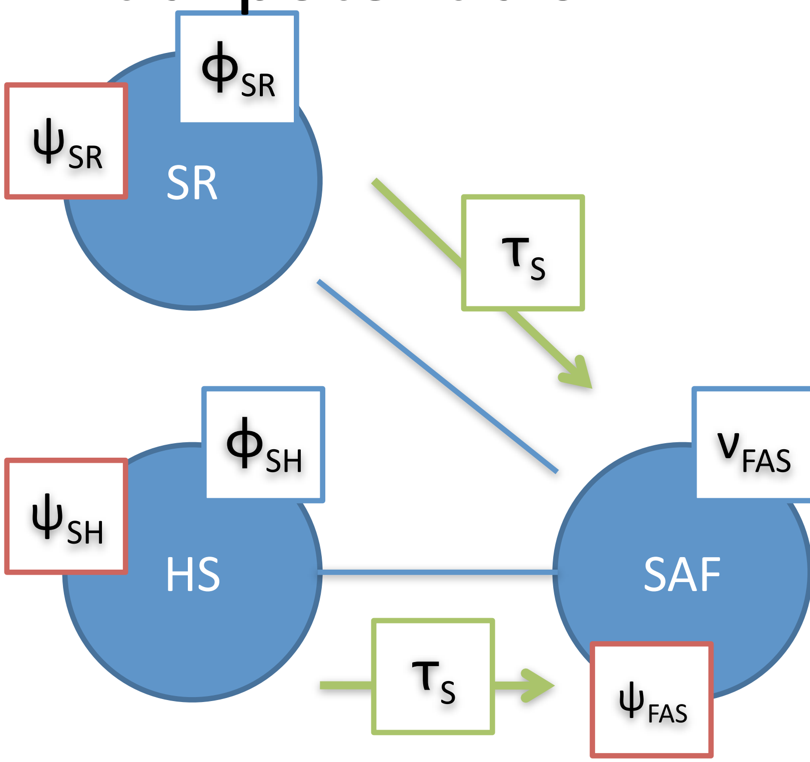
- {H, R, S, A, F}
- Reduced version

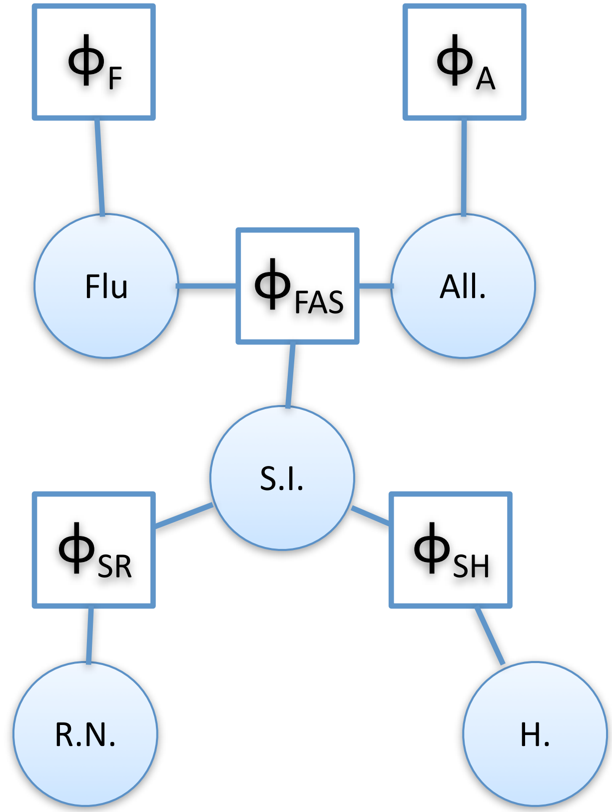




Example

- {H, R, S, A, F}
- Reduced version
- Initial potentials





$$\nu_{\mathrm{FAS}} = \phi_{\mathrm{FAS}}\phi_{\mathrm{A}}\phi_{\mathrm{F}}$$

Go through math that gets done for VE here.

Message Passing in the Clique Tree

- Calculate **initial potentials** for each clique-vertex.

$$\nu_j(\boldsymbol{C}_j) = \prod_{\phi \in \Phi_j} \phi$$

- Choose an arbitrary **root** clique $\mathbf{C}_r$. This imposes a directionality on the graph (think of the root as a *sink*).

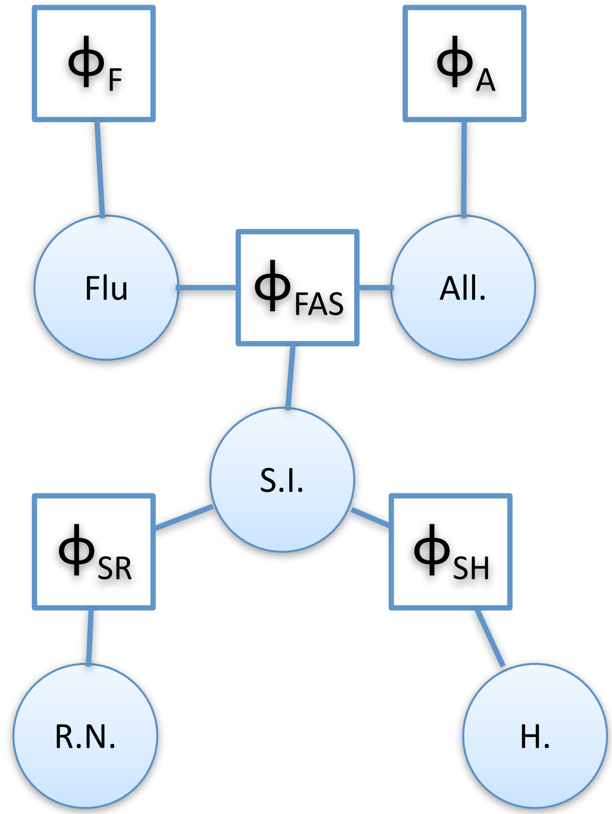
Ordering can be different from the VE ordering that might have created the clique tree.

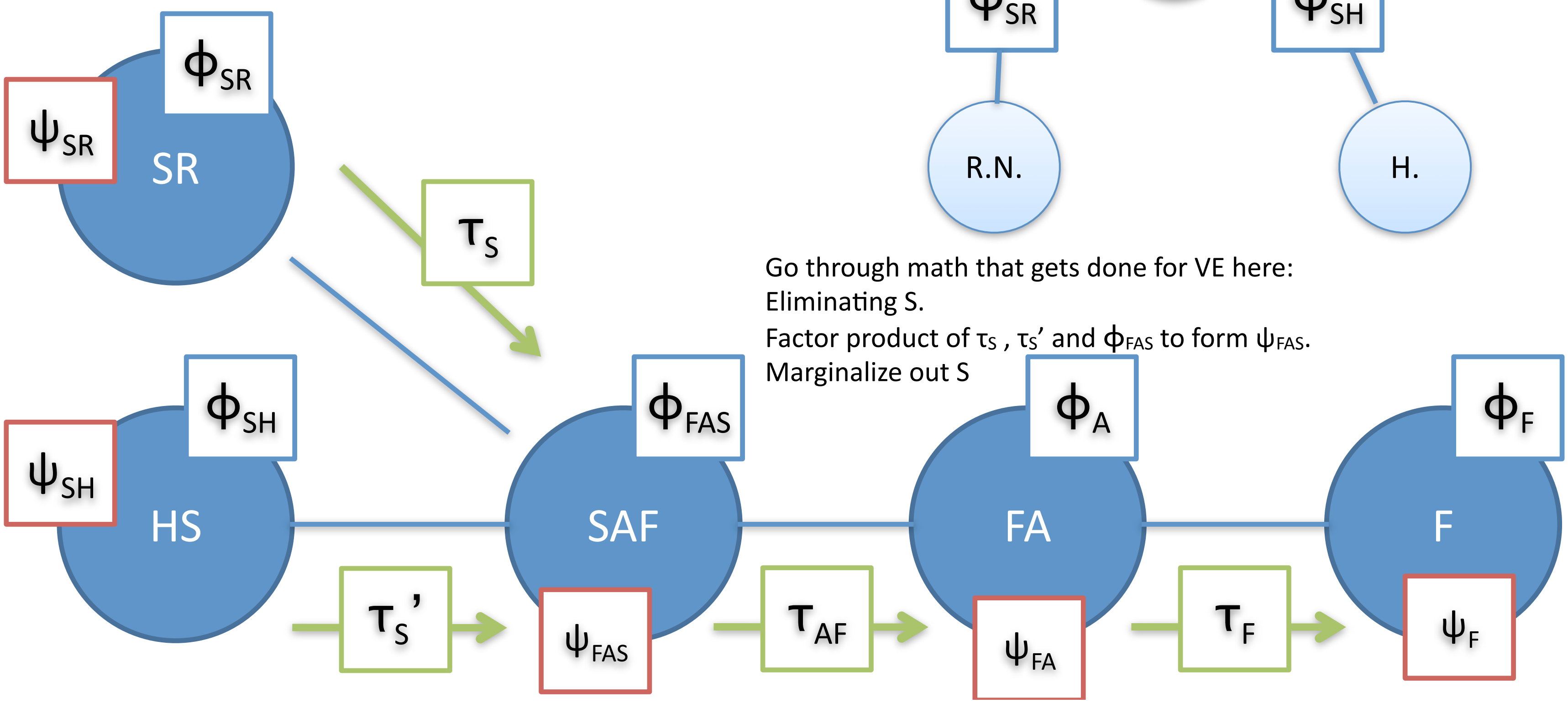
- Pass messages "upstream":

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

Example

- {H, R, S, A, F}





Go through math that gets done for VE here:
Eliminating S.
Factor product of τ_S , τ_S' and ϕ_{FAS} to form ψ_{FAS}.
Marginalize out S

Message Passing in the Clique Tree

sum out variables that are not involved in $\mathbf{C}_j$

original factors involving the family of $\mathbf{C}_i$

incoming messages from *other* neighbors of i

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

Message Passing in the Clique Tree

τ

ψ

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

Message Passing in the Clique Tree

- Notice that we have a recursive structure here: message-sum of products of message-sums of products of message-sums of …

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

Message Passing in the Clique Tree

- Notice that you have to work "upward."
- Root will be last.

$$\delta_{i \to j} \quad = \quad \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \to i}$$

Message Passing in the Clique Tree

- Notice that you have to work "upward."
- Root r will be last.

Unnormalized marginal distribution of root variables in C_i

Product of initial potentials

Messages into root

$$\beta_r = \nu_r \prod_{k \in \text{Neighbors}_r} \delta_{k \to r}$$

$$= \sum_{\boldsymbol{X} \setminus \boldsymbol{C}_i} \prod_{\phi \in \Phi} \phi$$

$$= Z \cdot P(\boldsymbol{C}_r)$$

The Algorithm

- Input: clique tree $\mathcal{T}$, factors $\mathbf{\Phi}$, root $\mathbf{C}_r$
- For each clique $\mathbf{C}_i$, calculate ν_i
- While $\mathbf{C}_r$ is still waiting on incoming messages:
  - Choose a $\mathbf{C}_i$ that *has* received all of its incoming messages.
  - Calculate and send the message from $\mathbf{C}_i$ to $\mathbf{C}_{\text{upstream-neighbor}(i)}$
- Return β_r

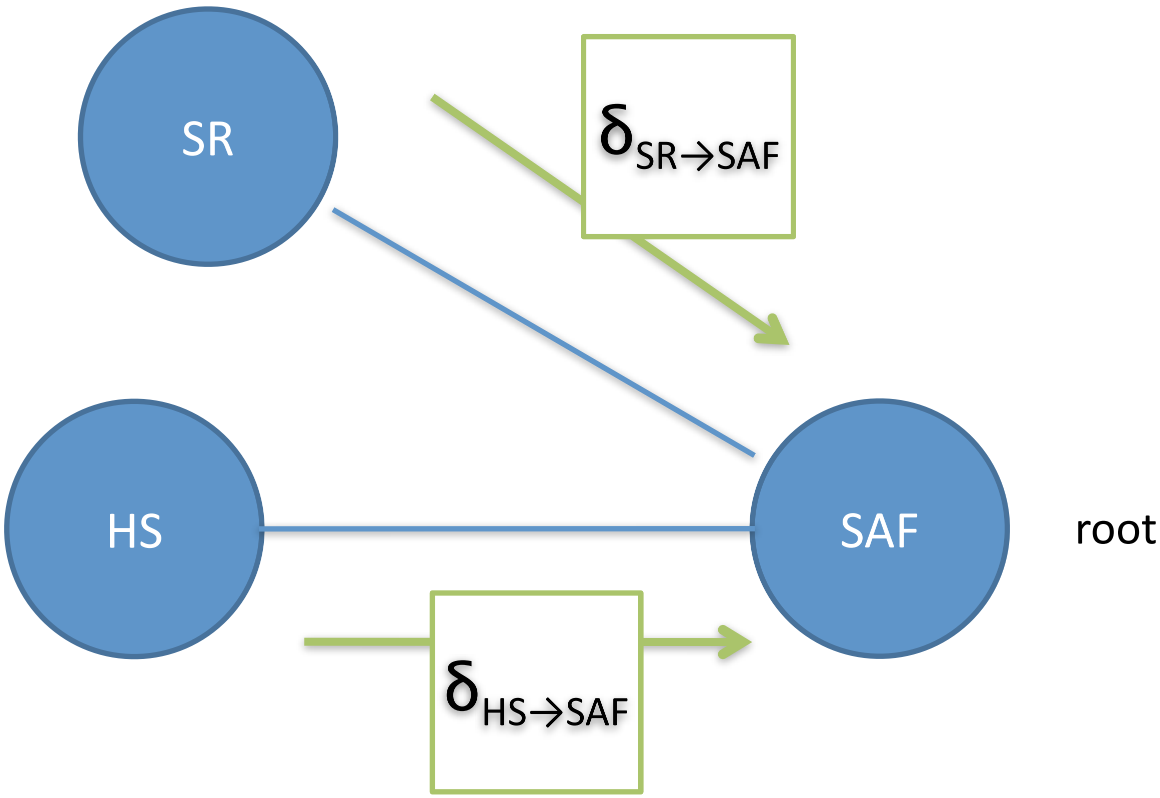
Obtaining Marginals

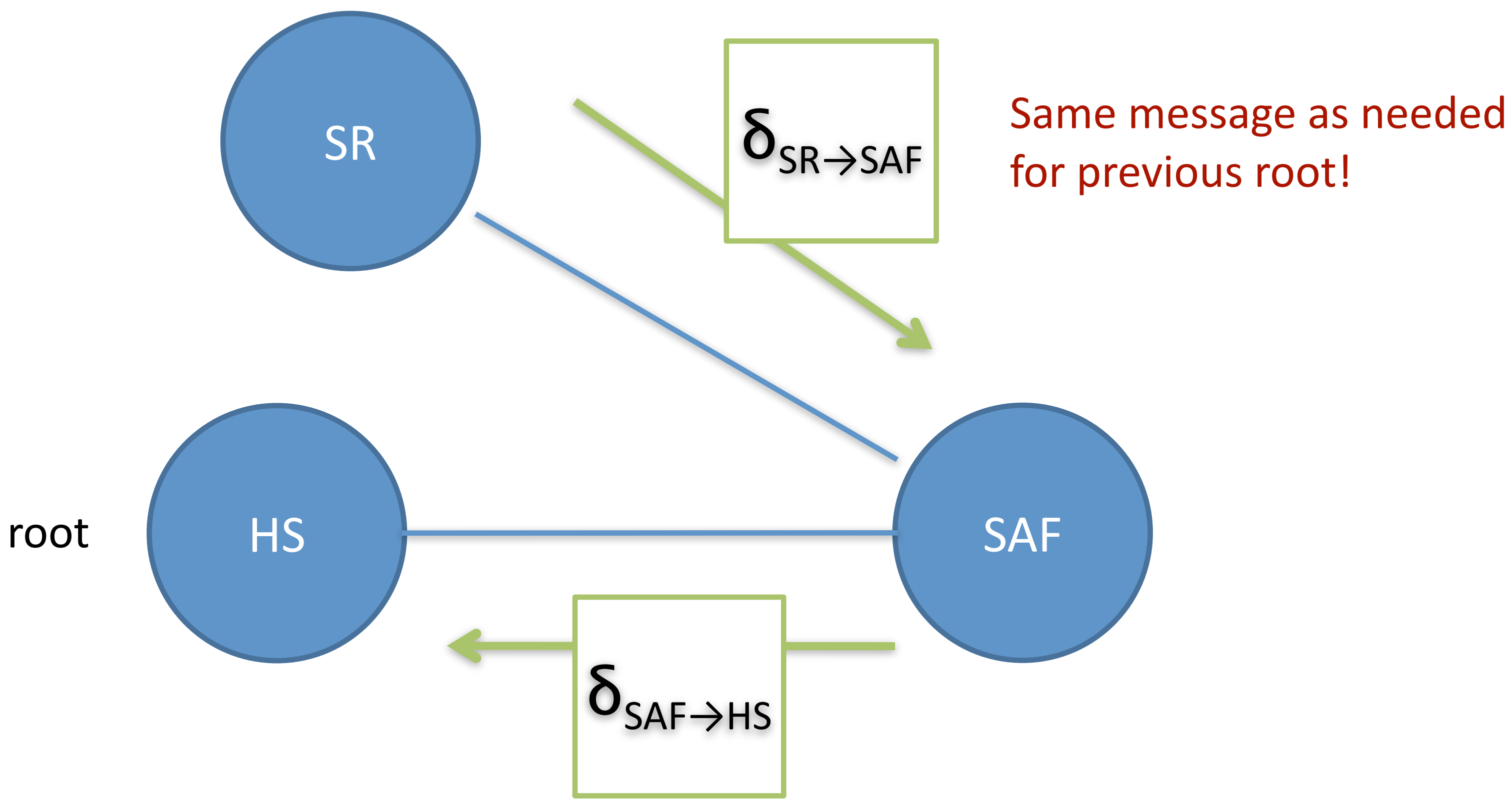
- Given a set of query variables **Q** that is contained in some clique **C**:
  - Make that clique the root.
  - Run the upward pass of clique tree VE, resulting in factor β.
  - Return $\sum_{\mathbf{C} \setminus \mathbf{Q}} \beta$.

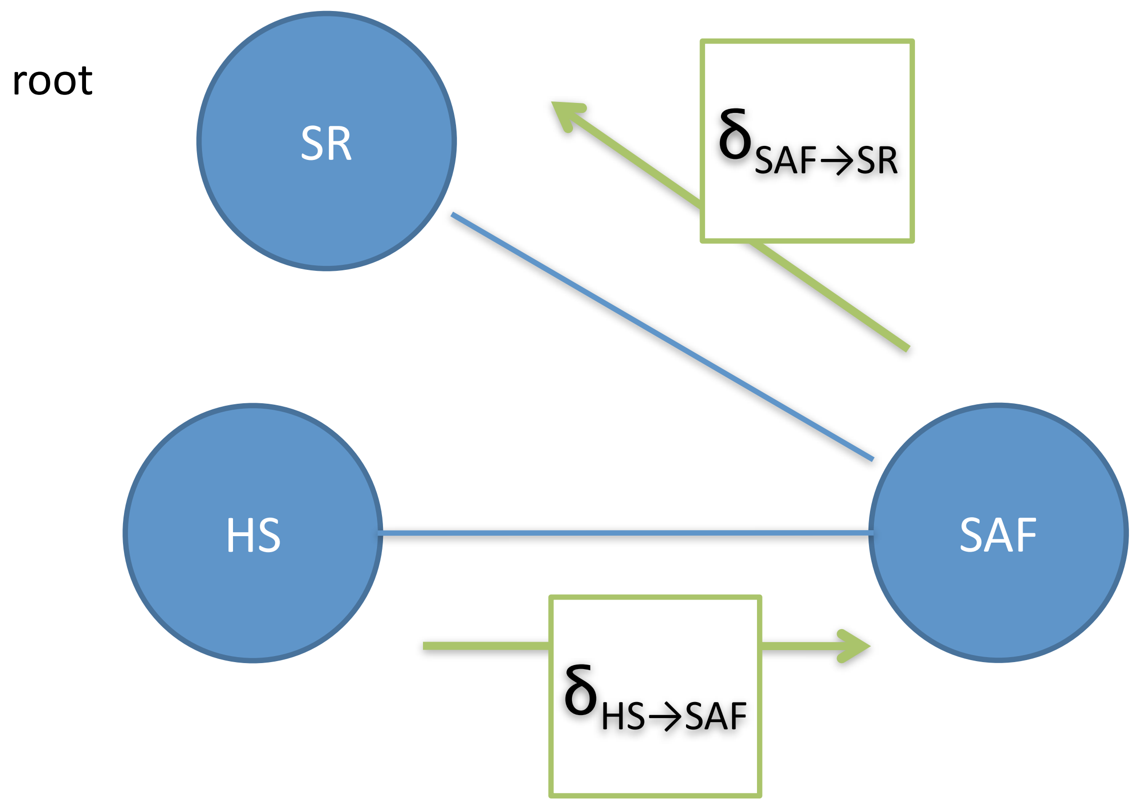
- Later: How to get marginals for **Q** that are not contained in one clique.

Going Farther

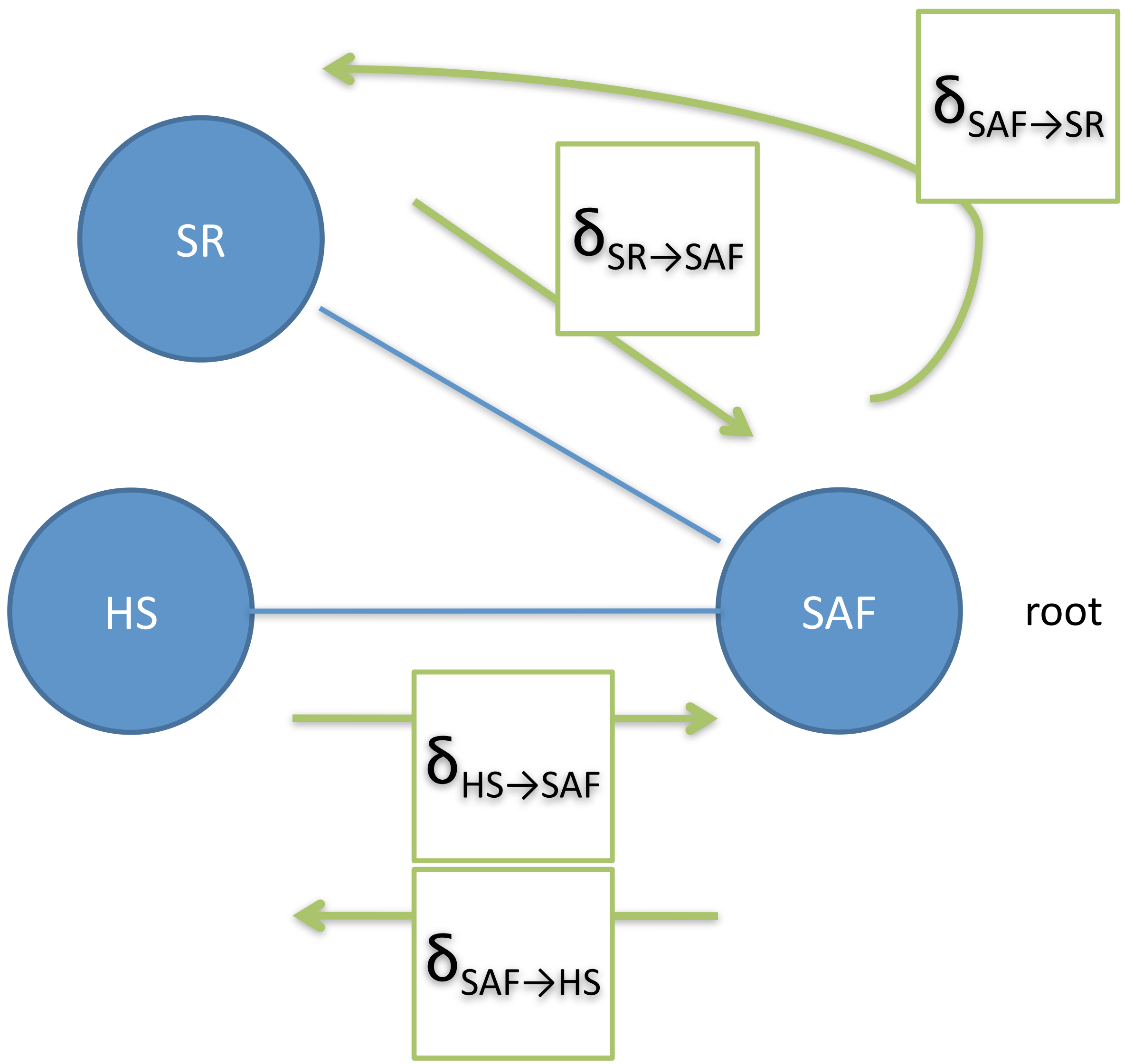
- We can modify this algorithm to give us the marginal probability of *every* random variable in the network.
  - Naively: run clique tree VE with each clique as the root.


SR
δSR→SAF
HS
SAF
root
δHS→SAF


SR
δSR→SAF
Same message as needed for previous root!
root
HS
SAF
δSAF→HS



root
SR
δSAF→SR
HS
SAF
δHS→SAF


$\delta_{SAF \to SR}$
SR
$\delta_{SR \to SAF}$
HS
SAF
root
$\delta_{HS \to SAF}$
$\delta_{SAF \to HS}$

Going Farther

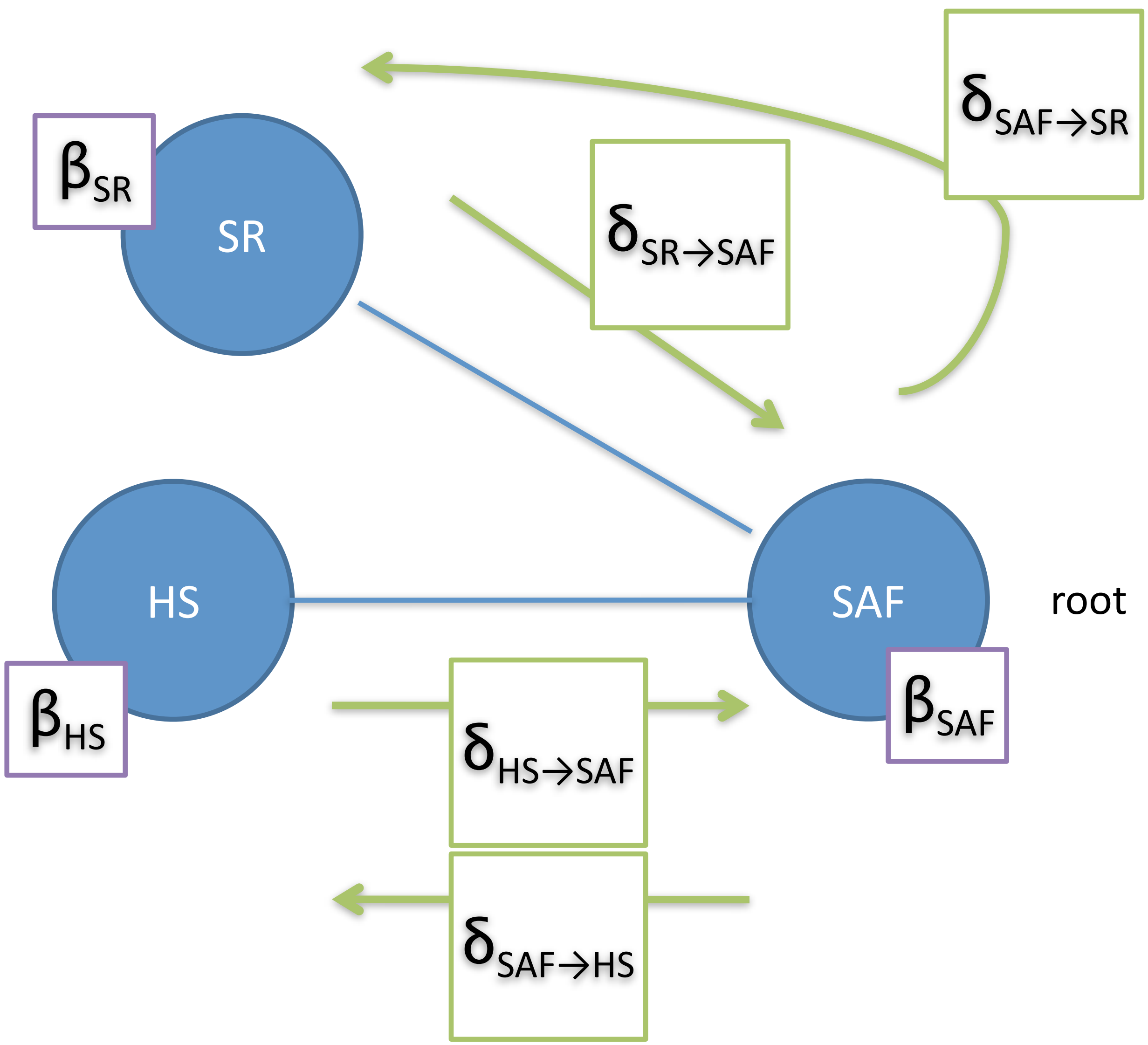
- We can modify this algorithm to give us the marginal probability of *every* random variable in the network.
  - Naively: run clique tree VE with each clique as the root.
  - Better: notice that the same messages get used on different runs; only two messages per clique tree edge.

Sum-Product Message Passing

- Each clique tree vertex $\mathbf{C}_i$ passes messages to each of its neighbors once it's ready to do so.
  - This is asynchronous;
    might want to be careful about *scheduling*.
- At the end, for all $\mathbf{C}_i$:

$$\beta_i = \nu_i \prod_{k \in \text{Neighbors}_i} \delta_{k \to i}$$

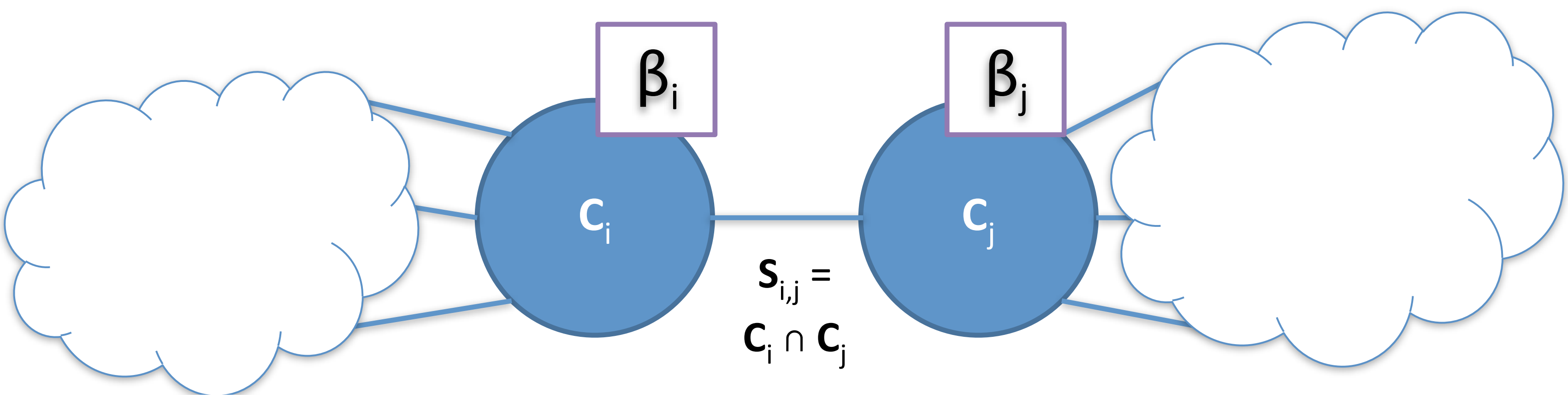
  - This is the unnormalized marginal for $\mathbf{C}_i$.



$\delta_{SAF \to SR}$
β_{SR}
SR
$\delta_{SR \to SAF}$
HS
SAF
root
β_{HS}
β_{SAF}
$\delta_{HS \to SAF}$
$\delta_{SAF \to HS}$

Calibration

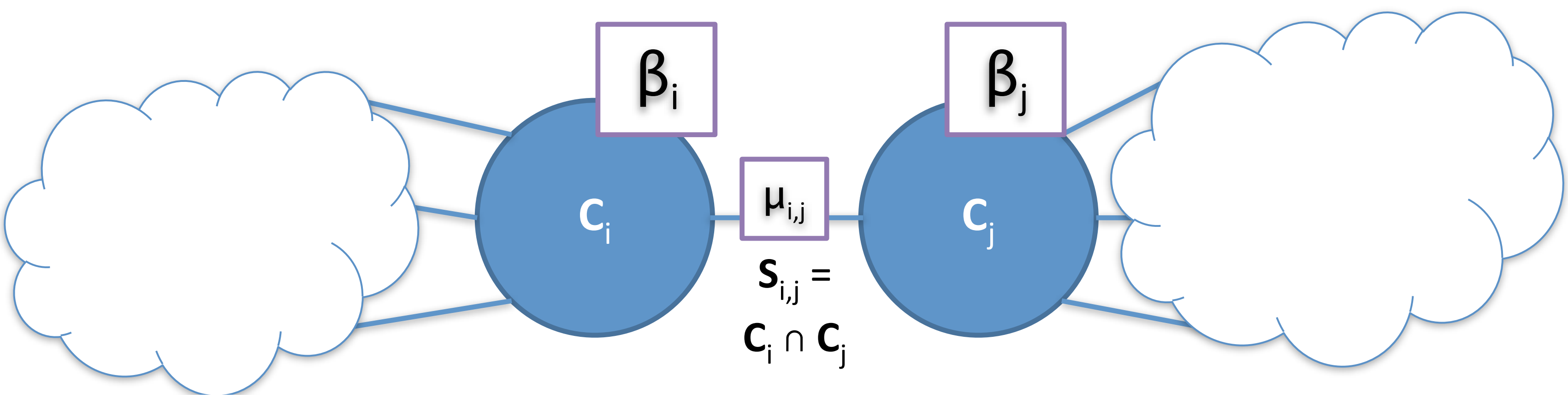
- Two adjacent cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ are calibrated when: $$\sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i = \sum_{\boldsymbol{C}_j \setminus \boldsymbol{S}_{i,j}} \beta_j$$



Calibration

- Two adjacent cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ are calibrated when:

$$\begin{aligned} \sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \beta_i &= \sum_{\boldsymbol{C}_j \setminus \boldsymbol{S}_{i,j}} \beta_j \\ &= \mu_{i,j}(\boldsymbol{S}_{i,j}) \end{aligned}$$



Sum-Product Message Passing

- Computes the marginal probability of all variables using only twice the computation of the upward pass.
- Results in a **calibrated clique tree**.
- Attractive if we expect different kinds of queries.

Calibrated Clique Tree as a Graphical Model

- Original (unnormalized) factor model and calibrated clique tree represent the same (unnormalized) measure:

$$\prod_{\phi \in \Phi} \phi = \frac{\prod_{\boldsymbol{C} \in \text{Vertices}(\mathcal{T})} \beta_{\boldsymbol{C}}}{\prod_{\boldsymbol{S} \in \text{Edges}(\mathcal{T})} \mu_{\boldsymbol{S}}}$$

clique **beliefs**

sepset **beliefs**

unnormalized Gibbs distribution from original factors **Φ**

Calibrated Clique Tree as a Graphical Model

$$\frac{\prod_{\boldsymbol{C} \in \text{Vertices}(\mathcal{T})} \beta_{\boldsymbol{C}}}{\prod_{\boldsymbol{S} \in \text{Edges}(\mathcal{T})} \mu_{\boldsymbol{S}}} = \frac{\prod_{\boldsymbol{C} \in \text{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}' \in \text{Neighbors}(\boldsymbol{C})} \delta_{\boldsymbol{C}' \to \boldsymbol{C}}}{\prod_{\boldsymbol{S} = \in \text{Edges}(\mathcal{T}) : \boldsymbol{S} = \boldsymbol{C} \cap \boldsymbol{C}'} \mu_{\boldsymbol{C} \cap \boldsymbol{C}'}}$$

$$= \frac{\prod_{\boldsymbol{C} \in \text{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}' \in \text{Neighbors}(\boldsymbol{C})} \delta_{\boldsymbol{C}' \to \boldsymbol{C}}}{\prod_{\boldsymbol{S} = \in \text{Edges}(\mathcal{T}) : \boldsymbol{S} = \boldsymbol{C} \cap \boldsymbol{C}'} \sum_{\boldsymbol{C} \setminus \boldsymbol{S}} \beta_{\boldsymbol{C}}}$$

$$= \frac{\prod_{\boldsymbol{C} \in \text{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}' \in \text{Neighbors}(\boldsymbol{C})} \delta_{\boldsymbol{C}' \to \boldsymbol{C}}}{\prod_{\boldsymbol{S} = \in \text{Edges}(\mathcal{T}) : \boldsymbol{S} = \boldsymbol{C} \cap \boldsymbol{C}'} \sum_{\boldsymbol{C} \setminus \boldsymbol{S}} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}'' \text{Neighbors}(\boldsymbol{C})} \delta_{\boldsymbol{C}'' \to \boldsymbol{C}}}$$

Calibrated Clique Tree as a Graphical Model

$$
\begin{aligned}
\frac{\prod_{\boldsymbol{C} \in \mathrm{Vertices}(\mathcal{T})} \beta_{\boldsymbol{C}}}{\prod_{\boldsymbol{S} \in \mathrm{Edges}(\mathcal{T})} \mu_{\boldsymbol{S}}} &= \ldots \\
&= \frac{\prod_{\boldsymbol{C} \in \mathrm{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}' \in \mathrm{Neighbors}(\boldsymbol{C})} \delta_{\boldsymbol{C}' \to \boldsymbol{C}}}{\prod_{\boldsymbol{S} = \in \mathrm{Edges}(\mathcal{T}) : \boldsymbol{S} = \boldsymbol{C} \cap \boldsymbol{C}'} \delta_{\boldsymbol{C}' \to \boldsymbol{C}} \sum_{\boldsymbol{C} \setminus \boldsymbol{S}} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}'' \in \mathrm{Neighbors}(\boldsymbol{C}) \setminus \{\boldsymbol{C}'\}} \delta_{\boldsymbol{C}'' \to \boldsymbol{C}}} \\
&= \frac{\prod_{\boldsymbol{C} \in \mathrm{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} \prod_{\boldsymbol{C}' \in \mathrm{Neighbors}(\boldsymbol{C})} \delta_{\boldsymbol{C}' \to \boldsymbol{C}}}{\prod_{\boldsymbol{S} = \in \mathrm{Edges}(\mathcal{T}) : \boldsymbol{S} = \boldsymbol{C} \cap \boldsymbol{C}'} \delta_{\boldsymbol{C}' \to \boldsymbol{C}} \delta_{\boldsymbol{C} \to \boldsymbol{C}'}} \\
&= \prod_{\boldsymbol{C} \in \mathrm{Vertices}(\mathcal{T})} \nu_{\boldsymbol{C}} \\
&= \prod_{\phi \in \boldsymbol{\Phi}} \phi
\end{aligned}
$$

$$
\mu_{i,j} = \delta_{\boldsymbol{C}_i \to \boldsymbol{C}_j} \delta_{\boldsymbol{C}_j \to \boldsymbol{C}_i}
$$

What You Now Know

- Can reinterpret variable elimination as message passing in a clique tree.
- Can share computation to get lots of marginals with only double the cost of VE.
  - Sum-product belief propagation
- Calibrated clique tree is an alternative representation of the original Gibbs distribution.