Contextual Memory Trees

Wen Sun
CMU —> MSR NYC
wensun@cs.cmu.edu

Joint work with Alina Beygelzimer, Hal Daumé III, Paul Mineiro, and John Langford



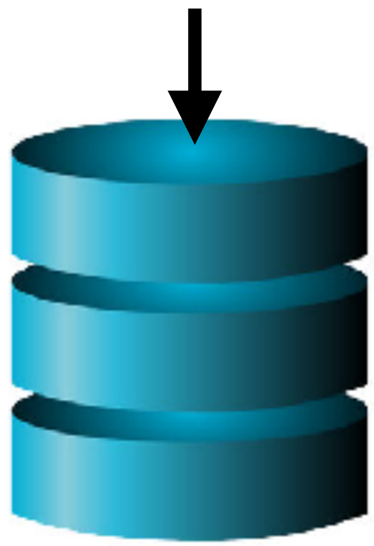

Carnegie Mellon University The Robotics Institute
YAHOO! RESEARCH
UNIVERSITY OF MARYLAND
Microsoft Research

External Memory

External Memory
(many query-answer pairs)

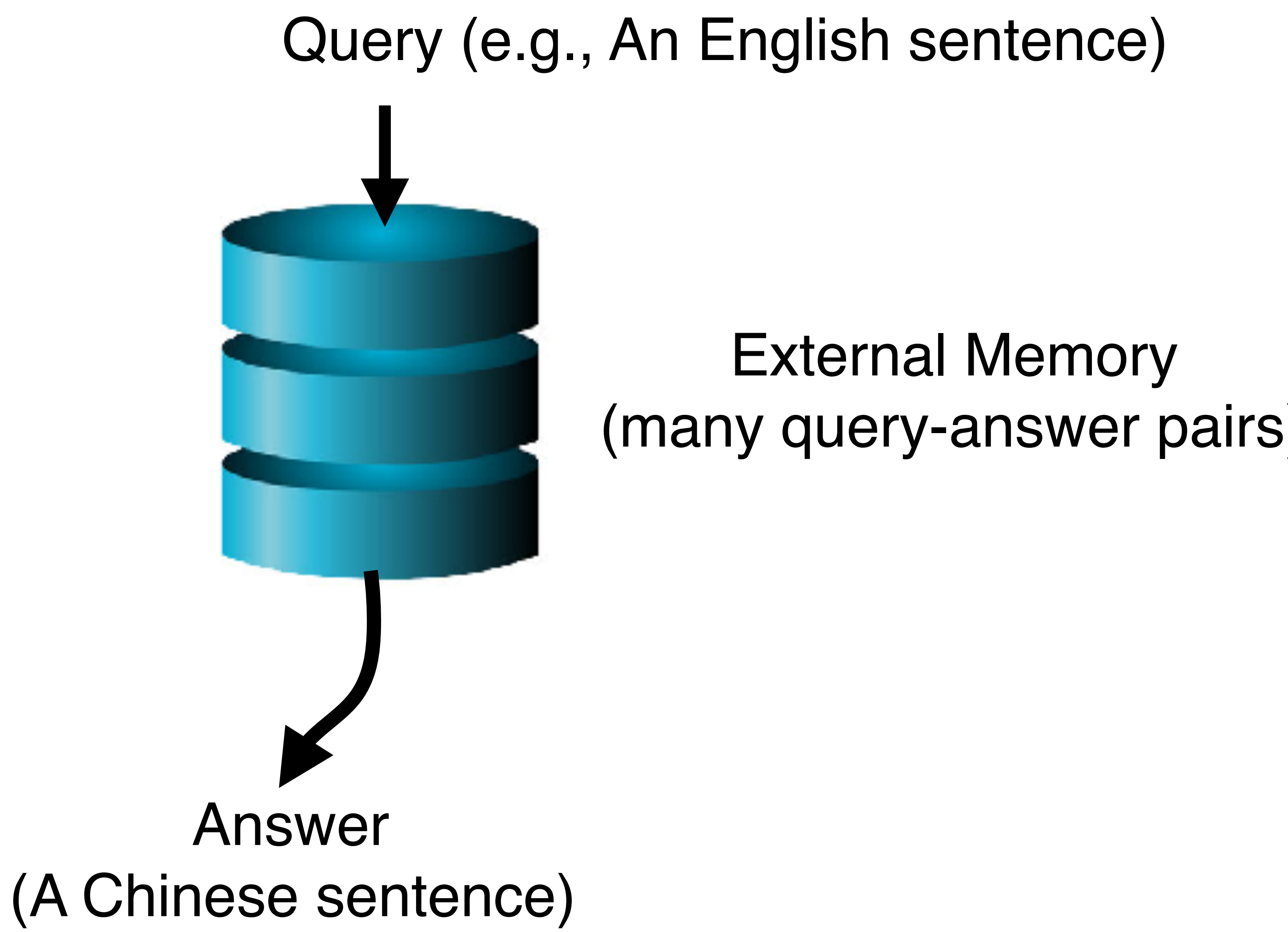
External Memory

Query (e.g., An English sentence)



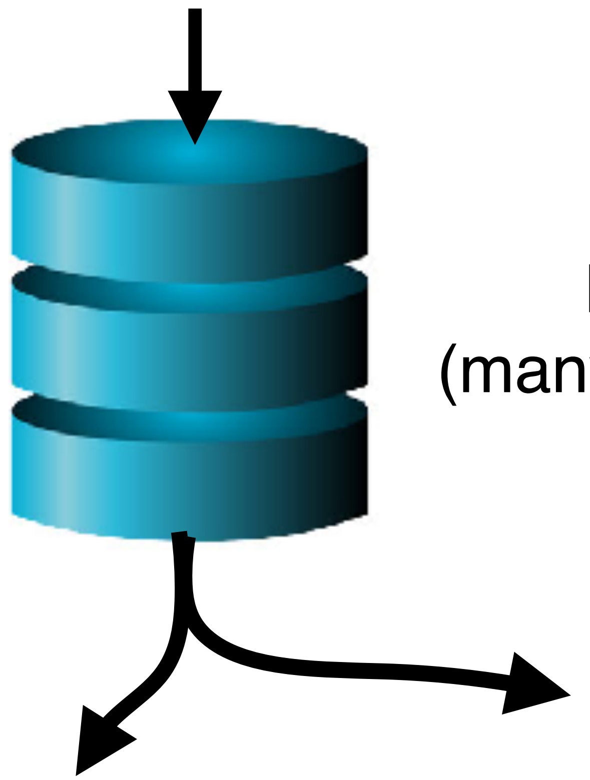
External Memory
(many query-answer pairs)

External Memory



External Memory

Query (e.g., An English sentence)

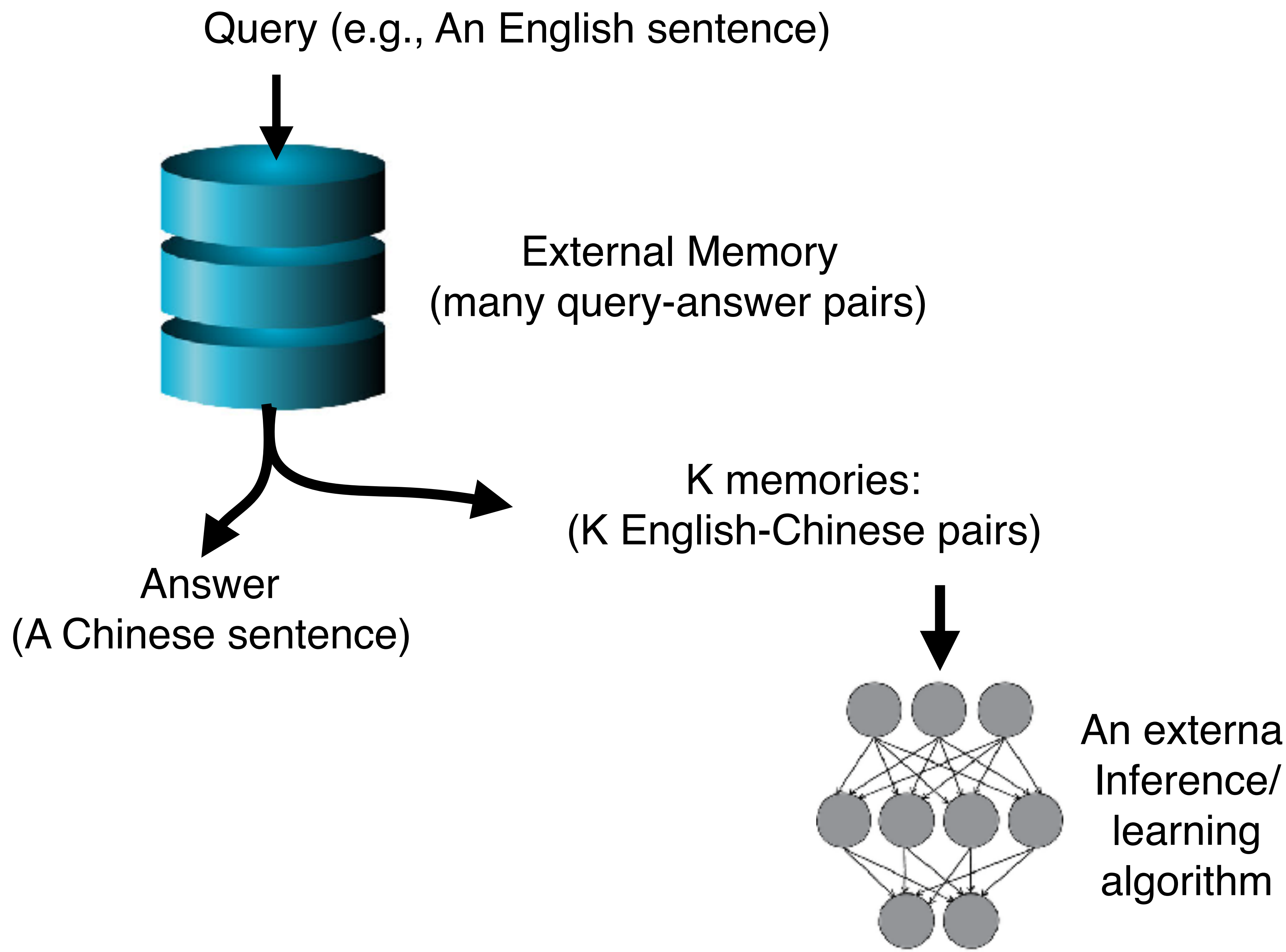


External Memory
(many query-answer pairs)

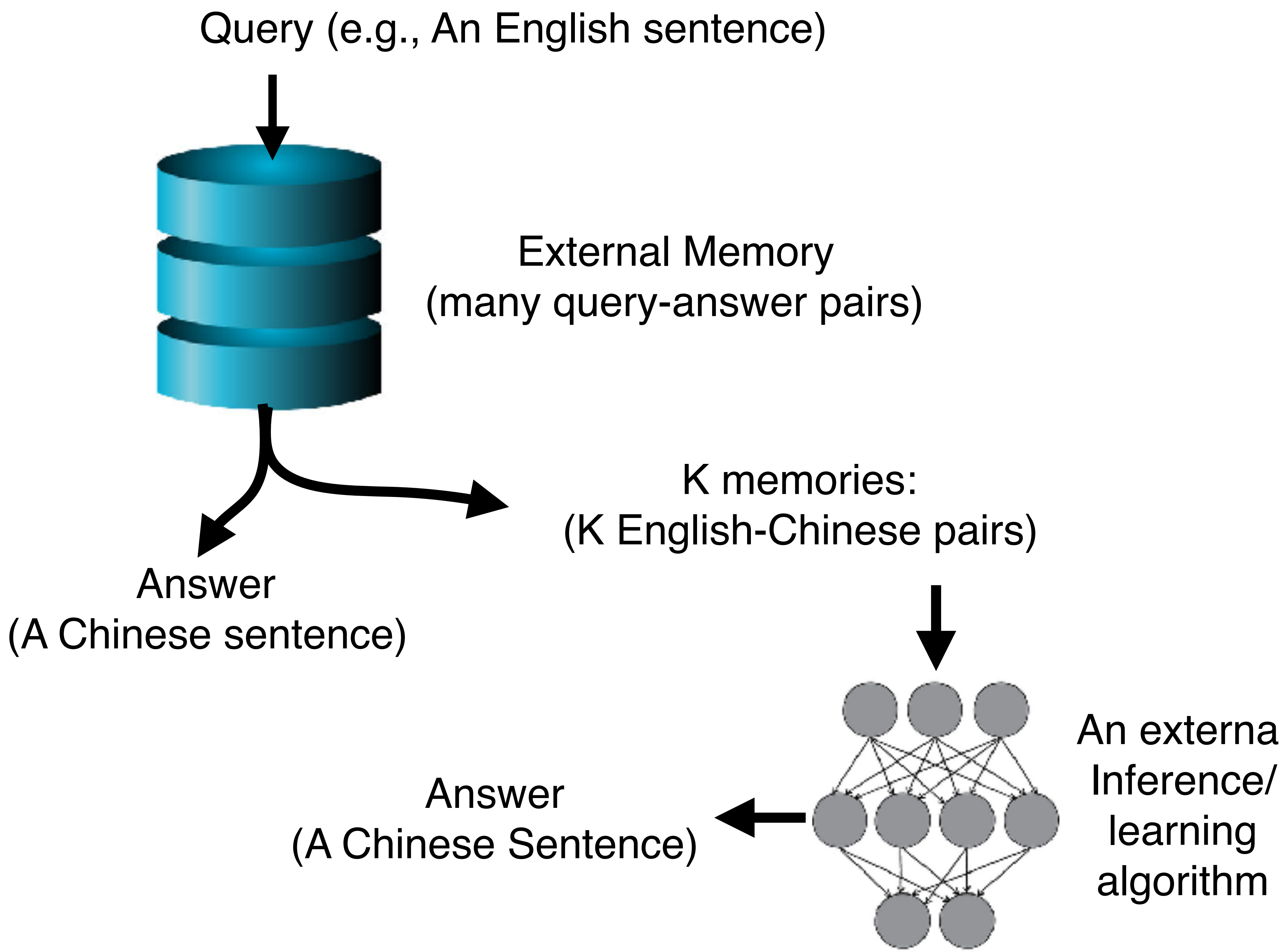
K memories:
(K English-Chinese pairs)

Answer
(A Chinese sentence)

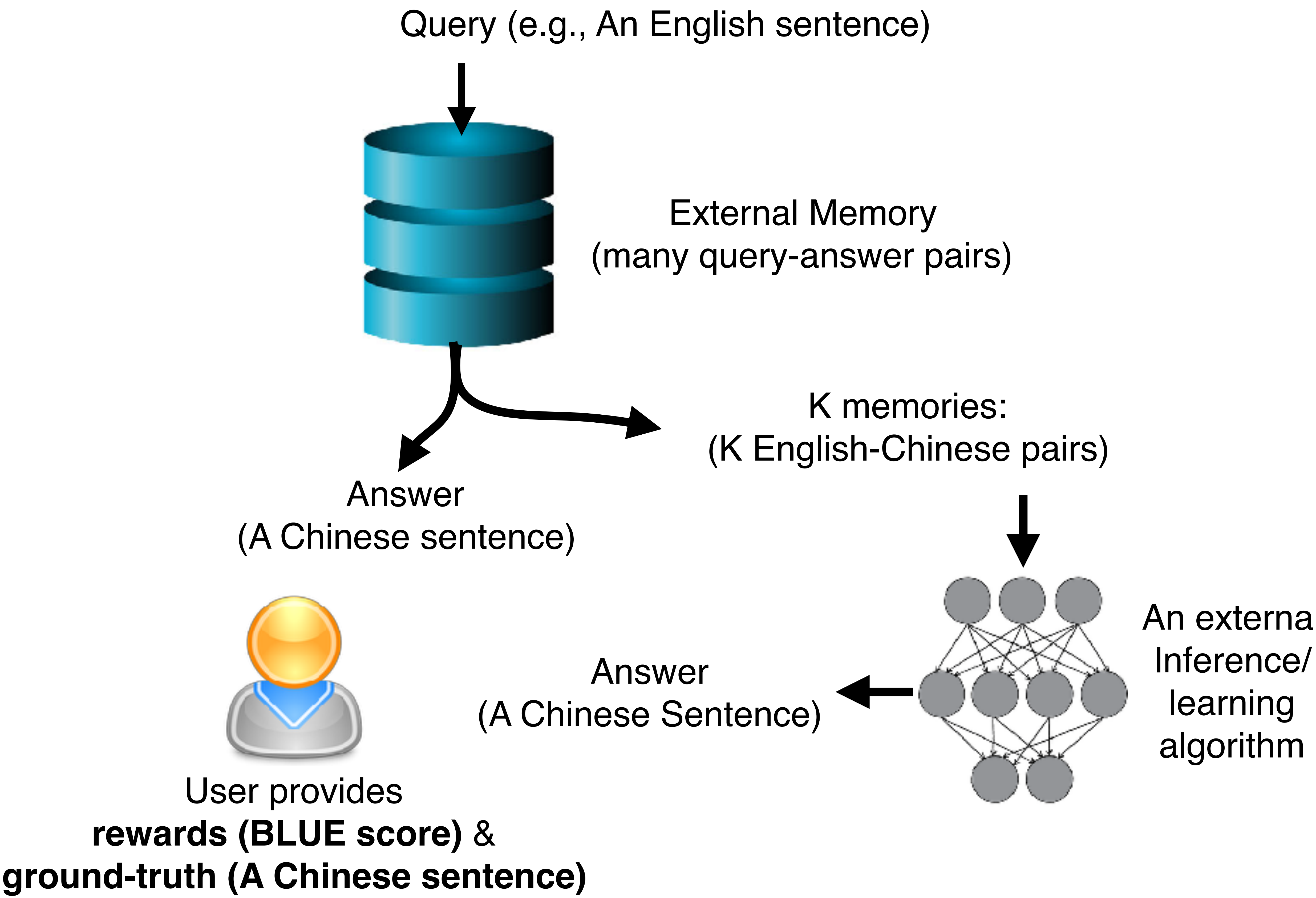
External Memory



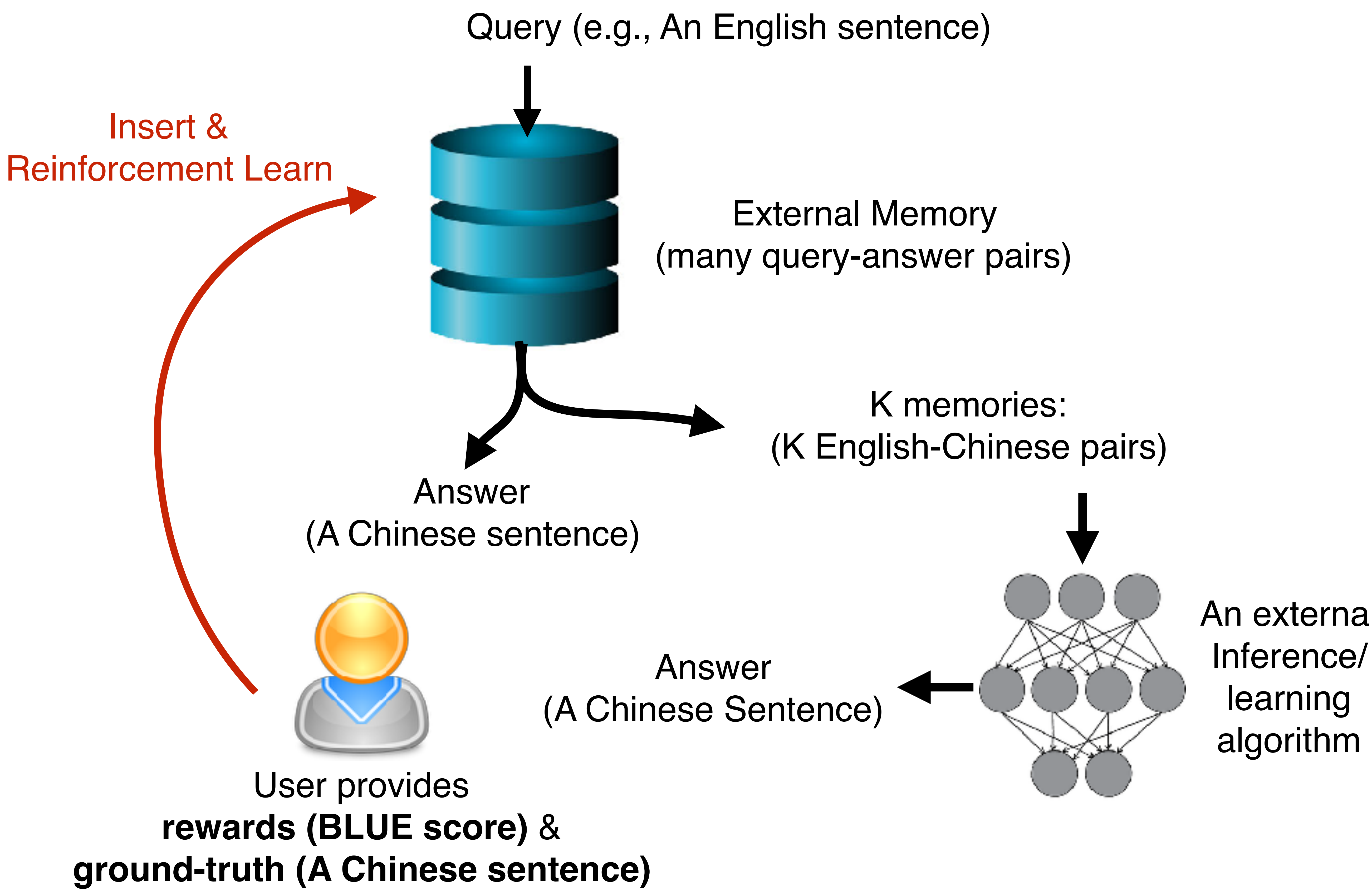
External Memory



External Memory



External Memory



Desired Properties

Desired Properties

Online: every operation works one example at a time

Desired Properties

Online: every operation works one example at a time

Linear Space: O(# of examples)

Desired Properties

Online: every operation works one example at a time

Linear Space: O(# of examples)

Fast: Logarithmic Read & Write (Log(# of examples))

Desired Properties

Online: every operation works one example at a time

Linear Space: O(# of examples)

Fast: Logarithmic Read & Write (Log(# of examples))

Learning-based: do not assume, e.g., Euclidean space

Desired Properties

Online: every operation works one example at a time

Linear Space: O(# of examples)

Fast: Logarithmic Read & Write (Log(# of examples))

Learning-based: do not assume, e.g., Euclidean space

Self-consistency: identify the item seen before

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

QUERY(q)

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

QUERY(q)

$$\downarrow$$

$$(m_1, \ldots, m_k)$$

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

QUERY(q)

$$\downarrow$$

$$(m_1, \ldots, m_k)$$

2. If reward r_i is given,

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

QUERY(q)

$$\downarrow$$

$$(m_1, \ldots, m_k)$$

2. If reward r_i is given,

UPDATE(q, m_i, r_i)

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

QUERY(q)

$$(m_1, \ldots, m_k)$$

2. If reward r_i is given,

UPDATE(q, m_i, r_i)

3. If ground-truth value v is given,

Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of **Query (q)** & **Value (v)**

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

QUERY(q)

$$(m_1, \ldots, m_k)$$

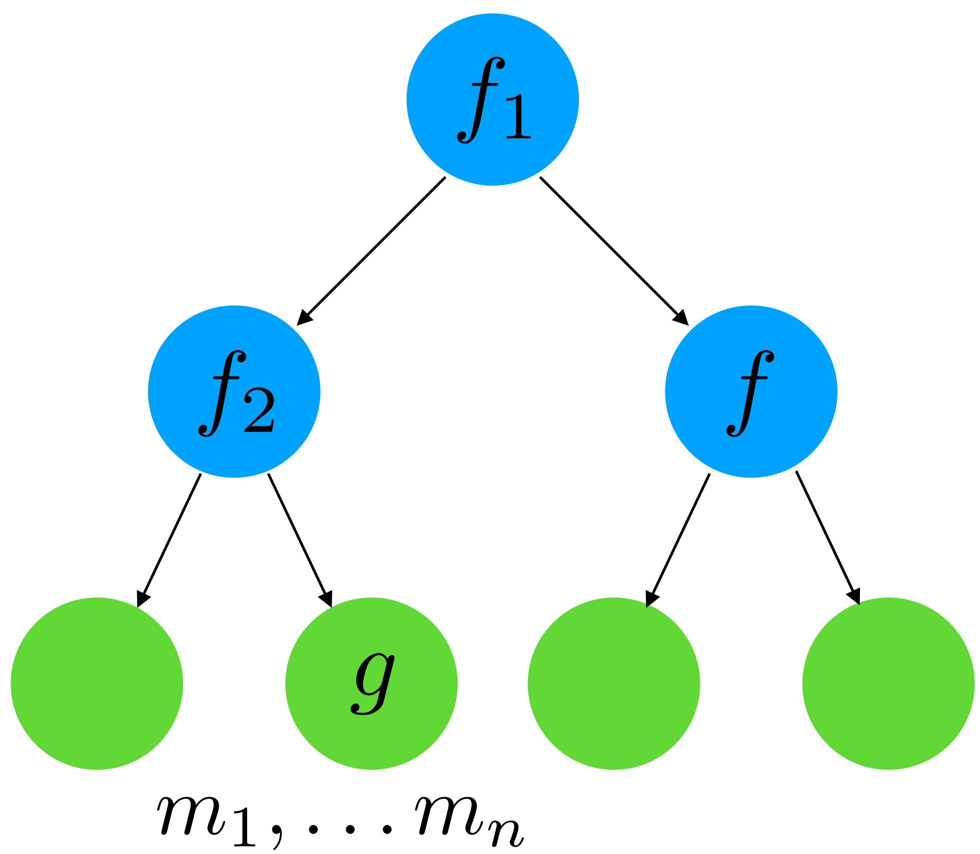
2. If reward r_i is given,

UPDATE(q, m_i, r_i)

3. If ground-truth value v is given,

INSERT(q, v)

Datastructure of CMT:
A Nearly Balanced Binary Tree



Datastructure of CMT: A Nearly Balanced Binary Tree

Internal node

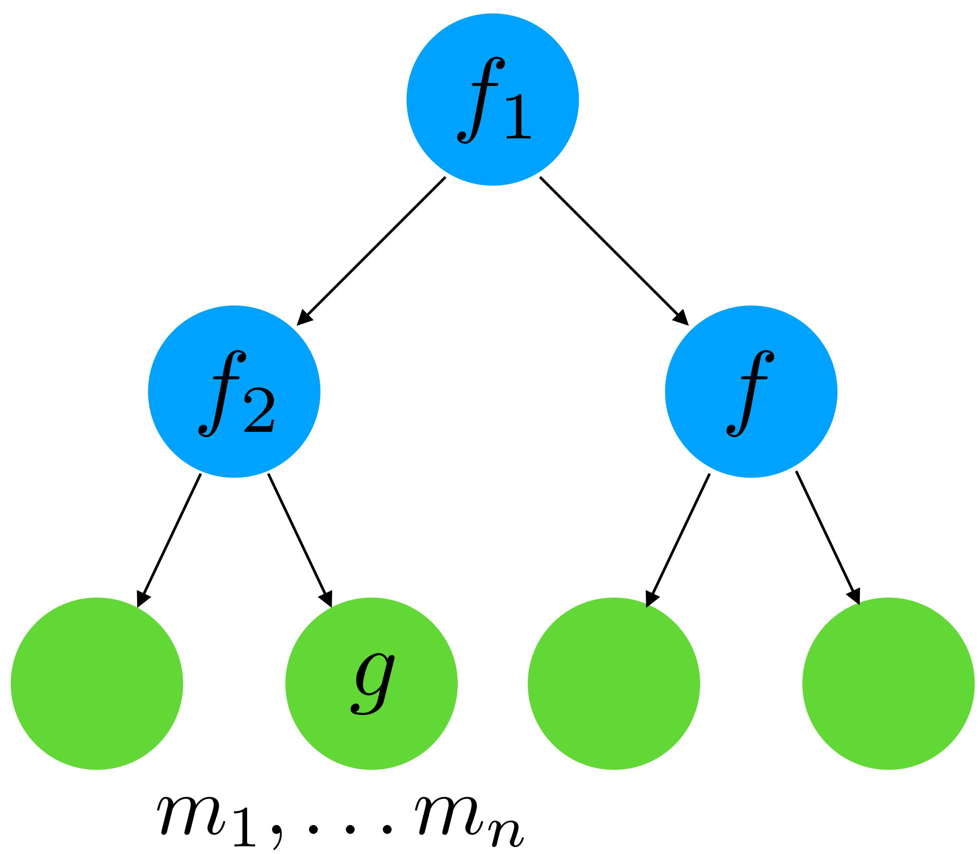
$n_L \, , n_R$

# of memories in the Left/Right subtree

Router (binary classifier):

$$f : \mathcal{Q} \rightarrow \{-1, +1\}$$

e.g.,$f(q) = \mathrm{sign}(\beta^\top \phi(q))$



Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

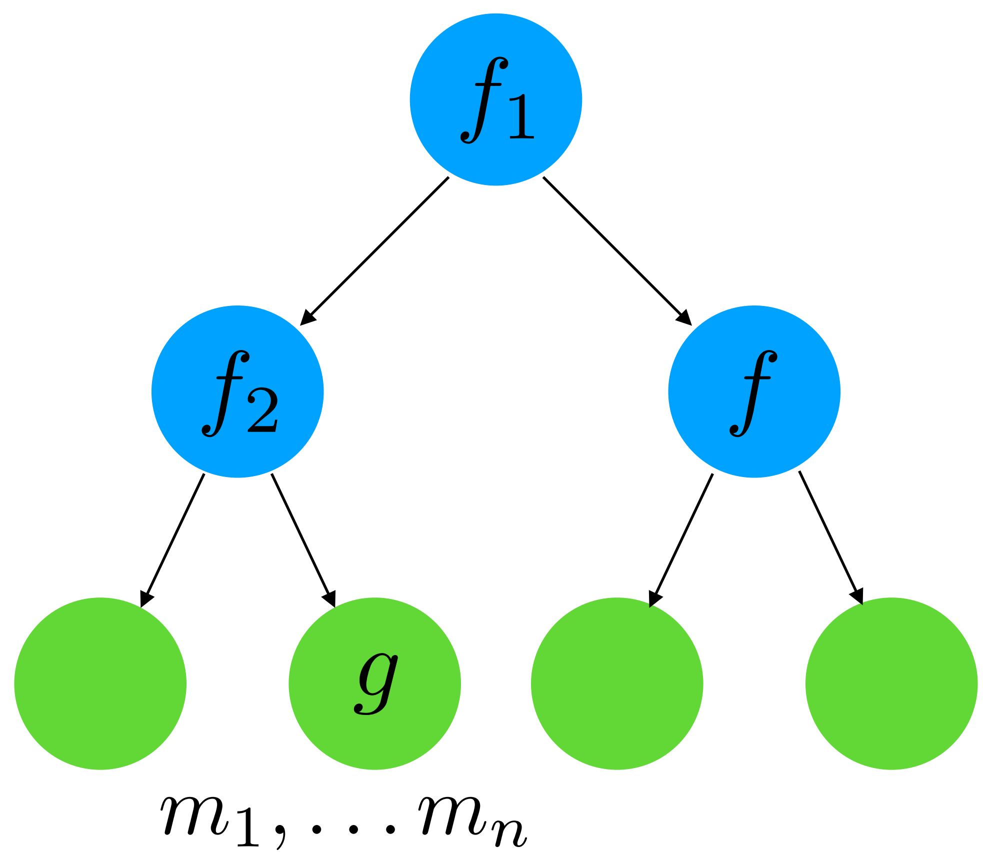
$$n_L \, , n_R$$

# of memories in the Left/Right subtree

Router (binary classifier):

$$f : \mathcal{Q} \rightarrow \{-1, +1\}$$

e.g., $f(q) = \mathrm{sign}(\beta^\top \phi(q))$



Leaf

Memories:

$m_1, \ldots m_n$

Scorer Function:

$$g(q, x) = w^T \phi(q, x)$$

Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

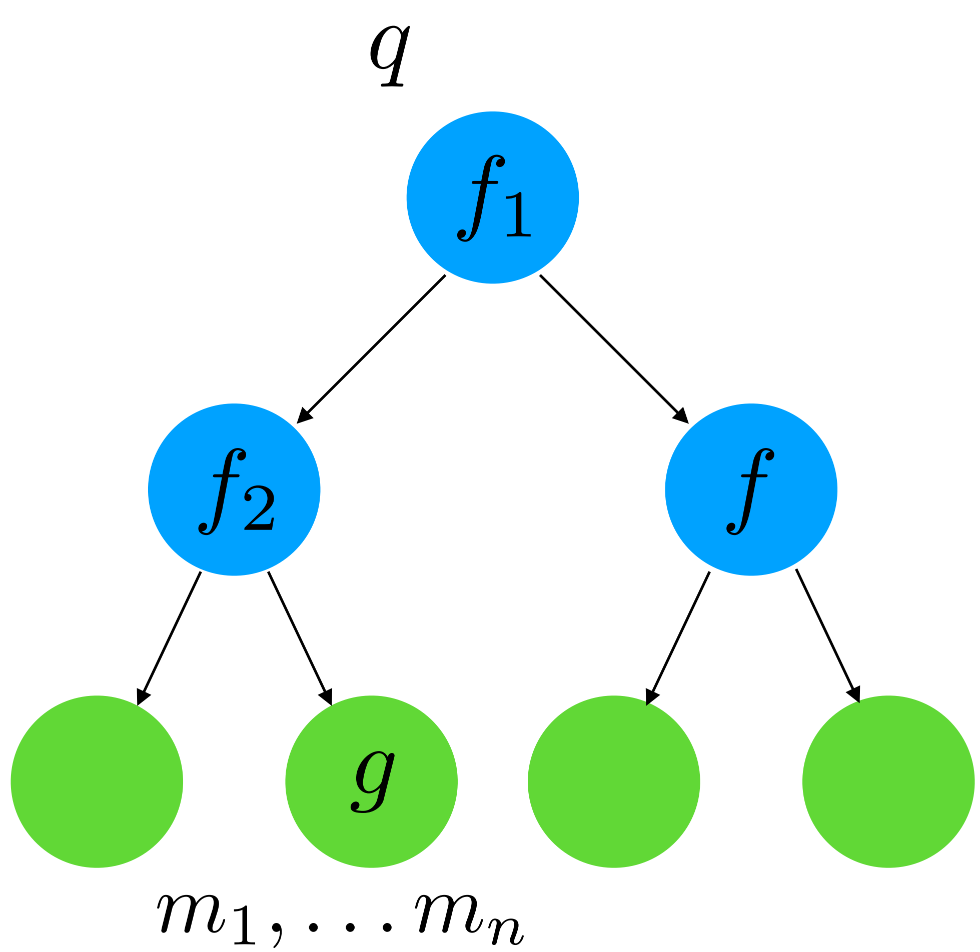
n_L , n_R

# of memories in the Left/Right subtree

Router (binary classifier):

$$f : \mathcal{Q} \rightarrow \{-1, +1\}$$

e.g., $f(q) = \text{sign}(\beta^\top \phi(q))$



Leaf

Memories:

$m_1, \ldots m_n$

Scorer Function:

$$g(q, x) = w^T \phi(q, x)$$

Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

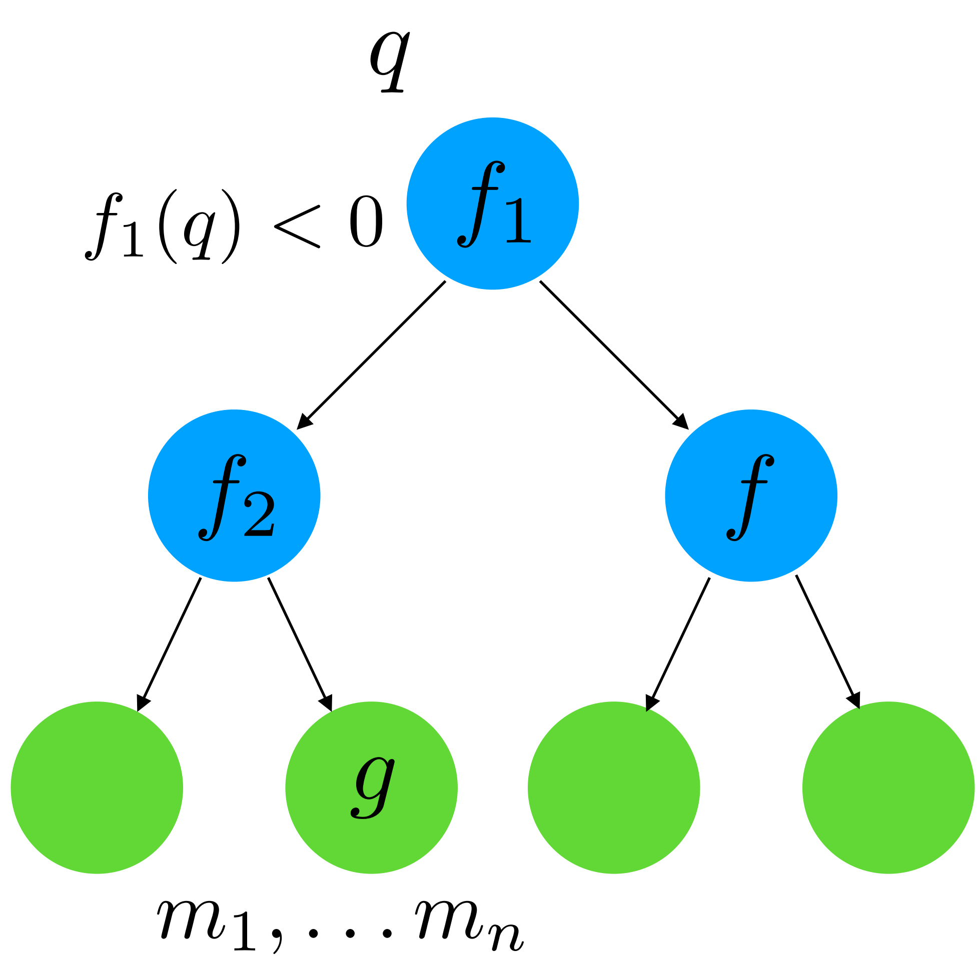
n_L , n_R

# of memories in the Left/Right subtree

Router (binary classifier):

$$f : \mathcal{Q} \rightarrow \{-1, +1\}$$

e.g.,$f(q) = \text{sign}(\beta^\top \phi(q))$



Leaf

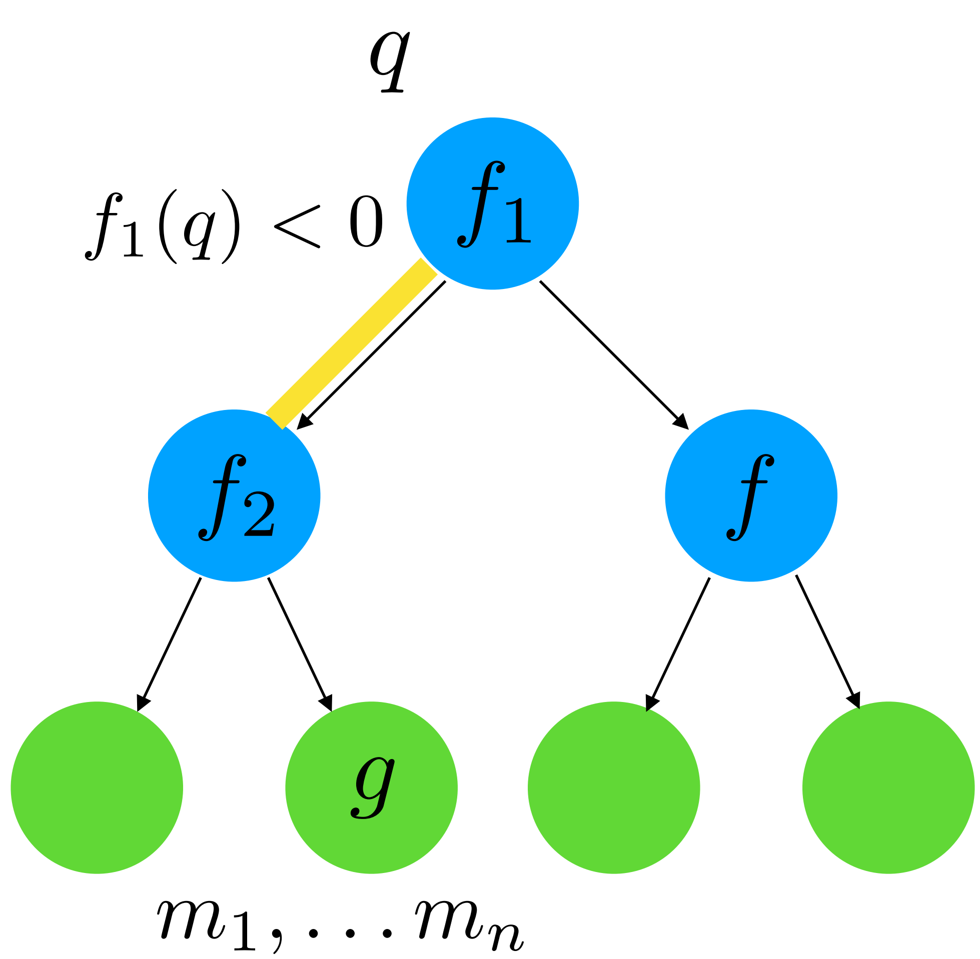
Memories:

$m_1, \ldots m_n$

Scorer Function:

$g(q, x) = w^T \phi(q, x)$

Datastructure of CMT: A Nearly Balanced Binary Tree



Internal node

n_L , n_R

\# of memories in the Left/Right subtree

Router (binary classifier):

$f : \mathcal{Q} \rightarrow \{-1, +1\}$

e.g.,$f(q) = \mathrm{sign}(\beta^\top \phi(q))$

Leaf

Memories: $m_1, \ldots m_n$

Scorer Function:

$g(q, x) = w^T \phi(q, x)$

Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

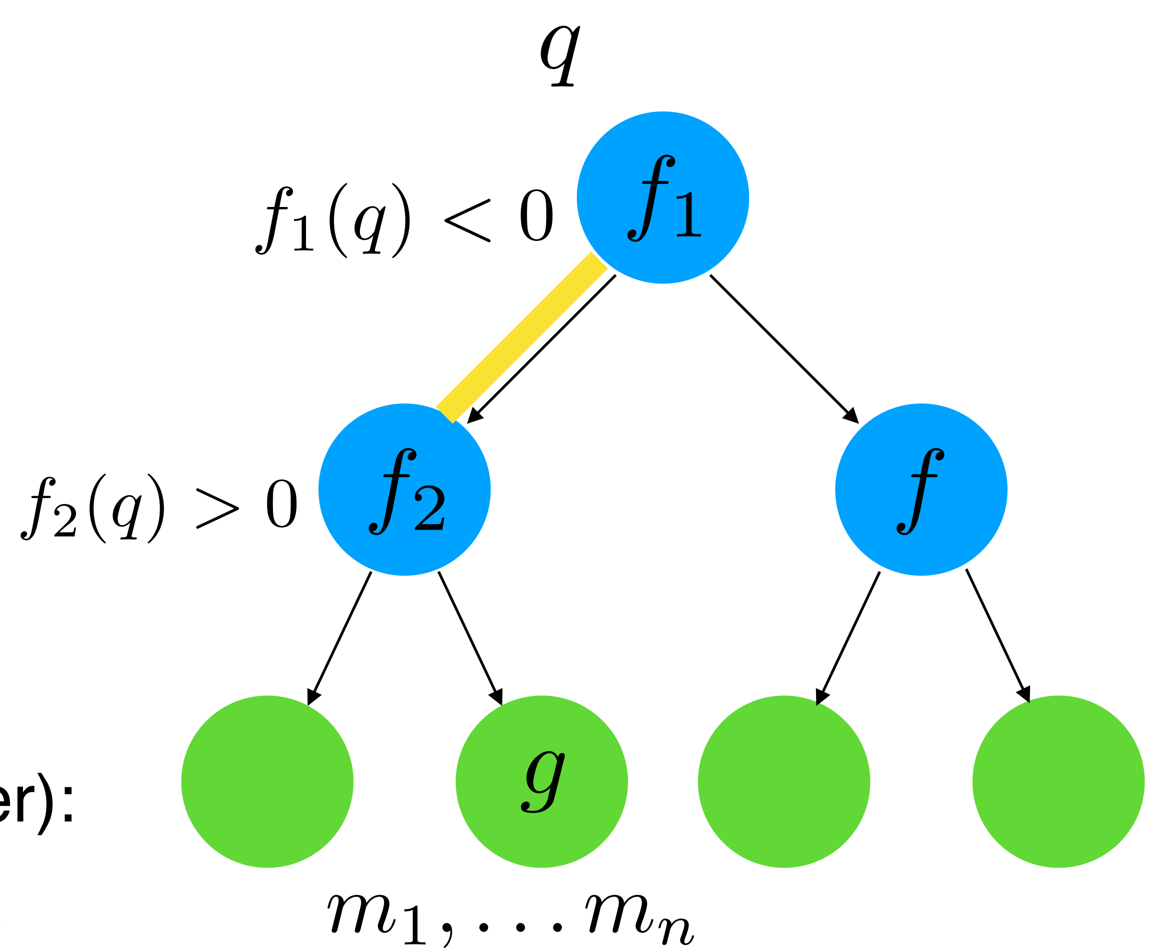
n_L , n_R

# of memories in the Left/Right subtree

Router (binary classifier):

$f : \mathcal{Q} \rightarrow \{-1, +1\}$

e.g., $f(q) = \mathrm{sign}(\beta^\top \phi(q))$



Leaf

Memories:

$m_1, \ldots m_n$

Scorer Function:

$g(q, x) = w^T \phi(q, x)$

Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

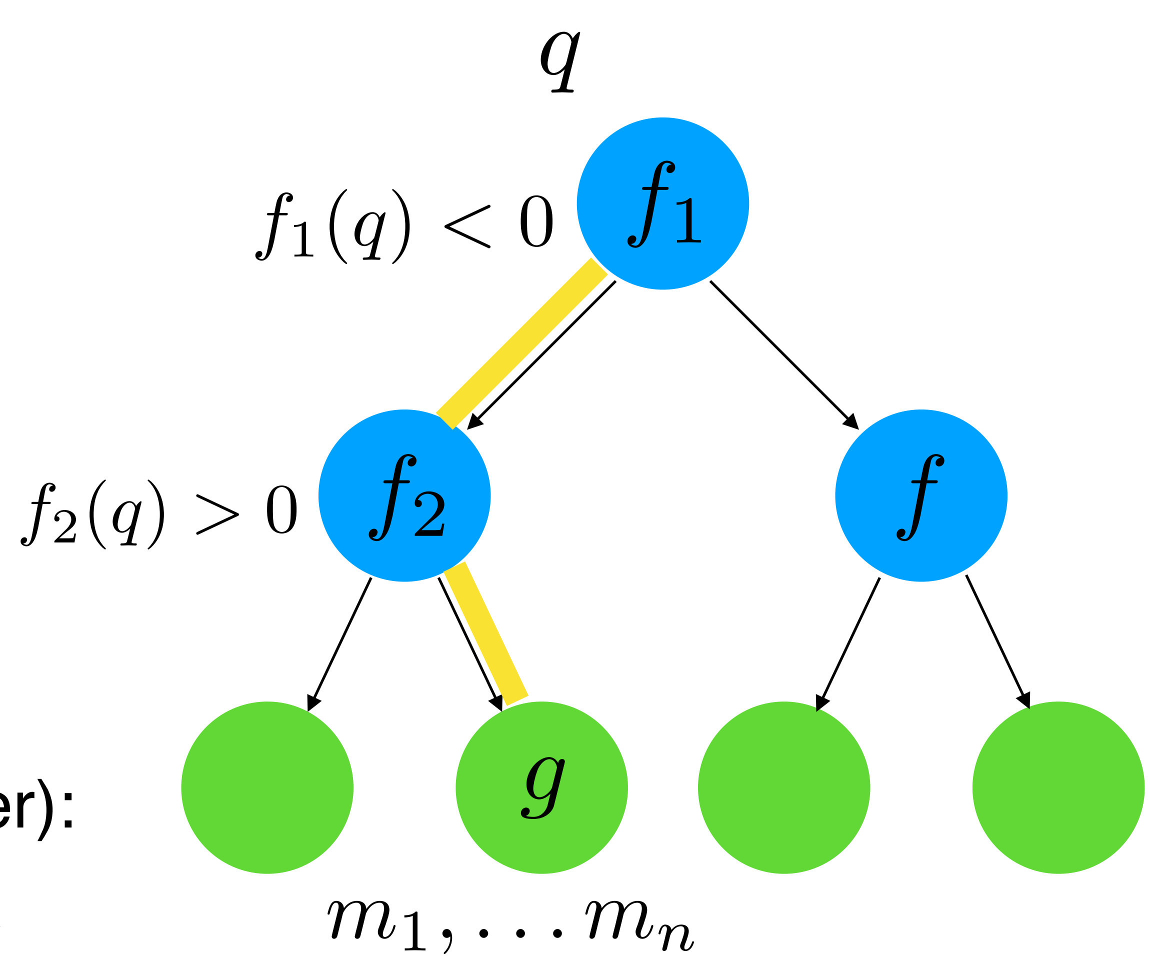
n_L , n_R

# of memories in the Left/Right subtree

Router (binary classifier):

$f : \mathcal{Q} \rightarrow \{-1, +1\}$

e.g., $f(q) = \text{sign}(\beta^\top \phi(q))$



Leaf

Memories:

$m_1, \ldots m_n$

Scorer Function:

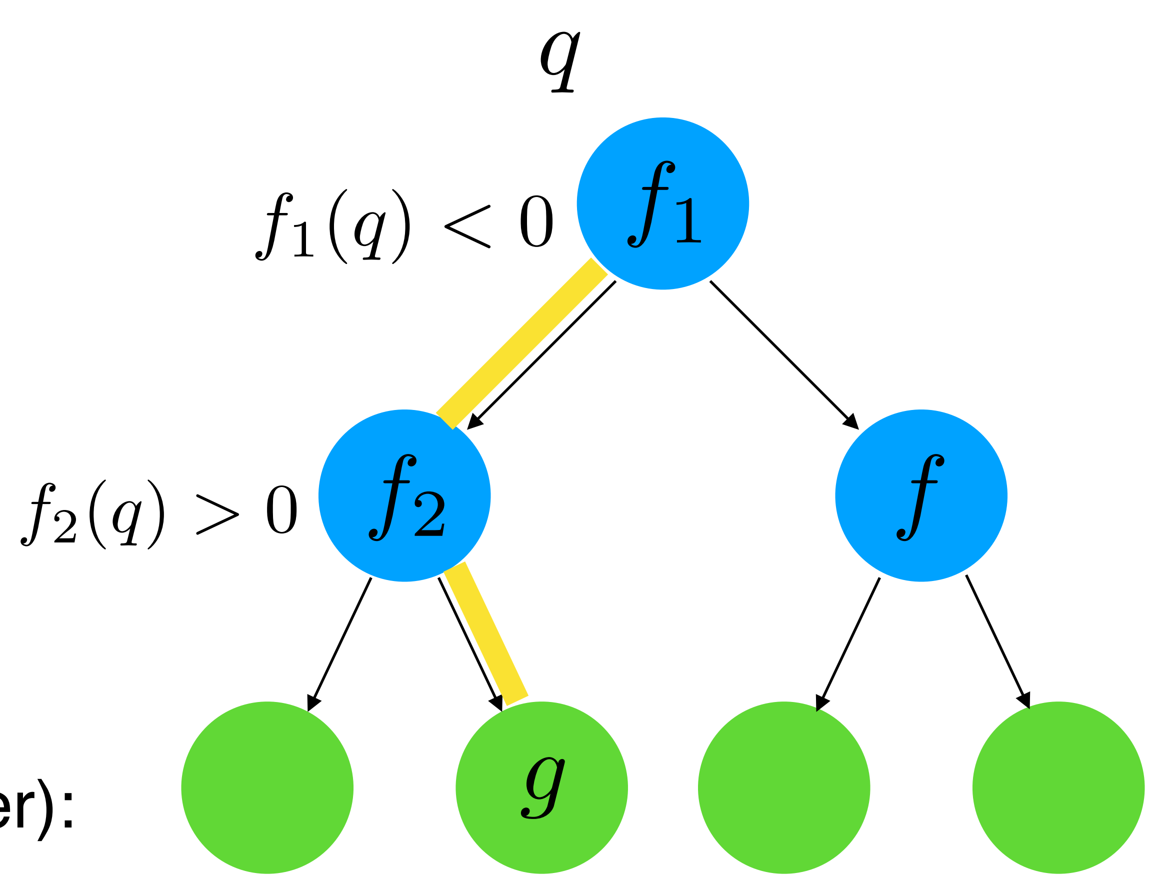
$g(q, x) = w^T \phi(q, x)$

Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

$n_L \, , n_R$

\# of memories in the Left/Right subtree



$m_1, \ldots m_n$

Leaf

Memories:

$m_1, \ldots m_n$

Router (binary classifier):

$f : \mathcal{Q} \rightarrow \{-1, +1\}$

e.g.,$f(q) = \mathrm{sign}(\beta^\top \phi(q))$

$$m^* = \arg\max_m g(q, m)$$

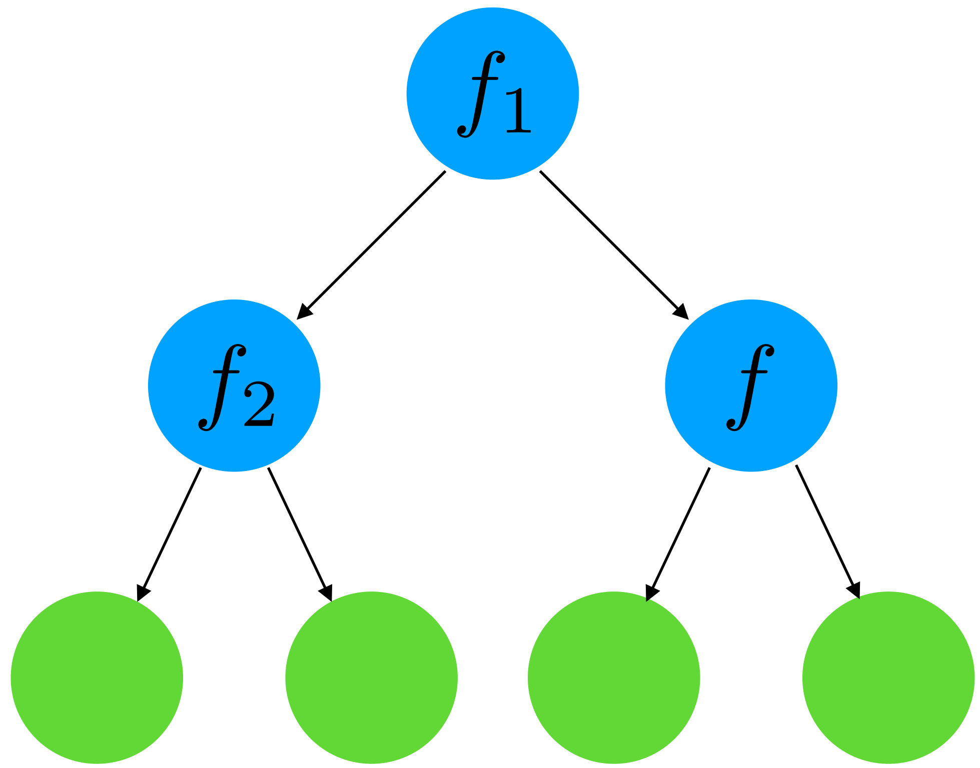
Select the most relevant memory

Scorer Function:

$g(q, x) = w^T \phi(q, x)$

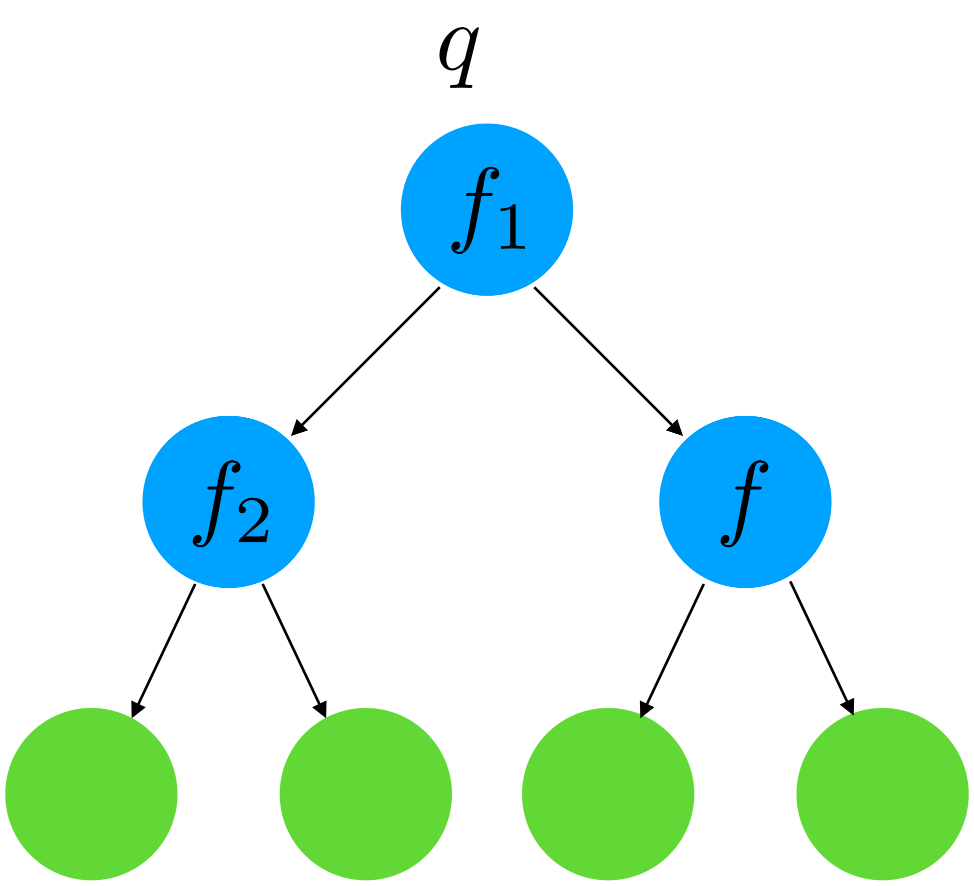
Update Routers:

Using reward signals to update routers



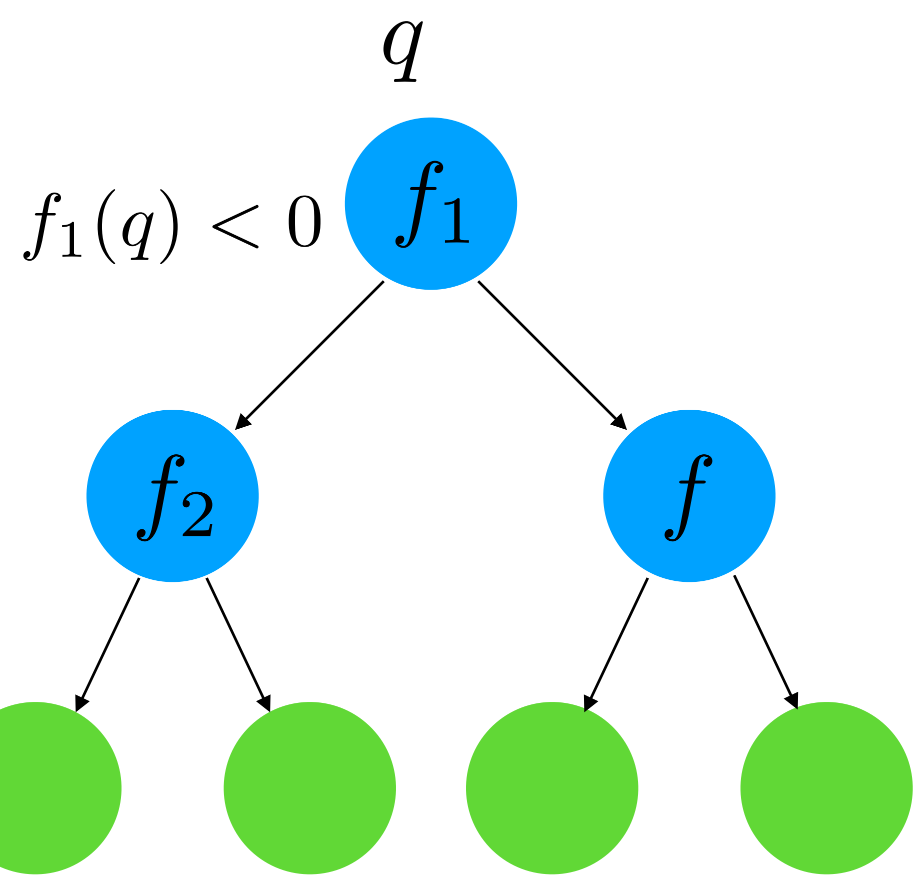
Update Routers:

Using reward signals to update routers



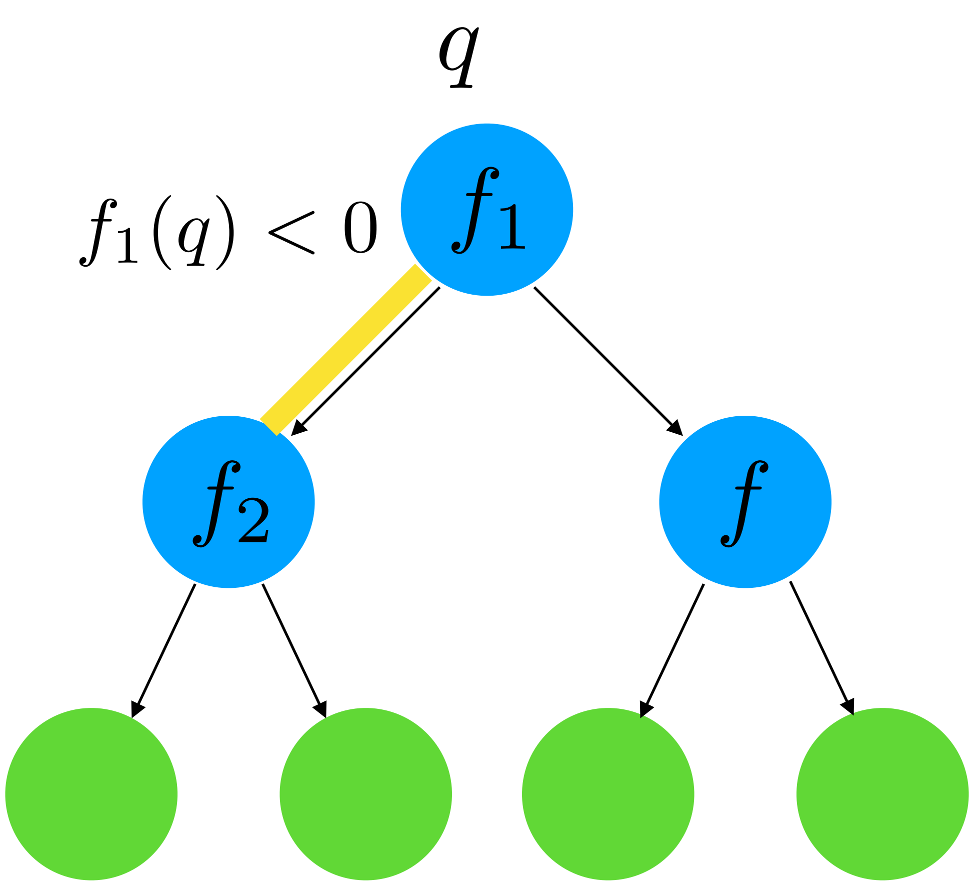
Update Routers:

Using reward signals to update routers



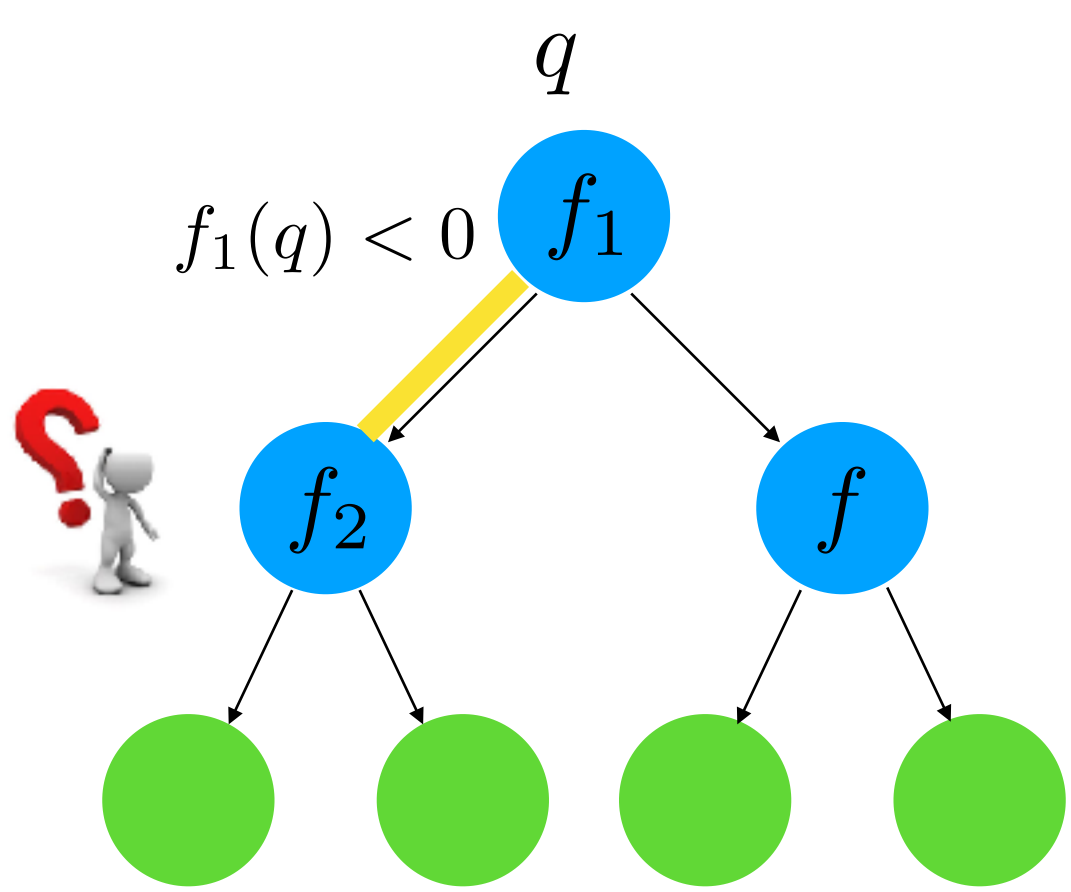
Update Routers:

Using reward signals to update routers



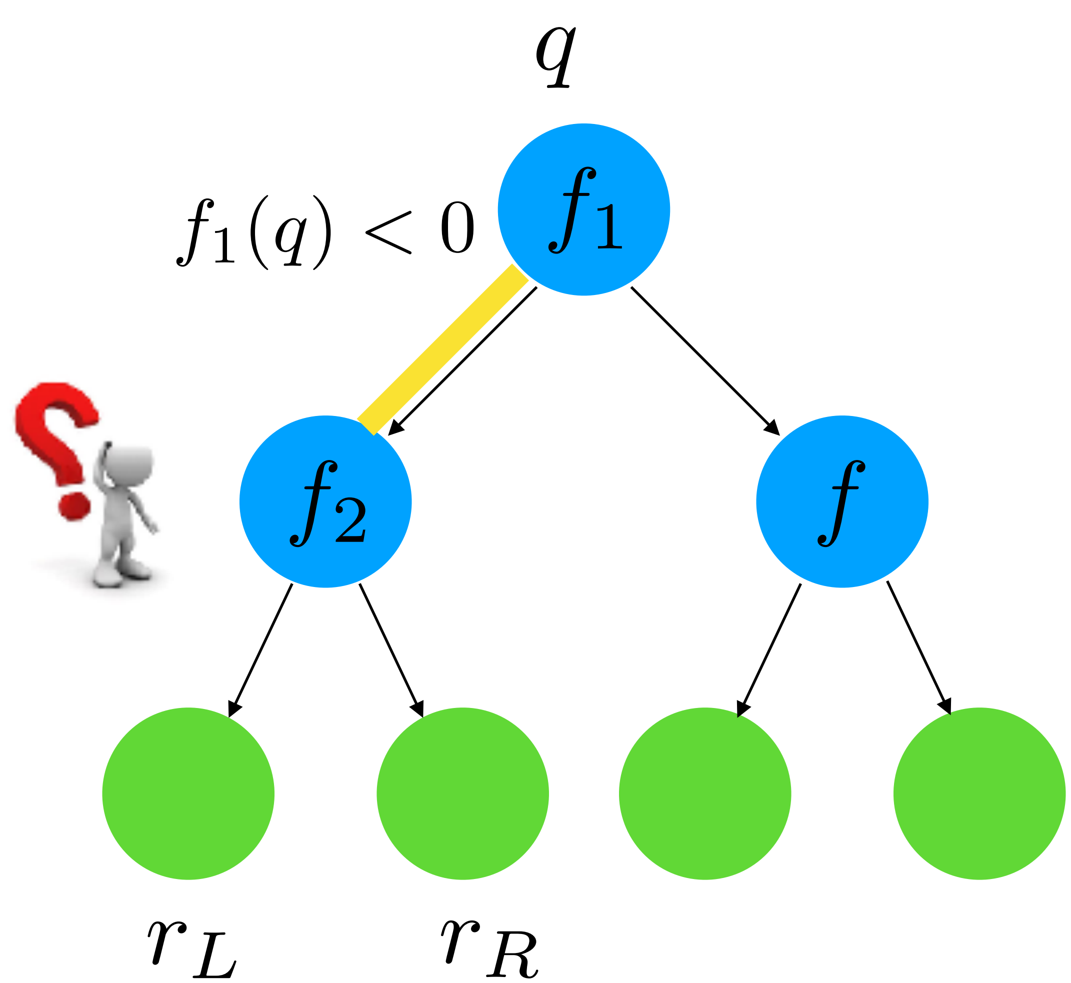
Update Routers:

Using reward signals to update routers



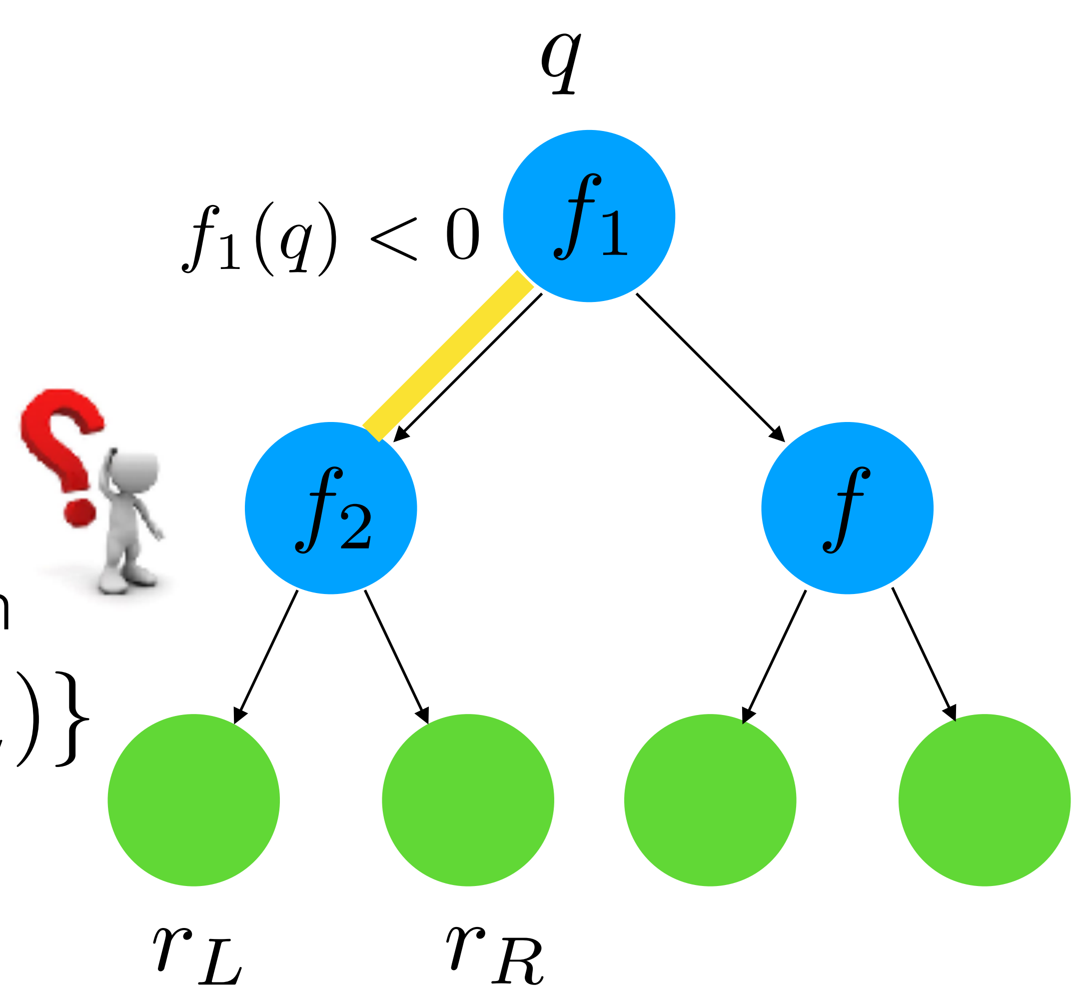
Update Routers:

Using reward signals to update routers



Update Routers:

Using reward signals to update routers

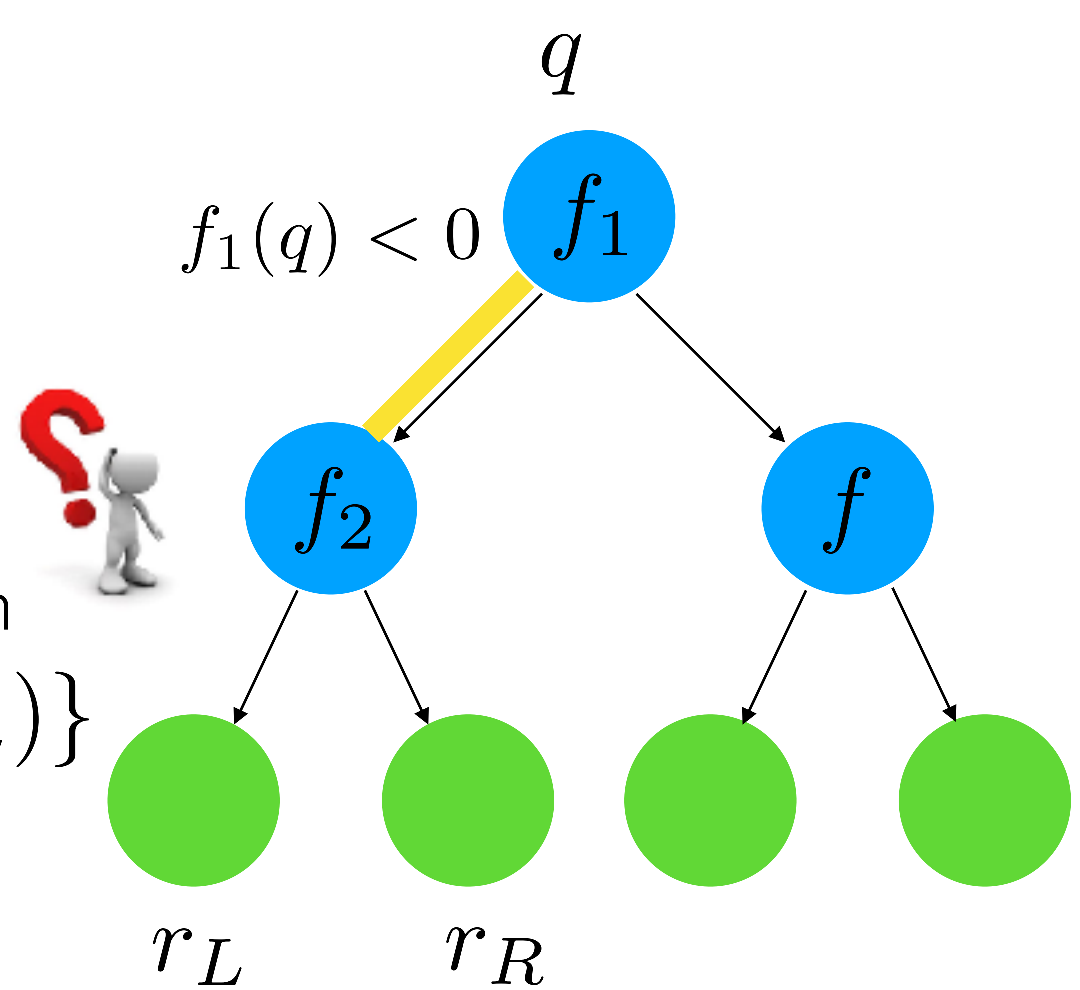


Binary Classification

$$\{q, \mathrm{sign}(r_R - r_L)\}$$

Update Routers:

Using reward signals to update routers



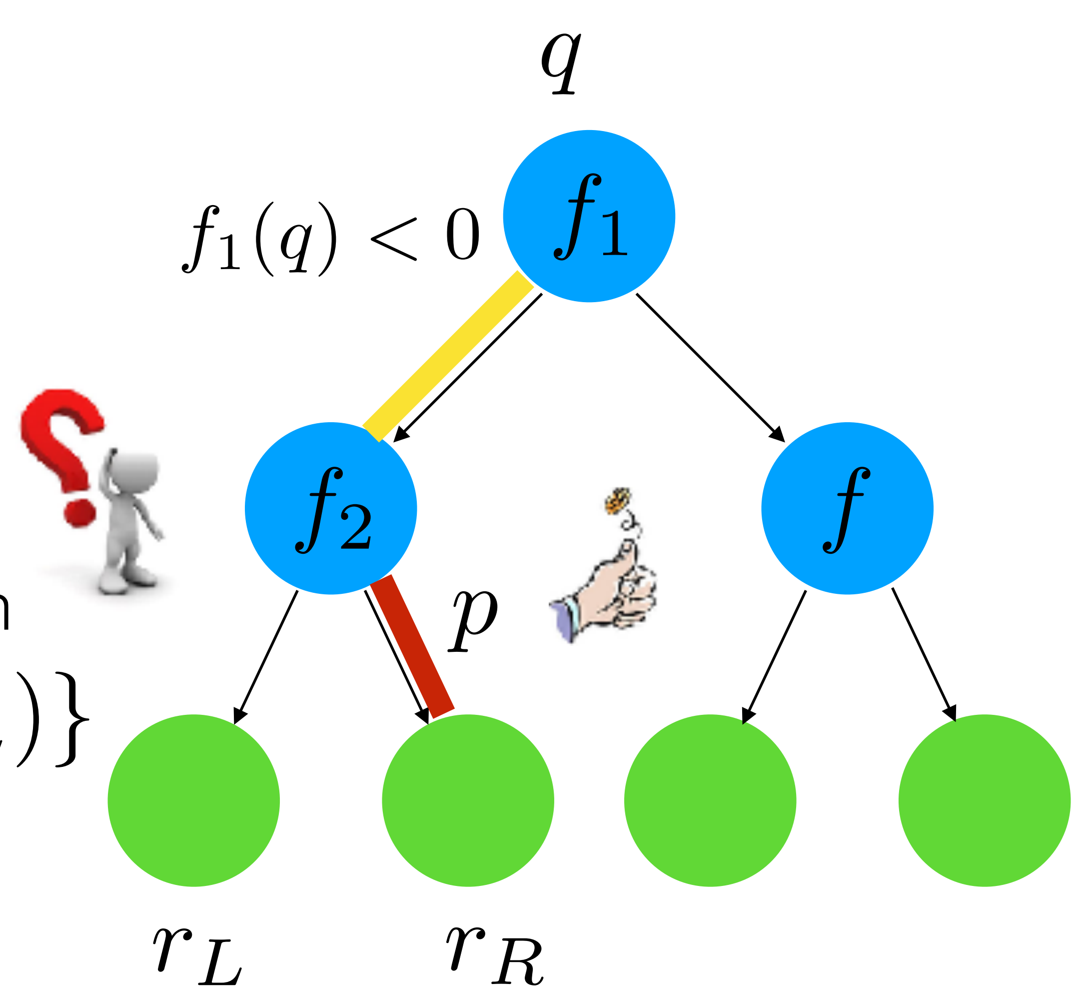
Binary Classification

$$\{q, \mathrm{sign}(r_R - r_L)\}$$

ϵ **-Greedy for exploration**

Update Routers:

Using reward signals to update routers



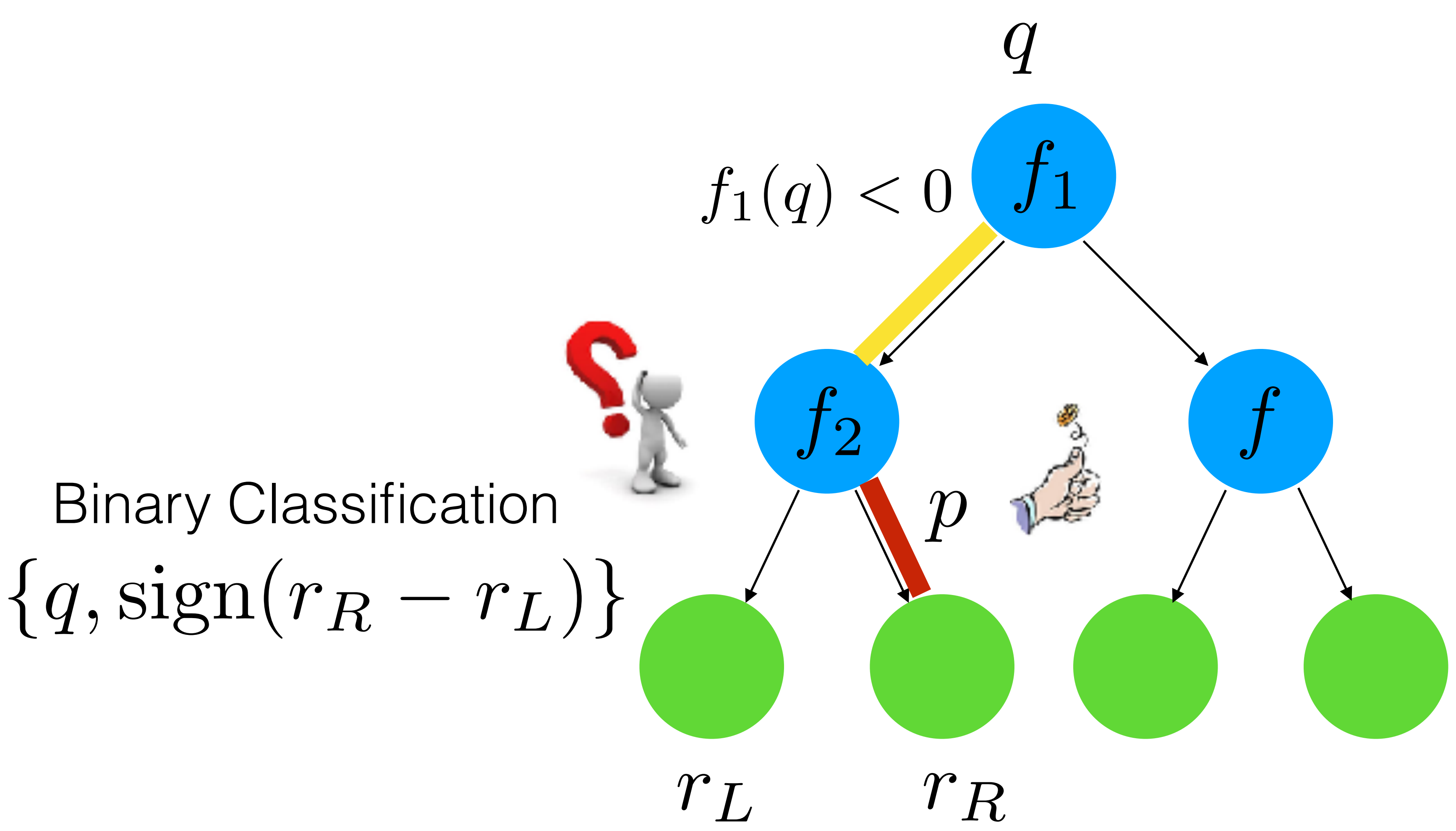
Binary Classification

$$\{q, \mathrm{sign}(r_R - r_L)\}$$

ϵ -Greedy for exploration

Update Routers:

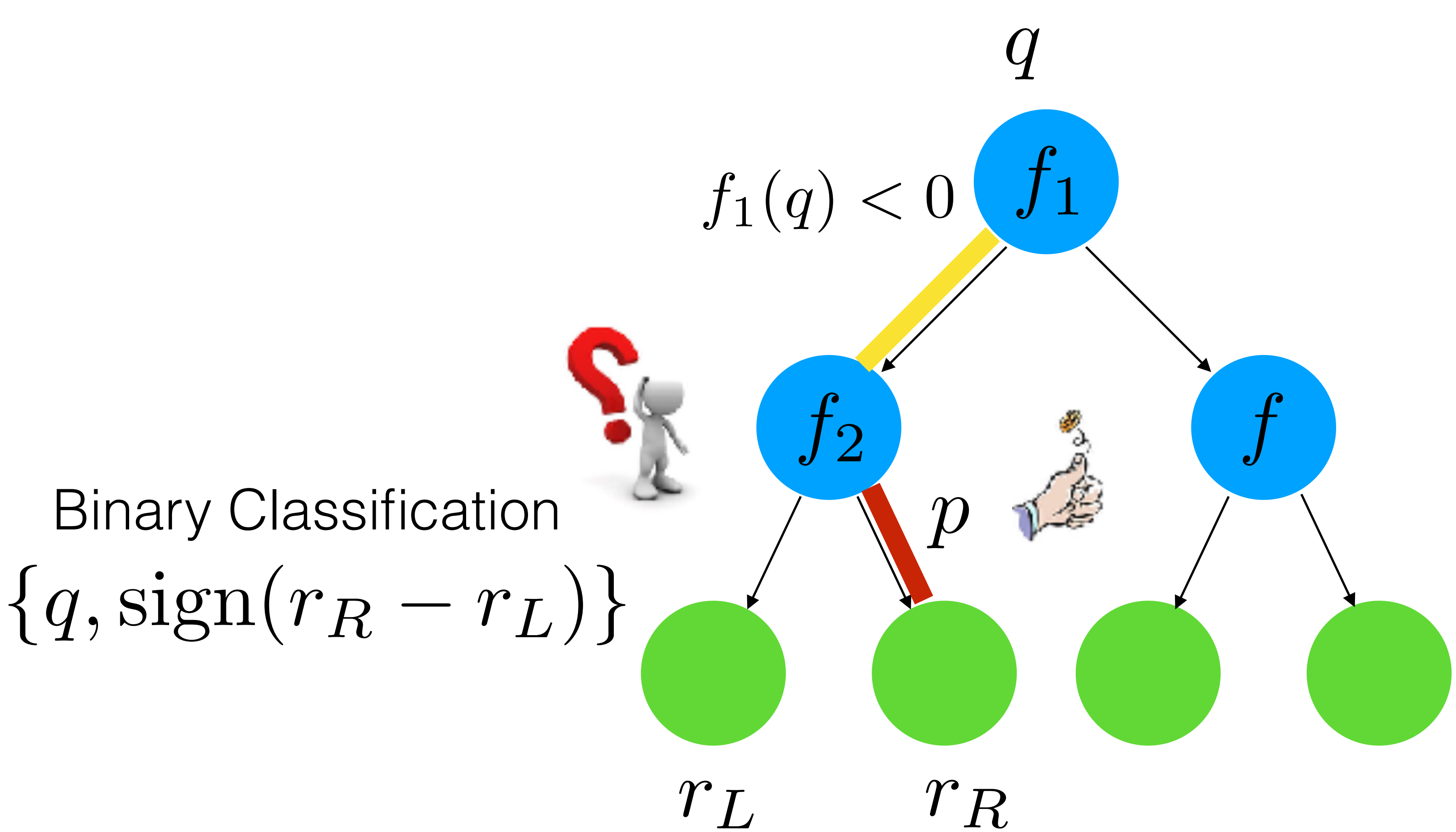
Using reward signals to update routers



ϵ **-Greedy for exploration** + **Importance Weight** (Propensity Score)

Update Routers:

Using reward signals to update routers

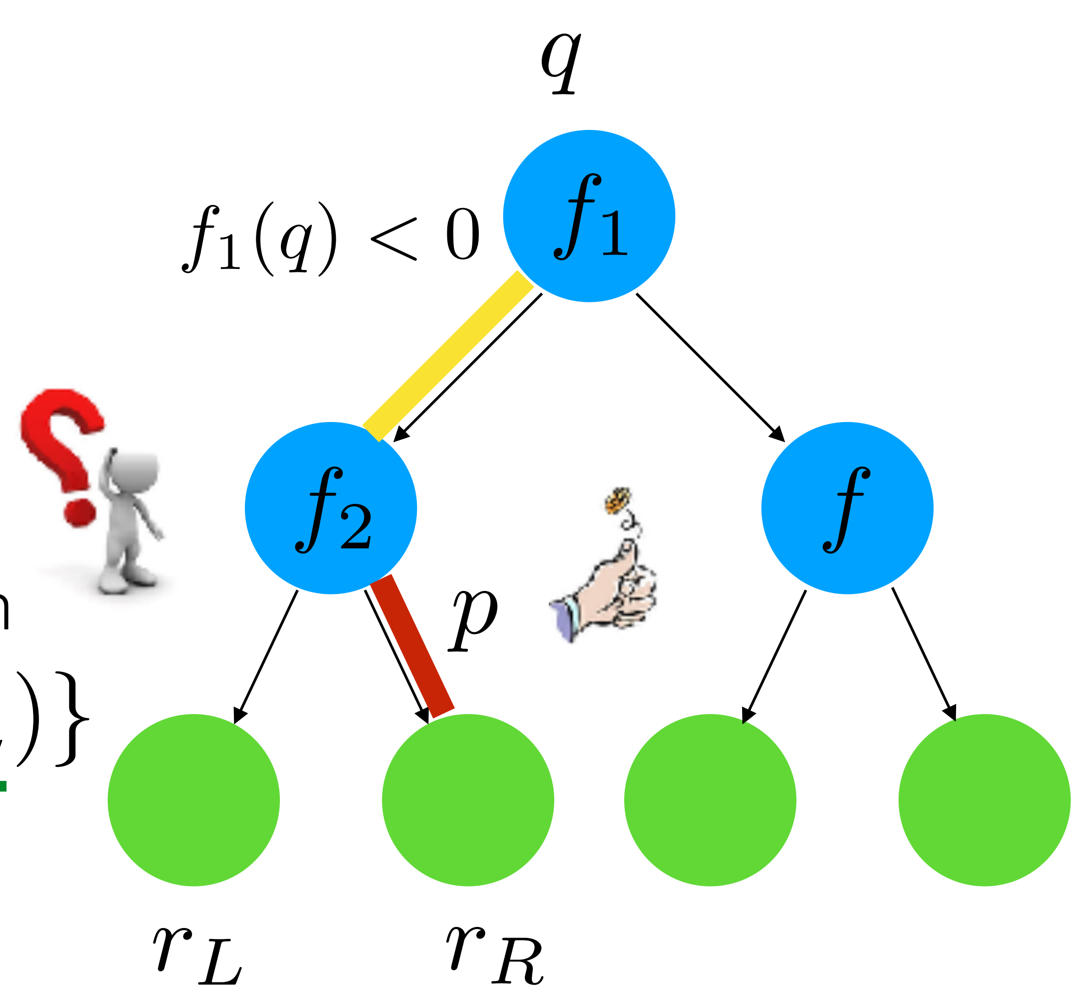


ϵ **-Greedy for exploration + Importance Weight** (Propensity Score)

$$\frac{r_R}{p} \approx r_R - r_L$$

Update Routers:

Using reward signals to update routers



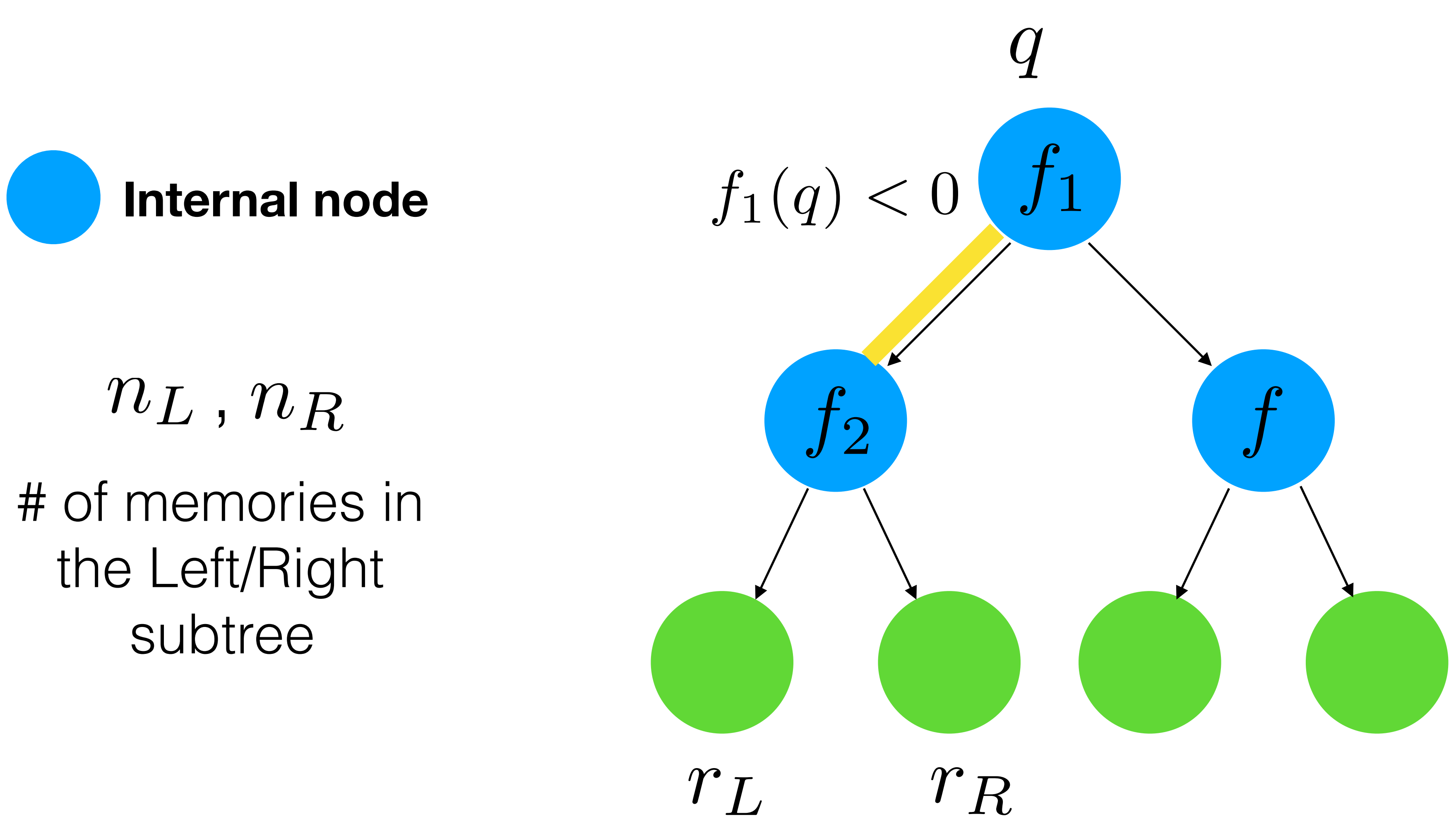
Binary Classification

$$\{q, \mathrm{sign}(\underline{r_R - r_L})\}$$

ϵ **-Greedy for exploration** + **Importance Weight** (Propensity Score)

$$\frac{r_R}{p} \approx r_R - r_L$$

Update Routers:
Add Regularization for **Balance**



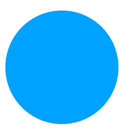
$$n_L\ ,n_R$$

# of memories in the Left/Right subtree

Binary Classification

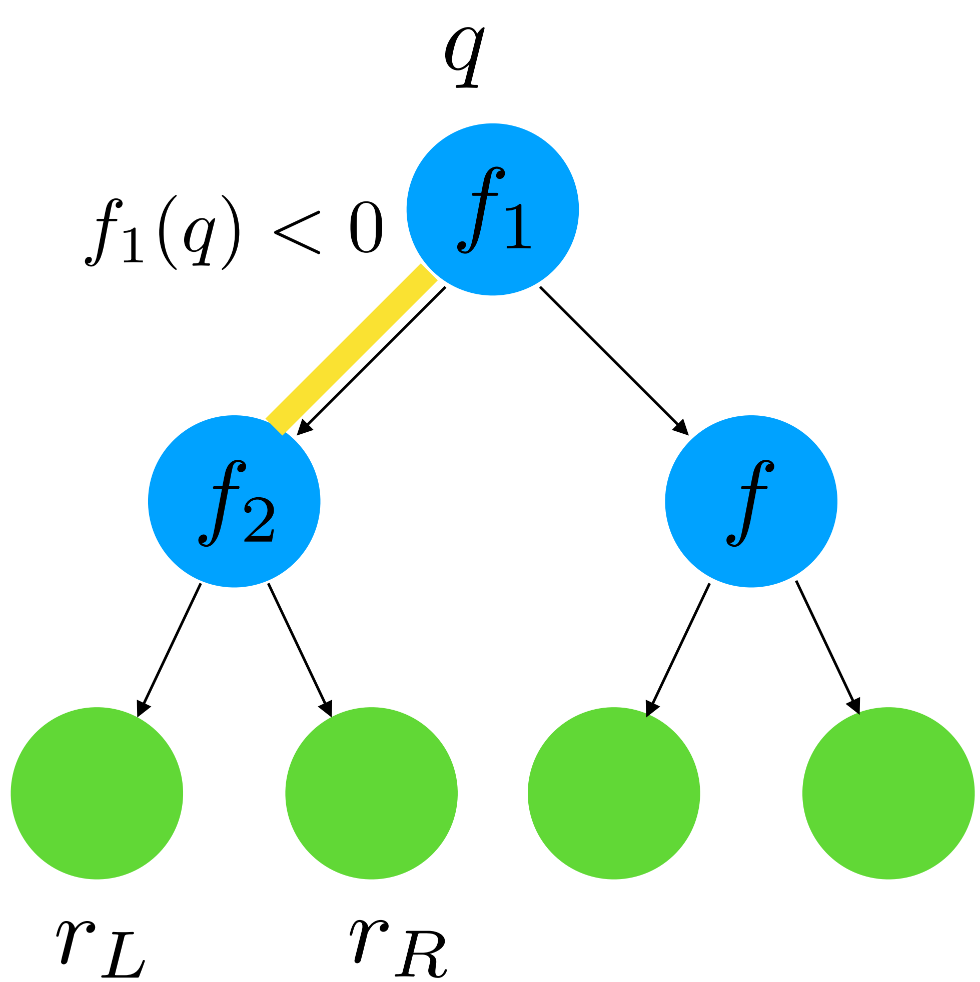
$$\{q, \mathrm{sign}(r_R - r_L)\}$$

Update Routers:
Add Regularization for **Balance**



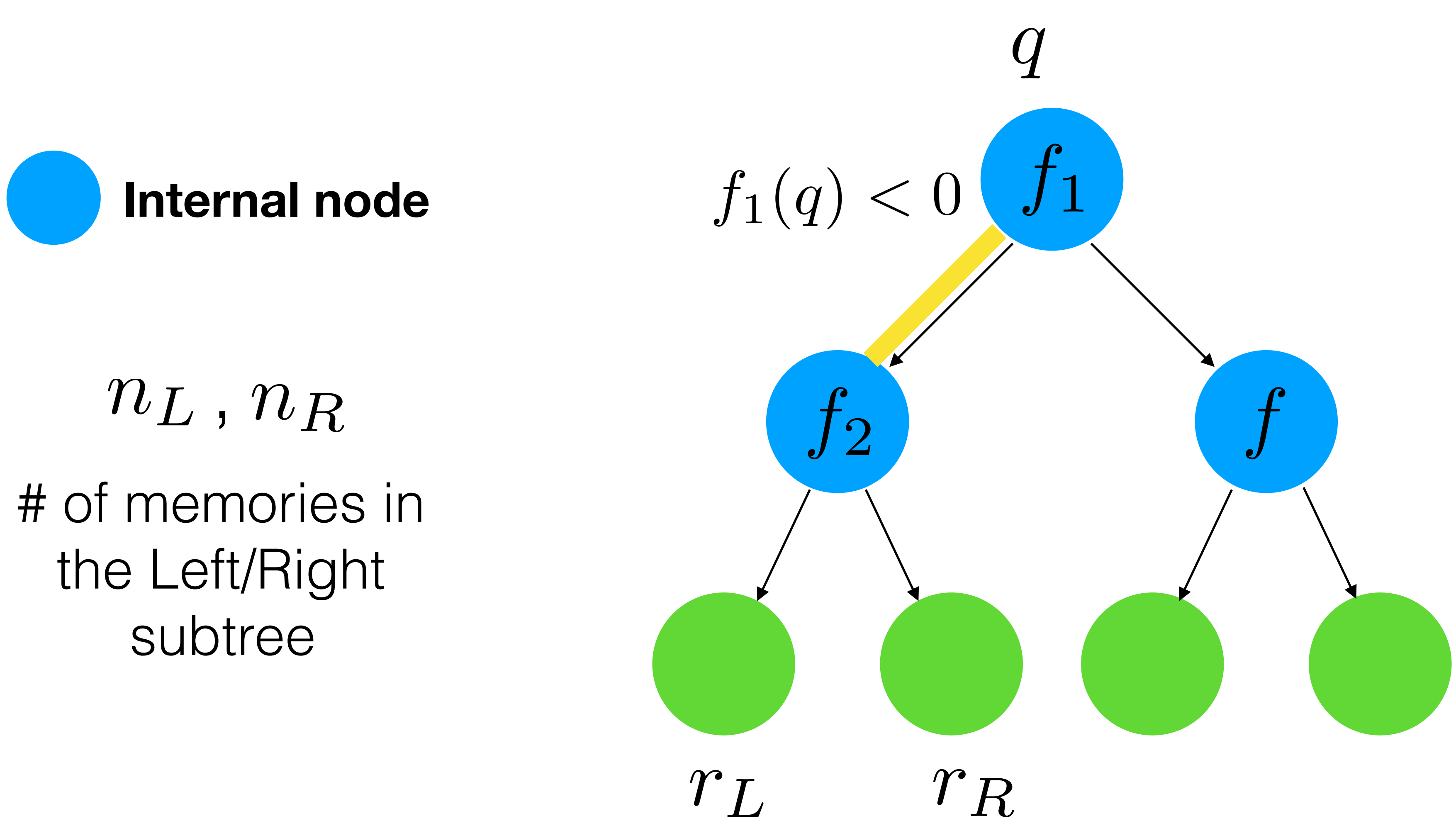
n_L , n_R

# of memories in the Left/Right subtree



Binary Classification

Update Routers:
Add Regularization for **Balance**



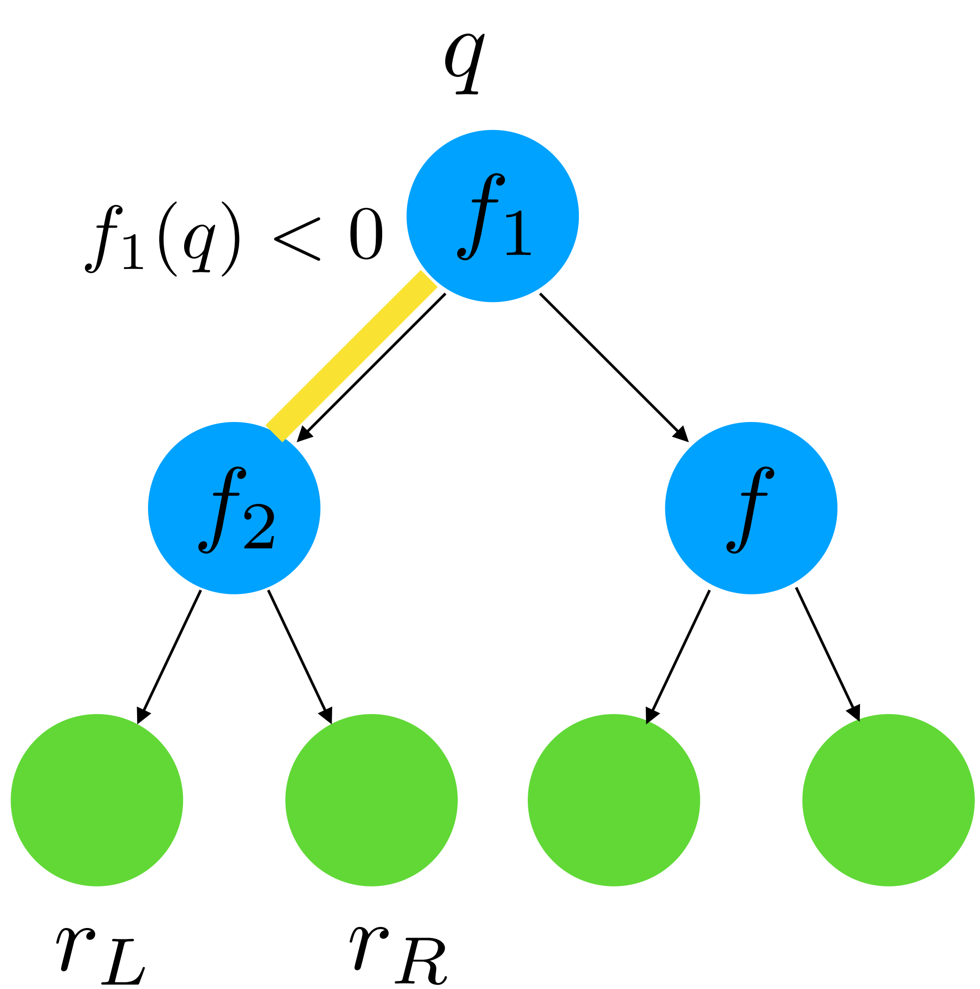
n_L , n_R

# of memories in the Left/Right subtree

Binary Classification

$$\left\{ q, \operatorname{sign}\left((1-\alpha)(r_R - r_L) + \alpha \log \frac{n_L}{n_R} \right) \right\}$$

Update Routers:
Add Regularization for **Balance**



Internal node

n_L , n_R

# of memories in the Left/Right subtree

Binary Classification

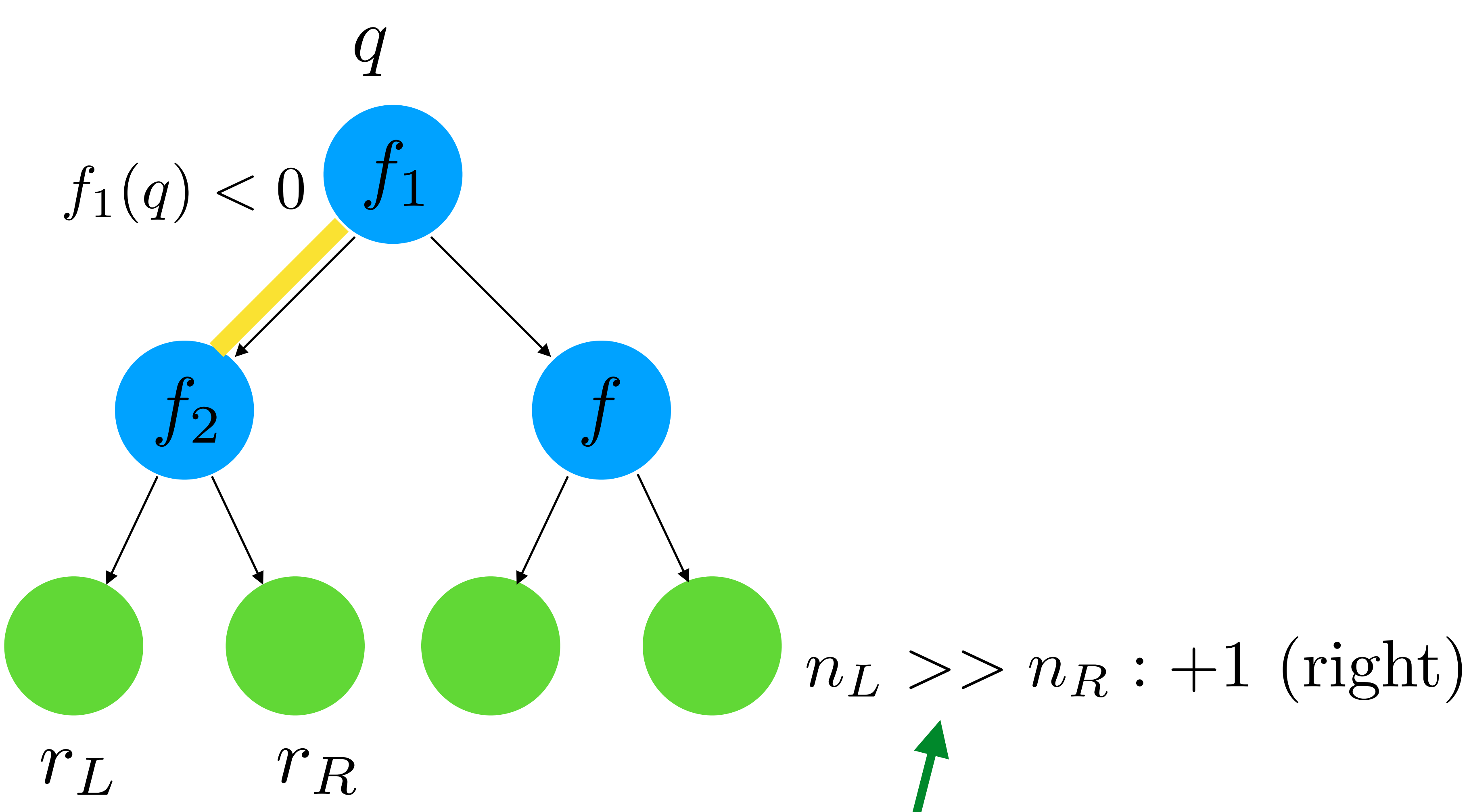
$$\left\{q, \operatorname{sign}\left((1-\alpha)(r_R - r_L) + \alpha \log \frac{n_L}{n_R}\right)\right\}$$

Update Routers:
Add Regularization for **Balance**

Internal node

$n_L \, , n_R$

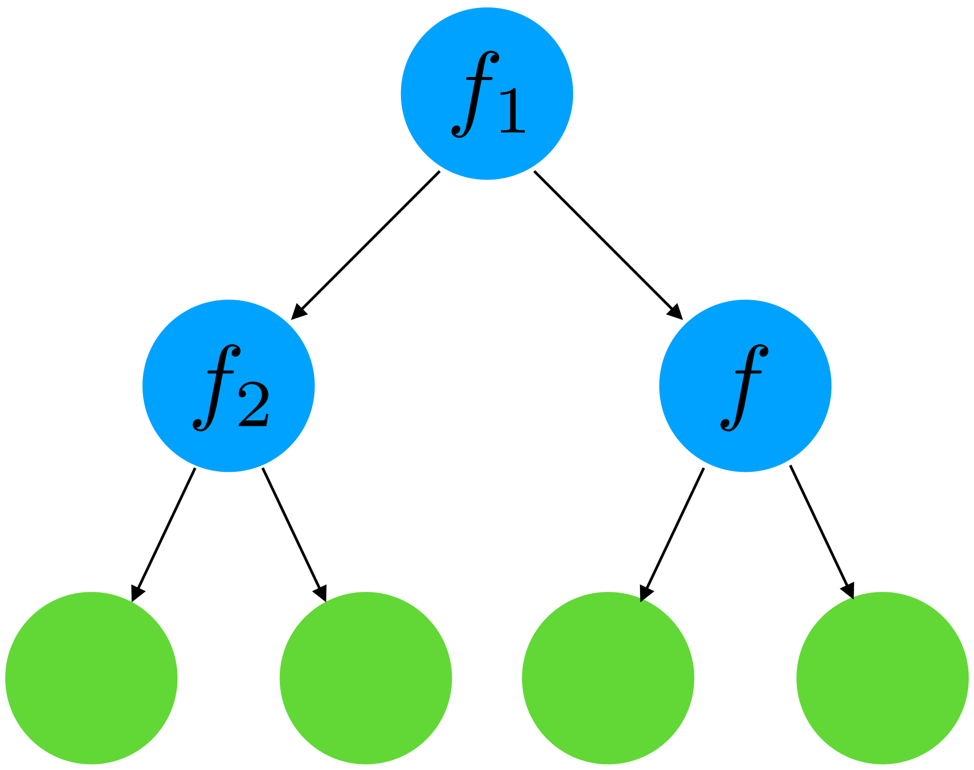
# of memories in the Left/Right subtree



Binary Classification

$$\left\{ q, \mathrm{sign}\left((1-\alpha)(r_R - r_L) + \alpha \log \frac{n_L}{n_R} \right) \right\}$$

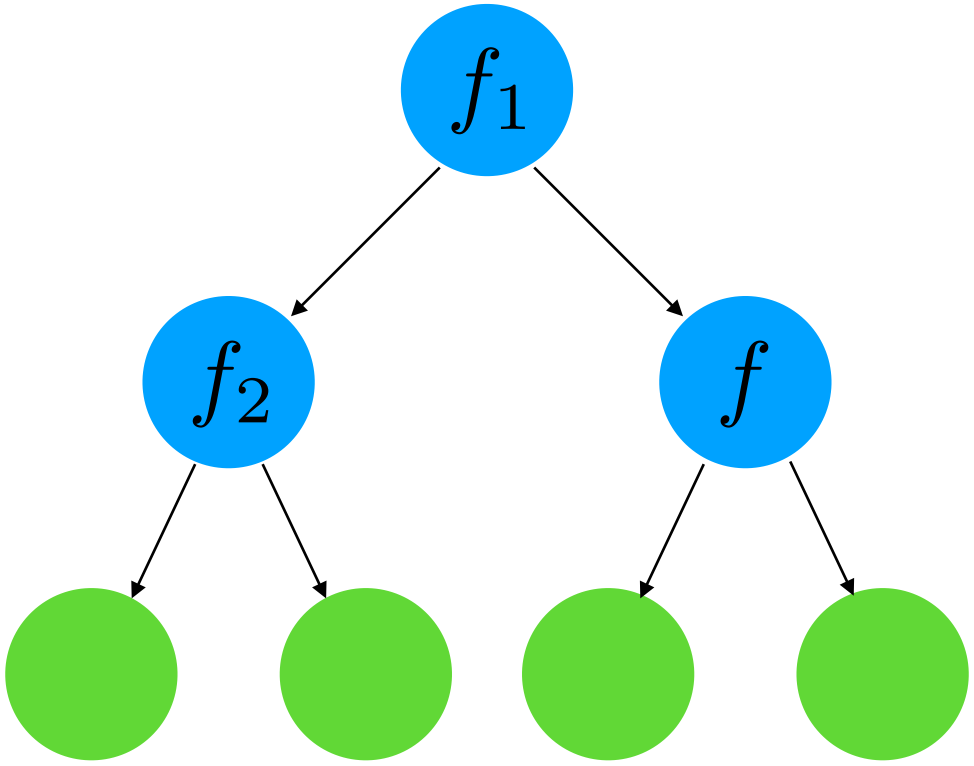
Insertion:
Self-Consistency



$m = \{q, v\}$

Insertion: Self-Consistency

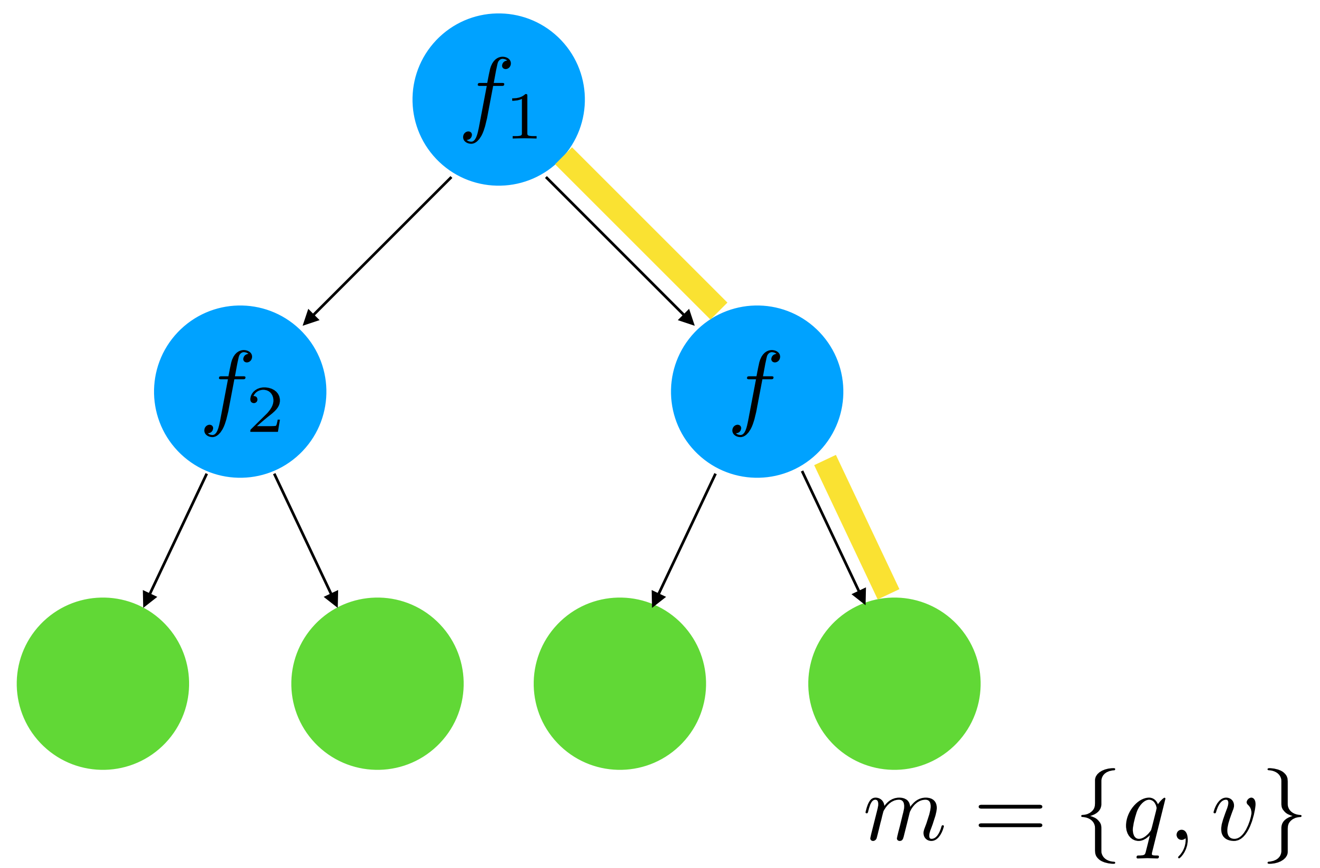
Same query: q



$m = \{q, v\}$

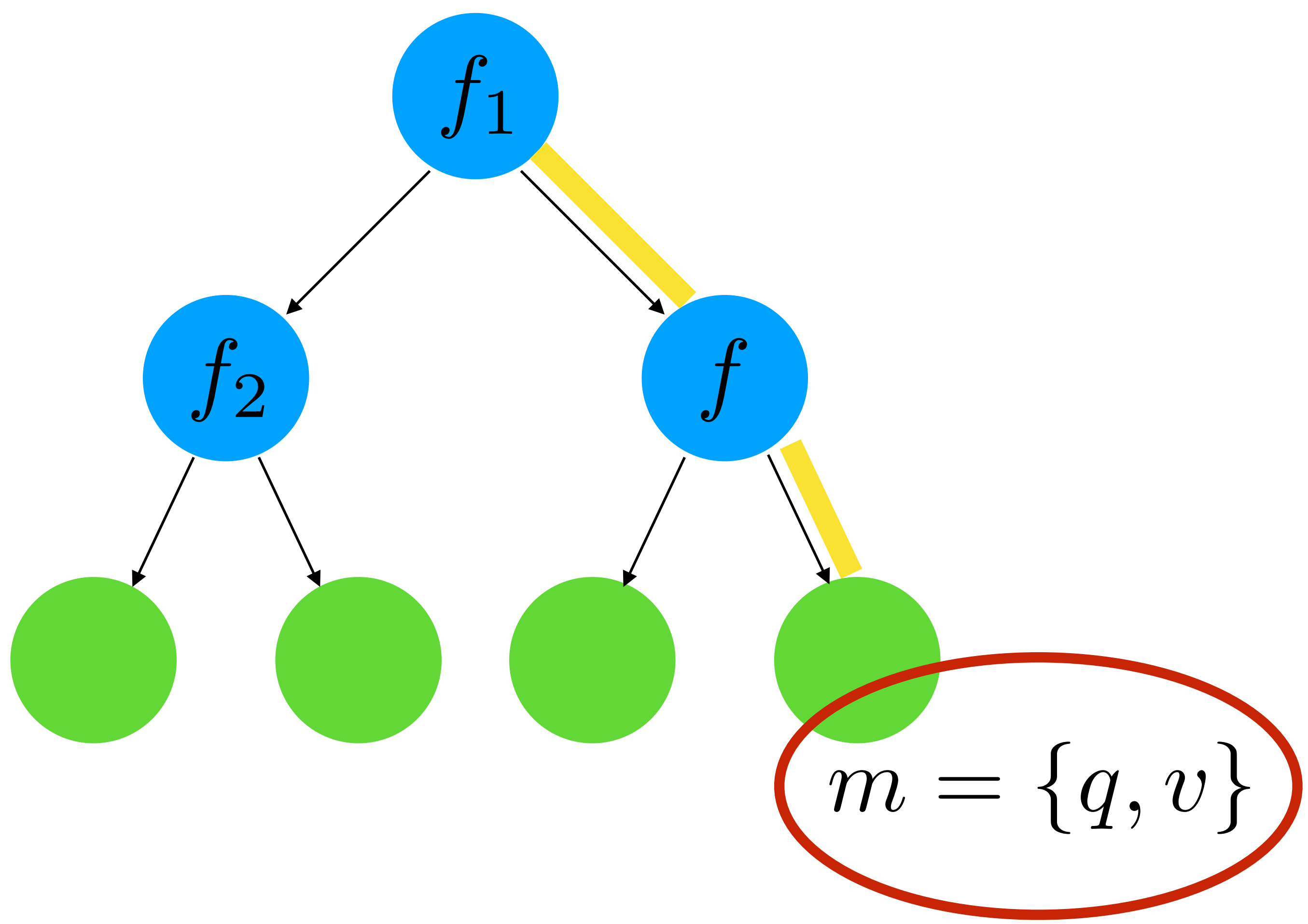
Insertion: Self-Consistency

Same query: q

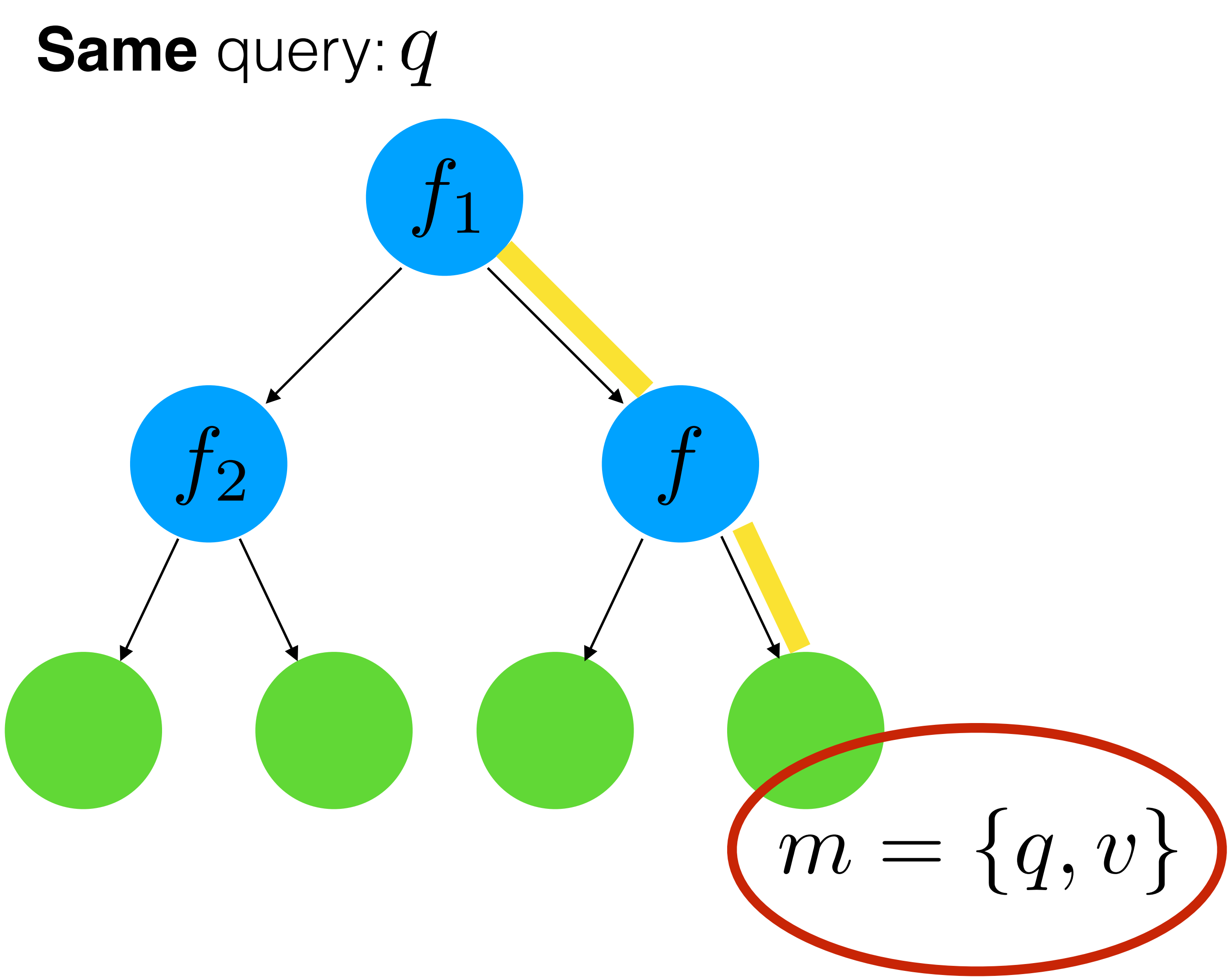


Insertion: Self-Consistency

Same query: q



Insertion: Self-Consistency



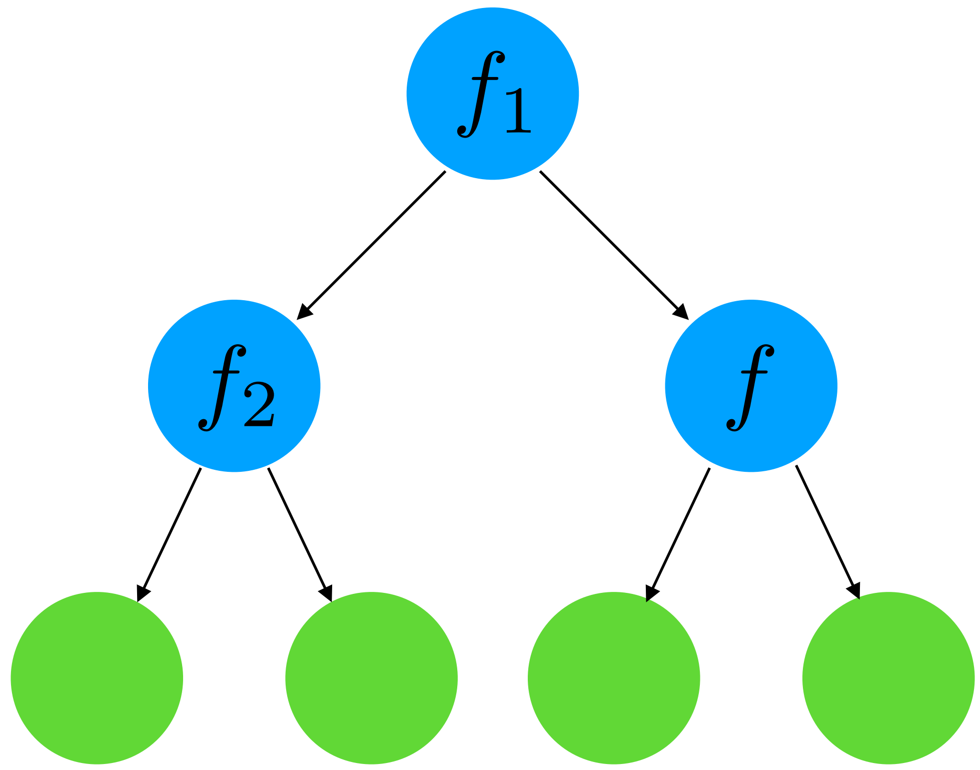
If a query has been seen before, we want to just retrieve it!

Insertion:
Encourage **Self-Consistency**

Internal node

$$n_L\,,\,n_R$$

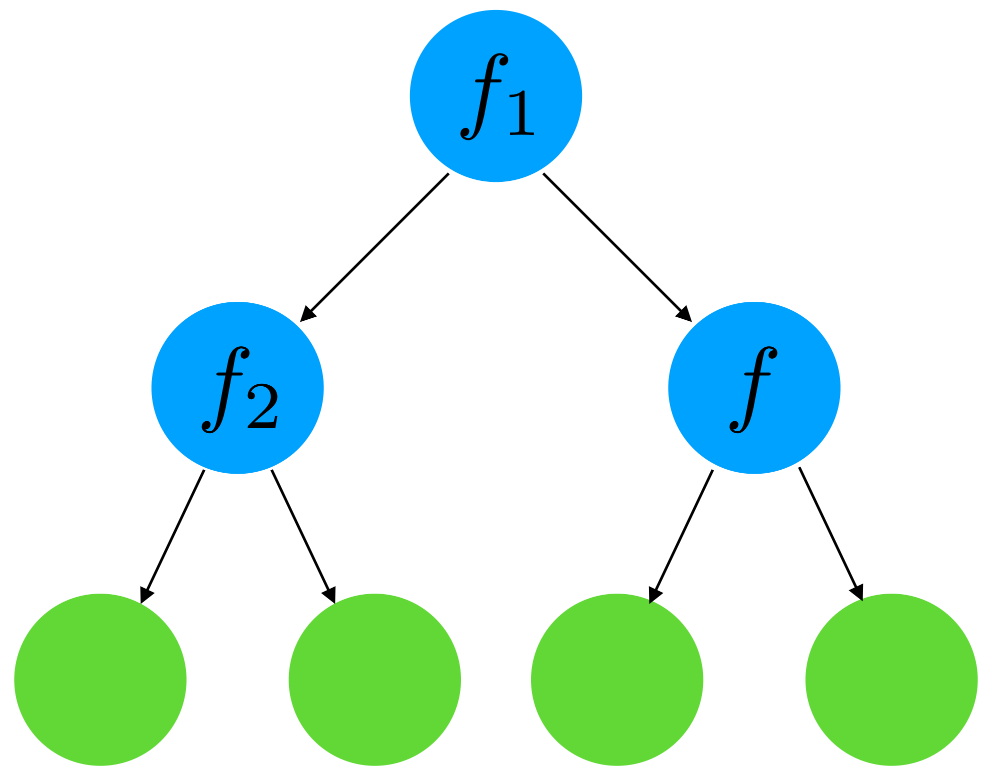
# of memories in the Left/Right subtree



Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$

Internal node

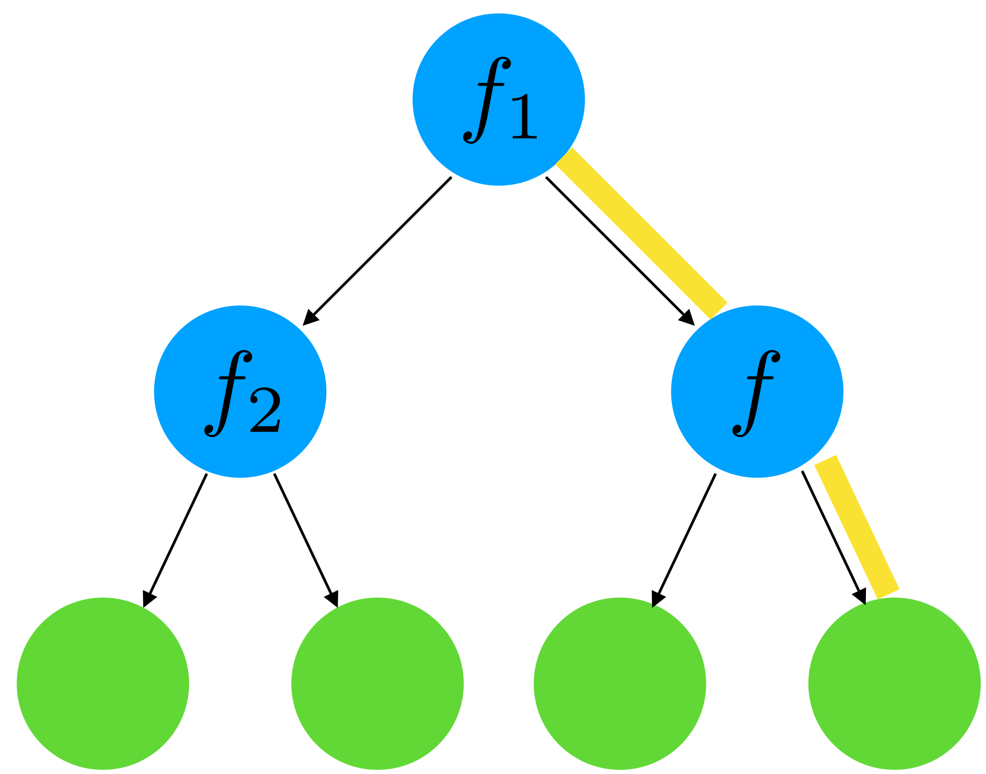


$n_L \, , n_R$

# of memories in the Left/Right subtree

Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$



Internal node

$n_L \, , n_R$

# of memories in the Left/Right subtree

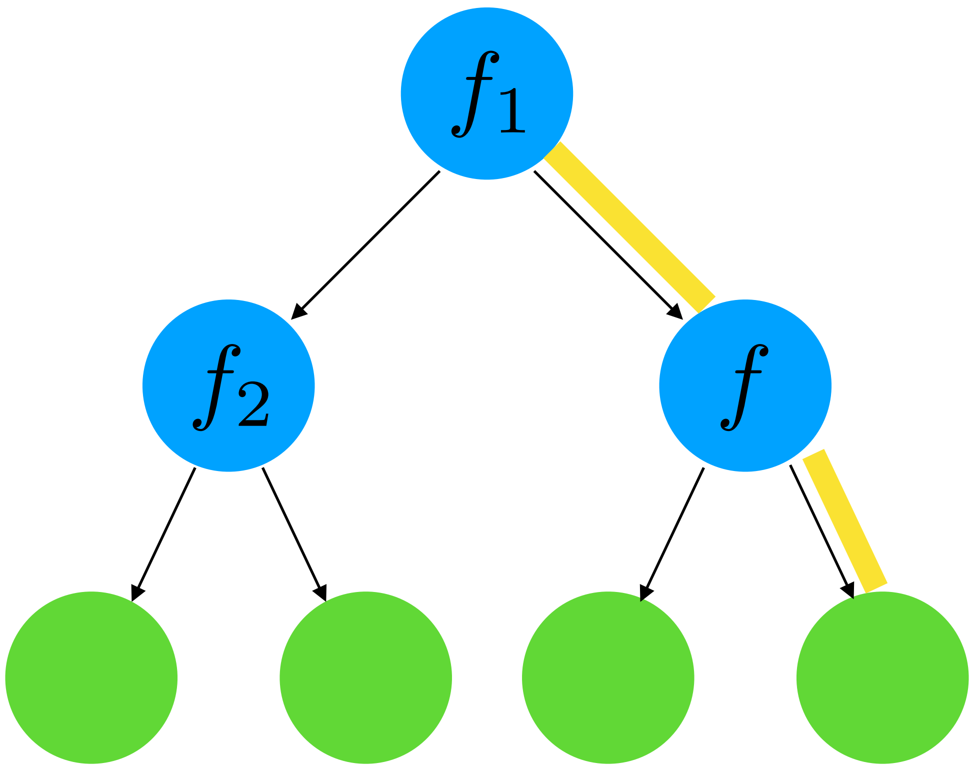
Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$

Internal node

$n_L \, , n_R$

# of memories in the Left/Right subtree



$m = \{q, v\}$

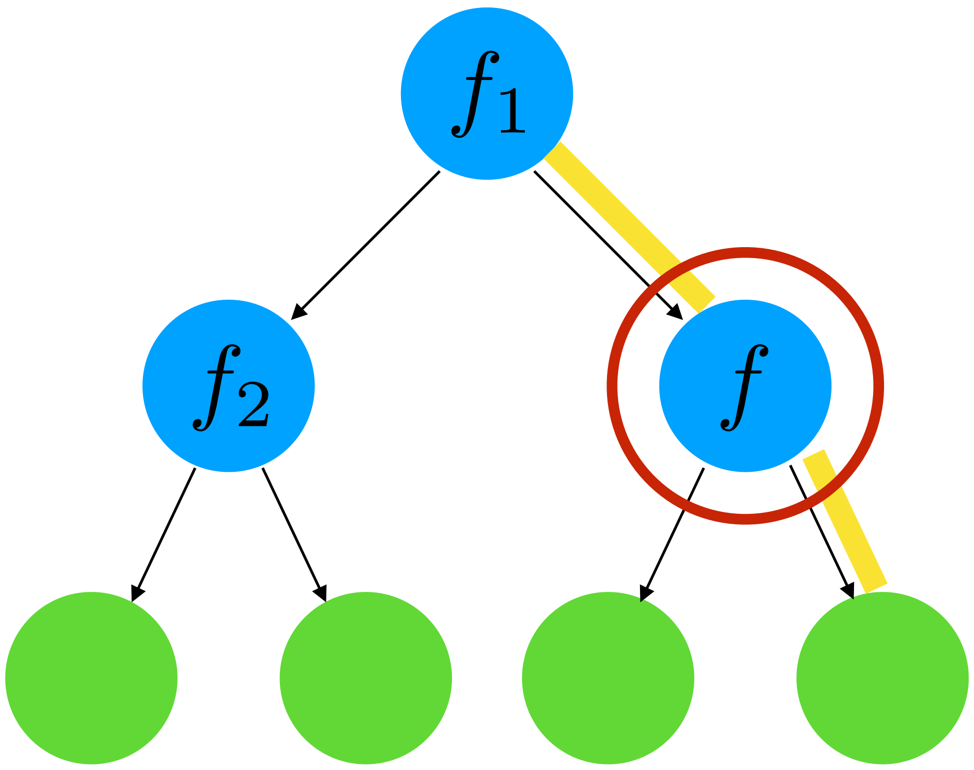
Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$

Internal node

$n_L\ ,n_R$

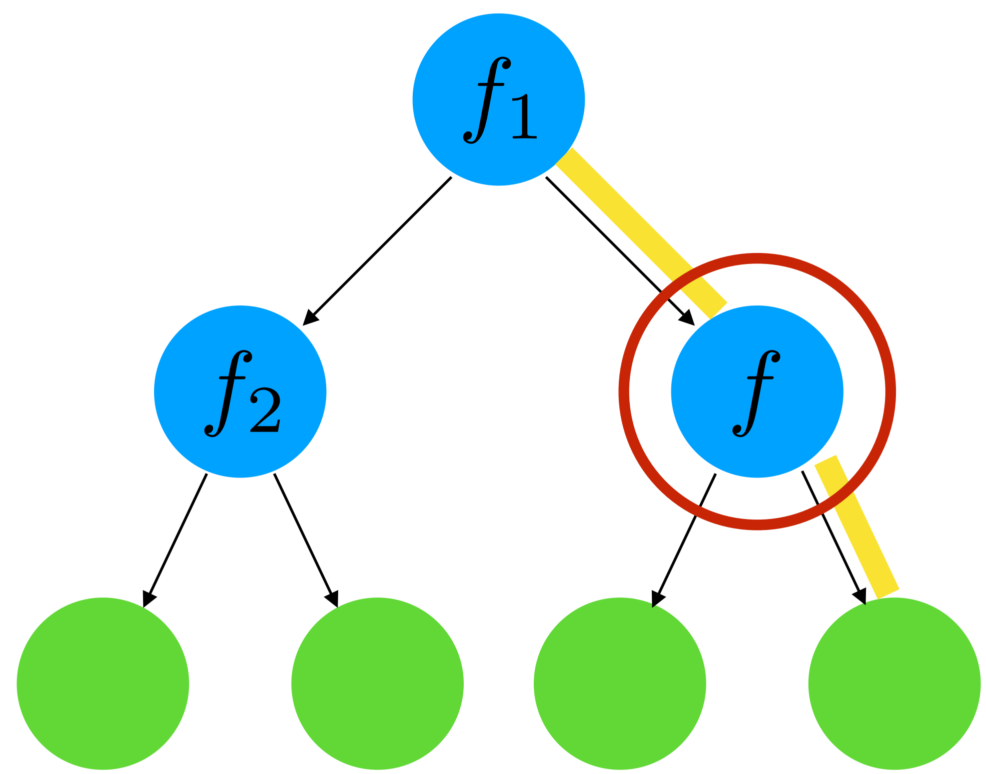
# of memories in the Left/Right subtree



$m = \{q, v\}$

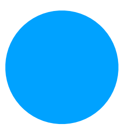
Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$



Enhance its prediction

$m = \{q, v\}$



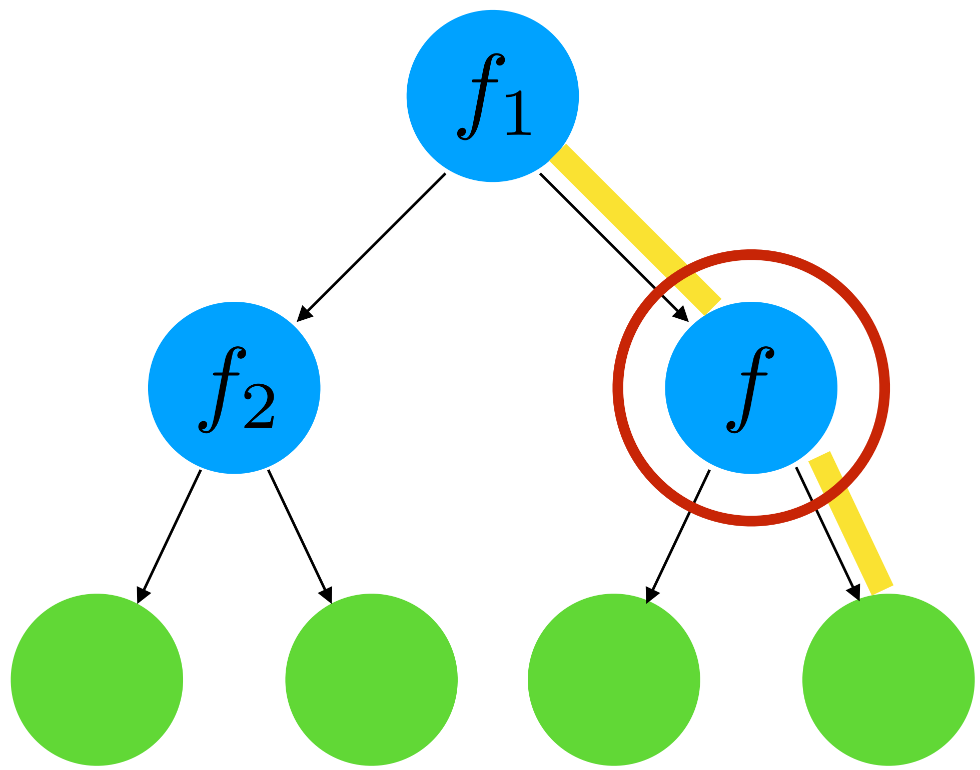
Internal node

$n_L \, , n_R$

# of memories in the Left/Right subtree

Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$



Enhance its prediction

$m = \{q, v\}$

Internal node

n_L , n_R

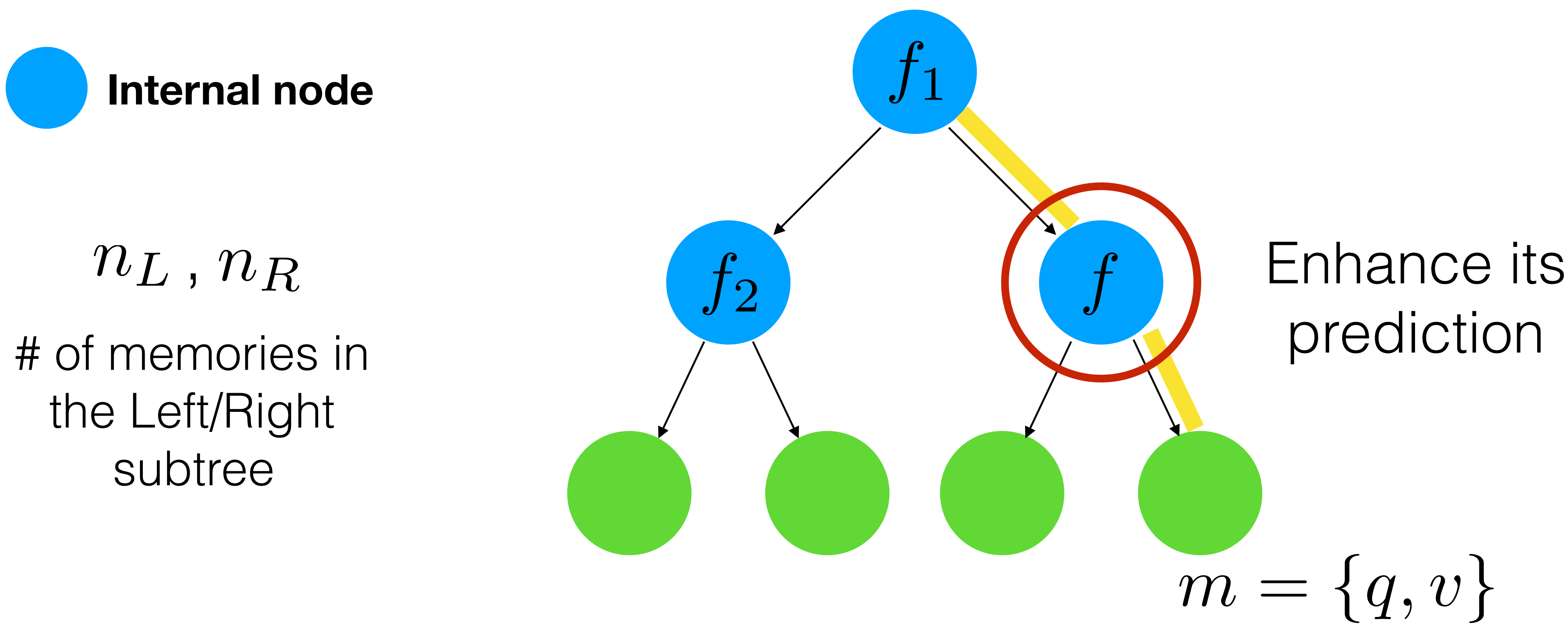
# of memories in the Left/Right subtree

Binary Classification

$$\left\{ q, \operatorname{sign}\left((1-\alpha) f(g) + \alpha \log\left(\frac{n_L}{n_R} \right) \right) \right\}$$

Insertion:
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$



n_L , n_R

# of memories in the Left/Right subtree

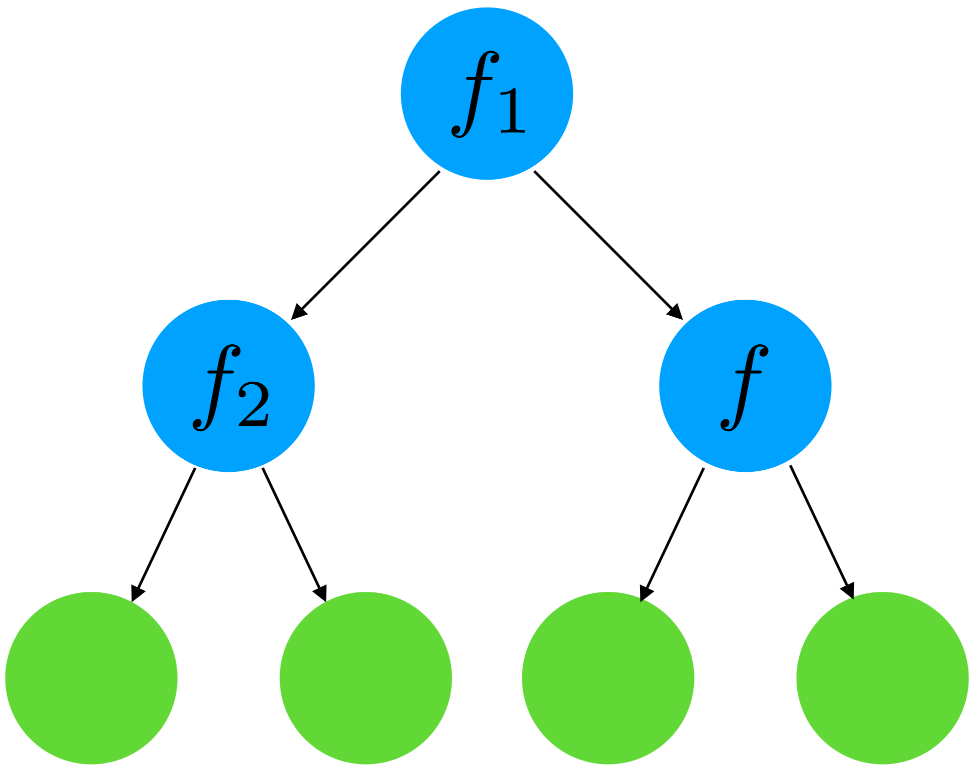
Binary Classification

$$\left\{ q, \operatorname{sign}\left((1-\alpha) f(g) + \alpha \log\left(\frac{n_L}{n_R} \right) \right) \right\}$$

Router's prediction

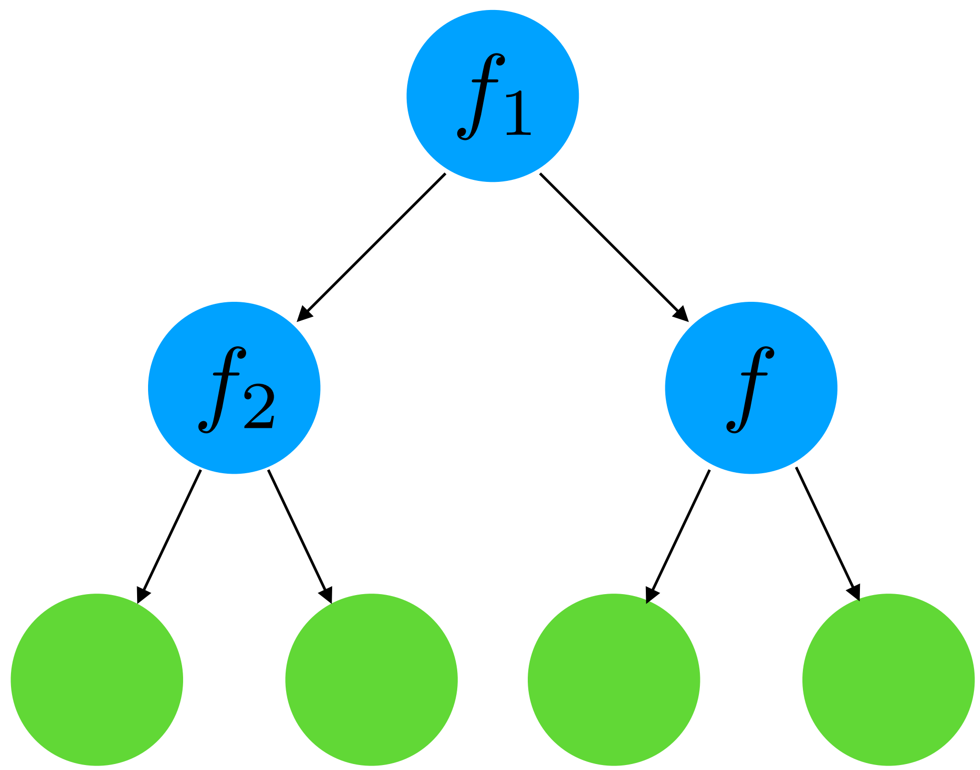
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions



Amortized Experience Replay

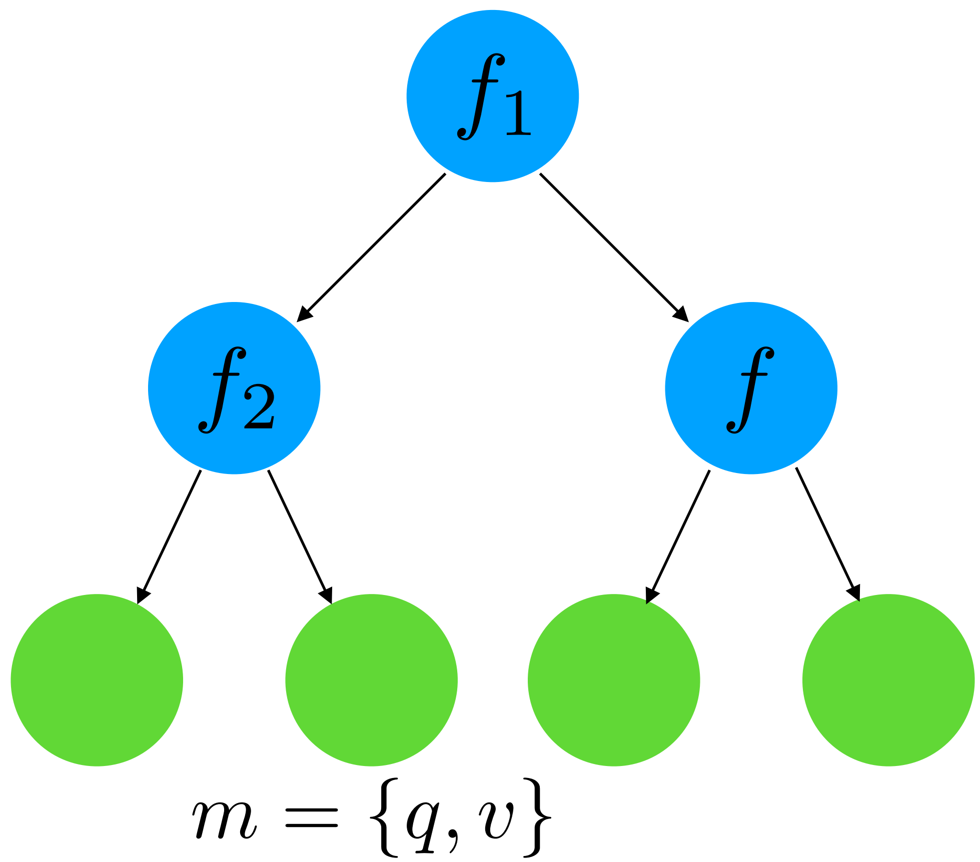
Routers are updated online and may result in a lack of self-consistency for previous insertions



1. Randomly sample a memory

Amortized Experience Replay

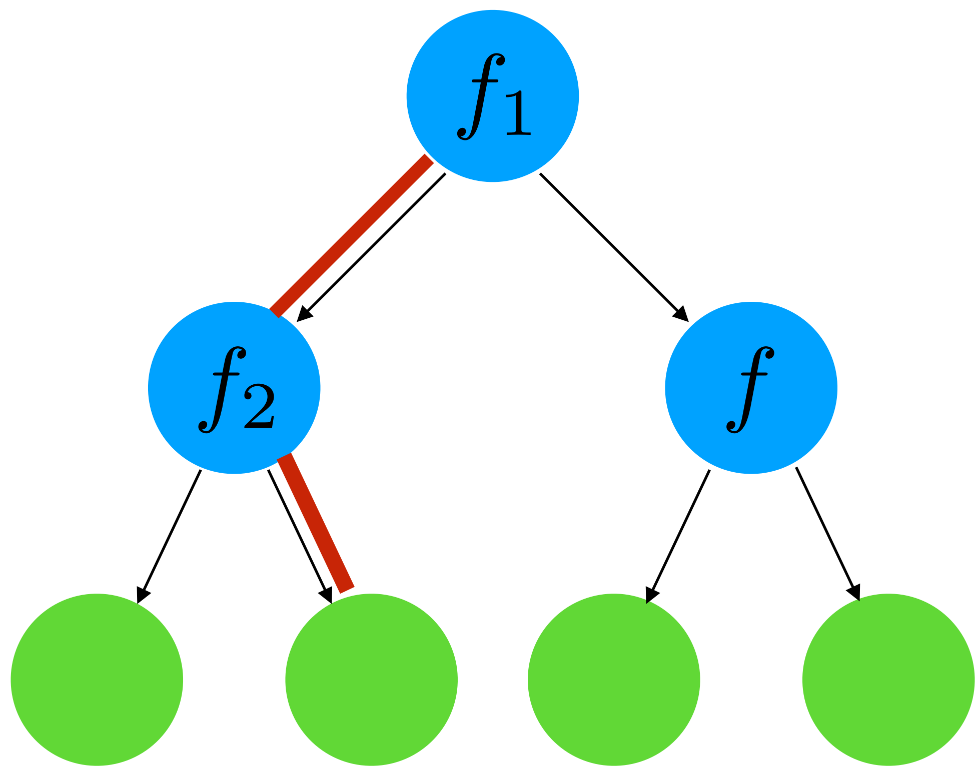
Routers are updated online and may result in a lack of self-consistency for previous insertions



1. Randomly sample a memory

Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions

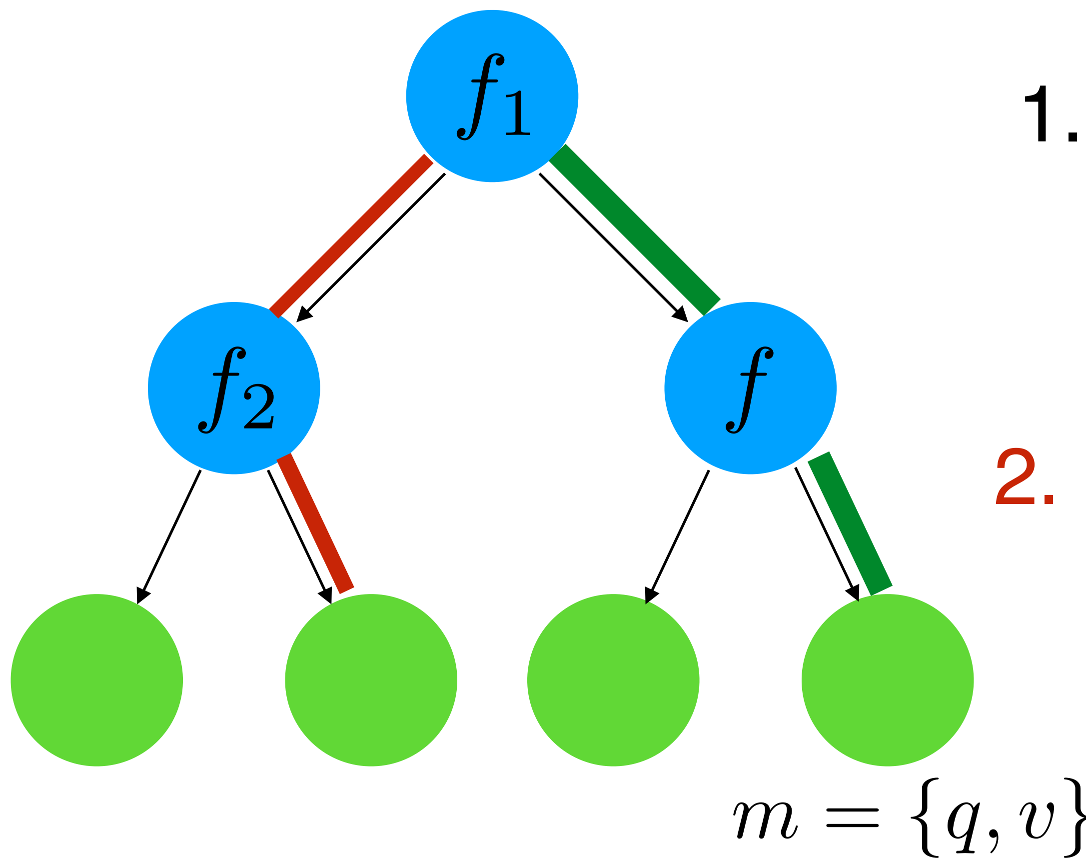


1. Randomly sample a memory

2. **Delete** it **&** its trace

Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions



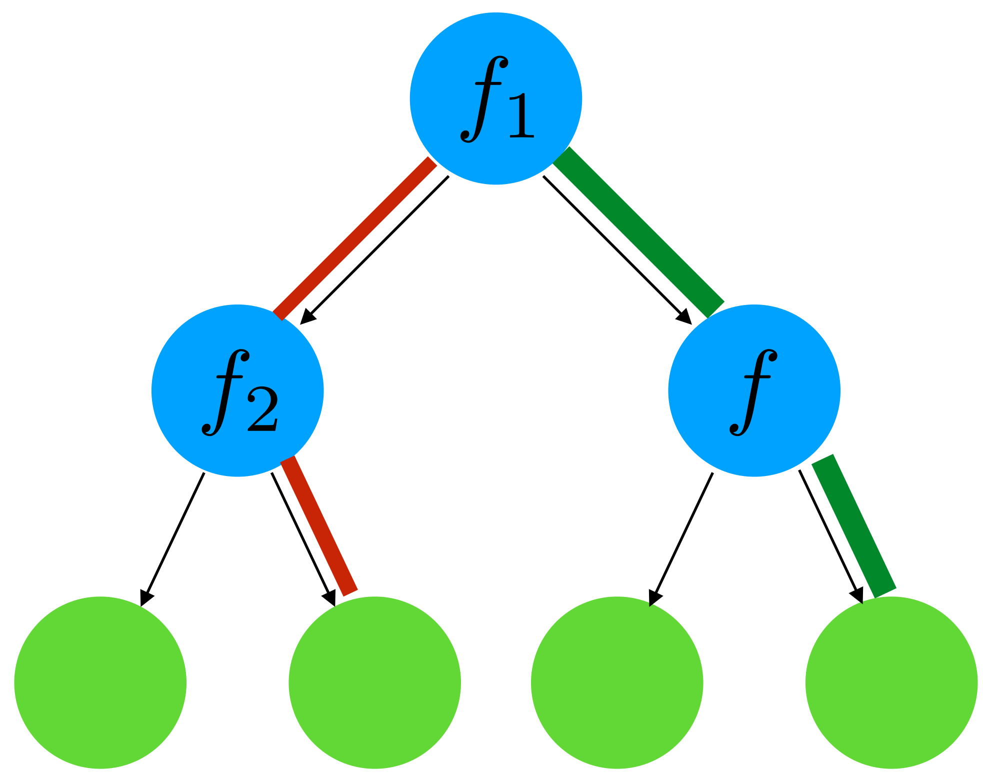
1. Randomly sample a memory

2. **Delete** it **&** its trace

3. Re-**insert** it

Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions



$m = \{q, v\}$

1. Randomly sample a memory

2. **Delete** it **&** its trace

3. Re-**insert** it

Apply replay constant times per insertion

Empirical Results

Empirical Results

Extreme Multi-Class Classification:

(feature, label), zero-one loss

Empirical Results

Extreme Multi-Class Classification:

(feature, label), zero-one loss

Extreme Multi-Label Classification:

(feature, set of labels), Hamming loss

Empirical Results

Extreme Multi-Class Classification:

(feature, label), zero-one loss

Extreme Multi-Label Classification:

(feature, set of labels), Hamming loss

Image Retrieval

(Caption, Image), Cosine Similarity

Extreme Multi-class Classification:
Online Progressive Performance

Extreme Multi-class Classification:
Online Progressive Performance

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

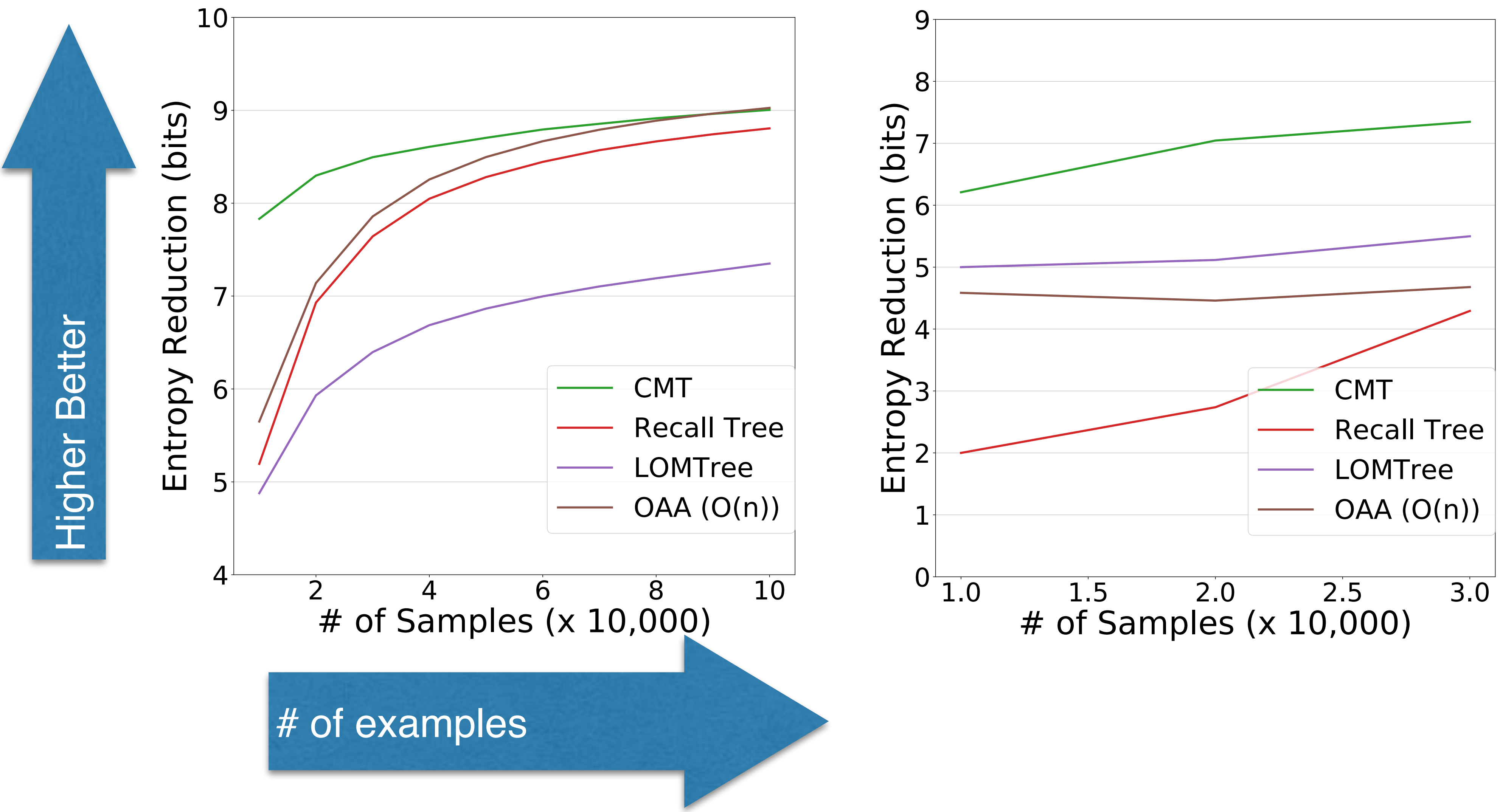
Extreme Multi-class Classification:
Online Progressive Performance

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!

Extreme Multi-class Classification:
Online Progressive Performance

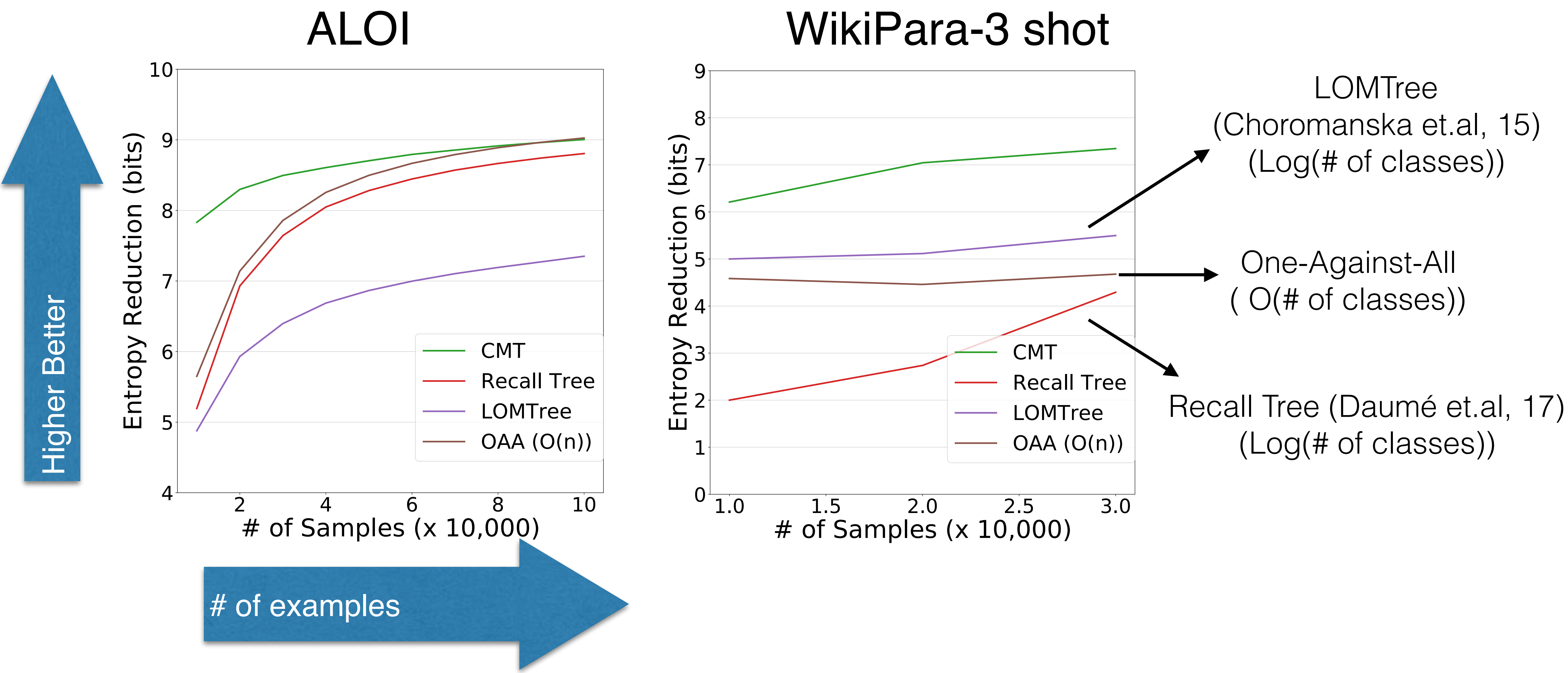
ALOI　　　　WikiPara-3 shot



ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!

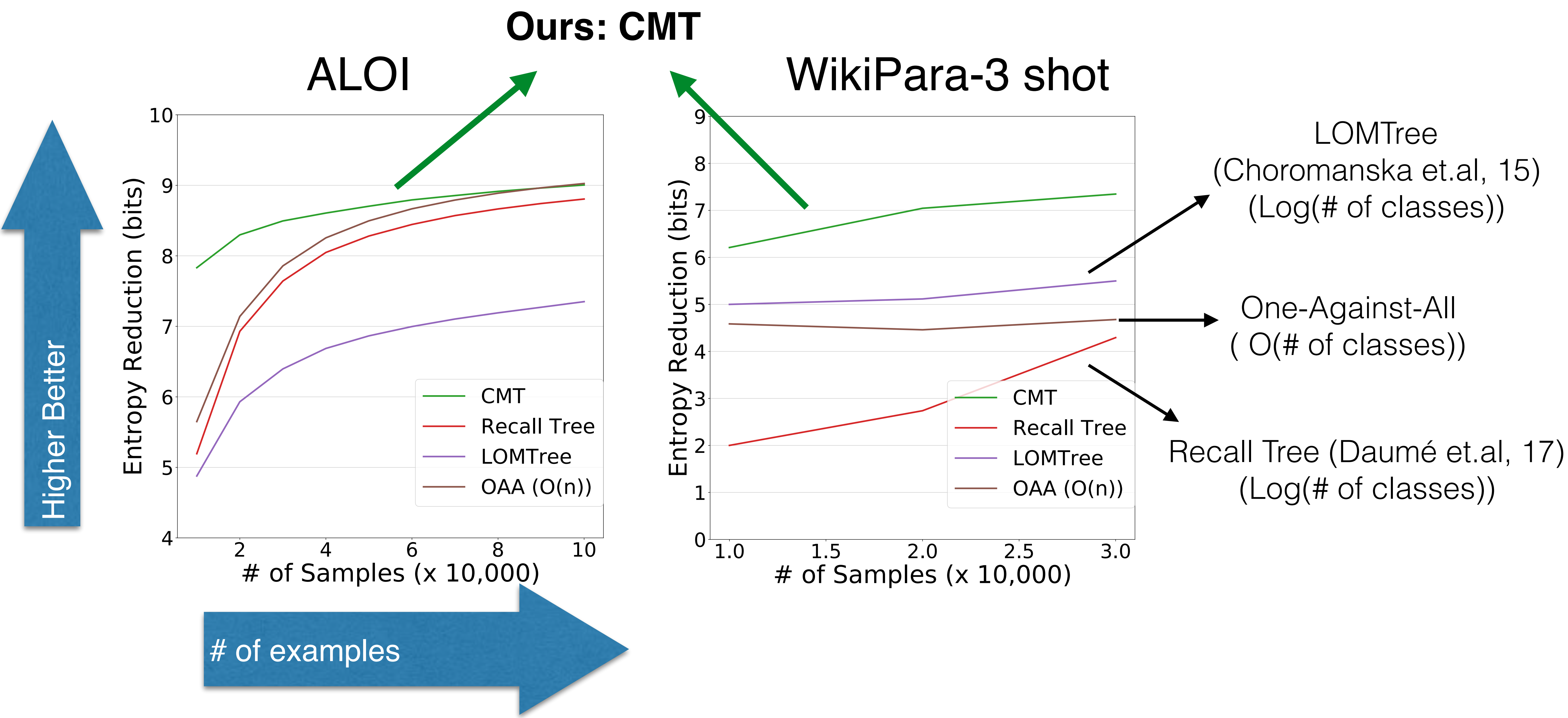
Extreme Multi-class Classification:
Online Progressive Performance



ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!

Extreme Multi-class Classification:
Online Progressive Performance



ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!

Extreme Multi-Label Classification:

Helping an **External** Inference Algorithm (One-Against-Some)

Extreme Multi-Label Classification:

Helping an **External** Inference Algorithm (One-Against-Some)

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme Multi-Label Classification:

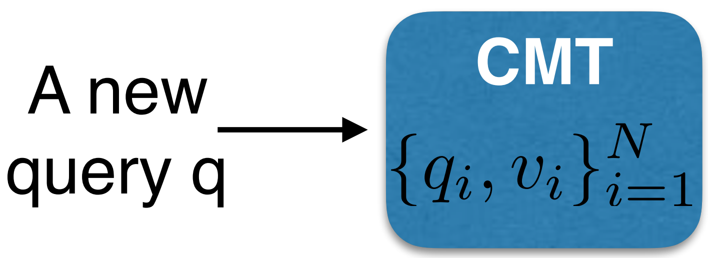
Helping an **External** Inference Algorithm (One-Against-Some)

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Extreme Multi-Label Classification:

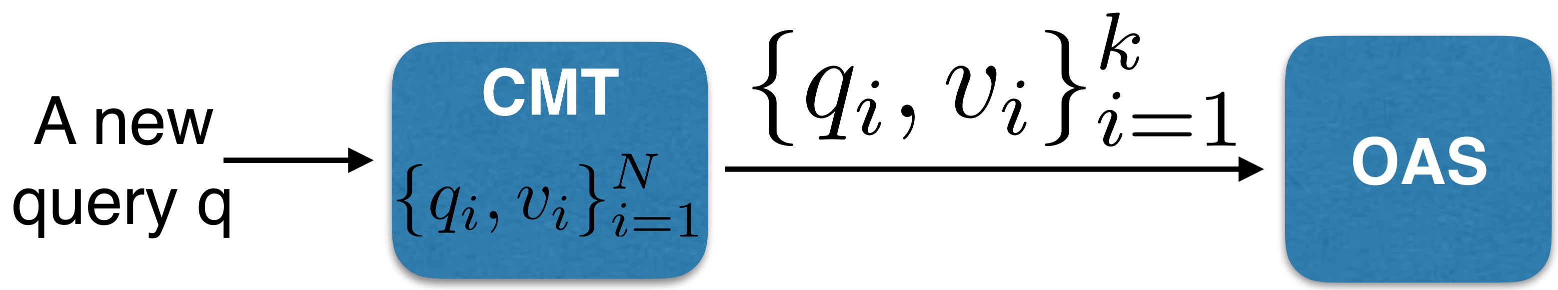
Helping an **External** Inference Algorithm (One-Against-Some)



(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Extreme Multi-Label Classification:
Helping an **External** Inference Algorithm (One-Against-Some)

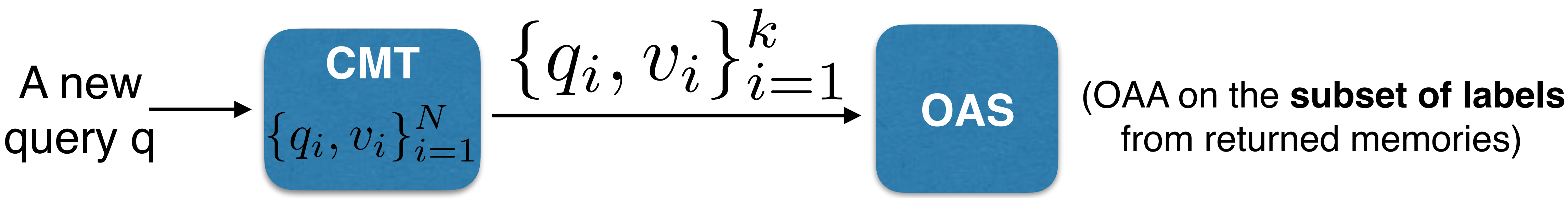


(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Extreme Multi-Label Classification:

Helping an **External** Inference Algorithm (One-Against-Some)

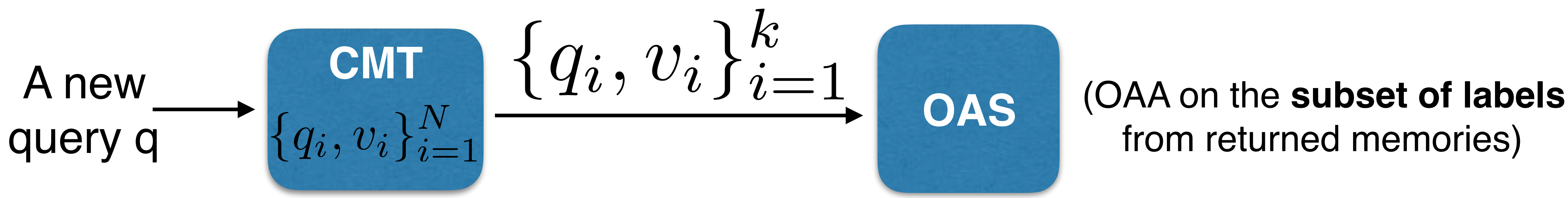


(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Extreme Multi-Label Classification:

Helping an **External** Inference Algorithm (One-Against-Some)



Approach	RCV1-1K loss	RCV1-1K Test time	RCV1-1K Train time	AmazonCat-13K loss	AmazonCat-13K Test time	AmazonCat-13K Train time	Wiki10-31K loss	Wiki10-31K Test time	Wiki10-31K Train time
CMT	2.5	1.4ms	1.9hr	3.2	1.7ms	5.3hr	18.3	25.3ms	1.3hr
OAA	2.6	0.5ms	1.3hr	3.0	8.7ms	15.5hr	20.3	327.1ms	7.2hr

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Extreme Multi-Label Classification:

Helping an **External** Inference Algorithm (One-Against-Some)

A new query q → **CMT** $\{q_i, v_i\}_{i=1}^{N}$ $\xrightarrow{\{q_i, v_i\}_{i=1}^{k}}$ **OAS**

(OAA on the **subset of labels** from returned memories)

Computation: faster

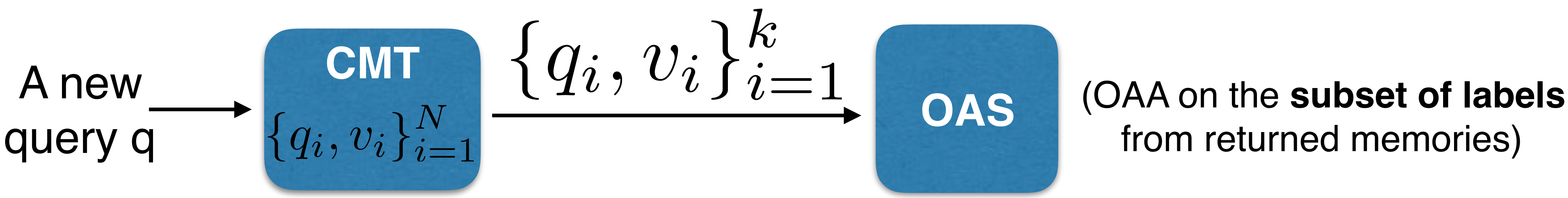
Approach	RCV1-1K loss	RCV1-1K Test time	RCV1-1K Train time	AmazonCat-13K loss	AmazonCat-13K Test time	AmazonCat-13K Train time	Wiki10-31K loss	Wiki10-31K Test time	Wiki10-31K Train time
CMT	2.5	1.4ms	1.9hr	3.2	1.7ms	5.3hr	18.3	25.3ms	1.3hr
OAA	2.6	0.5ms	1.3hr	3.0	8.7ms	15.5hr	20.3	327.1ms	7.2hr

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Extreme Multi-Label Classification:

Helping an **External** Inference Algorithm (One-Against-Some)



Computation: faster

Approach	RCV1-1K loss	RCV1-1K Test time	RCV1-1K Train time	AmazonCat-13K loss	AmazonCat-13K Test time	AmazonCat-13K Train time	Wiki10-31K loss	Wiki10-31K Test time	Wiki10-31K Train time
CMT	2.5	1.4ms	1.9hr	3.2	1.7ms	5.3hr	18.3	25.3ms	1.3hr
OAA	2.6	0.5ms	1.3hr	3.0	8.7ms	15.5hr	20.3	327.1ms	7.2hr

Statistical Performance: similar, sometimes even better

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)

Image Retrieval:
Comparison to Nearest Neighbor Approach

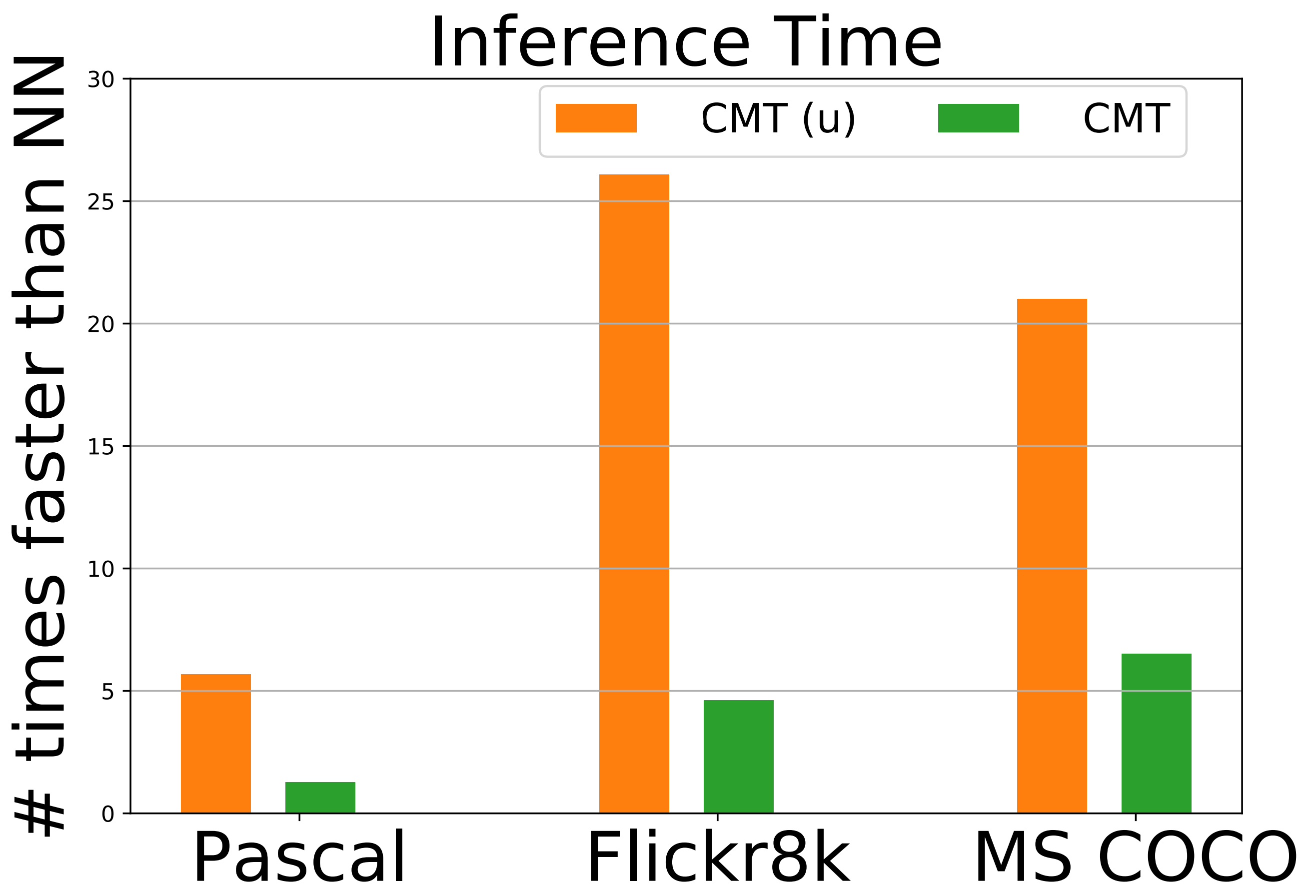
CMT(**u**):
Unsupervised version
i.e., NN (w/ Euclidean dis)
at leaf level

(1) Pascal (1K examples), (2) Flickr8k (8k examples), (2) MSCOCO (80K examples)

Image feature: HoG for Pascal & Flickr, Pre-trained VGG-19 for MSCOCO

Captions feature: Token Occurrences with hashing (high-dim, very sparse)

Image Retrieval:
Comparison to Nearest Neighbor Approach



# of times faster than NN

CMT(**u**):
Unsupervised version
i.e., NN (w/ Euclidean dis)
at leaf level

(1) Pascal (1K examples), (2) Flickr8k (8k examples), (2) MSCOCO (80K examples)

Image feature: HoG for Pascal & Flickr, Pre-trained VGG-19 for MSCOCO

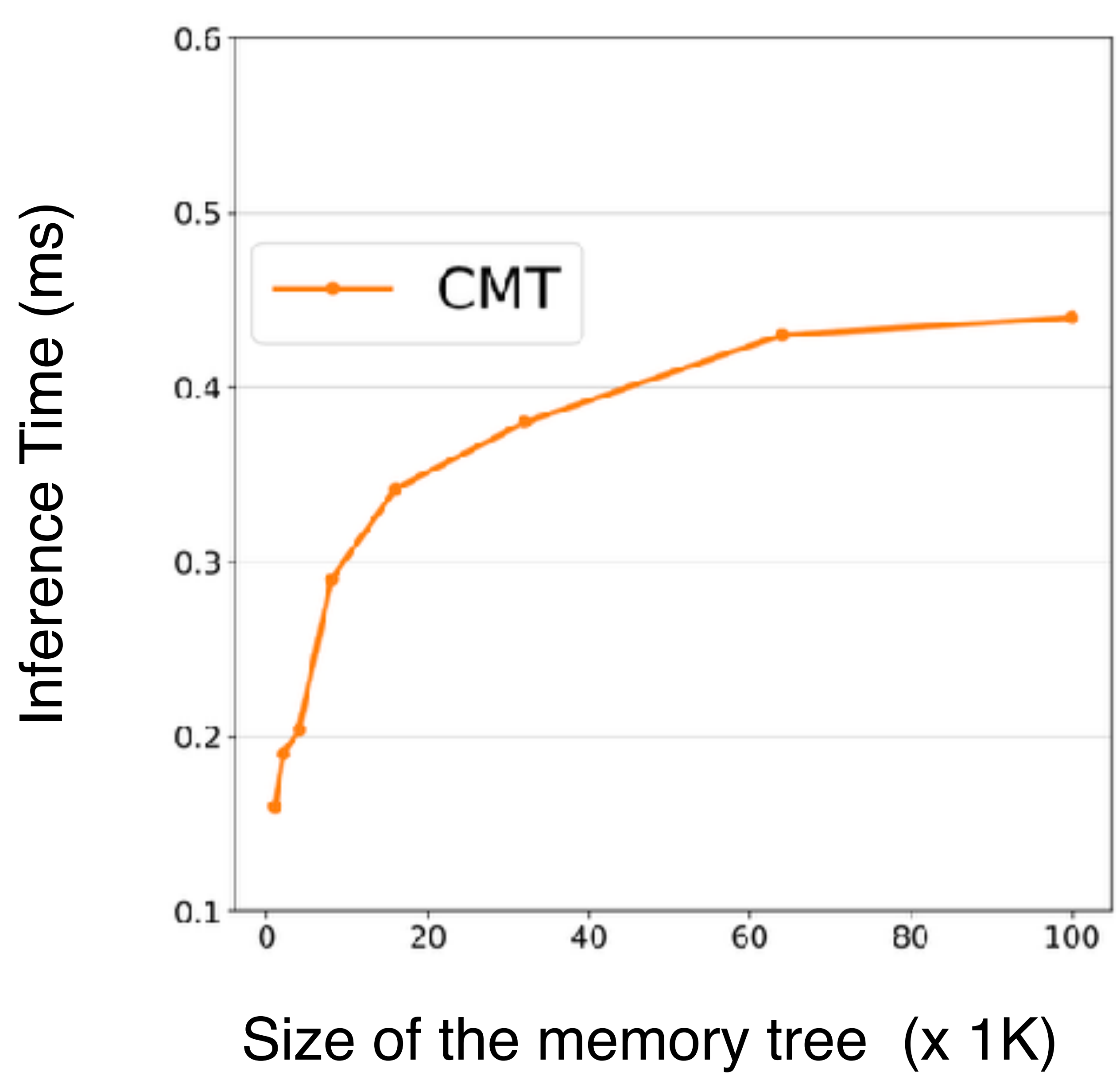
Captions feature: Token Occurrences with hashing (high-dim, very sparse)

How does computation scale wrt the size of memory tree?

On ALOI, we range in dataset size from 1K to 100K

How does computation scale wrt the size of memory tree?

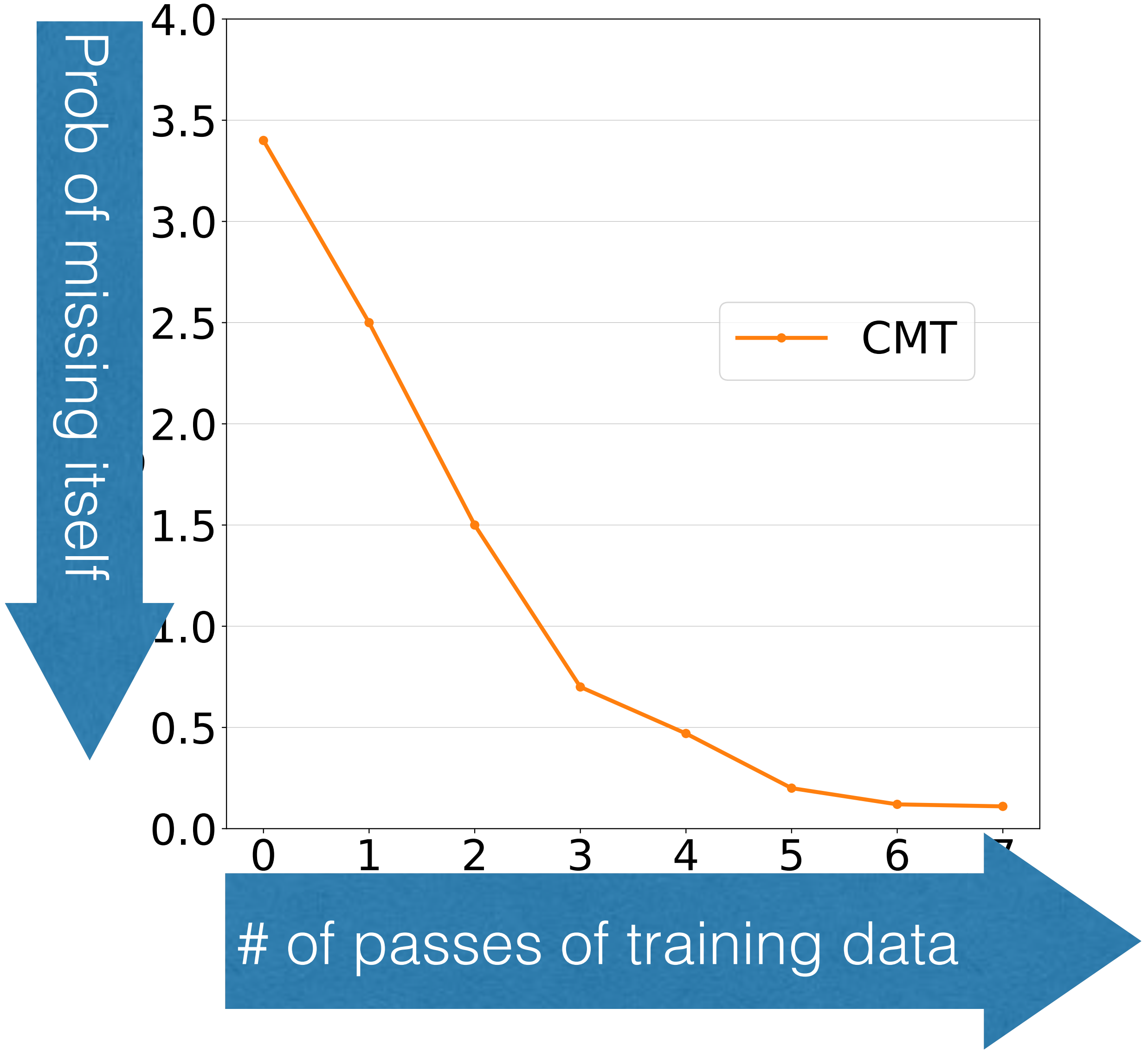
On ALOI, we range in dataset size from 1K to 100K



Increase logarithmically

How does Self-Consistency improve?

1-shot dataset: 10K classes, 10K examples



Self-Consistency improves over time

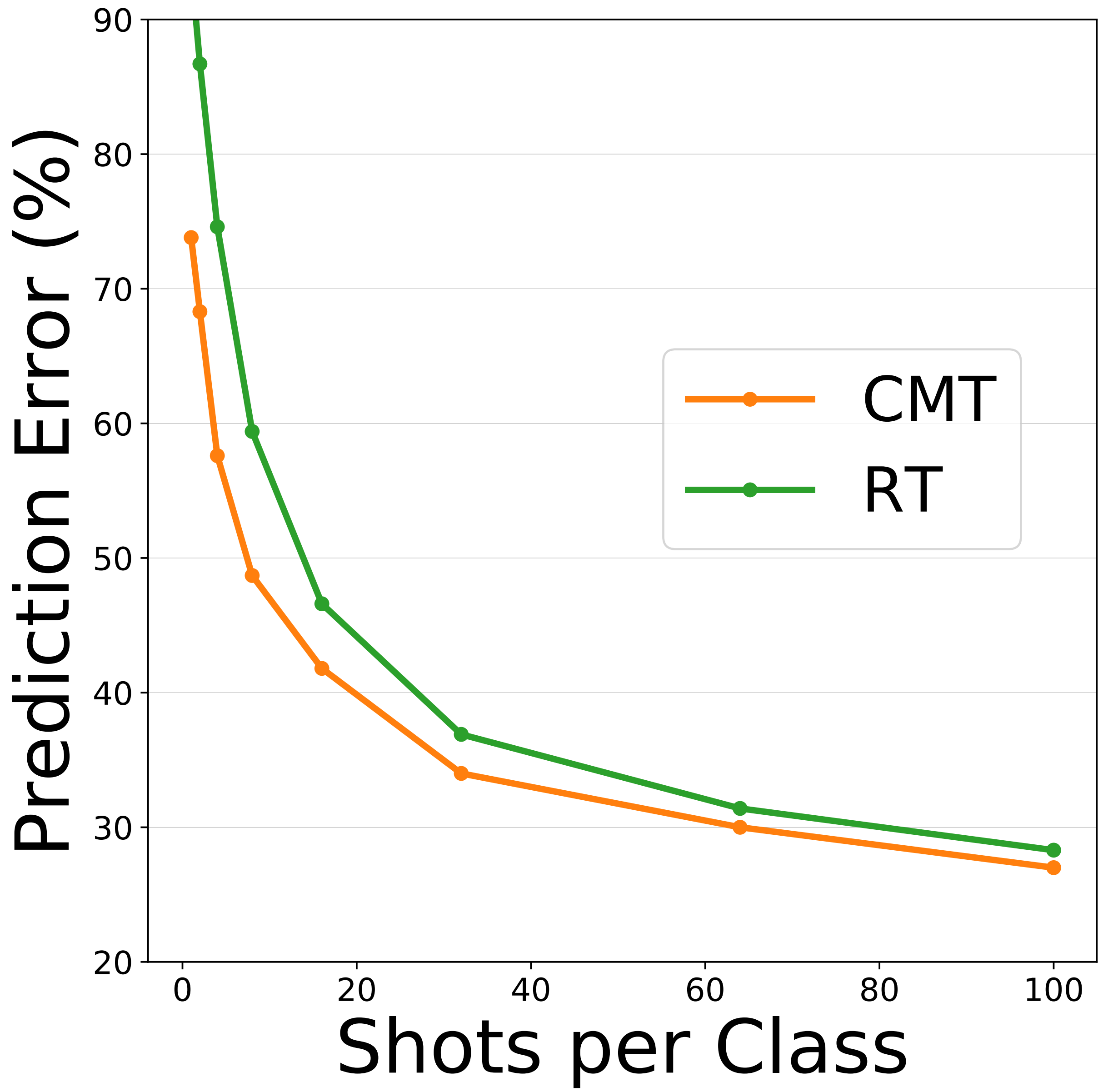
Zero training err
means
perfect self-consistency

How does performance scale wrt classification difficulty?

ALOI S-shot, with S from 1 to 100
(s-shot means s examples per class)

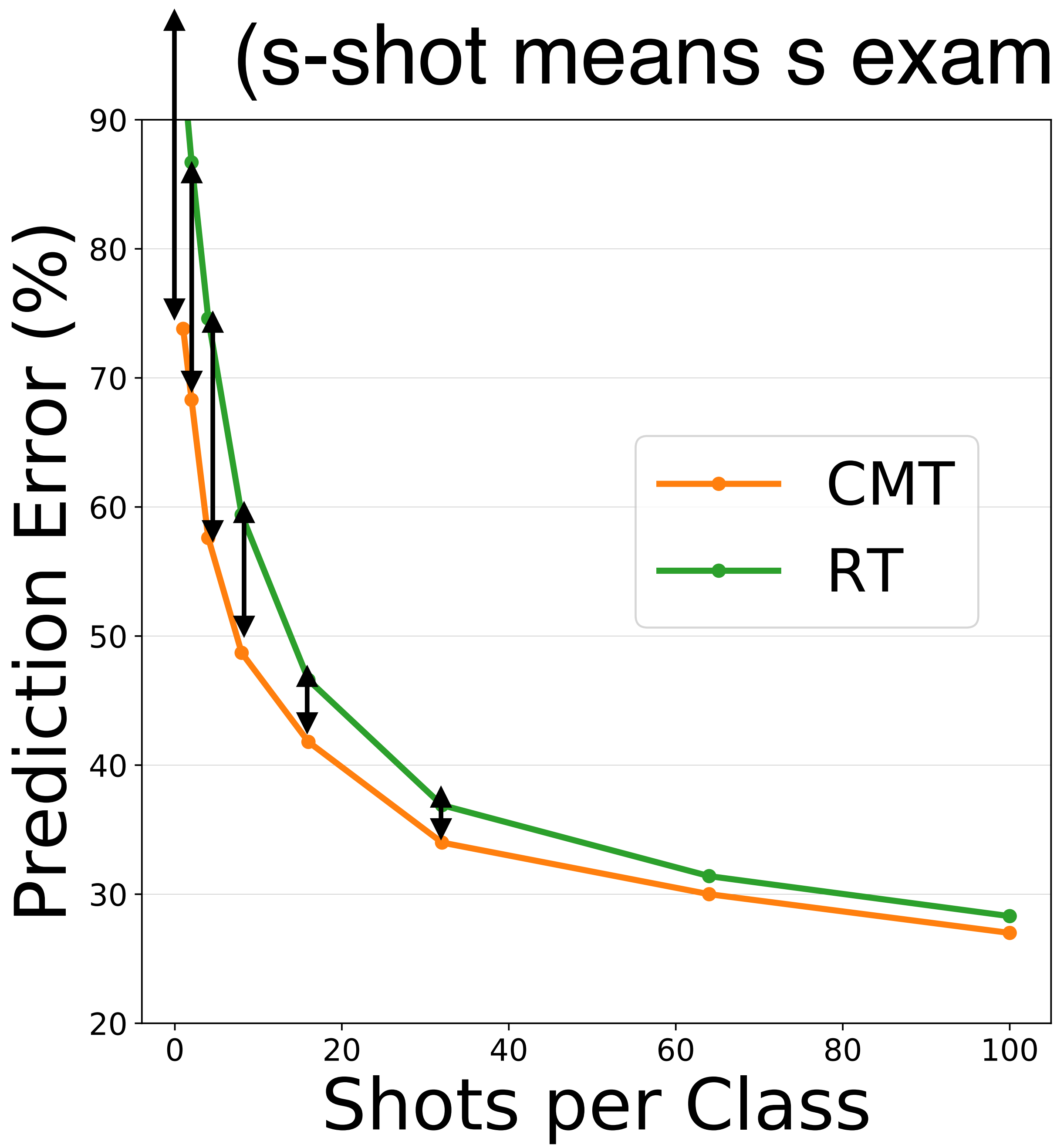
How does performance scale wrt classification difficulty?

ALOI S-shot, with S from 1 to 100
(s-shot means s examples per class)



How does performance scale wrt classification difficulty?

ALOI S-shot, with S from 1 to 100
(s-shot means s examples per class)



CMT (orange) significantly outperforms Recall Tree (Green) when s is small,

Revisit our Desired Properties

Revisit our Desired Properties

Online: Reduction to Online Classification

Revisit our Desired Properties

Online: Reduction to Online Classification

Linear Space: Store examples in leaves

$O(N/\log(N))$ many nodes

Revisit our Desired Properties

Online: Reduction to Online Classification

Linear Space: Store examples in leaves

$O(N/\log(N))$ many nodes

Logarithmic time: Regularization ensures a (provable) near-balanced tree

Revisit our Desired Properties

Online: Reduction to Online Classification

Linear Space: Store examples in leaves

$O(N/\log(N))$ many nodes

Logarithmic time: Regularization ensures a (provable) near-balanced tree

Learning-based: Reinforcement

Revisit our Desired Properties

Online: Reduction to Online Classification

Linear Space: Store examples in leaves

$O(N/\log(N))$ many nodes

Logarithmic time: Regularization ensures a (provable) near-balanced tree

Learning-based: Reinforcement

Self-consistency: an asymptotic guarantee (due to replay)

Thanks!

CMT is in the latest Vowpal Wabbit (VW)

https://github.com/VowpalWabbit/vowpal_wabbit

Try out CMT demos!