An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems

João Nogueira

Tecnologias de Middleware

DI - FCUL - Dez 2006

Agenda

- Motivation
- Key Issues
- The Matching Algorithm
- The Link Matching Algorithm
- Implementation and Performance

Motivation

- Earliest publish-subscribe systems were **subject-based**:
  - Each unit of information (an event) is classified as belonging to one of a fixed set of subjects (groups, channels or topics)

- An emerging alternative is **content-based** subscription:
  - Subscribers have the added flexibility of of choosing filtering criteria along multiple dimensions, they are not limited to a set of subjects and the pre-definition of that set is not required
  - This reduces the overhead of defining and maintaining a large number of groups, thereby making the system easier to manage
  - It is more general than the subject-based approach and can be used to implement it
  - Implementations of such systems have previously not been developed

Key Issues

- In order to implement a content-based publish-subscribe system, two key problems must be solved:
  - The problem of efficiently **matching** en event against a large number of subscribers on a single event broker
  - The problem of efficiently **multicasting** events within a network of event brokers. This problem becomes crucial in two settings:
    - When the pub/sub system is geographically distributed and event brokers are connected via a relatively low-speed WAN
    - When the pub/sub system has the scale to support a large number of publishers, subscribers and events.
    - In both cases, it becomes crucial to limit the distribution of a published event to only those brokers that have subscribers interested in that event

Key Issues (2)

- There are two straightforward approaches to solving the multicasting problem for content-based systems:
  - The **match-first** approach, where the event is first matched against all subscriptions, thus generating a destination list and the event is then routed to all entries on this list
  - The **flooding** approach, where the event is broadcast, or flooded, to all destinations using standard multicast and unwanted events are then filtered out at these destinations
- Both approaches may work well in small systems but can be inefficient in large ones:
  - The contribution of this work is a new distributed algorithm - link matching - introducing an efficient solution to the multicast problem.
  - The intuition is that each broker should perform just enough of the matching work to determine which neighbouring brokers should receive the event

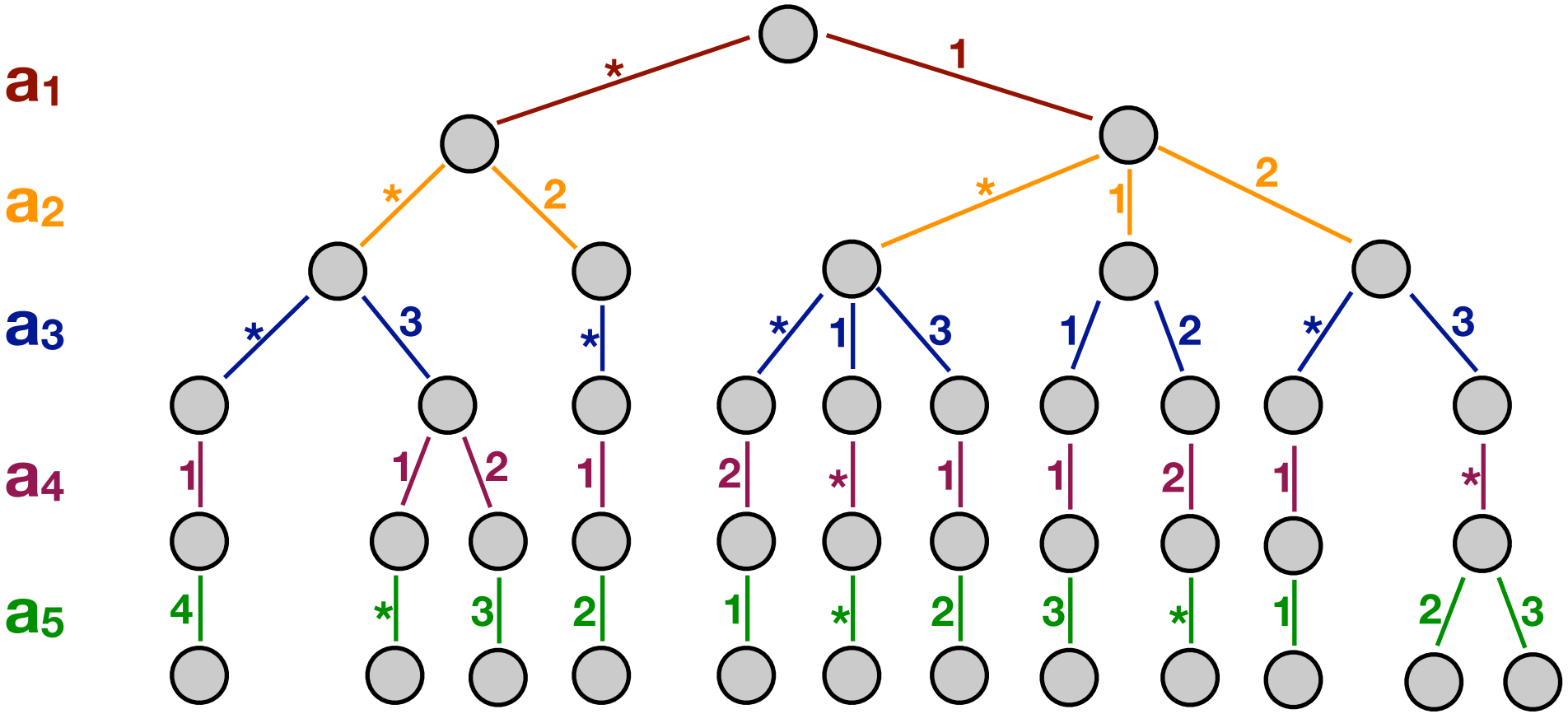
The Matching Algorithm

- Non-distributed algorithm for matching events to subscriptions
- Matching based on sorting and organising the subscriptions into a parallel search tree (PST)
  - Each subscription corresponds to a path from the root to a leaf
- Assumptions:
  - Addition and deletion of subscriptions are rare occurrences relative to the rate of published events
  - Changes to the subscription set are batched and periodically propagated to all brokers
  - The described algorithm is the “steady state” matching algorithm to be executed between changes to the set of subscriptions

The Matching Algorithm
How it works

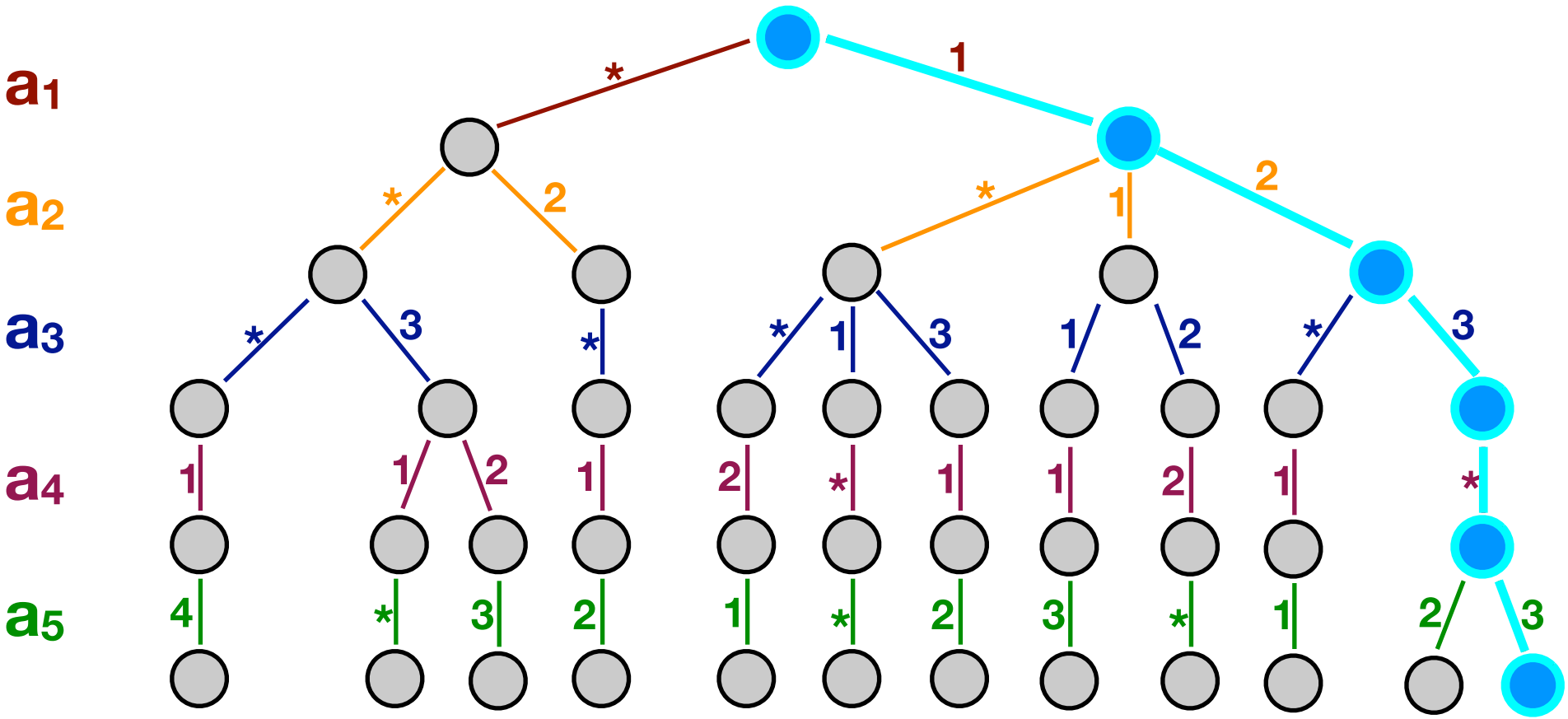
- Given a parallel search tree (PST), the matching algorithm proceeds as follows:
  - It starts at the root of the PST with attribute a_1
  - At any non-leaf node of the tree, we find value v_j of the current attribute a_j
  - We then transverse any of the following edges that apply:
    - The edge labelled v_j if there's one, *and*
    - The edge labelled * if there's one
  - This may lead to either 0, 1 or 2 successor nodes (or more if the tests are not strict equalities)
  - We then initiate parallel sub-searches at each successor node
  - When one search reaches a leaf, all the subscriptions in that leaf are added to the list of matching subscriptions

Example PST



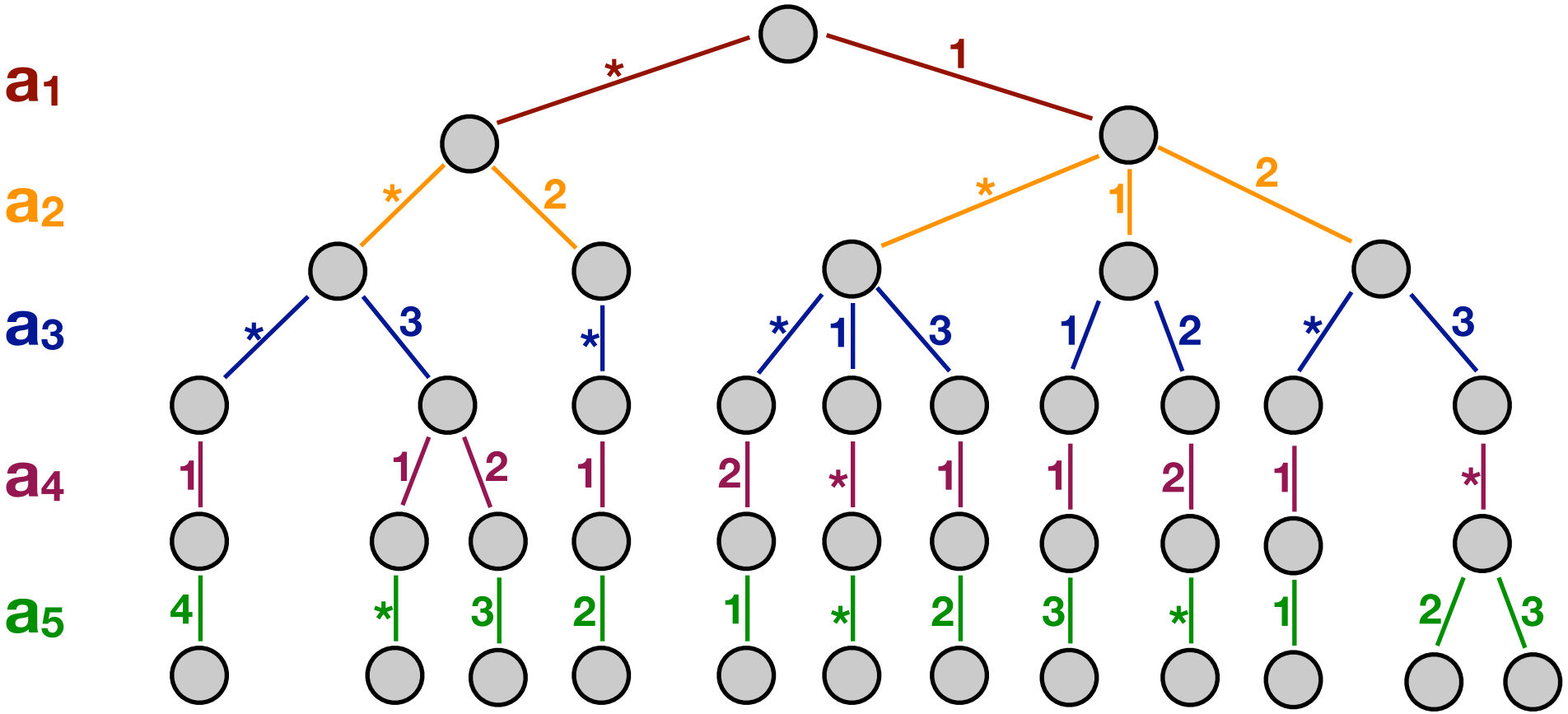
The Matching Algorithm

Example PST



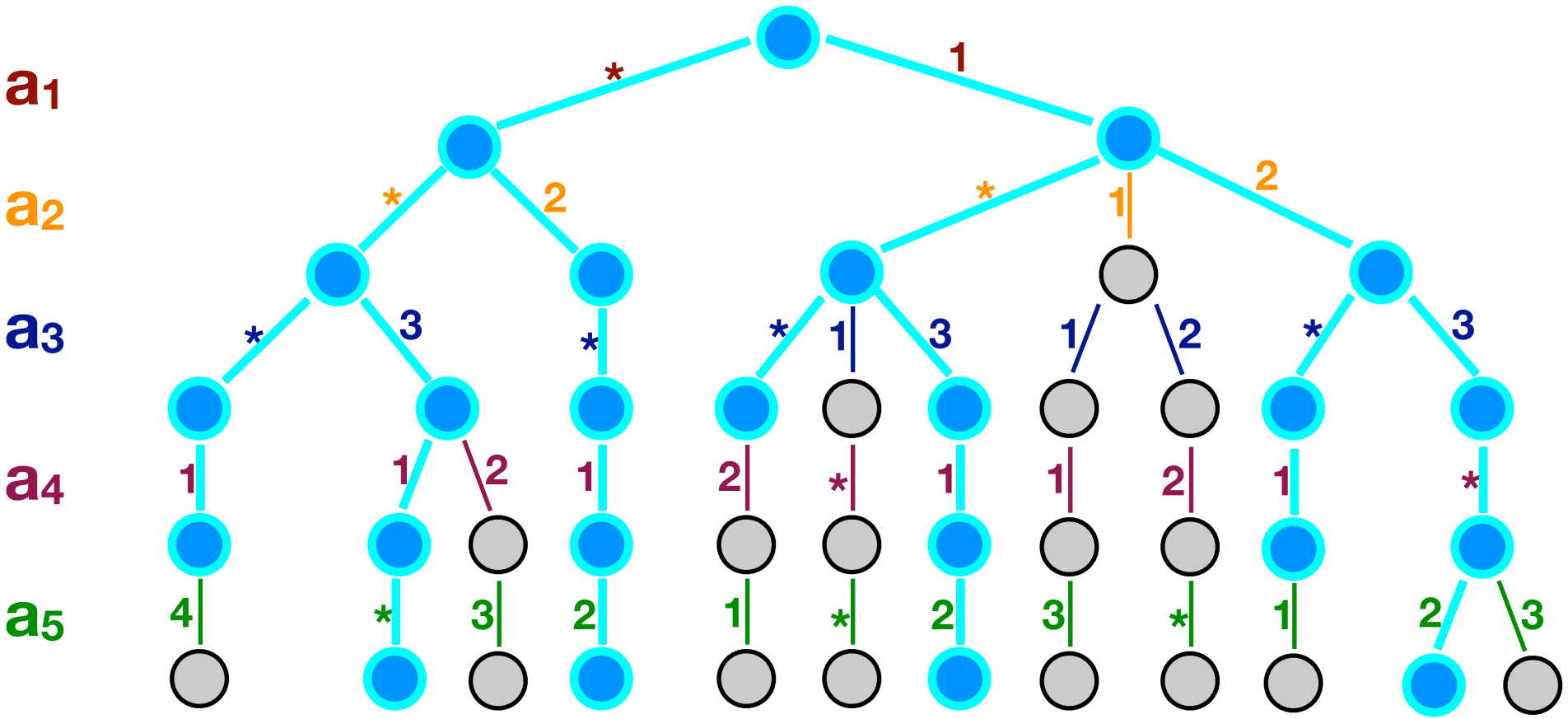
$(a_1=1 \;\&\&\; a_2=2 \;\&\&\; a_3=3 \;\&\&\; a_5=3)$

Example PST



The Matching Algorithm

Example PST



The Matching Algorithm
Considerations

- Other types of tests (besides equality) are also possible

- The way in which attributes are ordered from root to leaf in the PST can be arbitrary
  - The implemented system performs better if the attributes near the root are chosen to have the fewest number of subscriptions labelled with a *

- The cost of the matching algorithm increases *less* than linearly with the number of subscriptions

The Matching Algorithm
Optimisations

- **Factoring**: Some search steps can be avoided, at the cost of increased space, by factoring out certain attributes:
  - Some attributes (preferably those for which the subscriptions rarely contain “don’t care” tests) are selected as indices
  - A separate sub-tree is built for each possible value (or ranges, each distinguished value range) of the index attributes
- **Trivial Test Elimination**: Nodes with a single child which is reached by a *-branch may be eliminated
- **Delayed Branching**: Traversing *-branches may be delayed until after a set of predicate tests have been applied
  - This optimisation prunes paths from those *-branches which are inconsistent with the tests

The Link-Matching Algorithm

- Distributed matching algorithm for a network of brokers and publishing and subscribing clients
- After receiving an event, each broker performs just enough matching steps to determine which of its neighbours should receive it
- A broker is connected to its neighbours (brokers or clients) through links
- Therefore, rather than determining which subset of all subscribers is to receive the event, computes the subset of its neighbours that is to receive the event instead
    - i.e. determines those links along which it should transmit the event

The Link-Matching Algorithm

How it works

- Each broker in the network has a copy of all subscriptions organised into a PST data structure

- Each broker performs the following steps:
  - PST annotation (at PST preparation time)
  - Initialisation mask computation (at PST preparation time)
  - Event matching (at run-time)

The Link-Matching Algorithm
PST Annotation

- Each broker annotates each node of its PST with a vector of trits:
  - Each trit is a three-valued indicator with values “yes” (Y), “no” (N) or “maybe” (M)
  - The vector has one trit position per link from the given broker
- The trit’s values have the following meanings:
  - Yes: a search reaching the node is guaranteed to match a subscriber reachable by that link
  - No: a search reaching the node will have no sub-search reaching a subscriber through that link
  - Maybe: there may be some subscriber that matches the search reachable through that link

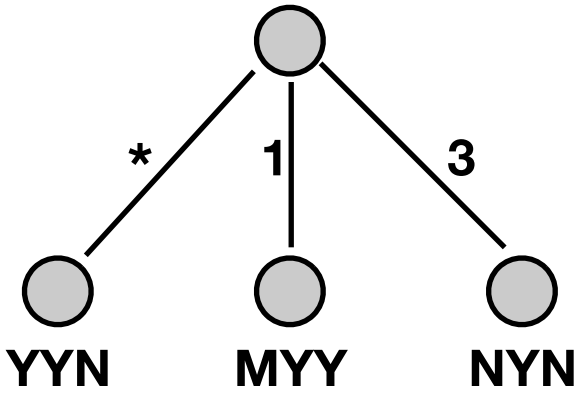
The Link-Matching Algorithm
PST Annotation (2)

- Annotation is a recursive process starting at the leaves of the PST, which represent the subscriptions
  - It starts by annotating leaf nodes: for each leaf, a trit vector is created and filled with Y's for the links on the path from the given broker to the subscribers associated with that leaf and N's for all other positions
    - Leaf nodes correspond to particular predicates and a set of subscribers
  - Annotations are then propagated back toward the root node using two operators:
    - *Alternative Combine*: used to combine the annotations of all non-* nodes
    - *Parallel Combine*: used to merge the results of alternative combine operations with the annotation of a child reached by a *-branch

PST Annotation (3)

Alternative	Yes	Maybe	No
Yes	Y	M	M
Maybe	M	M	M
No	M	M	N

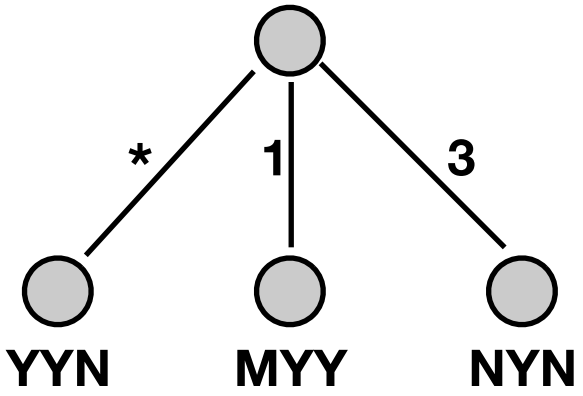
Parallel	Yes	Maybe	No
Yes	Y	Y	Y
Maybe	Y	M	M
No	Y	M	N



PST Annotation (3)

Alternative	Yes	Maybe	No
Yes	Y	M	M
Maybe	M	M	M
No	M	M	N

Parallel	Yes	Maybe	No
Yes	Y	Y	Y
Maybe	Y	M	M
No	Y	M	N

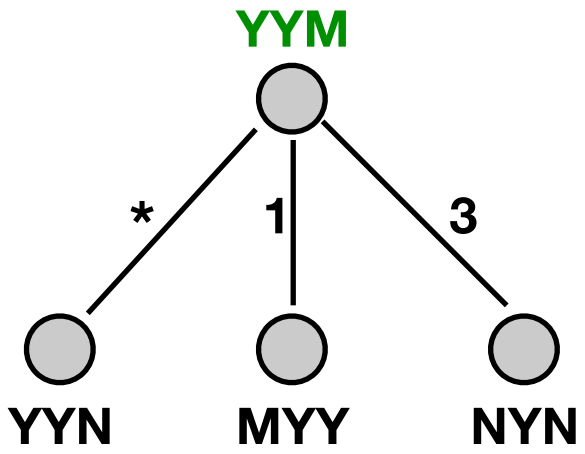


MYY Ⓐ NYN = MYM

PST Annotation (3)

Alternative	Yes	Maybe	No
Yes	Y	M	M
Maybe	M	M	M
No	M	M	N

Parallel	Yes	Maybe	No
Yes	Y	Y	Y
Maybe	Y	M	M
No	Y	M	N

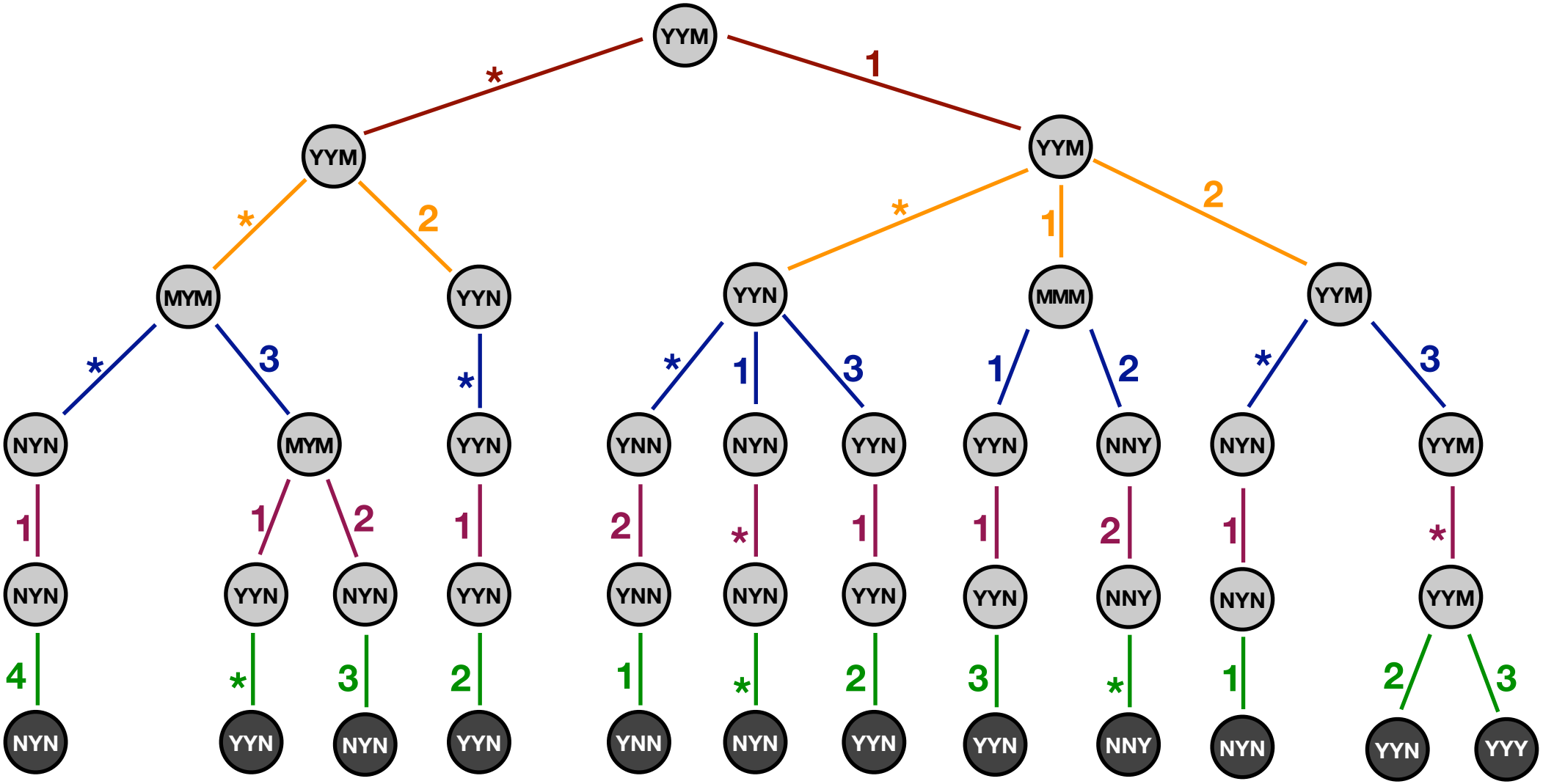


MYY Ⓐ NYN = MYM

MYM Ⓟ YYN = YYM

The Link-Matching Algorithm

PST Annotation (4)



The Link-Matching Algorithm
Initialisation Mask Computation

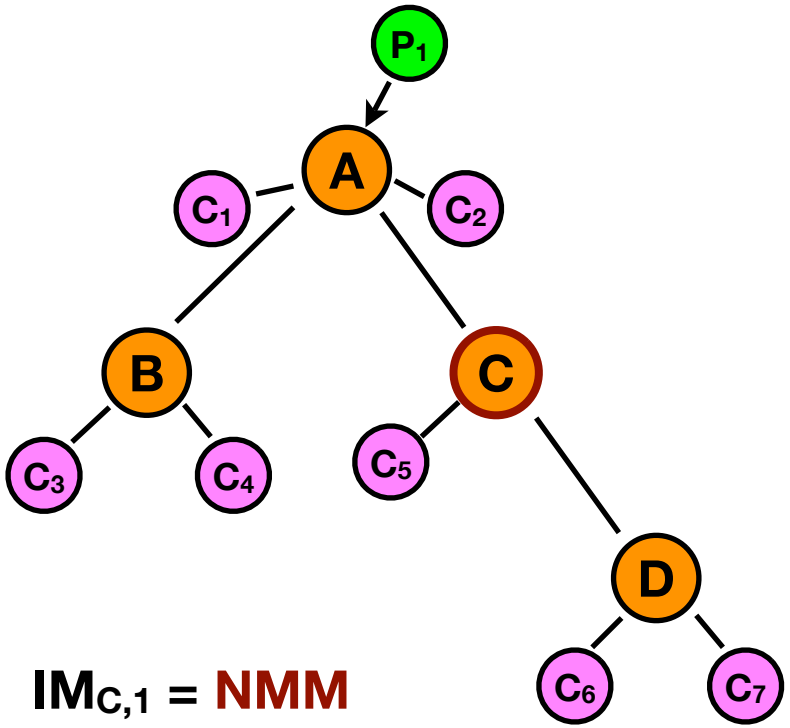
- Assumptions
  - Each broker knows the topology of the broker network as well as the best paths between each broker and each destination (i.e. subscriber)
  - From this topology, each broker constructs a routing table mapping each possible destination to the link which is the next hop along the best path to the destination
  - The broker knows the set of spanning trees, only one of which will ever be used for each publisher
    - At most, there will be one spanning tree for each broker that has publisher neighbours

The Link-Matching Algorithm
Initialisation Mask Computation (2)

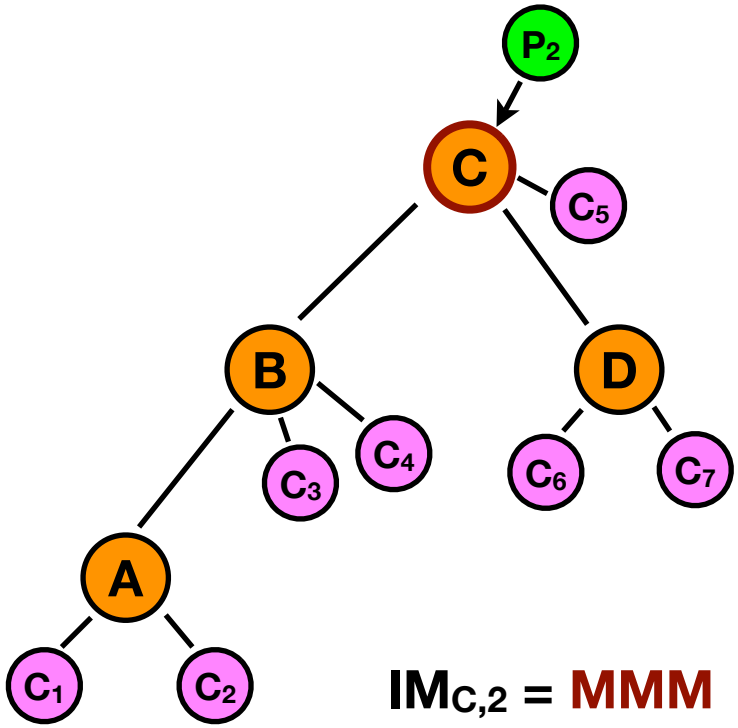
- Using these best paths and spanning trees, each broker computes the *downstream destinations* for each spanning tree
  - A destination is downstream from a broker when it is a descendant of the broker on the spanning tree
- Each broker then associates each spanning tree with an *initialisation mask*: one trit per link
  - The trit for link *l* has value M if at least one of the destinations routable via *l* is a descendant of the broker in the spanning tree, or N otherwise
- The significance of the mask is that an event arriving at a broker should only be propagated along those links leading away from the publisher
  - These will begin with a mask of M

The Link-Matching Algorithm

Initialisation Mask Computation (3)



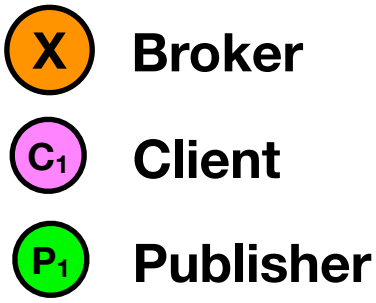
$IM_{C,1}$ = NMM



$IM_{C,2}$ = MMM

$Links_C$ = { L_1, L_2, L_3 }

- L_1 → Dests = { C_5 }
- L_2 → Dests = { D, C_6, C_7 }
- L_3 → Dests = { B, C_3, C_4, A, C_1, C_2 }



The Link-Matching Algorithm
Matching Events

- When an event originating at a publisher is received at a broker, the following steps are taken using the annotated search tree:

1) A mask is created and initialised to the *initialisation mask* associated with the publisher's spanning tree

2) Starting at the root node of the PST, the mask is *refined* using the trit vector annotation at the current node.

   - During refinement, any M in the mask is replaced with the corresponding trit vector annotation
   - If the mask is fully refined (i.e. has no M trits), the search ends, returning that mask

The Link-Matching Algorithm
Matching Events (2)

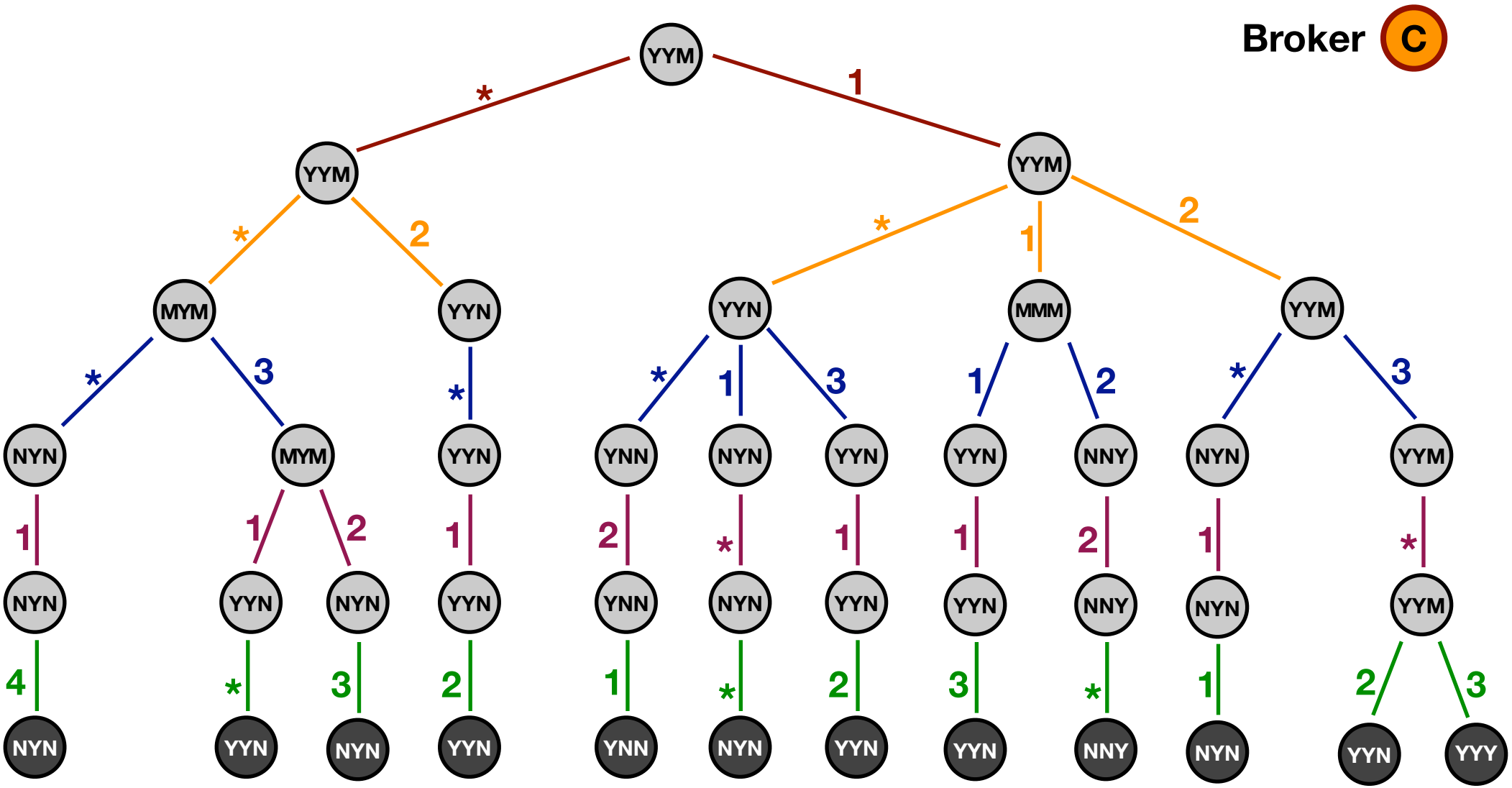
3) The designated test is performed on the PST and 0, 1 or 2 children are found for continuing the search according to the matching algorithm

- A sub-search is executed at each such child using a copy of the current mask
- On the return of each sub-search, all M trits in the current mask for which there's an Y trit in the sub-search mask
- After all the children have been searched, the remaining M trits are made N trits and the resulting mask is returned

4) The top-level search terminates and sends a copy of the event to all links corresponding to Y trits in the returned mask

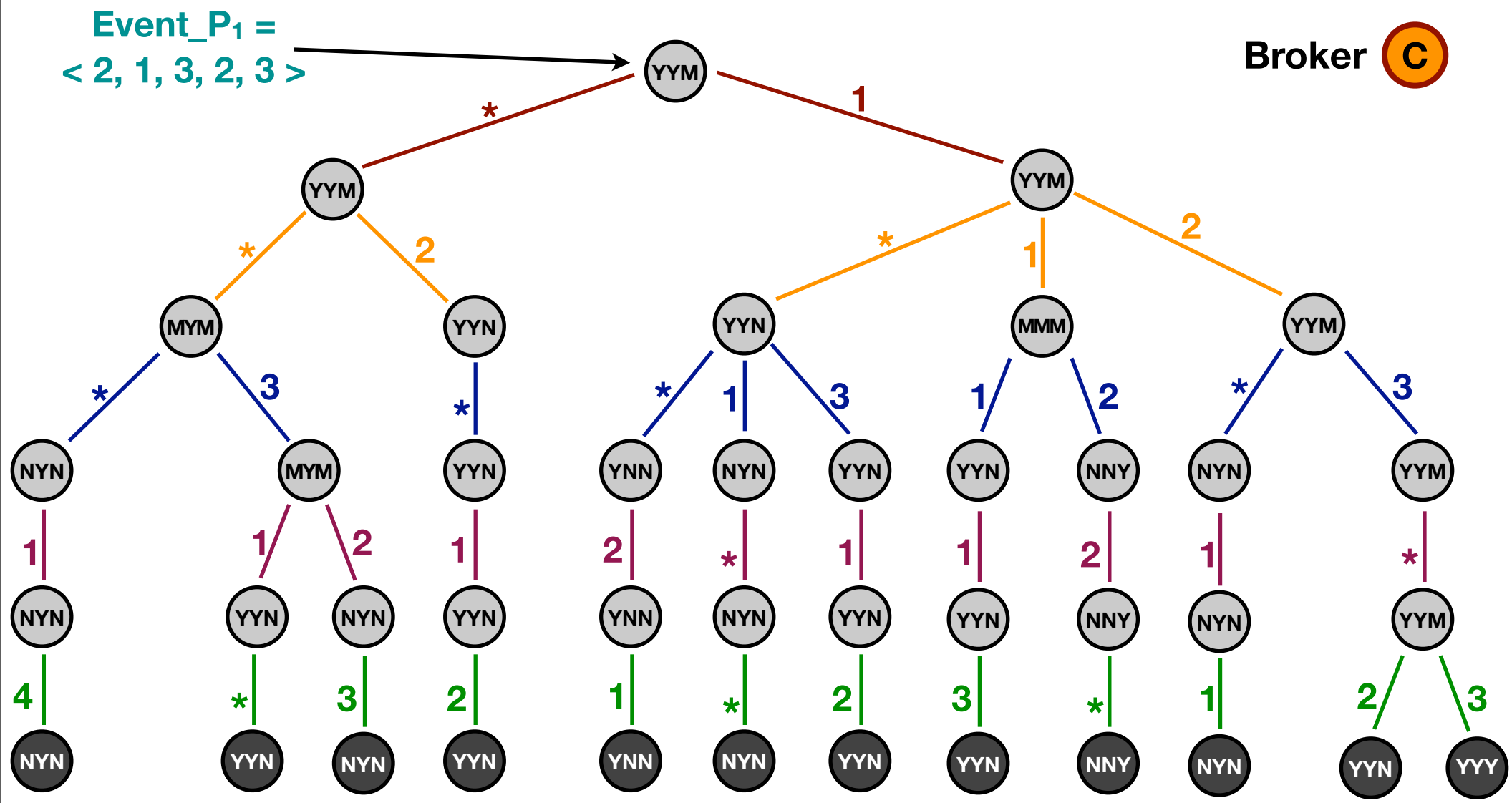
The Link-Matching Algorithm

Matching Events (3)



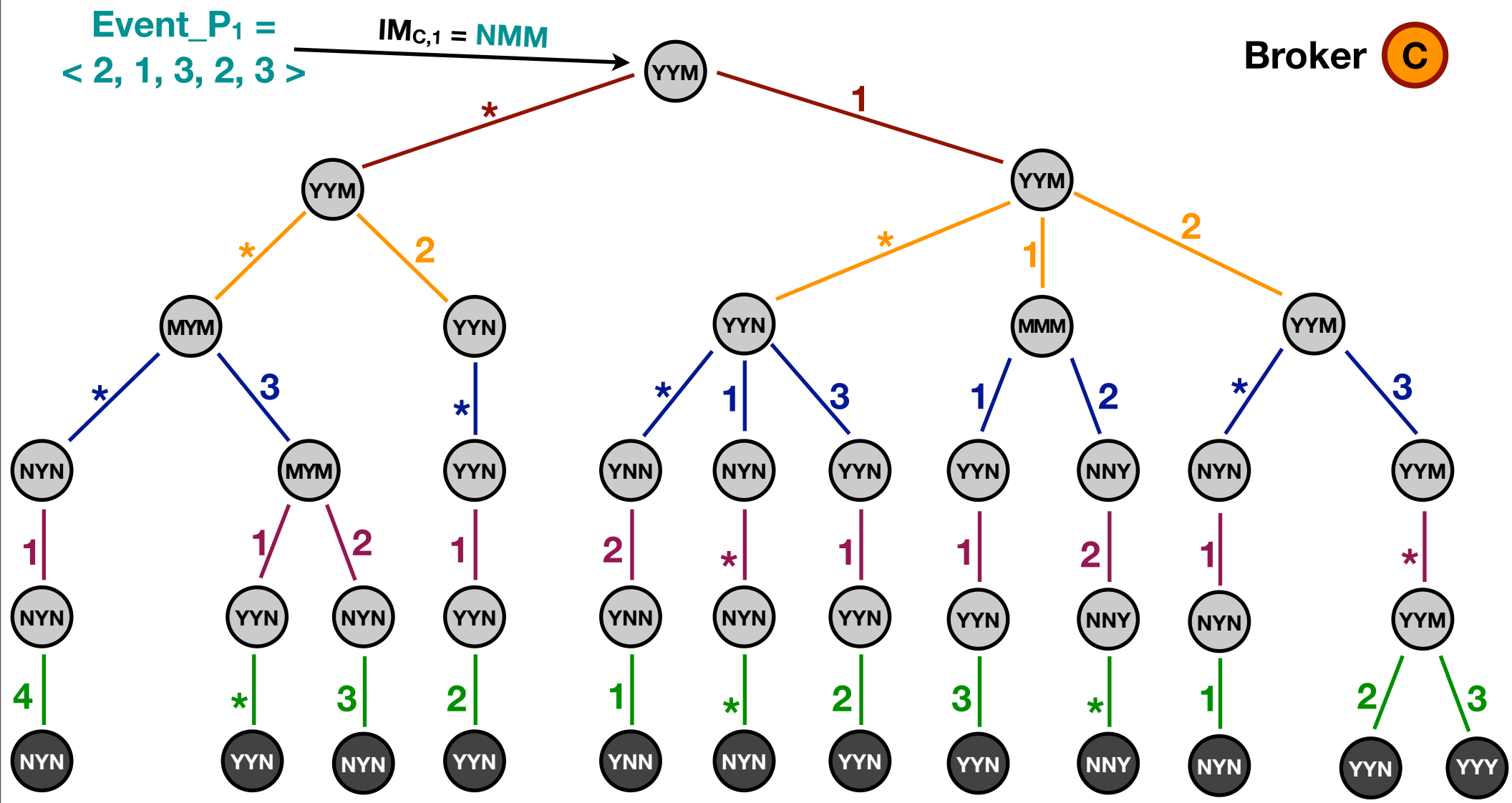
The Link-Matching Algorithm

Matching Events (3)



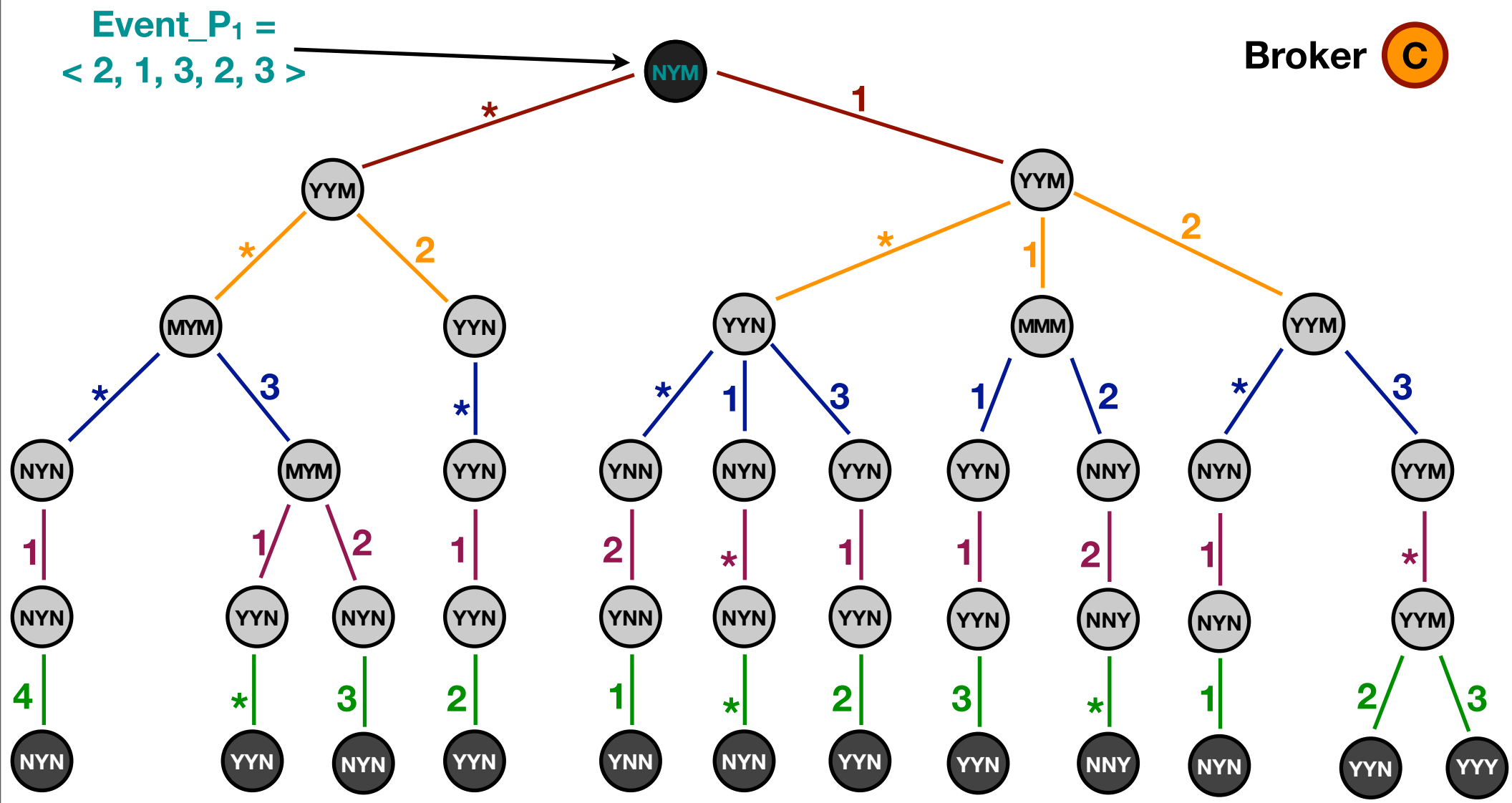
The Link-Matching Algorithm

Matching Events (3)



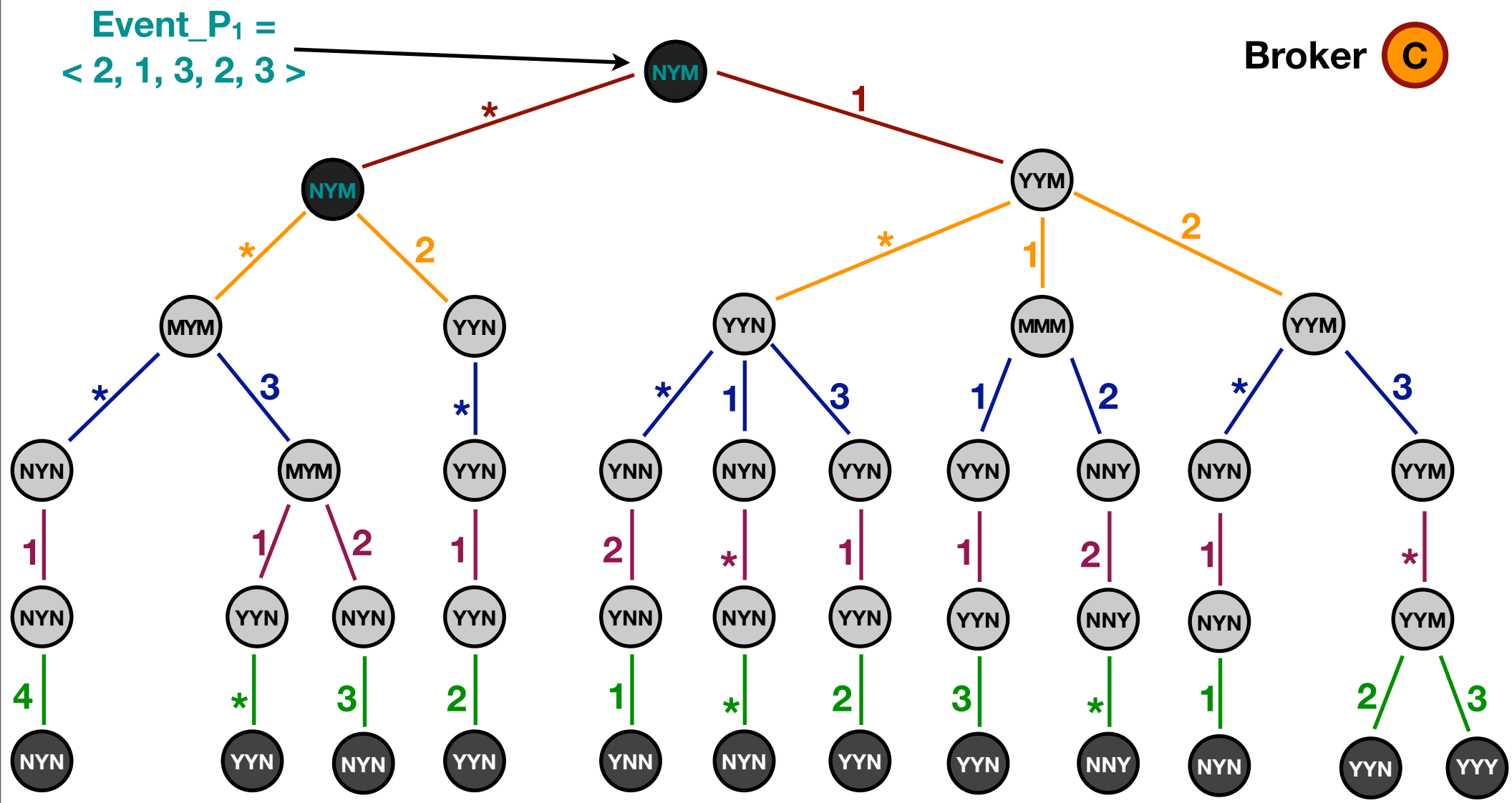
The Link-Matching Algorithm

Matching Events (3)



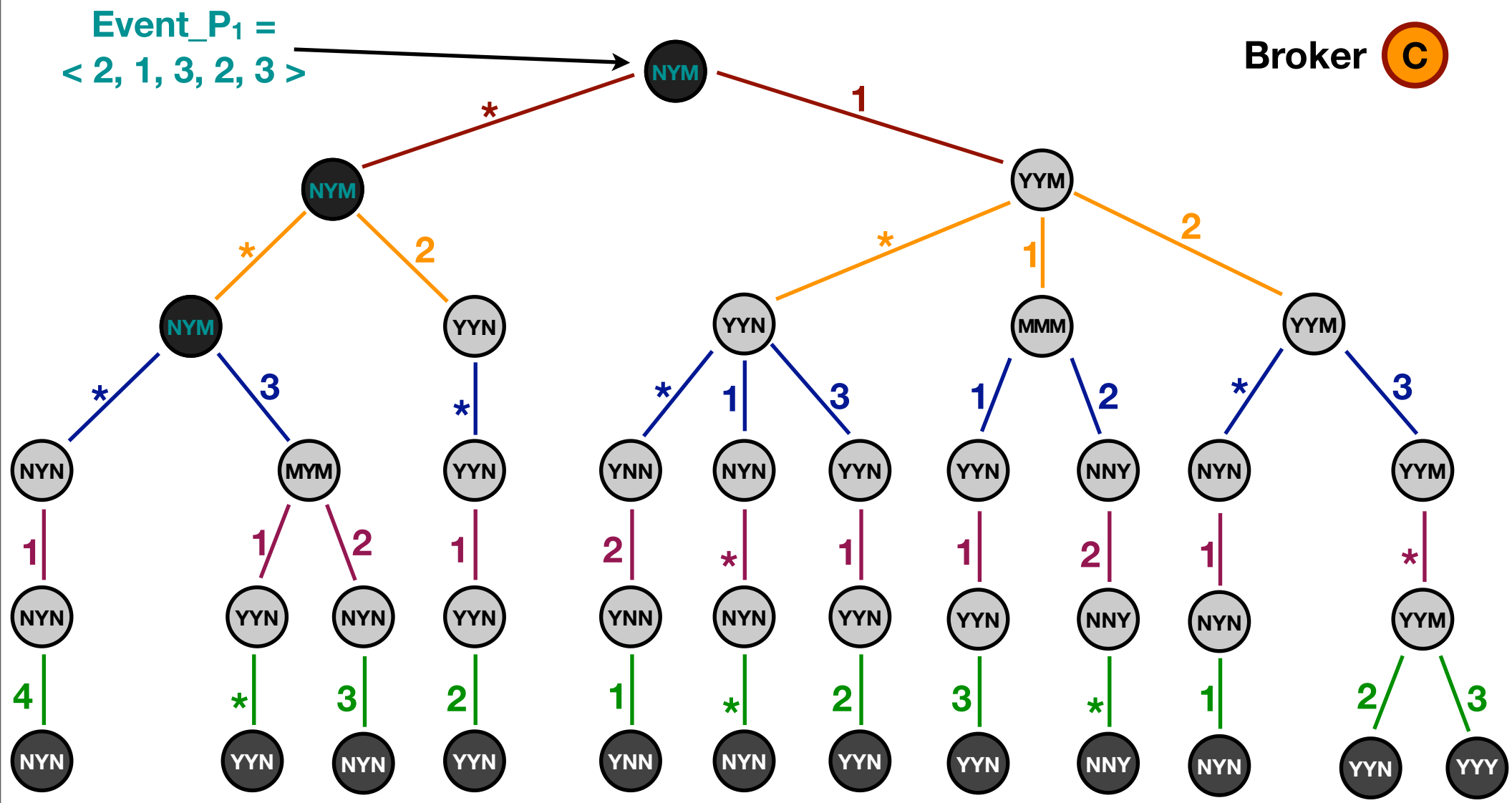
The Link-Matching Algorithm

Matching Events (3)



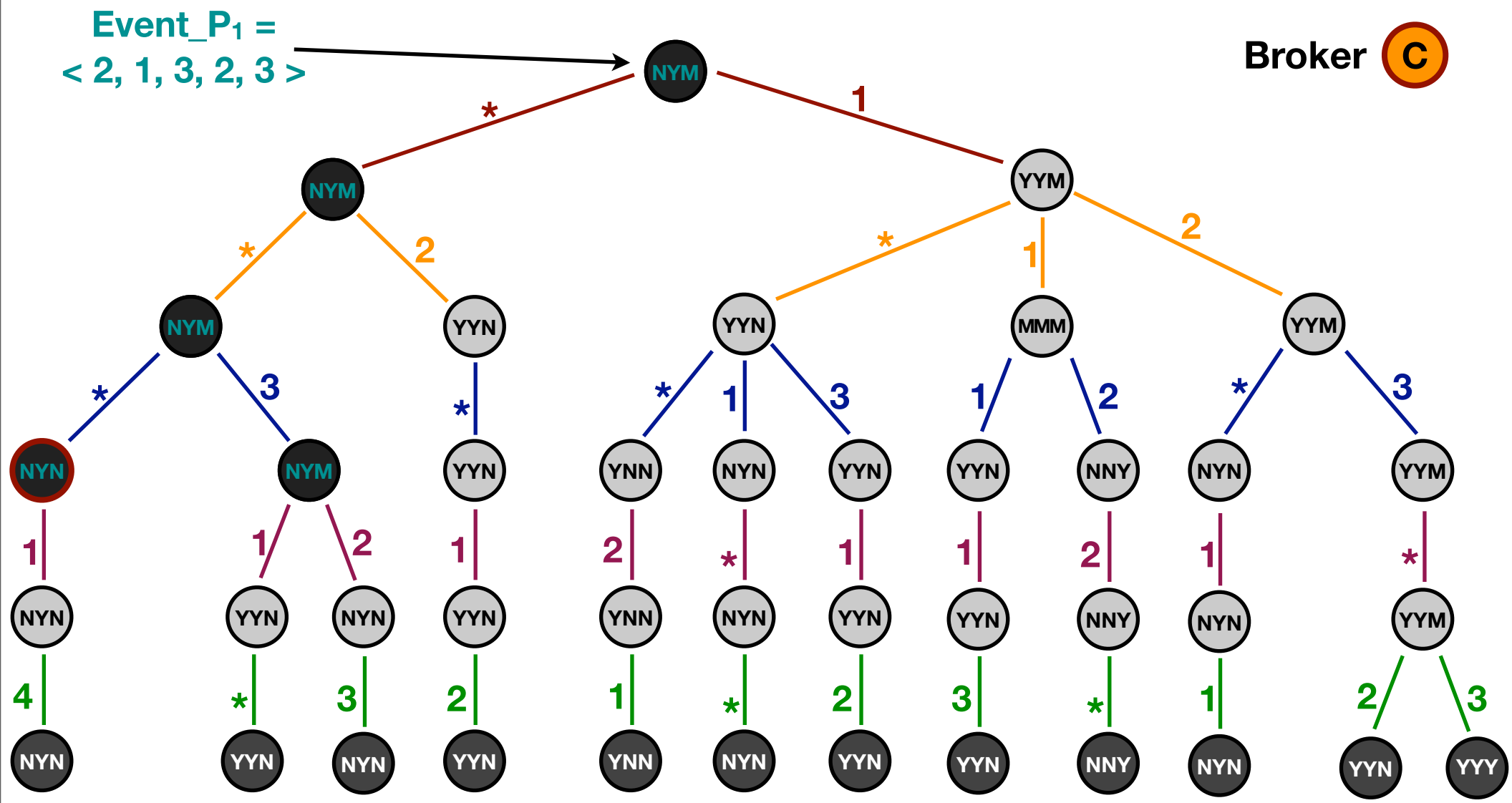
The Link-Matching Algorithm

Matching Events (3)



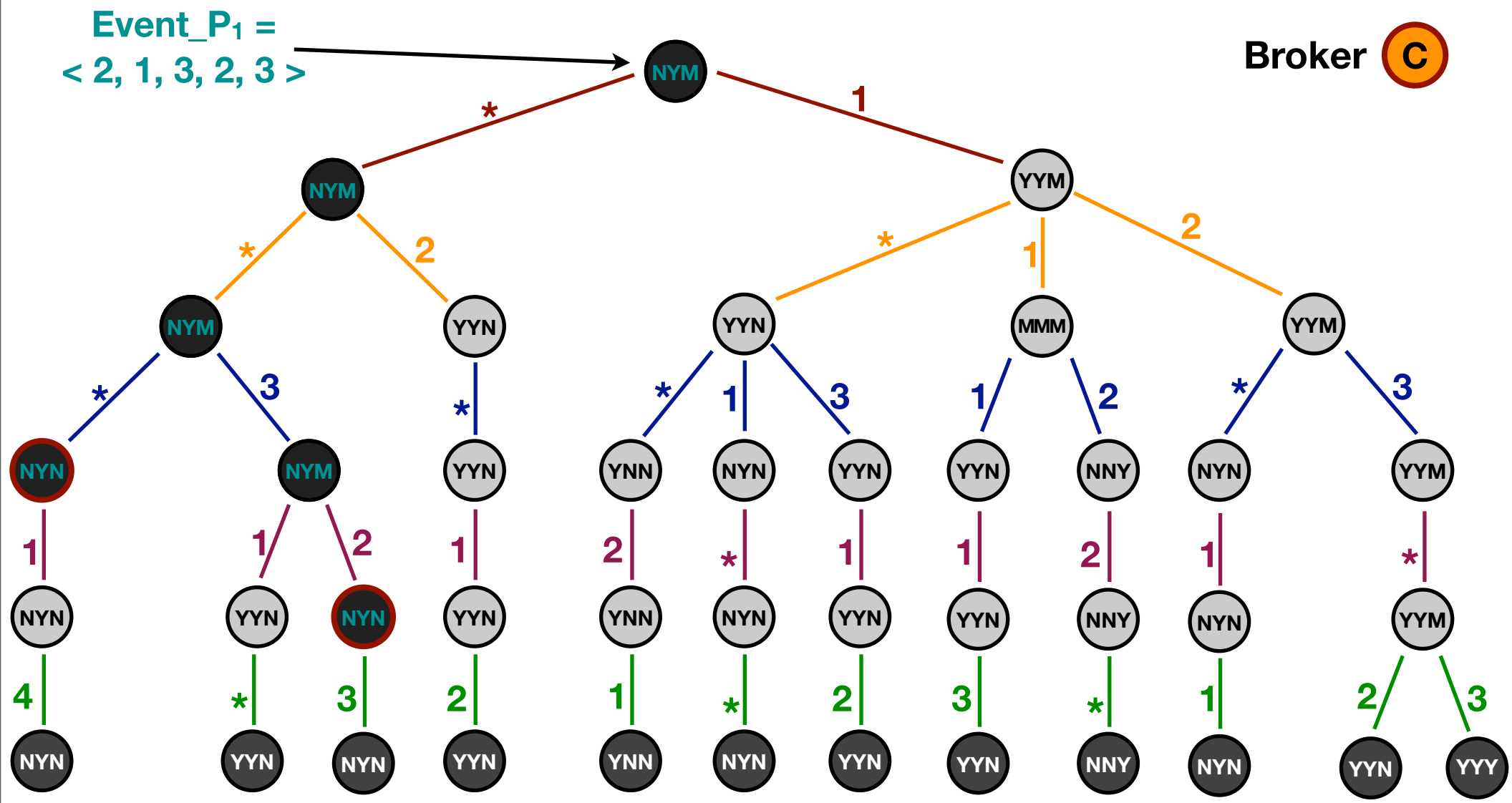
The Link-Matching Algorithm

Matching Events (3)



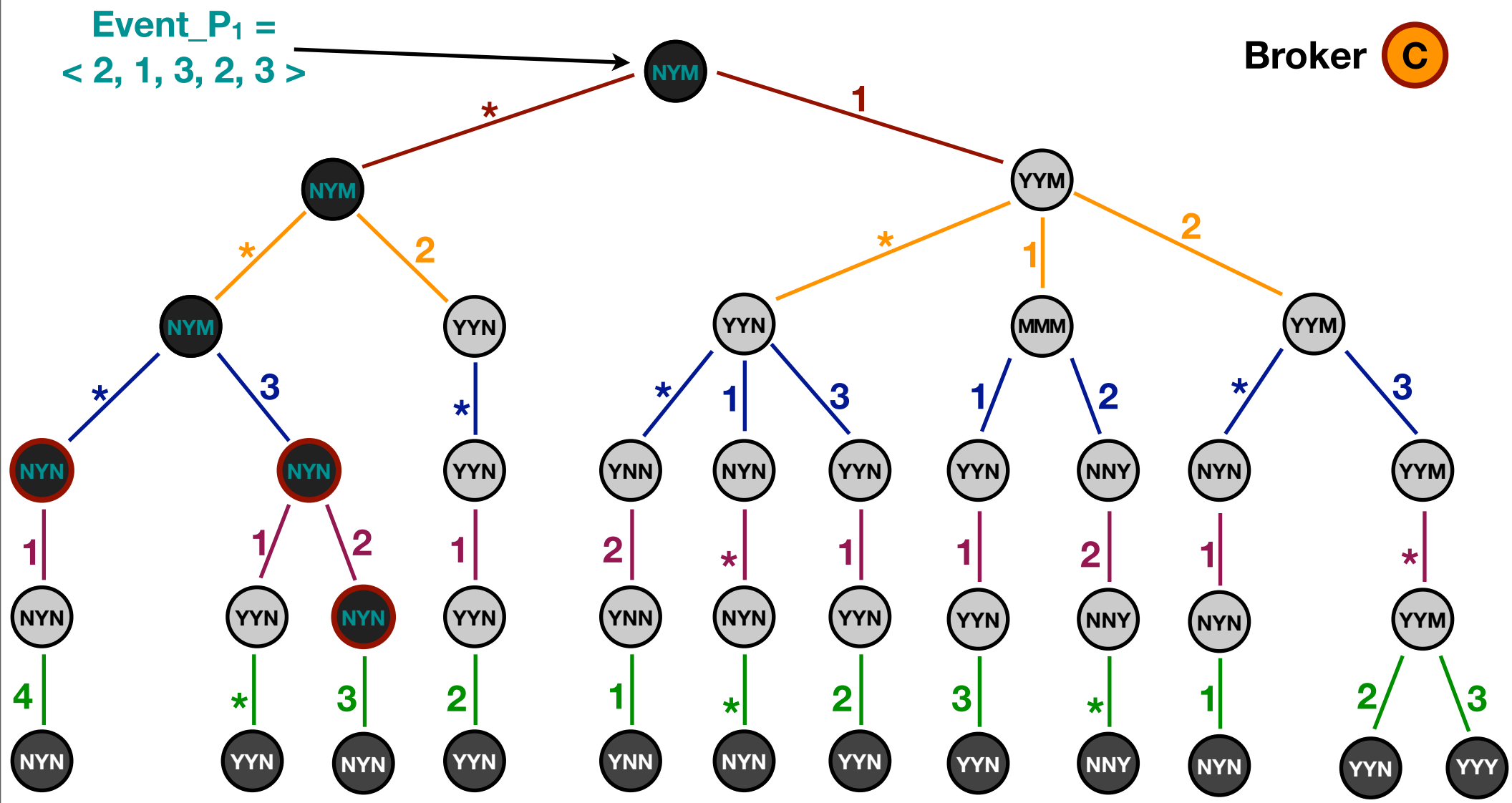
The Link-Matching Algorithm

Matching Events (3)



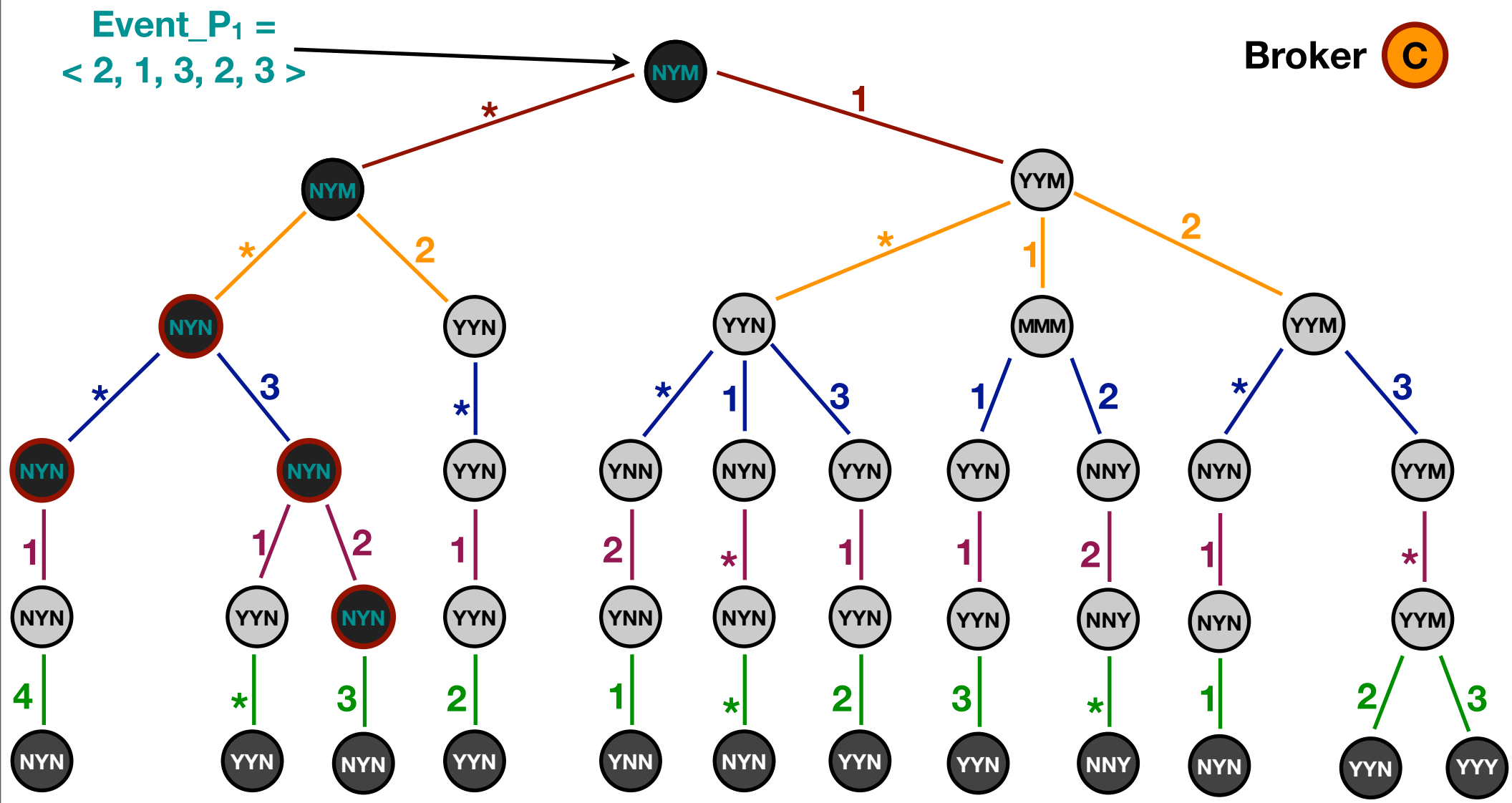
The Link-Matching Algorithm

Matching Events (3)



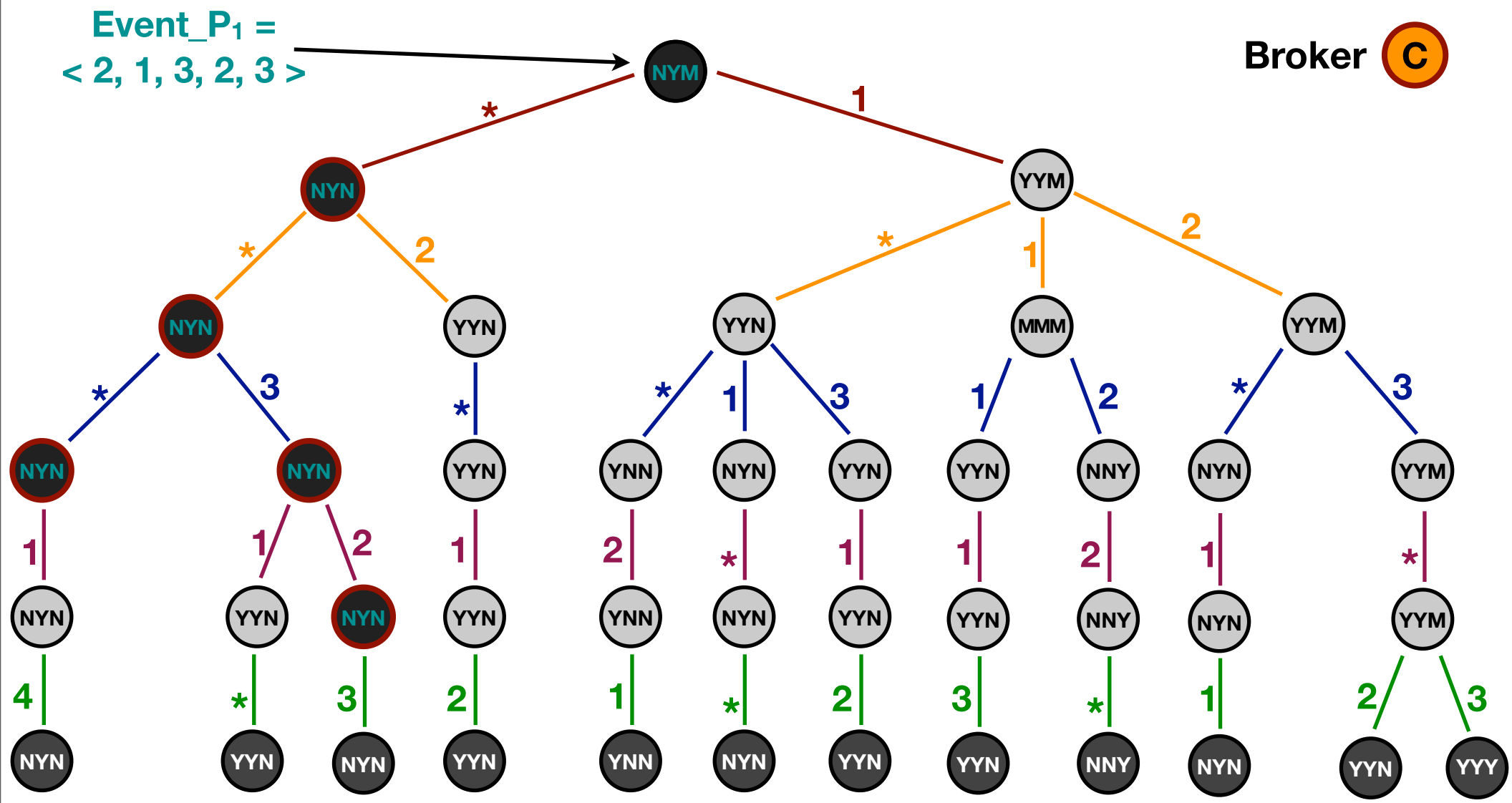
The Link-Matching Algorithm

Matching Events (3)



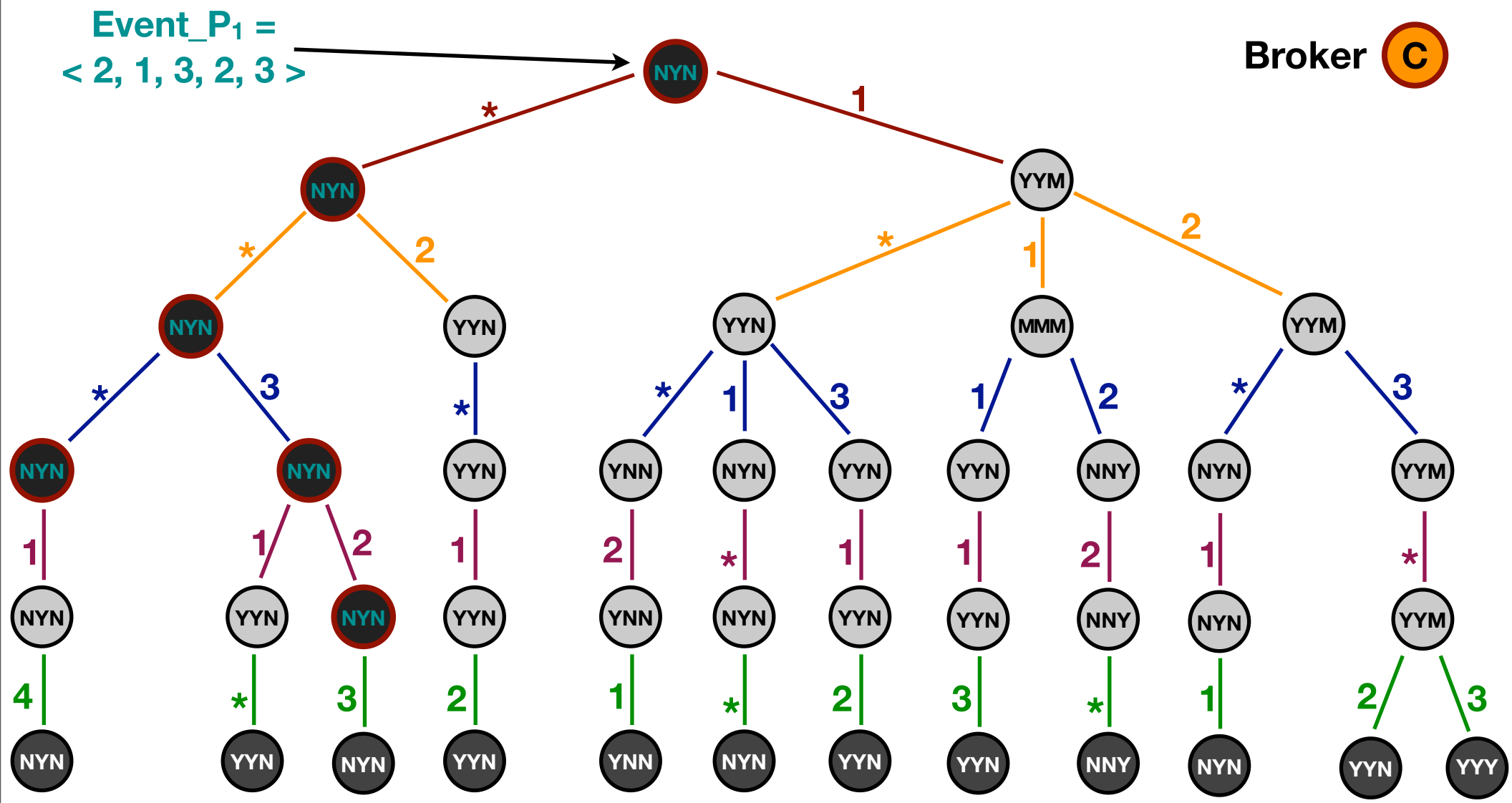
The Link-Matching Algorithm

Matching Events (3)



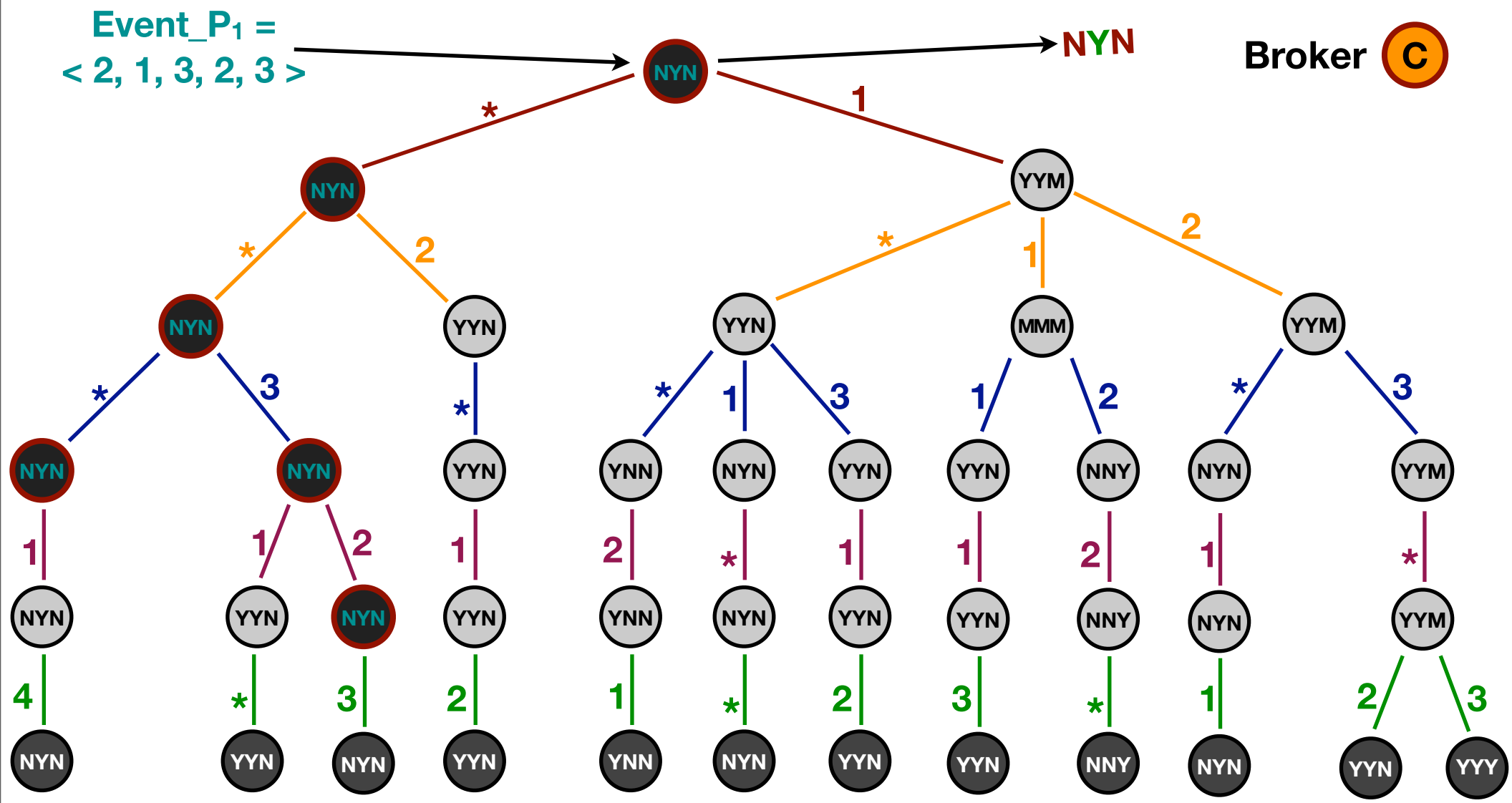
The Link-Matching Algorithm

Matching Events (3)



The Link-Matching Algorithm

Matching Events (3)

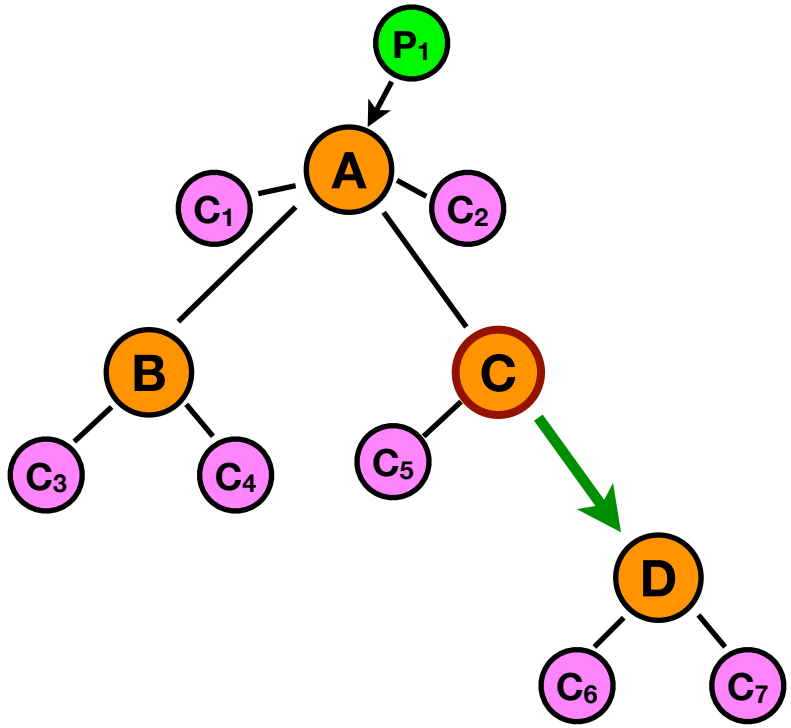


The Link-Matching Algorithm

Matching Events (4)

Event_P_1 =
< 2, 1, 3, 2, 3 >

Broker C



Links$_C$ = { L_1, L_2, L_3 }

mask = NYN

Dests = { C_5 }

Dests = { D, C_6, C_7 }

Dests = { B, C_3, C_4, A, C_1, C_2 }

Implementation and Performance

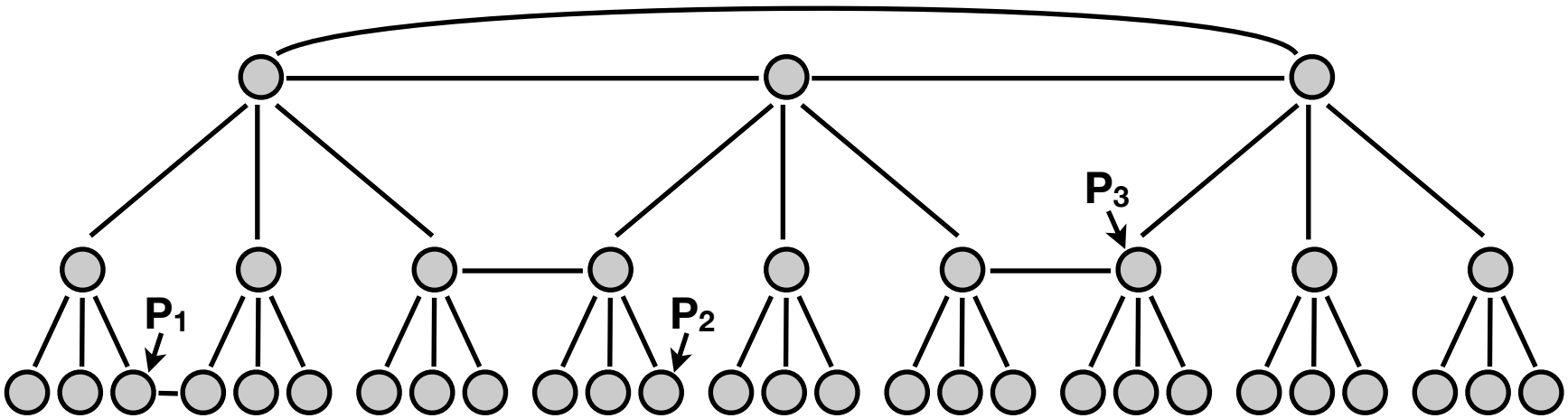
- The link-matching algorithm was implemented and tested on a simulated network topology as well as on a real LAN

- Simulation goals:
    - To measure the network loading characteristics of the link matching protocol and compare it to that of the flooding protocol
    - To measure the processing time taken by the link matching algorithm at individual broker nodes and compare it to that of centralised matching (non-trit)

Implementation and Performance

Simulated Network

- The simulated broker network is composed by:
  - 39 brokers and 10 subscribing clients per broker
  - Each client with potentially multiple subscriptions
  - The 39 brokers form three regional sub-trees of 13 brokers each
  - The roots of the sub-trees are interconnected
  - Top-level brokers have one-hop delays of 50ms, 65ms and 75ms
  - Next-level-hop delays are 25ms, 10ms and 1ms for 1st, 2nd and 3rd levels
  - Lateral links have delays of 50ms



Implementation and Performance
Simulation Characteristics

- Subscriptions are generated randomly using a given probability according to a Zipf distribution

- Events are also generated randomly at the publishers according to a Poisson distribution with values that follow a Zipf distribution
  - Values preferred by subscribers in a region are also the values most frequently published by publishers in that region

- The simulation models:
  - The passage of virtual time due to link traversal (hop delay)
  - Queue delay, CPU consumption and software latency at each broker

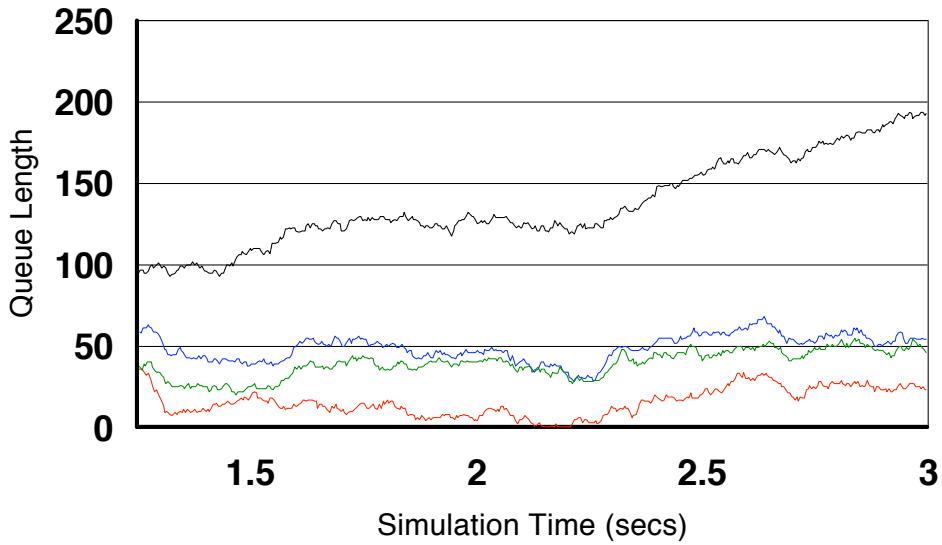
Implementation and Performance

Network Loading Results

- The purpose of this simulation was to determine, for both the link matching and flooding protocol, the event publish rate at which the network becomes overloaded
- A broker is overloaded when its input queue is growing at a rate greater than the rate at which the broker can dequeue events, leading to messages being dropped

- 450 curve: increasing monotonically
- 408 curve: occasionally draining
- 417 curve: eventually overloads
- 425 curve: eventually drains

- Estimated overload threshold queue length was 80 ± 12

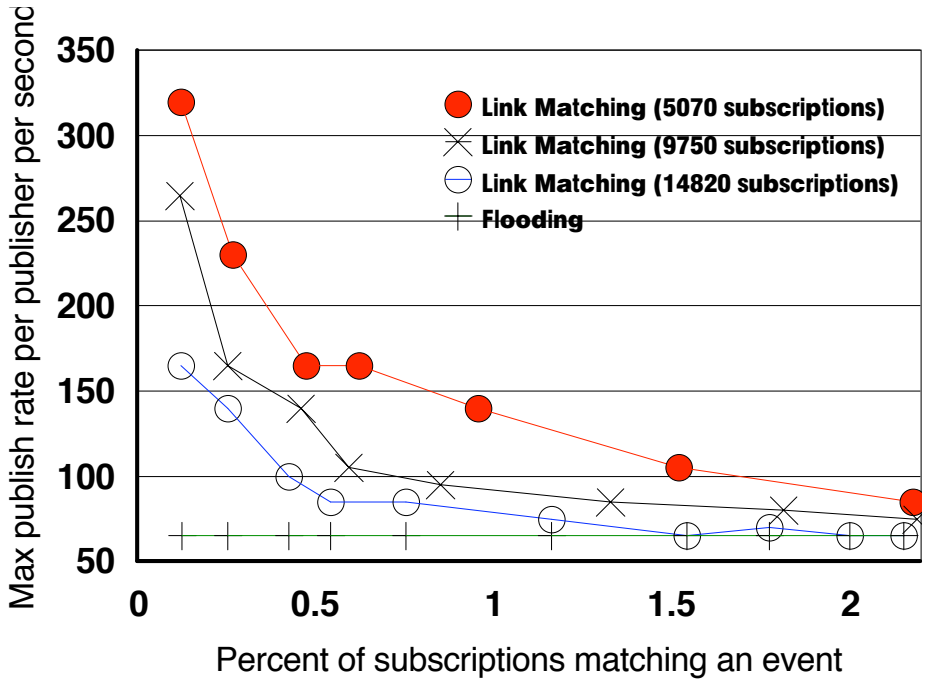


Bottom curve = 408 evts/sec; Top curve = 450 evts/sec
Middle curves = 417 & 425 evts/sec

Implementation and Performance
Network Loading Results (2)

- Event schema with 15 attributes, 3 values per attribute
- The broker network is considered overloaded when any one broker overloads
- The confidence interval for these runs is ± 5 events/sec

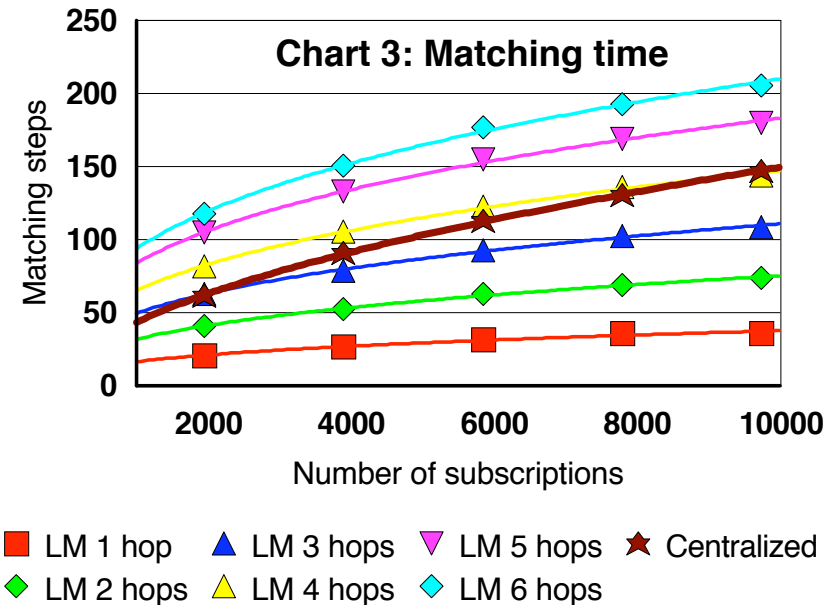
- The flooding protocol overloads at the same publish rate regardless of the number of subscriptions
- The link matching protocol is able to handle much higher publish rates without overloading when each event is destined to a small percentage of subscriptions (i.e. when subscriptions are highly selective)
- The difference is not as great when events are distributed quite widely



Implementation and Performance
Matching Time Results

- The purpose of this simulation is to measure the cumulative processing time taken by the link matching algorithm and the centralised matching algorithm
- The processing time taken per-event in the link matching algorithm is the sum of the times for all the partial matches at intermediary brokers along the way from the publisher to the subscriber
- The event schema has 10 attributes, each with 3 values
- A matching step is a visitation of a single node in the matching tree

- For 10.000 subscriptions, the cumulative matching steps for up to 4 hops using the link matching algorithm is not more than the number of matching steps taken by the centralised algorithm
- For more than 4 hops, the link matching protocol takes more matching steps than the centralised one



Implementation and Performance
Matching Time Results (2)

- The link matching protocol is a better choice over the centralised algorithm, event for more than four hops because:

  1) The extra processing time for link matching (of the order much less than 1ms) is insignificant compared to the network latency

  2) The improvement in latency from publishers to regional subscribers obtained by decentralising brokers is significant

  3) For really large numbers of subscribers (i.e. much beyond 10.000), the slopes in the lines in the chart indicate that centralised matching may take more steps than link matching

References

- G. Banavar et al., "**An efficient Multicast Protocol for Content-Based Publish-Subscribe Systems**", *in Proceedings of the 19th IEEE International Conference on Distributed Systems, 1999*

- M. Aguilera et al., "**Matching Events in a Content-Based Subscription System**", *in Proceedings of the 18th ACM Symposium on the Principles of Distributed Computing, May 1999*